

ЗАДАЧА №1468

Ближайшее нечётное

(Время: 1 сек. Память: 16 Мб Сложность: 8%)

Дано целое четное число X . Требуется вывести ближайшее для него нечётное число. Если вариантов несколько, то вывести то, которое короче в десятичной записи. Если вариантов все-равно несколько, вывести то, которое меньше.

Входные данные

Входной файл INPUT.TXT содержит целое четное число X ($|X| \leq 10^6$).

Выходные данные

В выходной файл OUTPUT.TXT выведите ближайшее нечётное число, удовлетворяющее условию задачи.

Примеры

Примечание

Минус также считается символом десятичной записи.

№	INPUT.TXT	OUTPUT.TXT
1	10	9
2	4	3
3	-6	-7

```
#include <iostream>

int main() {
    int x; std::cin >> x;
    auto a = std::to_string(x-1);
    auto b = std::to_string(x+1);
    std::cout << (a.size() > b.size() ? b : a);
    return 0;
}
```

ЗАДАЧА №1243

Стоимость

(Время: 1 сек. Память: 16 Мб Сложность: 10%)

В столовой ИКИТ СФУ пирожок стоит А рублей В копеек. Сколько рублей и копеек стоит N пирожков?

Входные данные

Входной файл INPUT.TXT содержит три целых числа A, B, N, разделенных пробелом ($0 \leq A, B, N < 100$).

Выходные данные

В выходной файл OUTPUT.TXT выведите через пробел два числа: количество рублей и копеек, которые нужно заплатить за N пирожков.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	1 25 2	2 50

```
#include <stdio.h>

int main() {
    int a, b, n;
    scanf("%d %d %d", &a, &b, &n);
    b += 100 * a;
    b *= n;
    a = b / 100;
    b %= 100;
    printf("%d %d", a, b);
    return 0;
}
```

ЗАДАЧА №1241

Квадрат

(Время: 1 сек. Память: 16 Мб Сложность: 13%)

Целое число N называется квадратом, если найдется такое целое число M, что $M^2 = N$.

Определите, является ли заданное целое число квадратом.

Входные данные

Входной файл INPUT.TXT содержит целое число N ($0 \leq N \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите «YES», если заданное число является квадратом, и «NO» в противном случае.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	25	YES
2	37	NO

```
#include <stdio.h>
#include <cmath>
#include <cassert>

bool solve(int x) {
    int root = std::sqrt(x);
    return root * root == x;
}

int main() {
    int x; scanf("%d", &x);
    printf(solve(x) ? "YES" : "NO");
    return 0;
}
```

ЗАДАЧА №1404

Простой шифр

(Время: 1 сек. Память: 16 Мб Сложность: 13%)

Каким может быть самый простой способ зашифровать строку? Есть способы, для которых не требуется применять много усилий и обладать специальными познаниями. Одним из таких способов является простой сдвиг букв. Под сдвигом понимается замена буквы на предыдущую в алфавите. Если предыдущей буквы нет, она заменяется на последнюю букву алфавита (в этой задаче мы имеем дело с английским алфавитом).

Вам прислали секретное сообщение, зашифрованное способом, описанным выше и состоящее из строчных английских букв. Расшифруйте его и выведите.

Входные данные

Входной файл INPUT.TXT содержит строку S длиной от 1 до 100 символов, состоящую из строчных английских букв.

Выходные данные

В выходной файл OUTPUT.TXT выведите расшифрованную строку.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	bnqqdbb	correct

```
#include <iostream>
#include <string>

int main() {
    std::string s;
    std::cin >> s;
    for (auto& it : s) {
        it = (it - 'a' + 1) % 26 + 'a';
    }
    std::cout << s;
    return 0;
}
```

ЗАДАЧА №1248

Расписание

(Время: 1 сек. Память: 16 Мб Сложность: 16%)

Каждый день в университете проходит следующим образом. В 00:00 наступает утро, которое длится до начала первой пары. Затем проходят 6 пар, разделенных переменами. Наконец, наступает вечер, который продолжается до 23:59 включительно. Зная текущее время и расписание звонков, определите, идет ли сейчас пара, и если идет, то какая. Пара включает в себя минуту своего начала, но не включает минуту своего окончания. Расписание звонков приведено в таблице:

Входные данные

Входной файл INPUT.TXT содержит текущее время в формате hh:mm (с ведущими нулями). Гарантируется, что hh от 00 до 23, mm от 00 до 59.

Выходные данные

В выходной файл OUTPUT.TXT выведите:

- «MORNING», если сейчас утро;
- «EVENING», если сейчас вечер;
- «BREAK», если сейчас перемена;
- целое число (номер пары), если сейчас пара.

Примеры

Номер пары	Время начала	Время окончания
1	8:30	9:55
2	10:10	11:35
3	11:50	13:15
4	13:45	15:10
5	15:25	16:50
6	17:05	18:30

№	INPUT.TXT	OUTPUT.TXT
1	11:20	2
2	07:00	MORNING

```
#include <stdio.h>
```

```
int main() {
    int h, m;
    scanf("%d:%d", &h, &m);
    m += 60 * h;
    if (m < 8 * 60 + 30) {
        printf("MORNING");
    } else if (m < 9 * 60 + 55) {
        printf("1");
    } else if (m < 10 * 60 + 10) {
        printf("BREAK");
    }
```

```
} else if (m < 11*60+35) {  
    printf("2");  
}  
else if (m < 11*60+50) {  
    printf("BREAK");  
}  
else if (m < 13 * 60 + 15) {  
    printf("3");  
}  
else if (m < 13*60+45) {  
    printf("BREAK");  
}  
else if (m < 15*60+10) {  
    printf("4");  
}  
else if (m < 15*60 + 25) {  
    printf("BREAK");  
}  
else if (m < 16*60+50) {  
    printf("5");  
}  
else if (m < 17*60+5) {  
    printf("BREAK");  
}  
else if (m < 18*60 + 30) {  
    printf("6");  
}  
else {  
    printf("EVENING");  
}  
return 0;  
}
```

ЗАДАЧА №1415

Подготовка к ЕГЭ

(Время: 1 сек. Память: 16 Мб Сложность: 17%)

Николай Петрович хочет, чтобы его ученики как можно лучше сдали ЕГЭ по информатике. Поэтому он специально усложняет некоторые задания, требуя, чтобы ребята предлагали решения для произвольных входных данных. Сейчас Николай Петрович изменил условие очередной задачи, которое теперь выглядит следующим образом.

Есть робот, который последовательно выполняет **k** операций. Каждая операция заключается или в прибавлении к **x** заданного целого числа **a** или вычитании из **x** заданного целого числа **b**. Изначально **x** равен 0. Требуется определить, сколько различных чисел может получить робот после **k** операций.

Так как Николай Петрович сомневается, не перестарался ли он в этот раз со сложностью, то решил вначале проверить, сколько людей смогут решить данную задачу на личном первенстве.

Входные данные

Входной файл INPUT.TXT содержит три целых числа **k**, **a** и **b** ($1 \leq k \leq 10^7$; $|a| \leq 10^7$; $|b| \leq 10^7$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число — ответ на задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 2 1	5
2	1 5 3	2

```
#include <iostream>
int main() {
    int k, a, b;
    std::cin >> k >> a >> b;
    std::cout << ((a+b) == 0 ? 1 : k+1);
    return 0;
}
```

ЗАДАЧА №1568

Юниты и ресурсы

(Время: 1 сек. Память: 16 Мб Сложность: 17%)

Вася играет в стратегическую компьютерную игру. В играх подобного жанра используются два важных понятия: юниты и ресурсы.

Ресурсы - это своеобразная валюта внутри игры. В разных играх ресурсами могут выступать золото, лес, минералы, нефть, камни, пища и другое. В одной игре может быть (и чаще всего так и есть) сразу несколько различных ресурсов.

Юнит - это боевая или рабочая живая единица в компьютерных играх, покупается за ресурсы. Каждый юнит имеет свою стоимость, выраженную в различных видах ресурсов. За некоторые юниты нужно расплачиваться не одним типом ресурсов, а несколькими. Если какого-либо из ресурсов имеется меньше, чем требуется, то покупка невозможна.

Вася накопил определенное количество ресурсов. Теперь Вася хочет на эти ресурсы приобрести какого-нибудь юнита.

В игре существует N видов ресурсов и M видов юнитов. Вам известны количество ресурсов, имеющееся у Васи и стоимость каждого из юнитов. Помогите Васе определить, кого из юнитов Вася мог бы купить, а на кого не хватит средств.

Входные данные

В первой строке входного файла INPUT.TXT содержатся два числа: N и M - количество различных ресурсов и количество различных юнитов ($1 \leq N, M \leq 100$).

В следующей строке содержится N чисел - имеющееся количество каждого из ресурсов.

Далее идет M строк, каждая из которых описывает соответствующего юнита. В каждой из этих строк содержится по N чисел - стоимости юнита в каждом из ресурсов, которые перечислены во второй строке входных данных в том же порядке.

Все числа во входном файле целые неотрицательные и не превышают 100.

Выходные данные

В выходной файл OUTPUT.TXT для каждого юнита выведите 1, если на его приобретение хватает средств и 0 в противном случае. Числа следует разделять пробелами.

Примеры

Примечание

В первом примере мы можем купить третьего и пятого юнита. Первый и четвертый юнит требуют слишком много второго ресурса, а второй требует слишком много третьего ресурса.

Во втором примере мы можем купить только последнего юнита.

№	INPUT.TXT	OUTPUT.TXT
1	3 5 10 6 9 5 9 3 4 6 10 9 3 6 2 8 7 4 5 6	0 0 1 0 1
2	3 5 10 6 5 7 9 7 1 9 2 6 9 3 9 7 5 4 4 5	0 0 0 0 1

```

#include <stdio.h>
#include <vector>

int main() {
    int nRes, nUnits;
    scanf("%d %d", &nRes, &nUnits);
    std::vector<int> res(nRes);
    for (auto& it : res) {
        scanf("%d", &it);
    }
    while (nUnits--) {
        bool flag = true;
        for (int i = 0; i < nRes; ++i) {
            int value; scanf("%d", &value);
            if (res[i] < value) {
                flag = false;
            }
        }
        printf("%c ", flag ? '1' : '0');
    }
    return 0;
}

```

ЗАДАЧА №688

Суслик и собака

(Время: 1 сек. Память: 16 Мб Сложность: 19%)

На большом поле находятся суслик и собака. Собака хочет суслика съесть, а суслик хочет оказаться в безопасности, добежав до одной из норок, выкопанных в поле. Ни собака, ни суслик в математике не сильны, но, с другой стороны, они и не беспросветно глупы. Суслик выбирает определенную норку и бежит к ней по прямой с определенной скоростью. Собака, которая очень хорошо понимает язык телодвижений, угадывает, к какой норке бежит суслик, и устремляется к ней со скоростью вдвое большей скорости суслика. Если собака добегает до норки первой (раньше суслика), то она съедает суслика; иначе суслик спасается.

Требуется написать программу, которая поможет суслику выбрать норку, в которой он может спастись, если таковая существует.

Входные данные

Во входном файле INPUT.TXT записано в первой строке два числа – координаты суслика. Во второй строке записаны два числа – координаты собаки. В третьей строке записано число n – число норок на поле. В следующих n строках записаны координаты норок. Все координаты являются целыми числами, по модулю не превышающими 10000, и записываются через пробел. Количество норок не превышает 1000.

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести число – номер норки, если у суслика есть возможность в ней спастись. Если у суслика есть возможность спрятаться в нескольких норках, то выведите ту, которая первая шла во входных данных. Если суслик не может спастись, то выведите в выходной файл «NO» (без кавычек).

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	10 10 20 20 1 15 15	NO
2	20 20 10 10 2 15 15 25 25	2

```
#include <stdio.h>

int main() {
    int x1, y1, x2, y2;
    scanf("%d %d %d %d", &x1, &y1, &x2, &y2);

    int n;
    scanf("%d", &n);
```

```

for (int i = 1; i <= n; ++i) {
    int x, y;
    scanf("%d %d", &x, &y);
    // assert(4*((x1-x)*(x1-x)+(y1-y)*(y1-y)) != (x2-x)*(x2-x)+(y2-y)*(y2-y)); RE 11
    if (4*(1LL*(x1-x)*(x1-x)+1LL*(y1-y)*(y1-y)) <= 1LL*(x2-x)*(x2-x)+1LL*(y2-y)*(y2-y))
        printf("%d\n", i);
        return 0;
    }
}
printf("NO\n");
return 0;
}

```

ЗАДАЧА №1244

Дартс

(Время: 1 сек. Память: 16 Мб Сложность: 20%)

Мишень дартса — круг радиуса R_1 с центром в начале координат. В мишень был сделан бросок, в результате которого дротик попал в точку (X, Y) (считается, что игла дротика имеет нулевой радиус).

Круг мишени разделен осями координат на четыре сектора, которые пронумерованы так же, как содержащие их четверти координатной плоскости. Попадание в каждый сектор дает определенное количество очков. Также на мишени нарисовано два кольца - большое и малое, с центрами в начале координат. Большое кольцо имеет внутренний радиус r_1 и внешний радиус R_1 ($r_1 < R_1$), второе кольцо имеет внутренний радиус r_2 и внешний радиус R_2 ($r_2 < R_2 < r_1$). Попадание в первое кольцо удваивает очки сектора, попадание во второе — утраивает. Попадание в границу кольца засчитывается за попадание в кольцо. В случае попадания в границу между секторами начисляется максимум из очков за эти сектора.

Зная количество очков, начисляемое за попадание в каждый сектор, а также координаты попадания, определите, сколько очков было набрано броском дротика в мишень (за непопадание в мишень дается 0 очков, попадание в границу мишени засчитывается).

Входные данные

Первая строка входного файла INPUT.TXT содержит четыре разделенных пробелами натуральных числа r_1, R_1, r_2, R_2 — радиусы колец ($r_2 < R_2 < r_1 < R_1 \leq 10^9$). Вторая строка содержит 4 разделенных пробелами неотрицательных целых числа: A, B, C, D — очки за попадание в первый, второй, третий и четвертый сектор соответственно ($0 \leq A, B, C, D \leq 10^9$). Третья строка содержит 2 разделенных пробелами целых числа: X, Y — координаты дротика ($|X|, |Y| \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите количество набранных очков.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 4 1 2 1 2 3 4 2 1	1

```
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <cassert>

typedef long double Real;

int main() {
    Real r1, R1, r2, R2;
    Real A, B, C, D;
    Real x, y;
```

```

std::cin >> r1 >> R1 >> r2 >> R2 >> A >> B >> C >> D >> x >> y;
int extra = 1;
Real answ = 0;
if (x*x+y*y > R1*R1) {
    printf("0");
    return 0;
} else if (x*x + y*y >= r1 * r1) {
    extra = 2;
} else if (R2 * R2 >= x*x + y*y && x*x + y*y >= r2 * r2) {
    extra = 3;
}
if (x > 0) {
    if (y > 0) {
        answ = extra * A;
    } else if (y < 0) {
        answ = extra * D;
    } else {
        assert(y == 0);
        answ = extra * std::max(A, D);
    }
} else if (x < 0) {
    if (y > 0) {
        answ = extra * B;
    } else if (y < 0) {
        answ = extra * C;
    } else {
        assert(y == 0);
        answ = extra * std::max(B, C);
    }
} else {
    assert(x == 0);
    if (y > 0) {
        answ = extra * std::max(A, B);
    } else if (y < 0) {
        answ = extra * std::max(C, D);
    } else {
        assert(y == 0);

        answ = extra * std::max({A, B, C, D});
    }
}
std::cout << std::fixed << std::setprecision(0) << answ;
return 0;
}

```

ЗАДАЧА №1406

Электронная очередь

(Время: 1 сек. Память: 16 Мб Сложность: 21%)

В отделении Битбанка используется электронная очередь. Клиенты банка, пользуясь специальным терминалом, получают талоны в зависимости от выбранной ими операции. После этого номера талонов отобразятся на информационном табло, и клиенты проходят к указанной на нём стойке. Завершив дела, клиенты оставляют талоны в специальной корзине у выхода.

Каждый талон содержит следующую информацию: уникальный идентификатор талона, время выдачи и тип операции. Уникальные идентификаторы присваиваются талонам не по порядку, при этом никакие два талона не могут иметь одинаковый идентификатор. Тип операции может быть одним из пяти вариантов: «card», «deposit», «loan», «transfer», «withdrawal». При этом клиенты по операциям «deposit» и «transfer» обслуживаются у стойки номер 1, по операциям «loan» и «withdrawal» – у стойки номер 2, а по операциям типа «card» – у стойки номер 3.

Руководство банка пригласило вас для отладки нового информационного табло. У вас в распоряжении имеется корзина с использованными талонами после рабочего дня банка, в которой все талоны беспорядочно перемешались. Требуется восстановить последовательность информационных сообщений, выводимых на табло, в порядке очереди.

Рабочий день банка начинается в 08:00 и заканчивается в 20:00. Поскольку автомат электронной очереди в отделении всего один, гарантируется, что на всех талонах указана различная метка времени.

Входные данные

В первой строке входного файла INPUT.TXT указано целое положительное число N – количество талонов в корзине ($1 \leq N \leq 43200$).

В следующих N строках содержатся описания талонов в формате «ID TIME TYPE», где ID – строка длины не более 10 символов, составленная из цифр и строчных букв английского алфавита, TIME – метка времени формата «ЧЧ:ММ:СС» в пределах рабочего дня банка: от 08:00:00 до 20:00:00, TYPE – один из возможных типов операций: «card», «deposit», «loan», «transfer», «withdrawal» (без кавычек).

Выходные данные

В выходной файл OUTPUT.TXT выведите N строк – информацию по талонам, отсортированную по возрастанию времени выдачи талона. Каждая строка должна задавать один талон и иметь следующий формат: «Ticket : counter » (без кавычек), где ID – идентификатор талона, а C – номер стойки (1, 2 или 3).

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 a 12:00:00 withdrawal aba 08:00:00 deposit abacaba 19:59:50 transfer	Ticket aba: counter 1 Ticket a: counter 2 Ticket abacaba: counter 1

```

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>
#include <string>

struct Record {
    std::string id, time; int type;

    static Record read() {
        char buf[11];
        scanf("%10s", buf);
        std::string id(buf);
        scanf("%8s", buf);
        std::string time(buf);
        scanf("%10s", buf);
        std::string type(buf);
        int t;
        if (type == "deposit" || type == "transfer") {
            t = 1;
        } else if (type == "loan" || type == "withdrawal") {
            t = 2;
        } else {
            assert(type == "card");
            t = 3;
        }
        return Record{id, time, t};
    }
};

int main() {
    int n; scanf("%d", &n);
    std::vector<Record> r(n);
    for (auto& it : r) {
        it = Record::read();
    }
    std::sort(r.begin(), r.end(), [](const Record& a, const Record& b){
        return a.time < b.time;
    });
    for (const auto& it : r) {
        printf("Ticket %s: counter %d\n", it.id.c_str(), it.type);
    }
    return 0;
}

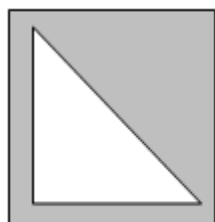
```

ЗАДАЧА №1030

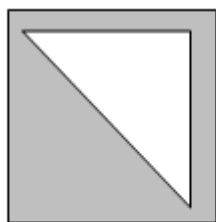
POBEDA-2014

(Время: 1 сек. Память: 16 Мб Сложность: 22%)

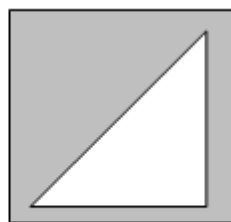
Как известно, современные видеокарты умеют формировать изображения с использованием только треугольников. Видеокарта POBEDA-2014 не отстает от современных тенденций. Известно, что она умеет отображать только прямоугольные равнобедренные треугольники четырех типов ориентации, представленные на рисунках ниже. Изменять ориентацию этих треугольников видеокарта не может.



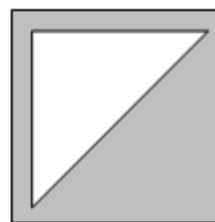
1 тип



2 тип



3 тип



4 тип

Длина катета каждого из представленных выше треугольников равна одному сантиметру. За один такт видеокарта не может отобразить более чем a_i треугольников i -того типа.

Необходимо определить максимально возможную длину стороны квадрата, который может быть изображен видеокарткой на экране монитора за один такт. При этом квадрат должен быть расположен так, чтобы его стороны были параллельны краям монитора.

Требуется написать программу, которая решает поставленную задачу.

Входные данные

Первая строка входного файла INPUT.TXT содержит разделенные пробелами четыре целых числа: a_1 , a_2 , a_3 , a_4 ($0 \leq a_1, a_2, a_3, a_4 \leq 10^{18}$).

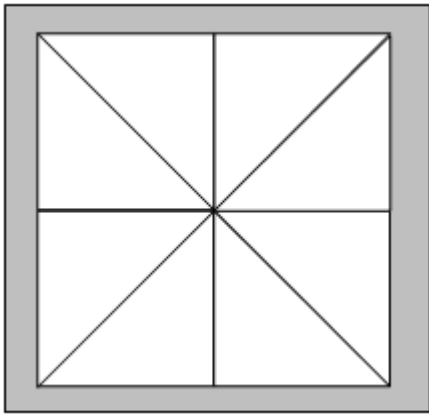
Выходные данные

В выходной файл OUTPUT.TXT выведите одно число — максимально возможную длину стороны квадрата.

Примеры

Пояснения

Иллюстрация для первого примера:



№	INPUT.TXT	OUTPUT.TXT
1	2 2 2 2	2
2	10 10 0 0	3

```
#include <iostream>
#include <cstdlib>
#include <cmath>

typedef long double Real;
typedef long long Int;

int main() {
    Real n1, n2, n3, n4;
    std::cin >> n1 >> n2 >> n3 >> n4;
    Int answ = (Int)std::floor(std::sqrt(std::min(n1, n2) + std::min(n3, n4)));
    std::cout << answ << std::endl;
    return 0;
}
```

ЗАДАЧА №1474

Крыша

(Время: 1 сек. Память: 16 Мб Сложность: 22%)

Петя на лето уехал к бабушке в деревню, где он помогает строить новый дом. Стены уже готовы, теперь нужно делать крышу. Бабушка хочет знать: сколько шифера понадобится, чтобы накрыть крышу? Петя знает ширину и длину дома, а так же необходимую высоту крыши. Но сама крыша может быть двух видов: двускатная или шатровая. Помогите Пете – для каждого типа крыши найдите площадь, которую необходимо покрыть шифером.

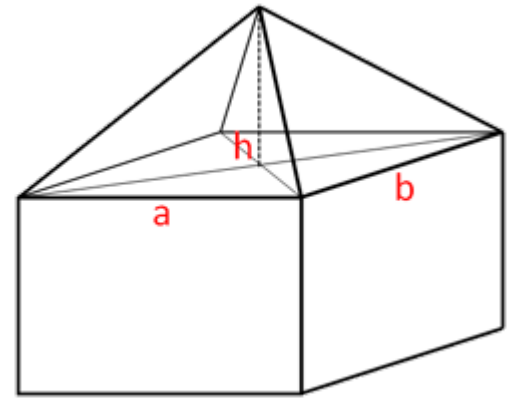
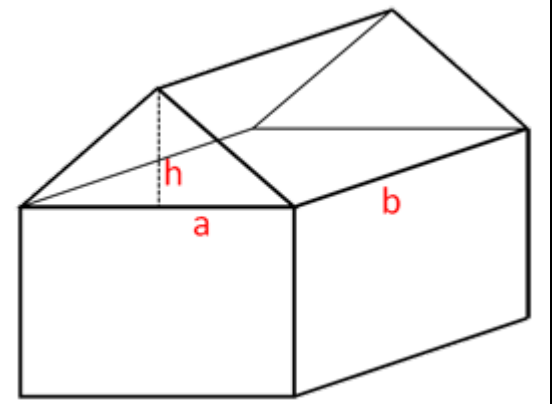
Входные данные

Входной файл INPUT.TXT содержит три целых положительных числа a , b , h – ширина, длина и высота крыши дома соответственно. Все числа не превосходят 10 000.

Выходные данные

В выходной файл OUTPUT.TXT выведите два вещественных числа – площадь двускатной и шатровой крыши с точностью не менее 6 знаков после запятой.

Пример



Двускатная крыша		Шатровая крыша	
№	INPUT.TXT	OUTPUT.TXT	
1	1 2 3	12.165525061 9.245040190	

```
#include <iostream>
#include <iomanip>
#include <cmath>

int main() {
    int a, b, h;
    std::cin >> a >> b >> h;
    std::cout << std::fixed << std::setprecision(6)
        << 2 * b * std::sqrt(h*h+a*a/4.0) << ' ';
    std::cout << std::fixed << std::setprecision(6)
        << std::sqrt(h*h+a*a/4.0) * b + std::sqrt(h*h+b*b/4.0) * a;
    return 0;
}
```

ЗАДАЧА №1477

Квадратов много не бывает

(Время: 1 сек. Память: 16 Мб Сложность: 24%)

Перед вами расположен прямоугольный лист клетчатой бумаги шириной W и высотой H клеток. Разрешается начертить на нём не более двух прямых, проходящих по линиям сетки. Каждая прямая должна проходить от края листа до края. После этого лист разрезается по начерченным прямым и, возможно, распадается на несколько новых листов.

Какое максимальное количество квадратных кусков можно получить таким образом?

Входные данные

Входной файл INPUT.TXT содержит два целых числа W и H , разделённых пробелом ($1 \leq W, H \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное целое число — ответ на задачу.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	2 3	2

```
#include <iostream>
#include <algorithm>

int solve(int w, int h) {
    // Прямые перпендикулярны - случай 4 квадратов:
    if (w == h && w % 2 == 0) {
        return 4;
    }

    // Прямые параллельны - случай 3 квадратов:
    if (h == 3LL * w || w == 3LL * h) {
        return 3;
    }

    // Прямые перпендикулярны - случай 2 квадратов:
    if ((w == h && w > 1) || (w % 2 == 0 && w / 2 <= h) || (h % 2 == 0 && h / 2 <= w)) {
        return 2;
    }

    // Прямые параллельны - случай 2 квадратов:
    if (h >= 2 * w || w >= 2 * h) {
        return 2;
    }

    // 1 квадрат можно получить всегда:
    return 1;
}

int main() {
```

```
int w, h;  
std::cin >> w >> h;  
std::cout << solve(w, h);  
return 0;  
}
```

ЗАДАЧА №1566

Виталия и кубик

(Время: 1 сек. Память: 16 Мб Сложность: 24%)

У Виталии есть белый кубик с одной красной гранью. Кубик стоит в левом верхнем углу шахматной доски красной гранью вверх.

Придумайте последовательность ходов, чтобы переместить кубик в правый верхний угол, перекатывая его по граням от клетки к клетке шахматной доски. При этом кубик не должен касаться поля доски красной гранью, а по завершении своего маршрута он должен остаться стоять красной гранью вверх. В каждой клетке поля кубик должен побывать хотя бы один раз. Количество ходов должно быть не больше 10^6 .

Входные данные

В единственной строке входного файла INPUT.TXT записаны целые числа N и M ($1 \leq N, M \leq 100$) — количество строк и столбцов шахматной доски.

Выходные данные

Если решения не существует, то в единственной строке выходного файла OUTPUT.TXT выведите «no solution» (без кавычек), в противном случае выведите искомую последовательность ходов, по одному ходу в каждой строке. Для перекатывания кубика влево, вправо, вверх и вниз используйте команды LEFT, RIGHT, UP и DOWN соответственно.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 2	DOWN RIGHT UP
2	2 3	DOWN RIGHT RIGHT UP LEFT RIGHT

```
#include <stdio.h>
#include <string>
#include <cassert>

void put(const std::string& s) {
    for (auto c : s) {
        if (c == 'u') {
            printf("UP\n");
        } else if (c == 'd') {
            printf("DOWN\n");
        } else if (c == 'r') {
            printf("RIGHT\n");
        } else if (c == 'l') {
```

```

        printf("LEFT\n");
    } else {
        throw 1;
    }
}

}

int main() {
    int nRows, nCols;
    scanf("%d %d", &nRows, &nCols);
    if (nCols == 1 && nRows == 1) {
        return 0;
    }
    if (nCols == 1 && nRows == 2) {
        put("du");
        return 0;
    }
    if (nRows == 1 || nCols == 1) {
        printf("no solution");
        return 0;
    }
    assert(nCols >= 2 && nRows >= 2);
    int row = 0, col = 0;
    while (col+1 < nCols) {
        assert(row == 0);
        while (row < nRows-1) {
            // r d l r d l
            put("rdl");
            ++row;
        }
        put("urdl");
        while (row > 0) {
            put("u");
            --row;
        }
        put("r");
        ++col;
    }
    assert(col == nCols-1 && row == 0);
    return 0;
}

```

ЗАДАЧА №501

Строение

(Время: 1 сек. Память: 16 Мб Сложность: 25%)

Администрация города подбирает площадку для строительства новых спортивных сооружений. На рассмотрении несколько проектов, каждый проект требует выделения некоторого прямоугольного участка земли. Некоторые участки оказались частично или полностью в пользовании частными лицами, а в случае утверждения проекта администрация будет вынуждена выкупить этот участок, поэтому для определения стоимости очень важно знать площадь пересечения участков. Участки частных лиц также прямоугольной формы (рис. 1) и стороны всех участков параллельны координатным осям. Для каждого проекта был построен план, включающий подобранный участок и его окружение. В приведенном примере показано пересечение участков частных лиц (тонкая линия) с участком, подобранным для строительства (толстая линия). Помогите определить суммарную площадь пересечения участков частных лиц с участком, выбранным для строительства. При этом следует учесть, что выкупаемая земля может принадлежать сразу нескольким участникам и в этом случае необходимо выкупать землю многократно (одна и та же площадь земли может быть посчитана несколько раз).

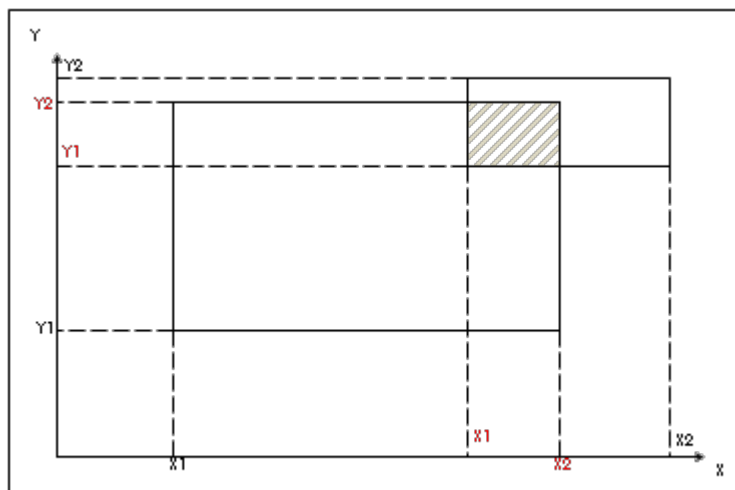


Рис. 1.

Входные данные

В первой строке входного файла INPUT.TXT содержится число N - количество участков частных, отображенных на плане. Затем следуют N строк с координатами двух вершин этих прямоугольных участков. В последней строке плана координаты участка, выбранного для строительства.

Координаты противоположных вершин одного прямоугольника описываются в формате $X_1 Y_1 X_2 Y_2$. Координатами вершин являются целые, неотрицательные числа, не больше 100. Количество исходных прямоугольников не больше 20.

Выходные данные

В единственную строку выходного файла OUTPUT.TXT следует вывести одно целое число — искомую суммарную площадь пересечения прямоугольников.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 15 15 25 25 10 10 20 20	25
2	2 15 15 25 25 5 5 12 12 10 10 20 20	29

```
#pragma GCC diagnostic ignored "-Wunused-result"

#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <string>
#include <vector>
#include <cassert>

struct Rect {
    int x_min, y_min, x_max, y_max;

    Rect(int x_min=0, int y_min=0, int x_max=0, int y_max=0)
        : x_min(x_min)
        , y_min(y_min)
        , x_max(x_max)
        , y_max(y_max)
    { }

    static Rect read() {
        int x1, y1, x2, y2;
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
        return Rect(
            std::min(x1, x2),
            std::min(y1, y2),
            std::max(x1, x2),
            std::max(y1, y2)
        );
    }

    Rect intersect(const Rect& r) const {
        return Rect(
            std::max(x_min, r.x_min),
            std::max(y_min, r.y_min),
            std::min(x_max, r.x_max),
            std::min(y_max, r.y_max)
        );
    }

    bool correct() const {
        return x_max >= x_min && y_max >= y_min;
    }

    int square() const {
        return (x_max - x_min) * (y_max - y_min);
    }
};

int main() {
    int n; scanf("%d", &n);

    std::vector<Rect> rects(n);
    for (auto& it : rects) {
        it = Rect::read();
    }

    Rect cur = Rect::read();
```



```
int s = 0;
for (const auto& it : rects) {
    auto temp = cur.intersect(it);
    s += temp.correct() ? temp.square() : 0;
}
printf("%d\n", s);
return 0;
}
```

ЗАДАЧА №1397

Космическое поселение

(Время: 1 сек. Память: 16 Мб Сложность: 25%)

Для освоения Марса требуется построить исследовательскую базу. База должна состоять из n одинаковых модулей. Каждый модуль представляет собой жилой отсек, который в основании имеет форму прямоугольника размером $a \times b$ метров.

Для повышения надежности модулей инженеры могут добавить вокруг каждого модуля дополнительный защитный слой. Толщина этого слоя должна составлять целое число метров, и все модули должны иметь одинаковую толщину защитного слоя. Модуль с защитным слоем, толщина которого равна d метрам, будет иметь в основании форму прямоугольника размером $(a + 2d) \times (b + 2d)$ метров.

Все модули должны быть расположены на заранее подготовленном прямоугольном поле размером $w \times h$ метров. При этом они должны быть организованы в виде регулярной сетки: их стороны должны быть параллельны сторонам поля, и модули должны быть ориентированы одинаково.

Требуется написать программу, которая по заданным количеству и размеру модулей, а также размеру поля для их размещения, определяет максимальную толщину дополнительного защитного слоя, который можно добавить к каждому модулю.

Входные данные

Входной файл INPUT.TXT содержит пять разделенных пробелами целых чисел: n , a , b , w и h ($1 \leq n$, a , b , w , $h \leq 10^{18}$). Гарантируется, что без дополнительного защитного слоя все модули можно разместить в поселении описанным образом.

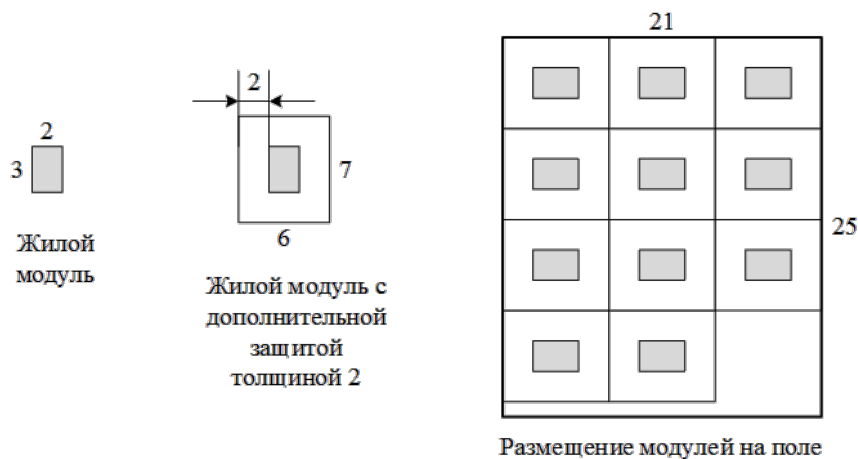
Выходные данные

Выходной файл OUTPUT.TXT должен содержать одно целое число: максимальную возможную толщину дополнительного защитного слоя. Если дополнительный защитный слой установить не удастся, требуется вывести число 0.

Примеры

Пояснение к примерам

В первом примере можно установить дополнительный защитный слой толщиной 2 метра и разместить модули на поле, как показано на рисунке.



Во втором примере жилой отсек имеет в основании размер 5×5 метров, а поле – размер 6×6 метров. Добавить дополнительный защитный слой к модулю нельзя.

№	INPUT.TXT	OUTPUT.TXT
1	11 2 3 21 25	2
2	1 5 5 6 6	0

```
#include <iostream>
#include <cstdlib>

typedef int64_t Int;

bool is_possible(const Int extra, const Int n, const Int a, const Int b, const Int A, const
    Int na = A / (a + 2 * extra);
    Int nb = B / (b + 2 * extra);
    if (nb == 0 || na == 0) return false;
    if (na >= n || nb >= n) return true;
    return na * nb >= n || na >= (n + nb - 1) / nb || nb >= (n + na - 1) / na;
}

int main() {
    Int n, a, b, A, B;
    std::cin >> n >> a >> b >> A >> B;
    Int low = 0, high = 1e18L;
    while (high - low > 1) {
        Int mid = (low + high) >> 1;
        if (is_possible(mid, n, a, b, A, B) || is_possible(mid, n, b, a, A, B)) {
            low = mid;
        } else {
            high = mid;
        }
    }
    std::cout << low << std::endl;
    return 0;
}
```

ЗАДАЧА №836

Четно-нечетная задача

(Время: 2 сек. Память: 32 Мб Сложность: 26%)

Задано n чисел a_1, a_2, \dots, a_n . Выберите из них четные числа, у которых третья справа цифра в их представлении в восьмеричной системе счисления нечетна.

Выбранные числа отсортируйте по неубыванию и выведите в выходной файл.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n ($1 \leq n \leq 10^5$). Вторая строка входного файла содержит n целых чисел: a_1, a_2, \dots, a_n . Они отделены друг от друга пробелами и заданы в десятичной системе счисления. Для всех i ($1 \leq i \leq n$) верно неравенство $64 \leq a_i \leq 10^9$.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите количество k искоемых чисел. Во второй строке выведите эти числа в указанном порядке в десятичной системе счисления.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 100 64 64 130	3 64 64 100
2	3 128 129 130	0

```
#include <stdio.h>
#include <algorithm>
#include <string>
#include <vector>

std::string convert(int number, int base) {
    std::string answer;
    do {
        answer.push_back(number % base + '0');
        number /= base;
    } while (number > 0);
    std::reverse(answer.begin(), answer.end());
    return answer;
}

int main() {
    int n;
    scanf("%d", &n);
    std::vector<int> arr;
    while (n--) {
        int value;
        scanf("%d", &value);
        auto s = convert(value, 8);
        if (s.size() < 3u) {
            s = std::string(8, '0') + s;
        }
    }
}
```

```
    }
    if ((s[s.size()-3u]-'0') % 2 == 1 && value % 2 == 0) {
        arr.push_back(value);
    }
}
std::sort(arr.begin(), arr.end());
printf("%d\n", (int)arr.size());
for (auto& it : arr) {
    printf("%d ", it);
}
return 0;
}
```

ЗАДАЧА №1081

Суммы на отрезках

(Время: 2 сек. Память: 16 Мб Сложность: 26%)

Задан числовой массив $A[1..N]$. Необходимо выполнить M операций вычисления суммы на отрезке $[L, R]$.

Входные данные

Первая строка входного файла INPUT.TXT содержит число N – размерность массива. Во второй строке записаны N чисел – элементы массива. Третья строка содержит число M – количество запросов суммы. Следующие M строк содержат пары чисел L и R ($L \leq R \leq N$), описывающие отрезки. Все числа во входных данных натуральные, не превосходящие 10^5 .

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса выведите результат суммы через пробел.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 4 4 8 7 8 2 1 2 1 3	8 16

```
#include <stdio.h>
#include <vector>
typedef long long ll;
int main() {
    int n;
    scanf("%d", &n);
    std::vector<ll> sum(1+n);
    for (int i = 1; i <= n; ++i) {
        int value;
        scanf("%d", &value);
        sum[i] = sum[i-1] + value;
    }
    int nQueries;
    scanf("%d", &nQueries);
    while (nQueries--) {
        int l, r;
        scanf("%d %d", &l, &r);
        printf("%I64d\n", sum[r] - sum[l-1]);
    }
    return 0;
}
```

ЗАДАЧА №1246

Дробь

(Время: 1 сек. Память: 16 Мб Сложность: 26%)

Вычислите результат арифметической операции (сложение, вычитание, умножение или деление) над двумя рациональными числами. Ответ выведите в виде несократимой дроби.

Рациональные числа вводятся и выводятся в следующем формате. Если число отрицательно, сначала записывается символ «-» (без кавычек). Затем записывается неотрицательное целое число – числитель дроби. Затем, если знаменатель дроби не равен 1, записывается символ «/» (без кавычек) и натуральное число – знаменатель дроби.

Входные данные

Первая строка входного файла INPUT.TXT содержит одно рациональное число – первый операнд. Во второй строке записан знак операции – символ «+», «-», «*» или «/». В третьей строке располагается рациональное число – второй операнд. Числители и знаменатели обоих операндов не превосходят 10^9 . Знаменатель не равен нулю.

Выходные данные

В выходной файл OUTPUT.TXT выведите рациональное число, являющееся результатом операции.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1/2 + 1/3	5/6
2	1 + 2	3
3	1/2 - 1/3	1/6

```
#include <stdio.h>
#include <iostream>
#include <cmath>
#include <cassert>

typedef long long ll;

ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}

int main() {
    int a, b = 1, c, d = 1;
    char buf1[30], buf2[30], op;
    scanf("%s %c %s", buf1, &op, buf2);
    sscanf(buf1, "%d/%d", &a, &b);
    sscanf(buf2, "%d/%d", &c, &d);
```

```

sscanf(buf2, "%d/%d", &c, &d);
ll p = 0, q = 1;
if (op == '*') {
    p = ll(a) * c;
    q = ll(b) * d;
} else if (op == '/') {
    p = ll(a) * d;
    q = ll(b) * c;
} else if (op == '-') {
    p = ll(a) * d - ll(b) * c;
    q = ll(b) * d;
} else {
    assert(op == '+');
    p = ll(a) * d + ll(b) * c;
    q = ll(b) * d;
}
ll gcd = ::gcd(std::abs(p), std::abs(q));
assert(p % gcd == 0);
assert(q % gcd == 0);
p /= gcd;
q /= gcd;
if (q < 0) {
    p *= -1;
    q *= -1;
}
std::cout << p;
if (p != 0 && q != 1) {
    std::cout << '/' << q;
}
return 0;
}

```


ЗАДАЧА №1567

Удаление чисел

(Время: 1 сек. Память: 16 Мб Сложность: 27%)

Василий в этом году заканчивает 9 класс и мечтает поступить в Физико-математическую школу СФУ (ФМШ). Для подготовки к вступительному экзамену он прорешивает задания прошлых лет. Одно задание показалась Василию особенно сложным, но очень интересным. Василий всё-таки смог решить это задание. А вы сможете?

В ряд выписаны все целые числа от 1 до N. За одну операцию можно удалить либо все числа на чётных позициях, либо на нечётных. Оставшиеся числа образуют новый ряд. Позиции пронумерованы с единицы, слева направо. Необходимо вывести последовательность операций, после которых в ряду останется единственное число A.

Входные данные

Входной файл INPUT.TXT содержит два целых числа: N — количество чисел и A — число, которое должно остаться после всех операций ($1 \leq A \leq N \leq 10^{18}$).

Выходные данные

В выходной файл OUTPUT.TXT выведите последовательность операций в одной строке. Для удаления всех чисел на чётных позициях выведите 0, для удаления всех чисел на нечётных позициях выведите 1.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	10 5	0 0 1

```
#include <iostream>

typedef long long Int;

int main() {
    Int n, a;
    std::cin >> n >> a;
    while (n > 1) {
        if (a % 2 == 1) {
            n = (n+1)/2;
            a = (a+1)/2;
            printf("0 ");
        } else {
            n /= 2;
            a /= 2;
            printf("1 ");
        }
    }
    return 0;
}
```

ЗАДАЧА №1245

Шифрование

(Время: 1 сек. Память: 16 Мб Сложность: 28%)

Шифр простой замены - когда каждая буква алфавита в тексте заменяется некоторой другой буквой того же алфавита (может быть, той же самой). Применяв такую замену к исходному открытому тексту, мы получим зашифрованный текст. А применив обратную замену к зашифрованному тексту, мы однозначно определим исходный открытый текст.

Вам даны две строки: **a** – исходный открытый текст и **b** - зашифрованный текст. Определите, существует ли такой шифр простой замены, применив который к **a** мы получим **b** , а применив обратную замену к **b** мы получим **a** .

Входные данные

Первая строка входного файла INPUT.TXT содержит строку с исходным открытым текстом. Вторая строка содержит строку с зашифрованным текстом. Строки непустые, содержат не более 10⁴ символов, состоят из английских букв (строчных и прописных, буквы **a** и **A** считаются различными), цифр и пробелов.

Выходные данные

В выходной файл OUTPUT.TXT выведите «YES», если искомый шифр существует, в противном случае следует вывести «NO».

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	How doth the little crocodile Hehtceowtow tdlood trierecld	YES
2	Dasha Misha	NO

```
#include <iostream>
#include <string>
#include <vector>
#include <cassert>

int main() {
    std::string a, b;
    std::getline(std::cin, a);
    std::getline(std::cin, b);
    while (a.back() == 10 || a.back() == 13) a.pop_back();
    while (b.back() == 10 || b.back() == 13) b.pop_back();
    assert(a.size() == b.size());
    std::vector<int> to(256, -1), from(256, -1);
    bool flag = true;
    for (int i = 0; flag && i < (int)a.size(); ++i) {
        const char c1 = a[i], c2 = b[i];
        if (to[c1] == -1) {
            if (from[c2] != -1) {
                flag = false;
            }
        }
    }
}
```

```
        } else {
            to[c1] = c2;
            from[c2] = c1;
        }
    } else {
        if (from[c2] == -1) {
            flag = false;
        } else {
            if (to[c1] != c2 || c1 != from[c2]) {
                flag = false;
            }
        }
    }
}
printf(flag ? "YES" : "NO");
return 0;
}
```

ЗАДАЧА №1082

Суммы в прямоугольнике

(Время: 2 сек. Память: 16 Мб Сложность: 29%)

Дана прямоугольная матрица целых чисел размером $M \times N$. Необходимо выполнить K операций вычисления суммы в прямоугольнике, принадлежащем исходной матрице, с координатами левого верхнего угла (x_1, y_1) и правого нижнего (x_2, y_2) .

Входные данные

В первой строке входного файла INPUT.TXT записаны 3 числа: N и M – число строк и столбцов матрицы ($1 \leq N, M \leq 1000$) и K - количество запросов ($1 \leq K \leq 10^5$). Каждая из следующих N строк содержит по M чисел - элементы A_{ij} соответствующей строки матрицы ($1 \leq A_{ij} \leq 10^4$). Последующие K строк содержат по 4 целых числа - y_1, x_1, y_2 и x_2 - запрос на сумму элементов в прямоугольнике ($1 \leq y_1 \leq y_2 \leq N, 1 \leq x_1 \leq x_2 \leq M$).

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса в отдельной строке выведите результат суммы.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 3 2	28 21
	1 2 3	
	4 5 6	
	7 8 9	
	2 2 3 3	
	1 1 2 3	

```
#include <stdio.h>
typedef long long ll;
int main() {
    int n, m, q;
    scanf("%d %d %d", &n, &m, &q);
    static ll sum[1+1000][1+1000];
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            int value;
            scanf("%d", &value);
            sum[i][j] = value + sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1];
        }
    }
    while (q--) {
        int i1, j1, i2, j2;
        scanf("%d %d %d %d", &i1, &j1, &i2, &j2);
        printf("%I64d\n", sum[i2][j2]-sum[i1-1][j2]-sum[i2][j1-1]+sum[i1-1][j1-1]);
    }
    return 0;
}
```

ЗАДАЧА №1410

Вася и отрезки

(Время: 1 сек. Память: 16 Мб Сложность: 29%)

Вася очень любит решать задачи на геометрию, поэтому он был очень счастлив, когда мама подарила ему на день рождения отрезок АВ. Он начал с ним играть, и вскоре у него возник вопрос, можно ли найти точки D и E, которые будут лежать на равном расстоянии от точки С – середины отрезка АВ, при этом отрезок DE должен быть перпендикулярен АВ, проходить через точку С, а $|CD|/|AB| = k$.

Помогите Васе найти ответ на его вопрос.

Входные данные

Входной файл INPUT.TXT содержит координаты двух точек: A_x, A_y, B_x, B_y, k ($0.01 \leq k \leq 1$). Все координаты являются целыми числами и не превосходят 1000 по абсолютному значению, k задано не более, чем с двумя знаками после десятичной точки.

Выходные данные

В выходной файл OUTPUT.TXT выведите координаты самой высокой точки. Если обе точки находятся на одинаковой высоте, выведите координаты самой левой из них. Координаты следует выводить с абсолютной или относительной погрешностью не хуже 10^{-4} .

Пример

№	INPUT.TXT	OUTPUT.TXT
1	0 0 2 2 0.5	0 2

```
#include <iostream>
#include <iomanip>
#include <cassert>
#include <cmath>

typedef long double Real;

int main() {
    Real x1, y1, x2, y2, k;
    std::cin >> x1 >> y1 >> x2 >> y2 >> k;
    assert(!(x1 == x2 && y1 == y2));
    // Находим середину отрезка АВ:
    Real xc = (x1+x2)/2, yc = (y1+y2) / 2;
    // Находим направляющий вектор:
    Real dx = x2-x1;
    Real dy = y2-y1;
    dx *= k;
    dy *= k;
    std::swap(dx, dy);
    dx *= -1;
    // Находим точки С и D:
    Real x3 = xc + dx;
    Real y3 = yc + dy;
    Real x4 = xc - dx;
    Real y4 = yc - dy;
    // (x3, y3) делаем точкой, удовлетворяющей условию:
```

```
if (y4 > y3 || (std::abs(y4 - y3) < 1e-4 && x4 < x3)) {  
    std::swap(x3, x4);  
    std::swap(y3, y4);  
}  
// Выводим (x3, y3):  
std::cout << std::setprecision(4) << std::fixed << x3 << ' '  
std::cout << std::setprecision(4) << std::fixed << y3 << ' '  
return 0;  
}
```

ЗАДАЧА №779

Строительство школы

(Время: 1 сек. Память: 16 Мб Сложность: 30%)

В деревне Интернетовка все дома расположены вдоль одной улицы по одну сторону от нее. По другую сторону от этой улицы пока ничего нет, но скоро все будет — школы, магазины, кинотеатры и т.д.

Для начала в этой деревне решили построить школу. Место для строительства школы решили выбрать так, чтобы суммарное расстояние, которое проезжают ученики от своих домов до школы, было минимально.

План деревни можно представить в виде прямой, в некоторых целочисленных точках которой находятся дома учеников. Школу также разрешается строить только в целочисленной точке этой прямой (в том числе разрешается строить школу в точке, где расположен один из домов — ведь школа будет расположена с другой стороны улицы).

Напишите программу, которая по известным координатам домов учеников поможет определить координаты места строительства школы.

Входные данные

В первой строке входного файла INPUT.TXT сначала записано число N — количество учеников ($1 \leq N \leq 100000$). Во второй строке записаны в строго возрастающем порядке координаты домов учеников — целые числа, не превосходящие $2 \cdot 10^9$ по модулю.

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести одно целое число — координату точки, в которой лучше всего построить школу. Если ответов несколько, выведите наибольший из них.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 1 2 3 4	3
2	3 -1 0 1	0

```
#include <stdio.h>
#include <vector>
#include <cmath>
#include <iostream>
#include <cassert>
```

```
typedef long long ll;
```

```
// Расстояние от точки p до точек x:
ll dist(ll p, const std::vector<ll>& x) {
    ll sum = 0;
```

```

    for (auto& it : x) {
        sum += std::abs(p-it);
    }
    return sum;
}

int main() {
    int n;
    scanf("%d", &n);

    std::vector<ll> x(n);
    for (auto& it : x) {
        int value;
        scanf("%d", &value);
        it = value;
    }

    // Тернарный поиск:
    ll l = -2e9, r = 2e9;
    while (r - l > 11) {
        assert(l <= r);
        ll m1 = (6 * l + 5 * r) / 11;
        ll m2 = (5 * l + 6 * r) / 11;
        assert(m1 <= m2);
        ll v1 = dist(m1, x);
        ll v2 = dist(m2, x);
        if (v1 < v2) {
            r = m2;
        } else {
            l = m1;
        }
    }
    // Находим минимум на отрезке [l, r]:
    int min = r, minv = dist(r, x);
    for (int i = r-1; i >= l; --i) {
        int curv = dist(i, x);
        if (curv < minv) {
            min = i;
            minv = curv;
        }
    }

    std::cout << min;
    return 0;
}

```


ЗАДАЧА №1571

Великая таблица умножения

(Время: 1 сек. Память: 16 Мб Сложность: 30%)

Поликарп приобрёл на барахолке кусок каменной плиты. Старик, показавший ему этот камень, утверждал, что это кусок Великой Таблицы Умножения — монумента, которому поклонялось племя коммутативных инков. Таблицу не зря называли великой — она была таблицей умножения размера 10^{18} строк на 10^{18} столбцов.

На куске камня, который попал к Поликарпу, вычерчено N строк по M целых чисел в каждой и, как утверждал старец, этот прямоугольник был вырезан из Великой Таблицы Умножения, то есть является его подпрямоугольником.

Поликарп переслал вам фотографию, на которой видно все числа и попросил проверить, может ли камень быть частью давно утерянного артефакта.

Входные данные

В первой строке входного файла INPUT.TXT содержатся два целых числа N и M — размеры таблицы ($1 \leq N, M \leq 500$).

В следующих N строках содержится по M чисел $a_{i,j}$, записанных через пробел — элементы таблицы ($1 \leq a_{i,j} \leq 10^{18}$).

Выходные данные

В выходной файл OUTPUT.TXT выведите «true», если Поликарпу достался кусок настоящего сокровища, и «false» иначе.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 2 1 2 2 4	true
2	3 3 3 4 5 4 5 6 5 6 7	false

```
/*
Решение задачи "1571. Великая таблица умножения" на астр.ru.

При помощи stdio.h:

Чтение 64-битных целых чисел как строка в стиле си:
    Время: 0.436, Память: 2252 Кб

Чтение 64-битных целых чисел как число %I64d:
    Время: 0.562, Память: 2252 Кб
```

При помощи `stdio` - TLE16 и TLE14 соответственно

Потоки ввода/вывода:

Чтение как `long long`:

Время: 0.842, Память: 2252 КБ

Чтение как строки в стиле `сi`:

Время: 0.436, Память: 2252 КБ

Буфферизованный ввод:

Время: 0.186, Память: 2252 КБ

Чисел на входе: $500 \times 500 = 250000$

*/

```
#include <iostream>
```

```
#include <cassert>
```

```
#include <vector>
```

```
#include <string>
```

```
typedef long long ll;
```

```
// Функция проверки того, что двумерный массив a является фрагментом таблицы умножения:
```

```
bool solve(const std::vector<std::vector<ll>>& a) {
    const int nRows = (int)a.size();
    const int nCols = (int)a[0].size();
    if (nRows == 1 && nCols == 1) {
        return true;
    }
    ll start_j = -1, start_i = -1;
    if (nCols > 1) {
        start_j = a[0][1] - a[0][0];
    }
    if (nRows > 1) {
        start_i = a[1][0] - a[0][0];
    }
    if (nRows == 1) {
        if (start_j <= 0 || a[0][0] % start_j != 0) {
            return false;
        }
        start_i = a[0][0] / start_j;
    }
    if (nCols == 1) {
        if (start_i <= 0 || a[0][0] % start_i != 0) {
            return false;
        }
        start_j = a[0][0] / start_i;
    }
    if (start_i <= 0 || start_j <= 0) {
        return false;
    }
    assert(start_i > 0 && start_j > 0);
    for (int i = 0; i < nRows; ++i) {
        for (int j = 0; j < nCols; ++j) {
            const ll nm = a[i][j], n = j+start_i, m = i+start_j;
            if (nm % n != 0 || nm % m != 0 || nm / n != m || nm / m != n) {
                return false;
            }
        }
    }
    return true;
}
```

```
int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0); std::cout.tie(0); std::cerr.tie(0);

    int nRows, nCols;
```

```
std::cin >> nRows >> nCols;
std::vector<std::vector<ll>> a(nRows, std::vector<ll>(nCols));
for (int i = 0; i < nRows; ++i) {
    for (int j = 0; j < nCols; ++j) {
        char buf[22];
        std::cin >> buf;
        a[i][j] = std::atoll(buf);
    }
}
std::cout << (solve(a) ? "true" : "false");
return 0;
}
```

ЗАДАЧА №946

Полка

(Время: 1 сек. Память: 16 Мб Сложность: 31%)

У Андрея есть младший брат Ванечка, который очень любит смотреть мультики. Ванечка вечно разбрасывал по дому и терял свои DVD с мультиками. Поэтому на день рождения Андрей подарил брату длинную полку для того, чтобы Ванечка ставил на нее свои диски. Чтобы на полке был порядок, Андрей просил Ванечку соблюдать простой порядок:

- если на полке нет ни одного диска, то Ванечка просто ставит его;
- если диск есть, то Ванечка ставит диск либо справа, либо слева от уже расставленных;
- забирает диски он так же, то есть снимает только с правого или левого края.

И теперь Андрей хочет узнать, выполнил Ванечка его инструкции или нет.

Входные данные

В первой строке входного файла INPUT.TXT указано целое число N ($1 \leq N \leq 10000$) - количество операций, которые выполнил Ванечка. Далее в N строках находится информация об операциях. Каждая операция постановки диска на полку описывается парой чисел. Первое из них (1 или 2) показывает, что диск ставится с левого края или с правого края соответственно. Второе целое число (от 0 до 10000) обозначает номер диска. Операции снятия диска с полки описывается одним числом 3 или 4, обозначающим с левого и правого края полки соответственно снимается диск.

В начальный момент полка пуста. Гарантируется, что последовательность операций корректна, нет команд снятия диска с пустой полки.

Выходные данные

В выходной файл OUTPUT.TXT для каждой операции снятия диска с полки выведите его номер.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	5 1 1 2 2 1 3 3 4	3 2
2	2 1 1 3	1

```
#include <stdio.h>
#include <bits/stdc++.h>

int main() {
    int nQueries;
    scanf("%d", &nQueries);
    std::deque<int> deque;
```

```
for (int q = 0; q < nQueries; ++q) {
    int type; scanf("%d", &type);
    if (type == 1) {
        int value; scanf("%d", &value);
        deque.push_front(value);
    } else if (type == 2) {
        int value; scanf("%d", &value);
        deque.push_back(value);
    } else if (type == 3) {
        printf("%d ", deque.front());
        deque.pop_front();
    } else {
        assert(type == 4);
        printf("%d ", deque.back());
        deque.pop_back();
    }
}
return 0;
}
```

ЗАДАЧА №1247

Слова

(Время: 1 сек. Память: 16 Мб Сложность: 31%)

Пусть даны два слова, состоящие из строчных английских букв. Если в этих словах различное число гласных букв, то более красивым считается слово, содержащее больше гласных (гласными буквами считаются буквы **а** , **е** , **і** , **о** , **u**). Если же число гласных в двух словах одинаково, более красивым считается то из слов, которое является лексикографически наименьшим.

Упорядочите набор слов так, чтобы для каждой пары слов в нем более красивое слово встретилось раньше, чем менее красивое.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N ($N \leq 10^4$) — количество слов. Далее следует N непустых строк – слова, состоящие не более чем из 100 строчных английских букв.

Выходные данные

В выходной файл OUTPUT.TXT выведите N строк – слова, упорядоченные по красоте.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 abcd efgh uggu ziii fooo	fooo ziii uggu abcd efgh

```
#include <stdio.h>
#include <vector>
#include <string>
#include <algorithm>

int count(const std::string& s) {
    static bool buf[256];
    buf['a'] = buf['e'] = buf['i'] = buf['o'] = buf['u'] = true;
    int answ = 0;
    for (auto c : s) {
        answ += buf[c];
    }
    return answ;
}

int main() {
    int n; scanf("%d", &n);
    std::vector<std::string> arr;
    for (int i = 0; i < n; ++i) {
        char buf[101];
        scanf("%100s", buf);
```

```
    arr.push_back(buf);
}
std::sort(arr.begin(), arr.end(), [](const std::string& a, const std::string& b) {
    int ca = count(a), cb = count(b);
    return ca > cb || (ca == cb && a < b);
});
for (const auto& s : arr) {
    printf("%s\n", s.c_str());
}
return 0;
}
```

ЗАДАЧА №1470

CSV Reader

(Время: 1 сек. Память: 16 Мб Сложность: 31%)

CSV (от англ. Comma-Separated Values – значения, разделённые запятыми) – текстовый формат, предназначенный для представления табличных данных.

Описание формата:

Каждая строка файла – это одна строка таблицы. Строки не могут быть пустыми. Каждая строка заканчивается символом перевода строки.

```
aaa,bbb,ccc  
xxx,yyy,zzz
```

Каждая строка содержит одну или более ячеек, разделённых запятыми. Разные строки могут содержать разное количество ячеек. Значение ячейки может быть пустым. Пробелы являются частью значения ячейки и не должны игнорироваться. После последней ячейки запятая не ставится.

```
aaa,a and b,bbb  
aaa,,ccc,ddd
```

Если значение содержит запятую, оно обязательно обрамляется двойными кавычками. В противном случае двойные кавычки могут отсутствовать.

```
aaa,"bbb","c , c"
```

Если обрамленное значение содержит двойные кавычки – они представляются в виде двух двойных кавычек подряд.

```
"aaa","a ""and"" b"
```

Ваша задача – написать программу, которая будет читать данные в CSV формате и выводить их на печать в отформатированном виде.

Входные данные

Входной файл INPUT.TXT имеет CSV формат и содержит не более 100 непустых строк. Каждая строка содержит не более 100 символов. Допустимые символы: строчные и прописные английские буквы, цифры, знаки препинания (точка, запятая, вопросительный и восклицательный знак, двоеточие, точка с запятой, двойные кавычки) и пробелы.

Выходные данные

В выходной файл OUTPUT.TXT выведите информацию из файла, отформатированную по следующим правилам:

- каждая строка содержит одинаковое количество ячеек. Строки могут дополняться справа необходимым количеством пустых ячеек;
- ширина каждого столбца подбирается автоматически по ширине самого длинного значения;
- остальные значения выравниваются по левому краю и дополняются справа необходимым количеством пробелов;
- разделитель между ячейками в строке – вертикальная черта «|».

Пример

№	INPUT.TXT	OUTPUT.TXT
1	a,b,c,d,e "aa",,"cc",dd "a,a,a","b,b","c" "a","a","","a and "b""",,a"b"	a b c d e aa cc dd a,a,a b,b c a,"a"," a and "b" a"b

```
#include <iostream>
#include <sstream>
#include <vector>
#include <cassert>

typedef std::vector<std::string> Row;
typedef std::vector<Row> Table;

Row parse(std::string s) {
    s = ',' + s + ',';
    Row answer;
    for (int p = 1; p < (int)s.size(); ) {
        if (s[p] == '"') {
            int j = p+1;
            while (true) {
                if (s[j] == '"') {
                    if (s[j+1] == '"') {
                        ++j;
                    } else {
                        break;
                    }
                }
                ++j;
            }
            assert(s[j] == '"');
            auto temp = s.substr(p+1, j-p-1);
            p = j+1;
            assert(s[p] == ',');
            ++p;
            std::string cur;
            for (int i = 0; i < (int)temp.size(); ++i) {
                if (temp[i] == '"') {
                    cur += temp[i];
                    if (i+1 < (int)temp.size() && temp[i+1] == '"') {
                        ++i;
                    }
                } else {
                    cur += temp[i];
                }
            }
            answer.push_back(cur);
        } else {
            int j = p;
            while (s[j] != ',') ++j;
            answer.push_back(s.substr(p, j-p));
            p = j+1;
        }
    }
    return answer;
}

std::string formate(Row& row, const std::vector<int>& width) {
    std::stringstream ss;
    while (row.size() < width.size()) row.push_back("");
    for (int i = 0; i < (int)width.size(); ++i) {
        if (i != 0) {
            ss << '|';
        }
    }
}
```

```

    }
    while ((int)row[i].size() < width[i]) row[i].push_back(' ');
    ss << row[i];
}
return ss.str();
}

int main() {
    std::string s;
    Table table;
    std::vector<int> width;
    int nCols = 0;
    while (std::getline(std::cin, s)) {
        while (s.back() == '\n' || s.back() == 13) {
            s.pop_back();
        }
        assert(s.back() != '\n');
        assert(s.back() != 13);
        table.push_back(parse(s));
        const auto& back = table.back();
        // Обновляем количество столбцов:
        nCols = std::max(nCols, (int)back.size());
        while ((int)width.size() < nCols) width.push_back(0);
        // Обновляем максимальную ширину столбца:
        for (int c = 0; c < (int)back.size(); ++c) {
            width[c] = std::max(width[c], (int)back[c].size());
        }
    }

    // Выводим ответ:
    for (auto& row : table) {
        std::cout << formate(row, width) << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №987

Обмен пакетами

(Время: 1 сек. Память: 16 Мб Сложность: 32%)

Вася получает доступ в Интернет с помощью мобильного телефона. Однако такая связь очень нестабильна, поэтому каждый раз после подключения ему приходится проверять подключение. Для этого он просто в командной строке набирает примерно следующую фразу: «ping name», где name – это имя удаленного сервера, который точно находится в сети. Затем идет обмен пакетами с сервером и выдается статистика. Однако у Васи недавно сломался модуль, отвечающий за подсчет и вывод статистики. Вам надо будет помочь Васе – написать аналогичный модуль.

После вызова команды «ping» на удаленный сервер по очереди посылаются 4 пакета по 32 байта. Как только удаленный сервер получил пакет, он отвечает на него. Если пакет не уложился в определенное время (он должен дойти до удаленного сервера и вернуться обратно) в силу тех или иных причин (низкая скорость, отсутствие подключения и т.д.), он считается утерянным.

Дана информация обо всех 4 пакетах. Требуется определить количество потерянных пакетов, максимальное, минимальное и среднее время обмена одного пакета.

Входные данные

Входной файл INPUT.TXT содержит ровно 5 строк. В первой строке находится фраза «ping name», где name – это имя сервера. Имя сервера представляет собой IP адрес. IP-адрес – это 4 однобайтных числа (т.е. числа от 0 до 255), отделенные друг от друга точкой. В каждой из следующих 4 строк содержится либо фраза «Time out», если пакет считается утерянным, либо «Reply from name Time=number», где name – это имя удаленного сервера, а number – время за которое вернулся пакет (number – целое число, $0 \leq \text{number} \leq 10^4$).

Выходные данные

В выходной файл OUTPUT.TXT выведите статистику по обмену пакетами с удаленным сервером. Следуйте формату, приведенному в примере. Среднее время округлите до целого числа по математическим правилам. Если все 4 пакета утеряны, то выведите только первые две строки.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	ping 209.85.135.147 Time out Reply from 209.85.135.147 Time=100 Reply from 209.85.135.147 Time=300 Reply from 209.85.135.147 Time=200	Ping statistics for 209.85.135.147: Packets: Sent = 4 Received = 3 Lost = 1 (25% loss) Approximate round trip times: Minimum = 100 Maximum = 300 Average = 200

/*

Задача: 987. Обмен пакетами

Решение: строки, реализация, $O(n)$

Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru

*/

```

#include <iostream>
#include <string>
#include <cassert>
#include <cmath>

int main() {
    std::string command, name;
    std::cin >> command >> name;
    assert(command == "ping");
    int sum = 0, min = 10001, max = 0, lost = 0;
    for (int i = 0; i < 4; ++i) {
        std::cin >> command;
        assert(command == "Time" || command == "Reply");
        if (command == "Time") {
            lost++;
            std::cin >> command;
            assert(command == "out");
        } else {
            std::cin >> command;
            assert(command == "from");
            std::cin >> command;
            assert(command == name);
            int value = 0;
            int code = scanf(" Time=%d", &value);
            assert(code == 1);
            min = std::min(min, value);
            max = std::max(max, value);
            sum += value;
        }
    }
    std::cout << "Ping statistics for " << name << ":\n";
    std::cout << "Packets: Sent = 4 Received = " << 4 - lost << " Lost = " << lost << " ("
    if (lost == 4) return 0;
    std::cout << "Approximate round trip times:\n";
    std::cout << "Minimum = " << min << " Maximum = " << max << " Average = " << (int)std::
    return 0;
}

```

ЗАДАЧА №1249

Изображение таблицы

(Время: 1 сек. Память: 16 Мб Сложность: 32%)

Таблица состоит из строк, каждая строка состоит из одной или нескольких ячеек, j -я ячейка i -й строки имеет ширину a_{ij} .

По заданным параметрам таблицы постройте символическое изображение ее структуры.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N — количество строк в таблице ($1 \leq N \leq 100$). Каждая из следующих N строк описывает одну строку таблицы.

Описание строки включает число m_i — количество ячеек в этой строке, и m_i целых чисел $a_{i,1}, a_{i,2}, \dots, a_{i,m_i}$ — ширину каждой из ячеек строки ($1 \leq m_i \leq 10, 1 \leq a_{ij} \leq 20$).

Выходные данные

В выходной файл OUTPUT.TXT выведите структуру таблицы.

Изображение i -й строки таблицы должно начинаться горизонтальной линией, составленной из символов «+» (плюс) и «-» (минус). Затем должна следовать строка, содержащая пробелы и символы «|» (вертикальная черта). Первым символом строки должна быть вертикальная черта, затем $a_{i,1}$ пробелов, затем вертикальная черта, затем $a_{i,2}$ пробелов, и так далее, всего m_i блоков пробелов. После последнего блока должна следовать вертикальная черта.

После последней строки таблицы также должна следовать горизонтальная линия.

В изображении горизонтальной линии используйте символ «+», если сверху или снизу от этой позиции находится вертикальная черта, и «-» в противном случае. Горизонтальная черта должна иметь минимальную возможную длину, чтобы над каждым символом вертикальной черты следующей строки и под каждым символом вертикальной черты предыдущей строки были символы «+».

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4	+-----++
	3 3 5 1	
	1 2	+-----++
	1 2	
	2 5 1	+-----++
		+-----++

```
#include <stdio.h>
#include <vector>
#include <string>
```

```

#include <algorithm>
#include <random>
#include <cstdlib>
#include <iostream>

int main() {
    int nRows;
    scanf("%d", &nRows);
    std::vector<std::string> rows(2*nRows+1);
    for (int i = 0; i < nRows; ++i) {
        const int id = 2*i+1;
        int q; scanf("%d", &q);
        while (q-->0) {
            int w; scanf("%d", &w);
            rows[id] += '|' + std::string(w, ' ');
        }
        rows[id].push_back('|');
    }
    for (int i = 0; i <= nRows; ++i) {
        const int id = 2*i;
        int size = 0;
        if (i > 0) {
            size = std::max(size, (int)rows[id-1].size());
        }
        if (i < nRows) {
            size = std::max(size, (int)rows[id+1].size());
        }
        rows[id] = std::string(size, '-');
        for (int j = 0; j < size; ++j) {
            if (i > 0 && j < (int)rows[id-1].size() && rows[id-1][j] != ' ') {
                rows[id][j] = '+';
            }
            if (i < nRows && j < (int)rows[id+1].size() && rows[id+1][j] != ' ') {
                rows[id][j] = '+';
            }
        }
    }
    for (auto& r : rows) {
        printf("%s\n", r.c_str());
    }
    return 0;
}

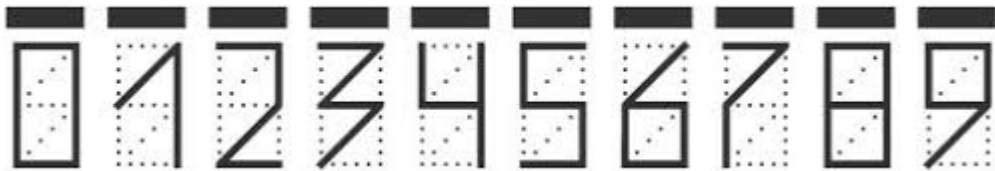
```

ЗАДАЧА №1572

Индекс.Контекст

(Время: 1 сек. Память: 16 Мб Сложность: 32%)

Даны два длинных целых числа А и В, выложенные спичками, и кучка с неограниченным количеством спичек. Числа одинаковой длины. Все цифры в числах выложены как индекс на почтовом конверте:



Найдите минимальное количество переключиваний спичек, за которое все цифры числа А можно превратить в соответствующие цифры числа В. Таким образом первая цифра числа А должна превратиться в первую цифру числа В, вторая - во вторую и т.д. За одно переключивание можно переложить спичку с одного места в числе на другое (можно переключивать спички между разными цифрами), убрать спичку из числа в кучку или взять спичку из кучки.

Входные данные

В первой строке входного файла INPUT.TXT записано число А, во второй строке - число В. Числа могут содержать ведущие нули. Каждое число содержит от 1 до 10⁵ цифр.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число - ответ на задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	010 112	7
2	12345 67890	14

```
#include <stdio.h>
#include <vector>
#include <string>
#include <cmath>

int sum(const std::string& s) {
    int answ = 0;
    for (auto c : s) {
        answ += c - '0';
    }
    return answ;
}

int same(const std::string& a, const std::string& b) {
```

```

int answ = 0;
for (int i = 0; i < (int)a.size(); ++i) {
    answ += (a[i] == '1' && '1' == b[i]);
}
return answ;
}

int main() {
    std::vector<std::string> mask = {
        "110101011", "001100010", "100100101",
        "101010100", "010110010", "110010011",
        "001011011", "101001000", "110111011", "110110100"
    };

    char buf[100000+1];
    scanf("%100000s", buf);
    std::string a(buf);
    scanf("%100000s", buf);
    std::string b(buf);

    int sum_a = 0;
    for (auto it : a) {
        sum_a += sum(mask[it-'0']);
    }

    int sum_b = 0;
    for (auto it : b) {
        sum_b += sum(mask[it-'0']);
    }

    int same = 0;
    for (int i = 0; i < (int)a.size(); ++i) {
        same += ::same(mask[a[i]-'0'], mask[b[i]-'0']);
    }

    printf("%d", std::min(sum_a, sum_b) - same + std::abs(sum_a-sum_b));
    return 0;
}

```


ЗАДАЧА №731

Проценты

(Время: 1 сек. Память: 16 Мб Сложность: 33%)

Списки ингредиентов на упаковках иногда сопровождаются их процентным содержанием, чаще всего округленным до целого числа процентов. Чтобы такой список выглядел правдоподобным, в сумме указанные числа должны давать 100%.

Однако и здесь есть определенные тонкости. Нетрудно убедиться, что, если округлять все дробные числа процентов по математическим правилам, то результирующая сумма может отличаться от нужной. Поэтому никто не сможет усомниться в вашей честности, если вы произведете округление так, как сочтете нужным. Осталось только найти лучший вариант.

Вам заданы количества всех ингредиентов, входящих в состав продукта. Для каждого ингредиента известно, положительно или отрицательно влияет на продажи его присутствие в составе. По этим данным необходимо рассчитать процентные доли каждого из ингредиентов от их суммарного количества и округлить их в нужную сторону. При этом не следует допускать, чтобы одновременно доля какого-либо вредного компонента была округлена вверх, а доля какого-либо хорошего - вниз.

Входные данные

В первой строке входного файла INPUT.TXT задано количество ингредиентов n ($1 \leq n \leq 30$). Следующие n строк описывают сами ингредиенты: знак «+» для положительно влияющих на продажи, и «-» для отрицательно влияющих, а затем, через пробел, количество соответствующего ингредиента - целое число от 1 до 1000.

Выходные данные

В выходной файл OUTPUT.TXT выведите n целых чисел, в сумме дающих 100, по одному на строке - процентные содержания ингредиентов.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 - 10 + 5	66 34
2	3 - 10 - 10 - 10	33 34 33

```
#include <stdio.h>
#include <vector>
#include <cassert>
#include <cmath>

struct Record {
    bool positive;
```

```

double value;

double rounded;

static Record read() {
    char sign; int val;
    scanf(" %c %d", &sign, &val);
    return Record{(sign == '+'), double(val), -1.0};
}

};

int main() {
    int n;
    scanf("%d", &n);

    std::vector<Record> r;
    int sum = 0;
    for (int i = 0; i < n; ++i) {
        r.push_back(Record::read());
        sum += r[i].value;
    }
    int stat = 0;
    for (auto& it : r) {
        (it.value /= sum) *= 100;
        if (it.positive) {
            it.rounded = std::ceil(it.value);
        } else {
            it.rounded = std::floor(it.value);
        }
        stat += int(it.rounded);
    }
    //printf("stat = %d\n", stat);
    int i = 0;
    while (stat > 100 && i < n) {
        while (i < n && !r[i].positive) ++i;
        assert(i < n);
        stat -= int(r[i].rounded);
        stat += int((r[i].rounded = std::floor(r[i].value)));
        ++i;
    }
    i = 0;
    while (stat < 100 && i < n) {
        while (i < n && r[i].positive) ++i;
        assert(i < n);
        stat -= int(r[i].rounded);
        stat += int(r[i].rounded = std::ceil(r[i].value));
        ++i;
    }
    assert(stat == 100);
    for (auto& it : r) {
        printf("%d\n", int(it.rounded));
    }
    return 0;
}

```

ЗАДАЧА №983

Бег по эскалатору

(Время: 2 сек. Память: 16 Мб Сложность: 33%)

Пусть N человек бегут вниз по эскалатору, причем i -ый пробегает одну ступеньку за t_i секунд. По технике безопасности бега по эскалатору, на эскалаторе запрещены «обгоны», то есть если человек A в процессе бега догнал человека B , который бежит с более низкой скоростью, то далее, до конца эскалатора, человек A бежит со скоростью человека B . Однако ступени эскалатора таковы, что на них может помещаться несколько человек одновременно.

Ваша задача написать программу, которая поможет работникам станции рассчитать, когда закончит свой бег по эскалатору каждый бегущий человек.

Входные данные

В первой строке входного файла INPUT.TXT записано число N ($1 \leq N \leq 10^5$). В следующих N строках перечислены пары чисел t_i, w_i ($1 \leq t_i, w_i \leq 10^6$) – время пробега одной ступени и количество ступеней до конца эскалатора для i -го человека. Гарантируется, что изначально всем людям осталось бежать различное число ступеней.

Выходные данные

В i -ой строке выходного файла OUTPUT.TXT выведите время в секундах, через которое i -ый человек сойдет с эскалатора.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3	20
	2 10	33
	3 11	33
	1 12	33

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <string>

struct Record {
    int time, count, id;
};

typedef long long ll;

int main() {
    int n;
    scanf("%d", &n);

    std::vector<Record> arr;
    for (int i = 0; i < n; ++i) {
        int t, w;
        scanf("%d %d", &t, &w);
```

```
        arr.push_back(Record{t,w,i});
    }

    std::sort(arr.begin(), arr.end(), [](const Record& a, const Record& b) {
        return a.count < b.count;
    });

    std::vector<ll> answ(n);
    ll t = 0;
    for (auto& r : arr) {
        t = answ[r.id] = std::max(ll(r.time) * r.count, t);
    }

    for (auto& it : answ) {
        printf("%s\n", std::to_string(it).c_str());
    }

    return 0;
}
```

ЗАДАЧА №1211

Вырубка леса

(Время: 1 сек. Память: 16 Мб Сложность: 33%)

Фермер Николай нанял двух лесорубов: Дмитрия и Федора, чтобы вырубить лес, на месте которого должно быть кукурузное поле. В лесу растут X деревьев.

Дмитрий срубает по A деревьев в день, но каждый K -й день он отдыхает и не срубает ни одного дерева. Таким образом, Дмитрий отдыхает в K -й, $2K$ -й, $3K$ -й день, и т.д.

Федор срубает по B деревьев в день, но каждый M -й день он отдыхает и не срубает ни одного дерева. Таким образом, Федор отдыхает в M -й, $2M$ -й, $3M$ -й день, и т.д.

Лесорубы работают параллельно и, таким образом, в дни, когда никто из них не отдыхает, они срубают $A + B$ деревьев, в дни, когда отдыхает только Федор — A деревьев, а в дни, когда отдыхает только Дмитрий — B деревьев. В дни, когда оба лесоруба отдыхают, ни одно дерево не срубается.

Фермер Николай хочет понять, за сколько дней лесорубы срубят все деревья, и он сможет засеять кукурузное поле.

Требуется написать программу, которая по заданным целым числам A , K , B , M и X определяет, за сколько дней все деревья в лесу будут вырублены.

Входные данные

Входной файл INPUT.TXT содержит пять целых чисел, разделенных пробелами: A , K , B , M и X ($1 \leq A, B \leq 10^9$, $2 \leq K, M \leq 10^{18}$, $1 \leq X \leq 10^{18}$).

Выходные данные

В выходной файл OUTPUT.TXT выведите искомое количество дней.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	2 4 3 3 25	7

```
#include <iostream>
#include <algorithm>
#include <functional>
#include <cstdint>

typedef uint64_t Int;

int main() {
    Int treesPerDay1, dayOff1, treesPerDay2, dayOff2, nTrees;
    std::cin >> treesPerDay1 >> dayOff1 >> treesPerDay2 >> dayOff2 >> nTrees;

    // Сколько деревьев будет срублено одним работником?
    std::function<Int(Int, Int, Int)> nCutTrees = [](Int nDays, Int treesPerDay, Int dayOff)
        return (nDays - nDays / dayOff) * treesPerDay;
};
```

```

// Сколько потребуется дней для одного работника?
std::function<Int(Int, Int, Int)> nDaysForOne = [&nCutTrees](Int nTrees, Int treesPerDay,
    Int low = 0, high = 2 * (nTrees + treesPerDay - 1) / treesPerDay;
    while (high - low > 1) {
        Int mid = (high + low) >> 1;
        if (nCutTrees(mid, treesPerDay, dayOff) >= nTrees) {
            high = mid;
        } else {
            low = mid;
        }
    }
    return high;
};

// Сколько потребуется дней для двух работников?
Int low = 0, high = std::min(
    nDaysForOne(nTrees, treesPerDay1, dayOff1),
    nDaysForOne(nTrees, treesPerDay2, dayOff2)
);

while (high - low > 1) {
    Int mid = (low + high) >> 1;
    if (nCutTrees(mid, treesPerDay1, dayOff1) + nCutTrees(mid, treesPerDay2, dayOff2) >
        high) high = mid;
    else {
        low = mid;
    }
}

std::cout << high << std::endl;

return 0;
}

```

ЗАДАЧА №1407

Производная

(Время: 1 сек. Память: 16 Мб Сложность: 33%)

Дан многочлен. Требуется вычислить его производную и вывести с использованием алгебраических соглашений: пусть *моном* – это выражение типа cx^p , где c - целое число, называемое коэффициентом, p – целое неотрицательное число, называемое показателем степени, тогда многочлен записывается как сумма мономов в соответствии со следующими правилами:

- 1. знак умножения между коэффициентом и x не выводится;
- 2. если коэффициент равен нулю, соответствующий моном не выводится;
- 3. если коэффициент равен единице или минус единице, при записи соответствующего монома единица не выводится;
- 4. если все коэффициенты равны нулю, выводится 0;
- 5. если показатель степени равен нулю, выводится только коэффициент;
- 6. если показатель степени равен единице, то единица и знак возведения в степень не выводятся;
- 7. если знак '+' предшествует отрицательному коэффициенту или стоит в начале выражения, знак '+' не выводится;
- 8. мономы выводятся строго в порядке убывания показателей степени.

Входные данные

Входной файл INPUT.TXT содержит строку длиной не более 1000 символов, описывающую многочлен. Коэффициенты многочлена целые, по модулю не превосходящие 10^4 . Показатели степени – целые неотрицательные числа, не превосходящие 10^4 . Гарантируется, что входной многочлен записан в соответствии с пунктами 1-7 правил, указанных в условии задачи.

Выходные данные

В выходной файл OUTPUT.TXT выведите производную многочлена в одну строку с использованием алгебраических соглашений.

Примеры

Примечание

Вычисление производной многочлена сводится к вычислению суммы производных каждого его члена. Производная одного члена вычисляется как производная степенной функции по формуле $(cx^p)' = cpx^{p-1}$.

№	INPUT.TXT	OUTPUT.TXT
1	$-5x^3+4x^2+x+1$	$-15x^2+8x+1$
2	$x^2+4x-10-x$	$2x+3$
3	7	0

Решение: разбор выражений, строки, $O(|s| \cdot \log(|s|))$

Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru

*/

```
#include <iostream>
#include <map>
#include <string>
#include <cassert>

typedef long long ll;

namespace Solution {

    int p; std::string data; std::map<ll, ll> coeff;

    bool input() {
        p = 0; coeff.clear();
        std::getline(std::cin, data);
        return !(std::cin);
    }

    bool getTerm() {
        if (p == (int)data.size()) {
            return false;
        }
        assert(data[p] == '+' || data[p] == '-' || data[p] == 'x' || ('0' <= data[p] && da
        bool positive = !(data[p] == '-');
        p += (data[p] == '+' || data[p] == '-');
        assert(p < (int)data.size());
        assert(('0' <= data[p] && data[p] <= '9') || (data[p] == 'x'));
        ll c = ((data[p] == 'x') ? (1) : (0));
        while ('0' <= data[p] && data[p] <= '9') {
            (c *= 10) += (data[p] - '0'); ++p;
        }
        assert(p == (int)data.size() || data[p] == 'x' || data[p] == '+' || data[p] == '-')
        if (!positive) { c = -c; }
        if (p == (int)data.size() || data[p] == '+' || data[p] == '-') {
            coeff[0] += c;
            return true;
        }
        ++p;
        assert(p == (int)data.size() || data[p] == '^' || data[p] == '+' || data[p] == '-')
        if (p == (int)data.size() || data[p] == '+' || data[p] == '-') {
            coeff[1] += c;
            return true;
        }
        ++p;
        ll power = 0;
        assert(p < (int)data.size() && '0' <= data[p] && data[p] <= '9');
        while (p < (int)data.size() && '0' <= data[p] && data[p] <= '9') {
            (power *= 10) += (data[p] - '0'); ++p;
        }
        assert(p == (int)data.size() || data[p] == '+' || data[p] == '-');
        assert(power >= 0);
        coeff[power] += c;
        return true;
    }

    std::string solve() {
        { // normalize input string
            std::string alp = "-+^0123456789x";
            std::string temp = data;
            data.clear();
            for (auto &it : temp) {
                if (alp.find(it) != std::string::npos) {
                    if (data.empty()) {
                        data.push_back(it);
                    }
                }
            }
        }
    }
}
```



```

        continue;
    }
    if (it == '+' && data.back() == '+') {
        continue;
    }
    data.push_back(it);
}
}
for (int i = 0; i+1 < (int)data.size(); ++i) {
    assert(!(data[i] == '+' && data[i+1] == '+'));
    assert(!((data[i] == '+' || data[i] == '-') && (data[i+1] == '+' || data[i+1] == '-')));
}
while (getTerm()) {}
for (auto &it : coeff) {
    auto pair = it;
    coeff[pair.first] -= pair.second;
    coeff[pair.first-1] += pair.first * pair.second;
}
std::string answer;
for (auto it = coeff.rbegin(); it != coeff.rend(); it++) {
    if (it->second == 0 || it->first < 0) { continue; }
    ll c = it->second;
    ll p = it->first;
    answer.push_back(c > 0 ? '+' : '-');
    if (c < 0) { c = -c; }
    if (c != 1 || (c == 1 && p == 0)) {
        answer += std::to_string(c);
    }
    if (p != 0) {
        answer += "x";
    }
    if (p <= 1) {
        continue;
    }
    answer += "^" + std::to_string(p);
}
if (answer.empty()) { answer = "0"; }
if (answer[0] == '+') { answer.erase(0, 1); }
return answer;
}

}

int main() {
    while (Solution::input()) { std::cout << Solution::solve() << std::endl; }
    return 0;
}

```

ЗАДАЧА №500

Агент

(Время: 1 сек. Память: 16 Мб Сложность: 34%)

Агент Джеймс Бонд пошел на пенсию, но неугомонный характер требовал новых впечатлений. Поэтому Джеймс Бонд с удовольствием согласился провести мастер-класс в некоторых группах школы «Молодого агента». Тема одного из занятий – работа агента с напарником. В таком опасном деле, как разведка, важно иметь очень надёжного напарника, поэтому напарниками могут стать только агенты, которые максимально близки по возрасту (т.е. два агента не могут стать напарниками, если в группе существует третий агент, который старше одного и младше другого).

Задание Бонда состоит в том, чтобы агенты нашли друг другу напарников таким образом, чтобы у каждого агента был хотя бы один напарник (всего у агента может быть 2 напарника – один младше, и один старше него, но эти двое не считаются напарниками между собой). Очевидно, что группа из 4 и более агентов может поделиться несколькими способами.

После нескольких занятий Бонд узнал способности групп, обучающихся в школе «Молодого агента», и оценил риск раскрытия каждого агента в отдельности. Но специфика работы с напарником такова, что в паре риску подвергается только старший из двух агентов, поэтому группу надо распределить так, чтобы суммарный риск был минимален.

Входные данные

В первой строке входного файла INPUT.TXT находится одно целое число N – количество агентов в группе ($2 \leq N \leq 10000$). Во второй строке находятся N пар целых положительных чисел, разделенных пробелом. Первое число в паре – это возраст агента (в днях) из диапазона $[5000, 16000]$, второе – риск раскрытия агента, число в диапазоне $[1, 1000]$. Известно, что в любой группе все агенты разного возраста.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное число – минимальное значение суммарного риска раскрытия группы.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 6000 2 5500 3 5000 4	5
2	5 5005 1 5004 2 5003 3 5002 4 5001 5	7

```
/*
    Выгодно объединять только в группы по 2 и 3. Остальные мы можем расформировать на групп

    Используем ДП по количество набранных участников. Переходы: последняя группа была либо
*/

#pragma GCC diagnostic ignored "-Wunused-result"
#include <stdio.h>
#include <bits/stdc++.h>
```

```

struct Agent {
    int age, risk;

    static Agent read() {
        int a, r; scanf("%d %d", &a, &r);
        return Agent{a, r};
    }
};

int main() {
    int n; scanf("%d", &n);
    std::vector<Agent> arr(1+n);
    for (int i = 1; i <= n; ++i) {
        arr[i] = Agent::read();
    }
    std::sort(arr.begin(), arr.end(), [](const Agent& a, const Agent& b){
        assert(a.age != b.age);
        return a.age < b.age;
    });
    const int INF = (int)1e9+1;
    std::vector<int> min(1+n, INF);

    min[2] = arr[2].risk;
    if (n >= 3) {
        min[3] = arr[3].risk + arr[2].risk;
    }
    if (n >= 4) {
        min[4] = arr[2].risk + arr[4].risk;
    }
    for (int len = 5; len <= n; ++len) {
        min[len] = std::min(min[len-2]+arr[len].risk, min[len-3]+arr[len-1].risk+arr[len].r
    }
    printf("%d\n", min[n]);

    return 0;
}

```

ЗАДАЧА №668

Змей Горыныч

(Время: 1 сек. Память: 16 Мб Сложность: 34%)

В некотором царстве жил Змей Горыныч. У него было N голов и M хвостов. Иван-царевич решил уничтожить губителя человеческих душ, для чего ему его кума Баба Яга подарила волшебный меч, так как только им можно убить Змея Горыныча. Если отрубить одну голову, то на её месте вырастает новая, если отрубить хвост, то вместо него вырастет 2 хвоста. Если отрубить два хвоста, то вырастает 1 голова, и только когда отрубить 2 головы, то не вырастет ничего. Змей Горыныч гибнет только в том случае, когда ему отрубить все головы и все хвосты. Определить минимальное количество ударов мечом, нужное для уничтожения Змея Горыныча.

Входные данные

В единственной строке входного файла INPUT.TXT записаны через пробел два числа N, M ($0 \leq N, M \leq 1000$).

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести одно число – минимальное количество ударов мечом, или -1, если уничтожить Змея Горыныча невозможно.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 3	9

```
#pragma GCC diagnostic ignored "-Wunused-result"
#include <stdio.h>
#include <bits/stdc++.h>

struct State {
    int heads, tails;
};

int solve(int nHeads, int nTails) {
    // bfs from position {0, 0}
    std::vector<std::vector<int>> min(1024, std::vector<int>(1024, -1));
    min[0][0] = 0;
    std::queue<State> queue;
    queue.push(State{0, 0});

    std::function<bool(const State&)> correct = [&](const State& s){
        return 0 <= s.heads && s.heads < 1024 && 0 <= s.tails && s.tails < 1024;
    };

    while (!queue.empty()) {
        auto curr = queue.front(); queue.pop();
        // cut two heads:
        { auto prev = curr; prev.heads += 2;
            if (correct(prev) && min[prev.heads][prev.tails] == -1) {
                min[prev.heads][prev.tails] = min[curr.heads][curr.tails]+1;
                queue.push(prev);
            }
        }
```

```

    }
    // cut two tails:
    if (curr.heads >= 1) {
        auto prev = curr; prev.heads -= 1; prev.tails += 2;
        if (correct(prev) && min[prev.heads][prev.tails] == -1) {
            min[prev.heads][prev.tails] = min[curr.heads][curr.tails]+1;
            queue.push(prev);
        }
    }
    // cut one tail:
    if (curr.tails >= 2) {
        auto prev = curr; prev.tails -= 1;
        if (correct(prev) && min[prev.heads][prev.tails] == -1) {
            min[prev.heads][prev.tails] = min[curr.heads][curr.tails]+1;
            queue.push(prev);
        }
    }
}
return min[nHeads][nTails];
}

int main() {
    int nHeads, nTails;
    scanf("%d %d", &nHeads, &nTails);
    printf("%d\n", solve(nHeads, nTails));
    return 0;
}

```

ЗАДАЧА №1131

Корень кубического уравнения

(Время: 1 сек. Память: 16 Мб Сложность: 34%)

Дано кубическое уравнение $ax^3 + bx^2 + cx + d = 0$ ($a \neq 0$). Известно, что у этого уравнения ровно один вещественный корень. Требуется его найти.

Входные данные

Входной файл INPUT.TXT содержит четыре целых числа: a, b, c и d – коэффициенты кубического уравнения, каждое из которых не превосходит 1000 по абсолютной величине.

Выходные данные

В выходной файл OUTPUT.TXT выведите корень кубического уравнения с точностью не менее 3 знаков после запятой.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 -6 6 -2	1
2	-1 -6 -12 -7	-1.0000

```
#include <iostream>
#include <iomanip>
#include <cassert>

typedef long double Real;

inline Real f(Real x, int a, int b, int c, int d) {
    return d + x * (c + x * (b + x * a));
}

inline int sign(Real value) {
    return value >= 0 ? 1 : -1;
}

Real solve(int a, int b, int c, int d) {
    // Корень уравнения находится бинарным поиском
    Real high = 1e6;
    Real low = -1e6;
    int s_high = sign(f(high, a, b, c, d));
    int s_low = sign(f(low, a, b, c, d));
    assert(s_high != s_low);
    const Real eps = 1e-3;
    while (high - low > eps) {
        auto mid = (low + high) / 2;
        int s_mid = sign(f(mid, a, b, c, d));
        if (s_mid == s_high) {
            high = mid;
        } else {
            assert(s_mid == s_low);
            low = mid;
        }
    }
}
```

```
    }  
    return (low + high) / 2;  
}  
  
int main() {  
    int a, b, c, d;  
    std::cin >> a >> b >> c >> d;  
    std::cout << std::fixed << std::setprecision(3) << solve(a, b, c, d) << std::endl;  
    return 0;  
}
```

ЗАДАЧА №458

Шифровка - 2

(Время: 1 сек. Память: 16 Мб Сложность: 35%)

Для кодирования сообщения используют следующие действия: сообщение записывают, опуская пробелы, в прямоугольник заданной высоты по столбцам, а затем прочитывают строки в заданном порядке.

1 П Р И А
2 Р А Р Н
3 О М О И
4 Г М В Е

а затем, если выбрать порядок строк 3, 1, 2, 4, получают закодированное сообщение ОМОИПРИАРАРНГМВЕ.

Требуется написать программу, которая по заданным высоте прямоугольника и порядке прочтения строк при кодировке декодирует заданное сообщение.

Входные данные

Входной файл INPUT.TXT содержит: в первой строке высоту прямоугольника H ($2 \leq H \leq 10$), во второй – порядок прочтения строк (числа записаны через пробел), в третьей – закодированное сообщение, длина которого составляет от 1 до 200 символов. Закодированное сообщение состоит из заглавных и строчных русских букв в DOS-кодировке (CP 866) и цифр.

Выходные данные

В выходной файл OUTPUT.TXT записывается декодированное сообщение.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 3 1 2 4 ОМОИПРИАРАРНГМВЕ	ПРОГРАММИРОВАНИЕ
2	2 2 1 ииафзк	физика

```
/*
    Задача: 458. Шифровка - 2

    Решение: строки, реализация, O(n)

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>

bool isSpaceNeeded(int rowIndex, int height, int length);
std::string insertSpaces(const int height, const std::vector<int>& order, const std::string
```



```

int main() {
    int height; std::cin >> height;
    std::vector<int> order(height);
    for (int i = 0; i < height; ++i) {
        std::cin >> order[i];
        order[i]--;
    }
    std::string s; std::cin >> s;
    s = insertSpaces(height, order, s);
    std::vector<int> newOrder(height);
    for (int idRow = 0; idRow < height; ++idRow) {
        newOrder[order[idRow]] = idRow;
    }
    const int rowLength = (int)s.size() / height;
    for (int idCol = 0; idCol < rowLength; ++idCol) {
        for (int idRow = 0; idRow < height; ++idRow) {
            int x = newOrder[idRow] * rowLength + idCol;
            if (s[x] != ' ') {
                std::cout << s[x];
            }
        }
    }
    return 0;
}

bool isSpaceNeeded(int rowIndex, int height, int length) {
    if (length % height == 0) {
        return false;
    }
    if (rowIndex >= length % height) {
        return true;
    }
    return false;
}

std::string insertSpaces(const int height, const std::vector<int>& order, const std::string
std::string answer;
int begin = 0;
const int rowLength = ((int)s.size() + height - 1) / height;
for (int idRow : order) {
    bool need = isSpaceNeeded(idRow, height, (int)s.size());
    int newBegin = (need ? begin + rowLength - 1 : begin + rowLength);
    answer += s.substr(begin, newBegin - begin);
    if (need) {
        answer += " ";
    }
    begin = newBegin;
}
return answer;
}

```

ЗАДАЧА №883

Самое экстравагантное дупло

(Время: 1 сек. Память: 16 Мб Сложность: 35%)

В Звнящем Лесу проводится конкурс на самое экстравагантное дупло. Правда, в целях экономии нервов зрителей, конкурс проводится среди макетов, а не самих дупел.

В качестве исходного материала каждому из участников предоставляется доска размером 20 000 x 20 000 метров. Макет элементарного дупла представляет собой окружность, аккуратно выдолбленную на плоскости. Макет же экстравагантного дупла представляет из себя несколько макетов элементарных дупел, выдолбленных на одной доске.

Каждому участнику уже заданы радиусы всех элементарных дупел, которые он должен выдолбить. Более того, даже последовательность их выдалбливания строго зафиксирована. Таким образом, единственное, что каждый из участников может варьировать — это координаты центра каждого элементарного дупла.

Казалось бы, все возможности для свободного творчества уже перекрыты, но это не так. Во-первых, каждому элементарному дуплу соответствует некоторое изначальное количество баллов. Во-вторых, сразу после выдалбливания очередного элементарного дупла за него начисляется итоговый балл, равный сумме изначального балла за это дупло и всех итоговых баллов за уже выдолбленные элементарные дупла, с которыми данное дупло пересекается. Итоговый балл за экстравагантное дупло начисляется как сумма итоговых баллов за все элементарные дупла, из которых оно состоит. Два дупла считаются пересекающимися, если соответствующие им на макете окружности пересекаются, либо одна лежит внутри другой. Касания окружностей на макете недостаточно, чтобы считать дупла пересекающимися.

Вам предлагается примерить на себя шкуру дятла и написать программу, которая по заданным радиусам элементарных дупел, баллов за них, а также последовательности их выдалбливания определила бы координаты центров элементарных дупел, да так, чтобы получившееся экстравагантное дупло имело бы максимальный возможный итоговый балл.

Входные данные

В первой строке входного файла INPUT.TXT задано число n — количество элементарных дупел, которые надо задействовать ($1 \leq n \leq 1\,000$). Далее следуют n строк, каждая из которых описывает соответствующее дупло и содержит два целых числа: r_i и s_i ($0 < r_i, s_i \leq 1000$), где r_i — радиус соответствующего дупла в метрах, а s_i — изначальное количество баллов, соответствующее данному элементарному дуплу. Дупла даны в порядке выдалбливания.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать ровно n строчек, в каждой по два числа — координаты центра соответствующего элементарного дупла в метрах с точностью не менее 10^{-6} метра. Кроме того, макеты дупел не должны выходить за пределы исходной доски, а расстояние между любыми двумя центрами должно быть не менее сантиметра. Точка (0, 0) соответствует центру исходной доски. Если возможны несколько вариантов расстановки дупел, дающих максимальный балл, выведите любой.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 10 10 20 20 30 30	0 1 0 2 0 3

```

/*
    "Самое экстравагантное дупло": геометрия, конструктив, O(n)

    Главный подвох: радиусы даны в метрах, а расстояние определяется в сантиметрах.

    Можно пересечь все окружности.
*/

#include <stdio.h>

int main() {
    int n;
    scanf("%d", &n);
    for (double x = 0; n > 0 && x < 1; x += 0.01) {
        for (double y = 0; n > 0 && y < 1; y += 0.01) {
            printf("%.2f %.2f", x, y);
            --n;
        }
    }
    return 0;
}

```

ЗАДАЧА №1035

Кондиционеры

(Время: 1 сек. Память: 16 Мб Сложность: 35%)

При реализации проекта «Умная школа» было решено в каждый учебный класс выбранной для этого школы установить по кондиционеру нового поколения для автоматического охлаждения и вентиляции воздуха. По проекту в каждом классе должен быть установлен только один кондиционер и мощность кондиционера должна быть достаточной для размеров класса. Чем больше класс, тем мощнее должен быть кондиционер.

Все классы школы пронумерованы последовательно от 1 до n . Известно, что для каждого класса с номером i , требуется ровно один кондиционер, мощность которого больше или равна a_i ватт.

Администрации школы предоставили список из m различных моделей кондиционеров, которые можно закупить. Для каждой модели кондиционера известна его мощность и стоимость. Требуется написать программу, которая определит, за какую минимальную суммарную стоимость кондиционеров можно оснастить все классы школы.

Входные данные

Первая строка входного файла INPUT.TXT содержит одно целое число n ($1 \leq n \leq 50\,000$) – количество классов в школе.

Вторая строка содержит n целых чисел a_i ($1 \leq a_i \leq 1000$) – минимальная мощность кондиционера в ваттах, который можно установить в классе с номером i .

Третья строка содержит одно целое число m ($1 \leq m \leq 50\,000$) – количество предложенных моделей кондиционеров.

Далее, в каждой из m строк содержится пара целых чисел b_j и c_j ($1 \leq b_j \leq 1000, 1 \leq c_j \leq 1000$) – мощность в ваттах j -й модели кондиционера и его цена в рублях соответственно.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать одно число – минимальную суммарную стоимость кондиционеров в рублях. Гарантируется, что хотя бы один корректный выбор кондиционеров существует, и во всех классах можно установить подходящий кондиционер.

Примеры

Пояснения к примерам

В первом примере нужно купить один единственно возможный кондиционер за 1000 рублей.

Во втором примере оптимально будет установить в первом и втором классах кондиционеры четвертого типа, а в третьем классе – кондиционер третьего типа. Суммарная стоимость этих кондиционеров будет составлять 13 рублей ($3 + 3 + 7$).

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	1 800 1 800 1000	1000
2	3 1 2 3 4 1 10 1 5 10 7 2 3	13

```

/*
   "Кондиционеры": динамическое программирование,  $O(n+m+\max W)$ 
*/

#include <stdio.h>
#include <vector>
#include <algorithm>

int main() {
    int n; scanf("%d", &n);
    std::vector<int> count(1+1000, 0);
    for (int i = 0; i < n; ++i) {
        int value; scanf("%d", &value);
        count[value]++;
    }

    std::vector<int> min(1+1000, (int)1e9);
    int m; scanf("%d", &m);
    while (m--) {
        int value, cost;
        scanf("%d %d", &value, &cost);
        min[value] = std::min(min[value], cost);
    }

    std::vector<int> suff(1+1000, (int)1e9);
    suff[1000] = min[1000];
    for (int i = 999; i >= 0; --i) {
        suff[i] = std::min(suff[i+1], min[i]);
    }

    int sum = 0;
    for (int i = 0; i <= 1000; ++i) {
        sum += count[i] * suff[i];
    }
    printf("%d", sum);

    return 0;
}

```

ЗАДАЧА №1180

Максимумы на отрезках

(Время: 1 сек. Память: 16 Мб Сложность: 35%)

Задан числовой массив $A[1..N]$. Необходимо выполнить M операций вычисления максимального элемента на отрезке $[L, R]$.

Входные данные

Первая строка входного файла INPUT.TXT содержит число N – размер массива ($N \leq 10^5$). Во второй строке записаны N чисел – элементы массива, целые числа от 1 до 10^5 . Третья строка содержит натуральное число M – количество запросов максимума ($M \leq 30\,000$). Следующие M строк содержат пары чисел L и R ($1 \leq L \leq R \leq N$), описывающие отрезки.

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса выведите значение максимума на отрезке через пробел.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 3 8 1 7 6 2 1 3 3 5	8 7

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <climits>

char getChar() {
    static char buffer[1024*1024];
    static int pos = 0;
    static int size = 0;
    if (pos == size) {
        size = fread(buffer, 1, 1024*1024, stdin);
        pos = 0;
    }
    if (pos == size) {
        return EOF;
    }
    return buffer[pos++];
}

void putChar(char c) {
    static char buffer[1024*1024];
    static int size = 0;
    if (size == 1024*1024 || c == -1) {
        fwrite(buffer, 1, size, stdout);
        size = 0;
    }
```

```

    }
    if (c != -1) {
        buffer[size++] = c;
    }
}

int getInt() {
    char c = '?';
    while (!(c == '-' || ('0' <= c && c <= '9'))) c = getChar();
    bool positive = true;
    if (c == '-') {
        positive = false;
        c = getChar();
    }
    int value = 0;
    while ('0' <= c && c <= '9') {
        (value *= 10) += (c - '0');
        c = getChar();
    }
    return positive ? value : -value;
}

void putInt(int number) {
    static char buf[12];
    sprintf(buf, "%d", number);
    for (char* it = buf; *it != '\0'; ++it) {
        putChar(*it);
    }
}

struct DataStruct {
    std::vector<int> arr, max, actual;

    const int GSIZE = 256;

    DataStruct(int size = 0, int item = 0) {
        arr.assign(size, item);
        max.assign((size+GSIZE-1) / GSIZE, item);
        actual.assign((size+GSIZE-1) / GSIZE, true);
    }

    void update_group(int g) {
        if (actual[g]) return;
        actual[g] = true;
        const int begin = g * GSIZE;
        const int after = std::min(begin+GSIZE, (int)arr.size());
        int value = arr[begin];
        for (int i = begin+1; i < after; ++i) {
            value = std::max(value, arr[i]);
        }
        max[g] = value;
    }

    void set(int p, int v) {
        const int g = p / GSIZE;
        arr[p] = v;
        actual[g] = false;
    }

    int get(int l, int r) {
        const int gl = l / GSIZE;
        const int gr = r / GSIZE;
        int answ = arr[l];
        if (gl == gr) {
            for (int i = l; i <= r; ++i) {
                answ = std::max(answ, arr[i]);
            }
        } else {
            for (int i = l, after = (gl+1)*GSIZE; i < after; ++i) {

```

```

        answ = std::max(answ, arr[i]);
    }
    for (int g = gl+1; g < gr; ++g) {
        if (!actual[g]) update_group(g);
        answ = std::max(answ, max[g]);
    }
    for (int i = gr * GSIZE; i <= r; ++i) {
        answ = std::max(answ, arr[i]);
    }
}
return answ;
}

};

int main() {
    int n = getInt();
    DataStruct ds(n, 0);
    for (int i = 0; i < n; ++i) {
        int value = getInt();
        ds.set(i, value);
    }
    int q = getInt();
    while (q--) {
        int l = getInt()-1, r = getInt()-1;
        putInt(ds.get(l, r));
        putChar(' ');
    }
    putChar(-1);
    return 0;
}

```


ЗАДАЧА №1475

Хардкорный массив

(Время: 1 сек. Память: 16 Мб Сложность: 35%)

Всё, что делает Генри, является хардкорным. Вот и сейчас он выписал N целых чисел, принадлежащих отрезку $[0, 10^9]$ и образующих *хардкорный массив* – массив, в котором все попарные разности различны. *Попарные разности массива* – это все числа $|a_i - a_j|$ для всех i и j таких, что $1 \leq i < j \leq N$.

Думаете, Вы такой же хардкорный, как Генри? Конечно, нет! Поэтому, Ваша задача немного проще: в выведенном Вами массиве количество различных попарных разностей должно быть хотя бы 90% от количества всех попарных разностей.

Входные данные

Входной файл INPUT.TXT содержит целое число N – требуемый размер массива ($2 \leq N \leq 10\,000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите N целых чисел a_i ($0 \leq a_i \leq 10^9$), удовлетворяющих заданным требованиям.

Примеры

Пояснение

Массив в третьем тестовом примере не является абсолютно хардкорным. Всего попарных разностей в нём 21, но две из них, $|3-9|$ и $|15-9|$, равны между собой. Тем самым в нём 20 различных разностей, что составляет примерно 95% и является приемлемым результатом.

№	INPUT.TXT	OUTPUT.TXT
1	2	404 42
2	4	322 228 1488 1337
3	7	3 14 15 92 65 35 9

```
#include <stdio.h>
#include <vector>
#include <cstdlib>
#include <algorithm>
#include <cassert>
#include <set>
```

```
typedef long long ll;
```

```
// Взятие модуля от числа num по основанию b:
int mod(ll num, int b) {
    return num >= 0 ? num % b : b - (-num) % b;
}
```

```
// Генерация массива случайных чисел из диапазона [0, 1e9]:
```

```

std::vector<int> gen(int n) {
    std::vector<int> arr(n);
    for (auto& it : arr) {
        it = mod(1LL * std::rand() * std::rand() * std::rand(), 10000000001);
        assert(0 <= it && it <= 10000000000);
    }
    return arr;
}

// Проверка выполнимости свойства задачи на каждом префиксе:
double count_diff(const std::vector<int>& arr) {
    std::set<int> diff;
    double min = 100;
    for (int r = 1; r < (int)arr.size(); ++r) {
        for (int l = r-1; l >= 0; --l) {
            diff.insert(std::abs(arr[r]-arr[l]));
        }
        min = std::min(min, 1.0 * diff.size() / ((r+1) * r / 2));
    }
    assert(min >= 0.95);
    return min;
}

int main() {
    std::srand(124070620);
    int n;
    scanf("%d", &n);
    for (auto it : gen(n)) {
        printf("%d ", it);
    }

    return 0;
}

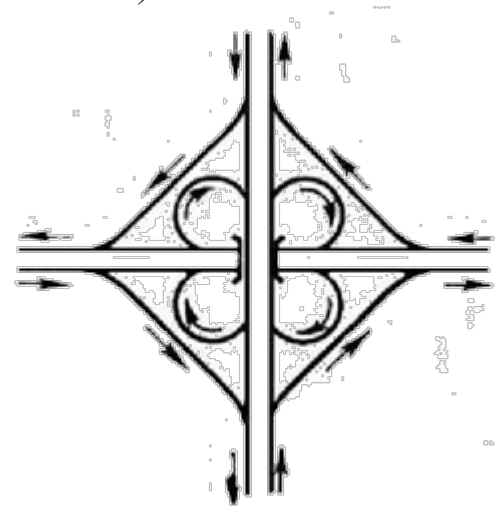
```

ЗАДАЧА №508

Клеверный лист

(Время: 1 сек. Память: 16 Мб Сложность: 36%)

В одном крупном городе строится обьездная автодорога. Она не является кольцевой, то есть имеет западное и восточное окончание. Наиболее интересными инженерными объектами на дороге, являются, конечно же, мосты. Очевидно, что железобетонный мост слишком дорогой для участка с одним автомобилем в день, а деревянный мост не способен обслуживать шоссе с десятком машин в минуту. Поэтому для оправданного проектирования мостов необходимо знать, сколько автомобилей проезжает через мост.



На дороге будут использоваться преимущественно развязки "клеверный лист", схема такой развязки изображена на рисунке. Согласно этой схеме, автомобиль, поворачивающий направо - не едет через мост, автомобиль, поворачивающий налево (по этой развязке он поворачивает на 270 градусов направо и пересекает свой путь на другом уровне) - всегда едет через мост, и автомобиль, проезжающий прямо - может ехать, а может не ехать через мост, тут всё зависит от направления движения.

Исследован поток автомобилей, которые едут по дорогам, пересекающим обьездную дорогу. Необходимо посчитать поток автомобилей, которые поедут через мосты обьездной дороги. Следует помнить, что для некоторых перекрёстков обьездная дорога проходит по земле, а мост содержит поперечная дорога - считать такие мосты не требуется. Хотя дорога двусторонняя, один тест исследует только одно направление обьездной дороги (например, с запада на восток).

Входные данные

Первая строка входного файла INPUT.TXT содержит число $N \leq 100$ – количество перекрёстков, перекрёстки перечислены с запада на восток. Следующие N строк содержат информацию о типе пересечения, затем о потоке поперечной дороги. Тип пересечения - английская буква L, означает что обьездная дорога проходит по земле, а поперечная проходит через мост, буква В означает что поперечная дорога проходит по земле, а обьездная через мост. Далее указаны 2 числа, показывающее количество автомобилей, покинувших обьездную дорогу, первое число - с поворотом налево, второе - с поворотом направо. Далее указаны 2 числа, показывающее количество автомобилей, выехавших на обьездную дорогу, первое число - с поворотом налево, второе - с поворотом направо. Количество машин по любой из поперечных дорог не превышает 10^9 .

Гарантируется корректность входных данных (изначально на трассе машин нет, покинуло трассу столько же машин, сколько и заехало на трассу, точка перекрёсток схода находится позднее перекрёстка захода).

Выходные данные

В выходной файл OUTPUT.TXT выводится одна строка, содержащая N чисел - потоки машин на мост обьездной дороги. Если обьездная дорога проходит по земле, а мост - над ней, то выводится -1.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 B 0 0 2 3 B 4 1 0 0	2 4
2	5 B 0 0 9 9 B 0 0 0 0 L 3 3 0 0 B 0 0 0 0 B 8 4 0 0	9 18 -1 12 8

```

/*
    "Клеверный лист": математическое моделирование
*/

#include <iostream>

typedef long long ll;

int main() {
    ll n, all = 0;
    std::cin >> n;
    while (n--) {
        char t; ll in_l, in_r, out_l, out_r;
        std::cin >> t >> out_l >> out_r >> in_l >> in_r;
        all -= out_l + out_r;
        ll answ = out_l + in_l + all;
        all += in_l + in_r;
        std::cout << (t == 'L' ? -1 : answ) << ' ';
    }
    return 0;
}

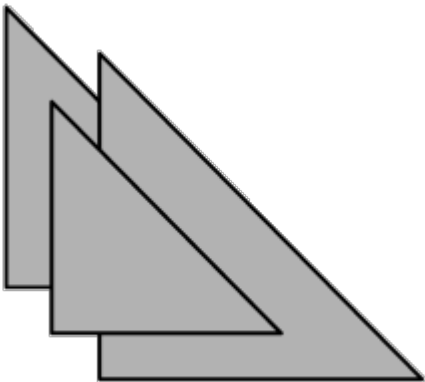
```

ЗАДАЧА №805

Атака инопланетян

(Время: 1 сек. Память: 16 Мб Сложность: 36%)

На город Энск нападает флот инопланетян. Флот состоит из n космических кораблей, каждый из которых имеет форму равнобедренного прямоугольного треугольника. Носом инопланетного корабля считается вершина, угол при которой прямой, а осью корабля называется высота, опущенная на гипотенузу.



Флот инопланетян прилетел с северо-востока, и застыл в таком положении, что все оси кораблей направлены строго на юго-запад.

Единственный способ нанести урон инопланетной армии это пустить из некоторой точки поверхности Земли лазерный луч вертикально вверх. Пущенный так луч прожигает насквозь все вражеские корабли, через которые он проходит (даже те, которые он задевает по границе). Но этот выстрел повредит инопланетянам только в случае, если все n кораблей будут при этом поражены.

Военные власти города Энска решили нанести удар по вражеским войскам. Для этого решено поставить лазер в одну из точек, над которыми находятся все n вражеских кораблей. Помогите военным определить площадь территории, на которой можно поставить лазер.

Входные данные

В первой строке входного файла INPUT.TXT содержится целое число n количество инопланетных кораблей ($1 \leq n \leq 100$). В каждой из следующих n строк описывается положение очередного корабля. Описание состоит из трех целых чисел x_i , y_i и s_i , где x_i и y_i координаты носа, а s_i размер корабля. Поскольку корабль имеет форму равнобедренного прямоугольного треугольника, размером корабля военные решили называть длину катета. Размеры кораблей положительные числа, не превышающие 1 000. Координаты носов кораблей не превышают по абсолютной величине 10^5 .

Выходные данные

В выходной файл OUTPUT.TXT выведите площадь территории, над которой находятся все инопланетные корабли. Выведите ответ с точностью до трех знаков после десятичной точки.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 2 4 6 4 2 7 3 3 5	4.500

/*
Пересечение двух равнобедренных треугольников со сторонами, параллельными осям координат

```

    Асимптотика  $O(n \log(n))$ 
*/

#include <stdio.h>
#include <algorithm>
#include <cmath>
#include <vector>
#include <cassert>

struct Triangle {
    int x, y, side;

    inline bool in(const int px, const int py) const {
        return px >= x && py >= y && py+px <= x+side+y;
    }

    static Triangle read() {
        int x, y, s;
        scanf("%d %d %d", &x, &y, &s);
        return Triangle{2*x, 2*y, 2*s};
    }
};

bool intersect(Triangle a, Triangle b, Triangle& c) {
    int x = std::max(a.x, b.x);
    int y = std::max(a.y, b.y);
    if (!a.in(x,y) || !b.in(x,y)) {
        c = Triangle{x,y,0};
        return false;
    }
    int low = 0, high = 1e9;
    while (high-low > 1) {
        int mid = (low+high) / 2;
        if (!a.in(x+mid,y) || !a.in(x,y+mid) || !b.in(x+mid,y) || !b.in(x,y+mid)) {
            high = mid;
        } else {
            low = mid;
        }
    }
    c = Triangle{x, y, low};
    return true;
}

int main() {
    int n;
    scanf("%d", &n);

    auto tr = Triangle::read();
    while (--n) {
        intersect(tr, Triangle::read(), tr);
    }
    printf("%.3f", tr.side * tr.side / 8.0);

    return 0;
}

```

ЗАДАЧА №1181

Число нулей на отрезке

(Время: 1 сек. Память: 16 Мб Сложность: 36%)

Задан числовой массив $A[1..N]$. Необходимо выполнить M операций вычисления количества нулевых элементов на отрезке $[L, R]$.

Входные данные

Первая строка входного файла INPUT.TXT содержит число N – размер массива ($N \leq 10^5$). Во второй строке записаны N чисел – элементы массива, целые числа от 0 до 10^5 . Третья строка содержит натуральное число M – количество запросов ($M \leq 30\,000$). Следующие M строк содержат пары чисел L и R ($1 \leq L \leq R \leq N$), описывающие отрезки.

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса выведите через пробел количество нулевых элементов.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 0 0 0 0 2 2 2 3 2 5	2 3

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <climits>

struct DataStruct {
    std::vector<int> arr, count;

    const int GSIZE = 128;

    DataStruct(int size = 0, int item = 0) {
        arr.assign(size, item);
        count.assign((size+GSIZE-1), 0);
        for (int g = 0; g < (int)count.size(); ++g) {
            const int begin = g * GSIZE;
            const int after = std::min(begin+GSIZE, size);
            for (int i = begin; i < after; ++i) {
                count[g] += (arr[i] == 0);
            }
        }
    }

    void set(int p, int v) {
        const int g = p / GSIZE;
        if (arr[p] == 0) {
```

```

        count[g]--;
    }
    arr[p] = v;
    if (arr[p] == 0) {
        count[g]++;
    }
}

int get(int l, int r) {
    const int gl = l / GSIZE;
    const int gr = r / GSIZE;
    int answ = 0;
    if (gl == gr) {
        for (int i = l; i <= r; ++i) {
            answ += (arr[i] == 0);
        }
    } else {
        for (int i = l, after = (gl+1)*GSIZE; i < after; ++i) {
            answ += (arr[i] == 0);
        }
        for (int g = gl+1; g < gr; ++g) {
            answ += count[g];
        }
        for (int i = gr * GSIZE; i <= r; ++i) {
            answ += (arr[i] == 0);
        }
    }
    return answ;
}

};

int main() {
    int n;
    scanf("%d", &n);
    DataStruct ds(n, 0);
    for (int i = 0; i < n; ++i) {
        int value;
        scanf("%d", &value);
        ds.set(i, value);
    }
    int q;
    scanf("%d", &q);
    while (q--) {
        int l, r;
        scanf("%d %d", &l, &r);
        --l, --r;
        printf("%d ", ds.get(l, r));
    }
    return 0;
}

```


ЗАДАЧА №1569

Задача без подвоха

(Время: 1 сек. Память: 16 Мб Сложность: 36%)

Перед вами задача, в которой отсутствует какой-либо подвох. Специальная комиссия изучила задачу на наличие подвохов и... Заверила, что количество подвохов в ней идеально — ровно ноль.

Если вы жаждите подвохов — эта задача не для вас.

Ну же, убедитесь в этом! Ах да, сама задача...

Вам дано множество цифр. Десятичных, никакого подвоха.

Вам нужно найти минимальное целое K такое, что $0 \leq K$ и что число 2^K содержит в своей десятичной записи все эти цифры.

Как видите, здесь нет подвоха. Просто напишите решение и получите свой Accepted!

Входные данные

В первой строке входного файла INPUT.TXT содержится целое число T ($1 \leq T \leq 32$). Подвох? Нет, это просто количество подтестов.

В каждой из следующих T строк содержится непустая строка, состоящая из различных десятичных цифр.

Выходные данные

Для каждого множества цифр в выходной файл OUTPUT.TXT выведите ответ в отдельной строке, спокойно и уверенно. Вы же не хотите, чтобы тестирующая система почувствовала подвох?

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5	1
	2	3
	8	5
	3	12
	69	17
	17	

```
#include <stdio.h>
#include <vector>
#include <string>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <cassert>
#include <sstream>
```

```
typedef unsigned long long ull;
typedef long long ll;
```

```

struct UInt {
    std::vector<int> digits;
    static const int BASE = (int)1e9;
    static const int WIDTH = 9;

    UInt (const int value) : digits(1, value) { }

    std::string to_string() const {
        std::stringstream os;
        os << digits.back();
        for (int i = (int)digits.size()-2; i >= 0; --i) {
            os << std::setw(UInt::WIDTH) << std::setfill('0') << digits[i];
        }
        return os.str();
    }

    UInt& normalize() {
        while (digits.size() > 1u && digits.back() == 0) {
            digits.pop_back();
        }
        return *this;
    }

    UInt& operator*= (const int value) {
        ll carry = 0;
        for (int i = 0; carry || i < (int)digits.size(); ++i) {
            if (i >= (int)digits.size()) {
                digits.push_back(0);
            }
            carry = carry + ll(digits[i]) * value;
            digits[i] = carry % BASE;
            carry /= BASE;
        }
        return *this;
    }

    std::string set_of_digits() const {
        auto s = to_string();
        std::sort(s.begin(), s.end());
        s.erase(std::unique(s.begin(), s.end()), s.end());
        return s;
    }
};

std::ostream& operator<<(std::ostream& os, const UInt& num) {
    return os << num.to_string();
}

int solve(std::string need) {
    //std::sort(need.begin(), need.end());
    UInt pow = 1;
    int answ = -1;
    for (int i = 0; i <= 300; ++i) {
        auto s = pow.set_of_digits();
        bool flag = true;
        for (auto c : need) {
            if (int(s.find(c)) == -1) {
                flag = false;
                break;
            }
        }
        if (flag) {
            answ = i;
            break;
        }
        pow *= 2;
    }
}

```

```
        return answ;
    }

int main() {
    int nQ; scanf("%d", &nQ);
    while (nQ-->0) {
        char buf[11]; scanf("%10s", buf);
        printf("%d\n", solve(buf));
    }
    return 0;
}
```

ЗАДАЧА №144

A*B

(Время: 1 сек. Память: 16 Мб Сложность: 37%)

Даны два целых неотрицательных числа A и B. Требуется найти их произведение.

Входные данные

Во входном файле INPUT.TXT записаны целые неотрицательные числа A и B по одному в строке ($A < 10^{100}$, $B \leq 10000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное число без лидирующих нулей: A*B.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 7	35

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <cstdint>
#include <cassert>
#include <functional>

struct Int;

std::istream& operator>>(std::istream&, Int&);
std::ostream& operator<<(std::ostream&, const Int&);
bool operator<(const Int&, const Int&);
bool operator>(const Int&, const Int&);
bool operator==(const Int&, const Int&);
bool operator!=(const Int&, const Int&);
bool operator<=(const Int&, const Int&);
bool operator>=(const Int&, const Int&);
bool operator!==(const Int&, const Int&);
Int operator+(const Int&, const Int&);
Int operator+(const int, const Int&);
Int operator-(const Int&, const Int&);
Int operator*(const int, const Int&);
Int operator/(const Int& a, const int);
int operator%(const Int& a, const int);
Int pow(Int, int);

struct Int {
    static const int BASE = (int)1e9; // Основание системы счисления
    static const int WIDTH = 9;      // Количество десятичных цифр, которые хранятся в одн

    // Вектор под цифры числа:
    std::vector<int> digits;

    // Нормализуем вид числа - удаляем лидирующие нули
```

```

Int& to_normal() {
    while (digits.back() == 0 && (int)digits.size() > 1) {
        digits.pop_back();
    }
    return *this;
}

// Конструктор от короткого целого
Int (int64_t number = 0) {
    assert(number >= 0);
    do {
        digits.push_back(number % BASE);
        number /= BASE;
    } while (number > 0);
    to_normal();
}

// Конструктор от вектора из цифр:
Int (const std::vector<int>& digits) : digits(digits) { to_normal(); }

// Конструктор от строки:
Int (std::string s) {
    const int size = (int)s.size();
    for (int idGroup = 1, nGroups = size / WIDTH; idGroup <= nGroups; ++idGroup) {
        digits.push_back(std::stoi(s.substr(size-idGroup * WIDTH, WIDTH)));
    }
    if (size % WIDTH != 0) {
        digits.push_back(std::stoi(s.substr(0, size % WIDTH)));
    }
    to_normal();
}

// Прибавление:
Int& operator+=(const int num) {
    if (num >= BASE) {
        return *this += Int(num);
    }
    int rem = num;
    for (int i = 0; rem > 0; ++i) {
        if (i >= size()) digits.push_back(0);
        rem += at(i);
        at(i) = rem % BASE;
        rem /= BASE;
    }
    return this->to_normal();
}

// Прибавление:
Int& operator+=(const Int& other) {
    if (other.size() == 1) {
        return *this += other.at(0);
    }
    const int s1 = this->size();
    const int s2 = other.size();
    int rem = 0;
    for (int i = 0; i < s1 || i < s2 || rem > 0; ++i) {
        int d1 = i < s1 ? this->at(i) : 0;
        int d2 = i < s2 ? other.at(i) : 0;
        rem += d1 + d2;
        if (i >= s1) digits.push_back(0);
        at(i) = rem % BASE;
        rem /= BASE;
    }
    return this->to_normal();
}

// Вычитание короткого:
Int& operator-=(const int num) {
    if (num >= BASE) {

```

```

        return *this -= Int(num);
    }
    int rem = -num;
    for (int i = 0; rem < 0; ++i) {
        assert(i < size());
        rem += at(i);
        if (rem < 0) {
            at(i) = (rem + BASE) % BASE;
            rem = -1;
        } else {
            at(i) = rem % BASE;
            rem = 0;
        }
    }
    return this->to_normal();
}

// Вычитание длинного:
Int& operator-=(const Int& other) {
    if (other.size() == 1) {
        return *this -= other.at(0);
    }
    assert(*this >= other);
    const int s1 = this->size();
    const int s2 = other.size();
    int rem = 0;
    for (int i = 0; i < s1 || i < s2; ++i) {
        int d1 = i < s1 ? this->at(i) : 0;
        int d2 = i < s2 ? other.at(i) : 0;
        rem += d1 - d2;
        if (i >= s1) digits.push_back(0);
        if (rem < 0) {
            at(i) = (rem + BASE) % BASE;
            rem = -1;
        } else {
            at(i) = rem % BASE;
            rem = 0;
        }
    }
    return this->to_normal();
}

// Умножение на короткое:
Int& operator*=(const int num) {
    // std::cout << "call Int& operator*=(const int num)" << std::endl;
    if (num >= BASE) {
        return *this *= Int(num);
    }
    int64_t rem = 0;
    for (int i = 0; i < size() || rem > 0; ++i) {
        // std::cout << "i = " << i << std::endl;
        if (i >= size()) digits.push_back(0);
        rem += 1LL * at(i) * num;
        at(i) = rem % BASE;
        rem /= BASE;
    }
    return this->to_normal();
}

// Умножение:
Int operator*(const int num) const {
    return num >= BASE ? *this * Int(num) : Int(*this) *= num;
}

// Произведение:
Int operator*(const Int& other) const {
    // std::cout << "call Int operator*(const Int& other)" << std::endl;
    if (other.size() == 1) {
        return *this * other.at(0);
    }

```

```

    }
    const int s1 = this->size();
    const int s2 = other.size();
    std::vector<int> temp(s1+s2);
    for (int i = 0; i < s1; ++i) {
        int64_t rem = 0;
        for (int j = 0; j < s2; ++j) {
            rem += temp.at(i+j) + 1LL * this->at(i) * other.at(j);
            temp.at(i+j) = rem % BASE;
            rem /= BASE;
        }
        if (rem > 0) {
            temp.at(i+s2) += rem;
            assert(0 <= temp.at(i+s2) && temp.at(i+s2) < BASE);
        }
    }
    return Int(temp);
}

// Умножение:
Int& operator*=(const Int& other) {
    // std::cout << "call Int& operator*=(const Int& other)" << std::endl;
    return other.size() == 1 ? *this *= other.at(0) : *this = *this * other;
}

// Деление на короткое:
Int& operator/=(const int num) {
    // std::cout << "call Int& operator/=(const int num)" << std::endl;
    if (num >= BASE) {
        return *this /= Int(num);
    }
    assert(0 < num && num < BASE);
    int64_t rem = 0;
    for (int j = size()-1; j >= 0; --j) {
        // std::cout << "j = " << j << std::endl;
        (rem *= BASE) += at(j);
        // std::cout << "at(j) = " << at(j) << std::endl;
        at(j) = rem / num;
        assert(0 <= at(j) && at(j) < BASE);
        rem %= num;
    }
    // std::cout << "gool end!" << std::endl;
    return this->to_normal();
}

// Взятие остатка от деления:
Int& operator%=(const int num) {
    return *this = *this % num;
}

static Int div(Int a, Int b) {
    // Данный метод работает за O(m*n)
    // Условия:
    // base^(m-1) <= a < base^m (1)
    // base^n / 2 <= b < base^n (2)
    // Условие (1) выполняется всегда для m = a.size()
    // Условие (2) необходимо обеспечить равносильным преобразованием следующим образом
    if (2 * b.digits.back() < Int::BASE) {
        int d = (1LL * Int::BASE + 2 * b.digits.back() - 1) / (2 * b.digits.back());
        a *= d;
        b *= d;
    }

    const int n = b.size(), m = a.size();

    std::function<Int(Int, Int)> special_div = [&special_div](const Int& a, const Int&
        // Данный метод работает за O(n), при следующих условиях:
        // 0 <= a <= base^(n+1)
        // b^n / 2 <= b < base^n

```

```

        const int n = b.size();
        // Проверки на соответствии условиям метода:
        assert(a.size() - b.size() == 1);
        if (a >= b * BASE) {
            return Int(BASE) + special_div(a-b * BASE, b);
        } else {
            int64_t q = (BASE * 1LL * a.at(n) + a.at(n-1)) / b.at(n-1);
            assert(0 <= q && q < BASE);
            auto t = q * b;
            if (t > a) { --q, t -= b; }
            if (t > a) { --q, t -= b; }
            assert(t <= a);
            return Int(q);
        }
    };

    if (m < n) {
        return 0; // O(1)
    } else if (m == n) {
        return a >= b ? 1 : 0; // O(n)
    } else if (m == n+1) {
        return special_div(a, b); // O(n)
    } else {
        Int a_temp = std::vector<int>(a.digits.begin()+m-n-1, a.digits.begin()+m); // O
        Int s = std::vector<int>(a.digits.begin(), a.digits.begin()+m-n-1); // O(n)
        Int q_temp = special_div(a_temp, b); // O(n)
        Int r_temp = a_temp - q_temp * b; // O(n)
        Int q = div(r_temp.shift_left(m-n-1)+s, b); // O(m-n) рекурсивных вызовов -> об
        return q_temp.shift_left(m-n-1) + q; // O(n)
    }
}

Int operator/(const Int& other) const {
    // std::cout << "call operator/(const Int& other)" << std::endl;
    return (other.size() == 1) ? *this / other.at(0) : div(*this, other);
}

Int& operator/=(const Int& other) {
    // std::cout << "call Int& operator/=(const Int& other)" << std::endl;
    return *this = *this / other;
}

Int operator%(const Int& other) const {
    // std::cout << "call operator%(const Int& other)" << std::endl;
    return *this - *this / other * other;
}

Int& operator%=(const Int& other) {
    return *this = *this % other;
}

// Сравнение: result < 0 (меньше), result == 0 (равно), result > 0 (больше)
int compare(const Int& other) const {
    if (this->size() > other.size()) return 1;
    if (this->size() < other.size()) return -1;
    for (int i = size()-1; i >= 0; --i) {
        if (this->at(i) > other.at(i)) return 1;
        if (this->at(i) < other.at(i)) return -1;
    }
    return 0;
}

int& at(int pos) {
    assert(0 <= pos && pos < size());
    return digits.at(pos);
}

const int& at(int pos) const {
    assert(0 <= pos && pos < size());

```



```

        return digits.at(pos);
    }

    inline int size() const {
        return (int)digits.size();
    }

    Int& shift_left(int pow) {
        assert(pow >= 0);
        digits.resize((int)digits.size()+pow);
        for (int i = (int)digits.size()-1; i >= pow; --i) {
            digits[i] = digits[i-pow];
        }
        for (int i = pow-1; i >= 0; --i) {
            digits[i] = 0;
        }
        return to_normal();
    }

    Int& shift_right(int pow) {
        assert(pow >= 0);
        if (pow >= (int)digits.size()) {
            return *this = 0;
        }
        for (int i = 0; i + pow < (int)digits.size(); ++i) {
            digits[i] = digits[i+pow];
        }
        digits.resize((int)digits.size()-pow);
        return to_normal();
    }
};

// Ввод из потока:
std::istream& operator>>(std::istream& is, Int& number) {
    std::string s;
    is >> s;
    number = Int(s);
    return is;
}

// Вывод в поток:
std::ostream& operator<<(std::ostream& os, const Int& number) {
    os << number.digits.back();
    for (int i = (int)number.digits.size()-2; i >= 0; --i) {
        os << std::setw(Int::WIDTH) << std::setfill('0') << number.digits[i];
    }
    return os << std::setfill(' ');
}

// Сложение:
Int operator+(const Int& a, const Int& b) {
    return Int(a) += b;
}

// Вычитание:
Int operator-(const Int& a, const Int& b) {
    return Int(a) -= b;
}

// Умножение:
Int operator*(const int a, const Int& b) {
    return b * a;
}

// Деление:
Int operator/(const Int& a, const int num) {
    // std::cout << "operator/(const Int& a, const int num)" << std::endl;
    return Int(a) /= num;
}

```

```

// Остаток от деления:
int operator%(const Int& a, const int num) {
    int64_t rem = 0;
    for (int i = a.size()-1; i >= 0; --i) {
        ((rem *= Int::BASE) += a.at(i)) %= num;
    }
    return rem;
}

// Возведение в степень:
Int pow(Int a, int n) {
    Int res = 1;
    while (n > 0) {
        if (n % 2 != 0) {
            res *= a;
        }
        a *= a;
        n /= 2;
    }
    return res;
}

// Операторы сравнения:
bool operator<(const Int& a, const Int& b) { return a.compare(b) < 0; }
bool operator>(const Int& a, const Int& b) { return a.compare(b) > 0; }
bool operator==(const Int& a, const Int& b) { return a.compare(b) == 0; }
bool operator<=(const Int& a, const Int& b) { return a.compare(b) <= 0; }
bool operator>=(const Int& a, const Int& b) { return a.compare(b) >= 0; }
bool operator!=(const Int& a, const Int& b) { return a.compare(b) != 0; }

int main() {
    Int a, b;
    std::cin >> a >> b;
    std::cout << a * b << std::endl;
    return 0;
}

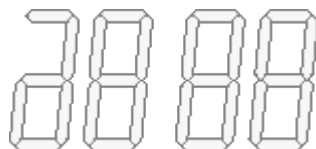
```

ЗАДАЧА №175

Наручные часы

(Время: 1 сек. Память: 16 Мб Сложность: 37%)

Вы приобрели новые электронные наручные часы с жидкокристаллическим дисплеем. Дисплей отображает часы и минуты с помощью четырех элементов, каждый из которых отображает одну цифру.



Три из них состоят из семи полосок, каждая из которых может быть либо белой (неотличимой от фона), либо черной. Вид такого элемента и отображаемые им цифры показаны на рисунке:



Четвертый элемент предназначен для отображения старшей цифры часа. Если она равна нулю, то элемент полностью неактивен (все полоски белые), иначе показывается соответствующая цифра. Вот как выглядит этот элемент с цифрами:



Вам хочется проверить: все ли в порядке с новым приобретением, а именно, нет ли таких полосок в каком-либо из элементов, которые либо всегда белые, либо всегда черные. Вы хотите начать проверку в некоторое начальное время. Требуется определить, сколько Вам потребуется минут для убеждения в исправности часов.

Входные данные

В первой строке входного файла INPUT.TXT находится время начала проверки в формате HH:MM. Часы и минуты записаны с лидирующими нулями, если таковые имеются. ($00 \leq HH \leq 23$, $00 \leq MM \leq 59$).

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное число минут, необходимое для проверки Ваших часов, если она началась в заданное время.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	00:00	1200

2	02:39	1041
---	-------	------

```
#pragma GCC diagnostic ignored "-Wunused-result"
```

```
#include <stdio.h>
```

```
#include <bits/stdc++.h>
```

```
struct Record {
```

```
    bool up, left1, right1, mid, left2, right2, down;
```

```
    bool good() const {
```

```
        return up && left1 && right1 && mid && left2 && right2 && down;
```

```
    }
```

```
    Record& update(const Record& other) {
```

```
        up    |= other.up;
```

```
        left1 |= other.left1;
```

```
        right1 |= other.right1;
```

```
        mid    |= other.mid;
```

```
        left2  |= other.left2;
```

```
        right2 |= other.right2;
```

```
        down   |= other.down;
```

```
        return *this;
```

```
    }
```

```
    Record inverse() const {
```

```
        return Record{!up, !left1, !right1, !mid, !left2, !right2, !down};
```

```
    }
```

```
};
```

```
const Record digits[10] = {
```

```
    Record{1,1,1,0,1,1,1},
```

```
    Record{0,0,1,0,0,1,0},
```

```
    Record{1,0,1,1,1,0,1},
```

```
    Record{1,0,1,1,0,1,1},
```

```
    Record{0,1,1,1,0,1,0},
```

```
    Record{1,1,0,1,0,1,1},
```

```
    Record{1,1,0,1,1,1,1},
```

```
    Record{1,0,1,0,0,1,0},
```

```
    Record{1,1,1,1,1,1,1},
```

```
    Record{1,1,1,1,0,1,1}
```

```
};
```

```
std::string get_time_str(int m) {
```

```
    char buf[5]; sprintf(buf, "%02d%02d", m / 60 % 24, m % 60);
```

```
    std::string s = buf;
```

```
    assert(s.size() == 4u);
```

```
    return s;
```

```
}
```

```
int main() {
```

```
    std::vector<Record> on, off, curr;
```

```
    on.push_back({0, 1, 0, 0, 0, 0, 0});
```

```
    on.push_back({0, 0, 0, 0, 0, 0, 0});
```

```
    on.push_back({0, 0, 0, 0, 0, 0, 0});
```

```
    on.push_back({0, 0, 0, 0, 0, 0, 0});
```

```
    curr = off = on;
```

```
    std::function<bool()> good = [&]() {
```

```
        for (auto& it : on) {
```

```
            if (!it.good()) return false;
```

```
        }
```

```
        for (auto& it : off) {
```

```
            if (!it.good()) return false;
```

```
        }
```

```
        return true;
```

```
    };
```

```
    int h, m; scanf("%d:%d", &h, &m);
```

```

m = 60 * h + m;

int inc = 0;
while (!good()) {
    assert(on[0].left1 == true && off[0].left1 == true);
    auto s = get_time_str(m + inc);
    for (int i = 0; i < 4; ++i) {
        if (i == 0 && s[i] == '0') {
            curr[i] = Record{0, 1, 0, 0, 0, 0, 0};
        } else {
            curr[i] = digits[s[i] - '0'];
        }
        on[i].update(curr[i]);
        off[i].update(curr[i].inverse());
    }
    ++inc;
}
printf("%d\n", inc-1);
return 0;
}

```

ЗАДАЧА №329

Лесенка-2

(Время: 1 сек. Память: 16 Мб Сложность: 37%)

Вова стоит перед лесенкой из N ступеней. На каждой из ступеней написаны произвольные целые числа. Первым шагом Вова может перейти на первую ступень или, перепрыгнув через первую, сразу оказаться на второй. Также он поступает и дальше, пока не достигнет N -ой ступени. Посчитаем сумму всех чисел, написанных на ступенях через которые прошел Вова.

Требуется написать программу, которая определит оптимальный маршрут Вовы, при котором, шагая, он получит наибольшую сумму.

Входные данные

Входной файл INPUT.TXT содержит в первой строке натуральное число N – количество ступеней лестницы ($2 \leq N \leq 1000$). Во второй строке через пробел заданы числа, написанные на ступенях лестницы, начиная с первой. Числа, написанные на ступенях, не превосходят по модулю 1000.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать в первой строке наибольшее значение суммы. Во второй строке должны быть записаны через пробел номера ступеней по возрастанию, по которым должен шагать Вова. Если существует несколько различных правильных маршрутов, то можно вывести любой из них.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 1 2 1	4 1 2 3
2	3 1 -1 1	2 1 3

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    int n;
    std::cin >> n;

    std::vector<int> a(1+n);
    for (int i = 1; i <= n; ++i) {
        std::cin >> a[i];
    }

    std::vector<int> best(1+n), from(1+n);
    for (int i = 1; i <= n; ++i) {
        best[i] = best[i-1] + a[i];
        from[i] = i-1;
        if (i-2 >= 0 && best[i-2] + a[i] > best[i]) {
            best[i] = a[i] + best[i-2];
            from[i] = i-2;
        }
    }

    std::cout << best[n] << "\n";
    for (int i = 1; i <= n; ++i) {
        std::cout << from[i] << " ";
    }
    std::cout << "\n";
}
```

```

    }
}

std::vector<int> answ;
for (int curr = n; curr >= 1; curr = from[curr]) {
    answ.push_back(curr);
}
std::reverse(answ.begin(), answ.end());

std::cout << best[n] << std::endl;
for (auto& it : answ) {
    std::cout << it << ' ';
}
std::cout << std::endl;
return 0;
}

```

Есть ли жизнь на Марсе?

(Время: 1 сек. Память: 16 Мб Сложность: 37%)

- Ты врешь, Коля! На Марсе жизни нет! Кто тебе такую чушь сказал?

- Петя. А ему сказал Саша.

- Да от Пети я в жизни правдивого слова не слышал! Ему что ни скажут, он все переврет. Да и Саше откуда знать?

- А Саше рассказал про это Владимир Алексеевич, наш учитель биологии.

- Ну, Владимиру Алексеевичу-то можно верить... Только вряд ли он так сказал, это либо Саша, либо Петя придумал. А может, это ты меня разыгрываешь?..

- Минуточку, ребята, - вмешался подошедший к спорящим учитель математики, Глеб Тимофеевич, - давайте подойдем к проблеме формально. Допустим, что все диалоги - Владимира Алексеевича с Сашей, Саши с Петей и Пети с Колей - действительно имели место. Пронумеруем ребят числами 1, 2 и 3. Предположим также, что каждый из ребят независимо друг от друга передал полученную информацию относительно жизни на Марсе верно с вероятностью p_i , а соврал с вероятностью $q_i = 1 - p_i$ для $i = 1, 2, 3$. Вероятности – это вещественные числа от нуля до единицы; событие, имеющее вероятность 0, никогда не произойдет, событие же с вероятностью 1 произойдет без всякого сомнения. Зная, что Коля после этого объявил, что жизнь на Марсе все-таки есть, найдите по данным p_i вероятность того, что так действительно сказал Владимир Алексеевич.

- А как искать эту вероятность? И что значит независимо друг от друга? – растерялись ребята.

- Независимость означает, что действие одного из ребят никак не отражается на том, как поступят другие. К примеру, Пете неважно, соврал ли Саша - в любом случае он передаст сказанное Сашей правильно с вероятностью ровно p_2 . Задача несложная, и можно рассмотреть все восемь возможных случаев. Первый случай - все ребята говорили правду, и вероятность этого случая равна $p_1 \cdot p_2 \cdot p_3$. В этом случае жизнь на Марсе, без сомнения, есть - Владимиру Алексеевичу мы верим, а ребята передали его слова правильно. Второй случай, когда соврал только Саша, имеет место с вероятностью $q_1 \cdot p_2 \cdot p_3$, и в этом случае жизни на Марсе нет. Далее переберем остальные шесть случаев, каждый раз перемножая соответствующие вероятности, а потом просуммируем вероятности тех случаев, в которых слова учителя переданы правильно. То, что вероятности для отдельных ребят в каждом случае надо перемножить - это и есть формальное определение независимости. Ну, в скольких случаях будет передано именно то, что говорил Владимир Алексеевич?

- В одном ...

- А вот и нет. Например, если Петя и Коля соврали, а Саша сказал правду, то истина, дважды исказившись, дойдет до нас в неизменном виде. И вообще, четное количество отрицаний, примененных к утверждению, дает само утверждение. В нашей задаче случаев с четным количеством отрицаний - четыре, и итоговая вероятность равна $p_1 \cdot p_2 \cdot p_3 + q_1 \cdot q_2 \cdot p_3 + q_1 \cdot p_2 \cdot q_3 + p_1 \cdot q_2 \cdot q_3$.

- То есть если Петя и Коля точно соврут, а Саша точно скажет правду, то от Коли мы услышим в точности то, что говорил учитель?

- Совершенно верно. А теперь решите-ка задачу для общего случая, когда ребят не трое, а n . Первому, кто решит - пятерка на следующей контрольной!

Входные данные

Входной файл INPUT.TXT содержит целое число n ($1 \leq n \leq 100$). Во второй строке через пробел записаны n вещественных чисел - это числа p_1, p_2, \dots, p_n ($0 \leq p_i \leq 1$). Числа даны с не более чем шестью десятичными знаками после запятой.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно вещественное число, округленное до шести знаков после запятой - вероятность существования жизни на Марсе.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 1 0.1 0.9	0.18

```
#include <stdio.h>
#include <tuple>

int main() {
    int n; scanf("%d", &n);
    double p = 1, q = 0;
    while (n--) {
        double cur;
        scanf("%lf", &cur);
        std::tie(p, q) = std::make_tuple(
            p * cur + q * (1-cur),
            p * (1-cur) + q * cur
        );
    }
    printf("%.6f", p);
    return 0;
}
```

ЗАДАЧА №578

Система счисления

(Время: 1 сек. Память: 16 Мб Сложность: 37%)

Вчера на уроке математики Саша узнал о том, что иногда полезно использовать вместо десятичной системы счисления какую-нибудь другую.

Однако, учительница не объяснила, почему в системе счисления по основанию b в качестве цифр выбирают числа от 0 до $b - 1$.

Немного подумав, Саша понял, что можно выбирать и другие наборы цифр. Например, вместо троичной системы счисления можно рассмотреть систему счисления, где вместо обычных цифр 0, 1, 2 есть цифры 1, 2 и 3.

Саша заинтересовался вопросом, а как перевести число n в эту систему счисления? Например, число 7 в этой системе записывается как 21, так как $7 = 2 \cdot 3 + 1$, а число 22 записывается как 211, так как $22 = 2 \cdot 9 + 1 \cdot 3 + 1$.

Входные данные

Входной файл INPUT.TXT содержит натуральное число n , $1 \leq n \leq 2 \cdot 10^9$.

Выходные данные

В выходной файл OUTPUT.TXT выведите число n записанное в указанной системе счисления.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	7	21
2	22	211

```
#include <iostream>
#include <algorithm>
#include <cassert>

int main() {
    int n; std::cin >> n;
    std::string s;
    do {
        s.push_back(n % 3 + '0');
        n /= 3;
    } while (n > 0);
    std::reverse(s.begin(), s.end());
    bool flag = true;
    while (flag) {
        flag = false;
        for (int i = 0; i+1 < (int)s.size(); ++i) {
            if (s[i+1] != '0') {
                continue;
            }
            if (s[i] > '0' && s[i+1] == '0') {
                s[i]--;
            }
        }
    }
}
```

```
        s[i+1] = '3';
        flag = true;
    }
}
int pos = 0;
while (s[pos] == '0') ++pos;
assert(pos < (int)s.size() && s[pos] != '0');
std::cout << s.substr(pos);
return 0;
}
```

ЗАДАЧА №638

Всероссийская олимпиада по информатике

(Время: 1 сек. Память: 16 Мб Сложность: 37%)

2127 год. Прошло уже много лет с тех пор, как состоялась первая Всероссийская олимпиада по информатике. Как и многие другие соревнования, наши олимпиады теперь проводятся в несколько дней. Теперь даже задача выбора подходящего времени для олимпиады представляет определенные трудности. Ведь на разных планетах, входящих в состав Российской Федерации используются разные способы отсчета времени: длина месяца, количество дней в неделе и те дни, по которым невозможно проведение олимпиады, могут различаться. Возникла необходимость написания программы, которая поможет решить эту задачу. И тогда в жюри вспомнят, что уже сейчас мы предвидели такую ситуацию и предложили вам решить подобную задачу.

В качестве первого шага найдите количество способов выбрать время проведения олимпиады.

Входные данные

В первой строке входного файла INPUT.TXT содержатся два целых числа n и k ($1 \leq k \leq n \leq 100000$) - количество дней месяца и продолжительность олимпиады соответственно. Во второй строке задаются количество дней в неделе w , количество дней, запрещенных еженедельно, d_w и день недели, на который приходится первый день месяца s ($1 \leq s \leq w \leq n$, $0 \leq d_w \leq w$). Третья строка содержит d_w номеров дней недели (например, выходных), в которые нельзя проводить олимпиаду. В четвертой строке записано количество дней месяца d_m , не подходящих для проведения олимпиады по причинам отличным от еженедельного расписания (например, такими днями являются государственные праздники). Последняя строка содержит d_m целых чисел - номера этих дней. Дни месяца так же нумеруются начиная с 1. Заметим, что некоторые дни могут быть запрещенными сразу по обоим причинам.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное целое число - количество способов выбрать k подряд идущих дней, в которые возможно проведение олимпиады.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	31 3 7 1 7 7 2 1 9	15

```
#include <stdio.h>
#include <vector>

int main() {
    int n, duration, daysPerWeek, nBlockedPerWeek, first;
    scanf("%d %d %d %d %d", &n, &duration, &daysPerWeek, &nBlockedPerWeek, &first);
    std::vector<bool> blockedPerWeek(1+daysPerWeek);
    while (nBlockedPerWeek--) {
```

```

        int it;
        scanf("%d", &it);
        blockedPerWeek[it] = true;
    }
    std::vector<bool> blocked(1+n);
    for (int i = 1, cur = first; i <= n; ++i) {
        if (blockedPerWeek[cur]) {
            blocked[i] = true;
        }
        ++cur;
        if (cur > daysPerWeek) cur = 1;
    }

    int nBlockedPerMonth;
    scanf("%d", &nBlockedPerMonth);
    while (nBlockedPerMonth-->0) {
        int it;
        scanf("%d", &it);
        blocked[it] = true;
    }

    int nBlocked = 0;
    for (int i = 1; i <= duration; ++i) {
        nBlocked += blocked[i];
    }
    int answ = (nBlocked == 0);
    int l = 2, r = duration+1;
    while (r <= n) {
        nBlocked += blocked[r] - blocked[l-1];
        answ += (nBlocked == 0);
        ++r, ++l;
    }
    printf("%d", answ);

    return 0;
}

```

ЗАДАЧА №789

Последовательность - 3

(Время: 1 сек. Память: 16 Мб Сложность: 37%)

Найдите n -й элемент строго возрастающей последовательности, которая описывается следующими правилами:

1. число 1 является элементом последовательности;
2. если a – элемент последовательности, то $2a$, $3a$, $5a$ тоже являются элементами последовательности;
3. последовательности принадлежат только элементы, заданные правилами 1 и 2.

Входные данные

Входной файл INPUT.TXT содержит одно натуральное число n ($n \leq 10000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – n -й элемент последовательности.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2	2
2	9	10

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <cassert>

typedef unsigned long long Int;

int main() {
    std::vector<Int> numbers;
    for (Int pow2 = 1; pow2 <= 1e18; pow2 *= 2) {
        for (Int pow3 = 1; pow2 * pow3 <= 1e18; pow3 *= 3) {
            for (Int pow5 = 1; pow2 * pow3 * pow5 <= 1e18; pow5 *= 5) {
                numbers.push_back(pow2 * pow3 * pow5);
            }
        }
    }
    std::sort(numbers.begin(), numbers.end());

    int n; std::cin >> n;
    assert((int)numbers.size() > n);
    std::cout << numbers[n-1];
    return 0;
}
```

ЗАДАЧА №840

Шары

(Время: 2 сек. Память: 16 Мб Сложность: 37%)

В пространстве расположен шар, заданный координатами своего центра X, Y, Z и радиусом R . К нему добавляется не более N новых шаров, которые также описываются координатами центров X_i, Y_i, Z_i и радиусами R_i ($1 \leq i \leq N$).

Шары добавляются до тех пор, пока есть хоть один шар, не пересекающийся с другими шарами. Требуется определить номер шара, после которого процесс добавления шаров можно прекратить.

Примечание:

- Взаимное пересечение абсолютно всех шаров не требуется, т.е. решением является и наличие непересекающихся групп пересекающихся шаров;
- Касание шаров не является пересечением.
- Вложение одного шара в другой является пересечением.

Входные данные

Входной файл INPUT.TXT содержит четыре вещественных числа разделенных пробелами X, Y, Z, R - параметры исходного шара ($|X, Y, Z| \leq 30000$; $0 < R \leq 30000$). Вторая строка содержит целое число N - количество шаров, которые предполагается добавить ($1 \leq N \leq 5000$). Следующие N строк содержат по четыре вещественных числа X_i, Y_i, Z_i, R_i - параметры i -го шара ($1 \leq i \leq N$; $|X_i, Y_i, Z_i| \leq 30000$; $0 < R_i \leq 30000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите номер шара, после которого процесс добавления шаров можно остановить, или 0 - если даже после добавления всех имеющихся шаров остается хотя бы один шар, не пересекающийся с другими.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	2 2 1 1 3 5 2 2 1 3 3 3 1.5 8 8 8 1	2

```
#include <stdio.h>
#include <vector>

struct Ball {
    double x, y, z, r;

    static Ball read() {
        double x, y, z, r;
        scanf("%lf %lf %lf %lf", &x, &y, &z, &r);
        return Ball{x, y, z, r};
    }
};
```

```

}

inline bool intersect(const Ball& other) const {
    double dx = x - other.x, dy = y - other.y, dz = z - other.z;
    return dx*dx+dy*dy+dz*dz < (r + other.r)*(r + other.r);
}

};

int main() {
    std::vector<bool> alone{true};
    std::vector<Ball> balls{Ball::read()};
    int nAlone = 1;

    int n;
    scanf("%d", &n);

    for (int i = 1; i <= n; ++i) {
        // Добавление нового шара:
        alone.push_back(true);
        balls.push_back(Ball::read());
        ++nAlone;

        // Пересечение со всеми предыдущими шарами:
        for (int j = i-1; j >= 0; --j) {
            if (balls[j].intersect(balls[i])) {
                if (alone[j]) {
                    alone[j] = false;
                    --nAlone;
                }
                if (alone[i]) {
                    alone[i] = false;
                    --nAlone;
                }
            }
        }

        // Проверка на существование обособленного шара:
        if (nAlone == 0) {
            printf("%d", i);
            return 0;
        }
    }
    printf("0");
    return 0;
}

```


ЗАДАЧА №1179

Суперминимум

(Время: 1 сек. Память: 32 Мб Сложность: 37%)

Дано N чисел. Требуется для каждого K подряд идущих чисел найти минимальное среди них.

Входные данные

В первой строке входного файла INPUT.TXT записаны два натуральных числа N и K ($N \leq 150000$, $K \leq 10000$, $K \leq N$), разделенные пробелом. Во второй строке записано N целых чисел через пробел. Числа находятся в диапазоне от -32768 до 32767.

Выходные данные

В выходной файл OUTPUT.TXT выведите N-K+1 целое число – минимальные значения для каждого K подряд идущих чисел.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	11 3 8 764 1 3 85 2 4 5 77 1 5	1 1 1 2 2 2 4 1 1

```
#include <stdio.h>
#include <queue>

struct Pair {
    int number, pos;
};

bool operator<(const Pair& a, const Pair& b) {
    return a.number > b.number || (a.number == b.number && a.pos > b.pos);
}

int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    std::priority_queue<Pair> queue;
    for (int i = 0; i < k; ++i) {
        int value; scanf("%d", &value);
        queue.push(Pair{value, i});
    }
    for (int i = k; i <= n; ++i) {
        while (queue.top().pos < i-k) {
            queue.pop();
        }
        printf("%d ", queue.top().number);
        if (i < n) {
            int value; scanf("%d", &value);
            queue.push(Pair{value, i});
        }
    }
    return 0;
}
```

ЗАДАЧА №267

Ксерокопии

(Время: 0,2 сек. Память: 16 Мб Сложность: 38%)

Секретарша Ирочка сегодня опоздала на работу и ей срочно нужно успеть к обеду сделать N копий одного документа. В ее распоряжении имеются два ксерокса, один из которых копирует лист за x секунд, а другой – за y секунд. (Разрешается использовать как один ксерокс, так и оба одновременно. Можно копировать не только с оригинала, но и с копии.) Помогите ей выяснить, какое минимальное время для этого потребуется.

Входные данные

Во входном файле INPUT.TXT записаны три натуральных числа N , x и y , разделенные пробелом ($1 \leq N \leq 2 \cdot 10^8$, $1 \leq x, y \leq 10$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – минимальное время в секундах, необходимое для получения N копий.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 1 1	3
2	5 1 2	4

```
/*
    Задача: 267. Ксерокопии

    Решение: нод, перебор, математика,  $O(x+y)$ 

    Автор: Дмитрий Козырев, github: dm kz, e-mail: dmkozyrev@rambler.ru
*/
```

```
#include <bits/stdc++.h>
template<typename T>
T gcd(T a, T b) {
    return b == 0 ? a : gcd(b, a % b);
}
uint32_t solve(uint32_t n, uint32_t x, uint32_t y) {
    auto g = x / gcd(x, y) * y;
    uint32_t cx = g / x;
    uint32_t cy = g / y;
    uint32_t k = n / (cx + cy);
    uint32_t ans = g * k;
    uint32_t delta = ~0;
    n -= k * (cx + cy);
    for (uint32_t nx = 0; nx <= n; ++nx) {
        delta = std::min(delta, std::max(nx * x, (n - nx) * y));
    }
    return ans + delta;
}
int main() {
    uint32_t n, x, y;
```

```
std::cin >> n >> x >> y;  
std::cout << std::min(x,y) + solve(n-1,x,y) << std::endl;  
return 0;  
}
```

ЗАДАЧА №601

Цветной лабиринт

(Время: 1 сек. Память: 16 Мб Сложность: 38%)

В одном из парков одного большого города недавно был организован новый аттракцион Цветной лабиринт. Он состоит из n комнат, соединенных m двунаправленными коридорами. Каждый из коридоров покрашен в один из s цветов, при этом от каждой комнаты отходит не более одного коридора каждого цвета. При этом две комнаты могут быть соединены любым количеством коридоров.

Человек, купивший билет на аттракцион, оказывается в комнате номер один. Кроме билета, он также получает описание пути, по которому он может выбраться из лабиринта. Это описание представляет собой последовательность цветов $c_1 \dots c_k$. Пользоваться ей надо так: находясь в комнате, надо посмотреть на очередной цвет в этой последовательности, выбрать коридор такого цвета и пойти по нему. При этом если из комнаты нельзя пойти по коридору соответствующего цвета, то человеку приходится дальше самому выбирать, куда идти.

В последнее время в администрацию парка стали часто поступать жалобы от заблудившихся в лабиринте людей. В связи с этим, возникла необходимость написания программы, проверяющей корректность описания и пути, и, в случае ее корректности, сообщаемой номер комнаты, в которую ведет путь.

Описание пути некорректно, если на пути, который оно описывает, возникает ситуация, когда из комнаты нельзя пойти по коридору соответствующего цвета.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа n ($1 \leq n \leq 10000$) и m ($1 \leq m \leq 100000$) - соответственно количество комнат и коридоров в лабиринте. Следующие m строк содержат описания коридоров. Каждое описание содержит три числа u ($1 \leq u \leq n$), v ($1 \leq v \leq n$), c ($1 \leq c \leq 100$) - соответственно номера комнат, соединенных этим коридором, и цвет коридора. Следующая, $(m+2)$ -ая строка входного файла содержит длину описания пути - целое число k ($0 \leq k \leq 100000$). Последняя строка входного файла содержит k целых чисел, разделенных пробелами, - описание пути по лабиринту.

Выходные данные

В выходной файл OUTPUT.TXT выведите строку INCORRECT, если описание пути некорректно, иначе выведите номер комнаты, в которую ведет описанный путь. Помните, что путь начинается в комнате номер один.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 2 1 2 10 1 3 5 5 10 10 10 10 5	3

2	3 2 1 2 10 2 3 5 5 5 10 10 10 10	INCORRECT
3	3 2 1 2 10 1 3 5 4 10 10 10 5	INCORRECT

```

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>
#include <map>

int main() {
    std::map<int, int> next[10000]; // next[curr][color] -> next vertex
    int nVert, nEdges;
    scanf("%d %d", &nVert, &nEdges);
    while (nEdges--) {
        int a, b, color;
        scanf("%d %d %d", &a, &b, &color);
        --a, --b;
        next[a][color] = b;
        next[b][color] = a;
    }

    int nSteps;
    scanf("%d", &nSteps);
    int cur = 0;
    while (nSteps--) {
        int color;
        scanf("%d", &color);
        if (next[cur].find(color) == next[cur].end()) {
            printf("INCORRECT");
            return 0;
        }
        cur = next[cur][color];
    }
    printf("%d", cur+1);
    return 0;
}

```

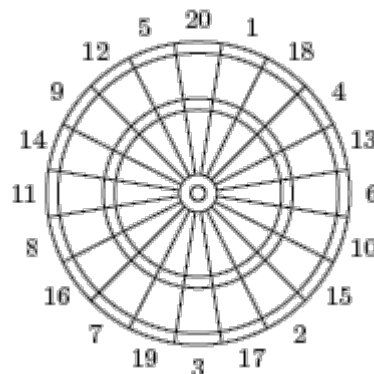
ЗАДАЧА №605

Дартс

(Время: 1 сек. Память: 16 Мб Сложность: 38%)

Игра в дартс очень популярна в Великобритании и Голландии. В игре принимают участие несколько игроков. Они по очереди бросают в мишень по три дротика.

В начале игры каждый игрок имеет некоторое количество очков, обычно 501. За каждое попадание дротика в мишень сумма игрока уменьшается на некоторое число, в зависимости от того, в какую часть мишени он попал. Первый, кто достигает нуля очков, считается победителем.



Внешний вид мишени показан на рисунке справа. Она разделена на 20 секторов, расположенных вокруг небольшого центрального круга. Этот круг, в свою очередь, делится на внутреннюю и внешнюю часть (иногда внутренняя часть называется «яблочко»). Попадание во внешнюю часть центрального круга оценивается 25 очков, а в «яблочко» - вдвое больше, то есть в 50 очков. Стоимость сектора равняется числу, которое на нем написано. Кроме того на мишени выделены два кольца - внешнее и внутреннее. Попадание в них оценивается соответственно в два и в три раза больше, чем в оставшуюся часть соответствующего сектора.

Существуют дополнительные правила для последней серии бросков, в которой игрок должен достичь нуля очков. В этой серии игроку придется бросить в мишень от одного до трех дротиков. Игрок должен достичь в точности нуля очков, получение отрицательной суммы считается ошибкой. Последний дротик должен быть «двойным», то есть попасть во внешнее кольцо какого-либо сектора или в «яблочко» - (оно считается удвоением внешней часть центрального круга).

Например, один из правильных способов закончить игру, имея 50 очков - бросить дротики в «18» и «D16».

Способы «D20», «10», или «20», «T10» не подходят: последний бросок не является удвоенным. Еще один возможный способ победить в этом случае - просто попасть в «яблочко» («Bull»). По количеству оставшихся очков, найдите все способы правильно закончить игру.

Входные данные

Входной файл INPUT.TXT содержит число n - количество оставшихся очков ($1 \leq n \leq 200$).

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите k - количество способов правильно завершить партию. Каждая из следующих k строк должна содержать описание одного правильного способа. При этом число от 1 до 20 отвечает попаданию в соответствующий сектор. Буква «D» перед числом обозначает попадание во внешнее (удваивающее) кольцо, а «T» - во внутреннее (утраивающее). Внешняя часть центрального круга обозначается как «25», а «яблочко» (bull eye) - словом «Bull».

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5	7 1 D1 D1 1 2 D1 1 D2 D1 1 D1 T1 D1 2 1 D1 3 D1

```

#include <iostream>
#include <functional>
#include <string>
#include <vector>

int count(const std::string& s) {
    if (s == "Bull") return 50;
    if (s[0] == 'D') {
        return 2 * count(s.substr(1));
    }
    if (s[0] == 'T') {
        return 3 * count(s.substr(1));
    }
    return std::stoi(s);
}

int main() {
    std::vector<std::string> steps{"25", "Bull"};
    for (int i = 1; i <= 20; ++i) {
        auto s = std::to_string(i);
        steps.push_back(s);
        steps.push_back("D"+s);
        steps.push_back("T"+s);
    }

    std::vector<std::vector<std::string>> answer;

    std::function<void(int,int,int,std::vector<std::string>&)> go =
    [&](int n, int cur, int need, std::vector<std::string>& temp) {
        if ((int)temp.size() >= n || cur >= need) {
            if (cur == need && (temp.back()[0] == 'D' || temp.back()[0] == 'B')) {
                answer.push_back(temp);
            }
            return;
        }
        for (auto& s : steps) {
            if (cur+count(s) <= need) {
                temp.push_back(s);
                go(n, cur+count(s), need, temp);
                temp.pop_back();
            }
        }
    };

    int cost; std::cin >> cost;
    std::vector<std::string> temp;
    go(3, 0, cost, temp);
    std::cout << answer.size() << "\n";
    for (auto& row : answer) {
        for (auto& it : row) {
            std::cout << it << ' ';
        }
        std::cout << '\n';
    }

    return 0;
}

```

ЗАДАЧА №906

Функция - 2

(Время: 1 сек. Память: 16 Мб Сложность: 38%)

Описана рекурсивная функция с тремя параметрами $F(a, b, c)$:

- если $a \leq 0$ или $b \leq 0$ или $c \leq 0$, то $F(a, b, c) = 1$
- если $a > 20$ или $b > 20$ или $c > 20$, то $F(a, b, c) = F(20, 20, 20)$
- если $a < b$ и $b < c$, то $F(a, b, c) = F(a, b, c-1) + F(a, b-1, c-1) - F(a, b-1, c)$
- иначе $F(a, b, c) = F(a-1, b, c) + F(a-1, b-1, c) + F(a-1, b, c-1) - F(a-1, b-1, c-1)$

Однако, если указанную функцию реализовать напрямую, то даже для небольших значений a, b и c (например, $a = 15, b = 15, c = 15$), программа будет работать несколько часов! Необходимо реализовать эффективный алгоритм вычисления функции F , который успеет найти любое ее значение менее чем за одну секунду!

Входные данные

Входной файл INPUT.TXT содержит три целых числа a, b, c - параметры функции F ($-10^4 \leq a, b, c \leq 10^4$).

Выходные данные

В выходной файл OUTPUT.TXT выведите значение функции $F(a, b, c)$.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 1 1	2
2	2 2 2	4
3	10 4 6	523
4	50 50 50	1048576

```
#include <stdio.h>
#include <map>

struct Key {
    int16_t a, b, c;
};

inline bool operator<(const Key& l, const Key& r) {
    return l.a < r.a || (l.a == r.a && (l.b < r.b || (l.b == r.b && l.c < r.c)));
}

std::map<Key, int> cache;

int f(int16_t a, int16_t b, int16_t c) {
    if (a <= 0 || b <= 0 || c <= 0) {
        return 1;
    }

    if (a > 20 || b > 20 || c > 20) {
```



```

        return f(20,20,20);
    }

    auto key = Key{a,b,c};

    auto it = cache.find(key);

    if (it != cache.end()) {
        return it->second;
    }

    if (a < b && b < c) {
        return cache[key] = f(a,b,c-1)+f(a,b-1,c-1)-f(a,b-1,c);
    }

    return cache[key] = f(a-1,b,c)+f(a-1,b-1,c)+f(a-1,b,c-1)-f(a-1,b-1,c-1);
}

int main() {
    int a, b, c;
    scanf("%d %d %d", &a, &b, &c);
    printf("%d", f(a,b,c));
    return 0;
}

```

ЗАДАЧА №1339

Последовательность Фибоначчи

(Время: 1 сек. Память: 16 Мб Сложность: 38%)

$\{F_k\}$ - бесконечная последовательность целых чисел, которая удовлетворяет условию Фибоначчи $F_k = F_{k-1} + F_{k-2}$ (для любого целого k).

Пример части последовательности:

Даны i, F_i, j, F_j, n ($i \neq j$). Требуется найти F_n .

Входные данные

Входной файл INPUT.TXT содержит пять целых чисел: i, F_i, j, F_j, n ($-1000 \leq i, j, n \leq 1000, -2 \cdot 10^9 \leq F_k \leq 2 \cdot 10^9$ ($k = \min(i, j, n) \dots \max(i, j, n)$)).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число F_n .

Пример

k	-2	-1	0	1	2	3	4	5	6
F _k	-5	4	-1	3	2	5	7	12	19
№	INPUT.TXT				OUTPUT.TXT				
1	3 5 -1 4 5				12				

```
/*
    Метод прогонки. При фиксированных концах находим первое число фибоначчи слева от правог
    От найденных двух соседних чисел Фибонначи линейно получаем нужное.
    Для надежности, чтобы избежать переполнения, решаем в вещественных числах.
*/

#include <iostream>
#include <iomanip>
#include <vector>
#include <cassert>

typedef long double Real;

std::vector<Real> solve_tridiag(int n, Real L, Real R) {
    // n - расстояние между индексами чисел Фибонначи, L и R - числа фибоначчи на концах
    if (n == 1) {
        return {L, R};
    }
    if (n == 2) {
        return {L, R-L, R};
    }
    /*
        Общий вид уравнений:
        x[i-1] + x[i] - x[i+1] = 0
    */
}
```

Для метода прогонки:

$$a[i] * x[i-1] + c[i] * x[i] + b[i] * x[i+1] = f[i]$$

При $i = 1$:

$$x[i] - x[i+1] = -L$$

При $i = n-1$:

$$x[i-1] + x[i] = R$$

*/

// Массивы коэффициентов системы:

```
std::vector<Real> a(n-1, 1), b(n-1, -1), c(n-1, 1), f(n-1, 0);
```

```
a.front() = b.back() = 0;
```

```
f.front() -= L;
```

```
f.back() += R;
```

// Прямой ход - вычисляем прогоночные коэффициенты:

```
std::vector<Real> alpha{0}, beta{0};
```

```
alpha.push_back(-b[0] / c[0]);
```

```
beta.push_back(f[0] / c[0]);
```

```
for (int i = 1; i < (int)c.size(); ++i) {  
    assert(i == (int)alpha.size()-1);  
    auto q = a[i] * alpha[i] + c[i];  
    alpha.push_back(-b[i] / q);  
    beta.push_back((f[i] - a[i] * beta[i]) / q);  
}
```

// Обратный ход - вычисление решения системы:

```
std::vector<Real> x(n-1);
```

```
x[n-2] = (f[n-2] - a[n-2] * beta[n-2]) / (c[n-2] + a[n-2] * alpha[n-2]);
```

```
for (int i = n-3; i >= 0; --i) {
```

```
    x[i] = alpha[i+1] * x[i+1] + beta[i+1];
```

```
}
```

```
x.insert(x.begin(), L);
```

```
x.insert(x.end(), R);
```

```
return x;
```

```
}
```

```
Real solve(int i_min, Real f_min, int i_max, Real f_max, int index) {
```

```
    auto f = solve_tridiag(i_max-i_min, f_min, f_max);
```

```
    if (index >= i_max) {
```

```
        Real prev = f[(int)f.size()-2];
```

```
        Real curr = f[(int)f.size()-1];
```

```
        for (int i = i_max; i < index; ++i) {
```

```
            Real next = prev + curr;
```

```
            prev = curr;
```

```
            curr = next;
```

```
        }
```

```
        return curr;
```

```
    } else {
```

```
        Real curr = f[(int)f.size()-1];
```

```
        Real next = curr + f[(int)f.size()-2];
```

```
        for (int i = i_max; i > index; --i) {
```

```
            Real prev = next-curr;
```

```
            next = curr;
```

```
            curr = prev;
```

```
        }
```

```
        return curr;
```

```
    }
```

```
}
```

```
int main() {
```

```
    int i1, i2, i3;
```

```
    Real f1, f2;
```

```
    std::cin >> i1 >> f1 >> i2 >> f2 >> i3;
```

```
    if (i2 < i1) {
```

```
        std::swap(i1, i2);
```

```
        std::swap(f1, f2);
```

```
}  
  
std::cout << std::fixed << std::setprecision(0) << solve(i1, f1, i2, f2, i3) << std::en  
return 0;  
}
```

ЗАДАЧА №1473

Раздел империи

(Время: 2 сек. Память: 16 Мб Сложность: 38%)

Короли прошлого создали великую империю, в которой было N городов, и соединили их M двусторонними дорогами таким образом, что между любыми двумя городами существует путь, возможно через другие города. Одну и ту же пару городов может соединять несколько дорог, также дороги могут выходить и входить в один и тот же город. Со временем K городов усилились и возвысились над остальными, между ними начались междоусобицы. И однажды империя развалилась на K государств, столицей в каждом из которых стал усилившийся город.

До наших дней не дошло ни одной карты, где показывается, какой город оказался в каком государстве после того, как империя канула в лету. Однако известно, что в образованных государствах, также как и в великой империи, существовал путь между любой парой городов, возможно через другие города того же государства. И каждый город великой империи вошел в одно из государств, появившихся на ее руинах.

Помогите историкам восстановить карту, которая показывает, какой город какому государству принадлежал. Если вариантов карты может быть несколько, то достаточно найти любой из них.

Входные данные

В первой строке входного файла INPUT.TXT записаны числа N ($1 \leq N \leq 1000$) и M ($N-1 \leq M \leq 10^5$) – количество городов и дорог соответственно. В следующих M строках идет описание дорог: каждая строка содержит два целых числа x_i и y_i ($1 \leq x_i, y_i \leq N$) – номера городов, соединенные дорогой.

В следующей строке записано целое число K ($1 \leq K \leq N$) – количество столиц. В последней строке перечислены K различных целых чисел s_i ($1 \leq s_i \leq N$) – номера столиц.

Выходные данные

В выходной файл OUTPUT.TXT для каждого из K усилившихся городов в первой строке требуется вывести количество городов государства, чьей столицей стал этот город; во второй строке – номера городов, вошедших в государство.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 2	2
	1 2	1 2
	2 3	1
	2	3
	1 3	

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>
#include <queue>
```

```

int main() {
    int nVert, nEdges;
    scanf("%d %d", &nVert, &nEdges);
    std::vector<std::vector<bool>> g(nVert, std::vector<bool>(nVert));
    while (nEdges--) {
        int a, b;
        scanf("%d %d", &a, &b);
        --a, --b;
        if (a != b) {
            g[a][b] = g[b][a] = true;
        }
    }
    std::vector<int> part(nVert, -1);
    int nParts;
    scanf("%d", &nParts);
    std::queue<int> queue;
    for (int i = 0; i < nParts; ++i) {
        int v;
        scanf("%d", &v);
        --v;
        part[v] = i;
        queue.push(v);
    }
    while (!queue.empty()) {
        auto curr = queue.front(); queue.pop();
        for (int next = 0; next < nVert; ++next) {
            if (g[curr][next] && part[next] == -1) {
                part[next] = part[curr];
                queue.push(next);
            }
        }
    }
    std::vector<std::vector<int>> answ(nParts);
    for (int v = 0; v < nVert; ++v) {
        assert(part[v] != -1);
        answ[part[v]].push_back(v+1);
    }
    for (const auto& part : answ) {
        assert(!part.empty());
        printf("%d\n", (int)part.size());
        for (const auto& it : part) {
            printf("%d ", it);
        }
        printf("\n");
    }
    return 0;
}

```

ЗАДАЧА №172

Деление с остатком

(Время: 1 сек. Память: 16 Мб Сложность: 39%)

Заданы два числа: N и K. Необходимо найти остаток от деления N на K.

Входные данные

Входной файл INPUT.TXT содержит два целых числа: N и K ($1 \leq N \leq 10^{100}$, $1 \leq K \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите остаток от деления N на K.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	239 16	15
2	4638746747645731289347483927 6784789	1001783

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <cstdint>
#include <cassert>
#include <functional>

struct Int;

std::istream& operator>>(std::istream&, Int&);
std::ostream& operator<<(std::ostream&, const Int&);
bool operator<(const Int&, const Int&);
bool operator>(const Int&, const Int&);
bool operator==(const Int&, const Int&);
bool operator<=(const Int&, const Int&);
bool operator>=(const Int&, const Int&);
bool operator!=(const Int&, const Int&);
Int operator+(const Int&, const Int&);
Int operator+(const int, const Int&);
Int operator-(const Int&, const Int&);
Int operator*(const int, const Int&);
Int operator/(const Int& a, const int);
int operator%(const Int& a, const int);
Int pow(Int, int);

struct Int {
    static const int BASE = (int)1e9; // Основание системы счисления
    static const int WIDTH = 9;      // Количество десятичных цифр, которые хранятся в одн

    // Вектор под цифры числа:
    std::vector<int> digits;

    // Нормализуем вид числа - удаляем лидирующие нули
    Int& to_normal() {
```

```

        while (digits.back() == 0 && (int)digits.size() > 1) {
            digits.pop_back();
        }
        return *this;
    }

// Конструктор от короткого целого
Int (int64_t number = 0) {
    assert(number >= 0);
    do {
        digits.push_back(number % BASE);
        number /= BASE;
    } while (number > 0);
    to_normal();
}

// Конструктор от вектора из цифр:
Int (const std::vector<int>& digits) : digits(digits) { to_normal(); }

// Конструктор от строки:
Int (std::string s) {
    const int size = (int)s.size();
    for (int idGroup = 1, nGroups = size / WIDTH; idGroup <= nGroups; ++idGroup) {
        digits.push_back(std::stoi(s.substr(size-idGroup * WIDTH, WIDTH)));
    }
    if (size % WIDTH != 0) {
        digits.push_back(std::stoi(s.substr(0, size % WIDTH)));
    }
    to_normal();
}

// Прибавление:
Int& operator+=(const int num) {
    if (num >= BASE) {
        return *this += Int(num);
    }
    int rem = num;
    for (int i = 0; rem > 0; ++i) {
        if (i >= size()) digits.push_back(0);
        rem += at(i);
        at(i) = rem % BASE;
        rem /= BASE;
    }
    return this->to_normal();
}

// Прибавление:
Int& operator+=(const Int& other) {
    if (other.size() == 1) {
        return *this += other.at(0);
    }
    const int s1 = this->size();
    const int s2 = other.size();
    int rem = 0;
    for (int i = 0; i < s1 || i < s2 || rem > 0; ++i) {
        int d1 = i < s1 ? this->at(i) : 0;
        int d2 = i < s2 ? other.at(i) : 0;
        rem += d1 + d2;
        if (i >= s1) digits.push_back(0);
        at(i) = rem % BASE;
        rem /= BASE;
    }
    return this->to_normal();
}

// Вычитание короткого:
Int& operator-=(const int num) {
    if (num >= BASE) {
        return *this -= Int(num);
    }

```



```

    }
    int rem = -num;
    for (int i = 0; rem < 0; ++i) {
        assert(i < size());
        rem += at(i);
        if (rem < 0) {
            at(i) = (rem + BASE) % BASE;
            rem = -1;
        } else {
            at(i) = rem % BASE;
            rem = 0;
        }
    }
    return this->to_normal();
}

// Вычитание длинного:
Int& operator-=(const Int& other) {
    if (other.size() == 1) {
        return *this -= other.at(0);
    }
    assert(*this >= other);
    const int s1 = this->size();
    const int s2 = other.size();
    int rem = 0;
    for (int i = 0; i < s1 || i < s2; ++i) {
        int d1 = i < s1 ? this->at(i) : 0;
        int d2 = i < s2 ? other.at(i) : 0;
        rem += d1 - d2;
        if (i >= s1) digits.push_back(0);
        if (rem < 0) {
            at(i) = (rem + BASE) % BASE;
            rem = -1;
        } else {
            at(i) = rem % BASE;
            rem = 0;
        }
    }
    return this->to_normal();
}

// Умножение на короткое:
Int& operator*=(const int num) {
    // std::cout << "call Int& operator*=(const int num)" << std::endl;
    if (num >= BASE) {
        return *this *= Int(num);
    }
    int64_t rem = 0;
    for (int i = 0; i < size() || rem > 0; ++i) {
        // std::cout << "i = " << i << std::endl;
        if (i >= size()) digits.push_back(0);
        rem += 1LL * at(i) * num;
        at(i) = rem % BASE;
        rem /= BASE;
    }
    return this->to_normal();
}

// Умножение:
Int operator*(const int num) const {
    return num >= BASE ? *this * Int(num) : Int(*this) *= num;
}

// Произведение:
Int operator*(const Int& other) const {
    // std::cout << "call Int operator*(const Int& other)" << std::endl;
    if (other.size() == 1) {
        return *this * other.at(0);
    }

```

```

const int s1 = this->size();
const int s2 = other.size();
std::vector<int> temp(s1+s2);
for (int i = 0; i < s1; ++i) {
    int64_t rem = 0;
    for (int j = 0; j < s2; ++j) {
        rem += temp.at(i+j) + 1LL * this->at(i) * other.at(j);
        temp.at(i+j) = rem % BASE;
        rem /= BASE;
    }
    if (rem > 0) {
        temp.at(i+s2) += rem;
        assert(0 <= temp.at(i+s2) && temp.at(i+s2) < BASE);
    }
}
return Int(temp);
}

// Умножение:
Int& operator*=(const Int& other) {
    // std::cout << "call Int& operator*=(const Int& other)" << std::endl;
    return other.size() == 1 ? *this *= other.at(0) : *this = *this * other;
}

// Деление на короткое:
Int& operator/=(const int num) {
    // std::cout << "call Int& operator/=(const int num)" << std::endl;
    if (num >= BASE) {
        return *this /= Int(num);
    }
    assert(0 < num && num < BASE);
    int64_t rem = 0;
    for (int j = size()-1; j >= 0; --j) {
        // std::cout << "j = " << j << std::endl;
        (rem *= BASE) += at(j);
        // std::cout << "at(j) = " << at(j) << std::endl;
        at(j) = rem / num;
        assert(0 <= at(j) && at(j) < BASE);
        rem %= num;
    }
    // std::cout << "gool end!" << std::endl;
    return this->to_normal();
}

// Взятие остатка от деления:
Int& operator%=(const int num) {
    return *this = *this % num;
}

static Int div(Int a, Int b) {
    // Данный метод работает за O(m*n)
    // Условия:
    // base^(m-1) <= a < base^m (1)
    // base^n / 2 <= b < base^n (2)
    // Условие (1) выполняется всегда для m = a.size()
    // Условие (2) необходимо обеспечить равносильным преобразованием следующим образом
    if (2 * b.digits.back() < Int::BASE) {
        int d = (1LL * Int::BASE + 2 * b.digits.back() - 1) / (2 * b.digits.back());
        a *= d;
        b *= d;
    }

    const int n = b.size(), m = a.size();

    std::function<Int(Int, Int)> special_div = [&special_div](const Int& a, const Int&
        // Данный метод работает за O(n), при следующих условиях:
        // 0 <= a <= base^(n+1)
        // b^n / 2 <= b < base^n
        const int n = b.size());

```

```

// Проверки на соответствии условиям метода:
assert(a.size() - b.size() == 1);
if (a >= b * BASE) {
    return Int(BASE) + special_div(a-b * BASE, b);
} else {
    int64_t q = (BASE * 1LL * a.at(n) + a.at(n-1)) / b.at(n-1);
    assert(0 <= q && q < BASE);
    auto t = q * b;
    if (t > a) { --q, t -= b; }
    if (t > a) { --q, t -= b; }
    assert(t <= a);
    return Int(q);
}
};

if (m < n) {
    return 0; // O(1)
} else if (m == n) {
    return a >= b ? 1 : 0; // O(n)
} else if (m == n+1) {
    return special_div(a, b); // O(n)
} else {
    Int a_temp = std::vector<int>(a.digits.begin()+m-n-1, a.digits.begin()+m); // O
    Int s = std::vector<int>(a.digits.begin(), a.digits.begin()+m-n-1); // O(n)
    Int q_temp = special_div(a_temp, b); // O(n)
    Int r_temp = a_temp - q_temp * b; // O(n)
    Int q = div(r_temp.shift_left(m-n-1)+s, b); // O(m-n) рекурсивных вызовов -> об
    return q_temp.shift_left(m-n-1) + q; // O(n)
}
}

Int operator/(const Int& other) const {
    // std::cout << "call operator/(const Int& other)" << std::endl;
    return (other.size() == 1) ? *this / other.at(0) : div(*this, other);
}

Int& operator/=(const Int& other) {
    // std::cout << "call Int& operator/=(const Int& other)" << std::endl;
    return *this = *this / other;
}

Int operator%(const Int& other) const {
    // std::cout << "call operator%(const Int& other)" << std::endl;
    return *this - *this / other * other;
}

Int& operator%=(const Int& other) {
    return *this = *this % other;
}

// Сравнение: result < 0 (меньше), result == 0 (равно), result > 0 (больше)
int compare(const Int& other) const {
    if (this->size() > other.size()) return 1;
    if (this->size() < other.size()) return -1;
    for (int i = size()-1; i >= 0; --i) {
        if (this->at(i) > other.at(i)) return 1;
        if (this->at(i) < other.at(i)) return -1;
    }
    return 0;
}

int& at(int pos) {
    assert(0 <= pos && pos < size());
    return digits.at(pos);
}

const int& at(int pos) const {
    assert(0 <= pos && pos < size());
    return digits.at(pos);
}

```

```

}

inline int size() const {
    return (int)digits.size();
}

Int& shift_left(int pow) {
    assert(pow >= 0);
    digits.resize((int)digits.size()+pow);
    for (int i = (int)digits.size()-1; i >= pow; --i) {
        digits[i] = digits[i-pow];
    }
    for (int i = pow-1; i >= 0; --i) {
        digits[i] = 0;
    }
    return to_normal();
}

Int& shift_right(int pow) {
    assert(pow >= 0);
    if (pow >= (int)digits.size()) {
        return *this = 0;
    }
    for (int i = 0; i + pow < (int)digits.size(); ++i) {
        digits[i] = digits[i+pow];
    }
    digits.resize((int)digits.size()-pow);
    return to_normal();
}

};

// Ввод из потока:
std::istream& operator>>(std::istream& is, Int& number) {
    std::string s;
    is >> s;
    number = Int(s);
    return is;
}

// Вывод в поток:
std::ostream& operator<<(std::ostream& os, const Int& number) {
    os << number.digits.back();
    for (int i = (int)number.digits.size()-2; i >= 0; --i) {
        os << std::setw(Int::WIDTH) << std::setfill('0') << number.digits[i];
    }
    return os << std::setfill(' ');
}

// Сложение:
Int operator+(const Int& a, const Int& b) {
    return Int(a) += b;
}

// Вычитание:
Int operator-(const Int& a, const Int& b) {
    return Int(a) -= b;
}

// Умножение:
Int operator*(const int a, const Int& b) {
    return b * a;
}

// Деление:
Int operator/(const Int& a, const int num) {
    // std::cout << "operator/(const Int& a, const int num)" << std::endl;
    return Int(a) /= num;
}

```

```

// Остаток от деления:
int operator%(const Int& a, const int num) {
    int64_t rem = 0;
    for (int i = a.size()-1; i >= 0; --i) {
        ((rem *= Int::BASE) += a.at(i)) %= num;
    }
    return rem;
}

// Возведение в степень:
Int pow(Int a, int n) {
    Int res = 1;
    while (n > 0) {
        if (n % 2 != 0) {
            res *= a;
        }
        a *= a;
        n /= 2;
    }
    return res;
}

// Операторы сравнения:
bool operator<(const Int& a, const Int& b) { return a.compare(b) < 0; }
bool operator>(const Int& a, const Int& b) { return a.compare(b) > 0; }
bool operator==(const Int& a, const Int& b) { return a.compare(b) == 0; }
bool operator<=(const Int& a, const Int& b) { return a.compare(b) <= 0; }
bool operator>=(const Int& a, const Int& b) { return a.compare(b) >= 0; }
bool operator!=(const Int& a, const Int& b) { return a.compare(b) != 0; }

int main() {
    Int n, k;
    std::cin >> n >> k;
    std::cout << n % k << std::endl;
    return 0;
}

```

ЗАДАЧА №502

Лягушонок

(Время: 1 сек. Память: 16 Мб Сложность: 39%)

Многие, вероятно, слышали песни про приключения лягушонка Crazy Frog. На этот раз неугомонное милое создание решило подкрепиться, но даже такое простое действие решило выполнить в виде игры. Итак, в каждой клетке квадратного игрового поля, разбитого на $N \times N$ ($N \leq 50$) клеток, находится один комар весом a_{ij} (вес комара – натуральное число ≤ 50), i - номер строки, j - номер столбца. Лягушонок, прыгая с клетки на клетку, ест комаров. Правила игры таковы - в каждом столбце можно съесть не более одного комара. Всякий раз при съедании комара запоминаем номер строки, откуда съеден комар, и сумма номеров строк, в которых были съедены комары, в конце игры должна быть в точности равна N . Учтите, если из какой-то строки съедено несколько комаров, то номер данной строки участвует в суммировании более одного раза.

Определите максимальный вес комаров, который можно съесть при следовании приведённым правилам.

Входные данные

Первая строка входного файла INPUT.TXT содержит число N . Следующие N строк содержат по N чисел a_{ij} , разделенных пробелами.

Выходные данные

В выходной файл OUTPUT.TXT выведите целое число – вес съеденных комаров.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 8 2 1 1 2 6 2 7 2	14
2	5 8 2 1 2 3 1 2 6 2 4 2 7 2 3 4 1 3 2 4 4 1 3 4 3 1	19

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```
int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0); std::cout.tie(0);
    int n; std::cin >> n;
    static int arr[51][51];
    for (int row = 1; row <= n; ++row) {
        for (int col = 1; col <= n; ++col) {
```

```

        std::cin >> arr[row][col];
    }
}
int max[51][52];
for (int i = 1; i < 51; ++i) {
    for(int j = 0; j < 52; ++j) {
        max[i][j] = -1;
    }
}
for (int sum = 0; sum < n; ++sum) {
    for (int col = 1; col <= n; ++col) {
        if (max[sum][col] == -1) continue;
        // Никого не едим:
        max[sum][col+1] = std::max(max[sum][col+1], max[sum][col]);
        // Съесть в строке row комапа:
        for (int row = 1; row <= n; ++row) {
            //std::cout << "max[" << sum << "]"[" << col << "]"=" << max[sum][col] << std
            if (sum+row <= n) {
                max[sum+row][col+1] = std::max(
                    max[sum+row][col+1],
                    max[sum][col] + arr[row][col]
                );
            }
        }
    }
}
}
int answ = 0;
for (int i = 1; i <= 51; ++i) {
    answ = std::max(answ, max[n][i]);
}
std::cout << answ << std::endl;
return 0;
}

```

ЗАДАЧА №588

Число

(Время: 1 сек. Память: 16 Мб Сложность: 39%)

Скажем, что число в b -ой системе счисления описывает себя, если оно - b -значное, его первая цифра равна числу единиц в нем, вторая цифра равна числу двоек, и т.д., $b-1$ -ая цифра равна числу цифр $b-1$, а последняя цифра равна числу нулей. Примером такого числа для $b = 5$ является 12002.

Задано число b . Требуется найти число в системе счисления по основанию b , которое описывает себя.

Входные данные

Входной файл INPUT.TXT содержит единственное целое число b ($2 \leq b \leq 1000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите любое описывающее себя число в системе счисления с основанием b , или -1, если такого числа не существует. Каждую цифру числа выводите на отдельной строке в десятичной записи.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	5	1 2 0 0 2
2	2	-1

```
#include <stdio.h>
#include <iostream>
#include <vector>

void rec(int base, std::vector<int>& digits, std::vector<int>& count, std::vector<int>& ans)
{
    if (!ans.empty()) return;
    if ((int)digits.size() == base) {
        if (digits.back() != count[0]) {
            return;
        }
        for (int i = 0; i < (int)digits.size()-1; ++i) {
            if (digits[i] != count[i+1]) {
                return;
            }
        }
        ans = digits;
        return;
    }
    for (int digit = (int)digits.empty(); digit < base; ++digit) {
        if (ans.empty()) {
            digits.push_back(digit);
            count[digit]++;
            rec(base, digits, count, ans);
        }
    }
}
```



```

        count[digit]--;
        digits.pop_back();
    }
}

void precalc(int s, int t) {
    for (int base = s; base <= t; ++base) {
        std::vector<int> count(base), digits, answ;
        rec(base, digits, count, answ);
        std::cout << "base = " << base << ":";
        if (answ.empty()) answ.push_back(-1);
        for (auto& it : answ) {
            std::cout << ' ' << it;
        }
        std::cout << std::endl;
    }
    std::exit(0);
}

std::vector<int> solve(int base) {
    std::vector<int> answ;
    if (base == 2 || base == 3 || base == 6) {
        answ.push_back(-1);
        return answ;
    }
    if (base == 4) {
        return {2,1,0,1};
    }
    if (base == 5) {
        return {1,2,0,0,2};
    }
    answ.assign(base, 0);
    answ.back() = base-4;
    answ[base-5]++;
    answ[0] = 2;
    answ[1] = 1;
    return answ;
}

int main() {
    //precalc(2, 10);
    int base; scanf("%d", &base);
    for (auto it : solve(base)) {
        printf("%d\n", it);
    }
    return 0;
}

```

ЗАДАЧА №732

Треугольная комната

(Время: 0,2 сек. Память: 16 Мб Сложность: 39%)

Во многих книгах по занимательной математике приводится такая задача. Расставить по периметру треугольной комнаты 3 стула так, чтобы у каждой стены стояло по 2. Ее решение - поставить по стулу в каждый из углов комнаты.

Эта задача легко обобщается. Пусть комната представляет собой треугольник ABC. Даны: общее количество стульев n , количество стульев n_{AB} , которое должно стоять у стены AB, количество стульев n_{BC} , которое должно стоять у стены BC, количество стульев n_{AC} , которое должно стоять у стены AC. Необходимо найти соответствующую расстановку стульев или установить, что ее не существует. При этом стулья можно ставить только в углы комнаты и вдоль стен, в центр комнаты стулья ставить нельзя. В любой из углов можно поставить произвольное количество стульев.

Входные данные

Входной файл INPUT.TXT содержит целые числа $n, n_{AB}, n_{BC}, n_{AC}$ ($0 \leq n, n_{AB}, n_{BC}, n_{AC} \leq 1000$).

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите NO, если стулья указанным способом расставить невозможно. В противном случае выведите YES в первой строке выходного файла, а во второй выведите 6 целых неотрицательных чисел: $k_A, k_{AB}, k_B, k_{BC}, k_C, k_{AC}$ - соответственно количество стульев, которые необходимо поставить в угол A, вдоль стены AB, в угол B, вдоль стены BC, в угол C и вдоль стены AC.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 2 2 2	YES 1 0 1 0 1 0
2	3 3 2 2	NO

```
#include <stdio.h>
```

```
int main() {
    int n, n_ab, n_bc, n_ac;
    scanf("%d %d %d %d", &n, &n_ab, &n_bc, &n_ac);

    for (int k_a = 0; k_a <= n; ++k_a) {
        for (int k_b = 0; k_b <= n - k_a; ++k_b) {
            int k_ab = n_ab - k_a - k_b;
            int k_c = k_ab + n_ac + n_bc - n;
            int k_ac = n_ac - k_a - k_c;
            int k_bc = n_bc - k_b - k_c;
            if (k_a >= 0 && k_ab >= 0 && k_b >= 0 && k_bc >= 0 && k_c >= 0 && k_ac >= 0) {
                printf("YES\n");
                printf("%d %d %d %d %d %d", k_a, k_ab, k_b, k_bc, k_c, k_ac);
                return 0;
            }
        }
    }
}
```

```
    }  
    printf("NO");  
    return 0;  
}
```

ЗАДАЧА №913

Автобус

(Время: 1 сек. Память: 16 Мб Сложность: 39%)

Василий работает водителем автобуса, и у него есть по-настоящему тяжелая работа: он перевозит пассажиров из пункта А в пункт В. При этом он должен выполнять свою работу безопасно и как можно быстрее. Кроме того, каждый водитель должен повиноваться дорожным правилам и нормам, иначе, он будет оштрафован местной полицией.

Маршрут Василия состоит из N отрезков, которые он проезжает последовательно, начиная от 1-го и завершая N -м отрезком. У каждого отрезка задана длина в километрах D_i .

Конечно, на маршруте Василия есть ограничения скорости. У каждого отрезка есть свое собственное ограничение скорости L_i км/ч. В случае превышения скорости на i -м отрезке, водитель автобуса будет немедленно остановлен полицейским, который потратит H_i часов на заполнение соответствующих бумаг. Водитель не может быть остановлен более одного раза на каждом отрезке.

Автобус едет вдоль маршрута с постоянной скоростью S , которую Василий выбирает в начале маршрута (в пункте А). Автобус не может превышать максимальную конструктивную скорость, которая определена значением M км/ч.

Пожалуйста, помогите Василию выбирать правильную скорость, чтобы достигнуть конечного пункта В как можно быстрее.

Входные данные

Первая строка входного файла INPUT.TXT содержит два числа, разделенные пробелом: количество отрезков N ($N \leq 1000$) в маршруте Василия и максимально возможную конструктивную скорость автобуса M ($M \leq 10^5$). Далее идут N строк, в каждой из которых определены разделенные пробелом значения D_i , L_i и H_i , описывающие i -й отрезок пути ($D_i, H_i \leq 1000$, $L_i \leq 10^5$). Все числа, определенные во входных данных натуральные.

Выходные данные

В выходной файл OUTPUT.TXT выведите оптимальную скорость автобуса. В случае неоднозначного ответа следует выбрать наибольшее значение.

Пример

Пояснение к примеру

Здесь у нас простой случай, когда максимальная скорость 100 км/ч является оптимальной для единственного отрезка пути. Таким образом, Василий может выбрать эту скорость и закончить свой маршрут через 10 часов. Поэтому ответ: 100.

№	INPUT.TXT	OUTPUT.TXT
1	1 100 1000 100 10	100

```

#include <stdio.h>
#include <vector>
#include <cassert>
#include <algorithm>
#include <functional>

struct Segment {
    int d, l, h;

    static Segment read() {
        int d, l, h;
        scanf("%d %d %d", &d, &l, &h);
        return Segment{d, l, h};
    }
};

int main() {
    int n, limit;
    scanf("%d %d", &n, &limit);

    std::vector<int> possible_speed{limit};

    std::vector<Segment> s;
    for (int i = 0; i < n; ++i) {
        s.push_back(Segment::read());
        if (s.back().l <= limit) {
            possible_speed.push_back(s.back().l);
        }
        if (s.back().l+1 <= limit) {
            possible_speed.push_back(s.back().l+1);
        }
    }

    std::sort(possible_speed.begin(), possible_speed.end(), std::greater<int>());

    double best = 1e9; int answ = -1;
    for (int speed : possible_speed) {
        double sum = 0;
        for (auto& it : s) {
            if (speed <= it.l) {
                sum += 1.0 * it.d / speed;
            } else {
                sum += 1.0 * it.d / speed + it.h;
            }
        }
        if (sum < best) {
            answ = speed;
            best = sum;
        }
    }
    assert(answ != -1);
    printf("%d", answ);
    return 0;
}

```

ЗАДАЧА №1327

Провода

(Время: 1 сек. Память: 16 Мб Сложность: 39%)

Дано N отрезков провода длиной L_1, L_2, \dots, L_N сантиметров.

Требуется с помощью разрезания получить из них K равных отрезков как можно большей длины, выражающейся целым числом сантиметров.

Входные данные

В первой строке входного файла INPUT.TXT находятся числа N и K ($1 \leq N \leq 10\,000$, $1 \leq K \leq 10\,000$, $100 \leq L_i \leq 10\,000\,000$). В следующих N строках – L_1, L_2, \dots, L_N , по одному числу в строке.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – полученную длину отрезков. Если нельзя получить K отрезков длиной даже 1 см, следует вывести 0.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4 11 802 743 457 539	200

```
#include <stdio.h>
#include <vector>
#include <algorithm>

int main() {
    int n, k;
    scanf("%d %d", &n, &k);

    std::vector<int> a(n);
    for (auto& it : a) {
        scanf("%d", &it);
    }

    int64_t sum = std::accumulate(a.begin(), a.end(), 0LL);
    if (sum < k) {
        printf("0\n");
        return 0;
    }

    int64_t low = 1, high = sum;
    while (high - low > 1) {
        int64_t mid = (low + high) / 2;
        int c = 0;
        for (auto it : a) {
            c += it / mid;
        }
    }
```

```
        if (c >= k) {
            low = mid;
        } else {
            high = mid;
        }
    }
    printf("%d\n", low);

    return 0;
}
```

ЗАДАЧА №145

A div B

(Время: 1 сек. Память: 16 Мб Сложность: 40%)

Даны два целых числа A и B. Требуется найти их целую часть от их частного.

Входные данные

Во входном файле INPUT.TXT записаны целые числа A и B по одному в строке ($0 \leq A \leq 10^{100}$, $0 < B \leq 10000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное число без лидирующих нулей: A div B.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	7 3	2

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <cstdint>
#include <cassert>
#include <functional>

struct Int;

std::istream& operator>>(std::istream&, Int&);
std::ostream& operator<<(std::ostream&, const Int&);
bool operator<(const Int&, const Int&);
bool operator>(const Int&, const Int&);
bool operator==(const Int&, const Int&);
bool operator!=(const Int&, const Int&);
bool operator<=(const Int&, const Int&);
bool operator>=(const Int&, const Int&);
Int operator+(const Int&, const Int&);
Int operator-(const Int&, const Int&);
Int operator*(const Int&, const Int&);
Int operator/(const Int& a, const Int& b);
int operator%(const Int& a, const Int& b);
Int pow(Int, int);

struct Int {
    static const int BASE = (int)1e9; // Основание системы счисления
    static const int WIDTH = 9;      // Количество десятичных цифр, которые хранятся в одн

    // Вектор под цифры числа:
    std::vector<int> digits;

    // Нормализуем вид числа - удаляем лидирующие нули
```



```

Int& to_normal() {
    while (digits.back() == 0 && (int)digits.size() > 1) {
        digits.pop_back();
    }
    return *this;
}

// Конструктор от короткого целого
Int (int64_t number = 0) {
    assert(number >= 0);
    do {
        digits.push_back(number % BASE);
        number /= BASE;
    } while (number > 0);
    to_normal();
}

// Конструктор от вектора из цифр:
Int (const std::vector<int>& digits) : digits(digits) { to_normal(); }

// Конструктор от строки:
Int (std::string s) {
    const int size = (int)s.size();
    for (int idGroup = 1, nGroups = size / WIDTH; idGroup <= nGroups; ++idGroup) {
        digits.push_back(std::stoi(s.substr(size-idGroup * WIDTH, WIDTH)));
    }
    if (size % WIDTH != 0) {
        digits.push_back(std::stoi(s.substr(0, size % WIDTH)));
    }
    to_normal();
}

// Прибавление:
Int& operator+=(const int num) {
    if (num >= BASE) {
        return *this += Int(num);
    }
    int rem = num;
    for (int i = 0; rem > 0; ++i) {
        if (i >= size()) digits.push_back(0);
        rem += at(i);
        at(i) = rem % BASE;
        rem /= BASE;
    }
    return this->to_normal();
}

// Прибавление:
Int& operator+=(const Int& other) {
    if (other.size() == 1) {
        return *this += other.at(0);
    }
    const int s1 = this->size();
    const int s2 = other.size();
    int rem = 0;
    for (int i = 0; i < s1 || i < s2 || rem > 0; ++i) {
        int d1 = i < s1 ? this->at(i) : 0;
        int d2 = i < s2 ? other.at(i) : 0;
        rem += d1 + d2;
        if (i >= s1) digits.push_back(0);
        at(i) = rem % BASE;
        rem /= BASE;
    }
    return this->to_normal();
}

// Вычитание короткого:
Int& operator-=(const int num) {
    if (num >= BASE) {

```

```

        return *this -= Int(num);
    }
    int rem = -num;
    for (int i = 0; rem < 0; ++i) {
        assert(i < size());
        rem += at(i);
        if (rem < 0) {
            at(i) = (rem + BASE) % BASE;
            rem = -1;
        } else {
            at(i) = rem % BASE;
            rem = 0;
        }
    }
    return this->to_normal();
}

// Вычитание длинного:
Int& operator-=(const Int& other) {
    if (other.size() == 1) {
        return *this -= other.at(0);
    }
    assert(*this >= other);
    const int s1 = this->size();
    const int s2 = other.size();
    int rem = 0;
    for (int i = 0; i < s1 || i < s2; ++i) {
        int d1 = i < s1 ? this->at(i) : 0;
        int d2 = i < s2 ? other.at(i) : 0;
        rem += d1 - d2;
        if (i >= s1) digits.push_back(0);
        if (rem < 0) {
            at(i) = (rem + BASE) % BASE;
            rem = -1;
        } else {
            at(i) = rem % BASE;
            rem = 0;
        }
    }
    return this->to_normal();
}

// Умножение на короткое:
Int& operator*=(const int num) {
    // std::cout << "call Int& operator*=(const int num)" << std::endl;
    if (num >= BASE) {
        return *this *= Int(num);
    }
    int64_t rem = 0;
    for (int i = 0; i < size() || rem > 0; ++i) {
        // std::cout << "i = " << i << std::endl;
        if (i >= size()) digits.push_back(0);
        rem += 1LL * at(i) * num;
        at(i) = rem % BASE;
        rem /= BASE;
    }
    return this->to_normal();
}

// Умножение:
Int operator*(const int num) const {
    return num >= BASE ? *this * Int(num) : Int(*this) *= num;
}

// Произведение:
Int operator*(const Int& other) const {
    // std::cout << "call Int operator*(const Int& other)" << std::endl;
    if (other.size() == 1) {
        return *this * other.at(0);
    }

```

```

    }
    const int s1 = this->size();
    const int s2 = other.size();
    std::vector<int> temp(s1+s2);
    for (int i = 0; i < s1; ++i) {
        int64_t rem = 0;
        for (int j = 0; j < s2; ++j) {
            rem += temp.at(i+j) + 1LL * this->at(i) * other.at(j);
            temp.at(i+j) = rem % BASE;
            rem /= BASE;
        }
        if (rem > 0) {
            temp.at(i+s2) += rem;
            assert(0 <= temp.at(i+s2) && temp.at(i+s2) < BASE);
        }
    }
    return Int(temp);
}

// Умножение:
Int& operator*=(const Int& other) {
    // std::cout << "call Int& operator*=(const Int& other)" << std::endl;
    return other.size() == 1 ? *this *= other.at(0) : *this = *this * other;
}

// Деление на короткое:
Int& operator/=(const int num) {
    // std::cout << "call Int& operator/=(const int num)" << std::endl;
    if (num >= BASE) {
        return *this /= Int(num);
    }
    assert(0 < num && num < BASE);
    int64_t rem = 0;
    for (int j = size()-1; j >= 0; --j) {
        // std::cout << "j = " << j << std::endl;
        (rem *= BASE) += at(j);
        // std::cout << "at(j) = " << at(j) << std::endl;
        at(j) = rem / num;
        assert(0 <= at(j) && at(j) < BASE);
        rem %= num;
    }
    // std::cout << "good end!" << std::endl;
    return this->to_normal();
}

// Взятие остатка от деления:
Int& operator%=(const int num) {
    return *this = *this % num;
}

static Int div(Int a, Int b) {
    // Данный метод работает за O(m*n)
    // Условия:
    // base^(m-1) <= a < base^m (1)
    // base^n / 2 <= b < base^n (2)
    // Условие (1) выполняется всегда для m = a.size()
    // Условие (2) необходимо обеспечить равносильным преобразованием следующим образом
    if (2 * b.digits.back() < Int::BASE) {
        int d = (1LL * Int::BASE + 2 * b.digits.back() - 1) / (2 * b.digits.back());
        a *= d;
        b *= d;
    }

    const int n = b.size(), m = a.size();

    std::function<Int(Int, Int)> special_div = [&special_div](const Int& a, const Int&
        // Данный метод работает за O(n), при следующих условиях:
        // 0 <= a <= base^(n+1)
        // b^n / 2 <= b < base^n

```

```

const int n = b.size();
// Проверки на соответствии условиям метода:
assert(a.size() - b.size() == 1);
if (a >= b * BASE) {
    return Int(BASE) + special_div(a-b * BASE, b);
} else {
    int64_t q = (BASE * 1LL * a.at(n) + a.at(n-1)) / b.at(n-1);
    assert(0 <= q && q < BASE);
    auto t = q * b;
    if (t > a) { --q, t -= b; }
    if (t > a) { --q, t -= b; }
    assert(t <= a);
    return Int(q);
}
};

if (m < n) {
    return 0; // O(1)
} else if (m == n) {
    return a >= b ? 1 : 0; // O(n)
} else if (m == n+1) {
    return special_div(a, b); // O(n)
} else {
    Int a_temp = std::vector<int>(a.digits.begin()+m-n-1, a.digits.begin()+m); // O
    Int s = std::vector<int>(a.digits.begin(), a.digits.begin()+m-n-1); // O(n)
    Int q_temp = special_div(a_temp, b); // O(n)
    Int r_temp = a_temp - q_temp * b; // O(n)
    Int q = div(r_temp.shift_left(m-n-1)+s, b); // O(m-n) рекурсивных вызовов -> об
    return q_temp.shift_left(m-n-1) + q; // O(n)
}
}

Int operator/(const Int& other) const {
    // std::cout << "call operator/(const Int& other)" << std::endl;
    return (other.size() == 1) ? *this / other.at(0) : div(*this, other);
}

Int& operator/=(const Int& other) {
    // std::cout << "call Int& operator/=(const Int& other)" << std::endl;
    return *this = *this / other;
}

Int operator%(const Int& other) const {
    // std::cout << "call operator%(const Int& other)" << std::endl;
    return *this - *this / other * other;
}

Int& operator%=(const Int& other) {
    return *this = *this % other;
}

// Сравнение: result < 0 (меньше), result == 0 (равно), result > 0 (больше)
int compare(const Int& other) const {
    if (this->size() > other.size()) return 1;
    if (this->size() < other.size()) return -1;
    for (int i = size()-1; i >= 0; --i) {
        if (this->at(i) > other.at(i)) return 1;
        if (this->at(i) < other.at(i)) return -1;
    }
    return 0;
}

int& at(int pos) {
    assert(0 <= pos && pos < size());
    return digits.at(pos);
}

const int& at(int pos) const {
    assert(0 <= pos && pos < size());

```

```

        return digits.at(pos);
    }

    inline int size() const {
        return (int)digits.size();
    }

    Int& shift_left(int pow) {
        assert(pow >= 0);
        digits.resize((int)digits.size()+pow);
        for (int i = (int)digits.size()-1; i >= pow; --i) {
            digits[i] = digits[i-pow];
        }
        for (int i = pow-1; i >= 0; --i) {
            digits[i] = 0;
        }
        return to_normal();
    }

    Int& shift_right(int pow) {
        assert(pow >= 0);
        if (pow >= (int)digits.size()) {
            return *this = 0;
        }
        for (int i = 0; i + pow < (int)digits.size(); ++i) {
            digits[i] = digits[i+pow];
        }
        digits.resize((int)digits.size()-pow);
        return to_normal();
    }
};

// Ввод из потока:
std::istream& operator>>(std::istream& is, Int& number) {
    std::string s;
    is >> s;
    number = Int(s);
    return is;
}

// Вывод в поток:
std::ostream& operator<<(std::ostream& os, const Int& number) {
    os << number.digits.back();
    for (int i = (int)number.digits.size()-2; i >= 0; --i) {
        os << std::setw(Int::WIDTH) << std::setfill('0') << number.digits[i];
    }
    return os << std::setfill(' ');
}

// Сложение:
Int operator+(const Int& a, const Int& b) {
    return Int(a) += b;
}

// Вычитание:
Int operator-(const Int& a, const Int& b) {
    return Int(a) -= b;
}

// Умножение:
Int operator*(const int a, const Int& b) {
    return b * a;
}

// Деление:
Int operator/(const Int& a, const int num) {
    // std::cout << "operator/(const Int& a, const int num)" << std::endl;
    return Int(a) /= num;
}

```

```

// Остаток от деления:
int operator%(const Int& a, const int num) {
    int64_t rem = 0;
    for (int i = a.size()-1; i >= 0; --i) {
        ((rem *= Int::BASE) += a.at(i)) %= num;
    }
    return rem;
}

// Возведение в степень:
Int pow(Int a, int n) {
    Int res = 1;
    while (n > 0) {
        if (n % 2 != 0) {
            res *= a;
        }
        a *= a;
        n /= 2;
    }
    return res;
}

// Операторы сравнения:
bool operator<(const Int& a, const Int& b) { return a.compare(b) < 0; }
bool operator>(const Int& a, const Int& b) { return a.compare(b) > 0; }
bool operator==(const Int& a, const Int& b) { return a.compare(b) == 0; }
bool operator<=(const Int& a, const Int& b) { return a.compare(b) <= 0; }
bool operator>=(const Int& a, const Int& b) { return a.compare(b) >= 0; }
bool operator!=(const Int& a, const Int& b) { return a.compare(b) != 0; }

int main() {
    Int a;
    int b;
    std::cin >> a >> b;
    std::cout << a / b << std::endl;
    return 0;
}

```

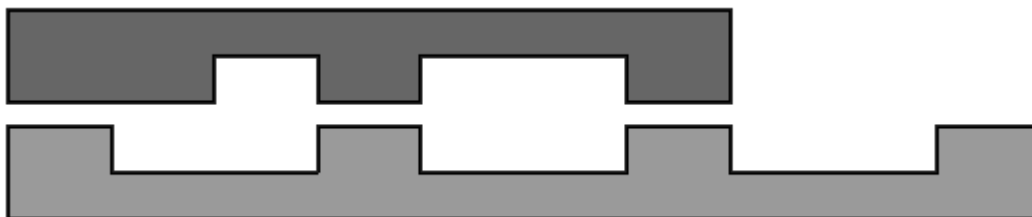
ЗАДАЧА №269

Тормозной механизм

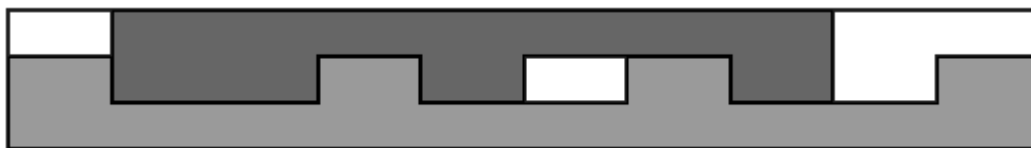
(Время: 1 сек. Память: 16 Мб Сложность: 40%)

Исследовательская лаборатория одной известной автомобильной компании разработала специальный механизм, позволяющий повысить эффективность тормозов путем равномерной нагрузки деталей, используемых в тормозах.

Одним из основных компонентов механизма являются 2 прокладки, которые в процессе взаимодействия накладываются друг на друга. Каждая прокладка длины n разделена на n разделов, каждый из которых имеет высоту h или $2h$. Таким образом, прокладки имеют зубчатую форму без закруглений.



В процессе взаимодействия прокладок важно, чтобы они накладывались друг на друга и при этом общая длина получившегося соединения была наименьшей.



По заданной конфигурации прокладок требуется определить наименьшую длину их возможного соединения, при котором общая высота конструкции не превышает значения $3h$. При этом вращать прокладки и удалять зубцы запрещено.

Входные данные

Входной файл INPUT.TXT содержит 2 строки с описанием конфигурации 2х прокладок. Каждая конфигурация определяется последовательностью цифр 1 и 2, соответствующих высоте каждого зубца прокладки. Каждая из строк не пуста и имеет длину, не превышающую 100.

Выходные данные

В выходной файл OUTPUT.TXT требуется вывести наименьшую длину конструкции из заданных прокладок.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2112112112 2212112	10

2	12121212 21212121	8
3	2211221122 21212	15

```

#include <stdio.h>
#include <string>
#include <algorithm>
#include <cassert>

int main() {
    char buf[101];
    scanf("%100s", buf);
    std::string s1(buf);
    scanf("%100s", buf);
    std::string s2(buf);
    if (s1.size() > s2.size()) {
        s1.swap(s2);
    }
    const int n1 = (int)s1.size();
    const int n2 = (int)s2.size();
    s2 = std::string(n1, '0') + s2 + std::string(n1, '0');
    s1 = s1 + std::string(n2+n1, '0');
    int answ = n1+n2;
    for (int i = 0; i < n1+n2; ++i) {
        std::rotate(s1.rbegin(), s1.rbegin()+1, s1.rend());
        assert(s1.front() == '0' || s1.back() == '0');
        int len = 0;
        for (int i = 0; i < (int)s1.size(); ++i) {
            if ((s1[i]-'0')+(s2[i]-'0') > 3) {
                len = n1+n2+1;
                break;
            }
            if (s1[i] > '0' || s2[i] > '0') {
                len++;
            }
        }
        answ = std::min(answ, len);
    }
    printf("%d", answ);
    return 0;
}

```


ЗАДАЧА №510

Шоколадка

(Время: 1 сек. Память: 16 Мб Сложность: 40%)

Саша, не сделал домашнюю работу, зато купил шоколадку. И, по глупости, начал распечатывать ее прямо на уроке... Шелест золотинки услышала учительница. Она хотела вызвать в школу родителей, но Саша уговорил ее не вызывать их, а дать дополнительное задание.

Учительница внимательно посмотрела на шоколадку (она была размером 3х4 плиток), разделила на кусочки по две плитки и угостила всех, кто сделал домашнюю работу. А Сашу попросила написать программу, которая определяет, сколько существует способов деления шоколадки размером 3×N плиток на кусочки по две плитки.

Для выполнения задания Саше нужна помощь.

Входные данные

Входной файл INPUT.TXT содержит натуральное число N – размер плитки, ($N < 33$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число – количество возможных способов.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2	3
2	4	11

```
#include <iostream>
#include <vector>

/*
    Варианты:
    1. Все вертикально
    ..11
    ..22
    ..33

    2. Одну вертикально - две другие горизонтально x2
    ..11
    ..23
    ..23

    ..23
    ..23
    ..11
    3. Когда заполняется не полностью:
    6441
    6331
    5522
```

```
*/
```

```
int main() {
    int n; std::cin >> n;
    std::vector<int> count(1+n);
    count[0] = 1;
    if (n >= 2) {
        count[2] = 3;
    }
    for (int curr = 3; curr <= n; ++curr) {
        count[curr] = 3 * count[curr-2];
        for (int prev = curr-4; prev >= 0; prev -= 2) {
            count[curr] += 2 * count[prev];
        }
    }
    std::cout << count[n] << std::endl;
    return 0;
}
```

ЗАДАЧА №527

Алгоритм Евклида

(Время: 1 сек. Память: 16 Мб Сложность: 40%)

Дима недавно начал изучать информатику. Одним из первых алгоритмов, который он изучил, был алгоритм Евклида для нахождения наибольшего общего делителя (НОД) двух чисел. Напомним, что наибольшим общим делителем двух чисел a и b называется наибольшее натуральное число x , такое, что и число a , и число b делится на него без остатка.

Алгоритм Евклида заключается в следующем:

1. Пусть a, b – числа, НОД которых надо найти.
2. Если $b = 0$, то число a – искомый НОД.
3. Если $b > a$, то необходимо поменять местами числа a и b .
4. Присвоить числу a значение $a - b$.
5. Вернуться к шагу 2.

Дима достаточно быстро освоил алгоритм Евклида и вычислил с его помощью много наибольших общих делителей. Поняв, что надо дальше совершенствоваться, ему пришла идея решить новую задачу. Пусть заданы числа a, b, c и d . Требуется узнать, наступит ли в процессе реализации алгоритма Евклида для заданной пары чисел (a, b) такой момент, когда число a будет равно c , а число b будет равно d .

Требуется написать программу, которая решает эту задачу.

Входные данные

Первая строка входного файла INPUT.TXT содержит количество наборов входных данных k ($1 \leq k \leq 100$). Далее идут описания этих наборов. Каждое описание состоит из двух строк. Первая из них содержит два целых числа: a, b ($1 \leq a, b \leq 10^{18}$). Вторая строка – два целых числа: c, d ($1 \leq c, d \leq 10^{18}$).

Выходные данные

Для каждого набора входных данных выведите в отдельной строке выходного файла OUTPUT.TXT слово «YES», если в процессе применения алгоритма Евклида к паре чисел (a, b) в какой-то момент получается пара (c, d) , или слово «NO» – в противном случае.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	2 20 10 10 10 10 7 2 4	YES NO

```
#include <iostream>
#include <cassert>
```

```

typedef long long ll;

bool solve(ll a, ll b, ll c, ll d) {
    if (a == c && b == d) return true;
    while (b != 0) {
        if (b > a) {
            std::swap(b, a);
            continue;
        }
        ll rem = a % b;
        if (b == d && c <= a && c >= rem && (a-c) % b == 0) {
            return true;
        }
        a = rem;
    }
    return a == c && b == d;
}

int main() {
    int nQ; std::cin >> nQ;
    while (nQ--) {
        ll a, b, c, d;
        std::cin >> a >> b >> c >> d;
        std::cout << (solve(a, b, c, d) ? "YES\n" : "NO\n");
    }
    return 0;
}

```

ЗАДАЧА №704

Деление

(Время: 1 сек. Память: 16 Мб Сложность: 40%)

На квадратном торте размером $N \times N$ расставлено M свечей. Определить, можно ли одним прямолинейным разрезом разделить торт на две части, равные по площади, так, чтобы все свечи оказались на одной половине. Свечи считаем точками. Разрез не может проходить через свечу.

Входные данные

Первая строка входного файла INPUT.TXT содержит число N ($1 < N \leq 100$) – длину стороны квадрата. Вторая строка теста содержит число M ($0 < M \leq 100$) – количество свечей на торте. Третья строка – координаты свечей, разделенные пробелами: $X_1 Y_1 X_2 Y_2 \dots X_m Y_m$ ($0 < X_i, Y_i < N$), заданные в системе координат с началом в одном из углов квадрата и осями – сторонами квадрата. Все исходные данные – целые положительные числа. Координаты всех свечей различны.

Выходные данные

В выходной файл OUTPUT.TXT выведите YES, если такое разделение возможно, или NO в противном случае.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	6 4 1 1 2 2 1 2 2 1	YES
2	20 10 1 1 2 2 1 2 2 1 10 10 3 4 7 2 3 8 2 11 11 3	NO

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cmath>
#include <cassert>

struct Point {
    int x, y;

    inline int norm2() const {
        return x * x + y * y;
    }
};

inline int cross(Point a, Point b) {
    return a.x * b.y - a.y * b.x;
}

bool solve(int side, std::vector<Point> p) {
```

```

for (auto& it : p) {
    it.x = 2 * it.x - side;
    it.y = 2 * it.y - side;
    if (it.x == 0 && it.y == 0) {
        return false;
    }
    if (it.x > 0 || (it.x == 0 && it.y > 0)) {
        positive.push_back(Point{it.x, it.y});
    } else {
        negative.push_back(Point{it.x, it.y});
    }
}

// Сортировка положительной половины и удаление повторяющихся углов:
std::sort(positive.begin(), positive.end(), [](const Point& a, const Point& b) {
    return a.y * b.x < b.y * a.x || (a.y * b.x == b.y * a.x && a.norm2() < b.norm2());
});

positive.erase(std::unique(positive.begin(), positive.end(), [](const Point& a, const P
    return a.y * b.x == b.y * a.x;
}), positive.end());

// Сортировка отрицательной половины и удаление повторяющихся углов:
std::sort(negative.begin(), negative.end(), [](const Point& a, const Point& b) {
    return a.y * b.x < b.y * a.x || (a.y * b.x == b.y * a.x && a.norm2() < b.norm2());
});

negative.erase(std::unique(negative.begin(), negative.end(), [](const Point& a, const P
    return a.y * b.x == b.y * a.x;
}), negative.end());

// Объединение двух половин точек:
p.clear();
p.insert(p.end(), positive.begin(), positive.end());
p.insert(p.end(), negative.begin(), negative.end());

if (p.size() == 1u) {
    return true;
}

const int n = p.size();

p.push_back(p[0]);

for (int i = 1; i <= n; ++i) {
    if (cross(p[i-1], p[i]) < 0) {
        return true;
    }
}

return false;
}

int main() {

    int side, n;
    scanf("%d %d", &side, &n);

    std::vector<Point> p;
    for (int i = 0; i < n; ++i) {
        int x, y;
        scanf("%d %d", &x, &y);
        assert(0 < x && x < side);
        assert(0 < y && y < side);
        p.push_back(Point{x,y});
    }
    printf(solve(side, p) ? "YES\n" : "NO\n");

    return 0;
}

```

ЗАДАЧА №911

Взвешивания

(Время: 1 сек. Память: 16 Мб Сложность: 40%)

Знаменитый химик Д. И. Менделеев, будучи директором Главной палаты мер и весов, интересовался задачей составления набора гирь, чтобы с их помощью можно было взвесить любой груз. Выяснилось, что если при взвешивании груза класть гири и на левую и на правую чашки весов, то самым удобным является набор гирь в троичной системе.

Для взвешиваний используют чашечные весы и большой набор гирь 1, 3, 9, 27, 81 грамм и т.д. (для любого $k \geq 0$ есть только одна гиря весом 3^k грамм).

На одну из чашек весов положили груз весом N грамм. Какие гири нужно положить на левую и правую чашки, чтобы их уравновесить?



Входные данные

Входной файл INPUT.TXT содержит символ C и число N , разделенные пробелом. C - символ «L» или «R», обозначающий соответственно левую или правую чашку весов, на которой лежит груз. N – масса груза в граммах ($1 \leq N \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите две строки: первая строка должна начинаться с "L:", вторая – с "R:", после чего через пробел должны идти в порядке возрастания массы веса гирь, которые нужно положить на левую и правую чашу весов, соответственно.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	L 5	L:1 3 R:9
2	R 3	L:3 R:

```
#include <stdio.h>
#include <vector>

int main() {
    char c; int n;
    scanf(" %c %d", &c, &n);

    std::vector<int> left, right;
    int pow3 = 1;
    do {
        int rem = n % 3;
        n /= 3;
        if (rem == 1) {
            right.push_back(pow3);
        } else if (rem == 2) {
```

```
        left.push_back(pow3);
        ++n;
    }
    pow3 *= 3;
} while (n > 0);

if (c == 'R') {
    left.swap(right);
}
printf("L:");
for(auto& it : left) {
    printf("%d ", it);
}
printf("\n");
printf("R:");
for(auto& it : right) {
    printf("%d ", it);
}
printf("\n");
return 0;
}
```


ЗАДАЧА №954

Стаканы

(Время: 1 сек. Память: 16 Мб Сложность: 40%)

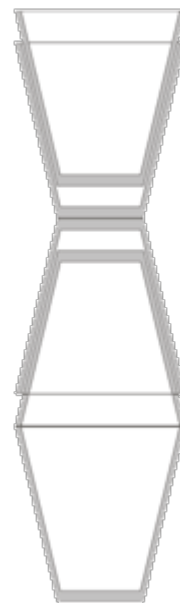
Как известно, стакан – предмет весьма функциональный. Самый банальный способ применения – ёмкость для жидкости, самый оригинальный ещё не изобретён. А мальчик Слава строит из стаканов башни, пользуясь удивительным свойством стаканов ставиться друг на друга или вставляться друг в друга.

Слава строит башни из стаканов высотой 10 сантиметров, которых у него имеется бесконечное количество. Стакан можно поставить на уже имеющуюся конструкцию либо дном вниз, либо дном вверх. Если предыдущий стакан установлен аналогично новому, то конструкция вырастет на 1 сантиметр, так как стаканы надеваются друг на друга. В противном случае башня вырастет на 10 сантиметров.

Однажды Слава заметил, что ни в коем случае нельзя вставлять друг в друга более трёх стаканов, иначе один из стаканов обязательно разобьётся.

На рисунке показан пример башни высотой 32 сантиметра из 5 стаканов.

Слава умудрился построить красивую башню высотой k сантиметров. Но когда он пошёл за фотоаппаратом, чтобы запечатлеть это достижение, случайно задел конструкцию, и башня упала. Пытаясь восстановить своё творение, Слава понял, что есть несколько способов построить башню аналогичной высоты. Помогите Славе вычислить точное количество способов.



Входные данные

Входной файл INPUT.TXT содержит натуральное число k ($k \leq 10^5$).

Выходные данные

В выходной файл OUTPUT.TXT выведите количество способов построить башню заданной высоты, взятое по модулю 10^6 .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	11	2
2	22	6
3	32	12

```
#include <iostream>
#include <vector>

int main() {
    int height;
    std::cin >> height;
    std::vector<int> count(height+1, 0);
    count[0] = 1;
    const int mod = 1000*1000;
    for (int i = 1; i <= height; ++i) {
```

```
    if (i-10 >= 0) {
        count[i] += count[i-10];
    }
    if (i-11 >= 0) {
        count[i] += count[i-11];
    }
    if (i-12 >= 0) {
        count[i] += count[i-12];
    }
    count[i] %= mod;
}
printf("%d\n", 2*count[height] % mod);

return 0;
}
```

ЗАДАЧА №955

Вирусы - 2

(Время: 1 сек. Память: 16 Мб Сложность: 40%)

Для моделирования различных объектов часто применяются так называемые клеточные поля. В простейшем случае – это прямоугольные таблицы, характеризующие некоторую область, а в каждой ячейке таблицы записывается какая-либо информация об исследуемом объекте. В биологии для моделирования распространения вирусов на плоской области в каждой ячейке помечается наличие вируса, а его распространение осуществляется в соседние ячейки по вертикали и горизонтали за одну единицу времени. Некоторые клетки обладают иммунитетом, заразить их невозможно и через них не распространяются вирусы. Напишите программу, которая найдёт минимально возможное число вирусов, с помощью которых можно заразить всю исследуемую прямоугольную область (за исключением защищённых клеток).

Входные данные

В первой строке входного файла INPUT.TXT содержится два натуральных числа n и m - размеры таблицы (количество строк и столбцов соответственно). Известно, что $1 \leq n, m \leq 100$. Во второй строке вначале записано одно число k – количество защищённых клеток, а далее записаны $2k$ чисел – координаты этих клеток y_i, x_i ($0 \leq k \leq n \times m, 1 \leq y_i \leq n, 1 \leq x_i \leq m$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – минимально возможное число вирусов.

Пример

Пояснения

В приведённом примере таблица имеет размер 4×5 , в ней символом «I» помечены защищённые клетки. Видно, что двух вирусов достаточно для заражения всей области. Их можно поместить, например, в клетки, помеченные символом «V».

V		I		
I	I			
			V	

№	INPUT.TXT	OUTPUT.TXT
1	4 5 3 1 3 2 1 2 2	2

/*
"Вирусы - 2": графы, заливка, компоненты связности, $O(n \times m)$
*/

```

#include <stdio.h>
#include <algorithm>
#include <functional>
#include <vector>

const int WALL = -1;
const int EMPTY = 0;

int main() {
    int nRows, nCols;
    scanf("%d %d", &nRows, &nCols);

    std::vector<std::vector<int>> map(1+nRows+1, std::vector<int>(1+nCols+1, EMPTY));
    for (int i = 1; i <= nCols; ++i) {
        map[0][i] = map[nRows+1][i] = WALL;
    }
    for (int i = 1; i <= nRows; ++i) {
        map[i][0] = map[i][nCols+1] = WALL;
    }
    int q;
    scanf("%d", &q);
    while (q--) {
        int r, c;
        scanf("%d %d", &r, &c);
        map[r][c] = WALL;
    }

    int nParts = 0;
    for (int r = 1; r <= nRows; ++r) {
        for (int c = 1; c <= nCols; ++c) {
            if (map[r][c] == EMPTY) {
                std::function<void(int,int)> visit = [&](const int r, const int c) {
                    for (int dr = -1; dr <= 1; ++dr) {
                        for (int dc = -1; dc <= 1; ++dc) {
                            if (dr * dr + dc * dc == 1 && map[r+dr][c+dc] == EMPTY) {
                                map[r+dr][c+dc] = nParts;
                                visit(r+dr, c+dc);
                            }
                        }
                    }
                };
                ++nParts;
                visit(r, c);
            }
        }
    }
    printf("%d", nParts);

    return 0;
}

```

ЗАДАЧА №150

Друзья

(Время: 1 сек. Память: 16 Мб Сложность: 41%)

В клубе N человек. Многие из них - друзья. Так же известно, что друзья друзей так же являются друзьями. Требуется выяснить, сколько всего друзей у конкретного человека в клубе.

Входные данные

В первой строке входного файла INPUT.TXT заданы два числа: N и S ($1 \leq N \leq 100$; $1 \leq S \leq N$), где N - количество человек в клубе, а S - номер конкретного человека. В следующих N строках записано по N чисел - матрица смежности, состоящая из единиц и нулей. Причем единица, стоящая в i -й строке и j -м столбце гарантирует, что люди с номерами i и j - друзья, а 0 - выражает неопределенность.

Выходные данные

В выходной файл OUTPUT.TXT выведите количество гарантированных друзей у человека с номером S , помня о транзитивности дружбы.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 1 0 1 0 1 0 1 0 1 0	2

```
#include <stdio.h>
#include <functional>
#include <vector>
#include <cassert>
#include <algorithm>
#include <numeric>
#include <functional>

int main() {
    int n, s;
    scanf("%d %d", &n, &s);
    --s;

    std::vector<std::vector<bool>> g(n, std::vector<bool>(n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            int v; scanf("%d", &v); g[i][j] = (v == 1);
        }
    }

    std::vector<int> visited(n, false);

    std::function<void(int)> visit = [&](const int curr) {
        visited[curr] = true;
        for (int next = 0; next < n; ++next) {
            if (!visited[next] && g[curr][next]) {
                visited[next] = true;
            }
        }
    };

    visit(s);

    printf("%d", std::count(visited.begin(), visited.end(), true));
    return 0;
}
```

```
        visit(next);
    }
};

visit(s);

printf("%d", std::accumulate(visited.begin(), visited.end(), -1));

return 0;
}
```

ЗАДАЧА №470

Земельный комитет

(Время: 5 сек. Память: 16 Мб Сложность: 41%)

Земельный комитет города принял решение о сдаче в аренду части муниципальной территории, имеющей форму прямоугольника размером H на W километров. Стоимость аренды каждого квадратного участка 1×1 км была определена с учётом локальных условий, и занесена в таблицу.

С целью организации открытого тендера на аренду, земельный комитет решил выставить на своём веб-сайте карту территории, и предоставить посетителям возможность узнавать суммарную стоимость аренды для произвольной прямоугольной группы соседних участков.

Данное предложение вызвало большой интерес у населения и предпринимателей, и нагрузка на сервер очень высока.

Требуется написать программу, позволяющую как можно более эффективно рассчитывать стоимость аренды для N запросов. В каждом запросе требуется определить общую стоимость участков внутри прямоугольной группы с противоположными углами, расположенными в элементах таблицы (a_i, b_i) и (c_i, d_i) .

Входные данные

В первой строке входного файла INPUT.TXT находятся числа H, W, N ($1 \leq H, W \leq 100, 1 \leq N \leq 10^6$). В следующих N строках содержится по W чисел (стоимости участков находятся в диапазоне от 0 до 10 000). Далее идут N строк с числами a_i, b_i, c_i и d_i ($1 \leq a_i \leq c_i \leq H, 1 \leq b_i \leq d_i \leq W$).

Выходные данные

В выходной файл OUTPUT.TXT должен содержать N чисел, по одному числу в строке.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 3 1 5 1 2 6 7 3 2 1 2 3	16

```
#include <stdio.h>
#include <stdlib.h>
#include <string>

char getChar() {
    static char buf[1 << 20];
    static int pos = 0;
    static int size = 0;
    if (pos == size) {
        size = fread(buf, 1, 1 << 20, stdin);
        pos = 0;
    }
    if (pos == size) {
```

```

        return EOF;
    }
    return buf[pos++];
}

template<typename T>
inline void read(T& v) {
    char c = '?';
    while (!(c == '-' || ('0' <= c && c <= '9'))) c = getChar();
    bool positive = true;
    if (c == '-') {
        positive = false;
        c = getChar();
    }
    v = 0;
    while ('0' <= c && c <= '9') {
        (v *= 10) += (c - '0');
        c = getChar();
    }
    if (!positive) v = -v;
}

void putChar(char c) {
    static char buf[1 << 20];
    static int size = 0;
    if (size == 1 << 20 || c == -1) {
        fwrite(buf, 1, size, stdout);
        size = 0;
    }
    if (c != -1) {
        buf[size++] = c;
    }
}

inline void writeln(int v) {
    char buf[15];
    sprintf(buf, "%d\n", v);
    for (char* cur = buf; *cur != '\0'; ++cur) {
        putChar(*cur);
    }
}

int main() {
    int nRows, nCols, nQueries;
    read(nRows), read(nCols), read(nQueries);

    static int sum[1+100][1+100];
    for (int r = 1; r <= nRows; ++r) {
        for (int c = 1; c <= nCols; ++c) {
            int value;
            read(value);
            sum[r][c] = sum[r-1][c]+sum[r][c-1]-sum[r-1][c-1]+value;
        }
    }

    while (nQueries--) {
        int r1, c1, r2, c2;
        read(r1), read(c1), read(r2), read(c2);
        writeln(sum[r2][c2] - sum[r1-1][c2] - sum[r2][c1-1] + sum[r1-1][c1-1]);
    }
    putChar(-1);
    return 0;
}

```

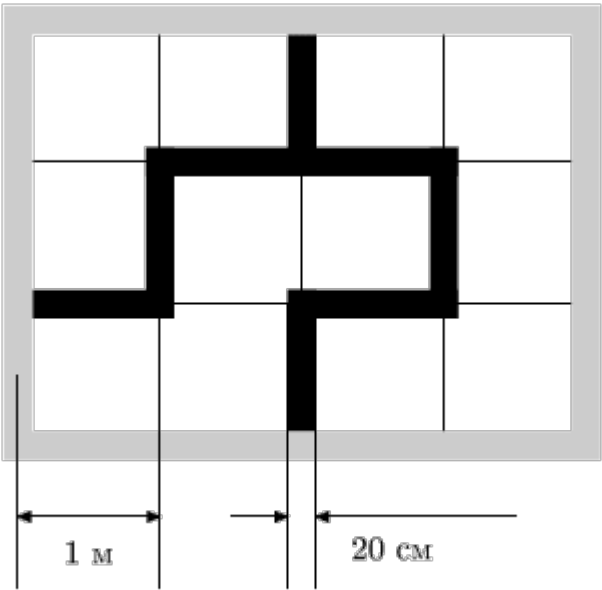

ЗАДАЧА №575

Строительство

(Время: 1 сек. Память: 16 Мб Сложность: 41%)

Фирма, в которой вы работаете, исполняет проект строительства суперсовременного бизнес-центра. И вот, когда заказ на склад строительных материалов почти готов к подписанию директором, оказывается, что туда забыли включить некоторые товары. Конкретно, в нем не были предусмотрены материалы, необходимые для возведения стен между отдельными секциями в подвальных помещениях. Необходимо срочно написать программу, которая сможет рассчитать необходимое количество материалов.

На плане подвальный этаж имеет вид прямоугольника, стороны которого лежат на линиях сетки с квадратными клетками. Сетка имеет такой масштаб, что сторона клетки соответствует одному метру стены подвального этажа. Каждая клетка на плане целиком принадлежит одному из помещений. Для каждой пары соседних по стороне клеток, отнесенных к разным помещениям, вдоль всей их общей стороны должна быть возведена стена толщиной 20 сантиметров и высотой три метра. Материалы для постройки всех внешних стен подвального этажа уже включены в заказ.



Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа N и M ($1 \leq N, M \leq 100$), разделенных пробелами - размеры подвала на плане. Каждая из N последующих строк содержит по M натуральных чисел, не превосходящих M x N, задающих номер помещения, к которому относится данная клетка. Эти числа разделены пробелами.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно вещественное число с точностью не менее 0.001 - общий объем возводимых стен в кубических метрах.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 4 1 1 3 3 1 2 2 3 2 2 3 3	4.56

```
#include <iostream>
#include <vector>
#include <iomanip>

int main() {
    std::ios_base::sync_with_stdio(false);
```

```

std::cin.tie(0); std::cout.tie(0);
int nRows, nCols; std::cin >> nRows >> nCols;
std::vector<std::vector<int>> arr(nRows, std::vector<int>(nCols));
for (int r = 0; r < nRows; ++r) {
    for (int c = 0; c < nCols; ++c) {
        std::cin >> arr[r][c];
    }
}
std::vector<std::vector<int>> map(10*nRows+2, std::vector<int>(10*nCols+2));
for (int r = 0; r < nRows; ++r) {
    for (int c = 0; c < nCols; ++c) {
        // 10 * r + 1, ..., 10 * r + 10
        // 10 * c + 1, ..., 10 * c + 10
        if (r > 0 && arr[r][c] != arr[r-1][c]) {
            const int i = 10 * r + 1;
            const int min_j = 10 * c + 1;
            const int max_j = 10 * c + 10;
            for (int j = min_j; j <= max_j; ++j) {
                map[i][j] = 1;
            }
        }
        if (c > 0 && arr[r][c] != arr[r][c-1]) {
            const int j = 10 * c + 1;
            const int min_i = 10 * r + 1;
            const int max_i = 10 * r + 10;
            for (int i = min_i; i <= max_i; ++i) {
                map[i][j] = 1;
            }
        }
        if (r+1 < nRows && arr[r][c] != arr[r+1][c]) {
            const int i = 10 * r + 10;
            const int min_j = 10 * c + 1;
            const int max_j = 10 * c + 10;
            for (int j = min_j; j <= max_j; ++j) {
                map[i][j] = 1;
            }
        }
        if (c+1 < nCols && arr[r][c] != arr[r][c+1]) {
            const int j = 10 * c + 10;
            const int min_i = 10 * r + 1;
            const int max_i = 10 * r + 10;
            for (int i = min_i; i <= max_i; ++i) {
                map[i][j] = 1;
            }
        }
        if (r > 0 && c > 0 && arr[r][c] != arr[r-1][c-1]) {
            map[10*r+1][10*c+1] = 1;
        }
        if (r > 0 && c+1 < nCols && arr[r][c] != arr[r-1][c+1]) {
            map[10*r+1][10*c+10] = 1;
        }
        if (r+1 < nRows && c > 0 && arr[r][c] != arr[r+1][c-1]) {
            map[10*r+10][10*c+1] = 1;
        }
        if (r+1 < nRows && c+1 < nCols && arr[r][c] != arr[r+1][c+1]) {
            map[10*r+10][10*c+10] = 1;
        }
    }
}
int count = 0;
for (int r = 2; r < 10 * nRows; ++r) {
    for (int c = 2; c < 10 * nCols; ++c) {
        count += map[r][c];
    }
}
std::cout << std::fixed << std::setprecision(3) << count / 100.0 * 3;
return 0;
}

```

ЗАДАЧА №687

Однонаправленная задача коммивояжёра

(Время: 1 сек. Память: 16 Мб Сложность: 41%)

Задана матрица размером $m \times n$ из целых чисел. Путь начинается в любой строке первого столбца и состоит из последовательности шагов, обрывающихся в столбце n . Каждый шаг состоит в переходе из столбца i в столбец $i+1$ в соседнюю (по горизонтали или диагонали) ячейку. Весом пути называется сумма целых чисел, записанных в каждой из n посещенных ячеек.

Требуется написать программу, которая вычисляет путь с минимальным весом с левого края матрицы до правого.

Входные данные

Входной текстовый файл INPUT.TXT содержит в первой строке количество строк и столбцов матрицы, которые обозначаются m и n соответственно. Далее следует m строк по n чисел в каждой. Числа отделяются друг от друга пробелами. Число строк не превышает 10, столбцов – 100. Вес любого пути не будет превышать целого числа, для хранения которого потребуется больше 30 бит.

Выходные данные

Выходной текстовый файл OUTPUT.TXT должен содержать две строки. Первая строка задает путь минимальной стоимости, а вторая – соответственно стоимость этого пути. Путь состоит из последовательности n целых чисел (разделенных одним или более пробелами), задающих номера строк, из которых состоит путь минимальной стоимости. Если путей минимальной стоимости больше одного, то должен быть выведен лексикографически минимальный путь.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 6 3 4 1 2 8 6 6 1 8 2 7 4 5 9 3 9 9 5 8 4 1 3 2 6 3 7 2 8 6 4	1 2 3 4 4 5 16

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
#include <cassert>

int main() {
    int nRows, nCols;
    std::cin >> nRows >> nCols;
    std::vector<std::vector<int>>> arr(nRows, std::vector<int>(nCols));
    for (auto& row : arr) {
        for (auto& it : row) {
            std::cin >> it;
        }
    }
}
```

```

std::vector<std::vector<int>> best(nRows, std::vector<int>(nCols, INT_MAX));
std::vector<std::vector<int>> from(nRows, std::vector<int>(nCols, -1));
for (int col = 0; col+1 < nCols; ++col) {
    for (int row = 0; row < nRows; ++row) {
        if (col == 0) {
            best[row][col] = arr[row][col];
        }
        if (row-1 >= 0 && best[row][col] + arr[row-1][col+1] < best[row-1][col+1]) {
            best[row-1][col+1] = best[row][col] + arr[row-1][col+1];
            from[row-1][col+1] = row;
        }
        if (best[row][col]+arr[row][col+1] < best[row][col+1]) {
            best[row][col+1] = best[row][col] + arr[row][col+1];
            from[row][col+1] = row;
        }
        if (row+1 < nRows && best[row][col] + arr[row+1][col+1] < best[row+1][col+1]) {
            best[row+1][col+1] = best[row][col] + arr[row+1][col+1];
            from[row+1][col+1] = row;
        }
    }
}

std::vector<int> path;
int curr = -1;
int result = INT_MAX;
for (int row = 0; row < nRows; ++row) {
    if (result > best[row].back()) {
        result = best[row].back();
        curr = row;
    }
}
assert(curr != -1);
path.push_back(curr);
for (int col = nCols-1; col >= 1; --col) {
    path.push_back(from[curr][col]);
    curr = from[curr][col];
}
std::reverse(path.begin(), path.end());
for (auto& it : path) {
    std::cout << it+1 << ' ';
}
std::cout << std::endl << result << std::endl;
return 0;
}

```

ЗАДАЧА №858

Площадь треугольника - 2

(Время: 1 сек. Память: 16 Мб Сложность: 41%)

Три попарно непараллельные прямые заданы коэффициентами a_i , b_i , c_i соответствующего уравнения $a_i x + b_i y = c_i$. Коэффициенты a_i и b_i не могут быть одновременно равны нулю.

Требуется написать программу, определяющую площадь треугольника, образованного этими прямыми.

Входные данные

Входной файл INPUT.TXT содержит 3 строки, в каждой из которых записаны коэффициенты a_i , b_i , c_i , разделенные пробелом. Все коэффициенты – целые числа, не превосходящие 10000 по абсолютной величине.

Выходные данные

В выходной файл OUTPUT.TXT выведите значение площади треугольника с точностью не меньшей, чем три знака после запятой.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	0 1 1 1 0 1 4 3 19	6.000

```
#include <iostream>
#include <iomanip>
#include <cmath>

typedef long double Real;

struct Point {
    Real x, y;
};

Point operator-(const Point& a, const Point& b) {
    return Point{a.x - b.x, a.y - b.y};
}

Real cross(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}

struct Line {
    int a, b, c;

    static Line read() {
        int a, b, c;
        std::cin >> a >> b >> c;
        return Line{a,b,c};
    }
};
```

```

}

Point intersect(const Line& other) const {
    const int a1 = a, b1 = b, c1 = c, a2 = other.a, b2 = other.b, c2 = other.c;
    // a1 * x + b1 * y = c1
    // a2 * x + b2 * y = c2
    const auto x = Real(c1 * b2 - c2 * b1) / (a2 * b1 - a1 * b2);
    const auto y = Real(a2 * c1 - a1 * c2) / (a2 * b1 - a1 * b2);
    return Point{x,y};
}

};

int main() {
    auto l1 = Line::read(), l2 = Line::read(), l3 = Line::read();
    auto A = l1.intersect(l2), B = l1.intersect(l3), C = l2.intersect(l3);
    std::cout << std::fixed << std::setprecision(3) << std::abs(cross(B-A,C-A)) / 2;
    return 0;
}

```

ЗАДАЧА №936

Алтайский технический университет

(Время: 1 сек. Память: 16 Мб Сложность: 41%)

Как известно, в Барнауле на протяжении многих лет проводятся олимпиады по программированию. Там бывали многие студенты и школьники из нашего города. Наверняка, все запомнили здание Алтайского технического университета и памятник Ползунову на площади перед ним.

Площадь перед университетом имеет форму круга с памятником Ползунову в центре. По ночам памятнику скучно, и он наблюдает окружающий мир, поворачиваясь вокруг своей оси, но, не сходя со своего пьедестала. К сожалению, растущие вокруг деревья затрудняют памятнику обзор, поэтому он видит хорошо на расстоянии, не превышающем радиус площади - R . А поскольку глаз на затылке у памятника нет, он может наблюдать только за теми событиями, которые расположены в полукруге радиуса R . Точки на границе полукруга памятник видит тоже хорошо.

Понятно, что памятник хочет наблюдать как можно больше людей на площади. Ваша задача — написать программу, определяющую максимальное количество людей, которые может наблюдать памятник.

Входные данные

Первая строка входного файла INPUT.TXT содержит три числа: X, Y — целые координаты памятника и R — вещественный радиус площади ($R > 0$). Во второй строке указано целое число N — количество людей на площади ($1 \leq N \leq 150$). Далее в N строках перечислены координаты точек (x_i, y_i) , в которых находятся люди. Все координаты являются целыми числами, не превышающими по модулю 1000.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число — максимальное количество людей, которые может наблюдать памятник.

Примеры

Пояснения к примерам

Пример 1

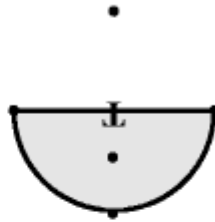


оптимальный поворот

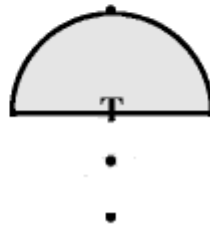


не оптимальный поворот

Пример 2



оптимальный поворот



не оптимальный поворот

№	INPUT.TXT	OUTPUT.TXT
1	25 25 3.5 7 25 28 23 27 27 27 24 23 26 23 24 29 26 29	3
2	350 200 2 5 350 202 350 199 350 198 348 200 352 200	4

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cmath>

typedef long double ld;

struct Point {
    int x, y;

    static Point read() {
        int x, y;
        scanf("%d %d", &x, &y);
        return Point{x,y};
    }

    inline int norm2() const {
        return x * x + y * y;
    }
}
```



```

    inline ld angle() const {
        return std::atan2(ld(y), ld(x));
    }
};

inline int cross(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}

int main() {
    int xc, yc; double R;
    scanf("%d %d %lf", &xc, &yc, &R);
    int n;
    scanf("%d", &n);

    std::vector<Point> points;

    while (n--) {
        auto p = Point::read();
        p.x -= xc;
        p.y -= yc;
        if (p.x*p.x+p.y*p.y > R*R) {
            continue;
        }
        points.push_back(p);
    }

    std::sort(points.begin(), points.end(), [](const Point& a, const Point& b) {
        return a.angle() < b.angle() || (a.angle() == b.angle() && a.norm2() < b.norm2());
    });

    n = points.size();

    if (n <= 1) {
        printf("%d", n);
        return 0;
    }

    int l = 0, r = 0, answ = 1;
    while (l < n) {
        while ((r+1) % n != l && cross(points[l], points[(r+1) % n]) >= 0) {
            (++r) %= n;
        }
        answ = std::max(answ, r < l ? r+n-l+1 : r-l+1);
        ++l;
    }
    printf("%d", answ);
    return 0;
}

```

ЗАДАЧА №944

Размен

(Время: 1 сек. Память: 16 Мб Сложность: 41%)

У вас имеется неограниченное количество монеток N разных достоинств. Определить, можно ли с их помощью разменять заданные K сумм денег.

Входные данные

В первой строке входного файла INPUT.TXT задано число N, далее во второй строке записаны N чисел, задающих достоинства монеток. В третьей строке задано число K – количество сумм. В четвертой строке располагаются K чисел, определяющих размеры сумм. Все числа во входном файле натуральны и не превосходят 1000.

Выходные данные

В выходной файл OUTPUT.TXT выведите K чисел через пробел: для каждой суммы следует вывести 0, если ее разменять нельзя, и 1, если можно.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 3 5 5 3 6 7 11 12	1 1 0 1 1
2	1 10 1 3	0

```
#include <stdio.h>
#include <vector>
#include <algorithm>

int main() {
    int n; scanf("%d", &n);
    std::vector<int> coin(n);
    for (auto& it : coin) {
        scanf("%d", &it);
    }
    std::vector<char> possible(1+1000, false);
    possible[0] = true;
    for (int i = 1; i <= 1000; ++i) {
        for (auto c : coin) {
            if (i-c >= 0 && possible[i-c]) {
                possible[i] = true;
            }
        }
    }
    int nQ;
    scanf("%d", &nQ);
    while (nQ-->
```

```
        int sum; scanf("%d", &sum);
        printf("%d ", (int)possible[sum]);
    }
    return 0;
}
```

ЗАДАЧА №1573

Юбилей

(Время: 1 сек. Память: 16 Мб Сложность: 41%)

Ромка недавно вернулся с юбилея своего друга Сашки. Праздник отмечался с размахом и Ромка подумал, а что если бы юбилеи были чаще? Можно же не ограничиваться десятичной системой счисления. Ромка ввёл понятие юбилейности числа, равное максимальному количеству нулей в конце записи этого числа в какой-то системе счисления с основанием В, где В — целое число, большее единицы. Например, юбилейность числа 256 равна 8, так как в двоичной системе счисления оно оканчивается на 8 нулей. Ромка хочет узнать, когда его ближайший значимый юбилей, если в прошлом месяце ему исполнилось X лет? Значимым юбилеем он считает количество лет, которое обладает юбилейностью хотя бы L.

Входные данные

Единственная строка входного файла INPUT.TXT содержит два целых числа X и L ($1 \leq X \leq 10^{12}$; $1 \leq L \leq 50$).

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное число — ответ на задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	9990 4	10000
2	100 8	256
3	100 2	104

```
#include <iostream>
#include <climits>
#include <cassert>

typedef long long ll;

inline ll pow(ll a, int n) {
    ll res = 1;
    while (n > 0) {
        if (n & 1) {
            res *= a;
        }
        a *= a;
        n >>= 1;
    }
    return res;
}

ll solve(ll num, int low) {
    if (low == 1) {
        return num+1;
    }
    if (num < (ll(1) << low)) {
        return ll(1) << low;
    }
}
```

```

    }
    ll answ = LLONG_MAX;
    bool finished = false;
    for (int base = 2; !finished; ++base) {
        ll pow = ::pow(base, low);
        ll temp = (num+pow-1) / pow * pow;
        if (temp == num) {
            temp += pow;
        } else if (pow > num) {
            finished = true;
        }
        answ = std::min(answ, temp);
    }
    assert(answ > num && answ < LLONG_MAX);
    return answ;
}

int main() {
    ll num; int low;
    std::cin >> num >> low;
    std::cout << solve(num, low);
    return 0;
}

```

ЗАДАЧА №181

Космический мусорщик

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

В околоземном космическом пространстве накопилось много мусора, поэтому ученые сконструировали специальный аппарат ловушку для космического мусора. Аппарат должен двигаться по достаточно сложной траектории, сжигая по пути мусор. Ловушка может передвигаться в пространстве по 6 направлениям: на север (N), на юг (S), на запад (W), на восток (E), вверх (U) и вниз (D). Движением ловушки управляет процессор. Программа движения задается шестью правилами движения, которые соответствуют каждому из указанных направлений. Каждое такое правило представляет собой строку символов из множества {N, S, W, E, U, D}.

Команда ловушки состоит из символа направления и целого положительного числа M. Если параметр больше 1, то ловушка перемещается на один метр в направлении, которое указано в команде, а затем последовательно выполняет команды, заданные правилом для данного направления, с параметром меньше на 1. Если же параметр равен 1, то просто перемещается на один метр в указанном направлении.

Входные данные

Первые шесть строк входного файла INPUT.TXT задают правила для команд с направлением N, S, W, E, U и D соответственно. Каждая строка содержит не более 100 символов (и может быть пустой). Следующая строка содержит команду ловушки: сначала символ из множества {N, S, W, E, U, D}, затем пробел и параметр команды – целое положительное число, не превышающее 100.

Выходные данные

Выведите в выходной файл OUTPUT.TXT единственное число - количество перемещений, которое совершит ловушка. Гарантируется, что ответ не превышает 10⁹.

Пример

Пусть, например, заданы правила, отраженные в таблице справа. Тогда при выполнении команды S(3) мусорщик сначала переместится на 1 метр в направлении S, а потом выполнит последовательно команды N(2), U(2), S(2), D(2), D(2), U(2), S(2), E(2).

Если далее проанализировать действия мусорщика, получим, что в целом он совершит ровно 34 перемещения.

Направление	Правило
N	N
S	NUSDDUSE
W	UEWWD
E	
U	U
D	WED

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	N NUSDDUSE UEWWD U WED S 3	34
---	---	----

```
#pragma GCC diagnostic ignored "-Wunused-result"
#include <stdio.h>
#include <bits/stdc++.h>

int main() {

    std::string alp = "NSWEUD";

    std::vector<std::vector<int>> coeff(6, std::vector<int>(6, 0));
    for (int i = 0; i < 6; ++i) {
        char buf[101] = {};
        scanf("%100[^\n]", buf);
        char c;
        scanf("%c", &c);
        assert(c == '\n');
        std::string s(buf);
        for (auto& it : s) {
            int pos = int(std::find(alp.begin(), alp.end(), it)-alp.begin());
            coeff[i][pos]++;
        }
        //scanf("\n");
    }
    char c = '?'; int level;
    scanf(" %c %d", &c, &level);

    std::vector<std::vector<int>> count(level+1, std::vector<int>(6, -1));
    std::function<int(int, int)> get = [&](const int level, const int pos) {
        if (level == 1) {
            return 1;
        }
        if (count[level][pos] == -1) {
            int answ = 1;
            for (int i = 0; i < 6; ++i) {
                answ += coeff[pos][i] * get(level-1, i);
            }
            count[level][pos] = answ;
        }
        return count[level][pos];
    };
    printf("%d\n", get(level, alp.find(c)));
    return 0;
}
```

Делимость на 7

Требуется определить делимость на 7 ряда целых чисел, записанных в двоичной системе счисления.

Входные данные

Выходные данные

Примеры

[illegible]

```
#include <stdio.h>
#include <string>
#include <algorithm>

int main() {
    int n;
    scanf("%d", &n);
    while (n--) {
        char buf[1000+1];
        scanf("%1000s", buf);
        std::string s(buf);
        std::reverse(s.begin(), s.end());
        int mod = 0;
        int pow = 1;
        for (auto& it : s) {
            (mod += pow * (it - '0')) %= 7;
            (pow *= 2) %= 7;
        }
        printf(mod ? "No\n" : "Yes\n");
    }
    return 0;
}
```


ЗАДАЧА №288

Комментарии

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

Программист Билл недавно узнал, что, чем больше комментариев содержит исходный текст, тем он лучше. Теперь он хочет проверить, насколько хороши его собственные программы, написанные на языке Pascal. Но поскольку самому считать комментарии очень утомительно, Билл попросил Вас сделать эту работу за него.

Исходный текст может содержать комментарии трёх типов:

1. // ...
2. { ... }
3. (* ... *)

Комментарий первого типа начинается составным символом // и продолжается до конца строки. Комментарий второго типа начинается символом { и заканчивается символом }. Он может размещаться в нескольких строках. Комментарий третьего типа начинается составным символом (* и заканчивается составным символом *). Он также может размещаться в нескольких строках.

Комментарии не могут быть вложены друг в друга, так что запись вида {...//...(*...*)...} является одним комментарием второго типа. Комментарии не могут размещаться внутри символьных строк, так что запись '...(**)...{}...' не содержит ни одного комментария.

Входные данные

Во входном файле INPUT.TXT записан исходный текст программы на языке Pascal. Размер текста не превосходит 16 Кб.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – количество комментариев в исходном тексте программы.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	{ my first program } begin writeln('hello world'); end.	1
2	const comments = '{ }(**)//'; begin writeln(comments); end.	0

3	// comment begin writeln('{ string }'); { while (true) do; } end.	2
---	---	---

```
#pragma GCC diagnostic ignored "-Wunused-result"
```

```
#include <stdio.h>
```

```
#include <bits/stdc++.h>
```

```
int main() {
    // Чтение текста в вектор:
    std::vector<char> text;
    for (char c; c = getchar(), c != EOF; text.push_back(c));
    text.push_back('\n'); text.push_back EOF);

    // Проход по вектору:
    int answ = 0;
    for (int i = 0; text[i] != EOF; ++i) {
        if (text[i] == '\\') { // Пропустить строку '..'
            ++i;
            while (text[i] != '\\') ++i;
            assert(text[i] == '\\');
        }
        if (text[i] == '{') { // Пропустить комментарий {...}
            ++answ;
            while (text[i] != '}') ++i;
            assert(text[i] == '}');
        }
        if (i > 0 && text[i-1] == '/' && text[i] == '/') {
            // Пропустить комментарий //...\n
            ++answ;
            while (text[i] != '\n') ++i;
            assert(text[i] == '\n');
        }
        if (i > 0 && text[i-1] == '(' && text[i] == '*') {
            // Пропустить комментарий (*...*)
            i += 2;
            ++answ;
            while (!(text[i-1] == '*' && text[i] == ')) ++i;
            assert(text[i-1] == '*' && text[i] == '));
        }
    }
    printf("%d\n", answ);
    return 0;
}
```

ЗАДАЧА №557

Матрицы

(Время: 2 сек. Память: 16 Мб Сложность: 42%)

Аня недавно узнала, что такое квадратная матрица размерности n . Это таблица $n \times n$ с целыми числами в ячейках. Число, стоящее на пересечении i -ой строки и j -ого столбца матрицы A , кратко обозначается $A[i, j]$. Матрицы можно умножать, и Аня быстро освоила, как запрограммировать эту операцию с помощью циклов. Результатом умножения двух матриц A и B будет матрица C , элементы которой определяются следующим образом:

$$C[i, j] = \sum_{k=1}^n A[i, k]B[k, j]$$

Матрицы ей понадобились для конкретной задачи, в которой надо узнать определенный элемент произведения нескольких матриц. Это уже достаточно сложная задача для Ани, но она усложняется тем, что все вычисления ведутся по модулю некоторого простого числа p , то есть если при арифметических операциях получается число, большее, либо равное p , оно заменяется на остаток при делении на p .

Помогите Ане вычислить нужный ей элемент.

Входные данные

В первой строчке входного файла INPUT.TXT стоят два числа: m - количество матриц, n - размер каждой из матриц ($1 \leq m \leq 130$, $1 \leq n \leq 130$). В следующей строчке содержатся номер строки и столбца, интересующего Аню элемента $1 \leq a \leq n$, $1 \leq b \leq n$. В третьей строке содержится простое число $p \leq 1000$. Далее следует описание m матриц. Описание каждой матрицы состоит из n строк. В каждой из строк содержится n неотрицательных целых чисел, меньших p . Соседние числа в строке разделены пробелом, а перед каждой матрицей пропущена строка.

Выходные данные

В выходной файл OUTPUT.TXT выведите нужный Ане элемент произведения матриц.

Пример

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	3 2	20
	1 2	
	239	
	1 2	
	3 4	
	4 2	
	1 3	
	1 2	
	2 1	

```
#pragma GCC diagnostic ignored "-Wunused-result"
#include <stdio.h>
```

```
char getChar() {
    static char buffer[1024*1024];
    static int pos = 0;
    static int size = 0;
    if (pos == size) {
        size = fread(buffer, 1, 1024*1024, stdin);
        pos = 0;
    }
    if (pos == size) {
        return EOF;
    }
    return buffer[pos++];
}

int getInt() {
    char c = '?';
    while (!(c == '-' || ('0' <= c && c <= '9'))) {
        c = getChar();
    }
    int sign = 1;
    if (c == '-') {
        sign = -1;
        c = getChar();
    }
    int answ = 0;
    while ('0' <= c && c <= '9') {
        (answ *= 10) += (c - '0');
        c = getChar();
    }
    return sign * answ;
}

int main() {
    int n = getInt(), dim = getInt(), need_row = getInt(), need_col = getInt(), mod = getIn
--need_row, --need_col;

    static int answ[130][130], temp[130][130], curr[130][130];

    for (int i = 0; i < dim; ++i) {
        answ[i][i] = 1;
    }

    for (int id = 0; id < n; ++id) {
        for (int row = 0; row < dim; ++row) {
            for (int col = 0; col < dim; ++col) {
                curr[col][row] = getInt();
            }
        }
        for (int row1 = 0; row1 < dim; ++row1) {
            for (int col1 = 0; col1 < dim; ++col1) {
```

```
        int sum = 0;
        for (int k = 0; k < dim; ++k) {
            sum += answ[row1][k] * curr[col1][k];
        }
        temp[row1][col1] = sum % mod;
    }
}
for (int row = 0; row < dim; ++row) {
    for (int col = 0; col < dim; ++col) {
        answ[row][col] = temp[row][col];
    }
}

printf("%d\n", answ[need_row][need_col]);

return 0;
}
```

ЗАДАЧА №619

Бросание кубика

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

Кубик, грани которого помечены цифрами от 1 до 6, бросают N раз. Требуется найти вероятность того, что сумма выпавших чисел будет равна Q.

Входные данные

Входной файл INPUT.TXT содержит натуральные числа N и Q ($N \leq 500$, $Q \leq 3000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное вещественное число – искомую вероятность, которая должна отличаться от истинного значения не более чем на 10^{-6} .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 6	0.166667
2	1 7	0
3	4 14	0.112654
4	100 100	1.530647E-78

```
#include <iostream>
#include <iomanip>
#include <vector>

typedef long double Real;

int main() {
    int n, sum; std::cin >> n >> sum;
    if (sum < n || sum > 6 * n) {
        std::cout << "0";
        return 0;
    }
    std::vector<Real> curr(1+sum), next(1+sum);
    curr[0] = 1;
    int limit = 0;
    for (int i = 1; i <= n; ++i) {
        limit = std::min(limit+6, sum);
        for (int s = 0; s <= limit; ++s) {
            Real value = 0;
            for (int score = 1; score <= 6; ++score) {
                if (s - score >= 0) {
                    value += curr[s-score];
                }
            }
            next[s] = value / 6;
        }
        curr.swap(next);
        for (int s = 0; s <= limit; ++s) {
            next[s] = 0;
        }
    }
}
```

```
    }  
}  
std::cout << std::fixed << std::setprecision(6) << curr[sum];  
return 0;  
}
```

ЗАДАЧА №630

Охрана

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

На секретной военной базе работает N охранников. Сутки поделены на 10000 равных промежутков времени, и известно когда каждый из охранников приходит на дежурство и уходит с него. Например, если охранник приходит в 5, а уходит в 8, то значит, что он был в 6, 7 и 8-ой промежуток. В связи с уменьшением финансирования часть охранников решено было сократить. Укажите: верно ли то, что для данного набора охранников, объект охраняется в любой момент времени хотя бы одним охранником и удаление любого из них приводит к появлению промежутка времени, когда объект не охраняется.

Входные данные

В первой строке входного файла INPUT.TXT записано натуральное число K ($1 \leq K \leq 30$) – количество тестов в файле. Каждый тест начинается с числа N ($1 \leq N \leq 10000$), за которым следует N пар неотрицательных целых чисел A и B - время прихода на дежурство и ухода ($0 \leq A < B \leq 10000$) соответствующего охранника. Все числа во входном файле разделены пробелами и/или переводами строки.

Выходные данные

В выходной файл OUTPUT.TXT выведите K строк, где в M -ой строке находится слово Accepted, если M -ый набор охранников удовлетворяет описанным выше условиям. В противном случае выведите Wrong Answer.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	2 3 0 3000 2500 7000 2700 10000 2 0 3000 2700 10000	Wrong Answer Accepted

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>
```

```
struct DataStruct {
    std::vector<int> arr, add;

    const int GSIZE = 128;

    DataStruct(int size, int value) {
        arr.assign(size, value);
        add.assign(size / GSIZE + 1, 0);
    }

    void push(int g) {
        if (add[g] == 0) {
            return;
        }
        const int begin = GSIZE * g;
```



```

        const int after = std::min(begin + GSIZE, (int)arr.size());
        int a = add[g];
        for (int i = begin; i < after; ++i) {
            arr[i] += a;
        }
        add[g] = 0;
    }

    void push_all() {
        for (int g = 0; g < (int)add.size(); ++g) {
            push(g);
        }
    }

    void inc(int left, int right) {
        assert(right < (int)arr.size());
        int gl = left / GSIZE;
        int gr = right / GSIZE;
        if (gl == gr) {
            push(gl);
            for (int i = left; i <= right; ++i) {
                arr[i]++;
            }
        } else {
            push(gl);
            push(gr);
            for (int i = left, after = (gl+1) * GSIZE; i < after; ++i) {
                arr[i]++;
            }
            for (int g = gl+1; g < gr; ++g) {
                add[g]++;
            }
            for (int i = gr * GSIZE; i <= right; ++i) {
                arr[i]++;
            }
        }
    }
};

bool solve(const std::vector<std::pair<int, int>>& input) {
    DataStruct ds(10000, 0);
    for (const auto& p : input) {
        ds.inc(p.first, p.second);
    }
    ds.push_all();
    for (int i = 0; i < (int)ds.arr.size(); ++i) {
        const auto& it = ds.arr[i];
        if (it == 0) {
            return false;
        }
    }
    std::vector<int> pref{0};
    for (auto& it : ds.arr) {
        pref.push_back(pref.back() + (it >= 2));
    }
    for (const auto& p : input) {
        if (pref[p.second+1] - pref[p.first] >= (p.second-p.first+1)) {
            return false;
        }
    }
    return true;
}

int main() {
    int nQueries;
    scanf("%d", &nQueries);
    while (nQueries--) {
        int n;
        scanf("%d", &n);
    }
}

```

```
std::vector<std::pair<int, int>> input;
for (int i = 0; i < n; ++i) {
    int a, b;
    scanf("%d %d", &a, &b);
    input.push_back(std::make_pair(a, b-1));
}
printf(solve(input) ? "Accepted\n" : "Wrong Answer\n");
}
return 0;
}
```

ЗАДАЧА №695

Мифические шахматы

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

Ваш друг Вася занимается разработкой компьютерной игры «Мифические шахматы». Он не укладывается в установленные сроки сдачи проекта.

Вася обратился к друзьям за помощью. Ему необходим модуль, вычисляющий оптимальные пути перемещения фигур из одной клетки в другую. Так как друзей у Васи много, то каждому досталась маленькая подзадача. Вам требуется написать программу, определяющую минимальное количество ходов, необходимое кентавру, чтобы добраться из одной клетки в другую.

В мифические шахматы играют на шахматной доске размером 9x9, угловые клетки которой окрашены в черный цвет. Кентавр – фигура мифических шахмат, объединяющая в себе свойства коня и слона. Когда кентавр стоит на белой клетке, он может ходить только как конь, а когда на черной – только как слон. На рисунках приведены варианты ходов для двух кентавров (буквой «К» отмечено местоположение кентавра, а звездочками – клетки, куда кентавр может сделать ход).

9					*		*	
8						K		
7					*		*	
6				*				*
5			*					*
4		*						
3	*							
2								
1								
	A	B	C	D	E	F	G	H

9								
8								
7			*		*			
6		*				*		
5				K				
4		*			*			
3			*		*			
2								
1								
	A	B	C	D	E	F	G	H

Входные данные

В первой строке входного файла INPUT.TXT содержатся координаты (большая английская буква и цифра) двух клеток доски для мифических шахмат, разделенные пробелом.

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное число ходов, необходимое кентавру, чтобы добраться из первой клетки во вторую. Если добраться невозможно, то следует вывести число «-1» (без кавычек).

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	H6 E5	2
2	A6 F6	3

```
#include <iostream>
#include <queue>
#include <vector>

int main() {
    char c1, r1, c2, r2;
    scanf(" %c%c %c%c", &c1, &r1, &c2, &r2);
    c1 -= 'A'; r1 -= '1';
    c2 -= 'A'; r2 -= '1';

    std::vector<std::vector<int>> dist(9, std::vector<int>(9, -1));
    dist[r1][c1] = 0;
    std::queue<int> queue;
```

```

queue.push(r1*16+c1);
while (!queue.empty()) {
    auto curr = queue.front(); queue.pop();
    const int r = curr / 16;
    const int c = curr % 16;
    if ((r + c) % 2) { // white
        for (int dr = -2; dr <= 2; ++dr) {
            for (int dc = -2; dc <= 2; ++dc) {
                if (dr*dr + dc*dc != 5) continue;
                int nr = r + dr;
                int nc = c + dc;
                if (nr < 0 || nr > 8 || nc < 0 || nc > 8) {
                    continue;
                }
                if (dist[nr][nc] == -1) {
                    dist[nr][nc] = dist[r][c]+1;
                    queue.push(nr*16+nc);
                }
            }
        }
    } else {
        for (int dr = -1; dr <= 1; ++dr) {
            for (int dc = -1; dc <= 1; ++dc) {
                if (dr*dr + dc*dc != 2) continue;
                int nr = r + dr;
                int nc = c + dc;
                while (0 <= nr && nr < 9 && 0 <= nc && nc < 9) {
                    if (dist[nr][nc] == -1) {
                        dist[nr][nc] = dist[r][c]+1;
                        queue.push(nr*16+nc);
                    }
                    nr += dr;
                    nc += dc;
                }
            }
        }
    }
}
std::cout << dist[r2][c2];
return 0;
}

```

ЗАДАЧА №781

Две цифры

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

Сколько N-значных чисел можно составить, используя цифры 5 и 9, в которых три одинаковые цифры не стоят рядом?

Входные данные

Во входном файле INPUT.TXT записано число N ($1 \leq N \leq 30$).

Выходные данные

В выходной файл OUTPUT.TXT нужно вывести одно число - количество чисел с указанным свойством.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3	6

```
#include <stdio.h>

int solve(int len) {
    static int prepared = false;
    static int count[2][1+30]; // count[last][len]
    if (!prepared) {
        prepared = true;
        count[0][1] = count[1][1] = 1;
        count[0][2] = count[1][2] = 2;
        count[0][3] = count[1][3] = 3;
        for (int len = 4; len <= 30; ++len) {
            for (int last = 0; last < 2; ++last) {
                count[last][len] = count[0][len-1]+count[1][len-1]-count[1-last][len-3];
            }
        }
    }
    return count[0][len] + count[1][len];
}

int main() {
    int n;
    scanf("%d", &n);
    printf("%d", solve(n));
    return 0;
}
```

ЗАДАЧА №881

Фатализм

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

Роботу Прайму (конструктору роботов) надоели его творения – они получаются слишком тупыми и бездумными, и среди них не найти собеседника и даже просто мыслителя, достойного общаться с самим Праймом, великим и неповторимым. Прайм испробовал уже все известные ему методы, но создать себе подобного до сих пор не получилось. В отчаянии он придумал себе игру, чтобы как то отвлечься от своей неразрешимой проблемы, преодолеть творческий кризис и заодно пустить побольше своих творений по правильному пути. Игра называется «Фатализм» и заключается в следующем.

Прайм создает лабиринт размера $m \times n$ клеток, огороженный со всех сторон стенами. Каждая клетка лабиринта – либо свободное пространство, либо стена. В начальный момент времени в некоторых клетках, где нет стены, стоят роботы и смотрят в одном из четырех направлений. Никакие два робота не стоят в одной клетке.

Каждый робот движется с постоянной скоростью – одна клетка в секунду – в направлении, в котором он смотрит, и светит лазером в этом же направлении до ближайшей стены. В конце каждой секунды некоторые роботы взрываются. Это происходит в трех случаях:

- если робот оказался за пределами лабиринта или в клетке, которая заполнена стеной, то он взрывается;
- если в какой-то клетке оказалось два или более робота, то все они взрываются;
- для оставшихся роботов, если какой-то из них оказался между другим роботом и стеной, до которой светит луч его лазера, то этот робот взрывается. Взрывы от лучей лазера происходят одновременно: тем не менее, даже если какой-то робот взорвался, луч его лазера успевает взорвать всех роботов, которые находились между ним и стеной, до которой светит его лазер.

Известно, что в начальный момент времени никакой робот не светит на другого робота лазером. Взорвавшиеся роботы не оказывают влияния на ситуацию в лабиринте в последующие секунды. Понятно, что жизнь любого робота будет в таких условиях очень недолгой. Прайм хочет выяснить, сколько времени просуществует самый долгоживущий из них. Выясните, сколько времени пройдет до того момента, как все роботы взорвутся.

Входные данные

В первой строке входного файла INPUT.TXT заданы через пробел три числа m , n и k ($1 \leq m, n, k \leq 100$). В следующих n строках содержится ровно по m символов в каждой: i -ый символ в j -ой из этих строк равен «X» (икс большое), если соответствующая клетка (i, j) занята стеной, и «.» (точка), если эта клетка пуста. Далее идут k строк, описывающие роботов. Каждая из них имеет вид (x_i, y_i, z_i) , где x_i и y_i – координаты робота ($1 \leq x_i \leq m, 1 \leq y_i \leq n$), а z_i – один из четырех символов направления: символ «U» (up) соответствует уменьшению координаты y , символ «L» (left) – уменьшению координаты x , символ «D» (down) – увеличению y , а символ «R» (right) – увеличению x . Все числа во входном файле целые.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – сколько секунд пройдет до того момента, как все роботы взорвутся.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	5 1 2 X...X 3 1 L 4 1 R	2
2	4 4 4 X..X X..X 1 3 R 3 4 U 4 2 L 2 1 D	1
3	4 5 3 1 4 R 3 5 U 4 5 U	4
4	3 6 5 .X. .X. .X.X. 1 4 R 2 5 U 3 5 U 1 6 R 3 6 L	5

```

#include <stdio.h>
#include <algorithm>
#include <vector>
#include <queue>
#include <cassert>
#include <iostream>
#include <string>

const int L = 0, R = 1, U = 2, D = 3;
const int drow[] = { 0, 0, -1, 1};
const int dcol[] = {-1, 1, 0, 0};

const int WALL = 1;
const int EMPTY = 0;

struct Robot {
    int row, col, dir, wall_row, wall_col;

    void step() {
        row += drow[dir];
        col += dcol[dir];
    }
}

```

```

void crush() {
    row = wall_row;
    col = wall_col;
}

inline bool alive() const {
    return !(row == wall_row && col == wall_col);
}

inline bool equal(const Robot& r) {
    return row == r.row && col == r.col;
}

inline bool include(const Robot& r) {
    int rmin = std::min(row, wall_row);
    int cmin = std::min(col, wall_col);
    int rmax = std::max(row, wall_row);
    int cmax = std::max(col, wall_col);
    return rmin <= r.row && r.row <= rmax && cmin <= r.col && r.col <= cmax;
}

};

int main() {
    int nCols, nRows, nRobots;
    scanf("%d %d %d", &nCols, &nRows, &nRobots);

    std::vector<std::vector<int>> field(1+nRows+1, std::vector<int>(1+nCols+1, WALL));

    for (int row = 1; row <= nRows; ++row) {
        for (int col = 1; col <= nCols; ++col) {
            char c; scanf(" %c", &c);
            field[row][col] = (c == 'X' ? WALL : EMPTY);
        }
    }

    std::vector<Robot> robots;

    for (int i = 0; i < nRobots; ++i) {
        int c, r; char dir;
        scanf("%d %d %c", &c, &r, &dir);
        int side = -1;
        switch (dir) {
            case 'L': side = L; break;
            case 'R': side = R; break;
            case 'U': side = U; break;
            case 'D': side = D; break;
        }
        assert(side >= 0);
        int nr = r, nc = c;
        while (field[nr][nc] == EMPTY) {
            nr += drow[side];
            nc += dcol[side];
        }
        robots.push_back(Robot{r,c,side,nr,nc});
    }

    int t = 0;
    while (true) {
        if (debug) printf("t = %d, nRobots = %d\n", t, nRobots);
        // 1. Если робот оказался за пределами лабиринта или в клетке, которая заполнена ст
        // Обрабатывается автоматически
        std::queue<int> queue;
        // 2. Если в какой-то клетке оказалось два или более робота, то все они взрываются;
        for (int i = 0; i < (int)robots.size(); ++i) {
            if (!robots[i].alive()) continue;
            for (int j = 0; j < (int)robots.size(); ++j) {
                if (i == j || !robots[j].alive()) continue;
                if (robots[i].equal(robots[j])) {

```



```

        queue.push(i);
    }
}
while (!queue.empty()) {
    auto id = queue.front(); queue.pop();
    if (robots[id].alive()) {
        robots[id].crush();
    }
}
// 3. Остальное
for (int i = 0; i < (int)robots.size(); ++i) {
    if (!robots[i].alive()) continue;
    for (int j = 0; j < (int)robots.size(); ++j) {
        if (i == j || !robots[j].alive()) continue;
        if (robots[j].include(robots[i])) {
            queue.push(i);
        }
    }
}
while (!queue.empty()) {
    auto id = queue.front(); queue.pop();
    if (robots[id].alive()) {
        robots[id].crush();
    }
}
// Шаг для всех роботов
bool flag = false;
for (auto& r : robots) {
    if (r.alive()) {
        flag = true;
        r.step();
    }
}
if (flag) {
    ++t;
} else {
    break;
}
}
printf("%d", t);
return 0;
}

```

ЗАДАЧА №899

Баланс скобок

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

Дана последовательность, состоящая из открывающихся и закрывающихся круглых, квадратных и фигурных скобок.

Требуется написать программу, которая определит возможность добавления в эту последовательность цифр и знаков арифметических действий таким образом, чтобы получилось правильное скобочное выражение.

Входные данные

Входной файл INPUT.TXT содержит не менее 1 и не более 10 строк. В каждой строке записана одна последовательность скобок. Длина последовательности от 1 до 255 символов.

Выходные данные

В выходной файл OUTPUT.TXT выведите слитно символы 0 или 1. Их общее количество равно количеству введенных строк. Для каждой строки выводится 0, если из нее может получиться правильное скобочное выражение, и 1 иначе.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	(({})) ({{	01

```
#pragma GCC diagnostic ignored "-Wunused-result"
#include <stdio.h>
#include <bits/stdc++.h>

bool is_opened(char c) {
    return c == '(' || c == '[' || c == '{';
}

bool is_one_type(char c1, char c2) {
    return (c1 == '(' && c2 == ')') || (c1 == '{' && c2 == '}') || (c1 == '[' && c2 == ']')
}

int solve(const std::string& s) {
    std::stack<char> stack;
    for (auto& it : s) {
        if (is_opened(it)) {
            stack.push(it);
        } else {
            if (stack.empty()) return 1;
            char c = stack.top(); stack.pop();
            if (!is_one_type(c, it)) {
                return 1;
            }
        }
    }
    return stack.empty() ? 0 : 1;
}
```

```
}

int main() {
    for (std::string s; std::cin >> s; std::cout << solve(s));
    return 0;
}
```

ЗАДАЧА №988

Последовательность чисел

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

Рассмотрим рекуррентно заданную последовательность чисел S_i :

- 1. $S_1 = 1$
- 2. $S_i = S_{i-1} + i, S_{i-1}$ для $i \geq 2$.

Ваша задача написать программу, которая будет находить число, стоящее на k -ой позиции в последовательности S_p .

Входные данные

В первой строке входного файла INPUT.TXT записано число тестов T , ($1 \leq T \leq 10^3$). В следующих T строках записаны пары чисел k_i, p_i , ($1 \leq k_i, p_i \leq 2^{63} - 1, 1 \leq i \leq T$).

Выходные данные

В выходной файл OUTPUT.TXT выведите для каждого теста значение k_i -го элемента последовательности S_{p_i} , если такого элемента нет, выведите «No solution» без кавычек.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4	1
	1 1	No solution
	2 1	1
	1 2	2
	2 2	

```
#include <iostream>
#include <cassert>

typedef unsigned long long ull;

std::string solve(ull k, ull p) {
    ull len = 1, lvl = 1;
    while (lvl < k && len < p) {
        len = 2 * len + 1;
        ++lvl;
    }
    if (lvl == k && p > len) {
        return "No solution";
    }
    while (lvl > 0) {
        len = len / 2;
        if (p == len+1) {
            break;
        } else if (p > len+1) {
            p -= len+1;
        }
    }
}
```

```
        }
        lvl--;
    }
    assert(lvl > 0);
    return std::to_string(lvl);
}

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0); std::cout.tie(0); std::cerr.tie(0);

    int nTests; std::cin >> nTests;
    while (nTests--) {
        ull k, p;
        std::cin >> k >> p;
        std::cout << solve(p, k) << '\n';
    }
    return 0;
}
```

ЗАДАЧА №1047

Задача о назначениях

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

Одной из классических задач комбинаторной оптимизации является так называемая «задача о назначениях». Формулируется она следующим образом.

Есть n работников, пронумерованных числами от 1 до n , и n работ, также пронумерованных числами от 1 до n . Если i -ый работник выполняет j -ую работу, то ему выплачивается зарплата в размере c_{ij} денежных единиц. Необходимо найти такое назначение работников на работы (каждый работник выполняет ровно одну работу, каждая работа выполняется ровно одним работником), что суммарная зарплата работников минимальна (соответствующая сумма называется стоимостью назначения).

Напишите программу, решающую задачу о назначениях.

Входные данные

Входной файл INPUT.TXT содержит натуральное число N ($N \leq 10$). Последующие N строк содержат по N чисел каждая. При этом j -ое число $(i + 1)$ -ой строки равно c_{ij} ($1 \leq c_{ij} \leq 1000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальную возможную стоимость назначения.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 1 2 2 1	2
2	2 1 2 3 4	5

```
#pragma GCC diagnostic ignored "-Wunused-result"

#include <stdio.h>
#include <bits/stdc++.h>

int main() {
    int n; scanf("%d", &n);
    std::vector<std::vector<int>>> c(n, std::vector<int>(n, 0));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            scanf("%d", &c[i][j]);
        }
    }
    std::vector<int> perm(n); for (int i = 0; i < n; ++i) perm[i] = i;

    std::function<int()> cur_sum = [&]() {
        int answ = 0;
        for (int i = 0; i < n; ++i) {
```

```
        answ += c[i][perm[i]];
    }
    return answ;
};
int min_sum = (int)1e9+1;
do {
    min_sum = std::min(min_sum, cur_sum());
} while (std::next_permutation(perm.begin(), perm.end()));
printf("%d\n", min_sum);
return 0;
}
```

ЗАДАЧА №1250

Волна

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

Поле размером N×M клеток заполнено целыми числами. Требуется найти на поле клетку, из которой волна, запущенная не более чем на K итераций, покроет площадь с максимальной суммой расположенных на ней чисел.

Пример распространения волны для поля размером 5 x 4. Волна запущена из клетки (3,3) и была остановлена после трех итераций. Белые клетки – клетки, не покрытые волной, серые и черные – клетки, покрытые волной. Клетки, покрытые волной на последней итерации, отмечены серым цветом.



Входные данные

Первая строка входного файла INPUT.TXT содержит натуральные числа N, M и K ($N, M \leq 15, K \leq N + M$). Следующие N строк содержат по M чисел, каждое из которых не превосходит 10^4 по абсолютной величине.

Выходные данные

В выходной файл OUTPUT.TXT выведите четыре числа R, C, P и S, где R – номер строки, C – номер столбца, из которых следует запустить волну, P – количество итераций распространения волны, S – максимальная сумма чисел, покрытых волной. Если существует несколько вариантов ответа, вывести тот, в котором число P минимально. Если таких вариантов несколько, вывести вариант, в котором число R минимально, если и таких несколько, вывести вариант, в котором число C минимально.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 4 3 1 2 3 4 1 6 7 8 9 10 11 12 0 0 0 0 2 0 0 1	3 3 3 66

```
#include <stdio.h>
#include <algorithm>
```



```

#include <cassert>
#include <vector>
#include <queue>

typedef std::vector<int> Vector;
typedef std::vector<Vector> Matrix;

struct Record {
    int row, col, side, sum;
};

bool operator>(const Record& a, const Record& b) {
    return a.sum > b.sum || (a.sum == b.sum && (a.side < b.side || (a.side == b.side && (a.
}

bool operator<(const Record& a, const Record& b) {
    return b > a;
}

Record bfs(const int sr, const int sc, const int limit, const Matrix& arr) {
    const int nRows = (int)arr.size();
    const int nCols = (int)arr[0].size();

    std::queue<int> queue;
    queue.push(sr*64+sc);

    Matrix dist(nRows, Vector(nCols, -1));
    dist[sr][sc] = 1;

    std::vector<int> sum(1+limit, 0);
    while (!queue.empty()) {
        const int r = queue.front() / 64;
        const int c = queue.front() % 64;
        queue.pop();
        assert(dist[r][c] > 0);
        sum[dist[r][c]] += arr[r][c];
        if (dist[r][c] == limit) {
            continue;
        }
        for (int dr = -1; dr <= 1; ++dr) {
            for (int dc = -1; dc <= 1; ++dc) {
                if (dr * dr + dc * dc == 1) {
                    const int nr = r+dr;
                    const int nc = c+dc;
                    if (nr < 0 || nc < 0 || nr >= nRows || nc >= nCols) {
                        continue;
                    }
                    if (dist[nr][nc] == -1) {
                        dist[nr][nc] = dist[r][c] + 1;
                        queue.push(nr*64+nc);
                    }
                }
            }
        }
    }

    Record best{-1, -1, 0, int(-1e9)};
    int s = 0;
    for (int i = 1; i <= limit; ++i) {
        s += sum[i];
        best = std::max(best, Record{sr+1, sc+1, i, s});
    }
    return best;
}

int main() {
    int nRows, nCols, limit;
    scanf("%d %d %d", &nRows, &nCols, &limit);
    Matrix arr(nRows, Vector(nCols, 0));
    for (auto& row : arr) {

```

```

        for (auto& it : row) {
            scanf("%d", &it);
        }
    }
    Record best{-1,-1,0,int(-1e9)};
    for (int row = 0; row < nRows; ++row) {
        for (int col = 0; col < nCols; ++col) {
            best = std::max(best, bfs(row, col, limit, arr));
        }
    }
    assert(best.row != -1);
    printf("%d %d %d %d", best.row, best.col, best.side, best.sum);
    return 0;
}

```

ЗАДАЧА №1325

Бросание кубика

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

Кубик, грани которого помечены цифрами от 1 до 6, бросают N раз. Требуется найти вероятность того, что сумма выпавших чисел будет равна Q.

Входные данные

Входной файл INPUT.TXT содержит натуральные числа N и Q ($N \leq 500$, $Q \leq 3000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное вещественное число, которое должно отличаться от истинного значения не более чем на 0.01 истинного значения.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 6	0.16666
2	1 7	0
3	4 14	0.11265
4	100 100	1.53E-78

```
#include <iostream>
#include <iomanip>
#include <vector>

typedef long double Real;

int main() {
    int n, sum; std::cin >> n >> sum;
    if (sum < n || sum > 6 * n) {
        std::cout << "0";
        return 0;
    }
    std::vector<Real> curr(1+sum), next(1+sum);
    curr[0] = 1;
    int limit = 0;
    for (int i = 1; i <= n; ++i) {
        limit = std::min(limit+6, sum);
        for (int s = 0; s <= limit; ++s) {
            Real value = 0;
            for (int score = 1; score <= 6; ++score) {
                if (s - score >= 0) {
                    value += curr[s-score];
                }
            }
            next[s] = value / 6;
        }
        curr.swap(next);
        for (int s = 0; s <= limit; ++s) {
            next[s] = 0;
        }
    }
```

```
    }  
    std::cout << curr[sum];  
    return 0;  
}
```

ЗАДАЧА №1472

Системы счисления

(Время: 1 сек. Память: 16 Мб Сложность: 42%)

Вам необходимо вывести все основания систем счисления, в которых запись числа A оканчивается на десятичную запись числа B .

Входные данные

Входной файл INPUT.TXT содержит целые числа A и B ($1 \leq A \leq 10^9$, $10 \leq B < 100$), записанные в десятичной системе счисления.

Выходные данные

В выходной файл OUTPUT.TXT в произвольном порядке выведите все основания систем счисления, в которых запись числа A оканчивается на десятичную запись числа B . Если таких систем нет, выведите -1 .

Пример

№	INPUT.TXT	OUTPUT.TXT
1	71 13	4 68

```
#include <iostream>
#include <vector>

int main() {
    int number, need;
    std::cin >> number >> need;
    std::vector<int> answer;
    int a = need % 10, b = need / 10;
    // number = a + b * base + base^2*(...)
    for (int base = 2; a + b * base + base * base <= number; ++base) {
        if ((number - a - b * base) % (base*base) == 0 && a < base && b < base) {
            answer.push_back(base);
        }
    }
    int base = (number - a) / b;
    if (number == a + b * base && a < base && b < base) {
        answer.push_back(base);
    }
    if (answer.empty()) {
        answer.push_back(-1);
    }
    for (auto& it : answer) {
        printf("%d ", it);
    }
    return 0;
}
```

ЗАДАЧА №525

Сумма степеней двойки

(Время: 1 сек. Память: 16 Мб Сложность: 43%)

Любое натуральное число можно представить в виде суммы натуральных слагаемых, каждое из которых является степенью числа 2. Суммы, различающиеся лишь порядком слагаемых, считаются одинаковыми. Например, для числа 7 таких представлений 6 (4+2+1, 4+1+1+1, 2+2+2+1, 2+2+1+1+1, 2+1+1+1+1+1, 1+1+1+1+1+1+1).

Требуется написать программу, которая найдет количество способов такого представления заданного числа N.

Входные данные

Входной файл INPUT.TXT содержит число N ($1 \leq N \leq 1000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – найденное количество способов представления числа N.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4	4
2	7	6

```
#include <iostream>
#include <functional>

int main() {
    int n; std::cin >> n;
    static int answ[1001][20];
    for (int i = 0; i < 1001; ++i) {
        for (int j = 0; j < 20; ++j) {
            answ[i][j] = (i == 0 || j == 0) ? 1 : -1;
        }
    }
    std::function<int(int, int)> get=[&](const int number, const int pow) {
        if (answ[number][pow] == -1) {
            int count = 0;
            for (int p = pow; p >= 0; --p) {
                if (number - (1 << p) >= 0) {
                    auto add = get(number - (1 << p), p);
                    count += add;
                }
            }
            answ[number][pow] = count;
        }
        return answ[number][pow];
    };
    std::cout << get(n, 19);
    return 0;
}
```

ЗАДАЧА №569

Зарплата - 2

(Время: 1 сек. Память: 16 Мб Сложность: 43%)

В одном государственном учреждении работают n сотрудников. Их фактические зарплаты равны d_i , но по закону всем сотрудникам положено платить одинаково, поэтому им всем официально выплачивают среднюю зарплату d , такую, чтобы общая сумма оставалась такой же, а потом сотрудники сами перераспределяют полученные деньги.

Короче, творится полный бардак. И чтобы этот бардак уменьшить, Самый Главный Начальник решил использовать недавно принятый закон «о материальной помощи», который позволяет любому сотруднику часть своей зарплаты (целое число рублей от 1 до $d-1$) передавать другому сотруднику в виде материальной поддержки. Однако по закону сотрудник, таким образом, может «помогать» только одному «малоимущему».

Например, если Петя и Вася получают по 100 рублей и Петя напишет заявление на передачу 30% своей зарплаты Васе, то Петя будет получать 70 рублей, а Вася - 130.

Теперь Самый Главный Начальник хочет узнать: кто, кому и сколько должен передавать, чтобы в результате все получали ровно столько, сколько нужно. Помогите ему это сделать.

Входные данные

Входной файл INPUT.TXT содержит целые числа n и d ($1 \leq n \leq 10^5$, $1 \leq d \leq 10^9$), и далее n целых чисел d_i ($1 \leq d_i \leq 10^9$). Сумма всех d_i равна $n \cdot d$.

Выходные данные

В выходной файл OUTPUT.TXT выведите n пар целых чисел a_i и b_i , означающих, что сотрудник i передает сотруднику a_i часть своей зарплаты в размере b_i рублей. Если сотрудник i ничего никому не передает, выведите вместо a_i и b_i два нуля.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 200 100 300 200	2 100 0 0 0 0
2	3 200 10 300 290	3 190 0 0 2 100

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <tuple>
#include <cassert>
```

```

struct Pair {
    int cost, id;
};

int main() {
    int n, d;
    scanf("%d %d", &n, &d);

    std::vector<Pair> arr;
    for (int i = 1; i <= n; ++i) {
        int cost; scanf("%d", &cost);
        arr.push_back(Pair{cost, i});
    }

    std::stable_sort(arr.begin(), arr.end(), [](const Pair& a, const Pair& b) {
        return a.cost < b.cost;
    });

    std::vector<int> curr(n, d);

    std::vector<Pair> answer(1+n, Pair{0,0});

    int low = 0, high = n-1;
    while (low < high) {
        int diff = curr[low] - arr[low].cost;
        if (diff == 0) {
            for (int i = 0; i < n; ++i) {
                assert(curr[i] == arr[i].cost);
            }
            break;
        }
        answer[arr[low].id] = Pair{diff, arr[high].id};
        curr[low] -= diff;
        curr[high] += diff;
        while (curr[high] > arr[high].cost) {
            int diff = curr[high] - arr[high].cost;
            curr[high] -= diff;
            curr[high-1] += diff;
            answer[arr[high].id] = Pair{diff, arr[high-1].id};
            --high;
        }
        ++low;
    }

    for (int i = 1; i <= n; ++i) {
        const auto& t = answer[i];
        printf("%d %d\n", t.id, t.cost);
    }

    return 0;
}

```


ЗАДАЧА №570

Квадрат

(Время: 1 сек. Память: 16 Мб Сложность: 43%)

На сайте сотового оператора BeepLine сделали защиту от роботов, рассылающих SMS-сообщения: прежде, чем отправить SMS, пользователь должен написать, какую фигуру он видит в специальном окошке: квадрат или круг. Причем, для усиления защиты, в рисунок внесены небольшие помехи.

Коле срочно нужно разослать всем друзьям сообщение, поэтому он просит Вас написать программу, распознающую изображение.

Экспериментально установлено, что система рисует квадрат с помехами следующим образом: сначала на белом фоне рисуется черный квадрат $k \times k$ клеток ($k \geq 3$), затем некоторые клетки на границе квадрата (на рисунке обозначены цифрой 1) закрашиваются белым, а некоторые клетки (если таковые существуют), граничащие с квадратом (на рисунке обозначены цифрой 2), закрашиваются черным.

	2	2	2	2	2	2	
	2	1	1	1	1	2	
	2	1			1	2	
	2	1			1	2	
	2	1	1	1	1	2	
	2	2	2	2	2	2	

Например, квадрат 4×4 после нанесения помех может выглядеть так:

Входные данные

Первая строка входного файла INPUT.TXT содержит целые числа n и m - размеры экрана ($1 \leq n, m \leq 1000$). Следующие n строк, по m символов в каждой, содержат описание картинки. Черные клетки обозначены символом «*», а белые - символом «.».

Выходные данные

В выходной файл OUTPUT.TXT выведите «SQUARE», если заданная картинка может быть квадратом после преобразований, описанных в условии, иначе выведите слово «CIRCLE».

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	<pre> 10 10***..... ..***..... ..***..... </pre>	SQUARE
2	<pre> 10 10 *..... ..*****... ..*****.. ..*****... ..*****... ..*****... ..* * *... ..*..... </pre>	SQUARE
3	<pre> 10 10****..... ..*****... ..*****... *****.* *****.. *****.. ..*****... ..****.....*..... </pre>	CIRCLE
4	<pre> 3 3 </pre>	CIRCLE

```
#pragma GCC diagnostic ignored "-Wunused-result"
#include <stdio.h>
#include <bits/stdc++.h>

int main() {
    int nRows, nCols;
    scanf("%d %d", &nRows, &nCols);
    std::vector<std::vector<char>> field(nRows, std::vector<char>(nCols));
    int min_row = nRows;
```

```

int min_col = nCols;
int max_row = -1;
int max_col = -1;
for (int row = 0; row < nRows; ++row) {
    char buf[1001];
    scanf("%1000s", buf);
    for (int col = 0; col < nCols; ++col) {
        char c = buf[col];
        field[row][col] = c;
        if (c == '*') {
            min_row = std::min(min_row, row);
            min_col = std::min(min_col, col);
            max_row = std::max(max_row, row);
            max_col = std::max(max_col, col);
        }
    }
}

int w = max_col - min_col + 1;
int h = max_row - min_row + 1;

if (max_col == -1 || std::abs(w-h) > 4) {
    std::cout << "CIRCLE\n";
    return 0;
}

// maxSide[r][c] = max side of square with 4 corners: [r-side+1][c-side+1], [r-side+1][
std::vector<std::vector<int>> maxSide(nRows, std::vector<int>(nCols));

for (int row = 0; row < nRows; ++row) {
    for (int col = 0; col < nCols; ++col) {
        if (field[row][col] == '.') continue;
        if (row == 0 || col == 0) {
            maxSide[row][col] = 1;
        } else {
            maxSide[row][col] = 1 + std::min({maxSide[row-1][col], maxSide[row][col-1],
        }
    }
}

for (int row = std::max(1, min_row); row <= std::min(max_row, nRows-2); ++row) {
    for (int col = std::max(1, min_col); col <= std::min(max_col, nCols-2); ++col) {
        int side = maxSide[row][col];
        if (side == 0) {
            continue;
        }

        assert(row-side+1 >= min_row && col - side + 1 >= min_col);
        assert(w >= side && h >= side);

        if (
            w - side <= 4 && h - side <= 4 &&
            row - side + 1 - min_row <= 2 && max_row - row <= 2 &&
            col - side + 1 - min_col <= 2 && max_col - col <= 2
        ) {
            std::cout << "SQUARE" << std::endl;
            return 0;
        }
    }
}

std::cout << "CIRCLE" << std::endl;
return 0;
}

```

ЗАДАЧА №681

Спички

(Время: 1 сек. Память: 16 Мб Сложность: 43%)

Какое минимальное количество спичек необходимо для того, чтобы выложить на плоскости N квадратов со стороной в одну спичку? Спички нельзя ломать и класть друг на друга. Вершинами квадратов должны быть точки, где сходятся концы спичек, а сторонами – сами спички.

Напишите программу, которая по количеству квадратов N , которые необходимо составить, находит минимальное необходимое для этого количество спичек.

Входные данные

В единственной строке входного файла INPUT.TXT записано одно целое число N ($1 \leq N \leq 10^9$).

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести одно целое число – минимальное количество спичек, требуемых для составления заданного количества квадратов.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4	12

```
/*
    Находим ближайший полный квадрат больший n, вырезаем из него лишние квадратики.
*/

#include <iostream>
#include <algorithm>
#include <cmath>

typedef long long Int;

int main() {
    Int n;
    std::cin >> n;
    Int side = (Int)std::ceil(std::sqrt(n));
    Int del = side*side-n;
    Int answ = side*side*2+2*side;
    if (del < side) {
        answ -= 2*del;
    }
    if (del >= side) {
        answ -= 2*del+1;
    }
    std::cout << answ;
    return 0;
}
```

ЗАДАЧА №719

Фотограф-псих

(Время: 1 сек. Память: 16 Мб Сложность: 43%)

Пришел как-то раз в гости к фотографу-зануде его двоюродный брат, и начали они спорить о том, кто сколько человек сфотографировал и кто больше фотографий сделал. После долгого подсчета выяснилось, что фотограф-зануда выиграл. Брату это, естественно, не понравилось, поэтому он пошел к себе ставить новый рекорд.

Для этого он пригласил к себе N человек и фотографировал их, располагая всеми различными способами вдоль одной линии. Всего получилось F фотографий.

Разумеется, сделать их за один день он не смог, так как в ближайших магазинах попросту не хватало фотопленки. Когда же он закончил работу, он приступил к рассылке извещений всем своим клиентам. Каждое извещение помещается в отдельный конверт и отправляется по адресу прописки клиента.

Помогите фотографу определить число конвертов, которое необходимо закупить, если известно число фотографий F. Известно также, что число клиентов положительное и не превышает 10^5 .

Входные данные

В единственной строке входного файла INPUT.TXT содержится натуральное число F, не превосходящее 10^{500000} .

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести одно натуральное число — количество конвертов для закупки.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2	2
2	6227020800	13

```
/*
"Фотограф-псих": факториалы, остатки от деления

Исследованный факт: пары остатков (a, b) по простым модулям 10^9+7 и 10^9+9 не совпадают
*/

#include <stdio.h>
#include <algorithm>
#include <string>

int hash(std::string s, const int mod) {
    int pow = 1, res = 0;
    for (auto it : s) {
        res = (res + 1LL * (it - '0') * pow) % mod;
        pow = 1LL * pow * 10 % mod;
    }
    return res;
}
```

```

}

int main() {
    const int mod1 = (1e9)+7, mod2 = (int)1e9+9;

    char buf[500001];
    scanf("%500000s", buf);
    std::string s(buf);
    std::reverse(s.begin(), s.end());

    const int hash1 = hash(s, mod1), hash2 = hash(s, mod2);

    int fact1 = 1, fact2 = 1;
    for (int i = 1; i <= 100000; ++i) {
        fact1 = 1LL * fact1 * i % mod1;
        fact2 = 1LL * fact2 * i % mod2;
        if (fact1 == hash1 && fact2 == hash2) {
            printf("%d", i);
            return 0;
        }
    }

    throw 1;
}

```

ЗАДАЧА №728

Закупка носков

(Время: 1 сек. Память: 16 Мб Сложность: 43%)

В одной военной части было принято революционное решение перейти от портянок к носкам. По такому случаю прапорщику Недалекому было поручено закупить n пар носков. Однако предложенная прапорщиком смета не удовлетворила начальство, и прапорщику было предложено очень-очень быстро переделать ее так, чтобы затраты были минимально возможными. Помогите бедному прапорщику составить такую смету.

Изучение рынка показало, что всего существует m различных поставщиков, которые продают носки разными пачками и по разным ценам. Пачка, содержащая a_i пар носков, продается за b_i рублей.

Разрешено покупать любое количество пачек у одного поставщика. Разрешено покупать пачки у нескольких поставщиков.

Входные данные

В первой строке входного файла INPUT.TXT содержатся числа n и m ($1 \leq n \leq 10000$, $1 \leq m \leq 100$). Далее идут m пар чисел a_i , b_i ($1 \leq a_i \leq 10000$, $1 \leq b_i \leq 10000$).

Выходные данные

Выведите в выходной файл OUTPUT.TXT минимальную сумму денег, которую нужно потратить на покупку n пар носок.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	9 2 1 1 10 8	8

```
#include <stdio.h>
#include <algorithm>
#include <vector>

struct Pair {
    int count, cost;
};

int main() {
    int need, m;
    scanf("%d %d", &need, &m);

    std::vector<Pair> pairs;
    while (m--) {
        int count, cost;
        scanf("%d %d", &count, &cost);
        pairs.push_back(Pair{count, cost});
    }
    const int INF = 1e9;
    std::vector<int> min(1+need, INF); // min[count]
```

```
min[0] = 0;
for (int i = 1; i <= need; ++i) {
    for (auto& p : pairs) {
        min[i] = std::min(min[i], min[std::max(0, i-p.count)]+p.cost);
    }
}
printf("%d", min[need]);
return 0;
}
```


ЗАДАЧА №1175

Минимумы на отрезках

(Время: 2 сек. Память: 16 Мб Сложность: 43%)

Задан числовой массив $A[1..N]$. Необходимо выполнить M операций вычисления минимального элемента на отрезке $[L, R]$.

Входные данные

Первая строка входного файла INPUT.TXT содержит число N – размер массива. Во второй строке записаны N чисел – элементы массива. Третья строка содержит число M – количество запросов минимума. Следующие M строк содержат пары чисел L и R ($L \leq R \leq N$), описывающие отрезки. Все числа во входных данных натуральные, не превосходящие 10^5 .

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса выведите значение минимума на отрезке через пробел.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 3 1 8 7 9 2 1 3 3 5	1 7

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <climits>

struct DataStruct {
    std::vector<int> arr, min;

    const int GSIZE = 258;

    DataStruct(int size = 0, int item = 0) {
        arr.assign(size, item);
        min.assign(size / GSIZE + 1, item);
    }

    void set(int p, int v) {
        arr[p] = v;
        const int g = p / GSIZE;
        min[g] = v;
        const int begin = g * GSIZE;
        const int after = std::min(begin + GSIZE, (int)arr.size());
        for (int i = begin; i < after; ++i) {
            min[g] = std::min(arr[i], min[g]);
        }
    }
}
```

```

int get(int l, int r) {
    const int gl = l / GSIZE;
    const int gr = r / GSIZE;
    int answ = INT_MAX;
    if (gl == gr) {
        for (int i = l; i <= r; ++i) {
            answ = std::min(answ, arr[i]);
        }
    } else {
        for (int i = l, after = (gl+1)*GSIZE; i < after; ++i) {
            answ = std::min(answ, arr[i]);
        }
        for (int g = gl+1; g < gr; ++g) {
            answ = std::min(answ, min[g]);
        }
        for (int i = gr * GSIZE; i <= r; ++i) {
            answ = std::min(answ, arr[i]);
        }
    }
    return answ;
}

};

int main() {
    int n;
    scanf("%d", &n);
    DataStruct ds(n, 0);
    for (int i = 0; i < n; ++i) {
        int value;
        scanf("%d", &value);
        ds.set(i, value);
    }
    int q;
    scanf("%d", &q);
    while (q--) {
        int l, r;
        scanf("%d %d", &l, &r);
        --l, --r;
        printf("%d ", ds.get(l, r));
    }
    return 0;
}

```

ЗАДАЧА №350

Перестановки

(Время: 1 сек. Память: 16 Мб Сложность: 44%)

Дана строка, состоящая из N попарно различных символов. Требуется вывести все перестановки символов данной строки.

Входные данные

Входной файл INPUT.TXT содержит строку, состоящую из N символов ($1 \leq N \leq 8$), символы - буквы английского алфавита и цифры.

Выходные данные

В выходной файл OUTPUT.TXT выведите в каждой строке по одной перестановке. Перестановки можно выводить в любом порядке. Повторений и строк, не являющихся перестановками исходной, быть не должно.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	AB	AB BA
2	IOX	XOI OIX IXO XIO OXI IOX

```
#pragma GCC diagnostic ignored "-Wunused-result"
```

```
#include <stdio.h>
```

```
#include <bits/stdc++.h>
```

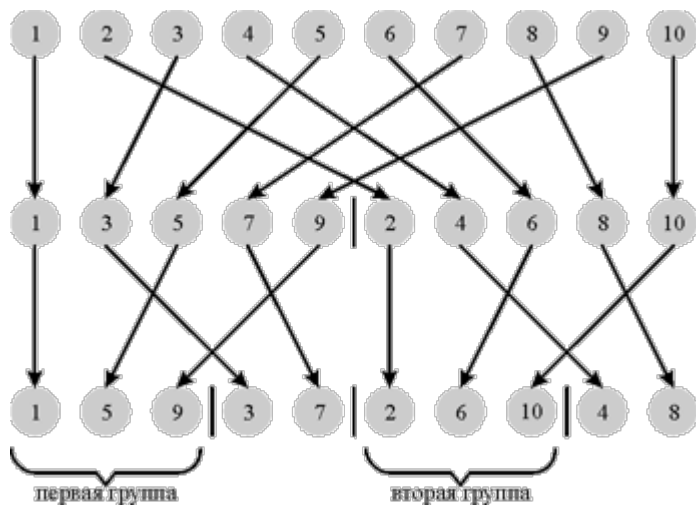
```
int main() {  
    std::string s; std::cin >> s; std::sort(s.begin(), s.end());  
    do {  
        printf("%s\n", s.c_str());  
    } while (std::next_permutation(s.begin(), s.end()));  
    return 0;  
}
```

ЗАДАЧА №923

Легион

(Время: 1 сек. Память: 16 Мб Сложность: 44%)

Легион – основная организационная единица в армии Древнего Рима. В разное время легионы имели разную численность и различное построение. Самым простым построением была шеренга. Чтобы из N солдат легиона, выстроенных в шеренгу, выбрать троих в разведку, выполнялись следующие операции: если солдат в шеренге больше трех, то шеренга разбивалась на две, одна из которых состоит из солдат, стоящих на четных позициях, а вторая – из стоящих на нечетных позициях. Для всех полученных шеренг эта процедура повторялась до тех пор, пока в каждой из них не останется не более трех солдат. Если солдат осталось трое, то данную группу можно послать в разведку.



Требуется определить, сколько групп по три человека может быть сформировано из исходной шеренги.

Входные данные

Входной файл INPUT.TXT содержит единственное целое число N - количество солдат в шеренге ($0 \leq N \leq 10^{18}$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число – количество групп по три человека, сформированных из исходной шеренги.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	10	2
2	4	0

```
#include <stdio.h>
#include <bits/stdc++.h>

typedef long long Int;
```

```
std::map<Int, Int> answer;

Int solve(Int n) {
    if (n < 3) {
        return 0;
    }
    if (n == 3) {
        return 1;
    }
    if (answer.find(n) == answer.end()) {
        answer[n] = solve((n+1) / 2) + solve(n / 2);
    }
    return answer[n];
}

int main() {
    Int n;
    std::cin >> n;
    std::cout << solve(n) << std::endl;
    return 0;
}
```

ЗАДАЧА №960

Три буквы

(Время: 1 сек. Память: 16 Мб Сложность: 44%)

Напомним, что строка $B = b_1 b_2 b_3 \dots b_m$, является подпоследовательностью строки $A = a_1 a_2 a_3 \dots a_n$, если существует строго возрастающая последовательность $\{i_1, i_2, i_3, \dots, i_m\}$ индексов A , такая, что для всех j от 1 до m выполняется $A_{i_j} = B_j$. Например, $B = \text{''aba''}$ является подпоследовательностью строки $A = \text{''abacaba''}$. Последовательность индексов в этом случае может быть такой: $\{1, 2, 3\}$.

Пусть Вам дана строка S , состоящая только из маленьких букв английского алфавита. Ваша задача заключается в том, чтобы посчитать количество ее подпоследовательностей ''abc'' .

Входные данные

Входной файл INPUT.TXT содержит строку S , длиной не более 10^5 символов.

Выходные данные

В выходной файл OUTPUT.TXT выведите ответ на задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	abc	1
2	ab	0

```
/*
    "Три буквы": динамическое программирование, префикс-суммы, суффикс-суммы, O(n)
*/
```

```
#include <stdio.h>
#include <vector>
#include <string>
#include <iostream>

int main() {
    char buf[1000000+1];
    scanf("%1000000s", buf);
    std::string s(buf);
    int n = (int)s.size();
    s = "?" + s + "?";
    std::vector<int> pref(n+2, 0), suff(n+2, 0);
    for (int i = 1; i <= n; ++i) {
        pref[i] = pref[i-1] + (s[i] == 'a');
    }
    for (int i = n; i >= 1; --i) {
        suff[i] = suff[i+1] + (s[i] == 'c');
    }
    long long answ = 0;
    for (int i = 1; i <= n; ++i) {
        if (s[i] == 'b') {
            answ += 1LL * pref[i-1] * suff[i+1];
        }
    }
    printf("%lld\n", answ);
}
```

```
    }  
}  
std::cout << answ;  
return 0;  
}
```

ЗАДАЧА №1412

Взрывчатка

(Время: 1 сек. Память: 16 Мб Сложность: 44%)

На территории базы противника расположены N арсеналов, которые нужно уничтожить. Вы, Агент-070, имеете запас взрывчатки, достаточный для того, чтобы взорвать все N арсеналов по одному. Но при этом велик риск обнаружения.

Однако Вам известно, что при взрыве каждого арсенала создаётся взрывная волна, которая может достичь других арсеналов и спровоцировать их взрыв. Эти арсеналы тоже будут создавать свои взрывные волны. В Центре вам выдали таблицу, в которой для каждого арсенала указано, какие другие арсеналы попадут под его взрывную волну.

Так как на минирование одного арсенала требуется время, а оно очень дорого, Вам нужно заминировать как можно меньше арсеналов, чтобы, взорвав их, гарантированно уничтожить все арсеналы на базе.

Входные данные

В первой строке входного файла INPUT.TXT содержится число N – количество арсеналов на территории базы ($1 \leq N \leq 100$).

Далее дана таблица отношений: в N строках содержится по N символов, j -ый символ i -ой строки равен «1», если при взрыве i -го арсенала j -ый попадёт под его взрывную волну и «0» иначе.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – минимальное количество арсеналов, которое требуется заминировать для уничтожения всех арсеналов на базе.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 000 100 100	2
2	3 010 001 000	1

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>

int main() {
    int n; scanf("%d", &n);
    std::vector<std::vector<int>> dist(n, std::vector<int>(n));
    for (int i = 0; i < n; ++i) {
```



```

    char buf[101];
    scanf("%100s", buf);
    for (int j = 0; j < n; ++j) {
        dist[i][j] = (buf[j] == '1');
    }
    dist[i][i] = 1;
}
for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (dist[i][k] && dist[k][j]) {
                dist[i][j] = 1;
            }
        }
    }
}
std::vector<bool> used(n, false);
int count = 0;
while (true) {
    int nUsed = 0;
    for (int i = 0; i < n; ++i) {
        nUsed += used[i];
    }
    if (nUsed == n) {
        break;
    }
    std::vector<int> rank(n, 0);
    for (int vert = 0; vert < n; ++vert) {
        if (!used[vert]) {
            for (int i = 0; i < n; ++i) {
                if (dist[i][vert]) {
                    ++rank[i];
                }
            }
        }
    }
    int v = (int)(std::max_element(rank.begin(), rank.end()) - rank.begin());
    for (int j = 0; j < n; ++j) {
        if (dist[v][j]) {
            used[j] = true;
        }
    }
    ++count;
}
printf("%d", count);
return 0;
}

```

ЗАДАЧА №536

Числа - 2

(Время: 1 сек. Память: 16 Мб Сложность: 45%)

Решая задачу по информатике, Вова в очередной раз допустил ошибку. Он снова вывел в выходной файл числа, забыв разделить их пробелами. Увидев полученный результат, Вова сначала огорчился, а потом задумался над следующим вопросом: сколько существует различных последовательностей неотрицательных целых чисел, таких что, если выписать их без пробелов, то получится тот же результат, что и у него. Он вспомнил также, что его программа смогла вывести не произвольные числа, а только те, что не превосходят S и не имеют ведущих нулей.

Чтобы ответить на поставленный вопрос, Вова решил написать программу, которая позволит ему найти число различных последовательностей неотрицательных целых чисел, в каждой из которых любое число не превосходит S . Он понимал, что такое число могло быть достаточно большим, поэтому ограничился поиском только последних K цифр этого числа.

Требуется написать программу, которая покажет Вове, как можно правильно решить поставленную им задачу.

Входные данные

Первая строка входного файла INPUT.TXT содержит три целых числа – N , S и K ($1 \leq N \leq 50000$, $1 \leq S \leq 10^8$, $1 \leq K \leq 18$). Во второй строке этого файла содержится результат работы Вовой программы, состоящий из N цифр.

Выходные данные

В выходной файл OUTPUT.TXT выведите последние K цифр искомого количества последовательностей без ведущих нулей.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 11 2 111	3
2	19 9 1 0123456789876543210	1
3	1 8 3 9	0

```
#pragma GCC diagnostic ignored "-Wunused-result"
#include <iostream>
#include <vector>
#include <string>
```

```
typedef long long ll;
```

```
int main() {
    int n, limit, pow;
    std::cin >> n >> limit >> pow;
    ll mod = 1;
```

```

for (int p = 0; p < pow; ++p) {
    mod *= 10;
}
std::string s;
std::cin >> s;
s = '?' + s;
std::vector<ll> answer(1+n, 0);
answer[0] = 1;
answer[1] = (s[1] - '0' <= limit);
for (int curr = 2; curr <= n; ++curr) {
    ll number = 0, pow = 1;
    for (int prev = curr; curr - prev < 9 && prev >= 1; --prev, pow *= 10) {
        number += pow * (s[prev] - '0');
        if (number > limit) break;
        if (s[prev] == '0' && prev < curr) continue;
        answer[curr] = (answer[curr]+answer[prev-1]) % mod;
    }
}
std::cout << answer[n] << std::endl;
return 0;
}

```

ЗАДАЧА №552

Зоопарк

(Время: 0,2 сек. Память: 16 Мб Сложность: 45%)

В городском зоопарке содержатся животные n разных видов. Для участия в международной выставке «Три твари» зоопарк должен представить трех животных различных видов.

Требуется написать программу, которая вычислит число способов выбрать трех животных для участия в выставке.

Например, если в зоопарке два медведя, тигр, лев и пингвин, то есть семь способов выбрать трех животных:

1. первый медведь, тигр и лев;
2. первый медведь, тигр и пингвин;
3. первый медведь, лев и пингвин;
4. второй медведь, тигр и лев;
5. второй медведь, тигр и пингвин;
6. второй медведь, лев и пингвин;
7. тигр, лев и пингвин.

Входные данные

Входной текстовый файл INPUT.TXT содержит в первой строке натуральное число n – количество видов животных в городском зоопарке ($1 \leq n \leq 1000$). Во второй строке через пробел записаны n натуральных чисел – количество животных соответствующего вида. Число животных каждого вида не превышает 1000.

Выходные данные

Выходной текстовый файл OUTPUT.TXT должен содержать одно число – количество способов выбрать трех животных для международной выставки.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 2 1 1 1	7
2	3 100 100 100	1000000

```
#pragma GCC diagnostic ignored "-Wunused-result"
#include <stdio.h>
#include <bits/stdc++.h>
```

```
typedef long long ll;
```

```
int main() {
    int n;
    scanf("%d", &n);
```

```

std::vector<int> a(n, 0);
for (auto& it : a) scanf("%d", &it);

// s1[i] = a[i] + a[i+1] + a[i+2] + ... + a[n-1]
std::vector<ll> s1(n+1, 0);
for (int i = n-1; i >= 0; --i) s1[i] = s1[i+1] + a[i];

// s2[i] = a[i] * (a[i+1] + a[i+2] + ... + a[n-1]) + a[i+1] * (a[i+2] + ... ) + ...
std::vector<ll> s2(n+1, 0);
for (int i = n-1; i >= 0; --i) {
    s2[i] = a[i] * s1[i+1] + s2[i+1];
}

// Sum all a[i] * a[j] * a[k] for 0 <= i < j < k < n
ll answer = 0;
for (int i = 0; i < n-2; ++i) {
    answer += a[i] * s2[i+1];
}
std::cout << answer << std::endl;
return 0;
}

```

ЗАДАЧА №594

Треугольник - 2

(Время: 2 сек. Память: 32 Мб Сложность: 45%)

После окончания многолетней войны короли-победители решили разделить между собой захваченную территорию. Для того, чтобы избежать лишних споров, короли решили делить территорию следующим образом.

Каждому королю был предоставлен набор, состоящий из n отрезков L_1, \dots, L_n . Королям разрешается присоединять к их королевствам территорию, имеющую форму треугольника, составленного из имеющихся отрезков.

Помогите королю вашего королевства максимизировать площадь треугольника.

Входные данные

На первой строке входного файла INPUT.TXT задано число n ($3 \leq n \leq 100000$) - количество отрезков в наборе. В следующих n строках заданы длины отрезков в следующем формате.

Сначала указывается целое положительное число L - длина отрезка, а затем единица измерения (m, km, mile, uin, kairi, zhang, sen).

Напомним, что: 1mile = 1609m, 1km = 1000m, 1uin = 33m, 1kairi = 1852m, 1zhang = 3m и 1sen = 38m.

Длины всех отрезков не превосходят десяти километров. Гарантируется, что из заданных отрезков можно выбрать три отрезка так, что из них можно составить треугольник.

Выходные данные

В выходной файл OUTPUT.TXT выведите максимальную площадь (в таланг вахах) с точностью 10^{-6} и номера трех отрезков в любом порядке (если ответов несколько, выведите любой). Помните, что один таланг вах равен четырем квадратным метрам.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	7 1mile 1km 100m 20uin 1kairi 300zhang 40sen	291590.0233624191 7 1 5

```
#pragma GCC diagnostic ignored "-Wunused-result"

#include <stdio.h>
#include <bits/stdc++.h>

typedef long double ld;
```

```

struct Pair { int len, id; };

int get_len(int len, std::string suf) {
    if (suf == "mile") return len * 1609;
    if (suf == "km") return len * 1000;
    if (suf == "uin") return 33 * len;
    if (suf == "kairi") return 1852 * len;
    if (suf == "zhang") return 3 * len;
    if (suf == "sen") return 38 * len;
    assert(suf == "m");
    return len;
}

int main() {
    int n; scanf("%d", &n);
    std::vector<Pair> arr;
    for (int i = 0; i < n; ++i) {
        int len;
        char buf[10];
        scanf("%d%s", &len, buf);
        arr.push_back(Pair{get_len(len, buf), i+1});
    }

    std::sort(arr.begin(), arr.end(), [](const Pair& left, const Pair& right){
        return left.len > right.len || (left.len == right.len && left.id < right.id);
    });

    long long best_s = -1; int best_i = -1;
    for (int i = 1; i+1 < n; ++i) {
        if (arr[i-1].len < arr[i].len + arr[i+1].len) {
            int a = arr[i-1].len, b = arr[i].len, c = arr[i+1].len;
            long long temp_s = 1LL * (a+b+c)*(a+b-c)*(a+c-b)*(b+c-a);
            if (temp_s > best_s) {
                best_s = temp_s, best_i = i;
            }
        }
    }
    assert(best_s != -1 && best_i != -1);
    int id1 = arr[best_i-1].id, id2 = arr[best_i].id, id3 = arr[best_i+1].id;
    std::cout << std::fixed << std::setprecision(6) << std::sqrt(1d(best_s)) / 16 << std::e
    std::cout << id1 << " " << id2 << " " << id3 << std::endl;

    return 0;
}

```

ЗАДАЧА №713

Булева функция

(Время: 1 сек. Память: 16 Мб Сложность: 45%)

Недавно на уроке информатики ученики одного из классов изучили булевы функции. Напомним, что булева функция f сопоставляет значениям двух булевых аргументов, каждый из которых может быть равен 0 или 1, третье булево значение, называемое результатом. Для учеников, которые выразили желание более подробно изучать эту тему, учительница информатики на дополнительном уроке ввела в рассмотрение понятие цепного вычисления булевой функции f .

Если задана булева функция f и набор из N булевых значений a_1, a_2, \dots, a_N , то результат цепного вычисления этой булевой функции определяется следующим образом:

- если $N = 1$, то он равен a_1 ;
- если $N > 1$, то он равен результату цепного вычисления булевой функции f для набора из $(N-1)$ булевого значения $f(a_1, a_2), a_3, \dots, a_N$, который получается путем замены первых двух булевых значений в наборе из N булевых значений на единственное булево значение – результат вычисления функции f от a_1 и a_2 .

Например, если изначально задано три булевых значения: $a_1 = 0, a_2 = 1, a_3 = 0$, а функция f – ИЛИ (OR), то после первого шага получается два булевых значения – $(0 \text{ OR } 1)$ и 0 , то есть, 1 и 0 . После второго (и последнего) шага получается результат цепного вычисления, равный 1 , так как $1 \text{ OR } 0 = 1$.

В конце дополнительного урока учительница информатики написала на доске булеву функцию f и попросила одного из учеников выбрать такие N булевых значений a_i , чтобы результат цепного вычисления этой функции был равен единице. Более того, она попросила найти такой набор булевых значений, в котором число единиц было бы как можно большим.

Требуется написать программу, которая решала бы поставленную учительницей задачу.

Входные данные

Первая строка входного файла INPUT.TXT содержит одно натуральное число N ($2 \leq N \leq 100\,000$). Вторая строка содержит описание булевой функции в виде четырех чисел, каждое из которых – ноль или единица.

Первое из них есть результат вычисления функции в случае, если оба аргумента – нули, второе – результат в случае, если первый аргумент – ноль, второй – единица, третье – результат в случае, если первый аргумент – единица, второй – ноль, а четвертый – в случае, если оба аргумента – единицы.

Выходные данные

В выходной файл OUTPUT.TXT необходимо вывести строку из N символов, определяющих искомый набор булевых значений a_i с максимально возможным числом единиц. Если ответов несколько, требуется вывести любой из них. Если такого набора не существует, выведите в выходной файл фразу «No solution».

Примеры

Пояснение

В первом примере процесс вычисления цепного значения булевой функции f происходит следующим образом: $1011 \rightarrow 111 \rightarrow 01 \rightarrow 1$.

Во втором примере вычисление цепного значения булевой функции f происходит следующим образом: $11111 \rightarrow 0111 \rightarrow 111 \rightarrow 01 \rightarrow 1$.

В третьем примере получить цепное значение булевой функции f , равное 1, невозможно.

№	INPUT.TXT	OUTPUT.TXT
1	4 0110	1011
2	5 0100	11111
3	6 0000	No solution

```
/*
    "Булева функция": динамическое программирование по профилю,  $O(n)$ 
*/

#include <stdio.h>
#include <algorithm>
#include <string>
#include <vector>

int main() {
    int n;
    scanf("%d", &n);

    std::vector<int> f(4);
    for (int i = 0; i < 4; ++i) {
        char c;
        scanf(" %c", &c);
        f[i] = (c - '0');
    }

    std::vector<std::vector<int>> max1(2, std::vector<int>(n, -1));
    std::vector<std::vector<int>> from(2, std::vector<int>(n, -1));
    // Инициализация:
    max1[0][0] = 0;
    max1[1][0] = 1;
    // Динамика на префиксе:
    for (int i = 1; i < n; ++i) {
        for (int mask = 0; mask < 4; ++mask) {
            int prev = (mask >> 1) & 1;
            int curr = (mask >> 0) & 1;
            if (max1[prev][i-1] == -1) {
                continue;
            }
            if (max1[f[mask]][i] < max1[prev][i-1] + (curr == 1)) {
                max1[f[mask]][i] = max1[prev][i-1] + (curr == 1);
                from[f[mask]][i] = mask;
            }
        }
    }

    if (max1[1][n-1] == -1) {
        printf("No solution");
        return 0;
    }

    // Формирование ответа:
    std::string answ;
```

```
int value = 1;
for (int i = n-1; i > 0; --i) {
    answ.push_back(((from[value][i] >> 0) & 1) + '0');
    value = (from[value][i] >> 1) & 1;
}
answ.push_back(value+'0');
std::reverse(answ.begin(), answ.end());

// Вывод ответа:
printf("%s", answ.c_str());

return 0;
}
```

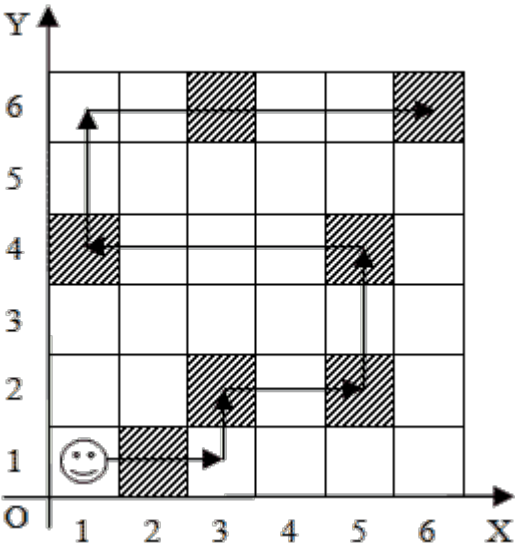
ЗАДАЧА №823

Мусорщик

(Время: 1 сек. Память: 16 Мб Сложность: 45%)

Всем известно, что работа уборщицы становится все менее престижной, чем работа программиста. Поэтому, все больше становится программистов и все меньше уборщиц. И скоро, возможно, совсем некому будет делать уборку помещений, а чистота – она всегда актуальна и важна для работы, например, для тех же программистов.

Сотрудники одной из фирм разработали специальную машину «Мусорщик-001», которая предназначена для уборки прямоугольных пустых помещений. Машина не совершенна и может пока двигаться на 1 метр только влево, вправо и вперед (вдоль оси OY). Каждое помещение можно разбить на квадратные сектора со стороной в 1 метр и обозначить те, которые загрязнены. Для уборки помещения достаточно, чтобы машина-уборщик побывала в каждом из загрязненных секторов. Известно, что перед уборкой машина всегда находится в клетке (1,1) .



Одна из компаний, где в штате нет уборщицы, но имеется полный штат программистов, приобрела «Мусорщик-001». Пока программистам никак не удастся написать программу, определяющую по заданному плану загрязнения помещения минимально возможную длину маршрута машины-уборщика, необходимого для уборки данной территории. Возможно, Вам удастся им помочь!

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число n ($n \leq 1000$). Следующие n строк содержат по два натуральных числа: x_i и y_i – координаты загрязненных секторов в заданном помещении ($x_i, y_i \leq 50$).

Выходные данные

В выходной файл OUTPUT.TXT выведите целое число, соответствующее минимальной длине маршрута в метрах, необходимого для уборки.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	7 2 1 3 2 5 2 5 4 1 4 3 6 6 6	18

```

"Мусорщик": динамическое программирование
*/

#include <stdio.h>
#include <algorithm>
#include <vector>
#include <cmath>

int main() {
    int n;
    scanf("%d", &n);

    const int INF = (int)1e9;
    std::vector<int> min, max;
    for (int i = 0; i < n; ++i) {
        int r, c;
        scanf("%d %d", &c, &r);
        while ((int)min.size() <= r) min.push_back( INF);
        while ((int)max.size() <= r) max.push_back(-INF);
        min[r] = std::min(min[r], c);
        max[r] = std::max(max[r], c);
    }

    std::vector<std::vector<int>> dist(2, std::vector<int>(min.size(), -1));
    // Инициализация:
    max[1] = std::max(max[1], 1);
    min[1] = std::min(min[1], 1);
    dist[0][1] = max[1]-1 + max[1]-min[1];
    dist[1][1] = max[1]-1;
    // Переходы:
    for (int curr = 2, prev = 1; curr < (int)min.size(); ++curr) {
        while (min[curr] == INF) ++curr;
        // Если последняя посещенная клетка на текущей строке - минимум
        dist[0][curr] = std::min(
            dist[0][prev]+std::abs(max[curr]-min[prev]),
            dist[1][prev]+std::abs(max[curr]-max[prev])
        )+curr-prev+max[curr]-min[curr];

        // Если последняя посещенная клетка на текущей строке - максимум
        dist[1][curr] = std::min(
            dist[0][prev]+std::abs(min[curr]-min[prev]),
            dist[1][prev]+std::abs(min[curr]-max[prev])
        )+curr-prev+max[curr]-min[curr];
        // Переход к следующей итерации:
        prev = curr;
    }

    printf("%d", std::min(dist[0].back(), dist[1].back()));
    return 0;
}

```

ЗАДАЧА №867

Экзамен - 2

(Время: 1 сек. Память: 16 Мб Сложность: 45%)

Экзамен по берляндскому языку проходит в узкой и длинной аудитории. На экзамен пришло N студентов. Все они посажены в ряд. Таким образом, позиция каждого человека задается координатой на оси Ox (эта ось ведет вдоль длинной аудитории). Два человека могут разговаривать, если расстояние между ними меньше или равно D . Какое наименьшее количество типов билетов должен подготовить преподаватель, чтобы никакие два студента с одинаковыми билетами не могли разговаривать? Выведите способ раздачи преподавателем билетов.

Входные данные

В первой строке входного файла INPUT.TXT содержится два целых числа N, D ($1 \leq N \leq 10^4$; $0 \leq D \leq 10^6$). Вторая строка содержит последовательность различных целых чисел X_1, X_2, \dots, X_N , где X_i ($0 \leq X_i \leq 10^6$) обозначает координату вдоль оси Ox i -го студента.

Выходные данные

В первую строку выходного файла OUTPUT.TXT выведите Q – наименьшее количество типов билетов, необходимых для проведения экзамена. Во вторую строку выведите последовательность N целых чисел от 1 до Q , i -ое число этой последовательности обозначает номер типа билета i -го студента. Если ответов несколько, выведите любой.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 1 11 1 12 2	2 1 1 2 2
2	4 0 11 1 12 2	1 1 1 1 1

```
/*  
    "Экзамен - 2": сортировка, два указателя,  $O(n \log(n))$   
*/
```

```
#include <stdio.h>  
#include <algorithm>  
#include <vector>  
#include <set>  
  
struct Pair {  
    int pos, id;  
};  
  
int main() {  
    int n, dist;  
    scanf("%d %d", &n, &dist);  
  
    std::vector<Pair> pairs;  
    for (int i = 0; i < n; ++i) {  
        int x; scanf("%d", &x);
```

```

    pairs.push_back(Pair{x, i});
}

std::sort(pairs.begin(), pairs.end(), [](const Pair& a, const Pair& b) {
    return a.pos < b.pos;
});

std::set<int> curr;
for (int i = 1; i <= n; ++i) {
    curr.insert(i);
}

std::vector<int> answ(n);

int l = 0, r = -1;
while (l < n) {
    while (r+1 < n && pairs[r+1].pos - pairs[l].pos <= dist) {
        ++r;
        answ[pairs[r].id] = *curr.begin();
        curr.erase(curr.begin());
    }
    curr.insert(answ[pairs[l].id]);
    ++l;
}

printf("%d\n", *std::max_element(answ.begin(), answ.end()));

for (auto& it : answ) {
    printf("%d ", it);
}

return 0;
}

```

ЗАДАЧА №945

Баллы

(Время: 2 сек. Память: 64 Мб Сложность: 45%)

Мир наш развивается, строятся города, люди улетают в космос, изменяется система аттестации студентов в СФУ. Но вот проблема - систему аттестации студентов изменили, а программное обеспечение, которое поставлено в деканатах для контроля успеваемости, оставили прежним. Поэтому Вам срочно требуется внедрить во всех деканатах новую программу поиска студентов с заданным баллом!

Входные данные

В первой строке входного файла INPUT.TXT содержатся натуральные числа N и K ($N, K \leq 200\,000$) – соответственно количество студентов, подлежащих аттестации, и число запросов декана об успеваемости студентов. Во второй строке находятся N целых чисел a_i , упорядоченных по возрастанию. Эти числа - аттестационные баллы студентов. В третьей строке располагаются K целых чисел b_i , определяющие искомый балл. ($0 \leq a_i, b_i \leq 2^{32}$)

Выходные данные

В выходной файл OUTPUT.TXT выведите для каждого из K запросов через пробел слово «YES», если студент с таким баллом есть, и «NO» в противном случае.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 4 1 6 9 7 9 10 1	NO YES NO YES
2	2 2 1 2 1 3	YES NO

```
/*
"Баллы": бинарный поиск в массиве,  $O(n \log(n))$ 
*/

#include <iostream>
#include <algorithm>
#include <vector>

typedef long long ll;

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0); std::cout.tie(0); std::cerr.tie(0);

    int n, q;
    std::cin >> n >> q;

    std::vector<ll> a(n);
    for (auto& it : a) {
```

```
        std::cin >> it;
    }

    while (q--) {
        ll value;
        std::cin >> value;
        std::cout << (std::binary_search(a.begin(), a.end(), value) ? "YES " : "NO ");
    }
    return 0;
}
```


ЗАДАЧА №1084

Дерево Фенвика

(Время: 1 сек. Память: 16 Мб Сложность: 45%)

Секретная корпорация, занимающаяся поиском инопланетных жизненных форм обнаружила на одной из планет созвездия Альфа удивительные живые организмы (даже не плоские, а одномерные). Она приняла решение вести наблюдение за развитием и изменением численности организмов, с этой целью на орбиту планеты был послан спутник - наблюдатель, который мог следить за изменениями численности организмов. Недостаток этого «наблюдателя» в том, что он может отслеживать изменения только на той территории планеты, которая находится непосредственно под ним.

С этой целью его траектория была разбита на равные интервалы. Они пронумерованы от 1 до N. По запросу с Земли о количестве живых форм в интервале с L по R ($L \leq R$) - спутник должен, пролетая над ними (L, L+1, ..., R-1, R интервалами) произвести подсчет и затем, в ответ на запрос, отправить полученные данные. Но количество организмов постоянно изменяется: в некоторое время в X интервале на Y единиц.

Помогите написать программу для спутника, которая будет отвечать на запросы и отслеживать количество единиц жизни в каждом интервале.

Входные данные

Первая строка входного файла INPUT.TXT содержит два натуральных числа N и M – количество интервалов и запросов соответственно. В последующих M строках расположены запросы в следующем формате:

add X Y – изменение числа организмов в интервале с номером X на Y единиц;

rsq L R – запрос суммарного количества организмов с L по R интервал.

Все числа во входных данных не превосходят 10^5 по абсолютной величине.

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса суммы в отдельной строке выведите результат.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 3	
	add 1 4	4
	rsq 1 1	4
	rsq 1 1	

2	4 8	0 2 3 11
	rsq 1 4	
	add 1 3	
	add 4 2	
	rsq 2 4	
	rsq 1 2	
	add 4 -2	
	add 2 8	
	rsq 1 4	

```

#include <stdio.h>
#include <vector>
#include <cassert>
#include <string>

typedef long long ll;

struct SqrtDecomposition {
    std::vector<int> arr;
    std::vector<ll> sum;

    const int GSIZE = 128;

    SqrtDecomposition(const std::vector<int>& vec) : arr(vec) {
        sum.assign(((int)arr.size() + GSIZE - 1) / GSIZE, 0);
        for (int g = 0; g < (int)sum.size(); ++g) {
            const int after = std::min((g+1)*GSIZE, (int)arr.size());
            auto& it = sum[g];
            for (int i = g * GSIZE; i < after; ++i) {
                it += arr[i];
            }
        }
    }

    void add(int pos, int value) {
        const int g = pos / GSIZE;
        sum[g] += value;
        arr[pos] += value;
    }

    ll get(int l, int r) {
        const int gl = l / GSIZE;
        const int gr = r / GSIZE;
        if (gl == gr) {
            ll s = 0;
            for (int i = l; i <= r; ++i) {
                s += arr[i];
            }
            return s;
        } else {
            ll s = 0;
            for (int i = l, after = (gl+1)*GSIZE; i < after; ++i) {
                s += arr[i];
            }
            for (int g = gl+1; g < gr; ++g) {
                s += sum[g];
            }
            for (int i = gr * GSIZE; i <= r; ++i) {
                s += arr[i];
            }
            return s;
        }
    }
};

int main() {

```

```
int n; scanf("%d", &n);
SqrtDecomposition sd(std::vector<int>(n, 0));
int q; scanf("%d", &q);
while (q--) {
    char buf[4]; int l, r;
    scanf("%3s %d %d", buf, &l, &r);
    std::string t(buf);
    if (t == "rsq") {
        printf("%I64d\n", sd.get(l-1, r-1));
    } else {
        assert(t == "add");
        sd.add(l-1, r);
    }
}
return 0;
}
```

ЗАДАЧА №1298

Возрастающая подпоследовательность

(Время: 2 сек. Память: 16 Мб Сложность: 45%)

Даны N целых чисел X_1, X_2, \dots, X_N . Требуется вычеркнуть из них минимальное количество чисел так, чтобы оставшиеся шли в порядке возрастания.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N . Во второй строке записаны N чисел, разделенные пробелом. ($N \leq 10\,000$, $1 \leq X_i \leq 60\,000$)

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите количество невычеркнутых чисел, во второй – сами невычеркнутые числа через пробел в исходном порядке. Если вариантов несколько, следует вывести любой.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 1 3 5 2 4	3 1 3 5
2	6 2 5 3 4 6 1	4 2 3 4 6

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cassert>

int main() {
    int n; std::cin >> n;
    std::vector<int> a(n); for (auto& it : a) std::cin >> it;
    std::vector<int> len(n, 1);
    for (int curr = 1; curr < n; ++curr) {
        for (int prev = curr-1; prev >= 0; --prev) {
            if (a[prev] < a[curr] && len[prev]+1 > len[curr]) {
                len[curr] = len[prev]+1;
            }
        }
    }
    std::vector<int> answ;
    int curr = int(std::max_element(len.begin(), len.end()) - len.begin());
    int max = len.at(curr);
    answ.push_back(a.at(curr));
    for (int prev = curr-1; prev >= 0; prev--) {
        if (a[prev] < a[curr] && len[prev]+1 == len[curr]) {
            answ.push_back(a[prev]);
            curr = prev;
        }
    }
    std::reverse(answ.begin(), answ.end());
    assert((int)answ.size() == max);
}
```

```
std::cout << answ.size() << std::endl;
for (auto& it : answ) {
    std::cout << it << ' ';
}
std::cout << std::endl;
return 0;
}
```

ЗАДАЧА №1413

Перестановка

(Время: 1 сек. Память: 16 Мб Сложность: 45%)

Сегодня вам предстоит очутиться в шкуре хакера и взломать сверхсекретный компьютер потенциального противника. Система защиты у него не простая, а сортировочная.

Вам известно, что изначально в памяти компьютера хранится матрица A , которая содержит N строк и 3 столбца. А так же что компьютер может выполнять команды, состоящие из двух аргументов $str1$ и $str2$ ($str1 \neq str2$), по следующему алгоритму:

- 1. Циклически сдвинуть строку $str1$ на один элемент вправо;
- 2. Циклически сдвинуть строку $str2$ на один элемент вправо;
- 3. Поменять местами строки $str1$ и $str2$.

Для взлома компьютера вам нужно ввести такую последовательность команд, чтобы 1-й столбец матрицы был упорядочен по неубыванию, то есть $A[1][1] \leq A[2][1] \leq \dots \leq A[N-1][1] \leq A[N][1]$.

Входные данные

В первой строке входного файла INPUT.TXT содержится натуральное число N ($1 \leq N \leq 1000$). Далее следует N строк, каждая из которых содержит по три целых числа: A_{i1} , A_{i2} , A_{i3} - элементы исходной матрицы ($-10^9 \leq A_{i1}, A_{i2}, A_{i3} \leq 10^9$).

Выходные данные

Первая строка выходного файла OUTPUT.TXT должна содержать целое число M – число запросов к компьютеру ($0 \leq M \leq 10^4$).

Далее должно идти M строк, каждая из которых описывает одну команду и должна состоять из двух различных чисел $str1_i$ и $str2_i$ ($1 \leq str1_i, str2_i \leq N$).

Примеры

Примечание

После циклического сдвига на один элемент вправо строка матрицы «1, 2, 3» будет иметь вид «3, 1, 2».

№	INPUT.TXT	OUTPUT.TXT
1	3 1 2 3 6 5 4 3 2 1	1 2 3

2	3	4
	1 2 3	2 3
	4 5 6	2 3
	7 8 9	2 3
		2 3

```

#include <stdio.h>
#include <vector>

int main() {
    int n; scanf("%d", &n);
    std::vector<int> a(n), b(n), c(n);
    for (int i = 0; i < n; ++i) {
        scanf("%d %d %d", &a[i], &b[i], &c[i]);
    }
    std::vector<std::pair<int, int>> answ;
    for (int i = 0; i < n; ++i) {
        int min = i;
        for (int j = i+1; j < n; ++j) {
            if (a[j] < a[min]) {
                min = j;
            }
        }
        if (min != i) {
            for (int c = 0; c < 3; ++c) {
                answ.push_back({i+1, min+1});
            }
            std::swap(a[min], a[i]);
            std::swap(b[min], b[i]);
            std::swap(c[min], c[i]);
        }
    }
    printf("%d\n", (int)answ.size());
    for (auto& p : answ) {
        printf("%d %d\n", p.first, p.second);
    }
    return 0;
}

```

ЗАДАЧА №152

Построение

(Время: 1 сек. Память: 16 Мб Сложность: 46%)

Группа солдат-новобранцев прибыла в армейскую часть N666. После знакомства с прапорщиком стало очевидно, что от работ на кухне по очистке картофеля спасти солдат может только чудо.

Прапорщик, будучи не в состоянии запомнить фамилии, пронумеровал новобранцев от 1 до N. После этого он велел им построиться по росту (начиная с самого высокого). С этой несложной задачей могут справиться даже совсем необученные новобранцы, да вот беда, прапорщик уверил себя, что знает про некоторых солдат, кто из них кого выше, и это далеко не всегда соответствует истине.

После трех дней обучения новобранцам удалось выяснить, что знает (а точнее, думает, что знает) прапорщик. Помогите им, используя эти знания, построиться так, чтобы товарищ прапорщик остался доволен.

Входные данные

Во входном файле INPUT.TXT сначала идут числа N и M ($1 \leq N \leq 100$, $1 \leq M \leq 5000$) - количество солдат в роте и количество пар солдат, про которых прапорщик знает, кто из них выше. Далее идут эти пары чисел A и B по одной на строке ($1 \leq A, B \leq N$), что означает, что, по мнению прапорщика, солдат A выше, чем B.

Выходные данные

В выходной файл OUTPUT.TXT выведите "Yes" если можно построиться так, чтобы прапорщик остался доволен и "No" если нельзя.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 4 1 3 1 4 4 3 5 2	Yes

```
/*
    Задача: 152. Построение

    Решение: жадный алгоритм, графы,  $O(n^3)$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <stdio.h>
#include <vector>

int main() {
    int n, m; scanf("%d %d", &n, &m);
    std::vector<std::vector<int>> compare(n, std::vector<int>(n));
    while (m--) {
```



```

    int a, b;
    scanf("%d %d", &a, &b);
    --a, --b;
    if (compare[a][b] == -1 || a == b) {
        printf("No\n");
        return 0;
    }
    compare[a][b] = 1;
    compare[b][a] = -1;
}
std::vector<bool> visited(n, false);
for (int i = 0; i < n; ++i) {
    // Поиск вершины, больше которой никого нет, из множества оставшихся вершин
    int res = -1;
    for (int u = 0; u < n; ++u) {
        if (visited[u]) continue;
        bool flag = true;
        for (int v = 0; v < n; ++v) {
            if (compare[u][v] == -1) {
                flag = false;
                break;
            }
        }
        if (flag) {
            res = u;
            break;
        }
    }
    if (res == -1) {
        printf("No\n");
        return 0;
    }
    visited[res] = true;
    for (int v = 0; v < n; ++v) {
        compare[res][v] = compare[v][res] = 0;
    }
}
printf("Yes\n");
return 0;
}

```

ЗАДАЧА №192

Следующая перестановка ...

(Время: 1 сек. Память: 16 Мб Сложность: 46%)

Перестановкой из N элементов называется упорядоченный набор из N различных чисел от 1 до N .

Найдите по заданной перестановке следующую в лексикографическом порядке (будем считать, что за перестановкой $(N, N-1, \dots, 3, 2, 1)$ следует тождественная перестановка, то есть $(1, 2, 3, \dots, N)$).

Входные данные

В первой строке входного файла INPUT.TXT содержится число N ($1 \leq N \leq 10^4$). Во второй строке содержится перестановка (последовательность натуральных чисел от 1 до N , разделенных пробелами).

Выходные данные

Выходной файл OUTPUT.TXT должен содержать искомую перестановку.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 1	1
2	5 2 4 5 3 1	2 5 1 3 4

```
#pragma GCC diagnostic ignored "-Wunused-result"

#include <stdio.h>
#include <bits/stdc++.h>

int main() {
    int n; scanf("%d", &n);

    std::vector<int> a(n);
    for (auto& it : a) scanf("%d", &it);

    if (!std::next_permutation(a.begin(), a.end())) {
        std::reverse(a.begin(), a.end());
    }

    for (auto& it : a) {
        printf("%d ", it);
    }
    printf("\n");

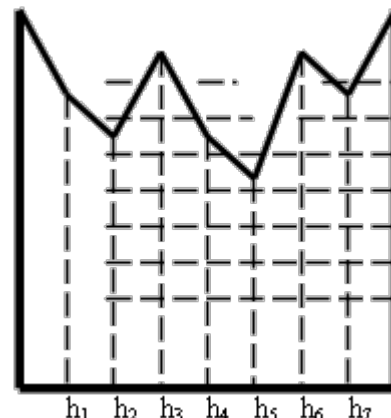
    return 0;
}
```

ЗАДАЧА №505

Забор

(Время: 2 сек. Память: 16 Мб Сложность: 46%)

Для парка культуры и отдыха было решено изготовить забор. Чтобы забор не портил своим невзрачным видом облик города, архитекторы решили сделать забор фигурным, и разработали несколько различных шаблонов для изготовления секций забора. Шаблон представляет собой многоугольник, где три стороны всегда одинаковы, а четвертая сторона (верхняя часть забора) представляет ломаную линию. Проекции вершин этой ломаной линии на основание забора следуют равномерно, и таким образом шаблон описывается как последовательность высот точек. Высота точек указывается в целых миллиметрах и варьируется от 0 до 2047.



Количество точек в шаблоне равняется L . Архитекторы разработали M шаблонов, и завод изготовил по шаблонам N секций. Пример секции с $L = 7$ изображен на рисунке.

При транспортировке секций забора до парка произошла авария, и готовые секции забора рассыпались и перемешались. При разборе завалов были выполнены измерения секций, и теперь, зная высоты, необходимо определить, к какому шаблону принадлежит каждая секция забора. Кроме того, при падении секции забора могли быть повреждены (например, могли обломаться зубья забора), поэтому необходимо также посчитать случаи, когда извлечённая из завала секция не совпадает ни с одним шаблоном.

Впрочем, дело несколько упрощается – внутренняя (обращённая в парк) сторона секции отличается по цвету от внешней стороны, поэтому можно сказать с уверенностью, что измерения высот секции проделаны в том же порядке, что и в шаблоне.

Входные данные

Входной файла INPUT.TXT содержит целые значения L – число точек в шаблоне и секции, M – число различных шаблонов, N – число найденных секций ($1 \leq L, M \leq 1000, N \leq 1000$). Далее M строчек содержат информацию о шаблоне: номер шаблона (натуральное число не более 1000) и L точек шаблона. Далее N строчек содержат информацию о секциях забора, в каждой строке содержится L точек одной секции забора (высота секции от 0 до 8191).

Выходные данные

В выходной файл OUTPUT.TXT следует вывести N строк, каждая строка содержит номер шаблона, с которым совпала секция, либо, если подходящего шаблона не найдено, то выводится символ дефиса '-'. В последней строке теста выводится слово "OK=" (без кавычек) и число секций, сопоставленных с шаблонами, через пробел символы "BAD=" и число испорченных секций.

Примеры

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	4 2 4	
	1 250 123 0 66	1
	2 22 31 120 100	2
	250 123 0 66	-
	22 31 120 100	1
	25 31 120 100	OK=3 BAD=1
	250 123 0 66	
2	3 3 4	
	1 55 11 12	1
	3 33 1 2	2
	2 14 15 2	3
	55 11 12	2
	14 15 2	OK=4 BAD=0
	33 1 2	
	14 15 2	

```
#pragma GCC diagnostic ignored "-Wunused-result"
#include <stdio.h>
#include <bits/stdc++.h>
```

```
template<typename T>
bool operator<(const std::vector<T>& a, const std::vector<T>& b) {
    assert(a.size() == b.size());
    for (int i = 0; i < (int)a.size(); ++i) {
        if (a[i] < b[i]) {
            return true;
        } else if (a[i] > b[i]) {
            return false;
        }
    }
    return false;
}
```

```
int main() {
    std::map<std::vector<int>, int> pattern;
    int len, nPatterns, nQueries;
    scanf("%d %d %d", &len, &nPatterns, &nQueries);

    std::vector<int> temp(len);
    for (int i = 0; i < nPatterns; ++i) {
        int id; scanf("%d", &id);
        for (auto& it : temp) scanf("%d", &it);
        pattern[temp] = id;
    }
    int ok = 0, bad = 0;
    for (int i = 0; i < nQueries; ++i) {
        for (auto& it : temp) scanf("%d", &it);
        auto it = pattern.find(temp);
        if (it == pattern.end()) {
            printf("-\n"); ++bad;
        } else {
            printf("%d\n", it->second); ++ok;
        }
    }
    printf("OK=%d BAD=%d\n", ok, bad);

    return 0;
}
```

ЗАДАЧА №553

Объединение блоков

(Время: 1 сек. Память: 16 Мб Сложность: 46%)

Изделие изготавливают из n блоков, каждый из которых имеет два технологических параметра – m_i и k_i . Известно, что $k_i = m_{i+1}$, $i=1, 2, \dots, n-1$. При этом условии два последовательных блока i и $i+1$ можно объединять в один новый, который будет иметь технологические параметры – m_i и k_{i+1} , и на это потребуется $m_i * k_{i+1}$ технологических операций. Новый блок можно опять объединять с одним из соседних и так далее. Меняя порядок сборки блоков можно добиться уменьшения количества технологических операций.

Поясним это на примере трех блоков: 34 и 29, 29 и 4, 4 и 15. Если собрать вначале 2 и 3 блок, а затем присоединить собранное к первому, то потребуется $29*15+34*15=435+510=945$ операций. Если собрать вначале блок из 1 и 2 исходных блоков, а затем присоединить 3 блок, то потребуется $34*4+34*15=136+510=646$ операций.

Требуется написать программу, которая найдет минимальное число технологических операций для изготовления изделия.

Входные данные

Входной файл INPUT.TXT содержит в первой строке число n – количество блоков ($1 \leq n \leq 100$). Последующие n строк содержат пары чисел (разделенных пробелом) – технологические параметры блоков. Технологические параметры – целые неотрицательные числа, не превышающие 100.

Выходные данные

Выходной текстовый файл OUTPUT.TXT должен содержать одно число – минимальное число технологических операций.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 34 29 29 4 4 15	646

```
#pragma GCC diagnostic ignored "-Wunused-result"
#include <stdio.h>
#include <vector>
#include <algorithm>
```

```
struct Record {
    int sum, left, right;
};
```

```
Record combine(const Record& a, const Record& b) {
    return Record{a.sum + b.sum + a.left * b.right, a.left, b.right};
}
```

```

bool operator<(const Record& a, const Record& b) {
    return a.sum < b.sum;
}

int main() {
    int n; scanf("%d", &n);

    const int INF = (int)1e9+1;
    std::vector<std::vector<Record>> min(n, std::vector<Record>(n, Record{INF, 0, 0})); //

    for (int i = 0; i < n; ++i) {
        int m, k;
        scanf("%d %d", &m, &k);
        min[i][i] = Record{0, m, k};
    }

    for (int len = 2; len <= n; ++len) {
        for (int i = 0, j = len-1; j < n; ++i, ++j) {
            for (int k = i; k < j; ++k) {
                min[i][j] = std::min(min[i][j], combine(min[i][k], min[k+1][j]));
            }
        }
    }

    printf("%d\n", min[0][n-1].sum);

    return 0;
}

```

ЗАДАЧА №628

Clear World and Brothers

(Время: 3 сек. Память: 16 Мб Сложность: 46%)

Наконец в деревнях Виллорибо и Виллобаджо закончились праздники. Перемыта вся посуда! Этот процесс прошел так быстро и непринужденно, что братьями Карлионе было решено открыть сеть агентств «Clear World and Brothers», специализирующихся на профессиональном мытье посуды. В области Новая Берляндия, где и находятся знаменитые деревни, всего N деревень. Система координат введена так, что Виллорибо имеет координаты $(x_1, 0)$, а Виллобаджо - $(x_2, 0)$.

Координаты всех деревень целые числа не превосходящие по модулю 10^6 . Вы работаете на мистера Берлионе старшего и ваша задача найти оптимальное расположение для регионального отделения «Clear World and Brothers», то есть сумма расстояний от агентства до всех деревень должна быть наименьшей и агентство обязательно должно располагаться на прямолинейном шоссе Виллорибо-Виллобаджо (возможно расположение не только внутри, но и на границе отрезка).

Входные данные

В первой строке входного файла INPUT.TXT записано натуральное число N ($2 \leq N \leq 15000$). Далее в N строках записаны пары координат всех вершин. Виллорибо и Виллобаджо первая и вторая деревня соответственно. Возможно, что сколько-то деревень расположены так близко, что их координаты совпадают.

Выходные данные

В выходной файл OUTPUT.TXT выведите абсциссу оптимального расположения агентства. Разрешается абсолютная погрешность не более единицы.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4 -10 0 10 0 3 1 1 -1	2.000000

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>
#include <cmath>
#include <iostream>
#include <iomanip>

typedef long double Real;

struct Point {
    Real x, y;

    Point(Real x = 0, Real y = 0) : x(x), y(y) {}

    static Point read() {
```

```

        int x, y;
        scanf("%d %d", &x, &y);
        return Point(x, y);
    }

    Real dist(const Point& other) const {
        Real dx = x - other.x;
        Real dy = y - other.y;
        return std::sqrt(dx*dx+dy*dy);
    }
};

Real dist(Point s, const std::vector<Point>& points) {
    Real sum = 0;
    for (auto it : points) {
        sum += it.dist(s);
    }
    return sum;
}

int main() {
    int n; scanf("%d", &n);
    std::vector<Point> points(n);

    for (auto &it : points) it = Point::read();

    Real L = std::min(points[0].x, points[1].x);
    Real R = std::max(points[0].x, points[1].x);
    Real dist_L = dist({L, 0}, points);
    Real dist_R = dist({R, 0}, points);

    const Real EPS = 1e-8;
    while (R - L > EPS) {
        Real l = (6 * L + 5 * R) / 11;
        Real r = (5 * L + 6 * R) / 11;
        Real dist_l = dist({l, 0}, points);
        Real dist_r = dist({r, 0}, points);
        if (dist_l < dist_r) {
            R = r;
            dist_R = dist_r;
        } else {
            L = l;
            dist_L = dist_l;
        }
    }
    std::cout << std::fixed << std::setprecision(8) << (L+R)/2 << std::endl;
    return 0;
}

```


ЗАДАЧА №744

Скобочки-3

(Время: 1 сек. Память: 16 Мб Сложность: 46%)

Напомним, что называется правильной скобочной последовательностью:

Циклическим сдвигом строки s называется строка $s_k s_{k+1} s_{k+2} \dots s_{|s|} s_1 s_2 \dots s_{k-1}$ для некоторого k , где $|s|$ — длина строки s .

Вам дана скобочная последовательность s — строка, состоящая из символов $\{, \}, (,), [,]$. Выясните, является ли s циклическим сдвигом правильной скобочной последовательности.

Входные данные

В первой строке входного файла INPUT.TXT записана строка s (длина строки s не превышает 1000).

Выходные данные

В выходной файл OUTPUT.TXT выведите «YES», если s является циклическим сдвигом правильной скобочной последовательности, иначе — выведите "NO".

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	}0[]{	YES
2	}([]){	NO
3	()[]	YES

```
/*
"Скобочки-3": циклические сдвиги, стек, правильная скобочная последовательность
*/
```

```
#include <stdio.h>
#include <algorithm>
#include <string>
#include <vector>
```

```
inline bool is_open(char c) {
    return c == '(' || c == '[' || c == '{';
}
```

```
inline bool same_type(char c1, char c2) {
    return
        (c1 == '(' && c2 == ')') ||
        (c1 == '[' && c2 == ']') ||
        (c1 == '{' && c2 == '}');
}
```

```
bool correct(const std::string& s) {
    std::vector<char> stack;
    for (auto c : s) {
        if (is_open(c)) {
            stack.push_back(c);
        }
    }
    return stack.empty();
}
```

```

        } else {
            if (stack.empty() || !same_type(stack.back(), c)) {
                return false;
            }
            stack.pop_back();
        }
    }
    return stack.empty();
}

int main() {
    char buf[1001] = {};
    scanf("%1000s", buf);
    std::string s(buf);
    for (int i = 0; i < (int)s.size(); ++i) {
        std::rotate(s.begin(), s.begin()+1, s.end());
        if (correct(s)) {
            printf("YES");
            return 0;
        }
    }
    printf(s.empty() ? "YES" : "NO");
    return 0;
}

```

ЗАДАЧА №806

Белоснежка и n гномов

(Время: 1 сек. Память: 64 Мб Сложность: 46%)

Ну не гномы, а наказание какое-то! Подумала Белоснежка, в очередной раз пытаясь уложить гномов спать. Одного уложишь, другой уже проснулся! И так всю ночь. У Белоснежки n гномов, и все они очень разные. Она знает, что для того, чтобы уложить спать i -го гнома нужно a_i минут, и после этого он будет спать ровно b_i минут. Помогите Белоснежке узнать, может ли она получить хотя бы минутку отдыха, когда все гномы будут спать, и если да, то в каком порядке для этого нужно укладывать гномов спать.

Например, пусть есть всего два гнома, $a_1=1, b_1=10, a_2=10, b_2=20$. Если Белоснежка сначала начнет укладывать первого гнома, то потом ей потребуется целых 10 минут, чтобы уложить второго, а за это время проснется первый. Если же она начнет со второго гнома, то затем она успеет уложить первого и получит целых 10 минут отдыха.

Входные данные

Первая строка входного файла INPUT.TXT содержит число n ($1 \leq n \leq 10^5$), вторая строка содержит числа a_1, a_2, \dots, a_n , третья - числа b_1, b_2, \dots, b_n ($1 \leq a_i, b_i \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите n чисел – порядок, в котором нужно укладывать гномов спать. Если Белоснежке отдохнуть не удастся, выведите число -1 .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 1 10 10 20	2 1
2	2 10 10 10 10	-1

```
/*
"Белоснежка и n гномов": жадный алгоритм, сортировка,  $O(n \log(n))$ 
*/

#include <stdio.h>
#include <algorithm>
#include <vector>

struct Pair {
    int prepare, sleep, id;
};

int main() {
    int n;
    scanf("%d", &n);
```

```

std::vector<Pair> arr(n);
for (int i = 0; i < n; ++i) {
    arr[i].id = i+1;
}

for (auto& it : arr) scanf("%d", &it.prepare);
for (auto& it : arr) scanf("%d", &it.sleep);

long long sum_prepare = 0;
for (auto& it : arr) {
    sum_prepare += it.prepare;
}

// Первым нужно выбрать того, кто спит больше общего времени на подготовку ко сну остал
// Среди кандидатов нужно выбрать того, кого дольше всего готовить ко сну.
int min = -1;
for (int i = 0; i < n; ++i) {
    if (arr[i].sleep + arr[i].prepare > sum_prepare && (min == -1 || arr[i].prepare > a
        min = i;
    }
}
if (min == -1) {
    printf("-1");
    return 0;
}
std::swap(arr[0], arr[min]);

// Остальных необходимо отсортировать по величине "время сна" + "время подготовки" в по
std::stable_sort(arr.begin(), arr.end(), [](const Pair& a, const Pair& b) {
    return a.sleep + a.prepare > b.sleep + b.prepare;
});

// Проход по отсортированным тномам и проверка на существование минуты отдыха:
int t = arr[0].sleep;
for (int i = 1; i < n; ++i) {
    t -= arr[i].prepare;
    if (t <= 0) {
        printf("-1");
        return 0;
    }
    t = std::min(arr[i].sleep, t);
}

// Вывод ответа:
for (auto& it : arr) {
    printf("%d ", it.id);
}
return 0;
}

```

ЗАДАЧА №953

Дроби

(Время: 1 сек. Память: 16 Мб Сложность: 46%)

В то время, пока другие дети бегали по улицам или гоняли мяч, мальчик Слава сидел дома и решал сложную математическую проблему. Вкратце, проблема выглядела так.

Каждое натуральное число, начиная с трёх, можно представить в виде суммы различных натуральных чисел, например, $5=3+2=4+1$. А возможно ли представить правильную дробь m/n в виде суммы различных членов гармонического ряда $1/2, 1/3, 1/4, \dots$, то есть $m/n=1/x+1/y+1/z+\dots$? При этом должно выполняться условие $x < y < z < \dots$. Если существует несколько решений, то надо найти то из них, у которого значение x минимально. Если неоднозначность не снимается, то надо найти решение с минимальным y , и так далее.

Входные данные

Входной файл INPUT.TXT содержит два натуральных числа m и n ($1 \leq m < n \leq 32$).

Выходные данные

В выходной файл OUTPUT.TXT выведите найденные числа x, y, z, \dots через пробел.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 6	2 3

```
/*
   "Дроби": математика, наибольший общий делитель, бинарный поиск
*/

#include <iostream>

typedef long long ll;

inline ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0); std::cout.tie(0); std::cerr.tie(0);

    ll p, q;
    std::cin >> p >> q;

    while (p != 0) {
        auto gcd = ::gcd(p, q);
        p /= gcd;
        q /= gcd;
        ll low = 1, high = q;
        while (high - low > 1) {
            ll mid = (low + high) / 2;
```

```
        if (p * mid >= q) {
            high = mid;
        } else {
            low = mid;
        }
    }
    std::cout << high << " ";
    p = p * high - q;
    q *= high;
}

return 0;
```

ЗАДАЧА №1155

Рекламный щит

(Время: 1 сек. Память: 16 Мб Сложность: 46%)

Для рекламы своей новой продукции в Китае одна компания решила разместить на небоскребе рекламный щит. Щит состоит из лампочек, организованных в форме прямоугольной сетки из N строк и M столбцов. В любой момент каждая из лампочек может быть либо включена, либо выключена.

Рекламное сообщение состоит из K иероглифов, которые будут показываться один за другим. Для каждого иероглифа известно, какие лампочки должны быть включены при отображении этого иероглифа. Остальные лампочки должны быть выключены.

Для управления рекламным щитом разрабатывается специальная система. Система может включать и выключать лампочки целыми группами. Все лампочки разбиваются на несколько групп так, что в каждом иероглифе лампочки из одной группы должны быть либо все включены, либо все выключены.

Для оптимизации работы системы управления необходимо разбить лампочки на минимальное возможное число таких групп. Помогите сотрудникам рекламного отдела компании решить эту задачу.

Входные данные

В первой строке входного файла INPUT.TXT заданы числа K , N и M ($1 \leq K, N, M \leq 100$) – количество иероглифов в рекламном сообщении, высота и ширина рекламного щита.

Далее, в $K \cdot N$ строках идет описание иероглифов. Каждый из K иероглифов задается N строками по M символов в каждой. Все эти строки состоят только из символов «*» и «.», «*» соответствует включенной лампочке, «.» – выключенной.

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное число групп, на которое можно разбить лампочки.

Пример

Пояснения к примеру

В приведенном примере можно разбить лампочки на группы следующим образом: две лампочки из первого столбца образуют одну группу, две лампочки из последнего столбца – вторую, а каждая из двух оставшихся лампочек образует отдельную группу.

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	3 2 3	4
	*..	
	*..	
	**.	
	*..	
	...	
	.*.	

```

#include <stdio.h>
#include <algorithm>
#include <vector>
#include <string>
#include <cassert>

typedef bool Item;
typedef std::vector<Item> Vector;
typedef std::vector<Vector> Matrix;
typedef std::vector<Matrix> Tensor;

int main() {
    int n, nRows, nCols;
    scanf("%d %d %d", &n, &nRows, &nCols);

    Matrix table(nRows*nCols, Vector(n));
    for (int i = 0; i < n; ++i) {
        for (int r = 0; r < nRows; ++r) {
            char buf[101];
            scanf("%100s", buf);
            for (int c = 0; c < nCols; ++c) {
                table[r*nCols+c][i] = (buf[c] == '*');
            }
        }
    }

    std::sort(table.begin(), table.end());
    table.erase(std::unique(table.begin(), table.end()), table.end());

    printf("%d", (int)table.size());
    return 0;
}

```


ЗАДАЧА №1183

НОД на отрезках

(Время: 1 сек. Память: 32 Мб Сложность: 46%)

Задан числовой массив $A[1..N]$. Необходимо выполнить M операций вычисления наибольшего общего делителя среди всех чисел отрезка $A[L..R]$.

Входные данные

Первая строка входного файла INPUT.TXT содержит число N – размер массива ($N \leq 10^5$). Во второй строке записаны N чисел – элементы массива (целые числа от 1 до 10^9). Третья строка содержит натуральное число M – количество запросов вычисления НОД ($M \leq 30\,000$). Следующие M строк содержат пары чисел L и R ($1 \leq L \leq R \leq N$), описывающие отрезки.

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса выведите значение НОД на отрезке через пробел.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 2 2 2 1 5 2 2 3 2 5	2 1

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <climits>

int gcd(int a, int b) {
    while (b != 0) {
        int rem = a % b;
        a = b;
        b = rem;
    }
    return a;
}

struct DataStruct {
    std::vector<int> arr, gcd;

    const int GSIZE = 128;

    DataStruct(int size = 0, int item = 0) {
        arr.assign(size, item);
        gcd.assign(size / GSIZE + 1, item);
    }

    void set(int p, int v) {
```

```

    arr[p] = v;
    const int g = p / GSIZE;
    gcd[g] = v;
    const int begin = g * GSIZE;
    const int after = std::min(begin + GSIZE, (int)arr.size());
    for (int i = begin; i < after; ++i) {
        gcd[g] = ::gcd(arr[i], gcd[g]);
    }
}

int get(int l, int r) {
    const int gl = l / GSIZE;
    const int gr = r / GSIZE;
    int answ = arr[l];
    if (gl == gr) {
        for (int i = l; i <= r; ++i) {
            answ = ::gcd(answ, arr[i]);
        }
    } else {
        for (int i = l, after = (gl+1)*GSIZE; i < after; ++i) {
            answ = ::gcd(answ, arr[i]);
        }
        for (int g = gl+1; g < gr; ++g) {
            answ = ::gcd(answ, gcd[g]);
        }
        for (int i = gr * GSIZE; i <= r; ++i) {
            answ = ::gcd(answ, arr[i]);
        }
    }
    return answ;
}

};

int main() {
    int n;
    scanf("%d", &n);
    DataStruct ds(n, 0);
    for (int i = 0; i < n; ++i) {
        int value;
        scanf("%d", &value);
        ds.set(i, value);
    }
    int q;
    scanf("%d", &q);
    while (q--) {
        int l, r;
        scanf("%d %d", &l, &r);
        --l, --r;
        printf("%d ", ds.get(l, r));
    }
    return 0;
}

```

ЗАДАЧА №541

Две строки

(Время: 1 сек. Память: 16 Мб Сложность: 47%)

Циклическим сдвигом строки $s_1 s_2 \dots s_n$ на k позиций назовем строку $s_{k+1} s_{k+2} \dots s_n s_1 \dots s_k$.

Например, циклическим сдвигом строки «abcde» на две позиции является строка «cdeab». В этой задаче далее будут рассматриваться только строки, состоящие из десятичных цифр от 0 до 9.

Произвольной такой строке, первый символ которой не является нулем, можно сопоставить число, десятичной записью которого она является. Строкам, которые начинаются с нуля, никакое число сопоставляться не будет. Например, строке 123 сопоставляется число сто двадцать три, а строке 0123 не сопоставляется никакое число.

Пусть заданы две строки: s и t . Обозначим как S набор всех циклических сдвигов строки s , а как T – набор всех циклических сдвигов строки t . Например, если $s = \text{«1234»}$, то S содержит строки «1234», «2341», «3412», «4123». Обозначим также как $\text{NUM}(A)$ набор чисел, соответствующих строкам из набора A .

Требуется написать программу, которая по строкам s и t определит, максимальное число, представимое в виде разности $(x - y)$, где x принадлежит $\text{NUM}(S)$, а y принадлежит $\text{NUM}(T)$. Например, если $s = \text{«25»}$, $t = \text{«12»}$, то $\text{NUM}(S)$ содержит числа 25 и 52, $\text{NUM}(T)$ – числа 12 и 21; их попарными разностями будут: $25 - 12 = 13$, $25 - 21 = 4$, $52 - 12 = 40$, $52 - 21 = 31$. Из этих разностей максимальным числом является 40.

Входные данные

Первая строка входного файла INPUT.TXT содержит строку s , вторая строка входного файла – строку t . Обе строки непустые. Они содержат только цифры, из которых хотя бы одна не является нулем. Строки имеют длину не более 3000 символов.

Выходные данные

В выходной файл OUTPUT.TXT выведите искомое число без ведущих нулей.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	25 12	40
2	100 1	99

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <cassert>
```

```
bool operator<(const std::string& s, const std::string& t) {
    const int size = (int)std::max(s.size(), t.size());
    for (int i = 0; i < size; ++i) {
        int d1 = i < size-(int)s.size() ? 0 : s[i-(size-(int)s.size())]-'0';
```

```

        int d2 = i < size-(int)t.size() ? 0 : t[i-(size-(int)t.size())]-'0';
        if (d1 > d2) {
            return false;
        }
        if (d1 < d2) {
            return true;
        }
    }
    return false;
}

bool operator>(const std::string& s, const std::string& t) {
    return t < s;
}

std::string max_shift(std::string s) {
    std::string answ = "?";
    for (int i = 0; i < (int)s.size(); ++i) {
        std::rotate(s.begin(), s.begin()+1, s.end());
        if (s[0] == '0') {
            continue;
        }
        if (answ == "?" || s > answ) {
            answ = s;
        }
    }
    assert(answ != "?");
    return answ;
}

std::string min_shift(std::string s) {
    std::string answ = "?";
    for (int i = 0; i < (int)s.size(); ++i) {
        std::rotate(s.begin(), s.begin()+1, s.end());
        if (s[0] == '0') {
            continue;
        }
        if (answ == "?" || s < answ) {
            answ = s;
        }
    }
    assert(answ != "?");
    return answ;
}

std::string& operator+=(std::string& a, std::string b) {
    assert(!(a < b));
    b = std::string(a.size()-b.size(), '0') + b;
    int rem = 0;
    for (int i = (int)b.size()-1; i >= 0; --i) {
        int digit = a[i] - b[i] - rem;
        if (digit < 0) {
            digit += 10;
            rem = 1;
        } else {
            rem = 0;
        }
        a[i] = digit+'0';
    }
    assert(rem == 0);
    return a;
}

std::string& normalize(std::string& a) {
    int pos = 0;
    while (pos < (int)a.size()-1 && a[pos] == '0') {
        ++pos;
    }
    assert(a[pos] != '0' || (pos == (int)a.size()-1));
}

```

```

    return a = a.substr(pos);
}

int main() {
    std::string a, b;
    std::cin >> a >> b;
    a = max_shift(a);
    b = min_shift(b);

    int sign = 1;
    assert(a[0] != '0');
    assert(b[0] != '0');

    if (!(a < b)) {
        a -= b;
    } else {
        b -= a;
        sign = -1;
        a = b;
    }
    normalize(a);
    a = (sign == -1 ? "-" : "") + a;
    assert(a != "-0");
    std::cout << a;
    return 0;
}

```

ЗАДАЧА №562

Слабая К-связность

(Время: 1 сек. Память: 16 Мб Сложность: 47%)

Ане, как будущей чемпионке мира по программированию, поручили очень ответственное задание. Правительство вручает ей план постройки дорог между N городами. По плану все дороги односторонние, но между двумя городами может быть больше одной дороги, возможно, в разных направлениях. Ане необходимо вычислить минимальное такое K , что данный ей план является слабо K -связным.

Правительство называет план слабо K -связным, если выполнено следующее условие: для любых двух различных городов можно проехать от одного до другого, нарушая правила движения не более K раз. Нарушение правил - это проезд по существующей дороге в обратном направлении. Гарантируется, что между любыми двумя городами можно проехать, возможно, несколько раз нарушив правила.

Входные данные

В первой строке входного файла INPUT.TXT записаны два числа $2 \leq N \leq 300$ и $1 \leq M \leq 10^5$ - количество городов и дорог в плане. В последующих M строках даны по два числа - номера городов, в которых начинается и заканчивается соответствующая дорога.

Выходные данные

В выходной файл OUTPUT.TXT выведите минимальное K , такое, что данный во входном файле план является слабо K -связным.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 2 1 2 1 3	1
2	4 4 2 4 1 3 4 1 3 2	0

```
/*
"Слабая К-связность": алгоритм Флойда-Уоршелла,  $O(n^3)$ 
*/

#include <stdio.h>
#include <vector>
#include <cassert>
#include <algorithm>

int main() {
    int n;
    scanf("%d", &n);
```

```

const int INF = (int)1e9;
std::vector<std::vector<int>> dist(n, std::vector<int>(n, INF));

for (int i = 0; i < n; ++i) {
    dist[i][i] = 0;
}

int m;
scanf("%d", &m);

while (m--) {
    int a, b;
    scanf("%d %d", &a, &b);
    --a, --b;
    dist[a][b] = 0;
    dist[b][a] = std::min(dist[b][a], 1);
}

for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (dist[i][k] < INF && dist[k][j] < INF) {
                dist[i][j] = std::min(dist[i][j], dist[i][k]+dist[k][j]);
            }
        }
    }
}

int answ = 0;
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        answ = std::max(answ, dist[i][j]);
    }
}
printf("%d", answ);
return 0;
}

```

ЗАДАЧА №582

Кубик

(Время: 1 сек. Память: 16 Мб Сложность: 47%)

Все мы в детстве играли в кубики. Были у нас и кубики с цифрами, и кубики с буквами. Были также и разноцветные кубики. Маленький Андрюша тоже любит играть в кубики. У него есть несколько наборов кубиков, причем все кубики из одного набора раскрашены одинаково, а кубики из разных наборов - по-разному.

На столе у Андрюши лежат два кубика. Помогите ему определить, принадлежат они одному набору или нет. Стол у Андрюши стеклянный, поэтому он видит цвета всех граней кубика. Кубики принадлежат одному набору, если один из них можно комбинацией поворотов вокруг осей, проходящих через середины противоположных граней, перевести в другой.

Входные данные

Первая строка входного файла INPUT.TXT содержит описание первого кубика в формате: цвет передней грани, цвет задней грани, цвет верхней грани, цвет нижней грани, цвет левой грани, цвет правой грани. Во второй строке находится описание второго кубика в таком же формате. Цвета граней кодируются числами. Все числа во входном файле - целые, положительные и не превосходят 100.

Выходные данные

В выходной файл OUTPUT.TXT выведите «YES», если первый и второй кубики принадлежат одному набору, и «NO» - в противном случае.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 2 3 4 5 6 1 2 3 4 5 6	YES
2	1 2 3 4 5 6 1 1 1 1 1 1	NO
3	1 2 3 4 5 6 5 6 3 4 2 1	YES

```
#include <bits/stdc++.h>

struct Cube {
    int f, l, b, r, u, d;

    Cube rotate_left() const {
        return Cube{l, b, r, f, u, d};
    }

    Cube rotate_up() const {
        return Cube{d, l, u, r, f, b};
    }

    static Cube read() {
        int f, b, u, d, l, r;
```



```

        std::cin >> f >> b >> u >> d >> l >> r;
        return Cube{f, l, b, r, u, d};
    }
};

bool operator<(const Cube& a, const Cube& b) {
    if (a.f < b.f) return true; else if (a.f > b.f) return false;
    if (a.l < b.l) return true; else if (a.l > b.l) return false;
    if (a.b < b.b) return true; else if (a.b > b.b) return false;
    if (a.r < b.r) return true; else if (a.r > b.r) return false;
    if (a.u < b.u) return true; else if (a.u > b.u) return false;
    return a.d < b.d;
}

int main() {
    auto s = Cube::read();
    auto t = Cube::read();
    std::set<Cube> cubes;
    std::queue<Cube> queue;
    queue.push(s);
    cubes.insert(s);
    while (!queue.empty()) {
        auto curr = queue.front(); queue.pop();
        for (auto& next : {curr.rotate_left(), curr.rotate_up()}) {
            if (cubes.find(next) == cubes.end()) {
                cubes.insert(next);
                queue.push(next);
            }
        }
    }
    std::cout << (cubes.find(t) == cubes.end() ? "NO" : "YES") << std::endl;
    return 0;
}

```

ЗАДАЧА №739

Дана строка

(Время: 1 сек. Память: 16 Мб Сложность: 47%)

Васе уже надоели задачи на строки! А Вам? А что делать? Что ж, приступим. Дана строка из маленьких букв английского алфавита. Разрешается любой ее символ сдвинуть не более, чем на K позиций в любую сторону так, чтобы в конечном счете они все встали на разные позиции (кроме случая, когда $K=0$). Например, если строка - aababac, а $K = 2$, то таким образом можно получить строки abaabac или aaaabbc, но нельзя - aaacbab или aaaacbb.

Вася хочет сделать так, чтобы получившаяся с помощью такой операции строка была минимально возможной лексикографически (т.е. расположена раньше всех по правилам упорядочивания слов в словаре). Как же ему быть?

Входные данные

В первой строке входного файла INPUT.TXT задано число K ($K \geq 0$). Во второй строке задана сама исходная непустая строка, длиной не более 100 000 маленьких английских букв. Гарантируется, что K не превосходит длины строки.

Выходные данные

В выходной файл OUTPUT.TXT выведите лексикографически минимальный из возможных результатов.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	2 aababac	aaaabbc

```
#include <stdio.h>
#include <algorithm>
#include <string>
#include <vector>
#include <set>
#include <cmath>
#include <cassert>

struct Pair {
    char item; int pos;
};

inline bool operator<(const Pair& a, const Pair& b) {
    return a.item < b.item || (a.item == b.item && a.pos < b.pos);
}

std::string fast(const std::string& s, const int k) {
    const int n = s.size();
    int last = -1;
    std::vector<bool> in(n);
    std::string ans;
    std::set<Pair> set;
```

```

for (int i = 0; i < n; ++i) {
    // Добавляем в std::set всех кандидатов:
    while (last+1-i <= k && last+1 < n) {
        ++last;
        set.insert(Pair{s[last],last});
        in[last] = true;
    }
    // Кандидат на вставку - текущий минимум:
    auto it = set.begin();
    // Если есть элемент, который двигать нельзя, то кандидатом становится он:
    if (i-k >= 0 && in[i-k]) {
        it = set.find(Pair{s[i-k], i-k});
        assert(it != set.end());
    }
    // Вставляем элемент в ответ:
    answ.push_back(it->item);
    // Удаляем из std::set:
    in[it->pos] = false;
    set.erase(it);
}
return answ;
}

int main() {
    int k;
    scanf("%d", &k);

    char buf[100000+1];
    scanf("%100000s", buf);
    std::string s(buf);

    printf("%s", fast(s, k).c_str());

    return 0;
}

```

ЗАДАЧА №887

Доказательство теоремы

(Время: 1 сек. Память: 16 Мб Сложность: 47%)

Преподаватель читает курс лекций, в рамках которого обычно доказывается N различных теорем. Некоторые теоремы могут ссылаться в доказательстве друг на друга. Более точно, каждая теорема T_i зависит от некоторого набора из S_i других теорем; доказать ее можно лишь доказав не менее половины теорем из данного набора. При этом структура курса такова, что нет такой теоремы, от которой зависели бы две или более различных теоремы, а также нет цепочки теорем $(T_{i_1}, T_{i_2}, \dots, T_{i_s})$ такой, что T_{i_1} зависит от T_{i_2} , T_{i_2} зависит от T_{i_3} , ..., $T_{i_{s-1}}$ зависит от T_{i_s} , а T_{i_s} — от T_{i_1} .

Однако, в этом семестре в связи с обилием праздников, перекрывающихся с лекциями, может не удастся доказать все теоремы курса. Тем не менее, нужно доказать основную теорему курса — это центральный результат всей теории, и именно его, скорее всего, придется применять слушателям в других курсах в следующем семестре. Поэтому преподаватель хочет расположить теоремы в таком порядке, чтобы основную теорему курса удалось доказать как можно раньше. Затем, если останется время, он сможет вернуться к доказательству других, менее важных теорем.

Для простоты будем считать, что все теоремы доказываются за одинаковое время. Нужно доказать такое множество теорем и в таком порядке, чтобы основная теорема оказалась доказанной и чтобы общее время доказательства было минимально.

Входные данные

В первой строке входного файла INPUT.TXT записано число N ($1 \leq N \leq 10\,000$) — количество теорем. Каждая из следующих N строк описывает теоремы, от которых зависит T_{i-1} , где i — номер этой строки во входном файле. Эти строки имеют вид $A_{i,1} A_{i,2} \dots A_{i,C_i} 0$; здесь $A_{i,j}$ — номер теоремы, от которой зависит T_{i-1} . Среди всех чисел $A_{i,j}$ во входном файле нет двух одинаковых. Основная теорема имеет номер 1. Все числа во входном файле целые.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите K — минимальное количество теорем, которые потребуются доказать. В последующих K строках выведите номера этих теорем в порядке их доказательства, по одному числу в каждой. Если ответов с минимальным K несколько, можно вывести любой из них.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 2 0 0	2 2 1

2	6 2 3 6 0 4 0 0 0 0 5 0	4 4 3 2 1
3	3 0 1 0 2 0	1 1

```

/*
    "Доказательство теоремы": деревья, поиск в глубину
*/

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <functional>

int main() {
    int nVert;
    scanf("%d", &nVert);

    // Чтение ребер:
    std::vector<std::vector<int>> edges(1+nVert);

    for (int id = 1; id <= nVert; ++id) {
        int value;
        while (scanf("%d", &value), value) {
            edges[id].push_back(value);
        }
    }

    // Поиск в глубину для нахождения множеств теорем, которые нужно доказать:
    std::vector<std::vector<int>> from(1+nVert);

    std::function<int(int,int)> visit = [&](const int curr, const int parent) {
        std::vector<std::pair<int,int>> values;
        for (int next : edges[curr]) {
            if (next != parent) {
                values.push_back(std::make_pair(visit(next, curr), next));
            }
        }
        std::sort(values.begin(), values.end());
        int sum = 0;
        for (int i = 0; 2 * from[curr].size() < values.size(); ++i) {
            from[curr].push_back(values[i].second);
            sum += values[i].first + 1;
        }
        return sum;
    };

    visit(1,0);

    // Формирование ответа:
    std::vector<int> answ;

    std::function<void(int)> get_answ = [&](const int curr) {
        for (int next : from[curr]) {
            get_answ(next);
        }
        answ.push_back(curr);
    };

```

```
get_answ(1);

// Вывод ответа:
printf("%d\n", (int)answ.size());
for (auto& it : answ) {
    printf("%d\n", it);
}

return 0;
}
```

ЗАДАЧА №927

$$A + B = C$$

(Время: 2 сек. Память: 16 Мб Сложность: 47%)

Часто для пробного тура на различных олимпиадах по информатике предлагается задача « $A + B$ », в которой по заданным целым числам A и B требуется найти их сумму.

При проведении городской олимпиады по информатике председатель жюри решил сам подготовить тесты для такой задачи. Для этого он использовал свою оригинальную методику, которая заключалась в следующем: сначала готовятся предполагаемые правильные ответы, а затем подбираются входные данные, соответствующие этим ответам.

Пусть председатель жюри выбрал число C , запись которого состоит из n десятичных цифр и не начинается с нуля. Теперь он хочет подобрать такие целые положительные числа A и B , чтобы их сумма была равна C , и запись каждого из них также состояла из n десятичных цифр и не начиналась с нуля. В дополнение к этому председатель жюри старается подобрать такие числа A и B , чтобы каждое из них было красивым. Красивым в его понимании является число, запись которого не содержит двух одинаковых подряд идущих цифр. Например, число 1272 считается красивым, а число 1227 — нет.

Требуется написать программу, которая для заданного натурального числа C вычисляет количество пар красивых положительных чисел A и B , сумма которых равна C . Поскольку количество пар красивых чисел может быть большим, необходимо вывести остаток от деления этого количества на число $10^9 + 7$.

Входные данные

Входной файл INPUT.TXT содержит одно целое положительное число C . Число C не начинается с нуля. Количество цифр в записи числа C не превышает 10 000.

Выходные данные

В выходной файл OUTPUT.TXT должен содержать одно целое число — остаток от деления количества искомых пар красивых чисел A и B на число $10^9 + 7$.

Примеры

Пояснения к примерам

Число 22 можно представить в виде суммы двузначных чисел тремя способами: $10 + 12$, $11 + 11$, $12 + 10$. Способ $11 + 11$ не подходит, поскольку число 11 не является красивым. Следовательно, ответ для числа 22 равен 2.

Число 200 можно представить в виде суммы трехзначных чисел единственным способом: $100 + 100$. Этот способ не подходит, поэтому ответ для числа 200 равен 0.

Число 1000 нельзя представить в виде суммы четырехзначных чисел, поэтому ответ для числа 1000 аналогично равен 0.

№	INPUT.TXT	OUTPUT.TXT
1	22	2
2	200	0
3	1000	0
4	239	16

```

/*
    "A + B = C": динамическое программирование
*/

#include <iostream>
#include <string>
#include <algorithm>

const int MOD = (int)1e9+7;

inline int add(int a, int b) {
    return (a += b) >= MOD ? a - MOD : a;
}

const bool debug = false;

int main() {
    std::string s;
    std::cin >> s;

    std::reverse(s.begin(), s.end());

    static int count[10001][10][10][2]; // [len][digA][digB][carry]

    for (int digA = 0; digA < 10; ++digA) {
        for (int digB = 0; digB < 10; ++digB) {
            if ((digA + digB) % 10 == s[0] - '0') {
                ++count[1][digA][digB][(digA+digB) / 10];
            }
        }
    }

    for (int len = 2; len <= (int)s.size(); ++len) {
        for (int currA = 0; currA < 10; ++currA) {
            for (int currB = 0; currB < 10; ++currB) {
                for (int carry = 0; carry < 2; ++carry) {
                    int digit = currA + currB + carry;
                    int rem = digit / 10;
                    digit %= 10;
                    if (digit != s[len-1] - '0') continue;
                    for (int prevA = 0; prevA < 10; ++prevA) {
                        if (prevA == currA) continue;
                        for (int prevB = 0; prevB < 10; ++prevB) {
                            if (prevB == currB) continue;
                            count[len][currA][currB][rem] = add(count[len][prevA][prevB][rem], count[len][currA][currB][rem]);
                        }
                    }
                }
            }
        }
    }

    int ans = 0;
    for (int digA = 1; digA < 10; ++digA) {
        for (int digB = 1; digB < 10; ++digB) {
            ans = add(ans, count[s.size()][digA][digB][0]);
        }
    }
    printf("%d", ans);

    return 0;
}

```


ЗАДАЧА №965

Мафия в городе

(Время: 1 сек. Память: 16 Мб Сложность: 47%)

Об этом еще никто не знает, но многие догадываются – мафия уже в городе. Поговаривают, что в планах главы мафиозного клана захват контроля над всем городом, однако поначалу он решил ограничиться захватом основных линий связи города.

В городе находятся n базовых телефонных станций, некоторые пары которых соединены двусторонними каналами связи. Для удобства, занумеруем базовые станции целыми числами от 1 до n , канал связи в этом случае задается парой чисел (u, v) – номерами станций, которые он соединяет.

Будем говорить, что канал связи (u, v) – контролируется мафией, если захвачена, либо станция u , либо станция v (либо обе).

Глава мафиозного клана хочет контролировать все каналы связи, захватив при этом как можно меньше базовых станций. Ваша задача помочь службе безопасности телефонной компании, составив возможный план захвата и определив количество таких планов.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа: n и m ($2 \leq n \leq 18, 0 \leq m$). Каждая из последующих m строк описывает один канал связи и содержит по два целых числа: u и v ($1 \leq u, v \leq n, u \neq v$) – номера базовых станций, соединенных этим каналом связи. Любая пара станций соединена не более, чем одним каналом.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите два числа: k и s – соответственно, минимальное количество базовых станций, которые необходимо захватить для того, чтобы контролировать все каналы связи, и число способов захватить такое количество станций так, чтобы контролировать все каналы связи.

Во второй строке выведите k чисел – номера базовых станций, соответствующих одному из способов захвата.

Примеры

Пояснение

В первом примере существует три способа захватить две станции так, чтобы контролировать все каналы связи: $\{1, 2\}, \{1, 3\}, \{2, 3\}$.

№	INPUT.TXT	OUTPUT.TXT
1	3 3 1 2 2 3 3 1	2 3 1 2

2	5 4	1 1 1
	1 2	
	1 3	
	1 4	
	1 5	

```

/*
"Мафия в городе": графы, рекурсивный перебор
*/

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <functional>

int main() {
    int nVert, nEdges;
    scanf("%d %d", &nVert, &nEdges);

    std::vector<int> a(nEdges), b(nEdges);

    std::vector<std::vector<int>> edges(1+nVert);

    for (int i = 0; i < nEdges; ++i) {
        scanf("%d %d", &a[i], &b[i]);
        if (a[i] > b[i]) std::swap(a[i], b[i]);
        edges[a[i]].push_back(i);
        edges[b[i]].push_back(i);
    }

    std::vector<bool> used(nEdges, false);

    std::vector<bool> set(1+nVert, false);

    int minCount = nVert, nUsed = 0, answer = 1;

    std::vector<bool> minSet(1+nVert, true);

    std::function<void(int,int)> go = [&](const int curr, const int count) {
        if (nUsed == nEdges) {
            // Пересчитать ответ
            if (count < minCount) {
                minCount = count;
                answer = 1;
                minSet = set;
            } else if (count == minCount) {
                answer++;
            }
            return;
        }
        if (curr > nVert) {
            return;
        }
        // Берем текущую вершину
        for (auto& e : edges[curr]) {
            if (!used[e]) {
                used[e] = true;
                ++nUsed;
            }
        }
        set[curr] = true;
        go(curr+1, count+1);

        // Возвращаем все на место и не берем текущую вершину
        set[curr] = false;
        for (auto& e : edges[curr]) {
            if (used[e] && !set[a[e]] && !set[b[e]]) {

```

```
        used[e] = false;
        --nUsed;
    }
}
go(curr+1, count);
};

go(1, 0);

printf("%d %d\n", minCount, answer);
for (int i = 1; i <= nVert; ++i) {
    if (minSet[i]) {
        printf("%d ", i);
    }
}
return 0;
}
```

ЗАДАЧА №985

Объявление массивов

(Время: 2 сек. Память: 16 Мб Сложность: 47%)

В языке Java для удобства работы любой объявленный массив является объектом, который содержит некоторую вспомогательную информацию и собственно сам массив.

Рассмотрим внутреннюю структуру такого объекта. В нем содержится 16 байт вспомогательной информации, 4 байта уходит на хранение длины массива, а затем сам массив, который занимает $\text{length} * \text{size}$ памяти, где length – это его длина, а size – размер объектов хранящихся в массиве.

Если мы объявляем в программе двумерный массив, то у нас создается объект, который будет содержать в себе массив, каждый элемент которого будет являться одномерным массивом. Так же не следует забывать, что на каждый объект создается внешняя ссылка размером 4 байта.

Аналогично, при объявлении массивов размерности k будет создан массив, каждым элементом которого является массив размерности $k-1$.

Ваша задача – по данным о размерах примитивных типов, из которых состоят массивы, выяснить, сколько памяти занимает каждый из заданных массивов.

Входные данные

В первой строке входного файла INPUT.TXT содержится количество различных примитивных типов N ($1 \leq N \leq 100$). В следующих N строках содержатся названия примитивных типов, состоящие из маленьких английских букв, длиной не более 30 символов, и через пробел размер типа в байтах.

Размеры типов не превосходят $2^{31} - 1$ байт. Следующая строка содержит число объявленных массивов M ($1 \leq M \leq 1000$). В следующих M строках содержатся описания массивов в формате «type name[number₁] ... [number_k];», где type – один из примитивных типов, описанных выше, name – имя массива, состоящее из маленьких английских букв, длиной не более 30 символов, $1 \leq \text{number}_i \leq 1000$. Размерность массива не превышает 50.

Выходные данные

В выходной файл OUTPUT.TXT выведите в M строках информацию об объеме памяти в байтах, занимаемым каждым из массивов, согласно последовательности, определенной во входных данных.

Пример

Пояснение

Первый массив занимает 4 байта на внешнюю ссылку, 16 вспомогательных байт, 4 байта на хранение длины массива и 1 байт на сам массив. Итого 25 байт.

Третий массив занимает 4 байта на внешнюю ссылку, 16 вспомогательных байт, 4 байта на хранение длины массива и $100 * \text{size}$ байт на массив, где size – размер одномерного массива из 100 чисел типа int. В данном случае $\text{size} = 4 + 16 + 4 + 4 * 100$. Итого 42424 байт.

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	4 byte 1 short 2 int 4 long 8 4 byte a[1]; short b[2][2][2]; int c[100][100]; long a[177];	25 184 42424 1440
---	---	----------------------------

```

/*
"Объявление массивов": длинная арифметика, умножение, сложение
*/

#include <stdio.h>
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <cstdint>
#include <cassert>
#include <functional>
#include <complex>
#include <sstream>
#include <map>

const long double PI = std::acos(-1.0L);

struct UInt {
    static const int BASE = (int)1e9; // Основание системы счисления
    static const int WIDTH = 9;      // Количество десятичных цифр, которые хранятся в одн

    // Вектор под цифры числа:
    std::vector<int> digits;

    // Конструкторы
    UInt(int64_t number = 0);
    UInt(const std::string& s);
    UInt(const std::vector<int>& digits);

    // Методы нормализации и сравнения:
    UInt& normalize(); // удаление лидирующих нулей и проверка на принадлежность цифр диапа
    int compare(const UInt& other) const; // Сравнение (меньше = -1, равно = 0, больше = 1)

    // Методы умножения:
    UInt slow_mult(const UInt& other) const; // Медленное произведение (работает довольно б
    UInt fast_mult(const UInt& other) const; // Быстрое произведение (на основе Быстрого Пр
    UInt mult(const UInt& other) const; // Комбинированный метод умножения на основе экспер

    // Метод деления:
    std::pair<UInt, UInt> div_mod(const UInt& other) const; // Целая часть и остаток от дел

    // Операторы:
    UInt& operator+=(const int num); // Прибавление короткого
    UInt& operator+=(const UInt& other); // Прибавление длинного
    UInt& operator-=(const int num); // Вычитание короткого
    UInt& operator-=(const UInt& other); // Вычитание длинного
    UInt& operator*=(const int num); // Умножение на короткое
    UInt& operator*=(const UInt& other); // Умножение на длинное
    UInt& operator/=(const int num); // Деление на короткое
    UInt& operator/=(const UInt& other); // Деление на длинное
    UInt& operator%=(const UInt& other); // Остаток от деления на длинное
};

std::istream& operator>>(std::istream&, UInt&); // Ввод из потока
std::ostream& operator<<(std::ostream&, const UInt&); // Вывод в поток

```

```

UInt pow(UInt, int); // Возведение в степень
UInt gcd(UInt, UInt); // Наибольший общий делитель

UInt operator+(const UInt&, const UInt&);
UInt operator-(const UInt&, const UInt&);
UInt operator*(const UInt&, const UInt&);
UInt operator/(const UInt&, const UInt&);
UInt operator%(const UInt&, const UInt&);

UInt operator+(const UInt&, const int);
UInt operator+(const int, const UInt&);
UInt operator-(const UInt&, const int);
UInt operator*(const UInt&, const int);
UInt operator*(const int, const UInt&);
UInt operator/(const UInt&, const int);
UInt operator^(const UInt&, const int); // возведение в степень

bool operator<(const UInt&, const UInt&);
bool operator>(const UInt&, const UInt&);
bool operator<=(const UInt&, const UInt&);
bool operator>=(const UInt&, const UInt&);
bool operator==(const UInt&, const UInt&);
bool operator!=(const UInt&, const UInt&);

UInt& UInt::normalize() {
    while (digits.back() == 0 && (int)digits.size() > 1) digits.pop_back();
    for (auto d : digits) assert(0 <= d && d < BASE);
    return *this;
}

// Конструктор от короткого целого
UInt::UInt(int64_t number) {
    assert(number >= 0);
    do {
        digits.push_back(number % BASE);
        number /= BASE;
    } while (number > 0);
    normalize();
}

// Конструктор от вектора из цифр:
UInt::UInt(const std::vector<int>& digits) : digits(digits) {
    normalize();
}

// Конструктор от строки:
UInt::UInt(const std::string& s) {
    const int size = (int)s.size();
    for (int idGroup = 1, nGroups = size / WIDTH; idGroup <= nGroups; ++idGroup) {
        digits.push_back(std::stoi(s.substr(size-idGroup * WIDTH, WIDTH)));
    }
    if (size % WIDTH != 0) {
        digits.push_back(std::stoi(s.substr(0, size % WIDTH)));
    }
    normalize();
}

// Прибавление короткого:
UInt& UInt::operator+=(const int num) {
    assert(num >= 0);
    if (num >= BASE) {
        return *this += UInt(num);
    }
    int rem = num;
    for (int i = 0; rem > 0; ++i) {
        if (i >= (int)digits.size()) digits.push_back(0);
        rem += digits[i];
        if (rem >= BASE) {

```

```

        digits[i] = rem - BASE;
        rem = 1;
    } else {
        digits[i] = rem;
        rem = 0;
    }
}
return this->normalize();
}

// Прибавление длинного:
UInt& UInt::operator+=(const UInt& other) {
    if (other.digits.size() == 1u) {
        return *this += other.digits[0];
    }
    const int s1 = this->digits.size();
    const int s2 = other.digits.size();
    int rem = 0;
    for (int i = 0; i < s1 || i < s2 || rem > 0; ++i) {
        int d1 = i < s1 ? this->digits[i] : (digits.push_back(0), 0);
        int d2 = i < s2 ? other.digits[i] : 0;
        rem += d1 + d2;
        auto div = rem / BASE;
        digits[i] = rem - div * BASE;
        rem = div;
    }
    return this->normalize();
}

// Вычитание короткого:
UInt& UInt::operator-=(const int num) {
    assert(num >= 0);
    if (num >= BASE) {
        return *this -= UInt(num);
    }
    int rem = -num;
    for (int i = 0; i < (int)digits.size() && rem < 0; ++i) {
        rem += digits[i];
        if (rem < 0) { // Занимаем разряд
            digits[i] = rem + BASE;
            rem = -1;
        } else {
            digits[i] = rem;
            rem = 0;
        }
    }
    assert(rem == 0);
    return this->normalize();
}

// Вычитание длинного:
UInt& UInt::operator-=(const UInt& other) {
    if (other.digits.size() == 1u) {
        return *this -= other.digits[0];
    }
    const int s1 = this->digits.size();
    const int s2 = other.digits.size();
    assert(s1 >= s2);
    int rem = 0;
    for (int i = 0; i < s1; ++i) {
        int d2 = i < s2 ? other.digits[i] : 0;
        rem += this->digits[i] - d2;
        if (rem < 0) {
            digits[i] = rem + BASE;
            rem = -1;
        } else {
            digits[i] = rem;
            rem = 0;
            if (i >= s2) break;
        }
    }

```



```

    }
}
assert(rem == 0); // Иначе *this < other
return this->normalize();
}

// Умножение на короткое:
UInt& UInt::operator*=(const int num) {
    assert(num >= 0);
    if (num >= BASE) {
        return *this *= UInt(num);
    }
    int64_t rem = 0;
    for (auto& d : digits) {
        rem += 1LL * d * num;
        auto div = rem / BASE;
        d = rem - div * BASE;
        rem = div;
    }
    if (rem > 0) digits.push_back(rem);
    return this->normalize();
}

// Медленное произведение:
UInt UInt::slow_mult(const UInt& other) const {
    if (other.digits.size() == 1u) {
        return *this * other.digits[0];
    }
    const int s1 = (int)this->digits.size();
    const int s2 = (int)other.digits.size();
    std::vector<int> temp(s1+s2);
    for (int i = 0; i < s1; ++i) {
        int64_t rem = 0;
        for (int j = 0; j < s2; ++j) {
            rem += temp[i+j] + 1LL * this->digits[i] * other.digits[j];
            auto div = rem / BASE;
            temp[i+j] = rem - div * BASE;
            rem = div;
        }
        if (rem > 0) temp[i+s2] += rem;
    }
    return UInt(temp);
}

// Быстрое умножение на основе быстрого преобразования Фурье:
UInt UInt::fast_mult(const UInt& other) const {
    if (other.digits.size() == 1u) {
        return *this * other.digits[0];
    }

    // Разворот битов в числе num:
    std::function<int(int, int)> reverse = [](int number, int nBits) {
        int res = 0;
        for (int i = 0; i < nBits; ++i) {
            if (number & (1 << i)) {
                res |= 1 << (nBits-1-i);
            }
        }
        return res;
    };

    typedef std::complex<long double> complex;
    // Быстрое преобразование Фурье:
    std::function<void(std::vector<complex>&, bool)> fft = [&reverse](std::vector<complex>
        const int n = (int)a.size();
        int nBits = 0;
        while ((1 << nBits) < n) ++nBits;

        for (int i = 0; i < n; ++i) {

```

```

        if (i < reverse(i, nBits)) {
            std::swap(a[i], a[reverse(i, nBits)]);
        }
    }

    for (int len = 2; len <= n; len <= 1) {
        auto ang = 2*PI / len * (invert ? -1 : 1);
        complex wlen (std::cos(ang), std::sin(ang));
        for (int i = 0; i < n; i += len) {
            complex w(1);
            for (int j = 0; j < len / 2; ++j) {
                complex u = a[i+j];
                complex v = a[i+j+len / 2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert) {
        for (int i = 0; i < n; ++i) {
            a[i] /= n;
        }
    }
};

// Подготавливаем вектора из комплексных коэффициентов fa и fb:
// Так как происходит потеря точности из-за арифметики с плавающей точкой, основание си
assert(BASE == 1000 * 1000 * 1000);
std::function<std::vector<complex>(const UInt&)> prepare = [] (const UInt& number) {
    std::vector<complex> result;
    result.reserve(3 * number.digits.size());
    for (auto d : number.digits) {
        result.push_back(d % 1000);
        result.push_back(d / 1000 % 1000);
        result.push_back(d / 1000000);
    }
    return result;
};

auto fa = prepare(*this);
auto fb = prepare(other);

// Округляем размер векторов до ближайшей степени двойки:
int n = 1;
while ((n < (int)std::max(fa.size(), fb.size())) n *= 2;
n *= 2;
fa.resize(n);
fb.resize(n);

// Вызываем прямое преобразование Фурье:
fft (fa, false);
fft (fb, false);
// Перемножаем результаты:
for (int i = 0; i < n; ++i) {
    fa[i] *= fb[i];
}
// Вызываем обратное преобразование Фурье:
fft (fa, true);
// Копируем ответ с округлениями:
std::vector<int64_t> temp(n);
for (int i = 0; i < (int)fa.size(); ++i) {
    temp[i] = int64_t (fa[i].real() + 0.5);
}
// Не забываем про переносы в старшие разряды:
int64_t carry = 0;
for (int i = 0; i < n || carry > 0; ++i) {
    if (i >= n) temp.push_back(0);
    temp[i] += carry;

```

```

        carry = temp[i] / 1000;
        temp[i] -= carry * 1000;
        assert(temp[i] >= 0);
    }
    // Формируем ответ:
    std::vector<int> res;
    res.reserve(this->digits.size() + other.digits.size());

    for (int i = 0; i < n; i += 3) {
        int c = temp[i];
        int b = i+1 < n ? temp[i+1] : 0;
        int a = i+2 < n ? temp[i+2] : 0;
        res.push_back(c + 1000 * (b + 1000 * a));
    }
    return UInt(res);
}

// Комбинированный метод умножения:
UInt UInt::mult(const UInt& other) const {
    // Выбор метода умножения:
    int len1 = (int)this->digits.size();
    int len2 = (int)other.digits.size();
    int temp = 3 * std::max(len1, len2);
    int pow = 1;
    while (pow < temp) pow *= 2;
    pow *= 2;
    int op1 = len1 * len2;
    int op2 = 3 * pow * std::log(pow) / std::log(2);
    return op1 >= 15 * op2 ? fast_mult(other) : slow_mult(other);
}

// Деление на короткое:
UInt& UInt::operator/=(const int num) {
    assert(num > 0);
    if (num >= BASE) {
        return *this /= UInt(num);
    }
    int64_t rem = 0;
    for (int j = (int)digits.size()-1; j >= 0; --j) {
        (rem *= BASE) += digits[j];
        auto div = rem / num;
        digits[j] = div;
        rem -= div * num;
    }
    return this->normalize();
}

// Остаток от деления на короткое:
int operator%(const UInt& a, const int num) {
    assert(num > 0);
    int64_t rem = 0;
    for (int i = (int)a.digits.size()-1; i >= 0; --i) {
        ((rem *= UInt::BASE) += a.digits[i]) %= num;
    }
    return rem;
}

// Целая часть и остаток от деления:
std::pair<UInt, UInt> UInt::div_mod(const UInt& other) const {
    if (other.digits.size() == 1u) {
        return {std::move(*this / other.digits[0]), *this % other.digits[0]};
    }
    const int norm = BASE / (other.digits.back() + 1);
    const UInt a = *this * norm;
    const UInt b = other * norm;
    const int a_size = (int)a.digits.size();
    const int b_size = (int)b.digits.size();
    UInt q, r;
    q.digits.resize(a_size);

```

```

    for (int i = a_size - 1; i >= 0; --i) {
        r *= BASE;
        r += a.digits[i];
        int s1 = (int)r.digits.size() <= b_size ? 0 : r.digits[b_size];
        int s2 = (int)r.digits.size() <= b_size - 1 ? 0 : r.digits[b_size - 1];
        int d = (1LL * BASE * s1 + s2) / b.digits.back();
        auto temp = b * d;
        while (r < temp) {
            r += b;
            --d;
        }
        r -= temp;
        q.digits[i] = d;
    }
    return {std::move(q.normalize()), std::move(r /= norm)};
}

// Сравнение: result < 0 (меньше), result == 0 (равно), result > 0 (больше)
int UInt::compare(const UInt& other) const {
    if (this->digits.size() > other.digits.size()) return 1;
    if (this->digits.size() < other.digits.size()) return -1;
    for (int i = (int)digits.size()-1; i >= 0; --i) {
        if (this->digits[i] > other.digits[i]) return 1;
        if (this->digits[i] < other.digits[i]) return -1;
    }
    return 0;
}

// Операторы сравнения:
bool operator< (const UInt& a, const UInt& b) { return a.compare(b) < 0; }
bool operator> (const UInt& a, const UInt& b) { return a.compare(b) > 0; }
bool operator==(const UInt& a, const UInt& b) { return a.compare(b) == 0; }
bool operator<=(const UInt& a, const UInt& b) { return a.compare(b) <= 0; }
bool operator>=(const UInt& a, const UInt& b) { return a.compare(b) >= 0; }
bool operator!=(const UInt& a, const UInt& b) { return a.compare(b) != 0; }

// Ввод из потока:
std::istream& operator>>(std::istream& is, UInt& number) {
    std::string s;
    is >> s;
    number = UInt(s);
    return is;
}

// Вывод в поток:
std::ostream& operator<<(std::ostream& os, const UInt& number) {
    os << number.digits.back();
    for (int i = (int)number.digits.size()-2; i >= 0; --i) {
        os << std::setw(UInt::WIDTH) << std::setfill('0') << number.digits[i];
    }
    return os << std::setfill(' ');
}

// Сумма:
UInt operator+(const UInt& a, const UInt& b) {
    return UInt(a) += b;
}

// Разность:
UInt operator-(const UInt& a, const UInt& b) {
    return UInt(a) -= b;
}

// Произведение:
UInt operator*(const UInt& a, const UInt& b) {
    return a.mult(b);
}

// Деление:

```

```

UInt operator/(const UInt& a, const UInt& b) {
    return a.div_mod(b).first;
}

// Взятие остатка:
UInt operator%(const UInt& a, const UInt& b) {
    return a.div_mod(b).second;
}

// Умножение:
UInt& UInt::operator*=(const UInt& other) {
    return *this = *this * other;
}

// Деление с присваиванием:
UInt& UInt::operator/=(const UInt& other) {
    return *this = *this / other;
}

// Взятие остатка с присваиванием:
UInt& UInt::operator%=(const UInt& other) {
    return *this = *this % other;
}

UInt operator+(const UInt& a, const int b) { return UInt(a) += b; }
UInt operator+(const int a, const UInt& b) { return b * a; }
UInt operator-(const UInt& a, const int b) { return UInt(a) -= b; }
UInt operator*(const UInt& a, const int b) { return UInt(a) *= b; }
UInt operator*(const int a, const UInt& b) { return b * a; }
UInt operator/(const UInt& a, const int b) { return UInt(a) /= b; }
UInt operator^(const UInt& a, const int n) { return pow(a, n); } // Возведение в степень

// Возведение в степень:
UInt pow(UInt a, int n) {
    UInt res = 1;
    while (n > 0) {
        if (n % 2 != 0) res *= a;
        a *= a;
        n /= 2;
    }
    return res;
}

// Наибольший общий делитель:
UInt gcd(UInt a, UInt b) {
    while (b != 0) {
        auto rem = a % b;
        a = b;
        b = rem;
    }
    return a;
}

typedef UInt Int;

int main() {
    int nTypes;
    scanf("%d", &nTypes);

    std::map<std::string, int> size_of;

    while (nTypes--) {
        char buf[31];
        int size;
        scanf("%30s %d", buf, &size);
        size_of[buf] = size;
    }

    int nQueries;

```

```

scanf("%d", &nQueries);

while (nQueries--) {
    char buf[1000];
    scanf("%30s", buf);
    int size = sizeof(buf);

    scanf("%s", buf);
    std::string s(buf);

    Int total = 0, mult = 1;

    int pos = 0;
    while (pos < (int)s.size()) {
        while (pos < (int)s.size() && !('0' <= s[pos] && s[pos] <= '9')) ++pos;
        int value = 0;
        while (pos < (int)s.size() && '0' <= s[pos] && s[pos] <= '9') {
            (value *= 10) += (s[pos] - '0');
            ++pos;
        }
        if (pos < (int)s.size()) {
            total += mult * (4 + 16 + 4);
            mult *= value;
        }
    }
    total += mult * size;
    std::stringstream ss;
    ss << total;
    printf("%s\n", ss.str().c_str());
}

return 0;
}

```

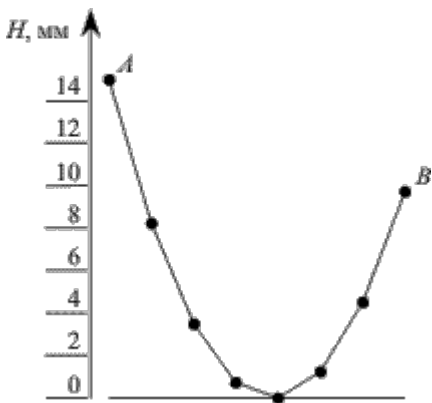
ЗАДАЧА №1333

Гирлянда

(Время: 1 сек. Память: 16 Мб Сложность: 47%)

Гирлянда состоит из N лампочек на общем проводе. Один её конец закреплён на заданной высоте A мм ($H_1 = A$). Благодаря силе тяжести гирлянда прогибается: высота каждой неконцевой лампы на 1 мм меньше, чем средняя высота ближайших соседей ($H_i = (H_{i-1} + H_{i+1}) / 2 - 1$ для $1 < i < N$).

Требуется найти минимальную высоту второго конца B ($B = H_N$) при условии, что ни одна из лампочек не должна лежать на земле ($H_i > 0$ для $1 \leq i \leq N$).



Входные данные

Входной файл INPUT.TXT содержит два числа N и A ($3 \leq N \leq 1000$ - целое, $10 \leq A \leq 1000$ - вещественное).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно вещественное число B с двумя знаками после запятой.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	8 15	9.75
2	692 532.81	446113.34

```
/*
    Бинарный поиск по правому концу.
    Для фиксированного правого конца можно решить трехдиагональную систему методом прогонки
*/
#include <iostream>
#include <iomanip>
#include <vector>
#include <cassert>

typedef long double Real;

std::vector<Real> solve_tridiag(int n, Real L, Real R) {
    // n - кол-во узлов, L и R - высоты левого и правого концов соответственно
    if (n == 3) {
        return {L, (L+R)/2-1, R};
    }
    /*
        Общий вид уравнений:
        x[i-1] - 2 x[i] + x[i+1] = 2

        Для метода прогонки:
        a[i] * x[i-1] + c[i] * x[i] + b[i] * x[i+1] = f[i]
```

```

    При i = 1:
        -2 * x[i] + x[i+1] = 2 - L

    При i = n-2:
        x[i-1] - 2 * x[i] = 2 - R
*/

// Массивы коэффициентов системы:
std::vector<Real> a(n-2, 1), b(n-2, 1), c(n-2, -2), f(n-2, 2);
a.front() = b.back() = 0;
f.front() -= L;
f.back() -= R;

// Прямой ход - вычисляем прогоночные коэффициенты:
std::vector<Real> alpha{0}, beta{0};
alpha.push_back(-b[0] / c[0]);
beta.push_back(f[0] / c[0]);
for (int i = 1; i < (int)c.size(); ++i) {
    assert(i == (int)alpha.size()-1);
    auto q = a[i] * alpha[i] + c[i];
    alpha.push_back(-b[i] / q);
    beta.push_back((f[i] - a[i] * beta[i]) / q);
}

// Обратный ход - вычисление решения системы:
std::vector<Real> x(n-2);
x[n-3] = (f[n-3] - a[n-3] * beta[n-3]) / (c[n-3] + a[n-3] * alpha[n-3]);
for (int i = n-4; i >= 0; --i) {
    x[i] = alpha[i+1] * x[i+1] + beta[i+1];
}
x.insert(x.begin(), L);
x.insert(x.end(), R);
return x;
}

int main() {
    int n; Real L;
    std::cin >> n >> L;

    Real low = -1, high = 1000*1000;
    const Real eps = 1e-3;
    while (high - low > eps) {
        auto mid = (low + high) / 2;
        auto x = solve_tridiag(n, L, mid);
        bool flag = true;
        for (auto it : x) {
            if (it <= 0) {
                flag = false;
                break;
            }
        }
        if (flag) {
            high = mid;
        } else {
            low = mid;
        }
    }
    std::cout << std::fixed << std::setprecision(2) << high << std::endl;
    return 0;
}

```


ЗАДАЧА №1334

Головоломка умножения

(Время: 1 сек. Память: 16 Мб Сложность: 47%)

В головоломку умножения играют с рядом карт, каждая из которых содержит одно положительное целое число. Во время хода игрок убирает одну карту из ряда и получает число очков, равное произведению числа на убранной карте и чисел на картах, лежащих непосредственно слева и справа от неё. Не разрешено убирать первую и последнюю карты ряда. После последнего хода в ряду остаётся только две карты.

Цель игры – убрать карты в таком порядке, чтобы минимизировать общее количество набранных очков.

Например, если карты содержат числа 10, 1, 50, 20 и 5, игрок может взять карту с числом 1, затем 20 и 50, получая очки:

$$10 * 1 * 50 + 50 * 20 * 5 + 10 * 50 * 5 = 500 + 5000 + 2500 = 8000.$$

Если бы он взял карты в обратном порядке, то есть 50, затем 20, затем 1, количество очков было бы таким:

$$1 * 50 * 20 + 1 * 20 * 5 + 10 * 1 * 5 = 1000 + 100 + 50 = 1150.$$

Входные данные

В первой строке входного файла INPUT.TXT находится число карт N ($3 \leq N \leq 100$), во второй – разделённые пробелами N чисел на картах (целые числа в диапазоне от 1 до 100).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число – минимально возможное число очков.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	6 10 1 50 50 20 5	3650

```
#include <iostream>
#include <functional>
#include <climits>

int main() {
    int n; std::cin >> n;

    static int arr[100];
    for (int i = 0; i < n; ++i) {
        std::cin >> arr[i];
    }

    static int min[100][100];
    for (int i = 0; i < 100; ++i) {
        for (int j = 0; j < 100; ++j) {
```

```

        min[i][j] = -1;
    }
}

std::function<int(int, int)> get = [&](const int left, const int right) {
    if (right <= left+1) {
        return 0;
    }
    if (min[left][right] == -1) {
        int value = INT_MAX;
        for (int p = left+1; p < right; ++p) {
            value = std::min(value, get(left, p) + get(p, right) + arr[left] * arr[righ
        ]
        min[left][right] = value;
    }
    return min[left][right];
};
std::cout << get(0, n-1);
return 0;
}

```

ЗАДАЧА №1462

Силовые поля

(Время: 2 сек. Память: 64 Мб Сложность: 47%)

В физико-биологической лаборатории исследуют воздействие излучения на растения при облучении через силовые поля.

Экспериментальная установка содержит квадратную платформу размером $10^9 \times 10^9$, заполненную плодородной почвой. Над платформой установлен источник излучения. Между источником излучения и платформой можно включать n силовых полей.

Генератор силовых полей установлен над точкой $(0, 0)$. При этом i -е силовое поле представляет собой прямоугольник со сторонами, параллельными границам платформы и координатами двух противоположных углов $(0, 0)$ и (x_i, y_i) .

В эксперименте планируется изучать воздействие излучения на растения при облучении через k силовых полей. Из заданных n полей необходимо выбрать k полей для эксперимента. Ученые хотят выбрать силовые поля таким образом, чтобы площадь участка платформы, над которой находятся все k выбранных силовых полей, была максимальна.

Требуется написать программу, которая по заданным целым числам n , k и описанию n силовых полей определяет, какие k силовых полей необходимо выбрать для эксперимента, чтобы площадь участка, покрытого всеми k силовыми полями, была максимальна, и выводит площадь этого участка.

Входные данные

Первая строка входного файла INPUT.TXT содержит целые числа n и k ($1 \leq k \leq n \leq 200\,000$) – общее количество силовых полей и количество силовых полей, которые необходимо выбрать для эксперимента.

Последующие n строк содержат по два целых числа x_i, y_i ($1 \leq x_i, y_i \leq 10^9$) – координаты дальнего от начала координат угла прямоугольного участка i -го силового поля.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число: максимальную площадь искомого участка.

Пример

Пояснение к примеру

На рис. 1 показаны пять силовых полей, заданных во входном файле. Оптимальный способ выбрать из них три поля для эксперимента показан на рис. 2.

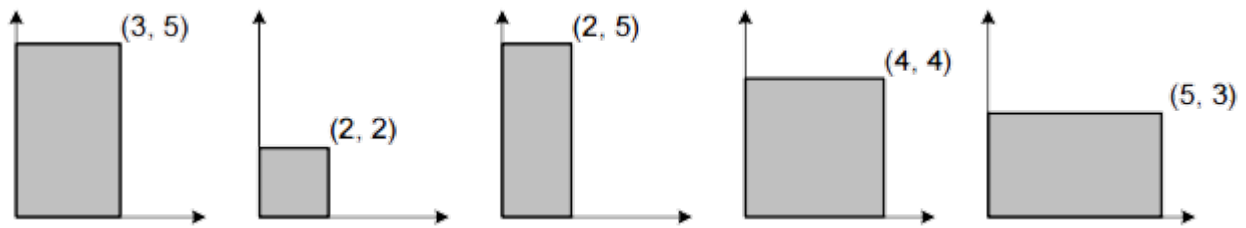


Рис 1. Силовые поля в примере описания входных данных.

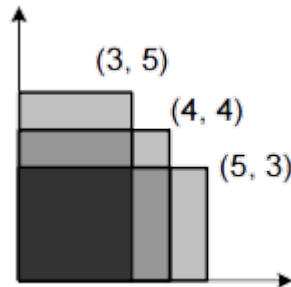


Рис 2. Оптимальный выбор трех из пяти силовых полей в данном примере.

№	INPUT.TXT	OUTPUT.TXT
1	5 3 3 5 2 2 2 5 4 4 5 3	9

```

/*
    Задача: 1462. Силовые поля

    Решение: сортировка, stack, set, структуры данных,  $O(n \log(n))$ 

    Автор: Дмитрий Козырев, https://github.com/dmkz , dmkozyrev@rambler.ru

    Отсортируем по `x` и `y`.
    Очевидно, что сторона искомого прямоугольника совпадет со стороной какого-то заданного
    Переберем все стороны y, поддерживая множество кандидатов среди `x`, у которых `y` >= т
*/

#include <iostream>
#include <algorithm>
#include <vector>
#include <set>

typedef long long ll;

struct Pair {
    int x, y;
};

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0); std::cout.tie(0); std::cerr.tie(0);

    int n, limit; std::cin >> n >> limit;

    auto cmpx = [](const Pair& a, const Pair& b){
        return a.x < b.x || (a.x == b.x && a.y < b.y);
    };

    auto cmpy = [](const Pair& a, const Pair& b){
        return a.y < b.y || (a.y == b.y && a.x < b.x);
    };

```

```

std::vector<Pair> ordered_x, ordered_y;
for (int i = 0; i < n; ++i) {
    int x, y; std::cin >> x >> y;
    ordered_x.push_back(Pair{x,y});
    ordered_y.push_back(Pair{x,y});
}
std::sort(ordered_x.begin(), ordered_x.end(), cmpx);
std::sort(ordered_y.begin(), ordered_y.end(), cmpy);

std::multiset<Pair, decltype(cmpx)> set(cmpx);

    ll answ = 0;

    for (int i = 0; i < n; ++i) {
        if (i != 0 && ordered_y[i].y != ordered_y[i-1].y) {
            for (int j = i-1; j >= 0 && ordered_y[j].y == ordered_y[i-1].y; --j) {
                if (set.find(ordered_y[j]) != set.end()) {
                    set.erase(ordered_y[j]);
                }
            }
        }
    }
    while ((int)set.size() < limit && !ordered_x.empty()) {
        auto p = ordered_x.back(); ordered_x.pop_back();
        if (p.y < ordered_y[i].y) continue;
        set.insert(p);
    }
    if ((int)set.size() == limit) {
        answ = std::max(answ, (ll) set.begin()->x * ordered_y[i].y);
    }
}
std::cout << answ;
return 0;
}

```

ЗАДАЧА №300

Радар

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Радар подвергается атаке из четырех точек, являющихся вершинами квадрата, в центре которого и стоит радар. Радар укомплектован специальным щитом, позволяющим блокировать удар, но щит может защищать радар только с одной из четырех сторон, и поворот щита требует времени. Изначально щит направлен в сторону той вершины, откуда будет первая атака. Известно время запуска и скорость ракет, ведущих атаку.

Требуется определить, сколько ракет удастся отбить.

Входные данные

Первые четыре строки входного файла INPUT.TXT содержат время запуска в секундах T_x ($0 \leq T_x \leq 1000$) и скорость полета в метрах в секунду V_x x -ой ракеты ($0 < V_x \leq 1000$). Ракеты перечисляются по часовой стрелке. Далее задано время в секундах, необходимое для поворота щита на 90 градусов $T_{\text{пов}}$ ($0 < T_{\text{пов}} \leq 1000$) и половина диагонали квадрата D – расстояние в метрах, предстоящее каждой из ракет ($0 < D < 1000$). Все числа – целые.

Выходные данные

В выходной файл OUTPUT.TXT выведите «ALIVE», если радар уцелеет при всех выстрелах, в противном случае следует вывести число успешно отраженных ракет. Если несколько ракет подлетают к радару одновременно, и радар может защититься от хотя бы одной из них, то он защищается от одной ракеты, и ее удар считается отраженным.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	0 10 5 10 10 10 15 10 5 100	ALIVE
2	0 10 10 10 5 10 15 10 5 100	1

```
/*
    Задача: 300. Радар

    Решение: перебор, геометрия, дроби, O(n! * n)

    Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
```

```

#define size(x) (int)(x).size()
typedef long long ll;

template<typename T> T gcd(T a, T b) { return b == 0 ? a : gcd(b, a % b); }

struct Frac {
    ll p, q;
    Frac(ll p_ = 0, ll q_ = 1) : p(p_), q(q_) {
        if (p == 0) { q = 1; }
        if (q < 0) { p = -p; q = -q; }
        assert(q != 0);
        ll g = gcd(std::abs(p), std::abs(q));
        p /= g; q /= g;
    }
    Frac operator*(const Frac& f) const { return Frac(p * f.p, q * f.q); }
    Frac operator+(const Frac& f) const { return Frac(p * f.q + f.p * q, q * f.q); }
    Frac operator-(const Frac& f) const { return Frac(p * f.q - f.p * q, q * f.q); }
    Frac operator/(const Frac& f) const { return Frac(p * f.q, q * f.p); }
};

bool operator<(const Frac& lhs, const Frac& rhs) {
    return (lhs - rhs).p < 0;
}

template<typename T> bool operator>(const T& lhs, const T& rhs) { return rhs < lhs; }
template<typename T> bool operator<=(const T& lhs, const T& rhs) { return !(lhs > rhs); }
template<typename T> bool operator>=(const T& lhs, const T& rhs) { return !(lhs < rhs); }
template<typename T> bool operator==(const T& lhs, const T& rhs) { return !(lhs < rhs || rhs < lhs); }
template<typename T> bool operator!=(const T& lhs, const T& rhs) { return !(lhs == rhs); }

struct Rocket {
    int t, v, s;
};

int getAngle(int curr, int next) {
    if (curr > next) { return getAngle(next, curr); }
    assert(curr <= next);
    return std::min(next - curr, 4 + curr - next);
}

int solve(const int tRot, const int dist, std::list<Rocket> rockets) {
    int answ = 1, side = rockets.begin()->s;
    Frac time = Frac(dist, rockets.begin()->v) + rockets.begin()->t;
    rockets.erase(rockets.begin());
    while (!rockets.empty()) {
        for (auto it = rockets.begin(); it != rockets.end(); it++) {
            if (Frac(dist, it->v) + it->t < time) {
                return answ-1;
            }
        }
        auto best = rockets.begin();
        if (Frac(time) + getAngle(side, best->s) * tRot > Frac(dist, best->v) + best->t) {
            break;
        }
        time = Frac(dist, best->v) + best->t;
        side = best->s;
        answ++;
        rockets.erase(best);
    }
    return answ;
}

int main() {
    std::vector<Rocket> arr(4);
    for (int i = 0; i < 4; ++i) {
        auto &it = arr[i];
        std::cin >> it.t >> it.v;
        it.s = i;
    }
}

```

```

int tRot, dist, ret = 0;
std::cin >> tRot >> dist;
std::vector<int> perm = {0,1,2,3};
do {
    std::list<Rocket> list;
    for (int i = 0; i < 4; ++i) {
        list.push_back(arr[perm[i]]);
    }
    ret = std::max(ret, solve(tRot, dist, list));
} while (std::next_permutation(all(perm)));
assert(ret > 0);
if (ret == 4) { std::cout << "ALIVE\n"; }
else { std::cout << ret << "\n"; }
return 0;
}

```


ЗАДАЧА №473

Автомобильные пробки

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Автомобильные пробки случаются везде, даже в нашем небольшом городе. Дороги у нас имеют по две полосы в одном направлении, а автомобили только двух видов: легковые (в пробке занимают квадратное место 1×1 от ширины одной полосы) и грузовые (занимают прямоугольное место 1×2). Автомобилисты очень дисциплинированы: не становятся поперек полосы, не занимают чужую площадь, но и не оставляют свободных мест.

Требуется написать программу, которая определит количество различных автомобильных пробок длины N .

Входные данные

Входной файл INPUT.TXT содержит одно натуральное число N ($N \leq 1000$).

Выходные данные

Выходной файл OUTPUT.TXT должен содержать найденное количество автомобильных пробок.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2	4
2	3	9

```
/*
    Две полосы независимы, поэтому решаем задачу отдельно для каждой полосы:
    f[1] = 1;
    f[2] = 2;
    f[n] = f[n-1] + f[n-2];
    Ответ: f[n] * f[n];
*/

#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <cstdint>
#include <cassert>
#include <functional>

struct Int;

std::istream& operator>>(std::istream&, Int&);
std::ostream& operator<<(std::ostream&, const Int&);
bool operator<(const Int&, const Int&);
bool operator>(const Int&, const Int&);
bool operator==(const Int&, const Int&);
bool operator<=(const Int&, const Int&);
bool operator>=(const Int&, const Int&);
bool operator!=(const Int&, const Int&);
Int operator+(const Int&, const Int&);
```

```

Int operator+(const int, const Int&);
Int operator-(const Int&, const Int&);
Int operator*(const int, const Int&);
Int operator/(const Int& a, const int);
int operator%(const Int& a, const int);
Int pow(Int, int);

struct Int {
    static const int BASE = (int)1e9; // Основание системы счисления
    static const int WIDTH = 9;      // Количество десятичных цифр, которые хранятся в одн

    // Вектор под цифры числа:
    std::vector<int> digits;

    // Нормализуем вид числа - удаляем лидирующие нули
    Int& to_normal() {
        while (digits.back() == 0 && (int)digits.size() > 1) {
            digits.pop_back();
        }
        return *this;
    }

    // Конструктор от короткого целого
    Int (int64_t number = 0) {
        assert(number >= 0);
        do {
            digits.push_back(number % BASE);
            number /= BASE;
        } while (number > 0);
        to_normal();
    }

    // Конструктор от вектора из цифр:
    Int (const std::vector<int>& digits) : digits(digits) { to_normal(); }

    // Конструктор от строки:
    Int (std::string s) {
        const int size = (int)s.size();
        for (int idGroup = 1, nGroups = size / WIDTH; idGroup <= nGroups; ++idGroup) {
            digits.push_back(std::stoi(s.substr(size-idGroup * WIDTH, WIDTH)));
        }
        if (size % WIDTH != 0) {
            digits.push_back(std::stoi(s.substr(0, size % WIDTH)));
        }
        to_normal();
    }

    // Прибавление:
    Int& operator+=(const int num) {
        if (num >= BASE) {
            return *this += Int(num);
        }
        int rem = num;
        for (int i = 0; rem > 0; ++i) {
            if (i >= size()) digits.push_back(0);
            rem += at(i);
            at(i) = rem % BASE;
            rem /= BASE;
        }
        return this->to_normal();
    }

    // Прибавление:
    Int& operator+=(const Int& other) {
        if (other.size() == 1) {
            return *this += other.at(0);
        }
        const int s1 = this->size();
        const int s2 = other.size();

```

```

    int rem = 0;
    for (int i = 0; i < s1 || i < s2 || rem > 0; ++i) {
        int d1 = i < s1 ? this->at(i) : 0;
        int d2 = i < s2 ? other.at(i) : 0;
        rem += d1 + d2;
        if (i >= s1) digits.push_back(0);
        at(i) = rem % BASE;
        rem /= BASE;
    }
    return this->to_normal();
}

// Вычитание короткого:
Int& operator-=(const int num) {
    if (num >= BASE) {
        return *this -= Int(num);
    }
    int rem = -num;
    for (int i = 0; rem < 0; ++i) {
        assert(i < size());
        rem += at(i);
        if (rem < 0) {
            at(i) = (rem + BASE) % BASE;
            rem = -1;
        } else {
            at(i) = rem % BASE;
            rem = 0;
        }
    }
    return this->to_normal();
}

// Вычитание длинного:
Int& operator-=(const Int& other) {
    if (other.size() == 1) {
        return *this -= other.at(0);
    }
    assert(*this >= other);
    const int s1 = this->size();
    const int s2 = other.size();
    int rem = 0;
    for (int i = 0; i < s1 || i < s2; ++i) {
        int d1 = i < s1 ? this->at(i) : 0;
        int d2 = i < s2 ? other.at(i) : 0;
        rem += d1 - d2;
        if (i >= s1) digits.push_back(0);
        if (rem < 0) {
            at(i) = (rem + BASE) % BASE;
            rem = -1;
        } else {
            at(i) = rem % BASE;
            rem = 0;
        }
    }
    return this->to_normal();
}

// Умножение на короткое:
Int& operator*=(const int num) {
    // std::cout << "call Int& operator*=(const int num)" << std::endl;
    if (num >= BASE) {
        return *this *= Int(num);
    }
    int64_t rem = 0;
    for (int i = 0; i < size() || rem > 0; ++i) {
        // std::cout << "i = " << i << std::endl;
        if (i >= size()) digits.push_back(0);
        rem += 1LL * at(i) * num;
        at(i) = rem % BASE;
    }
}

```

```

        rem /= BASE;
    }
    return this->to_normal();
}

// Умножение:
Int operator*(const int num) const {
    return num >= BASE ? *this * Int(num) : Int(*this) *= num;
}

// Произведение:
Int operator*(const Int& other) const {
    // std::cout << "call Int operator*(const Int& other)" << std::endl;
    if (other.size() == 1) {
        return *this * other.at(0);
    }
    const int s1 = this->size();
    const int s2 = other.size();
    std::vector<int> temp(s1+s2);
    for (int i = 0; i < s1; ++i) {
        int64_t rem = 0;
        for (int j = 0; j < s2; ++j) {
            rem += temp.at(i+j) + 1LL * this->at(i) * other.at(j);
            temp.at(i+j) = rem % BASE;
            rem /= BASE;
        }
        if (rem > 0) {
            temp.at(i+s2) += rem;
            assert(0 <= temp.at(i+s2) && temp.at(i+s2) < BASE);
        }
    }
    return Int(temp);
}

// Умножение:
Int& operator*=(const Int& other) {
    // std::cout << "call Int& operator*=(const Int& other)" << std::endl;
    return other.size() == 1 ? *this *= other.at(0) : *this = *this * other;
}

// Деление на короткое:
Int& operator/=(const int num) {
    // std::cout << "call Int& operator/=(const int num)" << std::endl;
    if (num >= BASE) {
        return *this /= Int(num);
    }
    assert(0 < num && num < BASE);
    int64_t rem = 0;
    for (int j = size()-1; j >= 0; --j) {
        // std::cout << "j = " << j << std::endl;
        (rem *= BASE) += at(j);
        // std::cout << "at(j) = " << at(j) << std::endl;
        at(j) = rem / num;
        assert(0 <= at(j) && at(j) < BASE);
        rem %= num;
    }
    // std::cout << "gool end!" << std::endl;
    return this->to_normal();
}

// Взятие остатка от деления:
Int& operator%=(const int num) {
    return *this = *this % num;
}

static Int div(Int a, Int b) {
    // Данный метод работает за O(m*n)
    // Условия:
    // base^(m-1) <= a < base^m (1)

```

```

// base^n / 2 <= b < base^n (2)
// Условие (1) выполняется всегда для m = a.size()
// Условие (2) необходимо обеспечить равносильным преобразованием следующим образом
if (2 * b.digits.back() < Int::BASE) {
    int d = (1LL * Int::BASE + 2 * b.digits.back() - 1) / (2 * b.digits.back());
    a *= d;
    b *= d;
}

const int n = b.size(), m = a.size();

std::function<Int(Int, Int)> special_div = [&special_div](const Int& a, const Int&
    // Данный метод работает за O(n), при следующих условиях:
    // 0 <= a <= base^(n+1)
    // b^n / 2 <= b < base^n
    const int n = b.size();
    // Проверки на соответствии условиям метода:
    assert(a.size() - b.size() == 1);
    if (a >= b * BASE) {
        return Int(BASE) + special_div(a-b * BASE, b);
    } else {
        int64_t q = (BASE * 1LL * a.at(n) + a.at(n-1)) / b.at(n-1);
        assert(0 <= q && q < BASE);
        auto t = q * b;
        if (t > a) { --q, t -= b; }
        if (t > a) { --q, t -= b; }
        assert(t <= a);
        return Int(q);
    }
};

if (m < n) {
    return 0; // O(1)
} else if (m == n) {
    return a >= b ? 1 : 0; // O(n)
} else if (m == n+1) {
    return special_div(a, b); // O(n)
} else {
    Int a_temp = std::vector<int>(a.digits.begin()+m-n-1, a.digits.begin()+m); // O
    Int s = std::vector<int>(a.digits.begin(), a.digits.begin()+m-n-1); // O(n)
    Int q_temp = special_div(a_temp, b); // O(n)
    Int r_temp = a_temp - q_temp * b; // O(n)
    Int q = div(r_temp.shift_left(m-n-1)+s, b); // O(m-n) рекурсивных вызовов -> об
    return q_temp.shift_left(m-n-1) + q; // O(n)
}

Int operator/(const Int& other) const {
    // std::cout << "call operator/(const Int& other)" << std::endl;
    return (other.size() == 1) ? *this / other.at(0) : div(*this, other);
}

Int& operator/=(const Int& other) {
    // std::cout << "call Int& operator/=(const Int& other)" << std::endl;
    return *this = *this / other;
}

Int operator%(const Int& other) const {
    // std::cout << "call operator%(const Int& other)" << std::endl;
    return *this - *this / other * other;
}

Int& operator%=(const Int& other) {
    return *this = *this % other;
}

// Сравнение: result < 0 (меньше), result == 0 (равно), result > 0 (больше)
int compare(const Int& other) const {
    if (this->size() > other.size()) return 1;

```

```

        if (this->size() < other.size()) return -1;
        for (int i = size()-1; i >= 0; --i) {
            if (this->at(i) > other.at(i)) return 1;
            if (this->at(i) < other.at(i)) return -1;
        }
        return 0;
    }

    int& at(int pos) {
        assert(0 <= pos && pos < size());
        return digits.at(pos);
    }

    const int& at(int pos) const {
        assert(0 <= pos && pos < size());
        return digits.at(pos);
    }

    inline int size() const {
        return (int)digits.size();
    }

    Int& shift_left(int pow) {
        assert(pow >= 0);
        digits.resize((int)digits.size()+pow);
        for (int i = (int)digits.size()-1; i >= pow; --i) {
            digits[i] = digits[i-pow];
        }
        for (int i = pow-1; i >= 0; --i) {
            digits[i] = 0;
        }
        return to_normal();
    }

    Int& shift_right(int pow) {
        assert(pow >= 0);
        if (pow >= (int)digits.size()) {
            return *this = 0;
        }
        for (int i = 0; i + pow < (int)digits.size(); ++i) {
            digits[i] = digits[i+pow];
        }
        digits.resize((int)digits.size()-pow);
        return to_normal();
    }
};

// Ввод из потока:
std::istream& operator>>(std::istream& is, Int& number) {
    std::string s;
    is >> s;
    number = Int(s);
    return is;
}

// Вывод в поток:
std::ostream& operator<<(std::ostream& os, const Int& number) {
    os << number.digits.back();
    for (int i = (int)number.digits.size()-2; i >= 0; --i) {
        os << std::setw(Int::WIDTH) << std::setfill('0') << number.digits[i];
    }
    return os << std::setfill(' ');
}

// Сложение:
Int operator+(const Int& a, const Int& b) {
    return Int(a) += b;
}

```

```

// Вычитание:
Int operator-(const Int& a, const Int& b) {
    return Int(a) -= b;
}

// Умножение:
Int operator*(const int a, const Int& b) {
    return b * a;
}

// Деление:
Int operator/(const Int& a, const int num) {
    // std::cout << "operator/(const Int& a, const int num)" << std::endl;
    return Int(a) /= num;
}

// Остаток от деления:
int operator%(const Int& a, const int num) {
    int64_t rem = 0;
    for (int i = a.size()-1; i >= 0; --i) {
        ((rem *= Int::BASE) += a.at(i)) %= num;
    }
    return rem;
}

// Возведение в степень:
Int pow(Int a, int n) {
    Int res = 1;
    while (n > 0) {
        if (n % 2 != 0) {
            res *= a;
        }
        a *= a;
        n /= 2;
    }
    return res;
}

// Операторы сравнения:
bool operator<(const Int& a, const Int& b) { return a.compare(b) < 0; }
bool operator>(const Int& a, const Int& b) { return a.compare(b) > 0; }
bool operator==(const Int& a, const Int& b) { return a.compare(b) == 0; }
bool operator<=(const Int& a, const Int& b) { return a.compare(b) <= 0; }
bool operator>=(const Int& a, const Int& b) { return a.compare(b) >= 0; }
bool operator!=(const Int& a, const Int& b) { return a.compare(b) != 0; }

Int solve(int n) {
    if (n == 1) return 1;
    if (n == 2) return 2;
    Int prev = 1, curr = 2;
    for (int i = 3; i <= n; ++i) {
        Int next = curr + prev;
        prev = curr;
        curr = next;
    }
    return curr;
}

int main() {
    int n; std::cin >> n;
    Int answ = solve(n);
    std::cout << answ * answ << std::endl;
    return 0;
}

```

ЗАДАЧА №545

Задача Пифагора

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Пифагор заказал ремесленнику изготовить несколько прямоугольных треугольников из ценных пород дерева для использования на занятиях по геометрии, но ремесленник перепутал размеры, и треугольники получились не прямоугольные. Чтобы не выбрасывать испорченный ценный материал, ремесленник решил переделать получившиеся треугольники в прямоугольные, постаравшись максимизировать их площади.

Требуется написать программу, которая по размерам сторон треугольника находит максимальную площадь прямоугольного треугольника, который можно вырезать из этого треугольника.

Входные данные

Входной файл INPUT.TXT содержит в первой строке три числа, разделенных пробелами – длины сторон треугольника. Все числа вещественные, больше 0 и меньше 1000.

Выходные данные

В выходной файл OUTPUT.TXT выведите максимальную площадь прямоугольного треугольника, получаемого из заданного треугольника, с точностью не хуже, чем 10^{-5} .

Пример

№	INPUT.TXT	OUTPUT.TXT
1	10.0 10.0 10.0	25.00000

```
/*
   "Задача Пифагора": геометрия, наибольшая площадь треугольника, O(1)
*/

#include <iostream>
#include <iomanip>
#include <algorithm>
#include <cmath>
#include <cassert>

typedef long double Real;

const Real PI = std::acos(Real(-1));

Real solve(Real a, Real b, Real c) {
    auto minAngle = std::acos((b*b+c*c-a*a)/(2*b*c));
    if (minAngle >= PI / 4) {
        return c * c / 4;
    } else if (c * std::cos(minAngle) >= b) {
        return b * b * std::tan(minAngle) / 2;
    } else {
        return c * std::sin(minAngle) * c * std::cos(minAngle) / 2;
    }
}

int main() {
```



```
Real a, b, c;  
std::cin >> a >> b >> c;  
if (a > b) std::swap(a, b);  
if (a > c) std::swap(a, c);  
if (b > c) std::swap(b, c);  
assert(a <= b && b <= c);  
std::cout << std::fixed << std::setprecision(6) << solve(a, b, c);  
return 0;  
}
```

ЗАДАЧА №559

Сосиска в тесте

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Аня еще учится в школе, однако она уже много слышала о трудностях студенческой жизни. Ее это очень волнует, так как она задумывается о своем будущем и понимает, что после окончания школы ей придется учиться в ВУЗе. Из многочисленных рассказов она узнала, что излюбленной пищей студентов являются сосиски. А так как студенты часто испытывают недостатки в средствах, сосиски часто приходится делить. Однако никто не доверяет делению «на глаз», всегда найдется кто-нибудь, утверждающий, что его обделили.

Сосиска представляет собой практически плоскую фигуру (таковы уж студенческие столовые), которую можно описать следующим образом. Рассмотрим некоторый горизонтальный отрезок на плоскости, а также некоторый круг. Тогда, если центр круга провести по всем точкам отрезка от начальной до конечной, то любая точка, лежавшая, хотя бы в какой-то момент в круге, принадлежит сосиске. Можно описать сосиску и другим способом. Рассмотрим тот же отрезок и два круга одинакового радиуса с центрами в его крайних точках. Тогда точка принадлежит сосиске ровно в том случае, если она лежит на некотором отрезке, у которого каждый из концов лежит в одном из кругов.

Подумав, Аня понимает, что при наличии линейки можно легко отмерить нужное расстояние от левого края сосиски и вертикально отрезать необходимую часть. Но если при этом отмерять одинаковые расстояния, то те, кому достанутся крайние куски, получают меньше. Помогите Ане разместить разрезы в нужных местах сосиски, чтобы все получили поровну.

Входные данные

Входной файл INPUT.TXT содержит в первой строке два числа: d - расстояние между центрами крайних кругов и r - радиус каждого из них. Оба числа вещественные, положительные и не превосходят 10 000 и располагаются на первой строке входного файла. На второй строке находится n – количество студентов, претендующих на сосиску ($2 \leq n \leq 1000$).

Выходные данные

В выходном файле OUTPUT.TXT должно содержаться $n-1$ строка. В i -ой строке должно содержаться расстояние от левого края сосиски до правого края i куска (куски нумеруются слева направо). Все числа должны быть вычислены с точностью до 6 десятичных знаков после запятой.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 1 2	1.500000
2	1 1 4	0.856810 1.500000 2.143190

```
#include <bits/stdc++.h>

typedef long double Real;
```

```

const Real PI = std::acos(-1.0L);
const Real EPS = 1e-14;

// Square of figure from 0 to x:
Real square_till(Real r, Real d, Real x) {
    if (x <= r) {
        Real alpha = std::acos((r-x)/r);
        return r * r * (alpha - std::sin(2 * alpha) / 2);
    }
    if (x >= r + d) {
        return PI * r * r + d * 2 * r - square_till(r, d, 2 * r + d - x);
    }
    return PI * r * r / 2 + (x - r) * 2 * r;
}

// Square of figure from x to y:
Real square_before(Real r, Real d, Real x, Real y) {
    return square_till(r, d, y) - square_till(r, d, x);
}

// Find root of equation  $x = s / r^2 + \sin(2x) / 2$  using binsearch:
Real solve_binsearch(Real s, Real r) {
    Real low = -100, high = 100;
    while (high - low > EPS) {
        Real mid = (low + high) / 2;
        Real value = mid - s / r / r - std::sin(2 * mid) / 2;
        if (value >= 0) {
            high = mid;
        } else {
            low = mid;
        }
    }
    return (low + high) / 2;
}

// Find root of equation  $x = s / r^2 + \sin(2x) / 2$  using newton iterations
Real solve_newton(Real s, Real r) {
    // next = curr - f(curr) / f'(curr)
    // f(x) = x - s / r^2 - sin(2*x) / 2
    // f'(x) = 1 - cos(2*x)
    Real curr = s / r / r + 0.5; // x0
    while (true) {
        Real next = curr - (curr - s / r / r - std::sin(2*curr) / 2) / (1 - std::cos(2*curr));
        if (std::abs(next - curr) < EPS) {
            break;
        }
        curr = next;
    }
    return curr;
}

Real solve_equation(Real s, Real r) {
    return solve_newton(s, r); // or solve_binsearch(s, r);
}

// Get next from x = x1 point x2: square between x1 and x2 = s
Real next(Real d, Real r, Real s, Real x1) {
    if (square_before(r, d, x1, r) + EPS >= s) {
        Real angle = solve_equation(square_till(r, d, x1) + s, r);
        return r - std::cos(angle) * r;
    }
    if (square_before(r, d, x1, r + d) + EPS >= s) {
        if (x1 <= r) {
            return (s - PI * r * r / 2 + square_till(r, d, x1)) / (2 * r) + r;
        } else {
            return s / (2 * r) + x1;
        }
    }
    if (square_before(r, d, x1, 2 * r + d) + EPS >= s) {

```

```

        Real angle = solve_equation(PI * r * r / 2 - (s + square_before(r, d, r+d, x1)), r)
        return r + d + r * std::cos(angle);
    }
    return -1;
}

int main() {
    Real d, r, n, x = 0;
    std::cin >> d >> r >> n;
    auto s = square_till(r, d, 2 * r + d) / n;
    for (int i = 1; i < n; ++i) {
        x = ::next(d, r, s, x);
        assert(x != -1);
        std::cout << std::fixed << std::setprecision(6) << x << "\n";
    }
    return 0;
}

```

ЗАДАЧА №603

Поиск

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Обычно программы, предоставляющие возможность поиска заданных строк в текстовых файлах, недостаточно гибко обрабатывают различные пробельные символы. Например, если в некотором тексте слова «Internet» и «olympiad» разделены переводом строки, словосочетание «Internet olympiad» чаще всего не будет обнаружено в этом месте. В данной задаче пробельными символами мы будем считать пробелы, символы табуляции (код символа 9), а так же переводы строк. Любую последовательность идущих подряд непробельных символов будем называть словом.

Ваша программа должна производить обработку одного запроса на поиск словосочетания в тексте. Словосочетание будет задано как последовательность слов, состоящих из цифр и строчных и прописных букв английского алфавита, каждые два из которых разделены пробелом. Будем считать, что некоторая последовательность символов, первый и последний из которой непробельные, является вхождением этого словосочетания в текст, если после замены каждого блока пробельных символов из этой последовательности на один пробел она совпадет с заданным словосочетанием с точностью до регистра букв. Для представления ответа перед каждым вхождением словосочетания в исходный текст следует поставить символ «@».

Входные данные

Первая строка входного файла INPUT.TXT, заканчивающаяся переводом строки, задает запрос. Длина словосочетания не превосходит 100 символов. Последующие строки описывают сам текст, размер которого не превосходит 2000 символов. Файл заканчивается переводом строки.

Выходные данные

В выходной файл OUTPUT.TXT выведите результат применения к тексту описанной процедуры. Он должен отличаться от исходного текста только добавлением символов «@».

Пример

№	INPUT.TXT	OUTPUT.TXT
1	internet olympiad Internet Olympiads Everyone is welcome to participate in internet olympiads. Jury of internet olympiads	@Internet Olympiads Everyone is welcome to participate in @internet olympiads. Jury of @internet olympiads

```
/*
"Поиск": поиск подстроки в тексте, O(n*m)
*/

#include <stdio.h>
#include <string>
#include <cassert>
#include <vector>
```

```

int main() {
    char buf[2001] = {}, c;
    scanf("%100[^\n]%c", buf, &c);
    assert(c == '\n');

    std::string s(buf);

    int pos = 0;
    while (scanf("%c", &c) != EOF) {
        buf[pos++] = c;
    }
    buf[pos] = '\0';

    std::string t(buf);

    std::vector<int> finded;
    for (pos = (int)t.size() - 1; pos >= 0; --pos) {
        int i = 0, j = pos;
        bool flag = true;
        while (i < (int)s.size() && j < (int)t.size()) {
            if ((int)std::isspace(s[i]) ^ (int)std::isspace(t[j])) {
                flag = false;
                break;
            }
            while (i < (int)s.size() && std::isspace(s[i])) ++i;
            while (j < (int)t.size() && std::isspace(t[j])) ++j;
            if (i == (int)s.size()) break;
            if (j == (int)t.size()) { flag = false; break; }
            if (std::tolower(s[i]) != std::tolower(t[j])) {
                flag = false; break;
            }
            ++i, ++j;
        }
        if (flag) {
            finded.push_back(pos);
        }
    }

    for (auto& it : finded) {
        t.insert(t.begin()+it, '@');
    }

    printf("%s", t.c_str());

    return 0;
}

```

ЗАДАЧА №627

Игра в слова

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Существует множество игр, которые называются «игра в слова». В одной из вариаций правила следующие: выбирается длинное слово, после чего два игрока пытаются вспомнить все слова, которые можно составить, используя некоторые буквы загаданного слова. После этого они по очереди называют придуманные слова (называть одно слово два раза запрещается). Если первый игрок назвал больше слов, то он побеждает, если у обоих игроков слова закончились одновременно, засчитывается ничья, в оставшемся случае побеждает второй.

Оказывается, в этой игре важно не только знать много слов, но и придерживаться правильной стратегии. Напишите программу, которая будет узнавать, кто победит, если оба игрока будут играть идеально, если первый игрок будет играть оптимально, а второй - наихудшим образом, и если первый игрок будет поддаваться. При этом можно считать, что, придумав слова, игроки записывают их на бумагу, и каждый видит записи предыдущего. Если на ходу того игрока, который поддается, у него остаются не упоминавшиеся ранее слова, он обязан назвать одно из них.

Входные данные

В первой строке входного файла INPUT.TXT записано загаданное слово. Затем описываются слова, которые знает первый игрок - на отдельной строке целое число n_1 ($0 \leq n_1 \leq 10000$), за которым следуют n_1 слов по одному на строке. После этого задается число n_2 слов ($0 \leq n_2 \leq 10000$), известных второму игроку, и описываются сами эти слова в таком же формате. Все слова состоят из маленьких букв английского алфавита, а их длины не превосходят 100 символов. В списках слов, известных игрокам, могут содержаться слова, которые нельзя составить из букв загаданного слова.

Выходные данные

В выходной файл OUTPUT.TXT выведите три числа по одному на строке - ответы для случаев, когда оба игрока играют оптимально и когда поддаются первый и второй игроки соответственно: номер выигрывающего игрока и 0 в случае ничьи.

Примеры

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	internationalization	
	6	
	zone	
	oil	
	rent	
	impression	
	noir	
	trail	0
	7	2
	teal	1
	creativity	
	rent	
	rain	
	oil	
	zealot	
	zone	

```

/*
Задача: 627. Игра в слова

Решение: сортировка, множества, пересечение, строки, O(n * log(n) * LEN)

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <stdio.h>
#include <vector>
#include <string>
#include <algorithm>

#define all(x) (x).begin(), (x).end()
#define unique(x) (x).erase(std::unique(all(x)), (x).end())

std::vector<std::string> input(const std::vector<int>& cnt) {
    int q; scanf("%d", &q); char buf[101];
    std::vector<std::string> answ;
    for (int i = 0; i < q; ++i) {
        scanf("%100s", buf);
        std::string s(buf);
        std::vector<int> tmp(26);
        for (auto it : s) {
            tmp[int(it-'a')]+=;
        }
        bool flag = true;
        for (int j = 0; j < 26; ++j) {
            if (tmp[j] > cnt[j]) {
                flag = false;
                break;
            }
        }
        if (!flag) continue;
        answ.push_back(s);
    }
    return answ;
}

int first(int a, int b, int ab) {
    // Первый поддается
    // Сначала называет все слова, которые у него уникальны
    int min = std::min(a, ab);
    a -= min;
    ab -= min;
    // Затем либо уникальные закончились:
    if (ab == 0) {
        if (a < b) {

```



```

        return 2;
    } else if (a > b) {
        return 1;
    } else return 0;
}
// Либо у первого слова закончились:
a += ab % 2;
if (a < b) {
    return 2;
} else if (a > b) {
    return 1;
} else return 0;
}

int second(int a, int b, int ab) {
// Второй поддается
    int min = std::min(b, ab);
    b -= min;
    ab -= min;
    if (ab == 0) {
        if (a < b) {
            return 2;
        } else if (a > b) {
            return 1;
        } else return 0;
    }
    // b == 0, ab != 0;
    a += (ab+1)/ 2;
    b += ab/2;
    if (a < b) {
        return 2;
    } else if (a > b) {
        return 1;
    } else return 0;
}

int equal(int a, int b, int ab) {
    a += ab % 2;
    if (a < b) {
        return 2;
    } else if (a > b) {
        return 1;
    }
    return 0;
}

int main() {
    char buf[100+1];
    scanf("%100s", buf);
    std::vector<int> cnt(26);
    for (auto it : std::string(buf)) {
        cnt[int(it-'a')]+=;
    }
    auto fi = input(cnt);
    auto se = input(cnt);
    std::sort(all(fi)); unique(fi);
    std::sort(all(se)); unique(se);
    std::vector<std::string> same;
    std::set_intersection(all(fi), all(se), std::back_inserter(same));
    int ab = (int)same.size();
    int a = (int)fi.size()-ab;
    int b = (int)se.size()-ab;
    printf("%d\n%d\n%d\n", equal(a,b,ab), first(a,b,ab), second(a,b,ab));
    return 0;
}

```

ЗАДАЧА №683

Числа - 3

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Дана последовательность чисел a_1, a_2, \dots, a_N . За одну операцию разрешается удалить любое (кроме крайних) число, заплатив за это штраф, равный произведению этого числа на сумму соседних. Требуется удалить все числа, кроме крайних, с минимальным суммарным штрафом.

Например:

- Начальная последовательность: **1 50 51 50 1**.
- Удаляем четвёртое число, штраф $50(51+1)=2600$, получаем **1 50 51 1**.
- Удаляем третье число, штраф $51(50+1)=2601$, получаем **1 50 1**.
- Удаляем второе число, штраф $50(1+1)=100$.
- Итого штраф **5301**.

Входные данные

В первой строке входного файла INPUT.TXT записано одно число N ($1 \leq N \leq 100$) - количество чисел в последовательности.

Во второй строке находятся N целых чисел a_1, a_2, \dots, a_N ; никакое из чисел не превосходит по модулю 100.

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести одно число - минимальный суммарный штраф.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 1 50 51 50 1	5301

```
#include <iostream>
#include <functional>
#include <climits>

int main() {
    int n; std::cin >> n;

    static int arr[100];
    for (int i = 0; i < n; ++i) {
        std::cin >> arr[i];
    }

    static int min[100][100];
    for (int i = 0; i < 100; ++i) {
        for (int j = 0; j < 100; ++j) {
            min[i][j] = -1;
        }
    }
}
```

```

std::function<int(int, int)> get = [&](const int left, const int right) {
    if (right <= left+1) {
        return 0;
    }
    if (min[left][right] == -1) {
        int value = INT_MAX;
        for (int p = left+1; p < right; ++p) {
            value = std::min(value, get(left, p) + get(p, right) + (arr[left] + arr[rig
        ]
        min[left][right] = value;
    }
    return min[left][right];
};
std::cout << get(0, n-1);
return 0;
}

```

ЗАДАЧА №708

Хомяки и кролики

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

В поисках пропитания большая дружная семья кроликов добрела до морковного поля. К сожалению, чуть раньше сюда же прибыла большая дружная семья голодных хомяков. Во избежание конфликта было решено собирать урожай по очереди. Поле представляет собой N грядку по M кустов; на каждом кусте растет некоторое количество морковок. Очередной собирающий стартует от любого куста первой грядки и движется к последней, переходя от одного куста к другому по следующему правилу: от куста номер K на грядке L можно перейти только на грядку $L+1$ к одному из трех кустов с номерами $K-1$, K , $K+1$ (конечно, если кусты с такими номерами есть). Каждый посещенный куст очищается от моркови полностью. Первым на сбор урожая выходит один из кроликов, следом идет хомяк, потом снова кролик и так до тех пор, пока на поле есть хоть одна морковка.

Кролики суетливы, поэтому они выбирают путь наиболее выгодный внешне: стартуют от самого богатого куста первой грядки, а из трех последующих вариантов всегда выбирают самый большой куст (при наличии нескольких кустов с одинаковым числом морковок выбирается куст с наибольшим номером). Хомяки, прибыв на поле раньше, успели составить подробную карту поля и поддерживают её в актуальном состоянии на основе оперативных данных о сборе урожая, поэтому они для каждого хомяка выбирают путь, позволяющий собрать максимальное количество морковок из возможных (при наличии нескольких вариантов с максимально возможным количеством морковок выбирается тот, где лексикографически больше последовательность номеров кустов в порядке посещения).

По известной карте поля определите, сколько моркови удалось собрать кроликам и хомякам по отдельности.

Входные данные

В первой строке входного файла INPUT.TXT содержится два целых числа N и M ($1 \leq N, M \leq 100$). Следом идут N строк, в каждой из которых M чисел X_{ij} ($0 \leq X_{ij} \leq 10$). X_{ij} - количество морковок на j -ом кусте i -ой грядки.

Выходные данные

В выходной файл OUTPUT.TXT выведите через пробел два числа: количество морковок, собранных кроликами и хомяками, соответственно.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 3 1 1 2 1 1 1 10 1 1	7 12
2	4 4 1 1 1 2 1 1 1 1 1 1 1 1 10 10 1 1	18 17

```

/*
    "Хомяки и кролики": динамическое программирование,  $O(N^3)$ 
*/

#include <stdio.h>
#include <algorithm>
#include <vector>

int main() {
    int nRows, nCols;
    scanf("%d %d", &nRows, &nCols);

    std::vector<std::vector<int>> arr(1+nRows, std::vector<int>(nCols, 0));
    int sum = 0;
    for (auto& row : arr) {
        for (auto& it : row) {
            scanf("%d", &it);
            sum += it;
        }
    }

    int s1 = 0, s2 = 0;
    do {
        // Жадное моделирование действий кролика:
        int maxj = nCols-1;
        for (int j = nCols-1; j >= 0; --j) {
            if (arr[0][j] > arr[0][maxj]) {
                maxj = j;
            }
        }
        s2 += arr[0][maxj];
        arr[0][maxj] = 0;

        for (int i = 1; i < nRows; ++i) {
            int j = maxj;
            if (j + 1 < nCols && arr[i][j+1] >= arr[i][maxj]) {
                maxj = j+1;
            }
            if (arr[i][j] > arr[i][maxj]) {
                maxj = j;
            }
            if (j - 1 >= 0 && arr[i][j-1] > arr[i][maxj]) {
                maxj = j-1;
            }
            s2 += arr[i][maxj];
            arr[i][maxj] = 0;
        }

        // Инициализация массивов для двумерного ДП:
        std::vector<std::vector<int>> smax(1+nRows, std::vector<int>(nCols, -1));
        for (int j = 0; j < nCols; ++j) {
            smax.back()[j] = 0;
        }
        std::vector<std::vector<int>> from(1+nRows, std::vector<int>(nCols, -1));
        // ДП снизу-вверх:
        for (int i = nRows-1; i >= 0; --i) {
            for (int j = nCols-1; j >= 0; --j) {
                if (j+1 < nCols && smax[i][j] < smax[i+1][j+1] + arr[i][j]) {
                    smax[i][j] = smax[i+1][j+1] + arr[i][j];
                    from[i][j] = j+1;
                }

                if (smax[i][j] < smax[i+1][j] + arr[i][j]) {
                    smax[i][j] = smax[i+1][j] + arr[i][j];
                    from[i][j] = j;
                }

                if (j-1 >= 0 && smax[i][j] < smax[i+1][j-1] + arr[i][j]) {
                    smax[i][j] = smax[i+1][j-1] + arr[i][j];
                }
            }
        }
    } while (s2 > 0);
}

```

```

        from[i][j] = j-1;
    }
}

// Моделирование действий хомяков:
maxj = nCols-1;
for (int j = nCols-1; j >= 0; --j) {
    if (smax[0][maxj] < smax[0][j]) {
        maxj = j;
    }
}

s1 += smax[0][maxj];

for (int i = 0; i < nRows; ++i) {
    arr[i][maxj] = 0;
    maxj = from[i][maxj];
}

} while (s1+s2 < sum);
printf("%d %d", s2, s1);
return 0;
}

```

ЗАДАЧА №717

Производство деталей

(Время: 2 сек. Память: 64 Мб Сложность: 48%)

Предприятие «Авто-2010» выпускает двигатели для известных во всем мире автомобилей. Двигатель состоит ровно из n деталей, пронумерованных от 1 до n , при этом деталь с номером i изготавливается за p_i секунд. Специфика предприятия «Авто-2010» заключается в том, что там одновременно может изготавливаться лишь одна деталь двигателя. Для производства некоторых деталей необходимо иметь предварительно изготовленный набор других деталей.

Генеральный директор «Авто-2010» поставил перед предприятием амбициозную задачу – за наименьшее время изготовить деталь с номером 1, чтобы представить ее на выставке.

Требуется написать программу, которая по заданным зависимостям порядка производства между деталями найдет наименьшее время, за которое можно произвести деталь с номером 1.

Входные данные

Первая строка входного файла INPUT.TXT содержит число n ($1 \leq n \leq 100000$) – количество деталей двигателя. Вторая строка содержит n натуральных чисел p_1, p_2, \dots, p_n , определяющих время изготовления каждой детали в секундах. Время для изготовления каждой детали не превосходит 10^9 секунд.

Каждая из последующих n строк входного файла описывает характеристики производства деталей. Здесь i -ая строка содержит число деталей k_i , которые требуются для производства детали с номером i , а также их номера. Сумма всех чисел k_i не превосходит 200000.

Известно, что не существует циклических зависимостей в производстве деталей.

Выходные данные

В первой строке выходного файла OUTPUT.TXT должны содержаться два числа: минимальное время (в секундах), необходимое для скорейшего производства детали с номером 1 и число k деталей, которые необходимо для этого произвести. Во второй строке требуется вывести через пробел k чисел – номера деталей в том порядке, в котором следует их производить для скорейшего производства детали с номером 1.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 100 200 300 1 2 0 2 2 1	300 2 2 1
2	2 2 3 1 2 0	5 2 2 1

3	4	9 3 3 2 1
	2 3 4 5	
	2 3 2	
	1 3	
	0	
	2 1 3	

```

/*
    "Производство деталей": графы, поиск в глубину, O(n)
*/

#include <stdio.h>
#include <vector>
#include <functional>
#include <algorithm>
#include <string>

int main() {
    int n; scanf("%d", &n);

    std::vector<int> time(n);
    for (auto& it : time) {
        scanf("%d", &it);
    }

    std::vector<std::vector<int>> next(n);
    for (int i = 0; i < n; ++i) {
        int k; scanf("%d", &k);
        while (k--) {
            int vert; scanf("%d", &vert); --vert;
            next[i].push_back(vert);
        }
    }

    std::vector<bool> visited(n, false);

    std::vector<int> answ;

    std::function<void(int)> visit = [&](const int curr) {
        for (auto vert : next[curr]) {
            if (!visited[vert]) {
                visited[vert] = true;
                visit(vert);
            }
        }
        answ.push_back(curr);
    };

    visited[0] = true;
    visit(0);
    long long sum = 0;
    for (auto& it : answ) {
        sum += time[it];
    }
    printf("%s %d\n", std::to_string(sum).c_str(), (int)answ.size());
    for (auto& it : answ) {
        printf("%d ", it+1);
    }
    return 0;
}

```


ЗАДАЧА №770

Покорение вселенной

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Два сообщества А и В начали делить пространство вселенной, устанавливая свой флаг на звездах и планетах. Сообщество А составило маршрут покорения вселенной, состоящий из названий объектов (планет и звезд) в порядке их посещения. Названия объектов состояли из английских букв и начинались с заглавной буквы, все остальные буквы – строчные. Маршрут представляет собой непрерывную цепочку символов, т.к. названия объектов записывались подряд без пробелов. Строка маршрута была зашифрована перестановками групп символов, сформированных из исходной цепочки. Первая группа включает первую букву цепочки. Вторая группа формируется из двух последних букв исходной цепочки в порядке их встречаемости, если они не попали в первую группу. Третья группа – из трех символов от начала цепочки, не вошедших в ни какую группу. Четвертая – из четырех символов от конца цепочки, также пока не включенных в другие группы и т.д. (рис.1). Формирование групп заканчивается, когда каждый символ цепочки включен в какую-либо группу. Если в момент завершения формирования групп последняя группа не набирает требуемое количество символов, то в нее включаются оставшиеся символы, не вошедшие ни в какую группу. Каждый символ принадлежит только одной группе.

При шифровании первая группа символов меняется местами со второй группой, третья группа с четвертой и т.д. Если сформировалось нечетное количество групп, то последняя группа ставится на свободное место, образовавшееся внутри шифруемой строки (рис.2). Приведенный метод шифрования гарантирует, что длина зашифрованной строки всегда равна длине исходной строки.

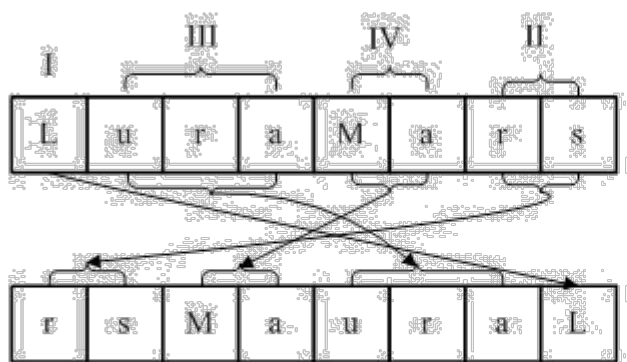


Рис. 1

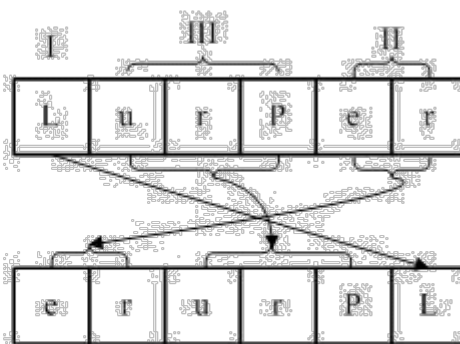


Рис. 2

Сообществу В удалось перехватить маршрут сообщества А. Но его слабые технические возможности не позволяют существенно нарушить планы соперников, поэтому сообществом В был составлен маршрут, включающий только те объекты, которые в исходном маршруте сообщества А находятся в позициях, соответствующих простым числам. Формально, если маршрут сообщества А состоит из объектов $p_1 p_2 p_3 \dots p_n$, то маршрут сообщества В будет иметь вид $p_{i_1} p_{i_2} p_{i_3} \dots p_{i_m}$, где i_k – простое число, $1 < i_k \leq n$, $1 \leq k \leq m$, $i_k < i_{k+1}$.

Входные данные

Входной файл INPUT.TXT содержит строку, представляющую зашифрованный маршрут сообщества А, содержащий от 1 до 500 000 символов.

Выходные данные

В выходной файл OUTPUT.TXT выведите строку, содержащую маршрут посещения планет и звезд сообществом В. Если в результате формирования получился пустой маршрут, не содержащий ни одного объекта, то в выходной файл следует вывести слово «Impossible» (без кавычек).

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	rsMauraL	Mars
2	erurPL	Per
3	iaiopeassMarsCunaL	MarsCassiopeia

```
/*
    "Покорение вселенной": обработка строк, решето Эратосфена, O(n)
*/

#include <stdio.h>
#include <string>
#include <vector>
#include <cassert>
#include <cmath>

bool is_prime(int n) {
    static bool evaluated = false;
    static const int MAX_N = 500000;
    static std::vector<bool> prime(1+MAX_N, true);
    if (!evaluated) {
        prime[1] = prime[0] = false;
        for (int i = 2; i * i <= MAX_N; ++i) {
            if (!prime[i]) {
                continue;
            }
            for (int j = i * i; j <= MAX_N; j += i) {
                prime[j] = false;
            }
        }
        evaluated = true;
    }
    return prime[n];
}

int main() {
    char buf[500000+1];
    scanf("%500000s", buf);
    std::string s(buf);
    const int n = (int)s.size();

    // Дешифрование:
    std::string f(n, '?');
    int rs = n-1, ls = 0, len = 1, lf = 0, rf = n-1;
    while (ls <= rs) {
        if (len % 2 == 1) {
            for (int i = std::max(rs-len+1, ls); i <= rs; ++i) {
                f[lf++] = s[i];
            }
            rs -= len;
        } else {
            for (int i = std::min(rs, ls+len-1); i >= ls; --i) {
                f[rf--] = s[i];
            }
            ls += len;
        }
        ++len;
    }
}
```

```

// Формирование ответа:
std::string answ;
int p = 0, id = 0;
while (p < n) {
    assert('A' <= f[p] && f[p] <= 'Z');
    ++id;
    int r = p+1;
    while (r < n && !('A' <= f[r] && f[r] <= 'Z')) {
        ++r;
    }
    if (is_prime(id)) {
        answ += f.substr(p, r-p);
    }
    p = r;
}
if (answ.empty()) {
    answ = "Impossible";
}
printf("%s\n", answ.c_str());
return 0;
}

```

ЗАДАЧА №788

Интересная игра с числами

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Рассмотрим следующую интересную игру для двух игроков. Для этой игры необходима таблица из 2-х строк и N столбцов, в клетках которой записаны натуральные числа, следующего вида:

Игроки делают ходы по очереди. Начинает игру 1-й игрок.

За один ход 1-й игрок выполняет следующие два действия:

- выбирает произвольный столбец (к примеру, j-й), который еще ни разу не был выбран одним из игроков на предыдущих ходах;
- прибавляет к своим очкам число A_j .

За один ход 2-й игрок выполняет следующие два действия:

- выбирает произвольный столбец (к примеру, j-й), который еще ни разу не был выбран одним из игроков на предыдущих ходах;
- прибавляет к своим очкам число B_j .

Игра заканчивается, когда какой-либо из игроков не сможет сделать ход (по той причине, что все столбцы уже были выбраны). Изначально, у каждого из игроков есть 0 очков.

После того, как игра закончилась, происходит взаиморасчет между игроками. К примеру, 1-й игрок набрал S_1 очков, а 2-й игрок - S_2 очков. В случае, когда $S_1 > S_2$, 2-й игрок отдает 1-му игроку $S_1 - S_2$ УДЕ (условных денежных единиц). В противном случае, 1-й игрок отдает 2-му игроку $S_2 - S_1$ УДЕ. С этих позиций, целью 1-го игрока является максимизация величины $S_1 - S_2$, а целью 2-го игрока - максимизация $S_2 - S_1$.

Назовем стоимостью игры величину $S_1 - S_2$ при оптимальной игре обоих игроков. Напишите программу, которая определяет стоимость игры.

Входные данные

В первой строке входного файла INPUT.TXT записано натуральное число N - количество столбцов в таблице ($1 \leq N \leq 300000$). Следующие N строк описывают числа в столбцах таблицы. i-я из этих строк содержит два натуральных числа A_i и B_i , разделенные одним пробелом ($1 \leq A_i, B_i \leq 3000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число - стоимость игры.

Примеры

A_1	A_2	A_3	...	A_N
B_1	B_2	B_3	...	B_N
№	INPUT.TXT	OUTPUT.TXT		

1	1 1 1	1
2	2 1 1 1 1	0
3	3 1 2 3 4 5 6	2

```

/*
    "Интересная игра с числами": сортировка, жадный алгоритм,  $O(n \log(n))$ 
*/

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <iostream>

struct Pair {
    int a, b;
};

int main() {
    int n;
    scanf("%d", &n);

    std::vector<Pair> pairs(n);
    for (auto& it : pairs) {
        scanf("%d %d", &it.a, &it.b);
    }

    std::stable_sort(pairs.begin(), pairs.end(), [](const Pair& lhs, const Pair& rhs) {
        return lhs.a + lhs.b > rhs.a + rhs.b;
    });

    long long sum = 0;
    for (int i = 0; i < n; ++i) {
        sum += (i % 2 == 0) ? (pairs[i].a) : (-pairs[i].b);
    }
    std::cout << sum;
    return 0;
}

```

ЗАДАЧА №832

Игра в фишки

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Вот уже 10 лет дядя Семен работает сторожем на складе, где хранятся старые процессоры. Его работа чрезвычайно скучна, поэтому все рабочее время он играет в увлекательную игру.

Семен берет A фишек красного цвета, B фишек синего цвета и C зеленого цвета. За один ход он может заменить две фишки разных цветов на одну фишку третьего цвета. Считается, что Семен «сыграл» игру, если после некоторого количества ходов осталась одна фишка.

За 10 лет Семен так научился играть в эту игру, что для произвольных неотрицательных A, B, C он сразу может сказать: можно ли «сыграть» игру или нет.

Ваша задача – научиться это делать за более короткий срок.

Входные данные

В первой строке входного файла INPUT.TXT записано натуральное число N – число тестов ($1 \leq N \leq 1000$). В каждой из последующих N строк содержится тест: три целых числа: A, B и C ($0 \leq A, B, C \leq 2^{63} - 1$).

Выходные данные

В выходной файл OUTPUT.TXT для каждого теста выведите «Yes», если «сыграть» игру можно, иначе выведите «No». Ответ для каждого теста должен располагаться в отдельной строке.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	2 1 0 0 1 1 1	Yes No

```
/*
"Игра в фишки": исследование, O(n)
*/

#include <iostream>
#include <functional>
#include <cstdlib>
#include <cassert>

typedef long long ll;

bool fast(ll a, ll b, ll c) {
    if (a > b) {
        return fast(b, a, c);
    }
    if (a > c) {
        return fast(c, b, a);
    }
    if (b > c) {
```

```

        return fast(a, c, b);
    }
    assert(0 <= a && a <= b && b <= c);
    if (a == 0 && b == 0) {
        return c == 1;
    }
    if (a == 0) {
        return b % 2 || c % 2;
    }
    if (a % 2 != b % 2) {
        return true;
    }
    return a % 2 != c % 2 || b % 2 != c % 2;
}

void research() {
    const int NMAX = 60;
    static char answ[1+NMAX][1+NMAX][1+NMAX];
    for (int i = 0; i <= NMAX; ++i) {
        for (int j = 0; j <= NMAX; ++j) {
            for (int k = 0; k <= NMAX; ++k) {
                answ[i][j][k] = -1;
            }
        }
    }
    answ[0][0][0] = 0;
    answ[1][0][0] = 1;
    answ[0][1][0] = 1;
    answ[0][0][1] = 1;
    for (int a = 0; a <= NMAX / 3; ++a) {
        for (int b = 0; b <= NMAX / 3; ++b) {
            for (int c = 0; c <= NMAX / 3; ++c) {
                std::function<int(int,int,int)> get = [&](const int a, const int b, const int c) {
                    if (a < 0 || b < 0 || c < 0) {
                        return 0;
                    }
                    if (answ[a][b][c] == -1) {
                        answ[a][b][c] = get(a-1, b+1, c-1) || get(a-1, b-1, c+1) || get(a+1, b-1, c-1) || get(a+1, b+1, c+1);
                    }
                    return (int)answ[a][b][c];
                };
                answ[a][b][c] = get(a,b,c);

                if (answ[a][b][c] != fast(a,b,c)) {
                    printf("a=%d, b=%d, c=%d: %s\n", a,b,c, answ[a][b][c] ? "YES":"NO");
                }
            }
        }
    }
    std::exit(0);
}

int main() {
    //research();
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0); std::cout.tie(0); std::cerr.tie(0);
    int nQ;
    std::cin >> nQ;
    while (nQ--) {
        ll a, b, c;
        std::cin >> a >> b >> c;
        std::cout << (fast(a,b,c) ? "Yes\n" : "No\n");
    }
    return 0;
}

```

ЗАДАЧА №968

Строки - 4

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Определим расстояние между равными по длине строками S_A и S_B (обозначим $d(S_A, S_B)$) как сумму для всех $1 \leq i \leq |S_A|$ кратчайших расстояний между буквами $S_A(i)$ и $S_B(i)$ в циклически замкнутом английском алфавите (т.е. после буквы «a» идет буква «b», ..., после буквы «z» идет «a»). Например $d(aba, aca) = 1$, а $d(aba, zbz) = 2$.

Напомним, что циклическим сдвигом строки S называется строка (обозначим, как $S \rightarrow k$) $S_{k+1} S_{k+2} S_{k+3} \dots S_{|S|} S_1 S_2 \dots S_k$ для некоторого k , где $|S|$ – длина строки S .

Степенью циклического расстояния между строками S_A и S_B ($|S_A| = |S_B|$) называется сумма:

$$\sum_{i=1}^{|S_A|} \sum_{j=1}^{|S_B|} d(S_A \rightarrow i, S_B \rightarrow j)$$

Требуется посчитать степень циклического расстояния заданных строк S_A и S_B .

Входные данные

Входной файл INPUT.TXT содержит две строки равной длины, не превышающей 10^5 символов. Строки состоят только из маленьких букв английского алфавита.

Выходные данные

В выходной файл OUTPUT.TXT выведите ответ на задачу.

Примеры

Пояснение

Во втором примере все циклические сдвиги строки «ab»: «ab» и «ba», все циклические сдвиги строки «ac»: «ac» и «ca». Искомое значение равно:

$$d(ab, ac) + d(ab, ca) + d(ba, ac) + d(ba, ca) = 1 + 3 + 3 + 1 = 8$$

№	INPUT.TXT	OUTPUT.TXT
1	a b	1
2	ab ac	8

/*
"Строки - 4": комбинаторика, математика, $O(n)$
*/


```

#include <stdio.h>
#include <algorithm>
#include <string>
#include <vector>
#include <cassert>
#include <iostream>

typedef long long ll;

inline int dist(char a, char b) {
    return a > b ? dist(b, a) : std::min(b-a, std::abs(a+26-b));
}

ll solve(std::string a, std::string b) {
    std::vector<int> ca(26), cb(26);
    for (auto& it : a) ca[it-'a']++;
    for (auto& it : b) cb[it-'a']++;
    ll answ = 0;
    for (int i = 0; i < 26; ++i) {
        for (int j = 0; j < 26; ++j) {
            answ += ll(a.size()) * ca[i] * cb[j] * dist('a' + i, 'a' + j);
        }
    }
    return answ;
}

int main() {
    char buf[100000+1];
    scanf("%100000s", buf);
    std::string a(buf);
    scanf("%100000s", buf);
    std::string b(buf);
    assert(a.size() == b.size());
    std::cout << solve(a, b);
    return 0;
}

```

ЗАДАЧА №994

Длиннейшая общая подпара

(Время: 2 сек. Память: 64 Мб Сложность: 48%)

Будем называть пару строк (α, β) подпарой строки γ , если $\gamma = \gamma_1 \alpha \gamma_2 \beta \gamma_3$ для некоторых (возможно пустых) строк γ_1, γ_2 и γ_3 . Длинной пары строк будем называть сумму длин составляющих ее строк: $|\alpha, \beta| = |\alpha| + |\beta|$.

По заданным двум строкам ξ и η найдите их длиннейшую общую подпару, то есть такую пару строк (α, β) , что она является подпарой как ξ , так и η , и ее длина максимальна.

Входные данные

Входной файл INPUT.TXT содержит две непустые строки ξ и η , состоящие из маленьких букв английского алфавита. Длина каждой из строк не превышает 2000.

Выходные данные

В выходной файл OUTPUT.TXT выведите α на первой строке выходного файла и β на второй строке. Если существует несколько решений, выведите любое из них.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	abacabadabacaba acabacadacabaca	acaba abaca
2	ab bc	b

```
/*
    Решение двумерным ДП:

    1. Предподсчитаем pref[i][j] = len - длина совпадающего суффикса на префиксе a[0..i]
    2. Предподсчитаем suff[i][j] = len - длина совпадающего префикса на суффиксе a[i..n]
    3. Предподсчитаем best_pref[i][j] = {pos, len} - наибольшая общая подстрока на преф
    4. Предподсчитаем best_suff[i][j] = {pos, len} - наибольшая общая подстрока на суфф
    5. Переберем позиции i и j и выберем максимальную пару общих строк на префиксе + су

    Асимптотика O(n^2) по времени и O(n^2) по памяти.

    Возможно, шаги 1 и 3, а также шаги 2 и 4 можно было объединить, но так нагляднее.
*/

#include <stdio.h>
#include <algorithm>
#include <vector>
#include <string>
#include <cassert>
#include <cstdint>
```

```

typedef int16_t Int;

struct Pair {
    Int pos, len;
};

inline bool operator<(const Pair& a, const Pair& b) {
    return a.len < b.len || (a.len == b.len && a.pos > b.pos);
}

inline bool operator>(const Pair& a, const Pair& b) {
    return b < a;
}

inline bool operator==(const Pair& a, const Pair& b) {
    return a.len == b.len && a.pos == b.pos;
}

typedef std::vector<Pair> Vector;
typedef std::vector<Vector> Matrix;

void solve(const std::string& a, const std::string& b, std::string& r1, std::string& r2) {
    const Int n1 = a.size(), n2 = b.size();
    r1.clear(); r2.clear();
    // Считаём наибольшую общую подстроку на префиксе, совпадающую с суффиксом:
    std::vector<std::vector<Int>> pref(n1, std::vector<Int>(n2));
    for (Int i = 0; i < n1; ++i) {
        for (Int j = 0; j < n2; ++j) {
            if (a[i] == b[j]) {
                pref[i][j] = (i > 0 && j > 0 ? pref[i-1][j-1] : 0) + 1;
            } else {
                pref[i][j] = 0;
            }
        }
    }
    // Считаём наибольшую общую подстроку на суффиксе, совпадающую с префиксом:
    std::vector<std::vector<Int>> suff(n1, std::vector<Int>(n2));
    for (Int i = n1-1; i >= 0; --i) {
        for (Int j = n2-1; j >= 0; --j) {
            if (a[i] == b[j]) {
                suff[i][j] = (i+1 < n1 && j+1 < n2 ? suff[i+1][j+1] : 0) + 1;
            } else {
                suff[i][j] = 0;
            }
        }
    }
    // Считаём наибольшую общую подстроку на префиксе:
    Matrix best_pref(n1, Vector(n2, Pair{-1, 0}));
    for (Int i = 0; i < n1; ++i) {
        for (Int j = 0; j < n2; ++j) {
            if (i == 0 && j == 0) {
                best_pref[i][j] = Pair{Int(i-pref[i][j]+1), pref[i][j]};
            } else if (i == 0) {
                best_pref[i][j] = std::max(
                    best_pref[i][j-1], Pair{Int(i-pref[i][j]+1), pref[i][j]}
                );
            } else if (j == 0) {
                best_pref[i][j] = std::max(
                    best_pref[i-1][j], Pair{Int(i-pref[i][j]+1), pref[i][j]}
                );
            } else {
                best_pref[i][j] = std::max({
                    best_pref[i-1][j], best_pref[i][j-1], Pair{Int(i-pref[i][j]+1), pref[i]
                });
            }
        }
    }
    // Считаём наибольшую общую подстроку на суффиксе:

```

```

Matrix best_suff(n1, Vector(n2, Pair{-1, 0}));
for (Int i = n1-1; i >= 0; --i) {
    for (Int j = n2-1; j >= 0; --j) {
        if (i == n1-1 && j == n2-1) {
            best_suff[i][j] = Pair{i, suff[i][j]};
        } else if (i == n1-1) {
            best_suff[i][j] = std::max(
                best_suff[i][j+1], Pair{i, suff[i][j]}
            );
        } else if (j == n2-1) {
            best_suff[i][j] = std::max(
                best_suff[i+1][j], Pair{i, suff[i][j]}
            );
        } else {
            best_suff[i][j] = std::max({
                best_suff[i+1][j], best_suff[i][j+1], Pair{i, suff[i][j]}
            });
        }
    }
}

// Находим ответ:
Pair p1{-1, 0}, p2{-1, 0}; Int best_len = 0;
for (Int i = 0; i < n1; ++i) {
    for (Int j = 0; j < n2; ++j) {
        // Префикс [0, i) и Суффикс [i, n)
        if (i > 0 && j > 0) {
            Int sum_len = best_pref[i-1][j-1].len + best_suff[i][j].len;
            if (sum_len > best_len) {
                best_len = sum_len;
                p1 = best_pref[i-1][j-1];
                p2 = best_suff[i][j];
            }
        } else if (i == 0 && j == 0) {
            Int sum_len = best_suff[i][j].len;
            if (sum_len > best_len) {
                best_len = sum_len;
                p2 = best_suff[i][j];
                p1 = Pair{-1, 0};
            }
        }
    }
}

if (p1.pos != -1) {
    r1 = a.substr(p1.pos, p1.len);
}
if (p2.pos != -1) {
    r2 = a.substr(p2.pos, p2.len);
}

}

int main() {
    char buf[2000+1];
    scanf("%2000s", buf);
    std::string a(buf);
    scanf("%2000s", buf);
    std::string b(buf), r1, r2;
    solve(a, b, r1, r2);
    if (r1.empty()) {
        std::swap(r1, r2);
    }
    if (r1.empty()) {
        return 0;
    }
    if (!r1.empty()) {
        printf("%s\n", r1.c_str());
    }
    if (!r2.empty()) {
        printf("%s\n", r2.c_str());
    }
}

```

```
    return 0;  
}
```

ЗАДАЧА №1320

Строки Фибоначчи

(Время: 1 сек. Память: 16 Мб Сложность: 48%)

Строку Фибоначчи $F(K)$ для натуральных чисел K определим так:

$$F(1) = 'A', F(2) = 'B',$$

$$F(K) = F(K-1) + F(K-2) \text{ при } K > 2, \text{ где "+" означает конкатенацию строк.}$$

Требуется найти количество вхождений строки S , состоящей из символов A и B , в строку Фибоначчи $F(N)$.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число N ($1 \leq N \leq 45$), во второй строке записана не пустая строка S , состоящая не более, чем из 25 символов A и B .

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число - количество вхождений строки S в строку Фибоначчи $F(N)$.

Примеры

Примечание

Длина строки $F(45)$ равна 1 134 903 170.

№	INPUT.TXT	OUTPUT.TXT
1	1 A	1
2	2 ABA	0
3	8 BBABAB	3
4	35 BBABAB	1346268

```
#include <iostream>
#include <vector>
#include <string>

int numberOf(std::string s, std::string sub) {
    int count = 0;
    int pos = 0;
    while (pos != -1) {
        pos = (int)s.find(sub, pos);
        if (pos != -1) {
            pos++;
            count++;
        }
    }
}
```

```

    }
}
return count;
}

int main() {
    std::vector<std::string> full(1+45);
    full[1] = "A";
    full[2] = "B";
    for (int i = 3; i <= 10; ++i) {
        full[i] = full[i-1] + full[i-2];
    }
    int number;
    std::string s;
    std::cin >> number >> s;

    if (number <= 10) {
        std::cout << numberOf(full[number], s) << std::endl;
        return 0;
    }

    std::vector<int> count(1+45);
    for (int i = 1; i <= 10; ++i) {
        count[i] = numberOf(full[i], s);
    }

    std::vector<std::string> begin(1+45), end(1+45);
    begin[10] = full[10];
    end[10] = full[10];
    begin[9] = full[9];
    end[9] = full[9];
    for (int curr = 11; curr <= number; ++curr) {
        begin[curr] = begin[curr-1].substr(0, 25);
        end[curr] = end[curr-2].substr((int)end[curr-2].size()-25, 25);
        count[curr] = count[curr-1] + count[curr-2];
        std::string mid = end[curr-1] + begin[curr-2];
        int len1 = end[curr-1].size();
        int len2 = begin[curr-2].size();
        for (int pos = 0; pos < len1 && pos+(int)s.size()-1 < len1+len2; ++pos) {
            if (pos+(int)s.size()-1 >= len1 && mid.substr(pos, (int)s.size()) == s) {
                count[curr]++;
            }
        }
    }
    std::cout << count[number] << std::endl;
    return 0;
}

```

ЗАДАЧА №425

Прямая и квадраты

(Время: 1 сек. Память: 16 Мб Сложность: 49%)

В прямоугольной декартовой системе координат прямая задана двумя принадлежащими ей точками (0, W) и (100•N, E). Также заданы N² квадратов со сторонами, параллельными осям координат. Квадрат S_{i,j} имеет координаты углов (100•i, 100•j) и (100•i - 100, 100•j - 100), i, j = 1, 2, ..., N.

Требуется найти количество квадратов, имеющих общую точку с прямой.

Входные данные

Входной файл INPUT.TXT содержит в одной строке числа N, W и E, разделенные пробелами. (1 ≤ N ≤ 100, 0 ≤ W, E ≤ 100•N)

Выходные данные

В выходной файл OUTPUT.TXT выведите количество квадратов, имеющих общую точку с заданной прямой.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 150 50	4
2	2 110 120	2

```
#pragma GCC diagnostic ignored "-Wunused-result"

#include <bits/stdc++.h>

struct Segment {
    int x1, y1, x2, y2;

    Segment(int x1 = 0, int y1 = 0, int x2 = 0, int y2 = 0)
        : x1(x1), y1(y1), x2(x2), y2(y2) { }

    int x_min() const { return std::min(x1, x2); }
    int y_min() const { return std::min(y1, y2); }
    int x_max() const { return std::max(x1, x2); }
    int y_max() const { return std::max(y1, y2); }

    bool intersect(const Segment& other) const {
        if (x_max() < other.x_min() || x_min() > other.x_max() || y_max() < other.y_min() ||
            y_min() > other.y_max()) return false;
        auto v1 = 1LL * (other.x1 - x1) * (y2 - y1) - 1LL * (other.y1 - y1) * (x2 - x1);
        auto v2 = 1LL * (other.x2 - x1) * (y2 - y1) - 1LL * (other.y2 - y1) * (x2 - x1);
        return v1 == 0 || v2 == 0 || (v1 < 0 && v2 > 0) || (v1 > 0 && v2 < 0);
    }
};

struct Point {
    int row, col;
};

bool square_intersect(int row, int col, const Segment& line) {
```



```

    Segment diag1(100 * row, 100 * col, 100 * (row+1), 100 * (col+1));
    Segment diag2(100 * (row+1), 100 * col, 100 * row, 100 * (col+1));
    return line.intersect(diag1) || line.intersect(diag2);
}

int solveSlow(int n, const Segment& line) {
    int answ = 0;
    for (int row = 0; row < n; ++row)
        for (int col = 0; col < n; ++col)
            answ += square_intersect(row, col, line);
    return answ;
}

int solveFast(int n, const Segment& line) {
    std::vector<std::vector<bool>> visited(n, std::vector<bool>(n, false));
    Point p1{0, std::min(n-1, line.y1 / 100)};
    assert(square_intersect(p1.row, p1.col, line));
    int answ = 0;
    std::queue<Point> queue;
    queue.push(p1);
    while (!queue.empty()) {
        auto curr = queue.front(); queue.pop();
        if (!visited[curr.row][curr.col]) {
            visited[curr.row][curr.col] = true;
            answ++;
        }
        for (int dr = -1; dr <= 1; ++dr) {
            for (int dc = -1; dc <= 1; ++dc) {
                if (dr * dr + dc * dc == 1) {
                    int nr = curr.row + dr, nc = curr.col + dc;
                    if (nr < 0 || nr >= n || nc < 0 || nc >= n || visited[nr][nc]) continue;
                    if (square_intersect(nr, nc, line)) {
                        visited[nr][nc] = true;
                        ++answ;
                        queue.push({nr, nc});
                    }
                }
            }
        }
    }
    return answ;
}

int main() {
    int n, y1, y2;
    scanf("%d %d %d", &n, &y1, &y2);
    Segment line(0, y1, n*100, y2);
    printf("%d\n", solveFast(n, line));
    return 0;
}

```

ЗАДАЧА №730

Пересылка файлов

(Время: 1 сек. Память: 16 Мб Сложность: 49%)

В постиндустриальную эпоху основной ценностью является информация. Поэтому особо важен контроль над каналами передачи информации. В одной стране все каналы связи контролируются государством.

Перед ИТ-отделом одной достаточно крупной фирмы, занимающейся консалтингом в области инновационных технологий, была поставлена задача распространить некий файл по филиалам этой фирмы, находящимся в различных городах страны.

Каналы передачи информации в этой стране, как уже говорилось, контролируются государством, поэтому за передачу по ним информации приходится платить деньги. Ситуация также осложняется тем, что каналы однонаправленные, то есть информацию по ним можно передавать только в одном направлении.

Пусть, для удобства, города пронумерованы натуральными числами от 1 до n . Главный офис находится в городе номер 1, таким образом, необходимо найти такой набор каналов связи, по которым можно доставить файл от города номер 1 до любого другого, а среди всех таких наборов выбрать имеющий наименьшую суммарную стоимость.

Задан список каналов связи, которыми может воспользоваться фирма. Напишите программу, находящую требуемый набор каналов связи.

Входные данные

Первая строка входного файла INPUT.TXT содержит числа n и m - количество городов и количество каналов связи соответственно ($1 \leq n \leq 22$, $0 \leq m \leq 22$). Последующие m содержат описания каналов связи. Каждое описание содержит три целых числа: u , v и c - соответственно номера городов, соединенных каналом и стоимость пересылки файла по этому каналу ($1 \leq u, v \leq n$, $0 \leq c \leq 1000$). Ни один из каналов не соединяет город с самим собой, но между двумя городами может быть больше одного канала.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите стоимость пересылки файла и число каналов, обеспечивающих такую стоимость. Во второй строке выведите номера каналов, составляющих такой набор. Каналы нумеруются от 1 до m в том порядке, в котором они перечислены во входном файле.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 2 1 2 3 1 2 4	3 1 1

2	3 3	14 2 2 3
	1 2 5	
	1 3 10	
	3 2 4	

```

/*
"Пересылка файлов": графы, рекурсивный перебор, сочетания,  $O(C(m, n-1) * m)$ 
*/

#include <stdio.h>
#include <vector>
#include <queue>

const int INF = (int)1e9;

struct Edge {
    int u, v, cost;
} edges[30];

int answ = INF, vect, used = 0;

int nVert, nEdges;

std::vector<int> g[30];

void go(const int id, const int need, const int sum) {
    if (need == 0) {
        std::vector<bool> visited(1+nVert, false);
        visited[1] = true;
        int nVisited = 1;
        std::queue<int> queue;
        queue.push(1);
        while (!queue.empty()) {
            auto curr = queue.front(); queue.pop();
            for (auto next : g[curr]) {
                if (!visited[next]) {
                    visited[next] = true;
                    ++nVisited;
                    if (!g[next].empty()) {
                        queue.push(next);
                    }
                }
            }
        }
        if (nVisited == nVert && sum < answ) {
            vect = used;
            answ = sum;
        }
        return;
    }
    // берем если need > 0
    if (need > 0) {
        const int u = edges[id].u;
        const int v = edges[id].v;
        used |= (1 << id);
        g[u].push_back(v);
        go(id+1, need-1, sum + edges[id].cost);
        g[u].pop_back();
        used &= ~(1 << id);
    }
    if (nEdges - id > need) {
        go(id+1, need, sum);
    }
}

int main() {
    scanf("%d %d", &nVert, &nEdges);
    if (nVert == 1) {

```

```

        printf("0 0");
        return 0;
    }

    for (int i = 0; i < nEdges; ++i) {
        scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].cost);
    }

    go(0, nVert-1, 0);

    std::vector<int> vector;
    for (int i = 0; i < nEdges; ++i) {
        if ((vect >> i) & 1) {
            vector.push_back(i+1);
        }
    }

    printf("%d %d\n", answ, (int)vector.size());
    for (auto& it : vector) {
        printf("%d ", it);
    }
    return 0;
}

```

ЗАДАЧА №206

Домой на электричках

(Время: 1 сек. Память: 16 Мб Сложность: 50%)

Одна из команд-участниц олимпиады решила вернуться домой на электричках. При этом ребята хотят попасть домой как можно раньше. К сожалению, не все электрички идут от города, где проводится олимпиада, до станции, на которой живут ребята. И, что еще более обидно, не все электрички, которые идут мимо их станции, останавливаются на ней (равно как вообще, электрички останавливаются далеко не на всех станциях, мимо которых они идут).

Все станции на линии пронумерованы числами от 1 до N. При этом станция номер 1 находится в городе, где проводится олимпиада, и в момент времени 0 ребята приходят на станцию. Станция, на которую нужно попасть ребятам, имеет номер E.

Напишите программу, которая по данному расписанию движения электричек вычисляет минимальное время, когда ребята могут оказаться дома.

Входные данные

Во входном файле INPUT.TXT записаны сначала числа N ($2 \leq N \leq 100$) и E ($2 \leq E \leq N$). Затем записано число M ($0 \leq M \leq 100$), обозначающее число рейсов электричек. Далее идет описание M рейсов электричек. Описание каждого рейса электрички начинается с числа K_i ($2 \leq K_i \leq N$) — количества станций, на которых она останавливается, а далее следует K_i пар чисел, первое число каждой пары задает номер станции, второе — время, когда электричка останавливается на этой станции (время выражается целым числом из диапазона от 0 до 10^9). Станции внутри одного рейса упорядочены в порядке возрастания времени. В течение одного рейса электричка все время движется в одном направлении — либо от города, либо к городу.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число — минимальное время, когда ребята смогут оказаться на своей станции. Если существующими рейсами электричек они добраться не смогут, выведите -1.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 3 4 2 1 5 2 10 2 2 10 4 15 4 5 0 4 17 3 20 2 35 3 1 2 3 40 4 45	20

```
#pragma GCC diagnostic ignored "-Wunused-result"

#include <stdio.h>
#include <bits/stdc++.h>

struct Edge {
    int vert, start, finish;
```

```

};

bool operator<(const Edge& a, const Edge& b) {
    return a.finish < b.finish || (a.finish == b.finish && (a.start < b.start || (a.start =
}

bool operator>(const Edge& a, const Edge& b) {
    return b < a;
}

int main() {
    int nVert, finish, nPaths;
    scanf("%d %d %d", &nVert, &finish, &nPaths);
    --finish;

    std::vector<std::vector<Edge>> edges(nVert);
    for (int i = 0; i < nPaths; ++i) {
        int k; scanf("%d", &k);
        int prev_vert, prev_time, curr_vert, curr_time = -1;
        for (int j = 0; j < k; ++j) {
            int v, t; scanf("%d %d", &v, &t); --v;
            prev_vert = curr_vert;
            prev_time = curr_time;
            curr_vert = v;
            curr_time = t;
            if (prev_time != -1) {
                edges[prev_vert].push_back(Edge{curr_vert, prev_time, curr_time});
            }
        }
    }

    for (auto& v : edges) std::sort(v.begin(), v.end());

    const int INF = (int)1e9+1;
    std::vector<int> dist(nVert, INF);
    dist[0] = 0;

    std::priority_queue<Edge, std::vector<Edge>, std::greater<Edge>> queue;
    queue.push(Edge{0, 0, 0});

    while (!queue.empty()) {
        auto curr = queue.top(); queue.pop();
        assert(dist[curr.vert] < INF);
        if (curr.vert == finish || curr.finish == INF) break;
        for (auto& next : edges[curr.vert]) {
            if (curr.finish > next.start) continue;
            if (next.finish < dist[next.vert]) {
                dist[next.vert] = next.finish;
                queue.push(Edge{next.vert, 0, next.finish});
            }
        }
    }

    printf("%d\n", dist[finish] == INF ? -1 : dist[finish]);
    return 0;
}

```

ЗАДАЧА №226

Перегоны

(Время: 1 сек. Память: 16 Мб Сложность: 50%)

На некоторой железнодорожной ветке расположено N станций, которые последовательно пронумерованы числами от 1 до N . Известны расстояния между некоторыми станциями. Требуется точно вычислить длины всех перегонов между соседними станциями или указать, что это сделать невозможно (то есть приведенная информация является противоречивой или ее недостаточно).

Входные данные

Во входном файле INPUT.TXT записаны сначала числа N — количество станций ($2 \leq N \leq 100$) и E — количество пар станций, расстояния между которыми заданы ($0 \leq E \leq 10000$). Далее, идет E троек чисел, первые два числа каждой тройки задают номера станций (это числа из диапазона от 1 до N), а третье — расстояние между этими станциями (все эти расстояния заданы точно и выражаются вещественными неотрицательными числами не более чем с 3-я знаками после десятичной точки).

Выходные данные

В случае, когда восстановить длины перегонов можно однозначно, в выходной файл OUTPUT.TXT выведите сначала «YES», а затем $N-1$ вещественное число. Первое из этих чисел должно соответствовать расстоянию от 1-й станции до 2-й, второе — от 2-й до 3-й, и так далее. Все числа должны быть выведены с точностью до 3-х знаков после десятичной точки (например, число 2.3 следует выводить как 2.300). Если приведенная информация о расстояниях между станциями является противоречивой или не позволяет однозначно точно восстановить длины перегонов, выведите в выходной файл «NO».

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 2 1 2 1.250 3 1 3	YES 1.250 1.750
2	4 4 1 2 1.250 3 1 1.255 2 4 0.010 1 1 0.000	YES 1.250 0.005 0.005
3	3 1 1 1 1	NO
4	3 3 1 2 1.250 1 3 1.300 2 3 1.000	NO

/*

Задача: 226. Перегоны

Решение: поиск в ширину, граф, восстановление графа, $O(n^3)$

*/

```
#include <bits/stdc++.h>
#define size(x) (int)(x).size()
typedef long long ll;
const ll UNDEF = LLONG_MIN;
struct Edge { int u, v; ll w; };
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::vector<ll> vl;
typedef std::vector<vl> vvl;
typedef std::vector<Edge> ve;
void assert_tle(bool q) {
    while (!q);
}
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int nV, nE;
    std::cin >> nV >> nE;
    vvi adj(nV);
    ve edges;
    bool ok = true;
    for (int i = 0, u, v; i < nE; ++i) {
        std::cin >> u >> v; u--, v--;
        std::string s; std::cin >> s;
        assert_tle(size(s) <= 14);
        ll fi = 0, se = 0;
        auto pos = s.find('.');
        if (pos == std::string::npos) {
            fi = std::stoll(s);
            se = 0;
        } else {
            fi = std::stoll(s.substr(0, pos));
            int cnt = 0;
            for (int j = (int)pos+1; j < size(s); ++j) {
                se *= 10;
                se += (s[j] - '0');
                ++cnt;
            }
            while (cnt < 3) {
                se *= 10;
                cnt++;
            }
        }
        assert_tle(se < 1000);
        ll w = fi * 1000 + se;
        assert(w <= 1000LL * 1000LL * 1000LL * 1000LL * 1000LL);
        edges.push_back(Edge{u, v, w});
        adj[u].push_back(size(edges)-1);
        adj[v].push_back(size(edges)-1);
        if (u == v && w != 0) {
            ok = false;
        }
    }
    if (!ok) {
        // assert(false); RE4
        std::cout << "NO\n"; return 0;
    }
    vvl dist(nV, vl(nV, UNDEF));
    for (int i = 0; i < nV; ++i) {
        dist[i][i] = 0;
    }
    std::queue<int> queue;
    queue.push(0);
    while (!queue.empty()) {
        auto u = queue.front(); queue.pop();
        assert(dist[0][u] != UNDEF);
        for (int id : adj[u]) {
```



```

        const auto& e = edges[id];
        int v = e.u + e.v - u;
        if (u != v && dist[0][v] == UNDEF) {
            dist[0][v] = dist[0][u] + (v < u ? -e.w : e.w);
            queue.push(v);
        }
    }
}
for (int u = 1; u < nV; ++u) {
    if (dist[0][u] < 0) {
        // assert(false); RE2
        std::cout << "NO\n";
        return 0;
    }
}
for (int u = 0; u < nV; ++u) {
    for (int v = u; v < nV; ++v) {
        if (dist[0][u] > dist[0][v]) {
            assert(false);
            std::cout << "NO\n";
            return 0;
        }
    }
}
for (int u = 1; u < nV; ++u) {
    for (int v = u+1; v < nV; ++v) {
        dist[u][v] = dist[0][v] - dist[0][u];
    }
}
for (int u = 0; u < nV; ++u) {
    for (int v = u+1; v < nV; ++v) {
        dist[v][u] = dist[u][v];
        /* WA14
        if (dist[u][v] == 0) {
            //assert(false);
            std::cout << "NO\n";
            return 0;
        }
        */
    }
}
for (int u = 0; u < nV; ++u) {
    for (int v = 0; v < nV; ++v) {
        if (dist[u][v] < 0) {
            //assert(false);
            std::cout << "NO\n";
            return 0;
        }
        for (int w = u; w <= v; ++w) {
            if (dist[u][w] + dist[w][v] != dist[u][v]) {
                //assert(false);
                std::cout << "NO\n";
                return 0;
            }
        }
    }
}
if (dist[u][u] != 0) {
    // assert(false);
    std::cout << "NO\n";
    return 0;
}
}
for (auto &e : edges) {
    if (dist[e.u][e.v] != e.w) {
        //assert(false);
        std::cout << "NO\n";
        return 0;
    }
}
}

```

```
std::cout << "YES\n";
for (int u = 0; u + 1 < nV; ++u) {
    ll d = dist[u][u+1];
    std::cout
        << d / 1000
        << '.' << std::setw(3) << std::setfill('0') << d % 1000
        << std::setfill(' ') << ' ';
}
return 0;
}
```

ЗАДАЧА №249

Скобки

(Время: 1 сек. Память: 16 Мб Сложность: 50%)

Назовем строку S правильной скобочной последовательностью, если она состоит только из символов '{', '}', '[', ']', '(', ')' и выполнено хотя бы одно из следующих трех условий:

1. S — пустая строка;
2. S можно представить в виде $S = S_1 + S_2 + S_3 + \dots + S_N$ ($N > 1$), где S_i — непустые правильные скобочные последовательности, а знак "+" обозначает конкатенацию (приписывание) строк;
3. S можно представить в виде $S = \{ ' + C + ' \}$ или $S = [' + C +]$ или $S = (' + C +)$, где C является правильной скобочной последовательностью.

Дана строка, состоящая только из символов '{', '}', '[', ']', '(', ')'. Требуется определить, какое минимальное количество символов надо вставить в эту строку для того, чтобы она стала правильной скобочной последовательностью.

Входные данные

В первой строке входного файла INPUT.TXT записана строка, состоящая только из символов '{', '}', '[', ']', '(', ')'. Длина строки не превосходит 100 символов.

Выходные данные

В выходной файл OUTPUT.TXT выведите ответ на поставленную задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	{() }	2
2	([{}])	0

```
#include <iostream>
#include <vector>
#include <algorithm>

bool correct(char l, char r) {
    return (l == '(' && r == ')') || (l == '[' && r == ']') || (l == '{' && r == '}');
}

int main() {
    std::string s; std::cin >> s;
    const int n = (int)s.size();
    std::vector<std::vector<int>> min(n, std::vector<int>(n)); // min[left][right]
    // min[i][i] = 1;
    // min[i][i+1] = 0 if "()" of "[]" of "{}", else 2

    for (int i = 0; i < n; ++i) {
        min[i][i] = 1;
    }

    for (int i = 0; i+1 < n; ++i) {
        auto sub = s.substr(i, 2);
```

```

        min[i][i+1] = correct(s[i], s[i+1]) ? 0 : 2;
    }

    for (int len = 3; len <= n; ++len) {
        for (int pos = 0; pos+len-1 < n; ++pos) {
            const int l1 = pos, r2 = pos+len-1;
            // Вариант S = '(' + C + ')':
            min[l1][r2] = min[l1+1][r2-1] + (correct(s[l1], s[r2]) ? 0 : 2);
            // Варианты S = S1 + S2:
            for (int r1 = l1; r1 < r2; ++r1) {
                const int l2 = r1+1;
                min[l1][r2] = std::min(min[l1][r2], min[l1][r1]+min[l2][r2]);
            }
        }
    }
    std::cout << min[0][n-1];
    return 0;
}

```

ЗАДАЧА №380

Площадь прямоугольников

(Время: 1 сек. Память: 16 Мб Сложность: 50%)

Дано N прямоугольников со сторонами, параллельными осям координат. Требуется определить площадь фигуры, образованной объединением данных прямоугольников.

Входные данные

В первой строке входного файла INPUT.TXT находится число прямоугольников - N. Затем идут N строк, содержащих по 4 числа: x_1, y_1, x_2, y_2 - координаты двух противоположных углов прямоугольника. Все координаты – целые числа, не превосходящие по абсолютной величине 10 000. ($1 \leq N \leq 100$)

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число – площадь фигуры.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 1 1 7 7	36
2	2 1 1 3 3 2 2 4 4	7

```
/*
    Задача: 380. Площадь прямоугольников

    Решение: Segment Tree, сканирующая прямая, сжатие координат, ленивое проталкивание, O(n)

    Автор: Дмитрий Козырев, github:dmkz, e-mail: dmkozyrev@rambler.ru
*/

/*
    Для начала сожмем координаты по OY, будем обрабатывать события по OX в порядке увеличения

    Имеем события:

        1) увеличить все значения на отрезке [y1, y2] на 1 в момент времени `x`
        2) уменьшить все значения на отрезке [y1, y2] на 1 в момент времени `x`
        3) посчитать количество ненулевых ячеек, которые были на отрезке [x1, x2]

    Таким образом, количество ненулевых ячеек от x_min до x_max и будет площадью объединения

    Для того, чтобы посчитать количество ненулевых ячеек, посчитаем количество нулевых, и в

    Для этого можно использовать sqrt-декомпозицию, или дерево отрезков. Нужно хранить (мин

    Значения в ячейках не могут стать меньше, чем 0.

    Авторское решение использует дерево отрезков, должно заходить за время O(n log(n)).

```

```

    Решение прокомментировано и содержит лишние проверки, потому что автор "плавает" в дере
*/

#include <stdio.h>
#include <iostream>
#include <vector>
#include <algorithm>
#include <cassert>
#include <numeric>
#include <ctime>
#include <cstdlib>

/*
    Структура узла в дереве отрезков:
    - `value` - значение, которое нужно присвоить всем элементам отрезка
    - `extra` - величина, которую нужно добавить всем элементам отрезка
    - `size` - количество элементов на отрезке (не равно длине отрезка), не изменяется
    - `min` - значение минимального элемента на отрезке
    - `cnt` - количество минимальных элементов на отрезке

    Соглашения:
    1) Если value != UNDEF: (в данной задаче только у листьев)
        а) extra = 0
        б) минимальный элемент равен value
        в) количество минимумов равно size
    2) Если extra != 0:
        б) минимальный элемент вычисляется как min + extra
        в) его актуальное количество в переменной cnt
*/

typedef long long ll;
const int INF = (int)1e9+1;
const int UNDEF = -1;

struct Node {
    int value, extra, size, min, cnt;

    Node(int value_ = UNDEF, int extra_ = 0, int size_ = 0, int min_ = INF, int cnt_ = 0)
        : value(value_), extra(extra_), size(size_), min(min_), cnt(cnt_) { }
};

// Пересчет текущего минимума и его количества относительно другого значения:
inline void remin(int& min, int& cnt, int value, int count) {
    if (min == value) {
        cnt += count;
    } else if (min > value) {
        min = value;
        cnt = count;
    }
}

struct SegmentTree {
    int size;

    std::vector<Node> data;

    // Конструктор от массивов из элементов на отрезках и длин отрезков:
    SegmentTree(const std::vector<int>& arr, const std::vector<int>& sizes) {
        size = (int)arr.size();
        int pow = 1;
        while (pow < size) pow *= 2;
        data.resize(2*pow);
        build(0, 0, size-1, arr, sizes);
    }

    // Пересчет значения в узле от значения в поддеревьях:
    void modify(int v, int l, int r, int m) {
        assert(data[v].value == UNDEF && data[v].extra == 0);
    }
};

```

```

    data[v].size = data[2*v+1].size + data[2*v+2].size;
    data[v].min = INF; data[v].cnt = 0;
    getmin(2*v+1, 1, m, 1, m, data[v].min, data[v].cnt);
    getmin(2*v+2, m+1, r, m+1, r, data[v].min, data[v].cnt);
}

// Проталкивание изменения от дерева в поддеревья:
void push(int v, int l, int r, int m) {
    // Если лист, то ничего проталкивать не нужно:
    if (l == r) return;
    assert(data[v].value == UNDEF);
    // Иначе если есть изменения, которые следует передать:
    if (data[v].extra != 0) {
        add(2*v+1, 1, m, 1, m, data[v].extra);
        add(2*v+2, m+1, r, m+1, r, data[v].extra);
        data[v].extra = 0;
    }
}

// Построение дерева рекурсивно:
void build(int v, int l, int r, const std::vector<int>& arr, const std::vector<int>& si)
    if (l == r) { // лист
        data[v] = Node(arr[l], 0, sizes[l], arr[l], sizes[l]);
    } else {
        int m = (l + r) >> 1;
        build(2*v+1, 1, m, arr, sizes);
        build(2*v+2, m+1, r, arr, sizes);
        modify(v, l, r, m);
    }
}

// Получение минимального элемента на отрезке и его количества
// предполагается, что изначально min = INF, cnt = 0
void getmin(int v, int l, int r, int ql, int qr, int& min, int& cnt) {
    if (qr < l || r < ql) return;
    ql = std::max(ql, l);
    qr = std::min(qr, r);
    if (l == ql && r == qr) { // Весь отрезок входит в запрос:
        if (data[v].value != UNDEF) {
            assert(data[v].extra == 0);
            remin(min, cnt, data[v].value, data[v].size);
        } else {
            remin(min, cnt, data[v].min + data[v].extra, data[v].cnt);
        }
    } else { // Отправляем запросы в поддеревья:
        int m = (l + r) >> 1;
        push(v, l, r, m);
        getmin(2*v+1, 1, m, ql, qr, min, cnt);
        getmin(2*v+2, m+1, r, ql, qr, min, cnt);
        modify(v, l, r, m);
    }
}

// Прибавление значения на отрезке:
void add(int v, int l, int r, int ql, int qr, int x) {
    if (qr < l || r < ql) return;
    ql = std::max(ql, l);
    qr = std::min(qr, r);
    if (l == ql && r == qr) { // Весь отрезок входит в запрос:
        if (data[v].value != UNDEF) {
            assert(data[v].extra == 0);
            data[v].value += x;
        } else {
            data[v].extra += x;
        }
    } else {
        int m = (l + r) >> 1;
        push(v, l, r, m);
        add(2*v+1, 1, m, ql, qr, x);
    }
}

```

```

        add(2*v+2, m+1, r, ql, qr, x);
        modify(v, l, r, m);
    }
}

// Более удобные функции под внешние запросы:
void add(int left, int right, int value) {
    add(0, 0, size-1, left, right, value);
}

void getmin(int left, int right, int& min, int& cnt) {
    min = INF, cnt = 0;
    getmin(0, 0, size-1, left, right, min, cnt);
}

};

struct Event { // Структура под событие arr[left...right] += value в момент времени time
    int time, left, right, value;
};

inline bool operator<(const Event& a, const Event& b) {
    if (a.time < b.time) return true;
    if (a.time > b.time) return false;
    if (a.value < b.value) return true;
    if (a.value > b.value) return false;
    return a.left < b.left || (a.left == b.left && a.right < b.right);
}

int main() {
    double time = (double)clock();
    // Чтение входных данных:
    int n; scanf("%d", &n);
    std::vector<Event> events;
    std::vector<int> coord;
    for (int i = 0; i < n; ++i) {
        int x1, y1, x2, y2;
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
        if (x2 < x1) std::swap(x1, x2);
        if (y2 < y1) std::swap(y1, y2);
        --y2; --x2;
        if (x2 < x1 || y2 < y1) continue;
        assert(x1 <= x2);
        assert(y1 <= y2);
        events.push_back(Event{ x1, y1, y2, 1});
        events.push_back(Event{x2+1, y1, y2, -1});
        coord.push_back(y1);
        coord.push_back(y2);
    }
    // Сжатие координат:
    std::stable_sort(coord.begin(), coord.end());
    coord.erase(std::unique(coord.begin(), coord.end()), coord.end());
    std::vector<int> arr, size;
    for (int i = 0; i < (int)coord.size(); ++i) {
        if (i > 0 && coord[i] - coord[i-1] > 1) {
            arr.push_back(coord[i-1]+1);
            size.push_back(coord[i] - coord[i-1]-1);
        }
        arr.push_back(coord[i]);
        size.push_back(1);
    }
    // Сортировка событий:
    std::sort(events.begin(), events.end());
    for (auto& e : events) {
        e.left = int(std::lower_bound(arr.begin(), arr.end(), e.left) - arr.begin());
        e.right = int(std::lower_bound(arr.begin(), arr.end(), e.right) - arr.begin());
    }
    // Создание дерева отрезков и обработка запросов:
    const int MAX = (int)size.size()-1;
    const int CNT = std::accumulate(size.begin(), size.end(), 0);

```



```

SegmentTree st(std::vector<int>(MAX+1,0), size);
int last = events.front().time-1; ll answ = 0;
for (auto& e : events) {
    if (e.time > last) {
        int min, cnt;
        st.getmin(0, MAX, min, cnt);
        int notZeros = CNT - (min == 0 ? cnt : 0);
        answ += std::max(0, e.time - last) * 1LL * notZeros;
        last = e.time;
    }
    st.add(e.left, e.right, e.value);
}
std::cout << answ << "\n";
fprintf(stderr, "time = %0.3fs\n", ((double)clock() - time) / CLOCKS_PER_SEC);
return 0;
}

```

ЗАДАЧА №611

Словарные квадраты

(Время: 1 сек. Память: 16 Мб Сложность: 50%)

Некоторые наборы из n слов длины n обладают интересным свойством - их можно расположить в клетках квадрата $n \times n$ так, что все слова набора можно прочесть как в вертикали, так и по горизонтали.

Примером такого набора слов является {"DATE", "FIND", "IDEA", "NEXT"}. Их можно расположить так:

F	I	N	D
I	D	E	A
N	E	X	T
D	A	T	E

Заметьте, что каждое слово можно прочесть как по горизонтали, так и по вертикали. Такие квадраты называются словарными квадратами, наибольший известный словарный квадрат в английском языке имеет размер 10×10 .

Рассмотрим еще один пример словарного квадрата:

C	R	A	B
R	A	R	E
A	R	T	S
B	E	S	T

Вам даны такие $2n$ слов, что из них можно построить два различных словарных квадрата размера $n \times n$. Ваша задача состоит в том, чтобы разбить эти слова на две группы, по n слов в каждой, и построить из слов каждой группы словарный квадрат.

Гарантируется, что все данные вам слова являются английскими словами (некоторые из них могут быть достаточно редкими словами, именами, или специальными терминами).

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n ($2 \leq n \leq 10$). Каждая из следующих $2n$ строк содержит слово, состоящее из заглавных букв английского алфавита. Каждое слово содержит ровно n букв.

Выходные данные

В выходной файл OUTPUT.TXT выведите два словарных квадрата, построенных из данных слов. Разделите квадраты пустой строкой.

Пример

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	4	CRAB
	ARTS	RARE
	BEST	ARTS
	CRAB	BEST
	DATE	
	FIND	FIND
	IDEA	IDEA
	NEXT	NEXT
	RARE	DATE

```

/*
    Задача: Словарные квадраты

    Решение: перебор, отсечение, O(n!)

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <stdio.h>
#include <cstdlib>

int n;
char field1[10][10], field2[10][10], empty1[10], empty2[10], word[20][10];

void input() {
    scanf("%d", &n);
    for (int i = 0; i < 2 * n; ++i) {
        for (int j = 0; j < n; ++j) {
            scanf(" %c", &word[i][j]);
        }
    }
}

void init() {
    for (int i = 0; i < n; ++i) {
        empty1[i] = empty2[i] = true;
    }
}

void go(int cur) {
    if (cur == 2*n) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                printf("%c", field1[i][j]);
            }
            printf("\n");
        }
        printf("\n");
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                printf("%c", field2[i][j]);
            }
            printf("\n");
        }
        std::exit(0);
    }
    // Определяем слово в первый квадрат
    for (int pos = 0; pos < n; ++pos) {
        if (empty1[pos]) {
            bool flag = true;
            for (int i = 0; i < n; ++i) {
                if (empty1[i]) continue;
                if (field1[i][pos] != word[cur][i]) {
                    flag = false;
                    break;
                }
            }
        }
    }
}

```

```

        if (flag) {
            empty1[pos] = false;
            for (int i = 0; i < n; ++i) {
                field1[i][pos] = word[cur][i];
                field1[pos][i] = word[cur][i];
            }
            go(cur+1);
            empty1[pos] = true;
        }
    }
}

// Определяем слово во второй квадрат:
for (int pos = 0; pos < n; ++pos) {
    if (empty2[pos]) {
        bool flag = true;
        for (int i = 0; i < n; ++i) {
            if (empty2[i]) continue;
            if (field2[i][pos] != word[cur][i]) {
                flag = false;
                break;
            }
        }
        if (flag) {
            empty2[pos] = false;
            for (int i = 0; i < n; ++i) {
                field2[i][pos] = word[cur][i];
                field2[pos][i] = word[cur][i];
            }
            go(cur+1);
            empty2[pos] = true;
        }
    }
}

}

int main() {
    input();
    init();
    go(0);
    throw 1;
}

```

ЗАДАЧА №625

SMS - 2

(Время: 1 сек. Память: 16 Мб Сложность: 50%)

В наше время непросто найти человека, который ни разу в жизни не использовал мобильный телефон для отправления текстовых сообщений. Старые телефоны поддерживали только один способ набора текста, упрощенно описывающийся следующими правилами: Каждой из восьми кнопок от «2» до «9» ставится в соответствие несколько букв, а кнопка «1» отводится для знаков препинания. Пробел ставится нажатием кнопки «0». Для ввода первого из соответствующих кнопке символов, ее надо нажать один раз, для ввода второго - два раза и так далее. Если два подряд идущие символа (кроме пробела) сопоставлены одной кнопке, то после ввода первого из них можно, либо подождать, либо нажать на кнопку перемещения курсора (второй вариант оказывается быстрее). Для переключения регистра букв используется кнопка «#». Кроме того, для удобства, после ввода вопросительного и восклицательного знаков, а так же точки (если на этот момент включен нижний регистр), активируется режим «первой заглавной буквы». При этом следующая буква автоматически печатается заглавной, после чего опять включается нижний регистр. В начале набора включен режим «первой заглавной буквы».

В последнее время в Интернете стали появляться результаты различных исследований, доказывающих неэффективность обычной раскладки телефонной клавиатуры, в которой буквы сопоставляются цифрам в алфавитном порядке. Составьте программу, которая по заданному сопоставлению букв кнопкам, будет находить минимальное количество нажатий, необходимых для максимально быстрого ввода данного текста при условии, что на каждое нажатие уходит одинаковое количество времени.

Входные данные

Строки с первой по девятую входного файла INPUT.TXT задают символы, сопоставленные соответствующим кнопкам. Следующая строка содержит сообщение длиной от 1 до 1000 символов. Гарантируется, что в первой строке находятся символы «?», «!» и «.», заданные в определенном порядке; в последующих 8 строках расположены все строчные английские символы от «a» до «z» без повторов, не менее одной буквы в строке. Текст сообщения содержит только те символы, которые возможно напечатать.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное число - минимальное количество нажатий на кнопки, требующееся для наискорейшего ввода сообщения.

Пример

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	.?! abc def ghi jkl mno pqrs tuv wxyz Hello. How do you do? i hope everything is fine See ya!	120
---	--	-----

```

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <string>
#include <functional>
#include <cassert>

struct Pair {
    int button, count;
};

enum State {LOWER, UPPER, FIRST};

int main() {
    // Чтение содержимого кнопок:
    std::vector<std::string> buttons(9);
    for (auto& it : buttons) {
        char buf[27]; char c;
        scanf("%26[^\n]%c", buf, &c); assert(c == '\n');
        it = buf;
    }

    // Получение номера кнопки и количества нажатий для символа c:
    std::function<Pair(char c)> get = [&](char c) {
        c = std::tolower(c);
        Pair answer{-1,-1};
        for (int id = 0; id < (int)buttons.size(); ++id) {
            if (int(buttons[id].find(c)) != -1) {
                answer = Pair{id, int(buttons[id].find(c)+1)};
                break;
            }
        }
        assert(answer.count > 0 && answer.button >= 0);
        return answer;
    };

    // Чтение исходной строки:
    char buf[1001];
    scanf("%1000[^\n]", buf);
    std::string s(buf);

    // Инициализация начального состояния и проход по строке:
    State state = FIRST;
    int prev = -1, answer = 0;
    for (auto c : s) {
        if (c == ' ') {
            answer++; prev = -1; continue;
        }
        auto res = get(c);
        if (std::isupper(c)) {
            answer += state == LOWER;
            if (state == LOWER) {
                state = UPPER;
            } else if (state == FIRST) {
                state = LOWER;
            }
        }
    }

```

```
    } else if (std::islower(c)) {
        answer += state != LOWER;
        state = LOWER;
    } else {
        assert(c == '.' || c == '?' || c == '!');
        if (state == LOWER) {
            state = FIRST;
        }
    }
    answer += (res.button == prev) + res.count;
    prev = res.button;
}
printf("%d", answer);
return 0;
}
```

ЗАДАЧА №640

Test-The-Best

(Время: 1 сек. Память: 16 Мб Сложность: 50%)

Конкурс Test-the-best, в котором участвуют лучшие программисты из Беларуси, России и других стран, проводит очный тур. Стараясь не отставать от времени, организаторы решили позаботиться о безопасности. В последнее время на рынке техники есть много аппаратуры, позволяющей осуществлять видеонаблюдение.

Широко распространены программы, позволяющие автоматически обрабатывать полученные результаты. Однако имея дело с участниками этих соревнований, на существующие разработки полагаться небезопасно. Поэтому у оргкомитета возникла необходимость написания собственной системы анализа изображений. Перед вами поставлена задача сравнения двух черно-белых изображений на клетчатой сетке. Изображения считаются одинаковыми, если множества черных пикселей в них могут быть получены друг из друга поворотом на 90, 180, или 270 градусов и, возможно, отражением относительно вертикальной оси.

Входные данные

Входной файл INPUT.TXT содержит описания двух изображений в следующем формате: первая строка содержит два целых числа n и m ($1 \leq n, m \leq 500$) - высоту и ширину изображения соответственно. Затем следуют n строк, содержащих по m символов: «#» обозначает черный пиксель, «.» - белый.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно слово: «Yes», если изображения одинаковы и «No» в противном случае.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	7 8###.. ..#..... 6 10#..... ...#..... ...##.....	Yes

2	1 1 # 1 1 .	No
3	1 3 #.# 1 4 #..#	No

```

/*
    "Test-The-Best": геометрия, преобразования фигур,  $O(n^2 \cdot \log(n))$ 
*/

```

```

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <set>
#include <queue>

```

```

typedef std::vector<char> Vector;
typedef std::vector<Vector> Matrix;

```

```

Matrix input() {
    int nRows, nCols;
    scanf("%d %d", &nRows, &nCols);

    Matrix temp(nRows, Vector(nCols));
    int minr = nRows, minc = nCols, maxr = -1, maxc = -1;
    for (int r = 0; r < nRows; ++r) {
        for (int c = 0; c < nCols; ++c) {
            char v; scanf(" %c", &v);
            temp[r][c] = (v == '#');
            if (temp[r][c]) {
                minr = std::min(minr, r);
                minc = std::min(minc, c);
                maxr = std::max(maxr, r);
                maxc = std::max(maxc, c);
            }
        }
    }
    if (maxr == -1) {
        return Matrix();
    }
    Matrix answ(maxr-minr+1, Vector(maxc-minc+1));
    for (int i = 0; i < (int)answ.size(); ++i) {
        for (int j = 0; j < (int)answ[0].size(); ++j) {
            answ[i][j] = temp[minr+i][minc+j];
        }
    }
    return answ;
}

```

```

Matrix rot(const Matrix& a) {
    const int nRows = a.size(), nCols = a[0].size();
    Matrix b(nCols, Vector(nRows));
    for (int r = 0; r < nRows; ++r) {
        for (int c = 0; c < nCols; ++c) {
            b[c][nRows-r-1] = a[r][c];
        }
    }
    return b;
}

```

```

Matrix sym(Matrix a) {
    for (auto& row : a) {
        std::reverse(row.begin(), row.end());
    }
}

```

```

    }
    return a;
}

int main() {
    auto a = input(), b = input();
    std::set<Matrix> set;
    std::queue<Matrix> queue;
    queue.push(a);
    while (!queue.empty()) {
        Matrix next;
        next = rot(queue.front());
        if (set.find(next) == set.end()) {
            set.insert(next);
            queue.push(next);
        }
        next = sym(queue.front());
        if (set.find(next) == set.end()) {
            set.insert(next);
            queue.push(next);
        }
        queue.pop();
    }
    for (auto& a : set) {
        if (a == b) {
            printf("Yes");
            return 0;
        }
    }
    printf("No");
    return 0;
}

```

ЗАДАЧА №649

Защищенный пароль

(Время: 2 сек. Память: 16 Мб Сложность: 50%)

Очень надежная и совершенно бесплатная операционная система «Vokna» известна своей безопасностью, так как при проектировании разработчики уделили большое внимание проблемам генерации паролей. Ядро операционной системы содержит в себе строку S длиной N символов. Генерация пароля происходит с использованием символов строки S . Паролем будем называть подстроку $S_{i:j}$ строки S длиной не менее одного и не более N символов. Подстрокой $S_{i:j}$ строки S называется строка, последовательно составленная из символов $S[i], S[i+1], S[i+2], \dots, S[j-1], S[j]$. Символы в строке нумеруются последовательно начиная с единицы. Пароль $S_{i:j}$ считается защищенным, если в нем встречается не более K одинаковых символов. Вашей задачей является по заданной строке S и числу K определить количество различных вариантов выбора защищенного пароля. Два варианта выбора пароля $S^1_{i:j}$ и $S^2_{i':j'}$ называются различными, если $i \neq i'$ или $j \neq j'$.

Входные данные

Первая строка входного файла INPUT.TXT содержит два натуральных числа N ($1 \leq N \leq 10^6$) и K ($1 \leq K \leq N$), разделенных одиночным пробелом, где N – количество символов в строке S ; K – максимальное количество одинаковых символов в пароле. Вторая строка входного файла содержит ровно N символов. Каждый символ является либо маленькой английской буквой, либо цифрой. Каждая строка входного файла заканчивается символом перевода строки.

Выходные данные

Единственная строка выходного файла OUTPUT.TXT должна содержать одно целое число – количество вариантов выбора защищенного пароля.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	6 2 7aaarr	15
2	4 1 auau	7

```
#include <stdio.h>
#include <string>
#include <iostream>

int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    static char s[1000000+1];
    scanf("%1000000s", s);
    static int count[256];
    int left = 0, right = 0;
    long long answ = 0;
    while (right < n) {
        const char cur = s[right];
        count[cur]++;
```

```
        while (count[cur] > k) {
            count[s[left]]--;
            ++left;
        }
        answ += right-left+1;
        ++right;
    }
    std::cout << answ;
    return 0;
}
```

ЗАДАЧА №653

Аттракцион

(Время: 1 сек. Память: 16 Мб Сложность: 50%)

На протяжении многих лет в Байтландии существует парк развлечений “Funny byte”, в котором представлено много различных аттракционов: колесо вычислений, веселые горки с трассами в форме интегралов, равнобедренная ромашка и многие другие.

В последние годы популярность “Funny byte” стала неуклонно падать. В целях привлечения посетителей руководство парка решило открыть новый аттракцион, который представляет собой сложный механизм.

Кресла аттракциона расположены в N рядов по M кресел в каждом. То есть каждое кресло характеризуется номером ряда и номером колонки, в котором оно находится. Ряды нумеруются последовательно сверху вниз начиная с единицы, колонки нумеруются слева направо начиная с единицы. Каждое кресло имеет свой уникальный номер. Кресло, находящееся в i -м ряду и в j -ой колонке, имеет номер $(i-1)*M + j$.

После посадки отдыхающих, кресла поднимают вверх над землей. И начинается веселье! Механизм аттракциона случайным образом производит некоторое количество операций. Под одной операцией понимается взаимная перестановка двух рядов либо двух колонок. При взаимной перестановке двух рядов или колонок каждое кресло в ряду или колонке заменится на соответствующее ему кресло в другом ряду или колонке.

И вот аттракцион завершил свою работу. Но есть одна трудность! Кресла надо вернуть в начальное положение при помощи таких же операций. Как количество, так и сами операции не обязательно должны быть идентичны тем, которые производил аттракцион во время сеанса. Руководство парка решило, что вернуть кресла в начальное положение необходимо не более чем за 1000 операций.

Такая задача оказалась не по силам разработчикам механизма. Помогите им! От вас требуется разработать программу, позволяющую вернуть кресла в начальное положение. Гарантируется, что решение существует.

Входные данные

В первой строке входного файла INPUT.TXT заданы два натуральных числа N и M ($1 \leq N, M \leq 250$). В последующих N строках задано по M натуральных чисел, где j -е число в $i+1$ -й строке соответствует номеру кресла после окончания сеанса аттракциона. Числа в строках разделяются одиночными пробелами.

Выходные данные

В первой строке выходного файла OUTPUT.TXT должно быть выведено количество операций перестановки K , которое не должно превышать 1000. Каждая из следующих K строк описывает одну операцию. Каждая операция описывается строкой вида $Q\ X\ Y$, где Q – символ 'R'(ASCII 82) либо символ 'C'(ASCII 67). Если Q равно 'R', то данная операция является перестановкой рядов, если Q равно 'C', то операция является перестановкой колонок. X и Y – два натуральных числа, соответствующие номерам рядов (колонок), которые будут переставлены в результате данной операции. Операции должны быть выведены в порядке осуществления, то есть последовательное применение которых позволит вернуть кресла в начальное положение.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 2 4 3 2 1	2 C 1 2 R 1 2
2	3 5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	0
3	4 5 10 7 9 8 6 15 12 14 13 11 20 17 19 18 16 5 2 4 3 1	5 R 1 4 C 1 5 C 3 4 R 2 4 R 3 4

```

/*
    "Аттракцион": сортировка, конструктив,  $O(n^2 \cdot m + n \cdot m^2)$ 
*/

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>

struct Triple {
    char type; int a, b;
};

int main() {
    int nRows, nCols;
    scanf("%d %d", &nRows, &nCols);

    std::vector<std::vector<int>> arr(nRows, std::vector<int>(nCols));

    int i1 = -1, j1 = -1;
    for (int i = 0; i < nRows; ++i) {
        for (int j = 0; j < nCols; ++j) {
            scanf("%d", &arr[i][j]);
            if (arr[i][j] == 1) {
                i1 = i;
                j1 = j;
            }
        }
    }
    assert(i1 >= 0 && j1 >= 0);

    std::vector<Triple> answer;

    // Сортировка столбцов:
    for (int j = 0; j < nCols; ++j) {
        int jmin = j;
        for (int k = j+1; k < nCols; ++k) {
            if (arr[i1][jmin] > arr[i1][k]) {
                jmin = k;
            }
        }
        if (jmin == j) {
            continue;
        }
        answer.push_back(Triple{'C', j+1, jmin+1});
        for (int i = 0; i < nRows; ++i) {
            std::swap(arr[i][jmin], arr[i][j]);
        }
    }
}

```

```

}

// Сортировка строк:
for (int i = 0; i < nRows; ++i) {
    int imin = i;
    for (int k = i+1; k < nRows; ++k) {
        if (arr[k][0] < arr[imin][0]) {
            imin = k;
        }
    }
    if (imin == i) {
        continue;
    }
    answer.push_back(Triple{'R', i+1, imin+1});
    for (int j = 0; j < nCols; ++j) {
        std::swap(arr[imin][j], arr[i][j]);
    }
}

// Вывод ответа:
printf("%d\n", (int)answer.size());
for (auto& t : answer) {
    printf("%c %d %d\n", t.type, t.a, t.b);
}

return 0;
}

```

ЗАДАЧА №724

Убить Вову

(Время: 1 сек. Память: 16 Мб Сложность: 50%)

Таня хочет убить Вову. Разумеется, виртуально. Сначала Вова выбирает одну из n тактик защиты, потом Таня, не зная, какую тактику выбрал Вова, атакует несколько раз, каждый раз выбирая одну из m тактик нападения. Для каждой тактики защиты известно, от каких тактик нападения она защищает. Помогите Тане узнать, за сколько атак она может гарантированно убить Вову и какие тактики нападения ей нужно для этого использовать.

Входные данные

Первая строка входного файла INPUT.TXT содержит два числа - n и m . ($1 \leq n, m \leq 20$). Далее идут n строк по m чисел, j -е число в i -ой строке равно 1, если тактика защиты i спасает от тактики нападения j и равно 0 в противном случае.

Выходные данные

Выведите в выходной файл OUTPUT.TXT число k - минимальное число атак, за которое можно гарантированно убить Вову и далее k чисел - номера стратегий нападения, которые нужно использовать. Если Вову убить невозможно, выведите строку «Impossible».

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 5 1 0 0 0 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1 0	3 2 4 5
2	3 3 1 1 1 0 1 0 0 0 1	Impossible

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>

int main() {
    int nRows, nCols;
    scanf("%d %d", &nRows, &nCols);
    std::vector<std::vector<int>> arr(nRows, std::vector<int>(nCols));
    for (auto& row : arr) {
        int nZeros = 0;
        for (auto& it : row) {
            scanf("%d", &it);
            nZeros += (it == 0);
        }
        if (nZeros == 0) {
            printf("Impossible");
        }
    }
}
```



```

        return 0;
    }
}

std::vector<int> used(nRows, false), answer;
int nUsed = 0;
while (nUsed < nRows) {
    int max = 0, max_j = -1;
    for (int j = 0; j < nCols; ++j) {
        int count = 0;
        for (int i = 0; i < nRows; ++i) {
            if (used[i] || arr[i][j] == 1) continue;
            ++count;
        }
        if (count > max) {
            max = count;
            max_j = j;
        }
    }
    assert(max_j != -1);
    const int j = max_j;
    answer.push_back(j+1);
    for (int i = 0; i < nRows; ++i) {
        if (used[i] || arr[i][j] == 1) continue;
        used[i] = true;
        ++nUsed;
    }
}
printf("%d\n", (int)answer.size());
for (auto& it : answer) {
    printf("%d ", it);
}
return 0;
}

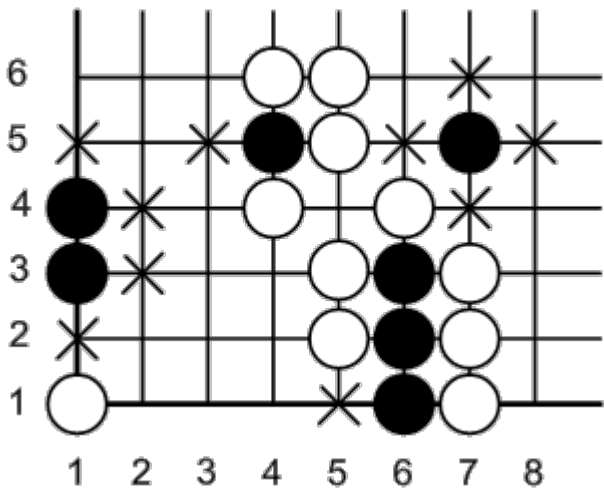
```

ЗАДАЧА №765

Го

(Время: 1 сек. Память: 16 Мб Сложность: 50%)

Совсем недавно Али-Баба узнал от своего брата Касима об удивительной игре Го. В Го играют на прямоугольной доске – гобане, расчерченном вертикальными и горизонтальными линиями. Все линии пронумерованы. В игре участвуют два игрока, которые по очереди выставляют на гобан камни – специальные круглые фишки. Каждый камень ставится на незанятую точку пересечения линий доски (пересечения называют пунктами). У одного игрока – черные камни, у другого – белые. Камни одного цвета, смежные по вертикали, либо по горизонтали (но не диагонали), объединяются в группу. Одиночный камень также считается группой.



Один из способов набрать очки в Го – захватить камни противника. Каждый камень может иметь от двух до четырех смежных с ним пунктов (по вертикали и горизонтали, но не по диагонали). Если такой пункт не занят камнем, то он называется «дамэ». Дамэ группы – это все дамэ камней, составляющих группу. Как только оппонент своими камнями закрывает все дамэ чужой группы, то эта группа считается захваченной и снимается с доски. Если у группы осталось лишь одно дамэ, то говорят, что эта группа находится в «атари» т.е. на один шаг от захвата соперником.

Дамэ черных камней на рисунке отмечены крестиком. Группа черных из камней (1, 3) и (1, 4) имеет 4 дамэ. Группа (6, 1), (6, 2) и (6, 3) имеет одно дамэ и находится в атари. Черный камень (4, 5) также находится в атари. Помогите Али-Бабе, который всегда играет черными, определить, какие его группы находятся в атари.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число N – размерность игровой доски ($6 \leq N \leq 19$). Далее следует N строк по N символов каждая. Каждый символ описывает один пункт доски. «B» означает черный камень, «W» – белый, «.» означает пустой пункт. Все группы на доске имеют хотя бы одно дамэ.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – количество групп черных камней, находящихся в атари.

Примеры

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	9WW.... ...BW.B.. B..W.W... B...WBW..WBW.. W....BW..	2
2	6 WB.WBB .B.W.B ..WW.W WWW..W ..W... BBW...	1

/*

Задача: 765. Го

Решение: графы, поиск в глубину, компоненты связности, $O(n^2)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

*/

```
#include <bits/stdc++.h>
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::vector<std::string> vs;
int main() {
    for (int n; std::cin >> n; ) {
        vs field(n);
        for (int i = 0; i < n; ++i) { std::cin >> field[i]; }
        vvi part(n, vi(n, 0));
        int nParts = 0, answ = 0;
        for (int row = 0; row < n; ++row) {
            for (int col = 0; col < n; ++col) {
                if (field[row][col] == 'B' && !part[row][col]) {
                    int cur = 0;
                    std::function<void(int,int)> dfs = [&](int r, int c) {
                        if (r < 0 || r >= n || c < 0 || c >= n) { return; }
                        if (part[r][c] == nParts || field[r][c] == 'W') { return; }
                        part[r][c] = nParts;
                        if (field[r][c] == '.') { cur++; return; }
                        for (int dr = -1; dr <= 1; ++dr) {
                            for (int dc = -1; dc <= 1; ++dc) {
                                if (dr * dr + dc * dc == 1) {
                                    dfs(r+dr, c+dc);
                                }
                            }
                        }
                    };
                    ++nParts;
                    dfs(row,col);
                    answ += (cur == 1);
                }
            }
        }
        std::cout << answ << std::endl;
    }
    return 0;
}
```

ЗАДАЧА №874

Гирлянда

(Время: 1 сек. Память: 16 Мб Сложность: 50%)

Приближается Новый Год, и в магазинах начинают появляться различные елочные украшения. На прилавках можно увидеть различные шарики, шишечки, звездочки, но все-таки самым красивым украшением является гирлянда из разноцветных лампочек. Одна из фирм, занимающихся изготовлением елочных украшений, решила в этом году изготавливать гирлянды на заказ.

Гирлянды, изготавливаемые этой фирмы, состоят из лампочек различных цветов, соединенных проводами. Всего в гирлянде n лампочек, каждая из которых покрашена в один из k цветов, и m проводов (каждый провод соединяет ровно две лампочки). Далее мы будем считать, что лампочки пронумерованы натуральными числами от 1 до n .

К сожалению, не каждый дизайн гирлянды соответствует эстетическим взглядам заказчиков. Во-первых, лампочки, соединенные одним проводом должны быть разного цвета, во-вторых, сама конфигурация гирлянды (то есть то, какие лампочки и как соединены проводами) не может быть любой.

Один из отделов фирмы уже провел исследование и нашел наиболее «удачную» конфигурацию. Ваша же задача состоит в том, чтобы найти число способов раскрасить лампочки, чтобы получившаяся гирлянда удовлетворяла эстетическим взглядам заказчиков.

Входные данные

Первая строка входного файла INPUT.TXT содержит три целых числа: n, k, m ($1 \leq n, k \leq 8, 0 \leq m \leq 10$). Последующие m строк описывают провода. Описание каждого провода состоит из двух чисел u и v ($1 \leq u, v \leq n, u \neq v$) – номеров лампочек, соединенных этим проводом.

Выходные данные

В выходной файл OUTPUT.TXT выведите ответ на задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 2 1 1 2	2
2	4 4 0	256
3	4 4 6 1 2 1 3 1 4 2 3 2 4 3 4	24

Решение: перебор, битмаски, $O((n+m) \cdot k^n)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

*/

```
#include <bits/stdc++.h>

struct Edge { int u, v; };

template<const int nC>
int solve(const int nV, const std::vector<Edge>& edges) {
    int answ = 0; const int limit = (int)std::pow(nC, nV);
    std::vector<int> color(nV);
    for (int mask = 0; mask < limit; ++mask) {
        for (int temp = mask, i = 0; i < nV; ++i) {
            color[i] = temp % nC;
            temp /= nC;
        }
        bool ok = true;
        for (auto &e : edges) {
            if (color[e.u] == color[e.v]) {
                ok = false;
                break;
            }
        }
        answ += ok;
    }
    return answ;
}

int main() {
    int nV, nC, nE;
    while (std::cin >> nV >> nC >> nE) {
        std::vector<Edge> edges(nE);
        for (auto &e : edges) { std::cin >> e.u >> e.v; e.u--, e.v--; }
        int answ = 0;
        switch(nC) {
            case 1: answ = solve<1>(nV, edges); break;
            case 2: answ = solve<2>(nV, edges); break;
            case 3: answ = solve<3>(nV, edges); break;
            case 4: answ = solve<4>(nV, edges); break;
            case 5: answ = solve<5>(nV, edges); break;
            case 6: answ = solve<6>(nV, edges); break;
            case 7: answ = solve<7>(nV, edges); break;
            case 8: answ = solve<8>(nV, edges); break;
        };
        std::cout << answ << std::endl;
    }
    return 0;
}
```

ЗАДАЧА №991

ePig

(Время: 2 сек. Память: 16 Мб Сложность: 50%)

Андрей и Аня разрабатывают новую P2P сеть для обмена файлами, они назвали свою сеть ePig. В этой задаче вам предлагается смоделировать работу сети при распространении одного большого файла.

Пусть сетью пользуются n клиентов, пронумерованных от 1 до n . Исходно файл целиком доступен на клиенте номер 1. Остальные клиенты хотели бы получить этот файл. Для оптимизации процесса файл разбивается на k идентичных фрагментов, пронумерованных от 1 до k . Передача файла состоит из нескольких раундов. Каждый раунд занимает одну минуту, за время раунда каждый клиент может получить ровно один фрагмент и/или передать ровно один фрагмент. После того как клиент скачивает фрагмент файла, он может раздавать его другим клиентам.

Перед каждым раундом каждый клиент решает, какой фрагмент он будет запрашивать. Клиент запрашивает фрагмент, который предоставляется наименьшим числом клиентов (разумеется, кроме тех фрагментов, которые у него уже есть). Если таких фрагментов несколько, он выбирает тот, у которого минимальный номер.

После этого клиенты делают запросы на фрагменты. Каждый клиент выбирает другого клиента, у которого он запрашивает выбранный фрагмент. Если несколько клиентов предоставляют необходимый фрагмент, то выбирается клиент, у которого минимальное количество предоставляемых им фрагментов. Если и таких клиентов несколько, выбирается клиент с минимальным номером.

Каждый клиент рассматривает все запросы, которые к нему поступили, и выбирает один из них. Клиент X удовлетворяет тот из запросов, который приходит от самого ценного клиента. Ценность клиента определяется количеством фрагментов, которые клиент X получал от него ранее. Если имеется несколько клиентов с одинаковой ценностью, то фрагмент отдается тому из клиентов, у которого перед раундом имеется минимальное количество фрагментов. Если и таких клиентов несколько, то фрагмент отдается клиенту с минимальным номером.

После того, как выбрано, какие запросы будут удовлетворены, начинается раунд. Клиенты, запросы которых были отклонены, ничего не скачивают в этот раунд, а остальные клиенты скачивают запрошенные фрагменты. После этого начинается новый раунд, и т. д.

По заданным n и k для каждого клиента определите число раундов, которое потребуется, чтобы он получил файл целиком.

Входные данные

Входной файл INPUT.TXT содержит два целых числа: n и k ($2 \leq n \leq 100$, $1 \leq k \leq 200$).

Выходные данные

В выходной файл OUTPUT.TXT выведите для каждого клиента кроме первого одно число — количество раундов перед тем, как он скачает файл целиком.

Пример

Пояснение

Распространение файла в данном случае происходит следующим образом: в первом раунде клиенты 2 и 3 запрашивают фрагмент 1 от клиента 1. Удовлетворяется запрос от клиента 2. После этого клиенты 2 и 3 запрашивают фрагмент 2 у клиента 1. Удовлетворяется клиент 3. Наконец в третьем раунде клиент 2 запрашивает фрагмент 2 у клиента 3, а клиент 3 запрашивает фрагмент 1 у клиента 2, оба запроса удовлетворяются и у каждого теперь есть целый файл.

№	INPUT.TXT	OUTPUT.TXT
1	3 2	3 3

```
/*
    Задача: 991. ePig

    Решение: моделирование, реализация,  $O(n^2 \cdot k^2)$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>
#include <string>

struct Query {
    int client, piece;
};

struct Transfer {
    int from, to, what;
};

int main() {
    int nClients, nPieces;
    scanf("%d %d", &nClients, &nPieces);

    std::vector<std::vector<bool>> has(1+nClients, std::vector<bool>(1+nPieces));
    std::vector<std::vector<int>> nSent(1+nClients, std::vector<int>(1+nClients));
    std::vector<int> countPerPiece(1+nPieces, 0);
    std::vector<int> countPerClient(1+nClients, 0);

    for (int k = 1; k <= nPieces; ++k) {
        has[1][k] = true;
        countPerClient[1]++;
        countPerPiece[k]++;
    }

    std::vector<std::vector<Query>> queries(1+nClients);

    int time = 0;

    std::vector<int> answer(1+nClients, 0);

    std::vector<Transfer> queue;

    while (true) {
        bool finished = true;
        for (int id = 1; id <= nClients; ++id) {
            if (countPerClient[id] != nPieces) {
                finished = false;
                break;
            }
        }
    }
```

```

    if (finished) {
        break;
    }
    ++time;

    for (int id = 2; id <= nClients; ++id) {
        if (answer[id]) continue;
        int need = 0;
        for (int i = 1; i <= nPieces; ++i) {
            if (!has[id][i] && (need == 0 || countPerPiece[i] < countPerPiece[need])) {
                need = i;
            }
        }
        assert(need != 0);
        int from = 0;
        for (int i = 1; i <= nClients; ++i) {
            if (has[i][need] && (from == 0 || countPerClient[i] < countPerClient[from]))
                from = i;
        }
        assert(from != 0);
        queries[from].push_back(Query{id, need});
    }

    for (int from = 1; from <= nClients; ++from) {
        if (queries[from].empty()) {
            continue;
        }
        int to = 0, what = 0;
        for (auto& q : queries[from]) {
            if (to == 0) {
                to = q.client;
                what = q.piece;
                continue;
            }
            if (nSent[q.client][from] < nSent[to][from]) {
                continue;
            }
            if (nSent[q.client][from] > nSent[to][from]) {
                to = q.client;
                what = q.piece;
                continue;
            }
            if (countPerClient[q.client] > countPerClient[to]) {
                continue;
            }
            if (countPerClient[q.client] < countPerClient[to]) {
                to = q.client;
                what = q.piece;
                continue;
            }
        }
        assert(to < q.client);
    }
    queue.push_back(Transfer{from, to, what});
    queries[from].clear();
    assert(to != 0 && what != 0 && !has[to][what] && has[from][what]);
}

for (auto& it : queue) {
    nSent[it.from][it.to]++;
    countPerPiece[it.what]++;
    countPerClient[it.to]++;
    has[it.to][it.what] = true;
    if (countPerClient[it.to] == nPieces) {
        answer[it.to] = time;
    }
}

queue.clear();

```



```
    }  
    for (int id = 2; id <= nClients; ++id) {  
        printf("%d ", answer[id]);  
    }  
    return 0;  
}
```

ЗАДАЧА №598

Друзья - 2

(Время: 1 сек. Память: 16 Мб Сложность: 51%)

Несколько человек решили поехать отдохнуть на природе, подышать свежим воздухом и т.п. Как это часто бывает, некоторые из них дружат друг с другом, а некоторые - нет. Для того, чтобы не испортить никому настроение, они решили разделиться на несколько групп. При этом, в каждой группе должно быть не более 5 человек и они должны дружить друг с другом.

Найдите такое разбиение людей на группы, в котором размер наибольшей группы был бы максимальным (среди всех разбиений).

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n ($1 \leq n \leq 15$) - количество людей. Следующие n строк содержат по n чисел. Если i -ый и j -ый люди дружат, то j -ое число $i + 1$ -ой строки равно 1, иначе - 0.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите число групп. Во второй строке выходного файла выведите n чисел (i -ое число - номер группы, в которой находится i -ый человек). Так как в любом случае количество групп не превышает n , нумеруйте группы целыми числами от 1 до n . Если решений несколько, то выведите любое.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 1 1 0 1 1 0 0 0 1	2 1 1 2
2	8 1	3 1 1 1 1 1 2 2 3

```
/*
    Задача: Друзья - 2

    Решение: перебор,  $O(2^n)$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <stdio.h>
```

```

#include <vector>

#define size(x) (int)(x).size()

int n, g[16][16];
std::vector<int> answ, team;

void go(int cur) {
    if (size(team) > size(answ)) {
        answ = team;
    }
    if (size(team) == 5 || size(team) == n || cur > n) {
        return;
    }
    bool flag = true;
    for (auto u : team) {
        if (!g[u][cur]) {
            flag = false;
            break;
        }
    }
    if (flag) {
        team.push_back(cur);
        go(cur+1);
        team.pop_back();
    }
    go(cur+1);
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) {
            scanf("%d", &g[i][j]);
        }
    }
    go(1);
    std::vector<int> group(1+n);
    for (auto it : answ) group[it] = 1;
    int cnt = 1;
    for (int i = 1; i <= n; ++i) {
        if (group[i] == 0) {
            group[i] = ++cnt;
        }
    }
    printf("%d\n", cnt);
    for (int i = 1; i <= n; ++i) {
        printf("%d ", group[i]);
    }
    return 0;
}

```

ЗАДАЧА №796

Форматирование текста

(Время: 1 сек. Память: 16 Мб Сложность: 51%)

Многие системы форматирования текста, например TEX или Wiki, используют для разбиения текста на абзацы пустые строки. Текст представляет собой последовательность слов, разделенных пробелами, символами перевода строк и следующими знаками препинания: «,», «.», «?», «!», «-», «:» и «'» (ASCII коды 44, 46, 63, 33, 45, 58, 39). Каждое слово в тексте состоит из заглавных и прописных букв английского алфавита и цифр. Текст может состоять из нескольких абзацев. В этом случае соседние абзацы разделяются одной или несколькими пустыми строками. Перед первым абзацем и после последнего абзаца также могут идти одна или несколько пустых строк.

Дальнейшее использование исходного текста предполагает его форматирование, которое осуществляется следующим образом. Каждый абзац должен быть разбит на строки, каждая из которых имеет длину не больше w . Первая строка каждого абзаца должна начинаться с отступа, состоящего из b пробелов. Слова внутри одной строки должны быть разделены ровно одним пробелом. Если после слова идет один или несколько знаков препинания, они должны следовать сразу после слова без дополнительных пробелов. Если очередное слово вместе со следующими за ним знаками препинания помещается на текущую строку, оно размещается на текущей строке. В противном случае, с этого слова начинается новая строка. В отформатированном тексте абзацы не должны разделяться пустыми строками. В конце строк не должно быть пробелов.

Требуется написать программу, которая по заданным числам w и b и заданному тексту выводит текст, отформатированный описанным выше образом.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа: w и b ($5 \leq w \leq 100$, $1 \leq b \leq 8$, $b < w$). Затем следует одна или более строк, содержащих заданный текст. Длина слова в тексте вместе со следующими за ними знаками препинания не превышает w , а длина первого слова любого абзаца вместе со следующими за ним знаками препинания не превышает $(w - b)$. Текст содержит хотя бы одну букву. Перед первой буквой каждого абзаца знаков препинания нет.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать заданный текст, отформатированный в соответствии с приведенными в условии задачи правилами. Размер входного файла не превышает 100 Кбайт. Длина каждой строки во входном файле не превышает 250.

Пример

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	20 4 Yesterday, All my troubles seemed so far away, Now it looks as though they're here to stay, Oh, I believe in yesterday. Suddenly, I'm not half the man I used to be, There's a shadow hanging over me, Oh, yesterday came suddenly...	Yesterday, All my troubles seemed so far away, Now it looks as though they' re here to stay, Oh, I believe in yesterday. Suddenly, I' m not half the man I used to be, There' s a shadow hanging over me, Oh, yesterday came suddenly...
---	--	---

```

/*
    Задача: 796. Форматирование текста

    Решение: строки, разбор строк, анализ текста, O(n)

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>

typedef std::list<std::string> ls;

bool isSpaces(const std::string& s) {
    for (auto it : s) {
        if (!std::isspace(it)) {
            return false;
        }
    }
    return true;
}

ls read() {
    ls ret;
    std::string cur;
    while (std::cin && cur.empty()) {
        cur.clear();
        std::getline(std::cin, cur);
        if (isSpaces(cur)) { cur.clear(); }
    }
    while (std::cin && !cur.empty()) {
        std::stringstream ss(cur);
        cur.clear();
        while (ss >> cur) {
            ret.push_back(cur);
            assert(!cur.empty());
        }
        cur.clear();
        std::getline(std::cin, cur);
        if (isSpaces(cur)) { cur.clear(); }
    }
    return ret;
}

bool isComma(char c) {
    return c == 44 || c == 46 || c == 63 || c == 33 || c == 45 || c == 58 || c == 39;
}

void solve(const int w, const int b) {
    ls data = read();
    if (data.empty()) { return; }
    std::string s(b, ' ');
    for (auto it = data.begin(); it != data.end(); it++) {
        for (int i = 0; i + 1 < (int)it->size(); ++i) {

```

```

        if (isComma((*it)[i]) && !isComma((*it)[i+1])) {
            assert(!std::isspace((*it)[i+1]));
            data.insert(std::next(it), it->substr(i+1));
            it->resize(i+1);
            break;
        }
    }
}

for (auto it = data.begin(); std::next(it) != data.end(); ) {
    auto next = std::next(it);
    int last = 0;
    while (last < (int)next->size() && isComma((*next)[last])) {
        ++last;
    }
    *it += next->substr(0, last);
    next->erase(0, last);
    if (next->empty() || isSpaces(*next)) {
        data.erase(next);
    } else {
        it = next;
    }
}

for (auto it = data.begin(); it != data.end(); ) {
    if (int(s.size() + it->size()) <= w) {
        s += *it;
        s += " ";
        it++;
    } else {
        if (s.back() == ' ') { s.pop_back(); }
        std::cout << s << '\n';
        s = "";
    }
}

if (s.empty()) return;
if (s.back() == ' ') { s.pop_back(); }
assert(!s.empty());
std::cout << s << '\n';
s = "";
}

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0);
    int w, b; std::cin >> w >> b;
    while (std::cin) { solve(w,b); }
    return 0;
}

```

ЗАДАЧА №835

Задача о рюкзаке

(Время: 1 сек. Память: 16 Мб Сложность: 51%)

Одной из классических NP-полных задач является так называемая «Задача о рюкзаке». Формулируется она следующим образом. Дано n предметов, каждый из которых характеризуется весом w_i и полезностью p_i . Необходимо выбрать некоторый набор этих предметов так, чтобы суммарный вес этого набора не превышал W , а суммарная полезность была максимальна.

Ваша задача состоит в том, чтобы написать программу, решающую задачу о рюкзаке.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральные числа n ($1 \leq n \leq 20$) и W ($1 \leq W \leq 10^9$). Каждая из последующих n строк содержит описание одного предмета. Каждое описание состоит из двух чисел: w_i — веса предмета и p_i — его полезности ($1 \leq w_i, p_i \leq 10^9$).

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите количество выбранных предметов и их суммарную полезность. Во второй строке выведите через пробел их номера в возрастающем порядке (предметы нумеруются с единицы в порядке, в котором они перечислены во входном файле).

Если искомым наборов несколько, выберите тот, в котором наименьшее число предметов. Если же после этого ответ по-прежнему неоднозначен, выберите тот набор, в котором первый предмет имеет наименьший возможный номер, из всех таких выберите тот, в котором второй предмет имеет наименьший возможный номер, и т.д.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 10 10 100 9 80	1 100 1
2	5 100 80 1000 50 550 50 550 50 550 50 550	2 1100 2 3
3	6 100 80 1000 50 550 50 550 50 550 50 550 100 1100	1 1100 6

Задача: 835. Задача о рюкзаке

Решение: рекурсивный перебор, $O(2^n)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

*/

```
#include <cstdio>
#include <vector>
#include <algorithm>

typedef long long ll;

int n, w[20], p[20];
ll curr_mask, curr_p, curr_w, best_mask, best_p, limit;

#define nOnes(x) __builtin_popcountll(x)

void brute(int i) {
    if (curr_w > limit) return;
    if (i == n) {
        if (curr_p > best_p) {
            best_mask = curr_mask;
            best_p = curr_p;
            return;
        }
        if (curr_p < best_p) return;
        int nc = nOnes(curr_mask);
        int nb = nOnes(best_mask);
        if (nc < nb || (nc == nb && curr_mask > best_mask)) {
            best_mask = curr_mask;
            best_p = curr_p;
        }
        return;
    }
    // берем i-го
    curr_w += w[i];
    curr_p += p[i];
    curr_mask |= 1ll(1) << (n-i);
    brute(i+1);
    curr_w -= w[i];
    curr_p -= p[i];
    curr_mask &= ~(1ll(1) << (n-i));
    // не берем i-го
    brute(i+1);
}

int main() {
    scanf("%d %lld", &n, &limit);
    for (int i = 0; i < n; ++i) {
        scanf("%d %d", &w[i], &p[i]);
    }
    brute(0);
    printf("%d %lld\n", nOnes(best_mask), best_p);
    for (int i = 0; i < n; ++i) {
        if ((best_mask >> (n - i)) & 1) {
            printf("%d ", i+1);
        }
    }
    return 0;
}
```


Заклинание Ауэrsa

(Время: 1 сек. Память: 16 Мб Сложность: 51%)

Заклинание Ауэrsa представляет собой набор звуков, записываемых малыми буквами английского алфавита без пробелов. Сам по себе этот набор не обладает ни секретностью, ни магической силой и приведен во всех учебниках по белой магии в соответствующем разделе. Там же указан способ его применения:

- а) разбить набор букв на палиндромы, то есть непустые слова, которые читаются одинаково как справа налево, так и слева направо;
- б) произносить их по порядку, после каждого, взмахивая умклайдетом (в крайнем случае, щелкая пальцами);
- в) магическая сила заклинания обратно пропорциональна числу палиндромов, на которые разбит набор букв.

Даже домовые знают, что это заклинание можно разбить на отдельные буквы и очень забавно наблюдать, как они, щелкая пальцами и выкрикивая отдельные звуки заклинания, выводят клопов. Однако истинную мощь заклинание обретает только тогда, когда оно разбито на наименьшее возможное число палиндромов. Знание этого разбиения доступно только самым великим магам, например Кристобалю Хозевичу Хунте.

Желая поднять свою квалификацию мага, Александр Привалов решил использовать современные средства для нахождения кратчайшего разбиения заклинания Ауэrsa на палиндромы. Уже неделю он бьется над составлением переборной программы для своего Алдана, но пока все без толку. У него уже начала расти шерсть на ушах, помогите ему.

Входные данные

В единственной строке файла INPUT.TXT содержится непустой исходный набор, состоящий из маленьких букв английского алфавита без пробелов (количество символов в наборе не больше 70).

Выходные данные

В первой строке выходного файла OUTPUT.TXT необходимо указать наименьшее число k палиндромов, на которые разбивается набор. Эти k палиндромов нужно привести в следующих k строках в том порядке, в котором они входят в исходный набор и должны произноситься при использовании заклинания. Если имеется несколько минимальных разбиений, достаточно вывести любое из них.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	baobab	4 b a o bab

2	aaaaa	1 aaaaa
3	rarabar	3 rar aba r

```

/*
    Задача: 848. Заклинание Ауэrsa

    Решение: динамическое программирование, строки,  $O(n^3)$ 

    Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru
*/

```

```

#include <iostream>
#include <string>
#include <vector>
#include <cassert>
#include <algorithm>

bool is_palindrome(std::string s) {
    for (int i = 0, j = (int)s.size()-1; i <= j; ++i,--j) {
        if (s[i] != s[j]) return false;
    }
    return true;
}

const int INF = (int)1e9+1;

int main() {
    std::string s;
    while (std::cin >> s) {
        const int n = (int)s.size();
        std::vector<int> min(1+n, INF), from(1+n, -1);
        min[0] = 0, from[0] = 0;
        for (int len = 1; len <= n; ++len) {
            for (int sub = 1; sub <= len; ++sub) {
                int j = len-1;
                int i = j - sub + 1;
                if (is_palindrome(s.substr(i, sub))) {
                    int val = min[i] + 1;
                    if (min[len] > val) {
                        min[len] = val;
                        from[len] = i;
                    }
                }
            }
        }
        std::vector<std::string> answ;
        for (int len = n; len > 0; ) {
            answ.push_back(s.substr(from[len]));
            s.resize(from[len]);
            len = from[len];
        }
        std::reverse(answ.begin(), answ.end());
        std::cout << answ.size() << "\n";
        for (auto& it : answ) std::cout << it << "\n";
    }
    return 0;
}

```

ЗАДАЧА №862

Домашняя работа

(Время: 1 сек. Память: 16 Мб Сложность: 51%)

Витя наконец-то прошёл последний уровень своей любимой игры "Прострели мне колено". На часах было уже полдвенадцатого, а он до сих пор не сделал домашнюю работу по математике. Прочитав задание, Витя сказал: "О, какое же оно скучное, посчитать сумму прогрессии... Вот посчитать сумму остатков - это весело!". Витя написал в тетрадке

$$\sum_{i=1}^n i \bmod m + \sum_{i=1}^n m \bmod i$$

и... задание было настолько весёлым, что он уснул через 5 минут прямо на столе. Завтра ему с утра нужно опять в школу, и опять он пойдёт с не сделанной домашней работой. Пока он спит, посчитайте для него ответ, на записанную им сумму, чтобы с утра у него было хорошее настроение.

Входные данные

В единственной строке входного файла INPUT.TXT содержатся два числа n и m - числа, записанные Витей. Оба числа натуральные и не превосходят миллиарда.

Выходные данные

В выходной файл OUTPUT.TXT выведете единственное число - ответ на задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	5 5	14
2	10 1	9

```
/*
    Задача: 862. Домашняя работа

    Решение: целочисленная арифметика, арифметическая прогрессия, O(sqrt(m))

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <iostream>

typedef long long ll;

ll sum(ll a, ll b) {
    if (b < a) return ll(0);
    return (a + b) * (b-a+1) / 2;
}

ll part1(ll n, ll m) {
    return n / m * sum(1, m-1) + sum(1, n % m);
}
```

```

ll part2(ll n, ll m) {
    ll res = n * m;
    ll last = n+1;
    for (ll k = 1; k * k <= m; ++k) {
        // [m / i] = k
        // m = i * k + rem
        ll high = m / k;
        ll low = m / (k+1) + 1;
        res -= k * sum(low, std::min(n, high));
        last = std::min(last, low);
    }
    for (ll i = 1; i < last; ++i) {
        res -= m / i * i;
    }
    return res;
}

ll solve(ll n, ll m) {
    return part1(n,m) + part2(n,m);
}

int main() {
    ll n, m;
    while (std::cin >> n >> m) {
        std::cout << solve(n,m) << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №957

Доказательство в HOL

(Время: 1 сек. Память: 16 Мб Сложность: 51%)

Студент Дима учится на втором курсе университета. На втором курсе в его университете читается курс математической логики. В этом курсе особое внимание акцентируется на автоматических доказательствах – программах, позволяющих с их помощью доказывать различные сложные теоремы.

Курсовая работа по математической логике у Димы такая: необходимо доказать с помощью автоматического доказателя HOL, что шахматную доску размером $2^N \times 2^N$, из которой вырезана одна клетка, можно покрыть в один слой уголками из трех клеток.

Дима не верит в то, что это правда, и пытается составить контрпример. Ваша задача доказать Диме, что он неправ, и решить задачу для Диминых входных данных.

Входные данные

Входной файл INPUT.TXT содержит три натуральных числа N , X , Y ($N \leq 6$; $X, Y \leq 2^N$). Этими числами задана доска $2^N \times 2^N$, из которой вырезана клетка с координатами (X, Y) . X – координата по горизонтали, Y – по вертикали, $(1, 1)$ – верхний левый угол доски.

Выходные данные

В выходной файл OUTPUT.TXT выведите 2^N строк по 2^N чисел – номера уголков, покрывающих соответствующие клетки. Каждый уголок характеризуется своим уникальным номером. Уголки пронумерованы начиная с единицы, без пропусков. Вырезанную клетку следует обозначить нулем.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 1 1	0 1 1 1
2	2 2 2	2 2 3 3 2 0 1 3 4 1 1 5 4 4 5 5

```
/*
    Задача: 957. Доказательство в HOL

    Решение: рекурсия, конструктив,  $O(4^n \cdot n)$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <iostream>
#include <vector>
#include <cassert>

const int LU = 0, LD = 1, RU = 2, RD = 3;
```

```

int erow, ecol, n, cnt;
std::vector<std::vector<int>> arr;

void set(int r, int c, int val) {
    assert(arr[r][c] == -1);
    arr[r][c] = val;
}

void fill(int r, int c, int side, int type) {
    assert(side >= 2);
    if (side == 2) {
        ++cnt;
        if (type == LU) {
            set(r,c,cnt);
            set(r+1,c,cnt);
            set(r,c+1,cnt);
        } else if (type == LD) {
            set(r,c,cnt);
            set(r+1,c,cnt);
            set(r+1,c+1,cnt);
        } else if (type == RU) {
            set(r,c,cnt);
            set(r,c+1,cnt);
            set(r+1,c+1,cnt);
        } else if (type == RD) {
            set(r+1,c,cnt);
            set(r,c+1,cnt);
            set(r+1,c+1,cnt);
        }
        return;
    }

    if (type == RD) {
        fill(r, c+side/2, side/2, RU);
        fill(r+side/2, c, side/2, LD);
        fill(r+side/2,c+side/2,side/2, RD);
        fill(r+side/4,c+side/4,side/2, RD);
    } else if (type == RU) {
        fill(r,c,side/2,LU);
        fill(r,c+side/2,side/2,RU);
        fill(r+side/2,c+side/2,side/2,RD);
        fill(r+side/4,c+side/4,side/2,RU);
    } else if (type == LD) {
        fill(r,c,side/2,LU);
        fill(r+side/2,c,side/2,LD);
        fill(r+side/2,c+side/2,side/2,RD);
        fill(r+side/4,c+side/4,side/2,LD);
    } else if (type == LU) {
        fill(r,c+side/2,side/2,RU);
        fill(r+side/2,c,side/2,LD);
        fill(r,c,side/2,LU);
        fill(r+side/4,c+side/4,side/2,LU);
    }
}

void solve(int r, int c, int side) {
    assert(side >= 2);
    if (side == 2) {
        cnt++;
        for (int i = r; i < r + side; ++i) {
            for (int j = c; j < c + side; ++j) {
                if (i == erow && j == ecol) continue;
                set(i,j,cnt);
            }
        }
        return;
    }
    if (erow < r + side / 2 && ecol < c + side / 2) {
        solve(r, c, side / 2);
    }
}

```

```

        fill(r, c+side/2, side/2, RU);
        fill(r+side/2, c, side/2, LD);
        fill(r+side/2, c+side/2, side/2, RD);
        fill(r+side/4, c+side/4, side/2, RD);
    } else if (erow >= r + side / 2 && ecol < c + side / 2) {
        solve(r+side/2, c, side/2);
        fill(r, c, side/2, LU);
        fill(r, c+side/2, side/2, RU);
        fill(r+side/2, c+side/2, side/2, RD);
        fill(r+side/4, c+side/4, side/2, RU);
    } else if (erow < r + side/2 && ecol >= c + side/2) {
        solve(r, c+side/2, side/2);
        fill(r, c, side/2, LU);
        fill(r+side/2, c, side/2, LD);
        fill(r+side/2, c+side/2, side/2, RD);
        fill(r+side/4, c+side/4, side/2, LD);
    } else if (erow >= r+side/2 && ecol >= c + side/2) {
        solve(r+side/2, c+side/2, side/2);
        fill(r, c+side/2, side/2, RU);
        fill(r+side/2, c, side/2, LD);
        fill(r, c, side/2, LU);
        fill(r+side/4, c+side/4, side/2, LU);
    }
}

int main() {
    scanf("%d %d %d", &n, &ecol, &erow);
    ecol--, erow--;
    arr.assign(1 << n, std::vector<int>(1 << n, -1));
    arr[erow][ecol] = 0;
    solve(0, 0, 1 << n);
    for (auto& row : arr) {
        for (auto& it : row) {
            std::cout << it << ' ';
        }
        std::cout << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №964

Ничего не потерялось

(Время: 5 сек. Память: 16 Мб Сложность: 51%)

Разработка новой поисковой системы, которая ведется группой компаний Giggle, проходит под лозунгом «Ничего и не терялось». Вы работаете в отделе передовых разработок, и на сегодняшний день вашей задачей является разработка тестовой версии поискового «движка».

Тестовая версия реализует лишь часть функциональности полной версии. В частности, отсутствуют такие функции, как использование логических выражений в запросах, перевод найденных страниц с одного языка на другой и т.д. Возможности тестовой версии поисковой системы ограничиваются обработкой трех видов запросов: запросов на добавление, на удаление и на поиск. Система работает следующим образом. В любой момент времени существует множество известных системе сайтов, причем для каждого сайта известно множество ключевых слов, встречающихся на нем.

Запрос на добавление содержит ключевое слово и название сайта. При его выполнении ключевое слово добавляется в множество ключевых слов, присутствующих на данном сайте. Если этого слова в соответствующем множестве еще нет, то результатом запроса является «OK», в противном случае – «Already exists».

Запрос на удаление содержит ключевое слово и название сайта. При его выполнении ключевое слово исключается из множества ключевых слов, присутствующих на данном сайте. Если этого слова в соответствующем множестве нет, то результатом запроса является «Not found», в противном случае – «OK».

Запрос на поиск содержит только ключевое слово. Результатом запроса является лексикографически отсортированный список сайтов, содержащих данное ключевое слово. При этом в результат выводятся только первые 10 сайтов из этого списка. Задан список запросов. Необходимо вывести результат их последовательного выполнения.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n – количество запросов, которые необходимо обработать ($0 \leq n \leq 2500$). Каждая из последующих n строк содержит запрос. Запрос на добавление имеет следующий формат:

Add keyword < keyword > to < site >, где < keyword > – ключевое слово, < site > – название сайта, на который добавляется это ключевое слово.

Запрос на удаление имеет следующий формат:

Remove keyword < keyword > from < site >, где < keyword > – ключевое слово, < site > – название сайта, на котором удаляется это ключевое слово.

Запрос на поиск имеет следующий формат:

Search < keyword >, где < keyword > – ключевое слово.

Все ключевые слова (< keyword >) состоят из строчных букв английского алфавита. Длины ключевых слов не превосходят 30 символов.

Все названия сайтов (< site >) состоят из строчных букв английского алфавита, символов «косая черта» («/») и точек («.»). Длины названий сайтов не превосходят 100 символов.

Выходные данные

В выходной файл OUTPUT.TXT выведите результат для каждого запроса. При этом придерживайтесь формата, приведенного в примерах. Не забудьте обратить внимание на второй пример. Результаты запросов разделяйте строкой из пяти символов «равно» («=»).

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	12 Add keyword "olympiads" to neerc.ifmo.ru/school/io Add keyword "neerc" to neerc.ifmo.ru Search "olympiads" Search "neerc" Add keyword "olympiads" to neerc.ifmo.ru Search "olympiads" Add keyword "olympiads" to neerc.ifmo.ru/school/io Remove keyword "olympiads" from neerc.ifmo.ru/school/io Search "olympiads" Remove keyword "olymp" from neerc.ifmo.ru Remove keyword "olympiads" from neerc.ifmo.ru Search "olympiads"	OK ===== OK ===== Results: 1 site(s) found 1) neerc.ifmo.ru/school/io ===== Results: 1 site(s) found 1) neerc.ifmo.ru ===== OK ===== Results: 2 site(s) found 1) neerc.ifmo.ru 2) neerc.ifmo.ru/school/io ===== Already exists ===== OK ===== Results: 1 site(s) found 1) neerc.ifmo.ru ===== Not found ===== OK ===== Results: 0 site(s) found

2		OK =====
		OK =====
		OK =====
		OK =====
		OK =====
		OK =====
	12	OK =====
	Add keyword "keyword" to site01	OK =====
	Add keyword "keyword" to site02	OK =====
	Add keyword "keyword" to site03	OK =====
	Add keyword "keyword" to site04	OK =====
	Add keyword "keyword" to site05	OK =====
	Add keyword "keyword" to site06	OK =====
	Add keyword "keyword" to site07	OK =====
	Add keyword "keyword" to site08	OK =====
	Add keyword "keyword" to site09	OK =====
	Add keyword "keyword" to site10	OK =====
	Add keyword "keyword" to site11	OK =====
	Search "keyword"	Results: 11 site(s) found 1) site01 2) site02 3) site03 4) site04 5) site05 6) site06 7) site07 8) site08 9) site09 10) site10

```

/*
    Задача: 964. Ничего не потерялось

    Решение: std::map, std::set, строки, структуры данных, обработка запросов, O(n * log(n))

    Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru
*/

```

```

#include <bits/stdc++.h>
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    std::map<std::string, std::set<std::string>> map;
    for (int n; std::cin >> n; ) {
        for (int i = 0; i < n; ++i) {
            if (i > 0) { std::cout << "=====\n"; }
            std::string s; std::cin >> s;
            if (s == "Add") {
                std::cin >> s >> s;
                assert(s.front() == '\\' && s.back() == '\\');
                s.erase(s.begin());
                s.erase(std::prev(s.end()));
                auto& curr = map[s];
                std::cin >> s >> s;
                auto it = curr.find(s);
                if (it != curr.end()) { std::cout << "Already exists\n"; }
                else { curr.insert(s); std::cout << "OK\n"; }
            } else if (s == "Remove") {
                std::cin >> s >> s;
            }
        }
    }
}

```

```

        assert(s.front() == '\"' && s.back() == '\"');
        s.erase(s.begin());
        s.erase(std::prev(s.end()));
        auto& curr = map[s];
        std::cin >> s >> s;
        auto it = curr.find(s);
        if (it == curr.end()) { std::cout << "Not found\n"; }
        else { curr.erase(it); std::cout << "OK\n"; }
    } else if (s == "Search") {
        std::cin >> s;
        assert(s.front() == '\"' && s.back() == '\"');
        s.erase(s.begin());
        s.erase(std::prev(s.end()));
        auto& curr = map[s];
        std::cout << "Results: " << curr.size() << " site(s) found\n";
        int cnt = 0;
        for (auto it = curr.begin(); cnt < 10 && it != curr.end(); it++) {
            cnt++;
            std::cout << cnt << " " << *it << "\n";
        }
    }
}

}

return 0;
}

```

ЗАДАЧА №1160

Префикс-функция

(Время: 2 сек. Память: 32 Мб Сложность: 51%)

Дана непустая строка S длиной N символов. Будем считать, что элементы строки нумеруются от 1 до N.

Для каждой i-й позиции строки S определим подстроку, заканчивающуюся в этой позиции, которая совпадает с некоторым началом всей строки S и имеет длину, меньшую, чем i (т.е. не равна i-му префиксу исходной строки). Значением префикс-функции P(i) будем считать длину этой подстроки.

Требуется для всех i от 1 до N вычислить значение P(i).

Входные данные

В единственной строке входного файла INPUT.TXT записана строка, состоящая из символов с кодами ASCII от 33 до 127. Длина строки не превышает 10⁶.

Выходные данные

В выходной файл OUTPUT.TXT выведите все значения префикс-функции.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	ABRACADABRA	0 0 0 1 0 1 0 1 2 3 4
2	ABACABADABACABA	0 0 1 0 1 2 3 0 1 2 3 4 5 6 7

```
#include <stdio.h>
#include <string>
#include <vector>
#include <algorithm>

std::vector<int> prefix(const std::string& s) {
    // Нахождение префикс-функции от строки s за линейное время
    const int n = (int)s.size();
    int len = 0;
    std::vector<int> pref(1+n, 0);
    for (int i = 1; i < n; ++i) {
        while (true) {
            if (s[i] == s[len]) {
                ++len;
                break;
            }
            if (len == 0) {
                break;
            }
            len = pref[len];
        }
        pref[i+1] = len;
    }
    return pref;
}
```

```
int main() {
    static char buf[1+1000000];
    scanf("%1000000s", buf);
    auto pref = prefix(buf);
    for (int i = 1; i < (int)pref.size(); ++i) {
        printf("%d ", pref[i]);
    }
    return 0;
}
```

ЗАДАЧА №1185

RMQ с изменением элемента

(Время: 1 сек. Память: 32 Мб Сложность: 51%)

Требуется реализовать эффективную структуру данных, позволяющую изменять элементы массива $A[1..N]$ и вычислять максимальный элемент на отрезке $[L, R]$.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N – размер массива ($N \leq 10^5$). Во второй строке записаны N целых чисел – элементы массива, целые числа, не превосходящие 10^9 по абсолютной величине. Третья строка содержит натуральное число M – количество запросов ($M \leq 30\,000$). Каждая из следующих M строк представляет собой описание запроса. Сначала вводится одна буква, кодирующая вид запроса («m» – вычислить максимум, «u» – обновить значение элемента). Следом за «m» идут два числа L и R – номера левой и правой границы отрезка. За «u» следуют два числа I и X – номер элемента и его новое значение, не превосходящее 10^9 по абсолютной величине ($1 \leq L \leq R \leq N$, $1 \leq I \leq N$).

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса на вычисление максимума выведите результат. Все числа следует выводить в одну строку, разделяя пробелом.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 1 2 3 4 5 5 m 1 5 u 3 10 m 1 5 u 2 12 m 1 3	5 10 12

```
#include <stdio.h>
#include <vector>
#include <cassert>

struct DataStruct {
    std::vector<int> arr, max, actual;

    const int GSIZE = 256;

    DataStruct(int size = 0, int item = 0) {
        arr.assign(size, item);
        max.assign((size+GSIZE-1) / GSIZE, item);
        actual.assign((size+GSIZE-1) / GSIZE, true);
    }

    void update_group(int g) {
        if (actual[g]) return;
```

```

        actual[g] = true;
        const int begin = g * GSIZE;
        const int after = std::min(begin+GSIZE, (int)arr.size());
        int value = arr[begin];
        for (int i = begin+1; i < after; ++i) {
            value = std::max(value, arr[i]);
        }
        max[g] = value;
    }

void set(int p, int v) {
    const int g = p / GSIZE;
    arr[p] = v;
    actual[g] = false;
}

int get(int l, int r) {
    const int gl = l / GSIZE;
    const int gr = r / GSIZE;
    int answ = arr[l];
    if (gl == gr) {
        for (int i = l; i <= r; ++i) {
            answ = std::max(answ, arr[i]);
        }
    } else {
        for (int i = l, after = (gl+1)*GSIZE; i < after; ++i) {
            answ = std::max(answ, arr[i]);
        }
        for (int g = gl+1; g < gr; ++g) {
            if (!actual[g]) update_group(g);
            answ = std::max(answ, max[g]);
        }
        for (int i = gr * GSIZE; i <= r; ++i) {
            answ = std::max(answ, arr[i]);
        }
    }
    return answ;
}

};

int main() {
    int n; scanf("%d", &n);
    DataStruct ds(n, 0);
    for (int i = 0; i < n; ++i) {
        int value; scanf("%d", &value);
        ds.set(i, value);
    }
    int q; scanf("%d", &q);
    while (q--) {
        char t; int l, r;
        scanf(" %c %d %d", &t, &l, &r);
        if (t == 'm') {
            printf("%d ", ds.get(l-1, r-1));
        } else {
            assert(t == 'u');
            ds.set(l-1, r);
        }
    }
    return 0;
}

```

ЗАДАЧА №1340

Скобки (3)

(Время: 1 сек. Память: 16 Мб Сложность: 51%)

Определим правильные скобочные выражения так:

- 1. Пустое выражение – правильное.
- 2. Если выражение S правильное, то (S) и [S] также правильные.
- 3. Если выражения A и B правильные, то и выражение AB – правильное.

Дана последовательность скобок «(«, «)», «[« и «]». Требуется найти самое короткое правильное выражение, в котором данная последовательность является подпоследовательностью, то есть такое, из которого можно вычеркнуть некоторые символы (возможно, ноль) и получить исходную последовательность, не меняя порядок оставшихся.

Входные данные

Входной файл INPUT.TXT содержит последовательность скобок «(«, «)», «[« и «]», без пробелов. Длина данной последовательности не превышает 100 символов.

Выходные данные

В выходной файл OUTPUT.TXT выведите искомую последовательность скобок без пробелов. Если существует несколько решений, выведите любое.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	[(]	O[O]
2	[[D]]	([[O]])
3	((D))	(([]))[]
4	(([[D]]))	OO[[[OO]]]

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

int main() {
    std::string s; std::cin >> s;
    const int n = (int)s.size();
    if (n == 0) {
        return 0;
    }
    std::vector<std::vector<std::string>> min(n, std::vector<std::string>(n));

    for (int i = 0; i < n; ++i) {
        if (s[i] == '[' || s[i] == ']') {
            min[i][i] = "[]";
        }
        if (s[i] == '(' || s[i] == ')') {
            min[i][i] = "()";
        }
    }
}
```



```

    }
}
for (int i = 0; i+1 < n; ++i) {
    const int j = i+1;
    if (s[i] == '[' && s[j] == ']') {
        min[i][j] = "[]";
    } else if (s[i] == '(' && s[j] == ')') {
        min[i][j] = "()";
    } else {
        min[i][j] = min[i][i]+min[j][j];
    }
}

for (int len = 3; len <= n; ++len) {
    for (int pos = 0; pos+len-1 < n; ++pos) {
        const int i1 = pos, j2 = pos+len-1;
        if ((s[i1] == '[' && s[j2] == ']') || (s[i1] == '(' && s[j2] == ')')) {
            min[i1][j2] = s[i1]+min[i1+1][j2-1] + s[j2];
        } else {
            min[i1][j2] = min[i1][i1]+min[i1+1][j2-1]+min[j2][j2];
        }
        for (int j1 = i1; j1 < j2; ++j1) {
            const int i2 = j1+1;
            if (min[i1][j1].size()+min[i2][j2].size() < min[i1][j2].size()) {
                min[i1][j2] = min[i1][j1]+min[i2][j2];
            }
        }
    }
}
printf("%s", min[0][n-1].c_str());
return 0;
}

```

ЗАДАЧА №1385

Получи дерево

(Время: 1 сек. Память: 16 Мб Сложность: 51%)

Дан связный неориентированный граф без петель и кратных ребер. Разрешается удалять из него ребра. Требуется получить дерево.

Входные данные

Входной файл INPUT.TXT содержит два целых числа: N и M – количество вершин и ребер графа соответственно ($1 \leq N \leq 100$, $0 \leq M \leq N \cdot (N-1)/2$). Далее идет M пар чисел, задающих ребра.

Выходные данные

В выходной файл OUTPUT.TXT выведите N-1 пару чисел – ребра, которые войдут в дерево. Если существует несколько решений, выведите любое. Ребра можно выводить в произвольном порядке.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4 4 1 2 2 3 3 4 4 1	1 2 2 3 3 4

```
/*
    Задача: 1385. Получи дерево

    Решение: графы, деревья, поиск в глубину, O(n+m)

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/
```

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <functional>
#include <cassert>

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    std::vector<std::vector<int>> next(1+n);
    for (int i = 0; i < m; ++i) {
        int u, v; scanf("%d %d", &u, &v);
        next[u].push_back(v);
        next[v].push_back(u);
    }
    std::vector<int> fi, se, parent(1+n, -1);
    std::function<void(int, int)> make_tree = [&](int u, int p) {
        parent[u] = p;
        if (p != 0) {fi.push_back(p); se.push_back(u);}
        for (int v : next[u]) {
```

```
        if (parent[v] == -1) {
            parent[v] = u;
            make_tree(v, u);
        }
    }
};
make_tree(1, 0);
assert((int)fi.size() == n-1);
for (int i = 0; i < n-1; ++i) {
    printf("%d %d\n", fi[i], se[i]);
}
return 0;
}
```

ЗАДАЧА №1438

Зубры

(Время: 2 сек. Память: 16 Мб Сложность: 51%)

Это задача про зубров и любые совпадения с бизонами считайте случайными. Страна зубров состоит из N городов, которые соединены M двусторонними дорогами (между двумя городами есть только одна дорога). Известно, что дороги были построены так, что из любого города по дорогам можно добраться до любого другого города.

Зубры Виталия и Рома постоянно играют в одну и ту же игру: Рома загадывает число L , а Виталия должен проверить можно ли из какого-нибудь города добраться до K других городов, если проходить только по дорогам, длина которых не меньше L .

Виталия очень устал, да и скоро ему встречаться с Поликарпом, поэтому он просит вас помочь решить эту задачу раз и навсегда: найдите максимально возможное число L , такое что, существует город из которого можно добраться до K других городов проходя по дорогам, длина которых не меньше L .

Входные данные

Первая строка входного файла INPUT.TXT содержит три целых числа: N ($2 \leq N \leq 10^5$), M ($N-1 \leq M \leq \min(N*(N-1)/2, 10^5)$) и K ($1 \leq K < N$) – общее число городов, число дорог и число городов, до которых нужно добраться.

Следующие M строк содержат по три натуральных числа: a_i, b_i ($1 \leq a_i, b_i \leq N$) и l_i ($1 \leq l_i \leq 10^9$) – первые два числа задают номера городов, которые соединяет дорога, а третье число – длину дороги.

Гарантируется, что любые два города соединяются не более чем одной дорогой, а так же, что нет дорог, которые соединяют город сам с собой.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное целое число – ответ на задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 3 2 1 2 2 3 2 3 3 1 1	2
2	4 6 3 1 3 9 3 2 3 1 2 2 4 3 9 2 4 8 1 4 3	8

```
#include <stdio.h>
```

```

#include <algorithm>
#include <vector>

struct DSU {
    // Система непересекающихся множеств: size - размер множества, parent - представитель м
    std::vector<int> size, parent;
    // Конструктор для множества фиксированного размера:
    DSU(const int size) : size(size, 0), parent(size, -1) { }
    // Добавление новой вершины в множество:
    void make_set(const int v) {
        parent[v] = v;
        size[v] = 1;
    }
    // Нахождение представителя множества, к которому относится вершина v:
    int find_set(const int v) {
        return v == parent[v] ? v : parent[v] = find_set(parent[v]);
    }
    // Нахождение размера множества для вершины v:
    int get_size(const int v) {
        return size.at(find_set(v));
    }
    // Объединение множеств для вершин a и b:
    void union_sets(int a, int b) {
        a = find_set(a);
        b = find_set(b);
        if (a != b) {
            if (size[a] < size[b]) {
                std::swap(a, b);
            }
            parent[b] = a;
            size[a] += size[b];
        }
    }
};

struct Edge {
    int a, b, cost;
};

int main() {
    int nVert, nEdges, size;
    scanf("%d %d %d", &nVert, &nEdges, &size);

    std::vector<Edge> edges;
    for (int i = 0; i < nEdges; ++i) {
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        --a, --b;
        edges.push_back(Edge{a,b,c});
        edges.push_back(Edge{b,a,c});
    }
    // Отсортируем ребра в порядке невозрастания их длин:
    std::stable_sort(edges.begin(), edges.end(), [](const Edge& l, const Edge& r){
        return l.cost > r.cost;
    });
    // Будем поддерживать размер текущих компонент связностей в системе непересекающихся мн
    DSU dsu(nVert);
    // Изначально занесем каждую вершину в dsu:
    for (int v = 0; v < nVert; ++v) {
        dsu.make_set(v);
    }
    // Будем добавлять текущее ребро, жадно объединяя множества для вершин, которые оно сое
    for (auto& e : edges) {
        dsu.union_sets(e.a, e.b);
        if (dsu.get_size(e.a) > size) {
            printf("%d\n", e.cost);
            return 0;
        }
    }
}

```

```
    throw 1;  
}
```

ЗАДАЧА №621

Ближайшее число

(Время: 0,5 сек. Память: 16 Мб Сложность: 52%)

Дана матрица A размером $N \times N$, заполненная неотрицательными целыми числами. Расстояние между двумя элементами A_{ij} и A_{pq} определено как $|i - p| + |j - q|$. Требуется заменить каждый нулевой элемент матрицы ближайшим ненулевым. Если есть две или больше ближайших ненулевых ячейки, нуль должен быть оставлен.

Входные данные

В первой строке входного файла INPUT.TXT содержится натуральное число N ($N \leq 200$). Затем идут N строк по N чисел, разделённых пробелами и представляющих собой матрицу. Элементы матрицы не превосходят значения 10^6 . Входные данные могут содержать несколько пробелов подряд, пробелы до первого числа строки и после последнего.

Выходные данные

В выходной файл OUTPUT.TXT выведите N строк по N чисел, разделённых пробелами, - модифицированную матрицу.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3	1 0 2
	0 0 0	1 0 2
	1 0 2	0 3 0
	0 3 0	

```
#pragma GCC diagnostic ignored "-Wunused-result"

#include <stdio.h>
#include <bits/stdc++.h>

char getChar() {
    static char buf[1024*1024];
    static int pos = 0;
    static int size = 0;
    if (size == pos) {
        size = fread(buf, 1, 1024*1024, stdin);
        pos = 0;
    }
    if (size == pos) {
        return EOF;
    }
    return buf[pos++];
}

int getInt() {
    int sign = 1;
    char c = '?';
    while (!(c == '-' || ('0' <= c && c <= '9'))) {
        c = getChar();
    }
```

```

    }
    if (c == '-') {
        sign = -1;
        c = getChar();
    }
    int answ = 0;
    while ('0' <= c && c <= '9') {
        (answ *= 10) += (c - '0');
        c = getChar();
    }
    return answ * sign;
}

void putChar(char c) {
    static char buf[1024*1024];
    static int size = 0;
    if (c == -1 || size == 1024*1024) {
        fwrite(buf, 1, size, stdout);
        size = 0;
    }
    if (c != -1) buf[size++] = c;
}

void putInt(int num) {
    char buf[10]; sprintf(buf, "%d", num);
    for (int i = 0; buf[i] != '\0'; ++i)
        putChar(buf[i]);
}

int main() {
    int n = getInt();
    std::vector<std::vector<int>> field(n, std::vector<int>(n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            field[i][j] = getInt();
        }
    }

    // Для каждой позиции [i][j] посчитаем ближайшие слева и справа ненулевые элементы:
    std::vector<std::vector<int>> left(n, std::vector<int>(n, -1));
    auto right = left;
    for (int i = 0; i < n; ++i) {
        for (int j = 1; j < n; ++j) {
            left[i][j] = field[i][j-1] == 0 ? left[i][j-1] : j-1;
        }
        if (field[i].back() != 0) {
            right[i].back() = n-1;
        }
        for (int j = n-2; j >= 0; --j) {
            right[i][j] = field[i][j] == 0 ? right[i][j+1] : j;
        }
    }

    for (int row = 0; row < n; ++row) {
        for (int col = 0; col < n; ++col) {
            if (field[row][col] != 0) continue;
            int minRow = -1, minCol = -1, minCount = 0, minDist = (int)1e9;
            // Пройдем по строчкам. На интересуют ближайшие слева и справа ненулевые элеме
            for (int r = 0; r < n; ++r) {
                int prev = left[r][col];
                if (prev != -1) {
                    int dist = std::abs(row - r) + std::abs(col - prev);
                    if (dist < minDist) {
                        minDist = dist, minRow = r, minCol = prev, minCount = 1;
                    } else if (dist == minDist) {
                        assert(minCount > 0); minCount++;
                    }
                }
            }
        }
    }
}

```



```

    }
    int curr = right[r][col];
    if (curr != -1) {
        int dist = std::abs(row - r) + std::abs(col - curr);
        if (dist < minDist) {
            minDist = dist, minRow = r, minCol = curr, minCount = 1;
        } else if (dist == minDist) {
            assert(minCount > 0); minCount++;
        }
    }
}
if (minCount == 1) {
    field[row][col] = field[minRow][minCol];
}
}
}
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        putInt(field[i][j]);
        putChar(' ');
    }
    putChar('\n');
}
putChar(-1);

return 0;
}

```

ЗАДАЧА №626

Преобразователь строк

(Время: 1 сек. Память: 16 Мб Сложность: 52%)

Преобразователь строк - это специальная компьютерная программа. Она получает на вход строку S и набор правил преобразования строки. Каждое из правил имеет вид $s_1 s_2 \rightarrow E$, где s_1 и s_2 - маленькие буквы английского алфавита. Символом E здесь обозначена пустая строка. При этом каждый символ присутствует не более, чем в одном правиле.

Преобразователь строк работает по шагам. За один шаг он находит в текущей строке подстроку, совпадающую с правой частью одного из правил, после чего эта подстрока удаляется из текущей строки (этот процесс называется применением правила). Этот процесс продолжается до тех пор, пока существует правило, которое можно применить. Строка, которая остается к моменту, когда нельзя применить никакое правило, считается результатом T работы преобразователя строк.

Пусть, например, набор правил таков: $\{ab \rightarrow E, cd \rightarrow E\}$, а исходная строка $S = aabbccdba$. Тогда работа преобразователя будет выглядеть так: $aabbccdba \rightarrow abccdba \rightarrow ccdba \rightarrow cba$, и результатом T работы преобразователя будет строка cba.

Ваша задача состоит в том, чтобы написать программу, моделирующую работу преобразователя строк с заданным набором правил на заданной строке S.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n ($0 \leq n \leq 13$) - количество правил. Далее идут n строк, каждая из которых описывает одно из правил. Гарантируется, что каждая из маленьких букв английского алфавита присутствует не более, чем в одном правиле.

Последняя, (n+2)-ая строка входного файла содержит исходную строку S. Она не пуста и состоит только из маленьких букв английского алфавита. Длина S не превосходит 100000.

Выходные данные

В выходной файл OUTPUT.TXT выведите строку T - результат работы преобразователя на строке S.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 ab cd aabbccdba	cba
2	0 abcdefghijklmnopqrstuvwxyz	abcdefghijklmnopqrstuvwxyz

/*
Задача: 626. Преобразователь строк
Решение: реализация, строки, stack, O(len)
Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

```

*/

#include <stdio.h>
#include <string>
#include <vector>

int main() {
    int n; scanf("%d", &n);
    static bool erase[26][26];
    for (int i = 0; i < n; ++i) {
        char a, b; scanf(" %c %c", &a, &b);
        int f = a - 'a', s = b - 'a';
        erase[f][s] = true;
    }
    std::vector<char> stack;
    char buf[1000000+1];
    scanf("%1000000s", buf);
    std::string s(buf);
    for (auto it : s) {
        if (stack.empty()) {
            stack.push_back(it);
            continue;
        }
        int i = stack.back() - 'a';
        int j = it - 'a';
        if (erase[i][j]) {
            stack.pop_back();
        } else {
            stack.push_back(it);
        }
    }
    s = std::string(stack.begin(), stack.end());
    printf("%s", s.c_str());
    return 0;
}

```

ЗАДАЧА №634

Кубок CBOSS

(Время: 1 сек. Память: 16 Мб Сложность: 52%)

В 2239 году команде-победителю Открытого кубка CBOSS достался весьма нетрадиционный приз - поездка по k самым красивым городам России. Так как в России красивых городов достаточно много, то победителям было предложено выбрать k городов из списка, содержащего n городов.

Для удобства занумеруем эти города целыми числами от 1 до n . Для каждого города известно t_i - время, требующееся на осмотр его достопримечательностей. Также для каждой пары (i, j) , $1 \leq i, j \leq n$ известно a_{ij} - время, которое требуется на проезд из i -ого города в j -ый. При этом может оказаться, что $a_{ij} \neq a_{ji}$, но a_{ii} всегда равно нулю.

У студентов, входящих в команду-победитель, не так много времени на посещение красивых городов, ведь скоро у них сессия. Поэтому они хотят выбрать k городов и посетить их в таком порядке, чтобы затраты времени были минимальны. Разумеется, посещать один город несколько раз им неинтересно. Также они не хотят приезжать в город, не осматривая при этом его достопримечательности.

Напишите программу, находящую нужные k городов и порядок, в котором их нужно посетить.

Входные данные

Первая строка входного файла INPUT.TXT содержит целые числа n и k ($1 \leq n \leq 7$, $1 \leq k \leq n$). Каждая из последующих n строк входного файла содержит по n целых чисел каждая: j -ое число $(i + 1)$ -ой строки входного файла - это время, требуемое на проезд из i -ого города в j -ый (a_{ij}). Последняя строка входного файла содержит n целых чисел t_1, \dots, t_n .

Все числа во входном файле не превосходят 100. Все времена неотрицательны.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите минимальное время, которое потребуется для посещения k городов с учетом осмотра достопримечательностей. Во второй строке выходного файла выведите номера городов в порядке посещения, гарантирующем такое время.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 3 0 3 2 1 8 0 6 5 1 2 0 4 5 6 7 0 1 2 3 4	10 3 1 4

2	4 4	18 3 1 4 2
	0 3 2 1	
	8 0 6 5	
	1 2 0 4	
	5 6 7 0	
	1 2 3 4	

```

/*
    Задача: 634. Кубок CBOSS

    Решение: перестановки, перебор,  $O(n! \cdot k)$ 

    Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru
*/

```

```

#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (int n, k; std::cin >> n >> k; ) {
        vvi a(n, vi(n, 0));
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                std::cin >> a[i][j];
            }
        }
        vi t(n);
        for (auto &it : t) { std::cin >> it; }
        vi perm(n), answ;
        for (int i = 0; i < n; ++i) { perm[i] = i; }
        int best = (int)1e9+7;
        do {
            int curr = 0;
            for (int i = 0; i < k; ++i) {
                int v = perm[i];
                if (i > 0) {
                    int u = perm[i-1];
                    curr += a[u][v];
                }
                curr += t[v];
            }
            if (curr < best) {
                best = curr;
                answ.assign(perm.begin(), perm.begin()+k);
            }
        } while (std::next_permutation(all(perm)));
        std::cout << best << '\n';
        for (auto &it : answ) { std::cout << it+1 << ' '; }
        std::cout << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №673

N-значные числа - 2

(Время: 1 сек. Память: 16 Мб Сложность: 52%)

Найти количество N-значных чисел, у которых сумма цифр равна их произведению. Вывести наименьшее среди таких чисел для заданного N.

Входные данные

В единственной строке входного файла INPUT.TXT записано одно натуральное число N, которое не превышает 20.

Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести через пробел 2 числа: количество искомых чисел и наименьшее среди них.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1	10 0
2	2	1 22

```
#include <iostream>
#include <string>
#include <cassert>

int main() {
    int n; std::cin >> n;
    if (n == 1) {
        std::cout << "10 0";
        return 0;
    }
    static int count[21][201][201]; // [len][sum][prod]
    for (int digit = 1; digit <= 9; ++digit) {
        count[1][digit][digit] = 1;
    }
    for (int len = 1; len < n; ++len) {
        for (int sum = 0; sum < 201; ++sum) {
            for (int prod = 0; prod < 201; ++prod) {
                for (int digit = 1; digit <= 9; ++digit) {
                    if (sum+digit < 201 && prod * digit < 201) {
                        count[len+1][sum+digit][prod * digit] += count[len][sum][prod];
                    }
                }
            }
        }
    }
    int answ = 0;
    for (int sum = 0; sum < 201; ++sum) {
        answ += count[n][sum][sum];
    }

    std::string number(n, '1');
    int sum = n, prod = 1;
```

```
while (prod != sum) {
    int pos = n-1;
    while (pos >= 0 && number[pos] == '9') {
        number[pos] = '1';
        sum -= 8;
        prod /= 9;
        --pos;
    }
    assert(pos >= 0);
    sum++;
    prod = prod / (number[pos] - '0') * (number[pos] - '0' + 1);
    number[pos]++;
}
std::cout << answ << ' ' << number;
return 0;
}
```

ЗАДАЧА №709

Булева алгебра

(Время: 1 сек. Память: 16 Мб Сложность: 52%)

В каждом языке программирования, даже самом простом, есть оператор ветвления, позволяющий проверить истинность логического выражения и, в зависимости от его результата, выполнить то или иное действие. Условие оператора ветвления представляет собой логическое (булевское) выражение, результатом которого может быть либо истина (TRUE), либо ложь (FALSE). Переменные, которые могут участвовать в логическом выражении, называются булевыми (boolean). Булевские переменные могут объединяться в сложные условия при помощи логических операций (функций):

- $AND(x_1, x_2, \dots, x_s)$. Операция «И», возвращает истинное значение, если все ее операнды истинны. ($2 \leq \text{количество операндов} \leq s$);
- $OR(x_1, x_2, \dots, x_s)$. Операция «ИЛИ», возвращает истинное значение, если хотя бы один ее операнд истинен. ($2 \leq \text{количество операндов} \leq s$);
- $NOT(x_1)$. Операция «НЕ», меняет значение операнда x_1 на противоположное (операнд всегда один).

В логическом условии может использоваться несколько логических функций, вложенных друг в друга, то есть результат одной функции может использоваться другой в качестве операнда. Например, $AND(A, B, OR(C, D))$. Данное выражение будет истинно тогда, когда истинны A, B и (C или D).

Требуется написать программу, которая по имеющемуся логическому выражению и значению логических переменных определит результат выражения. Количество операндов у функций AND и OR всегда равно двум ($s=2$).

Входные данные

Первая строка входного файла INPUT.TXT содержит логическое выражение (длина не более 255 символов). Вторая строка содержит два числа, разделенных одним или несколькими пробелами: N – количество блоков (не более 10), K – количество переменных (не более 26). Далее следует N блоков, каждый имеет следующую структуру: состоит из K строк, каждая содержит выражение типа <переменная = значение>. Переменные задаются заглавными английскими буквами, значение – константами TRUE или FALSE (заглавные буквы).

Выходные данные

В выходной файл OUTPUT.TXT выведите N строк со значением результата логического выражения для переменных соответствующего блока.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	AND(A,NOT(B)) 1 2 A=FALSE B=TRUE	FALSE

2	OR(NOT(AND(A,B)),A)	
	3 2	
	A=FALSE	
	B=TRUE	TRUE
	A=TRUE	TRUE
	B=TRUE	TRUE
	A=FALSE	
	B=FALSE	

```

/*
    Задача: 709. Булева алгебра

    Решение: рекурсия, разбор выражений, РВНФ,  $O(q * \text{len}(s))$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

```

```

#include <bits/stdc++.h>
#define isz(s) (int)(s).size()

std::string remove_spaces(std::string s) {
    std::string ret;
    for (auto it : s) {
        if (std::isspace(it)) { continue; }
        ret.push_back(it);
    }
    return ret;
}

namespace Solution {

    std::string s;
    int pos = 0;
    std::vector<bool> var(256);

    void init(std::string s_) {
        s = s_, pos = 0;
        var.assign(256, false);
    }

    void set(std::string s_) {
        auto p = s_.find('=');
        assert(p != std::string::npos);
        auto v = s_.substr(0,p);
        s_ = s_.substr(p+1);
        assert(v.size() == 1u);
        assert(s_ == "TRUE" || s_ == "FALSE");
        var[v[0]] = (s_ == "TRUE");
    }

    bool calc() {
        assert('A' <= s[pos] && s[pos] <= 'Z');
        std::string op;
        while (pos < isz(s) && 'A' <= s[pos] && s[pos] <= 'Z') {
            op.push_back(s[pos++]);
        }
        if (op == "NOT") {
            assert(s[pos++]=='(');
            bool res = calc();
            assert(s[pos++]==')');
            return !res;
        }
        if (op == "AND" || op == "OR") {
            assert(s[pos++]=='(');
            bool fi = calc();
            assert(s[pos++]==',');
            bool se = calc();

```

```

        assert(s[pos++]==' ');
        return op == "AND" ? fi && se : fi || se;
    }
    assert(isz(op) == 1);
    return var[op[0]];
}

std::string solve() {
    pos = 0;
    bool res = calc();
    return res ? "TRUE" : "FALSE";
}

}

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    std::string s;
    std::getline(std::cin, s);
    Solution::init(remove_spaces(s));
    int q, n; std::cin >> q >> n;
    std::getline(std::cin, s);
    s = remove_spaces(s);
    assert(s.empty());
    while (q--) {
        for (int i = 0; i < n; ++i) {
            std::getline(std::cin, s);
            Solution::set(remove_spaces(s));
        }
        std::cout << Solution::solve() << "\n";
    }
    return 0;
}

```

ЗАДАЧА №753

Различные слова

(Время: 3 сек. Память: 64 Мб Сложность: 52%)

Дана строка S , состоящая из N символов. Назовем ее подстрокой S_{ij} строку с i -го по j -й символ ($i \leq j$).
Ваша задача — посчитать количество различных подстрок заданной строки.

Входные данные

Входной файл INPUT.TXT содержит одну непустую строку S , состоящую из маленьких английских букв, длиной не более чем 1024 символа.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число — количество различных подстрок строки S .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	abc	6
2	aaa	3

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <string>
#include <cassert>
#include <iostream>

typedef unsigned long long ull;

std::vector<ull> prepare(std::string& s, const std::vector<ull>& pow) {
    std::vector<ull> pref{0};
    for (int i = 0; i < (int)s.size(); ++i) {
        s[i] = s[i] - 'a' + 1;
        assert(0 < s[i] && s[i] < pow[1]);
        pref.push_back(pref.back() + s[i] * pow[i]);
    }
    return pref;
}

int main() {
    const ull mod = 71;
    std::vector<ull> pow{1};
    while ((int)pow.size() <= 20000) {
        pow.push_back(pow.back() * mod);
    }
    const int mxPow = (int)pow.size()-1;

    char buf[1025];
    scanf("%1024s", buf);
    std::string s(buf);

    auto pref = prepare(s, pow);
```

```
std::vector<ull> hashes;
for (int i = 0; i < (int)s.size(); ++i) {
    for (int j = i; j < (int)s.size(); ++j) {
        auto hash = (pref[j+1] - pref[i]) * pow[mxPow-j];
        hashes.push_back(hash);
    }
}

std::sort(hashes.begin(), hashes.end());
hashes.erase(std::unique(hashes.begin(), hashes.end()), hashes.end());

printf("%d", (int)hashes.size());
return 0;
}
```

ЗАДАЧА №992

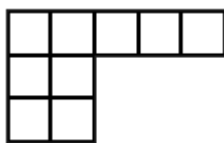
Неприводимые диаграммы Юнга

(Время: 2 сек. Память: 16 Мб Сложность: 52%)

Диаграммы Юнга используются для того, чтобы изобразить разбиение числа на слагаемые. Разбиение числа n на слагаемые представляет собой сумму вида $n = m_1 + m_2 + \dots + m_k$, где $m_1 \geq m_2 \geq \dots \geq m_k$.

Диаграмма состоит из n квадратиков, организованных в виде k рядов, где k — количество слагаемых в разбиении. Ряд, соответствующий числу m_i , содержит m_i квадратиков. Все ряды выровнены по левому краю и упорядочены от более длинного к более короткому.

Например, диаграмма Юнга, приведенная на рисунке, соответствует разбиению $9 = 5 + 2 + 2$.

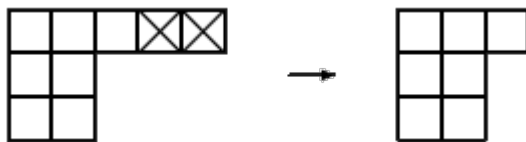


Рассмотрим один метод преобразования диаграмм Юнга. Разрешается выбрать любые два соседних квадратика и удалить их. При этом накладывается следующее ограничение: после преобразования оставшийся набор квадратиков должен быть корректной диаграммой Юнга, верхний левый квадратик которой расположен в том же месте, что и у исходной диаграммы (либо должна остаться пустая диаграмма).

Например, удалив последние квадратик второго и третьего ряда из приведенной выше диаграммы, мы получаем диаграмму для разбиения $7 = 5 + 1 + 1$.



Удаляя два последних квадратика первого ряда из этой исходной диаграммы, мы получаем диаграмму для разбиения $7 = 3 + 2 + 2$.



И еще один способ преобразовать эту диаграмму — удалить последний ряд целиком. Диаграмма, которую нельзя преобразовать указанным способом, называется неприводимой. В частности, пустая диаграмма Юнга является неприводимой.

Каждую диаграмму можно преобразовывать до тех пор, пока она не станет неприводимой. Вообще говоря, может быть несколько способов преобразовать диаграмму к неприводимой. По заданной диаграмме Юнга найдите все неприводимые диаграммы, в которые ее можно преобразовать.

Входные данные

Первая строка входного файла INPUT.TXT содержит число k — количество рядов в диаграмме ($1 \leq k \leq$

10^5). Вторая строка содержит k целых чисел: m_1, m_2, \dots, m_k . Сумма $n = m_1 + m_2 + \dots + m_k$ не превышает 10^8 .

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите число p – количество неприводимых диаграмм Юнга, к которым можно преобразовать заданную. Следующие p строк должны описывать эти диаграммы. Каждая строка должна начинаться с числа t – количества рядов в соответствующей диаграмме.

Далее должно следовать t чисел – количество квадратиков в соответствующих рядах.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 5 2 2	1 1 1
2	1 2	1 0

```
/*
    Задача: 992. Неприводимые диаграммы Юнга

    Решение: математическое моделирование, O(sum)

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
#define isz(x) (int)(x).size()
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (int n; std::cin >> n; ) {
        std::vector<int> arr(n+2);
        for (int i = 0; i < n; ++i) {
            std::cin >> arr[i];
        }
        for (int i = n-1; i >= 0; --i) {
            int delta = arr[i] - arr[i+1];
            arr[i] -= delta / 2 * 2;
            if (arr[i] == arr[i+1]) {
                delta = arr[i] - arr[i+2];
                arr[i] -= delta;
                arr[i+1] -= delta;
            }
        }
        while (!arr.empty() && arr.back() == 0) { arr.pop_back(); }
        if (arr.empty()) {
            std::cout << "1\n0\n";
            continue;
        }
        const int max = *std::max_element(all(arr));
        std::vector<int> cnt(max+1);
        for (auto it : arr) { cnt[it]++; }
        arr.clear();
        bool was = true;
        while (was) {
            was = false;
            for (int i = isz(cnt)-1; i >= 1; --i) {
```

```

        if (cnt[i] >= 2) {
            was = true;
            cnt[i-1] += cnt[i] / 2 * 2;
            cnt[i] %= 2;
            for (int j = i; j+1 < isz(cnt) && cnt[j] == 0; ++j) {
                cnt[j-1] += cnt[j+1];
                cnt[j+1] = 0;
            }
        }
    }
    if (isz(cnt) > 0) { cnt[0] = 0; }
    while (!cnt.empty() && cnt.back() == 0) {
        cnt.pop_back();
    }
}
std::cout << 1 << "\n";
for (int i = isz(cnt)-1; i >= 1; --i) {
    assert(cnt[i] == 1);
    arr.push_back(i);
}
std::cout << arr.size();
for (auto it : arr) { std::cout << ' ' << it; }
std::cout << '\n';
}
return 0;
}

```

ЗАДАЧА №1346

Упаковка символов

(Время: 1 сек. Память: 16 Мб Сложность: 52%)

Билл пытается компактно представить последовательности прописных символов от A до Z с помощью упаковки повторяющихся подпоследовательностей внутри них. Например, один из способов представить последовательность AAAAAAAAAABABABCCD – это 10(A)2(BA)B2(C)D. Он формально определяет сжатые последовательности символов и правила перевода их в несжатый вид следующим образом:

- Последовательность, содержащая один символ от A до Z, является упакованной. Распаковка этой последовательности даёт ту же последовательность из одного символа.
- Если S и Q – упакованные последовательности, то SQ – также упакованная последовательность. Если S распаковывается в S', а Q распаковывается в Q', то SQ распаковывается в S'Q'.
- Если S – упакованная последовательность, то X(S) – также упакованная последовательность, где X – десятичное представление целого числа, большего 1. Если S распаковывается в S', то X(S) распаковывается в S', повторённую X раз.

Следуя этим правилам, легко распаковать любую заданную упакованную последовательность. Однако Биллу более интересен обратный переход. Он хочет упаковать заданную последовательность так, чтобы результирующая сжатая последовательность содержала наименьшее возможное число символов.

Входные данные

Входной файл INPUT.TXT содержит последовательность символов от A до Z. Длина последовательности от 1 до 100.

Выходные данные

В выходной файл OUTPUT.TXT выведите упакованную последовательность наименьшей длины, которая распаковывается в заданную последовательность. Если таких последовательностей несколько, выведите любую.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	AAAAAAAAAABABABCCD	9(A)3(AB)CCD
2	NEERCYESYESYESNEERCYESYESYES	2(NEERC3(YES))

```
#include <string>
#include <iostream>
#include <vector>
#include <algorithm>
#include <cassert>

int main() {
    std::string s; std::cin >> s;
    const int n = (int)s.size();
```



```

if (n <= 4) {
    std::cout << s;
    return 0;
}

std::vector<std::vector<std::string>> min(n, std::vector<std::string>(n));
for (int len = 1; len <= 4; ++len) {
    for (int i = 0; i+len-1 < n; ++i) {
        min[i][i+len-1] = s.substr(i, len);
    }
}

for (int len = 5; len <= n; ++len) {
    for (int l1 = 0; l1+len-1 < n; ++l1) {
        const int r2 = l1+len-1;
        min[l1][r2] = s.substr(l1, len);
        for (int r1 = l1; r1 < r2; ++r1) {
            auto temp = s.substr(l1, r1-l1+1);
            const int len2 = r1-l1+1;
            if (len % len2 == 0) {
                bool flag = true;
                for (int i = l1; i+len2-1 <= r2; i += len2) {
                    if (s.substr(i, len2) != temp) {
                        flag = false;
                        break;
                    }
                }
                if (flag) {
                    temp = std::to_string(len / len2) + "(" + min[l1][r1] + ")";
                    if (temp.size() < min[l1][r2].size()) {
                        min[l1][r2] = temp;
                    }
                }
            }
        }
        for (int r1 = l1; r1 < r2; ++r1) {
            if (min[l1][r1].size() + min[r1+1][r2].size() < min[l1][r2].size()) {
                min[l1][r2] = min[l1][r1] + min[r1+1][r2];
            }
        }
    }
}

printf("%s", min[0][n-1].c_str());
return 0;
}

```

Пакетная обработка процессов

(Время: 1 сек. Память: 16 Мб Сложность: 53%)

Для ускорения прохождения «коротких» заданий на ЭВМ выбран пакетный режим работы с квантованием времени процессора. Это значит, что всем заданиям пакета по очереди представляется процессор на одинаковое время 10 с (круговой циклический алгоритм разделения времени). Если в течение этого времени заканчивается выполнение задания, оно покидает систему и освобождает процессор. Если же очередного кванта времени не хватает для завершения задания, оно помещается в конец очереди — пакета. Последнее задание пакета выполняется без прерываний. Пакет считается готовым к вводу в ЭВМ, если в нем содержится К заданий. Новый пакет вводится в ЭВМ после окончания обработки предыдущего. Задания поступают в систему с интервалом времени 60 ± 30 с и характеризуются временем работы процессора 50 ± 45 с.

Требуется смоделировать процесс обработки N заданий и определить время начала и окончания каждого процесса.

Входные данные

На первой строке входного файла INPUT.TXT находятся числа N и K - число процессов и количество процессов в пакете ($1 \leq N \leq 1000, 1 \leq K \leq 100$). Гарантируется, что N делится на K. Далее следуют N строк с информацией о времени формирования и необходимое время на выполнение для каждого процесса. Все процессы стартуют в один день и следуют в порядке возрастания времени ввода их в систему.

Выходные данные

Выведите в выходной файл OUTPUT.TXT для каждого процесса в отдельной строке время его старта и время окончания через пробел в формате ЧЧ:ММ:СС.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	6 2	00:02:14 00:04:17
	00:01:02 63	00:02:24 00:04:33
	00:02:14 76	00:04:33 00:04:57
	00:03:16 14	00:04:43 00:05:06
	00:04:02 19	00:05:45 00:06:12
	00:04:36 17	00:05:55 00:07:17
	00:05:45 75	

```
#pragma GCC diagnostic ignored "-Wunused-result"

#include <stdio.h>
#include <bits/stdc++.h>

int get_time() {
    int h, m, s; scanf("%d:%d:%d", &h, &m, &s);
    return s + 60 * (m + 60 * h);
}
```

```

void put_time(int s) {
    int h = s / 3600 % 24;
    int m = s / 60 % 60;
    s %= 60;
    printf("%02d:%02d:%02d", h, m, s);
}

struct Record {
    int id, time, dur;
};

int main() {
    int n, qSize; scanf("%d %d", &n, &qSize);
    std::deque<Record> deque;
    std::vector<std::pair<int, int>> answ(n, {-1, -1});
    int global_time = 0;
    for (int i = 0; i < n; ++i) {
        int time = std::max(get_time(), global_time), dur; scanf("%d", &dur);
        deque.push_back(Record{i, time, dur});
        if ((int)deque.size() == qSize) {
            while (!deque.empty()) {
                auto curr = deque.front(); deque.pop_front();
                if (answ[curr.id].first == -1) {
                    answ[curr.id].first = time;
                }
                int dec = std::min(10, curr.dur);
                curr.dur -= dec;
                if (curr.dur == 0) {
                    answ[curr.id].second = time + dec;
                } else {
                    deque.push_back(curr);
                }
                time += dec;
            }
            global_time = time;
        }
    }
    for (auto& it : answ) {
        put_time(it.first); printf(" "); put_time(it.second); printf("\n");
    }
    return 0;
}

```

ЗАДАЧА №483

Двоичная машина

(Время: 1 сек. Память: 16 Мб Сложность: 53%)

На вход некоторой двоичной машине подается n -разрядное двоичное число. Машина подвергает поданное число следующим преобразованиям:

- 1. Записывает исходное число в регистры А и В размером N двоичных разрядов (бит).
- 2. $N-1$ раз выполняет следующие действия: циклически сдвигает регистр В на один бит влево (при этом самый старший бит переходит в самый младший); прибавляет к значению регистра А значение регистра В по модулю 2^N .
- 3. К регистру А применяется побитовая операция логического отрицания
- 4. К регистру А прибавляется 1 по модулю 2^N .
- 5. Значение регистра А выдается в качестве результата с отбрасыванием ведущих (незначащих) нулей.

Например, если на вход машине подать число 1001, то будут выполнены следующие преобразования:

Напишите программу, которая по заданному числу возвращает результат преобразования согласно описанному алгоритму работы двоичной машины.

Входные данные

Первая строка входного файла INPUT.TXT содержит число N – количество двоичных разрядов ($N \leq 100\,000$). Вторая строка содержит ровно N двоичных разрядов (нулей или единиц) – входное число.

Выходные данные

В выходной файл OUTPUT.TXT выведите ответ – результат преобразования в двоичном виде без ведущих нулей.

Примеры

№ шага	A	B	Result
1.	1001	1001	
2. (повтор. 1)	1100	0011	
2. (повтор. 2)	0010	0110	
2. (повтор. 3)	1110	1100	
3.	0001	1100	
4.	0010	1100	
5.	0010	1100	10

№	INPUT.TXT	OUTPUT.TXT
1	4 1001	10
2	7 1101101	101

```

/*
    Главная проблема - второй шаг. Если делать явно, то асимптотика  $O(n^2)$ 
    Проанализируем сдвиги на примере 1001:
    1001
    0011
    0110
    1100
    ----
    1110

    Получается, что каждый единичный разряд побывает в каждом разряде.
    В итоге нужно посчитать следующую сумму:
     $n_1 + n_1 * 2 + n_1 * 4 + n_1 * 8 + \dots$ 
    Это можно сделать за  $O(n)$ 
*/

#include <stdio.h>
#include <string>
#include <algorithm>

int main() {
    int len;
    scanf("%d", &len);

    char buf[1+1000000];
    scanf("%1000000s", buf);
    std::string s(buf);
    std::reverse(s.begin(), s.end());

    // Подсчет количества единичных битов:
    int n1 = 0;
    for (auto it : s) n1 += (it == '1');

    // Суммирование сдвигов:
    int rem = 0;
    for (auto& it : s) {
        rem += n1;
        it = (rem % 2) + '0';
        rem /= 2;
    }

    // Логическое отрицание
    for (auto& it : s) it = char('1' - it + '0');

    // Прибавление 1
    rem = 1;
    for (auto& it : s) {
        rem += (it - '0');
        it = rem % 2 + '0';
        rem /= 2;
    }

    // Отбрасывание ведущих нулей:
    while (s.back() == '0' && (int)s.size() > 1) s.pop_back();

    // Вывод результата:
    std::reverse(s.begin(), s.end());
    printf("%s\n", s.c_str());

    return 0;
}

```

ЗАДАЧА №486

Рыбаки - 2

(Время: 1 сек. Память: 16 Мб Сложность: 53%)

Однажды N рыбаков отправились на рыбалку, где поймали X рыб. После этого рыбаки легли спать. Утром, просыпаясь друг за другом, каждый из них делил выловленную рыбу на N частей. Каждый раз в остатке оставалось ровно K рыб ($0 < K < N$). Эти K рыб выбрасывались обратно в море. Рыбак забирал свою часть улова и отбывал домой, не зная ничего о том, поступал ли уже кто-либо из остальных рыбаков таким же образом.

Ваша задача – определите при заданных N и K минимально возможное целое положительное значение X – число рыб, удовлетворяющее условию задачи.

Входные данные

Входной файл INPUT.TXT содержит два целых числа N и K , разделенные пробелом ($2 \leq N \leq 15$, $0 < K < N$).

Выходные данные

Выходной файл OUTPUT.TXT должен содержать одно целое положительное число X – наименьшее возможное количество выловленной рыбаками рыбы.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 1	25
2	4 3	247

```
/*
    Задача: 486. Рыбаки - 2

    Решение: математическое моделирование,  $O(\log(n))$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/
```

```
#include <bits/stdc++.h>
typedef long long ll;
ll pow(ll a, ll n) {
    ll ret = 1;
    while (n > 0) {
        if (n & 1) { ret *= a; }
        n >>= 1; a *= a;
    }
    return ret;
}
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    ll n, k;
    while (std::cin >> n >> k) {
        if (n == 2) {
            for (ll sum1 = 1; ; sum1++) {
                ll rem1 = sum1 % n;
```

```

        if (rem1 != k) {
            continue;
        }
        ll sum2 = sum1 - sum1 / n - k;
        ll rem2 = sum2 % n;
        if (rem2 != k || sum2 / n == 0) {
            continue;
        }
        std::cout << sum1 << std::endl;
        break;
    }
} else {
    std::cout << (ll)pow(n, n) - (n-1) * k << std::endl;
}
}
return 0;
}

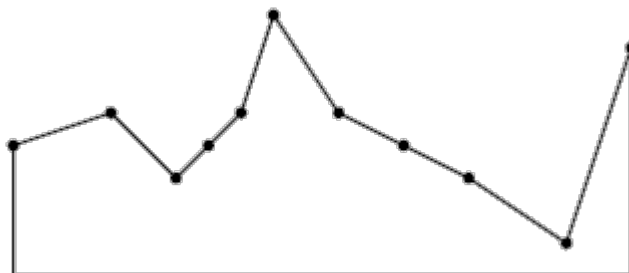
```

ЗАДАЧА №593

Башни - 2

(Время: 1 сек. Память: 16 Мб Сложность: 53%)

Для того, чтобы защититься от некоторых соседей, король решил построить стену, имеющую форму отрезка. С некоторыми соседями король находится в хороших отношениях, а некоторым готовится объявить войну. Король решил не загроживать себя от друзей очень высокой стеной. Однако, стена, отделяющая его от врагов, должна быть достаточно высокой. Было решено, что для наблюдения за прилегающей территорией нужно построить башни. При этом, на участках между башнями высота стен должна изменяться равномерно.



После того, как стена и башни были построены, король заметил, что башни могут быть использованы для наблюдения за состоянием других башен. Однако, некоторые башни оказались очень высокими и загорюдили другие.

Для каждой башни король попросил вас выяснить, сколько других башен из нее видно.

Входные данные

В первой строке входного файла INPUT.TXT находится n ($2 \leq n \leq 2000$) - количество башен стены. В следующих n строках находятся натуральные числа x_i и h_i ($0 \leq x_i \leq 100000$, $1 \leq h_i \leq 10000$) - координата и высота i -ой башни. Все x_i различны.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать n строк. В i -ой строке выведите количество башен, которые видно из башни номер i .

Пример

№	INPUT.TXT	OUTPUT.TXT
1	11	2
	0 4	5
	3 5	4
	5 3	4
	6 4	4
	7 5	10
	8 8	4
	10 5	4
	12 4	5
	14 3	3
	17 1	5
	19 7	

/*

Решение: геометрия, векторное произведение, сортировка, $O(n^2)$

Автор: Дмитрий Козырев, github: dmzk, e-mail: dmkozyrev@rambler.ru

```
*/  
  
#include <stdio.h>  
#include <vector>  
#include <algorithm>  
  
typedef long long ll;  
  
struct Vector {  
    int x, y;  
  
    Vector(int x_ = 0, int y_ = 0) : x(x_), y(y_) { }  
  
    Vector operator-() const {  
        return Vector(-x, -y);  
    }  
  
    Vector operator-(const Vector& p) const {  
        return Vector(x - p.x, y - p.y);  
    }  
  
    ll cross(const Vector& p) const {  
        return (ll)x * p.y - (ll)y * p.x;  
    }  
};  
  
struct Point : public Vector {  
  
    int id;  
  
    Point(int x_ = 0, int y_ = 0, int id_ = 0) : Vector(x_, y_), id(id_) { }  
};  
  
int main() {  
    int n; scanf("%d", &n);  
    std::vector<Point> pt(n);  
    for (int i = 0; i < n; ++i) {  
        scanf("%d %d", &pt[i].x, &pt[i].y);  
        pt[i].id = i;  
    }  
    std::sort(pt.begin(), pt.end(), [](const Point& a, const Point& b){return a.x < b.x;});  
    std::vector<int> answ(n);  
    for (int i = 0; i < n; ++i) {  
        for (int j = i+1, k = i; j < n; ++j) {  
            if ((pt[k] - pt[i]).cross(pt[j] - pt[i]) < 0) {  
                continue;  
            }  
            k = j;  
            answ[pt[i].id]++;  
            answ[pt[j].id]++;  
        }  
    }  
    for (auto& it : answ) {  
        printf("%d\n", it);  
    }  
    return 0;  
}
```

ЗАДАЧА №735

Циклическое k-расширение

(Время: 1 сек. Память: 16 Мб Сложность: 53%)

Вася недавно узнал, что такое циклическое k-расширение строки S. Его можно получить следующим образом: склеить k экземпляров строки S, а потом взять первые k символов результата.

Узнав это, Вася обрадовался, взял некоторую строку, и начал к ней применять описанную операцию, не запоминая, какое он каждый раз брал k.

Вам дана часть строки, получившейся у Васи. Ваша задача определить, не ошибся ли Вася в своих сложных преобразованиях, т. е., мог ли у него из первоначальной строки получиться ответ, содержащий данную строку в качестве подстроки.

Входные данные

В первой строке входного файла INPUT.TXT находится изначальная строка, которую Вася бережно записал перед тем, как приступить к своим действиям. Во второй строке находится подстрока результата, полученного Васей. Обе строки не пусты и по длине не превышают 5 000 символов. Строки могут состоять из больших и маленьких английских букв (с учетом регистра), а также цифр.

Выходные данные

В выходной файл OUTPUT.TXT выведите "NO", если можно точно сказать, что Вася ошибся, и "YES", если мог и не ошибиться.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	abc abc	YES
2	abcd bcabc	YES
3	abcabc abcA	NO

```
/*
    Задача: 735. Циклическое k-расширение

    Решение: строки, динамическое программирование, O(n^2)

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define isz(x) (int)(x).size()

typedef std::vector<int> vi;

bool solve(const std::string& s, const std::string& t) {
    vi can(isz(t)+1);
    can[isz(t)]=true;
```

```

for (int i = isz(t)-1; i >= 0; --i) {
    int j = 0;
    while (j < isz(s) && i+j < isz(t) && t[i+j] == s[j]) {
        can[i] = can[i] || can[i+j+1];
        ++j;
    }
}
vi queue, temp;
for (int i = 0; i < isz(s); ++i) {
    if (s[i] == t[0]) {
        queue.push_back(i);
    }
}
bool answ = can[0];
for (int len = 1; len <= isz(t); ++len) {
    temp.clear();
    for (int i : queue) {
        answ = answ || can[len];
        if (i + 1 < isz(s) && len < isz(t) && s[i+1] == t[len]) {
            temp.push_back(i+1);
        }
    }
    queue = temp;
}
return answ;
}
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (std::string s, t; std::cin >> s >> t; std::cout << (solve(s,t) ? "YES\n" : "NO\n"))
        return 0;
}

```

ЗАДАЧА №745

Карта

(Время: 1 сек. Память: 16 Мб Сложность: 53%)

Одним из разделов функционального анализа является теория сжимающих операторов. Важным фактом, который в ней доказывается, является теорема Банаха. Она гласит, что у оператора сжатия есть ровно одна неподвижная точка.

Интересным следствием из этой теоремы является следующее утверждение. Пусть есть карта небольшой части поверхности Земли (поверхность считается плоской). Если карту положить в некотором месте той части поверхности, которую она изображает, то будет существовать ровно одна точка, изображение которой на карте лежит на ней.

Для удобства будем считать, что изображенная на карте часть поверхности Земли имеет форму прямоугольника со сторонами $2W$ и $2H$ метров. Введем прямоугольную декартову систему координат так, что ось Ox направлена с запада на восток, а ось Oy – с юга на север. Единичный отрезок выберем равным одному метру. Кроме этого, поместим начало координат в центр рассматриваемой части поверхности Земли, а стороны рассматриваемого прямоугольника параллельны осям координат. Расположим карту размером $2a$ на $2b$ сантиметров так, что ее центр находится в точке с координатами (x, y) . Таким образом, изображенная на карте поверхность Земли имеет форму прямоугольника с углами (W, H) , $(-W, H)$, $(-W, -H)$, $(W, -H)$, а углы карты расположены в точках $(x + a/100, y + b/100)$, $(x - a/100, y + b/100)$, $(x - a/100, y - b/100)$, $(x + a/100, y - b/100)$.

Найдите точку, изображение которой лежит на ней при таком расположении карты.

Входные данные

Входной файл INPUT.TXT содержит целые числа W, H, x, y, a, b ($1 \leq W, H, x, y, a, b \leq 1000$). Гарантируется, что карта целиком лежит внутри той части поверхности Земли, которая на ней изображена.

Выходные данные

В выходной файл OUTPUT.TXT выведите координаты искомой точки с точностью до 10^{-6} .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	10 10 0 0 5 5	0.0 0.0
2	10 10 1 1 5 10	1.0050251256281408 1.0101010101010102

```
/*
    Задача: 745. Карта

    Решение: геометрия, двумерный тернарный поиск, O(log(1/EPS)^2)

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
typedef long double ld;
```

```

int main() {
    std::cout << std::setprecision(6) << std::fixed;
    ld w, h, cx, cy, a, b;
    std::cin >> w >> h >> cx >> cy >> a >> b;
    a /= 100;
    b /= 100;
    auto calc = [&](ld x, ld y) {
        ld mx = x / w * a + cx;
        ld my = y / h * b + cy;
        return (mx - x) * (mx - x) + (my - y) * (my - y);
    };
    auto search_x = [&](ld y, ld& x) {
        ld x1 = -w, x2 = w;
        for (int i = 0; i < 100; ++i) {
            ld l = (6 * x1 + 5 * x2) / 11;
            ld r = (5 * x1 + 6 * x2) / 11;
            ld dl = calc(l, y);
            ld dr = calc(r, y);
            if (dl <= dr) { x2 = r; }
            else { x1 = l; }
        }
        x = (x1+x2)/2;
        return calc(x,y);
    };
    auto search_y = [&](ld &x, ld &y) {
        ld y1 = -h, y2 = h;
        for (int i = 0; i < 100; ++i) {
            ld l = (6 * y1 + 5 * y2) / 11;
            ld r = (5 * y1 + 6 * y2) / 11;
            ld x1, dl, xr, dr;
            dl = search_x(l, x1);
            dr = search_x(r, xr);
            if (dl <= dr) { y2 = r; }
            else { y1 = l; }
        }
        y = (y1+y2)/2;
        return search_x(y, x);
    };
    ld x, y;
    search_y(x, y);
    std::cout << x << ' ' << y << std::endl;
    return 0;
}

```

ЗАДАЧА №826

Жизнь цвета индиго

(Время: 2 сек. Память: 32 Мб Сложность: 53%)

Мальчику Севе очень нравится цвет индиго (это такой темно-синий цвет). Сева всячески старается окружить себя вещами этого цвета.

Скоро Новый год, и Сева решил изготовить гирлянду с лампочками, чтобы украсить ей свою комнату. Он купил n патронов для лампочек и соединил $(n-1)$ -им проводом так, что в гирлянде не образовалось ни одного замкнутого контура. Его гирлянда, таким образом, с точки зрения математики, представляет собой дерево. Для того чтобы гирлянда была готова, осталось совсем немного надо только добавить провод для подключения гирлянды в электрическую сеть и вкрутить разноцветные лампочки в патроны.

У Севы есть лампочки трех разных цветов: синего, фиолетового и индиго. При этом, в некоторые патроны из соображений красоты нельзя устанавливать лампочки определенных цветов. Также, из соображений красоты, в гирлянде не должно быть двух лампочек одного цвета, напрямую соединенных проводом. Разумеется, Сева хочет, чтобы в гирлянде было как можно больше лампочек его любимого цвета.

Помогите Севе. Напишите программу, которая найдет максимальное количество лампочек цвета индиго, которые можно установить в собранную Севой гирлянду.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n количество патронов для лампочек в Севиней гирлянде ($1 \leq n \leq 50000$). Последующие $(n-1)$ строка содержат каждая по два числа: u_i и v_i ($1 \leq u_i, v_i \leq n$) – номера патронов, соединенных соответствующим проводом.

Далее следуют n строк с описанием патронов. Каждая из них не пуста и состоит не более, чем из трех различных символов «I», «B» или «V». i -ая из этих строк описывает i -ый патрон, а именно: если она содержит символ «I», то в i -ый патрон можно устанавливать лампочку цвета индиго, «B» - можно устанавливать лампочку синего цвета, «V» - можно устанавливать лампочку фиолетового цвета.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное число: ответ на задачу. Если же при наложенных ограничениях Севе вообще не удастся собрать гирлянду, выведите число -1 .

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 1 2 2 3 IB IV IB	2

Задача: 826. Жизнь цвета индиго

Решение: динамическое программирование, деревья, поиск в глубину, $O(n)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

*/

```
#include <bits/stdc++.h>
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
struct Tree {
    int n;
    vvi adj, can, max;
    vi vis;
    Tree(int n_ = 0) : n(n_) {
        vis.resize(n+1);
        adj.resize(n+1);
        can.assign(n+1, vi{0,0,0});
        max.assign(n+1, vi{0,0,0});
    }
    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void set_vert(int u, std::string s) {
        // I->0, B->1, V->2
        for (auto it : s) {
            if (it == 'I') { can[u][0] = 1; }
            if (it == 'B') { can[u][1] = 1; }
            if (it == 'V') { can[u][2] = 1; }
        }
    }
    void dfs(int u, int p) {
        if (vis[u]) { return; }
        vis[u] = true;
        for (int v : adj[u]) {
            if (v != p) { dfs(v,u); }
        }
        for (int i = 0; i < 3; ++i) {
            int sum = 0;
            for (int v : adj[u]) {
                int mx = 0;
                if (v == p) { continue; }
                bool temp = false;
                for (int j = 0; j < 3; ++j) {
                    if (i == j || !can[v][j]) { continue; }
                    temp = true;
                    mx = std::max(mx, max[v][j]);
                }
                can[u][i] = can[u][i] && temp;
                sum += mx;
            }
            if (can[u][i]) {
                max[u][i] = (i == 0) + sum;
            }
        }
    }
    int solve() {
        dfs(1,0);
        for (int u = 1; u <= n; ++u) {
            assert(vis[u]);
        }
        int ret = -1;
        for (int i = 0; i < 3; ++i) {
            if (!can[1][i]) { continue; }
            ret = std::max(ret, max[1][i]);
        }
        return ret;
    }
}
```

```
};

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (int n; std::cin >> n; ) {
        Tree tree(n);
        for (int i = 1, u, v; i < n; ++i) {
            std::cin >> u >> v;
            tree.add_edge(u,v);
        }
        std::string s;
        for (int i = 1; i <= n; ++i) {
            std::cin >> s;
            tree.set_vert(i, s);
        }
        std::cout << tree.solve() << std::endl;
    }
    return 0;
}
```

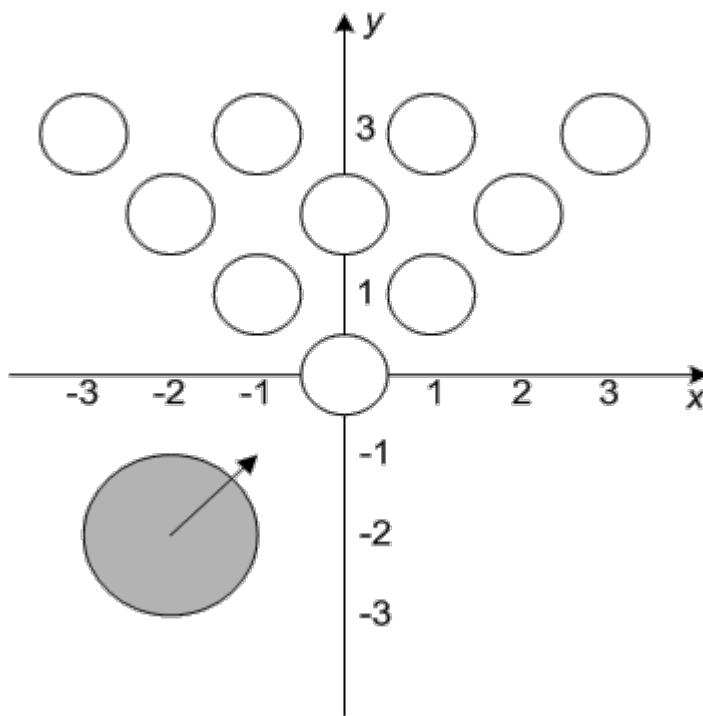

ЗАДАЧА №856

Космический кегельбан

(Время: 2 сек. Память: 16 Мб Сложность: 53%)

На планете Плюк открылся новый космический кегельбан. Поле для кегельбана представляет собой бесконечную плоскость, на которой расставлены кегли.

Каждая кегля представляет собой высокий цилиндр с основанием в виде круга радиусом r метров. Все кегли одинаковые. Кегли расставлены по следующим правилам. Кегли образуют n рядов, в первом ряду стоит одна кегля, во втором — две, и так далее. В последнем n -м ряду стоит n кеглей. Введем на плоскости систему координат таким образом, чтобы единица измерения была равна одному километру. Центр единственной кегли в первом ряду находится в точке $(0, 0)$. Центры кеглей во втором ряду находятся в точках $(-1, 1)$ и $(1, 1)$. Таким образом, центры кеглей в i -м ряду находятся в точках с координатами $(-i + 1, i - 1)$, $(-i + 3, i - 1)$, \dots , $(i - 1, i - 1)$.



Игра происходит следующим образом. Используется шар с радиусом q метров. Игрок выбирает начальное положение центра шара (x_c, y_c) и вектор направления движения шара (v_x, v_y) . После этого шар помещается в начальную точку и движется, не останавливаясь, в направлении вектора (v_x, v_y) . Считается, что шар сбил кеглю, если в процессе движения шара имеет место ситуация, когда у шара и кегли есть общая точка. Сбитые кегли не меняют направления движения шара и не сбивают соседние кегли при падении.

На рисунке приведен пример расположения кеглей для $r = 500$, $n = 4$ и шара для $q = 1000$, $x_c = -2$, $y_c = -2$, $v_x = 1$, $v_y = 1$.

Требуется написать программу, которая по заданным радиусу кегли r , количеству рядов кеглей n , радиусу шара q , его начальному положению (x_c, y_c) и вектору направления движения (v_x, v_y) определяет количество кеглей, сбитых шаром.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа: r и n , разделенных ровно одним пробелом ($1 \leq r \leq 700$, $1 \leq n \leq 200\,000$). Вторая строка входного файла содержит целое число q ($1 \leq q \leq 10^9$). Третья строка входного файла содержит два целых числа x_c и y_c , разделенных ровно одним пробелом ($-10^6 \leq x_c \leq 10^6$, $-10^6 \leq y_c \leq 10^6$, $1000 \times y_c < -(r + q)$). Четвертая строка входного файла содержит два целых числа v_x и v_y , разделенных ровно одним пробелом ($-10^6 \leq v_x \leq 10^6$, $0 < v_y \leq 10^6$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число — количество сбитых кеглей.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	500 4 1000 -2 -2 1 1	7

```
/*
    Задача: 856. Космический кетельбан

    Решение: геометрия, прямая, точка, окружность, бинарный поиск,  $O(n \log(n))$ 

    Автор: Дмитрий Козырев, https://github.com/dmkz , dmkozyrev@rambler.ru
*/

#include <iostream>
#include <cmath>
#include <algorithm>
#include <cassert>
#include <functional>

typedef long double Real;

const Real PI = std::acos(Real(-1));

struct Point {
    Real x, y;

    static Point read() {
        Real x_, y_;
        std::cin >> x_ >> y_;
        return Point{x_, y_};
    }

    inline Real norm() const {
        return std::sqrt(x * x + y * y);
    }

    inline Point operator-(const Point& other) const {
        return Point{x-other.x, y-other.y};
    }

    inline Point operator+(const Point& other) const {
        return Point{x+other.x, y+other.y};
    }

    inline Point operator/(const Real q) const {
        return Point{x / q, y / q};
    }

    inline Point operator*(const Real q) const {
        return Point{x * q, y * q};
    }

    inline Point rotate(const Real angle) const {
        return Point {
            std::cos(angle) * x - std::sin(angle) * y,
            std::sin(angle) * x + std::cos(angle) * y
        };
    }
};

inline Real det(const Point& a, const Point& b) {
```

```

    return a.x * b.y - a.y * b.x;
}

struct Line {

    Real A, B, C;

    Line(const Point& a, const Point& b) {
        //  $-(b.y-a.y)*(x-a.x) + (b.x-a.x)*(y-a.y) = 0$ 
        A = a.y - b.y;
        B = b.x - a.x;
        C = -(a.x * A + a.y * B);
    }

    inline Point intersect(const Line& L) const {
        auto q = det(Point{ A, B}, Point{ L.A, L.B});
        auto a = det(Point{-C, B}, Point{-L.C, L.B});
        auto b = det(Point{ A,-C}, Point{ L.A,-L.C});
        return Point{a / q, b / q};
    }

    inline Real value(const Point& p) const {
        return A * p.x + B * p.y + C;
    }
};

inline Real dist(const Point& p, const Line& L) {
    return std::abs(L.value(p)) / Point{L.A, L.B}.norm();
}

inline Real dist(const Point& a, const Point& b) {
    return (a-b).norm();
}

int solve(int n, Real r, Real R, Point C, Point V) {
    // Строим две прямые, отвечающие за пересечения кегель с шаром:
    Line L1(C+V.rotate(PI/2)*R, C+V.rotate(PI/2)*R+V);
    Line L2(C-V.rotate(PI/2)*R, C-V.rotate(PI/2)*R+V);
    // Тестовая точка, которая точно лежит справа от каждой прямой:
    Point test = C + V.rotate(-PI/2) * 2 * R;
    // Максимальный и минимальный номера кегель в ряду:
    int max_id = (n % 2 == 1) ? (n / 2) : ((n-1) / 2);
    int min_id = (n % 2 == 1) ? (-n / 2) : ((-n-1) / 2);
    // Центр кегли в ряду по ее номеру:
    std::function<Point(int)> point = [n](const int id) {
        if (n % 2 == 1) {
            return Point{Real(id) * 2000, Real(n-1) * 1000};
        } else {
            return Point{1000 + Real(id) * 2000, Real(n-1) * 1000};
        }
    };
    // Бинарный поиск по первой кегле слева от левой прямой, не пересекающей ее:
    int low = min_id - 1, high = max_id + 1;
    while (high - low > 1) {
        int mid = (low + high) / 2;
        if (L1.value(point(mid)) * L1.value(test) <= 0) {
            low = mid;
        } else {
            high = mid;
        }
    }
    while (low >= min_id && dist(point(low), L1) <= r) --low;
    // Количество целых кегель слева:
    int left = std::max(0, low - min_id + 1);
    // Бинарный поиск по первой кегле справа от правой прямой, не пересекающей ее:
    low = min_id - 1, high = max_id + 1;
    while (high - low > 1) {
        int mid = (low + high) / 2;
        if (L2.value(point(mid)) * L2.value(test) <= 0) {

```

```

        low = mid;
    } else {
        high = mid;
    }
}
low = high;
while (low <= max_id && dist(point(low), L2) <= r) ++low;
// Количество целых кегель справа:
int right = std::max(0, max_id - low + 1);
return n - left - right;
}

int main() {
    int r, n, R;
    std::cin >> r >> n >> R;

    Point P = Point::read();
    Point V = Point::read();
    P = P * 1000;
    V = V / V.norm();

    long long answ = 0;
    for (int i = 1; i <= n; ++i) {
        answ += solve(i, r, R, P, V);
    }
    std::cout << answ;
    return 0;
}

```

ЗАДАЧА №1186

RSQ с изменением элемента

(Время: 1 сек. Память: 32 Мб Сложность: 53%)

Требуется реализовать эффективную структуру данных, позволяющую изменять элементы массива $A[1..N]$ и вычислять сумму элементов отрезка $[L, R]$.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N – размер массива ($N \leq 10^5$). Во второй строке записаны N целых чисел – элементы массива, целые числа, не превосходящие 10^9 по абсолютной величине. Третья строка содержит натуральное число M – количество запросов ($M \leq 30\,000$). Каждая из следующих M строк представляет собой описание запроса. Сначала вводится одна буква, кодирующая вид запроса («s» – вычислить сумму, «u» – обновить значение элемента). Следом за «s» записаны два числа L и R – номера левой и правой границы отрезка. Следом за «u» записаны два числа I и X – номер элемента и его новое значение, не превосходящее 10^9 по абсолютной величине ($1 \leq L \leq R \leq N, 1 \leq I \leq N$).

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса на вычисление суммы выведите результат. Все числа следует выводить в одну строку, разделяя пробелом.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	10 613 263 312 670 216 142 976 355 488 370 10 s 2 7 s 4 8 u 7 969 u 1 558 s 2 7 u 2 731 s 4 9 s 1 3 u 8 76 u 5 377	2579 2359 2572 2840 1601

```
#include <stdio.h>
#include <vector>
#include <cassert>

typedef long long ll;

struct SqrtDecomposition {
    std::vector<int> arr;
    std::vector<ll> sum;

    const int GSIZE = 128;
```

```

SqrtDecomposition(const std::vector<int>& vec) : arr(vec) {
    sum.assign(((int)arr.size() + GSIZE - 1) / GSIZE, 0);
    for (int g = 0; g < (int)sum.size(); ++g) {
        const int after = std::min((g+1)*GSIZE, (int)arr.size());
        auto& it = sum[g];
        for (int i = g * GSIZE; i < after; ++i) {
            it += arr[i];
        }
    }
}

void set(int pos, int value) {
    const int g = pos / GSIZE;
    sum[g] += (value - arr[pos]);
    arr[pos] = value;
}

ll get(int l, int r) {
    const int gl = l / GSIZE;
    const int gr = r / GSIZE;
    if (gl == gr) {
        ll s = 0;
        for (int i = l; i <= r; ++i) {
            s += arr[i];
        }
        return s;
    } else {
        ll s = 0;
        for (int i = l, after = (gl+1)*GSIZE; i < after; ++i) {
            s += arr[i];
        }
        for (int g = gl+1; g < gr; ++g) {
            s += sum[g];
        }
        for (int i = gr * GSIZE; i <= r; ++i) {
            s += arr[i];
        }
        return s;
    }
}

};

int main() {
    int n; scanf("%d", &n);
    std::vector<int> arr(n);
    for (auto& it : arr) {
        scanf("%d", &it);
    }
    SqrtDecomposition sd(arr);
    int q; scanf("%d", &q);
    while (q--) {
        char t; int l, r;
        scanf(" %c %d %d", &t, &l, &r);
        if (t == 's') {
            printf("%I64d ", sd.get(l-1, r-1));
        } else {
            assert(t == 'u');
            sd.set(l-1, r);
        }
    }
    return 0;
}

```

ЗАДАЧА №1364

Водолей

(Время: 0,4 сек. Память: 32 Мб Сложность: 53%)

У исполнителя «Водолей» есть два сосуда: первый объемом А литров, второй объемом В литров, а также кран с водой. Водолей может выполнять следующие операции:

- 1. Наполнить сосуд А (>A).
- 2. Наполнить сосуд В (>B).
- 3. Вылить воду из сосуда А (A>).
- 4. Вылить воду из сосуда В (B>).
- 5. Перелить воду из сосуда А в сосуд В (A>B).
- 6. Перелить воду из сосуда В в сосуд А (B>A).

Команда переливания из одного сосуда в другой приводят к тому, что либо первый сосуд полностью опустошается, либо второй сосуд полностью наполняется.

Требуется составить алгоритм для «Водолея», который позволяет получить в точности N литров в одном из сосудов.

Входные данные

Входной файл INPUT.TXT содержит три натуральных числа А, В и N, не превосходящих 10^4 .

Выходные данные

В выходной файл OUTPUT.TXT выведите алгоритм действий «Водолея», который позволяет получить в точности N литров в одном из сосудов, если же такого алгоритма не существует, то следует вывести «Impossible».

Количество операций в алгоритме не должно превышать 10^5 . Гарантируется, что если задача имеет решение, то существует решение, состоящее не более, чем из 10^5 операций.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 5 1	>A A>B >A A>B
2	3 5 6	Impossible

```
/*
Задача: 1364. Водолей
Решение: диофантово уравнение, целочисленная арифметика, O(n)
*/
#include <bits/stdc++.h>
template<const bool out>
bool solve(int a, int b, const int n) {
    // Diophantine equation is x * a + y * b == n, solution exists if n % gcd(a,b) == 0
    int gcd = std::__gcd(a,b);
```

```

if (n % gcd != 0 || n > std::max(a,b)) {
    out && std::cout << "Impossible" << std::endl;
    return false;
}
char ca = 'A', cb = 'B';
if (a > b) {
    std::swap(a,b);
    std::swap(ca, cb);
}
int sa = 0, sb = 0;
while (sa != n && sb != n) {
    if (sa == 0) {
        out && std::cout << ">" << ca << "\n";
        sa = a;
    } else if (sb == b) {
        out && std::cout << cb << ">" << "\n";
        sb = 0;
    } else {
        out && std::cout << ca << ">" << cb << "\n";
        const int delta = std::min(sa, b - sb);
        sa -= delta;
        sb += delta;
    }
}
return true;
}
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int a, b, n;
    while (std::cin >> a >> b >> n) { solve<1>(a,b,n); }
    return 0;
}

```


ЗАДАЧА №1565

Страшный сон Егора

(Время: 1 сек. Память: 16 Мб Сложность: 53%)

Егор — школьник, совсем скоро ему участвовать в региональном этапе Всероссийской олимпиады школьников по информатике. К олимпиаде он готов, полностью уверен в победе в регионе, но чтобы пройти в следующий, заключительный этап, ему нужно постараться.

Накануне перед олимпиадой ему приснился страшный сон: Егор не набрал необходимые T проходных баллов по итогу олимпиады, а значит на заключительный этап он поедет только по квоте! Об отношении с квотниками на заключительном этапе Егор знает не понаслышке, поэтому он просит вас о помощи.

Помимо проходного балла T — минимального количества баллов, которое нужно набрать, ему приснилось, что на олимпиаде будет N задач, по каждой можно набрать целый балл от 0 до 100, при этом в i -й задаче можно будет набрать только балл, кратный k_i .

Егору не приснились сами задачи, но приснились затраты энергии s_i — сколько требуется энергии на прочтение задачи, и c_i — сколько потребуется энергии чтобы набрать по задаче один балл. Естественно, нельзя набирать баллы по задаче, не прочитав её.

Егор хочет набрать как минимум T баллов, но при этом хочет потратить суммарно как можно меньше энергии, чтобы потом смело можно было заявлять, что он писал олимпиаду на расслабоне.

Определите минимальное количество энергии, необходимое Егору, а также для каждой задачи, какой балл по ней нужно набрать, чтобы достичь этого результата.

Входные данные

В первой строке входного файла INPUT.TXT содержатся два целых числа N и T ($1 \leq N \leq 100$; $1 \leq T \leq 100 \times N$).

В следующих N строках содержится по три числа s_i , c_i и k_i для каждой задачи ($1 \leq s_i, c_i \leq 10^5$; $1 \leq k_i \leq 100$; k_i — делитель числа 100).

Выходные данные

В первую строку выходного файла OUTPUT.TXT выведите минимальное количество энергии, которого хватит для достижения цели Егора.

Во второй строке выведите N чисел b_i через пробел — баллы, которые нужно набирать по каждой задаче. Каждое b_i должно делиться на k_i , а также быть $0 \leq b_i \leq 100$.

Если существует несколько решений, выведите любое.

Пример

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	3 166 10 5 1 1 1 10 15 4 25	406 16 100 50
---	--------------------------------------	------------------

```

#include <stdio.h>
#include <vector>
#include <algorithm>

typedef unsigned int Int;

int main() {
    Int n, need;
    scanf("%u %u", &n, &need);
    std::vector<Int> s(1+n), c(1+n), k(1+n, 1);
    for (Int i = 1; i <= n; ++i) {
        scanf("%u %u %u", &s[i], &c[i], &k[i]);
    }
    const Int INF = 2u*1000u*1000u*1000u;
    static Int min[1+100][1+100*100], from[1+100][1+100*100];
    for (Int i = 0; i <= 100u; ++i) for (Int j = 0; j <= 100u*100u; ++j) {
        min[i][j] = INF;
        from[i][j] = -1;
    }
    min[0][0] = 0;
    for (Int t = 1; t <= n; ++t) {
        for (Int curr = 0; curr <= t * 100; ++curr) {
            for (Int score = 0; curr >= score && score <= 100; ++score) {
                if (score % k[t] != 0) {
                    continue;
                }
                Int sum = score == 0 ? min[t-1][curr] : min[t-1][curr-score] + s[t] + score;
                if (min[t][curr] > sum) {
                    min[t][curr] = sum;
                    from[t][curr] = curr-score;
                }
            }
        }
    }
    int answ = need;
    for (int curr = need; curr <= 100*100; ++curr) {
        if (min[n][answ] > min[n][curr]) {
            answ = curr;
        }
    }
    printf("%u\n", min[n][answ]);
    Int seq[101];
    for (Int i = n; i >= 1; --i) {
        seq[i] = answ - from[i][answ];
        answ = from[i][answ];
    }
    for (Int i = 1; i <= n; ++i) {
        printf("%u ", seq[i]);
    }

    return 0;
}

```

ЗАДАЧА №507

Адронный коллайдер

(Время: 1 сек. Память: 16 Мб Сложность: 54%)

Внутри адронного коллайдера образовалось N разновидностей новых частиц в количестве A_1, \dots, A_N единиц каждая. Большая часть новых частиц, однако, успевает прореагировать между собой раньше, чем эти частицы сможет уловить детектор, поэтому физикам очень важно предсказать конечный итог реакции между частицами.

Для простоты будем считать, что в реакции участвуют 2 частицы, с одним из следующих результатов, в зависимости от типа:

- Первая частица уничтожает вторую
- Частицы отскакивают друг от друга без какого-либо вреда

Необходимо определить все возможные исходы эксперимента.

Входные данные

В первой строке входного файла INPUT.TXT задано количество видов частиц N ($1 \leq N \leq 4$). В следующей строке записаны N чисел, определяющие начальное количество частиц каждого типа A_i ($1 \leq A_i \leq 2$). Следующие N строк формируют матрицу $B[N][N]$ из чисел 0 и 1. Ненулевое значение ячейки B_{ij} указывает, что частица типа i при столкновении уничтожает частицу типа j . Если B_{ij} не ноль и B_{ji} не ноль, то в результате взаимодействия уничтожается либо частица i , либо частица j , но не обе сразу.

Выходные данные

В первой строке выходного файла OUTPUT.TXT следует вывести количество возможных исходов K . В каждой из последующих K строк должно содержаться описание исхода эксперимента в формате исходного файла.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 1 1 2 0 0 1 1 0 0 1 1 1	3 0 1 0 0 0 1 1 0 0
2	1 2 0	1 2

/*
Задача: 507. Адронный коллайдер
Решение: BFS, $O(4^n)$
Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

```

*/

#include <stdio.h>
#include <set>
#include <queue>

struct State {

    int number;

    State(int num) : number(num) { }

    State(int n1, int n2, int n3, int n4) {
        number = n1 + (n2 << 2) + (n3 << 4) + (n4 << 6);
    }

    int get(int i) {
        return (number >> 2*i) & 3;
    }

    State dec(int i) {
        return State(number - (1 << 2 * i));
    }
};

inline bool operator<(const State& a, const State& b) {
    return a.number < b.number;
}

int main() {
    int n; scanf("%d", &n);
    State s(0);
    {
        int cnt[4] = {};
        for (int i = 0; i < n; ++i) {
            int c; scanf("%d", &c);
            cnt[i] = c;
        }
        s = State(cnt[0], cnt[1], cnt[2], cnt[3]);
    }
    int g[4][4];
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            scanf("%d", &g[i][j]);
        }
    }
    std::set<State> visited;
    std::set<State> stfinal;
    std::queue<State> queue;
    visited.insert(s);
    queue.push(s);
    while (!queue.empty()) {
        auto curr = queue.front(); queue.pop();
        bool flag = true;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (!g[i][j]) continue;
                if (i == j) {
                    if (curr.get(i) >= 2) {
                        flag = false;
                        auto next = curr.dec(j);
                        if (visited.find(next) == visited.end()) {
                            queue.push(next);
                            visited.insert(next);
                        }
                    }
                }
            }
        }
        if (flag) {
            if (curr.get(i) > 0 && curr.get(j) > 0) {
                flag = false;
                auto next = curr.dec(j);
            }
        }
    }
}

```

```

        if (visited.find(next) == visited.end()) {
            queue.push(next);
            visited.insert(next);
        }
    }
}
if (flag) {
    stfinal.insert(curr);
}
}
printf("%d\n", (int)stfinal.size());
for (auto curr : stfinal) {
    for (int i = 0; i < n; ++i) {
        printf("%d ", curr.get(i));
    }
    printf("\n");
}

return 0;
}

```

ЗАДАЧА №616

Отношения

(Время: 1 сек. Память: 16 Мб Сложность: 54%)

Бинарным отношением R на множестве X называется множество упорядоченных пар элементов из X . Если X конечно и содержит n элементов, то отношение можно задать как квадратную булеву матрицу размера $n \times n$.

В некоторых случаях бывает важно задать отношение в более компактной форме. Один из способов компактного задания отношений применяется, в частности, при описании грамматик операторного предшествования.

Рассмотрим две функции f и g , каждая из которых сопоставляет элементам X целые числа. Будем говорить, что эти функции описывают отношение R , если для любых x и y из X пара (x, y) принадлежит R тогда и только тогда, когда $f(x) \leq g(y)$.

По заданному отношению R , найдите способ описать его указанным образом с помощью двух функций f и g , либо выясните, что это невозможно сделать.

Входные данные

Первая строка входного файла INPUT.TXT содержит n - количество элементов в множестве $X = \{x_1, x_2, \dots, x_n\}$ ($1 \leq n \leq 1\,000$). Следующие n строк описывают отношение R . Каждая строка содержит n символов, j -й символ i -й из этих строк равен 1, если (x_i, x_j) принадлежит R , и 0 в противном случае.

Выходные данные

На первой строке выходного файла OUTPUT.TXT выведите "YES", если отношение можно описать указанным образом. В этом случае вторая строка должна содержать n целых чисел в диапазоне от -10^9 до 10^9 - значения функции f на элементах x_1, x_2, \dots, x_n , соответственно, а третья строка должна описывать функцию g аналогичным образом. Если отношение нельзя закодировать описанным образом с помощью двух функций, выведите "NO" на первой строке выходного файла.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 111 110 100	YES 0 1 2 2 1 0
2	3 110 101 011	NO

/*
Задача: 616. Отношения
Решение: конструктив, жадный алгоритм, $O(n^2)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

```
*/
#include <bits/stdc++.h>
#define isz(x) (int)(x).size()
#define all(x) (x).begin(), (x).end()
typedef std::vector<std::string> vs;
typedef std::vector<int> vi;
bool solve(const int n, const vs& arr, vi& f, vi& g) {
    vi x(n), y(n);
    for (int i = 0; i < n; ++i) {
        y[i] = x[i] = i;
    }
    vi nx1(n), ny0(n);
    for (int i : x) {
        for (int j : y) {
            nx1[i] += (arr[i][j] == '1');
            ny0[j] += (arr[i][j] == '0');
        }
    }
    int curr = 0;
    while (isz(x) > 0 || isz(y) > 0) {
        int candX = -1, candY = -1;
        for (int i : x) {
            if (nx1[i] == isz(y)) {
                candX = i;
                break;
            }
        }
        for (int j : y) {
            if (ny0[j] == isz(x)) {
                candY = j;
                break;
            }
        }
        if (candX == -1 && candY == -1) {
            return false;
        }
        if (candX != -1) {
            f[candX] = curr++;
            x.erase(std::find(all(x), candX));
            for (int j : y) {
                ny0[j] -= (arr[candX][j] == '0');
            }
        }
        if (candY != -1) {
            g[candY] = curr++;
            y.erase(std::find(all(y), candY));
            for (int i : x) {
                nx1[i] -= (arr[i][candY] == '1');
            }
        }
    }
    return true;
}

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0);
    for (int n; std::cin >> n; ) {
        vs arr(n); for (auto &it : arr) { std::cin >> it; }
        vi f(n), g(n);
        bool flag = solve(n, arr, f, g);
        if (flag) {
            std::cout << "YES\n";
            for (auto it : f) { std::cout << it << ' '; }
            std::cout << "\n";
            for (auto it : g) { std::cout << it << ' '; }
            std::cout << "\n";
        } else {
            std::cout << "NO\n";
        }
    }
}
```

```
        }  
    }  
    return 0;  
}
```


ЗАДАЧА №703

АСМ-шахматы

(Время: 1 сек. Память: 16 Мб Сложность: 54%)

Во многих видах спортивных соревнований имеются различные ритуалы, направленные для примирения конкурирующих команд или игроков. Это может быть рукопожатие, поклон или даже разбрызгивание шампанского. АСМ (Alliance of Chess Masters) собирается создать свой собственный ритуал, шахматную миниигру, в которой участвует два игрока в союзе друг с другом (а не как обычно, друг против друга). Игра проходит на шахматной доске размером 3×3, у каждого из игроков имеется два шахматных коня, которых они должны переместить так, чтобы добраться от одной позиции до другой (игроки могут делать шаги в любом порядке, не обязательно по очереди). При этом два коня не могут занимать одну и ту же клетку.

Стартовое и завершающее положения определены судьей. Оказывается, что некоторые такие задачи являются более трудными, чем другие, а некоторые могут быть даже неразрешимыми - поэтому, некоторые игроки иногда неспособны закончить ритуал. Ваша задача состоит в том, чтобы написать программу, которая по заданным положениям начальной и конечной расстановки, сможет определить возможность успешного окончания игры, а в случае успеха сможет так же определить сложность задачи – минимально возможное количество ходов, необходимых для разрешения данной задачи.

Входные данные

Входной файл INPUT.TXT содержит 3 строки по 7 символов в каждой. Первые 3 символа каждой строки описывают соответствующую строку шахматной доски для начальной позиции, затем идет пробел и замыкающие 3 символа, описывающие аналогичным образом строку конечной позиции. Белый конь обозначается символом «W», а черный – символом «B», пустые клетки помечаются «.» (точка).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число – сложность задачи. Если задача не имеет решения, то следует вывести -1.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	WBB ..W W...W ...BB	4
2	..B ..B W.B ..B W..WW.	-1

```
/*
Задача: 703. АСМ-шахматы

Решение: поиск в ширину, графы, строки, O(n*log(n))

Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/
```

```

#include <bits/stdc++.h>
int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0);
    std::string fi, se, tmp;
    while (std::cin >> fi >> se) {
        for (int i = 0; i < 2; ++i) {
            std::cin >> tmp; fi += tmp;
            std::cin >> tmp; se += tmp;
        }
        std::map<std::string, int> dist;
        dist[fi] = 0;
        std::queue<std::string> queue;
        queue.push(fi);
        while (!queue.empty()) {
            auto curr = queue.front(); queue.pop();
            int currDist = dist[curr];
            for (int i = 0; i < 3; ++i) {
                for (int j = 0; j < 3; ++j) {
                    if (curr[3*i+j] == 'W' || curr[3*i+j] == 'B') {
                        for (int di = -2; di <= 2; ++di) {
                            for (int dj = -2; dj <= 2; ++dj) {
                                if (di * di + dj * dj != 5) { continue; }
                                const int ni = i + di, nj = j + dj;
                                if (ni < 0 || nj < 0 || ni >= 3 || nj >= 3) { continue; }
                                if (curr[3*ni+nj] != '.') { continue; }
                                std::swap(curr[3*i+j], curr[3*ni+nj]);
                                auto it = dist.find(curr);
                                if (it == dist.end()) {
                                    dist[curr] = currDist+1;
                                    queue.push(curr);
                                } else {
                                    assert(it->second <= currDist+1);
                                }
                                std::swap(curr[3*i+j], curr[3*ni+nj]);
                            }
                        }
                    }
                }
            }
        }
        auto it = dist.find(se);
        int answ = (it == dist.end() ? -1 : it->second);
        std::cout << answ << '\n';
    }
    return 0;
}

```

ЗАДАЧА №849

Задача про ферзей

(Время: 2 сек. Память: 16 Мб Сложность: 54%)

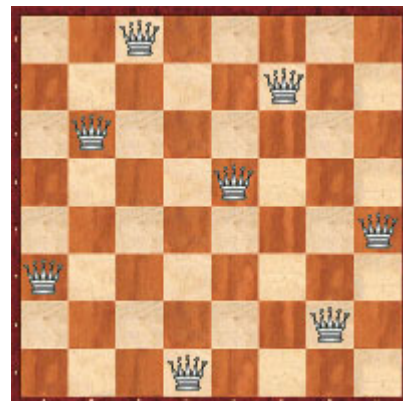
Требуется расставить N ферзей на клеточной шахматной доске размером $N \times N$ так, чтобы ни один из них не находился под боем другого.

Входные данные

Входной файл INPUT.TXT содержит натуральное число $N \leq 10^6$.

Выходные данные

В выходной файл OUTPUT.TXT выведите N пар целых чисел - координаты расстановки ферзей. Если для заданного N такой расстановки не существует, то следует вывести «No solution». Если существует несколько решений, то разрешается вывести любое из них.



Примеры

№	INPUT.TXT	OUTPUT.TXT
1	8	7 2 2 6 3 8 1 3 4 1 6 7 8 4 5 5
2	2	No solution

```
/*
    Задача: 849. Задача про ферзей

    Решение: конструктив, перебор, рекурсия,  $O(n)$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/
```

```
#include <bits/stdc++.h>
#define isz(x) (int)(x).size()
typedef std::pair<int,int> pii;
typedef std::vector<pii> vpii;

vpii greedy(const int n) {
    vpii answ;
    int row, col;
    for (row = 1, col = 2; col <= n; row++, col+=2) {
        answ.push_back(pii(row,col));
    }
    col = 1;
    while (row <= n) {
        answ.push_back(pii(row,col));
```

```

        row++; col += 2;
    }
    assert(isz(answ) == n);
    return answ;
}

vpaii greedy2(const int n) {
    vpaii answ;
    int row, col;
    for (row = 0, col = (n - 1) / 2; row < n / 2; ) {
        answ.push_back(paii(row+1,col+1));
        row++;
        col = (col + 2) % n;
    }
    for (row = n-1, col = n / 2; row >= n / 2; ) {
        answ.push_back(paii(row+1,col+1));
        row--;
        col -= 2;
        if (col < 0) {
            col += n;
        }
    }
    return answ;
}

vpaii greedy3(int n) {
    vpaii answ = greedy2(n / 2 * 2);
    if (n % 2 == 1) {
        answ.push_back(paii(n,n));
    }
    return answ;
}

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (int n; std::cin >> n; ) {
        if (n == 1) { std::cout << "1 1\n"; }
        else if (n <= 3) { std::cout << "No solution\n"; }
        else if (n % 6 != 2 && n % 6 != 3) {
            auto answ = greedy(n);
            for (auto it : answ) {
                assert(1 <= it.first && it.first <= n);
                assert(1 <= it.second && it.second <= n);
                std::cout << it.first << ' ' << it.second << "\n";
            }
        } else {
            auto answ = greedy3(n);
            for (auto it : answ) {
                assert(1 <= it.first && it.first <= n);
                assert(1 <= it.second && it.second <= n);
                std::cout << it.first << ' ' << it.second << "\n";
            }
        }
    }
    return 0;
}

```

ЗАДАЧА №873

Шифр «Решетка»

(Время: 1 сек. Память: 16 Мб Сложность: 54%)

Рассмотрим перестановочный шифр, называемый «Решетка» («перестановочный» означает, что символы, составляющие послание, не изменяются, но меняются местами). Суть его заключается в следующем. Выбирается четное число n , затем в квадрате $n \times n$ вырезается $n^2/4$ клеток. При этом клетки выбираются так, что если наложить решетку на квадрат $n \times n$, и затем последовательно развернуть ее на 90, 180 и 270 градусов, то каждый раз квадратики, совмещенные с вырезанными клетками, будут различны.

Такой квадрат $n \times n$ называется «правильным ключом». Ваша задача посчитать количество «правильных ключей». Так как это число может быть очень большим, мы предлагаем Вам найти его значение по модулю m . Ключи, получаемые поворотом на 90, 180 и 270 градусов считаются различными.

Входные данные

Входной файл INPUT.TXT содержит целые числа n и m ($2 \leq n, m \leq 10^6$), n - четно.

Выходные данные

В выходной файл OUTPUT.TXT выведите количество «правильных ключей» размером $n \times n$ по модулю m .

Пример

№	INPUT.TXT	OUTPUT.TXT
1	2 100	4

```
/*
    Задача: 873. Шифр "Решетка"

    Решение: быстрое возведение в степень, комбинаторика, O(log(n))

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <iostream>

typedef long long ll;

int pow(int a, ll k, int mod) {
    int ret = 1 % mod;
    while (k > 0) {
        if (k & 1) {
            ret = int(1LL * ret * a % mod);
        }
        k >>= 1;
        a = int(1LL * a * a % mod);
    }
    return ret;
}
```

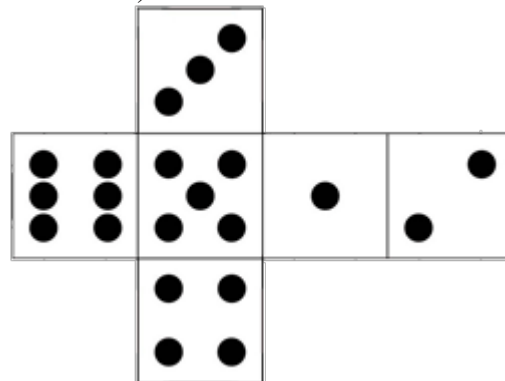
```
int main() {  
    int n, m;  
    std::cin >> n >> m;  
    std::cout << pow(4 % m, ll(n) * n / 4, m) << std::endl;  
    return 0;  
}
```

ЗАДАЧА №915

Кубик - 2

(Время: 1 сек. Память: 16 Мб Сложность: 54%)

В последнее время настольные игры стали очень популярны. В нашу жизнь возвращаются как некогда забытые, так и новые увлекательные игры. К вам попала совершенно новая, уникальная настольная игра. Для игры нужен игральный кубик (его развертка приведена на рисунке) и прямоугольное игровое поле, разбитое на клетки. В каждой клетке поля написано целое число.



Играть могут несколько человек по следующим правилам:

1. Первым ходом игрок ставит кубик в левую верхнюю клетку поля на любую грань.
2. Каждым следующим ходом игрок переходит на соседнюю справа или снизу клетку, перекатывая кубик на любую из 4 соседних граней.
3. За каждый ход игрок получает $K \cdot A$ очков, где A - число, записанное в текущей клетке поля, K - цифра на той грани кубика, которой он стоит на игровом поле.
4. Игрок ходит, пока не окажется в нижней правой клетке поля. Тогда очки за все его ходы суммируются.
5. Когда один игрок закончил ходить, начинает ходить второй игрок и т.д.
6. Победитель - игрок, набравший больше всех очков.

По заданному игровому полю определите максимальную сумму очков, которую можно получить, играя по указанным правилам.

Входные данные

Первая строка входного файла INPUT.TXT содержит разделенные пробелом числа N и M - размеры игрового поля ($1 \leq N \times M \leq 10^5$). Далее идет N строк по M чисел, разделенных пробелами - числа, записанные в клетках игрового поля. Все числа по модулю не превышают 10^3 .

Выходные данные

В выходной файл OUTPUT.TXT выведите целое число – наибольшее количество очков, которые можно получить при оптимальной игре.

Пример

Пояснение к примеру

Первоначально кубик следует установить на 6, далее вниз на 2, потом вниз на 6, вправо на 2 вправо на 6. Итого $6 \cdot 1 + 2 \cdot (-4) + 6 \cdot 7 + 2 \cdot (-8) + 6 \cdot 9 = 78$

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	3 3 1 -2 3 -4 5 -6 7 -8 9	78
---	------------------------------------	----

/*

Задача: 915. Кубик - 2

Решение: динамическое программирование, двумерное дп, $O(n*m)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

*/

```
#include <bits/stdc++.h>
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::vector<vvi> vvvi;
const int INF = (int)1e9+7;
template<typename T> std::istream& operator>>(std::istream& is, std::vector<T>& vec) {
    for (auto &it : vec) { is >> it; }
    return is;
}
template<typename T1, typename T2>
T1& remax(T1& a, const T2& b) {
    return a = (a >= b ? a : b);
}
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int n, m; std::cin >> n >> m;
    vvi a(n, vi(m));
    std::cin >> a;
    vvvi max(7, vvi(n, vi(m, -INF)));
    for (int side = 1; side <= 6; ++side) {
        max[side][0][0] = a[0][0] * side;
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            for (int curr = 1; curr <= 6; ++curr) {
                for (int next = 1; next <= 6; ++next) {
                    if (j+1 < m && next != curr && next + curr != 7) {
                        remax(max[next][i][j+1], a[i][j+1] * next + max[curr][i][j]);
                    }
                    if (i+1 < n && next != curr && next + curr != 7) {
                        remax(max[next][i+1][j], a[i+1][j] * next + max[curr][i][j]);
                    }
                }
            }
        }
    }
    int res = -INF;
    for (int s = 1; s <= 6; ++s) {
        remax(res, max[s].back().back());
    }
    std::cout << res << std::endl;
    return 0;
}
```


ЗАДАЧА №982

Шашки - 3

(Время: 1 сек. Память: 16 Мб Сложность: 54%)

Сема — любитель игр на шахматной доске. Больше всего он любит играть в шашки. Сема решил написать программу, которая будет играть в шашки. По его задумке, она будет показывать, какие шашки можно взять на данном ходу. Ваша задача — реализовать эту функцию.

Напомним, что взять можно только по диагонали одним из четырех способов:



После того, как белая шашка перемещается на пустое поле, черная шашка снимается с доски и считается взятой. При этом, если после перемещения белой шашки, у нее вновь появляется возможность взять, то она продолжает свой ход. Аналогичны правила и для черных шашек.

Отметим, что в рассматриваемом варианте игры в шашки отсутствует понятие «дамка», то есть возможности шашки по взятию не зависят от того, доходила она до последней горизонтали или нет.

Входные данные

В первых восьми строках входного файла INPUT.TXT записаны по восемь символов из множества {«.», «В», «W»}, которые обозначают пустое поле, черную шашку и белую шашку соответственно. Во входном файле не более 12 шашек каждого цвета. Все шашки расположены либо на черных, либо на белых полях.

Выходные данные

В выходной файл OUTPUT.TXT выдайте поля, на которых стоят шашки, которые можно взять, если ходят белые или черные. Используйте формат вывода аналогичный примерам.

Отсортируйте поля сначала по первой координате (измеряется по вертикали), а при равенстве первых — по второй (измеряется по горизонтали). Учитывайте, что первый символ первой строки входного файла соответствует полю (1, 1), последний символ первой строки — полю (1, 8), первый символ последней строки — полю (8, 1), последний символ последней строки — (8, 8).

Примеры

Пояснение

В первом примере никакая шашка не может брать.

Во втором примере белая шашка, стоящая на (4, 1) может взять сначала черную шашку на поле (5, 2) и, переместившись на поле (6, 3), взять черную шашку, стоящую на поле (5, 4). Также с поля (6, 3) можно взять еще одну черную шашку, стоящую на поле (7, 4), при этом придется двигаться другим путем. Черные же могут взять лишь белую шашку, которая стоит на поле (4, 3).

№	INPUT.TXT	OUTPUT.TXT
1	.W.W.W.W W.W.W.W. .W.W.W.W B.B.B.B. .B.B.B.B B.B.B.B.	White: 0 Black: 0
2	.W.W.W.W W.W.W.W.W.. W.W...W. .B.B.... B...B.B. .B.B.B.B B.B...B.	White: 3 (5, 2), (5, 4), (7, 4) Black: 1 (4, 3)

/*

Задача: 982. Шашки - 3

Решение: графы, поиск в глубину, шахматная доска, $O(n^4)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

*/

```
#include <iostream>
#include <string>
#include <vector>
#include <functional>
#include <set>
#include <map>

typedef std::vector<std::string> vs;
typedef std::vector<bool> vb;
typedef std::vector<vb> vvb;

int main() {
    vs field(8);
    for (int i = 0; i < 8; ++i) {
        std::cin >> field[i];
    }
    typedef std::pair<int,int> pii;
    std::map<std::string, std::set<pii>> answ;
    for (int r = 0; r < 8; ++r) {
        for (int c = 0; c < 8; ++c) {
            if (field[r][c] == '.') { continue; }
            vvb visited(8, vb(8, false));
            std::function<void(int,int)> visit = [&](const int row, const int col) {
                if (visited[row][col]) return;
                visited[row][col] = true;
                for (int dr = -2; dr <= 2; ++dr) {
                    for (int dc = -2; dc <= 2; ++dc) {
                        if (dr * dr + dc * dc != 8) { continue; }
                        int nr = row + dr;
                        int nc = col + dc;
                        int mr = row + dr / 2;
                        int mc = col + dc / 2;
                        if (nr < 0 || nr >= 8 || nc < 0 || nc >= 8) { continue; }
                        if (field[nr][nc] != '.') { continue; }
                        if (field[mr][mc] == 'W' + 'B' - field[r][c]) {
                            answ[field[r][c] == 'W' ? "White" : "Black"].insert(pii(mr+1, m
                                visit(nr, nc);
                        }
                    }
                }
            }
        }
    }
}
```

```

        }
    }
    };
    visit(r,c);
}
}
for (auto team : vs{"White", "Black"}) {
    printf("%s: %d\n", team.c_str(), (int)answ[team].size());
    std::string out;
    for (auto p : answ[team]) {
        out += "(" + std::to_string(p.first) + ", " + std::to_string(p.second) + "), ";
    }
    if (!out.empty()) {
        out.pop_back();
        out.pop_back();
        std::cout << out << std::endl;
    }
}
return 0;
}

```

ЗАДАЧА №1187

НОД с изменением элемента

(Время: 1 сек. Память: 32 Мб Сложность: 54%)

Требуется реализовать эффективную структуру данных, позволяющую изменять элементы массива $A[1..N]$ и вычислять наибольший общий делитель (НОД) для элементов отрезка $[L, R]$.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N – размер массива ($N \leq 10^5$). Во второй строке записаны N натуральных чисел – элементы массива, целые числа, не превосходящие 10^9 . Третья строка содержит натуральное число M – количество запросов ($M \leq 30\,000$). Каждая из следующих M строк представляет собой описание запроса. Сначала вводится одна буква, кодирующая вид запроса («g» – вычислить НОД, «u» – обновить значение элемента). Следом за «g» идут два числа L и R – номера левой и правой границы отрезка. Следом за «u» записаны два числа I и X – номер элемента и его новое значение, не превосходящее 10^9 ($1 \leq L \leq R \leq N, 1 \leq I \leq N$).

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса на вычисление НОД выведите результат. Все числа следует выводить в одну строку, разделяя пробелом.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 2 8 4 16 12 5 g 1 5 g 4 5 u 3 32 g 2 5 g 3 3	2 4 4 32

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <climits>

int gcd(int a, int b) {
    while (b != 0) {
        int rem = a % b;
        a = b;
        b = rem;
    }
    return a;
}

struct DataStruct {
    std::vector<int> arr, gcd;

    const int GSIZE = 128;
```

```

DataStruct(int size = 0, int item = 0) {
    arr.assign(size, item);
    gcd.assign(size / GSIZE+1, item);
}

void set(int p, int v) {
    arr[p] = v;
    const int g = p / GSIZE;
    gcd[g] = v;
    const int begin = g * GSIZE;
    const int after = std::min(begin + GSIZE, (int)arr.size());
    for (int i = begin; i < after; ++i) {
        gcd[g] = ::gcd(arr[i], gcd[g]);
    }
}

int get(int l, int r) {
    const int gl = l / GSIZE;
    const int gr = r / GSIZE;
    int answ = arr[l];
    if (gl == gr) {
        for (int i = l; i <= r; ++i) {
            answ = ::gcd(answ, arr[i]);
        }
    } else {
        for (int i = l, after = (gl+1)*GSIZE; i < after; ++i) {
            answ = ::gcd(answ, arr[i]);
        }
        for (int g = gl+1; g < gr; ++g) {
            answ = ::gcd(answ, gcd[g]);
        }
        for (int i = gr * GSIZE; i <= r; ++i) {
            answ = ::gcd(answ, arr[i]);
        }
    }
    return answ;
}

};

int main() {
    int n;
    scanf("%d", &n);
    DataStruct ds(n, 0);
    for (int i = 0; i < n; ++i) {
        int value;
        scanf("%d", &value);
        ds.set(i, value);
    }
    int q;
    scanf("%d", &q);
    while (q--) {
        char t; int l, r;
        scanf(" %c %d %d", &t, &l, &r);
        if (t == 'u') {
            ds.set(l-1, r);
        } else {
            printf("%d ", ds.get(l-1, r-1));
        }
    }
    return 0;
}

```

ЗАДАЧА №629

Сочетания

(Время: 1 сек. Память: 16 Мб Сложность: 55%)

Петя выписал все сочетания из N первых английских букв по K букв. В каждом сочетании он выписывал буквы в лексикографическом порядке. Сочетания он выписывал в лексикографическом порядке по одному в строке. Теперь он хочет узнать: какое слово записано в M-ой строке.

Входные данные

Во входном файле INPUT.TXT записаны целые числа N, K, M ($1 \leq N \leq 26$, $1 \leq K \leq N$). Гарантируется, что M не превосходит количества всех выписанных сочетаний.

Выходные данные

В выходной файл OUTPUT.TXT выведите M-ое выписанное сочетание.

Пример

Пояснение

Все сочетания в порядке их записи: ab, ac, ad, bc, bd, cd. Здесь 3м по счету сочетанием является ad.

№	INPUT.TXT	OUTPUT.TXT
1	4 2 3	ad

```
/*
    Задача: 629. Сочетания

    Решение: комбинаторика, биномиальные коэффициенты, сочетания, O(n^2)

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/

#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
#include <functional>

typedef long long ll;

int main() {

    std::vector<std::vector<ll>> bin(27, std::vector<ll>(27, 1));
    for (int n = 2; n < 27; ++n) {
        for (int k = 0; k <= n; ++k) {
            bin[n][k] = (k > 0 ? bin[n-1][k-1] : 0) + (k <= n-1 ? bin[n-1][k] : 0);
        }
    }

    std::function<void(int,int,int,ll)> solve = [&bin, &solve](int s, int n, int len, ll ne
        if (len == 0 || need == 0 || s > n) {
            return;
```

```

    }
    if (bin[n-s][len-1] >= need) {
        std::cout << char('a'+s-1);
        solve(s+1, n, len-1, need);
    } else {
        solve(s+1, n, len, need - bin[n-s][len-1]);
    }
};

int n, len; ll need;
std::cin >> n >> len >> need;
solve(1,n,len,need);
return 0;
}

```

ЗАДАЧА №661

«Стабильный» интернет

(Время: 1 сек. Память: 16 Мб Сложность: 55%)

Закончив университет, Петя решил заняться разработкой web-приложений. Получив первый заказ, Петя посчитал разумным на время выполнения заказа обеспечить себя доступом в Интернет. Провайдер, с которым Петя заключил контракт, обеспечивает доступ в Интернет только посредством сервис-карт. Когда Петя пришел в местное почтовое отделение за их покупкой, то оказалось, что в продаже имеется N сервис-карт.

Каждая сервис-карта характеризуется тремя числами B_i, E_i, S_i , где S_i – цена карты, а B_i и E_i – это начало и окончание промежутка времени её действия, т.е. сервис-карта позволяет получить доступ в Интернет в любой момент времени t такой, что $B_i \leq t \leq E_i$. Время отсчитывается от некоторого фиксированного момента в прошлом.

Петя решил, что будет работать над выполнением заказа, начиная с момента времени B и заканчивая моментом времени E . Основная проблема состоит в том, что Петя не богат, поэтому он хочет на время выполнения заказа обеспечить себя «стабильным» Интернетом и потратить на это как можно меньшую сумму денег. Будем говорить, что данное множество сервис-карт обеспечивает Пете «стабильный» Интернет, если для любого момента времени t , такого, что $B \leq t \leq E$, найдется хотя бы одна сервис-карта из данного множества, которая позволяет получить доступ в Интернет в этот момент времени.

Ваша задача состоит в том, чтобы определить минимальную сумму денег, необходимую Пете для приобретения множества карт, которое обеспечит ему «стабильный» Интернет на время выполнения заказа.

Входные данные

В первой строке входного файла INPUT.TXT находится одно число N . Во второй строке два числа B и E , разделенные пробелом, где B – момент начала промежутка времени, в который Петя будет работать над выполнением заказа, а E – момент окончания.

Следующие N строк описывают сервис-карты, i -ая из этих строк описывает i -ю сервис-карту и содержит три числа B_i, E_i и S_i , разделенные одиночными пробелами.

Ограничения: все числа натуральные, $1 \leq N \leq 10^5$, $1 \leq B_i < E_i \leq 10^9$, $1 \leq B < E \leq 10^9$, $1 \leq S_i \leq 20\,000$. Существует хотя бы одно множество карт, которое обеспечивает «стабильный» Интернет на время выполнения заказа

Выходные данные

Выходной файл OUTPUT.TXT должен содержать одно число, равное минимальной сумме денег, необходимой Пете для приобретения множества сервис-карт, которое обеспечит ему «стабильный» Интернет на время выполнения заказа.

Пример

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	7	49
	10 30	
	9 14 10	
	13 19 18	
	14 18 16	
	18 24 14	
	24 30 9	
	17 2005 24	
	15 20 14	

```

/*
Задача: 661. «Стабильный» интернет

Решение: std::priority_queue, динамическое программирование, O(n*log(n))

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define isz(x) (int)(x).size()
#define all(x) (x).begin(), (x).end()
#define unique(x) (x).erase(std::unique(all(x)), (x).end())
#define lowpos(y, x) int(std::lower_bound(all(y), (x)) - (y).begin())

typedef std::vector<int> vi;
typedef std::pair<int, int> pii;

struct Card {
    int lt, rt, cost;
};

struct GreaterByCost {
    bool operator()(const Card& a, const Card& b) {
        return a.cost > b.cost;
    }
};

int solve(int L, int R, vi lt, vi rt, vi cost) {
    // Сжимаем координаты:
    vi x{0, L, L+1, R-1, R};
    for (int l : lt) { x.insert(x.end(), {std::max(L, l-1), l, std::min(l+1, R)}); }
    for (int r : rt) { x.insert(x.end(), {std::max(L, r-1), r, std::min(r+1, R)}); }
    std::sort(all(x));
    unique(x);
    L = lowpos(x, L);
    R = lowpos(x, R);
    for (int& coord : lt) { coord = lowpos(x, coord); }
    for (int& coord : rt) { coord = lowpos(x, coord); }
    assert(L == 1);
    // Создаем события на вход в отрезок действия карты:
    std::vector<pii> needOpen;
    for (int i = 0; i < isz(cost); ++i) {
        needOpen.push_back(pii(lt[i], i));
    }
    std::sort(all(needOpen));
    std::reverse(all(needOpen));
    // Динамическое программирование:
    std::priority_queue<Card, std::vector<Card>, GreaterByCost> opened;
    vi dp(isz(x));
    for (int i = 1; i < isz(x); ++i) {
        while (isz(needOpen) > 0 && needOpen.back().first == i) {
            auto back = needOpen.back();
            needOpen.pop_back();
            int id = back.second;
            opened.push(Card{lt[id], rt[id], cost[id]+dp[i-1]});
        }
        while (isz(opened) > 0 && opened.top().rt < i) {

```

```

        opened.pop();
    }
    assert(isz(opened) > 0);
    dp[i] = opened.top().cost;
}
return dp[R];
}
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (int n; std::cin >> n; ) {
        int L, R; std::cin >> L >> R;
        vi lt, rt, cost;
        for (int i = 0; i < n; ++i) {
            int l, r, c; std::cin >> l >> r >> c;
            l = std::max(l, L);
            r = std::min(r, R);
            if (l <= r) {
                lt.push_back(2*l);
                rt.push_back(2*r);
                cost.push_back(c);
            }
        }
        std::cout << solve(2*L, 2*R, lt, rt, cost) << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №726

Остатки

(Время: 1 сек. Память: 16 Мб Сложность: 55%)

Саша устроился программистом в одну серьезную компанию. Однажды к нему зашел начальник и сказал: «Саша, что за бред выдает твоя программа? В одном месте у нее получается, что число $3 \cdot n$ - нечетное, а в другом - что число $5 \cdot n$ делится на 10. Ну и как такое может быть? У тебя что, по математике в школе было?». По математике у Саши была твердая тройка, поэтому, чтобы больше не попадать в такие неприятные ситуации, он просит вас написать программу, проверяющую, может ли число $a \cdot n$ давать остаток b по модулю c , и в то же время число $d \cdot n$ давать остаток e по модулю f .

Входные данные

Входной файл INPUT.TXT содержит шесть целых чисел: a, b, c, d, e и f ($1 \leq a, c, d, f \leq 10^9$, $0 \leq b \leq c$, $0 \leq e \leq f$).

Выходные данные

Выведите в выходной файл OUTPUT.TXT строку «YES», если это возможно и «NO» в противном случае.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 1 2 5 0 10	NO
2	3 1 2 5 5 10	YES

```
/*
    "Остатки": система диофантовых уравнений, расширенный алгоритм Евклида,  $O(\log(n))$ 
*/
```

```
#include <iostream>
#include <vector>
#include <cassert>
```

```
typedef long long ll;
```

```
bool gcd(ll a, ll b, ll c, ll& x, ll& y) {
    if (a < b) {
        return gcd(b, a, c, y, x);
    }
    if (c == 0) {
        x = b;
        y = -a;
        return true;
    }
    if (b == 0) {
        x = c;
        y = 0;
        return true;
    }
    std::vector<ll> r{a, b}, q;
    while (b != 0 && a % b != 0) {
        q.push_back(a / b);
```

```

        r.push_back(a % b);
        a = b;
        b = r.back();
    }
    assert(r.back() != 0);
    if (c % r.back() != 0) {
        return false;
    }

    x = 0, y = 1;
    while (!q.empty()) {
        ll z = -q.back() * y + x;
        x = y;
        y = z;
        q.pop_back();
    }
    x *= c / r.back();
    y *= c / r.back();
    return true;
}

int main() {
    bool flag = true;
    ll a1, b1, c1, x1, y1;
    std::cin >> a1 >> b1 >> c1;

    b1 %= c1;
    flag = gcd(a1, -c1, b1, x1, y1);
    if (!flag) {
        std::cout << "NO";
        return 0;
    }
    assert(a1 * x1 - c1 * y1 == b1);

    ll a2, b2, c2, x2, y2;
    std::cin >> a2 >> b2 >> c2;

    b2 %= c2;
    flag = gcd(a2, -c2, b2, x2, y2);
    if (!flag) {
        std::cout << "NO";
        return 0;
    }
    assert(a2 * x2 - c2 * y2 == b2);

    ll x, y;
    flag = gcd(c1, c2, x2-x1, x, y);
    if (!flag) {
        std::cout << "NO";
        return 0;
    }
    std::cout << "YES";
    return 0;
}

```

ЗАДАЧА №870

URL Validator

(Время: 1 сек. Память: 16 Мб Сложность: 55%)

Для идентификации ресурсов в сети Internet используются URL (Uniform Resource Locator). URL состоит из нескольких элементов: протокол, хост, порт, путь и файл. Некоторые элементы URL могут быть опущены. Рассмотрим упрощенный формат URL:

[http://]host[:port][/path][/file]

Заключенные в квадратные скобки элементы могут быть опущены, то есть, например, можно не указать протокол или файл. Элемент host представляет собой либо IP-адрес – четыре целых числа без лидирующих нулей от 0 до 255, разделенные точкой (например, 212.193.39.146), либо строковое имя ресурса. Во втором случае имя имеет вид prefix.domain, либо это просто имя компьютера.

В первом случае prefix это последовательность одного или более слов, разделенных точкой, а domain – слово из английских букв длиной 2 или 3 символа. А в случае, если host является просто именем компьютера, то host – это одно слово.

Элемент port – это целое число от 0 до 65535 без лидирующих нулей.

Элемент path – это последовательность одного или более слов, разделенных символом «/» (код 47).

Элемент file – это ноль или более слов, разделенных символом точка «.» (код 46).

Слово – это последовательность из одного или более символов. Если не оговорено специально, то допустимыми символами слова считаются английские буквы произвольного регистра, цифры и символ подчеркивание «_» (код 95).

Входные данные

Входной файл INPUT.TXT содержит не более 10000 строк. Все строки содержат символы с кодами от 33 до 127 включительно. Длина каждой строки не превосходит 1000 символов. Размер файла не превосходит 500Кб. Помните, что любой тест (как и любой корректный тестовый файл) заканчивается символом перевода строки.

Выходные данные

В выходной файл OUTPUT.TXT для каждой из строк выведите в отдельной строке «YES», если строка представляет корректную запись URL, либо «NO» в противном случае.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	http://acm.sgu.ru/index.html	YES
	212.193.39/index.jsp	NO
	http://acm.sgu.ru/01/index.php	YES
	212.193.39.146/start/index.jsp	YES

Решение: разбор выражений, синтаксический парсер, строки, $O(n)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

```
*/

#include <iostream>
#include <string>
#include <cassert>
#include <vector>

std::string solve(const std::string& s) {
    int p = 0;
    auto isDomain = [&](const std::string& t) {
        if (t.size() < 2u || t.size() > 3u) { return false; }
        for (auto it : t) {
            if (! (('A' <= it && it <= 'Z') || ('a' <= it && it <= 'z'))) {
                return false;
            }
        }
        return true;
    };
    auto isIp = [&](const std::vector<std::string>& v) {
        if (v.size() != 4u) { return false; }
        for (const auto& it : v) {
            if (it.size() > 1u && it.front() == '0') { return false; }
            for (auto jt : it) {
                if (jt < '0' || jt > '9') { return false; }
            }
            if (it.size() == 0u || it.size() > 3u || std::stoi(it) < 0 || std::stoi(it) > 255) {
                return false;
            }
        }
        return true;
    };
    auto readWord = [&]() {
        std::string ret;
        while (p < (int)s.size() && (
            ('0' <= s[p] && s[p] <= '9') ||
            ('a' <= s[p] && s[p] <= 'z') ||
            ('A' <= s[p] && s[p] <= 'Z') ||
            (s[p] == '_')))
        {
            ret.push_back(s[p++]);
        }
        return ret;
    };
    auto readHost = [&]() {
        std::vector<std::string> words;
        if (p < (int)s.size() && s[p] == '.') { return false; }
        while (! (p == (int)s.size() || s[p] == ':' || s[p] == '/')) {
            if (s[p] == '.') { ++p; }
            auto word = readWord();
            if (word.empty()) { return false; }
            words.push_back(word);
        }
        if (words.size() == 1u) { return true; }
        return ((words.size() > 0u && isDomain(words.back())) || isIp(words));
    };
    if (!readHost()) { return "NO"; }
    if (! (p == (int)s.size() || s[p] == ':' || s[p] == '/')) {
        return "NO";
    }
    auto readPort = [&]() {
        if (p == (int)s.size() || s[p] != ':') {
            return true;
        }
        assert(s[p] == ':');
        ++p;
    };
}
```

```

std::string it;
while (p < (int)s.size() && '0' <= s[p] && s[p] <= '9') {
    it.push_back(s[p++]);
}
if (it.size() > 1u && it.front() == '0') {
    return false;
}
for (auto jt : it) {
    if (jt < '0' || jt > '9') {
        return false;
    }
}
if (it.size() == 0u || it.size() > 5u || std::stoi(it) < 0 || std::stoi(it) > 65535)
    return false;
}
return true;
};
if (!readPort()) { return "NO"; }
auto readPath = [&]() {
    if (p == (int)s.size() || s[p] != '/') {
        return true;
    }
    assert(s[p] == '/');
    while (p < (int)s.size() && s[p] == '/') {
        ++p;
        auto word = readWord();
        if (word.empty()) { return true; }
    }
    while (p < (int)s.size() && s[p] == '.') {
        ++p;
        auto word = readWord();
        if (word.empty()) { return false; }
    }
    return true;
};
if (!readPath()) { return "NO"; }
return p == (int)s.size() ? "YES" : "NO";
}

int main() {
    std::string s;
    while (std::getline(std::cin, s)) {
        if (s.size() >= 7u && s.substr(0,7) == "http://") {
            s.erase(0, 7);
        }
        std::cout << solve(s) << "\n";
    }
    return 0;
}

```

ЗАДАЧА №1184

Range Maximum Query

(Время: 2 сек. Память: 32 Мб Сложность: 55%)

Задан числовой массив $A[1..N]$. Необходимо выполнить M операций вычисления максимального элемента и его индекса на отрезке $[L, R]$.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N – размер массива ($N \leq 10^5$). Во второй строке записаны N целых чисел – элементы массива, целые числа, не превосходящие 10^9 по абсолютной величине. Третья строка содержит натуральное число M – количество RMQ-запросов ($M \leq 10^5$). Следующие M строк содержат пары натуральных чисел L и R ($L \leq R \leq N$), описывающие отрезки.

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса в отдельной строке выведите значение максимума и его индекс через пробел. Если существует несколько элементов отрезка, равных максимальному, то следует выводить наименьший индекс.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	5	
	7 3 1 6 4	7 1
	3	6 4
	1 5	1 3
	2 4	
	3 3	
2	2	
	0 1	0 1
	3	1 2
	1 1	1 2
	1 2	
	2 2	

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <climits>

struct DataStruct {
    std::vector<int> arr, max, pos;

    const int GSIZE = 258;

    DataStruct(int size = 0, int item = 0) {
        arr.assign(size, item);
        max.assign((size+GSIZE-1) / GSIZE, item);
        pos.assign((size+GSIZE-1) / GSIZE, 0);
    }
};
```



```

        for (int g = 0; g < (int)pos.size(); ++g) {
            pos[g] = g * GSIZE;
        }
    }

    void set(int p, int v) {
        arr[p] = v;
        const int g = p / GSIZE;
        max[g] = v;
        pos[g] = p;
        const int begin = g * GSIZE;
        const int after = std::min(begin + GSIZE, (int)arr.size());
        for (int i = begin; i < after; ++i) {
            if (arr[i] > max[g]) {
                max[g] = arr[i];
                pos[g] = i;
            } else if (arr[i] == max[g]) {
                pos[g] = std::min(pos[g], i);
            }
        }
    }

    std::pair<int,int> get(int l, int r) {
        const int gl = l / GSIZE;
        const int gr = r / GSIZE;
        std::pair<int, int> answ = {arr[l], l};
        if (gl == gr) {
            for (int i = l; i <= r; ++i) {
                if (arr[i] > answ.first) {
                    answ = {arr[i], i};
                }
            }
        } else {
            for (int i = l, after = (gl+1)*GSIZE; i < after; ++i) {
                if (arr[i] > answ.first) {
                    answ = {arr[i], i};
                }
            }
            for (int g = gl+1; g < gr; ++g) {
                if (max[g] > answ.first) {
                    answ = {max[g], pos[g]};
                }
            }
            for (int i = gr * GSIZE; i <= r; ++i) {
                if (arr[i] > answ.first) {
                    answ = {arr[i], i};
                }
            }
        }
        return answ;
    }
};

int main() {
    int n;
    scanf("%d", &n);
    DataStruct ds(n, 0);
    for (int i = 0; i < n; ++i) {
        int value;
        scanf("%d", &value);
        ds.set(i, value);
    }
    int q;
    scanf("%d", &q);
    while (q--) {
        int l, r;
        scanf("%d %d", &l, &r);
        --l, --r;
    }
}

```

```
        auto answ = ds.get(l, r);
        printf("%d %d\n", answ.first, answ.second+1);
    }
    return 0;
}
```

ЗАДАЧА №1440

За стол!

(Время: 2 сек. Память: 16 Мб Сложность: 55%)

В известном во всём городе Центре Приличия и Самоконтроля каждую субботу проходят мастер-классы, на которых посетители тренируются быть скромными, учатся держать себя в руках и не говорить первое, что приходит в голову, контролируя свои эмоции.

Как это обычно бывает, после занятий все пришедшие садятся за круглый стол, который подготовила хранительница центра. Садится каждый, иначе это покажется неприличным. По краю стола расположены N разнообразных блюд, каждое блюдо имеет свою калорийность C_i . Когда гость садится за стол, ему достаются какие-то расположенные подряд блюда. Гости должны сесть так, чтобы не обидеть друг друга, то есть сумма калорийностей выбранных блюд должна у всех совпадать. И при этом не обидеть хранительницу, съев все блюда на столе.

К сожалению, это не всегда возможно, и если гости не смогут рассадиться по заданным критериям, они могут потерять контроль над собой и забудут про все правила приличия! Известно, что сегодня придёт от 1 до N человек. Хранительница попросила Вас написать программу, которая сообщит: для каких количеств такая рассадка будет возможна, а для каких нет.

Входные данные

В первой строке входного файла INPUT.TXT содержится натуральное число N – количество блюд на столе ($1 \leq N \leq 10^5$).

Во второй строке перечислены калорийности блюд C_i в порядке обхода стола по часовой стрелке.

Калорийность – неотрицательное целое число, сумма всех калорийностей не превосходит 10^9 .

Выходные данные

В выходной файл OUTPUT.TXT выведите строку из N символов: K -ый символ ($1 \leq K \leq N$) должен быть равен «1», если возможно рассадить K гостей за столом и «0» – иначе.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 3 2 2 1	1100
2	6 1 1 1 1 1 1	111001
3	5 1 0 0 1 1	10100

```
#include <stdio.h>
#include <vector>
#include <string>
#include <functional>
#include <cassert>
#include <cstdlib>
```

```

inline int prev(int p, int n) {
    return p == 0 ? n-1 : p-1;
}

inline int next(int p, int n) {
    return p == n-1 ? 0 : p+1;
}

std::string solve(const std::vector<int>& a) {
    const int n = a.size();

    // Считаем префикс-суммы:
    std::vector<int> pref{0};
    for (int i = 0; (int)pref.size() <= 2*n; i = next(i, n)) {
        pref.push_back(pref.back() + a[i]);
    }
    int sum = pref[n];

    // Обрабатываем особый случай:
    if (sum == 0) {
        return std::string(n, '1');
    }

    // Функция поиска разбиения массива на number отрезков с одинаковой суммой. Возвращает
    std::function<bool(int)> is_possible = [&](int number) {
        if (number == 1) {
            return true;
        }

        const int need = sum / number; // Нужная сумма каждого отрезка
        assert(need * number == sum);

        // Находим отрезок [0, r] такой, что sum[0,r] <= need, sum[0,r+1] > need
        int l = 0, r = -1, curr = 0;
        while (curr + a[next(r,n)] <= need) {
            curr += a[next(r,n)];
            r = next(r,n);
        }

        // Сдвигаем отрезок [l, r], получая все возможные отрезки, на которых сумма равна n
        while (r >= 0) {
            // Сдвигаем левую границу пока не накопили нужную сумму:
            while (curr + a[prev(l,n)] <= need) {
                curr += a[prev(l,n)];
                l = prev(l,n);
            }
            if (curr == need) {
                // Пробуем построить еще number-1 отрезков, сумма на которых равна need:
                int low = r+1, prev = r, count = number-1;
                while (count > 0) {
                    if (pref[low+1] - pref[low] > need) {
                        break;
                    }
                    // [low, high)
                    int high = l == 0 ? n : l;
                    while (high-low > 1) {
                        int mid = (low + high) / 2;
                        if (pref[mid+1] - pref[prev+1] > need) {
                            high = mid;
                        } else {
                            low = mid;
                        }
                    }
                    if (pref[low+1]-pref[prev+1] == need) {
                        --count;
                        prev = low++;
                    } else {
                        break;
                    }
                }
            }
            curr -= a[r];
            r = prev(r,n);
        }
    };

    if (!is_possible(number)) {
        return "No";
    }
    return "Yes";
}

```

```

        }
    }
    if (count == 0) {
        return true;
    }
}
// Сдвигаем правую границу:
curr -= a[r];
--r;
}
return false;
};

std::string answ(n, '0');
// Перебираем все делители суммы элементов массива:
for (int i = 1; i * i <= sum; ++i) {
    const int j = sum / i;
    if (j * i == sum) {
        if (i <= n) {
            assert(answ[i-1] == '0');
            answ[i-1] += is_possible(i);
        }
        if (j <= n && j != i) {
            assert(answ[j-1] == '0');
            answ[j-1] += is_possible(j);
        }
    }
}
assert(answ[0] == '1');
return answ;
}

int main() {
    int n;
    scanf("%d", &n);
    std::vector<int> a(n);
    for (auto& it : a) {
        scanf("%d", &it);
    }
    printf("%s\n", solve(a).c_str());
    return 0;
}

```

ЗАДАЧА №445

Покупки

(Время: 1 сек. Память: 16 Мб Сложность: 56%)

Во многих фирмах, занимающихся торговлей, существует должность менеджера по закупкам. Как известно, они занимаются тем, что по торговому плану, представляющему собой список наименований товаров, для каждого из которых указано необходимое количество, закупает указанные в нем товары на оптовых базах. Торговый план при этом составляется руководством компании. Иногда у менеджеров по закупкам возникает желание принести выгоду не только своей фирме, но и себе.

Только что, как раз после подписания очередного торгового плана на заказ n наименований товаров, открылась новая оптовая база. Как это часто бывает сразу после открытия, ее цены на многие товары ниже заложенных в план. Наверное, этим можно воспользоваться.

На закупку товаров были выделены деньги из расчета того, что все товары будут закупаться на старой оптовой базе. Менеджер хочет, воспользовавшись возможностью покупать товары на новой базе, потратить как можно меньше денег на закупку требуемого количества товаров (не потраченные деньги он, конечно, сможет забрать себе).

Чтобы не вызывать сильных подозрений, производить на новой базе закупки, суммарная запланированная стоимость которых была больше, чем d денежных единиц, не следует. Осталось только рассчитать, какие товары и в каком количестве следует закупать на новой базе, чтобы осталось как можно больше не потраченных денег.

Входные данные

Первая строка содержит четыре числа: n ($1 \leq n \leq 1000$), d , а так же k_1 и k_2 ($1 \leq k_1, k_2 \leq 1000$) - количества наименований товаров, имеющих на открытых ранее и новой базе соответственно. После этого идут n строк, каждая из которых содержит название товара в плане и его количество (положительное вещественное число). За ними следуют два блока из k_1 и k_2 строк соответственно, отделенные от предыдущего и разделенные между собой переводом строки - наименования товаров на базах и цены за единицу товара соответственно. Все цены являются положительными числами, даже на новой базе.

Названия товаров состоят из не более, чем 100 английских букв и символов подчеркивания, при этом регистр букв не учитывается. Вещественные числа заданы не более чем с двумя знаками после десятичной точки и по величине не превосходят 10^6 . Гарантируется, что все товары из плана можно купить на старой базе. Считайте, что любой товар на любой базе можно покупать в любом дробном количестве, а также что при оплате можно использовать любое дробное количество денежных единиц.

Выходные данные

В выходной файл OUTPUT.TXT выведите n вещественных чисел, по одному на строке, задающих количество соответствующего товара, закупаемого на новой базе. На i -ой строке выведите количество товара, идущего i -ым в плане. Ошибки менее 0.01 будут игнорироваться.

Пример


```

    }
}

// Сортируем в порядке возрастания отношений "новая цена" / "старая цена":
std::stable_sort(records.begin(), records.end(), [](const Record& a, const Record& b) {
    return 1.0L * a.price2 * b.pricel < 1.0L * b.price2 * a.pricel;
});

// Набираем жадно самые выгодные товары до предела:
std::vector<double> answer(n, 0);
double sum = 0;
for (auto& r : records) {
    if (r.pricel < r.price2) break;
    if (limit - sum >= r.need * r.pricel) {
        answer[r.id] = r.need;
    } else {
        answer[r.id] = (limit - sum) / r.pricel;
        break;
    }
    sum += answer[r.id] * r.pricel;
}
for (auto& it : answer) {
    printf("%.4f\n", it);
}

return 0;
}

```


ЗАДАЧА №636

Java Challenge

(Время: 2 сек. Память: 16 Мб Сложность: 56%)

Все участники олимпиад знают, что во время соревнования на счету каждая секунда. Иногда даже время, которое в суе затрачивается на переключение между окнами может оказаться критичным. В таких соревнованиях, как Java Challenge, количество окон может быть довольно большим (Java Challenge - это соревнование, проходящее в рамках финала чемпионата мира по программированию среди студентов. Оно состоит в разработке искусственного интеллекта для управления виртуальным роботом).

В данной задаче мы будем считать, что этот процесс выполняется следующим способом. В системе хранится циклический список открытых окон. При нажатии определенной комбинации клавиш k раз можно перейти в этом списке на k позиций в одну или в другую сторону. Кроме того, окна, относящиеся к каждому приложению так же организованы в циклический список. По этому списку также можно перемещаться в обе стороны, для перемещения на k позиций так же требуется k нажатий клавиш. При этом после своей активизации окно перемещаются в позицию перед первым элементом общего списка окон. Напишите программу, которая для каждого из окон будет определять минимальное количество нажатий клавиш, которое нужно затратить для его активизации. До начала выполнения операции активным является первое окно.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n ($1 \leq n \leq 50000$) - количество открытых окон. Следующие n строк описывают окна в том порядке, в котором они идут в списке. Для каждого из окон задается номер приложения, которому соответствует это окно, и его номер в циклическом списке окон этого приложения.

Выходные данные

На единственной строке выходного файла OUTPUT.TXT для каждого окна выведите минимальное количество нажатий клавиш, которое надо затратить для его активации.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	9	0 1 2 2 2 1 3 2 1
	3 2	
	2 2	
	2 3	
	1 3	
	2 1	
	3 1	
	4 1	
	1 2	
	1 1	

```
#include <stdio.h>
#include <vector>
#include <queue>
```

```

#include <algorithm>
#include <cassert>
#include <tuple>
#include <queue>
#include <map>

struct Pair {
    int level, pos;
};

bool operator<(const Pair& a, const Pair& b) {
    return a.level < b.level || (a.level == b.level && a.pos < b.pos);
}

struct State {
    int id, local;
};

int main() {
    int nV;
    scanf("%d", &nV);
    std::map<Pair, int> id;
    std::vector<Pair> item(nV);
    std::vector<int> item_size(nV,);
    for (int i = 0; i < nV; ++i) {
        int a, b;
        scanf("%d %d", &a, &b);
        item_size[a] = std::max(item_size[a], b);
        id[Pair{a,b}] = i;
        item[i] = Pair{a, b};
    }
    std::vector<int> dist(2*nV, -1);
    dist[0] = 0;
    std::queue<State> queue;
    queue.push(State{0, 0});
    while (!queue.empty()) {
        auto curr = queue.front(); queue.pop();
        int level = item[curr.id].level, pos = item[curr.id].pos;
        {
            Pair next; int id_next;
            next = Pair{level, pos-1};
            if (next.pos < 1) {
                next.pos = item_size[next.level];
            }
            id_next = id[next];
            if (dist[id_next+nV] == -1) {
                dist[id_next+nV] = dist[curr.id+curr.local*nV]+1;
                queue.push(State{id_next, 1});
            }
            next = Pair{level, pos+1};
            if (next.pos > item_size[next.level]) {
                next.pos = 1;
            }
            id_next = id[next];
            if (dist[id_next+nV] == -1) {
                dist[id_next+nV] = dist[curr.id+curr.local*nV]+1;
                queue.push(State{id_next, 1});
            }
        }
        if (!curr.local) {
            int id_next;
            id_next = curr.id-1;
            if (id_next < 0) {
                id_next += nV;
            }
            if (dist[id_next] == -1) {
                dist[id_next] = dist[curr.id]+1;
                queue.push(State{id_next, 0});
            }
        }
    }
}

```

```

        id_next = curr.id+1;
        if (id_next >= nV) {
            id_next -= nV;
        }
        if (dist[id_next] == -1) {
            dist[id_next] = dist[curr.id]+1;
            queue.push(State{id_next,0});
        }
    }
}
for (int i = 0; i < nV; ++i) {
    printf("%d ", std::min(dist[i], dist[i+nV]));
}
return 0;
}

```

ЗАДАЧА №737

ДНК

(Время: 2 сек. Память: 32 Мб Сложность: 56%)

Вася никогда не любил биологию. Но когда он узнал про ДНК, у него появился живой интерес. Он решил, что если все существа произошли друг от друга, то и ДНК у них должны быть похожими. У некоторых более похожие, у некоторых - менее, но у всех ДНК можно записать в виде строки, состоящей из символов А, С, G и Т. Поэтому он решил найти какой-нибудь показатель родства. И придумал следующее. Он берет из двух ДНК по подстроке. Если одна из них является анаграммой другой (т. е. получается перестановкой букв), то это хорошая пара подстрок. Естественно, в любой хорошей паре обе подстроки имеют одинаковую длину. Тогда степень родства двух ДНК – это максимально возможная длина подстрок в хорошей паре.

Входные данные

В первой строке входного файла INPUT.TXT находится ДНК Васи. А во второй строке - ДНК первого попавшегося Васе живого существа. Обе строки не пусты и состоят не более, чем из 1 300 символов А, С, G и Т.

Выходные данные

В первую строку выходного файла OUTPUT.TXT выведите степень родства Васи с подопытным существом. Если степень родства отлична от нуля, то во вторую следует вывести две начальные позиции подстрок из соответствующей хорошей пары в первой и второй ДНК соответственно. В случае неоднозначности последних двух чисел, выведите любые подходящие.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	ACGT GTC	3 2 1
2	ACGT CAT	2 1 1
3	ACA AC	2 2 1

```
/*
    Задача: 737. ДНК

    Решение: сортировка, бинарный поиск, два указателя, перебор, O(n^2*log(n))

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

int get_id(char c) {
    switch(c) {
        case 'A': return 0;
```

```

        case 'C': return 1;
        case 'G': return 2;
        case 'T': return 3;
    }
    assert(false);
    return -1;
}

struct Record {
    int len, p1, p2;
};

bool operator<(const Record& a, const Record& b) {
    return a.len < b.len || (a.len == b.len && (a.p1 < b.p1 || (a.p1 == b.p1 && a.p2 < b.p2)
}
bool operator>(const Record& a, const Record& b) {
    return b < a;
}
bool operator!=(const Record& a, const Record& b) {
    return b < a || a < b;
}
bool operator==(const Record& a, const Record& b) {
    return !(b == a);
}

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (std::string s, t; std::cin >> s >> t; ) {
        const int n = (int)s.size();
        const int m = (int)t.size();
        Record best{0,0,0};
        for (int len = 1; len <= std::min(n,m); ++len) {
            vvi fi;
            vi curr(5);
            for (int i = 0; i < len; ++i) {
                curr[get_id(s[i])]++;
            }
            fi.push_back(curr);
            for (int i = len; i < n; ++i) {
                curr.back()++;
                curr[get_id(s[i-len])]--;
                curr[get_id(s[i])]++;
                fi.push_back(curr);
            }
            std::sort(all(fi));
            curr.assign(5,0);
            for (int i = 0; i < len; ++i) {
                curr[get_id(t[i])]++;
            }
            auto equal = [](const vi& a, const vi& b, int begin, int after) {
                for (int i = begin; i < after; ++i) {
                    if (a[i] != b[i]) { return false; }
                }
                return true;
            };
            int p1 = -1, p2 = -1;
            auto search = [&]() {
                int save = curr.back();
                curr.back() = 0;
                auto it = std::lower_bound(all(fi), curr);
                curr.back() = save;
                if (it == fi.end() || !equal(*it, curr, 0, 4)) {
                    return false;
                }
                p1 = it->at(4);
                p2 = save;
                return true;
            };
            search();
            for (int i = len; i < m; ++i) {

```

```

        curr.back()++;
        curr[get_id(t[i-len])]--;
        curr[get_id(t[i])]++;
        search();
    }
    if (p1 == -1 || p2 == -1) { continue; }
    best = std::max(best, Record{len,p1,p2});

}
std::cout << best.len << std::endl;
if (best.len != 0) {
    std::cout << best.p1+1 << ' ' << best.p2+1 << std::endl;
}
}
return 0;
}

```

ЗАДАЧА №865

Проверка орфографии

(Время: 1 сек. Память: 16 Мб Сложность: 56%)

Профессор Далл разработал новую систему проверки и исправления орфографии. Расстоянием между двумя английскими буквами в одинаковом регистре называется кратчайшее расстояние между ними по алфавиту, записанному по кругу. Например, $d(B, G) = 5$, а $d(Z, A) = 1$. Расстояние между буквами разных регистров равно расстоянию между соответствующими буквами одного регистра. Например, $d(a, A) = 0$, $d(Z, a) = 1$.

Расстоянием между двумя строками одинаковой длины называется сумма расстояний между соответствующими буквами. Метод профессора основывается на словаре правильных слов и представляет введенную последовательность в виде конкатенации слов из словаря таким образом, чтобы расстояние между заданным текстом и результатом было наименьшим. Регистр букв в тексте сохраняется. Реализуйте метод Далла.

Слово из словаря может использоваться более одного раза.

Входные данные

В первой строке входного файла INPUT.TXT содержится N ($1 \leq N \leq 100$), обозначающее количество слов в словаре. Далее, в N строках содержатся слова из словаря. Длины слов от 1 до 32 символов включительно.

Последняя строка файла содержит текст. Текст состоит из английских букв, длина текста не менее 1 символа и не более 1000. Возможно, словарь содержит одинаковые слова. Слова в словаре записываются буквами произвольного регистра.

Выходные данные

В выходной файл OUTPUT.TXT выведите результат. Если решений несколько, выведите любое. Если решения не существует, выведите -1.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4 SaratoV StaTe The University SaratofStataUniversitI	SaratovStateUniversitY

```
/*
    Задача: 865. Проверка орфографии

    Решение: строки, динамическое программирование, O(n*len(text)*len(word))

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/
```

```

#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
#define isz(x) (int)(x).size()
typedef std::vector<std::string> vs;
int f(char c1, char c2) {
    c1 = std::tolower(c1);
    c2 = std::tolower(c2);
    if (c1 > c2) { std::swap(c1,c2); }
    return std::min(c2-c1,c1+26-c2);
}
int f(const std::string& fi, const std::string& se) {
    assert(isz(fi) == isz(se));
    int res = 0;
    for (int i = 0; i < isz(fi); ++i) {
        res += f(fi[i], se[i]);
    }
    return res;
}
const int INF = (int)1e9+7;
struct Record {
    int val; std::string* what;
};
bool operator<(const Record& a, const Record& b) {
    return a.val < b.val || (a.val == b.val && a.what < b.what);
}
std::string NULL_STR = "";
std::string solve(std::string s, vs& w) {
    s = "?" + s;
    std::vector<Record> dp(isz(s), Record{INF, &NULL_STR});
    dp[0].val = 0;
    for (int last = 1; last < isz(s); ++last) {
        for (std::string& what : w) {
            if (isz(what) <= last) {
                int res = dp[last-isz(what)].val;
                res += f(s.substr(last-isz(what)+1,isz(what)), what);
                res = std::min(res, INF);
                dp[last] = std::min(dp[last], Record{res, &what});
            }
        }
    }
    int res = dp.back().val;
    std::string ret;
    if (res == INF) {
        ret = "-1";
        return ret;
    }
    std::vector<std::string> answ;
    for (int last = isz(s)-1; last > 0; ) {
        answ.push_back(*dp[last].what);
        last -= isz(*dp[last].what);
    }
    std::reverse(all(answ));
    for (auto &it : answ) { ret += it; }
    s.erase(0,1);
    for (int i = 0; i < isz(ret); ++i) {
        if ('A' <= s[i] && s[i] <= 'Z') {
            ret[i] = std::toupper(ret[i]);
        } else if ('a' <= s[i] && s[i] <= 'z') {
            ret[i] = std::tolower(ret[i]);
        }
    }
    assert(res == f(ret,s));
    return ret;
}
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (int n; std::cin >> n; ) {
        vs w(n);
        for (auto &it : w) std::cin >> it;
    }
}

```



```
        std::string s; std::cin >> s;
        std::cout << solve(s, w) << std::endl;
    }
    return 0;
}
```

ЗАДАЧА №904

Crimsonland

(Время: 1 сек. Память: 16 Мб Сложность: 56%)

Гриша третий день играет в Crimsonland, он «застрял» на самом сложном уровне: Panic Rush, так как ему постоянно не хватает боеприпасов.

На уровне Panic Rush есть несколько особенностей. Персонаж Гриши вооружён плазменным дробовиком с углом атаки α и неограниченной дальностью. Все монстры, попадающие в угол атаки, при выстреле тут же погибают. Дробовик достаточно тяжёлый, переносить его нельзя, но можно быстро поворачивать вокруг своей оси на любой угол. Монстры появляются все одновременно в произвольных точках карты, при этом их местоположение не совпадает с местоположением персонажа.

Гриша нашел в интернете чит-код, и теперь он знает, где появятся монстры и какой будет угол атаки дробовика. Помогите Грише подсчитать минимальное количество выстрелов, необходимых для отражения атаки.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа N и α , где N ($1 \leq N \leq 10^4$) — количество монстров, а α ($1 \leq \alpha \leq 180$) — угол атаки дробовика в градусах. В следующей строке указано местоположение персонажа X_0 и Y_0 , затем в N строках описаны координаты появления монстров X_i и Y_i (все координаты — целые числа, не превосходящие по модулю 10^4).

Выходные данные

В выходной файл OUTPUT.TXT выведите наименьшее количество выстрелов, необходимых для отражения атаки.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4 90 1 1 2 2 0 2 0 0 2 0	2

```
/*
    Задача: 904. Crimsonland

    Решение: геометрия, бинарный поиск, перебор, O(n * log(n) * 360 / a)

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
#define isz(x) (int)(x).size()
```

```

typedef long double ld;
const ld PI = std::acos(ld(-1));
const ld EPS = 1e-14L;
struct Vec {
    ld x, y, angle;
};
ld dot(const Vec& a, const Vec& b) {
    return a.x * b.x + a.y * b.y;
}
ld norm(const Vec& a) {
    return std::sqrt(dot(a,a));
}
bool less(ld a, ld b) {
    return a < b * (1 + EPS);
}
int solve(ld angle, std::vector<Vec> arr) {
    for (auto &it : arr) {
        it.angle = std::atan2(it.y,it.x);
    }
    std::stable_sort(all(arr), [&](const Vec& a, const Vec& b) {
        return a.angle < b.angle;
    });
    const int n = isz(arr);
    for (int i = 0; i < n; ++i) {
        arr.push_back(arr[i]);
        arr.back().angle += 2 * PI;
    }
    auto subtask = [&](int id) {
        int limit = id + n, answ = 0;
        while (id < limit) {
            answ++;
            int low = id, high = isz(arr);
            while (high - low > 1) {
                int mid = (low + high) / 2;
                if (less(arr[mid].angle - arr[id].angle, angle)) {
                    low = mid;
                } else { high = mid; }
            }
            id = high;
        }
        return answ;
    };
    int answ = n;
    for (int i = 0; i < n; ++i) {
        answ = std::min(answ, subtask(i));
    }
    return answ;
}
int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0);
    for (int n, a; std::cin >> n >> a; ) {
        Vec p; std::cin >> p.x >> p.y;
        std::vector<Vec> arr(n);
        for (auto &it : arr) {
            std::cin >> it.x >> it.y;
            it.x -= p.x, it.y -= p.y;
        }
        std::cout << solve(a * PI / 180, arr) << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №1011

Треугольник и окружности

(Время: 1 сек. Память: 16 Мб Сложность: 56%)

Известно, что для любого треугольника существуют как вписанная в него, так и описанная около него окружности, которые определяются однозначно.

Центром вписанной окружности служит точка пересечения биссектрис треугольника, а точка пересечения серединных перпендикуляров к сторонам треугольника представляет собой центр описанной около него окружности. Радиус вписанной окружности равен расстоянию от ее центра до любой из сторон треугольника, в то время, как радиус описанной окружности соответствует расстоянию от ее центра до любой из вершин треугольника.

По заданным координатам вершин треугольника (x_1, y_1) , (x_2, y_2) и (x_3, y_3) требуется найти центр и радиус либо вписанной, либо описанной около него окружности.

Входные данные

Первая строка входного файла INPUT.TXT содержит 6 вещественных чисел $x_1, y_1, x_2, y_2, x_3, y_3$ – координаты вершин треугольника ненулевой площади. Все числа не превышают 100 по абсолютной величине. Во второй строке записана команда «In» (вписанная) или «Out» (описанная), определяющая тип окружности, которую следует найти.

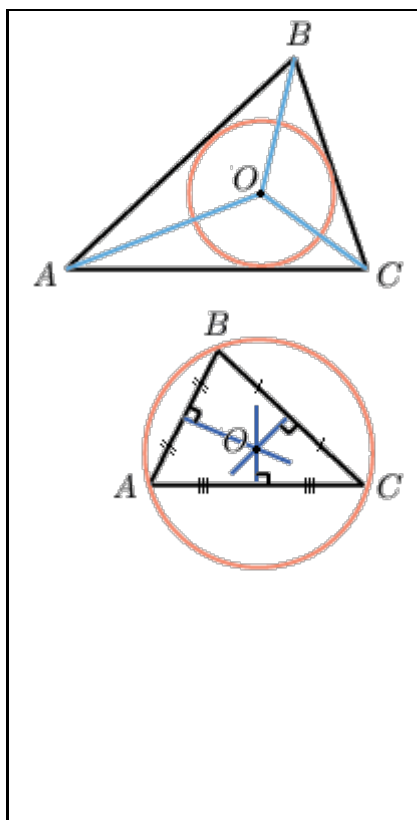
Выходные данные

В выходной файл OUTPUT.TXT выведите через пробел координаты центра (x, y) и радиус r искомой окружности. Все значения следует выводить с точностью не меньшей, чем 10^{-3} .

Примеры

Система оценки

Решения для равносторонних треугольников оцениваются в 20 баллов. Решения для прямоугольных треугольников с поиском описанной окружности оцениваются в 30 баллов.



№	INPUT.TXT	OUTPUT.TXT	Рисунок к примерам
1	2 3 6 1 7 5 In	5.14 2.87 1.288	
2	2 3 6 1 7 5 Out	4.72222 3.44444 2.758	

/*

Задача: 1011. Треугольник и окружности

Решение: геометрия, точка, вектор, вращение, прямая, треугольник, окружность, O(1)

Автор: Дмитрий Козырев, <https://github.com/dmkz> , dmkozyrev@rambler.ru

*/

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cassert>
```

```
typedef long double Real;
```

```
const Real PI = std::acos(Real(-1));
```

```
struct Point {
    Real x, y;

    static Point read() {
        Real x_, y_;
        std::cin >> x_ >> y_;
        return Point{x_, y_};
    }
};
```

```

inline Real norm() const {
    return std::sqrt(x * x + y * y);
}

inline Point operator-(const Point& other) const {
    return Point{x-other.x, y-other.y};
}

inline Point operator+(const Point& other) const {
    return Point{x+other.x, y+other.y};
}

inline Point operator/(const Real q) const {
    return Point{x / q, y / q};
}

inline Point rot(const Real angle) const {
    return Point {
        std::cos(angle) * x - std::sin(angle) * y,
        std::sin(angle) * x + std::cos(angle) * y
    };
}

};

inline Real det(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}

struct Line {

    Real A, B, C;

    Line(const Point& a, const Point& b) {
        // -(b.y-a.y)*(x-a.x) + (b.x-a.x)*(y - a.y) = 0
        A = a.y - b.y;
        B = b.x - a.x;
        C = -(a.x * A + a.y * B);
        assert(std::abs(A * a.x + B * a.y + C) <= 1e-14);
        assert(std::abs(A * b.x + B * b.y + C) <= 1e-14);
    }

    inline Point intersect(const Line& L) const {
        auto q = det(Point{ A, B}, Point{ L.A, L.B});
        auto a = det(Point{-C, B}, Point{-L.C, L.B});
        auto b = det(Point{ A,-C}, Point{ L.A,-L.C});
        return Point{a / q, b / q};
    }

    inline Real value(const Point& p) const {
        return A * p.x + B * p.y + C;
    }

};

inline Real dist(const Point& p, const Line& L) {
    return std::abs(L.value(p)) / Point{L.A, L.B}.norm();
}

inline Real dist(const Point& a, const Point& b) {
    return (a-b).norm();
}

int main() {
    Point A = Point::read();
    Point B = Point::read();
    Point C = Point::read();
    std::string query;
    std::cin >> query;
    if (query == "In") {

```

```

    auto v1 = (B-A);
    auto v2 = (C-A);
    v1 = v1 / v1.norm();
    v2 = v2 / v2.norm();
    Line L1(A, A+(v1+v2)/2);
    v1 = A-B;
    v2 = C-B;
    v1 = v1 / v1.norm();
    v2 = v2 / v2.norm();
    Line L2(B, B+(v1+v2)/2);
    auto Center = L1.intersect(L2);
    auto Radius = dist(Center, Line(A, B));
    std::cout << std::fixed << std::setprecision(3) << Center.x << ' ' << Center.y << '
} else {
    Point M1 = (A+B)/2;
    Point M2 = (B+C)/2;
    Line L1(M1, M1 + (B-A).rot(PI/2));
    Line L2(M2, M2 + (C-B).rot(PI/2));
    auto Center = L1.intersect(L2);
    auto Radius = dist(Center, A);
    std::cout << std::fixed << std::setprecision(3) << Center.x << ' ' << Center.y << '
}
return 0;
}

```

ЗАДАЧА №1178

Построение

(Время: 0,25 сек. Память: 16 Мб Сложность: 56%)

В одной военной части решили построить солдат в одну шеренгу по росту. Т.к. часть была далеко не образцовая, то солдаты часто приходили не вовремя, а то их и вовсе приходилось выгонять из шеренги за плохо начищенные сапоги. Однако солдаты в процессе прихода и ухода должны были всегда быть выстроены по росту – сначала самые высокие, а в конце – самые низкие. За расстановку солдат отвечал прапорщик, который заметил интересную особенность – все солдаты в части разного роста.

Ваша задача состоит в том, чтобы помочь прапорщику правильно расставлять солдат, а именно для каждого приходящего солдата указывать, перед каким солдатом в строе он должен становиться.

Входные данные

Первая строка входного файла INPUT.TXT содержит число N – количество команд ($1 \leq N \leq 30\,000$). В каждой следующей строке содержится описание команды: число 1 и X , если солдат приходит в строй (X – рост солдата, натуральное число до 100 000 включительно) и число 2 и Y , если солдата, стоящего в строе на месте Y надо удалить из строя (солдаты в строе нумеруются с нуля).

Выходные данные

В выходной файл OUTPUT.TXT выведите в отдельной строке для каждой команды 1 (добавление в строй) число K – номер позиции, на которую должен встать этот солдат (все стоящие за ним двигаются назад).

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5	
	1 100	0
	1 200	0
	1 50	2
	2 1	1
	1 150	

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>

struct SqrtDecomposition {
    std::vector<int> arr, sum;

    const int GSIZE = 256;

    SqrtDecomposition(int size = 0, int element = 0) {
        arr.assign(size, element);
        sum.assign((size+GSIZE-1)/GSIZE, 0);
        for (int g = 0; g < (int)sum.size(); ++g) {
            const int begin = g * GSIZE;
```



```

        const int after = std::min(begin+GSIZE, size);
        for (int i = begin; i < after; ++i) {
            sum[g] += arr[i];
        }
    }
}

void set(int pos, int value) {
    sum[pos/GSIZE] += (value-arr[pos]);
    arr[pos] = value;
}

int get(int l, int r) {
    const int gl = l / GSIZE;
    const int gr = r / GSIZE;
    if (gl == gr) {
        int s = 0;
        for (int i = l; i <= r; ++i) {
            s += arr[i];
        }
        return s;
    } else {
        int s = 0;
        for (int i = l, after = (gl+1)*GSIZE; i < after; ++i) {
            s += arr[i];
        }
        for (int g = gl + 1; g < gr; ++g) {
            s += sum[g];
        }
        for (int i = gr*GSIZE; i <= r; ++i) {
            s += arr[i];
        }
        return s;
    }
}

};

struct Container {
    // Эффективно (за корень) удаляет i-ый элемент в отсортированного множестве
    // Все значения ожидаются от 1 до MAX_VALUE
    std::vector<std::vector<int>> items;
    const int GSIZE = 256;

    Container(int MAX_VALUE) {
        items.resize((MAX_VALUE + GSIZE-1) / GSIZE);
    }

    void insert(int val) { // Вставка значения
        assert(val >= 1);
        const int g = (val-1) / GSIZE;
        auto it = std::upper_bound(items[g].begin(), items[g].end(), val);
        items[g].insert(it, val);
    }

    // Получение значения i-го по счету элемента (нумерация с нуля)
    int get(int pos, bool remove = true) {
        int answ = -1;
        for (auto& it : items) {
            if (pos >= (int)it.size()) {
                pos -= (int)it.size();
            } else {
                answ = *(it.begin() + pos);
                if (remove) {
                    it.erase(it.begin() + pos);
                }
                break;
            }
        }
        assert(answ != -1);
        return answ;
    }
};

```

```

    }
};

int main() {
    SqrtDecomposition sd(1+100000, 0);
    Container cont(100000);
    int q;
    scanf("%d", &q);
    while (q--) {
        int type;
        scanf("%d", &type);
        if (type == 1) {
            // Добавление
            int value; scanf("%d", &value);
            value = 100001-value;
            printf("%d ", sd.get(0, value));
            sd.set(value, 1);
            cont.insert(value);
        } else {
            assert(type == 2);
            int pos;
            scanf("%d", &pos);
            int value = cont.get(pos);
            sd.set(value, 0);
        }
    }
    return 0;
}

```

ЗАДАЧА №1197

Количество различных подстрок

(Время: 1 сек. Память: 16 Мб Сложность: 56%)

В заданной строке S требуется найти количество различных подстрок ненулевой длины.

Входные данные

Входной файл INPUT.TXT содержит непустую строку S , состоящую из строчных букв английского алфавита, длина строки S не превышает 3000 символов.

Выходные данные

В выходной файл OUTPUT.TXT выведите количество различных подстрок строки S .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	abcde	15
2	aaaaa	5
3	abacabadabacaba	85

```
/*
    Задача: 1197. Количество различных подстрок

    Решение: полиномиальный хэш, сортировка, строки,  $O(n^2 \log(n))$ 

    Автор: Dmitry Kozyrev, https://github.com/dmkz , dmkozyrev@rambler.ru
*/

#pragma GCC optimize ("O3")

#include <stdio.h>
#include <string>
#include <vector>
#include <random>
#include <chrono>
#include <array>
#include <iostream>
#include <algorithm>

typedef unsigned long long ull;

// Generate random seed:
inline ull gen_seed() {
    return ull(new ull) ^ ull(std::chrono::high_resolution_clock::now().time_since_epoch().
}

// Generate random base in (before, after) open interval:
int gen_base(int before, int after) {
    std::mt19937 gen(gen_seed());
    std::uniform_int_distribution<int> dist(before+2, after-1);
    int base = dist(gen);
    return base % 2 == 0 ? base - 1 : base;
```

```

}

struct PolyHash {
    // ----- Static variables -----
    static const ull mod = (ull(1) << 61) - 1; // prime mod of hashing
    static int base; // odd base of hashing
    static std::vector<ull> pow; // powers of base modulo mod;

    // ----- Static functions -----
    static inline ull add(ull a, ull b) {
        // Calculate (a + b) % mod, 0 <= a < mod, 0 <= b < mod
        return (a += b) < mod ? a : a - mod;
    }

    static inline ull sub(ull a, ull b) {
        // Calculate (a - b) % mod, 0 <= a < mod, 0 <= b < mod
        return (a -= b) < mod ? a : a + mod;
    }

    static inline ull mul(ull a, ull b){
        // Calculate (a * b) % mod, 0 <= a < mod, 0 <= b < mod
        ull l1 = (uint32_t)a, h1 = a >> 32, l2 = (uint32_t)b, h2 = b >> 32;
        ull l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
        ull ret = (l & mod) + (l >> 61) + (h << 3) + (m >> 29) + (m << 35 >> 3) + 1;
        ret = (ret & mod) + (ret >> 61);
        ret = (ret & mod) + (ret >> 61);
        return ret-1;
    }

    // ----- Variables of class -----
    std::vector<ull> pref; // polynomial hash on prefix

    // Constructor from string:
    PolyHash(const std::string& s)
        : pref(s.size()+1u, 0)
    {
        // Pre-calculate powers of base:
        while (pow.size() <= s.size()) {
            pow.push_back(mul(pow.back(), base));
        }
        // Calculate polinomial hash on prefix:
        for (int i = 0; i < (int)s.size(); ++i) {
            pref[i+1] = add(mul(pref[i], base), s[i]);
        }
    }

    // Get hash from [pos, pos+len-1] segment of string
    inline ull operator()(const int pos, const int len) const {
        return sub(pref[pos+len], mul(pref[pos], pow[len]));
    }
};

// Init static variables of class PolyHash:
int PolyHash::base((int)1e9+7);
std::vector<ull> PolyHash::pow{1};

int main() {
    // Generate random base:
    PolyHash::base = gen_base(256, 2e9);

    // Input and prepare hashes:
    char buf[3000+1];
    scanf("%3000s", buf);
    std::string s(buf);

    // Calculate rolling hashes on prefix:
    PolyHash hash(s);

```

```
// Get hashes from all substring and count unique:
int answer = 0;
for (int len = 1; len <= (int)s.size(); ++len) {
    std::vector<ull> hashes;
    for (int p = 0; p + len <= (int)s.size(); ++p) {
        hashes.push_back(hash(p, len));
    }
    std::sort(hashes.begin(), hashes.end());
    answer += int(std::unique(hashes.begin(), hashes.end()) - hashes.begin());
}
printf("%d", answer);
return 0;
}
```

ЗАДАЧА №1469

Бобры-Зомби

(Время: 3 сек. Память: 16 Мб Сложность: 56%)

Вы сидите на вершине дерева с небольшой командой выживших в центре бобро-зомби апокалипсиса. На земле вас окружили N бобров-зомби, которые вот-вот сточат ствол дерева! У каждого бобра-зомби есть свой размер S_i , выраженный натуральным числом.

Учёный, который недавно погиб от несчастного случая, рассказал, что, если стравить двух бобров-зомби между собой с помощью особых радиоволн, то они образуют могущественного Бобротрона, размер которого будет равен произведению размеров стравленных бобров. Последними словами учёного было уточнение, что если размер Бобротрона будет квадратом целого числа, то он будет на вашей стороне, вселит ужас в остальных бобров-зомби и заставит их убежать!

Срочно посчитайте, сколькими способами можно создать доброго Бобротрона, стравливая двух бобров-зомби!

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число N – количество бобров-зомби ($2 \leq N \leq 200\,000$).

Вторая строка содержит N целых чисел S_i , разделённых пробелами – размеры бобров-зомби ($1 \leq S_i \leq 200\,000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите целое число – количество пар бобров, которые смогут образовать доброго Бобротрона.

Примеры

Примечание

В первом примере подойдёт любая пара бобров.
Во втором примере подойдёт лишь пара бобров с размерами 2 и 8.

№	INPUT.TXT	OUTPUT.TXT
1	4 1 4 9 16	6
2	4 2 4 6 8	1

```
#include <stdio.h>
#include <cassert>
#include <string>
#include <vector>
#include <algorithm>
#include <iostream>
```

```

typedef long long ll;

std::vector<int> primes;

void prepare() {
// Генерация всех простых чисел при помощи решета Эратосфена:
    assert(primes.empty());
    const int n = 500; // ~sqrt(200000)
    std::vector<bool> is_prime(1+n, true);
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i*i <= n; ++i) {
        if (!is_prime[i]) {
            continue;
        }
        for (int j = i * i; j <= n; j += i) {
            is_prime[j] = false;
        }
    }
    for (int i = 2; i <= n; ++i) {
        if (is_prime[i]) {
            primes.push_back(i);
        }
    }
}

ll solve(const std::vector<int>& arr) {
// Делим все числа на квадраты простых чисел до тех пор пока делятся
// Остальные простые множители в разложении либо отсутствуют, либо имеют степень 1
// Тогда полный квадрат можно составить из одинаковых сомножителей
    std::vector<int> numbers;
    for (auto value : arr) {
        const int limit = value;
        for (auto p : primes) {
            if (value < p || p * p > limit) {
                break;
            }
            while (value % (p * p) == 0) {
                value /= p * p;
            }
        }
        assert(value != 0);
        numbers.push_back(value);
    }
    assert(!numbers.empty());
    std::sort(numbers.begin(), numbers.end());
    // Считаем размеры групп из одинаковых чисел:
    ll answ = 0;
    int last = numbers.front(), cnt = 0;
    for (auto it : numbers) {
        if (last == it) {
            cnt++;
        } else {
            answ += cnt * (cnt-1LL) / 2;
            last = it;
            cnt = 1;
        }
    }
    answ += cnt * (cnt-1LL) / 2;
    return answ;
}

int main() {
    prepare();
    int n; scanf("%d", &n);
    std::vector<int> arr(n);
    for (auto& it : arr) {
        scanf("%d", &it);
    }
}

```

```
std::cout << solve(arr);  
return 0;  
}
```


ЗАДАЧА №710

Булева алгебра - 2

(Время: 1 сек. Память: 16 Мб Сложность: 57%)

В каждом языке программирования, даже самом простом, есть оператор ветвления, позволяющий проверить истинность логического выражения и, в зависимости от его результата, выполнить то или иное действие. Условие оператора ветвления представляет собой логическое (булевское) выражение, результатом которого может быть либо истина (TRUE), либо ложь (FALSE). Переменные, которые могут участвовать в логическом выражении, называются булевскими (boolean). Булевские переменные могут объединяться в сложные условия при помощи логических операций (функций):

- $AND(x_1, x_2, \dots, x_s)$. Операция «И», возвращает истинное значение, если все ее операнды истинны. ($2 \leq \text{количество операндов} \leq s$);
- $OR(x_1, x_2, \dots, x_s)$. Операция «ИЛИ», возвращает истинное значение, если хотя бы один ее операнд истинен. ($2 \leq \text{количество операндов} \leq s$);
- $NOT(x_1)$. Операция «НЕ», меняет значение операнда x_1 на противоположное (операнд всегда один).

В логическом условии может использоваться несколько логических функций, вложенных друг в друга, то есть результат одной функции может использоваться другой в качестве операнда. Например, $AND(A, B, OR(C, D))$. Данное выражение будет истинно тогда, когда истинны A, B и (C или D).

Требуется написать программу, которая по имеющемуся логическому выражению и значению логических переменных определит результат выражения. Количество операндов у функций AND и OR может быть любым ($2 \leq s \leq 26$).

Входные данные

Первая строка входного файла INPUT.TXT содержит логическое выражение (длина не более 255 символов). Вторая строка содержит два числа, разделенных одним или несколькими пробелами: N – количество блоков (не более 10), K - количество переменных (не более 26). Далее следует N блоков, каждый имеет следующую структуру: состоит из K строк, каждая содержит выражение типа <переменная = значение>. Переменные задаются заглавными английскими буквами, значение – константами TRUE или FALSE (заглавные буквы).

Выходные данные

В выходной файл OUTPUT.TXT выведите N строк со значением результата логического выражения для переменных соответствующего блока.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	AND(A,NOT(B)) 1 2 A=FALSE B=TRUE	FALSE

2	OR(NOT(AND(A,B)),A)	
	3 2	
	A=FALSE	
	B=TRUE	TRUE
	A=TRUE	TRUE
	B=TRUE	TRUE
	A=FALSE	
	B=FALSE	

/*

Задача: 710. Булева алгебра - 2

Решение: рекурсия, строки, разбор выражений, РБНФ, $O(q \cdot \text{len}(s))$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

*/

```
#include <bits/stdc++.h>
#define isz(x) (int)(x).size()
#define all(x) (x).begin(), (x).end()
namespace Solution {
    int p;
    std::string s;
    std::vector<bool> value(256);
    void init(std::string s_) {
        s = s_;
        p = 0;
    }
    void set(std::string s_) {
        auto f = s_.find('=');
        auto var = s_.substr(0, f);
        auto val = s_.substr(f+1);
        assert(var.size() == 1u);
        assert(val == "FALSE" || val == "TRUE");
        value[var[0]] = (val == "TRUE");
    }
    bool calc() {
        assert(p < isz(s) && 'A' <= s[p] && s[p] <= 'Z');
        std::string op;
        while (p < isz(s) && 'A' <= s[p] && s[p] <= 'Z') { op.push_back(s[p++]); }
        if (isz(op) == 1) {
            return value[op[0]];
        }
        if (op == "NOT") {
            assert(s[p++] == '(');
            bool ret = calc();
            assert(s[p++] == ')');
            return !ret;
        }
        assert(op == "AND" || op == "OR");
        assert(s[p++] == '(');
        bool ret = calc();
        while (s[p] == ',') {
            p++;
            bool temp = calc();
            ret = (op == "AND" ? ret && temp : ret || temp);
        }
        assert(s[p++] == ')');
        return ret;
    }
    bool solve() {
        p = 0;
        return calc();
    }
}

std::string& remove_spaces(std::string& s) {
```

```

        s.erase(std::remove_if(all(s), [&](char c) { return std::isspace(c); }), s.end());
        return s;
    }

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    std::string s;
    std::getline(std::cin, s);
    Solution::init(remove_spaces(s));
    std::getline(std::cin, s);
    std::stringstream ss(s);
    int q, n; ss >> q >> n;
    while (q--) {
        for (int i = 0; i < n; ++i) {
            std::getline(std::cin, s);
            Solution::set(remove_spaces(s));
        }
        std::cout << (Solution::solve() ? "TRUE\n" : "FALSE\n");
    }
    return 0;
}

```

ЗАДАЧА №934

Балда

(Время: 2 сек. Память: 16 Мб Сложность: 57%)

А вы играли в «балду»? Это такая игра, когда из букв одного слова нужно составить как можно больше других слов. И чем длиннее такие слова, тем больше очков игрок заработает. Отсюда понятно, что самые выгодные слова – это те, которые получены перестановкой букв исходного слова.

Хитрый Дима решил написать программу, которая распечатает ему заготовки для игры в "балду". Дима их выучит, и будет побеждать всех своих друзей. Дима решил распечатать группы слов, которые получают перестановкой букв.

Таких групп может оказаться слишком много, поэтому Дима решил распечатать первые пять с самым большим количеством слов. Ну, а если в словаре окажется менее пяти групп, Дима распечатает их все. А, может быть, и Вы себе такую программу создадите? Глядишь, и пригодится!

Входные данные

Входной файл INPUT.TXT содержит число N – количество слов в словаре ($2 \leq N \leq 25000$). Далее идет N слов, по одному в строке. Каждое слово содержит не более 40 символов. Коды ASCII символов в словах превышают 32.

Выходные данные

В выходной файл OUTPUT.TXT выведите первые пять групп, отсортированных по количеству слов. Если групп меньше пяти, выведите все группы. Для каждой группы отсортируйте все слова в лексикографическом порядке. Повторяющиеся слова следует выводить однократно. Если есть несколько групп одного размера, отсортируйте их в лексикографическом порядке первого слова в группе (первое слово в группе – в лексикографическом порядке, а не в порядке добавления). Выводить группы слов следует согласно формату, описанному в примерах.

Примеры

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	16 undisplayed trace tea singleton eta eat displayed crate cater carte caret beta beat bate ate abet	Group of size 5: caret carte cater crate trace . Group of size 4: abet bate beat beta . Group of size 4: ate eat eta tea . Group of size 1: displayed . Group of size 1: singleton .
2	8 abc c++ cba abc pascal java scalpa basic	Group of size 3: abc cba . Group of size 2: pascal scalpa . Group of size 1: basic . Group of size 1: c++ . Group of size 1: java .

```

/*
    "Балда": сортировка, строки, O(n*log(n)*mxLen)
*/

#include <stdio.h>
#include <string>
#include <algorithm>
#include <vector>
#include <map>
#include <set>

int main() {
    int n; scanf("%d", &n);

    std::vector<std::string> words(n);

    std::map<std::string, std::vector<int>> groups;

    for (auto& it : words) {
        char buf[41]; scanf("%40s", buf);
        it = buf;
    }

    for (int i = 0; i < n; ++i) {
        auto s = words[i];
        std::sort(s.begin(), s.end());
        groups[s].push_back(i);
    }

    auto less = [&](const std::vector<int>& a, const std::vector<int>& b) {
        return a.size() > b.size() || (a.size() == b.size() && words[a.front()] < words[b.f
    };

    std::set<std::vector<int>, decltype(less)> set(less);

```

```

for (auto& it : groups) {
    std::sort(it.second.begin(), it.second.end(), [&](const int i, const int j) {
        return words[i] < words[j];
    });
    set.insert(it.second);
    if (set.size() > 5u) {
        set.erase(std::prev(set.end()));
    }
}

for (auto g : set) {
    printf("Group of size %d: ", (int)g.size());
    g.erase(std::unique(g.begin(), g.end(), [&](const int i, const int j){return words[
    for (auto& it : g) {
        printf("%s ", words[it].c_str());
    }
    printf(".\n");
}

return 0;
}

```

ЗАДАЧА №956

Юный гитарист

(Время: 2 сек. Память: 16 Мб Сложность: 57%)

Ваня решил заняться музыкой. Но какой музыкальный инструмент выбрать для изучения, он пока еще не решил. Его друг Женя, будучи немного постарше и опытнее, уже хорошо освоил гитару. Он хочет продемонстрировать Ване возможности игры на гитаре.

На гитаре шесть струн, каждая из них может быть зажата пальцем на определенном ладу или оставлена открытой (не прижатой ни на каком ладу), от этого изменяется нота, которая звучит, когда струна колеблется.

Чтобы продемонстрировать возможности инструмента, Женя ведет с Ваней такую беседу. Ваня называет некоторый аккорд, а Женя говорит ему, сколькими способами можно этот аккорд взять на гитаре, то есть выбрать на каждой струне лад и зажать его (или оставить ее открытой), чтобы этот аккорд зазвучал. Женя любит громкий звук, поэтому звучать должны все струны.

Нот в музыке, конечно, семь, но помимо них существуют еще полутона. Поэтому для удобства мы будем считать, что нот всего 12. Их обозначения такие: A, Bb, B, C, C#, D, D#, E, F, F#, G, G#.

Кроме этого, C# может быть обозначен как Db, Bb как A#, D# как Eb, F# как Gb, а G# как Ab. Обозначения нот приведены в порядке их следования, то есть за A идет Bb, за Bb идет B, . . . , за G# идет снова A. Расстояние между любыми двумя соседними нотами равно одному полутону.

Струн у гитары шесть, они занумерованы с 1 по 6. Каждая струна издает определенную ноту, когда звучит и не прижата ни на каком ладу. Совокупность нот, соответствующих открытым струнам, называется строем гитары. Если открытую струну, издающую ноту номер i , зажать на j -том ладу (лады нумеруются с единицы), то этой струной будет издаваться нота $i + j$, то есть, от ноты i надо сместиться на j нот по циклу. Например, если открытая струна издает ноту D, то зажав ее на третьем ладу, мы получим ноту F, а на восьмом – ноту Bb.

На гитаре Вани N ладов, поэтому $1 \leq j \leq N$. Аккордов существует большое множество, но строятся они по общей схеме. Вначале выбирается так называемая тоника аккорда – нота, от которой он будет строиться. Затем остальные ноты аккорда выписываются уже относительно нее.

Запись аккорда состоит из двух частей. Первая часть – это всегда обозначение тоники аккорда. Вторая часть описывает само звучание аккорда. Мы ограничимся несколькими частными случаями.

Мажорный аккорд

К тонике добавляются ноты, отстоящие на 4 и 7 полутонов. Вторая часть записи этого аккорда пуста.

Пример: Eb, состоит из нот Eb, G, Bb.

Минорный аккорд

К тонике добавляются ноты, отстоящие на 3 и 7 полутонов.

Вторая часть записи этого аккорда состоит из маленькой английской буквы m.

Пример: Am, состоит из нот A, C, E.

Мажорный септаккорд

Образуется из мажорного аккорда путем добавления ноты, отстоящей от тоники на 10 полутонов.

Вторая часть состоит из цифры 7.

Пример: E7, состоит из нот E, B, G#, D.

Минорный септаккорд

Образуется из минорного аккорда путем добавления ноты, отстоящей от тоники на 10 полутонов.

Вторая часть состоит из буквы m и цифры 7.

Пример: Gm7, состоит из нот G, D, Bb, F.

Напишите программу, помогающую Жене отвечать на вопросы Вани. Учтите, что аккорд считается взятым на гитаре, если ни одна нота аккорда не пропущена и не взята ни одна нота, не принадлежащая аккорду. В отличие от реальной игры на гитаре, расстояние между ладами, на которых зажата хотя бы одна струна, может быть любым.

Входные данные

В первой строке входного файла INPUT.TXT содержится N – число ладов гитары ($0 \leq N \leq 9$). Во второй строке записан строй гитары – шесть нот, соответствующие шести струнам гитары. Ноты обозначены, как в тексте задачи, и отделены друг от друга пробелами. Регистр букв важен. В третьей строке записан аккорд, который надо взять на гитаре.

Выходные данные

В выходной файл OUTPUT.TXT выведите количество способов взять данный аккорд на данной гитаре.

Примеры

Пояснения к примерам

Если обозначить вариант взятия аккорда шестью цифрами – номерами ладов, на которых зажаты струны с первой по шестую, считая открытую струну зажатой на нулевом ладу то в первом примере четыре способа таковы:

(0; 1; 0; 2; 3; 0)

(3; 1; 0; 2; 3; 0)

(0; 1; 0; 2; 3; 3)

(3; 1; 0; 2; 3; 3)

Во втором примере можно играть только на открытых струнах, зато гитара настроена ровно на требуемый аккорд.

№	INPUT.TXT	OUTPUT.TXT
1	3 E B G D A E C	4
2	0 A C E A C E Am	1

/*

Задача: 965. Юный гитарист

Решение: перебор, реализация, строки, $O(6 * n^6)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

```
*/

#include <iostream>
#include <string>
#include <algorithm>
#include <cmath>
#include <cassert>
#include <vector>

const std::vector<std::string> fi = {"A" , "Bb", "B" , "C" , "C#", "D" , "D#", "E" , "F" ,
const std::vector<std::string> se = {"A" , "A#", "B" , "C" , "Db", "D" , "Eb", "E" , "F" ,

#define all(x) (x).begin(), (x).end()
#define find(x, t) int(std::find(all(x), (t)) - (x).begin())

int getId(const std::string s) {
    int pos = (int)std::min(find(fi, s), find(se, s));
    return pos;
}

std::vector<int> getAccord(std::string s) {
    {
        int id = getId(s);
        if (id != 12) {
            return std::vector<int>{id, (id+4)%12, (id+7)%12};
        }
    }
    if (s.back() == 'm') {
        s.pop_back();
        int id = getId(s);
        assert(id != 12);
        return std::vector<int>{id, (id+3)%12, (id+7)%12};
    }
    if (s.back() == '7') {
        s.pop_back();
        if (s.back() == 'm') {
            s.pop_back();
            int id = getId(s);
            assert(id != 12);
            return std::vector<int>{id, (id+3)%12, (id+7)%12, (id+10)%12};
        }
        int id = getId(s);
        assert(id != 12);
        return std::vector<int>{id, (id+4)%12, (id+7)%12, (id+10) % 12};
    }
    assert(false);
    return std::vector<int>{};
}

int main() {
    int n;
    while (std::cin >> n) {
        std::vector<int> initial(6);
        for (int i = 0; i < 6; ++i) {
            std::string s; std::cin >> s;
            initial[i] = getId(s);
        }
        std::string s; std::cin >> s;
        auto need = getAccord(s);
        std::sort(need.begin(), need.end());
        assert(std::unique(need.begin(), need.end()) == need.end());
        n++;
        int limit = (int)std::pow(n, 6), answ = 0;
        for (int mask = 0; mask < limit; ++mask) {
            std::vector<int> tmp(6);
            for (int i = 0, v = mask; i < 6; ++i) {
                tmp[i] = (initial[i] + (v % n)) % 12;
            }
        }
    }
}
```

```
        v /= n;
    }
    std::sort(tmp.begin(), tmp.end());
    tmp.erase(std::unique(tmp.begin(), tmp.end()), tmp.end());
    answ += (tmp == need);
}
std::cout << answ << std::endl;
}
return 0;
}
```

ЗАДАЧА №979

Формула 001

(Время: 2 сек. Память: 32 Мб Сложность: 57%)

Мальчик Миша собирается участвовать в школьных соревнованиях по гонкам с препятствиями по версии «Формула 001». Но к любым соревнованиям необходимо готовиться. Младший брат Миши, Ральф, не так силен в гонках с препятствиями, как его старший брат, но зато он обладает незаурядной фантазией. Он решил помочь брату и придумал игру, играя в которую, можно существенно увеличить свой опыт в вождении гоночного автомобиля.

Игра состоит в следующем. Пусть у нас есть бесконечное игровое поле, покрытое бесконечной квадратной сеткой. Некоторое множество узлов этой сетки отмечено. Назовем это множество S . В игре участвуют несколько игроков. У каждого игрока есть машина – фигура, которая может находиться только в узлах из множества S . В начале игры все машины находятся в различных начальных узлах. Ход каждого игрока состоит в перемещении машины в некоторый узел из множества S , возможно, тот же самый. Цель игры – добраться до определенного, финишного узла первым.

Ход происходит по следующим правилам. Пусть на предыдущем шаге машина была перемещена на вектор (X, Y) (если это первый шаг, то $X=0$ и $Y=0$). Тогда за один текущий ход машину можно передвинуть на один из следующих векторов: $(X-1, Y-1)$, $(X-1, Y)$, $(X-1, Y+1)$, $(X, Y-1)$, (X, Y) , $(X, Y+1)$, $(X+1, Y-1)$, $(X+1, Y)$ и $(X+1, Y+1)$.

Конечно, на какой-либо из этих векторов машину переместить можно только при том условии, что после этого она попадет в узел из множества S . Если ход сделать нельзя, то игрок считается проигравшим и выбывает из игры.

Некоторое время поиграв в эту игру, Миша и Ральф занялись ее анализом. В данный момент они хотят узнать, за какое наименьшее число ходов из стартового узла возможно попасть в финишный. Сами они эту задачу решить не смогли и обратились за помощью к Вам. Помогите им!

Входные данные

В первой строке входного файла INPUT.TXT находится N – число элементов множества S . $2 \leq N \leq 1000$. В последующих N строках находятся координаты узлов из этого множества – целые числа X_i, Y_i ($-10^9 \leq X_i, Y_i \leq 10^9$).

Никакие два узла во входном файле не совпадают. Занумеруем эти узлы, начиная с 1, в порядке их описания во входном файле. Стартовым узлом будет являться узел с номером 1, финишным узел с номером N .

Выходные данные

В выходной файл OUTPUT.TXT выведите -1, если добраться до финишного узла невозможно. Иначе, в первой строке выведите минимальное требуемое число ходов K . Во второй выведите $K+1$ число – номера посещенных узлов в порядке посещения. Первым узлом должен быть узел с номером 1, последним – узел с номером N .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 0 0 0 1 0 2	2 1 2 3
2	3 0 0 0 2 0 3	-1

```

/*
    Задача: 979. Формула 001

    Решение: графы, поиск в ширину, геометрия,  $O(n^2 \cdot \log(n))$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
#define isz(x) (int)(x).size()
typedef std::vector<int> vi;
const int NMAX = 1024;
const int INF = (int)1e9+7;
struct Point {
    int x, y;
    Point(int x_ = 0, int y_ = 0) : x(x_), y(y_) { }
    Point operator+(const Point& p) const { return Point(x + p.x, y + p.y); }
    Point operator-(const Point& p) const { return Point(x - p.x, y - p.y); }
};
bool operator<(const Point& a, const Point& b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}
bool operator!=(const Point& a, const Point& b) {
    return a < b || b < a;
}
vi solve (const int s, const int t, const std::vector<Point>& pt) {
    const int n = isz(pt);
    static int dist[NMAX][NMAX], from[NMAX][NMAX];
    std::fill(&dist[0][0], &dist[0][0]+NMAX*NMAX, INF);
    std::fill(&from[0][0], &from[0][0]+NMAX*NMAX, -1);
    std::vector<Point> sorted;
    for (auto &it : pt) { sorted.push_back(it); }
    std::sort(all(sorted));
    vi index(n);
    for (int i = 0; i < isz(pt); ++i) {
        int j = int(std::lower_bound(all(sorted), pt[i]) - sorted.begin());
        index[j] = i;
    }
    dist[s][s] = 0;
    from[s][s] = s;
    std::queue<int> queue;
    queue.push(s * NMAX + s);
    while (!queue.empty()) {
        const int curr = queue.front() / NMAX;
        const int prev = queue.front() % NMAX;
        queue.pop();
        const int vx = pt[curr].x - pt[prev].x;
        const int vy = pt[curr].y - pt[prev].y;
        for (int dx = -1; dx <= 1; ++dx) {
            for (int dy = -1; dy <= 1; ++dy) {
                Point next_p = pt[curr] + Point(vx+dx,vy+dy);
                auto it = std::lower_bound(all(sorted), next_p);
                if (it == sorted.end() || *it != next_p) {
                    continue;
                }
            }
        }
    }
}

```

```

        const int next = index[int(it - sorted.begin())];
        if (dist[next][curr] > dist[curr][prev] + 1) {
            dist[next][curr] = dist[curr][prev] + 1;
            from[next][curr] = curr * NMAX + prev;
            queue.push(next * NMAX + curr);
        }
    }
}

int min = t;
for (int prev = 0; prev < n; ++prev) {
    if (dist[t][prev] < dist[t][min]) {
        min = prev;
    }
}
if (dist[t][min] >= INF) {
    return vi{};
}
int state = t * NMAX + min;
vi answ{t};
for (int i = dist[t][min]; i > 0; --i) {
    int next = state / NMAX;
    int curr = state % NMAX;
    state = from[next][curr];
    assert(state != -1);
    answ.push_back(curr);
}
std::reverse(all(answ));
return answ;
}

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (int n; std::cin >> n; ) {
        std::vector<Point> pt(n);
        for (auto &it : pt) { std::cin >> it.x >> it.y; }
        vi path = solve(0, n-1, pt);
        if (isz(path) == 0) { std::cout << -1 << "\n"; }
        else {
            std::cout << isz(path)-1 << "\n";
            for (auto it : path) { std::cout << it+1 << ' '; }
            std::cout << std::endl;
        }
    }
    return 0;
}

```

ЗАДАЧА №1172

Функция Эйлера

(Время: 1 сек. Память: 16 Мб Сложность: 57%)

Функция Эйлера $\varphi(n)$ – функция, равная количеству натуральных чисел, не превосходящих ее аргумента n и взаимно простых с n .

Требуется по заданному натуральному числу n вычислить значение функции Эйлера, т.е. найти количество таких натуральных чисел m , что $m \leq n$ и $\text{НОД}(m, n) = 1$.

Входные данные

Входной файл INPUT.TXT содержит натуральное число n ($n \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите $\varphi(n)$.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1	1
2	567	324
3	1280	512

```
#include <stdio.h>
```

```
int pow(int a, int n) {
    int res = 1;
    while (n > 0) {
        if (n & 1) {
            res *= a;
        }
        a *= a;
        n /= 2;
    }
    return res;
}
```

```
int euler(const int n) {
    int answer = 1, m = n;
    for (int i = 2; i * i <= n; ++i) {
        int count = 0;
        while (m % i == 0) {
            m /= i;
            ++count;
        }
        if (count > 0) {
            answer *= pow(i, count) - pow(i, count-1);
        }
    }
    if (m > 1) {
        answer *= (m-1);
    }
}
```

```
        return answer;
    }

int main() {
    int n;
    scanf("%d", &n);
    printf("%d", euler(n));
    return 0;
}
```

ЗАДАЧА №177

Склад

(Время: 1 сек. Память: 16 Мб Сложность: 58%)

На роботизированном складе имеется n отсеков, в которые робот может размещать грузы. Отсек с номером i имеет вместимость c_i . Груз с номером i имеет размер s_i , поступает на склад в момент времени a_i и забирается со склада в момент времени d_i . Вместимость отсека и размер груза имеют одну и ту же размерность. Если в отсеке с вместимостью c находится несколько грузов с суммарным размером d , то свободное место в этом отсеке равно $c - d$.

Когда груз с номером i поступает на склад, робот сначала пытается найти отсек, в котором достаточно свободного места для размещения этого груза. Если отсеков, в которых достаточно свободного места, несколько, то робот помещает груз в тот из них, в котором свободного места меньше. Если и таких отсеков несколько, то робот выбирает отсек с минимальным номером.

Если отсеков с достаточным количеством свободного места нет, робот пытается переместить грузы, уже расположенные в отсеках. Для этого он пытается найти такой отсек и такой груз в нем, что перемещение его в другой отсек обеспечивает достаточное количество свободного места для размещения поступившего груза. Если таких вариантов перемещения грузов несколько, то выбирается тот вариант, в котором потребуется перемещение груза с минимальным размером. Если и таких вариантов несколько, то выбирается вариант перемещения, при котором в отсеке, из которого перемещается груз, свободное место после перемещения этого груза будет минимально, а при прочих равных условиях — тот вариант, при котором в отсеке, куда осуществляется перемещение, свободное место после этого перемещения будет также минимально. Если и после этого остается более одного варианта, то выбирается тот вариант, при котором номер перемещаемого груза минимален и номер отсека, в который он перемещается, — также минимален. Если варианта с перемещением одного груза найти не удалось, то груз не принимается на склад.

Требуется написать программу, которая по списку грузов, поступающих для размещения на складе, выводит последовательность действий, выполняемых роботом.

Входные данные

Первая строка входного файла содержит два целых числа: n — количество отсеков, и m — количество грузов ($1 \leq n \leq 10$, $1 \leq m \leq 100$). Вторая строка содержит n целых чисел c_i , определяющих вместимости отсеков ($1 \leq c_i \leq 10^9$). Последующие m строк описывают грузы: каждый груз описывается тремя целыми числами: своим размером s_i , временем поступления на склад a_i и временем, когда его забирают со склада d_i ($1 \leq s_i \leq 10^9$, $1 \leq a_i < d_i \leq 1000$, все времена во входном файле различны, грузы упорядочены по возрастанию времени поступления на склад). Все числа в строках разделены пробелом.

Выходные данные

Выведите последовательность действий робота в том порядке, в котором они выполняются. Следуйте формату выходного файла, приведенного в примере. Возможны следующие сообщения:

- put cargo X to cell Y - разместить груз с номером X в отсеке с номером Y;
- move cargo X from cell Y to cell Z - переместить груз с номером X из отсека с номером Y в отсек с номером Z;
- take cargo X from cell Y - взять груз с номером X из отсека с номером Y.
- cargo X cannot be stored - груз X невозможно переместить

Пример

№	INPUT.TXT	OUTPUT.TXT
1	1 1 3 3 1 2	put cargo 1 to cell 1 take cargo 1 from cell 1
2	3 5 3 2 10 1 1 6 3 2 8 9 3 5 2 4 9 12 7 10	put cargo 1 to cell 2 put cargo 2 to cell 1 put cargo 3 to cell 3 move cargo 1 from cell 2 to cell 3 put cargo 4 to cell 2 take cargo 3 from cell 3 take cargo 1 from cell 3 cargo 5 cannot be stored take cargo 2 from cell 1 take cargo 4 from cell 2

```
/*
    Задача: 177. Склад

    Решение: моделирование,  $O(n*m)$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/
```

```
#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
#define isz(x) (int)(x).size()
typedef std::pair<int,int> pii;
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
struct Robot {
    vi cap;
    std::vector<std::vector<pii>> data;
    Robot(int nCells) {
        data.resize(1+nCells);
        cap.resize(1+nCells);
    }
    void set_cap(int id, int c) {
        cap[id] = c;
    }

    int choose_cell(pii cargo) {
        pii best(INT_MAX, 0);
        for (int i = 1; i < isz(cap); ++i) {
            if (cap[i] - cargo.second >= 0) {
                best = std::min(best, pii(cap[i] - cargo.second, i));
            }
        }
        return best.second;
    }

    bool try_move(pii cargo) {
        auto best = std::make_tuple(
            INT_MAX,
            INT_MAX,
            INT_MAX,
            INT_MAX,
            INT_MAX,
            INT_MAX);
        for (int i = 1; i < isz(cap); ++i) {
            for (auto & curr : data[i]) {
```

```

        if (cap[i] + curr.second >= cargo.second) {
            for (int j = 1; j < isz(cap); ++j) {
                if (j == i) { continue; }
                if (cap[j] >= curr.second) {
                    auto temp = std::make_tuple(
                        curr.second,
                        cap[i] + curr.second,
                        cap[j] - curr.second,
                        curr.first,
                        j, i);
                    best = std::min(best, temp);
                }
            }
        }
    }
    if (std::get<0>(best) == INT_MAX) {
        return false;
    }
    auto moved = pii(std::get<3>(best), std::get<0>(best));
    const int i = std::get<5>(best);
    const int j = std::get<4>(best);
    move_cargo(moved, i, j);
    put_cargo(cargo, i);
    return true;
}

void put_cargo(pii cargo, int c) {
    data[c].push_back(cargo);
    cap[c] -= cargo.second;
    assert(cap[c] >= 0);
    std::cout << "put cargo " << cargo.first << " to cell " << c << std::endl;
}

void move_cargo(pii cargo, int from, int to) {
    auto it = std::find(all(data[from]), cargo);
    assert(it != data[from].end());
    cap[from] += cargo.second;
    data[from].erase(it);
    data[to].push_back(cargo);
    cap[to] -= cargo.second;
    assert(cap[to] >= 0 && cap[from] >= 0);
    std::cout << "move cargo " << cargo.first << " from cell " << from << " to cell " <
}

void pop(pii cargo) {
    bool was = false;
    for (int i = 1; i < isz(cap); ++i) {
        auto it = std::find(all(data[i]), cargo);
        if (it == data[i].end()) {
            continue;
        }
        was = true;
        data[i].erase(it);
        cap[i] += cargo.second;
        std::cout << "take cargo " << cargo.first << " from cell " << i << std::endl;
        break;
    }
    //assert(was);
}

void put(pii cargo) {
    int target = choose_cell(cargo);
    if (target == 0) {
        bool flag = try_move(cargo);
        if (!flag) {
            std::cout << "cargo " << cargo.first << " cannot be stored" << std::endl;
        }
        return;
    }
}

```

```

        }
        put_cargo(cargo, target);
    }
};

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (int nCells, nCargos; std::cin >> nCells >> nCargos; ) {
        Robot robot(nCells);
        for (int i = 1; i <= nCells; ++i) {
            int cap; std::cin >> cap;
            robot.set_cap(i, cap);
        }
        vi size(nCargos+1), tin(nCargos+1), tout(nCargos+1);
        std::set<pii> events;
        for (int i = 1; i <= nCargos; ++i) {
            std::cin >> size[i] >> tin[i] >> tout[i];
            events.insert(pii(tin[i], +i));
            events.insert(pii(tout[i], -i));
        }
        while (!events.empty()) {
            auto curr = *events.begin(); events.erase(events.begin());
            if (curr.second > 0) {
                const int id = curr.second;
                robot.put(pii(id, size[id]));
            } else {
                const int id = -curr.second;
                robot.pop(pii(id, size[id]));
            }
        }
    }
    return 0;
}

```

ЗАДАЧА №758

Веревочный мост

(Время: 1 сек. Память: 16 Мб Сложность: 58%)

Однажды N путешественников решили ночью пересечь по веревочному мосту быструю горную речку. Без освещения перейти мост невозможно. К счастью, у одного из них оказался с собой фонарик. Известно, что мост выдерживает только двоих, а скорости людей могут различаться. Если мост пересекают два человека с разной скоростью, то они вынуждены двигаться со скоростью самого медленного из них. Скорость движения каждого из путников известна.

Помогите путешественникам как можно быстрее перебраться через мост. Требуется написать программу, определяющую минимальное время, которое потребуется для такого перехода. Например, если N=4, а время, требуемое для перехода по мосту для каждого, составляет 5, 10, 20 и 25 минут соответственно, то наименьшее время, требуемое для пересечения моста, составит ровно 60 минут.

Входные данные

В первой строке входного файла INPUT.TXT содержится натуральное число N – количество путешественников ($N \leq 10^5$). Во второй строке располагаются N натуральных чисел – скорости всех путников, разделенные пробелом и не превосходящие 10^6 . Здесь под скоростью человека понимается время в минутах, необходимое для перехода через мост.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно целое число – минимально возможное время, необходимое путникам для пересечения моста.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 10 20	20
2	4 5 10 20 25	60

```
/*
    Задача: 758. Веревочный мост

    Решение: динамическое программирование, жадные алгоритмы, O(n*log(n))

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
typedef long long ll;
typedef std::vector<int> vi;
typedef std::vector<ll> vl;
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (int n; std::cin >> n; ) {
        vi a(n); for (auto &it : a) { std::cin >> it; }
```

```
std::sort(all(a));
vl answ(n);
answ[0] = a[0];
if (n >= 2) {
    answ[1] = a[1];
}
for (int i = 2; i < n; ++i) {
    answ[i] = std::min(
        answ[i-1] + a[0] + a[i],
        answ[i-2] + a[1] + a[0] + a[i] + a[1]
    );
}
std::cout << answ.back() << std::endl;
}
return 0;
}
```

ЗАДАЧА №810

Число - 2

(Время: 1 сек. Память: 16 Мб Сложность: 58%)

По заданным числам a , b , c и d , найдите наименьшее натуральное число n , большее $a \cdot c$, которое нельзя представить в виде произведения двух натуральных чисел u и v , таких, что $a \leq u \leq b$ и $c \leq v \leq d$.

Входные данные

Входной файл INPUT.TXT содержит одну строку, состоящую из натуральных чисел a , b , c , d ($1 \leq a \leq b \leq 10^6$, $1 \leq c \leq d \leq 10^6$).

Выходные данные

В выходной файл OUTPUT.TXT выведите искомое число n .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 2 1 2	3
2	1 2 3 5	7

```
/*
   "Число - 2": делители числа, простые числа
*/

#include <iostream>

int get(int value, int s1, int t1, int s2, int t2) {
    bool flag = false;
    while (!flag) {
        ++value;
        flag = true;
        for (int i = 1; i * i <= value; ++i) {
            const int j = value / i;
            if (i * j == value && (
                (s1 <= i && i <= t1 && s2 <= j && j <= t2) ||
                (s1 <= j && j <= t1 && s2 <= i && i <= t2))
            ) {
                flag = false;
                break;
            }
        }
    }
    return value;
}

int main() {
    int s1, t1, s2, t2;
    std::cin >> s1 >> t1 >> s2 >> t2;
    if (s1 == 1 && s2 == 1) {
        std::cout << get(std::max(t1, t2), s1, t1, s2, t2);
    } else if (s1 == 1) {
        std::cout << get(t2, s1, t1, s2, t2);
    } else if (s2 == 1) {
        std::cout << get(s1, t1, s2, t2, t2);
    }
}
```

```
        std::cout << get(t1, s1, t1, s2, t2);
    } else {
        std::cout << 1LL*s1*s2+1;
    }
    return 0;
}
```

ЗАДАЧА №918

График

(Время: 1 сек. Память: 16 Мб Сложность: 58%)

Антон работает курьером. У него много заказов. На выполнение одного заказа у Антона уходит ровно один день. Для каждого заказа определена стоимость и срок его выполнения (количество дней, оставшихся до запланированного дня выполнения заказа). Однажды проснувшись, Антон изучил свой график и понял, что возможно он не сможет выполнить все заказы, и его могут уволить. Поэтому он решил выполнить лишь некоторые из них так, чтобы при этом получить максимальный доход.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число N ($1 \leq N \leq 1000$) – количество заказов. Затем в N строках описаны данные каждого заказа T_i и C_i (натуральные числа, не превосходящие 10^5). Где T_i – последний день, в который еще можно выполнить заказ, C_i – вознаграждение за выполнение заказа.

Выходные данные

В выходной файл OUTPUT.TXT выведите максимальное вознаграждение, которое можно получить, выполняя заказы.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 1 10 2 12	22
2	3 1 10 1 20 3 24	44

```
/*
    Задача: 918. График

    Решение: жадный алгоритм, динамическое программирование, двумерное дп,  $O(n^2)$ 

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/
#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
#define remax(x, y) (x) = (x) < (y) ? (y) : (x)
struct Order { int last, cost; };
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (int n; std::cin >> n; ) {
        std::vector<Order> orders(n);
        for (auto & it : orders) { std::cin >> it.last >> it.cost; }
        std::sort(all(orders), [&](const Order& a, const Order& b) {
            return a.last < b.last || (a.last == b.last && a.cost > b.cost);
        });
    }
```



```

// dp[seen][taken]
std::vector<std::vector<int>> dp(1+n, std::vector<int>(1+n));
for (int i = 0; i < n; ++i) {
    for (int take = 0; take <= i; ++take) {
        if (take+1 <= orders[i].last) {
            remax(dp[i+1][take+1], dp[i][take] + orders[i].cost);
        }
        remax(dp[i+1][take], dp[i][take]);
    }
}
std::cout << *std::max_element(all(dp[n])) << std::endl;
}
return 0;
}

```

ЗАДАЧА №1386

Связанное множество

(Время: 1 сек. Память: 16 Мб Сложность: 58%)

Даны несколько точек на плоскости, некоторые из которых соединены отрезками. Множество точек называется связанным, если из любой его точки можно перейти в любую точку, перемещаясь только по отрезкам (переходить с отрезка на отрезок возможно только в точках исходного множества). Можно за определенную плату добавлять новые отрезки (стоимость добавления равна длине добавляемого отрезка). Требуется за минимальную стоимость сделать данное множество связанным.

Входные данные

В первой строке входного файла INPUT.TXT содержится одно целое число N ($1 \leq N \leq 50$) – количество точек. Далее в N строках записано по 2 натуральных числа – координаты точек (координаты не превышают 100). Все точки различны. Далее дано число M – количество уже существующих отрезков. В следующих M строках записаны по 2 числа – номера начала и конца соответствующего отрезка.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное число – минимально возможную стоимость дополнения с точностью, не худшей, чем 5 знаков после запятой.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 1 1 1 2 10 1 1 2 1	9.0

```
/*
    Задача: 1386. Связанное множество

    Решение: DSU, алгоритм Краскала, минимальное остовное дерево,  $O(n^2 \log(n))$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cmath>

struct Edge {
    int u, v; double cost;

    Edge(int u_ = 0, int v_ = 0, double c_ = 0.0) : u(u_), v(v_), cost(c_) { }
};

bool operator<(const Edge& a, const Edge& b) {
```

```

    if (a.cost < b.cost) return true;
    if (a.cost > b.cost) return false;
    return a.u < b.u || (a.u == b.u && a.v < b.v);
}

struct DSU {
    std::vector<int> parent, size;
    DSU(int n) : parent(1+n,0), size(1+n,1) {
        for (int i = 1; i <= n; ++i) parent[i] = i;
    }
    int get_parent(int a) {
        return parent[a] == a ? a : parent[a] = get_parent(parent[a]);
    }
    void union_sets(int a, int b) {
        a = get_parent(a), b = get_parent(b);
        if (a != b) {
            if (size[a] < size[b]) std::swap(a,b);
            parent[b] = a;
            size[a] += size[b];
        }
    }
};

int main() {
    int n; scanf("%d", &n);
    std::vector<int> x(1+n), y(1+n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d %d", &x[i], &y[i]);
    }
    int m; scanf("%d", &m);
    std::vector<Edge> edges(m);
    for (auto& e : edges) {
        scanf("%d %d", &e.u, &e.v);
    }
    for (int i = 1; i <= n; ++i) {
        for (int j = i+1; j <= n; ++j) {
            int dx = x[i] - x[j], dy = y[i] - y[j];
            edges.push_back(Edge(i,j,std::sqrt(dx*dx+dy*dy)));
        }
    }
    std::sort(edges.begin(), edges.end());
    DSU dsu(n); double answ = 0;
    for (auto& e : edges) {
        int pu = dsu.get_parent(e.u), pv = dsu.get_parent(e.v);
        if (pu != pv) {
            answ += e.cost;
            dsu.union_sets(e.u,e.v);
        }
    }
    printf("%.5f", answ);
    return 0;
}

```

ЗАДАЧА №565

Сушка

(Время: 1 сек. Память: 16 Мб Сложность: 59%)

Тетя Люба только что постирала все белье и теперь перед ней стоит непростая задача - как его высушить, чтобы ни одна вещь не успела испортиться. Сразу после стирки, i -я постиранная вещь имеет влажность w_i . Если она сушится на веревке, то за минуту ее влажность уменьшается на 1, а если на батарее - то на r (если влажность была меньше r , то она становится равной 0). Причем веревок у тети Любы много (хватает для одновременной сушки всех вещей), а батарея только одна, причем такая маленькая, что на ней нельзя сушить две вещи одновременно. i -я вещь испортится, если не высохнет за время d_i . Помогите тете Любе составить план, когда какую вещь повесить на батарею.

Входные данные

Первая строка входного файла INPUT.TXT содержит целые числа n ($1 \leq n \leq 10^5$) - количество мокрых вещей, и r ($1 \leq r \leq 10^9$). Следующие n строк содержат описания постиранных вещей – пары чисел w_i и d_i ($1 \leq w_i, d_i \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите план сушки в виде пар целых чисел t_i и k_i , где t_i - время в минутах от начала сушки, а k_i - номер вещи, которую нужно повесить на батарею в этот момент. Выводите пары в порядке увеличения t_i . Пар не должно быть больше 10^5 . Не выводите числа больше 10^9 . Если высушить все вещи невозможно, выведите слово «Impossible».

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 3 2000 1000 2000 2000 2500 1500	0 3 500 1 1000 3
2	3 3 2000 1000 2000 1000 2000 1000	Impossible

```
/*
Задача: 565. Сушка

Решение: жадный алгоритм, сортировка,  $O(n \log(n))$ 

Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
struct Pair {
    int w, d, id;
};
```

```

bool operator<(const Pair& a, const Pair& b) {
    return a.d < b.d || (a.d == b.d && (a.w > b.w || (a.w == b.w && a.id < b.id)));
}
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    for (int n, r, t = 0; std::cin >> n >> r;) {
        std::vector<Pair> arr(n);
        typedef std::pair<int,int> pii;
        std::vector<pii> answ;
        for (int i = 0; i < n; ++i) {
            std::cin >> arr[i].w >> arr[i].d;
            arr[i].id = i+1;
        }
        std::sort(all(arr));
        if (r == 1) {
            bool ok = true;
            for (auto it : arr) {
                ok = ok && (it.d >= it.w);
            }
            if (!ok) {
                std::cout << "Impossible" << std::endl;
            }
            continue;
        }
        bool ok = true;
        for (auto it : arr) {
            // w - (d - k) - k * r <= 0
            // w - d + k - k * r <= 0
            // w - d + k * (1 - r) <= 0
            const int d = it.d;
            const int w = it.w;
            int k = (d - w) / (1-r);
            while (w - d - k * (r - 1) > 0) { ++k; }
            while (w - d - (r - 1) * (k-1) <= 0) { --k; }
            assert(w - d + k * (1 - r) <= 0);
            if (t + k > d) {
                ok = false;
                break;
            }
            if (k > 0) {
                answ.push_back(pii(t, it.id));
                t += k;
            }
        }
        if (!ok) {
            std::cout << "Impossible" << std::endl;
            continue;
        }
        for (auto it : answ) {
            std::cout << it.first << ' ' << it.second << '\n';
        }
    }
    return 0;
}

```

ЗАДАЧА №584

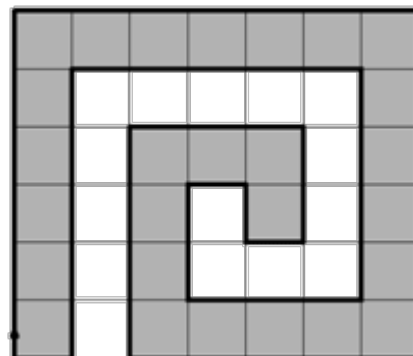
Спираль - 2

(Время: 1 сек. Память: 16 Мб Сложность: 59%)

При обучении школьников младших классов программированию часто используется язык «Лого», позволяющий рисовать на экране картинки хвостом виртуальной черепашки.

Рассмотрим упрощенную версию этого языка, в которой разрешается подавать черепашке следующие команды: переместиться вперед на некоторое количество сантиметров, рисуя за собой линию, повернуть налево на 90 градусов и повернуть направо на 90 градусов.

Вам предлагается написать на этом языке программу, изображающую контур спирали.



Спираль - это множество клеток, которое строится по следующим правилам. Рассмотрим прямоугольник, состоящий из $m \times n$ единичных клеток. Закрасим некоторые из его клеток в следующей последовательности.

Начнем красить левый столбец снизу вверх. Когда продвижение невозможно, то есть либо следующая клетка лежит вне прямоугольника, либо соприкасается с закрашенной ранее, направление движения поворачивается по часовой стрелке на 90 градусов. Если и после этого поворота дальнейшее движение невозможно, процесс заканчивается. Множество закрашенных таким образом клеток и образует спираль.

Входные данные

Входной файл INPUT.TXT содержит два целых числа m и n ($1 \leq m, n \leq 20\,000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите последовательность команд, в результате выполнения которой будет нарисован контур ломаной. Напомним формат, в котором следует выводить результат. Первая строка должна содержать k - количество команд. Последующие k строк задают либо длину отрезка d , который надо провести, в формате $f\ d$, либо направление поворота - l для поворота налево и r для поворота направо. Команды рисования отрезка и поворота должны чередоваться, начинаясь и заканчиваясь командой рисования отрезка. Считайте, что начало первого отрезка совпадает с левым нижним углом прямоугольника. При этом контур должен быть выведен в направлении обхода по часовой стрелке, и никакая его часть ненулевой длины не должна быть нарисована дважды.

Пример

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	6 7	31
		f6
		r
		f7
		r
		f6
		r
		f5
		r
		f4
		r
		f3
		r
		f2
		r
		f1
		r
		f1
		l
		f1
		l
		f2
		l
		f3
		l
		f4
		l
		f5
		l
		f5
		r
		f1

```

/*
    Задача: 584. Спираль - 2

    Решение: динамическое программирование, рекурсия, конструктив, O(n+m)

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>

const int UP = 1, DOWN = 3, LEFT = 0, RIGHT = 2;

struct Solution {
    std::vector<std::string> answ;
    Solution(const int nR, const int nC) {
        solve(nR,nC);
        rotr();
        move(1);
    }
    void move(int cnt) {
        if (!answ.empty() && answ.back()[0] == 'f') {
            int tmp;
            sscanf(answ.back().c_str(), "%*s %d", &tmp);
            cnt += tmp;
            answ.pop_back();
        }
        answ.push_back("f " + std::to_string(cnt));
    }
    void rotr() {

```

```

        assert(answ.size() > 0u);
        if (answ.back() == "l") {
            answ.pop_back();
            return;
        }
        answ.push_back("r");
    }
    void rotl() {
        assert(answ.size() > 0u);
        if (answ.back() == "r") {
            answ.pop_back();
            return;
        }
        answ.push_back("l");
    }
    void solve(int nR, int nC) {
        assert(nC > 0);
        if (nR == 0) {
            rotr();
            move(nC);
            rotr();
            return;
        }
        move(nR);
        rotr();
        if (nC == 2) {
            move(2);
            rotr();
            move(1);
            rotr();
            move(1);
            rotl();
            if (nR-1 > 0) {
                move(nR-1);
            }
        } else if (nC > 2) {
            move(2);
            solve(nC - 2, nR);
            move(1);
            rotl();
            if (nR-1 > 0) {
                move(nR-1);
            }
        } else {
            assert(nC == 1);
            move(1);
            rotr();
            move(nR);
        }
    }
};

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int nR, nC; std::cin >> nR >> nC;
    Solution solution(nR,nC);
    std::cout << solution.answ.size() << "\n";
    for (auto &it : solution.answ) {
        std::cout << it << '\n';
    }
    return 0;
}

```


ЗАДАЧА №774

Шляпа

(Время: 1 сек. Память: 16 Мб Сложность: 59%)

Фокусник Аркадий Великолепный разочаровался в доходности извлечения из цилиндра кроликов и голубей и решил заняться контрабандой драгоценностей. Его шляпа-цилиндр, высотой H и радиусом основания R , поможет ему в этом, ибо ни один таможенник не сможет обнаружить там ничего, кроме кучки заячьего помета. Перед Аркадием, однако, встал другой вопрос: что и куда везти? Разные драгоценности пользуются разной популярностью в разных городах; кроме того, драгоценности различаются между собой по размеру упаковки. Ценности хранятся в кубических контейнерах, которые размещаются в цилиндре так, что их дно параллельно дну цилиндра. Контейнеры могут помещаться друг на друга, образуя слои, причем в одном слое могут находиться только контейнеры с одинаковой длиной ребра. Из-за чисто технических ограничений более пяти контейнеров в один слой поместить нельзя. Контейнеры не должны выступать за границы цилиндра.

Аркадий выяснил, какой длины ребро у контейнеров всех типов драгоценностей, и по какой цене можно продать те или иные драгоценности в тех или иных городах. Для начала фокусник решил совершить поездку в один город, взяв с собой драгоценности одного типа. Помогите ему рассчитать прибыль от сделки при такой упаковке цилиндра контейнерами, когда помещается их максимально возможное количество.

Входные данные

Первая строка входного файла INPUT.TXT содержит четыре числа, разделенных пробелами: N – количество типов драгоценностей ($1 \leq N \leq 10$), M – количество городов ($1 \leq M \leq 10$), действительные числа R ($1.0 \leq R \leq 100.0$) и H ($1.0 \leq H \leq 100.0$).

Во второй строке расположены N действительных чисел a_1, a_2, \dots, a_N – длина ребер контейнеров каждого типа драгоценностей, $0.5 \leq a_i \leq 100.0$. Далее во входном файле M строк – по одной для каждого города. Каждая из строк содержит N целых чисел Q_{ij} – стоимость драгоценностей каждого типа в текущем городе ($0 \leq Q_{ij} \leq 1000$).

Все вещественные числа во входных данных имеют не более 6 значащих разрядов.

Выходные данные

В выходной файл OUTPUT.TXT выведите максимальную прибыль, которую фокусник может извлечь в текущей ситуации.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 2 1 2 1 2 10 20 30 40	60

2	3 3 1 2	600
	1 0.5 1	
	10 10 10	
	20 20 20	
	30 30 30	

```

/*
    Задача: 774. Шляпа

    Решение: геометрия, описанная окружность,  $O(n*m)$ 

    Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>

typedef long double ld;

typedef long long ll;

const ld PI = std::acos(ld(-1));

const ld EPS = 1e-16L;

bool less(ld a, ld b) {
    return a <= b * (1 + EPS);
}

bool check(const int n, ld a, ld r) {
    ld d = 0;
    switch (n) {
        case 1: d = a * std::sqrt(ld(1)/2); break;
        case 2: d = a * std::sqrt(ld(5)/4); break;
        case 3: d = a * std::sqrt(ld(256) + ld(169)) / 16; break;
        case 4: d = a * std::sqrt(ld(2)); break;
        case 5: d = a * std::sqrt(ld(5)/2); break;
    };
    return less(d, r);
}

auto max_number(ld a, ld r, ld h) {
    ll answ = 0;
    for (int i = 1; i <= 5; ++i) {
        if (check(i, a, r)) {
            answ = std::max(answ, ll(h / a + EPS) * i);
        }
    }
    return answ;
}

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int n, nC; ld r, h;
    while (std::cin >> n >> nC >> r >> h) {
        std::vector<ld> side(n);
        for (auto &it : side) { std::cin >> it; }
        ll answ = 0;
        for (int i = 0; i < nC; ++i) {
            for (int j = 0; j < n; ++j) {
                ll q; std::cin >> q;
                answ = std::max(answ, q * max_number(side[j], r, h));
            }
        }
        std::cout << answ << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №857

Abracadabra

(Время: 5 сек. Память: 64 Мб Сложность: 59%)

Строка *s* называется супрефиксом для строки *t*, если *t* начинается с *s* и заканчивается на *s*. Например, «абга» является супрефиксом для строки «abracadabra». В частности, сама строка *t* является своим супрефиксом. Супрефиксы играют важную роль в различных алгоритмах на строках.

В этой задаче требуется решить обратную задачу о поиске супрефикса, которая заключается в следующем. Задан словарь, содержащий *n* слов *t*₁, *t*₂, ..., *t*_{*n*} и набор из *m* строк-образцов *s*₁, *s*₂, ..., *s*_{*m*}. Необходимо для каждой строки-образца из заданного набора найти количество слов в словаре, для которых эта строка-образец является супрефиксом.

Требуется написать программу, которая по заданному числу *n*, *n* словам словаря *t*₁, *t*₂, ..., *t*_{*n*}, заданному числу *m* и *m* строкам-образцам *s*₁, *s*₂, ..., *s*_{*m*} вычислит для каждой строки-образца количество слов из словаря, для которых эта строка-образец является супрефиксом.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число *n* (1 ≤ *n* ≤ 200 000). Последующие *n* строк содержат слова *t*₁, *t*₂, ..., *t*_{*n*}, по одному слову в каждой строке. Каждое слово состоит из строчных букв английского алфавита. Длина каждого слова не превышает 50. Суммарная длина всех слов не превышает 10⁶. Словарь не содержит пустых слов.

Затем следует строка, содержащая целое число *m* (1 ≤ *m* ≤ 200 000). Последующие *m* строк содержат строки-образцы *s*₁, *s*₂, ..., *s*_{*m*}, по одной на каждой строке. Каждая строка-образец состоит из строчных букв английского алфавита. Длина каждой строки-образца не превышает 50. Суммарная длина всех строк-образцов не превышает 10⁶. Никакая строка-образец не является пустой строкой.

Выходные данные

В выходной файл OUTPUT.TXT выведите *m* чисел, по одному на строке. Для каждой строки-образца в порядке, в котором они заданы во входном файле, следует вывести количество слов словаря, для которых она является супрефиксом.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4	4 2 0
	abacaba	
	abracadabra	
	aa	
	abra	
	3	
	a	
	abra	
	abac	

Задача: 857. Abracadabra

Решение: полиномиальный хэш, сортировка, бинарный поиск, строки, $O(n \log(n))$

Автор: Dmitry Kozyrev, <https://github.com/dmkz> , dmkozyrev@rambler.ru

*/

```
#include <stdio.h>
#include <string>
#include <vector>
#include <random>
#include <chrono>
#include <array>
#include <iostream>
#include <algorithm>

typedef unsigned long long ull;

// Generate random seed:
inline ull gen_seed() {
    return ull(new ull) ^ ull(std::chrono::high_resolution_clock::now().time_since_epoch()).
}

// Generate random base in (before, after) open interval:
int gen_base(int before, int after) {
    std::mt19937 gen(gen_seed());
    std::uniform_int_distribution<int> dist(before+2, after-1);
    int base = dist(gen);
    return base % 2 == 0 ? base - 1 : base;
}

struct PolyHash {
    // ----- Static variables -----
    static const ull mod = (ull(1) << 61) - 1; // prime mod of hashing
    static int base; // odd base of hashing
    static std::vector<ull> pow; // powers of base modulo mod;

    // ----- Static functions -----
    static inline ull add(ull a, ull b) {
        // Calculate (a + b) % mod, 0 <= a < mod, 0 <= b < mod
        return (a += b) < mod ? a : a - mod;
    }

    static inline ull sub(ull a, ull b) {
        // Calculate (a - b) % mod, 0 <= a < mod, 0 <= b < mod
        return (a -= b) < mod ? a : a + mod;
    }

    static inline ull mul(ull a, ull b){
        // Calculate (a * b) % mod, 0 <= a < mod, 0 <= b < mod
        ull l1 = (uint32_t)a, h1 = a >> 32, l2 = (uint32_t)b, h2 = b >> 32;
        ull l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
        ull ret = (l & mod) + (l >> 61) + (h << 3) + (m >> 29) + (m << 35 >> 3) + 1;
        ret = (ret & mod) + (ret >> 61);
        ret = (ret & mod) + (ret >> 61);
        return ret-1;
    }

    // ----- Variables of class -----
    std::vector<ull> pref; // polynomial hash on prefix

    // Constructor from string:
    PolyHash(const std::string& s)
        : pref(s.size()+1u, 0)
    {
        // Pre-calculate powers of base:
        while (pow.size() <= s.size()) {
            pow.push_back(mul(pow.back(), base));
        }
    }
};
```

```

        // Calculate polinomial hash on prefix:
        for (int i = 0; i < (int)s.size(); ++i) {
            pref[i+1] = add(mul(pref[i], base), s[i]);
        }
    }

    // Get hash from [pos, pos+len-1] segment of string
    inline ull operator()(const int pos, const int len) const {
        return sub(pref[pos+len], mul(pref[pos], pow[len]));
    }

};

// Init static variables of class PolyHash:
int PolyHash::base((int)1e9+7);
std::vector<ull> PolyHash::pow{1};

int main() {
    // Generate random base:
    PolyHash::base = gen_base(256, 2e9);

    // Prepare hashes from strings:
    std::vector<ull> hashes;
    int nWords; scanf("%d", &nWords);
    for (int i = 0; i < nWords; ++i) {
        char buf[1+50]; scanf("%50s", buf);
        std::string s(buf);
        PolyHash hash(s);
        for (int len = 1; len <= (int)s.size(); ++len) {
            if (hash(0, len) == hash((int)s.size()-len, len)) {
                hashes.push_back(hash(0, len));
            }
        }
    }
    // Sorting hashes:
    std::sort(hashes.begin(), hashes.end());
    // Answer on queries:
    int nQueries; scanf("%d", &nQueries);
    for (int i = 0; i < nQueries; ++i) {
        char buf[1+50]; scanf("%50s", buf);
        std::string s(buf);
        PolyHash hash(s);
        auto p = std::equal_range(hashes.begin(), hashes.end(), hash(0, (int)s.size()));
        printf("%d\n", int(p.second - p.first));
    }
    return 0;
}

```

ЗАДАЧА №939

Золотоискатели

(Время: 1 сек. Память: 16 Мб Сложность: 59%)

Артель золотоискателей, состоящая из трех человек, добыла N самородков. Один из золотоискателей решил уехать, не ожидая конца вахты, так как у него на Большой земле родился сын.

Артельщики решили выдать отъезжающему ровно третью часть добытого золота. Выбрать такой набор камней оказалось непростой задачей. Вам надо написать программу, которая находит набор самородков, вес которого составляет третью часть от веса добытого золота, либо определить, что это невозможно сделать.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число N - количество добытых самородков ($1 \leq N \leq 100$). Во второй строке записано N целых чисел m_1, m_2, \dots, m_n ($1 \leq m_i \leq 100$), разделенные пробелами - веса добытых самородков.

Выходные данные

В первую строку выходного файла OUTPUT.TXT следует вывести число K - количество самородков в наборе, а в следующей строке - K чисел, задающих номера самородков. В случае неоднозначного ответа выведите любой. В случае, если выделить ровно третью часть невозможно, выведите один ноль

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	8 1 3 4 1 2 5 1 1	3 1 3 4
2	3 1 3 6	0

```
/*
    Задача: 939. Золотоискатели

    Решение: динамическое программирование, двумерное дп,  $O(n^2 \cdot w)$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/
```

```
#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
typedef std::pair<int,int> pii;
typedef std::vector<pii> vpii;
typedef std::vector<vpii> vvpii;
typedef std::vector<int> vi;
int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0);
    for (int n; std::cin >> n; ) {
        vi w(n);
        for (auto &it : w) { std::cin >> it; }
```

```

const int max = *std::max_element(all(w)) * n;
const int sum = std::accumulate(all(w), 0);
if (sum / 3 * 3 != sum) {
    std::cout << 0 << std::endl;
    continue;
}
vvp<pii> dp(1+n, v<pii>(1+max, pii(-1, -1)));
dp[0][0] = pii(0, -1);
for (int i = 1; i <= n; ++i) {
    const int t = w[i-1];
    for (int s = t; s <= max; ++s) {
        if (dp[i-1][s-t].first == -1) { continue; }
        dp[i][s] = pii(i-1, i-1);
    }
    for (int s = 0; s <= max; ++s) {
        if (dp[i][s].first == -1) {
            dp[i][s] = dp[i-1][s];
        }
    }
}
int s = sum / 3;
if (dp[n][s].first == -1) {
    std::cout << 0 << std::endl;
    continue;
}
pii last = dp[n][s];
vi answ;
while (last.second != -1) {
    answ.push_back(last.second+1);
    const int wt = w[last.second];
    s -= wt;
    assert(s >= 0);
    last = dp[last.first][s];
}
assert(s == 0);
std::cout << answ.size() << "\n";
for (auto it : answ) { std::cout << it << ' '; }
std::cout << std::endl;
}
return 0;
}

```

ЗАДАЧА №995

Голова на плечах

(Время: 3 сек. Память: 16 Мб Сложность: 59%)

Изготовитель всемирно известного шампуня, компания «Голова на плечах» всерьез заботится о качестве своей продукции. В частности, она постоянно улучшает различные показатели своего шампуня, с целью чего постоянно проводит исследования. Но, согласно законодательству, на человеке проводить исследования нельзя, а проводить исследования на животных компания считает ниже своего достоинства. Поэтому ученые, работающие в компании, разработали математическую модель человека и проводят исследования на ней.

Человек, согласно этой модели, состоит из головы и плеч. Голова представляет собой окружность с центром в точке $(0,0)$ и радиусом R , а плечи бесконечную прямую $y = -K$, где $R < K$.

Объектом изучения исследователей являются волосы. Каждый волос в данной модели представлен отрезком, начинающимся на голове (строго на окружности) и заканчивающимся на плечах (строго на прямой). При этом ни один волос не имеет с окружностью головы более одной общей точки. В данный момент ученые озабочены проблемой секущихся волос. Пара волос называется секущейся, если соответствующие этим волосам отрезки имеют общую точку.

Дана математическая модель человека. Найдите количество секущихся пар волос.

Входные данные

В первой строке входного файла заданы два целых числа R, K ($1 \leq R < K \leq 10^3$). Во второй строке записано целое число N ($0 \leq N \leq 10^5$) количество волос в модели человека. В следующих N строках находится по 4 вещественных числа X_h, Y_h, X_s, Y_s — координаты начала и конца очередного волоса. Первая пара чисел соответствует концу, лежащему на окружности головы, вторая пара соответствует концу, лежащему на плечах.

Гарантируется, что никакой волос не имеет с окружностью головы более одной общей точки. Также гарантируется, что среди начальных и конечных точек нет одинаковых.

Выходные данные

В выходной файл OUTPUT.TXT выведите число секущихся пар волос.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 2 3 0 -1 -2 -2 1 0 2 -2 -1 0 -1 -2	1

2	1 10	0
	3	
	-1 0 -1 -10	
	0 -1 0 -10	
	1 0 1 -10	

```

/*
"Голова на плечах": сортировка по полярному углу, дерево отрезков,  $O(n \log(n))$ 
*/

```

```

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>
#include <cmath>
#include <string>

struct SegmentTree {

    const int size;

    std::vector<int> data;

    void build(int v, int l, int r, const std::vector<int>& arr) {
        if (l == r) {
            data[v] = arr[l];
        } else {
            int m = (l + r) / 2;
            build(2*v+1, l, m, arr);
            build(2*v+2, m+1, r, arr);
            data[v] = data[2*v+1]+data[2*v+2];
        }
    }

    void set(int v, int l, int r, int pos, int value) {
        if (l == r) {
            data[v] = value;
        } else {
            int m = (l + r) / 2;
            if (pos <= m) {
                set(2*v+1, l, m, pos, value);
            } else {
                set(2*v+2, m+1, r, pos, value);
            }
            data[v] = data[2*v+1]+data[2*v+2];
        }
    }

    void set(int pos, int value) {
        set(0, 0, size-1, pos, value);
    }

    int get(int v, int l, int r, int ql, int qr) const {
        if (l == ql && r == qr) {
            return data[v];
        } else {
            int m = (l + r) / 2;
            if (qr <= m) {
                return get(2*v+1, l, m, ql, qr);
            } else if (ql > m) {
                return get(2*v+2, m+1, r, ql, qr);
            } else {
                return get(2*v+1, l, m, ql, m) + get(2*v+2, m+1, r, m+1, qr);
            }
        }
    }

    int get(int left, int right) const {

```

```

        return get(0, 0, size-1, left, right);
    }

    SegmentTree(const std::vector<int>& arr) : size((int)arr.size()) {
        int pow = 1;
        while (pow < size) pow *= 2;
        data.resize(2 * pow);
        build(0, 0, size-1, arr);
    }
};

typedef double Real;

const Real PI = std::acos(-1.0L);

struct Point {
    Real x, y; int id;

    inline Point rotate(Real angle) const {
        return Point{
            std::cos(angle) * x - std::sin(angle) * y,
            std::sin(angle) * x + std::cos(angle) * y,
            id
        };
    }

    inline Real angle() const {
        return std::atan2(y, x);
    }
};

int main() {
    int n; scanf("%d %d %d", &n);
    if (n == 0) {
        printf("0");
        return 0;
    }
    std::vector<Point> circle, line;
    circle.reserve(n);
    line.reserve(n);
    for (int i = 0; i < n; ++i) {
        double x1, y1, x2, y2;
        scanf("%lf %lf %lf %lf", &x1, &y1, &x2, &y2);
        circle.push_back(Point{Real(x1), Real(y1), i});
        line.push_back(Point{Real(x2), Real(y2), i});
    }

    std::sort(line.begin(), line.end(), [](const Point& a, const Point& b) {
        assert(a.x != b.x);
        return a.x < b.x;
    });

    std::sort(circle.begin(), circle.end(), [](const Point& a, const Point& b) {
        return a.rotate(PI/2).angle() < b.rotate(PI/2).angle();
    });

    int pos = 0;
    std::vector<int> f(n);
    for (auto& p : line) {
        f[p.id] = pos++;
    }

    SegmentTree st(std::vector<int>(n, 0));

    long long answ = 0;
    for (auto& p : circle) {
        answ += st.get(f[p.id], n-1);
        st.set(f[p.id], 1);
    }
}

```

```
    printf("%s", std::to_string(answ).c_str());  
    return 0;  
}
```

ЗАДАЧА №572

Технология программирования

(Время: 1 сек. Память: 16 Мб Сложность: 60%)

Толик придумал новую технологию программирования. Он хочет уговорить друзей использовать ее. Однако все не так просто. i -й друг согласится использовать технологию Толика, если его авторитет будет не меньше a_i (авторитет выражается целым числом). Как только он начнет ее использовать, к авторитету Толика прибавится число b_i (попадаются люди, у которых $b_i < 0$). Помогите Толику наставить на путь истинный как можно больше своих друзей.

Входные данные

На первой строке входного файла INPUT.TXT содержатся два числа: n ($1 \leq n \leq 1000$) – количество друзей у Толика, и первоначальный авторитет Толика. Следующие n строк содержат пары чисел a_i и b_i . Все числа целые, по модулю не больше 10^6 .

Выходные данные

В выходной файл OUTPUT.TXT выведите число m - максимальное число друзей, которых может увлечь Толик, и затем m чисел - номера друзей в том порядке, в котором их нужно агитировать.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	5 1 1 3 6 -5 6 -4 2 2 2 -1	4 1 4 3 5

```
/*
    Задача: 572. Технология программирования
    Решение: жадный алгоритм, сортировка, динамическое программирование,  $O(n^2)$ 
*/
#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
#define isz(x) (int)(x).size()
typedef std::vector<int> vi;
template<typename A, typename B> A& remax(A& a, B b) { return a = (a < b) ? b : a; }
struct Pair { int a, b, i; };
struct State {
    int res; int16_t id, prev_i, prev_j;
    State(int res_ = INT_MIN, int16_t id_ = 0, int16_t prev_i_ = -1, int16_t prev_j_ = -1)
        : res(res_), id(id_), prev_i(prev_i_), prev_j(prev_j_) { }
};
bool operator<(const State& a, const State& b) {
    return a.res < b.res;
}
int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0);
```

```

for (int n, st; std::cin >> n >> st; ) {
    std::vector<Pair> pos, neg;
    vi answ;
    for (int i = 0, a, b; i < n; i++) {
        std::cin >> a >> b;
        if (b >= 0) {
            pos.push_back(Pair{a, b, i+1});
        } else {
            neg.push_back(Pair{a, b, i+1});
        }
    }
    std::stable_sort(all(pos), [](auto lhs, auto rhs) { return lhs.a < rhs.a; });
    for (auto it : pos) {
        if (it.a <= st) {
            answ.push_back(it.i);
            st += it.b;
        }
    }
    std::stable_sort(all(neg), [](auto lhs, auto rhs) { return lhs.a + lhs.b > rhs.a +
    // dp[visited][taken];
    std::vector<std::vector<State>> dp(1+n, std::vector<State>(1+n, State()));
    dp[0][0] = State(st);
    n = isz(neg);
    for (int i = 0; i < n; i++) {
        int visited = i;
        for (int taken = 0; taken <= n; ++taken) {
            remax(dp[visited+1][taken], dp[visited][taken]);
        }
        for (int taken = 0; taken < n; ++taken) {
            auto prev = dp[visited][taken];
            if (prev.res >= neg[i].a) {
                prev.id = neg[i].i;
                prev.prev_i = visited;
                prev.prev_j = taken;
                prev.res += neg[i].b;
                remax(dp[visited+1][taken+1], prev);
            }
        }
    }
    int i = 0, j = 0;
    for (int taken = 0; taken <= n; ++taken) {
        if (dp[n][taken].res > INT_MIN) {
            i = n; j = taken;
        }
    }
    vi stack;
    while (i > 0 && j > 0) {
        stack.push_back(dp[i][j].id);
        int pi = dp[i][j].prev_i;
        int pj = dp[i][j].prev_j;
        i = pi; j = pj;
    }
    std::reverse(all(stack));
    answ.insert(answ.end(), all(stack));
    std::cout << isz(answ) << "\n";
    for (auto it : answ) { std::cout << it << ' '; }
    std::cout << std::endl;
}
return 0;
}

```

ЗАДАЧА №644

Временной ключ-2

(Время: 1 сек. Память: 16 Мб Сложность: 60%)

Практически каждый уважающий себя программист знает, что для полного успеха зачастую мало написать программный продукт, его также успешно надо уметь продать, и тем более также успешно защитить от взлома, а соответственно и от несанкционированного распространения.

Многие годы основным способом защиты программного обеспечения от незаконного распространения было использование, так называемого, активационного ключа. Вся проблема заключалась и заключается в том, что зачастую используется статический ключ, то есть активационный ключ для конкретного программного продукта не зависит ни от каких параметров и всегда является неизменным.

Знаменитая компания "Gold&Silver Soft" решилась на революционный шаг – было решено разработать принципиально новый способ динамической генерации активационного ключа. В данном алгоритме ключ зависит от времени и меняется каждую минуту, что существенно затрудняет взлом.

Будем считать, что активационным ключом является обычное целое положительное число. В данной версии алгоритма значение ключа на следующей минуте целиком и полностью зависит от значения ключа в текущий момент. Если в данный момент ключ равен N , то через минуту он будет равен $N + S(N)$, где $S(N)$ – это число, называемое контрольной суммой числа N и равняется количеству единиц в двоичной записи числа N . То есть если $N = 6$, то в следующую минуту значение ключа будет равно 8, если быть точнее, то $N' = N + S(N) = 6 + S(6) = 6_{10} + 110_2 = 8$.

Будем считать, что на данный момент времени значение ключа равно N , вашей задачей является вычислить значение ключа через K минут.

Входные данные

В первой и единственной строке входного файла INPUT.TXT находятся два натуральных числа – N ($1 \leq N \leq 2 \times 10^9$) и K ($1 \leq K \leq 2 \times 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число – значение активационного ключа через K минут, учитывая, что на данный момент времени значение ключа равно N .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 1	2
2	1 10	22
3	2 3	7

/*

После некоторого количества явных итераций от считанного числа мы получим число, которое так как разница между соседними числами не больше 37, то число явных итераций не такое. Предподсчитаем ряд для 1 с фиксированным шагом и решим задачу для 1.

*/

```

#include <iostream>
#include <fstream>
#include <cstdint>
#include <vector>
#include <algorithm>
#include <cassert>

typedef int64_t Int;

// Шаг предподсчета:
const Int STEP = 1024*1024;

// Опорные значения из 1 с шагом STEP:
std::vector<Int> numbers = {1,11751403,24708454,38020169,51644790,66144780,79275277,9333393

// Получение следующего числа:
inline Int next(Int number) {
    return number + __builtin_popcountll(number);
}

// Наивное решение задачи:
inline Int solve(Int number, Int times) {
    for (Int i = 0; i < times; ++i) {
        number = next(number);
    }
    return number;
}

// Предподсчет:
void precalc() {
    Int limit = solve(2*1000*1000*1000LL, 2*1000*1000*1000LL);
    Int number = 1;
    Int times = 0;
    std::ofstream fout("precalc.txt");
    fout << "numbers = {";
    bool first = true;
    while (number < limit) {
        number = next(number);
        ++times;
        if ((times & (1024*1024-1)) == 0) {
            if (!first) fout << ",";
            fout << number;
            if (first) first = false;
        }
    }
    fout << "};";
    fout.close();
}

int main() {
    Int number, times;
    std::cin >> number >> times;

    // Определяем левую границу интервала, в который попало число number:
    Int pos = Int(std::upper_bound(numbers.begin(), numbers.end(), number) - numbers.begin()

    // Текущее число из ряда для 1:
    Int curr_number = numbers[pos];
    Int curr_times = STEP * pos;

    // Пытаемся сравнить текущее число из ряда для 1 и считанное число:
    while (times > 0 && curr_number != number) {
        if (curr_number < number) {
            curr_number = next(curr_number);
            curr_times++;
        } else if (curr_number > number) {
            number = next(number);
            --times;
        }
    }
}

```

```

    }
}

if (times == 0) {
    std::cout << number << std::endl; return 0;
}

// Двигаемся до ближайшей границы предподсчитанного интервала:
while (times > 0 && (curr_times & (STEP-1)) != 0) {
    --times;
    curr_times++;
    curr_number = next(curr_number);
}

if (times == 0) {
    std::cout << curr_number << std::endl; return 0;
}

// Определяем текущий интервал, в который входит число:
pos = (Int)(std::lower_bound(numbers.begin(), numbers.end(), curr_number) - numbers.begin());
assert(numbers[pos] == curr_number);

// Инкрементируем от текущего числа с шагом STEP:
while (times >= STEP) {
    times -= STEP;
    ++pos;
    curr_number = numbers[pos];
}

// Добиваем остаток:
while (times > 0) {
    curr_number = next(curr_number);
    --times;
}

std::cout << curr_number << std::endl;

return 0;
}

```


ЗАДАЧА №646

Сладкие забавы

(Время: 1 сек. Память: 16 Мб Сложность: 60%)

Маленькие сладкоежки Сережа и Юля очень любят конфеты. Родители детей это знают, и потому у них не возникает вопросов о том, что ребятишкам дарить в день рождения.

А тут подвернулся особый случай – обоим ребятишкам исполняется по 10 лет, и потому родители подсказали приглашенным гостям, чему дети больше всего обрадуются на их общем юбилее.

Гостей в день рождения собралось достаточно много – N человек. Гость, пришедший i -м по счету, подарил детям коробку, содержащую A_i конфет. Коробки конфет были как совсем маленькие, так и невероятно большие. Коробки были с прозрачными крышками и было видно, сколько там конфет.

Чтобы все было честно, дети решили поделить коробки так, чтобы каждому досталось не менее K конфет. Однако дети обнаружили, что сделать это можно многими способами. Ваша задача – определить количество различных способов честного дележа конфет, учитывая, что коробки не вскрываются и конфеты поштучно не делятся, и каждая коробка должна достаться только одному из ребятишек. Два варианта деления конфет считаются различными, если существует коробка конфет, которая в данных вариантах принадлежит разным детям.

Входные данные

Первая строка входного файла INPUT.TXT содержит два натуральных числа N и K соответственно ($1 \leq N \leq 50$; $1 \leq K \leq 10000$). Числа в строке разделены одиночным пробелом.

Вторая строка содержит N натуральных чисел A_i ($1 \leq i \leq N$, $1 \leq A_i \leq 10^9$), разделенных одиночными пробелами, где число A_i – это количество конфет в коробке, подаренной гостем, пришедшим i -м по счету.

Выходные данные

Единственная строка выходного файла OUTPUT.TXT должна содержать одно целое число — количество способов честно поделить конфеты между детьми.

Примеры

Пояснение

Для первого примера варианты разбиения (номера коробок, доставшихся, например, Сереже): 3, 1 2, 1 3, 1 4, 2 3, 2 4, 3 4, 1 2 4.

№	INPUT.TXT	OUTPUT.TXT
1	4 3 1 2 3 2	8
2	5 67 10 22 30 41 50	6

3	3 6 4 1 5	0
---	--------------	---

/*
Решение динамикой по суммам, но вместо ответа на задачу будем считать ответ на обратную
Сколько есть способов распределить суммы, чтобы одному из них досталось $< k$.

Это можно сделать за $O(n*k)$. В силу симметрии, ответ нужно умножить на 2.
*/

```
#include <stdio.h>
#include <vector>
#include <iostream>

typedef long long ll;

ll solve(int k, const std::vector<int>& a) {
    const int n = a.size();
    ll sum = 0;
    for (auto& it : a) {
        sum += it;
    }
    if (sum < 2 * k || n == 1) {
        return 0;
    }
    std::vector<ll> curr(k), next(k);
    curr[0] = 1;
    for (auto elem : a) {
        for (int s = 0; s < k; ++s) {
            next[s] = curr[s] + (s >= elem ? curr[s - elem] : 0);
        }
        next.swap(curr);
    }
    sum = 0;
    for (auto& it : curr) {
        sum += it;
    }
    return (ll(1) << n) - 2 * sum;
}

int main() {
    int n, k;
    scanf("%d %d", &n, &k);

    std::vector<int> a(n);
    for (int i = 0; i < n; ++i) {
        scanf("%d", &a[i]);
    }

    std::cout << solve(k, a);

    return 0;
}
```

ЗАДАЧА №655

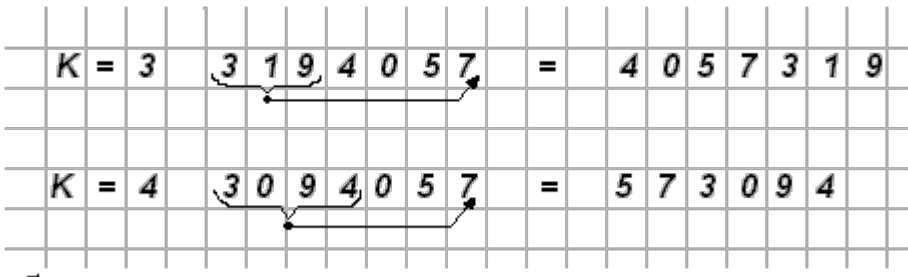
Кодирование данных

(Время: 1 сек. Память: 16 Мб Сложность: 60%)

В компании «Gold&Silver Soft» разработан новый алгоритм кодирования натуральных чисел. Этот алгоритм основан на операции циклического сдвига числа.

Циклическим сдвигом натурального числа N относительно числа K назовем операцию переноса K крайних слева цифр в десятичной записи числа N к цифре крайней справа. Обозначим через $\text{cyclic}(N, K)$ число, получаемое из N посредством циклического сдвига относительно числа K .

Введенное понятие можно проиллюстрировать на примерах:



В разработанном алгоритме предлагается для кодирования натурального числа N использовать число $\text{code}(N) = N + \text{cyclic}(N, K)$ (где K – известный параметр).

Вы работаете в «Gold&Silver Soft» и Вам была поручена разработка алгоритма декодирования, то есть восстановления числа N по числам $\text{code}(N)$ и K .

Входные данные

В первой строке входного файла INPUT.TXT находится целое число $\text{code}(N)$. Во второй строке входного файла находится целое число K . Ограничения: $10^{K+1} \leq \text{code}(N) \leq 10^{18}$, $1 \leq K \leq 17$.

Выходные данные

В первой и единственной строке выходного файла OUTPUT.TXT должно быть выведено целое число N . Если существует несколько вариантов восстановления числа N по числам $\text{code}(N)$ и K , то выведите любой из них.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	512 1	319

/*
Задача: 655. Кодирование данных
Решение: расширенный алгоритм Евклида, диофантово уравнение, $O(\log(\text{code}))$
Автор: Дмитрий Козырев, <https://github.com/dmkz> , dmkozyrev@rambler.ru

```

*/

#include <iostream>
#include <vector>
#include <cassert>

typedef long long ll;
typedef __int128_t Int;

template<typename T>
T gcd(T a, T b) {
    return b == 0 ? a : gcd(b, a % b);
}

template<typename T>
bool euclid(T a, T b, T c, T& x, T& y) {
    // Решение диофантова уравнения  $a \cdot x + b \cdot y = c$ 

    // Частные случаи:
    if (a < b) { return euclid(b, a, c, y, x); }
    if (c == 0) { x = b; y = -a; return true; }
    if (b == 0) { x = c; y = 0; return true; }
    // Прямой ход:
    std::vector<T> r{a, b}, q;
    while (b != 0 && a % b != 0) {
        q.push_back(a / b);
        r.push_back(a % b);
        a = b;
        b = r.back();
    }
    assert(r.back() != 0);
    if (c % r.back() != 0) {
        return false;
    }
    // Обратный ход:
    x = 0, y = 1;
    while (!q.empty()) {
        auto z = -q.back() * y + x;
        x = y;
        y = z;
        q.pop_back();
    }
    x *= c / r.back();
    y *= c / r.back();
    return true;
}

int len(ll number) {
    // Нахождение длины числа number в десятичной записи:
    int answ = 0;
    do answ++, number /= 10; while (number > 0);
    return answ;
}

ll solve(ll code, int k, bool less) {
    // Решение задачи. Параметр less отвечает, должна ли длина искомого числа быть меньше длины
    int len = ::len(code) - (int)less;
    if (k == len) return code;
    // Считаем необходимые степени десятки и находим частное решение диофантова уравнения:
    ll pow1 = 1, pow2 = 1;
    for (int i = 0; i < len - k; ++i) pow1 *= 10;
    for (int i = 0; i < k; ++i) pow2 *= 10;
    Int x, y;
    auto gcd = ::gcd(pow1+1, ::gcd(pow2+1, code));
    auto dx = (pow2+1) / gcd;
    auto dy = (pow1+1) / gcd;
    if (!euclid<Int>(dy, dx, code / gcd, x, y)) {
        return code;
    }
}

```

```

// Необходимо выбрать частное решение, удовлетворяющее условиям:
// pow2 / 10 <= x + t * dx < pow2
// 0 <= y - t * dy < pow1
// Примерные ограничения на параметры:
auto txmin = (pow2 / 10 - x + dx - 1) / dx;
auto txmax = (pow2 - x) / dx;
auto tymin = -(pow1 - y) / dy;
auto tymax = -(-y + dy - 1) / dy;
// Доводим примерные ограничения до точных:
while (x + txmax * dx >= pow2) --txmax;
while (x + (txmax+1) * dx < pow2) ++txmax;
while (x + txmin * dx < pow2 / 10) ++txmin;
while (x + (txmin-1) * dx >= pow2 * 10) --txmin;
while (y - (tymin-1) * dy < pow1) --tymin;
while (y - tymin * dy >= pow1) ++tymin;
while (y - (tymax+1) * dy >= 0) ++tymax;
while (y - tymax * dy < 0) --tymax;
// Смотрим пересечение отрезков и считаем ответ:
if (txmin > txmax) std::swap(txmin, txmax);
if (tymin > tymax) std::swap(tymin, tymax);
if (txmax < tymin || tymax < txmin) return code;
auto t = std::max(tymin, txmin);
return ll((x + t * dx) * pow1 + (y - t * dy));
}

ll get_code(ll number, int k) {
// Кодирование числа number со сдвигом k
int len = ::len(number);
if (len == k) return number;
assert(len > k);
ll pow1 = 1, pow2 = 1;
for (int i = 0; i < len - k; ++i) pow1 *= 10;
for (int i = 0; i < k; ++i) pow2 *= 10;
return number + number / pow1 + number % pow1 * pow2;
}

ll solve(ll code, int k) {
auto answ = solve(code, k, 0); // Пробуем найти число такой же длины
if (get_code(answ, k) != code) {
    answ = solve(code, k, 1); // Пробуем найти число длины на 1 меньше
}
assert(get_code(answ, k) == code);
return answ;
}

int main() {
ll code; int k;
std::cin >> code >> k;
std::cout << solve(code, k);
return 0;
}

```

ЗАДАЧА №660

Контрольная работа

(Время: 3 сек. Память: 16 Мб Сложность: 60%)

Петя для выполнения контрольной работы по математике аккуратно вырезал из бумаги в клеточку лист прямоугольной формы размером N клеток по вертикали и M клеток по горизонтали.

Перед уроком, в ожидании учителя, чтобы как-нибудь развлечься, Петя раскрасил ручкой на листе бумаги K различных клеток. Получив замечание учителя, Петя решил вырезать из этого листа бумаги меньший прямоугольный листок, на котором не было бы раскрашенных клеток, и сказал учителю, что знает сколькими различными способами это можно сделать.

Петя считает, что два способа вырезания прямоугольных листков являются различными, если на листе бумаги найдется хотя бы одна клетка, которая принадлежит одному из вырезаемых листков и не принадлежит другому. На самом деле, Петя не знает точного числа всевозможных способов вырезания листка прямоугольной формы. Однако он слышал о больших возможностях современных вычислительных машин, поэтому решил попросить помощи у участников республиканской олимпиады по информатике.

Производить разрезы разрешается только по линиям, разделяющим клетки друг от друга.

Входные данные

В первой строке входного файла INPUT.TXT находятся три числа N, M и K, разделенные пробелами, где N и M – размеры листа бумаги, а K – количество закрашенных Петей клеток. Каждая из следующих K строк содержит два числа, разделенные пробелом и описывающие одну закрашенную клетку. Первое число определяет номер строки листа бумаги, в которой находится закрашенная клетка, а второе число – номер столбца. Строки листа бумаги нумеруются сверху вниз от 1 до N, а столбцы – слева направо от 1 до M.

Ограничения: все числа натуральные, $K \leq 100000$; $N, M \leq 5000$; $K < NM$.

Выходные данные

Выходной файл OUTPUT.TXT должен состоять из одной строки, содержащей одно целое число, равное количеству способов, которыми можно вырезать прямоугольный листок, не содержащий раскрашенных клеток, из испорченного Петей листа бумаги.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4 3 3 1 2 4 3 3 1	22

```
#include <stdio.h>
#include <iostream>
#include <vector>
#include <stack>
#include <algorithm>
```

```

#include <cassert>

typedef long long ll;

struct Item {
    int col, height;
};

int main() {
    int nRows, nCols, nErased;
    scanf("%d %d %d", &nRows, &nCols, &nErased);
    std::vector<std::vector<int>> erased(nRows);
    for (int i = 0; i < nErased; ++i) {
        int row, col;
        scanf("%d %d", &row, &col); --row, --col;
        erased[row].push_back(col);
    }
    long long answ = 0;
    std::vector<int> height(nCols+1, 0);
    for (int row = 0; row < nRows; ++row) {
        for (int col = 0; col < nCols; ++col) {
            height[col]++;
        }
        for (int col : erased[row]) {
            height[col] = 0;
        }
        std::stack<Item, std::vector<Item>> stack;
        stack.push(Item{-1, 0});
        for (int col = 0; col <= nCols; ++col) {
            while (stack.top().height > height[col]) {
                auto top = stack.top(); stack.pop();
                // Сколькими способами можно выбрать верхнюю границу
                assert(top.height > stack.top().height);
                auto mull = std::min(
                    top.height-stack.top().height,
                    top.height-height[col]
                );
                // Сколькими способами можно выбрать два нижних угла
                auto mul2 = (col-top.col+1)*(col-top.col)/2;
                auto add = mull * 1LL * mul2;
                answ += add;
                if (stack.top().height < height[col]) {
                    stack.push(Item{top.col, height[col]});
                }
            }
            if (stack.top().height < height[col]) {
                stack.push(Item{col, height[col]});
            }
        }
    }
    std::cout << answ;
    return 0;
}

```

ЗАДАЧА №829

Строки - 3

(Время: 2 сек. Память: 32 Мб Сложность: 60%)

Циклическим сдвигом строки s называется строка $s_k s_{k+1} s_{k+2} \dots s_{|s|} s_1 s_2 \dots s_{k-1}$ для некоторого k , здесь $|s|$ - длина строки s . Подстрокой строки s называется строка $s_i s_{i+1} \dots s_{j-1} s_j$ для некоторых i и j . Вам даны две строки a и b . Выведите количество подстрок строки a , являющихся циклическими сдвигами строки b .

Входные данные

Первая строка входного файла INPUT.TXT содержит строку a ($1 \leq |a| \leq 10^5$). Во второй строке входного файла записана строка b ($1 \leq |b| \leq |a|$). Обе строки состоят только из символов английского алфавита и цифр.

Выходные данные

В выходной файл OUTPUT.TXT выведите целое число – ответ на задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	abcabc abc	4
2	abcabc acb	0
3	aaaaaaa aa	6
4	aAa8aaAa aAa	4

```
#include <stdio.h>
#include <vector>
#include <string>
#include <algorithm>
#include <cassert>
#include <cstdlib>
#include <ctime>

void prepare(std::string& s) {
    for (int i = 0; i < (int)s.size(); ++i) {
        if ('0' <= s[i] && s[i] <= '9') {
            s[i] = s[i] - '0';
        } else if ('a' <= s[i] && s[i] <= 'z') {
            s[i] = s[i] - 'a' + 10;
        } else if ('A' <= s[i] && s[i] <= 'Z') {
            s[i] = s[i] - 'A' + 10 + 26;
        }
        s[i]++;
    }
}
```



```

std::vector<int> get_powers(const int count, const int base, const int mod) {
    std::vector<int> powers(1+count, 1);
    for (int i = 1; i <= count; ++i) {
        powers[i] = 1LL * powers[i-1] * base % mod;
    }
    return powers;
}

std::vector<int> pref(const std::string& s, const std::vector<int>& powers, const int mod)
    std::vector<int> pref{0};
    pref.reserve(s.size()+1u);
    for (int i = 0; i < (int)s.size(); ++i) {
        pref.push_back((pref.back() + 1LL * s[i] * powers[i]) % mod);
    }
    return pref;
}

bool is_prime(int n) {
    for (int i = 2; i <= n / i; ++i) {
        if (n % i == 0) {
            return false;
        }
    }
    return n > 1;
}

int next_prime(int value, int steps = 1) {
    while (steps--) {
        while(!is_prime(++value));
    }
    return value;
}

int main() {
    std::srand(std::time(0));
    const int mod1 = next_prime(1e9, std::rand() % 77 + 33);
    const int mod2 = next_prime(mod1, std::rand() % 77 + 33);
    const int base = 73;

    char buf[1+1000000];
    scanf("%1000000s", buf);
    std::string a(buf);
    scanf("%1000000s", buf);
    std::string b(buf);
    int n = (int)b.size();
    b += b;
    prepare(a); prepare(b);

    const int mxPow = 2e5;
    auto powers1 = get_powers(mxPow, base, mod1);
    auto powers2 = get_powers(mxPow, base, mod2);
    auto pref1_a = pref(a, powers1, mod1);
    auto pref2_a = pref(a, powers2, mod2);
    auto pref1_b = pref(b, powers1, mod1);
    auto pref2_b = pref(b, powers2, mod2);

    std::vector<std::pair<int, int>> hashes;
    hashes.reserve(b.size());
    for (int i = 0; i + n - 1 < (int)b.size(); ++i) {
        const int hash1 = (0LL + pref1_b[i+n] - pref1_b[i] + mod1) * powers1[mxPow - (i + n)
        const int hash2 = (0LL + pref2_b[i+n] - pref2_b[i] + mod2) * powers2[mxPow - (i + n)
        hashes.push_back({hash1, hash2});
    }

    std::sort(hashes.begin(), hashes.end());

    int ans = 0;
    for (int i = 0; i + n - 1 < (int)a.size(); ++i) {
        const int hash1 = (0LL + pref1_a[i+n] - pref1_a[i] + mod1) * powers1[mxPow - (i + n

```

```
        const int hash2 = (0LL + pref2_a[i+n] - pref2_a[i] + mod2) * powers2[mxPow - (i + n  
        answ += std::binary_search(hashes.begin(), hashes.end(), std::make_pair(hash1, hash  
    }  
  
    printf("%d", answ);  
  
    return 0;  
}
```

ЗАДАЧА №898

Ленточка

(Время: 1 сек. Память: 16 Мб Сложность: 60%)

Расположенную вертикально прямоугольную бумажную ленточку с закрепленным нижним концом стали складывать следующим образом:

- на первом шаге ее согнули пополам так, что верхняя половина легла на нижнюю либо спереди (Р - сгибание) либо сзади (Z - сгибание),

- на последующих $n-1$ шагах выполнили аналогичное действие с получающейся на предыдущем шаге согнутой ленточкой, как с единым целым.

Затем ленточку развернули, приведя ее в исходное состояние. На ней остались сгибы - ребра от перегибов, причем некоторые из ребер оказались направленными выпуклостью к нам (К - ребра), а некоторые - от нас (О - ребра). Ребра пронумеровали сверху вниз числами от 1 до $2^n - 1$.

Требуется написать программу, которая по заданной строке символов из прописных букв «Р» и «Z», определяющей последовательность типов сгибаний, и номерам ребер сообщает тип этих ребер, получившийся после данной последовательности преобразований.

Входные данные

В первой строке входного файла INPUT.TXT содержится натуральное число n – количество сгибаний ленточки ($n \leq 60$), во второй строке – набор n символов из прописных английских букв «Р» и «Z». Третья строка содержит в начале число k – количество рассматриваемых ребер ($k \leq 10$), а далее их номера (числа от 1 до $2^n - 1$).

Выходные данные

В выходной файл OUTPUT.TXT выведите k символов (прописные английские буквы «К» или «О») – типы рассматриваемых ребер.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 PP 1 1	K
2	2 ZZ 2 1 2	OK

/*

Задача: 898. Ленточка

Решение: рекурсия, динамическое программирование, $O(k * n)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

*/

```

#include <bits/stdc++.h>
typedef long long ll;
char solve(int n, ll x, const std::string& s) {
    const ll mid = (ll(1) << (n - 1));
    if (mid == x) { return s[n-1] == 'Z' ? 'K' : 'O'; }
    if (x < mid) { return solve(n-1, x, s); }
    return char('K'-solve(n-1, mid - (x - mid), s)+'O');
}
int main() {
    int n; std::string s;
    while (std::cin >> n >> s) {
        int q; std::cin >> q;
        std::reverse(s.begin(), s.end());
        while (q--) {
            ll x; std::cin >> x;
            std::cout << solve(n, (ll(1) << n) - x, s);
        }
        std::cout << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №921

Турнир - 2

(Время: 1 сек. Память: 16 Мб Сложность: 60%)

Word-ландия – новая страна, образованная объединением двух древних государств под воздействием внешних угроз. Эти государства теперь являются провинциями Word-ландии, но референдумы по их наименованию ещё не прошли, поэтому они называются Старшая и Младшая Byte-ландии.

Для сплочения населения было решено провести чемпионат между провинциями. В Старшей Byte-ландии национальной игрой являются шахматы, а в Младшей – волейбол. Поэтому чемпионат решили провести по шахболу. Чтобы не затягивать чемпионат, объединённое правительство решило провести ровно K матчей. Каждый матч – это либо партия в шахматы, либо волейбольный матч. Победитель в матче получает одно очко в зачёт чемпионата, в случае ничьей в шахматной партии обе провинции получают по 0.5 очка.

Правительство заинтересовано в том, чтобы в чемпионате «победила дружба», то есть провинции набрали одинаковое количество очков. Поэтому высокие чины обратились к вам с просьбой определения минимального количества шахматных партий в чемпионате, чтобы разность математических ожиданий набранных провинциями очков была минимальна.

За долгую историю Мировых чемпионатов известно, что Старшая Byte-ландия выигрывает в шахматы у Младшей с вероятностью p_1 и проигрывает в волейбол с вероятностью p_2 . Ничья в шахматах достигалась с вероятностью p_3 . Примечание: Математическое ожидание - среднее значение случайной величины в теории вероятностей. Для дискретной случайной величины X с законом распределения $P(X = x_i) = p_i$ математическим ожиданием называется сумма парных произведений всех возможных значений случайной величины на соответствующие им вероятности, т.е.

$$M(X) = \sum_{i=1}^n x_i p_i$$

Входные данные

Входной файл INPUT.TXT содержит четыре целых числа: K ($1 < K \leq 10^{16}$), p_1, p_2, p_3 ($0 \leq p_1, p_2, p_3 \leq 100$) – количества матчей и вероятностей в процентах.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно натуральное число – минимальное количество шахматных партий (учтите, что чемпионат не должен превратиться в шахматный турнир).

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 50 50 50	1

2	4 0 0 100	3
---	-----------	---

```

/*
    Задача: 921. Турнир - 2

    Решение: теория вероятностей, тернарный поиск, математическое ожидание,  $O(\log(k))$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/
#include <bits/stdc++.h>
typedef long long ll;
typedef long double ld;
int main() {
    ll k; int a, b, c;
    while (std::cin >> k >> a >> b >> c) {
/*
        Всего k матчей. Пусть шахматных x.
        Математическое ожидание количества очков, набранных первой командой:
             $x * (p1 + 0.5 * p2) + q1 * (k - x)$ 
        Математическое ожидание количества очков, набранных второй командой:
             $x * (p3 + 0.5 * p2) + q2 * (k - x)$ 
        Разница:
             $x * (p1 - p3) + (q1 - q2) * (k - x)$ 
             $x * (p1 - p3 + q2 - q1) + (q1 - q2) * k$ 
        Подставляем значения:
            p1 = a
            p3 = 100 - a - c
            q2 = b
            q1 = 100 - b
        Получаем:
             $x * (2 * a + c + 2 * b - 200) + (100 - 2 * b) * k$ 
*/
        auto f = [&](ll x) {
            return std::abs(x * (2 * a + c + 2 * b - 200) + (100 - 2 * b) * k);
        };
        auto naive = [&](ll fi, ll se) {
            ll md = (ll)1e18L;
            ll mx = k;
            for (ll x = fi; x <= se; ++x) {
                if (f(x) < md) { mx = x; md = f(x); }
            }
            return mx;
        };

        auto fast = [&]() {
            ll low = 1, high = k - 1;
            while (high - low > 10) {
                ll x1 = (low + high) / 2;
                ll x2 = x1 + 1;
                ll y1 = f(x1);
                ll y2 = f(x2);
                if (y1 <= y2) {
                    high = x2;
                } else {
                    low = x1;
                }
            }
            return naive(low, high);
        };
        std::cout << fast() << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №1083

Sqrt-декомпозиция

(Время: 2 сек. Память: 16 Мб Сложность: 60%)

Задан массив $a[1..n]$ из n неотрицательных целых чисел. Необходимо реализовать следующие операции над ним:

- **get(i)** – извлечение элемента с номером i ;
- **update(l,r,x)** – присвоение значения x всем элементам подмассива $a[l..r]$;
- **add(l,r,x)** – увеличение всех элементов $a[l..r]$ на x ;
- **rsq(l,r)** – вычисление суммы всех элементов $a[l..r]$, т.е. значения $a[l]+a[l+1]+\dots+a[r]$;
- **rmq(l,r)** – вычисление минимального элемента среди всех значений $a[l..r]$.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число n – размер массива. Во второй строке записаны n чисел – элементы массива. Третья строка содержит натуральное число m – количество запросов. Следующие m строк содержат запросы в формате, представленном в примере. Все числа во входных данных целые неотрицательные и не превосходят 40 000.

Выходные данные

В выходной файл OUTPUT.TXT для запросов get, rsq и rmq выведите результат в отдельной строке в порядке их следования.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	10	
	1 2 3 4 5 6 7 8 9 10	
	5	
	rsq 1 10	55
	update 5 6 2	2
	rmq 3 9	10
	add 2 8 3	
	get 7	

```
/*
    Задача: 1083. Sqrt-декомпозиция

    Решение: Segment Tree, ленивое проталкивание, O(n + q * log(n))

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/
```

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>
#include <string>

typedef long long ll;
```

```

const int INF = (int)1e9+1;
const int UNDEF = -1;

/*
Структура узла дерева отрезков:
- `value` - значение, которое должно быть присвоено всему отрезку (UNDEF - не определен
- `extra` - значение, которое должно быть прибавлено к каждому элементу отрезка
- `sum` - сумма на отрезке
- `min` - минимум на отрезке

Соглашения:
1) Если value != UNDEF:
    а) extra = 0
    б) сумма вычисляется как value * len
    в) минимум вычисляется как value
2) Если extra != 0:
    а) сумма вычисляется как sum + len * extra
    б) минимум вычисляется как min + extra

*/

struct Node {

    int value, extra; ll sum; int min;

    Node(int value_ = UNDEF, int extra_ = 0, ll sum_ = 0, int min_ = 0)
        : value(value_), extra(extra_), sum(sum_), min(min_) { }

};

struct SegmentTree {

    int size;

    std::vector<Node> data;

    // Конструктор от массива:
    SegmentTree(const std::vector<int>& arr) : size((int)arr.size()) {
        int pow = 1;
        while (pow < size) pow *= 2;
        data.resize(2*pow);
        build(0,0,size-1,arr);
    }

    // Пересчет значений для текущего элемента от его поддеревьев:
    void modify(int v, int l, int r, int m) {
        assert(data[v].value == UNDEF && data[v].extra == 0);
        data[v].sum = sum(2*v+1, l, m, l, m) + sum(2*v+2, m+1, r, m+1, r);
        data[v].min = std::min(min(2*v+1, l, m, l, m), min(2*v+2, m+1, r, m+1, r));
    }

    // Построение дерева:
    void build(int v, int l, int r, const std::vector<int>& arr) {
        if (l == r) { // лист
            data[v] = Node{arr[l], 0, arr[l], arr[l]};
        } else {
            int m = (l + r) >> 1;
            build(2*v+1, l, m, arr);
            build(2*v+2, m+1, r, arr);
            modify(v, l, r, m);
        }
    }

    // Проталкивание изменений от корня к детям:
    void push(int v, int l, int r, int m) {
        // Если отрезок - лист, то ничего проталкивать не нужно:
        if (l == r) return;
        // Если всему отрезку было присвоено какое-то значение:

```



```

    if (data[v].value != UNDEF) {
        assert(data[v].extra == 0);
        update(2*v+1, 1, m, 1, m, data[v].value);
        update(2*v+2, m+1, r, m+1, r, data[v].value);
        data[v].value = UNDEF;
    }
    // Если ко всему отрезку было прибавлено какое-то значение:
    if (data[v].extra != 0) {
        add(2*v+1, 1, m, 1, m, data[v].extra);
        add(2*v+2, m+1, r, m+1, r, data[v].extra);
        data[v].extra = 0;
    }
}

// Запрос присвоения на отрезке:
void update(int v, int l, int r, int ql, int qr, int x) {
    if (qr < l || ql > r) return;
    ql = std::max(ql, l);
    qr = std::min(qr, r);
    if (ql == l && qr == r) { // отрезок целиком
        data[v] = Node{x, 0, x * ll(qr-ql+1), x};
    } else {
        int m = (l+r) >> 1;
        push(v, l, r, m);
        update(2*v+1, 1, m, ql, qr, x);
        update(2*v+2, m+1, r, ql, qr, x);
        modify(v, l, r, m);
    }
}

// Запрос прибавления на отрезке:
void add(int v, int l, int r, int ql, int qr, int x) {
    if (qr < l || ql > r) return;
    ql = std::max(ql, l);
    qr = std::min(qr, r);
    if (ql == l && qr == r) { // Если было присвоение на отрезке
        if (data[v].value != UNDEF) {
            assert(data[v].extra == 0);
            data[v].value += x;
        } else {
            data[v].extra += x;
        }
    } else {
        int m = (l+r) >> 1;
        push(v, l, r, m);
        add(2*v+1, 1, m, ql, qr, x);
        add(2*v+2, m+1, r, ql, qr, x);
        modify(v, l, r, m);
    }
}

// Запрос получения одного элемента:
int get(int v, int l, int r, int p) {
    assert(l <= p && p <= r);
    if (data[v].value != UNDEF) {
        assert(data[v].extra == 0);
        return data[v].value;
    } else {
        int m = (l+r) >> 1, ret;
        push(v, l, r, m);
        if (p <= m) {
            ret = get(2*v+1, 1, m, p);
        } else {
            ret = get(2*v+2, m+1, r, p);
        }
        modify(v, l, r, m);
        return ret;
    }
}

```

```

// Запрос получения суммы на отрезке:
ll sum(int v, int l, int r, int ql, int qr) {
    if (qr < l || ql > r) return 0;
    ql = std::max(l, ql);
    qr = std::min(r, qr);
    if (ql == l && qr == r) {
        if (data[v].value != UNDEF) {
            assert(data[v].extra == 0);
            return ll(qr-ql+1) * data[v].value;
        } else {
            return data[v].sum + data[v].extra * ll(qr-ql+1);
        }
    } else {
        int m = (l + r) >> 1;
        push(v, l, r, m);
        ll ret = sum(2*v+1, l, m, ql, qr) + sum(2*v+2, m+1, r, ql, qr);
        modify(v, l, r, m);
        return ret;
    }
}

// Запрос получения минимума на отрезке:
int min(int v, int l, int r, int ql, int qr) {
    if (qr < l || ql > r) return INF;
    ql = std::max(l, ql);
    qr = std::min(r, qr);
    if (ql == l && qr == r) {
        if (data[v].value != UNDEF) {
            assert(data[v].extra == 0);
            return data[v].value;
        } else {
            return data[v].min + data[v].extra;
        }
    } else {
        int m = (l + r) >> 1;
        push(v, l, r, m);
        int ret = std::min(min(2*v+1, l, m, ql, qr), min(2*v+2, m+1, r, ql, qr));
        modify(v, l, r, m);
        return ret;
    }
}

// Более удобные внешние запросы:
void update(int left, int right, int x) {
    update(0, 0, size-1, left, right, x);
}

void add(int left, int right, int x) {
    add(0, 0, size-1, left, right, x);
}

int get(int pos) {
    return get(0, 0, size-1, pos);
}

ll sum(int left, int right) {
    return sum(0, 0, size-1, left, right);
}

int min(int left, int right) {
    return min(0, 0, size-1, left, right);
}

};

int main() {
    int n; scanf("%d", &n);
    std::vector<int> arr(n);

```

```

for (auto& it : arr) scanf("%d", &it);
SegmentTree st(arr);
int q; scanf("%d", &q);
while (q--) {
    char buf[7]; scanf("%6s", buf);
    std::string op(buf);
    if (op == "get") {
        int pos; scanf("%d", &pos);
        printf("%d\n", st.get(pos-1));
    } else if (op == "update") {
        int l, r, x; scanf("%d %d %d", &l, &r, &x);
        st.update(l-1, r-1, x);
    } else if (op == "add") {
        int l, r, x; scanf("%d %d %d", &l, &r, &x);
        st.add(l-1, r-1, x);
    } else if (op == "rsq") {
        int l, r; scanf("%d %d", &l, &r);
        printf("%I64d\n", st.sum(l-1, r-1));
    } else if (op == "rmq") {
        int l, r; scanf("%d %d", &l, &r);
        printf("%d\n", st.min(l-1, r-1));
    }
}
return 0;
}

```

ЗАДАЧА №614

Скобки - 3

(Время: 1 сек. Память: 32 Мб Сложность: 61%)

Рассмотрим скобочные последовательности с одним типом скобок. Для заданной скобочной последовательности найдите количество ее подпоследовательностей, которые являются правильными скобочными последовательностями.

Например, для последовательности "((()())(" таких последовательностей восемь: "((()())", "(()())", "((())", "(()())", "(()", "()", "()" и "".

Входные данные

Входной файл INPUT.TXT содержит последовательность, состоящую не более чем из 300 круглых скобок.

Выходные данные

В выходной файл OUTPUT.TXT выведите количество различных правильных скобочных подпоследовательностей заданной последовательности.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	((()())(8

```
/*
    Двумерное динамическое программирование

    Фиксируем для текущей позиции ближайшую к ней открывающую и ближайшую к ней закрывающую
    count[pos][balance] = количество непустых правильных скобочных подпоследовательностей н
    Динамика справа-налево. Переход:

        count[pos][balance] = (balance ? 0 : 1) + count[rOpen][balance+1] + count[rClos][ba

    Ответом будет являться count[rOpen][1]+1
*/

#include <iostream>
#include <iomanip>
#include <vector>
#include <string>

struct UInt {
    static const int POW10 = (int)1e9;
    static const int WIDTH = 9;

    std::vector<int> digits;

    UInt(int number = 0) : digits(1, number) { }

    UInt& operator +=(const UInt& other) {
```

```

        const int s1 = this->digits.size();
        const int s2 = other.digits.size();
        int carry = 0;
        for (int i = 0; i < s1 || i < s2 || carry; ++i) {
            if (i >= s1) {
                digits.push_back(0);
            }
            carry += digits[i] + (i < s2 ? other.digits[i] : 0);
            digits[i] = carry % POW10;
            carry /= POW10;
        }
        return normalize();
    }

    UInt& normalize() {
        while ((int)digits.size() > 1 && digits.back() == 0) {
            digits.pop_back();
        }
        return *this;
    }
};

std::ostream& operator<<(std::ostream& os, const UInt& num) {
    os << num.digits.back();
    for (int i = (int)num.digits.size()-2; i >= 0; --i) {
        os << std::setw(UInt::WIDTH) << std::setfill('0') << num.digits[i];
    }
    return os << std::setfill(' ');
}

int main() {
    std::string s;
    std::cin >> s;
    const int n = s.size();

    std::vector<std::vector<UInt>> count(n+1, std::vector<UInt>(1+n));

    int rOpen = n, rClos = n;
    for (int pos = n-1; pos >= 0; --pos) {
        count[pos][0] = 1;
        count[pos][0] += count[rOpen][1];
        for (int balance = 1; balance < n; ++balance) {
            count[pos][balance] += count[rOpen][balance+1];
            count[pos][balance] += count[rClos][balance-1];
        }
        if (s[pos] == ')') {
            rClos = pos;
        } else {
            rOpen = pos;
        }
    }

    auto answ = count[rOpen][1];
    std::cout << (answ += 1);

    return 0;
}

```

ЗАДАЧА №752

2-3 Дерево

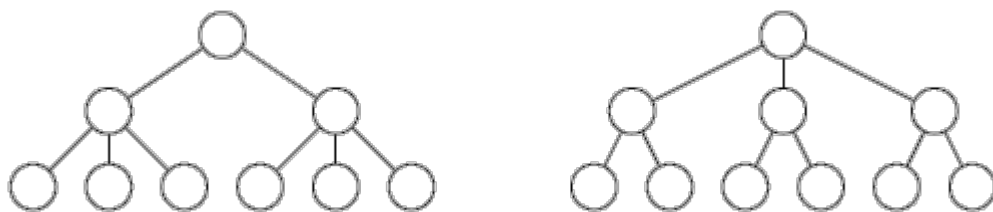
(Время: 3 сек. Память: 16 Мб Сложность: 61%)

2-3 дерево — элегантная структура данных, изобретенная Джоном Хопкрофтом. Она предназначена для использования с той же целью, что и двоичное дерево поиска. 2-3 дерево представляет собой дерево с корнем, которое обладает следующими свойствами:

- корень и каждая внутренняя вершина имеет либо 2 либо 3 ребенка;
- глубина всех листьев одна и та же.

Единственное исключение — это когда дерево содержит ровно одну вершину. В этом случае корень дерева является и листом, и поэтому не имеет детей. Основная суть приведенных свойств в том, что дерево с L листьями имеет высоту $O(\log L)$.

Вообще говоря, может существовать несколько 2-3 деревьев с L листьями. Например, на следующем рисунке показаны два возможных дерева с 6 листьями.



По заданному числу L найдите количество различных 2-3 деревьев с L листьями. Так как ответ может быть довольно большим, выведите его по модулю R .

Входные данные

Входной файл INPUT.TXT содержит два целых числа: L и R ($1 \leq L \leq 5\,000$, $1 \leq R \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число — количество различных 2-3 деревьев, имеющих ровно L листьев, взятое по модулю R .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	6 1000000000	2
2	7 1000000000	3

/*

Задача: 752. 2-3 Дерево

Решение: динамическое программирование, двумерное дп, комбинаторика, $O(\log(n) * n^2)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

*/

```

#include <bits/stdc++.h>

typedef unsigned long long ull;

int solve(const int need, const int mod) {
    if (need == 1) { return 1 % mod; }
    if (mod == 1) { return 0; }
    std::vector<std::vector<ull>> dp(14, std::vector<ull>(1024 * 8, 0)); // [depth][nLeafs]
    dp[1][2] = dp[1][3] = 1;
    for (int depth = 2; depth < 14; ++depth) {
        // 2 branches
        for (int sum = 0; sum <= need; ++sum) {
            for (int cnt = 0; cnt <= sum; ++cnt) {
                dp[depth][sum] += dp[depth-1][cnt] * dp[depth-1][sum - cnt];
                if (cnt % 16 == 0) {
                    dp[depth][sum] %= mod;
                }
            }
        }
        // 3 branches
        for (int sum = need; sum >= 0; --sum) {
            for (int cnt = sum; cnt >= 0; --cnt) {
                dp[depth][sum] += dp[depth][cnt] * dp[depth-1][sum - cnt];
                if (cnt % 16 == 0) {
                    dp[depth][sum] %= mod;
                }
            }
        }
    }
    ull answ = 0;
    for (int depth = 1; depth < 14; ++depth) {
        answ += dp[depth][need];
    }
    return int(answ % mod);
}

int main() {
    std::ios_base::sync_with_stdio(false);
    for (int n, mod; std::cin >> n >> mod; std::cout << solve(n, mod) << std::endl);
    return 0;
}

```

ЗАДАЧА №993

Симпатичные последовательности

(Время: 1 сек. Память: 16 Мб Сложность: 61%)

Рассмотрим последовательность a_1, a_2, \dots, a_n неотрицательных целых чисел. Обозначим как $c_{i,j}$ количество появлений числа i среди чисел a_1, a_2, \dots, a_j . Будем называть последовательность k -симпатичной, если для всех $i_1 < i_2$ и для всех j выполнено условие: $c_{i_1,j} \geq c_{i_2,j} - k$.

По заданной последовательности a_1, a_2, \dots, a_n и числу k , найдите ее максимальный префикс, который является k -симпатичным.

Входные данные

Первая строка входного файла INPUT.TXT содержит числа n и k ($1 \leq n \leq 200\,000, 0 \leq k \leq 200\,000$). Вторая строка содержит n целых чисел в диапазоне от 0 до n .

Выходные данные

В выходной файл OUTPUT.TXT выведите максимальное p такое, что последовательность a_1, a_2, \dots, a_p является k -симпатичной.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	10 1 0 1 1 0 2 2 1 2 2 3	8
2	2 0 1 0	0

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>

struct Node {
    int min, max;
};

Node combine(const Node& a, const Node& b) {
    return Node{std::min(a.min, b.min), std::max(a.max, b.max)};
}

struct SegmentTree {
    int size;
    std::vector<Node> data;

    void build(int v, int l, int r, const std::vector<int>& arr) {
        if (l == r) {
            data[v] = Node{arr[l], arr[l]};
        } else {
            int m = (l + r) / 2;
            build(2*v+1, l, m, arr);
```



```

        build(2*v+2, m+1, r, arr);
        data[v] = combine(data[2*v+1], data[2*v+2]);
    }
}

Node get(int v, int l, int r, int ql, int qr) const {
    if (l == ql && r == qr) {
        return data[v];
    } else {
        int m = (l + r) / 2;
        if (qr <= m) {
            return get(2*v+1, l, m, ql, qr);
        } else if (ql > m) {
            return get(2*v+2, m+1, r, ql, qr);
        } else {
            return combine(
                get(2*v+1, l, m, ql, m),
                get(2*v+2, m+1, r, m+1, qr)
            );
        }
    }
}

Node get(int left, int right) const {
    return get(0, 0, size-1, left, right);
}

void inc(int v, int l, int r, int p) {
    if (l == r) {
        data[v].min++;
        data[v].max++;
    } else {
        int m = (l + r) / 2;
        if (p <= m) {
            inc(2*v+1, l, m, p);
        } else {
            inc(2*v+2, m+1, r, p);
        }
        data[v] = combine(data[2*v+1], data[2*v+2]);
    }
}

void inc(int value) {
    inc(0, 0, size-1, value);
}

SegmentTree(const std::vector<int>& arr) {
    size = (int)arr.size();
    int pow = 1;
    while (pow < size) {
        pow *= 2;
    }
    data.resize(2 * pow);
    build(0, 0, size-1, arr);
}

};

int main() {
    int n, k;
    scanf("%d %d", &n, &k);

    std::vector<int> count(1+n, 0);

    SegmentTree st(count);

    int answ = 0;

    for (int i = 0; i < n; ++i) {
        int value;

```

```

scanf("%d", &value);
st.inc(value);
count[value]++;
if (value > 0) {
    auto res = st.get(0, value-1);
    if (res.min < count[value]-k) {
        break;
    }
}
if (value < n) {
    auto res = st.get(value+1, n);
    if (res.max-k > count[value]) {
        break;
    }
}
++answ;
}

printf("%d", answ);

return 0;
}

```

ЗАДАЧА №1156

Минимальный сдвиг

(Время: 1 сек. Память: 16 Мб Сложность: 61%)

Циклическим сдвигом строки s называется строка $s_{k+1} s_{k+2} \dots s_n s_1 s_2 \dots s_k$ для некоторого k ($0 \leq k < n$), где n – длина строки s .

Для заданной строки требуется определить ее лексикографически минимальный сдвиг, т.е. необходимо найти среди всех возможных циклических сдвигов строки тот, который идет первым в алфавитном порядке.

Входные данные

В единственной строке входного файла INPUT.TXT записана строка, состоящая из символов с кодами ASCII от 33 до 127. Длина строки не превышает 10^5 .

Выходные данные

В выходной файл OUTPUT.TXT выведите одну строку – минимальный лексикографический сдвиг исходной строки.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	program	amprogr
2	cab	abc
3	bbbbbb	bbbbbb

```
#include <stdio.h>
#include <algorithm>
#include <vector>
#include <string>
#include <cassert>
#include <cstdlib>
#include <ctime>

typedef unsigned long long ull;

bool is_prime(const int n) {
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}

int next_prime(int number, int steps = 1) {
    while (steps--) {
        while (!is_prime(++number));
    }
    return number;
}
```

```

int main() {
    std::srand(std::time(0));

    // Константы полиномиального хэша:
    const int mod1 = next_prime(1e9, std::rand() % 77 + 33);
    const int base = next_prime(256, std::rand() % 77 + 33);
    const int mxPow = 2e5;

    // Предподсчет степеней основания base по модулям полиномиального хэширования:
    std::vector<int> pow1(1+mxPow, 1);
    std::vector<ull> pow2(1+mxPow, 1);
    for (int i = 1; i <= mxPow; ++i) {
        pow1[i] = 1LL * pow1[i-1] * base % mod1;
        pow2[i] = pow2[i-1] * base;
    }

    // Чтение входных данных:
    char buf[1+1000000];
    scanf("%1000000s", buf);
    std::string s(buf); //assert(!worst_case(s));
    const int n = s.size();
    s += s;

    // Предпосчет полиномиального хэша на префиксе:
    std::vector<int> pref1{0};
    std::vector<ull> pref2{0};
    for (int i = 0; i < (int)s.size(); ++i) {
        pref1.push_back((pref1.back() + 1LL * s[i] * pow1[i]) % mod1);
        pref2.push_back(pref2.back() + s[i] * pow2[i]);
    }

    // Поиск минимального сдвига:
    std::vector<int> shifts(n);
    for (int i = 0; i < n; ++i) {
        shifts[i] = i;
    }
    auto min = std::min_element(shifts.begin(), shifts.end(), [&](const int p1, const int p
        if (s[p1] < s[p2]) {
            return true;
        } else if (s[p1] > s[p2]) {
            return false;
        }
        int low = 0, high = n;
        while (high - low > 1) {
            const int mid = (low + high) / 2;
            const auto hash1_a = (0LL + pref1[p1+mid+1] - pref1[p1] + mod1) * pow1[mxPow-(p
            const auto hash1_b = (0LL + pref1[p2+mid+1] - pref1[p2] + mod1) * pow1[mxPow-(p
            const auto hash2_a = (pref2[p1+mid+1] - pref2[p1]) * pow2[mxPow-(p1+mid)];
            const auto hash2_b = (pref2[p2+mid+1] - pref2[p2]) * pow2[mxPow-(p2+mid)];
            if (hash1_a == hash1_b && hash2_a == hash2_b) {
                low = mid;
            } else {
                high = mid;
            }
        }
        if (low+1 == n) {
            return p1 < p2;
        }
        assert(s[p1+low+1] != s[p2+low+1]);
        return s[p1+low+1] < s[p2+low+1];
    });

    s = s.substr(*min, n);
    printf("%s", s.c_str());

    return 0;
}

```

Электронная почта

(Время: 2 сек. Память: 32 Мб Сложность: 62%)

Современный мир немыслим без Интернета и электронной почты. Для того, чтобы людям было проще ориентироваться в потоке поступающих писем, были созданы специальные программы - почтовые клиенты. Фирма TIRLABS занимается разработкой почтового клиента The Bar!.

Недавно программисты компании завершили разработку очередной, 366239-ой, версии этого почтового клиента. Менеджеры по продажам и рекламе уже готовы всю рекламировать и продавать эту новую программу. Однако, генеральный директор компании TIRLABS считает, что любая программа должна быть подвергнута всестороннему тестированию, прежде чем она будет продаваться. «Да и не работающую программу, скорее всего, никто не купит!» - сказал он.

Одним из видов тестирования сложных программ является так называемое стресс-тестирование. При нем программа тестируется в экстремальных условиях, часто даже в таких, на какие она не рассчитана. Для тестирования The Bar! ver. 366239 был выбран такой метод: программа запускается на n компьютерах, стоящих в одной комнате, после чего с компьютеров друг на друга посылаются m писем. При этом никакие два события (отправление или прием письма) не происходят одновременно, а сеть настолько надежна, что все письма доходят до адресата. Адресат у каждого письма при этом только один.

Почтовый клиент The Bar! в процессе работы ведет протокол, в который заносятся идентификаторы отправленных и полученных писем в том порядке, в котором они были обработаны почтовым клиентом. При этом при оценке результатов тестирования в расчет принимаются только события, отраженные в этом протоколе.

Программа считается правильно работающей по результатам тестирования, если всем событиям, указанным в протоколах, можно сопоставить моменты времени таким образом, что никакие два события не происходят одновременно, и каждое из писем отправляется до того, как получается. При этом, разумеется, внутри каждого из протоколов порядок событий должен остаться прежним.

Даны протоколы работы почтового клиента на каждом из компьютеров. Напишите программу, проверяющую, можно ли по результатам этого тестирования признать программу правильно работающей.

Входные данные

Входной файл INPUT.TXT содержит несколько наборов входных данных. Первая строка содержит t - число наборов входных данных. Оставшиеся строки входного файла содержат эти наборы.

Описание каждого набора начинается со строки, содержащей два целых числа n ($1 \leq n \leq 50000$) и m ($1 \leq m \leq 100000$) - количество компьютеров и отправленных писем соответственно. Далее следуют n строк, i -ая из которых содержит протокол работы почтового клиента на i -ом компьютере. Протокол работы состоит из целого числа k_i ($0 \leq k_i \leq 2m$) и k_i чисел a_{ij} , описывающих события. Если $a_{ij} > 0$, то j -ым по счету событием на i -ом компьютере была посылка письма с идентификатором a_{ij} , если же $a_{ij} < 0$, то j -ым по счету событием на i -ом компьютере было получение письма с идентификатором $-a_{ij}$. Нулю a_{ij} равно быть не может.

Идентификатор письма - это целое число от 1 до 10^6 . Внутри одного набора входных данных все письма имеют различные идентификаторы. Гарантируется, что все письма, которые были отправлены, были кем-то приняты, то есть сумма всех k_i в одном наборе входных данных равна $2m$.

Сумма чисел n по всем наборам входных данных не превосходит 50000, сумма чисел m по всем наборам входных данных не превосходит 100000.

Выходные данные

В выходной файл OUTPUT.TXT для каждого набора входных данных выведите ровно одну строку. Эта строка должна содержать слово YES, если программу можно считать правильно работающей по результатам тестирования, и NO - в противном случае.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 2 2 2 1 -2 2 2 -1 2 2 2 -2 1 2 -1 2	YES NO
2	1 2 3 2 1 -1 4 239 -239 366 -366	YES

```
/*
    Задача: 624. Электронная почта

    Решение: графы, топологическая сортировка, циклы,  $O(n \cdot \log(n) + m)$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/
#include <bits/stdc++.h>
#define isz(x) (int)(x).size()
#define all(x) (x).begin(), (x).end()
#define reunique(x) (x).erase(std::unique(all(x)), (x).end())
#define lowpos(x, y) int(std::lower_bound(all(x), y) - (x).begin())
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
struct Graph {
    const int n;
    vvi next;
    vi color;
    bool ok;
    Graph(int n_) : n(n_), next(n_) { }
    void add_edge(int u, int v) { next[u].push_back(v); }
    void dfs(int u) {
        color[u] = 1;
        for (int v : next[u]) {
            if (color[v] == 0) {
                dfs(v);
            } else if (color[v] == 1) {
                ok = false;
            }
        }
        color[u] = 2;
    }
    bool solve() {
        ok = true;
        color.assign(n, 0);
        for (int u = 0; u < n; ++u) {
```

```

        if (color[u] != 2) {
            dfs(u);
        }
    }
    return ok;
}

};

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int t;
    while (std::cin >> t) {
        while (t--) {
            int n, m; std::cin >> n >> m;
            vvi order(n);
            vi coord;
            for (int i = 0; i < n; ++i) {
                int s; std::cin >> s;
                order[i].resize(s);
                for (auto &it : order[i]) {
                    std::cin >> it;
                    coord.push_back(std::abs(it));
                }
            }
            std::sort(all(coord));
            reunique(coord);
            Graph graph(2 * isz(coord));
            for (auto &vec : order) {
                for (int i = 0; i < isz(vec); ++i) {
                    bool send = (vec[i] > 0);
                    vec[i] = lowpos(coord, std::abs(-vec[i]));
                    if (!send) { vec[i] += isz(coord); }
                    if (i > 0) {
                        graph.add_edge(vec[i-1], vec[i]);
                    }
                }
            }
            for (int i = 0; i < isz(coord); ++i) {
                graph.add_edge(i, i + isz(coord));
            }
            std::cout << (graph.solve() ? "YES\n" : "NO\n");
        }
    }
    return 0;
}

```

ЗАДАЧА №659

Перетягивание каната

(Время: 3 сек. Память: 16 Мб Сложность: 62%)

Для участия в соревнованиях по перетягиванию каната зарегистрировалось N человек. Некоторые из участников могут быть знакомы друг с другом. Причем, если двое из них имеют общего знакомого, то это не означает, что они обязательно знакомы друг с другом.

Организаторы соревнований заинтересованы в их качественном проведении. Они хотят разделить всех участников на две команды так, чтобы в первой команде было K человек, а во второй – $N-K$ человек. Из всех возможных вариантов формирования команд, организаторы хотят выбрать такой вариант, при котором сумма сплоченностей обеих команд максимальна. Сплоченностью команды называется количество пар участников этой команды, знакомых друг с другом. Ваша задача – помочь организаторам найти требуемое разделение участников на две команды.

Входные данные

В первой строке входного файла INPUT.TXT задаются три числа N , K , M , разделенные одиночными пробелами, где N – общее число зарегистрированных участников, K – требуемое количество человек в первой команде, M – количество пар участников, знакомых друг с другом.

Каждая из следующих M строк содержит два различных числа, разделенные пробелом – номера двух участников, знакомых друг с другом. Все участники нумеруются от 1 до N .

Ограничения: все числа целые, $0 < K < N < 25$, $0 \leq M \leq N(N-1)/2$

Выходные данные

Выходной файл OUTPUT.TXT должен содержать одну строку, состоящую из K чисел, каждое из которых задает номер участника, попавшего в первую команду. Числа должны быть разделены пробелами. Если существует несколько решений данной задачи, то выведите любое из них.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 3 3 1 3 2 5 5 4	5 2 4

```
#pragma GCC diagnostic ignored "-Wunused-result"
```

```
#include <stdio.h>
```

```
#include <bits/stdc++.h>
```

```
int C(int n, int k) {  
    if (k > n - k) {  
        return C(n, n-k);  
    }  
    int answer = 1;  
    for (int i = 1; i <= k; ++i) {  
        answer *= (n - i + 1);  
    }  
}
```



```

        answer /= i;
    }
    return answer;
}

int main() {
    int n, k, m; scanf("%d %d %d", &n, &k, &m);

    char is_pair[24][24] = {};
    for (int i = 0; i < m; ++i) {
        int a, b; scanf("%d %d", &a, &b); --a, --b;
        is_pair[a][b] = is_pair[b][a] = 1;
    }

    int perm[24] = {};
    for (int i = 0; i < k; ++i) perm[n-1-i] = 1;

    // Формирование всех перестановок:
    std::vector<int> permutations;
    permutations.reserve(C(n, k));
    do {
        int number = 0;
        for (int i = 0; i < n; ++i) {
            number |= (perm[i] << i);
        }
        permutations.push_back(number);
    } while (std::next_permutation(perm, perm+n));

    // Перебор всех перестановок:
    std::vector<int> answer(n*n, -1);
    for (auto perm : permutations) {
        int sum = 0;
        sum += (~((perm >> 0) & 1) ^ ((perm >> 1) & 1)) & is_pair[0][1];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 2) & 1)) & is_pair[0][2];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 3) & 1)) & is_pair[0][3];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 4) & 1)) & is_pair[0][4];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 5) & 1)) & is_pair[0][5];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 6) & 1)) & is_pair[0][6];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 7) & 1)) & is_pair[0][7];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 8) & 1)) & is_pair[0][8];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 9) & 1)) & is_pair[0][9];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 10) & 1)) & is_pair[0][10];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 11) & 1)) & is_pair[0][11];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 12) & 1)) & is_pair[0][12];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 13) & 1)) & is_pair[0][13];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 14) & 1)) & is_pair[0][14];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 15) & 1)) & is_pair[0][15];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 16) & 1)) & is_pair[0][16];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 17) & 1)) & is_pair[0][17];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 18) & 1)) & is_pair[0][18];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 19) & 1)) & is_pair[0][19];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 20) & 1)) & is_pair[0][20];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 21) & 1)) & is_pair[0][21];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 22) & 1)) & is_pair[0][22];
        sum += (~((perm >> 0) & 1) ^ ((perm >> 23) & 1)) & is_pair[0][23];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 2) & 1)) & is_pair[1][2];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 3) & 1)) & is_pair[1][3];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 4) & 1)) & is_pair[1][4];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 5) & 1)) & is_pair[1][5];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 6) & 1)) & is_pair[1][6];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 7) & 1)) & is_pair[1][7];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 8) & 1)) & is_pair[1][8];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 9) & 1)) & is_pair[1][9];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 10) & 1)) & is_pair[1][10];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 11) & 1)) & is_pair[1][11];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 12) & 1)) & is_pair[1][12];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 13) & 1)) & is_pair[1][13];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 14) & 1)) & is_pair[1][14];
        sum += (~((perm >> 1) & 1) ^ ((perm >> 15) & 1)) & is_pair[1][15];
    }
}

```

[illegible]

[illegible]

[illegible]

```

sum += (~(((perm >> 15) & 1) ^ ((perm >> 20) & 1))) & is_pair[15][20];
sum += (~(((perm >> 15) & 1) ^ ((perm >> 21) & 1))) & is_pair[15][21];
sum += (~(((perm >> 15) & 1) ^ ((perm >> 22) & 1))) & is_pair[15][22];
sum += (~(((perm >> 15) & 1) ^ ((perm >> 23) & 1))) & is_pair[15][23];
sum += (~(((perm >> 16) & 1) ^ ((perm >> 17) & 1))) & is_pair[16][17];
sum += (~(((perm >> 16) & 1) ^ ((perm >> 18) & 1))) & is_pair[16][18];
sum += (~(((perm >> 16) & 1) ^ ((perm >> 19) & 1))) & is_pair[16][19];
sum += (~(((perm >> 16) & 1) ^ ((perm >> 20) & 1))) & is_pair[16][20];
sum += (~(((perm >> 16) & 1) ^ ((perm >> 21) & 1))) & is_pair[16][21];
sum += (~(((perm >> 16) & 1) ^ ((perm >> 22) & 1))) & is_pair[16][22];
sum += (~(((perm >> 16) & 1) ^ ((perm >> 23) & 1))) & is_pair[16][23];
sum += (~(((perm >> 17) & 1) ^ ((perm >> 18) & 1))) & is_pair[17][18];
sum += (~(((perm >> 17) & 1) ^ ((perm >> 19) & 1))) & is_pair[17][19];
sum += (~(((perm >> 17) & 1) ^ ((perm >> 20) & 1))) & is_pair[17][20];
sum += (~(((perm >> 17) & 1) ^ ((perm >> 21) & 1))) & is_pair[17][21];
sum += (~(((perm >> 17) & 1) ^ ((perm >> 22) & 1))) & is_pair[17][22];
sum += (~(((perm >> 17) & 1) ^ ((perm >> 23) & 1))) & is_pair[17][23];
sum += (~(((perm >> 18) & 1) ^ ((perm >> 19) & 1))) & is_pair[18][19];
sum += (~(((perm >> 18) & 1) ^ ((perm >> 20) & 1))) & is_pair[18][20];
sum += (~(((perm >> 18) & 1) ^ ((perm >> 21) & 1))) & is_pair[18][21];
sum += (~(((perm >> 18) & 1) ^ ((perm >> 22) & 1))) & is_pair[18][22];
sum += (~(((perm >> 18) & 1) ^ ((perm >> 23) & 1))) & is_pair[18][23];
sum += (~(((perm >> 19) & 1) ^ ((perm >> 20) & 1))) & is_pair[19][20];
sum += (~(((perm >> 19) & 1) ^ ((perm >> 21) & 1))) & is_pair[19][21];
sum += (~(((perm >> 19) & 1) ^ ((perm >> 22) & 1))) & is_pair[19][22];
sum += (~(((perm >> 19) & 1) ^ ((perm >> 23) & 1))) & is_pair[19][23];
sum += (~(((perm >> 20) & 1) ^ ((perm >> 21) & 1))) & is_pair[20][21];
sum += (~(((perm >> 20) & 1) ^ ((perm >> 22) & 1))) & is_pair[20][22];
sum += (~(((perm >> 20) & 1) ^ ((perm >> 23) & 1))) & is_pair[20][23];
sum += (~(((perm >> 21) & 1) ^ ((perm >> 22) & 1))) & is_pair[21][22];
sum += (~(((perm >> 21) & 1) ^ ((perm >> 23) & 1))) & is_pair[21][23];
sum += (~(((perm >> 22) & 1) ^ ((perm >> 23) & 1))) & is_pair[22][23];
answer[sum] = perm;
}
std::vector<char> best;
for (int i = (int)answer.size()-1; i >= 0; --i) {
    if (answer[i] != -1) {
        // printf("i = %d\n", i);
        for (int j = 0; j < n; ++j) {
            best.push_back((answer[i] >> j) & 1);
        }
        break;
    }
}
for (int i = 0; i < n; ++i) {
    if (best[i] == 1) {
        printf("%d ", i+1);
    }
}
printf("\n");
return 0;
}

```

ЗАДАЧА №741

Замечательные дороги

(Время: 1 сек. Память: 16 Мб Сложность: 62%)

В одной замечательной стране живут замечательные люди. По исследованиям замечательного правительства, большинство граждан на выходных садятся в машину, выбирают циклический маршрут между некоторыми городами и деревнями без повторяющихся населенных пунктов и катаются по этому маршруту, пока не надоест. Некоторые, правда, катаются по своему городу и никуда не выезжают.

Так как правительство заботится о своих гражданах, оно хочет сделать их выходные максимально красочными. По этой причине недавно было принято решение покрасить каждую дорогу между населенными пунктами в какой-нибудь цвет. Причем так, чтобы ни на каком "выходном" маршруте не было дорог одинакового цвета. Но так как цветов могло понадобиться довольно много, правительство решило минимизировать количество различных цветов. Вам предстоит помочь этому замечательному государству в осуществлении его планов.

Входные данные

В первой строке находятся два числа: $1 \leq n \leq 50\,000$ - количество городов и деревень в стране и $1 \leq m \leq 100\,000$ - количество дорог. В m последующих строках находится по два числа – номера населенных пунктов, концов дороги. Ни одна дорога не ведет из города в себя, и между двумя населенными пунктами не может быть более одной дороги. Все дороги двусторонние.

Выходные данные

В первой строке выведите минимальное количество цветов. В последующих m строках выведите по три числа: два конца дороги в любом порядке и ее цвет. Дороги разрешается выводить в произвольном порядке.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 1 1 2	1 2 1 1
2	3 2 1 2 2 3	1 2 3 1 1 2 1
3	3 3 1 2 2 3 3 1	3 1 2 1 3 2 2 3 1 3

/*
Задача: 741. Замечательные дороги

Решение: графы, поиск в глубину, точки сочления, компоненты вершинной двусвязности, $O(n)$

Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru
*/

```

#include <bits/stdc++.h>
#define isz(x) (int)(x).size()

typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

struct Edge {
    int u, v, c;
    Edge(int u_ = 0, int v_ = 0, int c_ = 0)
        : u(u_), v(v_), c(c_) { }
};

struct Graph {
    const int n;
    vvi adj;
    std::vector<Edge> edges;
    vi tin, fup, stack;
    int timer;
    Graph(int n_) : n(n_) {
        adj.resize(1+n);
    }
    void add_edge(int u, int v) {
        adj[u].push_back(isz(edges));
        adj[v].push_back(isz(edges));
        edges.push_back(Edge(u,v));
    }
    void dfs(int u, int p) {
        tin[u] = fup[u] = ++timer;
        bool isRoot = (p == 0);
        int nChildren = 0;
        for (auto eid : adj[u]) {
            const int v = edges[eid].u + edges[eid].v - u;
            if (v == p) { continue; }
            if (!tin[v]) {
                ++nChildren;
                if (nChildren > 1 && isRoot) {
                    for (int color = 1; !stack.empty(); stack.pop_back()) {
                        edges[stack.back()].c = color++;
                    }
                }
                stack.push_back(eid);
                dfs(v, u);
                if (!isRoot && tin[u] <= fup[v]) {
                    int color = 1;
                    edges[eid].c = color++;
                    while (stack.back() != eid) {
                        edges[stack.back()].c = color++;
                        stack.pop_back();
                    }
                    stack.pop_back();
                }
                fup[u] = std::min(fup[u], fup[v]);
            } else if (tin[u] > tin[v]) {
                stack.push_back(eid);
            }
            fup[u] = std::min(fup[u], tin[v]);
        }
    }
    int solve() {
        tin.assign(1+n, 0);
        fup.assign(1+n, 0);
        stack.clear();
        timer = 0;
        for (int u = 1; u <= n; ++u) {
            if (!tin[u]) {
                dfs(u, 0);
                for (int color = 1; !stack.empty(); stack.pop_back()) {
                    edges[stack.back()].c = color++;
                }
            }
        }
    }
};

```

```

        }
    }
    int max = 0;
    for (auto &e : edges) { assert(e.c != 0); max = std::max(max, e.c); }
    return max;
}

};

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int nV, nE;
    while (std::cin >> nV >> nE) {
        Graph graph(nV);
        for (int i = 0; i < nE; ++i) {
            int u, v; std::cin >> u >> v;
            graph.add_edge(u,v);
        }
        std::cout << graph.solve() << '\n';
        for (const auto &e : graph.edges) {
            std::cout << e.u << ' ' << e.v << ' ' << e.c << "\n";
        }
    }
    return 0;
}

```


ЗАДАЧА №776

Юпитер

(Время: 1 сек. Память: 64 Мб Сложность: 62%)

На поверхности Юпитера приземлился летающий робот, его задачей является исследование поверхности планеты. Поверхность Юпитера представляет собой прямоугольное поле из $N \times M$ клеток, каждая из которых представляет лаву или твердую поверхность различной высоты. Для снятия проб грунта роботу необходимо переместиться из клетки с координатами (1,1) в клетку с координатами (X,Y). Для передвижения робот может использовать два типа действий - это переезд и перелет.

Переезд - перемещение в одну из четырех соседних клеток, имеющих общую грань с заданной, при этом высота клеток не должна отличаться больше чем на единицу.

Перелет из одной клетки в другую позволяет преодолевать препятствия, но для него нужна энергия, которая расходуется из специальных аккумуляторов, количество которых на борту ограничено и равно K. Дальность перелета ограничена мощностью одного аккумулятора и составляет D единиц. Перелет возможен только по направлениям, параллельным границам поля. Во время перелета луноход не может менять направление, но при этом может менять высоту, облетая препятствия. На каждое перемещение на одну клетку вдоль выбранного направления, вверх или вниз на одну единицу по высоте, тратится ровно одна единица энергии. После перелета аккумулятор утилизируется и больше использоваться не может, оставшаяся в нем энергия пропадает.

Одним действием назовем один переезд или один перелет. Ваша задача для заданной поверхности найти наименьшее количество действий, необходимых для достижения заданной клетки.

Входные данные

Первая строка входного файла INPUT.TXT содержит размеры поля N, M ($1 \leq N, M \leq 100$), разделенные пробелом. Во второй строке идут координаты клетки, куда необходимо найти путь X,Y ($1 \leq X \leq N, 1 \leq Y \leq M$). На третьей строке через пробел указаны целые K и D ($0 \leq K \leq 10, 0 \leq D \leq 100$). Далее в M строках идут по N чисел через пробел – высотные отметки участка Юпитера, высота каждой клетки - целое число, лежащее в диапазоне от 0 до 10000 включительно. Высота 0 (ноль) обозначает лаву, на которой останавливаться нельзя, но можно пролететь над ней.

Выходные данные

В выходной файл OUTPUT.TXT выведите целое число - минимальное количество действий, необходимых для достижения заданной клетки. Если добраться до заданной клетки нельзя, то необходимо вывести в строке слово IMPOSSIBLE.

Примеры

№	INPUT.TXT	OUTPUT.TXT
---	-----------	------------

1	5 4 5 3 1 6 2 1 4 2 1 1 2 4 2 1 4 4 6 2 1 2 2 2 2 1	5
2	4 4 4 2 1 3 2 0 0 3 3 0 4 3 4 0 5 2 4 5 5 1	3
3	3 3 3 3 10 10 1 0 0 1 0 0 0 1 1	IMPOSSIBLE

```

/*
    Задача: 776. Юпитер

    Решение: графы, поиск в ширину,  $O(k \cdot n \cdot m \cdot (n+m))$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::vector<vvi> vvvi;
typedef const int cint;
template<typename T>
std::istream& operator>>(std::istream& is, std::vector<T>& vec) {
    for (auto &it : vec) { is >> it; }
    return is;
}

const int INF = (int)1e9+7;
struct Pos { int k, r, c; };

int solve(cint tr, cint tc, cint nRows, cint nCols, cint k, cint d, const vvi& h) {
    vvvi dist(1+k, vvi(nRows, vi(nCols, INF)));
    dist[k][0][0] = 0;
    std::queue<Pos> queue;
    queue.push(Pos{k,0,0});
    while (!queue.empty()) {
        auto curr = queue.front(); queue.pop();
        const int r = curr.r;
        const int c = curr.c;
        for (int dr = -1; dr <= 1; ++dr) {
            for (int dc = -1; dc <= 1; ++dc) {
                if (dr * dr + dc * dc != 1) { continue; }
                int nr = curr.r + dr;
                int nc = curr.c + dc;
                if (nr < 0 || nr >= nRows || nc < 0 || nc >= nCols) {
                    continue;
                }
                const int newDist = dist[curr.k][r][c] + 1;
                if (h[nr][nc] > 0 && std::abs(h[r][c] - h[nr][nc]) <= 1 && dist[curr.k][nr][nc] > newDist) {
                    dist[curr.k][nr][nc] = newDist;
                    queue.push(Pos{curr.k,nr,nc});
                }
            }
        }
    }
}

```

```

        if (curr.k == 0) { continue; }
        int max = h[r][c];
        while (!(nr < 0 || nr >= nRows || nc < 0 || nc >= nCols)) {
            max = std::max(max, h[nr][nc]);
            int need = max - h[r][c] + max - h[nr][nc] + std::abs(nr - r) + std::abs(nc - c);
            if (need <= d && h[nr][nc] > 0 && dist[curr.k-1][nr][nc] == INF) {
                dist[curr.k-1][nr][nc] = newDist;
                queue.push(Pos{curr.k-1, nr, nc});
            }
            nr += dr;
            nc += dc;
        }
    }
}

int min = INF;
for (int i = 0; i <= k; ++i) { min = std::min(min, dist[i][tr][tc]); }
return min;
}

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int nCols, nRows;
    while (std::cin >> nCols >> nRows) {
        int tc, tr, k, d; std::cin >> tc >> tr; --tc, --tr;
        vvi h(nRows, vi(nCols)); std::cin >> k >> d >> h;
        int ans = solve(tr, tc, nRows, nCols, k, d, h);
        std::cout << (ans == INF ? "IMPOSSIBLE" : std::to_string(ans)) << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №813

Игра в 24

(Время: 1 сек. Память: 16 Мб Сложность: 62%)

«Игра в 24» - это математическая игра, в которой используются специальные карточки. На каждой карточке записаны четыре числа. Задача игроков состоит в том, чтобы получить число 24, используя эти числа и арифметические операции (сложение, вычитание, умножение и деление, скобки при этом можно расставить произвольным образом). «Игра в 24» используется в некоторых школах при изучении математики в начальных классах.

В этой задаче рассматривается упрощенный вариант этой игры, в котором в число разрешенных операций входят только сложение, вычитание и умножение.

Назовем карточку для упрощенной «Игры в 24» правильной, если из указанных на ней чисел с помощью сложения, вычитания, умножения и расстановки скобок произвольным образом можно получить число 24.

Фирма *American Card Manufacturer* (ACM) занимается выпуском наборов карточек для этой игры. Однако, выпуск таких карточек сопряжен с некоторыми трудностями. Одна из них состоит в том, что не любой набор из четырех чисел задает «правильную» карточку.

По этой причине задача проверки «правильности» данной карточки является весьма актуальной. Ваша задача состоит в написании программы, которая будет осуществлять указанную проверку.

Входные данные

Входной файл INPUT.TXT содержит четыре натуральных числа, не превосходящих 30, которые написаны на исследуемой карточке.

Выходные данные

В выходной файл OUTPUT.TXT выведите слово YES, если карточка является правильной, и слово NO в противном случае.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 2 3 4	YES
2	1 1 1 1	NO

```
/*
    Задача: 813. Игра в 24

    Решение: перебор, рекурсия,  $O(n! \cdot 3^n)$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/
```

```
#include <bits/stdc++.h>
#define isz(x) (int)(x).size()
typedef std::vector<int> vi;
bool ok;
```

```

void brute(vi arr) {
    if (isz(arr) == 1) {
        ok = ok || (arr[0] == 24);
        return;
    }
    for (int i = 0; i < isz(arr); ++i) {
        for (int j = i + 1; j < isz(arr); ++j) {
            auto fi = arr[i];
            auto se = arr[j];
            arr.erase(arr.begin()+j);
            arr[i] = fi * se;
            brute(arr);
            arr[i] = fi - se;
            brute(arr);
            arr[i] = fi + se;
            brute(arr);
            arr[i] = fi;
            arr.insert(arr.begin()+j, se);
        }
    }
}

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    vi arr(4); for (auto &it : arr) { std::cin >> it; }
    brute(arr);
    std::cout << (ok ? "YES\n" : "NO\n");
    return 0;
}

```

ЗАДАЧА №1176

Минимумы в прямоугольнике

(Время: 2 сек. Память: 32 Мб Сложность: 62%)

Дана прямоугольная матрица целых чисел размером $M \times N$. Необходимо выполнить K операций вычисления минимального элемента в прямоугольнике, принадлежащем исходной матрице, с координатами левого верхнего угла (x_1, y_1) и правого нижнего (x_2, y_2) .

Входные данные

В первой строке входного файла INPUT.TXT записаны 3 числа: N и M – число строк и столбцов матрицы ($1 \leq N, M \leq 300$) и K - количество запросов ($1 \leq K \leq 10^5$). Каждая из следующих N строк содержит по M чисел - элементы A_{ij} соответствующей строки матрицы ($-10^9 \leq A_{ij} \leq 10^9$). Последующие K строк содержат по 4 целых числа - y_1, x_1, y_2 и x_2 - запрос на минимальный элемент в прямоугольнике ($1 \leq y_1 \leq y_2 \leq N, 1 \leq x_1 \leq x_2 \leq M$).

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса в отдельной строке выведите результат поиска минимального элемента.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4 5 3	0 1 3
	2 2 3 4 3	
	7 9 3 6 7	
	5 1 4 7 0	
	1 8 1 8 7	
	1 1 4 5	
	1 2 3 3	
	2 2 2 4	

```
/*
    Задача: 1176. Минимумы в прямоугольнике

    Решение: Двумерная Sparse Table,  $O(n*m*\log(n)*\log(m)+k)$ 

    Автор: Дмитрий Козырев, github: dmkgz, e-mail: dmkozyrev@rambler.ru
*/

#pragma GCC optimize ("O3")
#pragma GCC target ("sse,sse2,sse3,ssse3,sse4.1,sse4.2,sse4a")
// #pragma GCC target ("avx,avx2")

#include <iostream>
#include <algorithm>
#include <cassert>
#include <cstdlib>
#include <functional>
#include <iomanip>
#include <random>
```

```

namespace SparseTable {

    const int NMAX = 300, PMAX = 8, INF = (int)1e9+7;

    int data[1+PMAX][1+PMAX][1+NMAX][1+NMAX], log2[1+NMAX];

    void init() {
        log2[0] = log2[1] = 0;
        for (int i = 2; i <= NMAX; ++i) {
            log2[i] = log2[i / 2] + 1;
        }
        std::fill(&data[0][0][0][0], &data[0][0][0][0] + (1+PMAX) * (1+PMAX) * (1+NMAX) * (1+NMAX), I
    }

    void set(int r, int c, int x) {
        data[0][0][r][c] = x;
    }

    void build() {
        for (int pc = 1; pc <= PMAX; ++pc) {
            for (int r = 1; r <= NMAX; ++r) {
                for (int c = 1; c + (1 << pc) - 1 <= NMAX; ++c) {
                    int v1 = data[0][pc-1][r][c];
                    int v2 = data[0][pc-1][r][c+(1 << (pc - 1))];
                    data[0][pc][r][c] = std::min(v1, v2);
                }
            }
        }
        for (int pr = 1; pr <= PMAX; ++pr) {
            for (int pc = 0; pc <= PMAX; ++pc) {
                for (int r = 1; r + (1 << pr) - 1 <= NMAX; ++r) {
                    for (int c = 1; c + (1 << pc) - 1 <= NMAX; ++c) {
                        int v1 = data[pr-1][pc][r][c];
                        int v2 = data[pr-1][pc][r+(1 << (pr-1))][c];
                        data[pr][pc][r][c] = std::min(v1, v2);
                    }
                }
            }
        }
    }

    int get(int r1, int c1, int r2, int c2) {
        if (r1 > r2) std::swap(r1, r2);
        if (c1 > c2) std::swap(c1, c2);
        int pr = log2[r2-r1+1];
        int pc = log2[c2-c1+1];
        r2 = r2 - (1 << pr) + 1;
        c2 = c2 - (1 << pc) + 1;
        return std::min({
            data[pr][pc][r1][c1], data[pr][pc][r1][c2],
            data[pr][pc][r2][c1], data[pr][pc][r2][c2]
        });
    }

}

int main() {
    int nRows, nCols, nQueries;
    scanf("%d %d %d", &nRows, &nCols, &nQueries);
    SparseTable::init();
    for (int r = 1; r <= nRows; ++r) {
        for (int c = 1; c <= nCols; ++c) {
            int v; scanf("%d", &v);
            SparseTable::set(r, c, v);
        }
    }
    SparseTable::build();
    while (nQueries--) {
        int r1, c1, r2, c2;

```

```
        scanf("%d %d %d %d", &r1, &c1, &r2, &c2);
        printf("%d\n", SparseTable::get(r1,c1,r2,c2));
    }
    return 0;
}
```


ЗАДАЧА №1387

Остовное дерево

(Время: 2 сек. Память: 16 Мб Сложность: 62%)

Требуется найти в связном неориентированном графе остовное дерево минимального веса.

Входные данные

Первая строка входного файла INPUT.TXT содержит два натуральных числа N и M – количество вершин и ребер графа соответственно ($1 \leq N \leq 20\,000$, $0 \leq M \leq 100\,000$). Следующие M строк содержат описание ребер по одному на строке. Ребро номер i описывается тремя натуральными числами B_i , E_i и W_i – номера концов ребра и его вес соответственно ($1 \leq B_i, E_i \leq N$, $0 \leq W_i \leq 100\,000$).

Выходные данные

В выходной файл OUTPUT.TXT выведите целое число – вес минимального остовного дерева.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4 4 1 2 1 2 3 2 3 4 5 4 1 4	7

```
/*
    Задача: 1387. Остовное дерево

    Решение: графы, алгоритм Краскала, DSU, O(n log(n))

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <vector>
#include <cassert>

typedef long long ll;

struct Edge {
    int u, v, cost;
};

bool operator<(const Edge& a, const Edge& b) {
    if (a.cost < b.cost) return true;
    if (a.cost > b.cost) return false;
    return a.u < b.u || (a.u == b.u && a.v < b.v);
}

struct DSU {
```

```

std::vector<int> parent, size;
DSU(int n) : parent(1+n,0), size(1+n,1) {
    for (int i = 1; i <= n; ++i) parent[i] = i;
}
int get_parent(int a) {
    return parent[a] == a ? a : parent[a] = get_parent(parent[a]);
}
void union_sets(int a, int b) {
    a = get_parent(a), b = get_parent(b);
    if (b != a) {
        if (size[a] < size[b]) std::swap(a,b);
        size[a] += size[b];
        parent[b] = a;
    }
}
};

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    DSU dsu(1+n);
    std::vector<Edge> edges(m);
    for (auto& e : edges) {
        scanf("%d %d %d", &e.u, &e.v, &e.cost);
    }
    std::sort(edges.begin(), edges.end());
    ll answ = 0;
    for (auto& e : edges) {
        int pu = dsu.get_parent(e.u), pv = dsu.get_parent(e.v);
        if (pu != pv) {
            answ += e.cost;
            dsu.union_sets(e.u,e.v);
        }
    }
    std::cout << answ;
    return 0;
}

```

ЗАДАЧА №963

Морфизм

(Время: 2 сек. Память: 16 Мб Сложность: 63%)

Рассмотрим слова, состоящие из первых n букв английского алфавита. Морфизм – это функция f , которая по букве возвращает слово. Рассмотрим пример морфизма: $f(A) = ABC$, $f(B) = A$, $f(C) = BC$.

Если мы рассмотрим слово $w = c_1 c_2 \dots c_l$ и применим к нему морфизм f , мы получим слово $f(w) = f(c_1)f(c_2)\dots f(c_l)$. Например, для морфизма из предыдущего параграфа $f(ABC) = ABCABC$.

Мы можем применять морфизм к слову несколько раз. Положим $f_0(w) = w$, и для $k > 0$ положим $f_k(w) = f(f_{k-1}(w))$.

По заданному морфизму f , слову w , числу k и числу p , найдите p -й символ слова $f_k(w)$.

Входные данные

Первая строка входного файла INPUT.TXT содержит числа n , k и p ($1 \leq n \leq 10$, $0 \leq k \leq 10^9$, $1 \leq p \leq 20$). Вторая строка входного файла содержит слово w . Его длина не превышает 50. Следующие n строк содержат $f(A)$, $f(B)$, и т.д. Каждое значение – это строка, содержащая от 1 до 50 символов.

Выходные данные

В выходной файл OUTPUT.TXT выведите p -й символ $f_k(w)$, или «-» (минус) - если такой символ отсутствует.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 1 5 ABC ABC A BC	B
2	3 1 7 ABC ABC A BC	-

```
#include <bits/stdc++.h>
/*
    Задача: 963. Морфизм

    Решение: рекурсия, разделяй и властвуй, мемоизация, строки, std::map, O(log(k) * p)

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#define isz(x) (int)(x).size()
```

```

typedef std::vector<std::string> vs;
typedef std::pair<std::string, int> psi;
std::map<psi, std::string> cache;
std::string solve(const vs& f, std::string x, const int k, const int w) {
    if (isz(x) > w) { x.resize(w); }
    if (k == 0) { return x; }
    if (k == 1) {
        std::string answ;
        for (auto it : x) {
            answ += f[it];
            if (isz(answ) > w) {
                answ.resize(w);
                break;
            }
        }
        return answ;
    }
    auto key = psi(x, k);
    auto it = cache.find(key);
    if (it == cache.end()) {
        return cache[key] = solve(f, solve(f, x, k / 2, w), k - k / 2, w);
    }
    return it->second;
}

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int n, k, p;
    while (std::cin >> n >> k >> p) {
        std::string x; std::cin >> x;
        vs f(256);
        for (char c = 'A'; c < 'A' + n; ++c) { std::cin >> f[c]; }
        cache.clear();
        x = solve(f, x, k, 20);
        if (isz(x) > p-1) { std::cout << x[p-1] << std::endl; }
        else { std::cout << "-" << std::endl; }
    }
    return 0;
}

```

ЗАДАЧА №746

Гонки

(Время: 2 сек. Память: 16 Мб Сложность: 64%)

В области L находится n городов. Некоторые пары городов соединены проселочной дорогой с двусторонним движением. Начавшись в каком-то городе, дорога не может закончиться в нем же. В этом году состояние дорог позволило отделению ГИБДД области L провести гонки под лозунгом «Скажем НЕТ нарушениям скоростного режима». Было решено, что круговая трасса должна состоять из четырех дорог, но не может проходить через один город два раза. Естественно, свернуть с одной дороги на другую можно только в городе. Организаторы уже должны приступить к составлению отчета, и для этого требуется посчитать количество различных трасс.

Входные данные

В первой строке входного файла INPUT.TXT записаны количество городов n ($1 \leq n \leq 300$) и количество дорог m . В каждой из следующих m строк содержится два различных числа — номера городов, соединенных соответствующей дорогой.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число — количество круговых трасс из четырех дорог, которые могут составить организаторы.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4 6 1 2 2 3 3 4 4 1 1 3 2 4	3

```
/*
    Задача: 746. Гонки

    Решение: графы, комбинаторика,  $O(n^3)$ 

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0);
    int n, m;
    while (std::cin >> n >> m) {
        vvi adj(1+n);
        while (m--) {
            int u, v; std::cin >> u >> v;
```

```

        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    vi cnt(1+n);
    auto calc = [&](int u, int v) {
        int ret = 0;
        for (int next : adj[u]) {
            cnt[next]++;
        }
        for (int next : adj[v]) {
            cnt[next]++;
            ret += (cnt[next] == 2);
        }
        ret -= (cnt[u] == 2);
        ret -= (cnt[v] == 2);
        for (int next : adj[v]) { cnt[next]--; }
        for (int next : adj[u]) { cnt[next]--; }
        return ret;
    };
    int64_t answ = 0;
    for (int u = 1; u <= n; ++u) {
        for (int v = u+1; v <= n; ++v) {
            int ret = calc(u,v);
            answ += ret * (ret-1) / 2;
        }
    }
    std::cout << answ / 2 << std::endl;
}
return 0;
}

```

ЗАДАЧА №749

Неправильный RSA

(Время: 3 сек. Память: 16 Мб Сложность: 64%)

Рома, Сережа и Андрюша решили улучшить знаменитый алгоритм шифрования RSA. Они решили, что в RSA в качестве модуля можно использовать в качестве числа n не только произведение двух простых чисел, но и произведение вида $n = p^k q^k$, где p и q – простые числа, а k – некоторое натуральное число.

Однако Коля указал, что помимо различных математических трудностей, новая схема может оказаться менее устойчивой к взлому. А именно, большое число, равное произведению двух различных простых чисел, тяжело разложить на множители, в частности, поскольку у него существует ровно одно нетривиальное разложение. А у числа вида $n = p^k q^k$ их может быть больше. Например, у числа $100 = 2^2 \cdot 5^2$ есть целых восемь нетривиальных разложений на множители: $2 \cdot 50$, $2 \cdot 2 \cdot 25$, $2 \cdot 2 \cdot 5 \cdot 5$, $2 \cdot 5 \cdot 10$, $4 \cdot 25$, $4 \cdot 5 \cdot 5$, $5 \cdot 20$ и $10 \cdot 10$.

Теперь Рома, Сережа и Андрюша думают – сколько же различных нетривиальных разложений на множители есть у числа $n = p^k q^k$?

Входные данные

Входной файл INPUT.TXT содержит число n ($6 \leq n \leq 10^{18}$, гарантируется, что $n = p^k q^k$ для различных простых p и q и натурального k).

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число — количество нетривиальных разложений на множители числа n .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	6	1
2	100	8

```
/*
    Задача: 749. Неправильный RSA

    Решение: рекурсия, перебор, разложение на множители, мемоизация

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/
```

```
#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
#define isz(x) (int)(x).size()
typedef long long ll;
typedef std::pair<ll,ll> pll;
struct Brute {
    std::vector<ll> data;
    std::map<pll, ll> cache;
```

```

ll solve(ll last, ll prod) {
    if (prod == 1) { return 1; }
    auto key = pll(last, prod);
    auto it = cache.find(key);
    if (it == cache.end()) {
        ll answ = 0;
        for (auto curr = std::lower_bound(all(data), last); curr != data.end() && *curr
            if (prod % (*curr) == 0) {
                answ += solve(*curr, prod / *curr);
            }
        }
        if (isz(cache) < (1 << 18)) {
            return cache[key] = answ;
        }
        return answ;
    }
    return it->second;
}

ll operator()(int x, int y) {
    data.clear();
    ll pp = 1;
    for (int cx = 0; cx <= x; ++cx) {
        ll pq = 1;
        for (int cy = 0; cy <= y; ++cy) {
            if (cx + cy > 0) {
                data.push_back(pq * pp);
            }
            pq *= 3;
        }
        pp *= 2;
    }
    std::sort(all(data));
    data.erase(std::unique(all(data)), data.end());
    return solve(2, data.back()) - 1;
}

};

ll solve(ll n) {
    int k = 0;
    for (ll x = 2; x * x <= n && x <= (1 << 20); ++x) {
        if (n % x == 0) {
            while (n % x == 0) { n /= x; k++; }
            break;
        }
    }
    if (k == 0) { return 1; }
    return Brute()(k, k);
}

int main() {
    for (ll n; std::cin >> n; std::cout << solve(n) << "\n");
    return 0;
}

```


ЗАДАЧА №989

Окопы

(Время: 1 сек. Память: 16 Мб Сложность: 64%)

Однажды Андрей и Петя решили пострелять друг в друга из пистолетов. Нет, конечно, не из настоящих, а из игрушечных. Для того, чтобы игра стала еще более интересной, каждый из них вырыл себе окоп. Окоп представляет собой отрезок, соединяющий две точки. Известно, что окопы Андрея и Пети не пересекаются, более того, они вообще не имеют общих точек.

Теперь им предстоит выбрать «линию фронта», такую прямую, которую в процессе игры пересекать запрещается. Только вот проблема – окопы уже вырыты, а ребята не могут сообразить, как им провести линию, так чтобы окопы оказались по разные стороны от нее и не имели общих точек с ней. Так как рытье окопов дело трудоемкое, то они попросили помощи у Вас.

Входные данные

Входной файл INPUT.TXT содержит не более 1000 тестов. Каждый тест описывается двумя строками. Первая строка теста содержит 4 целых числа x_1, y_1, x_2, y_2 – координаты концов отрезка, задающего первый окоп. Вторая строка теста в аналогичном формате описывает второй окоп. Входной файл заканчивается двумя строчками, каждая из которых содержит 4 нуля. Тесты разделены пустой строкой. Все координаты не превышают 10^4 по абсолютной величине.

Выходные данные

В выходной файл OUTPUT.TXT выведите числа a, b, c – коэффициенты уравнения прямой $ax + by + c = 0$, разделяющей окопы. Числа a, b, c должны быть целыми и не должны превышать 10^9 по абсолютной величине.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	0 1 1 2	1 -1 0 1 1 0
	0 -1 1 0	
	0 1 -1 2	
	0 -1 -1 0	
	0 0 0 0	
	0 0 0 0	

```
/*
    Задача: 989. Окопы

    Решение: продвинутая геометрия, точка, дробь, прямая, O(q)

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>

typedef long long ll;
```

```

ll gcd(ll a, ll b) { return b == 0 ? a : gcd(b, a % b); }
ll gcd(ll a, ll b, ll c) { return gcd(a, gcd(b, c)); }
ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
ll lcm(ll a, ll b, ll c) { return a / gcd(a, b, c) * b * c; }

struct Frac {
    ll p, q;
    Frac(ll p_ = 0, ll q_ = 1) : p(p_), q(q_) { normalize(); }
    Frac& normalize() {
        if (q < 0) { p = -p; q = -q; }
        if (p == 0) { q = 1; }
        auto g = gcd(std::abs(p), std::abs(q)); p /= g; q /= g;
        return *this;
    }
    Frac operator-() const { return Frac(-p, q); }
    Frac operator+(const Frac& f) const { return Frac(p * f.q + q * f.p, q * f.q); }
    Frac operator-(const Frac& f) const { return *this + (-f); }
    Frac operator*(const Frac& f) const { return Frac(p * f.p, q * f.q); }
    Frac operator/(const Frac& f) const { return Frac(p * f.q, q * f.p); }
};

bool operator<(const Frac& a, const Frac& b) { return (a - b).p < 0; }

template<typename T> bool operator>(const T& a, const T& b) { return b < a; }
template<typename T> bool operator>=(const T& a, const T& b) { return !(a < b); }
template<typename T> bool operator<=(const T& a, const T& b) { return !(a > b); }
template<typename T> bool operator!=(const T& a, const T& b) { return a < b || b < a; }
template<typename T> bool operator==(const T& a, const T& b) { return !(a != b); }

struct Point {
    Frac x, y;
    Point(Frac x_ = Frac(), Frac y_ = Frac()) : x(x_), y(y_) { }
    Point operator-() const { return Point(-x, -y); }
    Point operator+(const Point& p) const { return Point(x + p.x, y + p.y); }
    Point operator-(const Point& p) const { return *this + (-p); }
    Point operator*(const Frac& val) const { return Point(x * val, y * val); }
    Point operator/(const Frac& val) const { return Point(x / val, y / val); }
    Point ort() const { return Point(-y, x); }
};

bool operator<(const Point& a, const Point& b) { return a.x < b.x || (a.x == b.x && a.y < b.y); }

Frac dot(const Point& a, const Point& b) { return a.x * b.x + a.y * b.y; }
Frac cross(const Point& a, const Point& b) { return a.x * b.y - a.y * b.x; }

struct Line {
    Frac A, B, C;
    Line(Point a, Point b) {
        // dot((x - a), (b-a).ort()) == 0
        auto norm = (b-a).ort();
        A = norm.x; B = norm.y; C = dot(-a, norm);
        auto l = lcm(A.q, B.q, C.q);
        A = A * l; B = B * l; C = C * l;
        auto g = gcd(std::abs(A.p), std::abs(B.p), std::abs(C.p));
        A = A / g; B = B / g; C = C / g;
        if (A < Frac(0)) { A = -A; B = -B; C = -C; }
        if (A == Frac(0) && B < Frac(0)) { B = -B; C = -C; }
        assert(A != Frac(0) || B != Frac(0));
    }
    int sign(Point p) const {
        auto val = A * p.x + B * p.y + C;
        return val > Frac(0) ? +1 : val < Frac(0) ? -1 : 0;
    }
};

auto solve(Point a, Point b, Point c, Point d) {
    Point p, q;
    if (cross(b-a, d-c) == Frac(0) && cross(c-a, b-a) == Frac(0)) {

```

```

        Point t = a, k = c;
        for (auto i : {a,b}) {
            for (auto j : {c,d}) {
                if (dot(i-j,i-j) < dot(t-k,t-k)) {
                    t = i; k = j;
                }
            }
        }
        p = (t + k) / 2;
        q = p + (k-t).ort();
    } else {
        p = a + (c-a) / 2;
        q = b + (c-a) / 2;
    }
    return Line(p,q);
}

bool check(Line L, Point a, Point b, Point c, Point d) {
    return (L.sign(a) * L.sign(b) == 1 && L.sign(c) * L.sign(d) == 1 && L.sign(a) * L.sign(
}

auto brute(Point a, Point b, Point c, Point d) {
    for (auto i : {a,b}) {
        for (auto j : {c,d}) {
            auto res = solve(i,a+b-i,j,c+d-j);
            if (check(res,a,b,c,d)) { return res; }
            res = solve(j,c+d-j,i,a+b-i);
            if (check(res,a,b,c,d)) { return res; }
        }
    }
    assert(false);
}

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int x1, y1, x2, y2, x3, y3, x4, y4;
    while (std::cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3 >> x4 >> y4) {
        Point a(x1,y1), b(x2,y2), c(x3,y3), d(x4,y4);
        if (a == b && a == c && a == d) { break; }
        auto res = brute(a,b,c,d);
        for (auto it : {res.A.p, res.B.p, res.C.p}) { std::cout << it << ' '; }
        std::cout << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №1182

Билеты на электричку

(Время: 1 сек. Память: 16 Мб Сложность: 64%)

В новых элитных электричках каждому пассажиру положено сидячее место. Естественно, количество сидячих мест ограничено и на всех их может не хватить. Маршрут электрички проходит через N станций, пронумерованных от 0 до $N-1$. Когда человек хочет купить билет, он называет два числа X и Y – номера станций, откуда и куда он хочет ехать. При наличии хотя бы одного сидячего места между каждыми двумя соседними станциями этого участка на момент покупки ему продается билет, иначе выдается сообщение «билетов нет» и билет не продается.

Ваша задача – написать программу, обслуживающую такого рода запросы в порядке их прихода.

Входные данные

В первой строке входного файла INPUT.TXT записаны три натуральных числа: N , K и M – количество станций, количество мест в электричке и количество запросов соответственно ($2 \leq N \leq 10\,000$, $K \leq 1000$, $M \leq 50\,000$). В следующих M строках описаны запросы, каждый из которых состоит из двух целых чисел X и Y ($0 \leq X < Y < N$).

Выходные данные

В выходной файл OUTPUT.TXT выведите в отдельной строке для каждого запроса «Yes», если билет был продан, и «No» в противном случае.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 2 4	
	0 4	Yes
	1 2	Yes
	1 4	No
	2 4	Yes

```
#include <stdio.h>
#include <vector>
#include <string>
#include <algorithm>

void assert_tle(bool q) {
    if (!q) {
        while (true);
    }
}

struct SqrtDecomposition {

    struct Group {
        int begin, after, add;
        std::vector<int> values, sorted;

        Group(int begin, int after, const std::vector<int>& arr)
```

```

        : begin(begin), after(after)
    {
        add = 0;
        values = arr;
        sorted = values;
        std::sort(sorted.begin(), sorted.end());
        assert_tle(after-begin == (int)values.size());
    }

    bool include(int x) const {
        x -= add;
        return binary_search(sorted.begin(), sorted.end(), x);
    }

    bool include(int l, int r, int x) const {
        assert_tle(begin <= l && l <= r && r < after);
        if (l == begin && r+1 == after) {
            return include(x);
        }
        for (int i = l; i <= r; ++i) {
            if (values[i-begin] + add == x) {
                return true;
            }
        }
        return false;
    }

    void inc(int x) {
        add += x;
    }

    void inc(int l, int r, int x) {
        assert_tle(begin <= l && l <= r && r < after);
        if (l == begin && r+1 == after) {
            inc(x);
            return;
        }
        for (int i = begin; i < after; ++i) {
            values[i-begin] += add;
            if (l <= i && i <= r) {
                values[i-begin] += x;
            }
        }
        add = 0;
        sorted = values;
        std::sort(sorted.begin(), sorted.end());
    }

};

std::vector<Group> groups;

const int GSIZE = 64;

SqrtDecomposition(const std::vector<int>& arr) {
    int count = ((int)arr.size() + GSIZE - 1) / GSIZE;
    for (int g = 0; g < count; ++g) {
        auto begin = arr.begin() + g * GSIZE;
        auto after = begin + std::min(GSIZE, int(arr.end()-begin));
        groups.push_back(
            Group(begin - arr.begin(), after-arr.begin(), std::vector<int>(begin, after
        ));
    }
}

bool include(int l, int r, int x) {
    int gl = l / GSIZE;
    int gr = r / GSIZE;
    if (gl == gr) {

```

```

        return groups[gl].include(l, r, x);
    } else {
        bool flag = groups[gl].include(l, (gl+1)*GSIZE-1, x);
        if (flag) return true;
        flag = groups[gr].include(gr*GSIZE, r, x);
        if (flag) return true;
        for (int g = gl+1; g < gr; ++g) {
            if (groups[g].include(x)) {
                return true;
            }
        }
        return false;
    }
}

void inc(int l, int r, int x) {
    int gl = l / GSIZE;
    int gr = r / GSIZE;
    if (gl == gr) {
        groups[gl].inc(l, r, x);
    } else {
        groups[gl].inc(l, (gl+1)*GSIZE-1, x);
        groups[gr].inc(gr*GSIZE, r, x);
        for (int g = gl+1; g < gr; ++g) {
            groups[g].inc(x);
        }
    }
}

};

int main() {
    int n, limit, q;
    scanf("%d %d %d", &n, &limit, &q);
    SqrtDecomposition sd(std::vector<int>(n, 0));
    while (q--) {
        int l, r;
        scanf("%d %d", &l, &r);
        if (sd.include(l, r-1, limit)) {
            printf("No\n");
        } else {
            sd.inc(l, r-1, 1);
            printf("Yes\n");
        }
    }
    return 0;
}

```

ЗАДАЧА №705

Оценка

(Время: 1 сек. Память: 16 Мб Сложность: 65%)

Вася очень любит программировать. Еще он очень любит научную фантастику. Как-то на каникулах, начитавшись книжек со своим другом Петей, Вася решил вывести формулу, показывающую рост популяции на Марсе. По прикидкам Васи после N лет жизни планеты популяция марсиан должна составлять S_N марсиан, где S_N определяется по формуле:

$$S_N = \sum_{k=0}^{N-1} k^2 \cdot 2^k$$

Петя, после того как Вася рассказал ему свою теорию, засомневался и решил вычислить это число на компьютере, чтобы убедиться в его достоверности. К тому же, Петя недавно прочитал умную книжку по программированию, поэтому он думает, что это не займет много времени.

Вам предлагается сделать то же самое, но без чтения умной книжки. Заметьте, что для Вашего удобства необходимо вывести S_N по модулю M .

Входные данные

Входной файл INPUT.TXT содержит число N – возраст планеты Марс и число M - модуль ($0 < N, M \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT выведите ожидаемое число марсиан после N лет жизни планеты по модулю M .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 1000	0
2	2 1000	2
2	8 427	328

```
/*
    Задача: 705. Оценка

    Решение: динамическое программирование, рекурсия,  $O(\log(n))$ 

    Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>

struct Record {
    int x, s0, s1, s2;
};
```

```

Record fast(int x, int n, int mod) {
    if (n == 1) {
        return Record{x % mod, 1 % mod, 0, 0};
    } else if (n % 2 == 1) {
        auto prev = fast(x, n - 1, mod);
        n /= 2;
        Record curr;
        curr.s2 = int((prev.s2 + 4LL * n * n % mod * prev.x) % mod);
        curr.s1 = int((prev.s1 + 2LL * n * prev.x) % mod);
        curr.s0 = int((prev.x + prev.s0) % mod);
        curr.x = int(1LL * prev.x * x % mod);
        return curr;
    } else {
        auto prev = fast(x, n / 2, mod);
        n /= 2;
        Record curr;
        curr.x = int(1LL * prev.x * prev.x % mod);
        curr.s0 = int((prev.s0 + 1LL * prev.s0 * prev.x) % mod);
        curr.s1 = int((prev.s1 + (prev.s1 + 1LL * n * prev.s0) % mod * prev.x) % mod);
        curr.s2 = int((prev.s2 + (prev.s2 + 2LL * n * prev.s1 + 1LL * n * n % mod * prev.s0
        return curr;
    }
}

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int n, mod;
    while (std::cin >> n >> mod) { std::cout << fast(2, n, mod).s2 << std::endl; }
    return 0;
}

```

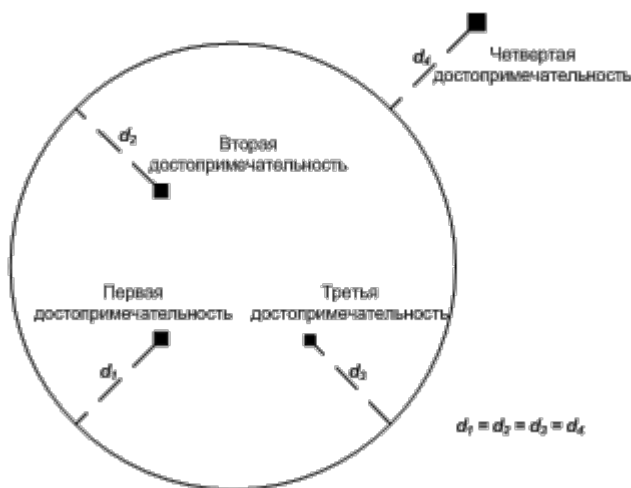

ЗАДАЧА №714

Кольцевая автодорога

(Время: 1 сек. Память: 16 Мб Сложность: 65%)

К 2110 году город Флэтбург, являясь одним из крупнейших городов мира, не имеет обходной автомагистрали, что является существенным препятствием для его развития как крупнейшего транспортного центра мирового значения. В связи с этим еще в 2065 году при разработке Генерального плана развития Флэтбурга была определена необходимость строительства кольцевой автомобильной дороги.

В Генеральном плане также были обозначены требования к этой дороге. Она должна соответствовать статусу кольцевой – иметь форму окружности. Кроме этого, четыре крупные достопримечательности Флэтбурга должны быть в одинаковой транспортной доступности от дороги. Это предполагается обеспечить тем, что они будут находиться на равном расстоянии от нее. Расстоянием от точки расположения достопримечательности до дороги называется наименьшее из расстояний от этой точки до некоторой точки, принадлежащей окружности автодороги.



Дирекция по строительству города Флэтбурга, ответственная за постройку кольцевой автодороги, решила привлечь передовых программистов для выбора оптимального плана постройки дороги.

Требуется написать программу, которая вычислит число возможных планов постройки кольцевой автомобильной дороги с соблюдением указанных требований и найдет такой план, для которого длина дороги будет минимальной.

Входные данные

Входной файл INPUT.TXT содержит четыре строки. Каждая из них содержит по два целых числа: x_i и y_i – координаты места расположения достопримечательности. Первая строка описывает первую достопримечательность, вторая – вторую, третья – третью, четвертая – четвертую. Никакие две достопримечательности не находятся в одной точке. Все числа во входном файле не превосходят 100 по абсолютной величине.

Выходные данные

В первой строке выходного файла OUTPUT.TXT требуется вывести число возможных планов постройки кольцевой автомобильной дороги. Если таких планов бесконечно много, необходимо вывести в первой строке выходного файла слово Infinity.

На второй строке требуется вывести координаты центра дороги минимальной длины и ее радиус. Если существует несколько разных способов построения дороги минимальной длины, необходимо вывести координаты центра и радиус любой из них. Входные данные таковы, что существует хотя бы один вариант дороги. Координаты центра и радиус дороги должны быть выведены с точностью не хуже 10^{-5} .

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	0 0 0 1 1 0 2 2	7 1.5 0.5 1.14412281
2	0 0 0 1 1 0 1 1	Infinity 0.5 0.5 0.0

```
/*
    Задача: 714. Кольцевая автодорога

    Решение: тернарный поиск, геометрия, окружность, срединный перпендикуляр,  $O(\log(\text{MAX}))$ 

    Автор: Дмитрий Козырев, https://github.com/dmkz , dmkozyrev@rambler.ru
*/
```

```
#include <stdio.h>
#include <cmath>
#include <algorithm>
#include <vector>
#include <set>
#include <tuple>
#include <iostream>
#include <iomanip>
#include <cassert>

typedef long double Real;

const Real PI = std::acos(Real(-1.0));

const Real EPS = 1e-14;

bool equal(Real a, Real b) {
    return std::abs(a-b) <= std::max(Real(1), std::abs(a) + std::abs(b)) * EPS;
}

struct Vector {
    Real x, y;

    Vector(Real x_ = 0, Real y_ = 0) : x(x_), y(y_) { }

    Vector operator-() const { return Vector(-x, -y); }
    Vector operator+(const Vector& p) const { return Vector(x+p.x, y+p.y); }
    Vector operator-(const Vector& p) const { return Vector(x-p.x, y-p.y); }
    Vector operator*(const Real val) const { return Vector(x*val, y*val); }
    Vector operator/(const Real val) const { return Vector(x/val, y/val); }
    Real scal(const Vector& b) const { return x * b.x + y * b.y; }
    Real cross(const Vector& b) const { return x * b.y - y * b.x; }
    Real norm() const { return std::sqrt(scal(*this)); }

    Vector rotate(const Real angle) const {
```

```

        return Vector(
            x * std::cos(angle) - y * std::sin(angle),
            x * std::sin(angle) + y * std::cos(angle)
        );
    }

    static Vector read() { int x, y; scanf("%d %d", &x, &y); return Vector(x,y); }
};

typedef Vector Point;

struct Line {
    // p' = p + v * t
    Point p, v;

    Line(const Point& a, const Point& b) : p(a), v(b-a) { v = v / v.norm(); }

    Point get_point(Real t) const { return p + v * t; }

    bool contains(const Point& p_) const {
        return equal((p_ - p).cross(v), 0);
    }

    int intersect(const Line& L, Point& p_) const {
        auto q = v.cross(L.v);
        if (equal(q, 0)) {
            return contains(L.p) ? 2 : 0;
        }
        p_ = get_point((L.p - p).cross(L.v) / q);
        return 1;
    }
};

Line bisector(const Point& a, const Point& b) {
    return Line((a+b)/2, (a+b)/2+(b-a).rotate(PI/2));
}

struct Circle {
    Point c; Real r;

    Circle(const Point& c_, Real r_) : c(c_), r(r_) { }
};

bool operator<(const Circle& a, const Circle& b) {
    bool eq_r = equal(a.r, b.r);
    bool eq_x = equal(a.c.x, b.c.x);
    bool eq_y = equal(a.c.y, b.c.y);
    return (!eq_r && a.r < b.r) || (eq_r && ((!eq_x && a.c.x < b.c.x) || (eq_x && (!eq_y &&
}

int main() {
    std::vector<Point> pt;
    for (int i = 0; i < 4; ++i) {
        pt.push_back(Point::read());
    }

    std::vector<Line> bisectors;
    for (int i = 0; i < 4; ++i) {
        for (int j = i+1; j < 4; ++j) {
            bisectors.push_back(bisector(pt[i], pt[j]));
        }
    }

    bool infinity = false;
    std::set<Circle> set;

    for (int i = 0; i < (int)bisectors.size(); ++i) {
        for (int j = i+1; j < (int)bisectors.size(); ++j) {

```

```

const auto& L1 = bisectors[i];
const auto& L2 = bisectors[j];
Point p;
int code = L1.intersect(L2, p);

if (code == 0) {
    continue;
} else if (code == 2) {
    // using ternary search by optimal position on bisector:
    auto f = [&](const Real t) {
        Real dist = 0;
        for (int k = 0; k < 4; ++k) {
            dist += (L1.get_point(t) - pt[k]).norm();
        }
        return dist / 4;
    };
    Real a = -1e9, b = 1e9;
    while (b - a >= EPS / 2) {
        Real x1 = a + (b-a) / 3;
        Real x2 = b - (b-a) / 3;
        if (f(x1) < f(x2)) {
            b = x2;
        } else {
            a = x1;
        }
    }
    p = L1.get_point((a+b)/2);
}

auto f = [&](const Real value) {
    Real max = -1e9, min = +1e9;
    for (int k = 0; k < 4; ++k) {
        Real dist = std::abs(value - (p-pt[k]).norm());
        max = std::max(max, dist);
        min = std::min(min, dist);
    }
    return max-min;
};
// Ternary search by optimal radius:
Real low = 1e9, high = -1e9;
for (int k = 0; k < 4; ++k) {
    low = std::min(low, (p-pt[k]).norm());
    high = std::max(high, (p-pt[k]).norm());
}
while (high - low >= EPS / 2) {
    Real x1 = low + (high - low) / 3;
    Real x2 = high - (high - low) / 3;
    if (f(x1) < f(x2)) {
        high = x2;
    } else {
        low = x1;
    }
}
Real radius = (equal(f(0), 0)) ? (infinity = true, 0) : (low+high) / 2;
set.insert(Circle(p, radius));
}

if (infinity) {
    printf("Infinity\n");
} else {
    printf("%d\n", (int)set.size());
}
assert(set.size() >= 1u);
auto ans = *set.begin();
std::cout << std::fixed << std::setprecision(5) << ans.c.x << ' ' << ans.c.y << ' ' <<
return 0;
}

```

ЗАДАЧА №747

Декомпозиция строки

(Время: 3 сек. Память: 16 Мб Сложность: 65%)

Для строки T и целого числа n определим n -ю степень строки T^n как конкатенацию n копий строки T . Например, $aab^4 = aabaabaabaab$.

Любая строка S может быть представлена в виде разложения $S = S_1^{d_1} S_2^{d_2} \dots S_k^{d_k}$. Вообще говоря, такое разложение может быть не единственным. Весом разложения строки S в указанном виде назовем сумму $|S_1| + |S_2| + \dots + |S_k|$, где $|Z|$ означает длину строки Z .

По заданной строке S найдите ее разложение с минимальным весом.

Входные данные

Входной файл INPUT.TXT содержит строку S . S состоит из заглавных английских букв и имеет длину не более 5000.

Выходные данные

Первая строка выходного файла OUTPUT.TXT должна содержать w – минимальный возможный вес разложения строки S . Пусть k – число элементов в таком разложении. Тогда следующие k строк должны содержать элементы разложения: строку S_i и степень d_i , разделенные ровно одним пробелом.

Если существует несколько оптимальных решений, выведите любое из них.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	АВАВАААВАВА	5 АВ 2 А 3 ВА 2

```
/*
    Задача: 747. Декомпозиция строки

    Решение: строки, хеширование, полиномиальный хеш, динамическое программирование,  $O(n^2)$ 

    Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define isz(x) (int)(x).size()
typedef unsigned long long ull;

int gen_base(int before, int after) {
    auto seed = std::chrono::high_resolution_clock::now().time_since_epoch().count();
    std::mt19937 gen(seed ^ ull(new ull));
    std::uniform_int_distribution<int> dist(before+2, after-1);
    int base = dist(gen);
```

```

    return base % 2 == 0 ? base - 1 : base;
}

struct PolyHash {
    // ----- Static variables -----
    static const ull mod = (ull(1) << 61) - 1; // prime mod of hashing
    static int base; // odd base of hashing
    static std::vector<ull> pow; // powers of base modulo mod;

    // ----- Static functions -----
    static inline ull add(ull a, ull b) {
        // Calculate (a + b) % mod, 0 <= a < mod, 0 <= b < mod
        return (a += b) < mod ? a : a - mod;
    }

    static inline ull sub(ull a, ull b) {
        // Calculate (a - b) % mod, 0 <= a < mod, 0 <= b < mod
        return (a -= b) < mod ? a : a + mod;
    }

    static inline ull mul(ull a, ull b){
        // Calculate (a * b) % mod, 0 <= a < mod, 0 <= b < mod
        ull l1 = (uint32_t)a, h1 = a >> 32, l2 = (uint32_t)b, h2 = b >> 32;
        ull l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
        ull ret = (l & mod) + (l >> 61) + (h << 3) + (m >> 29) + (m << 35 >> 3) + 1;
        ret = (ret & mod) + (ret >> 61);
        ret = (ret & mod) + (ret >> 61);
        return ret-1;
    }

    // ----- Variables of class -----
    std::vector<ull> pref; // polynomial hash on prefix

    // Constructor from string:
    PolyHash(const std::string& s)
        : pref(s.size()+1u, 0)
    {
        // Pre-calculate powers of base:
        while (pow.size() <= s.size()) {
            pow.push_back(mul(pow.back(), base));
        }
        // Calculate polinomial hash on prefix:
        for (int i = 0; i < (int)s.size(); ++i) {
            pref[i+1] = add(mul(pref[i], base), s[i]);
        }
    }

    // Get hash from [pos, pos+len-1] segment of string
    inline ull operator()(const int pos, const int len) const {
        return sub(pref[pos+len], mul(pref[pos], pow[len]));
    }
};

// Init static variables of class PolyHash:
int PolyHash::base((int)1e9+7);
std::vector<ull> PolyHash::pow{1};

struct State {
    int result, len, cnt;
};

bool operator<(const State& a, const State& b) {
    return a.result < b.result;
}

int main() {
    // Generate random base:

```

```

PolyHash::base = gen_base(256, 2e9);
std::ios_base::sync_with_stdio(false); std::cin.tie(0);
std::string s; std::cin >> s;
PolyHash hash(s);
std::vector<State> dp(isz(s)+1, State{INT_MAX,0,0});
dp[isz(s)].result = 0;
for (int i = isz(s)-1; i >= 0; --i) {
    for (int len = 1; i + len - 1 < isz(s); ++len) {
        ull h[20]; int l[20], last = 0;
        h[0] = hash(i, len);
        l[0] = len;
        while (i + l[last] - 1 < isz(s)) {
            #define add PolyHash::add
            #define mul PolyHash::mul
            #define pow PolyHash::pow
            h[last+1] = add(h[last], mul(h[last], pow[l[last]]));
            l[last+1] = 2 * l[last];
            #undef add
            #undef mul
            #undef pow
            last++;
        }
        int p = i + len;
        while (last >= 0) {
            if (p + l[last] - 1 < isz(s) && hash(p, l[last]) == h[last]) {
                p += l[last];
            }
            --last;
        }
        assert((p-i) % len == 0);
        for (int cnt = 1; cnt <= (p - i) / len; ++cnt) {
            State res{dp[i + cnt * len].result + len, len, cnt};
            dp[i] = std::min(dp[i], res);
        }
    }
}
int pos = 0;
std::cout << dp[pos].result << "\n";
while (pos < isz(s)) {
    std::cout << s.substr(pos, dp[pos].len) << " " << dp[pos].cnt << "\n";
    pos += dp[pos].len * dp[pos].cnt;
}
return 0;
}

```

ЗАДАЧА №815

Обходчик лабиринтов

(Время: 2 сек. Память: 16 Мб Сложность: 65%)

В скором времени на телеэкраны одной страны выйдет новое шоу «Двое в лабиринте». Его сюжет будет состоять в том, что два участника будут помещены в лабиринт. Их целью является поиск выхода из данного лабиринта. Первый, кто найдет выход, получит крупный денежный приз.

Однако, прежде чем шоу выйдет на экраны, лабиринты должны быть сертифицированы Государственным Бюро по Сертификации Лабиринтов. В своей работе бюро использует специальные машины, называемые обходчиками лабиринтов.

Поскольку в силу специфики работы этих машин для каждого лабиринта приходится строить нового обходчика, Вам поручено провести компьютерное моделирование обхода лабиринта обходчиком.

Лабиринт состоит из n комнат, соединенных m коридорами. На концах коридора имеются две двери, одна из которых открывается только из коридора, в вторая только из комнаты, из которой коридор выходит, таким образом, движение по коридору разрешено только в одну сторону. Кроме этого, каждый из коридоров покрашен в один из k цветов (это сделано для того, чтобы немного облегчить участникам нахождение выхода из лабиринта). Цвет коридора указан на соответствующей ему двери в комнате, из которой он выходит. При этом из комнаты могут выходить несколько коридоров одного цвета.

Обходчик лабиринтов работает по программе, которая состоит из L инструкций. Каждая инструкция – это номер цвета (число от 1 до k). Обход лабиринта начинается в комнате номер s и совершается следующим образом: обходчик поочередно считывает инструкции и на каждом шаге выбирает один из коридоров, покрашенных в цвет, указанный в этой инструкции. Если такого коридора не находится, то обходчик «зависает».

Так как на каждом шаге у обходчика может быть не один вариант выбора коридора, то комната, в которой он окажется после выполнения программы может определяться неоднозначно.

Ваша задача состоит в том, чтобы по описанию лабиринта и программе для обходчика определить, в каких комнатах обходчик может оказаться после выполнения соответствующей программы.

Входные данные

Первая строка входного файла INPUT.TXT содержит три целых числа: n, m, k ($1 \leq n, k \leq 1000, 0 \leq m \leq 10000$). Далее идут m строк, описывающих коридоры. Описание каждого коридора состоит из трех целых чисел: u, v, c ($1 \leq u, v \leq n, 1 \leq c \leq k$). Их значения таковы: u - номер комнаты, из которой выходит коридор, v - номер комнаты, в которую ведет коридор, c цвет этого коридора. Коридор может вести из комнаты в саму себя, между двумя комнатами может существовать несколько коридоров (более того, несколько коридоров одного цвета).

$(m+2)$ -ая строка входного файла содержит целое число L ($1 \leq L \leq 1000$). $(m+3)$ -ая строка содержит L целых чисел от 1 до k – программы для обходчика лабиринта.

Последняя строка входного файла содержит целое число s ($1 \leq s \leq n$).

Выходные данные

В выходной файл OUTPUT.TXT следует вывести слово Hangs в случае, если обходчик «зависает» независимо от того, какие коридоры он выбирает при существовании нескольких коридоров одного

цвета.

Иначе, выведите на первой строке выходного файла слово ОК, во второй – количество г комнат, в которых обходчик может оказаться после выполнения программы. В третьей строке выходного файла в этом случае выведите номера этих комнат в возрастающем порядке.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 6 2 1 2 1 1 2 2 1 3 1 1 3 2 3 4 2 3 3 2 2 1 2 1	ОК 2 3 4
2	4 6 2 1 2 1 1 2 2 1 3 1 1 3 2 3 4 1 3 3 1 2 1 2 1	Hangs

```
/*
    Задача: 815. Обходчик лабиринтов

    Решение: графы, поиск в ширину, динамическое программирование,  $O(L * (N + M))$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define isz(x) (int)(x).size()
#define all(x) (x).begin(), (x).end()
typedef std::vector<int> vi;
int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int n, nE, k;
    while (std::cin >> n >> nE >> k) {
        vi begin(nE), end(nE), color(nE);
        for (int i = 0; i < nE; ++i) {
            std::cin >> begin[i] >> end[i] >> color[i];
        }
        int nOp; std::cin >> nOp;
        vi op(nOp);
        for (auto &it : op) { std::cin >> it; }
        int s; std::cin >> s;
        vi curr(1+n), next(1+n);
        curr[s] = 1;
        bool ok = true;
        for (auto c : op) {
            bool was = false;
            for (int i = 0; i < nE; ++i) {
```

```

        if (color[i] == c && curr[begin[i]]) {
            next[end[i]] = 1;
            was = true;
        }
    }
    if (!was) {
        ok = false;
        break;
    }
    curr = next;
    std::fill(all(next), 0);
}
if (!ok) { std::cout << "Hangs\n"; continue; }
vi answ;
for (int i = 1; i <= n; ++i) {
    if (curr[i]) { answ.push_back(i); }
}
std::cout << "OK\n";
std::cout << isz(answ) << "\n";
for (auto it : answ) { std::cout << it << ' '; }
std::cout << std::endl;
}
return 0;
}

```

ЗАДАЧА №902

Ленточка - 2

(Время: 1 сек. Память: 16 Мб Сложность: 65%)

Расположенную вертикально прямоугольную бумажную ленточку с закрепленным нижним концом стали складывать следующим образом:

- на первом шаге ее согнули пополам так, что верхняя половина легла на нижнюю либо спереди (Р - сгибание) либо сзади (Z - сгибание),
- на последующих $n-1$ шагах выполнили аналогичное действие с получающейся на предыдущем шаге согнутой ленточкой, как с единым целым.

Затем ленточку развернули, приведя ее в исходное состояние. На ней остались сгибы - ребра от перегибов, причем некоторые из ребер оказались направленными выпуклостью к нам (К - ребра), а некоторые - от нас (О - ребра). Ребра пронумеровали сверху вниз числами от 1 до $2^n - 1$.

Требуется написать программу, которая по заданной строке символов из прописных букв "О" и "К", где нахождение на i -ом месте символа "О" или "К" определяет тип ребра на расправленной полоске, находит строку из прописных "Р" и "Z", определяющих последовательность типов сгибаний, посредством которых получена ленточка с этой последовательностью ребер.

Входные данные

В первой строке входного файла INPUT.TXT записано число n – количество сгибаний (n не более 20), во второй строке - строка из $2^n - 1$ символов "О" или "К", определяющих типы ребер на расправленной ленточке.

Выходные данные

В выходной файл OUTPUT.TXT выведите строку из n символов "Р" и "Z", задающую последовательность сгибаний. Если такой последовательности сгибаний не существует, то вывести в файл "NO".

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 OOK	PZ
2	2 OOO	NO

```
/*
    Задача: 902. Ленточка-2

    Решение: динамическое программирование, рекурсия, строки, O(n)

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
```

```

#define all(x) (x).begin(),(x).end()
#define isz(x) (int)(x).size()
std::string solve(std::string& s) {
    if (isz(s) == 1) { return s[0] == 'O' ? "P" : "Z"; }
    for (int i = 0, j = isz(s)-1; i < j; i++,--j) {
        if (s[i] != char('O' + 'K' - s[j])) {
            return "NO";
        }
    }
    int n = isz(s);
    assert(n % 2 == 1);
    for (int i = 0; i < n / 2; ++i) { s.pop_back(); }
    auto back = s.back(); s.pop_back();
    auto ret = solve(s);
    if (ret == "NO") { return "NO"; }
    else { return (back == 'O' ? "P" : "Z") + ret; }
}
int main() {
    int n; std::string s;
    while (std::cin >> n >> s) { std::reverse(all(s)); std::cout << solve(s) << "\n"; }
    return 0;
}

```

ЗАДАЧА №974

Странный ним

(Время: 1 сек. Память: 16 Мб Сложность: 65%)

Алиса и Боб играют в игру, которую они называют «странный ним». На столе расположены три кучки камней, содержащие a , b и c камней, соответственно. Алиса ходит первой.

Тот игрок, который делает ход, выбирает одну из кучек и берет из нее несколько камней. При этом должно выполняться следующее условие: после хода каждого игрока на столе не должно быть двух кучек, содержащих равное количество камней. Забирать все камни из кучки разрешается. Тот, кто берет последний камень, выигрывает.

Пусть, например, кучки содержат 1, 3 и 5 камней, тогда разрешены следующие ходы:

- взять 1 камень из первой кучки;
- взять 1 камень из второй кучки;
- взять 3 камня из второй кучки;
- взять 1 камень из третьей кучки;
- взять 3 камня из третьей кучки;
- взять 5 камней из третьей кучки.

По заданным a , b и c , определите, кто выиграет, если оба игрока играют оптимально.

Входные данные

Входной файл INPUT.TXT содержит несколько тестовых примеров (не более 1000). Каждый тестовый пример состоит из трех целых чисел a , b и c , расположенных на одной строке ($1 \leq a, b, c \leq 10^9$, $a \neq b$, $a \neq c$, $b \neq c$). Последняя строка входного файла содержит три нуля, ее обрабатывать не требуется.

Выходные данные

В выходной файл OUTPUT.TXT для каждого тестового примера выведите, кто выиграет в игре, если оба игрока играют оптимально.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	1 2 3 1 3 5 0 0 0	Alice wins the game. Bob wins the game.

```
/*
    Задача: 974. Странный ним

    Решение: теория игр, ним, исследование, O(t)

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/
#include <iostream>
int main() {
```

```
for (int a, b, c; std::cin >> a >> b >> c, a; ) {  
    printf("%s wins the game.\n", (a+1)^(b+1)^(c+1) ? "Alice" : "Bob");  
}  
return 0;  
}
```

ЗАДАЧА №1388

Школы

(Время: 1 сек. Память: 16 Мб Сложность: 65%)

С целью подготовки к проведению олимпиады по информатике мэр решил обеспечить надежным электроснабжением все школы города. Для этого необходимо провести линию электропередач от альтернативного источника электроэнергии «Майбуття» к одной из школ города (к какой неважно), а также соединить линиями электропередач некоторые школы между собой.

Считается, что школа имеет надежное электроснабжение, если она напрямую связана с источником «Майбуття», либо с одной из тех школ, которые имеют надежное электроснабжение.

Известна стоимость соединения между некоторыми парами школ. Мэр города решил выбрать одну из двух наиболее экономичных схем электроснабжения (стоимость схемы равняется сумме стоимостей соединений пар школ).

Напишите программу, которая вычисляет стоимость двух наиболее экономных схем альтернативного электроснабжения школ.

Входные данные

В первой строке входного файла INPUT.TXT два натуральных числа N и M – количество школ в городе и количество возможных соединений между ними соответственно ($3 \leq N \leq 100$, $N \leq M \leq N \cdot (N-1)/2$).

В каждой из последующих M строк находятся по три целых числа: A_i , B_i , C_i , где C_i – стоимость прокладки линии электроснабжения от школы A_i до школы B_i ($1 \leq C_i \leq 300$; $i=1, 2, \dots, N$).

Гарантируется, что существует хотя бы две различные схемы электроснабжения.

Выходные данные

В выходной файл OUTPUT.TXT выведите два натуральных числа S_1 и S_2 – две наименьшие стоимости различных схем электроснабжения ($S_1 \leq S_2$).

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 8 1 3 75 3 4 51 2 4 19 3 2 95 2 5 42 5 4 31 1 2 9 3 5 66	110 121

/*

Решение: DSU, алгоритм Краскала, Sparse Table, dfs, $O(n \log(n))$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru

*/

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>
#include <functional>

struct Edge {
    int u, v, cost;
};

bool operator<(const Edge& a, const Edge& b) {
    if (a.cost < b.cost) return true;
    if (a.cost > b.cost) return false;
    return a.u < b.u || (a.u == b.u && a.v < b.v);
}

struct DSU {
    std::vector<int> parent, size;
    DSU (int n) : parent(1+n,0), size(1+n,1) {
        for (int i = 1; i <= n; ++i) parent[i] = i;
    }
    int get_parent(int a) {
        return parent[a] == a ? a : parent[a] = get_parent(parent[a]);
    }
    void union_sets(int a, int b) {
        a = get_parent(a), b = get_parent(b);
        if (a != b) {
            if (size[a] < size[b]) std::swap(a,b);
            size[a] += size[b];
            parent[b] = a;
        }
    }
};

const int INF = (int)1e9+1;

int main() {
    // Ввод ребер и их сортировка:
    int n, m; scanf("%d %d", &n, &m);
    std::vector<Edge> edges(m); std::vector<bool> used(m);
    for (auto& e : edges) {
        scanf("%d %d %d", &e.u, &e.v, &e.cost);
    }
    std::sort(edges.begin(), edges.end());
    // Построение минимального остова в виде дерева:
    DSU dsu(n);
    std::vector<std::vector<Edge>> next(1+n);
    int min1 = 0;
    for (int i = 0; i < m; ++i) {
        const auto& e = edges[i];
        int pu = dsu.get_parent(e.u), pv = dsu.get_parent(e.v);
        if (pu != pv) {
            next[e.u].push_back(e);
            next[e.v].push_back(e);
            dsu.union_sets(e.u, e.v);
            min1 += e.cost;
            used[i] = true;
        }
    }
    // Построение LCA и Sparse Table на дереве:
    const int PMAX = 10, NMAX = 512;
    static int max[PMAX][NMAX], par[PMAX][NMAX];
    std::vector<int> dep(1+n);
```



```

std::function<void(int,int,int)> visit = [&](int u, int p, int c) {
    max[0][u] = c;
    par[0][u] = p;
    dep[u] = dep[p]+1;
    for (auto& e : next[u]) {
        int v = e.u + e.v - u;
        if (v == p) continue;
        visit(v,u,e.cost);
    }
};
visit(1,0,0);
for (int p = 1; p < PMAX; ++p) {
    for (int i = 1; i <= n; ++i) {
        int j = par[p-1][i];
        par[p][i] = par[p-1][j];
        max[p][i] = std::max(max[p-1][i], max[p-1][j]);
    }
}
// Функция получения ответа: вес максимального ребра на пути от `u` до `v`
std::function<int(int,int)> get_max = [&](int u, int v) {
    if (dep[u] > dep[v]) std::swap(u,v);
    int delta = dep[v] - dep[u];
    int ret = 0;
    for (int p = PMAX-1; p >= 0; --p) {
        if ((delta >> p) & 1) {
            ret = std::max(ret, max[p][v]);
            v = par[p][v];
        }
    }
    for (int p = PMAX-1; p >= 0; --p) {
        int pu = par[p][u], pv = par[p][v];
        if (pu != pv) {
            ret = std::max(ret, max[p][u]);
            ret = std::max(ret, max[p][v]);
            u = pu, v = pv;
        }
    }
    if (u == v) return ret;
    ret = std::max(ret, max[0][u]);
    ret = std::max(ret, max[0][v]);
    return ret;
};
// Находим вес второго минимального остова:
int min2 = INF;
for (int i = 0; i < m; ++i) {
    if (used[i]) continue;
    const auto& e = edges[i];
    min2 = std::min(min2, min1 - get_max(e.u, e.v) + e.cost);
}
printf("%d %d", min1, min2);
return 0;
}

```

ЗАДАЧА №533

Треугольники - 3

(Время: 2 сек. Память: 16 Мб Сложность: 66%)

Петя достаточно давно занимается в математическом кружке, поэтому он уже успел освоить не только правила выполнения простейших операций, но и такое достаточно сложное понятие как симметрия. Для того, чтобы получше изучить симметрию Петя решил начать с наиболее простых геометрических фигур – треугольников. Он скоро понял, что осевой симметрией обладают так называемые равнобедренные треугольники. Поэтому теперь Петя ищет везде такие треугольники.

Напомним, что треугольник называется равнобедренным, если его площадь положительна, и у него есть хотя бы две равные стороны.

Недавно Петя, зайдя в класс, увидел, что на доске нарисовано n точек. Разумеется, он сразу задумался, сколько существует троек из этих точек, которые являются вершинами равнобедренных треугольников.

Требуется написать программу, решающую указанную задачу.

Входные данные

Входной файл INPUT.TXT содержит целое число N ($3 \leq N \leq 1500$). Каждая из последующих строк содержит по два целых числа – x_i и y_i – координаты i -ой точки. Координаты точек не превосходят 10^9 по абсолютной величине. Среди заданных точек нет совпадающих.

Выходные данные

В выходной файл OUTPUT.TXT выведите ответ на задачу.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	3 0 0 2 2 -2 2	1
2	4 0 0 1 1 1 0 0 1	4

```
/*
    Задача: Треугольники - 3

    Решение: сортировка, бинарный поиск, геометрия, отрезки,  $O(n^2 \log(n))$ 

    Автор: Дмитрий Козырев, https://github.com/dmkz , dmkozyrev@rambler.ru
*/

#include <stdio.h>
```

```

#include <algorithm>
#include <vector>

typedef long long ll;

struct Seg {

    int x, y, dx, dy;

    ll norm2;

    Seg(int x1 = 0, int y1 = 0, int x2 = 0, int y2 = 0) : x(x1), y(y1), dx(x2-x1), dy(y2-y1) {
        norm2 = ll(dx)*(dx)+ll(dy)*(dy);
    }

    inline Seg inv() const {
        Seg ret = *this;
        ret.dx = -ret.dx;
        ret.dy = -ret.dy;
        return ret;
    }
};

inline bool operator<(const Seg& a, const Seg& b) {
    if (a.norm2 < b.norm2) return true;
    if (a.norm2 > b.norm2) return false;
    return a.x < b.x || (a.x == b.x && (a.y < b.y || (a.y == b.y && (a.dx < b.dx || (a.dx =
}

int main() {
    int n; scanf("%d", &n);
    std::vector<int> x(n), y(n);
    for (int i = 0; i < n; ++i) {
        scanf("%d %d", &x[i], &y[i]);
    }
    int ans = 0;
    for (int i = 0; i < n; ++i) {
        // Проводим отрезки из i-й точки до всех остальных и сортируем их:
        std::vector<Seg> segs;
        segs.reserve(n);
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            segs.push_back(Seg(x[i],y[i],x[j],y[j]));
        }
        std::sort(segs.begin(), segs.end());
        // Для каждого отрезка ищем количество вариантов комбинирования с другими отрезками
        // Не забываем вычесть количество вариантов получения вырожденных треугольников
        for (int j = 0; j < (int)segs.size(); ++j) {
            auto pair = std::equal_range(segs.begin()+j+1, segs.end(), segs[j], [](const Se
                return a.norm2 < b.norm2;
            ));
            ans += int(pair.second - pair.first) - (int)std::binary_search(segs.begin()+j+1
        }
    }
    printf("%d", ans);
    return 0;
}

```

ЗАДАЧА №622

Прямоугольное деление

(Время: 1 сек. Память: 16 Мб Сложность: 66%)

Дано N прямоугольников со сторонами, параллельными осям координат. Требуется определить, на сколько частей эти прямоугольники разбивают плоскость (внутри частей не должно быть границ прямоугольников).

Входные данные

В первой строке входного файла INPUT.TXT содержится число прямоугольников N ($1 \leq N \leq 100$). Далее идут N строк, содержащих по 4 числа x_1, y_1, x_2, y_2 - координаты двух противоположных углов прямоугольника. Координаты представляют собой целые числа и по абсолютной величине не превосходят 10 000.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число - количество частей, на которые разбивается плоскость.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 10 20 50 30 40 10 50 25 40 25 80 30	6

/*
Сведем задачу к задаче на графы:
- Читаем все прямоугольники, сохраняя куда-нибудь их координаты x и y
- Сортируем координаты, удаляем повторы
- Каждой координате сопоставляем ее удвоенный индекс в полученном массиве
- Таким образом, мы свели задачу к задаче на прямоугольном поле со стенками и комнатами
- Считаем количество областей смежности

Пример для первого теста:

```
3
10 20 50 30
40 10 50 25
40 25 80 30
```

Полученное после всех преобразований поле:

```
#####
#000000000#
#000##000#
#000#1#000#
#0####000#
#0#2#3#000#
#0#2####0#
#0#2#4#5#0#
#0#####0#
#000000000#
#####
```

Ответ: 6

```
*/

#include <stdio.h>
#include <vector>
#include <queue>
#include <algorithm>
#include <cassert>
#include <functional>

struct Rect {
    int x_min, y_min, x_max, y_max;

    Rect(int x1, int y1, int x2, int y2)
        : x_min(std::min(x1, x2))
        , y_min(std::min(y1, y2))
        , x_max(std::max(x1, x2))
        , y_max(std::max(y1, y2))
    { }

    static Rect read() {
        int x1, y1, x2, y2;
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
        return Rect(x1, y1, x2, y2);
    }
};

struct Point {
    int row, col;
};

int main() {
    int nRect;
    scanf("%d", &nRect);

    std::vector<Rect> rects;
    for (int i = 0; i < nRect; ++i) {
        rects.push_back(Rect::read());
    }

    std::vector<int> x{-100001, 100001}, y{-100001, 100001};
    for (auto r : rects) {
        x.push_back(r.x_min);
        x.push_back(r.x_max);
        y.push_back(r.y_min);
        y.push_back(r.y_max);
    }

    std::sort(x.begin(), x.end());
    x.erase(std::unique(x.begin(), x.end()), x.end());

    std::sort(y.begin(), y.end());
    y.erase(std::unique(y.begin(), y.end()), y.end());

    const int WALL = -2;
    const int EMPTY = -1;

    std::vector<std::vector<int>> map(2*y.size()-1, std::vector<int>(2*x.size()-1, EMPTY));

    for (int i = 0; i < (int)map.size(); ++i) {
        map[i].back() = map[i].front() = WALL;
    }

    for (int j = 0; j < (int)map[0].size(); ++j) {
        map.front()[j] = map.back()[j] = WALL;
    }

    for (const auto& it : rects) {
```

```

    int l = 2 * int(std::lower_bound(x.begin(), x.end(), it.x_min) - x.begin());
    int r = 2 * int(std::lower_bound(x.begin(), x.end(), it.x_max) - x.begin());
    int u = 2 * int(std::lower_bound(y.begin(), y.end(), it.y_min) - y.begin());
    int d = 2 * int(std::lower_bound(y.begin(), y.end(), it.y_max) - y.begin());
    for (int row = u; row <= d; ++row) {
        map[row][l] = map[row][r] = WALL;
    }
    for (int col = l; col <= r; ++col) {
        map[d][col] = map[u][col] = WALL;
    }
}

int count = 0;
for (int row = 0; row < (int)map.size(); ++row) {
    for (int col = 0; col < (int)map[0].size(); ++col) {
        if (map[row][col] == EMPTY) {
            std::function<void(int, int, int)> bfs = [&map](int row, int col, int color) {
                std::queue<Point> queue;
                map[row][col] = color;
                queue.push(Point{row, col});
                while (!queue.empty()) {
                    auto curr = queue.front(); queue.pop();
                    for (int dr = -1; dr <= 1; ++dr)
                        for (int dc = -1; dc <= 1; ++dc) {
                            if (dr * dr + dc * dc == 1) {
                                int nr = curr.row + dr;
                                int nc = curr.col + dc;
                                if (map[nr][nc] == EMPTY) {
                                    map[nr][nc] = color;
                                    queue.push(Point{nr, nc});
                                }
                            }
                        }
                }
            };
            bfs(row, col, count++);
        }
    }
}

printf("%d\n", count);

return 0;
}

```

ЗАДАЧА №1188

Дерево с изменением отрезка

(Время: 1 сек. Память: 32 Мб Сложность: 66%)

Требуется реализовать эффективную структуру данных, позволяющую хранить элементы массива $A[1..N]$ и прибавлять ко всем элементам отрезка $[L, R]$ одно и то же число.

Входные данные

Первая строка входного файла INPUT.TXT содержит натуральное число N – размер массива ($N \leq 10^5$). Во второй строке записаны N целых чисел – элементы массива, целые числа, не превосходящие 10^9 по абсолютной величине. Третья строка содержит натуральное число M – количество запросов ($M \leq 30\,000$). Каждая из следующих M строк представляет собой описание запроса. Сначала вводится одна буква, кодирующая вид запроса («g» – извлечь элемент массива, «a» – прибавление значения на отрезке). Следом за «g» идет целое число I – номер извлекаемого элемента. Следом за «a» записаны три числа L, R и X – границы отрезка $[L, R]$ и значение X , на которое их следует увеличить ($1 \leq L \leq R \leq N, 1 \leq I \leq N, |X| \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса на извлечение элемента выведите результат. Все числа следует выводить в одну строку через пробел.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5 2 4 3 5 2 5 g 2 g 5 a 1 3 10 g 2 g 4	4 2 14 5

```
/*
    Задача: 1188. Дерево с изменением отрезка

    Решение: Дерево Фенвика,  $O(n \log(n))$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <iostream>
#include <algorithm>
#include <vector>

typedef long long ll;

struct Fenwick {
    std::vector<ll> data;
```

```

Fenwick(int n = 0) : data(n) { }

void inc(int p, int x) {
    for (int i = p; i < (int)data.size(); i |= i+1) {
        data[i] += x;
    }
}

ll get(int r) const {
    ll ret = 0;
    for (int i = r; i >= 0; i = (i & (i + 1)) - 1) {
        ret += data[i];
    }
    return ret;
}
};

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0); std::cout.tie(0); std::cerr.tie(0);
    int n; std::cin >> n;
    Fenwick fenwick(1+n+1);
    for (int i = 1, v; i <= n; ++i) {
        std::cin >> v;
        fenwick.inc( i, +v);
        fenwick.inc(i+1, -v);
    }
    int q; std::cin >> q;
    while (q--) {
        char t; std::cin >> t;
        if (t == 'g') {
            int r; std::cin >> r;
            std::cout << fenwick.get(r) << " ";
        } else {
            int l, r, x; std::cin >> l >> r >> x;
            fenwick.inc( l,+x);
            fenwick.inc(r+1,-x);
        }
    }
    return 0;
}

```


ЗАДАЧА №1390

Маршрут для гонца

(Время: 1 сек. Память: 16 Мб Сложность: 67%)

В королевстве N городов, пронумерованных от 1 до N . Столица имеет номер 1. Каждый город окружен городской стеной с 4 воротами. Ворота пронумерованы следующим образом: ворота i -го города ($1 \leq i \leq N$) имеют номера $4i-3$, $4i-2$, $4i-1$, $4i$. Через каждые ворота проходит ровно 1 дорога, которая ведет до некоторых ворот другого города (заметьте, что может существовать несколько дорог между двумя городами). По всем дорогам можно двигаться в обоих направлениях. Благодаря системе туннелей и мостов дороги не пересекаются вне городов.

Королевский гонец должен развесить копии Очень важного Королевского Указа на внешней стороне всех ворот каждого города. Гонец может свободно передвигаться от одних ворот к другим в пределах города, но вне города он может двигаться только по дорогам. Гонец выезжает из столицы и должен туда вернуться после выполнения задания.

Может ли гонец выполнить поручение, проходя через каждые ворота только один раз? Выход из города через ворота только для того, чтобы вывесить на их внешней стороне указ, а затем немедленное возвращение в город, считается за один проход через ворота.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число N ($2 \leq N \leq 1000$). Каждая из последующих $2N$ строк описывает одну дорогу и содержит 2 целых числа, разделенных пробелом: номера ворот, соединенных дорогой.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите «Yes», если поручение гонца может быть выполнено, и «No» в противном случае. В случае если это возможно, вторая строка выходного файла должна содержать $4N$ целых чисел: номера ворот в порядке прохождения через них гонцом. Если существует несколько решений, выведите любое из них.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4 1 9 2 10 3 13 4 8 5 11 6 14 7 16 15 12	Yes 3 13 14 6 7 16 15 12 9 1 2 10 11 5 8 4

/*
Задача: 1390. Маршрут для гонца
Решение: графы, Эйлеров обход, поиск в глубину, $O(n)$

Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru
*/

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>

struct Edge {
    int u, v, id;

    Edge(int u_ = 0, int v_ = 0, int id_ = 0)
        : u(u_), v(v_), id(id_) { }
};

int main() {
    // Читаем ребра:
    int n, nAdded = 0; scanf("%d", &n);
    std::vector<Edge> edges;
    for (int i = 0; i < 2*n; ++i) {
        int u, v; scanf("%d %d", &u, &v);
        int s = (u+3) / 4, t = (v+3) / 4;
        assert(s != t);
        ++nAdded;
        int mid = n + nAdded;
        // Добавим новую вершину `mid` между вершинами `s` и `t`
        // Ребро `s` -- `mid` будет иметь идентификатор `u`
        // Ребро `t` -- `mid` будет иметь идентификатор `v`
        edges.push_back(Edge(s, mid, u));
        edges.push_back(Edge(t, mid, v));
    }
    // Создаем списки смежности и считаем валентности вершин:
    std::vector<int> nEdges(1+n+nAdded);
    std::vector<std::vector<int>> next(1+n+nAdded);
    for (int i = 0; i < (int)edges.size(); ++i) {
        const auto& e = edges[i];
        next[e.u].push_back(i); nEdges[e.u]++;
        next[e.v].push_back(i); nEdges[e.v]++;
    }
    // Эйлеров обход графа в глубину:
    std::vector<int> stack{1}, stackEdges, answ;
    while (!stack.empty()) {
        auto curr = stack.back();
        while (true) {
            if (next[curr].empty()) {
                if (!stackEdges.empty()) {
                    answ.push_back(stackEdges.back());
                    stackEdges.pop_back();
                }
                stack.pop_back();
                break;
            }
            int e = next[curr].back();
            if (edges[e].id == 0) {
                next[curr].pop_back();
                continue;
            }
            // Удаление ребра `e` и переход в следующую вершину:
            assert(nEdges[curr] > 0);
            nEdges[curr]--;
            curr = edges[e].u + edges[e].v - curr;
            assert(nEdges[curr] > 0);
            nEdges[curr]--;
            stackEdges.push_back(edges[e].id);
            edges[e].id = 0;
            stack.push_back(curr);
        }
    }
    if ((int)answ.size() != 4 * n) {
```

```
        printf("No\n");
        return 0;
    }
    printf("Yes\n");
    for (auto& it : answ) printf("%d ", it);
    return 0;
}
```

ЗАДАЧА №612

Подстрока

(Время: 1 сек. Память: 32 Мб Сложность: 68%)

Рассмотрим слова, состоящие из букв «А», «В», «а» и «b». Скажем, что два слова эквивалентны, если одно может быть получено из другого с помощью следующих операций:

- удалить в любой позиции подстроку, равную Aa, aA, Bb или bB;
- добавить в любую позицию подстроку, равную Aa, aA, Bb или bB.

Например, слова abAaBBabbA и aAaBabaAbA эквивалентны:

abAaBBabbA → abBBabbA → aBabbA → aAaBabbA → aAaBabaAbA,

а слова abAB и baBA - нет.

Интересно отметить, что для произвольных слов X и Y найдется такое слово Z, эквивалентное X, которое содержит Y в качестве подстроки. Ваша задача - найти кратчайшее такое слово.

Входные данные

Первая строка входного файла INPUT.TXT содержит X. Вторая строка содержит Y. Оба слова непустые и каждое из них имеет длину не больше 2000.

Выходные данные

В выходной файл OUTPUT.TXT выведите одну строку, содержащую минимальное по длине слово, эквивалентное X, содержащее Y в качестве подстроки. Если решений несколько, выведите любое.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	abAaBBabbA AaBaba	aAaBabaAbA

```
/*
    Задача: 612. Подстрока

    Решение: конструктив, динамическое программирование, двумерное дп, O(n^2)

    Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>
#define isz(x) (int)(x).size()
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
bool can_remove(char fi, char se) {
    return std::tolower(fi) == std::tolower(se) && fi != se;
}
std::string split(std::string s) {
    std::string ret;
```

```

    for (auto it : s) {
        if (ret.empty()) {
            ret.push_back(it);
        } else if (can_remove(ret.back(), it)) {
            ret.pop_back();
        } else {
            ret.push_back(it);
        }
    }
    return ret;
}

struct Record {
    int len, i, j;
};

bool operator<(const Record& a, const Record& b) {
    return a.len < b.len || (a.len == b.len && (a.i < b.i || (a.i == b.i && a.j < b.j)));
}

bool operator>(const Record& a, const Record& b) {
    return b < a;
}

Record max_substr(std::string s, std::string t) {
    s = "$" + s;
    t = "^" + t;
    vvi mxTill(isz(s), vi(isz(t)));
    Record best{0,1,1};
    for (int i = 1; i < isz(s); ++i) {
        for (int j = 1; j < isz(t); ++j) {
            if (s[i] == t[j]) { mxTill[i][j] = mxTill[i-1][j-1] + 1; }
            best = std::max(best, Record{mxTill[i][j], i-mxTill[i][j]+1, j-mxTill[i][j]+1})
        }
    }
    return best;
}

char opposite(char c) {
    return char('A' <= c && c <= 'Z' ? c - 'A' + 'a' : c - 'a' + 'A');
}

std::string solve(std::string s, std::string t) {
    s = split(s);
    auto t1 = split(t);
    auto best = max_substr(s,t1);
    best.i--, best.j--;
    for (int i = isz(t1)-1; i >= best.j + best.len; --i) {
        t.push_back(opposite(t1[i]));
    }
    for (int i = 0; i < best.j; ++i) {
        t.insert(t.begin(), opposite(t1[i]));
    }
    return s.substr(0, best.i) + t + s.substr(best.i+best.len);
}

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0);
    std::string s, t;
    std::cin >> s >> t;
    std::cout << solve(s, t) << std::endl;
    return 0;
}

```

ЗАДАЧА №632

Отрезки

(Время: 1 сек. Память: 16 Мб Сложность: 68%)

Дан прямоугольник на координатной плоскости с левым нижним углом в точке (0, 0), а правым верхним - в точке (W, H) и отрезки, параллельные осям координат. Отрезки задаются координатами своих концов. Эти отрезки разрезают прямоугольник на несколько частей (возможно, одну). Требуется определить их площади. Отрезки могут пересекаться, накладываться и вырождаться в точку. Все координаты - целые числа по модулю не превосходящие 10000.

Входные данные

В первой строке входного файла INPUT.TXT указываются числа W и H ($1 \leq W, H \leq 10000$). Во второй строке N ($0 \leq N \leq 50$) - количество отрезков. Далее в N строках через пробел указываются числа A_i, B_i, C_i, D_i - координаты концов i-го отрезка: (A_i, B_i) и (C_i, D_i) .

Выходные данные

Выходной файл OUTPUT.TXT должен содержать последовательность положительных чисел — площади областей, записанные в порядке не возрастания.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	3 3	5
	3	2
	1 3 1 1	1
	1 2 4 2	1
	2 0 2 8	

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>
#include <functional>

struct Segment {
    int x1, y1, x2, y2;

    static Segment read() {
        int x1, y1, x2, y2;
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
        if (x2 < x1 || (x2 == x1 && y2 < y1)) {
            std::swap(x1, x2);
            std::swap(y1, y2);
        }
        return Segment{x1, y1, x2, y2};
    }
};

int main() {
    int W, H, nSeg;
    scanf("%d %d %d", &W, &H, &nSeg);
```

```

std::vector<int> x{0, W}, y{0, H}; // вектора сжатых x и y координат
std::vector<Segment> segments{
    Segment{0, 0, 0, H},
    Segment{W, 0, W, H},
    Segment{0, 0, W, 0},
    Segment{0, H, W, H}
};
for (int i = 0; i < nSeg; ++i) { // чтение отрезков
    auto s = Segment::read();
    if (s.x1 >= W || s.x2 <= 0 || s.y1 >= H || s.y2 <= 0) {
        continue;
    }
    s.x2 = std::min(W, s.x2);
    s.x1 = std::max(0, s.x1);
    s.y2 = std::min(H, s.y2);
    s.y1 = std::max(0, s.y1);
    x.push_back(s.x1);
    x.push_back(s.x2);
    y.push_back(s.y1);
    y.push_back(s.y2);
    segments.push_back(s);
}

std::sort(x.begin(), x.end()); // сжатие координат x
x.erase(std::unique(x.begin(), x.end()), x.end());

std::sort(y.begin(), y.end()); // сжатие координаты y
y.erase(std::unique(y.begin(), y.end()), y.end());

// Отмечаем сегменты как стенки на двумерном поле:
const int WALL = -2, NOT_VISITED = -1;
const int nRows = (int)y.size()*2-1, nCols = (int)x.size()*2-1;
std::vector<std::vector<int>> part(nRows, std::vector<int>(nCols, NOT_VISITED));
std::vector<std::vector<int>> square(nRows, std::vector<int>(nCols, -1));
for (auto& s : segments) {
    s.x1 = int(std::lower_bound(x.begin(), x.end(), s.x1) - x.begin()) * 2;
    s.x2 = int(std::lower_bound(x.begin(), x.end(), s.x2) - x.begin()) * 2;
    s.y1 = int(std::lower_bound(y.begin(), y.end(), s.y1) - y.begin()) * 2;
    s.y2 = int(std::lower_bound(y.begin(), y.end(), s.y2) - y.begin()) * 2;
    assert(s.x1 == s.x2 || s.y1 == s.y2);
    if (s.y1 == s.y2) {
        for (int i = s.x1; i <= s.x2; ++i) {
            part[s.y1][i] = WALL;
            square[s.y1][i] = 0;
        }
    } else if (s.x1 == s.x2) {
        for (int i = s.y1; i <= s.y2; ++i) {
            part[i][s.x1] = WALL;
            square[i][s.x1] = 0;
        }
    } else {
        assert(false);
    }
}

// Вычисляем площадь каждого сжатого сегмента:
for (int r = 0; r < nRows; ++r) {
    for (int c = 0; c < nCols; ++c) {
        if (r % 2 == 0 || c % 2 == 0) {
            square[r][c] = 0;
        } else {
            assert(r % 2 == 1 && c % 2 == 1);
            square[r][c] = (y[r/2+1]-y[r/2])*(x[c/2+1]-x[c/2]);
        }
    }
}

// Заливка - выделяем компоненты связности:
int nParts = 0;
for (int r = 0; r < nRows; ++r) {
    for (int c = 0; c < nCols; ++c) {

```

```

        if (part[r][c] == NOT_VISITED) {
            std::function<void(int, int)> visit = [&](const int row, const int col) {
                part[row][col] = nParts;
                for (int dr = -1; dr <= 1; ++dr) {
                    for (int dc = -1; dc <= 1; ++dc) {
                        if (dr * dr + dc * dc == 1 && part[row+dr][col+dc] == NOT_VISIT
                            part[row+dr][col+dc] = nParts;
                            visit(row+dr, col+dc);
                        }
                    }
                }
            };
            visit(r, c);
            ++nParts;
        }
    }
}

// Считаем площадь каждой компоненты связности:
std::vector<int> s(nParts);
for (int r = 0; r < nRows; ++r) {
    for (int c = 0; c < nCols; ++c) {
        if (part[r][c] != WALL) {
            assert(part[r][c] >= 0);
            s[part[r][c]] += square[r][c];
        }
    }
}

std::sort(s.begin(), s.end(), std::greater<int>());
for (auto& it : s) {
    printf("%d\n", it);
}
return 0;
}

```


ЗАДАЧА №1414

Многословие

(Время: 5 сек. Память: 16 Мб Сложность: 68%)

Поликарп приехал в столицу Байтландии, чтобы как следует осмотреть Байтландский Музей. В музее все экспонаты выстроены в ряд и пронумерованы от 1 до N. Он решил провести в столице Q дней. Каждый день он планирует посещать музей, просматривая отрезок экспонатов, начиная с экспоната L_k и заканчивая экспонатом R_k .

Просматривая экспонаты, Поликарп записывает свои впечатления в блокнот. Так как просмотр начинается с утра, то вначале Поликарп не многословен и записывает всего один эпитет про первый просмотренный экспонат. Далее Поликарп чувствует прилив эмоций и записывает всё больше слов об очередном экспонате. Формально, на i -ом просмотренном экспонате Поликарп записывает i^2 эпитетов в описание этого экспоната в свой блокнот.

После приезда домой Поликарп решил структурировать свои записи, но их было так много, что он попросил вас написать программу, считающую количество эпитетов для каждого экспоната. Единственное, что Поликарп запомнил, это какие отрезки он посещал в музее каждый день. Эту информацию он и предоставил вам. Помогите Поликарпу, он в долгу не останется!

Входные данные

В первой строке входного файла OUTPUT.TXT содержатся два целых числа N и Q ($1 \leq N, Q \leq 500\,000$). В следующих Q строках следуют параметры отрезков в виде пары чисел L_k и R_k ($1 \leq L_k, R_k \leq N, 1 \leq k \leq Q$).

Выходные данные

В выходной файл OUTPUT.TXT выведите N строк, в i -ой из которых должно содержаться количество эпитетов, записанных для i -го экспоната.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	6 3	1
	1 3	4
	6 3	25
	9	9
	5 6	5
		5

```
/*
    Задача: 1414. Многословие

    Решение: offline-запросы, прибавление квадрата расстояния на отрезке, сортировка, O(n log n)

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/

#include <iostream>
#include <vector>
```

```

#include <algorithm>
#include <cassert>

#define all(x) (x).begin(), (x).end()

typedef long long ll;

struct Pair {
    int index, value;
    Pair(int index_ = 0, int value_ = 0) : index(index_), value(value_) { }
};

typedef std::vector<Pair> vp;

bool operator<(Pair lhs, Pair rhs) {
    return lhs.index < rhs.index || (lhs.index == rhs.index && lhs.value < rhs.value);
}

void solve(const int n, const std::vector<Pair>& queries, std::vector<ll>& answ, bool rever
ll s2 = 0, s1 = 0, k = 0; int id = 0;
for (int i = 1; i <= n; ++i) {
    while (id < (int)queries.size() && queries[id].index == i) {
        int sign = queries[id].value >= 0 ? +1 : -1;
        int value = std::abs(queries[id].value);
        id++;
        k += sign;
        s2 += sign * (ll)value * value;
        s1 += sign * (ll)value;
    }
    s2 += 2 * s1 + k;
    s1 += k;
    answ[reverseOrder ? n - i + 1 : i] += s2;
}
}

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0); std::cerr.tie(0); std::cout.tie(0);
    int n, q;
    while (std::cin >> n >> q) {
        vp queriesLR, queriesRL;
        for (int i = 0, l, r; i < q; ++i) {
            std::cin >> l >> r;
            if (l <= r) {
                queriesLR.push_back(Pair(l,0));
                queriesLR.push_back(Pair(r+1,-(r-l+1)));
            } else {
                queriesRL.push_back(Pair(n-l+1,0));
                queriesRL.push_back(Pair(n-(r-1)+1,-(l-r+1)));
            }
        }
        std::sort(all(queriesLR));
        std::sort(all(queriesRL));
        std::vector<ll> answ(1+n);
        solve(n, queriesLR, answ, false);
        solve(n, queriesRL, answ, true);
        for (int i = 1; i <= n; ++i) { std::cout << answ[i] << "\n"; }
    }
    return 0;
}

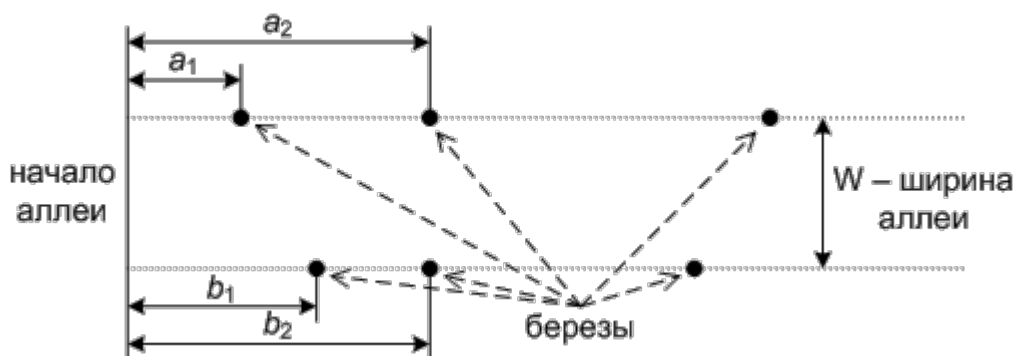
```

ЗАДАЧА №928

Березовая аллея

(Время: 10 сек. Память: 128 Мб Сложность: 70%)

На краю деревни растет старая березовая аллея. Аллея имеет форму прямой полосы шириной W метров. Вдоль левой стороны аллеи растет N берез, а вдоль правой — M берез, при этом i -я береза с левой стороны аллеи находится на расстоянии a_i метров от начала аллеи, а j -я береза с правой стороны — на расстоянии b_j метров от начала аллеи.



Отдыхая в деревне прошедшим летом, один юный информатик обнаружил, что кору берез стали грызть зайцы. Чтобы защитить деревья от зайцев, мальчик решил окружить березы красной лентой (зайцы не любят красный цвет и не станут заходить на огражденную лентой территорию). К сожалению, в его распоряжении оказалась только лента длиной L метров, которую, к тому же, нельзя было разрезать. Единственное, что можно было делать в этом случае — окружить этой лентой как можно больше берез. При этом, чтобы сохранить аллею, необходимо окружить на каждой стороне аллеи хотя бы одну березу.

Требуется написать программу, которая по заданной длине ленты, ширине аллеи и положению берез на ней определяет максимальное количество берез, которое можно окружить этой лентой. Считается, что березы представляются точками, толщиной берез и шириной ленты следует пренебречь.

Входные данные

Первая строка входного файла INPUT.TXT содержит два разделенных пробелом целых числа: длину ленты L и ширину аллеи W ($1 \leq L \leq 2 \times 10^5$, $1 \leq W \leq 10^4$).

Вторая и третья строки описывают березы вдоль левой стороны аллеи. Вторая строка содержит число N — количество берез ($1 \leq N \leq 2000$), а третья строка содержит N различных целых чисел a_1, a_2, \dots, a_N , заданных по возрастанию ($0 \leq a_i \leq 10^5$).

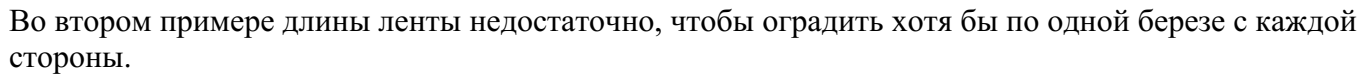
Четвертая и пятая строки описывают березы вдоль правой стороны аллеи. Четвертая строка содержит число M — количество берез ($1 \leq M \leq 2000$), а пятая строка содержит M различных целых чисел b_1, b_2, \dots, b_M , заданных по возрастанию ($0 \leq b_i \leq 10^5$).

Выходные данные

Выходной файл OUTPUT.TXT должен содержать одно целое число: максимальное количество берез, которое можно оградить заданной лентой. Гарантируется, что если максимальное число берез, которое можно оградить лентой длины L , равно X , то нет способа оградить $(X + 1)$ березу лентой длины $(L + 10^{-5})$.

Пояснения к примерам

В первом примере можно, например, оградить березы способом, указанным на рисунке ниже.



№	INPUT.TXT	OUTPUT.TXT
1	18 4 3 0 3 6 4 0 3 6 10	5
2	5 3 1 0 1 0	0

```

/*
    Задача: 928. Березовая аллея

    Решение: Sqrt-декомпозиция, дерево отрезков, динамическое программирование, бинарный по

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <cassert>
#include <limits>

/**
 * Traits for min-queries on ranges, modifications disabled
 */
template<typename T>
struct MinRangeTraits {
    static T merge(const T& lhs, const T& rhs) { return std::min(lhs, rhs); }
    static T neutral() { return std::numeric_limits<T>::max(); }
};

/**
 * SegmentTree class. Effective bottom-to-top implementation
 */
template<typename ItemType, typename ItemTraits = MinRangeTraits<ItemType>>
struct SegmentTree {
    /**
     * Public data: `n` - number of items in array and `data` - tree's container
     */

```

```

    int n; std::vector<ItemType> data;

/**
 * Main methods: resize(nItems), build(array), get(left, right), where 0 <= left <= right <
 */
    void resize(const int n_){
        n = n_;
        int pow = 1;
        while (pow < n) { pow *= 2; }
        data.assign(2 * pow, ItemTraits::neutral());
    }

    template<typename T>
    void build(const std::vector<T>& arr) {
        resize((int)arr.size());
        for (int v = 0; v < n; ++v) {
            data[v + n] = arr[v];
        }
        for (int v = n-1; v >= 1; --v) {
            data[v] = ItemTraits::merge(data[2*v], data[2*v+1]);
        }
    }

    ItemType get(int ql, int qr) const {
        ql += n, qr += n;
        ItemType ret = ItemTraits::neutral();
        while (ql <= qr) {
            if (ql % 2 == 1) { ret = ItemTraits::merge(ret, data[ql++]); }
            if (qr % 2 == 0) { ret = ItemTraits::merge(ret, data[qr--]); }
            ql /= 2; qr /= 2;
        }
        return ret;
    }
}; /** SegmentTree class end */

/**
 * SqrtDecomposition class
 */
template<typename ItemType, typename ItemTraits = MinRangeTraits<ItemType>>
struct SqrtDecomposition {
/**
 * Public data: `data` - original array, `group` - additional information per group, `gsize`
 */
    int gsize;
    std::vector<ItemType> data, group;

/**
 * Main methods: resize(nItems), build(array), get(left, right), where 0 <= left <= right <
 */
    void resize(const int n) {
        gsize = std::max(1, (int)std::sqrt((double)(n) / 2));
        data.assign(n, ItemTraits::neutral());
        group.assign((n + gsize - 1) / gsize, ItemTraits::neutral());
    }

    template<typename T>
    void build(const std::vector<T>& arr) {
        resize((int)arr.size());
        for (int i = 0; i < (int)data.size(); ++i) {
            data[i] = arr[i];
        }
        for (int g = 0; g < (int)group.size(); ++g) {
            const int begin = g * gsize;
            const int after = std::min(begin + gsize, (int)data.size());
            for (int i = begin; i < after; ++i) {
                group[g] = ItemTraits::merge(group[g], data[i]);
            }
        }
    }
};

```

```

ItemType get(const int l, const int r) const {
    const int gl = l / gsize;
    const int gr = r / gsize;
    ItemType result = ItemTraits::neutral();
    if (gl == gr) {
        for (int i = l; i <= r; ++i) {
            result = ItemTraits::merge(result, data[i]);
        }
    } else {
        for (int i = l; i < (gl+1) * gsize; ++i) {
            result = ItemTraits::merge(result, data[i]);
        }
        for (int g = gl+1; g < gr; ++g) {
            result = ItemTraits::merge(result, group[g]);
        }
        for (int i = gr * gsize; i <= r; ++i) {
            result = ItemTraits::merge(result, data[i]);
        }
    }
    return result;
}
}; /** SqrtDecomposition class end */

/**
 * Solution of original problem
 */
template<typename RMQ>
int solve(const int mxLen, const int width, const std::vector<int>& a, const std::vector<int>& b) {
    const int n = (int)(a.size()), m = (int)(b.size());
    // i + j == const
    std::vector<RMQ> diag(n + m - 1);
    auto dist = [&](const int i, const int j) {
        return (double)std::sqrt(double(a[i] - b[j]) * (a[i] - b[j]) + (double)(width) * width);
    };
    std::vector<int> minOnDiag(n + m - 1, (int)1e9), maxOnDiag(n + m - 1, -(int)1e9);
    for (int d = 0; d < n + m - 1; ++d) {
        std::vector<double> arr;
        for (int i = 0; i <= d; ++i) {
            const int j = d - i;
            if (j < 0 || j >= m) { continue; }
            double p = dist(0,0) + (a[i] - a[0]) + (b[j] - b[0]) + dist(i, j);
            arr.push_back(p);
            minOnDiag[d] = std::min(i, minOnDiag[d]);
            maxOnDiag[d] = std::max(i, maxOnDiag[d]);
        }
        diag[d].resize((int)arr.size());
        diag[d].build(arr);
    }

    auto can = [&](const int count) {
        //return canSlow(count);
        bool ok = false;
        for (int d = count - 2; d < n + m - 1; ++d) {
            for (int i = 0; i < n; ++i) {
                const int j = d - count - i + 2;
                if (j < 0 || j >= m) { continue; }
                // (t-i+1)+(k-j+1) == count
                // t + k = count + i + j - 2
                // t = i --> k = count + j - 2
                // k = j --> t = count + i - 2
                // t + k == d
                // i <= t && t <= n-1
                // j <= k && k <= m-1

                // i <= t && t <= (n-1)
                // j <= (d-t) && (d-t) <= (m-1)

                // i <= t && t < n
            }
        }
    };
}

```

```

        // t <= d - j && (d - m + 1) <= t

        int r = std::min(n-1, d - j);
        int l = std::max(i, d - m + 1);
        if (l > r) { continue; }
        assert(i <= l && l <= n-1);
        assert(i <= r && r <= n-1);
        assert(j <= (d-1) && (d-1) <= m-1);
        assert(j <= (d-r) && (d-r) <= m-1);
        assert(l >= minOnDiag[d]);
        assert(r <= maxOnDiag[d]);
        l -= minOnDiag[d];
        r -= minOnDiag[d];
        auto res = diag[d].get(l, r) - diag[i+j].get(i-minOnDiag[i+j], i-minOnDiag[
        ok = ok || (res <= mxLen + 1e-5);
        if (ok) { return true; }
    }
}
return ok;
};
int low = 1, high = (int)(n + m + 1);
while (high - low > 1) {
    int mid = (low + high) / 2;
    if (can(mid)) { low = mid; }
    else { high = mid; }
}
return (low == 1) ? 0 : low;
}

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0); std::cout.tie(0); std::cerr.tie(0);
    int mxLen, width, n, m;
    while (std::cin >> mxLen >> width) {

        std::cin >> n;
        std::vector<int> a(n);
        for (auto &it : a) { std::cin >> it; }

        std::cin >> m;
        std::vector<int> b(m);
        for (auto &it : b) { std::cin >> it; }

        std::sort(a.begin(), a.end());
        std::sort(b.begin(), b.end());
        std::cout << solve<SegmentTree<double>>(mxLen, width, a, b) << std::endl;
    }
    return 0;
}

```

ЗАДАЧА №1391

Конденсация графа

(Время: 3 сек. Память: 32 Мб Сложность: 70%)

Вам задан связный ориентированный граф. Найдите компоненты сильной связности заданного графа и топологически отсортируйте его конденсацию.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа N и M ($1 \leq N \leq 20\,000$, $1 \leq M \leq 200\,000$) – число вершин и ребер соответственно. Каждая из следующих M строк содержит описание ребра: два целых числа из диапазона от 1 до N – номера начала и конца ребра.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите число K – количество компонент сильной связности в заданном графе. На следующей строке выведите N чисел – для каждой вершины от 1 до N выведите номер компоненты сильной связности, которой принадлежит эта вершина. Компоненты сильной связности должны быть занумерованы таким образом, чтобы для любого ребра номер компоненты сильной связности его начала не превышал номера компоненты сильной связности его конца.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	10 19	2 1 2 2 1 1 2 2 2 2 1
	1 4	
	7 8	
	5 10	
	8 9	
	9 6	
	2 6	
	6 2	
	3 8	
	9 2	
	7 2	
	9 7	
	4 5	
	3 6	
	7 3	
	6 7	
	10 8	
	10 1	
	2 9	
	2 7	

/*

Задача: 1391. Конденсация графа

Решение: графы, топологическая сортировка, dfs, конденсация, $O(n + m)$

Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <functional>
#include <cassert>
#include <queue>

int main() {
    int n, m, nParts = 0; scanf("%d %d", &n, &m);
    std::vector<std::vector<int>> next(1+n), prev(1+n);
    for (int i = 0; i < m; ++i) {
        int u, v; scanf("%d %d", &u, &v);
        next[u].push_back(v);
        prev[v].push_back(u);
    }
    std::vector<int> visited(1+n), order, part(1+n);
    std::function<void(int)> topsort = [&](int u) {
        if (visited[u]) return;
        visited[u] = 1;
        for (int v : next[u]) {
            topsort(v);
        }
        order.push_back(u);
    };
    for (int u = 1; u <= n; ++u) topsort(u);
    std::reverse(order.begin(), order.end());
    for (int u : order) {
        if (!part[u]) {
            part[u] = ++nParts;
            std::queue<int> queue; queue.push(u);
            while (!queue.empty()) {
                auto curr = queue.front(); queue.pop();
                for (int v : prev[curr]) {
                    if (!part[v]) {
                        part[v] = nParts;
                        queue.push(v);
                    }
                }
            }
        }
    }
    printf("%d\n", nParts);
    for (int u = 1; u <= n; ++u) printf("%d ", part[u]);
    return 0;
}
```

ЗАДАЧА №1405

Лягушка

(Время: 5 сек. Память: 128 Мб Сложность: 71%)

По кругу расположены N камней, пронумерованных от 0 до $N-1$ в направлении по часовой стрелке. На i -ом камне написано число a_i .

Лягушка OSFrog начинает своё путешествие с камня под номером x и хочет совершить h прыжков. На j -ом прыжке ($0 \leq j \leq h-1$), находясь на i -ом камне, она прыгает на a_i камней по или против часовой стрелки. Направление определяется следующим образом: если количество единиц в двоичной записи j чётно, то она прыгает по часовой, иначе – против.

Дано M запросов с изначальным положением лягушки и количеством прыжков, которое она совершит. Определите номер камня, на котором она закончит прыжки.

Входные данные

В первой строке входного файла INPUT.TXT содержится целое число N – количество камней ($1 \leq N \leq 2 \times 10^5$). Во второй строке содержатся N целых чисел a_i , написанных на камнях ($0 \leq a_i \leq 10^9$). В третьей строке содержится целое число M – количество запросов ($1 \leq M \leq 2 \times 10^5$). В следующих M строках содержатся запросы в виде пары чисел x_k и h_k ($0 \leq x_k \leq N-1$; $0 \leq h_k \leq 10^9$).

Выходные данные

В выходной файл OUTPUT.TXT для каждого запроса в отдельной строке выведите номер камня, на котором лягушка закончит путешествие.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	5	1 2 2 3
	1 4 2 5 3	
	4	
	0 1	
	0 2	
	2 3	
	3 100	

```
/*
    Задача: 1405. Лягушка

    Решение: Sparse Table,  $O((n+q) \cdot \log(\text{MAX}))$ 

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/

#include <stdio.h>
#include <vector>
#include <functional>

inline int mod(int a, int MOD) {
```

```

    return a >= 0 ? a % MOD : MOD - (-a) % MOD;
}

inline int add(int a, int b, int MOD) {
    return (a += b) >= MOD ? a - MOD : a;
}

inline int sub(int a, int b, int MOD) {
    return (a -= b) < 0 ? a + MOD : a;
}

int main() {
    int n; scanf("%d", &n);

    std::vector<int> a(n);
    for (auto& it : a) {
        scanf("%d", &it);
        it = mod(it, n);
    }

    static int jump[2][32][200000]; // [inverse][pow][pos] --> jump

    for (int i = 0; i < n; ++i) {
        jump[0][0][i] = add(i, a[i], n);
        jump[1][0][i] = sub(i, a[i], n);
    }

    for (int pow = 1; pow < 32; ++pow) {
        for (int i = 0, next; i < n; ++i) {
            next = jump[0][pow-1][i];
            jump[0][pow][i] = jump[1][pow-1][next];

            next = jump[1][pow-1][i];
            jump[1][pow][i] = jump[0][pow-1][next];
        }
    }

    std::function<int(int,int)> query = [&](int p, int nSteps) {
        for (int pow = 31, inv = 0; pow >= 0; --pow) {
            if ((nSteps >> pow) & 1) {
                p = jump[inv][pow][p];
                inv = 1 - inv;
            }
        }
        return p;
    };

    int nQueries; scanf("%d", &nQueries);
    while (nQueries--) {
        int pos, nSteps;
        scanf("%d %d", &pos, &nSteps);
        printf("%d\n", query(pos, nSteps));
    }
    return 0;
}

```

ЗАДАЧА №931

Две окружности - 2

(Время: 2 сек. Память: 16 Мб Сложность: 72%)

Юный футболист Митя обнаружил на школьном футбольном поле две различные окружности, нарисованные едва заметной белой краской. Вспомнив истории о загадочных кругах на полях, он отметил эти окружности с помощью небольших камушков. Митя разложил на поле n камушков так, чтобы каждый из них находился на одной из окружностей или даже на их пересечении, если эти окружности пересекаются. Получилось так, что на каждой окружности размещался хотя бы один камушек. Обладая отличным глазомером, Митя расположил камушки на окружностях абсолютно точно, без какой-либо погрешности.

На следующий день пошел дождь, краска стерлась, и нарисованные окружности исчезли, но все камушки остались на своих местах. Теперь Мите очень нужно найти доказательство необычного явления, свидетелем которого он был, то есть, восстановить окружности.

Требуется написать программу, которая по координатам камушков на поле находит вариант размещения их на двух несовпадающих окружностях.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n — количество размещенных Митей камушков на поле ($2 \leq n \leq 2000$). Последующие n строк содержат целочисленные координаты камушков (x_i, y_i) — в каждой строке по одной паре координат, разделенных пробелом ($-10^6 \leq x_i, y_i \leq 10^6$).

Никакие два камушка не размещаются в одной точке. Гарантируется, что ответ для заданного набора камушков существует.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать две строки. Первая строка должна содержать последовательность номеров всех камушков, которые принадлежат первой окружности, вторая строка — последовательность номеров всех камушков, которые принадлежат второй окружности. Считается, что камушки пронумерованы от 1 до n в порядке их следования во входных данных.

Каждый камушек должен встречаться хотя бы в одной из двух последовательностей. Если камушек встречается в обеих последовательностях, то это обозначает, что он находится на пересечении окружностей.

Нумерация окружностей не имеет значения, то есть выводить две последовательности можно в любом порядке. Числа в последовательностях можно также выводить в произвольном порядке. Каждая из последовательностей должна содержать не менее одного числа. Все числа в строках должны быть разделены пробелами.

Если вариантов расположения окружностей несколько, можно выбрать любой из них.

Примеры

Пояснения к примерам

В первом примере одна из искомых окружностей имеет центр в точке с координатами (1, 0), а вторая — в точке с координатами (3, 0). Обе окружности имеют радиус равный 1.

Во втором примере центр первой окружности совпадает с началом координат, а радиус равен 10^6 . Вторая окружность — любая, проходящая через точки $(-10^6, 0)$ и $(0, 0)$.

В обоих примерах возможны и другие правильные ответы.

№	INPUT.TXT	OUTPUT.TXT
1	7 1 -1 0 0 1 1 3 1 3 -1 2 0 4 0	6 1 2 3 4 7 5 6
2	5 -1000000 0 0 1000000 1000000 0 0 -1000000 0 0	1 2 3 4 1 5

```

/*
    Задача: 931. Две окружности - 2

    Решение: длинная арифметика, геометрия, окружность, окружность по трем точкам, O(n)

    Автор: Дмитрий Козырев, https://github.com/dmkz , dmkozyrev@rambler.ru
*/

```

```

#include <stdio.h>
#include <algorithm>
#include <vector>
#include <cassert>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <iomanip>
#include <complex>
#include <climits>

const int BASE_DIGITS = 9;
const int BASE = 1000000000;

struct BigInt {

    int sign;
    std::vector<int> a;

    // ----- Constructors -----
    // Default constructor.
    BigInt() : sign(1) {}

    // Constructor from long long.
    BigInt(long long v) {
        *this = v;
    }
    BigInt& operator = (long long v) {

```

```

        sign = 1;
        if (v < 0) {
            sign = -1;
            v = -v;
        }
        a.clear();
        for (; v > 0; v = v / BASE)
            a.push_back(int(v % BASE));
        return *this;
    }

    // Copy constructor.
    BigInt(const BigInt& other) {
        sign = other.sign;
        a = other.a;
    }
    // Assignment operator using Copy-and-swap idiom.
    friend void swap(BigInt& a, BigInt& b) {
        std::swap(a.sign, b.sign);
        std::swap(a.a, b.a);
    }
    BigInt& operator = (BigInt other) {
        swap(*this, other);
        return *this;
    }
    BigInt(BigInt&& other) : BigInt() {
        swap(*this, other);
    }

    // Initialize from string.
    BigInt(const std::string& s) {
        read(s);
    }

    // ----- Input / Output -----
    void read(const std::string& s) {
        sign = 1;
        a.clear();
        int pos = 0;
        while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+')) {
            if (s[pos] == '-')
                sign = -sign;
            ++pos;
        }
        for (int i = (int)s.size() - 1; i >= pos; i -= BASE_DIGITS) {
            int x = 0;
            for (int j = std::max(pos, i - BASE_DIGITS + 1); j <= i; j++)
                x = x * 10 + s[j] - '0';
            a.push_back(x);
        }
        trim();
    }
    friend std::istream& operator>>(std::istream &stream, BigInt &v) {
        std::string s;
        stream >> s;
        v.read(s);
        return stream;
    }

    friend std::ostream& operator<<(std::ostream &stream, const BigInt &v) {
        if (v.sign == -1 && !v.isZero())
            stream << '-';
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int) v.a.size() - 2; i >= 0; --i)
            stream << std::setw(BASE_DIGITS) << std::setfill('0') << v.a[i];
        return stream << std::setfill(' ');
    }

    // ----- Comparison -----

```

```

bool operator<(const BigInt &v) const {
    if (sign != v.sign)
        return sign < v.sign;
    if (a.size() != v.a.size())
        return (int)a.size() * sign < (int)v.a.size() * v.sign;
    for (int i = ((int) a.size()) - 1; i >= 0; i--)
        if (a[i] != v.a[i])
            return a[i] * sign < v.a[i] * sign;
    return false;
}

bool operator>(const BigInt &v) const {
    return v < *this;
}

bool operator<=(const BigInt &v) const {
    return !(v < *this);
}

bool operator>=(const BigInt &v) const {
    return !(*this < v);
}

bool operator==(const BigInt &v) const {
    return !(*this < v) && !(v < *this);
}

bool operator!=(const BigInt &v) const {
    return *this < v || v < *this;
}

// Returns:
// 0 if |x| == |y|
// -1 if |x| < |y|
// 1 if |x| > |y|
friend int __compare_abs(const BigInt& x, const BigInt& y) {
    if (x.a.size() != y.a.size()) {
        return x.a.size() < y.a.size() ? -1 : 1;
    }

    for (int i = ((int) x.a.size()) - 1; i >= 0; --i) {
        if (x.a[i] != y.a[i]) {
            return x.a[i] < y.a[i] ? -1 : 1;
        }
    }
    return 0;
}

// ----- Unary operator - and operators +- -----
BigInt operator-() const {
    BigInt res = *this;
    if (isZero()) return res;

    res.sign = -sign;
    return res;
}

// Assumption: *this and v have same sign.
void __internal_add(const BigInt& v) {
    if (a.size() < v.a.size()) {
        a.resize(v.a.size(), 0);
    }
    for (int i = 0, carry = 0; i < (int) std::max(a.size(), v.a.size()) || carry; ++i)
        if (i == (int) a.size()) a.push_back(0);

    a[i] += carry + (i < (int) v.a.size() ? v.a[i] : 0);
    carry = a[i] >= BASE;
    if (carry) a[i] -= BASE;
}

// Assumption: *this and v have same sign, and |*this| >= |v|
void __internal_sub(const BigInt& v) {

```

```

        for (int i = 0, carry = 0; i < (int) v.a.size() || carry; ++i) {
            a[i] -= carry + (i < (int) v.a.size() ? v.a[i] : 0);
            carry = a[i] < 0;
            if (carry) a[i] += BASE;
        }
        this->trim();
    }

    BigInt operator += (const BigInt& v) {
        if (sign == v.sign) {
            __internal_add(v);
        } else {
            if (__compare_abs(*this, v) >= 0) {
                __internal_sub(v);
            } else {
                BigInt vv = v;
                swap(*this, vv);
                __internal_sub(vv);
            }
        }
        return *this;
    }

    BigInt operator -= (const BigInt& v) {
        if (sign == v.sign) {
            if (__compare_abs(*this, v) >= 0) {
                __internal_sub(v);
            } else {
                BigInt vv = v;
                swap(*this, vv);
                __internal_sub(vv);
                this->sign = -this->sign;
            }
        } else {
            __internal_add(v);
        }
        return *this;
    }

    // Optimize operators + and - according to
    // https://stackoverflow.com/questions/13166079/move-semantics-and-pass-by-rvalue-refer
    template< typename L, typename R >
    typename std::enable_if<
        std::is_convertible<L, BigInt>::value &&
        std::is_convertible<R, BigInt>::value &&
        std::is_lvalue_reference<R&&>::value,
        BigInt>::type friend operator + (L&& l, R&& r) {
        BigInt result(std::forward<L>(l));
        result += r;
        return result;
    }

    template< typename L, typename R >
    typename std::enable_if<
        std::is_convertible<L, BigInt>::value &&
        std::is_convertible<R, BigInt>::value &&
        std::is_rvalue_reference<R&&>::value,
        BigInt>::type friend operator + (L&& l, R&& r) {
        BigInt result(std::move(r));
        result += l;
        return result;
    }

    template< typename L, typename R >
    typename std::enable_if<
        std::is_convertible<L, BigInt>::value &&
        std::is_convertible<R, BigInt>::value,
        BigInt>::type friend operator - (L&& l, R&& r) {
        BigInt result(std::forward<L>(l));
        result -= r;
    }

```



```

    return result;
}

// ----- Operators * / % -----
friend std::pair<BigInt, BigInt> divmod(const BigInt& a1, const BigInt& b1) {
    assert(b1 > 0); // divmod not well-defined for b < 0.

    int norm = BASE / (b1.a.back() + 1);
    BigInt a = a1.abs() * norm;
    BigInt b = b1.abs() * norm;
    BigInt q = 0, r = 0;
    q.a.resize(a.a.size());

    for (int i = (int)a.a.size() - 1; i >= 0; i--) {
        r *= BASE;
        r += a.a[i];
        int s1 = (int)r.a.size() <= (int)b.a.size() ? 0 : r.a[b.a.size()];
        int s2 = (int)r.a.size() <= (int)b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
        long long d = ((int64_t) BASE * s1 + s2) / b.a.back();
        assert(0 <= d && d < INT_MAX);
        r -= b * (int)d;
        while (r < 0) {
            r += b, --d;
        }
        assert(0 <= d && d < BASE);
        q.a[i] = (int)d;
    }

    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
    r.trim();
    auto res = std::make_pair(q, r / norm);
    if (res.second < 0) res.second += b1;
    return res;
}

BigInt operator/(const BigInt &v) const {
    return divmod(*this, v).first;
}

BigInt operator%(const BigInt &v) const {
    return divmod(*this, v).second;
}

void operator/=(int v) {
    assert(v > 0); // operator / not well-defined for v <= 0.
    if (llabs(v) >= BASE) {
        *this /= BigInt(v);
        return ;
    }
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
        long long cur = a[i] + rem * (long long) BASE;
        a[i] = (int) (cur / v);
        rem = (int) (cur % v);
    }
    trim();
}

BigInt operator/(int v) const {
    assert(v > 0); // operator / not well-defined for v <= 0.

    if (llabs(v) >= BASE) {
        return *this / BigInt(v);
    }
    BigInt res = *this;
    res /= v;
    return res;
}

```

```

}
void operator/=(const BigInt &v) {
    *this = *this / v;
}

int operator%(int v) const {
    assert(v > 0); // operator / not well-defined for v <= 0.
    assert(v < BASE);
    int m = 0;
    for (int i = (int)a.size() - 1; i >= 0; --i)
        m = int((a[i] + m * (long long) BASE) % v);
    return m * sign;
}

void operator*=(int v) {
    if (llabs(v) >= BASE) {
        *this *= BigInt(v);
        return ;
    }
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
        if (i == (int) a.size())
            a.push_back(0);
        long long cur = a[i] * (long long) v + carry;
        carry = (int) (cur / BASE);
        a[i] = (int) (cur % BASE);
    }
    trim();
}

BigInt operator*(int v) const {
    if (std::abs(v) >= BASE) {
        return *this * BigInt(v);
    }
    BigInt res = *this;
    res *= v;
    return res;
}

// Convert BASE 10^old --> 10^new.
static std::vector<int> convert_base(const std::vector<int> &a, int old_digits, int new_digits) {
    std::vector<long long> p(std::max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int) p.size(); i++)
        p[i] = p[i - 1] * 10;
    std::vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int) a.size(); i++) {
        cur += a[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.push_back(int(cur % p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int) cur);
    while (!res.empty() && !res.back())
        res.pop_back();
    return res;
}

static void fft(std::vector<std::complex<double> > &a, bool invert) {
    int n = (int) a.size();

    for (int i = 1, j = 0; i < n; ++i) {
        int bit = n >> 1;

```

```

        for (; j >= bit; bit >>= 1)
            j -= bit;
        j += bit;
        if (i < j)
            std::swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <= 1) {
        double ang = 2 * 3.14159265358979323846 / len * (invert ? -1 : 1);
        std::complex<double> wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            std::complex<double> w(1);
            for (int j = 0; j < len / 2; ++j) {
                std::complex<double> u = a[i + j];
                std::complex<double> v = a[i + j + len / 2] * w;
                a[i + j] = u + v;
                a[i + j + len / 2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert)
        for (int i = 0; i < n; ++i)
            a[i] /= n;
}

static void multiply_fft(const std::vector<int> &a, const std::vector<int> &b, std::vec
    std::vector<std::complex<double> > fa(a.begin(), a.end());
    std::vector<std::complex<double> > fb(b.begin(), b.end());
    int n = 1;
    while (n < (int) std::max(a.size(), b.size()))
        n <<= 1;
    n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; ++i)
        fa[i] *= fb[i];
    fft(fa, true);

    res.resize(n);
    long long carry = 0;
    for (int i = 0; i < n; ++i) {
        long long t = (long long) (fa[i].real() + 0.5) + carry;
        carry = t / 1000;
        res[i] = int(t % 1000);
    }
}

BigInt mul_simple(const BigInt &v) const {
    BigInt res;
    res.sign = sign * v.sign;
    res.a.resize(a.size() + v.a.size());
    for (int i = 0; i < (int) a.size(); ++i)
        if (a[i])
            for (int j = 0, carry = 0; j < (int) v.a.size() || carry; ++j) {
                long long cur = res.a[i + j] + (long long) a[i] * (j < (int) v.a.size()
                    carry = (int) (cur / BASE);
                    res.a[i + j] = (int) (cur % BASE);
                }
    res.trim();
    return res;
}

typedef std::vector<long long> vll;

static vll karatsubaMultiply(const vll &a, const vll &b) {

```

```

int n = (int)a.size();
vll res(n + n);
if (n <= 32) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            res[i + j] += a[i] * b[j];
    return res;
}

int k = n >> 1;
vll a1(a.begin(), a.begin() + k);
vll a2(a.begin() + k, a.end());
vll b1(b.begin(), b.begin() + k);
vll b2(b.begin() + k, b.end());

vll alb1 = karatsubaMultiply(a1, b1);
vll a2b2 = karatsubaMultiply(a2, b2);

for (int i = 0; i < k; i++)
    a2[i] += a1[i];
for (int i = 0; i < k; i++)
    b2[i] += b1[i];

vll r = karatsubaMultiply(a2, b2);
for (int i = 0; i < (int) alb1.size(); i++)
    r[i] -= alb1[i];
for (int i = 0; i < (int) a2b2.size(); i++)
    r[i] -= a2b2[i];

for (int i = 0; i < (int) r.size(); i++)
    res[i + k] += r[i];
for (int i = 0; i < (int) alb1.size(); i++)
    res[i] += alb1[i];
for (int i = 0; i < (int) a2b2.size(); i++)
    res[i + n] += a2b2[i];
return res;
}

BigInt mul_karatsuba(const BigInt &v) const {
    std::vector<int> a6 = convert_base(this->a, BASE_DIGITS, 6);
    std::vector<int> b6 = convert_base(v.a, BASE_DIGITS, 6);
    vll a_(a6.begin(), a6.end());
    vll b_(b6.begin(), b6.end());
    while (a_.size() < b_.size())
        a_.push_back(0);
    while (b_.size() < a_.size())
        b_.push_back(0);
    while (a_.size() & (a_.size() - 1))
        a_.push_back(0), b_.push_back(0);
    vll c_ = karatsubaMultiply(a_, b_);
    BigInt res;
    res.sign = sign * v.sign;
    long long carry = 0;
    for (int i = 0; i < (int) c_.size(); i++) {
        long long cur = c_[i] + carry;
        res.a.push_back((int) (cur % 1000000));
        carry = cur / 1000000;
    }
    res.a = convert_base(res.a, 6, BASE_DIGITS);
    res.trim();
    return res;
}

void operator*=(const BigInt &v) {
    *this = *this * v;
}

BigInt operator*(const BigInt &v) const {
    if (a.size() * v.a.size() <= 1000111u) return mul_simple(v);
    if (a.size() > 500111u || v.a.size() > 500111u) return mul_fft(v);
}

```

```

        return mul_karatsuba(v);
    }

BigInt mul_fft(const BigInt& v) const {
    BigInt res;
    res.sign = sign * v.sign;
    multiply_fft(convert_base(a, BASE_DIGITS, 3), convert_base(v.a, BASE_DIGITS, 3), res);
    res.a = convert_base(res.a, 3, BASE_DIGITS);
    res.trim();
    return res;
}

// ----- Misc -----
BigInt abs() const {
    BigInt res = *this;
    res.sign *= res.sign;
    return res;
}

void trim() {
    while (!a.empty() && !a.back())
        a.pop_back();
    if (a.empty())
        sign = 1;
}

bool isZero() const {
    return a.empty() || (a.size() == 1 && !a[0]);
}

friend BigInt gcd(const BigInt &a, const BigInt &b) {
    return b.isZero() ? a : gcd(b, a % b);
}

friend BigInt lcm(const BigInt &a, const BigInt &b) {
    return a / gcd(a, b) * b;
}

friend BigInt sqrt(const BigInt &a1) {
    BigInt a = a1;
    while (a.a.empty() || (int)a.a.size() % 2 == 1)
        a.a.push_back(0);

    int n = (int)a.a.size();

    int firstDigit = (int) std::sqrt((double) a.a[n - 1] * BASE + a.a[n - 2]);
    int norm = BASE / (firstDigit + 1);
    a *= norm;
    a *= norm;
    while (a.a.empty() || (int)a.a.size() % 2 == 1)
        a.a.push_back(0);

    BigInt r = (long long) a.a[n - 1] * BASE + a.a[n - 2];
    firstDigit = (int) std::sqrt((double) a.a[n - 1] * BASE + a.a[n - 2]);
    int q = firstDigit;
    BigInt res;

    for(int j = n / 2 - 1; j >= 0; j--) {
        for(; ; --q) {
            BigInt r1 = (r - (res * 2 * BigInt(BASE) + q) * q) * BigInt(BASE) * BigInt(
            if (r1 >= 0) {
                r = r1;
                break;
            }
        }
        res *= BASE;
        res += q;

        if (j > 0) {
            int d1 = res.a.size() + 2u < r.a.size() ? r.a[res.a.size() + 2] : 0;
            int d2 = res.a.size() + 1u < r.a.size() ? r.a[res.a.size() + 1] : 0;

```

```

        int d3 = res.a.size() < r.a.size() ? r.a[res.a.size()] : 0;
        long long temp = ((long long) d1 * BASE * BASE + (long long) d2 * BASE + d3
        assert(0 <= temp && temp < INT_MAX);
        q = (int)temp;
    }
}

    res.trim();
    return res / norm;
}
};

typedef BigInt Real;

struct Frac {
    Real p, q;

    Frac(Real p_ = 0, Real q_ = 1) : p(p_), q(q_) { normalize(); }

    Frac(Frac p_, Frac q_) {
        *this = p_ / q_;
    }

    void normalize() {
        if (q < 0) { p = -p; q = -q; }
    }

    inline Frac operator+(const Frac& f) const {
        return Frac(p * f.q + q * f.p, q * f.q);
    }

    inline Frac operator-(const Frac& f) const {
        return Frac(p * f.q - q * f.p, q * f.q);
    }

    inline Frac operator*(const Frac& f) const {
        return Frac(p * f.p, q * f.q);
    }

    inline Frac operator/(const Frac& f) const {
        return Frac(p * f.q, q * f.p);
    }

    inline bool operator==(const Frac& f) const {
        return p * f.q == q * f.p;
    }

    inline bool operator!=(const Frac& f) const {
        return !(*this == f);
    }
};

struct Point {
    Frac x, y;

    Point(Frac x_ = Frac(), Frac y_ = Frac()) : x(x_), y(y_) { }

    inline Point operator-(const Point& p) const {
        return Point(x - p.x, y - p.y);
    }

    inline Point operator+(const Point& p) const {
        return Point(x + p.x, y + p.y);
    }

    inline Point operator*(const Frac& f) const {
        return Point(x * f, y * f);
    }
};

```

```

    }

    inline Point operator/(const Frac& f) const {
        return Point(x / f, y / f);
    }

    inline Frac norm2() const {
        return x * x + y * y;
    }
};

inline Frac det(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}

inline bool get_circle_center(const Point& p1, const Point& p2, const Point& p3, Point& Ans)
{
    auto d = det(p2 - p1, p3 - p1) * Real(2);
    if (d == Frac(0)) return false;
    auto T = Point(p1.norm2() - p2.norm2(), p1.norm2() - p3.norm2());
    auto x = Frac(det(T, Point(p1.y-p2.y, p1.y-p3.y)), d);
    auto y = Frac(det(Point(p1.x-p2.x, p1.x-p3.x), T), d);
    Ans = Point(x, y);
    return true;
}

inline bool same_circle_dist(const Point& C, const Point& A, const Point& B) {
    return (A-C).norm2() == (B-C).norm2();
}

int main() {
    int n; scanf("%d", &n);
    if (n == 2) {
        printf("1\n2\n");
        return 0;
    } else if (n == 3) {
        printf("1 2\n3\n");
        return 0;
    } else if (n == 4) {
        printf("1 2\n3 4\n");
        return 0;
    }

    std::vector<Point> pt(n);
    std::vector<int> id(n);
    for (int i = 0; i < n; ++i) {
        int x, y;
        scanf("%d %d", &x, &y);
        pt[i] = Point(Real(x), Real(y));
        id[i] = i+1;
    }

    std::srand((uint64_t)std::time(0) ^ (uint64_t)new(char));
    for (int k = 0; k < n; ++k) {
        int i = std::rand() % n;
        int j = std::rand() % n;
        std::swap(pt[i], pt[j]);
        std::swap(id[i], id[j]);
    }

    int m = std::min(n, 5);
    bool success = false;

    std::vector<int> part, group1, group2;
    for (int i1 = 0; !success && i1 < m; ++i1) {
        for (int i2 = i1+1; !success && i2 < m; ++i2) {
            for (int i3 = i2+1; !success && i3 < m; ++i3) {
                Point C1, C2;
                if (!get_circle_center(pt[i1], pt[i2], pt[i3], C1)) continue;
            }
        }
    }
}

```

```

part.assign(n, 0);

part[i1] = part[i2] = part[i3] = 1;
for (int t = 0; t < n; ++t) {
    if (!part[t] && same_circle_dist(C1, pt[i1], pt[t])) {
        part[t] = 1;
    }
}

int j1 = -1, j2 = -1, j3 = -1, t = 0;
while (t < n && part[t]) ++t;
if (t == n) { // All points at one circle
    success = true;
    //continue;
}
if (!success) {
    j1 = t++; part[j1] = 2;
    while (t < n && part[t]) ++t;
    if (t == n) { // One point on another circle
        success = true;
        //continue;
    }
}
if (!success) {
    j2 = t++; part[j2] = 2;
    while (t < n && part[t]) ++t;
    if (t == n) { // Two points on another circle
        success = true;
        //continue;
    }
}
if (!success) {
    j3 = t++; part[j3] = 2; // Main case
    if (!get_circle_center(pt[j1], pt[j2], pt[j3], C2)) continue;
    success = true;
    for (; t < n; ++t) {
        if (part[t]) continue;
        if (!same_circle_dist(C2, pt[j1], pt[t])) {
            success = false;
            break;
        }
        part[t] = 2;
    }
}
if (success) {
    for (t = 0; t < n; ++t) {
        if (part[t] == 1) {
            group1.push_back(t);
        }
        if (part[t] == 2 || (j1 != -1 && same_circle_dist(C2, pt[t], pt[j1])) {
            group2.push_back(t);
        }
    }
}

}

}

assert(success);

if (group2.empty()) {
    group2.push_back(0);
}
for (auto& it : group1) {
    printf("%d ", id[it]);
}
printf("\n");
for (auto& it : group2) {
    printf("%d ", id[it]);
}

```



```
    }  
    printf("\n");  
    return 0;  
}
```

ЗАДАЧА №1173

Обратный элемент

(Время: 1 сек. Память: 16 Мб Сложность: 73%)

Кольцо вычетов – система операций сложения и умножения над целыми числами по модулю m . Все операции и их результат представляются числами диапазона от 0 до $m-1$.

Обратный элемент числа a в кольце вычетов по модулю m – это такое число a^{-1} , что $a \cdot a^{-1} = 1$.

Требуется определить обратный элемент для числа a в кольце вычетов по модулю m .

Входные данные

Входной файл INPUT.TXT содержит целые числа a и m – число и основание кольца вычетов ($0 \leq a < m \leq 10^9$, $m > 1$).

Выходные данные

В выходной файл OUTPUT.TXT выведите значение обратного элемента для числа a , если он существует в кольце вычетов по модулю m . В противном случае выведите «No solution».

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	7 10	3
2	3217 16384	13425
3	4 16	No solution

```
#include <stdio.h>
#include <cassert>

void euclid(int a, int b, int& x, int& y, int& d) {
    // Вычисление:  $a \cdot x + b \cdot y = \gcd(a, b) = d$ 
    if (a < b) {
        return euclid(b, a, y, x, d);
    }

    assert(a >= b && b >= 0);

    if (b == 0) {
        d = a, x = 1, y = 0;
        return;
    }

    int q, r, x2 = 1, x1 = 0, y2 = 0, y1 = 1;

    while (b > 0) {
        q = a / b;
        r = a - q * b;
        x = x2 - q * x1;
        y = y2 - q * y1;
        a = b;
        b = r;
```

```

        x2 = x1, x1 = x, y2 = y1, y1 = y;
    }
    d = a, x = x2, y = y2;
}

int main() {
    int a, m;
    scanf("%d %d", &a, &m);
    int x, y, d;
    euclid(a, m, x, y, d);
    if (x < 0) x += m;
    assert(0 <= x && x < m);
    if (1LL * a * x % m != 1) {
        printf("No solution");
    } else {
        printf("%d", x);
    }
    return 0;
}

```

ЗАДАЧА №532

Трамвай

(Время: 3 сек. Память: 16 Мб Сложность: 74%)

С окраины в центр города каждое утро по одному маршруту едут в трамвае N человек. За долгое время поездок они достаточно хорошо узнали друг друга. Чтобы никому не было обидно, они захотели решить, кто из них и между какими остановками маршрута должен сидеть, а кто должен стоять. Все остановки пронумерованы от 1 до P .

Один из пассажиров оказался знатоком теории математического моделирования. Он предложил рассмотреть значение суммарного удовлетворения пассажиров. Для каждого i -го пассажира он оценил две величины — a_i и b_i . Если в течение одного переезда между остановками пассажир сидит, то к суммарному удовлетворению прибавляется a_i , если же он стоит, то прибавляется b_i .

Всего в трамвае M сидячих мест. Вставать и садиться пассажиры могут мгновенно на любой остановке. Кроме того, некоторые пассажиры предпочитают ехать стоя, даже если в трамвае есть свободные места (для них $a_i < b_i$).

Требуется написать программу, которая вычисляет значение максимально достижимого суммарного удовлетворения, если для каждого i -го пассажира известны величины a_i и b_i , а также номера остановок, на которых он садится и выходит из трамвая.

Входные данные

Первая строка входного файла INPUT.TXT содержит разделенные пробелом три целых числа N , M и P — число пассажиров, число сидячих мест и число остановок на маршруте соответственно ($1 \leq N$, $M, P \leq 100\,000$; $2 \leq P$).

Каждая из следующих N строк содержит информацию об очередном пассажире в виде четырех целых чисел a_i , b_i , c_i , d_i , где первые два числа определяют вклад в параметр счастья, третье — номер остановки, на которой пассажир садится в трамвай, и последнее — номер остановки, на которой он выходит из трамвая ($-10^6 \leq a_i, b_i \leq 10^6$; $1 \leq c_i < d_i \leq P$).

Выходные данные

В выходной файл OUTPUT.TXT необходимо вывести одно целое число — максимальное суммарное удовлетворение, которого могут добиться пассажиры.

Пример

Пояснение к примеру

Максимальное суммарное довольство достигается следующим образом:

- На первой остановке входят и садятся второй и третий пассажиры;
- На второй остановке входят первый и четвертый пассажиры, второй уступает место первому;
- На третьей остановке встают и выходят первый и третий пассажиры, второй и четвертый садятся на их места;
- На четвертой остановке выходят второй и четвертый пассажиры.

№	INPUT.TXT	OUTPUT.TXT
1	4 2 4 10 -10 2 3 -1 -3 1 4 6 -6 1 3 7 4 2 4	28

```
#pragma GCC diagnostic ignored "-Wunused-result"
#include <stdio.h>
#include <bits/stdc++.h>

typedef long long ll;

struct Record {
    int sit_score, stay_score, id;

    inline int diff_score() const {
        return sit_score - stay_score;
    }
};

const int IN = 1;
const int OUT = 0;

struct Event {
    int time, type, id;
};

bool operator<(const Record& a, const Record& b) {
    int a_diff = a.diff_score();
    int b_diff = b.diff_score();
    return a_diff > b_diff || (a_diff == b_diff && a.id < b.id);
}

int main() {
    int n, nPlaces, nStops;
    scanf("%d %d %d", &n, &nPlaces, &nStops);

    std::vector<Record> records(n);
    std::vector<Event> events;
    for (int i = 0; i < n; ++i) {
        int a, b, c, d; scanf("%d %d %d %d", &a, &b, &c, &d);
        records[i] = Record{a, b, i};
        events.push_back(Event{c, IN, i});
        events.push_back(Event{d, OUT, i});
    }
    std::sort(events.begin(), events.end(), [](const Event& a, const Event& b){
        return a.time < b.time || (a.time == b.time && (a.type < b.type || (a.type == b.type && a.id < b.id)));
    });

    std::set<Record> sit, stay;
    ll answer = 0, sum_stay = 0, sum_sit = 0;
    int cur = 0;
    std::vector<bool> is_stay(n, false);
    for (auto& e : events) {
        answer += (sum_stay + sum_sit) * (e.time - cur);
        cur = e.time;
        if (e.type == OUT) {
            if (is_stay[e.id]) {
                stay.erase(records[e.id]);
                sum_stay -= records[e.id].stay_score;
            } else {
                sit.erase(records[e.id]);
                sum_sit -= records[e.id].sit_score;
                if (stay.empty()) continue;
                auto top = stay.begin();

```

```

        if (top->diff_score() <= 0) continue;
        assert((int)sit.size() < nPlaces);
        sit.insert(records[top->id]);
        sum_stay -= records[top->id].stay_score;
        sum_sit += records[top->id].sit_score;
        is_stay[top->id] = false;
        stay.erase(stay.begin());
    }
} else {
    if (records[e.id].diff_score() <= 0) {
        is_stay[e.id] = true;
        stay.insert(records[e.id]);
        sum_stay += records[e.id].stay_score;
        continue;
    }
    if ((int)sit.size() < nPlaces || sit.upper_bound(records[e.id]) != sit.end()) {
        is_stay[e.id] = false;
        sit.insert(records[e.id]);
        sum_sit += records[e.id].sit_score;
        if ((int)sit.size() > nPlaces) {
            auto it = std::prev(sit.end());
            is_stay[it->id] = true;
            sum_sit -= records[it->id].sit_score;
            sum_stay += records[it->id].stay_score;
            stay.insert(records[it->id]);
            sit.erase(it);
        }
        continue;
    }
    is_stay[e.id] = true;
    stay.insert(records[e.id]);
    sum_stay += records[e.id].stay_score;
}
}
assert(sum_sit == 0 && sum_stay == 0);
std::cout << answer << std::endl;

return 0;
}

```

ЗАДАЧА №537

Перестановки - 3

(Время: 1 сек. Память: 16 Мб Сложность: 74%)

Задано множество из N различных натуральных чисел. Перестановку элементов этого множества назовем K -перестановкой, если для любых двух соседних элементов этой перестановки их наибольший общий делитель не менее K . Например, если задано множество элементов $S = \{6, 3, 9, 8\}$, то перестановка $\{8, 6, 3, 9\}$ является 2-перестановкой, а перестановка $\{6, 8, 3, 9\}$ – нет.

Перестановка $\{p_1, p_2, \dots, p_N\}$ будет лексикографически меньше перестановки $\{q_1, q_2, \dots, q_N\}$, если существует такое натуральное число i ($1 \leq i \leq N$), для которого $p_j = q_j$ при $j < i$ и $p_i < q_i$.

В качестве примера упорядочим все K -перестановки заданного выше множества в лексикографическом порядке. Например, существует ровно четыре 2-перестановки множества $S: \{3, 9, 6, 8\}, \{8, 6, 3, 9\}, \{8, 6, 9, 3\}$ и $\{9, 3, 6, 8\}$. Соответственно, первой 2-перестановкой в лексикографическом порядке является множество $\{3, 9, 6, 8\}$, а четвертой – множество $\{9, 3, 6, 8\}$.

Требуется написать программу, позволяющую найти M -ую K -перестановку в этом порядке.

Входные данные

Входной файл INPUT.TXT в первой строке содержит три натуральных числа – N ($1 \leq N \leq 16$), M и K ($1 \leq M, K \leq 10^9$). Вторая строка содержит N различных натуральных чисел, не превосходящих 10^9 . Все числа в строках разделены пробелом.

Выходные данные

В выходной файл OUTPUT.TXT необходимо вывести M -ую K -перестановку заданного множества или -1 , если такой нет.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	4 1 2 6 8 3 9	3 9 6 8
2	4 4 2 6 8 3 9	9 3 6 8
3	4 5 2 6 8 3 9	-1

```
/*
"Перестановки - 3": динамическое программирование по профилю,  $O(n \cdot 2^{(2n)})$ 
*/

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>

inline int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
```

```

}

typedef long long ll;

int main() {
    int n, number, minGCD;
    scanf("%d %d %d", &n, &number, &minGCD);

    std::vector<int> a(n);
    for (auto& it : a) {
        scanf("%d", &it);
    }
    std::sort(a.begin(), a.end());

    std::vector<std::vector<bool>> is_pair(n, std::vector<bool>(n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            is_pair[i][j] = (i != j) && (gcd(a[i], a[j]) >= minGCD);
        }
    }

    std::vector<std::vector<ll>> count(n, std::vector<ll>(1 << n, 0));

    for (int code = 1; code < (1 << n); ++code) {
        for (int first = 0; first < n; ++first) {
            if (code == (1 << first)) {
                count[first][code] = 1;
            } else {
                int without_first = code & ~(1 << first);
                for (int second = 0; second < n; ++second) {
                    if (is_pair[first][second] && ((without_first >> second) & 1)) {
                        count[first][code] += count[second][without_first];
                    }
                }
            }
        }
    }

    std::vector<int> answ(n);
    int code = (1 << n) - 1;
    for (int pos = 0; pos < n; ++pos) {
        bool founded = false;
        for (int i = 0; i < n; ++i) {
            if (((code >> i) & 1) == 1 && (pos == 0 || is_pair[answ[pos-1]][i])) {
                if (count[i][code] >= number) {
                    answ[pos] = i;
                    code &= ~(1 << i);
                    founded = true;
                    break;
                } else {
                    number -= (int)count[i][code];
                }
            }
        }
        if (!founded) {
            printf("-1");
            return 0;
        }
    }

    for (auto& it : answ) {
        printf("%d ", a[it]);
    }

    return 0;
}

```


ЗАДАЧА №219

Симпатичные таблицы

(Время: 1 сек. Память: 16 Мб Сложность: 75%)

Рассмотрим таблицу размера $N \times M$, в клетках которой стоят целые неотрицательные числа. Скажем, что таблица является симпатичной, если для всех i сумма чисел ее i -ой строки не превышает R_i и для всех j сумма чисел ее j -го столбца не превышает C_j .

Вам задана таблица Z размера $N \times M$ (N строк и M столбцов), в некоторых клетках которой уже стоят целые неотрицательные числа. Найдите симпатичную таблицу с максимальной суммой элементов такую, что она совпадает с Z на тех клетках, в которых в Z стоят числа.

Входные данные

Первая строка входного файла INPUT.TXT содержит числа N и M ($1 \leq M, N \leq 20$). Следующая строка содержит N целых неотрицательных чисел - R_1, R_2, \dots, R_N . Следующая строка содержит M целых неотрицательных чисел C_1, C_2, \dots, C_M . Все числа не превышают 10^6 . Следующие N строк содержат по M целых чисел, которые задают Z . Если на некотором месте в таблице отсутствует число, то на этом месте во входном файле стоит число -1.

Выходные данные

Выведите в выходной файл OUTPUT.TXT выведите целое число - сумму элементов найденной таблицы. Если решение не существует, то выведите единственное число -1.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	2 2	3
	2 2	
	1 2	
	1 -1	
	-1 -1	

```
/*
    Задача: 219. Симпатичные таблицы

    Решение: максимальный поток, алгоритм Диница, графы,  $O((N+M)^3)$ 

    Автор: Дмитрий Козырев, github: dmkgz, e-mail: dmkozyrev@rambler.ru
*/

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>
#include <queue>

const int INF = (int)1e9+1;

struct Graph {
```

```

int n, s, t;

std::vector<std::vector<int>> flow, cost;

std::vector<int> dist, ptr;

Graph(int n_, int s_, int t_) : n(n_), s(s_), t(t_) {
    flow.assign(n, std::vector<int>(n));
    cost.assign(n, std::vector<int>(n));
}

void add_edge(int u, int v, int c) {
    cost[u][v] = c;
}

bool bfs() {
    dist.assign(n, -1);
    std::queue<int> queue;
    queue.push(s);
    dist[s] = 0;
    while (!queue.empty()) {
        auto u = queue.front(); queue.pop();
        for (int v = 0; v < n; ++v) {
            if (dist[v] == -1 && flow[u][v] < cost[u][v]) {
                dist[v] = dist[u] + 1;
                queue.push(v);
            }
        }
    }
    return dist[t] != -1;
}

int dfs(int u, int flow_) {
    if (!flow_) return 0;
    if (u == t) return flow_;
    for (int& v = ptr[u]; v < n; ++v) {
        if (dist[v] != dist[u] + 1) continue;
        int pushed = dfs(v, std::min(flow_, cost[u][v] - flow[u][v]));
        if (pushed) {
            flow[u][v] += pushed;
            flow[v][u] -= pushed;
            return pushed;
        }
    }
    return 0;
}

int dinic() {
    int answ = 0;
    while (bfs()) {
        ptr.assign(n, 0);
        int pushed;
        while ((pushed = dfs(s, INF))) answ += pushed;
    }
    return answ;
}

};

int main() {
    int nRows, nCols;
    scanf("%d %d", &nRows, &nCols);
    std::vector<int> srow(1+nRows), scol(1+nCols);
    for (int r = 1; r <= nRows; ++r) scanf("%d", &srow[r]);
    for (int c = 1; c <= nCols; ++c) scanf("%d", &scol[c]);
    int n = nRows + nCols + 2, s = 0, t = nRows+nCols+1, answ = 0;
    Graph graph(n, s, t);
    for (int r = 1; r <= nRows; ++r) {
        for (int c = 1; c <= nCols; ++c) {
            int val; scanf("%d", &val);

```

```

        if (val == -1) {
            graph.add_edge(r, nRows + c, INF);
        } else {
            srow[r] -= val;
            scol[c] -= val;
            answ += val;
            if (srow[r] < 0 || scol[c] < 0) {
                printf("-1"); return 0;
            }
        }
    }
}

for (int r = 1; r <= nRows; ++r) {
    graph.add_edge(s, r, srow[r]);
}
for (int c = 1; c <= nCols; ++c) {
    graph.add_edge(c+nRows, t, scol[c]);
}
printf("%d", answ+graph.dinic());
return 0;
}

```

ЗАДАЧА №223

Анаграммер

(Время: 1 сек. Память: 16 Мб Сложность: 75%)

Анаграммер — специальное устройство для получения из слова его анаграмм (то есть слов, записанных теми же буквами, но в другом порядке). Это устройство умеет выполнять 2 операции:

- 1. Взять очередную букву исходного слова и поместить ее в стек.
- 2. Взять букву из стека и добавить ее в конец выходного слова.

Стек — это хранилище данных, которое работает по принципу "первый пришел — последний ушел". Стек можно представить себе в виде пирамидки. Когда мы добавляем букву в стек, это соответствует тому, что на стержень пирамидки сверху мы надеваем кольцо, на котором написана соответствующая буква. Когда берем букву из стека, то это соответствует тому, что мы снимаем со стержня верхнее кольцо, и смотрим, какая буква на нем написана.

Например, слово TROT в слово TORT может быть преобразовано анаграммером двумя различными последовательностями операций: 11112222 или 12112212.

Напишите программу, которая по двум заданным словам вычисляет количество различных последовательностей операций анаграммера, которые преобразуют первое из этих слов во второе.

Входные данные

Первая строка входного файла INPUT.TXT содержит исходное слово, а вторая — слово, которое необходимо получить. Слова состоят только из заглавных английских букв и имеют длину от 1 до 50 символов. Оба слова имеют одинаковую длину. В этих строках не содержится пробелов.

Выходные данные

В первой строке выходного файла OUTPUT.TXT должно содержаться количество последовательностей операций анаграммера, с помощью которых можно преобразовать первое слово во второе.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	TROT TORT	2
2	MADAM ADAMM	4
3	LONG GONG	0
4	AAAAAAAAA AAAAAAAAA	1430

```
#include <stdio.h>
#include <algorithm>
#include <string>
#include <vector>
```

```

#include <cassert>
#include <functional>
#include <iostream>
#include <iomanip>

typedef __float128 Int;

std::ostream& operator<<(std::ostream& os, __float128 number) {
    auto high = (long long)(number / 1e15L);
    auto low = (long long)(number - __float128(high) * 1e15L);
    if (high == 0) {
        return os << low;
    } else {
        return os << high << std::setw(15) << std::setfill('0') << low << std::setfill(' ');
    }
}

Int solve(const std::string& a, const std::string& b) {
    const int n = a.size();
    assert(n == (int)b.size());

    static Int count[1+50][1+50][1+50]; // count[len][pos1][pos2]

    // Начальная инициализация:
    for (int len = 0; len <= n; ++len) {
        for (int p1 = 0; p1 <= n; ++p1) {
            for (int p2 = 0; p2 <= n; ++p2) {
                if (len == 0) {
                    count[len][p1][p2] = 1;
                } else if (len == 1) {
                    count[len][p1][p2] = (p1 == n || p2 == n || a[p1] == b[p2]);
                } else {
                    count[len][p1][p2] = -1;
                }
            }
        }
    }

    // Ленивое ДП:
    std::function<Int(int,int,int)> get = [&](const int len, const int p1, const int p2) {
        if (count[len][p1][p2] == -1) {
            Int answ = 0;
            for (int pos = p2; pos < p2 + len; ++pos) { // a[p1] --> b[pos]
                if (a[p1] == b[pos]) {
                    const int len1 = pos-p2;
                    const int len2 = p2+len-1-pos;
                    // a[ p1+1, ..., p1+len1] --> b[ p2, p2+len1-1]
                    // a[p1+1+len1, ..., p1+len-1] --> b[pos+1, pos+1+len2]
                    answ += get(len1, p1+1, p2) * get(len2, p1+len1+1, pos+1);
                }
            }
            count[len][p1][p2] = answ;
        }
        return count[len][p1][p2];
    };

    return get(n, 0, 0);
}

int main() {
    char buf[1+50];
    scanf("%50s", buf);
    std::string a(buf);
    scanf("%50s", buf);
    std::string b(buf);
    std::cout << solve(a, b);
    return 0;
}

```

ЗАДАЧА №1389

Печатная схема

(Время: 1 сек. Память: 16 Мб Сложность: 75%)

Печатной схемой называется плоская поверхность содержащей узлы и перемычки, соединяющие пары узлов. Мы будем рассматривать печатные схемы специального вида, где все узлы расположены в узлах прямоугольной сетки, а перемычки (вертикальные или горизонтальные) соединяют пары соседних узлов. Печатная схема называется связной, если любые два узла соединены друг с другом последовательностью перемычек. На вход дается печатная схема, где некоторые соседние узлы уже соединены перемычками. К ней необходимо добавить некоторое количество перемычек таким образом, чтобы схема стала связной. Стоимость вертикальной перемычки – 1, а горизонтальной – 2.

Ваша программа должна по начальной печатной схеме определить количество добавляемых перемычек, минимальную стоимость такого добавления, а также вывести сами добавляемые перемычки.

Входные данные

Первая строка входного файла INPUT.TXT содержит два натуральных числа N и M – количество строк и количество столбцов соответственно ($1 \leq N, M \leq 100$). Каждый узел определяется своими координатами, нумерация начинается с верхнего левого угла (координаты $(1, 1)$). Далее дается описание решетки в виде N строк по M чисел. Каждое число обозначает связь узла (i, j) с узлами $(i+1, j)$ и $(i, j+1)$ в следующем формате:

0 – узел (i, j) не соединен ни с одним из узлов $(i+1, j)$ и $(i, j+1)$

1 – узел (i, j) соединен только с узлом $(i+1, j)$

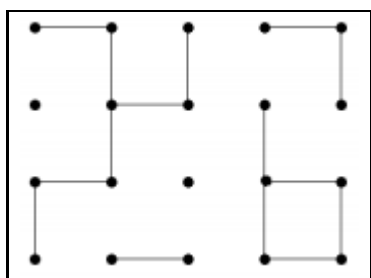
2 – узел (i, j) соединен только с узлом $(i, j+1)$

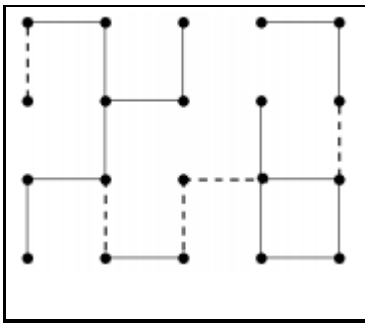
3 – узел (i, j) соединен и с узлом $(i+1, j)$, и с узлом $(i, j+1)$.

Выходные данные

Первая строка выходного файла OUTPUT.TXT должна содержать два числа K и V – количество добавленных перемычек и стоимость добавления соответственно. Каждая из следующих K строк должна содержать описание добавленной перемычки в формате i, j, d , где (i, j) – координаты начального узла, а d может принимать значения 1 или 2. $d=1$ обозначает, что соединены узлы (i, j) и $(i+1, j)$, $d=2$ – узлы (i, j) и $(i, j+1)$. Если решений несколько – выведите любое из них.

Пример





№	INPUT.TXT	OUTPUT.TXT
1	4 5 2 1 1 2 1 0 3 0 1 0 3 0 0 3 1 0 2 0 2 0	5 6 1 1 1 2 5 1 3 2 1 3 3 1 3 3 2

```

/*
    Задача: 1389. Печатная схема

    Решение: DSU, алгоритм Краскала, минимальное остовное дерево,  $O(n * m * \log(n * m))$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <stdio.h>
#include <vector>
#include <algorithm>
#include <cassert>

struct Edge { int u, v, cost; };

bool operator<(const Edge& a, const Edge& b) {
    if (a.cost < b.cost) return true;
    if (a.cost > b.cost) return false;
    return a.u < b.u || (a.u == b.u && a.v < b.v);
}

struct DSU {
    std::vector<int> parent, size;
    DSU(int n) : parent(1+n), size(1+n,1) {
        for (int i = 1; i <= n; ++i) parent[i] = i;
    }
    int get_parent(int u) {
        return parent[u] == u ? u : parent[u] = get_parent(parent[u]);
    }
    void union_sets(int u, int v) {
        u = get_parent(u), v = get_parent(v);
        if (u != v) {
            if (size[u] < size[v]) std::swap(u,v);
            size[u] += size[v];
            parent[v] = u;
        }
    }
};

void int2pos(int number, int& r, int& c, int nRows, int nCols) {
    assert(1 <= number && number <= nRows * nCols);
    r = (number+nCols-1) / nCols;
    c = number - (r-1) * nCols;
    assert(1 <= r && r <= nRows);
    assert(1 <= c && c <= nCols);
    assert(number == (r-1) * nCols + c);
}

void pos2int(int& number, int r, int c, int nRows, int nCols) {

```

```

    assert(1 <= r && r <= nRows);
    assert(1 <= c && c <= nCols);
    number = (r-1) * nCols + c;
    assert(1 <= number && number <= nRows * nCols);
    assert(r == (number+nCols-1) / nCols);
    assert(c == number - (r-1) * nCols);
}

int main() {
    int nRows, nCols;
    scanf("%d %d", &nRows, &nCols);
    DSU dsu(nRows*nCols);
    std::vector<Edge> edges;
    for (int r = 1; r <= nRows; ++r) {
        for (int c = 1; c <= nCols; ++c) {
            int u, val; pos2int(u,r,c,nRows,nCols);
            scanf("%d", &val);
            if ((val >> 0) & 1) {
                assert(r+1 <= nRows);
                int v; pos2int(v,r+1,c,nRows,nCols);
                dsu.union_sets(u,v);
            }
            if ((val >> 1) & 1) {
                assert(c+1 <= nCols);
                int v; pos2int(v,r,c+1,nRows,nCols);
                dsu.union_sets(u,v);
            }
            if (r+1 <= nRows) {
                int v; pos2int(v,r+1,c,nRows,nCols);
                edges.push_back(Edge{u,v,1});
            }
            if (c+1 <= nCols) {
                int v; pos2int(v,r,c+1,nRows,nCols);
                edges.push_back(Edge{u,v,2});
            }
        }
    }
    std::sort(edges.begin(), edges.end());
    int sum = 0; std::vector<Edge> answ;
    for (auto& e : edges) {
        int pu = dsu.get_parent(e.u), pv = dsu.get_parent(e.v);
        if (pu != pv) {
            sum += e.cost;
            answ.push_back(e);
            dsu.union_sets(e.u,e.v);
        }
    }
    printf("%d %d\n", (int)answ.size(), sum);
    for (auto& e : answ) {
        int r1, c1; int2pos(e.u,r1,c1,nRows,nCols);
        int r2, c2; int2pos(e.v,r2,c2,nRows,nCols);
        printf("%d %d", r1, c1);
        if (c2 != c1) printf(" 2\n");
        if (r2 != r1) printf(" 1\n");
    }
    return 0;
}

```


ЗАДАЧА №1463

Повышение квалификации

(Время: 3 сек. Память: 128 Мб Сложность: 75%)

Взаимодействие сотрудников в некоторой компании организовано в виде иерархической структуры. Всего в компании работают n сотрудников. Каждому сотруднику присвоен уникальный номер от 1 до n , директору присвоен номер 1. У каждого сотрудника, кроме директора, есть ровно один непосредственный начальник. Непосредственный начальник сотрудника i имеет номер p_i , причем $p_i < i$.

Сотрудник x является подчиненным уровня 1 сотрудника u , если $p_x = u$. Для $k > 1$ сотрудник x является подчиненным уровня k сотрудника u , если сотрудник p_x является подчиненным уровня $k - 1$ сотрудника u .

У директора компании появилась возможность направить некоторых сотрудников на курсы повышения квалификации. Для этого он решил выбрать два числа L и R и направить на курсы всех сотрудников с номерами i , такими что $L \leq i \leq R$.

Перед тем, как выбрать числа L и R , директор получил m пожеланий от сотрудников компании, j -е пожелание задается двумя числами u_j и k_j и означает, что сотрудник u_j просит отправить на курсы одного из своих подчиненных уровня k_j . Для экономии средств директор хочет выбрать такие L и R , чтобы количество сотрудников, направленных на повышение квалификации, было минимальным возможным, но при этом все пожелания были выполнены.

Требуется написать программу, которая по заданным в компании отношениям начальник-подчиненный и пожеланиям сотрудников определяет такие числа L и R , что если отправить на курсы повышения квалификации всех сотрудников с номерами от L до R включительно, то все пожелания будут выполнены, а количество сотрудников, направленных на повышение квалификации, будет минимальным возможным. Если оптимальных пар чисел L, R будет несколько, требуется найти ту из них, в которой значение L минимально.

Входные данные

Первая строка входного файла INPUT.TXT содержит число n – количество сотрудников компании ($2 \leq n \leq 200\,000$). Вторая строка содержит $(n - 1)$ чисел: p_2, p_3, \dots, p_n ($1 \leq p_i < i$) – номера непосредственных начальников сотрудников. Третья строка содержит число m ($1 \leq m \leq 200\,000$) – количество пожеланий от сотрудников. Последующие m строк задают пожелания сотрудников и содержат по два целых числа u_j, k_j ($1 \leq u_j < n, 1 \leq k_j < n$, гарантируется, что у сотрудника u_j есть хотя бы один подчиненный уровня k_j).

Выходные данные

В выходной файл OUTPUT.TXT выведите два искоемых числа: L и R . Если оптимальных пар (L, R) несколько, требуется вывести ту, в которой значение L минимально.

Пример

Пояснение к примеру

На повышение квалификации будут направлены сотрудники с номерами 3, 4, 5 и 6. Сотрудник с

номером 3 является подчиненным уровня 1 сотрудника с номером 1, сотрудник с номером 4 – подчиненным уровня 2 сотрудника с номером 1, а сотрудник с номером 6 – подчиненным уровня 1 сотрудника с номером 3.

№	INPUT.TXT	OUTPUT.TXT
1	7 1 1 2 2 3 3 3 1 1 3 1 1 2	3 6

```

/*
    Задача: 1463. Повышение квалификации
    Решение: dfs, bfs, euler tour, binary search, std::set, sorting, queries offline, two p
*/
#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
#define isz(x) (int)(x).size()
using vi = std::vector<int>
using vvi = std::vector<vi>
const auto io_sync_off = []() {std::ios_base::sync_with_stdio(0);std::cin.tie(0);return 0;}()
int main() {
    // Input:
    int nVert; std::cin >> nVert;
    vvi next(1+nVert);
    for (int u = 2, p; u <= nVert; u++) {
        std::cin >> p;
        next[p].push_back(u);
    }
    // dfs:
    vi tin(1+nVert), tout(1+nVert), depth(1+nVert);
    int timer = 0;
    std::function<void(int)> dfs = [&](int u) {
        tin[u] = timer++;
        for (int v : next[u]) {
            depth[v] = depth[u] + 1;
            dfs(v);
        }
        tout[u] = timer++;
    };
    dfs(1);
    // bfs:
    vi arr_tin, arr_depth, arr_vert;
    std::queue<int> queue;
    queue.push(1);
    while (isz(queue)) {
        int u = queue.front(); queue.pop();
        arr_tin.push_back(tin[u]);
        arr_depth.push_back(depth[u]);
        arr_vert.push_back(u);
        for (int v : next[u]) {
            queue.push(v);
        }
    }
    // Queries:
    int nQ; std::cin >> nQ;
    struct Query { int left, right; };
    std::vector<Query> queries(nQ);
    for (int i = 0; i < nQ; i++) {
        int u, d; std::cin >> u >> d;
        d += depth[u];
        auto begin = arr_tin.begin() + int(std::lower_bound(all(arr_depth), d) - arr_depth.
        auto after = arr_tin.begin() + int(std::upper_bound(all(arr_depth), d) - arr_depth.

```

```

        begin = std::lower_bound(begin, after, tin[u]);
        after = std::lower_bound(begin, after, tout[u]);
        queries[i] = Query{int(begin-arr_tin.begin()), int(after-arr_tin.begin())-1};
    }
    std::sort(all(queries), [](auto a, auto b){return a.right-a.left < b.right-b.left;});
    // Processing queries:
    vi color(1+nVert);
    int nColors = 0;
    std::set<int> set;
    for (auto &q : queries) {
        auto rt = set.upper_bound(q.right);
        auto lt = set.lower_bound(q.left);
        if (rt == lt) {
            nColors++;
            set.insert(q.left);
            set.insert(q.right);
            for (int i = q.left; i <= q.right; i++) {
                const int vert = arr_vert[i];
                assert(color[vert] == 0);
                color[vert] = nColors;
            }
        }
    }
    // Two pointers
    vi count(1+nColors, 0); count[0]++;
    int lt = 1, rt = 0, left = 1, right = nVert, used = 1;
    while (lt <= nVert) {
        while (rt + 1 <= nVert && used != isz(count)) {
            const int c = color[++rt];
            used += (++count[c] == 1);
        }
        if (used == isz(count)) {
            if (right - left > rt - lt) {
                left = lt;
                right = rt;
            }
        }
        const int c = color[lt++];
        used -= (--count[c] == 0);
    }
    std::cout << left << ' ' << right << std::endl;
    return 0;
}

```

ЗАДАЧА №1158

Циклические сдвиги

(Время: 5 сек. Память: 16 Мб Сложность: 76%)

Циклическим сдвигом строки s называется строка $s_{k+1} s_{k+2} \dots s_n s_1 s_2 \dots s_k$ для некоторого k ($0 \leq k < n$), где n – длина строки s .

Для заданной строки требуется построить все ее циклические сдвиги, упорядочить их лексикографически, выписать последний столбец и найти в данном списке позицию исходной строки.

Входные данные

В единственной строке входного файла INPUT.TXT записана строка, состоящая из символов с кодами ASCII от 33 до 127. Длина строки не превышает 10^5 .

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите номер позиции исходного слова в отсортированном списке циклических сдвигов. Если таких позиций несколько, то следует вывести позицию с наименьшим номером. Во второй строке выведите последний столбец таблицы циклических сдвигов.

Пример

Пояснение к примеру

Первоначально выпишем все 6 циклических сдвигов строки «abraka» и упорядочим получившийся список лексикографически:

Теперь видно, что в упорядоченном списке исходное слово находится на 2-й позиции, а последний столбец представляет собой строку «karaab».

№	INPUT.TXT	OUTPUT.TXT
1	abraka	2 karaab
Исходный список		Упорядоченный список
abraka		aabra k
brakaa		<u>ab</u> ra k <u>a</u>
rakaab		akaab r
akaabr		braka a
kaabra		kaabr a
aabrak		rakaa b

/*
Решение влоб:

- Отсортируем подстроки длины n , при этом необходимо сохранить исходный порядок.

- Сравнение двух подстрок выполним за $O(\log(n))$ при помощи бин. поиска + двойного п

- Используем то, что до первого различия строки совпадают - условие бин. поиска. На

Параметры хэширования:

- первый модуль - простое случайное число порядка $1e9$

- второй модуль 2^{64}

Сдвиг представляем как целое число - позиция начала подстроки.

Для выполнимости устойчивости сортировки в случае совпадения подстрок необходимо сравни

Итого, решение имеет асимптотику $O(n \cdot \log(n)^2)$, если сортировка реализована за $O(n \cdot \log($
*/

```
#include <stdio.h>
#include <algorithm>
#include <string>
#include <vector>
#include <cassert>
#include <ctime>
#include <cstdlib>

bool is_prime(int n) {
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            return false;
        }
    }
    return n > 1;
}

int next_prime(int number, int steps = 1) {
    while (steps--) {
        while (!is_prime(++number));
    }
    return number;
}

typedef unsigned long long ull;

int main() {
    std::srand(std::time(0));

    // Константы хэширования:
    const int mod1 = next_prime(1e9, std::rand() % 77 + 33);
    const int base = next_prime(256, std::rand() % 77 + 33);

    // Предподсчет степеней основания base по модулям mod1 и mod2:
    const int mxPow = 2e5;
    std::vector<int> pow1(mxPow+1, 1);
    std::vector<ull> pow2(mxPow+1, 1);
    for (int i = 1; i <= mxPow; ++i) {
        pow1[i] = 1LL * pow1[i-1] * base % mod1;
        pow2[i] = pow2[i-1] * base;
    }

    // Читаем входные данные и строим полиномиальный хэш на префиксе:
    char buf[1+1000000];
    scanf("%1000000s", buf);
    std::string s(buf);
    const int n = s.size();
    s += s;
    std::vector<int> pref1{0};
    std::vector<ull> pref2{0};
    for (int i = 0; i < (int)s.size(); ++i) {
        pref1.push_back((pref1.back() + 1LL * s[i] * pow1[i]) % mod1);
        pref2.push_back(pref2.back() + s[i] * pow2[i]);
    }
}
```

```

}
std::vector<int> shifts(n);
for (int i = 0; i < n; ++i) {
    shifts[i] = i;
}
// Сортировка циклических сдвигов за  $O(n \cdot \log(n)^2)$ , используя полиномиальные хэши:
std::stable_sort(shifts.begin(), shifts.end(), [&](const int p1, const int p2) {
    if (s[p1] < s[p2]) {
        return true;
    } else if (s[p1] > s[p2]) {
        return false;
    }
    int low = 0, high = n;
    while (high - low > 1) {
        const int mid = (low + high) / 2;
        const ull hash2_a = (pref2[p1+mid+1] - pref2[p1]) * pow2[mxPow-(p1+mid)];
        const ull hash2_b = (pref2[p2+mid+1] - pref2[p2]) * pow2[mxPow-(p2+mid)];
        const int hash1_a = (0LL + pref1[p1+mid+1] - pref1[p1] + mod1) * pow1[mxPow-(p1)];
        const int hash1_b = (0LL + pref1[p2+mid+1] - pref1[p2] + mod1) * pow1[mxPow-(p2)];
        if (hash1_a == hash1_b && hash2_a == hash2_b) {
            low = mid;
        } else {
            high = mid;
        }
    }
    if (low+1 == n) {
        return p1 < p2;
    }
    return s[p1+low+1] < s[p2+low+1];
});

printf("%d\n", int(std::find(shifts.begin(), shifts.end(), 0) - shifts.begin()) + 1);
std::string answer;
for (auto p : shifts) {
    answer.push_back(s[p+n-1]);
}
printf("%s", answer.c_str());
return 0;
}

```

ЗАДАЧА №573

НЛО

(Время: 1 сек. Память: 16 Мб Сложность: 77%)

В маленьком городке М начала действовать служба контроля за незаконными полетами НЛО. Первая задача службы - выяснить, сколько НЛО действует в окрестности города.

Агенты службы опросили множество свидетелей и составили список случаев встречи с НЛО, произошедших за одни сутки, с указанием места и времени наблюдения.

Теперь аналитики хотят понять, сколько же на самом деле было НЛО. Из данных разведки известна максимальная скорость, с которой может лететь НЛО. Аналитики просят вас узнать, какое минимальное количество НЛО могли наблюдать свидетели.

Входные данные

На первой строке входного файла INPUT.TXT содержатся целые числа n и v - количество случаев наблюдения и максимальная скорость НЛО ($1 \leq n \leq 100$, $1 \leq v \leq 10000$). Следующие n строк содержат описания случаев встречи с НЛО в формате «ЧЧ:ММ х у», где ЧЧ:ММ – время встречи, x и y - координаты места, в котором наблюдался НЛО (для простоты будем считать, что все встречи происходили на плоскости). Координаты целые и по модулю не превышают 1000. Скорость выражена в км/ч, координаты - в км. Гарантируется, что во входных данных нет совпадающих записей.

Выходные данные

В выходной файл OUTPUT.TXT выведите одно число - минимальное возможное количество НЛО.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	4 1 12:00 0 0 13:10 0 1 14:00 1 0 15:00 1 1	2

```
/*
    Задача: 573. НЛО

    Решение: алгоритм Диница, графы, потоки, паросочетания,  $O(\sqrt{n} * n^2)$ 

    Автор: Дмитрий Козырев, github: dmksz, e-mail: dmkozyrev@rambler.ru
*/

#include <stdio.h>
#include <vector>
#include <queue>
#include <algorithm>

#define size(x) (int)(x).size()
#define all(x) (x).begin(), (x).end()

const int INF = (int)1e9+7;
```

```

typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

struct Node {
    int t, x, y;

    static Node read() {
        int h, m, x_, y_;
        scanf("%d:%d %d %d", &h, &m, &x_, &y_);
        return Node{h*60+m, x_, y_};
    }
};

inline bool operator<(const Node& a, const Node& b) {
    return a.t < b.t || (a.t == b.t && (a.x < b.x || (a.x == b.x && a.y < b.y)));
}

bool can_connect(const Node& a, const Node& b, const int speed) {
    int dx = b.x - a.x;
    int dy = b.y - a.y;
    int dt = b.t - a.t;
    if (dt <= 0) return false;
    // sqrt(dx*dx+dy*dy) / speed <= dt / 60
    return 3600LL * (dx*dx+dy*dy) <= 1LL * dt * dt * speed * speed;
}

struct Graph {
    int n, s, t;

    vvi cost, flow;

    vi dist, pointer;

    Graph(int n_, int s_, int t_) : n(n_), s(s_), t(t_) {
        cost.assign(n, vi(n,0));
        flow.assign(n, vi(n,0));
        dist.assign(n,-1);
        pointer.assign(n,0);
    }

    void add_edge(int u, int v, int c) {
        cost[u][v] = c;
    }

    bool bfs() {
        dist.assign(n,-1);
        dist[s] = 0;
        std::queue<int> queue;
        queue.push(s);
        while (!queue.empty()) {
            auto u = queue.front(); queue.pop();
            for (int v = 0; v < n; ++v) {
                if (dist[v] == -1 && flow[u][v] < cost[u][v]) {
                    dist[v] = dist[u]+1;
                    queue.push(v);
                }
            }
        }
        return dist[t] != -1;
    }

    int dfs(int u, int value) {
        if (!value) return 0;
        if (u == t) return value;
        for (int& v = pointer[u]; v < n; ++v) {
            if (dist[v] != dist[u]+1) continue;
            int pushed = dfs(v, std::min(value, cost[u][v] - flow[u][v]));
            if (pushed) {
                flow[u][v] += pushed;
            }
        }
    }
};

```



```

        flow[v][u] -= pushed;
        return pushed;
    }
}
return 0;
}

int dinic() {
    int value = 0, pushed = 0;
    while (bfs()) {
        pointer.assign(n, 0);
        while ((pushed = dfs(s, INF))) {
            value += pushed;
        }
    }
    return value;
}

};

int main() {
    int n, speed;
    scanf("%d %d", &n, &speed);
    std::vector<Node> a;
    for (int i = 0; i < n; ++i) {
        a.push_back(Node::read());
    }
    std::sort(a.begin(), a.end());
    // s = 0
    // first part from 1 to n
    // second part from n+1 to 2*n
    // t = 2*n+1
    Graph G(2*n+2, 0, 2*n+1);
    for (int i = 0; i < n; ++i) {
        for (int j = i+1; j < n; ++j) {
            if (can_connect(a[i], a[j], speed)) {
                G.add_edge(1+i, n+1+j, 1);
            }
        }
    }
    for (int i = 1; i <= n; ++i) G.add_edge(0, i, 1);
    for (int i = 1; i <= n; ++i) G.add_edge(i+n, 2*n+1, 1);
    int value = n-G.dinic();
    printf("%d", value);
    return 0;
}

```

ЗАДАЧА №1648

Автоматизация склада

(Время: 3 сек. Память: 64 Мб Сложность: 77%)

Компания занимается автоматизацией склада. На складе хранятся n видов товаров, пронумерованных от 1 до n , каждый вид товара хранится в своём помещении. Товар вида i хранится в помещении с номером i .

Специальный робот обслуживает запросы по получению товаров со склада. Для доступа в помещения склада робот использует специальные электронные карты. Карты у робота хранятся в специальном отсеке, из которого он может вынуть верхнюю карту. Вынутую карту робот может вернуть в отсек на любое место: на верхнюю позицию, между любыми двумя картами или на самую нижнюю позицию.

Чтобы открыть помещение, робот действует следующим образом. Он вынимает карты из отсека для их хранения и возвращает их обратно в отсек, пока на верхней позиции не окажется карта от помещения, которое ему необходимо открыть. После этого, вынув эту карту, робот использует её, чтобы открыть помещение, и затем также возвращает в отсек для хранения карт. Если суммарно роботу потребовалось вынуть из отсека x карт, включая ту, которой он в итоге открыл помещение, будем говорить, что для открытия помещения робот совершил x действий.

В начале рабочего дня роботу поступил заказ на выдачу m товаров: a_1, a_2, \dots, a_m . Робот должен выдать товары именно в этом порядке. Для этого он последовательно выполняет следующие действия: открывает помещение, в котором лежит очередной товар, берет товар, закрывает помещение и выдаёт товар клиенту. После этого робот переходит к выдаче следующего товара.

Исходно электронные карты лежат в отсеке в следующем порядке, от верхней к нижней: b_1, b_2, \dots, b_n . Для каждого помещения в отсеке лежит ровно одна карта.

Время выдачи товаров со склада зависит от того, сколько раз суммарно роботу придётся вынимать верхнюю карту из отсека для их хранения, чтобы найти карту от очередного помещения. Необходимо таким образом выбрать места, куда робот должен возвращать вынутые карты, чтобы минимизировать суммарное количество действий робота для открытия помещений.

Требуется написать программу, которая по заданным целым числам n и m , последовательности выдаваемых товаров a_1, a_2, \dots, a_m и начальному положению карт в отсеке для хранения b_1, b_2, \dots, b_n определяет, какое минимальное количество действий придётся совершить роботу, чтобы открыть все помещения в необходимом порядке. Для каждой вынутой карты необходимо также указать позицию, на которую её необходимо вернуть, чтобы добиться оптимального количества действий.

Входные данные

Первая строка входного файла INPUT.TXT содержит два целых числа n и m ($1 \leq n, m \leq 3 \times 10^5$) — количество видов товаров и количество товаров, которые необходимо выдать со склада.

Вторая строка содержит m целых чисел a_1, a_2, \dots, a_m ($1 \leq a_i \leq n$) — типы товаров, которые необходимо выдать со склада, перечисленные в том порядке, в котором это необходимо сделать.

Третья строка содержит n различных целых чисел b_1, b_2, \dots, b_n ($1 \leq b_i \leq n$) — порядок, в котором карты исходно находятся в отсеке для их хранения, перечисленные от верхней к нижней.

Выходные данные

Первая строка выходного файла OUTPUT.TXT должна содержать число k — минимальное количество действий, которое потребуется совершить роботу, чтобы выдать товары в заданном порядке.

Далее выведите k чисел. Для каждого действия робота выведите одно число: позицию, на которую ему следует вернуть вынутую карту в отсек для хранения. Если карта возвращается на самую верхнюю позицию, следует вывести 1, если после одной карты, 2, и так далее, для последней позиции следует вывести n .

Если существует несколько способов минимизировать суммарное число действий, выведите любой из них.

Примеры

Пояснения к примерам

Во втором примере карты в отсеке робота перемещаются следующим образом:

№	INPUT.TXT	OUTPUT.TXT
1	1 1 1 1	1 1
2	4 5 4 1 2 4 4 4 3 2 1	7 4 4 2 4 4 1 4
3	2 2 1 2 2 1	3 2 2 2

Действие	Перед действием	Извлеченная карта	Открытое помещение	Позиция, куда помещается карта	После действия
1	4, 3, 2, 1	4	4	4	3, 2, 1, 4
2	3, 2, 1, 4	3	—	4	2, 1, 4, 3
3	2, 1, 4, 3	2	—	2	1, 2, 4, 3
4	1, 2, 4, 3	1	1	4	2, 4, 3, 1
5	2, 4, 3, 1	2	2	4	4, 3, 1, 2
6	4, 3, 1, 2	4	4	1	4, 3, 1, 2
7	4, 3, 1, 2	4	4	4	3, 1, 2, 4

```
/*
    Задача: 1648. Автоматизация склада

    Решение: дерево отрезков, ленивое проталкивание, связный список,  $O((n+m) \cdot \log(n))$ 

    Автор: Дмитрий Козырев, github: dmkz, e-mail: dmkozyrev@rambler.ru
*/

#include <bits/stdc++.h>

namespace SegmentTreeLazy {

    /*****
    * SegmentTree<Value, Extra, Traits> - segment tree class with lazy propagation, 0-ind
    * Default operations: minimal value on segment and addition on segment for int64_t ty
    * Use Traits<Value,Extra> for definition of:
    *****/
```

```

*      1) neutral element for `Value`;
*      2) neutral element for `Extra`;
*      3) how should combine `Extra` with `Value`;
*      4) how should combine `Value` with `Value` (children to root);
*      5) how should combine `Extra` with `Extra`;
* See examples below: TraitsMinAdd<Value, Extra>
*****/

/*****
* Traits for minimal value on segment.
* Get-query: get minimal value in segment [l, r]
* Update-query: add const to each value in segment [l, r]
*****/
template<typename Value, typename Extra>
struct TraitsMinAdd {
    // Definition of neutral element for `Value`:
    static Value valueNeutral() { return std::numeric_limits<Value>::max(); }
    // Definition of neutral element for `Extra`:
    static Extra extraNeutral() { return Extra(0); }
    // Definition of how should combine `Extra` with `Value`:
    template<typename Node>
    static Value getValue(const Node& src) {
        return src.value() + src.extra();
    }
    // Definition of how should combine `Value` with `Value` (children to root):
    template<typename NodeRoot, typename NodeLt, typename NodeRt>
    static void pull(NodeRoot root, const NodeLt& lt, const NodeRt& rt) {
        root.value() = std::min(getValue(lt), getValue(rt));
    }
    // Definition of how should combine `Extra` with `Extra`:
    template<typename NodeDst, typename NodeSrc>
    static void push(NodeDst dst, const NodeSrc& src) {
        dst.extra() += src.extra();
    }
};

/*****
* Additional traits, implemented below
*****/
template<typename Value, typename Extra> struct TraitsMaxAdd;

/*****
* SegmentTree, see description above
*****/
template<typename Value = int64_t, typename Extra = int64_t, typename Traits = TraitsMi>
struct SegmentTree {

    /*****
    * Node class
    *****/
    struct Node {
        Value value;

        Extra extra;

        Node(Value value_ = Traits::valueNeutral(), Extra extra_ = Traits::extraNeutral()
            : value(value_), extra(extra_) { }

        Value getValue(int l, int r) const { return Traits::getValue(NodeWrapper<Node>(
    };

    /*****
    * NodeWrapper class
    *****/
    template<typename NodeType>
    struct NodeWrapper {
        int l, r;
        NodeType node;
        NodeWrapper(int l_, int r_, NodeType node_)

```

```

        : l(l_), r(r_), node(node_) { }
    int left() const { return l; }
    int right() const { return r; }
    int mid() const { return (l+r)/2; }
    int len() const { return r - l + 1; }
    Value& value() { return node.value; }
    Extra& extra() { return node.extra; }
    const Value& value() const { return node.value; }
    const Extra& extra() const { return node.extra; }
};

/*****
 * SegmentTree public data: n - number of items, data - vector for nodes
 *****/
int n; std::vector<Node> data;

/*****
 * Resize segment tree data to needed size
 *****/
void resize(int n_) {
    n = n_;
    data.assign(2 * n - 1, Node());
}

/*****
 * Lazy propagation from node to its children
 *****/
void push(int v, int l, int r, int m) {
    if (data[v].extra != Traits::extraNeutral()) {
        Traits::push(
            NodeWrapper<Node&>(l, m, data[v+1]),
            NodeWrapper<const Node&>(l, r, data[v])
        );
        Traits::push(
            NodeWrapper<Node&>(m+1, r, data[v+2*(m-l+1)]),
            NodeWrapper<const Node&>( l, r, data[v])
        );
        data[v].extra = Traits::extraNeutral();
    }
}

/*****
 * Update node using children values
 *****/
void pull(int v, int l, int r, int m) {
    assert(data[v].extra == Traits::extraNeutral());
    Traits::pull(
        NodeWrapper<Node&>( l, r, data[v]),
        NodeWrapper<const Node&>( l, m, data[v+1]),
        NodeWrapper<const Node&>(m+1, r, data[v+2*(m-l+1)])
    );
}

/*****
 * Build segtree from array with given values
 *****/
template<typename T>
void build(const std::vector<T>& arr, const int v, const int tl, const int tr) {
    if (tl == tr) {
        data[v] = Node(arr[tl]);
    } else {
        const int tm = (tl + tr) / 2;
        build(arr, v+1, tl, tm);
        build(arr, v+2*(tm-tl+1), tm+1, tr);
        pull(v, tl, tr, tm);
    }
}

```

```

template<typename T>
void build(const std::vector<T>& arr) {
    resize((int)arr.size());
    build(arr, 0, 0, n-1);
}

/*****
*   Get-query on range [ql, qr]
*****/
Node get(int ql, int qr, const int v, const int tl, const int tr) {
    if (ql == tl && qr == tr) {
        return data[v];
    } else {
        int tm = (tl + tr) / 2;
        push(v, tl, tr, tm);
        Node ret;
        if (qr <= tm) {
            ret = get(ql, qr, v+1, tl, tm);
        } else if (ql > tm) {
            ret = get(ql, qr, v+2*(tm-tl+1), tm+1, tr);
        } else {
            const auto lt = get(ql, tm, v+1, tl, tm);
            const auto rt = get(tm+1, qr, v+2*(tm-tl+1), tm+1, tr);
            Traits::pull(
                NodeWrapper<Node&>(ql, qr, ret),
                NodeWrapper<const Node&>(ql, tm, lt),
                NodeWrapper<const Node&>(tm+1, qr, rt)
            );
        }
        pull(v, tl, tr, tm);
        return ret;
    }
}

Value get(const int ql, const int qr) { return get(ql, qr, 0, 0, n-1).getValue(ql,

/*****
*   Update query on range [ql, qr] by extra
*****/
void update(const int ql, const int qr, const Extra& extra, const int v, const int
    if (ql == tl && tr == qr) {
        Traits::push(
            NodeWrapper<Node&>(tl, tr, data[v]),
            NodeWrapper<Node>(ql, qr, Node(Traits::valueNeutral(), extra))
        );
    } else {
        int tm = (tl + tr) / 2;
        push(v, tl, tr, tm);
        if (qr <= tm) {
            update(ql, qr, extra, v+1, tl, tm);
        } else if (ql > tm) {
            update(ql, qr, extra, v+2*(tm-tl+1), tm+1, tr);
        } else {
            update(ql, tm, extra, v+1, tl, tm);
            update(tm+1, qr, extra, v+2*(tm-tl+1), tm+1, tr);
        }
        pull(v, tl, tr, tm);
    }
}

void update(const int ql, const int qr, const Extra& extra) {
    update(ql, qr, extra, 0, 0, n-1);
}

};

/*****
*   Traits for maximal value on segment.
*   Get-query:    get maximal value in segment [l, r]
*****/

```

```

* Update-query: add const to each value in segment [l, r]
*****/
template<typename Value, typename Extra>
struct TraitsMaxAdd {
    // Definition of neutral element for `Value`:
    static Value valueNeutral() { return std::numeric_limits<Value>::min(); }
    // Definition of neutral element for `Extra`:
    static Extra extraNeutral() { return Extra(0); }
    // Definition of how should combine `Extra` with `Value`:
    template<typename Node>
    static Value getValue(const Node& src) {
        return src.value() + src.extra();
    }
    // Definition of how should combine `Value` with `Value` (children to root):
    template<typename NodeRoot, typename NodeLt, typename NodeRt>
    static void pull(NodeRoot root, const NodeLt& lt, const NodeRt& rt) {
        root.value() = std::max(getValue(lt), getValue(rt));
    }
    // Definition of how should combine `Extra` with `Extra`:
    template<typename NodeDst, typename NodeSrc>
    static void push(NodeDst dst, const NodeSrc& src) {
        dst.extra() += src.extra();
    }
};

const int INF = (int)2e8;

struct MaxPos {
    int max, pos;
    MaxPos(int max_ = -INF, int pos_ = 0) : max(max_), pos(pos_) {}
};

bool operator<(const MaxPos& a, const MaxPos& b) {
    return a.max < b.max || (a.max == b.max && a.pos > b.pos);
}

MaxPos operator+(const MaxPos& a, int extra) {
    return MaxPos(a.max+extra, a.pos);
}

/*****
* Traits for maximal value and its leftmost position on segment.
* Get-query: get maximal value in segment [l, r] and its leftmost position
* Update-query: add const to each value in segment [l, r]
*****/
template<typename Value, typename Extra>
struct TraitsMaxPosAdd {
    // Definition of neutral element for `Value`:
    static Value valueNeutral() { return Value(); }
    // Definition of neutral element for `Extra`:
    static Extra extraNeutral() { return Extra(0); }
    // Definition of how should combine `Extra` with `Value`:
    template<typename Node>
    static Value getValue(const Node& src) {
        return src.value() + src.extra();
    }
    // Definition of how should combine `Value` with `Value` (children to root):
    template<typename NodeRoot, typename NodeLt, typename NodeRt>
    static void pull(NodeRoot root, const NodeLt& lt, const NodeRt& rt) {
        root.value() = std::max(getValue(lt), getValue(rt));
    }
    // Definition of how should combine `Extra` with `Extra`:
    template<typename NodeDst, typename NodeSrc>
    static void push(NodeDst dst, const NodeSrc& src) {
        dst.extra() += src.extra();
    }
};

```

```

typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
#define all(x) (x).begin(), (x).end()

template<typename Value>
struct SqrtList {

    std::vector<std::vector<Value>> data;

    static const int gsize = 256;

    SqrtList() : data(1, std::vector<Value>{}) {}

    void split(int g) {
        if ((int)data[g].size() >= 2 * gsize) {
            data.insert(data.begin()+g+1, std::vector<Value>(data[g].begin()+gsize, data[g].begin()+2*gsize));
            data[g].resize(gsize);
        }
    }

    void mergeTo(int dst, int src) {
        if ((int)(data[dst].size()+data[src].size()) >= gsize) {
            data[dst].insert(data[dst].end(), data[src].begin(), data[src].end());
            data.erase(data.begin() + src);
        }
    }

    void merge(int g) {
        if (g+1 < (int)data.size()) { mergeTo(g,g+1); }
        if (g-1 >= 0 ) { mergeTo(g-1,g); }
    }

    bool getGroupPos(int index, int& g, int& p) const {
        g = (int)data.size()-1;
        p = (int)data[g].size();
        for (int id = 0; id < (int)data.size(); ++id) {
            if (index >= (int)data[id].size()) {
                index -= (int)data[id].size();
            } else {
                g = id;
                p = index;
                return true;
            }
        }
        return false;
    }

    const Value& get(int index) const {
        int g, p;
        bool flag = getGroupPos(index, g, p);
        assert(flag);
        return data[g][p];
    }

    Value& get(int index) {
        int g, p;
        bool flag = getGroupPos(index, g, p);
        assert(flag);
        return data[g][p];
    }

    void erase(int index) {
        int g, p;
        bool flag = getGroupPos(index, g, p);
        assert(flag);
        data[g].erase(data[g].begin()+p);
        merge(g);
    }
}

```



```

void insert(int index, Value x) {
    int g, p;
    getGroupPos(index, g, p);
    assert(0 <= g && g < (int)data.size());
    assert(0 <= p && p <= (int)data[g].size());
    data[g].insert(data[g].begin()+p, x);
    split(g);
}

};

struct Permutation {
    int front, back;
    std::vector<int> next;
    void build(const std::vector<int>& arr) {
        next.resize(arr.size());
        front = arr.front();
        back = arr.back();
        next[back] = -1;
        for (int i = 0; i + 1 < (int)arr.size(); ++i) {
            next[arr[i]] = arr[i+1];
        }
    }
    void moveFront(int after) {
        if (after == front) {
            return;
        }
        if (after == back) {
            int newFront = next[front];
            next[back] = front;
            next[front] = -1;
            back = front;
            front = newFront;
            return;
        }
        int newFront = next[front];
        next[front] = next[after];
        next[after] = front;
        front = newFront;
    }
};

vi fast(vi queries, vi queueInit) {
    const int nQueries = (int)queries.size();
    const int nItems = (int)queueInit.size();

    Permutation queue;
    queue.build(queueInit);

    vi posQueryInQueue(nQueries), posItemInQueue(nItems), answer;
    for (int i = 0; i < nItems; ++i) {
        const int item = queueInit[i];
        posItemInQueue[item] = i;
    }
    vvi posItemInQueries(nItems);
    {
        vi was(nItems);
        for (int i = 0; i < nQueries; ++i) {
            const int item = queries[i];
            posItemInQueries[item].push_back(i);
            if (was[item]) {
                posQueryInQueue[i] = -INF;
            } else {
                posQueryInQueue[i] = posItemInQueue[item];
                was[item] = 1;
            }
        }
    }
}

// Building SegmentTree

```

```

SegmentTreeLazy::SegmentTree<MaxPos,int,TraitsMaxPosAdd<MaxPos,int>> segtree;
{
    std::vector<MaxPos> temp(queries.size());
    for (int i = 0; i < (int)queries.size(); ++i) {
        temp[i] = MaxPos(posQueryInQueue[i], i);
    }
    segtree.build(temp);
}
auto getMaxPos = [&](int l, int r) {
    return segtree.get(l,r);
};
auto inc = [&](int l, int r, int x) {
    assert(0 <= l && l <= r && r < (int)posQueryInQueue.size());
    segtree.update(l,r,x);
};
auto set = [&](int p, int x) {
    assert(0 <= p && p < (int)posQueryInQueue.size());
    segtree.update(p, p, x-segtree.get(p,p).max);
};

auto moveFront = [&](int from) {
    int front = queue.front();
    auto up = std::upper_bound(all(posItemInQueries[front]), from);
    int after, right, pos;
    if (up == posItemInQueries[front].end()) {
        after = queue.back();
        pos = nItems-1;
        right = nQueries-1;
    } else {
        right = *up-1;
        assert(from <= right);
        auto maxPos = getMaxPos(from, right);
        pos = maxPos.max;
        after = queries[maxPos.pos];
    }
    answer.push_back(pos);
    queue.moveFront(after);

    inc(from, right,-1);
    if (right+1 < nQueries) { set(right+1, pos); }
};
for (int p = 0; p < nQueries; ++p) {
    int query = queries[p];
    while (queue.front != query) { moveFront(p); }
    moveFront(p);
}
return answer;
}

namespace FastIO {

void putChar(char c) {
    static const int SIZE = 1 << 16;
    static char buffer[SIZE];
    static int size = 0;
    if (size == SIZE || c == EOF) {
        fwrite(buffer, 1, size, stdout);
        size = 0;
    }
    if (c != EOF) { buffer[size++] = c; }
}

template<typename T>
void putInt(T x) {
    static char stack[25];
    int size = 0;
    bool positive = true;
    if (x < 0) { positive = false; x = -x; }
    do { stack[size++] = char('0' + x % 10); x /= 10; } while (x > 0);
}

```

```

        if (!positive) {stack[size++] = '-'; }
        while (size--) { putchar(stack[size]); }
    }

char getChar() {
    static const int SIZE = 1 << 16;
    static char buffer[SIZE];
    static int pos = 0;
    static int size = 0;
    if (pos == size) {
        size = (int)fread(buffer, 1, SIZE, stdin);
        pos = 0;
    }
    return (pos == size) ? EOF : buffer[pos++];
}

template<typename T>
T getInt() {
    char c = '?';
    while (!(c == '-' || ('0' <= c && c <= '9'))) { c = getChar(); }
    T v(0);
    bool positive = true;
    if (c == '-') { positive = false; c = getChar(); }
    while ('0' <= c && c <= '9') { (v *= 10) += (c - '0'); c = getChar(); }
    return positive ? v : -v;
}

}

int main() {
    using namespace FastIO;
    int n, q;
    n = getInt<int>();
    q = getInt<int>();
    vi queries(q), queue(n);
    for (auto &it : queries) { it = getInt<int>()-1; }
    for (auto &it : queue) { it = getInt<int>()-1; }
    vi answ = fast(queries, queue);
    putInt((int)answ.size());
    putchar('\n');
    for (auto it : answ) { putInt(it+1); putchar(' '); }
    putchar('\n');
    putchar(EOF);
    return 0;
}

```

ЗАДАЧА №884

Дорога

(Время: 3 сек. Память: 16 Мб Сложность: 85%)

В Древнем государстве Оссия было два города, между которыми была проложена дорога длиной S метров. Через каждый метр стояли столбики, на каждом из которых по некоторому принципу (этот секретный принцип был известен только древним монахам Шамбалы) было написано по букве (а алфавит там у них был английский). Однажды князь-король Василий I решил, что человек, когда он едет по этой дороге, слишком редко вспоминает о нем. Он решил это исправить. Для этого он повелел на некоторых столбиках вместо буквы написать «Здесь был Вася». По его представлению, человек, проехав любой участок дороги длиной K метров, должен обязательно хоть раз увидеть такую надпись. Иными словами, среди каждых K идущих подряд столбиков должен оказаться хоть один, на котором буква заменена на надпись. При этом, чтобы не слишком раздражать монахов (а они люди обидчивые), Василий I приказал выбрать для надписи такие столбики, чтобы среди стертых букв оказалось как можно меньше различных букв английского алфавита.

Помогите боярам выполнить приказ своего повелителя.

Входные данные

В первой строке входного файла INPUT.TXT написано одно целое число K ($1 \leq K \leq 100\,000$). Во второй строке – без пробелов написано S заглавных английских букв в той последовательности, в которой ими помечены столбики вдоль дороги. Гарантируется, что $K \leq S \leq 100\,000$.

Выходные данные

В первой строке выходного файла OUTPUT.TXT выведите N – минимальное количество различных букв английского алфавита, которые хотя бы на одном столбике придется стереть, чтобы написать «Здесь был Вася». В следующих N строках выведите те заглавные буквы английского алфавита, которые потребуется хоть раз стереть. Буквы можно выводить в любом порядке. Если ответов с минимальным N несколько, можно вывести любой из них.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	2 ABA	1 A
2	2 ABBAА	2 A B

```
/*
    Задача: 884. Дорога

    Решение: динамическое программирование, dp по подмножествам, битовые маски,  $O(2^n \cdot n)$ 

    Автор: Дмитрий Козырев, github: dmkg, e-mail: dmkozyrev@rambler.ru
*/

#pragma GCC optimize("Ofast")
#include <bits/stdc++.h>
```

```

#define all(x) (x).begin(), (x).end()
#define isz(x) (int)(x).size()
typedef uint32_t ui;
typedef std::vector<int> vi;
typedef std::pair<int,int> pii;
vi getMasks(int k, std::string s) {
    s += 'A';
    vi ret, cnt(26);
    for (int i = 0; i < k; ++i) {
        cnt[s[i]-'A']++;
    }
    for (int i = k; i < isz(s); ++i) {
        int mask = 0;
        for (int bit = 0; bit < 26; ++bit) {
            mask |= (cnt[bit] > 0) << bit;
        }
        ret.push_back(mask);
        cnt[s[i+0]-'A']++;
        cnt[s[i-k]-'A']--;
    }
    return ret;
}
vi remove_covered(vi arr, int mask) {
    int sz = 0;
    for (int i = 0; i < isz(arr); ++i) {
        if ((arr[i] & mask) == 0) {
            arr[sz++] = arr[i];
        }
    }
    arr.resize(sz);
    return arr;
}

int main() {
    std::ios_base::sync_with_stdio(false); std::cin.tie(0);
    int k; std::string s;
    while (std::cin >> k >> s) {
        vi masks = getMasks(k, s);
        const int H = 21, L = 5;
        std::vector<ui> cannot(1 << H);
        for (auto it : masks) {
            auto mask = ~it & ((1 << (H + L)) - 1);
            auto high = mask >> L;
            auto low = mask & ((1 << L) - 1);
            for (int sub = 0; sub < (1 << L); sub++) {
                if ((low & sub) == sub) {
                    cannot[high] |= ui(1) << sub;
                }
            }
        }
        for (int bit = H-1; bit >= 0; --bit) {
            for (int mask = (1 << H) - 1; mask >= 0; --mask) {
                if ((mask >> bit) & 1) {
                    cannot[mask ^ (1 << bit)] |= cannot[mask];
                }
            }
        }
        pii res(26, (1 << 26) - 1);
        for (int mask = 1; mask < (1 << 26); ++mask) {
            auto high = mask >> L;
            auto low = mask & ((1 << L) - 1);
            if ((cannot[high] >> low) & 1) { continue; }
            res = std::min(res, pii(__builtin_popcount(mask), mask));
        }
        int answ = res.second;
        std::cout << __builtin_popcount(answ) << std::endl;
        for (char c = 'A'; c <= 'Z'; c++) {
            if ((answ >> (c - 'A')) & 1) {
                std::cout << c << std::endl;
            }
        }
    }
}

```

```
        }  
    }  
}  
return 0;  
}
```