Университет ИТМО

Алгоритмы и структуры данных

Отчеты по задачам

ФИО студента: Готовко Алексей Владимирович
Направление подготовки: 09.03.04 (СППО)
Учебная группа: P32101
ФИО преподавателей:
Косяков М.С.
Тараканов Д.С.

Санкт-Петербург
2023г.

Содержание

1 Задачи Yandex Contest

1.1 A. Агроном-любитель

Условие

Ограничение времени	2 секунды
Ограничение памяти	64Mb

Городской школьник Лёша поехал на лето в деревню и занялся выращиванием цветов. Он посадил n цветков вдоль одной длинной прямой грядки, и они успешно выросли. Лёша посадил множество различных видов цветков, i-й от начала грядки цветок имеет вид a_i, где a_i — целое число, номер соответствующего вида в «Каталоге юного агронома».
Теперь Лёша хочет сделать фотографию выращенных им цветов и выложить ее в раздел «мои грядки» в социальной сети для агрономов «ВКомпосте». На фотографии будет виден отрезок из одного или нескольких высаженных подряд цветков.
Однако он заметил, что фотография смотрится не очень интересно, если на ней много одинаковых цветков подряд. Лёша решил, что если на фотографии будут видны три цветка одного вида, высаженные подряд, то его друзья — специалисты по эстетике цветочных фотографий — поставят мало лайков.
Помогите ему выбрать для фотографирования как можно более длинный участок своей грядки, на котором нет трех цветков одного вида подряд.

Формат ввода

В первой строке содержится целое число n ($1 \le n \le 200\ 000$) — количество цветов на грядке.
Во второй строке содержится n целых чисел a_i ($1 \le a_i \le 10^9$), обозначающих вид очередного цветка. Одинаковые цветки обозначаются одинаковыми числами, разные — разными.

Формат вывода

Выведите номер первого и последнего цветка на самом длинном искомом участке. Цветки нумерются от 1 до n.
Если самых длинных участков несколько, выведите участок, который начинается раньше.

Решение

Будем решать с помощью двух указателей, рассматривая подпоследовательности цветков.
Назовем *тройкой* подпоследовательность из трех цветков, а *плохой тройкой* – тройку из одинаковых цветков. Также назовем *хорошей подпоследовательностью* подпоследовательность, удовлетворяющую условию.
Левый указатель – начало текущей подпоследовательности, правый – конец. На каждой итерации считываем следующее значение и двигаем правый указатель. Нам достаточно хранить лишь последние три считанных значения, так как если до k-ой итерации в подпоследовтельности не было плохой тройки, то на $k+1$-ой итерации плохая тройка может появится лишь в последних трех элементах.
При очередной итерации мы проверяем, появилась ли в рассматриваемой подпоследовательности плохая тройка. Если таковая нашлась, то мы сравниваем длину подпоследовательности без последнего рассмотренного элемента (так как на предыдущей итерации плохой тройки не было) с длиной максимальной найденной до текущей итерации хорошей подпоследовательности и по необходимости обновляем переменные, содержащие данные для ответа (левый и правый указатели и длину максимальной хорошей подпоследовательности). Наконец перемещаем левый указатель на позицию, предшествующую текущей позиции правого указателя (в таком случае на следующей итерации мы будем рассматривать подпоследовательность из трех элементов).

Сложность: $O(n)$

Исходный код

```cpp
#include <iostream>

using namespace std;

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

    int ansLeft = 0;
    int ansRight = 0;
    int maxDist = 0;
    int curDist;

    // last three elements
    int c;
    int b = -1;
    int a = -1;

    // left and right pointers
    int left = 0;
    int right = 0;

    while (n--) {
        c = b;
        b = a;
        cin >> a;

        if (a == b && b == c) {
            curDist = right - left;
            if (curDist > maxDist) {
                maxDist = curDist;
                ansLeft = left;
                ansRight = right - 1;
            }

            left = right - 1;
        }

        right++;
    }
    right--;

    curDist = right - left + 1;
    if (curDist > maxDist) {
        ansLeft = left;
        ansRight = right;
    }

    cout << ++ansLeft << ' ' << ++ansRight << '\n';

    return 0;
}
```

1.2 B. Зоопарк Глеба

Условие

Ограничение времени	1 секунда
Ограничение памяти	256Mb

Недавно Глеб открыл зоопарк. Он решил построить его в форме круга и, естественно, обнёс забором. Глеб взял вас туда начальником охраны. Казалось бы все началось так хорошо, но именно в вашу первую смену все животные разбежались. В зоопарке n животных различных видов, также под каждый из видов есть свои ловушки. К сожалению некоторые животные враждуют с друг другом в природе (они обозначены разными буквами), а зоопарк обнесён забором и имеет форму круга. С помощью камер удалось выяснить, где находятся все животные. Умная система поддержки жизнедеятельности зоопарка уже просканировала зоопарк и вывела id всех животных и ловушек в том порядке, в котором они видны из центра зоопарка. Получилось так, что все животные и все ловушки находятся на краю зоопарка. Вы хотите понять, могут ли животные прийти в свою ловушку так, чтобы их путь не пересекался с другими. Если да, также предъявите какую-нибудь из схем поимки животных.

Формат ввода

На вход подается строчка из $2 \cdot n$ символов латинского алфавита, где маленькая буква - животное, а большая - ловушка. Размер строки не более 100 000.

Формат вывода

Требуется вывести $Impossible$, если решения не существует или $Possible$, если можно вернуть всех животных в клетки. Если можно, то для каждой ловушки в порядке обхода требуется вывести индекс животного в ней.

Решение

Очевидно, что существует отображение задачи в задачу соединения пар точек на окружности непересекающимися хордами.
Докажем вспомогательный факт.

Утверждение:
Для некоторой конфигурации точек $\Omega' = [..\alpha\alpha'..]$, где $\alpha\alpha'$ – одинаковые буквы разного регистра, стоящие рядом в последовательности, существует решение задачи тогда и только тогда, когда решение существует для конфигурации $\Omega = [....]$, отличающейся от предыдущей лишь парой $\alpha\alpha'$.
Доказательство:
$\Rightarrow$. Пусть Ω' имеет некоторое решение. Геометрически это означает, что существует способ соединить все пары точек без самопересечения хорд. Рассмотрим пару точек $\alpha\alpha'$ и хорду, их соединяющую. По построению эти точки находятся рядом на окружности, а значит на малой дуге, их соединяющей, нет ни одной другой точки. Следовательно, при любом проведении оставшихся хорд ни одна из них не будет иметь конца на малой дуге $\alpha\alpha'$, что, в силу существования решения Ω', равносильно наличию решения у конфигурации Ω.
$\Leftarrow$. Пусть Ω имеет решение. Тогда при вставке пары соседствующих точек $\alpha\alpha'$ в любую часть окружности и соединении их соответствующей хордой не возникнет пересечений, так как ни одна другая проведенная хорда в решении Ω не может иметь конца на малой дуге $\alpha\alpha'$. Таким образом, если Ω имеет некоторое решение, то решение имеет и Ω'.

На данном факте будет основан алгоритм решения задачи.
Мы будем выбрасывать из последовательности все пары $\alpha\alpha'$, пока это возможно. Доказанное Утверждение гарантирует, что такие действия не меняют решения задачи. Если к концу алгоритма у нас не осталось точек, то решение существует, и наоборот.
Будем поочередно считывать значения и класть их на стек. Также будем вести подсчет количества считанных ловушек и животных. При каждом чтении значения данные величины будут соответствовать индексу текущего животного или ловушки, что позволит нам хранить в двух других стеках индексы уже считанных, но еще не определенных в пары ловушек и животных.
Как только мы находим пару животное-ловушка ($\alpha\alpha'$), мы фиксируем его в массиве, где индекс элемента соответсвует индексу ловушки, а значение элемента – индексу животного.

Сложность: $O(n)$

Исходный код

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

using namespace std;

#define ASCII_CASE_DIFFERENCE 32


int main() {
    string line;
    cin >> line;

    size_t sz = line.size();

    stack<size_t> animalIdxStack;
    stack<size_t> trapIdxStack;
    stack<char> valueStack;

    size_t trapMapping[sz / 2 + 1];

    char cur;
    size_t animalsCnt = 0;

    for (size_t i = 0; i < sz; i++) {
        cur = line[i];

        if (islower(cur)) {
            animalIdxStack.push(++animalsCnt);
        } else {
            trapIdxStack.push(i - animalsCnt + 1);
        }

        if (valueStack.empty() || abs(valueStack.top() - cur) != ASCII_CASE_DIFFERENCE) {
            valueStack.push(cur);
        } else {
            trapMapping[trapIdxStack.top()] = animalIdxStack.top();
            trapIdxStack.pop();
            animalIdxStack.pop();
            valueStack.pop();
        }

    }

    if (valueStack.empty()) {
        cout << "Possible" << '\n';
        for (size_t i = 1; i < sz / 2 + 1; i++) {
            cout << trapMapping[i] << ' ';
        }
        cout << '\n';
        return 0;
    }

    cout << "Impossible" << '\n';
    return 0;
}

```

1.3 C. Конфигурационный файл

Условие

Ограничение времени	1 секунда
Ограничение памяти	64Mb

Вадим разрабатывает парсер конфигурационных файлов для своего проекта. Файл состоит из блоков, которые выделяются с помощью символов «{» — начало блока, и «}» — конец блока. Блоки могут вкладываться друг в друга. В один блок может быть вложено несколько других блоков.
В конфигурационном файле встречаются переменные. Каждая переменная имеет имя, которое состоит из не более чем десяти строчных букв латинского алфавита. Переменным можно присваивать числовые значения. Изначально все переменные имеют значение 0.
Присваивание нового значения записывается как $< variable >=< number >$, где $< variable >$ — имя переменной, а $< number >$ — целое число, по модулю не превосходящее 10^9. Парсер читает конфигурационный файл построчно. Как только он встречает выражение присваивания, он присваивает новое значение переменной. Это значение сохраняется до конца текущего блока, а затем восстанавливается старое значение переменной. Если в блок вложены другие блоки, то внутри тех из них, которые идут после присваивания, значение переменной также будет новым.
Кроме того, в конфигурационном файле можно присваивать переменной значение другой переменной. Это действие записывается как $< variable1 >=< variable2 >$. Прочитав такую строку, парсер присваивает текущее значение переменной $variable2$ переменной $variable1$. Как и в случае присваивания константного значения, новое значение сохраняется только до конца текущего блока. После окончания блока переменной возвращается значение, которое было перед началом блока.
Для отладки Вадим хочет напечатать присваиваемое значение для каждой строки вида
$< variable1 >=< variable2 >$. Помогите ему отладить парсер.

Формат ввода

Входные данные содержат хотя бы одну и не более 10^5 строк. Каждая строка имеет один из четырех типов:

- { — начало блока;
- } — конец блока;
- $< variable >=< number >$ — присваивание переменной значения, заданного числом;
- $< variable1 >=< variable2 >$ — присваивание одной переменной значения другой переменной. Переменные $< variable1 >$ и $< variable2 >$ могут совпадать.

Гарантируется, что ввод является корректным и соответствует описанию из условия. Ввод не содержит пробелов.

Формат вывода

Для каждой строки типа $< variable1 >=< variable2 >$ выведите значение, которое было присвоено.

Решение

Для каждой переменной мы будем хранить стек ее значений, в котором верхушка будет соответствовать текущему значению. Каждое новое значение конкретной переменной будем класть на соответствующий стек. При входе в новый блок будем создавать вектор, в который при каждом изменении значения переменной внутри этого блока будем записывать имя переменной, и класть его на еще один стек. В случае выхода из блока мы будем стирать верхушку стека для каждой переменной столько раз, сколько она встретилась в соответствующем векторе. После прохождения по всему вектору будем стирать его из стека векторов. Эти действия, очевидно, вернут каждой переменной значение, которое она имела до входа в блок.

Сложность: $O(n)$, где n – количество строк.

Исходный код

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

using namespace std;


stack<int> *mapGet(unordered_map<string, stack<int>> *map, string *key) {
    return &((*map)[*key]);
}


pair<string, string> parseLine(string *line) {
    pair<string, string> parsed;

    size_t pos = line->find('=');
    if (pos != string::npos) {
        parsed.first = line->substr(0, pos);
        parsed.second = line->substr(pos + 1, line->length());
        return parsed;
    }

    parsed.first = *line;
    parsed.second = string();
    return parsed;
}


int main() {
    unordered_map<string, stack<int>> valuesMap;

    stack<vector<string>> changedVarsVectorsStack;
    changedVarsVectorsStack.push(vector<string>());

    string line;
    char ch;
    int newVal;
    pair<string, string> parsed;
    stack<int> *curStack;

    while (getline(cin, line) && !line.empty()) {
        parsed = parseLine(&line);
        string var1 = parsed.first;

        if (var1 == "{") {
            changedVarsVectorsStack.push(vector<string>());
            continue;
        }

        if (var1 == "}") {
            for (const string &var: changedVarsVectorsStack.top()) {
                valuesMap[var].pop();
            }
            changedVarsVectorsStack.pop();
            continue;
        }

        changedVarsVectorsStack.top().push_back(var1);

        string var2 = parsed.second;
```

```
        ch = var2[0];
        if (ch == '-' || isdigit(ch)) {
            newVal = stoi(var2);
            valuesMap[var1].push(newVal);
            continue;
        }

        curStack = mapGet(&valuesMap, &var2);
        newVal = curStack->empty() ? 0 : curStack->top();
        valuesMap[var1].push(newVal);

        cout << newVal << '\n';
    }

    return 0;
}
```

1.4 D. Профессор Хаос

Условие

Ограничение времени	1 секунда
Ограничение памяти	64Mb

В секретной лаборатории профессора Хаоса проходит эксперимент по выращиванию особо опасных бактерий. В начале первого дня эксперимента у Хаоса имеется a особо опасных бактерий.
Каждый день эксперимента устроен следующим образом. Рано утром профессор достает из контейнера все свои бактерии и помещает их в инкубатор, где бактерии начинают делиться. Вместо каждой бактерии образуется b новых бактерий.
После извлечения бактерий из инкубатора c из них используются для проведения различных опытов и затем уничтожаются. Если после извлечения из инкубатора имеется менее c бактерий, для проведения опытов используются все имеющиеся бактерии, и эксперимент заканчивается.
Оставшиеся бактерии в конце дня необходимо поместить в контейнер и продолжить использовать в эксперименте. Однако в контейнер можно поместить не более d бактерий, поэтому если число оставшихся бактерий больше d, то в контейнер помещаются d бактерий, а остальные уничтожаются.
Теперь профессор Хаос хочет выяснить, сколько особо опасных бактерий будет у него в контейнере после k-го дня эксперимента. Помогите ему найти ответ на этот вопрос.

Формат ввода

В единственной строке входного файла содержится пять целых чисел a, b, c, d и k ($1 \le a, b \le 1000$, $0 \le c \le 1000$, $1 \le d \le 1000$, $a \le d$, $1 \le k \le 10^{18}$).

Формат вывода

Выведите одно число — количество бактерий у Хаоса к концу k-го дня. Если эксперимент завершится в k-й день или ранее, выведите число 0.

Решение

Заметим, что количество доступных бактерий в начале следующего дня a_{n+1} задается соотношением $a_{n+1} = a_n \cdot b - c$, где a_n – количество бактерий в начале текущего дня. Это соотношение знакопостоянно (так как $a, b, c, d = const$ и $a_i >= 0$ для всех $i \in \{1, ..., k\}$), а значит у нас есть три возможных варианта развития событий:

1. $a_{n+1} < a_n$. В этом случае функция убывает, а значит ее значение в некоторый день x станет равно нулю. Нам остается лишь проверить, верно ли $k < x$.

2. $a_{n+1} = a_n$. В этом случае функция – константа, а значит в любой день, в частости в k-ый, ее значение будет равно a.

3. $a_{n+1} > a_n$. В этом случае функция возрастает, и, так как она ограничена сверху значением d, нам остается либо "упереться"в потолок в некоторый день $x < k$, либо, в противном случае, посчитать ее значение в k-ый день.

Так как количество бактерий каждый день меняется на целое число и мы имеем ограничения $1 \le a, d \le 1000$, то ни при каких обстоятельствах мы не сделаем больше 1000 итераций: даже если каждый день мы будем добавлять ровно одну бактерию (что, на самом деле, невозможно в силу целостности параметров), то не более чем за 999 дней мы достигнем потолка в $d \le 1000$ бактерий; если же каждый день отнимать одну бактерию, то мы достигнем нуля не более чем за 1000 дней.

Сложность: $O(d)$ (но при данных ограничениях – $O(1000) \equiv O(1)$).

Исходный код

```cpp
#include <iostream>

using namespace std;


int main() {
    int a, b, c, d;
    long k;

    cin >> a >> b >> c >> d >> k;

    if (a * b - c == a) {
        cout << a << '\n';
        return 0;
    }

    while (k--) {
        a = a * b - c;

        if (a <= 0) {
            cout << 0 << '\n';
            return 0;
        }

        if (a >= d) {
            cout << d << '\n';
            return 0;
        }
    }

    cout << a << '\n';
    return 0;
}

```

1.5 E. Коровы в стойла

Условие

Ограничение времени	0.1 секунды
Ограничение памяти	256Mb

На прямой расположены стойла, в которые необходимо расставить коров так, чтобы минимальное расстояние между коровами было как можно больше.

Формат ввода

В первой строке вводятся числа N ($2 < N \leq 10^5$) – количество стойл и K ($1 < K < N$) – количество коров. Во второй строке задаются N натуральных чисел в порядке возрастания – координаты стойл (координаты не превосходят 10^9).

Формат вывода

Выведите одно число – наибольшее возможное допустимое расстояние.

Решение

Понятно, что при достаточно больших входных данных и маленьких ограничениях по времени решить задачу даже оптимизированным перебором не получится. Поэтому будем угадывать ответ с помощью бинарного поиска.
Пусть x_0 и x_{n-1} – координаты первого и последнего стойла соответственно. Тогда очевидно, что максимальное расстояние между коровами всегда будет меньше чем $x_{n-1} - x_0 + 1$. Возьмем это значение за правую границу, а ноль – за левую.
Далее для каждого предполагаемого значения ответа M будем проверять, существует ли расстановка K коров, в которой минимальное расстояние между коровами будет равно M. Для проверки достаточно жадной расстановки коров с расстоянием между соседними не меньше M.

Сложность: $O(NlogN)$.

Исходный код

```cpp
#include <iostream>
#include <vector>

using namespace std;

bool isPlaceable(vector<int> *coords, int m, int k) {
    int placedCnt = 1;
    int placedLast = (*coords)[0];

    for (int i = 1; i < coords->size(); i++) {
        if ((*coords)[i] - placedLast >= m) {
            if (++placedCnt == k) return true;
            placedLast = ((*coords)[i]);
        }
    }

    return false;
}

int main() {
    int n, k;
    cin >> n >> k;

    vector<int> coords(n);
    for (int i = 0; i < n; i++) {
        cin >> coords[i];
    }

    int left = 0;
    int right = coords[n - 1] - coords[0] + 1;
    int middle;
    while (right - left > 1) {
        middle = (right + left) / 2;
        if (isPlaceable(&coords, middle, k)) {
            left = middle;
        } else {
            right = middle;
        }
    }

    cout << left << '\n';
}

```

1.6 F. Число

Условие

Ограничение времени	1 секунда
Ограничение памяти	256Mb

Вася написал на длинной полоске бумаги большое число и решил похвастаться своему старшему брату Пете этим достижением. Но только он вышел из комнаты, чтобы позвать брата, как его сестра Катя вбежала в комнату и разрезала полоску бумаги на несколько частей. В результате на каждой части оказалось одна или несколько идущих подряд цифр.
Теперь Вася не может вспомнить, какое именно число он написал. Только помнит, что оно было очень большое. Чтобы утешить младшего брата, Петя решил выяснить, какое максимальное число могло быть написано на полоске бумаги перед разрезанием. Помогите ему!

Формат ввода

Входной файл содержит одну или более строк, каждая из которых содержит последовательность цифр. Количество строк во входном файле не превышает 100, каждая строка содержит от 1 до 100 цифр. Гарантируется, что хотя бы в одной строке первая цифра отлична от нуля.

Формат вывода

Выведите в выходной файл одну строку — максимальное число, которое могло быть написано на полоске перед разрезанием.

Решение

Очевидно, что если для любых двух строк a и b мы научимся определять, какое значение "больше" в условиях данной задачи (то есть какую из двух данных строк нужно поставить раньше другой в их конкатенации, чтобы результат был "наибольшим"), то сможем решить задачу сортировкой по убыванию.
Для сравнения любых двух строк a и b будем просто лексикографически сравнивать строки ab и ba и возвращать результат.
Для сортировки массива всех данных строк будем использовать сортировку слиянием, в среднем работающую за $O(NlogN)$, где N – количество строк.

Сложность: $O(NlogN)$.

Исходный код

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

using namespace std;

int stringCompare(string *s1, string *s2) {
    if (*s1 + *s2 > *s2 + *s1) {
        return 1;
    }
   return -1;
}

vector<string> sort_desc(vector<string> *vec, size_t left, size_t right) {
    size_t dist = right - left;
    if (dist == 0) {
        vector<string> subv{(*vec)[left]};
        return subv;
    }
    if (dist == 1) {
        vector<string> subv{(*vec)[left], (*vec)[left + 1]};
        if (stringCompare(
                &subv[0],
```

```
                &subv[1]) < 0) {
            swap(subv[0], subv[1]);
        }
        return subv;
    }

    size_t mid = (left + right) / 2;
    vector<string> subvLeft = sort_desc(vec, left, mid);
    vector<string> subvRight = sort_desc(vec, mid + 1, right);

    vector<string> sorted;
    auto leftIt = subvLeft.begin();
    auto leftEnd = subvLeft.end();
    auto rightIt = subvRight.begin();
    auto rightEnd = subvRight.end();
    while (leftIt < leftEnd && rightIt < rightEnd) {
        if (stringCompare(&(*leftIt), &(*rightIt)) > 0) {
            sorted.push_back(*leftIt);
            leftIt++;
        } else {
            sorted.push_back(*rightIt);
            rightIt++;
        }
    }

    while (leftIt < leftEnd) sorted.push_back(*(leftIt++));
    while (rightIt < rightEnd) sorted.push_back(*(rightIt++));

    return sorted;
}

int main() {
    string str;
    vector<string> v;
    while (cin >> str) {
        v.push_back(str);
    }

    vector<string> sorted = sort_desc(&v, 0, v.size() - 1);
    for (const string &s: sorted) cout << s;
    cout << '\n';

    return 0;
}
```

1.7 G. Кошмар в замке

Условие

Ограничение времени	2 секунды
Ограничение памяти	256Mb

Ходят легенды, что пока Аврора спала, ей снилось, что она ходит по разным местам: леса, поля, города и сёла. И вот однажды она наткнулась на пещеру, в которой сидел мудрец. Когда мудрец поднял на Аврору глаза, он изрёк: «Дорогая Аврора! Ты уже годами скитаешься по этим землям. Я хочу предложить тебе задачку. Вот тебе строка s. Каждая буква из алфавита имеет свой вес c_i. Вес строки, которую ты можешь получить из s многократным обменом любых двух букв, вычисляется так: для каждой буквы алфавита посчитай максимальное расстояние между позициями, в которых стоит эта буква, и перемножь его с весом этой буквы. Принеси мне строку максимально возможного веса, и я тебе расскажу, в чём смысл жизни». К счастью, когда Аврора уже шла со строкой к мудрецу, её поцеловал Филипп, и Аврора вышла из этого кошмара. Теперь вам предлагается самим окунуться в этот кошмар и решить поставленную задачу.

Формат ввода

Дана строка, состоящая из строчных букв латинского алфавита ($1 \le |s| \le 10^5$). Следующая строка ввода содержит 26 чисел — веса букв латинского алфавита от «a» до «z», веса неотрицательны и не превосходят $2^{31} - 1$.

Формат вывода

Выведите строку s, в которой переставлены буквы так, чтобы полученный вес был максимально возможным. Если искомых вариантов несколько, выведите любой из них.

Решение

Очевидно, что при возможности менять любые две буквы мы легко можем получить любую перестановку исходной строки.
Для начала заметим, что вес строки будут увеличивать только те буквы, которых в исходной строке хотя бы две. При этом для любого количества вхождений больше двух максимальный итоговый вес не будет отличаться от максимального итогового веса конфигурации с ровно двумя вхождениями.
Тогда нас интересуют только те буквы, которые встретились хотя бы два раза в данной строке. Найдем все такие буквы и отсортируем их по убыванию их веса. Далее начнем расставлять буквы следующим образом.
Пусть A – массив букв, встречающихся хотя бы два раза в исходной строке, отсортированный по убыванию их веса (в нем нет повторяющихся букв), а n – его размер. Тогда строку, которую мы ищем, можно представить в виде $A[0]A[1]...A[n-1]...A[n-1]...A[1]A[0]$. При такой конфигурации бóльшие веса будут иметь бóльшие коэффициенты умножения при подсчете веса строки. Посередине же останутся лишь те буквы, которые либо встречались ровно один раз, либо строго больше двух раз. Они никак не будут влиять на вес получившейся строки.
Для сортировки встречающихся хотя бы два раза букв по их весам будем использовать сортировку вставкой, быстро работающую при небольших размерах массива, а в условиях данной задачи – за $O(1)$, так как размер массива ограничен 26 значениями.

Сложность: $O(n)$, где n – длина строки.

Исходный код

```
#include <iostream>
#include <vector>
#include <stack>

using namespace std;

const char A_CODE = 'a';

struct letter {
    char value{};
    int weight = -1;
    int count = 0;
};

void insertionSortDesc(vector<letter> *vec, int left, int right) {
    letter key;
    int j;
    for (int i = left + 1; i <= right; i++) {
        key = (*vec)[i];
        j = i - 1;
        while (key.weight > (*vec)[j].weight && j >= 0) {
            (*vec)[j + 1] = (*vec)[j];
            j--;
        }
        (*vec)[j + 1] = key;
    }
}

int main() {
    string s;
    cin >> s;

    vector<letter> vec(26, letter{});
    for (int i = 0; i < 26; i++) {
        vec[i] = letter{
                .value = (char) (i + A_CODE),
                .weight = -1,
                .count = 0
        };
        cin >> vec[i].weight;
    }

    for (char c: s) {
        vec[c - A_CODE].count++;
    }

    insertionSortDesc(&vec, 0, 26);

    size_t sz = s.size();
    char result[sz];
    char value;
    int count;
    int i = 0;
    stack<char> leftovers;

    for (const auto ltr: vec) {
        count = ltr.count;
        if (count == 0) continue;
```

```
59
60          value = ltr.value;
61
62          if (count >= 2) {
63              result[i] = value;
64              result[sz - i - 1] = value;
65              i++;
66
67              count -= 2;
68              while (count--) leftovers.push(value);
69              continue;
70          }
71
72          leftovers.push(value);
73      }
74
75      while (!leftovers.empty()) {
76          result[i++] = leftovers.top();
77          leftovers.pop();
78      }
79
80      for (char c: result) cout << c;
81      cout << '\n';
82
83      return 0;
84  }
85
```

1.8 H. Магазин

Условие

Ограничение времени	2 секунды
Ограничение памяти	256Mb

У Билла большая семья: трое сыновей, девять внуков. И всех надо кормить. Поэтому Билл раз в неделю ходит в магазин.
Однажды Билл пришел в магазин и увидел, что в магазине проводится акция под названием «каждый k-й товар бесплатно». Изучив правила акции, Билл выяснил следующее. Пробив на кассе товары, покупатель получает чек. Пусть в чеке n товаров, тогда n/k округленное вниз самых дешевых из них достаются бесплатно.
Например, если в чеке пять товаров за 200, 100, 1000, 400 и 100 рублей, соответственно, и $k = 2$, то бесплатно достаются оба товара по 100 рублей, всего покупатель должен заплатить 1600 рублей.
Билл уже выбрал товары и направился к кассе, когда сообразил, что товары, которые он хочет купить, можно разбить на несколько чеков и благодаря этому потратить меньше денег.
Помогите Биллу выяснить, какую минимальную сумму он сможет заплатить за выбранные товары, возможно разбив их на несколько чеков.

Формат ввода

Первая строка входного файла содержит два целых числа n, k ($1 \le n \le 100\,000$, $2 \le k \le 100$) — количество товаров, которые хочет купит Билл, и параметр акции «каждый k-й товар бесплатно».
Следующая строка содержит n целых чисел a_i ($1 \le a_i \le 10\,000$) — цены товаров, которые покупает Билл.

Формат вывода

Выведите единственное число – минимальную сумму, которую Билл может заплатить за выбранные товары

Решение

Рассмотрим число m – стоимость произвольного товара в корзине. Для того, чтобы этот товар достался бесплатно, необходимо, чтобы вместе с ним в чеке было пробито еще $k-1$ товаров дороже данного. Тогда максимальное значение m, которое мы можем получить, будет равно стоимости k-ого по убыванию цены товара в корзине.
Значит, если мы упорядочим массив стоимостей товаров по убыванию (назовем такой массив A), то максимум суммы стоимости всех товаров, полученных бесплатно, будет равен

$$\sum_{i=1}^{i \cdot k \le n} A[i \cdot k].$$

Массив отсортируем с помощью сортировки вставкой.

Сложность: $O(nlogn)$.

Исходный код

```cpp
#include <iostream>

using namespace std;

void insertonSortDesc(int arr[], int left, int right) {
    int key;
    int j;
    for (int i = left + 1; i <= right; i++) {
        key = arr[i];
        j = i - 1;
        while (key > arr[j] && j >= 0) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int n;
    int k;

    cin >> n >> k;
    int arr[n];
    int s = 0;

    for (int i = 0; i < n; i++) {
        cin >> arr[i];
        s += arr[i];
    }

    insertonSortDesc(arr, 0, n - 1);
    for (int i = k - 1; i < n; i += k) {
        s -= arr[i];
    }

    cout << s << '\n';
    return 0;
}

```

1.9 I. Машинки

Условие

Ограничение времени	1 секунда
Ограничение памяти	256Mb

Петя, которому три года, очень любит играть с машинками. Всего у Пети N различных машинок, которые хранятся на полке шкафа так высоко, что он сам не может до них дотянуться. Одновременно на полу комнаты может находиться не более K машинок. Петя играет с одной из машинок на полу и если он хочет поиграть с другой машинкой, которая также находится на полу, то дотягивается до нее сам. Если же машинка находится на полке, то он обращается за помощью к маме. Мама может достать для Пети машинку с полки и одновременно с этим поставить на полку любую машинку с пола. Мама очень хорошо знает своего ребенка и может предугадать последовательность, в которой Петя захочет играть с машинками. При этом, чтобы не мешать Петиной игре, она хочет совершить как можно меньше операций по подъему машинки с пола, каждый раз правильно выбирая машинку, которую следует убрать на полку. Ваша задача состоит в том, чтобы определить минимальное количество операций. Перед тем, как Петя начал играть, все машинки стоят на полке.

Формат ввода

В первой строке содержаться три числа N, K и P ($1 \leq K, N \leq 100\ 000$, $1 \leq P \leq 500\ 000$). В следующих P строках записаны номера машинок в том порядке, в котором Петя захочет играть с ними.

Формат вывода

Выведите единственное число: минимальное количество операций, которое надо совершить Петиной маме.

Решение

Зная последовательность машинок, с которыми захочет играть Петя, для каждой из них мы можем построить очередь индексов итераций, на которых Петя захочет с ними играть. Также нам понадобятся сет машинок, которые в настоящий момент находятся на полу, и приоритетная очередь машинок, которые мы будем постепенно убирать с пола.
Для каждой машинки, находящейся на полу, ее приоритетность в очереди на установку на полку будет равна индексу следующей итерации, на которой эта машинка вновь понадобится. В случае если машинка больше не будет нужна, ее приоритетность равна INF. Таким образом, если все место на полу уже занято, но нужной машинки нет, то мы минимизируем количество операций путем установки на полку машинки, которая понадобится через наибольшее число итераций.

Сложность: $O(p \cdot logn)$.

Исходный код

```
#include <iostream>
#include <queue>
#include <unordered_set>

#define INF 2.14748364e9

using namespace std;

int queue_getAndPop(queue<int> *q) {
    int val = q->front();
    q->pop();
    return val;
}

pair<int, int> priority_queue_getAndPop(priority_queue<pair<int, int>> *pq) {
    auto val = pq->top();
    pq->pop();
    return val;
}

int main() {
    int n, k, p;
    cin >> n >> k >> p;

    queue<int> carSequence;

    // key is a car id, value is a sequence of iteration indices when this car is needed
    queue<int> carsIterIndices[n];

    int curCar;
    for (int i = 0; i < p; i++) {
        cin >> curCar;
        curCar--;
        carSequence.push(curCar);
        carsIterIndices[curCar].push(i);
    }

    // pairs of <idx, car> where car is present on the floor
    // and idx is an index of the next iteration when this car is needed
    priority_queue<pair<int, int>> carsPriorities;

    // set of cars that are currently present on the floor
    unordered_set<int> carsOnFloor;

    int cnt = 0;
    while (p--) {
        curCar = queue_getAndPop(&carSequence);
        carsIterIndices[curCar].pop();

        if (carsOnFloor.find(curCar) == carsOnFloor.end()) {

            cnt++;

            if (carsOnFloor.size() == k) {
                carsOnFloor.erase(
                        priority_queue_getAndPop(&carsPriorities).second
                );
            }
```

```
59
60              carsOnFloor.insert(curCar);
61          }
62
63          carsPriorities.emplace(
64              carsIterIndices[curCar].empty() ? INF : carsIterIndices[curCar].front(),
65              curCar
66          );
67      }
68
69      cout << cnt << '\n';
70
71      return 0;
72  }
73
```

1.10 J. Гоблины и очереди

Условие

Ограничение времени	0.6 секунды
Ограничение памяти	256Mb

Гоблины Мглистых гор очень любях ходить к своим шаманам. Так как гоблинов много, к шаманам часто образуются очень длинные очереди. А поскольку много гоблинов в одном месте быстро образуют шумную толку, которая мешает шаманам проводить сложные медицинские манипуляции, последние решили установить некоторые правила касательно порядка в очереди.
Обычные гоблины при посещении шаманов должны вставать в конец очереди. Привилегированные же гоблины, знающие особый пароль, встают ровно в ее середину, причем при нечетной длине очереди они встают сразу за центром.
Так как гоблины также широко известны своим непочтительным отношением ко всяческим правилам и законам, шаманы попросили вас написать программу, которая бы отслеживала порядок гоблинов в очереди.

Формат ввода

В первой строке входных данный записано число N ($1 \leq N \leq 10^5$) – количество запросов. Следующие N строк содержат описание запросов в формате:

- $+\ i$ – гоблин с номером i ($1 \leq i \leq N$) встаёт в конец очереди;
- $*\ i$ – привилегированный гоблин с номером i встает в середину очереди;
- $-$ – первый гоблин из очереди уходит к шаманам (гарантируется, что на момент такого запроса очередь не пуста).

Формат вывода

Для каждого запроса типа '$-$' программа должна вывести номер гоблина, который должен зайти к шаманам.

Решение

Для решения задачи создадим особую структуру данных – очередь, которая будем поддерживать указатель на центральный элемент.
Очевидно, что после добавлении гоблина в конец или удаления гоблина из начала очереди индекс центрального элемента i_{mid} будет менятся следующим образом:

$$if\ (queue.size()\ mod\ 2\ !=\ 0)\ i_{mid}\ =\ (i_{mid}\ +\ 1)\ mod\ queue.size(),$$

что дает нам сложность вставки в центр $O(1)$. Аналогично, сложности вставки в конец и удаления из начала – $O(1)$.

Сложность: $O(N)$.

Исходный код

```
#include <iostream>

#define uint unsigned int

using namespace std;

struct goblinNode {
    uint idx;
    goblinNode *next;

    goblinNode() = default;
};

goblinNode *getGoblinNode(uint idx) {
    auto *node = new goblinNode();
    node->idx = idx;
    node->next = nullptr;
    return node;
}

goblinNode *getRootNode() {
    return getGoblinNode(0);
}

struct goblinQueue {
    goblinNode rootNode = *getRootNode();
    goblinNode *end = &rootNode;
    goblinNode *middle = &rootNode;
    bool isLengthEven = true;
};

void goblinQueue_updateMiddle(goblinQueue *queue) {
    queue->isLengthEven = !queue->isLengthEven;
    if (!queue->isLengthEven) {
        queue->middle = queue->middle->next;
    }
}

void goblinQueue_pushBack(goblinQueue *queue, uint idx) {
    auto node = getGoblinNode(idx);
    queue->end->next = node;
    queue->end = queue->end->next;
    goblinQueue_updateMiddle(queue);
}

void goblinQueue_insertMiddle(goblinQueue *queue, uint idx) {
    bool midIsEnd = queue->middle == queue->end;

    auto node = getGoblinNode(idx);
    node->next = queue->middle->next;
    queue->middle->next = node;

    if (midIsEnd) queue->end = queue->end->next;
    goblinQueue_updateMiddle(queue);
}

goblinNode goblinQueue_popFirst(goblinQueue *queue) {
    goblinNode node = *queue->rootNode.next;
```

```
59      queue->rootNode.next = node.next;
60
61      if (queue->rootNode.next == nullptr) {
62          queue->end = &queue->rootNode;
63          queue->middle = &queue->rootNode;
64          queue->isLengthEven = true;
65      } else {
66          goblinQueue_updateMiddle(queue);
67      }
68
69      return node;
70  }
71
72  int main() {
73      uint n;
74      cin >> n;
75
76      char opCode;
77      goblinQueue queue;
78      while (n--) {
79          cin >> opCode;
80          uint idx;
81          switch (opCode) {
82              case '+':
83                  cin >> idx;
84                  goblinQueue_pushBack(&queue, idx);
85                  break;
86              case '*':
87                  cin >> idx;
88                  goblinQueue_insertMiddle(&queue, idx);
89                  break;
90              case '-':
91                  cout << goblinQueue_popFirst(&queue).idx << '\n';
92              default:
93                  break;
94          }
95      }
96
97      return 0;
98  }
99
```

1.11 K. Менеджер памяти-1

Условие

Ограничение времени	1 секунда
Ограничение памяти	188Gb

Пете поручили написать менеджер памяти для новой стандартной библиотеки языка varphi++. В распоряжении у менеджера находится массив из N последовательных ячеек памяти, пронумерованных от 1 до N. Задача менеджера – обрабатывать запросы приложений на выделение и освобождение памяти. Запрос на выделение памяти имеет один параметр K. Такой запрос означает, что приложение просит выделить ему K последовательных ячеек памяти. Если в распоряжении менеджера есть хотя бы один свободный блок из K последовательных ячеек, то он обязан в ответ на запрос выделить такой блок. При этом непосредственно перед самой первой ячейкой памяти выделяемого блока не должно располагаться свободной ячейки памяти. После этого выделенные ячейки становятся занятыми и не могут быть использованы для выделения памяти, пока не будут освобождены. Если блока из K последовательных свободных ячеек нет, то запрос отклоняется. Запрос на освобождение памяти имеет один параметр T. Такой запрос означает, что менеджер должен освободить память, выделенную ранее при обработке запроса с порядковым номером T. Запросы нумеруются, начиная с единицы. Гарантируется, что запрос с номером T – запрос на выделение, причем к нему еще не применялось освобождение памяти. Освобожденные ячейки могут снова быть использованы для выделения памяти. Если запрос с номером T был отклонен, то текущий запрос на освобождение памяти игнорируется. Требуется написать менеджер памяти, удовлетворяющий приведенным критериям.

Формат ввода

Первая строка входного файла содержит числа N и M – количество ячеек памяти и количество запросов соответственно ($1 \le N \le 2^{31} - 1$; $1 \le M \le 10^5$). Каждая из следующих M строк содержит по одному числу: $(i + 1)$-я строка входного файла ($1 \le i \le M$) содержит либо положительное число K, если i-й запрос – запрос на выделение с параметром K ($1 \le K \le N$), либо отрицательное число $-T$, если i-й запрос – запрос на освобождение с параметром T ($1 \le T < i$).

Формат вывода

Для каждого запроса на выделение памяти выведите в выходной файл результат обработки этого запроса: для успешных запросов выведите номер первой ячейки памяти в выделенном блоке, для отклоненных запросов выведите число -1. Результаты нужно выводить в порядке следования запросов во входном файле.

Решение

Для каждого свободного блока памяти с индексом начала idx и размером $size$ мы будем хранить записи в двух структура – sorted map A и sorted multimap B, где A содержит значение $size$ по ключу idx, а B – значение idx по ключу $size$. Также в начале каждой итерации мы будем гарантировать, что A не содержит пары записей x и y, где $x.idx < y.idx$ и $x.idx + x.size = y.idx$.
Так как все значения в A и B отсортированы по ключам, то за $O(log(map.size()))$ мы можем выяснить, возможно ли выделить память при очередном запросе.
В случае освобождения памяти мы будем освобождать нужный блок и проверять, есть ли слева и справа от него свободные блоки. Так как мы гарантируем, что с каждой стороны может быть не более одного свободного блока, нам остается лишь объединить освобожденный блок с его свободными соседями.

Сложность: $O(M \cdot logN)$.

Исходный код

```
#include <iostream>
#include <map>

#define uint unsigned int

using namespace std;

void addBlock(pair<uint, uint> idxSizePair,
              map<uint, uint> *blocksFreeIdx,
              multimap<uint, uint> *blocksFreeSize) {

    blocksFreeIdx->insert(idxSizePair);
    blocksFreeSize->insert({idxSizePair.second, idxSizePair.first});
}

void removeBlockSizeIdx(multimap<uint, uint>::iterator *sizeIdxIter,
                        map<uint, uint> *blocksFreeIdx,
                        multimap<uint, uint> *blocksFreeSize) {

    blocksFreeIdx->erase((*sizeIdxIter)->second);
    blocksFreeSize->erase(*sizeIdxIter);
}

void removeBlockIdxSize(map<uint, uint>::iterator *idxSizeIter,
                        map<uint, uint> *blocksFreeIdx,
                        multimap<uint, uint> *blocksFreeSize) {

    auto sameSizeIt = blocksFreeSize->find((*idxSizeIter)->second);
    while (sameSizeIt->second != (*idxSizeIter)->first) sameSizeIt = next(sameSizeIt);
    blocksFreeIdx->erase(*idxSizeIter);
    blocksFreeSize->erase(sameSizeIt);
}


int main() {
    map<uint, uint> blocksFreeIdx;
    multimap<uint, uint> blocksFreeSize;

    uint memSize, requestCnt;
    cin >> memSize >> requestCnt;

    // stores pairs of size requested and memory index
    pair<uint, uint> requests[requestCnt + 1];
    for (uint i = 0; i < requestCnt + 1; i++) requests[i] = {0, 0};

    addBlock({1, memSize}, &blocksFreeIdx, &blocksFreeSize);

    int req;
    uint beginIdx, size;
    pair<uint, uint> idxSizePair;
    multimap<uint, uint>::iterator iterBySize;
    map<uint, uint>::iterator iterByIdx, iterLeft, iterRight;
    for (uint reqIdx = 1; reqIdx <= requestCnt; reqIdx++) {
        cin >> req;

        if (req > 0) {
            iterBySize = blocksFreeSize.lower_bound(req);
            if (iterBySize == blocksFreeSize.end()) {
```

```
                cout << -1 << '\n';
                continue;
            }

            beginIdx = iterBySize->second;
            size = iterBySize->first;

            removeBlockSizeIdx(&iterBySize, &blocksFreeIdx, &blocksFreeSize);

            if (req < size) {
                idxSizePair = make_pair(beginIdx + req, size - req);
                addBlock(idxSizePair, &blocksFreeIdx, &blocksFreeSize);
            }

            requests[reqIdx] = {beginIdx, req};
            cout << beginIdx << '\n';
            continue;
        }

        req = abs(req);
        idxSizePair = requests[req];
        if (idxSizePair.first == 0 || idxSizePair.second == 0) continue;

        beginIdx = idxSizePair.first;
        size = idxSizePair.second;

        iterRight = blocksFreeIdx.upper_bound(beginIdx);
        iterLeft = iterRight == blocksFreeIdx.begin() ? blocksFreeIdx.end() : prev(iterRight);

        // merge right if possible
        if (iterRight != blocksFreeIdx.end() && iterRight->first == beginIdx + size) {
            size += iterRight->second;
            removeBlockIdxSize(&iterRight, &blocksFreeIdx, &blocksFreeSize);
        }

        // merge left if possible
        if (iterLeft != blocksFreeIdx.end() && iterLeft->first + iterLeft->second == beginIdx) {
            beginIdx = iterLeft->first;
            size += iterLeft->second;
            removeBlockIdxSize(&iterLeft, &blocksFreeIdx, &blocksFreeSize);
        }

        addBlock({beginIdx, size}, &blocksFreeIdx, &blocksFreeSize);
        requests[reqIdx] = {0, 0};
    }


    return 0;
}
```

1.12 L. Минимум на отрезке

Условие

Ограничение времени	0.5 секунды
Ограничение памяти	256Mb

Рассмотрим последовательность целых чисел длины N. По ней с шагом 1 двигается «окно» длины K, то есть сначала в «окне» видно первые K чисел, на следующем шаге в «окне» уже будут находиться K чисел, начиная со второго, и так далее до конца последовательности. Требуется для каждого положения «окна» определить минимум в нём.

Формат ввода

В первой строке входных данных содержатся два числа N и K ($1 \leq N \leq 150\,000$, $1 \leq K \leq 10\,000$, $K \leq N$) — длины последовательности и «окна», соответственно. На следующей строке находятся N чисел – сама последовательность. Числа последовательности не превосходят по модулю 10^5.

Формат вывода

Выходые данные должны содержать $N - K + 1$ строк – минимумы для каждого положения «окна».

Решение

Построим A – массив минимумов и их индексов по суффиксам следующим образом.

- При добавлении элемента a_r справа (сдвиг окна) удалим из A все значения, большие a_r, и добавим в конец пару (r, a_r).
- Удаление элемента a_l слева будет происходить только если расстояние между первым и последним элементами A достигло K.

Тогда, очевидно, на j-ой итерации ($j > k$) значение $A[0].second$ будет являться минимумом на текущем отрезке.

Сложность: $O(N)$.

Исходный код

```cpp
#include <iostream>
#include <deque>

using namespace std;

int main() {
    int n, k;
    cin >> n >> k;

    // pairs of <index, value>
    deque<pair<int, int>> suffixMinDeque;

    int val;
    for (int i = 0; i < n; i++) {
        if (
                !suffixMinDeque.empty()
                && i - suffixMinDeque.front().first == k
        ) suffixMinDeque.pop_front();

        cin >> val;
        while (
                !suffixMinDeque.empty()
                && suffixMinDeque.back().second >= val
        ) suffixMinDeque.pop_back();

        suffixMinDeque.emplace_back(i, val);

        if (i >= k - 1) cout << suffixMinDeque.front().second << ' ';
    }
    cout << '\n';

    return 0;
}

```

1.13 M. Цивилизация

Условие

Ограничение времени	1.5 секунды
Ограничение памяти	256Mb

Карта мира в компьютерной игре «Цивилизация» версии 1 представляет собой прямоугольник, разбитый на квадратики. Каждый квадратик может иметь один из нескольких возможных рельефов, для простоты ограничимся тремя видами рельефов — поле, лес и вода. Поселенец перемещается по карте, при этом на перемещение в клетку, занятую полем, необходима одна единица времени, на перемещение в лес — две единицы времени, а перемещаться в клетку с водой нельзя.
У вас есть один поселенец, вы определили место, где нужно построить город, чтобы как можно скорее завладеть всем миром. Найдите маршрут переселенца, приводящий его в место строительства города, требующий минимального времени. На каждом ходе переселенец может перемещаться в клетку, имеющую общую сторону с той клеткой, где он сейчас находится.

Формат ввода

Во входном файле записаны два натуральных числа N и M, не превосходящих 1000 — размеры карты мира (N — число строк в карте, M — число столбцов). Затем заданы координаты начального положения поселенца x и y, где x — номер строки, y — номер стролбца на карте ($1 \le x \le N, 1 \le y \le M$), строки нумеруются сверху вниз, столбцы — слева направо. Затем аналогично задаются координаты клетки, куда необходимо привести поселенца.
Далее идет описание карты мира в виде N строк, каждая из которых содержит M символов. Каждый символ может быть либо «.» (точка), обозначающим поле, либо «W», обозначающим лес, либо «#», обозначающим воду. Гарантируется, что начальная и конечная клетки пути переселенца не являются водой.

Формат вывода

В первой строке выходного файла выведите количество единиц времени, необходимое для перемещения поселенца (перемещение в клетку с полем занимает 1 единицу времени, перемещение в клетку с лесом — 2 единицы времени). Во второй строке выходного файла выведите последовательность символов, задающих маршрут переселенца. Каждый символ должен быть одним из четырех следующих: «N» (движение вверх), «E» (движение вправо), «S» (движение вниз), «W» (движение влево). Если таких маршрутов несколько, выведите любой из них.
Если дойти из начальной клетки в конечную невозможно, выведите число -1.

Решение

Для решения воспользуемся алгоритмом Дейкстры. На каждой итерации будем выбирать клетку A, имеющую наименьшее посчитанное расстояние до начальной позиции.
Очевидно, что для клетки с координатами (x, y) ее соседи будут иметь координаты
$(x+1, y), (x-1, y), (x, y+1), (x, y-1)$, что позволит нам быстро их находить.
Для клетки A затем рассмотрим всех ее соседей и отсортируем их по возрастанию стоимости перемещения в них. Каждого соседа B, для которого расстояние до начальной позиции обновится в текущей итерации, мы поставим в приоритетную очередь, как клетку, через которую потенциально будет проходить оптимальный путь.
Также при обновлении расстояния до исходной клетки мы будем записывать в клетку с обновляемым значением идентификатор клетки, из которой мы вызвали это обновление, что впоследствии позволит нам восстановить последовательность перемещений в оптимальном пути.

Сложность: $O(E \cdot \log V) = \Big[p = N \cdot M\Big] = O(2 \cdot p \cdot \log p) = O(p \cdot \log p) = O(NM \cdot \log(NM))$

Исходный код

```
#include <iostream>
#include <queue>
#include <set>

#define INF INT32_MAX
#define coords pair<int, int>

using namespace std;

struct mapCell {
    int x;
    int y;
    int id;
    int moveCost;
    int dist;
    int pathAncestorId;
};

struct cellCompareDist {
    bool operator()(mapCell const *cell1, mapCell const *cell2) const {
        return cell1->dist > cell2->dist;
    }
};

struct cellCompareMoveCost {
    bool operator()(mapCell const *cell1, mapCell const *cell2) const {
        return cell1->moveCost < cell2->moveCost;
    }
};

int getEnteringEdgeWeight(const char *cellType) {
    switch (*cellType) {
        case '.':
            return 1;
        case 'W':
            return 2;
        default:
            return 0;
    }
}

void pathfindShortest(mapCell cellArray[], int &n, int &m, int &startId, int &endId) {
    cellArray[startId].dist = 0;
    cellArray[startId].moveCost = 0;

    priority_queue<mapCell *, vector<mapCell *>, cellCompareDist> cellProcessQueue;
    cellProcessQueue.push(&cellArray[startId]);

    multiset<mapCell *, cellCompareMoveCost> reachableNeighbours;

    while (!cellProcessQueue.empty()) {
        auto curCell = cellProcessQueue.top();
        cellProcessQueue.pop();

        if (curCell->id == endId) return;

        reachableNeighbours.clear();
        if (curCell->x < n && cellArray[curCell->id + m].moveCost)
```

```
            reachableNeighbours.insert(&cellArray[curCell->id + m]);
        if (curCell->x > 1 && cellArray[curCell->id - m].moveCost)
            reachableNeighbours.insert(&cellArray[curCell->id - m]);
        if (curCell->y < m && cellArray[curCell->id + 1].moveCost)
            reachableNeighbours.insert(&cellArray[curCell->id + 1]);
        if (curCell->y < m && cellArray[curCell->id + 1].moveCost)
            reachableNeighbours.insert(&cellArray[curCell->id + 1]);
        if (curCell->y > 1 && cellArray[curCell->id - 1].moveCost)
            reachableNeighbours.insert(&cellArray[curCell->id - 1]);

        for (auto neighbour: reachableNeighbours) {

            if (!neighbour->moveCost) continue;

            int newDist = curCell->dist + neighbour->moveCost;

            if (neighbour->dist > newDist) {
                neighbour->dist = newDist;
                neighbour->pathAncestorId = curCell->id;
                cellProcessQueue.push(neighbour);
            }
        }
    }
}


int main() {
    int n, m;
    coords startPoint, endPoint;
    cin >> n >> m >> startPoint.first >> startPoint.second >> endPoint.first >> endPoint.second;

    int startId = (startPoint.first - 1) * m + startPoint.second;
    int endId = (endPoint.first - 1) * m + endPoint.second;

    int cellCnt = n * m;
    mapCell cellArray[cellCnt + 1];

    char cellType;
    for (int i = 0; i < cellCnt; i++) {
        cin >> cellType;
        cellArray[i + 1].x = i / m + 1;
        cellArray[i + 1].y = i % m + 1;
        cellArray[i + 1].id = i + 1;
        cellArray[i + 1].moveCost = getEnteringEdgeWeight(&cellType);
        cellArray[i + 1].dist = INF;
        cellArray[i + 1].pathAncestorId = 0;
    }

    pathfindShortest(cellArray, n, m, startId, endId);

    auto cell = cellArray[endId];
    if (!cell.pathAncestorId) {
        cout << -1 << '\n';
        return 0;
    }

    cout << cell.dist << '\n';
    string path;
    while (cell.id != startId) {
        auto ancestor = cellArray[cell.pathAncestorId];
        int diff = ancestor.id - cell.id;
```

```
        if (diff == 1)
            path += 'W';
        else if (diff == -1)
            path += 'E';
        else if (diff == m)
            path += 'N';
        else if (diff == -m)
            path += 'S';

        cell = ancestor;
    }

    for (auto it = path.end() - 1; it >= path.begin(); it--) cout << *it;
    cout << '\n';

    return 0;
}
```

1.14 N. Свинки-копилки

Условие

Ограничение времени	1 секунда
Ограничение памяти	256Mb

У Васи есть n свинок-копилок, свинки занумерованы числами от 1 до n. Каждая копилка может быть открыта единственным соответствующим ей ключом или разбита.
Вася положил ключи в некоторые из копилок (он помнит, какой ключ лежит в какой из копилок). Теперь Вася собрался купить машину, а для этого ему нужно достать деньги из всех копилок. При этом он хочет разбить как можно меньшее количество копилок (ведь ему еще нужно копить деньги на квартиру, дачу, вертолет...). Помогите Васе определить, какое минимальное количество копилок нужно разбить.

Формат ввода

В первой строке содержится число n — количество свинок-копилок ($1 \leq n \leq 100$). Далее идет n строк с описанием того, где лежит ключ от какой копилки: в i-й из этих строк записан номер копилки, в которой находится ключ от i-й копилки.

Формат вывода

Выведите единственное число: минимальное количество копилок, которые необходимо разбить.

Решение

Построим ориентированный граф, где вершина – копилка, а ориентированное ребро из A в B означает, что ключ от B лежит в A.
Заметим, что так как от каждой копилки существует максимум один ключ, то ни в одну вершину не может входить больше одного ребра.
Мы будем хранить массив вершин, а для каждой вершины – были ли мы в ней или нет. Далее будем проходить по массиву и рекурсивно запускать поиск в глубину следующим образом:

- при первом вызове `dfs` на конкретную вершину передадим параметр `traverseParent = true`;
- на каждой итерации будем запускать `dfs` на всех детей (для этих запусков `traverseParent = false`), а в случае `traverseParent = true` – еще и на родителя, если таковой имеется (в этом случае передаем далее `traverseParent = true`).

Таким образом мы сможем пройти всю компоненту связности, даже если компонента – дерево, а поиск был начат не с корня.

Сложность: $O(V + E) = \Big[E \leq V\Big] = O(n + n) = O(n)$

Исходный код

```cpp
#include <iostream>
#include <vector>

using namespace std;

struct node {
    bool isVisited = false;
    vector<ushort> children;
    ushort parent = 0;
};

void traverse(node graph[], ushort v, bool traverseParent) {

    if (graph[v].isVisited) return;

    graph[v].isVisited = true;

    if (traverseParent && graph[v].parent) traverse(graph, graph[v].parent, true);

    for (auto &i: graph[v].children) {
        if (!graph[i].isVisited) traverse(graph, i, false);
    }

}

int main() {
    ushort n;
    cin >> n;

    node graph[n + 1];

    ushort idx;
    for (ushort i = 1; i <= n; i++) {
        cin >> idx;
        graph[idx].isVisited = false;
        graph[idx].children.push_back(i);
        graph[i].parent = idx;
    }

    ushort componentCnt = 0;
    for (ushort i = 1; i <= n; i++) {
        if (!graph[i].isVisited) {
            traverse(graph, i, true);
            componentCnt++;
        }
    }

    cout << componentCnt << '\n';

    return 0;
}

```

1.15 O. Долой списывание!

Условие

Ограничение времени	2 секунды
Ограничение памяти	64Mb

Во время теста Михаил Дмитриевич заметил, что некоторые лкшата обмениваются записками. Сначала он хотел поставить им всем двойки, но в тот день Михаил Дмитриевич был добрым, а потому решил разделить лкшат на две группы: списывающих и дающих списывать, и поставить двойки только первым. У Михаила Дмитриевича записаны все пары лкшат, обменявшихся записками. Требуется определить, сможет ли он разделить лкшат на две группы так, чтобы любой обмен записками осуществлялся от лкшонка одной группы лкшонку другой группы.

Формат ввода

В первой строке находятся два числа N и M — количество лкшат и количество пар лкшат, обменивающихся записками ($1 \leq N \leq 100, 0 \leq M \leq \frac{N(N-1)}{2}$). Далее в M строках расположены описания пар лкшат: два различных числа, соответствующие номерам лкшат, обменивающихся записками (нумерация лкшат идёт с 1). Каждая пара лкшат перечислена не более одного раза.

Формат вывода

Необходимо вывести ответ на задачу Павла Олеговича. Если возможно разделить лкшат на две группы, выведите «`YES`»; иначе выведите «`NO`».

Решение

Построим неориентированный граф, где вершины – ученики, а ребра – факт передачи записки от одного ученика другому.
Понятно, что нам нужно проверить, является ли получившийся граф двудольным. Известно, что граф двудолен тогда и только тогда, когда существует правильная раскраска его вершин в два цвета. Проверим, существует ли такая раскраска.
Будем проходить по каждой вершине и, если она еще не покрашена, запускать рекурсивный алгоритм покраски, основанный на `dfs`. Случай нахождения компоненты, которую не удается покрасить, означает что граф не является двудольным.

Сложность: $O(N + M)$

Исходный код

```cpp
#include <iostream>
#include <vector>

using namespace std;

bool colourGraph(pair<short, vector<ushort>> graph[], ushort v, short colour) {
    graph[v].first = colour;
    for (auto &j: graph[v].second) {
        if (graph[j].first == colour) return false;
        if (graph[j].first == 0) {
            if (!colourGraph(graph, j, -colour)) return false;
        }
    }
    return true;
}

int main() {
    ushort n;
    ushort m;
    cin >> n >> m;

    pair<short, vector<ushort>> graph[n];

    ushort a, b;
    for (ushort i = 0; i < m; i++) {
        cin >> a >> b;
        graph[--a].second.push_back(--b);
        graph[b].second.push_back(a);
    }

    for (ushort i = 0; i < n; i++) {
        if (graph[i].first) continue;
        if (!colourGraph(graph, i, 1)) {
            cout << "NO" << '\n';
            return 0;
        }
    }
    cout << "YES" << '\n';
    return 0;
}

```

1.16 P. Авиаперелёты

Условие

Ограничение времени	2 секунды
Ограничение памяти	256Mb

Главного конструктора Петю попросили разработать новую модель самолёта для компании «Air Бубундия». Оказалось, что самая сложная часть заключается в подборе оптимального размера топливного бака.
Главный картограф «Air Бубундия» Вася составил подробную карту Бубундии. На этой карте он отметил расход топлива для перелёта между каждой парой городов.
Петя хочет сделать размер бака минимально возможным, для которого самолёт сможет долететь от любого города в любой другой (возможно, с дозаправками в пути).

Формат ввода

Первая строка входного файла содержит натуральное число n ($1 \le n \le 1000$) — число городов в Бубундии. Далее идут n строк по n чисел каждая. j-е число в i-й строке равно расходу топлива при перелёте из i-го города в j-й. Все числа не меньше нуля и меньше 10^9. Гарантируется, что для любого i в i-й строчке i-е число равно нулю.

Формат вывода

Первая строка выходного файла должна содержать одно число — оптимальный размер бака.

Решение

Для решения задачи будем угадывать оптимальный объем X с помощью бинарного поиска. В качестве границ возмьем значения $\min(W)$ и $\max(W)$, где W – множество всех весов ребер.
Для каждого конкретного значения X будем строить новый граф g, в котором ребро ab существует тогда и только тогда, когда в исходном графе G вес этого ребра $w(ab) \le X$.
Очевидно, что если в g существует вершина v такая, что:

$$\forall a \in V_g \quad \exists p, p' : a \overset{p}{\rightsquigarrow} v, v \overset{p'}{\rightsquigarrow} a,$$

то g является сильно связным, а значит проверяемое значение X нам подходит.
При этом указанное выше выполняется для какой-либо вершины v_0 тогда и только тогда, когда оно также истинно $\forall v \in V_g \setminus v_0$. Занчит алгоритм проверки графа g на сильную связность мы можем запускать с произвольной вершины v.

Сложность (пусть m – максимально допустимое значение веса ребра):

$$O(n^2) + O\left(\left(n + \frac{n(n-1)}{2}\right) \cdot \log m\right) = O(n^2) + O(n \cdot \log m + n^2 \cdot \log m) = O(n^2) + O(n^2 \cdot \log m) = O(n^2 \cdot \log m)$$

Исходный код

```
#include <iostream>

#define INF SIZE_MAX

using namespace std;

void calculateAccess(size_t val, size_t n, bool **matrix, size_t **costs) {
    for (size_t i = 0; i < n; i++) {
        for (size_t j = 0; j < n; j++) {
            matrix[i][j] = costs[i][j] <= val;
        }
    }
}

void resetVisited(bool *isVisited, size_t n) {
    for (size_t i = 0; i < n; i++) isVisited[i] = false;
}

void checkConnectivity(size_t v, size_t n, bool **accessMatrix, bool *isVisited, bool reverseEdges) {
    isVisited[v] = true;
    for (size_t i = 0; i < n; i++) {
        if (i == v || isVisited[i]) continue;
        if (reverseEdges ? accessMatrix[i][v] : accessMatrix[v][i])
            checkConnectivity(i, n, accessMatrix, isVisited, reverseEdges);
    }
}

bool allWasVisited(const bool *isVisited, size_t n) {
    for (size_t i = 0; i < n; i++) {
        if (isVisited[i]) continue;
        return false;
    }
    return true;
}

int main() {
    size_t n;
    cin >> n;

    if (n == 1) {
        cin >> n; // stub second input
        cout << 0 << '\n';
        return 0;
    }

    size_t *costMatrix[n];
    bool *accessMatrix[n];

    size_t left = INF;
    size_t right = 0;

    for (size_t i = 0; i < n; i++) {
        costMatrix[i] = new size_t[n];
        accessMatrix[i] = new bool[n];

        for (size_t j = 0; j < n; j++) {
            cin >> costMatrix[i][j];
            if (i != j) {
```

```
                left = min(costMatrix[i][j], left);
                right = max(costMatrix[i][j], right);
            }
        }
    }

    bool isVisited[n];
    bool hasConnectivity;
    size_t middle;
    while (left < right) {
        middle = (left + right) / 2;

        calculateAccess(middle, n, accessMatrix, costMatrix);
        resetVisited(isVisited, n);

        checkConnectivity(0, n, accessMatrix, isVisited, false);
        if (allWasVisited(isVisited, n)) {
            resetVisited(isVisited, n);
            checkConnectivity(0, n, accessMatrix, isVisited, true);
            hasConnectivity = allWasVisited(isVisited, n);
        } else hasConnectivity = false;

        if (hasConnectivity)
            right = middle;
        else
            left = middle + 1;
    }

    cout << left << '\n';

    return 0;
}
```

2 Задачи Timus

2.1 1005. Куча камней

Условие

Ограничение времени	1 секунда
Ограничение памяти	64Mb

У вас есть несколько камней известного веса $w_1, \ldots, w_n$. Напишите программу, которая распределит камни в две кучи так, что разность весов этих двух куч будет минимальной.

Формат ввода

Ввод содержит количество камней n ($1 \leq n \leq 20$) и веса камней $w_1, \ldots, w_n$ ($1 \leq w_i \leq 100\ 000$) — целые, разделённые пробельными символами.

Формат вывода

Ваша программа должна вывести одно число — минимальную разность весов двух куч.

Решение

В силу маленького ограничения n мы можем позволить себе полный перебор всех вариантов с помощью рекурсии. На каждой итерации будем добвлять следующий камень сначала в одну кучу, а затем в другую, и запускать две новых ветки рекурсии.

Сложность: $O(2^n)$

Исходный код

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

using namespace std;


int iterate(int fstPile, int sndPile, vector<int> *v, size_t pos) {
    if (pos == v->size()) {
        return abs(fstPile - sndPile);
    }

    int curValue = (*v)[pos];
    return min(
            iterate(fstPile + curValue, sndPile, v, pos + 1),
            iterate(fstPile, sndPile + curValue, v, pos + 1)
    );
}

int main() {
    size_t n;
    cin >> n;

    vector<int> stones;
    stones.resize(n);
    for (size_t i = 0; i < n; i++) cin >> stones[i];

    cout << iterate(0, 0, &stones, 0) << '\n';
    return 0;

}

```

2.2 1296. Гиперпереход

Условие

Ограничение времени	1 секунда
Ограничение памяти	64Mb

Гиперпереход, открытый ещё в начале XXI-го века, и сейчас остаётся основным способом перемещения на расстояния до сотен тысяч парсеков. Но совсем недавно физиками было открыто новое явление. Оказывается, длительностью альфа-фазы перехода можно легко управлять. Корабль, находящийся в альфа-фазе перехода, накапливает гравитационный потенциал. Чем больше накопленный гравитационный потенциал корабля, тем меньше энергии потребуется ему на прыжок сквозь пространство. Ваша цель — написать программу, которая позволит кораблю за счёт выбора времени начала альфа-фазы и её длительности накопить максимальный гравитационный потенциал.
В самой грубой модели грави-интенсивность — это последовательность целых чисел p_i. Будем считать, что если альфа-фаза началась в момент i и закончилась в момент j, то накопленный в течение альфа-фазы потенциал — это сумма всех чисел, стоящих в последовательности на местах от i до j.

Формат ввода

В первой строке входа записано целое число N — длина последовательности, отвечающей за грави-интенсивность ($0 \le N \le 60\ 000$). Далее идут N строк, в каждой записано целое число p_i ($-30\ 000 \le p_i \le 30\ 000$).

Формат вывода

Максимальный гравитационный потенциал, который может накопить корабль в альфа-фазе прыжка. Считается, что потенциал корабля в начальный момент времени равен нулю.

Решение

Будем считывать значения p_i и добавлять к некоторой текущей сумме. На каждом шаге будем проверять, стала ли текущая сумма больше лучшей суммы, которую мы смогли накопить ранее, и по необходимости обновлять наилучшее значение. Если в некоторый момент текущая сумма стала отрицательной, то мы можем занулить ее (так как в наихудшем случае накопленный потенциал будет равен нулю) и продолжить выполнение программы.

Сложность: $O(n)$

Исходный код

```cpp
#include <iostream>

using namespace std;


int main() {
    size_t n;
    cin >> n;

    int bestSum = 0;
    int curSum = 0;

    int cur = 0;
    while (n--) {
        cin >> cur;
        curSum += cur;
        bestSum = max(bestSum, curSum);
        if (curSum < 0) {
            curSum = 0;
        }
    }

    cout << max(bestSum, curSum) << '\n';
    return 0;
}

```

2.3 1444. Накормить элефпотама

Условие

Ограничение времени	0.5 секунды
Ограничение памяти	64Mb

Гарри Поттер сдаёт экзамен по предмету «Уход за магическими существами». Его задание — накормить карликового элефпотама. Гарри помнит, что элефпотамы отличаются прямолинейностью и невозмутимостью. Они настолько прямолинейны, что ходят строго по прямой, и настолько невозмутимы, что заставить их идти можно, только если привлечь его внимание к чему-нибудь действительно вкусному. И главное, наткнувшись на цепочку своих собственных следов, элефпотам впадает в ступор и отказывается идти куда-либо. По словам Хагрида, элефпотамы обычно возвращаются домой, идя в обратную сторону по своим собственным следам. Поэтому они никогда не пересекают их, иначе могут заблудиться. Увидев свои следы, элефпотам детально вспоминает все свои перемещения от выхода из дома (поэтому-то они и ходят только по прямой и лишний раз не меняют направление — так легче запоминать). По этой информации элефпотам вычисляет, в какой стороне расположена его нора, после чего поворачивается и идет прямо к ней. Эти вычисления занимают у элефпотама некоторое (довольно большое) время. А то, что некоторые невежды принимают за ступор, на самом деле есть проявление выдающихся вычислительных способностей этого чудесного, хотя и медленно соображающего животного!
Любимое лакомство элефпотамов — слоновьи тыквы, именно они и растут на лужайке, где Гарри должен сдавать экзамен. Перед началом испытания Хагрид притащит животное к одной из тыкв. Скормив элефпотаму очередную тыкву, Гарри может направить его в сторону любой оставшейся тыквы. Чтобы сдать экзамен, надо провести элефпотама по лужайке так, чтобы тот съел как можно больше тыкв до того, как наткнется на свои следы.

Формат ввода

В первой строке входа находится число N ($3 \le N \le 30\ 000$) — количество тыкв на лужайке. Тыквы пронумерованы от 1 до N, причем номер один присвоен той тыкве, у которой будет стоять элефпотам в начале экзамена. В следующих N строках даны координаты всех тыкв по порядку. Все координаты — целые числа от -1000 до 1000. Известно, что положения всех тыкв различны, и не существует прямой, проходящей сразу через все тыквы.

Формат вывода

В первой строке выхода вы должны вывести K — максимальное количество тыкв, которое может съесть элефпотам. Далее по одному числу в строке выведите K чисел — номера тыкв в порядке их обхода. Первым в этой последовательности всегда должно быть число 1.

Решение

Возьмем координаты первой тыквы (x_0, y_0) за точку отстчета системы координат. Для каждой точки (x, y) найдем $\angle BAC$, где $B = (x, y)$, $A = (x_0, y_0)$, $C = (x, y_0)$, через $\arctan(\angle BAC)$ в треугольнике BAC. Далее скажем, что на плоскости мы будем отсчитывать углы от точки $(0, -\infty)$ против часовой стрелки. Тогда сдвинем результаты вычисления arctan на $\frac{\pi}{2}$, если точка в I-ой или IV-ой четверти, и на $\frac{3 \cdot \pi}{2}$, если точка во II-ой или III-ей четверти. Также для каждой точки вычислим расстояние до начала координат как $\sqrt{(x - x_0)^2 + (y - y_0)^2}$.
Теперь зададим правило сравнения двух точек A и B. Если $\angle A \neq \angle B$, то $A < B \Leftrightarrow \angle A < \angle B$ и наоборот. В противном случае, $A < B \Leftrightarrow dist(A) < dist(B)$ и наоборот, где $dist(X)$ – расстояние от точки X до точки (x_0, y_0).
Для того чтобы начать простраивать маршрут, по которому мы будем двигаться, нужно найти две таких соседних точки P_1 и P_2 в уже отсортированном массиве, чтобы разница их *углов* была наибольшей. В таком случае мы исключим возможное пересечение маршрута с первым пройденным отрезком (от первой точки до второй).
Работу алгоритма можно представить так.
Будем откладывать на плоскости некоторый угол α, который изначально равен $\angle P_2$. Далее мы начинаем постепенно увеличивать угол, проводя луч из центра системы координат. Как только на луч попадает точка, мы идем в нее. Если же на луч одновременно попало несколько точек, то начинаем обходить их по увеличению расстояния до центра координат. Таким образом, мы можем гарантировать обход всех точек, а также отсутствие пересечения пути в силу выбора начала обхода, описанного выше.
Остается лишь отсортировать массив точек.

Сложность: $O(NlogN)$.

Исходный код

```
#define _USE_MATH_DEFINES

#include <iostream>
#include <cmath>
#include <vector>
#include <algorithm>

using namespace std;

#define ushort unsigned short

const double PI = M_PI;
const double PI_OVER_2 = M_PI_2;
const double PI_TIMES_3_OVER_2 = 3.0 * M_PI_2;

const double EPS = 1e-10;

struct point {
    ushort idx;
    double dist;
    double angle;
};

double distance(const short *x1, const short *y1, const short *x2, const short *y2) {
    return sqrt(pow((*x1 - *x2), 2) + pow((*y1 - *y2), 2));
}

bool point_compare(const point &a, const point &b) {
    double angleDiff = a.angle - b.angle;
    if (angleDiff < -EPS) return true;
    if (angleDiff > EPS) return false;
    return a.dist < b.dist;
}

double calculateAngle(const short *x1, const short *y1, const short *x2, const short *y2) {
    if (*x2 == *x1) {
        if (*y2 > *y1) return PI;
        return 0;
    }
    if (*y2 == *y1) {
        if (*x2 < *x1) return PI_TIMES_3_OVER_2;
        return PI_OVER_2;
    }

    double shift = (*x2 > *x1) ? PI_OVER_2 : PI_TIMES_3_OVER_2;
    double t = atan(((double) (*y2 - *y1)) / ((double) (*x2 - *x1)));
    return t + shift;
}

int main() {
    ushort n;
    cin >> n;

    short startX;
    short startY;
    cin >> startX >> startY;
    ushort idx = 2;

```

```
    short x;
    short y;
    vector<point> vec;
    while (idx <= n) {
        cin >> x >> y;
        struct point p{
                .idx = idx,
                .dist = distance(&startX, &startY, &x, &y),
                .angle = calculateAngle(&startX, &startY, &x, &y)
        };
        vec.push_back(p);
        idx++;
    }

    sort(vec.begin(), vec.end(), point_compare);

    ushort begin = 0;
    double maxAngleDiff = 2.0 * PI - (vec[n - 2].angle - vec[0].angle);
    double diff;
    for (ushort i = 0; i < n - 2; i++) {
        diff = vec[i + 1].angle - vec[i].angle;
        if (diff > maxAngleDiff) {
            begin = i + 1;
            maxAngleDiff = diff;
        }
    }

    cout << n << '\n' << 1 << '\n';
    for (ushort i = begin; i < n - 1; i++) {
        cout << vec[i].idx << '\n';
    }
    for (ushort i = 0; i < begin; i++) {
        cout << vec[i].idx << '\n';
    }

    return 0;
}

```

2.4 1726. Кто ходит в гости...

Условие

Ограничение времени	1 секунда
Ограничение памяти	64Mb

Программный комитет школьных соревнований по программированию, проходящих в УрГУ — многочисленная, весёлая и дружная команда. Дружная настолько, что общения в университете им явно не хватает, поэтому они часто ходят друг к другу в гости. Все ребята в программном комитете очень спортивные и ходят только пешком.

Однажды хранитель традиций олимпиадного движения УрГУ подумал, что на пешие прогулки от дома к дому члены программного комитета тратят слишком много времени, которое могли бы вместо этого потратить на придумывание и подготовку задач. Чтобы доказать это, он решил посчитать, какое расстояние в среднем преодолевают члены комитета, когда ходят друг к другу в гости. Хранитель традиций достал карту Екатеринбурга, нашёл на ней дома всех членов программного комитета и выписал их координаты. Но координат оказалось так много, что хранитель не смог справиться с этой задачей самостоятельно и попросил вас помочь ему.

Город Екатеринбург представляет собой прямоугольник со сторонами, ориентированными по сторонам света. Все улицы города идут строго с запада на восток или с севера на юг, проходя через весь город от края до края. Дома всех членов программного комитета расположены строго на пересечении каких-то двух перпендикулярных улиц. Известно, что все члены комитета ходят только по улицам, поскольку идти по тротуару гораздо приятнее, чем по дворовым тропинкам. И, конечно, при переходе от дома к дому они всегда выбирают кратчайший путь. Программный комитет очень дружный, и все его члены ходят в гости ко всем одинаково часто.

Формат ввода

Первая строка содержит целое число n — количество членов программного комитета ($2 \le n \le 10^5$). В i-й из следующих n строк через пробел записаны целые числа x_i, y_i — координаты дома i-го члена программного комитета ($1 \le x_i, y_i \le 10^6$).

Формат вывода

Выведите среднее расстояние, которое проходит член программного комитета от своего дома до дома своего товарища, округлённое вниз до целых.

Решение

Заметим, что расстояние между двумя точками с координатами (x_0, y_0) и (x_1, y_1) будет равно $|x_0 - x_1| + |y_0 - y_1|$. Мы хотим посчитать суммарное расстояние между всеми парами домов. При этом в силу линейности выражения выше мы можем отдельно просуммировать все значения по каждой координате. Рассмотрим координаты по x в отсортированном массиве.

Искомая сумма будет равна

$$x_{n-1} \cdot (n-1) + \sum_{i=1}^{n} \sum_{j=0}^{i-1} (x_i - x_j),$$

что равносильно

$$x_{n-1} \cdot (n-1) + \sum_{i=0}^{n-1} x_i \cdot (2 \cdot i - n).$$

Аналогичные рассуждения справедливы и для координат по y.

Остается лишь исключить посчитанные несколько раз пути.

Сложность: $O(nlogn)$.

Исходный код

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

#define llong long long

using namespace std;

int main() {
    llong n;
    cin >> n;
    llong arrX[n];
    llong arrY[n];

    for (llong i = 0; i < n; i++) {
        cin >> arrX[i] >> arrY[i];
    }

    sort(arrX, arrX + n);
    sort(arrY, arrY + n);

    llong ttlSum = (arrX[n - 1] + arrY[n - 1]) * (n - 1);
    for (llong i = 0; i < n - 1; i++) {
        ttlSum += (arrX[i] + arrY[i]) * (2 * i - n + 1);
    }

    cout << ttlSum * 2 / n / (n - 1) << '\n';
    return 0;
}

```

2.5 1067. Структура папок

Условие

Ограничение времени	2 секунды
Ограничение памяти	64Mb

Хакер Билл случайно потерял всю информацию с жесткого диска своего компьютера, и у него нет резервных копий его содержимого. Но он сожалеет не о потере самих файлов, а о потере очень понятной и удобной структуры папок, которую он создавал и сохранял в течение многих лет работы. К счастью, у Билла есть несколько копий списков папок с его жесткого диска. С помощью этих списков он смог восстановить полные пути к некоторым папкам (например, «WINNT\SYSTEM32\CERTSRV\CERTCO~1\X86»). Он поместил их все в файл, записав каждый найденный путь в отдельную строку.
Напишите программу, которая восстановит структуру папок Билла и выведет ее в виде отформатированного дерева.

Формат ввода

Первая строка содержит целое число N – количество различных путей к папкам ($1 \leq N \leq 500$). Далее следуют N строк с путями к папкам. Каждый путь занимает одну строку и не содержит пробелов, в том числе, начальных и конечных. Длина каждого пути не превышает 80 символов. Каждый путь встречается в списке один раз и состоит из нескольких имен папок, разделенных обратной косой чертой («\»). Имя каждой папки состоит из $1 - 8$ заглавных букв, цифр или специальных символов из следующего списка: восклицательный знак, решетка, знак доллара, знак процента, амперсанд, апостроф, открывающаяся и закрывающаяся скобки, знак дефиса, собаки, циркумфлекс, подчеркивание, гравис, открывающаяся и закрывающаяся фигурная скобка и тильда.

Формат вывода

Выведите отформатированное дерево папок. Каждое имя папки должно быть выведено в отдельной строке, перед ним должно стоять несколько пробелов, указывающих на глубину этой папки в иерархии. Подпапки должны быть перечислены в лексикографическом порядке непосредственно после их родительской папки; перед их именем должно стоять на один пробел больше, чем перед именем их родительской папки. Папки верхнего уровня выводятся без пробелов и также должны быть перечислены в лексикографическом порядке.

Решение

Для каждой папки в иерархии будем хранить ее имя и sorted map ее потомков, где имя потомка – ключ, а сам потомок – значение. Тогда для каждого пути мы можем следовать по дереву, пока не найдем несуществующей папки, а затем добавить ее. Вывод всей иерархии с табуляцией осуществим рекурсивно.

Сложность: $O(NlogN)$.

Исходный код

```
#include <iostream>
#include <set>
#include <map>
#include <utility>
#include <vector>

using namespace std;

const string DELIMITER = "\\";

struct fileNode {
    string filename;
    map<string, fileNode *> children;

    fileNode(string filename) {
        this->filename = std::move(filename);
    }
};

vector<string> string_split(const string *s, const string *delimiter) {
    vector<string> split;
    size_t last = 0;
    size_t next = s->find(*delimiter, last);

    while (next != string::npos) {
        split.push_back(s->substr(last, next - last));
        last = ++next;
        next = s->find(*delimiter, last);
    }

    split.push_back(s->substr(last, s->size() - last));

    return split;
}

void fileNode_add(fileNode *root, vector<string> *tokens) {
    fileNode *cur = root;
    auto tokenIter = tokens->begin();
    map<string, fileNode *>::iterator mapIter;

    while (tokenIter != tokens->end()) {
        mapIter = cur->children.find(*tokenIter);
        if (mapIter != cur->children.end()) {
            cur = mapIter->second;
        } else {
            cur->children.insert({*tokenIter, new fileNode(*tokenIter)});
            cur = cur->children[*tokenIter];
        }
        tokenIter = next(tokenIter);
    }
}

void fileNode_print(fileNode *node, const string &offset) {
    for (auto const &child: node->children) {
        cout << offset << child.first << '\n';
        fileNode_print(child.second, offset + " ");
    }
}
```

```
59
60  int main() {
61      unsigned short n;
62      cin >> n;
63
64      auto root = new fileNode("");
65
66      string path;
67      vector<string> tokens;
68      while (n--) {
69          cin >> path;
70          tokens = string_split(&path, &DELIMITER);
71          fileNode_add(root, &tokens);
72      }
73
74      fileNode_print(root, "");
75
76      return 0;
77  }
78
```

2.6 1521. Военные учения 2

Условие

Ограничение времени	1 секунда
Ограничение памяти	64Mb

В соответствии с этой схемой учения делятся на N раундов, в течение которых N солдат, последовательно пронумерованных от 1 до N, маршируют друг за другом по кругу, т.е. первый следует за вторым, второй за третьим, ..., $(N-1)$-й за N-м, а N-й за первым. В каждом раунде очередной солдат выбывает из круга и идёт чистить унитазы, а оставшиеся продолжают маршировать. В очередном раунде выбывает солдат, марширующий на K позиций впереди выбывшего на предыдущем раунде. В первом раунде выбывает солдат с номером K.
Разумеется, г-н Шульман не питал никаких надежд на то, что солдаты в состоянии сами определить очерёдность выбывания из круга. «Эти неучи даже траву не могут ровно покрасить», – фыркнул он и отправился за помощью к прапорщику Шкурко.

Формат ввода

Единственная строка содержит целые числа N ($1 \le N \le 100\ 000$) и K ($1 \le K \le N$).

Формат вывода

Вывести через пробел номера солдат в порядке их выбывания из круга.

Решение

Для быстрого получения солдата на нужной позиции воспользуемся деревом отрезков. Построим его следующим образом:

- в каждом листе будем хранить пару `(1, i)`, где `i` – номер солдата;
- в каждой вершине, являющейся родителем, будем хранить количество всех ее потомков вплоть до листов.

Тогда, зная индекс k и стартовую позицию p на некоторой итерации, мы можем сделать рекурсивный запрос к дереву на поиск нужного листа (солдата). Для этого рассмотрим количество потомков обоих детей текущей вершины: если левый ребенок имеет значение хотя бы $k + p$, то искомый лист находится в левом поддереве, иначе – в правом. В последнем случае для следующего запроса в качестве нужной позиции передадим значение $k + p - m$, где m – количество детей левого поддерева. Также, переходя на новый уровень рекурсии, будем декрементировать значение текущей вершины, так как после возвращения из функции в кругу не станет одного солдата.
Такая структура дерева и запроса позволит получать нужную позицию за время $logN$. С учетом того, что всего будет сделано N запросов, получим сложность $O(NlogN)$.

Сложность: $O(NlogN)$.

Исходный код

```cpp
#include <iostream>

using namespace std;

struct treeNode {
    size_t segmentSum;
    size_t index;
};

void tree_build(treeNode *tree, size_t vertex, size_t left, size_t right) {
    if (left == right) {
        tree[vertex].segmentSum = 1;
        tree[vertex].index = left + 1;
        return;
    }

    size_t middle = (left + right) / 2;
    tree_build(tree, vertex * 2, left, middle);
    tree_build(tree, vertex * 2 + 1, middle + 1, right);
    tree[vertex].segmentSum = tree[vertex * 2].segmentSum + tree[vertex * 2 + 1].segmentSum;
}

size_t tree_slaughterMercilessly(
        treeNode *tree, size_t vertex, size_t left, size_t right, size_t position) {
    while (left != right) {
        tree[vertex].segmentSum--;

        if (tree[vertex * 2].segmentSum >= position) {
            right = (left + right) / 2;
            vertex = vertex * 2;
            continue;
        }
        position -= tree[vertex * 2].segmentSum;
        left = (left + right) / 2 + 1;
        vertex = vertex * 2 + 1;
    }
    tree[vertex].segmentSum--;
    return tree[vertex].index;
}

int main() {
    size_t n, k;
    cin >> n >> k;
    treeNode tree[4 * n];
    tree_build(tree, 1, 0, n - 1);

    size_t position = k;
    for (size_t i = 0; i < n; i++) {
        cout << tree_slaughterMercilessly(tree, 1, 1, n, position) << ' ';
        if (i == n - 1) break;

        position = (position + k - 1) % (n - i - 1);
        if (position == 0) position = n - i - 1;
    }

    cout << '\n';
    return 0;
}
```

2.7 1162. Currency Exchange

Условие

Ограничение времени	1 секунда
Ограничение памяти	64Mb

Several currency exchange points are working in our city. Let us suppose that each point specializes in two particular currencies and performs exchange operations only with these currencies. There can be several points specializing in the same pair of currencies. Each point has its own exchange rates, exchange rate of A to B is the quantity of B you get for $1A$. Also each exchange point has some commission, the sum you have to pay for your exchange operation. Commission is always collected in source currency.
For example, if you want to exchange 100 US Dollars into Russian Rubles at the exchange point, where the exchange rate is 29.75, and the commission is 0.39 you will get $(100 - 0.39) \cdot 29.75 = 2963.3975$ RUR.
You surely know that there are N different currencies you can deal with in our city. Let us assign unique integer number from 1 to N to each currency. Then each exchange point can be described with 6 numbers: integer A and B - numbers of currencies it exchanges, and real RAB, CAB, RBA and CBA - exchange rates and commissions when exchanging A to B and B to A respectively.
Nick has some money in currency S and wonders if he can somehow, after some exchange operations, increase his capital. Of course, he wants to have his money in currency S in the end. Help him to answer this difficult question. Nick must always have non-negative sum of money while making his operations.

Формат ввода

The first line contains four numbers: N - the number of currencies, M - the number of exchange points, S - the number of currency Nick has and V - the quantity of currency units he has. The following M lines contain 6 numbers each - the description of the corresponding exchange point - in specified above order. Numbers are separated by one or more spaces. $1 \le S \le N \le 100, 1 \le M \le 100$, V is real number, $0 \le V \le 10^3$.
For each point exchange rates and commissions are real, given with at most two digits after the decimal point, $10^{-2} \le \text{rate} \le 10^2$, $0 \le \text{commission} \le 10^2$.
Let us call some sequence of the exchange operations simple if no exchange point is used more than once in this sequence. You may assume that ratio of the numeric values of the sums at the end and at the beginning of any simple sequence of the exchange operations will be less than 10^4.

Формат вывода

If Nick can increase his wealth, output `YES`, in other case output `NO`.

Решение

Построим граф так, что вершины – это валюты, а ребра – пункты обмена между данными валютами. В каждой вершине будем хранить максимальное количество данной валюты, которое мы можем получить какими-либо обменами.
Нам нужно определить, существует ли какой-либо цикл, позволяющий преумножить количество какой-то валюты. Если таковой существует, то мы можем за конечное число проходов по нему получить достаточное количество валюты X, чтобы при переводе ее в валюту S (возможно, несколькими обменами) было получено большее количество валюты, чем было изначально. Также заметим, что если описанный цикл существует, то существует и путь из вершины S в этот цикл.
Посчитаем максимально возможное количество каждой валюты после одной итерации (итерацией назовем проход по каждому ребру с обновлением максимума в конечной вершине). Заметим, что длина наибольшего простого пути не превышает $N-1$. Значит нам достаточно совершить $N-1$ итерацию, чтобы пройти по всевозможным простым путям.
После этого нам достаточно проверить, найдется ли такое ребро, после прохода по которому мы снова увеличим максимум какой-либо валюты. Существование такого ребра будет означать существование цикла, преумножающего количество некоторой валюты.

Сложность: $O(N \cdot M)$.

Исходный код

```cpp
#include <iostream>
#include <vector>

using namespace std;

#define ushort unsigned short

struct exchange {
    ushort src;
    ushort target;
    double rate;
    double commission;

    exchange() {
        rate = 0;
        commission = 0;
        src = 0;
        target = 0;
    }

    exchange(ushort src, ushort target, double rate, double commission) {
        this->src = src;
        this->target = target;
        this->rate = rate;
        this->commission = commission;
    }
};

int main() {
    ushort currencyCnt, exPointCnt, startCurrency;
    double startQuantity;
    cin >> currencyCnt >> exPointCnt >> startCurrency >> startQuantity;

    double quantityMax[currencyCnt];

    for (ushort i = 0; i < currencyCnt; i++) {
        quantityMax[i] = 0;
    }

    quantityMax[--startCurrency] = startQuantity;

    ushort a, b;
    double rateAB, commissionAB, rateBA, commissionBA;

    vector<struct exchange> exchangePointArr(exPointCnt);
    for (ushort i = 0; i < exPointCnt; i++) {
        cin >> a >> b >> rateAB >> commissionAB >> rateBA >> commissionBA;
        exchangePointArr.emplace_back(--a, --b, rateAB, commissionAB);
        exchangePointArr.emplace_back(b, a, rateBA, commissionBA);
    }

    while (currencyCnt--) {
        for (auto &exPoint: exchangePointArr) {
            a = exPoint.src;
            b = exPoint.target;
            quantityMax[b] = max(quantityMax[b],
                                 (quantityMax[a] - exPoint.commission) * exPoint.rate);
        }
```

```
59      }
60
61      for (auto &exPoint: exchangePointArr) {
62          if ((quantityMax[exPoint.src] - exPoint.commission) * exPoint.rate
63              > quantityMax[exPoint.target]) {
64              cout << "YES" << '\n';
65              return 0;
66          }
67      }
68      cout << "NO" << '\n';
69      return 0;
70  }
71
```

2.8 1806. Мобильные телеграфы

Условие

Ограничение времени	3 секунды
Ограничение памяти	64Mb

Каждому бойцу 25-й стрелковой дивизии выдали новейшее средство связи — мобильный телеграф. С его помощью можно отправлять телеграммы командованию и боевым товарищам прямо на поле битвы. К сожалению, конструкция телеграфов ещё далека от совершенства — передавать сообщения можно только между некоторыми парами телеграфов.
Каждому устройству присвоен уникальный номер — строка из десяти десятичных цифр. С телеграфа a можно отправить сообщение на телеграф b только в том случае, если из номера a можно получить номер b, изменив в нём ровно одну цифру либо поменяв в нём две цифры местами. Время передачи сообщения с телеграфа a на телеграф b зависит от длины наибольшего общего префикса их номеров — чем больше его длина, тем быстрее передаётся сообщение.
Во время очередного сражения Анка из своей хорошо замаскированной позиции увидела небольшую группу белых, пытающуюся обойти обороняющихся красноармейцев с тыла. Какое минимальное время понадобится на доставку этой информации от Анки до Чапаева по телеграфу, возможно, с помощью других красноармейцев?

Формат ввода

В первой строке записано целое число n ($2 \le n \le 50\ 000$) — количество бойцов в дивизии. Во второй строке через пробел в порядке невозрастания записаны десять целых чисел в пределах от 1 до 10000 — время передачи сообщения с одного телеграфа на другой при длине общего префикса их номеров, равной нулю, единице, двум, ..., девяти. Далее идут n строк, содержащие номера телеграфов, выданных бойцам дивизии. Номер телеграфа Анки указан первым, а номер телеграфа Чапаева — последним. Все номера телеграфов попарно различны.

Формат вывода

Если передать Чапаеву сообщение нельзя, выведите в единственной строке «−1». В противном случае в первой строке выведите минимальное время, за которое можно доставить сообщение. Во второй строке выведите количество бойцов, которые поучаствуют в его доставке, а в третьей строке выведите через пробел их номера в порядке от Анки к Чапаеву. Бойцы 25-й дивизии занумерованы числами от 1 до n в том порядке, в котором описаны номера их мобильных телеграфов на входе. Если существует несколько способов передать сообщение за минимальное время, выведите любой из них.

Решение

Воспользуемся алгоритмом Дейкстры.
На каждой итерации для текущей вершины мы будем "угадывать" вершины, с которыми она соседствует. Так как нас интересуют лишь вершины, номера которых могут получится из номера текущего телеграфа заменой одной цифры или перестановкой двух, то мы можем перебрать варианты и проверить их существование. В этом случае нам не придется тратить время на проверку каждой пары телеграфов: проверены будут лишь те, что соседствуют с вершинами оптимального пути.
Все номера телеграфов будем хранить в паре с их идентификатором в hash map'е. На каждой итерации при нахождении соседей текущей вершины будем хранить найденных соседей в set'е, отсортированном по стоимости перемещения в вершину.
Для построения последовательности выбора вершин для каждой итерации будем использовать приоритетную очередь.

Сложность:

$$O(E \cdot \log V) = \left[E \le \frac{n(n-1)}{2}\right] = O(n^2 \cdot \log n)$$

Исходный код

```cpp
#include <iostream>
#include <set>
#include <vector>
#include <queue>
#include <unordered_map>

#define ushort unsigned short
#define TEL_NUM_LEN 10
#define COST_INF SIZE_MAX

using namespace std;

struct node {
    ushort id;
    string num;
    size_t minCost;
    int pathAncestorId;
    // pair of the cost to deliver and neighbour id
    set<pair<ushort, ushort>> neighbours;
};

struct nodeCompare {
    bool operator()(node const *a, node const *b) const {
        return a->minCost > b->minCost;
    }
};

ushort getMaxPrefix(string &a, string &b) {
    ushort cnt = 0;
    for (ushort i = 0; i < TEL_NUM_LEN; i++) {
        if (a[i] != b[i]) return cnt;
        cnt++;
    }
    return cnt;
}

int main() {
    ushort n;
    cin >> n;

    ushort deliveryCostArr[TEL_NUM_LEN];
    for (ushort &i: deliveryCostArr) cin >> i;

    vector<node *> graph(n);
    unordered_map<string, ushort> nodeNumbers;

    for (ushort i = 0; i < n; i++) {
        graph[i] = new node();
        cin >> graph[i]->num;
        graph[i]->id = i;
        graph[i]->minCost = COST_INF;
        graph[i]->pathAncestorId = -1;

        nodeNumbers.insert({graph[i]->num, i});
    }

    graph[0]->minCost = 0;

```

```
    priority_queue<node *, vector<node *>, nodeCompare> processQueue;
    processQueue.push(graph[0]);
    while (!processQueue.empty()) {
        auto curNode = processQueue.top();
        processQueue.pop();

        if (curNode->id == n - 1) break;

        /*
         * Here we'll try to guess the number of possible neighbour.
         * We're searching number that can be obtained either
         * from replacing single character or by swapping two of them
         * in the current string.
         */
        set<pair<ushort, ushort>> neighboursCostIdPairSet;

        string curNum = curNode->num;

        for (ushort i = 0; i < TEL_NUM_LEN; i++) {
            char oldChar = curNum[i];

            for (ushort j = 0; j < TEL_NUM_LEN; j++) {
                char newChar = (char) ('0' + j);
                if (newChar == oldChar) continue;

                string nextNum = curNum;
                nextNum.replace(i, 1, string(1, newChar));
                auto it = nodeNumbers.find(nextNum);
                if (it != nodeNumbers.end())
                    neighboursCostIdPairSet.insert({
                        deliveryCostArr[getMaxPrefix(nextNum, curNum)],
                        it->second
                    });
            }

            for (ushort j = i + 1; j < TEL_NUM_LEN; j++) {
                string nextNum = curNum;
                swap(nextNum[i], nextNum[j]);
                auto it = nodeNumbers.find(nextNum);
                if (it != nodeNumbers.end())
                    neighboursCostIdPairSet.insert({
                       deliveryCostArr[getMaxPrefix(nextNum, curNum)],
                       it->second
                   });
            }
        }

        for (auto costIdPair: neighboursCostIdPairSet) {
            auto newCost = curNode->minCost + costIdPair.first;
            if (graph[costIdPair.second]->minCost > newCost) {
                graph[costIdPair.second]->minCost = newCost;
                graph[costIdPair.second]->pathAncestorId = curNode->id;
                processQueue.push(graph[costIdPair.second]);
            }
        }
    }

    if (graph[n - 1]->pathAncestorId == -1) {
        cout << -1 << '\n';
        return 0;
    }
```

```
120
121     vector<ushort> pathIds;
122     auto curNode = graph[n - 1];
123     while (curNode->id != 0) {
124         pathIds.push_back(curNode->id);
125         curNode = graph[curNode->pathAncestorId];
126     }
127
128     cout << graph[n - 1]->minCost << '\n';
129     cout << pathIds.size() + 1 << '\n';
130     cout << 1 << ' ';
131     for (auto it = pathIds.end() - 1; it >= pathIds.begin(); it--) cout << *it + 1 << ' ';
132     cout << '\n';
133
134     return 0;
135 }
136
```
