Бинарный и тернарный поиск

А. Просто бинарный поиск.

Условие: Всё, что требуется в этой задаче – реализовать алгоритм бинарного поиска элемента в массиве.

Входные данные: В первой строке входных данных содержатся натуральные числа N и K (1 ≤ N, K ≤ 100000). Во второй строке записаны N элементов первого массива, отсортированного по неубыванию. В третьей строке записаны K элементов второго массива. Элементы обоих массивов – целые числа, каждое из которых по модулю не превосходит 109.

Выходные данные: Для каждого элемента второго массива выведите слово "YES" (без кавычек), если он присутствует в первом массиве, и слово "NO" (без кавычек), если не присутствует.

Пример:  
Входные данные:  
4 3 1 5 5 8 1 5 9  
Выходные данные:  
YES YES NO

Реализуем алгоритм двоичного поиска элемента в массиве, путем деления его пополам и проверки соседних элементов через сравнение.

1. Определение значения элемента в середине структуры данных. Полученное значение сравнивается с ключом (переменная sought).   
2. Если ключ меньше значения середины, то поиск осуществляется в первой половине элементов, иначе — во второй.  
3. Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.  
4. Процесс продолжается до тех пор, пока не будет найден элемент со значением ключа или не станет пустым интервал для поиска.

#include <iostream>

using namespace std;

int main()

{

    const int SIZE = 100000;

    int N, K, sought, left, right, midd, arr[SIZE];

    cin >> N >> K;

    for (int i = 0; i < N; i++) //заполняем первый массив

        cin >> arr[i];

    for (int i = 0; i < K; i++)

    {

        cin >> sought; //вводим элемент второго массива

        left = 0;  //крайний левый элемент

        right = N - 1; //крайний правый

        while (left < right) //пока левая граница меньше правой

        {

            midd = left + (right - left) / 2; //находим средний элемент в первом массиве

            if (arr[midd] < sought) //если средний элемент первого массива меньше, чем введенный

                left = midd + 1; //то ищем в левой

            else right = midd; //если введенный больше, чем средний то переходим к правой

        }

        if (arr[left] == sought) //если введенный элемент равен элементу массива

            cout << "YES" << endl;

        else cout << "NO" << endl;

    }

    return 0;

}

B. Приближенный бинарный поиск.

Условие: В данной задаче вам нужно реализовать алгоритм бинарного поиска элемента в массиве, наиболее близкого к данному.  
Входные данные:  
В первой строке входных данных содержатся натуральные числа N и K (1 ≤ N, K ≤ 100000). Во второй строке записаны N элементов первого массива, отсортированного по неубыванию. В третьей строке записаны K элементов второго массива. Элементы обоих массивов – целые числа, каждое из которых по модулю не превосходит 109.  
Выходные данные:  
Для каждого элемента второго массива выведите элемент первого массива, наиболее близкий к нему. Если таких элементов больше одного, то выведите меньший из них.  
Пример:  
Входные данные:  
4 3 1 5 5 8 3 4 8  
Выходные данные:  
1 5 8  
Реализуем алгоритм двоичного поиска элемента в массиве, путем деления его пополам и сравнения соседних элементов, с нахождением наименьшего близкого к искомому.

#include <iostream>

using namespace std;

int main()

{

    const int SIZE = 100000;

    int N, K, left, right, midd, arr[SIZE], arr1[SIZE];

    cin >> N >> K;

    for (int i = 0; i < N; i++)

        cin >> arr[i];              //заполняем первый массив

    for (int i = 0; i < K; i++)

        cin >> arr1[i];             //второй

    for (int i = 0; i < K; i++)

    {

        left = 0;

        right = N - 1;

        while (right - left > 1)          //

        {

            midd = (left + right) / 2;      //находим средний в первом массиве

            if (arr[midd] < arr1[i])        //сравниваем элемент второго со средним первого

                left = midd;                //если меньше, то идем в левую часть

            else right = midd;      //если нет, то в правую

        }

        if (arr1[i] - arr[left] <= arr[right] - arr1[i]) //находим наиболее близкий к первомуэлементу второго массива элемент в первом

            cout << arr[left] << endl;

        else cout << arr[right] << endl;

    }

    return 0;

}

C. Уравнение с тангенсом.

Условие: Требуется найти корень уравнения tg(x) = x + k, лежащий в интервале от 0 до π / 2.  
Входные данные:  
Одно вещественное число k в диапазоне от 1.0 до 1000.0  
Выходные данные:  
Выведите корень уравнения x с точностью не менее 6 знаков после десятичной точки.  
Пример:  
Входные данные:  
1  
Выходные данные:  
1.13226773

#include <iostream>

#include <math.h>

#include <iomanip>

#define M\_PI          3.141592653589793238462643383279502884L /\* pi \*/

using namespace std;

int main()

{

    double left, right, midd, K;

    cin >> K;

    left = 0;           //левая граница интервала

    right = M\_PI / 2;  //правая граница интервала

    while (left < right - 0.00000001)   //пока левая граница не превышает правую(0.00000001 это погрешность)

    {

        midd = (left + right) / 2; //находим среднее значение интервала(корень)

        if (tan(midd) > (midd + K)) //если тангенс от среднего больше, среднего то идем в правую часть интервала

            right = midd;

        else left = midd; //если нет, то в левую

    }

    cout << fixed << setprecision(7) << right; // fixed и setprecision(7) для вывода 7 знаков после запятой, в условий нужно не меньше 6, чтобы норм округлило

    return 0;

}

D. Распилка досок

Условие: Имеется N досок разной длины. Из них нужно выпилить K одинаковых досок максимально возможной длины. Определите и выведите эту максимальную длину.  
Входные данные:  
В первой строке входных данных записаны через пробел числа N и K (1 ≤ N, K ≤ 1000). Далее вводятся N целых чисел – длины досок (в интервале от 1 до 1000). Числа разделяются пробелами и/или переводами строк.  
Выходные данные:  
Выведите одно вещественное число – результирующую длину с точностью не менее 4 знаков после десятичной точки.  
Пример:  
Входные данные:  
3 6   
5 7 6  
Выходные данные:  
2.500000

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <algorithm>

#include <cmath>

#include <iomanip>

using namespace std;

int main() {

    int n, k, arr[1000];

    double right, left, midd, result;

    cin >> n >> k;

    for (int i = 0; i < n; i++)

        cin >> arr[i];   //вводим доски

    right = arr[n - 1];  //почему-то правая граница это длинна последней доски

    left = 0;

    while (right - left > 0.000001) //тут опятьт погрешность

    {

        midd = (right + left) / 2; //средний

        result = 0;

        for (int i = 0; i < n; i++) //тут мы тип выпиливаем, но я не помню как это работает

            result += floor(arr[i] / midd);  //floor округляет в большую сторону

        if (result < k)

            right = midd;

        else left = midd;

    }

    cout << fixed << setprecision(6) << midd;

    return 0;

}

Е.Вася и строка.

Условие: Школьник Вася получил в подарок на день рождения строку длины n, состоящую из букв «a» и «b». Вася называет привлекательностью строки максимальную длину подстроки (последовательности соседних символов), состоящей из одинаковых символов.  
Вася может поменять в исходной строке не более k символов. Какой максимальной привлекательности данной строки он сможет добиться?  
Входные данные:  
В первой строке входных данных содержатся два числа n и k (1 ≤ n ≤ 100 000, 0 ≤ k ≤ n) — длина строки Васи и максимальное разрешённое количество изменений.  
Во второй строке следует строка Васи, состоящая из букв «a» и «b».  
Выходные данные:  
Выведите единственное целое число — максимальная привлекательность строки, которую Вася может получить, изменив в исходной строке не более k символов.  
Пример:  
Входные данные:  
4 2   
abba  
Выходные данные:  
4

Считываем входные данные.  
Через цикл обходим строку и увеличиваем переменные в зависимости от встреченных букв.  
Если значения полученных переменных меньше или равны количеству разрешенных для изменения символов, то присваиваем переменной максимальное значение из суммы полученных переменных.

#include <iostream>

#include <string>

#include <cmath>

#include <algorithm>

using namespace std;

int main() {

    int n, k;

    string str;

    int d = 0, f = 0, c = 0;

    cin >> n >> k;

    cin >> str;

    for (int i = 0; i < str.length(); i++)

    {

        if (str[i] == 97) //97 - это буква "а" в таблице символов, я пиздила поэтому не поменяла

            d++; //увеличиваем переменную отвечающию за количество "а"

        else

            f++; //увеличиваем переменную отвечающию за количество "b"

        if (d <= k || f <= k) //если поменять символы еще можно,

            c = max(c, d + f); //то мы записываем в "с" максимальное значение из "с", либо суммы "d+f"(сумма-это сколько символов идет подряд)

        else //это условие нужно для того чтобы, если перестанавливать уже нельзя, но в правой части элементов можно применить такое же количество перестановок и подряд идущих элементов будет больше

            if (str[i - (d + f) + 1] == 97) //если элемент в левой части равен "а"

                d--; //то вычитаем из количества "а"

            else f--; //или из "b"

    }

    cout << c;

    return 0;

}

Линейные алгоритмы и структуры

В. Правая скобка.

Условие: Дана строка, состоящая из различных видов скобок – то есть символов '(', ')', '[', ']', '{', '}'.  
Разрешается любую открывающую скобку заменить на скобку другого вида, но тоже открывающую.  
Аналогично, разрешается любую закрывающую скобку заменить на скобку другого вида, но тоже закрывающую.  
Определите, за какое наименьшее число замен можно получить правильное скобочное выражение (в котором каждой открывающей скобке соответствует закрывающая такого же вида). Например, выражение '{[])' можно превратить в правильное выражение за одну замену.  
Входные данные:  
Одна строка, содержащая не более 1000 символов, где каждый символ – это открывающая или закрывающая скобка.  
Выходные данные:  
Выведите наименьшее количество замен, которое нужно сделать, чтобы получить правильное скобочное выражение. Если его получить не удастся, выведите -1.  
Пример:  
Входные данные:  
{[])  
Выходные данные:  
1  
Описание алгоритма:

Для решения задачи воспользуемся реализацией структуры данных «Стек», использующей принцип LIFO(Last-In-First-Out). Присваиваем входные данные строковой переменной. Далее обходим полученную строку циклом и через условные операторы отправляем открывающие скобки в «Стек». Далее при помощи условного оператора сравниваем последний элемент «Стека» и закрывающую скобку из входных данных. При совпадении типа скобок – удаляем элемент из «Стека» и инкрементируем переменную-счетчик скобок.

СТЭК – ПОСЛЕДНИЙ ЗАШЕЛ – ПЕРВЫЙ ВЫШЕЛ, ПЕРВЫЙ ЗАШЕЛ- ПОСЛЕДНИЙ ВЫШЕЛ

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

    string str;

    vector<char> openingBrackets;

    int countPermutation = 0, j = 0;

    bool flag = true;

    cin >> str;

    for (int i = 0; i < str.length(); i++)

    {

        if (str[i] == '(' || str[i] == '[' || str[i] == '{') //если скобка открывающаяся,

            openingBrackets.push\_back(str[i]); //то добавляем ее в вектор(тип стэк) так все открывающиеся скобки

        else {

            if (openingBrackets.size() == 0) //если нет, и вектор пустой

            {

                flag = false; //то ставим флаг отрицательным,

                break; //выходим из цикла

            }

            if (openingBrackets[openingBrackets.size() - 1] == '(' && str[i] != ')') //если последняя скобка, добавленная в стэк не совпадает со соответствующей ей закрывающейся, то

                countPermutation++; //увеличиваем счетчик замен на 1

            if (openingBrackets[openingBrackets.size() - 1] == '[' && str[i] != ']')

                countPermutation++;

            if (openingBrackets[openingBrackets.size() - 1] == '{' && str[i] != '}')

                countPermutation++;

            openingBrackets.pop\_back(); //удаляем из вектора(стека) последний элемент

        }

    }

    if (openingBrackets.size() != 0) //если вектор не пустой

        flag = false;  //то получить правильное скобочное выражение нельзя

    if (flag) //проверка можно ли получить правильное скобочное выражение

        cout << countPermutation;

    else cout << "-1";

    return 0;

}

D. Выгодный бизнес

Начинающий предприниматель Сигизмунд хранит на компьютере сведения о прибыли своей фирмы за каждый день её работы с момента основания. Поскольку в последнее время дела у него идут как-то не очень, для поднятия настроения Сигизмунд решил сформировать следующий отчёт: для каждого дня найти ближайший предшествующий день, в который прибыль была ещё меньше, чем в этот. Помогите ему это сделать.  
Входные данные  
В первой строке входных данных записано натуральное число N – количество дней. В следующей строке записаны через пробел N неотрицательных целых чисел p1, p1, ..., pN – прибыль компании в день 1, 2, ..., N.  
Выходные данные  
Выведите N целых чисел, где i-е число – это наибольший номер дня, меньший i, в который прибыль была меньше, чем в день i. Если такого дня не существует, следует вывести число 0.  
Ограничения: 1 ≤ N ≤ 105, 0 ≤ pi ≤ 106  
Пример  
входные данные  
5  
2 5 3 4 5  
выходные данные  
0 1 1 3 4

Не знаю какая структура данных тут реализована , никакая наверное, должна быть стэк или очередь

#include <iostream>

using namespace std;

int main()

{

const int SIZE = 100000;

int n, arr[SIZE], result[SIZE];

cin >> n;

for (int i = 0; i < n; i++)

cin >> arr[i];  //массив с прибылью за день

for (int i = 0; i < n; i++)

{

result[i] = 0; //записываем в результирующий массив 0

for (int j = i - 1; j >= 0;) //цикл для поиска предыдшествующих дней

{//цикл пока j >=0(при первой итерациии не заходит) у нулевого дня нет предшествующих дней

if (arr[i] > arr[j]) //если в данный день, на котором итерация прибыль больше, чем в предыдущий

{

result[i] = j + 1; //то записываем в результирующий массив номер дня(+1 потому что итерации с нуля, а дни с 1)

break; //выходим из цикла

}

else j = result[j] - 1; //иначе возвращаем j на последний предшествующем дне, который в результируещем массиве

}

cout << result[i] << " "; //выводим дни

}

return 0;

}

Е. Пароль

Во время сложной и длительной экспедиции космический корабль попал под метеоритный поток, что привело к его повреждениям. В частности, нарушена целостность системы хранения числового пароля: из строя выведено несколько ячеек, в каждой из которых хранилась одна из его цифр.  
К счастью, у космонавта сохранилась копия старого пароля. Теперь ему необходимо срочно создать новый, для этого он должен из старого пароля убрать несколько цифр. Количество убираемых цифр соответствует количеству вышедших из строя ячеек. Обязательное условие для нового пароля – оставшееся после удаления цифр число должно быть наибольшим из всех возможных.  
Напишите программу, которая поможет космонавту быстро определить новый пароль для системы.  
Входные данные  
В первой строке входных данных записано натуральное число N без ведущих нулей – исходный пароль. Пароль состоит не более чем из 106 цифр.  
Во второй строке записано натуральное число K – количество цифр, которые надо удалить (известно, что после аварии уцелело не менее одной ячейки).  
Выходные данные  
Одно число – новый пароль системы  
Пример  
входные данные  
25376  
2  
выходные данные  
576

Это похоже на реализацию очереди, первый зашел – первый вышел

#include <iostream>

#include <list>

#include <string>

using namespace std;

int main() {

list <int> result;

list <int>::iterator it, ip; //итераторы для листа объявляем

int k;

string s;

cin >> s;

cin >> k;

for (int i = 0; i < s.length(); i++)

result.push\_back(s[i] - '0'); //добавляем в лист старый пароль, "-'0'" это конвертация из стринг в инт чот такое

it = ip = result.begin(); //оба равный первой цифре старого пароля

ip++; //увличиваем на один, получается равен второй цифре пароля

while (k != 0) { //пока количество цифр, которые надо удалить не равно 0

if (\*ip > \*it) { //сравниваем элементы в итератора, если элемент под индексом it меньше, то

it = result.erase(it); //удаляем меньший, it становится равен следующему

if (it != result.begin()) it--; //если it равен не первом элементу в листе, то уменьшаем его

else ip++; //иначе увеличиваем ip, чтобы он стал следующим после it

k--; //уменьшаем количество удаляемых цифр

}

else {  //если элемент под индексом it > элемента под индексом ip

if (ip != result.end()) { //если ip не последний элемент в листе

it++; //увеличиваем оба итератора

ip++;

}

else break; //иначе выходим из цикла

}

}

if (k != 0) { //если количество удаляемых цифр не равно 0

while (k != 0) { //то пока, оно не станет = 0

result.pop\_back(); //удаляем из листа последний элемент

k--;

}

}

for (auto elem : result) //цикл по коллекции

cout << elem; //выводим результат

return 0;

}

Битовые операции

А. Разворот байта.  
Дано целое неотрицательное число N размером 1 байт, то есть 8 бит. Поменяйте порядок битов в данном числе на обратный.  
Рассмотрим пример. Пусть N = 5. В двоичном виде это будет 00000101. Переворачиваем — получаем 10100000. В десятичном представлении это будет 160.  
Входные данные  
Входные данные – целое число N в диапазоне от 0 до 255.  
Выходные данные  
Одно целое число — ответ.  
Пример  
входные данные  
5  
выходные данные  
160

#include <iostream>

#include <string>

#include <math.h>

using namespace std;

int main()

{

    int n, result = 0;

    cin >> n;

    string buffer = ""; //сюда в двоичном виде записывать будем

    do                    //цикл переводит чило в двоичную систему счисления, записывает в строку

    {

        buffer += char('0' + n % 2);  //'0' - коневертирует из инт в чар наверное, как то так это рабоатет

        n = n / 2;

    } while (n > 0);

    for (int i = 0; buffer.length() < 8; i++) //добавляем соответствующее число 0 чтобы число было 8ми битовым

        buffer += '0';

    for (int i = 0; i < 8; i++)  //тут переводим обратно в десятичную систему счисления

        if (buffer[i] == '1')

            result += pow(2, 8 - (i + 1));

    cout << result;

    return 0;

}

В. Битовый массив

Дан массив b[0..N - 1], элементами которого являются беззнаковые 8-битные числа. Соответственно, размер данного массива равен N байтов, или 8·N битов. Изначально все элементы равны нулю.  
Все биты массива последовательно пронумерованы следующим образом:  
0 – младший бит в элементе b[0],  
...  
7 – старший бит в элементе b[0],  
8 – младший бит в элементе b[1],  
...  
8·N - 1 - старший бит в элементе b[N - 1].  
Вам нужно выполнить M операций, где каждая операция может быть одного из трёх видов:  
set i — установить бит с номером i (то есть записать в него единицу)  
reset i — сбросить бит с номером i (то есть записать в него ноль)  
invert i — инвертировать бит с номером i (поменять 0 на 1, а 1 на 0)  
Выведите содержимое массива b после выполнения всех операций.  
Входные данные  
Первая строка входных данных содержит два целых числа N и M — размер массива и количество операций (1 ≤ N, M ≤ 105).  
Каждая из следующих M строк содержит название операции и далее через пробел номер бита.  
Выходные данные  
Выведите N целых чисел – содержимое массива b после выполнения всех операций.  
Пример  
входные данные  
2 4  
set 1  
invert 10  
reset 8  
invert 8  
выходные данные  
2 5

#include <iostream>

#include <string>

#include <regex>

#include <vector>

#include <math.h>

using namespace std;

int main()

{

    int n, m, result = 0;

    string comand, set = "set", invert = "invert", reset = "reset";

    vector<int> b;

    cin >> n >> m;

    for (int i = 0; i < n \* 8; i++)

        b.push\_back(0);                 //записываем в вектор биты, n\*8 подсчитывает количество битов в цисле

    for (int i = 0; i <= m; i++)

    {

        getline(cin, comand);   //ввод комнады

        comand = " " + comand;     //в начало введенной команды добавляем пробел

        if (comand.find(" " + set + " ") != string::npos) //ищем символы "set" в веденной строке

        {

            comand.erase(0, 4);     //если находим, то удаляем 5 символов с начала строки

            int iter = stoi(comand); //stoi находит есть ли буквы и символы в строке, если нет, то отправляет оставшееся(цифры) в тип инт, возвращает целочисленные значения

            b[iter] = 1; //добавляем в массив 1 в место номера бита

        }

        if (comand.find(" " + invert + " ") != string::npos) //ищем символы "invert" в веденной строке

        {

            comand.erase(0, 7); //удаляем 8 символов (короче удаление везде,удаляет пробел в начале, команду и пробел после строки, оставляет только цифорку введенную - номер бита

            int iter = stoi(comand);//возвращаем номер бита

            if (b[iter] == 0) b[iter] = 1; //выполняем команду invrt (если бит = 0, томеняем на 1 и наоборот)

            else b[iter] = 0;

        }

        if (comand.find(" " + reset + " ") != string::npos)//ищем символы "reset" в веденной строке

        {

            comand.erase(0, 6); //удаляем 7 символов с начала строки

            int iter = stoi(comand); //возвращаем номер бита

            b[iter] = 0; //удаляем бит

        }

    }

    for (int i = 0; i < b.size(); )//цикл по всему массиву двоичному

    {

        for (int j = 0; j < 8; j++) //цикл для перевода одного цисла

        {

            if (b[i] == 1)

                result += pow(2, j); //переводим число в десятичное

            i++;

        }

        cout << result << " "; //выводим такое количество чисел которое задано в N

        result = 0;

    }

    return 0;

}

С. И/ИЛИ

Даны два 32-битных беззнаковых числа x и z. Найдите такое 32-битное беззнаковое число y, что x and y = z либо x or y = z.  
Примечание: операция and означает побитовое И, операция or — побитовое ИЛИ.  
Входные данные  
Два целых числа x и z (0 ≤ x, z < 232).  
Выходные данные  
Выведите искомое число y. В случае нескольких правильных ответов выведите любой. Если решения не существует, выведите -1.  
Примеры  
входные данные  
6 4  
выходные данные  
5  
входные данные  
6 7  
выходные данные  
5  
входные данные  
6 1  
выходные данные  
-1

#include <iostream>

#include <string>

#include <math.h>

using namespace std;

int main()

{

    unsigned long long int x, z, y = 0;

    string xStr, zStr, yStrAnd, yStrOr;

    cin >> x >> z;

    unsigned long long int tempX = x;

    unsigned long long int tempZ = z;

    do                     //переводим числа в двоичную систему

    {

        xStr += char('0' + tempX % 2);

        zStr += char('0' + tempZ % 2);

        tempX = tempX / 2;

        tempZ = tempZ / 2;

    } while ((tempX > 0) && (tempZ > 0));

    if (xStr.length() == 1)//если у первого длина 1, то решений не существует

    {

        cout << "-1";

        return 0;

    }

    for (int i = 0; i < xStr.length(); i++) //здесь мы находим y учитывая это условие: x and y = z либо x or y = z

    {

        if (xStr[i] == '1' && zStr[i] == '1') //если они такому равны        1   x

        {                                //                                  1   y

            yStrAnd += '1'; //то в таблице логической значение  И будет это  1  AND

            yStrOr += '0'; //или это                                         0   OR

        }

        else if (xStr[i] == '0' && zStr[i] == '0') // 0   x

        {                                          // 0   y

            yStrAnd += '1';                        // 1  AND

            yStrOr += '0';                         // 0  OR

        }

        else if (xStr[i] == '1' && zStr[i] == '0')  // 1  x

        {                                           // 0  y

            yStrAnd += '0';                         // 0 AND

        }

        else if (xStr[i] == '0' && zStr[i] == '1')  // 0  x

        {                                           // 1  y

            yStrOr += '1';                          // 1  OR

        }

    }

    int size = 0;

    string del;

    if (yStrAnd.length() > yStrOr.length()) //если И длиннее ИЛИ

    {

        size = yStrAnd.length();       //то размер равен И

        del = yStrAnd;           //делитель наверное равен И

    }

    else

    {

        size = yStrOr.length();    //обратное предыдущему

        del = yStrOr;

    }

    for (int i = size - 1; i >= 0; i--)  //переводим из двоичной в десятичную в зависимости какая операция была обнаружена при высчитывании у

    {

        if (del[i] == '1')

            y += pow(2, i);

    }

    if ((x & y) == z || (x | y) == z)       //проверяем условие: x and y = z либо x or y = z, если сходится, выводим у

        cout << y;

    else cout << -1;

    return 0;

}

Перебор

А. Наибольшее произведение

Дано N целых чисел. Требуется выбрать из них три таких числа, произведение которых максимально.  
Входные данные  
Сначала вводится число N — количество чисел в последовательности (3 ≤ N ≤ 100). Далее вводится сама последовательность: N целых чисел, по модулю не превышающих 1000.  
Выходные данные  
Выведите три искомых числа в любом порядке. Если существует несколько ответов, то выведите любой.  
Примеры  
входные данные  
9  
3 5 1 7 9 0 9 -3 10  
выходные данные  
9 10 9  
входные данные  
3  
-5 -300 -12  
выходные данные  
-5 -300 -12

using System;

using System.Linq;

namespace vinni

{

class Program

{

static void Main(string[] args)

{

var N = int.Parse(Console.ReadLine());

int[] arr = Console.ReadLine().Split(' ').Select(s => int.Parse(s)).ToArray(); //это мы запросом LINQ вводим массив

Array.Sort(arr); //сортируем его

var prPositive = arr[arr.Length - 1] \* arr[arr.Length - 2] \* arr[arr.Length - 3]; //умножаем три наибольших положительных числа

var prNegative = arr[0] \* arr[1] \* arr[arr.Length - 1]; //на случай, если есть отрицательные, умножаем первые два (минимальные), и последнее (максимальное)

if(prPositive > prNegative) //находим наибольшее произведение

Console.Write($"{arr[arr.Length - 1]} {arr[arr.Length - 2]} {arr[arr.Length - 3]}");

else

Console.WriteLine($"{arr[0]} {arr[1]} {arr[arr.Length - 1]}");

}

}

}

B. Наибольшее произведение – 2. Так же как и в предыдущей

С. Равенство.

Даны N целых чисел. Расставить между ними знаки '+' и '-' так, чтобы значение получившегося выражения было равно заданному целому S.  
Входные данные  
В первой строке входных данных записано целое число N (2 ≤ N ≤ 24). Во второй строке записаны целые числа X1, X2, ..., XN (0 ≤ Xi ≤ 5·107). В третьей строке записано целое число S ( - 109 ≤ S ≤ 109).  
Выходные данные  
Результирующее выражение следует выводить без пробелов, как показано в примере. Если решений несколько, выведите любое. Если решения нет, выведите "No solution".  
Примеры  
входные данные  
3  
15 25 30  
10  
выходные данные  
15+25-30=10  
входные данные  
2  
10 10  
100  
выходные данные  
No solution

using System;

using System.Linq;

namespace vinni

{

class C\_Equality

{

static void Main(string[] args)

{

var N = int.Parse(Console.ReadLine());

int[] arr = Console.ReadLine().Split(' ').Select(s => int.Parse(s)).ToArray(); //вводим массив

var S = int.Parse(Console.ReadLine()); //вводим число которое доолжно получится

char[] sign = new char[25]; //массив символов инициализируем

void output() //функция для ввыода данных

{

int sum = arr[0]; //первое число

Console.Write($"{arr[0]}");//выводим первое число

for(var i = 1; i < arr.Length; i++) Console.Write($"{sign[i]}{arr[i]}"); //выводим знак и число, которое прибавляем, либо вычитаем

Console.Write($"={S}"); //выводим сумму

Environment.Exit(0); //завершаем работу программы

}

void rec(int curSum, int curIndex)

{

if(curSum == S && curIndex != 1)  //сравниваем элемент с числом, которое должно получится

output();

if(curIndex == N) //сравниваем переменную для индекса с переменной, которыую мы вводили для количества значений в массиве

return; //выходим из функции, если равны

sign[curIndex] = '-'; //добавляем знак минус в массив символов

rec(curSum - arr[curIndex], curIndex + 1); //передаем в функцию разность элементов массива, увеличиваем индекс на 1

sign[curIndex] = '+'; //добавляем +

rec(curSum + arr[curIndex], curIndex + 1); //передаем сумму, так же увеличиваем индекс

}

rec(arr[0], 1); //передаем первый элемент и индекс второго

Console.WriteLine("No solution"); //выводим, если не будет выполнена функция out

}

}

}

Е. Скобочные выражения.

Выведите все правильные скобочные выражения длины N, состоящие из круглых и квадратных скобок.  
Входные данные  
Входные данные содержат одно целое чётное число N (2 ≤ N ≤ 16)  
Выходные данные  
Выведите все правильные скобочные выражения в произвольном порядке без пробелов внутри.  
Пример  
входные данные  
4  
выходные данные  
(())  
([])  
()()  
()[]  
[()]  
[[]]  
[]()  
[][]

using System;

using System.Linq;

namespace vinni

{

class E\_BracketExpressions

{

static void Main(string[] args)

{

void gen(int n, int count, string stack, string ans)//count - количесвто открывающихся скобок, stack - стэк для открывающихся скобок, ans -ответ

{

if(count == n / 2 && stack.Length == 0) //если стэк пуст, и количество открывающихся скобок верно

{

Console.WriteLine(ans); //выводим скобочное выражение

return;

}

if(count < n / 2) //помещаем открывающиеся скобки в стэк и в ans

{

gen(n, count + 1, stack + '(', ans + '(');

gen(n, count + 1, stack + '[', ans + '[');

}

if(stack.Length > 0) //если стэк не пустой

{

if(stack[stack.Length - 1] == '(') gen(n, count, stack.Remove(stack.Length - 1), ans + ')'); //если последний элемент в стэке "(", удаляем его, в результат вводим нужную закрывающуся скобку

if(stack[stack.Length - 1] == '[') gen(n, count, stack.Remove(stack.Length - 1), ans + ']');

}

}

int N = int.Parse(Console.ReadLine()); //число вводим

gen(N, 0, "", ""); //передаем в функцию число, остальные данные

}

}

}

Хеш-таблицы.

А. Хеш-функция.

В задачах поиска часто используются так называемые хеш-функции. Хеш-функции могут определяться самым разнообразным образом — например, так:  
Вначале число возводится в квадрат.  
Если полученное число имеет три и более цифр, то первая и последняя его цифры отбрасываются.  
Ответом будет остаток от деления числа на M (где M – заданное заранее значение).  
Рассмотрим пример. Пусть M = 10, входное число = -37.  
Возводим в квадрат:  - 372 = 1369.  
Число имеет более двух цифр, поэтому отбрасываем первую и последнюю цифру — получаем 36.  
Берём остаток от деления на M — получаем ответ 6.  
Реализуйте вычисление такой хеш-функции и вычислите её для каждого из N входных чисел.  
Входные данные  
В первой строке входных данных записаны два целых числа M и N (1 ≤ M, N ≤ 105).  
Во второй строке записаны N целых чисел, для каждого из которых вы должны сосчитать и вывести значение хеш-функции. Все числа не превышают по модулю 109.  
Выходные данные  
Выведите N целых чисел — ответы.  
Пример  
входные данные  
10 5  
-37 37 5 2 -352145261  
выходные данные  
6 6 5 4 2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace vinni

{

class A\_HashFunction

{

static void Main(string[] args)

{

int[] NM = Console.ReadLine().Split(' ').Select(s => int.Parse(s)).ToArray(); //вводим числа первые которые надо, я не нашла как здесь вводить два числа через пробел, поэтому через  массив

long[] arr = Console.ReadLine().Split(' ').Select(s => long.Parse(s)).ToArray(); //массив вводим

for(var i = 0; i < NM[1]; i++)

{

long num = arr[i] \* arr[i]; //взводим входное число в квадрат

string number = Convert.ToString(num); //конвертируем квадрат входного в строку

if(number.Length > 2) //если число имеет больше двух цифр (все дальнейшее по условию)

{

number = number.Remove(0, 1); //отбрасываем первую

number = number.Remove(number.Length - 1); //отбрасываем вторую

}

arr[i] = long.Parse(number); //переводим его, уже обработанное обратно в инт

Console.Write($"{arr[i] % NM[0]} "); //выводим ответ(он получается, если взять остаток от деления

}

}

}

}

В. Хеширование с цепочками.

Одним из эффективных способов реализации множеств в памяти компьютера является хеш-таблица.  
Вставка нового элемента в хеш-таблицу начинается с вычисления хеш-функции от ключа. Получающееся значение является индексом в хеш-таблице. В данной задаче используется простейший вариант хеш-функции: h(x) = x mod M (где M – размер хеш-таблицы).  
Заметим, что возможна ситуация, когда для разных ключей хеш-функция даст одно и то же значение. Такая ситуация называется коллизией. Одним из способов разрешения коллизий является метод цепочек: элементы множества, которым соответствует одно и то же хеш-значение, связываются в цепочку — связный список.  
На рисунке показан пример результирующей хеш-таблицы после вставки последовательности чисел 75, 66, 42, 192, 91, 40, 49, 87, 67, 16, 417, 130, 372, 227 в хеш-таблицу размера M = 10.  
Вам дана последовательность неповторяющихся ключей – целых чисел. Требуется определить длину самой длинной цепочки в итоговой хеш-таблице. Например, в примере на рисунке самая длинная цепочка имеет длину 4.  
Входные данные  
Первая строка входных данных содержит два натуральных числа M и N (1 ≤ M, N ≤ 105).  
Во второй строке записаны N различных неотрицательных целых чисел, не превосходящих по модулю 109.  
Выходные данные  
Выведите одно число — максимальную длину цепочки в результирующей хеш-таблице.  
Пример  
входные данные  
10 14  
75 66 42 192 91 40 49 87 67 16 417 130 372 227  
выходные данные  
4

1. using **System**;
2. using **System**.**Collections**.**Generic**;
3. using **System**.**Linq**;
4. using **System**.**Text**;
5. using **System**.**Threading**.**Tasks**;
7. namespace vinni
8. {
9. **class** B\_HashingChines
10. {
11. **static** **void** **Main**(**string**[] args)
12. {
13. **int**[] MN = **Console**.**ReadLine**().**Split**(**' '**).**Select**(s => **int**.**Parse**(s)).**ToArray**();
14. **int**[] arr = **Console**.**ReadLine**().**Split**(**' '**).**Select**(s => **int**.**Parse**(s)).**ToArray**();
15. **long**[] hash = **new** **long**[MN[0]];

18. **for**(**var** i = 0; i < MN[1]; i++)
19. hash[(arr[i] % MN[0])]++;
21. **Console**.**WriteLine**($**"{hash.Max()}"**);
22. }
23. }
24. }

D. Коллизии.

В задачах поиска часто используются так называемые хеш-функции. Одна из самых простых хеш-функций выглядит так: h(x) = x mod M, где M – заранее заданное число.  
Назовём коллизией ситуацию, когда для двух различных аргументов x1 и x2 оказывается, что h(x1) = h(x2).  
Вам дано N различных неотрицательных целых чисел. Определите, сколько пар среди этих чисел образуют коллизию.  
Входные данные  
Первая строка входных данных содержит два натуральных числа M и N (1 ≤ M, N ≤ 105).  
Во второй строке записаны N различных неотрицательных целых чисел, не превосходящих по модулю 109.  
Выходные данные  
Определите, сколько пар входных чисел образуют коллизию.  
Пример  
входные данные  
10 5  
123 456 7893 458 13  
выходные данные  
3

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace vinni

{

class D\_Collisions

{

static void Main(string[] args)

{

int[] MN = Console.ReadLine().Split(' ').Select(s => int.Parse(s)).ToArray(); //вводим M N

int[] arr = Console.ReadLine().Split(' ').Select(s => int.Parse(s)).ToArray(); //массив

int[] hash = new int[MN[0]]; //инициализируем массив для хеш-таблицы

int count = 0;

for(var i = 0; i < MN[1]; i++) //находим места в хеш-таблице, в котороые можно записать элементы массива

hash[(arr[i] % MN[0])]++; //индекс в [] это место элемента в хэш-таблице

for(var i = 0; i < hash.Length; i++)

{

if(hash[i] > 1) //если в ячейке больше одного значения(значит коллизия)

{

count += hash[i] \* (hash[i] - 1) / 2; //поиск пар коллизий(формула такая, потому что нужны пары)

}

}

Console.WriteLine($"{count}");

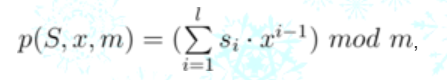
}

}

}

Е. Взлом хеш-функции.

В некоторых задачах защиты информации используются так называемые хеш-функции. Одним из важнейших классов хеш-функций являются полиномиальные хеш-функции.  
Пусть дана строка S = s1s2...sl, состоящая из цифр от 0 до 9. Тогда значение полиномиальной хеш-функции p(S, x, m) вычисляется следующим образом:,



где a mod b обозначает остаток от деления числа a на число b. Например, пусть S = "0123", тогда p (S, 2, 5)  =  (0·1 + 1·2 + 2·4 + 3·8) mod 5  =  4.  
Одним из способов применения хеш-функций является хранение паролей. Часто бывает так, что пароли приходится хранить в незащищенной таблице базы данных, поэтому вместо них хранят хеш-функции от них. При проверке пароля вычисляется хеш-функция от введенной строки и сравнивается со значением, хранящимся в таблице.  
Ваша задача состоит в том, чтобы по заданным числам x, m, L и v найти строку S из цифр от 0 до 9 длины L, такую что значение полиномиальной хеш-функции p (S, x, m) равно v.  
Входные данные  
Входные данные содержат четыре целых числа: x (x – простое число, 5 ≤ x ≤ 100), m (m является степенью двойки, 1 ≤ m ≤ 256), L (10 ≤ L ≤ 100) и v (0 ≤ v ≤ m - 1).  
Выходные данные  
Выведите искомую строку или NO SOLUTION, если такой строки не существует. Если решений несколько, выведите любое.  
Пример  
входные данные  
5 16 10 9  
выходные данные  
0422207956

Сортировка

А. Флеш-карты.

У Эдварда есть n флеш-карт объемом a1, a2, ..., an мегабайт и большой файл размера m мегабайт.  
Найдите минимальное количество флеш-карт, на которые можно записать файл Эдварда, если он может разделить свой файл на части произвольного размера.  
Входные данные  
В первой строке входных данных содержится целое положительных число n (1 ≤ n ≤ 100) — количество флеш-карт.  
Во второй строке входных данных содержится целое положительное число m (1 ≤ m ≤ 105) — размер файла Эдварда в мегабайтах.  
В следующих n строках входных данных содержится последовательность целых положительных чисел a1, a2, ..., an (1 ≤ ai ≤ 1000) — размеры флеш-карт в мегабайтах.  
Гарантируется, что ответ существует, то есть сумма всех ai не меньше чем m.  
Выходные данные  
В единственной строке выходных данных должно содержаться целое положительное число — минимальное количество флеш-карт, на которые можно записать файл Эдварда, если он может разделить свой файл на части произвольного размера.  
Примеры  
входные данные  
3  
5  
2  
1  
3  
выходные данные  
2

using System;

using System.Linq;

namespace vinnieEight

{

class A\_FlashMap

{

static void Main(string[] args)

{

var N = int.Parse(Console.ReadLine());

var M = int.Parse(Console.ReadLine());

int[] arr = new int[N];

int sum = 0, count = 0;

for(var i = 0; i < arr.Length; i++)

arr[i] = int.Parse(Console.ReadLine()); //вводим массив

Array.Sort(arr); //сортируем

for(var i = arr.Length - 1; i >= 0; i--) //идем с конца массива(с наибольших)

{

count++;  //уыеличиваем счетчик

sum += arr[i]; //считаем сумму

if(sum >= M) break; //если сумма превышает размер файла, то цикл завершаем

}

Console.WriteLine($"{count}");

}

}

}

В. Сортировка времени

Вам дано *N* моментов времени, каждый момент задан тремя числами - часы, минуты и секунды. Требуется отсортировать все моменты времени по неубыванию.

**Входные данные**

В первой строке входных данных записано целое число *N* (1 ≤ *N* ≤ 100). В каждой из следующих *N* строк записаны моменты времени в виде 3 целых чисел через пробел – часы (от 0 до 23), минуты (от 0 до 60) и секунды (от 0 до 60).

**Выходные данные**

Выведите моменты времени в таком же формате, упорядоченные по неубыванию.

**Пример**

**входные данные**

4  
10 20 30  
7 30 00  
23 59 59  
13 30 30

**выходные данные**

7 30 0  
10 20 30  
13 30 30  
23 59 59

using System;

using System.Collections.Generic;

using System.Linq;

namespace vinnieEight

{

class B\_SortTime

{

static void Main(string[] args)

{

var N = int.Parse(Console.ReadLine());

int[] arr = new int[N]; //массив строк

string[] s;

for(int i = 0; i < N; i++)

{

s = Console.ReadLine().Split(' '); //вводим часы минуты и секунды

arr[i] = int.Parse(s[0]) \* 3600 + int.Parse(s[1]) \* 60 + int.Parse(s[2]); //конвертируем в инт, переводим в секунды(находим сколько секунд ввели)

}

Array.Sort(arr); //сортируем

foreach(var time in arr)

{

int hh = time / 3600; //вычислем часы

int mm = time / 60 - hh \* 60; //вычисляем минуты

int ss = time - hh \* 3600 - mm \* 60; //вычислем секунды

Console.WriteLine($"{hh} {mm} {ss}");

}

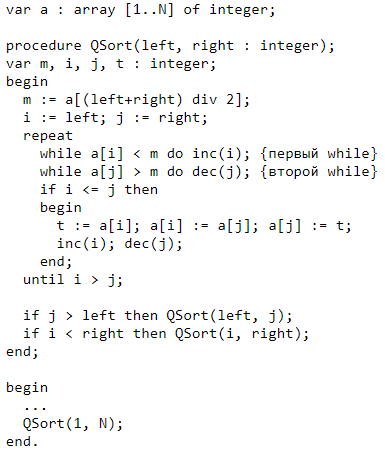
}

}

}

D. Быстрая сортировка.

Для сортировки последовательности чисел широко используется быстрая сортировка - QSort. Ниже приведена программа на языке Pascal, которая сортирует массив a, используя данный алгоритм:



Хотя QSort является самой быстрой сортировкой в среднем, существуют тесты, на которых она работает очень долго. Будем оценивать время работы алгоритма количеством сравнений, сделанных с элементами массива (т.е. суммарным количеством сравнений в первой и втором while). Требуется написать программу, генерирующую тест, на котором быстрая сортировка сделает наибольшее число таких сравнений.

**Входные данные**

Цело число *N* (1 ≤ *N* ≤ 70000)

**Выходные данные**

Выведите перестановку чисел от 1 до *N*, на которой быстрая сортировка выполнит максимальное число сравнений. Если таких перестановок несколько, выведите любую из них.

**Пример**

**входные данные**

3

**выходные данные**

1 3 2

Быстрая сортировка

Алгоритм состоит из трёх шагов:

1. Выбрать элемент из массива. Назовём его опорным.
2. *Разбиение*: перераспределение элементов в массиве таким образом, что элементы, меньшие опорного, помещаются перед ним, а большие или равные - после.
3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.

В наиболее сбалансированном варианте при каждой операции разделения массив делится на две одинаковые (плюс-минус один элемент) части, следовательно, максимальная глубина рекурсии, при которой размеры обрабатываемых подмассивов достигнут 1, составит {\displaystyle \log \_{2}n}. В результате количество сравнений, совершаемых быстрой сортировкой, было бы равно значению рекурсивного выражения {\displaystyle C\_{n}=2\cdot C\_{n/2}+n}, что даёт общую сложность алгоритма {\displaystyle O(n\cdot \log \_{2}n)}.

using System;

using System.Collections.Generic;

using System.Linq;

namespace vinnieEight

{

class D\_FastSort

{

static void Main(string[] args)

{

int[] input = Console.ReadLine().Split(' ').Select(s => int.Parse(s)).ToArray(); //тут не нужен массив я видимо замечталась, надо просто вводить одно число

int[] arr = new int[input[0]];

for(int i = 0; i < arr.Length; i++)

arr[i] = i + 1; //заполняем массив числами от 1 до N

for(int i = 2; i < arr.Length; i++) //так наилучшим вариантом работы быстрой сортировки является, когда массив делится на две равные части, поэтому эта реализация дальше

{

int temp = arr[i]; //обычная замена ( алгоритм задачи в описании будет)

arr[i] = arr[i / 2];

arr[i / 2] = temp;

}

Console.WriteLine(string.Join(" ", arr)); //это вывод элементов массива через пробел

}

}

}

Е. Количество инверсий.

Дана последовательность неповторяющихся целых чисел. Инверсией назовём такую ситуацию, когда большее число стоит впереди меньшего. Например, в последовательности 4 3 2 5 1 всего 7 инверсий: 4 и 3, 4 и 2, 4 и 1, 3 и 2, 3 и 1, 2 и 1, 5 и 1. Ваша задача – определить количество инверсий в заданной последовательности.

**Входные данные**

В первой строке записано целое *N* – длина последовательности (1 ≤ *N* ≤ 50000). В следующей строке через пробел записаны *N* неповторяющихся целых чисел – члены последовательности. Каждое число лежит в диапазоне от 1 до 1000000.

**Выходные данные**

Выведите одно целое число - количество инверсий.

**Пример**

**входные данные**

5  
4 3 2 5 1

**выходные данные**

7

Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый [массив](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один.  
Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по возрастанию подмассива. Тогда:  
3.2. Слияние двух подмассивов в третий результирующий массив.  
На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.  
3.3. «Прицепление» остатка.  
Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

using System;

using System.Collections.Generic;

using System.Linq;

namespace vinnieEight

{

class E\_NumberInvers

{

private static long inverseCount;

public static int[] sort(int[] array)

{

inverseCount = 0; //инициализируем счетчик

if(array == null) return null; //если массив пустой

if(array.Length == 0) return array;

return sort(array, 0, array.Length - 1); //передаме для разделения по частям правую и левую границы массива

}

static int[] sort(int[] array, int low, int high)

{

if(low > high - 1) return new int[] { array[low] };

int mid = low + (high - low) / 2; //разбиваем массив на две части примерно одинакового размера

return merge(sort(array, low, mid), sort(array, mid + 1, high)); //сортруем рекурсивно жве части массива(отдельно)

}

static int[] merge(int[] array1, int[] array2) //функция для слияния

{

int cursor1 = 0;

int cursor2 = 0;

int[] merged = new int[array1.Length + array2.Length]; //размер равен размеру прошлого массива, который мы разделили на две части

int mergedCursor = 0;

while(cursor1 < array1.Length && cursor2 < array2.Length) //проходимся по подмассиввам

{

                                                 //это из первого массива если берем

if(array1[cursor1] <= array2[cursor2]) //берем меньший из элементов подмассивов

merged[mergedCursor++] = array1[cursor1++]; //записываем его в результирующий, увеличиваем счетчики результирующего и подмассива

else

{              //это часть из второго прицепляем

merged[mergedCursor++] = array2[cursor2++];

inverseCount += (array1.Length - cursor1); //число инверсии равно количеству шагов элемента

}

}

//далее прицепление оставшейся части , либо из первого массива, либо из второго

while(cursor1 < array1.Length)

merged[mergedCursor++] = array1[cursor1++];

while(cursor2 < array2.Length)

merged[mergedCursor++] = array2[cursor2++];

return merged;

}

static void Main(string[] args)

{

int N = int.Parse(Console.ReadLine());

int[] arr = Console.ReadLine().Split(' ').Select(s => int.Parse(s)).ToArray();

sort(arr);

Console.WriteLine($"{inverseCount}");

}

}

}

Динамическое программирование

А. Мячик

На вершине лесенки, содержащей *N* ступенек, находится мячик, который начинает прыгать по ним вниз, к основанию. Мячик может прыгнуть на следующую ступеньку, на ступеньку через одну или через две. Например, если пронумеровать ступеньки сверху вниз, а мячик лежит на 5-ой ступеньке, то он может переместиться с неё на 6-ую, 7-ую или 8-ую.

Определить число всевозможных "маршрутов" мячика на землю.

**Входные данные**

Одно целое число *N* (1 ≤ *N* ≤ 40).

**Выходные данные**

Выведите одно целое число – количество маршрутов

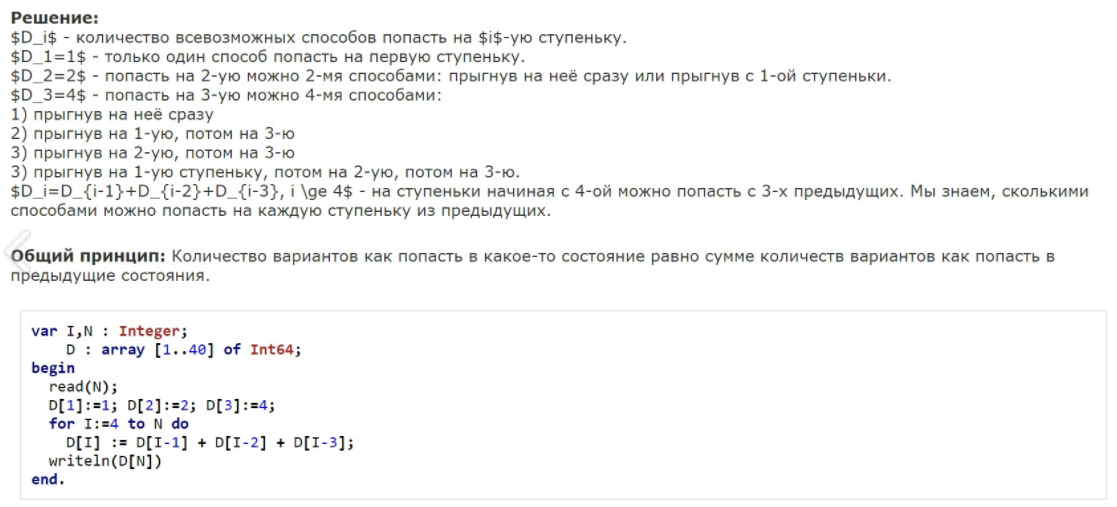
**Пример**

**входные данные**

3

**выходные данные**

4



using System;

using System.Linq;

namespace vinniSeven

{

class A\_Ball

{

static void Main(string[] args)

{

long N = int.Parse(Console.ReadLine());

long[] arr = new long[41];

arr[1] = 1; arr[2] = 2; arr[3] = 4; //количество вариантов первых трех состояний (до них можно легко дойти головой своей, у меня не получилось)

for(int i = 4; i <= N; i++) //начинаем с четвертого состояния

arr[i] = arr[i - 1] + arr[i - 2] + arr[i - 3]; //количесвто вариантов как попасть в какое-то состояние равно сумме количеств вариантов, как попасть в преддущее состояние

Console.WriteLine(arr[N]);

}

}

}

В. Оценки по алгебре

Строгая учительница по алгебре ставит в журнал только двойки, тройки и четвёрки, причём никогда не ставит одному ученику две двойки подряд. Известно, что она поставила Ивану *n* оценок за четверть. Сколькими различными способами она могла это сделать?

**Входные данные**

Дано натуральное 1 ≤ *n* ≤ 30

**Выходные данные**

Выведите ответ на задачу

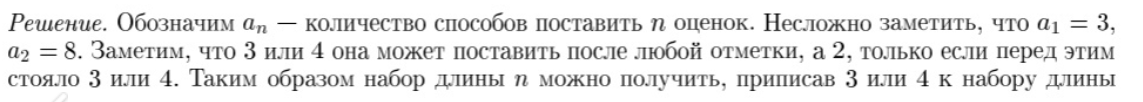
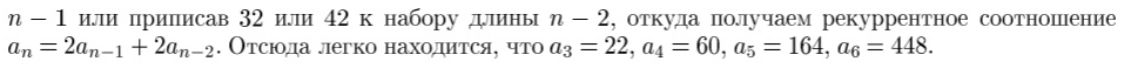
**Пример**

**входные данные**

6

**выходные данные**

448

using System;

using System.Linq;

namespace vinniSeven

{

class A\_Ball

{

static void Main(string[] args)

{

long N = int.Parse(Console.ReadLine());

long[] arr = new long[31];

arr[1] = 3; arr[2] = 8; //если ставит одну оценку, то всего три возможных способа, если две, то восемь(до этого ты тоже легко можешь додуматься исходя из условий задачи)

for(int i = 3; i <= N; i++) //начинаем с трех оценок

arr[i] = 2 \* arr[i - 1] + 2 \* arr[i - 2]; //не ебу, там в решении написано, но я не понимаю

Console.WriteLine(arr[N]);

}

}

}

С. У магазина

На расстоянии *n* шагов от магазина стоит *А*. Каждую минуту он выбирает, куда сделать шаг: к магазину или в противоположном направлении.

Требуется написать программу, которая определит, сколькими способами *А* может попасть в магазин, пройдя ровно *k* шагов и оказавшись в магазине только после выполнения последнего шага.

**Входные данные**

Два числа *n* и *k*, записанные через пробел (1 ≤ *n* ≤ *k* ≤ 37).

**Выходные данные**

Одно целое число – количество способов попадания в магазин

**Пример**

**входные данные**

2 4

**выходные данные**

2

НЕЕБУ

using System;

using System.Linq;

namespace vinniSeven

{

class A\_Ball

{

static void Main(string[] args)

{

int[] input = Console.ReadLine().Split().Select(s => int.Parse(s)).ToArray();

int[] arr = new int[38];

int[] arr2 = new int[38];

arr[input[0]] = 1;

for(int i = 0; i < input[1]; i++)

{

for(int j = 0; j < input[1]; j++)

{

arr2[j] = 0;

if(j > 1) arr2[j] += arr[j - 1];

if(j < input[1]) arr2[j] += arr[j + 1];

}

for(int j = 0; j < input[1]; j++)

arr[j] = arr2[j];

}

Console.WriteLine(arr[0]);

}

}

}