**Задача 1.** Таверна «Доля ангелов»

Где-то посреди одуванчиковых полей стоит, возвышаясь, обдуваемый ветрами, город свободы – Мондштадт. В нём испокон веков стоит таверна с интригующим названием, именуемая «Долей ангелов». Господин Рангвиндр – потомственный хозяин этой таверны. Он захотел узнать, как идут дела в заведении, и поручил своему помощнику в течение N дней записывать количество посетителей таверны. Это поможет узнать, сколько человек посещало «Долю ангелов» за определенные отрезки времени от *l* до *r*.

Помощник господина Рагнвиндра оказался человеком неглупым и придумал алгоритм, с помощью которого можно будет узнать число гостей быстро. Необходимо реализовать этот алгоритм в программе.

**Входные данные.**

На вход подаётся четыре целых, ненулевых, положительных числа: на первой строке вводится N – число дней наблюдения (1 N 365), на следующей строке через пробел подаётся N значений количество посетителей в *n-*ый день (.

За ними следует числоM – количество запросов статистики (1 ), а после М строк со значениями – границами дней сбора статистики ( N, .

**Выходные данные.**

Число – ответ на каждый из запросов, т.е. M значений, представляющих собой количество посетителей за каждый из отрезков от (оба конца отрезка включаются). Каждое значение – с новой строки.

**Пример 1**

*На входе:*  
5  
14 22 17 9 12  
1  
1 5  
*На выходе:*  
74

**Решение.**

Входные данные будет удобно записывать в массив длиной N, создадим его и назовём *array*. Для работы с этим массивом прямо в цикле ввода будет предподсчитываться массив префиксных сумм *pref*, причём размер данного массива будет на единицу больше, чем у исходногопо причине того, что нулевой элемент массива префиксных сумм .

Ввод и вывод данных в программу осуществляется с помощью текстовых файлов input и output.

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

ifstream input("input.txt");

ofstream output("output.txt");

if (!input.is\_open())

{

cout << "ERROR: FILE IS NOT OPEN";

}

else

{

int n, m, y, z, l, r;

int\* array, \* pref;

input >> n;

array = new int[n];

pref = new int[n + 1];

pref[0] = 0;

for (int i = 0; i < n; i++)

{

input >> array[i];

pref[i + 1] = pref[i] + array[i];

}

input >> m;

for (int i = 0; i < m; i++)

{

input >> y >> z;

l = y - 1;

r = z;

output << pref[r] - pref[l] << "\n";

}

}

input.close();

output.close();

return 0;

}

Рассмотрим *группы тестов* для проверки решения данной задачи. Пусть сначала количество запросов будет небольшим. Проверим «крайние» случаи, то есть, например, когда имеется только один элемент в массиве. Второй случай – все значения одинаковы. Третий – значения идут в возрастающем порядке.

|  |  |  |
| --- | --- | --- |
| 1 4  1 1 | 15 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 5 6 9 3 10 10 15 8 11 1 3 | 10 1 2 3 4 5 6 7 8 9 10 4 3 3 6 10 3 4 2 3 |
| 4 | 8 16 12 8 6 | 3 40 7 5 |

Теперь увеличим количество запросов для проверки эффективности алгоритма. Для удобства представления результаты запросов записаны в строчку.

|  |  |  |
| --- | --- | --- |
| 7 35 38 92 50 90 6 43 11 4 4 1 4 5 6 1 2 3 3 1 3 2 4 4 5 1 2 1 1 1 2 | 4 31 36 8 40 7 1 4 3 4 2 3 2 4 1 4 4 4 | 8 22 48 49 7 27 33 0 39 8 3 5 2 4 1 4 1 2 2 8 2 7 4 6 5 8 |
| 50, 215, 96, 73, 92, 165, 180, 140, 73, 35, 73 | 115, 48, 44, 84, 115, 40, 36 | 84, 104, 126, 70, 203, 164, 67, 99 |

*Описание групп тестов:*

№1-3: Массивы данных небольшие, количество запросов не превышает 10.  
№4-6: Для аналогичных массивов данных число запросов не будет превосходить 100.  
№7-25: Алгоритм может применяться и при значительном количестве запросов, до , однако представить в таблице все результаты такого большого числа запросов не представляется возможным, поэтому они будут представлены в отдельном файле.

**Задача 2.** Доходы и расходы

Один из старейших виноградных наделов в Мондштадте – это поместье господина Рагнвиндра. Винокурня «Рассвет» ведёт дела не только в пределах столицы, но и в других регионах: изысканные виноградные напитки, особые сочетания вкусов, смелые пробы ингредиентов – всё это, безусловно, приносит им добрую славу, но также напрямую влияет на доходы и расходы винодельни.

Владелец поместья имеет полезную привычку записывать все свои денежные сделки. Журнал доходов ведётся им в течение N дней и содержит записи о прибыли или расходах.

Господину Рагнвиндру нужно понять, за какой промежуток времени происходит выход «в ноль». Для этого требуется найти среди его записей такой подотрезок [*l, r*], когда сумма трат и доходов становится равной нулю. Если таких несколько, то вывести первый из них. Если таковых не имеется, вывести «EMPTY».

**Входные данные.**

Число дней журналирования N (1 200), далее с новой строки N значений (), соответствующих доходу в денежных единицах, если число положительно, или же расходу, если оно отрицательно, за *i*-ый день. Отсчёт дней начинается с i = 0 (нулевого, а не первого дня).

**Выходные данные.**

Два числа, разделённые пробелом – границы первого найденного нулевого подотрезка, то есть такого, на котором сумма трат и доходов становится равной нулю. Первым найденным подходящим отрезком считается тот, у которого начало раньше.

**Пример 1**

*На входе:*  
5  
1 2 0 -2 5

*На выходе:*  
1 3

**Решение**.

Для организации входных данных образуем массив *a* размером *N.* Для него насчитаем массив префиксных сумм *s,* с помощью которого мы и будем искать интересующий нас подотрезок. Размер массива *s* на единицу больше, чем у массива входных данных, потому что для корректности работы должно быть учтено = 0. Как было сказано ранее в теоретической части, суммы на отрезках — это разности префиксных сумм, то есть для ответа на вопрос задачи необходимо найти два одинаковых элемента в массиве *s* и вывести их индексы.  
  
Поскольку контекст задачи подразумевает довольно большие числа, для массива префиксных сумм был выбран тип данных long с модификатором long. Для того, чтобы вложенный цикл поиска двух одинаковых элементов в массиве префиксных сумм завершился правильно (то есть на первой найденной паре чисел), используется оператор break.

Алгоритм решения данной задачи представлен также в виде блок-схемы (Приложение А). Ввод и вывод данных в программу осуществляется с помощью текстовых файлов input и output.

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

ifstream input("input.txt");

ofstream output("output.txt");

if (!input.is\_open()) { cout << "ERROR: FILE IS NOT OPEN"; }

else

{

int n, k = 0;

input >> n;

long long int \*a, \* s;

int l, r;

a = new long long int[n];

s = new long long int[n + 1];

s[0] = 0;

for (int i = 0; i < n; i++)

{

input >> a[i];

s[i + 1] = s[i] + a[i];

}

for (int j = 0; j < n + 1; j++)

{

for (int i = 1; i < n + 1; i++)

{

if ((s[i] == s[j]) & (i != j))

{

if (i > j) output << j << " " << i - 1;

k++;

break;

}

}

if (k != 0) break;

}

if (k == 0) output << "EMPTY";

}

input.close();

output.close();

return 0;

}  
  
Рассмотрим *группы тестов* для проверки решения данной задачи. Сперва проверим алгоритм на небольшом количестве запросов. Проверим «крайние» случаи, то есть, например, когда имеется только один элемент в массиве. Второй случай – все значения одинаковы. Третий – нулевых подотрезков несколько – это [3, 6] и [8, 10], но выводится только первый.

|  |  |  |
| --- | --- | --- |
| 1 1000000 | 10  53 53 53 53 53 53 53 53 53 53 | 10  2560 2440 3500 8002 -8001 -3501 5000 -4000 6000 -2000 |
| EMPTY | EMPTY | 3 6 |

*Описание групп тестов:*№1-3: значения не превышают по модулю 10000, учтен случай с несколькими подотрезками.

№4-25: вводимые числа существенно увеличены, однако ввиду большого количества данных они будут приведены в отдельном файле.

**Задача 3.** Кристальные бабочки

Недалеко от Мондштадта есть чудесное место, наполненное ароматом трав и теплом вечного лета – Одуванчиковое море. Конечно, поэтичное название «море», данное этому месту бардами, немного преувеличено. На самом же деле это живописная поляна, сплошь покрытая золотистыми цветами. Одуванчиковое море очень приглянулось редкому виду насекомых – кристальным бабочкам.

Поле представляет собой полосу из N условных клеток. На каждой клетке поля живёт определенное количество бабочек *x.* Бабочки не сидят на месте – они то прилетают, то улетают сразу по *y* штук с некоторых отрезков поля вида [*l, r*], где оба конца отрезка включены. Всего таких перелётов – M штук.

Требуется написать программу, которая определит, сколько останется бабочек на каждой из клеток после всех изменений. Считается, что за один запрос с участка от *l* до *r* улетает (или прилетает) одинаковое количество бабочек *y*.

**Входные данные.**

Число клеток на поле N, далее с новой строки вводится N значений (, соответствующих количеству бабочек наклетке. На следующей строке вводится число запросов M, а на последующих M строчках вводятся через пробел два числа: *l* – левая граница диапазона, *r* – правая граница (1 ≤ *l* ≤ *r*, *r* ≤ N) и *y*  – численное изменение бабочек на каждой клетке.

**Выходные данные.**

Количество бабочек на каждой клетке поля после выполнения всех запросов, через пробел.

**Пример 1**

*На входе:*  
3  
34 42 88  
1  
0 2 2  
*На выходе:*  
36 44 90

**Решение**.

Для ответа на запросы о добавлении или вычитании числа на определенном диапазоне удобно использовать массив разниц, назовём его *diff* и после получения основного массива *a* пересчитаем разницу между имеющимися значениями по формуле (1):

Далее воспользуемся свойством массива разниц и изменим в нём пограничные значения: левую границу изменим на величину +x, а правую на обратную величину –х. Таким образом выполним все M запросов, а после по массиву *diff* восстановим интересующий нас изменённый массив, использовав для вычисления *i-*го элемента массива *a* равенство (2), полученное из формулы (1):

Алгоритм решения данной задачи представлен также в виде блок-схемы (Приложение Б). Ввод и вывод данных в программу осуществляется с помощью текстовых файлов input и output.

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

ifstream input("input.txt");

ofstream output("output.txt");

if (!input.is\_open()) { cout << "ERROR: FILE IS NOT OPEN"; }

else

{

int n, k = 1, l, r, x, m;

input >> n;

int\* a = new int[n];

for (int i = 0; i < n; i++)

{

input >> a[i];

}

int\* diff = new int[n + 1];

diff[0] = a[0];

diff[n] = 0;

for (int i = 1; i < n; i++)

{

diff[i] = a[k] - a[k - 1];

k++;

}

input >> m;

for (int j = 0; j < m; j++)

{

input >> l >> r >> x;

diff[l-1] += x;

diff[r] -= x;

for (int i = 0; i < n; i++)

{

if (i == 0) a[i] = diff[i];

else a[i] = diff[i] + a[i - 1];

}

}

for (int i = 0; i < n; i++)

{

output << a[i] << " ";

}

}

input.close();

output.close();

return 0;

}

Рассмотрим *группы тестов* для проверки решения данной задачи. Сперва проверим алгоритм на небольшом количестве запросов. Проверим «крайние» случаи, то есть, например, когда имеется только один элемент в массиве. Второй случай – все значения одинаковы. Третий – значения идут в убывающем порядке.

|  |  |  |
| --- | --- | --- |
| 1  19  1  1 1 2 | 11  8 8 8 8 8 8 8 8 8 8 8  2  1 5 -2  1 5 -3 | 7  88 84 56 50 16 3  3  1 6 14  1 2 -10  2 3 1 |
| 21 | 3 3 3 3 3 8 8 8 8 8 8 | 88 84 56 50 16 4 4 |

*Описание групп тестов:*№1-3: количество запросов невелико, как и число клеток поля.

№4-25: число запросов существенно больше и достигает , однако ввиду большого количества данных они будут приведены в отдельном файле.

**Задача 4.** Винокурня «Рассвет»

Господин Рагнвиндр давным-давно получил в наследство от отца винокурню, и вполне успешно справляется с ней – семейное дело процветает и приносит винные дары во все уголки Мондштадта. На винокурне имеется поле размерами MxN условных клеток, засаженное различными сортами винограда. Со временем виноградные лозы разрослись, ведь они такие же свободолюбивые, как и сама страна свободных ветров. Теперь на каждой условной клетке поля имеется различное количество виноградных лоз.

Владельцу винокурни стало однажды любопытно, насколько рационально работает его надел, и он сосчитал, сколько на каждой клетке поля имеется виноградных лоз. Известно, что чётное количество лоз в полтора раза плодороднее, нежели нечётное. Необходимо узнать, какая часть поля является самой большой и при этом самой плодородной, то есть вывести координаты двух её противоположных краёв. Если таких участков несколько, вывести первый из них.

**Входные данные.**

Размеры поля M, N, далее M\*N значений , соответствующих количеству лоз наклетке с координатами *i* и *j*.

**Выходные данные.**

Четыре целых числа, разделенные пробелом – координаты краёв самого плодородного участка в следующем порядке: сначала координаты левого верхнего угла (, затем координаты правого нижнего (.

**Пример 1**

*На входе:*2 3  
1 1 0 0 0 0 *На выходе:*1 1 2 1

**Решение**.

Поле размером MxN образует двумерный массив *array*, каждому элементу которого соответствует количество лоз на клетке. Если предпосчитать массив префиксных сумм *pref* для исходной матрицы, далее по нему будет удобно узнавать, является ли суммарное количество лоз на определенной площади поля чётным или нечётным – и ставить в соответствии этим площадям некоторое число k – коэффициент плодородности, равное количеству лоз n, умноженных на полтора, если n чётное, или умноженных на единицу (т.е. остаётся равным n). Таким образом в цикле можно перебрать все возможные площади и определить, какая из них наиболее полезна и плодородна.   
  
Если большая часть клеток массива *array* ненулевые, может оказаться так, что всё поле целиком суммарно производительнее, чем отдельно взятые участки. Если же, напротив, нулевых клеток много, то отдельный участок, наоборот, будет плодороднее целого.

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

ifstream input("input.txt");

ofstream output("output.txt");

if (!input.is\_open()) { cout << "ERROR: FILE IS NOT OPEN"; }

else

{

int\*\* array, \*\* pref, m, n, c, count\_null = 0, sum\_g = 0, sum\_v = 0, max\_s = 0, x1 = 1, y1 = 1, x2 = 1, y2 = 1, l, r;

int b\_x1, b\_y1, b\_x2, b\_y2;

input >> m >> n;

array = new int\* [m];

pref = new int\* [m + 1];

for (int i = 0; i < m; i++) array[i] = new int[n];

for (int i = 0; i < m + 1; i++) pref[i] = new int[n + 1];

for (int i = 0; i < n + 1; i++) pref[0][i] = 0;

for (int j = 0; j < m + 1; j++) pref[j][0] = 0;

for (int i = 0; i < n; i++)

{

sum\_g = 0;

for (int j = 0; j < m; j++)

{

input >> array[j][i];

sum\_g += array[j][i];

pref[j + 1][i + 1] = sum\_g + pref[j + 1][i];

}

}

while (x1 <= m)

{

y1 = 1;

while (y1 <= n)

{

if (array[x1 - 1][y1 - 1] == 0) { y1++; continue; }

while (x2 <= m)

{

y2 = 1;

while (y2 <= n)

{

if ((x1 > m) || (y1 > n) || (x2 > m) || (y2 > n)) break;

c = pref[x2][y2] - pref[x1 - 1][y2] - pref[x2][y1 - 1] + pref[x1 - 1][y1 - 1];

if (c % 2 == 0) c = (int)(c \* 1.5);

if (c > max\_s)

{

max\_s = c;

b\_x1 = x1; b\_y1 = y1; b\_x2 = x2; b\_y2 = y2;

}

y2++;

}

x2++;

}

y1++;

}

x1++;

}

output << b\_x1 << " " << b\_y1 << " " << b\_x2 << " " << b\_y2;

}

input.close();

output.close();

return 0;

}  
Рассмотрим *группы тестов* для проверки решения данной задачи. Сначала количество нулевых и ненулевых клеток приблизительно одинаковое.

|  |  |  |
| --- | --- | --- |
| 5 5  0 0 0 0 0  2 2 2 4 2  0 1 1 1 1  3 3 0 0 0  0 0 7 2 2 | 3 5  2 4 2  0 0 1 2 8 4  0 2 2 0 0 2 | 3 4  345 7 22  50 515 6  35 6 2  0 43 0 |
| 1 2 4 5 | 1 1 3 5 | 1 1 3 3 |

В следующей группе тестов количество нулевых клеток преобладает.

|  |  |  |
| --- | --- | --- |
| 5 3  0 0 0 0 0  0 0 150 151 0  0 0 0 0 0 | 3 5  2 4 6  8 2 0  0 0 0  0 0 0  1 1 1 | 4 2  0 0 2 2  0 0 4 4 |
| 3 2 4 2 | 1 1 3 2 | 3 1 4 2 |

Все или почти все клетки заполнены числами больше нуля.

|  |  |  |
| --- | --- | --- |
| 7 4  45 21 12 28 28 49 30  41 11 29 46 50 22 41  9 1 30 25 47 25 31  27 38 21 23 41 45 33 | 6 5  0 21 1 43 26 48  40 29 18 9 45 4  28 47 45 5 19 43  30 19 26 50 3 2  11 8 28 24 6 22 | 5 7 35 7 22 45 40 50 15 6 37 7 35 6 2 27 8  0 43 16 0 22  5 33 27 19 36  33 3 23 28 6  14 20 39 48 3 |
| 1 1 6 4 | 1 2 5 5 | 1 1 5 7 |

*Описание групп тестов:*№1-3: количество нулевых и ненулевых клеток приблизительно одинаковое, результат охватывает большую часть поля.

№4-6: количество нулевых клеток преобладает, результат является отдельным участком поля.  
№7-9: все клетки заполнены числами, результат охватывает большую часть поля.

№10-25: вводимые числа существенно увеличены, однако ввиду большого количества данных они будут приведены в отдельном файле.