**!!!!** в C++ функция std::partial\_sum, которая как раз таки считает префиксные суммы

Продолжим увеличивать количество запросов. Для удобства представления входные данные запросов будут приведены в таблице, а результаты запросов – в строчку.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 72 58 14 22 8 20 74 52 95 96 10 6 8 2 3 2 3 1 2 5 5 9 10 6 10 1 1 3 5 1 3 | 5 20 29 60 94 94 20   |  |  | | --- | --- | | 2 2 1 1  1 2  2 4  2 4  1 1  1 1  1 5  2 2  3 4 | 1 1  3 3  3 3  5 5  1 2  1 3  1 1  4 5  1 3  2 5 | | 14 41 7 42 32 33 20 38 30 26 24 37 13 37 31 14   |  |  | | --- | --- | | 2 2 2 3  5 8  13 14  2 14  3 14  4 14  5 14  6 14  7 14 | 8 14  2 7  3 10  4 13 | |
| 146, 72, 72, 130, 8, 191, 337, 72, 44, 144 | 29, 20, 49, 183, 183, 20, 20, 297, 29, 154, 20, 60, 60, 94, 49, 109, 20, 188, 109, 277 | 7, 49, 121, 68, 370, 363, 321, 289, 256, 236, 198, 172, 245, 290 |

Введение

1 Постановка задачи

2 Описание исследуемого (-ых) алгоритма (-ов) (Теоретическая часть)

3 Возможности применения алгоритма

4 Решение задач с использованием алгоритма

Заключение

Список литературы

Приложения

Введение должно отражать историю рассматриваемого вопроса или ме-тода, его теоретическую и практическую значимость (объем введения – ок. 1 страницы).

Основная часть содержит постановку задачи, изложение теоретических положений с необходимыми выводами, варианты реализации алгоритма для решения задач, описание разработанных задач для демонстрации применения алгоритма, обоснование и описание разработанных тестов для проверки рабо-тоспособности решений.

Пример оформления основной части КР приведен в приложении. Если задание на КР включает составление задач по изучаемым алгоритмам, то опи-сание задач должно включать следующие части: условие, описание входных и выходных данных, алгоритм решения, решение на одном или двух языках про-граммирования, примеры входных и выходных данных, описание групп тестов для проверки работоспособности решения.

Заключение должно содержать краткие выводы по работе, оценку полу-ченных результатов. В нем отмечается, какие новые знания и навыки получены студентом при выполнении курсовой работы, указываются перспективы разви-тия, мероприятия по совершенствованию разработанных алгоритмов, компо-нентов и т.д.

В приложения включаются таблицы с исходными данными, тексты про-грамм, громоздкие результаты работы программы или результаты, не представ-ленные в разделе, схемы алгоритмов.

**Определение.** Пусть дан массив A: [​, ​, ​, …, ​]. Для него можно подсчитать такие суммы, что k-тая сумма равна сумме первых k элементов данного массива, то есть с до . Такие *префиксные суммы* имеют вид = + *,* причём обязательно = 0, а =. Их удобно хранить в массиве S: [​, ​, ​, …, ], причём его длина всегда на единицу больше, чем длина первоначального массива A.

Можно представить, что элементы массива находятся в ячейках, а префиксные суммы находятся между ними — на перегородках, и содержат в себе суммы всего того, что находится перед этой перегородкой.



#include <iostream>

#include <stdlib.h>

#include <iomanip>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

int n;

cout << "Введите длину массива: "; cin >> n;

int \*a,\*s;

a = new int[n];

s = new int[n + 1];

s[0] = 0;

srand((unsigned)time(NULL));

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

{

a[i] = rand() % 100;

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

cout << a[i] << setw(5);

}

cout << "\nМассив префиксных сумм:\n";

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

{

cout << s[i] << setw(5);

}

return 0;

}

Префиксные суммы могут использоваться в задачах типа «найти сумму *на полуинтервале* с позиции *l* до позиции *r*». Для *l* = 0 ответ уже просчитан заранее и равен , однако если в задаче *l* ≠ 0, то справедливо равенство***:***

В префиксной сумме содержатся все нужные элементы от до, однако есть ещё и лишние:​, , …, ​. Эта сумма в свою очередь равна уже посчитанной префиксной сумме ​. ​Тогда для ответа на вопрос задачи о поиске суммы на произвольном полуинтервале нужно просто вычесть друг из друга две предподсчитанные префиксные суммы. Использовать префиксные суммы удобно, поскольку получается, что суммы на отрезках — это разности префиксных сумм, но только если массив в ходе запросов не меняется. Потому что, если какой-то элемент массива поменялся, нужно пересчитать все префиксные суммы, в которые он входит, а это долго.

**Пример 1.** Дан массив. Найти сумму на отрезке [*l , r*].

Задача о поиске суммы *на отрезке* сводится к поиску на полуинтервалах путём исключения левого конца (*l*). Полуинтервалы использовать удобнее в виду отсутствия необходимости включать второй конец и учитывать единицу. В данном примере программы массив исходных данных имеет длину n, а значения *l* и *r* лежат в пределе [0, n-1], причём *l < r*, поскольку *l* – левый конец отрезка.

#include <iostream>

#include <stdlib.h>

#include <iomanip>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

int n, sum=0, r,l;

cout << "Введите длину массива: "; cin >> n;

int \*a,\*s;

a = new int[n];

s = new int[n + 1];

s[0] = 0;

srand((unsigned)time(NULL));

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

{

a[i] = rand() % 100;

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

cout << a[i] << setw(5);

}

cout << "\nМассив префиксных сумм:\n";

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

{

cout << s[i] << setw(5);

}

cout << "\nЗапрос суммы на отрезке. Введите отрезок [l, r]:\n";

cout << "l = "; cin >> l;

cout << "r = "; cin >> r;

sum = s[r+1] - s[l]; //сумма на полуинтервале (l-1,r]

cout << "Сумма на отрезке [l, r] равна ";

cout << sum;

return 0;

}

**Пример 2.** Дан массив. Необходимо найти любой его непустой подотрезок с нулевой суммой элементов.

**Решение.** Поскольку суммы на отрезках — это разности префиксных сумм, то если сумма на отрезке равна нулю, это равносильно тому, что префиксные суммы его концов равны.



Таким образом, задача сводится к нахождению подотрезка нулевой суммы, т.е. к задаче нахождения двух одинаковых элементов в массиве S префиксных сумм.

#include <iostream>

#include <stdlib.h>

#include <iomanip>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

int n, k=0;

cout << "Введите размер массива: "; cin >> n;

int \*a, \*s, l, r;

a = new int[n];

s = new int[n + 1];

s[0] = 0;

srand((unsigned)time(NULL));

if (a!=NULL)

{

cout << "\nМассив создан успешно:\n";

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

{

a[i] = -10+rand() % 21;

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

cout << setw(5) << a[i];

}

cout << "\nМассив префиксных сумм:\n";

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

{

cout << setw(5) << s[i];

}

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

{

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

{

if ((s[i] == s[j])&(i!=j)&(s[i]!=0)&(s[j]!=0))

{

if (i>j) cout << "\nНепустой подотрезок с нулевой суммой находится на [" << j << ", " << i-1 << "].\n";

else cout << "\nНепустой подотрезок с нулевой суммой находится на [" << i << ", " << j-1 << "].\n";

k++;

break;

}

if (k != 0) break;

}

}

if (k == 0) cout << "\nНепустых подотрезков с нулевой суммой нет.\n";

}

return 0;

}

Префиксные суммы могут быть не только «суммами». Необходимым условием является только тот факт, что функция, которая считается на отрезке, является обратимой, то есть для неё должна быть реализуема возможность по двум префиксам восстановить значение на отрезке. Операция суммы обратима, потому что, если мы прибавили лишнее, потом это можно отнять обратно.

Кроме суммы одна из самых популярных операций – это операция «побитового исключающего или», которая еще называется XOR и обозначается ⊕. При этом для неё выполнено тождество x ⊕ x = 0 для любого числа x, что означает, что операция XOR обратна сама себе, так что формула вычисления побитового исключающего или на отрезке получается такая:

**Определение.** По массиву префиксных сумм S: [​, ​, ​, …, ] можно восстановить исходный A: [​, ​, ​, …, ​], который при этом будет являться *разностным массивом* для S и определяться следующим образом:   
=

=

=

…

=

Таким образом получается, что формула= — это просто преобразованная рекуррентная формула для поиска префиксных сумм: . Проводя аналогию с математикой, можно сделать вывод о том, что переход к разностному массиву — это дискретное дифференцирование, а переход к массиву префиксных сумм — дискретное интегрирование.

Помимо префиксных сумм можно также рассмотреть *суффиксные суммы*, которые представляю из себя суммы концов, а не начал массива.

=

=

=

...

=

С ними всё аналогично: их можно вычислить аналогичным циклом (только теперь он пойдёт с конца), и их можно точно так же использовать для вычисления суммы на отрезке, если принять .

Префиксные суммы работают не только для одномерных, но и для многомерных массивов. Пусть дан двумерный массив , тогда массив его префиксных сумм *B* можно найти, подсчитав сперва префиксные суммы на одномерных массивах, а потом насчитать префиксные суммы уже на них, т.е. провести сложение сперва по горизонталям, а после сложить полученные суммы по вертикалям:



Рекуррентная формула для

Иначе можно рассмотреть это так: сперва мы берём сумму двух меньших префиксных сумм, которые перекрывают по площади своих прямоугольников искомую, однако их пересечение ( учитывается дважды, поэтому его вычитаем и прибавляем новый элемент — .

Это также можно записать условным образом с помощью коэффициентов x, y, z, где ***x*** – сумма по i-й строке и j-му столбцу ( – сумма по i-й строке и (j+1)-му столбцу ( – сумма по (i+1)-й строке и j-му столбцу ( т.е. .



Таким образом, если преобразовать рекуррентную формулу с учётом уже насчитанных сумм, получится: .

Элементы, через которые считается очередной  имеют меньшие индексы, поэтому подсчет двумерных префиксных сумм можно вести просто двумя вложенными циклами по возрастанию. Массив префиксных сумм двумерного массива можно хранить и одномерно.

#include <iostream>

#include <stdlib.h>

#include <iomanip>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

srand((unsigned)time(NULL));

int \*\*a, \*\*s, m, n, k = 0,sum\_g=0,sum\_v=0;

cout << "Введите размеры массива MxN: "; cin >> m >> n;

a = new int\*[m];

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

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

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

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

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

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

{

sum\_g = 0;

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

{

a[j][i] = rand() % 10;

cout << setw(3) << a[j][i];

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

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

}

cout << "\n";

}

cout << "\nМассив префиксных сумм данного массива:\n";

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

{

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

{

cout << setw(3) << s[j][i];

}

cout << "\n";

}

return 0;

}

**Задача 2.** Милка и скачки

Сэр Кэйа – капитан кавалерии Мондштадта. На праздник урожая в городе по традиции проводятся скачки, и ему, как рыцарю, необходимо достойно пройти это соревнование. В связи с этим сэр Кэйа со своей лошадью по имени Милка тренируют выносливость на ипподроме перед скачками. До соревнований у них остаётся N дней (1 ≤ N ≤ 100). Каждую тренировку они проходят определённую дистанцию S км (0 ≤ S ≤ 50). Всадник всегда записывает пройденное расстояние.

У Милки обязательно есть выходной: она отдыхает и пасётся на лугу каждый третий день, соответственно, в этот день пройденная дистанция равна нулю.

Необходимо придумать, как быстро на основе данных о пройденном расстоянии определить, является ли динамика прогресса положительной, нейтральной или отрицательной (то есть определить, каких чисел в ней получилось больше – отрицательных, положительных или нулей), а также вычислить для контроля общее пройденное расстояние со второго дня по *k*-ый (k < N), а также .

**Входные данные**: N – число дней наблюдения (7 ≤ N ≤ ), далее с новой строки N значений через пробел – длины дистанций S (0 ≤ S ≤ 50).

**Выходные данные**: вывести «POSITIVE», если динамика положительна, в противном случае, если динамика отрицательна, вывести «NEGATIVE», а если она нейтральна, тогда вывести «NEUTRAL». На следующей строке вывести пройденное расстояние за последние 3 дня.  
  
**Пример 1**

*На входе:*  
5  
4 6 7 7   
*На выходе:*  
POSITIVE  
14

**Решение.** Ввод значений длины дистанции в массив *S* удобно организовать в цикле. Одновременно с вводом будет заполняться второй массив префиксных сумм *pref*, в котором будет подсчитываться общее пройденное расстояние. Он необходим в данной задаче для ответа

Также будет организован третий массив *dynamic*, который будет отвечать за разницу между расстояниями, т.е. за динамику прогресса в дни тренировок. В массив *dynamic* не входит значение нулевого дня, а также не учитываются выходные дни, поэтому его размер меньше, чем у *pref* и *s,* и равен (то есть от берётся только целая часть)

В данном случае k – счётчик пройденных дней, необходимый для корректного вывода дат, поскольку отсчёт дней идёт с первого, а отсчёт элементов массива – с нулевого. В свою очередь переменная d – счётчик активных дней тренировок, необходимый для массива результатов *dynamic.*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;

input >> n;

int\* s, \* dynamic, \* pref, plus = 0, minus = 0, neutral = 0, k, d = 1;

s = new int[n];

pref = new int[n + 1];

dynamic = new int[n - (int)n / 3];

pref[0] = 0;

dynamic[0] = 0;

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

{

k = i + 1;

if (k % 3 == 0) { s[i] = 0; }

else

{

input >> s[i];

if (i > 0)

{

if ((k - 1) % 3 == 0) dynamic[d] = s[i] - s[i - 2];

else dynamic[d] = s[i] - s[i - 1];

if (dynamic[d] > 0) plus++;

else

{

if (dynamic[d] < 0) minus++;

else neutral++;

}

d++;

}

}

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

}

if ((plus > minus) & (plus > neutral)) output << "POSITIVE\n";

else

{

if ((neutral > plus) & (neutral > minus)) output << "NEUTRAL\n";

if ((plus < minus) & (minus > neutral)) output << "NEGATIVE\n";

}

if ((n % 3 != 0) & ((n - 1) % 3 != 0) & ((n - 2) % 3 != 0)) output << pref[n] - pref[n - 2];

else output << pref[n] - pref[n - 3];

}

return 0;

}

Рассмотрим *группы тестов* для проверки решения данной задачи. Пусть сперва прирост пройденных расстояний будет незначителен.

|  |  |  |
| --- | --- | --- |
| 10 5 5 5 4 3 5 5 | 11 7 6 8 8 8 8 8 8 | 12 3 3 3 5 5 5 5 5 |
| NEUTRAL 10 | NEUTRAL 16 | NEUTRAL 10 |

Следующая группа тестов будет иметь более выраженный характер прироста расстояний.

|  |  |  |
| --- | --- | --- |
| 30 2 2 3 2 3 5 7 10 8 7 10 11  15 15 13 16 17 18 16 15 |  |  |
| POSITIVE 31 |  |  |

Наблюдается отрицательный рост, т.е. расстояния больше уменьшаются, чем растут.

|  |  |  |
| --- | --- | --- |
| 15 10 9 8 7 5 5 5 4 3 3 | 7 8 6 5 5 4 | 24 15 16 17 17 17 16 14 13 11 10 10 12 12 11 10 8 |
| NEGATIVE 6 | NEGATIVE 9 | NEGATIVE 18 |

*Описание групп тестов:*№1-3: пройденные расстояния сопоставимы, преобладает нейтральная динамика  
№4-6: дистанции возрастают, преобладает положительная динамика  
№7-9: пройденные расстояния уменьшаются, наблюдается отрицательная динамика.  
№10-25: алгоритм тестируется на большем количестве входных данных, но поскольку представить все результаты такого большого числа запросов не представляется возможным, они будут представлены в отдельном файле.

**Задача 2.** Юноша по имени Антон копит деньги на исполнение своей мечты – тур по городам со своим юмористическим шоу. До цели ему осталось ещё 23 000 рублей. Если стоит задача накопить определенную денежную сумму, то необходимо с умом распределять свой бюджет – это правило он знает отлично. Некоторые траты избежать нельзя, например, арендную плату за съём жилья и расходы на еду. Примем необходимые траты в месяц за прожиточный минимум – 12 000 рублей. Записывать всё в блокнот и считать вручную скучно и долго, поэтому Антон обратился за помощью к другу-программисту Диме, и тот написал для него программу учёта финансов.

Однако эта программа оказалась несовершенной и не смогла учесть тот факт, что Антон помимо основной работы начал подрабатывать официантом в кафе, где каждую смену получал почасовую оплату – 200 рублей в час. Также программа Димы не имела графы дополнительных расходов и пропускала транспортные расходы или тот факт, что иногда Антону хочется купить себе что-нибудь.

Задача программы – проанализировать доходы и расходы Антона в течение недели и вычислить средний недельный доход и расход, а также подсчитать, какую максимальную сумму в неделю он сможет откладывать, чтобы побыстрее накопить на поездку, и сколько ещё ему понадобится для этого месяцев (в одном месяце 4 недели). Также Антон хочет узнать, как быстро он сможет накопить нужную сумму, если будет стабильно подрабатывать по 4 часа пять дней в неделю.

**Входные данные:** X – основной доход Антона в рублях (12 000 < X < 25 000). Далее вводится значения в следующем формате для каждого из семи дней:

А) *y*, где *y –* количество часов подработки за текущий день (0 .

Б) *z,* где *z –* дополнительные расходы в рублях.

Для удобства ввод осуществляется из файла.

**Выходные данные:** максимальная сумма, которую Антон сможет отложить при имеющихся расходах.

**Решение.** Из основного заработка X сразу можно вычесть 12 000 рублей. Далее на вход в программу будут поступать числа *y,* умножив которые на 300 (з/п за час), будем получать доход за день, а также числа *z –* расходы за день. Если из *y\**200 вычесть число *z,* можно получить **Δ** – изменение состояния финансов.

Параллельно вводу будет насчитываться массив префиксных сумм *pref,* в котором будет указано текущее состояние финансов, то есть (X – 12 000) *+*Δ. В конце недели, что будет соответствовать последнему элементу массива *pref,* получится остаток средств, который и будет являться максимально возможной суммой, которую можно отложить в копилку.

#include <iostream>

#include <stdlib.h>

#include <fstream>

#include <iomanip>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

ifstream input("task2.in.txt");

if (!input.is\_open())

{

cout << "ФАЙЛ НЕ ОТКРЫТ.";

}

else

{

int x, y, z, sr\_minus = 0, sr\_plus = 0, k = 1;

input >> x; cout << "ЦЕЛЬ: 23000 рублей.\nНАЧИСЛЕНИЕ ЗАРПЛАТЫ: " << x;

int\* pref1, \* pref2;

pref1 = new int[29];

pref2 = new int[29];

pref1[0] = x - 12000;

pref2[0] = pref1[0];

for (int j = 0; j < 4; j++)

{

if (j>0) x = pref1[k];

for (int i = 0; i < 7; i++)

{

cout << "\nНЕДЕЛЯ " << j+1 << ". ДЕНЬ " << i + 1;

input >> y; //cout << "\nДОХОДЫ (часы подработки): " << y;

input >> z; //cout << "\nРАСХОДЫ (в рублях): " << z;

sr\_minus += z;

sr\_plus += y \* 300;

pref1[k] = pref1[k-1] + (y \* 300 - z);

if (k%5 > 0) pref2[k] = pref2[k-1] + (4\*300 - z);

if (k%5 <= 0) pref2[k] = pref2[k-1] - z;

cout << "\nСОСТОЯНИЕ БЮДЖЕТА НА ТЕКУЩИЙ ДЕНЬ (в рублях): " << pref1[k] << "\n";

k++;

}

cout << "\n-----------------------------------------\n";

}

sr\_minus /= 4;

sr\_plus /= 4;

cout << "\nСРЕДНИЙ РАСХОД ЗА МЕСЯЦ (в рублях): " << sr\_minus;

cout << "\n\n\nСОСТАВЛЕНИЕ ПРОГНОЗОВ...\n\n";

cout << "\nСРЕДНИЙ ЗАРАБОТОК ЗА МЕСЯЦ (в рублях): " << sr\_plus - sr\_minus;

cout << "\n--> ДИНАМИКА БЮДЖЕТА НА ОСНОВАНИИ ИМЕЮЩИХСЯ ДАННЫХ:\n";

for (int i = 0; i < 29; i++)

{

cout << setw(9) << pref1[i] << setw(9);

if (i % 6 == 0) cout << "\n";

}

if ((23000 / (4 \* (sr\_plus - sr\_minus))) > 0)

cout << "\nПРИ ТЕКУЩИХ ДОХОДАХ И РАСХОДАХ ПОТРЕБУЕТСЯ " << 23000 / (4 \* (sr\_plus - sr\_minus)) << " МЕСЯЦ(ЕВ).\n";

else cout << "\nПРИ ТЕКУЩИХ ДОХОДАХ И РАСХОДАХ ПОТРЕБУЕТСЯ ПОТРЕБУЕТСЯ МЕНЬШЕ МЕСЯЦА.";

cout << "\n\nСРЕДНИЙ ЗАРАБОТОК ЗА МЕСЯЦ (в рублях): " << 300\*20 - sr\_minus;

cout << "\n\n--> ОЖИДАЕМАЯ ДИНАМИКА БЮДЖЕТА ПРИ РАБОТЕ 4 ЧАСА 5 ДНЕЙ В НЕДЕЛЮ:\n";

for (int i = 0; i < 29; i++)

{

cout << setw(9) << pref2[i] << setw(9);

if (i % 6 == 0) cout << "\n";

}

if ((23000 / (4 \* (300 \* 20 - sr\_minus))) > 0)

cout << "\nПРИ РАБОТЕ 4 ЧАСА 5 ДНЕЙ В НЕДЕЛЮ И ТЕХ ЖЕ РАСХОДАХ ПОТРЕБУЕТСЯ " << 23000 / (4 \* (300 \* 20 - sr\_minus)) << " МЕСЯЦ(ЕВ).\n";

else cout << "\nПРИ РАБОТЕ 4 ЧАСА 5 ДНЕЙ В НЕДЕЛЮ И ТЕХ ЖЕ РАСХОДАХ ПОТРЕБУЕТСЯ МЕНЬШЕ МЕСЯЦА.\n\n";

input.close();

}

return 0;

}

Задача 3. На двумерный массив

**Источники:**

<https://peltorator.ru/posts/prefix_sums/>

https://notes.algoprog.ru/shortideas/03\_x\_prefix\_sums.html

<https://www.youtube.com/watch?time_continue=86&v=5iW84xlL0j0&feature=emb_logo>

https://peltorator.ru/cp\_book.pdf

https://foxford.ru/wiki/informatika/odnomernoe-dinamicheskoe-programmirovanie-kolichestvo-sposobov