Алгоритмы и структуры данных. Линейные алгоритмы

densine, w1nSkEeper

5 октября 2025 г.

Содержание

1. Префиксы/суффиксы

1.1. Определение

Пусть у нас есть массив из n чисел - например $[a_1, a_2 \dots a_n]$, а также q запросов $[l, r]$. Для каждого запроса вывести:

$$a_l + a_{l+1} + \dots + a_r$$

Как нам эффективно считать такую сумму?

- Если мы будем пересчитывать ее каждый раз, то выйдет сложность $O(n * q)$. Скажем, слишком долго.
- Введем понятие префикс массива - отрезок от 0 до текущего элемента массива – $a_1, a_2 \dots a_r$. Суффикс – аналогино от n до текущего элемента. Что мы можем делать с этим префиксом? Посчитать его сумму. Из его определения следует, что префиксная сумма для k равна префиксной сумме для $k - 1$ плюс a_k.

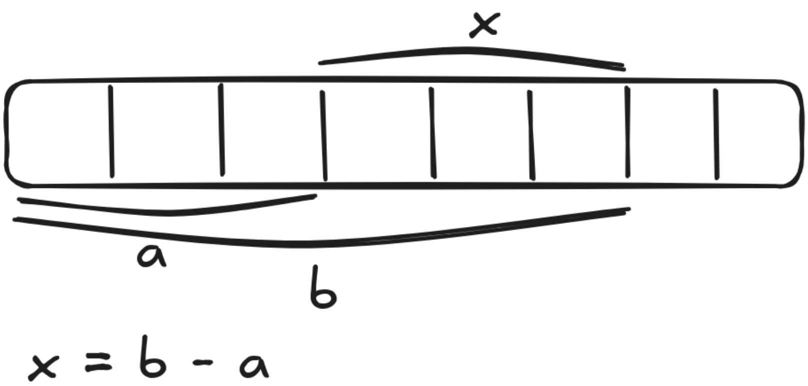
```
vector<int> pref(n + 1);
// pref[i] - сумма на [0;i), т.е. i - длина префикса
for (int i = 0; i < n; ++i) {
    pref[i + 1] = pref[i] + a[i];
}
```

Пример: есть массив $x = [1, 6, 7, 3, 5, 8]$. Тогда $pref_x = [0, 1, 7, 14, 17, 22, 30]$.

Затем, с помощью префиксных сумм можно очень быстро находить суммы любых сплошных отрезков в массиве! Нужно просто вычесть из одной суммы другую - т.е. из отрезка длины r отрезок длины l. Тогда из суммы первых r элементов мы вычтем сумму первых l элементов. Логично, что останется сумма элементов $(l, r]$.

1.2. Код

Выглядит это как-то так:



Или с помощью кода:

```
for (int i = 0; i < q; ++i) {
    int l, r;
    std::cin >> l >> r;
    l--;
    std::cout << pref[r] - pref[l] << std::endl; // '\n'
}
```

Дальше шло 5-10 минут распинания о том, что использование символа новой строки лучше чем $std :: endl$. По-моему же, если ваш код не справляется со временем, то никакое отсутсвие $std :: endl$ вам не поможет :).

Если вы вдруг задумали использовать этот алгоритм в задаче, в которой меняется массив - имейте ввиду, что по идее после каждой операции все нужно будет пересчитывать.

Префиксы можно использовать и для хранения произведения. Также для максимумов, но тогда их и использовать как максимумы префиксов.

Итак, сложность такого алгоритма выходит $O(n + q)$.

1.3. Примеры

- Дан массив $a_1, a_2 \dots a_n$. Найти максимальную сумму подотрезка (l, r).

  ```
  pref[r] - pref[l] // Это значение должно быть максимальным;
  ```

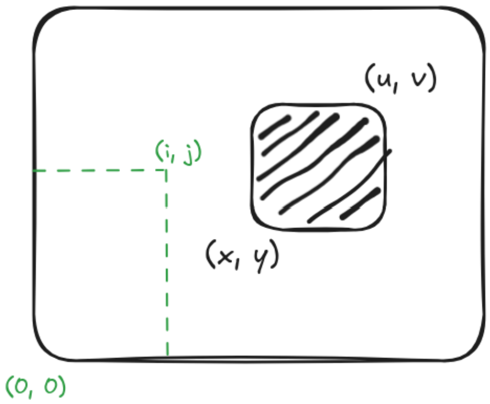
  r перебираем. Берем l <= n такое, что pref[l] - min. Полный код:

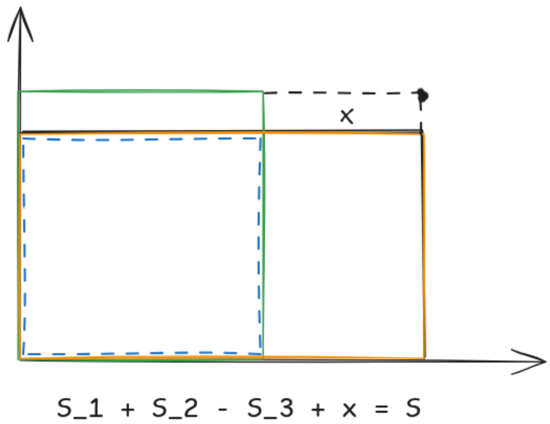
  ```
  int ans = 0;
  int min_pref = 0;
  for (int i = 0; i < n; ++i) {
  ```

```
    min_pref = std::min(min_pref, pref[i + 1]);
    if (pref[i + 1] - min_pref > ans) {
        ans = pref[i + 1] - min_pref;
    }
}
```

1.4. Двумерные префиксные суммы

Пусть нам дана матрица n x m. Запросы имеют вид x y u v. Необходимо найти сумму элементов в подматрице от (x, y) до (u, v). Для удобства иллюстрации площадью будем называть сумму всех чисел в подматрице от (x, y) до (u, v).

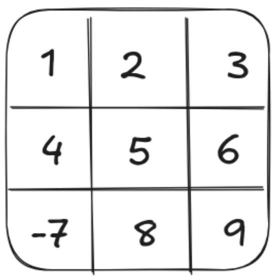


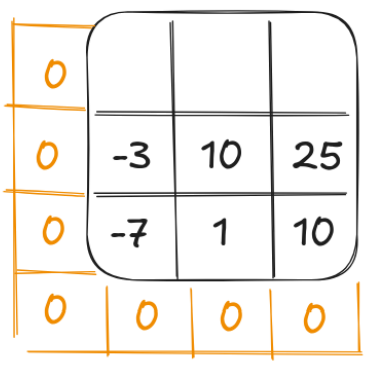


Для клетки (i,j) считаем сумму как:

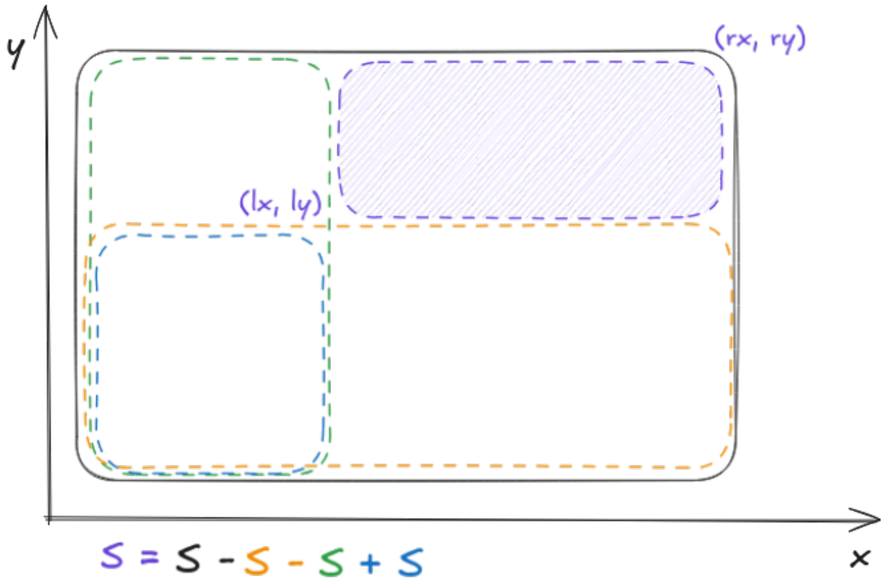
```
pref[i][j] = x[i][j] +
    pref[i - 1][j] + pref[i][j - 1] - pref[i - 1][j - 1];
```

где $x[i][j]$ соотвествует сумме белого прямоугольника, $pref[i-1][j]$ площади зеленого, $pref[i][j-1]$ – оранжевого, а $pref[i-1][j-1]$ – остатка. Иллюстрация на примере:





Аналогично можно посчитать сумму не только следущей клетки в матрице префиксных сумм, но и сумму подматрицы в данном двумерном массиве! Снизу, аналогично, площадь фиолетового прямоугольника – искомая. Пусть его левый нижний угол имеет координаты (lx, ly), а правый верхний (rx, ry):



Тогда площадью фиолотового треугольника является:

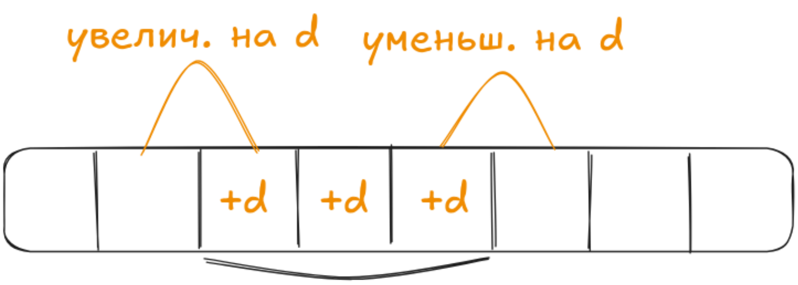
```
ans = prefs[rx][ry]
    - prefs[rx][ly - 1] - prefs[lx - 1][ry]
    + prefs[lx - 1][ly - 1]
```

1.5. Алгоритм, поддерживающий изменения

Пусть у нас есть массив $[a_1, a_2 \dots a_n]$, q запросов вида "прибавить d к каждому числу $[a_l \dots a_r]$". Будем хранить массив разностей!

$$x = [1, 5, 3, 8, 10]; diffs_x = [1, 4, -2, 5, 2]$$

В $diffs_x$ на первую позицию записываем x_1. Далее $diffs_{x_i} = a_i - a_{i-1}$. Тогда x становится массивом префиксных сумм для $diffs_x$. Затем, если мы хотим увеличить какие-либо $l - r + 1$ чисел, то просто добавляем к l-ому элементу $diffs_x$ число d, а из r-ого элемента $diffs_x$ число d вычитаем:



Тогда код выглядит примерно так:

```
std::vector<int> diff(n);
diff[0] = a[0];

// Заполняем массив разностей
```

```
for (int i = 1; i < n; ++i) {
    diff[i] = a[i] - a[i - 1];
}

// Обрабатываем каждый из запросов
for (int i = 0; i < q; ++i) {
    std::cin >> l >> r >> d;
    l--;
    diff[l] += d;
    diff[r] -= d;
}
```

1.6. Трехмерные префиксные суммы

Реализация аналогична двумерному пространству – советую насчет данной преблемы найти информация в интернете. Замечу лишь что поиск подсуммы имеет общее с формулой включений и исключений для множеств.

2. Два указателя

Какого-то общего описания алгоритма нет – подход рассмотрим на задачах, т.е. конкретных проблемах.

- Допустим, у нас есть два массива: $a_1 < a_2 < \cdots < a_n$ и $b_1 < b_2 < \cdots < b_m$. Задача – за $O(n + m)$ найти $(i, j) : a_i = b_j$. Например, если $a = (2, 3, 5)$ а $b = (1, 2, 4, 5)$, то пары это $(i, j) = (0, 1)$ и $(2, 3)$. Для каждого i и j:

  1. Если $a_i = b_j$, то добавляем пару (i, j) в ответ и инкрементируем оба указателя.
  2. Если $a_i \neq b_j$ и $a_i < b_j$, инкрементируем i. Иначе (если $a_i > b_j$), j.

  ```
  std::vector<int> res;
  int i = 0, j = 0;

  while (i < n && j < m) {
      if (a[i] == b[j]) {
          res.push_back(a[i]);
          i++;
          j++;
      } else if (a[i] > b[j]) {
          j++;
      } else {
          i++;
      }
  }
  ```

- Даны два массива: $a_1 \leq a_2 \leq \cdots \leq a_n$ и $b_1 \leq b_2 \leq \cdots \leq b_m$. Задача - слить в один массив по порядку все числа. Просто по очереди перебираем числа из обоих массивов, и добавляем то, что меньше и инкрементируем соответствующий указатель. Затем, как только итерация по одному из массивов закочилась, докладываем оставшиеся в другом массиве элементы. Код выглядит примерно так:

```cpp
int i = 0; j = 0;
std::vector<int> ans;
while (i < n && j < m) {
    if (a[i] <= b[j]) ans.push_back(a[i++]);
    else ans.push_back(b[j++]);
}

while (i < n) ans.push_back(a[i++]);
while (j < m) ans.push_back(b[j++]);
```

- Пусть нам дана прямая чисел $x_1 < x_2 < \cdots < x_n$. $(i, j) : |x_i - x_j| \leq K$, где $i \leq j$ за O(n). Количество пар для i-ой точки – количество чисел на отрезке $[x_i; x_i + k]$, т.е. нам просто нужно перебрать для каждого x_i по несколько x_j, находящихся после x_i и которые меньше чем k, а затем просто перейти к следующему x.

```cpp
int i = 0, j = 0;
int count = 0;

for (i = 0; i < n; i++) {
    while (j < n && x[j] - x[i] <= k) j++;
    count += j - i;
}
```