Модуль 3 Динамическое программирование
Лекция 3.3 Суммы в прямоугольниках

Эта лекция посвящена ещё одной классической задаче, которая часто возникает как подзадача при реализации более сложных алгоритмов.

> **Суммы в прямоугольниках**
> Имеется поле $n \times m$, разбитое на единичные клетки. В клетках записаны числа. Нам поступают запросы. Каждый запрос — это некоторый прямоугольник со сторонами, параллельными осям координат. Нужно для каждого прямоугольника определить сумму чисел в нём (см. пример на рис. 1).

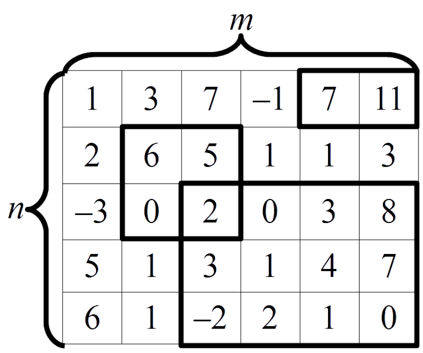

Рис. 1. Пример к задаче «Суммы в прямоугольниках»

Пусть строки нумеруются числами от 1 до n, столбцы — от 1 до m. Каждый запрос задаётся четвёркой чисел (x_1, x_2, y_1, y_2), где x_1 и x_2 — номера строк, $1 \leqslant x_1 \leqslant x_2 \leqslant n$, y_1 и y_2 — номера столбцов, $1 \leqslant y_1 \leqslant y_2 \leqslant m$. Прямоугольнику принадлежат все клетки с координатами (x, y), такие, что $x_1 \leqslant x \leqslant x_2$, $y_1 \leqslant y \leqslant y_2$. Например, запрос на рис. 2 расположен в строках с номерами с 3 до 5 и столбцах с номерами с 3 до 6, поэтому его будет задавать четвёрка чисел $(3, 5, 3, 6)$.

Наивное решение задачи состоит в том, чтобы поместиить числа на поле в двумерный массив и затем для каждого запроса проходить двумя вложенными циклами по соответствующей части массива и суммировать числа. Заметим, что в худшем случае все запросы будут совпадать с максимальным прямоугольником размера $n \times m$, и наивное решение выполнит $O(nmq)$ действий. Это не самое оптимальное по времени решение.

Разберём более быстрое решение. Сначала упростим задачу. Перейдём от двумерного случая к одномерному. Дана полоса, и требуется быстро находить суммы на подотрезках (см. пример на рис. 3). Требуется отвечать на запросы быстрее, чем за $O(n)$.

	1	2	3	4	5	6
1	1	3	7	–1	7	11
2	2	6	5	1	1	3
3	–3	0	2	0	3	8
4	5	1	3	1	4	7
5	6	1	–2	2	1	0

Рис. 2. Пример запроса

a_1	a_2	a_3	a_4	a_5	a_6
1	2	–1	3	1	–2

Рис. 3. Одномерная задача

Для решения задачи можно сделать предподсчёт. Посчитаем частичные суммы на префиксах (начальных подотрезках):

$$s_i = a_1 + a_2 + \cdots + a_i,$$

то есть $s_1 = a_1$, $s_2 = a_1 + a_2$, $s_3 = a_1 + a_2 + a_3$, и т.д. (см. пример на рис. 4). Для их подсчёта можно использовать динамическое программирование, поскольку

$$s_i = s_{i-1} + a_i. \tag{1}$$

Подсчёт начинается с суммы на нулевом префиксе: $s_0 = 0$.

a_1	a_2	a_3	a_4	a_5	a_6
1	2	–1	3	1	–2

s_1	s_2	s_3	s_4	s_5	s_6
1	3	2	5	6	4

Рис. 4. Частичные суммы

При помощи сумм на префиксах можно считать суммы на любых подотрезках. Достаточно заметить, что сумма чисел на отрезке $[l, r]$ равна разности суммы на отрезке $[1, r]$ и суммы на отрезке $[1, l-1]$ (см. рис. 5). Иначе говоря,

$$a_l + \cdots + a_r = s_r - s_{l-1}.$$

Чтобы ответить на запрос, нужно просто найти разность двух предподсчитанных сумм.

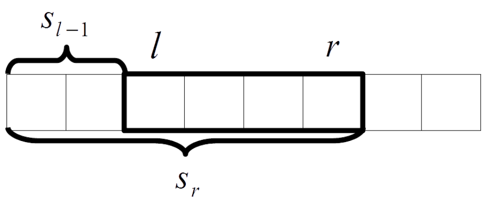

Рис. 5

```
s[0] = 0;
for (int i = 1; i <= n; i++)
{
    cin >> a[i];
    s[i] = s[i - 1] + a[i];
}

cin >> q;
for (int i = 0; i < q; i++)
{
    int L, R;
    cin >> L >> R;
    cout << s[R] - s[L - 1] << endl;
}
```

Рис. 6. Решение одномерной задачи

Описанная идея реализуется в программе следующим образом (см. рис. 6).

Сначала значение s[0] присваивается нулю:

```
s[0] = 0;
```

Затем читаем заданные числа a[i] и сразу же вычисляем частичные суммы по формуле (1):

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

На этом заканчивается предподсчёт и начинаются ответы на запросы. Читаем число запросов q.

```
cin >> q;
```

Затем в цикле читаем сами запросы.

```
for (int i = 0; i < q; i++)
```

В переменной L будет начало отрезка, в R — конец отрезка в 1-индексации.

```
cin >> L >> R;
```

И сразу выводим ответ на запрос:

```
cout << s[R] - s[L - 1] << endl;
```

Проверим крайние случаи, чтобы убедиться, что не происходит выхода за границы. Если запрос начинается с первого элемента $L = 1$, то $L - 1 = 0$. Значит, у нас вычтется значение s[0], которое равно нулю, и будет просто сумма первых R чисел. Если отрезок касается правой границы, то $R = n$. Сумма s[R] в данном случае равна сумме всех чисел до $(n - 1)$-го в 0-индексации. Всё верно.

Упражнение 3.3.1
Оцените время работы описанного решения.

Подведём итоги. Решение на рис. 6 состоит из предподсчёта, который выполняет $O(n)$ действий, и ответов на запросы, каждый из которых требует $O(1)$ времени (это просто вычитание двух сумм). Наивное решение, без предподсчёта, с проходом по массиву и суммированием, тратило бы $O(n)$ времени на каждый запрос. Это распространённая идея: сделать предподсчёт, который позволит тратить меньше времени на каждый запрос.

Вернёмся к двумерному случаю. Посчитаем суммы в прямоугольниках, у которых верхний левый угол совпадает с верхним левым углом всего поля. Пусть s_{ij} — сумма в таком прямоугольнике с правым нижним углом в клетке (i, j) (см. рис. 7). Зная эти суммы, нетрудно найти ответ для произвольного прямоугольника.

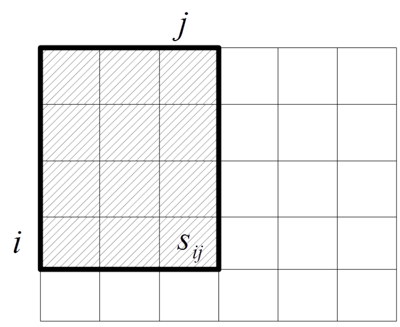

Рис. 7

Действительно, пусть нам нужно посчитать сумму в прямоугольнике, который на рис. 8 обозначен буквой D. Мы можем взять сумму в большом прямоугольнике $(A + B + C + D)$, вычесть из неё суммы в прямоугольниках $(A + B)$ и $(A + C)$. При этом прямоугольник A вычтется два раза, а значит, его нужно прибавить. Получаем следующую формулу:

$$(A + B + C + D) - (A + B) - (A + C) + A = D. \tag{2}$$

В координатах формула (2) примет вид

$$s[x_2][y_2] - s[x_1 - 1][y_2] - s[x_2][y_1 - 1] + s[x_1 - 1][y_1 - 1].$$

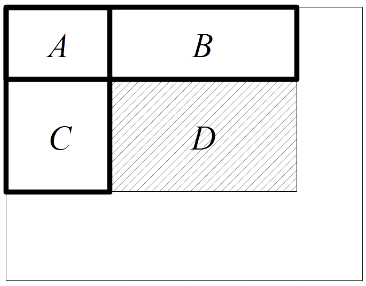

Рис. 8

В результате получится сумма в прямоугольнике, заключенном между x_1 и x_2 по вертикали и между y_1 и y_2 по горизонтали (см. рис. 9).

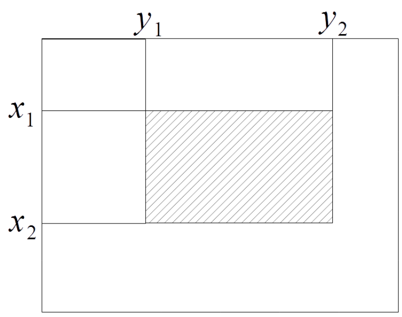

Рис. 9

Аналогичную идею можно использовать при вычислении частичных сумм. Чтобы посчитать сумму в прямоугольнике с правым нижним углом (i, j), нужно сложить два прямоугольника с правыми нижними углами $(i-1, j)$ и $(i, j-1)$, после чего вычесть общую часть с правым нижним углом $(i-1, j-1)$ и прибавить новый элемент a_{ij} (см. рис. 10):

$$s_{ij} = s_{i-1,j} + s_{i,j-1} - s_{i-1,j-1} + a_{ij}. \tag{3}$$

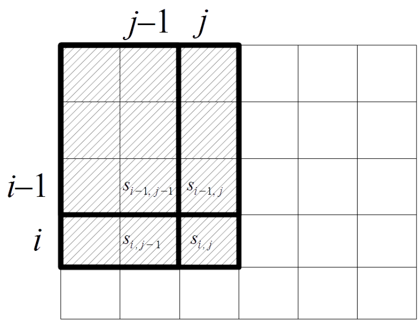

Рис. 10

На основе полученной формулы (3) можно выполнить предподсчёт частичных сумм для всех клеток поля, и потом использовать их, чтобы отвечать на запросы. Код программы по этой задаче остаётся на самостоятельную реализацию.

Упражнение 3.3.2
Оцените время работы описанного решения задачи в двумерном случае, учитывая предподсчёт и ответы на запросы, в зависимости от размеров поля n, m и числа запросов q.