* [Статическая задача RSQ](https://foxford.ru/lessons/32282/conspects/1)
* [Статическая задача RMQ](https://foxford.ru/lessons/32282/conspects/2)

**Статическая задача RSQ**

**Статическая задача RSQ**

В задаче RSQ (Range Sum Query) дан числовой массив a, и поступают запросы "по данным i и j (i<j) найти сумму элементов массива от ai до aj, то есть f(i,j)=ai+ai+1+…+aj.

В статической задаче RSQ массив предполагается неизменным. В динамической задаче RSQ элементы массива могут изменяться между запросами.

Статическая задача RSQ несложна и решается при помощи сумм на префиксах. Заведём вспомогательный массив s, в котором будет хранить сумму элементов массива a на префиксе длины i. То есть si=a0+a1+…+ai−1.

Вычислить значения массива s можно за O(n) (n - длина массива):

**s[0] = 0;**

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

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

Заметим, что s[0] соответствует сумме элементов на пустом префиксе, а s[n] — сумме всех элементов массива a, поэтому массив s должен иметь длину n+1.

Тогда отвечать за запросы можно по формуле: f(i,j)=sj+1−si. Реализуем эту функцию:

**int f(int i, int j)**

**{**

**return s[j + 1] - s[i];**

**}**

Тем самым статическая задача RSQ легко решается используя предподсчет сложности O(n) и со сложностью обработки запроса O(1).

**Корневая оптимизация**

Интересней задача динамического RSQ, в котором помимо запросов на вычисление суммы на подотрезке могут быть запросы изменения элементов массива. Например, пусть запросы имеют вид - нужно изменить в массиве один элемент. После изменения элемента будет меняться ответ на все запросы суммы на подотрезке, включающие этот элемент.

Если запросов на изменение будет мало, то можно можно просто в отдельном массиве хранить список всех произведённых изменений в виде "элемент номер i нужно изменить на величину di". При обработке запроса посчитаем ответ для нужного подотрезка, а потом просмотрим все ранее сделанные изменяющие запросы, проверив, какие из запросов модифицировали элементы на данном отрезке. Учтём эти изменения в ответе.

Сложность такого алгоритма будет O(N+MK), где N - количество чисел, M - количество запросов на сумму, K - количество изменяющих запросов.

При большом K можно существенно улучшить решение, сделав его сложность меньше, чем O(NM). Аналогичным образом будем отдельно хранить изменяющие запросы и учитывать их в ответе. Пусть d=N. Если количество изменяющих запросов достигло d, то применим все эти запросы непосредственно к исходному массиву и снова пересчитаем массивы частичных сумм. То есть изменяющие запросы будут "накапливаться" в буфере, и когда их значение достигнет критического числа d все запросы будут одновременно применены к массиву, для чего нужно будет заново пересчитать частичные суммы. Эта операция будет выполняться долго, но после этого все полученные изменяющие запросы "исчезают", то есть массив накопленных изменений уменьшается до нулевого размера.

Значение d=N выбрано не случайно. Если взять значение d большим, то слишком долго будет производиться ответ на запрос о сумме на подотрезке, так как число изменяющих запросов будет большим. Если взять d маленьким, то пересчет частичных сумм будет происходить часто, а он выполняется за O(N).

Пусть общее число запросов (как изменяющих, так и запросов на сумму) равно M. Тогда сложность ответов на эти запросы без учета пересчета частичных сумм будет равно O((dM)=O(NM). Изменяющих запросов, которые приводят к пересчету частичных сумм будет не более O(M/d)=O(M/N), а общая сложность этих запросов есть O(NM/N)=O(NM).

Таким образом, суммарная сложность обработки всех запросов будет O(NM), то есть в среднем один запрос будет обрабатываться за O(N), а не за O(N), как при наивной реализации.

Такая технология называется *корневой оптимизацией*, также используются термины "корневая эвристика", "корневая декомпозия", "sqrt-декомпозиция".

Рассмотренные на последующих занятиях алгоритмы построения дерева отрезков помогут решить эту задачу за сложность O(log⁡N) на запрос.

#### Статическая задача RMQ

## Статическая задача RMQ

Рассмотрим статическую задачу RMQ (range max query) - дан числовой массив и запросы нахождения наибольшего значения на отрезке [i,j].

Можно решить статическую задачу RMQ используя метод корневой декомпозиции. Пусть k=⌈N⌉ (целая часть N, округленная вверх). Разобьем массив из N элементов на k отрезков длины k (последний отрезок может быть короче k). Для каждого из отрезков посчитаем наибольшее значение на этом подотрезке. Назовем эти отрезки "крупными".

При получении запроса на сумму элементов, этот запрос покрывает какое-то число "крупных" отрезков, а также конец одного "крупного" отрезка и начало другого. Возможен и частный случай - например, когда запрос целиком попадает внутрь крупного отрезка, тогда он покрывает часть одного "крупного" отрезка, но ни один "крупный" отрезок не покрывает целиком.

В любом случае для решения задачи нужно разбить искомый отрезок на несколько "крупных" и не более двух частей, находящихся внутри "крупных". В качестве ответа нужно взять максимальное значение из максимума на всех частях. При этом ответ для каждого "крупного" отрезка уже подсчитан, а каждая из частей "крупного" отрезка обрабатывается проходом по всей части. Количество "крупных" отрезков не превосходит k, количество элементов в каждой части, не совпадающей с "крупным" отрезком также не превосходит k, поэтому обработка одного запроса будет иметь сложность O(k)=O(N).

Можно реализовать идею корневой декомпозиции для запросов, изменяющих значение отдельных элементов (при условии, что значение каждого элемента может только увеличиваться) аналогично решению задачи RSQ.

## Sparse table

Идею разбиения на отрезки для задачи RMQ можно улучшить, если заметить, что отрезки в разбиении могут пересекаться. То есть если A - отрезок и A=B∪C, где B и C - меньшие отрезки, тогда max(A)=max(max(B),max(C)).

В алгоритме RMQ с использованием Sparse table рассматриваются отрезки, длины которых равны степеням двойки: 1, 2, 4, … А именно, cначала нужно посчитать сумму на всех подотрезках данного массива, длины которых являются степенями двойки. Пусть:

si,j=max(ai,ai+1,…,ai+2j−1)

то есть si,j есть максимум на подотрезке, первым элементом которого является ai, а длина которого равна 2j. Таких отрезков не более, чем N⌈log⁡N⌉.

Для хранения значений si,j заведём двумерный массив размера N×(L+1), где L=⌊log⁡N⌋. Элементы массива заполняются по формулам:

si,0=ai

si,j=max(si,j−1,si+2j−1,j−1)

Каждый отрезок длины 2j разбивается на два отрезка длины 2j−1. Таким образом, для хранения этого массива нужно O(Nlog⁡N) памяти и также O(Nlog⁡N) времени для заполнения.

После этого для вычисления максимума на отрезке [i,j] нужно выполнить следующее:

1. Найти значение k=⌊log⁡(j−i+1)⌋.

2. Отрезок [i,j] разбивается на два отрезка длины k - с началом в ai и с началом в aj−2k+1.

3. Ответом будет max(si,k,sj−2k+1,k).

Таким образом, задача RMQ будет решаться при помощи предподсчета сложности ONlog⁡N) и с обработкой одного запроса за O(1).

Мы не учитывали в оценке O(1) сложность вычисления логарифма длины отрезка. Для того, чтобы вычислять логарифм за O(1) нужно предподсчитать заранее логарифмы всех чисел от 1 до N и сохранить их в массиве.

Sparse table не позволяет обрабатывать запросы, модифицирующие значения массива.