

Запросы на отрезках

Шовкопляс Григорий

Введение в алгоритмы и структуры данных



Что такое запросы на отрезках?

Запросы на отрезках

- Есть массив чисел, к нему можно последовательно применять запросы:
 - Сумма на отрезке с L по R: $\sum_{i=L}^{R} a[i]$
 - Минимум/максимум на отрезке с L по R: $min/max_{i=L}^R a[i]$
 - Изменить і-й элемент массива: a[i] = x
 - Прибавить x на отрезке с L по R: for i in [L; R]: a[i] += x
 - И другие...

Префиксные суммы

Префиксные суммы

- Сможем за O(1) отвечать на запрос RSQ(l, r)
- Пусть дан массив a[i]
- Посчитаем массив $sum[i] = \sum_{j=0}^{i} a[i]$
 - Как считать?
 - sum[i] = sum[i-1] + a[i]
- Чему равно *RSQ(I, r)*?
 - sum[r] sum[l-1]
 - l = 0?
 - sum[r]!

Префиксные суммы

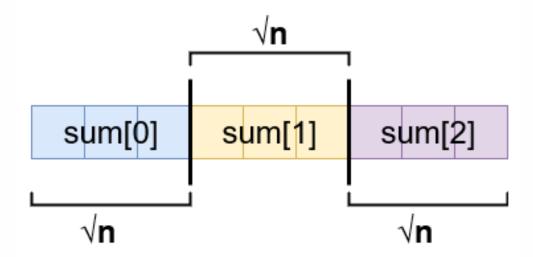
Как писать?

```
rsq(l, r)
  if 1 = 0
    return sum[r]
  return sum[r] - sum[l - 1]
```

Корневая эвристика (Sqrt decomposition)

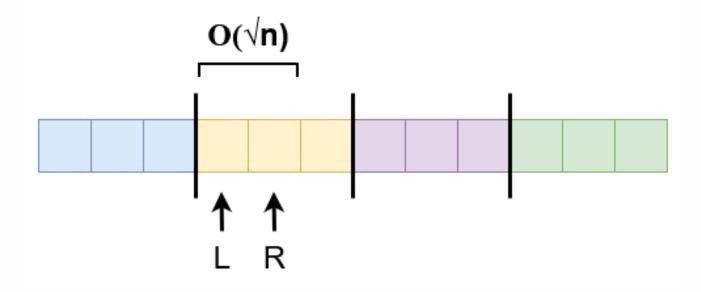
Корневая эвристика

- Есть ассоциативная операция (сумма/максимум/минимум)
 - a + (b + c) = (a + b) + c
- Отвечаем на запрос на отрезке за $O(\sqrt{n})$



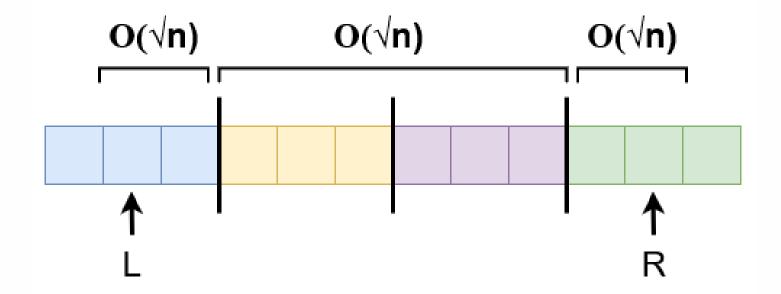
Корневая эвристика

- Отвечаем на запрос: L и R внутри одного блока



Корневая эвристика

• Отвечаем на запрос: L и R в разных блоках



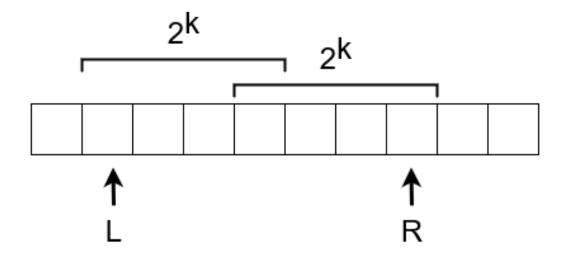
Разреженная таблица (Sparse Table)

- Хотим отвечать на запросы RMQ за O(1)
- Префиксные суммы не работают для минимума/максимума
 - Минимум не обратим
- Наивная идея:
 - Для всех I, r храним: min[I][r] $min_{i=1}^r a[i]$
 - *O*(1) запрос
 - $O(n^3)$ предподсчет

- Улучшим предподсчет $O(n^3) \to O(n^2)$
- Динамическое программирование:
 - min[i][i] = a[i]
 - min[l][r] = min(min[l][r-1], a[r])
- Идемпотентность: min(a, a) = a

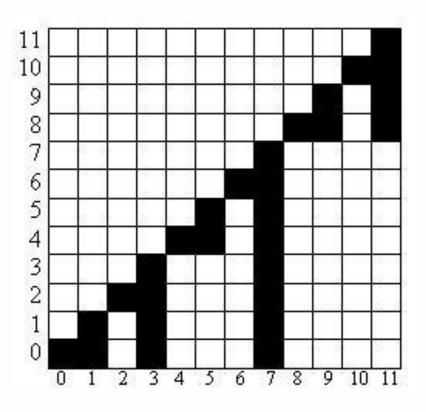
- Улучшим предподсчет и память $O(n^2) \rightarrow O(nlogn)$
- min[l][k] $min_{i=l}^{l+2^k-1} a[i]$: минимум на отрезке $[l; l+2^k)$
 - Размер таблицы O(nlogn)
- Как считать?
- Динамическое программирование:
 - min[i][0] = a[i]
 - $min[l][k] = min(min[l][k-1], min[l+2^{k-1}][k-1])$
 - Предподсчет O(nlogn)

- Как отвечать на запрос за *O(1)*?
- $k : \max\{i \mid 2^i \le r l + 1\}$
- rmq(l, r) = min(min[i][k], min[?][k])
- $rmq(l, r) = min(min[i][k], min[r-2^k+1][k])$



- Есть три вида запросов:
 - rsq(l, r)
 - set(i, x)
 - add(i, x)

- Обобщим префиксные суммы:
 - $T[i] = \sum_{j=F(i)}^{i} a[j]$
- В дереве Фенвика F(i) = i & (i+1)
- На картинке заштрихованы элементы массива, которые входят в сумму Т[i]



- Запрос rsq(l, r)
- Научимся считать префиксные суммы
 - Научимся считать сумму на отрезке
- sum[i] = T[i] + sum[F(i) 1]
- Утверждение: sum[i] раскладывается в O(log i) элементов Т[j]

Запросы суммы

```
get(i)
  res = 0
  while i >= 0
   res += T[i]
    i = F(i) - 1
  return res
rsq(l, r)
  if 1 = 0
   return get(r)
  return get(r) - get(l - 1)
```

- Запрос add(i, x)
- Утверждение: a[i] входит в O(log n) элементов T[j]
- $F(j) \le i \le j$
- ј можно вычислить по рекуррентной формуле:
 - $j_0 = i$
 - $j_k = j_{k-1} | (j_{k-1} + 1)$

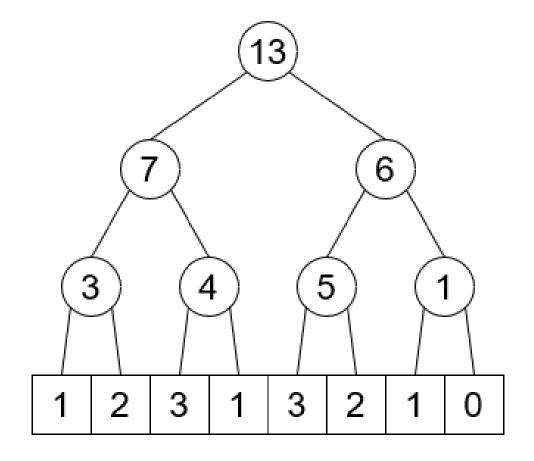
Запросы add(i, x)

```
add(i, x)
  while j < n
    T[j] += x
    j = j | (j + 1)
```

Запросы set(i, x)

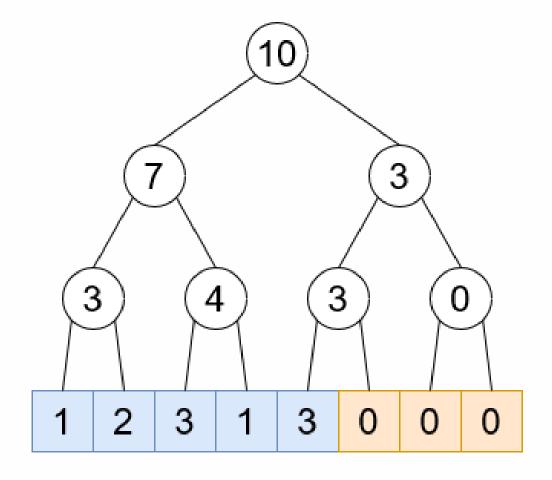
```
set(i, x)
 d = x - a[i]
 a[i] = x
  add(i, d)
```

- Полное двоичное дерево
- Листы элементы массива
- T[v] = T[2v + 1] + T[2v + 2]
- Работает для любой ассоциативной операции



- Если длина массива не степень двойки?
- Нейтральный элемент

- $X = \min\{ 2^k \mid 2^k \ge n \}$
- t[X-1+i] = a[i]

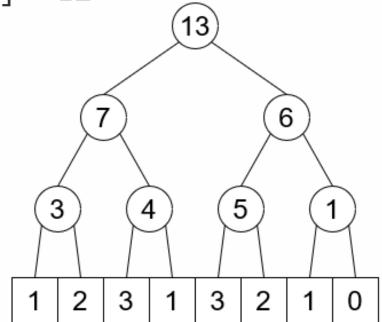


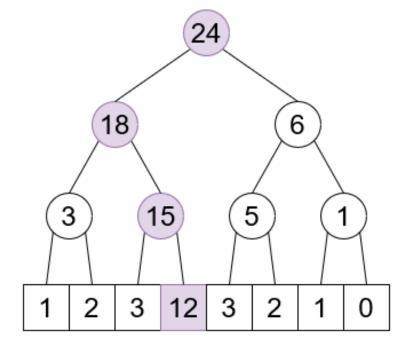
Построение

```
buildTree(a, n)
  x = 1
  while x < n
   x *= 2
  for i = 0 to n - 1
   t[i + x - 1] = a[i]
  for v = x - 2 to 0
   t[v] = t[2 * v + 1] + t[2 * v + 2]
```

• Обновление в точке

• a[3] = 12





Обновление в точке

```
update(i, k)

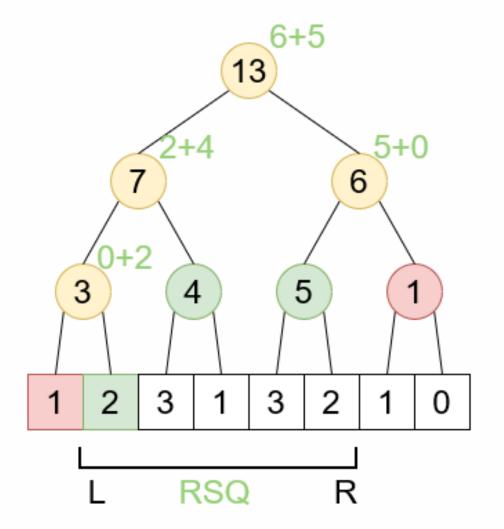
t[i + x - 1] = k

while v - \text{ не корень } // v \neq 0

v = (v - 1) / 2

t[v] = t[2 * v + 1] + t[2 * v + 2]
```

- rsq(a, b)
- Пусть [l, r] отрезок, за который отвечает вершина v
- Три типа вершин
 - \blacksquare [l, r] \in [a, b]
 - $\bullet [l,r] \cap [a,b] = \emptyset$
 - $[l, r] \cap [a, b] \neq \emptyset$



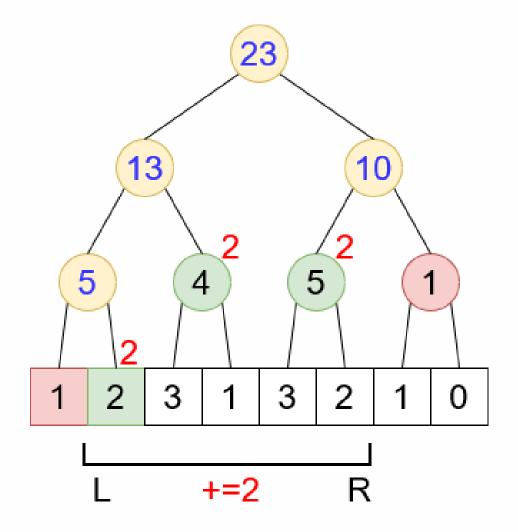
RSQ(a, b)

```
rsq(v, l, r, a, b)
  if l > b or r < a
   return 0
  if l >= a and r <= b
   return t[v]
 m = (1 + r) / 2
  return rsq(2 * v + 1, 1, m, a, b)
       + rsq(2 * v + 2, m+1, r, a, b)
```

RMQ(a, b)

```
rmq(v, 1, r, a, b)
 if l > b or r < a
   return INF/-INF
  if l >= a and r <= b
   return t[v]
 m = (l + r) / 2
 return min(rmq(2 * v + 1, l, m, a, b),
            rmq(2 * v + 2, m+1, r, a, b))
```

- Групповое обновление
- Пусть [l, r] отрезок, за который отвечает вершина v
- Три типа вершин
 - \blacksquare [l, r] \in [a, b]
 - $\bullet [l,r] \cap [a,b] = \emptyset$
 - $[l, r] \cap [a, b] \neq \emptyset$
- Добавим пометку в каждую вершину v – upd[v]



Групповое обновление: вспомогательные функции

```
get(v, l, r) //для add и rsq
  return t[v] + (r - l + 1) * upd[v]
get(v, l, r) //для set и rsq
  if upd[v] = null
    return t[v]
  return (r - l + 1) * upd[v]
get(v, l, r) //для add и rmq
  return t[v] + upd[v]
```

Групповое обновление: вспомогательные функции

```
push(v, l, r)
  if v - \pi v  //l == r
   t[v] += upd[v]
  else
   upd[2 * v + 1] += upd[v]
    upd[2 * v + 2] += upd[v]
   m = (1 + r) / 2
    t[v] = get(2 * v + 1,    1, m)
         + get(2 * v + 2, m + 1, r)
 upd[v] = 0
```

Групповое обновление

```
update(v, l, r, a, b, x)
 push(v, l, r)
  if l > b or r < a
   return
  if l >= a and r <= b
   upd[v] += x
   return
  m = (1 + r) / 2
 update (2 * v + 1, 1, m, a, b, x)
 update(2 * v + 2, m+1, r, a, b, x)
```

RSQ(a, b) при групповых обновлениях

```
rsq(v, 1, r, a, b)
 push(v, 1, r)
  if l > b or r < a
   return 0
  if l >= a and r <= b
   return get(v)
  m = (1 + r) / 2
  return rsq(2 * v + 1, 1, m, a, b)
       + rsq(2 * v + 2, m+1, r, a, b)
```

Bce!