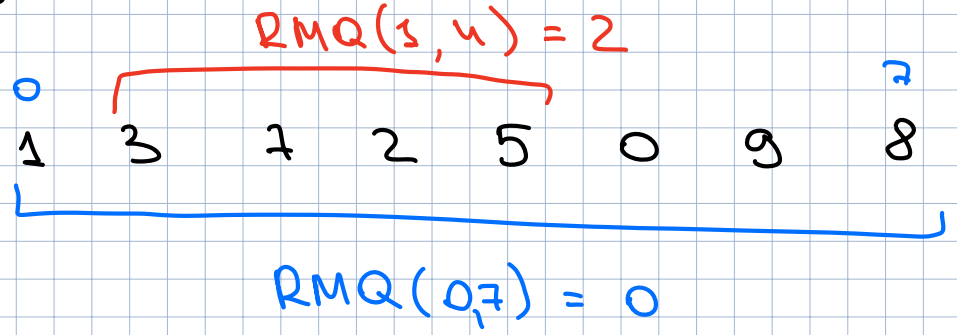
RMQ

range minimum (maximum) query

Запрос минимума (максимума) на подотрезке.

I Пример



Тривиальная RMQ → O(N)

II Виды RMQ

static

эл-ты массива не меняются

dynamic

эл-ты массива меняются

offline

online

изначально
знаем
запрос

запрос поступают
online

Нас интересуют: static online RMQ

(III) Как это происходит?

Массив
↓
препроцессинг → можно не очень читаться за оптимальное.
↓
запрос → ответ
↓
запрос → ответ
...

$N \to \infty$

(IV) Тривиальный RMQ

A = [....] - данное

Опр $RMQ(i,j) = \min(A[i], A[i+1], \dots A[j])$

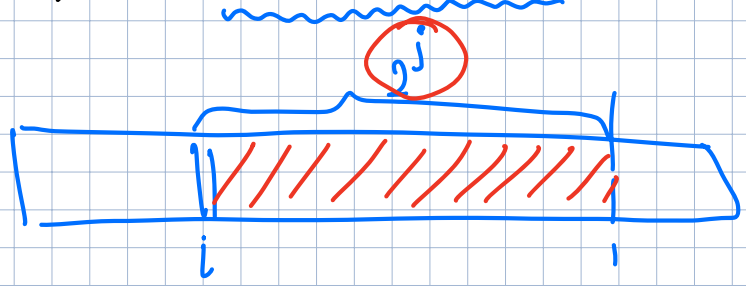
BAD [i] [j] = RMQ(i,j) → $O(N^2)$ - $T(N)$ = $M(N)$

→ RMQ(3, 7) → print(BAD[5][7]) → $O(1)$

(V) Sparse Table (разреженная таблица)

$ST[i][j] = \min(A[i], A[i+1], \dots A[i+2^j-1])$

- i — индекс начала отрезка
- j — логарифм длины отрезка



$RMQ(i, i+2^j-1)$

Хотим: $RMQ(s, f) \neq ST[s][f]$

- s — индекс
- f — длина

$ST[i][0] = RMQ(i, i) = A[i]$

$ST[i][1] = RMQ(i, i+1) = \min(A[i], A[i+1])$

i: $0 \dots N-1$ (N)

j: $\log N$

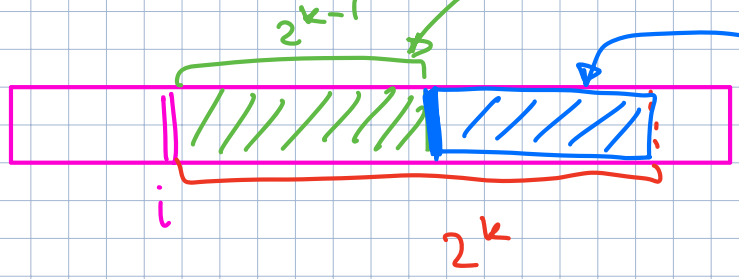
$M(N) = O(N \log N)$

$T(N) = O(N \log N)$

} препроцессинг.

(VI) Заполнение ST.

$$ST[i][k] = \begin{cases} A[i], & k = 0 \\ \min\left(ST[i][k-1],\ ST[i+2^{k-1}][k-1]\right), & k > 0 \end{cases}$$



(VII) Вычисление RMQ

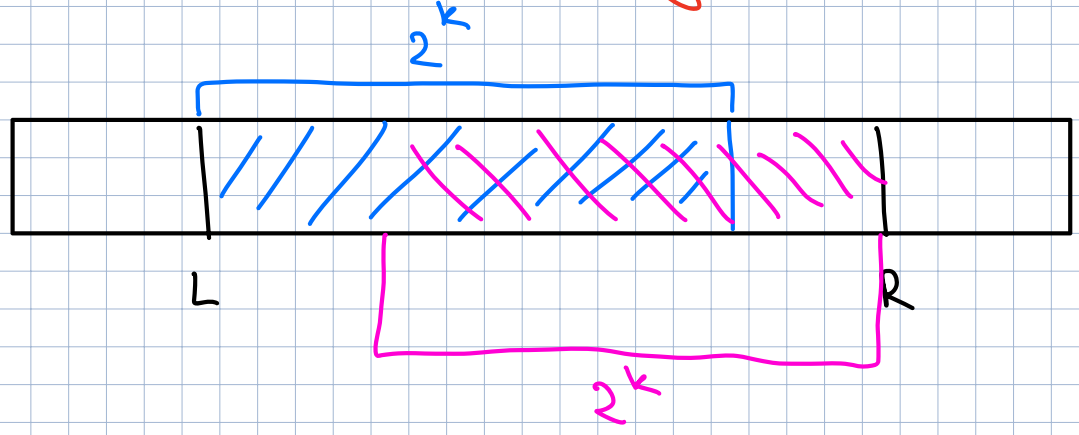
$$RMQ(L,R) = \min(A[L], A[L+1], \dots A[R]) =$$

$$= \min\left(ST[L][k],\ ST[R-2^k+1][k]\right)$$

$$k = \max\{k \mid 2^k \le R-L+1\}$$

$$k = \lfloor \log(R-L+1) \rfloor$$

$T(N) = O(1)$



VIII Асимптотики: препроцессинг за $\underline{O}(N \log N)$

времени

запрос за $\underline{O}(1)$

$M(N) = \underline{O}(N \log N)$

IX

```
f log (int N):
    if N == 1:
        return 0
    else:
        return f log (⌊N/2⌋) + 1
```

$\underline{O}(\log N)$

X Пример

$A = [5, 7, 9, 1, 3]$

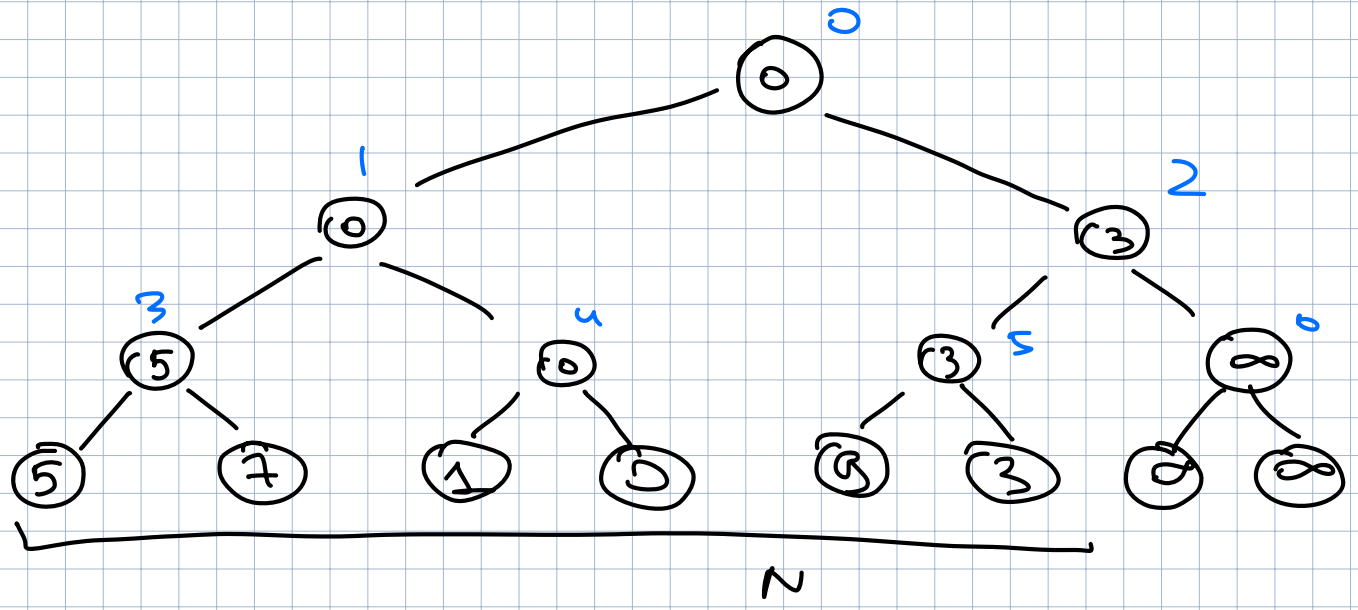
$ST = \begin{bmatrix} [5,5,1] \\ [7,7,1] \\ [9,1,1] \\ [1,1,17] \\ [3,3,1] \end{bmatrix}$

$St[i][0] = A[i] - \min(A[i] .. A[i+2^0-1])$

V++

RMQ (решение с деревом отрезков)

1) Дерево отрезков (хранится в виде пирамиды)
2) Длина - степень двойки
   ↳ добиваем до степени двойки $+\infty$
3) Листья в дереве - элементы массива (изначально)
4) Не листья - различного вида RMQ



VI++ N элементов

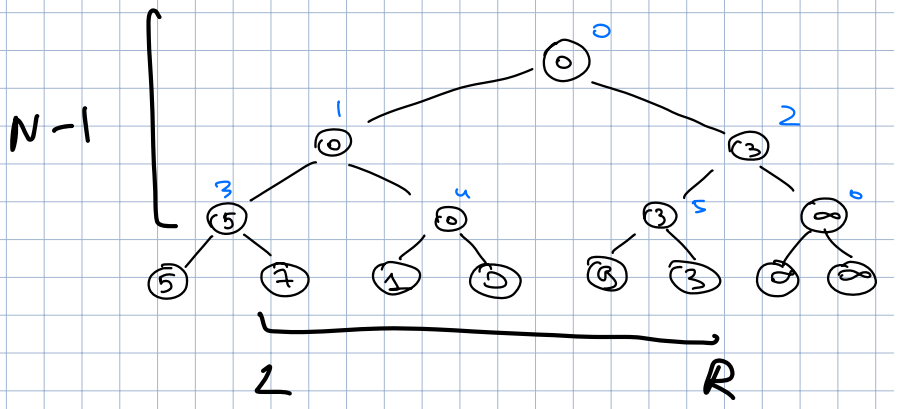
T[0] - корень T[V] - вершина

нечетный индекс → левый ребенок: T[2V+1]

четный индекс → правый ребенок: T[2V+2]

Элементов в дереве отрезков: $2N-1$

$M(N) = O(2N-1) = O(N)$



VII++ Ответ на запрос

1) RMQ (L, R)

L = L + N
R = R + N
} индексы эл-в в дереве отрезков.

2) ans = $+\infty$

3) while (L <= R)
{
if (L - правый сын)
↳ ans = min (ans, T[L])

if (R - левый сын)

↳ ans = min(ans, T[R])

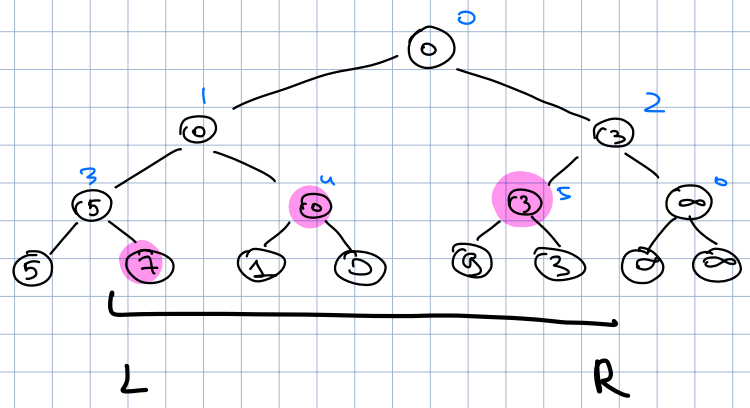
L = L/2

R = R/2

более не logN

}

return ans



VIII++ Асимптотика $T(N) = O(\log N)$ – запроса

$\begin{cases} M(N) = O(N) \\ T(N) = O(N) - \text{препроцессинг} \end{cases}$

Алгоритм препроцессинга

for i = N-1 → 0

T[i] = min(T[2i+1], T[2i+2])

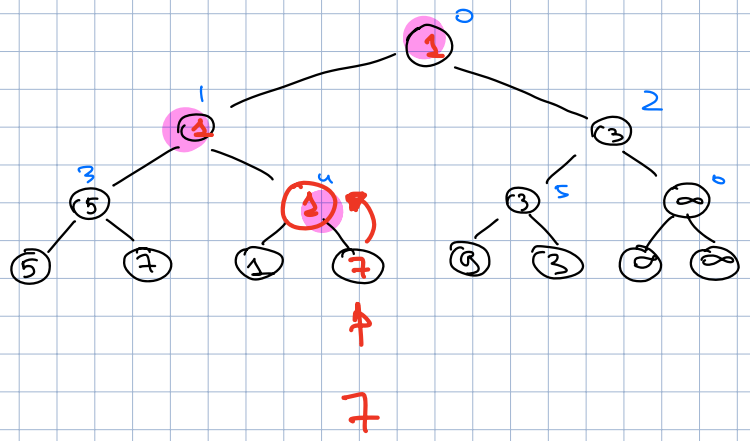
не забыть добить до степени двойки ∞

IX++ Алгоритм online Dynamic RMQ

change (i , value)

за

$\underline{\Omega}(\log N)$



LCA

Least common ancestor

Наименьший общий предок

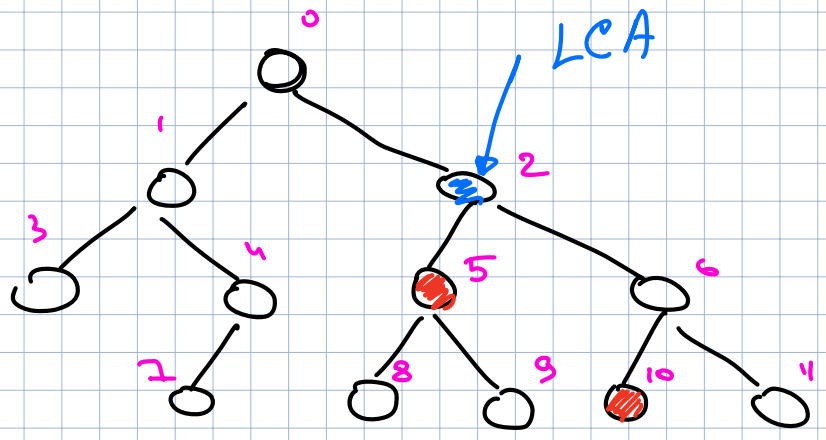
Деревья

BST

$L < V < R$

BT

не дерево поиска



1) Не знаем родителей.

$O(n)$

① Найти родителя (BFS)

DFS → найти родителя

↓

найти высоту

parent[i] → Node* родителя

index[i] → Node* себя

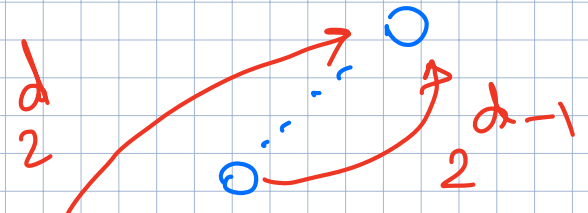
depth[i] → глубина i-го элемента

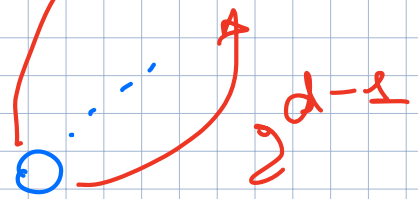
② Если не знаем о существовании RMQ.

для вершины v хотим предпосчитать ее предков → up[v][d], где лежит предок вершины v на высоте 2^d

Идея:

Предок вершины v на расстоянии 2^d это предок вершины на расстоянии 2^{d-1} с расстоянием 2^{d-1}





II.1 Заполнение таблицы (препроцессинг)

$$up[v][d] = \begin{cases} parent[v], & d = 0 \\ up[up[v][d-1]][d-1] \end{cases}$$

$up[v][d-1]$ — предок вершины v высоты $d-1$

dfs(0) → parent, index, depth

```
for i = 1 ... n:
    up[i][0] = parent[i]

for i = 1 ... logN:
    for j = 1 ... N:
        up[i][j] = up[up[i-1][j-1]][j-1]
```

II.2 LCA(u, v)

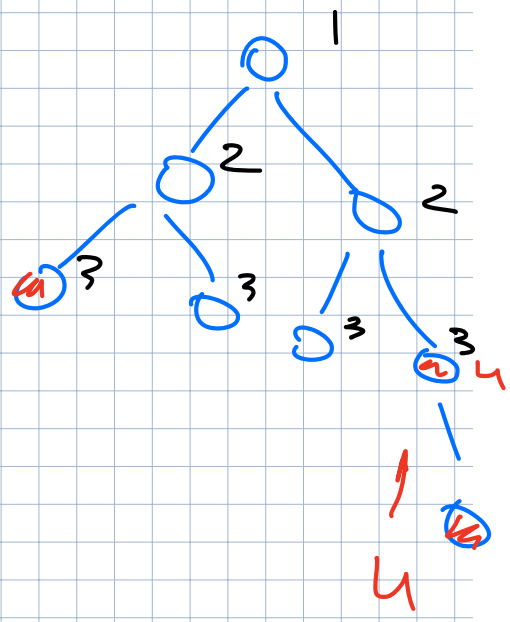
```
if depht(v) > depht(u):
    swap(u, v)
```

→ u теперь

```
for i = logn ... 0:
    if depht[up[u][i]] >= depht[v]:
        u = up[u][i]
```

на одном уровне

```
if u == v:
    return u
```




```
for i = logN ... 0:
    if up[v][i] != up[u][i]:
        v = up[v][i]
        u = up[u][i]
return parent[v]
```

идем до момента до нахождения

③ Знаем RMQ

DFS → depht(u) = { 0, u = root; depht(v)+1, v.parent = u

index

parent

depht

RMQ на depht — id вершины

lca(u,v) = rmq(u, v))

возвращает индекс мин. по высоте под. под. дер.

LCA сводится к RMQ