Разреженная таблица и LCA

Denis Bakin

Постановка задачи

Дан массив a из n целых чисел. Поступают запросы:

- **Минимум на отрезке**: $\min(a_l, a_{l+1}, \dots, a_{r-1})$

Массив **не меняется** между запросами — это **static RMQ** Хотим запрос за $O(1)$ ценой какого-то препроцессинга

Какие у нас варианты?

Подход	Препроцессинг	Запрос
Массив «в лоб»	$O(1)$	$O(n)$
Дерево отрезков	$O(n)$	$O(\log n)$
Разреженная таблица	$O(n \log n)$	$O(1)$

Какие у нас варианты?

Подход	Препроцессинг	Запрос
Массив «в лоб»	$O(1)$	$O(n)$
Дерево отрезков	$O(n)$	$O(\log n)$
Разреженная таблица	$O(n \log n)$	$O(1)$

Раз массив статический — можно потратить больше памяти на препроцессинг и получить запрос за константу

Идея: минимумы по степеням двойки

Заведём двумерный массив:

$$\mathsf{mn}[k][i] = \min(a_i, a_{i+1}, \dots, a_{i+2^k-1})$$

В ячейке `mn[k][i]` хранится минимум на отрезке длины 2^k, начинающемся с i

Построение

Базовый случай — отрезок длины 1:

$$\mathsf{mn}[0][i] = a_i$$

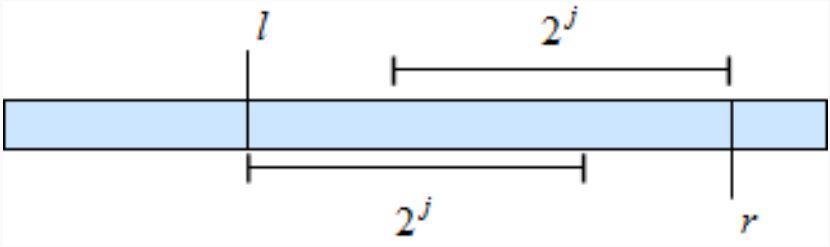
Отрезок длины 2^k — это два отрезка длины 2^{k-1}:

$$\mathsf{mn}[k][i] = \min(\mathsf{mn}[k-1][i],\ \mathsf{mn}[k-1][i+2^{k-1}])$$

Каждая ячейка считается за $O(1) \Rightarrow$ построение за $O(n \log n)$

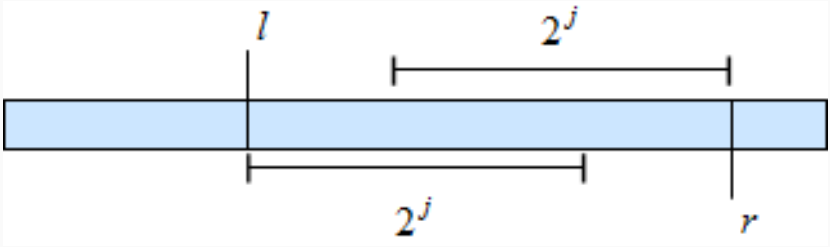
Запрос: ключевое наблюдение

Любой полуинтервал $[l, r)$ можно покрыть **двумя** отрезками длины степени двойки



Запрос: ключевое наблюдение

Любой полуинтервал $[l, r)$ можно покрыть **двумя** отрезками длины степени двойки



Берём $k = \lfloor \log_2(r - l) \rfloor$ и два отрезка длины 2^k — упирающихся в l и в r

Запрос: формула

$$\mathsf{rmq}(l, r) = \min(\mathsf{mn}[k][l],\ \mathsf{mn}[k][r - 2^k])$$

Отрезки **пересекаются**, но для минимума это не страшно: $\min(x, x) = x$

Реализация: построение

```
constexpr int LOGN = 20;  // достаточно для n <= 10^6
std::vector<std::vector<int>> mn;

void Build(const std::vector<int>& a) {
    int n = a.size();
    mn.assign(LOGN, std::vector<int>(n));
    mn[0] = a;
    for (int k = 1; k < LOGN; ++k) {
        for (int i = 0; i + (1 << k) <= n; ++i) {
            mn[k][i] = std::min(mn[k - 1][i],
                                mn[k - 1][i + (1 << (k - 1))]);
        }
    }
}
```

Реализация: запрос

```
int Rmq(int l, int r) {  // [l, r)
    int k = __lg(r - l);
    return std::min(mn[k][l], mn[k][r - (1 << k)]);
}
```

Какие операции подходят?

От операции $\circ$ требуются:

- **ассоциативность**: $(a \circ b) \circ c = a \circ (b \circ c)$
- **коммутативность**: $a \circ b = b \circ a$
- **идемпотентность**: $a \circ a = a$

Идемпотентность нужна, чтобы пересечение двух покрывающих отрезков «не мешало»

Какие операции подходят?

От операции $\circ$ требуются:

- **ассоциативность**: $(a \circ b) \circ c = a \circ (b \circ c)$
- **коммутативность**: $a \circ b = b \circ a$
- **идемпотентность**: $a \circ a = a$

Идемпотентность нужна, чтобы пересечение двух покрывающих отрезков «не мешало»

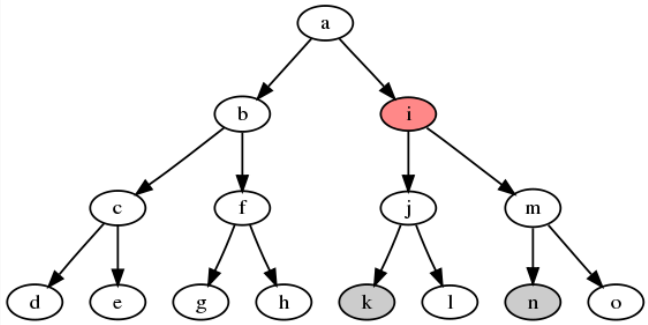
Подходят: $\min$, $\max$, $\gcd$, побитовые `AND`, `OR`

Не подходят: сумма, произведение

Задача LCA

Дано **корневое дерево** из n вершин

LCA (*least common ancestor*) — для двух вершин u, v найти самую глубокую вершину, лежащую на пути от корня одновременно к u и к v



Простой алгоритм за $O(n)$

После DFS у каждой вершины известны времена входа `tin` и выхода `tout`

Простой алгоритм за $O(n)$

После DFS у каждой вершины известны времена входа `tin` и выхода `tout`

`u` — предок `v` $\iff [\text{tin}_u, \text{tout}_u] \supseteq [\text{tin}_v, \text{tout}_v]$

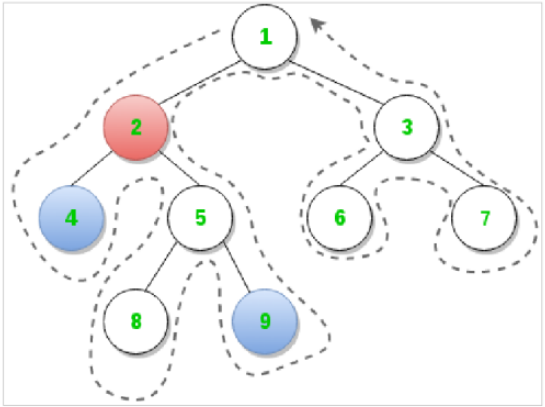
```
bool IsAncestor(int u, int v) {
    return tin[u] <= tin[v] && tout[v] <= tout[u];
}

int Lca(int u, int v) {
    while (!IsAncestor(u, v)) {
        u = parent[u];
    }
    return u;
}
```

Время — $O(h)$ на запрос. Худший случай — $O(n)$

Идея: эйлеров обход

- Запустим DFS и будем выписывать вершину **каждый раз, когда мы её посещаем** — и при заходе, и при возврате из ребёнка
- Получится последовательность длины $2n - 1$ — **эйлеров обход** дерева
- Параллельно запоминаем массив **глубин** для каждой позиции и `first[v]` — позицию первого вхождения каждой вершины

Эйлеров обход



Euler Tour

An euler tour of the tree starting from node 1 will yield:

1	2	4	2	5	8	5	9	5	2	1	3	6	3	7	3	1

The corresponding levels for every node in Euler tour:

0	1	2	1	2	3	2	3	2	1	0	1	2	1	2	1	0

LCA через RMQ

Пусть нужно найти LCA вершин u и v. Без ограничения общности $\text{first}[u] \leq \text{first}[v]$

- На отрезке $[\text{first}[u], \text{first}[v]]$ массива глубин LCA — это вершина с минимальной глубиной

LCA через RMQ

Пусть нужно найти LCA вершин u и v. Без ограничения общности $\text{first}[u] \leq \text{first}[v]$

- На отрезке $[\text{first}[u], \text{first}[v]]$ массива глубин LCA — это вершина с минимальной глубиной

Почему? Двигаясь по эйлерову обходу от u до v, мы проходим по единственному пути между ними: поднимаемся до LCA и спускаемся к v. Выше LCA подняться нельзя

Алгоритм запроса

1. Взять отрезок $[\text{first}[u], \text{first}[v]]$ массива глубин
2. Найти на нём **позицию минимума**
3. По этой позиции в эйлеровом обходе — искомый LCA

Алгоритм запроса

1. Взять отрезок $[\text{first}[u], \text{first}[v]]$ массива глубин
2. Найти на нём **позицию минимума**
3. По этой позиции в эйлеровом обходе — искомый LCA

Применим разреженную таблицу к массиву глубин $\Rightarrow$ запрос LCA за $O(1)$

Итоги

- **Static RMQ** — запросы минимума на статическом массиве
- **Разреженная таблица**: $O(n \log n)$ препроцессинг, $O(1)$ запрос
- Работает для **идемпотентных** операций ($\min$, $\max$, $\gcd$, ...)
- **LCA** на дереве сводится к Static RMQ через **эйлеров обход**
- Получаем $O(n \log n)$ препроцессинг + $O(1)$ на каждый LCA-запрос