Алгоритмы и структуры данных

RMQ и LCA

CS Center, Новосибирск

Range Min Query

Последовательность чисел $a_1, a_2, \ldots a_n$

Требуется обработать запросы вида «$\min\{a_i\}$ – ?» $i \in [l, r]$

Range Min Query

Последовательность чисел $a_1, a_2, \ldots a_n$

Требуется обработать запросы вида «$\min\{a_i\}$ – ?» $i \in [l, r]$

- Статическая постановка задачи:
  - только запросы на минимум

Range Min Query

Последовательность чисел $a_1, a_2, \ldots a_n$

Требуется обработать запросы вида «$\min\{a_i\}$ – ?» $i \in [l, r]$

- Статическая постановка задачи:
  - только запросы на минимум
- Динамическая постановка задачи:
  - кроме запросов на минимум запросы на изменение «$a_i := x$»

Range Min Query

Последовательность чисел $a_1, a_2, \ldots a_n$

Требуется обработать запросы вида «$\min\{a_i\}$ – ?» $i \in [l, r]$

- Статическая постановка задачи:
  - только запросы на минимум
- Динамическая постановка задачи:
  - кроме запросов на минимум запросы на изменение «$a_i := x$»
- Offline:
  - можно получить все запросы, а потом ответить на них

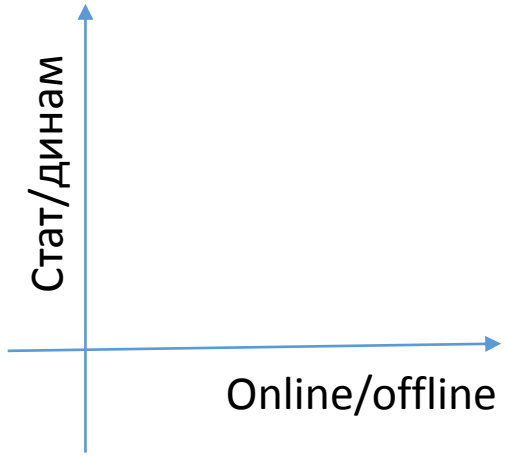
Range Min Query

Последовательность чисел $a_1, a_2, \ldots a_n$

Требуется обработать запросы вида «$\min\{a_i\}$ – ?» $i \in [l, r]$

- Статическая постановка задачи:
  - только запросы на минимум
- Динамическая постановка задачи:
  - кроме запросов на минимум запросы на изменение «$a_i := x$»
- Offline:
  - можно получить все запросы, а потом ответить на них
- Online:
  - следующий запрос получаем после того, как даём ответ на текущий

Range Min Query



Последовательность чисел $a_1, a_2, \ldots a_n$

Требуется обработать запросы вида «$\min\{a_i\}$ – ?» $i \in [l, r]$

- Статическая постановка задачи:
  - только запросы на минимум
- Динамическая постановка задачи:
  - кроме запросов на минимум запросы на изменение «$a_i := x$»
- Offline:
  - можно получить все запросы, а потом ответить на них
- Online:
  - следующий запрос получаем после того, как даём ответ на текущий

Статическая RMQ

- Построить таблицу с ответами

Статическая RMQ

- Построить таблицу с ответами:
  - ans[i, l] = RMQ([i, i + l - 1])

Статическая RMQ

- Построить таблицу с ответами:
  - ans[i, l] = RMQ([i, i + l - 1])
  - Размер таблицы квадратичный – неэффективно

Статическая RMQ

- Построить таблицу с ответами:
  - ans[i, l] = RMQ([i, i + l - 1])
  - Размер таблицы квадратичный – неэффективно
- Не нужно хранить всю таблицу?

Статическая RMQ

- Построить таблицу с ответами:
  - ans[i, l] = RMQ([i, i + l - 1])
  - Размер таблицы квадратичный – неэффективно
- Не нужно хранить всю таблицу – достаточно размера NlogN

Статическая RMQ

- Построить таблицу с ответами:
  - ans[i, l] = RMQ([i, i + l - 1])
  - Размер таблицы квадратичный – неэффективно
- Не нужно хранить всю таблицу – достаточно размера NlogN:
  - Только степени двойки в качестве l

Статическая RMQ

- Построить таблицу с ответами:
  - ans[i, l] = RMQ([i, i + l - 1])
  - Размер таблицы квадратичный – неэффективно
- Не нужно хранить всю таблицу – достаточно размера NlogN:
  - Только степени двойки в качестве l
  - RMQ([x, y]) = min(ans[x, l], ans[y – l + 1, l]) при подходящем l

Статическая RMQ

- Построить таблицу с ответами:
  - ans[i, l] = RMQ([i, i + l - 1])
  - Размер таблицы квадратичный – неэффективно
- Не нужно хранить всю таблицу – достаточно размера NlogN:
  - Только степени двойки в качестве l
  - RMQ([x, y]) = min(ans[x, l], ans[y – l + 1, l]) при подходящем l
    - O(1) на запрос

Статическая RMQ

- Построить таблицу с ответами:
  - ans[i, l] = RMQ([i, i + l - 1])
  - Размер таблицы квадратичный – неэффективно
- Не нужно хранить всю таблицу – достаточно размера NlogN:
  - Только степени двойки в качестве l
  - RMQ([x, y]) = min(ans[x, l], ans[y – l + 1, l]) при подходящем l
    - O(1) на запрос
  - Построение:
    - ans[i, 1] = a_i

Статическая RMQ

- Построить таблицу с ответами:
  - ans[i, l] = RMQ([i, i + l - 1])
  - Размер таблицы квадратичный – неэффективно
- Не нужно хранить всю таблицу – достаточно размера NlogN:
  - Только степени двойки в качестве l
  - RMQ([x, y]) = min(ans[x, l], ans[y – l + 1, l]) при подходящем l
    - O(1) на запрос
  - Построение:
    - ans[i, 1] = a_i
    - ans[i, 2 * l] = min(ans[i, l], ans[i + l, l])

Range Sum Query

Сумма вместо минимума

Range Sum Query

Сумма вместо минимума

- Статический случай RSQ

Range Sum Query

Сумма вместо минимума

- Статический случай RSQ решается с помощью кумулятивных сумм

Range Sum Query

Сумма вместо минимума

- Статический случай RSQ решается с помощью кумулятивных сумм
  - Предобработка за O(N) времени и памяти, запрос за O(1)

Range Sum Query

Сумма вместо минимума

- Статический случай RSQ решается с помощью кумулятивных сумм
  - Предобработка за O(N) времени и памяти, запрос за O(1) – оптимально

Range Sum Query

Сумма вместо минимума

- Статический случай RSQ решается с помощью кумулятивных сумм
  - Предобработка за O(N) времени и памяти, запрос за O(1) – оптимально
- Динамический случай RSQ/RMQ

Range Sum Query

Сумма вместо минимума

- Статический случай RSQ решается с помощью кумулятивных сумм
  - Предобработка за O(N) времени и памяти, запрос за O(1) – оптимально
- Динамический случай RSQ/RMQ – O(logN) на запрос каждого типа

Range Sum Query

Сумма вместо минимума

- Статический случай RSQ решается с помощью кумулятивных сумм
  - Предобработка за O(N) времени и памяти, запрос за O(1) – оптимально
- Динамический случай RSQ/RMQ – O(logN) на запрос каждого типа
  - Дерево отрезков (рассматривается на семинаре)

Least common ancestor

Дано дерево

- p(v) – родитель v

Least common ancestor

Дано дерево

- p(v) – родитель v
- w – предок v, если w = v или w – предок p(v)

Least common ancestor

Дано дерево

- p(v) – родитель v
- w – предок v, если w = v или w – предок p(v)
- lca(u, v) = w, если:
  - w – предок u
  - w – предок v

Least common ancestor

Дано дерево

- p(v) – родитель v
- w – предок v, если w = v или w – предок p(v)
- lca(u, v) = w, если:
  - w – предок u
  - w – предок v
  - при этом расстояние от корня до w максимально

Least common ancestor

Дано дерево

- p(v) – родитель v
- w – предок v, если w = v или w – предок p(v)
- lca(u, v) = w, если:
  - w – предок u
  - w – предок v
  - при этом расстояние от корня до w максимально
- Требуется обработать запросы вида «lca(u, v) – ?»

LCA: бинарный подъём

- Определим *предка вершины v уровня k* (обозначим «$p_k(v)$»):
  - $p_0(v) = v$

LCA: бинарный подъём

- Определим *предка вершины v уровня k* (обозначим «$p_k(v)$»):
  - $p_0(v) = v$
  - $p_k(v) = p_{k-1}(p(v))$

LCA: бинарный подъём

- Определим *предка вершины v уровня k* (обозначим «$p_k(v)$»):
  - $p_0(v) = v$
  - $p_k(v) = p_{k-1}(p(v))$, p(v) – родитель v, при этом p(root) = root

LCA: бинарный подъём

- Определим *предка вершины v уровня k* (обозначим «$p_k(v)$»):
  - $p_0(v) = v$
  - $p_k(v) = p_{k-1}(p(v))$, p(v) – родитель v, при этом p(root) = root
- Для каждой вершины запомним предков p_1, p_2, p_4, p_8...

LCA: бинарный подъём

- Определим *предка вершины v уровня k* (обозначим «$p_k(v)$»):
  - $p_0(v) = v$
  - $p_k(v) = p_{k-1}(p(v))$, p(v) – родитель v, при этом p(root) = root
- Для каждой вершины запомним предков p_1, p_2, p_4, p_8...
- Используем функцию is_upper(v, u), которая определяет является ли v предком u

LCA: бинарный подъём

```
lca(u, v) {
    if (is_upper(u, v)) return u;
```

LCA: бинарный подъём

```
lca(u, v) {
    if (is_upper(u, v)) return u;
    if (is_upper(v, u)) return v;
```

LCA: бинарный подъём

```
lca(u, v) {
    if (is_upper(u, v)) return u;
    if (is_upper(v, u)) return v;
    for (k = maxk; k > 1; k /= 2) {
        pu = pk(u);
        if (?) u = pu;
    }
```

LCA: бинарный подъём

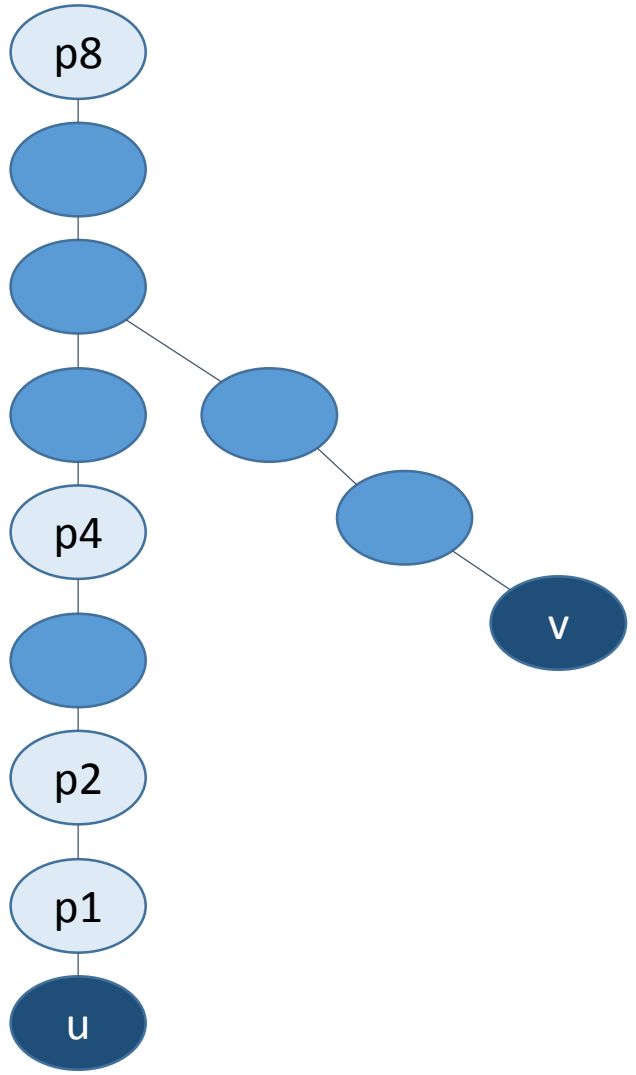
```
lca(u, v) {
    if (is_upper(u, v)) return u;
    if (is_upper(v, u)) return v;
    for (k = maxk; k > 1; k /= 2) {
        pu = pk(u);
        if (!is_upper(pu, v)) u = pu;
    }
```

LCA: бинарный подъём

```
lca(u, v) {
    if (is_upper(u, v)) return u;
    if (is_upper(v, u)) return v;
    for (k = maxk; k > 1; k /= 2) {
        pu = pk(u);
        if (!is_upper(pu, v)) u = pu;
    }
    return p(u);
}
```

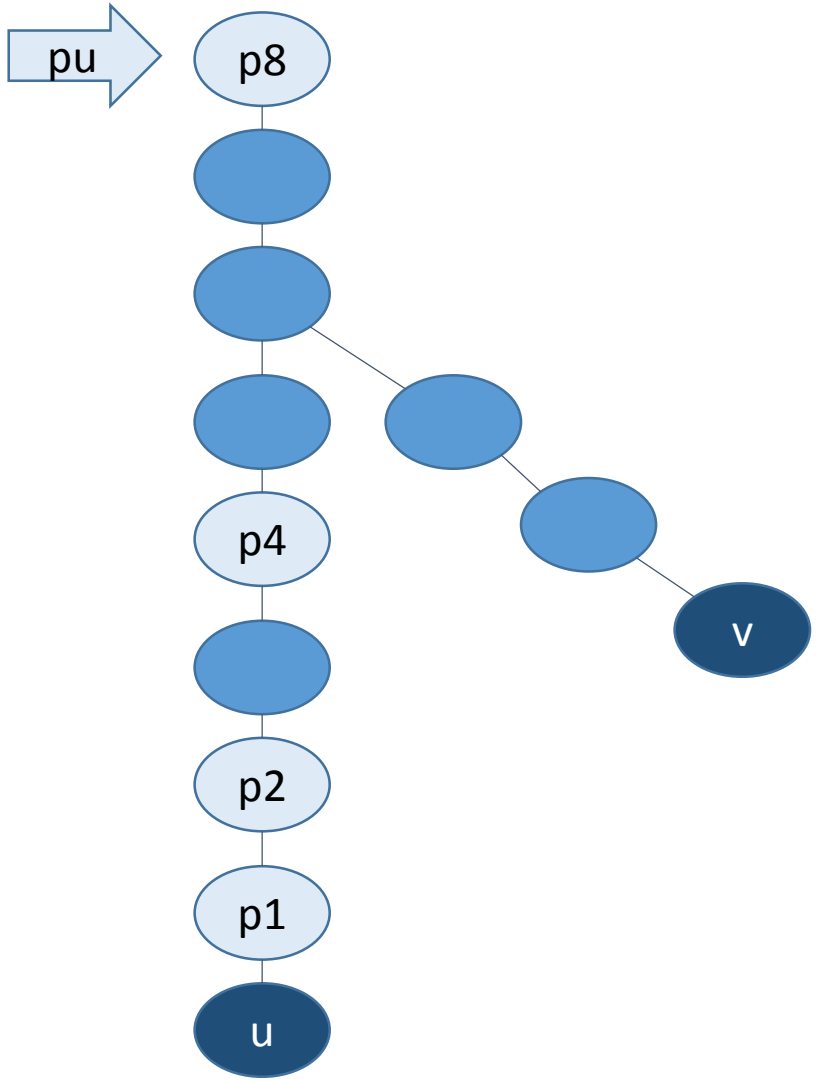
LCA: бинарный подъём

```
lca(u, v) {
    if (is_upper(u, v)) return u;
    if (is_upper(v, u)) return v;
    for (k = maxk; k > 1; k /= 2) {
        pu = pk(u);
        if (!is_upper(pu, v)) u = pu;
    }
    return p(u);
}
```



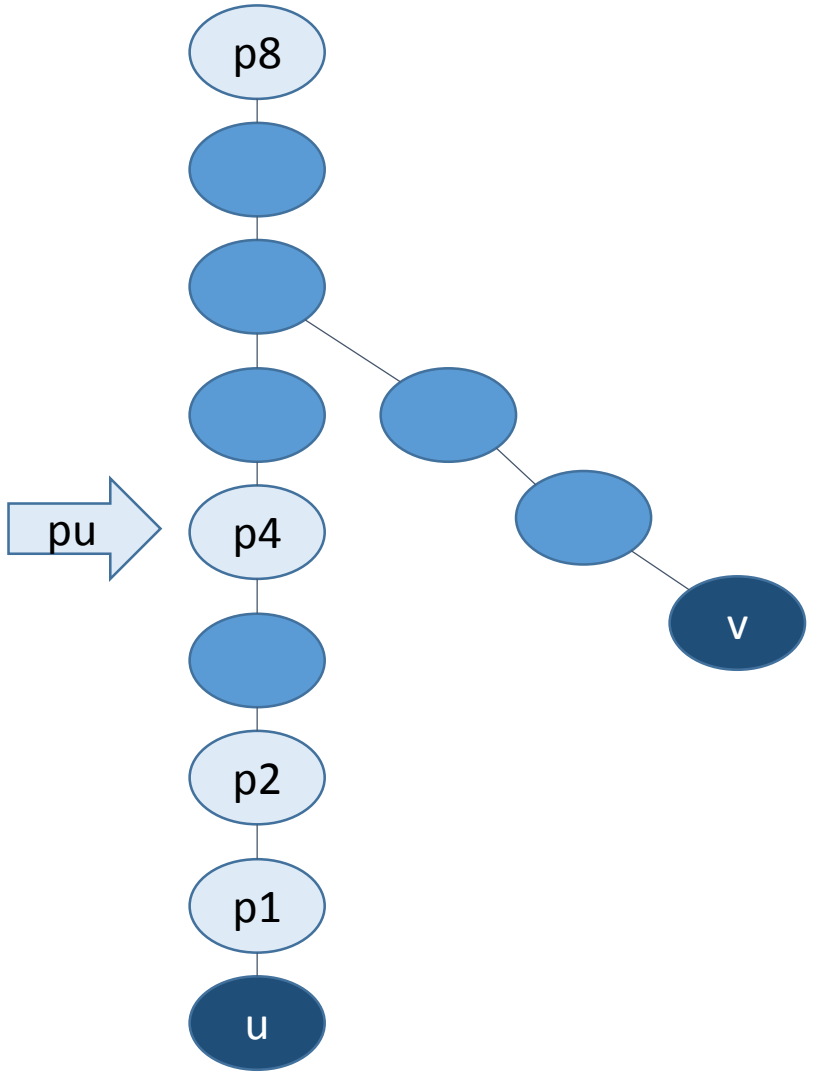
LCA: бинарный подъём

```
lca(u, v) {
    if (is_upper(u, v)) return u;
    if (is_upper(v, u)) return v;
    for (k = maxk; k > 1; k /= 2) {
        pu = pk(u);
        if (!is_upper(pu, v)) u = pu;
    }
    return p(u);
}
```



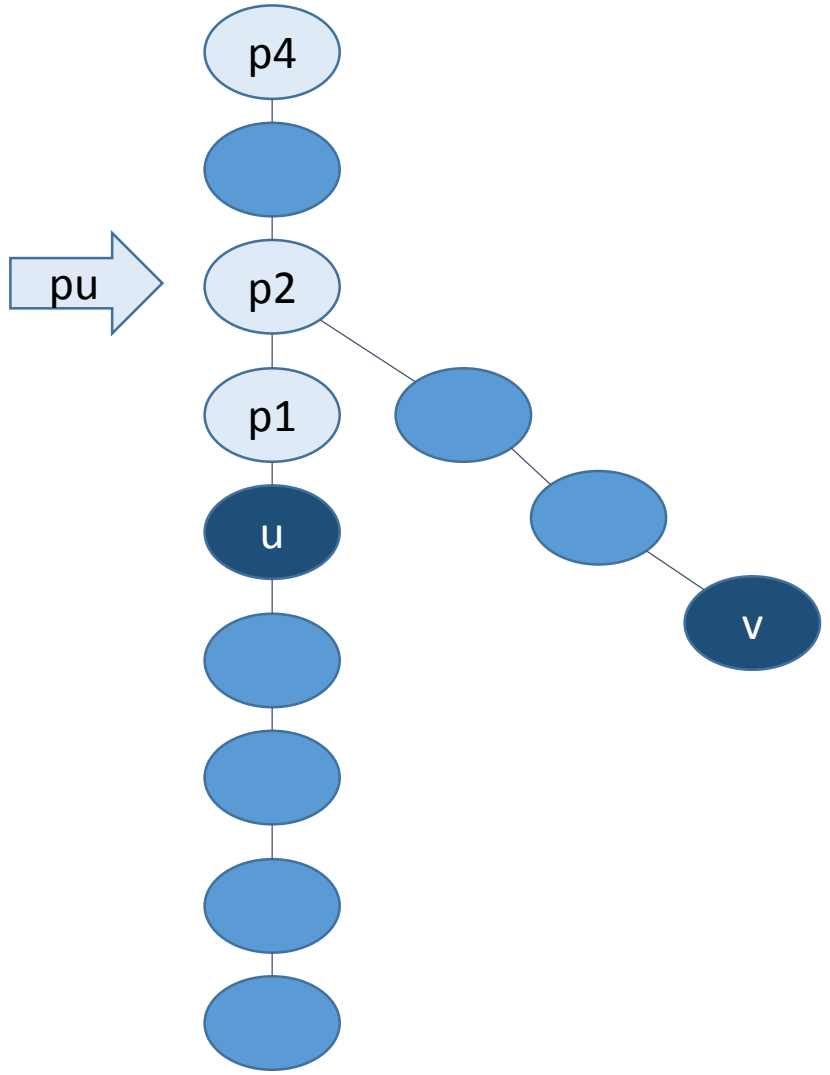
LCA: бинарный подъём

```
lca(u, v) {
    if (is_upper(u, v)) return u;
    if (is_upper(v, u)) return v;
    for (k = maxk; k > 1; k /= 2) {
        pu = pk(u);
        if (!is_upper(pu, v)) u = pu;
    }
    return p(u);
}
```



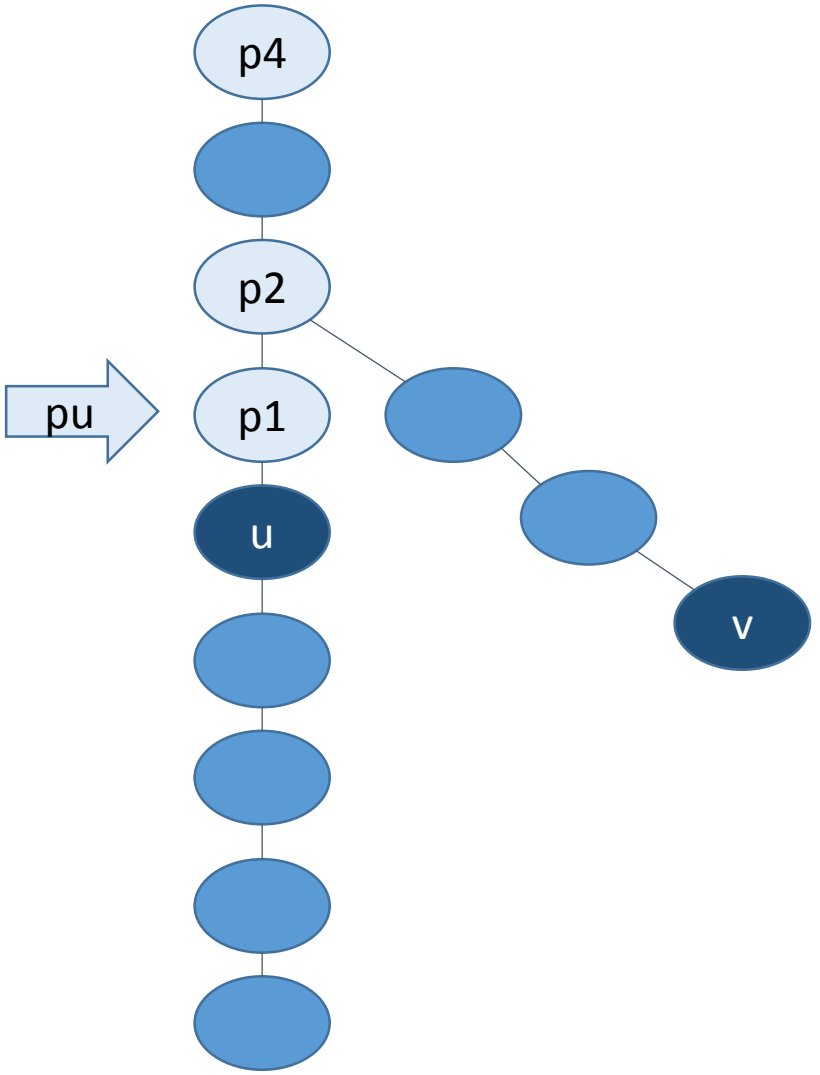
LCA: бинарный подъём

```
lca(u, v) {
    if (is_upper(u, v)) return u;
    if (is_upper(v, u)) return v;
    for (k = maxk; k > 1; k /= 2) {
        pu = pk(u);
        if (!is_upper(pu, v)) u = pu;
    }
    return p(u);
}
```



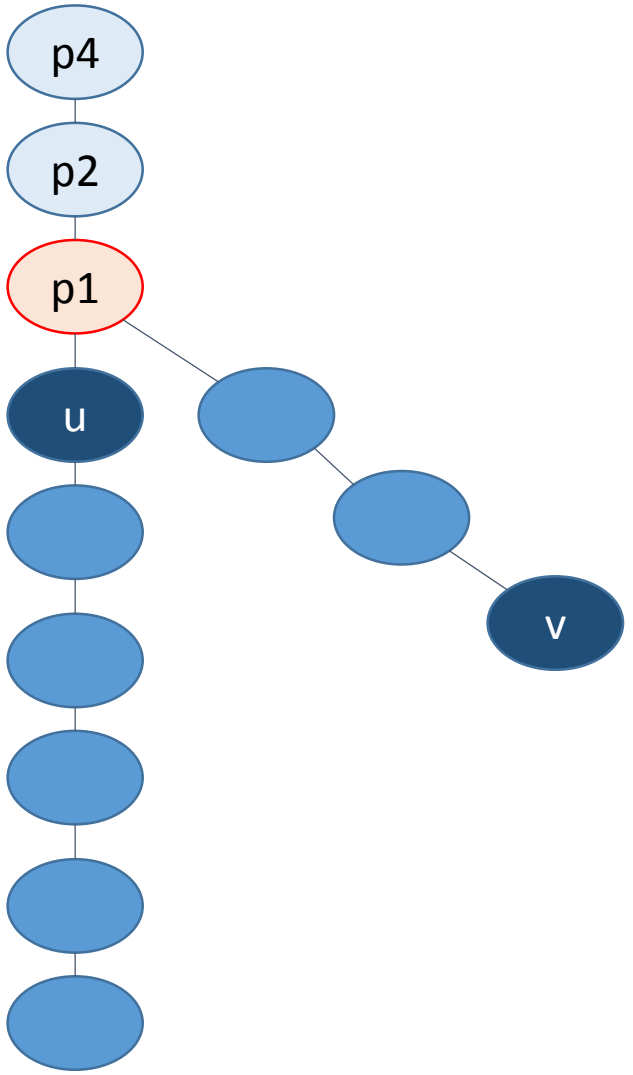
LCA: бинарный подъём

```
lca(u, v) {
    if (is_upper(u, v)) return u;
    if (is_upper(v, u)) return v;
    for (k = maxk; k > 1; k /= 2) {
        pu = pk(u);
        if (!is_upper(pu, v)) u = pu;
    }
    return p(u);
}
```



LCA: бинарный подъём

```
lca(u, v) {
    if (is_upper(u, v)) return u;
    if (is_upper(v, u)) return v;
    for (k = maxk; k > 1; k /= 2) {
        pu = pk(u);
        if (!is_upper(pu, v)) u = pu;
    }
    return p(u);
}
```



LCA: бинарный подъём

- Сложность
  - logN памяти для каждой вершины для хранения ссылок на предков

LCA: бинарный подъём

- Сложность
  - logN памяти для каждой вершины для хранения ссылок на предков
  - is_upper за O(1) с O(N) на препроцессинг

LCA: бинарный подъём

- Сложность
  - logN памяти для каждой вершины для хранения ссылок на предков
  - is_upper за O(1) с O(N) на препроцессинг

  - Препроцессинг за O(NlogN) времени и памяти

LCA: бинарный подъём

- Сложность
  - logN памяти для каждой вершины для хранения ссылок на предков
  - is_upper за O(1) с O(N) на препроцессинг

  - Препроцессинг за O(NlogN) времени и памяти
  - O(logN) времени на запрос

Сведение RMQ к LCA

- На основе последовательности a_1, a_2, ... a_n сделаем пары ($x=i$, $y=a_i$)

Сведение RMQ к LCA

- На основе последовательности a_1, a_2, ... a_n сделаем пары ($x=i$, $y=a_i$)
- Из таких пар построим декартово дерево

Сведение RMQ к LCA

- На основе последовательности a_1, a_2, ... a_n сделаем пары ($x=i$, $y=a_i$)
- Из таких пар построим декартово дерево, сложность = O(N)

Сведение RMQ к LCA

- На основе последовательности a_1, a_2, ... a_n сделаем пары ($x=i$, $y=a_i$)
- Из таких пар построим декартово дерево, сложность = O(N)
- Индексу j в последовательности соответствует вершина n(j)

Сведение RMQ к LCA

- На основе последовательности a_1, a_2, ... a_n сделаем пары ($x=i$, $y=a_i$)
- Из таких пар построим декартово дерево, сложность = O(N)
- Индексу j в последовательности соответствует вершина n(j)

Вычисляем ответ на RMQ([l, r])

Сведение RMQ к LCA

- На основе последовательности a_1, a_2, ... a_n сделаем пары ($x=i$, $y=a_i$)
- Из таких пар построим декартово дерево, сложность = O(N)
- Индексу j в последовательности соответствует вершина n(j)

Вычисляем ответ на RMQ([l, r])

- Пусть m = LCA(n(l), n(r))

Сведение RMQ к LCA

- На основе последовательности a_1, a_2, ... a_n сделаем пары ($x=i$, $y=a_i$)
- Из таких пар построим декартово дерево, сложность = O(N)
- Индексу j в последовательности соответствует вершина n(j)

Вычисляем ответ на RMQ([l, r])

- Пусть m = LCA(n(l), n(r))
- Тогда
  - m.x ∈ [l, r]

Сведение RMQ к LCA

- На основе последовательности a_1, a_2, ... a_n сделаем пары ($x=i$, $y=a_i$)
- Из таких пар построим декартово дерево, сложность = O(N)
- Индексу j в последовательности соответствует вершина n(j)

Вычисляем ответ на RMQ([l, r])

- Пусть m = LCA(n(l), n(r))
- Тогда
  - m.x ∈ [l, r]
  - m.y – минимальный приоритет среди вершин с x ∈ [l, r]

Сведение RMQ к LCA

- На основе последовательности a_1, a_2, ... a_n сделаем пары ($x=i$, $y=a_i$)
- Из таких пар построим декартово дерево, сложность = O(N)
- Индексу j в последовательности соответствует вершина n(j)

Вычисляем ответ на RMQ([l, r])

- Пусть m = LCA(n(l), n(r))
- Тогда
  - m.x ∈ [l, r]
  - m.y – минимальный приоритет среди вершин с x ∈ [l, r]
- Ответом будет m.y

Сведение LCA к ±1-RMQ

- ±1-RMQ: RMQ при ограничении $|a_i - a_{i+1}| = 1$

Сведение LCA к ±1-RMQ

- ±1-RMQ: RMQ при ограничении $|a_i - a_{i+1}| = 1$
- Для сведения задачи LCA к RMQ из дерева сформируем:
    - последовательность $\{a_i\}$, $|a_i - a_{i+1}| = 1$

Сведение LCA к ±1-RMQ

- ±1-RMQ: RMQ при ограничении $|a_i – a_{i+1}| = 1$
- Для сведения задачи LCA к RMQ из дерева сформируем:
  - последовательность $\{a_i\}$, $|a_i – a_{i+1}| = 1$
  - вспомогательную последовательность $\{v_i\}$

Сведение LCA к ±1-RMQ

- ±1-RMQ: RMQ при ограничении $|a_i - a_{i+1}| = 1$
- Для сведения задачи LCA к RMQ из дерева сформируем:
  - последовательность $\{a_i\}$, $|a_i - a_{i+1}| = 1$
  - вспомогательную последовательность $\{v_i\}$
  - Эйлеров обход дерева:
    - по каждому ребру дерева проходим дважды (вверх и вниз)

Сведение LCA к ±1-RMQ

- ±1-RMQ: RMQ при ограничении $|a_i - a_{i+1}| = 1$
- Для сведения задачи LCA к RMQ из дерева сформируем:
  - последовательность $\{a_i\}$, $|a_i - a_{i+1}| = 1$
  - вспомогательную последовательность $\{v_i\}$
  - Эйлеров обход дерева:
    - по каждому ребру дерева проходим дважды (вверх и вниз)
  - Выполним эйлеров обход дерева начиная с корня

Сведение LCA к ±1-RMQ

- ±1-RMQ: RMQ при ограничении $|a_i - a_{i+1}| = 1$
- Для сведения задачи LCA к RMQ из дерева сформируем:
  - последовательность $\{a_i\}$, $|a_i - a_{i+1}| = 1$
  - вспомогательную последовательность $\{v_i\}$
  - Эйлеров обход дерева:
    - по каждому ребру дерева проходим дважды (вверх и вниз)
  - Выполним эйлеров обход дерева начиная с корня
  - Всякий раз, заходя в вершину, выписаваем
    - a_i – глубину вершины

Сведение LCA к ±1-RMQ

- ±1-RMQ: RMQ при ограничении $|a_i - a_{i+1}| = 1$
- Для сведения задачи LCA к RMQ из дерева сформируем:
  - последовательность $\{a_i\}$, $|a_i - a_{i+1}| = 1$
  - вспомогательную последовательность $\{v_i\}$
  - Эйлеров обход дерева:
    - по каждому ребру дерева проходим дважды (вверх и вниз)
  - Выполним эйлеров обход дерева начиная с корня
  - Всякий раз, заходя в вершину, выписаваем
    - a_i – глубину вершины
    - v_i – номер вершины

Сведение LCA к ±1-RMQ

- ±1-RMQ: RMQ при ограничении $|a_i - a_{i+1}| = 1$
- Для сведения задачи LCA к RMQ из дерева сформируем:
  - последовательность $\{a_i\}$, $|a_i - a_{i+1}| = 1$
  - вспомогательную последовательность $\{v_i\}$
  - Эйлеров обход дерева:
    - по каждому ребру дерева проходим дважды (вверх и вниз)
  - Выполним эйлеров обход дерева начиная с корня
  - Всякий раз, заходя в вершину, выписаваем
    - a_i – глубину вершины
    - v_i – номер вершины
  - Длина $\{a_i\}$ и $\{v_i\}$ линейна от размера дерева

Сведение LCA к ±1-RMQ

- ±1-RMQ: RMQ при ограничении $|a_i - a_{i+1}| = 1$
- Для сведения задачи LCA к RMQ из дерева сформируем:
  - последовательность $\{a_i\}$, $|a_i - a_{i+1}| = 1$
  - вспомогательную последовательность $\{v_i\}$
  - Эйлеров обход дерева:
    - по каждому ребру дерева проходим дважды (вверх и вниз)
  - Выполним эйлеров обход дерева начиная с корня
  - Всякий раз, заходя в вершину, выписаваем
    - a_i – глубину вершины
    - v_i – номер вершины
  - Длина $\{a_i\}$ и $\{v_i\}$ линейна от размера дерева
    - O(N) времени и памяти

Сведение LCA к ±1-RMQ

- Получили $\{a_i\}$ и $\{v_i\}$, теперь решаем LCA

Сведение LCA к ±1-RMQ

- Получили $\{a_i\}$ и $\{v_i\}$, теперь решаем LCA
  - Пусть $p(u) = j : v_j = u$

Сведение LCA к ±1-RMQ

- Получили $\{a_i\}$ и $\{v_i\}$, теперь решаем LCA
  - Пусть $p(u) = j : v_j = u$
- Находим LCA(u, w)

Сведение LCA к ±1-RMQ

- Получили $\{a_i\}$ и $\{v_i\}$, теперь решаем LCA
  - Пусть $p(u) = j : v_j = u$
- Находим LCA(u, w)
  - Решаем RMQ[p(u), p(w)]

Сведение LCA к ±1-RMQ

- Получили $\{a_i\}$ и $\{v_i\}$, теперь решаем LCA
  - Пусть $p(u) = j : v_j = u$
- Находим LCA(u, w)
  - Решаем RMQ[p(u), p(w)]
  - Пусть ответ на этот запрос имеет индекс k

Сведение LCA к ±1-RMQ

- Получили $\{a_i\}$ и $\{v_i\}$, теперь решаем LCA
  - Пусть $p(u) = j : v_j = u$
- Находим LCA(u, w)
  - Решаем RMQ[p(u), p(w)]
  - Пусть ответ на этот запрос имеет индекс k
  - Тогда $LCA(u, w) = v_k$

Сведение LCA к ±1-RMQ

- Получили $\{a_i\}$ и $\{v_i\}$, теперь решаем LCA
  - Пусть $p(u) = j : v_j = u$
- Находим LCA(u, w)
  - Решаем RMQ[p(u), p(w)]
  - Пусть ответ на этот запрос имеет индекс k
  - Тогда $LCA(u, w) = v_k$
- p сохраним в массиве

Сведение LCA к ±1-RMQ

- Получили $\{a_i\}$ и $\{v_i\}$, теперь решаем LCA
  - Пусть $p(u) = j : v_j = u$
- Находим LCA(u, w)
  - Решаем RMQ[p(u), p(w)]
  - Пусть ответ на этот запрос имеет индекс k
  - Тогда $LCA(u, w) = v_k$
- p сохраним в массиве
  - O(N) памяти и времени на предобработку

Сведение LCA к ±1-RMQ

- Получили $\{a_i\}$ и $\{v_i\}$, теперь решаем LCA
  - Пусть $p(u) = j : v_j = u$
- Находим LCA(u, w)
  - Решаем RMQ[p(u), p(w)]
  - Пусть ответ на этот запрос имеет индекс k
  - Тогда $LCA(u, w) = v_k$
- p сохраним в массиве
  - O(N) памяти и времени на предобработку
  - O(1) времени на запрос дополнительно к RMQ

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Сохраним минимум на каждом префиксе и суффиксе каждого блока

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Сохраним минимум на каждом префиксе и суффиксе каждого блока
  - O(N) памяти

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Сохраним минимум на каждом префиксе и суффиксе каждого блока
  - O(N) памяти
- Минимум в блоке j – элемент новой последовательности b_j

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Сохраним минимум на каждом префиксе и суффиксе каждого блока
  - O(N) памяти
- Минимум в блоке j – элемент новой последовательности b_j
- RMQ([l, r])

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Сохраним минимум на каждом префиксе и суффиксе каждого блока
  - O(N) памяти
- Минимум в блоке j – элемент новой последовательности b_j
- RMQ([l, r])

= min {

минимум на суффиксе начального блока

минимум на префиксе конечого блока

RMQ на $\{b_j\}$

}

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Сохраним минимум на каждом префиксе и суффиксе каждого блока
  - O(N) памяти
- Минимум в блоке j – элемент новой последовательности b_j
- RMQ([l, r])

  = min {

    минимум на суффиксе начального блока

    минимум на префиксе конечого блока

    RMQ на $\{b_j\}$

  }

- Если K = O(log(N)), получаем O(N) на предобработку $\{b_j\}$

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока?

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)
  - O(K^2 * N/K) = O(NK) памяти

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)
  - O(K^2 * N/K) = O(NK) памяти – не годится

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)
  - O(K^2 * N/K) = O(NK) памяти – не годится
  - Используем ±1-свойство

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)
  - O(K^2 * N/K) = O(NK) памяти – не годится
  - Используем ±1-свойство: разных типов блоков O(2^K)

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)
  - O(K^2 * N/K) = O(NK) памяти – не годится
  - Используем ±1-свойство: разных типов блоков O(2^K)
  - Достаточно O(K^2 * 2^K) блоков

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)
  - O(K^2 * N/K) = O(NK) памяти – не годится
  - Используем ±1-свойство: разных типов блоков O(2^K)
  - Достаточно O(K^2 * 2^K) блоков
    - K = logN

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)
  - O(K^2 * N/K) = O(NK) памяти – не годится
  - Используем ±1-свойство: разных типов блоков O(2^K)
  - Достаточно O(K^2 * 2^K) блоков
    - K = logN: O(K^2 * 2^K) = O(N * logN^2)

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)
  - O(K^2 * N/K) = O(NK) памяти – не годится
  - Используем ±1-свойство: разных типов блоков O(2^K)
  - Достаточно O(K^2 * 2^K) блоков
    - K = logN: O(K^2 * 2^K) = O(N * logN^2) – опять не годится

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)
  - O(K^2 * N/K) = O(NK) памяти – не годится
  - Используем ±1-свойство: разных типов блоков O(2^K)
  - Достаточно O(K^2 * 2^K) блоков
    - K = logN: O(K^2 * 2^K) = O(N * logN^2) – опять не годится
    - K = logN/2

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)
  - O(K^2 * N/K) = O(NK) памяти – не годится
  - Используем ±1-свойство: разных типов блоков O(2^K)
  - Достаточно O(K^2 * 2^K) блоков
    - K = logN: O(K^2 * 2^K) = O(N * logN^2) – опять не годится
    - K = logN/2: O(K^2 * 2^K) = O(sqrt(N) * logN^2)

±1-RMQ: алгоритм Фарах-Колтона-Бендера

- Последовательность $\{a_i\}$ поделим на блоки длины K
- Что, если запрос внутри блока
  - Используем предподсчёт
    - Начало и конец в блоке: O(K^2)
    - Всего блоков O(N/K)
  - O(K^2 * N/K) = O(NK) памяти – не годится
  - Используем ±1-свойство: разных типов блоков O(2^K)
  - Достаточно O(K^2 * 2^K) блоков
    - K = logN: O(K^2 * 2^K) = O(N * logN^2) – опять не годится
    - K = logN/2: O(K^2 * 2^K) = O(sqrt(N) * logN^2) – подходит

Сведение задач

Задача статическая, online

- RMQ -> LCA
- LCA -> ±1-RMQ
- ±1-RMQ -> RMQ

Сведение задач

Задача статическая, online

- RMQ -> LCA
- LCA -> ±1-RMQ
- ±1-RMQ -> RMQ

- Любую из этих задач решаем за O(1) на запрос и O(N) на предобработку

Offline LCA: алгоритм Тарьяна

- Есть структура данных DCU с представителями

Offline LCA: алгоритм Тарьяна

- Есть структура данных DCU с представителями
- В начале работы все множества одноэлементны

Offline LCA: алгоритм Тарьяна

- Есть структура данных DCU с представителями
- В начале работы все множества одноэлементны
- Методы:
  - join(x, y) – объединяет множества содержащие элементы x и y

Offline LCA: алгоритм Тарьяна

- Есть структура данных DCU с представителями
- В начале работы все множества одноэлементны
- Методы:
  - join(x, y) – объединяет множества содержащие элементы x и y
    - представителем нового множества назначается представитель множества x

Offline LCA: алгоритм Тарьяна

- Есть структура данных DCU с представителями
- В начале работы все множества одноэлементны
- Методы:
  - join(x, y) – объединяет множества содержащие элементы x и y
    - представителем нового множества назначается представитель множества x
  - get(x) – получаем представителя множества, содержащего элемент x

Offline LCA: алгоритм Тарьяна

- В вершине v записываем список таких u, что есть запрос LCA(u, v)

Offline LCA: алгоритм Тарьяна

- В вершине v записываем список таких u, что есть запрос LCA(u, v)
- Совершаем обход дерева в глубину

Offline LCA: алгоритм Тарьяна

- В вершине v записываем список таких u, что есть запрос LCA(u, v)
- Совершаем обход дерева в глубину
- Когда обход посещает v, проходим по её списку

Offline LCA: алгоритм Тарьяна

- В вершине v записываем список таких u, что есть запрос LCA(u, v)
- Совершаем обход дерева в глубину
- Когда обход посещает v, проходим по её списку
  - Если u из списка ещё не посещена – пропускаем её

Offline LCA: алгоритм Тарьяна

- В вершине v записываем список таких u, что есть запрос LCA(u, v)
- Совершаем обход дерева в глубину
- Когда обход посещает v, проходим по её списку
  - Если u из списка ещё не посещена – пропускаем её
  - Если u посещена, LCA(u, v) – это представитель множества с u

Offline LCA: алгоритм Тарьяна

- В вершине v записываем список таких u, что есть запрос LCA(u, v)
- Совершаем обход дерева в глубину
- Когда обход посещает v, проходим по её списку
  - Если u из списка ещё не посещена – пропускаем её
  - Если u посещена, LCA(u, v) – это представитель множества с u
    - LCA(u, v) = get(u)

Offline LCA: алгоритм Тарьяна

- В вершине v записываем список таких u, что есть запрос LCA(u, v)
- Совершаем обход дерева в глубину
- Когда обход посещает v, проходим по её списку
  - Если u из списка ещё не посещена – пропускаем её
  - Если u посещена, LCA(u, v) – это представитель множества с u
    - LCA(u, v) = get(u)
- Когда обход выходит из v, добавляем множество вершин поддерева v к множеству её родителя

Offline LCA: алгоритм Тарьяна

- В вершине v записываем список таких u, что есть запрос LCA(u, v)
- Совершаем обход дерева в глубину
- Когда обход посещает v, проходим по её списку
  - Если u из списка ещё не посещена – пропускаем её
  - Если u посещена, LCA(u, v) – это представитель множества с u
    - LCA(u, v) = get(u)
- Когда обход выходит из v, добавляем множество вершин поддерева v к множеству её родителя
  - join(p[v], v)