SPb HSE, 1 курс, весна 2024/25
Практика по алгоритмам #29

RMQ и LCA

21 мая

Собрано 20 мая 2025 г. в 19:24

Содержание

RMQ и LCA

Все задачи по умолчанию online, static.

1. **Полный П.**

   Есть бинарная, **возможно, не ассоциативная** функция f.
   Запросы: посчитать f на отрезке. $\langle \mathcal{O}(n^2), \mathcal{O}(1) \rangle$.

2. **gcd ±1**

   Решите задачу gcd ±1. $\langle \mathcal{O}(n), \mathcal{O}(1) \rangle$.

3. **Рекурсивный Sparse Table**

   Используя только Sparse Table, решите RMQ за $\langle \mathcal{O}(n \log^* n), \mathcal{O}(\log^* n) \rangle$.
   Выпишите явно рекуррентные соотношения для времени и памяти работы алгоритма.

4. **Короткий код LCA**

   Пусть можно сделать $\mathcal{O}(n)$ предподсчета и отвечать на $\mathrm{LCA}(a, b)$ запрос за $\mathcal{O}(\mathrm{dist}(a, b))$.
   Напишите как можно более короткий код ответа на запрос LCA.
   Сам предподсчёт не пишите, его длину также не учитывайте.

5. **LCA без памяти**

   Дерево задано массивом предков. Кроме массива предков никакого предподсчёта нет.
   Найдите `LCA(a, b)` за $\mathcal{O}(n)$ с $\mathcal{O}(1)$ дополнительной памяти.

6. **LA в offline**

   a) Для заданного фиксированного k посчитайте для всех вершин дерева `up[v,k]` за $\mathcal{O}(n)$.
   b) Решите задачу LA в offline за $\mathcal{O}(n + m)$.
   c) Используя умение из предыдущего пункта, решите LCA за $\langle \mathcal{O}(n), \mathcal{O}(\sqrt{n}) \rangle$.

7. **LCA – не только LCA**

   Дано дерево, у каждой вершины есть вес.
   Запросы: минимум на пути из a в b. $\langle \mathcal{O}(n \log n), \mathcal{O}(\log n) \rangle$.

8. **Двоичные подъёмы – не только LCA**

   Дан **ор**граф, $\forall v \ \deg_{\text{out}} v = 1$. Запросы: из вершины v сделать k шагов вперёд.
   a) $\langle \mathcal{O}(n \log(\max k)), \mathcal{O}(\log k) \rangle$.
   b) $\langle \mathcal{O}(n \log n), \mathcal{O}(\log n) \rangle$.

9. **Динамические двоичные подъёмы**

   Дано дерево и запросы: LCA; подвесить лист. $\langle \mathcal{O}(n \log n), \mathcal{O}(\log n) \rangle$.

10. **Фарах-Колтон-Бендер для произведения**

    Дан непростой целый модуль m и массив из n чисел. Различных чисел в массиве не более $\sqrt{n}$, $\forall i \ |a_{i+1} - a_i| = 1$. Запросы: произведение на отрезке по модулю m. $\langle \mathcal{O}(n), \mathcal{O}(1) \rangle$.

    Указание: попробуйте придумать аналог алгоритма Фараха-Колтона-Бендера.
    Почему sparse table не работает? Почему недостаточно такого же предподсчёта?
    Решите проблему с предподсчётом. Проблема со sparse table разрешится в дз.

11. **Поддерево – это отрезок**

    Дано дерево. Запросы: пометить вершину, снять пометку с вершины, число помеченных вершин в поддереве. $\langle \mathcal{O}(n), \mathcal{O}(\log n) \rangle$.

12. **Два поддерева – два отрезка**

    Даны два дерева из n вершин. Вершины каждого помечены числами от 1 до n без повторений. Запрос `get`(v, u): вершина v из первого дерева, вершина u из второго, вопрос — сколько чисел лежат одновременно в поддеревьях и v, и u? $\langle \mathcal{O}(n \log n), \mathcal{O}(\log n) \rangle$.

13. **Пересечение путей**

    Дано дерево. Запросы: длина пересечения путей $v_1 \rightsquigarrow v_2$ и $u_1 \rightsquigarrow u_2$.

14. (*) **LCA на отрезке**

    Дано дерево. Дана перестановка его вершин. Запрос – LCA всех вершин отрезка.

15. (*) **Леонард**

    Пусть у вас есть корневое дерево и нужно уметь $LCA(a, b)$ и $parent[a] = b$ (гарантируется, что останется деревом).

16. (*) **Dynamic online max на пути**

    Дано дерево с весами на вершинах.
    Запросы: изменить вес вершины; найти max на пути $u \rightsquigarrow v$. $\langle \mathcal{O}(n \log n), \mathcal{O}(\log^2 n) \rangle$.

    Указание: запрос на пути – это запрос на прямоугольнике.

17. (*) **Dynamic online Σ на пути**

    Дано дерево с весами на ребрах. Запросы: изменить вес ребра; найти сумму на пути $u \rightsquigarrow v$.

18. (*) **Дуги на окружности**

    Даны дуги на окружности. Выбрать max число попарно непересекающихся дуг.
    Нужно детерминированное решение за $\mathcal{O}(\text{sort} + n)$.

    Обратите внимание: мы уже решали эту задачу пару раз, но теперь вы знаете LA!

19. (*) **LCA и подвешивание деревьев**

    Дан лес подвешенных деревьев.
    Запросы: LCA; подвесить дерево с корнем v к вершине u другого дерева.

    a) Двоичными подъёмами за $\mathcal{O}(\log^2 n)$.
    b) Эйлеровым обходом за $\mathcal{O}(\log n)$.

20. (*) **ФКБ и иностранные агенты**

    Напишите код предподсчёта для ФКБ, использующий $\mathcal{O}(2^k)$ памяти и времени. Код нужно написать полностью (максимально подробно). Следите, чтобы он получился коротким.

Разбор задач практики

1. **Полный П.**

   Пример неассоциативной функции: возведение в степень, $a^{(b^c)} \neq (a^b)^c$.
   Пусть $f(a_l, \ldots, a_r) = f(f(a_l, \ldots, a_{r-1}), a_r)$ (левая ассоциативность).
   Просто предподсчитаем ответы на все запросы.

```
for (int i = 0; i < n; ++i):
    ans[i][i] = a[i];
    for (int j = i + 1; j < n; ++j)
        ans[i][j] = f(ans[i][j - 1], a[j]);
```

2. **gcd ±1**

   Задача-шутка, $\gcd(x, x+1) = 1$.
   $\Rightarrow$ для отрезка длины один ответ – сам элемент отрезка, а для других отрезков ответ 1.

3. **Рекурсивный Sparse Table**

   Бьём массив на куски длины $k = \log n$, предподсчитаем минимумы на суффиксах и префиксах в каждом куске. Заведем Sparse Table на массиве из минимумов кусков, как в алгоритме Фараха-Колтона-Бендера.

   Сделанный предподсчёт весит $\mathcal{O}(n)$ и позволяет ответить на все запросы кроме маленьких отрезков, целиком лежащих в одном из кусков. Для них в каждом куске заведем такую же структуру. Оценим время построения $b(n)$ и время запроса $q(n)$:

$$b(n) = \mathcal{O}(n) + \frac{n}{\log n} b(\log n)$$

$$q(n) = \mathcal{O}(1) + q(\log n)$$

   Получаем $q(n) = \mathcal{O}(\log^* n)$ и $b(n) = \mathcal{O}(n \log^* n)$.
   Если не делать предподсчёт на префиксах и суффиксах, то время запроса $q_{slow}(n)$:

$$q_{\text{slow}}(n) = \mathcal{O}(1) + 2q_{\text{slow}}(\log n) = 2^{\log^* n}$$

4. **Короткий код LCA**

```
int lca(int a, int b): // O(distance between a and b)
    while (depth[a] > depth[b]) a = parent[a];
    while (depth[b] > depth[a]) b = parent[b];
    while (a != b) a = parent[a], b = parent[b];
    return a;
```

   А если посчитаны времена входа и выхода, можно так.

```
int lca(int a, int b):
    while (tin[a] > tin[b] || tout[b] > tout[a]) a = parent[a];
    return a;
```

   Что можно записать в одну строку:

```
int lca(int a, int b):
    return isAncestor(a, b) ? a : lca(p[a], b);
```

5. **LCA без памяти**

   Считаем глубины `a` и `b`, просто поднимаясь из них в корень.
   Затем выравниваем глубины и параллельно поднимаемся, пока указатели не встретятся.

6. **LA в offline**

   a) Обходя дерево `dfs`-ом, поддерживаем в стеке `path` путь от корня до вершины.
   `up[v,k] = path[top-k]`.
   b) `dfs` из предыдущего пункта умеет не только на фиксированное k вверх, а на любое.
   c) Выберем $k = \sqrt{n}$. Поднимаясь вверх, сперва пытаемся делать шаги на $\sqrt{n}$, потом на 1.

7. **LCA – не только LCA**

   Двоичные подъёмы. Кроме $\texttt{up}[v, 2^k]$ вычисляем минимумы на пути от v до $\texttt{up}[v, 2^k]$.
   `minw[v, k] = min(minw[v, k-1], minw[up[v, k-1], k-1])`.

8. **Двоичные подъёмы – не только LCA**

   a) Предподсчитаем шаги на степени двойки, как в двоичных подъемах.
   b) Наш граф – набор циклов, у которых на каждой вершине может расти дерево.
   Предподсчитаем для каждой вершины длину цикла, на котором она находится (если находится). Посчитаем двоичные подъемы за $\mathcal{O}(n \log n)$.
   При запросе после обычных прыжком прыгаем еще на $k' \bmod c$, где k' – сколько осталось, c – длина цикла, в который попали.

9. **Динамические двоичные подъёмы**

   Просто насчитываем двоичные подъемы из нового листа за $\mathcal{O}(\log n)$.

10. **Фарах-Колтон-Бендер для произведения**

    Вспомним идею ФКБ. Разбили массив на куски длины $k{+}1$. На массиве длины $\frac{n}{k+1}$ сделали sparse table и Для битовых масок длины k сделали предподсчёт. Две проблемы:

    a) Произведение – не идемпотентная функция, а куски в sparse table перекрываются.
    b) Произведение зависит от первого элемента.

    Первую проблему решит задача из дз. Вторую решим топорно – предподсчёт будет зависеть и от первого элемента тоже. Итого $\mathcal{O}(\sqrt{n} \cdot 2^k)$ значений. Возьмём $k = \frac{1}{3} \log n$, получим $\mathcal{O}(n)$.

11. **Поддерево – это отрезок**

    Возьмем Эйлеров обход дерева, в котором каждая вершина встречается один раз (сперва записали вершину, затем рекурсивно детей). Каждое поддерево соответствует отрезку обхода. Начало отрезка – позиция корня поддерева. Длина – размер поддерева. Отвечаем на запросы деревом отрезков.

12. **Два поддерева – два отрезка**

    Каждому числу назначим две координаты: позицию в Эйлеровом обходе первого и второго дерева. Итого у нас n 2D точек. Ответ на $\texttt{get}(u, v)$ – число точек в прямоугольнике. Делать это в online за $\langle \mathcal{O}(n \log n), \mathcal{O}(\log n) \rangle$ умеем. Например, деревом отрезков сортированных массивов.

13. **Пересечение путей**

    Разбиваем пути по LCA на вертикальные, пересекаем вертикальные.
    Чтобы пересечь вертикальные пути $v_1 \nearrow v_2$ и $u_1 \nearrow u_2$, находим $w = \mathrm{LCA}(v_1, u_1)$.

Отвечаем $\min(\text{depth}(v_2), \text{depth}(u_2)) - \text{depth}(w)$.

14. **(*) LCA на отрезке**

Сделаем `dfs`, посчитаем время входа `tin` всех вершин.
Ответ – LCA двух вершин, с min и max `tin` на отрезке.

Альтернативное решение. Дерево отрезков и sparse table же можно делать с любой операцией? LCA – вполне себе. Дерево отрезков даст $\langle \mathcal{O}(LCA \cdot n), \mathcal{O}(LCA \cdot \log n) \rangle$, sparse table даст $\langle \mathcal{O}(LCA \cdot n \log n), \mathcal{O}(LCA) \rangle$.

15. **(*) Леонард**

Храним Эйлеров обход в Treap. $LCA = \min$ высот на отрезке. При переподвешиваем кроме `split/merge` делаем `+=` на отрезке всем высотам.

16. **(*) Dynamic online max на пути**

Умеем LCA $\Rightarrow$ достаточно уметь max на вертикальном пути.
Вертикальный путь $\langle a, b \rangle$ – множество вершин v: $\text{in}[a] \leqslant \text{in}[v] \leqslant \text{in}[b] \wedge \text{out}[a] \geqslant \text{out}[v] \geqslant \text{out}[b]$.
Запрос на прямоугольнике.

17. **(*) Dynamic online Σ на пути**

Достаточно уметь сумму на вертикальном пути.
Эйлеров обход ребер. На ребре вверх пишем w_e, на ребре вниз $-w_e$.
Тогда при запросе на отрезок $[\text{start}_v, \text{start}_u]$ сократятся все ребра, кроме ребер на пути.
Дерево отрезков с суммой и изменением элемента, $\mathcal{O}(\log n)$.

Способ #2.
Путь разбивается по LCA на два вертикальных.
На вертикальном $\texttt{sum}(v, u) = \texttt{sum}(\text{root}, u) - \texttt{sum}(\text{root}, v)$.
Итого нужно уметь считать только сумму на пути до корня, то есть сумму в предках v.
Два дерева отрезков. В T_1 вершины упорядочены по `tin`, в T_2 по `tout`.
Отвечаем $T_1.\texttt{get}(0, \texttt{tin}[v]) - T_2.\texttt{get}(0, \texttt{tout}[v])$.
Не сократятся только вершины u: $\texttt{tin}[u] < \texttt{tin}[v] \wedge \texttt{tout}[v] < \texttt{tout}[u]$, это как раз предки v.

18. **(*) Дуги на окружности**

Для удобства для каждого $[l, r]$ заведем копию $[l + M, r + M]$.
Для отрезка $[l_i, r_i]$ найдем $\texttt{next}_i : l_{\texttt{next}_i} > r_i, r_{\texttt{next}_i} = \min$. Делается за $\mathcal{O}(\text{sort} + n)$.
Ссылки $\texttt{next}_i$ задают лес, на пути вверх строго растет r. Если бы мы решали задачу на прямой, нужно было бы просто пройти от отрезка с min r по ссылкам `next` до упора.

Будем перебирать отрезок $[l, r]$, который точно войдет в ответ. Тогда нужно решать задачу на отрезке от r до $l+M$. Можно сделать на `next` двоичные подъемы. Подняться по ним до самой высокой i: $r_i < l + M$. Получили решение за $\mathcal{O}(\text{sort} + n \log n)$.

Для решения за $\mathcal{O}(\text{sort} + n)$ будем подниматься не двоичными подъемами, а втупую. Но не с нуля каждый раз, а начиная с прыжка на `current_max` вверх. Для этого нужно уметь задачу LA за $\langle \mathcal{O}(n), \mathcal{O}(1) \rangle$. См. также прошлый семестр и хабр.

19. **(*) LCA и подвешивание деревьев**

a) Ленивые двоичные подъёмы. Изначально все они не посчитаны (храним -1).
LCA на двоичных подъёмах. Когда пытаемся подняться и видим -1, пробуем посчитать.
Чтобы посчитать $\texttt{up}[v, 2^k]$, нужно знать два прыжка по 2^{k-1}. Пытаемся сперва посчитать первый, и, только если получилось не -1, считаем второй.

На каждое из $\mathcal{O}((n+m)\log n)$ полезных действий придётся $\leqslant \log n$ бесполезных: попыткок посчитать подъем, вернувших -1, не более одной на каждом уровне.
Успешная попытка посчитать подъем заполнит клетку в таблице двоичных подъемов, которая уже не изменится.
Поэтому амортизированное время одного запроса $\mathcal{O}(\log^2 n)$.
Для нахождения LCA также надо уметь считать глубину вершины. Ее можно считать двоичными подъемами.

b) Используем вариант Эйлерова обхода из сведения LCA $\to$ RMQ.
Будем хранить его в treap по неявному ключу, которое умеет min и += на отрезке.
LCA – вершина минимальной глубины на отрезке.
Подвешивание v к u – вставка отрезка v после u и прибавление на нем $(\text{depth}(u)+1)$.

20. **(*) Экономия предподсчета в ФКБ**

```
for (int mask = 0; mask < (1 << k); mask++)
    auto [i, m] = ans[mask / 2]
    ans[mask] = {i + 1, m + (mask % 2 ? -1 : +1)};
    if (ans[mask].second > 0)
        ans[mask] = {0, 0};
for (int i = n; i >= 1; i--)
    add_bit(mask[i / k], a[i] - a[i-1] == 1 ? 0 : 1) // младший
int get(l, r): // [l, r)
    int il = l / k
    int ir = r / k
    if (il != ir)
        return suf + sparse + pref
    auto [i, m] = ans[(mask[il] >> (l % k)) & ((1 << (r % k - l % k)) - 1)]
    return i + l % k;
```

Домашнее задание

3.1. Обязательная часть

1. **(2) Нельзя не пройти**

   Дан неорграф. Сделайте предподсчёт за линию на полилог, чтобы за $\mathcal{O}(\log n)$ в online отвечать на запрос: сколько вершин нельзя не пройти при путешествии из a_i в b_i?

2. **(3) Помечающиеся вершины**

   Дано дерево. Изначально все вершины не помечены. Online запросы: пометить вершину, найти самого нижнего не помеченного общего предка u и v.

3. **(2) Вспомним прошлое**

   Выбрать из массива длины n подпоследовательность длины ровно L, в которой разность индексов соседних элементов $\leqslant k$, а сумма элементов $\to \min$.

   Полный балл можно получить за время $\mathcal{O}(nL)$.
   Другие полиномиальные решения получат частичный балл.

4. **(3) Размер меняющегося дерева**

   В дерево добавляются листья. Нужно быстро отвечать на запрос «размер поддерева».

   a) **(2)** Online с декартовым деревом.
   b) **(1)** Offline с деревом отрезков.

5. **(2) Сумма на пути дерева**

   Придумайте структуру данных, которая умеет считать сумму весов рёбер на пути в дереве за $\mathcal{O}(LCA) + \mathcal{O}(1)$. Предподсчёт должен работать при $n \leqslant 10^6$.

3.2. Дополнительная часть

В этих задачах *нельзя* пользоваться HLD. Все они решаются без неё `;-)`

1. **(2) Разреженная таблица**

   Дерево отрезков выделяет $\mathcal{O}(n)$ отрезков, и любой отрезок представляется как дизъюнктное объединение $\mathcal{O}(\log n)$ из них. Предложите способ выделить $\mathcal{O}(n \log^* n)$ отрезков в массиве размера n так, что любой отрезок $[L, R]$ можно было представить в виде объединения $\mathcal{O}(1)$ *непересекающихся* выделенных отрезков.

   (+1) Предподсчёт $\mathcal{O}(n\alpha(n))$ и разбиение на $\mathcal{O}(\alpha(n))$ отрезков.

2. **(2) Помечающиеся и разпомечающиеся вершины**

   Дано дерево. Помечаются и разпомечаются вершины.
   В online за $\langle \mathcal{O}(n \log n), \mathcal{O}(\log n) \rangle$ искать самого нижнего не помеченного общего предка.

3. **(3) Σ в поддереве, и + на пути**

   Дано дерево. Запросы: сумма в поддереве, `+=` на пути. $\mathcal{O}(\log n)$.

4. **(3) Дерево namespace'ов.**

   Дано дерево изначально пустых namespace-ов. Все переменные в задаче булевы.
   Значение переменной x в вершине v равно значению в ближайшем предке v, где x определена.

   a) $add(v, x, value)$ – добавить в namespace v запись `x := value`, `value` $\in \{$`true`, `false`$\}$;
   b) $change(v, x, value)$ – аналогично, но поменять значение
   c) $get(x)$ – сказать, в скольких вершинах переменная x равна `true`.

   (1.5) Упрощение задачи:
   Сперва вызываются все запросы вида $add(v, x, value)$, затем вперемешку get и $change$.
   Запросы $get(x)$ теперь интересуются количеством листьев, в которых x равна `true`/`false`.