Второй курс, осенний семестр 2022/23
Практика по алгоритмам #2

HLD, Euler Tour, Link-cut

16 сентября

Собрано 20 сентября 2022 г. в 16:13

Содержание

HLD, Euler Tour, Link-cut

1. **Минимум на пути меняющегося дерева**

   Дано дерево с весами на ребрах, надо эффективно обрабатывать запросы:
   a) $\min(a, b)$ – минимум на пути $a \rightsquigarrow b$.
      $\texttt{set}(e, x)$ – присвоение ребру e веса x.
   b) $\min(a, b)$ – минимум на пути $a \rightsquigarrow b$.
      $\texttt{add}(a, b, x)$ – прибавление числа x к весам всех ребер на пути $a \rightsquigarrow b$.

2. **Ближайший хороший предок**

   Дано дерево с весами в вершинах. Веса из $\{0, 1\}$.
   a) Запросы: сделать вес вершины равным 1, найти ближайшего предка с весом 0.
   b) Запросы: сделать вес вершины равным 1, найти ближайшего предка с весом 1.
   c) Веса из $\mathbb{F}_2$. Запросы: изменить вес вершины, найти ближайшего предка с заданным весом.
   d) Веса из $\mathbb{N}$. Запросы: изменить вес вершины, найти ближайшего предка с заданным весом.
   e) Веса из $\mathbb{N}$. Запросы: изменить вес вершины, по x найти ближайшего предка с весом $\geqslant x$.

3. **Число тяжелых вершин на пути**

   Дерево, веса в вершинах. Запросы $\texttt{count}(a, b, x)$ – число вершин на пути $a \rightsquigarrow b$ с весом $\geqslant x$.
   a) Веса не меняются. $\mathcal{O}(\log n)$.
   b) Теперь веса меняются. Запрос за $\mathcal{O}(\log^3 n)$, изменение веса за $\mathcal{O}(\log^2 n)$.
      А корневая поможет?
   c) `(*)` Веса меняются. Эйлеров обход. $\mathcal{O}(\log^2 n)$.

4. **Dynamic MST**

   Взвешенный неорграф. Online запросы **уменьшения** веса ребра. Поддерживать вес MST.

5. `(*)` **Рандомизированное MST**

   В алгоритме с лекции сделайте первый рекурсивный вызов от βm рёбер $(0 < \beta < 1)$.
   Оцените время работы и сколько шагов Борувки надо делать, чтобы время было линейным.

6. **k-я статистика на пути дерева**

   Дано взвешенное дерево. Online запросы `get(a,b,k)` – k-я статистика на пути $a \rightsquigarrow b$. $\mathcal{O}(\log n)$.

7. **Dynamic Connectivity Offline (Easy)**

   Уже умеем решать за $\mathcal{O}(m\sqrt{m})$. Научимся решать её за $\mathcal{O}(m \log^2 m)$.
   Пригодится разделяй и властвуй или дерево отрезков по запросам.

8. **Link-Cut-Tree**

   Введите правильный потенциал и покажите, что амортизированное время операции Expose в Link-Cut-Tree со Splay деревом на путях – $\mathcal{O}(\log n)$.

9. **Добавление листьев в Heavy-Light-Decomposition**

   Придумайте модификацию HLD, поддерживающую добавлении новых листьев.
   Запросы: изменение веса ребра, минимум на пути.
   a) $\mathcal{O}(\sqrt{n})$ на добавление листа, $\mathcal{O}(\sqrt{n} + \log^2 n)$ на запрос `get`.
   b) Добавление за $\mathcal{O}(\log^2 n)$. *Подсказка: мы хотим быстро* `split`*-ить и* `merge`*-ить пути.*

10. **RMQ в дереве. Веса в вершинах.**

    Научитесь за $\mathcal{O}(1)$ искать минимум весов *вершин* на пути в дереве.

11. `(*)` **LCA дружит с link-cut**

    Граф – всегда лес корневых деревьев, изначально рёбер нет.
    Запросы: $link(a, b)$, $cut(a, b)$, $lca(a, b)$. $\mathcal{O}(\log n)$. Запрос $link$ ориентированный.

12. `(*)` **Link-Cut-Sum**

    Дан лес с положительными целыми весами на ребрах.
    Используя `Euler-Tour-Tree` обрабатывать следующие запросы за $\mathcal{O}(\log n)$:

    a) `link`(v, root, w) – подвесить корень одного дерева к вершине другого ребром веса w;
    b) `cut`(a, b) – удалить ребро;
    c) `sum`(a, b) – сумма весов рёбер на пути.

13. `(*)` **Dynamic Connectivity Offline**

    Уже умеем решать за $\mathcal{O}(m\sqrt{m})$. Научимся решать её за $\mathcal{O}(m \log m)$.
    Сделаем дерево отрезков по запросам.
    Когда спускаемся вниз, какие-то рёбра добавятся, сожмём полученные компоненты связности. Когда поднимаемся вверх, ничего не делаем.

    a) Dynamic Connectivity Offline за $\mathcal{O}(m \log m)$.
    b) Dynamic MST Offline за $\mathcal{O}(m \log m)$.
    c) Dynamic 2-Connectivity Offline за $\mathcal{O}(m \log m)$.

14. `(*)` **Покраска дерева в offline**

    Для взвешенного дерева из n вершин заданы n операций покраски ребер `paint(a,b,x,y)`.
    Операция покрасит все ребра пути $a \rightsquigarrow b$ с весом из $[x, y]$.
    Найдите общее число покрашенных ребер после всех покрасок. $\mathcal{O}(n \log n)$.

15. `(*)` **Число путей, пересекающих данный**

    Дано дерево. В $\forall$ дереве между $\forall$ двумя вершинами $\exists$ единственный простой путь.
    Online запрос `get(x,y)` – число путей в дереве, пересекающихся с путём $x \rightsquigarrow y$.

Разбор задач практики

1. **Минимум на пути меняющегося дерева**

   Строим HLD, для каждого пути HLD храним дерево отрезков на min с операцией `+=`.
   Получили решение за $\langle \mathcal{O}(n), \mathcal{O}(\log^2 n) \rangle$. При этом изменение в точке за $\mathcal{O}(\log n)$, так как это обращение только к одному ДО.

2. **Ближайший хороший предок**

   a) **Сделать вес вершины** 1**, найти ближайшего предка с весом** 0
   Используем СНМ как в предыдущей домашке. Если вершина не подходит, то приклеиваем её к её предку.

   b) **Сделать вес вершины** 1**, найти ближайшего предка с весом** 1
   Способ #1. Offline. Обрабатываем запросы с конца, получаем предыдущий пункт.
   Способ #2. Online. HLD, для каждого пути HLD поддерживаем `set<int>` позиций всех единиц.

   c) **Изменить вес вершины, найти ближайшего предка с заданным весом.**
   Так же, как предыдущий пункт, `set<int>` нулей и `set<int>` единиц, при изменении перемещаем из одного `set` в другой.

   d) **Предыдущий пункт. Веса из** $\mathbb{N}$**.**
   Внутри пути HLD храним `unoredered_map` из значения в сет позиций. В остальном как в предыдущей задаче.

   e) **Веса из** $\mathbb{N}$**, менять вес вершины, находить ближайшего предка с весом** $\geqslant x$**.**
   HLD, для каждого пути HLD поддерживаем ДО на максимум, с его помощью уже умеем находить ближайший слева $\geqslant x$.
   Обновления за $\mathcal{O}(\log n)$, просто изменение одного элемента в нужном ДО.
   `get` за $\mathcal{O}(\log^2 n)$: прыгаем вверх по путям, пытаемся найти в каждом за $\mathcal{O}(\log n)$, пока не найдем.

3. **Число тяжелых вершин на пути**

   a) **Static**
   Разобьем запрос (a, b) на запросы на путях до корня: `+1` в a, `+1` в b, `-2` в $\texttt{lca}(a, b)$.
   Теперь в offline можно решить dfs-ом, обойдя дерево и поддерживая дерево отрезков.
   Делаем $\texttt{add}(x)$ при спуске, $\texttt{del}(x)$ при подъёме, это `+1` и `-1` к позиции x.
   Запрос на пути до корня: $\texttt{count}(\geqslant x)$ (сумма на суффиксе).
   Время обработки всех запросов $\mathcal{O}((n+m)\log n)$.
   Как обычно, в online это переделывается использованием персистентного ДО.
   Итого $\langle \mathcal{O}(n \log n), \mathcal{O}(\log n) \rangle$.

   b) **Dynamic,** $\mathcal{O}(\log^3 n)$
   HLD, для каждого пути HLD храним дерево отрезков Treap-ов.
   Изменение за $\mathcal{O}(\log^2 n)$, удаляем старый вес и добавляем новый в каждый treap в вершинах ДО, покрывающих измененную вершину.
   Запрос за $\mathcal{O}(\log^3 n)$: в $\mathcal{O}(\log n)$ путях HLD выделяем вершины ДО, в каждой вершине делаем `Split` по весу x.

   А корневая поможет? Да. По запросам. $\sqrt{n \log n}$.

c) (*) **Dynamic,** $\mathcal{O}(\log^2 n)$

Перенесем веса с вершин на ребра, соединяющие их с предком.

Эйлеров обход на ребрах, для версии ребра вверх храним пару $\langle w_e, +1\rangle$, для версии вниз пару $\langle w_e, -1\rangle$.

Для каждой вершины v помним $\texttt{pos}[v]$ – время выхода из v (длину Эйлерового обохода в момент выхода из v).

На этом массиве строим ДО treap-ов, которое умеет по отрезку массива посчитать за $\mathcal{O}(\log^2 n)$ сумму плюс-минус единиц на отрезке весов.

При изменении веса ребра, нам нужно поменять два элемента массива. Двумерное дерево обновляется за $\mathcal{O}(\log^2 n)$.

При запросе $\texttt{count}(a, b, x)$ находим $c = \texttt{lca}(a, b)$. $\texttt{count}(a, b) = \texttt{sum}(\texttt{pos}[a], \texttt{pos}[c], x, +\infty) + \texttt{sum}(\texttt{pos}[b], \texttt{pos}[c], x, +\infty)$.

Пока мы идём по Эйлеровому обходу от $\texttt{pos}[a]$ до $\texttt{pos}[c]$, мы пройдём все рёбра пути из a в c вверх. Остальные рёбра, что мы пройдём, посчитаются два раза с разными знаками и сократятся в 0.

Поскольку c выше a и b, то $\max(\texttt{pos}[a], \texttt{pos}[b]) \leqslant \texttt{pos}[c]$.

4. **Dynamic MST**

Link-Cut Tree. Если вес ребра из остова уменьшился, MST не изменилось.

Если вес ребра не из остова уменьшился, нужно посмотреть max на пути (`get`).

Если наше ребро теперь меньше, удалить из MST max ребро цикла (`cut`) и вместо него добавить наше (`link`).

5. (*) **Рандомизированное MST**

Число рёбер во втором рекурсивном вызове будет $(\frac{1}{\beta} - 1)(n' - 1)$.

Здесь $n' = \frac{n}{2^k}$ – число вершин в дереве после k шагов Борувки.

Суммарный размер рекурсивных вызовов $\leqslant (n' + \beta m) + (n' + n' + (\frac{1}{\beta} - 1)n') = (2 + \frac{1}{\beta})n' + \beta m$.

Время работы линейно $\Leftrightarrow (2 + \frac{1}{\beta})n' + \beta m < C \cdot (n + m)$, $0 < C < 1$.

Потребуем $2 + \frac{1}{\beta} < 2^k$, этого достаточно.

6. k**-я статистика на пути дерева**

Насчитаем для каждого пути от корня персистентное ДО. Одновременный спуск по 3 деревьям с одинаковой структурой: `tree[a]`, `tree[b]`, `tree[lca(a,b)]`.

Каждый раз смотрим на `tree[a].data[v] + tree[b].data[v] - 2*tree[lca(a,b)].data[v]`.

7. **Dynamic Connectivity Offline (Easy)**

Дерево отрезков по запросам чтения. Каждое ребро нужно добавить в log вершин.

Обойдём дерево с CHM. Когда входим в вершину добавляем все рёбра в CHM. Когда выходим — откатываем.

8. **Link-Cut-Tree**

Нарисуем все Splay-деревья, корень каждого Splay-дерева подвесим за вершину Splay-дерева, соответствующую отцу начала пути в исходном дереве.

В теорему о времени работы Splay-дерева подставим размеры поддеревьев получившегося мега-дерева.

Более формально, каждой вершине припишем вес, равный суммарному размеру ее поддере-

вьев в исходном дереве, кроме тех, которые лежат в том же Splay.
Время операции Expose не более $\sum 3(\log a_i - \log b_i) + 1 = k + 3\sum(\log a_i - \log b_i) \leqslant k + 3\log n$, так как $b_{i+1} \geqslant a_i$.
Мы знаем, что k компенсируется потенциалом «чичло тяжелых ребер в путях». Итоговый потенциал – сумма потенциалов Splay и числа тяжелых ребер в путях.
При отрезании ребенка из Splay и привешивании ребенка из исходного дерева вес вершины изменился, но размер ее поддерева в Splay не изменился ⇒ не изменился и потенциал.

9. **Добавление листьев в Heavy-Light-Decomposition**

   a) Отложенные операции. Когда добавляем лист, создаём путь из одной вершины. Как только листьев накопилось $\sqrt{n}$, за $\mathcal{O}(n)$ перестраиваем HLD.
   Можно сделать так, чтобы `get` работал за $\mathcal{O}(\log^2 n)$, а не $\mathcal{O}(\sqrt{n} + \log^2 n)$. Для этого при каждом добавлении листа будем перестраивать HLD на кусочке из новых $\sqrt{n}$ вершин.

   b) Пути храним в деревьях по неявному ключу с операциями `split` и `merge`.
   Когда добавили лист, на пути до корня увеличились размеры. Значит, тяжёлые рёбра остались тяжёлыми, а лёгкие могли стать тяжёлыми. Лёгких рёбер на пути не более $\mathcal{O}(\log n)$, перебираем их и, если нужно, делаем `split` старого пути и `merge` нового. Получилось $\mathcal{O}(\log^2 n)$ на добавление листа.
   Надо уметь пересчитывать размеры поддеревьев при добавлени листа. Можно использовать решение из последнего ДЗ, а можно просто делать `+=` на пути от листа до корня, используя уже имеющуюся HLD.
   Кстати, если использовать `splay tree`, то и HLD, и добавление листьев работают за $\mathcal{O}(\log n)$ вместо $\mathcal{O}(\log^2 n)$.

10. **RMQ в дереве. Веса в вершинах.**

    Вершине соответствует ребро в отца. RMQ весов вершин = RMQ весов рёбер на пути + отдельно учесть вес `LCA`.

11. `(*)` **LCA дружит с link-cut**

    ?

12. `(*)` **Link-Cut-Sum**

    Euler Tour Trees. Делать `link` и `cut` умеем.
    Пишем на версии ребра вверх его вес, на версии ребра вниз его вес со знаком минус.
    Поскольку мы подвешиваем только корни деревьев, направления ребер не меняются.
    При запросе `sum` берем сумму от a до `lca` и сумму от b до `lca`. Лишние рёбра посчитаются два раза с разными знаками и сократятся.
    `lca` – вершину с min глубиной на отрезке от a до b.
    При подвешивании и отрезании нужно делать `+=` или `-=` соответствующей глубины на отрезке.

    Другой способ. Поддерживать для каждой вершины v сумму на пути до корня.
    При `link` и `cut` она пересчитываются операциями `+=` или `-=` на отрезке, соответствующем поддереву. `LCA` искать так же.

13. `(*)` **Dynamic Connectivity Offline**

    a) Разбили запросы на половины. Также у нас есть граф с какими-то добавленными ребрами.

Рекурсивный запуск от половин. В запуск от отрезка запросов S также передаем копию графа, где стянем все ребра, которые добавлены до S, но не удаляются в S.
Тогда если у нас отрезок запросов длины 1, и это запрос связности, надо просто проверить, в одной ли стянутой компоненте эти вершины.

b) Dynamic MST Offline за $\mathcal{O}(m \log m)$.
c) Dynamic 2-Connectivity Offline: так же, но стягиваем компоненты двусвязности.

14. `(*)` **Покраска дерева в offline**

Научимся во время `paint(a,b,x,y)` прибавлять единички к нужным ребрам. Тогда в ответе все ребра, у которых счетчик > 0.
Чтобы делать `+1` на $a \rightsquigarrow b$, делаем `+1` на $a \rightsquigarrow$ `root` и $b \rightsquigarrow$ `root`, и `-2` на $\mathrm{lca}(a, b) \rightsquigarrow$ `root`. То есть теперь все запросы от вершины до корня.
`dfs`. Поддерживаем дерево T ребер от корня до текущей вершины. T упорядочено по весам. Обработав вершину, проходим по запросам, у которых она нижняя, и делаем `+=` на нужный отрезок весов в T.
Поднимаясь по ребру, смотрим на его пометку (спускаясь сверху и делая `push`). Если она > 0, это ребро в ответе.

15. `(*)` **Число путей, пересекающих данный**

?

Домашнее задание

3.1. Обязательная часть

1. **(4) Max отрезок без циклов**

   Дан взвешенный неорграф. Найти максимальный по длине отрезок весов $0 \leqslant l \leqslant r \leqslant M$, что множество рёбер с весами из $[l, r]$ не содержит циклов. $\mathcal{O}((m+n)\log n)$.

2. **(4) Статически оптимальный HLD**

   Дано дерево с **фиксированным корнем**.
   Даны запросы на произвольных путях $\langle a_i, b_i \rangle$.
   Нужно построить статически оптимальное покрытие дерева **вертикальными** путями.
   Статически оптимальным называется покрытие путями, минимизируеющее общее число переходов между путями в течение ответов на все запросов.

3. **(3) Завершаем оценку Link-Cut**

   Оцените амортизированное время операций `link` и `cut` относительно потенциала «минус число тяжёлых рёбер, покрытых путями».

4. **(3) Вес под путём**

   Представьте, что на плоскости нарисовано дерево с весами в вершинах. Корень сверху, ветви свисают вниз, все листья на оси X. Любой *путь из листа в лист* делит дерево на две части «внутри» и «снаружи». Ту часть, которая «внутри», будем называть «под путём».

   Вам дано корневое дерево. У каждой вершины известен порядок детей слева-направо. Координат никаких нет, они вам нужны были только чтобы понять задачу.

   Нужно уметь быстро отвечать на запросы двух типов: поменять вес вершины; посчитать сумму весов вершин «под путём из *листа* a в *лист* b».

   Эйлеров обход и HLD — ваши лучшие друзья.

3.2. Дополнительная часть

1. **(4) MST за $\mathcal{O}(n+m)$**

   Посмотрите внимательно на рандомизированный алгоритм построения MST за линейное время и докажите, что в худшем случае он работает за $\min(n^2, m \log n)$.

2. **(4) Link-Cut + Splay**

   Докажите амортизированную сложность $\mathcal{O}(\log n)$ операции MakeRoot в Link-Cut Tree со Splay деревом внутри.

3. **(5) Улучшенный Euler-Tour-Trees**

   Придумайте модификацию Euler-Tour-Trees для хранения леса подвешенных деревьев, поддерживающую операции `Link`, `Cut`, `MakeRoot`, `IsAncestor(a, b)`.

4. **(5) Оптимизируем HLD**

   В HLD изменение веса ребра – быстрая операция.
   А вот запрос на вертикальном пути происходит за $\mathcal{O}(\log n)$ запросов к дереву отрезков.
   Но не к произвольным отрезкам дерева отрезков, а к префиксам, причём каждый раз почти одним и тем же префиксам.
   Давайте в каждом дереве отрезков кешировать ответ: `tree.cache[i] = <T,f>` – в последний раз от префикса длины i мы считали функцию в момент времени T и получили значение f.
   Если T больше времени последнего изменения, вернём f, иначе посчитаем заново.
   Докажите или опровергните гипотезу, что функция на пути теперь вычисляется за $\mathcal{O}(\log n)$.