Второй курс, осенний семестр 2022/23
Практика по алгоритмам #1

Корневая, Splay

9 сентября

Собрано 9 сентября 2022 г. в 08:49

Содержание

Корневая, Splay

1. **Модифицируем `lower_bound` под Splay**

```
Node* lower_bound(Node* v, int x) { // обычная версия
  Node* ans = 0;
  while (v != 0)
    if (v->x < x) v = v->r;
    else ans = v, v = v->l;
  return ans;
}
```

2. **Столько всего прекрасного можно делать на отрезке**
   a) `sum[l..r]` за $\mathcal{O}(\sqrt{n})$, `a[i] = x` за $\mathcal{O}(1)$.
   b) `sum[l..r]` за $\mathcal{O}(1)$, `a[i] = x` за $\mathcal{O}(\sqrt{n})$.
   c) `(*)` Улучшите (a) до $[\mathcal{O}(n^{1/3}), \mathcal{O}(1)]$.
   d) `(*)` Улучшите (a) и (b) до $\mathcal{O}(k \cdot n^{1/k})$ и $\mathcal{O}(k)$.
   e) `sum(l,r)`, `+=(l,r)`, `=(l,r)`, `a[i]`, `insert(i,x)`, `erase(i)`, `reverse(l,r)` за $o(n)$.
   f) То же и `k-th-stat(l,r,k)`. Как выбрать размеры кусков m и частоту вызова `rebuild`?

3. **Странности на отрезках в offline**

   Запросы: для отрезка $[l, r]$ увеличить все элементы меньшие x до x. Для i-го числа массива интересно, когда он станет хотя бы z_i.

   Идея решения: если бы нужно было просто проверить «стал ли элемент в конце $\geqslant z_i$», как решалась бы задача? а теперь пусть запросы приходят пачками по $\sqrt{m}$.

4. **Dynamic connectivity offline**

   Неорграф. В offline 10^5 запросов вида: добавить ребро, удалить ребро, связны ли a и b.

   Есть два решение: dfs-ом и СНМ-ом.

5. **Копирование памяти**

   Дана строка. Нужна структура данных, которая умеет быстро делать операции `memmove(a + i, a + j, sizeof(a[0]) * k)` и `a[i]`.

   a) $\mathcal{O}(\log n)$ на операцию.
   b) $\mathcal{O}(\log n)$ на операцию, $\mathcal{O}(n)$ памяти.
   c) `(*)` Придумайте свой `memmove` за $\mathcal{O}(\log \frac{n}{c})$, где $\frac{n}{c}$ – время работы обычного `memmove(n)`.

6. **`split` и `merge` для skip-list**
   Придумайте операции `split` и `merge` для skip-list за $\mathcal{O}(\log n)$.

7. **Splay иногда работает за линию**
   Приведите последовательность n добавлений в пустое дерево, дающую дерево высоты $\Theta(n)$.

8. **Splay: zig-ов недостаточно**
   Приведите n запросов, работающих за $\Omega(n^2)$, если делать только zig (поворот вокруг отца).

9. **Тёплые воспоминания**

   Вспомним задачу с прошлогодней практики. Даны отрезки на прямой. Запросы `get(x,k)` «какой максимальный вес отрезка с длиной не более k, покрывающего точку x?»

10. **Точки в многоугольнике**
    Дан невыпуклый многоугольник из n вершин.
    a) Даны m точек. За $\mathcal{O}((m+n)\log n)$ для каждой точки определить, внутри она или снаружи.
    b) Теперь отвечать в online за $\mathcal{O}(\log n)$. Можно предподсчёт за $\mathcal{O}(n \log n)$.

11. **Веса вершин в Splay**

    Пусть w_i, вес вершины с номером i, равен $\frac{1}{i^2}$. Какой вывод мы можем сделать про время работы splay-дерева?

    Потенциал получится отрицательным. Как это скажется на времени работы?

12. `(*)` **SQRT: strings**

    Задача: разбить текст на словарные слова.
    Медленное решение: динамика за $\mathcal{O}(|text| \cdot |dictionary|)$. Ускорить.

13. `(*)` **SQRT: graphs**

    Дан граф. Запросы: покрасить вершину v в цвет c; узнать число разных цветов у соседей v.

14. `(*)` **Static Finger Theorem**

    Пусть splay-дерево хранит ровно n ключей $\{1, 2, \ldots, n\}$. Пусть мы знаем, что будут выполнены m запросов `find` к ключам $a_1, a_2, \ldots, a_m$. Пусть t – любой выделенный ключ («finger»).

    Докажите, что Splay работает за $\mathcal{O}(m+n\log n+\sum_i \log(|a_i-t|+1))$. Вам пригодится потенциал из предыдущей задачи.

15. `(*)` **Splay-tree rocks!**

    Бор – дерево для хранения строк. У вершин есть ссылки на детей: `map<char,Node>`.
    Если `map` реализован как сбалансированное дерево, спуск по пути p работает за $\mathcal{O}(|p| \log |\Sigma|)$.
    Задача: докажите, что, используя splay-tree, мы получим $\mathcal{O}(|p| + \log |\sum s_i|)$.

16. `(**)` **Быстрая реализация Splay**
    a) Придумайте реализацию без поддержки отцов.
    b) Одним проходом сверху вниз (без обратного хода рекурсии, вообще без рекурсии). *Это техническая задача. Важна не столь идея, сколько конкретные действия.*

Разбор задач практики

1. **LowerBound**

   В конце нужно вызвать Splay от последнего ненулевого значения v.

2. **Столько всего прекрасного можно делать на отрезке**

   a) `sum[l..r]` за $\mathcal{O}(\sqrt{n})$, `a[i] = x` за $\mathcal{O}(1)$.
   Куски длины $\sqrt{n}$. Для каждого куска храним сумму.

   b) `sum[l..r]` за $\mathcal{O}(1)$, `a[i] = x` за $\mathcal{O}(\sqrt{n})$.
   Куски длины $\sqrt{n}$. Для каждого куска массив префиксных сумм.
   Обозначим sum_i сумму на i-м куске, храним префиксные суммы массива sum.

   c) `(*)` За $[\mathcal{O}(n^{1/3}), \mathcal{O}(1)]$.

   d) `(*)` За $\mathcal{O}(kn^{1/k})$ и $\mathcal{O}(k)$.
   Дерево, ветвящееся на $n^{1/k}$.
   На каждом уровне только две вершины ветвятся, на каждом уровне смотрим $\mathcal{O}(n^{1/k})$ вершин.
   Изменение за высоту, равную k.

   e) `sum(l,r)`, `+=(l,r)`, `=(l,r)`, `get(i)`, `insert(i, x)`, `erase(i)`, `reverse(l,r)`.
   `split + rebuild`.
   На исходном массиве считаем префиксные суммы.
   Для каждого отрезка храним отложенные операции `+=`, `=`, `reverse`:
   `struct Segment { int l, r, add, x; bool isReversed; }`.
   Все операции за число кусков, внутри кусков за $\mathcal{O}(1)$ (в том числе `reverse` – он влияет только на то, какую часть куска отделить).
   Раз в $\mathcal{O}(\sqrt{n})$ операций `rebuild` за $\mathcal{O}(n)$.

   f) `ith-stat(l, r, i)`.
   Для каждого отрезка храним отсортированную копию.
   Теперь `split` можно делать за $\mathcal{O}(m \log m)$, где m – длина отрезка.
   Но если для каждого элемента в сортированной версии помнить его индекс, то можно за $\mathcal{O}(m)$ разделить сортированный кусок на два сортированных куска.
   Поэтому изначально разобьем массив на k кусков длины $m = \frac{n}{k}$, чтобы куски были короткими.
   Если `rebuild` раз в $t = \Omega(k)$ операций, то кусков всегда $\leqslant k + 2t = \mathcal{O}(t)$.
   Так что старые операции за $\mathcal{O}(t + m)$.
   `ith-stat(l, r, i)` бинпоиском по ответу. $\mathcal{O}(t + m + t \log m \log C) = \mathcal{O}(m + t \log n \log C)$.
   Итого с учетом амортизированной стоимости `rebuild` время $\mathcal{O}(m + t \log n \log C + \frac{n \log n}{t})$.
   Надо подобрать параметры так, чтобы все слагаемые уравнялись.
   Подбираем t: $m = \frac{n \log n}{t} \Rightarrow t = \frac{n \log n}{m} = k \log n$.
   Подбираем k: $m = \frac{n}{k} = k \log^2 n \log C \Rightarrow k = \frac{\sqrt{n}}{\log n \sqrt{\log C}}$.
   Итого операция в среднем стоит $\mathcal{O}(\sqrt{n} \log n \sqrt{\log C})$.

3. **Странности на отрезке**

   Чтобы обработать все запросы «увеличения до x» можно пройти слева направо с событиями «отрезок начался/закончился» и `set`-ом x-ов открытых. Решение корневой: делаем так для

каждого блока длины $\sqrt{m}$. Если элемент впервые превысил его z_i, для него за $\sqrt{m}$ перебираем все запросы блока, выбираем, после какого это случилось.

4. **Dynamic connectivity**

Рассмотрим моменты времени $[i, i+k)$.
Некоторые рёбра в моменты времени $[i, i+k)$ всегда есть в графе. Некоторых всегда нет.
И только $\leqslant k$ рёбер добавляются и удаляются.

Решение dfs-ом. Возьмём рёбра, которые в моменты $[i, i+k)$ всегда есть в графе, сожмем компоненты связности `dfs`-ом за $\mathcal{O}(i+k)$. Ответ на запрос $t \in [i, i+k)$. Смотрим, какие из $\leqslant k$ рёбер есть в момент t. Запускаем на них `dfs`. Итого $\mathcal{O}(k)$.
Каждый запрос за $\mathcal{O}(k)$. Перед каждым блоком размера k предподсчет за $\mathcal{O}(m)$.
В сумме $\mathcal{O}(km + m \cdot \frac{m}{k}) \Rightarrow k = \sqrt{m}$, решили задачу за $\mathcal{O}(m\sqrt{m})$.

Решение СНМ-ом. Возьмём рёбра, которые в моменты $[i, i+k)$ всегда есть в графе, добавим их в СНМ. Ответ на запрос $t \in [i, i+k)$. Смотрим, какие из $\leqslant k$ рёбер есть в момент t. Добавляем их в СНМ, отвечаем на запрос, откатываем СНМ.

5. **Копирование памяти**

a) Используем персистентное дерево со `Split` и `Merge`.
Пусть нужно $[l_1, r_1)$ скопировать в $[l_2, r_2)$.
Разделим дерево на $[0, l_1)[l_1, r_1)[r_1, n) = A_1 B_1 C_1$.
Разделим то же дерево на $[0, l_2)[l_2, r_2)[r_2, n) = A_2 B_2 C_2$.
Склеим $A_2 B_1 C_2$, получим требуемое.

b) В каждый момент полезных вершин только n.
Но мы порождаем $6h$ новых каждой операцией: 4 `Split` + 2 `Merge`.
При большом числе операций это проблема. Надо собирать мусор.
Способ #1. Счетчик ссылок у каждой вершины. Когда обнуляется, удаляем.
Способ #2. Раз в M операций строить новое дерево по текущему массиву, остальное удалять.

c) `(*)` **Максимально крутой аналог memmove**
Идея: персистентное 2-3-Tree со split и merge. Ключ – подстрока исходной длины от 128 до 256. Технически сложная часть: модифицировать split и merge, следить за длинами ключей.

6. **Split и Merge для skip-list**

У нас есть $\log n$ списков со ссылками вниз. Храним ссылки на начала списков.

split(t, x). Стартуем в верхнем списке, самом разреженном.
Идем по списку, пока не видим ребро в $> x$. Разрезаем это ребро, переходим вниз в более плотный список.
`split(t, x)` проделает тот же путь, что и `find(t, x)`, время работы такое же.

merge(l, r). Стартуем в верхнем списке l.
Доходим до конца списка, приклеиваем к нему соответствующий список r, переходим вниз.
Время работы такое же, как при выполнении `find(l,`∞`)`.

7. **Splay иногда работает за линию**

```
1 for (int i = 1; i <= n; i++)
2 add(i);
```

8. **Splay: zig-ов недостаточно**

   Отличие будет в случае zig-zig: в правильном Splay сначала повернем деда, затем отца.
   Если сделать zig-zig от бамбука, то его глубина уменьшится вдвое.
   Если же поднимать низ бамбука только zig-ами, то глубина уменьшится на 1.
   Строим бамбук, как в прошлом пункте. Последовательно запрашиваем $1, 2, \ldots, n$.

9. **Тёплые воспоминания**

   Пусть Offline.

   Решение #1 за $\mathcal{O}(m \log n)$. Идём слева-направо, обрабатываем события отрезок начался-закончился, точка. Отвечая на запрос «точка», ищем максимум по всем отрезкам подходящей длины (максимум на отрезке).

   Решение #2 за $\mathcal{O}(m \log n)$. Перебираем отрезки в порядке возрастания длины. Делаем `max=` на отрезке одномерным деревом отрезков. Точка: вопрос к Д.О.

   Делаем `max=` на отрезке одномерным деревом отрезков. Точка: вопрос к Д.О.

   Пусть Online. В любом из приведённых решений сделаем все деревья персистентными, когда приходит точка-запрос, выбираем бинпоиском нужное дерево, спускаемся.

10. **Точки в многоугольнике**

    Подробно почитать можно здесь.

    a) Сканирующая прямая по увеличению x.
       Храним в BST прямые, на которых лежат пересеченные сканирующей прямой стороны многоугольника.
       Встречая вершину, кладем стороны, которые идут вправо, вынимаем те, которые влево.
       Встречая запрос, считаем, сколько сторон выше него. Если четное число, то снаружи, иначе внутри. Отдельно обработать, если запрос лежит на стороне.
       Нужно уметь сравнивать только точки и прямые, прямые друг с другом незачем.
       Аккуратно надо с вертикальными отрезками. Если при равном x сортировать по y, то можно обработать попадание запросов на такие стороны без обращения к дереву.
       Еще можно избавиться от вертикальных отрезков, повернув на случайный угол. Или на половину минимального ненулевого угла, под которым отклоняются стороны от оси y.
    b) Стандартный переход от offline к online в задачах на сканирующую прямую. Делаем персистентое BST, по версии для каждого положения сканирующей прямой. На запрос находим бинпоиском нужную версию и обращаемся к ней.

11. **Веса вершин в Splay**

    Потенциал вершины – логарифм суммы весов в поддереве. Потенциал корня – всегда $\log \sum \frac{1}{i^2} = \mathcal{O}(1)$. Время работы запроса – разность потенциалов, логарифм – монотонная функция $\Rightarrow time_i \leqslant 1 + 3(\log \sum \frac{1}{i^2} - \log \frac{1}{i^2}) = \mathcal{O}(1) + 3 \cdot 2 \log i$.

    Реальное время работы отличается от амортизированного на разность потенциалов.
    Потенциал может быть отрицательным: $\varphi = \sum \log \frac{1}{i^2} \geqslant -2n \log n \Rightarrow$
    время работы $\mathcal{O}(m) + 2n \log n + \sum_i 6 \log i$.

12. **(*) SQRT: strings**

    Различных длин строк не более $\sqrt{2 \sum |word_i|} \Rightarrow$ и переходов в динамике столько.
    Переходы можно за $\mathcal{O}(1)$ делать хешами.

13. (*) **SQRT: graphs**

Есть жирные вершины степени $\geqslant \sqrt{2E}$, таких не более $\sqrt{2E}$.
Для нежирной вершины ответить можно за $\mathcal{O}(deg)$, для жирной будем поддерживать уже готовый ответ и отвечать за $\mathcal{O}(1)$.

14. (*) **Static Finger Theorem**

Пусть вес вершины с ключом i: $w_i = (|i - t| + 1)^{-2}$.

Подставим веса в теорему, получим требуемое время. Важгл не забыть, что потенциал может быть отрицательным, при этом $\varphi \geqslant -Cn \log(n)$.

15. (*) **Splay-tree rocks!**

Пусть в боре из v идёт ребро в u по символу c. Вершинам v и u соответствуют splay-деревья t_v и t_u, пустим ребро из вершины t_v, где хранился c в корень t_u. Итого у нас бор превратился в одно большое splay дерево T.

Используем теорему о времени splay. Подставим в неё веса w_v – размер поддерева v в T.
Получаем, что время спуска по бору $X = \sum_i \big(3(R(\text{root}_i) - R(\text{leaf}_i)) + 1\big)$. root_{i+1} – ребёнок $\text{leaf}_i \Rightarrow R(\text{leaf}_i) \geqslant R(\text{root}_{i+1}) \Rightarrow X \leqslant 3(R(\text{root}_1) - R(\text{leaf}_k)) + \sum_i 1 = \mathcal{O}(|p| + \log|T|)$.

16. (*) **Быстрая реализация Splay**

Гуглится по «splay top-down implementation».
Реализация на чистом Си.
Статья Тарьяна 1985-го года (стр. 668 или 17/35) сразу включала описание этого варианта.

Если мы хотим только избавиться от отцов, то надо просто делать два шага по дереву и рекурсивный вызов от внука.
Тогда остается сделать zig-zig или zig-zag.
Если уже в сыне искомый ключ, то внук не нужен, просто zig.
Заметим разницу: раньше при нечетной длине пути zig был вверху, теперь он внизу.
Это ничего не портит. Важно только то, что zig всего один $\Rightarrow$ всего одна `+1` в амортизированной стоимости.

Теперь выкинем рекурсивный вызов.
Рассмотрим, например, zig-zig пути влево $z \leftarrow y \leftarrow x$. Что мешает нам сделать повороты до рекурсивного вызова? Мы не знаем, какой был бы правый ребенок z после рекурсии.
Но мы знаем, что потом y станет правым сыном z, а правый сын z левым сыном y.
Так что мы можем накапливать будущих левого и правого детей, спускаясь вниз.
См. картинку в статье Тарьяна.

Домашнее задание

При выполнении *обязательных* задания важно помнить, что мы знаем деревья поиска, корневую по массиву, корневую через split и rebuild, метод отложенных операций. Двумерные деревья мы не знаем, знания двухмерных деревьев можно будет применить через пару недель.

3.1. Обязательная часть

1. **(2) `split` и `merge` для splay-tree**

   Придумайте операции `split` и `merge` для splay tree. Операции должны работать за $\mathcal{O}(\log n)$.

2. **(2) Инверсии перестановки**

   Дан массив инверсий перестановки: $k_i = \#\{j \mid j < i \wedge a_j > a_i\}$.
   Восстановите перестановку за $\mathcal{O}(n \log n)$.

   Подсказка: какой элемент поставить на последнее место?

3. **(3) Уровень жизни растёт!**

   Придумайте структуру данных, умеющую считать сумму на отрезке и обрабатывать запрос `inc(l,r,x)` – увеличить все значения отрезка $[l, r]$, меньшие x, до x ($n, m \leqslant 10^5$).

4. **(3) MST и равновесие**

   Постройте за $o(E^2)$ остовное дереве, в котором $(\max w_e - \min w_e)$ минимально.
   (+1) балл можно получить, если для $E \leqslant 100\,000$ ваше решение делает $\leqslant 10^8$ операций.

5. **(3) 2D-мир**

   Даны точки на плоскости. Запросы:

   a) Удалить все точки в прямоугольнике $[x_1, x_2] \times [y_1, y_2]$.
   b) Добавить новую точку.
   c) Сказать, сколько точек в прямоугольнике $[x_1, x_2] \times [y_1, y_2]$?

3.2. Дополнительная часть

1. **(3) Шары и урны**

   Рассмотрим n различных шаров и n различных урн, стоящих в ряд. Изначально каждая урна содержит ровно один из шаров. С урнами производят m операций вида `move(i,j,k)` – поднять все шары из отрезка урн $[i, i+k)$, и опустить все эти шары в таком же порядке в отрезок урн $[j, j+k)$. Отрезки могут пересекаться. По данной последовательности операций выясните для каждого шара, в какой урне он будет находиться после всех перемещений.

   Подсказка: задача не про корневую.

2. **(4) Запросы в полуплоскости**

   Даны n точек на плоскости. В online приходят m произвольных запросов-полуплоскостей $ax + by + c \geqslant 0$. Нужно отвечать на запрос «сколько точек в полуплоскости».

   (2) Научитесь делать предподсчёт для $n = 1000$, чтобы отвечать на запрос за $\mathcal{O}(\log n)$.

   (2) Решите задачу для ограничений $n = 10^5$, $m = 10^5$.

3. **(3) Ретроактивность**

   Представьте себе последовательность запросов к структуре данных $q_1, q_2, \ldots, q_m$. Среди q_i есть как запросы изменения, так и get-запросы. Структура данных называется *полностью ретроактивной*, если можно делать операции `q.insert(i, new query)`, `q.erase(i)` и узнавать результат выполнения `q[i]`. То есть, мы можем менять историю запросов в произвольном месте, и хотим, чтобы эти изменения становились видны во всех последующих запросах.

   Задача: придумайте полностью ретроактивную очередь (`push/pop/front`). Операция `pop` из пустой очереди ничего не делает. Операция `front` из пустой очереди достаёт `IND`.

4. **(4) Ретроактивный deque**

   Придумайте ретроактивный deque с операциями за $\mathcal{O}(poly(\log n))$.