1 Практика 1. AVL деревья

1.1 Практика

1. Пусть вам даны два AVL-дерева T_1 и T_2. Придумайте, как построить AVL-дерево T, являющееся объединением деревьев T_1 и T_2 за время:
   (a) $\mathcal{O}(size(T_1) + size(T_2))$
   (b) $\mathcal{O}(height(T_1) \times size(T_2))$
2. Дано двоичное дерево, в котором AVL-свойство выполнено везде, кроме корня.
   (a) Пусть `root.l.h = root.r.h + 3`. Как восстановить AVL-свойство всюду за $\mathcal{O}(1)$?
   (b) Пусть `root.l.h = root.r.h + k`. Как восстановить AVL-свойство всюду за $\mathcal{O}(k)$?
3. Придумайте `merge` для AVL-деревьев за $\mathcal{O}(\log n)$.
4. Придумайте `split` для AVL-деревьев за $\mathcal{O}(\log^2 n)$.
5. Докажите, что придуманный `split` на самом деле работает за $\mathcal{O}(\log n)$.
6. Пусть дано AVL-дерево T, ключ k и число n. Придумайте алгоритм, который выведет n элементов, следующих за k, за время $\mathcal{O}(h(T) + n)$.
7. Придумайте структуру данных, позволяющую online за $\mathcal{O}(\log n)$ отвечать на следующие запросы:
   - добавить элемент x
   - удалить элемент x
   - найти i-ый по порядку элемент
   - найти порядок элемента x
8. Запросы online за $\mathcal{O}(\log n)$:
   - добавить пару $\langle x, y \rangle$
   - удалить пару $\langle x, y \rangle$
   - посчитать сумму y по всем парам таким, что $l \le x \le r$
9. Запросы online за $\mathcal{O}(\log n)$:
   - добавить пару $\langle x, y \rangle$
   - посчитать сумму y по всем парам таким, что $l \le x \le r$
   - посчитать сумму x по всем парам таким, что $l \le y \le r$
10. Запросы online за $\mathcal{O}(\log n)$:
    - `add(i, x)`
    - `del(i)`
    - `add(l, r, value)` — добавить *value* ко всем x, для которых $l \le i \le r$
    - `sum(l, r)` — сумма всех x, для которых $l \le i \le r$
11. Уменьшите количество дополнительной информации в AVL-дереве до двух бит на вершину.
12. Покажите, что любые два корректные дерева поиска, построенные на одном и том же множестве ключей, можно получить друг из друга последовательностью поворотов.

1.2 Домашнее задание

1. Вершина дерева является *единственным ребенком*, если у ее родителя только один ребенок (корень единственным ребенком не является). Определим для дерева степень одиночества LR таким образом:

   $LR(T) = (\text{ Количество единственных детей в } T)/(\text{ Количество вершин в } T)$.

   (a) Покажите, что в любом непустом AVL-дереве T $LR(T) \leq \frac{1}{2}$.

   (b) Правда ли, что если для бинарного дерева T верно $LR(T) \leq \frac{1}{2}$, то $h(T) \leq \log size(T)$?

   (c) Пусть семейство бинарных деревьев $\{T_i\}_{i \in \mathbb{N}}$ и число $0 < \varepsilon < \frac{1}{10}$ таковы, что $size(T_i)$ — строго возрастающая функция от i, $\forall i$ все единственные дети T_i — листья, $\lim\limits_{i \to \infty} LR(T_i) \geq \varepsilon$. Обязательно ли $h(T_i) = \mathcal{O}(\log size(T_i))$?

2. Покажите, что:

   (a) Добавление в AVL-дерево требует $\mathcal{O}(1)$ вращений.

   (b) Удаление из AVL-дерева может потребовать $\Omega(\log n)$ вращений.

3. Пусть в обычное несбалансированное бинарное дерево поиска добавляются различные элементы в порядке $x_1, x_2, \ldots, x_n$. Придумайте алгоритм, поддерживающий для этого дерева массив из отцов всех его вершин и реализующий добавление элемента в дерево вместе с обновлением массива отцов за $\mathcal{O}(\log n)$.

   Пояснение: рассмотрим несбалансированное дерево поиска T. На первом шаге оно состоит из одного элемента x_1. На i-ом шаге в корректное согласно порядку элементов место дерева добавляется элемент x_i, при этом балансировка (подобная АВЛ-дереву или любая другая) не происходит, следовательно, высота дерева может не быть логарифмической. При этом в "массив отцов" на позицию i записывается номер вершины, к которой был подвешен элемент x_i. От алгоритма не требуется явно хранить в памяти дерево T, но требуется за $\mathcal{O}(\log n)$ пересчитывать массив отцов с каждым добавлением нового элемента.

4. Придумайте, как реализовать структуру данных, поддерживающую следующие операции на последовательности из n чисел:

   - Обмен местами последовательных пар соседних чисел на заданном отрезке четной длины (пример: $[1, 2, 3, 4, 5, 6], (2, 5) \to [1, 3, 2, 5, 4, 6]$),
   - Вывод числа на заданной позиции.

   Время работы — $\mathcal{O}(\log n)$ на запрос.

5. Придумайте структуру данных на основе AVL дерева, позволяющую online за $\mathcal{O}(\log n)$ отвечать на следующие запросы:

   - `add(i, x)`
   - `del(i)`
   - `add(l, r, value)` — добавить по модулю 5 *value* ко всем x, для которых $l \leq i \leq r$
   - `sum(l, r)` — сумма всех x, для которых $l \leq i \leq r$

Дополнительные задачи

6. (кроме группы Слободкина) Напишите на каком-нибудь строго типизированном функциональном языке реализацию добавления в AVL дерево так, чтобы AVL-инвариант гарантировался типизацией (вероятно, для этого потребуется язык с зависимыми типами).

7. Постройте тест, на котором недо-AVL-дерево, которое делает только малые вращения, делает $\Theta(n^2)$ операций после n запросов.

   Формально: изначально дерево пусто, нужно n раз вызвать `add(root,` x_i`)` для некоторой последовательности x_i, что суммарное время работы $\Theta(n^2)$.

   Либо докажите, что такого теста нет.

2 Практика 2. Splay-деревья

2.1 Практика

1. Уменьшите количество дополнительной информации в AVL-дереве до двух бит на вершину.

2. Покажите, что любые два корректные дерева поиска, построенные на одном и том же множестве ключей, можно получить друг из друга последовательностью поворотов.

3. Задача "вставка ключа": изначально все ячейки пусты, требуется online за $\mathcal{O}(\log n)$ обрабатывать запросы: 1) `Insert(i, x)` — вставить x в ячейку i. Если i-я ячейка занята, все ячейки, начиная с i-й, сдвигаются вправо; 2) Узнать элемент на позиции i
Пример: $(5,1);(5,2);(5,3);(1,7);(1,8);(2,9) \to 8,9,7,0,3,2,1$.

4. Придумайте структуру данных, позволяющую online за $\mathcal{O}(\log n)$ отвечать на следующие запросы:

   - добавить элемент x на позицию i
   - удалить элемент с позиции i
   - $(l, r, k, right/left)$ — переместить подотрезок на k позиций влево или вправо. Гарантируется, что подотрезок не вылезет за границы массива.

5. Дано дерево, на вершинах которого могут быть пометки. Запросы: пометить вершину, снять пометку с вершины, число помеченных вершин в поддереве. Предобработка за $\mathcal{O}(n)$, online-запросы за $\mathcal{O}(\log n)$.

6. Даны подвешенное дерево и его Эйлеров обход. Придумайте, как за $\mathcal{O}(\log n)$ обновить Эйлеров обход при переподвешивании дерева за другую вершину (для разных вариантов обхода).

7. Дано дерево с весами на ребрах, нужно online обрабатывать запросы:

   - Изменить вес ребра e.
   - Вывести вес простого пути между данной вершиной и корнем.

   $\mathcal{O}(n)$ на предобработку, $\mathcal{O}(\log n)$ на запрос.

8. Дан лес подвешенных деревьев. Нужно отвечать на следующие запросы:

   - Подвесить дерево с корнем v к вершине u другого дерева.
   - Отрезать поддерево с корнем в вершине v от ее дерева.
   - Проверить, в одном ли дереве лежат вершины u и v.

   Время: $\mathcal{O}(\log n)$.

2.2 Домашнее задание

1. Придумайте структуру данных, поддерживающую упорядоченный список S целых чисел, которая умеет отвечать на запросы:

   - `insert(x)` — вставить x в S, если его там не было.
   - `delete(x)` — удалить x из S, если он там был.
   - `S[k]` — вернуть k-тый по порядку элемент из S.
   - `max(l, r)` — найти $\max_{l \le j < k \le r} |S[j] - S[k]|$. Гарантируется $r - l \ge 1$.
   - `min(l, r)` — найти $\min_{l \le j < k \le r} |S[j] - S[k]|$. Гарантируется $r - l \ge 1$.

   Каждый запрос должен обрабатываться за $\mathcal{O}(\log |S|)$.

2. Пусть есть произвольное `splay`-дерево, построенное на ключах $1, 2, 3 \dots n$. Покажите, что в результате последовательных вызовов `splay(1)`, `splay(2)`, … `splay(n)` получится бамбук.

3. Дан взвешенный неорграф. Найдите максимальный по длине отрезок $[l, r]$ весов ($W_{\min} \le l \le r \le W_{\max}$) такой, что множество рёбер с весами из (l, r) не содержит циклов. $\mathcal{O}((E + V) \log V)$.

4. Даны число K и изначально пустая последовательность. Вам поступает n запросов, каждый одного из двух типов:

   - дописать элемент x в конец последовательности
   - развернуть K последних элементов последовательности (если в данный момент их всего меньше K, то развернуть всю последовательность).

   Вам нужно один раз после всех запросов вывести получившуюся последовательность. Число K общее для всех запросов, x — свой для каждого запроса. Все x занимают $\mathcal{O}(1)$ памяти.

   (a) $\mathcal{O}(n \log n)$

   (b) $\mathcal{O}(n)$

5. Попробуем построить сливаемую кучу по минимуму на основе дерева следующим образом: пусть каждая вершина дерева хранит один ключ и односвязный список ссылок на дочерние поддеревья. На ключах выполняется свойство кучи. Операции будем выполнять следующим образом:

   - *create-heap* — создать пустое дерево
   - *find-min* — вернуть ключ, хранящийся в корне дерева
   - *merge* — добавить ссылку на дерево с большим (по ключу) корнем в начало списка детей корня дерева с меньшим (если оба дерева не пустые)
   - *insert* — *merge* с деревом из одного элемента
   - *delete-min* — удалить корень и слить его детей таким образом: идем по списку слева направо, сливая детей попарно (1-2, 3-4, 5-6 и т. д.). Потом идем справа налево и сливаем все деревья в одно. Например, из списка детей $[c_1, c_2, c_3, c_4, c_5]$ получится дерево:

   $$merge(merge(c_1, c_2), merge(merge(c_3, c_4), c_5))$$

   Докажите, что любая корректная последовательность из n таких операций работает за $\mathcal{O}(n \log n)$ (в начальный момент куч нет). Подсказка: это должно быть как-то связано со `splay`-деревьями.

Дополнительные задачи

6. `C-c-c-combo!`
   Напишите псевдокод для структуры, умеющей отвечать на запросы:

   - `add(i, x)`
   - `del(i)`
   - `add(l, r, value)` — добавить *value* ко всем x, для которых $l \leq i \leq r$
   - `set(l, r, value)` — установить в *value* все x, для которых $l \leq i \leq r$
   - `sum(l, r)` — сумма всех x, для которых $l \leq i \leq r$
   - `reverse(l, r)` — изменить порядок всех x, для которых $l \leq i \leq r$, на обратный

   Время работы всех запросов $\mathcal{O}(\log n)$, online.

3 Практика 3. Задачи RMQ и LCA

3.1 Практика

1. Вам дано дерево из одной вершины. Придумайте, как online отвечать на следующие запросы:
   - Подвесить новую вершину к дереву — $\mathcal{O}(1)$.
   - `LCA` двух вершин — $\mathcal{O}(h)$ времени и $\mathcal{O}(1)$ дополнительной памяти, где h — высота дерева на данный момент.
2. Решите предыдущую задачу за $\mathcal{O}(\log n)$ используя $\mathcal{O}(\log n)$ памяти на вершину (n — число добавлений; для простоты считайте, что n известно заранее).
3. Вам дано подвешенное дерево из n вершин. Нужно отвечать на запросы:
   - Найти `LCA` двух вершин,
   - Переподвесить дерево за новую вершину.

   Время обработки запроса: $\mathcal{O}(\log n)$. Предподсчет линейный.
4. Придумайте, как online отвечать на запросы `RMQ` за

   a) за $\mathcal{O}(1)$ с предподсчетом $\mathcal{O}(n \log(\log n))$.

   b) за $\mathcal{O}(\log^* n)$ с предподсчетом $\mathcal{O}(n \log^* n)$.
5. Есть массив из нулей и единиц. Запросы: поменять элемент; найти ближайший слева/справа ноль к позиции i. Online за $\mathcal{O}(\log n)$ с линейным предподсчетом.
6. Дана таблица $n \times n$ с целыми числами. Поступают запросы:
   - Изменить значение в ячейке (i, j),
   - Посчитать сумму чисел в прямоугольнике $((l, t), (r, b))$.

   Обработать онлайн. $\mathcal{O}(\log^2 n)$ на запрос, $\mathcal{O}(n^2 \log n)$ на предподсчет.
7. Дано подвешенное дерево из n вершин с весами на ребрах, нужно научиться online отвечать на запросы суммы весов на пути между парой вершин. $\mathcal{O}(n)$ на предподсчет, $\mathcal{O}(1)$ на запрос.
8. k-инверсией в перестановке p называется набор индексов $i_1 < i_2 < \ldots < i_k$, такой, что $p[i_1] > p[i_2] > \ldots > p[i_k]$. Найти число k-инверсий за $\mathcal{O}(nk \log n)$.

Дополнительные задачи

9. Дан массив из n элементов, можно сделать предобработку за $\mathcal{O}(n \log n)$. Запросы: количество различных чисел на отрезке $[L, R]$. *Тут будет подсказка про* $prev[i]$.
   (a) online за $\mathcal{O}(\log^2 n)$.
   (b) offline за $\mathcal{O}(\log n)$.
10. Запросы: k-е по порядку среди различных чисел на отрезке $[L, R]$.
    (a) offline за $\mathcal{O}(\log^3 n)$.
    (b) online за $\mathcal{O}(\log^3 n)$.

3.2 Домашнее задание

1. Дано дерево из одной вершины. Требуется уметь отвечать online за $\mathcal{O}(\log n)$ на запрос: подвесить новую вершину u к вершине дерева v и вернуть диаметр дерева. Диаметр дерева — длина самого длинного простого пути в дереве.

2. Дан ориентированный граф, в котором исходящая степень каждой вершины равна единице. Запросы online: из вершины v сделать k шагов вперед.

   (a) Предподсчет: $\mathcal{O}(n \log k_{\max})$, время на запрос: $\mathcal{O}(\log k)$.

   (b) Предподсчет: $\mathcal{O}(n \log n)$, время на запрос: $\mathcal{O}(\log min(k, n))$.

3. Дана скобочная последовательность из круглых скобок длины n. Запросы: является ли отрезок $[L, R]$ правильной скобочной последовательностью; изменить i-ю скобку. $\mathcal{O}(\log n)$, online.

4. Попробуем модифицировать идею `SparseTable` так, чтобы она работала для произвольных ассоциативных функций: 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. Заметим, что дерево отрезков выделяет $\mathcal{O}(n)$ отрезков, и любой отрезок представляется как объединение $\mathcal{O}(\log n)$ из них.

5. (только для группы Слободкина) Дано подвешенное дерево из n вершин с весами на ребрах, нужно научиться online отвечать на запросы суммы весов на пути между парой вершин. $\mathcal{O}(n \log(n))$ на предподсчет, $\mathcal{O}(1)$ на запрос.

6. k-инверсией в перестановке p называется набор индексов $i_1 < i_2 < \ldots < i_k$, такой, что $p[i_1] > p[i_2] > \ldots > p[i_k]$. Найти число k-инверсий за $\mathcal{O}(nk \log n)$.

Дополнительные задачи

7. Дан массив из n элементов, можно сделать предобработку за $\mathcal{O}(n \log n)$. Запросы: количество различных чисел на отрезке $[L, R]$. *Тут будет подсказка про prev[i].*

   (a) online за $\mathcal{O}(\log^2 n)$.

   (b) offline за $\mathcal{O}(\log n)$.

8. Запросы: k-е по порядку среди различных чисел на отрезке $[L, R]$.

   (a) offline за $\mathcal{O}(\log^3 n)$.

   (b) online за $\mathcal{O}(\log^3 n)$.

9. Дан набор точек на плоскости, лежащих в квадрате $[0, S] \times [0, S]$. Найти за $\mathcal{O}(n \log n)$ квадрат max площади, лежащий целиком в $[0, S] \times [0, S]$ и не содержащий ни одной точки.

10. Дано m отрезков на прямой. Запросы: даётся точка, вывести все отрезки, которые ее покрывают, и которые ещё не были выведены раньше (таким образом, каждый отрезок будет выведен не более одного раза за всё время). Суммарное время работы на n запросов: $\mathcal{O}((m + n) \log n)$

11. Придумайте, как в задаче 4 обойтись $\mathcal{O}(n \log \log n)$ отрезков.