SPb HSE, ПАДИИ, 1 курс, осень 2024/25
Практика по алгоритмам #17

BST

30 января

Собрано 30 января 2025 г. в 22:12

Содержание

BST

1. Чем BST лучше хеш-таблицы?
2. `find` за $\mathcal{O}(1)$.
3. `next`/`prev` за $\mathcal{O}(1)$.
4. C++: `set`, `multiset`, `set::lower_bound`
5. Как через BST отсортировать массив?
6. Почему не бывает `add` быстрее $\mathcal{O}(\log n)$?
7. Обход xLR $\to$ дерево (а если нельзя хранить отца?)
8. Точки на прямой: `add`, `del`, поддерживать две ближайших.
9. По k найти k-ый элемент дерева.
10. По x вернуть позицию в дереве.
11. AVL: короткий код малого вращения (в одну строку).
12. AVL: короткий код большого вращения (в две строки).
13. Запросы в дереве
    a) Минимальный x точек $\langle x, y\rangle$: $y \in [l..r]$.
    b) Сумма y пар $\langle x, y\rangle$ у которых $x \in [l..r]$.
    c) Запросы: добавить x; посчитать сумму x: $l \leqslant x \leqslant r$; посчитать сумму x, добавленных в моменты времени с l по r.
14. Точки в стакане. Найти любую точку $\langle x, y\rangle$: $l \leqslant x \leqslant r$, и $y \geqslant d$.
15. Как все элементы сделать различными?
16. Пусть у нас есть массив. Пусть ключ – позиция в массиве. Реализуйте вставку нового элемента на i-ую позицию: $[2, 7, 8, 3] \to [2, 7, 4, 8, 3]$.
17. `(*)` **Монотонные пути**

    Дан взвешенный орграф. Найти самый **длинный** по суммарному весу рёбер путь, на котором рёбра строго возрастают и по номеру, и по весу.
18. `(*)` **Вставка ключевых значений**

    Изначально все ячейки пусты. Нужно обрабатывать запросы вида `Insert(i, x)`. При этом, если i-я ячейка занята, она и все прилегающие справа занятые ячейки сдвигаются вправо. Пример: $(5, 1); (5, 2); (5, 3); (1, 7); (1, 8); (2, 9) \to [8, 9, 7, 0, 3, 2, 1]$.

Разбор задач практики

1. BST умеет `next`/`lower_bound`. Если нужны только `add`/`del`/`find` хеш-таблица быстрее.
2. `find` за $\mathcal{O}(1)$: добавим `hash_map: x` $\to$ `node*`.
3. `next`/`prev` за $\mathcal{O}(1)$: добавим поля `node->next`, `node->prev` (двусвязный список).
4. `C++`: `set` – RB-дерево; `multiset` – храним одинаковые, удалять `s.erase(s.find(x))`; `s.lower_bound` – спуск по дереву, не путайте с `lower_bound(s.begin(), s.end())`; `min`: `s.begin()`; `next`/`prev` – `++it`/`--it` на итераторах.
5. Сортировка через BST: всё добавить, обойти дерева dfs-ом.
6. Сортировок за $o(n \log n)$ нет $\Rightarrow$ `add` за $o(\log n)$ также нет.
7. Обход xLR $\to$ дерево: стек, в котором храним убывающие x, как только нашли что-то большее – поднимаемся, цепляем, как правого сына последнего снятого со стека.
8. Точки на прямой. 2 BST: по x; и хранящее расстояние между соседними по x.
9. По k найти k-ый элемент дерева:
   храним размеры поддеревьев; пересчитываем при `add`/`del`; если `v->L->size` $\geqslant k$, идём влево.
10. По x вернуть позицию в дереве:
    размеры уже есть $\Rightarrow$ просто спускаемся; при повороте направо прибавляем `v->L->size+1`.
11. `rotateRight`: `return new node {v->L->x, v->L->L, new node v->x, v->L->R, v->R};`
    Тут `node {x, L, R}` и есть специальная циклическая вершина `null`:
    `null->L = null->R = null` $\Rightarrow$ обращаясь к `v->L->x` мы не получим проблем.
12. Вытаскиваем в корень `v->L->R->x`: `v->L = rotateLeft(v->L); return rotateRight(v);`
13. a) Храним BST по y, в вершине `minX` поддерева.
    b) Храним BST по x, в вершине `sumX` поддерева.
    c) Храним BST по x и массив по времени, в BST `sumX`, на массиве преф-суммы.
14. Точки в стакане: BST по x, в вершине храним $\max y$ поддерева. Решение `=` спуск по дереву.
15. Вместо x храним $\langle x, i \rangle$, где i – номер элемента.
16. Спускаемся к i-й позиции, вставляем рядом.
    Ко всем позициям, которые правее (больше), нужно сделать `+=1`. На самом деле ключ-позицию можно не хранить, а восстанавливать, зная размеры поддеревьев, тогда дерево будет называться «деревом по неявному ключу».
17. **(*) Монотонные пути**

    Добавляем рёбра в пустой граф в порядке возрастания номера. Новое ребро $e\colon a \to b$ веса w_e можно дописать в конец пути, заканчивающемся в a, если предыдущее ребро пути было меньше w_e. У каждого пути есть два параметра — длина d и вес последнего ребра x. Среди всех путей, заканчивающихся в a и с $x < w_e$ нужен максимальный по d. Если есть два пути (x_1, d_1) и $(x_2, d_2)\colon x_1 \geqslant x_2$ и $d_1 \leqslant d_2$, то первый хуже по обоим параметрам и не нужен.

    $\Rightarrow \forall v$ поддерживаем массив пар $\langle x_1, d_1 \rangle, \langle x_2, d_2 \rangle, \ldots : x_i < x_{i+1},\ d_i < d_{i+1}$ – параметры путей, кончающихся в v. Храним эти пары, конечно, в дереве поиска $tree[v]$.

Заметим, спускаться по такому дереву можно как по x, так и по d.

Добавление ребра $e\colon a \to b$ веса w_e:
1. `<d,x> = prev(lower_bound(tree[a],`w_e`))`,
2. Выкинуть из `tree[b]` бесполезные из-за `<d+`w_e`,`w_e`>` пары,
3. `tree[b].add(<d+`w_e`,`w_e`>)`.

Пар всего m, каждая один раз пришла и один раз ушла.
Обработка ребра за амортизированное $\mathcal{O}(\log n)$. Итого время $\mathcal{O}(m \log n)$.

18. `(*)` **Вставка ключевых значений**

За n вставок в позиции $1..m$ мы используем $\leqslant n+m$ первых ячеек $\Rightarrow$ заведём BST по неявному ключу (ключ=позиция) на массиве длины $n+m$.

`Insert`(i, x). Вставим x на позицию i. Тогда все после i сдвинулись на один вправо. Но нам нужно, чтобы сдвинулись только те, которые прилегали к i. Поэтому надо удалить ближайшую к i справа пустую клетку.

Для поиска «ближайшей пустой клетки справа от i» храним число пустых клеток в поддереве каждой вершины. Пересчитывается также, как и размер поддерева $\Rightarrow$ мы умеем искать j-ю по порядку пустую клетку в дереве. Если у нас есть `Split`, отрежем суффикс $[i, n+m]$ и найдем в нем первую пустую. Если операции `Split` нет, посчитаем число пустых клеток слева от i, пусть их k, найдём $(k+1)$-ю пустую клетку.

Другой подход: хранить дерево для каждой связной занятой области, помнить размер области и ее начало. Если области в какой-то момент соприкасаются, сливать соответствующие деревья.

Домашнее задание

1. **(2) k-min**

   Придумайте, как модифицировать AVL-дерево, чтоб за $\mathcal{O}(k)$ отвечать на запрос «дайте k минимальных ключей»? Что дополнительно нужно хранить? (есть два решения).

2. **(2) Глубины вершин в несбалансированном дереве**

   Рассмотрим процесс добавления ключей в обычное несбалансированное BST.
   Одна операция добавления может работать за $\Omega(n)$.
   В процессе мы поддерживаем `struct Node { Node *l, *r, *p; int depth, key; }`.
   Научитесь моделировать этот процесс за $\mathcal{O}(\log n)$ на запрос добавления.

3.1. Дополнительная часть

1. **(2) Операции над AVL**

   Пусть есть BST, дети которого – корректные AVL деревья, глубина которых отличается на k.
   Научитесь за $\mathcal{O}(k)$ вращений делать из дерева корректное AVL.

2. **(2) Площадь покрывающего прямоугольника**

   Придумать структуру данных, которая поддерживает множество точек $S \subseteq \mathbb{N}^2$ со следующими операциями, каждая за $\mathcal{O}(\log n)$, где $|S| = n$:

   - добавить/удалить точку (x, y);
   - найти min площадь прямоугольника со сторонами, параллельными осям координат, покрывающего все такие точки из S, что $x \in [l, r]$.