Первый курс, весенний семестр

Практика по алгоритмам #1

AVL, Treap, Implicit key, Persistent

Contents

1 Новые задачи

1. Количество деревьев поиска из n различных элементов.
2. Задачи про AVL
   a) Пусть `root.l.h = root.r.h + 3`. Как перебалансировать дерево за $\mathcal{O}(1)$?
   b) Пусть `root.l.h = root.r.h + k`. Как перебалансировать дерево за $\mathcal{O}(k)$?
   c) Придумайте merge за $\mathcal{O}(\log n)$.
   d) Придумайте split за $\mathcal{O}(\log^2 n)$.
   e) Уменьшите количество дополнительной информации до двух бит на вершину.
3. Простые задачи
   a) Запросы: добавить пару $\langle x, y \rangle$; удалить пару $\langle x, y \rangle$; посчитать сумму y по всем парам таким, что $l \le x \le r$.
   b) Запросы: 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 $l \le y \le r$.
   c) Запросы: добавить x; посчитать сумму $x \colon l \le x \le r$; посчитать сумму x, добавленных в моменты времени с l по r.
   d) Предыдущая задача плюс запрос удаления x.
4. Научиться обрабатывать запросы: `add(i, x)`, `del(i)`, `add(l, r, value)`, `sum(l, r)`
5. Тоже самое, но прибавление по модулю 5, а сумма по-прежнему без модуля.
6. Задачи про Treap
   a) Придумайте по аналогии с insert реализацию операции delete через 1 merge
   b) Нужно обрабатывать запросы `add`, `del`, `find`. Пусть мы решаем задачу декартовым деревом (без хеш-таблицы). Пусть есть два типа элементов – маленькое множество A и большое множество B. Пусть k_A запросов k A, k_B запросов к B. Как сделать суммарное время работы равным $\mathcal{O}(k_A \log |A| + k_B \log |B|)$.
   c) Мы хотим хранить пары $\langle a_i, b_i \rangle$. Можно ли по ключу a_i построить декартово дерево, а в каждой вершине дерева поддерживать сумму всех b_i в поддереве? За сколько будут работать операции с таким деревом? А можно поддерживать декартово дерево всех b_i в поддереве? За сколько будут работать операции с таким деревом?
   d) Нарисуйте все деревья, которые могут получиться в результате операции `Merge(бамбук идущий влево-вниз, вершина)` и `Merge(бамбук идущий вправо-вниз, вершина)`.
   e) Проблема с одинаковыми ключами. Что будет, если `Add = GoDown + Split`? Заметим, что `GoDown` можно делать двумя способами: `if(root.x > x)` и `if(root.x` $\geq$ `x)`.
7. Предъявите n операций с persistent RBST (add, delete, split, merge), которые работают за $\Theta(n^2)$. Есть ли такая последовательность операций для AVL-дерева (у которого есть только add и delete)?
8. Persistent List: запросы `merge`, `(push/pop)(front/back)`.
   a) Всего не более N запросов.
   b) А если изначально число запросов не известно?
9. Задача "вставка ключа". Изначально все ячейки пусты. Нужно обрабатывать запросы вида `Insert(i, x)`. При этом, если i-я ячейка занята, все элемент сдвигаются вправо. Пример: $(5, 1); (5, 2); (5, 3); (1, 7); (1, 8); (2, 9) \to 8, 9, 7, 0, 3, 2, 1$.

2 Домашнее задание

2.1 Обязательная часть

2. **(3) Глубины вершин в несбалансированном дереве.**
   В обычное несбалансированное бинарное дерево поиска добавляем различные элементы в порядке $x_1, x_2, \ldots, x_n$. Мы хотим поддерживать дерево в виде "массив отцов всех вершин". Нужно после каждого добавления за $\mathcal{O}(\log n)$ обновлять массив отцов.

3. **(3) Площадь покрывающего прямоугольника.**
   Придумать стурктуру данных, которая поддерживает множество точек $S \subseteq \mathbb{N}^2$ со следующими операциями, каждая за $O(\log n)$, где $|S| = n$:
   a) Добавить точку x.
   b) Удалить точку x.
   c) Найти минимальную площадь прямоугольника со сторонами параллельными осям координат, который покрывает все точки из S, чьи координаты по оси Ox лежат в отрезке $[l, r]$.

2.2 Дополнительная часть

1. **(5) Двухмерный treap.**
   Придумайте аналог treap для хранения точек на плоскости с операциями `splitX`, `mergeX`, `splitY`, `mergeY`. Все операции должны работать за $o(n)$.

2. **(5) Тест против недо-AVL-дерева.**
   Постройте тест, на котором недо-AVL-дерево, которое при $L.h \geq R.h + 2$ всегда делает ровно одно малое вращение вправо (и симметрично при $R.h \geq 2L.h + 2$), 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 $\Theta(n^2)$.

```
void add( node* &t, int x ) {
  if (t == node::null)
    t = new node(x);
  else if (t->x == x)
    return;
  else if (x < t->x) {
    add(t->l, x);
    if (t->l->h > t->r->h + 1)
      t = rot_left(t);
  } else {
    add(t->r, x);
    if (t->r->h > t->l->h + 1)
      t = rot_right(t);
  }
  t->calc();
}
```