Декартовы деревья

Теория

- Введение — необходимость структуры для хранения и быстрого поиска нужных элементов, и других подобных операций.

- Обычные деревья поиска, выполнение поиска, добавления в них, необходимость в малой глубине для эффективной работы (что легко сделать в оффлайне, но трудно в онлайне — АВЛ-деревья, красно-чёрные, и т.д. — они поддерживают дерево с помощью различных вращений).

- Определение декартового дерева (treap), конструктивное доказательство существования (и единственности, если всё различно), эффективный (за $O(n \log n)$) алгоритм построения в оффлайне.

- История: придуманы Vuillemin'ом в 1980-м году, переоткрыты Seidel и Aragon в 1996 г.

- Построение алгоритмов поиска, вставки, удаления.

- Введение операций split (разрезание) и join (конкатенация) и выражение через них всех остальных операций.

- Доказательство оценок при условии различности ключей и приоритетов. Для этого доказывается, что мат. ожидание глубины дерева есть $O(\log n)$.

  – Вводим случайную величину $A_{i,j}$ — i — предок j.
  – Глубина $D(x_k) = \sum_{i=1}^{n} A_{i,k}$.
  – Лемма: x_i предок x_j тогда и только тогда, когда приоритет p_i является максимальным на отрезке $[i;j]$ (считая, что элементы упорядочены по ключу). Доказательство. Докажем это для корня: пусть корень — это элемент x_r, тогда для всех пар (r,j) и (i,r) лемма очевидно выполнена. Понятно, что отрезки $[1; r-1]$ и $[r+1; n]$ — левое и правое поддеревья корня; для всех пар (i,j), лежащих внутри одного из них, лемма верна по индукции. Наконец, для всех пар (i,j) таких, что i лежит в левом поддереве, а j — в правом, лемма тоже выполнена, т.к. один не может быть предком другого.
  – Отсюда сразу получаем, что мат. ожидание $M(A_{i,j}) = \frac{1}{|i-j|+1}$. Это следует из случайности распределения приоритетов.
  – Отсюда получаем формулу для мат. ожидания глубины: $M(D(x_k)) = \sum_{i=1}^{n} \frac{1}{|i-k|+1} = H_k + H_{n-k+1} - 1 < 1 + 2\ln n = O(\log n)$, где $H_m = \sum_{i=1}^{m} \frac{1}{i}$. Мы показали, что мат. ожидание глубины любой вершины есть величина $O(\log n)$.
  – Без доказательства: оценка распределения глубины всего дерева:

  $$P\{D(x_k) \geq 1 + 2c \ln n\} < 2 \left(\frac{n}{e}\right)^{-c \ln \frac{c}{e}}$$

  для любых $k = 1 \dots n$ и $c > 1$.

  Это и означает, что глубина всего дерева есть случайная величина, которая с большой вероятностью есть $O(\log n)$.

- Замечание о том, что различность приоритетов не так важна при условии, что они случайные. А вот много одинаковых ключей приведут к вырожденности дерева, если не принять специальных мер.

- Реализация декартова дерева: структура данных, split, merge, вставка, поиск, удаление, избавление от многочисленных выделений памяти.

- Избавление от явного хранения приоритетов:

  – Приоритеты как хеши от ключей. Этот метод хорош тем, что тогда одинаковому набору ключей будут всегда соответствовать одинаковые декартовы деревья. Недостаток — необходимость многочисленного вычисления хешей.
  – Адреса структур как приоритеты. Этот метод хорош своей скоростью, однако при добавлении нового элемента возможно придётся обменять его с каким-то старым, а, значит, придётся поддерживать ссылки на предка, и следить, чтобы корень дерева не потерялся.

Задачи

- Модификация для нахождения k-го элемента и, наоборот, подсчёта порядкового номера элемента. Для этого просто вводим параметр cnt, задающий размер поддерева, и поддерживаем его значение на выходе из каждой из функций. Переформулировка этой задачи: реализация структуры данных "skip lists", которая предоставляет связанный список с возможностью быстрого поиска не только следующего, а любого элемента.

- Модификация для нахождения суммы на заданном отрезке (сумма всех ключей, попадающих в заданный диапазон $[l; r]$). Для этого вводим параметр sum, содержащий сумму на всём поддереве. Аналогично можно решать задачу, когда суммировать надо не ключи, а соответствующие им значения. Понятно, что вместо суммы можно реализовать какую-то другую операцию.

- Так называемые неявные декартовы деревья. Это дерево, построенное над массивом, где в качестве ключей берутся индексы в этом массиве. Тогда получается, что можно за $O(\log n)$ имитировать вставку/удаление в любое место этого массива, искать k-ый элемент массива (см. первую задачу), вырезать любой подотрезок этого массива (два splita), переставлять два подотрезка местами (комбинация нескольких splitов и joinов), находить сумму на заданном подотрезке (см. вторую задачу), решать задачу о покраске подотрезков (ввести дополнительный флаг — покрашен ли подотрезок), инвертировать данный подотрезок (ввести в вершину дополнительный флаг — инвертирована ли она), и т.д.

  В этом понимании неявное декартово дерево — это мощное обобщение дерева отрезков, которое умеет делать всё то, что умело дерево отрезков, но в онлайне, и даже ещё больше.

- Построение декартова дерева в оффлайне за линейное время (при условии, что данные уже отсортированы по ключам).

  - Первый алгоритм таков. Добавляем значения в дерево по очереди, в порядке увеличения ключей. Для этого сначала встаём в вершину, соответствующую предыдущему добавленному элементу, и начинаем подниматься от этой вершины, пока не найдём элемент, у которого приоритет больше, чем у добавляемого. Тогда останавливаемся, и вставляем новую вершину сюда; правого сына старой вершины отправляем в левого сына добавленной. Суммарная асимптотика будет линейной, потому что один подъём по предкам укоротит самый правый путь в дереве на один, но этот правый путь удлиняется только на один за одно добавление элемента, поэтому в сумме удалений будет $O(n)$.
  - Второй алгоритм. Заметим, что если какая-то вершина является левым сыном своего предка, то этот предок — не что иное, как ближайшая справа по ключу вершина, имеющая бОльший приоритет, чем наша вершина. Аналогично, если вершина — правый сын своего предка, то этот предок — ближайшая слева по ключу, имеющая бОльший приоритет. Если у какой-то вершины есть сразу две вершины, обладающие этими свойствами, то предком будет та, у которой приоритет меньше. Отсюда можно получить такой алгоритм: для каждой вершины заранее найти этого ближайшего слева и справа, для чего воспользоваться двумя проходами со стеком.

- Решение задачи RMQ с помощью декартовых деревьев. Заметим, что если мы по заданному массиву построим декартово дерево (взяв в качестве ключей — позиции, а в качестве приоритетов — содержимое массива), то для любых двух позиций (i, j) их $\mathrm{lca}(i, j)$ даст вершину с ключом в отрезке $[i; j]$, и с приоритетом, являющимся наибольшим в данном отрезке. Таким образом, мы свели произвольную задачу RMQ к задаче LCA на дереве, которая, как известно, решается за линейное время путём сведения к задаче RMQ частного вида.