Алгоритмы. Домашнее задание №1

Горбунов Егор Алексеевич

17 февраля 2016 г.

Задание №1 Пусть `splay`-дерево поддерживает множество S. Каждый элемент $x_i \in S$ был запрошен $p_i m$ раз, где m — общее число запросов. Гарантируется, что $0 < p_i \leqslant 1$ и $\sum_i p_i = 1$. Докажите, что `splay`-дерево обрабатывает все запросы за время $\mathcal{O}\Big(m \cdot \big[1 + \sum_i p_i \cdot \log \frac{1}{p_i}\big]\Big)$.

Решение: Время обработки запроса элемена равно времени работы операции $splay$. Посчитаем амортизационную стоимость этой операции.

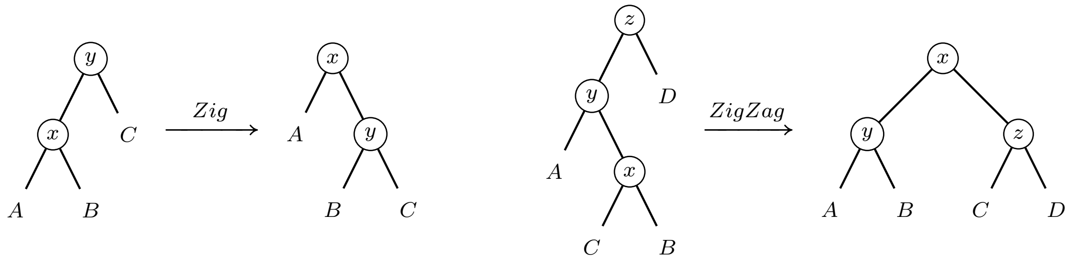

Рис. 1: Повороты

Всё дерево будем обозначать как T, а поддерево с корнем в x как $T(x)$. Рассмотрим весовую функцию равную числу запросов на вершинах в поддереве $T(x)$:

$$\omega(x) = \sum_{i \in T(x)} p_i m$$

Введём следующий потенциал для вершины и для дерева:

$$\Phi(x) = \log \omega(x)$$
$$\Phi(T) = \sum_{x \in T} \Phi(x)$$

Важно заметить, что если x предок y, то $\Phi(x) \geqslant \Phi(y)$, а так же, функция $\omega(x)$ аддитивна. Таким образом, весь анализ, который мы проделывали для операций Zig, $ZigZag$ и $ZigZig$ с использованием весовой функции равной числу вершин в поддереве, верен и для введённых сейчас ω и Φ. Т.е. легко показывается, что учётная стоимость операции Zig:

$$\tilde{c}(Zig) \leqslant 3(\Phi'(x) - \Phi(x)) + 1$$

А учётная стоимость операции $ZigZag$ и $ZigZig$:

$$\tilde{c}(ZigZag) = \tilde{c}(ZigZig) \leqslant 3(\Phi'(x) - \Phi(x))$$

Таким образом, учётная стоимость операции $splay(x)$ равна:

$$\begin{aligned}\tilde{c}(Splay(x)) \le 3(\Phi_{final}(x) - \Phi_{start}(x)) + 1 = 3(\log \sum_{i\in T} p_i m - \Phi_{start}(x)) + 1 \\ = 3(\log m - \Phi_{start}(x)) + 1 \leqslant \left\{\text{всяко: } \Phi_{start}(x) \geqslant \log p_x m\right\} \\ \leqslant 3(\log m - \log p_x m) + 1 = 3\log\frac{m}{p_x m} + 1 = 1 - 3\log p_x\end{aligned}$$

Таким образом, амортизированная стоимость всех запросов элемента x равна $p_x m \cdot (1 - 3\log p_x)$, а тогда амортизированная стоимость всех запросов равна:

$$\sum_{i\in T} p_i m \cdot (1 - 3\log p_i) = m - 3m\sum_{i\in T} p_i \log p_i = m + 3m\sum_{i\in T} p_i \log\frac{1}{p_i} = \mathcal{O}\left(m \cdot (1 + \sum_{i\in T} p_i \log\frac{1}{p_i})\right)$$

■

Задание №2 Придумать структуру, поддерживающую упорядоченный список S целых чисел со следующими операциями с временем обработке $\mathcal{O}(\log|S|)$:

- `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 $\max_{l\leqslant j<k\leqslant r} |S[j] - S[k]|$
- `min(l,r)` — найти $\min_{l\leqslant j<k\leqslant r} |S[j] - S[k]|$

Решение: Не умаляя общности будем считать, что S упорядочен по возрастанию. Заметим, что тогда для любого отрезка $[l, r]$ верно:

$$S[l] \leqslant S[l+1] \leqslant S[l+2] \leqslant \ldots \leqslant S[r-2] \leqslant S[r-1] \leqslant S[r]$$

Отсюда выполнение операции `max(l,r)` вырождается в нахождение разности:

$$\max_{l\leqslant j<k\leqslant r} |S[j] - S[k]| = S[r] - S[l]$$

А выполнение операции `min(l,r)` вырождается в нахождение минимальной разности между лишь соседними элементами:

$$\min_{l\leqslant j<k\leqslant r} |S[j] - S[k]| = \min_{i\in[l,r)} (S[i+1] - S[i])$$

Будем поступать так: заведём и будем поддерживать следующие структуры над S:

- BT — сбалансированное дерево поиска, ключом в котором являются элементы S. С таким деревом мы умеем проделывать операцию вставки $BT.insert(key)$, удаления $BT.delete(key)$, нахождения $BT[k]$ — k-го по порядку элемента за $\mathcal{O}(\log n)$, где n — число элементов в дереве.
- CT — декартово дерево по неявному ключу, моделирующее массив S' такой, что $S'[i] = S[i+1]-S[i]$. В этом дереве поддержим операцию минимума на подотрезке. Нам известно, что в таком дереве можно выполнять операции:
  - $CT[k] = v$ — изменения k-ого по порядку элемента
  - $CT.min(l,r)$ — нахождение минимума на подотрезке
  - $CT.insert(k, value)$ — вставка элемента $value$ на k-ую позицию
  - $CT.delete(k)$ — удаление элементы на k-ой позиции

  Причём асимптотическая сложность (в среднем) этих операций $\mathcal{O}(\log n)$, где n — число элементов в дереве.

Теперь покажем, как за $\mathcal{O}(\log|S|)$ выполнять операции из условия задачи:

- `insert(x)`: При вставке x на какую-то позицию i массива S у нас происходит вставка элемента в массив S' на позицию i и изменения $S'[i-1]$. Т.е. последовательно выполняются операции: $i \leftarrow BT.insert(x)$; $CT.insert(i, S[i+1]-S[i])$; $CT[i-1] = S[i]-S[i-1]$. Видим, что все операции занимают $\mathcal{O}(\log|S|)$; $(size(CT) = |S'| = |S|-1)$
- `delete(x)`: $i \leftarrow BT.delete(x)$, где i — позиция, с которой был удалён x. Далее: $CT.delete(i)$; $CT[i$ $S[i+1]-S[i]$
- `S[k]`: просто выполняем $BT[k]$ (умеем)
- `max(l,r)`: возвращаем $S[r]-S[l]$
- `min(l,r)`: возвращаем $CT.min(l, r-1)$

Видно, что все операции выполняются за $\mathcal{O}(\log n)$. ∎

Альтернативное понимание. Если список и вставка происходит на определённую позицию. Список моделируем декартовым деревом по неявному ключу с операцией минимума и максимума. Тогда `insert`, `delete`, `S[k]` реализуются классическим образом. А остальные операции так:

- `max(l, r)` $= max(T) - min(T)$
- `min(l, r)` $= min(min(T.L), min(T.R), T.k - max(T.L), min(T.R) - T.k)$

Задание №3 Пусть приоритеты случайны, а ключи все разные. Найти матожидание количества листьев в Декартовом дереве из n вершин.

Решение: Пусть x_i — вершина i-ая по порядку. Тогда x_i является листом только тогда, когда $priority(x_i) > priority(x_{i+1})$ и $priority(x_i) > priority(x_{i-1})$. Расположим элементы в порядке возрастания:

$$x_1,\ x_2,\ x_3,\dots,x_n$$

Для всех вершин x_i, где $1 < i < n$: $P[x_i$ — лист$] = \frac{1}{3}$, а для x_1 и x_2: $P[x_1$ — лист$] = P[x_n$ — лист$] = \frac{1}{2}$

Задание №4 Пускай приоритеты случайны, а ключи различны. x_k — вершина Декартова дерева, содержащая k-ый по порядку ключ. Всего в дереве n вершин.

- Пусть $1 \leqslant i \leqslant j \leqslant k \leqslant n$. Найти вероятность того, что x_j — общий предок x_i и x_k
- Пусть $1 \leqslant i \leqslant k \leqslant n$. Найти матожидание длины пути между x_i и x_k

Решение:

- Заметим, что x_j является предком x_i только если:

  $$\min\{p(x_i), p(x_{i+1}), p(x_{i+2}), \dots, p(x_k)\} = p(x_j)$$

  Где $p(x_i)$ — приоритет вершины Декартова дерева. Действительно: самая высокя вершина на кратчайшем пути в дереве от x_i до x_j точно содержится в множестве $\{p(x_i), p(x_{i+1}), \dots, p(x_k)\}$. Т.к. кратчайший путь – это сначала подъём до какой-то вершины, а потом спуск до нужной, то если эта самая высокая (близкая к корню) вершина на этом пути не является x_j, то x_j уже не предок x_i, а это эквивалентно тому, что если $p(x_j) \neq \min\{p(x_i), p(x_{i+2}), \dots, p(x_k)\}$, то x_j — не предок x_i (вершины из $x_i \dots x_j$, не попавшие в кратчайший путь не могут содержать вершин с меньшим приоритетом, т.к. все они лежат ниже этого кратчайшего пути).

  Таким образом x_j — предок x_i, если $\min(p(x_i)\dots p(x_j)) = p(x_j)$. Аналогично x_j — предок x_k, если $\min(p(x_j)\dots p(x_k)) = p(x_j)$. В последовательности $p(x_i)\dots p(x_j)$ $|j-i|+1$ чисел, каждое из которых равновероятно может оказаться минимумом, таким образом:

  $$Prob\big[x_j \text{ — предок } x_i\big] = Prob\big[\min(p(x_i)\dots p(x_j)) = p(x_j)\big] = \frac{1}{|j-i|+1}$$
  $$Prob\big[x_j \text{ — предок } x_k\big] = \frac{1}{|j-k|+1}$$

  Но тогда:

  $$Prob\big[x_j \text{ — общий предок } x_i \text{ и } x_k\big] = Prob\big[x_j \text{ — предок } x_i\big]Prob\big[x_j \text{ — предок } x_k\big] = \\ = \frac{1}{(|j-k|+1)(|j-i|+1)}$$

- Введём следующие случайные индикаторные случайные величины:

$$V_{ij} = \begin{cases} 1, \text{ если } x_i \text{ предок } x_j \\ 0, \text{ иначе} \end{cases} \qquad U_{ijk} = \begin{cases} 1, \text{ если } x_j \text{ общий предок } x_i \text{ и } x_k \\ 0, \text{ иначе} \end{cases}$$

  Заметим теперь, что кратчайший путь от x_i до x_k состоит из предков x_i не являющихся общими предками x_i и x_k, а так же из вершин, которые являются предками x_k не являющихся общими предками x_k и x_i плюс 1. Запишем:

$$P(i,k) = 1 + \sum_{1 \le j \le n} (V_{ji} - U_{ijk}) + \sum_{1 \le j \le n} (V_{jk} - U_{ijk})$$

  Ясно, что если при $j > k$ $V_{ji} = 1$, т.е. x_j — предок x_i, то x_j будет и предком x_k (т.е. $C_i jk = 1$) так же, т.к. $i \le k$. Аналогично если при $j < i$ $V_{jk} = 1$, то $U_{ijk} = 1$, т.е:

$$P(i,k) = 1 + \sum_{1 \le j \le k} (V_{ji} - U_{ijk}) + \sum_{i \le j \le n} (V_{jk} - U_{ijk})$$

  Тогда матожидание $P(i,k)$ равно:

$$E[P(i,k)] = 1 + \sum_{1 \le j \le k} (E[V_{ji}] - E[U_{ijk}]) + \sum_{i \le j \le n} (E[V_{jk}] - E[U_{ijk}]) =$$

$$= 1 + \sum_{1 \le j \le k} \left[\frac{1}{|j-i|+1} - \frac{1}{(|j-k|+1)(|j-i|+1)} \right] + \sum_{i \le j \le n} \left[\frac{1}{|j-k|+1} - \frac{1}{(|j-k|+1)(|j-i|+1)} \right]$$

  Последнее равенство, конечно, нужно упростить, но времени нет! ■

Задание №5 Задача про случайное дерево поиска.

Решение: 1) Вероятность того, что вершина – корень равна $\frac{1}{n}$ (показыкается по индукции и разбором операции split и merge). Тогда вероятность дерева: $\prod_i \frac{1}{size(i)}$

Задание №6 Придумайте структуру данных, которая поддерживает следующие операции за $\mathcal{O}(|text|)$:

- Вставка символа на позицию i
- Удаление символа с позиций $[l,r)$
- Копирование подстроки $[l,r)$ в позицию i

Решение: Рассмотрим декартово дерево по неявному ключу CT, в котором будем хранить текст $text$. Только условимся, что операции $split(CT,k)$ и $merge(T_1,T_2)$ всегда возвращают новое дерево и пересоздают все узлы, которые затрагивают.

- Вставка на позицию i будет выполняться как обычная вставка в декартово дерево по неявному ключу.

- Удаление интервала $[l, r)$:

$$T_1, T_2 \leftarrow split(CT, l)$$
$$T_3, T_4 \leftarrow split(T_2, r)$$
$$return(merge(T_1, T_4))$$

- компирование подстроки $[l, r)$ в позицию i:

$$T_{<i}, T_{\geqslant i} \leftarrow split(CT, i)$$
$$T_{<l}, T_{\geqslant l} \leftarrow split(CT, l)$$
$$T_{[l,r)}, T_{\geqslant r} \leftarrow split(T_{\geqslant l}, r)$$
$$return(merge(merge(T_{<i}, T_{[l,r)}), T_{\geqslant i}))$$

  В силу того, что изменяемые элементы будут пересоздаваться, то при изменении одной части дерева другая не сможет испортиться.

Ясно, что по построению эти операции работают в среднем за $\mathcal{O}(\log n)$. ■