SPb HSE, ПАДИИ, 1 курс, осень 2024/25
Практика по алгоритмам #18

Treap

6 февраля

Собрано 10 февраля 2025 г. в 21:32

Содержание

Treap

1. **Повторение**

   Пусть у нас есть массив. Пусть ключ – позиция в массиве. Реализуйте вставку нового элемента на i-ую позицию: $[2, 7, 8, 3] \to [2, 7, \mathbf{4}, 8, 3]$.

2. **Дерево по неявному ключу**

   a) Запросы: `insert(i,x)`, `del(i)`, `get_sum(l,r)` (не забывайте про `split`)
   b) Отложенные операции: `set(l,r,value)`.
   c) Добавим запросы `reverse(l,r)` и `rotate(k)`.

3. **Excel**

   Есть excel-табличка. Научитесь за $\mathcal{O}(\log n)$ обрабатывать запросы

   a) Столбец j подвинуть влево-вправо на d.
   b) Строку i подвинуть вверх-вниз на d.
   c) Поменять/прочитать ячейку $[i, j]$.

4. **Улучшаем и изучаем декартово дерево**

   a) `insert` через один спуск и один `split`.
   b) `del` через один спуск и один `merge`.
   c) Дан отсортированный по x_i массив пар $\langle x_i, y_i \rangle$. Постройте Treap за $\mathcal{O}(n)$ (есть простое решение).

5. **Корректность**

   Покажите, что от фиксированного набора пар $\langle x_i, y_i \rangle$ из различных x_i и y_i $\exists!$ treap.

6. **Время работы Treap**

   Пусть дан массив $x_1, x_2, \ldots, x_n$, по нему был построен Treap.
   На лекции мы показали $\forall i$ глубина узла с ключом x_i имеет матожидание $\mathcal{O}(\log n)$.

   Докажите, что матожидание времени работы `split` и `merge` есть $\mathcal{O}(\log n)$.

7. **Копирование памяти**

   Соделайте массив, который за $\mathcal{O}(\log n)$ умеет $read(i)$, $write(i, x)$, $copy(l, r, i)$. В процессе того, как мы это делаем, нам понадобится магия `copy(treap)` за $\mathcal{O}(1)$. Пусть она уже есть.

8. **Детская персистентность**

   Придумайте персистентный массив, который умеет делать

   a) Обращение за $\mathcal{O}(1)$, модификацию за $\mathcal{O}(n)$.
   b) Обращение за $\mathcal{O}(m)$, модификацию за $\mathcal{O}(1)$.
   c) Обращение за $\mathcal{O}(1)$, модификацию за $\mathcal{O}(1)$, offline.
   d) Частичная персистентность (модифицировать можно только последнюю версию). Обращение за $\mathcal{O}(\log m)$, модификацию за $\mathcal{O}(1)$.

9. **Персистентность для взрослых**

   Придумайте персистентный массив, который умеет делать обращение за $\mathcal{O}(\log n)$, модификацию за $\mathcal{O}(\log n)$.

10. **Персистентное удаление**

    Помните, мы умели делать *ленивое* удаление из BST за $\mathcal{O}(1)$?
    Напишите явно его персистентную версию за $\mathcal{O}(\log n)$.

11. (*) Персистентный СНМ

12. (*) **Зоопарк**

    Вам нужно за $\mathcal{O}(\log^2 n)$ отвечать на запрос на отрезке «количество рекордов в массиве равном отрезку». Рекордами массива называются x, которые строго больше чем все левее него.

13. (*) Найдите матожидание максимальной глубины вершины в случайном дереве.

Разбор задач практики

1. 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».

2. **Дерево по неявному ключу**

   a) `add(i,x)`, `del(i)`, `get_sum(l,r)`, `set(l,r,value)`.

   Дерево по неявному ключу на нашем массиве. В каждой вершине храним:

   - `size` – размер поддерева (для неявного ключа),
   - `sum` – сумма в поддереве,
   - `pending` – число, которое мы лениво хотим присвоить в поддерево.
   - `time` – отметка времени, когда присвоили `pending`.

   Каждый раз, когда заходим в вершину, делаем `push` – проталкиваем `pending` вниз, ставя вершине корректный `sum`.
   Каждый раз, когда меняем вершину, делаем `update` – пересчитываем `size` и `sum`.
   Код приводится для случая, когда определен пустой `node* null`.

```
void push(node* t):
    if (t == null || t->pending == -1) return;
    t->l->pending = t->r->pending = t->pending;
    t->sum = t->size * t->pending;
    t->pending = -1;
int sum(node* t):
    return t->pending == -1 ? t->sum : t->pending * t->size;
void update(node* t):
    if (t == null) return;
    t->size = 1 + t->l->size + t->r->size;
    t->sum = t->x + sum(t->l) + sum(t->r);
```

   - `insert` и `del` делаются обычным образом, но с добавлением вызовов `push` и `update`.
   - `get_sum(l, r)` как в прошлой задаче. Либо, если есть `Split` и `Merge`,

   вырезанием нужного куска, взятием `sum` корня и обратным слиянием.
   - `set(l,r,value)` аналогично `get_sum`, но

   вместо возвращения `sum` происходит `pending = value, time = timer++`

   b) `reverse(l,r)`, `rotate(k)`

   Храним `rev` – нужно ли разворачивать поддерево. Лениво проталкиваем (как и `pending`).

```
def push(t):
    ...
    if t.rev == 1:
        t.l.rev ^= 1; t.r.rev ^= 1; t.rev = 0
        t.l, t.r = t.r, t.l
```

   `rotate(k) = split(k) + merge` в обратном порядке.
   `rotate(k) = reverse(0, n-k) + reverse(n-k, n) + reverse(0, n)`

3. **Excel**

Храним два дерева по неявному ключу: перестановку строк, перестановку столбцов.
Запрос $cell[i, j]$: $table.get(rows.get(i), columns.get(j))$, где $table$ – мэп, хеш-таблица.

4. **Улучшаем и изучаем декартово дерево**

   a) `add` **через один** `split`

   Спускаемся по дереву до позиции вставки v.
   Ставим вершину с x вместо v, делаем ее детьми $\texttt{split}(v, x)$.

   b) `delete` **через один** `merge`

   Спускаемся по дереву до удаляемой вершины, заменяем ее на `merge` ее детей.

   c) *Наивный алгоритм построения дерева*

   В худшем случае $T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$.
   В среднем $\mathcal{O}(n \log n)$, та же рекуррента, что в qsort.

   d) *Декартово дерево за $\mathcal{O}(n)$ по отсортированным x*

   Добавляем вершины по одной. У новой вершины самый большой x, она добавится в правую ветвь. Держим правую ветвь в стеке. Ищем позицию вставки снизу вверх.

```
while (!right_branch.empty() && new_node->y < right_branch.back()->y) {
    new_node->l = right_branch.back();
    right_branch.pop_back();
}
if (!right_branch.empty())
    right_branch.back()->r = new_node;
right_branch.push_back(new_node);
```

5. **Корректность**

   Единственным образом можем выбрать корень, далее единственным образом разделяем на L и R, далее по индукции.

6. **Время работы Treap**

   См. конспект ПМИ.

7. **Копирование памяти**

```
Split(Root, L, A, B)
Split(B, R-L, B, C)
Split(Root1, i, A1, B1)
Split(B1, R-L, B1, C1)
ans = Merge(A, Merge(B1, C))
```

8. **Детская персистентность**

   Изменение за $\mathcal{O}(n)$ – копировать массив полностью (copy on write).
   Изменение за $\mathcal{O}(1)$ – поддерживать дерево версий (всего массива, а не ячеек отдельно!) Для каждой версии хранить версию-отца и изменение относительно отца.
   `get(version, i)` – пройти по дереву версий от `version` до корня, найти последнее изменение i-й ячейки массива.

   Offline-персистентность: обходим дерево версий dfs-ом, вниз делаем изменения, вверх откатываем их.

Частичная персистентность: в каждой ячейке массива последовательность её изменений, добавление – pushback, запрос – бинпоиск.

9. **Персистентность для взрослых**

   Картинка с копированием путей. Напоминание, что все деревья, к которым запросы сверху, одинаковы: BST AVL, treap, ДО, BST по неявному ключу. Массив = BST по неявному ключу. Мы умеем через treap (т.к. у него есть split/merge), вообще split/merge есть у всего, например, у AVL, но мы его не проходили.

10. **Персистентное удаление**

    Лениво: сделали `find` за $\mathcal{O}(1)$, пометили вершину как удалённую.

    В персистентном мире нельзя поменять вершину, можно только создать копию $\Rightarrow$ и копию отца и т.д. Но к отцу мы подниматься не умеем, т.к. у нас потенциально много отцов в разных версиях $\Rightarrow$ единственный вариант персистентного ленивого удаления – рекурсивный `find`, который на обратном ходу рекурсии копирует вершины:

```
pnode Del(pnode v, int x):
    if (v->x == x)
        return new node {v->l, v->r, v->x, 1}; // deleted
    if (x < v->x)
        return new node {Del(v->l, x), v->r, v->x, v->deleted};
    else
        return new node {v->l, Del(v->r, x), v->x, v->deleted};
```

11. **(*) Персистентный СНМ**

    Любая структура данных состоит из массивов, а массив мы умеем делать персистентным.

12. **(*) Зоопарк**

    ?

13. **(*)** Найдите матожидание максимальной глубины вершины в случайном дереве.

    ?

Домашнее задание

1. **(2) Декартовы картины**

   Нарисуйте все деревья, которые могут получиться в результате операции `Merge(бамбук идущий влево-вниз, вершина)` и `Merge(бамбук идущий вправо-вниз, вершина)`.

2. **(2) Новые возможности**

   У вас есть Treap. Вы знаете про персистентность. Создайте структуру данных, которая что-то такое хранит для каждого префикса массива, что может за $\mathcal{O}(\log n)$ отвечать на запрос `get(k,i)` «k-е по значению число на префиксе $[0,i)$». Длина массива $n \leqslant 200\,000$.

3.1. Дополнительная часть

1. **(2) Странная очередь**

   В ряд стоят n групп людей, суммарное количество денег у i-й группы равно a_i. Происходят события – пришло новая группа людей с x деньгами и встала между i-й и $i{+}1$ группами в текущей нумерации; k раз подряд среди всех пар «соседние группы», та пара, в которой сумма денег меньше, объединилась в одну группу (нумерация сдвинулась), если таких пар несколько, самая левая. После каждого запроса возвращать номер самой богатой группы. Обработайте m запросов за $\mathcal{O}((n{+}m)\log n)$.

2. **(3) Шары и урны**

   Рассмотрим n различных шаров и n различных урн, стоящих в ряд. Изначально каждая урна содержит ровно один из шаров. С урнами производят m операций вида `move(i,j,k)` – поднять все шары из отрезка урн $[i, i{+}k)$, и опустить все эти шары в таком же порядке в отрезок урн $[j, j{+}k)$. Отрезки могут пересекаться. По данной последовательности операций выясните для каждого шара, в какой урне он будет находиться после всех перемещений.