Олимпиадное программирование

Занятие 24. Динамическое программирование. Декартово дерево

Труфанов Павел Николаевич

Онлайн-школа  Фоксфорд

Foxford.ru 2019-2020

Симпатичные узоры

Дано прямоугольное поле $n \times m \leq 100$. Требуется покрасить каждую клетку в один из двух цветов так, чтобы не было квадратика со стороной 2 покрашенного в один цвет. Найдите количество таких способов покраски.

Доминошки

Дано прямоугольное поле $n \times m$. Некоторые клетки заняты. Требуется найти количество способов замостить оставшиеся клетки доминошками 1×1 и 1×2.

- $n \times m \leq 100$
- $n \times m \leq 300$

Бинарное дерево поиска

Задача: хотим поддерживать множество, в которое можно добавлять элементы, удалять элементы и проверять наличие элемента в множестве.

Такую задачу может решить бинарное дерево поиска: это двоичное дерево, в каждой вершине будет записано число множества, при этом для каждой вершины верно, что все значения в ее левом поддереве меньше значения вершины, а все значения в правом больше.

Операция поиска

Встаем в корень дерева. Если значение в корне искомое, то мы нашли число. Если оно больше, чем искомое число, то тогда наше число может быть только в левом сыне. В противном случае оно справа.

Данная операция будет работать за время, пропорциональное высоте дерева. Проблема – бинарное дерево поиска при определенных операциях добавления может стать слишком высоким и тогда операции будут выполняться долго.

Декартово дерево

Решение - сбалансировать дерево. Дерево размера n называется сбалансированным, если его высота $O(\log n)$. Мы хотим получить дерево с такими же свойствами, только сбалансированное. Давайте в вершину добавим еще один параметр. Назовем его приоритетом. Пусть приоритет соблюдает свойство кучи, то есть в вершине приоритет будет больше, чем в его детях. Если теперь мы для каждой вершины приоритет будем выбирать случайно и добавлять элементы в дерево, поддерживая это свойство, то дерево магически станет сбалансированным.

Структура вершины

```
struct Node {
    int key, prior;
    Node *l, *r;
    Node():l(nullptr), r(nullptr){}
    Node(int key):
        key(key), prior(rand()),
        l(nullptr), r(nullptr){}
};
typedef Node* PNode;
```

Вспомогательные функции

Чтобы реализовать почти любую операцию в декартовом дереве, мы напишем две вспомогательные функции: split и merge. Функция split будет брать дерево с корнем в root и число x. Она будет делить наше дерево на два, где все ключи в первом меньше x, а во втором больше либо равны x. Функция merge будет делать обратное, брать два дерево с таким свойством и соединять его обратно.

Split

```
void split(PNode root, PNode& l,
           PNode& r, int x) {
    if (!root) {
        return void(l = r = nullptr);
    }
    if (root->key < x) {
        split(root->r, root->r, r, x);
        l = root;
    } else {
        split(root->l, l, root->l, x);
        r = root;
    }
}
```

Merge

```
void merge(PNode& root, PNode l,
           PNode r) {
    if (!l || !r) {
        return void(root = l ? l : r);
    }
    if (l->prior > r->prior) {
        merge(l->r, l->r, r);
        root = l;
    } else {
        merge(r->l, l, r->l);
        root = r;
    }
}
```

Операции

Операцию вставки теперь реализовать легко.
Пусть мы хотим вставить ключ x. Создадим дерево из одной вершины с ключом x. Теперь поделим наше исходное дерево по ключу x функцией split на части l, r. Осталось всего лишь склеить три дерева в порядке l–x–r.
Операция деления происходит похоже. Мы просто вырежем вершину с нужным ключом x, порезав дерево по x и x+1.
Операция поиска тривиальна.

Более умные операции

До этого операции вставки и удаления работали за три логарифма. Хотим сделать за один.

Прокачаем дерево

Теперь хочу ввести индексацию над деревом. Для этого давайте для каждого поддерева поддерживать функцию его размера. Мы сможем ее обновлять в функциях split и merge.

Получение элемента по индексу

Операция похожа на операцию спуска по дереву отрезков. Мы каждый раз идем в нужную сторону в зависимости от размера поддеревьев

Получение индекса по элементу

Операция по сути является поиском элемента в дереве. В процессе мы подсчитываем сколько элементов идут до нас.