МФТИ, ФПМИ

Алгоритмы и структуры данных, осень 2021

Семинар №7. Амортизационный анализ, разреженные таблицы

1. Покажите, как можно реализовать `deque`, динамический массив, позволяющий также вставлять элементы в начало и удалять элементы из начала. Учётное время работы всех операций должно быть $O(1)$.

2. Рассмотрим стек с операцией `multipop`(k): в дополнение к классическим `push` и `pop` добавляется операция извлечения k верхних элементов стека. Покажите, что амортизационная сложность всех операций (при обычной реализации стека односвязным списком) равна $O(1)$. Покажите, почему нельзя сказать, что каждая операция работает за чистое $O(1)$.

3. Куча Фибоначчи позволяет отвечать на запросы типа `insert` за $O(1)$, на запросы типа `decreaseKey` — за $O^*(1)$ (то есть за амортизированное $O(1)$), на запросы типа `extractMin` — за $O^*(\log n)$. Алгоритм Дейкстры использует структуру данных, которая n раз вызывает `insert`, не больше $m \in [n-1, n^2]$ раз вызывает `decreaseKey` и n раз вызывает `extractMin`. Докажите, что при использовании кучи Фибоначчи получится асимптотика $O(m+n\log n)$. Сравните это время с временем, получаемым при использовании двоичной кучи.

4. Пусть имеется массив a длины n, а также m запросов $foo(l,r)$ и $bar(l,r,x)$, реализация которых приведена ниже. Используя метод амортизационного анализа с помощью потенциалов покажите, что количество вызовов функции $doMagic()$ имеет порядок $O(n+m)$. Функция $doMagic()$ не изменяет массив.

```
void foo(int l, int r) {
    for (int i = l + 1; i <= r; ++i) {
        if (a[i] != a[i - 1]) {
            doMagic();
        }
    }
    for (int i = l + 1; i <= r; ++i) {
        a[i] = a[i - 1];
    }
}

void bar(int l, int r, int x) {
    for (int i = l; i <= r; ++i) {
        a[i] += x;
    }
}
```

---

5. Пусть есть куча, которая умеет выполнять `insert`, `merge`, `extractMin` за $O(\log n)$, а `heapify` (построить кучу по данному множеству ключей) — за $O(n)$. На её основе постройте кучу, которая умеет делать всё то же (амортизированно), но `insert` — за чистое $O(1)$.

6. Реализуем бинарный поиск на расширяющемся множестве чисел (массиве). Если массив отсортирован и не меняется, то проверять наличие в нём конкретного числа можно за $O(\log n)$, где n — длина массива. Пусть теперь в массив могут вставляться новые элементы.

Будем хранить массив как объединение нескольких отсортированных массивов, размер каждого из которых равен какой-то степени двойки, причём все степени различны (по аналогии с биномиальной кучей). Как тогда проверить, входит ли данное число в массив? За сколько это будет работать?

Как вставлять новый элемент в такую структуру? Проанализируйте время выполнения этой операции в худшем случае, а также её амортизированное время работы.

7. Объясните, как в статическом массиве находить не только минимум на отрезке, но и позицию его вхождения.

8. Предложите структуру данных, которая бы поддерживала массив чисел и позволяла бы: а) добавлять элемент в конец массива за $O^*(\log n)$, где n — текущий размер массива; б) узнавать минимальное значение на подотрезке за $O(1)$.

9. На какие запросы помимо `min` можно отвечать с помощью sparse table?

10*. Реализуем sparse table, позволяющий работать с другими типами ассоциативных функций f (например, сложение, произведение, особенно произведение матриц, и пр.).

а) Для каждого числа $i \in [1, n-1]$ определим $zeros(i)$ как длину блока из нулей в конце двоичной записи i. В частности, если i нечётно, то $zeros(i) = 0$. Для каждого $i \neq 0$ насчитайте $f(a_i, a_{i+1}, \ldots, a_r)$ для всех $i \leqslant r < i + 2^{zeros(i)}$. Покажите, что суммарная длина таких массивов равна $O(n \log n)$.

б) Для каждого числа $i \in [0, n-1]$ определим $ones(i)$ как длину блока из единиц в конце двоичной записи i. В частности, если i чётно, то $ones(i) = 0$. Для каждого i насчитайте $f(a_l, a_{l+1}, \ldots, a_i)$ для всех $i - 2^{ones(i)} < l \leqslant i$. Покажите, что суммарная длина таких массивов равна $O(n \log n)$.

в) С помощью предподсчёта за $O(n)$ научитесь определять старший бит в любом числе $i \in [0, n]$ за $O(1)$.

г) Теперь найдём $f(a_l, \ldots, a_r)$ для произвольной пары $l < r$. Пусть l и r имеют одинаковый префикс x в двоичной записи, то есть $l = \overline{x0\ldots}, r = \overline{x1\ldots}$. Положим $m = \overline{x100\ldots0}$. Как построить такое m за $O(1)$? После этого остаётся получить ответ, зная $f(a_l, \ldots, a_{m-1})$ и $f(a_m, \ldots, a_r)$, которые уже подсчитаны.