ДЕРЕВО ОТРЕЗКОВ

Школа::Кода

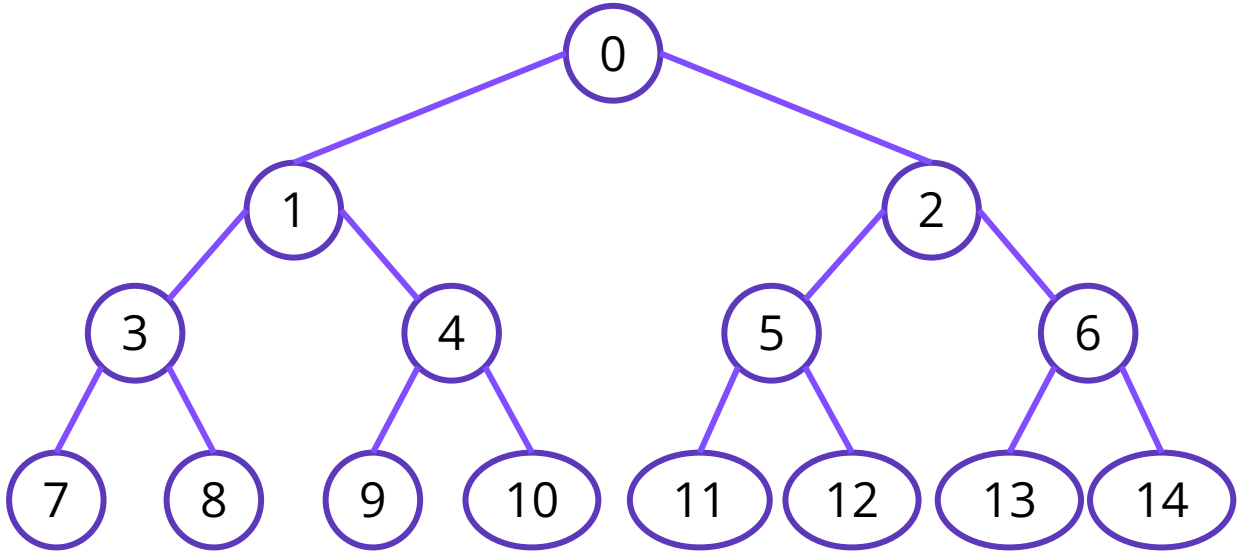
Олимпиадное программирование

2020-2021 Таганрог

Определение

Дерево отрезков (англ. Segment tree) — структура данных, требующая 4*n памяти и позволяющая эффективно (за O(log(n))) выполнять следующие операции:

- изменять значение любого элемента в массиве или элементов на отрезке [i,j];
- выполнять некоторую операцию на отрезке [i,j] (сумма, максимумб минимум и др.).



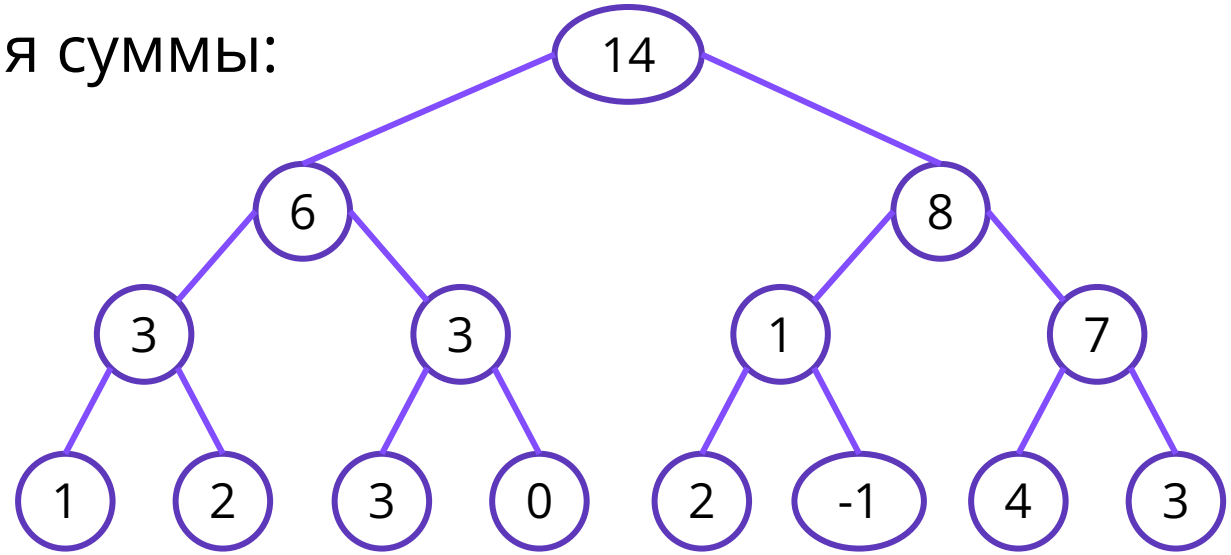
Основная идея

Пусть дан массив, и запросы, требующие изменять элементы этого массива или вычислять функцию на отрезке массива.

Строится бинарное дерево, в листьях которого содержаться значения элементов массива, а во всех остальных вершинах – функция, вычисленная на основании значений в детях этой вершины, что позволит экономить время при вычислении функции на отрезке.

Тогда при изменении элемента требуется пересчитывать значения во всех вершинах от одного из листов до корня.

Пример ДО для суммы:



Принцип работы

- Будем хранить массив значений во всех вершинах дерева. Пусть корню соответствует индекс 0, его левому ребёнку индекс 1, а правому – 2. Тогда левый ребёнок вершины 1 – 3, а правый – 4. Т.е. для вершины с индексом i левым ребёнком будет вершина с индексом i*2+1, а для правой – i*2+2.
- Пусть размер исходного массива равен n. Тогда в корне будет храниться значение функции на отрезке [0; n-1], в первой вершине – на отрезке [0, (n-1)/2], во второй – на отрезке [(n-1)/2 + 1; n-1] и т.д.
- Тогда все операции над деревом отрезков можно определить как рекурсивные функции обхода по дереву, запускаемые из корня (вершины 0) с поддержкой того, что в текущей вершине рассматривается значение функции на отрезке [tl; tr], где:
  - для нулевой вершины tl = 0, tr = 1;
  - для левого ребёнка tl’ = tl, tr’ =(tl + tr) / 2;
  - для правого tl’ = (tl + tr) / 2 + 1, tr’ = tr.

Реализация ДО для суммы

```
#define L(i) (i * 2 + 1)
#define R(i) (i * 2 + 2)
#define M(l, r) (l + r) / 2

struct ST
{
    vector<int> t;

    void set(int tl, int tr, int i, int p, int v)
    {
        if (tl > p || tr < p)
            return;
        if (tl == p && tr == p)
        {
            t[i] = v;
            return;
        }
        int m = M(tl, tr);
        set(tl, m, L(i), p, v);
        set(m + 1, tr, R(i), p, v);
        t[i] = t[L(i)] + t[R(i)];
    }

    int get(int tl, int tr, int i, int l, int r)
    {
        if (tl > r || tr < l)
            return 0;
        if (tl == l && tr == r)
            return t[i];
        int m = M(tl, tr);
        return get(tl, m, L(i), l, min(r, m)) +
            get(m + 1, tr, R(i), max(l, m + 1), r);
    }
```

```
    int sz;

    ST(int n)
        :sz(n)
    {
        t.assign(4 * n, 0);
    }

    ST(vector<int>& v)
        :sz(v.size())
    {
        t.resize(4 * sz);
        for (int i = 0; i < sz; ++i)
            set(0, sz - 1, 0, i, v[i]);
    }

    void set(int p, int v)
    {
        set(0, sz - 1, 0, p, v);
    }

    int get(int l, int r)
    {
        return get(0, sz - 1, 0, l, r);
    }
};
```

Реализация ДО для суммы (интерфейсная часть)

Отложенные операции (на примере присвоения на отрезке)

- Пусть дан массив чисел и для него нужно поддерживать операции присвоения на отрезке и получения значения элемента по индексу.
- В листьях дерева по прежнему будем хранить значения элементов массива, а в остальных изначально UNDEF.
- Во время операции присвоения, если отрезок, за который отвечает текущая вершина, целиком покрывается отрезком, на котором происходит присвоение, запомним в этой вершине присваиваемое число и не будем выполнять операцию присвоения у детей.
- Значения в листьях после таких операций присвоения могут оказаться неактуальными. Для исправления этого при рассмотрении вершины во время обхода необходимо «протолкнуть» значение из текущей вершины вниз, то есть если в вершине было значение не UNDEF, то присвоить это значение её детям, а самой вершине присвоить UNDEF.

Реализация ДО с отложенными операциями

```
#define L(i) (i * 2 + 1)
#define R(i) (i * 2 + 2)
#define M(l, r) (l + r) / 2
#define UNDEF -1

struct ST
{
    vint t;

    void push(int ll, int rr, int i)
    {
        if (t[i] == UNDEF || ll == rr)
            return;
        t[L(i)] = t[R(i)] = t[i];
        t[i] = UNDEF;
    }
    void set(int ll, int rr, int i, int l, int r, int v)
    {
        if (ll > r || rr < l)
            return;
        if (ll == l && rr == r)
        {
            t[i] = v;
            return;
        }
        push(ll, rr, i);
        int m = M(ll, rr);
        set(ll, m, L(i), l, min(r, m), v);
        set(m + 1, rr, R(i), max(l, m + 1), r, v);
    }
    int get(int ll, int rr, int i, int p)
    {
        if (ll > p || rr < p)
            return UNDEF;
        push(ll, rr, i);
        if (p == ll && rr == p)
            return t[i];
        int m = M(ll, rr);
        if(p <= m)
            return get(ll, m, L(i), p);
        return get(m + 1, rr, R(i), p);
    }
};
```

Задачи, решаемые деревом отрезков

- Нахождение суммы/минимума/максимума/наибольшего общего делителя/наименьшего общего кратного на отрезке массива.
- Модификация элементов массива на отрезке или в отдельности.
- Поиск k-го нуля в массиве.
- Поиск префикса массива с заданной суммой.
- Поиск подотрезка с максимальной суммой.
- Поиск наименьшего числа, больше либо равного заданного, в указанном отрезке.
- И другие.