ДЕРЕВО ФЕНВИКА

Школа::Кода

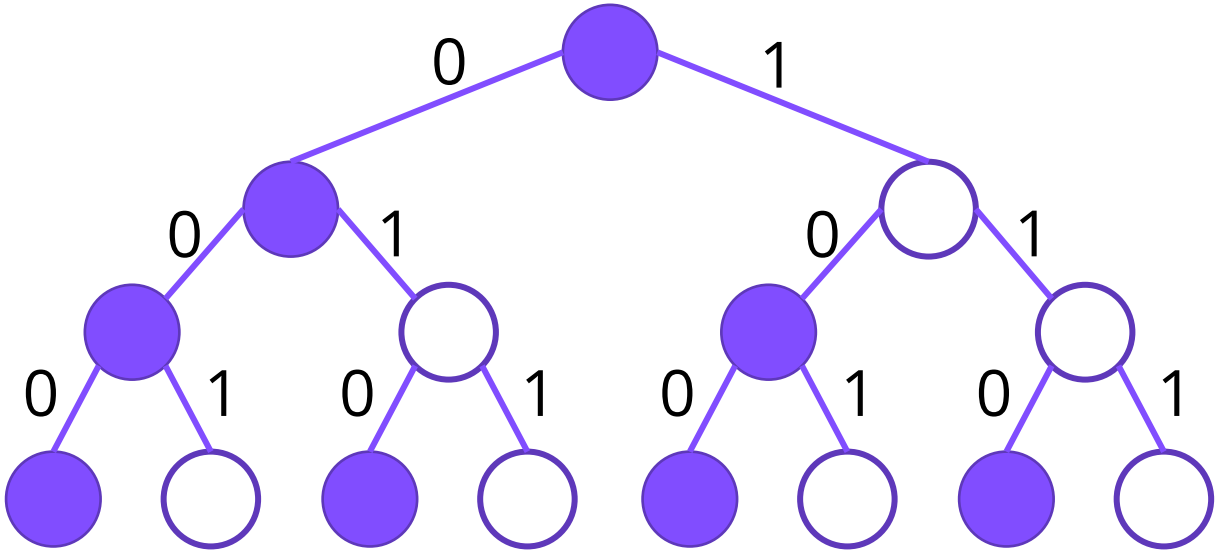
Олимпиадное программирование

2020-2021 Таганрог

Определение

Дерево Фенвика (англ. Binary indexed tree) — структура данных, требующая O(n) памяти и позволяющая эффективно (за O(log(n))) выполнять следующие операции:

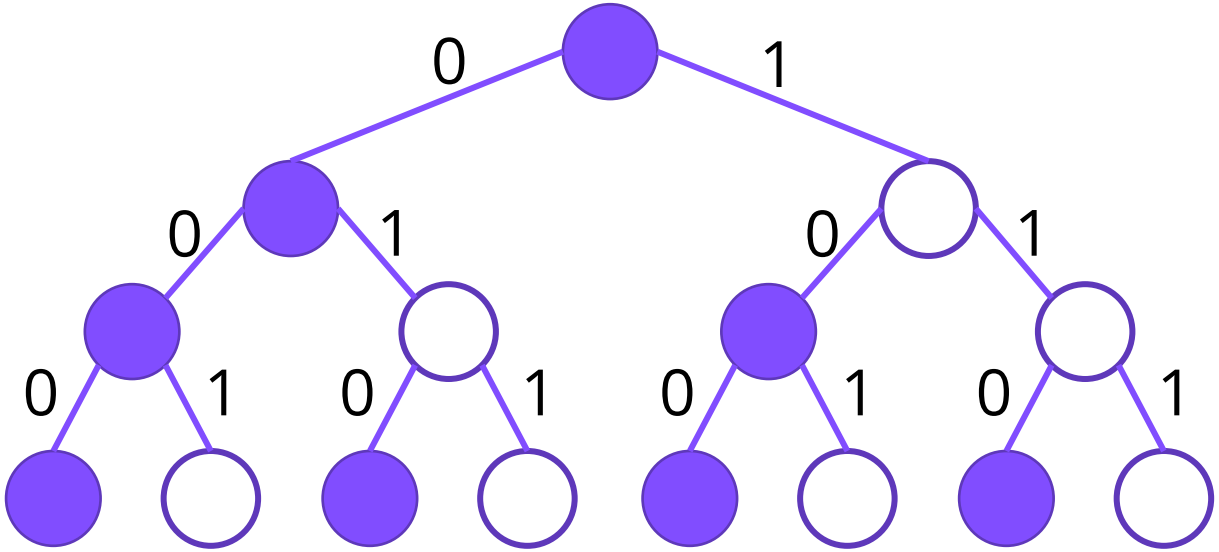
- изменять значение любого элемента в массиве,
- выполнять некоторую ассоциативную, коммутативную, обратимую операцию на отрезке [i,j] (обычно это сумма).



Основная идея

Представим бинарное дерево, в листьях которого хранятся значения исходного массива, а в остальных вершинах – сумма значений в детях этих вершин.

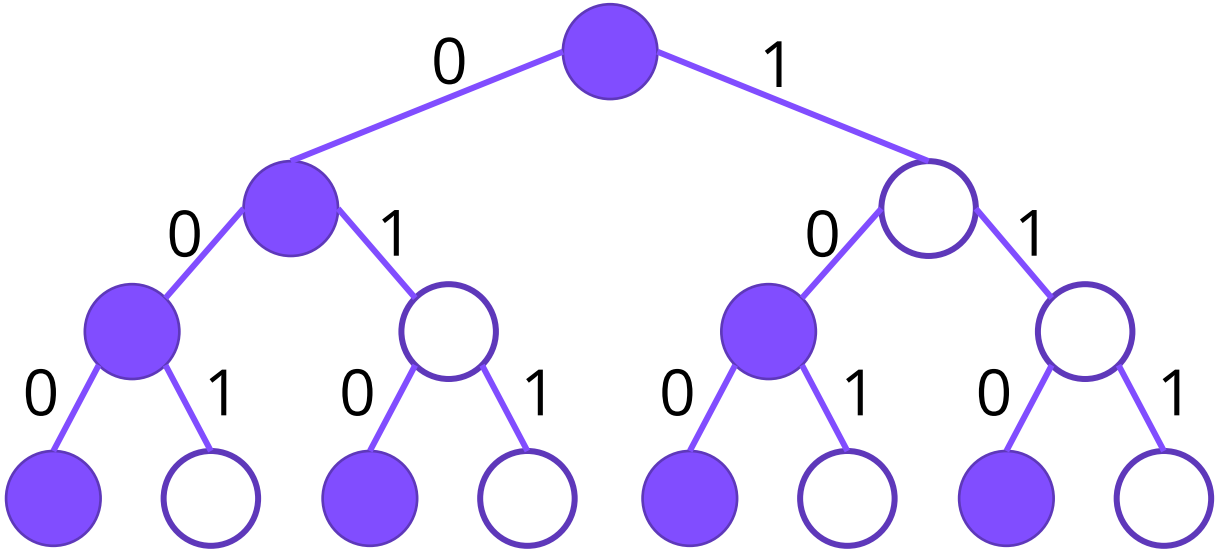
Вместо того, что бы хранить все вершины дерева, будем хранить значения только выделенных вершин. Заметим, что просуммировав значения в определённом наборе закрашенных вершин, можно получить сумму на любом префиксе исходного массива.



Принцип работы

Для массива A будем хранить массив T такой, что T[i] равно сумме элементов массива A на отрезке $[i - 2^k + 1; i]$, где k – это максимальное число, для которого i + 1 делится на 2^k без остатка.

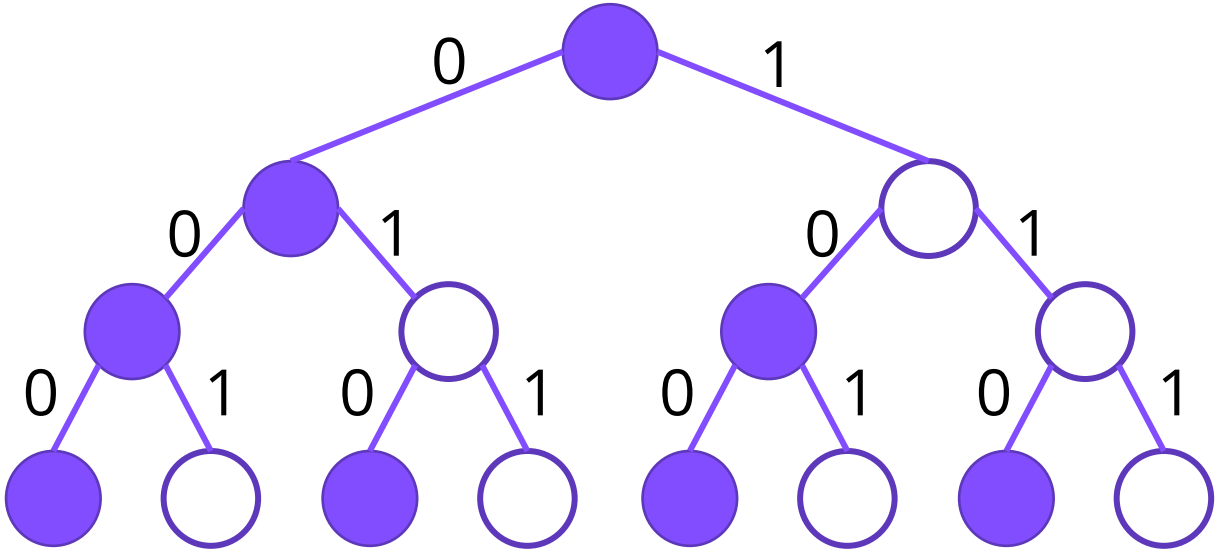
При изменении значения A[i] в массиве T следует изменить все элементы, значение которых учитывает A[i]. Первый элемент будет иметь такой же индекс i, а для последующих вычисляется по формуле $i_{новое} = i_{старое} \mid (i_{старое} + 1)$, пока $i_{новое}$ не станет больше размера массива.



Принцип работы

Для вычисления суммы на отрезке [0; r] массива A нужно сложить элементы массива T, начиная с индекса r, где каждый следующий индекс вычисляется по формуле $i_{новое} = i_{старое} \& (i_{старое} + 1) - 1$, пока $i_{новое}$ не станет меньше 0.

Для вычисления суммы на отрезке [l; r] массива A достаточно вычесть из суммы на отрезке [0; r] сумму на отрезке [0; l – 1], каждую из которых можно вычислить по алгоритму, описанному выше.



Реализация

```
struct FenwickTree
{
    vector<int> tree;
    int size;

    void increase(int index, int value)
    {
        for (int i = index; i < size; i = (i | (i + 1)))
            tree[i] += value;
    }

    int sum(int right)
    {
        int result = 0;
        for (int r = right; r >= 0; r = (r & (r + 1)) - 1)
            result += tree[r];
        return result;
    }
```

Реализация (продолжение)

```
    FenwickTree(int size)
        :size(size)
    {
        tree.assign(size, 0);
    }

    FenwickTree(const vector<int>& initial_array)
    {
        size = initial_array.size();
        tree.assign(size, 0);
        for (int i = 0; i < size; ++i)
            increase(i, initial_array[i]);
    }

    int sum(int left, int right)
    {
        return sum(right) - sum(left - 1);
    }
};
```

Реализация двухмерного случая

```
struct FenwickTree
{
    vector<vector<int>> tree;
    int sizex;
    int sizey;

    void increase(int indexx, int indexy, int value)
    {
        for (int i = indexx; i < sizex; i = (i | (i + 1)))
            for (int j = indexy; j < sizey; j = (j | (j + 1)))
                tree[i][j] += value;
    }

    int sum(int rightx, int righty)
    {
        int result = 0;
        for (int rx = rightx; rx >= 0; rx = (rx & (rx + 1)) - 1)
            for (int ry = righty; ry >= 0; ry = (ry & (ry + 1)) - 1)
                result += tree[rx][ry];
        return result;
    }
```

Задача

- Дан массив целых чисел размером N и Q запросов. Запрос типа 1 прибавляет некоторое X ко всем элементам на отрезке [l; r] ($0 \le l \le r < N$). Запрос типа 0 просит узнать значение элемента с индексом id ($0 \le id < N$) после всех предыдущих запросов.

- Решение:
  Заведём массив B размера N + 1 для хранения изменений, изначально заполненный нулями.
  При получении запроса прибавления, будем прибавлять X к элементу массива B с индексом l и отнимать X от элемента того же массива с индексом r + 1.
  При получении запроса на значение элемента с индексом id будем прибавлять к начальному значению этого элемента сумму первых id элементов из массива B.

```
int n, q;
cin >> n >> q;
vector<int> a(n);
for (int i = 0; i < n; ++i)
    cin >> a[i];
vector<int> b(n + 1, 0);
for (int i = 0; i < q; ++i)
{
    int t;
    cin >> t;
    if (t)
    {
        int l, r, x;
        cin >> l >> r >> x;
        b[l] += x;
        b[r + 1] -= x;
    }
    else
    {
        int id;
        cin >> id;
        int s = 0;
        for (int j = 0; j <= id; ++j)
            s += b[j];
        cout << a[id] + s << '\n';
    }
}
```

Реализация за $O(N^2)$

Задача

- Необходимо посчитать количество инверсий в массиве A. Количество инверсий – это количество таких пар чисел i и j, что i < j и A[i] > A[j].
- Решение:
  Заведём массив B размера A_{max} + 1, где A_{max} равно максимально возможному значению элемента массива A, и заполним его нулями. Переберём все элементы массива A. Для каждого элемента A[i] будем проставлять значение B[A[i]] равное 1. Тогда при рассмотрении A[i] количество элементов A[j] таких, что i > j и A[i] < A[j], будет равно сумме элементов массива B на отрезке [A[i] + 1; A_{max}], что является количеством инверсий, в которых меньший элемент – A[i]. Сумма этих сумм и будет являться количеством инверсий всего массива A.

```cpp
int n;
cin >> n;
vector<int> b(1e6 + 1, 0);
int res = 0;
for (int i = 0; i < n; ++i)
{
    int a;
    cin >> a;
    for (int j = a + 1; j < b.size(); ++j)
        res += b[j];
    b[a] = 1;
}
cout << res;
```

Реализация за $O(N^2)$