Санкт-Петербургский политехнический университет Петра Великого

Институт металлургии, машиностроения и транспорта

Высшая школа автоматизации и робототехники

**КУРСОВАЯ РАБОТА**

**Дерево Фенвика**

по дисциплине «Объектно-ориентированное программирование»

Выполнил

студент гр. 3331506/70401 Е.И. Чернов

Руководитель М.С. Ананьевский

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Санкт-Петербург

2020

Содержание

[1. Введение 4](#_Toc41297175)

[2. Описание алгоритма 5](#_Toc41297176)

[3. Реализация алгоритма 7](#_Toc41297177)

[3.1. Реализация дерева Фенвика для суммы для одномерного случая 7](#_Toc41297178)

[3.2. Реализация дерева Фенвика для минимума для одномерного случая 8](#_Toc41297179)

[3.3. Реализация дерева Фенвика для суммы для двумерного случая 9](#_Toc41297180)

# Введение

Дерево Фенвика — это структура данных, дерево на массиве, обладающее следующими свойствами:

1. позволяет вычислять значение некоторой обратимой операции G на любом отрезке [*L*; *R*] за время *O* (*log* *N*);
2. позволяет изменять значение любого элемента за *O* (*log* *N*);
3. требует *O* (*N*) памяти, а точнее, ровно столько же, сколько и массив из *N* элементов;
4. легко обобщается на случай многомерных массивов.

Наиболее распространённое применение дерева Фенвика — для вычисления суммы на отрезке, т.е. функция *G (X1, ..., Xk) = X1 + ... + Xk*.

Дерево Фенвика было впервые описано в статье "*A new data structure for cumulative frequency tables*" (*Peter M. Fenwick*, 1994).

# Описание алгоритма

Для простоты описания мы предполагаем, что операция *G*, по которой мы строим дерево, — это сумма.

Пусть дан массив *A* [0…*N*-1]. Дерево Фенвика — массив *T* [0...*N*-1], в каждом элементе которого хранится сумма некоторых элементов массива A:

где *F(i)* — некоторая функция, которую мы определим несколько позже.

Теперь мы уже можем написать псевдокоддля функции вычисления суммы на отрезке [0; *R*] и для функции изменения ячейки:

int sum (int r)

{

int result = 0;

while (r >= 0) {

result += t[r];

r = f(r) - 1;

}

return result;

}

void inc (int i, int delta)

{

для всех j, для которых F(j) <= i <= j

{

t[j] += delta;

}

}

Функция *sum* работает следующим образом. Вместо того чтобы идти по всем элементам массива *A*, она движется по массиву *T*, делая "прыжки" через отрезки там, где это возможно. Сначала она прибавляет к ответу значение суммы на отрезке [*F*(*R*); *R*], затем берёт сумму на отрезке [*F*(*F*(*R*)-1); *F*(*R*)-1], и так далее, пока не дойдёт до нуля.

Функция *inc* движется в обратную сторону — в сторону увеличения индексов, обновляя значения суммы *Tj* только для тех позиций, для которых это нужно, т.е. для всех *j*, для которых *F(j) <= i <= j*.

Очевидно, что от выбора функции *F* будет зависеть скорость выполнения обеих операций. Сейчас мы рассмотрим функцию, которая позволит достичь логарифмической производительности в обоих случаях.

Определим значение *F(X)*следующим образом. Рассмотрим двоичную запись этого числа и посмотрим на его младший бит. Если он равен нулю, то *F(X) = X*. Иначе двоичное представление числа *X* оканчивается на группу из одной или нескольких единиц. Заменим все единицы из этой группы на нули, и присвоим полученное число значению функции *F(X)*.

Этому довольно сложному описанию соответствует очень простая формула:

где *&* — это операция побитового логического "И".

Нетрудно убедиться, что эта формула соответствует словесному описанию функции, данному выше.

Нам осталось только научиться быстро находить такие числа *j*, для которых *F(j) <= i <= j*.

Однако нетрудно убедиться в том, что все такие числа *j* получаются из *i* последовательными заменами самого правого (самого младшего) нуля в двоичном представлении. Например, для *i* = 10 мы получим, что *j* = 11, 15, 31, 63 и т.д. Как ни странно, такой операции (замена самого младшего нуля на единицу) также соответствует очень простая формула:

где | — это операция побитового логического "ИЛИ".

# Реализация алгоритма

## Реализация дерева Фенвика для суммы для одномерного случая

vector<**int**> t;  
**int** n;  
  
**void** init(**int** nn) {  
 n = nn;  
 t.assign(n, 0);  
}  
  
**int** sum(**int** r) {  
 **int** result = 0;  
 **for** (; r >= 0; r = (r & (r + 1)) - 1) result += t[r];  
 **return** result;  
}  
  
**void** inc(**int** i, **int** delta) {  
 **for** (; i < n; i = (i | (i + 1))) t[i] += delta;  
}  
  
**int** sum(**int** l, **int** r) {  
 **return** sum(r) - sum(l - 1);  
}  
  
**void** init(vector<**int**> a) {  
 init((**int**) a.size());  
 **for** (**unsigned** i = 0; i < a.size(); i++) inc(i, a[i]);  
}

## Реализация дерева Фенвика для минимума для одномерного случая

Следует сразу заметить, что, поскольку дерево Фенвика позволяет найти значение функции в произвольном отрезке [0; *R*], то мы никак не сможем найти минимум на отрезке [*L*; *R*], где *L* > 0. Далее, все изменения значений должны происходить только в сторону уменьшения (опять же, поскольку никак не получится обратить функцию *min*).

Это значительные ограничения.

vector<**int**> t;  
**int** n;  
  
**const int** INF = 1000 \* 1000 \* 1000;  
  
**void** init(**int** nn) {  
 n = nn;  
 t.assign(n, INF);  
}  
  
**int** getmin(**int** r) {  
 **int** result = INF;  
 **for** (; r >= 0; r = (r & (r + 1)) - 1) result = min(result, t[r]);  
 **return** result;  
}  
  
**void** update(**int** i, **int** new\_val) {  
 **for** (; i < n; i = (i | (i + 1))) t[i] = min(t[i], new\_val);  
}  
  
**void** init(vector<**int**> a) {  
 init((**int**) a.size());  
 **for** (**unsigned** i = 0; i < a.size(); i++) update(i, a[i]);  
}

## Реализация дерева Фенвика для суммы для двумерного случая

Как уже отмечалось, дерево Фенвика легко обобщается на многомерный случай.

vector <vector<**int**>> t;  
**int** n, m;  
  
**int** sum(**int** x, **int** y) {  
 **int** result = 0;  
 **for** (**int** i = x; i >= 0; i = (i & (i + 1)) - 1)  
 **for** (**int** j = y; j >= 0; j = (j & (j + 1)) - 1) result += t[i][j];  
 **return** result;  
}  
  
**void** inc(**int** x, **int** y, **int** delta) {  
 **for** (**int** i = x; i < n; i = (i | (i + 1)))  
 **for** (**int** j = y; j < m; j = (j | (j + 1))) t[i][j] += delta;  
}

# Анализ

# Заключение

# Список литературы

1. Роналд Грэхем, Дональд Кнут, Орен Паташник. Конкретная математика. Основание информатики. —М.: Высшая школа, 1986. — С. 298-310.