Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Объектно-ориентированное программирование
Тема: Дерево Фенвика

Студент гр. 3331506/00401 Зорькин Е. Р.
Преподаватель Ананьевский М.С.

Санкт-Петербург
2023

Оглавление

Введение 3
Описание алгоритма 9
Реализация алгоритма 11
Анализ 13

Введение

Цель этой работы раскрыть такую структуру данных как дерево Фенвика или как его второе название двоичное индексированное дерево (Binary indexed tree). Дерево Фенвика далеко не распространая структура данных, так как ей пользуются для решения весьма специфичных задач, а именно чаще всего для нахождения суммы подряд идущих элементов, как от начала массива до необходимого элемента, так и суммы нахождения на любом отрезке элементов массива.

Для рассмотрения преимуществ дерева Фенвика предлагаю просмотреть на реализацию ранее описанной задачи нахождения суммы элементов в массиве А через массив сумм Б, элементы которого будут содержать в себе последовательное суммирование каждого элемента массива А. Изобразим данные массивы на рисунке 1. Также для наглядности нарисуем график на рисунке 2, в котором по оси x индекс элемента массива, по оси y будут закрашены те индексы просуммировав которые получим значение элемента в массиве по индексу указанному в оси x.

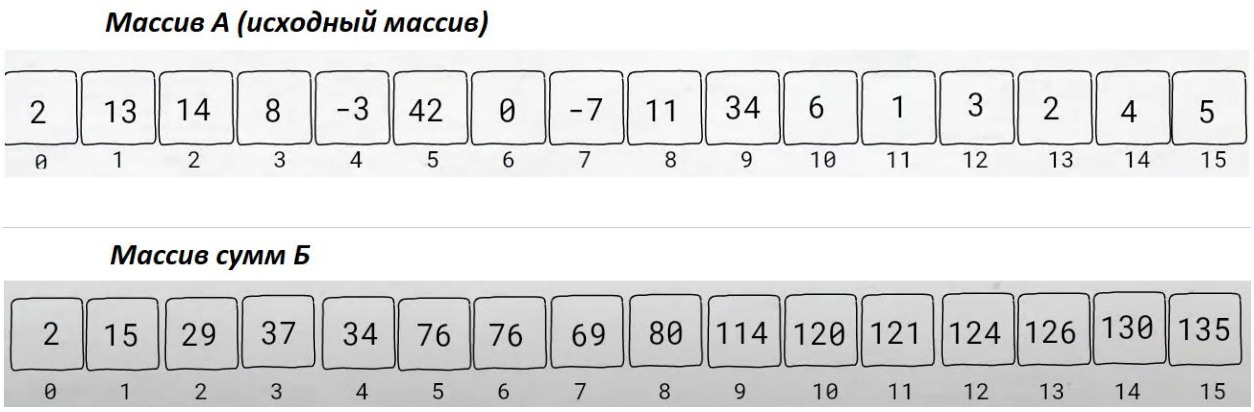

Рис. 1 Представление массивов А и Б

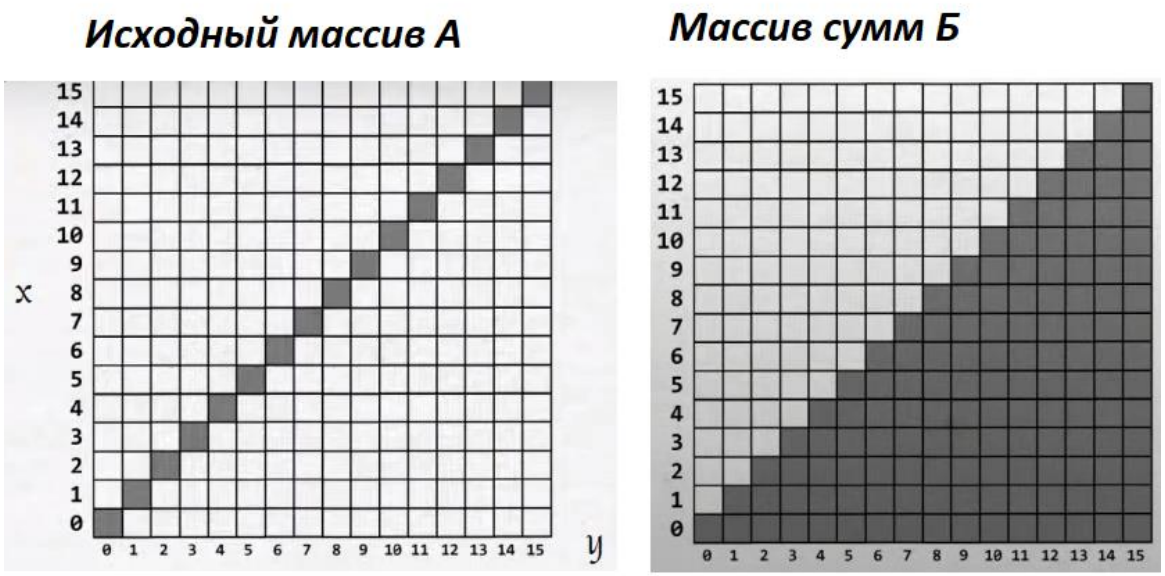

Рис. 2 Графическое представление массивов А и Б

Как можно заметить в 3 элементе (ячейке) массива Б находится сумма элементов массива А от 0 до 3, то есть $2 + 13 + 14 + 8 = 37$, кроме того, на рисунке 2 отображено, что 3 элемент массива Б содержит в себе сумму от 0 до 3 элемента массива А. При этом исходные данные массива А в элементах не пропадают, а остаются в виде разности элементов в массиве Б.

Например, при нахождении элемента 1 исходного массива А в массиве Б, необходимо взять разность в массиве Б от 1 элемента вычитая его 0 элемент.

Операция по нахождении суммы элементов А от элемента $l = 5$ до $r = 7$ находится также разностью элементов в массиве Б путем вычитания данных в 7 элементе из 4 элемента.

Однако оба массива обладают недостатками для решения задачи нахождения суммы элементов. Массив А будет обладает высокой (постоянная сложность) скоростью обновления данных в массиве, но медленно (линейная сложность) считать сумму своих элементов. Массив Б наоборот, быстро (постоянная сложность) посчитает сумму, но его сложнее обновить (линейная сложность обновления). Таким образом при подсчете и обновлении больших массивов, оба массива не годятся, но с этими задачами справляется массив Фенвика или же дерево Фенвика. Дерево Фенвика позволяет проводить

операции сложения и обновления элементов за логарифмическую сложность, например суммирование 1 миллиарда элементов займет около 30 операций. Итоги данного рассуждения приведены в таблице 1.

Таблица 1

	Обновление заданного элемента	Нахождение суммы элементов
Массив А (исходный массив)	$0(1)$	$0(n)$
Массив Б (массив сумм)	$0(n)$	$0(1)$
Дерево Фенвика	$0(\log(n))$	$0(\log(n))$

Четные элементы в дереве Фенвика хранят элементы исходного массива, а также дерево Фенвика, аналогично массиву Б хранит сумму элементов исходного массива А, однако есть элементы не только от 0 до текущего элемента, а также присутствуют от определенного элемента до текущего элемента. Эти правила в примере реализованы на рисунке 3.

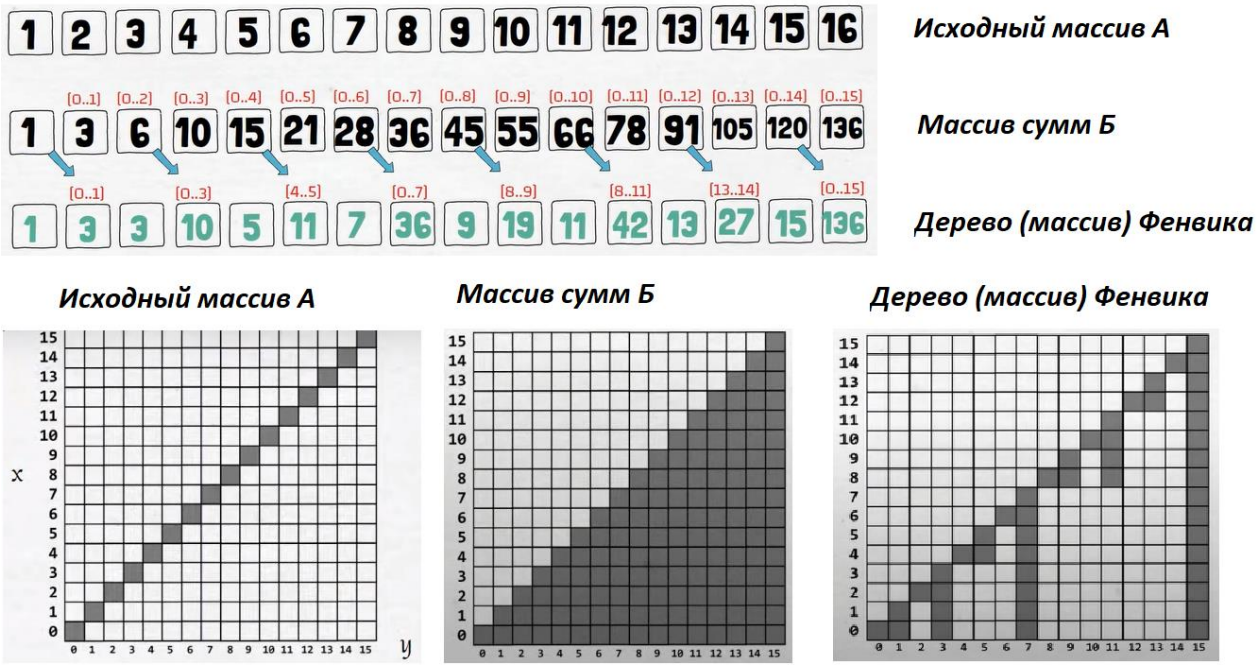

Рис. 3 Сравнение массивов А, Б и дерева Фенвика

При построении дерева Фенвика использую формулу определяющая какую сумму элементов хранит $f[i]$. Или с какого элемента исходного массива A и до текущего необходимо хранить в $f[i]$. Данная формула приведена ниже:

$$f[i] = i \,\&\, (i+1), \text{ где } \& \text{ – операция побитового И}$$

Для того, чтобы окончательно ответить на вопрос – от какого элемента $f[i]$ необходимо начинать проводить суммирование для построения дерева Фенвика построим визуальную модель на рисунке 4 и определим закономерность на примере нахождения 11 элемента. Или, иными словами, от какого элемента исходного массива A и до текущего $f[11] = f[i]$ необходимо проводить суммирование для построения $11 = i$ элемента дерева Фенвика.

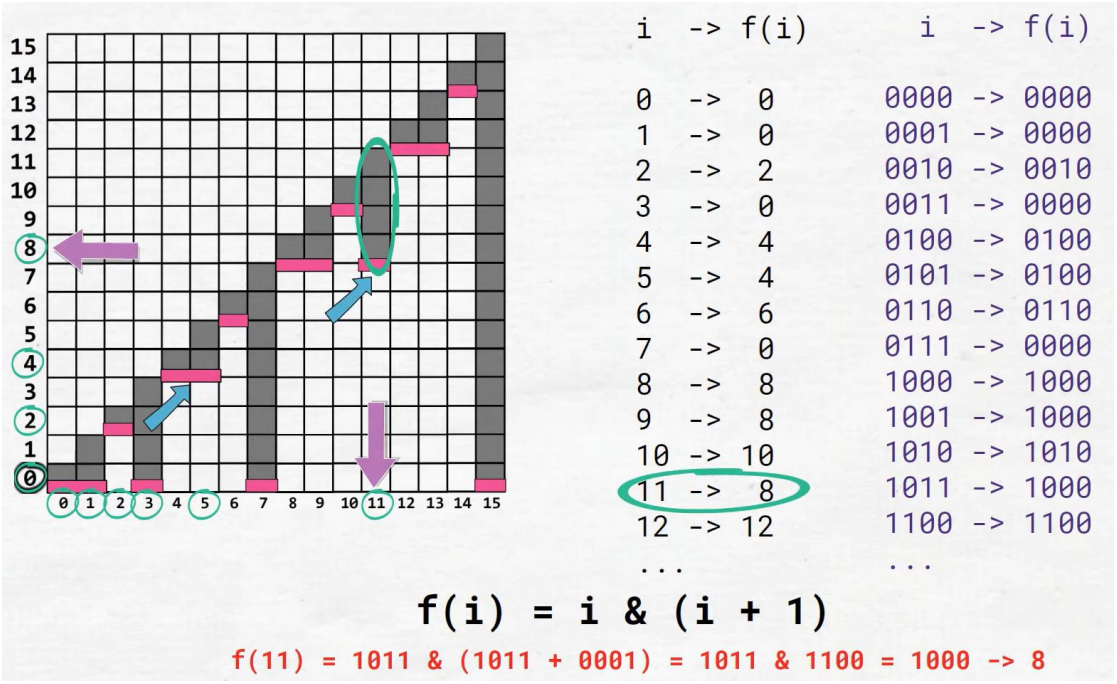

Рис. 4 Пример нахождения номера исходного элемента массива A до текущего для построения 11 элемента дерева Фенвика

После обновления значения исходного элемента i в массиве A, необходимо обновить элемент i в массиве (дереве) Фенвика, затем по формуле:

$$f[i] = i \mid (i+1), \text{ где } \mid \text{ – операция побитового ИЛИ}$$

вычислить индекс следующего элемента для обновления в дереве Фенвика и повторять данную процедуру.

Предлагаю разобраться на примере. Итак, в исходном массиве A состоящий из 15 элементов поменяли данные в 8 элементе и теперь, для обновления дерева Фенвика необходимо – обновить элемент 8 по формуле:

$$f[i] = i \mid (i+1), \text{ где } \mid \text{ – операция побитового ИЛИ,}$$

$$f[8] = 8 \mid (8+1) = 1000 \mid (1001) = 9$$

которая позволяет найти индекс следующего 9 элемента и продолжить повторение процедуры до окончания дерева (массива) Фенвика. Визуализация примера представлена на рисунке 5.1.

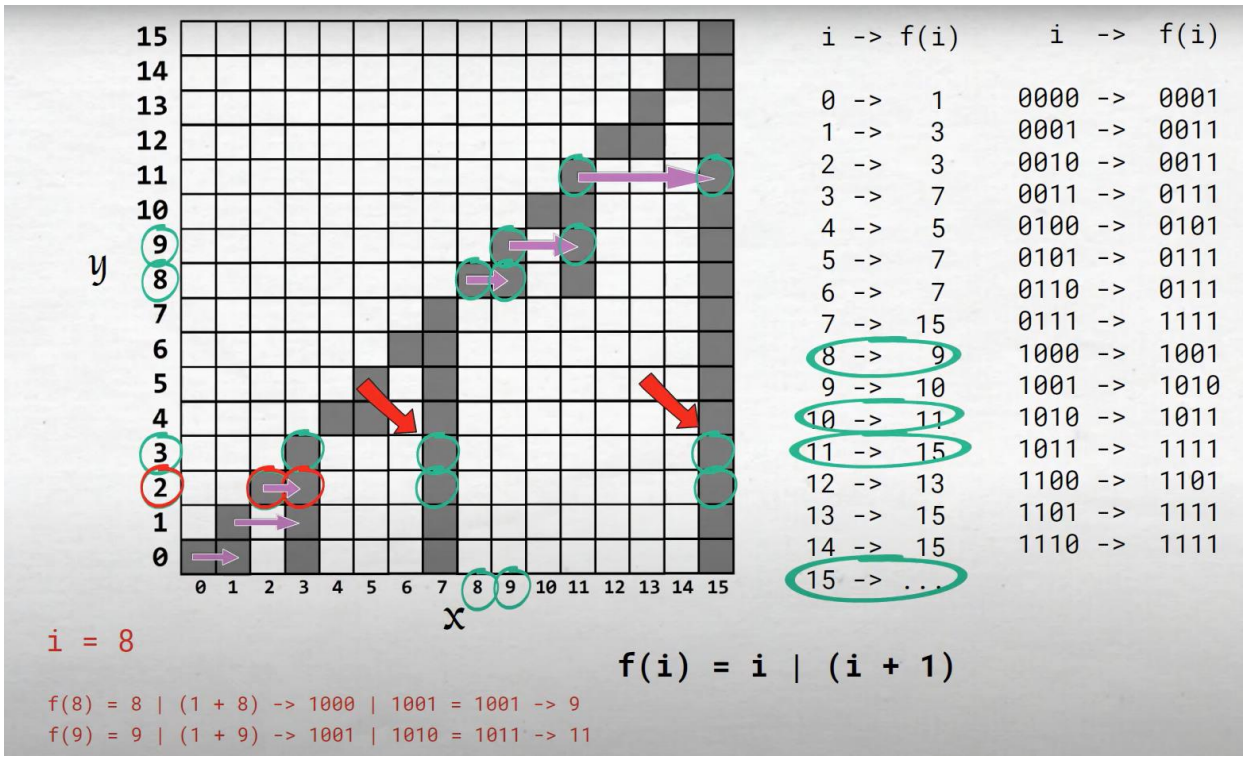

Рис. 5.1 Пример обновления дерева Фенвика

Подведем итоги главных преимуществ и особенностей дерева Фенвика:

- позволяет вычислять значение некоторой обратимой операции G на любом отрезке $[l:r]$ за время $0(\log(n))$
- позволяет изменять значение любого элемента за $0(\log(n))$
- требует $0(n)$ памяти, а точнее, ровно столько же, сколько и массив из n элементов

Графическое изображение дерева Фенвика представлено на рисунке 5.2.

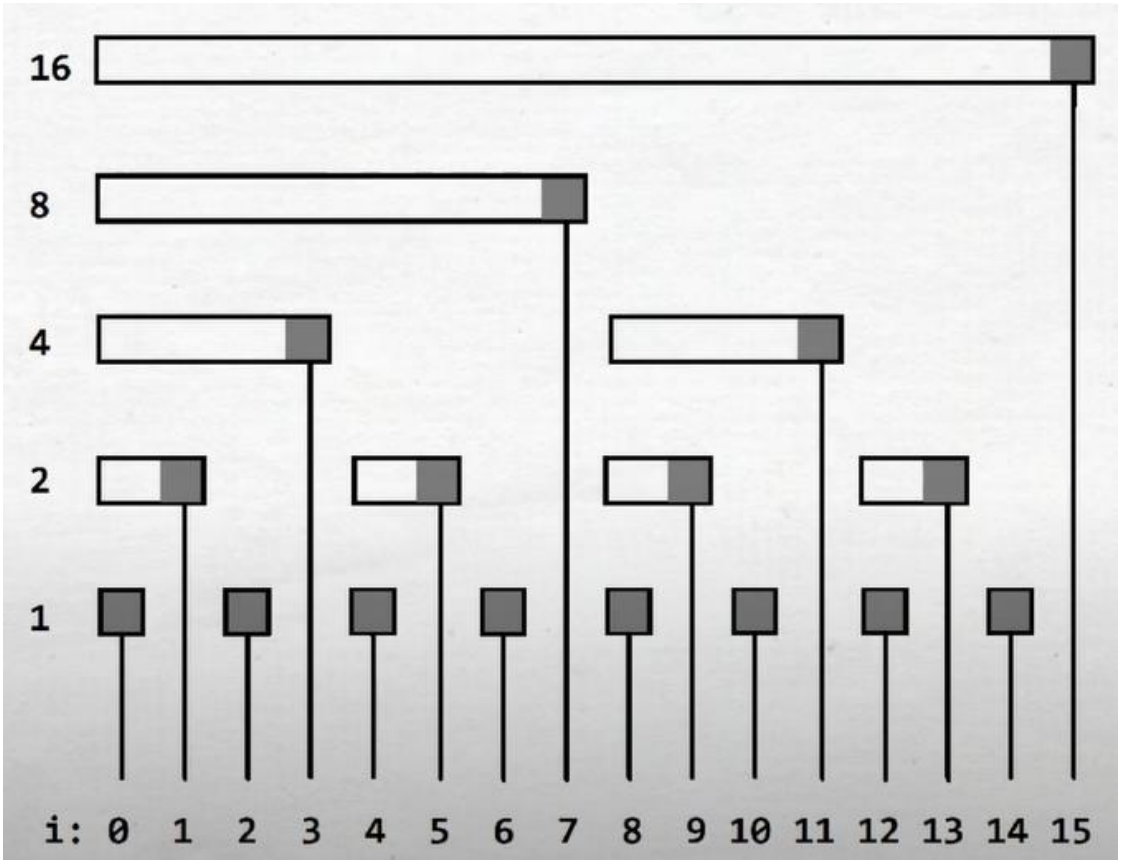

Рис. 5.2 Пример обновления дерева Фенвика

Описание алгоритма

Для простоты описания алгоритма предполагается, что операция G является суммирующей, по которой мы строим дерево.

Дано массив $a[0 \ldots N-1]$, и максимальный размер массива $[N]$, тогда дерево Фенвика — массив $f[0 \ldots N-1]$, в каждом элементе которого хранится сумма некоторых элементов массива a: Ti =сумма Aj для всех $F(i) <= j <= i$, где *F(i)* — некоторая функция, от которой зависит скорость работы.

Начнем с описание функции суммирования от 0 элемента и до текущего представленного на рисунке 6.

```
// Функция нахождения суммы элементов от начала до элемента => на промежутке [0:x]
// Реализация формулы F(x) = x and (x+1) => (& - все 0 кроме 1&1=1)
int getSumTo(int x) {
    int result = 0;

    for (; x >= 0; x = (x & (x + 1)) - 1) {
        result += f[x];
    }

    return result;
}
```

Рис. 6 Функция *getSumTo*

Функция *getSumTo* работает следующим образом: вместо того чтобы идти по всем элементам массива a, она движется по массиву *f*, делая "прыжки" через отрезки. 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 *update* движется в обратную сторону — в сторону увеличения индексов, обновляя значения суммы *fj* только для тех позиций, для которых это нужно. Описание функции обновления элементов в дереве Фенвика представлено на рисунке 7.

```
// Функция обновления значения элемента массива
// H(x) = x or (x+1) => (| - все 1 кроме 0|0=0)
//увеличение a[idx] на delta (обновление ячейки массива)
void update(int idx, int delta) {
    a[idx] += delta;

    for (; idx < n; idx |= idx + 1) {
        f[idx] += delta;
    }
}
```

Рис. 7 Функция *update*

Функция *getSumFromTo* является логическим продолжением функции *getSumT*, так как для выполнения задачи нахождения суммы на отрезке от $[l:r]$ необходимо произвести разность сумм, записанных в данных элементах. Описание функции суммирования элементов на отрезке в дереве Фенвика представлено на рисунке 8.

```
// Функция нахождения суммы элементов на промежутке [l:r]
int getSumFromTo(int l, int r) {
    if (l) {
        return getSumTo( x: r) - getSumTo( x: l - 1);
    } else {
        return getSumTo( x: r);
    }
}
```

Рис. 8 Функция *getSumFromTo*

Также была добавлена функция вывода дерева Фенвика *printFenv* в консоль пользователя для проверки работоспособности кода, показана на рисунке 9.

```
// Функция вывода дерева (массива) Фенвика
void printFenv() {
    for (auto& el : int & : f) {
        cout << el << " ";
    }
    cout << "\n";
}
```

Рис. 9 Функция *printFenv*

Реализация алгоритма

```
/// Реализация дерева Фенвика

#include ...

using namespace std;

const int N = 10;
int n;
int a[N];  //массив
int f[N];  //дерево Фенвика

// Функция нахождения суммы элементов от начала до элемента => на промежутке [0:x]
// Реализация формулы F(x) = x and (x+1) => (& - все 0 кроме 1&1=1)
int getSumTo(int x) {...}

// Функция обновления значения элемента массива
// H(x) = x or (x+1) => (| - все 1 кроме 0|0=0)
//увеличение a[idx] на delta (обновление ячейки массива)
void update(int idx, int delta) {...}

// Функция нахождения суммы элементов на промежутке [l:r]
int getSumFromTo(int l, int r) {...}

// Функция вывода дерева (массива) Фенвика
void printFenv() {...}


int main() {...}
```

Рис. 10 Код для построения дерева Фенвика

Проверка работоспособности проекта представлена на рисунке 11 телом функции main и 12 выводом консоли.

```cpp
int main() {
    /// ^^^^ ввод массива и заполнение дерева Фенвика состоят из 2 блоков
    cout << "Enter the number of elements in the array => ";
    cin >> n;                          // ввод пользователем количества элементов в новом массиве

    for (int i = 0; i < n; i++) {
        int t;
        cout << "Enter an element in an array => ";
        cin >> t;                      // ввод пользователем по порядку каждого элемента нового массива
        update( idx: i,  delta: t);
    }


    /// Обработка запросов для удовстоверенности работоспособности кода

    cout << "\nBuilding a Fenwick Tree => " << endl;
    printFenv(); // вывод дерева Фенвика из заданного массива

    cout << "\nSumma from 0 and up to #2 elemenents => " << getSumTo( x: 2) << endl; // проверка функции суммы от 0 до x

    cout << "\nSumma from #2 and up to #4 elements => " << getSumFromTo( l: 2,  r: 4) << endl;


    cout << "\nBuilding a Fenwick Tree before changes => " << endl;
    printFenv(); // вывод дерева Фенвика ДО зменения
    cout << "\nUpdate elements #0 => delta (-6) \n" << endl;
    update( idx: 0,  delta: -6);
    cout << "\nBuilding a Fenwick Tree after changes => " << endl;
    printFenv(); // вывод дерева Фенвика ПОСЛЕ зменения
}
```

Рис. 11 Описание для проверки проекта

```
Enter the number of elements in the array => 5
Enter an element in an array => 1
Enter an element in an array => 2
Enter an element in an array => 3
Enter an element in an array => 4
Enter an element in an array => 5

Building a Fenwick Tree =>
1 3 3 10 5 0 0 0 0 0

Summa from 0 and up to #2 elemenents => 6

Summa from #2 and up to #4 elements => 12
```

```
Building a Fenwick Tree before changes =>
1 3 3 10 5 0 0 0 0 0

Update elements #0 => delta (-6)


Building a Fenwick Tree after changes =>
-5 -3 3 4 5 0 0 0 0 0

Process finished with exit code 0
```

Рис. 12 Вывод консоли для проверки

Список литературы

1. Роналд Грэхем, Дональд Кнут, Орен Паташник. Конкретная математика. Основание информатики. — М.: Высшая школа, 1986. — С. 298-310.
2. Rodica Ceterchi, Mircea Dimama. Efficient Range Minimum Queries using Binary Indexed Trees.
3. Девятериков Иван. Структура данных Дерево отрезков и её применение в задачах.