ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ **«КАЗАНСКИЙ(ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

ОТЧЕТ ПО ТЕМЕ:

**«ДЕРЕВО ФЕНВИКА»**

Выполнила: Федорова Елена, гр. 11-002

**Казань 2021г.**

**Краткая историческая справка**

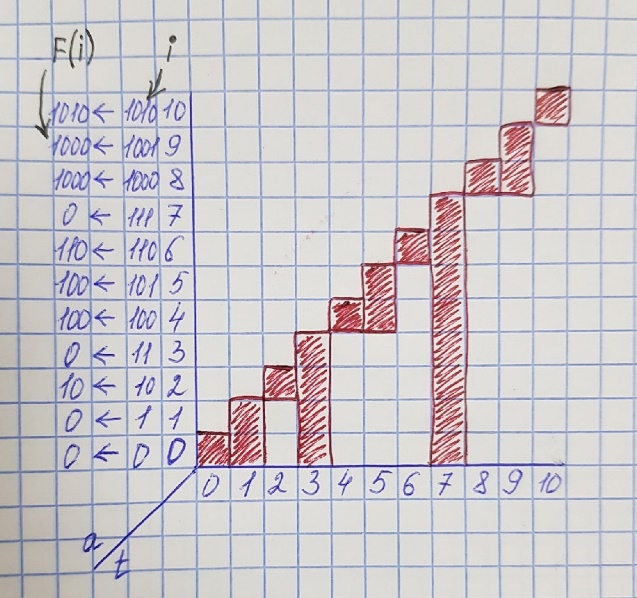
Дерево Фенвика (Бинарное Индексированное Дерево) – структура данных, которая была описана Питером Фенвиком в 1994 году. За основу данной структуры берется массив, элементами которого являются суммы подотрезков исходного массива. Данная структура примечательна тем, что она позволяет нам вычислять сумму на некотором отрезке и изменять значение некоторого элемента за логарифмическое время O(log(N)). Также, эта структура требует лишь для хранения O(N) памяти.

**Основной принцип устройства. Особенности**

Дерево Фенвика содержит в качестве элементов сумму подотрезков исходного массива в диапазоне [F(i); i + 1] (i – индекс элемента дерева). Ключевой особенностью данного алгоритма является функция F(i) = i & (i+1). Данное побитовое выражение обнуляет последнюю группу единиц индекса в двоичном представлении. Благодаря данной операции мы можем в последующем вычислять суммы подотрезков массива за O(log(N)).

Также следует сказать об изменении элемента. Данная операция выполняется за логарифмическое время O(log(N)), используя выражение i | (i+1). Данное побитовое выражение также используется для индекса, и в отличие от F(i), работает в обратную сторону: добавляет единицу в двоичном представлении индекса.

**Оценка временной сложности**



На данном рисунке схематично изображена структура дерева Фенвика. Снизу представлены индексы дерева (массив Т), а слева – исходный массив A.

Операция вычисления суммы (префиксной) выполняется за O(log(n)):

Для массива данная задача выполняется за O(n). Пусть нам необходимо найти префиксную сумму 5 элементов. В данном случае сумма есть: t[4:4] + t[0:4] (также видно из рисунка). В данном случае, из 5-ти элементов для вычисления суммы, нам потребовались 2. Рассмотрим 11 элементов: t[10:10] + t[8:10] + t[0:8]. Из 11 элементов нам потребовалось лишь 3 элемента для вычисления префиксной суммы. За каждое применение функции f(i) уменьшается количество единиц в двоичной записи i как минимум на 1. Из чего следует, что подсчет суммы будет произведен за O(log(N)).

Операция изменения элемента дерева выполняется за O(log(n)):

Для массива данная задача выполняется за O(1). Для изменения элемента дерева нам необходимо изменить те суммы, в которых находится данный элемент. Для этого используется функция, обратная f(i), которая равняется i | (i + 1). Данная функция возвращает каждый раз индекс суммы подотрезка, в котором находится изменяемый нами элемент. Несложно заметить, что данная функция строго возрастает и в худшем случае будет выполнена за O(log(N)), так как она добавляет каждый раз по одной единице в двоичном разложении числа i.

**Графики зависимости времени (в секундах) от размера входных данных основных функций структуры данных**

В исследовании проводились измерения времени алгоритмов со 100 наборами от 157 до 254 элементов.

Данные наблюдения показывают зависимость времени выполнения работы алгоритмов структуры данных от размера входных данных. Все данные алгоритмы (за исключением initFenwickTreeFromArray()) выполняются за логарифмическое время.

Функция initFenwickTreeFromArray() выполняется за O(N\*log(N)). Из графика мы можем увидеть некоторую зависимость от размера входных данных. Видно, что есть стремление вверх, то есть необходимо больше времени на выполнение в зависимости от размера входных данных.

Функции addToElement(), prefixSum(), ElemByIdx(), sumSlice() выполняются за логарифмическое время. Данные функции при работе c нашими наборами выдавали время 1-2 микросекунды (в среднем 1.5 микросекунды). Мы не увидели явной зависимости скорости работы при увеличении из-за недостаточного объема входных данных и также из-за того, что алгоритм является достаточно быстрым.

**Выводы**

Данная структура данных позволяет достаточно быстро находить сумму элементов исходного массива и модифицировать ее элемент. Данная структура будет полезна тогда, когда есть необходимость в быстром изменении и получении результатов.

Преимущества:

- Поиск суммы и изменение элемента происходит за время O(log(N))

- Занимает O(N) памяти

Недостатки:

- Данная структура не позволяет производить поиск минимума и максимума

**Список использованной литературы(источники)**

- «Лекция. Дерево Фенвика» <https://www.youtube.com/watch?v=BzFN9YwR-NM&t=5290s>

- «Хабр. Дерево Фенвика» <https://habr.com/ru/post/112828/>

- «Викиконспекты. Дерево Фенвика» <https://neerc.ifmo.ru/wiki/index.php?title=Дерево_Фенвика>

**Код программы.**

#include <iostream>

using namespace std;

void addToElement(int T[], int n, int idx, int value);

int prefixSum(int T[], int r);

int ElemByIdx(int T[], int idx);

void changeValue(int T[], int n, int idx, int value);

int sumSlice(int T[], int idx1, int idx2);

int \*initFenwickTreeFromArray(int a[], int n);

int \*createFenwickTree(int n, int startValue = 0);

int main() {

    int n = 12;

    int \*t1 = createFenwickTree(n);

    for (int i = 0; i < n; i++) {

        changeValue(t1, n, i, i);

    }

    cout << "(T): \t\t";

    for (int i = 0; i < n; i++) {

        cout << t1[i] << " ";

    }

    cout << endl << "(T -> A): \t";

    for (int i = 0; i < n; i++) {

        cout << ElemByIdx(t1, i) << " ";

    }

    cout << endl << "Output of sumSlice(T1, 3, 6): " << sumSlice(t1, 3, 6) << endl;

   int arr[] = {12, 3, 34, 42, 123, 23, 79};

   int size = sizeof(arr)/sizeof(arr[0]);

   int \*t2 = initFenwickTreeFromArray(arr, size);

   cout << "Idx: \t\t";

   for (int i = 0; i < size; i++) {

       cout << i << " ";

   }

   cout << endl << "A: \t\t";

   for (int i = 0; i < size; i++) {

       cout << arr[i] << " ";

   }

   cout << endl << "T: \t\t";

   for (int i = 0; i < size; i++) {

       cout << t2[i] << " ";

   }

   cout << endl << "T -> A: \t";

   for (int i = 0; i < size; i++) {

       cout << ElemByIdx(t2, i) << " ";

   }

   changeValue(t2, size, 2, 5);

   cout << endl << "Modified A: \t";

   for (int i = 0; i < size; i++) {

       cout << ElemByIdx(t2, i) << " ";

   }

   cout << endl << "Modified T: \t";

   for (int i = 0; i < size; i++) {

       cout << t2[i] << " ";

   }

}

void addToElement(int T[], int n, int idx, int value) {

    while (idx < n) {

        T[idx] += value;

        idx = idx | (idx + 1);

    }

}

int prefixSum(int T[], int r) {

    int result = 0;

    do {

        result += T[r];

        r = ((r + 1) & r) - 1;

    } while (r > 0);

    return result;

}

int ElemByIdx(int T[], int idx) {

    if (idx <= 0) {

        return prefixSum(T, 0);

    } else {

        return prefixSum(T, idx) - prefixSum(T, idx - 1);

    }

}

void changeValue(int T[], int n, int idx, int value) {

    addToElement(T, n, idx, value - ElemByIdx(T, idx));

}

int sumSlice(int T[], int idx1, int idx2) {

    return prefixSum(T, idx2 - 1) - prefixSum(T, idx1 - 1);

}

int \*initFenwickTreeFromArray(int a[], int n) {

    int \*T = new int[n];

    for (int i = 0; i < n; i++) {

        T[i] = 0;

    }

    for (int i = 0; i < n; i++) {

        addToElement(T, n, i, a[i]);

    }

    return T;

}

int \*createFenwickTree(int n, int startValue = 0) {

    int \*T = new int[n];

    for (int i = 0; i < n; i++) {

        T[i] = 0;

    }

    return T;

}

**Примечание**

Входные данные:

https://docs.google.com/spreadsheets/d/1pTJ1kf1BXCZvpk4GfbF1yN4z9QtmofeBYkr9IlUWlSQ/edit?usp=sharing