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

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Дерево Фенвика (сумма, максимум, минимум)

Выполнил студент гр. 3331506/20101 Султанов К.В.
Преподаватель Ананьевский М. С.

«___»_____2025 г.

Санкт-Петербург
2025

Оглавление

Введение

В ходе работы рассмотрим такую структура данных как Дерево Фенвика (сумма, максимум, минимум).

Дерево Фенвика (Fenwick Tree, Binary Indexed Tree, BIT) — это эффективная структура данных, предназначенная для быстрого выполнения операций префиксных сумм, поиска максимума/минимума и обновления элементов в динамическом массиве. Оно было предложено Питером Фенвиком в 1994 году как альтернатива дереву отрезков (Segment Tree) с меньшим объемом кода и более высокой скоростью работы на практике. На рисунке 1 представлено упрощенное понимание Дерева Фенвика.

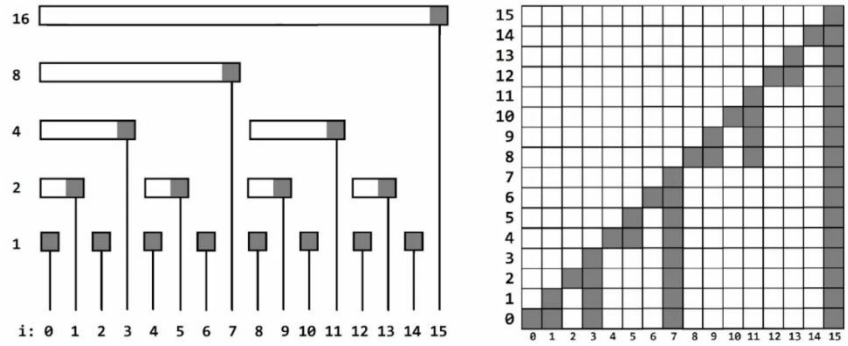

Рисунок 1 - Дерево Фенвика.

Основные преимущества:

- Операции за O(log N):
- Получение суммы/максимума/минимума на префиксе или отрезке.
- Обновление элемента.
- Экономия памяти: Требует O(N) дополнительной памяти (против O(4N) у дерева отрезков).
- Простота реализации: Код короче и понятнее, чем у Segment Tree.

Недостатки:

- Ограниченная функциональность: Не все операции поддерживаются так легко, как у дерева отрезков
- Сложность для неассоциативных операций: Для операций, не являющихся ассоциативными (например, среднее значение), реализация усложняется

График зависимости времени от количества N элементов приведен на рисунке 2.

Рисунок 2 – График зависимости

1. Реализация для различных операций

Дерево Фенвика, хотя и является по идее деревом, представляется в виде массива f,) заменяет группу единичных битов, находящихся в конце числа (младших) на нули. Если x заканчив в котором f [i] - сумма элементов от F(i) до i.

Функция F(x) связана с битовым представлением числа x. Её можно описать следующим образом: F(хается на нулевой бит, то F(x)=x. С помощью битовых операций F(x) записывается следующим образом:

$$F(x)=x \text{ and } (x+1)$$

Значит, функцию нахождения суммы на промежутке [0;x] можно реализовать следующим образом:

2. Дерево Фенвика для нахождения суммы

```
#include <iostream>

const int MAX_N = 1000;
int f[MAX_N];


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


int main() {
    f[1] = 1;
    f[2] = 3;
    std::cout << sum(2) << std::endl; // PIC‹PIPµPrPµC, 4 (1 + 3)
    return 0;
}
```

3. Дерево Фенвика нахождения для максимума

```
#include <iostream>
#include <climits>
#include <stdexcept>

class FenwickMin {
private:
    static const int MAX_SIZE = 1000;
    int tree[MAX_SIZE];
    int actual_size;

public:
    FenwickMin(int n) : actual_size(n) {
        if(n <= 0 || n > MAX_SIZE)
            throw std::invalid_argument("Size must be between 1 and " +
std::to_string(MAX_SIZE));

        for(int i = 0; i < MAX_SIZE; ++i)
            tree[i] = INT_MAX;
    }

    // Обновление значения в позиции pos
    void update(int pos, int value) {
        if(pos < 0 || pos >= actual_size)
            return;

        while(pos < actual_size) {
```

```
            if(tree[pos] > value) {
                tree[pos] = value;
                pos |= pos + 1;  // Переход к следующему индексу
            } else {
                break;
            }
        }
    }

    // Запрос минимума на префиксе [0..pos]
    int query(int pos) const {
        if(pos < 0 || pos >= actual_size)
            return INT_MAX;

        int result = INT_MAX;
        while(pos >= 0) {
            if(tree[pos] < result) {
                result = tree[pos];
            }
            pos = (pos & (pos + 1)) - 1;  // Переход к предыдущему индексу
        }
        return result;
    }

    // Вспомогательная функция для вывода состояния дерева
    void print() const {
        std::cout << "FenwickMin Tree (size " << actual_size << "):\n";
        for(int i = 0; i < actual_size; ++i) {
            std::cout << "[" << i << "] = " << (tree[i] == INT_MAX ? "INF"
: std::to_string(tree[i])) << "\n";
        }
    }
};

int main() {
    try {
        FenwickMin fm(9);

        fm.update(1, 5);
        fm.update(2, 3);
        fm.update(3, 8);
        fm.update(4, 9);
        fm.update(5, 9);
        fm.update(6, 2);

        fm.print();

        std::cout << "\nMin in [1..2]: " << fm.query(3) << "\n";
        std::cout << "Min in [1..6]: " << fm.query(6) << "\n";

        fm.update(2, 1);
        std::cout << "\nAfter update:\n";
        std::cout << "Min in [1..6]: " << fm.query(6) << "\n";

    } catch(const std::exception& e) {
        std::cerr << "Error: " << e.what() << "\n";
        return 1;
    }

    return 0;
}
```

4. Дерево Фенвика для нахождения минимума

```
#include <iostream>
#include <climits>
#include <stdexcept>

class FenwickMin {
private:
    static const int MAX_SIZE = 1000;
    int tree[MAX_SIZE];
    int actual_size;

public:
    FenwickMin(int n) : actual_size(n) {
        if(n <= 0 || n > MAX_SIZE)
            throw std::invalid_argument("Size must be between 1 and " +
std::to_string(MAX_SIZE));

        for(int i = 0; i < MAX_SIZE; ++i)
            tree[i] = INT_MAX;
    }

    // Обновление значения в позиции pos
    void update(int pos, int value) {
        if(pos < 0 || pos >= actual_size)
            return;

        while(pos < actual_size) {
            if(tree[pos] > value) {
                tree[pos] = value;
                pos |= pos + 1;  // Переход к следующему индексу
            } else {
                break;
            }
        }
    }

    // Запрос минимума на префиксе [0..pos]
    int query(int pos) const {
        if(pos < 0 || pos >= actual_size)
            return INT_MAX;

        int result = INT_MAX;
        while(pos >= 0) {
            if(tree[pos] < result) {
                result = tree[pos];
            }
            pos = (pos & (pos + 1)) - 1;  // Переход к предыдущему индексу
        }
        return result;
    }

    // Вспомогательная функция для вывода состояния дерева
    void print() const {
        std::cout << "FenwickMin Tree (size " << actual_size << "):\n";
        for(int i = 0; i < actual_size; ++i) {
            std::cout << "[" << i << "] = " << (tree[i] == INT_MAX ? "INF"
: std::to_string(tree[i])) << "\n";
        }
```

```
    }
};

int main() {
    try {
        FenwickMin fm(9);

        fm.update(1, 5);
        fm.update(2, 3);
        fm.update(3, 8);
        fm.update(4, 9);
        fm.update(5, 9);
        fm.update(6, 2);

        fm.print();

        std::cout << "\nMin in [1..2]: " << fm.query(3) << "\n";
        std::cout << "Min in [1..6]: " << fm.query(6) << "\n";

        fm.update(2, 1);
        std::cout << "\nAfter update:\n";
        std::cout << "Min in [1..6]: " << fm.query(6) << "\n";

    } catch(const std::exception& e) {
        std::cerr << "Error: " << e.what() << "\n";
        return 1;
    }

    return 0;
}
```

5. Заключение

В ходе работы был изучен и продемонстрирован алгоритм работы дерева Фенвика, который является эффективным методом для нахождения суммы, максимума и минимума на префиксе или отрезке. Алгоритм позволяет динамически обрабатывать строки с логарифмической временной сложностью, что значительно превышает производительность наивных методов.

6. Список литературы

1. Борис Рябко (1989). «Быстрый онлайн-код» (PDF). Докл. АН СССР. Математика. 39 (3): 533–537.

2. Борис Рябко (1992). «Быстрый адаптивный код для онлайн-кодирования» (PDF). IEEE Transactions on Information Theory. 28 (1): 1400–1404.

3. Рябко Б.Я. "Быстрый последовательный код", Доклады АН СССР, том 306, номер 3, стр. 548–552.

4. А. Лахно Дерево Фенвика. Курс «Структуры данных», МЦНМО.

5. Бьерн Страуструп. Язык программирования C++. – Москва: Бином, 2006.
   – 1104 с. – ISBN 5-7989-0223-4.

6. Скотт Мейерс. Эффективный и современный C++: 42 рекомендации по использованию C++11 и C++14. – Москва: Вильямс, 2016. – 304 с. – ISBN 978-5-8459-2052-1.