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

К У Р С О В А Я Р А Б О Т А

Дерево Фенвика

по дисциплине «Объектно-ориентированное программирование»

Выполнил
студент гр. 3331506/80401 С.В. Долгов

Руководитель Е.М.Кузнецова

«___» _____ 2021 г.

Санкт-Петербург
2021

Содержание

1. 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;

Наиболее распространённое применение дерева Фенвика - вычисление суммы на отрезке.

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

2. Описание алгоритма

Для простоты описания предполагается, 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:

$$Ti = \text{сумма } Aj \text{ для всех } F(i) <= j <= i,$$

где *F(i)* — некоторая функция, от которой зависит скорость работы.

В этом случае псевдокод будет выглядеть примерно так:

```
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;
     }
}
```

Здесь функция *sum* работает следующим образом: вместо того чтобы идти по всем элементам массива *A*, она движется по массиву *T*, 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 *inc* движется в обратную сторону — в сторону увеличения индексов, обновляя значения суммы T_j только для тех позиций, для которых это нужно, т.е. для всех j, для которых *F(j) <=(меньший из элементов i ; j)*.

Очевидно, что от выбора функции F будет зависеть скорость выполнения обеих операций. Так как в разница во времени выполнения операции может значительно отличаться в зависимости, относится ли данный случай к «лучшим» или же «худшим», было принято решение выбирать функцию F таким образом, чтобы «усреднить» это время, достигнув показателя $O(\log N)$.

Для этого 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)*.

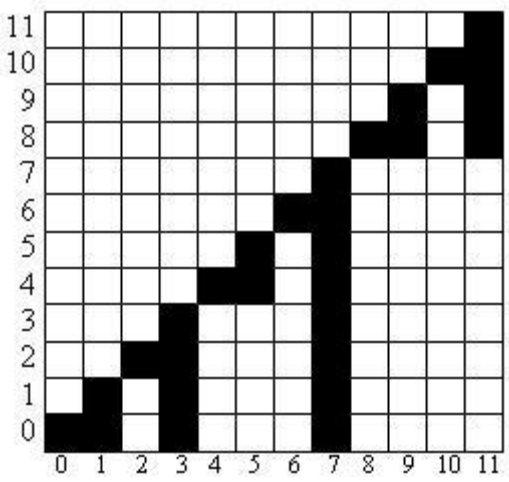
Этому довольно сложному описанию соответствует очень простая формула:

$$F(X) = X \,\&\, (X + 1),$$

где & — это операция побитового логического "И".

Обобщая вышенаписанное, суть алгоритма заключается в изменении исходного массива элементов на массив, содержащий частичные суммы, т.е. суммы элементов на отрезках (на которые разбивается исходный массив). В итоге мы получаем массив, с которым, к примеру, можем вычислить сумму элементов исходного массива на интересующем нас отрезке просто вычтя из его конца начало. От способа разбиения отрезка на подотрезки зависит скорость пересчёта получаемого массива при изменении элемента исходного массива. Чтобы достичь показателя $O(\log N)$ следует разделить исходный массив на отрезки по правилу $[F(R); R]$, где $F(R)$ – левая граница отрезка (R-соответственно, правая), вычисляемая по правилу R = R&(R+1). Деление происходит от конца к началу исходного массива.

Для более наглядного рассмотрения возможностей алгоритма можно воспользоваться следующим изображением:



Здесь по горизонтали изображены номера ячеек полученного массива, а по вертикали – исходного. Закрашенные клетки показывают, какие элементы исходного массива входят в состав частичных сумм, результат которых хранится в соответствующих ячейках получаемого массива. Т.е. , к примеру в ячейке 5 хранится сумма 5 и 4 элемента, а в ячейке 7 – сумма с 0 по 7 элемент.

В случае, если нам требуется, к примеру, вычислить сумму на отрезке [0;9] мы можем, вместо поочерёдного складывания всех 10 элементов просто сложить значения 9, 8 и 7 ячейки полученного массива.

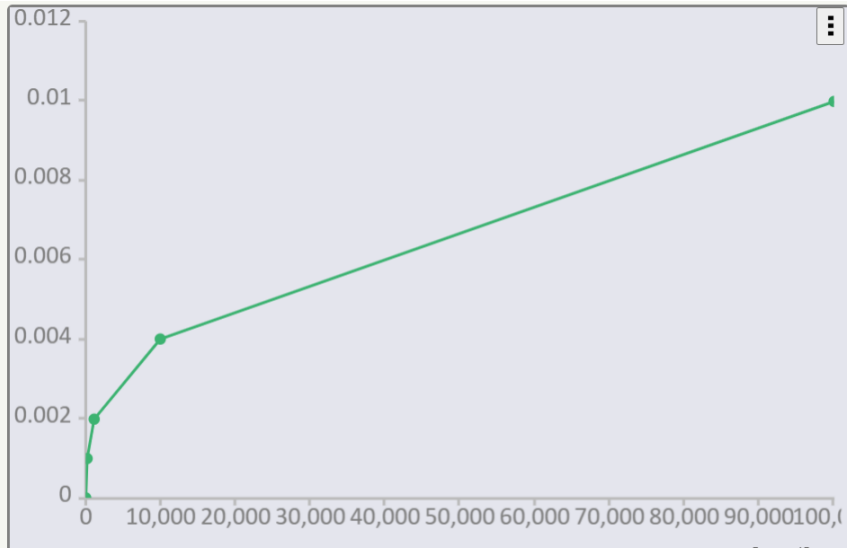
Также, если требуется внести изменения, к примеру, во 2 элемент исходного массива, это отразится только на значениях ячеек 2, 3 и 7 (в нашем случае).

3. Реализация алгоритма

```
#include <iostream>
#include <vector>
std::vector<int> t;
int n;

const int INF = 1000000000;

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 = fmin(result, t[r]);
    return result;
}

void update(int i, int new_val) {
    for (; i < n; i = (i | (i + 1))) t[i] = fmin(t[i], new_val);
}

void init(std::vector<int> a) {
    init((int) a.size());
    for (unsigned i = 0; i < a.size(); i++) update(i, a[i]);
}

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

int sum(int l, int r) {
    return sum(r) - sum(l - 1);
}


int main() {

  for(int k = 1;k< 100000000;k*=10) {
    clock_t start, end;
    start = clock();
    init(k);
    for (int i = 0; i < k; i++) { update(i, i); }
    printf("Sum - %d " , sum(1, k-1));
    end = clock();
    printf("Time - %.4f second(s)  ",
           ((double) end - start) / ((double) CLOCKS_PER_SEC));
    printf("number of elements %d\n",k);
  }

  return 0;

}
```

4. Анализ

Для оценки времени на выполнение операции обновления элемента и пересчёта сумм были произведены замеры времени в зависимости от числа элементов. В итоге была получена следующая зависимость для изменения 1 элемента (горизонтальная ось - число элементов, вертикальная - время):



что соответствует ожидаемому результату.

Список литературы

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