САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №2

по курсу «Алгоритмы и структуры данных»

Тема: Двоичные деревья поиска

Вариант 3

Выполнил:

Бунос М.В.

К3141

Проверила:

Артамонова В.Е.

Санкт-Петербург

2023 г.

Содержание отчета

Задачи по варианту

Задача №3. Простейшее BST [2 s, 256 Mb, 1 балл]

В этой задаче вам нужно написать простейшее BST по явному ключу и отвечать им на запросы:

«+ x» – добавить в дерево x (если x уже есть, ничего не делать).

«> x» – вернуть минимальный элемент больше x или 0, если таких нет.

Листинг кода:

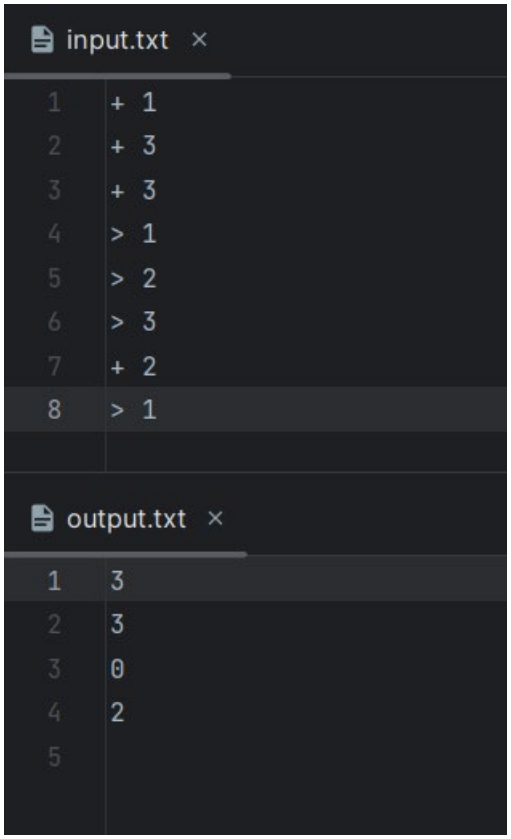
```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>

using namespace std;

//-----------------------------------

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
&pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
<< " MB\n";
}

void getFirstTime() {
    start = clock();
}

void printTimeUse() {
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
" sec\n";
}

//-----------------------------------
struct node {
    int key;
    node *left, *right;
```

```
    node(int x) {
        key = x;
        left = right = NULL;
    }
};

node* insert(node* root, int key) {
    if (!root) {
        return new node(key);
    }
    if (key < root->key) {
        root->left = insert(root->left, key);
    } else if (key > root->key) {
        root->right = insert(root->right, key);
    }
    return root;
}

int get_min_greater(node* root, int x) {
    int min_greater = 0;
    while (root) {
        if (root->key > x) {
            min_greater = root->key;
            root = root->left;
        } else {
            root = root->right;
        }
    }
    return min_greater;
}


int main() {
    getFirstTime();

    // ---- code starts here -----

    node* root = NULL;
    string query;
    while (fin >> query) {
        int x;
        fin >> x;
        if (query == "+") {
            root = insert(root, x);
        } else if (query == ">") {
            fout << get_min_greater(root, x) << endl;
        }
    }

    // ---- code ends here -----

    printTimeUse();
    printMemoryUse();

    fin.close();
    fout.close();
    return 0;
}
```

Текстовое объяснение решения:

Написали простейшее двоичное дерево, которое хранит меньшие ключи в левом поддереве, большие – в правом, рекурсивно смотрим и возвращаем.

Результат работы кода на примерах из текста задачи:



Проверка задачи на (openedu, acmp и тд при наличии в задаче): -

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.800781 MB
Пример из задачи	0.000000 sec	0.800781 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.812500 MB

Вывод по задаче: Легчайшая задача.

Задача №8. Высота дерева возвращается [2 s, 256 Mb, 2 балла]

Высотой дерева называется максимальное число вершин дерева в цепочке, начинающейся в корне дерева, заканчи-вающейся в одном из его листьев, и не содержащей никакую вершину дважды.
Так, высота дерева, состоящего из единственной вершины, равна единице.
Высота пустого дерева равна нулю.
Высота дерева, изображенного на рисунке, равна четырем.
Дано двоичное дерево поиска. В вершинах этого дерева записаны ключи – целые числа, по модулю не превышаю-
щие 109. Для каждой вершины дерева V выполняется следующее условие:
• все ключи вершин из левого поддерева меньше ключа вершины V ;
• все ключи вершин из правого поддерева больше ключа вершины V .
Найдите высоту данного дерева.

Листинг кода:

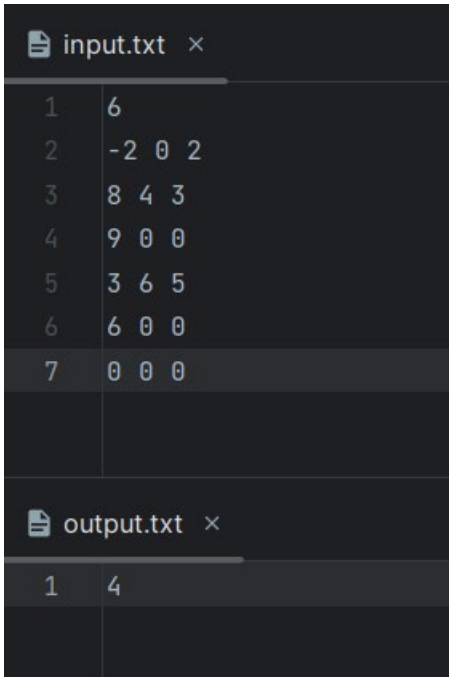
```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>

using namespace std;

//-------------------------------------

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
&pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
<< " MB\n";
}

void getFirstTime() {
    start = clock();
}

void printTimeUse() {
```

```
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
" sec\n";
}

//-----------------------------------
int dfs(int i, vector <vector <int> > &g, int d = 1, int p = -1) {
    int maxDepth = d;
    for(int j : g[i]) {
        if(j == p)
            continue;

        maxDepth = max(dfs(j, g, d + 1, i), maxDepth);
    }

    return maxDepth;
}


int main() {
    getFirstTime();

    // ---- code starts here -----

    int n;
    fin >> n;

    vector <vector <int> > g(n, vector <int> ());

    for(int i = 0; i < n; ++ i) {
        int k, l, r;
        fin >> k >> l >> r;

        if(l)
            g[i].push_back(l - 1);
        if(r)
            g[i].push_back(r - 1);
    }

    fout << dfs(0, g);

    // ---- code ends here -----

    printTimeUse();
    printMemoryUse();

    fin.close();
    fout.close();
    return 0;
}
```

Текстовое объяснение решения:

Формируем список смежности исходя из входных данных, а дальше с помощью DFS смотрим максимальную глубину.

Результат работы кода на примерах из текста задачи:



Проверка задачи на (openedu, acmp и тд при наличии в задаче): -

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.808594 MB
Пример из задачи	0.000000 sec	0.808594 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.812500 MB

Вывод по задаче: Очень легчайшая задача.

Задача №13. Вставка в АВЛ-дерево [2 s, 256 Mb, 3 балла]

Вставка в АВЛ-дерево вершины V с ключом X при условии, что такой вершины в этом дереве нет, осуществляется
следующим образом:
• находится вершина W, ребенком которой должна стать вершина V ;
• вершина V делается ребенком вершины W;
• производится подъем от вершины W к корню, при этом, если какая-то из вершин несбалансирована, произво-
дится, в зависимости от значения баланса, левый или правый поворот.
Первый этап нуждается в пояснении. Спуск до будущего родителя вершины V осуществляется, начиная от корня,
следующим образом:
• Пусть ключ текущей вершины равен Y .
• Если X < Y и у текущей вершины есть левый ребенок, переходим к левому ребенку.
• Если X < Y и у текущей вершины нет левого ребенка, то останавливаемся, текущая вершина будет родителем
новой вершины.
• Если X > Y и у текущей вершины есть правый ребенок, переходим к правому ребенку.
• Если X > Y и у текущей вершины нет правого ребенка, то останавливаемся, текущая вершина будет родителем
новой вершины.
Отдельно рассматривается следующий крайний случай – если до вставки дерево было пустым, то вставка новой
вершины осуществляется проще: новая вершина становится корнем дерева.

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>

using namespace std;

//-----------------------------------
```

```
ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
&pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
<< " MB\n";
}

void getFirstTime() {
    start = clock();
}

void printTimeUse() {
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
" sec\n";
}

//-----------------------------------
class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    int height;
    int index;
    TreeNode(int val, int i) {
        this->val = val;
        left = NULL;
        right = NULL;
        height = 1;
        index = i;
    }
};

struct Kek {
    int val;
    int left = -1;
    int right = -1;
};

class AVL_Tree {
public:
    TreeNode* insert(TreeNode* root, int key, int index) {
        if (!root) {
            return new TreeNode(key, index);
        }
        else if (key < root->val) {
            root->left = insert(root->left, key, index);
        }
        else {
            root->right = insert(root->right, key, index);
```

```
        }
        root->height = 1 + max(getHeight(root->left), getHeight(root-
>right));
        int balance = getBalance(root);
        if (balance > 1 && key < root->left->val) {
            return rightRotate(root);
        }
        if (balance < -1 && key > root->right->val) {
            return leftRotate(root);
        }
        if (balance > 1 && key > root->left->val) {
            root->left = leftRotate(root->left);
            return rightRotate(root);
        }
        if (balance < -1 && key < root->right->val) {
            root->right = rightRotate(root->right);
            return leftRotate(root);
        }
        return root;
    }

    void prepareTree(TreeNode* currentNode, int index, vector <Kek> &g) {
        if(g[index].left != -1) {
            currentNode->left = new TreeNode(g[g[index].left].val,
g[index].left);
            prepareTree(currentNode->left, g[index].left, g);
        }
        if(g[index].right != -1) {
            currentNode->right = new TreeNode(g[g[index].right].val,
g[index].right);
            prepareTree(currentNode->right, g[index].right, g);
        }
    }

    int getHeight(TreeNode* root) {
        if (!root) {
            return 0;
        }
        return root->height;
    }

    int getBalance(TreeNode* root) {
        if (!root) {
            return 0;
        }
        return getHeight(root->left) - getHeight(root->right);
    }

    TreeNode* leftRotate(TreeNode* z) {
        TreeNode* y = z->right;
        TreeNode* t = y->left;
        y->left = z;
        z->right = t;
        z->height = 1 + max(getHeight(z->left), getHeight(z->right));
        y->height = 1 + max(getHeight(y->left), getHeight(y->right));
        return y;
    }

    TreeNode* rightRotate(TreeNode* z) {
        TreeNode* y = z->left;
        TreeNode* t = y->right;
```

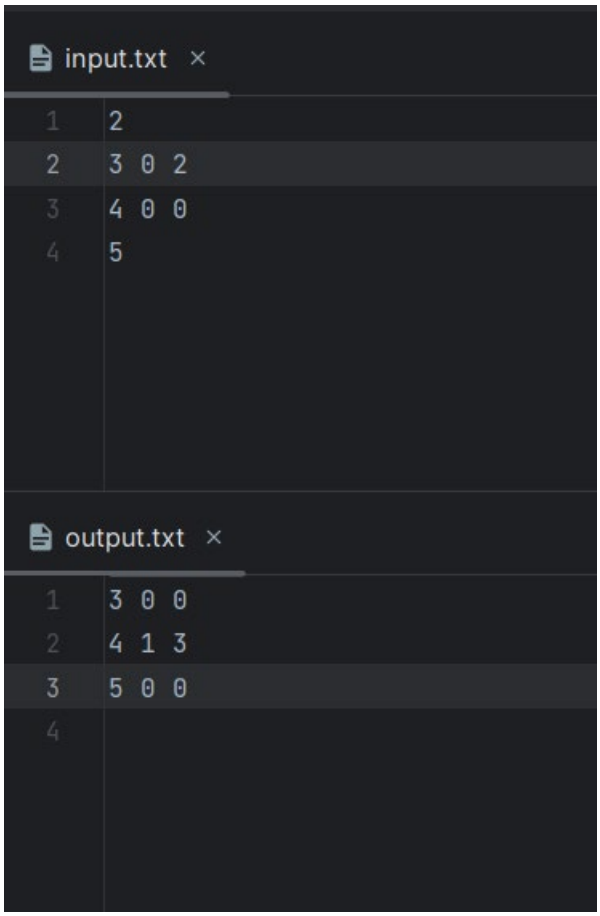
```
        y->right = z;
        z->left = t;
        z->height = 1 + max(getHeight(z->left), getHeight(z->right));
        y->height = 1 + max(getHeight(y->left), getHeight(y->right));
        return y;
    }

    void preOrder(TreeNode* root, int index, vector <Kek> &g) {
        g[index] = {root->val, -1, -1};

        if(root->left) {
            g[index].left = root->left->index;
            preOrder(root->left, root->left->index, g);
        }

        if(root->right) {
            g[index].right = root->right->index;
            preOrder(root->right, root->right->index, g);
        }

    }
};


int main() {
    getFirstTime();

    // ---- code starts here -----

    AVL_Tree myTree;
    TreeNode* root = NULL;
    int rootID = 0;

    int n;
    fin >> n;

    vector <Kek> g (n);
    vector <int> used (n);

    for(int i = 0; i < n; ++ i) {
        int k, l, r;
        fin >> k >> l >> r;

        g[i] = {k, l - 1, r - 1};
        used[l - 1] = 1;
        used[r - 1] = 1;
    }

    for(int i = 0; i < n; ++ i)
        if(!used[i]) {
            rootID = i;
            break;
        }

    root = new TreeNode(g[rootID].val, rootID);
    myTree.prepareTree(root, rootID, g);

    int x;
    fin >> x;

    root = myTree.insert(root, x, n);
```

```
    vector <Kek> ans (n + 1);
    myTree.preOrder(root, root->index, ans);

    for(int i = 0; i <= n; ++ i)
        fout << ans[i].val << ' ' << ans[i].left + 1 << ' ' << ans[i].right
+ 1 << '\n';


    // ---- code ends here -----

    printTimeUse();
    printMemoryUse();

    fin.close();
    fout.close();
    return 0;
}
```

Текстовое объяснение решения:

Реализуем то, что написано в условии, используя два класса и структуру. Также для поддержания индексов, будем использовать список смежности.

Результат работы кода на примерах из текста задачи:



Проверка задачи на (openedu, acmp и тд при наличии в задаче): -

	Время выполнения	Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.807363 MB
Пример из задачи	0.000000 sec	0.812500 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.010000 sec	0.813600 MB

Вывод по задаче: Не очень простая, но и не очень сложная задача. Чисто на реализацию.

Дополнительные задачи

Задача №16. K-й максимум [2 s, 512 Mb, 3 балла]

Напишите программу, реализующую структуру данных, позволяющую добавлять и удалять элементы, а также находить k-й максимум.

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

//-----------------------------------

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
&pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
<< " MB\n";
}

void getFirstTime() {
    start = clock();
}

void printTimeUse() {
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
" sec\n";
}

//-----------------------------------

vector <int> fenwick;
void add(int i, int val) {
```

```
    for(int j = i; j < fenwick.size(); j += j & -j)
        fenwick[j] += val;
}

int getsum(int i) {
    int sum = 0;
    for(int j = i; j > 0; j -= j & -j)
        sum += fenwick[j];
    return sum;
}

int main() {
    getFirstTime();

    // ---- code starts here -----

    int n;
    fin >> n;

    vector <pair <int, int> > requests (n);
    for(auto &i : requests)
        fin >> i.first >> i.second;

    set <int> allElements;
    for(auto &i : requests)
        allElements.insert(i.second);

    gp_hash_table <int, int> number;
    int j = 1;
    for(int i : allElements)
        number[i] = j ++;

    vector <int> elementByNumber (1e5 + 1);
    fenwick.assign(1e5 + 1, 0);
    for(int i : allElements) {
        elementByNumber[number[i]] = i;
    }

    int cnt = 0;

    for(auto& [type, value] : requests) {
        if(type == -1) {
            add(number[value], -1);
            -- cnt;
        } else if(type == 0) {
            int l = 1, r = 1e5 + 1;
            while(l < r) { // 1 0 1 1 0 0
                int m = (l + r + 1) >> 1;
                if(getsum(m - 1) > cnt - value) {
                    r = m - 1;
                } else l = m;
            }

            fout << elementByNumber[l] << '\n';


        } else {
            add(number[value], 1);
            ++ cnt;
        }
    }
```

```
    // ---- code ends here -----

    printTimeUse();
    printMemoryUse();

    fin.close();
    fout.close();
    return 0;
}
```

Текстовое объяснение решения:

Для решения данной задачи напишем неявное дерево Фенвика. Для получения k-ого максимума будем дихать, асимптотика O(N log^2 N), что вполне приемлемо для данной задачи.

Результат работы кода на примерах из текста задачи:

input.txt

```
11
+1 5
+1 3
+1 7
0 1
0 2
0 3
-1 5
+1 10
```

output.txt

```
7
5
3
10
7
3
```

Проверка задачи на (openedu, acmp и тд при наличии в задаче): -

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.000000 sec	0.808594 MB

входных данных из текста задачи		
Пример из задачи	0.001000 sec	1.578125 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.042000 sec	6.351562 MB

Вывод по задаче: Легчайшая задача для величайшего.

Вывод

Очень познавательная лабораторная работа, но делать полностью не горю желанием.