**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Курсовая РАБОТА

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Случайные бинарные деревья поиска. Исследование.

|  |  |  |
| --- | --- | --- |
| Студент гр. 7383 |  | Кирсанов А. Я. |
| Преподаватель |  | Размочаева Н. В. |

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Кирсанов А. Я. | | |
| Группа 7383 | | |
| Тема работы: Случайные бинарные деревья поиска. Исследование. | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание функций работы с бинарными деревьями * Примеры работы программы * Исследование * Заключение * Список использованных источников | | |
| Предполагаемый объем пояснительной записки:  Не менее 15 страниц. | | |
|  | | |
| Дата выдачи задания: | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Кирсанов А. Я. |
| Преподаватель |  | Размочаева Н. В. |

**Аннотация**

В работе была реализована программа на языке программирования С++ с использованием фреймворка Qt, обеспечивающая построения случайных бинарных деревьев поиска, выполняющая вставку и удаления элементов дерева. Было проведено исследование алгоритмов вставки и удаления в среднем и в худшем случае.

**Summary**

The program was implemented in the C ++ programming language using the Qt framework, which provides for the construction of random binary search trees that performs the insertion and deletion of tree elements. A study was conducted on the insertion and deletion algorithms on average and in the worst case.

**содержание**

[Введение 5](#_Toc533490099)

1. [Теоретические сведения 6](#_Toc533490100)

2. [Описание функций работы со случайными бинарными деревьями поиска 7](#_Toc533490101)

3. [Примеры работы программы](#_Toc533490102) 8

4. [Исследование 9](#_Toc533490103)

[Заключение 1](#_Toc533490104)2

[Список использованных источников 1](#_Toc533490105)3

[Приложение А Исходный код программы](#_Toc533490106) 14

### ВВЕДЕНИЕ

Целью работы является создание программы на языке C++ с использованием фреймворка Qt для работы со случайными бинарными деревьями поиска.

Для достижения цели были реализованы функции для работы с БДП: создание дерева, вставка и удаление элемента дерева по ключу, удаление дерева.

### 1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Бинарное дерево поиска (БДП) — это [двоичное дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), для которого выполняются следующие дополнительные условия (*свойства дерева поиска*):

* Оба поддерева — левое и правое — являются двоичными деревьями поиска.
* У всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше*, нежели значение ключа данных самого узла X.
* У всех узлов *правого* поддерева произвольного узла X значения ключей данных *больше либо равны*, нежели значение ключа данных самого узла X.

Очевидно, данные в каждом узле должны обладать ключами, на которых определена операция сравнения *меньше*.

Если структура БДП полностью зависит от того порядка, в котором элементы расположены во входной последовательности, такое БДП называется случайным.

### 2. ОПИСАНИЕ ФУНКЦИЙ РАБОТЫ СО СЛУЧАЙНЫМИ БИНАРНЫМИ ДЕРЕВЬЯМИ ПОИСКА

1. Структура бинарного дерева **node** состоит из указателей на левое и правое поддерево, целое значение узла, а также поля, в котором хранится размер (в вершинах) дерева с корнем в данном узле.
2. **Insert –** функция вставки нового элемента в дерево. Также инициализирует дерево, если в качестве указателя на дерево подается нулевой указатель. Производит вставку элемента по правилам построения бинарного дерева поиска.
3. **Remove –** функция удаления элемента из дерева. При удалении БДП перестраивает дерево таким образом, чтобы сохранялись все свойства БДП.
4. **Join –** функция объединения двух поддеревьев. Используется в работе функции **remove**.
5. Функции **getsize** и **fixsize** используются соответственно для получения размера в вершинах дерева с корнем в данном узле и для изменения размера.
6. **Destroy –** функция, удаляющая дерево. Возвращает нулевой указатель.

### 3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Программа собрана в операционной системе Windows 10 в Qt 4.7.1. В других ОС и средах тестирование не проводилось.

На рисунке 1 представлен пример работы программы.

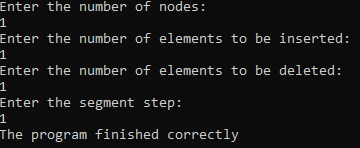


Рисунок 1 – Пример работы программы

Пользователю предлагается ввести начальное количество элементов БДП и диапазон их значений. Элементы генерируются случайным образом.

Затем программа просит ввести количество вставляемых и удаляемых элементов и шаг вывода значений функций.

На рисунке 2 представлен пример генерируемого программой файла, отражающий работу функций для 5 начальных узлов, 5 добавляемых и 5 удаляемых с шагом 1.

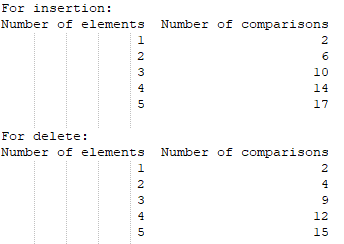
****

Рисунок 2 – Иллюстрация файла лога программы

### 4. ИССЛЕДОВАНИЕ

Для проведения исследования реализованных алгоритмов в среднем и в худшем случае была использована зависимость количества операций сравнений *C* от количества вставляемых или удаляемых узлов *N*.

Так как случайное дерево хорошо сбалансированно, должна наблюдаться разница в зависимости *C* от *N* в среднем и в худшем случае.

Найдем зависимость *C* от *N*, в среднем и в худшем случае для вставки и для удаления. Для нахождения первой зависимости будем подавать элементы, сгенерированные случайном образом с равной, а во втором случае будем последовательно подавать упорядоченные по возрастанию элементы – худший случай, при котором БДП вырождается в линейный список. В обоих случаях создадим дерево, состоящее из 5000 элементов и добавим к нему еще 5000 элементов с шагом 250 элементов. Зависимость *C* при вставке от количества элементов *N* в среднем показана в таблице 1.

Таблица 1 – Зависимость *C* при вставке от N в среднем.

|  |  |  |
| --- | --- | --- |
| *N* | *C* при вставке | *C* при удалении |
| 250 | 3888 | 3112 |
| 500 | 7803 | 6313 |
| 750 | 11781 | 9644 |
| 1000 | 15954 | 13263 |
| 1250 | 20161 | 16842 |
| 1500 | 24305 | 20406 |
| 1750 | 28563 | 24084 |
| 2000 | 32808 | 27750 |
| 2250 | 37255 | 31605 |
| 2500 | 41572 | 35290 |
| 2750 | 45883 | 38992 |
| 3000 | 50316 | 42815 |
| 3250 | 54694 | 46715 |
| 3500 | 59100 | 50535 |
| 3750 | 63555 | 54379 |
| 4000 | 68026 | 58204 |
| 4250 | 72452 | 61921 |
| 4500 | 76873 | 65632 |
| 4750 | 81358 | 69445 |
| 5000 | 102714 | 71929 |

На рисунке 3 соответственно изображены графики зависимостей *C* при вставке и при удалении от *N*.

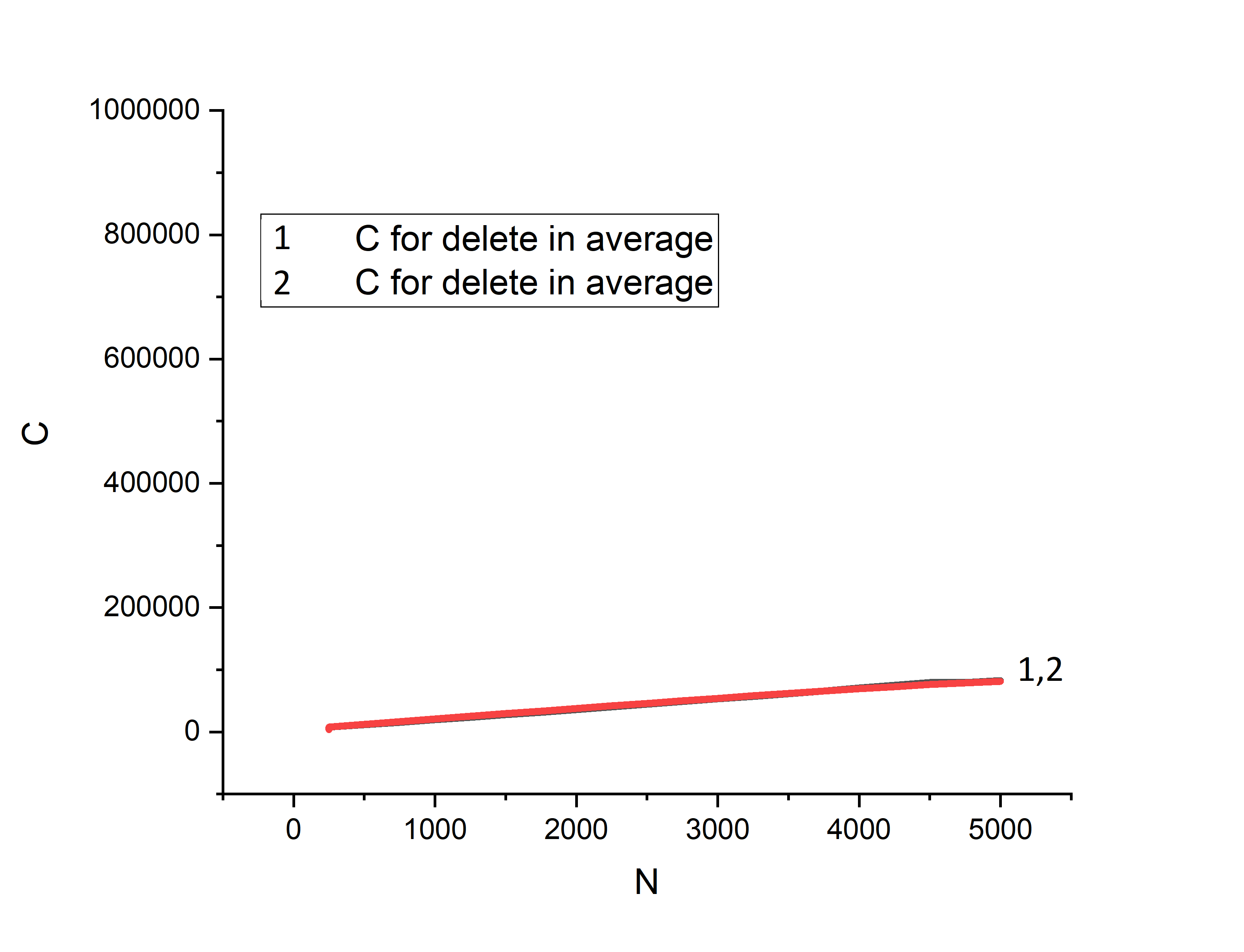


Рисунок 3 – Зависимость *С* при вставке от *N* в среднем

Аналогично для удаления элементов. Данные представлены в таблице 2.

Таблица 2 – Зависимость времени удаления от количества элементов.

|  |  |  |
| --- | --- | --- |
| *N* | *C* при вставке в худшем случае | *C* при удалении в худшем случае |
| 250 | 94125 | 186910 |
| 500 | 250750 | 1,3983E6 |
| 750 | 469875 | 2,55218E6 |
| 1000 | 751500 | 3,64355E6 |
| 1250 | 1,09563E6 | 4,67243E6 |
| 1500 | 1,50225E6 | 5,6388E6 |
| 1750 | 1,97138E6 | 6,54268E6 |
| 2000 | 2,503E6 | 7,38405E6 |
| 2250 | 3,09713E6 | 8,16293E6 |
| 2500 | 3,75375E6 | 8,8793E6 |
| 2750 | 4,47288E6 | 9,53318E6 |
| 3000 | 5,2545E6 | 1,01246E7 |
| 3250 | 6,09863E6 | 1,06534E7 |
| 3500 | 7,00525E6 | 1,11198E7 |
| 3750 | 7,97438E6 | 1,15237E7 |
| 4000 | 9,006E6 | 1,18651E7 |
| 4250 | 1,01001E7 | 1,21439E7 |
| 4500 | 1,12568E7 | 1,23603E7 |
| 4750 | 1,24759E7 | 1,25142E7 |
| 5000 | 1,26806E7 | 1,26056E7 |

На рисунке 4 изображены графики зависимостей времени *С* при удалении от количества элементов *N* в худшем случае.

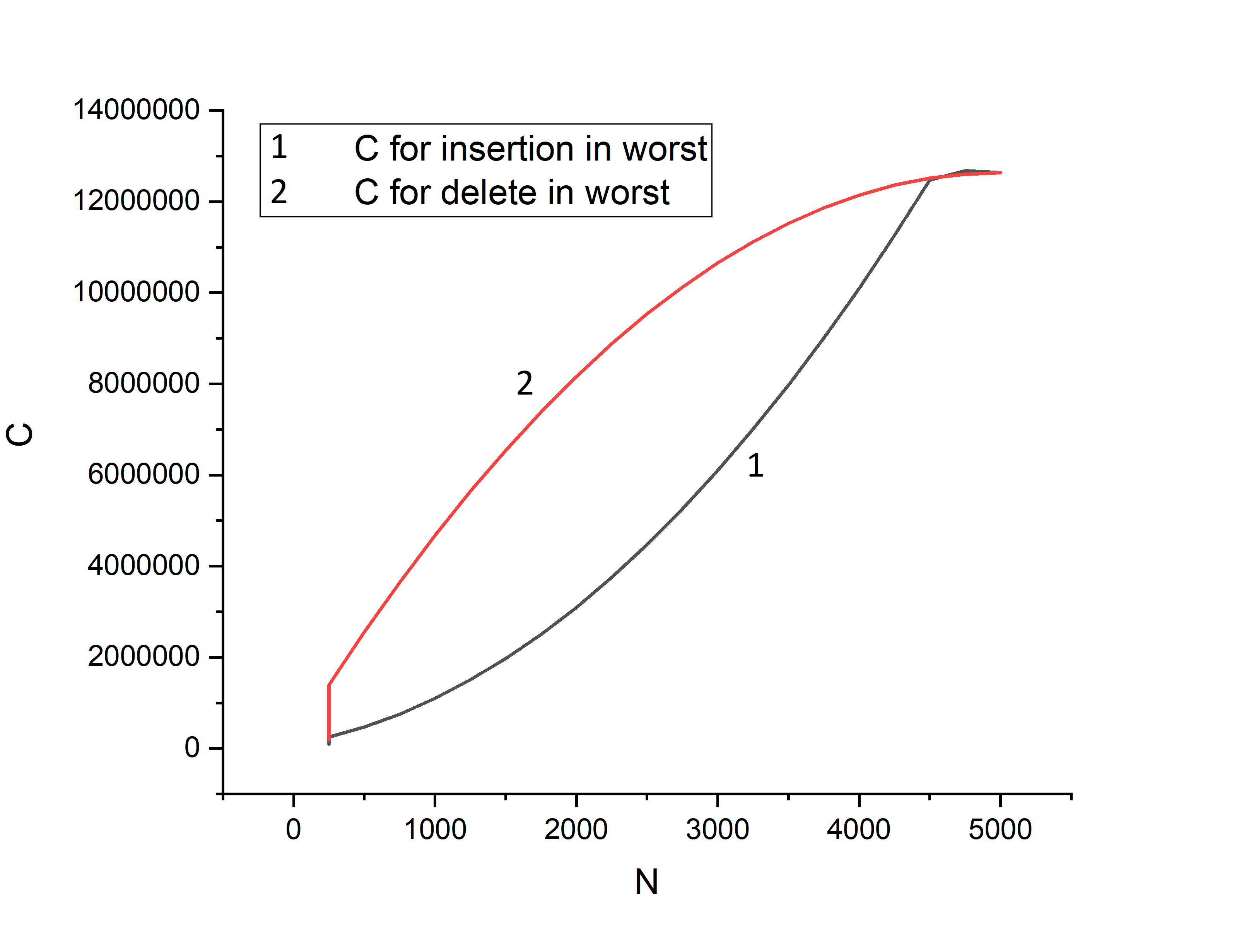


Рисунок 4 – Зависимость С при удалении от *N* в худшем случае

### ЗАКЛЮЧЕНИЕ

В данной работе создана программа для исследования свойств случайных бинарных деревьев. Исследовано поведение написанных функций для работы со случайными БДП в среднем и в худшем случае. В худшем случае БДП вырождается в линейный список. Выявлена зависимость между C и N при вставке и при удалении в среднем и в худшем случаях (см. рис. 3,4). В худшем случае произведение операции вставки и удаления потребует в больше времени, чем в среднем.

### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хабр. URL: <https://habr.com/post/145388/>

2. Randomized Binary Search Trees. URL: http://akira.ruc.dk/~keld/teaching/algoritmedesign\_f08/Artikler/03/Martinez97.pdf

### ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

#include <iostream>

#include <ctime>

#include <vector>

#include <random>

#include <algorithm>

#include <iterator>

#include <iomanip>

#include <fstream>

using namespace std;

struct node // структура для представления узлов дерева

{

int key;

int size;

node\* left;

node\* right;

node(int k) { key = k; left = right = 0; size = 1; }

};

int getsize(node\* T) // обертка для поля size, работает с пустыми деревьями (t=NULL)

{

if( !T ) return 0;

return T->size;

}

void fixsize(node\* T) // установление корректного размера дерева

{

T->size = getsize(T->left)+getsize(T->right)+1;

}

node\* join(node\* T, node\* q) // объединение двух деревьев

{

if( !T ) return q;

if( !q ) return T;

if( rand()%(T->size+q->size) < T->size )

{

T->right = join(T->right,q);

fixsize(T);

return T;

}

else

{

q->left = join(T,q->left);

fixsize(q);

return q;

}

}

node\* insert(node\* T, int k, unsigned int\* count) // вставка нового узла с ключом k в дерево p

{

if( !T ) return new node(k);

if( T->key>k ){

(\*count)++;

T->left = insert(T->left,k, count);

}

else{

(\*count)++;

T->right = insert(T->right,k, count);

}

fixsize(T);

return T;

}

node\* remove(node\* T, int k, unsigned int\* comparison) // удаление из дерева p первого найденного узла с ключом k

{

if( !T ) return T;

if( T->key==k )

{

(\*comparison)++;

node\* q = join(T->left,T->right);

delete T;

return q;

}

else if( k<T->key ){

(\*comparison)++;

T->left = remove(T->left,k, comparison);

}

else{

(\*comparison)++;

T->right = remove(T->right,k, comparison);

}

return T;

}

node\* destroy(node\* T){

if (T->left)

delete T->left;

if (T->right)

delete T->right;

delete T;

return T = NULL;

}

int main()

{

srand(unsigned(time(0)));

node \*T = NULL;

unsigned int count, ins, del, step, comparison = 0;

ofstream file;

file.open("output.txt");

cout << "Enter the number of nodes:" << endl;

cin >> count;

cout << "Enter the number of elements to be inserted: " << endl;

cin >> ins;

cout << "Enter the number of elements to be deleted: " << endl;

cin >> del;

cout << "Enter the segment step:" << endl;

cin >> step;

if(count > 50000) count = 50000;

if(ins > 50000) ins = 50000;

vector<unsigned int> x(count);

for(unsigned int i = 0; i < count; i++){

x[i] = i;

}

random\_shuffle(x.begin(), x.end());

for(unsigned int i = 0; i < count; i++){

T = insert(T, x[i], &comparison);

}

file << "For insertion:" << endl;

file << "Number of elements\tNumber of comparisons" << endl;

comparison = 0;

for(unsigned int i = step; i <= ins; i+=step){

for (int k = i; k < i+step; k++) {

T = insert(T, x[k], &comparison);

}

file.width(18);

file << i;

file.width(23);

file << comparison;

file << endl;

}

file << endl << "For delete:" << endl;

file << "Number of elements\tNumber of comparisons" << endl;

comparison = 0;

if(del > count+ins) del = count + ins;

for(int i = step; i <= del; i+=step){

for (int k = i; k < i+step; k++) {

T = remove(T, x[k], &comparison);

}

file.width(18);

file << i;

file.width(23);

file << comparison;

file << endl;

}

destroy(T);

cout << "The program finished correctly";

return 0;

}