МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: «БДП»

Студент гр. 8381 _____ **Переверзев Д.Е.**

Преподаватель _____ **Жангиров Т.Р.**

Санкт-Петербург

2019

Цель работы

Ознакомиться с основными характеристиками и особенностями такой структуры данных, как БДП, изучить особенности ее реализации на языке программирования C++. Разработать программу, которая строит изображение БДП и удаляет заданный элемент.

Задание

1) По заданному файлу F (типа file of *Elem*), все элементы которого различны, построить структуру данных определённого типа – БДП;
2) б) Для построенной структуры данных проверить, входит ли в неё элемент *e* типа *Elem*, и если входит, то удалить элемент *e* из структуры данных. Предусмотреть возможность повторного выполнения с другим элементом.

Выполнение работы

1. Созданы функции:
   - void push_5(TREE_5 *&tree, int value);
   - void del_elem_5(TREE_5 *&tree, int value);
   - TREE_5 *&find_elem_5(TREE_5 *&tree, int value);
   - void bypass_5(TREE_5 *&tree, string &bin_str);
   - void del_help_5(TREE_5 **buffer, int value);
   - int test_5(string str);

   которые подключены к серверной части через файл addon.cc, использующий библиотеку node.h

2. Входная строка, в которой записаны элементы БДП, передается на сервер, записывается в виде БДП.
3. По запросу клиента на сервере происходит обход этого дерева ,полученная строка скобочной записи бинарного дерева передается клиенту и строится дерево.
4. Также реализован ввод из файла.
5. Рисовка деревьев на клиентской части реализована с помощью функций d3.mini.js .
6. Был разработан WEB UI. Серверная часть была написана на node.js, для обработки строки были написаны методы на c++. Для клиентской части использовались язык разметки HTML, язык таблиц стилей CSS,

JavaScript для обработки действий на странице и передачи данных без обновления страницы с помощью объекта XMLHttpRequest.

Оценка эффективности алгоритма

Алгоритм создания БДП по строке является итеративным, каждый элемент строки обрабатывается один раз, а значит сложность алгоритма можно оценить как $O(N)$.

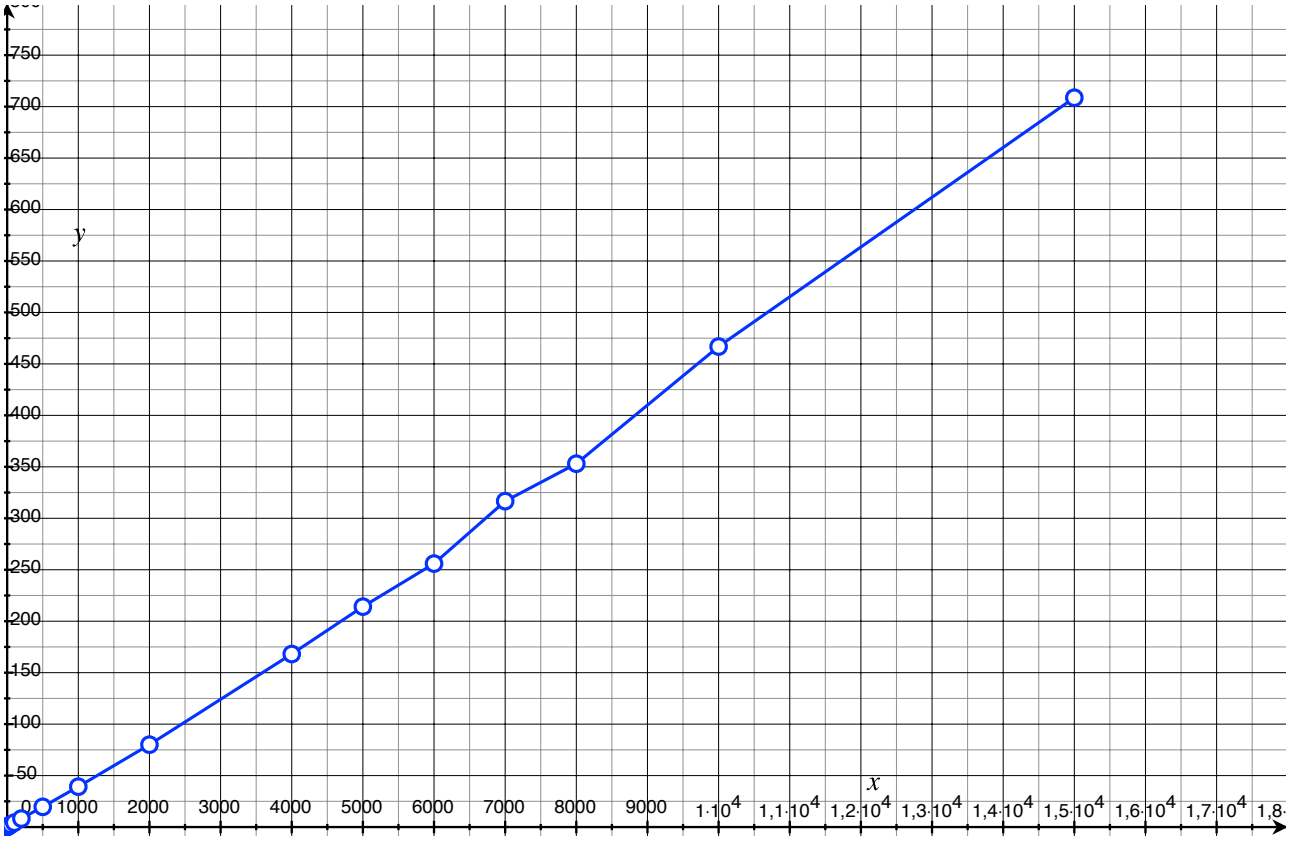

График 1 — Зависимость количества элементов к времени строительства (sec**1e+5*);

Алгоритмы нахождения и удаления заданного элемента является рекурсивным, каждый узел дерева обрабатывается один раз, следовательно, сложность алгоритма также $O(N)$.

Выводы

В ходе лабораторной работы был изучен способ преобразования леса в бинарное дерево и метод обхода дерева в ширину. А также реализован WEB GUI.

Приложение А

Исходный код программы.

main.cpp

```
#include "lr_5_methods.h"
#include "lr_5_methods.cpp"

int main(int argc, const char *argv[])
{
   TREE_5 *tree = NULL;
   char run = 'y';
   int value;
   while (true)
   {
      cout << "Add new element?(y/n)\n";
      cin >> run;
      while (run != 'y' && run != 'n')
      {
         cout << "Wrong answer, enter 'y' or 'n'.\n";
         cin >> run;
      }
      if (run == 'n')
         break;
      cout << "Enter value of new element: ";
      cin >> value;
      push_5(tree, value);
   }

   cout << "Finde elem: ";
   cin >> value;
   if (find_elem_5(tree, value) != NULL)
      tree = NULL;

   return 0;
}
```

lr_4_methods.cpp

```
#include "lr_5_methods.h"
struct TREE_5
{
   int value;
   TREE_5 *left;
   TREE_5 *right;
};

void push_5(TREE_5 *&tree, int value)
{
   if (tree == NULL)
   {
      tree = new TREE_5;

      tree->value = value;
      tree->left = NULL;
      tree->right = NULL;
   }
   else
   {
      if (tree->value < value)
         push_5(tree->right, value);
      else if (tree->value > value)
         push_5(tree->left, value);
   }

   return;
}

void del_elem_5(TREE_5 *&tree, int value)
{
   TREE_5 **buffer;
   if (tree->left)
      if (tree->left->value == value)
         buffer = &tree->left;

   if (tree->right)
      if (tree->right->value == value)
         buffer = &tree->right;

   del_help_5(buffer, value);
```

```
}

void del_help_5(TREE_5 **buffer, int value)
{
   if ((*buffer)->left == NULL && (*buffer)->right == NULL)
   {
      (*buffer) = NULL;
      return;
   }

   //2 - only left branch
   if ((*buffer)->left != NULL && (*buffer)->right == NULL)
   {
      (*buffer) = (*buffer)->left;
      return;
   }
   //3 - have right branch
   if ((*buffer)->right != NULL)
   {
      //3.1 - right leaf has no left branch
      if ((*buffer)->right->left == NULL)
      {
         (*buffer)->right->left = (*buffer)->left;
         (*buffer) = (*buffer)->right;
         return;
      }

      //3.2 - right leaf has left branch
      else
      {

         TREE_5 *&to_left = (*buffer)->right;
         while (to_left->left->left != NULL)
            to_left = to_left->left;
         // _1
         TREE_5 *help = to_left->left->right;
         // _2
         to_left->left->left = (*buffer)->left;
         to_left->left->right = (*buffer)->right;
         (*buffer) = to_left->left;
         to_left->left = help;
```

```
        }
        return;
    }
}

TREE_5 *&find_elem_5(TREE_5 *&tree, int value)
{
    TREE_5 *exist = NULL;
    if (tree != NULL)
    {
        if (tree->value < value)
            exist = find_elem_5(tree->right, value);
        else if (tree->value > value)
            exist = find_elem_5(tree->left, value);
        else if (tree->value == value)
        {
            exist = tree;
            return exist;
        }
    }
    //del elem
    if (exist != NULL)
    {
        del_elem_5(tree, value);
        exist = NULL;
    }

    return exist;
}

void bypass_5(TREE_5 *&tree, string &bin_str)
{
    cout << tree->value;
    bin_str += to_string(tree->value);
    if (tree->left)
    {
        cout << '(';
        bin_str += "(";
        bypass_5(tree->left, bin_str);
    }
```

```
    if (tree->right)
    {
        cout << '(';
        bin_str += "(";
        bypass_5(tree->right, bin_str);
    }
    cout << ')';
    bin_str += ")";
}

int test_5(string str)
{
    if (!isdigit(str[0]))
        return 1;
    for (int i = 1; i < str.length(); i++)
    {
        if (!isdigit(str[i]) && str[i] != ' ')
            return 2;
        if (str[i - 1] == ' ' && str[i] == ' ')
            return 3;
        }
    return 0;
}
```

lr_4_methods.h

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <sstream>

using namespace std;

struct TREE_5;

void push_5(TREE_5 *&tree, int value);
void del_elem_5(TREE_5 *&tree, int value);
TREE_5 *&find_elem_5(TREE_5 *&tree, int value);
void bypass_5(TREE_5 *&tree, string &bin_str);
void del_help_5(TREE_5 **buffer, int value);
int test_5(string str);
```