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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: «Деревья»

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 бинарное дерево, изучить особенности ее реализации на языке программирования C++. Разработать программу, которая строит изображение леса и бинарного дерева.

Задание

6. Для заданного леса с произвольным типом элементов:
- получить естественное представление леса бинарным деревом;
- вывести изображение леса и бинарного дерева;
- перечислить элементы леса в горизонтальном порядке (в ширину).

Выполнение работы

1. Созданы функции:
   - int add_4(TREE *&tree, int top, int *lvl, string *arr_val, int &index);
   - int push_4(TREE *&tree, int top, int *lvl, string *arr_val, int &index);
   - int create_4(int &ind, int *&lvl, string *&arr_val, string forest);
   - void bypass_4(TREE *&tree, string &bin_str);
   - int test_4(string forest);
   - void width_4(TREE *tree, string &bypass_width);

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

2. Входная строка, в которой записано скобочное представление леса, передаваемая с сервера записывается в виде бинарного дерева.
3. По запросу клиента на сервере происходит обход этого дерева ,полученная строка скобочной записи бинарного дерева передается клиенту.
4. Также реализован обход в ширину бинарного дерева.
5. Рисовка деревьев на клиентской части реализована с помощью функций d3.mini.js .
6. Был разработан WEB GUI. 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 бинарного дерева по строке является итеративным, каждый элемент строки обрабатывается один раз, а значит сложность алгоритма можно оценить как $O(N)$.

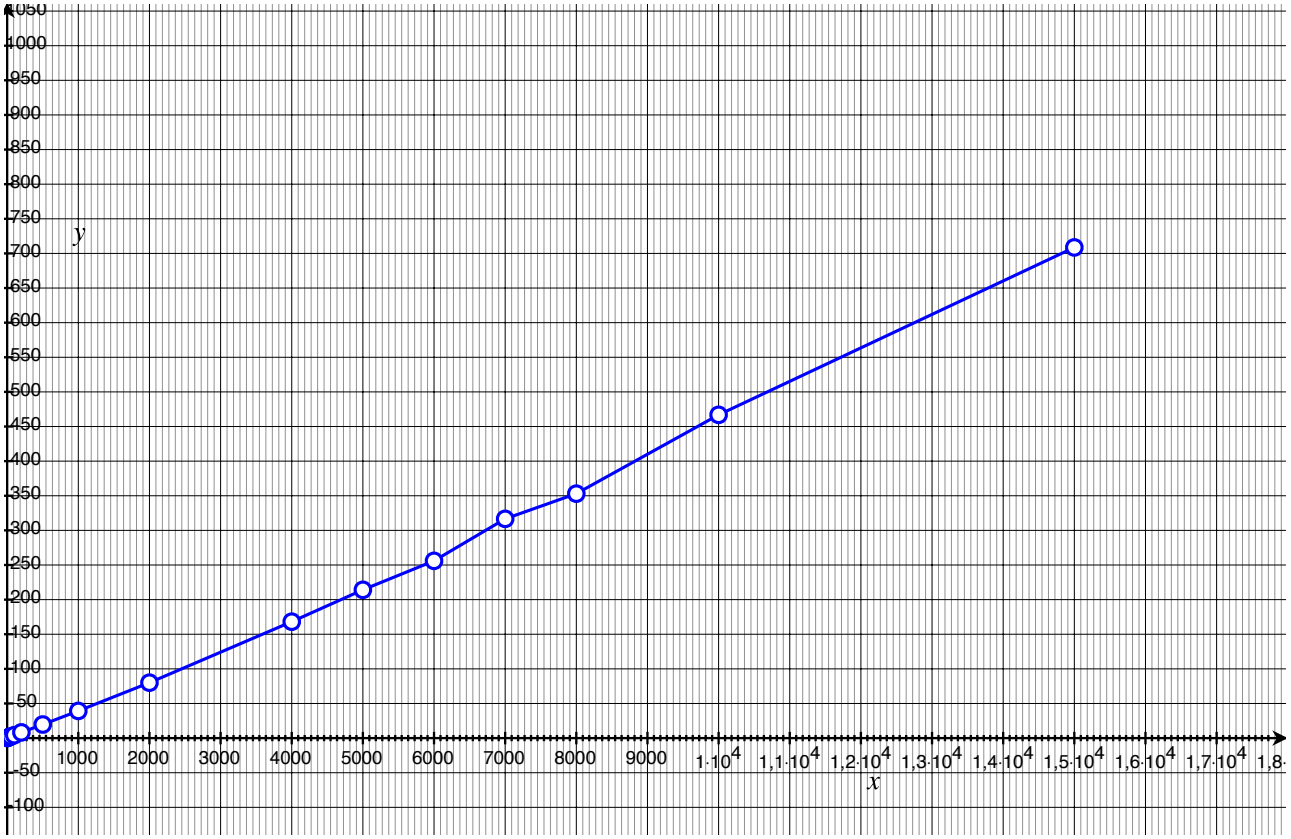

График 1 — Зависимость количества элементов к времени строительства (sec**1e+5*);

Алгоритм вывода элементов дерева в горизонтальном порядке является рекурсивным, каждый узел дерева обрабатывается один раз, следовательно, сложность алгоритма также $O(N)$.

Тестирование программы

Тест 1:

Скобочная запись леса:

(1(2(3(4(5(6(7(8(9(10)(23))))(19(20(24)(21)))))(18)))(11(12(13(14(16))(15(17(20)))))))(1(2(3)(567)))(gfdhgfjkhglkh;glfjghjfkgjk)

Скобочная запись бинарного дерева:

(1(2(3(4(5(6(7(8(9(10(23))))(19(20(24(21))))))(18)))(11(12(13(14(16)(15(17(20)))))))))(1(2(3(567)))(gfdhgfjkhglkh;glfjghjfkgjk)))

Обход бинарного дерева в ширину:

1 | 2 | 1 | 3 | 2 | gfdhgfjkhglkh;glfjghjfkgjk | 4 | 11 | 3 | 5 | 12 | 567 | 6 | 18 | 13 | 7 | 14 | 8 | 19 | 16 | 15 | 9 | 20 | 17 | 10 | 24 | 20 | 23 | 21

Изображение леса:

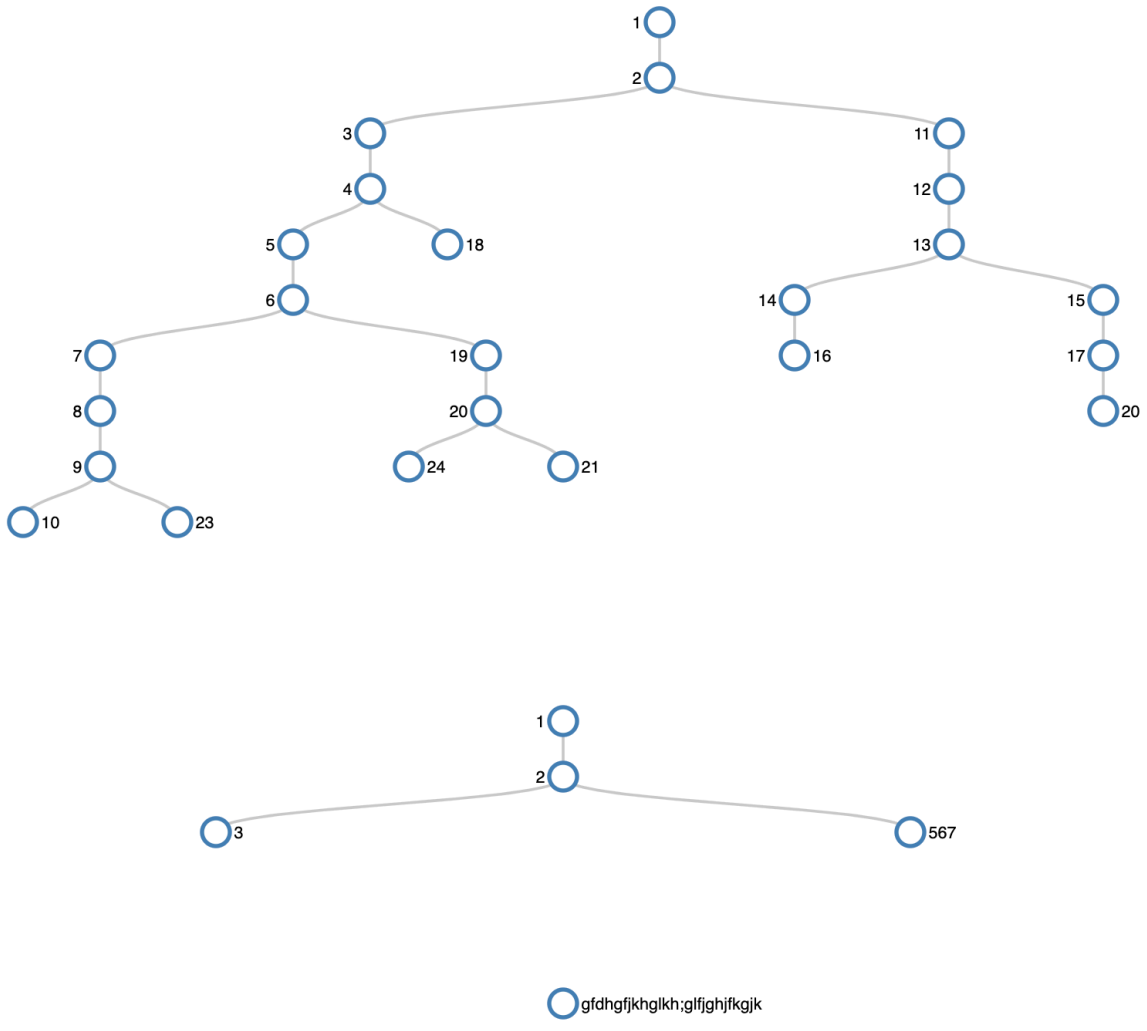

Рисунок 1 - лес(1).

Изображение бинарного дерева:

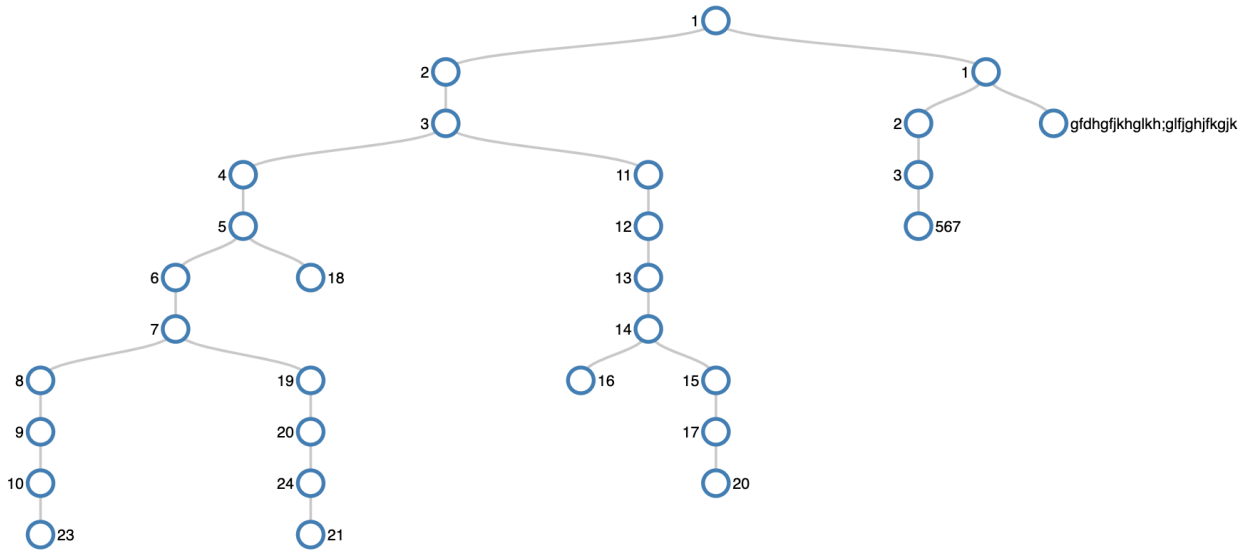

Рисунок 2 - бинарное дерево(1).

Тест 2:

Скобочная запись леса:

(1fcgjh#$%^&vkjb\nlkn)

Скобочная запись бинарного дерева:

(1fcgjh#$%^&vkjb\nlkn)

Обход бинарного дерева в ширину:

1fcgjh#$%^&vkjb\nlkn

Изображение леса и бинарного дерева совпадают:

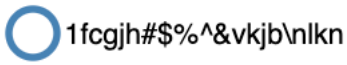

Рисунок 3 - лес и бинарное дерево(2).

Тест 3:

Скобочная запись леса:

(*(л(а(б(а(№(4))))))(с(д(е(л(а(л(:(я)))))))(т(е(м(а(:(д(е(р(е(в(ь(я))))))))))))

Скобочная запись бинарного дерева:

(*(л(а(б(а(№(4)))))(с(д(е(л(а(л(:(я))))))(т(е(м(а(:(д(е(р(е(в(ь(я)))))))))))))

Обход бинарного дерева в ширину:

| л | а | с | б | д | т | а | е | е | № | л | м | 4 | а | а | л | : | : | д | я | е | р | е | в | ь

Изображение леса:

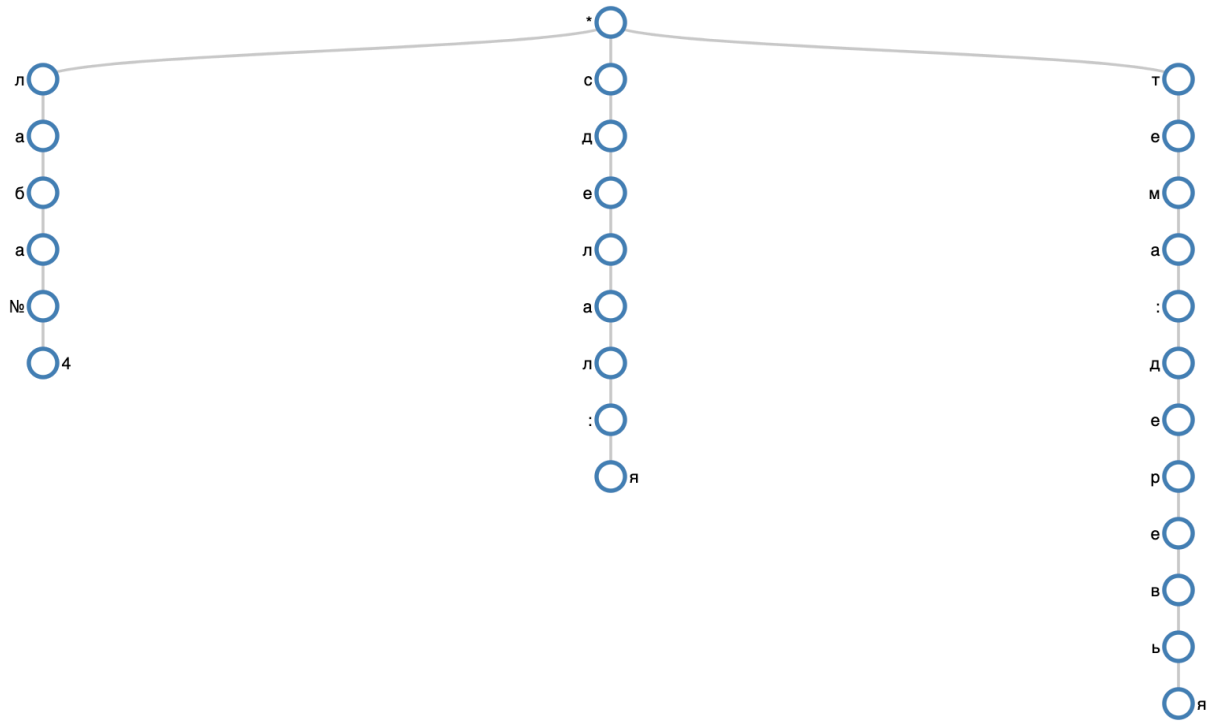

Рисунок 4 - лес(3).

Изображение бинарного дерева:

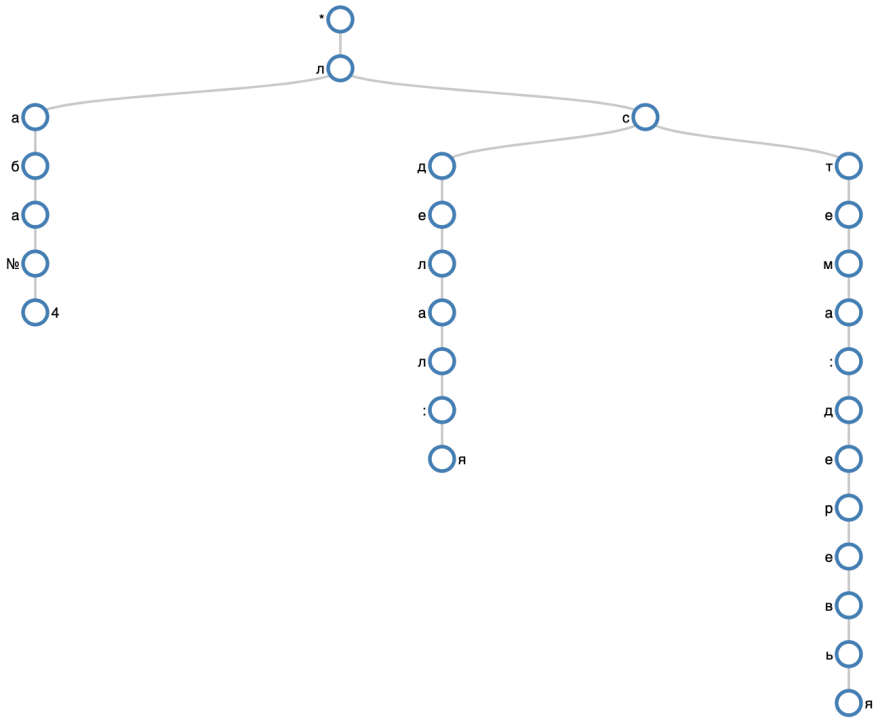

Рисунок 5 - бинарное дерево(3).

Тест 4:

Скобочная запись леса:

()

Error found: err#1

Тест 5:

Скобочная запись леса:

( ) `пробел`

Скобочная запись бинарного дерева:

( ) `пробел`

Обход бинарного дерева в ширину:

`пробел`

Изображение леса и бинарного дерева совпадают:

Рисунок 6 - лес и бинарное дерево(2).

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_4_methods.h"
#include "lr_4_methods.cpp"

int main(int argc, const char *argv[])
{
   bool flag_bypass=false;
   bool flag_bin_str=false;



   for(int i=1;i<argc;i++)
   {
      if(!strcmp("gui",argv[i]))
      {
         system("open http://163.172.163.152:3000/");
         return 0;
      }
      if(!strcmp("bypass",argv[i]))
         flag_bypass=true;
      if(!strcmp("bin_str",argv[i]))
         flag_bin_str=true;
   }



   string forest;
   string bypass_width = "";
   string bin_tree = "(";
   cout<<"Введите скобочное представление леса:\n";
   cin >> forest;
   int *lvl = new int[100];
   string *arr_val = new string[100];
   int i_lvl = 1;
   int index = 1;

   TREE *tree = NULL;
   lvl[0] = -1;
   lvl[i_lvl] = -1;
   //test
   int err=test_4(forest);
   if(err)
   {
      cout << "Errors: ErrNum#" << err << endl;
      return 0;
```

```
    }
    create_4(i_lvl, lvl, arr_val, forest);

    //add(tree);
    add_4(tree, i_lvl, lvl, arr_val, index);

    //bypass
    if(flag_bypass)
    {
        bypass_4(tree, bin_tree);
        cout<<"Скобочное представление бинарного дерева:\n"<<bin_tree<<endl;
    }
    //bin_str
    if(flag_bin_str)
    {
        width_4(tree,bypass_width);
        cout<<"Обход в ширину бинарного дерева:\n"<<bypass_width<<endl;
    }
    return 0;
}
```

lr_4_methods.cpp

```
#include "lr_4_methods.h"
struct TREE
{
    string value;
    TREE *left;
    TREE *right;
};

struct Q
{
    TREE **leaf;
    int size;
};

int test_4(string forest)
{
    int open = 0;
    int close = 0;

    for (int i = 0; i < forest.size(); i++)
    {
        //    2(1)
        if (forest[i] == '(')
            open++;
```

```
        if (forest[i] == ')')
            close++;
    }
    for (int i = 0; i < forest.size() - 1; i++)
    {
        //    1
        if (forest[i] == '(' && (forest[i + 1] == '(' || forest[i + 1] == ')'))
        {
            return 1;
        }
        //4
        if (forest[i] == ')' && (forest[i + 1] != '(' && forest[i + 1] != ')'))
        {
            return 4;
        }
    }

    //    2(2)
    if (open != close)
        return 2;

    //    3
    if (forest[0] != '(')
        return 3;
    return 0;
}

void bypass_4(TREE *&tree, string &bin_str)
{
    bin_str += tree->value;
    if (tree->left)
    {
        bin_str += "(";
        bypass_4(tree->left, bin_str);
    }

    if (tree->right)
    {
        bin_str += "(";
        bypass_4(tree->right, bin_str);
    }
    bin_str += ")";
}
int create_4(int &ind, int *&lvl, string *&arr_val, string forest)
{
    string value;
```

```
    char buffer;
    int i = 0;
    for (; i < forest.length(); i++)
    {
        buffer = forest[i];
        if (value != "" && (buffer == ')' || buffer == '('))
        {
            lvl[ind + 1] = lvl[ind];
            arr_val[ind] = value;
            value = "";
            ind++;
        }
        if (buffer == '('
            lvl[ind] += 1;
        if (buffer == ')')
            lvl[ind] -= 1;
        if (buffer != ')' && buffer != '('
            value += buffer;
        }
    return 0;
}
int add_4(TREE *&tree, int top, int *lvl, string *arr_val, int &index)
{
    int lvl_index = lvl[index];
    int next_lvl = -2;
    while (true)
    {
        if (index == top)
            return -2;
        if (lvl[index] >= lvl[index - 1])
        {
            next_lvl = push_4(tree, top, lvl, arr_val, index);
            if (lvl_index == next_lvl && index != top)
                next_lvl = push_4(tree, top, lvl, arr_val, index);
        }
        else
            return lvl[index];

        if (next_lvl == lvl_index w&& index != top)
        {
            next_lvl = push_4(tree, top, lvl, arr_val, index);
        }
        else
            return lvl[index];
    }
}
```

```
int push_4(TREE *&tree, int top, int *lvl, string *arr_val, int &index)
{
   TREE *NEW = new TREE;
   NEW->left = NULL;
   NEW->right = NULL;
   NEW->value = arr_val[index];
   if (tree == NULL)
   {
      tree = NEW;
   }
   else
   {
      if (tree->left == NULL)
         tree->left = NEW;
      else
         tree->right = NEW;
   }
   index += 1;
   int next_lvl = add_4(NEW, top, lvl, arr_val, index);
   return next_lvl;
}

void width_4(TREE *tree, string &bypass_width)
{
   if (tree == NULL)
      return;
   Q q;
   q.leaf = new TREE *[100];
   q.leaf[0] = tree;
   q.size = 1;
   while (q.size != 0)
   {
      TREE **new_leaf = new TREE *[100];
      int new_size = 0;
      for (int i = 0; i < q.size; i++)
      {
         if (q.leaf[i]->left != NULL)
         {
            new_leaf[new_size] = q.leaf[i]->left;
            new_size++;
            if (q.leaf[i]->right != NULL)
            {
               new_leaf[new_size] = q.leaf[i]->right;
               new_size++;
            }
```

```
            }
            bypass_width += q.leaf[i]->value;
            bypass_width += "\n";
        }
        q.size = new_size;
        q.leaf = new_leaf;
    }
}
```

lr_4_methods.h

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

using namespace std;

struct TREE;
struct Q;

int add_4(TREE *&tree, int top, int *lvl, string *arr_val, int &index);
int push_4(TREE *&tree, int top, int *lvl, string *arr_val, int &index);
int create_4(int &ind, int *&lvl, string *&arr_val, string forest);
void bypass_4(TREE *&tree, string &bin_str);
int test_4(string forest);
void width_4(TREE *tree, string &bypass_width);
int test(string str);
```