**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. 7382 Дрозд А. С.

Преподаватель Фирсов М. А.

Санкт-Петербург

2018

**Задание.**

8. Рассматриваются бинарные деревья с элементами типа char. Заданы

перечисления узлов некоторого дерева b в порядке ЛКП и ЛПК. Требуется:

а) восстановить дерево b и вывести его изображение;

б) перечислить узлы дерева b в порядке КЛП.

**Пояснение задания.**

На вход программе подаются 2 строчки: ЛКП и ЛПК кокого-то бинарного дерева. Программа должна вывести КЛП и изображение этого бинарного дерева.

**Описание алгоритма.**

Программа считывает ЛКП и ЛПК. Создает вектор из структур, в который после будет записано дерево. Вызывает функцию **maker** обрабатывающую строчки. В функции строки проверяются на корректность,если проверка пройдена функция берет последний символ ЛПК, который является корнем дерева, находит этот символ в ЛКП, и делает строку на 2 части относительно этого символа. Левая часть разделенной строчки это ЛКП правой ветви, правая соответственно правой.

Подсчитывается длинна правой части ЛКП и от ЛПК отделяется часть такой же длины. В ЛПК так же как и в ЛКП левая часть строчки это ЛПК левой ветви, а правая правой. Из полученных ЛПК и ЛКП правой и левой ветви рекурсивно извлекаем данные считая их отдельными деревьями. Все найденные корни в определенной последовательности записываются в вектор.

**Описание структур**

В программе использовалась одна структура:

* struct bin\_tree

В структуре находится 3 символа. Символ char root содержит корень дерева. Символ char left содержит корень левого поддерева. Символ char right содержит корень правого поддерева.

**Описание функций**

В программе использовалось 7 функций:

* **int tweest(int lvl)**

Принимает целое число и возвращает значение двойки в степени принятого числа. При принятом отрицательном числе возвращает 1.

* **int start\_checking(char\* lkp,char\* lpk)**

Принимает 2 строки. Проверяет, могут ли эти строки быть бинарным деревом относительно количества символов и их повторения.

* **int maker(char\* lkp,char\* lpk,vector<bin\_tree>\* vec,int lvl,int step)**

Принимает 2 строки,указатель на вектор структур **bin\_tree**, 2 числа. Функция заполняет вектор данными строки, считается что 2 строки это ЛКП и ЛПК бинарного дерева. Функция выполняет проверку строки с помощью предыдущей функции. Если проверка пройдена то функция записывает в в вектор корень и 2 корня его поддеревьев, а после рекурсивно запустит функцию для поддеревьев. Числа **lvl** и **step** нужны для работы с вектором.

* **void klp(vector<bin\_tree>\* vec,char\* str,int lvl,int step)**

Принимает указатель на вектор структур **bin\_tree** , строку и 2 числа. Функция записывает в строку КЛП бинарного дерева, находящегося в векторе. Числа **lvl** и **step** нужны для работы с вектором.

* **shift(char\* str,int start)**

Принимает строку и число. Сдвигает строку на один в право на 1 символ, начиная от **start**-ового символа.

* **visuale(vector<bin\_tree>\* vec,char\* str)**

Принимает указатель на вектор структур **bin\_tree** и строку. Функция записывает в строку визуализация бинарного дерева, расположенного в векторе.

* **int main()**

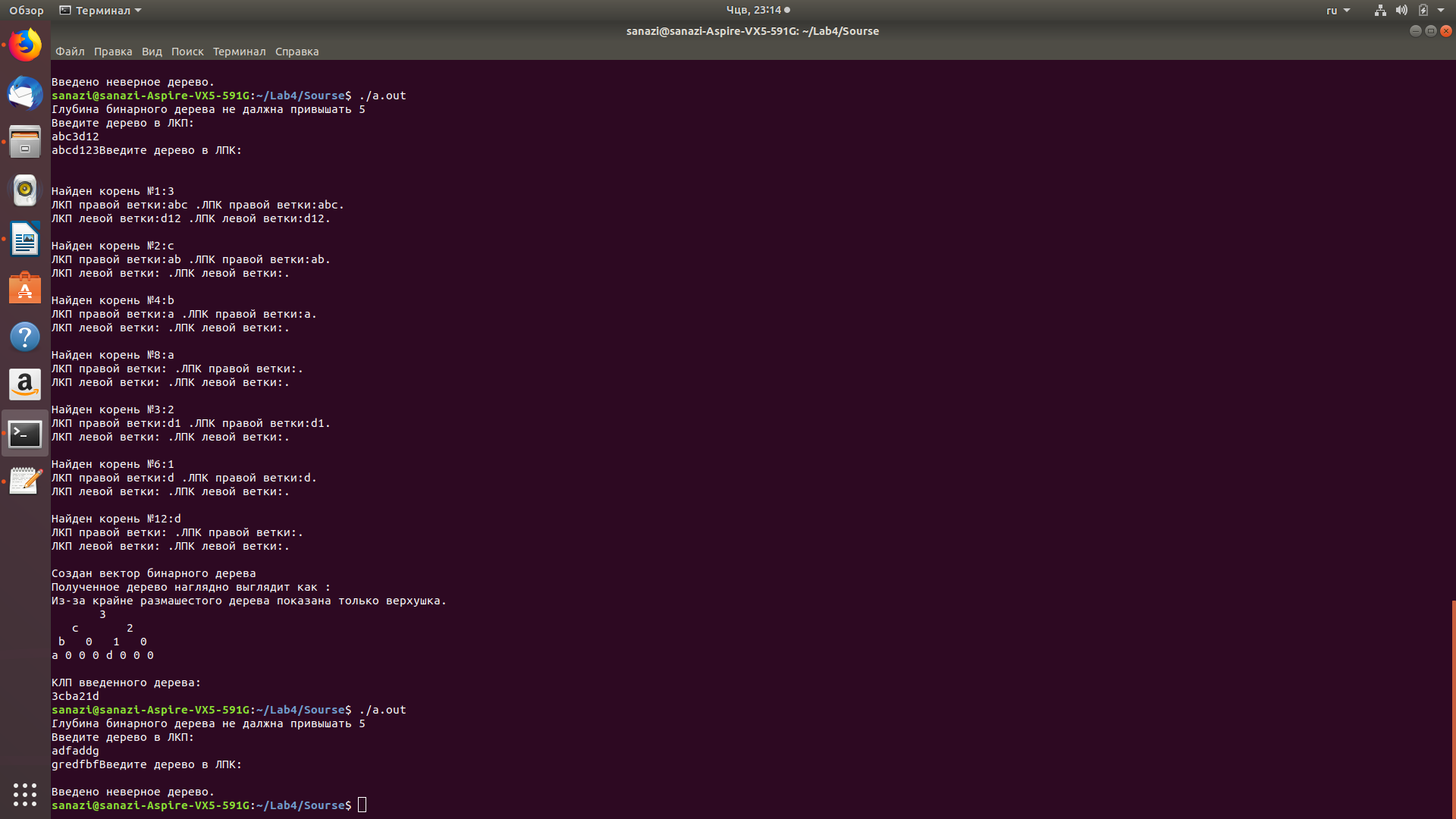
Главная функция программы. Сначала выводит огранечения программы. После просит ввести данные и считывает их. После получения данных пытается записать в вектор бинарное дерево, попутно выводя информационные сообщения. Если входные данные неверны, выведет сообщение об ошибке. Если получилось записать бинарное дерево в вектор, функция выведет ее визуализацию(с некоторыми ограничениями). В конце программы создается и выводится КЛП. Вектор очищается, динамическая память высвобождается.

**Тестирование.**

|  |  |
| --- | --- |
| Вводимые данные | Результат: |
| abcdefghijkl  abcdefghijkl | Lkjihgfedcba |
| abcdefg&$  abcefg&$d | dcba$&gfe |
| abcdefghfi  abcdefghif | Введено неверное дерево. |
| abc3d12  abcd123 | 3cba21d |
| adfaddg  gredfbf | Введено неверное дерево. |

**Пример визуализации дерева:**

Тестирование №4



**Вывод.**

В процессе выполнения лабораторной работы были изучены деревья, а также способы работы с ними. Был рассмотрен способ хранения бинарного дерева в векторе структур, а также способы визуализации бинарного дерева.

**Приложение 1. Код программы.**

* **[test.sh](https://github.com/makometr/AiSD/pull/37/files" \l "_blank)**

gcc ./Sourse/main.c

echo -e '\_\_\_\_\_\_\_\nTest 1:'

cat ./Tests/test1.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < Tests/test1.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 2:'

cat ./Tests/test2.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test2.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 3:'

cat ./Tests/test3.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test3.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 4:'

cat ./Tests/test4.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test4.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 5:'

cat ./Tests/test5.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test5.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 6:'

cat ./Tests/test6.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test6.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 7:'

cat ./Tests/test7.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test7.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 8:'

cat ./Tests/test8.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./a.out < ./Tests/test8.txt

* **main[.](https://github.com/makometr/AiSD/pull/37/files" \l "_blank)cpp**

#include <iostream>

#include <fstream>

#include <cmath>

#include <vector>

using namespace std;

struct bin\_tree{ //Структура узла бинарного дерева

char root; //Корень

char left; //Левое плечо

char right; //Правое плечо

};

int tweest(int lvl) //Функция подсчета 2^lvl

{

if(lvl>=0) //2^0=1

return 1;

int a=2;

while(lvl!=1)

{

a=a\*2;

lvl--;

}

return a;

}

int start\_checking(char\* lkp,char\* lpk) //Первоначальная проверка входных данных

{

int size1=0,size2=0;

while(lkp[size1]!='\0') //Подсет размера строк

size1++;

while(lpk[size2]!='\0')

size2++;

if(size1!=size2) //Проверка ,одинаковое ли количество узлов в ЛПК и ЛКП

return 1;

size1=0;

while(lkp[size1]!='\0') //Проверка есть ли в строке ЛПК все узлы из строки ЛКП

{

size2=0;

while(lpk[size2]!='\0' && lpk[size2]!=lkp[size1])

size2++;

if(lpk[size2]=='\0') //Если прошли строку но не нашли нужный узел

return 2;

size1++;

}

size1=0;

size2=0;

while(lkp[size1]!='\0') //Проверка строки ЛКП на одноименные узлы

{

size2=size1+1;

while(lkp[size2]!='\0')

{

if(lkp[size2]==lkp[size1])

return 3;

size2++;

}

size1++;

}

return 0;

}

int maker(char\* lkp,char\* lpk,vector<bin\_tree>\* vec,int lvl,int step) //Функция для создания вектора бинарного дерева

{

if(start\_checking(lkp,lpk)!=0)

return 1;

if(lkp[0]=='\0' || lpk[0]=='\0') //Если строка нулевая заканчиваем рекурсию

return 0;

int count=0;

while(lpk[count]!='\0') //Доходим до конца строки ЛКП

count++;

count--; //Выбираем последний символ ЛКП

int count1=0;

char litr=lpk[count]; //Запоминаем последний символ ЛКП

while(lkp[count1]!=lpk[count]) //Находим последний символ ЛКП в ЛПК

count1++;

lkp[count1]='\0'; //Удаляем найденные

lpk[count]='\0'; //символы из строк

int save=count;

while(count!=count1) //Разделяем строку ЛПК онтосительно последнего символа ЛКП

{

lpk[count]=lpk[count-1];

count--;

}

lpk[count]='\0';

struct bin\_tree \*wood;

int size=vec->size();

if(size<tweest(lvl)+step-1) //Проверяем достаточно ли длинный вектор для нас, если недостаточен

vec->resize(tweest(lvl)+step);

wood=&vec[0][tweest(lvl)+step-1];

//Забиваем данные бинарного дерева в вектор

wood->root=litr;

wood->left=lpk[count-1];

wood->right=lpk[save];

cout<<endl<<"Найден корень №"<<tweest(lvl)+step<<":"<<litr<<endl;

cout<<"ЛКП правой ветки:"<<lkp<<" .ЛПК правой ветки:"<<lpk<<"."<<endl;

cout<<"ЛКП левой ветки:"<<lkp+count+1<<" .ЛПК левой ветки:"<<lpk+count+1<<"."<<endl;

int m1=maker(lkp,lpk,vec,lvl+1,step\*2); //Рекурсивная запись правого и левого узла

if(m1==1)

return 1;

m1=maker(lkp+count+1,lpk+count+1,vec,lvl+1,step\*2+1);

if(m1==1)

return 1;

return 0;

}

void klp(vector<bin\_tree>\* vec,char\* str,int lvl,int step) //Функция написания КЛП

{

struct bin\_tree \*wood=&vec[0][tweest(lvl)+step-1];

int count=0;

while(str[count]!='\0') //Доходим до конца строки

count++;

str[count]=wood->root; //Записываем корень

str[count+1]='\0';

if(wood->left!='\0')

klp(vec,str,lvl+1,step\*2); //Записываем левый узел

if(wood->right!='\0')

klp(vec,str,lvl+1,step\*2+1); //Записываем правый узел

}

void shift(char\* str,int start) //Функция сдвигает строку на один в право на 1 символ, начиная от start-ового символа.

{

int count=start;

while(str[count]!='\0') //Доходим до конца строки

count++;

while(count!=start) //Сдвигаем строку

{

str[count+1]=str[count];

count--;

}

str[count+1]=str[count];

}

void visuale(vector<bin\_tree>\* vec,char\* str) //Функция создоет строку визуализации дерева

{

int check=1;

int start\_lvl=0;

int lvl=0;

int step=0;

int count=0;

int count1=0;

int count2=0;

struct bin\_tree \*wood=&vec[0][0]; //Проверка первого узла дерева

if(wood->root=='\0')

check=0;

str[0]='\0';

while(check!=0 && start\_lvl<7) //Пока у корней не будет поддеревьев и пока глубина дерева меньше 7

{

check=0;

while(lvl!=0) //Пока не дойдем до незаписонного уровня дерева

{

count=0;

while(count!=tweest(start\_lvl-lvl)) //Пока не пройдем уровень

{

if(str[count1]!=' ' && str[count1]!='\n') //Если встретили символ

{

while(count2!=tweest(lvl-1)) //Добавляем отступы

{

shift(str,count1);

str[count1]=' ';

count1++;

count2++;

}

count1++;

count2=0;

while(count2!=tweest(lvl-1)) //Добавляем отступы

{

shift(str,count1);

str[count1]=' ';

count1++;

count2++;

}

count1--;

count2=0;

count++;

}

count1++;

}

while(str[count1]!='\n') //Доходим до последнего символа уровня

count1++;

count1++;

lvl--;

}

while(step!=tweest(start\_lvl)) //Пока не запишем все узлы уровня

{

wood=&vec[0][tweest(start\_lvl)+step-1];

str[count1+2]='\0';

if(wood->left!='\0' || wood->right!='\0')

check=1;

if(wood->root!='\0')

str[count1]=wood->root;

else

str[count1]='0';

str[count1+1]=' ';

count1=count1+2;

step++;

}

str[count1-1]='\n';

str[count1]='\0';

step=0;

start\_lvl++;

lvl=start\_lvl;

count1=0;

}

if(start\_lvl=7) //Если визуализация закончилась из-за большой глубины дерева выводим дополнительное сообщение

cout<<"Из-за крайне размашестого дерева показана только верхушка."<<endl;

}

int main() //Основная функция

{

char str\_lkp[100];

char str\_lpk[100];

vector <bin\_tree>\* tree=new vector <bin\_tree>(128);

cout << "Глубина бинарного дерева не далжна привышать 5" << endl;

cout << "Введите дерево в ЛКП: " << endl;

cin >> str\_lkp; //Ввод ЛКП

cout << "Введите дерево в ЛПК: " << endl;

cin >> str\_lpk; //Ввод ЛПК

int check=maker(str\_lkp,str\_lpk,tree,0,0);

if(check!=0) //Проверка, было ли дерево верным

{

cout << "Введено неверное дерево." << endl;

return 0;

}

cout << endl << "Создан вектор бинарного дерева" << endl;

char str\_tree[100000];

cout<<"Полученное дерево наглядно выглядит как :"<<endl;

visuale(tree,str\_tree);

cout<<str\_tree<<endl;

char str[100];

str[0]='\0';

klp(tree,str,0,0); //Создание КЛП

cout << "КЛП введенного дерева:"<<endl;

cout<<str<<endl;

tree->clear(); //Отчистка вектора

free(tree); //Высвобождение памяти

}