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: Программирование алгоритмов с бинарными деревьями

Студент гр. 7383 _____ Тян Е.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург
2018

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

Цель работы: познакомиться с такой часто используемой на практике, особенно при решении задач кодирования и поиска, нелинейной структурой данных, как бинарное дерево, способами ее представления и реализации, получить навыки решения задач и обработки бинарных деревьев.

Формулировка задачи: рассматриваются бинарные деревья с элементами типа *char*. Заданы перечисления узлов некоторого дерева *b* в порядке КЛП и ЛКП. Требуется:

а) восстановить дерево *b* и вывести его изображение;

б) перечислить узлы дерева *b* в порядке ЛПК.

Входные данные: перечисление узлов в порядке КЛП и ЛКП.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В данной работе используются главная функция (int main()) и дополнительные функции (base RootBT(binTree b), binTree ConsBT(const base &x, binTree &lst, binTree &rst), binTree EnterBT(char* arrayKLP, char* arrayLKP, int &i), binTree Left (binTree b), binTree Right(binTree b), void outBT(binTree b, int i, bool left, binTree ptr), void LPK(binTree b), bool check(char* arrayKLP, char* arrayLKP)).

Параметры передаваемые в функцию base RootBT(binTree b):

- b – узел дерева.

Параметры передаваемые в функцию binTree ConsBT(const base &x, binTree &lst, binTree &rst):

- x – элемент типа char;
- lst – левое поддерево;
- rst – правое поддерево.

Параметры передаваемые в функцию binTree EnterBT(char* arrayKLP, char* arrayLKP, int &i):

- arrayKLP – перечисление узлов в порядке КЛП;
- arrayLKP – перечисление узлов в порядке ЛКП.

Параметры передаваемые в функцию binTree Left (binTree b):

- b – узел дерева.

Параметры передаваемые в функцию binTree Right(binTree b):

- b – узел дерева.

Параметры передаваемые в функцию void outBT(binTree b, int i, bool left, binTree ptr):

- b – узел дерева;
- i – счетчик для корректного вывода отступов;
- left – флаг, показывающий был при родитель левым поддеревом;
- ptr – узел, хранящий корень.

Параметры передаваемые в функцию void LPK(binTree b):

- b – узел дерева.

Параметры передаваемые в функцию bool check(char* arrayKLP, char* arrayLKP):

- arrayKLP – перечисление узлом в порядке КЛП;
- arrayLKP – перечисление узлов в порядке ЛКП.

В функции main() выводится меню на консоль, где можно выбрать число, соответствующее выполняемой операции. Считывается целое число и, при помощи switch(), выбирается необходимая опция. При выборе «1» пользователь вводит перечисление узлов в порядке КЛП и ЛКП. При выборе «2» программа считывает перечисление узлов в порядке КЛП и ЛКП из файла «test1.txt». При выборе «3» программа завершает работу. При выборе другого значения программа выводит сообщение: «Проверьте введенные данные и повторите попытку» и ожидает дальнейших указаний. Считывание из файла отличается от считывания с консоли тем, что с консоли считывается оба перечисления в фиктивную строку, и, потом, в две строки записываются перечисления узлов в КЛП и ЛКП порядке по очереди, в файле же перечисления записываются сразу же в сами строки для перечислений. Далее в обоих случаях программа ведет себя одинаково. После считывания строк программа проверяет равны ли по длине они или нет, если они не равны, то выводится ошибка и программа прекращает свою работу. Если же строки равны по длине, то вызывается функция check(char* arrayKLP, char* arrayLKP), которая проверяет схоже ли содержимое строк, если содержимое различается, то программа выводит ошибку и завершает работу. Если содержимое схоже, то программа вызывает функцию EnterBT(char* arrayKLP, char* arrayLKP, int &i), которая строит дерево с помощью функции ConsBT(const base &x, binTree &lst, binTree &rst). Далее выводится дерево на консоль при помощи функции outBT(binTree b, int i, bool left, binTree ptr), где используются функции Left (binTree b) и Right(binTree b) для перехода к левому и правому подогреву соотвественно. Также для вывода самого значения, хранящегося в структуре, используется функция RootBT(binTree b). Далее вызывается функция LPK(binTree b), которая выполняет ЛПК – обход

дерева и выводит перечисление узлов в данном порядке на консоль. После программа заканчивает работу и ждет дальнейших указаний от пользователя.

Для более понятного описания реализации задачи рассмотрим пример работы программы. Пусть были введены перечисления: abcgdef cgbdeaf. Тогда можно разделить каждую строку на три подстроки как это показано на рис. 1. Далее выполняет вывод дерева на консоль и вывод перечисления узлов в ЛПК порядке.

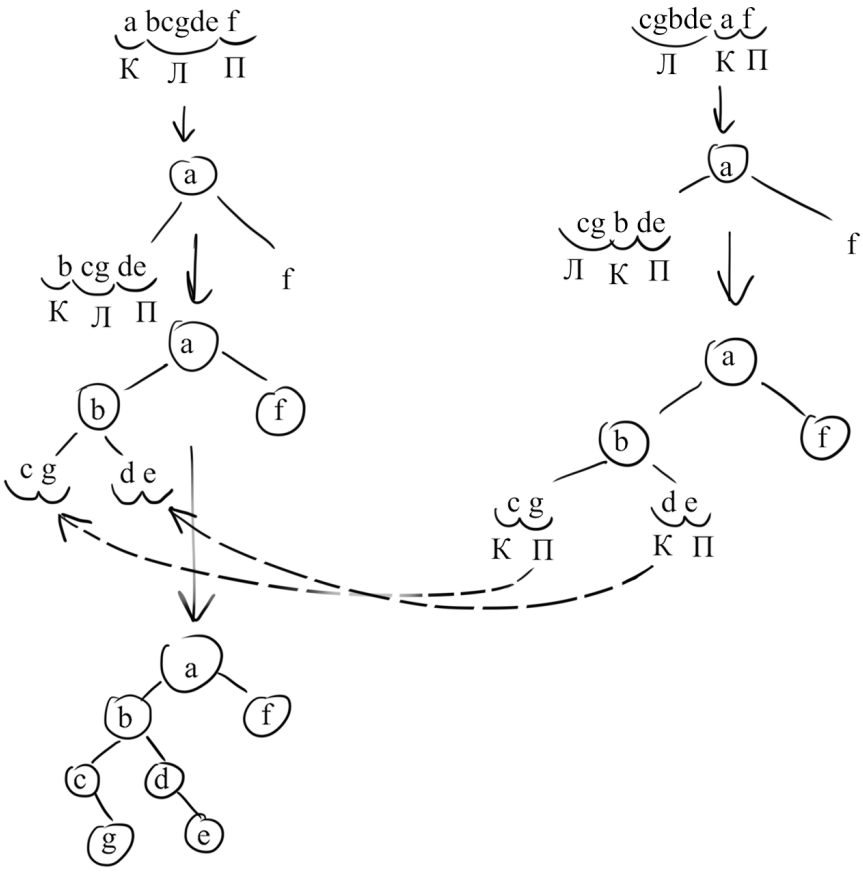

Рисунок 1 — представление бинарного дерева через перечисления узлов в КЛП и ЛКП порядках

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе G++ в OS Linux Ubuntu 12.04. Программа может быть скомпилирована с помощью команды:

g++ <имя файла>.cpp

Тестовые случаи представлены в Приложении А.

Исходя из тестовых случаев можно увидеть, что тестовые случаи не выявили некорректной работы программы, что говорит о том, что по результатам тестирования было показано: поставленная задача была выполнена.

4. ВЫВОД

В ходе выполнения лабораторной работы было выполнено ознакомление с часто используемой на практике, особенно при решении задач кодирования и поиска, нелинейной структуры данных, как бинарное дерево, способами её представления и реализации, получены навыки решения задач и обработки бинарных деревьев на языке C++.

Была реализована программа восстанавливающая дерево по заданным перечислениям узлов в порядке КЛП и ЛПК и выводящая его изображение, а также перечисляющая узлы дерева в порядке ЛПК.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

№	Ввод	Вывод
1	abcgdef cgbdeaf	.---c `---g .---b `---d `---e ---a `---f gcedbfa
2	apelri aelpri	---a .---e `---l `---p `---r `---i leirpa
3	adejkflbghcimn djkelfaghbmnic	.---d .---j `---k `---e .---l `---f ---a .---g `---h `---b .---m `---n .---i `---c kjlfedhgnmicba
4	5314972 1345792	.---1 .---3 `---4 ---5 .---7 `---9 `---2 1437295

5	smadntzx admnstxz	.---a `---d .---m `---n ---s `---t .---x `---z danmxzts
6	Vafdvfdjh jdfhvkjsdvnkdj	Strings are not the same.

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp:

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <fstream>

using namespace std;

typedef char base;
struct node{
   base info;
   node *lt;
   node *rt;
};
typedef node *binTree;

base RootBT(binTree b){
   if(b == NULL){
      cerr << "Error: RootBT(null) \n";
      exit(1);
   }else
      return b->info;
}

binTree ConsBT(const base &x, binTree &lst, binTree &rst){
   binTree p;
   p = new node;
   if ( p != NULL) {
      p ->info = x;
      p ->lt = lst;
      p ->rt = rst;
      return p;
   }else {
      cerr << "Memory not enough\n";
      exit(1);
   }
}

binTree EnterBT(char* arrayKLP,char* arrayLKP, int &i){
   binTree p,q;
```

```
    p=NULL;
    q=NULL;
    char c,c1;
    if(i<strlen(arrayKLP) && strlen(arrayLKP)!=0){
        c=arrayKLP[i];
        c1=arrayLKP[strlen(arrayLKP)-1-i];
        for(int j=0;j<strlen(arrayLKP);j++){
            if(arrayKLP[i]==arrayLKP[j]){
                char* gap_left_str=(char*)calloc(20,sizeof(char));
                strncpy(gap_left_str,arrayLKP,j);
                char* gap_right_str=(char*)calloc(20,sizeof(char));
                strncpy(gap_right_str,arrayLKP+j+1,strlen(arrayLKP)-j-1);
                i++;
                p=EnterBT(arrayKLP,gap_left_str,i);
                q=EnterBT(arrayKLP,gap_right_str,i);
                return ConsBT(c,p,q);
            }
        }
    }else
        return NULL;
}

binTree Left(binTree b){
    if (b == NULL){
        cerr << "Error: Left(null) \n";
        exit(1);
    }else
        return b->lt;
}

binTree Right(binTree b){
    if (b == NULL){
        cerr << "Error: Right(null) \n";
        exit(1);
    }else
        return b->rt;
}

void outBT(binTree b,int i,bool left, binTree ptr){
    if (b!=NULL){
        outBT(Left(b),i+4,true,b);
        if(ptr==NULL){
```

```
            for(int k=0;k<i;k++)
                cout<<" ";
            cout<<"---";
            cout << RootBT(b)<<endl;
        }else if(left){
            for(int k=0;k<i;k++)
                cout<<" ";
            cout<<".---";
            cout << RootBT(b)<<endl;
        }else{
            for(int k=0;k<i;k++)
                cout<<" ";
            cout<<"`---";
            cout << RootBT(b)<<endl;
        }
        outBT(Right(b),i+4,false,b);
    }else
        return;
}

void LPK(binTree b){
    if(b!=NULL){
        LPK(Left(b));
        LPK(Right(b));
        cout<<RootBT(b);
    }else
        return;
}

bool check(char* arrayKLP,char* arrayLKP){
    int k=0;
    for(int i=0;i<strlen(arrayKLP);i++){
        for(int j=0;j<strlen(arrayLKP);j++){
            if(arrayLKP[j]==arrayKLP[i])
                k++;
        }
    }
    if(k==strlen(arrayKLP))
        return true;
    else
        return false;
}
```

```
int main(){
    int num=0;
    while(num != 3){
        char* arrayKLP=(char*)calloc(20,sizeof(char));
        char* arrayLKP=(char*)calloc(20,sizeof(char));
        cout << "Выберите дальнейшие действия и введите цифру:"<<endl;
        cout << "1. Ввести перечисления узлов вручную."<<endl;
        cout << "2. Считать перечисления узлов из файла test1.txt."<<endl;
        cout << "3. Завершить работу."<<endl;
        cin >> num;
        switch(num){
            case 1:{
                getchar();
                char c;
                c=getchar();
                int j=0;
                while(!isspace(c)){
                    arrayKLP[j]=c;
                    j++;
                    c=getchar();
                }
                c=getchar();
                j=0;
                while(!isspace(c)){
                    arrayLKP[j]=c;
                    j++;
                    c=getchar();
                }
                int i=0;
                binTree b;
                if(strlen(arrayKLP)==strlen(arrayLKP)){
                    if(check(arrayKLP,arrayLKP)==true){
                        b=EnterBT(arrayKLP,arrayLKP,i);
                        i=0;
                        binTree ptr;
                        ptr=NULL;
                        outBT(b,i,false,ptr);
                        LPK(b);
                        cout<<endl;
                    }else{
                        cerr<<"Strings are not the same."<<endl;
```

```
            exit(1);
        }
    }else{
        cerr<<"Strings are not the same."<<endl;
        exit(1);
    }
    free(arrayLKP);
    free(arrayKLP);
    break;
}
case 2:{
    string array;
    ifstream infile("test1.txt");
    getline(infile,array);
    array=array+'\n';
    int k=0;
    while(!isspace(array[k])){
        arrayKLP[k]=array[k];
        k++;
    }
    arrayKLP[strlen(arrayKLP)]='\0';
    k++;
    int j=0;
    while(!isspace(array[k])){
        arrayLKP[j]=array[k];
        k++;
        j++;
    }
    arrayLKP[strlen(arrayLKP)]='\0';
    int i=0;
    binTree b;
    if(strlen(arrayKLP)==strlen(arrayLKP)){
        if(check(arrayKLP,arrayLKP)==true){
            b=EnterBT(arrayKLP,arrayLKP,i);
            i=0;
            binTree ptr;
            ptr=NULL;
            outBT(b,i,false,ptr);
            LPK(b);
            cout<<endl;
        }else{
            cerr<<"Strings are not the same."<<endl;
```

```
                exit(1);
            }
        }else{
            cerr<<"Strings are not the same."<<endl;
            exit(1);
        }

        free(arrayLKP);
        free(arrayKLP);
        break;
    }
    case 3:
        return 0;
    default:
        cerr << "Проверьте введенные данные и повторите попытку." << endl;
        break;
  }
 }
 return 0;
}
```