**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: Бинарные деревья и лес

|  |  |  |
| --- | --- | --- |
| Студентка гр. 7382 |  | Головина Е.С. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2018

**Цель работы.**

Изучить структуру бинарного дерева и леса. Научиться выполнят операции с бинарными деревьями и лесом.

**Задание.**

Вариант 6-в:

Для заданного бинарного дерева:

а) получить лес, естественно представленный этим бинарным деревом;

б) вывести изображение бинарного дерева и леса;

в) перечислить элементы леса в горизонтальном порядке (в ширину)

**Постановка задачи.**

Нужно написать программу, которая считает бинарное дерево, получит из него лес и выведет их графическое представление.

**Описание алгоритма.**

1. Программа запрашивает как пользователь хочет ввести данные: из файла или из терминала. Если из файла — п.2, если из терминала — п.3.

2. Программа запрашивает название файла, данные откуда надо взять и и считывает строку из файла, если он существует, и закрывает файл. Далее п.4.

3. Программа считывает строку. Далее п.4.

4. Начинается запись элементов бинарного дерева в соответствующую структуру.

5. Происходит вывод графического представления бинарного дерева в виде уступчатого списка.

6. Происходит конвертация бинарного дерева в лес с дальнейшем выводом графического представления леса на экран.

7. Начинается обход леса по горизонтали с последующим выводом получившейся последовательности.

**Методы класса, функции и структуры данных:**

1. struct bt\_element{

Type value;

int left;

int right;

};

value — значение хранящееся в листе бинарного дерева;

left — индекс левого сына в массиве;

right — индекс правого сына в массиве.

2. struct binTree{

int size;

bt\_element \* storage;

int depth;

};

size — размер массива для хранения бинарного дерева;

storage — указатель на массив элементов бинарного дерева;

depth — глубина бинарного дерева.

3. **void binTree::print\_tree(int index,int depth){**

index — текущая позиция в бинарном дереве.

depth — глубина дерева, включающая текущий лист и сыновей.

Метод печатает представление бинарного дерева в виде уступчатого списка.

4. **struct forest\_element{**

**Type value;**

**int \* sons;**

**};**

value — значение хранящееся в листе леса;

sons — указатель на массив индексов сыновей элемента.

5. **struct Forest{**

**int size;**

**int max\_sons;**

**forest\_element \* storage;**

**int depth;**

**};**

size — размер массива для хранения леса;

max\_sons — максимальное число сыновей для конкретного леса;

storage — указатель на массив элемнтов леса;

depth — глубина леса.

6. **void Forest::print\_forest(int index,int depth)**

index — текущая позиция в лесе.

depth — глубина леса, включающая текущий лист и сыновей.

Метод печатает представление леса в виде уступчатого списка.

7. **void print\_horizontal(std::string \* out,int index, int level)**

out — указатель на массив строк, в которых содержатся перечисление элементов каждого уровня;

index — текущая позиция в лесе.

level — текущий уровень.

8. **int fill\_from\_str(const binTree<char> \* bt,const std::string str\_bt, int \* position, int root);**

bt — указатель на бинарное дерево;

str\_bt — строка, в которой записано скобочное представление бинарного дерева;

position — текущая позиция в строке;

root — отец, к которому присоединяем сыновей (если они есть).

9. **void convert\_bt\_to\_forest(const binTree<char> \* bt, Forest<char> \* bt\_forest,int father,int num\_son,int \* posf,int pos);**

bt — указатель на бинарное дерево;

bt\_forest - указатель на лес;

father — отец текущего элемента;

num\_son — количество братьев элемента и сыновей отца;

posf — текущая позиция в лесе;

pos — текущая позиция в бинарном дереве.

Тестирование.

Было сделано 9 тестов для демонстрации и проверки работы программы.

|  |  |  |
| --- | --- | --- |
| Тест № | Данные | Результат |
| 1 | (a (b) (c)) | Binary tree:  a--  b-  c-  Forest:  a--  b-  c--  Forest in horizontal order: acb |
| 2 | (a (b #(d # (e))) (c)) | Binary tree:  a----  b---  d--  e-  c---  Forest:  a----  b---  d---  e---  c----  Forest in horizontal order: acbde |
| 3 | (a (b (s) (d (k(l) (z)) (e))) (c (f) (g #(i)) ) ) | Binary tree:  a-----  b----  s---  d---  k--  l-  z-  e--  c----  f---  g---  i--  Forest:  a-----  b----  s---  d----  k---  l--  z---  e----  c-----  f----  g-----  i-----  Forest in horizontal order: acgibdefskzl |
| 4 | (a #(b#(c#(d#(e))))) | Binary tree:  a-----  b----  c---  d--  e-  Forest:  a-----  b-----  c-----  d-----  e-----  Forest in horizontal order: abcde |
| 5 | (a (b(c(d(e))))) | Binary tree:  a-----  b----  c---  d--  e-  Forest:  a-----  b----  c---  d--  e-  Forest in horizontal order: abcde |
| 6 | (a (b) (c) | Error! Brackets aren't closed!  Program terminated! |
| 7 | sdsjdhs | Error! It is not a tree!  Program terminated! |
| 8 | s | Error! It is not a tree!  Program terminated! |
| 9 | a(a)(a)) | Error! Brackets aren't closed!  Program terminated! |

Для более наглядной демонстрации работы программы был создан bash-скрипт, последовательно выводящий содержимое очередного теста и результат работы программы для этого теста. Код bash-скрипта представлен в приложении Г, результат работы скрипта — для тестов 1 и 9 представлен в приложении Д.

Выберем 1 тест для детального обзора. Программа считывает скобочное представление бинарного дерева в строку из терминала или из файла в зависимости от выбора пользователя.

1. Перед заполнением бинарного дерева идут подготовительные работы. Проверяется все ли скобки закрыты, высчитывается максимальная глубина вложенности. В данном случае все корректно.

2. По полученным данным глубина 2 и по формуле считается максимальное количество ячеек массива, которые могут понадобиться для хранения бинарного дерева этой глубины.

3. Создается объект класса бинарное дерево и начинается его заполнение. Учитываем, что знак # указывает на отсутствие левого сына.

4. Запускается метод класса бинарное дерево, отвечающий за печать элементов. Печатается уступчатое представление бинарного дерева.

5. Создаем лес, зная параметры бинарного дерева и начинаем копировать элементы из бинарного дерева в лес, учитывая, что по правилу перевода в лес правый сын является братом, а левый сын сыном.

6. Запускается метод класса лес, отвечающий за печать элементов. Печатается уступчатое представление леса.

7. Запускается метод класса лес, отвечающий за печать горизонтального обхода дерева. Рекурсивно проходя по уровням и записывая значения элементов в соответствующие строки получаем массив строк, который при последовательном выводе дает правильную последовательность элементов леса при горизонтальном обходе.

8. Программа завершается без ошибок.

**Вывод**

В ходе лабораторной работы были изучены принципы работы с бинарным деревом и лесом, его заполнение, печать элементов, конвертация одного в другое.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД, СОДЕРЖАЩИЙ ГОЛОВНУЮ ФУНКЦИЮ**

#include <cmath>

#include <string>

#include <iostream>

#include <fstream>

#include "structures.h"

int main(){

int level=0,bt\_depth=0;

int length=0,elMax=0;

std::ifstream file;

char name[20];

//string which contains bt

std::string str\_bt;

//choice of terminal or file mode

std::cout << "Type 1 to proceed in terminal, and 2 to use file.\n(else you will be forced to use file)" << std::endl;

std::getline(std::cin,str\_bt);

if (str\_bt[0]=='1'){

std::cout << "Enter a binary tree, which you want to convert in forest:" << std::endl;

std::getline(std::cin,str\_bt);

}

else{

std::cout << "Please, type the name of the file in format <name>.txt" << std::endl;

std::cin.getline(name,20);

file.open(name);

if (!file.is\_open()){

std::cout << "\nFile couldn't be open\nProgram terminated!" << std::endl;

return 0;

}

std::getline(file,str\_bt);

file.close();

}

std::cout << "Your output is " << str\_bt << std::endl;

//length of the string

length=str\_bt.length();

//level - уровень, где находимся, bt\_depth считает глубину дерева

for (int i=0;i<length;i++){

if (str\_bt[i]=='(')

level++;

if (str\_bt[i]==')')

level--;

if (bt\_depth<level) bt\_depth=level;

}

if (level){

std::cout <<"\nError! Brackets aren't closed!\nProgram terminated!" << std::endl;

return 0;

}

if (!bt\_depth){

std::cout << "\nError! It is not a tree!\nProgram terminated!" << std::endl;

return 0;

}

//counting maximum numbers of elements in tree (2^bt\_depth - 1)

elMax=(int)pow(2.0,(double)bt\_depth)-1;

//creation of bin tree class

binTree<char> user\_bt(bt\_depth);

int curr\_pos=0;

if (str\_bt[curr\_pos]=='('){

curr\_pos++;

fill\_from\_str(&user\_bt,str\_bt,&curr\_pos,-1);

}

else{

std::cout << "\nError! No open bracket!\nProgram terminated!" << std::endl;

return 0;

}

std::cout << "\n"<< "Binary tree:" << std::endl;

//printing binary tree

user\_bt.print\_tree(0,bt\_depth);

// creation of forest

Forest<char> user\_forest(elMax+1,bt\_depth,bt\_depth);

//for position in forest

int posf=1;

//filling forest from binary tree with convertation

convert\_bt\_to\_forest(&user\_bt,&user\_forest,0,0,&posf,0);

std::cout << "\n"<< "Forest:" << std::endl;

//printing forest

user\_forest.print\_forest(0,bt\_depth);

//for output forest in horizontal order

std::string \* levels= new std::string[bt\_depth];

//filling strings with levels

user\_forest.print\_horizontal(levels,0,0);

int index=0;

std::cout << "\n"<< "Forest in horizontal order: ";

//output in horizontal order

while(index<bt\_depth){

std::cout << levels[index];

index++;

}

std::cout << std::endl;

delete [] levels;

return 0;

}

**ПРИЛОЖЕНИЕ Б**

**ИСХОДНЫЙ КОД ЗАГОЛОВОЧНИКА**

#include <cmath>

#include <string>

#include <iostream>

template <class Type>

struct binTree{

binTree(int n = 0): depth(n)

{

size=(int)pow(2.0,(double)depth)-1;

storage = new bt\_element[size];

for (int i=0;i<size;i++){

storage[i].value=0;

storage[i].left=0;

storage[i].right=0;

}

}

~binTree()

{

delete [] storage;

}

void print\_tree(int index,int depth){

std::string tabs(depth,'-');

if (storage[index].value){

std::cout.width(this->depth+3);

std::cout << storage[index].value + tabs << std::endl;

}

if (storage[index].left>0){

binTree::print\_tree(storage[index].left,depth-1);

}

if (storage[index].right>0){

binTree::print\_tree(storage[index].right,depth-1);

}

}

struct bt\_element{

Type value;

int left;

int right;

};

int size;

bt\_element \* storage;

int depth;

};

template <class Type>

struct Forest{

Forest(int n = 0,int num\_sons = 3,int fdepth=0): size(n),max\_sons(num\_sons+1),depth(fdepth)

{

storage = new forest\_element[size];

for (int i=0;i<size;i++){

storage[i].value=0;

storage[i].sons = new int[max\_sons];

for (int j=0;j<max\_sons;j++)

storage[i].sons[j]=-1;

}

}

~Forest()

{

for (int i=0;i<size;i++)

delete [] storage[i].sons;

delete [] storage;

}

void print\_forest(int index,int depth){

std::string tabs(depth+1,'-');

int son\_ind=0;

if (storage[index].value){

std::cout.width(this->depth+3);

std::cout << storage[index].value + tabs << std::endl;

}

while (storage[index].sons[son\_ind]>-1){

Forest::print\_forest(storage[index].sons[son\_ind],depth-1);

son\_ind++;

}

}

void print\_horizontal(std::string \* out,int index, int level){

int son\_ind=0;

while (storage[index].sons[son\_ind]>-1){

out[level]+=storage[storage[index].sons[son\_ind]].value;

son\_ind++;

}

son\_ind=0;

while (storage[index].sons[son\_ind]>-1){

Forest::print\_horizontal(out,storage[index].sons[son\_ind],level+1);

son\_ind++;

}

}

struct forest\_element{

Type value;

int \* sons;

};

int size;

int max\_sons;

forest\_element \* storage;

int depth;

};

int fill\_from\_str(const binTree<char> \* bt,const std::string str\_bt, int \* position, int root);

void convert\_bt\_to\_forest(const binTree<char> \* bt, Forest<char> \* bt\_forest,int father,int num\_son,int \* posf,int pos);

**ПРИЛОЖЕНИЕ В**

**ИСХОДНЫЙ КОД СОДЕРЖАЩИЙ ФУНКЦИИ**

#include "structures.h"

//filling binary tree from string with bracket binary tree

int fill\_from\_str(const binTree<char> \* bt,const std::string str\_bt, int \* position, int root){

int curr\_pos=0;

int left\_son=2\*root+1;

int right\_son=2\*root+2;

if (root>-1){

if (bt->storage[root].left==0)

curr\_pos=left\_son;

else

curr\_pos=right\_son;

}

if (curr\_pos>=bt->size) return 0;

while(1){

if (str\_bt[\*position]==')'){

(\*position)++;

if (bt->storage[curr\_pos].left==0)

bt->storage[curr\_pos].left=-1;

if (bt->storage[curr\_pos].right==0)

bt->storage[curr\_pos].right=-1;

break;

}

if (str\_bt[\*position]=='#'){

(\*position)++;

bt->storage[curr\_pos].left=-1;

continue;

}

if (str\_bt[\*position]=='('){

(\*position)++;

fill\_from\_str(bt,str\_bt,position,curr\_pos);

continue;

}

if (isalpha(str\_bt[\*position])){

bt->storage[curr\_pos].value=str\_bt[\*position];

(\*position)++;

if (root==-1) continue;

if (curr\_pos==left\_son)

bt->storage[root].left=left\_son;

if (curr\_pos==right\_son)

bt->storage[root].right=right\_son;

continue;

}

(\*position)++;

}

return 0;

}

// convertation binaty tree into forest

void convert\_bt\_to\_forest(const binTree<char> \* bt, Forest<char> \* bt\_forest,int father,int num\_son,int \* posf,int pos){

bt\_forest->storage[\*posf].value=bt->storage[pos].value;

bt\_forest->storage[father].sons[num\_son]=\*posf;

if (bt->storage[pos].left>-1){

(\*posf)++;

convert\_bt\_to\_forest(bt,bt\_forest,(\*posf)-1,0,posf,bt->storage[pos].left);

}

if (bt->storage[pos].right>-1){

(\*posf)++;

convert\_bt\_to\_forest(bt,bt\_forest,father,num\_son+1,posf,bt->storage[pos].right);

}

return;

}

**ПРИЛОЖЕНИЕ Г**

**ИСХОДНЫЙ КОД СКРИПТА**

#!/bin/bash

make

echo " "

echo "Tests with correct data:"

echo " "

echo "Test 1"

echo " "

./a.out <./Tests/test1.txt

echo " "

echo "Test 2"

echo " "

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

echo " "

echo "Test 3"

echo " "

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

echo " "

echo "Test 4"

echo " "

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

echo " "

echo "Test 5"

echo " "

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

echo " "

echo "Tests with incorrect data:"

echo " "

echo " "

echo "Test 6"

echo " "

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

echo " "

echo "Test 7"

echo " "

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

echo " "

echo "Test 8"

echo " "

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

echo " "

echo "Test 9"

echo " "

./a.out <./Tests/test9.txt

make clean

**ПРИЛОЖЕНИЕ Д**

**РЕЗУЛЬТАТ РАБОТЫ СКРИПТА**

Test 1

Your output is (a (b) (c))

Binary tree:

a--

b-

c-

Forest:

a--

b-

c--

Forest in horizontal order: acb

Test 9

Your output is a(a)(a))

Error! Brackets aren't closed!

Program terminated!