Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра інформаційних систем та мереж

### ЗВІТ

про виконання лабораторної роботи №5

на тему: «Робота з динамічними структурами»

з предмету «Алгоритми та структури даних»

Виконала:

ст. гр. СА-32

Малинка А.А.

Прийняв:

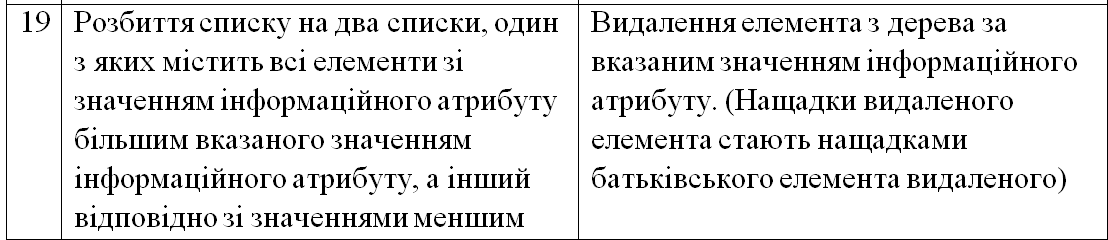
доцент

Щербак С.С.

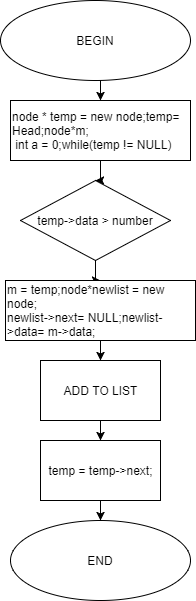
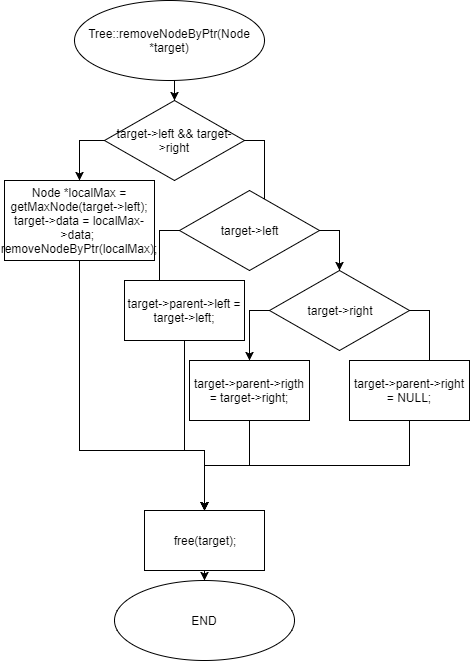
Львів-2017

**Мета роботи:** набуття навичок моделювання зв’язаних динамічних структур даних та роботи з ними

Лабораторне завдання: (варіант 19(3))



Розробити програми, які виконують операції вказані в індивідуальному завданні.

* Програму для роботи з двонапрааленими звязними списками. Кожен елемент списку містить зсилки на наступний і попередній елемент в списку. Програма повинна забезпечувати ввід і побудову списку.
* Програму для роботи для роботи з деревами. Кожен елемент дерева містить зсилку на батьківський елемент і зсилки на елементи-нащадки (необмежена кількість). Програма повинна забезпечувати ввід і побудову дерева.
* Кожен елемент списку містить інформаційне поле(атрибут) деякого простого типу: символ, стрічка, число.
* Всі операції над динамічними структурами повинні супроводжуватись відповідним виводом на екран.
* В контрольних прикладах забезпечити опрацювання структур з 1000 елементами. Вихідні дані завантажуйте через XML-файл.

Блок-схеми реалізації функцій, 1) розділяє список на дві частини, 2) видаляє елемент з дерева

Рис 1. Блок-схема функції, що розділяє список на дві частини

Реалізований алгоритм надзвичайно простий, він проходить по, вже створеному, списку якщо елемент більший за той з яким порівнюємо (заданий користувачем) то він потрапляє у першу чергу, якщо ж ні, або дорівнює йому - у другу.

Рис 2. Блок-схема функції, що видаляє елемент з дерева

Даний алгоритм працює на основі 2х функцій. Перша шукає елемент у дереві, а друга власне його видаляє. Перша – визначає який саме елемент потрібно видалити. Друга функція працює на основі алгоритму пошуку у дереві. Якщо елемент правий і лівий звіряємо його з максимальним та видаляємо, якщо лівий змінюємо вказівник на його на його лівого нащадка, якщо його не існує то на правого, інверсійно діємо і для того якщо елемент лівий, якщо елемент листок ( незалежно правий чи лівий) змінюємо його вказівник на 0.

Код програми ( мовою с++):

**///part1.h**

#include "Dividebyvalue.h"

void main()

{

STARTUPINFO sif;

PROCESS\_INFORMATION pif;

TCHAR CommandLine[] = TEXT("lab5.02.exe");

ZeroMemory(&sif, sizeof(sif));

sif.cb = sizeof(sif);

ZeroMemory(&pif, sizeof(pif));

cout << "Work of the first process\n";

List L;

CSV C;

int n = 0;

vector <int> myValues;

int number, numbers = 1000, lines = 1;

try {

C.OpenFile(myValues, numbers, lines);

C.ShowArray(myValues, numbers, lines);

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

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

L.AddTail(myValues[i]);

}

}

//cout << "\nList L:\n";

//L.Print();

cout << "\nInput number to separate" << endl;

if (!(cin >> number) || number < 1 || number>1000)

{

throw 2;

}

L.CopyLeft(number);

L.CopyRight(number);

cout << "\nFist part\n";

L.PrintNewLeft();

cout << "\nSecond part\n";

L.PrintNewRight();

}

catch (int error) {

if (error == 1)

cout << "Couldn't open a file\n";

if (error == 2)

cout << " Value have to be JUST digit from 1 to 1000\n";

}

if (!CreateProcess(NULL, CommandLine, NULL, NULL, FALSE, 0, NULL, NULL, &sif, &pif))

cout << "CreateProcess failed.\n";

WaitForSingleObject(pif.hProcess, INFINITE);

cout << "End of child process.\n";

CloseHandle(pif.hProcess);

CloseHandle(pif.hThread);

system("pause");

}

**/// Dividebyvalue.cpp**

#include "Dividebyvalue.h"

List::List()

{

Head = Tail = NULL;

Count = 0;

}

List::List(const List & L)

{

Head = Tail = NULL;

Count = 0;

node \* temp = L.Head;

while (temp != 0)

{

AddTail(temp->data);

temp = temp->next;

}

}

void List::AddHead(int n)

{

node \* temp = new node;

temp->prev = 0;

temp->data = n;

temp->next = Head;

if (Head != 0)

Head->prev = temp;

if (Count == 0)

Head = Tail = temp;

else

Head = temp;

Count++;

}

void List::AddTail(int n)

{

node \* temp = new node;

temp->next = 0;

temp->data = n;

temp->prev = Tail;

if (Tail != 0)

Tail->next = temp;

if (Count == 0)

Head = Tail = temp;

else

Tail = temp;

Count++;

}

void List::CopyLeft(int number)

{

node \* temp = new node;

temp = Head;

node \*m; int a = 0;

while (temp != NULL)

{

a++;

if (temp->data <= number)

{

m = temp;

node \*newlist = new node;

newlist->next = NULL;

newlist->data = m->data;

if (HeadL != NULL)

{

newlist->prev = TailL;

TailL->next = newlist;

TailL = newlist;

}

else

{

newlist->prev = NULL;

HeadL = TailL = newlist;

}

}

temp = temp->next;

}

}

void List::CopyRight(int number)

{

node \* temp = new node;

temp = Head;

node \*m; int a = 0;

while (temp != NULL)

{

a++;

if (temp->data > number)

{

m = temp;

node \*newlist = new node;

newlist->next = NULL;

newlist->data = m->data;

if (HeadR != NULL)

{

newlist->prev = TailR;

TailR->next = newlist;

TailR = newlist;

}

else

{

newlist->prev = NULL;

HeadR = TailR = newlist;

}

}

temp = temp->next;

}

}

void List::Print()

{

if (Count != 0)

{

node \* temp = Head;

cout << "( ";

while (temp->next != 0)

{

cout << temp->data << ", ";

temp = temp->next;

}

cout << temp->data << " )\n";

}

}

void List::PrintNewLeft()

{

if (Count != 0)

{

node \* temp = HeadL;

cout << "( ";

while (temp->next != 0)

{

cout << temp->data << ", ";

temp = temp->next;

}

cout << temp->data << " )\n";

}

}

void List::PrintNewRight()

{

if (Count != 0)

{

node \* temp = HeadR;

cout << "( ";

while (temp->next != 0)

{

cout << temp->data << ", ";

temp = temp->next;

}

cout << temp->data << " )\n";

}

}

int List::GetCount()

{

return Count;

}

void CSV::OpenFile(vector<int>&myValues, int numbers, int lines)

{

int x = 0; // a variable that will contain a value from the file

ifstream ifs("numbers.csv");

if (ifs.fail())

{

throw 1;

ifs.close();

}

char tmp;

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

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

ifs >> x;

myValues.push\_back(x);

if (i < (numbers - 1))

ifs >> tmp;

}

}

}

void CSV::ShowArray(vector<int>&myValues, int numbers, int lines)

{

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

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

cout << myValues[i] << ' ';

}

}

}

**///Dividebyvalue.h**

#pragma once

//#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <fstream>

#include <vector>

#include <windows.h>

using namespace std;

struct node

{

int data;

node \* next, \*prev;

};

class List

{

node \* Head, \*Tail, \*HeadL, \*TailL, \*HeadR, \*TailR;

int Count;

public:

List();

List(const List&);

int GetCount();

void CopyLeft(int number);

void CopyRight(int number);

void AddTail(int n);

void AddHead(int n);

void Print();

void PrintNewLeft();

void PrintNewRight();

};

class CSV

{

public:

void OpenFile(vector<int>&myValues, int numbers, int lines);

void ShowArray(vector<int>&myValues, int numbers, int lines);

};

**/// part2.cpp**

#include "Deletenode.h"

void main() {

cout << "\n\n\nStart of second program";

Tree T;

CSV C;

Node \*root = NULL;

vector <int> myValues;

int number, numbers = 1000, lines = 1, value;;

try {

C.OpenFile(myValues, numbers, lines);

C.ShowArray(myValues, numbers, lines);

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

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

T.insert(&root, myValues[i]);

}

}

T.printTree(root, "root", 0);

cout << "------------------\n Input value to delete ";

cin >> value;

if (value < 1 || value > 1000)

throw 3;

T.deleteValue(root, value);

T.printTree(root, "root", 0);

cout << "------------------\n";

}

catch (int error)

{

if (error == 1)

cout << "Couldn't open file ";

if (error == 2)

cout << "File is empty, can't read";

if (error == 3)

cout << "No such element in tree";

}

system("pause");

}

**/// Deletenode.cpp**

#include "Deletenode.h"

Node\* getFreeNode(int value, Node \*parent) {

Node\* tmp = (Node\*)malloc(sizeof(Node));

tmp->left = tmp->right = NULL;

tmp->data = value;

tmp->parent = parent;

return tmp;

}

Node\* getMinNode(Node \*root) {

while (root->left) {

root = root->left;

}

return root;

}

Node\* getMaxNode(Node \*root) {

while (root->right) {

root = root->right;

}

return root;

}

Node \*getNodeByValue(Node \*root, int value) {

while (root) {

if (CMP\_GREAT(root->data, value)) {

root = root->left;

continue;

}

else if (CMP\_LESST(root->data, value)) {

root = root->right;

continue;

}

else {

return root;

}

}

return NULL;

}

void Tree::insert(Node \*\*head, int value) {

Node \*tmp = NULL;

Node \*ins = NULL;

if (\*head == NULL) {

\*head = getFreeNode(value, NULL);

return;

}

tmp = \*head;

while (tmp) {

if (CMP\_GREAT(value, tmp->data) || CMP\_EQUIL(value, tmp->data)) {

if (tmp->right) {

tmp = tmp->right;

continue;

}

else {

tmp->right = getFreeNode(value, tmp);

return;

}

}

else if (CMP\_LESST(value, tmp->data)) {

if (tmp->left) {

tmp = tmp->left;

continue;

}

else {

tmp->left = getFreeNode(value, tmp);

return;

}

}

//else {

//exit(2);

//}

}

}

void Tree::removeNodeByPtr(Node \*target) {

if (target->left && target->right) {

Node \*localMax = getMaxNode(target->left);

target->data = localMax->data;

removeNodeByPtr(localMax);

return;

}

else if (target->left) {

if (target == target->parent->left) {

target->parent->left = target->left;

}

else {

target->parent->right = target->left;

}

}

else if (target->right) {

if (target == target->parent->right) {

target->parent->right = target->right;

}

else {

target->parent->left = target->right;

}

}

else {

if (target == target->parent->left) {

target->parent->left = NULL;

}

else {

target->parent->right = NULL;

}

}

free(target);

}

void Tree::deleteValue(Node \*root, int value) {

Node \*target = getNodeByValue(root, value);

removeNodeByPtr(target);

}

void Tree::printTree(Node \*root, const char \*dir, int level) {

if (root) {

cout << "lvl " << level << " " << dir << " = " << root->data << endl;

printTree(root->left, "left", level + 1);

printTree(root->right, "right", level + 1);

}

}

void CSV::OpenFile(vector<int>&myValues, int numbers, int lines)

{

int x = 0;

ifstream ifs("test.csv");

if (ifs.fail())

{

throw 1;

}

if (ifs.eof()) {

throw 2;

}

char tmp;

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

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

ifs >> x;

myValues.push\_back(x);

if (i < (numbers - 1))

ifs >> tmp;

}

}

}

void CSV::ShowArray(vector<int>&myValues, int numbers, int lines)

{

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

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

cout << myValues[i] << ' ';

}

}

cout << endl;

}

**/// Deletenode.h**

#pragma once

#include <iostream>

#include <fstream>

#include <vector>

using namespace std;

#define CMP\_EQUIL(a, b) ((a) == (b))

#define CMP\_LESST(a, b) ((a) < (b))

#define CMP\_GREAT(a, b) ((a) > (b))

typedef struct Node {

int data;

struct Node \*left;

struct Node \*right;

struct Node \*parent;

} Node;

class Tree {

public:

void insert(Node \*\*head, int value);

void removeNodeByPtr(Node \*target);

void deleteValue(Node \*root, int value);

void printTree(Node \*root, const char \*dir, int level);

};

class CSV {

public:

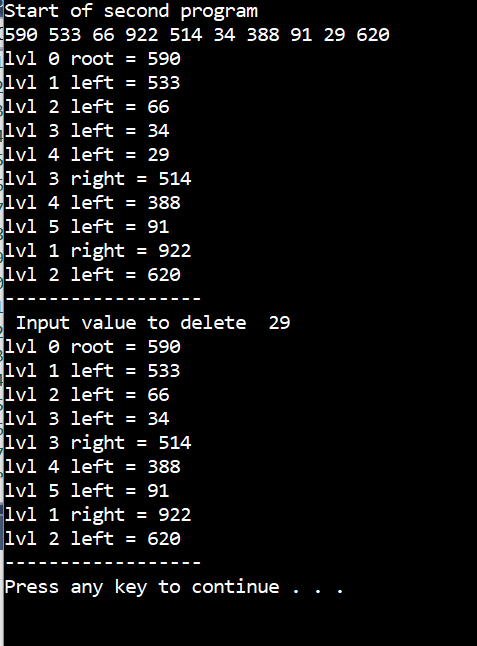
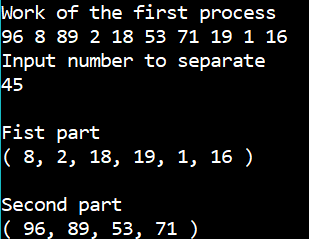
void OpenFile(vector<int>&myValues, int numbers, int lines);

void ShowArray(vector<int>&myValues, int numbers, int lines);

};

Приклад роботи програм:

Для прикладу взято лише 10 значень так як важко відтворити роботу з 1000 елементами у звіті



**Висновок**

В результаті виконання цієї лабораторної роботи, я навчилася реалізовувати операції з списком та деревами, модифікувати її, видаляти та додавати елементи