**ЛАБОРАТОРНА РОБОТА №6-7**

## Графи, дерева та їх застосування

**Мета:**

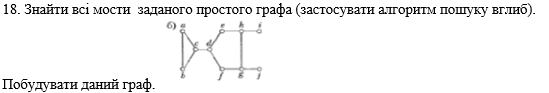
* навчитись матрично задавати графи, відображати їх графічно; використовуючи властивості матриць графів, розв'язувати відповідні задачі.

**Хід роботи:**

1. Створити програму що відповідає своєму варіанту.
2. Для створення звіту вибрати відповідну своєму варіанту задачу і розв'язати її за допомогою програми.
3. Результат показати викладачу.

**Задача 1**

**Варіант 18, завдання 18.**

1. **Примітка: для завдання 1 відобразити графічно на формі введений граф за допомогою матриці. Продумати можливість збереження та завантаження графів в формі матриць в БД.**

**2)Лістинг програми;**

#include <iostream>

#include <iomanip>

#include <Windows.h>

using namespace std;

string vertix[10] = { "a", "b", "c", "d", "e", "f", "g", "h", "i", "j" };

const int MAXN = 10;

int n;

int a[MAXN][MAXN];

bool visited[MAXN];

void dfs(int v);

int main()

{

setlocale(LC\_ALL, "ukr");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int start\_vertex; // стартова вершина

cout << "Введіть кількість вершин: ";

cin >> n;

cout << "Введіть матрицю суміжності:" << endl;

cout << "a b c d e f g h i j" << endl;

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

{

for (int j = 0; j < n; ++j)

{

cin >> a[i][j];

}

}

cout << "Усі ребра, що є в графі:" << endl;

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

for (int j = i + 1; j < n; ++j) {

if (a[i][j] == 1)

{

cout << i << " - " << j << endl;

}

}

}

memset(visited, false, sizeof(visited));

cout << "Введіть вершину з якої почнется пошук: ";

cin >> start\_vertex;

start\_vertex--;

cout << "DFS почнеться з:" << vertix[start\_vertex] << endl;

dfs(start\_vertex);

return 0;

}

void dfs(int v)

{

visited[v] = true;

cout << vertix[v] << " ";

for (int u = 0; u < n; ++u)

{

if (a[v][u] && !visited[u])

{

dfs(u);

}

}

/\* тест нам

0 1 1 0 0 0 0 0 0 0

1 0 1 0 0 0 0 0 0 0

1 1 0 1 0 0 0 0 0 0

0 0 1 0 1 1 0 0 0 0

0 0 0 1 0 0 0 1 0 0

0 0 0 1 0 0 1 0 0 0

0 0 0 0 0 1 0 1 0 1

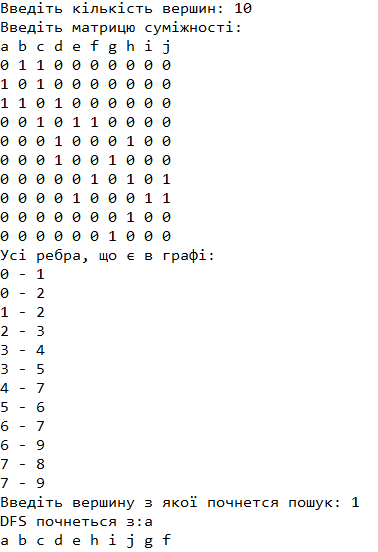
0 0 0 0 1 0 0 0 1 1

0 0 0 0 0 0 0 1 0 0

0 0 0 0 0 0 1 0 0 0

\*/

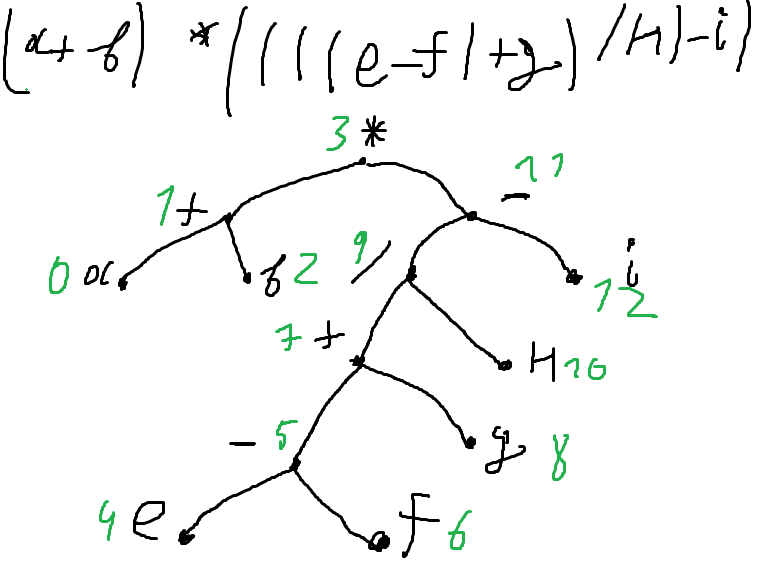
**5)Результати**



**Задача 2**

**Варіант 18, завдання 18.**

**1)Заданий інфіксний запис (a+ b)\* (((( e - f)+g) / h) – i). Побудуйте бінарне дерево, яке задає цей вираз. Запишіть префіксний і постфіксний запис цього виразу.**

****

2)Лістинг програми

#include <iostream>

#include <iomanip>

#include <Windows.h>

using namespace std;

struct Node

{

int data;

Node\* left;

Node\* right;

};

string Deshif\_num\_to\_sym [] = { "a", "+", "b", "\*", "e", "-", "f", "+", "g", "-", "h", "-", "i" };

int temp;

Node\* CreateNode(int data);

Node\* BuildTreeUtil(int array\_of\_value[], int\* index\_ptr, int low, int high, int n);

Node\* BuildTree(int array\_of\_value[], int n);

void PrintInorder(Node\* root);

void LEVEL(Node\* root, int& i, int& max\_level);

void Display\_level(Node\* root, int& i, int level);

void Display\_all(Node\* root);

void PrintBinaryTree(Node\* root, int space = 0, int height = 10);

void PostorderTraversal(Node\* root);

int main()

{

setlocale(LC\_ALL, "ukr");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

string expr = "(a+ b)\* (((( e - f)+g) / h) – i)";

int array\_of\_value[] = { 3, 1, 0, 2, 11, 9, 7, 5, 4, 6, 8, 10, 12 };// номерування дерева числами, для подальшого шифрування

int n = sizeof(array\_of\_value) / sizeof(array\_of\_value[0]);

Node\* root = BuildTree(array\_of\_value, n);

cout << "Початковий вираз" << endl;

cout << expr << endl;

cout << "Префіксний обхід графа: " << endl;

PrintInorder(root);

cout << endl << "Постфіксний обхід графа: " << endl;

PostorderTraversal(root);

cout << endl;

cout << "Вивід дерева графічно по рівнях: " << endl;

PrintBinaryTree(root, 0, 5);

cout << "Вивід дерева по рівнях: " << endl;

Display\_all(root);

return 0;

}

void PostorderTraversal(Node\* root)

{

if (root == nullptr)

{

return;

}

PostorderTraversal(root->left);

PostorderTraversal(root->right);

temp = root->data;

cout << Deshif\_num\_to\_sym[temp] << " ";

}

void PrintBinaryTree(Node\* root, int space, int height)

{

int temp;

if (root == NULL)

{

return;

}

space += height;

PrintBinaryTree(root->right, space, height);

cout << endl;

for (int i = height; i < space; i++)

{

cout << " ";

}

temp = root->data;

cout << Deshif\_num\_to\_sym[temp] << endl;

PrintBinaryTree(root->left, space, height);

}

Node\* CreateNode(int data)

{

Node\* newNode = new Node();

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

Node\* BuildTreeUtil(int array\_of\_value[], int\* index\_ptr, int low, int high, int n)

{

if (\*index\_ptr >= n || low > high)

{

return NULL;

}

Node\* root = CreateNode(array\_of\_value[\*index\_ptr]);

\*index\_ptr = \*index\_ptr + 1;

if (low == high)

{

return root;

}

int i;

for (i = low; i <= high; ++i)

{

if (array\_of\_value[i] > root->data)

{

break;

}

}

root->left = BuildTreeUtil(array\_of\_value, index\_ptr, \*index\_ptr, i - 1, n);

root->right = BuildTreeUtil(array\_of\_value, index\_ptr, i, high, n);

return root;

}

Node\* BuildTree(int array\_of\_value[], int n)

{

int index = 0;

return BuildTreeUtil(array\_of\_value, &index, 0, n - 1, n);

}

void PrintInorder(Node\* root)

{

if (root == nullptr)

{

return;

}

temp = root->data;

cout << Deshif\_num\_to\_sym[temp] << " ";

PrintInorder(root->left);

PrintInorder(root->right);

}

void LEVEL(Node\* root, int& i, int& max\_level)

{

if (root->left != NULL)

{

i++;

if (max\_level < i)

{

max\_level = i;

}

LEVEL(root->left, i, max\_level);

i--;

}

if (root->right != NULL)

{

i++;

if (max\_level < i)

{

max\_level = i;

}

LEVEL(root->right, i, max\_level);

i--;

}

}

void Display\_level(Node\* root, int& i, int level)

{

if (i == level)

{

cout << " ";

temp = root->data;

cout << Deshif\_num\_to\_sym[temp] << endl;

}

if (root->left != NULL)

{

i++;

Display\_level(root->left, i, level);

i--;

}

if (root->right != NULL)

{

i++;

Display\_level(root->right, i, level);

i--;

}

}

void Display\_all(Node\* root)

{

int i = 0;

int max\_level = 0;

LEVEL(root, i, max\_level);

cout << "max рівень = " << max\_level << endl;

cout << " ";

for (int level = 0; level <= max\_level; level++)

{

if (level == 0)

{

cout << "0 рівень: " << endl;

}

else

{

cout << "(" << level << " рівень): " << endl;

}

Display\_level(root, i, level);

}

cout << endl;

}

**5)Результати**



**Висновок:** навчився шукати найкоротший шлях у графах на основі методу Дейкстри та Флойда.