**ЛАБОРАТОРНА РОБОТА № 6-7**

**Графи. Дерева. Алгоритми пошуку в глибину та ширину**

***Мета:*** засвоїти та закріпити прийоми роботи з даними різного типу, організованими у вигляді дерев та їх часткового випадку – бінарних дерев. Отримати практичні навички роботи з графами

**Зміст роботи**

Завдання 1. Записав матрицю зміжності згідно зі завданням:

Київ – **К**, Житомир – **Ж**, Новоград-Волинський – **НВ**, Рівно – **Р**, Луцьк - **Л**, Бердичів – **Б**, Вінниця – **В**, Хмельницький – **Хм**, Тернопіль – **Т**, Шепетовка – **Ш**, Біла Церква – **БЦ**, Умань – **У**, Черкаси – **Ч**, Кременчуг – **Кр**, Полтава – **П**, Харків – **Х**, Прилуки – **Пр**, Суми – **С**, Миргород – **М**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **К** | **Ж** | **НВ** | **Р** | **Л** | **Б** | **В** | **Хм** | **Т** | **Ш** | **БЦ** | **У** | **Ч** | **Кр** | **П** | **Х** | **Пр** | **С** | **М** |
| **К** | ***-*** | ***135*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***78*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***128*** | ***0*** | ***0*** |
| **Ж** | ***135*** | ***-*** | ***80*** | ***0*** | ***0*** | ***38*** | ***0*** | ***0*** | ***0*** | ***115*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **НВ** | ***0*** | ***80*** | ***-*** | ***100*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **Р** | ***0*** | ***0*** | ***100*** | ***-*** | ***68*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **Л** | ***0*** | ***0*** | ***0*** | ***68*** | ***-*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **Б** | ***0*** | ***38*** | ***0*** | ***0*** | ***0*** | ***-*** | ***73*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **В** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***73*** | ***-*** | ***110*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **Хм** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***110*** | ***-*** | ***104*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **Т** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***104*** | ***-*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **Ш** | ***0*** | ***115*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***-*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **БЦ** | ***78*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***-*** | ***115*** | ***146*** | ***0*** | ***181*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **У** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***115*** | ***-*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **Ч** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***146*** | ***0*** | ***-*** | ***105*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **Кр** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***105*** | ***-*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| **П** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***181*** | ***0*** | ***0*** | ***0*** | ***-*** | ***130*** | ***0*** | ***0*** | ***0*** |
| **Х** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***130*** | ***-*** | ***0*** | ***0*** | ***0*** |
| **Пр** | ***128*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***-*** | ***175*** | ***109*** |
| **С** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***175*** | ***-*** | ***0*** |
| **М** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***109*** | ***0*** | ***-*** |

Запис від’ємних чисел у зворотному напрямку руху є неможливим для відображення через великий розмір таблиці.

Завдання 2. Створив програму демонстрації роботи DFS (DLS) та BFS:

Алгоритм DFS аналізує можливі переходи з поточного елементу на інші, доступні з нього, та, якщо програма у сусідніх елементах ще не була, переходить у них, записучи поточний елемент у стек, після чого знову аналізує новий елемент. За відсутності доступних шляхів, програма повертається у попередній елемент, витягуючи зі стеку інформацію про нього, та знову проводиться аналіз шляхів або повторне повернення до попереднього елементу поточного. В кінці роботи програми зі стеку дістається кореневий елемент та алгоритм закінчує роботу.

Алгоритм BFS аналізує усі доступні шляхи та записує їх у чергу, після чого дістає найперший з них та аналізує його, додаючи нові шляхи від нього у кінець черги. Якщо елемент вже є у черзі, він не буде записаний. Алгоритм повторює звертання та перехід по черзі, поки вона не скінчиться. Тоді граф буде повністю проаналізовано, а алгоритм скінчить роботу.

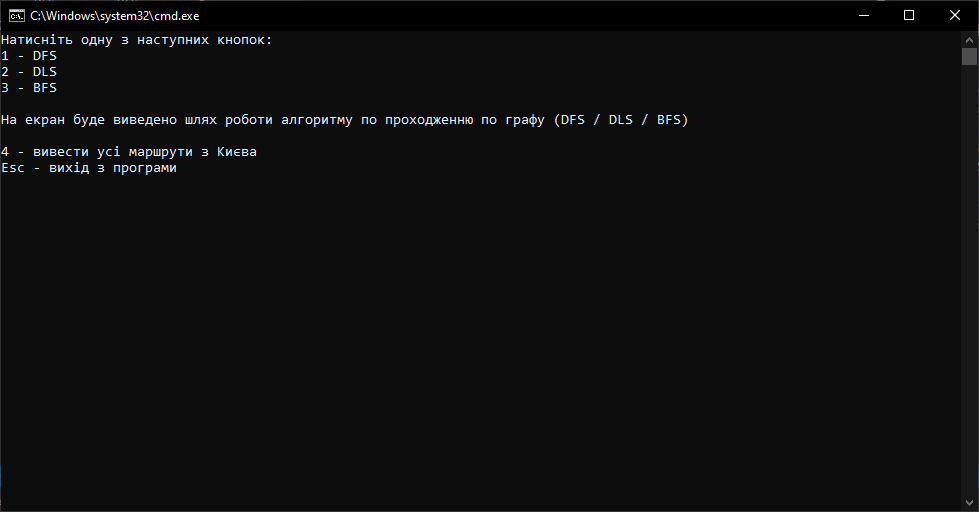


Рис.6-7.1 – стартове меню програми

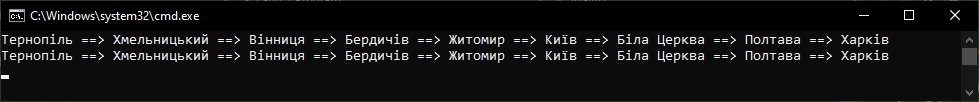


Рис.6-7.2 – результат обходження за допомоги DFS



Рис.6-7.3 – результат обходження за допомоги DLS



Рис.6-7.4 – результат обходження за допомоги BFS

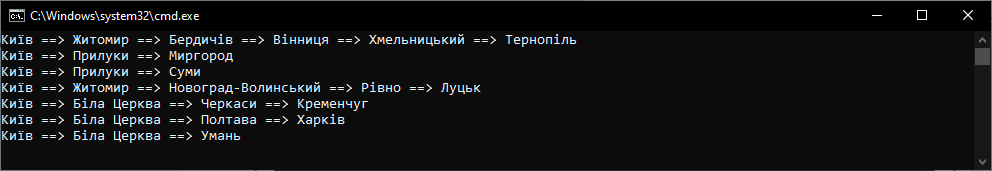


Рис.6-7.5 – усі шляхи з Києва

Лістинг програми, консольний додаток:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using static System.Console;

namespace Lab6\_7

{

class Program

{

static void Main()

{

InputEncoding = Encoding.Unicode;

OutputEncoding = Encoding.Unicode;

Node[] nodes = { new Node("Київ"), new Node("Житомир"), new Node("Новоград-Волинський"),

new Node("Рівно"), new Node("Луцьк"), new Node("Бердичів"), new Node("Вінниця"),

new Node("Хмельницький"), new Node("Тернопіль"), new Node("Шепетовка"), new Node("Біла Церква"),

new Node("Умань"), new Node("Черкаси"), new Node("Кременчуг"), new Node("Полтава"),

new Node("Харків"), new Node("Прилуки"), new Node("Суми"), new Node("Миргород") };

nodes[0].AddChildren(nodes[1]).AddChildren(nodes[10]).AddChildren(nodes[16]);

nodes[1].AddChildren(nodes[9]).AddChildren(nodes[5]).AddChildren(nodes[2]);

nodes[2].AddChildren(nodes[3]);

nodes[3].AddChildren(nodes[4]);

nodes[5].AddChildren(nodes[6]);

nodes[6].AddChildren(nodes[7]);

nodes[7].AddChildren(nodes[8]);

nodes[10].AddChildren(nodes[11]).AddChildren(nodes[12]).AddChildren(nodes[14]);

nodes[12].AddChildren(nodes[13]);

nodes[14].AddChildren(nodes[15]);

nodes[16].AddChildren(nodes[17]).AddChildren(nodes[18]);

ConsoleKeyInfo keySwitcher;

bool repeat;

do

{

WriteLine("Натисніть одну з наступних кнопок:\n" +

"1 - DFS\n" +

"2 - DLS\n" +

"3 - BFS\n\n" +

"На екран буде виведено шлях роботи алгоритму по проходженню по графу (DFS / DLS / BFS)\n\n" +

"4 - вивести усі маршрути з Києва\n" +

"Esc - вихід з програми");

keySwitcher = ReadKey();

if (keySwitcher.Key != ConsoleKey.Escape)

{

repeat = true;

}

else

{

repeat = false;

}

Clear();

switch (keySwitcher.Key)

{

case ConsoleKey.D1:

var search\_DFS = new DFS();

var path\_DFS = search\_DFS.DFSList(nodes[8], nodes[15]);

PrintPath\_DFS(path\_DFS);

path\_DFS = search\_DFS.DFSList(nodes[8], nodes[15]);

PrintPath\_DFS(path\_DFS);

ReadKey();

break;

case ConsoleKey.D2:

var search\_DLS = new DLS();

var path\_DLS = search\_DLS.DLSList(nodes[9], nodes[17], 4);

PrintPath\_DLS(path\_DLS);

path\_DLS = search\_DLS.DLSList(nodes[9], nodes[17], 4);

PrintPath\_DLS(path\_DLS);

ReadKey();

break;

case ConsoleKey.D3:

var search\_BFS = new BFS();

var path\_BFS = search\_BFS.BFSList(nodes[0], nodes[15]);

PrintPath\_BFS(path\_BFS);

path\_BFS = search\_BFS.BFSList(nodes[0], nodes[15]);

PrintPath\_BFS(path\_BFS);

ReadKey();

break;

case ConsoleKey.D4:

var search = new DFS();

var path = search.DFSList(nodes[0], nodes[8]);

PrintPath\_DFS(path);

path = search.DFSList(nodes[0], nodes[18]);

PrintPath\_DFS(path);

path = search.DFSList(nodes[0], nodes[17]);

PrintPath\_DFS(path);

path = search.DFSList(nodes[0], nodes[4]);

PrintPath\_DFS(path);

path = search.DFSList(nodes[0], nodes[13]);

PrintPath\_DFS(path);

path = search.DFSList(nodes[0], nodes[15]);

PrintPath\_DFS(path);

path = search.DFSList(nodes[0], nodes[11]);

PrintPath\_DFS(path);

ReadKey();

break;

case ConsoleKey.Escape:

break;

}

Clear();

}

while (repeat);

}

private static void PrintPath\_DFS(LinkedList<Node> path\_DFS)

{

if (path\_DFS.Count == 0)

{

WriteLine("Граф не має членів");

}

else

{

WriteLine(string.Join(" ==> ", path\_DFS.Select(x => x.Name)));

}

}

private static void PrintPath\_DLS(LinkedList<Node> path\_DLS)

{

if (path\_DLS.Count == 0)

{

WriteLine("Граф не має членів");

}

else

{

WriteLine(string.Join(" ==> ", path\_DLS.Select(x => x.Name)));

}

}

private static void PrintPath\_BFS(LinkedList<Node> path\_BFS)

{

if (path\_BFS.Count == 0)

{

WriteLine("Граф не має членів");

}

else

{

WriteLine(string.Join(" ==> ", path\_BFS.Select(x => x.Name)));

}

}

}

class Node

{

public string Name { get; }

public List<Node> Children { get; }

public Node(string name)

{

Name = name;

Children = new List<Node>();

}

public Node AddChildren(Node node, bool bidirect = true)

{

Children.Add(node);

if (bidirect)

{

node.Children.Add(this);

}

return this;

}

}

}

Лістинг класу BFS.cs:

using System.Collections.Generic;

using System.Linq;

namespace Lab6\_7

{

class BFS

{

private HashSet<Node> WasVisited;

private LinkedList<Node> Path;

private Node Goal;

public LinkedList<Node> BFSList(Node start, Node goal)

{

WasVisited = new HashSet<Node>();

Path = new LinkedList<Node>();

this.Goal = goal;

BFSRecursion(start);

if (Path.Count > 0)

{

Path.AddLast(start);

}

return Path;

}

private bool BFSRecursion(Node node)

{

if (node == Goal)

{

return true;

}

WasVisited.Add(node);

foreach (var child in node.Children.Where(x => !WasVisited.Contains(x)))

{

if (BFSRecursion(child))

{

Path.AddFirst(child);

return true;

}

}

return false;

}

}

}

Лістинг класу DLS.cs:

using System.Collections.Generic;

using System.Linq;

namespace Lab6\_7

{

class DLS

{

private HashSet<Node> WasVisited;

private LinkedList<Node> Path;

private Node Goal;

private bool Limit;

public LinkedList<Node> DLSList(Node start, Node goal, int limit)

{

WasVisited = new HashSet<Node>();

Path = new LinkedList<Node>();

Limit = true;

this.Goal = goal;

DLSRecurion(start, limit);

if (Path.Count > 0)

{

Path.AddFirst(start);

}

return Path;

}

private bool DLSRecurion(Node node, int limit)

{

if (node == Goal)

{

return true;

}

if (limit == 0)

{

Limit = false;

return false;

}

WasVisited.Add(node);

foreach (var child in node.Children.Where(x => !WasVisited.Contains(x)))

{

if (DLSRecurion(child, limit - 1))

{

Path.AddFirst(child);

return true;

}

}

return false;

}

}

}

Лістинг класу DFS.cs:

using System.Collections.Generic;

using System.Linq;

namespace Lab6\_7

{

class DFS

{

private HashSet<Node> WasVisited;

private LinkedList<Node> Path;

private Node Goal;

public LinkedList<Node> DFSList(Node start, Node goal)

{

WasVisited = new HashSet<Node>();

Path = new LinkedList<Node>();

Goal = goal;

DFSRecursion(start);

if (Path.Count > 0)

{

Path.AddFirst(start);

}

return Path;

}

private bool DFSRecursion(Node node)

{

if (node == Goal)

{

return true;

}

WasVisited.Add(node);

foreach (var child in node.Children.Where(x => !WasVisited.Contains(x)))

{

if (DFSRecursion(child))

{

Path.AddFirst(child);

return true;

}

}

return false;

}

}

}

***Висновок:*** було засвоєно та закріплено прийоми роботи з даними різного типу, бінарними деревами та графами.