Лабораторна робота № 7-8

ГРАФИ. ДЕРЕВА. АЛГОРИТМИ ПОШУКУ В ГЛИБИНУ ТА В ШИРИНУ

Мета: Освоїти та закріпити прийоми роботи з даними різного типу, організованими у вигляді дерев та їх окремого випадку – бінарних дерев. Здобути практичні навички роботи з графами.

8.1 Хід роботи

Завдання

Маршрути руху автобусів зі станції Києва;

1. Київ –(135) Житомир –(80) Новоград-Волинський –(100) Рівно –(68) Луцьк
2. Київ –(135) Житомир –(38) Бердичів –(73) Вінниця –(110) Хмельницький –(104) Тернопіль
3. Київ –(135) Житомир –(115) Шепетівка
4. Київ –(78) Біла церква –(115) Умань
5. Київ –(78) Біла церква –(146) Черкаси –(105) Кременчук
6. Київ –(78) Біла церква –(181) Полтава – (130) Харків
7. Київ –(128) Прилуки –(175) Суми
8. Київ –(128) Прилуки –(109) Миргород

Порядок виконання роботи

1. Записати матрицю суміжності відповідно до завдання.
2. Реалізувати алгоритми DFS та BFS (самостійно для підвищеної оцінки).
3. Записати можливі маршрути руху та відстані всього маршруту від центрального вокзалу (кореня) до всіх інших вершин.
4. Результати вивести на екран.

Зміст звіту

1. Описати алгоритм (словесна форма, блок-схема алгоритму).
2. Побудувати граф.
3. Привести текст функцій DFS та BFS із коментарями.
4. Висновки щодо роботи.

8.1.1

Завдання:

Лістинг:

```
namespace lab_7_8
{
    class Program
    {
        static void Main()
        {
            Console.OutputEncoding = System.Text.Encoding.Unicode;
            Console.InputEncoding = System.Text.Encoding.Unicode;
            int v = 19;
            int[] dfs = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18 };
            string[] city = { "Київ", "Житомир", "Новоград-Волинський", "Рівне", "Луцьк",
                    "Бердичів","Вінниця","Хмельницький", "Тернопіль",
                    "Шепетівка",
                    "Біла церква", "Умань",
                    "Черкаси", "Кременчук",
                    "Полтава", "Харків",
                    "Прилуки", "Суми",
                    "Миргород"};
            int[] km = { 135, 80, 100, 68, 38, 73, 110, 104, 115, 78, 115, 146, 105, 181,
130, 128, 175, 109 };
            int[,] edges = new int[v, v];
            bool[] visited = new bool[v];
            edges[0, 1] = 1;
            edges[1, 2] = 1;
            edges[2, 3] = 1;
            edges[3, 4] = 1;
            edges[1, 5] = 1;
            edges[5, 6] = 1;
            edges[6, 7] = 1;
            edges[7, 8] = 1;
            edges[1, 9] = 1;
            edges[0, 10] = 1;
            edges[10, 11] = 1;
            edges[10, 12] = 1;
            edges[12, 13] = 1;
            edges[10, 14] = 1;
            edges[14, 15] = 1;
            edges[0, 16] = 1;
            edges[16, 17] = 1;
            edges[16, 18] = 1;
            Console.WriteLine("DFS");
            for (int i = 0; i < v; i++)
            {
                if (!visited[i])
                    DFS(edges, v, visited, i, city);
            }
            for (int i = 0; i < v; i++)
            {
                visited[i] = false;
            }
            Console.WriteLine();
            Console.WriteLine("BFS");
            for (int i = 0; i < v; i++)
            {
                if (!visited[i])
```

```
                    BFS(edges, v, visited, i, dfs[i], city);
                }
                Console.WriteLine($"\nКиїв – Житомир:{km[0]}");
                Console.WriteLine($"Київ – Житомир – Новоград-Волинський:{km[0] + km[1]}");
                Console.WriteLine($"Київ – Житомир – Новоград-Волинський – Рівне:{km[0] +
km[1] + km[2]}");
                Console.WriteLine($"Київ – Житомир – Новоград-Волинський – Рівне –
Луцьк:{km[0] + km[1] + km[2] + km[3]}");
                Console.WriteLine($"Київ – Житомир – Бердичів:{km[0] + km[4]}");
                Console.WriteLine($"Київ – Житомир – Бердичів – Вінниця:{km[0] + km[4] +
km[5]}");
                Console.WriteLine($"Київ – Житомир – Бердичів – Вінниця – Хмельницький:{km[0]
+ km[4] + km[5] + km[6]}");
                Console.WriteLine($"Київ – Житомир – Бердичів – Вінниця – Хмельницький –
Тернопіль:{km[0] + km[4] + km[5] + km[6] + km[7]}");
                Console.WriteLine($"Київ – Житомир – Шепетівка: {km[0] + km[8]}");
                Console.WriteLine($"Київ – Біла церква – Умань: {km[9] + km[10]}");
                Console.WriteLine($"Київ – Біла церква – Черкаси – Кременчук: {km[9] + km[11]
+ km[12]}");
                Console.WriteLine($"Київ – Біла церква – Полтава – Харків: {km[9] + km[13] +
km[14]}");
                Console.WriteLine($"Київ – Прилуки – Суми: {km[15] + km[16]}");
                Console.WriteLine($"Київ – Прилуки – Миргород: {km[15] + km[17]}");
            }
            static void DFS(int[,] edges, int v, bool[] visited, int si, string[] city)
            {
                visited[si] = true;//відвідане
                Console.Write($"{city[si]} ");
                for (int i = 0; i < v; i++)
                {
                    if (i == si)
                        continue;
                    if (!visited[i] && edges[si, i] == 1)
                    {
                        DFS(edges, v, visited, i, city);//пошук зв'язаниг графів
                    }
                }
            }
            static void BFS(int[,] edges, int v, bool[] visited, int j, int si, string[]
city)
            {
                Queue<int> queue = new Queue<int>();//створення черги (перший зайшов, перший
вийшов)
                queue.Enqueue(si);//запис в кінець черги
                visited[si] = true;
                while (queue.Count != 0)
                {
                    int currentVertex = queue.Dequeue();
                    Console.Write($"{city[currentVertex]} ");
                    for (int i = 0; i < v; i++)//пошук графів на цьому рівні
                    {
                        if (i == j)
                            continue;
                        if (!visited[i] && edges[currentVertex, i] == 1)
                        {
                            queue.Enqueue(i);
                            visited[i] = true;
                        }
                    }
                }
            }
        }
    }
```

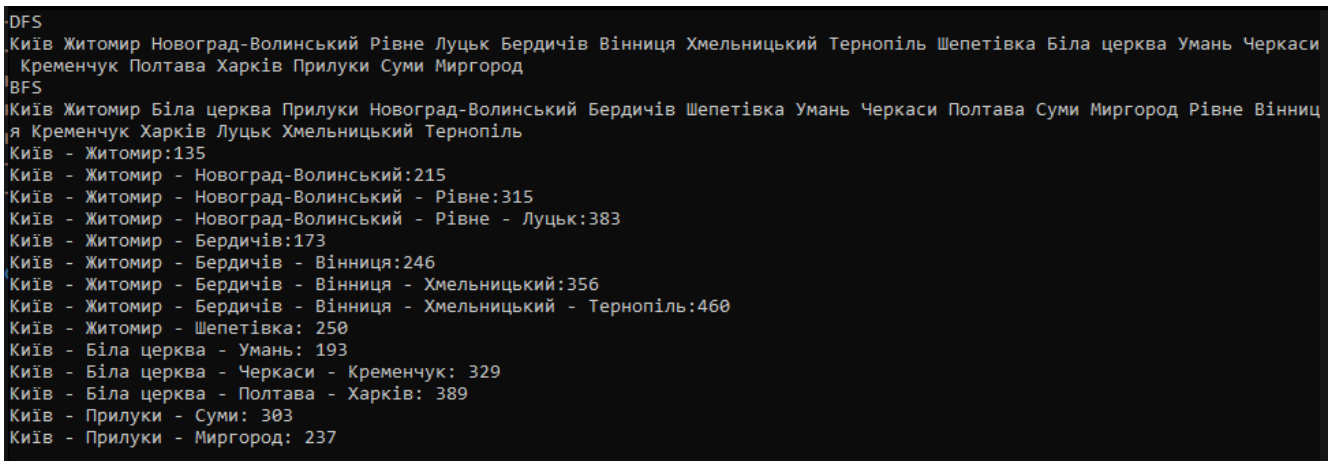

Рисунок 8.1 – Результат виконання завдання

Словесний опис алгоритмів:

Алгоритм пошуку в глибину (DFS) — алгортм для обходу дерева, структури подібної до дерева, або графу. Робота алгоритму починається з кореня дерева (або іншої обраної вершини в графі) і здійснюється обхід в максимально можливу глибину до переходу на наступну вершину.

Наведемо кроки алгоритму

1. Почати з довільної вершини v. Виконати DFS(v):=1. Включити цю вершину в стек.
2. Розглянути вершину у верхівці стеку: нехай це вершина x. Якщо всі ребра, інцидентні вершині x, позначено, то перейти до кроку 4, інакше — до кроку 3.
3. Нехай $\{x, y\}$ — непозначене ребро. Якщо DFS(y) уже визначено, то позначити ребро $\{x, y\}$ штриховою лінією та перейти до кроку 2. Якщо DFS(y) не визначено, то позначити ребро $\{x, y\}$ потовщеною суцільною лінією, визначити DFS(y) як черговий DFS-номер, включити цю вершину в стек і перейти до кроку 2.
4. Виключити вершину x зі стеку. Якщо стек порожній, то зупинитись, інакше — перейти до кроку 2.

Обчислювальна складність: O(n+m)

Пошук у ширину — алгоритм пошуку на графі.Алгоритм має назву пошуку в ширину, оскільки «фронт» пошуку (між пройденими та непройденими вершинами) одноманітно розширюється вздовж всієї своєї ширини. Тобто, алгоритм проходить всі вершини на відстані k перед тим як пройти вершини на відстані k+1.

Наведемо кроки алгоритму

1. Почати з довільної вершини v. Виконати BFS(v):=1. Включити вершину v у чергу.
2. Розглянути вершину, яка перебуває на початку черги; нехай це буде вершина x. Якщо для всіх вершин, суміжних із вершиною x, уже визначено BFS-номери, то перейти до кроку 4, інакше - до кроку 3.
3. Нехай {x,y} - ребро, у якому номер BFS(y) не визначено. Позначити це ребро потовщеною суцільною лінією, визначити BFS(y) як черговий BFS-номер, включити вершину y у чергу й перейти до кроку 2.
4. Виключити вершину x із черги. Якщо черга порожня, то зупинитись, інакше - перейти до кроку 2.

Обчислювальна складність: O(|V|+|E|)

Граф та матриця суміжності:

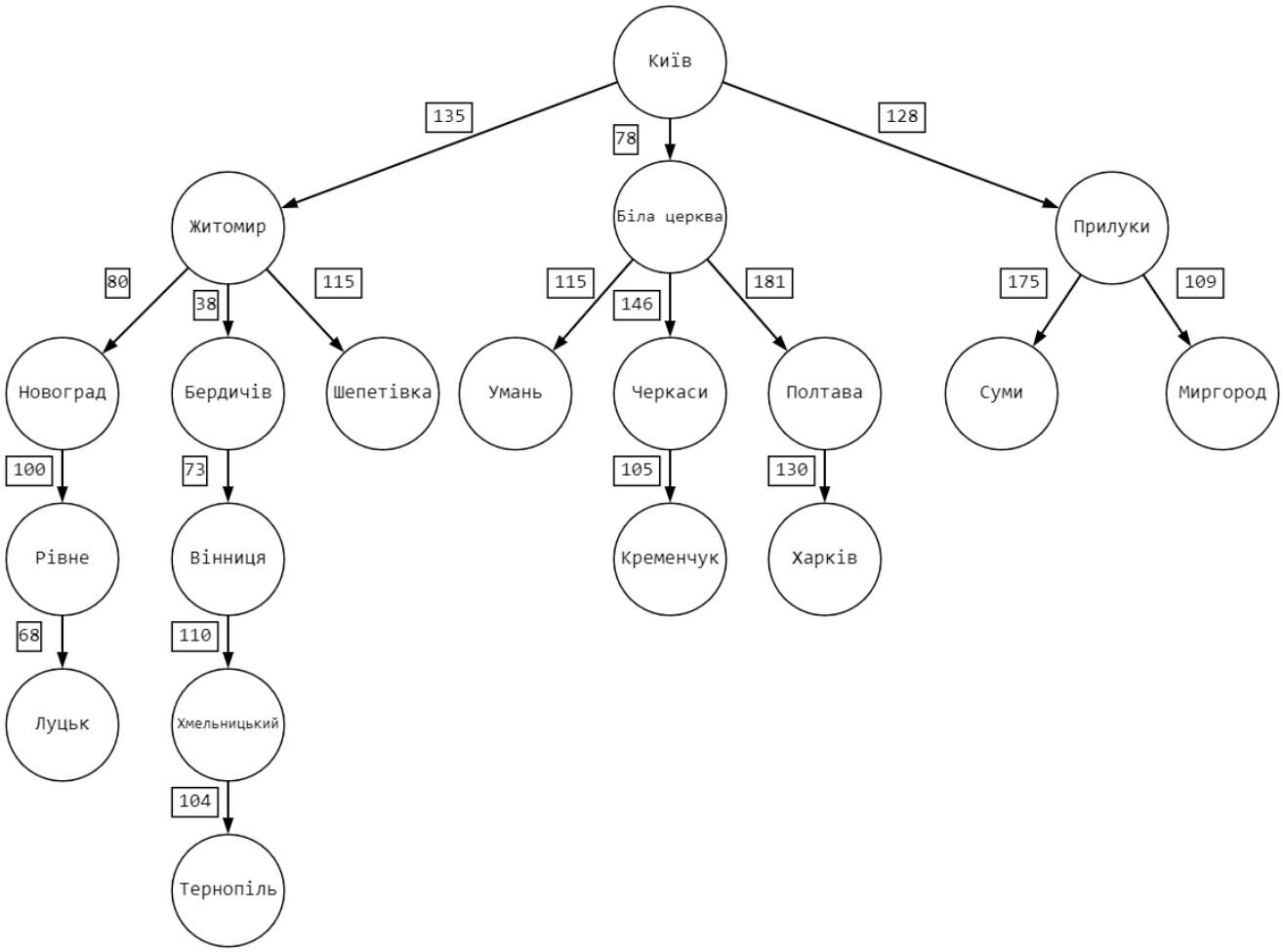

Рисунок 8.2 – Граф

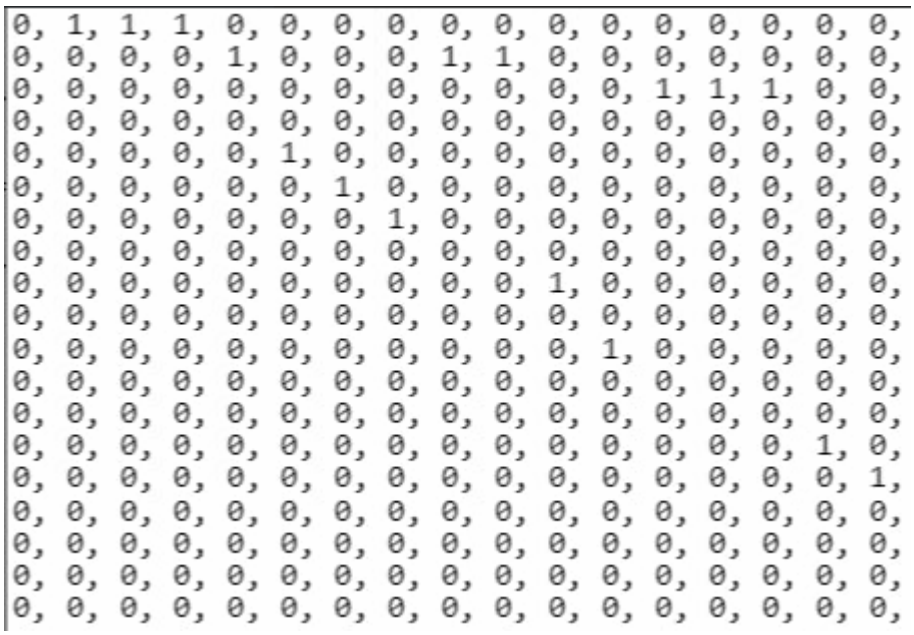

Рисунок 6.3 – Матриця суміжності

Висновки: я освоїв та закріпив прийоми роботи з даними різного типу, організував у вигляді дерев та їх окремого випадку – бінарних дерев. Здобув практичні навички роботи з графами.