Прізвище: Долінський
Ім'я: Олег
Група: КН-406
Варіант: 8
Кафедра: САПР
Дисципліна: Дискретні моделі в САПР
Прийняв: Кривий Р.З.
Посилання: https://github.com/olehdol/labs.git

ЗВІТ
до лабораторної роботи №2
на тему «**Задача листоноші**»

Мета роботи

Отримати практичні навички роботи з пошуку мінімального остового дерева.

Короткі теоретичні відомості

Будь-який листоноша перед тим, як відправитись у дорогу повинен відібрати на пошті листи, що належать до його дільниці, потім він повинен рознести їх адресатам, які розмістились вздовж маршруту його проходження, і повернутись на пошту. Кожен листоноша, бажаючи втратити якомога менше сил, хотів би подолати свій маршрут найкоротшим шляхом. Загалом, задача листоноші полягає в тому, щоб пройти всі вулиці маршруту і повернутися на пошту, мінімізуючи при цьому довжину пройденого шляху.

Якщо граф парний, то оптимальний розв'язок задачі є завжди і ним буде ейлеровий маршрут.

Якщо вершина **"x"** в графі **G** має непарний степінь, то для того, щоб в графі **G*** вершина **"x"** мала парний степінь, листоноша повинен вдруге обійти парну кількість ребер, інцидентних даній вершині. Аналогічно, якщо вершина **"x"** в графі **G** має парний степінь, то для того, щоб в графі **G*** вершина **"x"** мала парний степінь, листоноша повинен ще раз обійти парну кількість ребер, інцидентних цій вершині (нуль є парне число). Якщо ми до кінця простежимо ланцюг ребер, що починається у вершині з непарним степенем, то він обов'язково повинен закінчитись в іншій вершині з непарним степенем. Отже, ребра, які обходять вдруге, створюють ланцюги, початком і кінцем яких є вершини з непарним степенем

Індивідуальне завдання:

```
8
0 0 0 0 86 94 51 82
0 0 81 0 20 87 0 0
0 81 0 83 41 0 0 0
0 0 83 0 8 0 0 0
86 20 41 8 0 40 0 54
94 87 0 0 40 0 89 0
51 0 0 0 0 89 0 18
82 0 0 0 54 0 18 0
```

Виконання:

Якщо вершина **"x"** в графі **G** має непарний степінь, то для того, щоб в графі **G*** вершина **"x"** мала парний степінь, листоноша повинен вдруге обійти парну кількість ребер, інцидентних даній вершині. Аналогічно, якщо вершина **"x"** в графі **G** має парний степінь, то для того, щоб в графі **G*** вершина **"x"** мала парний степінь, листоноша повинен ще раз обійти парну кількість ребер, інцидентних цій вершині (нуль є парне число). Якщо ми до кінця простежимо ланцюг ребер, що починається у вершині з непарним степенем, то він обов'язково повинен закінчитись в іншій вершині з непарним степенем. Отже, ребра, які обходять вдруге, створюють ланцюги, початком і кінцем яких є вершини з непарним степенем. Тому листоноша повинен:

1). вирішити, які вершини з непарним степенем будуть з'єднані ланцюгом ребер, які обходить вдруге.

2). знати точний склад кожного такого ланцюга.

Листоноша може за допомогою будь-якого з алгоритмів побудови найкоротшого шляху визначити на графі **G** найкоротший шлях між кожною парою вершин з непарним степенем.

Для визначення пари вершин з непарним степенем, які повинні бути з'єднані ланцюгом ребер, які обходить вдруге, листоноша може вчинити так. Побудувати граф **G = (X, E)**, множина вершин якого складається з усіх вершин з непарним степенем, а множина ребер з'єднує кожну пару вершин. Присвоїти кожному реброві графу вагу, яка дорівнює деякому дуже великому числу за обрахунком довжини найкоротшого шляху між відповідними вершинами графу **G**.

```
public static Graph PairingOddVertices(Graph graph, Node[] oddNode)
{
    ChinesePostman.graph = (Graph)graph.Clone();
    var pairs = GetOddNodesCombinations(oddNode.ToList());
    var newEdges = CreateEdgeBetweenOddVertices(pairs);
    ChinesePostman.graph.Edges = newEdges.ToArray();
    ChinesePostman.graph.EdgesCount = ChinesePostman.graph.Edges.Length;
    return ChinesePostman.graph;
}
```

```
public static bool IsEvenDegree(Node[] nodes)
{
    foreach (var node in nodes)
    {
        if (node.Rank % 2 != 0)
        {
            return false;
        }
    }
    return true;
}
2 references
private static List<(Node, Node)> GetOddNodesCombinations(List<Node> oddNodes)
{
    Node firstNode = oddNodes.First();
    int minWeight = int.MaxValue;
    List<(Node, Node)> bestPairs = new List<(Node, Node)>();
    (Node, Node) currentPair = (null, null);
    if (oddNodes.Count == 2)
    {
        return new List<(Node, Node)>() { (firstNode, oddNodes.Last()) };
    }
    else
    {
        foreach (var node in oddNodes.Skip(1))
        {
            currentPair = (firstNode, node);
            List<(Node, Node)> newPair = GetOddNodesCombinations(oddNodes.Except(new List<Node>() { node, firstNode }).ToList());
            int weight =
                CalculateWeightOfPairs(newPair);
            if (weight < minWeight)
            {
                minWeight = weight;
                bestPairs = new List<(Node, Node)>() { currentPair };
                bestPairs.AddRange(newPair);
            }
        }
    }
```

+Далі на графі **G** треба побудувати паросполучення з мінімальною вагою. Ці паросполучення з'єднують на графі **G** пари вершин з непарними степенями. Листоноша вдруге повинен обійти ребра, що складають ланцюг найменшої довжини і які з’єднують пару інцидентних паросполученню вершин. Оскільки ці паросполучення мають найменшу загальну вагу, то отриманий внаслідок цього маршрут листоноші повинен мати мінімальну загальну довжину.

```
private static List<Edge> CreateEdgeBetweenOddVertices(List<(Node, Node)> pairsOfOddNodes)
{
    var edges = graph.Edges.ToList();
    foreach (var pair in pairsOfOddNodes)
    {
        if (pair.Item1 != null && pair.Item2 != null)
        {
            var temporary = graph.Edges.FirstOrDefault(x =>
            {
                int source = x.Source;
                Node item1 = pair.Item1;
                int id = item1.Id;
                int source1 = x.Source;
                Node item2 = pair.Item2;
                int id1 = item2.Id;
                return source == pair.Item1.Id
                && x.Destination == pair.Item2.Id
                || x.Destination == id
                && source1 == id1;
            });
            if (temporary != null)
            {
                edges.Add(new Edge()
                {
                    Source = pair.Item1.Id,
                    Destination = pair.Item2.Id,
                    Weight = temporary.Weight
                });
                graph.Nodes.First(x => x.Id == pair.Item1.Id).Rank++;
                graph.Nodes.First(x => x.Id == pair.Item2.Id).Rank++;
            }
        }
    }
    return edges;
}
```

Пошук ейлеревого шляху.

```
while (nodesStack.Count != 0)
{
    var vNode = nodesStack.Peek();
    if (edgesToCompile.Any(x => x.Source == vNode.Id || x.Destination == vNode.Id))
    {
        Edge edgeToRemove = new Edge();
        Node vNodeConnected = new Node();
        if (edgesToCompile.Any(x => x.Source == vNode.Id))
        {
            vNodeConnected = nodesToCompile.First(x => x.Id
            == edgesToCompile.First(x =>
            {
                int source = x.Source;
                return source == vNode.Id;
            }).Destination);
            nodesStack.Push(vNodeConnected);
            edgeToRemove = edgesToCompile.First(x =>
            {
                int destination = x.Destination;
                int source = x.Source;
                int id = vNodeConnected.Id;
                int id1 = vNode.Id;
                return destination == id && source == id1;
            });
        }
        else if (edgesToCompile.Any(x => x.Destination == vNode.Id))
        {
            vNodeConnected = nodesToCompile.First(x =>
            {
                int id = x.Id;
                return id == edgesToCompile.First(x =>
                                                  {
                                                      int destination = x.Destination;
                                                      return destination == vNode.Id;
                                                  }).Source;
            });
            nodesStack.Push(vNodeConnected);
            edgeToRemove = edgesToCompile.First(x =>
            {
                int source = x.Source;
                int destination = x.Destination;
                int id = vNode.Id;
                return source == vNodeConnected.Id && destination == id;
            });

        }
        if (edgeToRemove != null) edgesToCompile.Remove(edgeToRemove);
    }
    else
    {
        nodesStack.Pop();
        result.Add(vNode);
    }
```

Результати:

```
Задача комівояжера
Початкова матриця
        -1      0       0       0       86      94      51      82
        0       -1      81      0       20      87      0       0
        0       81      -1      83      41      0       0       0
        0       0       83      -1      8       0       0       0
        86      20      41      8       -1      40      0       54
        94      87      0       0       40      -1      89      0
        51      0       0       0       0       89      -1      18
        82      0       0       0       54      0       18      -1

Ребра, що додались:
Нова кількість ребер: 16

Джерело Ц?ль    Вага
6       7       18
1       2       81
Шлях

0 ==> 6 ==> 7 ==> 4 ==> 2 ==> 1 ==> 4 ==> 3 ==> 2 ==> 1 ==> 5 ==> 0 ==> 7 ==> 6 ==> 5 ==> 4 ==> 0 =
```

Висновок:

Розв’язано задачу листоноші, що полягає у вирішенні задачі проходження усіх вулиць(ребер) хоча б один раз та повернутись в на початок.