

АЛГОРИТМ ДЕЙКСТРЫ ДЛЯ ПОИСКА КРАТЧАЙШЕГО ПУТИ

Структура графа

```
public class Graph<T>
{
    private NodeList<T> nodeSet; // список узлов графа
    public Graph() : this(null) { }
    public Graph(NodeList<T> nodeSet)
    {
        if (nodeSet == null)
            this.nodeSet = new NodeList<T>();
        else
            this.nodeSet = nodeSet;
    }
    public NodeList<T> Nodes {get{return nodeSet;}}
    public int Count {get{return nodeSet.Count;}}
```

Функционал графа

```
public void AddNode(GraphNode<T> node)
{
    // добавляет узел в граф
    nodeSet.Add(node);
}

public void AddNode(T value)
{
    // добавляет узел в граф
    nodeSet.Add(new GraphNode<T>(value));
}

public void AddDirectedEdge(GraphNode<T> from, GraphNode<T> to, int cost)
{ //Создание направленного ребра сводится
    from.Neighbors.Add(to); //к добавлению узлу, из которого выходим, соседа,
    from.Costs.Add(cost); //а также веса ребра
}

public void AddUndirectedEdge(GraphNode<T> from, GraphNode<T> to, int cost)
{ //Создание ненаправленного ребра сводится к аналогичным действиям как и выше,
    from.Neighbors.Add(to);
    from.Costs.Add(cost);
    to.Neighbors.Add(from); //но в обе стороны
    to.Costs.Add(cost);
}
```

Структура узла

```
public class Node<T>
{
    private T data;
    private NodeList<T> neighbors = null;

    public Node() { }
    public Node(T data) : this(data, null) { } //ссылка на другой конструктор, но neighbors = 0
    public Node(T data, NodeList<T> neighbors)
    {
        this.data = data;
        this.neighbors = neighbors;
    }

    public T Value {get { return data; } set { data = value; } }
    protected NodeList<T> Neighbors { get { return neighbors; } set { neighbors = value; } }
}
```

Родительский класс узла, содержащий свой тип и соседей

```

public class GraphNode<T> : Node<T>
{
    private List<int> costs;
    public GraphNode() : base() { }
    public GraphNode(T value) : base(value) { }
    public GraphNode(T value, NodeList<T> neighbors) : base(value, neighbors) { }

    new public NodeList<T> Neighbors
    {
        get
        {
            if (base.Neighbors == null)
                base.Neighbors = new NodeList<T>();
            return base.Neighbors;
        }
    }
    public List<int> Costs
    {
        get
        {
            if (costs == null)
                costs = new List<int>();
            return costs;
        }
    }
}

```

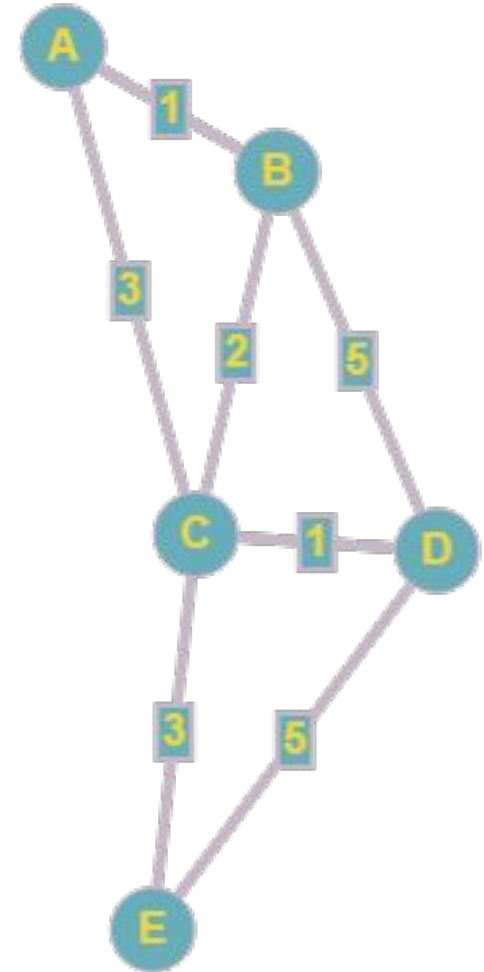
Производный класс узла графа, дополняющий родительский класс весами по отношению к соседям

Создание графа

```
Graph<string> web = new Graph<string>();  
GraphNode<string> node1 = new GraphNode<string>("A");  
GraphNode<string> node2 = new GraphNode<string>("B");  
GraphNode<string> node3 = new GraphNode<string>("C");  
GraphNode<string> node4 = new GraphNode<string>("D");  
GraphNode<string> node5 = new GraphNode<string>("E");
```

```
web.AddNode(node1);  
web.AddNode(node2);  
web.AddNode(node3);  
web.AddNode(node4);  
web.AddNode(node5);
```

```
web.AddUndirectedEdge(node1, node2, 1);  
web.AddUndirectedEdge(node2, node3, 2);  
web.AddUndirectedEdge(node1, node3, 3);  
web.AddUndirectedEdge(node2, node4, 5);  
web.AddUndirectedEdge(node3, node4, 1);  
web.AddUndirectedEdge(node4, node5, 5);  
web.AddUndirectedEdge(node3, node5, 3);
```



Алгоритм Дейкстры

```
class DijkstraData<T>//данные алгоритма
{
    public double Price { get; set; }//кратчайшее расстояние до заданного узла
    public GraphNode<T> Previous { get; set; }//прошлый в обработке узел
}
class Dijkstra<T>
{
    public static List<string> DijkstraAlgorithm(Graph<T> graph, GraphNode<T> choosedNode)//в аргументах граф и нужный узел
    {
        NodeList<T> notVisited = graph.Nodes;//список непосещённых узлов
        //словарь, хранящий в ключе узел и в значении данные алгоритма
        Dictionary<GraphNode<T>, DijkstraData<T>> track = new Dictionary<GraphNode<T>, DijkstraData<T>>();
        track[choosedNode] = new DijkstraData<T> { Previous = null, Price = 0 };//добавление выбранного узла в активные данные
        while (notVisited.Count != 0)//пока непосещённых узлов не останется
        {
            GraphNode<T> toOpen = null;
            double bestPrice = double.PositiveInfinity;//создание лучшего пути и присвоение ему бесконечности
            foreach (var item in notVisited)
            {
                //если в активных данных есть узел из непосещённых и у него путь до выбранного узла меньше нынешнего лучшего пути
                if (track.ContainsKey(item) && track[item].Price < bestPrice)
                {
                    toOpen = item;//то он выбирается для обработки
                    bestPrice = track[item].Price;//и лучший путь обновляется приравниваясь к пути данного узла
                }
            }
            }////!!!в конце цикла будет найден ближайший сосед для обработки
            if (toOpen == null) return null;//если вдруг граф несвязный, в определённый момент алгоритм завершится без ошибок
        }
    }
}
```

Часть 2 и вывод

```
for (int i = 0; i < toOpen.Costs.Count; i++)//цикл по соседям выбранного для обработки узла
{
    var currentPrice = track[toOpen].Price + toOpen.Costs[i];//создание пути для данного соседа сложением пути
    //обрабатываемого узла и веса ребра между соседом и обрабатываемым узлом
    var nextNode = toOpen.Neighbors[i];//создание предполагаемого следующего узла в виде данного соседа
    //если такого узла в активных данных нет или его путь лучше(меньше) записанного в активных данных
    if (!track.ContainsKey(nextNode) || track[nextNode].Price > currentPrice)
    {//то узел будет добавлен в активные данные со своим весом и прошлым узлом, находящимся сейчас на раскрытии(обработке)
        track[nextNode] = new DijkstraData<T> { Price = currentPrice, Previous = toOpen };
    }
}////!!!в конце цикла все недобавленные в активные данные соседи будут добавлены, а у уже добавленных будут обновлены
//величина пути и предыдущий узел, если его предыдущая величина пути была больше
notVisited.Remove(toOpen);//из непосещённых узлов удаляется уже обработанный
}
var result = new List<string>();
foreach (var item in track)
{
    result.Add("К узлу " + item.Key.Value +
        " минимальный путь от узла " +
        choosedNode.Value + " = " + item.Value.Price);
}
return result;
}
```

В итоге, данный алгоритм состоит из двух частей:

- нахождение ближайшего узла к выбранному узлу из ещё необработанных;
- работа с его раскрытием.

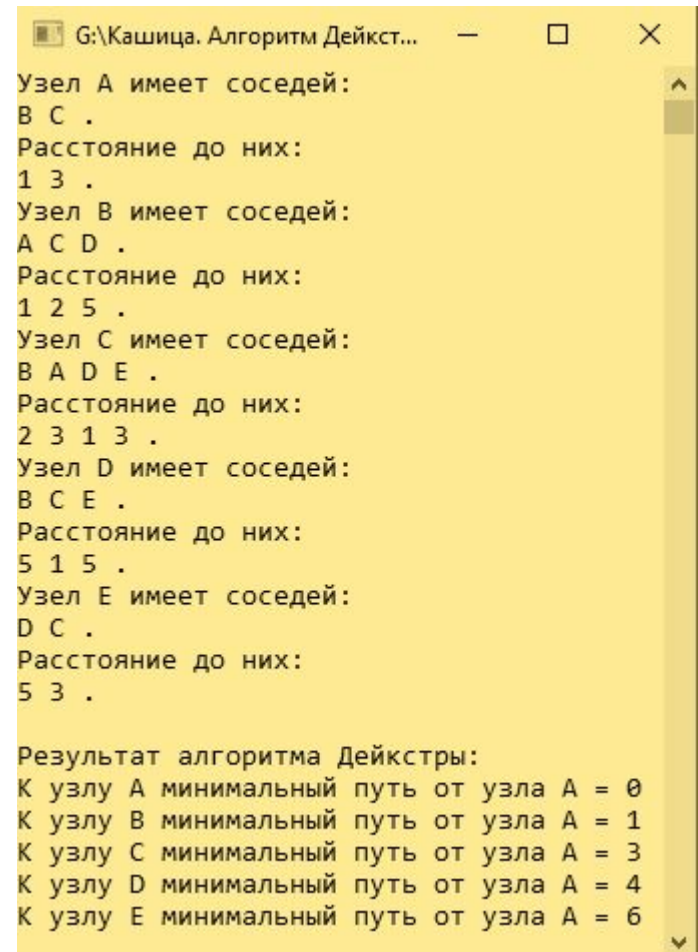
Вывод графа и работы

Алгоритма

```
foreach (var item in web.Nodes)//Вывод
{
    if (item.Neighbors.Count == 0)
    {
        Console.WriteLine("Узел " + item.Value + " не имеет соседей");
    }
    else
    {
        Console.WriteLine("Узел " + item.Value + " имеет соседей:");
        foreach (var neigh in item.Neighbors)
        {
            Console.Write(neigh.Value + " ");
        }
        Console.WriteLine(".");
        Console.WriteLine("Расстояние до них: ");
        foreach (var cost in item.Costs)
        {
            Console.Write(cost + " ");
        }
        Console.WriteLine(".");
    }
}

//Алгоритм Дейкстры
List<string> resultlist = Dijkstra<string>.DijkstraAlgorithm(web, node1);
Console.WriteLine("\nРезультат алгоритма Дейкстры:");
foreach (var item in resultlist)//Вывод алгоритма Дейкстры
{
    Console.WriteLine(item);
}

Console.ReadKey();
```



```
G:\Кашица. Алгоритм Дейкст...
Узел А имеет соседей:
В С .
Расстояние до них:
1 3 .
Узел В имеет соседей:
А С D .
Расстояние до них:
1 2 5 .
Узел С имеет соседей:
В А D E .
Расстояние до них:
2 3 1 3 .
Узел D имеет соседей:
В С E .
Расстояние до них:
5 1 5 .
Узел E имеет соседей:
D С .
Расстояние до них:
5 3 .

Результат алгоритма Дейкстры:
К узлу А минимальный путь от узла А = 0
К узлу В минимальный путь от узла А = 1
К узлу С минимальный путь от узла А = 3
К узлу D минимальный путь от узла А = 4
К узлу E минимальный путь от узла А = 6
```

Сравнение результата вывода

```
G:\Кашица. Алгоритм Дейкст...  
Узел A имеет соседей:  
B C .  
Расстояние до них:  
1 3 .  
Узел B имеет соседей:  
A C D .  
Расстояние до них:  
1 2 5 .  
Узел C имеет соседей:  
B A D E .  
Расстояние до них:  
2 3 1 3 .  
Узел D имеет соседей:  
B C E .  
Расстояние до них:  
5 1 5 .  
Узел E имеет соседей:  
D C .  
Расстояние до них:  
5 3 .  
  
Результат алгоритма Дейкстры:  
К узлу A минимальный путь от узла A = 0  
К узлу B минимальный путь от узла A = 1  
К узлу C минимальный путь от узла A = 3  
К узлу D минимальный путь от узла A = 4  
К узлу E минимальный путь от узла A = 6
```

