Άσκηση 1: Βασική Αναζήτηση Κατά Βάθος

Γράψτε ένα πρόγραμμα που χρησιμοποιεί την μη αναδρομική υλοποίηση του DFS για να διασχίσει ένα γράφημα και να εκτυπώσει τους κόμβους του.

Λύση:

```
using System;
using System.Collections.Generic;

class Graph
{
    private int Vertices;
    private List<int>[] adj;

    public Graph(int vertices)
    {
        Vertices = vertices;
        adj = new List<int>[vertices];
        for (int i = 0; i < vertices; i++)
        {
            adj[i] = new List<int>();
        }
    }

    public void AddEdge(int v, int w)
    {
        adj[v].Add(w);
    }

    public void DFS(int startVertex)
    {
        bool[] visited = new bool[Vertices];
        Stack<int> stack = new Stack<int>();

        stack.Push(startVertex);

        while (stack.Count > 0)
        {
            int vertex = stack.Pop();

            if (!visited[vertex])
            {
                Console.Write(vertex + " ");
                visited[vertex] = true;
            }
```

```
            foreach (int neighbor in adj[vertex])
            {
                if (!visited[neighbor])
                {
                    stack.Push(neighbor);
                }
            }
        }
    }

    static void Main()
    {
        Graph g = new Graph(5);

        g.AddEdge(0, 1);
        g.AddEdge(0, 2);
        g.AddEdge(1, 2);
        g.AddEdge(2, 0);
        g.AddEdge(2, 3);
        g.AddEdge(3, 3);

        Console.WriteLine("Following is Depth First Traversal " +
                          "(starting from vertex 2)");

        g.DFS(2);
    }
}
```

Άσκηση 2: Εύρεση Συνδεδεμένων Συνιστωσών

Γράψτε ένα πρόγραμμα που χρησιμοποιεί την μη αναδρομική υλοποίηση του DFS για να βρει και να εκτυπώσει όλες τις συνδεδεμένες συνιστώσες ενός μη κατευθυνόμενου γραφήματος.

Λύση:

```
using System;
using System.Collections.Generic;

class Graph
{
    private int Vertices;
    private List<int>[] adj;

    public Graph(int vertices)
    {
        Vertices = vertices;
        adj = new List<int>[vertices];
        for (int i = 0; i < vertices; i++)
        {
            adj[i] = new List<int>();
        }
    }

    public void AddEdge(int v, int w)
    {
        adj[v].Add(w);
        adj[w].Add(v);
    }

    private void DFSUtil(int startVertex, bool[] visited)
    {
        Stack<int> stack = new Stack<int>();

        stack.Push(startVertex);

        while (stack.Count > 0)
        {
            int vertex = stack.Pop();

            if (!visited[vertex])
            {
                Console.Write(vertex + " ");
                visited[vertex] = true;
            }
```

```
            foreach (int neighbor in adj[vertex])
            {
                if (!visited[neighbor])
                {
                    stack.Push(neighbor);
                }
            }
        }
    }

    public void ConnectedComponents()
    {
        bool[] visited = new bool[Vertices];

        for (int v = 0; v < Vertices; v++)
        {
            if (!visited[v])
            {
                DFSUtil(v, visited);
                Console.WriteLine();
            }
        }
    }

    static void Main()
    {
        Graph g = new Graph(5);

        g.AddEdge(1, 0);
        g.AddEdge(2, 3);
        g.AddEdge(3, 4);

        Console.WriteLine("Following are connected components");
        g.ConnectedComponents();
    }
}
```

Άσκηση 3: Ανίχνευση Κύκλων σε Κατευθυνόμενο Γράφημα

Γράψτε ένα πρόγραμμα που χρησιμοποιεί την μη αναδρομική υλοποίηση του DFS για να ανιχνεύσει αν υπάρχει κύκλος σε ένα κατευθυνόμενο γράφημα.

Λύση

```
using System;
using System.Collections.Generic;

class Graph
{
    private int Vertices;
    private List<int>[] adj;

    public Graph(int vertices)
    {
        Vertices = vertices;
        adj = new List<int>[vertices];
        for (int i = 0; i < vertices; i++)
        {
            adj[i] = new List<int>();
        }
    }

    public void AddEdge(int v, int w)
    {
        adj[v].Add(w);
    }

    public bool IsCyclic()
    {
        bool[] visited = new bool[Vertices];
        bool[] recursionStack = new bool[Vertices];

        for (int i = 0; i < Vertices; i++)
        {
            if (IsCyclicUtil(i, visited, recursionStack))
                return true;
        }

        return false;
    }

    private bool IsCyclicUtil(int v, bool[] visited, bool[] recursionStack)
```

```
{
    Stack<int> stack = new Stack<int>();
    stack.Push(v);

    while (stack.Count > 0)
    {
        int node = stack.Peek();

        if (!visited[node])
        {
            visited[node] = true;
            recursionStack[node] = true;
        }

        bool hasUnvisitedNeighbors = false;
        foreach (int neighbor in adj[node])
        {
            if (!visited[neighbor])
            {
                stack.Push(neighbor);
                hasUnvisitedNeighbors = true;
            }
            else if (recursionStack[neighbor])
            {
                return true; // Cycle found
            }
        }

        if (!hasUnvisitedNeighbors)
        {
            recursionStack[node] = false;
            stack.Pop();
        }
    }

    return false;
}

static void Main()
{
    Graph g = new Graph(4);
    g.AddEdge(0, 1);
    g.AddEdge(0, 2);
    g.AddEdge(1, 2);
    g.AddEdge(2, 0);
    g.AddEdge(2, 3);
```

```
        g.AddEdge(3, 3);

        if (g.IsCyclic())
            Console.WriteLine("Graph contains cycle");
        else
            Console.WriteLine("Graph doesn't contain cycle");
    }
}
```

Άσκηση 4: Τοπολογική Ταξινόμηση Κατευθυνόμενου Ακυκλικού Γραφήματος (DAG)

Γράψτε ένα πρόγραμμα που χρησιμοποιεί την μη αναδρομική υλοποίηση του DFS για να εκτελέσει τοπολογική ταξινόμηση σε ένα κατευθυνόμενο ακυκλικό γράφημα (DAG).

Λύση:

```
using System;
using System.Collections.Generic;

class Graph
{
    private int Vertices;
    private List<int>[] adj;

    public Graph(int vertices)
    {
        Vertices = vertices;
        adj = new List<int>[vertices];
        for (int i = 0; i < vertices; i++)
        {
            adj[i] = new List<int>();
        }
    }

    public void AddEdge(int v, int w)
    {
        adj[v].Add(w);
    }

    public void TopologicalSort()
    {
        bool[] visited = new bool[Vertices];
        Stack<int> stack = new Stack<int>();
```

```
    for (int i = 0; i < Vertices; i++)
    {
        if (!visited[i])
        {
            TopologicalSortUtil(i, visited, stack);
        }
    }

    while (stack.Count > 0)
    {
        Console.Write(stack.Pop() + " ");
    }
}

private void TopologicalSortUtil(int v, bool[] visited, Stack<int> stack)
{
    Stack<int> dfsStack = new Stack<int>();
    dfsStack.Push(v);

    while (dfsStack.Count > 0)
    {
        int node = dfsStack.Peek();

        if (!visited[node])
        {
            visited[node] = true;

            foreach (int neighbor in adj[node])
            {
                if (!visited[neighbor])
                {
                    dfsStack.Push(neighbor);
                }
            }
        }
        else
        {
            if (dfsStack.Peek() == node)
            {
                stack.Push(dfsStack.Pop());
            }
        }
    }
}
```

```
    static void Main()
    {
        Graph g = new Graph(6);
        g.AddEdge(5, 2);
        g.AddEdge(5, 0);
        g.AddEdge(4, 0);
        g.AddEdge(4, 1);
        g.AddEdge(2, 3);
        g.AddEdge(3, 1);

        Console.WriteLine("Following is a Topological Sort of the given graph:");
        g.TopologicalSort();
    }
}
```

Άσκηση 5: Εύρεση όλων των διαδρομών από κόμβο Α σε κόμβο Β

Γράψτε ένα πρόγραμμα που χρησιμοποιεί την μη αναδρομική υλοποίηση του DFS για να βρει όλες τις διαδρομές από έναν κόμβο αρχής Α σε έναν κόμβο προορισμού Β.

```
using System;
using System.Collections.Generic;

class Graph
{
    private int Vertices;
    private List<int>[] adj;

    public Graph(int vertices)
    {
        Vertices = vertices;
        adj = new List<int>[vertices];
        for (int i = 0; i < vertices; i++)
        {
            adj[i] = new List<int>();
        }
    }

    public void AddEdge(int v, int w)
    {
        adj[v].Add(w);
    }

    public void PrintAllPaths(int s, int d)
    {
```

```
bool[] visited = new bool[Vertices];
List<int> path = new List<int>();
Stack<(int, List<int>)> stack = new Stack<(int, List<int>)>();

path.Add(s);
stack.Push((s, new List<int>(path)));

while (stack.Count > 0)
{
    var (current, currentPath) = stack.Pop();

    if (current
```