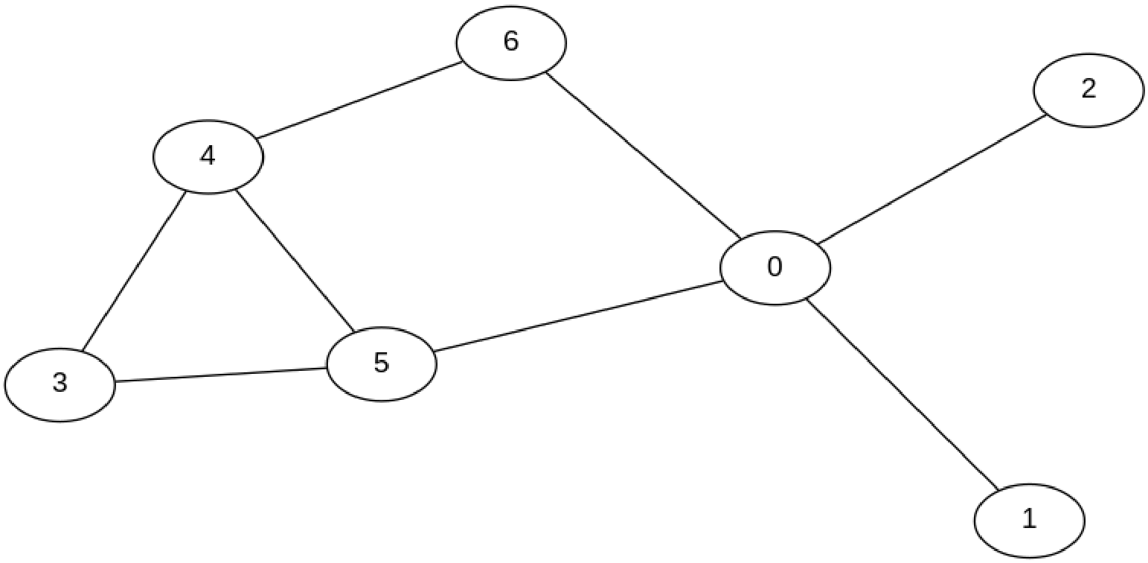
Lecture 11.2 : Searching graphs

Introduction

- We present an approach to searching through a graph called *depth-first search*.
- Depth-first search (DFS) is a recursive algorithm that uses back-tracking to identify and explore novel paths.
- DFS can be used to find all vertices connected to a given vertex.
- DFS can be used to find a path between two vertices (should one exist).
- Before we code it, let's look at DFS in action so we can see how it works.
- Here's a video I made.
- Here's a website with DFS animations.
- Here's a website with lots of algorithm animations.

Our graph



Graph description

- As usual, we describe a graph with a simple text file where the first line defines the number of vertices in the graph and the following lines define the edges (i.e. which vertices are connected to which).

```
$ cat graph01.txt
7
0 1
0 2
0 5
0 6
3 4
3 5
4 5
4 6
```

Basic graph class

- Our basic graph class looks as follows.

```
class Graph(object):

   def __init__(self, V):
      self.V = V
      self.adj = {}
      for v in range(V):
         self.adj[v] = list()

   def addEdge(self, v, w):
      self.adj[v].append(w)
      self.adj[w].append(v)
```

Coding DFS

- We will not add a new method to the `Graph` class but will instead create a new `DFSPaths` class.
- The input to the `DFSPaths` class is the graph we wish to explore using DFS and a starting vertex.

```
class DFSPaths(object):

   def __init__(self, g, s):
      self.g = g
      self.s = s
      self.visited = [False for _ in range(g.V)]
      self.parent = [False for _ in range(g.V)]
      self.dfs(s)

   def dfs(self, v):
      self.visited[v] = True
      for w in self.g.adj[v]:
         if not self.visited[w]:
            self.parent[w] = v
            self.dfs(w)

   # Return True if there is a path from s to v
   def hasPathTo(self, v):
      # This is for you to write
      pass

   # Return path from s to v (or None should one not exist)
   def pathTo(self, v):
      # This is for you to write
      pass
```

Applying DFS to a graph

```
from graph import Graph, DFSPaths

with open('graph01.txt') as f:

   V = int(f.readline())

   g = Graph(V)
```

```
    for line in f:

        v, w = [int(t) for t in line.strip().split()]
        g.addEdge(v, w)

    paths = DFSPaths(g, 0)

    print(paths.hasPathTo(6))

    print(paths.pathTo(6))
```

```
True
[0, 5, 3, 4, 6]
```