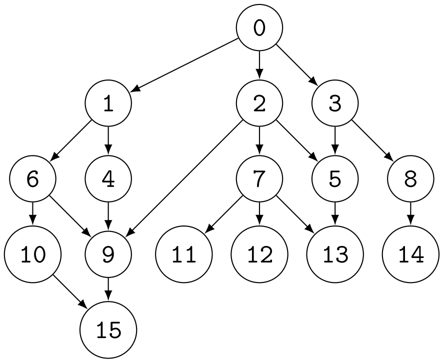
I pledge that I have neither given nor received any unauthorized aid on this assignment.

Input Data

Graph



Adjacency Matrix

```
0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0
0,0,0,0,0,1,0,1,0,1,0,0,0,0,0,0
0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0
0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Data Structures

Graph

Python:

```python
class Graph:
    def __init__(self, num_vertices = 0):
        self.p = [[0 for x in range(num_vertices)] for x in range(num_vertices)];

    def get_adjacent(self, i):
        return [x for x in xrange(self.num_nodes()) if self.p[i][x] != 0];

    def is_edge(self, i, j):
        return self.p[i][j] != 0;

    def num_nodes(self):
        return len(self.p[0]);

    def set_edge(self, i, j, value):
        self.p[i][j] = value;
```

Discussion

For all of the homework problems, I implemented and made use of the simple graph specification (CS 319 Lecture 7, Slide 5). I also added a convenience method, `get_adjacent`, which queries the internal edge matrix for a particular node to find adjacent node indices. For stack and queue structures, the Python `list` contains the functions necessary to avoid deep implementation.

Problem 1 [COMPLETE]

Implementation

Python:

```
def DFS(graph, vertex = 0, visited = None, depth = -1):

    if visited is None:
        visited = [False for x in xrange(graph.num_nodes())];

    if not visited[vertex]:
        visited[vertex] = True;
        sys.stdout.write(str(vertex) + " ");

    for adjacent in graph.get_adjacent(vertex):
        if adjacent is vertex:
            continue;
        if depth != 0 and not visited[adjacent]:
            DFS(graph, adjacent, visited, depth - 1);
```

Output

```
0 1 4 9 15 6 10 2 5 13 7 11 12 3 8 14
```

Problem 2 [COMPLETE]

Implementation

Python:

```
def DFS(graph, vertex = 0, visited = None, depth = -1):

    if visited is None:
        visited = [False for x in xrange(graph.num_nodes())];

    if not visited[vertex]:
        visited[vertex] = True;
        sys.stdout.write(str(vertex) + " ");

    for adjacent in graph.get_adjacent(vertex):
        if adjacent is vertex:
            continue;
        if depth != 0:
            DFS(graph, adjacent, visited, depth - 1);

def IDS(graph):

    visited = [False for x in xrange(graph.num_nodes())];

    for depth in xrange(graph.num_nodes()):
        if all(visited):
            break;
        DFS(graph, 0, visited, depth);
```

Output

```
0 1 2 3 4 6 5 7 9 8 10 13 11 12 15 14
```

Problem 3 [COMPLETE]

Implementation

Python:

```python
def improved_DFS(graph, vertex = 0, visited = None, depth = -1):

    if visited is None:
        visited = [False for x in range(graph.num_nodes())];

    if not visited[vertex]:
        visited[vertex] = True;
        sys.stdout.write(str(vertex) + " ");

    memory = [];

    for adjacent in graph.get_adjacent(vertex):
        if adjacent is vertex:
            continue;
        if depth != 0:
            memory.extend(improved_DFS(graph, adjacent, visited, depth - 1));
        elif not visited[adjacent]:
            memory.append(adjacent);

    return memory;

def improved_IDS(graph, vertex = 0):

    visited = [False for x in range(graph.num_nodes())];
    memory = [vertex];

    if not all(visited):
        while len(memory) > 0:
            memory.extend(improved_DFS(graph, remembered_vertices.pop(0), visited, 0));
            memory = unique(memory);
```

Output

```
0 1 2 3 4 6 5 7 9 8 10 13 11 12 15 14
```