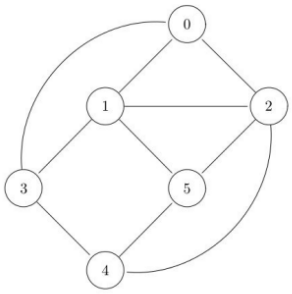
1. Write down the correct sequence of depth first search of the given graph starting at node 0? (The node with larger index will get more priority while visiting) **(2 marks)**

Answer: 0 3 4 5 2 1



2. Write down the adjacency list of the graph given in question 1. **(2 marks)**

Answer:

0-> 1, 2, 3
1-> 0, 2, 3, 5
2-> 0, 1, 4, 5
3-> 0, 1, 4
4-> 2, 3, 5
5-> 1, 2, 4

3. Suppose, you have the adjacency list and the adjacency matrix of the graph given in question 1. Which graph representation will be more efficient (faster to add/delete a node from the graph) and why? **(2 marks**)

Answer:

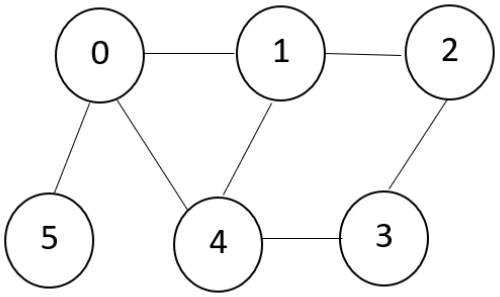
Number of edges = 10. Density = 10/15 = 0.67. So dense graph. Adjacency matrix will be more efficient in terms of time complexity.

4. Write a function 'printSinkNodes' that takes the adjacency list (dictionary) of a directed graph and prints the sink nodes. (Sink node is the node that has no outgoing edges. It can have any number of incoming edges.) **(4 marks)**

```
def printSinkNodes():
  sink_nodes = []
  for x in graph:
    if len(graph[x])==0:
      sink_nodes.append(x)
  print(sink_nodes)
```

1. If we run BFS on the given graph starting at node 4, write down the sequence of visiting the nodes? (The node with smaller index will get more priority while visiting) **(2 marks)**

Answer:4 0 1 3 5 2



2. Write down the adjacency list of the graph given in question 1. **(2 marks)**

Answer:

0-> 1, 4, 5
1-> 0, 1, 2
2-> 1, 3
3-> 2, 4
4-> 0, 1, 3
5-> 0

3. Suppose, you have the adjacency list and the adjacency matrix of the graph given in question 1. Which graph representation will be more efficient (faster to add/delete a node from the graph) and why? **(2 marks**)

Answer:

Number of edges = 7. Density = 7/15 = 0.467. So sparse graph. Adjacency list will be more efficient in terms of time complexity.

4. Write a function 'isReachable' that takes the adjacency list (dictionary) of a graph, a source node (S) and a destination node (D) as inputs. It returns true if there is a path from S to D, otherwise returns false.**(4 marks)**

Answer:

```
visited = set()
def dfs(visited, graph, node):
  if node not in visited:
    visited.add(node)
    for neighbour in graph[node]:
      dfs(visited, graph, neighbour)

def isReachable(graph, S, D):
  dfs(visited, graph, S)
  if D in visited:
    return True
  else:
    return False
```