Worksheet
Graphs

Summary

1. Subject: Data Structures
2. Topic: BFS, DFS
3. Level: Easy

Graph Implementations

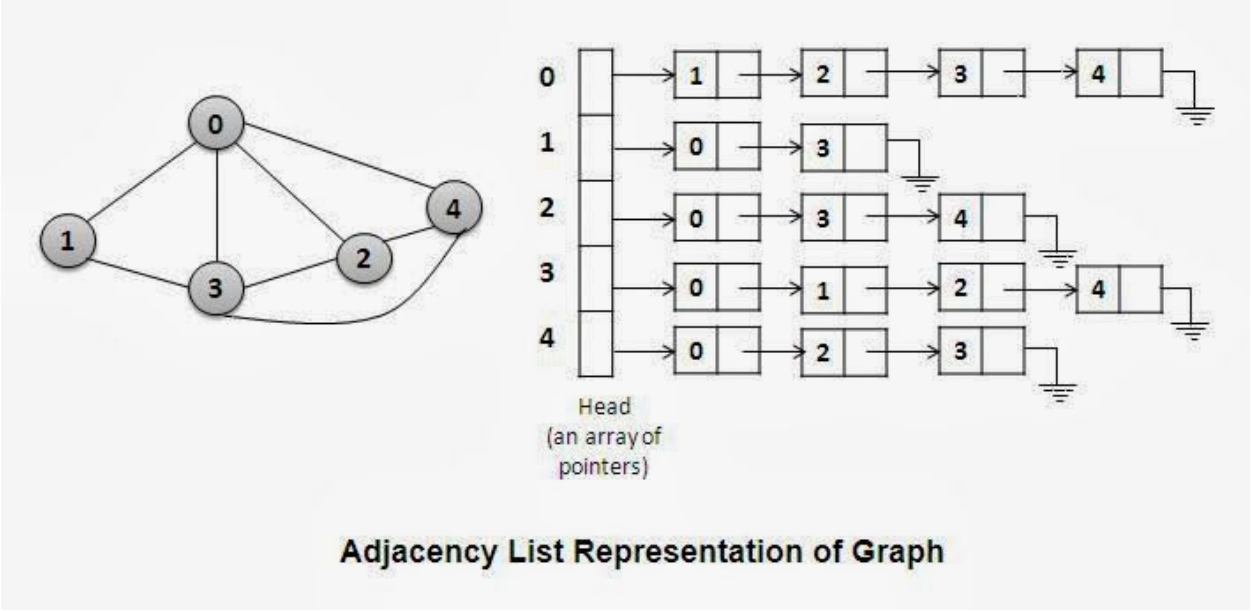
There are three core ways graphs are represented in code. The code snippets are written in python.

Node Model

```
class Node:

     def __init__(self, value):

          self.value = value

          self.neighbors = []
```

In the node model, we initialize a value when we instantiate the object. In order to add “edges” we append nodes into its neighbor field.

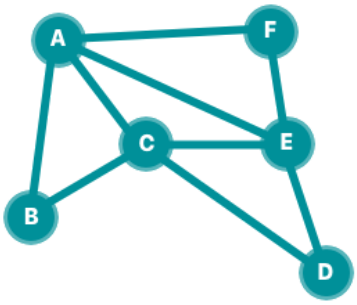
Adjacency List



Adjacency List Representation of Graph

The way we visualize this representation of a graph is that we have a linear collection (array or linked list). Every index represents a node in the graph. At every index, we will store the head reference of another linked list. This other linked list represents the neighbors of the node at an index.

Adjacency Matrix



V: {A,B,C,D,E,F}
E: {AB,AC,AF,BC,CD,CE,DE,EF}

=

	A	B	C	D	E	F
A	0	1	1	0	1	1
B	1	0	1	0	0	0
C	1	1	0	1	1	0
D	0	0	1	0	1	0
E	1	0	1	1	0	0
F	1	0	0	0	1	0

This is the most common representation of graphs in interview problems because it is the easiest to traverse. The matrix will always be n x n and the major diagonal will always be

0's assuming no self loops. Every row and column represents a node. In order to denote if there is an edge between two nodes, we place a 1 in that cell. If there is no edge, we place a 0. Interview problems that use this data structure **often times have** $O(n^2)$ **as the most efficient time complexity** since we must scan the entire matrix to see all possible nodes and edges.

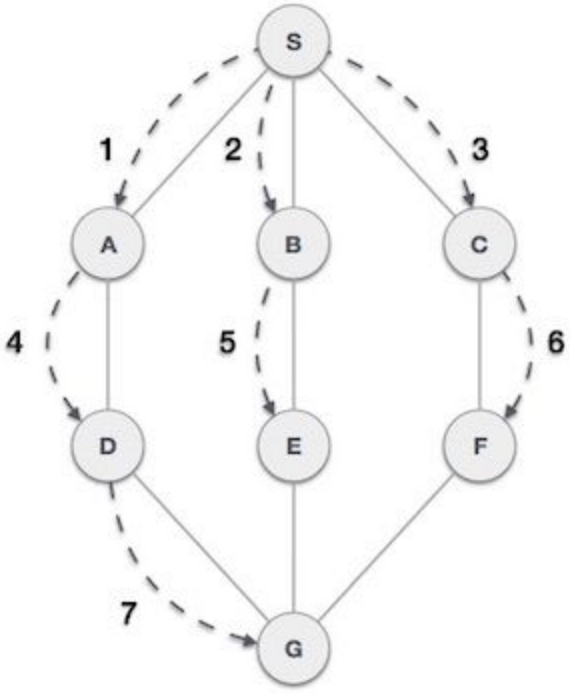
Graph Traversals

Graph traversals come in two flavors: Breadth First Search (BFS) and Depth First Search (DFS). The distinguishing factor between these two algorithms is the data structure used to implement them.

Breadth First Search

This is a 3 step algorithm:

1. Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a **queue**.
2. If no adjacent vertex is found, remove the first vertex from the queue.
3. Repeat Rule 1 and Rule 2 until the queue is empty.



Depth First Search

This is a 3 step algorithm:

1. Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a **stack**.
2. If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
3. Repeat Rule 1 and Rule 2 until the stack is empty.