Data Structures
Graph Traversal

COMP128 Data Structures

Graph Traversal

A traversal is efficient if it visits all the vertices and edges in time proportional to their number (V+E), that is in linear time.

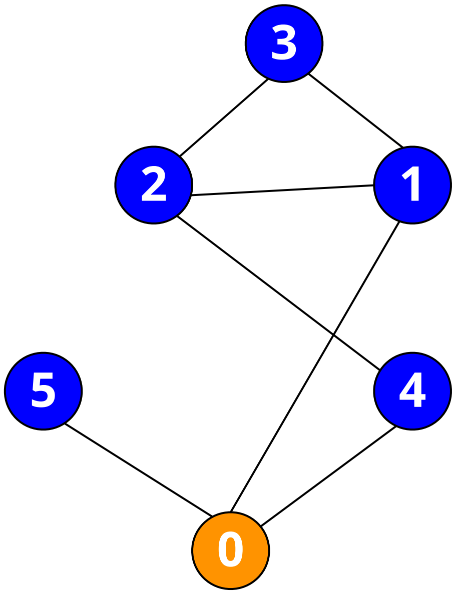
Graph traversal algorithms are key to answering many fundamental questions about graphs involving the notion of reachability.

- Computing a path from vertex u to vertex v, or reporting that no such path exists.
- Identifying a cycle in G, or reporting that G has no cycles.

Breadth First Search (BFS)

- (level 0) BFS starts at vertex s
- (level 1) In the first round, we paint as “visited”, all vertices adjacent to the start vertex s; these vertices are on step away from the beginning;
- (level 2) In the second round, we go two steps (i.e., edges) away from the starting vertex. These new vertices, which are adjacent to level 1 vertices and not previously assigned to a level, are marked as visited.
- This process continues in similar

Breadth First Search (BFS)



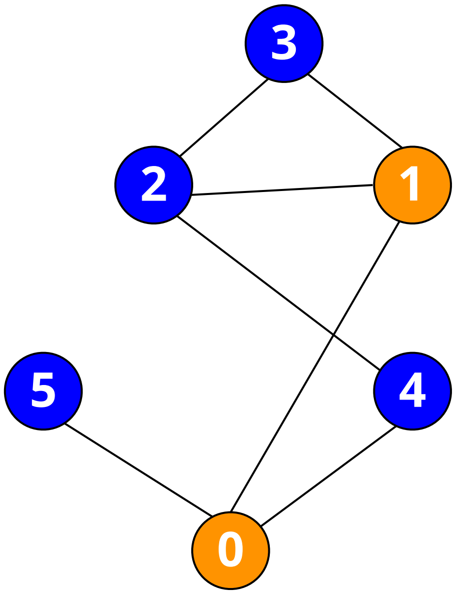
Queue:

0

Adjacency List:

0 -> 1 4 5

Breadth First Search (BFS)



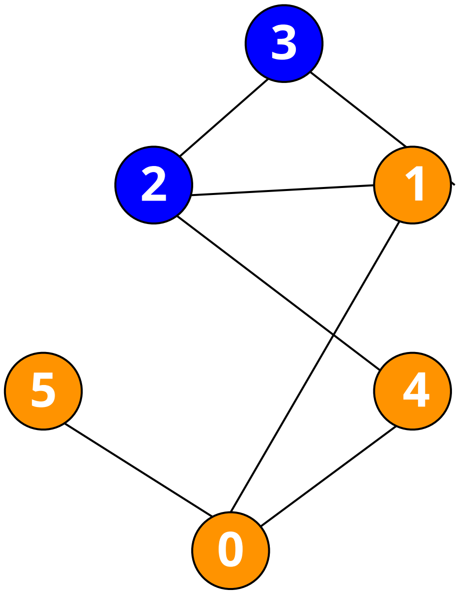
Queue:

0 1

Adjacency List:

0 -> 1 4 5

Breadth First Search (BFS)



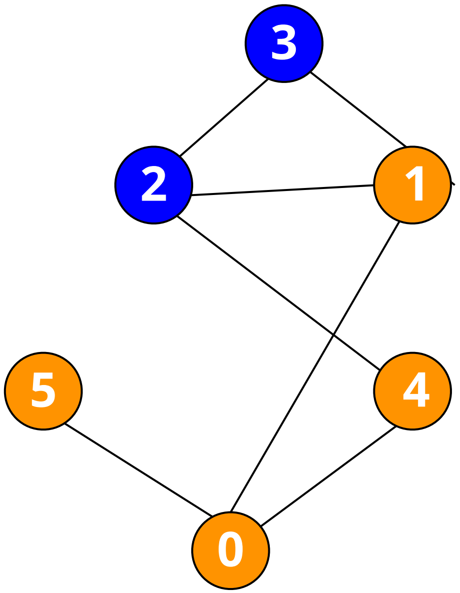
Queue:

1 4 5

Adjacency List:

0 -> 1 4 5

Breadth First Search (BFS)



Queue:

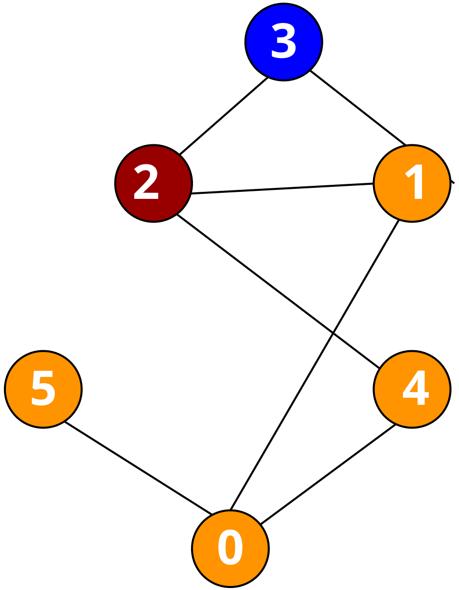
1 4 5

Adjacency List:

~~0 -> 1 4 5~~

1 -> 0 2 3

Breadth First Search (BFS)



Queue:

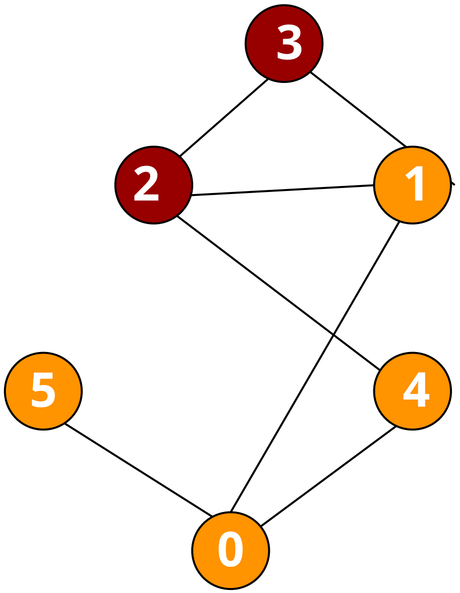
1 4 5 2

Adjacency List:

~~0 -> 1 4 5~~

1 -> 0 2 3

Breadth First Search (BFS)



Queue:

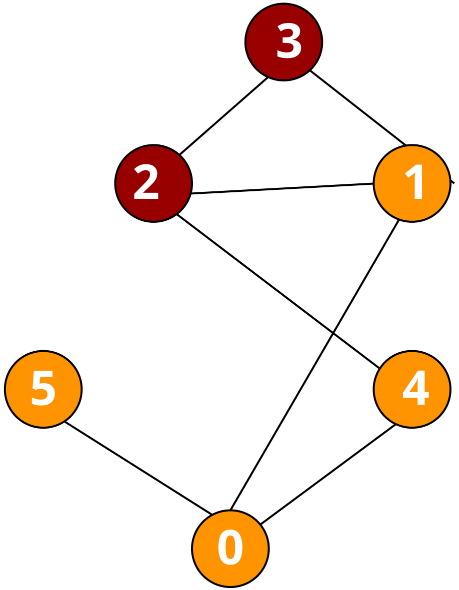
1 4 5 2 3

Adjacency List:

~~0 -> 1 4 5~~

1 -> 0 2 3

Breadth First Search (BFS)



Queue:

4 5 2 3

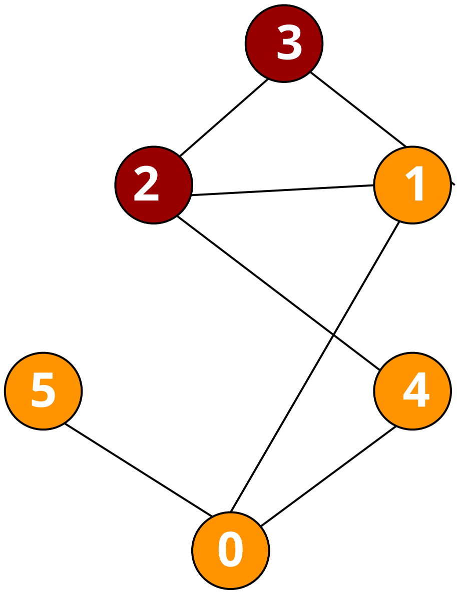
Adjacency List:

~~0 -> 1 4 5~~

~~1 -> 0 2 3~~

4 -> 0 2

Breadth First Search (BFS)



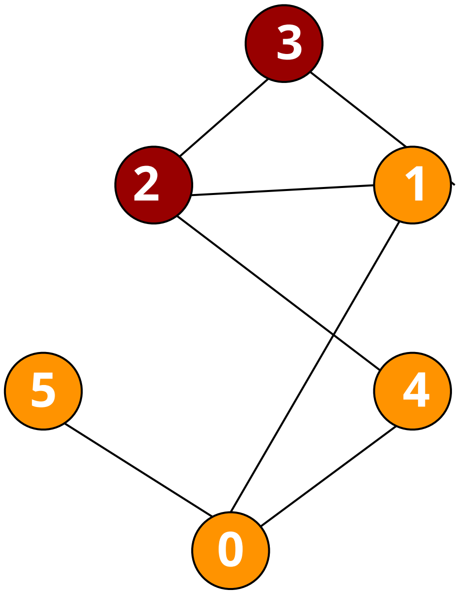
Queue:

5 2 3

Adjacency List:

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

Breadth First Search (BFS)



Queue:

2 3

Adjacency List:

~~0 -> 1 4 5~~

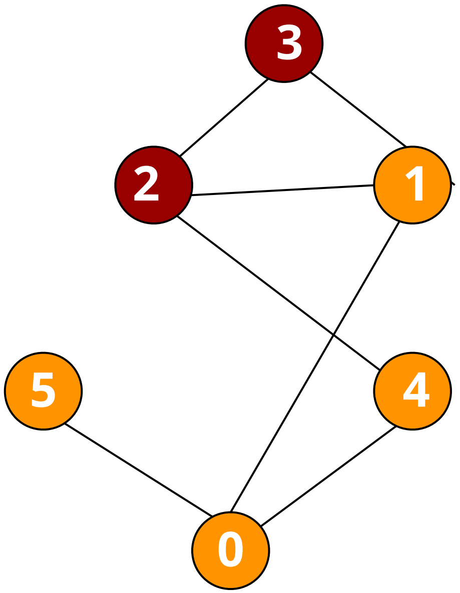
~~1 -> 0 2 3~~

~~4 -> 0 2~~

~~5 -> 0~~

2 -> 1 4

Breadth First Search (BFS)



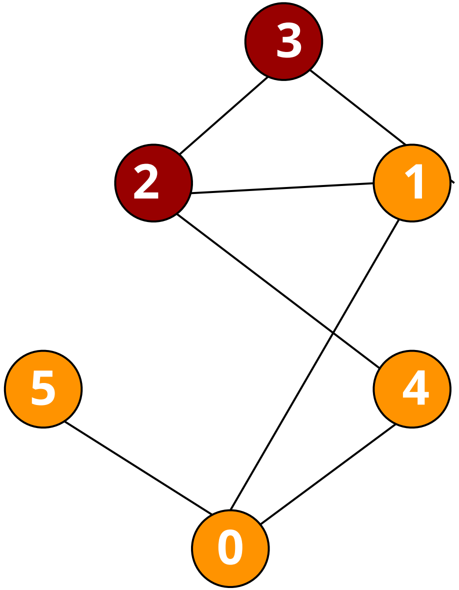
Queue:

3

Adjacency List:

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

Breadth First Search (BFS)



Queue:

Adjacency List:

~~0 -> 1 4 5~~

~~1 -> 0 2 3~~

~~4 -> 0 2~~

~~5 -> 0~~

~~2 -> 1 4~~

~~3 -> 1 2~~

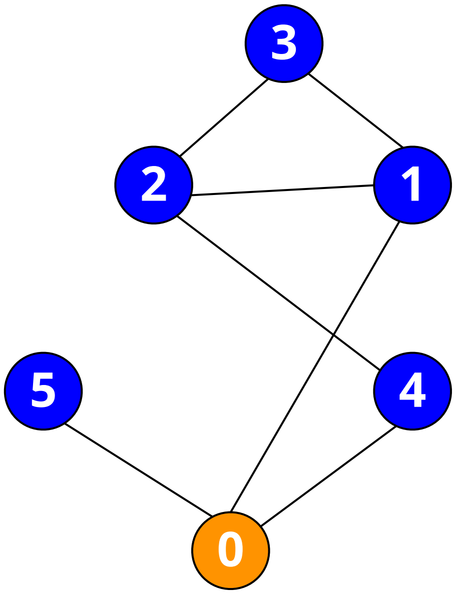
BFS Performance

- Let G be a graph with V vertices and E edges represented with the adjacency list structure. A BFS traversal of G takes O(V+E) time.
- Both DFS and BFS can be used to efficiently find the set of vertices that are reachable from a given source, and to determine paths to those vertices. However, BFS guarantees that those paths use as few edges as possible.

Depth First Search (DFS)

- DFS starts at vertex s
- Mark vertex s as visited.
- Recursively visit each unvisited vertex attached to vertex s.
- This process continues until all vertices are visited.

Depth First Search (DFS)



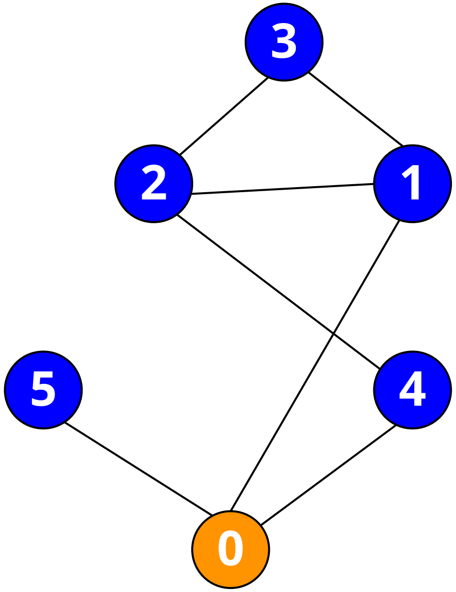
Stack: **Top**

0 **1**

Adjacency List:

0 -> 1 4 5

Depth First Search (DFS)



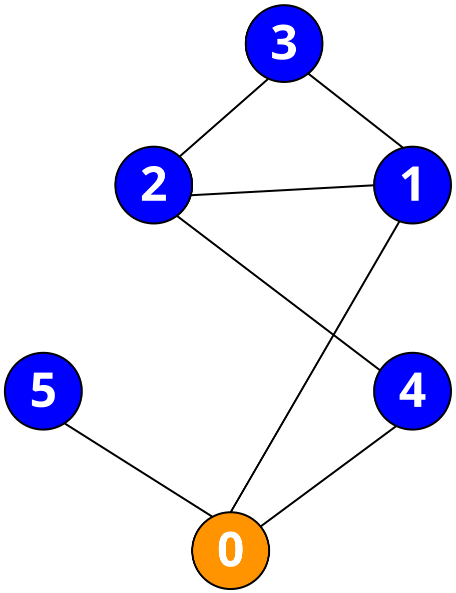
Stack: **Top**

0 1

Adjacency List:

0 -> ~~1~~ 4 5

Depth First Search (DFS)



Stack: **Top**

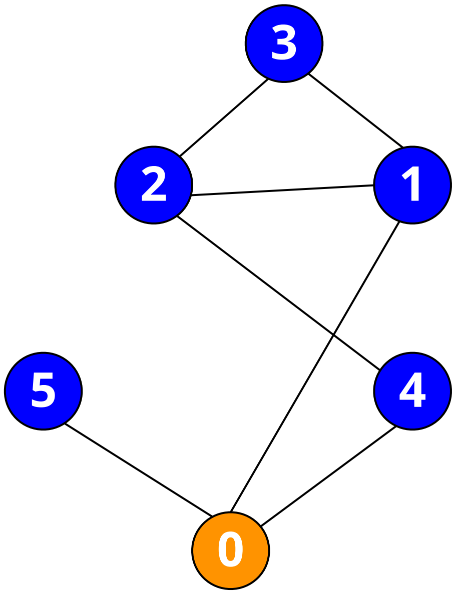
0 **1**

Adjacency List:

0 -> ~~1~~ 4 5

1 -> ~~0~~ 2 3

Depth First Search (DFS)



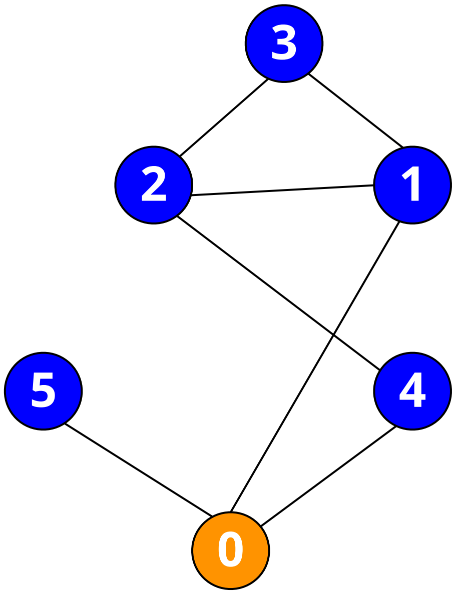
Stack: **Top**

0 1 2

Adjacency List:

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

Depth First Search (DFS)



Stack: **Top**

0 1 2 3

Adjacency List:

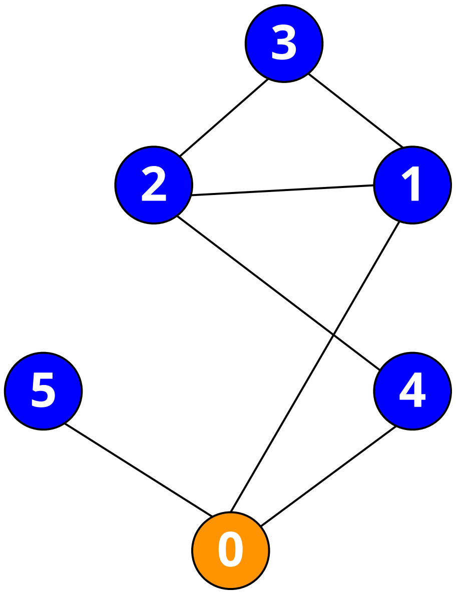
0 -> ~~1~~ 4 5

1 -> ~~0~~ ~~2~~ 3

2 -> ~~1~~ ~~3~~ 4

3 -> ~~1~~ ~~2~~

Depth First Search (DFS)



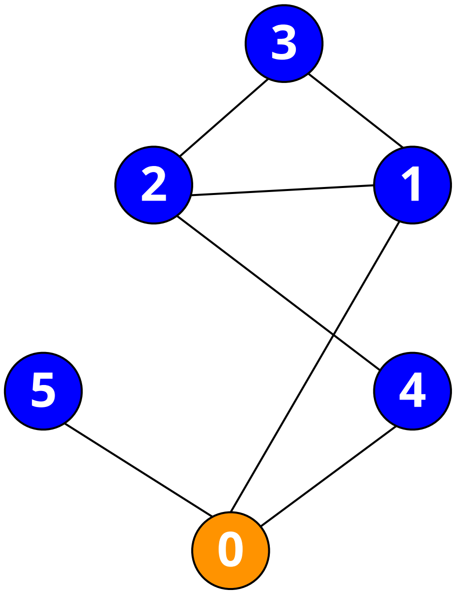
Stack: **Top**

0 1 2

Adjacency List:

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

Depth First Search (DFS)



Stack: **Top**

0 1 2 4

Adjacency List:

0 -> ~~1~~ 4 5

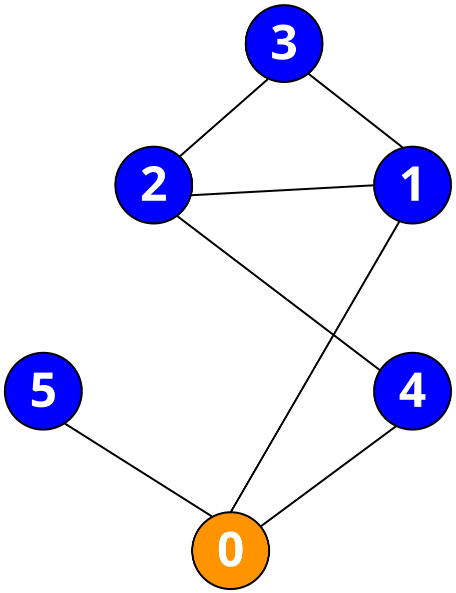
1 -> ~~0~~ ~~2~~ 3

2 -> ~~1~~ ~~3~~ ~~4~~

~~3 -> 1~~ ~~2~~

4 -> ~~0~~ ~~2~~

Depth First Search (DFS)



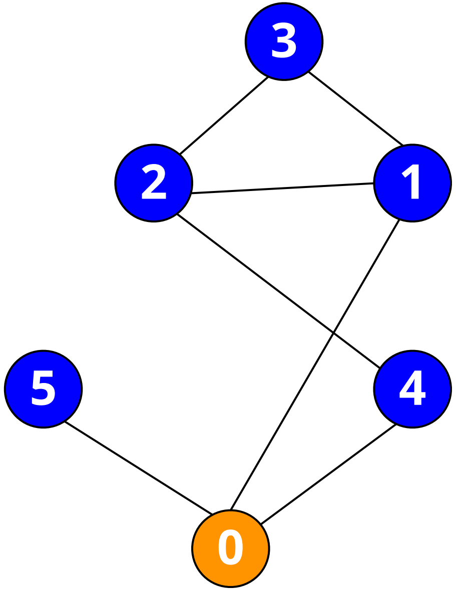
Stack: **Top**

0 1 2

Adjacency List:

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

Depth First Search (DFS)



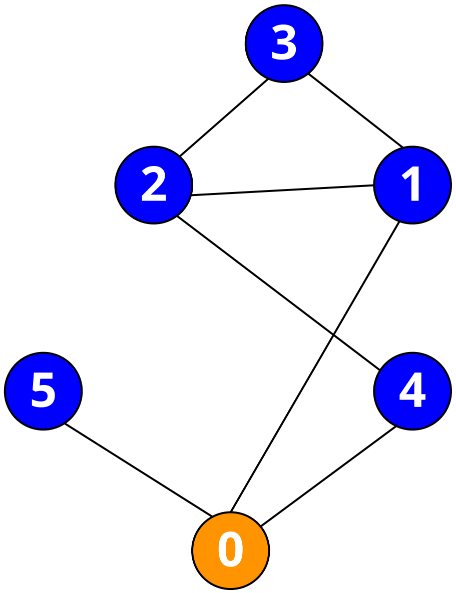
Stack: **Top**

0 1

Adjacency List:

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

Depth First Search (DFS)



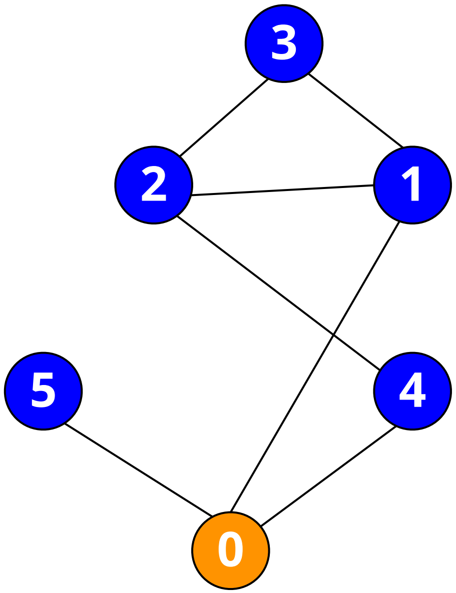
Stack: **Top**

0

Adjacency List:

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

Depth First Search (DFS)



Stack: **Top**

0

Adjacency List:

0 -> ~~1~~ 4 5

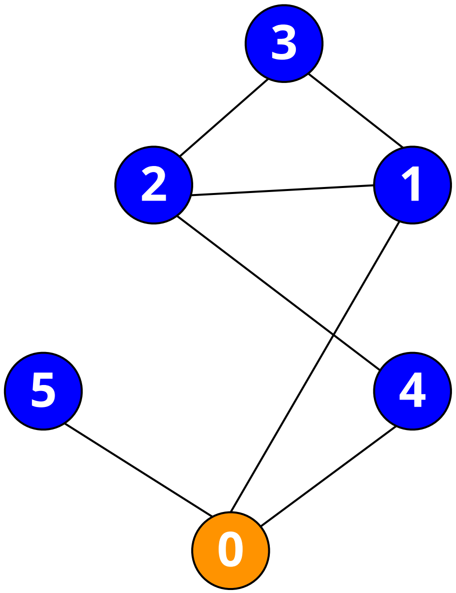
~~1 -> 0 2 3~~

~~2 -> 1 3 4~~

~~3 -> 1 2~~

~~4 -> 0 2~~

Depth First Search (DFS)



Stack: **Top**

0 5

Adjacency List:

0 -> ~~1~~ 4 ~~5~~

~~1 -> 0~~ ~~2~~ ~~3~~

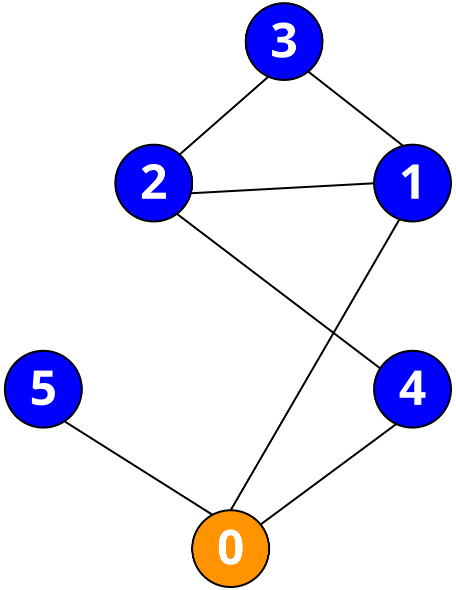
~~2 -> 1~~ ~~3~~ ~~4~~

~~3 -> 1~~ ~~2~~

~~4 -> 0~~ ~~2~~

5 -> ~~0~~

Depth First Search (DFS)



Stack:

Top

0

Adjacency List:

0 -> ~~1~~ 4 ~~5~~

~~1 -> 0~~ ~~2~~ ~~3~~

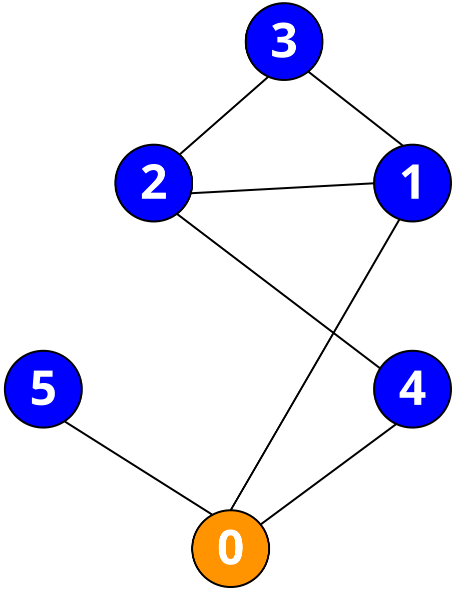
~~2 -> 1~~ ~~3~~ ~~4~~

~~3 -> 1~~ ~~2~~

~~4 -> 0~~ ~~2~~

~~5 -> 0~~

Depth First Search (DFS)



Stack:

Top

Adjacency List:

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

DFS Performance

- Let G be an undirected graph with V vertices and E edges. A DFS traversal of G can be performed in O(V+E) time, and can be used to solve the following problems in O(V+E) time:
  - Computing a path between two given vertices of G, if one exists
  - Testing whether G is connected
  - Computing a spanning tree of G, if G is connected

In-class Activity

Graph Maze Activity