ET0736

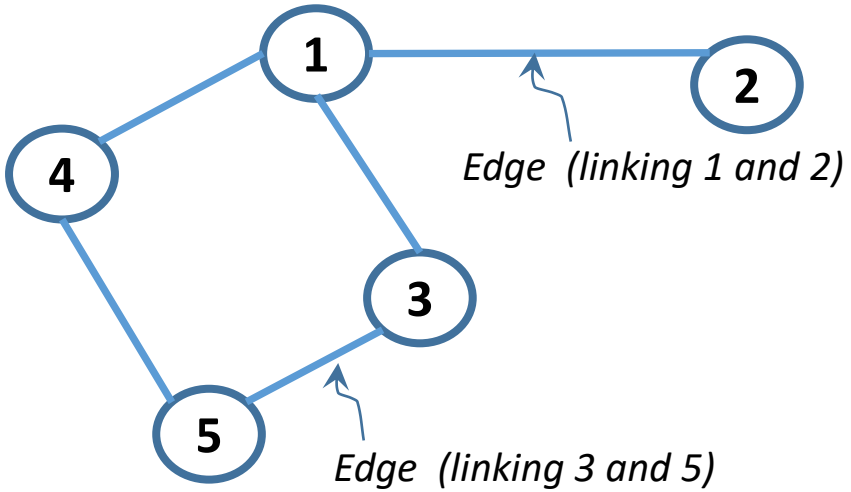
Lesson 9

Graph

Graph

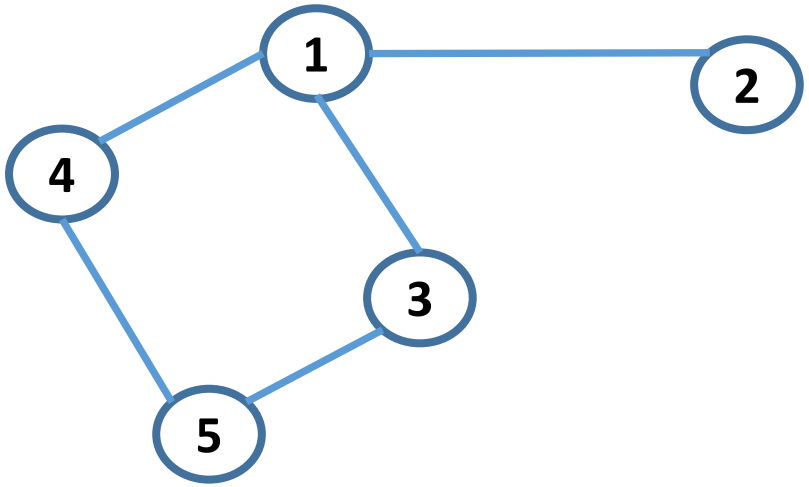
- is a non-linear data structure consisting of
  - vertices
  - edges
- vertices are connected by edges.
- used to represent relationships between different entities.

- Example: There are 5 vertices in the following graph: 1, 2, 3, 4 and 5



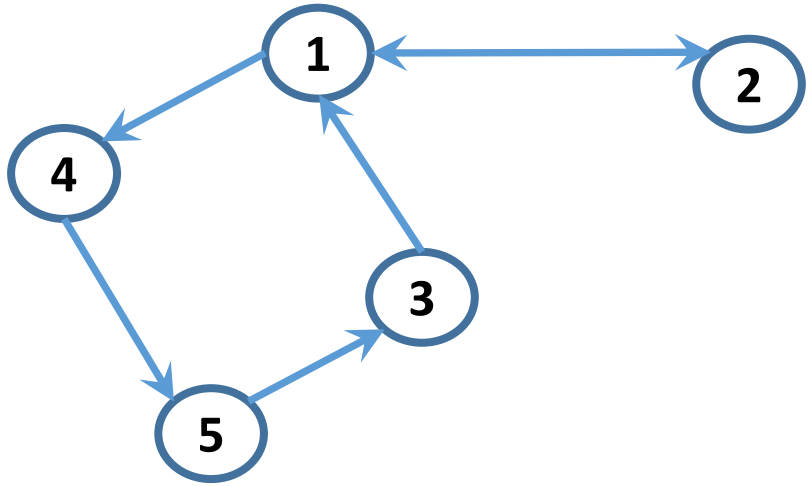
Graph

- A simple undirected graph has the edges as bidirectional (i.e. vertex 2 can go to vertex 1 and vis versa)
- A simple unweighted graph has no value (such as distance) attached to its edge



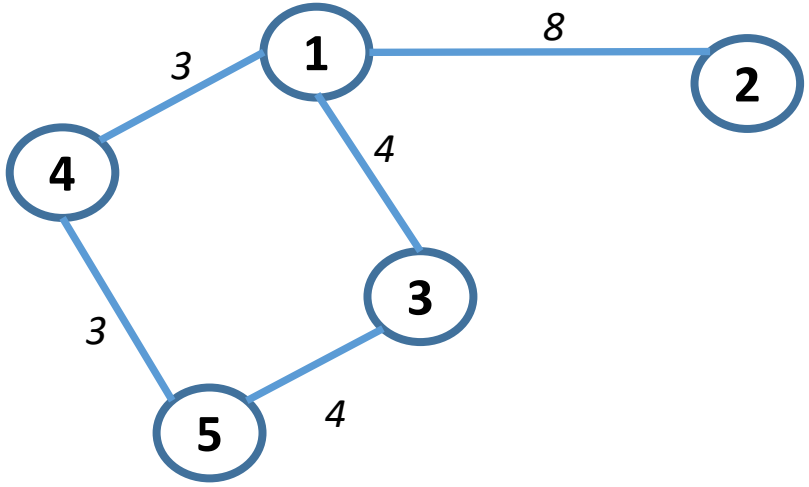
Graph

- Example of directed graph



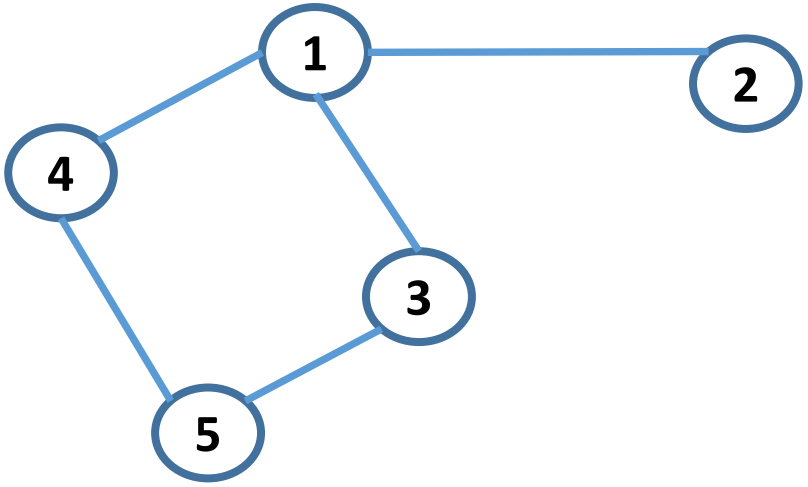
Graph

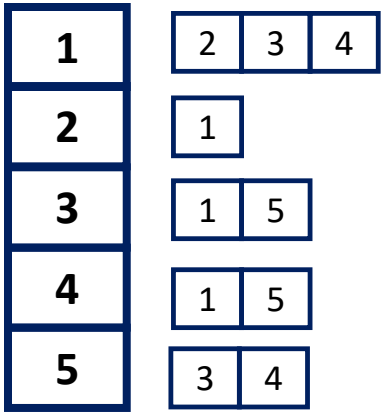
- Example of weighted graph



Representation of Simple Graph

- Use a simple ***ArrayList*** to house all the vertices 1, 2, 3, 4 and 5
- Use another ***ArrayList*** for each vertex to house all its neighboring vertices, aka adjacent list (i.e. edges)





Traversal of Graph

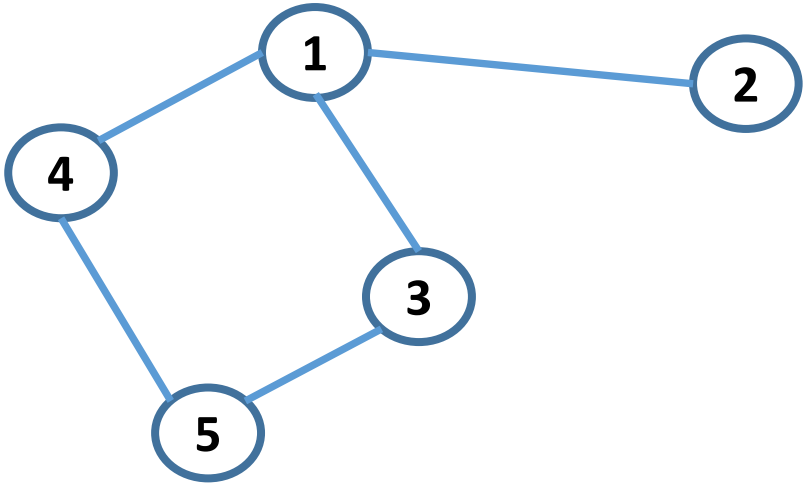
There are 2 fundamental methods in traversing a graph:

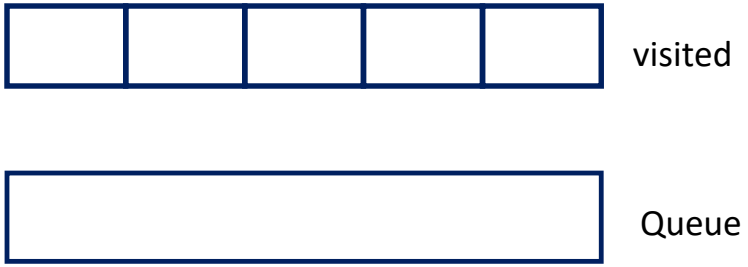
- Breadth First Search (BFS)
- Depth First Search (DFS)

The aim is to visit all the vertices and every vertex can only be visited once.

Breath First Search

- Prepare a visited List (initially empty)
- Prepare a Queue (initially empty)



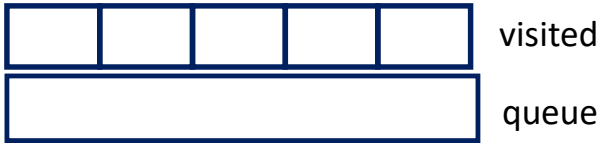


Breath First Search

Steps:

put the root (starting) vertex in queue
while (queue is not empty)
    poll() a vertex from queue
    if (vertex has not been visit before)
      process the vertex (whatever process is needed)
      push all the neighbours (from adjacent list) into queue

repeat till queue is empty



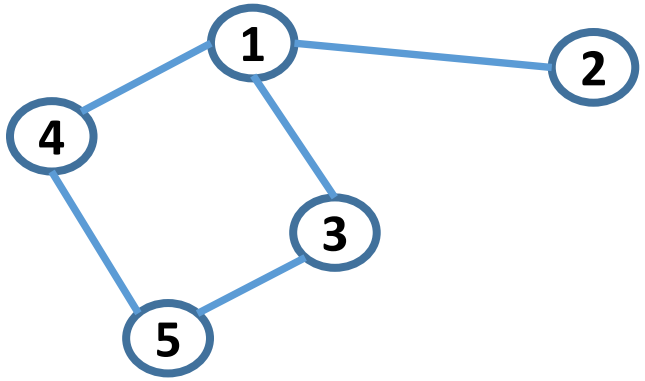
Breath First Search

- Starting from a root (say vertex 2)
- Insert the root into Queue

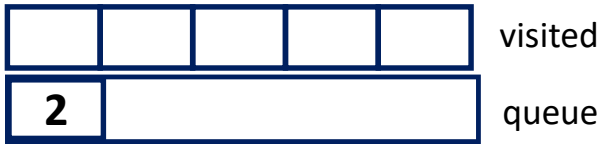
Loop until queue is empty

- poll() a vertex from Queue (it is 2)
- 2 has not been visited
- mark 2 the as visited (insert 2 into visited)
- process the vertex
- push all its adjacent vertices in the queue (which is 1)

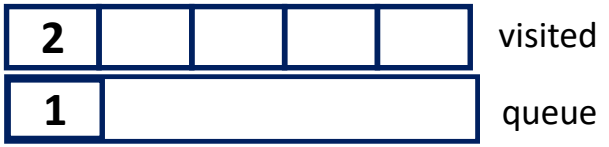
repeat till queue is empty



Before 1st loop



At end of 1st loop



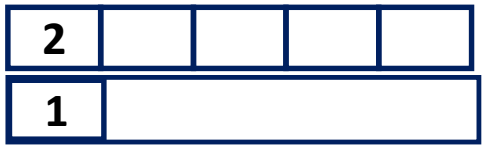
Breath First Search

Loop until queue is empty

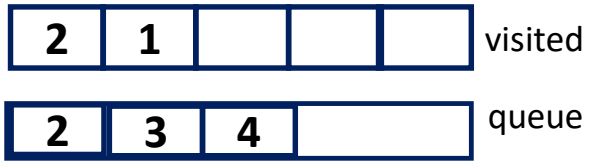
- poll() a vertex from Queue (it is 1)
- 1 has not been visited
- mark 1 the as visited (insert 1 into visited)
- process the vertex
- push all its adjacent vertices in the queue (which are 2, 3 and 4)

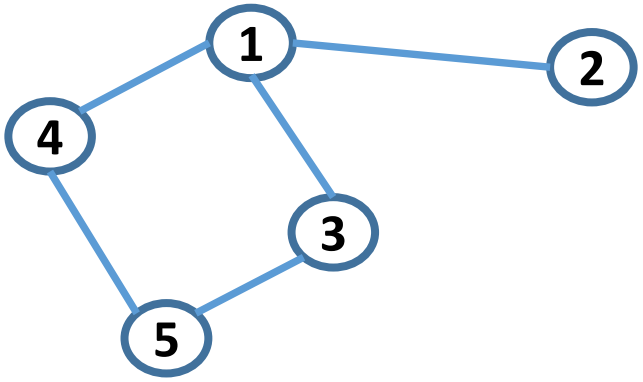
repeat till queue is empty

Before 2nd loop



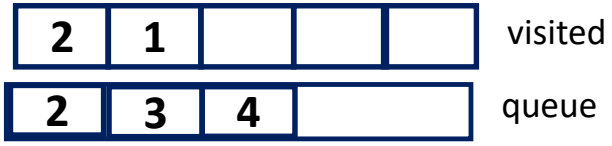
End of 2nd loop





Breath First Search

Before 3rd loop



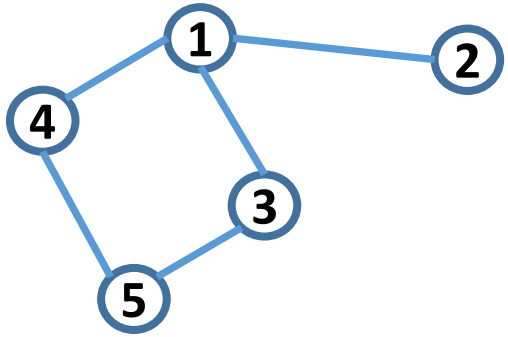
Loop until queue is empty

- poll() a vertex from Queue (It is 2. It has been visited before. Skip. Go to next loop)

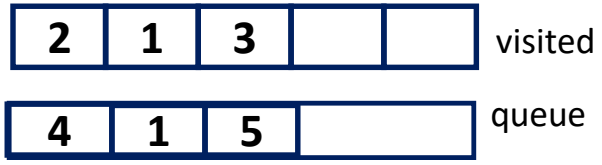
Loop until queue is empty

- poll() a vertex from Queue (it is 3)
- 3 has not been visited
- mark 3 the as visited (insert 3 into visited)
- process the vertex
- push all its adjacent vertices in the queue (which are 1 and 5)

repeat till queue is empty

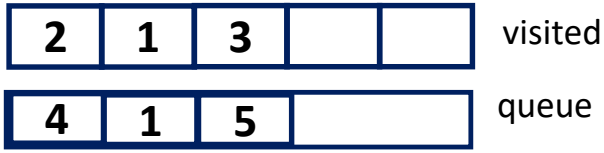


End of 4th loop



Breath First Search

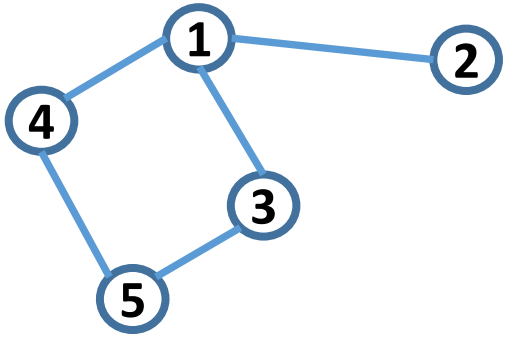
Before 5th loop



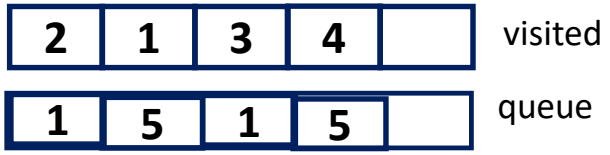
Loop until queue is empty

- poll() a vertex from Queue (it is 4)
- 4 has not been visited
- mark 4 the as visited (insert 4 into visited)
- process the vertex
- push all its adjacent vertices in the queue (which are 1 and 5)

repeat till queue is empty

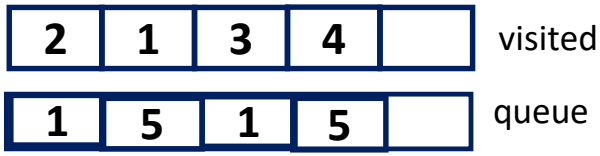


End of 5th loop



Breath First Search

Before 6th loop



Loop until queue is empty

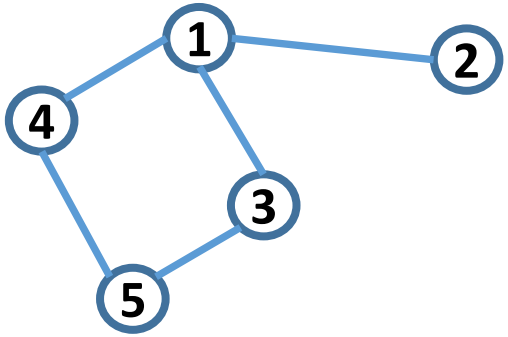
- poll() a vertex from Queue (It is 1. It has been visited before. Skip. Go to next loop)

Loop until queue is empty

- poll() a vertex from Queue (it is 5)
- 5 has not been visited
- Mark 5 the as visited (insert 5 into visited)
- process the vertex
- push all its adjacent vertices in the queue (which are 3 and 4)

repeat till queue is empty

End of 7th loop



Breath First Search

Before 8th loop

2	1	3	4	5	visited
1	5	3	4		queue

Loop until queue is empty

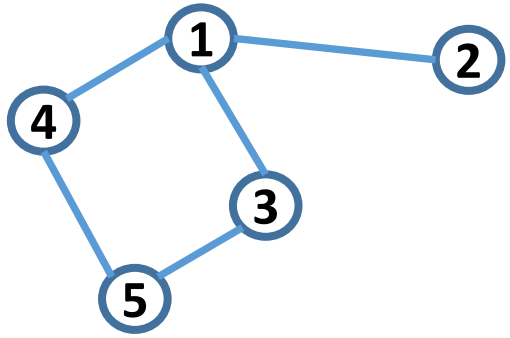
- poll() a vertex from Queue (It is 1. It has been visited before. Skip. Go to next loop)
- poll() a vertex from Queue (It is 5. It has been visited before. Skip. Go to next loop)
- poll() a vertex from Queue (It is 3. It has been visited before. Skip. Go to next loop)
- poll() a vertex from Queue (It is 4. It has been visited before. Skip. Go to next loop)

Queue is empty. Stop.

All vertices have been visited in the order 2, 1, 3, 4 then 5.

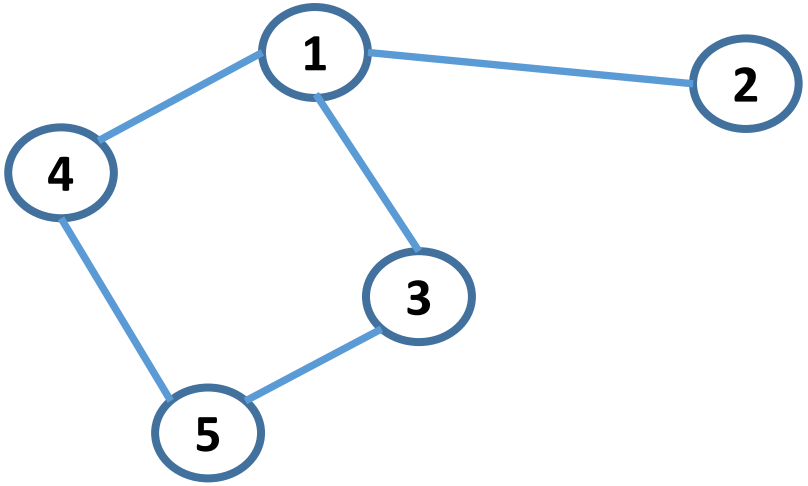
End of 11th loop

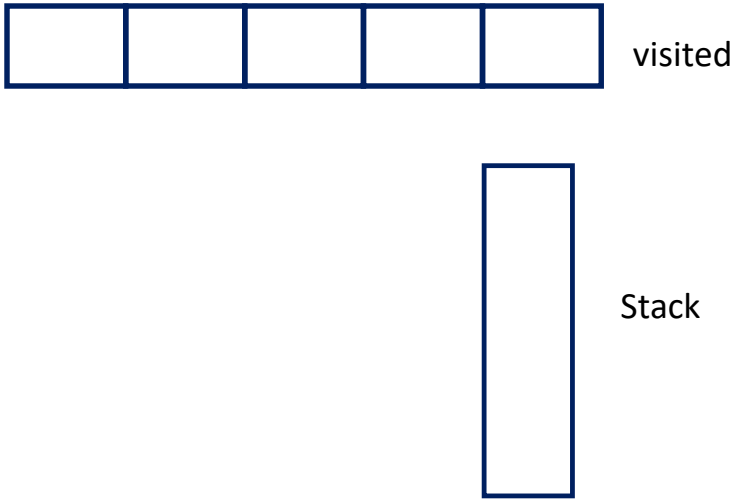
2	1	3	4	5	visited
					queue



Depth First Search

- The aim again is to visit all the vertices and every vertex can only be visited once
- Prepare a visited List (initially empty)
- Prepare a Stack (initially empty)



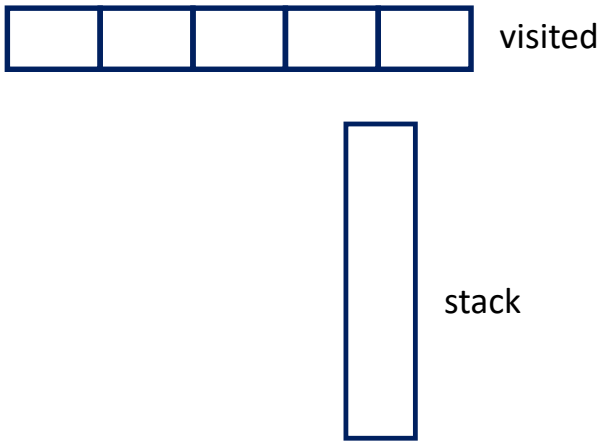


Breath First Search

Steps:

put the root (starting) vertex in stack
while (stack is not empty)
    pop() a vertex from stack
    if (vertex has not been visit before)
      process the vertex (whatever process is needed)
      push all the neighbours (from adjacent list) onto stack

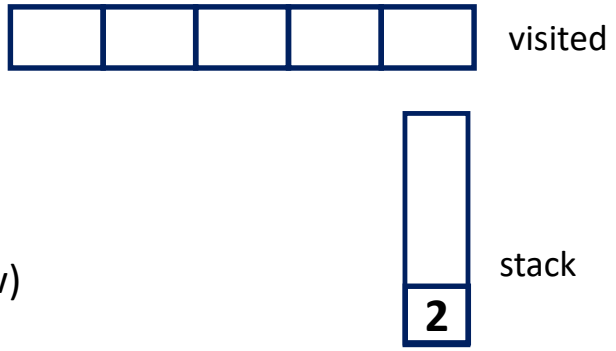
repeat till stack is empty



Depth First Search

- Starting from a root (say 2)
- Insert the root into Stack

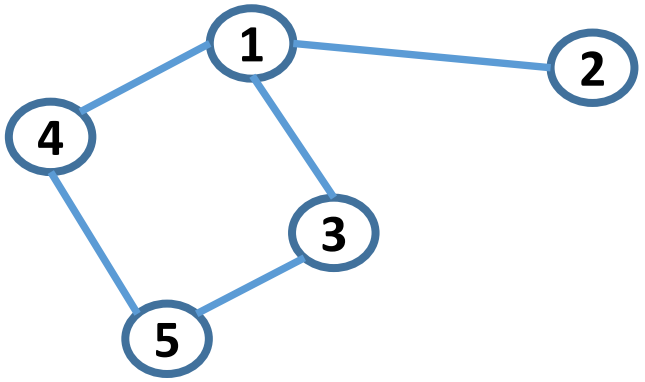
Before 1st loop



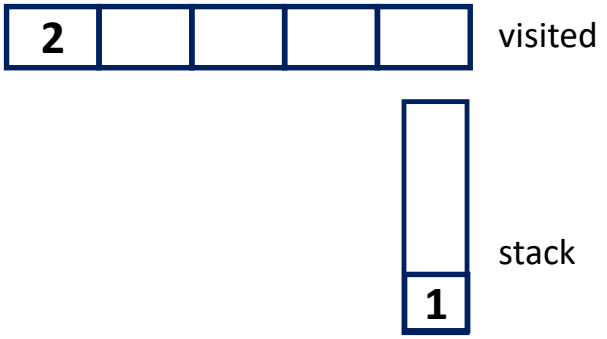
Loop until stack is empty

- pop() a vertex from top of Stack (2 is only one in the stack now)
- 2 has not been visited
- mark the 2 as visited (insert 2 into visited)
- process 2 (can be as simple as print it out)
- push all its adjacent vertices onto stack (has only 1 adjacent, which is 1)

repeat till stack is empty

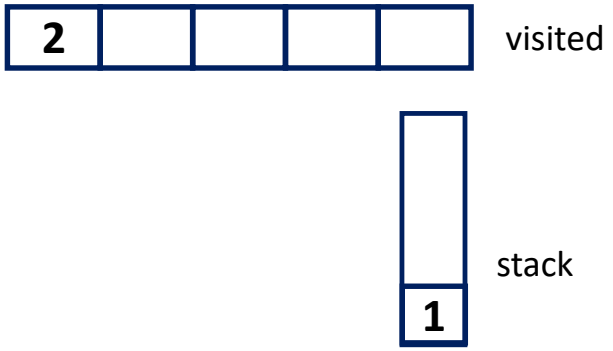


At end of 1st loop



Depth First Search

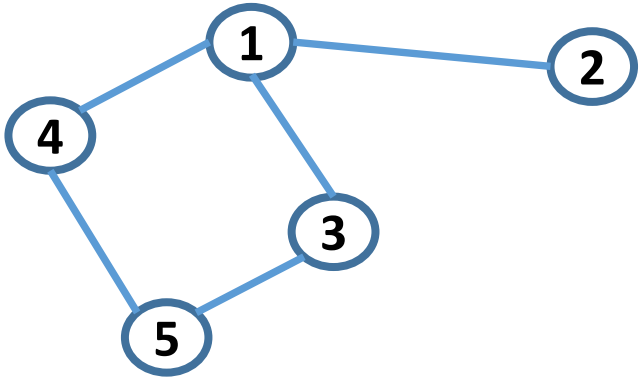
Before 2nd loop



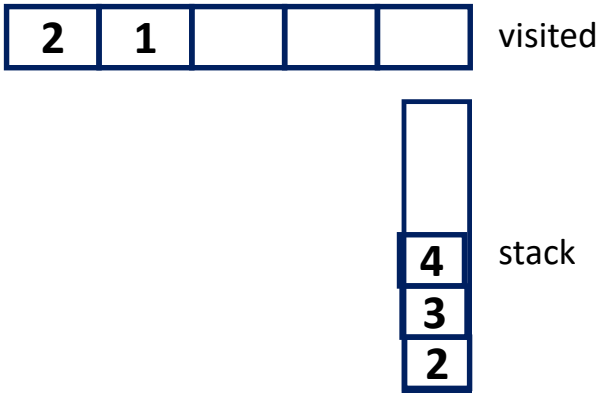
Loop until stack is empty

- pop() a vertex from top of Stack (1 is only one in the stack now)
- 1 has not been visited
- mark the 1 as visited (insert 1 into visited)
- process 1 (can be as simple as print it out)
- push all its adjacent vertices onto stack (which are 2, 3 and 4)

repeat till stack is empty



At end of 2nd loop

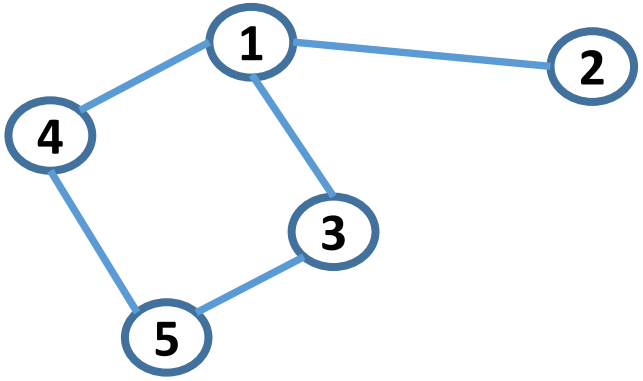


Depth First Search

Loop until stack is empty

- pop() a vertex from top of Stack (it is 4)
- 4 has not been visited
- mark the 4 as visited (insert 4 into visited)
- process 4 (can be as simple as print it out)
- push all its adjacent vertices onto stack (which are 1 and 5)

repeat till stack is empty



Before 3rd loop

visited

stack

At end of 3rd loop

visited

stack

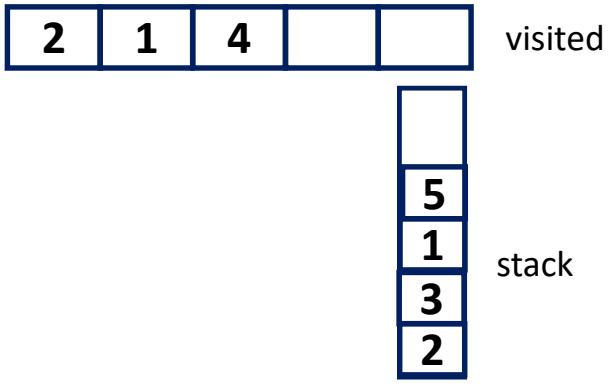
Depth First Search

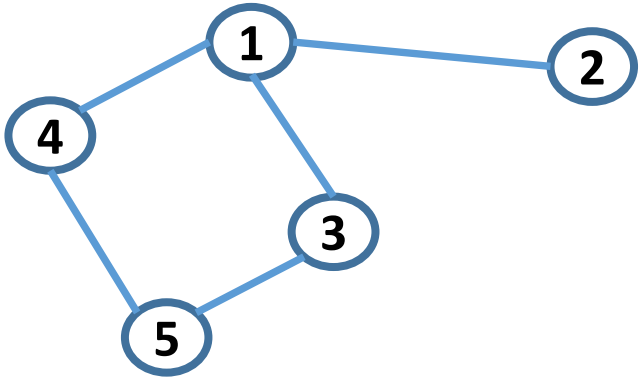
Loop until stack is empty

- pop() a vertex from top of Stack (it is 5)
- 5 has not been visited
- mark the 5 as visited (insert 5 into visited)
- process 5 (can be as simple as print it out)
- push all its adjacent vertices onto stack (which are 3 and 4)

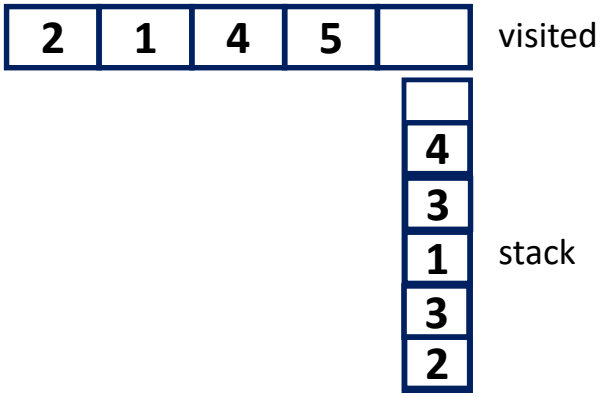
repeat till stack is empty

Before 4th loop





At end of 4th loop



Depth First Search

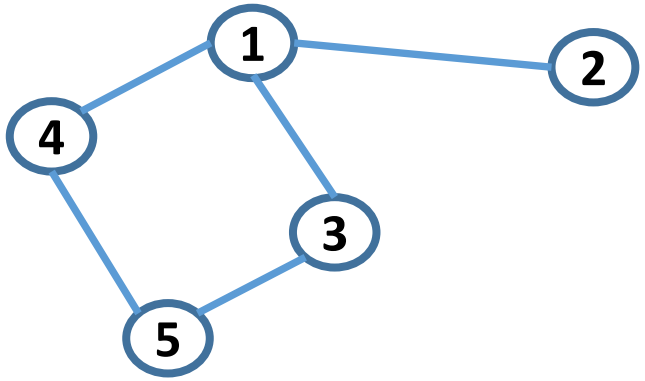
Loop until Stack is empty

- pop() a vertex from Stack (It is 4. It has been visited before. Skip. Go to next loop)

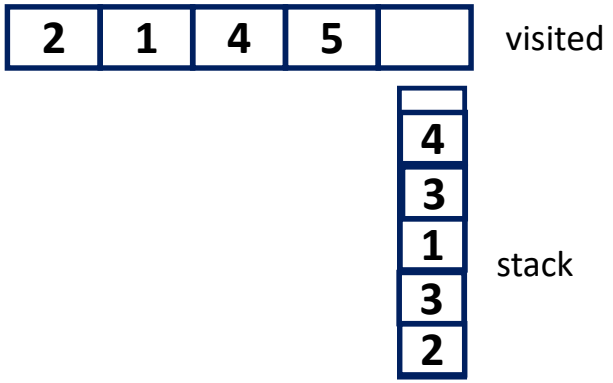
Loop until Stack is empty

- pop() a vertex from Stack (it is 3)
- 3 has not been visited
- mark 3 the as visited (insert 3 into visited)
- process the vertex
- push all its adjacent vertices onto the Stack (which are 1 and 5)

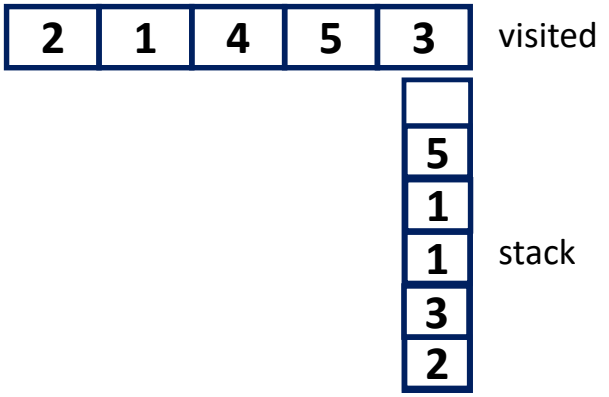
repeat till queue is empty



Before 5th loop



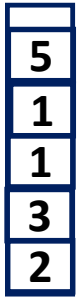
At end of 6th loop



Depth First Search

Before 6th loop

visited



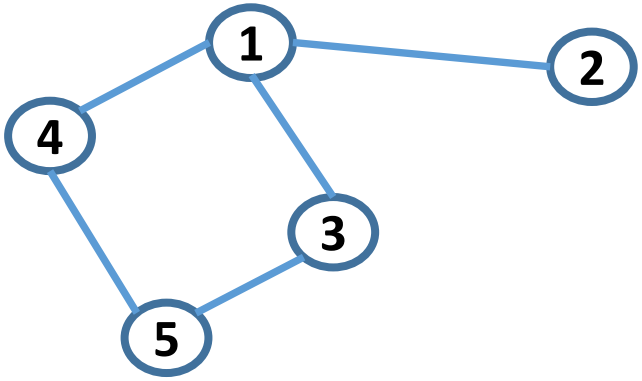
stack

Loop until queue is empty

- pop() a vertex from Stack (It is 5. It has been visited before. Skip. Go to next loop)
- pop() a vertex from Stack (It is 1. It has been visited before. Skip. Go to next loop)
- pop() a vertex from Stack (It is 1. It has been visited before. Skip. Go to next loop)
- pop() a vertex from Stack (It is 3. It has been visited before. Skip. Go to next loop)
- pop() a vertex from Stack (It is 2. It has been visited before. Skip. Go to next loop)

Stack is empty. Stop.

All vertices have been visited in the order 2, 1, 4, 5 then 3.



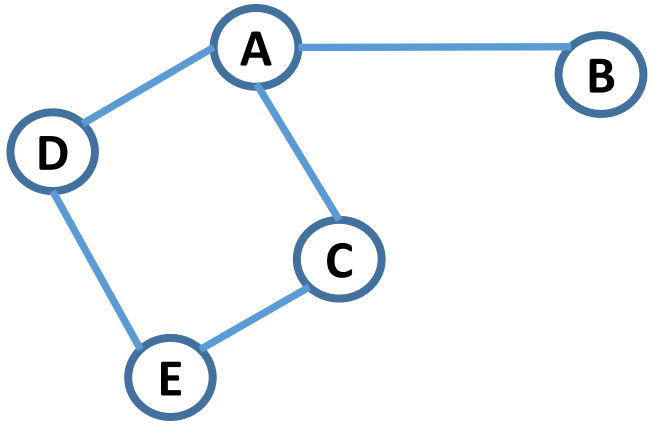
At end of 10th loop

visited

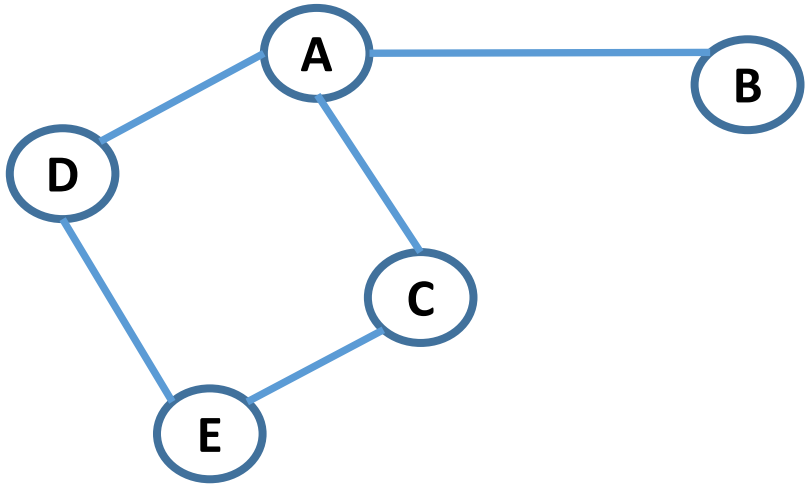
stack

Implementing Traversal in BFS

```
class Vertex{
    String label;
    ArrayList<Vertex> adjList = new ArrayList<>();

    Vertex (String label) {
        this.label = label;
    }
}
```



Implementing Traversal in BFS

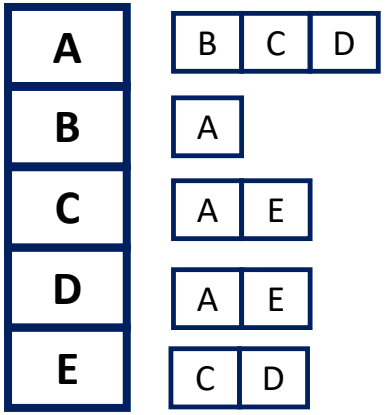


A	B C D
B	A
C	A E
D	A E
E	C D

Implementing Traversal in BFS

Representing vertices and edges for the graph:

```
Vertex a = new Vertex("A");
Vertex b = new Vertex("B");
Vertex c = new Vertex("C");
Vertex d = new Vertex("D");
Vertex e = new Vertex("E");

a.adjList.add(b);
a.adjList.add(c);  // what if d is added
a.adjList.add(d);  // before c?

b.adjList.add(a);

c.adjList.add(a);
c.adjList.add(e);

d.adjList.add(a);
d.adjList.add(e);

e.adjList.add(c);
e.adjList.add(d);
```



Implementing Traversal in BFS

Prepare visited ArrayList and Queue.
Start by putting the root vertex (say B) into Queue.

```
ArrayList<Vertex> visited = new ArrayList<>();
Queue<Vertex> q = new ArrayDeque<>();

// set root
q.add(b);
```

Implementing Traversal in BFS

Start the loop with the algorithm for BFT that we have covered.

```
while (!q.isEmpty()){
    Vertex theVertex = q.poll();
    if (!visited.contains(theVertex)){
       visited.add(theVertex);
       for (int i=0; i<theVertex.adjList.size(); i++){
          q.add(theVertex.adjList.get(i));
       }
    }
}

visited.forEach(z->System.out.print(z.label+" "));
```

Output:

B A C D E

Implementing Traversal in DFS

Instead of Queue, a Stack is used.
Start by putting the root vertex (say B) into Queue.

```
ArrayList<Vertex> visited = new ArrayList<>();
Stack<Vertex> s = new Stack<>();

// set root
s.push(b);
```

Implementing Traversal in DFS

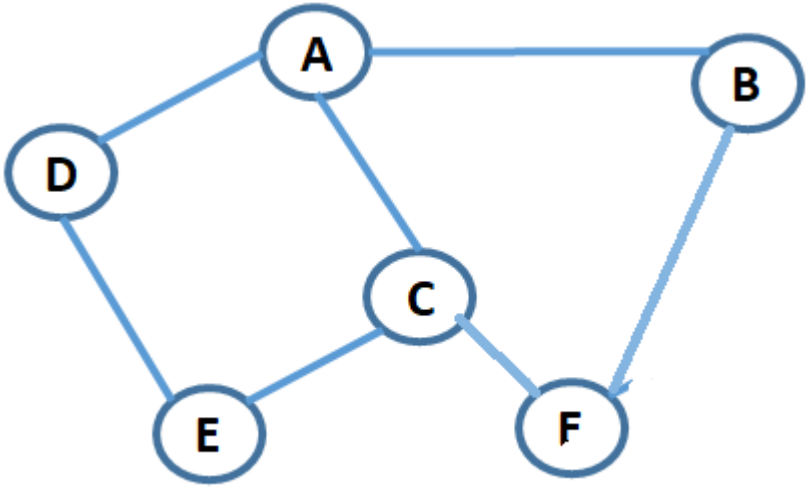
Similar to BFS:

```
while (!s.isEmpty()){
    Vertex theVertex = s.pop();
    if (!visited.contains(theVertex)){
       visited.add(theVertex);
       for (int i=0; i<theVertex.adjList.size(); i++){
          s.add(theVertex.adjList.get(i));
       }
    }
}

visited.forEach(z->System.out.print(z.label+" "));
```

Output:

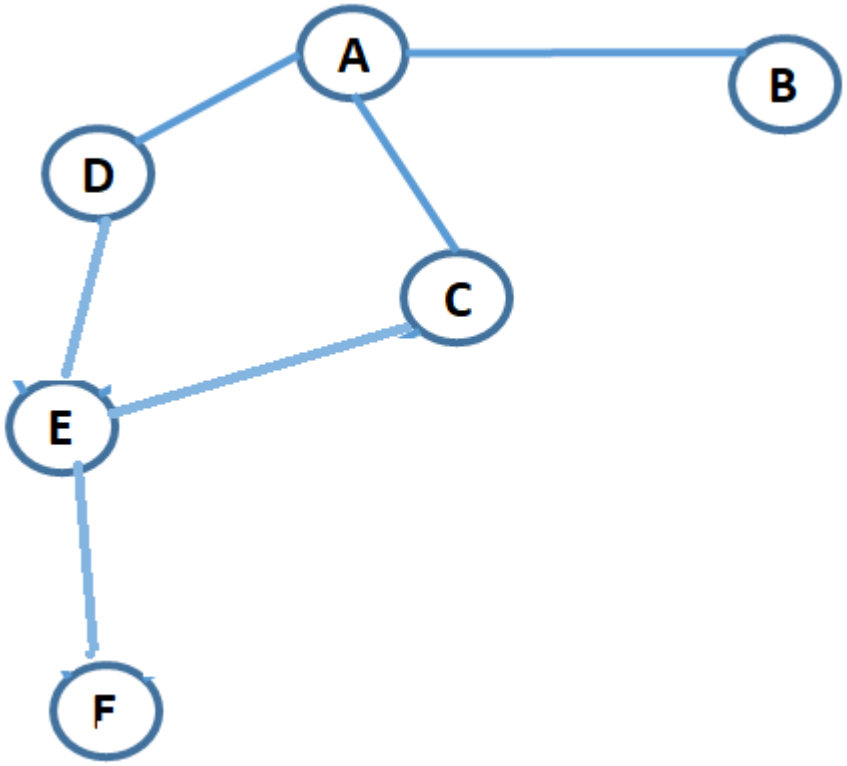
B A D E C

Try




```
Vertex a = new Vertex("A");
Vertex b = new Vertex("B");
Vertex c = new Vertex("C");
Vertex d = new Vertex("D");
Vertex e = new Vertex("E");
Vertex e = new Vertex(“F");

a.adjList.add(b);
a.adjList.add(c);
a.adjList.addf(d);

b.adjList.add(a);

c.adjList.add(a);
c.adjList.add(e);
c.adjList.add(f);

d.adjList.add(a);
d.adjList.add(e);

e.adjList.add(c);
e.adjList.add(d);

f.adjList.add(c);
f.adjList.add(b);
```

Try




```
Vertex a = new Vertex("A");
Vertex b = new Vertex("B");
Vertex c = new Vertex("C");
Vertex d = new Vertex("D");
Vertex e = new Vertex("E");
Vertex e = new Vertex(“F");

a.adjList.add(b);
a.adjList.add(c);
a.adjList.addf(d);

b.adjList.add(a);

c.adjList.add(a);
c.adjList.add(e);

d.adjList.add(a);
d.adjList.add(e);

e.adjList.add(c);
e.adjList.add(d);
e.adjList.add(f);

f.adjList.add(e);
```