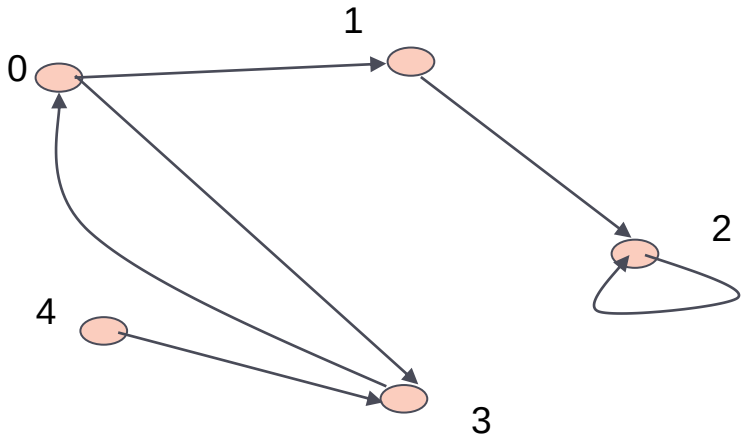
Graph Traversal Algorithm (BFS Algorithm)

Presented by Nabanita Das

Definition of Graphs

- A graph is a finite set of nodes or vertices with edges between nodes
- Formally, a graph G is a structure (V,E) consisting of
  - a finite set V called the set of vertices, and
  - a set E that is a subset of VxV. That is, E is a set of pairs or edges of the form (x,y) where x and y are vertices in V
- V={0,1,2,3,4}
- E={(0,1), (1,2), (0,3), (3,0), (2,2), (4,3)}



When (x,y) is an edge,
we say that x is adjacent to y, and y is adjacent from x.

0 is adjacent to 1.
1 is not adjacent to 0.
2 is adjacent from 1.

Graph Representation

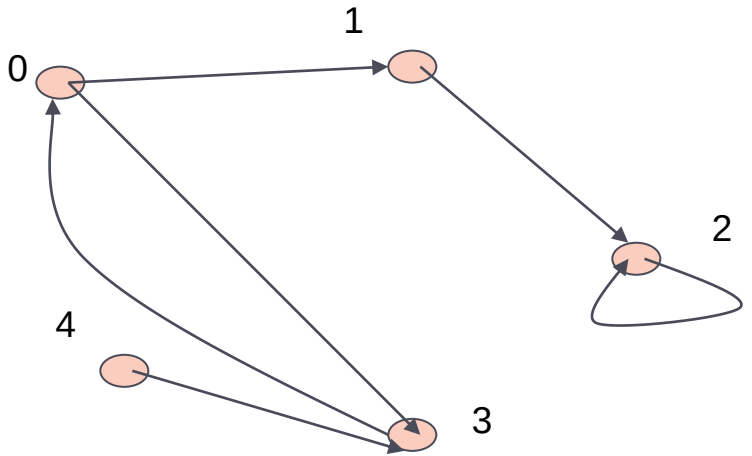
- For graphs to be computationally useful, they have to be conveniently represented in programs
- There are two computer representations of graphs:
  - **Adjacency matrix representation**
  - **Adjacency lists representation**

Adjacency Matrix Representation

- In this representation, each graph of n nodes is represented by an n x n matrix A, that is, a two-dimensional array A
- The nodes are (re)-labeled 1,2,…,n
- A[i][j] = 1 if (i,j) is an edge
- A[i][j] = 0 if (i,j) is not an edge

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \end{bmatrix}$$

5 x 5

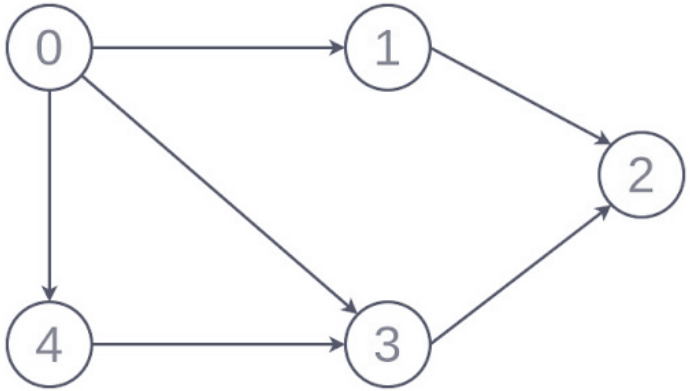


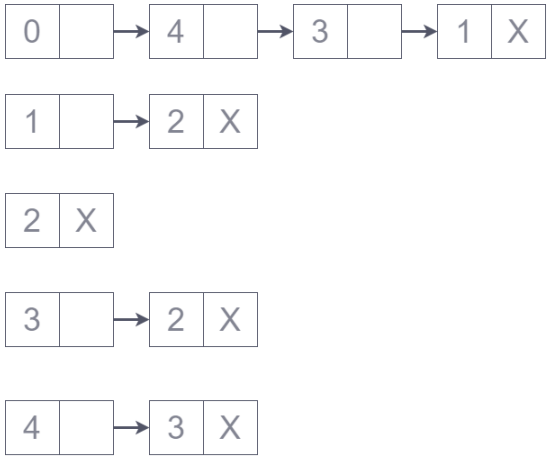
Pros and Cons of Adjacency Matrices

- Pros:
  - Simple to implement
  - Easy and fast to tell if a pair (i,j) is an edge: simply check if A[i][j] is 1 or 0
- Cons:
  - No matter how few edges the graph has, the matrix takes O(n^2) in memory.

Adjacency Lists Representation

- A graph of n nodes is represented by a one-dimensional array L of linked lists, where
  - L[i] is the linked list containing all the nodes adjacent from node i.
  - The nodes in the list L[i] are in no particular order.



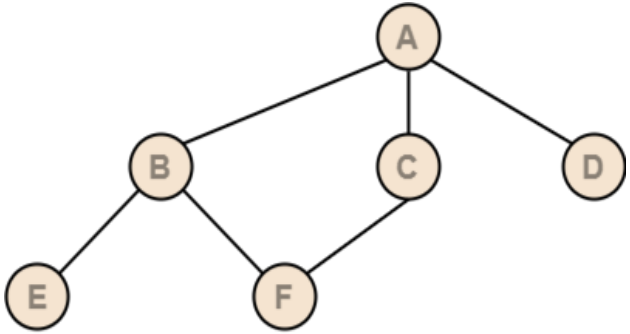


Pros and Cons of Adjacency Lists

- Pros:
  - Saves on space (memory): the representation takes as many memory words as there are nodes and edge.
- Cons:
  - It can take up to O(n) time to determine if a pair of nodes (i,j) is an edge: one would have to search the linked list L[i], which takes time proportional to the length of L[i].
  - For undirected graph it can take up to O(n+2e) time to determine.

Breadth First Search

- Breadth First Search or BFS is a graph traversal algorithm.
- It is used for traversing or searching a graph in a systematic fashion.
- BFS uses a strategy that searches in the graph in breadth first manner whenever possible.
- Queue data structure is used in the implementation of breadth first search.



- The breadth first search traversal order of the above graph is-

A, B, C, D, E, F

Breadth First Search Algorithm

```
BFS (V,E,s)
for each vertex v in V – {s}
do
        color[v] ← WHITE
        d[v] ← ∞
        π[v] ← NIL
color[s] = GREY
d[s] ← 0
π[s] ← NIL
Q ← { }
ENQUEUE (Q,s)
While Q is non-empty
        do v ← DEQUEUE (Q)
        for each u adjacent to v
                do if color[u] ← WHITE
                        then color[u] ← GREY
                                d[u] ← d[v] + 1
                                π[u] ← v
                                ENQUEUE (Q,u)
        color[v] ← BLACK
```

Step-01

Create and maintain 3 variables for each vertex of the graph.
For any vertex ‘v’ of the graph, these 3 variables are-

1. color[v]-

This variable represents the color of the vertex v at the given point of time.
The possible values of this variable are- WHITE, GREY and BLACK.
WHITE color of the vertex signifies that it has not been discovered yet.
GREY color of the vertex signifies that it has been discovered and it is being processed.
BLACK color of the vertex signifies that it has been completely processed.

2. Π[v]-

This variable represents the predecessor of vertex ‘v’.

3. d[v]-

This variable represents the distance of vertex v from the source vertex.

Step-02

For each vertex v of the graph except source vertex, initialize the variables as-

color[v] = WHITE

$\pi[v] = NIL$

$d[v] = \infty$

For source vertex S, initialize the variables as-

color[S] = GREY

$\pi[S] = NIL$

$d[S] = 0$

Step-03

Now, enqueue source vertex S in queue Q and repeat the following procedure until the queue Q becomes empty-

1. Dequeue vertex v from queue Q.
2. For all the adjacent white vertices u of vertex v,

do-

color[u] = GREY

d[u] = d[v] + 1

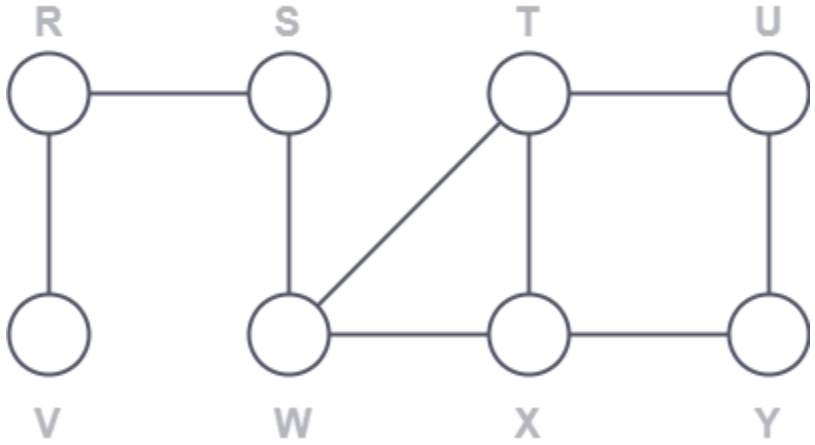
π[u] = v

Enqueue (Q,u)

3. Color vertex v to black.

Problem

Traverse the following graph using Breadth First Search Technique-



Consider vertex S as the starting vertex.

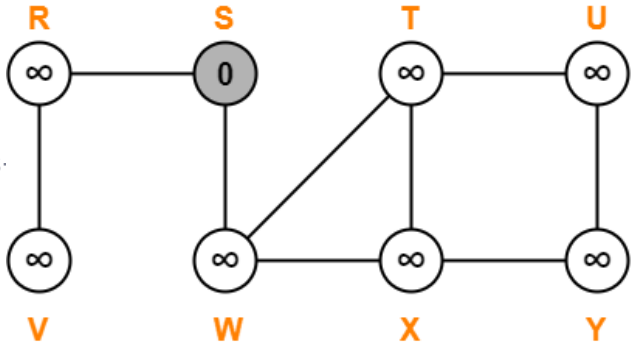
Step-01

For all the vertices v except source vertex S of the graph, we initialize the variables as-

- color[v] = WHITE
- π[v] = NIL
- d[v] = ∞

For source vertex S, we initialize the variables as-

- color[S] = GREY
- π[S] = NIL
- d[S] = 0



We enqueue the source vertex S in the queue Q.

BFS: S

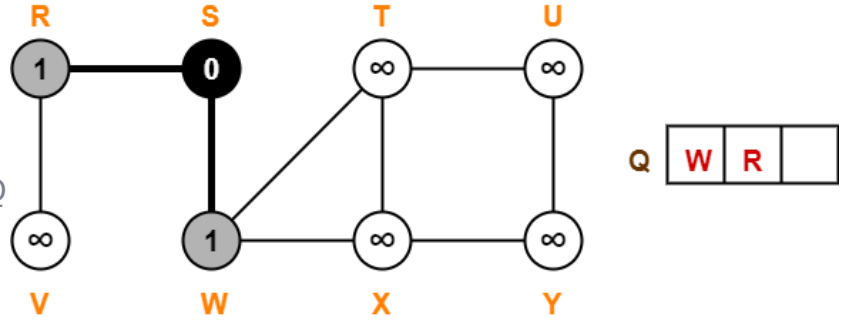
Step-02

Dequeue vertex S from the queue Q

For all adjacent white vertices ‘v’ (vertices R and W) of vertex S, we do-

1. color[v] = GREY
2. d[v] = d[S] + 1 = 0 + 1 = 1
3. π[v] = S
4. Enqueue all adjacent white vertices of S in queue Q

color[S] = BLACK



BFS: S

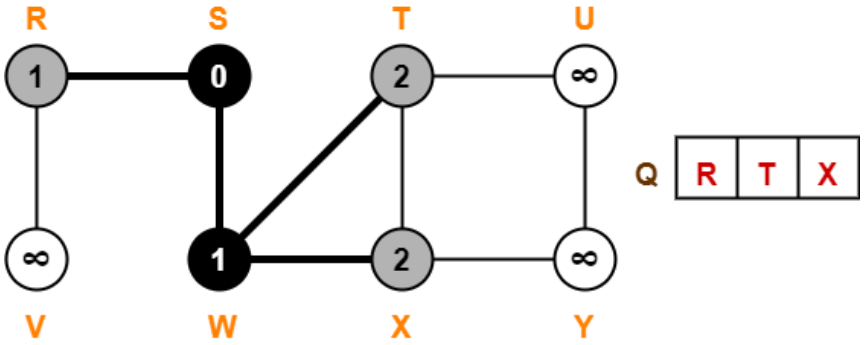
Step-03

Dequeue vertex W from the queue Q

For all adjacent white vertices ‘v’ (vertices T and X) of vertex W, we do-

1. color[v] = GREY
2. d[v] = d[W] + 1 = 1 + 1 = 2
3. π[v] = W
4. Enqueue all adjacent white vertices of W in queue Q

color[W] = BLACK



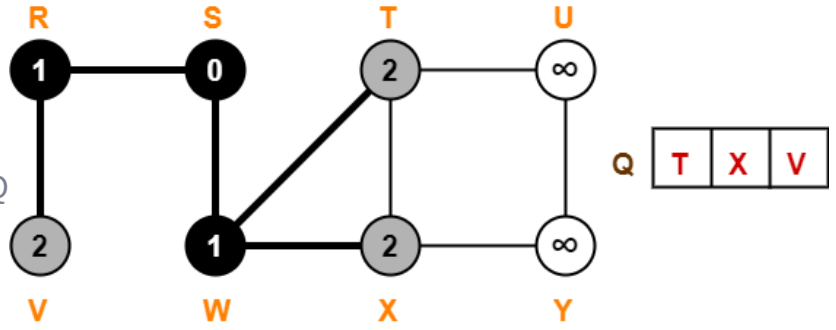
BFS: S W

Step-04

- Dequeue vertex R from the queue Q
- For all adjacent white vertices ‘v’ (vertex V) of vertex R, we do-

1. color[v] = GREY
2. d[v] = d[R] + 1 = 1 + 1 = 2
3. π[v] = R
4. Enqueue all adjacent white vertices of R in queue Q

- color[R] = BLACK



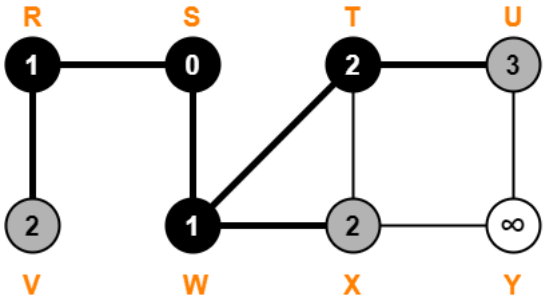
BFS: S W R

Step-05

- Dequeue vertex T from the queue Q
- For all adjacent white vertices ‘v’ (vertex U) of vertex T, we do-

1. color[v] = GREY
2. d[v] = d[T] + 1 = 2 + 1 = 3
3. π[v] = T
4. Enqueue all adjacent white vertices of T in queue Q

- color[T] = BLACK



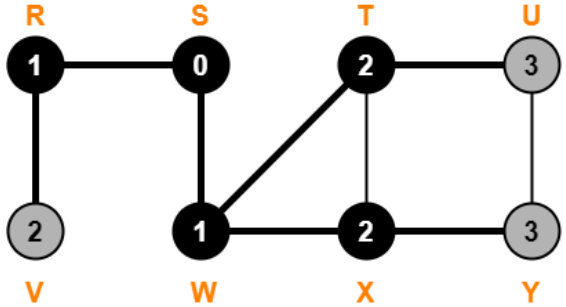
BFS: S W R T

Step-06

- Dequeue vertex X from the queue Q
- For all adjacent white vertices ‘v’ (vertex Y) of vertex X, we do-

1. color[v] = GREY
2. d[v] = d[X] + 1 = 2 + 1 = 3
3. π[v] = X
4. Enqueue all adjacent white vertices of X in queue Q

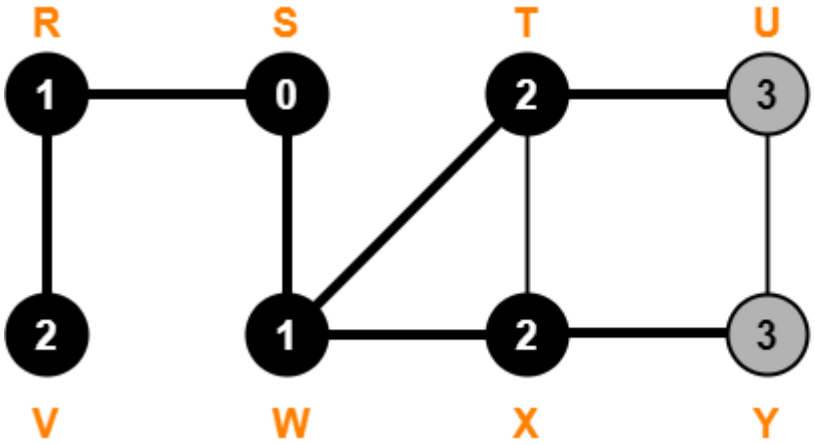
- color[X] = BLACK



BFS: S W R T X

Step-07

- Dequeue vertex V from the queue Q
- There are no adjacent white vertices to vertex V.
- color[V] = BLACK

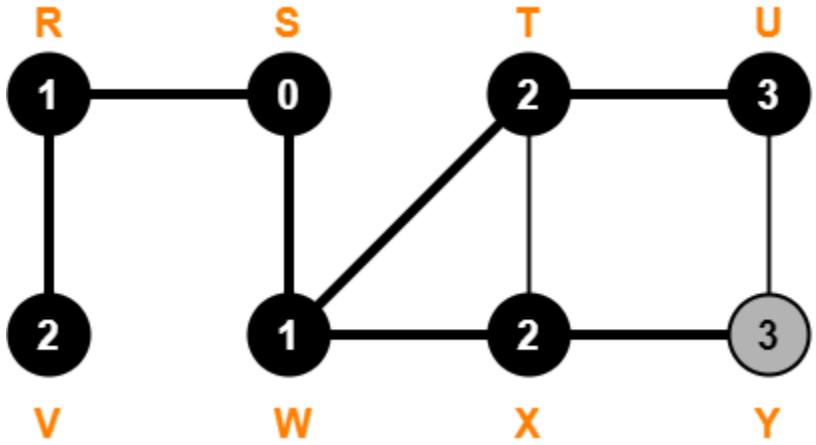


Q | U | Y |

BFS: S W R T X V

Step-08

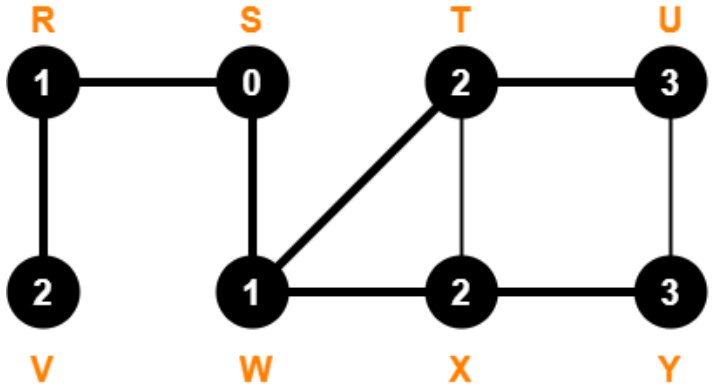
- Dequeue vertex U from the queue Q
- There are no adjacent white vertices to vertex U.
- color[U] = BLACK



BFS: S W R T X V U

Step-09

- Dequeue vertex Y from the queue Q
- There are no adjacent white vertices to vertex Y.
- color[Y] = BLACK



BFS: S W R T X V U Y

- Since, all the vertices have turned black and the queue has got empty, so we stop.

Complexcity of bfs algorithm

- Each node Enqueues once & Dequeues once to the queue.
- Enqueue, dequeue operations take O(1).
- That’s why for V vertices it is O(V).
- We pass through each edge only once.
- For every node we go through all its adjacent neighbours. It takes O(E).
- So, all together the runtime is O(V+E).

THANKS!

Any questions?