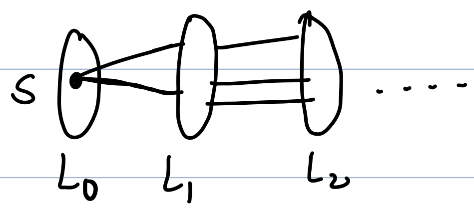
Breadth-first Search

Overview

- Explore nodes in layers



- Used for Computing Shortest Paths
- Used to find Connected Components of a graph G.

We assume that the input graph $G = (V, E)$ is represented using adjacency lists.

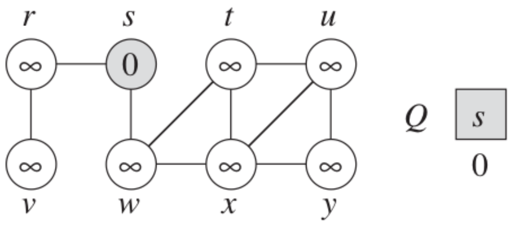
Notation:

- $u.color$: Color stored for the vertex u
- $u.\pi$: Predecessor of u. If u has no predecessor then $u.\pi = NIL$.
- $u.d$: distance of u from the source s computed by the algorithm.

The algorithm uses Queue Q (first-in, first out).

[Source: Coreman's book]

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

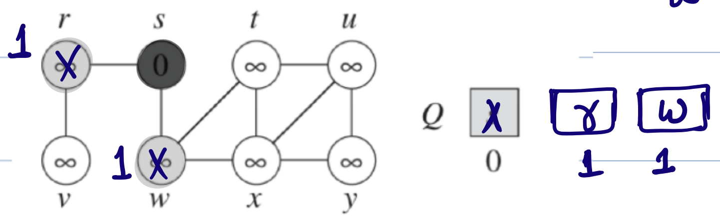


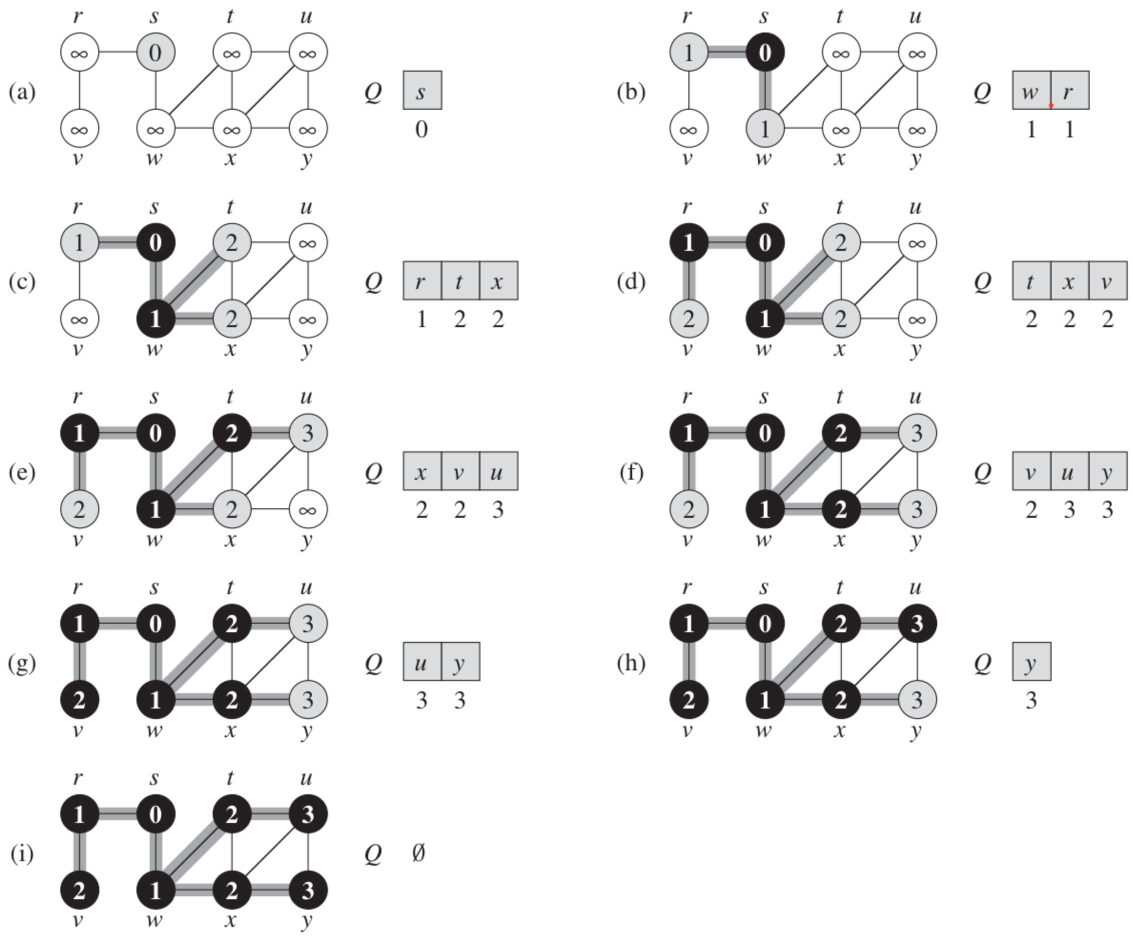
$s.\pi = NIL$

$u = s$

$r.\pi = s$

$w.\pi = s$





(a)
Q s
0
(b)
Q w r
1 1
(c)
Q r t x
1 2 2
(d)
Q t x v
2 2 2
(e)
Q x v u
2 2 3
(f)
Q v u y
2 3 3
(g)
Q u y
3 3
(h)
Q y
3
(i)
Q Ø

Runtime - Analysis:

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
```

$\Theta(V+E)$

```
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
```

$\Theta(1)$

```
while Q ≠ ∅
    u = DEQUEUE(Q)
```

$O(V)$

```
    for each v ∈ G.Adj[u]
```

$\sum_u \deg(u) = O(E)$

```
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

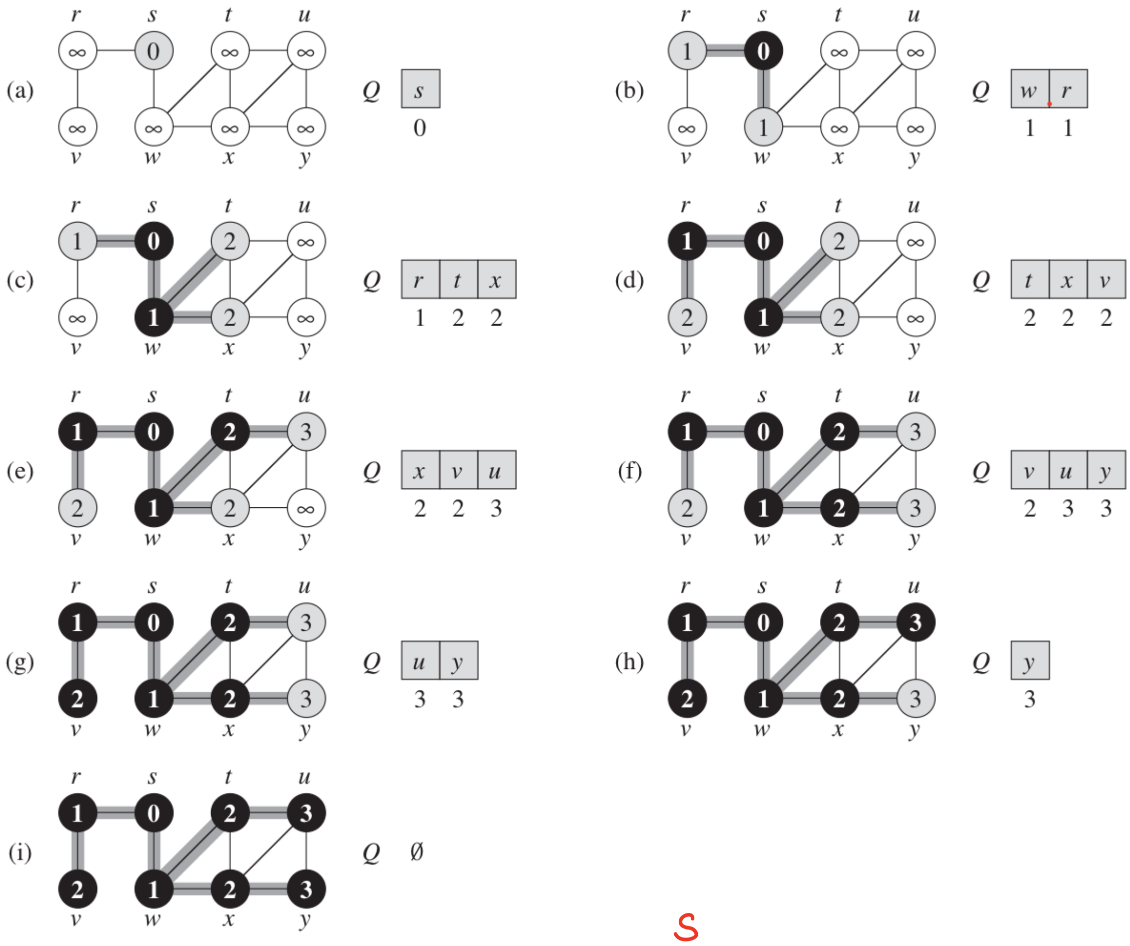
Breadth first Search trees

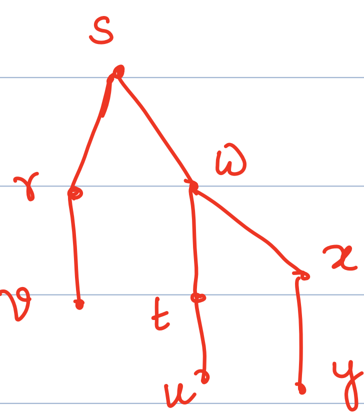
The Procedure BFS builds a BFS tree as it searches the graph.

let $G = (V, E)$ with source s, BFS tree of G is defined as $G_\pi = (V_\pi, E_\pi)$ (also called Predecessor Subgraph)

$$V_\pi = \{ v \in V : v.\pi \neq NIL \} \cup \{s\}$$

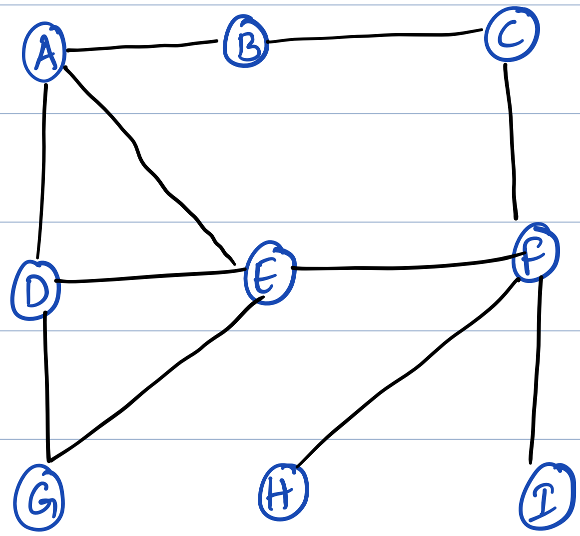
$$E_\pi = \{ (v.\pi, v) : v \in V_\pi - \{s\} \}$$





BFS tree

Exercise:



Perform a BFS from node A, with a Preterence for visiting lower-character veetices before higher-character veetices.

Applications of BFS

BFS can be used to solve many Problems in graph algorithms. For example

- Finding Shortest Path between two vertices [Coreman]
- Testing bipartiteness of a graph [KT : Chapter 3.4]
- Finding the number of connected components [KT : Chap 3.2]

etc ...

Shortest Paths using BFS

Notation:

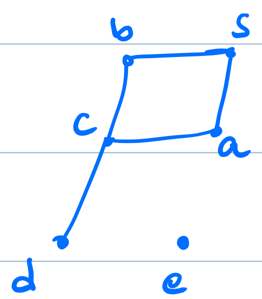
$G=(V,E)$, $s \in V$

$\delta(s,v)$: denotes the shortest path distance from s to v.

if there is no path from s to v then $\delta(s,v)=\infty$

Any path of length $\delta(s,v)$ from s to v is called a shortest path.

Eg



$\delta(s,d)=3$

$\delta(s,e)=\infty$

P: s b c d is a shortest s to d path.

Theorem : (Correctness of BFS)

Let $G=(V,E)$ be a directed | undirected graph and Suppose BFS is run on G from a given source $s \in V$. Then during the execution, BFS discovers every vertex $v \in V$ that is reachable from s and upon termination $v.d = \delta(s,v)$ for all $v \in V$.

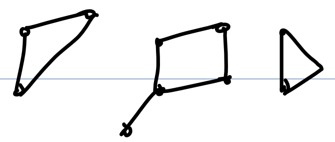
In otherwords, BFS(G,s) computes the length of the shortest path from s to every other vertex of G.

[For Proof | correctness details refer to Coreman book]

Finding the number of Connected Components

Aim: Compute all Connected Components of a graph G.

Ex:



Graph G.

G has 3 $\underbrace{\text{Connected Components}}_{\text{Pieces of } G}$.

Def: Two Vertices (u and v) are in Same Connected Component iff there is a Path from u to v.

[It is an equivalence relation, ie, $u \sim v \iff$ there is a Path from u to v]

Pseudo Code : Nodes are labelled 1 to n

- Components = 1
- All Nodes unexplored

- For $i = 1$ to n

  - if i not explored

    - BFS (G, i) → finds one Connected Component.

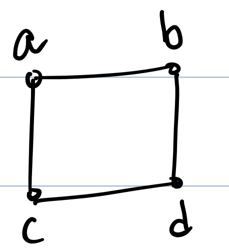
    - Components = Components + 1

Running time: $O(n+m)$ → $\sum_i$ Run time of BFS(G, i)

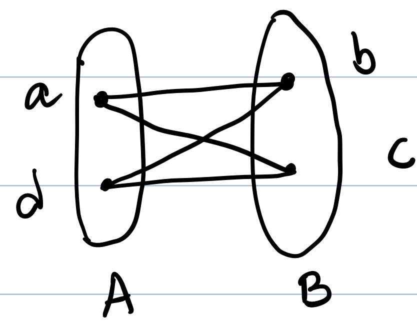
Bipartite Graphs

③ **Bipartite Graph :** A Graph is bipartite if $V(G)$ admits a partition into two sets such that every edge has its ends in different sets.

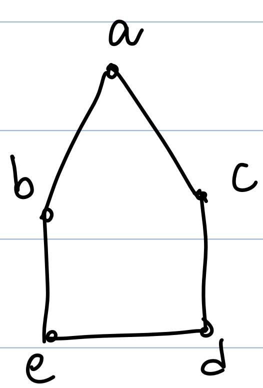
Ex. ①



Cycle C_4



Ex ②



C_5

C_5 is not bipartite.

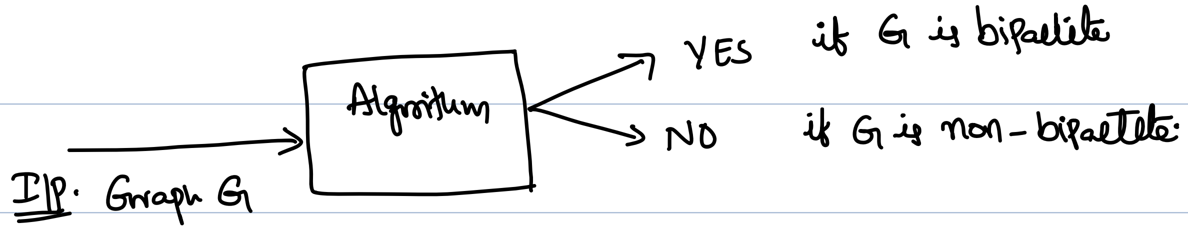
Theorem: A graph G bipartite iff G has no odd length cycles.

[For proof refer to any standard graph theory book]

Testing Bipartiteness : An application of BFS

Input : An undirected graph G

Question : Is G Bipartite ?

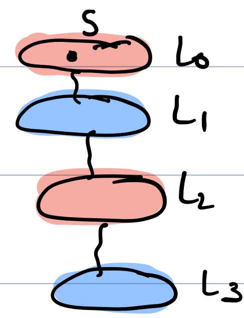


Simple Algorithm: s: Starting node

We Perform BFS, coloring s red, all of layer L_1 blue, all of layer L_2 red and so on.

Odd numbered layers colored blue

even numbered layers color red



We can implement this by adding an extra array Color over the nodes.

Whenever we get to a step in BFS where we are adding a node v to a list $L[i+1]$, we assign Color[v] = red if $i+1$ is an even number

= blue if " " " odd " .

At the end for every edge $uv \in E(G)$ Check whether both end received the different Color or not.

Total running time $= O(n+m)$

Correctness: [KT - Chapter 3]

Lemma:- Let G be a connected graph and let $L_1, L_2, \ldots$ be the layers produced by BFS starting at node s. Then exactly one of the following two things must hold.

ⓐ There is no edge G joining two nodes of the same layer. In this case G is bipartite in which the nodes in even-numbered layers can be colored red and the nodes in odd numbered layers can be colored blue.

ⓑ There is an edge of G joining two nodes of the same layer. In this case G contains an odd length cycle and so it cannot be bipartite.