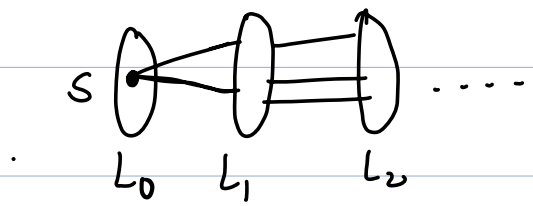
Breadth-first Search

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.

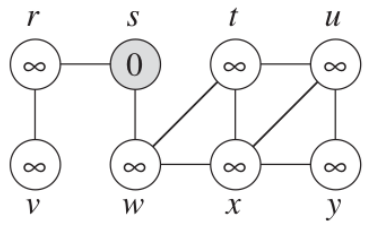
Notation:

- $u.color$: 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)
 1  for each vertex u ∈ G.V − {s}
 2       u.color = WHITE
 3       u.d = ∞
 4       u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       u = DEQUEUE(Q)
12       for each v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```



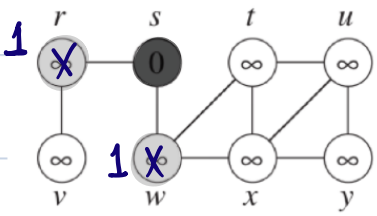
Q [s]
0

$s.\pi = NIL$

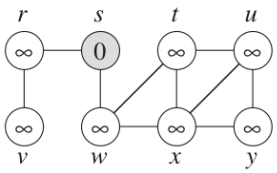
$u = s$

$r.\pi = s$

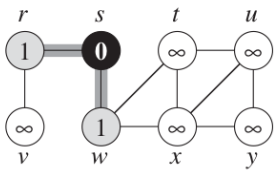
$w.\pi = s$



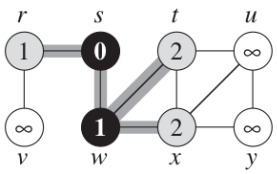
Q [X] [r] [w]
0 1 1

(a) 

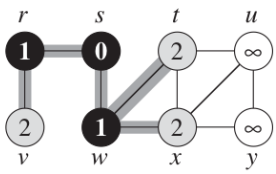
Q | s | 0

(b) 

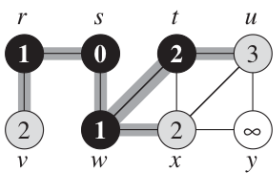
Q | w | r | 1 1

(c) 

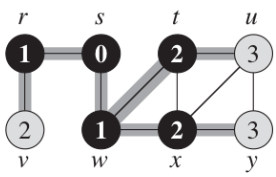
Q | r | t | x | 1 2 2

(d) 

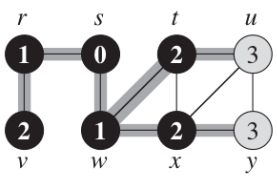
Q | t | x | v | 2 2 2

(e) 

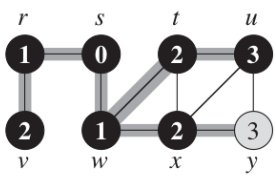
Q | x | v | u | 2 2 3

(f) 

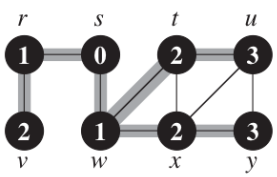
Q | v | u | y | 2 3 3

(g) 

Q | u | y | 3 3

(h) 

Q | y | 3

(i) 

Q $\emptyset$

Runtime - Analysis:

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

$\Theta(V+E)$

$\Theta(1)$

$O(V)$

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

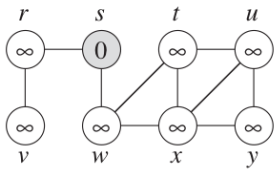
Breadth first Search trees

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

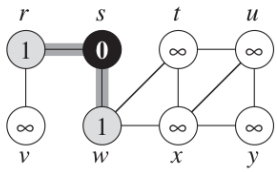
let $G = (V, E)$ with source s, BFS tree of G is defined as $G_\pi = (V_\pi, E_\pi)$ (also called Predecessor subgraph)

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

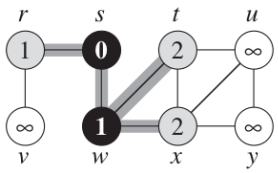
$$E_\pi = \{ (v.\pi, v) : v \in V_\pi - \{s\} \}$$

(a) 

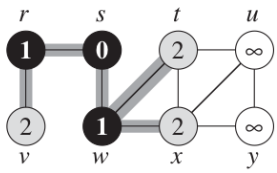
Q | s | | |
0

(b) 

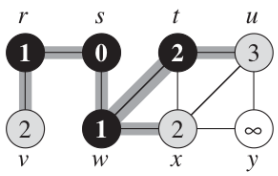
Q | w | r | |
1 1

(c) 

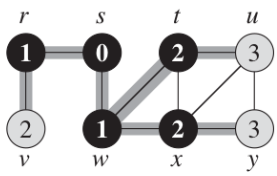
Q | r | t | x |
1 2 2

(d) 

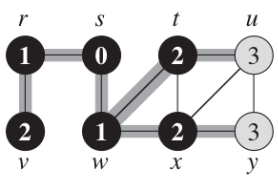
Q | t | x | v |
2 2 2

(e) 

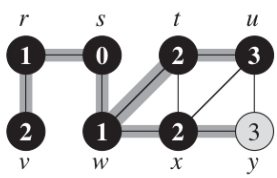
Q | x | v | u |
2 2 3

(f) 

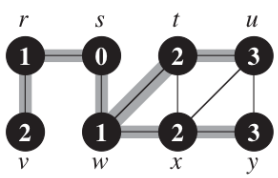
Q | v | u | y |
2 3 3

(g) 

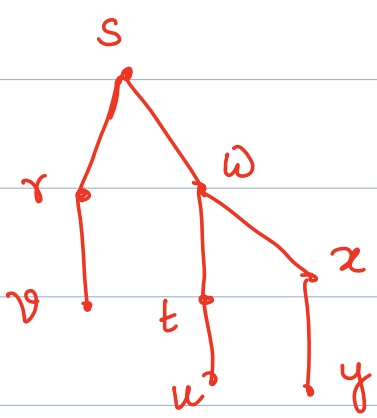
Q | u | y |
3 3

(h) 

Q | y |
3

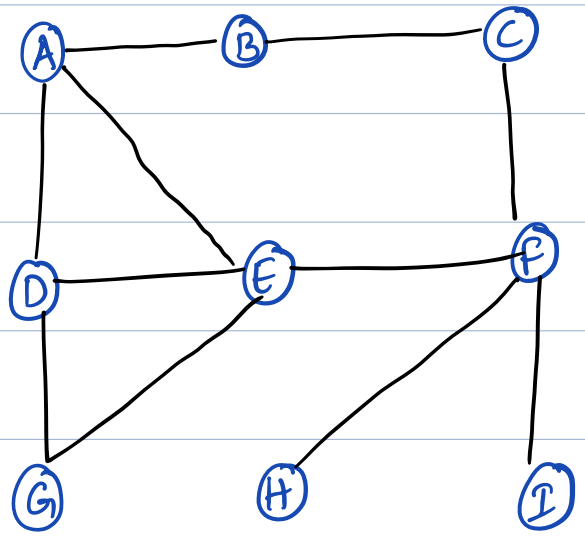
(i) 

Q $\emptyset$



BFs tree

Exercise:



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

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:

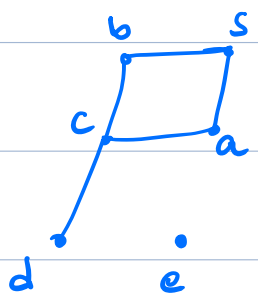
$G = (V, E)$, $s \in V$

$\delta(s, v)$: 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 $v \in V$.

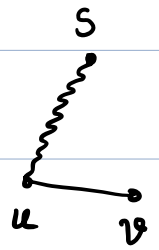
In otherwords, BFS(G,s) computes the length of the shortest path from s to every other vertex of G.

[For Proof | correctness details refer to Coreman book]

Lemma (1) Let $G = (V, E)$ be a directed / undirected graph, and let $s \in V$ be an arbitrary vertex.

Then, for any edge $uv \in E$

$$\delta(s, v) \leq \delta(s, u) + 1$$



Proof if there is a path from s to u

then, there is a path from s to v.

∴ The shortest path from s to v cannot be longer than the shortest path from s to u followed by the edge uv.

if there is no path from s to u in G

then $\delta(s, u) = \infty$, Also $\delta(s, v) = \infty$.

Goal is to show BFS computes $\delta(s,v)$ for each vertex $v \in V$.

Lemma:- Let $G=(V,E)$ be a undirected graph. and suppose BFS is run on G from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, $v.d \geq \delta(s,v)$

Proof:- Induction on the number of ENQUEUE operations.

Base case: It occurs at line 9 of BFS.

We have $s.d = 0 = \delta(s,s)$

$v.d = \infty \geq \delta(s,v)$ $\forall v \in V - \{s\}$

Inductive Step:

Consider a White vertex v that is discovered during search from a vertex u.

By induction hypothesis $u.d \geq \delta(s,u)$ — ③

From Lemma ① we have $\delta(s,v) \leq \delta(s,u) + 1$ — ①

and from Line 15 we have $v.d = u.d + 1$ — ②

using ①, ② & ③

$$v.d = u.d + 1$$
$$\geq \delta(s,u) + 1$$
$$\geq \delta(s,v)$$

Then vertex v is ENQUEUED and it is never enqueued again as its color changed to gray.

$\therefore$ $v.d$ never changes again and inductive hypothesis maintained.

Lemma (A): [Coreman 22.3]

Suppose that during the execution of BFS on a graph $G=(V,E)$, the queue Q contains the vertices $v_1 \ldots v_r$ where v_1 is the head of Q and v_r is the tail. Then $v_r.d \leq v_1.d+1$ and $v_i.d \leq v_{i+1}.d$ for $i=1,2,\ldots r-1$

Lemma (B): [Coreman 22.4]

Suppose that vertices v_i and v_j are enqueued during the execution of BFS and that v_i is enqueued before v_j. Then $v_i.d \leq v_j.d$ at the time that v_j is enqueued.

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 $v \in V$.

Proof:- Proof by contradiction

for some node x, $x.d \neq \delta(s,x)$

let v be the closest node from s such that $v.d \neq \delta(s,v)$, clearly $v \neq s$.

From the above lemma $v.d > \delta(s,v)$ —①

ie, v must be reachable from s,

if not then $\delta(s,v) = \infty \geq v.d$.

Consider a shortest path from s to v (there may be many such paths fix one)

Let uv be the last edge on that shortest path

then $\delta(s,v) = \delta(s,u) + 1$ — ②

Since u is closer to s than v, $\delta(s,u) = u.d$

From ① and ②

$$v.d > \delta(s,v) = \delta(s,u) + 1 = u.d + 1$$

$$v.d > u.d + 1 \quad — (A)$$

Now consider the time when BFS chooses to dequeue vertex u from queue Q.

At this time vertex v is either

(a) white (not yet discovered)

(b) gray (discovered but not explored)

(c) black (discovered & explored)

We shall show that in each of these cases, we get a contradiction to eqn (A).

(a) If v is white

then line 15 sets $v.d = u.d + 1$

Contradiction to (A)

(b) If v is black

then it was already remove from the queue

and from lemmaB we have $v.d \leq u.d$

again Contradiction to (A)

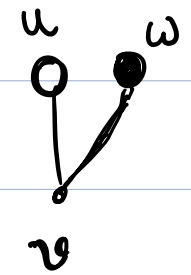
(c) If v is gray.

ie, v is changed to gray, when dequeuing

Some vertex w, which was removed from queue Q

earlier than u and $v.d = w.d + 1$

By lemmaB we have $w.d \leq u.d$



So we have

$v.d \leq u.d + 1$

again Contradiction to (A)

Thus we conclude that $v.d = \delta(s, v) \quad \forall v \in V$.

Applications of BFS

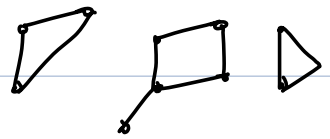
① Connected Components [KT: Chap 3.2]

② Bipartite graphs (detection) [KT: Chapter 3.4]

Finding the number of Connected Components

Aim: Compute all Connected Components of a graph G.

Ex:



Graph G.

G has 3 Connected Components.
↓
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

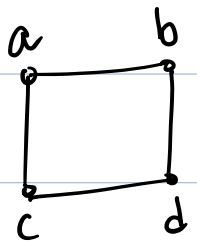
- Components = 1
- All Nodes unexplored
- For $i = 1$ to n
  - if i not explored
    - BFS (G, i) → finds one connected component.
    - Components = Components + 1

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

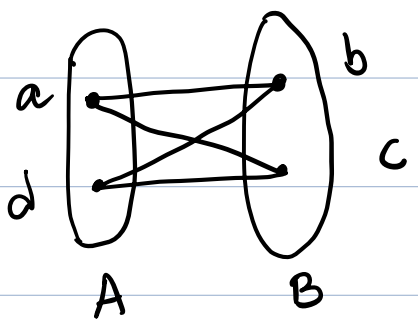
Bipartite Graphs

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.

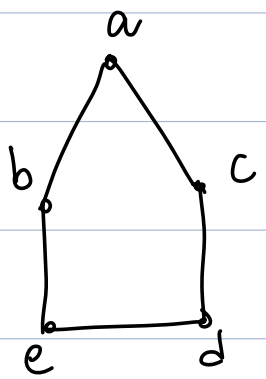
Ex. ①



Cycle C_4



Ex ②



C_5

C_5 is not bipartite.

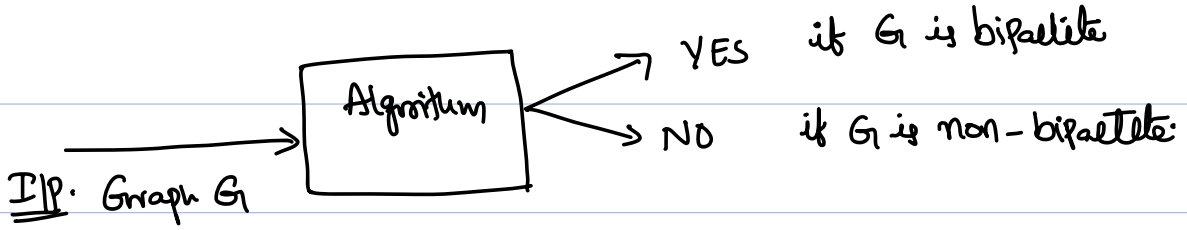
Theorem: A graph G bipartite iff G has no odd length cycles.

[For proof refer to any standard graph theory book]

Testing Bipartiteness : An application of BFS

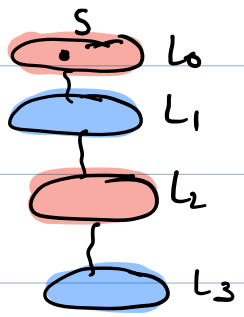
Input : An undirected graph G

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.

(a) There is no edge of 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.

(b) 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.