Algorithms: Lecture 10

Chalmers University of Technology

Today's Topics

Basic Definitions

Path, Cycle, Tree, Connectivity, etc.

Graph Traversal

Depth First Search

Breadth First Search

Testing Bipartatiness (One Graph Two colors)

Cycles

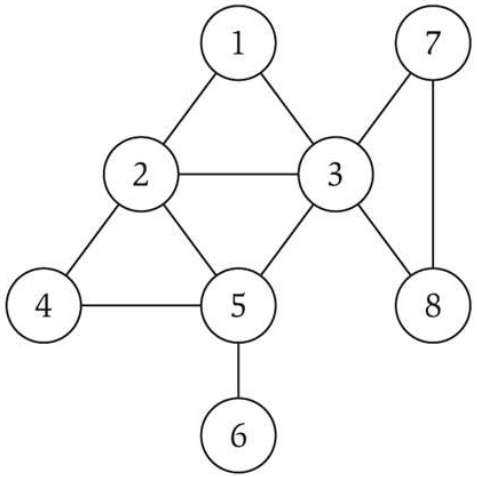
Def. A path is a sequence of $v_1, v_2, \ldots, v_{k-1}, v_k$ nodes with the property that each consecutive pair v_i, v_{i+1} is joined by an edge in G.

Def. A cycle is a path $v_1, v_2, \ldots, v_{k-1}, v_k$ in which $v_1 = v_k$, $k > 2$, and the first k-1 nodes are all distinct.

Directed G: directed path, cycle

Every pair (v_i, v_{i+1}) in the path or cycle is joined by a directed edge

- must respect the directionality of edges.

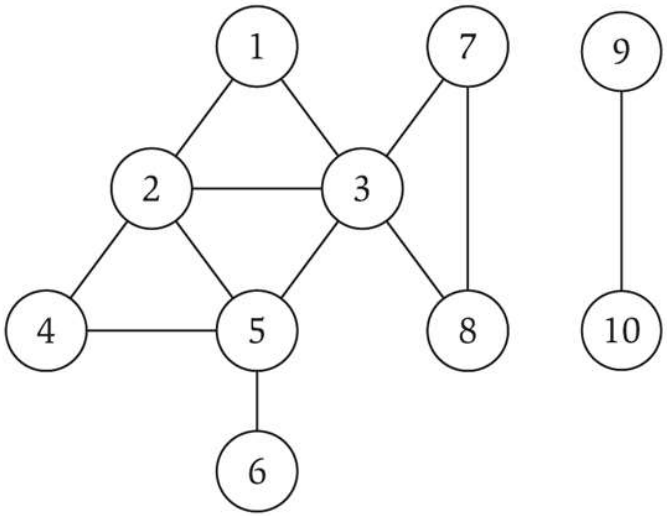


cycle C = 1-2-4-5-3-1

Not a cycle: 1-3-8-7-3-1

Connected Graph

Def. An undirected graph is *connected* if, for every pair of nodes u and v, there is a path from u to v.

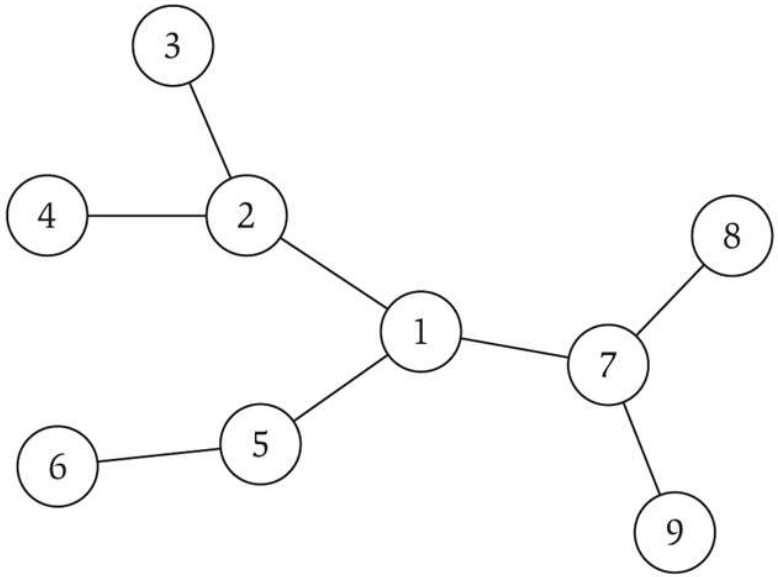


Trees

Def. An undirected graph is a tree if it is connected and does not contain a cycle.

Theorem. Let G be an undirected graph on n nodes. Any two of the following statements imply the third.

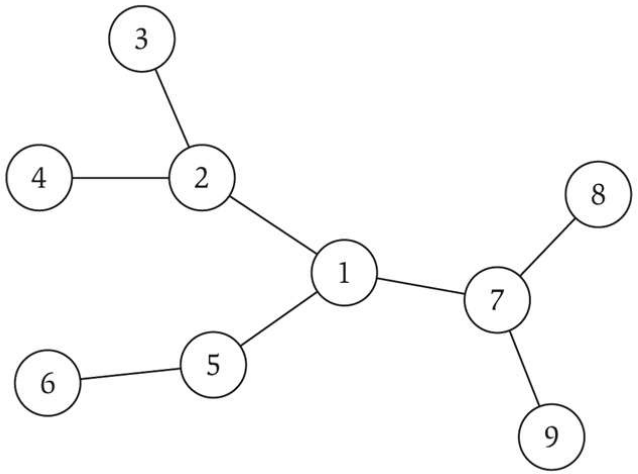
- G is connected.
- G does not contain a cycle.
- G has n-1 edges.



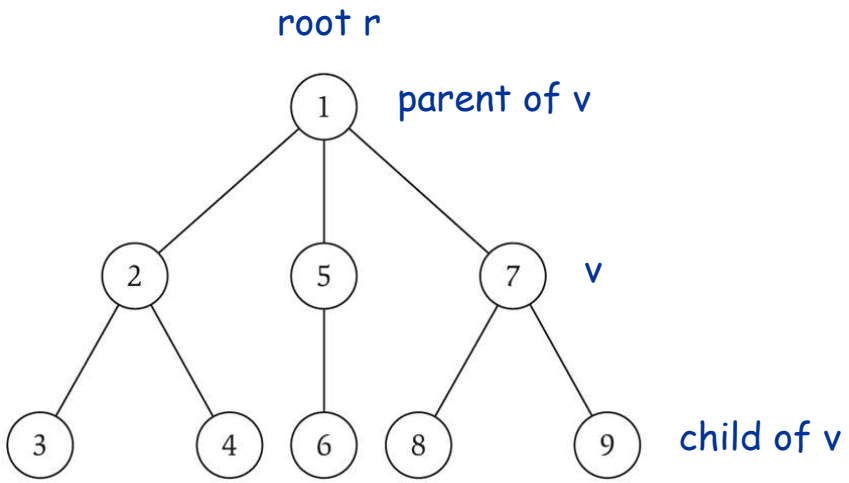
Rooted Trees

Rooted tree. Given a tree T, choose a root node r and orient each edge away from r.

Importance. Models hierarchical structure.



a tree

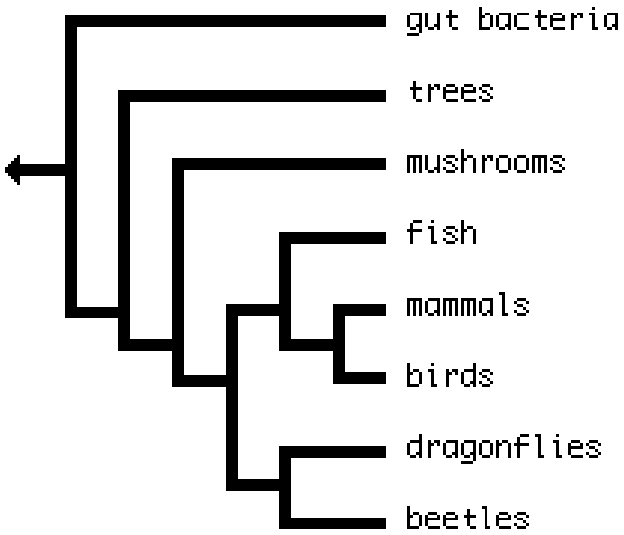


the same tree, rooted at 1

Phylogeny Trees

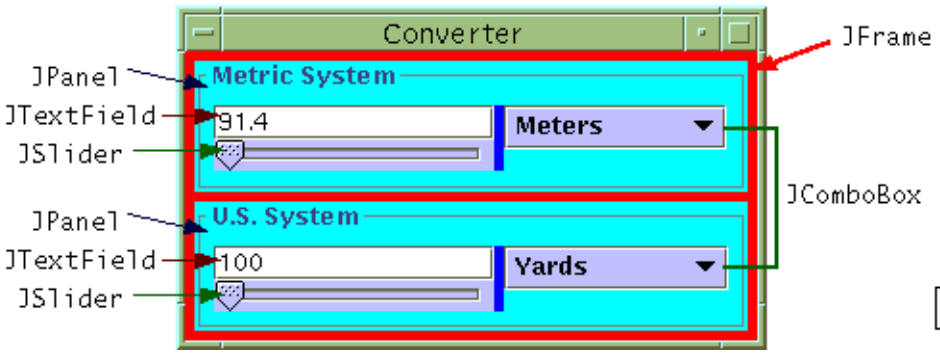
Phylogeny trees. Describe evolutionary history of species.

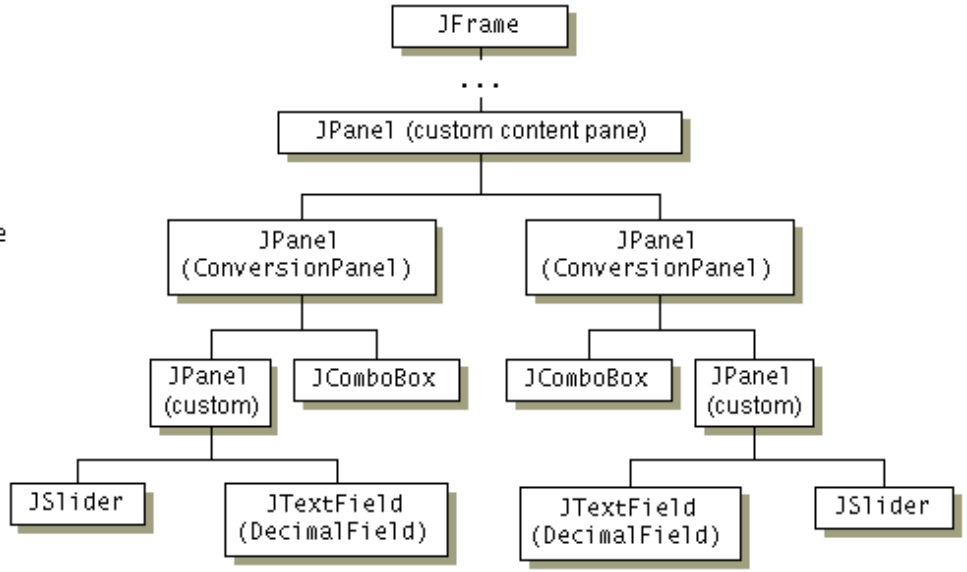
- biologists draw their tree from left to right



GUI Containment Hierarchy

GUI containment hierarchy. Describe organization of GUI widgets.





Reference: http://java.sun.com/docs/books/tutorial/uiswing/overview/anatomy.html

Graph Traversal

Connectivity

s-t connectivity problem. Given two node s and t, is there a path between s and t?

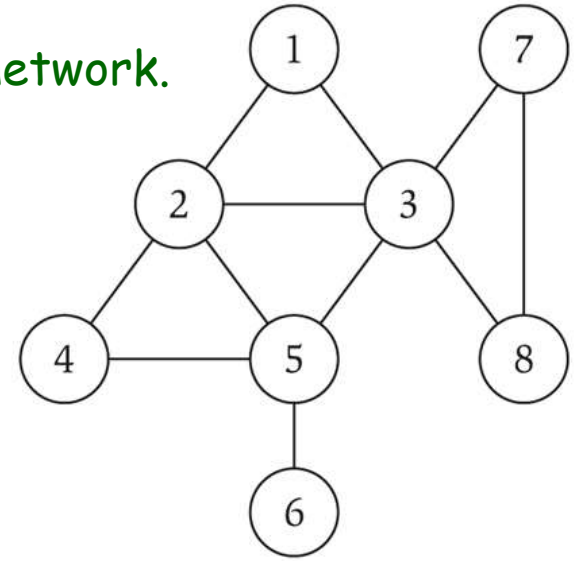
s-t shortest path problem. Given two node s and t, what is the length of the shortest path between s and t?

Applications.

- Social Network: deciding whether two persons are connected through a chain of friends.
- Fewest number of hops in a communication network.
- Maze traversal.

How to proceed to find 1-6 connectivity?

➢ Need a systematic exploration of the graph



Breadth First Search

Breadth First Search (BFS):

BFS(*s*) // Find a BFS tree rooted at *s*, which includes all nodes reachable from node *s*.

Create a Boolean array Discovered[1…*n*], Set Discovered[s] = true and Discovered[v] = false for all other v.

Create an empty FIFO queue Q, add node *s* to Q.

while Q is not empty

    dequeue a node *u* from Q

    for each node *v* adjacent to node *u*

        if Discovered[*v*] is false then

            add node *v* to Q, set Discovered[*v*] to true

        endif

    endfor

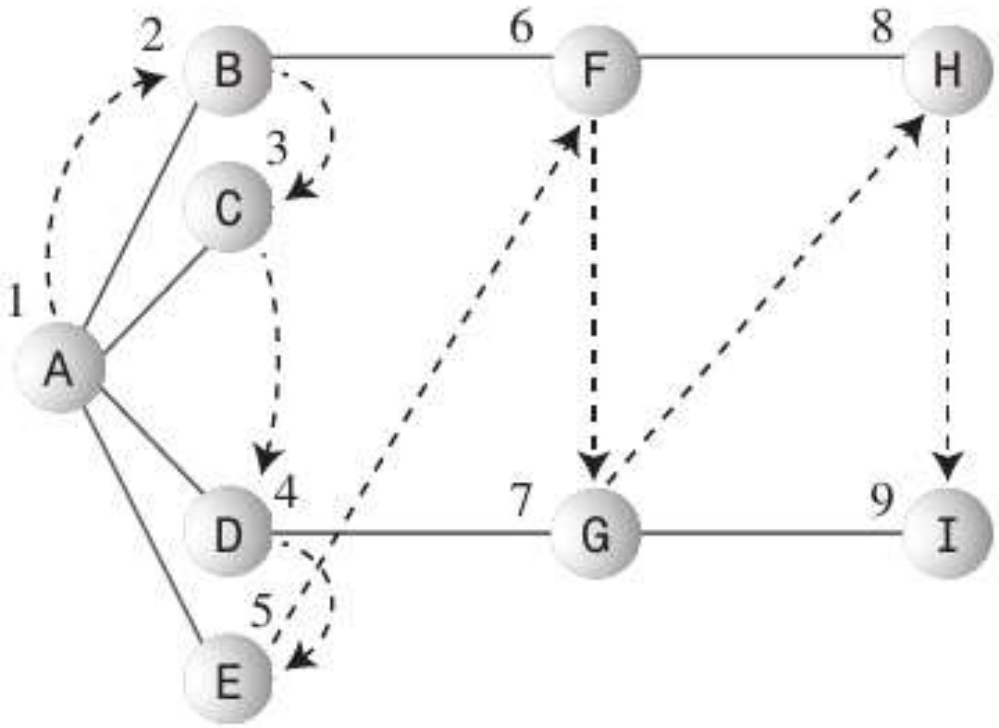
endwhile

Breadth First Search

BFS Tree:

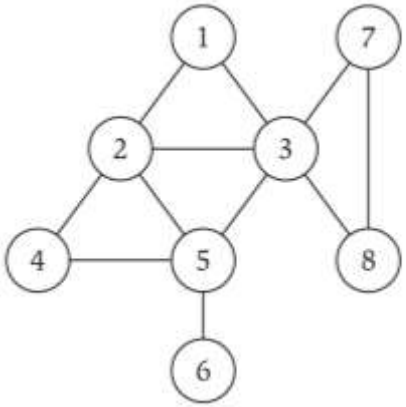
Create an empty tree T

Add edge (u, v) to the tree T, when v is discovered the first time



http://i.stack.imgur.com/TjhfH.png

Breadth First Search: Analysis



Analysis:

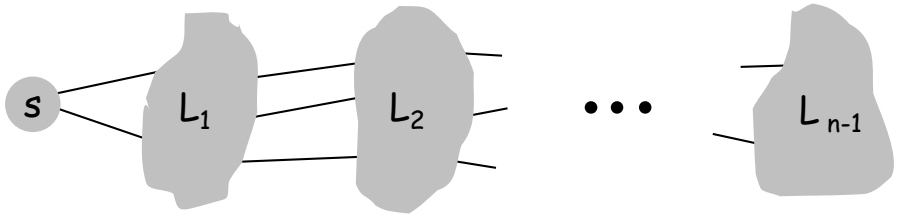
- A node *u* enters Q at most once, and the for loop needs nodes adjacent to every such u
- O(1) to process an edge
- Finding all *v* adjacent to *u*:
  - Adjacency Matrix:
    - ➢ we have to check all matrix entries in u's row: O(n)
    - ➢ total time required to process all rows of the Matrix: $O(n^2)$
  - Adjacency List:
    - ➢ when we consider node u, there are deg(u) incident edges (u, v)
    - ➢ total time processing all the edges is $\Sigma_{u \in V} \deg(u) = 2m \Rightarrow O(m)$
    - ➢ setup time for the array Discovered is $O(n), \Rightarrow O(m + n)$
    - ➢ m is at least n-1 for connected graph, m dominates $\Rightarrow O(m)$

each edge (u, v) is counted exactly twice in sum: once in deg(u) and once in deg(v)

Breadth First Search: Properties

BFS intuition. Explore outward from s in all possible directions, adding nodes one "layer" at a time.

BFS algorithm partitions the nodes into layers:



- $L_0 = \{ s \}$.
- L_1 = all neighbors of L_0.
- L_2 = all nodes that do not belong to L_0 or L_1, and that have an edge to a node in L_1.
- L_{i+1} = all nodes that do not belong to an earlier layer, and that have an edge to a node in L_i.

- Implementation using Queue processes the nodes exactly layer by layer
  - explores in order of distance from s.

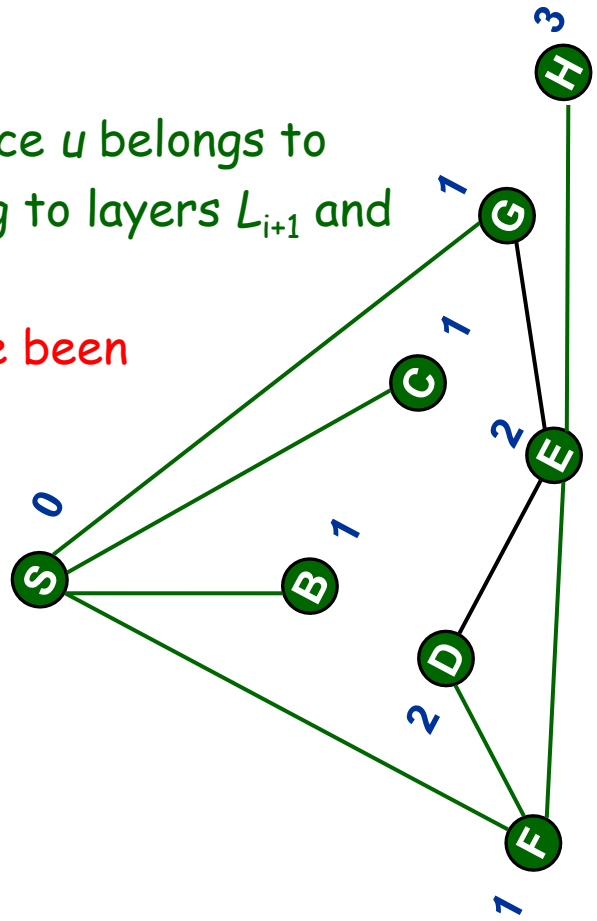
Breadth First Search: Properties

Property. Let T be a BFS tree of $G = (V, E)$, nodes u, v belong to T, and let (u, v) be an edge of G. Then the level of u and v differ by at most 1.

Let u, v belong to layers L_i and L_j respectively.

Suppose $i < j - 1$. (a contradiction)

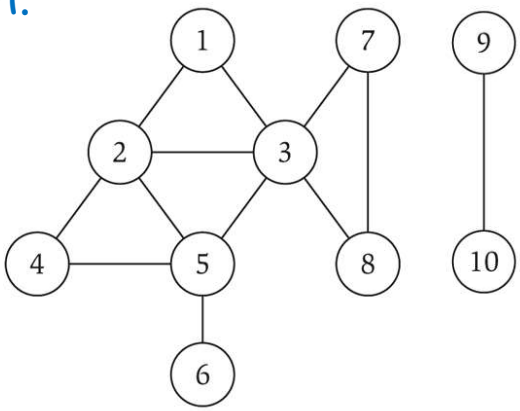
- When BFS examines the edges incident to *u*, since *u* belongs to layer L_i, the only nodes discovered from *u* belong to layers L_{i+1} and earlier;
- hence, if *v* is a neighbor of *u*, then it should have been discovered by this point at the latest, and
- should belong to layer L_{i+1} or earlier

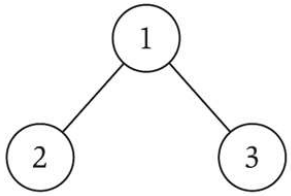


Breadth First Search: Properties

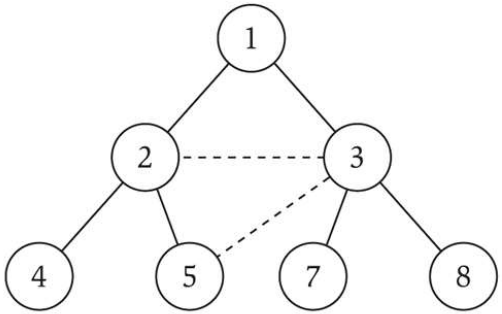
What follows:

- For each i, L_i consists of all nodes at distance exactly i from s.
- There is a path from s to t iff t appears in some layer.
- Moreover: s-t is a shortest path.

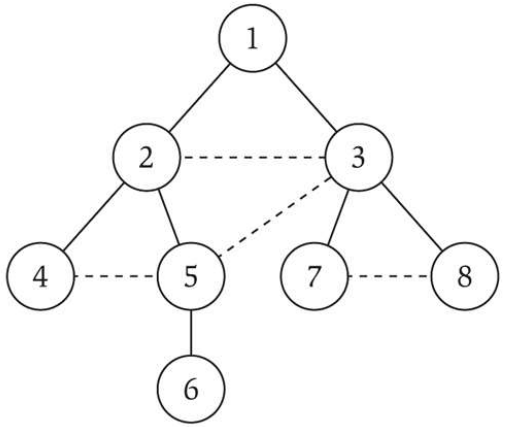




(a)



(b)



(c)

L_0

L_1

L_2

L_3

Depth First Search

Depth First Search (DFS):

Create a Boolean array Explored[1…*n*], initialized to false for all.

DFS(u)

```
set Explored[u] to true
for each node v adjacent to node u
        if Explored[v] is false then
                                DFS(v)
        endif
endfor
```

- Call DFS(*s*)
  - ➢ each recursive call is done only after termination of the previous call, this gives the desired depth first behavior.

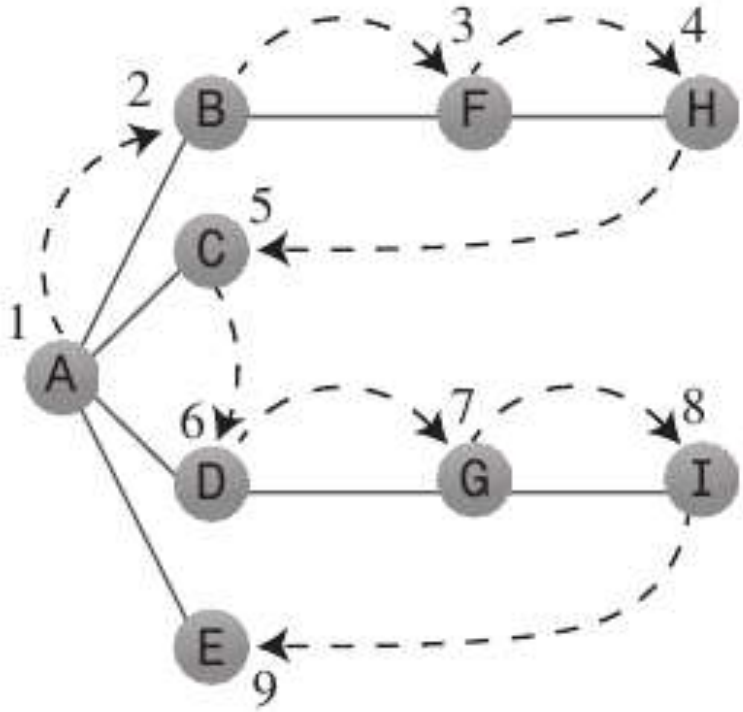
- In iterative implementation, maintain a stack explicitly.

Depth First Search

DFS tree:

Take an array parent, set parent[*v*]= *u* when calling DFS(v) due to edge *(u, v)*.

While setting u (*u* ≠ *s*) as Explored, add the edge *(u*,parent[*u*]) to the tree.



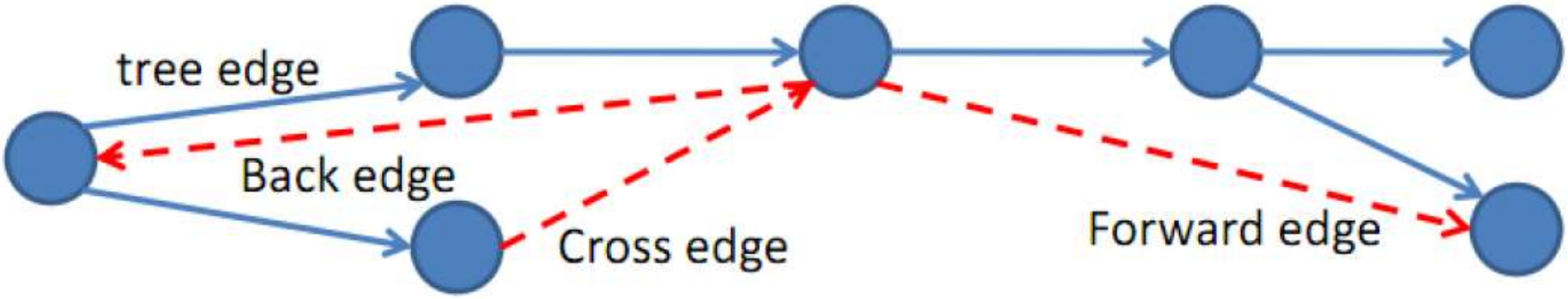
Reference: http://i.stack.imgur.com/gh0T1.png

Depth First Search: Edge Classification

Edge Classification:

As we execute DFS, an edge *(u, v)* can be classified into four edge types.

- IF v is visited for the first time as we traverse the edge (u, v),
  1. then the edge is a **tree edge**.
- ELSE, v has already been visited:
  2. If v is an ancestor of u, then (u, v) is a **back edge**.
  3. Else, if v is a descendant of u, then (u, v) is a **forward edge**.
  4. Else, if v is neither an ancestor or descendant of u, then (u, v) is a **cross edge**. (u and v belong to different paths from the root)

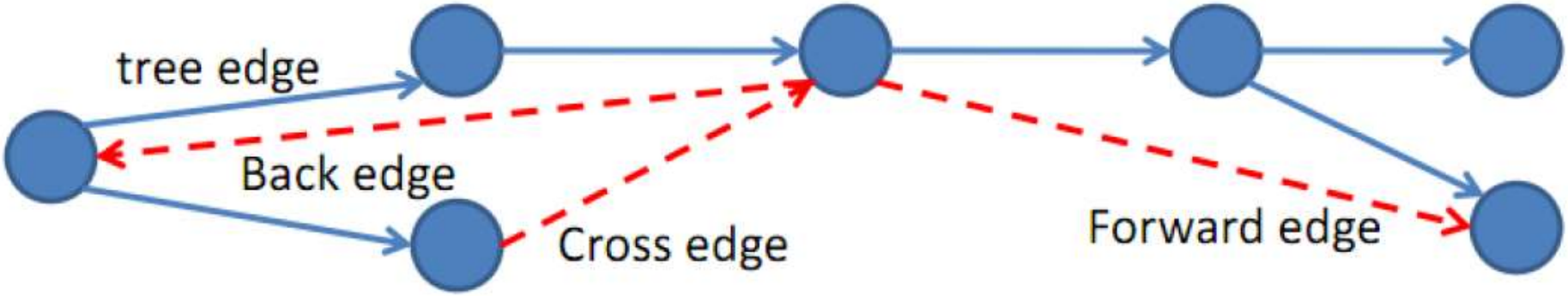


Reference: https://courses.csail.mit.edu/6.006/spring11/rec/rec13.pdf

Depth First Search: Edge Classification

Important Properties:

- tree edges form the DFS tree in G.
- G has a cycle if and only if DFS finds at least one back edge.
- G is undirected graph: it cannot contain forward edges and cross edges
  - the edge (v, u) would have already been traversed during DFS before we reach u and try to visit v via edge (u, v).

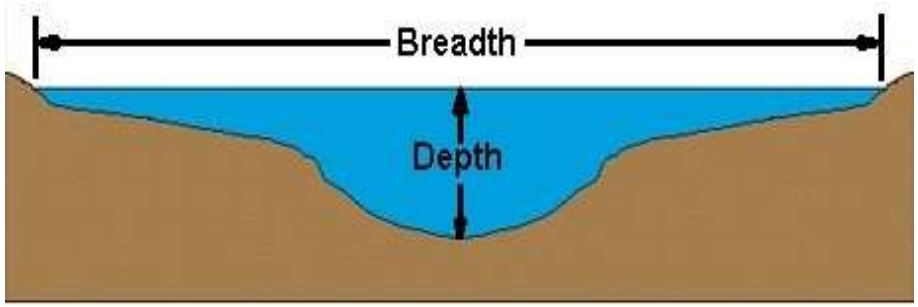


Reference: https://courses.csail.mit.edu/6.006/spring11/rec/rec13.pdf

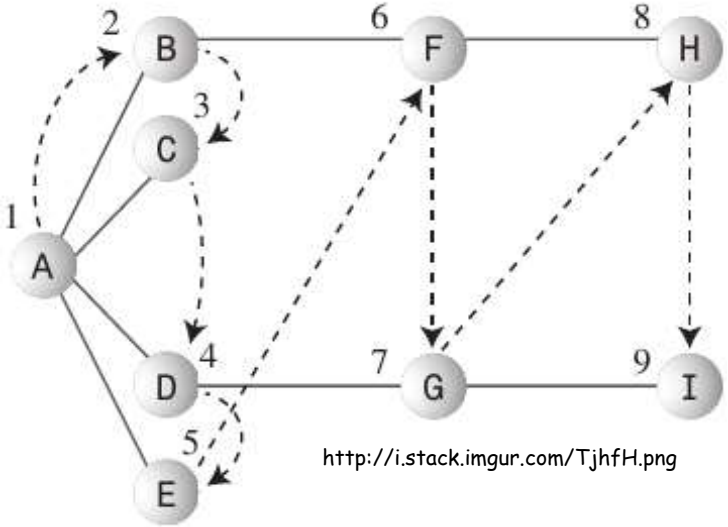
BFS vs. DFS

BFS: Put unvisited vertices on a queue.

- Examines vertices in increasing distance from s.
- Using adjacency list requires O(m).



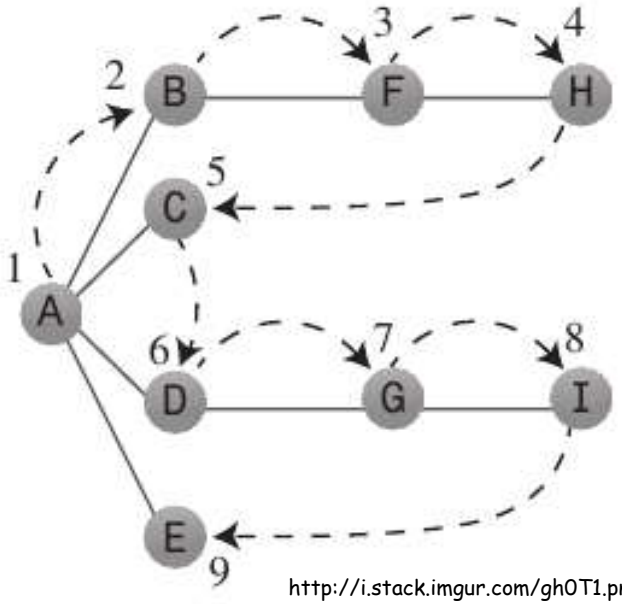
http://i.stack.imgur.com/QtYo8.jpg



http://i.stack.imgur.com/TjhfH.png

DFS: Put unvisited vertices on a stack.

- tries to explore as deeply as possible
  - Mimics maze exploration.
- O(m) due to similar reasoning.



http://i.stack.imgur.com/gh0T1.png

Finding all Articulation Points

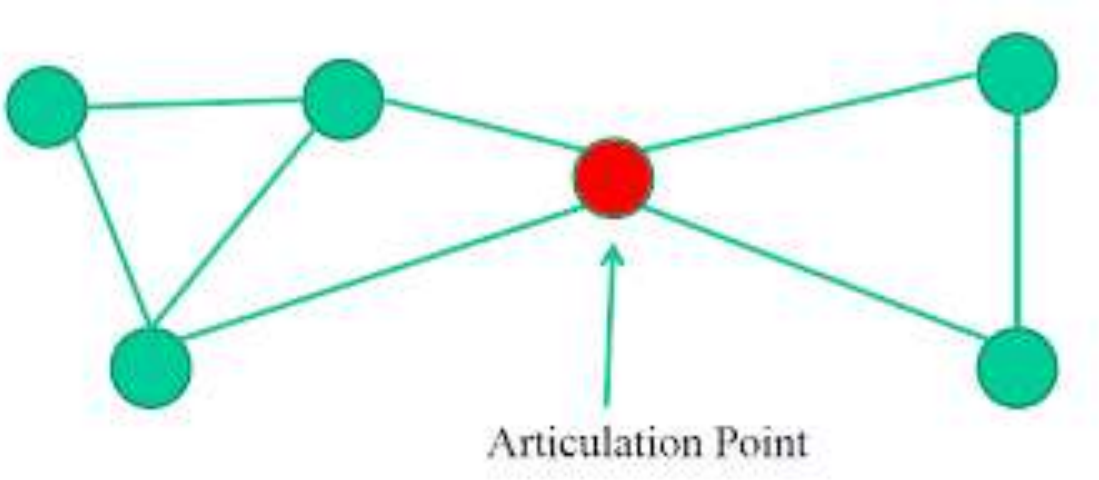
Def. An articulation point in a connected, undirected graph is a node v such that removal of v and of the edges incident to v makes the graph disconnected.

Given: an undirected graph $G = (V, E)$.

Goal: Find all articulation points.

Motivation:

reliability (failure tolerance) of communication networks



Reference: http://1.bp.blogspot.com/OykmfN7Z3tI/Us75fUCD1jI/AAAAAAAAAPo/sPZQnW0TL9Y/s1600/articulation+point.png

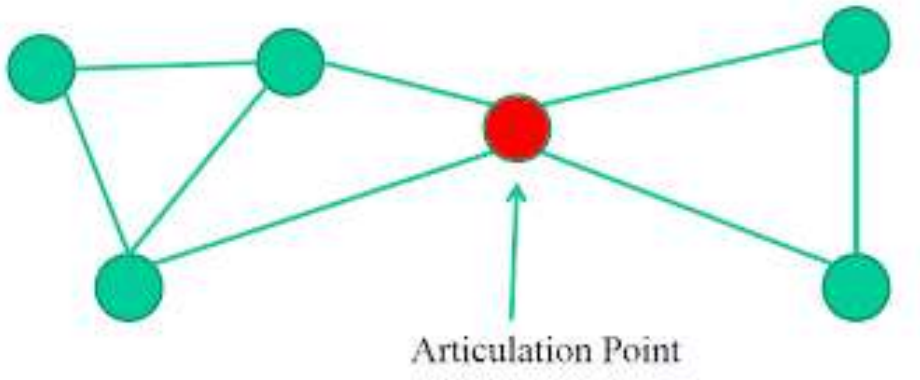
Finding all Articulation Points

Given: an undirected graph $G = (V, E)$.

Goal: Find all articulation points.

Observation: there no cross edges in DFS tree on undirected graphs.

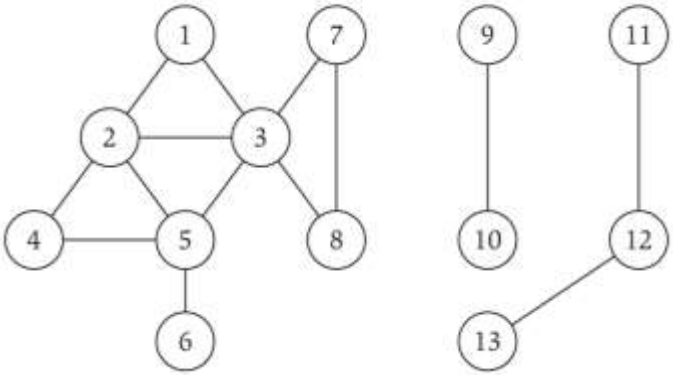
- Chose arbitrary start node s and find a DFS tree rooted at s
  - s is an articulation point iff s has more than one child in the DFS tree
- run DFS n times, once from every start node, which costs O(nm) time.
  - Amazingly, it is possible achieve O(m) time, using only one DFS tree and a variant of dynamic programming



Reference: http://1.bp.blogspot.com/OykmfN7Z3tI/Us75fUCD1jI/AAAAAAAAAPo/sPZQnW0TL9Y/s1600/articulation+point.png

Connected Component

Connected component. Find all nodes reachable from s.

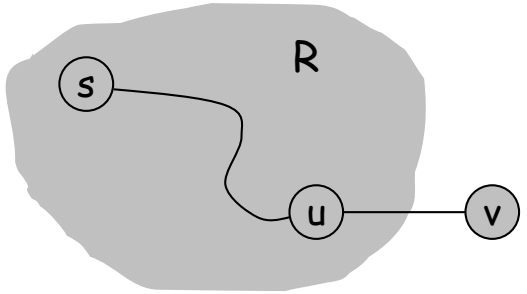


Connected component containing node 1 = { 1, 2, 3, 4, 5, 6, 7, 8 }.

Connected Component

Connected component. In undirected graph G, Find all nodes reachable from s.

R will consist of nodes to which s has a path
Initially $R = \{s\}$
While there is an edge (u, v) where $u \in R$ and $v \notin R$
  Add v to R
Endwhile



it's safe to add v

Upon termination, R is the connected component containing s.

- BFS, DFS $\Rightarrow$ O(m)

G is connected?

- iff all nodes are reachable from arbitrary node s

All connected components of G?

- O(m + n): If the search has aborted without finding all nodes, restart the search in a yet unmarked node, and so on.

Strong Connectivity

A directed graph G is strongly connected if, for every two nodes u and v, there is a path from u to v and a path from v to u.

- The street map of a city with one-way streets should be strongly connected, or the traffic planners made a mistake.

G is a directed graph, Is G Strongly Connected?

- G^{rev} :obtained from G simply by reversing the direction of every edge
- Run BFS (or DFS) with an arbitrary start node s, once on each G, and G^{rev}
  - both searches reach all nodes IFF G is Strongly Connected

Strong Connectivity

All Strongly connected components of G?

- Find out the strongly connected component containing s (same as in Strong Connectivity check)
- Restart the search in a node, yet not part of any strong component,
  - and so on.
- O(nm): Worst case G has many small connected components, O(m) time for each one using the above approach.

- Remark: O(m) is achievable by some tricky use of DFS, we skip.

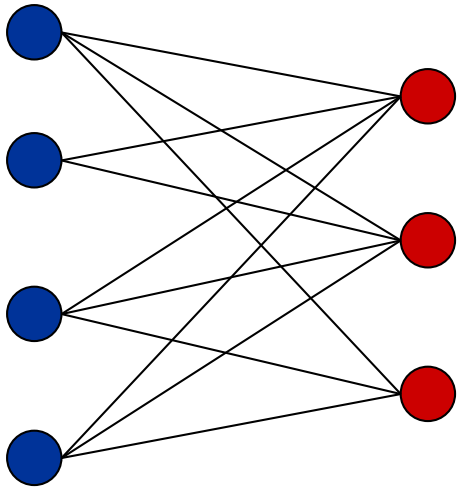
Testing Bipartiteness (One Graph Two colors)

Bipartite Graphs

Def. An undirected graph G = (V, E) is bipartite if the nodes can be colored red or blue such that every edge has one red and one blue end.

Applications.

- Scheduling: machines = red, jobs = blue.

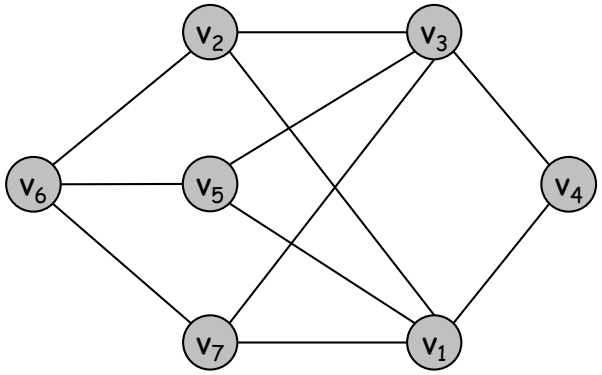


a bipartite graph

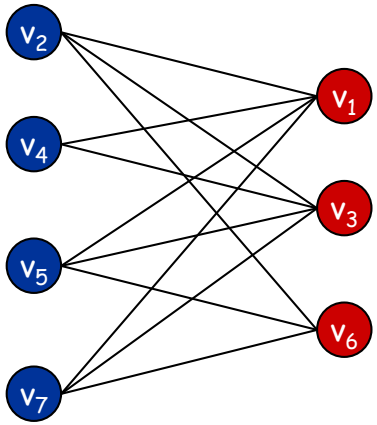
Testing Bipartiteness

Testing bipartiteness. Given a graph G, is it bipartite?

- Many graph problems become:
  - easier if the underlying graph is bipartite (matching)
  - tractable if the underlying graph is bipartite (independent set)

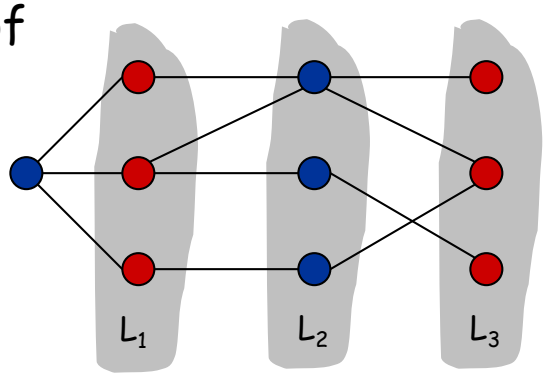


a bipartite graph G



another drawing of G

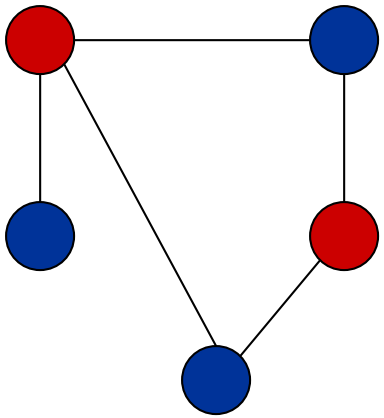
- we need to understand structure of bipartite graphs.



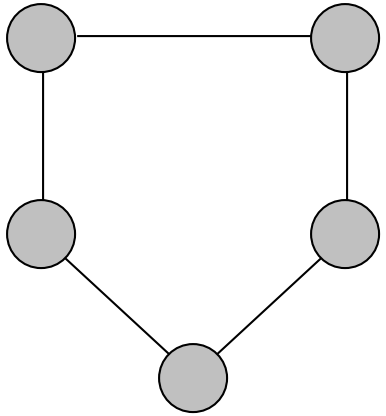
An Obstruction to Bipartiteness

Lemma. If a graph G is bipartite, it cannot contain an odd length cycle.

Pf. Not possible to 2-color the odd cycle, let alone G.



bipartite
(2-colorable)



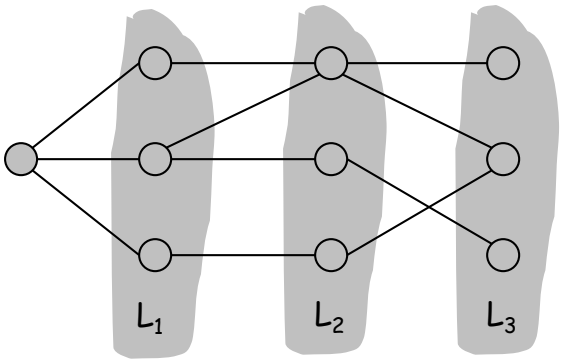
not bipartite
(not 2-colorable)

Bipartite Graphs

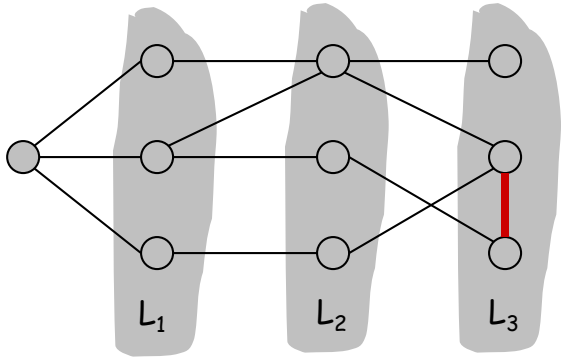
Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds.

(i) No edge of G joins two nodes of the same layer, and G is bipartite.

(ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).



Case (i)



Case (ii)

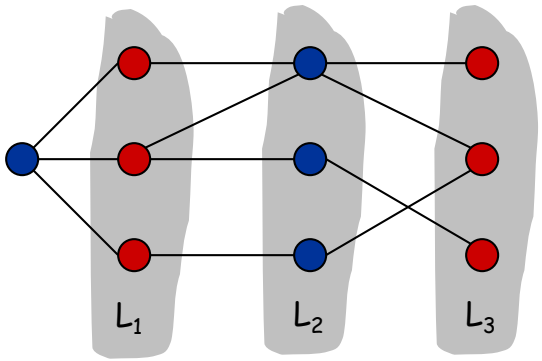
Bipartite Graphs

Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds.

(i) No edge of G joins two nodes of the same layer, and G is bipartite.

Pf. (i)

- By assumption edges don't join nodes on same layer
  - Every edge joins two nodes in adjacent layers.
- Bipartition: red = nodes on odd levels, blue = nodes on even levels.

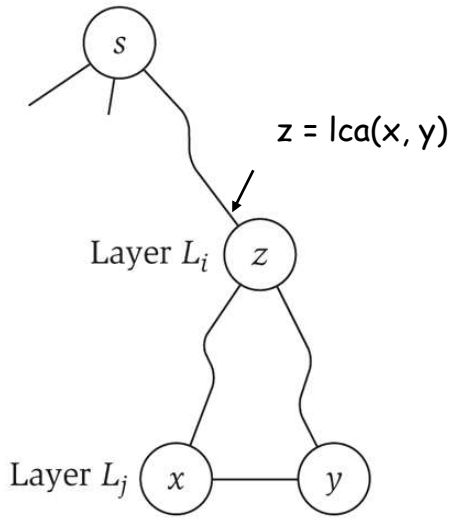


Case (i)

Bipartite Graphs

Lemma. Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds.

(ii) An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).



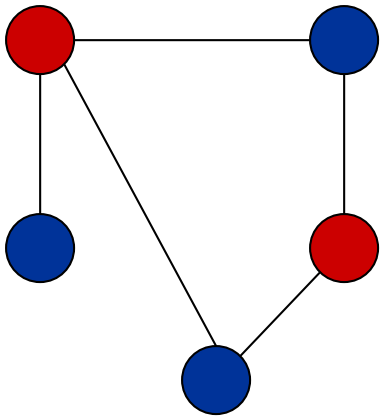
Pf. (ii)

- Suppose (x, y) is an edge with x, y in same level L_j.
- Let z = lca(x, y) = lowest common ancestor.
- Let L_i be level containing z.
- Consider cycle that takes edge from x to y, then path from y to z, then path from z to x.
- Its length is 1 + (j-i) + (j-i), which is $2(j - i) + 1$, an odd number. ▪

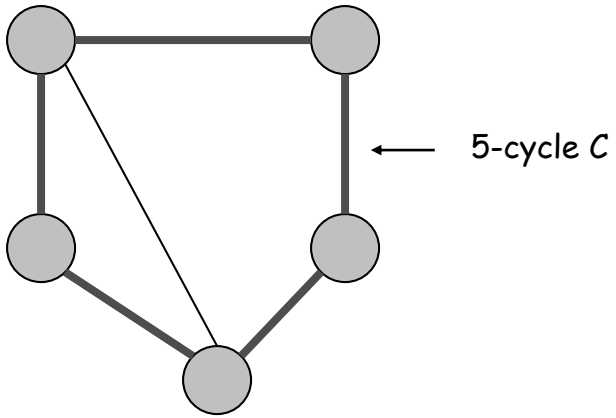
1: (x, y)
(j-i): path from y to z
(j-i): path from z to x

Obstruction to Bipartiteness

Corollary. A graph G is bipartite iff it contain no odd length cycle.



bipartite
(2-colorable)



not bipartite
(not 2-colorable)