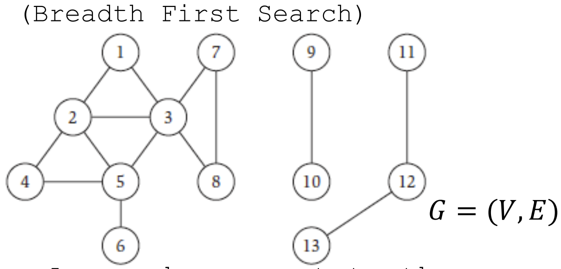
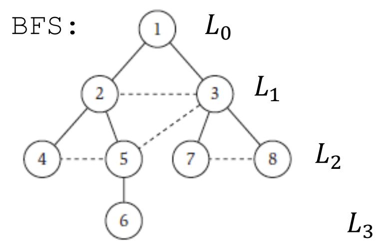
CSCI 323 Lecture 7 Note — Monday, 02/28/2022

Announcements:
- The Lecture Recordings will be available on the following YouTube Playlists Link: https://youtube.com/playlist?list=PLZaTmV9UMKlgYpo2cAiMaEWxqyvbiXDFd

Graph
References:
Algorithm Design - Chapter 3

BFS (Breadth First Search) & DFS (Depth First Search)
- Input to both algorithms is the graph $G = (V, E)$ and a source/root node s.
- Output a tree rooted at s (containing all vertices connected to s).
- Both algorithms run in $O(n + m)$ time.
- However, they output different trees.
- BFS (Breadth First Search)



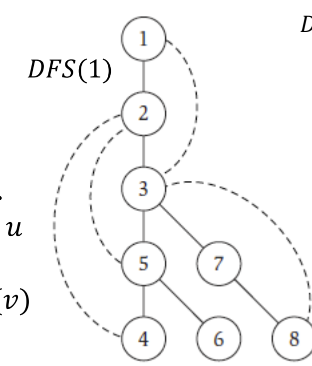


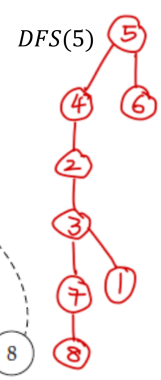
  - Any node connect to the source/root appears in this tree.
  - Layer j contains nodes that are distance j from the source/root. Where distant between two nodes s and t = least number of edges required to get from s and t. [3.3]
  - BFS give the distance (shortest path) from the source node to any other nodes. - single source shortest paths.
  - (There are edges in the graph that not in the tree.) Edges in G that do not appear in the BFS tree T connect nodes that are either in the same or adjacent layers. [3.4]
  - Pseudocode for BFS:

    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 $v \in R$
        Add v to R
    Endwhile

- DFS (Depth First Search)
  - It's a recursive algorithm
  - It has a source/root s.
  - Pseudocode for DFS:
    - $DFS(u)$:

        Mark u as explored and add u to R.
            For each edge (u, v) incident to u
                If v is not explored,
                then recursively invoke $DFS(v)$
            Endif
        Endfor

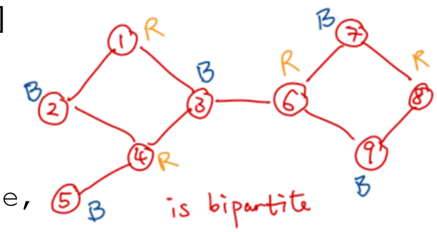


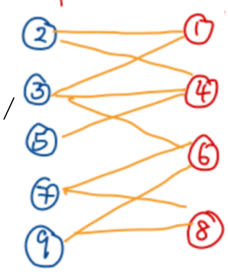


  - Gives a long & skinning trees compare to DFS trees.
  - Returns connected component.
  - Does not give distances!
  - Edges in G not in DFS tree connect ancestor to a descendant.

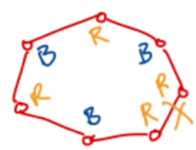
- Read Section 3.3 Implementation of BFS and DFS.
  - DFS – use stack
  - BFS – use either stack or queue
  - When implemented correctly, both return in $O(m+n)$.
    - $O(m+n)$ – linear time, because just to read the input takes $O(m+n)$ time (size of adjacency list).
    - In general, just to read in the input will take the linear time algorithm.

Application of BFS: Testing Bipartiteness. [Section 3.4]

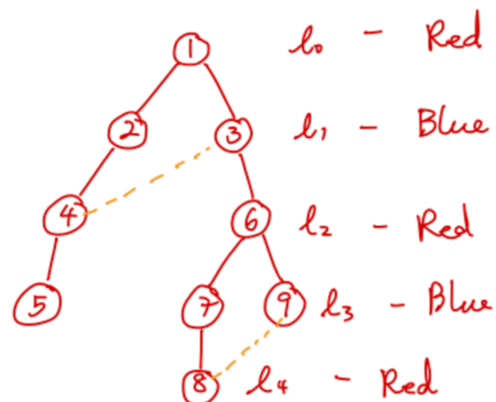




- Question: Given a graph G, is it bipartite?
- Definition: $G=(V,E)$ is *bipartite* if its vertex set V can be partitioned into two sets V_1 and V_2 such that no edge has endpoints in the same set.
  - You can think of the V_1 and V_2 as red and blue, and you want to have edges with different color endpoints.
  - For example, the graph on the right is bipartite. Some of you seem bipartite graph as the same color set/ vertices on one side.
- If G is bipartite, it cannot contain an odd cycle (cycle of length 3, 5, 7, …).
  - Proof: By contradiction if odd cycle.



- Algorithm:
  - Step 1: Run BFS, let's start at v_1.
  - Step 2: Give layers alternating colors.
  - Step 3: For every edge in G not in T, check if it connects vertices on the same layer or adjacent layers.
    - If same layer, stop, return no.
    - If all such edges connect vertices in adjacent layers, return yes.
  - Correctness?
    - If adjacent → coloring is correct.
    - If same layer → odd cycle in the graph.



- Algorithm Using DFS:
  - Step 1: Run DFS at any vertex.
  - Step 2: Give layers alternating colors.
  - Step 3: For every edge in G not in T, check if it connects vertices with even or odd distance in the tree T.
    - If the distance is even, stop, return no.
    - If all the distances are odd, return yes.
  - Correctness?
    - If the distance is even → odd cycle in the graph.
    - If the distance is odd → even cycle, coloring is good.

Directed Graph

- In a directed graph, edges have directions.
  In a *directed graph* or *digraph*, (v_i, v_j) means there is an edge from v_i to v_j and (v_j, v_i) means there is an edge from v_j to v_i.

- A path from u to v, looks like this: 

  If it exits, does not imply that a path from v to u exists.
- Connectivity is not symmetric.
- It does not affect the traversal algorithms: BFS and DFS.
  - Change: While exploring a vertex u, we previously looked at all edges incident to u, but now only look at edges with u as a "source".
  - It's still run in $O(m + n)$ time.
  - If we run BFS/DFS with this change, what kind of nodes are returned in the tree?
    - Previously (undirected) - return component of the source s.
    - Directed – return set of vertices that have a path from the source s, $P_{from}(s)$.
      - What if we want set of vertices that have a path to the source s, $P_{to}(s)$?
        - Reverse all directions in G to get the reverse graph G^{rev}, then run $\text{BFS/DFS}(s, G^{rev})$.

What to expect or prepare for the next class:
- Directed Graph
- DAG
- Topological Ordering

Reading Assignment
Algorithm Design: 3.1 – 3.4