COMP 3711

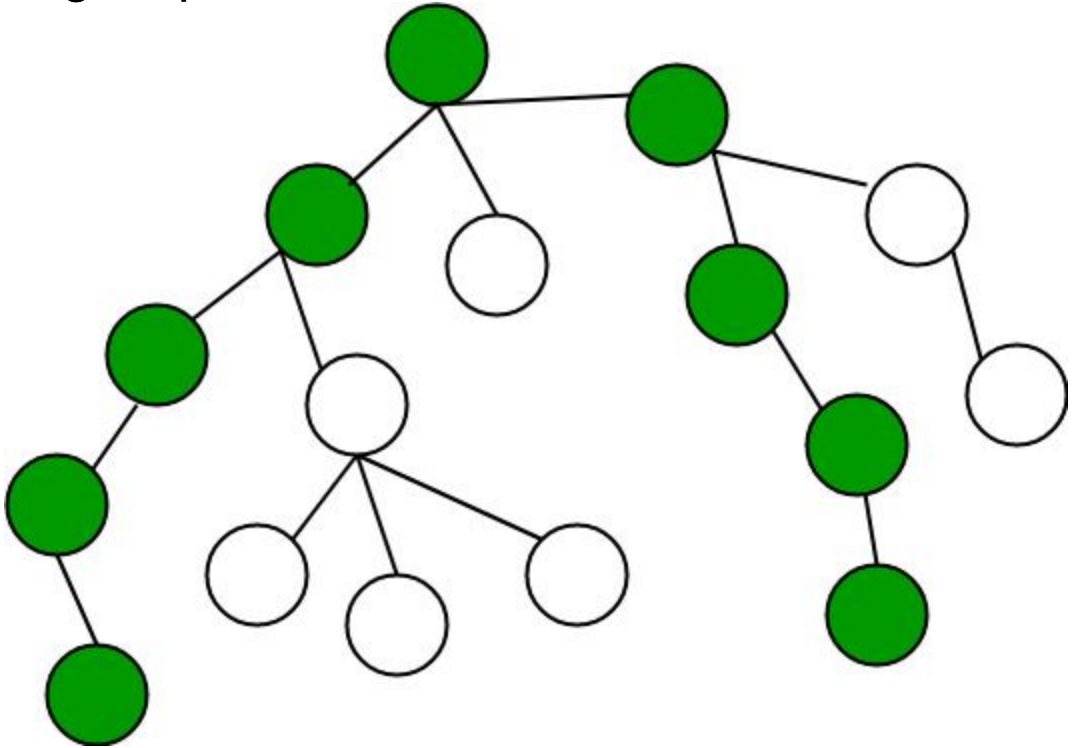
Design and Analysis of Algorithms

BFS,DFS

Arman Haghighi

Problem 1: Find the longest path in a Tree (diameter)

Problem 1: Consider an undirected unweighted Tree T=(V,E) (Or assume the weight of each edge is 1). Find the longest path in this tree. Below is an example of the longest path.



Problem 1: Find the longest path in a Tree (diameter)

Idea1: Every edge has weight 1. Use Floyd-Warshall algorithm to find the shortest path between all the node pairs.

+ Easy solution
- There are more optimized algorithms. This algorithm is $O(|V|^3)$

Question: Can you find an algorithm that is of $O(|V|)$

Remark: In a tree, the number of edges is V-1. So algorithms like BFS have the complexity of O(|V|).

Question: Can you solve it by using two BFS?!

Problem 1: Find the longest path in a Tree (diameter)

Idea: Running BFS on a node can determine the closest and furthest nodes.

If we know one of the ending points of this path we can simply find the longest path.

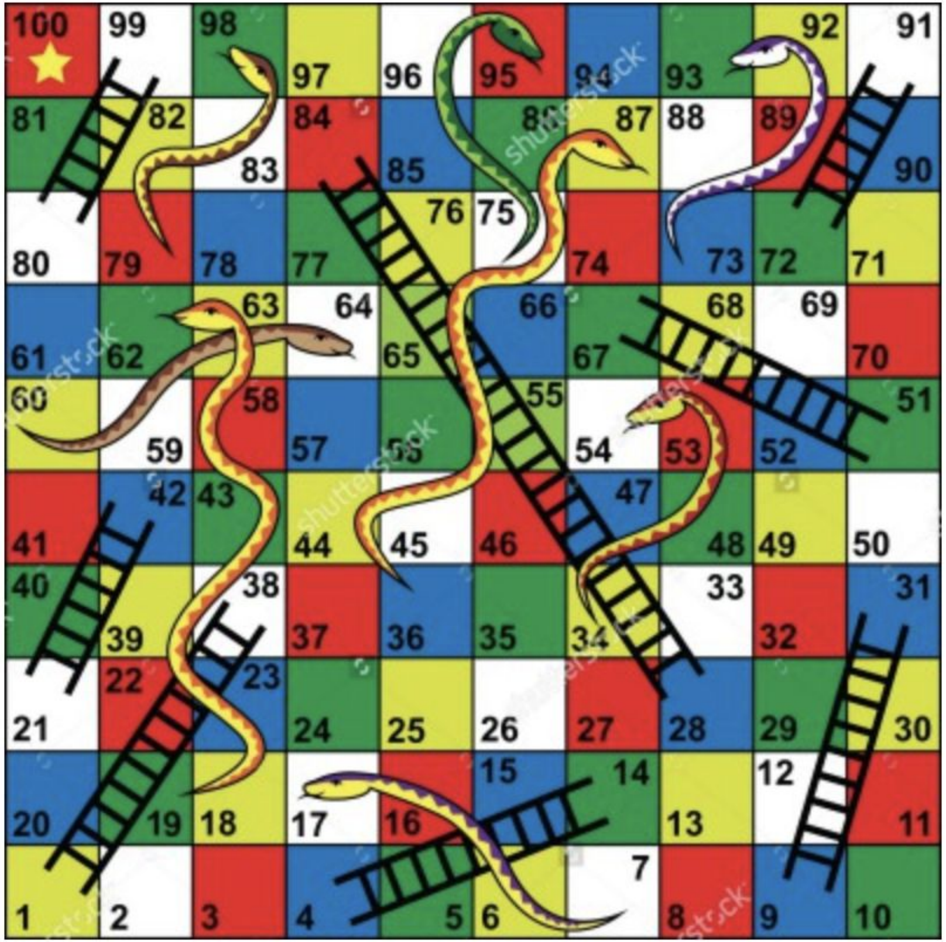
How to find one of the ending points?

- Choose any random node.
- Run BFS from random node. The furthest node to this node is **ALWAYS** one of the ending points of the longest path. (why? This is not obvious at all! Try to prove it using contradiction)
- Run BFS again from that furthest point as it is always one of the ending points. That will be the longest path !

Problem 2: Snake and Ladder Problem

Problem 2:

Find the minimum number of throws required to win a given Snakes and Ladders board game.



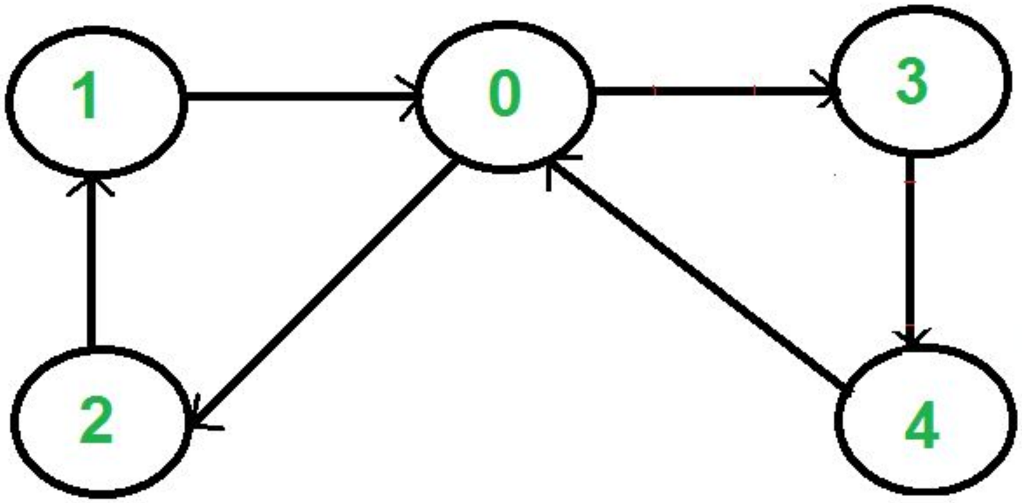
Problem 2: Snake and Ladder Problem

Idea: As this is an unweighted graph, we can find shortest path to a node by running BFS. We need to calculate shortest path from 1 to 100?
But what are the edges and nodes in this graph?

- Each cell is a node.
- Each cell "i" is originally connected to i+1, i+2, i+3, … , i+6.
    - If there exists a ladder or snake from some j to k and an edge from i to j. Simply remove these two edge and directly connect i to k.
- Run BFS to find the shortest path from 0 to 100.

Problem 3: Cycle with odd length

Problem 3: Consider a directed unweighted graph. You need to output if there exists a cycle including node “s” with odd length.



Problem 3: Cycle with odd length

Idea: Run BFS from "s". If you visit "s" again later in an odd step then there exists a cycle with odd length. The complexity is the same as BFS.

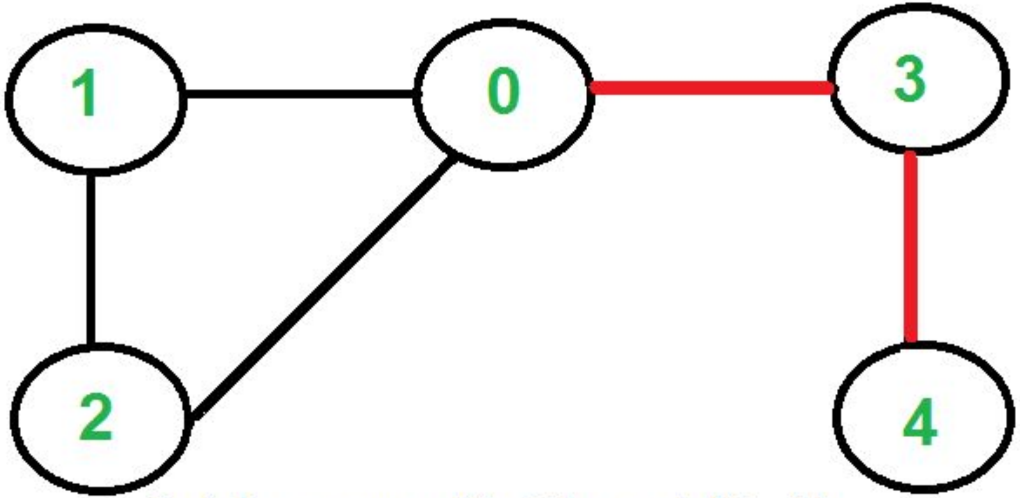
Another elegant idea!:

- Create a new graph, Duplicate each node x and name them as x and x'.
- For each edge connecting nodes a to b, connect a to b' and a' to b.
- Check if you there exists a path from s to s'

Problem 4: Finding Bridges in a graph

Problem 4: Bridge is an edge that removing it creates more component in a graph.

Design an algorithm to print all bridges!



Bridges are (0, 3) and (3, 4)

Problem 4: Finding Bridges in a graph

Idea1: Remove each edge one by one and run BFS or DFS to see if you can visit all the nodes. If you can't visit all the nodes then its a bridge.

Whats the time complexity? $O(E \,.\, (V+E))$

Question: Can you solve it in $O(V+E)$

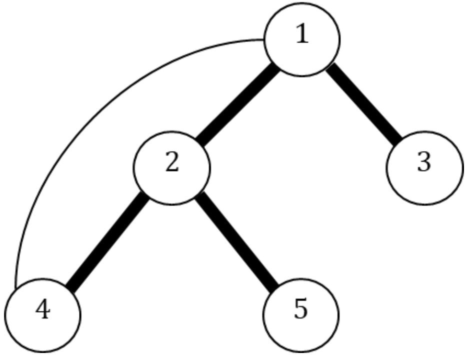
Idea2: Use Tarjan's Algorithm!

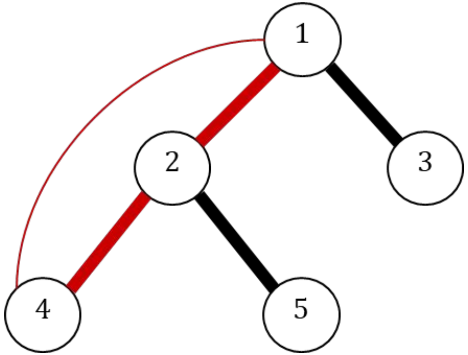
Problem 4: Finding Bridges in a graph

Equivalent question: Describe an algorithm to find all edges that are not part of a cycle in a graph.

Think of the DFS tree: we know there are no **cross edges**, only **tree edges** and **back edges**

A back edge “creates” a cycle, so no edge from v to u can be a bridge





We keep 3 values for each node u.

- Dis[u] : Discovery time for u.
- Low[u]: The earliest ancestor of u except its parent node
- Parent[u] : The parent of u.

Start DFS. When you see a node that is already visited it is a backedge! So update the low[u] value.

When we backtrack, we update the low value again based on the minimum (or earliest) ancestor. The complexity is the same as DFS now!

When can we claim an edge is a bridge using low and discovery of its ending points?

Check Tarjan's Algorithm for more details!