CPSC 320: Intermediate Algorithm Design 2019/20 Term 2

Tutorial 3

Prof. Nick Harvey *University of British Columbia*

1. **(Testing Bipartiteness)** A bipartite graph is one whose vertices can be colored red and blue, such that there are no blue-blue edges and no red-red edges.

   (a): **(Odd Cycles)** Let G be a graph that contains a cycle of length n, where n is odd. Explain why G cannot be bipartite.

   Surprisingly, the existence of odd cycles is the only reason for a graph not being bipartite.

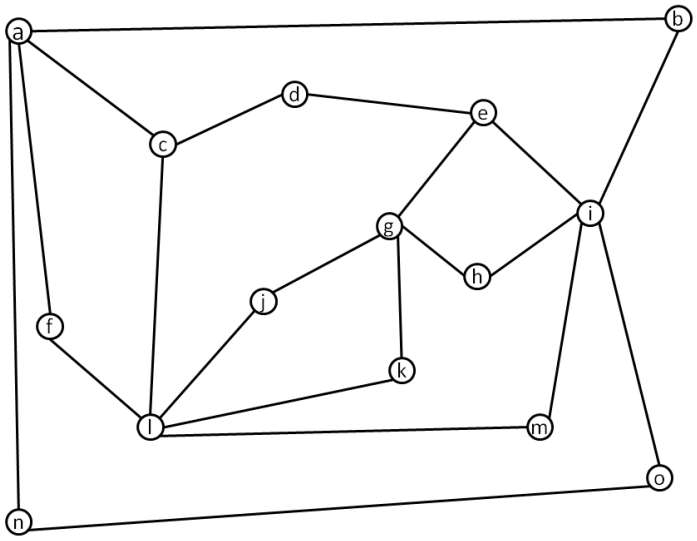
   Claim 1. A graph is bipartite if and only if it has no odd cycle.

   We claim that the following algorithm proves this claim.

Algorithm 1 Algorithm to test bipartiteness.

Run BFS starting from any vertex. Let L_i be the levels it produces.
Let $B = L_0 \cup L_2 \cup L_4 \cdots$
Let $R = L_1 \cup L_3 \cup L_5 \cdots$
if there is an edge between two vertices in B or two vertices in R **then**
  return "non-bipartite"
else
  return "bipartite"
end if

(b): As an example, run that algorithm on the following graph, where the BFS starts from, say, vertex d.



Let us now discuss correctness of the algorithm.

Case 1: If the algorithm returns "bipartite" then that is obviously correct because has created a coloring with no blue-blue or red-red edges.

Case 2: If the algorithm returns "non-bipartite" then we will find an odd cycle using the BFS tree. Suppose the algorithm finds an edge u-v with both $u, v \in R$. Let w be the least-common ancestor in the BFS tree of u and v. (Meaning that, if you follow the path of parent pointers from u to s and follow the path from v to s, then w is the first vertex where these paths meet.)

Claim 2. (length of uw path) + (length of vw path) is even.

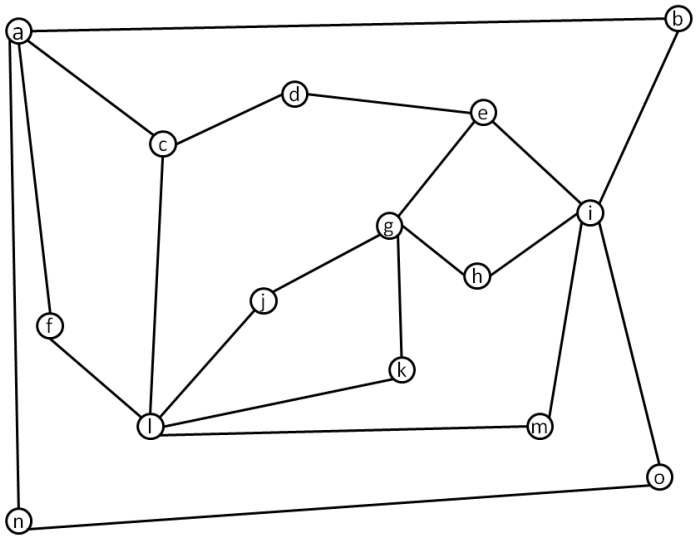
Proof. As in class, let $d(v)$ denote the shortest-path distance from s to v. Then

$$\begin{aligned}(\text{length of } uw \text{ path}) + (\text{length of } vw \text{ path}) &= (d(u) - d(w)) + (d(v) - d(w)) \\ &= \underbrace{(d(u) + d(v))}_{\text{always even}} - \underbrace{2 \cdot d(w)}_{\text{always even}} .\end{aligned}$$

This is even, because $d(u)$ and $d(v)$ are either both even or both odd. □

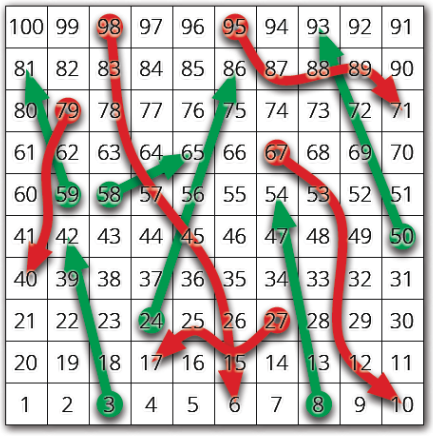
So in Case 2, we obtain an odd cycle by combining the path from u to w, the path from v to w, and the edge u-v.

(b): Let us continue the same example from before:



Look at the edge you found for which both endpoints have the same color. Find the least-common ancestor of its endpoints, and the corresponding odd cycle. How long is your odd cycle?

2. **(Snakes and Ladders)** This is a classic board game, originating in India no later than the 16th century. The board consists of an $g \times g$ grid of squares, numbered consecutively from 1 to g^2, starting in the bottom left corner and proceeding row by row from bottom to top, with rows alternating to the left and right. Certain pairs of squares in this grid, always in different rows, are connected by either "snakes" (leading down) or "ladders" (leading up). Each square can be an endpoint of at most one snake or ladder.



A typical Snakes and Ladders board.
Upward straight arrows are ladders; downward wavy arrows are snakes.

You start with a token in cell 1, in the bottom left corner. In each move, you advance your token *up to* k positions, for some fixed constant k. If the token ends the move at the top end of a snake, it slides down to the bottom of that snake. Similarly, if the token ends the move at the bottom end of a ladder, it climbs up to the top of that ladder. Describe and analyze an algorithm to compute the smallest number of moves required for the token to reach the last square of the grid.