

Tutorial 3

Prof. Nick Harvey

University of British Columbia

1. (**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.

100	99	98	97	96	95	94	93	92	91
81	82	83	84	85	86	87	88	89	90
80	79	78	77	76	75	74	73	72	71
61	62	63	64	65	66	67	68	69	70
60	59	58	57	56	55	54	53	52	51
41	42	43	44	45	46	47	48	49	50
40	39	38	37	36	35	34	33	32	31
21	22	23	24	25	26	27	28	29	30
20	19	18	17	16	15	14	13	12	11
1	2	3	4	5	6	7	8	9	10

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.

2. **(Testing Bipartiteness)** In lecture we talked about bipartite graphs, which are useful for modeling situations like stable matching, matching clients to servers, etc. Recall that 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.

We observed that an odd cycle (a cycle with an odd number of vertices) cannot possibly be bipartite. This is because its colors must alternate red-blue-red-blue-... until at the end we have two consecutive vertices of the same color. For the same reason, any graph containing an odd cycle cannot be bipartite. And, surprisingly, that is the only possible reason for being not bipartite.

Claim 0.1. A graph is bipartite if and only if it has no odd cycle.

We claim that the following algorithm proves this claim.

Run BFS starting from any vertex.

Let $dist[v]$ denote the distance to vertex v computed by BFS.

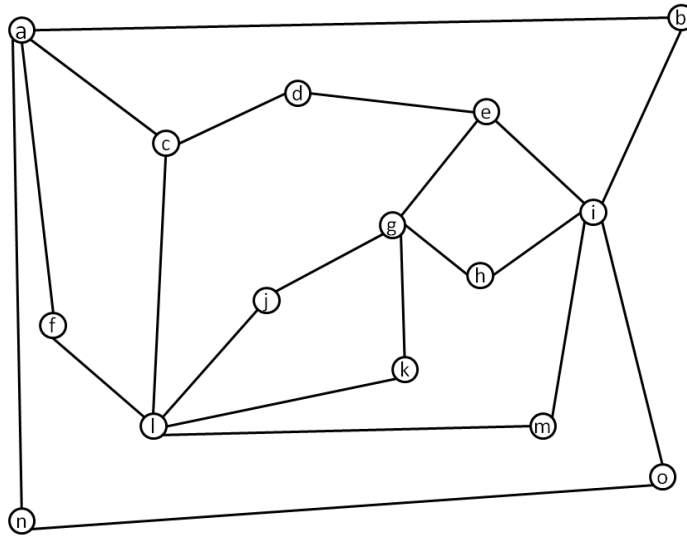
Let $B = \{ v : dist[v] \text{ is even} \}$.

Let $R = \{ v : dist[v] \text{ is odd} \}$.

If there is an edge between two vertices in B , or two vertices in R return “non-bipartite”.

Otherwise, let all vertices in B be blue, let all vertices in R be red, and return “bipartite”.

- (a): As an example, run that algorithm on the following graph, where the BFS starts from the labeled “root” (vertex d).



Let us now discuss correctness of the algorithm.

Case 1: If the algorithm returns “bipartite” then that is obviously correct because the algorithm checked that there are 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 w and follow the path from v to w , then w is the first vertex where these paths meet.)

Then

$$(dist[u] - dist[w]) + (dist[v] - dist[w])$$

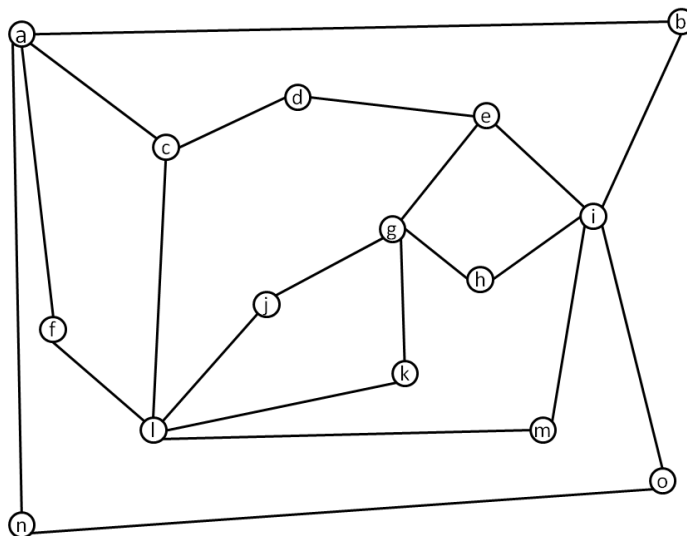
is always even because it equals

$$\underbrace{(dist[u] + dist[v])}_{\text{always even}} - \underbrace{2 \cdot dist[w]}_{\text{always even}}$$

which is even, because $dist[u]$ and $dist[v]$ are either both even or both odd.

So 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?