

1. We discussed DFS and BFS in class. Modify one of these two algorithms to take a node v and find the length of the shortest path from v to u for every node u in the graph.

For the problems below, transform the input into a graph and apply algorithm above. Whenever you use a graph algorithm, you **must** provide the following information. (I recommend actually using a bulleted list.)

- What are the vertices?
 - What are the edges? Are they directed or undirected?
 - If the vertices and/or edges have associated values, what are they?
 - What problem do you need to solve on this graph?
 - What standard algorithm are you using to solve that problem?
 - What is the running time of your entire algorithm, *including* the time to build the graph, as a function of the original input parameters?
2. **Snakes and Ladders** is a classic board game, originating in India no later than the 16th century. The board consists of an $n \times n$ grid of squares, numbered consecutively from 1 to n^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, 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 (typically 6). If the token ends the move at the *top* end of a snake, you **must** slide the token down to the bottom of that snake. If the token ends the move at the *bottom* end of a ladder, you **may** move the token 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.

3. Let G be a connected undirected graph. Suppose we start with two coins on two arbitrarily chosen vertices of G . At every step, each coin **must** move to an adjacent vertex. Describe

and analyze an algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph $G = (V, E)$ and two vertices $u, v \in V$ (which may or may not be distinct).

To think about later:

4. Let G be an undirected graph. Suppose we start with 374 coins on 374 arbitrarily chosen vertices of G . At every step, each coin must move to an adjacent vertex. Describe and analyze an efficient algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph $G = (V, E)$ and starting vertices s_1, s_2, \dots, s_{374} (which may or may not be distinct).