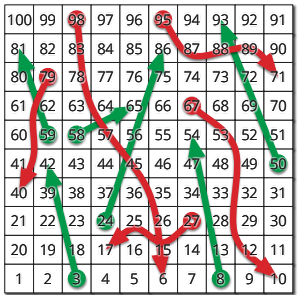
For each of the problems below, transform the input into a graph and apply a standard graph algorithm that you've seen in class. Whenever you use a standard graph algorithm, you ***must*** provide the following information. (I recommend actually using a list as the one used below.)

- 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*?

1 ***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.



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.

Solution:

We reduce to a shortest-path problem in a directed graph $G = (V, E)$ as follows:

- The vertices of G correspond to cells on the board, identified by integers 1 to n^2.
- The edges of G correspond to legal moves. From each cell there are at most $2k$ possible moves: for each integer i from 1 to k, we can move forward i spaces, and then if we are at the bottom of a ladder, we can either move to the top of that ladder or not. Edges are directed.
- We do not need to associate additional values with the vertices or edges.

- We need to find the shortest path from vertex 1 to vertex n^2.
- We can solve this problem using breadth-first search.
- The algorithm runs in $O(V + E) = O(n^2 + 2kn^2) = \boldsymbol{O(kn^2)}$ ***time***.

2 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).

Solution:

[product construction] Let $G = (V, E)$ denote the input graph, and let s and t denote the initial locations of the two coins. We reduce to a shortest-path problem in an undirected graph $G' = (V', E')$ as follows:

- $V' = V \times V = \{(u, v) \mid u \in V \text{ and } v \in V\})$; the vertices of G' correspond to possible placements of the two coins.
- $E' = \{(u, v)(u', v') \mid uu' \in E \text{ and } vv' \in E\}$. The edges of G' correspond to legal moves by the two coins. Edges are undirected, because any move by the two coins can be reversed.
- We do not need to associate additional values with the vertices or edges.
- We need to find the shortest-path distance from vertex (s, t) to any vertex of the form (v, v).
- First we compute the shortest-path distance from (s, t) to every vertex in G' that is reachable from (s, t) using breadth-first search. Then a for-loop over the vertices of G' finds the minimum distance to any marked vertex of the form (v, v). In particular, if no vertex (v, v) is reachable from (s, t), then no vertex (v, v) will be marked by the breadth-first search, and so the algorithm will correct report $\min \varnothing = \infty$.
- The resulting algorithm runs in $O(V' + E') = O(V^2 + E^2)$ time.

Solution:

[parity construction] Let $G = (V, E)$ denote the input graph, and let s and t denote the initial locations of the two coins. Any sequence of k moves that bring the two coins to a common vertex v defines a walk of length $2k$ from s through v to t. Thus, we are looking for the shortest walk from s to t with even length. We reduce to a standard shortest-path problem in a new graph $G' = (V', E')$ as follows:

- $V' = V \times \{0, 1\} = \{(v, b) \mid b \in V \text{ and } b \in \{0, 1\}\})$.
- $E' = \{(u, b)(v, 1 - b) \mid uv \in E \text{ and } b \in \{0, 1\}\}$. Then for any walk $v_0 \to v_1 \to v_2 \to \cdots \to v_\ell$ in G, there is a corresponding walk $(v_0, 0) \to (v_1, 1) \to (v_2, 0) \to \cdots \to (v_\ell, \ell \bmod 2)$ in G'. Thus, every even-length walk from s to t in G corresponds to a walk in G' from $(s, 0)$ to $(t, 0)$ and vice versa.
- We do not need to associate additional values with the vertices or edges.
- We need to find the shortest-path distance in G' from vertex $(s, 0)$ to $(t, 0)$.
- We can compute this shortest-path distance using breadth-first search starting at $(s, 0)$. In particular, if there is no even-length path from s to t in G, the breadth-first search will not mark $(t, 0)$.
- The resulting algorithm runs in $O(V' + E') = O(V + E)$ time.