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 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*?

---

1. Snakes and Ladders is played on 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.

   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 a token to move from the first square to the last square on an $n \times n$ Snakes and Ladders board. The input to your algorithm consists of an $n \times n$ grid where each cell either contains a snake or ladder pointing towards another cell, or nothing at all.

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; if we are at the top of a snake, we must move to the bottom of the snake. 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.
- Constructing the graph by brute force takes $O(V + E)$ time, as does running breadth-first search. Thus the algorithm runs in $O(V + E) = O(n^2 + 2kn^2) = \boldsymbol{O(kn^2)}$ ***time***. ■

2. You are given a list $D[n]$ of n words each of length k over an alphabet Σ in a language you don't know, although you are told that words are sorted in lexicographic order. Using $D[n]$, describe an algorithm to efficiently identify the order of the symbols in Σ. For example, given the alphabet $\Sigma = \{Q, X, Z\}$ and the list $D = \{QQZ, QZZ, XQZ, XQX, XXX\}$, your algorithm should return QZX. You may assume D always contains enough information to completely determine the order of the symbols. (Hint: use a graph structure, where each node represents one letter.)

Solution: Consider two words, $D[i]$, $D[i+1]$. Consider j such that $D[i][j] \neq D[i+1][j]$ and $\forall k < j$, $D[i][k] = D[i+1][k]$. That is, j is the index of the first different letter between $D[i]$ and $D[i+1]$. The comparison of $D[i][j]$ and $D[i+1][j]$ reveals the order between the two letters. Any further comparison of $D[i]$ and $D[j]$ would not help, since the following letters do not affect lexicographic order of $D[i]$ and $D[i+1]$. Also, for arbitrary x, y, z such that $x < y < z$, if you are given the comparisons of $D[x], D[y]$ and $D[y], D[z]$, then the comparison of $D[x], D[z]$ does not reveal any additional information about the order(Why? Let j, k be the first different index between $D[x], D[y]$ and $D[y], D[z]$ respectively. Reason about three cases: $j < k$, $j = k$, $j > k$). Therefore, the problem can be solved by constructing the following directed graph.

- $V = \{v \mid v \in \Sigma\}$
- $E = \{(u, v) \mid u, v \in \Sigma, u \neq v, \exists i, j \text{ s.t. } D[i][j] = u, D[i+1][j] = v, \forall k < j, D[i][k] = D[i+1][k]\}$

Note that for any edge $(u, v) \in E$, there is a corresponding pair of consecutive words $(D[i], D[i+1])$ such that if j is the index of the first different letter, then $D[i][j] = u$ and $D[i+1][j] = v$. This means for any edge (u, v), we know for sure that u comes before v in their language. Since there can be no cycle in the graph, it can be topologically sorted to obtain the order of symbols. The running time of the algorithm is $O(nk)$, since in worst case we should iterate over every symbol in D to construct the graph.

■

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).

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 simple for-loop over the vertices of the input graph 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 correctly report $\min \varnothing = \infty$.
- Constructing the graph by brute force takes $O(V' + E')$ time, as does running breadth-first search. Thus the resulting algorithm runs in $O(V'+E') = \boldsymbol{O(V^2+E^2)}$ ***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 x defines a walk of length $2k$ from s through x 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:

- $V' = V \times \{0, 1\} = \{(v, b) \mid b \in V \text{ and } b \in \{0, 1\}\})$.
- $E' = \{(u, b)(v, 1 - b) \mid uv \in E \text{ and } b \in \{0, 1\}\}$. Edges in G' are undirected, because edges in the original graph G are undirected.

  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 from $(s, 0)$ to $(t, 0)$ in G' 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 $(t, 0)$.

- Constructing the graph by brute force takes $O(V' + E')$ time, as does running breadth-first search. Thus the resulting algorithm runs in $O(V' + E') = \boldsymbol{O(V + E)}$ ***time***.

■

Think about later:

3. 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 all 374 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 $s_1, s_2, \ldots, s_{374}$ (which may or may not be distinct).

Solution (product construction): We reduce to a shortest-path problem in an undirected graph $G' = (V', E')$ as follows:

- $V' = V^{374} = \overbrace{V \times V \times \cdots \times V}^{374} = \{(v_1, v_2, \ldots, v_{374}) \mid v_i \in V \text{ for all } i\}$; the vertices of G' correspond to possible placements of the 374 coins.
- $E' = \{(u_1, u_2, \ldots, u_{374})(v_1, v_2, \ldots, v_{374}) \mid u_i v_i \in E \text{ for all } i\}$. The edges of G' correspond to legal moves by the 374 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 $s = (s_1, s_2, \ldots, s_{374})$ to any vertex of the form $(v, v, \ldots, v)$.
- First we compute the shortest-path distance from s to every vertex in G' that is reachable from s using breadth-first search. Then a simple for-loop over the vertices of the input graph G finds the minimum distance to any marked vertex of the form $(v, v, \ldots, v)$. In particular, if no vertex $(v, v, \ldots, v)$ is reachable from s, then no vertex $(v, v, \ldots, v)$ will be marked by the breadth-first search, and so the algorithm will correctly report $\min \varnothing = \infty$.
- Constructing the graph by brute force takes $O(V' + E')$ time, as does running breadth-first search. Thus the resulting algorithm runs in $O(V' + E') = \boldsymbol{O(V^{374} + E^{374})}$ ***time***.

■

Solution (parity construction): I claim that (1) there is a sequence of k steps that move all coins to some vertex t if and only if (2) there is a walk of length k_i from starting vertex s_i to t, for each vertex i, such that $k = \max_i k_i$ and either all k_i are even or all k_i are odd.

- The implication (1) $\implies$ (2) follows immediately by setting $k_i = k$ for all i.
- Any walk of length ℓ can be turned into a walk of length $\ell + 2$ with the same endpoints by repeating an edge, and therefore into a walk of length $\ell + 2j$ for any integer $j \geq 0$. The implication (2) $\implies$ (1) follows immediately.

Now for any vertex v and any index i, we define two integers:

- $\mathit{deven}(s_i, v)$ is the length of the shortest *even*-length walk from s_i to v (or ∞ if there is no such walk).

- $dodd(s_i, v)$ is the length of the shortest *odd*-length walk from s_i to v (or ∞ if there is no such walk).
- $MinSteps(v)$ is the minimum number of steps required to move all coins to v. My earlier claim implies that

$$MinSteps(v) = \min\left\{\max_i deven(s_i, v),\ \max_i dodd(s_i, v)\right\}$$

We need to compute $\min\{MinSteps(v) \mid v \in V\}$.

Consider the unweighted undirected graph $G' = (V', E')$ where $V' = V \times \{0, 1\}$ and $E' = \{(u, 0)(v, 1) \mid uv \in E\} \cup \{(u, 1)(v, 0) \mid uv \in E\}$. We immediately have

$$deven(s_i, v) = d'((s_i, 0), (v, 0)) \qquad \text{and} \qquad dodd(s_i, v) = d'((s_i, 0), (v, 1)),$$

where $d'(u', v')$ is the shortest-path distance from u' to v' in G'. Thus, we can compute $deven(s_i, v)$ and $dodd(s_i, v)$ for every vertex $v \in V$ and every index i by running 374 breadth-first searches in G', each starting at some vertex $(s_i, 0)$, in total $374 \times O(V' + E') = O(V + E)$ time. After computing these distances, we can easily compute $\min\{MinSteps(v) \mid v \in V\}$ in $O(V)$ time by brute force, because $374 = O(1)$.

Altogether, our algorithm runs in $\boldsymbol{O(V + E)}$ ***time***. ■