1. Inspired by the previous lab, you decide to organize a Snakes and Ladders competition with n participants. In this competition, each game of Snakes and Ladders involves three players. After the game is finished, they are ranked first, second, and third. Each player may be involved in any (non-negative) number of games, and the number need not be equal among players.

   At the end of the competition, m games have been played. You realize that you forgot to implement a proper rating system, and therefore decide to produce the overall ranking of all n players as you see fit. However, to avoid being too suspicious, if player A ranked better than player B in any game, then A must rank better than B in the overall ranking.

   You are given the list of players and their ranking in each of the m games. Describe and analyze an algorithm that produces an overall ranking of the n players that is consistent with the individual game rankings, or correctly reports that no such ranking exists.

Solution: We reduce to the topological sorting problem in a directed acyclic graph $G = (V, E)$ as follows:

- V is the set n of players.
- E contains a directed edge $i \to j$ if player i ranked higher than player j in any game. Since each game ranks three pairs of players, E contains $3m$ edges.
- No additional values are associated with the nodes or edges.
- We need to compute a topological order for this dag or report correctly that no such order exists.
- We can find a topological order using depth-first search.
- The algorithm runs in $O(V + E) = \boldsymbol{O(n + m)}$ ***time***. ■

2. There are n galaxies connected by m intergalactic teleport-ways. Each teleport-way joins two galaxies and can be traversed in both directions. However, the company that runs the teleport-ways has established an extremely lucrative cost structure: Anyone can teleport *further* from their home galaxy at no cost whatsoever, but teleporting *toward* their home galaxy is prohibitively expensive.

   Judy has decided to take a sabbatical tour of the universe by visiting as many galaxies as possible, starting at her home galaxy. To save on travel expenses, she wants to teleport away from her home galaxy at every step, except for the very last teleport home.

   Describe and analyze an algorithm to compute the maximum number of galaxies that Judy can visit. Your input consists of an undirected graph G with n vertices and m edges describing the teleport-way network, an integer $1 \le s \le n$ identifying Judy's home galaxy, and an array $D[1..n]$ containing the distances of each galaxy from s.

Solution: We reduce this problem to finding the length of the longest path in a dag $G = (V, E)$ as follows:

- V is the set of n galaxies, plus an artificial target node t. Let s denote Judy's home galaxy.
- E contains a directed edge $u \to v$ if either of the following conditions is satisfied
  - There is a teleport-way between galaxy u and galaxy v, and v is farther from s than u.
  - $v = t$ and there is a teleportway between galaxy u and s.
- We need to compute the length of the longest path in G from s to t.
- We can compute this length using dynamic programming as described in Thursday's lecture (and in the lecture notes).
- The algorithm runs in $O(V + E) = \boldsymbol{O(n + m)}$ ***time.*** ■

To think about later:

3. Just before embarking on her universal tour, Judy wins the space lottery, giving her just enough money to afford *two* teleports toward her home galaxy. Describe and analyze a new algorithm to compute the maximum number of galaxies Judy can visit; if she visits the same galaxy twice, that counts as two visits. After all, argues the travel agent, who can see an entire galaxy in just one visit?

Solution: We reduce this problem to finding the length of the longest path in a dag $G = (V, E)$ as follows. Let s denote Judy's home galaxy.

- V consists of $2n + 1$ vertices: Two vertices $(g, 1)$ and $(g, 2)$ for each galaxy g, plus an artificial target vertex $(s, 3)$. Vertex (g, i) represents the ith visit to galaxy g.
- E contains a directed edge $(g, i) \to (h, j)$ if either of the following conditions is satisfied:
  - There is a teleport-way from galaxy g to galaxy h, and h is farther from s than g, and $j = i$.
  - There is a teleport-way from galaxy g to galaxy h, and h is closer to s than g, and $j = i + 1$.
- We need to compute the length of the longest path in G from $(s, 1)$ to $(s, 3)$.
- Because G is a dag, we can compute this length using dynamic programming as described in Thursday's lecture (and in the lecture notes).
- The algorithm runs in $O(V + E) = \boldsymbol{O(n + m)}$ ***time.***

■

*4. Judy replies angrily to the travel agent that *she* can see an entire galaxy in just one visit, because 99% of every galaxy is exactly the same glowing balls of plasma and lifeless chunks of rock and McDonalds and Starbucks and prefab "Irish" pubs and overpriced souvenir shops and Peruvian street-corner musicians as every other galaxy.

Describe and analyze an algorithm to compute the maximum number of *distinct* galaxies Judy can visit. She is still *allowed* to visit the same galaxy more than once, but only the first visit counts toward her total.

Solution: This problem is quite a bit more ~~difficult~~ interesting than it looks!

Number the vertices from 0 to $n - 1$ in increasing order of distance from Judy's home galaxy. (In particular, Judy's home galaxy is galaxy 0.) Define a directed graph $G = (V, E)$ as follows:

- $V = \{0, 1, \ldots, n\}$, where n is an artificial target node.
- $E = \{i \to j \mid i < j \text{ and Judy can teleport from } i \text{ to } j\} \cup \{i \to n \mid 0 \to i \in E\}$.

Intuitively, we imagine two clones of Judy moving outward through the graph, one from node 0 to some node $k < n$, and the other moving outward from some node $j \geq 0$ to node n, where galaxies j and k are connected by a teleport way.

For all indices i, j, and k, let $MaxGalaxies(i, j, k)$ denote the maximum number of nodes touched by two directed paths in G, one from node i to node k, the other from node j to node n. We need to compute

$$\max\left\{MaxGalaxies(0, j, k) \;\middle|\; 0 \le j < k < n \text{ and } j{\to}k \in E\right\}.$$

The function *MaxGalaxies* obeys the following (admittedly ugly) recurrence. Intuitively, to handle collisions correctly, the recurrence tries to find the next jump from either galaxy i or galaxy j, whichever is smaller, similarly to the Erhan-and-Nancy sled-jumping problem. However, when $i = k$, then we can safely advance j even if $j > i$. Each clone gets a point for *leaving* their current galaxy, except when $i = j$.

$$MaxGalaxies(i, j, k) = \begin{cases} 1 & \text{if } i = k \text{ and } j = n \\ \max\left\{MaxGalaxies(i, j', k) \;\middle|\; j{\to}j' \in E\right\} & \text{if } i = j \\ 1 + \max\left\{MaxGalaxies(i', j, k) \;\middle|\; i{\to}i' \in E\right\} & \text{if } i < j \text{ and } i < k \\ 1 + \max\left\{MaxGalaxies(i, j', k) \;\middle|\; j{\to}j' \in E\right\} & \text{if } i = k \text{ and } j < n \\ 1 + \max\left\{MaxGalaxies(i, j', k) \;\middle|\; j{\to}j' \in E\right\} & \text{if } i > j \text{ and } j < n \end{cases}$$

An easy inductive argument implies that whenever this function is called recursively, neither clone is at a galaxy previously visited by the other, except when $i = j$. If either i or j is a sink but not that clone's final destination, the recurrence correctly returns $-\infty = \max\varnothing$.

Because the third parameter of this function never changes, we can consider each value of k independently, in an outermost loop.

For each fixed value of k, we can memoize the function into a two-dimensional array $MaxGalaxies[0..n, 0..n]$. Each entry $MaxGalaxies[i, j]$ depends on entries with larger first or second index, so we can fill the array by decreasing i and j in the inner two loops (in either nesting order).

For each entry $MaxGalaxies[i, j]$, we need to consider either all edges $i{\to}i'$ or all edges $j{\to}j'$, so conservatively, the total time to fill the array is at most

$$\sum_{i=0}^{n}\sum_{j=0}^{n} O(\deg(i) + \deg(j)),$$

where $\deg(x)$ denotes the number of edges leaving node x. This sum counts each edge $x{\to}y$ exactly $2(n+1)$ times, once for each value of i when $j = x$, and once for each value of j when $i = x$. Thus, we can fill the array in $O(mn)$ time.

Since we fill the memoization array n times, once for each possible value of k, the overall algorithm runs in $\boldsymbol{O(mn^2)}$ ***time***. ■