1. Rick has given Morty a detailed map of the Clackspire Labyrinth, which consists of a directed graph $G = (V, E)$ with non-negative edge weights (indicating distance in flinks), along with two disjoint subsets $P \subset V$ and $F \subset V$, indicating the plumbus storage units and fleebholes, respectively. Morty needs to identify a start vertex s, a plumbus storage unit $p \in P$, and a fleebhole $f \in F$, such that the shortest-path distance from p to f is at most 137 flinks long, and the length of the shortest walk $s \rightsquigarrow p \rightsquigarrow f \rightsquigarrow s$ is as short as possible.

   Describe and analyze an algo(burp)rithm to so(burp)olve Morty's problem. You can assume that it is in fact possible for Morty to succeed.

Solution: The length of a closed walk does not depend on the choice of starting vertex. Thus, we can assume without loss of generality that Morty will teleport into the Labyrinth at a plumbus storage unit p, walk along a shortest path to a fleebhole f, and then walk back to p to teleport out. In other words, we can assume that $s = p$.

We start by computing the shortest-path distances between every pair of vertices in G using the Floyd-Warshall algorithm. Then we consider each plumbus storage unit $p \in P$ and fleebhole $f \in F$ by brute force. If the distance from p to f is at most 137 flinks, then we compare the shortest walk $p \rightsquigarrow f \rightsquigarrow p$ to the best legal walk found so far.

```
InterdimensionalCable(G, P, F):
  dist[·,·] ← FloydWarshall(G)
  best ← ∞
  for all vertices p ∈ P
      for all vertices f ∈ F
          if dist[p, f] ≤ 137
              best ← min{best, dist[p, f] + dist[f, p]}
  return best
```

The overall algorithm runs in $O(V^3)$ ***time***; the running time is dominated by the initial all-pairs shortest-path computation. ■

Rubric: 10 points: 5 for reduction to all-pairs shortest paths (or lots of single-source shortest paths) + 3 for other algorithm details + 2 for running time. This is not the only correct solution. No penalty for time bounds like $O(VE \log V)$ that are near-cubic in the worst case (for example, running Dijkstra V times instead of Floyd-Warshall). No penalty for assuming $V = O(E)$.

If the input graph is sparse, and/or there are relatively few plumbus storage units and fleebholes, we can reduce the time to $O((P + F)E \log V + PF) = O(VE \log V + V^2)$ by computing shortest paths from each vertex in $P \cup F$ using Dijkstra's algorithm, instead of running Floyd-Warshall. (We can remove the final V^2 term from the running time by considering each strong component of G separately, because Morty's walk stays entirely within one strong component.)

But we can't avoid the all-pairs shortest-path computation in the worst case, because we might need the distances between every plumbus and every fleebhole. For dense graphs with lots of plumbuses and fleebs, $O(V^3)$ is (essentially) the best we can hope for.

2. You are planning a hiking trip in Jasper National Park in British Columbia over winter break. You have a complete map of the park's trails, which indicates that hikers on certain trails have a higher chance of encountering a sasquatch. All visitors to the park are required to purchase a canister of sasquatch repellent. You can safely traverse a high-risk trail segment only by *completely* using up a *full* canister of sasquatch repellent. The park rangers have helpfully installed several refilling stations around the park, where you can refill empty canisters at no cost. The canisters themselves are expensive and heavy, so you can only carry one. The trails are narrow, so each trail segment allows traffic in only one direction.

   You have converted the trail map into a directed graph $G = (V, E)$, whose vertices represent trail intersections, and whose edges represent trail segments. A subset $R \subseteq V$ of the vertices indicate the locations of the *R*epellent *R*efilling stations, and a subset $H \subseteq E$ of the edges are marked as *H*igh-risk. Each edge e is labeled with the length $\ell(e)$ of the corresponding trail segment. Your campsite appears on the map as a particular vertex $s \in V$, and the visitor center is another vertex $t \in V$.

   (a) Describe and analyze an algorithm that finds the shortest *safe* hike from your campsite s to the visitor center t. Assume there is a refill station at your campsite, and another refill station at the visitor center.

Solution: Let $G = (V, E)$ be the input graph, and let $\ell(e)$ denote the length of any edge in G. We construct a new directed graph $G' = (V', E')$ as follows:

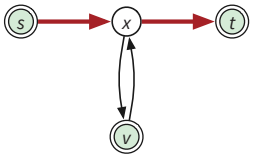
- $V' = V \times \{0, 1\}$ — Each vertex (v, i) denotes being at vertex v with i full cans of bigfoot repellent.
- E' contains four types of edges:
  - $(u, 1) \to (v, 0)$ with weight $\ell(u \to v)$ for each edge $u \to v \in H$ — high-risk trails with bigfoot repellent
  - $(u, 1) \to (v, 1)$ with weight $\ell(u \to v)$ for each edge $u \to v \notin H$ — low-risk trails with bigfoot repellent
  - $(u, 0) \to (v, 0)$ with weight $\ell(u \to v)$ for each edge $u \to v \notin H$ — low-risk trails without bigfoot repellent
  - $(v, 0) \to (v, 1)$ with weight 0 for each vertex $v \in R$ — refill stations

Paths in G' correspond exactly to safe hikes in G.

We need to find the shortest path from $(s, 1)$ to $(t, 1)$ in G'. We an answer this question using Dijkstra's algorithm in $O(E' \log V') = \boldsymbol{O(E \log V)}$ ***time***. ■

Rubric: 5 points: standard graph reduction rubric. Max 3 points for $O(VE \log V)$ or $O(V^3)$-time algorithm. Max 2 points for any slower algorithm.

Algorithms that only consider *simple paths* in the input graph G cannot be correct. Consider the the following graph, where $R = \{s, t, v\}$ and $H = \{s \to x, x \to t\}$. Every safe hike from s to t in this graph visits x at least twice; in particular, the walk $s \to x \to v \to x \to t$ is safe.



(b) Describe and analyze an algorithm to decide if you can walk safely from any refill station any other refill station. In other words, for *every* pair of vertices u and v in R, is there a safe hike from u to v?

Solution: There is a safe hike from any refill station to any other refill station if and only if (1) there is a safe hike from s to every refill station and (2) there is a safe hike from every refill station to s.

We start by building the same graph G' as in part (a). Using a single whatever-first search from $(s, 1)$, we mark all vertices that are reachable from $(s, 1)$ in G'. Then using a second whatever-first search *in the reversal of* G', we mark all vertices that can reach $(s, 1)$. Finally, we verify by brute force whether every vertex $(r, 1)$ such that $r \in R$ is marked twice.

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

Solution: First we construct the same graph G' as in part (a).

We need to determine whether there is a walk in G' from $(u, 1)$ to $(v, 1)$, for every pair of vertices $u, v \in R$. In other words, we need to determine whether every vertex $(u, 1)$ such that $u \in R$ lies in the same strong component of G'.

We can compute the strong components of G' in $O(V' + E') = O(V + E)$ time, using either of the algorithms in the textbook, after which we can check in $O(V')$ time that every refill node $(u, 1)$ lies in the same strong component.

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

Solution (4/5): Give every edge in H weight 1 and every edge in $E \setminus H$ weight 0, and then compute the shortest-path distance from every refill station to every other refill station, either by running Floyd-Warshall once or by running Dijkstra's algorithm at each vertex $r \in R$.

Now construct a new graph G' whose vertices are the refill station, with directed edges $r \to r'$ indicating that the distance from r to r' in is at most 1. Finally, if G' is strongly connected, return TRUE; otherwise, return FALSE.

Altogether, this algorithm runs in $\boldsymbol{O(V^3)}$ ***time*** if we use Floyd-Warshall, or $\boldsymbol{O(VE \log V)}$ ***time*** if we use Dijkstra. (We can get rid of the $\log V$ factor with more care, because all edge weights are either 0 or 1.) ■

Solution (3/5): Run the algorithm from part (a) for every pair $s, t \in R$. The overall running time is $O(V^2(V + E))$. ■

Rubric: 5 points: standard graph reduction rubric. I'm sure these are not the only correct solutions.

- −½ for using Dijkstra instead of WFS.
- Max 4 points for $O(VE)$ of $O(VE \log V)$ or $O(V^3)$ time (for example, consider each refill station separately)
- Max 3 points for $O(V^2E)$ time (for example, consider each pair of refill stations separately)
- No penalty for assuming $V = O(E)$.