Problem Set 4

All parts are due on November 15, 2016 at 11:59PM. Please download the .zip archive for this problem set. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A

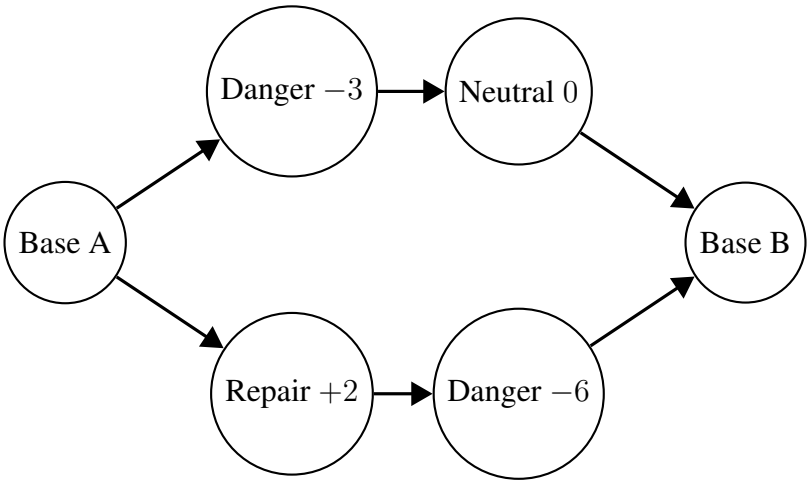
Problem 4-1. [30 points] **Runaround**

Powell and Donovan were sent to Mercury with the latest robot model, SPD-13 (Speedy) to revitalize an abandoned mining station. After several near disasters following the rules of Prof. Asimov, they have learned to be more careful when sending Speedy on missions to obtain selenium from other bases. Speedy has been programmed to balance his need to follow human orders with his need to protect himself.

Speedy can travel through Mercury along a certain network of established, directed roads. At the intersections of these roads, there may be dangerous gases that could hurt him, nothing, or occasional repair outposts. Dangerous intersections reduce Speedy's armor integrity, repair posts increase his integrity, and neutral posts do nothing. The starting base can be assumed to be neutral.

When they need to send Speedy on a mission, Powell and Donovan reset Speedy's armor integrity to a starting integer value C. When Speedy receives an order, he leaves his starting base A immediately. If there exists a route from A to his destination base B with an integrity cost $\leq C$, Speedy will complete the route sucessfully every time. If, however, Speedy's armor integrity falls below 0, he will become unable to move, and will call for emergency repairs.

For example, consider the terrain depicted below. If $C < 3$, Speedy will get stuck at one of the danger nodes and call for emergency repairs, since the total damage he accumulates will have exceeded the integrity points he was willing to sacrifice. If $3 \leq C < 4$, Speedy will take the top route to B. Finally, if $C \geq 4$, Speedy could take either path to B, since the total damage accumulated on either path does not exceed his starting value.



(a) [3 points] View the scenario as a directed graph. Note that so far in this course, we have only discussed graphs with *edge* costs, and yet in this scenario, there appears to be *vertex* costs. How can we modify the graph such that 1) there are only edge costs, and 2) the sum of the edge costs of a path from A to any node v is equivalent to the change in armor integrity Speedy will incur if he uses this path to get to v?

(b) [10 points] Design an $O(VE)$-time algorithm to determine whether Speedy can successfully complete a given mission from Base A to Base B with a given initial armor integrity value C.

(c) [5 points] Yesterday, Powell and Donovan remotely assigned Speedy a mission to return to home base from another outpost, but Speedy has neither made it back sucessfully, nor has he called for repairs. Speedy would have returned to base if he could have. Furthermore, if at any point Speedy's armor integrity dropped below 0, Speedy would have called for emergency repairs. Since neither of these things happened, what happened to Speedy? Give an example of a graph in which this situation could occur.

Hint: Read "Runaround" from Asimov's *I Robot*.

(d) [12 points] Thanks to your answer in part (c), Powell and Donovan have some idea of what could have happened to Speedy. Now they just need to find him. But they don't even know where Speedy started his trip! Design an $O(VE)$-time algorithm that outputs exactly the set of vertices where Speedy could be.

Problem 4-2. [25 points] **It's Hanna Barbera Time!**

Jerry wants to get to the fridge f from his starting location s, without being caught by Tom. Unfortunately, Jerry is a small mouse, and can only travel along a very specific graph of edges in the kitchen. Each of these edges e costs a certain amount of effort, $w(e) > 0$. A subset $H \subseteq V$ of the vertices in this graph make good hiding places, where Tom cannot see Jerry. Other vertices (in $V \setminus H$) leave Jerry exposed. Tom, meanwhile is patrolling the house, and glances into the kitchen once every three minutes. It takes Jerry a single minute to traverse any edge in the graph. To avoid

Tom's watchful eyes, Jerry must be at a hiding spot in H on every third minute. Thankfully, the fridge $f \in H$ is a hiding spot.

In other words, at start time, Jerry can traverse three edges and must wind up in a hiding spot in H after traversing these three edges. He can then traverse three more edges, and again must wind up in a hiding spot in H, and so on, until he reaches the fridge f.

(a) [10 points] First assume that the graph forms an $X \times Y$ grid graph, where $X \cdot Y = |V|$, each vertex has coordinates (x, y) where $1 \leq x \leq X$ and $1 \leq y \leq Y$, and edges connect all horizontally and vertically adjacent vertices. Further assume that the starting point s is in the southwest corner, $(0, 0)$, and fridge f is in the northeast corner of the kitchen, (X, Y). At each minute, Jerry may only go one step north or one step east. Each such traversal requires a positive effort given by edge-weight function w. Jerry may not wait in any location. How can Jerry find the least-effort path from his starting location s to the fridge f, whilst also not getting caught by Tom? Your algorithm should run in $O(V + E)$ time.

(b) [15 points] Now consider the general case of an arbitrary graph $G = (V, E, w)$ and vertices $s, f \in V$. In addition, suppose that Jerry can now also wait at a vertex for as many minutes as he wants. (In other words, Jerry can now traverse at most three edges between safe hiding places.) Help Jerry find the least-effort path to the fridge f from his starting location s, whilst still not getting caught by Tom. Your algorithm should run in $O(V \log V + E)$ time.

Hint: Formulate the problem as a graph problem, but not with the obvious graph. What is relevant in this problem is not just Jerry's location, but the pair of Jerry's location and how many minutes are left before Tom glances over.

Problem 4-3. [15 points] **Rubinfeld's Cube**

You are given an undirected graph $G = (V, E)$ in which each vertex represents a unique configuration of an $r \times r \times r$ Rubik's cube, and each edge represents a ***move***, that is, a 90° rotation of one slab of the cube. (The graph is given to you, so you don't need to worry about the details of moves in Rubik's cubes.)

(a) [5 points] Given a specific configuration $c \in V$, design an $O(V + E)$-time algorithm that finds a minimum-length sequence of configurations that solves the cube.

(b) [10 points] Design an $O(VE)$-time algorithm to preprocess G that will then allow you, given two configurations c_1 and c_2, to output in $O(k)$ time a minimum-length sequence of configurations that transforms c_1 into c_2, where k is the length of this shortest sequence.

Part B

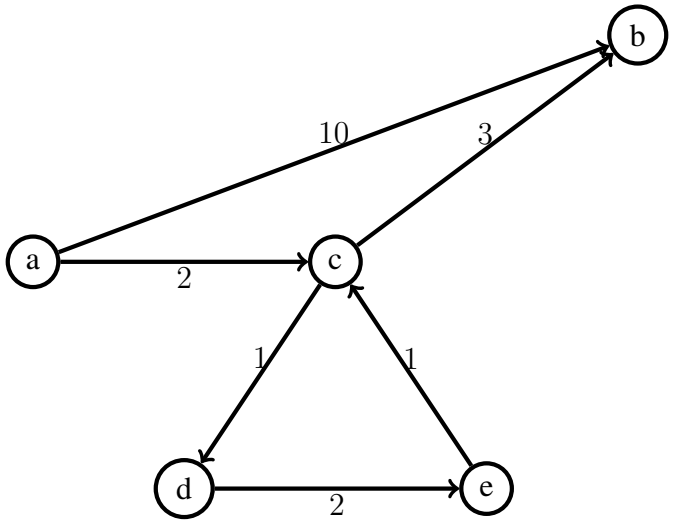
Problem 4-4. [30 points] k**th shortest paths**

You would like to estimate the latency of search queries to the new 6006LE search engine that is running over a distributed network of n servers. Each server has a direct connection to some subset of servers in the network; these connections are represented by the edges E of a given graph $G = (V, E)$ where $V = \{0, 1, \ldots, n-1\}$ is the set of n servers. You are also given the latency $L(u, v)$ for each server connection $(u, v) \in E$. Note that the links in our network may be asymmetric, so $L(u, v) \neq L(v, u)$. Computing the latency from a server u to a server v corresponds to computing the shortest path from u to v in G with respect to the latency function L.

(a) [10 points] Implement the Floyd-Warshall algorithm to solve this all-pairs shortest-paths problem in the `latencies(N, L)` method. Your inputs will be N, the number of servers (or vertices) in the network, and the latency function L, which takes two servers u, v and returns the real-valued, positive latency of the connection from server u to server v. If there is no direct connection from u to v (i.e., $(u, v) \notin E$), then $L(u, v)$ will return $+\infty$. You should return an $N \times N$ matrix A such that A_{ij} is the latency from server i to server j.

Unfortunately, you realize that the 6006LE network can be unreliable, and that the routing algorithm that decides where to forward each packet can sometimes make mistakes. You decide to make a more conservative estimate of the latencies between servers, by computing not the shortest path, but the *second-shortest* path between each pair of servers u and v. We define the ***second-shortest path*** from u to v to be the path of minimum latency among all paths from u to v that are distinct from the shortest path. This definition implies that, if there are two paths from u to v of minimal latency, then both the shortest path and the second shortest path have the same latency. Also recall that paths can have repeated vertices, so the second shortest path may contain a cycle.

For example, consider the network of servers shown below.



In this case, the shortest path from a to b is $a \to c \to b$, with latency 5, and the second shortest path from a to b is $a \to c \to d \to e \to c \to b$, with latency 9. Furthermore, the shortest path from a to c is $a \to c$, with latency 2, and the second shortest path from a to c is $a \to c \to d \to e \to c$, with latency 6.

(b) [20 points] Implement an algorithm to solve this all-pairs second-shortest-paths problem in the `conservative_latencies(N,L)` method, with the same inputs as the method from the previous part. This time, you should return an $N \times N$ matrix B, such that B_{ij} is the latency of the second shortest path from server i to server j. Your algorithm should run in $O(N^3)$ time.

Hint: This is a pretty tricky algorithm to get right, so we'll walk you through the general steps. First modify your Floyd-Warshall algorithm from the previous part to maintain two matrices, A and B, with A storing the solutions to all of the subproblems of shortest paths, and B storing the solutions to all of the subproblems of second-shortest paths. In each iteration of the algorithm, for each pair of servers i and j, you should now update both the shortest and the second shortest path from i to j among all paths which use only the first k vertices. What is the correct relaxation recurrence when calculating $B(i, j, k)$ in general? What about when either $k = i$ or $k = j$? How will this algorithm handle the case of multiple shortest paths?

Hint: Just modifying the relaxation recurrence might not detect whether there exists a second-shortest path which is simply the shortest path plus a cycle around one of the vertices on this path. First, modify your Floyd-Warshall implementation to store the computed shortest path from server i to server j, for each i and j, in addition to the latencies. Then, after you have completed your Floyd-Warshall pass, you will need to make sure that you've actually found the second shortest path. Specifically, for each pair (i, j), you will need to iterate through each vertex v on the computed shortest path from i to j, and check whether the shortest-path length plus the shortest nontrivial cycle around v is less than the computed second shortest path from i to j. How will you know the length of the shortest nontrivial cycle around v?

(c) [10 points] **(Extra credit, no collaboration)** Design and analyze an efficient algorithm for solving the all-pairs kth-shortest-path problem for general k.

Note that no collaboration is allowed for this extra credit part.