Problem 5-1. Lair

As a new evil overlord who’s had a stroke of luck, you’re now in the market for building your very own secret lair. One important aspect is your treasure chamber. You’ve purchased an archipelago \( I \) of islands and decided that you will scatter your treasure across some subset \( S \subseteq I \) of these islands. You decide to build a set \( B \) of bridges between pairs of islands, each of which will collapse after a single traversal. (You also plan to set up an advanced system of lasers that will shoot down any boats or planes which approach the islands, thus necessitating the use of your devious collapsing bridges.) You want to design your bridge system to guarantee that, in case of emergency, you can get your treasure back off the islands in \( S \) to a single home island \( h \). Treasure is very heavy, so to do so, you would need to follow a path \( p_{i,1} \) from \( h \) to one island \( i \) in \( S \), then follow a second path \( p_{i,2} \) to return to \( h \); then follow a path \( p_{i’,1} \) to another island \( i’ \) in \( S \), and a return path \( p_{i’,2} \); and so on.

(a) Give an algorithm to determine whether there are paths \( p_{i,1}, p_{i,2} \) from \( h \) to every island \( i \) in \( S \) such that all \( 2|S| \) paths \( p_{i,j} \) share no bridges, so they can all be traversed in turn without ever traversing a collapsed bridge. Your algorithm should run in \( O(IB^2) \) time.

Solution: We can find disjoint paths using Max-Flow with the following construction. First, convert all islands into nodes and all bridges into two directed edges, each of capacity 1. Next connect all the treasure containing islands to one super-source with capacity two edges. We then check to see if the graph has a max-flow equal to \( 2|S| \). Intuitively, if the max-flow is a certain value, it must have found ways to push that many units of flow from the source to the sink. The capacity two constraints to each treasure holding island ensure each of them has two paths to the home island. We now need to prove that if there is a solution to our island and bridge problem, we will find a max-flow of \( 2|S| \), and if we find this max-flow, it corresponds to a solution to our original problem.

It is fairly simple to see why our bridge constraint admits the requested max-flow. Simply take every path from the islands to the home as a flow path. Since these are all disjoint from each other, saturating them will not interfere with the flow we can send through another path. Since there are two from every island, the capacity two edges from the super-source are saturated and we never need more capacity to saturate the paths.
Proving that we have the disjoint paths is a little harder. First, from the integral theorem, we know that all the flows are either 1 or 0. Second, we note that although we doubled the number of edges (potentially allowing flow forward and backward along each bridge) by the conservation theorem in lecture, we can always cancel these out and find a new flow of the same value that uses zero of the edges rather than both. We will now prove this is true by backward induction. Assume that when there is a flow of $k$, then the set of edges with flow equal to 1 contains $k$ disjoint paths. We now find an edge with flow 1 from $s$ to some vertex $u_1$. We then go from $u_1$ to another vertex $u_2$ and continue until we either create a cycle or reach $t$. If we create a cycle, we can remove the cycle from the flow without reducing the max-flow of the system. If we reach $t$ then we have found a path from $s$ to $t$. If we now consider the graph with this path removed, it now has a max-flow of $k - 1$ and contains $k - 1$ disjoint paths. Our base case of $k = 1$ can be easily seen to be true, proving the induction.

The construction has $O(I)$ vertices and $O(B + |S|)$ edges. However, if you throw out the parts of the graph not connected to $h$, then you only have $O(B)$ edges, leading to a running time of $O(IB^2)$.

(b) Building all of those bridges is going to be harder than you thought — perhaps it would be easier if they lasted longer. You notice in the most recent Evil Genius Catalog that they sell reinforced bridges which will collapse after any specified number of traversals. You decide to build several such reinforced bridges, each with a possibly different number of maximum traversals. Give an $O(IB^2)$-time algorithm to determine whether your treasure can be recovered, i.e., there are $2|S|$ paths $p_{i,1}, p_{i,2}$ from $h$ to every island $i$ in $S$ such that the paths can all be traversed in turn without ever traversing a collapsed bridge.

**Solution:** This is still solved by Max-Flow, although we need to augment the graph so the edge capacity corresponds to the number of times the bridge can be traversed. The proof of correctness either follows from a simple augmentation to the induction in part a) or by deconstruction of higher-capacity into multiple edges.

(c) The catalog also has an advertisement for a bridge stabilizing field, which uses a magnetic field to stabilize a single specified bridge for one additional traversal (i.e., increases its maximum traversal count by 1). You don’t want to have to re-run your algorithm from (b) every time you consider adding a bridge stabilizing field. Give an $O(I + B)$-time algorithm to update the answer to whether your treasure can be recovered after adding a single bridge stabilizing field to a specified bridge.

**Solution:** Use the max-flow found for the previous graph on the new graph. Constructing the updated graph and it’s residual after assuming the old max-flow takes $O(I + B)$ time. Now search for an augmenting path which contains the updated edge. Performing this BFS also takes $O(B)$ time, giving the required running time.
(d) You become ecstatic when you learn about island self-destruct systems, which cause an island to self-destruct after a certain number of visits to the island. With this newly found tool, you endeavor to design an even better treasure chamber. First, however, you need to update your $O(IB^2)$-time algorithm for detecting treasure recovery to take into account these exploding islands.

**Solution:** This can be solved with the same method as part b) with a small adaption to represent the vertex capacities. This is achieved by splitting every vertex into two vertices, $v_{in}$ and $v_{out}$ which will be connected to represent incident edges and outgoing edges. A single edge is added from $v_{in}$ to $v_{out}$ with capacity equal to the vertex capacity.

This increases the number of vertices to $O(2I)$ and the number of edges to $O(I + B^2)$, so the running time is $O(I(I^2 + B^2))$. However, note the following: if there are no edges incident to an island, then that island can never be visited, so you do not need to split it into two vertices. This guarantees that the number of edges in the new graph is $O(2B)$, so the running time is $O(IB^2)$ as desired.

**Problem 5-2. Updating MST**

Assume that you have already computed a minimum spanning tree $T$ for an undirected, connected graph $G = (V, E)$, where each edge $e \in E$ has a nonnegative weight $w(e)$. Let $u$ and $v$ be two nonadjacent vertices in $V$. Now a new edge $(u, v)$ with weight $w$ is added to $G$.

(a) Give an $O(V)$-time algorithm to determine whether $T$ remains a minimum spanning tree in the new graph. You must specify the data structure used to represent the tree $T$.

**Solution:** Given a minimum spanning tree $T$ of $G$, we claim that if we add an edge $e$ to $G$, the minimum spanning tree of the new graph has the same weight as $T$ if and only if $w(e)$ is not less than the weight of any edge (from $T$) in the cycle formed when $e$ is added to $T$. The only if direction is obvious—if $e$ has smaller weight than some edge in the cycle, we take that edge out of $T$ and add in $e$. The resulting tree is has $n$ vertices, i.e., a spanning tree, with a smaller weight.

For the if direction, we consider a partition of $T$ into two components $U_1$ and $U_2$ such that $u \in U_1$ and $v \in U_2$, where $e = (u, v)$. Consider the cycle in $T$ containing $e$ when $e$ is added to $T$. There must be an edge $e'$ in that cycle connecting some vertex in $U_1$ and some other vertex in $U_2$, and $w(e') \leq w(e)$ by our assumption. By the greedy-choice property of MST, replacing $e'$ with $e$ cannot produce a smaller MST. So $T$ is still a MST in the new graph.

So we only need to identify the maximum edge weight in the unique path from $u$ to $v$ in $T$ and compare this weight to $w(e)$. We can represent $T$ with an adjacency list, carry out BFS/DFS on $T$ to find the $u - v$ path, and keep track of the maximum edge weight along the path. This takes $O(E + V) = O(V)$ time, as $|E| = O(V)$ for a tree.
(b) Suppose that $T$ is no longer a minimum spanning tree in the new graph. Give an
algorithm that updates $T$ to the new minimum spanning tree in $O(V)$ time.

**Solution:** If our algorithm from part (a) reports that $T$ is no longer minimum, then
we can update $T$ by adding $e$ and removing the edge of maximum edge weight in the
$u − v$ path. This directly gives us a $O(V)$ algorithm.

Let $e' = (u', v')$ be the removed edge. Consider a different partition of $T$ into two
components $U_1$ and $U_2$ such that $u' ∈ U_1$ and $v' ∈ U_2$. Note that since $e$ is on a cycle
containing $e'$, the end points of $e$ must be in different parts of this partition. By the
optimal substructure property of MST, we know that the connected subsets of $T$ that
span $U_1$ and $U_2$ are the MST’s of the induced sub-graphs on $U_1$ and $U_2$ (both in the
old graph and the new graph), respectively. By the greedy-property, $e'$ must have been
the minimum-weight edge connecting $U_1$ and $U_2$ in the old graph. Now we replace it
with an even better edge $e$, so $e$ must be the minimum-weight edge connecting $U_1$ and
$U_2$ in the new graph. So the new tree is a MST of the new graph.

**Problem 5-3. Difference Constraints**

Prove that the Bellman-Ford algorithm can be used to solve the following type of problem over
variables $x_1, x_2, \ldots, x_n$, or report that there is no solution, in $O(nm)$ time:

$$
\text{minimize } \max_i x_i - \min_i x_i \\
\text{subject to } m \text{ constraints of the form } x_i - x_j \leq c_{i,j} \\
\text{(where each } c_{i,j} \text{ is a specified, possibly negative constant)}
$$

**Solution:** Build a graph with a node $i$ corresponding to each variable, and put an edge from $j$ to $i$
of weight $c_{ij}$. What do paths in this graph mean? If we have a path from $i_1$ to $i_2$, etc. to $i_k$, and the
total length of this path is $c$, then that means our system has the following set of equations:

$$
-x_{i_j} + x_{i_{j+1}} \leq c_{i_j, i_{j+1}}
$$

And the $c_{ij}$ satisfy:

$$
\sum_{j=1}^{k} c_{i_j, i_{j+1}} = c
$$

So if we sum all $k - 1$ equations, we get that our system necessarily satisfies:

$$
-x_{i_1} + x_{i_k} \leq c
$$

Which can be rearranged to yield:
\[ x_{i_1} - x_{i_k} \geq -c \]

In other words, if we have a large negative path from \( i_1 \) to \( i_k \), then the discrepancy is necessarily large between \( x_{i_1} \) and \( x_{i_k} \) in any feasible solution. That means that every negative path between two nodes of length \( c \) implies that the best possible discrepancy is at least \(-c\). Similarly, if we have a cycle in this graph, then \( i_k = i_1 \), so when we sum the equations we would get:

\[ 0 \leq c \]

In other words, if there is any negative cycle in this graph, then no feasible solution exists. So it seems like a good idea would be to try and find the most negative path in this graph, and whether or not there exists a negative cycle, and somehow use this information to obtain a good solution. To do this, we can use Bellman-Ford, but we have to make a small modification to the graph first, by adding a source \( s \) with an edge of weight 0 to every other node (and no incoming edges). Now run Bellman-Ford on this updated graph to find the shortest path from \( s \) to \( i \) for all \( i \) in time \( O(nm) \).

First, it is clear that if there is a negative cycle in the original graph, it is reachable from \( s \) in the new graph. So Bellman-Ford will detect the negative cycle, and we can output that no feasible solution exists. If no negative cycle is detected, then Bellman-Ford outputs a list \( D_1, \ldots, D_n \) of the shortest path from \( s \) to \( i \) for all \( i \) in the modified graph. Let’s check this as a candidate solution to the discrepancy problem.

We first need to make sure that the \( D_i \) are a feasible assignment. Because there is an edge in the graph from \( i \) to \( j \) of weight \( c_{ij} \), one valid path from \( s \) to \( j \) would be to first go from \( s \) to \( i \), and then take this edge to \( j \). Therefore, we see immediately that the distances necessarily satisfy \( D_j \leq D_i + c_{ij} \), which can be rewritten as \( D_j - D_i \leq c_{ij} \), for all \( i, j \), so the solution is feasible.

Next, it is clear that all \( D_i \leq 0 \) because there is an edge from \( s \) to \( i \) of weight 0. Finally, if any \( D_i = c \), then there is a path from \( s \) to \( i \) that uses the edge \((s, j)\) for some node \( j \). Because the weight of the edge from \( s \) to \( j \) is 0, this necessarily means that there is a path from \( j \) to \( i \) of weight \( c \). Therefore, if \( c \) denotes the weight of the most negative path in this graph (between any two nodes), we have \( c \leq D_i \leq 0 \) for all \( i \), and this solution has discrepancy \(-c\). Our work above shows that the minimum possible discrepancy is at least \(-c\), so therefore this is the best possible discrepancy, and the solution is optimal.