H5.

- The assignment is due on Gradescope on Friday, February 26 at 5pm.
- You can either type your homework using LaTeX or scan your handwritten work. We will provide a LaTeX template for each homework. If you writing by hand, please fill in the solutions in this template, inserting additional sheets as necessary. This will facilitate the grading.
- You are permitted to discuss the problems with up to 2 other students in the class (per problem); however, *you must write up your own solutions, in your own words*. Do not submit anything you cannot explain. If you do collaborate with any of the other students on any problem, please do list all your collaborators in your submission for each problem.
- Similarly, please list any other source you have used for each problem, including other textbooks or websites.
- *Show your work.* Answers without justification will be given little credit.

Problem 1 (35 points). NASA's Perseverance Rover has just landed on Mars! You are the scientist now in charge of navigating the rover as it visits locations on the Martian surface to perform experiments. The region that the rover has landed in is modeled as a topographical map represented by an $N \times N$ grid. For each point (i, j), there is an associated **height** $h_{i,j} \geq 0$ and **scientific value** $v_{i,j} \geq 0$ for visiting the location.

Because we want the rover to run as long as possible, you are also required to conserve energy during navigation and are subsequently *forbidden from moving the rover uphill*. The Rover's movements can thus be summarized as follows.

- The rover starts at position $\left(\frac{N}{2}, \frac{N}{2}\right)$
- The rover is allowed to navigate from positions (i, j) to $\{(i+1, j), (i-1, j), (i, j+1), (i, j-1)\}$; the rover must stay on the grid!
- The rover can move from (i, j) to (i', j') if and only if $h_{i',j'} \leq h_{i,j}$; the rover cannot climb uphill.
- Image surveys determine that nothing outside the grid is reachable – you may treat this boundary as having $h_{ij} = +\infty$.
- The rover can revisit locations provided the above are satisfied, but a location's scientific value can only be collected once.

Subject to these constraints, construct an algorithm that outputs a path, represented as a sequence of (i, j), that maximizes the sum of scientific values in Perseverance's journey. Prove that your algorithm is correct, and that it runs in polynomial time with respect to N. In your running time analysis, provide an explicit $O(\cdot)$ bound.

Solution: Your solution goes here.

Lewis Arnsten, Collaborator: Brian McManus

I apologize for this homeworks lateness. I have had COVID all week and symptoms unexpectedly impact my ability to work. I thought I would be able to finish in time, however I underestimated the impact of COVID on my ability to do quality work.

Problem 1:
To devise an algorithm that outputs the path that maximizes the sum of the scientific values in Perseverance's journey, we must first find the sum of scientific values for the path of maximum achievable scientific value for each position. For a given position, this is done by finding the neighboring position of greater or equal height whose path of maximum scientific value achieves the largest sum, and adding this sum to the scientific value of the current position. To ensure that, for a given point, we have found the optimal values for all neighboring positions of greater or equal height we must work in descending height order.

Main Algorithm:
The first part of our algorithm will output a two-dimensional array, where each position k has a value OPT_k equal to the sum of the scientific values in the path to that position achieving maximum scientific value. To do so, we will maintain a queue (descending order) of the heights of all neighbors for the points we have looked at (excluding those that are unreachable). Thus, if we are at position k and have calculated some OPT_k, in order to find position $k + 1$, we will look at the available neighbor of largest reachable height (top of queue). This will ensure that when calculating the OPT_{k+1}, we know the optimal values for all the neighbors to position $k + 1$ with heights greater than or equal to that of position $k + 1$. As stated, OPT_{k+1} will then be calculated by finding the neighboring position k with maximal value OPT_k, and computing $OPT_{k+1} = OPT_k + V_{k+1}$.

```
h → N by N heights matrix
v → N by N scientific values matrix
queue = []
max_v = []
max_v[0] = [n/2,n/2,0]
//first and second elements are i and j respectively
//third element is maximum possible sum of scientific value to reach this position
For p = 0, to length(max_v):
        i = max_v[p][0]
        j = max_v[p][1]
        neighbors = {(i+1,j),(i−1,j),(i,j+1),(i,j−1)}
        heights = {h(i+1,j),h(i−1,j),h(i,j+1),h(i,j−1)}
```

```
        For height in heights:
                If height <= h(i,j):
                        append corresponding element in neighbors to queue ordered by height
        next_neighbor = top of queue
        max_v[p+1][0] = next_neighbor[0]  //i of element at the top of queue
        max_v[p+1][1] = next_neighbor[1]  //j of element at the top of queue
        max_v[p+1][2] = max(max_v[p][2] where max_v[p] is a neighbor to max_v[p+1]) +
        v[next_neighbor]
        remove next_neighbor from top of queue
Return max_v
```

Correctness: When considering the k^{th} position, we define the sum of the scientific values in the path of maximum achievable scientific value as OPT_k. Thus, when looking at the *k+1* position, we can define $OPT_{k+1} = V_{k+1} + OPT_k$. Given our algorithm's design it will always be true that the height of position *k* is greater than or equal to that of position *k+1*. This algorithm ensures that we are maximizing our scientific value when moving to space *k+1*, as given all possible neighbor locations, we are choosing to move from the location that achieved a path of maximum scientific value. Thus, we can use induction to show that OPT_n = max_v[n] = maximum sum of scientific values in journey to each point 1,...,n.

<u>Secondary algorithm:</u>

However, we are not done. We have found the maximum possible sum of scientific value to reach each point and must now find the path that maximizes the sum of the scientific values. To do so we can sort our list max_v by the third element of each sublist (maximum possible sum of scientific value to reach this position). Then we loop backwards over the list, starting with the last element, and adding elements to the end of our path so long as they are neighbors. Since we built our list using the neighboring location that achieved a path of maximum scientific value, working backwards from the point whose path achieved maximum scientific value overall will necessarily return our desired result.

```
sorted_reversed = max_v sorted by the third element of each sublist, ordered biggest to smallest
Path = [(sorted_reversed[0][0],sorted_reversed[0][1])]
For item in sorted_reversed:
        If last element in path is neighbor of item:
                Append item to path
Reverse order of path
Return path
```

Time complexity: The first part of this algorithm must check 4 possible neighbors for at most N*N points in our grid. The second half of our algorithm must check all N*N items in the set created in the first part. Thus, although the exact complexity of this algorithm is hard to determine, a sufficient upper bound is O(n^4)

PROBLEM 2 (30 points). In this exercise, we explore the effect of perturbations on the capacity of a single edge on the value of the maximum flow in a flow network with integral capacities. Consider the following types of edges in a flow network.

- An edge of a flow network is called **critical** if decreasing the capacity of this edge results in a decrease in the maximum flow.
- An edge of a flow network is called a **bottleneck** edge if increasing its capacity results in an increase in the maximum flow.

Given a connected flow network $G = (V, E)$ with integer capacities $c_e \geq 0$, prove or provide a counterexample for each of the following statements.

(a) All critical edges are bottleneck edges.

(b) A critical edge always exists.

(c) A bottleneck edge always exists.

Solution: Your solution goes here.

Problem 2:

a) No, all critical edges are not bottleneck edges. Counterexample: consider a graph with three nodes a,b,c. Edges (a,b) and (b,c) each have a capacity of 1. Increasing the capacity of either of these edges will not change the maximum flow as it will remain 1. However, decreasing the capacity of either edge will decrease maximum flow. Thus, these edges are critical edges and not bottleneck edges.

b) Yes, an edge belonging to the minimum cut must be a critical edge as decreasing its capacity necessarily decreases the minimum cut--and thus also the max flow.

c) No; we can show that a bottleneck edge does not always exist using the same counterexample used in part a. Consider the graph with nodes a,b,c and edges (a,b) (b,c), each with a capacity of 1. Increasing the capacity of one of these edges will not change the max flow.

PROBLEM 3 (35 points). In a particular network $G = (V, E)$ whose edges have integer capacities c_e, we have already found the maximum flow f from node s to node t. However, we now find out that one of the capacity values we used was wrong: for edge (u, v) we used c_{uv} whereas it should have been $c_{uv} - 1$. This is unfortunate because the flow f uses that particular edge at full capacity: $f_{uv} = c_{uv}$. We could redo the flow computation from scratch, but there's a faster way.

Show how a new optimal flow can be computed in $O(|V| + |E|)$ time. Precisely, construct an algorithm that given the following input, produces the following output.

- Input – A flow network $G = (V, E)$ with capacities $c_e \geq 0$ for each $e \in E$, a precomputed maximum s-t flow f, and an edge (u, v) whose capacity is incorrect.
- Output – A new maximum flow f' using capacities $c'_{uv} = c_{uv} - 1$ and $c'_e = c_e$ for all $e \neq (u, v)$.

Prove that your algorithm is correct and runs in $O(|V| + |E|)$ time. Hint: it may be helpful to consider the set of vertices reachable from u in the residual graph.

Solution: Your solution goes here.

Problem 3:

We are given a flow network G(V,E) with capacities $c_e \geq 0$ for each edge e in E, a precomputed maximum s-t flow *f*, and an edge (u,v) whose capacity is incorrect. For edge (u,v) we used capacity c_{uv} where we should have used c_{uv}-1. We now want to find a new maximum flow *f'* using capacities $c'_{uv} = c_{uv} - 1$ and $c'_e = c_e$ for all e not equal to (u,v). Additionally, we know the flow *f* uses the edge (u,v) at full capacity.

We know the flow into *u* is one unit more than the flow out of *u* and the flow into *v* is one unit less than the flow out of *v*. Thus we cannot transfer a unit flow from *u* to *v*. So we search the residual flow graph for a path P from *u* to *v* along which flow increases by one. This can be done in time O(|V| + |E|). If path P exists we will update the new maximum flow *f'* = (*f* updated to include this new flow along path P). If no such path P exists, we must reduce the flow from *s* to *u* and from *v* to *t* by one unit.

Algorithm:

1. Let E' be the edges for which $f_e > 0$ and let G' = (V,E'). We must find a *s*-*u* path P_1 and a *v*-*t* path P_2 in G'.
2. Then we reduce flow by one unit on $P_1 \cup \{(u,v)\} \cup P_2$
3. Run Ford-Fulkerson with this starting flow

Correctness: If the original flow has size *f*, this algorithm produces a flow that has size *f*-1 and satisfies the new capacity constraint. Ford-Fulkerson will thus give us the optimal flow under this new capacity constraint.

Time complexity: Since we know this optimal flow is at most *f*, Ford-Fulkerson runs for one iteration--resulting in an upper bound of O(|V| + |E|).