Please enter your name and uID below.

Name: Jason Crandall

uID: u0726408

Submission notes

- **(PS 1 specific) This is a warm-up assignment. Full points will be given for answers that have some correct elements and that clearly been given a legitimate effort.**

- *Solutions must be typeset* using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the semester, so please retain the original files.
- Please remember that for problem sets, collaboration with other students must be limited to a high-level discussion of solution strategies. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

1.

My dynamic programming algorithm began with the knowledge that we will need to look ahead in the data in order to determine the best path. In a sense this means looking to the right in the data, and as such, the direction that I populated my solution tables starts and the far right and populating to the left. The tables themselves are the same size as the initial matrix provided, plus 1 extra column at the end initialized with zeros. The main logic of the algorithm is essentially, starting from the right, looking at the previous column, take the item in the current row, and take the smallest found item on any row in the previous column. Compare the current row item with the previous smallest item plus the switch cost (as would be the case if its on a different row), and mark which one is bigger. Finally you add the item at the current row, current column, to whichever is the smallest of the values you found in the previous step. This repeats for each row, setting the new smallest item to the smallest in the current column, then moving one column to the left. This logic is for one of the tables, the one that holds the value of the amount of time taken. The next table, which is filled in at the same time holds a string of the path taken thus far. The only difference with this table is that instead of adding the value of the previous column plus current item, we create a string with the current index concatenated with the string found at the smallest item in the previous column. With both of these methods in place it will continue to work through the columns until ending back at the very first column. Once here, the solution will be the smallest number. Whichever index that is found at, then the corresponding path string is found in the same index on the other table.

Following this logic, the run time for the algorithm would be O(sn). After all of the input is received, the calculations to solving the algorithm only walk through the entire matrix just one time. Essentially this implies that for each row s, the algorithm will walk through each column once, n. Therefore, for any case, the run time would be bounded above by O(sn).

2.

One potential greedy algorithm for PS3 consists of using the number of connections as a baseline measurement for the problem. As a starting point, we would first find the island with the most number of connections. From that island's connections, we would then begin to look for the island with the next largest amount of connections and proceed from that point. This is inevitably faulty however, as it would not provide the globally optimal solution as it excludes mutually distinct island chains. For example, let's say that island 1 connects to islands 2 and 3, and island 4 connects to islands 5 and 6. If we follow the island 1 chain, we will be able to see that island 1 is able to reach 2, and that 2 and 3 do not connect to anything else, but the solution would completely miss on including the island 4 chain and thus remains incomplete.

Another attempt at a greedy algorithm would be to, similarly to the last one, sort the islands based on the number of connections they have. In this sense, the island that has the most connections would appear first in the list, and the island that would have the lowest or no connections would appear last. From this stand point we can then walk the array and mark the island with the most connections, then choose the next island in the array that is not included in the connections of the first island found. For example, if island 1 has connections to 2 and 3, then the next island that would be searched after 1 is island 4 if it has the next greatest amount of connections. This would continue through the end of the array. This is also a flawed approach given the following example: Let there be 6 islands total with the following connections

1 2
1 3
1 4
2 3
4 6
4 5

Given these parameters, the islands would be sorted in the order 1,2,3,4,5,6 as island 1 has 3 connections, islands 2 and 3 have 2 connections, and islands 5 and 6 have 1 connection. The globally optimal solution would be a shop on islands 1 and 4 as it would cover everything but the above algorithm would place a shop on islands 1, 5, and 6 because it would look at island 1 and its connections, see that island 4 is included, so it would skip that and move to island 5 then 6. Thus disproving its usefulness as a greedy solution.

3.

One attempt at a greedy algorithm for this problem would be to simply find the first positive number in the array. From this point continue adding the sequential values until the becomes negative. In doing so we attempt to maximize the sum of our current subset so that we minimize the number of subsets we must fit a global solution. We repeat this process until we have walked the remainder of the array. This algorithm does break however if the array starts with a set of negative numbers. If the sum of the initial negative numbers plus the first positive number results in a positive number, then the above algorithm misses a subset and the problem fails.

A second attempt at solving this with a greedy solution is to assume that the best solution exists, being the entire array. If this isn't the case, then we proceed to treat it as a binary search tree, splitting the array in half and summing the values in each sub-array in optimal time. This also is a faulty approach to the problem as this would only work in circumstances in which the "tree" is balanced, which is not a guarantee. Suppose we have the array: -3, +2, -5, +4. The algorithm above would first check the entirety of the array, having a total of -2 which isn't positive. Next it would divide into the two subarrays, left and right. The left array adds up to -1, and the right array adds up to -1, which neither is positive so the algorithm would continue to split, eventually see the two positive numbers marking the solution to the entire array as 2.. In reality however, we can see that the subset exists: +2, -5, +4 which adds up to 1 which is positive so the actual globally correct solution is in fact 1, not 2.

Finally, a third attempt at solving this problem greedily is to find the global negative value of the array. Once found, proceed right until the result becomes positive. Repeat this process with the next most negative number that wasn't in the previous subset. The idea behind this algorithm is to make the subsets as large as possible so that we can reduce the number of positive intervals. This algorithm, however, is not a proper solution. A counterexample to this is the following array: -1,+3,+4,-2,+5. Here, the greatest global negative is -2, so it would begin there until it became positive adding +5. The next most negative is -1, which would progress until adding together with +3. This would therefore provide a solution of 2. However, looking at the array, we see that the globally correct solution is 1, as +3, +4, -2, +5 add up to a positive number and is a valid subset, thus disproving this algorithm as well.