Please enter your name and uID below.

Name: Devin White

uID: u0775759

Collaborators, if any, and how you collaborated:

Submission notes

- Due at 11:59 pm (midnight) on WEDNESDAY, Nov 23rd.
- 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 grading process, so please retain these until an assignment has been returned.
- Please remember that for this problem set, you are allowed to collaborate in detail with your peers, as long as you cite them. However, you must write up your own solution, alone, from memory. 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. (Commute Times)
   The algorithm to build a new road to calculate the shortest distance is as follows:

   (a) Parse the input to get the value n

   (b) initialize a new array of nodes we will call coordinates, each node will hold an x and y value, and will be stored in coordinates in the order of parsing. For example, coordinates[i] will hold a node which contains xi, yi.

   (c) initialize a second array which will hold **all possible distances between intersections**

   (d) using a nested for loop, fill in the array of possible distances, such that distances[i,j] is the distance between intersection i and intersection j

   (e) initialize an array, which will represent the **ACTUAL** distances in our graph. initialize all values in graph to infinity using a nested for loop, then set each intersection to be 0 distance from itself(e.g. graph[i,i] = 0)

   (f) parse the currently existing roads, and populate the graph with the applicable distances already calculated in the distances array

   (g) We then use the Floyd Warshall algorithm to compute the shortest distances, and fill our graph array with these distances(e.g, graph[j,k] = result of floyd Warshall)

   (h) using a nested for loop, we iterate through each i, j value in graph and then do floyd Warshall again with the possible new roads in order to find the best possible one. This is done using a total of four nested loops. The K loop in Floyd Warshall is not needed as these values are already calculated. this ends up looking something in the end like newGraph[i,j] = min(newGraph[i,j], newGraph[i, newIntersection1] + newGraph[newIntersection2, j] + newDistances[newIntersection1, newIntersection2];

   (i) after the new graph is created, we use a nested for loop to calculate the new total sum.

   (j) if the new total sum is less than the sum before, this sum is our new total and the new road is the one we pick

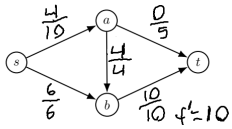
   (k) Steps h through J are done for each possible new road between every intersection. this gives us our 4x nested loop mentioned earlier

   (l) after looking at at every possible combination and picking the best one using the above, we have our correct answer

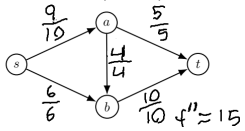
   The run time of this algorithm is approximately $O(n^4)$. We have a total of something like 7 nested for loops, 5 of these for loops are simple nested loops which run in $O(n^2)$ time, one of these loops, the one using Floyd Warshall, is a three nested loop, which runs at $O(n^3)$ and finally, the last loop, which looks at all possible new roads and selected the best one, is 4 nested loops, which runs at $O(n^4)$ $5(O(n^2) + O(n^3) + O(n^4) = O(n^4)$

2. (Feasible Flows)

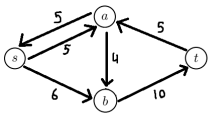
(a) **T**he flow f(s→a) = 4, f(a→t) = 0, f(a→b) = 4, f(s→b) = 6, f(b→t) = 10



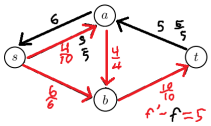
(b) **T**he flow f(s→a) = 10, f(a→t) = 5, f(a→b) = 4, f(s→b) = 6, f(b→t) = 10



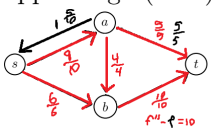
(c) **R**esidual graph gf for flow f(s→a) = (a→t) = 5:



(d) **f**or $|f'| - |f| = 5$ f' = f(s→a) = 4, f(a→b) = 4, f(s→b) = 6, f(b→t) = 10: 10 - 5 = 5. OR f' = f(s→b) = 6, f(b→t) = 6, f = f(s→a) = 1 f(a→b) = 1, f(b→t) = 1: 6 - 1 = 6



for $|f''| - |f| = 10$. f(s→a) = 9, f(a→b) = 4, f(s→b) = 6, f(b→t) = 10: 15 - 5 = 10 $|f''| - |f| = 10$. is only possible by using the upper edge (a→t) of capacity 5 and saturating the bottom edge (b→t) of capacity 10



(e) let there be an arbitrary graph, G, and two feasible glows f' and f, such that $|f'| > |f|$ and that for any f' and f $|f'| - |f|$ = some feasible flow f".Given that f' and f are feasible flows, it stands to reason by the definition of a feasible flow, that the maximum capacity of the graph cg ≥ f' ≥ f and the maximum possible flow fm ≥ f' ≥ f. as such, there then must exist a residual graph, Gf of f' where each edge e contains a forward edge e'=(u,v) with capacity ce≤ $f'e$, as well as a backwards edge e"=(v,u) with a capacity ce ≥ f'e. as $|f'| > |f|$ any edge with a valid flow for f' must also have a valid flow for f. Take, for example, given the graph of vertices vs,a,b,c,t and edges(with capacities) (s→a) = 5, (s→b) = 10,(b→a) = 2 (b→c) = 8 (c→a) = 4, (a→t) = 20, (c→t) = 5, and the feasible flows f = (s→a) = 5, (s→b) = 5, (b→c) = 5, (c→t) = 5, (a→t) = 5 = $|f|$ = 10 and f' = (s→a) = 5, (s→b) = 9, (b→c) = 8, (b→a) = 1, (c→a) = 3, (a→t) = 9 (c→t)= 5 $|f'|$ = 14, a valid feasible flow $|f'| - |f| = 4$ exists on any path where ce ≥ 4 Going further to show that this works by induction, Assume that, for the sake of contradiction that for an arbitrary graph g' where $|f'| > |f|$ and that for any f' and f $|f'| - |f|$ there **does NOT** exist a feasible flow f". Let $|f'|$ = n and $|f|$ = n-1 $|f'| - |f|$ = n-(n-1) = 1. Intuitively, using the definition that any feasible flow ≤ the capacity of the graph Cg, we can conclude that for any edge, if $|f'| \leq$ Ce and $|f| \leq$ Ce, and $|f'| \geq |f| \geq 1$ then Ce ≥ 1. This makes 1 a feasible flow, and contradicts the assumption that an arbitrary graph g' contains no feasible flows where $|f'| - |f|$ = f", in any case where —f'— > —f—> 0 and Cg != 0. Conversely, as the difference between f' and f gets larger, at most will always be f"¡ f' and at most f" >f