COMP 3711 – Honors Design and Analysis of Algorithms
2021 Spring Semester – Written Assignment # 4
Distributed April 12 2021 – April 24, 2021

Your solutions should contain (i) your name, (ii) your student ID #, and (iii) your email address

Some Notes:

- Please write clearly and briefly. Your solutions should follow the guidelines given at
  https://canvas.ust.hk/courses/36180/pages/assignment-submission-guidelines

  In particular, your solutions should be written or printed on *clean* white paper with no watermarks, i.e., student society paper is not allowed.

- Please also follow the guidelines on doing your own work and avoiding plagiarism as described on the class home page.
  You must acknowledge individuals who assisted you, or sources where you found solutions. Failure to do so will be considered plagiarism.

- The term *Documented Pseudocode* means that you must include clear comments *inside* your pseudocode.

- Many questions ask you to explain things, e.g., what an algorithm is doing, why it is correct, etc. To receive full points, the explanation must also be *understandable* as well as correct.

- Please make a *copy* of your assignment before submitting it. If we can't find your submission, we will ask you to resubmit the copy.

- Submit a SOFTCOPY of your assignment to CASS by the deadline. The softcopy should be one PDF file (no word or jpegs permitted, nor multiple files).

  If your submission is a scan of a handwritten solution, make sure that it is of high enough resolution to be easily read. At least 300dpi and possibly denser.

P1: [25pts] **MSTs**

Let $G = (V, E)$ be a weighted undirected connected graph inputted as an adjacency list. Assume that all edges have distinct weights so the *Minimum Spanning Tree* T is unique and that you have already run a MST algorithm that has found and stored T.

Now arbitrarily *increase* the weight of any one edge $e = (u, v)$ in E. You should assume that the edges in the graph still all have distinct weights, i.e, the new weight is different than the weights of all of the other edges.

Let T' be the MST of the graph with the new weights. i.e., after replacing $w(u, v)$ with its new weight $w'(u, v)$.

(a) Prove that either $T' = T$ or that T and T' differ by at most one edge.

(b) Give an $O(|E|)$ time algorithm for finding T'.

After describing your algorithm, you must then explicitly justify why your algorithm is correct and why it runs in $O(|E|)$ time.

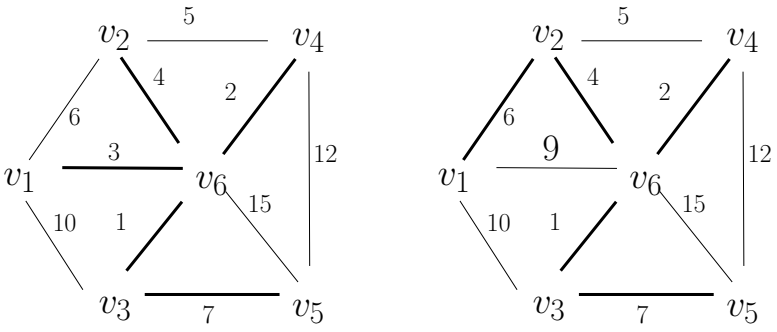
You may use any algorithm we taught in class or the tutorials as a subroutine. If you do use such an algorithm as a subroutine, you must explicitly state the name of that algorithm, its run time, why the input you are feeding into is of the correct type and the output of that algorithm.

Your description does NOT have to be in pseudocode but does have to be clear.

A "description" NOT in pseudocode would be to write something like:
(1) Read the input data I and construct a modified input I'.
(2) Then run Algorithm $\mathcal{A}$ from class on input I'.
(3) Take $\mathcal{A}$'s output and run that through Algorithm $\mathcal{B}$ from class.
...
If you do provide pseudocode, make sure to document it.

Hints, Comments and Recommendations:

- In the graph on the left, the MST is the set of 5 heavy edges. If the weight of (v_1, v_6) was increased from 3 to 5.5, the MST would stay the same. But, if the weight of (v_1, v_6) was increased from 3 to 9 the new MST would be T', the 5 heavy edges on the right, i.e., T' is T with the edge (v_1, v_2) being swapped for (v_1, v_6).



- Hint. (a) is equivalent to saying that all edges in the original MST, except possibly e, are in the new MST.

  Now consider the "cut lemma" and how it was used to prove the correctness of Prim's algorithm by showing that every edge in the tree T built by Prim's must be in ALL MSTs. How can you modify that observation to show (a) above?

  More explicitly, how can you use the ideas in the proof to show that if $e' \neq e$ was an edge in the MST before the weight of e was changed, then e' will also be in the MST after the weight is changed.

- For (b) consider again WHY the cut lemma implies that e was originally in T.

  What property did e satisfy that no other edge in the graph satisfied? If another edge e' replaces e in the MST after the weight change, what property does e' now need to satisfy?

  How can you test for the edge satisfying this property in $O(E + V)$ time?

P2: [34 pts] **Graph Algorithms**
Modified from [KT] Chapter 4.

Security analysts are trying to track the spread of an online virus in a collection of networked computers.

- There are n computers in the collection, labelled as $C_1, \ldots, C_n$.
- The input to the problem is a collection of m *trace-data* triples.
- Each triple is of the form (C_{j_i}, C_{k_i}, t_i), $i = 1, \ldots, m$ where $j_i \neq k_i$. A triple indicates that computer C_{j_i} and computer C_{k_i} exchanged bits at time t_i.
- You may assume that the triples are sorted in non-decreasing order of time, i.e., $t_i \leq t_{i+1}$.

The analysts would like to be able to answer the following type of question: *If the virus was inserted into computer C_a at time x could it possibly have infected computer C_b by time y?'*

The mechanics of infection are that: if an infected computer C_i exchanges bits with computer C_j at time t (i.e., either (C_i, C_j, t) or (C_j, C_i, t) appear in the trace data) then C_j becomes immediately infected.

This means that infection spreads from one computer to another through a sequence of communications that occur in non-decreasing order of time.

The input to your algorithm also includes the extra information that a specific computer C_a has been infected at time t_{start}

Describe an $O(m + n)$ time algorithm which determines, for every other computer, the earliest time at which it can become infected.

More specifically, your algorithm should determine the set $\mathcal{C}$ of computers that can be infected by a sequence of communications starting at C_a on or after time t_{start} and then create a directed infection tree T for $\mathcal{C}$.

- The nodes of T are the computers in $\mathcal{C}$. The root of T is C_a.
- T is represented by two arrays $P[1 \ldots n]$ and $Time[1 \ldots n]$.
- $Time[i]$ should be the earliest time that computer C_i can be infected
- Each edge $C_i \rightarrow C_j$ in the tree corresponds to one existing trace-data triple (C_i, C_j, t) or (C_j, C_i, t).
  $P[j] = i$ and $Time[j] = t$.
- Note that $P[a]$ is undefined and $Time[a] = t_{\text{start}}$.
  If $C_i \notin \mathcal{C}$ then $P[i] = Time[i] = \infty$.
- A consequence of the definitions is that $Time[i] \geq Time[P[i]]$ for all $i \neq a$.

(a) (i) Give documented pseudocode for creating T and the associated arrays.

(ii) Explain in words what your code does

(b) Prove that your algorithm is correct.

Please remember to be clear.

Put every new idea and short argument in a separate paragraph with sufficient space between the paragraphs. We can't understand what you've written if we can't read it.

(c) Explain why your algorithm runs in $O(m + n)$ time.

Hints, Comments and Recommendations:

- Worked Example: Let $C_a = C_1$ and the triples be

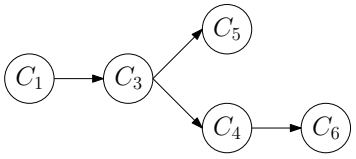
  $(C_1, C_2, 1)$, $(C_2, C_4, 4)$, $(C_1, C_3, 5)$, $(C_5, C_3, 6)$, $(C_4, C_6, 7)$, $(C_3, C_4, 7)$.

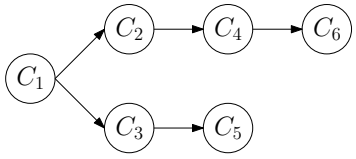
  Let $t_{\text{start}} = 3$. Then
  - C_3 becomes infected at time $t = 5$ by $(C_1, C_3, 5)$.
  - C_5 becomes infected at time $t = 6$ by $(C_1, C_3, 5)$, $(C_5, C_3, 6)$.
  - C_4 becomes infected at time $t = 7$ by $(C_1, C_3, 5)$, $(C_3, C_4, 7)$.
  - C_6 becomes infected at time $t = 7$ by $(C_1, C_3, 5)$, $(C_3, C_4, 7)$, $(C_4, C_6, 7)$.
  - C_2 does not become infected.

  But if $t_{\text{start}} = 1$, then
  - C_2 becomes infected at time $t = 1$ by $(C_1, C_2, 1)$.
  - C_4 becomes infected at time $t = 4$ by $(C_1, C_2, 1)$, $(C_2, C_4, 4)$.
  - C_3 becomes infected at time $t = 5$ by $(C_1, C_3, 5)$.
  - C_5 becomes infected at time $t = 6$ by $(C_1, C_3, 5)$, $(C_5, C_3, 6)$.
  - C_6 becomes infected at time $t = 7$ by $(C_1, C_2, 1)$, $(C_2, C_4, 4)$, $(C_4, C_6, 7)$.

  That is, in the second example, C_2 becomes infected and C_4 becomes infected earlier.

  These two examples are illustrated below.





i	$P[i]$	$Time[i]$
1		3
2	∞	∞
3	1	5
4	3	7
5	3	6
6	4	7

i	$P[i]$	$Time[i]$
1		1
2	1	1
3	1	5
4	2	4
5	3	6
6	4	7

- Note that the times might not be distinct, e.g., the last two triples in the input above have the same time stamp. All you know is that the triples are given to you sorted by non-decreasing time. The ordering of triples with identical time stamps is arbitrary.
- Infection is instantaneous. As an example, see C_4 and C_6 in the first example. They are both infected at time 7 even though C_6 is infected through C_4.

- Your algorithm must work correctly even when there are multiple triples with the same time stamp. Your proof of correctness must also deal with this case.
- Before solving the problem we recommend that you first think separately about the design of the solution to different subproblems.
  - First, try to solve the stated problem when all of the times t_i are different.
  - Second, assume that all of the t_i are the same. How would you solve the problem then?
    Hint: Consider running a modified form of BFS or DFS.
  - Now consider that you start the problem with a *collection of known computers* $C_{a_1}, C_{a_2}, \ldots, C_{a_k}$ infected at time t_{start} instead of just one computer C_a being infected. Also, assume that all of the t_i in the triples are the same.
    Modify your second algorithm to solve this new version.
  - Finally, split the time-sorted triples into *epochs,* with each epoch containing all triples with the same time stamp. Now run your third algorithm on each epoch, consecutively, one after another. After each epoch is over you have a collection of currently infected computers, that can be used to infect some computers in the next epoch.

  The fourth algorithm is the one that should solve the stated assignment problem.

  The first three problems were only defined to assist you with the *design process* for developing the final algorithm. We only want to see the final algorithm.
- The input is provided as three arrays $j[1 \ldots m]$, $k[1 \ldots m]$, $t[1 \ldots m]$, where $j[i]$ and $k[i]$ are integers between 1 and n. $t[i]$ is a nonnegative real number. The i'th input triple is $(C_{j[i]}, C_{k[i]}, t[i])$. For clarity, in your code and descriptions, you may refer to $j[i]$ as j_i, $k[i]$ as k_i and $t[i]$ as t_i.
- You may assume the existence of the following procedure. Given a list $\mathcal{L}$ of E edges on V vertices, the procedure $Build(\mathcal{L})$ can build an adjacency list representation of the graph $G = (V, E)$ in $O(|V| + |E|)$ time. You do not have to write out code for this procedure. If you do use this procedure, you need to include its running time in the running time analysis of your algorithm.
- Your code does not have to be one procedure. You may construct other procedures that are called by your main procedure.

P3: [16 pts] **BFS and DFS**

In this problem you will have to describe the breadth and depth first search trees calculated for particular graphs.

(A) Consider the following graph $G = (V, E)$ with 16 vertices $V = \{u_0, \ldots, u_7\} \cup \{v_0, \ldots, v_7\}$ and the edges E as given by the following adjacency lists:

Each node u_i in $\{u_0, \ldots, u_7\}$ is connected, in the following order, to its four neighbors: $u_{(i+1) \mod 8}$, $u_{(i-1) \mod 8}$, $v_{(i+1) \mod 8}$, $v_{(i-1) \mod 8}$.

For example:

$$u_0 : u_1 \to u_7 \to v_1 \to v_7.$$

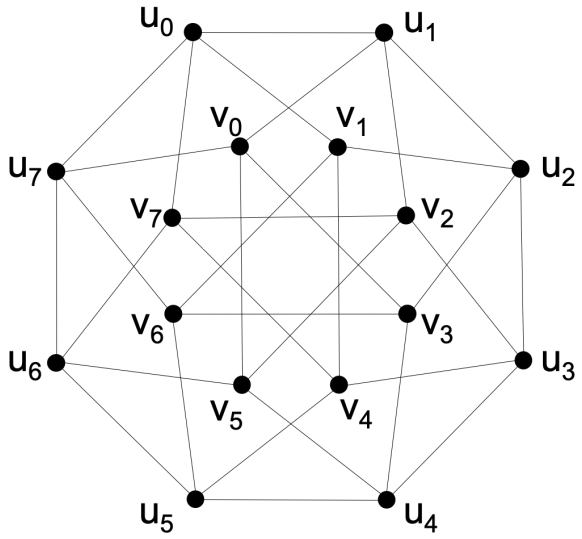
$$u_4 : u_5 \to u_3 \to v_5 \to v_3.$$

Each node v_i in $\{v_0, \ldots, v_7\}$ is connected, in the following order, to its four neighbors: $u_{(i+1) \mod 8}$, $u_{(i-1) \mod 8}$, $v_{(i+3) \mod 8}$, $v_{(i-3) \mod 8}$.

For example:

$$v_0 : u_1 \to u_7 \to v_3 \to v_5.$$

$$v_4 : u_5 \to u_3 \to v_7 \to v_1.$$



(i) Recall that BFS is *Breadth First Search*. Draw the edges of the BFS tree of G that results when starting from the node $\mathbf{u_1}$, with G represented using the adjacency lists described above.

(ii) Recall that DFS is *Depth First Search*. Draw the edges of the DFS tree of G that results when starting from the node $\mathbf{u_1}$, with G represented using the adjacency lists described above.

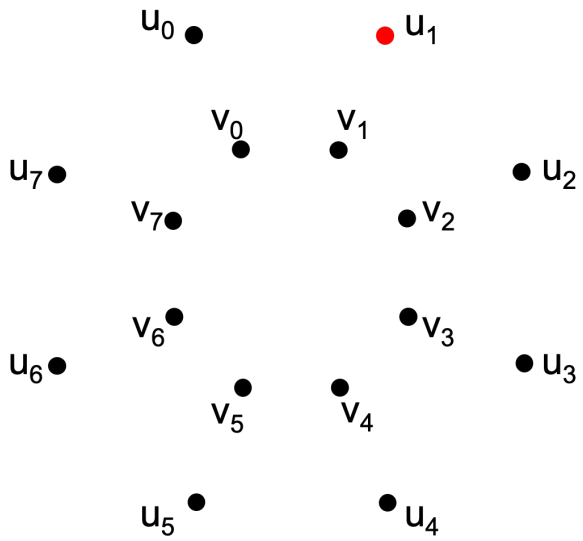
(B) Consider the same graph $G = (V, E)$ shown above and the same adjacency lists as illustrated in (A).

(i) Draw the edges of the BFS tree of G that results when starting from node $\mathbf{v_1}$.
Again, G is represented using the adjacency lists described above.

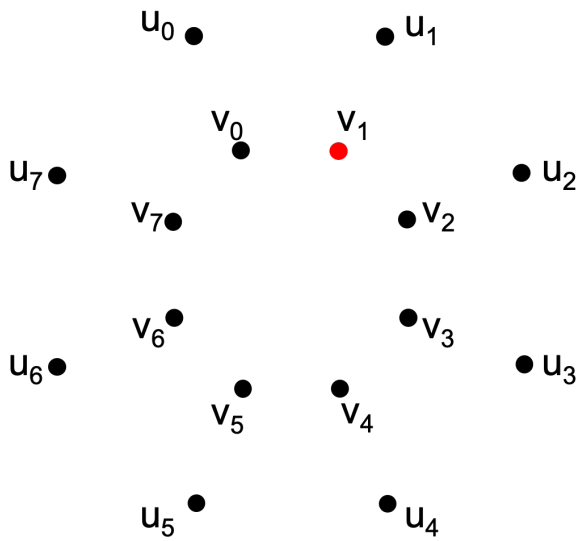
(ii) Draw the edges of the DFS tree of G that results when starting from node $\mathbf{v_1}$.
Again, G is represented using the adjacency lists described above.

Hints, Comments and Recommendations:

- For (A)(i)(ii), you can use the following template to draw the BFS and DFS tree starting on u_1. If you would like to draw your own graph, make sure that your vertices are in the same pattern as in the following template.



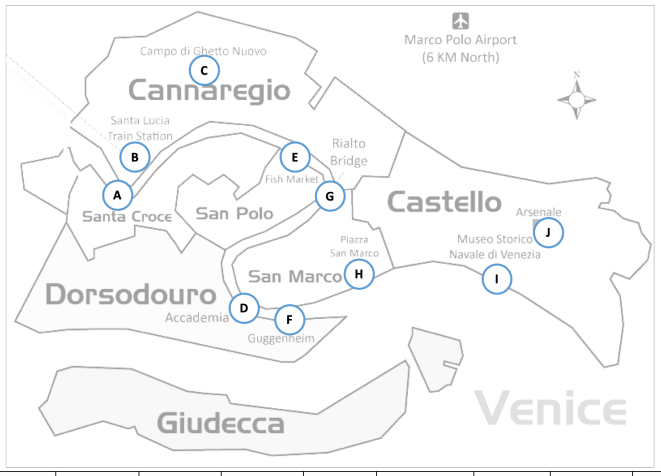
- For (B)(i)(ii), you can use the following template to draw the BFS and DFS tree starting on v_1. If you would like to draw your own graph, make sure that your vertices are in the same pattern as in the following template.



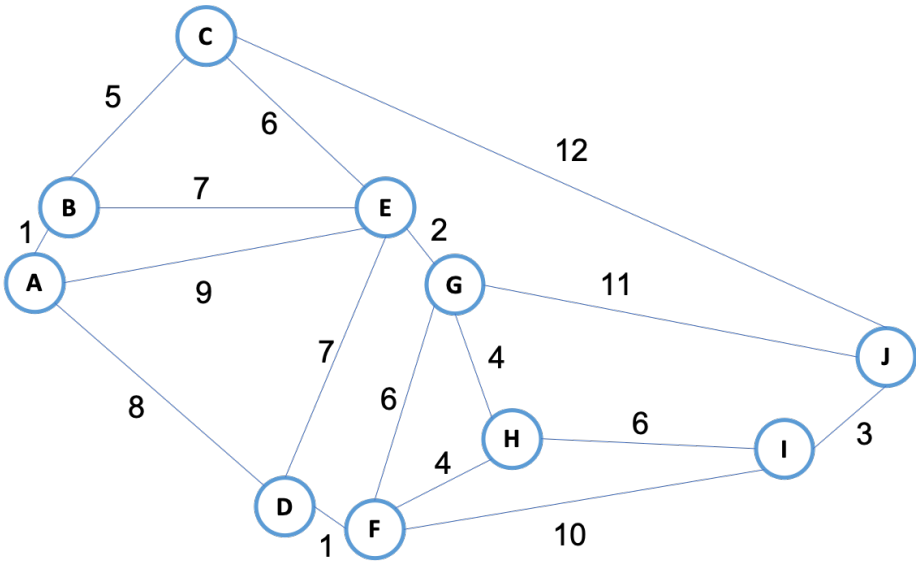
P4: [25pts] **Find Minimum Spanning Tree (MST) and Shortest Path**

The tourist map of Venice can be modeled as a connected, edge-weighted undirected graph with nodes being its major attractions and edges being tourist routes connecting two attractions.

(A) The following graph depicts the tourist routes and their corresponding distance in terms of kilometers among 10 major Venice attractions shown in the following figures.



Node	A	B	C	D	E	F	G	H	I	J
Attraction	Santa Croce	Santa Lucia Train Station	Campo di Ghetto Nuovo	Accademia	Fish Market	Guggenheim	Rialto Bridge	Piazza San Marco	Museo Storico Navale di Venezia	Arsenale



(a) Run **Prim's** algorithm taught in 16_MST on the graph, *starting from vertex A,* to find G's MST. (This minimizes the total distance covered, in kilometers, of the spanning tree.)

To answer this question, you need to follow the example in 16a_Prims_Worked_Example (Page 12 to 20) and provide the following information for each step:

(1) S that keeps the current explored nodes; initially $S = \{A\}$.
(2) T that keeps the current selected MST edges; initially $T = \{\}$.
(3) A table that tracks the key value and parent of each node.

Also, upon completion of the algorithm, please:

(4) draw the final MST, and
(5) provide the minimum total route length (in kilometers) achieved by this MST.

(b) Run **Kruskal's** algorithm taught in 16_MST on the same graph to again find its MST. *Recall that Kruskal's algorithm using the Union operation on Page 28.*

To answer this question, you need to follow the example in 16c_Kruskal_UF and provide the following information for each step:

(1) The current edge that is being tested, and the decision (*i.e., "YES" if this edge is added to the MST, "NO" otherwise*).
(2) The current Union-Find data structure after adding the edge to MST if the decision is "YES".

Also, upon completion of the algorithm, please:

(3) draw the final MST, and
(4) provide the minimum total route length (in kilometers) achieved by this MST.

(B) Run **Dijkstra's** Algorithm on the same graph to find the shortest path from Santa Lucia Train Station (node B) to all the other tourist destinations.

To answer this question, you need to follow the example in 17c_Dijkstra_Worked_Example and provide the following information for each step:

(1) A table that tracks the distance value ($d[u]$) and parent ($p[u]$) of each node.
Also, upon completion of the algorithm, please:
(2) draw the final shortest-path tree.

Hints, Comments and Recommendations:

- We strongly recommend typesetting the answers to these questions rather than handwriting them. Each step would require a lot of writing and you will have to document multiple steps.

  Typesetting rather than handwriting permits using cut-and-paste. Also, if you find that you made a small mistake at the beginning it permits you to easily fix those errors rather than having to rewrite the entire solution.

  We provide starting templates in powerpoint that you can use to typeset these problems. If you are more comfortable with other typesetting packages, please feel free to use them instead.

- If your algorithms encounter ties, they can break them arbitrarily.