CS4102 - ALGORITHMS - SPRING 2020

DUE SUNDAY, APRIL 12 AT 11P VIA COLLAB

H7(YL4DF)

Collaboration Policy: You are encouraged to collaborate with up to 4 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in the comments at the top of the tex file. You **are** permitted to collaborate through online tools such as Google Docs, interactive whiteboards, Google Meet, Google Hangouts, Zoom, Skype, etc, however you **must** limit written/typed details to high-level algorithm design. Each person is responsible for taking those ideas and turning them into pesudocode and a writeup. Do **NOT** copy and paste from shared documents, which includes re-typing verbatim or trying to disguise text that you are essentially copying. Over-collaboration of that form is fairly easy to detect with plagiarism tools. **Do not seek published or online solutions, including pseudocode, for this assignment.** If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite them. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: mw5ew

Sources: list any sources, e.g. Cormen, et al, Introduction to Algorithms

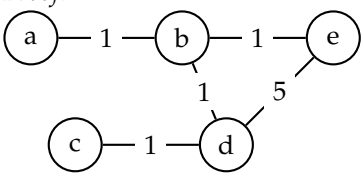
PROBLEM 1 *Unique Minimum Spanning Trees* [14 points]

1. Show that a graph $G = (V, E)$ has a unique minimum spanning tree T if, for every cut of G, there is a unique minimum weight edge crossing the cut.

   Proof. Prove by contradiction. Suppose there exist two different MSTs, T and T'. Let us sat that they differ by at least one edge e. That is, $e \in T$ and $e \notin T'$. If we remove e, T will be disconnected, so there exists a cut $(S, V - S)$ which e crosses. Since $e \in T$, we can say e is a minimum weight edge. For $e \notin T'$, we can still find a path in T' connecting the two vertices which are connected by e in T, because T' is MST. This implies there exists an edge $e' \in T'$ crossing the cut $(S, V - S)$ and e' is a minimum weight edge since T' is a MST. Since we know that for every cut of G there is a unique minimum weight edge crossing the cut, we can say $e = e'$. This lead to contradiction regarding that $e \notin T'$. We can conclude that the graph has a unique minimum spanning tree. □

2. Show the converse is not true by providing a counterexample.

   Proof.

   The graph has a unique MST with edge $\{(c, d), (d, b), (b, a), (b, e)\}$. However, for the cut $((a, b), (c, d, e))$, there are two minimum weight edges crossing the cut, namely (b, d) and (b, e). Thus, the converse is not true. □

PROBLEM 2 *Divide and Conquer with MSTs* [13 points]

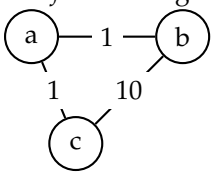
Professors Horton and Hott have been discussing minimum spanning trees and attempting to create new algorithms to compute them. Professor Horton claims to have created a divide and conquer algorithm as follows:

> Given a graph $G = (V, E)$, partition the set of vertices V into two sets V_L and V_R such that $|V_L|$ and $|V_R|$ differ by at most 1. Let E_L be the set of edges that are incident only on

> vertices in V_L and E_R be the set of edges incident only on vertices in V_R. Recursively compute the minimum spanning tree on each of the two subgraphs $G_L = (V_L, E_L)$ and $G_R = (V_R, E_R)$, then select the minimum weight edge $e \in E$ that crosses the cut (V_L, V_R). Use e to combine the two minimum spanning trees into a single minimum spanning tree for G.

Help us to evaluate his algorithm. Either prove that it correctly computes a minimum spanning tree of graph G or provide a counterexample for which the algorithm fails.

Proof. The algorithm fails. Counterexample and explanation is provided as follows.



Following the algorithm, we partition V into $V_L = (a)$ and $V_R = (b, c)$. Then $E_L = ()$ and $E_R = ((b, c))$. For G_L, the MST is just the vertex a. For G_R, the MST only consists of the edge (b, c). Then select the minimum weight edge (a, b) to combine two MSTs, so the final MST consists of edges (a, b) and (b, c). We also could use (a, c) to perform combining, but the final conclusion does not differ, so I use (a, b) to illustrate. However, the result is incorrect because the MST for this graph has the edges (a, b) and (a, c). Thus, the algorithm fails. □

PROBLEM 3 *Counting Shortest Paths* [13 points]

Given a graph $G = (V, E)$, and a starting node s, let $\ell(s, t)$ be the length of the shortest path in terms of number of edges between s and t. Design an algorithm that computes the number of distinct paths from s to t that have length exactly $\ell(s, t)$.

The algorithm is modified from BFS. For each node, we created two arrays, lpath[] and npath[]. lpath[] stores the length of shortest path from s to the current node. npath[] stores the number of shortest path from s to the current node. For all nodes, initialize lpath= ∞ and npath= 0. We begin the algorithm on s, so set lpath[s]= 0 and npath[s]= 1. Then we transverse like BFS. For every adjacent node u of each vertex v in G, we consider two cases. If lpath[u] > lpath[v]+1, we update lpath[u] = lpath[v]+1 and set npath[u]=napth[v]. If lpath[u] = lpath[v]+1, then npath[u]=napth[v]+npath[u]. The algorithm returns npath[t]. The pseudo-code is provided.

```
CSP(G, s):
    Initialize a flag d_v = 0, lpath[v]= ∞ and npath[v]= 0 for each node v
    Initialize the start node s as lpath[s]= 0 and npath[s]= 1
    Q.push(s)
    While Q is not empty:
        v=Q.pop() and set d_v = 1
        For each u ∈ V such that (u, v) ∈ E:
            If d_u = 0:
                Q.push(u)
            If lpath[u] > lpath[v]+1:
                lpath[u] = lpath[v]+1
                npath[u]=napth[v]
            Else if lpath[u] = lpath[v]+1:
                npath[u]=napth[v]+npath[u]
    return npath[t]
```

PROBLEM 4 *Lecture Review Questions* [20 points]

1. If you were given a list of nodes P and you needed to determine if P is a valid path in graph $G = (V, E)$, which of the two graph data structures for G would be more efficient, or would they be the same? (Name one of the choices, or say "same". No explanation required.)

   Adjacent Matrix

2. If a graph is not connected, can it have an MST? (Answer "yes" or "no." No explanation needed.)

   No

3. Briefly explain how Prim's algorithm insures that an edge that results in a cycle is never selected.

   It always selects the edge which connects a node in tree with a node not in tree

4. Which of the following are true? If nodes in a graph are partitioned by a cut into two subsets, S and $V - S$, then a set of edges that respects this cut can contain:

   a) an edge (a, b) where $a \in V$ and $b \in V - S$.
   b) two edges (a, b) and (c, d) where a and b are in V and c and d are in $V - S$
   c) only edges where all the edges' nodes are either in V or in $V - S$

   b

5. How is the key stored in the heap for Dijkstra's shortest path algorithm different that the key stored in Prim's MST algorithm?

   Instead of just the weight of an edge, Dijkstra's SP algorithm stores key as the total weight of shortest path from start node to the current node.

6. If you had an unusual weighted graph where all the edge-weights had the same value, name an algorithm other than Dijkstra's shortest path algorithm that you could easily use to find the shortest path between two nodes. Would this algorithm's time-complexity be better or worse than using Dijkstra's algorithm for this purpose?

   We could use BFS. It is better regarding time-complexity. BFS takes $O(V + E)$ whereas Dijkstra takes $O((V + E) \log V)$

EC2(yl4df)

Extra Credit Instructions: For each of the next three written homeworks (HW6, HW7, HW9), you will be given an extra credit problem, with the opportunity to replace one homework at the end of the course. Combined, the extra credit portions are worth 60 points—the same as one normal homework assignment. Each Extra Credit portion (EC1, EC2, EC3) will be combined into one *optional* replacement homework, meaning that the total scored out of all 60 extra credit points will replace your lowest homework grade for the semester. Note: this extra credit will **not** be applied to this homework (HW7). To make the most of this opportunity, you should commit to solving **all** extra credit portions for the next three homework assignments.

Since the extra credit is optional, no office hours will be provided to aid in solving the additional *optional* problems.

EXTRA CREDIT 1 *Martian Morels*

NEWSFLASH: SUBTERRANEAN HUMANOID RACE DISCOVERED ON MARS!
Since the Martians live in caves, their diet consists of a variety of Martian mushrooms. The tastiest is the Martian Morel, said to be more delicious than any food on Earth. Problematically, one subspecies of the Martian Morel is deathly toxic to humans. The Martians seem capable of distinguishing between toxic and edible mushrooms, but we humans need to verify that they can do so consistently before we can market the mushrooms to humans. More problematically, the Martian language is unlike any human languages, so the only of their words we have learned so far are "same", "different", and "uncertain".

We have n samples of Martian morels $s_1, ..., s_n$ each of which are either edible or toxic (we don't know which for any). A Martian was given all n^2 possible pairs of samples (s_i, s_j). The Martian then decided whether each sample's mushrooms were (a) the same (both edible or both toxic), (b) different (one edible and one toxic), or (c) that the Martian was uncertain. (All pairs were tested, but not all have "same" or "different" decisions).

At the end of the testing, the Martian decided that a total of m pairs were "same" or "different". Give an algorithm that determines whether these m are consistent. The decisions are consistent if there is a way to label each sample s_i with "edible" or "toxic" such that if the Martian indicated the pair (s_i, s_j) was "same" our labels for s_i, s_j matched, and if the Martian indicated the pair (s_i, s_j) was "different" our labels for s_i, s_j differed. Your algorithm should run in time $O(m + n)$.
Algorithm:

1. Construct a graph using n samples as nodes and m judgements as edges. If the judgement is "same", then we set the weight of that edge is 1. If the judgement is "different", then we set the weight of that edge is -1. If there are multiple edges between two nodes, we return false if different values appear among them. Pick up a start node s. It does not matter whether you set it to "edible" or "toxic", so I just set it to "edible". Mark it as visited. Then we transverse the graph using BFS. If the graph is no connected, we apply the algorithm on each connected part of it.

2. If a node is NOT visited, we assign "edible" or "toxic" based on the judgement. That is, for neighbor node x of node n, if edge weight is 1, we assign the same label to x as n. If edge weight is -1, we assign the different label to x from n. After assigning, we mark it visited.

3. If a node is visited, we perform checking based on the judgement. That is, for neighbor node x of node n, if edge weight is 1, we check whether x and n have same label. If not, return false. If edge weight is -1, we check whether x and n have different labels. If not, return false.

4. After we run the algorithm to assign label for each node and check against each judgement, it returns true.

Since we perform assigned on each node, it takes $O(n)$. Since we check against each judgement, it takes $O(m)$. Thus, the total running time is $O(n + m)$.