CS 170 Efficient Algorithms and Intractable Problems
Fall 2016 Christos Papadimitriou and Luca Trevisan

Midterm 1

Name: Joe Solutions

SID: 12345678

GSI and section time:

Write down the names of the students on your left and right as they appear on their SID.

Name of student on your left:

Name of student on your right:

Name of student behind you:

Name of student in front of you:

Instructions:

Write your name and SID on each sheet in the spaces provided.

You may consult one handwritten, double-sided sheet of notes. You may not consult other notes, textbooks, etc. Cell phones and other electronic devices are not permitted.

There are 6 questions. The last page is page 11, plus 3 blank pages for scratch.

Answer all questions. On questions asking for an algorithm, make sure to respond in the format we request. Write in the space provided. Good luck!

Note: If you finish in the last 15 minutes, please remain seated and do not leave early, to avoid distracting your fellow classmates.

Do not turn this page until an instructor tells you to do so.

1. **(12 pts.) Fun with Recurrences and FFT**

Write your answer in the box. You don't need to justify your answer.

(a) What is the solution of the recurrence $T(n) = 6T(n/3) + n^2$?

(b) Find a so that the solution of the recurrence $T(n) = aT(n/3) + n^2$ is $\Theta(n^3)$.

(c) What is the output of the FFT when the input is [0,-1,0,0]?

(d) In a flash of insight, you realize the FFT algorithm would work just fine splitting each polynomial into three! Fill in the missing pieces of the modified algorithm, which should have the same asymptotic runtime as the FFT we've seen.

Algorithm 1 TRIFFT(polynomial $A(x)$ of degree $n-1$, with n a power of 3)

1: Set $\omega := e^{2\pi i/n}$
2: **if** _____ **then**
3: **return** _____
4: Find polynomials A_0, A_1, and A_2 such that $A(x) :=$ _____
5:
6: For $j = 0$ to $n-1$:
7: _____
8: **return** _____

2. (17 pts.) Divide and Conquer with Roots

Let's find a divide-and-conquer solution to the following problem.

Input: a list $[a_1, \ldots, a_n]$ of n numbers, representing the roots of a polynomial $P(x)$. Thus, $P(x) = (x - a_1) \cdot (x - a_2) \cdots (x - a_n)$.

Output: The list $[c_0, \ldots, c_{n-1}]$ of coefficients of the same polynomial: $P(x) = c_0 + c_1 x + \ldots + c_{n-1} x^{n-1}$

For example, if $n = 2$ and the input is $[1, -3]$ then the output would be $[-3, 2, 1]$, because $(x-1) \cdot (x+3) = -3 + 2x + x^2$.

(a) Write pseudocode to solve this problem. You do not need to write a main idea, or prove correctness.

(b) Write a recurrence for your algorithm. *You don't need to justify your answer.*

(c) Solve the recurrence: find the runtime of your algorithm.

(d) **(1pt extra credit)**
$(x - a) \cdot (x - b) \cdots (x - z) =$

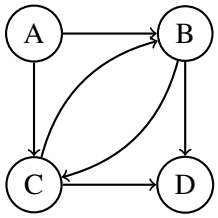
(*Hint:* Short answer expected.)

3. (28 pts.) True/False

Write T or F in the box. Then, if the statement is true, justify briefly. If the statement is false, give a counterexample.

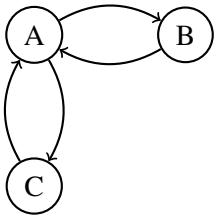
(a) ☐ Every DAG has a source vertex and a sink vertex.

(b) ☐ Every graph that has a source vertex and a sink vertex is a DAG.

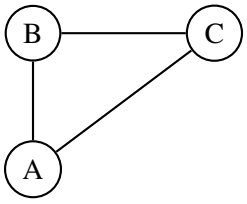


(c) ☐ A directed graph is a DAG if and only if it is linearizable.

(d) ☐ If all cycles in a directed graph are of length 2 (they consist of two edges), then its SCCs have sizes of at most 2 vertices. (By *cycle* we mean a closed path that does not pass through a vertex twice).



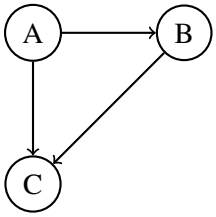
(e) ☐ In an undirected, weighted graph, if Dijkstra and Prim remove vertices from the priority queue in the same order, then the shortest path tree and the MST are the same.



(f) ☐ For any vertex in a weighted, undirected graph, the edge incident to that vertex that has smallest weight must be part of some MST of the graph.

(g) ☐ If the weights of the edges of a graph are all different, the MST is unique.

(h) ☐ If the weights of the edges of a graph are all different, the shortest path tree is unique.

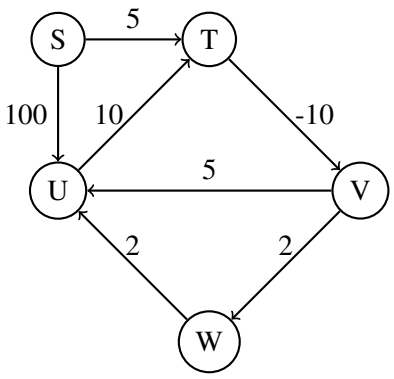


(i) ☐ If we add the same number to the weights of all edges of a graph, the shortest path tree remains the same.

(j) ☐ If the output of the FFT is [0,0,0,0] in list form, then the input was necessarily also [0,0,0,0].

(k) ☐ The MST is also the tree that minimizes the *maximum* weight of an edge on the tree.

4. (12 pts.) Bellman-Ford Ordering



(a) Consider the directed graph above, with positive edge weights. Find the distances of all the vertices from S.

Vertex v	dist(v)
S	
T	
U	
V	
W	

(b) Say we run Bellman-Ford to find these distances, and we can choose the order in which the edges are updated (this order is the same in every iteration of updates). Give an ordering of the edges (formatted as a list like $[ST, UT, VW]$) for which all distances are correct after only one iteration of updates (after each edge has been updated only once).
You don't need to justify your answer.

(c) With the same situation as part (b), give an ordering of the edges for there is at least one distance which is only correct after the last iteration of updates. *You don't need to justify your answer.*

5. (19 pts.) Shortest paths with tie-breaking

Let's modify Dijkstra's algorithm for shortest paths so when there are two or more shortest paths from the start vertex s to some vertex v, the algorithm returns the one that has the *fewest edges*.

For this question, we'll consider directed graphs with nonnegative integer edge lengths.

(a) To accomplish this, besides the `dist` array, we also maintain at each node v an integer `noe(v)` (the number of edges), initialized the same way as `dist`.

Modify the Dijkstra code fragment below, by editing the lines and/or adding new lines as needed, to accomplish this. After the algorithm runs, the `prev` pointers from v should indicate, in reverse, the shortest path from s to v with the fewest edges.

In the code fragment, `length(u,v)` is the length of edge (u,v). Q is the priority queue of vertices, and E is the set of edges.

Briefly explain your modifications.

```
while Q is not empty do:
    u = delete-min(Q)
    for all edges (u,v) in E:
        if dist(v) > dist(u) + length(u,v)
            ____________________________________________________
            dist(v) = dist(u) + length(u,v)
            ___________________________________
            prev(v) = u
```

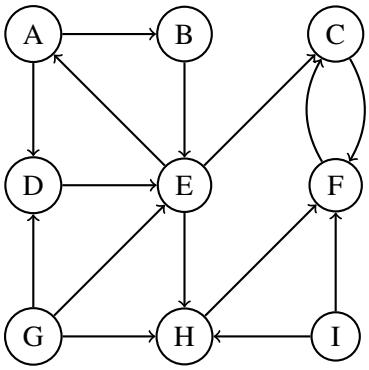
(b) Now suppose the algorithm is on the cloud somewhere, pitifully undocumented, or in C++, and you cannot edit it. There is a way to solve this problem using the original Dijkstra code, by modifying the input to Dijkstra's algorithm.

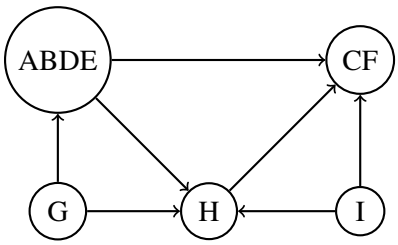
The given graph (V, E, ℓ) has nonnegative integer edge lengths $\ell(u, v)$ for all edges $(u, v) \in E$. Describe how to generate a new input (V, E, ℓ'), where you keep the same graph but modify the weights, so that the output of Dijkstra's solves the shortest paths with tie-breaking problem on the original graph.

Briefly justify the correctness of your solution.

6. (12 pts.) Linearization and Beyond

(a) Draw the DAG of the strongly connected components of the directed graph below.





(b) How many sink SCCs are there in the above graph? *No justification necessary.*

(c) Consider a directed graph G with multiple SCCs, and a single source SCC: give a short description for the number of edges needed to make the whole graph strongly connected.
Briefly justify your answer.