CMSC 27200 - Spring 2024 — Lorenzo Orecchia
due 4/19/24 via Gradescope

H5.jshe

- The assignment is due at Gradescope on 4/19/24.
- A LaTeX template will be provided for each homework. You are strongly encouraged to type your homework into this template using LaTeX. If you are writing by hand, please fill in the solutions in this template, inserting additional sheets as necessary. This will help facilitate the grading.
- You are permitted to discuss the problems with up to 2 other students in the class (per problem); however, *you must write up your own solutions, in your own words*. Do not submit anything you cannot explain. If you do collaborate with any of the other students on any problem, please list all your collaborators in the appropriate spaces.
- Similarly, please list any other source you have used for each problem, including other textbooks or websites.
- *Show your work.* Answers without justification will be given little credit.
- Your homework is *resubmittable*. Please refer to the course syllabus on Canvas for a more detailed description of this. For any problem that you have not changed from your last submission, please make sure to indicate this in your submission to help our graders grade faster.
- Is anyone still reading these?

Problem 1 (Midterm True / False) *For the following statements, state whether the statement is true or false. If the answer is true, provide a brief explanation why; if the answer is false, provide a counter example.*

(a) *For any pair of functions $f(n), g(n)$, if $f(n) = O\big(g(n)\big)$, then $\log\big(f(n)\big) = O\big(\log(g(n))\big)$.*

(b) *For any pair of functions $f(n), g(n)$, if $f(n) = \Omega\big(g(n)\big)$, then $\log\big(f(n)\big) = \Omega\big(\log(g(n))\big)$.*

(c) *Let $\mathcal{A}$ be a divide-and-conquer algorithm which takes an input of size n, and recursively calls itself. In a single call, it recursively calls itself 8 times, on inputs of size $n/2$. Suppose the work done outside of the recursion (i.e. the merging step) takes time $O(n^d)$ for some constant d. For any $d \geq 2$, the running time of the algorithm can be then be bounded by $O\big(n^d\big)$.*

(d) *Every fourth root of unity is also an eighth root of unit.*

(e) *Every DAG has at least one source and one sink vertex.*

(f) *Every directed graph with at least one source, and one sink will be a DAG.*

(g) *Given any graph $G = (V, E)$ with edge weights w_e, Dijkstra's algorithm can be used to find the shortest path between any two vertices $s, t \in V$, even if G has negative weight.*

Collaborators:

Solution:

(a) False.

Consider $f = 2$ and $g = 1$. $f = O(g)$ because $2 \leq 2 \cdot 1$.

However, $\log f \neq O(\log g)$ because $\log f = 1$, $\log g = 0$, and no $c > 0$ exists such that $1 \leq c \cdot 0$.

(b) False.

Consider $f = 1$ and $g = 2$. $f = \Omega(g)$ because $1 \geq \frac{1}{2} \cdot 2$.

However, $\log f \neq \Omega(\log g)$ because $\log f = 0$, $\log g = 1$, and no $c > 0$ exists such that $0 \geq c \cdot 1$.

(c) False

$$T(n) = \begin{cases} 8 \cdot T(\frac{n}{2}) + O(n^d) & n > 1 \\ 1 & n = 1 \end{cases}$$

By the Master Theorem, $T(n) = O(n^d \cdot \sum_{k=0}^{\log n} (\frac{8}{2^d})^k)$.

In the case $d = 2$, $\frac{8}{2^d} > 1$ so $T(n) = O(n^{\log 8}) = O(n^3)$

(d) True.

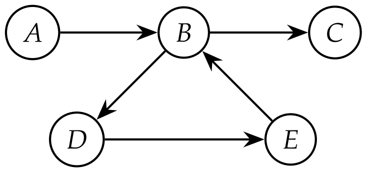
The eighth root of unity must be a ω_8 such that $\omega_8^8 = 1$.

Let ω_4 be the fourth root of unity such that $\omega_4^4 = 1$. Then $\omega_4^8 = (\omega_4^4)^2 = 1^2 = 1$. So ω_4^4 satisfies the definition of an eigth root of unity.

(e) Assume for contradiction there exists a DAG with no source. The path created by backtracing inbound edges of an arbitrary vertex will then be infinite. But on a finite graph this implies a cycle.

Assume for contradiction there exists a DAG with no sink. The path created by following outbound edges of an arbitrary vertex will then be infinite. But on a finite graph this implies a cycle.

(f) False. See counterexample:



(g) False. Dijkstra's algorithm breaks on negative weights.

Problem 2 *Show the running of Dijkstra's algorithm for finding the shortest distance from A to all other vertices in the graph from Figure 1. You should show the updated distance estimates to all the vertices (i.e., the key of the vertex in the priority queue) after each time a vertex is extracted from the priority queue in the algorithm. (Fill in the following table with the distances from A to all other vertices. Each row represents the distances after one more vertex is removed from the queue.)*

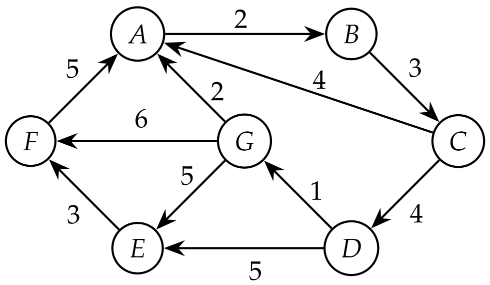


Figure 1: A directed weighted graph G.

Key in Queue:	*A*	*B*	*C*	*D*	*E*	*F*	*G*
Step 1: -	0	∞	∞	∞	∞	∞	∞
Step 2: A	0	2	∞	∞	∞	∞	∞
Step 3: B	0	2	5	∞	∞	∞	∞
Step 4: C	0	2	5	9	∞	∞	∞
Step 5: D	0	2	5	9	14	∞	10
Step 6: G	0	2	5	9	14	16	10
Step 7: E	0	2	5	9	14	16	10
Step 8: F	0	2	5	9	14	16	10