Midterm 1A

CSE 101 Spring 2022

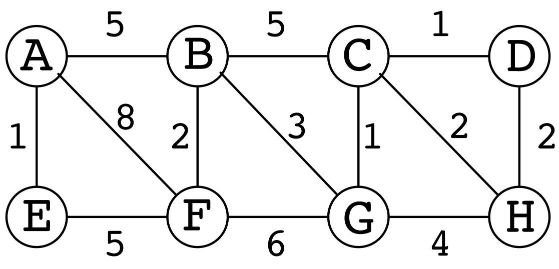
Name: ______________________________________

Student ID: ___________________________________

Section (Russell/Sanjoy): _________________________

INSTRUCTIONS: Be clear and concise. Write your answers in the space provided. Use the backs of pages for scratchwork.

1. **(10 points)** Consider the following graph with edge weights.



(a) Give distances from node E to all other nodes.

A	B	C	D	E	F	G	H

(b) Show the corresponding shortest-path tree.

You just have to fill in the numbers and tree, but to show how you are intended to compute them, I'll describe simulating Dijkstra's algorithm on this graph.

We start at E, which is given distance 0. From E we can reach A at 1, F at 5. So now our queue looks like : A, 1, E; F, 5, E.

Since A has the smallest key, we lock in distance 1 and parent pointer E for A. Exploring from A, we can reach B at 1+5=6, F at 1+8=9 which is worse than its current distance. So we delete A and add B, but don't change F. The queue looks like F, 5, E; B, 6, A.

So we lock in F and explore from F. From F , we can reach B at 5+2=7¿6 and G at 5+6=11. Now the queue is B, 6, A; G, 11, F.

We lock in B, and explore. From B we can reach G at 9, which is an improvement, and C at 11. So now the queue is G, 9, B; C, 11, B.

We lock in G and explore. We can reach C at 9+1=10 and H at 9+4=13. The queue is C,10,G; H,13, G.

We lock in C and explore. We can reach H at 10+2 =12, and D at 10+1=11. The queue is H,12, C; D, 11, C.

We lock in D and explore. We can reach H at 11+2 =13¿ 12. So we keep H at 12 from C, which is the last vertex left.

So the distances found are : E:0, A:1; F: 5; B:6; G: 9; C:10, D: 11, H:12. Rearranging, the array should look like : A:1,B:6, C:10, D:11, E:0, F:5,G:9, H:12.

The parent pointers in the tree are: A: E, F: E, B: A, G: B, C: G; D: C; H:C. Those are the edges in the shortest path tree.

2. **(10 points)** For each of the following statements, say whether it is TRUE or FALSE. No explanation is needed.

   While no explanations are needed for you, I'll try to provide them.

   (a) $2^{2n} \in \Theta(2^n)$ False. $2^{2n}/2^n = 2^n$ which goes to infinity, not a constant.

   (b) $7n \log n + 20n \in \Theta(n \log n)$
   Yes, the leading term is of the form $n \log n$, so it is $\Theta(n \log n)$.

   (c) If $T(1) = 1$ and $T(n) = T(n-1) + O(1)$ for $n \geq 1$, then $T(n) \in O(n)$
   True. We can see this by unravelling. $T(n) = c + T(n-1) = 2c + T(n-2) = 3c + T(n-3) + ...(n-1)c + T(1) = c(n-1) + 1 \in O(n)$.

   (d) If f and g are functions from positive integers to positive integers, and $f(n) \in O(g(n))$, then $f(n)^2 \in O(g(n)^2)$
   True. If $f(n) < Cg(n)$ for $n > N$, $f(n)^2 < C^2 g(n)^2$ for $n > N$.

   (e) For any directed acyclic graph G, $|E| \in O(|V|)$
   False. We could have a complete bipartite graph with n vertices on each side, and direct the edges from left to right. This is a DAG, because there are no directed cycles. $|E| = n^2, |V| = 2n$, so $|E|$ is asymptotically larger than $|V|$.

3. **(10 points)** A subsequence of a word is one that can be obtained by deleting some characters and listing the remaining characters in the same order. For example, $MATH$ is a subsequence of $AMATEURISH$ by keeping the second, third, fourth and last characters, but is not a subsequence of $ARITHMETIC$ because the only H comes before the only M. Here is an algorithm that, given two words $u_1 \cdots u_n$ and $v_1 \cdots v_m$, decides whether $u_1 \cdots u_n$ is a subsequence of $v_1 \cdots v_m$.

   Subsequence($u[1 \dots n], v[1 \dots m]$: words)

   ```
   (a) $I \leftarrow 1, J \leftarrow 1$
   (b) While $I \leq n$ and $J \leq m$ do:
   (c)     While $J \leq m$ AND $v[J] \neq u[I]$ do: J++
   (d)     If $J \leq m$: I++; J++
   (e) IF $I > n$: return $True$
   (f) Return $False$
   ```

   Give a time analysis, up to order, for this algorithm. Be sure to explain your answer.

   We are just asked to give the time analysis, not to prove correctness. Note that each iteration of either While loop, we increment J and never decrease J anywhere. So the total number of times we go through either while loop combined, is at most m, since both loops stop when $J > m$. All inside operations in either While loop are constant time, so the total time is $O(m)$. $O(n + m)$ is equivalent and also correct since $n < m$.

4. **(20 points)** Explain how we can modify or use one of the graph algorithms from class to solve the following problem.

   > Given an undirected graph G, give a minimum sized set of edges $e_1, \dots, e_k$ so that adding $e_1, \dots, e_k$ to G causes G to become connected. (If G is already connected, you should return the empty set.)

   (5 points correct algorithm, 5 points correctness proof, 5 points efficiency, 5 points time analysis)

   Say that G has k connected components. (Since it is undirected, we don't need to worry about whether these are strongly connected; they are.) Each edge we add can only decrease the number of connected components by 1, so to make the graph fully connected, we must add at least $k-1$ edges. On the other hand, if we pick one vertex from each connected component, $v_1, ..v_k$, and put edges in a path from v_1 to v_2, to $...v_k$, we've added $k-1$ edges and connected one vertex in each component. Since all other vertices have paths to the v_i in its component, the graph has become fully connected by adding $k-1$ edges.

   In order to find these edges algorithmically, we can use a complete DFS to identify the connected components of the graph. Then we can go through each vertex and find the first vertex in each connected component. Finally, we can compute the list of edges that form a path through these vertices. This takes time $O(|V|+|E|)$ for the complete DFS, and time $O(|V|)$ for the rest, giving total time $O(|V|+|E|)$.