Final Exam
Introduction to Algorithms/Algorithms 1
600.363/463

Monday, May 13th, 2-5pm

Ethics Statement

I agree to complete this exam without unauthorized assistance from any person, materials, or device.

Name

Signature Date

1 Problem #1: (100 points) YES/NO QUESTIONS (10 questions, 10 points per question)

For every question answer yes or no.

1. We can use Red-Black Tree to solve the disjoint sets problem but the solution will be slower than the algorithm that uses union by rank and path compression.

   yes no

2. $\sum_{i=1}^{n} 3^i = \Theta(3^{n-10})$

   yes no

3. There exist two strictly positive functions f, g such that $f = o(g)$ and $g = o(f)$.

   yes no

4. $\log(n) = O(n^{-1/2})$.

   yes no

5. Consider Red-Black Tree T with n nodes. For any n there exist two nodes a, b such that the length of a path from a to b in T is at least $0.1n$.

   yes no

6. For any optimization problem it is possible to design a constant approximation algorithm using dynamic programming or greedy approach.

yes no

7. The recurrence $T(n) = 5T(0.33n) + n^7$ can be solved with Master Theorem.

yes no

8. There exists a known polynomial time algorithm for finding a longest path in a graph.

yes no

9. Let G be undirected, weighted graph. If all weights are distinct then there is a unique MST.

yes no

10. Let D_1 be a data structure that achieves the amortized complexity of $O(g(n))$. Let D_2 be a data structure that achieves the worst-case complexity of $O(g(n))$ (for the same set of operations as D_1). If running time is the most important factor in our decision-making process then we should always prefer D_2 to D_1.

yes no

Problem #2: 50 points

Let A be an array of integers of size n and let x, y be two integers such that $x < y$. Design an algorithm that checks whether there exist three elements of A whose sum is between x and y. Thus, find if there exist i, j, k such that $x \leq A[i] + A[j] + A[k] \leq y$. Your algorithm must report such i, j, k or output "ERROR" if no such i, j, k exist (it is sufficient to output any triple if more than one exists.) Full credit will be given if your solution is correct, your algorithm works in $O(n^2 \log(n))$ time and you clearly explain why your algorithm works correctly.

Solution: Sort all arrays. Loop over all $A[i]$, $A[j]$ and compute $a = x - A[i] - A[j]$ and $b = y - A[i] - A[j]$. Using binary search find indices in C such that $A[s] < a \leq A[s+1]$ and $A[t] \leq b < A[t+1]$. If $s < t$ then take $k = s + 1$ and report it. The running time of all sorting is $O(n \log(n))$. The number of loop iterations is n^2 and the running time of binary search is $O(\log(n))$. Thus, the total time is $O(n^2 \log(n))$.

Correctness: Such triple exists if and only if there exists (i,j,k) such that $a \leq A[k] \leq b$, where $a = x - A[i] - A[j]$ and $b = y - A[i] - A[j]$ as we defined. Equivalently, let s, t be such that $A[s] < a \leq A[s+1]$ and $A[t] \leq b < A[t+1]$. There exists an element of the array in the range $[a, b]$ if and only if $s + 1 \leq t$. Note that $A[s] < a \leq b$ and thus $s \leq t$. However, it might be the case that $t = s$ in which case i, j, k is not the right triple.

Problem #3: 50 points

Let $G = (V, E)$ be a directed acyclic graph and let s, t be two vertices. Give an algorithm that computes the number of different paths from s to t in G. Full credit will be given if your solution is correct, your algorithm works in $O(V + E)$ time and you clearly explain why your algorithm works correctly.

Solution: Topologically sort the graph. Count the number of different paths from s to any other node. Let m_a be a variable that will represent the number of different paths from s to a. Initialize all with $m_a = 0$ for every node before s in the topological order. Then traverse all nodes in the topological order and for every node b after s in the topological order, the number of paths is equal to the $x + sum_{(a,b)} m_a$ where $x = 1$ if $(s, b) \in E$ and 0 otherwise.

Correctness: Since G is a DAG, there is no path from s to any node before s in the top. order, so for any such node m_a must be 0. Let us prove by induction on the order of the node that is after s that m_a is a correct value. For the node that comes right after s we have two cases: if there is an edge between s and b then there is exactly one path; otherwise, there must be zero paths. In both cases, the formula gives the right answer.

Inductive step: every path from s to b must pass through a vertex before b. Precisely, $P : s \rightsquigarrow b$ is a path from s to b if either $P = \{(s, b)\}$ or P is concatenation of $P_1 : s \rightsquigarrow a$ such that $(a, b) \in E$. Since G is a DAG, a must be before b in the top. order. Thus, by induction m_a must have a correct value for a at the moment that we compute it for b.

Problem #4: (50 points)

Let G = (V, E) be a directed weighted graph that is strongly connected. All weights are non-negative. Let $a, b \in V$ be two vertices. You can choose exactly one edge $e \in E$ and set $w(e) = 0$. Your goal is to minimize the length of the resulting shortest path from a to b. Design an algorithm that will output such edge and the length of a shortest path from a to b in G with the new weights. Full credit will be given if your solution is correct, your algorithm works in $O(V \log(V) + E)$ time and you clearly explain why your algorithm works correctly.

Find the lengths of all shortest paths from a to all nodes. Find the lengths of all shortest paths from all nodes to b. (This can be done by reversing all edges and running shortest paths alg. from b.) For each edge $(s,t) \in E$, consider a new path P which is obtained as follows. Concatenate path $P_1 : a \rightsquigarrow s$ with (s,t) and with $P_2 : t \rightsquigarrow b$ where P_1, P_2 are shortest paths in G. If we set $w(s,t) = 0$ then the new path length is $\delta(s,a) + \delta(b,t)$. Note that the length of P_1 or P_2 won't be affected since shortest paths do not contain loops. Also, note that we can improve the existing shortest path. Thus, the minimum will be of the above form and it is sufficient to check all edges and find the one that gives minimum. If we use Dijkstra, and an additional $O(E)$ steps to compute the minimum then the running time is $O(V \log(V) + E)$.8

Problem #5: (Dynamic Programming) (50 points)

A palindrome is a nonempty string that reads the same backward and forward. For example, "madam," "dodod" and "zzzzz" are palindromes. Design a dynamic programming algorithm that finds the length of a longest palindromic subsequence of a given input string. For example if the input is "aacbba" then the output should be 4 (the length of "abba"). If the input is "acdc" then the output should be 2 (the length of "cc"). Full credit will be given if your solution is correct, your solution uses dynamic programming and works in polynomial time and you clearly explain why your algorithm works correctly.

SOLUTION: Fill the matrix: Solution: Let $P[i, j]$ be the length of the longest palindrome in the substring $A[i], \ldots, A[j]$. Then, $P[i, i] = 1$, and also $P[i, i+1] = 1$ if $A[i] \neq A[i+1]$ and $P[i, i+1] = 2$ if $A[i] = A[i+1]$

Consider $j > i + 1$. If $A[i] = A[j]$ then $P[i, j] = P[i+1, j-1] + 2$. Else $P[i, j] = \max\{P[i, j-1], P[i+1, j]$.

We can fill this matrix in quadratic time.

Problem #6 : Formal proofs (50 points)

Let $G = (V, E)$ be a flow network with all capacities being zero or one. (Recall that we consider networks where $(a, b) \in E$ implies $(b, a) \notin E$. Prove formally (e.g., by induction on the number of iterations) that every iteration (except the last iteration) of Ford-Fulkerson will increase the flow by exactly one unit. Prove that the number of iterations is at most $|V|$.

Solution:

We prove the following claim by induction on the number of the iterations M:

1. During the M-th iteration the flow will increase from $M - 1$ to M.
2. For every edge $(a, b) \in E$ it is true that $f(a, b) \in \{0, 1\}$.

$M = 1$: initially the flow is 0. If there is a path from s to t in the residual network then all residual capacities are 1 on that path. Thus, the flow will be increased by exactly one unit and $|f| = 1$. Also, since we increase the flow by one on a single path then for every edge $(a, b) \in E$ it is true that $f(a, b) \in \{0, 1\}$.

$M \to M + 1$: Assume that this is not the last iteration and thus the flow can be increased. Thus, there must be a path from s to t in the residual network. By induction the current value of the flow is M. By the theorem that we proved in class, the flow will increase by $c_f(P)$ where $c_f(P)$ is a residual capacity of the path P that we found during the $M + 1$-th iteration. By the second claim, $f(a, b) \in \{0, 1\}$ and thus $c_f(a, b) \in \{0, 1\}$. Thus, $c_f(P) = 1$. Thus, the flow will be increased by exactly one unit and $|f| = M + 1$.

To prove the second part, consider $(a, b) \in P$. If $(a, b) \in E$ then it must be the case that $f(a, b) = 0$ before augmenting the flow with P (indeed, otherwise by induction $f(a, b) = 1$ and $(a, b) \notin E_f$, i.e., not in the residual network.) Thus, $f(a, b) = 1$ after augmenting the path. If $(a, b) \notin E$ then it must be the case that $f(b, a) = 1$ before augmenting the flow with P and $f(a, b) = 0$ after (indeed, otherwise by induction $f(b, a) = 0$ and $(a, b) \notin E_f$, i.e., not in the residual network.) Thus, $f(b, a) = 0$ after augmenting the path. Other values did not change. Thus, the statement is correct.

Problem #7: Concepts (30 points)

In a few sentences explain what is a FPTAS. Full credit will be given to the right idea. No formal definition is required.