CSCI-GA.3520-001: Honors Analysis of Algorithms

Final Exam, Dec 15, 2022, 1:00pm-5:00pm

- This is a four hour exam. There are six questions, worth 10 points each. Answer all questions and all their subparts.
- **No** access is allowed to textbooks, course notes, any other written or published materials, any online materials, and any other materials stored on devices.
- In the past years, to pass this exam, one needed to answer 3 or 4 problems well, instead of answering all the problems poorly.
- You must submit separate answers for separate questions.
- Read the questions carefully. Keep your answers legible, and brief but precise. Assume standard results and algorithms (i.e., those taught or referred to in class or homeworks).
- You must prove correctness of your algorithm and prove its time bound unless stated otherwise. The algorithm can be written in plain English (preferred) or as a pseudo-code.

Best of luck!

Problem 1

Let T be a binary tree with n vertices. Assume that the vertices are labeled 1 through n, with the root labeled as 1. Assume also arrays $\mathsf{left}(v)$ and $\mathsf{right}(v)$ that denote, respectively, the left and right children of a vertex (with label) v. The convention is that $\mathsf{left}(v) = 0$ when v has no left child and similarly $\mathsf{right}(v) = 0$ when v has no right child. For every vertex v, there is a non-negative value $\mathsf{val}(v)$ associated with it. Let $1 \leqslant k \leqslant n$ be a given integer. Design a polynomial time algorithm to find a subtree H with maximum total value such that

- Root $1 \in H$
- H has k vertices.

The total value of H is the sum of values of all vertices in it. H must contain the root.

We note that a subtree is just a connected subgraph of T. A related but *different* notion is $\mathsf{tree}(v)$, which, for a vertex v, denotes the *entire* subtree rooted at v consisting of v and all its descendants (chidren, their children, and so on).

Problem 2

Suppose you are storing a stack in an array. That is, a stack with elements $x_1, \ldots, x_s$, from the bottom to the top, is stored as the first s elements $A[1], \ldots, A[s]$ in an array of size k, where $s \leqslant k$ and the remaining $k - s$ array locations are empty.

However, you want to be space efficient. So if the stack holds s items it is to be stored in an array of size $O(s)$. That is, in the above notation, k must be $O(s)$ at all times.

Suppose you can request the memory management system to provide a new empty array of any requested size and it will do so in $O(1)$ time. You will have to copy the contents of the current array into the new array (this takes time proportional to the size of the new array) and then return the previous array to the memory management system (this takes $O(1)$ time).

Show how to maintain an initially empty stack S under a series of n pops and pushes, in time $O(n)$ in total, while maintaining space efficiency at all times. Each pop and push involves only a single element.

Hint: Use amortization and an appropriate potential function.

Problem 3

1. Consider the recurrence relation (C is a fixed constant)

$$T(n) \leqslant 2 \cdot T\left(\frac{n}{2}\right) + Cn \log n.$$

   Show that $T(n)$ is $O(n \log^2 n)$.

2. Let $A[1], \ldots, A[n]$ be an array of integers; they could be either negative or positive or mixed. Give an $O(n \log^2 n)$ algorithm to find a consecutive subarray $A[i:j]$, $1 \leqslant i \leqslant j \leqslant n$, for which $\sum_{h=i}^{j} A[h]$ is closest to 0, i.e. minimizing $\left|\sum_{h=i}^{j} A[h]\right|$.

Problem 4

You are given a directed graph $G = (V, E)$ with non-negative edge weights $\mathtt{wt}(u, v)$ representing the time it takes to go from a vertex u to v if there is a u to v edge. The graph is given in the adjacency list representation. Some subset $F \subseteq V$ of vertices are pharmacies, while some other (disjoint) subset $D \subseteq V$ are doctor's offices. You live at a node $s \in V, s \notin F \cup D$, your mother lives at a node $t \in V, t \notin F \cup D$ and she has a medical emergency. You now need to go to some doctor's office $d \in D$ for a prescription for a drug, *then* to some pharmacy $f \in F$ to purchase the drug, and finally to t to deliver the drug to your mother.

1. Design an efficient algorithm to find the route with the least total weight. Specifically, design a weighted graph G' such that the answer reduces to a single call to Dijkstra algorithm on G'. Explicitly define G', and compute upper bounds on its number of vertices n' and edges m' as functions of $n = |V|$ and $m = |E|$ of the original graph. State the final run-time of your algorithm (as a function of n and m).

2. Now suppose that the graph $G(V, E)$ is acyclic. Design an $O(m + n)$ time algorithm for the same task. Replace Dijksta's algorithm earlier by an appropriate $O(m+n)$ algorithm to solve the problem (for the special case when G is acyclic, G' should also be acyclic). Justify why your algorithm works.

Problem 5

A 3-uniform hypergraph $H(V, E)$ consists of a set of vertices V and a set of hyperedges E; let $n = |V|$ and $m = |E|$. Each hyperedge $e \in E$ is a size 3 subset of V (i.e. it is a triple of vertices).

A $\{\mathsf{red}, \mathsf{blue}\}$-vertex coloring is a map $\Phi : V \to \{\mathsf{red}, \mathsf{blue}\}$. Given such a coloring, an edge e is said to be *properly colored* if it has exactly one red vertex and two blue vertices.

Show that there exists a $\{\mathsf{red}, \mathsf{blue}\}$-vertex coloring Φ that properly colors (at least) $\frac{4m}{9}$ hyperedges.

Give a polynomial time randomized algorithm that finds such a coloring. Specifically, your algorithm should output, with probability (at least) 0.99, a coloring that properly colors at least $\frac{4m}{9}$ hyperedges.

Problem 6

Recall that an instance ψ of 3SAT consists of n Boolean variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$ where every clause is of the type $\ell_i \vee \ell_j \vee \ell_k$, i, j, k are distinct, and ℓ_i is a literal, either x_i or $\overline{x}_i$. A Boolean assignment to the variables is called satisfying if it satisfies every clause.

Let K be a fixed constant (e.g. $K = 20$). An instance φ of K-Bounded Occurrence 3SAT (K-BO-3SAT for short) is similar except for these conditions:

- Every literal ℓ_i occurs in at most K clauses.
- Clauses of size one and two are allowed (i.e. clauses of form ℓ_i and of form $\ell_i \vee \ell_j$).

Let

$$K\text{-BO-3SAT} = \{\ \varphi \mid \varphi \text{ is a } K\text{-BO-3SAT instance that has a satisfying assignment}\}.$$

Show that K-BO-3SAT is NP-complete. The specific value of K is not important (i.e. up to you to choose) as long as it is a fixed constant.