CS362 Exam III

Last Name:

First Name:

Email:

Honesty Policy:

1. This is a close-book exam. You may only use the double-sided handwritten US letter sized "cheating sheet" and a dumb calculator that you have brought to the exam. Nothing else permitted.
2. You are not allowed to talk to anyone except the professor in charge during the exam. You will be **thrown out** if caught talking to other students or peeking at other students' exams.
3. It is your responsibility to guard your work. Don't let other people copy your solutions.

Exam Instructions:

1. Please silence your cell phones.
2. Write your answers legibly.
3. There are 5 problems in this exam.
4. For algorithmic problems, do not give detailed pseudo-code. Describe the key ideas and key steps of your solutions, correctness argument, and provide running time analysis. In the case, where you are using a well-known algorithm (e.g., merge sort or linear time selection) as a subroutine without modification, just cite the well-known algorithm and its running time. Do not repeat the steps of the well-known algorithms.
5. Don't spend too much time on any single problem. If you get stuck, move on to something else and get back later.
6. Don't leave multiple solutions for the same problem. You may be graded based on the one that's wrong.
7. Good luck and enjoy the exam!

1. Let $G(V, E)$ be an undirected graph with positive edge weights. let $v_0 \in V$ be a special node in G. Design an efficient algorithm for finding shortest paths between all paris of vertices with the added restriction that all these paths must pass through v_0.

   Solution:

   Find the shortest paths from all vertices to v_0, and from v_0 to all other vertices. For any pair of vertices u, w, combine the shorest paths from from u to v_0 and from v_0 to w.

2. Consider the following sampling algorithm called **reservior sampling**. Suppose we have a sequence of items passing by one at a time. You may assume that in the $j-$th iteration, the item a_j appears.

   The goal is to maintain **one** sample item that is uniformly distributed over all the items that we have seen so far. In other words, at any moment, this sample we have taken is equally likely to be any of the items that we have seen so far.

   The reservior sampling alogorithm works as follows. When the first item a_1 appears, it is taken as the sample, when the second item a_2 appears, it replaces the current sample a_1 as the new sample with a probability $\frac{1}{2}$, ..., when the $k-$th item a_k appears, it replaces the current sample as the new sample with a probability $\frac{1}{k}$.

   What is the probability that after the $n-th$ iteration, some item a_j is taken as the sample? Please explain your answer in details.

   Solution:

   For vertex a_j, the probability that it becomes the sample is $\frac{1}{j}$. The probabiliyt that it remains to be the sample at the $n-$th iteration is:

$$\frac{1}{j} \cdot \left(1 - \frac{1}{j+1}\right) \cdot \left(1 - \frac{1}{j+2}\right) \cdot ... \cdot \left(1 - \frac{1}{n}\right) = \frac{1}{n}$$

3. Given a directed graph $G = (V, E)$ and two vertices $s, t \in V$. Two paths from s to t are said to be **independent** if they do not have a vertex in common apart from s and t. Design an efficient algorithm to find the **maximum** number of **independent paths** from s to t.

   Solution:

   Assign unit capacity to **each edge** and **each vertex**. Note that unit weight for **each vertex** is **required**, while the unit weight on the edge is not. The maximum flow than correspond to the maximum number of disjoint paths.

   Alternatively, one can also split each vertex into two dummy vertices connected by a unit capacity edge.

   Because of each vertex must be assigned a unit weight, the min-cut on the graph is **NOT** the number of disjoint paths on the original graph.

4. The input are two strings of n letters from the English alphabet: $a[1..n]$ and $b[1..n]$. Design an efficient algorithm to determine whether a is a cyclic shift of b.

   Solution: Using rolling hash algorithm as discussed in class, this can be done in linear time.

5. A topic that we didn't discuss in class is *parallel computing.*

   Parallel algorithms are usually designed for a specific parallel architecture. One popular such architecture is called **Concurrent Read and Concurrent Write (CRCW)**. In the CRCW computing model, one assumes that there are multiple processors that can read and write simultaneously to a shared main memory.

   Suppose you have a CRCW computer, which is equipped with unlimited number of processors and unlimited shared memory. Design an algorithm that can merge two sorted arrays $A[1..n]$ and $B[1..n]$ into a sorted array $C[1..2n]$ in $O(1)$ time.

   (Hints (1) You may assume that these $2n$ numbers are distinct. (2) You can use as many as $O(n^2)$ processors.)

   Solution:

   If I remembered correctly, this problem was designed by one of the former CEOs of Google, and has been used as interview questions before.

   The problem has been simplified with the assumptions that all numbers are distinct. You can try the version with duplicates.

   This problem has two key points. The first part is how to assign the processors, and the seocnd part is how to write to $C[1..2n]$.

   - Taking care of the array A:
     (a) Assign a processor $P_{i,j}$ for each pair of numbers $A[i]$ and $B[j]$. This uses $O(n^2)$ processors.
     (b) Each processor will perform **two** comparisons: $A[i]$ vs. $B[j]$, and $A[i]$ vs. $B[j+1]$.
     (c) If $B[j] < A[i] < B[j+1]$, the processor $P_{i,j}$ will write $A[i]$ to meomory location $C[i+j]$, else **do nothing**
   - Taking care of the array B:
     (a) Assign a processor $P_{j,i}$ for each pair of numbers $B[j]$ and $A[i]$.
     (b) Each processor will perform **two** comparisons: $B[j]$ vs. $A[i]$, and $B[j]$ vs. $A[i+1]$.
     (c) If $A[i] < B[j] < A[i+1]$, the processor $P_{j,i}$ will write $B[j]$ to meomory location $C[i+j]$, else **do nothing**