

Quiz 2
Sections A1, A2, A3

90 mins

Name:

GT number:

Email:

Write your answers on this handout. WRITE YOUR NAME ON THIS PAGE NOW!

Good Luck!

Question 1: Median, Recursive Design, 25 Points

Let a be a sorted array of n distinct positive or negative integers $a_1 < a_2 < \dots < a_n$. Give an algorithm that finds an i such that $a_i = i$, provided that such an i exists. Your algorithm should use $O(\log n)$ comparisons. You may assume that n is a power of 2. You should give a clear description of your algorithm in English and use pseudocode, only if you find it necessary. You should argue about the running time.

In order to get an $O(\log n)$ comparison algorithm for an input of size n , we must design an algorithm which, in one round, reduces the size of the instance by a constant fraction, while using a constant number of comparisons. So we are seeking for an algorithm whose complexity is $T(n) = T(n/c) + c'$, for constants c and c' . How can we use the structure of the problem to design such an algorithm?

Here is the structure in the problem. Suppose that we probe the input at some position i and compare a_i to the number i . If $a_i > i$ thus $(a_i - i) > 0$ then, for all $j > i$, we will have $a_j > j$. This is because all the a_i 's are distinct so $a_j \geq a_i + (j - i) = (a_i - i) + j > j$. Similarly, if $a_i < i$ then, for all $j < i$, $a_j < j$.

We now have an algorithm. Compare $a_{\frac{n}{2}}$ to $\frac{n}{2}$. If $a_{\frac{n}{2}}$ is equal to $\frac{n}{2}$ then we have found the fixed point. If it is larger then we can discard the right half of the array and recurse on the left part, which is half the size. If it is smaller then we can discard the left half of the array and recurse on the right part, which is also half the size.

Question 2: MST, Kruskal, 25 Points

Let $G(V, E)$ be an undirected weighted graph given in its adjacency list representation. Let $e = \{u, v\} \in E$ be a specified edge. Explain how Kruskal's algorithm can be modified to find a spanning tree of G that contains e and is of minimum cost among all spanning trees that contain e .

Kruskal's algorithm first sorts the edges by weight. It then considers the edges in this sorted order and, while a tree has not been obtained, it adds the next edge which does not create a cycle.

The edge e must be in all spanning trees considered in this problem. One way to ensure this is by defining a graph G' which is the same as G , except that the cost of e has been reduced to 0, and run Kruskal's algorithm on G' . The edge e will be certainly considered and added first. Let T' be a minimum cost spanning tree produced by this algorithm for G' . Then, the cost of the corresponding spanning tree T of G will be $\text{cost}(T) = \text{cost}(T') + \text{cost}(e)$.

We need to argue that all spanning trees of G that contain e have cost at least $\text{cost}(T)$. Suppose otherwise. Let T^* be a spanning tree of G that contains e and has $\text{cost}(T^*) < \text{cost}(T)$. But $\text{cost}(T^*) = \text{cost}(T^* - e) + \text{cost}(e)$, which implies that there exists a spanning tree $T^{*'}$ in G' with $\text{cost}(T^{*'}) < \text{cost}(T')$, contradiction.

Question 3: Adjacency List Representation, 25 Points

Let $G(V, E)$ be a graph given in its adjacency list representation. Suppose that the adjacency lists contain multiple edges, in the sense that vertices may appear more than once on the same list ($x \rightarrow y, z, z, y, y, z$). Give an algorithm that finds an equivalent simple graph $G'(V, E')$, where E' consists of the edges in E with all multiple edges between two vertices replaced by a single edge ($x \rightarrow y, z$ or $x \rightarrow z, y$). Your algorithm should have running time $O(|V| + |E|)$. You should give a clear description of your algorithm in English and use pseudocode, only if you find it necessary. You should argue about the running time.

Hint: Use an additional array of length $|V|$ that maintains, for each vertex y , the names of the vertices x such that y appears on the list of x .

Let us first realize where the difficulty of the problem lies. Suppose that I scan through $x \rightarrow y, z, z, z, z, z, z, y, y, z$ wanting to produce $x \rightarrow y, z$. In order for the whole algorithm to terminate in time $O(|V| + |E|)$, this task of removing multiplicities from the adjacency list of x should finish in time proportional to the length of the adjacency list of $x \rightarrow y, z, z, z, z, z, z, y, y, z$. In particular, you cannot afford to spend time $O(|V|)$ for each x , since this would result in total time $O(|V| \times |V|) = O(|V|^2)$. So, the second time you see y (after the bunch of z 's) you should be able to decide that you have already seen it and should not re-include it in constant time. This is what the additional array of length V can help you with.

The additional array Help will be initialized to all entries 0 or NIL. As we are scanning the adjacency lists of G , and when we are working with the adjacency list of x , for each y seen in that adjacency list we may:

Either write the number x at the Help(y) entry of the help array. If it is the first time that we write the number x at the Help(y) entry then we should include y in the list of x without multiplicities. If the number x was already written at the Help(y) entry then we should not include y in the list of x without multiplicities.

Here's pseudocode:

Initialize:

for $v := 1$ to n $L'(v) := \text{NIL}$

for $v := 1$ to n $\text{Help}(v) := 0$

Main body:

for all $v \in V$

$y :=$ first element in $L(v)$

 while $y \neq \text{NIL}$

 if $\text{Help}(y) \neq x$ then

$\text{Help}(y) := x$

 append y to $L'(v)$

Or append x at the end of a list hanging off the y -th entry of the array Help(y). If it is the first time that we append the number x at the Help(y) list then we should include y in the list of x without multiplicities. If the number x was already appended and is therefore the tails If it is the first time that we append the number x then we should not include y in the list of x without multiplicities. Note that the array of lists Help(y) is the adjacency list of the reverse graph.

Question 4: DFS, Reachability, 25 Points

Let $G(V, E)$ be a directed graph in its adjacency list representation. Think of V as representing different points in a city. Think of an edge in E directed from u to v as a bus service that runs from u to v in the mornings. Let $G'(V, E')$ be another directed graph in its adjacency list representation. Think of V as representing the same points in the same city and of an edge in E' directed from u to v as a bus service that runs from u to v in the afternoons. Let $s \in V$ be a specified source, which you may think of as your home. Suppose that you start from your home in the morning. Give an algorithm that determines all the points in the city that you can reach by the end of the afternoon. (In this ideal world all bus services can transport you from point u to point v instantaneously, and morning bus services run continuously through the morning while afternoon bus services run continuously through the afternoon). Your algorithm should have running time $O(|V| + |E| + |E'|)$. You should give a clear description of your algorithm in English and use pseudocode, only if you find it necessary. You should argue about the running time.

We first run DFS on G starting from s . Let V_m be the set of all vertices visited by this DFS. This is precisely the set of vertices that can be reached using all combinations of morning schedules. So, at the end of the morning, we can be at any vertex of V_m that we choose. Where can we go further in the afternoon?

Let G^* be a graph whose vertex set is V and an additional supervertex s^* . The edges of G^* are as follows: (a) There is an edge from s^* to every vertex in V_m . (b) We include in G^* all the afternoon edges E' . We then run DFS on G^* . Let V_{total} be all the vertices visited by this second DFS. We output V_{total} .