COMP 271 Design and Analysis of Algorithms
2002 Spring Semester – Cunsheng Ding
Sample Solutions to the Midterm Exam

Problem 1: Suppose $T_1(n) = O(f(n))$ and $T_2(n) = O(f(n))$. Which of the following are true? Just write down your answer **without** giving any justification.

(a) $T_1(n) + T_2(n) = O(f(n))$

(b) $\frac{T_1(n)}{T_2(n)} = O(1)$

(c) $T_1(n) = O(T_2(n))$

Solution: (a)

Problem 2: A *local minimum* of an array $A[a..b]$ is an element $A[k]$ satisfying $A[k-1] \geq A[k] \leq A[k+1]$. We assume that $a \leq b-2$, $A[a] \geq A[a+1]$ and $A[b-1] \leq A[b]$; these conditions guarantee that a local minimum must exist. Design an algorithm for finding a local minimum of the array $A[a..b]$ which is substantially faster than the obvious liner-time $O(b-a)$ one, in the worst case. Justify the correctness of your algorithm and analyze its running time.
Solution: Divide the array $A[a..b]$ into two subarrays $A[a..c]$ and and $A[c+1..b]$, where $c = \lfloor (a+b)/2 \rfloor$. There are three cases:

1. $A[c-1] \geq A[c] \leq A[c+1]$: In this case, c is a local minimum.
2. $A[c-1] \geq A[c] \geq A[c+1]$: In this case, $A[c..b]$ must have a local minimum.
3. $A[c-1] \leq A[c] \leq A[c+1]$: In this case, $A[a..c]$ must have a local minimum.

The above analysis suggests the following divide-and-conquer algorithm.

```
LocalMinimun(A, a, b) {
  if b - a = 2, output a + 1;
  else {
          c = [(a+b)/2];
          if  A[c-1] \geq A[c] \leq A[c+1]; output c;
          else {
                  if  A[c-1] \geq A[c] \geq A[c+1]; LocalMinimun(A, c, b);
                   else {
                          LocalMinimun(A, a, c);
                         }
                 }
       }
}
```

Let $T(n)$ denote the running time of this algorithm for the array with length n. For simplicity, we assume that n is a power of 2. Clearly, we have

$$T(n) \leq T(n/2) + 4.$$

Hence $T(n) = O(\log_2 n) = O(\log_2(b-a+1))$. The correctness is justified by the three-case analysis above.

Problem 3: Design an algorithm for detecting whether any undirected graph is connected. Prove the correctness of your algorithm and analyze its running time. Your algorithm should be as efficient as possible. (The grading will be based on the efficiency and correctness of your algorithm.)

Solution: For any given undirected graph G, take any vertex s and name it as the source vertex. Run the BFS on (G, s). Then check the values $d[v]$ for each v. If $d[v] = \infty$ for some vertex v, output "disconnected". Otherwise output "connected".

The running time is the same as the BFS, i.e., $O(|V| + |E|)$. The correctness is justified as follows. If there is a path from s to every other vertex, clearly, the graph is connected. If there is no path from s to some other vertex, then G is disconnected. Hence G is connected if and only if there is a path from s to every other vertex. This is equivalent to $d[v] \neq \infty$ for every vertex v.

Other Solutions: The second solution is to check the color of all vertices. The third one is to count the total number of vertices visited.

Problem 4: Given a digraph G with n vertices and e edges, we run the DFS on G and obtain a DFS forest F_G. Answer the following questions:

(4.1) Prove that the number of edges in F_G is at most $\min\{n-1, e\}$.

Solution: Let $T_1, T_2, \cdots, T_k$ denote all the DFS trees, and let n_i denote the number of vertices in T_i. Then the number of edges in the tree is $n_i - 1$. Hence the total number of edges in F_G is

$$\sum_{i=1}^{k}(n_i - 1) = (\sum_{i=1}^{k} n_i) - k = n - k \leq n - 1,$$

because $k \geq 1$. On the other hand, the total number of edges in F_G is at most e, the total number of edges in G. Hence the number of edges in F_G is at most $\min\{n-1, e\}$.

(4.2) Is it possible that $e = n - 1$ and at the same time F_G has no edge at all? If your answer is yes, give an example. Otherwise prove that it is impossible.

Solution: Yes, it is possible. Let $G = (V, E)$ be a graph defined by

$$V = \{v_1, ..., v_n\}, E = \{(v_n, v_{n-1}), (v_{n-1}, v_{n-2}), \cdots, (v_2, v_1)\}.$$

If the source vertices are chosen in the order $v_1, v_2, ..., v_n$, the DFS forest consists of n DFS trees, each consisting of one vertex without any edge.

Problem 5: Recall that a DFS forest of a digraph G can be used to classify the edges of G into four types: tree edges, back edges, forward edges, and cross edges. We know that the time stamp $f[v]$ can be used to distinguish back edges from the other three types of edges, i.e., an edge (u, v) is a back edge if and only if $f[u] < f[v]$. Your tasks are the following.

(5.1) Consider the following statement.

"if $d[v] = d[u] + 1$, an edge (u, v) must be a tree edge."

If the statement is true, give a proof for it. Otherwise give a counter example to show that it is false.

Solution: It is true. If $d[v] = d[u] + 1$, then u was discovered just before v and thus the DFS discovered v from u. Hence (u, v) must be a tree edge.

(5.2) Consider the following statement.

"An edge (u, v) is a tree edge only if $d[v] = d[u] + 1$."

If the statement is true, give a proof for it. Otherwise give a counter example to show that it is false.

Solution: Consider the graph $G = (V, E)$ defined by

$$V = \{a, b, c\},\ E = \{(a, b), (a, c)\}.$$

Note that G is a DFS forest of itself with

$$(d[a], f[a]) = (1, 6), (d[b], f[b]) = (2, 3), (d[c], f[c]) = (4, 5).$$

(a, c) is a tree edge, but $d[a] = 1$ and $d[c] = 4$. Hence $d[c] \neq d[a] + 1$.

(5.3) If $d[u] + 2 \leq d[v]$ and $f[u] \geq f[v] + 2$, is it possible for an edge (u, v) to be a cross edge? Justify your answer.

Solution: It is impossible. If it were a cross edge, then the intervals $[d[u], f[u]]$ and $[d[v], f[v]]$ would be disjoint. Since $d[u] < d[v]$, we have

$$d[u] < f[u] < d[v] < f[v].$$

This is contrary to $f[u] \geq f[v] + 2$. Hence (u, v) cannot be a cross edge.

(5.4) If $d[u] + 2 \leq d[v]$ and $f[u] \geq f[v] + 2$, is it possible for an edge (u, v) to be a forward edge? Justify your answer.

Solution: It is possible. Consider the graph $G = (V, E)$ defined by

$$V = \{a, b, c\},\ E = \{(a, b), (b, c), (a, c)\}.$$

The $T = (V, E')$ is a DFS tree defined by

$$E' = \{(a, b), (b, c)\}$$

with

$$(d[a], f[a]) = (1, 6), (d[b], f[b]) = (2, 5), (d[c], f[c]) = (3, 4).$$

The edge (a, c) is a forward edge.