Name _____ Student No. _____

No aids allowed. Answer all questions on test paper. Use backs of sheets if necessary.

Total Marks: **60**

[30] 1. Explain briefly each the following:

(a) What is the difference between software *correctness* and *robustness*?

(b) What is the IEEE definition of software *maintainability*?

(c) Why is it that software "deteriorates as it ages"?

Solution: Correctness is accomplished by satisfying requirements, while robustness is accomplished by satisfying unstated (or implicit) requirements.

Maintainability: the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.

Software deteriorates as it ages: a system's conceptual integrity degrades as changes are made to it.

2. To find the shortest path in a network we can use a dynamic programming approach with an array of partial solutions $A(k, i, j)$, with the following semantics:

   $A(k, i, j)$:="the cost of the cheapest path from i to j with all intermediate nodes in $[k] = \{1, 2, \ldots, k\}$"

   We let $A(0, i, j)$ be the cost of a direct connection from i to j, if there is one, and ∞ otherwise.

[10] (a) Justify the following recurrence for $k > 0$:

$$A(k, i, j) = \min\{A(k-1, i, j), A(k-1, i, k) + A(k-1, k, j)\}$$

Solution: the shortest path from i to j, using only nodes in $[k]$ as intermediate nodes, either uses node k or it does not use it. If it does not use it, then $A(k, i, j) = A(k-1, i, j)$. If it does use it, then by the "subpath property of a shortest path" we have that $A(k, i, j) = A(k-1, i, k) + A(k-1, k, j)$. Now taking the min of the two values computes the minimum cost.

[10] (b) Suppose that we have $A(k, i, j)$ for all $k, i, j \in [n]$. How can we use that to get *the* shortest path for any pair of nodes i, j?

Solution: The shortest path is allowed to use all nodes in the graph; so the solution for every i, j is $A(n, i, j)$, where $n = |V|$.

[10] (c) The program that computes the distances is given by:

```
1: for i : 1..n do
2:    for j : 1..n do
3:       B(i, j) ⟵ C(i, j)
4:    end for
5: end for
6: for k : 1..n do
7:    for i : 1..n do
8:       for j : 1..n do
9:          B(i, j) ⟵ min{B(i, j), B(i, k) + B(k, j)}
10:      end for
11:   end for
12: end for
13: return  D ⟵ B
```

For the sake of saving memory, the program replaced the 3-dim array $A(*,*,*)$ with a 2-dim one $B(*,*)$. Explain why this space saving technique does not violate the consistency of the computation.

Solution: When the updating is done "in place" we might have the problem that instead of getting the value from the $k-1$ phase, we are getting an already updated value from phase k.

For example, from the recurrence we see:

$$A(4,4,5) = \min\{A(3,4,5), A(3,4,3) + A(3,3,5)\}$$

and therefore the procedure computes

$$B(4,5) = \min\{B(4,5), B(4,3) + B(3,5)\}$$

but note that $B(4,3)$ and $B(3,5)$ are updated before $B(4,5)$, and therefore what we really have is this:

$$\begin{aligned} B(4,5) &= \min\{B(4,5), B(4,3) + B(3,5)\} \\ &= \min\{A(3,4,5), A(\boxed{4},4,3) + A(\boxed{4},3,5)\} \end{aligned}$$

So the concern is that the space-saving procedure is no longer consistent. But note that $A(k,k,p) = A(k-1,k,p)$ and $A(k,q,k) = A(k-1,q,k)$ because the shortest path from k to any other node, or from any node to k, never contains k as an intermediate node. Thus, the computation is still consistent.