Final

Note

This is a closed-book exam. Each problem accounts for 10 points, unless otherwise marked.

Problems

1. The *Partition* procedure for the Quicksort algorithm discussed in class is as follows, where *Middle* is a global variable.

   Partition $(X, Left, Right)$;
   begin
   $pivot := X[left]$;
   $L := Left$; $R := Right$;
   while $L < R$ **do**
   while $X[L] \leq pivot$ and $L \leq Right$ **do** $L := L + 1$;
   while $X[R] > pivot$ and $R \geq Left$ **do** $R := R - 1$;
   if $L < R$ **then** $swap(X[L], X[R])$;
   $Middle := R$;
   $swap(X[Left], X[Middle])$
   end

   Find an adequate loop invariant for the main while loop, which is sufficient to show that after the execution of the last two assignment statements the array is properly partitioned by $X[Middle]$. Please express the loop invariant as precisely as possible, using mathematical notation. Explain why the invariant is sufficient.

2. Compute the *next* table as in the KMP algorithm for string $B[1..11] = bbaabbbaaba$. Please show how $next[8]$ and $next[11]$ are computed from using preceding entries in the table.

3. Give a binary de Bruijn sequence of 2^4 bits, which is a cyclic sequence of 2^4 bits $a_1 a_2 \cdots a_{2^4}$ such that each binary sequence of size 4 appears somewhere in the sequence. Explain how you can systematically produce the sequence.

4. Design an algorithm for determining whether a given *acyclic* directed graph $G = (V, E)$ contains a directed Hamiltonian path. (Note: a directed *Hamiltonian path* in a directed graph is a simple directed path that includes all vertices of the graph. An acyclic directed graph is one without directed cycles.) Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary. The more efficient your algorithm is, the more points you will be credited for this problem. Explain why your algorithm is correct and give an analysis of its time complexity.

5. What is wrong with the following algorithm for computing the minimum-cost spanning tree of a given weighted undirected graph (assumed to be connected)?

   > If the input is just a single-node graph, return the single node. Otherwise, divide the graph into two subgraphs, recursively compute their minimum-cost spanning trees, and then connect the two spanning trees with an edge between the two subgraphs that has the minimum weight.

6. Below is an algorithm discussed in class for determining the strongly connected components of a directed graph. Is the algorithm still correct if we replace the line "$v.high := \max(v.high, w.DFS\_Number)$" by "$v.high := \max(v.high, w.high)$"? Why? Please explain.

```
Algorithm Strongly_Connected_Components(G, n);
begin
    for every vertex v of G do
        v.DFS_Number := 0;
        v.component := 0;
    Current_Component := 0;  DFS_N := n;
    while v.DFS_Number = 0 for some v do
        SCC(v)
end

procedure SCC(v);
begin
    v.DFS_Number := DFS_N;
    DFS_N := DFS_N - 1;
    insert v into Stack;
    v.high := v.DFS_Number;
    for all edges (v, w) do
        if w.DFS_Number = 0 then
```

```
            SCC(w);
            v.high := max(v.high, w.high)
        else if w.DFS_Number > v.DFS_Number
                    and w.component = 0 then
                v.high := max(v.high, w.DFS_Number)
    if v.high = v.DFS_Number then
        Current_Component := Current_Component + 1;
        repeat
            remove x from the top of Stack;
            x.component := Current_Component
        until x = v
end
```

7. Finding a small vertex cover for an arbitrary undirected graph is difficult, but is much easier for trees; a *vertex cover* of a graph G is a set of vertices such that every edge in G is incident to at least one of these vertices. Design an efficient algorithm to find a minimum-size vertex cover for a given tree. Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary. The more efficient your algorithm is, the more points you will be credited for this problem. Explain why your algorithm is correct and give an analysis of its time complexity.

8. Below is a solution to the single-source shortest path problem using the dynamic programming approach, which we have discussed in class:

   Denote by $D^l(u)$ the length of a shortest path from v (the source) to u containing *at most* l edges; particularly, $D^{n-1}(u)$ is the length of a shortest path from v to u (with no restrictions).

$$D^1(u) = \begin{cases} length(v,u) & \text{if } (v,u) \in E \\ 0 & \text{if } u = v \\ \infty & \text{otherwise} \end{cases}$$

$$D^l(u) = \min\{D^{l-1}(u), \min_{(u',u)\in E}\{D^{l-1}(u') + length(u',u)\}\}, \quad 2 \le l \le n-1$$

   Please explain why the solution allows edges with a negative weight (as long as there is no cycle with a negative weight). How is this different from Dijkstra's algorithm? Please explain.

9. In the proof (discussed in class) of the NP-hardness of the clique problem by reduction from the SAT problem, we convert an arbitrary boolean expression in CNF (input of the SAT problem) to an input graph of the clique problem. Please illustrate the conversion by drawing the graph for the following boolean expression: $(w + \overline{z}) \cdot (w + x + \overline{y} + z) \cdot (\overline{w} + y + z)$. Explain how you have drawn the graph systematically.

10. Solve one of the following two problems. (Note: if you try to solve both problems, I will randomly pick one of them to grade.)

    (a) The knapsack problem is as follows.

    > Given a set X, where each element $x \in X$ has an associated size $s(x)$ and value $v(x)$, and two other numbers S and V, is there a subset $B \subseteq X$ whose total size is $\leq S$ and whose total value is $\geq V$?

    Prove that the knapsack problem is NP-complete.

    (b) The subgraph isomorphism problem is as follows.

    > Given two graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$, does G have a subgraph that is isomorphic to H? (Two graphs are isomorphic if there exists a one-one correspondence between the two sets of vertices of the two graphs that preserves adjacency, i.e., if there is an edge between two vertices of the first graph, then there is also an edge between the two corresponding vertices in the second graph, and vice versa.)

    Prove that the subgraph isomorphism problem is NP-complete.

Appendix

- The partition problem: given a set X where each element $x \in X$ has an associated size $s(x)$, is it possible to partition the set into two subsets with exactly the same total size?

  The partition problem is NP-complete.

- The Hamiltonian cycle problem: given an undirected graph G, does G have a Hamiltonian cycle? (A Hamiltonian cycle in a graph is a cycle that contains each vertex, except the starting vertex of the cycle, exactly once.)

  The Hamiltonian cycle problem is NP-complete.