Final

Note

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

Problems

1. A Hamiltonian path in a directed graph is a simple directed path that contains each vertex exactly once. Given an undirected complete graph, if we arbitrarily orient every edge in the graph (turning every undirected edge $\{u, v\}$ into either (u, v) or (v, u)), then the resulting directed graph must contain a Hamiltonian path. (Note: this problem is a reformulation of a problem from the midterm exam.)

2. Consider the *next* table as in the KMP algorithm for string $B[1..9] = abaababaa$.

1	2	3	4	5	6	7	8	9
a	b	a	a	b	a	b	a	a
-1	0	0	1	1	2	3	2	3

   Suppose that, during an execution of the KMP algorithm, $B[6]$ (which is an a) is being compared with a letter in A, say $A[i]$, which is not an a and so the matching fails. The algorithm will next try to compare $B[next[6] + 1]$, i.e., $B[3]$ which is also an a, with $A[i]$. The matching is bound to fail for the same reason. This comparison could have been avoided, as we know from B itself that $B[6]$ equals $B[3]$ and, if $B[6]$ does not match $A[i]$, then $B[3]$ certainly will not either. $B[5]$, $B[8]$, and $B[9]$ all have the same problem, but $B[7]$ does not. Please adapt the computation of the *next* table, reproduced below, so that such wasted comparisons can be avoided.

```
Algorithm Compute_Next (B, m);
begin
    next[1] := −1;  next[2] := 0;
    for i := 3 to m do
        j := next[i − 1] + 1;
        while B[i − 1] ≠ B[j] and j > 0 do
            j := next[j] + 1;
        next[i] := j
end
```

3. Design an algorithm that, given a set of integers $S = \{x_1, x_2, \ldots, x_n\}$, finds a nonempty subset $R \subseteq S$, such that

$$\sum_{x_i \in R} x_i \equiv 0 \pmod{n}.$$

(Note: such a subset R must exist.) Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary. Explain why the algorithm is correct and give an analysis of its time complexity. The more efficient your algorithm is, the more points you get for this problem.

4. Given as input a connected undirected graph G, a spanning tree T of G, and a vertex v, design an algorithm to determine whether T is a valid DFS tree of G rooted at v. In other words, determine whether T can be the output of DFS under some order of the edges starting with v. Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary. Explain why the algorithm is correct and give an analysis of its time complexity. The more efficient your algorithm is, the more points you get for this problem.

5. Consider Dijkstra's algorithm for single-source shortest paths as shown below. You may find in the literature two bounds, namely $O(|V|^2)$ and $O((|E| + |V|) \log |V|)$, for its time complexity. Why is this so? What does this difference imply?

```
Algorithm Single_Source_Shortest_Paths(G, v);
begin
    for all vertices w do
        w.mark := false;
        w.SP := ∞;
    v.SP := 0;
    while there exists an unmarked vertex do
        let w be an unmarked vertex s.t. w.SP is minimal;
        w.mark := true;
        for all edges (w, z) such that z is unmarked do
            if w.SP + length(w, z) < z.SP then
                z.SP := w.SP + length(w, z)
end
```

6. 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.

7. Let $G = (V, E)$ be a directed graph, and let T be a DFS tree of G. Prove that the intersection of the edges of T with the edges of any strongly connected component of G form a subtree (rather than two or more separate subtrees) of T.

8. 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
```

9. To prove that "P = NP" (which is unlikely though), it suffices to show that some NP-complete problem is in P. Why? Please explain.

10. A variant of the directed Hamiltonian path problem is as follows.

    > Given a directed graph $G(V, E)$ and $u, v \in V$, does G contain a Hamiltonian path from u to v? (A Hamiltonian path in a directed graph is a simple directed path that contains each vertex exactly once.)

    Prove that this variant of the directed Hamiltonian path problem is NP-complete. (Hint: one proof of NP-hardness is by reduction from the directed Hamiltonian cycle problem. Given the input graph $G(V, E)$ of the directed Hamiltonian cycle problem, we obtain the input graph $G'(V', E')$ of the directed Hamiltonian path problem as follows: Add to V two new vertices u and v to get V'. To get E', keep all edges in E, add a new edge (u, v') for every vertex v' in V, and for every edge (u', v') in E, add another new edge (u', v) if it has not been added yet.)

Appendix

- The KMP Algorithm:

```
Algorithm String_Match (A, n, B, m);
begin
    j := 1;  i := 1;
    Start := 0;
    while Start = 0 and i ≤ n do
        if B[j] = A[i] then
            j := j + 1;  i := i + 1
        else
            j := next[j] + 1;
            if j = 0 then
                j := 1;  i := i + 1;
        if j = m + 1 then Start := i − m
end
```

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

  The directed Hamiltonian cycle problem is NP-complete.