Final

Note

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

Problems

1. 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, so that such wasted comparisons can be avoided. Also, please give the values of the *next* table for the same string $B[1..9] = abaababaa$, according to the adaptation.

2. Please give a binary de Bruijn sequence of 2^4 bits, which is a *cyclic* sequence of 16 bits $a_1 a_2 \cdots a_{16}$ such that each binary sequence of size 4 appears somewhere in the sequence. Explain how you can systematically produce the de Bruijn sequence.

3. Design an algorithm that, given a weighted directed graph, detects the existence of a negative-weight cycle (the sum of the weights of its edges is negative). Please present your algorithm in adequate pseudocode and make assumptions wherever necessary. Explain why your algorithm is correct and give an analysis of its time complexity. The more efficient your algorithm is, the more points you will be credited for this problem.

4. 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.

5. Let $G = (V, E)$ be a connected weighted undirected graph and T be a minimum-cost spanning tree (MCST) of G. Suppose that the cost of one edge $\{u, v\}$ in G is *increased*; $\{u, v\}$ may or may not belong to T. Design an algorithm either to find a new MCST or to determine that T is still an MCST. Explain why your algorithm is correct and give an analysis of its time complexity. The more efficient your algorithm is, the more points you will be credited for this problem.

6. In the computation of SCCs by DFS, groups of vertices are discovered as SCCs one after another. The order of discovery depends on the vertex from which the search starts and the order via which the edges are considered from a vertex. Those vertices getting 1 as their component number are considered discovered first, those getting 2 are second, and so on.

   (a) Draw a directed graph with three SCCs, for which it is possible to produce any order of discovery via a DFS that always starts from the same SCC but may try the edges in different orders.

   (b) Show all 6 (= 3!) different orders of discovery by drawing 6 different DFS numberings of the graph, each of which starts from the same SCC (but maybe from a different vertex).

7. The most common approach to finding an augmenting path (if one exists) in a network with some given flow is breadth-first search (BFS). Please present such an algorithm in suitable pseudocode.

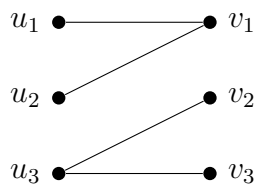
8. The cost of finding a key value in a binary search tree is linearly proportional to the depth/level of the node where the key value is stored, with the root considered to be at level 0. Obviously, for a key value that is known to be looked up more frequently, it is better stored in a node at a smaller level.

   Consider designing by dynamic programming an algorithm that, given the look-up frequencies of n key values, determines the least total cost for performing all the look-ups on an optimal binary search tree.

   (a) Formulate the solution using recurrence relations; let $F[1..n]$ be the look-up frequencies of the n key values $K[1..n]$, which are in sorted order.

   (b) Present the algorithm in suitable pseudocode, based on the previous recursive formulation. What is the time complexity of your algorithm?

9. The bipartite matching problem (the maximum-cardinality matching problem for bipartite graphs) can be reduced to the network flow problem, which in turn can be reduced to the linear programming problem. Please illustrate the reductions, using the graph below as input to the bipartite matching problem.



(a) Draw the network resulted from the conversion of the bipartite graph and show a maximum flow of the resulting network.

(b) Give the linear-programming objective function and constraints for the network.

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 hitting set problem is as follows.

> Given a collection C of subsets of a set S and a positive integer k, does S contain a hitting set for C of size k or smaller, that is, a subset $S' \subseteq S$ with $|S'| \leq k$ such that S' contains at least one element from each subset in C?

Prove that the hitting set problem is NP-complete.

(b) The traveling salesman problem is as follows.

> Given a weighted complete graph $G = (V, E)$ (representing a set of cities and the distances between all pairs of cities) and a number D, does there exist a circuit (traveling-salesman tour) that includes all the vertices (cities) and has a total length $\leq D$?

Prove that the traveling salesman problem is NP-complete.

Appendix

- The KMP algorithm (assuming *next*):

```
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 algorithm for computing the *next* table in the KMP algorithm:

```
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
```

- Below is a theorem useful for discovering an MCST of a connected weighted undirected graph $G = (V, E)$:

  Let V_1 and V_2 be a partition of V and $E(V_1, V_2)$ be the set of edges connecting nodes in V_1 to nodes in V_2. An edge with the minimum weight in $E(V_1, V_2)$ must be in an MCST of the given G.

- We say that problem/language L_1 is *polynomially reducible* to problem/language L_2 if there exists a conversion algorithm AC satisfying the following conditions:

  1. AC runs in polynomial time (deterministically).
  2. $u_1 \in L_1$ *if and only if* $AC(u_1) = u_2 \in L_2$.

- The vertex cover problem: given an undirected graph $G = (V, E)$ and an integer k, determine whether G has a vertex cover containing $\leq k$ vertices. (A *vertex cover* of G is a subset C of vertices such that every edge in G is incident to at least one of the vertices in C.)

  The vertex cover 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.