Final

Note

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

Problems

1. Two computers, each with a set of n integers, try to collaboratively find the n-th smallest element of the union of the two sets. The two computers can communicate by sending messages and they can perform any kind of local computation. A message can contain one element or one integer; a message with two numbers should be counted as two messages. Design an algorithm for the search task so that the number of messages exchanged is minimized. You can assume, for simplicity, that all the elements are distinct.

   Please present your algorithm in adequate pseudocode and make assumptions wherever necessary. Explain why your algorithm is correct and give an analysis of its message complexity (the number of messages exchanged). The more efficient your algorithm is, the more points you will get for this problem.

2. Construct a Huffman code tree for a text composed from seven characters A, B, C, D, E, F, and G with frquencies 13, 4, 2, 6, 18, 3, and 8 respectively.

3. 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
```

4. Describe an efficient implementation of the algorithm discussed in class for finding an Eulerian circuit in a graph. The algorithm should run in linear time and space. (Hint: the discovery of a cycle and that of the Eulerian circuits in individual connected components with the cycle removed, in the induction step, can be interweaved.)

5. Below is the algorithm discussed in class for determining the strongly connected components of a directed graph. The algorithm is based on depth-first search. During the exploration of the neighbors of a particular node v on which the SCC procedure is invoked, a neighboring node w may be found to have been visited. The neighbor w is reached from v either via a cross edge or a back edge. How can these two cases be distinguished and how does the algorithm correctly handle these two cases? 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

6. 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).

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

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

   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.

7. Design an algorithm that, given a set $\{x_1, x_2, \ldots, x_n\}$ of n positive integers and another integer m ($1 \le m \le n$), divides the n integers into m groups $C_1, C_2, \ldots, C_m$ such that the following sum is as small as possible:

$$\sum_{i=1}^{m}((|C_i| - 1) \times \sum_{x \in C_i} x).$$

   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.

8. Describe how the maximum(-cardinality) matching problem for bipartite graphs can be reduced to the network flow problem. In particular, you should explain why a maximum flow corresponds to a maximum matching.

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.

   (a) Please illustrate the conversion by drawing the graph that will be obtained from the following boolean expression:

$$(x + \overline{z}) \cdot (\overline{w} + x + \overline{y} + \overline{z}) \cdot (\overline{x} + \overline{y} + z) \cdot (w + y + z).$$

(b) The original boolean expression is satisfiable. As a demonstration of how the reduction works, please use the resulting graph to argue that it is indeed the case.

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 independent set problem is as follows.

        An independent set in an undirected graph is a set of vertices no two of which are adjacent. The problem is to determine, given a graph G and an integer k, whether G contains an independent set with $\geq k$ vertices.

        Prove that the independent set problem is NP-complete.

    (b) 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.

Appendix

- The clique problem: given an undirected graph $G = (V, E)$ and an integer k, determine whether G contains a clique of size $\geq k$. (A *clique* of G is a subgraph C of G such that every vertex in C is adjacent to all other vertices in C.)

  The clique problem is NP-complete.

- 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 complete.