Final

Note

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

Problems

1. The *next* table is a precomputed table (for $B = b_1 b_2 \cdots b_m$) that plays a critical role in the KMP algorithm. Under what condition regarding $b_1 b_2 \cdots b_i$, $2 \leq i \leq m$, will $next[i]$ get a 0 in the preprocessing? And under what condition can it be safely set to -1 (without missing a potential match when searching for B in another input string)?

2. Design an algorithm for finding an Eulerian circuit in an undirected graph. 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. (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.)

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. 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 *updated*; $\{u, v\}$ may or may not belong to T. Prove that T is still an MCST of G under any of the following two conditions:

   (a) $\{u, v\}$ belongs to T and its cost decreases or

   (b) $\{u, v\}$ does not belong to T and its cost increases.

   You may assume that the costs of all edges are distinct before and after the cost update to $\{u, v\}$.

5. 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.

6. Below is an algorithm, based on the dynamic programming approach, for solving the single-source shortest path problem.

```
Algorithm Single_Source_Shortest_Paths(length);
begin
    D[v] := 0;
    for all u ≠ v do
        if (v, u) ∈ E then
            D[u] := length(v, u)
        else D[u] := ∞;
    for l := 2 to n − 1 do
        for all u ≠ v do
            for all u' such (u', u) ∈ E do
                if D[u'] + length[u', u] < D[u] then
                    D[u] := D[u'] + length[u', u]
end
```

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).

In the for loop with index l iterating from 2 to $n-1$, it is possible that, for certain $l = k$, $D[u]$ acquires the value of $D^{k'}[u]$, where $k < k'$. Why? Please explain with an example.

7. Consider designing an algorithm by dynamic programming to determine the length of a longest common subsequence of two strings (sequences of letters). For example, "abbcc" is a longest common subsequence of "abcabcabc" and "aaabbbccc", and so is "abccc".

   (a) Formulate the solution using recurrence relations.

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

8. Every problem in P is polynomially reducible to any other non-trivial problem in P. Why? Please explain. (Note: a decision problem is *non-trivial* if there exists an input such that the answer is *yes* and there also exists an input such that the answer is *no*. In other words, a decision problem is non-trivial if its corresponding language is neither the universe nor the empty set.)

9. In the proof (discussed in class) of the NP-hardness of the 3SAT problem by reduction from the SAT problem, we convert an arbitrary Boolean expression in CNF (input of the SAT problem) to a Boolean expression in 3CNF (where each clause has exactly three literals).

   (a) Please illustrate the conversion by giving the Boolean expression that will be obtained from the following Boolean expression:

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

(b) The original Boolean expression is satisfiable. As a demonstration of why the reduction is correct, please use the resulting Boolean expression to show 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 subset sum problem (a variant of the knapsack problem) is as follows.

        The input is a multiset of numbers $\{a_1, a_2, \cdots, a_n\}$ and another number k. The problem is to determine whether the multiset contains a subset such that the sum of numbers in the subset is exactly k.

        Prove that the subset sum 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
```

- 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 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 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.