Final

Note

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

Problems

1. Suppose that you are given an algorithm/function called *subsetSum* as a *black box* (you cannot see how it is designed) that has the following property: if you input any sequence X of real numbers and an integer k, $subsetSum(X, k)$ will answer "yes" or "no", indicating whether there is a subsequence of the numbers whose sum is exactly k. Show how to use this black box to find the subsequence whose sum is k, if it exists.

   Please present your algorithm in adequate pseudocode and make assumptions wherever necessary. You should use the black box $O(n)$ times (where n is the size of the sequence). Note that a sequence of n numbers are stored in an array of n elements, where the elements are indexed from 1 through n.

2. Design an algorithm for finding an Eulerian circuit in an undirected graph (assumed to be simple, with at most one edge between any pair of vertices). 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 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 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. 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 currently being visited. 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 four SCCs and as few vertices as possible, for which there is only one order of discovery no matter where the DFS starts.

   (b) Draw a directed graph with four SCCs and as few vertices as possible, for which one may produce at least six (= 3!) different orders of discovery via DFSs that start from the same SCC but maybe from a different vertex (and try the edges in different orders).

6. Finding a small vertex cover is difficult for an arbitrary undirected graph, but is much easier for trees; a *vertex cover* of a graph is a subset of the vertices where every edge in the graph 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 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.

7. 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.

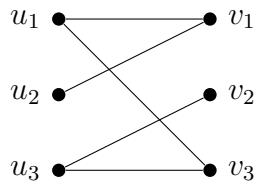
8. Consider a checkerboard with $n \times n$ squares, each square (i, j) (at row i and column j) is associated with a number $r(i, j)$ indicating the reward for visiting the square. A path through the checkerboard consists of a sequence of squares of the checkerboard, starting anywhere in the first row (Row 1) and terminating in the last row (Row n). Two consecutive squares in a path constitute a step of the path and should go from one row to the next row and either remain on the same column or go left or right by one column. The reward of a path is the sum of all the rewards associated with the squares along the path.

   Consider designing an algorithm by dynamic programming to determine the maximum reward among all paths through the checkerboard.

   (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?

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 about NP-completeness. (Note: if you try to solve both problems, I will randomly pick one of them to grade.)

    (a) The double satisfiability problem is as follows.

    > Given a Boolean expression in conjunctive normal form, determine whether it has two different satisfying assignments.

    Prove that the double satisfiability problem is NP-complete. (Hint: reduction from SAT; you may introduce new Boolean variables and add new clauses in the input conversion.)

    (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. (Hint: reduction from the vertex cover problem; an undirected edge is a set of two elements.)

Appendix

- The satisfiability (SAT) problem: given a Boolean expression in conjunctive normal form, determine whether it is satisfiable (i.e., it has a satisfying truth assignment to its Boolean variables).

  The SAT 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.

  The vertex cover problem is NP-complete.