Final

(June 20, 2002)

Note

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

Problems

1. The Knapsack Problem is defined as follows: Given a set S of n items, where the ith item has an integer size $S[i]$, and an integer K, find a subset of the items whose sizes sum to exactly K or determine that no such subset exists.

   Below is an algorithm for determining whether a solution to the problem exists.

```
Algorithm Knapsack (S, K);
begin
    P[0,0].exist := true;
    for k := 1 to K do
        P[0,k].exist := false;
    for i := 1 to n do
        for k := 0 to K do
            P[i,k].exist := false;
            if P[i-1,k].exist then
               P[i,k].exist := true;
               P[i,k].belong := false
            else if k - S[i] ≥ 0 then
                    if P[i-1,k-S[i]].exist then
                       P[i,k].exist := true;
                       P[i,k].belong := true
end
```

   (a) Modify the algorithm to solve a variation of the knapsack problem where each item has an unlimited supply. In your algorithm, please change the type of $P[i,k].belong$ into integer and use it to record the number of copies of item i needed.

(b) Design an algorithm to recover the solution recorded in the array P of the algorithm in (a).

2. Consider a chain A_1, A_2, A_3, A_4 of four matrices with dimensions 40×20, 20×30, 30×40, and 40×10, respectively. Compute (by immitating an algorithm based on dynamic programming) the minimum number of scalar multiplications needed to evaluate the product $A_1 A_2 A_3 A_4$.

3. 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. The more efficient your algorithm is, the more points you get for this problem. Explain why the algorithm is correct and give an analysis of its time complexity. (15 points)

4. 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?

```
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 such that 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
```

5. Consider an $m \times n$ matrix of non-negative integers. A path through the matrix consists of a sequence of cells of the matrix, starting anywhere in the first column (Column 1) and terminating in the last column (Column n). Two consecutive cells in a path constitute a step of the path and should go from one column to the next column and

either remain on the same row or go up or down one row. The weight of a path is the sum of the integers in the cells along the path. Design an algorithm to determine the minimum weight of such paths. Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary. Give an analysis of its time complexity. The more efficient your algorithm is, the more points you get for this problem. (Hint: one possible approach is to reduce the problem into the standard shortest path problem in graphs.) (15 points)

6. 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 *decreased*; $\{u, v\}$ 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. The more efficient your algorithm is, the more points you get for this problem. Explain why your algorithm is correct and analyze its time complexity.

7. Below is the main procedure for determining the biconnected components of an undirected graph. Is the algorithm still correct if we replace the last second line "$v.high :=$ $\max(v.high, w.DFS\_Number)$" by "$v.high := \max(v.high, w.high)$"? Why? Please explain.

```
procedure BC(v);
begin
    v.DFS_Number := DFS_N;
    DFS_N := DFS_N − 1;
    v.high := v.DFS_Number;
    for all edges (v, w) do
        if w is not the parent of v then
            insert (v, w) into Stack;
            if w.DFS_Number = 0 then
                BC(w);
                if w.high ≤ v.DFS_Number then
                    remove all edges from Stack
                        until (v, w) is reached;
                v.high := max(v.high, w.high)
            else
                v.high := max(v.high, w.DFS_Number)
end
```

8. Describe a connected undirected graph all of whose edges have distinct weights such that its minimum bottleneck weight spanning tree and minimum weight spanning tree are different. A bottleneck weight is defined as the maximum weight of an edge in the subgraph. (You do not need to design any algorithm for this problem.) (5 points)

9. Solve one of the following two problems. (Note: If you try to solve both problems, I will randomly pick one of them to grade.) (15 points)

   (a) The hitting set problem can be described 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 bin packing problem is as follows.

   > The input is a set of numbers $\{a_1, a_2, \cdots, a_n\}$ and two other numbers b and k. The problem is to determine whether the set can be partitioned into k subsets such that the sum of numbers in each subset is $\leq b$.

   Prove that the bin packing problem is NP-complete.

Appendix

- The vertex cover problem can be described as follows: 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 set of vertices such that every edge in G is incident to at least one of these vertices.
  The 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 problem is NP-complete.