CSC 364H L0101 Test 1 Summer 2000

Name _____ Student No. _____

No aids allowed. Answer all questions on test paper. Use backs of sheets for scratch work.

Total Marks: 30

[8] 1. The Simple Knapsack problem *search* (SKS) takes as input: $d, w_1, w_2, \ldots, w_d, C \in \mathbb{N}$ and requires as output a set $S_0 \subseteq \{1, \ldots, d\}$ such that $K(S_0) = M$, where:

$$M = \max_{S \subseteq \{1,\ldots,d\}} \{K(S) | K(S) \leq C\}$$

and $K(S) = \sum_{i \in S} w_i$. Recall that the greedy method for SKS produces an answer $\overline{M}$ such that $\overline{M} \leq C$ but possibly $\overline{M} < M$.

Lemma: Either $\overline{M} = M$, or $\overline{M} > \frac{1}{2}C$.

Your task is to prove this lemma by following the outline below:

Proof of Lemma: Assume that $\overline{M} \neq M$, and let S_0 be the result of greedy, i.e. a subset of $\{1, \ldots, d\}$ such that $K(S_0) = \overline{M}$.

(a) Show that there is at least one weight in S_0:

If $S_0 = \emptyset$, then it must be the case that $w_i > C$ for all i, in which case $\overline{M} = M$. Contradiction.

(b) Show that there is at least one weight not in S_0:

If all the weights are in S_0, then $M = w_1 + w_2 + \cdots + w_d = \overline{M}$. Contradiction.

(c) Show now that $\overline{M} > \frac{1}{2}C$ by considering the first weight, call it w_j, which has been rejected after at least one weight has been added:

If $w_j \leq \frac{1}{2}C$, then the sum of the weights which are already in is $> \frac{1}{2}C$, so $\overline{M} > \frac{1}{2}C$. If $w_j > \frac{1}{2}C$, then, since the ojects are ordered by greedy in non-increasing order of weights, the weights that are already in are $> \frac{1}{2}C$, so again $\overline{M} > \frac{1}{2}C$.

[9] 2. Recall the Job Scheduling with Deadlines and Profits problem, where each job takes unit time to complete:

Input: A list of jobs $(d_1, g_1), \ldots, (d_n, g_n)$ where $d_i \in \mathbb{N}$ is the deadline of job i and $g_i \in \mathbb{R}^+$ is the profit.

Problem: Find an optimal schedule S.

(a) Define a *schedule* S:

A schedule S is an array $S(1), \ldots, S(d)$, where $d = \max_i d_i$, s.t. if $S(t) = i$ then job i is scheduled at time t, and if $S(t) = 0$ then no job is scheduled at time t.

(b) Define a *feasible* schedule S:

S is feasible if two conditions are met: (i) if $S(t) = i > 0$, then $t \leq d_i$, and (ii) if $t_1 \neq t_2$ and $S(t_1) \neq 0$, then $S(t_1) \neq S(t_2)$.

(c) Define what it means for a feasible schedule S to be *optimal*:

$P(S) = \sum_{1 \leq t \leq d} g_{S(t)}$ is as large as possible, and the convention is that $g_0 = 0$.

(d) Define what it means for a schedule S' to *extend* a schedule S:

For all $1 \leq t \leq d$, if $S(t) \neq 0$, then $S(t) = S'(t)$.

(e) Define what it means for a schedule S to be *promising*:

S can be extended to an optimal schedule S_{opt}.

(f) Fill in the blanks in the following greedy algorithm for finding an optimal schedule:

Sort the jobs as follows: $\underline{g_1 \geq g_2 \geq \ldots \geq g_n}$
$d \longleftarrow \underline{\max_i d_i}$
for $t : 1..d$
    $S(t) \longleftarrow \underline{0}$
end for
for $i : 1..n$
    Find the largest t such that $\underline{(S(t) = 0 \text{ and } t \leq d_i)}$ and let $S(t) \longleftarrow i$
end for

[7] 3. In our dynamic programming solution to the Simple Knapsack Problem (SKS), we defined a Boolean array $R(i,j)$ for $0 \leq i \leq d$ and $0 \leq j \leq C$, as follows:

$$R(i,j) = \begin{cases} \mathsf{T} & \text{if } \exists S \subseteq \{1,\ldots,i\} \text{ s.t. } K(S) = j \\ \mathsf{F} & \text{otherwise} \end{cases}$$

(where the convention is that if $i = 0$, then $\{1,\ldots,i\} = \emptyset$).

First, give the recurrence used for computing this array:

(a) Initialization:
For all $0 \leq i \leq d$, $R(i,0) = \mathsf{T}$, and for all $0 < j \leq C$, $R(0,j) = \mathsf{F}$.

(b) For $i > 0$, $R(i,j) = \mathsf{T}$ iff
$R(i-1,j) = T$ or $(j \geq w_i$ and $R(i-1, j-w_i) = T)$.

Now, recall the General Knapsack Problem (GKS):

Input: $d, w_1, \ldots, w_d, v_1, \ldots, v_d, C \in \mathbb{N}$

Problem: Find $\max_{S \subseteq \{1,\ldots,d\}}\{V(S) | K(S) \leq C\}$, where $K(S) = \sum_{i \in S} w_i$, and $V(S) = \sum_{i \in S} v_i$.

To solve the general Knapsack problem, we start by computing the same Boolean array $R(i,j)$ that was used to solve SKS. Thus $R(i,j)$ ignores the values v_i, and only depends on the weights w_i. Next we define another array $V(i,j)$ that depends on the values v_i as follows: For $0 \leq i \leq d$, $0 \leq j \leq C$

$$V(i,j) = \max\{V(S) | S \subseteq \{1,\ldots,i\} \text{ and } K(S) = j\}$$

Give a recurrence for V:

$$V(i,j) = \begin{cases} V(i-1,j) & \text{if } (j < w_i \text{ or } R(i-1, j-w_i) = \mathsf{F}) \\ \max\{V(i-1,j), V(i-1,j-w_i) + v_i\} & \text{otherwise} \end{cases}$$

Here is a justification (which you didn't have to provide): if $j < w_i$, then object i cannot be included, so $V(i,j) = V(i-1,j)$. So assume that $j \geq w_i$. If $R(i-1, j-w_i) = \mathsf{F}$, then we cannot build j with the first i objects, so $V(i,j) = 0$, but if we cannot build j with the first i objects, then we cannot build it with the first $i-1$ objects either, so $V(i-1,j) = 0$ as well, so we can just let $V(i,j) = V(i-1,j)$ in this case. Finally, if $R(i-1, j-w_i) = \mathsf{T}$, then there are two possibilities: object i is included or object i is not included, and hence $V(i,j) = \max\{V(i-1,j), V(i-1,j-w_i) + v_i\}$.

[6] 4. The Activity Selection with Profits problem is defined as follows:

Input: A list of activities $(s_1, f_1, p_1), \ldots, (s_n, f_n, p_n)$, where each activity consists of a start time s_i and a finish time f_i (such that $0 \leq s_i < f_i$), and a profit $p_i > 0$ obtained if the activity is selected.

Problem: Find a set $S \subseteq \{1, \ldots, n\}$ of selected activities such that no two activities in S are in conflict and $\sum_{i \in S} p_i$ is maximal.

The dynamic programming solution that was given in lectures involved computing an array $A(j)$ whose value is the maximum profit that can be obtained by selecting a set of activities that completes by time u_j (where $u_1 < u_2 < \ldots < u_k$ is the list of all the distinct finish times, sorted in increasing order).

Assuming that the array just described has already been evaluated, show how to use the values in the array to compute the actual set S of selected activities:

Hint: consider what happens if $A(k) = A(k-1)$, and what happens if $A(k) > A(k-1)$.

Suppose $k > 0$. If $A(k) = A(k-1)$, then no activity has been scheduled to end at time u_k, so we proceed recursively to examine $A(k-1)$.

If, on the other hand, $A(k) \neq A(k-1)$, then we know that some activity has been scheduled to end at time u_k. We have to find out which one it is. We know that in this case

$$A(k) = \max_{1 \leq i \leq n} \{p_i + A(H(i)) | f_i = u_k\}$$

so we examine all activities i, $1 \leq i \leq n$, and output the (first) activity i_0 such that $A(k) = p_{i_0} + A(H(i_0))$ and $f_{i_0} \leq u_k$.

Now we repeat the procedure with $A(H(i_0))$.

We end when $k = 0$.