McGill University COMP360 Winter 2011 **Instructor: Phuong Nguyen**

Assignment 6 Solution

Question 1 (10pt) Let S be a non-empty set. Then a non-empty family $\mathcal{L}$ of subsets of S is said to be *nice* if it satisfies the following conditions:

1. **Inclusion property**: For every subsets $A, B \subseteq S$, if $B \in \mathcal{L}$ and $A \subset B$ then $A \in \mathcal{L}$ as well. (Note that the empty set $\varnothing$ is necessarily a member of $\mathcal{L}$.)

2. **Exchange property**: If $A \in \mathcal{L}$ and $B \in \mathcal{L}$ and $|A| < |B|$ (here $|A|$, $|B|$ denote the cardinalities of A and B, respectively), then there is some element $x \in (B - A)$ such that $A \cup \{x\}$ is an element of $\mathcal{L}$.

A subset A in $\mathcal{L}$ is called a *top* set if there is no other subset B in $\mathcal{L}$ such that $A \subset B$.

Here are some examples:

1. The family of all subsets of S is nice.

2. $S = \{a, b, c, d\}$ and $\mathcal{L} = \{\varnothing, \{a\}, \{b\}, \{a, b\}\}$. Then $\mathcal{L}$ is nice. But $\mathcal{L}' = \{\varnothing, \{a\}, \{b\}, \{a, b\}, \{a, b, c\}\}$ is not nice, because it violates the Inclusion property: $\{b, c\} \subset \{a, b, c\}$ and $\{a, b, c\} \in \mathcal{L}'$ but $\{b, c\} \notin \mathcal{L}'$.

3. $S = \{a, b, c, d, e\}$. $\mathcal{L}_1 = \{\varnothing, \{a\}, \{b\}, \{a, b\}\}$ and $\mathcal{L}_2 = \{\varnothing, \{c\}, \{d\}, \{e\}, \{c, d\}, \{d, e\}, \{e, c\}, \{c, d, e\}\}$ are both nice. But $\mathcal{L}_3 = \{\varnothing, \{a\}, \{b\}, \{a, b\}, \{c\}, \{d\}, \{e\}, \{c, d\}, \{d, e\}, \{e, c\}, \{c, d, e\}\}$ is not nice, because it violates the Exchange property: $\{a, b\}$ and $\{c, d, e\}$ are both in $\mathcal{L}_3$ and $|\{a, b\}| < |\{c, d, e\}|$, but there is not element x in $\{c, d, e\}$ such that $\{a, b\} \cup \{x\}$ is a member of $\mathcal{L}_3$.

Now suppose that S is a set of n elements, $n > 0$, and let $\mathcal{L}$ be a nice family of subsets of S. Suppose that each element x of S has a positive weight $w(x)$. The weight of a subset A of S is defined to the be total weight of all elements in A:

$$w(A) = \sum_{x \in A} w(x)$$

The problem here is to fine a subset in $\mathcal{L}$ that has maximum total weight. Notice that any such subset is necessarily a top set.

You are asked to solve this problem efficiently by a greedy algorithm. Note that $\mathcal{L}$ can potentially have up to 2^n members, so we don't want to go through all of them. In fact, your algorithm must be a polytime algorithm (assuming that the function $t(n)$ below is a polynomial).

a) Give a greedy algorithm that finds a subset in $\mathcal{L}$ of maximum total weight. Prove that your algorithm is correct.

b) To analyze the running time of your algorithm, we assume that checking whether a subset A is a member of $\mathcal{L}$ takes time $t(n)$. What is the running time of your algorithm in terms of n and $t(n)$? (State your answer using $\mathcal{O}$ notation.)

Solution

(a): The Algorithm [4pt] The idea of the greedy algorithm is to go through the elements of S in non-increasing order of weight and including them in the output set if the resulted set belongs to $\mathcal{L}$.

1. Sort the elements of S in non-increasing order of their weights: $w(1) \geq w(2) \geq \ldots \geq w(n)$.
2. $A \leftarrow \varnothing$ % The output
3. For $i = 1$ to n do
4. If $A \cup \{i\} \in \mathcal{L}$ then $A \leftarrow A \cup \{i\}$ End If
5. End For
6. Return A

Proof of Correctness [5pt] Now we prove that the greedy algorithm does return a set in $\mathcal{L}$ of maximum weight. Let A_i be the set A after the i-th iteration. In particular, $A_0 = \varnothing$ and A_n is the output of the algorithm.

We say that A_i is "promising" if there is a maximum weight subset in $\mathcal{L}$ that extends A_i using only elements in $\{i+1, i+2, \ldots, n\}$. We will show that A_i are promising, and the fact that A_n is promising implies that it must be an optimal subset.

The proof is by induction on i.

Base case: $i = 0$. Since $A_0 = \varnothing$, any subset in $\mathcal{L}$ extends A_0 using only elements in $\{1, 2, \ldots, n\}$.

Induction step: Suppose that A_i is promising, we show that A_{i+1} is also promising.

Since A_i is promising, there is a maximum weight subset OPT in $\mathcal{L}$ that extends A_i using only elements in $\{i+1, i+2, \ldots, n\}$, i.e.,

$$A_i \subseteq OPT \subseteq A_i \cup \{i+1, i+2, \ldots, n\}$$

We show that there is a maximum weight subset OPT' that satisfies

$$A_{i+1} \subseteq OPT' \subseteq A_{i+1} \cup \{i+2, i+3, \ldots, n\} \tag{1}$$

We consider the following cases:

Case I: $i+1 \notin A_{i+1}$ (i.e., $A_{i+1} = A_i$).

This only happens when $A_i \cup \{i+1\} \notin \mathcal{L}$. Then $i+1 \notin OPT$, since otherwise $A_i \cup \{i+1\} \subseteq OPT$ and by the Inclusion Property, $OPT \in \mathcal{L}$ implies that $A_i \cup \{i+1\} \in calL$, a contradiction.

So $i+1 \notin OPT$, and so OPT extends A_{i+1} using only elements from $\{i+2, i+3, \ldots, n\}$.

Case II: $i+1 \in A_{i+1}$, i.e., $A_{i+1} = A_i \cup \{i+1\}$.

This implies that $A_{i+1} \in \mathcal{L}$. We consider two subcases:

Case IIa: $i+1 \in OPT$. Then OPT also extends A_{i+1} (using only elements from $\{i+2, i+3, \ldots, n\}$), and we are done.

Case IIb: $i+1 \notin OPT$. We show how to modify OPT to obtain OPT' that satisfies (1). Let $x_1, x_2, \ldots, x_k$ be all elements in $OPT - A_i$, i.e.,

$$OPT = A_i \cup \{x_1, x_2, \ldots, x_k\}$$

where $i+2 \leq x_1 < x_2 < \ldots < x_k$.

Now apply the Exchange Property for A_{i+1} and OPT, there must some x_j so that $A_{i+1} \cup \{x_j\} \in \mathcal{L}$. Apply the Exchange Property again for $A_{i+1} \cup \{x_j\} \in \mathcal{L}$ and OPT, there must be another x_ℓ so that $A_{i+1} \cup \{x_j, x_\ell\} \in \mathcal{L}$. Doing this $k-1$ times, we conclude that there is some x_t so that $(A_{i+1} \cup \{x_1, x_2, \ldots, x_k\} - \{x_t\}) \in \mathcal{L}$. Let $OPT' = A_{i+1} \cup \{x_1, x_2, \ldots, x_k\} - \{x_t\}$, then

$$w(OPT') = w(OPT) - w(x_t) + w(i+1) \geq w(OPT)$$

(because $w(i+1) \geq w(x_t)$). Since OPT has maximum weight, it must be the case that $w(OPT') = w(OPT)$ and hence OPT' also has maximum weight. Clearly OPT' satisfies (1). QED

(b) [1pt] The sorting step takes time $\mathcal{O}(n \log n)$. There are n iteration. Each iteration takes time $t(n)$. So in total the running time of the algorithm is $\mathcal{O}(n \log n + nt(n))$.

Question 2 (10pt) Consider the following variant of the Load Balancing problem, called Weighted Load Balancing problem here. The input is a set of k (normal) processors and m *fast* processors, that can run twice as fast as the normal processors. There are n jobs where each job i has a duration t_i and needs to be processed on one processor. On a normal processor job i takes up a contiguous duration of t_i units of time, but on a fast processor it takes up only $t_i/2$ units of time. The problem is to schedule jobs on processors in such a way that minimizes the maximum processing time (that is, the load) of the processors.

For example, suppose that there are one normal processor and one fast processor (i.e., $k = m = 1$), and there are three jobs with durations $t_1 = 15, t_2 = 2, t_3 = 6$. The the best schedule is to have job 1 on the fast processor, jobs 2 and 3 on the normal processor (maximum load is 8 here). On the other hand, if we jobs 1 and 2 on the fast processor and job 3 on the normal processor, the maximum load is 8.5.

You are asked to give a greedy approximation algorithm for this problem that achieves approximation ratio 2. That is, the output schedule must have maximum load at most twice the optimal maximum load. Prove that this is indeed the case. Your algorithm must run in time polynomial in m, k, n and $\sum_{i=1}^{n} \log t_n$. (Here all t_i are positive integers.)

Solution

The algorithm [5pt] We follow the approach for the Load Balancing problem discussed in class. Note that here the load of a fast processor is half of the total duration of all jobs assigned to it. The idea is to sort the jobs in non-increasing order of duration, and assign each job to the processor that has smallest total load so far.

1. sort the jobs so that $t_1 \geq t_2 \geq t_3 \geq \ldots \geq t_n$
2. for i from 1 to n do
3. assign job i to the processor with smallest load so far; if there are both fast and normal processors with smallest load, break tie by choosing a fast processor
4. update the load of the processor
5. end for

The sorting can be done in polytime, and each loop can be done in polytime, so overall the algorithm runs in polytime.

Proof of approximation ratio [5pt] Now we show that the output schedule has maximum load at most twice that of the optimal value. First note that if the number of jobs is at most the number of processor, then the algorithm assigns one job to each processor, with the longest jobs to the fast processors. Therefore the output is an optimal schedule.

Now suppose that there are more jobs than processors. Let m be the number of processors, and k be the number of fast processors. So there are $(m-k)$ normal processors. Let *opt* be the maximum load in an optimal schedule. We need to show that the maximum load in our schedule is at most $2opt$.

Observe that

$$opt(2k+m-k) \geq \sum_{i=1}^{n} t_i$$

so

$$opt \geq \frac{1}{m+k} \sum_{i=1}^{n} t_i$$

Suppose that processor 1 is the processor with maximum load in our schedule. Let t_j be the last job assigned to it, and suppose that at the time before t_j is assigned, the load on this processor is T. So $T + t_j$ is the total load on this processor, and we have to show that

$$T + t_j \leq 2opt$$

It suffices to show that $T \leq opt$ and $t_j \leq opt$.

For the first inequality, at the time before t_j is assigned to processor 1, the total load on this processor is the smallest among all processors, therefore the total duration of all jobs that have been assigned so far is at least

$$(2k+m-k)T = (m+k)T$$

Hence

$$(m+k)T < \sum_{i=1}^{n} t_i$$

so

$$T < \frac{1}{m+k} \sum_{i=1}^{n} t_i$$

Consequently $T < opt$.

For the second inequality, note that the first m jobs are assigned one to each of the processors. Therefore $j \geq m+1$. Now consider an arbitrary optimal schedule. Then there must be at least one processor that are assigned at least two jobs from jobs $1, 2, \ldots, j$. If this processor is a normal processor then its load is at least $2t_j$. Otherwise its load is at least $2t_j/2 = t_j$. In any case, the maximum load in this schedule is at least t_j. In other words, $t_j \leq opt$. QED