

CSC 373 H 1 Y — Summer 2006

University of Toronto — St. George Campus

Lecture Summary for Week 3

Schedule to Minimize Lateness Section 4.2 in the text. (Although the proof given here is slightly different.)

Given n jobs J_1, \dots, J_n , where job J_i has deadline d_i and processing time t_i . There is a single processor that can process only one job at a time.

A schedule is just an ordering of the jobs. For a schedule S , let $f_S(J_i)$ be the finishing time of job J_i . If a job J_i misses its deadline, i.e. $f_S(J_i) > d_i$, its lateness is defined to be $f_S(J_i) - d_i$. The lateness of the schedule is defined to be the maximum lateness of all the late jobs in the schedule.

The problem is to schedule the job with smallest possible lateness. In other words, we want to minimize the maximum lateness of all late jobs.

The Greedy Algorithm

The idea is to get the early jobs done first.

1. Sort the jobs so that

$$d_1 \leq d_2 \leq \dots \leq d_n$$

2. Output the schedule J_1, \dots, J_n .

Proof of Correctness

We follow the same framework for the proofs of correctness of other greedy algorithms. Let

$$P_i = \langle J_1, \dots, J_i \rangle$$

We show that every P_i is “promising” in the sense that there is an optimal schedule that extends P_i , i.e. of the form

$$\langle J_1, J_2, \dots, J_i, J'_{i+1}, \dots, J'_n \rangle$$

where J'_{i+1}, \dots, J'_n is a permutation of J_{i+1}, \dots, J_n . Then, the output P_n must be itself an optimal schedule.

Claim: P_i is promising for every $0 \leq i \leq n$.

We prove the Claim by induction on i .

Base case: $i = 0$, $P_0 = \emptyset$. The Claim holds because every (optimal) schedule extends the empty sequence.

Induction step: Assume that P_i is promising. We show that P_{i+1} is also promising. Since P_i is promising, there is an optimal schedule of the form

$$OPT = \langle J_1, \dots, J_i, J'_{i+1}, \dots, J'_n \rangle$$

Consider two cases:

Case 1 $J'_{i+1} = J_{i+1}$. Then OPT also extends P_{i+1} , so P_{i+1} is promising, and we are done.

Case 2 $J'_{i+1} \neq J_{i+1}$. Then J_{i+1} must occur at some position $k + 1$ in OPT , where $k > i$. So

$$OPT = \langle J_1, \dots, J_i, J'_{i+1}, \dots, J'_k, J_{i+1}, J'_{k+2}, \dots, J'_n \rangle$$

Modify OPT by placing J_{i+1} just after J_i : let

$$OPT' = \langle J_1, \dots, J_i, J_{i+1}, J'_{i+1}, \dots, J'_k, J'_{k+2}, \dots, J'_n \rangle$$

Then OPT' extends P_{i+1} . We show that OPT' is an optimal schedule. In fact, we will show that the maximum lateness of the jobs in OPT' will be at most as large as that of OPT .

Notice that OPT and OPT' differ only in the block

$$J'_{i+1}, \dots, J'_k, J_{i+1} \tag{1}$$

for OPT and

$$J_{i+1}, J'_{i+1}, \dots, J'_k \tag{2}$$

for OPT' . Also, since J_{i+1} has smallest deadline amongs $J'_{i+1}, \dots, J'_k, J_{i+1}$, the maximum lateness among the jobs in (1) is that of J_{i+1} . The following exercise is to show that for the schedule OPT' , all the jobs in (2) has lateness \leq the lateness of J_{i+1} in OPT .

Exercise Show that in OPT' , all jobs in (2) has lateness at most as large as the lateness of J_{i+1} in OPT .

Longest Common Subsequence

Input: Two sequences (arrays) $X[1 \dots n]$, $Y[1 \dots m]$.

Output: A longest common subsequence of X , Y .

Let

$$S_{i,j} = LCS(X[1 \dots i], Y[1 \dots j])$$

Observation:

$$S_{n,m} = \begin{cases} S_{n-1,m-1} \circ X[n] & \text{if } X[n] = Y[m] \\ \text{longer of } \{S_{n-1,m}, S_{n,m-1}\} & \text{otherwise} \end{cases}$$

A Recursive Program

LCS(X, Y, n, m):

1. If $n = 0$ or $m = 0$: return \emptyset
2. If $X[n] = Y[m]$ return $LCS(X, Y, n - 1, m - 1) \circ X[n]$
3. Else
4. $L_1 = LCS(X, Y, n - 1, m)$
5. $L_2 = LCS(X, Y, n, m - 1)$
6. If $|L_1| > |L_2|$
7. return L_1
8. Else
9. return L_2
10. End If
11. End If

Running Time (Assume $n \leq m$):

$$\Omega(2^n)$$

Reason: Repeated calls to small (sub)problems

$$LCS(X, Y, i, j) \quad \text{for } 1 \leq i \leq n, 1 \leq j \leq m$$

There are only $n \times m$ subproblems.

Improvement: Solve the subproblems *once* and *store* their solutions !

LCS2(X, Y, n, m):

1. S : $n \times m \times n$ array % For storing solution.
2. For $j = 1$ to m do $S[0, j] \leftarrow \emptyset$
3. For $i = 1$ to n do $S[i, 0] \leftarrow \emptyset$
4. For $i = 1$ to n do
5. For $j = 1$ to m do
6. If $X[i] = Y[j]$ do $S[i, j] \leftarrow S[i - 1, j - 1] \circ X[i]$
7. Else $S[i, j] \leftarrow \text{longer of } \{S[i - 1, j], S[i, j - 1]\}$
8. End If
9. End For
10. End For
11. Return $S[n, m]$

Space: $n \times m \times n$

Observation: No need to store $S[i, j]$

Can just store the *lengths* of $S[i, j]$, and *recover* $S[n, m]$ at the end. Will continue next week.