Computation of $p(j)$ in Weighted Interval Scheduling

Clarification from 2010-Jul-12 lecture

**Setting:** We are given $n$ jobs, each of which has a start time and finish time, $[\text{starttime}(i), \text{finishtime}(i)]$. Let us define

$$p(j) = \max_i \{\text{finishtime}(i) \leq \text{starttime}(j)\}.$$ 

Below, we give a $O(n \log n)$ time algorithm for finding $p(\cdot)$ values.

We sort the jobs according to the start times and finish times both (in two separate lists of course). We go down the finish-time list, and assign $p(j)$ values for the jobs in the start-time list. The idea is to keep two separate pointers, and keep them in such a way that we can assign the $p(\cdot)$ value of the job of second pointer equal to the job of the first pointer. Details below.

Let $\text{starttime}(i)$ denote the start time of job $i$, and $\text{finishtime}(i)$ denote the finish time of job $i$.

Sort the jobs according to start times. Let $s_1$ be the first job (with smallest start time), $s_2$ be the second job and so on. Therefore, $\text{starttime}(s_1) \leq \text{starttime}(s_2) \leq \cdots \leq \text{starttime}(s_n)$.

Sort the jobs according to finish times. Let $f_1$ be the first job (with smallest finish time), $f_2$ be the second job and so on. Therefore, $\text{finishtime}(f_1) \leq \text{finishtime}(f_2) \leq \cdots \leq \text{finishtime}(f_n)$.

Let us also assume we have a job called 0 with finish time and start time both equal to 0. So, $s_0 = 0$, and $f_0 = 0$ in the above sorted order.

\begin{verbatim}
1 i = 1, j = 1
2 while ( i <= n && j <= n ) {
3     if ( \text{finishtime}(f_i) \leq \text{startime}(s_j) ) {
4         i = i + 1
5     } else if ( \text{finishtime}(f_i) > \text{startime}(s_j) ) {
6         \text{p}(s_j) \leftarrow f_{i-1} // if i - 1 = 0, then f_{i-1} = 0 by the assumption of an extra job above.
7         j = j + 1
8     }
9 }
\end{verbatim}

**Lemma 1.** The above procedure assigns the $p(k)$ values correctly for $k \in \{1, 2, ..., n\}$.

**Proof.** Let us focus on a job $k$. Let us say it occurs at index $j'$ in the sorted start-time list, that is $s_{j'} = k$. There are two cases to analyze. If $p(s_{j'}) = 0$, then we can easily see that the assigned value is correct. This is because if $p(s_{j'})$ is assigned 0, then $i$ has not been increased yet (and $p(s_{j'})$ was assigned $f_0$ which is equal to 0), and $\text{finishtime}(f_i) > \text{startime}(s_{j'})$ (the condition in the else if part of the loop). In this case, $p(s_{j'})$ must be 0.

If $p(s_{j'})$ is assigned some value other than 0, it must be $f_{i'-1}$ for some $i' > 1$. At the time of assigning the value (let us call that time $t'$), $i$ must have been equal to $i'$. Consider the first instance of time when $i$ was increased from $i' - 1$ to $i'$. Call this time $t$ (note that $t \leq t'$). At time $t$, $\text{finishtime}(f_{i'-1})$ must have been less than or equal to $\text{startime}(s_{j'})$ (where $j''$ is the value of $j$ at time $t$). We also note that $j'' \leq j'$ since $j$ is an increasing index. Therefore, we have

$$\text{finishtime}(f_{i'-1}) \leq \text{startime}(s_{j''}) \leq \text{startime}(s_{j'})$$,

where the first inequality follows because $i$ was increased at time $t$ from $i' - 1$ to $i'$ and the second inequality holds because the value of $j$ at time $t$ can only be at most the value of $j$ at time $t'$ and jobs in the s-list are

Computation of $p(j)$ in Weighted Interval Scheduling – Page 1
sorted by start time. Also notice that when \( j' \) was assigned \( p(j') \) equal to \( i' - 1 \) in the “else if” condition, we had

\[
\text{finish}(f_{i'}) > \text{start}(s_{j'}).
\]

Combining the two relations above, we have

\[
\text{finish}(f_{i'-1}) \leq \text{start}(s_{j''}) \leq \text{start}(s_{j'}) < \text{finish}(f_i).
\]

It follows that the \( p(j') \) value is correct, since it must be the largest finish time which is still at most \( \text{start}(s_{j'}) \).

**Lemma 2.** The above procedure takes \( O(n) \) time.

**Proof.** In every iteration of the while loop, either \( i \) is incremented by 1, or \( j \) is incremented by 1. This can happen for at most \( 2n \) iterations, since the maximum values of either of those in \( n \).