CS580 Algorithm Design, Analysis, and Implementation

Lecture notes, Feb 03, 2022
Wufei Ma

1 Dynamic Programming

Subsequence. Given two sequences $X = x_1 x_2 \dots x_m$ and $Z = z_1 z_2 \dots z_k$. Z is a subsequence of X if there is an increasing sequence of indices $i_1 i_2 \dots i_k$ such that for all $j = 1 \dots k$, $x_{i_j} = z_j$.

Common sequence. Given two sequences X and Y, we say Z is a common subsequence of X and Y if Z is a subsequence of X and Z is a subsequence of Y.

Longest common sequence (LCS). Given two sequences X and Y, we need to find the longest common subsequence.

Given $X = x_1 \dots x_m$, the i-th prefix of X is $X_i = x_1 \dots x_i$.

Optimal substructure of LCS. Let $X = x_1 \dots x_m$ and $Y = y_1 \dots y_n$ be sequences and $Z = z_1 \dots z_k$ be the LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ and Z is an LCS of X_{m-1} and Y, or $z_k \neq y_n$ and Z is an LCS of X and Y_{n-1}.

A recursive solution. Let $\mathrm{C}(i,j)$ be the length of an LCS of prefixes X_i and Y_j. We have

$$\mathrm{C}(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \mathrm{C}(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max\{\mathrm{C}(i, j-1), \mathrm{C}(i-1, j)\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

```
Function LCS(X, Y):
    m, n = len(X), len(Y)
    for i = 1 to m do:
        C[i, 0] = 0
    for j = 0 to n do:
        C[0, j] = 0
    for i = 1 to m:
        for j = 1 to n:
            if x[i] == y[j]:
                C[i,j] = C[i-1,j-1] + 1
```

```
                B[i,j] = "leftup"
            else if C[i-1,j] >= C[i,j-1]:
                C[i,j] = C[i-1,j]
                B[i,j] = "up"
            else:
                C[i,j] = C[i,j-1]
                B[i,j] = "left"
            endif
        endfor
    endfor
```

To find the LCS, we use

```
Function PrintLCS(B, X, i, y):
    if i == 0 or j == 0:
        return
    if B[i,j] == "leftup":
        PrintLCS(B, X, i-1, j-1)
        print X[i]
    elseif B[i,j] == "up":
        PrintLCS(B, X, i-1, j)
    else:
        PrintLCS(B, X, i, j-1)
```

2 Greedy Algorithm

An example. Given a set S of objects A_i with weights $w_i > 0$. Choose a subset $T \subseteq S$ such that $\sum_{A_i \in T} w_i \leq W$ and $|T|$ is maximized.

The greedy strategy is to choose the objects with the smallest weights.

A scheduling problem. Let $S = \{1, \ldots, n\}$ be a set of activities. Activity i has start time s_i and finish time f_i. If activity i is scheduled, it occupies the resource in the time interval $[s_i, f_i)$ with $s_i < f_i$. The problem is to maximize the number of activities scheduled.

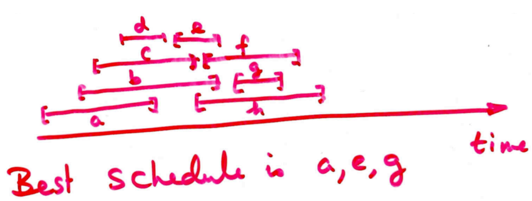

Figure 1: A scheduling problem.

The strategy is to always choose the one that ends earliest.

1. Sort the activities such that $f_i \leq f_j$ if $i < j$.
2. $T = \{1\}$, last $= 1$.
3. For $i = 2$ to n, if $f_{\text{last}} \leq s_i$ then $T = T \cup \{i\}$ and last $= i$.

Proof of correctness. This greedy algorithm schedules the largest number of activities.

1. First, observe that activity 1 (with the earliest finish time) can always be chosen. This is because the first activity of any schedule can be replaced by activity 1 without any conflict and gives the same number of activities.
2. After removing all activities i that conflict with activity 1, this correctness of the algorithm is shown recursively.

Proof by contradiction. Assume the greedy algorithm is not optimal. Let $i_1, \ldots, i_k$ be the greedy algorithm solution. Let $j_1, \ldots j_m$ be the jobs in the optimal solution with $i_1 = j_1, \ldots, i_r = j_r$ for the largest r. In this case we can substitute j_{r+1} with i_{r+1} and finds another optimal solution with first $r + 1$ jobs same as the greedy solution, which is contradicted to our assumption.

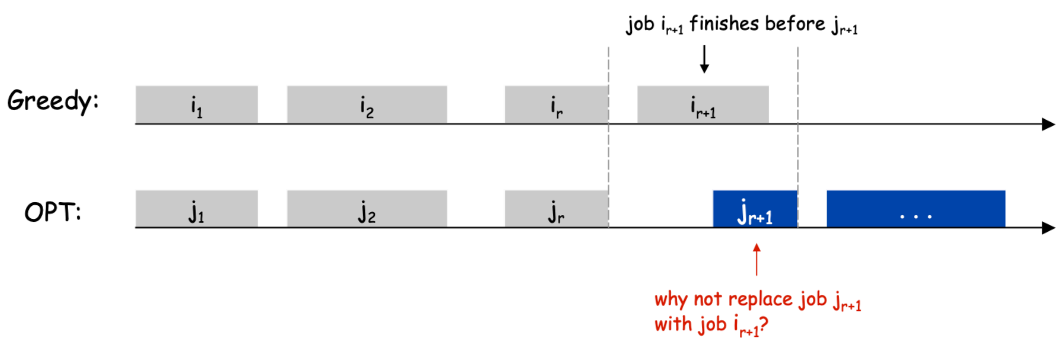

Figure 2: Proof by contradiction.

Interval partitioning. Lecture j starts at s_j and finishes at f_j. The goal is to find the minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Depth. The depth of a set of open intervals is the maximum number that contains at any given time. The number of classrooms needed must be larger or equal to the depth.

Greedy algorithm. Consider lectures in increasing order of start time. Assign lectures to any compatible classroom. We open a new classroom is there's no compatible classroom.

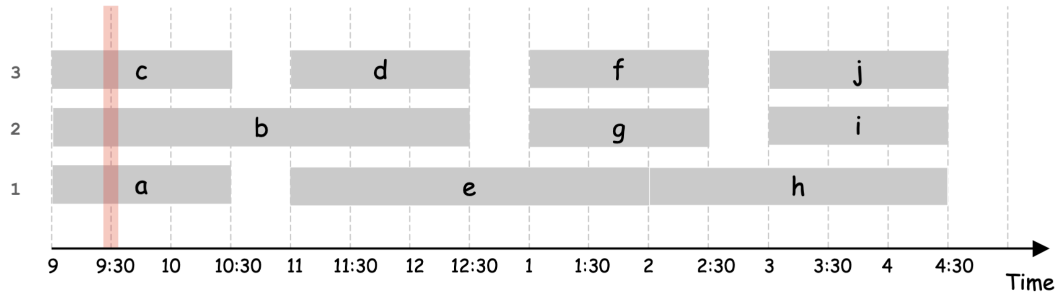

Figure 3: Depth of the interval partitioning problem.

The greedy algorithm is optimal. Let d be the number of classrooms that the greedy algorithm allocates. Classroom d is opened because we need to schedule lecture j that is incompatible with all other $d-1$ classrooms. These d jobs finish after s_j and starts no later than s_j. Thus we have d lecture overlapping at time $s_j + \epsilon$. Therefore, all schedules use at least d classrooms and the schedule found by the greedy algorithm is optimal.