CS21003 - Tutorial 6

Solution Sketch

1. Consider the following variation of interval scheduling problem – we have n requests labeled $1, \ldots, n$ with each request a_i having a start time s_i and a finish time f_i. Each request a_i also comes with a value, or weight v_i. The goal is to select a subset $S \subseteq \{1, \ldots, n\}$ of mutually compatible intervals, so as to maximize the sum of the values of the selected intervals $\sum_{i \in S} v_i$. [**Hint:** The complexity is much better than $O(n^3)$]

   $\Rightarrow$ Let us first sort the requests in increasing order of their finish times. To take care of compatibility, for each request j, let us define $p(j)$ to be the largest index $j < i$ such that $f(j) \leq s(i)$.

   Let us assume that there is an optimal solution O. The last interval n either belongs to O or doesn't. Let's explore both sides.

   If $n \in O$, no interval indexed between $p(n)$ and n can belong to O. Further O must include an optimal solution for the requests $1, \ldots, p(n)$.

   If $n \notin O$ then O is the optimal solution of requests $\{1, \ldots, n-1\}$.

   Subproblems: For $1 \leq j \leq n$, let O_j denote the optimal solution to the problem consisting of requests $\{1, \ldots, n\}$ and let $OPT(j)$ denote its value.

   Recurrence:

   $$OPT(j) = max(v_j + OPT(p(j)), OPT(j-1))$$

   Accordingly, we can find whether or not request j belongs to O_j.

2. Given a sequence of n real numbers $a_1, \ldots, a_n$, determine a contiguous subsequence $a_i, \ldots, s_j$ for which the sum of elements in the subsequence is maximized. E.g., for input $\{3, -4, 9, -8, 8, 7\}$, the output will be $\{9, -8, 8, 7\}$ with sum = 16. Propose a dynamic programming algorithm for this problem and show the working of your algorithm using the example provided.

   $\Rightarrow$ The idea is to maintain two values at every point (index i). The global maximum sum that we have seen (MX), and the maximum sum of contiguous elements including the element i (max_i).

   Subproblems: (max_i) for $i = 1$ to n, MX

   Recurrence:

   Suppose you are at the $i+1$th element in the array.

   $max_{i+1} = max(a_i, a_i + max_i)$

   $MX = max(max_{i+1}, MX)$

   Final answer is MX and you may make small changes to be able to come up with the actual contiguous sequence.

3. You are traveling by a canoe down a river and there are n trading posts along the way. Before starting your journey, you are given for each $1 \leq i < j \leq n$, the fee $f_{i,j}$ for renting a canoe from post i to post j. These fees are arbitrary. For example it is possible that $f_{1,3} = 10$ and $f_{1,4} = 5$. You begin at trading

post 1 and must end at trading post n (using rented canoes). Your goal is to minimize the rental cost. Give the most efficient algorithm you can for this problem.

$\Rightarrow$ Let $m[i]$ be the rental cost for the best solution to go from post i to post n for $1 \leq i < j \leq n$. The final answer is in $m[1]$, and $m[n] = 0$.

Recurrence: $m[i] = min_{j:i<j\leq n}(m[j] + f_{i,j})$

The subproblems can be solved in the order $m[n], m[n-1], \ldots, m[1]$. The time complexity if $O(n^2)$.

4. You are given k coin values $c_0 < c_1 < \ldots < c_{k-1}$ (where $c_0 = 1$), and a value v, find a way to give the value v in change using as few coins as possible.

   $\Rightarrow$

   Subproblems: Let $opt(n)$ be the minimum number of coins required to make a change of value n, $n = 1, \ldots, v$

   Recurrence: $opt(n) = 1 + min_{0\leq i\leq k-1:c_i<n} opt(n - c_i)$

5. Design a dynamic programming algorithm for the following problem: Given a sequence $a_1 \circ_1 a_2 \circ_2 \ldots a_{n-1} \circ_{n-1} a_n$, in which each a_i is a postive integer and each $\circ_i$ is '+' or '-', compute a parenthesization of the expression such that the resulting value is the maximum possible. It suffices to compute the resulting value instead of the parenthesization. Estimate the time complexity of the algorithm.

   For example, if the given sequence is 3-4-5, ((3-4)-5) results in -6 while (3-(4-5)) results in 4. The second parenthesization results in the maximum possible value, and the output is 4. Show the working of your algorithm on this example.

   $\Rightarrow$ We need to record both the maximum and minimum value of each subproblem to take care of both the signs.

   Subproblems: Let $M[i, j]$ denote the max value of the expression $a_i \circ_i \ldots x_j$ and $m[i, j]$ denote its minimum value.

   Recurrence: Consider $M[i, j]$. You will first parenthesize at $[i, k]$ and $[k+1, j]$ for some $i \leq k \leq j-1$. If $\circ_k =$ '$+$', $M[i, j]$ can be written as:

   $M[i, j] = M[i, k] + M[k + 1, j]$

   On the other hand, if $\circ_k =$ '$-$', it can be written as:

   $M[i, j] = M[i, k] - m[k + 1, j]$

   Since $M[i, j]$ denotes the max value,

   $M[i, j] = max_{i\leq k\leq j-1} M[i, k] + M[k + 1, j]$, if $\circ_k =$ '$+$', $M[i, k] - m[k + 1, j]$ if $\circ_k =$ '$-$'

   Analogously, you can define recurrence for $m[i, j]$.

   The complexity will be $O(n^3)$. The table of subproblems should be filled in a very similar way as we did for matrix chain multiplication problem.

6. A sequence is called a good sequence if $a_1 < a_2 > a_3 < a_4 \ldots, a_k$, i.e., $a_i < a_{i+1}$ if i is odd and $a_i > a_{i+1}$ if i is even for all $i < k$. You are given a sequence A containing n integers. You need to find the length of longest good subsequence of A using dynamic programming algorithm. For example, for the input sequence $\{1, 2, 6, 5, 3, 4\}$, the largest such sequence is $\{2, 6, 3, 4\}$. Show the working of your algorithm using this example.

   $\Rightarrow$

   Note that similar to the last problem, you will need to maintain the length of longest sequences ending either with $<$ or $>$.

   $maxL_i$: Length of the longest good sequence of the form $a_1 < a_2 > a_3 < ... < a_i$ ending at i

   $maxR_i$: Length of the longest good sequence of the form $a_1 < a_2 > a_3 < ... > a_i$ ending at i

Recurrence:

$maxL_i = max(1, max\{maxR_j + 1 | j < i, A_j < A_i\})$

$maxR_i = max(1, max\{maxL_j + 1 | j < i, A_j > A_i\})$

Final answer: max of all $maxL_i$ and $maxR_i$

7. You are given a collection C of n positive integers $a_0, a_1, \ldots, a_n$ with sum $S = \sum_0^{n-1} a_i$. Your task is to partition C into two subcollections A and B ($C = A \cup B$) such that the sum of the members of A is as close as possible to the sum of the members of B. In other words, if $S_1 = \sum_{a \in A} a$ and $S_2 = \sum_{b \in B} b$, your objective is to minimize $|S_1 - S_2|$. Design an efficient dynamic programming algorithm to solve this problem. What is the complexity of your algorithm? Take an example set $C = \{1, 2, 4\}$ and show the working of your algorithm.

   $\Rightarrow$

   We first solve the *subset-sum* problem, that is, given the collection C, is there a subset that sums to s

   Subproblems: $C[i, s]$ for $i = 0, \ldots, n$ and $s = 0, \ldots, S = 0$ if there is subset from the elements $1, \ldots, i$ that adds to s?

   Why should this subproblem help? An ideal solution would correspond to two partitions, each subset with a sum of $S/2$. So, we can find if $C[n, S/2]$ has a solution. If so, there is a subset from $1, \ldots, n$ that sums to $S/2$, and since all the elements add up to S, there would be the complementary set that adds up to $S/2$ and $|S_1 - S_2| = 0$. However, if $C[n, S/2]$ does not have a solution, we can search for $C[n, S/2 - 1]$ and so on.

   Recurrence: Very similar to the 0-1 knapsack problem, except that the value of an item v_i is same as the weight w_i, and instead of filling the knapsack to the best, you just need to store 0/1, corresponding to the fact whether there is a subset that sums to exactly the sum.