CS 218 : Design and Analysis of Algorithms

Lecture 11: Dynamic Programming: Order and Method. Weighted Interval Scheduling.

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering   Indian Institute of Technology, Bombay

1 Weighted Interval Scheduling

Consider the following problem. **Input:** A set S of n intervals given by their left and right end-points and a positive integral weight for each interval.
Output: A subset T of S of maximum weight such that no two intervals in T overlap.

A typical problem that we will encounter many times is to find a subset of a set of maximum or minimum weight so that some constraints are satisfied.

For such problems we will describe a procedure that only returns the weight of the optial solution. The actual solution can be returned using some additional bookkeeping. We omit this part to keep the description simple. We will follow this approach through the course.

Step 1: Find a recursive solution.

Either a subset contains the first element or it does not. We will consider both cases separately and put them together. Suppose the original set of intervals is S. Let $MWNOSI(S)$ denote a procedure which returns a maximum weight non overlapping subset of intervals. Let I be an interval in S. Either the optimum contains I or it does not. Based on this the procedure then returns the maximum of the following two calls.

$$MWNOSI(S \setminus \{I\})$$

$$MWNOSI(S') + W_I$$

Here S' is the set of all intervals that do not overlap with I. Note that correctness can be easily proved by induction. Note also the brute force nature of the algorithm we have just designed. We do not know whether the interval I is chosen or not; we try both cases and choose the best. So, correctness is obvious.

There are more than one recursive calls (here two). Also, the input to the two calls have overlap. If we unroll the recursion, we see that we may repeat calls.

Step 2 is to determine what the distinct calls to MWNOSI are. By distinct we mean distinct values of the input parameters. Here the best we can say is that any subset of the intervals is possible depending on the input and the way we choose I at each stage. So the best we can say about the number of recursive calls is 2^n. This running time is not acceptable.

Step 3. If the number of distinct calls is large, try ordering the input. We begin by taking any order, in other words considering the input as $I_1, \ldots, I_n$, say the order in which the intervals are presented.

So the procedure we have to design is

$$MWNOSI(I_1, \ldots, I_n)$$

In the recursive step we return the maximum of the following two calls:

$$MWNOSI(I_2, \ldots, I_n)$$

$$MWNOSI(\text{intervals in the above order removing those that overlap with} I_1)$$

We notice that if we only use the first recursive step repeatedly then we are now left with only suffixes. This is good news. However the second recursive call has no such bound.

The thing to notice about the second call is that the input splits into two: those intervals that end before l_1 (the left end point of I_1) and those that begin after r_1 (the right end point of I_1). And these two can be solved separately and the results can be added. So our second tip is this: **Split** the input if needed.

Here is the updated recurrence.

$$MWNOSI(I_1, \ldots, I_n)$$

In the recursive step we return the maximum of:

$$MWNOSI(I_2, \ldots, I_n)$$

$$MWNOSI(\text{intervals in} I_2, \ldots, I_n \text{ending before } l_1) + MWNOSI(\text{intervals in} I_2, \ldots, I_n \text{starting after} r_1) + W_1.$$

We have souped up the induction. To describe a call we need to add more parameters. Here is the third tip. **Add extra parameters.** We need to describe a generic call to the subroutine. If we open this once more we see that this can be specified by:

$$MWNOSI(I_1, \ldots, I_n; s; t)$$

This procedure returns the maximum weight among all subsets of non-overlapping intervals containing intervals from the first list which begin after s and end before t. Here is the recurrence:

If I_1 begins before s or ends after t then return

$$MWNOSI(I_2, \ldots, I_n; s; t).$$

Otherwise return the maximum of the following two quantities:

$$MWNOSI(I_2, \ldots, I_n; s; t)$$

$$MWNOSI(I_2, \ldots, I_n; s; l_1) + MWNOSI(I_2, \ldots, I_n; r_1; t) + W_1.$$

What are the number of distinct calls? Notice that the first parameter is a suffix, and there are $2n$ possiblities for the second and third parameters (they have to be either a left end-point or a right end-point of an interval) yielding a bound of n^3. We emphasize that if we had not split the second call into two disjoint parts, and worked on just the intervals which did not intersect with I_i then we would not have a polynomial number of distinct calls. So the trick here is that if the input splits into disjoint pieces such that the solution to one does not affect the solution to others, and the solutions to the pieces can be put together to get the final solution then use different recursive calls for each piece.

2 Adding to the Key Steps from last time.

1. Give the procedure a name. Clearly describe the input and output.
2. Write a recursive procedure.
3. Determine and describe the number of distinct calls.
4. If the number of distinct calls is large, try ordering the input. Find an order such that the number of distinct calls is as small as possible. Split the input into distinct calls during the recursion if possible. Add extra parameters is necessary.
5. Allocate a table, with one entry per distinct call.