How To Properly Answer Questions in CIS502

Mingyang Li

April 29, 2018

Contents

Part I
Dynamic Programming

1 Example Problem: Booking Flights

With the seemingly arbitrary ways in which airlines set their fares, you need dynamic programming to find the cheapest way to get from one city to another, especially when you care only about the cost and not about the number of stops you make. For this problem assume that we have n cities, 1, 2, . . . n arranged on a line. You are given the fare F(i,j) for going from city i to city j. Every flight you take must go from a lower-numbered city to a higher-numbered city. Design a dynamic programming algorithm to find the cheapest sequence of flights that will get you from city 1 to city n. Make your algorithm as efficient as possible and analyze its running time.

1.1 Top-Level Question

Should I take the flight from $n-1$ to n? In other words: From which city should I settle for city n?

1.2 Base Case

Airline Fare is zero for going to the first city.

1.3 Recurrence

Let $L(k)$ be the lowest rate to reach city $\#k$:

$$L(k) = \begin{cases} \min_{i=1}^{k-1} \{L(i) + F(i,k)\} & \text{if k>1} \\ 0 & \text{if } k = 1 \end{cases}.$$

1.4 Final Answer

$L(n)$.

1.5 Running Time

Let's look at how the algorithm will visit nodes at depth 1 (Lv.1 in the figure).

1. When visiting $L(1)$: 0 node in Lv.2 needs to be visited.
2. When visiting $L(2)$: 1 node in Lv.2 needs to be visited.
3. When visiting $L(3)$: 2 nodes in Lv.2 need to be visited.
   ...

4. When visiting $L(n-1)$: $n-2$ nodes in Lv.2 need to be visited.

Sum of the nodes in Lv.2 that need to be visited is

$$S = \frac{1}{2}\left[0 + (n-2)\right] \cdot (n-1) = \frac{1}{2}(n-2) \cdot (n-1).$$

Also consider:

1. Nodes in Lv.1 themselves need another $n-1$ visits.
2. Nodes in Lv.0, i.e. the root itself, needs another visit.

In total, that is $\frac{1}{2}(n-2) \cdot (n-1) + (n-1) + 1 \implies O(n^2)$ time.

1.6 Prove Correctness

1.6.1 Prove "optimal substrcture" property

1. **A solution to this problem involves making a choice.** Here, it's "to reach city k, from which city should I start my flight?"
2. **One of these choices must lead to the optimal solution.** This is an assumption.
3. **Let's define the subproblem after taking this optimal choice.** If we decide to settle for k from city m, then the sub-problem is just "to reach city m, from which city should I start my flight?"
4. **Prove that the sub-solutions in the optimal solution must themselves be optimal:**
   (a) **Declare the optimal solution to a problem:** Suppose the optimal solution to go from city 1 to city k says "we should leave from city w": $(1 \rightsquigarrow w \to k)$.
   (b) **Assume there exists a better solution solution:** Suppose also a better solution says "we should leave from city p" which offers lower fare: $(1 \rightsquigarrow p \to k)$.
   (c) **Using cut-and-paste technique, disprove the optimality of the original solution, creating a contradiction:** Then we can just go via p instead of w, contradicting to the assumption of "leaving from city w" is the optimal solution.
   (d) **Prove the "optimal substrcture" property:** Therefore, in order for an optimal solution to be optimal, every of its subproblems (in this case, only one) should be optimal.

1.6.2 Prove "overlapping subproblems" property

Suppose there are flights between every city in the map A->B->C->D. When querying L(D), we need query L(B) and L(C) respectively. When computing for the latter ($L(C)$), we need to solve for $L(B)$, for which we have already calculated when dealing with $L(D)$. That is to say, we have overlapping subproblems.

Part II
Greedy Algorithms

2 Example Problem: Activity Selection

S: list of activities.
f: list of finishing times.
s: list of starting times.
n: number of activities.

2.1 Subproblems Exist In This Problem – Answers to this problem is a series of choices

We make a choice, and are left with a subproblem to solve.

2.1.1 Description In This Case

We can rephrase our problem as: "given a set of activities, which should we choose, leaving the rest activities to be dealt with later as a subproblem?"
Answer to this problem is: "take the one finishes the first."

2.1.2 (No Proof Required)

If the question only requires an algorithm to be designed, this is it. Done. Don't bother to continue.

2.2 Greedy Choice Property

Greedy Choice Property: We can assemble a globally optimal solution by making locally optimal (greedy) solutions. Point it that, we don't have to make use of information outside, nor a layer inside, the scope of the current problem.

2.2.1 Description In This Case

Consider a non-empty problem S_k. Let a_1 be the activity in S_k w/ earliest finsihing time (i.e. our greedy choice). Let's show that $a_1$is indeed a member

of the optimal solution (i.e. a maximum-size subset of muutually comatible activities) to S_k.

2.2.2 Proof by Exchange

Suppose our greedy choice, the activity finishes first (at f_1), is not in the optimal solution A_k, and that the first activity our optimal solution had chosen finishes at f_2.

Since, by definition, our greedy choice finishes first, $f_1 \leq f_2$. Then, $A'_k = (A_k - \{a_2\}) \cup \{a_1\}$must also be feasible, because a_1 must be compatible just like a_2 is. Therefore, we can always find a optimal solution that includes a_1.

2.3 Optimal Substructure Property

> Optimal Substructure Property: An optimal solution to the problem contains within it optimal solutions to its subproblems.

2.3.1 Description In This Case

The optimal solution (i.e. the size of the maximum-size subset of mutually compatible activities) to the problem S should contain the greedy choice in every (in the case, only 1) subproblem S'.

2.3.2 Proof

1. (Trival steps omitted.)
2. **Prove that the sub-solutions in the optimal solution must themselves be optimal:**
   (a) **Declare the optimal solution to a problem:** Suppose the optimal solution A to taking the best activity in the subset of S is to take a_1. Then, according to A, the optimal solution to the subproblem $S' = \{i \in S : s_i \geq f_1\}$.
   (b) **Assume there exists a better solution solution:** Suppose a better solution to A' is actually B'. i.e., size(B') > size(A').
   (c) **Using cut-and-paste technique, disprove the optimality of the original solution, creating a contradiction:** Then, by prepending a_1 to B', we can create a solution to S, called B. Since $|B'| > |A'|$, $|B| = |B'| + 1$ and $|A| = |A'| + 1$, then $|B| > |A|$, contradicting to the assumption that A should be the optimal solution (i.e. the max-size solution) to S.
   (d) **Prove the "optimal substrcture" property:** Therefore, the greedy solution must contain the optimal solution of every subproblem.