CSE 5311 Homework 3 Solution

Problem 15.1-1

Show that equation (15.4) follows from equation (15.3) and the initial condition $T(0) = 1$.

Answer

We can verify that $T(n) = 2^n$ is a solution to the given recurrence by the substitution method. We note that for $n = 0$, the formula is true since $2^0 = 1$. For $n > 0$, substituting into the recurrence and using the formula for summing a geometric series yields

$$\begin{aligned} T(n) &= 1 + \sum_{j=0}^{n-1} 2^j \\ &= 1 + (2^n - 1) \\ &= 2^n \end{aligned}$$

Problem 15.1-2

Show, by means of a counterexample, that the following "greedy" strategy does not always determine an optimal way to cut rods. Define the **density** of a rod of length i to be p_i, that is, its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where $1 \leq i \leq n$, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length $n - i$.

Answer

Here is a counterexample for the "greedy" strategy:

length i	1	2	3	4
price p_i	1	20	33	36
p_i/i	1	10	11	9

Let the given rod length be 4. According to a greedy strategy, we first cut out a rod of length 3 for a price of 33, which leaves us with a rod of length 1 of price 1. The total price for the rod is 34. The optimal way is to cut it into two rods of length 2 each fetching us 40 dollars.

Problem 15.1-3

Consider a modification of the rod-cutting problem in which, in addition to a price p_i for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a dynamic-programming algorithm to solve this modified problem.

Answer

```
MODIFIED–CUT–ROD(p, n, c)
    let r[0..n] be a new array
    r[0] = 0
    for j = 1 to n
        q = p[j]
        for i= 1 to j−1
                q = max(q, p[i]+r[j−i]−c)
        r[j] = q
    return r[n]
```

The major modification required is in the body of the inner **for** loop, which now reads $q = \max(q, p[i] + r[j - i] - c)$. This change reflects the fixed cost of making the cut, which is deducted from the revenue. We also have to handle the case in which we make no cuts (when i equals j); the total revenue in this case is simply $p[j]$. Thus, we modify the inner **for** loop to run from i to $j - 1$ instead of to j. The assignment $q = p[j]$ takes care of the case of no cuts. If we did not make these modifications, then even in the case of no cuts, we would be deducting c from the total revenue.

Problem 15.2-4

Describe the subproblem graph for matrix-chain multiplication with an input chain of length n. How many vertices does it have? How many edges does it have, and which edges are they?

Answer

The vertices of the subproblem graph are the ordered pairs v_{ij}, where $i \le j$. If $i = j$, then there are no edges out of v_{ij}. If $i < j$, then for every k such that $i \le k \le j$, the subproblem graph contain edges (v_{ij}, v_{ik}) and $(v_{ij}, v_{i+1,j})$. These edges indicate that to solve the subproblem of optimal parenthesizing the product $A_i \cdots A_j$, we need to solve subproblems of optimally parenthesizing the products $A_i \cdots A_k$ and $A_{k+1} \cdots A_j$. The number of vertices is

$$\sum_{i=1}^{n} \sum_{j=i}^{n} 1 = \frac{n(n+1)}{2},$$

and the number of edges is

$$\sum_{i=1}^{n}\sum_{j=i}^{n}(j-i) = \sum_{i=1}^{n}\sum_{t=0}^{n-i} t \qquad \text{(substituting } t = j-i\text{)}$$
$$= \sum_{i=1}^{n}\frac{(n-i)(n-i+1)}{2}.$$

Substituting $r = n - i$ and reversing the order of summation, we obtain

$$\begin{aligned}
&\sum_{i=1}^{n}\frac{(n-i)(n-i+1)}{2} \\
&= \frac{1}{2}\sum_{r=0}^{n-1}(r^2+r) \\
&= \frac{1}{2}\left(\frac{(n-1)n(2n-1)}{6} + \frac{(n-1)n}{2}\right) \qquad \text{(by equations (A.3) and (A.1))} \\
&= \frac{(n-1)n(n+1)}{6}
\end{aligned}$$

Thus, the subproblem graph has $\Theta(n^2)$ vertices and $\Theta(n^3)$ edges.

Problem 15.2-5

Let $R(i,j)$ be the number of times that table entry $m[i,j]$ is referenced while computing other table entries in a call of MATRIX-CHAIN-ORDER. Show that the total number of references for the entire table is

$$\sum_{i=1}^{n}\sum_{j=1}^{n} R(i,j) = \frac{n^3-n}{3}$$

(Hint: You may find equation (A.3) useful.)

Answer

Each time the l-loop executes, the i-loop executes $n-l+1$ times. Each time the i-loop executes, the k-loop executes $j - i = l - 1$ times, each time referencing m twice. Thus the total number of times that an entry of m is referenced while

computing other entries is $\sum_{i=2}^{n}(n-l+1)(l-1)/2$. Thus,

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{j=i}^{n} R(i,j) &= \sum_{l=2}^{n}(n-l+1)(l-1)2 \\
&= 2\sum_{l=1}^{n-1}(n-l)l \\
&= 2\sum_{l=1}^{n-1} nl - 2\sum_{l=1}^{n-1} l^2 \\
&= 2\frac{n(n-1)n}{2} - 2\frac{(n-1)n(2n-1)}{6} \\
&= n^3 - n^2 - \frac{2n^3 - 3n^2 + n}{3} \\
&= \frac{n^3 - n}{3}
\end{aligned}$$

Problem 16.1-1

Give a dynamic-programming algorithm for the activity-selection problem, based on recurrence (16.2). Have your algorithm compute the sizes $c[i, j]$ as defined above and also produce the maximum-size subset of mutually compatible activities. Assume that the inputs have been sorted as in equation (16.1). Compare the running time of your solution to the running time of GREEDY-ACTIVITY-SELECTOR.

Answer

The tricky part is determining which activities are in the set S_{ij}. If activity k is in S_{ij} , then we must have $i < k < j$, which means that $j - i \geq 2$, but we must also have that $f_i \leq s_k$ and $f_k \leq s_j$. If we start k at $j - 1$ and decrement k, we can stop once k reaches i, but we can also stop once we find that f k.

We create two fictitious activities, a_0 with $f_0 = 0$ and a_{n+1} with $s_{n+1} = \infty$. We are interested in a maximum-size set $A_{0,n+1}$ of mutually compatible activities in $S_{0.n+1}$. We'll use tables $c[0..n+1, 0..n+1]$as in recurrence (16.2) (so that $c[i, j] = |A_{ij}|$, and $act[0..n+1, 0..n+1]$, where $act[i, j]$ is the activity k that we choose to put into A_{ij}.

We fill the tables in according to increasing difference $j-i$, which we denote by l in the pseudocode. Since $S_{ij} = \emptyset$ if $j-i < 2$, we initialize $c[i, j] = 0$ for all i and $c[i, i+1] = 0$ for $0 \leq i \leq n$. As in RECURSIVE-ACTIVITY -SELECTOR and GREEDY -ACTIVITY-SELECTOR , the start and finish times are given as arrays s and f , where we assume that the arrays already include the two fictitious activities and that the activities are sorted by monotonically increasing finish time.

```
DYNAMIC–ACTIVITY–SELECTOR(s, f, n)
        let c[0..n+1, 0..n+1] and act[0..n+1, 0..n+1] be new tables
    for i = 0 to n
        c[i, i] = 0
```

```
        c[i, i+1] = 0
    c[n+1, n+1] = 0
    for l = 2 to n+1
        for i = 0 to n-l+1
            j = i+l
            c[i, j] = 0
            k = j -1
            while f[i] < f[k]
                if f[i] <= s[k] and f[k] <= s[j]
                    and c[i, k] + c[k, j] + 1 > c[i, j] do
                    c[i, j] = c[i, k] + c[k, j] + 1
                    act[i, j] = k
                k = k -1
     print "A max size set of mutually compatible activities "
     print c[0, n+1]
     print "The set contains "
     PRINT-ACTIVITIES(c, act, 0, n+1)

PRINT-ACTIVITIES(c, act, i, j)
        if c[i, j] > 0
        k = act[i, j]
        print k
        PRINT-ACTIVITIES(c, act, i, k)
        PRINT-ACTIVITIES(c, act, k, j)
```

The PRINT-ACTIVITIES procedure recursively prints the set of activities placed into the optimal solution A_{ij} . It first prints the activity k that achieved the maximum value of $C[i, j]$, and then it recurses to print the activities in A_{ik} and A_{kj}. The recursion bottoms out when $c[i, j] = 0$, so that $A_{ij} = \emptyset$.

Whereas GREEDY-ACTIVITY-SELECTOR runs in $\Theta(n)$ time, the DYNAMIC-ACTIVITY -S ELECTOR procedure runs in $O(n^3)$ time.

Problem 16.1-2

Suppose that instead of always selecting the first activity to finish, we instead select the last activity to start that is compatible with all previously selected activities. Describe how this approach is a greedy algorithm, and prove that it yields an optimal solution.

Answer

The proposed approach – selecting the last activity to start that is compatible with all previously selected activities – is really the greedy algorithm but starting from the end rather than the beginning.

Another way to look at it is as follows. We are given a set $S = \{a_1, a_2, \cdots, a_n\}$ of activities, where $a_i = [s_i, f_i)$, and we propose to find an optimal solution by selecting the last activity to start that is compatible with all previously selected activities. Instead, let us create a set $S' = \{a'_1, a'_2, \cdots, a'_n\}$, where $a'_i = [f_i, s_i)$. That is, a'_i is a_i in reverse. Clearly, a subset of $\{a_1, a_2, \cdots, a_n\} \subset S$ is mutually compatible if and only if the corresponding subset $\{a'_1, a'_2, \cdots, a'_n\} \subset S'$ is

also mutually compatible. Thus, an optimal solution for S maps directly to an optimal solution for S' and vice versa.

The proposed approach of selecting the last activity to start that is compatible with all previously selected activities, when run on S, gives the same answer as the greedy algorithm from the text – selecting the first activity to finish that is compatible with all previously selected activities – when run on S'. The solution that the proposed approach finds for S corresponds to the solution that the text's greedy algorithm finds for S', and so it is optimal.

Problem 16.1-3

Not just any greedy approach to the activity-selection problem produces a maximum-size set of mutually compatible activities. Give an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities does not work. Do the same for the approaches of always selecting the compatible activity that overlaps the fewest other remaining activities and always selecting the compatible remaining activity with the earliest start time.

Answer

- For the approach of selecting the activity of least duration from those that are compatible with previously selected activities: This approach selects

i i	1	2	3
s_i	0	2	3
f_i	3	4	6
duration	3	2	3

  just a_2, but the optimal solution selects a_1, a_3.

- For the approach of always selecting the compatible activity that overlaps the fewest other remaining activities: This approach first selects a 6 , and

i i	1	2	3	4	5	6	7	8	9	10	11
s_i	0	1	1	1	2	3	4	5	5	5	6
f_i	2	3	3	3	4	5	6	7	7	7	8
# of overlapping activities	3	4	4	4	4	2	4	4	4	4	3

  after that choice it can select only two other activities (one of a_1, a_2, a_3, a_4 and one of a_8, a_9, a_10, a_11). An optimal solution is a_1, a_5, a_7, a_11.

- For the approach of always selecting the compatible remaining activity with the earliest start time, just add one more activity with the interval Œ0; 14/ to the example in Section 16.1. It will be the first activity selected, and no other activities are compatible with it.

Problem 16.2-4

Professor Gekko has always dreamed of inline skating across North Dakota. He plans to cross the state on highway U.S. 2, which runs from Grand Forks, on the eastern border with Minnesota, to Williston, near the western border with Montana. The professor can carry two liters of water, and he can skate m miles before running out of water. (Because North Dakota is relatively flat, the professor does not have to worry about drinking water at a greater rate on uphill sections than on flat or downhill sections.) The professor will start in Grand Forks with two full liters of water. His official North Dakota state map shows all the places along U.S. 2 at which he can refill his water and the distances between these locations.

The professor's goal is to minimize the number of water stops along his route across the state. Give an efficient method by which he can determine which water stops he should make. Prove that your strategy yields an optimal solution, and give its running time.

Answer

The optimal strategy is the obvious greedy one. Starting with both bottles full, Professor Gekko should go to the westernmost place that he can refill his bottles within m miles of Grand Forks. Fill up there. Then go to the westernmost refilling location he can get to within m miles of where he filled up, fill up there, and so on. Looked at another way, at each refilling location, Professor Gekko should check whether he can make it to the next refilling location without stopping at this one. If he can, skip this one. If he cannot, then fill up. Professor Gekko doesn't need to know how much water he has or how far the next refilling location is to implement this approach, since at each fillup, he can determine which is the next location at which he'll need to stop.

This problem has optimal substructure. Suppose there are n possible refilling locations. Consider an optimal solution with s refilling locations and whose first stop is at the kth location. Then the rest of the optimal solution must be an optimal solution to the subproblem of the remaining $n - k$ stations. Otherwise, if there were a better solution to the subproblem, i.e., one with fewer than $s - 1$ stops, we could use it to come up with a solution with fewer than s stops for the full problem, contradicting our supposition of optimality.

This problem also has the greedy-choice property. Suppose there are k refilling locations beyond the start that are within m miles of the start. The greedy solution chooses the kth location as its first stop. No station beyond the kth works as a first stop, since Professor Gekko would run out of water first. If a solution chooses a location $j < k$ as its first stop, then Professor Gekko could choose the kth location instead, having at least as much water when he leaves the kth location as if he'd chosen the jth location. Therefore, he would get at least as far without filling up again if he had chosen the kth location.

Problem 16.3-1

Explain why, in the proof of Lemma 16.2, if $x.freq = b.freq$, then we must have $a.freq = b.freq = x.freq = y.freq$.

Answer

We are given that $x.freq \leq y.freq$ are the two lowest frequencies in order, and that $a.freq \leq b.freq$. Now,

$$\begin{aligned} & b.freq = x.freq \\ \Rightarrow & a.freq \leq x.freq \\ \Rightarrow & a.freq = x.freq \qquad \text{(since } x.freq \text{ is the lowest frequency)}, \end{aligned}$$

and since $y.freq \leq b.freq$,

$$\begin{aligned} & b.freq = x.freq \\ \Rightarrow & y.freq \leq x.freq \\ \Rightarrow & y.freq = x.freq \qquad \text{(since } x.freq \text{ is the lowest frequency)}, \end{aligned}$$

Thus, if we assume that $x.freq = b.freq$, then we have that each of $a.freq$, $b.freq$, and $y.freq$ equals $x.freq$, and so $a.freq = b.freq = x.freq = y.freq$.