Project 2: Dynamic Programming

Author: Gabriel Hofer

Instructor: Dr. Rebenitsch
Course: CSC-372 Analysis of Algorithms
Date: October 8, 2020

Question 1 [40 points]

Location #1

Amount	Proposal	Revenue
0	0	**0**
1	0,1	**4**
2	0,1,2	~~4,~~ **5**
3	0,1,2	~~4,~~ **5**
4	0,1,2	~~4,~~ **5**
5	0,1,2	~~4,~~ **5**

Location #1 and #2

Amount	Proposal	Revenue
0	0	**0**
1	0,1	**4** ~~,2~~
2	0,1,2	~~5,~~ **6** ~~,4~~
3	0,1,2,3	~~5, 7,~~ **8** ~~, 8~~
4	0,1,2,3	~~5, 7, 9,~~ **12**
5	0,1,2,3	~~5, 7, 9,~~ **12**

Location #1 and #2 and #3

Amount	Proposal	Revenue
0	0	**0**
1	0,1	**4,** ~~3~~
2	0,1,2	~~6,~~ **7,** ~~4~~
3	0,1,2	~~8,~~ **9,** ~~8~~
4	0,1,2	~~12, 11,~~ **14**
5	0,1,2	~~12,~~ **15,** ~~12~~

For the following questions (b) and (c), I interpreted the questions to be asking for the maximum revenue from location i such that we can also spend money on locations j<i such that j>0.

Justification for this interpretation:

1. This is how the capital budgeting tables were calculated in the lecture video (Week 5 (Sep 21), Monday). After filling out the table for location 1, each subsequent location uses results from previous locations to calculate the revenue and optimal proposal for the current location.
2. The following questions would be trivial if we assumed that each location didn't use results from previous locations to calculate the optimal revenue for the current location.

b. [5 points] What is the best revenue and the proposals for each location if you have a starting amount of 5?

Amount: 5
Location #1: revenue: 5, proposal: 2 (loc 1)
Location #1, #2: revenue: 12, proposals: 1 (loc 1), 3 (loc 2)
Location #1, #2, #3: revenue: 15, proposals: 1 (loc 1), 3 (loc 2), 1 (loc 3)

c. [5 points] What is the best revenue and the proposals for each location if you have a starting amount of 4?

Amount: 4
Location #1: revenue: 5, proposal: 2 (loc 1)
Location #1, #2: revenue: 12, proposal: 1 (loc 1), 3 (loc 2)
Location #1, #2, #3: revenue: 14, proposal: 1 (loc 1), 3 (loc 2), 2 (loc 3)

Question 2 [34 points]

a. [4 points] How can you describe a particular location in the problem? (This helps you know how many parameters the T() will need)

There is one parameter ***i*** in our recursive formula. Using zero-based indexing, pointer ***i*** points to the ***i***th element in arr[ ]. In other words, ***i*** is an index of array arr. In context, ***i*** represents the second element of the current pair. We use an auxiliary pointer ***j*** to iterate *backwards* in array arr[ ] to find a valid "split" that will mark the beginning of the current pair. Pointer j is bounded according to the following inequalities: j<i and j>=0 and $arr[i] - arr[j] <= height$.

b. [4 points] What is the optimal substructure? I require the object being removed. (This helps you determine what will be removed inside the T())

Definition of "optimal substructure: optimal solutions to a problem incorporate optimal solutions to related subproblems (Cormen 2009)." The optimal substructure for this problem is that we will find the optimal solutions for *prefixes* of the original array arr[ ]. Each prefix of the array is broken down into smaller and smaller prefix-arrays until the base case is reached. The base case is when our prefix-array has one item which means that we can't make a pair. Below, $disjointPairs[i]$ represents the number of pairs that can be made with the first i items in array $arr[\,]$.

$$disjointPairs[i] = 0 \; if \; i <= 0$$

$$disjointPairs[i] = 0 \; if \; arr[i] - arr[j] > height$$

$$disjointPairs[i] = \sum_{j<i} (1 + disjointPairs[j-1]) \; if \; i > 0$$

c. [4 points] What are the overlapping sub problems? I will take "proof by example." (This helps you know how many T(…) you need)

Overlapping subproblems occur when we arrive at a problem state (stage/location) multiple times when solving a problem. The value of the parameters determine the state. For example, let's assign values to our parameter: i=4. If we arrive at this same state (i.e. if we see the same index i=4) again in the program, that would be considered an overlapping subproblem.

d. [4 points] What are you checking at each stage/step? (This fills in the T(...))

We check to see which case applies to the current state:

$$T(i) = 1 \ if \ i <= 0$$

$$T(i) = 1 \ if \ arr[i] - arr[j] > height$$

$$T(i) = \sum_{j<i} 1 + T(j-1) \ if \ i > 0$$

e. [4 points] What is the cost of dividing and combining the subproblem? I will take "proof by example." (This helps in finding +f(n) in the formula)

$$T(i) = \sum_{j<i} 1 + T(j-1) \ if \ i > 0$$

f. [4 points] What is the smallest legal problem and what is it worth? (This is the base case)

The smallest legal problem is T(0), where the number of elements in the "subarray" of the subproblem is 1. The number of pairs that can be made with one item is zero:

$$disjointPairs[i] = 0 \ if \ i <= 0$$

g. [10 points (2 pt per blank)] From the above, what is the mathematical optimization formula you are trying to solve here? You must write this in a recursive formula.

Honestly, I don't understand what this question is asking, so I'm going to copy & paste the recursive math/theory from above to try to get some points.

Complexity Analysis

$$T(i) = 1 \ if \ i = 0$$

$$T(i) = 1 \ if \ arr[i] - arr[j] > height$$

$$T(i) = \sum_{j<i \text{ and } arr[i]-arr[j]<=height} [1 + T(j-1)] \ if \ arr[i] - arr[j] <= height$$

Dynamic Programming Relations

$$disjointPairs[i] = 0 \ if \ i <= 0$$

$$disjointPairs[i] = 0 \ if \ arr[i] - arr[j] > height$$

$$disjointPairs[i] = \sum_{j<i} (1 + disjointPairs[j-1]) \ if \ i > 0$$

Question 3 [16 points]

a. [5pt] Determine which dynamic algorithm the problem is closest.

Algorithm-> Knapsack
Map: plant_width-> weight
plant_cost-> value

b. [5pt] This may not be an exact an exact match. What adjustments will be needed to make this work?

In the classical knapsack problem, the goal is to maximize total value of the items in the knapsack while the total weight of the items in the knapsack is less than or equal to some bound *W*. However, in this planting problem, we want to minimize the total value or "cost" of the items while the total weight or "lengths" of the plants is less than or equal to the length of the single row.

c. [6pt] Now determine what will be used as an input into the algorithm.

Input:

1. First line: L - an integer representing length of the row of plants
2. Second line: N - and integer representing the number of possible plants to choose from
3. The next n lines contain n pairs of integers (w, c) where w is the width of the plant and c is the cost of the plant.