Tristan Santiago

October 12, 2018

Homework 3

Problem 1. Canoe Rental Problem

There are n trading posts numbered 1 to n as you travel downstream. At any trading post i, you can rent a canoe to be returned at any of the downstream trading posts j, where j ≥ i. You are given an array R[i, j] defining the costs of a canoe which is picked up at post I and dropped off at post j, for 1 ≤ i ≤ j ≤ n. Assume that R[i, i] = 0 and that you can't take a canoe upriver. Your problem is to determine a sequence of rentals which start at post 1 and end at post n, and that has the minimum total cost.

With the assistance of a homework discussion group member, I was able to deduce that this problem is essentially the Rod-Cutting example from the textbook and Rod-Cutting algorithm presented in the Week 3 module. We have a defined array called R, which has points on its table[i, j], i for the rows and j for the columns. R represents the total costs of a canoe one it has been rented from post i, and dropped off at post j.

The following is an arbitrary example of the problem:

	Arrival Posts j				
	Cost (i, j)	1	2	3	4
Departure Posts i	1	0	2	3	7
	2	x	0	2	4
	3	x	x	0	2
	4	x	x	x	0

In the arbitrary example above (with the prices known), the numbers within the cells denotes cost, while x means that renting from these locations is not possible, as you cannot travel back upstream. (Therefore, j cannot be less than post i) The cheapest sequence above is to rent from post 1 to post 3 (which costs 3), then to change canoes from post 3 to post 4 (which costs 2), for a total of 5.

At post 1, the cost is 0 because if you start and leave at post 1, you haven't traveled at all, and thus haven't incurred any costs. If you travel from post 1 to post 2 (1, 2) the cost would be 2. From post 1 to 3 (1, 3), the cost would be 3, and so on. We can continue solving costs all the way until we reach our final destination n.

So, in order to solve this problem, we create a new array to hold the optimal costs.

Post #	1	2	3	k	i	N
R[i, j]	0	[1,2]	[1,3]	[1,k]	[1,i]	[n]

A. **Describe verbally and give pseudo code for a DP algorithm called CanoeCost to compute the cost of the cheapest sequence of canoe rentals from trading post 1 to n. Give the recursive formula you use to fill in the table or array.**

Referenced from Introduction to Algorithms pg. 369

```
CanoeCost(array, n)
Let r[0…n] and s[1…n] be new arrays // One for all costs, another for optimal costs.
swapLocations[0…n ]          // Array to store where canoes are swapped.
R[0] = 0        // Cost set to zero
For j = 1 to n  // Loop through destinations (ending posts)
    q = -∞ // Unknown optimal until minimum price is discovered.
    For i = 1 to j   // Loop through starting posts.
        If q < p[i] + r[j – i]    // If current min is less than current cost
            q = p[i] + r[j – 1]      // Set new min
            swapLocations = p[i] + r[j-1]        // Also store in array
            s[j] = I// s[j] holds the optimal size of i of the first stop when
                solving a sub-problem of size j
        r[j] = q           // Set optimal cost
return r and s
```

B. **Using your results from part a, how can you determine the sequence of trading posts from which canoes were rented? Give a verbal description and pseudo code for an algorithm called PrintSequence to retrieve the sequence.**

The following procedure takes a rental cost table R and a number of posts size n, and calls it first-post sizes and then prints out the complete list of post sizes in an optimal decomposition of number of posts size n:

```
PrintSequence
(r, s) = CanoeCost(array, n)
while n > 0
    print s[n]
    n = n – s[n]
```

To print the sequence of trading posts, used the same procedure on the array we stored the indexes of the locations where the canoes were changed.

```
PrintSequence
(swapLocations, s) = CanoeCost(array, n)
while n > 0
    print s[n]
    n = n – s[n]
```

This should print the indexes where canoes were changed, as the rest of the array should be unchanged from the initial values at run time.

C. **What is the running time of your algorithms?**

I think the running time for CanoeCost depends on the sub-problems of size n. Whenever a canoe reaches a new trading post, the algorithm has to determine whether or not an exchange should occur. Therefore, every trading post must be examined. This is accomplished using two nested for loops (each O(n)), which equates to a running time of $O(n^2)$ when multiplied together.

The PrintSequence function only needs to iterate through the values stored in the array that's passed to it. Therefore, there will be n items which results in an O(n) running time, in the worst case.

Problem 2

Acme Super Store is having a contest to give away shopping sprees to lucky families. If a family wins a shopping spree each person in the family can take any items in the store that he or she can carry out, however each person can only take one of each type of item. For example, one family member can take one television, one watch and one toaster, while another family member can take one television, one camera and one pair of shoes. Each item has a price (in dollars) and a weight (in pounds) and each person in the family has a limit in the total weight they can carry. Two people cannot work together to carry an item. Your job is to help the families select items for each person to carry to maximize the total price of all items the family takes. Write an algorithm to determine the maximum total price of items for each family and the items that each family member should select.

A. **A verbal description and give pseudo-code for your algorithm. Try to create an algorithm that is efficient in both time and storage requirements.**
This problem resembles the "0-1 knapsack problem" from the lecture video this week, because each item must be taken or not taken. Given a set of items, in this case, every item in the store, each with a weight and a benefit (in the form of monetary value). Each person can only hold a fixed amount of weight.

Let S be the set of items represented by the ordered pairs (w_i, b_i) and W be the capacity each person can hold. Find a $T \subseteq S$ such that:

$$max \sum_{i \in T} b_i \; subject \; to \sum_{i \in T} w_i \leq W$$

The subproblem will be to compute B[k,w] which is the maximum benefit for total capacity w and items {1, 2, … k}.

The recursive formula would be:

$$B[k,w] \begin{cases} B[k-1,w] if \; w_k > w \\ \max\{B[b-1,w], B[k-1,w-w_k]+b_k\} \; w_k \leq w \end{cases}$$

The best subset of S_k that has the total weight w, either contains item k or not.

Pseudo code:

```
for w = 0 to W
        B[0,w] = 0      // 0 items
```

```
for i = 0 to n
        B[i,0] = 0          // 0 weight
        for w = 1 to W
                if $w_i \le w$          // Item i can be part of the solution
                        if $b_i$ + B[i-1,w-$w_i$] > B[i-1, w]
                                B[i,w] = $b_i$ + B[i-1,w-$w_i$]
                        Else
                                B[I,w] = B[i-1, w]
        Else B[i,w] = B[i-1,w]          // $w_i$ > w item it is too big
```

B. What is the theoretical running time of your algorithm for one test case given N items, a family size of F, and family members who can carry at most M_i pounds for $1 \le i \le F$.

Let n = 5 (number of items)
W =7 (max weight)
For this example, the quantity of each item is 1.
Elements (weight, benefit):

			Sum of weights							
Item	Value	Weight	0	1	2	3	4	5	6	7
A	1	1	0	1	1	1	1	1	1	1
E	4	2	0	1	4	5	5	5	5	5
B	4	3	0	1	4	5	5	8	9	9
C	5	4	0	1	4	5	5	8	9	10
D	7	5	0	1	4	5	5	8	9	11

*Numbers in the cell represent the value of the item.
*If the item weight > sum of weights, then copy the value the cell above.
*Value = max (value by excluding the new weight, value by including the new weight)

```
for w = 0 to W          // O(W)
        B[0,w] = 0
for i = 0 to n          // Repeat n times
        B[i,0] = 0
        for w = 1 top W          // O(W)
                <the rest of the code>
```

Therefore, the running time of this algorithm is O(nW) pseudo-polynomial, which I believe matches the theoretical running time of the algorithm.

C. Implement your algorithm by writing a program named "shopping" that compiles/runs on the OSU engineering servers and follows the specifications.
File uploaded to TEACH.