Arthur Liou
CS325

HW 1

Problem 1. Canoe Rental Problem: (10 points): 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 $j \geq i$. 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 $1 \leq i \leq j \leq n$. 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.

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 used to fill in the table or array.

After looking at this problem, I realized this is very similar to the "Given x coins, find the least number of coins that would make Y cents". We can utilize an "answer" array that contains the optimal sequence of canoe rentals, which starts at post 1 and ends at post I, where $1 \leq i \leq n$. Also we don't care about j, since in this answer array, j is just the next i. Once we get to the "last" i, then we have the optimal answer array.

```
//define global variable
CanoeOptimalSequence = [];
//define function. R = array; C = current cost
CanoeCost(R, C)
//if array is len(1), then return that + cost
        Return R[0] + C;
//else array is longer than length 2
//find the next optimal path
        For k > 2, until i – 1
                If C + R[k, i] < R[1, i] (if new cost is less than current minimal cost)
//add the cost of R[i] to C
                        C = C + R[k, i]
//recursion on Canoe Cost, passing in a spliced array, and C
        CanoeCost(R[i:n], C)
```

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.

Instead of using CanoeCost specifically, I would use a ConoeSequence that would adapt the function from part a to log the Sequence (as S) instead of C. and return S at the base case function, instead of tracking the final cost.

```
//define function. P = optimal sequence array, I =
PrintSequence(P, i)
// if list is greater than 1 / exists
```

1. if $i > 1$ {
//Recursively call on this index i
2. PrintSequence(P, $P[i]$)
//print message for what to do
3. print "Rent a canoe at post " + $P[i]$ + " and drop it off at post " + i }

c) What is the running time of your algorithms?
At worst case, CanoeCost() runs in O(n^2). This is because for each post, there are n subproblems that need to be solved, each of which takes O(n) time. Thus O(n^2).
At worst case, PrintSequence() runs in O(n). This is because the depth of recursion is the number of canoes that are rented in the optimal sequence from post 1 to post n.

Problem 2: Shopping Spree: (20 points) 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.

Notes & Thoughts: 0-1 Knapsack Problem, treat each person individually since what one person carries cannot affect what others carry. The max "carry" for each person will total up to the max total price for the family. Subproblems are for each person.

Input: Two: one array (each element is a family member and contains the max weight one can carry), and another array of items (each element is an array of two – a[0] is price, a[1] is weight
Output: Array of arrays (Each element represents a family member & each element will contain an array of items). Will term this results array
Edge Case: Not considered for now.
Constraints: See below

//define algo (take two inputs)
//create total price variable
//create empty results array
//iterate through family member array and apply helper function to each
//append optimal price to total price
//append result to results array
//optional: grab the total price for the family
//return total price & results array

//helper function (Knapsack)
Input: weights and values of n items; person's max capacity, W
//create a temporary array of arrays / table
// Build up the array of arrays / table
// base case: if index is = or capacity is 0, return 0
// elif – weight is less than capacity, find and insert the max of the value + or weight
// else add current cost/price to current spot for optimal/max price
// return final cost/price at the [number of items][capacity]

b) What is the theoretical running time of your algorithm for one test case given N items, a family of size F, and family members who can carry at most M_i pounds for $1 \leq i \leq F$.
The 0-1 knapsack problem itself is of time complexity of O(nM), adapting it to a multiple, it would be more of O(FnM), where F is constant and M would be the highest capacity of any family member. F would be considered constant as increasing or decreasing F would not impact the time complexity; calculations, yes, but not the complexity. This last sentence is part of the rationale for dropping constants when looking at time complexity as when analyzing time complexity, constants are difficult and irrelevant to calculate.
Thus, removing F and leaving with O(nM)

Time Complexity: O(nM)

c) Implement your algorithm by writing a program named "shopping" that compiles/runs on the OSU engineering servers and follows the specifications below.

Input: The input file named "shopping.txt" consists of T test cases
- T ($1 \leq T \leq 100$) is given on the first line of the input file.
- Each test case begins with a line containing a single integer number N that indicates the number of items ($1 \leq N \leq 100$) in that test case
- Followed by N lines, each containing two integers: P and W. The first integer ($1 \leq P \leq 5000$) corresponds to the price of object and the second integer ($1 \leq W \leq 100$) corresponds to the weight of object.
- The next line contains one integer ($1 \leq F \leq 100$) which is the number of people in that family.

- The next F lines contains the maximum weight ($1 \leq M \leq 200$) that can be carried by the i th person in the family ($1 \leq i \leq F$).

Output: Written to a file named "shopping.out". For each test case your program should output the maximum total price of all goods that the family can carry out during their shopping spree and for each the family member, numbered $1 \leq i \leq F$, list the item numbers $1 \leq N \leq 100$ that they should select.