Name
Email

CS 325 - Homework Assignment 3

Problem 1: (3 points) **Rod Cutting**: (from the text CLRS) 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 / 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*.

Length i	1	2	3	4
Price Pi	1	20	33	32
Density(Pi/i)	1	10	11	8

For the given table, if we use the "greedy" strategy to find the optimal way, then for length 4, the biggest density is length3, so the rod will be cut like length 3 of price 33, length 1 of price 1, so the total benefit would be 33+1 = 34.
However, the true optimal solution will be cutting of the rod with two rods of length 2 of price 20, then the total would be 20*2 = 40.
Which is a counterexample.

Problem 2: *(3 points)* **Modified Rod Cutting**: (from the text CLRS) 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.

```
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 ]              # no cut situation when j = n, revenue = price of [j]
        for i = 1 to j-1        # make sure this is a cut at least, so j -1.
        q = max( q , p [ i ]+ r[ j−i ] − c )  #only need to add a c here with each cut loop
        r[ j ] = q
return   r [ n ]
```

Problem 3: *(6 points)* **Making Change**: Given coins of denominations (value) $1 = v_1 < v_2 < \dots < v_n$, we wish to make change for an amount A using as few coins as possible. Assume that v_i's and A are integers. Since $v_1 = 1$ there will always be a solution. Formally, an algorithm for this problem should take as input an array V where V[i] is the value of the coin of the ith denomination and a value A which is the amount of change we are asked to make. The algorithm should return an array C where C[i] is the number of coins of value V[i] to return as change and m the minimum number of coins it took. You must return exact change so

$$\sum_{i=1}^{n} V[i] \cdot C[i] = A$$

The objective is to minimize the number of coins returned or:

$$m = \min \sum_{i=1}^{n} C[i]$$

a) Describe and give pseudocode for a dynamic programming algorithm to find the minimum number of coins needed to make change for A.

For this problem, it’s similar with backpack problem, $f(x)=mini=0nf(x-coins[i])+1$, the total amount A is the f(x) here, to find the minimum number of coins. we need to go through each element in the coin array, and each element that can make up the total of A amount. The loop should only continue when the V[i] is still smaller or equal to the A amount, then the minimum number of coins would be 1+ min(A-V[i]) , then continue to loop the min [A- V[i]] until the remainder is 0 , otherwise, return -1, because the different combo of coins should equal exact the A amount. The pseudocode is listing below:

```
def coinChange ( V[i], A)
    dp = int [amount +1]
    dp[0] = 1
    for i in range (0 ,V[i]):     #go through each element in the coins
        for j in range (0,A):    #go through each element in amount,
            if ( j > = V[i]): # only compare when the A of j is bigger than V[i]
                dp[j] + = dp[j-V[i]]  # recursively find the best solution.

return dp [amount]
```

b) What is the theoretical running time of your algorithm?
The time complexity should be O(A*V), because Usually the complexity of a Dynamic Program algorithm is: # of sub-problems × choices for each sub-problem, for here, subproblems is equal to A is the amount, choices would be V[i], different kinds of coins.

Problem 4: **Shopping Spree:** *(18 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.

Submit to Canvas

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 is an exact knapsack problem, the algorithm should read number of test cases (T) from input file, should run a loop T times:

In each iteration read number of items (N),

Run a loop N times and save the corresponding prices and weights of N items into two Separate arrays.

Read number of family members from input file

For each family member, it should calculate the maximum price of items that can be carried by the family member with knapsack function.

Keep track the total of maximum prices

Return the maximum total price to the output file.

The pseudocode should be:

```
    Read T from input file
    for k in range (T):
        read N from input file
        for i in range (N):
                read P[i] from input file
                read W[i] from input file
        maxPrice = 0
        read F from input file
        for j in range (F):
                read M from input file
                create a table called B(benefits) pf size (N+1) (M+1)
                        for i  in range (N+1):
                                for j in range (M+1):
                                        if i == 0 or j == 0:
                                                B[i][j] = 0
                                        elif B[i-1] <= j:
                                                B[i][j] = max (P[i-1] + K [i-1] [j- W[i-1]], B[i-1][j] ]
                                        else
                                                B[i][j] = B[i-1] [j];
                        maxPrice = maxPrice + B [N] [M]
        write the maxPrice to the output file
```

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 $1 \leq i \leq F$.

Then the theoretical running time = $O(NM_1 + NM_2 + \ldots + NM_F) = O(N.\sum_{i=1}^{F} M_i)$

c) Implement your algorithm by writing a program named "**shopping**" (in C, C++ or Python) that compiles and runs on the OSU engineering servers. The program should satisfy the specifications below.

SEE Teach files

Input: The input file named "**shopping.txt**" consists of T test cases

- T (1 ≤ T ≤ 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 ≤ N ≤ 100) in that test case
- Followed by N lines, each containing two integers: P and W. The first integer (1 ≤ P ≤ 5000) corresponds to the price of object and the second integer (1 ≤ W ≤ 100) corresponds to the weight of object.
- The next line contains one integer (1 ≤ F ≤ 30) which is the number of people in that family.
- The next F lines contains the maximum weight (1 ≤ M ≤ 200) that can be carried by the ith person in the family (1 ≤ i ≤ F).

Output: Written to a file named "**results.txt**". 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 ≤ i ≤ F, list the item numbers 1 ≤ N ≤ 100 that they should select.

Sample Input

```
2
3
72 17
44 23
31 24
1
26
6
64 26
85 22
52 4
99 18
39 13
54 9
4
23
20
20
36
```

Sample Output:

```
Test Case 1
Total Price 72
Member Items
1: 1
3

Test Case 2
Total Price 568
Member Items
1: 3 4
2: 3 6
```

3: 3 6
4: 3 4 6

Submit to TEACH a zipped file containing your code files and README file.

Note: You will not be collecting experimental running rimes.