Trevor Stahl

1/24/19

HW 3

Problem 1

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.

This is a function that takes a two-dimensional array and a integer n. It then generates an array S and returns S[n]. S[k] is the total distance from post 1 to post k. S is generated by iteratively finding the min between the cost to get to a previous trading post and to get to the current trading post being calculated for. Thus S[n] is the cheapest way to get from trading post 1 to the final trading post.

```
CanoeCost(R[], n) {
        Let S be an array of N values
        Let S[0] = 0
        For  j >= 1,  j <= n
                S[j] = inf;
                for i = 0,  i < j
                        S[j] = min { S[i] +  R[i,j], S[j]}
        Return:  S[n]
}
```

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 difference between this function and the previous one is that this one additionally utilizes an array P which is an array of arrays. P[k] represents the sequence to get to the k trading post. Each P[k] is generated by allowing P[0] = [0] and P[k] = P[the ideal previous trading post] + this trading post. The function prints P[n] which is an array of the path of trading posts which is ideally used to get to the last trading post.

```
PrintSequence(R[], n) {
        Let S be an array of N values
        Let P be an array of N values
        Let S[0] = 0
        Let P[0] = [0]
        For  j >= 1,  j <= n
                S[j] = inf;
                 for i = 0,  i < j
                        Let dist = S[i] + R[I,j]
                        If dist < S[j]
                                S[j] = dist
                                P[j] = P[i] + [j]

        Print(P[n])

}
```

c) What is the running time of your algorithms?

CanoeCost

Theta(n^2)

Print Sequence

O(n^3)

Omega(n^2)

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.

For each person, this function uses recursion to calculate every possible combination of carrying and not carrying each item, stopping when they cannot carry more. The base case here is when there are no more items. Thus the function through comparisons returns at the end the ideal set of items for the person to carry.

```
Ideal_Items (weights, prices, starting_index, carry_weight) {
        If starting_index >= len(weights):
                Return (0, [])

        (price_without, items_without) = Ideal_Items(weights, prices, starting_index, carry_weight)
        If carry_weight < weights[starting_index]
                Return (price_without, items_without)
        (price_with, items_with) = Ideal_Items(weights, prices, starting_index + 1,  carry_weight-
weights[starting_index])
        Price_with += prices[starting_index]
        Items_with += [starting_index]

        If price_without > price_with
                Return price_without, items_without
        else
                return price_with, items_with
}


Shopping() {
        Convert shopping.txt into python lists and usable data

        Call ideal items for each person
        Add the families up
        Create results.txt file
        Write results to 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 Mi pounds for $1 \leq i \leq F$.

O(F(2^N))

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.

Indentation may not be correct below. Download and test code from .py file

```
def ideal_items (weights, prices, starting_index, carry_weight):
    if starting_index >= len(weights):
        return (0, [])

    (price_without, items_without) = ideal_items(weights, prices, starting_index + 1, carry_weight)
    if carry_weight < weights[starting_index]:
        return (price_without, items_without)
    (price_with, items_with) = ideal_items(weights, prices, starting_index + 1, carry_weight-
weights[starting_index])
    price_with += prices[starting_index]
    items_with += [starting_index]

    if price_without > price_with:
        return (price_without, items_without)
    else:
        return (price_with, items_with)



def main ():
```

```
with open("shopping.txt") as file, open("results.txt", "w") as output:
    T = int(file.readline())
    for test_index in range(T):
        N = int(file.readline())
        weights = []
        prices = []
        for item_index in range(N):
            numbers = file.readline().split(" ")
            weights.append(int(numbers[1]))
            prices.append(int(numbers[0]))
        family_size = int(file.readline())

        total_price = 0
        all_items = []
        for family_index in range(family_size):
            carry_weight = int(file.readline())
            (price, items) = ideal_items(weights, prices, 0, carry_weight)
            total_price  += price
            all_items.append(items)

        output.write("Test Case {}\n".format(test_index+1))
        output.write("Total Price {}\n".format(total_price))
        output.write("Member Items\n")

        for family_index in range(family_size):
            output.write("{}: {}\n".format(family_index+1, " ".join(str(x+1) for x in all_items[family_index])))
```

main()