Name : Maahir Kalkan (ert547)

CS5463 Foundations of Software

Homework 4

Due 11/17/20 before 11:59pm (Central Time)

1. Coding - Red-Black Tree Balancing (10 points)

Complete the provided Red-Black tree program by implementing the method "rebalanceTree" in "RedBlackTree.java".

2. Coding - Segment Tree (10 points)

Complete the code in "SegmentTree.java" and "SegmentNode.java".

3. Choose Function (4 points)

Given n and k with $n \geq k \geq 0$, we want to compute the choose function $\binom{n}{k}$ using the following recurrence:

Base Cases: $\binom{n}{0} = 1$ and $\binom{n}{n} = 1$, for $n \geq 0$
Recursive Case: $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$, for $n > k > 0$

(1) (1 point) Compute $\binom{5}{3}$ using repeated calls to the above recurrence.
(2) (2 points) Give pseudo-code for a **bottom-up** dynamic programming algorithm to compute $\binom{n}{k}$ using the above recurrence.
(3) (1 point) Show the dynamic programming table your algorithm creates for $\binom{5}{3}$.

	0	1	2	3
0	1			
1	1	1		
2	1	2	1	
3	1	3	1	1
4	1	4	2	1
5	1	5	2	1

4. Perfect Play (8 points)

In this problem we describe a simple turn-based game.

Let's suppose you're given an array, A, of n integers. Two players alternate removing 1, 2, or 3 numbers off of the end of the array. The game ends when the array is empty. A player's score is the sum of all of the numbers they have

chosen. Suppose both players play optimally (i.e., try to win by the most points possible and, if winning is not possible, lose by the fewest points possible).

Find the difference between the players scores at the end of the game (first player score - second player score).

Array position i:	1	2	3	4	5	6	7	8	9
Value at $A[i]$:	50	-60	5	30	25	-20	-100	-10	30

(1) (1 point) Example: For the array A given above, what would be the optimal scores of player 1 and player 2?
Hint: the difference between the scores should be 140
(2) Let "$S(k)$ = first player score − second player score" where $0 \leq k \leq n$ and we k values in the array. Give a recursive definition of S:
   (a) (1 point) Base case
   Hint: this will occur when $k = 0$ and thus there are no elements left in the array to consider.
   (b) (4 points) Recursive case
   Hint: compare the 3 cases of items player 1 can take from the array. Specifically, notice that this removes 1, 2, or 3 elements from the end of A. It is then player 2's turn to select how many elements to remove.
(3) (2 points) Give pseudo-code for an algorithm which uses memoization to compute $S(n)$ based on the above recurrence (assume you are passed an array A of the points).

5. Inventory Management (10 points)

Suppose you are playing a video game. In this game you can only carry a limited amount of items. Every item has a value and your goal is to maximize the total value of items you are carrying.

Specifically, you are choosing which of m items to carry where item number i has value $v[i]$ (for your recurrence/pseudo-code you can assume you are passed the item values in an array, v, of size m).

(1) Suppose you can only carry n items of the m available.
   (a) (1 point) Example: Let $n = 3$, $m = 5$. If the item values are $v = \{5, 30, 17, 32, 40\}$ which 3 should you choose to carry?
   (b) (1 point) Give a short description of a greedy algorithm which maximizes your total value for any given n,m, and v.
(2) Suppose, the game is updated so that every item, i, now has weight, $w[i]$, in kilograms (you can assume you are passed the item weights in an array, w, of size m). You can only carry n kilograms of weight.
   (a) (1 point) Example: Let $n = 15$, $m = 5$. If the item values are $v =$

$\{5, 30, 17, 32, 40\}$ and the item weights are $w = \{2, 4, 3, 6, 15\}$ which should you choose to carry? Can you just greedily select the most valuable items to carry until you run out of capacity?

(b) Let $V(n, m)$ denote your maximum total value for any given n, m, v, and w. Give a recursive definition of V:

(i) (1 point) Base case

Hint: your base case will occur when $m = 0$ since you have no items left to consider.

(ii) (4 points) Recursive case

Hint: compare the total value obtained from taking the m^{th} item to the total value obtained from not taking the m^{th} item.

(c) (2 points) Based on your recurrence, write the pseudo-code for a dynamic programming algorithm to compute the maximum total value (you can use bottom up dynamic programming or memoization).

3. (1) $n = 5$ and $k = 3$

$$\begin{aligned} 5/3 &= (5-1/3-1) + (5-1/3) \\ &= (4/2) + (4/3) \\ &= \frac{12+8}{6} \\ &= 10/3 \end{aligned}$$

(2) Pseudo code

```
initialize n to 5 and k to 3
call the function binomial (n, k)
binomial (n, k)
declare i and c[k+1]
initialize the array c[0] to 1
for i = 1 to k then
c[i] = ((c[i-1] * (n-i+1))/i
return c[k]
end for
```

4. (1)

Player 1 Selection	Player 2 Selection
30	-10
-100	-20 + 25 = 5
30 + 5 + (-60) = 25	50
sum = 95	sum = 45

Difference = 45 - (-95) = 140

(2) (a) Base Case:

if k = 0 (ie, there is on elements in the array), there is nothing to select. Thus the S(0) = 0.

(b) Recursive case:

if k is not zero. Then the player 1 selects either 1 or 2 or 3 elements from the end of A, then the second player selects 1 or 2 or 3 elements from the remaining array.

Thus, to get optimal solution, we should select a minimum of the difference of S(k) - S(k-1), S(k) - S(k-2) and S(k) - S(k-3).

The recursive definition of S is as follows:

$$S(k) = \begin{cases} 0 & \text{if } k = 0 \\ \min\{S(k) - S(k-1), S(k) - S(k-2), S(k) - S(k-3)\} & \text{otherwise} \end{cases}$$

5. (1) (a) $n = 3$ and $m = 5$

Value of items $v = \{5, 30, 17, 32, 40\}$

So from the above items, 3 items which provides maximum profit 40, 32, 30.

(b) 1. Sort the given items array in descending order.

2. Choose from index zero as per capacity of carrying for maximum profit.

(2) (a) To consider an effective case for profit

We carry only n kilograms of weight, so will do here is divide value from weight to get effective value for profit and then choose highest from that to complete 15 kilogram to gain maximum profit.

We have $n = 15$, $m = 5$ and $v = \{5, 30, 17, 32, 40\}$ and weights are $w = \{2, 4, 3, 6, 15\}$.

So $v/w = \{2.5, 7.5, 5.667, 5.333, 2.667\}$

now lets sort these weights in descending order

$\{7.5, 5.667, 5.333, 2.667, 2.5\}$

So first we will take 7.5 which is 4kg then add the value is 30, then 5.67 whose weight is 3 and the value is 17, then 6 whose value is 32 and then 2kg will take from 2.67 which will be calculated as profit of 5.34

So total profit will be $30 + 17 + 32 + 5.35 = 84.34$.

Greedy algorithm will give profit of 40.

(b) (i) Base case:

```
if (m == 0 || itemName == weight.length){
    return 0;
}
```

(ii) Recursive problem.

```
if (weight[itemNum] > m)
    return knapsack(weight, val, m, itemName + 1);

int rightMaximum = val[itemName] + knapsack(weight, val, w-weight[itemNum], itemNum-1);
item leftMaximum = knapsack(weight, val, m, itemName + 1);
    return max(rightMaximum, leftMaximum);
}
```