

CS5800: Algorithms — Virgil Pavlu

Homework 4

Name:

Collaborators:

Instructions:

- Make sure to put your name on the first page. If you are using the \LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3rd edition. While the 2nd edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3rd edition.

1. (15 points) Exercise 16.2-3. Use induction to argue correctness.

Solution:

let us take an example to understand this better :

w	10	20	30
v	30	16	15

Let the maximum capacity be $W = 50$ lbs

This becomes a case of Greedy Algorithm . **WHY ?**

Because as per the greedy approach we will take the highest valued item first and then move to subsequent items in the list , maximising the VALUE.

we take the highest valued item first : **item 1 weighing 10lbs with value 30 unit**

Our Remaining capacity is $50 - 10 \text{ lbs} = 40 \text{ lbs}$ AND Current Value is 30

Now we move to the second item : **weighing 20 lbs and value 16 unit**. We take this.

Our Remaining capacity is $40 \text{ lbs} - 20 \text{ lbs} = 20 \text{ lbs}$ AND Current Value is $30 + 16 = 46$ Units

Now moving to the third item , since the item weight (30 lbs) is more than the remaining weight in the knapsack (20 lbs) , we do not take this item and this is the maximum value we can get.

Max value : 46 units

Weight of items : 30 lbs

Proof that this solution is optimal :

Let $S = [p_1, p_2, \dots, p_n]$ be the greedy solution where p represents items taken AND and $\tilde{S} = [q_1, q_2, \dots, q_k]$ is the optimal solution.

Using mychoice (greedy) algorithm, we are choosing the highest value item first. This implies:

$$V_s > V'_s$$

Optimal solution is only possible when $S[p_1+p_2+\dots+p_n] = \tilde{S} [q_1+q_2+\dots+q_n]$

Case 1 :

$V_s = V'_s$, then we are done!!!

Case 2 :

$V_s > V'_s$, EXCHANGE

we can select any other item and exchange it with other selected item of same weight, and make the my choice as the optimal solution

Case 3 :

$V_s < V'_s$. This case is NOT POSSIBLE because the optimal solution will take the MAX value.

2. (15 points) Exercise 16.2-4.

Solution:

ALGORITHM :

```
While start < end
|  $x \leftarrow \text{position where water finishes};$ 
| If  $x > \text{end}$  :
|   find refill station
|   fill water
|   currentPosition = x
| Else
|   currentPosition = end;
| :
End while
```

We will use Greedy strategy to solve this problem . Greedy solution because the professor covers MAX Distance he can with 2l of water before he takes his first stop (pitstop)

Assumptions :

1. We are assuming there exists stop(s) denoted by X before the Professor runs out of 2l water, where $x_0 < x_1 < \dots < x_n$
2. At every stop , Professor refills his water bottle to the fullest , ie, 2l

As per the Greedy approach, we cover the maximum distance with 2l of water

- Each of our subproblem is covering Max distance with 2 l of water till first pitstop x_0 .
- Then from x_0 , again with 2 l of water till x_1 .
- Repeat till final destination is reached.

Let greedy solution (my choice) be represented by S and optimal solution be represented by \tilde{S} .
Let y_0 be the first pitstop taken by \tilde{S} algorithm .

Then ,

For $k = 1 : y_0 \leq x_0$ (because Greedy Solution S covers the maximum possible distance)

Using inductive hypothesis, if something is true for $k=1$, it will be true for any value of k :
 $y_1 \leq x_1$ and so on till $y_k \leq x_n$

This proves :

$\tilde{S} \leq S$ and thus my choice (Greedy) yeilds the optimal solution to this problem.

3. (15 points) Exercise 16.2-5.

Solution:

Greedy Algorithm :

1. Sort all the points of Set X . Let "s" be the sorted array value for X where , $s = s_1 < s_2 \dots < s_n$
2. Iterate through "s".
3. For any $i \in 1, 2 \dots n$, take the first interval as s_i
4. Since the interval is "unit sized", the ending of the interval would be s_{i+1} , ie, $[s_i, s_{i+1}]$
5. Remove the points falling in this interval
6. Repeat for all the elements till set "s" becomes empty

Running Time:

time to sort + time to iterate through elements

$$T(n * \log n) + T(n)$$

$$\theta(n * \log n)$$

Proof why Greedy will work :

Let greedy solution (my choice) be represented by S and optimal solution be represented by \tilde{S} .

let intervals using S be $x_0, x_1, x_2 \dots$

let intervals using \tilde{S} be $y_0, y_1, y_2 \dots$

We take an example : $s = [0.5, 1.3, 1.5, 2.3, 3.2]$

in this example, using the greedy choice we will have 2 intervals $[0.5, 1.5]$ AND $[2.3, 3.3]$

Since my choice (greedy) for every subproblem, chooses the leftmost value as the starting interval of the set, there is no point below that , so even if we take any starting interval lesser than the leftmost point, it will not be able to cover the maximum number of points in least number of intervals.

Using inductive hypothesis, if something is true for $k=1$, it will be true for any value of k :

For $k=1$: S choses the first value of the array as its first interval, ie, first interval x_0 will be 0.5 to 1.5 .

For \tilde{S} : it will have any value lesser than or equal to x_0 , ie, lesser than or equal to 0.5 in this case .

This is because greedy takes the MAX VALUE .

$$x_0 \geq y_0$$

Similarly, $x_n \geq y_k$

Using this , we can say $S \geq \tilde{S}$

Therefore, solution S which uses the Greedy approach covers the maximum number of points with minimum number of intervals and is the optimal solution to the problem .

4. (15 points) Exercise 16.2-6.

Solution:

collaborator - Dhruv Saini

We can change the sorting algorithm used in fractional knapsack to solve it in linear time. Sorting Algo such as Quickselect algorithm can be used here .

QUICKSELECT ()

1. divide the input array into $[n/5]$ groups

2. Run insertion sort to sort the elements
3. find the median of $[n/5]$ elements from the sorted groups
4. now call quickselect to find the MEDIAN of all the groups.
5. Using middle elements as a pivot, we call PARTITION recursively.
6. Call QUICKSELECT recursively

After the Sorting, we proceed as we normally do for Fractional Knapsack .

```

FractionalKnap()
{
    qi = vi/wi //calculating value by weight for all items
    Median = Quickselect(qi, n/2) \\sorting the value/weight ratio
    \\now we partition into subarray and
    left : qi < median
        mid : qi = median
        right : qi > median

    if (right wi > W ) // W is the max capacity
    {
        recurse in right side of array and fill the knapsack
    }
    if (right wi <= W)
    { fill knapsack with all items in the right array }
}

```

Solving Fractional Knapsack :

1. Find Value/Weight for all items .
2. Use Quickselect() to sort all Value/Weight
3. now we have to select the first item with the highest Value/Weight .
4. Now check if the weight of item is less than max capacity of the Knapsack, then move to the next

item .

5. Repeat till we reach max capacity.

5. **(Extra Credit 15 points)** Exercise 16.3-3. First prove by induction that $\sum_{i=0}^{n-1} F(i) = F(n+1) - 1$

6. **(20 points)** Problem 16-1, (a), (b) and (c).

Solution:

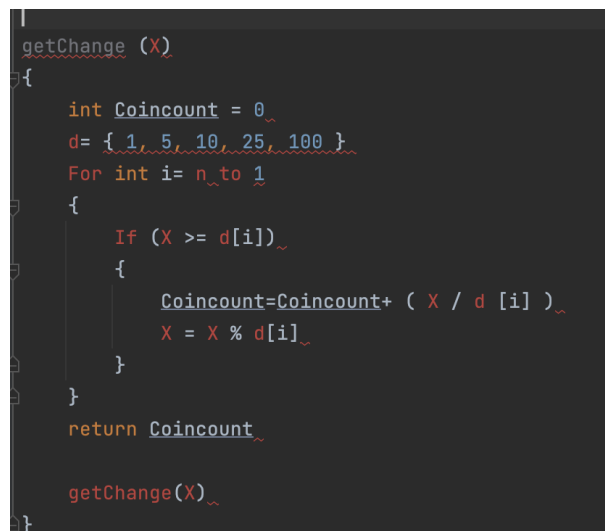
(a) To approach this problem, we will use Greedy strategy . Suppose we want change for an amount denoted by X in denominations denoted by D where D= [1,5,10,25]

Algorithm:

1. Use the greatest value denomination as a change for X
2. Find remainder R as X- R 3. Again use the greatest value denomination as a change for X
4. Repeat till X = 0

Pseudocode :

(NOTE: I couldnt make proper alignment of pseudocode in latex, therefore added an img)



```
getChange (X)
{
    int Coincount = 0
    d= { 1, 5, 10, 25, 100 }
    For int i= n to 1
    {
        If (X >= d[i])
        {
            Coincount=Coincount+ ( X / d [i] )
            X = X % d[i]
        }
    }
    return Coincount
    getChange(X)
}
```

Proof :

Let greedy solution (my choice) be represented by S and optimal solution be represented by \tilde{S} .

LET solution S (greedy) use c_q quarters, c_d dimes, c_n nickles and c_p pennies

LET solution \tilde{S} use c_Q quarters, c_D dimes, c_N nickles and c_P pennies

We define 4 cases for the optimal solution :

Case 1 : Pennies

We can use at most 4 pennies . This is because for every 5 pennies, we can replace it with one nickle

$$1 \leq c_p < 5$$

Case 2 : Nickle

We can use at most 1 nickle . This is because for every 2 nickles, we can replace it with 1 Dime

$$1 \leq c_n < 3$$

Case 3 : Dime

We can use at most 2 dimes . This is because for every 3 dimes, we can replace it with 1 quarter

$$1 \leq c_d < 2$$

Case 4 : Quarter

We can use any number of quarters.

Example : change for 34

using cases for optimal solution defined above, we can have at max 1 quarter + at max 1 nickle + at max 4 pennies

$$\text{ie , } c_Q = 1, c_N = 1, c_P = 4$$

Now let us solve the same problem using greedy approach :

For change for amoutn 34 using greedy , we will have $c_q = 1, c_n = 1, c_p = 4$

which is same as the Optimal solution \tilde{S}

This proves greedy is the optimal solution for this problem.

(b) denomination in power of c , $c^0, c^1 \dots c^k$

as in the pseudo code above, we are calculating "Coincount" , the count must follow this property :

$$\sum_{i=1}^k \text{Coincount}[i] * d[i] = X$$

where d[i] is the denominations AND X is the amount for which change is required
counts for each denomination must be less than c .

We take k=2, and c=5 , we get : d=[1,5,25]

in the (a) part above we have proved that the given denomination yield correct result using the greedy approach and greedy is the optimal solution .

(c) Let us take an example where denominations d = [1,9,10] and amount for which change required is 18.

if we use greedy solution here, the answer would be [10 , 1 , 1 , 1, 1, 1, 1, 1, 1]

This is clearly wrong becuase the optimal solution would be [9,9]

7. (15 points) Exercise 15.4-5. Hint: try to solve this problem using a greedy approach – it may not work; if it doesn't work, it means you must use DP and you can leave it for HW5.

Solution:

Collaborator : https://en.wikipedia.org/wiki/Longest_increasing_subsequenceEfficient_algorithms

:

The algorithm is using Dynamic Approach here.

let A be the array having $A=[a_1, a_2, \dots, a_n]$ elements.

Let $L[i]$ be the length of longest subsequence which contains $A[1]..A[i]$ elements

We can use Binary Search to search for the longest subsequence found so far and store it in L
We will have $M[l]$ at k where $M[l]$ will store the smallest value of the subsequence which is breaking the longest subsequence flow

Since this problem can not be broken down into sub problems where each subproblem can be solved individually and then merged together to get a solution , it will not use a greedy strategy.