Assignment 02

SCS 2201 - Data Structures and Algorithms III

Greedy Algorithms

Group Members :

DILHARI B.G.R.
DINELKA B.V.
DISSANAYAKA D.M.T.S.

Question 1

To solve the first question under the greedy algorithm methods we can use the knapsack method as the best approach. In knapsack, there are two types of approaches. 0/1 knapsack problem and fractional knapsack problem. In this question, the best type is using a fractional knapsack to solve this question.

According to this question, Sadun wants to maximize his wine value. so that it is an optimization problem. The weight and value of the products will be the wine barrel and bottle value respectively.

To solving the problem lets take the example in the scenario, The number of bottles in wine barrels and the price of each is in order as 1,2,3 and 6,10,12.

According to this, we have to fill five bottles from these three barrels of 1,2,2 type. It will give the maximum amount.

A candidate set

The array of the volume of barrels in bottles and the price of each barrel. Initialize two arrays volume and price. These two arrays act as the candidate set.

float volume[M]
float price[M] M=number of barrels

A selection function

Sort the candidate array with the highest price. Sorted the above mention two array according to,
price[i]/volume[i])<(price[j]/volume[j]

A feasibility function

Check the number of bottles in the current barrel and the remaining number of bottles to fill.
If selected one is feasible to add it to x[i] array

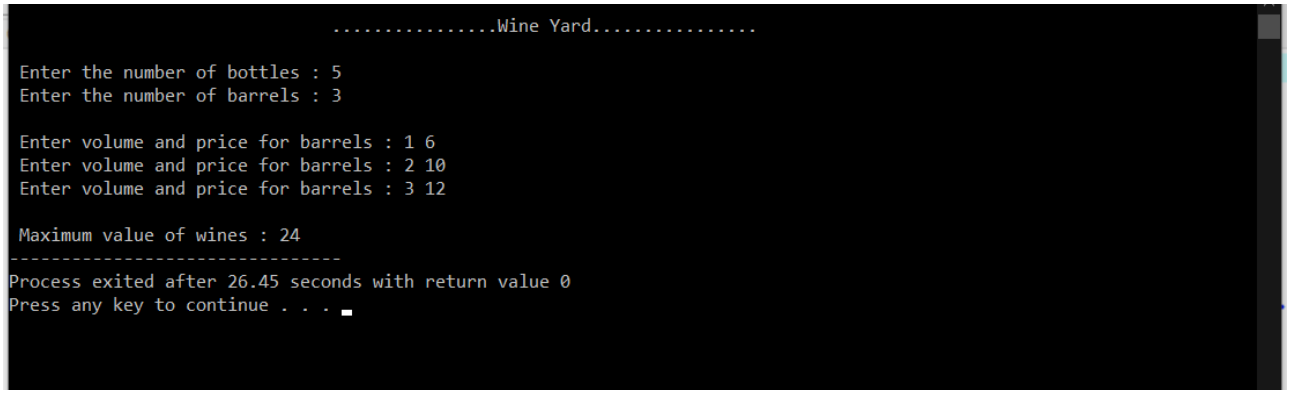
An objective function

Add the price of each filled bottles.
if solution feasible then result=result+x[i]*price

Solution function

The maximum value of wines Sandun gets from the offer
Output the result value by returning the total values of the wine bottles.

Maximum amount = (1/1*6) +(2/2*10) +(2/3*12) =24//



Solution Complexity :- O(nlogn)

Question 2

In the second question, the greedy approach is using via the position of the student and assigns a number of masks for them by considering the adjacent two students.
According to the scenario we need to compare the students by their marks. First, we compare the marks left to right by,

Marks[i+1] > Marks[i] then increment the previous value
Marks[i+1] < Marks[i] then value assign as 1

Initially, assign a value to the leftmost student as 1. Because the minimum amount of masks a student can have is 1.

Then the same process has to be done from right to left because if two students have equal marks then we need to assign a different number of masks. Here also we assign 1 for the rightmost element and compare the adjacent two student marks.

Finally, we should consider the maximum value of both of the iteration and calculate the minimum amount of mask the manel should buy. Taking the maximum value of the two values will lead us to the minimum value because it avoids the mask amount being repeated within equal marks students. So it outputs the minimum value.

A candidate set

set of marks of the student according to the seating order. According to the program marks list will be our candidate set(Marks[0..n]).

A selection function

As described in the above initially we did the comparison from left to right and then from right to left comparison to the list. Then select the max value from each mask value as per the selection function.

A feasibility function

To implement the comparison step we get the initial value of each process. Left to right marks[0] and right to left marks[n-1] as the initial comparison value and then incrementally change the value throughout the loop. Feasibility value would be the value of the mark as per the problem.

Marks[i+1] > Marks[i] then increment the previous value
Marks[i+1] < Marks[i] then value assign as 1

An objective function

The value that we get according to the comparison type we got the object for the problem. So partially it will get the value solution for the minimum mask value. After the maximum value took from the maximum function we took the value for the solution.

As per the example given in the scenario,

4 6 4 5 6 2

1 2 1 2 3 1 <= left to right comparison
1 2 1 1 2 1 <= right to left comparison
1 2 1 2 3 1 <= max values of two comparison iterations

Solution function

After executing the maximum value function we get the minimum value for the total masks. By how we predict it is a minimum. By executing the same process to the left to right and right and left we got the comparison and avoid getting the same mask amount for the equal mark students. So it should be the minimum value and others got the minimum mask amount.

Mask amount = 1 + 2 + 1 + 2 + 3 + 1 = 10//

Solution Complexity: O(n)

Question 3

The greedy approach of the third question has characteristics of the knapsack problem.

First, we need to sort out the product weight array(we are using a vector - variable-sized array) in ascending order in order to efficiently taking the minimum value throughout the algorithm.then by considering the first element of the array we are taking the weighted product which has a weigh of less than or equal to 4 units plus of minimum value. So it's container has a behavior of like a knapsack of range of 4 weights. Our knapsack is now a container.

Already array is in increasing order, so the minimum value is at the beginning. So we incrementally pop out the elements while the list has been empty. In each iteration process, we count the container we need to have for the lowest-cost way for Seetha.

By assuming that weights of products the inner loop will iterate a minimum of 4(best case) times or maximum of n times if all the values are the same(worst case) for each outer loop value. So the program will take the worst case of O(n^2) complexity for the two loops in the program.

Candidate Set

The solution list has to be modified by doing the sorting according to the ascending order. The sorting order of the product weight list provides an efficient way to execute the selection process. Let's take an example.

1 2 3 21 7 12 14 21 <= Before Sort
1 2 3 7 12 14 21 21 <= After Sorting become candidate set

Selection Set

In the selection function, we initially took the leftmost value of the list and inherently the minimum value of the list because of the sorting. Then minimum selected in the list should resulting in incrementing the container amount of the program. It is the selection function.

Feasibility Set

We took the minimum value as the feasibility value for comparing the next range values.

```
While list is not empty {
        tmp = products[0];//get the front value before pop

        while(tmp + 4 >= products[k] && products.size() != 1){
                products.erase(products.begin()); // remove the element in range
        }

        products.erase(products.begin());//remove the pivot
        containers++;
        k = 1;
}
Return the containers
```

Objective Function

According to the above pseudo-code after the comparison, we increment the value of the container variable because for the range we should assign a container. In the partial solution for 4 weight range.
While list is empty
Take 1
Pop 1,2,3 in range <= 1 container
Take 7
Pop 7 in range <= 1 container
Take 12
Pop 12,14 in range <= 1 container
Take 21
Pop 21,21 in range <= 1 container

Solution Function

Ending of the outer loop program output the total amount of the container should be assigned minimum for the products as the lowcost path.

Total containers = 1 + 1 + 1 + 1 = 4// Solution Function

Solution Complexity:- Best Case O(n) worst-case O(n^2)