## Experiment No: 5

***Title :*** Write a program to find solution to Knapsack Problem Instance.

##### Theory/Description:

Greedy algorithms are simple and straight forward. They are short sighted in their approach in the sense that they take decisions on the basis of information at hand without worrying about the effect these decisions may have in the future. They are easy to invent, easy to implement and most of the time quite efficient. Many problems cannot be solved correctly by greedy approach. Greedy algorithms are used to solve optimization problems

Greedy Approach

Greedy Algorithm works by making the decision that seems most promising at anymoment; it never reconsiders this decision, whatever situation may arise later.

As an example consider the problem of "Making Change".

Coins available are:

* + - dollars (100cents)
    - quarters (25cents)
    - dimes (10cents)
    - nickels (5cents)
    - pennies (1cent)

Problem Make a change of a given amount using the smallest possible number of coins.

Informal Algorithm

* + - Start with nothing.
    - At every stage without passing the given amount.
      * Add the largest to the coins already chosen.

Example Make a change for 2.89 (289 cents) here n = 2.89 and the solution contains 2 dollars (200 cents) 3 quarters (75 cents), 1 dime (10 cents) and 4 pennies (4 cents). The algorithm is greedy because at every stage it chooses the largest coin without worrying about the consequences. Moreover, it never change sits mind in the sense that once a coin has been included in the solution set, it remains there.

To construct the solution in an optimal way.Algorithm maintains two sets. One contains chosen items and the other contains rejected items.

The greedy algorithm consists of four (4)function.

1. A function that checks whether chosen set of items provide a solution.
2. A function that checks the feasibility of a set.
3. The selection function tells which of the candidates is the most promising.
4. An objective function, which doesnot appear explicitly, gives the value of a solution.

Structure Greedy Algorithm

* + - Initially the set of chosen items is empty i.e., solution set.
    - At each step
      * Item will be added in a solution set by using selection function.
      * IF the set would no longer be feasible
        + Reject items under consideration (and is never consider again).
      * ELSE IF set is still feasible THEN

add the current item.

Definitions of feasibility

A feasible set (of candidates) is promising if it can be extended to produce not merely a solution, but an optimal solution to the problem. In particular, the empty set is always promising why? (because an optimal solution always exists)

A greedy strategy usually progresses in atop-down fashion, making one greedy choice after another, reducing each problem to a smaller one.

Greedy-Choice Property

The "greedy-choice property" and" optimal substructure" are two ingredients in the problem that lend to a greedy strategy.

Greedy-Choice Property

It says that a globally optimal solution can be arrived at by making a locally optimal choice.

Knapsack Problem

Statement We are given n objects and a knapsack or bag. Object i has a weight wi and the knapsack has a capacity m.

There are two versions of problem

* 1. Fractional knapsack problem

The setup is same, we can take fractions of items, meaning that the items can be broken into smaller pieces so that we may decide to carry only a fraction of xiof item i, where 0 ≤ xi≤ 1. If a fraction xiof object i is placed into the knapsack, then a profit pixi is earned.

The objective is to obtain a filling of the knapsack that maximizes the total profit earned.

The idea is to calculate for each object the ratio of value/cost, and sort them according to this ratio. Then you take the objects with the highest ratios and add them until you can’t add the next object as whole. Finally add as much as you can of the next object.

So, for our example:

v(weight) = {4, 2, 2, 1, 10}

c(profit) = {12, 1, 2, 1, 4}

r = {1/3, 2, 1, 1, 5/2}

From this it’s obvious that you should add the objects: 5, 2, 3, and 4 and then as much as possible of 1.

We can choose objects like this:

Added object 5 (10$, 4Kg) completely in the bag. Space left: 11. Added object 2 (2$, 1Kg) completely in the bag. Space left: 10. Added object 3 (2$, 2Kg) completely in the bag. Space left: 8.

Added object 4 (1$, 1Kg) completely in the bag. Space left: 7. Added 58% (4$, 12Kg) of object 1 in the bag.

Filled the bag with objects worth 15.48$.

* 1. Knapsack problem

The setup is the same, but the items may not be broken into smaller pieces, so we may decide either to take an item or to leave it (binary choice), but may not take a fraction of an item.

Algorithm fractional-knapsack (w, v,W)

{

for i =1 to n

do x[i]=0

weight = 0

while weight < W

do i = best remaining item if weight + w[i] ≤W

then x[i] = 1

weight = weight + w[i] else

x[i] = (w - weight) / w[i] weight = W

return x

}

Conclusion:

Performance Analysis

If the items are already sorted into decreasing order of vi/ wi,then the while-loop takes a time in O(n);

Therefore, the total time including the sort is in O(n log n).

If we keep the items in heap with largest vi/wiat the root. Then

* + - creating the heap takes O(n)time
    - while-loop now takes O(log n) time (since heap property must be restored after the removal of root)

Although this data structure does not alter the worst-case, it may be faster if only a small number of items are needed to fill the knapsack.

One variant of the 0-1 knapsack problem is when order of items are sorted by increasing weight is the same as their order when sorted by decreasing value.

The optimal solution to this problem is to sort by the value of the item in decreasing order. Then pick up the most valuable item which also has a least weight. First, if its weight is less than the total weight that can be carried. Then deduct the total weight that can be carried by the weight of the item just pick. The second item to pickis the most valuable item among those remaining. Keep follow the same strategy until we cannot carry more item (due to weight).