CSEN3001:
DESIGN AND ANALYSIS OF ALGORITHMS

UNIT-II: THE GREEDY METHOD
Introduction

Greedy Method – general idea

- The Greedy method is the most straightforward designed technique.
- These methods are short-sighted in their approach, deciding based on the information immediately at hand without worrying about the effect this decision may have in the future.

DEFINITION:

- A problem with N inputs will have some constraints, and any subsets that satisfy these constraints are called feasible solutions.
- An optimal solution is a feasible solution that either maximizes or minimizes a given objective function.

Greedy Method – general idea

```
Algorithm Greedy (A,n)
//A[1:n] contain the 'n' inputs
{       solution:=Φ;//Initialize the solution.
        for i=1 to n do
        {
           x:=Select(A);
           if Feasible(solution) then
              solution:=Union(solution, x);
        }
        return solution;
}
```

The function selects an input from A[] and removes it. The select input value is assigned to ***x***.

- Feasible is a Boolean value function that determines whether ***x*** can be included into the solution vector.
- The function Union combines ***x*** with The solution and updates the objective function.
- The function Greedy describes the essential way that a greedy algorithm will once a particular problem is chosen and s the function subset, feasible & union are properly implemented.

Greedy Method – general idea

Suppose we have following coins are available :

1-Dollars = 100 cents

1-Quarters = 25 cents

1-Dimes = 10 cents

1-Nickel = 5 Cents

1-Pennies = 1 cent

Our aim is to pay a given amount to a customer using the smallest possible number of coins.

For example, if we must pay 276 cents possible solution, then,

Solution:

- One dollar + Seven Quarters + One penny ➔9 coins
- Two dollars + Three Quarters + One penny➔6 coins
- Two dollars + Seven dimes + One Nickel + One Penny ➔11 coins.

Knapsack Problem

We are applying the Greedy method to the Knapsack problem.

- Inputs: n objects or a bag. The object i weighs w_i, and the capacity of the knapsack is m.
- If a fraction x_i, $0 \leq x_i \leq 1$, of object i is placed into the knapsack, then a profit $p_i x_i$ is earned.
- The objective is to obtain a filling of the knapsack that maximizes the total profit earned.
- Maximize $\sum_{i=1}^{n} p_i x_i$ $\quad subject\ to\ \sum_{i=1}^{n} w_i x_i \leq m$ and $0 \leq x_i \leq 1, 1 \leq i \leq n$

- *Lemma: If case the sum of all the weights is $\leq m$, then $x_i = 1$, $1 \leq i \leq n$ is an optimal solution,. So let us assume the sum of the weights exceeds m. Now all x_i's can not be 1.*
- *Lemma:* ***An optimal solution will fill the knapsack exactly***

Knapsack Problem

- There are so many ways to solve this problem, which will give many feasible solutions for which we have to find the optimal solution.
- It will generate only one solution that is feasible and optimal.
- First, we find the profit & weight rates of every object and sort it according to the descending order of the ratios.
- Select an object with the highest profit/weight ratio and check whether its weight exceeds the bag's capacity.
- If so, place 1 unit for the first object and decrease the bag's capacity by the weight of the object you have placed.
- Repeat the above steps until the bag's capacity becomes less than the weight of the object you have selected In this case, place a fraction of the object and come out of the loop.

Knapsack Algorithm

Algorithm Greedyknapsack (m,n)

// p[1:n] and the w[1:n]contain the profit & weight ration of the n object ordered. such that p[i]/w[i] >=p[i+1]/W[i+1] m is the Knapsack size, and x[1:n] is the solution vertex.

```
{
        for i=1 to n do
        x[i]=0.0; U=m;
                for i=1 to n do
                {
                        if (w[i]>u)then break;
                        x[i]=1.0;U=U-w[i]
                }
                if(i<=n)then x[i]=U/w[i];
}
```

Knapsack Algorithm

Theorem 4.3 If $p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n$ then algorithm GREEDY_KNAPSACK generates an optimal solution to the given instance of the knapsack problem.

Proof: Let $X = (x_1, \ldots, x_n)$ be the solution generated by GREEDY_KNAPSACK. If all the x_i equal one then clearly the solution is optimal. So, let j be the least index such that $x_j \neq 1$. From the algorithm it follows that $x_i = 1$ for $1 \leqslant i < j$, $x_i = 0$ for $j < i \leqslant n$ and $0 \leqslant x_j < 1$. Let $Y = (y_1, \ldots, y_n)$ be an optimal solution. Without loss of generality we may assume that $\sum w_i y_i = M$. Let k be the least index such that $y_k \neq x_k$. Clearly, such a k must exist. It also follows that $y_k < x_k$. To see this, consider the three possibilities: $k < j$, $k = j$ or $k > j$.

Knapsack Algorithm

(i) If $k < j$ then $x_k = 1$. But, $y_k \neq x_k$ and so $y_k < x_k$.
(ii) If $k = j$ then since $\sum w_i x_i = M$ and $y_i = x_i$ for $1 \leq i < j$, it follows that either $y_k < x_k$ or $\sum w_i y_i > M$.
(iii) If $k > j$ then $\sum w_i y_i > M$ which is not possible.

Now suppose we increase y_k to x_k and decrease as many of $(y_{k+1}, \ldots, y_n)$ as is necessary so that the total capacity used is still M. This results in a new solution $Z = (z_1, \ldots, z_n)$ with $z_i = x_i$, $1 \leq i \leq k$ and $\sum_{k<i\leq n} w_i(y_i - z_i) = w_k(z_k - y_k)$. Then, for Z we have

$$\sum_{1\leq i\leq n} p_i z_i = \sum_{1\leq i\leq n} p_i y_i + (z_k - y_k)w_k p_k/w_k - \sum_{k<i\leq n} (y_i - z_i)w_i p_i/w_i$$

$$\geq \sum_{1\leq i\leq n} p_i y_i + [(z_k - y_k)w_k - \sum_{k<i\leq n} (y_i - z_i)w_i]p_k/w_k$$

$$= \sum_{1\leq i\leq n} p_i y_i$$

Knapsack Algorithm

If $\sum p_i z_i > \sum p_i y_i$ then Y could not have been an optimal solution. If these sums are equal then either $Z = X$ and X is optimal or $Z \neq X$. In this latter case, repeated use of the above argument will either show that Y is not optimal or will transform Y into X, showing that X too is optimal. □

THANK YOU