UNIT 1 GREEDY TECHNIQUE

Structure

1.0 INTRODUCTION

In Greedy algorithm, the solution that looks best at a moment is selected with the hope that it will lead to the optimal solution. This is one of the approaches to solve optimization problems. What is an optimization problem? An optimization problem is a problem which needs the best (or optimal) solution among all the possible solutions of a problem. For example, suppose we are planning to travel from location 'A' to location 'B' and there are different modes like bike, car, train, or plane, to complete this journey. The problem is to determine which mode is the best to complete this journey. So, this is an optimization problem as we want the best mode among all the possible modes to complete this journey and the best mode represents the optimal solution for this problem.

Let say, if we need to cover this journey with minimum cost, then this optimization problem will be termed as a minimization problem. Further, we need to complete this journey within 12 hours due to some urgency. This will act as a constraint for our problem. So, the optimization problem can be framed as a minimization problem in which an optimal solution is the mode which incurs minimum cost and does not take more than 12 hours to complete this journey.

Suppose, it is given that the time taken to complete this journey on bike and car is more than 12 hours while it takes less than 12 hours on a train and a plane. Thus, train and plane are the possible modes (or solutions) to complete this journey. Therefore, these both solutions are termed as feasible solutions as they are satisfying the constraint of our problem. Formally, a solution which satisfies all constraints of the problem is termed as feasible solution. A solution which is best among all feasible solutions is termed as optimal solution. Assuming train incurs the minimum cost to complete this journey, then train will be the optimal solution for our problem. In case of maximization

ignou THE PEOPLE'S UNIVERSITY

optimization problem, the optimal solution to the problem will be a maximum solution.

One more example to understand greedy approach, consider the scenario of a job recruitment process. During job interview, there are different rounds like, written, technical, and HR, to select the best candidate. Now, one possible approach to do so is to allow every candidate to appear in each round and selects the best candidate. However, this approach would be time consuming. Another possible approach would be to filter out some candidates at each round based on some criteria and then, selects the one who clears the last round. This would be less time consuming. So, the idea behind greedy approach is to select the currently best solutions among the given set of solutions based on some criteria with the hope that it will lead to overall best solution.

However, the solution returned by greedy approach may not be always optimal and depends upon the criteria. Moreover, greedy approach may not be appropriate for some optimization problems where other approaches can be used for solving them like, Dynamic programming and Branch and Bound.

Problem Formulation:-In an optimization problem, problem definition is termed as cost function (*f(x)*) and the maximization/minimization of the cost function corresponds to the objective function of the optimization problem. For example, the general mathematical formulation of a maximization optimization problem is defined as follows:

$$\text{Maximize}_{\{x \in S\}} f(x)$$

$$\text{such that: } a_i(x) \geq 0,$$

$$b_j(x) = 0, \text{ and}$$

$$c_k(x) \leq 0.$$

where, x is the considered solution from the set of solutions (S) while $a_i(x)$, $b_j(x)$, and $c_k(x)$ correspond to the possible set of constraints on an optimization problem. For solving such problems, greedy approach is widely popular and sometimes applicable.

Generally an optimization problem has n inputs (call this set as **input domain** or **Candidate set**, A), we are required to obtain a subset of A (call it ***solution set***, S) that satisfies the given constraints or conditions. Any subset S, which satisfies the given constraints, is called a ***feasible*** solution. We need to find a feasible solution that maximizes or minimizes a given objective function. The feasible solution that does this is called an **optimal solution**.

How does a greedy algorithm work?

A greedy algorithm proceeds step–by-step, by considering one input at a time. At each stage, the decision is made regarding whether a particular input (say x) chosen gives an optimal solution or not. Our choice of selecting input x is being guided by the selection function (say ***select***). If the inclusion of x gives an optimal solution, then this input x is added into the partial solution set. On the other hand, if the inclusion of that input x results in an infeasible solution, then this input x is not added to the partial solution. The input we tried and rejected is never considered again. When a greedy algorithm works correctly, the first solution found in this way is always optimal. In brief, at each stage, the following activities are performed in greedy method:

1. First we select an element, say, from input domain A.
2. Then we check whether the solution set S is feasible or not. That is we check whether x can be included into the solution set S or not. If yes, then solution set. If no, then this input x is discarded and not added to the partial solution set S. Initially S is set to empty.
3. Continue until S is filled up (i.e. optimal solution found) or A is exhausted whichever is earlier.

Note: From the set of feasible solutions, particular solution that satisfies or nearly satisfies the objective of the function (either maximize or minimize, as the case may be), is called ***optimal solution***.

1.1 OBJECTIVES

After going through this unit , you will be able to :

- Formulate a problem as an optimization problem.
- Define the basic concept of greedy technique.
- Write a general formfor greedy technique.
- Formulate fractional Knapsack , task scheduling algorithm and Huffman code problems
- Write algorithms for fractional Knapsack problem. Task scheduling and Huffman code.
- Calculate time complexities of algorithms.

1.2 SOME MORE EXAMPLES TO UNDERSTAND GREEDY TECHNIQUES

Example 1: Suppose there is a list of tasks along with the time taken by each task. However, you are given with limited time only. The problem is which set of tasks will you be doing so that you can complete maximum number of tasks in the given amount of time.

Solution: The intuitive approach would be greedy approach and the task selection criteria would be to select the task with the slowest amount of time. So, the first task will be that task which takes minimum time. The next task would be the one that takes minimum time among the remaining set of tasks and so on.

Example 2: Consider a bus which can travel up to 40 kilometers (Km) with full tank. We need to travel from location 'A' to location 'B' which has distance of 95 Km as depicted in Figure 1.

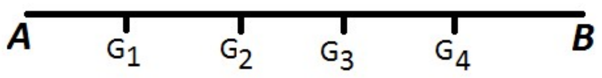


Figure 1: Representing the scenario of Example 2.

In between ‘A’ and ‘B’, there are four gas stations, G_1, G_2, G_3, and G_4, which are at distance of 20 KM, 37.5 KM, 55 KM, and 75 KM, from location ‘A’ respectively. The problem is to determine the minimum number of refills needed to reach the location ‘B’ from location ‘A’.

Solution: Suppose, the tank refill decision criteria is considered to refill at the gas station which is nearest when the tank is about to get empty. According to the criteria, if we start from location ‘A’, the tank will be refilled at the second gas station (G_2) as it is 37.5 KM from ‘A’. After this, we need to refill at the fourth gas station (G_4) which is at a distance of 37.5 KM from G_2. From G_4, we can easily reach the location ‘B’ as it is 20 KM. Therefore, the number of refills required is two as represented in the Figure 2.

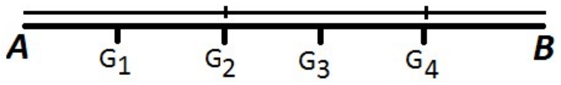


Figure 2: Solution to the scenario of Example 2. Here, the marks over G_2 and G_4 represent the refilling gas stations

1.3 FORMALIZATION OF GREEDY TECHNIQUE

For a given problem with ‘n’ input values, the greedy approach will decide criteria to select values, termed as selection criteria, and will run for ‘n’ times. In each run, following steps will be performed:

1. It will perform selection based upon the selection criteria. This returns a value from the considered input values and also removes the selected value from the input values.
2. It will check the feasibility of the selected value.
3. If the solution is feasible then the selected value is added to the solution set. Else step 1 is repeated.

Based on the above discussion, we develop a template for writing a greedy algorithm

```
Greedy_Algo_Template(A, n)

    /* Input: (a) A input domain (or Candidate set ) A of size n, from which solution is
to be obtained. */
              (b) Criteria to select input values
    /* function select (A: candidate_set) return an element (or candidate). */
    /* function solution (S: solution set_set) return Boolean  */
    /* function feasible (S: solution_set) return Boolean  */
    /* Output: An optimal  solution set S, where S, which maximize or minimize the
              selection criteria w. r. t. given constraints */

{
        S ← { }                               /*Initially a solution set S is empty */
        While ( not solution(S) andA ≠ ϕ )
        { $x \leftarrow \mathbf{select}(A)$
        A ← $\mathbf{A} - \{x\}$                       /* once x is selected , it is removed from A*/
        if ( feasible(S U $\{x\}$) then           /* x is now checked for feasibility*/
        S ← S U$\{x\}$
        }
        If (solution (S))                      /* if a solution is found */
         return S;
        else
        return " No Solution"
        }                                     /* end of while*/
```

Figure 3: Greedy Algorithm Template

Applying the template, let us solve a problem.

Example- 3: To illustrate the same, consider a list of jobs as given in Table 1:

Table 1: List of the jobs.

Jobs (J)	**J1**	**J2**	**J3**	**J4**	**J5**
Time required(T)	3	7	1	2	8

In this problem, second row represents the time taken to accomplish the corresponding job. The problem is to count maximum number of jobs that are possible in the given time limit, i.e., T = 6.

Solution:

Input:

a. A = {J1, J2, J3, J4, J5} and n = 5.

b. Criteria: Select the task that has minimum time.

Step1: Select the task 'J3' as it has minimum time of 1.

Step2: This solution is feasible according to the problem definition

Step3: Add this solution to the set S; S = {J3}. Remaining time limit = 5.

Step4: Removing this task (i.e., J3) from 'A'. Now' A = {J1, J2, J4, J5}.

Step5: Select the task 'J4' as it has minimum time of 2.
Step6: This solution is feasible according to the problem definition.
Step7: Add this solution to the set S; S = {J3, J4}. Remaining time limit = 3.
Step8: Remove this task from 'A'. Now' A = {J1, J2, J5}.
Step9: Select the task 'J1' as it has minimum time of 3.
Step10: This solution is feasible according to the problem definition
Step11: Add this solution to the set S; S = {J3, J4, J1}. Remaining time limit = 0.
Step12: Removing this task from 'A'. Now', A = {J2, J5}.
Step13: Select the task 'J2' as it has minimum time of 7.
Step14: This solution is infeasible according to the problem as remaining time is 0.
Step15: Removing this task from 'A'. Now, A = {J5}.
Step16: Select the task 'J5' as it has minimum time of 8.
Step17: This solution is infeasible according to the problem as remaining time is 0.
Step18: Removing this task from 'A'. Now', A = {}.

Therefore, maximum number of jobs that can be done within the given time limit is three, if we follow greedy approach

1.4 AN OVERVIEW OF LOCAL AND GLOBAL OPTIMA

Local optima or local optimal solutions correspond to the set of all the feasible solutions that are best locally for an optimization problem. Global optimal solution corresponds to the best solution of the optimization problem. In an optimization problem, there may be many local optima but there is only one global optimal solution. To understand these concepts, consider the 2D- plot illustrated in Figure 3which represents the solution space of some function for the minimization problem. Here, x-axis represents the range of *x* variable while y-axis corresponds to the different feasible solutions attained by the considered function *f(x)*over different values of *x*. This type of curve is generally termed as a non-convex curve. Here, the global optimal solution is only one as no other solution is better than this objective function value in terms of minimization. However, there are many local optimal solutions as they have better (or minimum) objective function values than other solutions in the nearby regions.

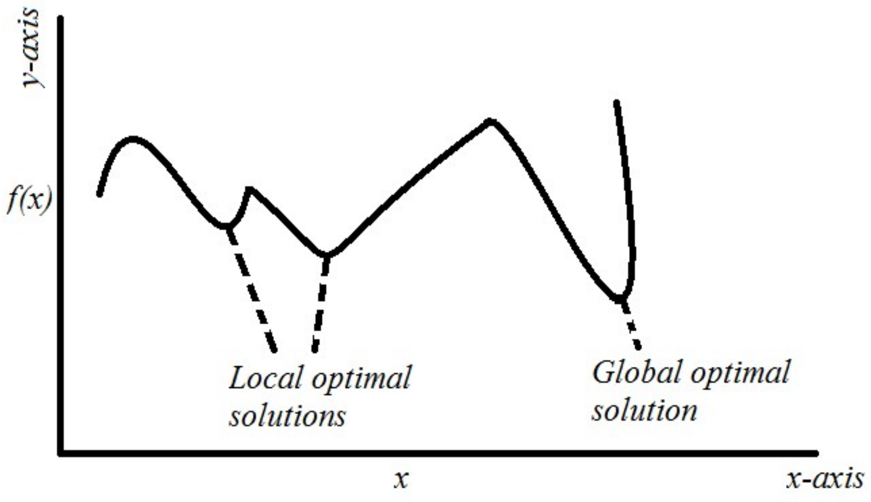


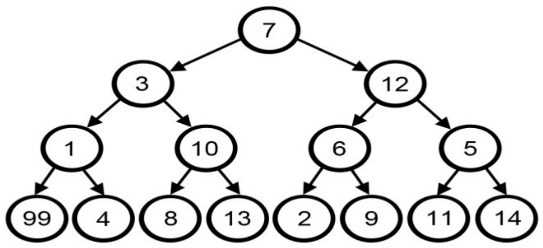
Figure 3: A solution space of some function for the minimization problem.

In Example 2, the solution for minimum number of refills is two according to the considered decision criteria. This is the global optimal solution. However, if tank refilling is performed at each station, then the solution for number of refills would be four. This will correspond to the local optimal solution as possible solution for minimum number of refills is two. Therefore, it is worthy to note that every feasible solution is not local optimal solution and every local optimal solution is not considered as global optimal solution.

☞ **Check Your Progress-1**

Question 1: What is the main step of greedy approach that effects the obtained solution?

Question 2: In the graph below, a greedy algorithm is trying to find the longest path through the graph (the number inside each node contributes to a total length). To do this, it selects the largest number at each step of the algorithm. With a quick visual inspection of the graph, it is clear that this algorithm will not arrive at the correct solution. What is the correct solution? Why is a greedy algorithm ill-suited for this problem?



1.5 FRACTIONAL KNAPSACK PROBLEM

This is an optimization problem which we want to solve it through greedy technique. In this problem, a Knapsack (or bag) of some capacity is considered which is to be filled with objects. We are given some objects with their weights and associated profits. The problem is to fill the given Knapsack with objects such that the sum of the profit associated with the objects that are included in the Knapsack is maximum. The constraint in this problem is that the sum of the weight of the objects that are included in the Knapsack should be less than or equal to the capacity of the Knapsack. However, the objects can be included in fractions to the Knapsack because of which this problem is termed as Fractional Knapsack problem. There is another kind of Knapsack problem, termed as 0-1 knapsack problem, in which the objects are not be considered in fraction i.e., the whole object is included in the Knapsack. That why, it is termed as 0-1 knapsack problem. Here we focus on a Fractional Knapsack Problem because Greedy technique works for fractional problem only.

Formulation of a problem

The fractional knapsack problem is defined as:

- Given a list of n objects say $\{O_1, O_2, \dots\dots, O_n\}$ and a Knapsack (or a bag).
- Capacity of Knapsack is W
- Each object O_i has a w_i weight and a profit of p_i
- If a fraction x_i (where $x_i \in \{0, \dots\dots, 1.\}$)of an object O_i is placed into a knapsack then a profit of $p_i x_i$ is earned.

The **problem** (or Objective) is to fill a knapsack (up to its maximum capacity M) which maximizes the total profit earned.
Mathematically:

Maximumize (the profit) $\sum_{i=1}^{n} p_i x_i$; subjected to the constraints

$$\sum_{i=1}^{n} w_i x_i \le W \text{ and } x_i \in \{0, \dots\dots 1\}, 1 \le i \le n$$

Note that the value of x_i will be any value between 0 and 1 (inclusive). If any object is completely placed into a knapsack then its value is 1 (i.e. $x_i = 1$) , if we do not pick (or select) that object to fill into a knapsack then its value is 0 (i.e. $x_i = 0$) . Otherwise if we take a fraction of any object then its value will be any value between 0 and 1.

To understand this problem, consider the following instance of a knapsack problem:

number of objects; $n = 3$
Capicity of Knapsack; W = 20
$(p_1, p_2, p_3) = (25, 24, 15)$
$(w_1, w_2, w_3)=(18, 15, 10)$

To solve this problem, Greedy method may apply any one of the following strategies:

- From the remaining objects, select the object with maximum profit that fit into the knapsack.
- From the remaining objects, select the object that has minimum weight and also fits into knapsack.
- From the remaining objects, select the object with maximum p_i/w_i that fits into the knapsack.

Let us apply all above 3 approaches on the given knapsack instance:

Approach	(x_1, x_2, x_3)	$\sum_{i=1}^{3} w_i\, x_i$	$\sum_{i=1}^{3} p_i\, x_i$
1	$\left(1, \frac{2}{15}, 0\right)$	18+2+0=20	28.2
2	$\langle 0, \frac{2}{3}, 1\rangle$	0+10+10=20	31.0
3	$\langle 0,1, \frac{1}{2}\rangle$	0+15+5=20	31.5

Approach 1: (selection of object in decreasing order of profit):

In this approach, we select those object first which has maximum profit, then next maximum profit and so on. Thus we select 1st object (since its profit is 25, which is maximum among all profits) first to fill into a knapsack, now after filling this object (w_1=18)(into knapsack remaining capacity is now 2 (i.e. 20-18 = 2). Next we select the 2nd object, but its weight w_2=15, so we take a fraction of this object (i.e.$x_2 = \frac{2}{15}$). Now knapsack is full (i.e.$\sum_{i=1}^{3} w_i x_i = 20$) so 3rd object is not selected. Hence we get total profit $\sum_{i=1}^{3} p_i x_i$= 28 units and the solution set $(x_1, x_2, x_3) = \left(1, \frac{2}{15}, 0\right)$

Approach 2: (Selection of object in increasing order of weights):

In this approach, we select those object first which has minimum weight, then next minimum weight and so on. Thus we select objects in the sequence 2nd then 3rd then 1st. In this approach we have total profit $\sum_{i=1}^{3} p_i x_i$ = 31.0 units and the solution set $(x_1,\ x_2,\ x_3) = \left(0,\ \frac{2}{3},\ 1\right)$.

Approach 3: (Selection of object in decreasing order of the ratio p_i/w_i). In this approach, we select those object first which has maximum value of p_i/w_i , that is we select those object first which has maximum profit per unit weight. Since (p_1/w_1 , p_2/w_2 , p_3/w_3) = (1.3, 1.6, 1.5). Thus we select 2nd object first , then 3rd object then 1st object. In this approach we have total profit $\sum_{i=1}^{3} p_i x_i$ = 31.5 units and the solution set $(x_1,\ x_2,\ x_3) = \left(0,\ 1,\ \frac{1}{2}\right)$.

Thus from above all 3 approaches, it may be noticed that

- Greedy approaches **may not always yield an optimal solution**. In such cases the greedy method is frequently the basis of a heuristic approach. Sometimes a clever strategy can indeed lead to optimal solution
- Approach3 **(S**election of object in decreasing order of the ratio p_i/w_i) gives a optimal solution for knapsack problem.

A pseudo-code for solving knapsack problem using greedy approach is:

```
        Greedy Fractional-Knapsack (p[1..n], w[1..n], X [1..n], W)
Input: a) List of weights and profits of n objects {O_1, O_2, …., O_n}stored in
w[1..n] and p[1..n] respectively.
        b) Maximum capacity of the Knapsack.: W
Assume profits and weights are sorted by $\frac{p_i}{w_i}$
X[1..n] is a solution set
Output: Optimal solution

        {
1: for i← 1 to n do
2: X[i] ← 0
3: profit ← 0  /*Total profit of items packed in Knapsack
4: weight ← 0 /*Total weight of items packed in KnapSack
5: i←1
6: while (weight < W) // W is the Knapsack Capacity
{
7:if (weight + w[i] ≤ W)
8: X[i] = 1
9:weight = weight + w[i]
10: else
11: X[i] = (W-weight)/w[i]
12: weight = M
13: profit =  profit + p [i]*X[i]
14:
} end of while
}//end of Algorithm
```

Complexity for Fractional Knapsack problem using Greedy algorithm:

Sorting of n items (or objects) in decreasing order of the ratio p_i/w_i takes $O(nlogn)$ *time* Since this is the lower bound for any comparison based sorting algorithm. Line 6 of ***Greedy Fractional-Knapsack*** takes $O(n)$ time. Therefore, the total time including sort is $O(nlogn)$.

Example1 -Suppose there is a knapsack of capacity 5 Kg, we have 7 items with weight and profit as mentioned in the following table 2:

Objects	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Weight	2	3	5	7	1	4	1
Profit	10	5	15	7	6	18	3

The problem is to fill the given knapsack with objects such that profit is maximized. However, objects can be included in fractions. So, this is a maximization optimization problem with a constraint that the total weight of objects that are included in the Knapsack should be less than or equal to 5 Kg.

The appropriate criteria will be to include objects based on profit per weight as it will calculate profit by considering per unit of weight of the object. So, the selection criteria for filling the Knapsack will be to select the object with highest profit per weight. To apply this criteria, let us first calculate the profit per weight for each object which is mentioned in Table 3.

Table 3: Profit per weight for each object.

Objects	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Weight	2	3	5	7	1	4	1
Profit	10	5	15	7	6	18	3
Profit per weight	5	1.3	3	1	6	4.5	3

According to highest profit per weight, object 5 is selected as weight of this object is less than the capacity of Knapsack. So, the remaining capacity of Knapsack is 4 Kg. Next, object 1 is included as available capacity of Knapsack is larger than the total weight of object 1. The Knapsack is left with the capacity of 2 Kg. Now, the next object with highest profit per weight is object 6. But the remaining capacity of Knapsack is 2 Kg. So, we will take a fraction of object 6, i.e. 2/4 portion of the object 6, in the Knapsack. Now the capacity of Knapsack is 0 after the inclusion of 2 Kg of object 6. So, no further object will be included in the Knapsack.

The total weight of the included objects: 1x2 + 0x3 + 0x5 + 0x7 + 1x1 + (2/4)x4 + 0x1= 5
The total profit on included objects: 1x10 + 0x5 + 0x15 + 0x7 + 1x6 + (2/4)x18 + 0x3= 24

1.6 TASK SCHEDULING ALGORITHM

A task scheduling problem is formulated as an optimization problem in which we need to determine the set of tasks from the given tasks that can be accomplished within their deadlines along with their order of scheduling such that the profit is maximum. So, this is a maximization optimization problem with a constraint that tasks must be completed within their specified deadlines.
This problem is related to scheduling tasks on a single machine for their processing.Suppose you are given a set of n tasks for p on a single machine.

Each task is assigned a profit pi,and a deadline di and takes only one unit of time on a machine to get completed. There is availability of a single machine for processing of all the tasks. Abrute force approach would be to generate all subsets S of a given set of tasks , examine completion of each job in every subset by its deadline, calculate the profit among all feasible subsets and find out the feasible solution. To calculate S. i.e. the feasible solution pi of each tasks i in S needs to be summed up i.e. $\sum_{i \in s} p_i$. There could be multiple subsets of tasks is S. An optimal solution for this problem is one which gives the maximum profit value. Trying out all possible permutation of tasks in S and selecting whether the tasks in S can be processed in any of these permutation without violating the deadline. will require n permutation of S, which is a time consuming.

The pseudo-code of the task scheduling problem is as follows:

Input: 1) List of *t* tasks $\{T_1, T_2, \ldots, T_t\}$ and
2) List of deadlines, defined with each task.
3) List of Profit, associated with each task.

Output: Optimal solution

```
1. Sort the given set of tasks according to the profit associated with each
   task.
2. Make n unfilled time slots where n corresponds to the highest deadline.
3. Perform following steps for each task by considering tasks in
   decreasing order:
4. {
5.     Find an empty time slot which is closest and not missing the deadline
       of the considered task.
6.     Mark the identified slot as filled.
7.     If no such slot is found, then ignore the task.
8. }
```

To understand the above pseudo-code, consider the following example.

Example 1: Let us consider an example to understand. Suppose, there are 5 tasks and each task has associated profit in rupees. The amount of time required to complete each task is one hour. However, the deadline of each task is given in specified in Table 4.

Table 4: Details of tasks which needs to be scheduled on a machine.

Task	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**
Profit	18	22	5	7	6
Deadline	2	2	3	1	3

Solution: To solve this problem, consider the highest deadline and prepare that many number of slots of unit time to schedule the tasks. This will help in scheduling the tasks easily as there will be no task to be completed after the highest deadline. According to the given problem, the highest deadline is 3. So, there will be three slots as illustrated in Figure 4, each slot of unit time.

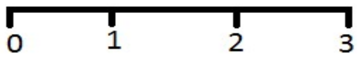


Figure 4: List of slots to schedule a task.

Now, greedy approach has to select the set of three tasks and schedule them in such a way that the total profit is maximum on their completion. For this, the selection criteria will be to select the task with highest profit. According to this criteria, task ‘T_2’ is selected and place it in the slot closest to its deadline i.e., from 1 to 2. Next, task ‘T_1’ is selected from the remaining set of tasks according to selection criteria. The deadline for task ‘T_1’ is 2, but we have already scheduled ‘T_2’ in the slot 1-2. So, look for an empty slot before this slot. The 0-1 slot is empty. So, task ‘T_1’ is placed in slot 0-1. Now, the next task with highest profit is ‘T_4’. The appropriate slot for ‘T_4’ is 0-1 according to its deadline. However, this slot is already filled and there is no other slot before 0-1. So, this task is not scheduled. Next, task ‘T_5’ is selected and placed in the 2-3 slot as it has deadline of 3. Now, the remaining task i.e., ‘T_3’, is having deadline as 3 but there is no empty slot available. So, this task is also discarded.

The schedule of given tasks is presented in Figure 5:

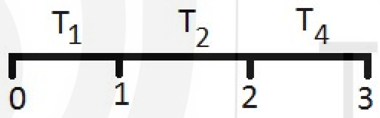


Figure 5: Schedule of selected tasks.

Therefore, the sequence of selected tasks is as follows: {T_1, T_2, T_4}
The total profit is: 18+22+7= 47

Complexity for task scheduling problem using Greedy algorithm:
In general, there will be maximum n slots for scheduling n tasks. To search, each task will explore around n slots. Therefore, the time complexity for the task scheduling is $O(n^2)$. This can be studied from the following pseudo-code too.

```
for (i=0; i<n; i++) \\ n: number of given tasks;
{
        \\ for searching the suitable slot
        for (j=0; j<n; j++) \\ There will n slots
        {
           Search empty slot for i^th task.
           Remove the j^th slot from the list of empty slots.
        }
}
```

☞ **Check Your Progress-2**

Question 1: A thief enters a house for robbing it. He can carry a maximal weight of 60 kg into his bag. There are 5 items in the house with the following weights and values. What items should thief take if he can even take the fraction of any item with him?

Item	Weight	Value
1	5	30
2	10	40
3	15	45
4	22	77
5	25	90

Question 2: Let us consider that the capacity of the knapsack W = 60 and the list of provided items are shown in the following table −

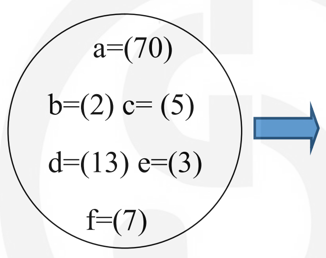
Item	**A**	**B**	**C**	**D**
Profit	280	100	120	120
Weight	40	10	20	24
Ratio (p_i/w_i)	7	10	6	5

Find the optimal solution for gaining maximum profit.

1.7 HUFFMAN CODES

Huffman coding is a greedy algorithm that is used to compress the data. Data can be sequence of characters and each character can occur multiple times in a data called as frequency of that character. Data transmitted through transmission media that can be digital communication or analog communication. That means we need to represent data in the form of bits. If there is more number of bits to represent the data, it will take more time to transmit from one source to other source. Thus, Huffman compression algorithm is applied to represent the data in compressed form called as Huffman codes. Basically compression is a technique to reduce the size of the data. Huffman coding compresses data by 70-80%. Huffman algorithm checks the frequency of each data, represent those data in the form of Huffman tree and build an optimal solution to represent each character as a binary string. Huffman coding is also known as variable length coding.

Fixed Length Encoding: Suppose we have 100-characters data file, each character having different frequency shown in the Figure 1. An order of a character can be in any form. To transfer the data, each character will be encoded into ASCII representation without using any compression algorithm. ASCII code takes 7-bits to represent the data in a binary form. Therefore, in a message each character takes 7-bits to represents the data and there are 100 characters in the message. Thus, a total 700 bits needed to transfer a message from one source to other source without any compression algorithm as shown in a Figure 6.This is also called as fixed length encoding because each character has fixed length encoding of 7-bits

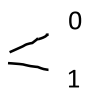


character	ASCII Code	Binary Form	Total bit/character
a	97	1100001	7*70= 490
b	98	1100010	7*2=14
c	99	1100011	7*5=35
d	100	1100100	7*13=91
e	101	1100101	7*3=21
f	102	1100110	7*7=49
Total =			700 bit

Figure 6: Frequency Table of the characters in a message

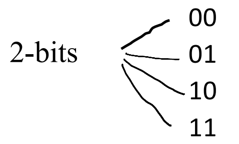
In case the above message is sent without applying any compression algorithm, a total number of bits it takes 700bit to transfer the message. In the above example we are not using all English alphabets and symbols. We are using only 6 alphabets in the message. Do we really need 7-bits to represent the data? Not really. only few bits are sufficient to represent the data. So can we use less bits to represent the data? Yes, then how many bits are needed to represent the data? As we know we can represent 128 characters with 7 bits. We have only 6 characters, we need definitely lesser than 7-bits, but how many bits are really needed?

Let’s see if we have only

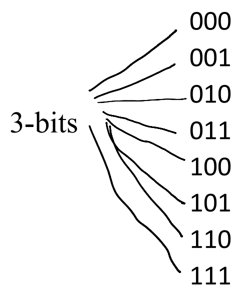
1-bit 

Then only two character can be represented.

ignou THE PEOPLE’S UNIVERSITY



Then only four characters can be represented



Then only 8 characters can be represented. We have 6 characters in the message. We need only 6 out of 8 combination to represent the message. Therefore, we need only 3-bit to represent the message.

Table 1: Shows the message and corresponding 3-bit binary representation

character	Frequency	Code	Total bit/character
a	70	000	3*70= 210
b	3	001	3*2=6
c	5	010	3*5=15
d	13	011	3*13=39
e	3	100	3*3=9
f	7	101	3*7=21
Total =			300 bits

The above method requires 3-bits only to encode each character of the message having 100 characters and take 300 bits to encode the entire message. Can we do better? Let us look at the approate algorithm.

Huffman Coding Algorithm: Variable Length Encoding

A variable-length coding can do considerably better than a fixed-length code using Huffmanencoding. Huffman encoding says, we don't need to take fixed size code for each character. In a message some characters may be appearing less number of times and some may be appearing more number of times. So if you give a small size of code for the morefrequentoccurrence of characters then, the size of the entire message will be definitely reduced. Huffman invented a greedy algorithm that finds an optimal prefix code called a Huffman code. If there is a encoding of message at sender end, there must be decoding at receiver end to read the message. In Huffman coding prefix codes (which is one type of variable length encoding scheme) are assigned to each character in such a way that no code for one character constitute the beginning of another code. For example if 10 is a code for a then 101 cannot be a code for b. Therefore, while decoding the encoded the message, Huffman coding ensures that there must be no ambiguity in prefix code assigned to each character. The main advantage of prefix code it that it simplifiesdecoding of the massage.

Some more examples of Prefix Code: Set of binary sequence P, such that no sequence in P is a prefix of any other sequence in P.

- $P = \{01, 010, 10\}$ are binary sequences/code. In the code we can see that 01 is a pre-fix of 010 sequence. Therefore, this particular set is not a prefix code.
- $P1 = \{01, 100, 101\}$ are binary sequences. In the code we can see that 01 is not prefix of 100 and 101 sequence. Just remember that 01 is present in 101 sequence but it is not pre-fix, it is a postfix. Similarly 100 is not a pre-fix in any other code and 101 is not a pre-fix in any other code as well. Therefore, this particular set of sequences is a prefix code and each prefix code can be represented as a tree.

Huffman coding works in two steps:

1. Build a Huffman tree.
2. Find Huffman code for each character.

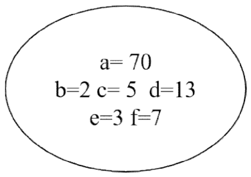
Steps for Building Huffman Tree:

1. Input is a set of M characters and each character $m \in M$ has a frequency $mfrequency$
2. Store all the characters in min-priority queue using frequencies as their values (Priority queue is a data structure that allows the opreation of search min (or max), insert, delete min (or max, respectively). If heap is used to implement priority queue , it will take O(logn) time.
3. Extract two minimum nodes x, y from min-priority queuePQ.
4. Replacing x, y in the queue with a new-node z representing their merger. The frequency of z is computed as sum of the frequencies of$x$and$y$. The node z has x as its left child and y as its right child.
5. Repeat steps 3 and 4 until one node left in the queue, which is the root of the Huffman tree.
6. Return the root of the tree.

Steps to find Huffman code for each character.

1. Traverse the tree starting from the root node.
2. An edge connecting to the left child node is labeled as 0 and 1 if it is connecting to the right child node.
3. Traverse the complete treethrough the left and right child based on the assigned value.
4. Prefix code/ Huffman code for a letter is the sequence of labels on the edge connecting the root to the leaf for that letter.

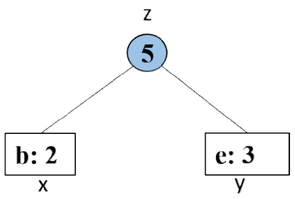
Let us understand the algorithm with an example:



(a)

Build a min-priority queue based on the frequency. Initially PQ contains 6 nodes (6 letters) and each node represents the root of tree with single node and requires fivemerges to build a tree.

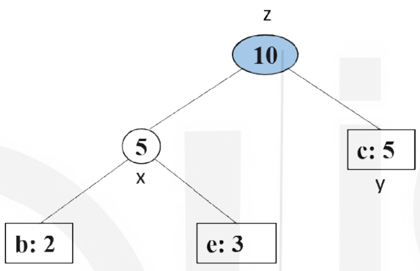
b: 2	e: 3	c: 5	f: 7	d: 13	a: 70



(b)

z: 5	c: 5	f: 7	d: 13	a: 70

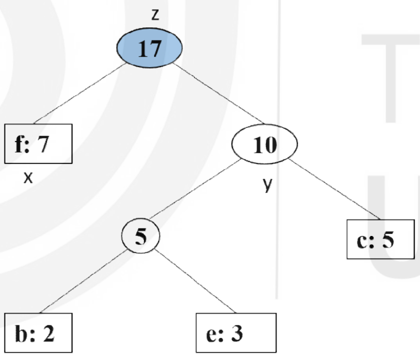
Extract two minimum node x, y from PQ. Add internal z having with frequency 2+3 = 5 in the PQ.



(c)

z: 10	f: 7	d: 13	a: 70

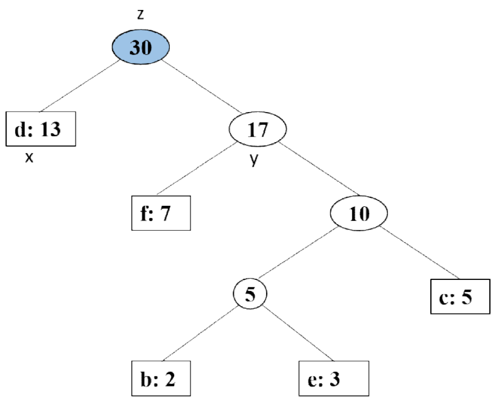
Extract two minimum node x, y from PQ. Add internal z having with frequency 5+5 = 10 in the PQ. In this tree node z=5 and c=5 have same frequency. Same frequency character become the right child in the newly created node z.



(d)

z: 17	d: 13	a: 70

Extract two minimum node x, y from PQ. Add internal z having with frequency 7+10 = 17 in the PQ.

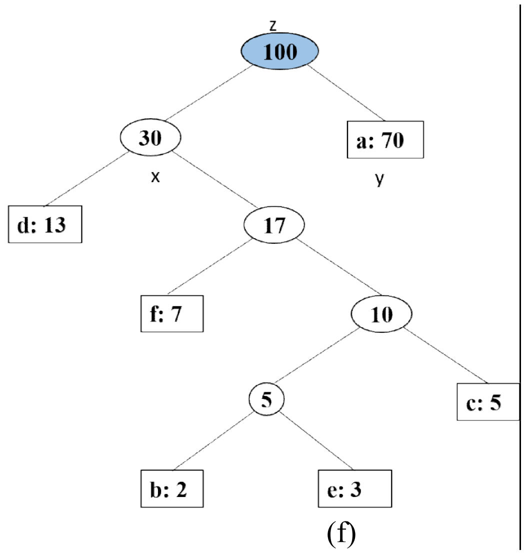


(e)

z: 30	a: 70

Extract two minimum node x, y from PQ. Add internal z having with frequency 13+17 = 30 in the PQ.

ignou
THE PEOPLE'S
UNIVERSITY



z: 100

Extract two minimum node x, y from PQ. Add internal z having with frequency 30+70 = 100 in the PQ.

Now min-priority queue contains single element. The algorithm stops here.

Figure 7: The steps of Huffman's algorithm for the frequencies given in (a).

Each part shows the content of min-priority queue sorted into increasing order by frequency. At each step, the two trees with lowest frequencies are merged. Leaves are shown as a rectangle containing a character and its frequency. Internal nodes are shown as a circles containing the sum of the frequencies of their children. (b)-(e) intermediate stages. The final Huffman tree is shown in (f)

Now in Figure 7, traverse the tree starting from root node. Take an auxiliary array, while moving to the left child write 0 in the array, while moving to the right child write 1 in array. Print array when a leaf node is encountered. The Huffman code for a character is the sequence of labels on the edges connecting the root to the leaf node for that character.

In the table 2 we can see variable size codes are used. Some character have 1 bit code some have 2 bits codes and some characters have 5 bits codes. Thus, to encode the entire 100 character message we need only 162 bits and average 1.62 bits/character. These Huffman code can be used to encode and decode the message. We need to pass the entire tree or entire table for decoding of message.

Table 2: The message and corresponding Huffman code

character	Frequency	Code	Total bit/character
a	70	1	1*70= 70
b	3	01100	5*2=10
c	5	0111	4*5=20
d	13	00	2*13=26
e	3	01101	5*3=15
f	7	010	3*7=21
Total bit =			162 bits

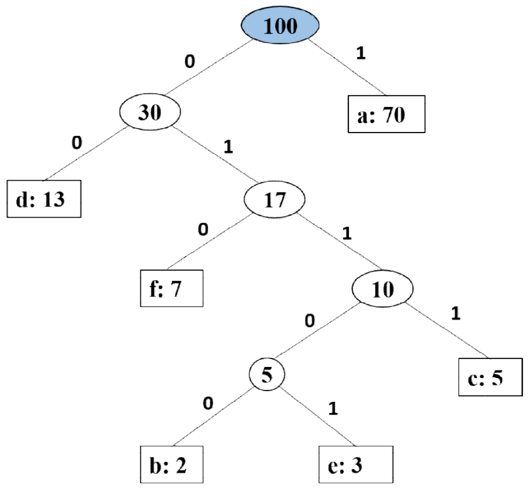


Figure 8: Huffman tree

Example

- Encode the message M= aabefdaac…..

  Corresponding encoding of message will be:
  11011000110101000110111……

a	a	b	e	f	d	a	a	c
1	1	01100	01101	010	00	1	1	0111

- Decode the message M= 0110011011100011010110101 0

  Corresponding decoding of message will be: baacdeef

01100	1	1	0111	00	01101	01101	010
b	a	a	c	d	e	e	f

Time Complexity of Huffman Algorithm

Priority queue is implemented using binary-min heap. Building min heap procedure takes $O(n)$ time in step 2. Steps 3 and 4 run exactly n-1 times. Since in each step a new node z is added in the heap. When new node added it take $O(logn)$ time to heapify. Thus, total running time of Huffman algorithm on a set of n characters id $\boldsymbol{O(n\ logn)}$.

☞ **Check Your Progress-3**

Question 1: What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers?

a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

Can you generalize your answer to find the optimal code when the frequencies, based on the first n Fibonacci numbers?

Question 2: What is an optimal Huffman tree and Huffman code for the following set of frequencies?

M1: .45 M2: .18 M3: .002 M4: .11 M5: .24

Decode the encoding: 11101100110110111110 0

Question 3: What is an optimal Huffman tree and Huffman code for the following set of frequencies? Find out average number of bit required per character.

A:15 B:25 C:5 D:7 E:10 F:13 G:9

1.8 SUMMARY

- A typically optimization problem is solved by applying Greedy approach.
- An optimization problem is a problem in which the best (or optimal) solution among all the possible solutions of a problem is needed which maximizes/minimizes the objective function under some constraints or conditions.
- For an optimization problem, the input domain defines the candidate set and the subset of candidate set which satisfy all the constraints is termed as feasible set. The feasible solution which is best in terms of maximizing/minimizing the given objective function is called as optimal solution.
- Local optimal solutions correspond to the set of all the feasible solutions that are best locally for an optimization problem. Global optimal solution corresponds to the best solution of the optimization problem.
- Greedy approach does not guarantee to return global optimal solution, but it does on many problems.
- In Fractional Knapsack problem, a Knapsack (or bag) of some capacity is to be filled with objects which can be included in fractions. Each object is given with a weight and associated profit. The problem is to fill the given Knapsack with objects such that the sum of the profit associated with the objects that are included in the Knapsack is maximum. The constraint in this problem is that the sum of the weight of the objects in the Knapsack should be less than or equal to the capacity of the Knapsack. Greedy approach is a good approach to attain global solution for this problem.
- A task scheduling problem is an optimization problem in which a set of tasks is to be determined from the given tasks. Each task takes one unit of time for completion and has some associated profit. However, there is a deadline to complete every task. The objective is to accomplish a set of tasks within their deadlines such that the profit is maximum.
- Huffman coding is a greedy algorithm that is used to compress the data. As data is transmitted through transmission media, data is represented in the form of bits. If there is more number of bits to represent the data,

it will take more time to transmit. Thus, apply Huffman compression algorithm to compress the data in form, termed as Huffman codes.

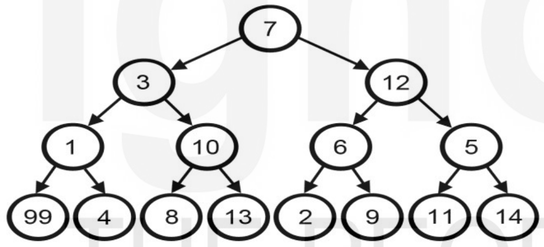
1.9 ANSWER TO CHECK YOUR PROGRESS

☞ **Check Your Progress-1**

Question 1: What is the main step of greedy approach that effects the obtained solution?
Solution: Selection criteria.

Question 2: In the graph below, a greedy algorithm is trying to find the longest path through the graph (the number inside each node contributes to a total length). To do this, it selects the largest number at each step of the algorithm. With a quick visual inspection of the graph, it is clear that this algorithm will not arrive at the correct solution. What is the correct solution? Why is a greedy algorithm ill-suited for this problem?

1



Solution: The correct solution for the longest path through the graph is 7, 3, 1, 99. This is clear because we can see that no other combination of nodes will come close to a sum of 99, so whatever path we choose, we know it should have 99 in the path. There is only one option that includes 99: 7, 3, 1, 99. However, greedy algorithm fails to solve this problem because it makes decisions purely based on what the best answer at the time is: at each step it *did* choose the largest number. However, since there could be some huge number that the algorithm hasn't seen yet, it could end up selecting a path that does not include the huge number. The solutions to the sub problems for finding the largest sum or longest path do not necessarily appear in the solution to the total problem. The optimal substructure and greedy choice properties don't hold in this type of problem.

Check Your progress-2

Question 1: A thief enters a house for robbing it. He can carry a maximal weight of 60 kg into his bag. There are 5 items in the house with the following weights and values. What items should thief take if he can even take the fraction of any item with him?

Item	Weight	Value
1	5	30
2	10	40
3	15	45
4	22	77
5	25	90

Solution:

Step 1: Compute the value / weight ratio for each item-

Items	Weight	Value	Ratio
1	5	30	6
2	10	40	4
3	15	45	3
4	22	77	3.5
5	25	90	3.6

Step 2: Sort all the items in decreasing order of their value / weight ratio-

I1	**I2**	**I5**	**I4**	**I3**
(6)	(4)	(3.6)	(3.5)	(3)

Step 3: Start filling the knapsack by putting the items into it one by one.

Knapsack Weight	**Items in Knapsack**	**Cost**
60	Ø	0
55	I1	30
45	I1, I2	70
20	I1, I2, I5	160

Now,

- Knapsack weight left to be filled is 20 kg but item-4 has a weight of 22 kg.
- Since in fractional knapsack problem, even the fraction of any item can be taken.
- So, knapsack will contain the following items-

  < I1 , I2 , I5 , (20/22) I4 >

Total cost of the knapsack = 160 + (20/27) x 77 = 160 + 57 = 217 units

Question 2: Let us consider that the capacity of the knapsack W = 60 and the list of provided items are shown in the following table −

Item	**A**	**B**	**C**	**D**
Profit	280	100	120	120
Weight	40	10	20	24
Ratio (p_i/w_i)	7	10	6	5

Find the optimal solution for gaining maximum profit.

Solution: After sorting all the items according to p_i/w_i. First all of ***B*** is chosen as weight of ***B*** is less than the capacity of the knapsack. Next, item ***A*** is chosen, as the available capacity of the knapsack is greater than the weight of ***A***. Now, ***C*** is chosen as the next item. However, the whole item cannot be chosen as the remaining capacity of the knapsack is less than the weight of ***C***.

Hence, fraction of ***C*** (i.e. (60 − 50)/20) is chosen.

Now, the capacity of the Knapsack is equal to the selected items. Hence, no more item can be selected.

The total weight of the selected items is **10 + 40 + 20 * (10/20) = 60**

And the total profit is **100 + 280 + 120 * (10/20) = 380 + 60 = 440**

This is the optimal solution. We cannot gain more profit selecting any different combination of items.

Check Your Progress-3

Question 1: What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers?

a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

Can you generalize your answer to find the optimal code when the frequencies, based on the first *n* Fibonacci numbers?

Solution: a:1111110; b:1111111; c:111110; d:11110; e:1110; f:110; g:10; h:0.

Yes, we can generalize for *n* numbers.

Question 2: What is an optimal Huffman tree and Huffman code for the following set of frequencies?

M1: .45 M2: .18 M3: .002 M4: .11 M5: .24

Decode the encoding: 1110110011011011111100

Solution: M2M1M3M4M5M2M3

Question 3: What is an optimal Huffman tree and Huffman code for the following set of frequencies? Find out average number of bits required per character.

A:15 B:25 C:5 D:7 E:10 F:13 G:9

Solution: 2.67bits/char

1.10 FURTHER READINGS

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein , Introduction to Algorithms, MIT Press, 3rd Edition, 2009.