2. Greedy Method

* **General Method:** In Greedy Method, the problem is solved based on the information available. the Greedy Method is straight forward method. for obtaining optimal solution.

Optimal Solution: from a set of feasible solutions, a feasible solution that satisfies the objective function is called as Optimal Solution.

Objective Function: A function which is used to determine a better solution is called as Objective function.

Feasible Solution: The subset of n inputs which satisfies some constraints are called as feasible solutions.

For example, What is the max even number b/w 1 to 50?
Here inputs are 1, 2, 3. . . . 50.
Constraint is even number

∴ Feasible Solutions are: 2, 4, 6, 8. . . . 50
Objective function is max even number
Optimal Solution is 50.

In Greedy method,

1. for every input a solution is obtained.
2. Then Feasibility of solution is performed.
3. for each set of feasible solutions the solution which satisfy the given objective function is obtained.
4. Such solution is called optimal Solution.

⊛ Algorithm for Greedy method:

```
Algorithm Greedy (a,n)
// a[1:n] contains the n inputs.
{
    solution := ϕ;
    for i:=1 to n do
    {
        x := Select(a);
        if Feasible(solution, x) then
            solution := Union(solution, x);
    }
    return solution;
}
```

⊛ Difference between Divide and Conquer and Greedy method.

Divide and Conquer:

① Divide and Conquer is used to obtain a solution to given problem.

② In this technique, the problem is divided into small subproblems. these subproblems are solved independently. Finally, all the solutions of subproblems are collected together to get the solution to the given problem.

③ In this method, duplications in subsolutions are neglected.

④ Divide and Conquer is less efficient.

⑤ Examples are Quick Sort, Binary Search..etc.

Greedy Method:

① Greedy Method is used to obtain Optimal Solution.

② In Greedy Method, a set of feasible is generated and optimum solution is obtained.

(3) In Greedy Method, the optimal solution is obtained without revising previous solutions.

(4). Greedy Method is compaxitively efficient

(5). Examples are : 0/1 Knapsack Problem, Minimum Spanning Tree.

Applications of Greedy Method :

(1) 0/1 Knapack Problem : Suppose there are n objects from $i = 1, 2, 3 \cdots n$. Each object i has some positive weight w_i and profit p_i. Consider a Knapsack (or) Bag with capacity m. Place these objects into the knapsack such that weight of all objects in the knapsack must be less than or equal to capacity of the knapsack 'm'.

The objective is to obtain maximized $\sum_{1 \le i \le n} p_i x_i$ subject to $\sum_{1 \le i \le n} w_i x_i \le m$ and $0 \le x_i \le 1, 1 \le i \le n$. this can be solved based on the following three strategies.

(1). Greedy about Profit

(2) Greedy about Weight

(3) Greedy about Profit per Unit Weight

(1) Greedy about Profit :

(a) In this strategy, choose the objects having more profit

(b). Total weight of the objects in the knapsack must be less than or equal to m.

(2) Greedy about Weight :

(a) In this strategy, choose the objects having less weigh

(b) Total weight of the objects in the knapsack must be less than or equal to m.

③ Greedy about Profit per Unit Weight:

ⓐ In this strategy, choose the objects having maximum profit per unit weight.

ⓑ Total weight of the objects in the knapsack must be less than or equal to m.

Example: Find an optimal solution to the knapsack instance $n=3, m=20, (P_1, P_2, P_3) = (30, 21, 18), (w_1, w_2, w_3) = (18, 15, 10)$. using Greedy method.

Sol:

	x_1	x_2	x_3	$\sum_{1 \le i \le n} P_i x_i$
case 1 :	1	2/15	0	32.8
case 2 :	0	2/3	1	32
case 3 :	5/9	0	1	34.6

Case 1 : Greedy about Profit : Choose the objects having more profit. Since first object has more profit, place it into the knapsack (i.e $x_1 = 1$). Since weight of the ~~knapsack~~ first object is 18, after placing first object into the bag remaining capacity of the bag is $m = 20 - 18 = 2$.

Now, identify the object having more profit among the remaining objects. Since 2nd object has more profit with weight 15. Since remaining capacity of the bag is 2, place fraction of second object. i.e $x_2 = 2/15$. After placing fraction of 2nd object into the bag remaining capacity of the bag is $m = 2 - 15 \times \frac{2}{15} = 0$.

Hence $x_i = 0$ for all remaining objects

$\therefore x_3 = 0$.

Now compute $\sum_{i=1}^{n} P_i x_i = P_1x_1 + P_2x_2 + P_3x_3$

$= 30 \times 1 + 21 \times \frac{2}{15} + 18 \times 0$

$= 32.8.$

Case 2: Greedy about Weight: Choose the object having less weight. Since 3rd. object has less weight, place it into the bag i.e $(x_3 = 1)$. After placing 3rd object into the knapsack remaining capacity of the knapsack is $m = 20 - 10 = 10$.

Now, identify the objects having less weight among the remaining objects. Since 2nd object has less weight ~~but~~ place it into the knapsack. Here, weight of the knapsack is 10 and weight of the object is 10. Hence place a fraction of 2nd object i.e $x_2 = 10/15 = 2/3$. After placing fraction of 2nd object into the knapsack, remaining capacity is $m = 10 - 15 \times \frac{10}{15} = 0$.

Hence $x_i = 0$ for all remaining objects i.e $x_1 = 0$.

Now Compute $\sum_{i=1}^{n} P_i x_i = P_1x_1 + P_2x_2 + P_3x_3$

$= 30 \times 0 + 21 \times \frac{2}{3} + 18 \times 1$

$= 0 + 14 + 18 = 32.$

Case 3: Greedy about profit per unit weight: Here, first compute

$\frac{P_1}{w_1} = \frac{30}{18} = 1.66, \frac{P_2}{w_2} = \frac{21}{15} = 1.4, \frac{P_3}{w_3} = \frac{18}{10} = 1.8.$

Choose the object having maximum profit per Unit weight. Since 3rd object has maximum profit per unit weight, place it into the knapsack i.e $x_3 = 1$. After placing 1st object into the knapsack, remaining capacity of the knapsack is $m = 20 - 10 = 10$.

Now, identify the object having maximum profit per unit weight among the remaining objects. First object has maximum profit per unit weight. Since weight of the 1st object is greater than weight of the knapsack fraction of 1st object is placed into the knapsack i.e $x_1 = \frac{10}{18} = 5/9$. After placing fraction of 1st object into the knapsack, remaining capacity is

$$m = 10 - 18 \times \frac{10}{18} = 0.$$

Hence $x_i = 0$ for all remaining objects

$\therefore x_2 = 0.$

Now, Compute $\sum_{i=1}^{n} P_i x_i = P_1 x_1 + P_2 x_2 + P_3 x_3$

$$= 30 \times \frac{5}{9} + 21 \times 0 + 18 \times 1$$

$$= 34.6$$

Among these three cases third case gives more profit which is 34.6. Hence optimal solution is

$$(x_1, x_2, x_3) = (5/9, 0, 1)$$

Algorithm for Greedy Knapsack:

```
Algorithm GreedyKnapsack(m,n)
// P[1:n] and w[1:n] contain the profits and weights respectively
// of the n objects. m is knapsack size and x[1:n] is
// solution vector
{
    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];
}
```

Note: Time Complexity is $O(n \log n)$

② **Job Sequencing With Deadlines:** Consider that there are n jobs to be executed from $i=1, 2, 3 \ldots n$. Each job i has some profit $P_i > 0$ and deadline $d_i \geq 0$. These profits are gained by corresponding jobs for obtaining feasible solution the job must completed within their given deadlines. the following are the rules to obtain the feasible solution.

① Each job takes one unit of time.

② If job starts before or its deadline, profit is obtained, otherwise no profit

③ Goal is to schedule jobs to maximize total profit

④ Consider all possible schedules and compute the minimum total time in the system.

* Optimal Storage on Tapes :

There are n programs that are to be stored on a computer tape of length l. Associated with each program i is a length l_i, $1 \le i \le n$.

Assume that whenever a program is to be retrieved from this tape, the tape is initially positioned at the front. Hence, if the programs are stored in the order $I = i_1, i_2, \cdots i_n$, the time t_j needed to retrieve program i_j is propotional to $\sum_{1 \le K \le j} l_{iK}$. If all programs are retrieved equally, then the expected Mean Retrieval Time is

$$MRT = \sum_{1 \le j \le n} t_j / n .$$

In the Optimal Storage on tape problem, we are required to find a permutation for the 'n' programs so that when they are stored on the tape in this order MRT is minimized.

The Greedy approach simply requires use to store the programs in nondecreasing order of their lengths. this Ordering can be carried out in $O(n \log n)$ time using an efficient sorting algorithm.

Example: Find an optimal placement for $n=3$, $(l_1, l_2, l_3) = (5, 10, 3)$.

Sol: There are $n! = 3! = 6$ possible ordering the Ordering and their respected values are

Ordering	Total Retrieval Time	MRT
1 2 3	$5+(5+10)+(5+10+3) = 38$	38/3
1 3 2	$5+(5+3)+(5+10+3) = 31$	31/3
2 1 3	$10+(10+5)+(10+5+3) = 43$	43/3
2 3 1	$10+(10+3)+(10+3+5) = 41$	41/3
3 1 2	$3+(3+5)+(3+5+10) = 29$	29/3 ✓
3 2 1	$3+(3+10)+(3+10+5) = 34$	34/3.

Optimal Ordering is $(3, 1, 2)$

Note: the tape storage problem can be extended to several tapes. If there are $m>1$ tapes, $T_0, T_1 \cdots T_{m-1}$, then the programs are to be distributed over these tapes. For each tape a storage permutation is provided. If I_j is the storage permutation for the subset of programs on tape j, then $d(I_j)$ is total retrieval time on one tape. So, here

$$\text{Total Retrieval Time (TD)} = \sum_{0 \le j \le m-1} d(I_j).$$

The objective is to store the programs in such a way to minimize TD.

Example: Find an Optimal Placement for n=13, m=3, lengths are 12, 5, 8, 32, 7, 5, 18, 26, 4, 3, 11, 10 and 6. Find out the total retrieval time

Sol: G.T Total no: of programs are 13

No: of tapes m=3. (T_0, T_1 and T_2).

Initially, Arrange the programs in ascending order based on their lengths.

3, 4, 5, 5, 6, 7, 8, 10, 11, 12, 18, 26, 32.

T_0 : 3 5 8 12 32

T_1 : 4 6 10 18

T_2 : 5 7 11 26

Total Retrieval Time for T_0 tape = $3+(3+5)+(3+5+8)+(3+5+8+12)+(3+5+8+12+32) = 115$

TD for T_1 tape = $4+(4+6)+(4+6+10)+(4+6+10+18)$

$= 72$

TD for T_2 tape = $5+(5+7)+(5+7+11)+(5+7+11+26)$

$= 89.$

Final total Retrieval time for three tapes i.e

T_0, T_1 and T_2 = 115+72+89

= 276.

Algorithm

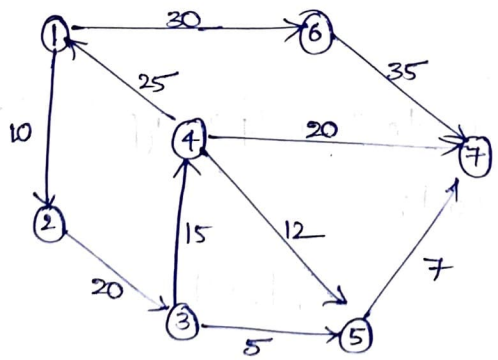
```
Algorithm Store (n,m)
{
   j:=0;
   for i:= 1 to n step do
    {
       write ("append program", i, "to permutation
              for tape", j);
      j:= (j+1) mod m;
    }
} //
```

④ Single Source Shortest Path Problem: Graphs can be used to represent the highway structure of a state or country with vertices representing cities and edges representing sections of highway. A motorist wishing to drive from city A to city B would be interested in answers to the following questions:

- Is there a path from A to B?
- If there is more than one path from A to B, which is the shortest path?

Let G(V, E) be a directed or undirected graph. In Single Source shortest path, the shortest path from vertex V_0 to all the remaining vertex is determined. the vertex V_0 is called as source vertex and the last vertex is called as destination vertex. the length of the path is defined to be the sum of weights of the edges on that path.

For example, Consider a graph G given below.



Source vertex	Destination vertex	path	Distance
1	2	1→2	10.
1	3	1→2→3	30
1	4	1→2→3→4	45
1	5	1→2→3→5	35
1	6	1→6	30
1	7	1→2→3→5→7	42.

(*) **Algorithm for Single Source Shortest Problem:**

```
Algorithm ShortestPaths (v, dist, cost, n)
// dist[j] is set to the length of the shortest path from
// vertex v to vertex j in a digraph G with n vertices.
// dist[v] is set to 0.
{
   for i:= 1 to n do
   {
      s[i]:= false; dist[i]:= cost[v,i];
   }
   s[v]:= true; dist[v]:=0.0;
   for num:=2 to n do
   {
      choose u from among those vertices not in S
      such that dist[u] is minimum;
      s[u]:= true;
      for (each w adjacent to u with s[w]=false) do
         if (dist[w] > dist[u] + cost[u,w]) then
            dist[w]:= dist[u] + cost[u,w];
   }
}//
```

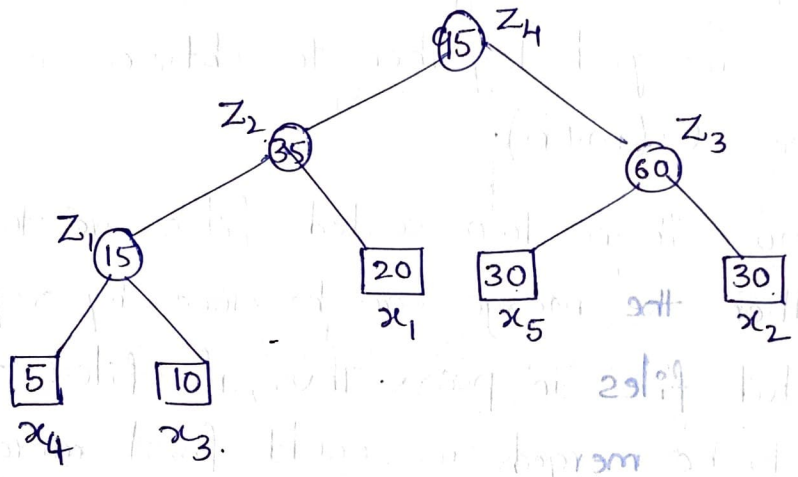
Note: Running time is $O(E \log V)$

Optimal Merge Patterns: Optimal Merge pattern is a pattern that relates to the merging of two or more sorted files in a single sorted file. this type of merging can be done by two-way merging method. If we have two sorted files containing m and n records then they could be merged together to obtain one sorted file in time $O(m+n)$.

When more than two sorted files are to be merged together the merge can be done by repeatedly merging sorted files in pairs. Thus, if files x_1, x_2, x_3 and x_4 are to be merged we could first merge x_1 and x_2 to get a file y_1. then merge y_1 and x_3 to get y_2. Finally merge y_2 and x_4 to get desired sorted file. Alternatively we could first merge x_1 and x_2 getting y_1, then merge x_3 and x_4 getting y_2 and finally y_1 and y_2 getting the desired sorted file. Different pairings require different amounts of computing time.

for eample, x_1, x_2 and x_3 are three sorted files of length 30, 20 and 10 records each. Merging x_1 and x_2 need 50 record moves. Merging the result with x_3 need another 60 moves. the total number of record moves required to merge the three files this way is 110. Instead, if we first merge x_2 and x_3 and then x_1, the total record moves made is only 90.

A greedy attempt to obtain optimal merge pattern is, at each stage merge the two smallest files together. If we have five files $x_1, x_2 \cdots x_5$ with sizes 20, 30, 10, 5, 30 our greedy rule would generate the following merge pattern



Binary Merge Tree Representing a Merge Pattern

Here, first merge x_4 and x_3 to get z_1 ($|z_1|=15$); merge z_1 and x_1 to get z_2 ($|z_2|=35$); merge x_5 and x_2 to get z_3 ($|z_3|=60$); Finally merge z_2 and z_3 to get z_4 ($|z_4|=95$). the total number of record moves is $15+35+60+95=205$.

In the above tree leaf nodes are drawn as squares and represent the given 5 files. these nodes are called as external nodes. the remaining nodes are drawn circular and are called internal nodes. Each internal node has exactly two children and it represents the file obtained by merging the files represented by its two children. the number in each node is the length (i.e no: of records) of the file represented by that node.

the external node x_4 is at a distance of 3 from the root node Z_4. Hence, the records of file x_4 will be moved three times, once to get Z_1, once again to get Z_2 and finally one more time to get Z_4. If d_i is the distance from the root to the external node for file x_i and q_i is length of x_i then the total number of record moves for this binary merge tree is

$$\sum_{i=1}^{n} d_i q_i$$

This sum is called the weighted external path length of the tree.

Algorithm:

```
Algorithm Tree (L, n)
//L is a list of n single node Binary tree .
{
   for i := 1 to n-1 do
    {
        GETNODE(T);
        LCHILD (T) ← LEAST (L);
        RCHILD (T) ← LEAST (L);
        WEIGHT (T) ← WEIGHT (LCHILD(T)) + WEIGHT (RCHILD(T))
        INSERT (L,T)
    }
   return  LEAST(L)
}//.
```

the algorithm has a list of L trees as input. Each node in a tree has three fields, LCHILD, RCHILD and WEIGHT. Initially, each tree in L has exactly one node. this node

is an external node and has LCHILD and RCHILD fields zero while the WEIGHT is the length of one of the n files to be merged. GETNODE(T) provides a new node for use in building the tree. LEAST(L) finds a tree in L whose root has least WEIGHT. this tree is removed from L. INSERT (L,T) inserts the tree with root T into the list L.

Analysis: The main loop is executed n-1 times. If L is kept in non decreasing order according to the WEIGHT value in the roots, then the LEAST(L) requires only O(1) time and INSERT (L,T) can be done in O(n) time. Hence the total time taken is $O(n^2)$.

In case L is represented as a min heap where the root values is ≤ the values of its children, then LEAST(L) and INSERT (L,T) can be done in O(logn) time. In this case the computime is O(nlogn).

Frequently Asked Questions.

1. What is greedy method? Explain with example?
2. Give the control abstraction for Greedy Method?
3. What is a Knapsack Problem? Find an optimal solution to the knapsack instance $n=7$, $m=18$, $(P_1, P_2 \cdots P_7) = (15,5,6,7,4,10,1)$ and $(w_1, w_2 \cdots w_7) = (7,4,8,2,1,4,1)$.
4. Write an algorithm for Greedy knapsack?
5. Find an optimal solution to the knapsack instance $n=7$, $m=15$, $(P_1, P_2 \cdots P_7) = (18,15,5,7,16,8,13)$ and $(w_1, w_2 \cdots w_7) = (2,3,5,7,1,4,1)$.
6. Find an optimal solution to the knapsack instance $n=7, m=15$, $(P_1, P_2 \cdots P_7) = (10,5,15,7,6,18,3)$ and $(w_1, w_2 \cdots w_7) = (2,3,5,7,1,4,1)$.
7. Find an optimal solution to the following knapsack problem: $n=3, m=20$, $(P_1, P_2, P_3) = (25,24,15)$ and $(w_1, w_2, w_3) = (18,15,10)$.
8. Find an optimal solution to the following knapsack instance: $n=7, m=10$, $(P_1, P_2 \cdots P_7) = (7,18,6,7,5,3,4)$, $(w_1, w_2 \cdots w_7) = (1,4,3,2,3,6,7)$
9. ~~Explain in detail about Job Sequencing with Deadlines problem~~
10. ~~What is the solution generated by the function JS when $n=7$, $P[1:7] = (3,5,20,18,1,6,30)$ and $d[1:7] = (1,3,4,3,2,1,2)$.~~
11. ~~Let $n=5$, $P[1:5] = (10,3,33,11,40)$ and $d[1:5] = (2,1,1,2,2)$. Find the optimal solution using greedy method.~~
12. ~~Let $n=5$, $P[1:5] = (40,33,30,14,10)$ and $d[1:5] = (2,1,2,3,3)$. Find the optimal solution using greedy method.~~
13. ~~Let $n=5$, $P[1:5] = (20,13,10,4,1)$ and $d[1:5] = (2,1,2,3,3)$. Find the optimal solution using greedy method.~~
14. ~~Present a Greedy algorithm for Sequencing Unit time jobs with deadlines and profits.~~

15. What is single source shortest path problem? Explain with an example.

16. Give the greedy algorithm to generate Single Source shortest paths?

17. A motorist wishing to ride from city A to B. Formulate greedy based algorithms to generate shortest path and explain with an example graph.

18. Explain the process of merging sorted files with an example?

19. Write an algorithm for optimal merge patterns?

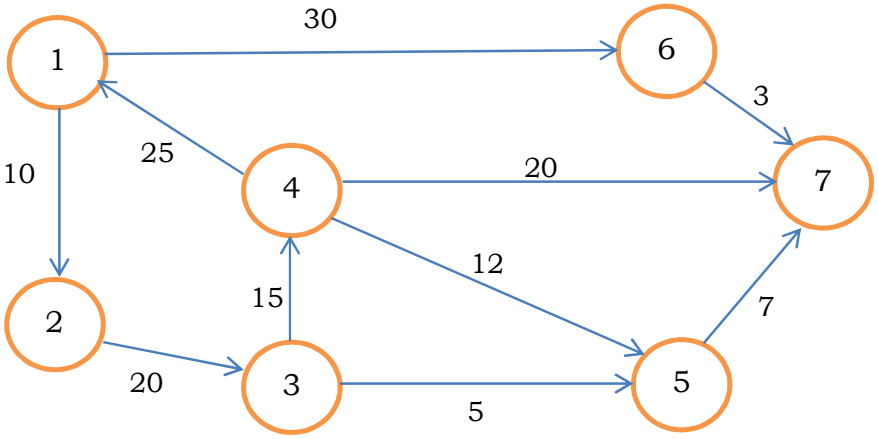
20. Find an optimal binary merge pattern for files whose lengths are

a. 20, 30, 10, 5, 30

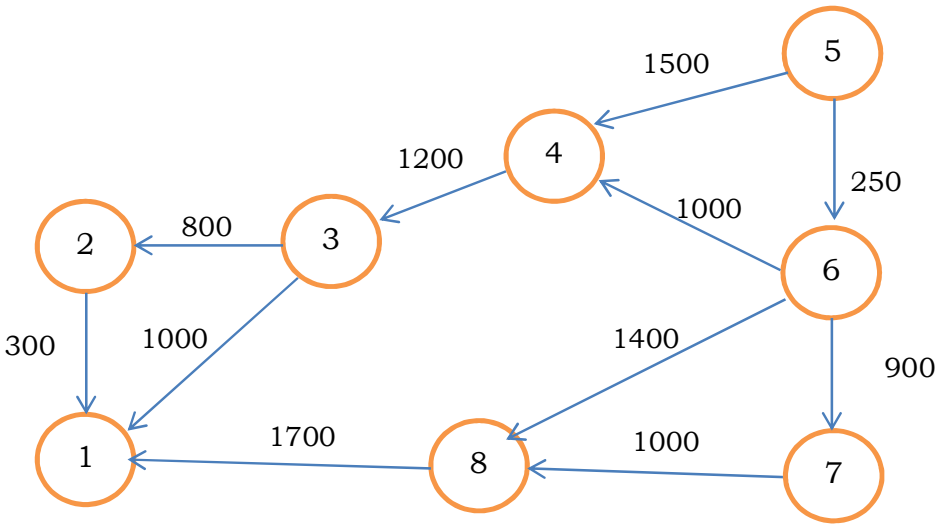
b. 28, 32, 12, 5, 84, 53, 91, 35, 3, and 11.

c. 2, 3, 5, 7, 9, 13

21. Apply Djkstra's algorithm to find shortest path from vertex 1.



22. Apply Djkstra's algorithm to find shortest path from vertex 5.



23. Apply Djkstra's algorithm to find shortest path from vertex A.