Greedy Strategy is an algorithmic approach in computer Science and optimization problems where the optimal choise is made at each step with the hope of finding a global optimum. it is called greedy because it always makes the locally optimal choice without considering the bigger picture or future consequences.

the decision making process involves selecting the best ~~availability~~ available option at each step without reconsidering the choices made previously. the algorithm assumes that choosing best option at each step will lead to the best overall solution. this assumption does not always hold true and the greedy strategy can sometimes lead to suboptimal or incorrect solutions.

the main advantage is the efficiency, Greedy algorithms often have a time complexity that is less than other algorithms such as dynamic programming or exhaustive search. They are relatively easy to implement and understand making them a popular choice for solving certain types of problems.

aray a[1..n] Store n input
Soln Set initialize as null
x is choosen with Selection criteria, if fesible Soln then added to Soln set.

Control Abstraction for Greedy Algorithm

```
Greedy_Algorithm (a,n)
{
    Solution = ϕ
    for i=1 to n do
    {
        x = Select (a)
        if Feasible (Solution, x) then
            Solution = Union (Solution, {x})
    }
} return Solution.
```

Application of greedy method

- Knapsack problem
- Minimum cost spanning tree
- Activity Selection problem.
- Huffman code
- Job Sequencing.
- TSP
- Graph coloring
- Single Source Shortest path.

Fractional knapsack problem.

It is a classical optimization problem. it involves selecting items from a set, each with a weight and a value to maximize the total value while ensuring that the total weight of the selected items does not exceed a given capacity.

greedy approach is used to solve the fractional knapsack problem. as it select items with highest value to weight ratio at each step. by considering fractional items the greedy approach can often provide an optimal solution.

object	1	2	3	4	5	6	7	
P	5	10	15	7	8	9	4	W = 15
W	1	3	5	4	1	3	2	n = 7
R	5	3.3	3	1.75	8	3	2	

Max profit

obj	P	W	Rem
3	15	5	15-5 = 10
2	10	3	10-3 = 7
6	9	3	7-3 = 4
5	8	1	4-1 = 3
4	$\frac{7 \times 3}{4}$ = 5.25	4(3)	3-3 = 0
	47.25		

Min W

obj	P	W	rem ~~obj~~
1	5	1	14
5	8	1	13
7	4	2	11
2	10	3	8
6	9	6	5
4	7	4	1
3	$\frac{15 \times 1}{5}$ = 3	$\frac{1}{5}$	0

Max P/W ratio

obj	P	W	rem
5	8	1	14
1	5	1	13
2	10	3	10
3	15	5	5
6	9	3	2
7	4	2	0
	51		

knapsack problem are categorized as

1. Fractional knapsack problem.
   - fractions of items can be taken.
   - Solved by greedy method.
2. 01 knapsack problem.
   - Items are indivisible. (either take an item or not)
   - can solved by dynamic programming.

Fractional knapsack problem - greedy algo gives optimal solution

Steps - ① Compute profit / weight ratio for each item
② Sort all the items in decreasing order of P/w ratio
③ Start filling the knapsack by putting items one by one.

Algorithm.

```
Greedy knapsack (m, n)
{
    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]
}
```

1 find P/w for each obj
2 Sort P/w in dec order
3 obj with highest P/w is selected first
4 Mark the obj with 1 if it's completely selected or the fract part if not selected completely.
5. when selected deduct the knapsack size by it particular obj size
6. Repeat 4 & 5
7. Note final fraction part and count that obj in the knapsack.
8. Find the total weight
Find total profit

Time Complexity is O(nlogn)

Spanning tree

Spanning tree of a graph (G) is a subset of G that covers all of its vertices using the minimum number of edges

properties of Spanning tree.

- a connected graph have more than one spanning tree
- if n nodes, Spanning tree has n-1 edges
- all Spanning trees of graph G have the same number of edges and vertices.
- Spanning tree does not have any cycle (loops)
- removing one edje from the spanning tree will make the graph disconnected
- adding one edge to the spanning tree will create circular loop.
- a complete graph can have maximum n^{n-2} no. of spanning tree.

Minimum Spanning tree

MST is a Spanning tree with minimum edge weight.

cost of Spanning tree = sum of ~~all~~ cost of its edges.

MST algos — kruskal's Algo

Prims Algo.

Kruskals Algorithm.

first Remove all loops & parallel edges

1. Sort all edges in increasing order of weight.
2. Pick smallest edge, check if it form a cycle
   if not, include the edge.
   else discard it
3. Repeat Step 2 until there is V-1 edges in the Spanning tree.

time Complexity. O(E log V)

Application of Spanning tree

Civil Network planning

Computer Network Routing Protocol

Cluster Analysis

Image Segmentation.

Handwriting recognition.

Prims algorithm.

1. determine a arbitary vertex as the starting vertex of the MST
2. Follow steps 3 to 5 till there are vertices that are not included in the MST (fringe vertex)
3. Find edges connecting any three vertex with the fringe vertices
4. Find the minimum among these edges.
5. Add the chosen edge to the MST if it does not form any cycle.
6. Return MST and exit.

Job Sequencing with Deadline.

it is another classical optimization problem it involves scheduling a set of jobs with associated profits and deadlines to maximize the total profit while meeting the given deadlines. greedy stategy is used to solve this problem.

Algo :-
- Find maximum deadline value from input set of jobs
- once deadline is decided arrage the jobs in decending order of their profits.
- Select the job with highest profit, their time period not exceeding the max deadline.
- the selected set of jobs are the output.

Q1

Jobs	J1	J2	J3	J4	J5
P	20	15	10	5	1
dL	2	2	1	3	3

n = 5.

Machine 1 u.

$0 \underset{20}{\overset{J_1}{\text{---}}} 1 \underset{15}{\overset{J_2}{\text{---}}} 2 \underset{5}{\overset{J_4}{\text{---}}} 3$

$J_1 \rightarrow J_2 \rightarrow J_4$ $\quad$ 20 + 15 + 5 = 40.

$J_2 \rightarrow J_1 \rightarrow J_4$

Jobs	J1	J2	J3	J4	J5
P	20	15	10	5	1
dL	2	2	1	3	3

Job Consider	Slot	Soln	Profit
J1	[1, 2]	J1	20
J2	[0, 1] [1, 2]	J1 J2	35
J3 ×	[0, 1] [1, 2]	J1 J2	35
J4	[0, 1] [1, 2] [2, 3]	J1 J2 J4	40
J5 ×	"	"	"

Q2.

Jobs	J1	J2	J3	J4	J5	J6	J7
P	35	30	25	20	15	12	5
DL	3	4	4	2	3	1	2

$0 \overset{J_4}{\text{---}} 1 \overset{J_3}{\text{---}} 2 \overset{J_1}{\text{---}} 3 \overset{J_2}{\text{---}} 4$

20 + 25 + 35 + 30 = 110

Dynamic Programming

it is a technique used to solve optimization problems by breaking them down into smaller overlapping subproblems. and Solving each subproblem only once. Soln to subproblems are stored and reused to avoid redundant computations, leading to more efficient algorithms. recursive solution that has repeated calls for same input we can optimize using dynamic programming. optimiztn reduces time com from exp to linear.

Principle of Optimality In Dynamic Programming.

the POO is a fundamental aspect of dynamic programming which states that the optimal Solution to a dynamic optimization problem can be found by combining the optimal solutions to its sub-problems.

```
int fib(int n)
{   if (n<=1)
        return n;
    else
        return fib(n-1)+fib(n-2)
}
```

```
f[0] = 0
f[1] = 1
for (i=2 ; i<n ; i++)
{
    f[i] = f[i-1] + f[i-2]
}
return f[i]
```

Travelling Saleman Problem.

classical optimization problem. it involves finding ~~best~~ shortest possible route that a salesman can take to visit a given set of cities exactly once and return to the starting city.

Common approaches to solve TSP } Bruteforce, Heuristic algorithms. DP, Approximation algorithms

0/1 Knapsack Problem

It is classical optimization problem, it involves selecting items from a given set, each with a weight and a value, to maximize the total value while ensuring that the total weight of the selected items does not exceed a given capacity.

P { 1, 2, 5, 6 } m = 8
w { 2, 3, 4, 5 } n = 4

			0	1	2	3	4	5	6	7	8
P	w	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3	3
3	4	3	0	0	1	2	5	5	6	7	7
4	5	4	0	0	1	2	5	6	6	7	8

$\left\{ \begin{matrix} x_1 & x_2 & x_3 & x_4 \\ 0 & 1 & 0 & 1 \end{matrix} \right\}$ 8-6 = 2

Longest Common Subsequence

Classical computational problem. Given two strings the LCS problem involves finding the longest subsequence that is common to both sequences. A subsequence is a sequence that can be derived by deleting zero or more elements from the original sequence without changing the order of the remaining elements

A [b|d] (1 2)   B [a|b|c|d] (1 2 3 4)

		0	1	2	3	4
	0	0	0	0	0	0
b	1	0	0	1	1	1
d	2	0	0	1	1	2
		0	a	b	c	d

[b|d]