Greedy Algorithm

Dr. Mervat Mekhaeil

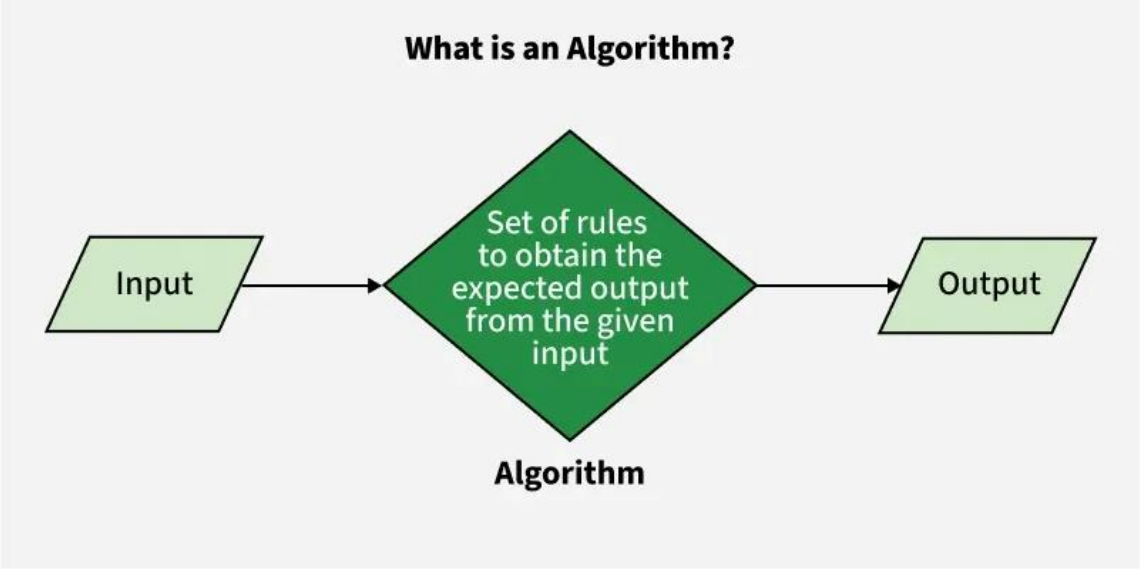
Eng. Ahmed Yahia

Members ?

Rodina Mohamed	22101313
Marc Hany	22100540
Aya Mamdouh	22101087
Raghad Abo Refaey	22101912
Mazen Ashraf	22100105

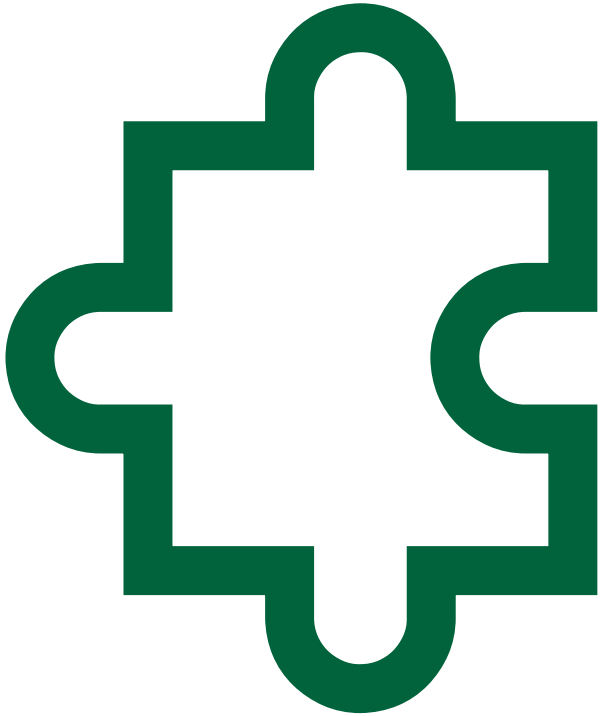
Algorithm ?

- *refers to a sequence of finite steps to solve a particular problem.*
- *Algorithms can be simple and complex depending on what you want to achieve.*



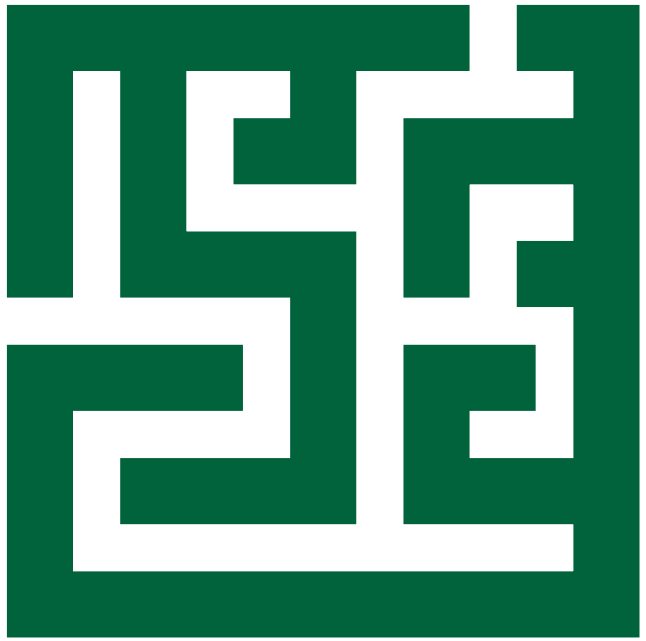
Greedy Algorithm ?

- *an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit.*



Characteristics of Greedy Algorithm ?

- *Greedy algorithms are simple and easy to implement.*
- *They are efficient in terms of time complexity, often providing quick solutions. Greedy Algorithms are typically preferred over Dynamic Programming for the problems where both are applied. For example, Jump Game problem and Single Source Shortest Path Problem (Dijkstra is preferred over Bellman Ford where we do not have negative weights).*
- *These algorithms do not reconsider previous choices, as they make decisions based on current information without looking ahead.*



How does the Greedy Algorithm work?

1. *Start with the initial state of the problem. This is the starting point from where you begin making choices.*
2. *Evaluate all possible choices you can make from the current state. Consider all the options available at that specific moment.*
3. *Choose the option that seems best at that moment, regardless of future consequences. This is the "greedy" part – you take the best option available now, even if it might not be the best in the long run.*
4. *Move to the new state based on your chosen option. This becomes your new starting point for the next iteration.*
5. *Repeat steps 2-4 until you reach the goal state or no further progress is possible. Keep making the best local choices until you reach the end of the problem or get stuck.*

Greedy Algorithm's Applications ?

1. ***Dijkstra's Algorithm***

   Finds the shortest path from a single source node to all other nodes in a graph with non-negative edge weights.

2. ***Huffman Coding***

   Used in data compression to build an optimal prefix-free binary code, minimizing the total encoded length.

3. ***Kruskal's/Prim's Algorithm***

   Constructs a ***Minimum Spanning Tree (MST)*** *in a graph, connecting all nodes with the least total edge weight.*

4. ***Activity Selection Problem***

   Selects the maximum number of non-overlapping activities (or intervals) that can be performed in a given time.

5. ***Coin Change Problem (Greedy Version)***

   Provides the minimum number of coins needed to make change when coin denominations are in a canonical system (e.g., 1, 5, 10, 25 cents).

Pseudocode ?

```
GreedyAlgorithm(problem):
    solution = empty_set
    candidates = initialize_candidates(problem)

    while candidates is not empty:
        best = select_best_candidate(candidates)
        if is_feasible(solution, best):
            solution = solution U {best}
        remove best from candidates

    return solution
```

initialize_candidates(problem):
Create the list of available choices.
select_best_candidate(candidates):
Use a greedy criterion to pick the best.
is_feasible(solution, best):
Check if adding this candidate keeps the solution valid.
solution:
Gradually constructed final answer.

Mathematical Foundations of Greedy Algorithms ?

For a problem to be solved correctly by a greedy approach, it must exhibit:

1. Greedy Choice Property

This means a global optimum can be reached by choosing the optimal choice at each step.

Formally:
Let ***S*** *be the solution space. A problem exhibits the greedy-choice property if making a greedy choice* $g \in S$ *at a stage leads to an optimal solution that includes* ***g***.

2. Optimal Substructure

A problem has this property if an optimal solution to the problem contains optimal solutions to its subproblems.

Formally:
If ***OPT(n)*** *is the optimal solution for input size* ***n****, and*
OPT(n) = OPT(k) + greedy step*, for some* $k < n$*, then the problem has optimal substructure.*

General Time & Space Complexity of Greedy Algorithms ?

Time Complexity

Let ***n*** be the input size (number of elements, activities, nodes, etc.).

A greedy algorithm typically involves two main phases:

Sorting or Prioritization

Time: ***O(nlogn)***, if sorting is required (e.g., by weight, deadline, cost).

Linear Selection Loop

Time: O(n), iterating through elements and applying the greedy condition.

General Time Complexity:

T(n)=O(nlogn)

(if sorting is required), otherwise:

T(n)=O(n)

Space Complexity

Let ***n*** be the input size (number of elements, activities, nodes, etc.).

Depends on implementation:

In-place operations → ***O(1)*** extra space.

Storing results or temporary structures → ***O(n)***

General Space Complexity:

S(n)=O(n)

Comparison with alternatives ?

Criteria	Greedy	Dynamic Programming	Divide & Conquer	Brute Force
Guarantees Optimal	(not always)	✅	✅ (in many)	✅
Time Efficiency	High	Moderate	Moderate	Low
Space Usage	Low	High	Moderate	High
Problem Size	Large	Medium	Medium	Small
Recursion Used	Rarely	Often	Always	(unless recursive BF)