GREEDY ALGORITHM

- DESCRIPTION OF THE GREEDY ALGORITHM
- WHAT IS A GREEDY ALGORITHM?
- STEPS FOR CREATING A GREEDY ALGORITHM
- ADVANTAGES AND DISADVANTAGES
- LIMITATIONS OF GREEDY ALGORITHM
- APPLICATIONS OF GREEDY ALGORITHM
- PRACTICE: STONES DISTRIBUTION PROBLEM

Author: prof. Yevhenii Borodavka

DESCRIPTION OF THE GREEDY ALGORITHM

A Greedy algorithm is an approach to solving a problem that selects the most appropriate option based on the current situation. This algorithm ignores the fact that the current best result may not bring about the overall optimal result. Even if the initial decision was incorrect, the algorithm never reverses it.

This simple, intuitive algorithm can be applied to solve any optimization problem which requires the maximum or minimum optimum result. The best thing about this algorithm is that it is easy to understand and implement.

WHAT IS A GREEDY ALGORITHM?

The runtime complexity associated with a greedy solution is pretty reasonable. However, you can implement a greedy solution only if the problem statement follows two properties mentioned below:

- Greedy Choice Property: Choosing the best option at each phase can lead to a global (overall) optimal solution.
- Optimal Substructure: If an optimal solution to the complete problem contains the optimal solutions to the subproblems, the problem has an optimal substructure.

STEPS FOR CREATING A GREEDY ALGORITHM

By following the steps given below, you will be able to formulate a greedy solution for the given problem statement:

- Step 1: In a given problem, find the best substructure or subproblem.
- Step 2: Determine what the solution will include (e.g., largest sum, shortest path).
- Step 3: Create an iterative process for going over all subproblems and creating an optimum solution.

ADVANTAGES AND DISADVANTAGES

Advantages of Greedy Approach:

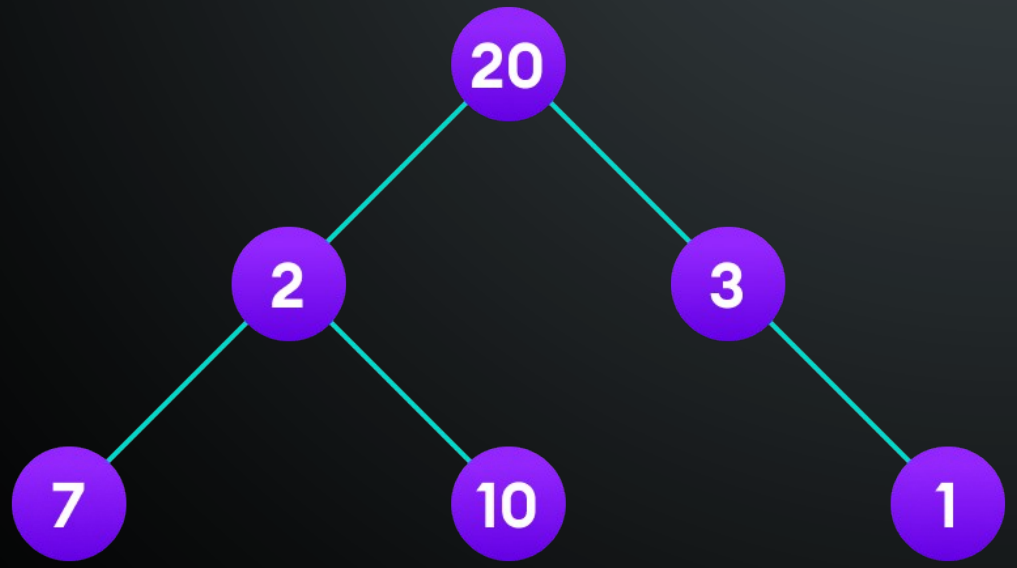
- The algorithm is easier to describe.
- This algorithm can perform better than other algorithms (but, not in all cases).

Drawback of Greedy Approach:

- As mentioned earlier, the greedy algorithm doesn't always produce the optimal solution. This is the major disadvantage of the algorithm.

ADVANTAGES AND DISADVANTAGES

For example, suppose we want to find the longest path in the graph below from root to leaf. Let's use the greedy algorithm here.



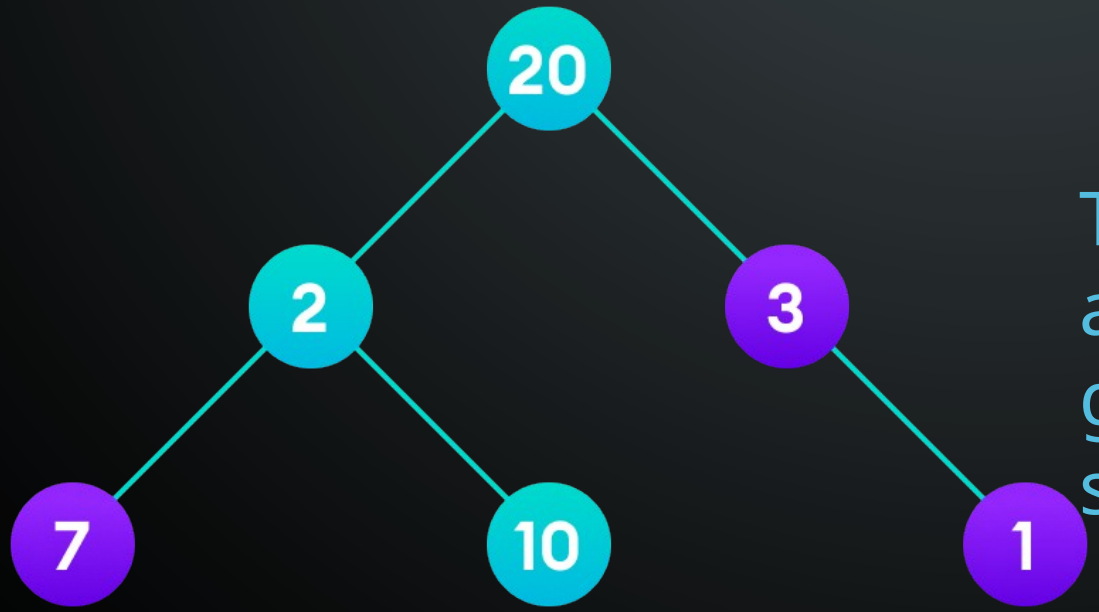
1. Let's start with the root node 20. The weight of the right child is 3 and the weight of the left child is 2.

2. Our problem is to find the largest path. And, the optimal solution at the moment is 3. So, the greedy algorithm will choose 3.

3. Finally the weight of an only child of 3 is 1. This gives us our

ADVANTAGES AND DISADVANTAGES

However, it is not the optimal solution. There is another path that carries more weight (20 + 2 + 10 = 32) as shown in the image below.



Therefore, greedy algorithms do not always give an optimal/feasible solution.

LIMITATIONS OF GREEDY ALGORITHM

Factors listed below are the limitations of a greedy algorithm:

1. The greedy algorithm makes judgments based on the information at each iteration without considering the broader problem; hence it does not produce the best answer for every problem.
2. The problematic part for a greedy algorithm is analyzing its accuracy. Even with the proper solution, it is difficult to demonstrate why it is accurate.
3. Optimization problems (Dijkstra's Algorithm) with negative graph edges cannot be solved using a greedy algorithm.

APPLICATIONS OF GREEDY ALGORITHM

Following are few applications of the greedy algorithm:

- Used for Constructing Minimum Spanning Trees: Prim's and Kruskal's Algorithms used to construct minimum spanning trees are greedy algorithms.
- Used to Implement Huffman Encoding: A greedy algorithm is utilized to build a Huffman tree that compresses a given image, spreadsheet, or video into a lossless compressed file.
- Used to Solve Optimization Problems: Graph - Map Coloring, Graph - Vertex Cover, Knapsack Problem, Job Scheduling Problem, and activity selection problem are solved using a greedy algorithmic paradigm.

PRACTICE: STONES DISTRIBUTION PROBLEM

Problem. You have **N** stones (10<=**N**<=100) and you need to distribute them into **P** stacks (**P**<=**N**) in the way to achieve the maximum result of the multiplication stones count in each stack.

Example. If **N**=10 and **P**=3 we can distribute stones in this way: 10=1+1+8. In this case, the multiplication result will be 1*1*8=8. But the optimal distribution will be 10=3+3+4. In this case, the multiplication result will be 3*3*4=36.

Task. Create a program using C/C++/Python to solve this problem.

Input. A single string with two numbers **N** and **P** divided by space.

PRACTICE: STONES DISTRIBUTION PROBLEM

Try to use the greedy approach to solve this problem.

The maximal result will be obtained in case of regular stones distribution. So, we can formulate greedy algorithm approach to get the maximal result.

- Step 1: the best subproblem is to distribute one stone between **P** stacks.
- Step 2: solution include **P** stacks with maximal amount of stones.
- Step 3: we need to add one stone per iteration to the stack with smallest amount of stones. Thus we have to do **N** iterations.

PRACTICE: STONES DISTRIBUTION PROBLEM

Step 1. The first stone distribution between **P**=3 stacks.

Stack 1	Stack 2	Stack 3
1	0	0

PRACTICE: STONES DISTRIBUTION PROBLEM

Step 2. The second stone distribution between **P**=3 stacks.

Stack 1	Stack 2	Stack 3
1	1	0

PRACTICE: STONES DISTRIBUTION PROBLEM

Step 3. The third stone distribution between **P**=3 stacks.

Stack 1
1

Stack 2
1

Stack 3
1

PRACTICE: STONES DISTRIBUTION PROBLEM

Step 4. The fourth stone distribution between **P**=3 stacks.

Stack 1	Stack 2	Stack 3
2	1	1

PRACTICE: STONES DISTRIBUTION PROBLEM

Step 5. The fifth stone distribution between **P**=3 stacks.

Stack 1	Stack 2	Stack 3
2	2	1

PRACTICE: STONES DISTRIBUTION PROBLEM

Step 6. The sixth stone distribution between **P**=3 stacks.

Stack 1	Stack 2	Stack 3
2	2	2

PRACTICE: STONES DISTRIBUTION PROBLEM

Step 7. The seventh stone distribution between **P**=3 stacks.

Stack 1	Stack 2	Stack 3
3	2	2

PRACTICE: STONES DISTRIBUTION PROBLEM

Step 8. The eighth stone distribution between **P**=3 stacks.

Stack 1	Stack 2	Stack 3
3	3	2

PRACTICE: STONES DISTRIBUTION PROBLEM

Step 9. The ninth stone distribution between **P**=3 stacks.

Stack 1
3

Stack 2
3

Stack 3
3

PRACTICE: STONES DISTRIBUTION PROBLEM

Step 10. The tenth stone distribution between **P**=3 stacks.

Stack 1	Stack 2	Stack 3
4	3	3

PRACTICE: STONES DISTRIBUTION PROBLEM

After **N**=10 iterations we get the final result of the stones distribution. Now we only need to multiply the amount of stones in each stack to get the final answer to the problem.

Of course, this problem has a much smarter and faster solution. So, I hope, you can find it and implement it.

PRACTICE: STONES DISTRIBUTION PROBLEM

Code example:

```
#include <iostream>

int N, P, A;

int main()
{
    cin >> P >> N;

    // Your code here

    cout << A;

    return 0;
}
```

THANK YOU!