What Is Algorithm Design?

An algorithm design technique means a unique approach or mathematical method for creating algorithms and solving problems. While multiple algorithms can solve a problem, not all algorithms can solve it efficiently. Therefore, we must create algorithms using a suitable algorithm design method based on the nature of the problem. An algorithm created with the right design technique can solve the problem much more efficiently with respect to the computational power required.

Algorithm Design Techniques

* Greedy: Selecting each part of a solution only because it is immediately beneficial
* Backtracking: Solving all possible combinations then backtracking if the current solution doesn't look desirable
* Divide and conquer: Solving the problem by dividing it into sub-problems
* Brute Force: Finding all the possible solutions and trying each one-by-one
* Dynamic Programming: Solving problems with overlapping sub-problems

1. Greedy Algorithm

Greedy algorithms craft a solution piece by piece, and their selection criteria when selecting the next piece is that it should be instantly fruitful. Hence, the algorithm evaluates all the options at each step and chooses the best one at the moment. However, they aren't beneficial in all situations. A greedy algorithm solution isn't necessarily an overall optimal solution since it only goes from one best solution to the next. Additionally, there is no backtracking involved if it chooses the wrong option or step.

Example

Greedy algorithms are the best option for certain problems. A popular example of greedy algorithm is sending some information to the closest node in a network. Some other graph-based greedy algorithm examples are: Dijkstra's Algorithm Prim and Kruskal's Algorithm Huffman Coding Tree.

2. Backtracking

A backtracking algorithm finds all the possible combinations of a solution and evaluates if it isn't optimal. If it isn't, the algorithm backtracks and starts evaluating other solutions. Backtracking algorithms share a common approach with the brute force algorithm design technique. However, they are much faster than brute-force algorithms. There are different kinds of backtracking algorithms based on the kind of problems they solve:

Decision Problem – Find a feasible solution

Optimization Problem – Find the most optimal solution

Enumeration Problem – Find all feasible solutions

Example

Backtracking algorithms are the most optimal for problems where we may need to go back a few steps and make different decisions. For example, one of the most famous backtracking algorithm examples is the one for solving crossword puzzles. Similarly, the eight queen’s puzzle also requires going back if the current solution isn't the right one.

3. Divide and Conquer

A divide and conquer algorithm breaks down the complexity of its problem so it can solve smaller and easier sub-problems. It involves three major steps:

Divide – Divide the problem into multiple sub-problems of the same nature

Solve – Solve each resulting sub-problem

Combine – Combine the solutions to the sub-problems to get the solution to the starting problem

A divide and conquer algorithm handles each sub-problem separately. Such algorithms give the most optimal solution for problems like efficiently sorting a collection of elements. This is a simple approach, and easy to understand. There are many divide and conquer algorithm examples in the real world. For example, take the common problem of looking for a lost item in a huge space. It is easier to divide the space into smaller sections and search in each separately.

4. Brute Force

A brute force algorithm uses the most straightforward way of achieving a problem's solution: keep trying until you find the right one. One example of a brute force algorithm is having multiple keys and trying to open a lock. Such algorithms create all solutions from the input and try each to solve the problem. In principle, brute force and backtracking use the same approach. The only difference is that the latter backtracks if they find a solution unsuitable.

Example: Cracking the password of an application is a popular brute force algorithm example. Given that there are unlimited retries, the only way is to try every possible password combination until we find the right one. Another example is visiting multiple locations and finding the shortest routes. Such examples show that brute force algorithms rely on having plenty of computational power.

5. Dynamic Programming

Dynamic programming is a class of algorithms that solve problems that have overlapping sub-problems. Therefore, they are well-suited for problems where certain sub-problems get solved repeatedly. Hence, a dynamic programming algorithm optimizes the solution by storing the answers to sub-problems in an optimal structure and retrieving them when needed.

Example: The problem of generating a Fibonacci sequence is one of the popular dynamic programming algorithm examples. After all, we keep solving the sub-problems repeatedly. For example, if we found the 5th number, we must have found all the ones before. Therefore, they are handy for finding the 6th number.

def FibSequence(n):

fib = {}

# Calculating Fibonacci sequence

fib[0] = 0

fib[1] = 1

for i in range (2,n):

fib[i] = fib[i-1] + fib[i-2]

return fib

print(FibSequence(10))