

Algorithm Design Techniques



An Algorithm is a procedure to solve a particular problem in a finite number of steps for a finite-sized input.

Algorithm Design Technique

Algorithm design technique is a specific approach to create a design process for solving problems. These techniques can be applied to different computing problems. These general techniques provide guidelines for designing algorithms for new problems and to represent an useful collection of tools.

Classification of Algorithm Design Techniques

1. Implementation Method

- **Recursion or Iteration:** A recursive algorithm is an algorithm which calls itself again and again until a base condition is achieved whereas iterative algorithms use loops and/or data structures like stacks, queue to solve any problem. Every recursive solution can be implemented as an iterative solution and vice versa.
- **Exact or Approximate:** Algorithms that are capable of finding an optimal solution for any problem are known as the exact algorithm. For all those

problems, where it is not possible to find the most optimized solution, an approximation algorithm is used. Approximate algorithms are the type of algorithms that find the result as an average outcome of sub outcomes to a problem.

- **Serial or Parallel or Distributed Algorithms:** In serial algorithms, one instruction is executed at a time while parallel algorithms are those in which we divide the problem into subproblems and execute them on different processors. If parallel algorithms are distributed on different machines, then they are known as distributed algorithms.

2. Design Method

- **Greedy Method:** In the greedy method, at each step, a decision is made to choose the *local optimum*, without thinking about the future consequences.
- **The Divide and Conquer:** Divide and Conquer strategy involves dividing the problem into sub-problem, recursively solving them, and then recombining them for the final answer.
- **Dynamic Programming:** The approach of Dynamic programming is similar to divide and conquer. The difference is that whenever we have recursive function calls with the same result, instead of calling them again we try to store the result in a data structure in the form of a table and retrieve the results from the table. Thus, the overall time complexity is reduced. "Dynamic" means we dynamically decide, whether to call a function or retrieve values from the table.
- **Linear Programming:** In Linear Programming, there are inequalities in terms of inputs and maximizing or minimizing some linear functions of inputs.
- **Reduction(Transform and Conquer):** In this method, we solve a difficult problem by transforming it into a known problem for which we have an optimal solution. Basically, the goal is to find a reducing algorithm whose complexity is not dominated by the resulting reduced algorithms.
- **Backtracking:** This technique is very useful in solving combinatorial problems that have *a single unique solution*. Where we have to find the correct combination of steps that lead to fulfillment of the task. Such problems have multiple stages and there are multiple options at each

stage. This approach is based on exploring each available option at every stage one-by-one. While exploring an option if a point is reached that doesn't seem to lead to the solution, the program control backtracks one step, and starts exploring the next option. In this way, the program explores all possible course of actions and finds the route that leads to the solution.


- **Branch and Bound:** This technique is very useful in solving combinatorial optimization problem that have *multiple solutions* and we are interested in find the most optimum solution. In this approach, the entire solution space is represented in the form of a state space tree. As the program progresses each state combination is explored, and the previous solution is replaced by new one if it is not the optimal than the current solution.

3. Design Approached

- **Top-Down Approach:** Breaking down a complex problem into smaller, more manageable sub-problems and solving each sub-problem individually. Designing a system starting from the highest level of abstraction and moving towards the lower levels.
- **Bottom-up approach:** The bottom-up approach is also known as the reverse of top-down approaches. Building a system by starting with the individual components and gradually integrating them to form a larger system. Solving sub-problems first and then using the solutions to build up to a solution of a larger problem.

References

Algorithms Design Techniques - GeeksforGeeks

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive  <https://www.geeksforgeeks.org/algorithms-design-techniques/>



Design of Algorithms

In this module we will discuss the various algorithm design techniques. We will explain these techniques using some illustrative examples.

📖 <https://ebooks.inflibnet.ac.in/csp01/chapter/design-of-algorithms/>

✍️ Author → Serhat Kumas

<https://www.linkedin.com/in/serhatkumas/>

SerhatKumas - Overview

Computer engineering student who loves coding in different fields instead of focusing on a one specific area. -

SerhatKumas

🌐 <https://github.com/SerhatKumas>

