Competitive programming

Tags

Introduction

Competitive programming is the practice of solving algorithmic problems under time constraints, often in a competitive setting such as a programming contest. The goal is to write efficient and correct code that can solve a given problem within the time limits of the contest. Some common topics in competitive programming include:

- Data structures: arrays, linked lists, stacks, queues, trees, graphs, etc.
- Algorithms: sorting, searching, dynamic programming, greedy algorithms, etc.
- Number theory and combinatorics
- Game theory
- Graph algorithms: graph traversals, shortest paths, minimum spanning tree, etc.

- String algorithms: pattern matching, string compression, etc.
- Geometry
- Combinatorial optimization
- Advanced data structures like Trie, Segment Tree, Fenwick tree, etc.
- Computational geometry

Studying

There are several ways to study competitive programming, some of the most effective ways include:

1. Practicing problems: The best way to improve at competitive programming is to practice solving problems. There are many online platforms such as CodeForces, Leetcode, HackerRank, CodeChef, etc. where you can find a wide variety of problems to solve.
2. Participating in contests: Participating in online programming contests is a great way to get a feel for the time pressure and competition of a real contest. It will also help you to learn from other participants' solutions and approaches.
3. Reading books and tutorials: There are many books and tutorials available online that cover the topics and techniques used in competitive programming. Some popular books include "Introduction to Algorithms" by Cormen, "Algorithms Unlocked" by Thomas H. Cormen and "Competitive Programming" by Steven and Felix Halim
4. Learning from others: Join online communities or forums where competitive programmers share their tips and solutions. This can help you learn from more experienced programmers and stay up-to-date on the latest techniques and trends.
5. Implementing algorithms: It's important to understand the theory behind the algorithms but equally important to be able to implement them. Try to implement the algorithms you learn, it will help you to understand it better and also help with debugging your code.
6. Regularly revise the topics: Competitive programming requires a lot of practice and revision. Make sure to revise the concepts and the problems you solved regularly, it will help you to keep the concepts fresh in your mind.

Practice

1. Sorting and searching: questions that involve sorting and searching algorithms, such as finding the kth smallest element in an array or sorting a list of numbers in a specific order.
2. Dynamic programming: questions that involve breaking a problem down into smaller subproblems and storing solutions to those subproblems to avoid redundant computation.
3. Graph algorithms: questions that involve traversing or manipulating graph data structures, such as finding the shortest path between two nodes in a graph or determining the minimum spanning tree of a graph.
4. String algorithms: questions that involve manipulating strings, such as finding the longest common substring or checking if a string is a palindrome.
5. Combinatorial optimization: questions that involve finding the best solution among a large number of possibilities, such as the traveling salesman problem or the knapsack problem.
6. Number theory: questions that involve understanding of number properties or manipulation of numbers.
7. Divide and conquer: questions that involve breaking the problem into smaller subproblems and solve them independently.
8. Greedy algorithm: questions that involve making the locally optimal choice at each stage with the hope of finding the global optimum.
9. Backtracking: questions that involve finding all possible solutions by incrementally building candidate solutions and abandoning a candidate as soon as it is determined that the candidate cannot possibly be completed to a valid solution.
10. Bit manipulation: questions that involve manipulating the individual bits of a number, such as counting the number of set bits in a binary representation of a number.

Complete Roadmap for Beginners

1. Understand the Basics of Programming Languages:

The first step to start Competitive Programming is to have a clear understanding of a programming language. A few popular programming languages for competitive programming are C++, Python, and Java. Once you select your preferred language, start learning the syntax, structure, and basic constructs like loops, conditional statements, and functions.

2. Choose a Platform:

There are several online platforms available for Competitive Programming. Some of the popular ones include Codeforces, CodeChef, HackerRank, and LeetCode. Choose a platform based on your skill level, and join their community. These platforms have multiple rounds of contests, allowing you to participate in regular competitions.

3. Start with Simple Problems:

Once you understand the basics of programming languages, start solving simple problems. These problems are generally divided into categories based on difficulty levels, like easy, medium, and hard. Start with easy problems and gradually move on to harder ones as you gain experience.

4. Analyze Sample Solutions:

After solving a problem, analyze the sample solutions. Try to understand how they have approached the problem and solved it. This will give you an idea of how to approach similar problems in the future.

5. Participate in Contests:

Participating in contests is an excellent way to improve your skills. These contests have a set of problems with varying difficulty levels. Try to solve as many problems as possible within the given time frame. After the contest, analyze your solutions and compare them with others.

6. Learn Data Structures and Algorithms:

Data structures and algorithms are essential concepts in Competitive Programming. They help in solving problems efficiently. Some of the essential data structures and algorithms are arrays, linked lists, stacks, queues, trees, graphs, sorting algorithms, and searching algorithms. Learn these concepts and practice implementing them.

7. Practice, Practice, Practice:

Practice is the key to becoming a successful Competitive Programmer. Spend time daily practicing problems and participating in contests. This will help you identify your weak areas and work on them.

8. Collaborate and Learn from Others:

Collaborating with other Competitive Programmers is an excellent way to learn new concepts and strategies. Join online communities and forums, and discuss your problems and solutions with other programmers. This will help you gain new perspectives and improve your problem-solving skills.

9. Keep Yourself Updated:

Competitive Programming is constantly evolving, and new concepts and techniques emerge regularly. Keep yourself updated with the latest trends and changes in the field. Read blogs, watch videos, and participate in discussions to stay on top of the latest trends.

10. Don't Give Up (Most Important)

Competitive Programming can be challenging and frustrating at times. But don't give up! Keep practicing, and eventually, you will become better. Remember, becoming a successful Competitive Programmer takes time, effort, and patience.

Common topics

- Data structures: arrays, linked lists, stacks, queues, trees, graphs, etc.
- Algorithms: sorting, searching, dynamic programming, greedy algorithms, etc.
- Number theory and combinatorics
- Game theory
- Graph algorithms: graph traversals, shortest paths, minimum spanning tree, etc.
- String algorithms: pattern matching, string compression, etc.
- Geometry

- Combinatorial optimization
- Advanced data structures like Trie, Segment Tree, Fenwick tree, etc.
- Computational geometry

In addition to the topics listed above, some other important areas in competitive programming include:

- Dynamic programming: This involves breaking a problem down into smaller subproblems and storing solutions to those subproblems to avoid redundant computation.
- Divide and conquer: This involves breaking the problem into smaller subproblems and solving them independently.
- Greedy algorithm: This involves making the locally optimal choice at each stage with the hope of finding the global optimum.
- Backtracking: This involves finding all possible solutions by incrementally building candidate solutions and abandoning a candidate as soon as it is determined that the candidate cannot possibly be completed to a valid solution.
- Bit manipulation: This involves manipulating the individual bits of a number, such as counting the number of set bits in a binary representation of a number.

It's important to have a good understanding of these topics if you want to become a successful competitive programmer. You can start by practicing problems related to these topics on online platforms or participating in online programming contests.

Platforms

There are several platforms available for practicing competitive programming, some of the most popular ones being CodeForces, LeetCode, HackerRank, and CodeChef.