← Notes

Learn to Code by Competitive Programming

47 Competitive Programming

How do I Learn to Code? This is probably the most nagging question at the back of your mind, once you have decided that you want to learn programming. Like learning anything else, there is no standard process for learning to code. Of course there are guidelines, there are courses, there are ideologies and there are set traditions, but there is no one single correct way.

One school of thought which is very popular and fairly simple to begin with is Competitive Programming. Getting started with it is quite easy and if one devotes sufficient amount of time and effort, you can develop a very strong grasp of programming logic in relatively short amount of time.

Here are some steps to get started and be good at it.

- Get comfortable writing code in either of one of these languages **C, C++ or Java**. Why only C, C++ or Java? Because these are the standard languages allowed in any programming competition.
- If you are already good at C, it is suggested to **learn C++**. It is the most popular language among competitive programmers because of its speed and an excellent library in the form of STL (Standard Template Library).
- Pick an online judge. Recommended ones are **Topcoder** and **Codeforces**. These sites have high quality of problems and also allow you to see other's code post contest completion. These also categorize problems based on the topic. Some other popular judges include SPOJ, CodeChef (powered by SPOJ) and HackerEarth.
- To begin with, **start with simple problems** that typically require transforming English to code and does not require any knowledge on algorithms. Solving Div 2 250 (Division 2, 250 points) in Topcoder or Div 2 Problem A in Codeforces is a good start.
- At the early stages of programming one tends to write long pieces of code, which is actually not required. Try to keep codes **short and simple**.
- **Practice** these problems until you become comfortable that you can submit it for 240 odd points on any day.
- Start implementing basic(or standard) algorithms. It is suggested to read them from

Topcoder tutorials or Introduction to algorithms.

Some basic concepts that you should learn are

1. Graph algorithms: Breadth first search(BFS), Depth first search(DFS), Strongly connected components(SCC), Dijkstra, Floyd-Warshall, Minimum spanning tree(MST), Topological sort.
2. Dynamic programming: Standard dynamic programming problems such as Rod Cutting, Knapsack, Matrix chain multiplication etc.
3. Number theory: Modular arithmetic, Fermat's theorem, Chinese remainder theorem(CRT), Euclidian method for GCD, Logarithmic Exponentiation, Sieve of Eratosthenes, Euler's totient function.
4. Greedy: Standard problems such as Activity selection.
5. Search techniques: Binary search, Ternary search and Meet in the middle.
6. Data structures (Basic): Stacks, Queues, Trees and Heaps.
7. Data structures (Advanced): Trie, Segment trees, Fenwick tree or Binary indexed tree(BIT), Disjoint data structures.
8. Strings: Knuth Morris Pratt(KMP), Z algorithm, Suffix arrays/Suffix trees. These are bit advanced algorithms.
9. Computational geometry: Graham-Scan for convex hull, Line sweep.
10. Game theory: Basic principles of Nim game, Grundy numbers, Sprague-Grundy theorem.

The list is not complete but these are the ones that you encounter very frequently in the contests. There are other algorithms but are required very rarely in the contests.

You can find description and implementation of standard algorithms here.

- Once you have sufficient knowledge of popular algorithms, you can start solving the medium level problems. That is Div 2 all problems in Topcoder and Codeforces. It is advisable not to go for Div 1 500 at this point.

- Learning to code is all about practicing. **Participate regularly** in the programming contests. Solve the ones that you cannot solve in the contest, after the contest. Apart from Topcoder and Codeforces you can also look at HackerEarth Challenges or Codechef contests.

- **Read the codes** of high rated programmers. Compare your solution with them. You can observe that it is simple and shorter than your solution. Analyse how they have approached and improve your implementation skills.

- **Read the editorials** after the contest. You can learn how to solve the problems that you were not able to solve in the contest and learn alternative ways to solve the problems which you could solve.

- Always **practice the problems that you could solve in the contest**. Suppose if you are able to solve Div 2 250 and 500 in the contest but not Div 2 1000 then practice as many Div 2 1000 problems as as you can.
- **Do not spend too much time** if you are not getting the solution or are stuck somewhere.
- After you feel that you have spent enough time, look at the editorials. Understand the algorithm and code it. Do not look at the actual solution before you have attempted to write the code on your own.
- Programming is a very practical and hands on skill. You have to continuously do it to be good at it. It's not enough to solve the problem theoretically, **you have to code it and get the solution accepted**. Knowing which algorithm/logic to use and implementing it are two different things. It takes both to be good at programming.
- Programming learning phase is going to take a lot of time and the key is **practicing regularly**. It takes some time before you can attempt Div 1 500 and other tough problems. Do not give up on reading the editorials and implementing them, even if it takes many hours/days. Remember everything requires practice to master it.

It takes considerable amount of time before you get good at it. You have to keep yourself motivated throughout. Forming a team and practicing is a good choice. **Not giving up is the key here**.

Hackerearth conducts monthly online programming contests starting from this month. Register here to get started and receive further updates.

AUTHOR

MV Kaushik
Hyderabad
1 note

Write Note
My Notes
Drafts

TRENDING NOTES

Strings And String Functions
written by Vinay Singh

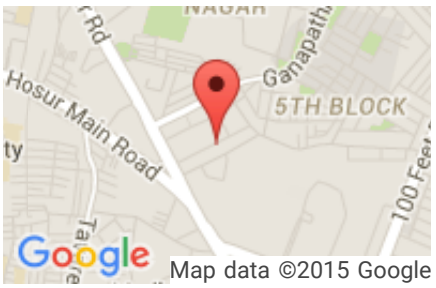
Segment Tree and Lazy Propagation
written by Akash Sharma

Number Theory - II
written by Tanmay Chaudhari

Matrix exponentiation
written by Mike Koltsov

Graph Theory - Part II
written by Pawel Kacprzak

more ...



Bangalore, Karnataka 560095, India.

contact@hackerearth.com

+91-80-4155-4695

+1-650-461-4192