A Study Guide for Cracking the <Coding> Tests and Interviews

By Dahab Shakeel

Table of Contents

1.Before Getting Started

- This guide is based on my personal experience and is highly inspired by a lot of useful resources I came across during my college years.

- I don't guarantee you a job at FAANG but if you follow this guide with full dedication you will have a way better chance to land a good job offer.

- This guide covers all fundamentals of Algorithms you may need to land your dream job.

- It is not necessary that your dream job is FAANG or even in the US so this guide will prepare you to be ready for coding interviews at any highly skilled company.

- This guide will also prepare you for coding tests at companies in South Korea (e.g. SAMSUNG).

- This guide is not explicit for any programming language but since I used to practice JAVA, there are some extra notes for JAVA users.

- The tasks within this guide are aimed to make you prepared within 2-4 months depending on how much time you put every week.

- Enjoy < coding> and don't stress too much about your future!

2.Weekly Schedule

This weekly schedule is aimed towards getting comfortable with the programming language of your preference and getting the hang of the basic algorithms.

- Week-1:

1) Pick a programming language (Java and C++ recommended)
2) Review Basics of your Programming language:
   A- Read/Write from files
   B- Read Input from console
   C- Split Strings based on a delimiter
   D- Change Strings to other data types and vice-versa
   E- String functions
   F- Arrays
   G- Copying and sorting arrays (and other array functions)
   H- Classes/ Functions/Array of class instances
   I- Dynamic Arrays (e.g. Vectors/ ArrayList)

3) Data Structures Review (Just practice how to define and work with each of these):
   A- Stacks
   B- Queues (e.g. Linkedlist Normal Queues)
   C- LinkedList
   D- Trees (General/Binary Search/MST)
   E- Trie
   F- Graphs (Directed/Undirected)
   G- HashTable/ HashMap/ LinkedHashMap
   H- Set/HashSet
   I- Heap/Priority Queue

- ## Week-2:

1) Implement simple BFS (Iterative)
2) Implement simple DFS (Recursive and Iterative)
3) Kruskal's Algorithm
4) Prim's Algorithm
5) Bellman Ford's Algorithm
6) Dijkstra's Algorithm (Implementation is important)
7) Floyd Warshall's Algorithm

- ## Week-3:

1) Sorting
    a. Insertion
    b. Selection
    c. Merge
    d. Heap
    e. Quick
    f. Bubble
    g. Radix
    h. Bucket
    i. Counting
2) Searching:
    a. Linear
    b. Binary
    c. Jump
3) Get Familiar with what Dynamic Programming is
4) Get Familiar with Backtracking
5) Get Familiar with Greedy Algorithm

- ## Week-4:

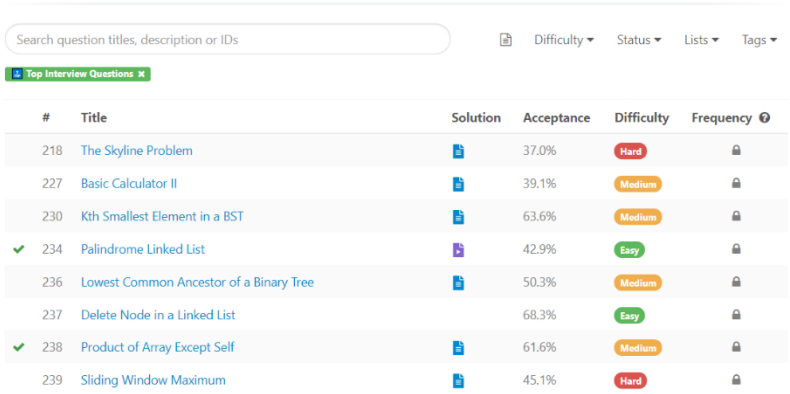
1) Practice simple algorithmic problems:
    a. Remove Even Integers from an array
    b. Merge two sorted arrays
    c. Find first non-repeating integer in an array
    d. Find the second maximum value in an array
    e. Create all possible subsets (power set) + permutations of a string
    f. Binary Search
    g. Reverse words in a sentence
2) Practice simple Linked List problems:
    a. find the length of a linked list
    b. Search in a Linked List
    c. Find the middle value of a linked list
    d. Reverse a Linked List
    e. Find Intersection of two linked lists
3) Practice simple Queue/Stack problems:
    a. Sort values in stack
    b. Implement two stacks using one array

- ## Week-5:

1) Practice tree problems:
    c. Find a minimum value BST
    d. Traverse and print a BST
    e. Find height of a BST
    f. Remove a node in BST
    g. Search in BST
    h. Insert in BST
    i. Check if two BSTs are identical
2) Practice graph problems:
    a. Detect a cycle in a graph
    b. Clone a directed graph
3) Practice heap problems:
    a. Find K smallest elements in a list
    b. Find K largest elements in an array
4) Find if there are 2 integers in an array that add up to equal K in O(n)
5) Find if there are 3 integers in an array that add up to equal K in less than O(n^3)
6) Maximum subarray problem

3.Daily Practice

- Focus on using:
    a. LeetCode
    b. Baekjoon (For Korean companies)
- Filter “Top Interview Questions” in LeetCode



- Start with easy problems
- Spend the most time doing medium problems
- Practice some hard problems
- Before your interview, filter the problems based on the company.
- Practice every single day!

4.Programming Patterns Notes

- **These notes will only make sense once you finish the previous practice and reach to an okay or good level of understanding.**
- **Notes on DP:**

- Generally used when the question is about "Maximize" or in "How many ways".
- The key to solve dynamic programming problems is to decide between a single or a 2D dp array.
- Try to always think about the simplest way and what dp[i]/dp[i][j] represents.
- If you do have multiple lengths of sequences: the outer loop should be the Length (e.g. Palindrome problem) so you have to think about Length,i, j. In other questions you have to think about a split point too (e.g. Matrix Chain Multiplication problems) so you have to think about Length, i, j, k (where j is not a looping variable and is calculated using i and Length).
- If it is a 2d maze (array/map) directly go for a 2D dp array.
- In Maximum/ Minimum Path/Value/Sum problems: <u>Think about whether you should pick the current element or not (What is better?).</u>
- Remember that DP is an optimization for DFS so every DP problem is solvable through DFS.

- **Notes on DFS and BFS:**

- Always visit the tree nodes as following (this way it will reduce time limit and adding unnecessary children)
  1) If not visited (the child).
  2) Mark it as visited (the child).
  3) Visit the child (add to queue or call DFS).

- **Notes on DFS:**

- Goes deep in the recursive tree.
- Exhaustive search for all paths/ combinations.
- It can be used for backtracking IF you want to try picking different items.
- Most importantly think about what the children at a specific point are.
- Remember DP is an optimization for DFS.
- If the original map is being changed each DFS make sure to create a new one every time before you call DFS.
- Sometimes loops are enough to try all possible cases (think simpler)

- **Notes on BFS:**

- Goes wide in the recursive tree
- Checks the shortest path between two nodes (Keyword: minimum)

- Since BFS visits the tree in a wide not a deep fashion, we can use it to traverse level by level.

```
while(!q.isEmpty)  // traverse level by level all the tree you can change this to loop
                   // if you want some level not all levels
{
        int size=q.size();
        for(int i=0;i<size;i++) //traverse a single level
        {
                // Normal BFS operations
        }
}
```

- Most importantly you have to keep track of the visited nodes and eliminate any repetitive traversal and for that you can use the following:
  1) HashSet: if the node states are really far away from each other. Also, you can use a HashSet for checking if the node has been visited or not provided the node is a 2d array (e.g. PROBLEM) or anything complicated in this case change the complicated thing to a string and use the HashSet.
  2) 2d Boolean check array/ 1d Boolean check array: Most common.
  3) 3d Boolean check array: where 2 dimensions are the coordinates in the 2d array and one dimension is a state where this state can be: seconds, direction travelled to reach here, number of broken walls.
     a. In other words, you have to ask yourself, "Have I visited this position in the map when the state was *X*".
  4) 4d Boolean check array.
  5) Integer 2d array: very uncommon (e.g PROBLEM).
- You can reset the visited array if the situation is totally different and now you can visit previously visited states.
- You may need to use a priority queue if you want to move based on the value of something rather than the distance.

- **Notes on Java:**
- Most important data structures are:
  1) ArrayList
  2) Queue/ Priority Queue
  3) HashSet
  4) HashMap
  5) Dequeue (adds to both ends in O(1))

- StringBuilder is faster than a string and you can use it if you have a lot of test cases so that you add the answer to StringBuilder and just print once at the end
- Hashmap is faster
- ArrayList is faster
- You may need a comparator for a priority queue as the following:

```
Comparator<node> comp = new Comparator<node>() {

                @Override
                public int compare(node o1, node o2) {
                        // Pick the node with least number of days
                        // Ascending order
                        return o1.day - o2.day;
                }
        };
```

- **Other Algorithmic Notes:**

- To rotate an n*n array clockwise: rotated[i][j]=original[(n-1)-j][i]
- To rotate an n*n array counter clockwise: rotated[(n-1)-j][i]=original[i][j]
- If you want to do swapping remember prvTemp and curTemp.
- You can maintain an ArrayList<point> to move things on the map if you have groups since it is easier to move and once you move the List<point> you can update the map easily (e.g. PROBLEM).
- You can also take it a step further and make a List<ArrayList<point>> to store the groups (e.g. PROBLEM).
- Think about grouping when there are similar points in an array.

5.Programming Patterns Notes

1) Describe the process you have for a programming task, from requirements to delivery?

- Choosing Waterfall or Agile/Iterative model.
- Requirement Analysis and Specification
- Software Architecture
- Implementation
- Testing
- Documentation
- Training and Support/ Maintenance

2) What programming languages do you use? Which three do you prefer or are most familiar with?

(Sample Answer)

- Java – Android and Spring
- C - FPGA
- Python – AI

3) How do you implement error handling?

- Write tests
- Catching exceptions (try/catch)

4) What is the software development life cycle? What are the differences between them?

- The process of making a software project from requirement to maintenance with highest quality and lowest cost in the shortest time possible.
- Waterfall/Iterative/ Spiral

5) How comfortable are you in a startup environment and why do you prefer it over big companies?

- Whatever your answer is always mention:
    - Decision making
    - Gaining exponential experience

6) How much Salary do you expect?

- Never mention a single number, instead give a range which matches the position and try put your preferred number right in the middle of the range.

7) Strength/ Weakness?

(Sample Answer)

- Strength: Put a lot of time and effort/ team leading (mention a uni project).
- Weakness: I rarely say no to people and because of my strength I end up with little to no time.

8) What are your goals you want to accomplish with us?

- Whatever your answer is mention long and short-term goals

9) Do you have questions for me (The recruiter)?

- Never say NO!
- Ask about the position you are applying to
- Get to know about the projects that the company is currently having

10) More technical questions can be found here:

- Reference-1
- Reference-2
- Reference-3
- Reference-4
- Reference-5

6.Useful Links

- Whiteboard Problems
- Famous DP Problems
- Important LeetCode Problems
- Important BaekJun Porblems
- Basic Data Structure Course