Interview Preparation Guide

Contents

Data structures 1

Algorithms (Problem solving) 1

System Design 2

Object oriented programming 3

Java 3

Database Queries 3

Operating System 3

General preparation References 4

General interview Questions: 4

Questions to ask the interviewer 4

# Data structures

* Big O cheat sheet for each DS
* Stack
* Queue
* LinkedList
* Hashtable
* StringBuffer
* Tree representation & traversal (Binary, BST, AVL and RedBlack)
* Graphs representation & traversal (DFS, BFS)
* Trie
* Heap (Min and Max)
* Resources
  + CTCI

# Algorithms (Problem solving)

* Graph Algorithms
  + Minimum spanning tree (Prim’s & kruskals)
  + Dijkstra’s shortest path
  + Topological Sort
* String Algorithms
  + KMP pattern
* Sorting algorithms (merge, quick)
* Greedy
* Backtracking
* Recursion
* Dynamic programming (Knapsack)
* NP complete problems (travelling salesman)
* Bit manipulation
* Resources
  + http://codercareer.blogspot.com/?m=1
  + GeeksOfGeeks
  + Leetcode
  + Hackerrank
  + Slack-sde-skills(amazon)
  + Careercup
  + CTCI
  + Refer [https://github.com/jwasham/google-interview-university#the-daily-plan](https://github.com/jwasham/google-interview-university%23the-daily-plan) for Recursion, DP and Graph

# System Design

* Cache’ing techniques
  + Updating cache
    - Write through
    - Time based update
  + Eviction techniques
    - LRU
* Data distribution in distributed architecture
  + Data based
  + Using lookup table
* Resources
  + https://www.hiredintech.com/courses/system-design
  + <https://github.com/shashank88/system_design>
  + CTCI
  + Check my System\_Design doc notes

# Object oriented programming

* Resources
  + CTCI

# Java

* Threads & Concurrency
* Collections api thoroughly (including methods)
* Rest and SOAP difference and understanding of each with Apple POS
* Current project in HP

# Database Queries

* Resources
  + CTCI

# Operating System

* Resources
  + https://github.com/jwasham/google-interview-university#the-daily-plan

# General preparation References

* https://trananh.github.io/blog/work/2014/02/24/preparing-for-a-google-phone-interview.html - google interview prep
* [https://github.com/jwasham/google-interview-university#the-daily-plan](https://github.com/jwasham/google-interview-university%23the-daily-plan)

# General interview Questions:

* Why do you want this job?
* What's a tough problem you've solved?
* Biggest challenges faced?
* Best/worst designs seen?
* Ideas for improving an existing Google product.
* How do you work best, as an individual and as part of a team?
* Which of your skills or experiences would be assets in the role and why?
* What did you most enjoy at [job x / project y]?
* What was the biggest challenge you faced at [job x / project y]?
* What was the hardest bug you faced at [job x / project y]?
* What did you learn at [job x / project y]?
* What would you have done better at [job x / project y]?

# Questions to ask the interviewer

* How large is your team?
* What does your dev cycle look like? Do you do waterfall/sprints/agile?
* Are rushes to deadlines common? Or is there flexibility?
* How are decisions made in your team?
* How many meetings do you have per week?
* Do you feel your work environment helps you concentrate?
* What are you working on?
* What do you like about it?
* What is the work life like?

Schedule

|  | **Topic1** | **Topic2** | **Topic3** |  |
| --- | --- | --- | --- | --- |
| **Jan 8-14** | Trees & Graphs | Trie | Heap |  |
| **Jan 15-21** | Bit Manipulation |  |  |  |
| **Jan 22-28** | Recursive and DP |  |  |  |
| **Jan 29-Feb 4** | Sorting and searching algorithm | Backtracking | Greedy |  |
|  | System Design and scalability |  |  | Practice Problem Solving |
|  | Database |  |  | Practice Problem Solving |
|  | Object oriented design |  |  | Practice Problem Solving |
|  | Operating System |  |  | Practice Problem Solving |
|  | Java |  |  | Practice Problem Solving |
|  | DS |  |  | Practice Problem Solving |
|  | Arrays & Strings |  |  | Practice Problem Solving |
|  | Stacks & queues |  |  | Practice Problem Solving |
|  | LinkedList |  |  | Practice Problem Solving |

**Problem solving Techniques:**

* Pre-compute information which will not impact the runtime.
* Use Hashtable
* Think of best conceivable runtime. Try working from that to optimize
* Use BUD
* Look for unused information in question (to optimize) and make sure to use all information in Qn.

**Big O Notes:**

* [1+2+3 … + (n-1)] = n(n-1)/2
* [1+2+3+…+n] = n(n+1)/2
* 2 pow1 + 2 pow2 +…. 2 pow n = 2 pow (n+1)
* Recursive call runtime = branches pow depth (for binary the branches would be 2)
* Generally algorithm with multiple recursive calls, you are looking at exponential runtime (2 pow n)
* Number of times we can halve n until to get 1 is O(log n)

**Bit Manipulation**

1. Left shift is equivalent to multiplying the bit pattern with 2(pow)k. Example - a<<k =a\*2(pow)k
2. Right shift is equivalent to dividing the bit pattern with 2(pow)k. Example - a>>k =a\*2(pow)k
3. x-1 can be obtained by simply flipping all the bits to the right of rightmost 1 in x including the rightmost 1. Examples
   1. x =6 = 110, x-1 = 5 = 101
4. x & (x-1) will have all the bits equal to the x except for the rightmost 1 in x. Example
   1. x = 4 = 100, x-1 = 3 = 011
   2. x & (x-1) = 000
5. Properties for numbers which are powers of 2, is that they have one and only one bit set in their binary representation.
6. So if x is a power of 2 then (x & (x-1)) will be 0
7. 2(pow)k can be written as (1<<k). Example 2(pow)2 = 4 = 100
8. x ^ ( x &(x-1)) : Returns the rightmost 1 in binary representation of x.