Google Interview Prep

- Binary Tree Longest Consecutive Sequence
  - help(node, parent, len)
- Longest Increasing Subsequence
  - dp[i]: max ending here
- Number of Islands
  - find '1' and recursively call merge which set bits to '0'
- Paint Fence
  - 2 dp: dp1[i]=dp2[i-1]; dp2[i]=(k-1)(dp1[i-1]+dp2[i-1])
  - recursion
  - queue of size 3?
- Power of Two
  - n > 0 && (n & (n - 1)) === 0
  - while n /= 2, n === 1
- Plus One
  - if digit[i] === 9 change to 0, else return; if not return until end, change 1st bit to 1 and push 0 Link
  - Follow Up
    - String not Listening
- Wiggle Sort
- LRU Cache **
  - O(n): Queue as List + HashTable (Look up O(1))
  - O(1): Queue as Double Linked List + Hashtable + dummy head & tail to avoid check null Link
- H-Index
  - H-Index II
- Merge Intervals
  - merge two
  - insert into a list
- Walls and Gates
- 3Sum
- 3Sum Smaller

    - for (var k = 2; k < nums.length; k++) { count += twoSumSmaller(nums, target - nums[k], k - 1); }
- Lowest Common Ancestor of a Binary Tree
- Merge Two Sorted Lists
    - Recursion Link
    - Dummy Node
- Merge Sorted Array **
    - in place, ptr for both arr
    - start from total length end and both end then set to front Link
- Excel Sheet Column Number
    - Number + Order = s.length - 1
- Excel Sheet Column Title
- Reverse Linked List
    - Recursion (set head.next = null at end)
- N-Queens, N-Queens II
    - 1D array store Q pos info, validity check ALL 4
    - return when n === row
- Multiply Strings **
- Add Two Numbers
    - Dummy Node
- Serialize and Deserialize Binary Tree
    - Pre-order traversal + null
- Sqrt(x)
    - binary search x^2
- Perfect Squares Solutions
    - dp: For each i, it must be the sum of some number (i - j*j) and a perfect square number (j*j)
    - BFS
    - Mathematics: Lagrange's four-square theorem
- Fraction to Recurring Decimal **
- Sort List
    - Extract value, sort
    - 2 Ptr to Mid Point, Merge Sort

        - two ptr init pos diff
- Missing Ranges
    - push start - 1 and end - 1, check diff between nums[i + 1] and nums[i] ?= 1 ?= 2 ?> 2
- Search a 2D Matrix
    - row binary search, col binary search
    - treat as single list Link
- Letter Combinations of a Phone Number
- Factorial Trailing Zeroes
- Valid Parentheses
- Strobogrammatic Number,Strobogrammatic Number II, Strobogrammatic Number III **
- Shortest Palindrome
- Decode Ways
    - dp
- Search Insert Position
- Roman to Integer
- Find Peak Element
    - Binary Search to find local maximum
- Single Number
    - sum
    - bitwise
- Swap Nodes in Pairs
    - dummy node
    - recursion Link
- Trapping Rain Water
- Convert Sorted Array to Binary Search Tree
- Two Sum
- Longest Substring with At Most Two Distinct Characters **
    - hash table: record occurrence
    - 2 ptr: start of the substring; current pos
    - Link
- Kth Largest Element in an Array
- Unique Word Abbreviation

- Rotate Image
  - line to line reconstruction
  - flip then mirror symmetry Link
- House Robber
  - 2 dp: Robbed, Unrobbed
- House Robber II
  - UnRob first OR UnRob last
- Find Minimum in Rotated Sorted Array, Find Minimum in Rotated Sorted Array II
- Generate Parentheses
  - only left paren is added, then right paren can be added
- Encode and Decode Strings
- Populating Next Right Pointers in Each Node, Populating Next Right Pointers in Each Node II
- Expression Add Operators
- Container With Most Water
  - https://leetcode.com/discuss/11482/yet-another-way-to-see-what-happens-in-the-o-n-algorithm
- Smallest Rectangle Enclosing Black Pixels
- Climbing Stairs
  - Permutation x + 2y = n
  - Fibonacci
- Flatten 2D Vector
- Add Digits
  - Digit Root: dr(n) = 1 + (n - 1) % 9
- Count and Say
- Path Sum II
- Closest Binary Search Tree Value, Closest Binary Search Tree Value II
  - Update Min Diff when Traversal

- Longest Consecutive Sequence
- Clone Graph
- Palindrome Permutation II
- Word Ladder
- Word Ladder II
- Unique Binary Search Trees
  - dp
- Spiral Matrix, Spiral Matrix II
- Distinct Subsequences
- Permutations II
- Sliding Window Maximum
- Best Time to Buy and Sell Stock
  - dp[i + 1] = Math.max(dp[i], prices[i] - min)
- Best Time to Buy and Sell Stock II
- Best Time to Buy and Sell Stock III
- Best Time to Buy and Sell Stock IV
- Unique Paths II
- Verify Preorder Sequence in Binary Search Tree
- Word Search, Word Search II
- Word Ladder, Word Ladder II
- Spiral Matrix

- Coins in a Line
  - dp(i, j) = sum{Ai ... Aj} - min { dp(i+1, j), dp(i, j-1) }
    - Ai..Aj left and your turn to pick

    - dp(i, j) = max { Ai + min { P(i+2, j), P(i+1, j-1) }, Aj + min { P(i+1, j-1), P(i, j-2) } }
    - base cases: dp(1,1), dp(2,2), … dp(n, n)
- Height/Level of a Binary Tree
    - Recursion
    - Iteration BFS, level-wise traversal, dequeue all level at once
- Implement strcspn
    - hash table
    -
    - ```
      var strcspn = function (str1, str2) {
        var hash = {};
        for (var i = 0; i < str1.length; i++) {
          if (!hash[str1[i]]) { hash[str1[i]] = i; }
        }

        var min = str1.length - 1;
        for (var i = 0; i < str2.length; i++) {
          if (hash[str2[i]] < min) {
            min = hash[str2[i]];
          }
        }
        return min;
      };
      ```
- Moving Average
- Moving Median
- Random number within a triangle
- Given string composed of 0,1,? output all permutations
    - backtracking
- Word Abbreviation, number replace length, output all possibility
    - abbr or not abbr, 2^n
- Find Inserted Char
    - bitwise
- Deep Copy Tree
- Mirror Binary Tree
- Given x and y = ax*2 + b*x + c, Sort y
- Reverse Vowels of a String
      ```

- Segregate Even and Odd numbers
  - 2 ptr, left & right, meet diff match, swap
- Count smaller elements on right side
- Vertical Traversal
- Value 1-n, given string i (incr) d (decr), output 1 possible permutation O(n)?

```
var pattern = function (str) {
  var c = [1,2,3,4,5,6,7,8,9];
  if (str.length === 0) { return [1];}
  var darr = [];

  if (str[0] === 'd') {
    darr.push(0);
  }
  for (var i = 1; i < str.length; i++) {
    if (str[i] === 'd') { darr.push(i); }
  }
  var limit = darr.length;
  if (limit > c.length / 2) {
    limit = c.length/2;
  }
  for (var i = 0; i < limit; i++) {
    swap(c, darr[i], c.length - i - 1);
  }
  return c.slice(0, str.length + 1);
}
```

- 1D list of points, find sum of distance minimal
  - median
- Unsorted Array Remove Duplicates
- k stack, take n, return maximum sum
  - backtracking
- Check Input in Ranges
  - binary search, l <= r, else return l - 1
- Find Inserted Char of 2 (Shuffled/Unshuffled) Strings
- 1234567891011… Find nth char

```
var getc = function (index) {
  var n = 0;
  while (index > n*9*Math.pow(10, n - 1)) { n++; }

  var left = index - (n - 1)*9*Math.pow(10, n - 2) - 1;

```

```
  var num = Math.pow(10, n - 1) + Math.floor(left/n);
  var rem = left%n;
  return num.toString()[rem];
};
```

- Merge Quad Tree

```
function Node(val) {
  this.val = val;
  this.children = new Array(4).fill(null);
}

var merge = function(r1, r2, target) {

  if (!r1 || !r2) { return; }

  if (r1.val !== undefined && r2.val !== undefined) {
    target.val = r1.val & r2.val;
  } else {
    for (var i = 0; i < 4; i++) {
      target.children[i] = new Node();
    }
    if (r1.val !== undefined) {
      for (var i = 0; i < 4; i++) {
        merge(new Node(r1.val), r2.children[i],
target.children[i]);
      }
    } else if (r2.val !== undefined) {
      for (var i = 0; i < 4; i++) {
        merge(new Node(r2.val), r1.children[i],
target.children[i]);
      }
    } else {
      for (var i = 0; i < 4; i++) {
        merge(r1.children[i], r2.children[i],
target.children[i]);
      }
    }
  }
};

var mergeQuadTree = function (r1, r2) {
  if (!r1 || !r2 ) { return null; }
  var roo = new Node();
  merge(r1, r2, roo);
  return roo;
};
```

- Binary Tree AND
- Find common directory path
  - split, check at the same pos whether exist
  - Trie, if different, mark and quit

- Calculate log(x)
- Give you a sorted dictionary with 1M words and a prefix. Find the first valid prefix string for this dictionary
  - binary search
  - prefix tree
- Decoding Morse Code
- List of boarding passes, return Travel Itinerary
  - DFS
- Design a algorithm to initialize the board of Candy Crush Saga. With M x N board, Q types of candies

Resume

Course Work

Operating Systems
Databases
Algorithms and Data Structure
Software Engineering
Program Design and Development
Animation and Planning in Games
Machine Architecture and Organization
Applied Linear Algebra

Language

JavaScript

C/C++
Python
Shell Script
PHP
MatLab
Java

Library

Three.js
D3.js
jQuery
Node.js
PostgreSQL

The interviewer(s) will be interested in the specifics of your past projects, implementations and how you arrived at your conclusions.

- Most Challenging
- What You Learned
- Most Interesting
- Hardest Bug
- Enjoy Most
- Conflicts with Teammates

Water Monitoring System

Features

1. Re-designed w/ responsiveness
2. Individual waterbody changes over time (grow, shrink) w/ interactive, animated chart (D3.js), visualized by a) playing polygon movie, google earth engine
3. Range changes over time: tile, basin, custom region
4. Query over waterbodies: size, score
5. Different types of score visualization
7. Scripts to automate data processing and uploading
8. Optimize flow of data, separate spatial/non-spatial data

9. Optimize loading speed: preload polygon data

Implementation

1. Front-end: jQuery: AJAX call, gradually replace by vanilla JS; Git: version control, add feature
2. Back-end: PostgreSQL/PostGIS: store data/spatial data, Geoserver: host raster image, tile caching, Linux/Shell, Python: data processing automation

Conclusion

1. Framework like MVC
2. Security
3. Optimization: code optimization, CDN, no jQuery

Course Search App

Features

1. Responsive Design
2. Faster than UMN Official One

Implementation

1. Front-end: Javascript
2. Back-end: Node.js/Express: express generator is fast; Heroku deployment is easy

Conclusion

1. Try Angular, ReactJS for better performance/further updates

Chat App

Features

1. Sign in/sign up
2. Minimalist Design
3. Global access, user number limitation

Implementation

1. Front-end: Socket.io
2. Back-end:
   Node.js
   Express: handle request & response, router

Socketio: message between server & client, server emit/broadcast to client

MongoDB: store user login information

Passport.js: handle user authentication

Conclusion

1. No history stored because of MongoLab space limitation
2. Non responsive: prototype of integrating multiple libraries
3. No front-end & backend framework MVC

Image Server and Client

Features

1. TCP, web socket implementation

Implementation

1.

Conclusion

1.

Page Replacement Policy Implementation

Features

1. Replace LRU w/ FIFO

Implementation

1. locate code, change implementation, re-make, re-make install

Conclusion

1. Compression is hard because printing cost a lot of time, profiling tool is not powerful enough
2. Learned how to install PostgreSQL from scratch, different process problems and how to calm down

UNIX File System

Features

1. Support file read, write with implementation of i-node

table

Implementation

1.

Conclusion

Bus Routine Simulator

Features

1. Simulate bus circulation throught the day

Implementation

1. Bus class
2. Stop class
3. Rider class

Conclusion

1. OOD
2. Generate report on how many buses should be running at the same time

Heat Research

1. Heat as new media to communication, not visual, not able to hear, feel -> can simulate touch
2. No definite meaning of hotness, coldness, heating up, cooling down
3. Different interpretation -> different way of communication
4. Under prototype process
5. Use Peltier element to generate hotness and coldness, two hot side attached together because hot is more effective
6. Combined w/ other units like HP Sprout apps developed by one PhD in the research group, vibration mat

==========

- Google products (i.e. what you use)
- Coding Ability (you will be asked to code on the whiteboard - brush up on syntax and libraries)

- Algorithm Design/Analysis
- Systems Design
- Open-Ended Discussion (remember to think out loud and clarify the problem they ask you)

Coding
- construct / traverse data structures
- implement system routines
- distill large data sets to single values
- transform one data set to another

Algorithm Design / Analysis
- big-O analysis
- sorting and hashing, searching
- handling obscenely large amounts of data
- also see topics listed under 'Coding'

System Design
- feature sets
- interfaces
- class hierarchies
- designing a system under certain constraints
- simplicity and robustness
- tradeoffs

Open-Ended Discussions
- biggest challenge faced
- best / worst designs seen
- performance analysis and optimization
- testing
- ideas for improving existing Google products

Interviewing at Google - http://www.youtube.com/watch?v=w887NIa_V9w

Google Products - http://www.google.com/intl/en/options/
The Official Google Blog: Baby steps to a new job by Gretta Cook (GoogleEngineer) http://googleblog.blogspot.com/2008/01/baby-steps-to-new-job.html
How to Get Hired by Steve Yegge (Google Engineer) http://steve-yegge.blogspot.com/2008/03/get-that-job-at-google.html
How to Get Hired by Dan Kegel (Google Engineer) http://www.kegel.com/academy/getting-hired.html
Five Essential Phone Screen Questions by Steve Yegge (Google Engineer) https://sites.google.com/site/steveyegge2/five-essential-phone-screen-questions
Binary Search - http://en.wikipedia.org/wiki/Binary_search
Project Euler - http://projecteuler.net/
Types of algorithm questions Google asks: Top Coder Tutorials http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=alg_index
Industry News: Search Engine land - http://searchengineland.com/

==========

Algorithm Complexity

Coding: C++
Programming Interviews Exposed; Secrets to landing your next job

System Design
Distributed Systems and Parallel Computing, Google File System, Google Bigtable, Google MapReduce

1. Use case

    1. direct cases, other services
2. Constrains
3. Abstract Design: that illustrates the basic components of the system and the relationships between them
4. Bottlenecks these components face when the system scales
    1. handling a lot of users
    2. handling a lot of data
5. Address these bottlenecks by using the fundamentals principles of scalable system design (Vertical scaling, Horizontal scaling)
    1. [horizontal scale] clone, load balancer
    2. database
        1. master-slave replication (read from slaves, write to master), [vertical scale] more RAM on master
        2. OR NoSQL (no join)
    3. in-memory caches, e.g. Memcached or Redis
        1. Cached Database Queries
        2. Cached Objects
    4. Asynchronism

Sorting

quicksort: avg: O(n*lg(n)) worst: O(n^2)
merge sort
insertion sort
heap sort
    Priority Queue

Hash tables

hash function h: map the universe U of all keys into {0, 1, ..., m–1}
    1. division method
    2. multiplication method: h(k) = (A·k mod 2w) >> (w – r)
operation: insert(S, x) search(S, k) delete(S, x)

resolving collision:

1. chaining
2. open addressing: h(k,0…m-1)
    a. linear probing: (h'(k) + i) mod m
    b. quadratic probing: h(k,i)=(h'(k)+c1*i+c2*i^2) mod m
    c. double hashing: h(k,i) = (h1(k) + i*h2(k)) mod m

implement w/ only array: tinirlove.bitbucket.org/Algo/hashTable.js

Trees

basic tree construction

tree traversal:

- in-order: left, root, right Θ(n)
- pre-order: root, left, right O(h)
- post-order: left, right, root O(h)

tree manipulation algorithms

- search O(h)

```
TREE-SEARCH(x, k) {
   if (!x || k === x.key) {
      return x;
   }
   if (k < x.key) {
   return TREE-SEARCH(x.left, k);
   } else {
   return TREE-SEARCH(x.right, k);
   }
}

INTERATIVE-TREE-SEARCH(x, k) {
   while (x && k !== x.key) {
      if (k < x.key) {
         x = x.left;
      } else {
         x = x.right;
      }
```

```
        }
        return x;
    }
```

minimum O(h)

```
    while (node.left) { node = node.left; }
    return node;
```

maximum O(h)

```
    while (node.right) { node = node.right; }
    return node;
```

successor O(h)

```
    if (node.right) { return
node.right.min(); }
    var y = node.parent;
    while (y && node === y.right) {
        node = y;
        y = y.parent;
    }
    return y;
```

predecessor

```
    if (node.left) { return node.left.max(); }
    var y = node.parent;
    while (y && node === y.left) {
        node = y;
        y = y.parent;
    }
    return y;
```

insert O(h): fist locate where inserted node's parent is (y in while loop), then decide whether left or right

```
    y = null
    x = T.root
    while (x) {
        y = x
        if (z.key > x.key) {
            x = x.right
        } else {
```

```
            x = x.left
        }
    }
    z.parent = y
    if (!y) {
        T.root = z;
    } else if (z.key > y.key) {
        y.right = z;
    } else {
        y.left = z;
    }
```

delete O(h)

```
    TRANSPLANT(T, u, v) {
        if (!u.p) {
            T.root = v;
        } else if (u === u.p.left) {
            u.p.left = v;
        } else {
            u.p.right = v;
        }
        if (v) {
            v.p = u.p;
        }
    }

    TREE-DELETE(T, z) {
        if (!z.left) {
            TRANSPLANT(T, z, z.right);
        } else if (!z.right) {
            TRANSPLANT(T, z, z.left);
        } else {
            y = TREE-MINIMUM(z.right); // y is
z's successor
            if (y.p !== z) {
                TRANSPLANT(T, y, y.right);
                y.right = z.right;
```

```
                y.right.p = y;
            }
            TRANSPLANT(T, z, y);
            y.left = z.left;
            y.left.p = y;
        }
    }
```

n-ary trees
trie-trees
red/black tree: balanced binary tree O(lg(n))
height bh(x) <= 2 log(n + 1)

implementation:
splay tree optional
AVL tree optional
, and know how it's implemented
tree traversal algorithms

full binary tree: every node other than the leaves has two children
complete binary tree: every level, except possibly the last, is completely filled, and all nodes are as far left as possible

Graphs

Representation

- objects and pointers: Using node objects, and pointers to adjacent nodes (with weights) to represent edges. Alternatively, using an Edge object to represent a connection between two nodes.
- Adjacency Matrix: For n nodes, a matrix of bool[n][n], with cell[a][b] set to True to show an edge exists between nodes a and b. For weighted graph, this would be a matrix of int[n][n], with cell [a][b] set to the edge weight (otherwise zero.) For a non-directed graph, [a][b] = [b][a]. For a directed graph [a][b] is independent of [b][a].

- Adjacency lists: Each node has a linked list of nodes where edges exist.

(objects and pointers, matrix, and adjacency list); familiarize yourself with each representation and its pros & cons. You should know the basic graph traversal algorithms: breadth-first search and depth-first search. Know their computational complexity, their tradeoffs, and how to implement them in real code. If you get a chance, try to study up on fancier algorithms, such as Dijkstra and A*.

Dijkstra: O(E*V*log(V))

A*:

Other data structures

You should study up on as many other data structures and algorithms as possible. You should especially know about the most famous classes of NP-complete problems, such as traveling salesman and the knapsack problem, and be able to recognize them when an interviewer asks you them in . Find out what NP-complete means.

NP: the set of all decision problems(questions with a yes-or-no answer) for which the 'yes'answers can be verified in polynomial time (O(n^k) by a deterministic Turing machine.

P: the set of all decision problems which can be solved in polynomial time by a deterministic Turing machine.Since they can be solved in polynomial time, they can also be verified in polynomial time. Therefore P is a subset of NP.

NP-complete: (non-deterministic polynomial) no one knows a good way to solve them optimally; an efficient algorithm for one

NP-complete problem is applicable to all NP-complete problems

P=NP: if a problem takes polynomial time on a nondeterministic TM, then one can build a deterministic TM which would solve the same problem also in polynomial time.

Mathematics

combination: order doesn't matter
    w/ repetition: (n + r - 1)!/(r!(n-1)!)
    w/o repetition (n-choose-k): n!/(r!(n - r)!)
permutation: order does matter
    w/ repetition: n^r
    w/o repetition: n!/(n - r)!

discrete math questions
counting problems
probability problems
combinatorics and probability
n-choose-k problems and their ilk – the more the better

Operating Systems

issue
    processes
    thread
    concurrency

locks
mutex
semaphores
monitors

deadlock

how to avoid them
livelock
how to avoid them

what resources a processes needs
what resources a thread needs
how context switching works
how context switch is initiated by the operating system and underlying hardware
scheduling
fundamentals of "modern" concurrency constructs (multi-core)

Misc

research publications
Google's Hybrid Approach to Research
Programming Pearls

linear Diophantine equation: ax + by = c has an integer solution iff gcd(a, b) | c

dynamic programming: *'Remember your Past'*
top-down with memorization: recursive, but stores subproblem solutions
bottom-up: solve smaller subproblems first

CSS
visibility: hidden - will not download image
display: none - will download image

TODO

- [ ] stable marriage problem
- [ ] maximum flow problem

- Fork-Fulkerson algorithm

Experience

if parameter is node, check **null**

if delete element from list, go back by 1: i = i - 1

MAX / MIN
    DP
    Greedy

==========

OO design
    class, object (and the difference between the two)
    instantiation
    method
    virtual method, pure virtual method
        virtual function can be overridden and the pure virtual must be implemented
    class/static method
    static/class initializer
    constructor
    destructor/finalizer
    superclass or base class
    subclass or derived class
    inheritance
    encapsulation
    multiple inheritance (and give an example)
    delegation/forwarding
    composition/aggregation
    abstract class: to be subclassed
    interface/protocol (and different from abstract class)

interface must provide an implementation of all the methods of that interface

method overriding
method overloading (and difference from overriding)
polymorphism (without resorting to examples)
is-a versus has-a relationships (with examples)
method signatures (what's included in one)
method visibility (e.g. public/private/other)

==========

LZW

LZW Compression

```
string s;
char ch;

s = empty string;

while (there is still data to be read) {
   ch = read a character;
   if (dictionary contains s+ch) {
      s = s+ch;
   } else {
      encode s to output file;
      add s+ch to dictionary;
      s = ch;
   }
}
encode s to output file;
```

LZW Decompression

```
string entry;
```

```
char ch;

int prevcode, currcode;
prevcode = read in a code;
decode/output prevcode;
while (there is still data to read) {
   currcode = read in a code;
   entry = translation of currcode from
dictionary;
   output entry;
   ch = first char of entry;
   add ((translation of prevcode)+ch) to
dictionary;
   prevcode = currcode;
}
```

Steps

1. clarify problem
   consider an example that is rich enough but not tedious
   disambiguate expected result
   state and clarify key assumptions: expect result, any memory or performance requirement
   clarify the function signature, input, output
2. start with first solution that comes to mind
   run at least 1-2 examples
   check edge cases
   clean up with reasonable var name
   ask interviewer if any questions before refine
3. refine the solution
   clarify assumption
   rinse, repeat
   compare the solution

analytic skills
sound design
    limitation
    corner cases
    error checking

big O for data structure