# Meeting coding challenges for engineering students

# Course descriptions

It has become increasingly important for engineering students working in the areas such as artificial intelligence, machine learning, data science, digital communications and multimedia to be proficient in solving both numerical and non-numerical problems using programming languages. Many companies now use coding challenges to screen job candidates. This course describes effective problem solving strategies and demonstrate using these strategies to solve typical coding challenge problems. The students are expected to know at least one programming languages and expected to complete many programming exercises to master these techniques. The course will illustrate strategies such as divide-and-conquer, greedy, dynamic programming, backtracking, branch-and-bound and space-time-trade-off. They will be used to solve example problems in artificial intelligence, string processing, graph search and others.

# Textbook

Cracking the Coding Interview: 189 Programming Questions and Solutions, by

Gayle Laakmann McDowell

Publisher: CareerCup; 6th edition (July 1, 2015)

ISBN-10: 0984782869

ISBN-13: 978-0984782857

ASIN: 0984782850

# Outline

|  |  |  |
| --- | --- | --- |
| week | Techniques | Examples |
| 1a | Basic:   * I/O of the problem * Problem partition, how * Class, function * Structure, pseudo code * Trace a simple example | * Merge sort: function I/F * Tower of Hanoi: function I/F, info representation:   + # of disks, source peg, destination peg, extra peg   + Use function to encapsulate a logic computational unit to make the program logic more clear and ease debugging * Fibonacci:   + Iterative implementation   + Recursive implementation   + Tail recursive implementation |
| 1b | Brute-force, Intuition | * TSP * Assignment * DSP/BFS * Searching * String matching * Sorting |
| 2a | Exhaustive search | * Closest pair * 8 queen * TSP * Knapsack * Combinatorial listing: all subsets, permutations, combinations, etc |
| 2b | Using the right ADT | List, tuple, set, dict, tree, or graph?   * Matching parentheses prob from the interview |
| 3a | Decrease & conquer, top-down and bottom up | * Polynomial evaluation * Exponentiation (reduce by half) * Binary search of sorted array * BST (binary search tree) * Selection/insertion sort * Facebook problem from the interview |
| 3b | Divide and conquer | * Element uniqueness problem: check whether all the elements in a given array of n elements are distinct. * Fake coin puzzle (Levitin’s book) or weighing problem (David Mackay’s book) * Find all Euler circuits of a graph * Merge/quick sort |
| 4a | Transform and conquer (reduction) | * Finding the median reduces to sorting. * Element distinctness reduces to sorting. * CPM (Critical Path Method) reduces to topological sort.   + PERT & CPM are usually used together * Arbitrage reduces to shortest paths. * 3-collinear reduces to sorting. [assignment]   + Cost of sorting N times + cost of reduction (angle computation) = * Convex-Hull * Burrows-Wheeler transform reduces to suffix sort. |
| 4b | Backtracking, e.g.   * 8 queen * Subset problem, e.g.   + xs = {1, 2, 5, 6, 8}, d= 9   + xs = {3, 5, 6, 7}, d = 15 * Find all Hamilton circuits of a graph | * Construct the state-space tree   + nodes: partial solutions   + edges: choices in extending partial solutions * Explore the state space tree using depth-first search * “Prune” nonpromising nodes: stop exploring subtrees rooted at nodes that cannot lead to a solution and backtracks to such a node’s parent to continue the search |
| 5 | Greedy | * 0-1 Knapsack * DFS, BFS, Dijkstra * MST:   + Kruskal’s   + Prim’s |
| 6 | Dynamic programming:   * Recurrence solution * Avoid overlapping/redundant computation by saving the solutions needed later | Fibonacci number   * Brute-force recursive implementation * Dynamic programming   + Iterative implementation   + Recursive * Tail-recursive |
| 7 | Backtracking, e.g.   * 8 queen * Subset problem, e.g.   + xs = {1, 2, 5, 6, 8}, d = 9   + xs = {3, 5, 6, 7}, d = 15 (see diagram below) * Find all Hamilton circuits of a graph | * Construct the state-space tree   + nodes: partial solutions   + edges: choices in extending partial solutions * Explore the state space tree using depth-first search * “Prune” nonpromising nodes: stop exploring subtrees rooted at nodes that cannot lead to a solution and backtracks to such a node’s parent to continue the search |
| 8 | Branch and bound:   * An enhancement of backtracking * Applicable to optimization problems * Examples:   + Assignment/allocation   + TSP   + Knapsack | For each node (partial solution) of a state-space tree, computes a bound on the value of the objective function for all descendants of the node (extensions of the partial solution). Use the bound for:   * ruling out certain nodes as “nonpromising” to prune the tree – if a node’s bound is not better than the best solution seen so far * guiding the search through state-space |
| 9 | Iterative improvement | Linear programming  Network flow |
| 10 | Space and time trade-off: two varieties | * Union find using an auxiliary array with height of the tree * Input enhancement — preprocess the input (or its part) to store some info to be used later in solving the problem   + counting sorts   + string searching algorithms * Pre-structuring — preprocess the input to make accessing its elements easier   + hashing   + indexing schemes (e.g., B-trees) |