Competitive Programming 3 Textbook

Notes / Attempted Questions

# 1.3 Easy Problems

Super Easy

* **UVa 11172 - Relational Operators \*** (ad hoc, very easy, one liner)
  + Accepted
* **UVa 11498 - Division of Nlogonia \*** (just use if-else statements)
  + Accepted
* **UVa 11727 - Cost Cutting \*** (sort the 3 numbers and get the median)
  + Accepted

Easy

* **UVa 10114 – Loansome Car Buyer \*** (just simulate the process)
  + Accepted
* **UVa 11559 – Event Planning \*** (one linear pass)
  + Accepted
* **UVa 11799 – Horror Dash \*** (one linear scan to find the max value)
  + Accepted

Medium

* **UVa 00573 – The Snail \*** (simulation, beware of boundary cases)
  + Accepted
* **UVa 10141 – Request for Proposal \*** (solvable with one linear scan)
  + Accepted
* **UVa 11507 – Bender B. Rodriguez \*** (simulation, if-else)
  + Accepted

# 1.4 Ad Hoc

Game (Card)

* **UVa 00462 - Bridge Hand Evaluator \*** (simulation; card)
  + Accepted
* **UVa 10646 - What is the Card? \*** (shuffle cards with some rule and  then get certain card)
  + Accepted
* **UVa 12247 - Jollo \*** (interesting card game; simple, but requires good logic to get all test cases correct)
  + Accepted
  + Current solution is a bit of a cop out, uses checking for win rather than explicit if statements with full understanding of cases

Game (Chess)

* **UVa 00278 - Chess \*** (ad hoc, chess, closed form formula exists)
  + Accepted
* **UVa 00696 - How Many Knights \*** (ad hoc, chess)
  + Accepted
* **UVa 10284 - Chessboard in FEN \*** (FEN = Forsyth-Edwards Notation is a standard notation for describing board positions in a chess game)
  + Passes all uDebug cases but gets Wrong Answer

Game (Others)

* **UVa 00489 - Hangman Judge \*** (just do as asked)
  + Accepted
* **UVa 10189 - Minesweeper \*** (simulate Minesweeper, similar to UVa 10279)
  + Accepted

Palindrome

* **UVa 00401 - Palindromes \*** (simple palindrome check)
  + Accepted
* **UVa 11221 - Magic Square Palindrome \*** (we deal with a matrix)
  + Accepted

# 2.2 Linear DS

1D Array Manipulation

* **UVa 10038 - Jolly Jumpers \*** (use 1D boolean flags to check [1..n − 1])
  + Accepted
* **UVa 11340 - Newspaper \*** (DAT; see Hashing in Section 2.3)
  + TLE (Python)
  + Accepted (C++)
* **UVa 12356 - Army Buddies \*** (similar to deletion in doubly linked lists, but we can still use a 1D array for the underlying data structure)
  + TLE (Python)
  + Accepted (C++)

2D Array

* **UVa 10855 - Rotated squares \*** (string array, 90o clockwise rotation)
  + Accepted (C++)
  + TLE (Python)
* **UVa 10920 - Spiral Tap \*** (simulate the process)
  + Accepted (C++)
* **UVa 11581 - Grid Successors \*** (simulate the process)
  + Come back to this later – solution probably involves finding the first duplicate grid as the start of an infinite cycle

C++ STL algorithm

* **UVa 00146 - ID Codes \*** (use next permutation)
  + Accepted
* **UVa 10107 - What is the Median? \*** (find median of a growing/dynamic  list of integers; still solvable with multiple calls of nth element in algorithm)
  + Accepted
* **UVa 10258 - Contest Scoreboard \*** (multi-fields sorting, use sort)
  + Accepted

Bit Manipulation

* **UVa 10264 - The Most Potent Corner \*** (heavy bitmask manipulation)
  + Accepted (Python) – could be made more efficient, almost too slow
* **UVa 11926 - Multitasking \*** (use 1M bitset to check if a slot is free)
  + Accepted (C++)
  + TLE (Python)
* UVa 11933 - Splitting Numbers \* (an exercise for bit manipulation)
  + Accepted

Linked List

* **UVa 11988 - Broken Keyboard ... \*** (rare linked list problem)
  + Accepted (C++)

STL Stack

* **UVa 00514 - Rails \*** (use stack to simulate the process)
  + Accepted
* **UVa 00732 - Anagram by Stack \*** (use stack to simulate the process)
  + TLE (Python)
* **UVa 01062 - Containers \*** (LA 3752, WorldFinals Tokyo07, simulation with stack; maximum answer is 26 stacks; O(n) solution exists)
  + Accepted – solution not very efficient though, could be improved maybe using linked list

STL Queue and Deque

* **UVa 10172 - The Lonesome Cargo ... \*** (use both queue and stack)
  + Accepted – could be written more efficiently though, should redo in C++
* **UVa 10901 - Ferry Loading III \*** (simulation with queue)
  + Accepted – solution very messy & not using proper queue data structure
* **UVa 11034 - Ferry Loading IV \*** (simulation with queue)
  + Accepted – easier than Ferry Loading III

# 2.3 Non-Linear DS

C++ STL map

* **UVa 10226 - Hardwood Species \*** (use hashing for a better performance)
  + Accepted (C++)
  + TLE (Python)
* **UVa 11286 - Conformity \*** (use map to keep track of the frequencies)
  + Accepted
* **UVa 11572 - Unique Snowflakes \*** (use map to record the occurrence index of a certain snowflake size; use this to determine the answer in O(n log n))
  + Accepted

C++ STL set

* **UVa 00978 - Lemmings Battle \*** (simulation, use multiset)
  + Accepted (C++)
  + TLE (Python)
* **UVa 11136 - Hoax or what \*** (use multiset)
  + Accepted (C++)
* **UVa 11849 - CD \*** (use set to pass the time limit, better: use hashing!)
  + Accepted (C++)

C++ STL priority\_queue

* UVa 01203 - Argus \* (LA 3135, Beijing04; use priority queue)
* UVa 10954 - Add All \* (use priority queue, greedy)
* UVa 11995 - I Can Guess ... \* (stack, queue, and priority queue)  Also see the usage of priority queue for topological sorts (see Section 4.2.1), Kruskal’s19 (see Section 4.3.2), Prim’s (see Section 4.3.3), Dijkstra’s (see Section 4.4.3), and the A\* Search algorithms (see Section 8.2.5)

# 3.2 Complete Search

Iterative 1 (Linear Scan)

* **UVa 00927 - Integer Sequence from ... \*** (use sum of arithmetic series)
  + Accepted
* **UVa 01237 - Expert Enough \*** (LA 4142, Jakarta08, input is small)
  + Accepted
* **UVa 10976 - Fractions Again ? \*** (total solutions is asked upfront; there- fore do brute force twice)
  + Accepted (C++)
  + TLE (Python)

Iterative 2 (Two Nested Loops)

* **UVa 01260 - Sales \*** (LA 4843, Daejeon10, check all)
  + Accepted (C++)
  + TLE (Python)
* **UVa 10487 - Closest Sums \*** (sort and then do O(n2) pairings)
  + Accepted (Python) using nlogn method
* **UVa 11242 - Tour de France \*** (plus sorting)
  + Accepted

Iterative 3 (Easier)

* **UVa 00441 - Lotto \*** (6 nested loops)
  + Accepted (Python) – used itertools.combinations() instead of loops
* **UVa 00735 - Dart-a-Mania \*** (3 nested loops, then count)
  + TLE (Python)
  + Accepted (C++)
* **UVa 10102 - The Path in the Colored Field \*** (4 nested loops will do, we do not need BFS; get max of minimum Manhattan distance from a ‘1’ to a ‘3’.)
  + Accepted (C++)

Iterative 3 (Harder)

* **UVa 11236 - Grocery Store \*** (3 nested loops for a, b, c; derive d from a, b, c; check if you have 949 lines of output)
  + Accepted (C++)
  + Python too slow

# 3.4 Greedy Algorithms

Classical

* **UVa 11264 - Coin Collector \*** (coin change variant)
  + Accepted
* **UVa 11389 - The Bus Driver Problem \*** (load balancing)
  + Accepted
* **UVa 12405 - Scarecrow \*** (simpler interval covering problem)
  + Accepted
* UVa 12321 - Gas Station (interval covering)
  + Accepted

Involving Sorting

* **UVa 11100 - The Trip, 2007 \*** 
  + Runtime Error despite passing uDebug, might be broken for Python
* **UVa 11292 - Dragon of Loowater \*** 
  + Accepted
* **UVa 12210 - A Match Making Problem \*** 
  + Accepted
* UVa 10763 - Foreign Exchange
  + Accepted
* UVa 10026 – Shoemaker’s problem
  + Wrong answer, probably due to float comparisons
* UVa 10785 - The Mad Numerologist
  + Accepted

Non-Classical

* **UVa 10656 - Maximum Sum (II) \*** 
  + Accepted
* **UVa 10718 - Bit Mask \***
  + Wrong Answer
* **UVa 11157 - Dynamic Frog \***
  + Accepted
  + Solved by alternating usage of the small rocks
* UVa 11240 - Antimonotonicity
  + Accepted
* UVa 11900 – Boiled eggs
  + Accepted
* UVa 10340 – All in All
  + Accepted

# 3.5 Dynamic Programming

Max 1D Range Sum

* UVa 00507 - Jill Rides Again (standard problem)
  + WA (Python and C++)
* **UVa 00787 - Maximum Sub ... \*** (max 1D range product, be careful with 0, use Java BigInteger, see Section 5.3)
  + Accepted
* **UVa 10684 - The Jackpot \*** (standard problem; easily solvable with the given sample source code)
  + Accepted (C++)
  + Runtime error (Python)
* **UVa 10755 - Garbage Heap \*** (combination of max 2D range sum in two of the three dimensions—see below—and max 1D range sum using Kadane’s algorithm on the third dimension)

Max 2D Range Sum

* **UVa 00108 - Maximum Sum \*** (discussed in this section with sample  source code)
  + TLE (Python)

Longest Increasing Subsequence

* **UVa 00481 - What Goes Up? \*** (use O(n log k) LIS; print solution; see our sample source code)
  + Accepted – possibly dodgy though, fails two uDebug cases and may not have implemented correct selection from multiple answers
* **UVa 11456 - Trainsorting \*** (max(LIS(i) + LDS(i) - 1), ∀i ∈ [0 . . . n-1])
  + Accepted
* **UVa 11790 - Murcia’s Skyline \*** (combination of LIS+LDS, weighted)
  + Accepted (C++)
  + TLE (Python)

# Knapsack

* UVa 00562 - Dividing Coins (use a one dimensional table)
  + Hint about using a 1D table seems bad – 2D table required?
  + Accepted (C++)
  + TLE (Python)
* **UVa 10616 - Divisible Group Sum \*** (input can be -ve, use long long)
  + Decently difficult knapsack-style problem using a 3D table
  + Accepted (Python)
* **UVa 10819 - Trouble of 13-Dots \*** (0-1 knapsack with ‘credit card’ twist!)
  + Difficult problem

# Coin Change

* **UVa 00357 - Let Me Count The Ways \*** (similar to UVa 147/674)
  + Accepted (C++)
  + TLE (Python)
* **UVa 10306 - e-Coins \*** (variant: each coin has two components)
  + Accepted
* **UVa 11517 - Exact Change \*** (a variation to the coin change problem)
  + Accepted (C++)
  + TLE (Python)

# Travelling Salesman Problem (TSP)

* **UVa 00216 - Getting in Line \*** (TSP, still solvable with backtracking)
  + Accepted
  + Solved using exhaustive search
* **UVa 10496 - Collecting Beepers \*** (discussed in this section with sam- ple source code; actually, since n ≤ 11, this problem is still solvable with recursive backtracking and sufficient pruning)
  + Accepted
* **UVa 11284 - Shopping Trip \*** (requires shortest paths pre-processing; TSP variant where we can go home early; we just need to tweak the DP TSP re- currence a bit: at each state, we have one more option: go home early) See more examples in Section 8.4.3 and Section 9.2.

# 5.2 Ad Hoc Mathematics

Simpler

* **UVa 10773 - Back to Intermediate ... \*** (several tricky cases)
  + Accepted
* **UVa 11723 - Numbering Road \*** (simple math)
  + Accepted
* **UVa 11875 - Brick Game \*** (get median of a sorted input)
  + Accepted

Mathematical Simulation (Brute Force), Easier

* **UVa 00382 - Perfection \*** (do trial division)
  + Accepted – solution must deal with arbitrary line breaks
* **UVa 01225 - Digit Counting \*** (LA 3996, Danang07, N is small)
  + Accepted
* **UVa 10346 - Peter’s Smoke \*** (interesting simulation problem)
  + Accepted

Mathematical Simulation (Brute Force), Harder

* **UVa 00616 - Coconuts, Revisited \*** (brute force up to n, get pattern)
  + TLE (Python)
  + Accepted (C++)
* **UVa 11130 - Billiard bounces \*** (use billiard table reflection technique:  mirror the billiard table to the right (and/or top) so that we will only deal  with one straight line instead of bouncing lines)
  + Accepted – using the hint makes this into a super neat O(1) problem
* **UVa 11254 - Consecutive Integers \*** (use sum of arithmetic progres-  sion: n = r × (2 × a + r − 1) or a = (2 × n + r − r2)/(2 × r); as n is given,  2  brute force all values of r from 2n down to 1, stop at the first valid a)
  + Accepted (C++)
  + TLE (Python)