Final Lecture

COMP4128 Programming Challenges

School of Computer Science and Engineering
UNSW Australia

Table of Contents

Congratulations!

- Congratulations on making it to the end! There was a lot of difficult material and the problem sets were non-trivial.
- There were also a fair few rough edges, thanks for sticking through it all!

Remaining Dates

- Assignment due date: 21:00:00 25th November. I don't think I have everyone's groups yet so please send an email if you haven't already.
- Remaining problem sets: 29th November and 1st December and 3rd December. The last one only has 1 problem. Close due dates but last two should be pretty straight forward.
- Finals: 11th December. Hopefully I'll have time confirmed soon. More on this in a bit.
- Bonus Assignment due date: 14th December. Let me know in advance if you are planning to do it.

Remaining Admin Stuff

- Please fill out myexperience: https://myexperience.unsw.edu.au This is particularly valuable since this course is still iterating and I chose a different emphasis from previous years.
- I'd like feedback on any part you have feedback on, e.g:
  - Covered topics. Any topics you particularly disliked or any topics you wished we covered.
  - Lecturing style or content. E.g: Whether you'd like more examples in lectures.
  - Also speed of lectures? I feel I slowed down since midsem, I would like to know if this was an improvement.
  - Problem sets. Difficulty? Interesting?
  - Labs. How can we make them more useful?
  - Lack of Piazza
- If you have already submitted but have more feedback, I'd still really appreciate it. You can anonymously send me a private post on Piazza.

Remaining Admin Stuff

- A practice Contest 2 will be uploaded shortly.
- I have also added up a Tips section onto the website. Let me know if you have anything to contribute. I would particularly be interested in what works for you debugging wise and your debugging advice.

Finals

- 5 hours, around 8 full problems of the same weight, with subtasks.
- Topics wise, covers the entire course.
- Expect problems akin to Contest 1 and most problem set problems. They will use the theory we covered but require some modifications first.

Finals

- I will point out what I think are the 3 easiest. I think these should all be doable. Rest are in arbitrary order (even if I told you what I think the order is, I'm not sure this would be at all helpful!)
- Easiest 3 problems should be around difficulty of the 1st/2nd problems of a problem set. Emphasis of the problems will be on what I consider to be important themes.
- Harder problems may require more observations.

Finals

- Roughly same setup as Contest 1. Submission on DOMJudge, open book with provided code and slides and with access to cppreference.
- I will upload last year's finals some time soon. Note that they covered different topics.

Finals Advice

- Read and attempt the subtasks.
- Read all the problems. Everyone has different strengths and I am poor at judging difficulty. Also remember you don't have to get every problem.
- Remember your basic problem solving techniques.

Finals Advice

- Be prepared for bugs to occur, don't let them throw you off. Try to get some realistic gauge of how much of your time is spent debugging on problem sets, what your most common bugs are and what works for you debugging wise.
- If you feel something is suspicious with your submissions, feel free to send us a clarification. We will hopefully not have issues this time around but the worst that can happen is we confirm you are getting the intended feedback.

Study Advice

- Review problem sets and contest 1. Try to recall what the key parts of solutions were and how you derived them.
- Look through examples in classes, make sure you understand roughly how to do them.
- I've also created a summary of what I think were some of the key parts of each week.

Table of Contents

Summaries Summary

- This is a list of what I think the biggest themes of each week are. So things I've tried to convey and think are important.
- Note: This does **NOT** mean if something isn't on this slide then it won't be on the finals.

- Not much to say here. Just standard problem solving technique stuff, consider if a problem has a greedy observation and if you're stuck try starting from a brute force.
- Just don't code and submit a way too slow brute force and expect AC (do complexity analysis!)
- **Problem Set:** It's worth remembering how to do a recursive brute force like in CD or 8-Queens.

- Basic data structures. Make sure you remember what sets, maps and order statistics trees do.
- They don't just store elements, they also maintain order! Useful things they do:
  - Store elements.
  - Find the first element less than or greater than X.
  - Find number of elements less than X.
  - Find the Kth element in order.
- Keep these in mind, so you are prepared whenever you find yourself needing to answer a similar query or just find yourself ever needing a data structure.

- Cumulative array is also helpful.
- So is union find. But remember the main limitation, they don't do well when you need to support both adds and deletes.
- Will talk about range trees in the notes on Lecture Set 4.

- Pillars: Good example of sweep + range tree. By now hopefully you recognize this is a common problem.
- Ancient Berland Roads: Both the idea of considering a different order (the reverse) as well as how to maintain extra metadata along with union find (e.g: total population)
- Classrooms: Good example of a sweep with set with a greedy observation.
- So are Committee and Farmer's Market from Contest 1.

- Many basic things you will be assumed to know: Representing a graph, representing a tree in particular, DFS, BFS, MST, Dijkstra's, Floyd Warshall.
- For trees specifically, know how to compute LCA and the data structure for doing so.
- The binary composition structure more generally answers path queries on trees without updates.
- Being able to construct and recognize implicit graphs is the most useful skill for graph problems.

- First 4 problems just test your understanding of the basic algorithms.
- MST for every edge: 2 parts, some theory about MSTs (essentially that they are greedily built) and modifying the binary composition data structure to find sum of weights of paths. Both are important ideas.
- Legacy: Not crucial but a nice example of how to build the right graph for a problem.
- Along the same veins, Cloudy Skies and Traveller from Contest 1 are good examples.

- Pretty much the entire lecture is important. At minimum definitely make sure you know how to do basic range/point queries/updates that may involve lazy propagation.
- Make sure you understand specifically how to do range sets and range adds and e.g: how to do range sum queries in combination with these.
- And you know how to do these over trees/subtrees too (this shouldn't be any different from on a line).
- Rest is all important to know too.
- Same with example problems. Make sure in particular you understand Mapping Neptune, it is a standard sweep with a range tree.

- Multiples of 3, The Problem Set and Ada and Mulberry are all basic important problems to know how to do.
- The idea behind On Changing Tree is a good example of breaking up a formula into multiple parts. This is very useful later on, especially in DP 2.
- It is worth noting that it was essentially irrelevant the original problem was on a tree. This is generally true for range trees over trees.
- Points is a good example of searching for a specific criterion.
- Bessie's Journey and Make It Gray from Contest 1 are good practice for range trees over more unusual operations.

- The key here is making sure you know how to do DP problems by YOURSELF. The theory is secondary to this. I presented a method, by no means is it the only method, but you want something that works for you.
- I expect you know how to do tree DP, exponential DP and DP involving data structures.
- You should at least be comfortable with each example except Art, Key, Texture where comfortable means you can see how you'd solve them yourself.
- **Problem Set:** It is important you know how to solve the first 5 problems yourself.

- All of it. I will expect you know how to do a linear sweep and binary search. I will also expect you to figure out when these might be useful.
- You should always consider trying to do a problem in an order. This is the essence of linear sweeping.
- Whenever you are asked to minimize or maximize a value, at least consider binary search. Then think about how fixing a value might simplify the situation.

- Be familiar enough with Dinics that you can copy paste and use it.
- Be familiar with the basic flow constructions (vertex capacities, undirected graphs, multiple sources and sinks, link between bipartite graphs and max flow).
- Know that min cut is equal to max flow. Understand the common constructions (project selection, image segmentation)
- Know how to construct a flow graph to fit a set of constraints (Jumping Frogs, Magic Hours).

Problem Set 6 (Paradigms and Flow)

- Power Transmission and Magic Potion are both good exercises in flow graph construction.
- Magic Ship and Delivery Bears are both good, standard examples where binary search is useful. Make sure you can recognize something like this if it comes up in finals.

- Understand the assumptions that CHT and Divide and Conquer Optimization make.
- Be comfortable enough with the code for both to use them.
- For CHT, understand how we transformed the in-class example into CHT.
- For Divide and Conquer, have some feel for what types of cost functions work with Divide and Conquer.
- Feel free to skip the more gory math details. I've added some overview and Key Takeaways slides within the lecture deck itself.

Problem Set 7 (DP 2 and Flow)

- MobiZone vs VinaGone: Just implement image segmentation. Make sure you understand why the flow network in image segmentation is correct.
- Kalila and Dimmna in the Logging Industry: Once you understand it, just implement CHT.
- Other problems (except DayAndNight) are good examples of transformations of the fundamental examples and algorithms.

- Know how to do all operations modulo a prime.
- Know how to compute binomial coefficient modulo a prime.
- Understand the Combinatorics examples. Especially important: understand how to set up a combinatorial DP.
- Understand how to do exponentiation quickly and the covered examples of why matrix exponentiation is powerful.

- Guardians of the Lunatics is a basic DP optimization example.
- Rest (excluding Yet Another Minimization Problem) are all good examples of fundamental math problems. They all directly correspond to one of the themes of the lecture.