Exam Revision

COMP4128 Programming Challenges

School of Computer Science and Engineering
UNSW Sydney

Term 3, 2023

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

- Problem sets due 7th December
- Diaries due 7th December, submit in Moodle
- Final exam: 7th December, 11am–4pm.
- Consultation: TBA.

Feedback

- This course is still iterating, and your feedback is very valuable to us.
- From student feedback so far, I intend to:
  - improve some diagrams in lectures,
  - replace a couple of the problem set questions, and
  - review the selection of topics, particularly Data Structures II.

myExperience

- Please fill out the myExperience survey at https://myexperience.unsw.edu.au.
- Some aspects you might wish to give feedback on:
  - Topics. What should we add/remove?
  - Lecturing style or content, e.g. pacing, choice of examples.
  - Problem sets. Difficulty? Interesting? Deadlines?
  - Problem diaries. Should we keep doing them?
  - Contests/exam. Difficulty, format, etc.
  - Tut/labs. How can we make them more useful? *New format*
  - Forum. How can we best help you outside of class time?
  - Anything we should include on the Tips page of the website, e.g. debugging advice.

Feedback

- In addition to myExperience, you can also give us feedback by:
  - commenting on the pinned feedback thread on Ed, anonymously if you want, or
  - emailing me.

Final Exam

- 5 hours, 7 problems[1] worth 100 points each, with subtasks.
- All topics except Computational Geometry.
- Expect similar difficulty and style to the contest problems. Problems comparable to a problem set E or F will only appear as the very hardest problem or two in the set, in order to distinguish the top couple of students.
- Problems will use the theory we covered but typically require some modifications first.

[1] might be six, TBC

Final Exam

- Problems will be in approximate order of difficulty of the full problem.
- Average last year was about 313/800; we'll be aiming for higher (at least 350/700), but it's hard to predict.
- Read all the problems! Difficulty is subjective, and everyone has different strengths. Also, hard problems may have easy subtasks.
- Use the scoreboard to see what problems other students have had success with, and prioritise those. This is particularly useful to identify easy subtasks.

Final Exam

- Roughly same setup as the contests - submissions on DOMJudge.
- Open book - make use of lecture slides and code.
- You will have access to your home directory.
- Documentation available at https://cppreference.com only; *not* https://cplusplus.com.
- A practice final exam will be made available shortly.

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.
- Reread your old diaries!
- If you feel something is suspicious with your submissions, feel free to send us a clarification.

Study Advice

- Perhaps the most difficult skill required is to recognise what tools to use for which problem.
- This isn't a concern in the weekly problem sets, but the contests should give you an idea of what to expect.
- This skill is best acquired through experience - practice on Codeforces by doing virtual participation in past rounds or doing assorted problems with *tags off*.

Study Advice

- Review problem sets and contests. Try to recall what the key parts of solutions were and how you derived them.
- Look through examples in lectures and tutorials, 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.

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 exam.

- Always ask first whether brute force is fast enough - complexity analysis.
- It's worth remembering how to do a recursive brute force like in N-Queens.
- If brute force would TLE, instead try to make observations that could form the basis of a greedy algorithm.
- Often want to process in some order, so sorting is useful and leads naturally to linear sweep.

- Binary search is an efficient way to search sorted data - use `lower_bound` or `upper_bound` to avoid bugs.
- We can also use it to search a monotonic function by computing function values on the fly.
- Particularly useful case is discrete binary search, where function values are true/false.
- Whenever you are asked to find the largest/smallest x for which some property holds, at least consider binary search. Consider whether the property is monotonic, and if so, can you test the property quickly for fixed x?

- Basic data structures - vector, stack, queue, heap (pq).
- 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.

- Cumulative array is also helpful - allows sum over range in $O(1)$.
- Union find manages disjoint sets. But remember the main limitation, they don't do well when you need to support both insertion and deletion.
- **PS2:**
  - Classrooms: good example of sweep with set using a greedy observation.
  - 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).

Lectures 6–8 (Dynamic Programming)

- You must be able to DP problems by YOURSELF - the theory is secondary. I presented an iterative method to design an appropriate state and recurrence - not the only method, use whatever thought process 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.
- Also review the week 6 revision lecture for more examples.
- **PS3:**
  - Coloring Trees: good exercise in designing a good state here.
  - Wi-Fi: typical example of DP with a data structure.

Lectures 8–10 (Graph Algorithms)

- Many basic things you will be assumed to know: representing a graph, representing a tree in particular, DFS, DAGs, SCCs, MST.
- DFS is surprisingly useful - DFS tree analysis provides structure, both for directed and undirected graphs
- For trees specifically, know how to compute LCA and the data structure for doing so. The same binary composition structure answers most path queries on trees without updates.
- **PS4:** all problems are instructive.

- Single source:
  - Unweighted? BFS
  - Non-negative weights? Dijkstra
  - Negative weights, detect negative cycles? Bellman-Ford
- All pairs shortest path: Floyd-Warshall.
- The most important skill is recognising and constructing implicit graphs.
- **PS5:** Again all problems but in particular Jzzhu and Cities and President's Path for creative applications of Dijkstra and Floyd-Warshall.

- 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).
- The 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.

- **PS6:**
  - Multiples of 3 and The Problem Set are important basic problems to know how to do.
  - The idea behind On Changing Tree is a good example of breaking up a formula into multiple parts. 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.

Lectures 14–16 (Network Flow)

- Max flow is useful for some optimisation problems which don't permit a greedy or DP solution - clues include lack of useful order.
- Be familiar enough with Dinic's that you can copy paste and use it.
- Be familiar with the basic flow constructions (vertex capacities, undirected graphs, multiple sources and sinks, bipartite matching).
- Know that min cut is equal to max flow - commonly used for assignment problems. 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).

- **PS7:**
  - Power Transmission and Magic Potion are both good exercises in flow graph construction.
  - Delivery Bears is a nice combination of flow with a technique from earlier in the course.
  - Oil Skimming is a non-obvious but very elegant application of flow.

- Know how to do all operations modulo a prime.
- Know how to do primality testing, prime factorisation and finding all divisors both for a single n and for all $n \leq N$.
- 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.

- **PS8:**
  - SETDIFF: often a counting problem is easier if you group the expression in a different way.
  - Math: understand prime factorisation properties.
  - The rest are all good examples of combinatorial DP.