CS 512: Midterm 2 Notes

16:198:512
Instructor: Wes Cowan

1 Chapter 3: Graph Structure and Traversals

This is going back a bit, but I strongly recommend a) paying attention to the things that the TAs have been doing in recitation lately, b) looking at the homework problems and exercises from this section, and c) draw yourself some graphs (directed and undirected) and practice analyzing their structure. What are the structural questions we have asked about graphs (connectedness, strongly connected components, bi connectedness, trees, etc) and how can we use the algorithms discussed (particularly DFS) to analyze these questions? Generate some small examples and *practice.*

2 Chapter 4: Shortest Distances and Binary Heaps

- Know how to implement Dijkstra, the process of the code, and *why* it works. What are the assumptions Dijkstra's requires on the underlying graph? Do some examples as practice. What is the complexity? What are special cases?
- What is the difference in implementation / code between Dijkstra and Bellman-Ford? What are the assumptions for Bellman-Ford on the underlying graph? What is the complexity of Bellman-Ford? What are special cases? Do some examples as practice. Can you generate an example where Bellman-Ford gives the correct distance measurements, but Dijkstra's does not?
- Bellman-Ford can be used to detect negative-weight cycles - what is the complexity of this? How else could you identify and detect cycles, and what are the complexities of this? Give an example of an application where negative-weight cycles might need to be identified and dealt with / removed.
- What are the assumptions of a binary heap? What are the base operations? What are the operations for a *linearly stored* binary heap? What is the complexity of identifying whether or not a given value is in a binary heap? Construct some examples and practice the operations.
- What is the process of identifying shortest paths in a DAG? What is the complexity of this operation? How does knowing that the graph is a DAG save you time and effort?

3 Chapter 5: Greedy-Type Algorithms

- Minimum Spanning Trees: Construct small examples, practice doing Kruskal and Prim. What data structures do you need to organize these two algorithms, and why? What are good implementations for these two structures?
- What is a union-find data structure, and what is it used for? What do the various variants (union by rank, path compression) mean for the overall complexity of using these structures? Construct some examples and practice the union-find operations on them, working out how the underlying data structure evolves and changes with the operations.
- Huffman Encoding - know it, love it, be able to do it. Construct small examples and practice doing Huffman encoding.

- What are the assumptions that go into the mathematical model of language and communication, and how do these assumptions break down in actual language and communication?
- For a string like "PIUWRTPUSPFIGPUSRTPU", what are the frequencies of each symbol? What is the *entropy* for this distribution of symbols? What is the importance of entropy with regards to encoding and Huffman encoding?
- What is the set covering problem? What is the greedy algorithm for the set covering problem? *Is the set cover problem inherently a geometry problem?* Construct small examples and practice the greedy algorithm on them. Why are we interested in the greedy algorithm, as opposed to some kind of optimal or more complex algorithm? What can we say about the performance of the greedy algorithm?
- Can you construct an example where the greedy set cover is not optimal?
- Try to construct your own set cover algorithm. On a few examples, how does it compare to the greedy algorithm?

4 Chapter 6: Dynamic Programming

- What distinguishes a dynamic programming formulation of a problem from a simple recursive formulation of the problem?
- Practice small examples of classic DP problems: computing change, knapsack (with or without repetition), longest common substring, edit distance, etc. What are the complexities of the solutions, and *why?*
- What does the underlying DAG structure of a DP problem mean / represent (why is it a DAG)? Draw out an example of the state / sub-problem graph for an example like Knapsack, with a total capacity of 15, with possible item weights of $\{1, 4, 7\}$. What do the 'values' of these items represent in this DAG?
- In the example above, what are the 'necessary' states to solve the DP, and what are the actual states solved for the general solution? Is there a way to efficiently generate the 'necessary' states to solve and minimize the amount of work done? What would be the complexity of this? Construct small examples and explore.
- What does the optimal solution to the DP problem represent for the underlying DAG? What does the 'greedy' solution to the knapsack problem (or others) represent with respect to the underlying DAG? How do they compare? Construct small examples.
- What is the Floyd-Warshall algorithm, why does it work, and how does it compare to using Bellman-Ford or Dijkstra? What is the complexity of Floyd-Warshall, vs using Bellman-Ford or Dikjstra to solve a related problem? Construct small examples and practice Floyd-Warshall. What is the underlying DAG of the DP approach to Floyd-Warshall?
- Consider the following problem: You have two knapsacks, one with capacity W_1 and the other with capacity W_2. There are two classes of items, Class A with weights and values $\{(w_1^A, v_1^A), (w_2^A, v_2^A), \ldots, (w_n^A, v_n^A)\}$ and Class B with weights and values $\{(w_1^B, v_1^B), (w_2^B, v_2^B), \ldots, (w_n^B, B_n^A)\}$. If you are *not* allowed to put item i from Class A and Class B in the same bag, how can you fit the most value into your two bags? Formulate a dynamic programming solution to this problem. What is its complexity?

5 Chapter 7: Linear Programming

- What is a linear program? What is the 'classical' form of the a linear program? Be comfortable with the matrix or vector notation for LPs.
- Suppose that you have 100 gallons of red paint, 200 gallons of blue paint, and 300 gallons of yellow paint. Brown paint is equal parts red, blue, and yellow mixed together, and sells for $20 per gallon. Green paint, equal parts blue and yellow, sells for $25 per gallon. Orange paint, equal parts yellow and red, sells for $23 per gallon. How much green, orange, and brown paint should you make to maximize your profits? Will you have any paint left over in the end?
- What is the Primal LP and what is the Dual LP? What is the relationship between them? (What are the conditions when their optimal values are the same?) Generate some small examples, and practice writing the Dual from the Primal. When is the Dual easier to solve than the Primal?
- Generate some small examples of LPs in two dimensions. Practicing drawing the feasible solution set. Once you have identified the feasible solution set, how does this help you find the solution to the LP?
- Generate some small DAGs with simple capacities, and practice solving the max flow problem. What does the direct flow solution algorithm 'represent' practically, and why does it work? What is the correspondence between max-flow and min-cut, and why? For some small examples, which is easier to solve, max flow or min cut?
- Consider the problem to maximize $x_1 + x_2$ given that $x_1 \geq 0, x_2 \geq 0$, $x_1 - x_2 \leq 10$ and $3x_1 + 2x_2 \geq 1$. What is the solution? What is the dual? What is the solution to the dual?