COMP3121/9101
Algorithm Design

Tutorial 7
Dynamic Programming

Before the Tutorial

Before coming to the tutorial, try and answer these discussion questions yourself to get a firm understanding of how you're pacing in the course. No solutions to these discussion points will be officially released, but feel free to discuss your thoughts on the forum. Your tutor will give you some comments on your understanding.

- Read through the problem prompts and think about how you would approach the problems.

Tutorial Problems

Problem 1. Houses in the country of DynaProg are positioned on an $m \times n$ grid with the top left house positioned at $(1, 1)$ and the bottom right house positioned at (m, n). You live in the top left house and you need to get to your friend's house which is positioned in the bottom right, but you walk such that you're always making progress towards their house (i.e. you can only walk down or to the right).

(a) Design an $O(mn)$ algorithm that counts the number of ways to get from your house to your friend's house, moving in only two directions.

(b) Find a closed formula for the number of such paths.

> Interpret the solution to part (a) combinatorially.

(c) Now, suppose that there are certain houses you want to avoid passing through. Extend your algorithm in part (a) to find the number of such paths whilst also avoiding such houses.

Problem 2. A *palindrome* is a word that can be read the same both forwards and backwards. For example, the word "kayak" is a palindrome as reading it forwards is the same as reading it backwards. Similarly, "kaayak" is not a palindrome because reading it backwards reads "kayaak" which is not the same as "kaayak". Given a string of length n, design an $O(n^2)$ algorithm that finds the minimum number of characters to delete such that the resulting string after deletion is a palindrome.

For example, "kaayak" only requires one deletion to form a palindrome, while the string "abccab" requires two deletions.

Problem 3. Let $G = (V, E)$ be an unweighted, undirected, and connected graph with n vertices and m edges, and consider two vertices $s, t \in V$.

(a) Design an $O(m)$ algorithm that computes the length of the shortest path from s to u for any vertex $u \in V$, where

we measure the shortest path by the smallest number of edges connecting s and u.

(b) Using the result from the previous part, design an $O(n^2)$ algorithm that computes the number of shortest paths connecting s and t.

- Define your subproblems.
- How do each of these subproblems interact with each other?
- What is your final solution and how should each of these subproblems be solved?

A sharper bound for the same algorithm is actually $O(m)$; for each subproblem, how many subproblems do we *actually* care about?

After the Tutorial

After your allocated tutorial (or after having done the tutorial problems), review the discussion points. Reflect on how your understanding has changed (if at all).

- In your own time, try and attempt some of the practice problems marked [K] for further practice. Attempt the [H] problems once you're comfortable with the [K] problems. All practice problems will contain fully-written solutions.
- If time permits, try and implement one of the algorithms from the tutorial in your preferred language. How would you translate high-level algorithm design to an implementation setting?