Exploring Dynamic Programming Problems Swansea University

Bradley Hunt

May 2017

Contents

Chapter 1

Introduction

Chapter 2

Detailing The Problems

2.1 Minimum Coin Change Problem

The minimum coin change problem is mathematically defined as follows: given an integer N and a set of integers $C\{C_1, C_2, ..., C_n\}$, find the minimum number of values from C which can be added together to make N. More simply it can be defined as: given a total amount and some coins, find the minimum number of coins required to make change for the total amount.
Example 1: First we set the amount A to be 7 and use the set of coins 1,2,5. From these values we can work out that there are 6 possible ways to make change for the amount A. These would be: {1,1,1,1,1,1,1}, {1,1,1,1,1,2}, {1,1,1,2,2}, {1,1,5} and {2,5}. From these values we can see that the minimum number of coins needed to make change for A would be 2.
Example 2: Set the amount A to be 1000 and use the set of coins {50, 300, 600}. Using these values, many possible change combinations can be produced. However the minimum combination would be {600, 300, 50, 50}, which has a length of 4, which will be our output.
A dynamic programming strategy can be applied to solving this problem as it exhibits optimal substructure. For each coin in the given set, there are two options; to select it or not to select it. So we wish to find the solution for both options and then choose the optimal one, which in this case is the minimum among all the obtained solutions. Once the first coin of the set, C_1, has been selected, the sub-problem is to find the minimum number of coins required to make change for amount A - C_1. This process of creating sub-problems is repeated for each coin value in set C up until C_n. A strategy to implement solve this problem using Java would be to use two separate arrays, one to store the optimal solutions for each coin value

and the other to store solutions for a single sub-problem, from which the optimal is chosen and the array is reset, ready for the next sub-problem's solutions to be stored. Finally the optimal solution from the first array is picked and returned.
The minimum coin change problem is useful for implementation within electronic vendors such as vending machines, coffee dispensers or self-checkouts at supermarkets, in order to provide the least amount of change possible to prevent hassle for a customer. This problem could also be applied to compute the possible ways a nine dart finish can be made in a game of darts.

2.2 Longest Increasing Subsequence

The longest increasing subsequence problem is to find an increasing subsequence of maximum length in a given sequence such that all elements of the subsequence are sorted in increasing order. More technically it is defined as: given an array of natural numbers A1,2,...,n, calculate B[1,2,...n] where B[i] $\leq$ B[i+1] and i = 1,2...,n such that n is the maximum.
Example 1: given array A=[3,7,5,6,2] we can find three possible increasing subsequences (that aren't simply lone values). These are {3,7}, {5,6} and {3,5,6}. Clearly {3,5,6} is the longest increasing subsequence which has a length of 3.
Example 2: given array B=[1,9,3,7,25,30,22,27,34,50,44,48,75,21] we can find many increasing subsequences, but the longest increasing subsequence is of length 9 and can be found in two different subsequences; {1,3,7,22,27,34,44,48,75} and {1,3,7,25,30,34,44,48,75}.
An application of the longest increasing subsequence problem is a specific diffing algorithm called patience diff. This algorithm is an advantageous extension of the regular diff algorithm. It was created by Bram Cohen, who is best known for creating the BitTorrent peer to peer network. The patience diff algorithm allows for differences to be computed to a greater accuracy. It works by computing the longest increasing subsequence, finding the longest matching sequence of lines and then recurses the algorithm over each range of lines between the already matched lines.

Chapter 3

Summary