Robert Ottolia

Andrew Brown

Luke Brewbaker

Stephen Krueger

CS 325 Winter 2015

**Project 2: Coin Change**

**1. Filling in the table using the Dynamic Programming approach.**

For our dynamic programming algorithm to solve the coin change problem, we compute the solutions to the smaller sub-problems first in a bottom up manner in a table. Based on the results in the table, then, the solution to the ‘top’ (original) problem is then computed. This is a valid way to fill the table because at each step, we are computing the optimal solution to subproblems that in turn can be used to find the optimal solution of the original problem. This technique can be used when the problem exhibits Optimal substructure, which in this, it does.

**2. Pseudocode for each Algorithm**

**2.1 Dynamic Programming Approach**

*coinList* coin denomination input array

*minCoins*  table that will be built bottom-up

*coinsUsed* table that will keep track of coins used

**for** *i* = 0 to length of *total* + 1 **do:**

set table for use case of coin denominations of 1:

*minCoins[0][i]*  *i*

*coinsUsed[0][i]*  *i*

**for** *i = 1* to length of  *coinList* - 1 **do:**

**for** *j = 1* to length of *total + 1* **do:**

**if** *j* < *coinList[i]* **then:**

Current coin to big, get previous best coin total count:

*minCoins[i][j]* *minCoins[i – 1][j]*

**else if** *j >= coinList[i]* **then:**

See if current coin can get us to total with lower count:

*minCoins[i][j]* minimum between *minCoins[i – 1][j]* and *minCoins[i][j – coinList[i]]+1*

Add the coin to coin tracker:

*coinsUsed[i][j]* += 1

**endif**

**endfor**

**endfor**

**return** table position containing minimum # of coins *minCoins[len(coinList)-1][total]]*

**2.2 Greedy Algorithm Approach**

*coinList* coin denomination input array

*n* length of *coinList*

*total* total value desired

*coinCount* 0

*dictionaryCount* An empty dictionary to keep track of coin frequency

**for** *i = 0* to *n* **do:**

Divide total sum by largest denomination (assuming *coinList* sorted):

*temp = total/coinList[i]*

Get remainder of total after division:

*total = total % coinList[i]*

Update coin count:

*coinCount* *coinCount + temp*

*dictionaryCount[coinList[i]]* = *temp*

**endfor**

**return** *coinCount, dictionaryCount*

**2.3 Brute Force Approach**