CS 325 Project 3 Write Up

CS 325 – November 9, 2014

Group 4

Group Members: Emma Paul, Ian Paul, Abdulhalim Bambang

1)

The changedp dynamic programming table is filled by first creating a multi-dimensional array. The number of columns is equal the number of coins in V, or the number of denomination coins. So if V = {1, 5, 10, 25}, the number of rows is 4. The number of columns is one more than the amount C in question that we are trying to solve for, to range from 0 to C. The multi-dimensional array for our algorithm is called A. We also create a multi-dimensional array of the same size, called used.

The rows of the table stand for the denomination value, so if V[0] = 1, the first row of A[][] stands for 1 cent. The columns in A[][] stand for the amount in question. The first row of the table is filled in first (using a for loop) with the amount of V[0] times the column number. If V[0] is equal to 1 cent, then the first row signifies how many 1 cent pennies it would take to make the amount in question (which is the column number). This for loop also sets the first row of the used array to 1, which stands for true.

The table then starts to be filled starting at row 2, which stands for V[1], and column 2. Using a double for loop, the table is filled going down the column. Inside the double for loop, three conditions are considered.

First, if the denomination value is equal to the amount in question (so if V[i] equals the column number, j), then the position in the array is filled with 1, and the same position in used in filled with 1 as well. For example, if we are on the column where j = 5, and we are on the row where V[i] = 5, then a 1 will be placed in that position, signifying that it takes 1 coin of 5 cents to make 5 cents.

The second condition that is considered is if the denomination value V[i] is greater than the amount in question. If it is, then that position in the array A[i][j] is filled with the value from the row above. For example, if we are on the column where j = 5, and V[i] = 10, A[i][j] receives the value of the column above. Also, used gets place with a 0, signifying that the coin in question is not used to make that amount.

The last condition that is considered is if the coin denomination value is less than j. This is where the recursion step occurs. First, a temp variable stores the amount in question, or j, minus the denomination of V[i]. So if V[i] = 5, and the amount in question (ie the column number, ie j) is 9, temp = 9 – 5, or 4. Knowing that subtracting 5 is the same as using a 5 cent coin, we can fill that position of A[i][j] with 1 plus the number of coins it took to find A[i][temp] (1 coin using the 5 cent coin, plus the number of coins it took to find 4 cents, which was 2, so 3 coins total). Used[i][j] is also filled with 1, signifying that that coin was used. One condition is considered last. If it is found that it takes 4 coins for A[i][j], but it only took 3 coins to find A[i-1][j], then A[i][j] is filled with how many it took for the row above. Used[i][j] is also filled with 0, signifying the coin of denomination V[i] was not used. This last condition that is considers assures that the algorithm will find the least amount of coins to make the amount in question.

2)

Pseudocode for changegreedy:

```
int C = changed asked to makes
int V[] = array of denomination values //ex: V[] = {1, 5, 25, 50}
int O[] = array to store number of coins of each denomination value used
int size = number of denomination values in V – 1 (because 0-size equals number of denom values)

for(i = size; i>=0; i—) //starting at the highest denomination of V to the lowest
    while V[i] <= C //while the denomination value is less than or equal to the cost
        C = C – V[i] //subtract the denomination value from C
        O[i] += 1 //add 1 to the number of coins used for that denomination value
```

Pseudocode for changedp:

```
int C = changed asked to makes
int V[] = array of denomination values //ex: V[] = {1, 5, 25, 50}
int O[] = array to store number of coins of each denomination value used
int size = number of elements in V //if  V[] = {1, 5, 25, 50}, size = 4
int numCols = C+1//so that the table is ranging from values 0-40.
int A[size][C+1] //multi-dimensional array to create dynamic programming table
int used[size][C+1] //multi-dimensional array to store if coins are used or not

for(j = 0; j < numCols; j++) //fill the first row of the table with the number of coins it would take
    A[i][j] = j * V[0] //using the first denom value only.
```

```
for(j = 1; j < numCols; j++) //j stands for the columns
    for ( i= 1; i< size ; i++) //i stands for the rows, so filling each column iterating through the rows
        if V[i] == j //if the denomination value is equal to the column amount (the C in question)
            A[i][j] = 1//one coin of that denomination is used
            used[i][j] = 1//set true, that coin is used, in table used

        else if V[i] > j //if the denomination value is greater than the amount in question
            A[i][j] = A[i-1][j] // the position receives the amount of coins used in row above
            used[i][j] = 0//set false, that coin is not used

        else // V[i] < j, the denomination value is less than the amount in question
            temp = j – V[i] //temp is amount (j) minus denomination value
            A[i][j] = 1 + A[i][temp] //position is filled recursively
                //1 coin is used (V[i]), plus the number of coins used to find j – V[i]

            used[i][j] = 1; //set true, that coin V[i] is used

            if  A[i][j] > A[i-1][j] //if the number of coins in position is more than row above
                A[i][j] = A[i-1][j] //the position is set with the number of coins from
                        //the row above
                used[i][j] = 0 //set false, that coin is not used because row above
                        //used less coins
```

3)

Prove that the dynamic programming approach is correct by induction. That is, prove that T[v] = min_(i:V[i]≤v) {T[v − V [i]] + 1}, T[0] = 0 is the minimum number of coins possible to make change for value v.

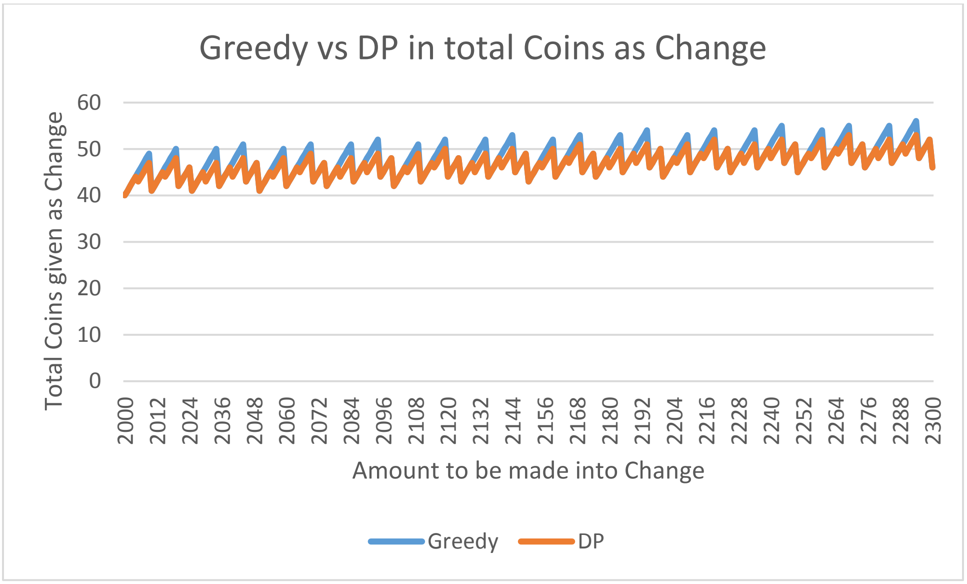
Base Case: T[0] = 0.

Inductive Case:

Assume the dynamic programming algorithm approach always gives minimum number of coins for given amount of change and possible denominations.

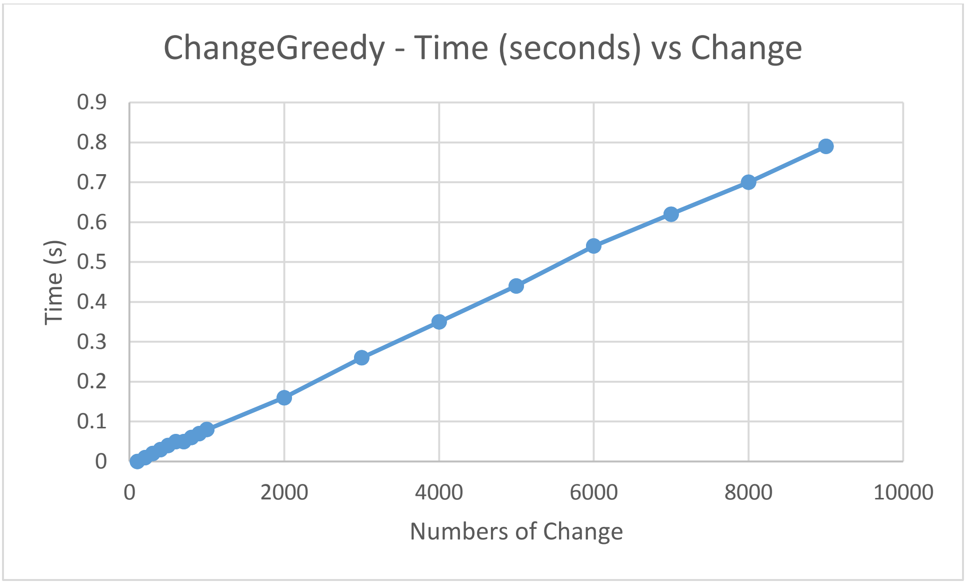
The best solution in making change of v cents, there exists some first coin V[i] where V[i] is less than v. The remaining coins in the optimal solution must be making change for v - V[i] cents. For example, if V[i] is the first coin in an optimal solution to v cents then T[v - V[i]] + 1 will optimally make change for v cents. It will check for all possibilities such that V[i] < v. The value of optimal solution must be the minimum value of T[v - V[i]] + 1.

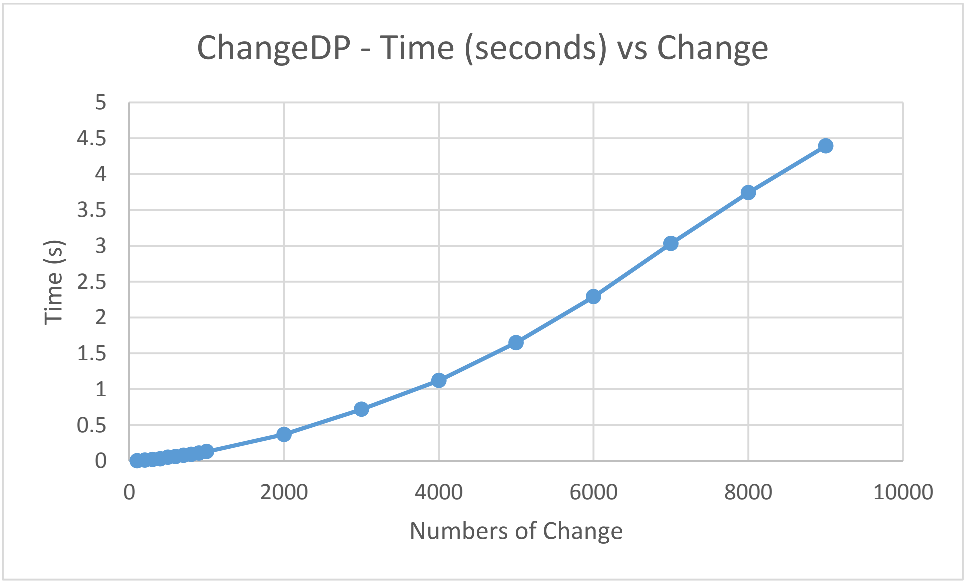
4)



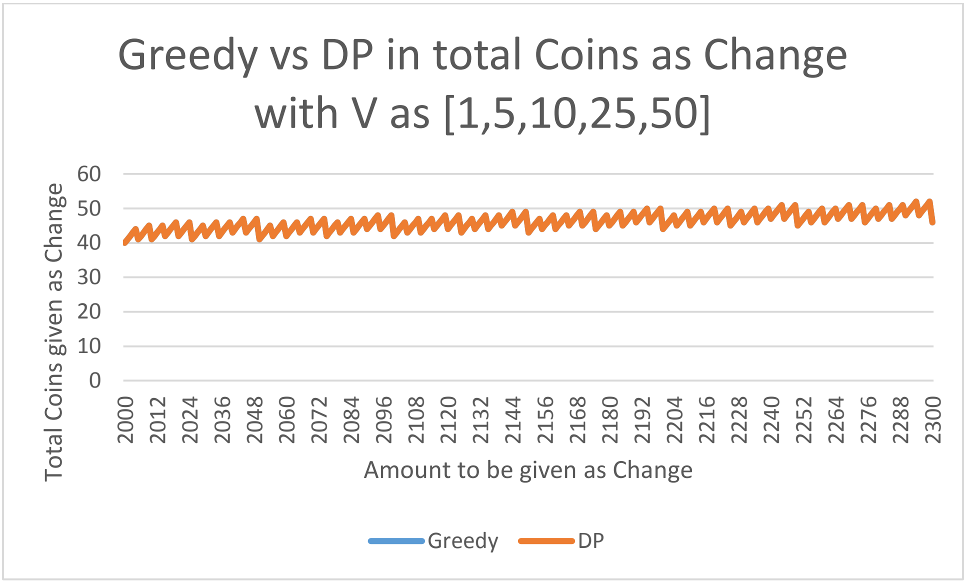
The two approaches behave similarly in the sense that they both increase and then drop down. The greedy algorithm usually gives more coins back as change than the dp algorithm, yet there are times when they both give back the same total of coins in change.

5)


ChangeGreedy - Time (seconds) vs Change
Time (s)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2000
4000
6000
8000
10000
Numbers of Change


ChangeDP - Time (seconds) vs Change
Time (s)
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
2000
4000
6000
8000
10000
Numbers of Change

6)



The line for the Greedy algorithm is hidden behind the DP algorithm because the amounts given are the exact same. Both algorithms return the same coins for the amount given to make change.

7)

In a country where coins have are a value to a power, for examples 2 (V = [1,2,4,8,16])or 7 (V = [1,7,49,343]), both algorithms give the same values as change. One way we derived the answer is that we inserted a set V and computed values from 2000-2300. The reason why they find the same coins to be given as change is because doing the power of a number for the values of V is the same as doing a multiple of that number. The reason the dp algorithm found less coins with V = [1,10,25,50] was because a combination of coins could make up for the 5 cent coin missing which otherwise forced 5 pennies to be utilized (in the greedy algorithm). By doing powers, all multiples are covered, so the greedy algorithm gives the same change values (and thus same amount of coins) as the dp algorithm would.

8)

If the set of coins had a lowest denomination value of 2, that is for example if V[] = {2, 5, 10}, and the amount in question, C, was an odd number, the greedy algorithm could not be able to find a solution, or would fail, because it wouldn't be able to find the correct amount of coins. For example, if C was equal to 13, it would first do 13-10, then 3-2, and then it would fail because it wouldn't be able to do 1 minus 2, which would give -1 rather than 0. Essentially, depending on V, on certain odd numbers of C, the changegreedy algorithm would fail if the lowest coin denomination possible was even, or all the numbers if V was even, because it wouldn't be able to solve for odd numbers. On cases where the changegreedy algorithm would get stuck, changedp would also get stuck, because it wouldn't be able to solve for the odd number either. Therefore, there is no set of coins such that the greedy algorithm will get suck and not be able to find a set of coins whose value adds up to the required amount C, but the dynamic programming will not get stuck. Both would get stuck on certain cases where C is an odd number, and the lowest value of C is an even number.

Sources: Project 3 Links

http://www.programiz.com/c-programming/c-file-input-output

http://stackoverflow.com/questions/3061135/can-we-write-an-eof-character-ourselves

http://stackoverflow.com/questions/19596361/c-passing-char-array-as-parameter-in-function

http://stackoverflow.com/questions/18649547/how-to-find-the-legnth-of-argv-in-c

http://stackoverflow.com/questions/18468229/how-to-concat-two-char-string-in-c-program

http://www.eskimo.com/~scs/cclass/int/sx9a.html

http://www.mathcs.emory.edu/~cheung/Courses/323/Syllabus/DynProg/money-change.html

http://stackoverflow.com/questions/19413569/can-i-use-fscanf-to-get-only-digits-from-text-that-contain-chars-and-ints

http://stackoverflow.com/questions/12048576/how-to-find-eof-through-fscanf

http://pubs.opengroup.org/onlinepubs/7908799/xsh/fscanf.html

http://www.cprogramming.com/tutorial/c/lesson14.html http://stackoverflow.com/questions/20378430/reading-numbers-from-a-text-file-into-an-array-in-c

http://www.happybearsoftware.com/implementing-a-dynamic-array.html

http://stackoverflow.com/questions/20378430/reading-numbers-from-a-text-file-into-an-array-in-c

http://stackoverflow.com/questions/13221844/fscanf-and-newline-character

http://stackoverflow.com/questions/8773346/how-to-initialize-a-dynamic-int-array-elements-to-0-in-c

http://stackoverflow.com/questions/20721245/how-fscanf-know-this-is-the-end-of-the-line

http://www.tutorialspoint.com/c_standard_library/c_function_fscanf.htm

http://stackoverflow.com/questions/8019615/strings-and-character-with-printf

http://en.cppreference.com/w/c/io/fscanf

http://www.asciitable.com/index/asciifull.gif

https://github.com/davidmerrick/CS-325/commit/895e2bb580e345d408544bce83c43024d86eaa50#diff-5998e7ec6815b7ce17aac60a7bbe4f13R4

https://github.com/brent-p/CoinChange/blob/master/src/coinchange/CoinChange.java

http://www.geeksforgeeks.org/pass-2d-array-parameter-c/

http://condor.depaul.edu/rjohnson/algorithm/coins.pdf