

CS 325 Project 2: Coin Change

Your report must be typed and submitted online. Each team member's name must be listed as well as any resources used to finish the project.

For this project, you will investigate the coin change problem:

Given coins of denominations (value) $1 = v_1 < v_2 < \dots < v_n$, we wish to make change for an amount A using as few coins as possible. Assume that v_i 's and A are integers. All values of A will have a solution since $v_1 = 1$.

Formally, an algorithm for this problem should take as input:

- An array V where $V[i]$ is the value of the coin of the i^{th} denomination.
- A value A which is the amount of change we are asked to make

The algorithm should return an array C where $C[i]$ is the number of coins of value $V[i]$ to return as change and m the minimum number of coins it took. You must return exact change so

$$\sum_{i=1}^n V[i] \cdot C[i] = A$$

The objective is to minimize the number of coins returned or:

$$\min \sum_{i=1}^n C[i]$$

Implementation:

You may use any language you choose to implement your algorithms. You will implement three algorithms for this problem. The algorithms may be combined in one program or implemented as separate programs. The algorithms are to be based on these ideas:

1. Implement a Brute Force or Divide and Conquer Algorithm:

It is your choice of which algorithm to implement. This implementation will be called **changeslow** since either a brute force or recursive divide and conquer algorithm will be very slow. A brute force algorithm would try all possible combinations of coins that could add to A cents and keep track of the minimum number of coins used. A divide and conquer algorithm can be based on the following idea.

To make change for A cents, start by letting $K = A$

- If there is a K -cent coin, then that one coin is the minimum
- Otherwise, for each value $i < K$,
 - 1) Find the minimum number of coins needed to make i cents
 - 2) Find the minimum number of coins needed to make $K - i$ cents
- Choose the i that minimizes the sum of 1) and 2)

CS 325 Project 2: Coin Change

Another variation of a recursive algorithm would let $MC(K)$ represents the minimum number of coins required to make change of amount K and then the options are:

- Select 1st coin (value = v_1), Now Smaller problem is minimum number of coins required to make change of amount $(K - v_1)$, so call $MC(K - v_1)$
- Select 2nd coin (value = v_2), Now Smaller problem is minimum number of coins required to make change of amount $(K - v_2)$, so call $MC(K - v_2)$
- Likewise to up to n
- Select n th coin (value = v_n), Now Smaller problem is minimum number of coins required to make change of amount $(K - v_n)$, so call $MC(K - v_n)$.

We need to find the minimum number of coins required to make change for K amount. So we will select the minimum of all the smaller problems and add 1 to it because we have select one coin. Now smaller problems will be solved recursively.

2. Greedy Algorithm:

Another approach to coin change problem is the greedy approach. This implementation is called **change greedy**. This is also “naive” since it may not be optimal.

- Use the largest value coin possible.
- Subtract the value of this coin from the amount of change to be made.
- Repeat.

3. Dynamic Programming:

One dynamic programming approach uses table T indexed by values of change $0, 1, 2, \dots, A$ where $T[v]$ is the minimum number of coins needed to make change for v .

$$T[v] = \min_{V[i] \leq v} \{T[v - V[i]] + 1\}$$

We initialize $T[0] = 0$. How do you store and return the number of each type of coin to return? (That is, how do you build $C[i]$?) This implementation is called **changedp**. Note: there are other versions of the DP algorithm you may use one but need to explain in your report.

The execution of the program(s) should be as follows:

- User runs the programs on the command-line, specifying a file ([input filename].txt) in which the first line contains the array V , formatted as 1 5 10 25, denominations in increasing order.
- The next line contains one integer value for which we must make change.

Program output should be written to a file named [input filename]change.txt where [input filename].txt is the input filename, and should be formatted with the name of the algorithm used, the denominations, the change result and the minimum number of coins m , per line. For example

CS 325 Project 2: Coin Change

Amount.txt:

```
1 2 5
10
1 3 7 26
22
```

Amountchange.txt:

```
Algorithm changeslow:
1 2 5
0 0 2
2
1 3 7 26
1 0 3 0
4
```

Testing for correctness. Above all else your algorithm should be correct. You can test your algorithms on the following:

1. Suppose $V = [1, 2, 4, 8]$ and $A = 15$. All algorithms should return $C = [1, 1, 1, 1]$ and $m = 4$.
2. Suppose $V = [1, 3, 7, 12]$ and $A = 29$. The **changegreedy** should return $C = [2, 1, 0, 2]$ with $m = 5$ and **changedp** and **slowchange** should return $C = [0, 1, 2, 1]$ with $m = 4$. The minimum number of coins is four. The greedy algorithm is suboptimal.
3. If A is changed above to 31, all algorithms should return $C = [0, 0, 1, 2]$, with $m = 3$.

Project Report

Your team's report should include the following:

1. Give pseudocode for each algorithm and asymptotic running time for each algorithm.
2. Describe, in words, how you fill in the dynamic programming table in **changedp**. Justify why is this a valid way to fill the table?
3. Suppose $V = [1, 5, 10, 25, 50]$. For each integer value of A in $[2010, 2015, 2020, \dots, 2200]$ determine the number of coins that changegreedy and changedp requires. Attempt to run changeslow however if it takes too long you can select smaller values of A . Plot the **number of coins as a function of A** for each algorithm. How do the approaches compare?
4. Suppose $V_1 = [1, 2, 6, 12, 24, 48, 60]$ and $V_2 = [1, 6, 13, 37, 150]$. For each integer value of A in $[2000, 2001, 2002, \dots, 2200]$ determine the number of coins that changegreedy and changedp requires. If your algorithms run too fast try $[10,000, 10,001, 10,003, \dots, 10,100]$. You can attempt to run changeslow however if it takes too long you can select smaller values of A . Plot the **number of coins as a function of A** for each algorithm. How do the approaches compare?

CS 325 Project 2: Coin Change

5. Suppose $V = [1, 2, 4, 6, 8, 10, 12, \dots, 30]$. For each integer value of A in $[2000, 2001, 2002, \dots, 2200]$ determine the number of coins that `changeGreedy` and `changeDP` requires. Again you can attempt to run `changeSlow` however if it takes too long you can select smaller values of A . Plot the **number of coins as a function of A** for each algorithm.
6. For the above situations, determine (experimentally) the running times of the algorithms by fitting trend lines to the data or analyzing the log-log plot. Plot the **running time as a function of A** . Compare the running times of the different algorithms.
7. Use the data from questions 4-6 and any new data you have generated. Plot running times as a function of number of denominations. (i.e. $V=[1, 10, 25, 50]$ has four different denominations so $n=4$). Does the size of n influence the running times of any of the algorithms?
8. Suppose you are living in a country where coins have values $V = [1, 3, 9, 27]$. How do you think the dynamic programming and greedy approaches would compare? Explain.
9. Give some more examples of denominations sets V for which the greedy method is optimal. Why does the greedy method produce optimal values in these cases?

What to Submit

Your elected submitter must upload

- To TEACH: a ZIP file containing (1) Project Report PDF, (2) README, (3) CODE
- To Canvas: (1) Project Report PDF