جامعة الإمام عبدالرحمن بن فيصل
IMAM ABDULRAHMAN BIN FAISAL UNIVERSITY

Imam Abdulrahman Bin Faisal University
College of Computer Science & Information Technology
Department of Computer Science

Coin changing

CS Year 4, G1

Term project

Dec. 5, 2018

CS 412

Ms. Hussah Albinali

Team members:

#	Name	ID	Role	Sign
1	Tasneem Hamdy dosoqi	2160007430	Leader	
2	Rahaf Saleh Alzahrani	2160006662	Member	
3	Hana Nasser Alkiwany	2160006714	Member	
4	Jumanah adel alsh	2160006615	Member	

Table of Content

1. Introduction 3
2. Background 3
3. Problem Definition 3
4. Algorithms and mathematical analysis 4
    4.1. Algorithm 4
    4.2. Analysis 6
5. Experiment and Results analysis 6
6. Conclusion 8
Definition 9
References 9

1. Introduction

Problems in programming in general has a lot and different kind of problems and solving them needs different kinds of techniques, the good programmer can use all techniques depends on the type of problem, some of commonly techniques are:

- Divide and conquer
- Randomized algorithms
- Greedy algorithms
- Dynamic programming

The main problem will be discussed in this project is coin changing. Coin Change is the problem of finding the number of ways of making changes for a particular number of cents, n, using a given set of denominations $d_1 \ldots d_m$. It is a general case of Integer Partition. [1] in a simple language we can say the idea of coin changing is a given specific amount of cents and the goal is to find the number of ways of changing these cents from giving supply of value coins with considering that order of coins dos not matter.

2. Background

Coin Change can be solved with some techniques like recursion, dynamic programming or greedy algorithm. but the focusing in this project will be on greedy algorithm as it is the technique we select; A greedy algorithm is an algorithmic strategy that makes the best optimal choice at each small stage with the goal of this eventually leading to a globally optimum solution. This means that the algorithm picks the best solution at the moment without regard for consequences. It picks the best immediate output but does not consider the big picture [2][a].

For understand more how the greedy technique works we need to ask ourselves how to decide who is the optimal choice? If we assumed having an objective function is optimized needed (it does not matter if maximized or minimized) at a certain point, choosing the greedy choice in each certain step to ensure that the objective function is indeed optimized, what exactly a greedy algorithm do to handle the problem, it also never goes back and reverses the decision[c].

Greedy algorithm has advantages like:

1. It concerted easy to create a greedy algorithm for problems
2. Run time analyzing is almost much easier then outer algorithm techniques
3. Much faster in execution

But on the other hand, it has also disadvantages:

1. Working much harder to understand correctness issues
2. Even with the correct algorithm, it is hard to prove why it is correct
3. Most greedy algorithms are not correct

Coin change (making-changing) problem has some applications that can be found rather than coin change only, such as:

- computing the ways one can make a nine dart finish in a game of darts: A nine-dart finish is a sport of darts, using only nine darts, the fewest possible, to achieve maximum point.
- computing the maximum possible atomic composition that can be found for a given mass or charge peak in mass spectrometry [3].

3. Problem Definition

In order to define the relationship between the input and output of the algorithm selected, defining inputs and outputs first is needed.

Input:

n stand for number of cents given.

Set of coin denomination.

And in some techniques following is given to be used also:

$q = \textit{floor}\ (n/25)$ quarters. That leaves $n_q = n \bmod 25$ cents to make change.

d = *floor* (n_q/10) dimes. That leaves n_d=n_q mod 10 cents to make change.
k = *floor*(n_d/5) nickels. That leaves n_k= n_d mod 5 cents to make change.
Finally, p= n_k pennies.

Output:

Optimal number of ways of making changes.
The set of coin used to make optimal choice

An equivalent formulation is the following. The problem we wish to solve is making change for n cents. If n = 0, the optimal solution is to give no coins. If n > 0, determine the largest coin whose value is less than or equal to n. Let this coin have value c. Give one such coin, and then recursively solve the sub problem of making change for n− c cents.

4. Algorithms and Mathematical Analysis

This section introduces the algorithm that is used in this research for coin change and the analysis for all possible cases.

4.1. Algorithm

In coin change problem, the idea is, for any given valued amount n, there should be a minimum number of valued coins to change, and it is called the optimal solution for change. The coin changing problem is formally described as following:
Suppose D = {d_1, d_2, d_3, …., d_k} is a finite set of distinct coins denominations. We assume that, each d_i is an integer and d_1> d_2> d_3> ….> $d_{k.}$ making change for n cents using minimum number of these coins. Assuming d_k is equal to 1 so there will be always a solution. [b]

Coin change problem can be solved using different algorithms. Starting with dynamic programming solution, there is several steps to solve the problem described briefly bellow:

<1> Characterize the structure of a coin-change optimal solution. Consider any optimal solution that make change for n cents using coins of denominations d_1, d_2, d_3, …., d_k as described in the formal definition. Then, break that solution into two different pieces along with any coin boundary. C[p] is the minimum number of coins that needed to make change for p cents and based of that:

$$C[p] = 1 + C[p - d_i]$$

Thus, the optimal solution for the coin changing problem is consist of the optimal solutions for the smaller subproblems.

<2> Recursively define the value for the optimal solution. To change p cents, there should exist a first coin d_i where $d_{i \leq}$ p. the remaining coins will be the optimal solution to making change for p – d_i cents. Since the problem is breaking into smaller subproblems as described above and since d_i is the first coin to make change for p cents in the optimal solution, then C[p] = 1 + min {C [p − di]} and C [p − di] will be the optimal solution to making change for p – d_i cents. However, it must be mentioned the constrains that 1 ≤ i ≤ k. furthermore, the optimal solution for changing 0 cents is 0 coins. The following recurrence show that:

$$C[p] = \begin{cases} \infty & \text{If } p < 0 \\ 0 & \text{If } p = 0 \\ 1 + \min\limits_{1 \leq i \leq k} \{C[p - di]\} & \text{If } p \geq 1 \end{cases}$$

<3> Compute the Values of the Optimal Solution in a bottom-up fashion. In the following pseudocode, d refer to array of denomination values, k is the number of denominations, and n is the amount for which change is to be made.

```
CHANGE($d, k, n$)                                   cost   times
1  $C[0] \leftarrow 0$                              c1     1
2  for $p \leftarrow 1$ to $n$                      c2     n
3      $min \leftarrow \infty$                      c3     n-1
4      for $i \leftarrow 1$ to $k$                  c4     $\sum_{k=1}^{n}$ $t_k$
5          if $d[i] \leq p$ then                    c5     $\sum_{k=1}^{n}$ $(t_k-1)$
6              if $1 + C[p - d[i]] < min$ then      c6     $\sum_{k=1}^{n}$ $(t_k-1)$
7                  $min \leftarrow 1 + C[p - d[i]]$ c7     $\sum_{k=1}^{n}$ $(t_k-1)$
8                  $coin \leftarrow i$              c8     $\sum_{k=1}^{n}$ (tk-1)
9      $C[p] \leftarrow min$                        c9     n-1
10     $S[p] \leftarrow coin$                       c10    n-1
11 return $C$ and $S$                               c11    1
```

C[p] will contain the minimum number of coins needed to make a change for p cents, and S[p] will contain the first coin in an optimal solution to make a change for p cents. It is important to mention that, line 1 is handle the base case while line 2-10 are handle the recursive part.

<4> Construct the Optimal Solution. In the following pseudocode, it outputs the optimal set of coins that make change for n cents where S refer to the array computed above, d refer to array of denomination values, and n refer to the amount for which change is to be made [4][5].

```
MAKE-CHANGE($S, d, n$)
1  while $n > 0$
2      Print $S[n]$
3      $n \leftarrow n - d[S[n]]$
```

Another solution for the coin changing problem is the greedy method. The way that the greedy algorithm work is, build up a solution piece by piece and always choose the piece that is optimal at any moment and offers an immediate benefit with no regard for how that choice will affect future choices. In other words, the choice is the best decision for the current stage. At the end, it will get the optimal solution of the complete problem [6].
In the coin changing problem, for a given set of denominations D = {d_1, d_2, d_3, …., d_k} there must be a minimum number of coins for a given amount of money that can be paid. Using the greedy method, the aim is find the largest possible coin with a minimum number of coins. The greedy algorithm will compute the values of the optimal solution in a top down fashion. In the following pseudocode, d refer to array of denomination values, k is the number of denominations, n is the amount for which change is to be made, result will hold number of the optimal solutions at the end, and optimal_solution refer to the set that will holds the optimal solutions at the end [7].

```
Make-Change (n, d)                                  cost   time
1   result ← 0                                      c1     1
2   optimal_solution ← {}                           c2     1
3   while n! = 0                                    c3     $\sum_{k=1}^{n}$ $t_k$
4       for i ← (k -1) downto 0                     c4     $\sum_{k=1}^{n} n$
5           if n ≥ d[i]                             c5     $\sum_{k=1}^{n}(n-1)$
6               n ← n – d[i]                        c6     $\sum_{k=1}^{n}(n-1)$
7               optimal_solution.add() ← d[i]       c7     $\sum_{k=1}^{n}(n-1)$
8               i++                                 c8     $\sum_{k=1}^{n}(n-1)$
9               result++                            c9     $\sum_{k=1}^{n}(n-1)$
10  Print result                                    c10    1
11  Print optimal_solution                          c11    1
```

Through this algorithm it will be clear to us why greedy algorithm is the best one in solving this problem. The idea for finding the optimal solution for the coin problem using greedy algorithm is simple.

Starting from the largest denomination that is less than or equal to the amount for which change is to be made and keep adding denominations while remaining value is greater than 0. In previous code, line 2 initialize the optimal_solution array list as empty. In line 3-9, first find the largest denomination value that is less than or equal to n. then, subtract that denomination value from n and add that denomination value to the result array. Line 3-9 will repeat until n=0 is met. In line 10, the number of optimal denomination that saved in the result variable will be printed and in line 11, the optimal denomination values that saved in the optimal_solution array list will be printed [8].

Solving the coin change problem with greedy algorithm has many advantages. It is easier to implement, and the execution is faster than other algorithms like dynamic programming algorithm. Also, it does not need to much computing resources. For that reasons, this research will focus only on the greedy strategy.

4.2. Analysis

For the dynamic programming [d] solution, for the CHANGE procedure:

- **running time:** θ(nk) due to line 2 and 4 (nested loop) which could become θ(n^2) when n is close to k.
- **Space Requirements:** θ(n) due to the additional space needed (C [ ] and S [ ] arrays).

for the Make-CHANGE procedure:

- **running time:** θ(n) due to the reduce of n in each pass through while loop.
- **space Requirements:** no additional space needed.

Thus, the total running time is θ(nk) and the total space requirement is θ(n).
For the greedy algorithm solution, for the Make-Change procedure:

- **running time:** θ(n).
- **space Requirements:** θ(n) due to the additional space needed (optimal_solution array list).

Based on the previous analysis we found that both dynamic programming and greedy require θ(n) in term of space but notice that, dynamic programming need 2 additional spaces while greedy need only 1 additional space which is much better. Moreover, in term of the running time, the dynamic programming need θ(nk) to run and execute, and in worst cases when n is close or equal to k it will need θ(n^2). For greedy method to run and execute, it will need θ(n) and when comparing that with dynamic programming running time, greedy is faster so it is better.

5. Experiment and Results analysis

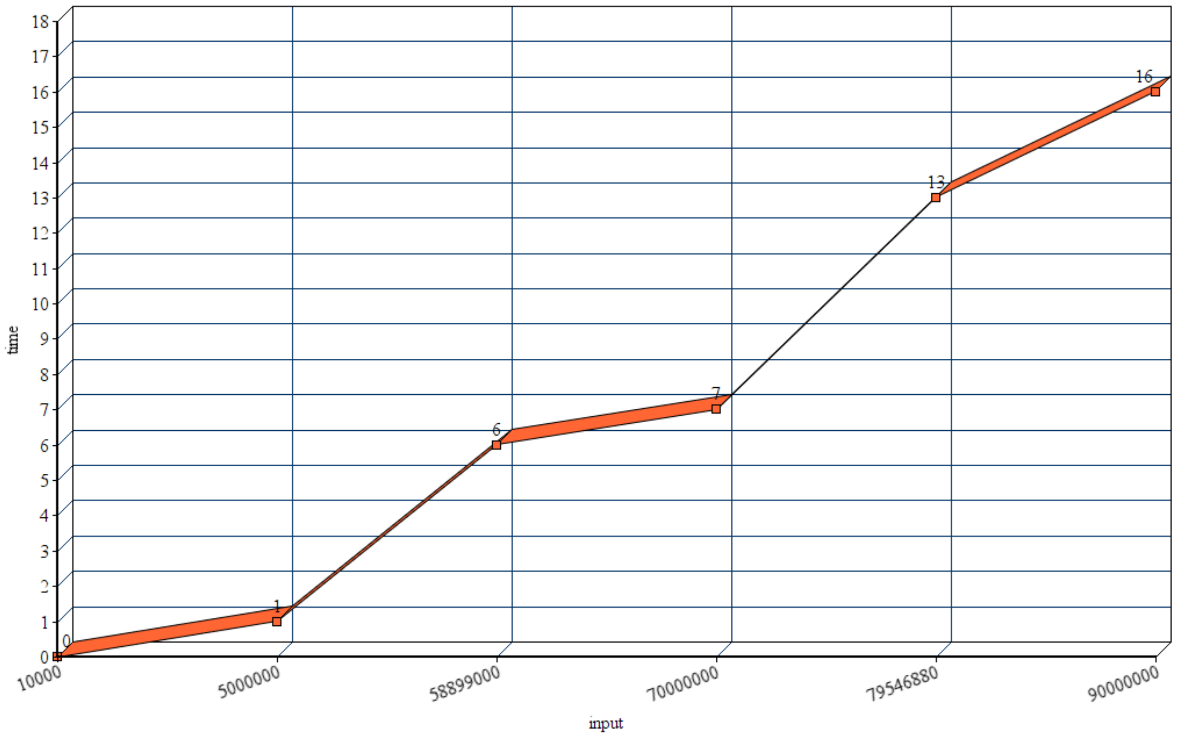
There is a lot of algorithms help us to solve the coin change problem. After comparing the complex time to solve all the algorithms that could solve coin changing problem, it turned out that the greedy algorithm is the best among them in solving this problem.
There are 2 java programs, one for integers and one for fractional values. Starting with integer change program bellow:
The way algorithm was experimented it was by the same number denomination = 6, and the same values of denomination = {1,3,5,10,30,50} and every value of input was tested individually.

input size	starting time (sec)	ending time (sec)	execution time (sec)	theoretical estimate(msec)
10000	0	0	0 = less than 1 sec	1e+7
5000000	0	1	1	$5*10^9$
58899000	0	6	6	$58899*10^6$
70000000	0	7	7	$70*10^9$
79546880	0	13	13	$7954688*10^4$
90000000	0	16	16	$90*10^9$

Execution table summary for integer values

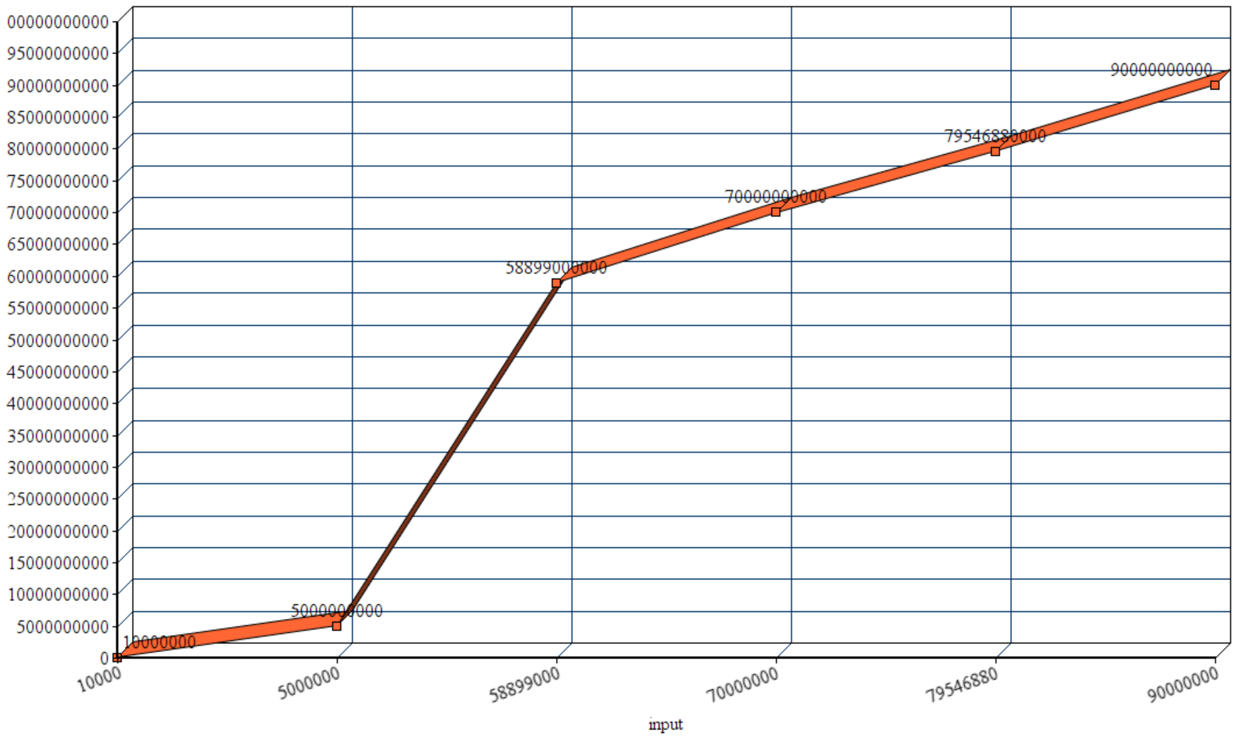
Theoretical estimate: as shown on part 4.2 before the theoretical running time θ(n) which lead to a huge different than the running time in the experiment , even after doing math calculation, to turn it to millisecond, but at end both theoretical and experimental leads to the same point that input and execution time have direct relationship.

And as showing in graph bellow that the direct relationship, if input size increases, execution time also increases, and vice versa. Also, if input size decreases, execution time also decreases and vice versa.



Experimental Complexity Analysis

There is direct relationship between input size and execution time, as increase in input quantity leads to a corresponding increase in the execution time.



Theoretical Complexity Analysis

The huge different between the theoretical and experimental running time analysis because in case of theoretical the analysis was based on best case other than the experimental was based on average case.

With greedy algorithm, there is a disadvantage that the greedy algorithm isn't always optimal. On the other hand, depending on the size of the coins used it is often returns a suboptimal solution that is not optimal solution, but it is a good solution [9]. For example, consider using coins of size 1, 7, and 10. when the greedy algorithm is used to measure 15, it returns five 1s and one 10 which is 6 coins, instead of an optimal solution of two 7s and one 1 which is 3 coins. The greedy algorithm produces an amount c of a suboptimal number of coins, this c is called a counterexample.

To be surer, the algorithm was tested using fractional values. At the end we conclude to that, both theoretical and experimental leads to the same result that input, and execution time have direct relationship which is in fact same as the theoretical and experimental result for integer. This direct relationship shows that as long as the input size increased, execution time will increase as well. This research focus on using greedy algorithm to solve the coin changing problem for integer values.

6. Conclusion

As a conclusion for this research, the greedy algorithm seems a reasonable solution for the coin changing problem although it has suboptimal solutions in some cases. A greedy technique is going with the best and optimal solution for a particular moment until the final result obtained. In term of running time, the greedy approach is much better than a dynamic programing approach because the dynamic programming is slow and it could become θ(n2) when n is close to k. also, in term of space requirement, the greedy approach require 1 additional space only for the final optimal result when compared with dynamic programing approach which need 2 additional space one for the result and one for the number of coins.

Definition

a. Algorithm: collection of rules for solving a problem in a finite number of steps.
b. Optimal solution: favorable and suitable solution.
c. Greedy algorithm: A greedy algorithm is an algorithm that makes the best optimal choice at each partial stage to achieve the globally optimum solution.
d. Running time: duration when the program starts executing until it finishes [12].

References

1. Algorithmist.com. (2018). *Coin Change - Algorithmist*. [online] Available at: http://www.algorithmist.com/index.php/Coin_Change [Accessed 23 Nov. 2018].
2. Techopedia.com. (2018). What is a Greedy Algorithm? - Definition from Techopedia. [online] Available at: https://www.techopedia.com/definition/16931/greedy-algorithm [Accessed 23 Nov. 2018].
3. Change-making problem - Wikipedia. (n.d.). Retrieved December 2, 2018, from https://en.wikipedia.org/wiki/Change-making_problem.
4. The Coin Changing problem. (2018). Retrieved from http://ace.cs.ohiou.edu/~razvan/courses/cs4040/lecture19.pdf
5. Ccs.neu.edu. (2018). Dynamic Programming Solution to the Coin Changing Problem. [online] Available at: http://www.ccs.neu.edu/home/jaa/CS7800.12F/Information/Handouts/dyn_prog.pdf [Accessed 25 Nov. 2018].
6. C# – Coin change problem : Greedy algorithm – Csharp Star. (n.d.). Retrieved November 25, 2018, from https://www.csharpstar.com/csharp-coin-change-problem-greedy-algorithm/.
7. Greedy algorithms. (n.d.). Retrieved November 25, 2018, from https://codility.com/media/train/14-GreedyAlgorithms.pdf.
8. Find minimum number of coins that make a given value . (n.d.). Retrieved November 25, 2018, from https://www.geeksforgeeks.org/find-minimum-number-of-coins-that-make-a-change/.
9. Greedy method Meenakshi Devi. (n.d.). Retrieved November 25, 2018, from https://www.slideshare.net/MeenakshiDevi/greedymethod.
10. What is a Greedy Algorithm? - Definition from Techopedia. (n.d.). Retrieved December 2, 2018, from https://www.techopedia.com/definition/16931/greedy-algorithm.
11. Basics of Greedy Algorithms Tutorials & Notes | Algorithms .. (n.d.). Retrieved December 2, 2018, from https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/.
12. ***Dictionary.com | Meanings and Definitions of Words at Dictionary.com.*** (n.d.). Retrieved December 2, 2018, from https://www.dictionary.com/.