Akoh Atadoga

CS 5120 Project 2A and 2B Report

Change-Making problem is a problem in which a user tries to find the minimum number of coins of a certain denomination that add up to give a certain amount.

In this project, I try to solve the Change-Making problem using 3 different algorithms namely:

1. Straight forward recursive algorithm
2. Greedy algorithm
3. Bottom-up dynamic algorithm

**Data**:

The input data for each of the algorithms are the same. First of all, I used the US based coin system to test the amount of 11, 23, 31, 51, 73, 83, 91 and 99. Then I used a random coin-based system on the amount of 69.

|  |  |  |  |
| --- | --- | --- | --- |
| Row Number | Amount | Number of coins | Coin system |
| 1 | 11 | 4 | 1 5 10 25 |
| 2 | 23 | 4 | 1 5 10 25 |
| 3 | 31 | 4 | 1 5 10 25 |
| 4 | 51 | 4 | 1 5 10 25 |
| 5 | 73 | 4 | 1 5 10 25 |
| 6 | 83 | 4 | 1 5 10 25 |
| 7 | 91 | 4 | 1 5 10 25 |
| 8 | 99 | 4 | 1 5 10 25 |
| 9 | 69 | 5 | 1 5 10 23 25 |

**Results**

1. Straight forward recursive algorithm:

In this algorithm, I use a recursive approach to attempt solving the Change-Making problem. The approach follows the divide and conquer strategy of trying to break the problem into sub problems and solving the sub problems recursive to obtain the solution the problem.

The pseudocode of the Recursive Algorithm is show below:

*RECURSIVE-ALGORITHM (amount)*

*If amount = 0:*

*return 0*

min\_coins = infinity

for coin in coin\_values <= amount:

min\_coins = min (*RECURSIVE-ALGORITHM(amount-coin) + 1, min\_coins)*

*return min\_coins*

The time complexity of the algorithm above is Θ (2n) where n is the amount.

After running the recursive algorithm, the output is shown in table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Row Number | Coin System | Time taken | Amount | Coins used |
| 1 | 1 5 10 25 | 0.012159347534179688 | 11 | 2 |
| 2 | 1 5 10 25 | 1.550912857055664 | 23 | 5 |
| 3 | 1 5 10 25 | 5.835771560668945 | 31 | 3 |
| 4 | 1 5 10 25 | 1783.7140560150146 | 51 | 3 |
| 5 | 1 5 10 25 | 1486962.3069763184 | 73 | 7 |
| 6 | 1 5 10 25 | "it takes more than 30 minutes and so no result got collected". | 83 |  |
| 7 | 1 5 10 25 | "it takes more than 30 minutes and so no result got collected". | 91 |  |
| 8 | 1 5 10 25 | "it takes more than 30 minutes and so no result got collected". | 99 |  |
| 9 | 1 5 10 23 25 | 423310.50515174866 | 69 | 3 |

It can be seen from the table above that recursive dynamic algorithm is very slow, and it gives the correct result all of the time, even in the case of an amount of 69 in Row 9 that uses a weird coin system of 1, 5, 10, 23, and 25. The algorithm is so slow that I was not able to get the execution time for amount 83, 91, and 99 because it took more than 2 hours to run. The slow behavior is caused by overlapping sub problems which it solves over and over again.

1. Greedy algorithm:

In this algorithm, I use a greedy approach that tries to solve the Change-Making problem by first of all sorting the coin denominations in descending order, then coin with the highest value is taken from the amount before other coins with smaller denominations are considered until the change for the amount is gotten.

The pseudocode of the Greedy Algorithm is show below:

*GREEDY-ALGORITHM (coins\_array, amount)*

*coins\_used = 0*

*n = coins\_array.length*

*For i = 0 to n:*

*while amount > 0:*

*coins\_used = floor (amount / coins\_array[i])*

*amount = amound mod coins\_array[i]*

*return coins\_used*

The time complexity of the algorithm above is Θ (n)

After running the greedy algorithm, the output is shown in table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Row Number | Coin System | Time taken | Amount | Coins used |
| 1 | 1 5 10 25 | 0.00691414 | 11 | 2 |
| 2 | 1 5 10 25 | 0.00500679 | 23 | 5 |
| 3 | 1 5 10 25 | 0.00786781 | 31 | 3 |
| 4 | 1 5 10 25 | 0.00596046 | 51 | 3 |
| 5 | 1 5 10 25 | 0.00524521 | 73 | 7 |
| 6 | 1 5 10 25 | 0.00429153 | 83 | 7 |
| 7 | 1 5 10 25 | 0.00715256 | 91 | 6 |
| 8 | 1 5 10 25 | 0.00500679 | 99 | 9 |
| 9 | 1 5 10 23 25 | 0.00691414 | 69 | 8 |

It can be seen from the table above that the Greedy algorithm is very fast however it does not give the correct result all of the time. For example, it gets the correct minimum number of coins used for all the amounts in which the denomination is the US based coin system. However, it does not get the correct minimum number of coins in row number 9 that uses a random coin system of 1, 5, 10, 23, 25 to get the minimum change of an amount of 69. According to greedy algorithm, the coins used is 8 (coins: 25,25,10,5,1,1,1,1) which is false. The correct answer is 3 (coins 23, 23, 23) as gotten from the recursive and bottom-up dynamic algorithm.

1. Bottom-up dynamic algorithm:

In this algorithm, I use a dynamic programming approach that tries to solve the Change-Making problem by considering all the possible minimum values of coins needed to make change for the given amount. After considering all the value it returns the minimum values for the given amount.

The pseudocode of the Bottom-up dynamic Algorithm is show below:

*BOTTOM-UP-DYNAMIC-ALGORITHM (amount):*

*Initialize min\_coins\_array = [0] \* amount*

*for I = 1 to amount:*

*min\_coins\_array[i] = infinity*

*for coin in coin\_values less than <= amount:*

*if min\_coins\_array[i-coin] + 1 < min\_coins\_array[i]:*

*min\_coins\_array[i] = min\_coins\_array[I – coin] + 1*

*return min\_coins\_array[amount]*

The time complexity of the algorithm above is Θ (n . K) where K is amount, due to the nested for loop.

After running the bottom-up dynamic algorithm, the output is shown in table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Row Number | Coin System | Time taken | Amount | Coins used |
| 1 | 1 5 10 25 | 0.0371933 | 11 | 2 |
| 2 | 1 5 10 25 | 0.06604195 | 23 | 5 |
| 3 | 1 5 10 25 | 0.09083748 | 31 | 3 |
| 4 | 1 5 10 25 | 0.13232231 | 51 | 3 |
| 5 | 1 5 10 25 | 0.20980835 | 73 | 7 |
| 6 | 1 5 10 25 | 0.21696091 | 83 | 7 |
| 7 | 1 5 10 25 | 0.30374527 | 91 | 6 |
| 8 | 1 5 10 25 | 0.28300285 | 99 | 9 |
| 9 | 1 5 10 23 25 | 0.22506714 | 69 | 3 |

It can be seen from the table above that the bottom-up dynamic algorithm is very fast, and it gives the correct result all of the time. For example, row number 9 that uses a random coin system of 1, 5, 10, 23, 25 to get the minimum change from an amount of 69. According to bottom-up dynamic algorithm, the minimum coins used is 3 (coins 23, 23, 23) which is entirely correct as opposed to greedy algorithm that gets the correct options some of the times but not all the time.

A graphical representation of the running time of all of the algorithms is shown below:

As from the graph it can also be seen that both the Greedy algorithm and Bottom up dynamic algorithm takes less amount of time while the recursive algorithm takes a huge amount of time.

**Conclusion**

The theoretical analysis and practical analysis of all the algorithms are consistent. No anomalies encountered in the project.

|  |  |  |
| --- | --- | --- |
|  | Theoretical Analysis | Practical Analysis |
| Recursive Algorithm (time) | Slow | Slow |
| Greedy Algorithm (time) | Fast | Fast |
| Bottom-up Dynamic Algorithm (time) | Fast | Fast |
| Recursive Algorithm (correctness) | Always correct | Always correct |
| Greedy Algorithm (correctness) | Not all the time | Not all the time |
| Bottom-up Dynamic Algorithm (correctness) | Always correct | Always correct |

The result of the project conforms to my expectations.