Application of Dynamic Programming to Calculate Denominations Used in Changes Effectively

Kelvin Valensius (13511009)
Computer Science
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13511009@std.stei.itb.ac.id

***Abstract*— Nowadays, transaction has been a part of everyone's life. In this paper, the author will discuss about how to calculate denominations used in changes effectively. The author will compare many approaches to solve this problem. There are 3 approaches that will be discussed in this paper. The first is brute force approach. The second is greedy approach. And the last one is dynamic programming approach.**

***Index Terms*— algorithm, bottom-up, coin change, dynamic programming**

I. Introduction

Long ago, when people have not had monetary system yet, they purchased items by bargaining what they got with items those they wanted. Farmers would trade their rice or fruits with items those they wanted, and so do other people.

But nowadays, people have already familiar with monetary system. We can purchase things with our money or even with credit or debit card if the seller accepts it. Millions of transactions (buying and selling things) happen all over the world each single day. That is why transactions have become part of our life, which is why the author want to discuss about it.

Fig. 1.1 Denominations of Rupiah. Retrieved from http://www.suarasahabat.com/wp-content/uploads/bentuk-uang.jpg.

In this paper, the author will only focus on transaction with cash. In cash payment there are many denominations that is used, as we can see in Fig. 1.1. From that fact, the question arise, "If I have to pay exactly N Rupiah, can I pay it with my money without getting changes? If so, how is the most effective ways to do so?"

II. Theoretical Basis

A. Brute Force

Brute force, also known as complete search is an approach for solving a problem by iterating every possible search space or just part of it to find the solution. There are two type of brute force, iterative complete search and recursive complete search. Iterative complete search searches search space by iterations and loops. Recursive complete search searches search space by recursive. Both of them have their own usages. But, if there is a problem that can be solved by both of them, it is recommended to solve it using iterative complete search since recursive is quite heavy for large data.

There are many problems that can be solved using this approach, such as N-queen problem. This problem is about how to put N queens in N x N board, so that there is no queen that attacks each other. This problem can be solved using recursive complete search by iterating all possible placement of N queens.

B. Greedy

Greedy approach is used to solve problem by choosing a locally optimal choice in each step with hope of eventually reaching the globally optimal solution. A problem that can be solved using greedy approach must have two properties:

1. It has optimal sub-structures.
   Optimal solution to the problem contains optimal solutions to the sub-problems.
2. It has the greedy property.
   If we make a choice that seems like the best at the moment and proceed to solve the remaining sub-problem, we reach the optimal solution. We will never have to reconsider our previous choices.

There are many problems that can be solved using this approach, such as activity scheduling problem. This problem is about choosing activities so that we can take as many activity as possible without overlapping with the schedule given. The idea is to sort the activities in increasing order of finish time of each activities. Then after that we try to take the activities from the beginning (from the earliest finish time). If that activity is not overlapping with activities that have been taken previously then you can take it. But if it is overlapping, just skip it. It has the greedy property where we will never have to reconsider our previous choice of activity. It also has optimal sub-structures by choosing activity with the earliest finish time. This is why it is a problem that can be solved with greedy approach.

C. Dynamic Programming

The key of dynamic programming is to determine the problem states and to determine the relationships or transitions between current problems and their sub-problems. There are two types of dynamic programming, top-down and bottom-up. Top-down dynamic programming is a bit similar to recursive complete search, but with memorization. Bottom-up dynamic programming is harder to find compared to top-down dynamic programming, but it is easier to code because the code will be similar to iterative complete search.

This is the pros and cons of top-down dynamic programming and bottom-up dynamic programming:

Top-down dynamic programming	Bottom-up dynamic programming
Pros: 1. It is a natural transformation from the normal Complete Search recursion. 2. Computes the sub-problems only when necessary (sometimes this is faster).	Pros: 1. Faster if many sub-problems are revisited as there is no overhead from recursive calls.
Cons: 1. Slower if many sub-problems are revisited due to function call overhead. 2. If there are M states, an O(M) table size is required, it will take lots of space.	Cons: 1. For programmers who are inclined to recursion, this style may not be intuitive. 2. If there are M states, bottom-up dynamic programming visits and fills the value of all these M states.

There are many problems that can be solved using dynamic programming, such as 0/1 knapsack problem. This problem is about compute maximum value of items that we can carry with given maximum weight that we can carry. This problem can be solved by both bottom-up dynamic programming approach and top-down dynamic programming approach.

III. PROBLEM

Imagine, when you purchasing things in supermarket or buying foods in canteen, then you pay for it. After that, when you are waiting for the cashier to give you changes, instead of giving you the changes the cashier say this, “I’m sorry, but we are lack of changes, please wait while we find changes for you.” Imagine if this kind of event often happens. First, the customer will not satisfied with the service. Second, it is wasting time.

So, because it is inconvenient, the author will explain more about it and how to solve it. The cause of that problem is the cashier cannot choose the denominations that will be used in changes effectively. Choosing denominations that will be used effectively can reduce the chance of lack of changes like the problem described above. The next question is “How?” The author will explain the solution in the next section.

IV. Solution

A. Some Approaches

Since we do not know the next customer will need how much changes, we can assume that the most effective way to choose the denominations used in changes is to pick the least number of banknotes and coins used for changes in each transaction. There are some ineffective and inefficient approaches. The author will discuss about those approaches for comparison with the best one for this problem, dynamic programming. Approaches that the author will discuss first is brute force approach and greedy approach.

A.1. Brute Force Approach

To solve this problem, one of the approaches that we can try is by brute force approach. The idea is to try all the combinations of banknotes and coins that the cashier has at that time and pick the one that used the least number of banknotes and coins and of course, have total amount exactly the same as needed for changes.

With this approach, it can be proven to be effective since it is iterating all the possible combinations. To compare with other approaches, the author will try some test cases to each approaches and compare the results in the end.

For example, there are 5 coins, 1 cent, 1 cent, 3 cent, 3 cent, and 4 cent. Then the change that is needed to be given is 6 cents. This is the result table of brute force approach for the cases above:

No	Chosen coins	Total value
1	{}	0
2	{1}	1
3	{1}	1
4	{3}	3
5	{3}	3
6	{4}	4
7	{1,1}	2
8	{1,3}	4
9	{1,3}	4
10	{1,4}	5
11	{1,3}	4
12	{1,3}	4
13	{1,4}	5
14	{3,3}	6
15	{3,4}	7
16	{3,4}	7
17	{1,1,3}	5
18	{1,1,3}	5
19	{1,1,4}	6
20	{1,3,3}	7
21	{1,3,4}	8
22	{1,3,4}	8
23	{1,3,3}	7
24	{1,3,4}	8
25	{1,3,4}	8
26	{3,3,4}	10
27	{1,1,3,3}	8
28	{1,1,3,4}	9
29	{1,1,3,4}	9
30	{1,3,3,4}	11
31	{1,3,3,4}	11
32	{1,1,3,3,4}	12

From the result table above, the solution is {3,3} with using only 2 coins (highlighted above with yellow color). This is the most optimal solution for this cases.

Although this approach is effective, but we have not count the complexity of this approach yet. Because for every banknotes and coins there are only two options, use it or not. Then the complexity is O(2^N) for N banknotes and coins. This approach become inefficient for larger N. For example, if the cashier has 100 banknotes and coins. Then the complexity will be $2^{100} \approx 1.27 \times 10^{30}$. If we assume that 100000000 (10^8) process can be processed in 1 second then this example case will take about 1.27×10^{22} seconds. It will take forever to finish so this approach is not efficient.

This is the pseudo code of brute force approach:

```
begin
        ans = INFINITE;
        for i = 1 to 2^N do
        begin
                sum = 0;
                tot = 0;
                for j = every 1 bit in i
do
                begin
                        sum += arr[j];
                        tot++;
                end
                if (sum == M) then
                        ans = min(ans,tot);
        end
        writeln(ans); // the least
number of coins that is used for
changes with value M
end
```

A.2. Greedy Approach

In this section, the author will discuss about solving the problem above with greedy approach. The idea is to sort in decreasing order of denominations value. After that, we pick from the beginning (from the denomination that has biggest value) as long as it can still fits in the required value of changes. Then after we pick the denomination, we subtract it from the value of changes. We have to repeat this process until the value of changes reduced to zero (there is a solution) or until we have tried all possible denominations (which means there is no solution).

This approach assumes that if we do as described above and found a solution, the solution found will be optimal. To prove the effectiveness of this approach and also to compare with other approaches, the author will try some test cases to this approach too (just like the author did in previous section).

For example, there are 5 coins, 1 cent, 1 cent, 3 cent, 3 cent, and 4 cent. Then the change that is needed to be given is 6 cents. This is the same test case as in brute force approach section. The result of this approach is {4,1,1}. It is not optimal because we can make 6 cents with {3,3}, as described in brute force section.

The key property to make the greedy algorithm works is that each denomination's value is a multiple of the value of the next smaller denomination [2]. The test case above fails because 4 is not multiple of 3, it is contradicts with the statement above.

So, we have known that this approach is not effective. Now, the author will count the complexity of this approach (for comparison purpose). Assume we have N banknotes and coins. Then we will need O(N log N) for sorting them in decreasing order of denomination value and O(N) for finding the solution. So the complexity is O(N log N) + O(N) = O(N log N). This is faster than the brute force approach, but it is not effective for this problem.

This is the pseudo code of greedy approach:

```
begin
        ans = 0;
        sort_descending(arr); // sort
descending by value
        i = 0;
        while (M > 0)
        begin
                if (M >= arr[i])
                begin
                        ans++;
                        M -= arr[i];
                end
                i++;
        end
        writeln(ans); // the least
number of coins that is used for
changes with value M
end
```

B. Dynamic Programming Approach

In this section, the author will discuss about dynamic programming approach that will be used to solve this problem. The idea is to try whether it is good or not to use this banknotes or coins. The author will give test cases to show how the dynamic programming works.

For example, there are 5 coins, 1 cent, 1 cent, 3 cent, 3 cent, and 4 cent. Then the change that is needed to be given is 6 cents. This is the same test case as in brute force approach and greedy approach sections. The step to solve this test case using dynamic programming approach is like this:

1. Initialize dynamic programming table with zero.

Step	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0

2. Try the first 1 cent.

Step	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0

3. Try the second 1 cent.

Step	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	1	2	0	0	0	0

4. Try the first 3 cent.

Step	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	1	2	0	0	0	0
3	0	1	2	1	2	3	0

5. Try the second 3 cent.

Step	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	1	2	0	0	0	0
3	0	1	2	1	2	3	0
4	0	1	2	1	2	3	2

6. Try the 4 cent.

Step	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	1	2	0	0	0	0
3	0	1	2	1	2	3	0
4	0	1	2	1	2	3	2
5	0	1	2	1	1	2	2

From the tables above we can see in row 5 and column 6 that the most optimal solution is only using 2 coins. To generate what coin that is used we have to iterate the coin that is used using parent array of the coins. And the result is the same as brute force approach.

This approach is effective for this problem since it always give the optimal solution. Now, we have to count the complexity of this approach. Assume there are N banknotes and coins and the value of change that is needed is M then the complexity of this dynamic programming approach is O(NM). This is faster than brute force approach and also effective.

V. Conclusion

From all the approaches above to solve this problem, we can conclude that brute force approach is effective but not efficient enough. Greedy approach is not effective for some cases and can lead to wrong solution but it is faster than brute force approach. Dynamic programming approach is the best of this three approaches. It is both effective and efficient to solve this problem. With this, the cashier problem can be minimalized.

References

[1] Halim, S. and Halim, F. (2013). *Competitive Programming 3 The New Lower Bound of Programming Contests*. 69-120.

[2] http://stackoverflow.com/questions/13557979/why-does-the-greedy-coin-change-algorithm-not-work-for-some-coin-sets; Access date : December 20 2013.

Acknowledgment

Kelvin Valensius, as the author of this paper, want to express his deepest gratitude to Dr. Ir. Rinaldi Munir, M.T. and Masayu Leylia Khodra, S.T., M.T. as the lecturers of IF2211 – “Strategi Algoritma”. Special thanks to God, my family, and all my friend that supporting the making of this paper.

Statement

In this statement, I declare that this paper is my own writing, not paraphrasing or translating from other papers, and not plagiarism.

Bandung, 20 December 2013

[illegible]

Kelvin Valensius (13511009)