Find total number of ways to make change using given set of coins

ideserve.co.in/learn/coin-change-problem-number-of-ways-to-make-change

Given an infinite supply of coins of denominations {20,10,5,1}, find out total number of way to make change of given amount - 'n'?
For example, if given amount is 20, there are 10 ways to make this change as shown below -
(1 of 20),(1 of 10 + 2 of 5),(1 of 10 + 1 of 5 + 5 of 1),(1 of 10 + 10 of 1), (2 of 10), (1 of 5 + 15 of 1),(2 of 5 + 10 of 1),(3 of 5 + 5 of 1),(4 of 5),(20 of 1)

If the amount given is 0 then the total number of ways to make change is 1 - using 0 coins of every given denomination.

Algorithm/Insights

Let's try to understand this algorithm using an example. If we are make change of 50 using infinite number of coins of denominations {20,10,5,1} then

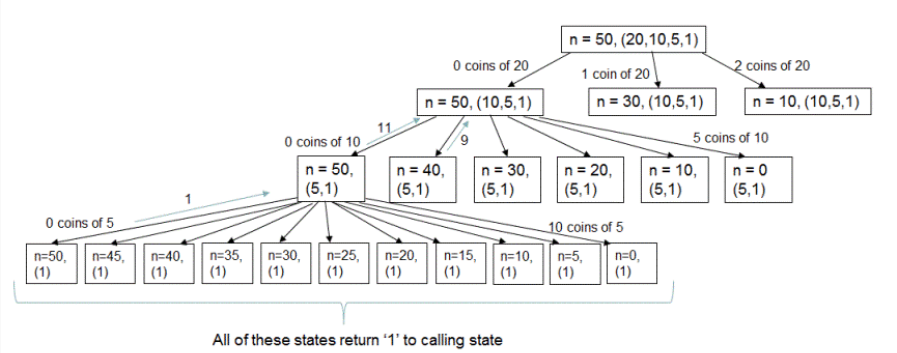
total number of ways to make change of 50 using denominations {20,10,5,1} = total number of ways to make change of 50 using 0 coins of 20 + total number of ways to make change of 50 using 1 coin of 20 + total number of ways to make change of 50 using 2 coins of 20

Now first term on the right hand side of above equation that is - total number of ways to make change of 50 using 0 coins of 20 can be restated as total number of ways to make change of 50 using denominations {10,5,1}

And second term that is total number of ways to make change of 50 using 1 coin of 20 = total number of ways to make change of 30 using denominations {10,5,1}

Similarly, total number of ways to make change of 50 using 2 coins of 20 = total number of ways to make change of 10 using denominations {10,5,1}.

As you can see, this algorithm is recursive in nature and the recursion tree for the above example looks like following. Only one complete path is shown in recursion tree due to space constraint.

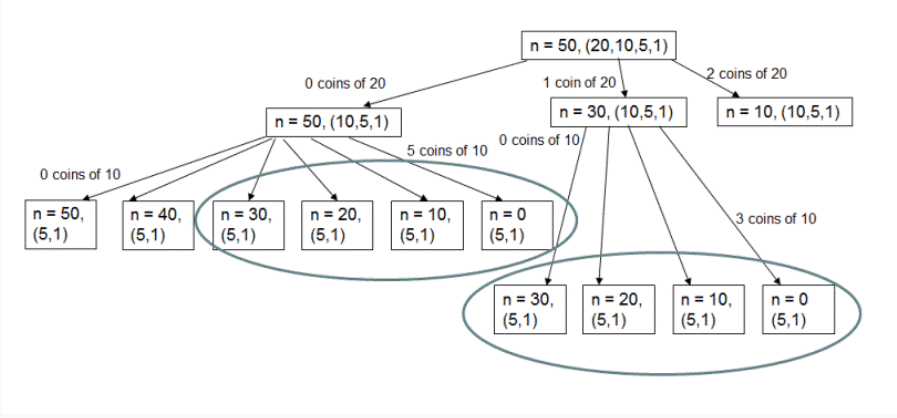


The base case for this algorithm would be when the denomination set has only coins of 1 in it. In that case total number of ways to make change would be 1. Also when amount to make change of is 0, total number of ways to make change would be 1(0 coins of all denominations).

The formal steps of this algorithm are -
1. If the current denomination is 1 then return 1. This is the base case.
2. If current denomination is 20, set next denomination as 10; if current denomination is 10, set next denomination as 5 and if current denomination is 5, set next denomination as 1.
3. Now implement the recurrence relation: numberOfWays(amount, denom) = numberOfWays(amount - 0*denom, nextDenom) + numberOfWays(amount - 1*denom, nextDenom) + ... + numberOfWays(0, nextDenom) using a while loop.

The time complexity of this algorithm is exponential as can be easily observed from recursion tree.

Dynamic Programming - Memoization approach: For the same example, if we look at the recursion tree shown below which highlights the re-computations for the sub-problems of n = 30 and denominations = {5,1}, n = 20 and denominations = {5,1} and so on.



To avoid these re-computations, we could store the results when computed and re-use them if required again. This reduces the time complexity of this algorithm to O(nm) where n is total amount to make change for and m is total number of denominations. For the example shown in the recursion tree n would be 50 and m would be 4. This approach takes extra space of O(nm). Please checkout function countNumberOfWays(int amount, int denom, HashMap numberOfWays) from code snippet for implementation details.

Please add comments below in case you have any feedback/queries.

Algorithm Visualization

```
1  import java.util.HashMap;
2  import java.util.Objects;
3
4  public class WaysOfMakingChange
```

```
5    {
6        class AmountDenom
7        {
8            int amount;
9            int denom;
10
11           public AmountDenom(int amount, int denom)
12           {
13               this.amount = amount;
14               this.denom = denom;
15           }
16
17
18           @Override
19           public int hashCode()
20           {
21
22               return Objects.hash(this.amount, this.denom);
23           }
24
25           @Override
26           public boolean equals(Object obj){
27               if (obj instanceof AmountDenom) {
28                   AmountDenom keyObj = (AmountDenom) obj;
29                   return (keyObj.amount == this.amount && keyObj.denom == this.denom);
30               } else {
31                   return false;
32               }
33           }
34       }
35
36
37
38       public int countNumberOfWays(int amount, int denom, HashMap<AmountDenom, Integer> numberOfWays)
39       {
40
41           if (denom == 1)
42           {
43
44               numberOfWays.put(new AmountDenom(amount, denom), 1);
45               return 1;
46           }
47
48           int nextDenom = 0;
49
50           if (denom == 20)
51           {
52               nextDenom = 10;
53           }
54           else if (denom == 10)
55           {
56               nextDenom = 5;
```

```
57          }
58          else if (denom == 5)
59          {
60              nextDenom = 1;
61          }
62
63
64          int numberOfCoins = 0, ways = 0, modifiedAmount;
65          while ((numberOfCoins*denom) <= amount)
66          {
67              modifiedAmount = amount - (numberOfCoins*denom);
68
69
70              if (numberOfWays.get(new AmountDenom(modifiedAmount, denom)) != null)
71              {
72                  ways += numberOfWays.get(new AmountDenom(modifiedAmount, denom));
73              }
74              else
75              {
76                  ways += countNumberOfWays(modifiedAmount, nextDenom, numberOfWays);
77              }
78              numberOfCoins += 1;
79          }
80
81
82          numberOfWays.put(new AmountDenom(amount, denom), ways);
83          return ways;
84      }
85
86
87
88      public int countNumberOfWays(int amount, int denom)
89      {
90
91          if (denom == 1)
92          {
93              return 1;
94          }
95
96          int nextDenom = 0;
97
98          if (denom == 20)
99          {
100             nextDenom = 10;
101         }
102         else if (denom == 10)
103         {
104             nextDenom = 5;
105         }
106         else if (denom == 5)
107         {
108             nextDenom = 1;
```

```
109         }
110
111
112         int numberOfCoins = 0, ways = 0;
113         while ((numberOfCoins*denom) <= amount)
114         {
115             ways += countNumberOfWays(amount - (numberOfCoins*denom), nextDenom);
116             numberOfCoins += 1;
117         }
118
119         return ways;
120     }
121
122
123     public static void main(String[] args)
124     {
125         WaysOfMakingChange solution = new WaysOfMakingChange();
126
127         int amount = 20;
128         HashMap<AmountDenom, Integer> numberOfWays = new HashMap();
129         System.out.println("Number of ways of making change for 20 using denominations of 20,10,5 and 1 are:\n"  +solution.countNumberOfWays(amount, 20, numberOfWa
130     }
131 }
```

Code Snippet

```
package com.ideserve.questions.nilesh;
```

```
import
java.util.HashMap;

import
java.util.Objects;


public class
WaysOfMakingChange

{

    class
    AmountDenom

    {

        int
        amount;

        int
        denom;


        public AmountDenom(int amount, int
        denom)
```

```
    {

            .amount =
        thisamount;

            .denom =
        thisdenom;

    }



    @Override

    public int
    hashCode()

    {


        return Objects.hash(this.amount,this.denom);

    }


    @Override

    public boolean equals(Object obj)
    {

                          AmountDenom)
        if (objinstanceof {

            AmountDenom keyObj = (AmountDenom)
            obj;

                  (keyObj.amount       .amount && keyObj.denom
            return ==               this==                          this.denom);

        }else {

            return false;

        }

    }

}




public int countNumberOfWays(int amount, int denom, HashMap<AmountDenom, Integer>
numberOfWays)

{


       (denom ==
    if 1)

    {


                            AmountDenom(amount, denom),
        numberOfWays.put(new 1);

        return 1;

    }


    int nextDenom =
    0;


       (denom ==
    if 20)

    {

        nextDenom =
        10;

    }

           (denom ==
    else if 10)

    {
```

```
        nextDenom =
        5;

    }

           (denom ==
    else if 5)

    {

        nextDenom =
        1;

    }



    int numberOfCoins = 0, ways = 0,
    modifiedAmount;

          ((numberOfCoins*denom) <=
    while amount)

    {

        modifiedAmount = amount -
        (numberOfCoins*denom);




                                AmountDenom(modifiedAmount, denom))
        if (numberOfWays.get(new !=                                    null)

        {

            ways +=                       AmountDenom(modifiedAmount,
            numberOfWays.get(         new denom));

        }

        else

        {

            ways += countNumberOfWays(modifiedAmount, nextDenom,
            numberOfWays);

        }

        numberOfCoins +=
        1;

    }



                         AmountDenom(amount, denom),
    numberOfWays.put(new ways);

    return ways;

}



public int countNumberOfWays(int amount, int
denom)

{


       (denom ==
    if 1)

    {

        return 1;

    }


    int nextDenom =
    0;


       (denom ==
    if 20)

    {
```

```
            nextDenom =
            10;

        }

               (denom ==
        else if 10)

        {

            nextDenom =
            5;

        }

               (denom ==
        else if 5)

        {

            nextDenom =
            1;

        }




        int numberOfCoins = 0, ways =
        0;

              ((numberOfCoins*denom) <=
        while amount)

        {

            ways += countNumberOfWays(amount - (numberOfCoins*denom),
            nextDenom);

            numberOfCoins +=
            1;

        }


        return ways;

    }




    public static void main(String[]
    args)

    {

        WaysOfMakingChange solution
        =                            new WaysOfMakingChange();


        int amount =
        20;

        HashMap<AmountDenom, Integer> numberOfWays
        =                                          new HashMap();

                            "Number of ways of making change for 20 using denominations of 20,10,5 and 1
        System.out.println(are:\n"
+solution.countNumberOfWays(amount, 20,
numberOfWays));

    }

}
```

Order of the Algorithm

Time Complexity is O(nm)
Space Complexity is O(nm)
