

CSCI 132:

Basic Data Structures and Algorithms

Recursion (Part 3)

Reese Pearsall
Spring 2023

Announcements

Program 5 due Sunday May 7th

Fill out the course evaluation

Next Wednesday will be a help session for program 5 (no lecture)

Typo Fix: Binary Search w/ Recursion *(there is no while loop)*

```
private static int binary_search(?????????) {  
  
    if(low <= high) {  
        int mid = (low + high) / 2;  
        if(n == array[mid]) {  
            return mid;  
        }  
        else if(n > array[mid]) {  
            return binary_search(?????????);  
        }  
        else {  
            return binary_search(?????????);  
        }  
    }  
    else {  
        return -1;  
    }  
}
```

Change Making Problem

Given a set of coin denominations \mathbf{D} , how can you represent \mathbf{K} cents with the smallest number of coins?

Change Making Problem

Given a set of coin denominations \mathbf{D} , how can you represent \mathbf{K} cents with the smallest number of coins?

$$D = [1, 5, 10, 25]$$

$$K = 37$$

Change Making Problem

Given a set of coin denominations \mathbf{D} , how can you represent \mathbf{K} cents with the smallest number of coins?

$$D = [1, 5, 10, 25]$$

$$K = 37$$

Answer = 4

(Quarter, dime, two pennies)

Change Making Problem

Given a set of coin denominations \mathbf{D} , how can you represent \mathbf{K} cents with the smallest number of coins?

$$D = [1, 5, 10, 25]$$

$$K = 37$$

Answer = 4

(Quarter, dime, two pennies)

Algorithm?

Change Making Problem

Given a set a coin denominations **D**, how can you represent **K** cents with the smallest number of coins?

$$D = [1, 5, 10, 25]$$

$$K = 37$$

Answer = 4

(Quarter, dime, two pennies)

Use as many quarters as possible, then as many dimes as possible, ...

Change Making Problem

Given a set a coin denominations \mathbf{D} , how can you represent \mathbf{K} cents with the smallest number of coins?

$$D = [1, 5, 10, 25]$$

$$K = 37$$

Answer = 4

(Quarter, dime, two pennies)

Use as many quarters as possible, then as many dimes as possible, ...

This is known as the **greedy** approach

Change Making Problem

Given a set a coin denominations \mathbf{D} , how can you represent \mathbf{K} cents with the smallest number of coins?

Greedy Algorithm

$$D = [1, 5, 10, 25]$$

$$K = 37$$

Use as many quarters as possible, then as many dimes as possible, ...

Change Making Problem

Given a set of coin denominations \mathbf{D} , how can you represent \mathbf{K} cents with the smallest number of coins?

Greedy Algorithm

$$D = [1, 5, 10, 18, 25]$$

$$K = 37$$

Use as many quarters as possible, then as many 18 cent pieces as possible, then dimes, ...

What if there were also an 18-cent coin?

Change Making Problem

Given a set of coin denominations \mathbf{D} , how can you represent \mathbf{K} cents with the smallest number of coins?

Greedy Algorithm

$$D = [1, 5, 10, 18, 25]$$

$$K = 37$$

Use as many quarters as possible, then as many 18-cent pieces as possible, then dimes, ...

25, 10, 1, 1 (4 coins)

What if there were also an 18-cent coin?

Change Making Problem

Given a set of coin denominations \mathbf{D} , how can you represent \mathbf{K} cents with the smallest number of coins?

Greedy Algorithm

$$D = [1, 5, 10, 18, 25]$$

$$K = 37$$

Use as many quarters as possible, then as many 18 cent pieces as possible, then dimes, ...

25, 10, 1, 1 (4 coins)

What if there were also an 18-cent coin?

Real Answer = 18, 18, 1 (3 coins)

Change Making Problem

Given a set of coin denominations D , how can you represent K cents with the smallest number of coins?

Greedy Algorithm

$$D = [1, 5, 10, 18, 25]$$

$$K = 37$$

Use as many quarters as possible, then as many 18 cent pieces as possible, then dimes, ...

25, 10, 1, 1 (4 coins)

What if there were also an 18-cent coin?

Real Answer = 18, 18, 1 (3 coins)

Lesson Learned: The Greedy approach works for the United States denominations, but not for a general set of denominations

Change Making Problem

Suppose I tell you that 2 quarters, 1 dime, and 3 pennies are the minimum number of coins needed to make **63 cents**

(We will assume we have the standard US denominations [1, 5, 10, 25] (NO 50 CENT PIECE))

$$25 + 25 + 10 + 1 + 1 + 1 = 63$$



What can you conclude?

Does this provide an answer to any other change making problems?

Change Making Problem

Suppose I tell you that 2 quarters, 1 dime, and 3 pennies are the minimum number of coins needed to make **63 cents**

(We will assume we have the standard US denominations [1, 5, 10, 25] (NO 50 CENT PIECE))



This is the minimum coins needed to make 38 cents

Change Making Problem

Suppose I tell you that 2 quarters, 1 dime, and 3 pennies are the minimum number of coins needed to make **63 cents**

(We will assume we have the standard US denominations [1, 5, 10, 25] (NO 50 CENT PIECE))

$$25 + 25 + 10 + 1 + 1 + 1 = 63$$



This is the minimum coins needed to make 63 cents

Change Making Problem

Suppose I tell you that 2 quarters, 1 dime, and 3 pennies are the minimum number of coins needed to make **63 cents**

(We will assume we have the standard US denominations [1, 5, 10, 25] (NO 50 CENT PIECE))

$$25 + 25 + 10 + 1 + 1 + 1 = 63$$



This is the minimum coins needed to make 63 cents

Change Making Problem

Suppose I tell you that 2 quarters, 1 dime, and 3 pennies are the minimum number of coins needed to make **63 cents**

(We will assume we have the standard US denominations [1, 5, 10, 25] (NO 50 CENT PIECE))

$$25 + 25 + 10 + 1 + 1 + 1 = 63$$



This is the minimum coins needed to make 63 cents

Change Making Problem

Suppose I tell you that 2 quarters, 1 dime, and 3 pennies are the minimum number of coins needed to make **63 cents**

(We will assume we have the standard US denominations [1, 5, 10, 25] (NO 50 CENT PIECE))

$$25 + 25 + 10 + 1 + 1 + 1 = 63$$



This is the minimum coins needed to make 1 cent

Change Making Problem

Suppose I tell you that 2 quarters, 1 dime, and 3 pennies are the minimum number of coins needed to make **63 cents**

(We will assume we have the standard US denominations [1, 5, 10, 25] (NO 50 CENT PIECE))

$$25 + 25 + 10 + 1 + 1 + 1 = 63$$



The solution to the change making problems consists of solutions to smaller change making problems

We can use **recursion** to solve this problem

Change Making Problem

In general, suppose a country has coins with denominations:

$$1 = d_1 < d_2 < \dots < d_k \quad (\text{US coins: } d_1 = 1, d_2 = 5, d_3 = 10, d_4 = 25)$$

Algorithm: To make change for p cents, we are going to figure out change for every value $x < p$. We will build solution for p out of smaller solutions.

Change Making Problem

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

Change Making Problem

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

$$C(p) = 1 + C(p - x).$$

$$C(37) = 1 + C(12)$$

We used one quarter

Now find the minimum number
of coins needed to make 12
cents

Change Making Problem

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

$$C(p) = 1 + C(p - x).$$

$$C(37) = 1 + \underbrace{C(12)}_{\text{We used one dime}}$$
$$C(12) = 1 + C(2)$$

Now find the minimum number of coins needed to make 2 cents

Change Making Problem

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

$$C(p) = 1 + C(p - x).$$

$$C(37) = 1 + C(12) \img alt="dime coin" data-bbox="348 564 398 654"/>$$

$$C(12) = 1 + C(2) \img alt="dime coin" data-bbox="554 660 604 740"/>$$

$$C(2) = 1 + C(1) \img alt="penny coin" data-bbox="756 744 806 824"/>$$

$$C(1) = 1 + C(0) \img alt="penny coin" data-bbox="956 854 1000 925"/>$$

Change Making Problem

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

$$C(p) = 1 + C(p - x).$$

$$C(37) = 1 + C(12) \img alt="Dime coin" data-bbox="348 564 398 654"/>$$

$$C(12) = 1 + C(2) \img alt="Nickel coin" data-bbox="556 660 602 740"/>$$

$$C(2) = 1 + C(1) \img alt="Penny coin" data-bbox="756 744 802 825"/>$$

$$C(1) = 1$$

Change Making Problem

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

$$C(p) = 1 + C(p - x).$$

$$C(37) = 1 + C(12) \img alt="A US dime coin" data-bbox="348 564 398 654"/>$$

$$C(12) = 1 + C(2) \img alt="A US dime coin" data-bbox="556 661 603 741"/>$$

$$C(2) = 1 + 1$$

Change Making Problem

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

$$C(p) = 1 + C(p - x).$$

$$C(37) = 1 + C(12) \img alt="dime coin" data-bbox="348 564 398 653"/>$$

$$C(12) = 1 + C(2) \img alt="dime coin" data-bbox="556 660 603 741"/>$$

$$C(2) = 2$$

Change Making Problem

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

$$C(p) = 1 + C(p - x).$$

$$C(37) = 1 + C(12) \img alt="A US dime coin" data-bbox="348 563 398 653"/>$$

$$C(12) = 1 + 2 \img alt="A US dime coin" data-bbox="555 658 603 741"/>$$

Change Making Problem

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

$$C(p) = 1 + C(p - x).$$

$$C(37) = 1 + C(12)$$
$$C(12) = 3$$



Change Making Problem

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

$$C(p) = 1 + C(p - x).$$

$$C(37) = 1 + 3$$



Change Making Problem

$C(p)$ – minimum number of coins to make p cents.

x – value (e.g. \$0.25) of a coin used in the optimal solution.

$$C(p) = 1 + C(p - x).$$

$$C(37) = 4$$

The minimum number of coins needed to make 37 cents is 4

Change Making Problem

In general, suppose a country has coins with denominations:

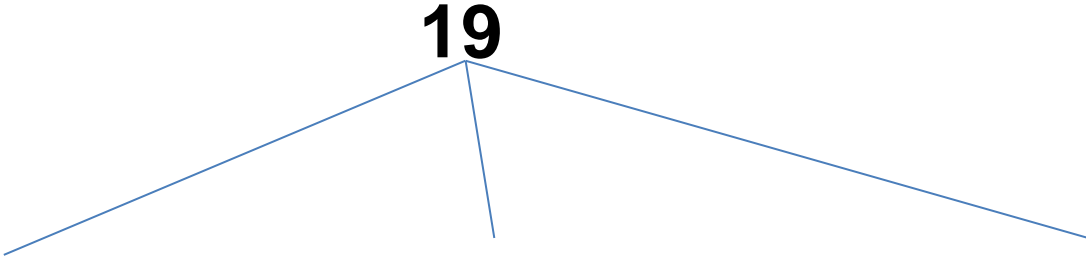
$$1 = d_1 < d_2 < \dots < d_k \quad (\text{US coins: } d_1 = 1, d_2 = 5, d_3 = 10, d_4 = 25)$$

(This algorithm must work for ALL denominations)

Algorithm: To make change for p cents, we are going to figure out change for every value $x < p$. We will build solution for p out of smaller solutions.

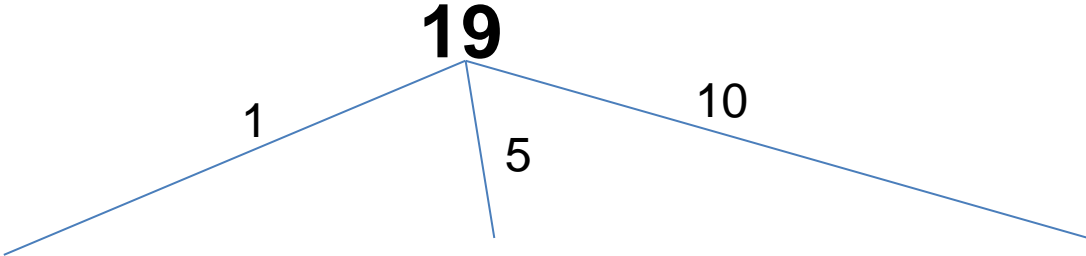
Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10



Change Making Problem

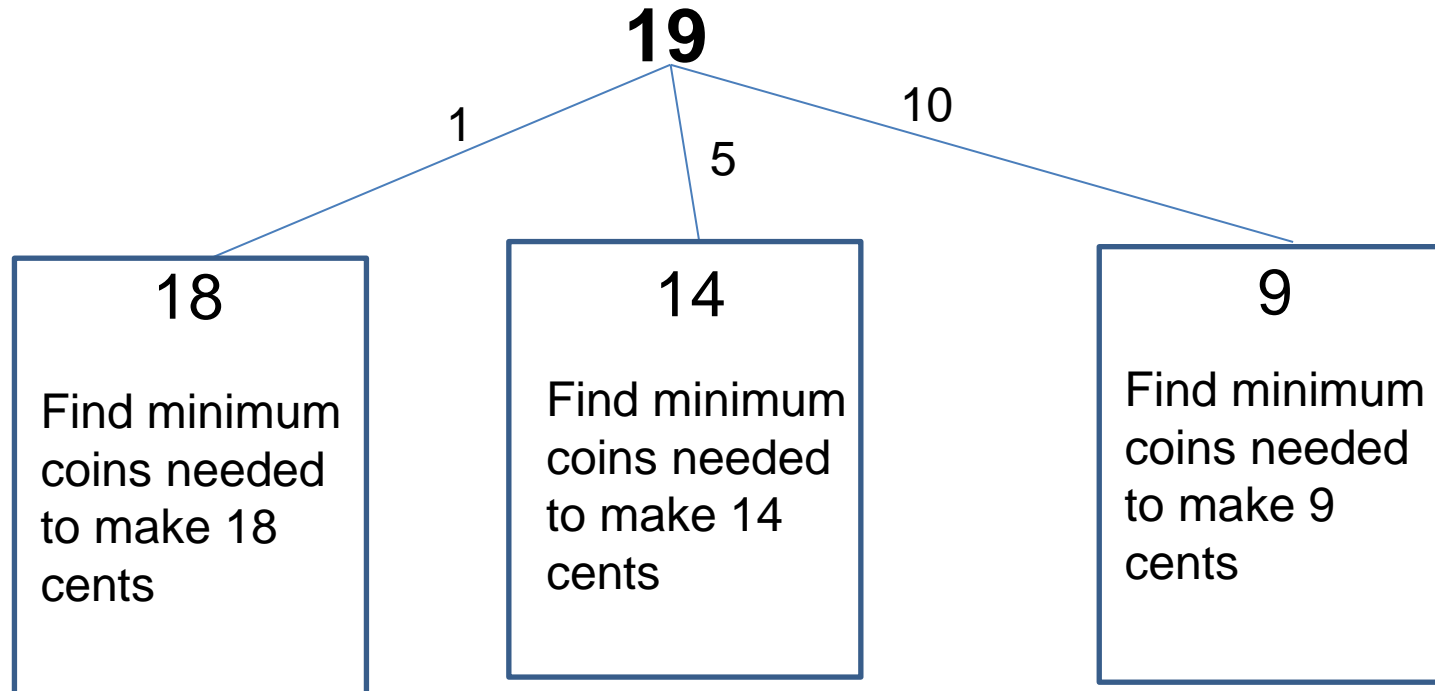
Make \$0.19 with \$0.01, \$0.05, \$0.10



Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

$k = \#$ denominations

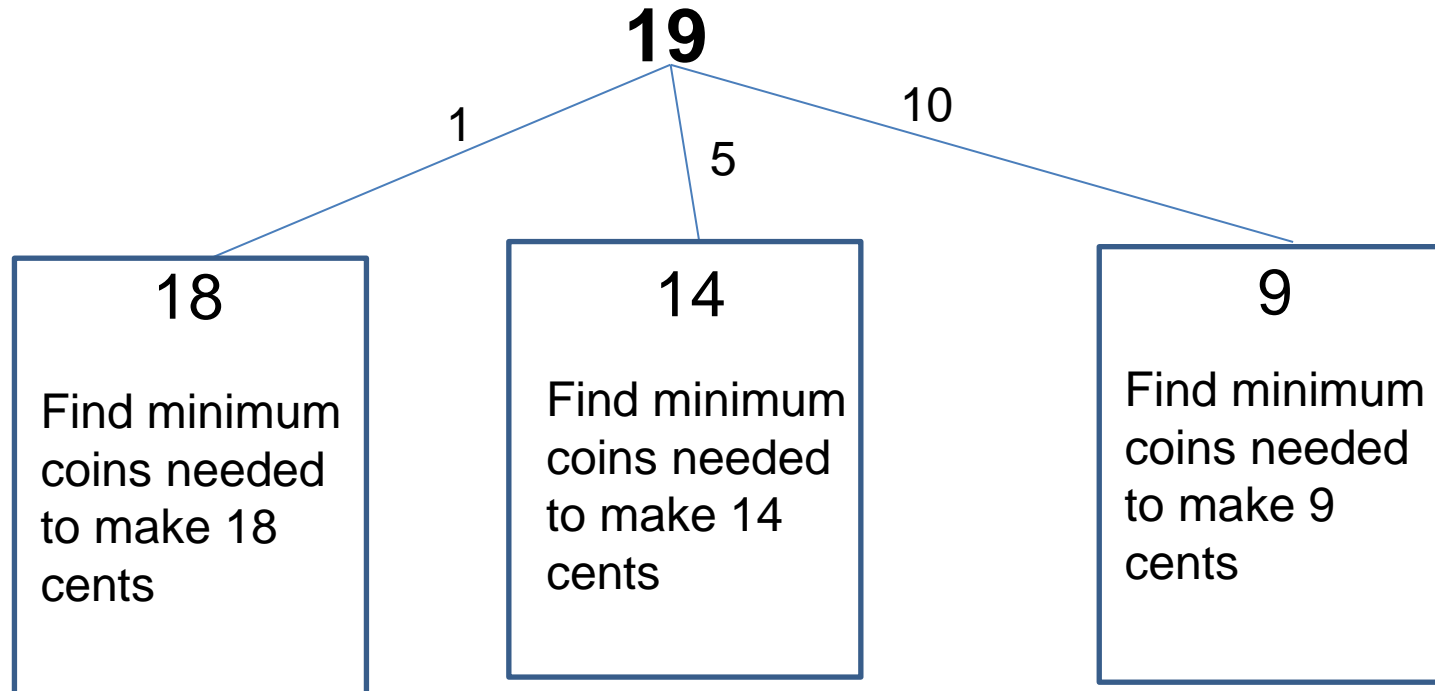


To find the minimum number of coins needed to create 19 cents, we generate k subproblems

Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

k = # denominations

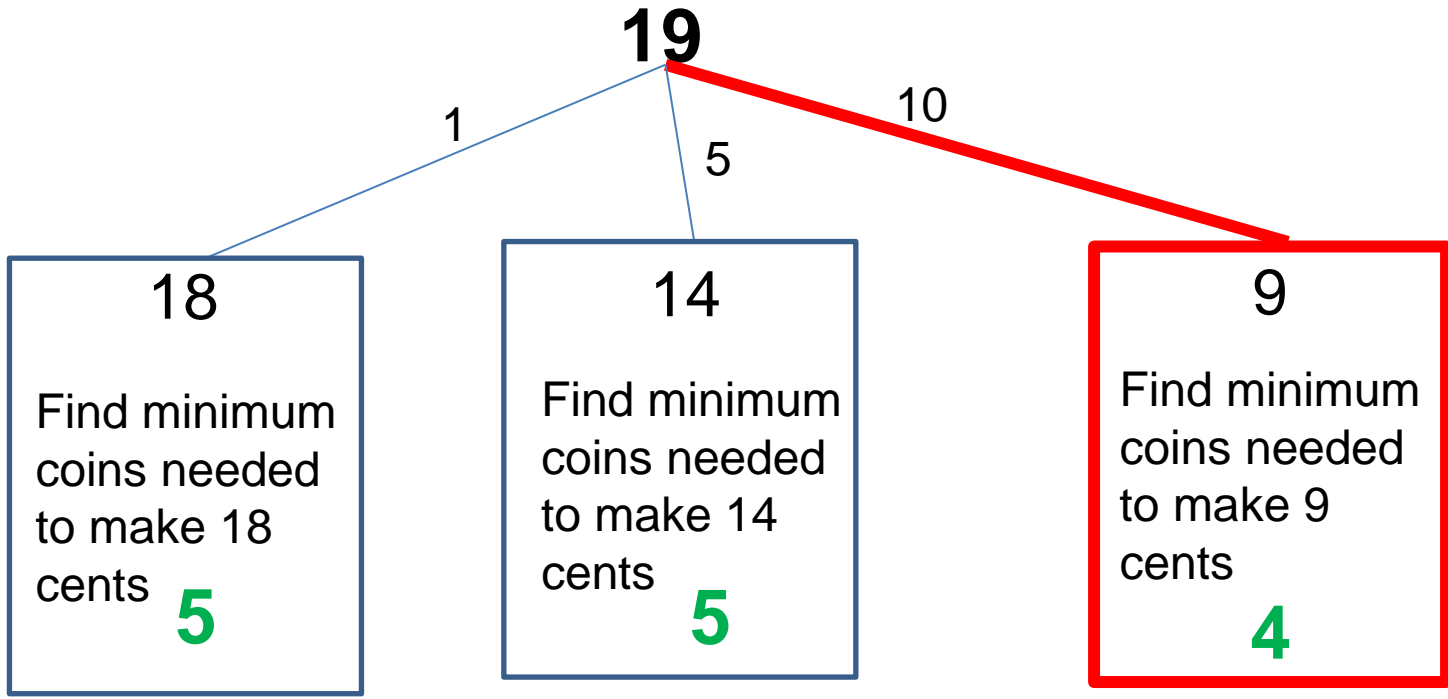


We want to select the **minimum** solution of these three subproblems

Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

k = # denominations

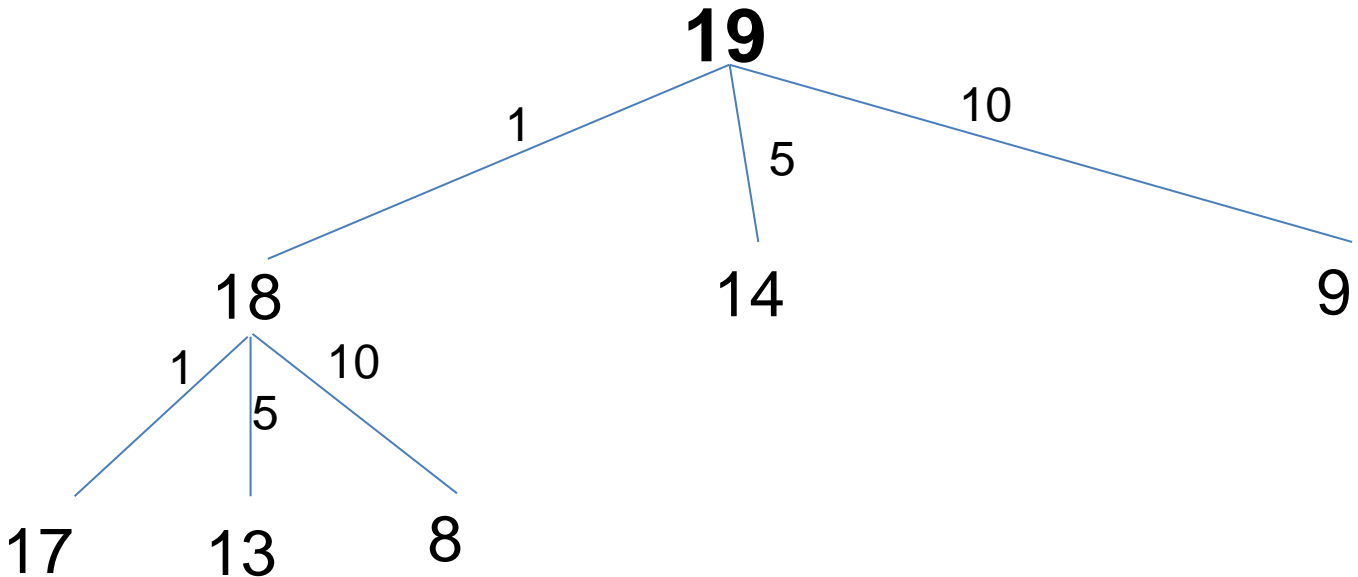


For the solution of our original problem (19), we want to select this branch (one dime used)

Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

k = # denominations



Find minimum coins needed to make 17 cents

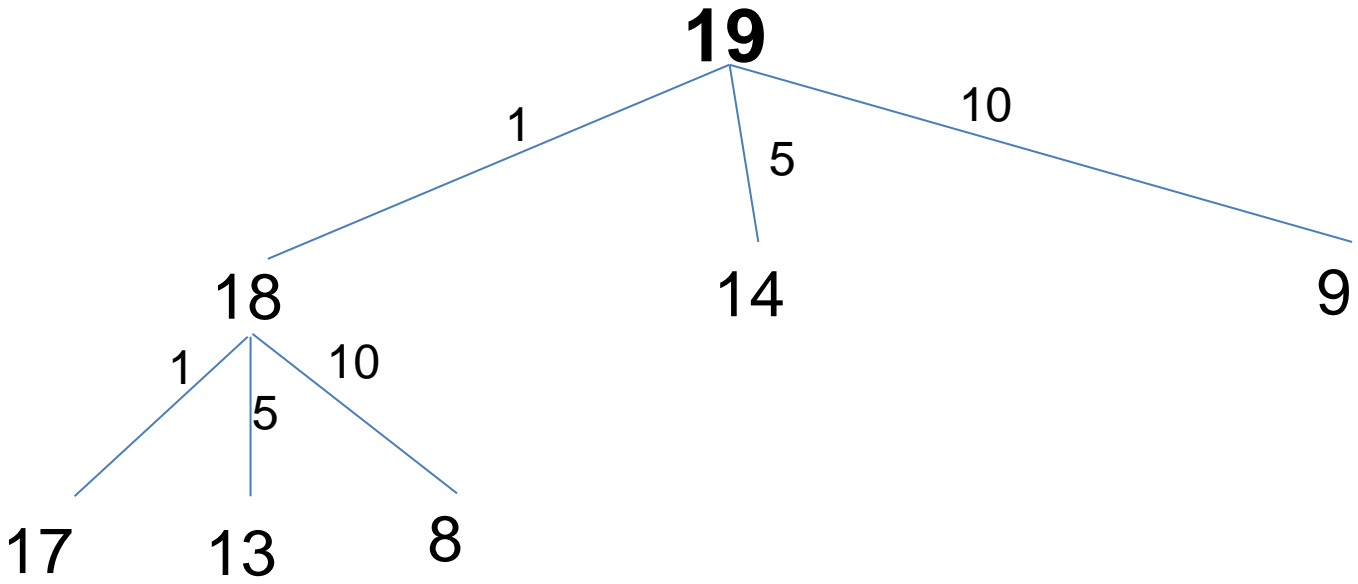
Find minimum coins needed to make 13 cents

Find minimum coins needed to make 8 cents

Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

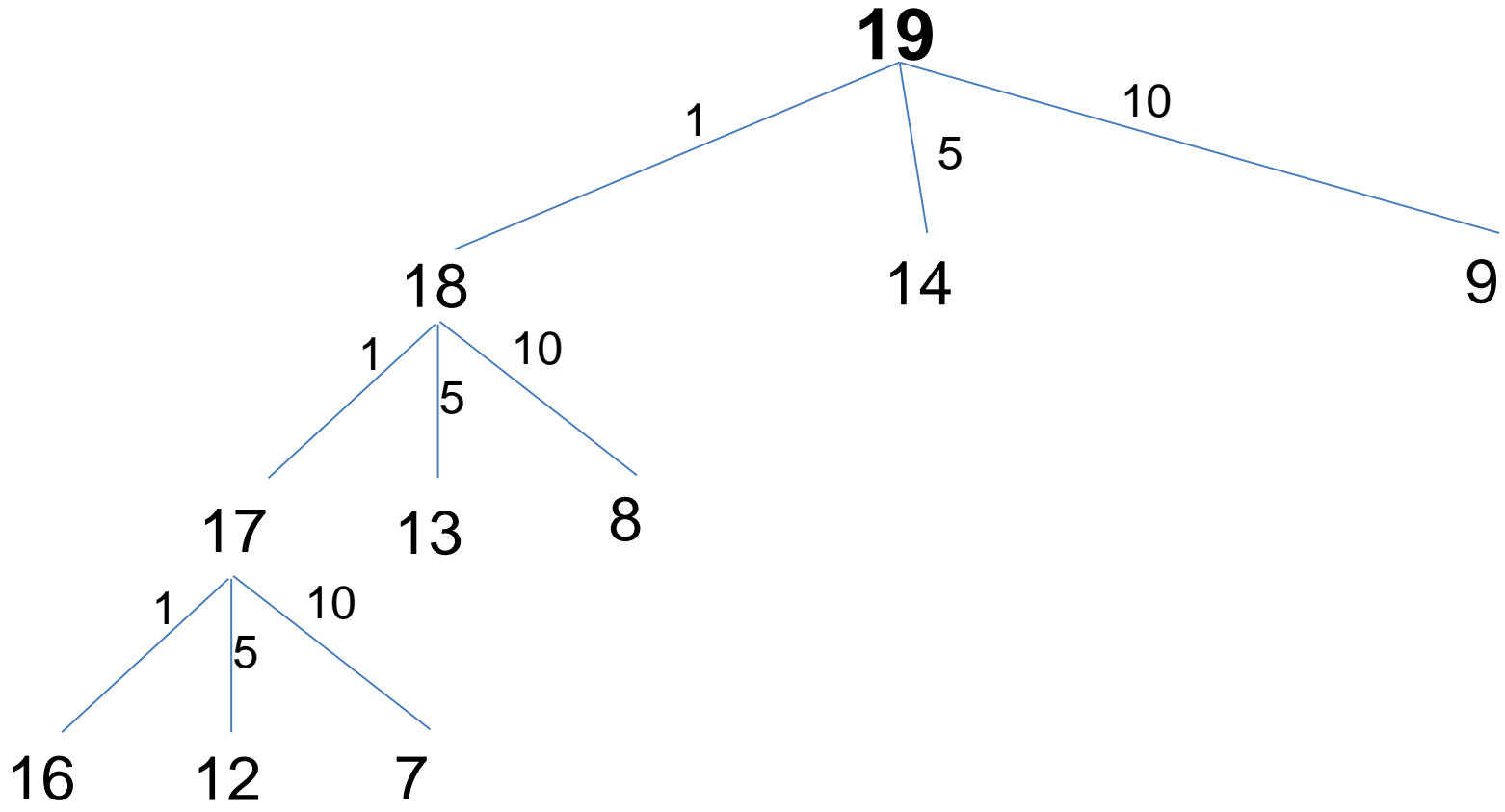
k = # denominations



Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

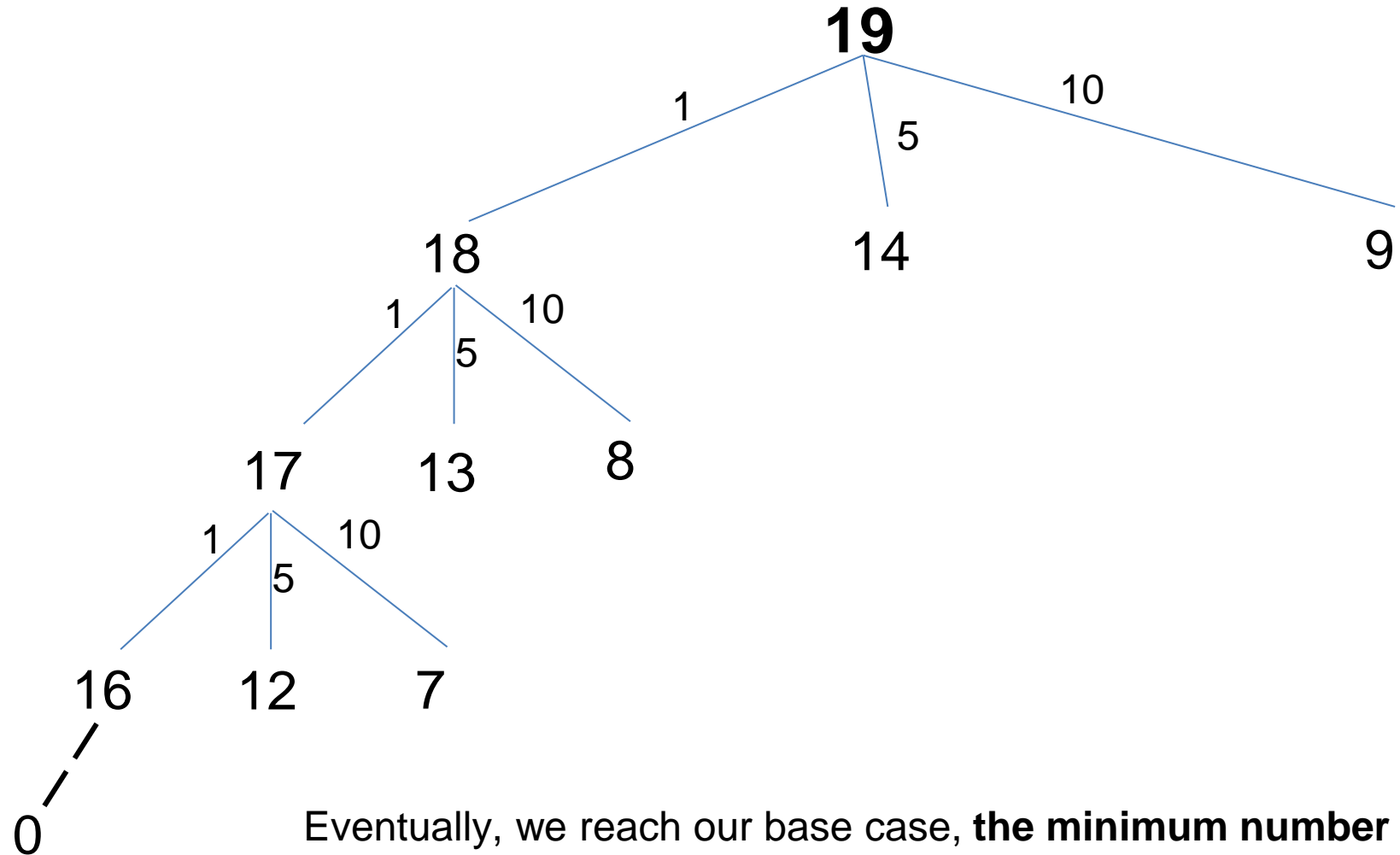
k = # denominations



Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

k = # denominations

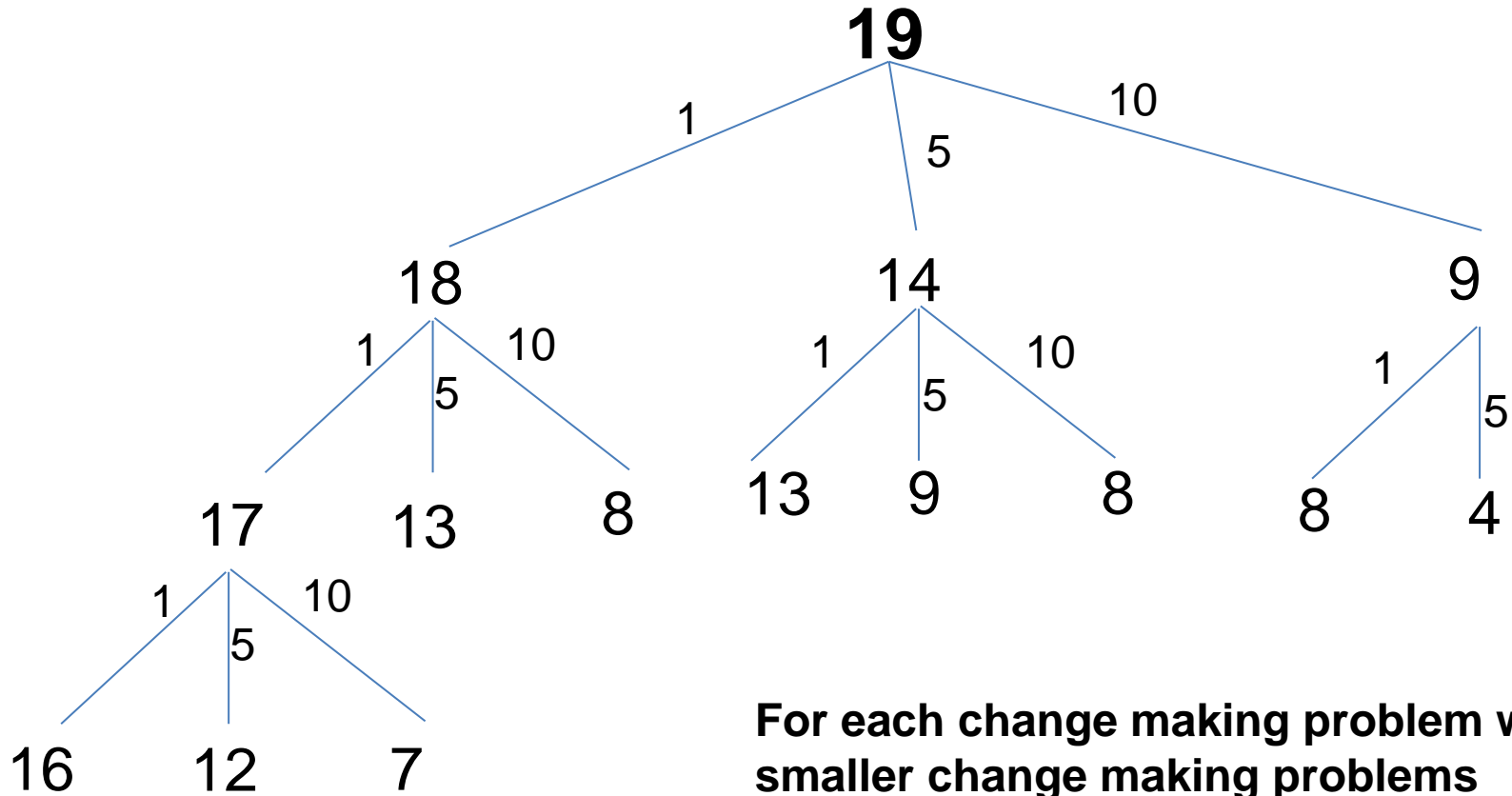


Eventually, we reach our base case, **the minimum number of coins needed to make 0 cents**

Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

$k = \#$ denominations



When $C(9)$, we cant use a 10 cent piece...

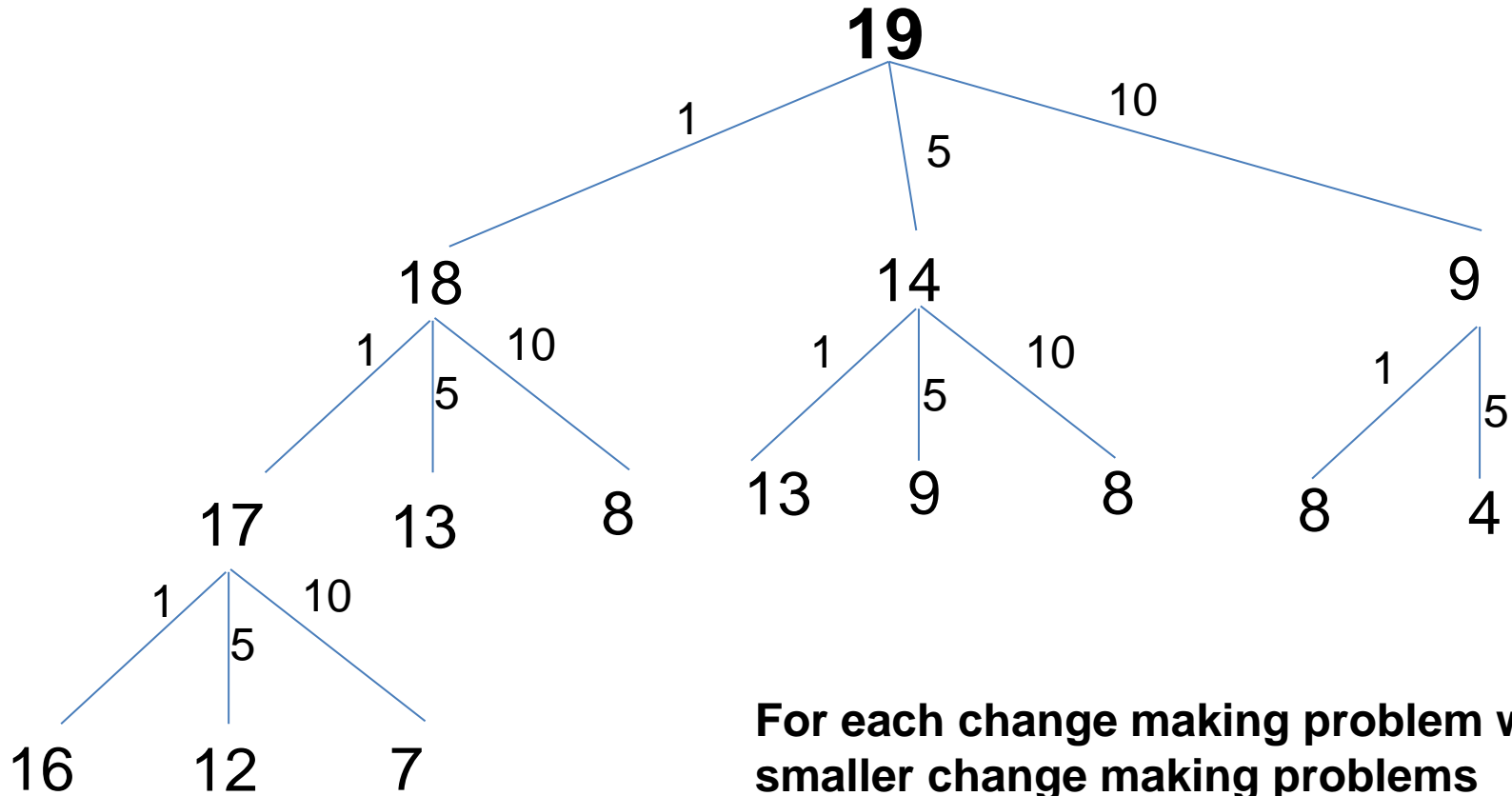
For each change making problem we solve, we must solve at most 3 smaller change making problems

Once we solve the smaller problems, we must select the branch that has the minimum value

Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

$k = \#$ denominations



When $C(9)$, we can't use a 10 cent piece...

For each change making problem we solve, we must solve at most 3 smaller change making problems

Once we solve the smaller problems, we must select the branch that has the minimum value

Change Making Problem

$$C(p) = \begin{cases} \min_{i:d_i \leq p} C(p - d_i) + 1, & p > 0 \\ 0, & p = 0 \end{cases}$$

Least change for 19 cents = minimum of:

- least change for 19-10 = 9 cents
- least change for 19-5 = 14 cents
- least change for 19-1 = 18 cents

For each problem P , we will solve the problem for $(P - d)$, where d represents each possible denomination

Change Making Problem

$$C(p) = \begin{cases} \min_{i:d_i \leq p} C(p - d_i) + 1, & p > 0 \\ 0, & p = 0 \end{cases}$$

Least change for 19 cents = minimum of:

- least change for 19-10 = 9 cents
- least change for 19-5 = 14 cents
- least change for 19-1 = 18 cents

For each problem P, we will solve the problem for (P - d), where d represents each possible denomination

We want to select only the branch that yields the minimum value

Change Making Problem

$$C(p) = \begin{cases} \min_{i:d_i \leq p} C(p - d_i) + 1, & p > 0 \\ 0, & p = 0 \end{cases}$$

Least change for 19 cents = minimum of:

- least change for 19-10 = 9 cents
- least change for 19-5 = 14 cents
- least change for 19-1 = 18 cents

For each problem P, we will solve the problem for (P - d), where d represents each possible denomination

If we ever need to make change for 0 cents, return 0

We want to select only the branch that yields the minimum value

Change Making Problem

$$C(p) = \begin{cases} \min_{i:d_i \leq p} C(p - d_i) + 1, & p > 0 \\ 0, & p = 0 \end{cases}$$

Least change for 19 cents = minimum of:

- least change for 19-10 = 9 cents
- least change for 19-5 = 14 cents
- least change for 19-1 = 18 cents

For each problem P, we will solve the problem for (P - d), where d represents each possible denomination

If we ever need to make change for 0 cents, return 0

We want to select only the branch that yields the minimum value

Change Making Problem

D = array of denominations [1, 5, 10, 18, 25]
p = desired change (37)

```
min_coins(D, p)
```

Change Making Problem

D = array of denominations [1, 5, 10, 18, 25]
p = desired change (37)

```
min_coins(D, p)
```

```
    if p == 0  
        return 0;
```

Base Case

Change Making Problem

D = array of denominations [1, 5, 10, 18, 25]
p = desired change (37)

```
min_coins(D, p)
```

```
if p == 0  
    return 0;
```

```
else  
    min = ∞  
    a = ∞
```

Base Case

```
int min = Integer.MAX_VALUE;  
int a = Integer.MAX_VALUE;;
```

Change Making Problem

D = array of denominations [1, 5, 10, 18, 25]
p = desired change (37)

```
min_coins(D, p)
```

```
if p == 0  
    return 0;
```

Base Case

```
else
```

```
    min = ∞  
    a = ∞
```

```
int min = Integer.MAX_VALUE;  
int a = Integer.MAX_VALUE;;
```

```
    for each  $d_i$  in D  
        if  $(p - d_i) \geq 0$   
             $a = \min\_coins(D, p - d_i)$ 
```

Recurse, and find the minimum number of coins needed using each valid denomination

Change Making Problem

D = array of denominations [1, 5, 10, 18, 25]

p = desired change (37)

```
min_coins(D, p)
```

```
if p == 0  
    return 0;
```

Base Case

```
else
```

```
    min = ∞
```

```
    a = ∞
```

```
int min = Integer.MAX_VALUE;  
int a = Integer.MAX_VALUE;;
```

```
for each  $d_i$  in D
```

```
    if  $(p - d_i) \geq 0$ 
```

```
         $a = \text{min\_coins}(D, p - d_i)$ 
```

```
    if  $a < \text{min}$ 
```

```
         $\text{min} = a$ 
```

Recurse, and find the minimum number of coins needed using each valid denomination

Select the branch that has the minimum value

Change Making Problem

D = array of denominations [1, 5, 10, 18, 25]

p = desired change (37)

```
min_coins(D, p)
  if p == 0
    return 0;
  else
    min = ∞
    a = ∞
```

Base Case

```
int min = Integer.MAX_VALUE;
int a = Integer.MAX_VALUE;;
```

```
  for each  $d_i$  in D
    if  $(p - d_i) \geq 0$ 
       $a = \text{min\_coins}(D, p - d_i)$ 
    if  $a < \text{min}$ 
       $\text{min} = a$ 
```

Recurse, and find the minimum number of coins needed using each valid denomination

Select the branch that has the minimum value

Change Making Problem

D = array of denominations [1, 5, 10, 18, 25]

p = desired change (37)

```
min_coins(D, p)
  if p == 0
    return 0;
```

Base Case

```
else
```

```
  min = ∞
```

```
  a = ∞
```

```
int min = Integer.MAX_VALUE;
int a = Integer.MAX_VALUE;;
```

```
  for each  $d_i$  in D
```

```
    if  $(p - d_i) \geq 0$ 
```

```
      a = min_coins(D,  $p - d_i$ )
```

```
      if  $a < min$ 
```

```
        min = a
```

Recurse, and find the minimum number of coins needed using each valid denomination

Select the branch that has the minimum value

```
  return 1 + min
```

Once, our for loop finishes, we should know the branch that had the minimum, so return $(1 + min)$, 1 because one coin was used in the current method call

Change Making Problem

```
min_coins(D, p)
  if p == 0
    return 0;
  else
    min = ∞
    a = ∞

    for each di in D
      if (p - di) >= 0
        a = min_coins(D, p - di )
        if a < min
          min = a

    return 1 + min
```

Change Making Problem

```
min_coins(D, p)
  if p == 0
    return 0;
  else
    min = ∞
    a = ∞

    for each di in D
      if (p - di) >= 0
        a = min_coins(D, p - di )
        if a < min
          min = a

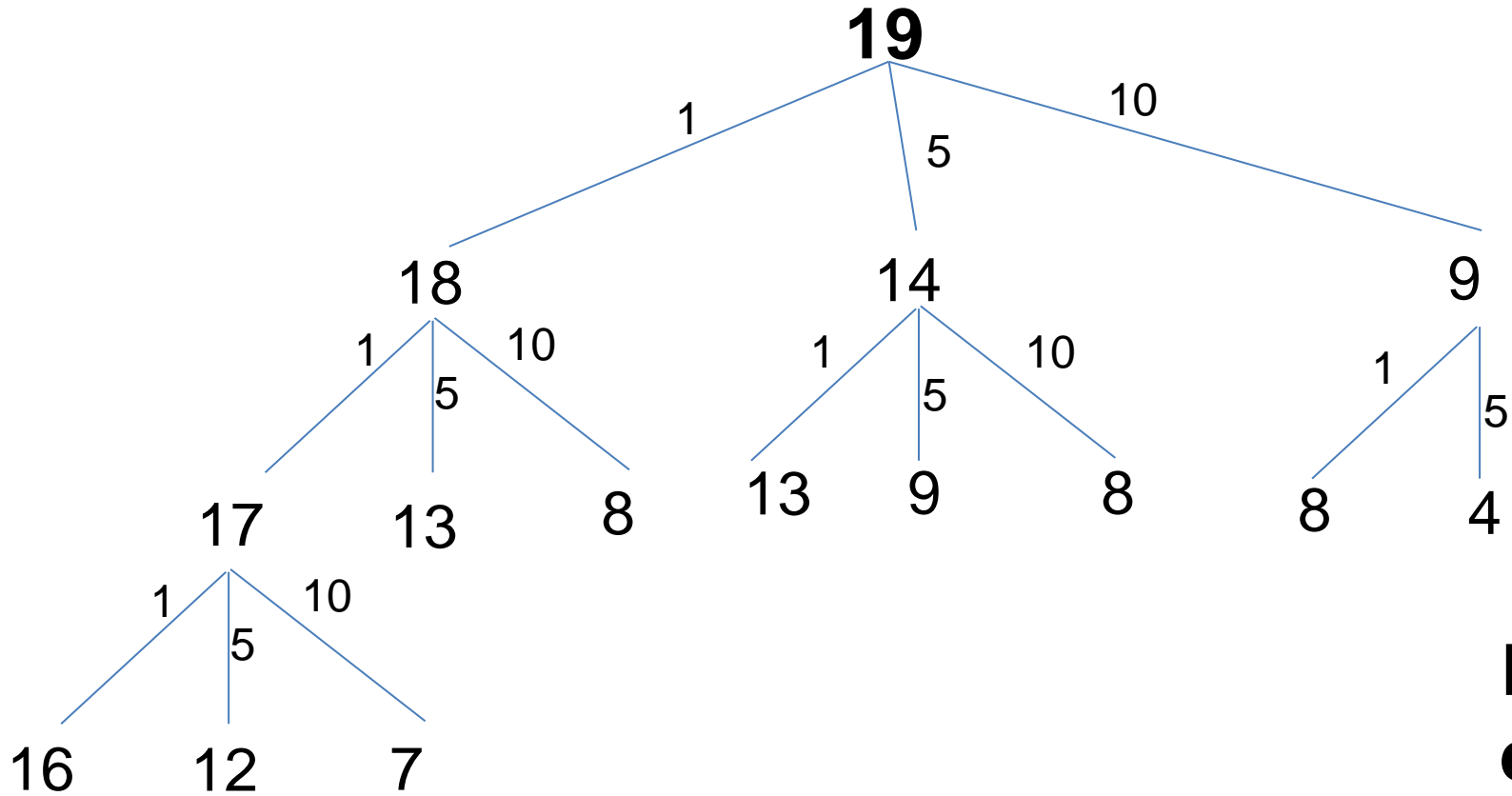
    return 1 + min
```

Running time?

Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

k = # denominations
p = value to make change for

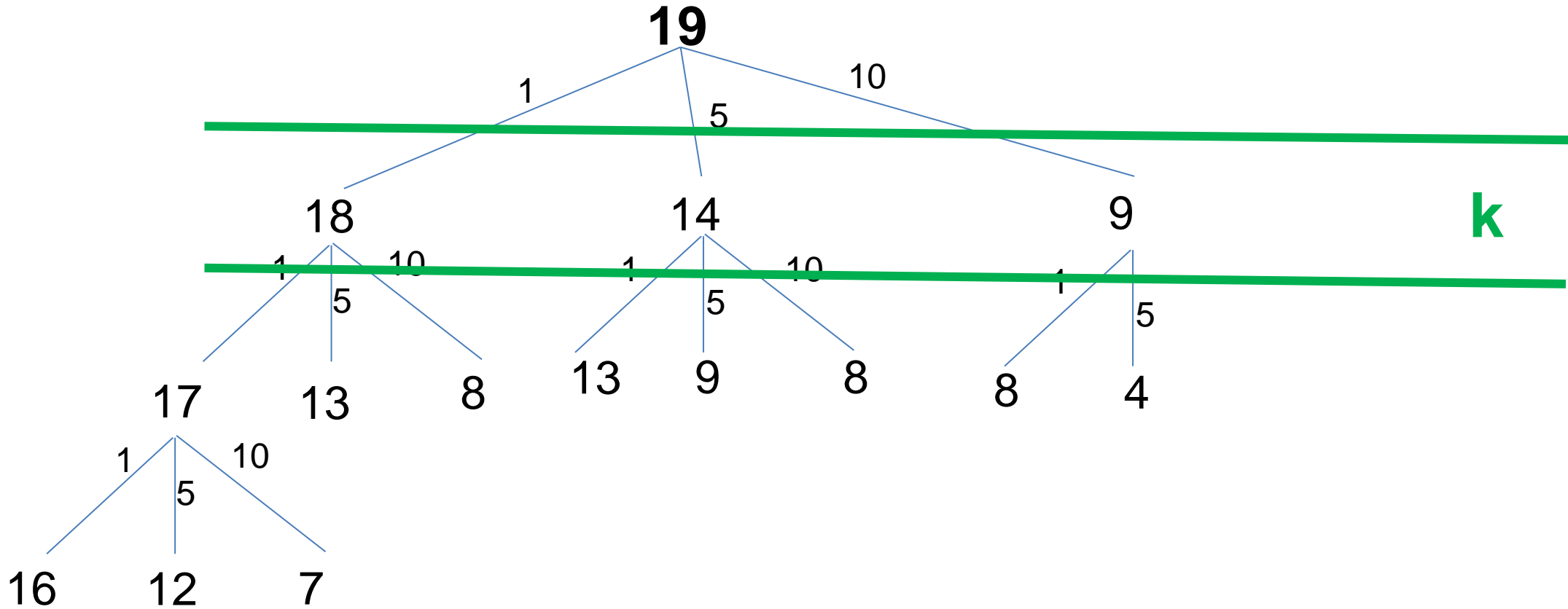


For sufficiently large p, **every permutation** of denominations is included.

Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

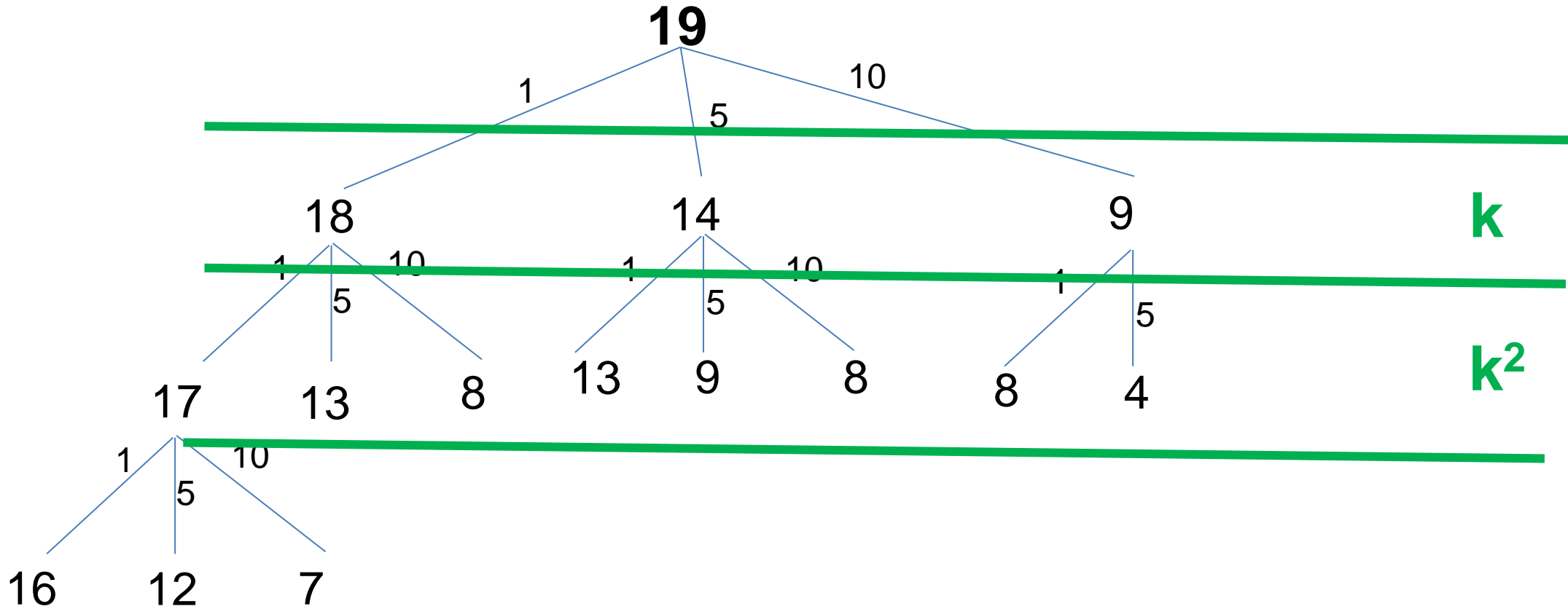
k = # denominations
p = value to make change for



Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

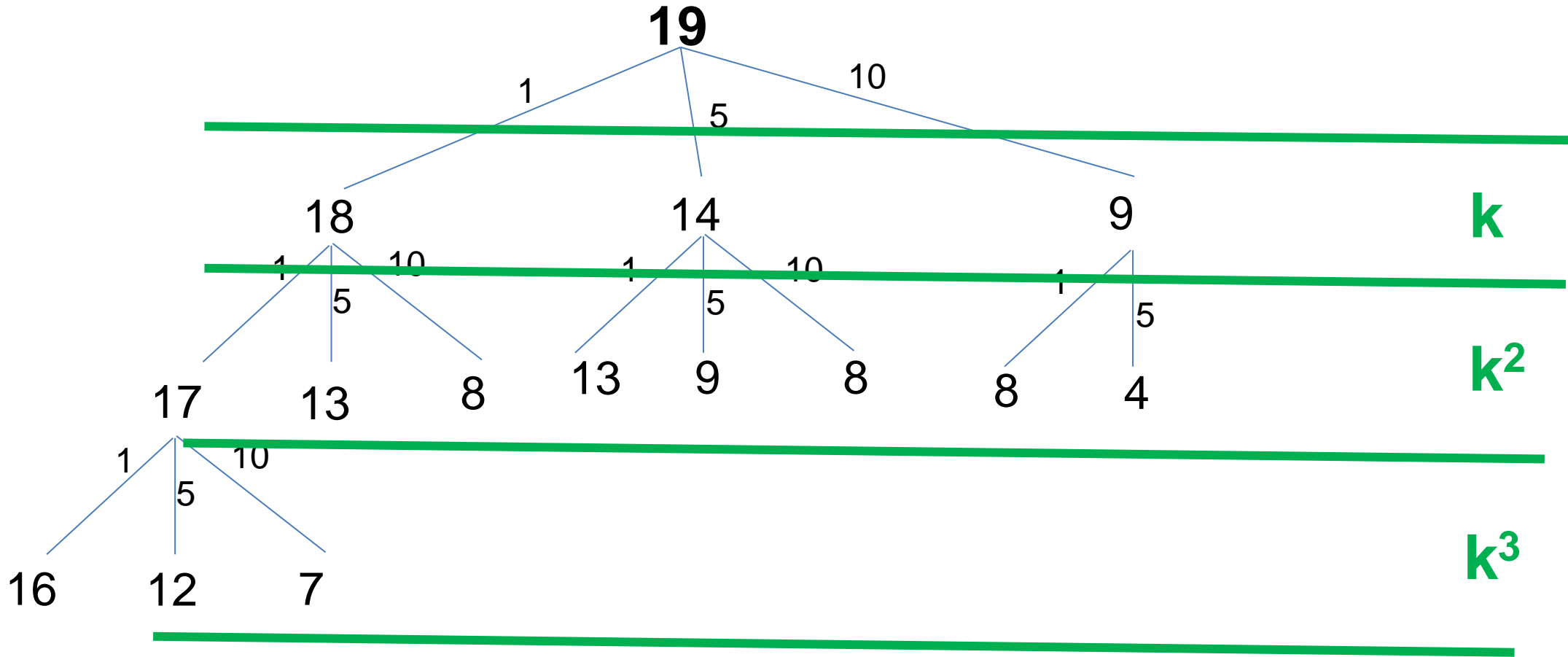
$k = \#$ denominations
 $p =$ value to make change for



Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

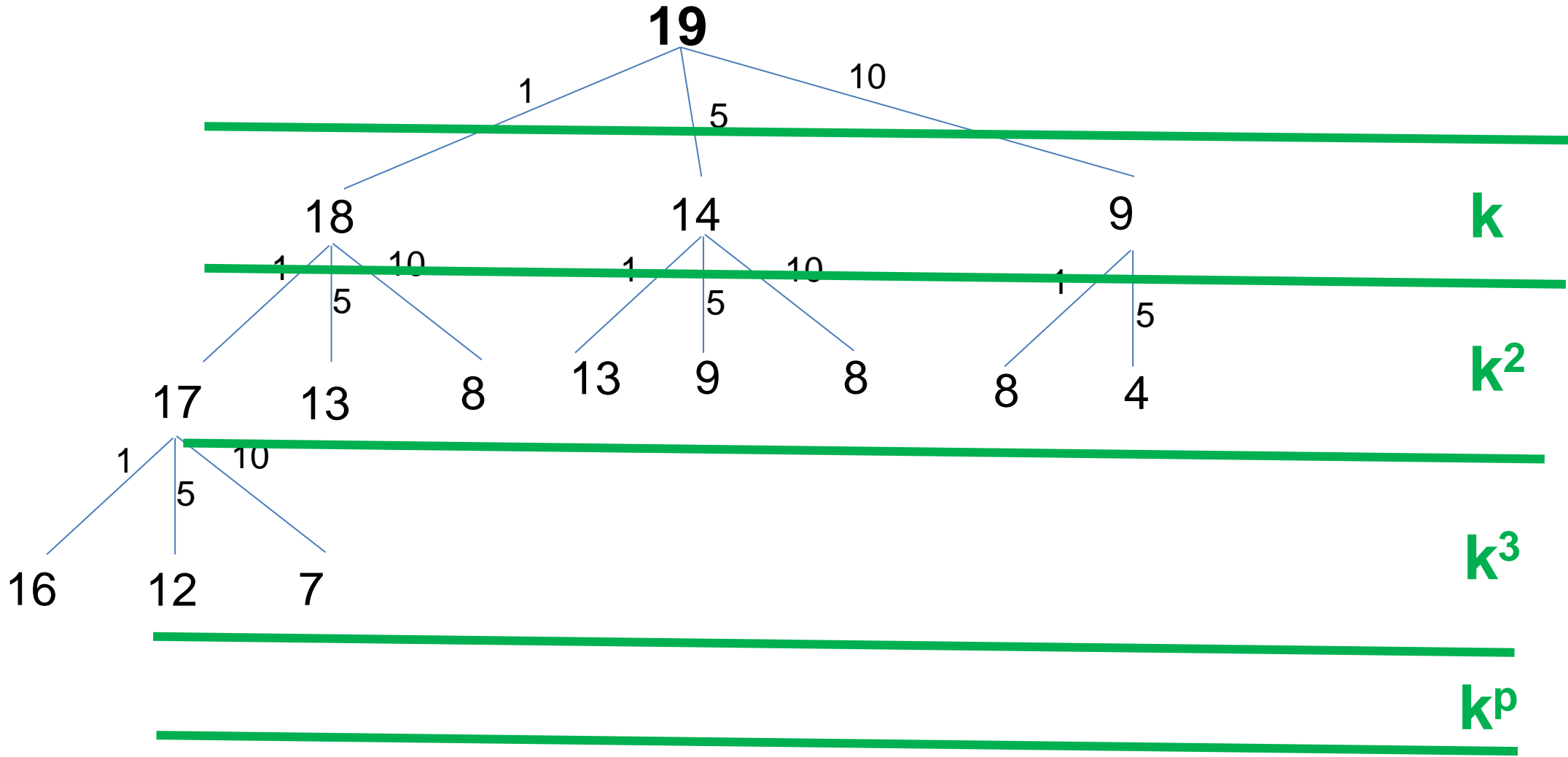
$k = \#$ denominations
 $p =$ value to make change for



Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

$k = \#$ denominations
 $p =$ value to make change for



Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

k = # denominations

p = value to make change for

10

As k and p both grow, the number of recursive calls being made will grow **exponentially**

If we have a lot of coin denominations, we will have **a lot** of branching

k

k^2

k^3

k^p

Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

k = # denominations

p = value to make change for

10

As k and p both grow, the number of recursive calls being made will grow **exponentially**

Running time: $O(\text{skull})$

k

k^2

k^3

k^p

Change Making Problem

Make \$0.19 with \$0.01, \$0.05, \$0.10

k = # denominations

p = value to make change for

10

As k and p both grow, the number of recursive calls being made will grow **exponentially**

Running time: probably $O(k^p)$ or $O(k!)$

For a large set of denominations, or a large p , this algorithm will take a long time to run

k

k^2

k^3

k^p

Change Making Problem

This algorithm returns the minimum number of coins needed (ie 4), but it does not tell us what coins were used in that solution

Change Making Problem

This algorithm returns the minimum number of coins needed (ie 4), but it does not tell us what coins were used in that solution

D = Array of coin denominations [1, 5, 10, 25]

p = value to make change for

n = minimum number of coins used to make p cents

Goal: Find an **n -length** combination of coins from **D** that were used to make **p**

Change Making Problem

This algorithm returns the minimum number of coins needed (ie 4), but it does not tell us what coins were used in that solution

D = Array of coin denominations [1, 5, 10, 25]

p = value to make change for

n = minimum number of coins used to make p cents

Goal: Find an **n -length** combination of coins from **D** that were used to make **p**

To do this, we will compute **all n -length combinations**, but only return the combinations that add up to be **p** (not very efficient)

For $n=3$, these are the combinations to be generated:

- [1, 1, 1]
- [1, 1, 5]
- [1, 1, 10]
- [1, 5, 5]
- [1, 5, 10]
- [1, 10, 10]
- [5, 5, 5]
- [5, 5, 10]
- [5, 10, 10]
- [10, 10, 10]

Note:

[5, 1, 1] is not a “valid” combination, because it is the same thing as [1, 1, 5]

Generating Combinations for finding coins

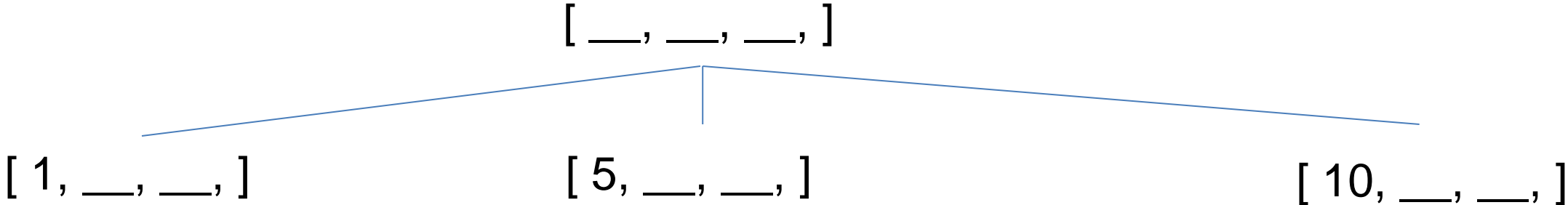
Denominations (D) = [1, 5, 10]
n = 3

[__, __, __,]



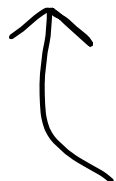
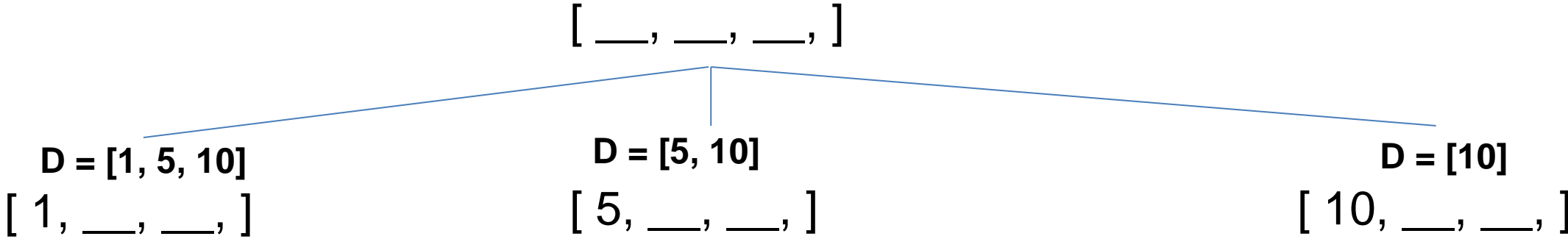
Generating Combinations for finding coins

Denominations (D) = [1, 5, 10]
n = 3



Generating Combinations for finding coins

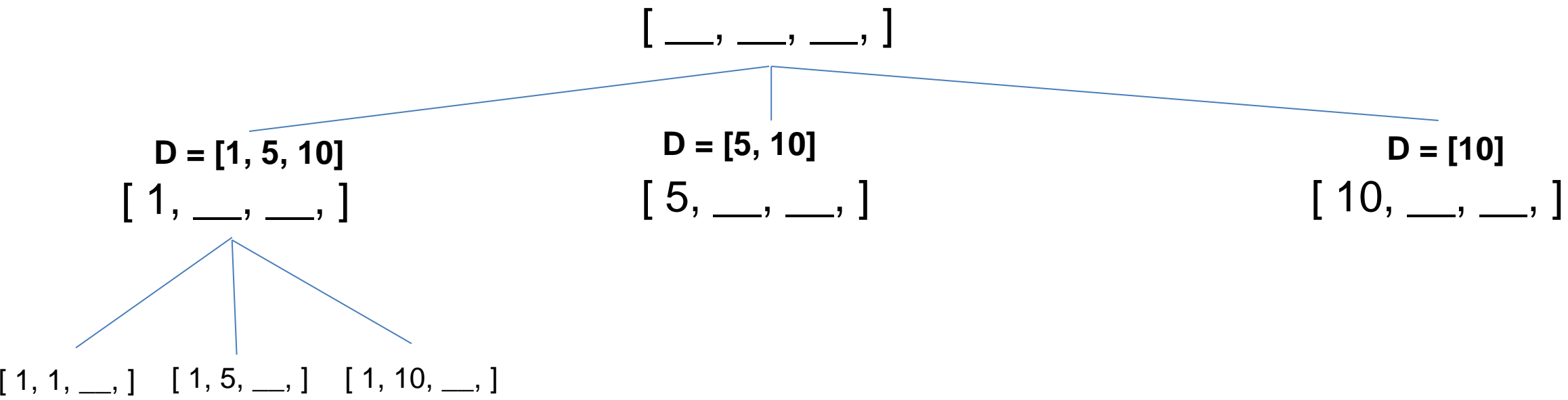
Denominations (D) = [1, 5, 10]
n = 3



This branch does not get 1 included in the denomination set, because all the combinations involving 1 will be handled by the left tree

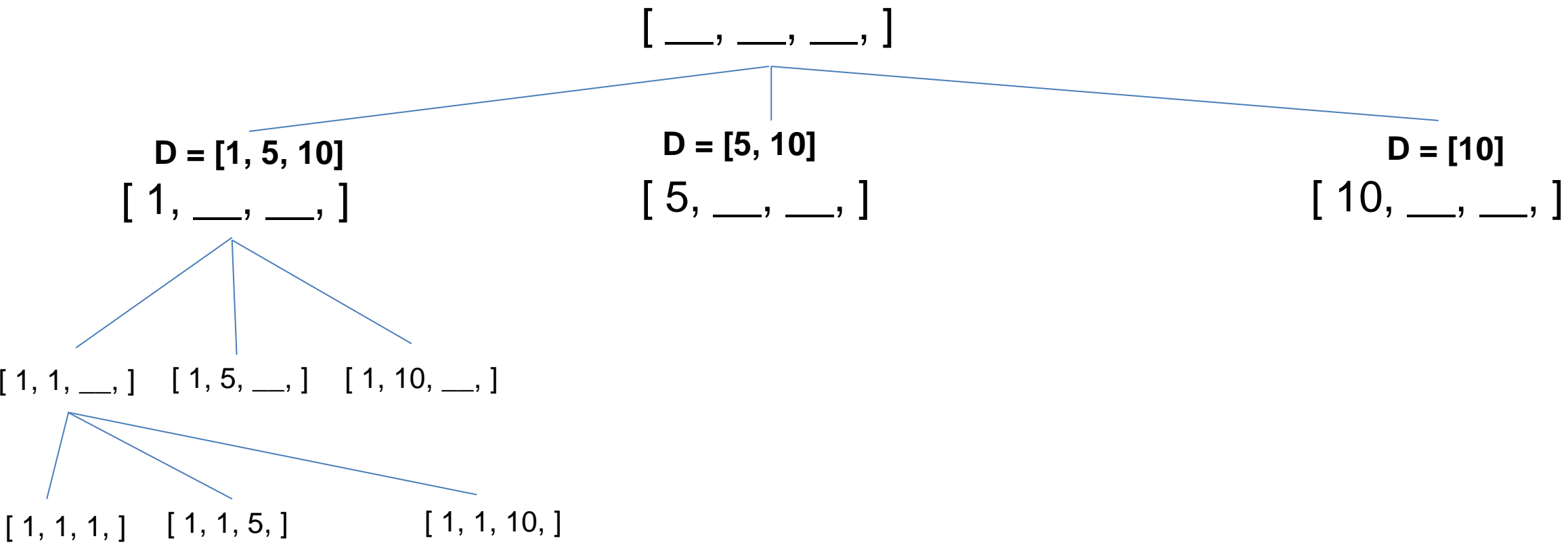
Generating Combinations for finding coins

Denominations (D) = [1, 5, 10]
n = 3



Generating Combinations for finding coins

Denominations (D) = [1, 5, 10]
n = 3

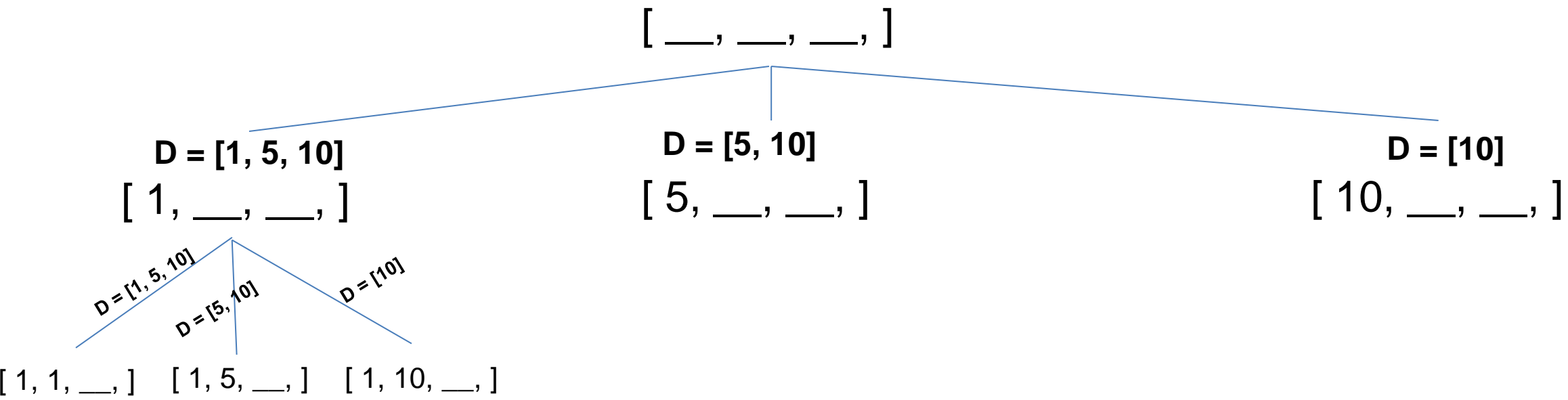


(Base case) When are combinations reach a length of 3, we will stop recursing

[1, 1, 1] [1, 1, 5] [1, 1, 10]

Generating Combinations for finding coins

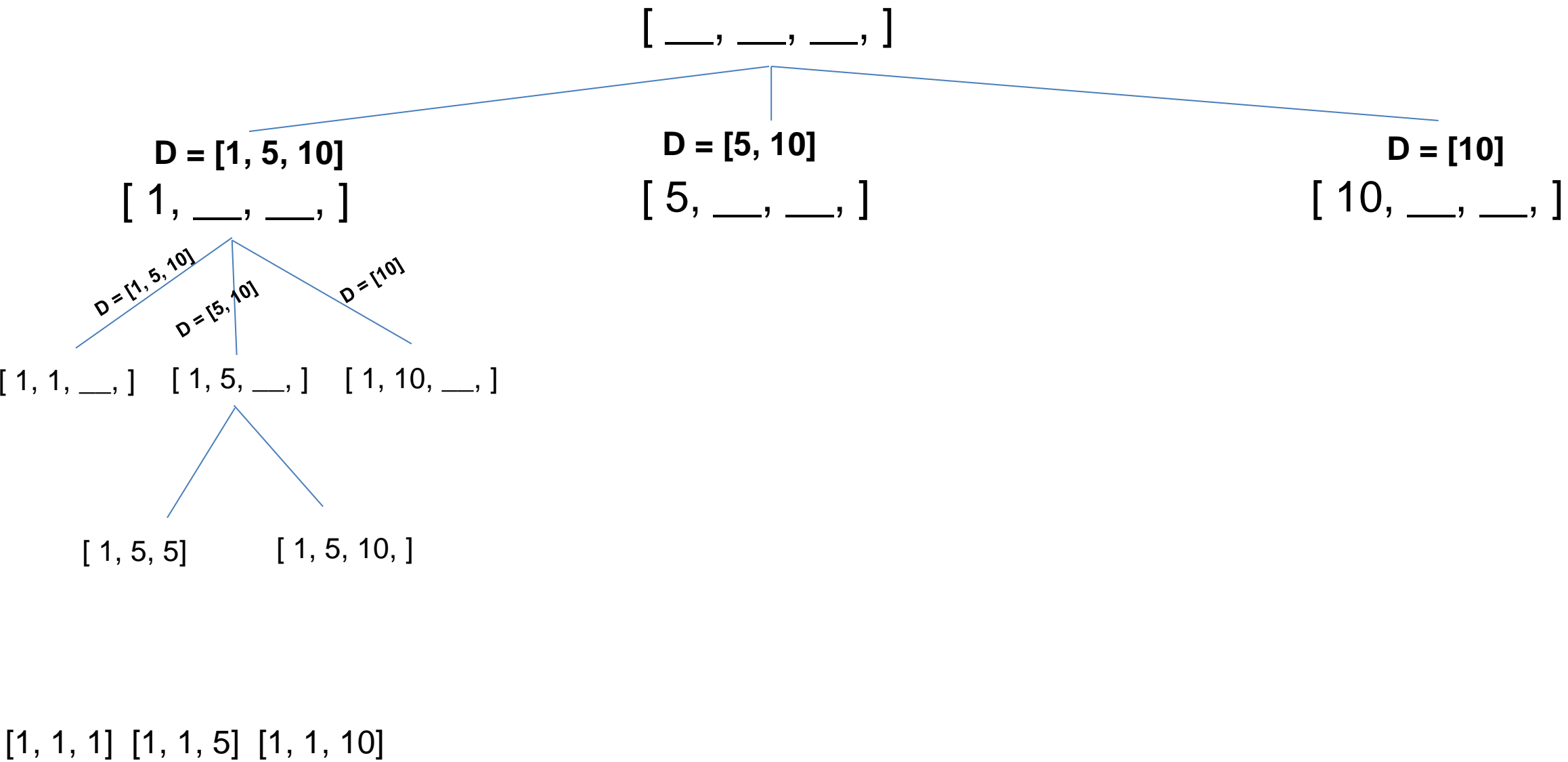
Denominations (D) = [1, 5, 10]
n = 3



[1, 1, 1] [1, 1, 5] [1, 1, 10]

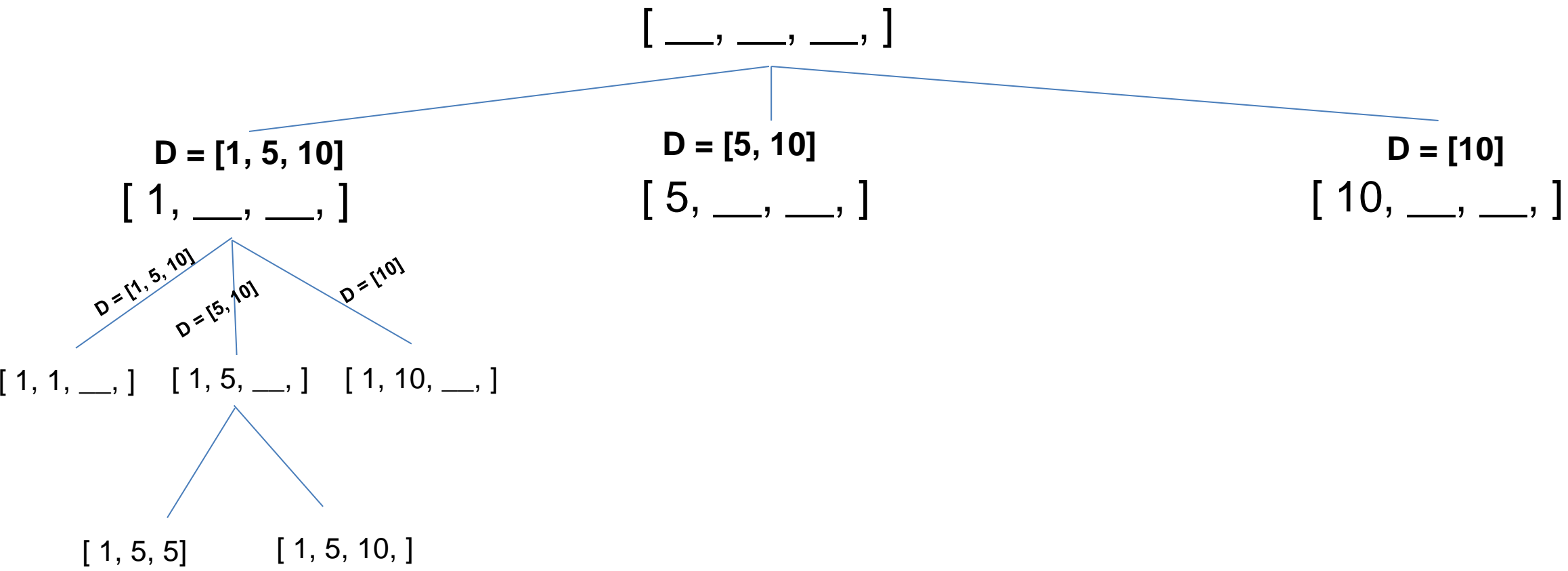
Generating Combinations for finding coins

Denominations (D) = [1, 5, 10]
 n = 3



Generating Combinations for finding coins

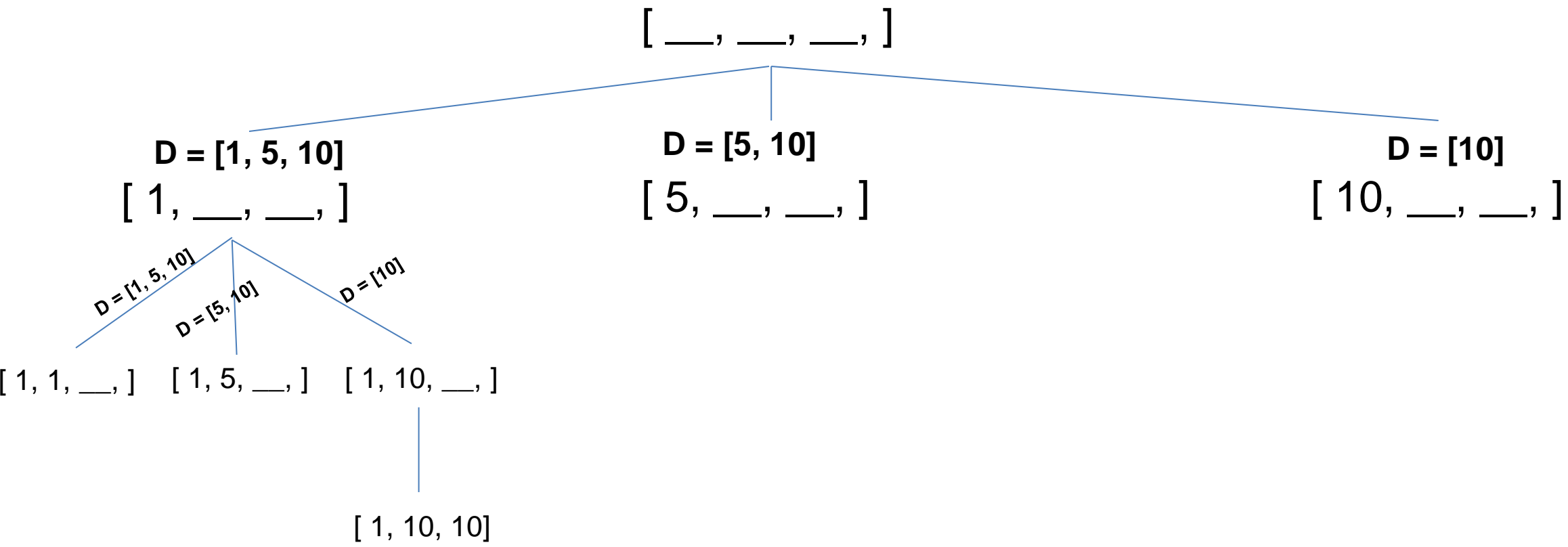
Denominations (D) = [1, 5, 10]
 n = 3



[1, 1, 1] [1, 1, 5] [1, 1, 10] [1, 5, 5] [1, 5, 10]

Generating Combinations for finding coins

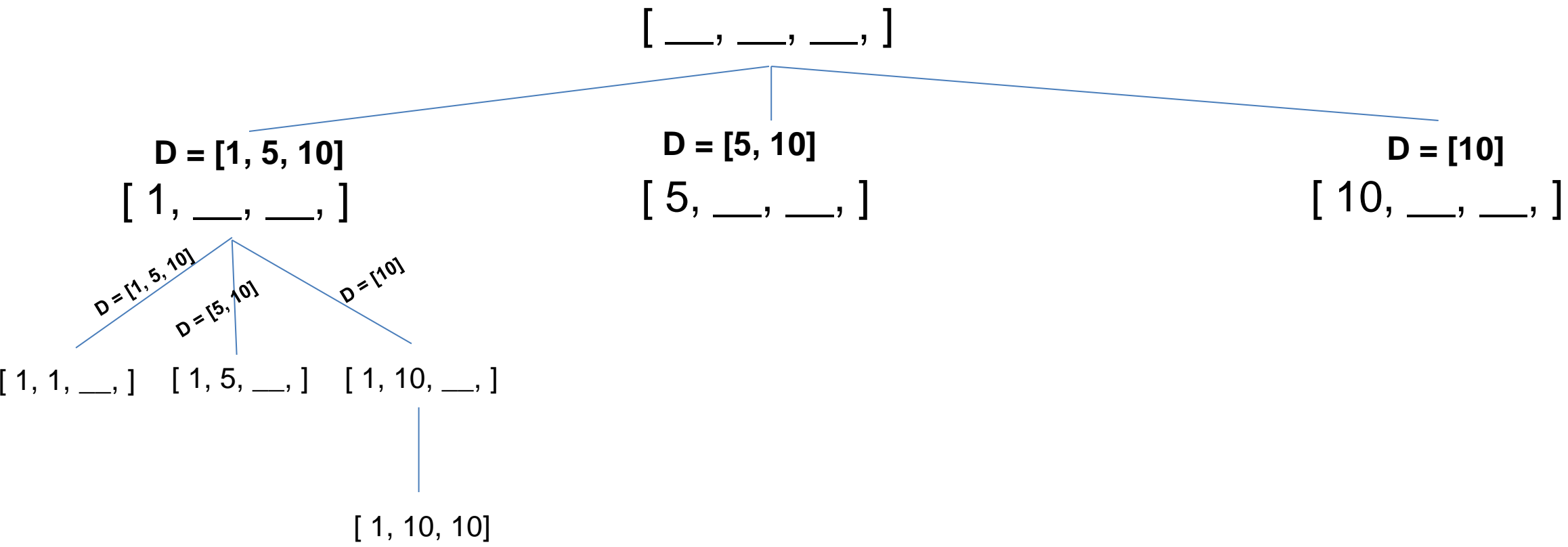
Denominations (D) = [1, 5, 10]
 n = 3



[1, 1, 1] [1, 1, 5] [1, 1, 10] [1, 5, 5] [1, 5, 10]

Generating Combinations for finding coins

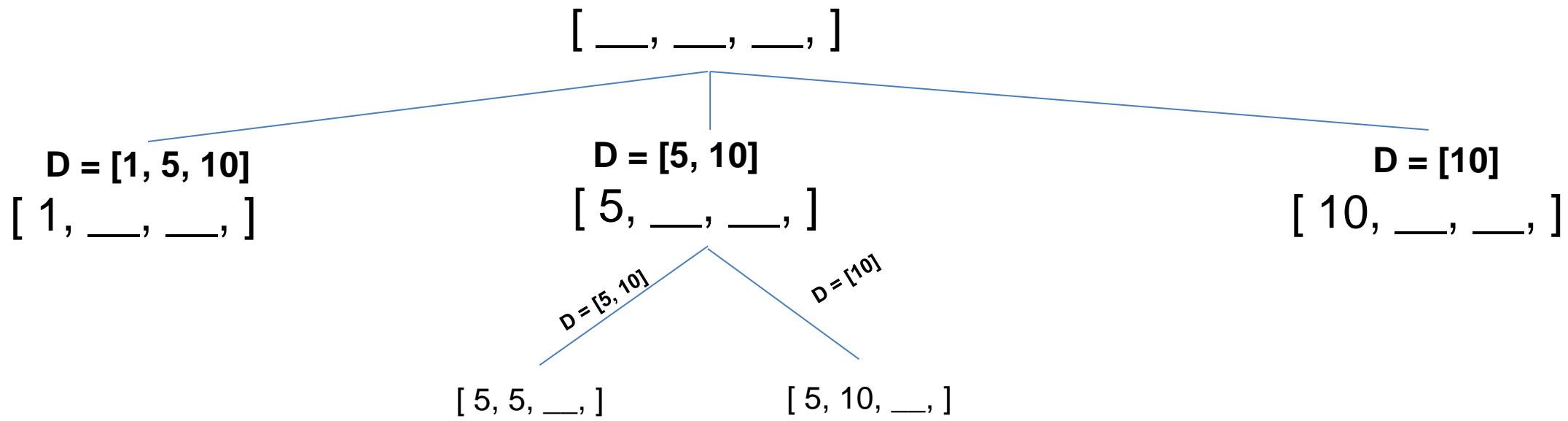
Denominations (D) = [1, 5, 10]
 n = 3



[1, 1, 1] [1, 1, 5] [1, 1, 10] [1, 5, 5] [1, 5, 10] [1, 10, 10]

Generating Combinations for finding coins

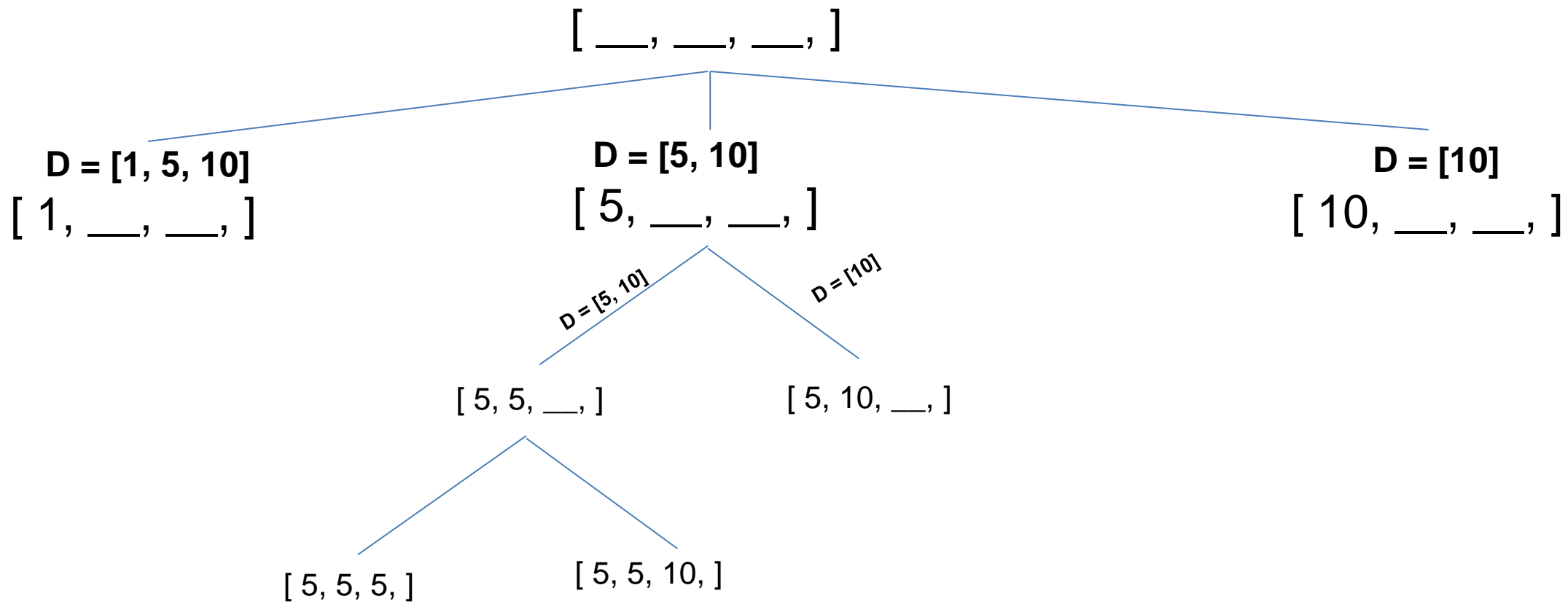
Denominations (D) = [1, 5, 10]
n = 3



[1, 1, 1] [1, 1, 5] [1, 1, 10] [1, 5, 5] [1, 5, 10] [1, 10, 10]

Generating Combinations for finding coins

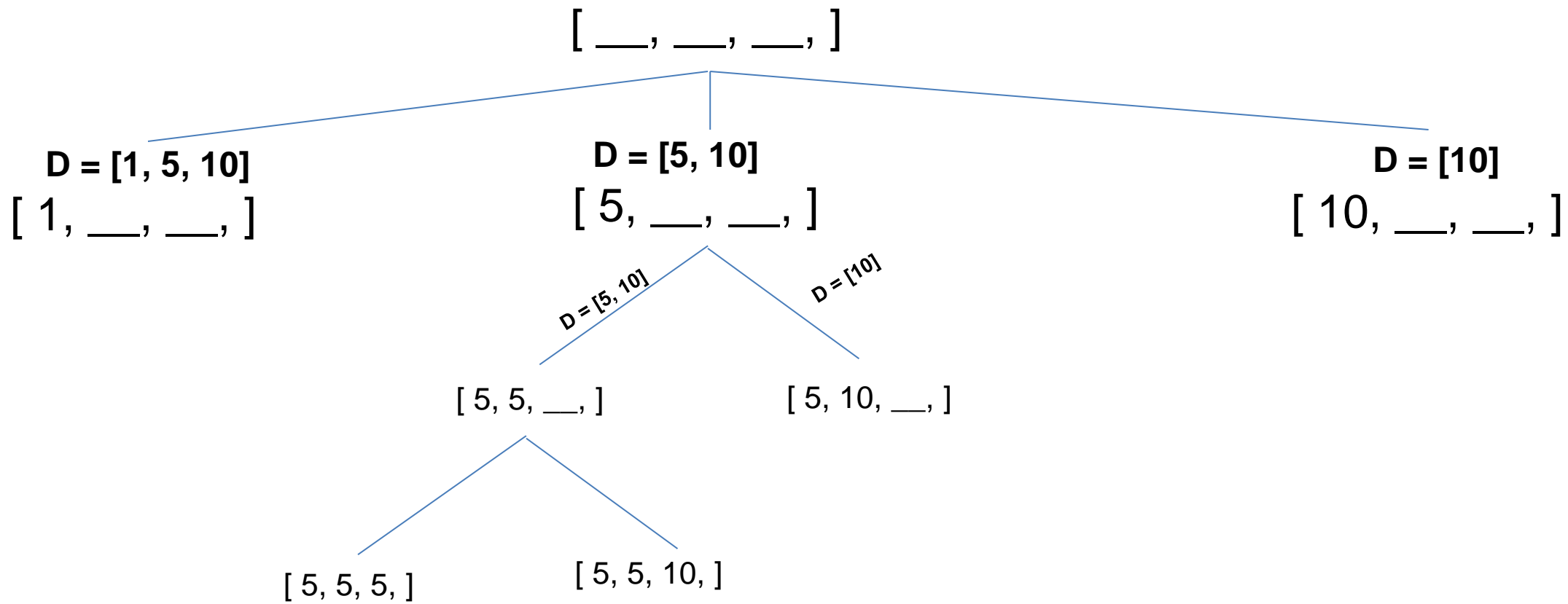
Denominations (D) = [1, 5, 10]
 n = 3



[1, 1, 1] [1, 1, 5] [1, 1, 10] [1, 5, 5] [1, 5, 10] [1, 10, 10]

Generating Combinations for finding coins

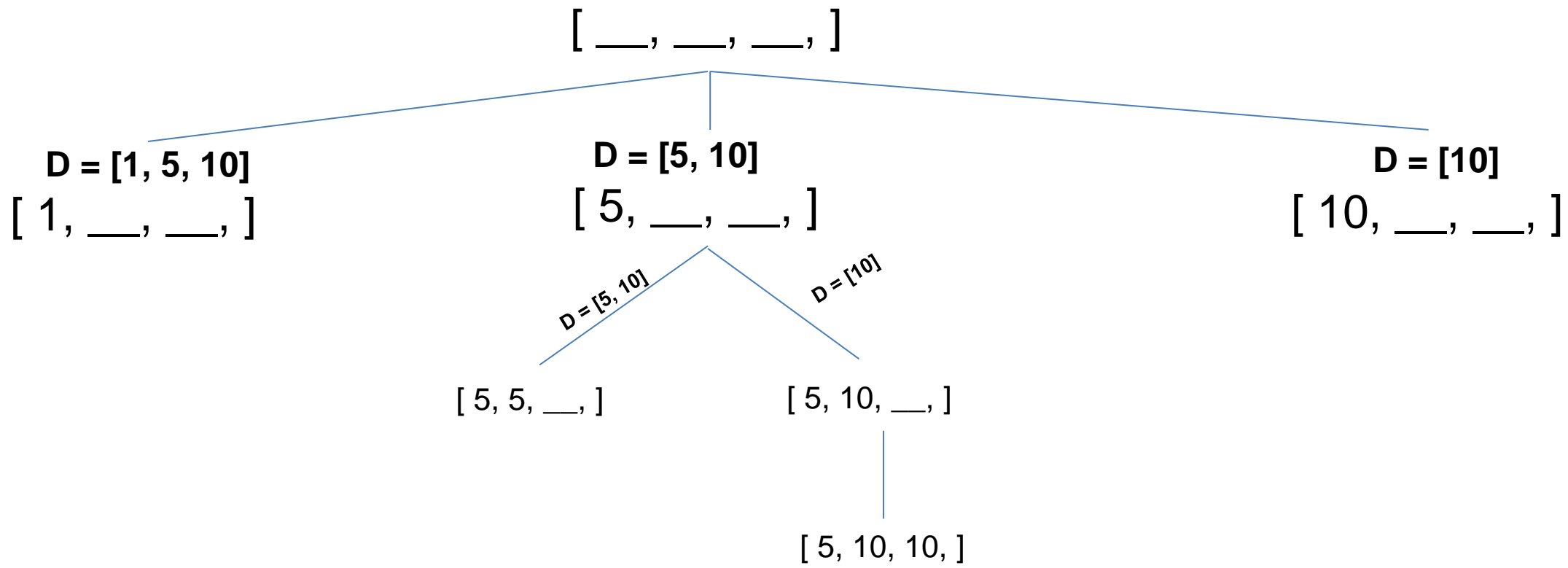
Denominations (D) = [1, 5, 10]
 n = 3



[1, 1, 1] [1, 1, 5] [1, 1, 10] [1, 5, 5] [1, 5, 10] [1, 10, 10] [5, 5, 5] [5, 5, 10]

Generating Combinations for finding coins

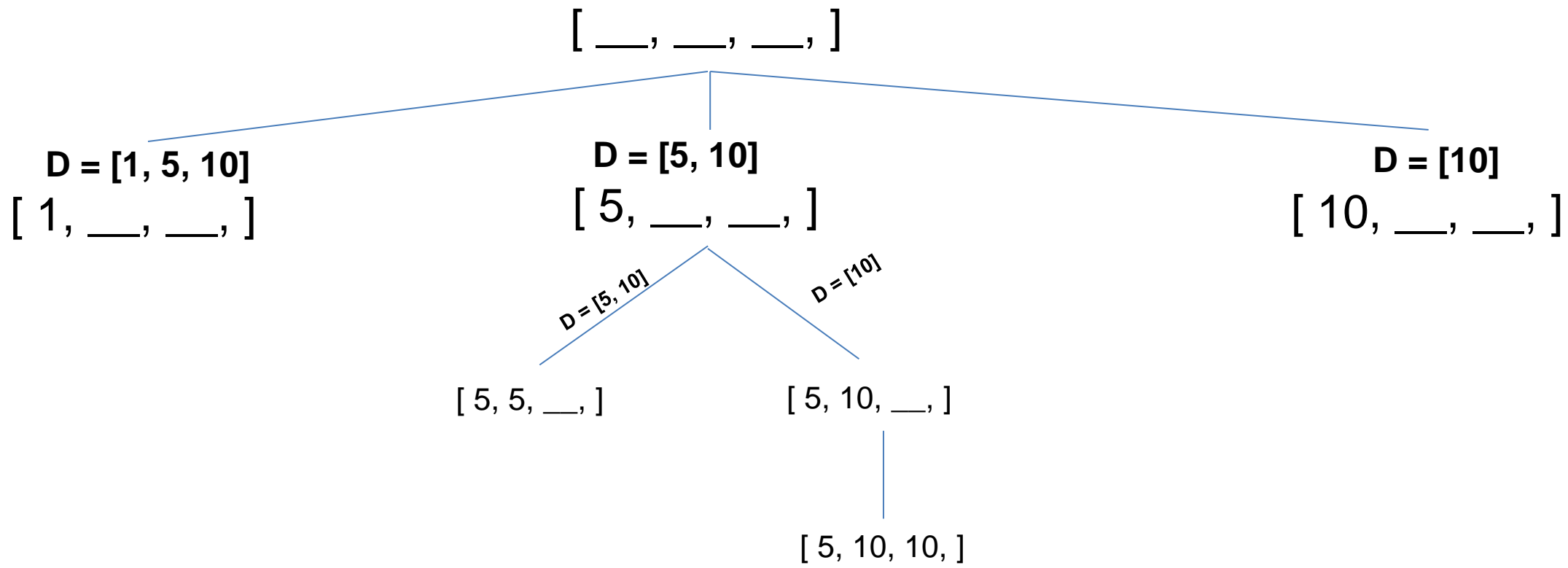
Denominations (D) = [1, 5, 10]
 n = 3



[1, 1, 1] [1, 1, 5] [1, 1, 10] [1, 5, 5] [1, 5, 10] [1, 10, 10] [5, 5, 5] [5, 5, 10]

Generating Combinations for finding coins

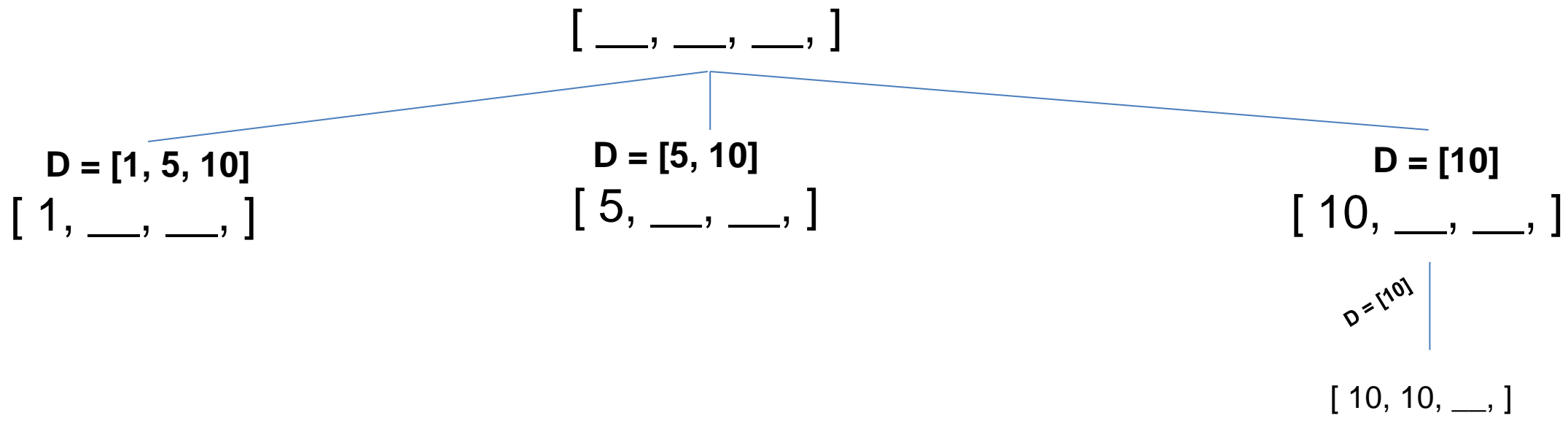
Denominations (D) = [1, 5, 10]
 n = 3



[1, 1, 1] [1, 1, 5] [1, 1, 10] [1, 5, 5] [1, 5, 10] [1, 10, 10] [5, 5, 5] [5, 5, 10] [5, 10, 10]

Generating Combinations for finding coins

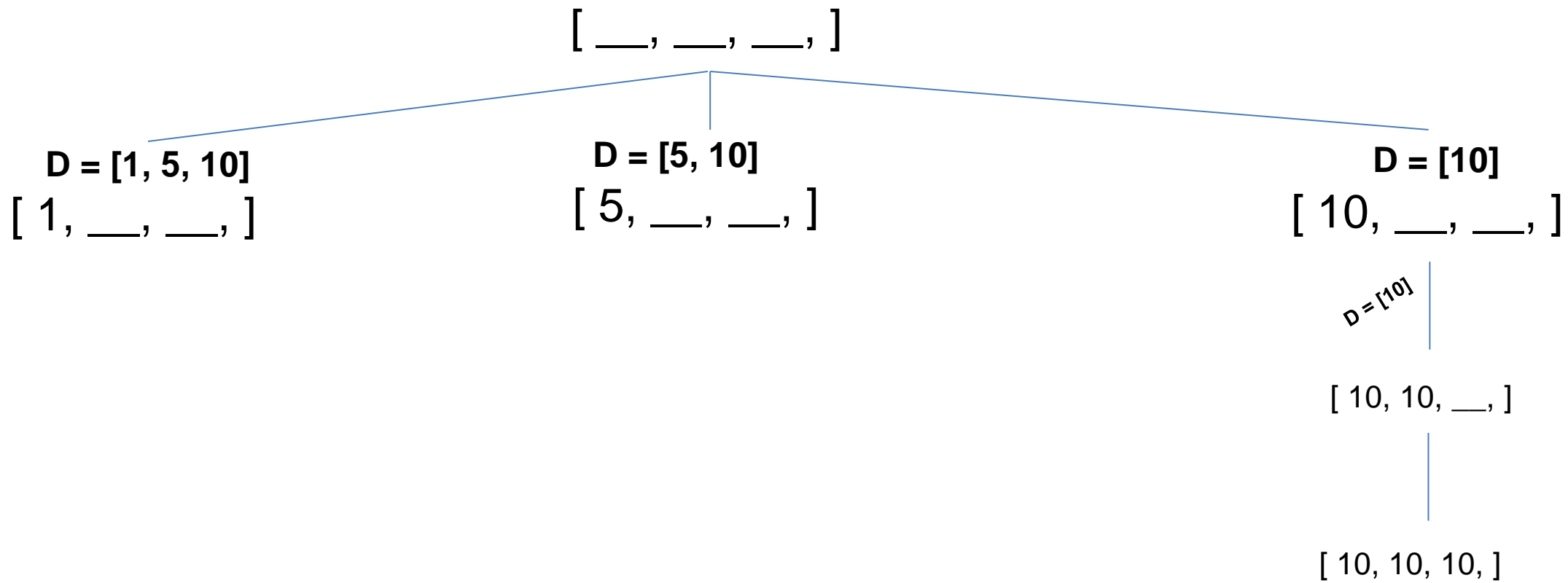
Denominations (D) = [1, 5, 10]
 n = 3



[1, 1, 1] [1, 1, 5] [1, 1, 10] [1, 5, 5] [1, 5, 10] [1, 10, 10] [5, 5, 5] [5, 5, 10] [5, 10, 10]

Generating Combinations for finding coins

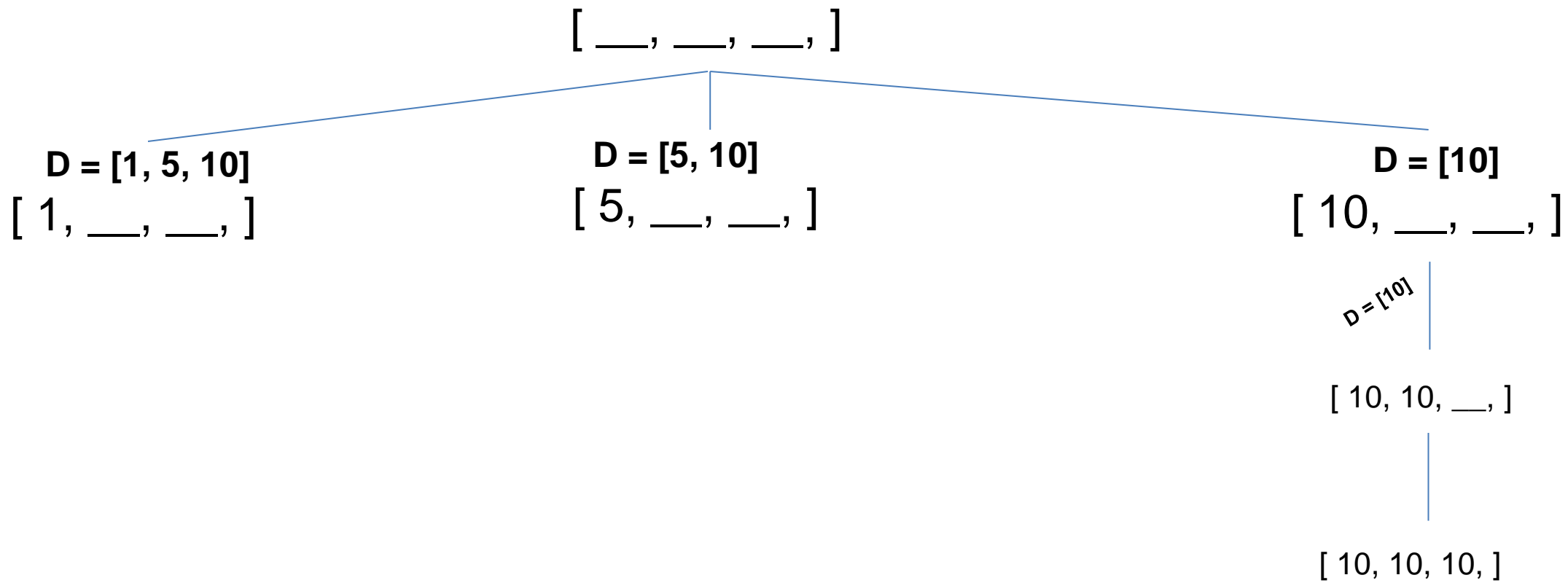
Denominations (D) = [1, 5, 10]
 n = 3



[1, 1, 1] [1, 1, 5] [1, 1, 10] [1, 5, 5] [1, 5, 10] [1, 10, 10] [5, 5, 5] [5, 5, 10] [5, 10, 10]

Generating Combinations for finding coins

Denominations (D) = [1, 5, 10]
 n = 3



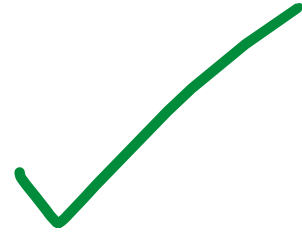
[1, 1, 1] [1, 1, 5] [1, 1, 10] [1, 5, 5] [1, 5, 10] [1, 10, 10] [5, 5, 5] [5, 5, 10] [5, 10, 10] [10, 10, 10]

Generating Combinations for finding coins

Denominations (D) = [1, 5, 10]

n = 3

1. [1, 1, 1]
2. [1, 1, 5]
3. [1, 1, 10]
4. [1, 5, 5]
5. [1, 5, 10]
6. [1, 10, 10]
7. [5, 5, 5]
8. [5, 5, 10]
9. [5, 10, 10]
10. [10, 10, 10]



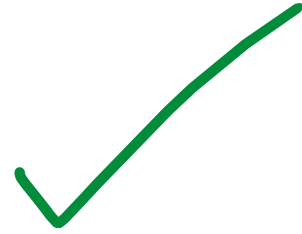
We've generated all combinations of length 3

Generating Combinations for finding coins

Denominations (D) = [1, 5, 10]

n = 3

1. [1, 1, 1]
2. [1, 1, 5]
3. [1, 1, 10]
4. [1, 5, 5]
5. [1, 5, 10]
6. [1, 10, 10]
7. [5, 5, 5]
8. [5, 5, 10]
9. [5, 10, 10]
10. [10, 10, 10]



We've generated all combinations of length 3

Now, we only want to print out the combinations that add up to **K**

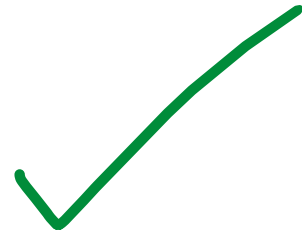
Generating Combinations for finding coins

Denominations (D) = [1, 5, 10]

$n = 3$

$K = 16$

1. [1, 1, 1]
2. [1, 1, 5]
3. [1, 1, 10]
4. [1, 5, 5]
5. [1, 5, 10]
6. [1, 10, 10]
7. [5, 5, 5]
8. [5, 5, 10]
9. [5, 10, 10]
10. [10, 10, 10]



We've generated all combinations of length 3

Now, we only want to print out the combinations that add up to K

Suppose $K = 16$ (a minimum of 3 coins is needed to make 16 cents)

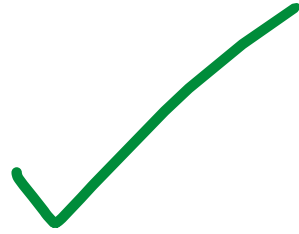
Generating Combinations for finding coins

Denominations (D) = [1, 5, 10]

$n = 3$

$K = 16$

1. [1, 1, 1]
2. [1, 1, 5]
3. [1, 1, 10]
4. [1, 5, 5]
5. [1, 5, 10]
6. [1, 10, 10]
7. [5, 5, 5]
8. [5, 5, 10]
9. [5, 10, 10]
10. [10, 10, 10]



We've generated all combinations of length 3

Now, we only want to print out the combinations that add up to K

Suppose $K = 16$ (a minimum of 3 coins is needed to make 16 cents)

Answer = [1, 5, 10]

LET'S CODE THIS!!

If you don't fully understand this code, that is fine.

YOU WILL REUSE THIS CODE FOR PROGRAM 5

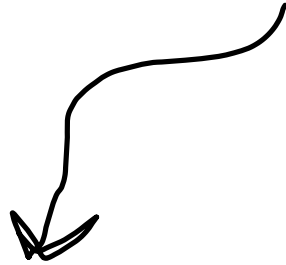
10.[10, 10, 10]

Answer = [1, 5, 10]

o to K

16 cents)

```
private static void find_coins(int[] d, int k, int n) {  
    int chosen[] = new int[n + 1];  
    calculate_combinations(chosen, d, 0, n, 0, d.length - 1, k);  
}
```



```
void calculate_combinations(int[] chosen, int[] arr, int index, int r, int start, int end, int target) {
```

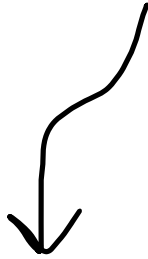
Array that we build up over time. Holds **indices** of currently selected denominations for some combination

[1, __, __]

[1, 1, __]

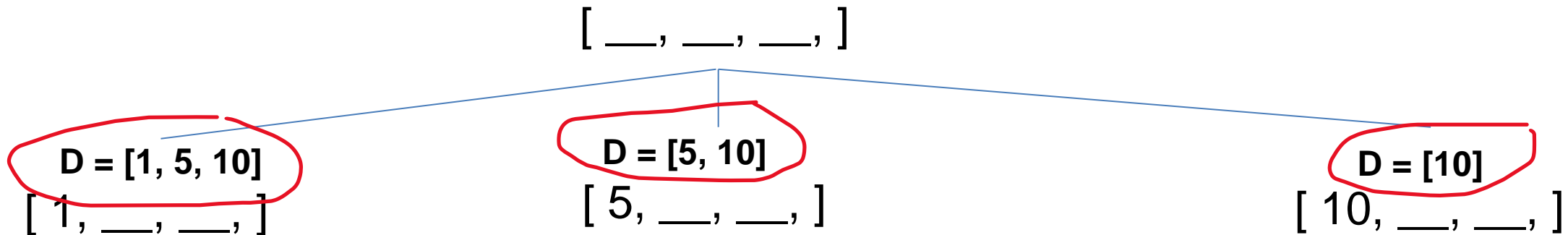
[1, 1, 5]

```
private static void find_coins(int[] d, int k, int n) {
    int chosen[] = new int[n + 1];
    calculate_combinations(chosen, d, 0, n, 0, d.length - 1, k);
}
```



```
void calculate_combinations(int[] chosen, int[] arr, int index, int r, int start, int end, int target) {
```

Array of denominations we pass for each recursive call



```
private static void find_coins(int[] d, int k, int n) {  
    int chosen[] = new int[n + 1];  
    calculate_combinations(chosen, d, 0, n, 0, d.length - 1, k);  
}
```



```
void calculate_combinations(int[] chosen, int[] arr, int index, int r, int start, int end, int target) {
```

The next index that we need to insert at for chosen array

[__, __, __]



index

[1, __, __]



index

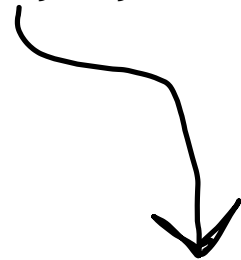
[1, 1, __]



index

[1, 1, 5]

```
private static void find_coins(int[] d, int k, int n) {
    int chosen[] = new int[n + 1];
    calculate_combinations(chosen, d, 0, n, 0, d.length - 1, k);
}
```



```
void calculate_combinations(int[] chosen, int[] arr, int index, int r, int start, int end, int target) {
```

The desired size of the combination. When `index == r`, we have reached the desired combination size

[__, __, __]



index = 0

[1, __, __]



index = 1

[1, 1, __]



index = 2

[1, 1, 5]



index = 3

**We've hit our
desired size!**

```

void calculate_combinations(int[] chosen, int[] arr, int index, int r, int start, int end, int target) {
    if (index == r) {
        int counter = 0;
        ArrayList<Integer> coins = new ArrayList<Integer>();
        for (int i = 0; i < r; i++) {
            counter += arr[chosen[i]];
            coins.add(arr[chosen[i]]);
        }
        if(counter == target) {
            System.out.println(coins);
        }
        return;
    }
    for (int i = start; i <= end; i++) {
        chosen[index] = i;
        calculate_combinations(chosen, arr, index + 1, r, i, end, target);
    }
    return;
}
}

```

```

void calculate_combinations(int[] chosen, int[] arr, int index, int r, int start, int end, int target) {
    if (index == r) {
        int counter = 0;
        ArrayList<Integer> coins = new ArrayList<Integer>();
        for (int i = 0; i < r; i++) {
            counter += arr[chosen[i]];
            coins.add(arr[chosen[i]]);
        }
        if(counter == target) { ←
            System.out.println(coins);
        }
        return;
    }
    for (int i = start; i <= end; i++) {
        chosen[index] = i;
        calculate_combinations(chosen, arr, index + 1, r, i, end, target);
    }
    return;
}

```

Base case

If we hit our base case, we know we have N things, so put them in an ArrayList and add them up

Only print out the combination if it adds up to target

```

void calculate_combinations(int[] chosen, int[] arr, int index, int r, int start, int end, int target) {
    if (index == r) {
        int counter = 0;
        ArrayList<Integer> coins = new ArrayList<Integer>();
        for (int i = 0; i < r; i++) {
            counter += arr[chosen[i]];
            coins.add(arr[chosen[i]]);
        }
        if(counter == target) {
            System.out.println(coins);
        }
        return;
    }
}

```

Recursive Case

```

    for (int i = start; i <= end; i++) {
        chosen[index] = i;
        calculate_combinations(chosen, arr, index + 1, r, i, end, target);
    }
    return;
}

```

Otherwise, insert selected coin into the `chosen` array

create $(end - start)$ branches, and give it a smaller section of `D`

