CSCI 232

Dynamic Programming (Pt 1)

Dynamic Programming

Dynamic Programming is an algorithm technique used for optimization problems that involves smartly using recursion to solve a problem with many overlapping subproblems

Dynamic Programming

Dynamic Programming is an algorithm technique used for optimization problems that involves smartly using recursion to solve a problem with many overlapping subproblems

To use dynamic programming, we must first identify two characteristics of some problem

Dynamic Programming

Dynamic Programming is an algorithm technique used for optimization problems that involves smartly using recursion to solve a problem with many overlapping subproblems

To use dynamic programming, we must first identify two characteristics of some problem

Optimal substructure- an optimal solution can be constructed from optimal solutions of its sub problems

Dynamic Programming

Dynamic Programming is an algorithm technique used for optimization problems that involves smartly using recursion to solve a problem with many overlapping subproblems

To use dynamic programming, we must first identify two characteristics of some problem

Optimal substructure- an optimal solution can be constructed from optimal solutions of its sub problems

Overlapping Subproblems- we solve the same sum problem several times during the algorithm

Change Making Problem

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

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

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)

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)

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 **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, …

This is known as the **greedy** approach

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

Greedy Algorithm

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

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, 18, 25]

K = 37

Greedy Algorithm

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

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

K = 37

Greedy Algorithm

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

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

K = 37

Greedy Algorithm

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

Greedy Algorithm

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

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

K = 37

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)

$$25 + \boxed{25 + 10 + 1 + 1 + 1} = 63$$

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 13 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)

$$25 + 25 + 10 \boxed{+ 1 + 1 + 1 =} 63$$

This is the minimum coins needed to make 3 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 2 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:

$$1 = d_1 < d_2 < \cdots < d_k$$ (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.

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

$$C(37) = 1 + \underbrace{C(12)}$$

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) 

C(12) = 1 + C(2) 

C(2) = 1 + C(1) 

C(1) = 1 + C(0) 

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) = 1 + C(2) 

C(2) = 1 + C(1) 

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) 

C(12) = 1 + C(2) 

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) 

C(12) = 1 + C(2) 

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) 

C(12) = 1 + 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) 

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:

$$1 = d_1 < d_2 < \cdots < d_k$$

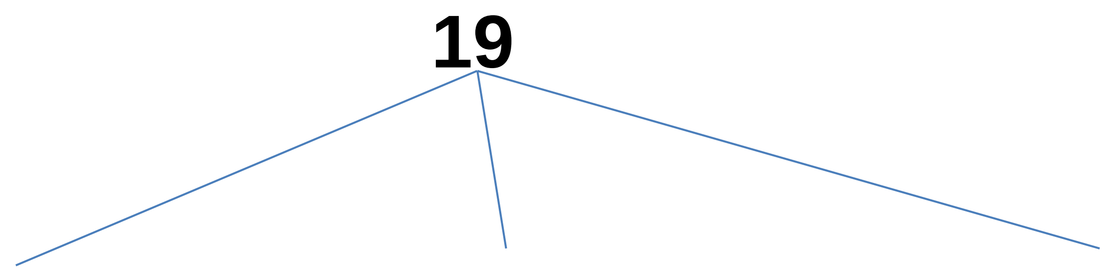
(US coins: $d_1 = 1, d_2 = 5, d_3 = 10, d_4 = 25$)

(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.

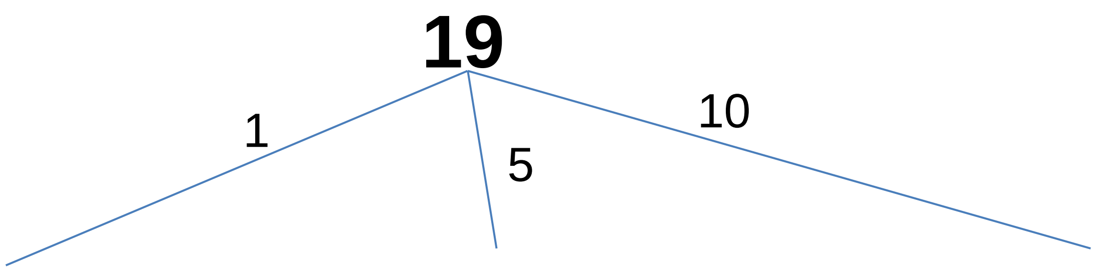
Change Making Problem

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



Change Making Problem

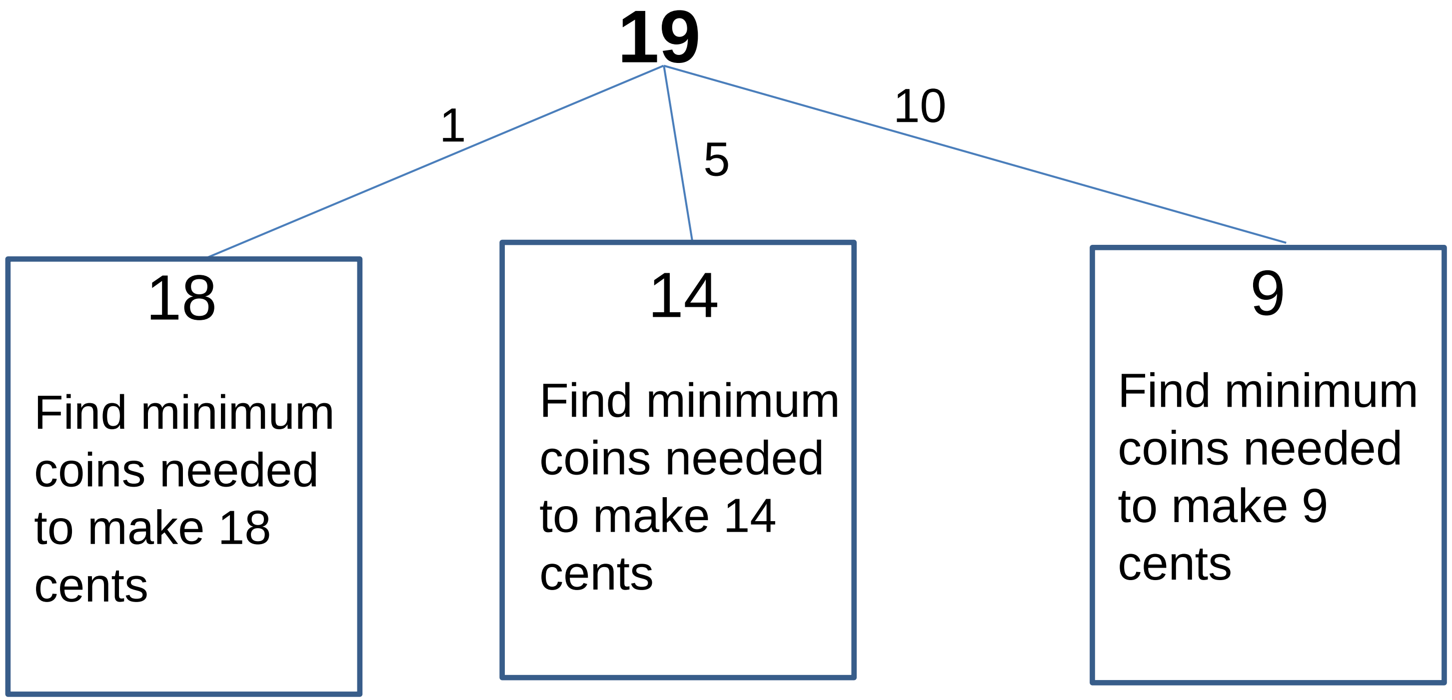
Make \$0.19 with \$0.01, \$0.05, \$0.10



Change Making Problem

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

k = # denominations

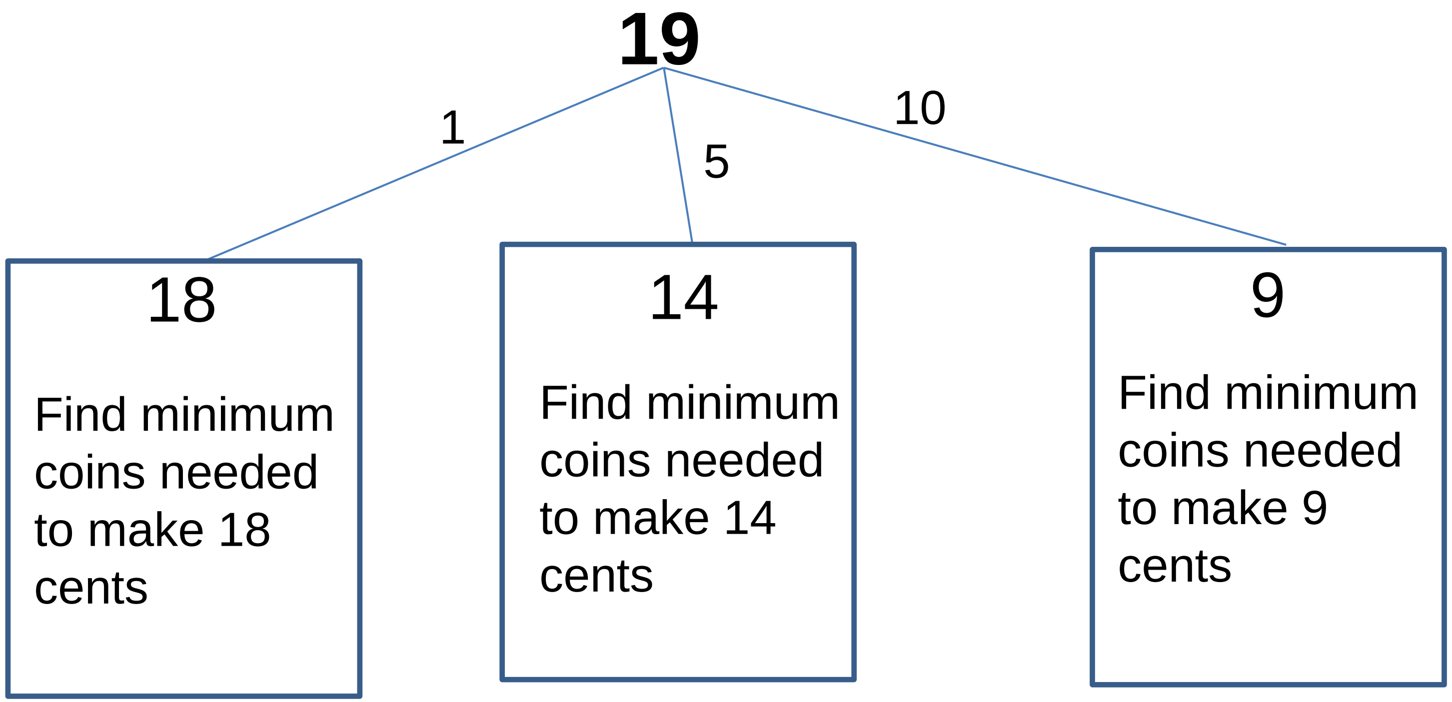


To find the minimum number of coins needed to create 19 cents, we generate **k** subproblems

Change Making Problem

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

k = # denominations

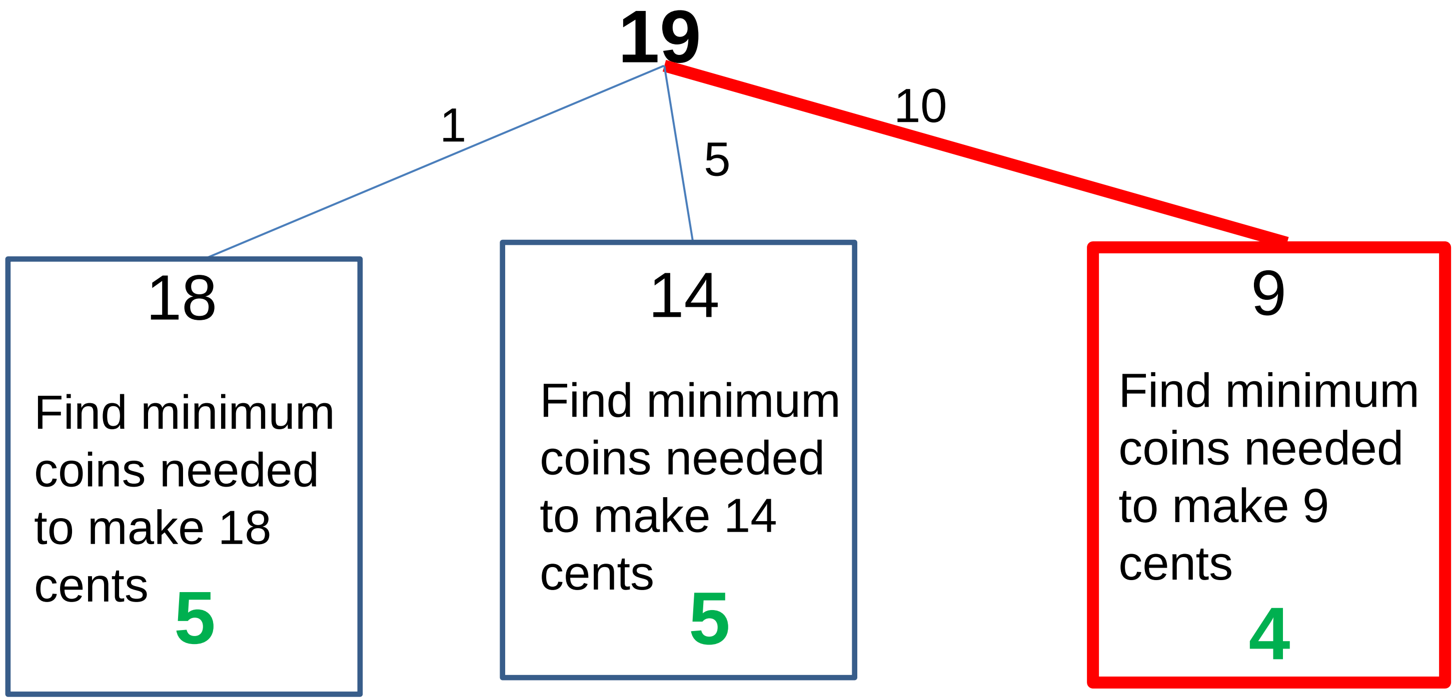


We want to select the **minimum** solution of these three subproblems

Change Making Problem

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

k = # denominations

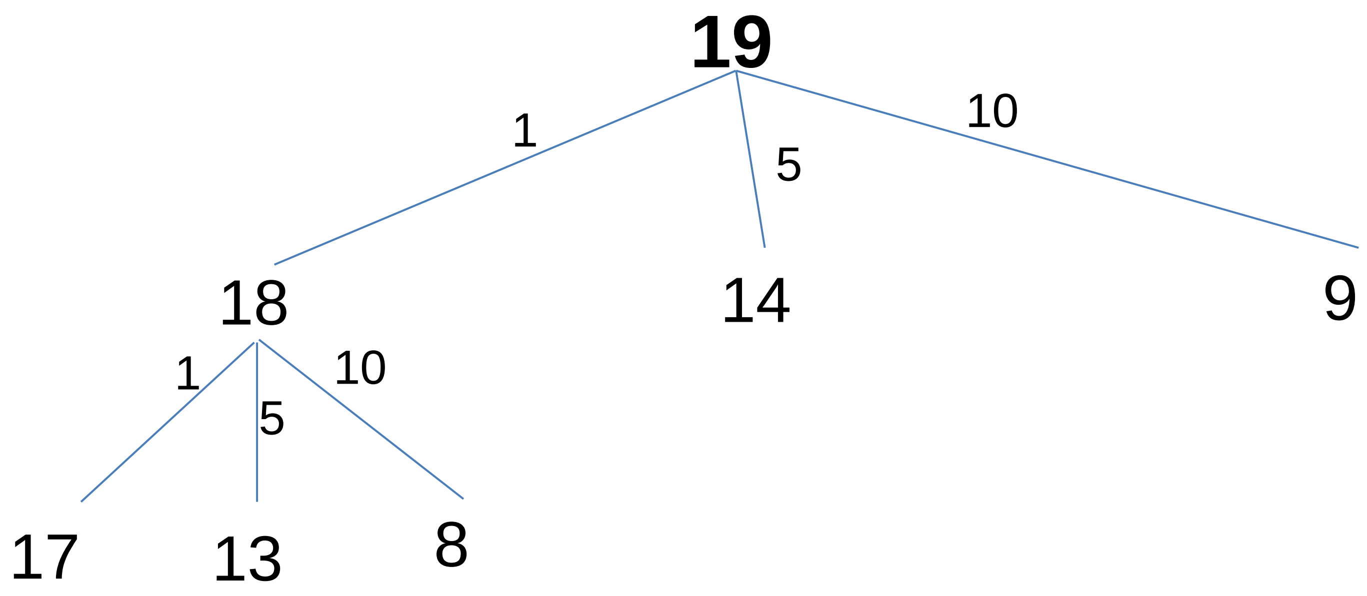


For the solution of our original problem (19), we want to select this branch (one dime used)

Change Making Problem

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

k = # denominations



Find minimum coins needed to make 17 cents

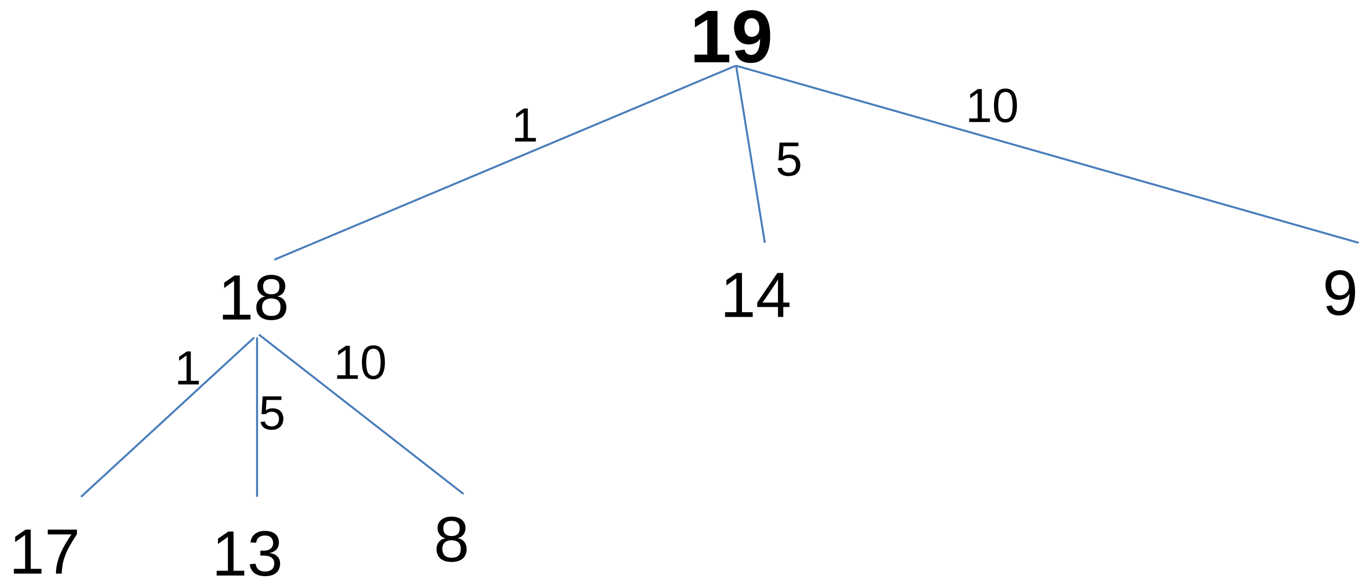
Find minimum coins needed to make 13 cents

Find minimum coins needed to make 8 cents

Change Making Problem

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

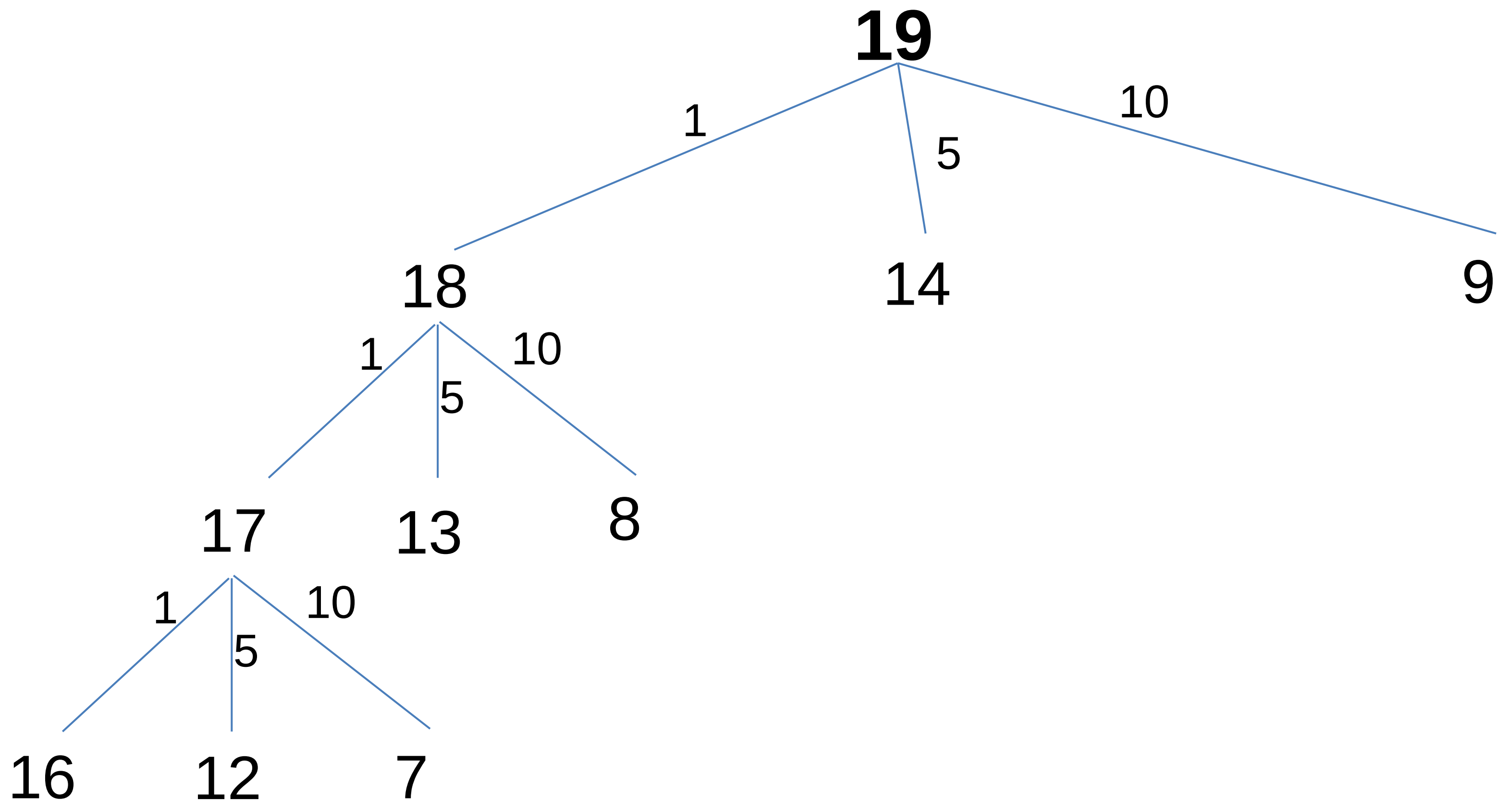
k = # denominations



Change Making Problem

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

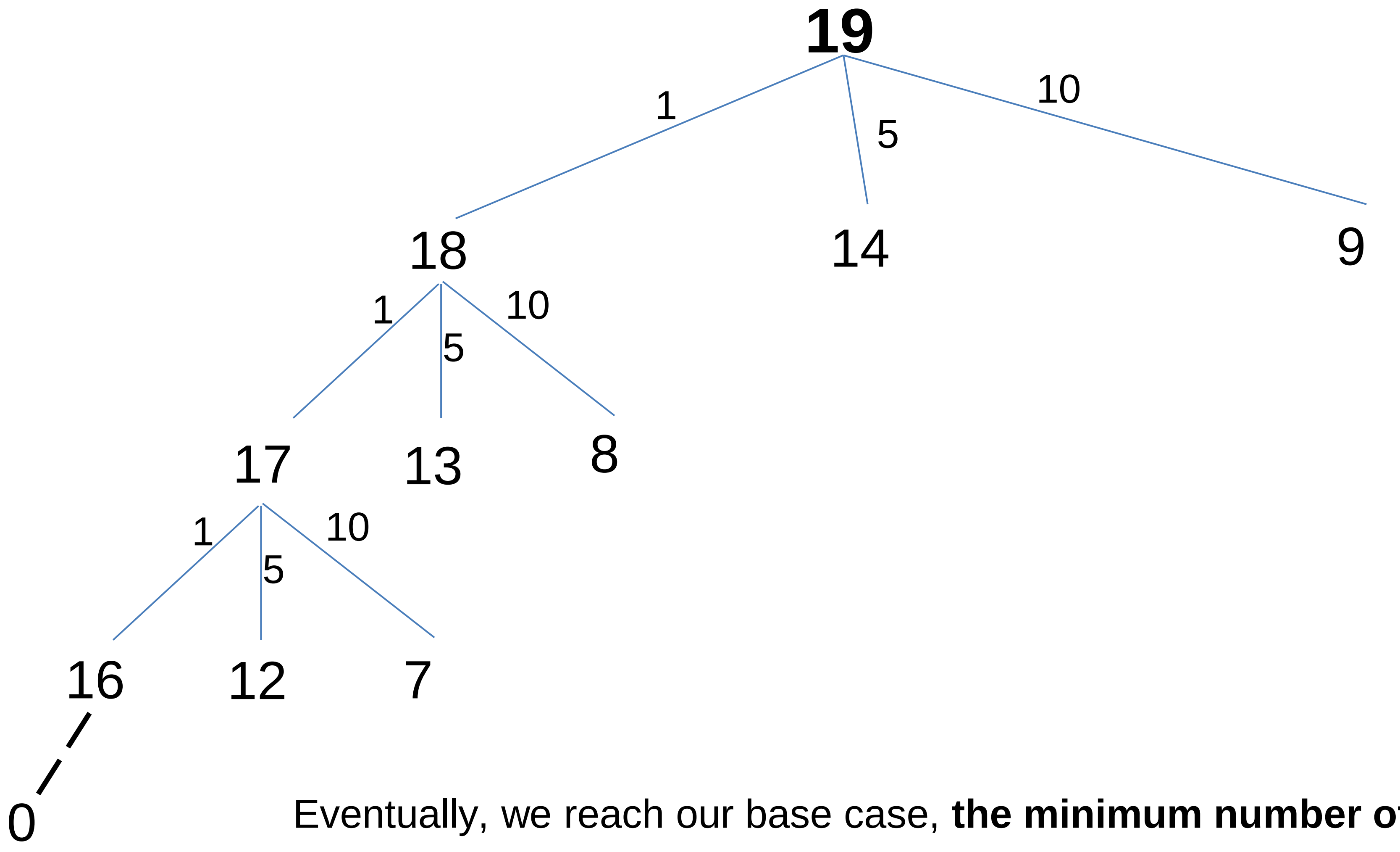
k = # denominations



Change Making Problem

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

k = # denominations

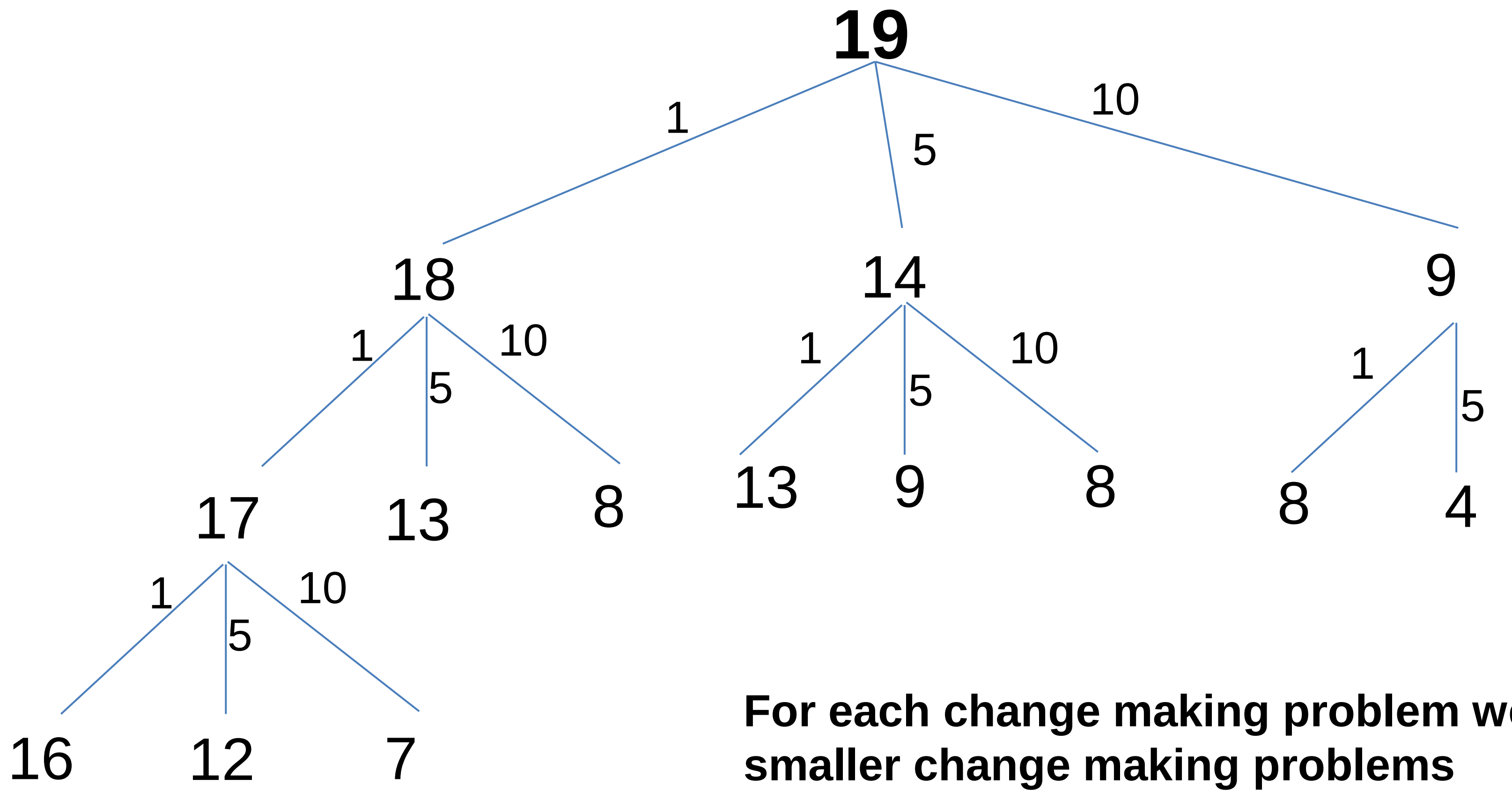


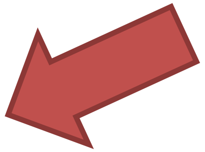
Eventually, we reach our base case, **the minimum number of coins needed to make 0 cents**

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…

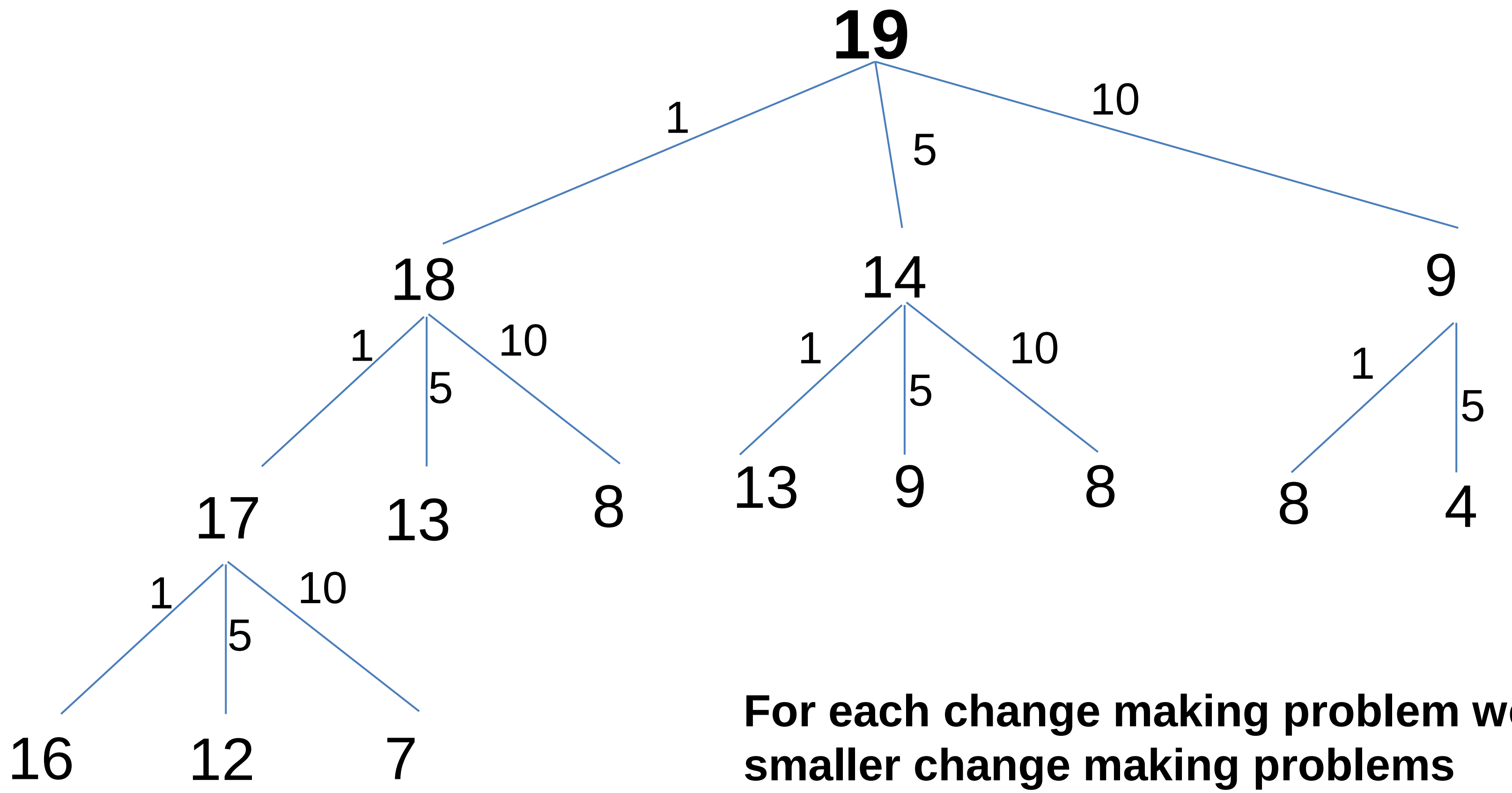
For each change making problem we solve, we must solve at most 3 smaller change making problems

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 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

$$C(p) = \begin{cases} \min\limits_{i:d_i \le p} C(p - d_i) + 1, p > 0 \\ 0, p = 0 \end{cases}$$

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

$$C(p) = \begin{cases} \min\limits_{i:d_i \le p} C(p - d_i) + 1, p > 0 \\ 0, p = 0 \end{cases}$$

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 the yields the minimum value

Change Making Problem

$$C(p) = \begin{cases} \min\limits_{i:d_i \le p} C(p - d_i) + 1, p > 0 \\ 0, p = 0 \end{cases}$$

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 the yields the minimum value

Change Making Problem

$$C(p) = \begin{cases} \min\limits_{i:d_i \le p} C(p - d_i) + 1, p > 0 \\ 0, p = 0 \end{cases}$$

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 the 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;

  else
    min = ∞
    a = ∞

    for each d_i in D
      if (p - d_i) >= 0
        a = min_coins(D, p - d_i )
```

Base Case

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

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;

  else
     min = ∞
     a = ∞

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

Base Case

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

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 = ∞

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

Base Case

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

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 = ∞

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

        return 1 + min
```

Base Case

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

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

Select the branch that has the minimum value

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 d_i in D
            if (p - d_i) >= 0
                a = min_coins(D, p - d_i )
            if a < min
                min = a

    return 1 + min
```

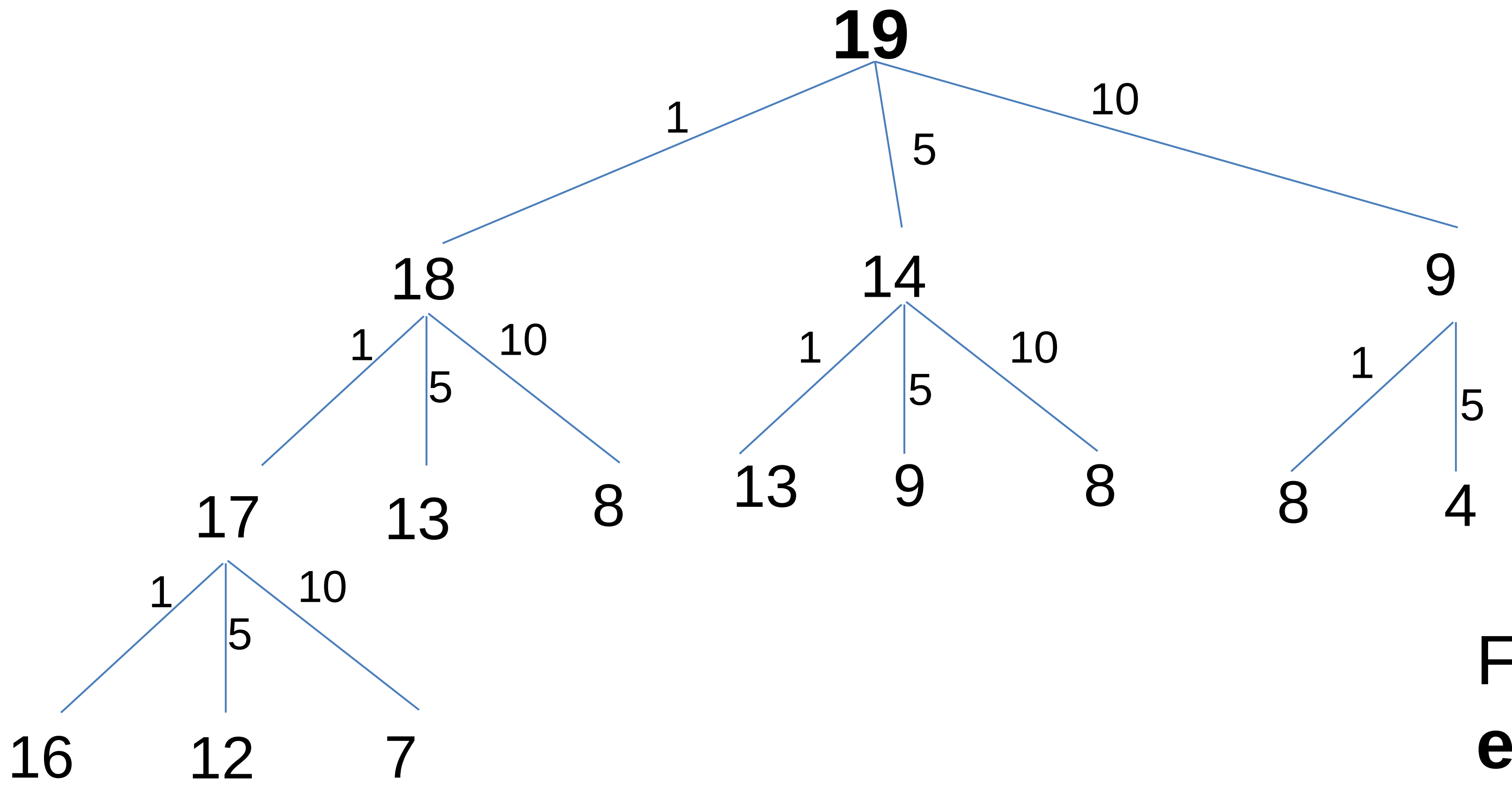
Running time?

Change Making Problem

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

k = # denominations

p = value to make change for



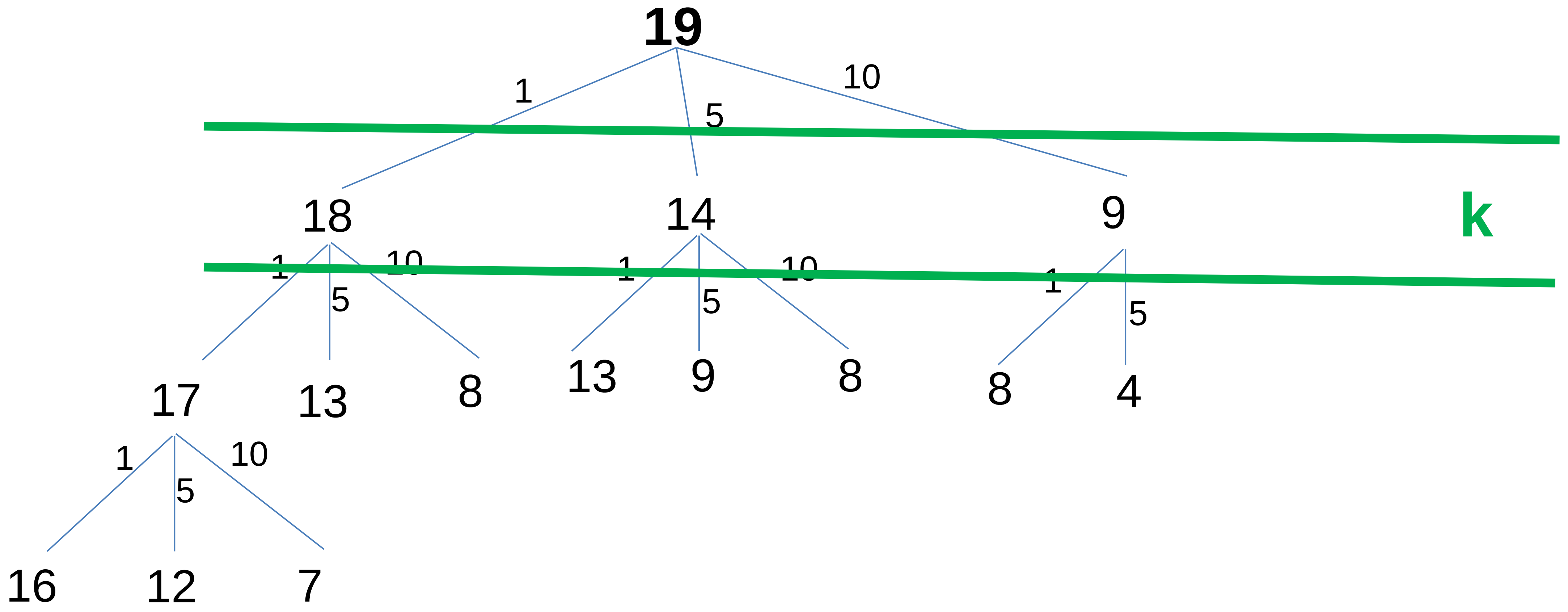
For sufficiently large p, **every permutation** of denominations is included.

Change Making Problem

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

k = # denominations

p = value to make change for

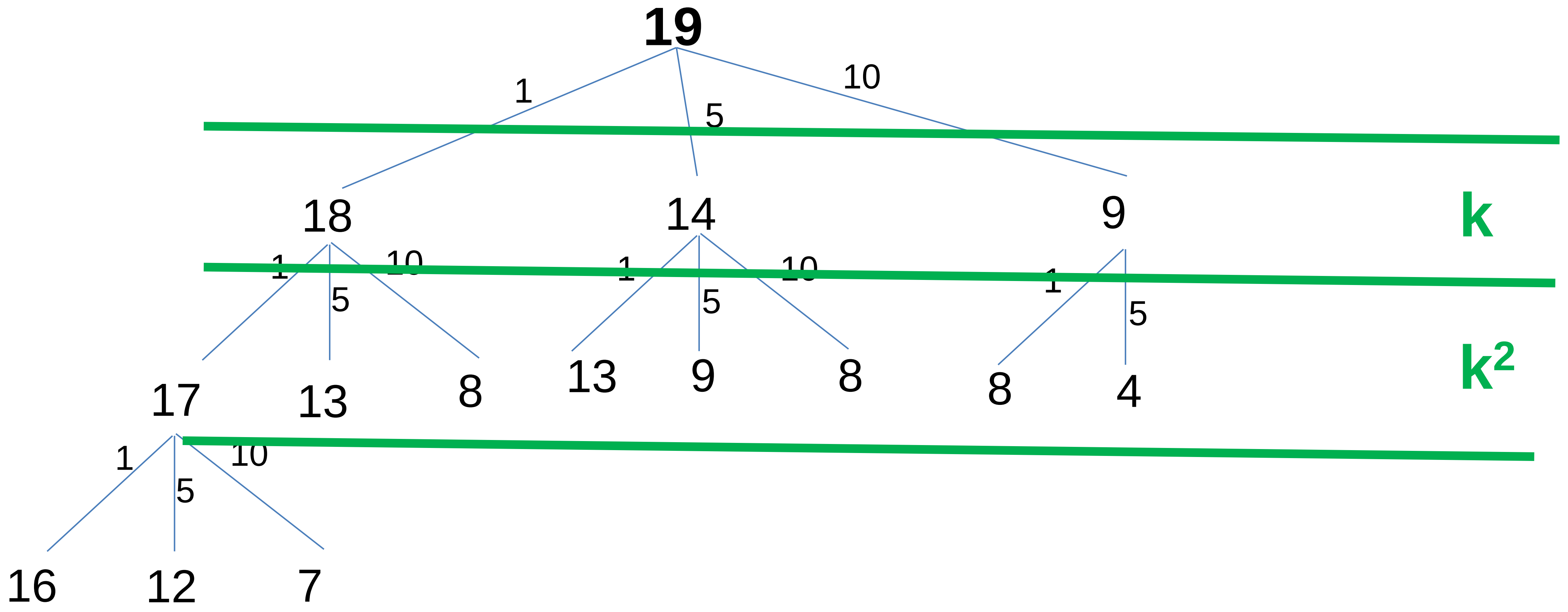


Change Making Problem

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

k = # denominations

p = value to make change for

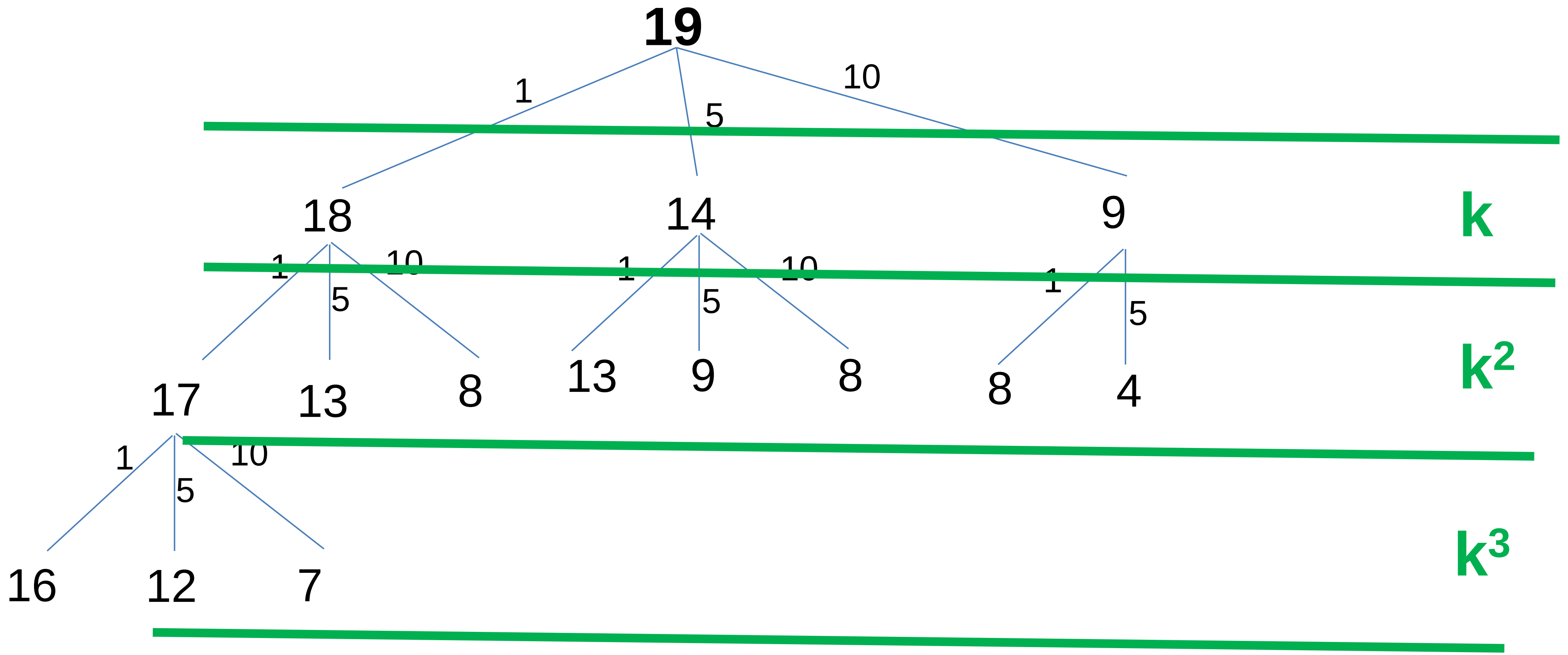


Change Making Problem

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

k = # denominations

p = value to make change for

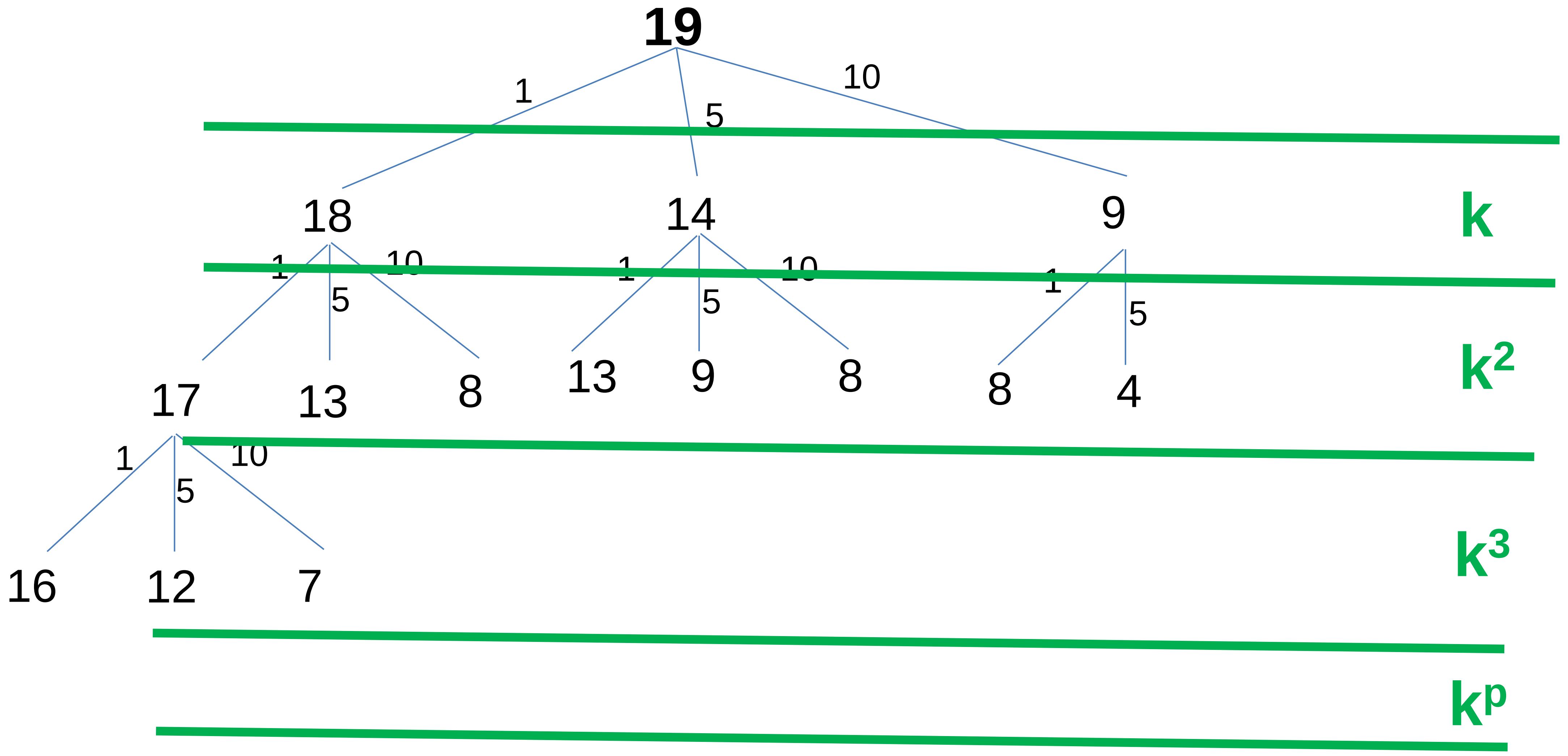


Change Making Problem

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

19

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

19

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

Running time: O(💀)

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

19

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

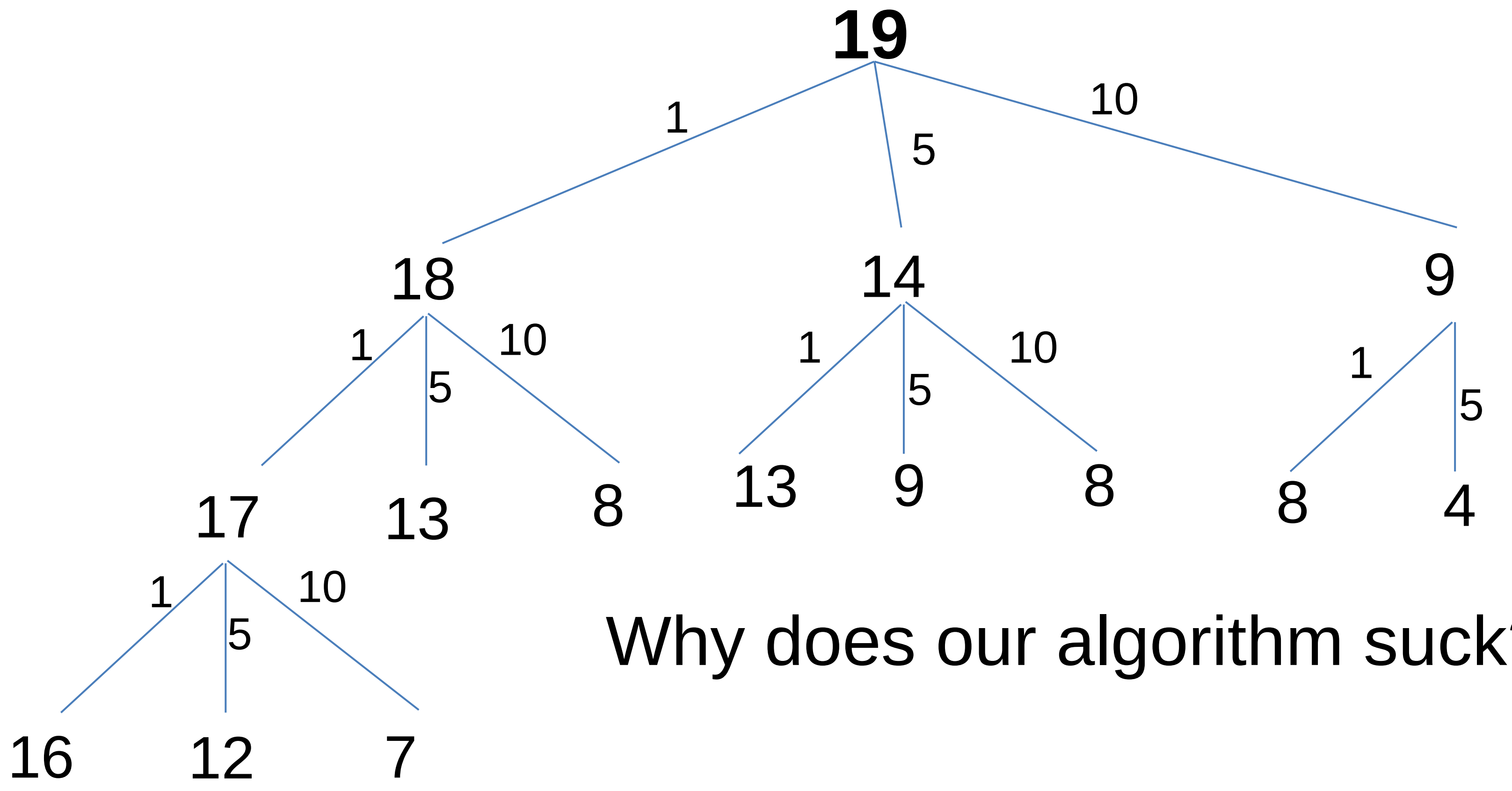
k

k^2

k^3

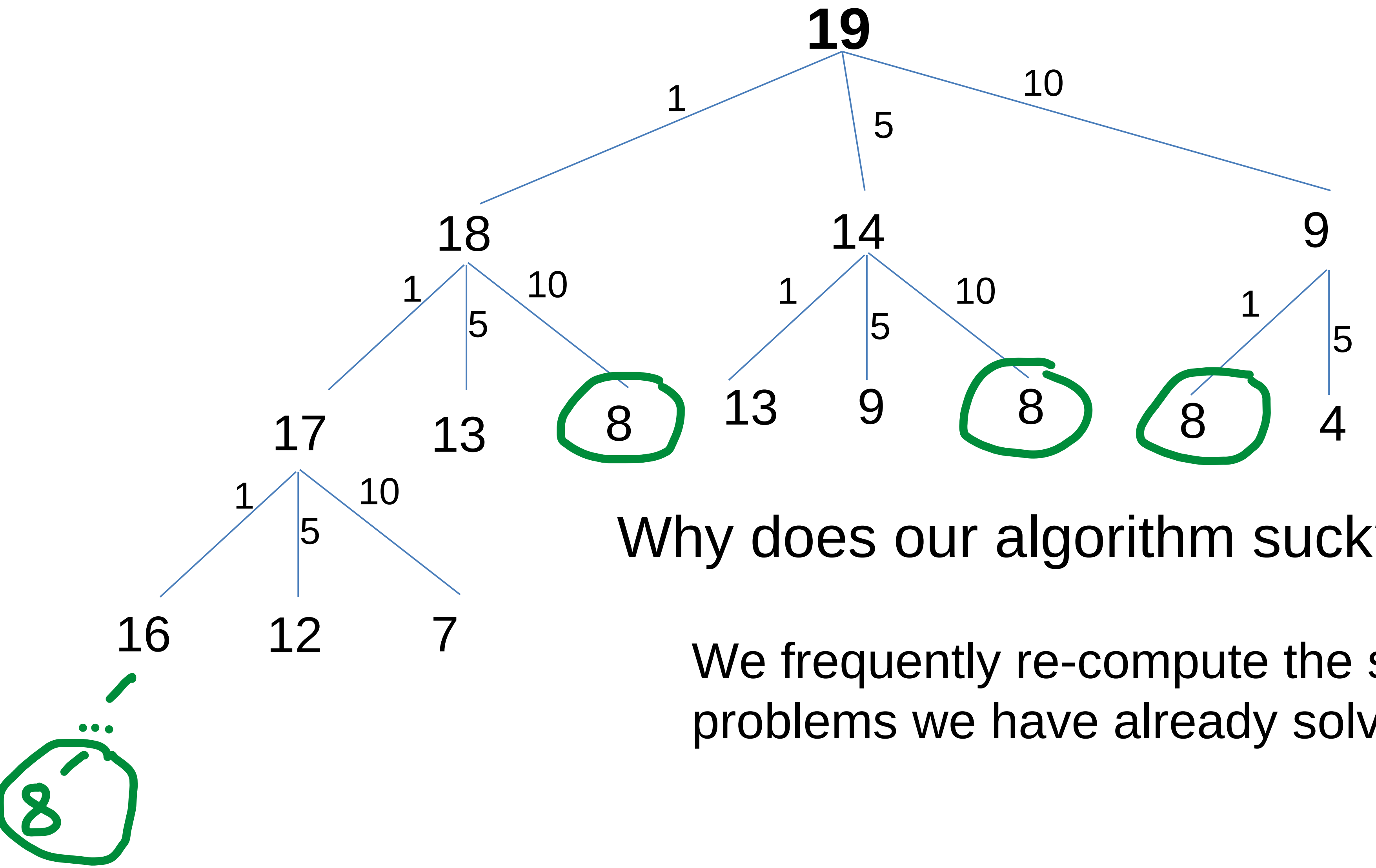
k^p

Change Making Problem



Why does our algorithm suck?

Change Making Problem

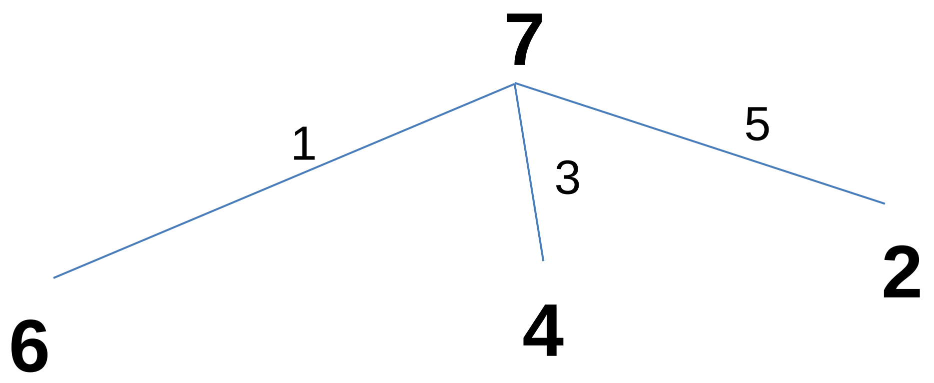


Why does our algorithm suck?

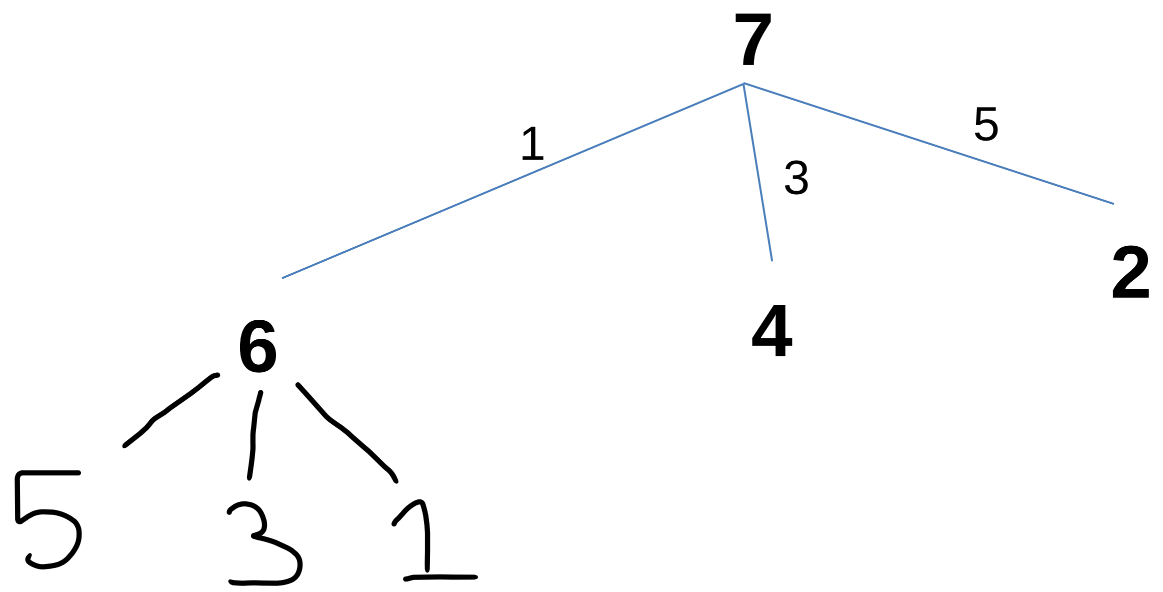
We frequently re-compute the solution to the sub problems we have already solved

Big idea of dynamic programming: use **memoization** to store solutions of sub problems we have already solved, and don’t re compute them

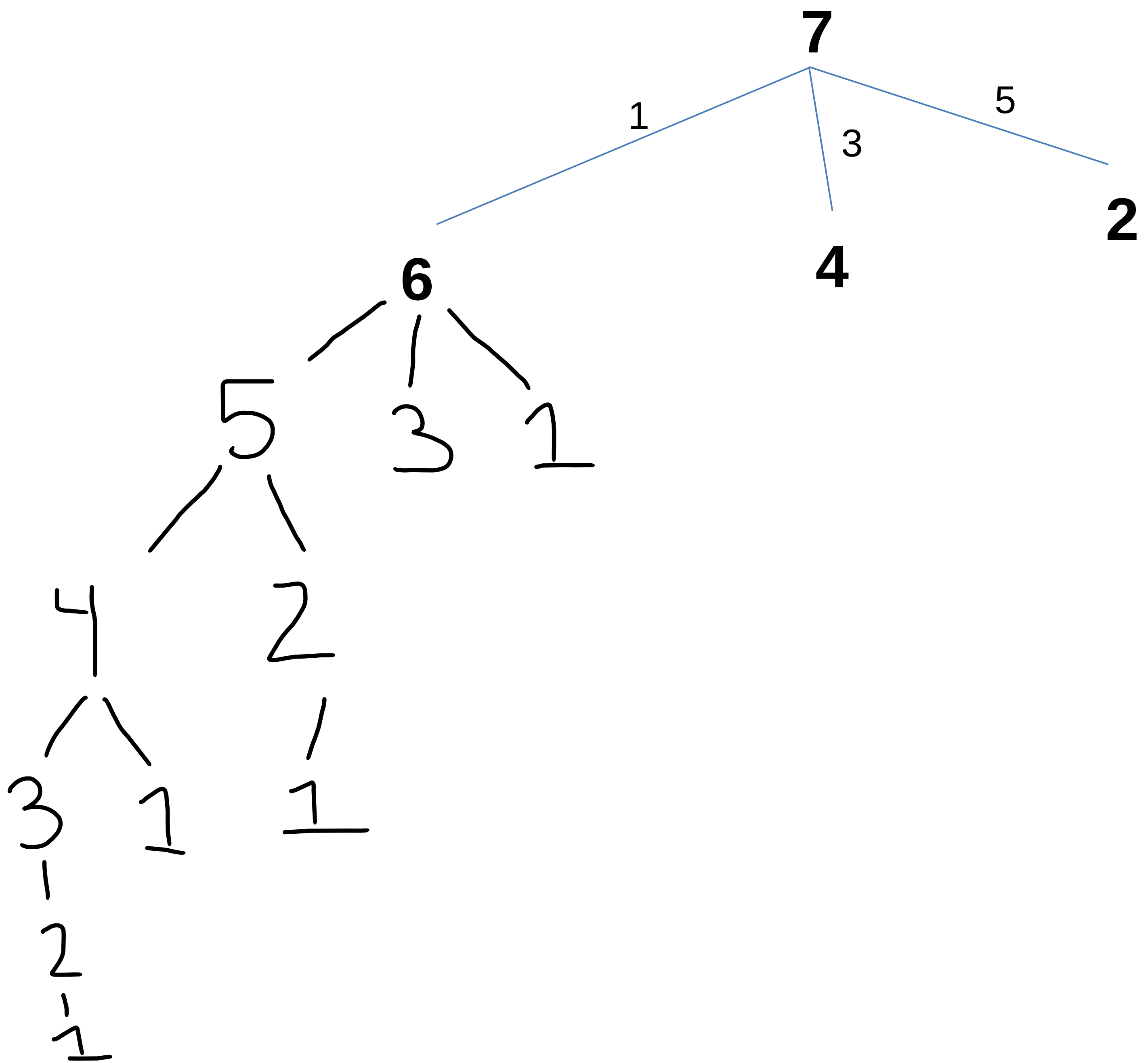
Change Making Problem



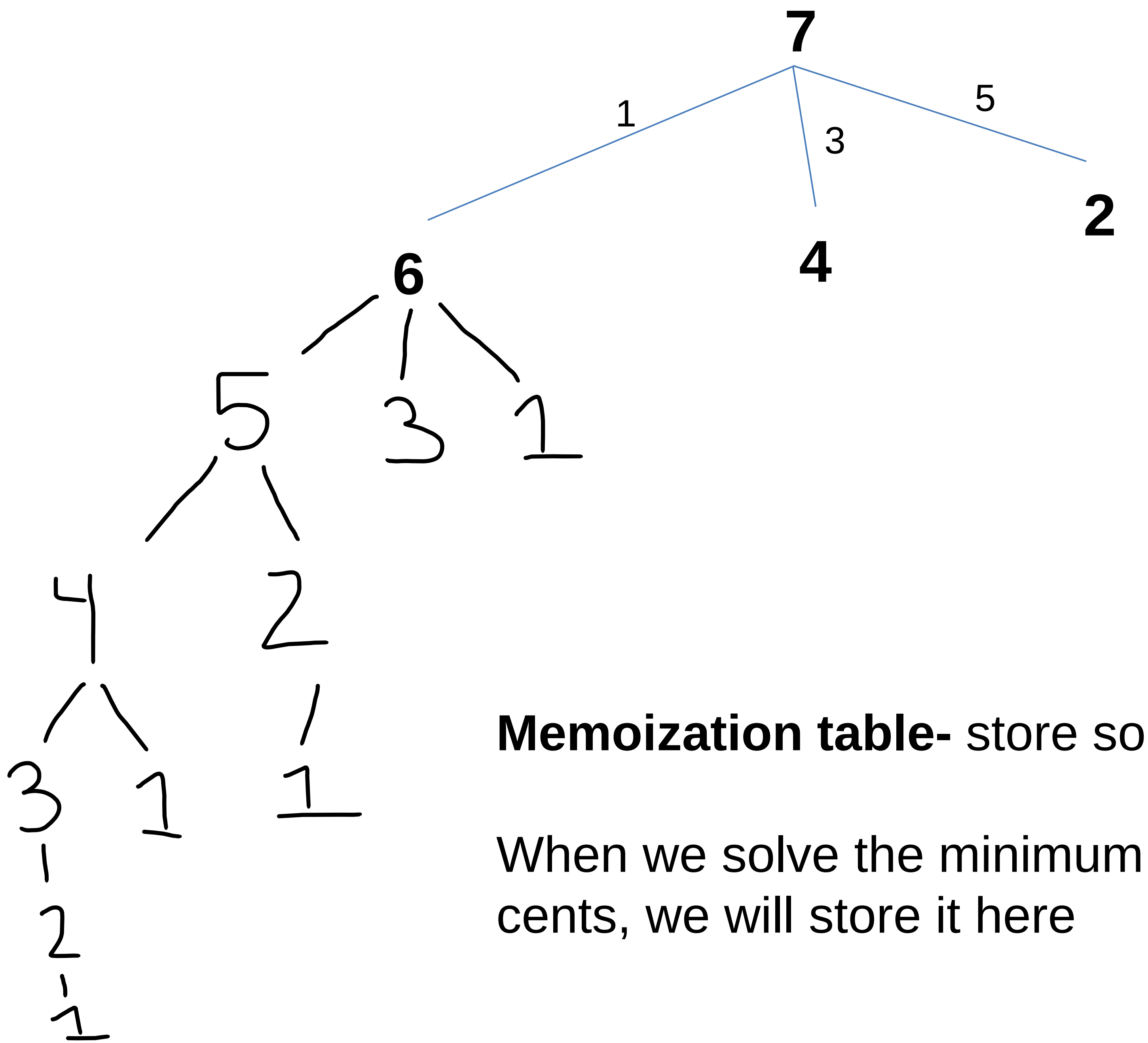
Change Making Problem

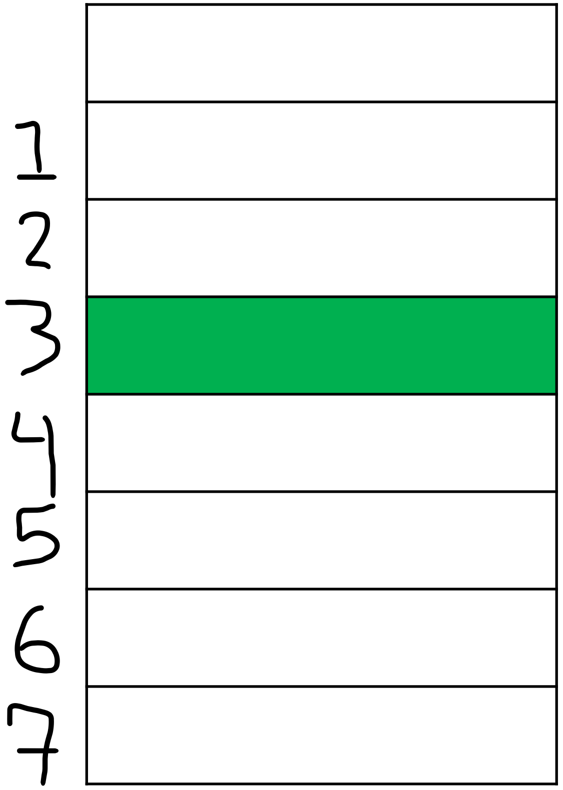


Change Making Problem



Change Making Problem

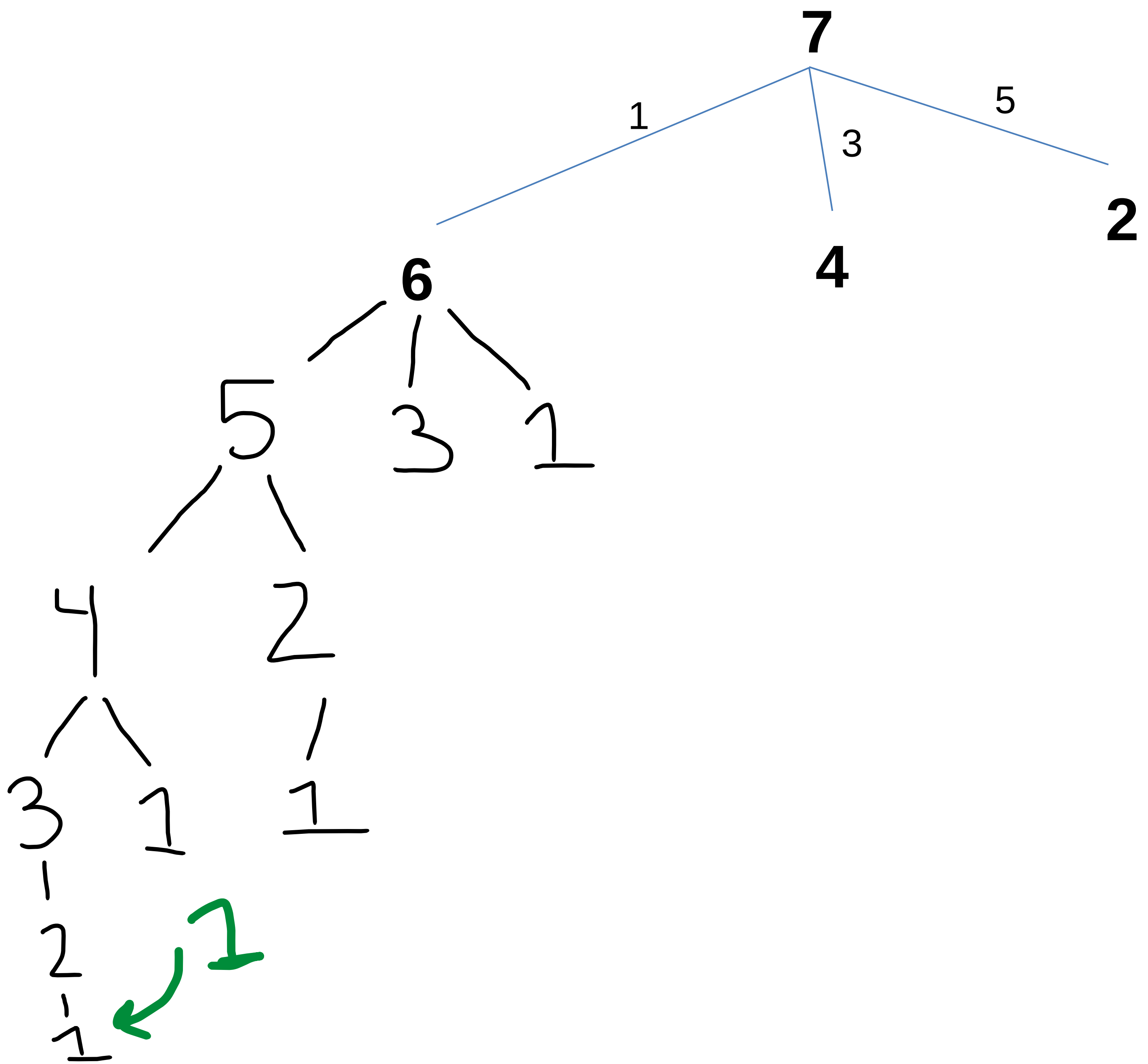




Memoization table- store solutions to sub problems.

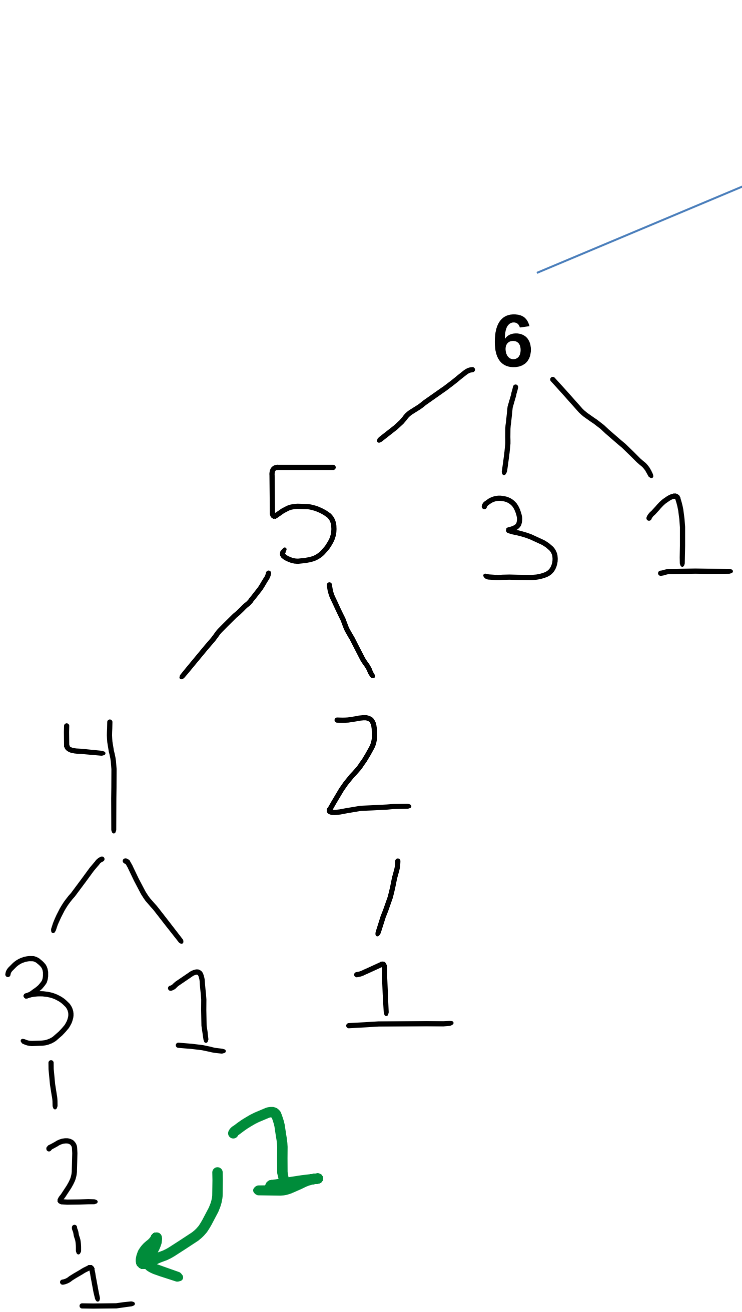
When we solve the minimum coin needed to make three cents, we will store it here

Change Making Problem



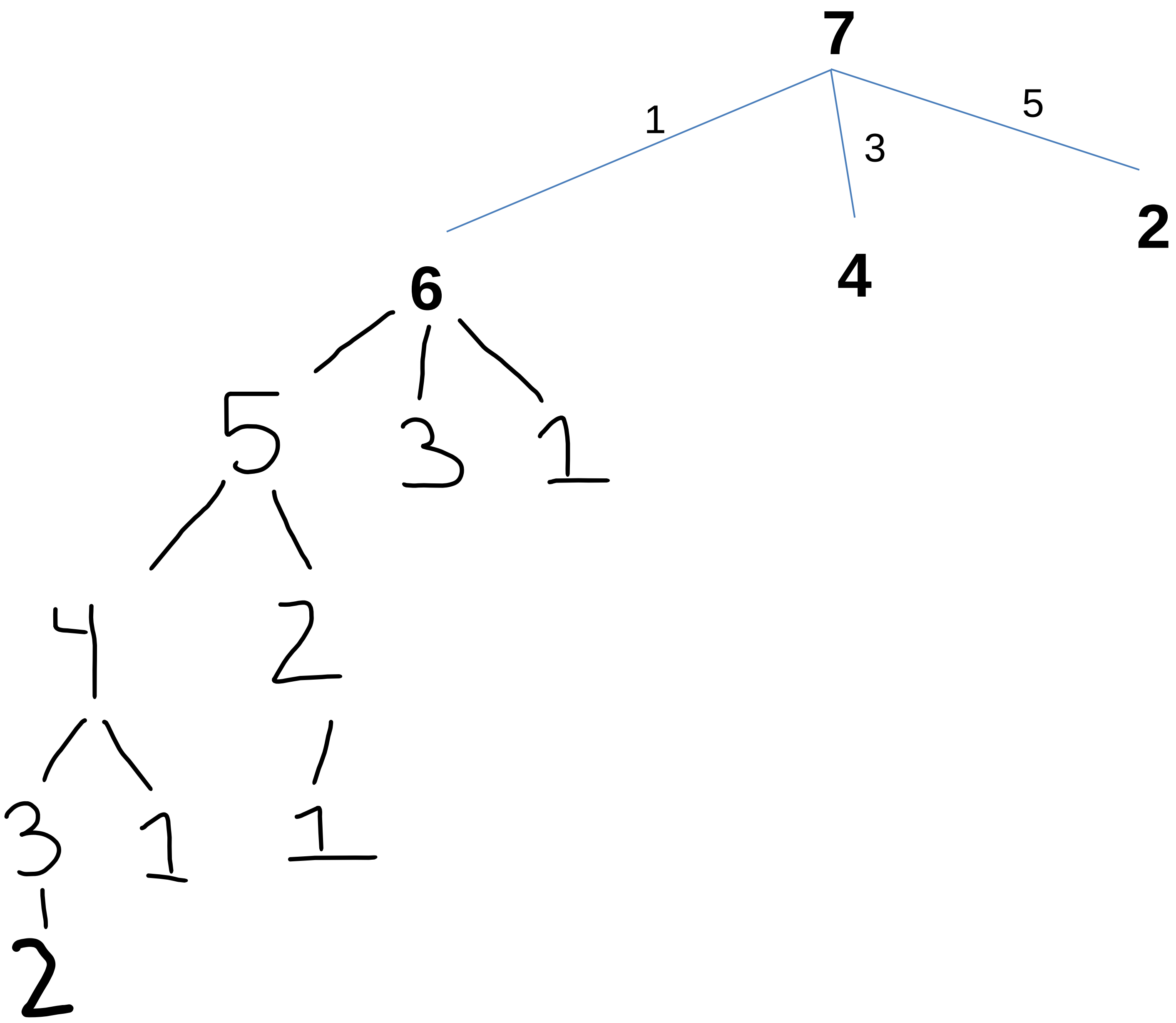
1	
2	
3	
4	
5	
6	
7	

Change Making Problem



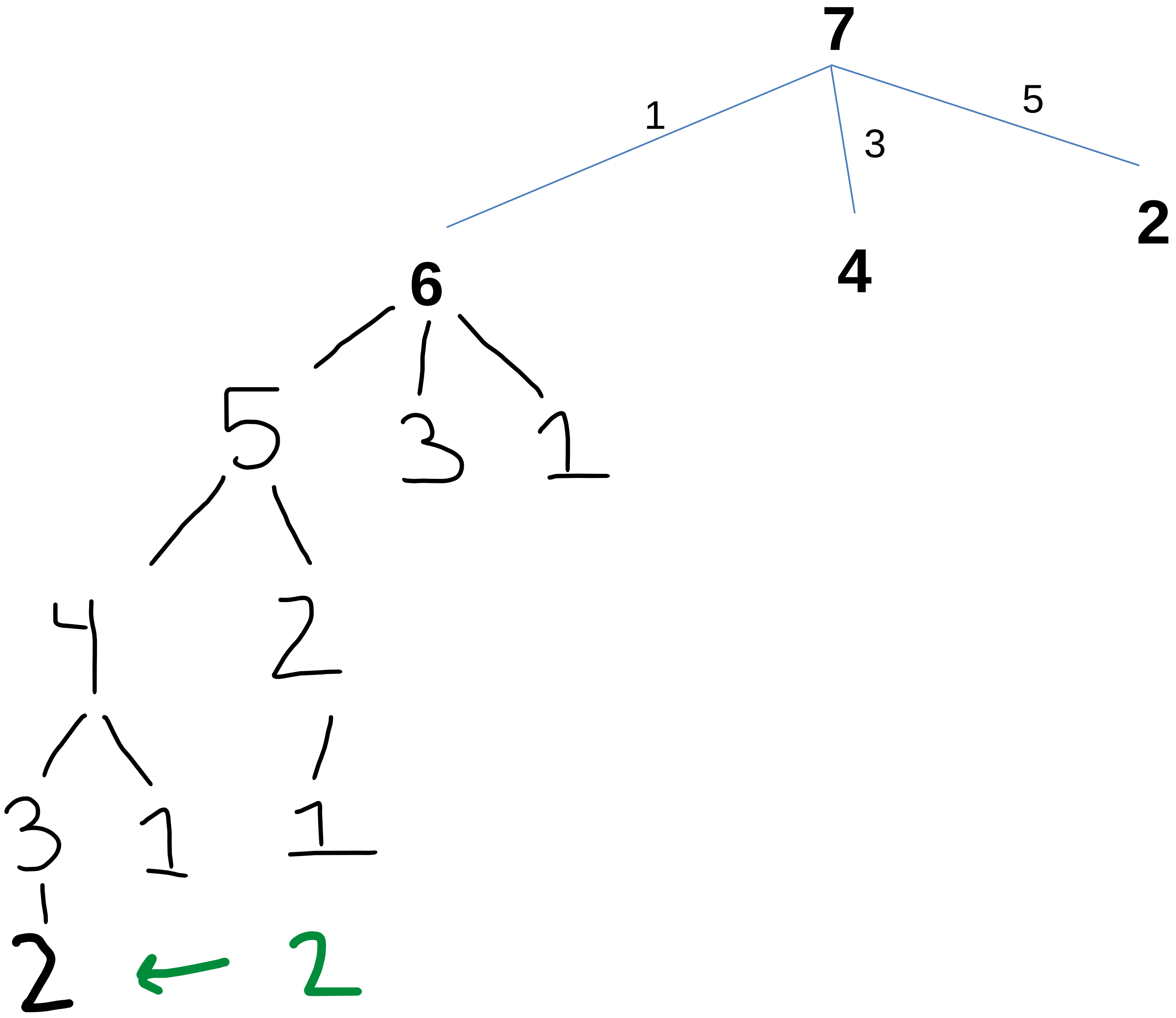
1	1
2	
3	
4	
5	
6	
7	

Change Making Problem



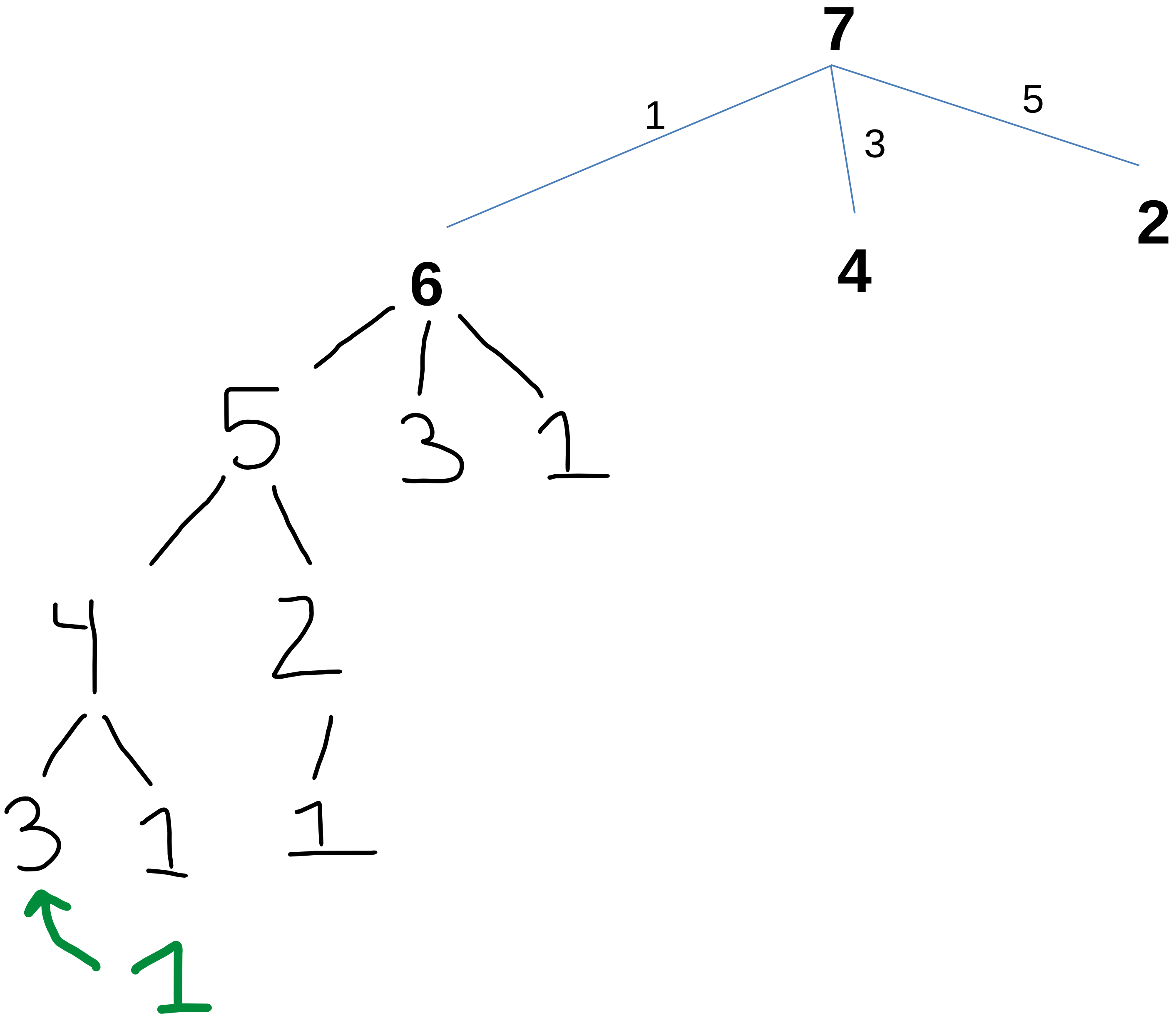
1	1
2	
3	
4	
5	
6	
7	

Change Making Problem



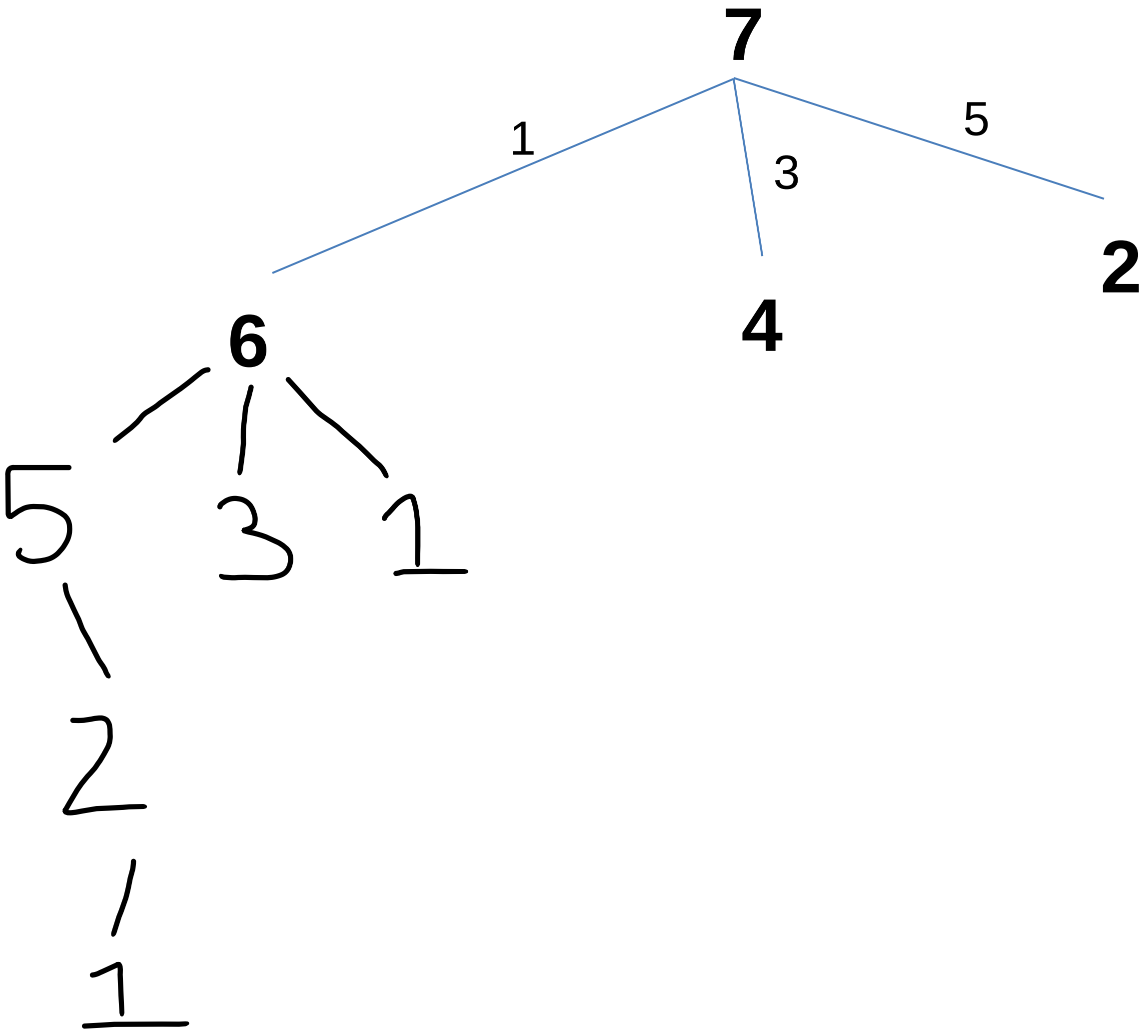
1	1
2	2
3	
4	
5	
6	
7	

Change Making Problem



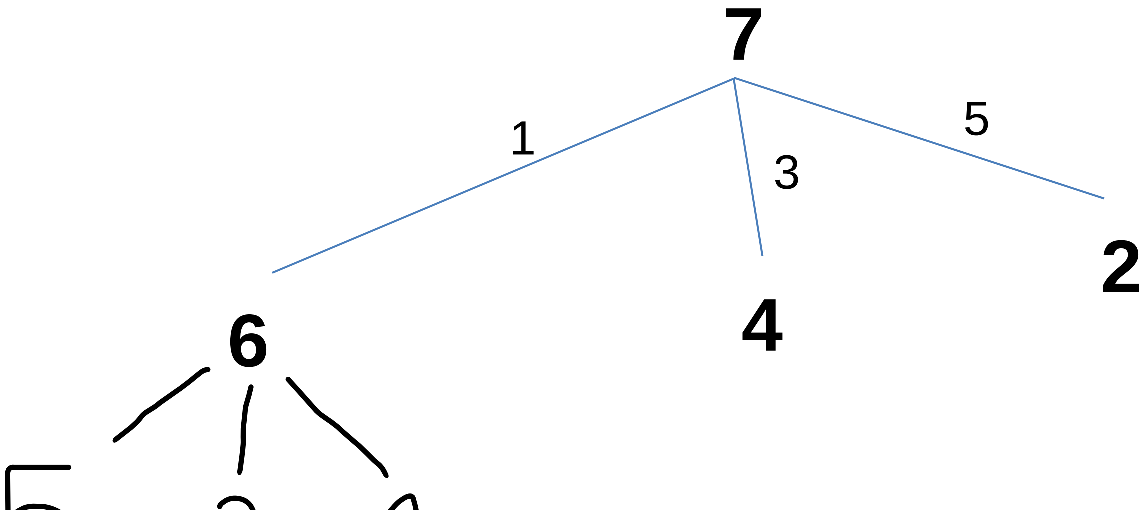
1	1
2	2
3	1
4	
5	
6	
7	

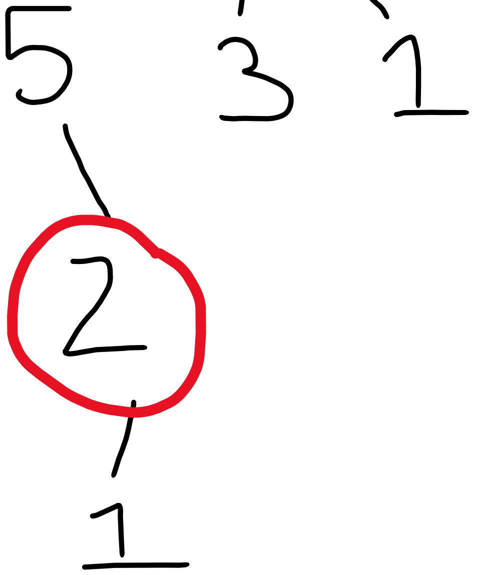
Change Making Problem



1	1
2	2
3	1
4	2
5	
6	
7	

Change Making Problem



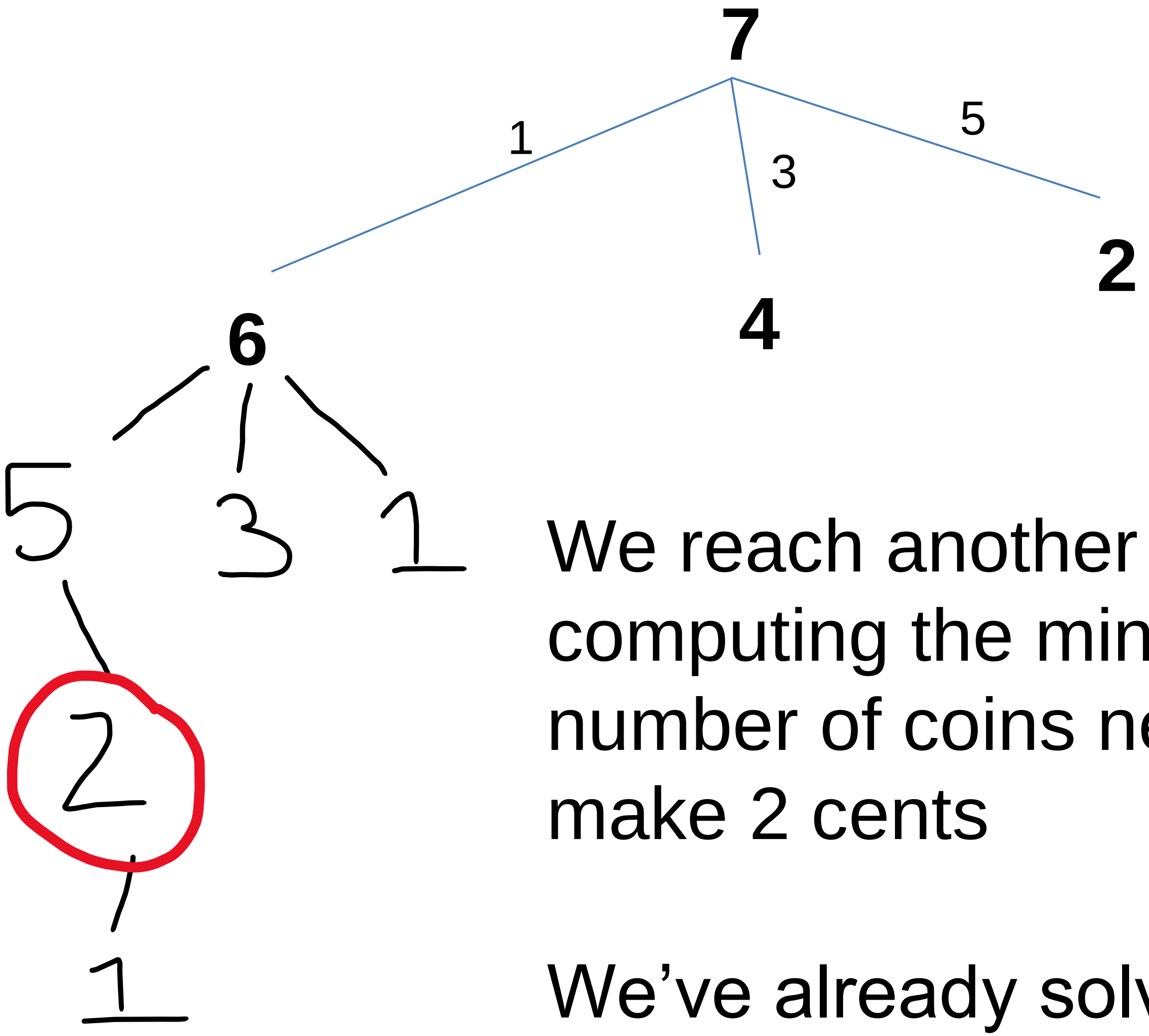


We reach another instance of computing the minimum number of coins needed to make 2 cents

We've already solved this. No need to recompute. Just use value from DP table

1	1
2	2
3	1
4	2
5	
6	
7	

Change Making Problem

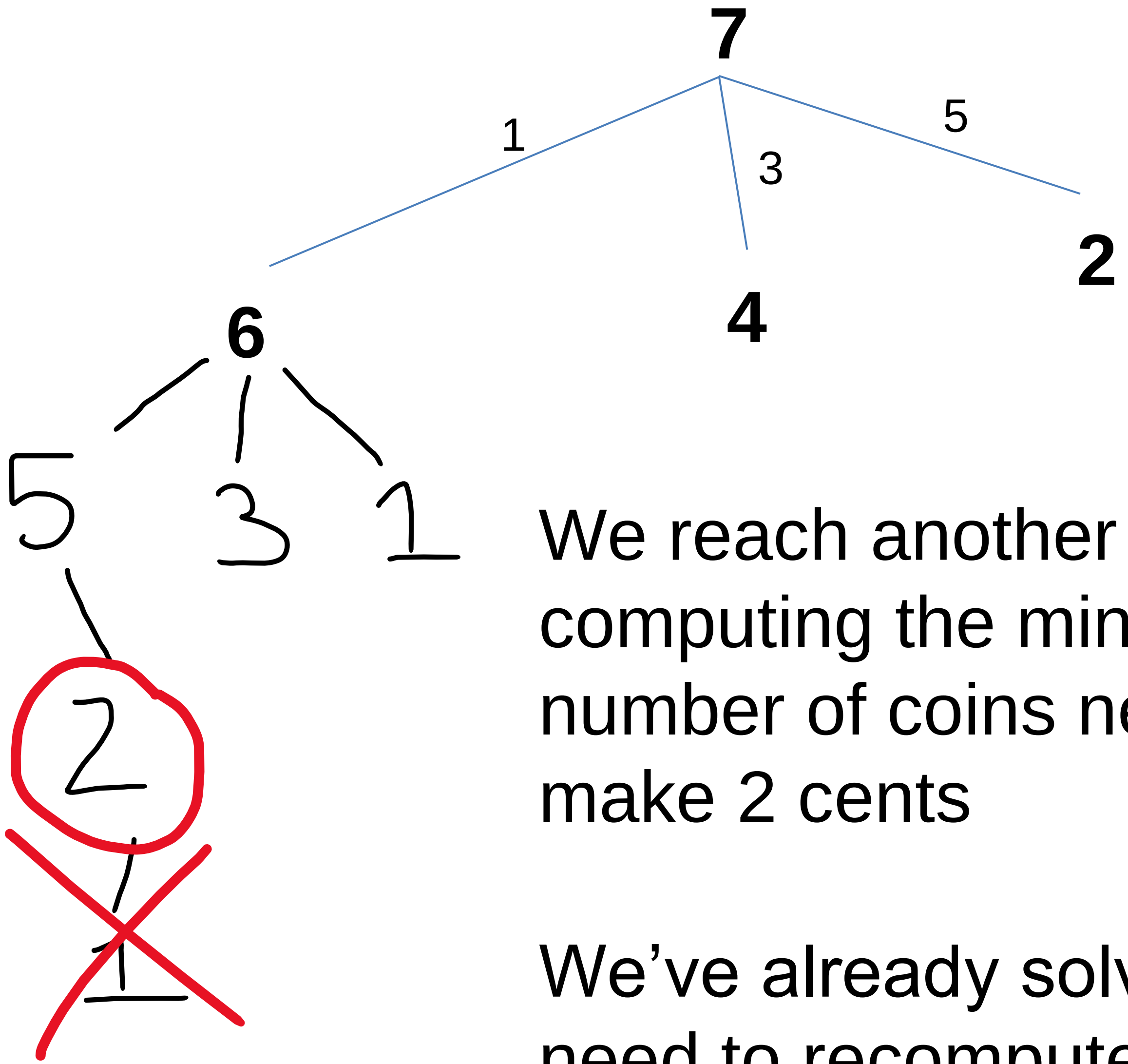


We reach another instance of computing the minimum number of coins needed to make 2 cents

We’ve already solved this. No need to recompute. Just use value from DP table

1	1
2	2
3	1
4	2
5	
6	
7	

Change Making Problem

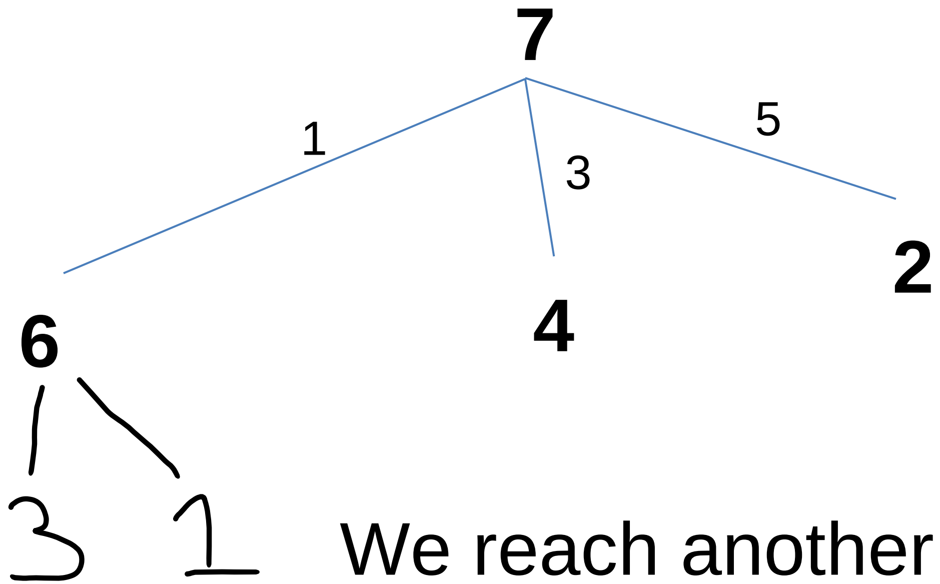


We reach another instance of computing the minimum number of coins needed to make 2 cents

We've already solved this. No need to recompute. Just use value from DP table

1	1
2	2
3	1
4	2
5	
6	
7	

Change Making Problem

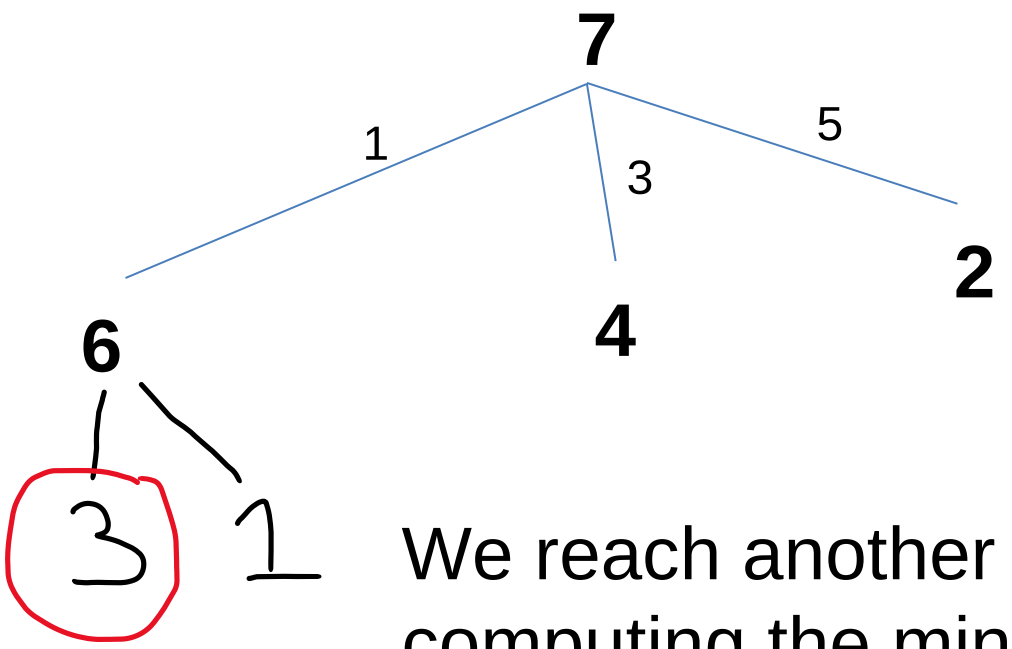


We reach another instance of computing the minimum number of coins needed to make 2 cents

We've already solved this. No need to recompute. Just use value from DP table

1	1
2	2
3	1
4	2
5	1
6	
7	

Change Making Problem

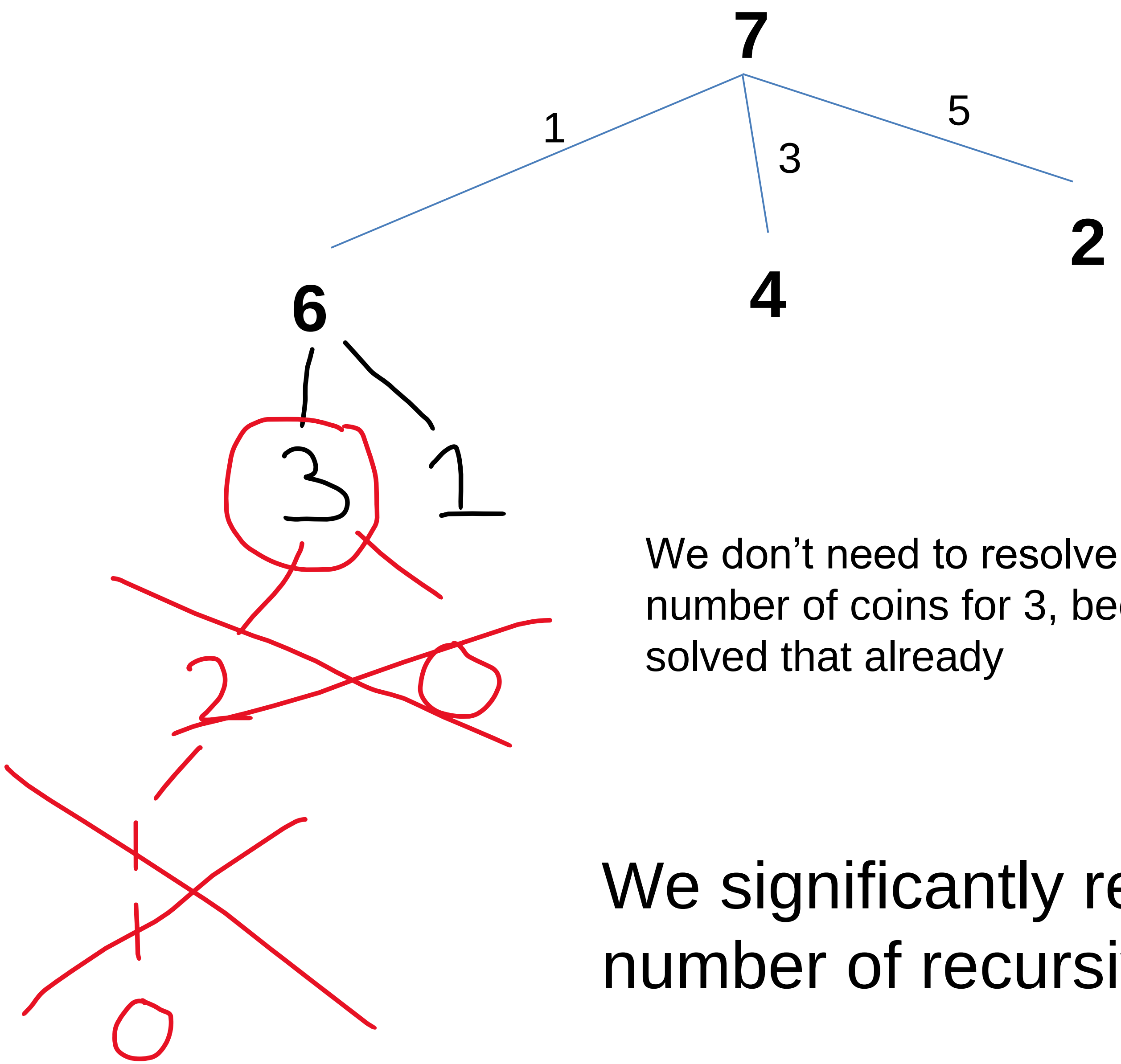


We reach another instance of computing the minimum number of coins needed to make 2 cents

We've already solved this. No need to recompute. Just use value from DP table

1	1
2	2
3	1
4	2
5	1
6	
7	

Change Making Problem



We don't need to resolve the minimum number of coins for 3, because we have solved that already

1	1
2	2
3	1
4	2
5	1
6	
7	

We significantly reduce the number of recursive calls made

Change Making Problem

7

By the end, we will have a data structure that holds the solutions to all the sub problems

Don’t recurse if we already have the solution → **Smart Recursion**

1	1
2	2
3	1
4	2
5	1
6	2
7	3

Top-down vs bottom up

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   for m = 1 to p
      min = ∞
      for d_i ≤ m
         if Chng[m − d_i] + 1 < min
            min = Chng[m − d_i] + 1
      Chng[m] = min
   return Chng[p]
```

Bottom up = don't use recursion

Fill out table using a for a loop of some kinds.

Once table is filled out, query the necessary pieces of information

Also called **tabulation**

1	1
2	2
3	1
4	2
5	1
6	2
7	3

Change Making Problem (Top down approach)

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

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

        return 1 + min
```

Rod Cutting