Dynamic Programming
CSCI 432

Dynamic Programing

{1 broom + 2 “cool rocks” + 1 chair} are the smallest number of items possible that cost $17. One broom costs $10. What can you conclude?

Dynamic Programing

{1 broom + 2 “cool rocks” + 1 chair} are the smallest number of items possible that cost $17. One broom costs $10. What can you conclude?

{2 “cool rocks” + 1 chair} are the smallest number of items possible that cost $7.

Dynamic Programing

{1 broom + 2 “cool rocks” + 1 chair} are the smallest number of items possible that cost $17. One broom costs $10. What can you conclude?

{2 “cool rocks” + 1 chair} are the smallest number of items possible that cost $7.

A problem exhibits optimal substructure if removing part of an optimal solution results in an optimal solution to a smaller problem.

Dynamic Programing

{1 broom + 2 “cool rocks” + 1 chair} are the smallest number of items possible that cost $17. One broom costs $10. What can you conclude?

{2 “cool rocks” + 1 chair} are the smallest number of items possible that cost $7.

A problem exhibits optimal substructure if removing part of an optimal solution results in an optimal solution to a smaller problem.

Central tenant of Dynamic Programming:
Leverage optimal sub-structure.

Fundamental Algorithmic Techniques

"Sledgehammers":

- Dynamic Programming
- Linear Programming

Precision Tools:

- Greedy
- Randomization
- Reductions

Dynamic Programing vs Divide and Conquer

Dynamic Programming

- Optimal substructure.
- Overlapping subproblems.

Divide and Conquer

- Independent, recursive subproblems (mergesort).

Dynamic Programming Process:

1. Characterize structure of optimal solution.
2. Recursively define value of optimal solution.
3. Compute value of optimal solution.
4. Construct optimal solution from computed information.

Making Change

How can I represent 37 cents with the smallest number of coins?

Making Change

How can I represent 37 cents with the smallest number of coins?

Quarter, dime, two pennies (25 + 10 + 2 = 37) – four coins.

Algorithm: ?

Making Change

How can I represent 37 cents with the smallest number of coins?

Quarter, dime, two pennies (25 + 10 + 2 = 37) – four coins.

Algorithm: Max quarters + max dimes + ...

What if there were also an 18-cent coin?

Making Change

How can I represent 37 cents with the smallest number of coins?

Quarter, dime, two pennies (25 + 10 + 2 = 37) – four coins.

Algorithm: Max quarters + max dimes + ...

What if there were also an 18-cent coin?

Two 18-cent coins, penny – three coins.

Making Change

In general, suppose a country has denominations:

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

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

Making Change

In general, suppose a country has denominations:

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

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

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

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

Making Change

In general, suppose a country has denominations:

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

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

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

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

Can we characterize $C(p)$ in terms of $C(p - x)$?

$C(p) =$???

Making Change

In general, suppose a country has denominations:

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

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

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

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

Can we characterize $C(p)$ in terms of $C(p - x)$?

$$C(p) = \; C(p - x) + 1$$

Making Change

In general, suppose a country has denominations:

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

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

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

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

But we don't know what actual coin is in the optimal solution.

$$C(p) = \; C(p - x) + 1$$

Making Change

In general, suppose a country has denominations:

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

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

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

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

So, we'll check all possibilities.

But we don't know what actual coin is in the optimal solution.

$$C(p) = \min_{d_i \leq p} C(p - d_i) + 1$$

Making Change

In general, suppose a country has denominations:

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

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

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

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

So, we'll check all possibilities.

But we don't know what actual coin is in the optimal solution.

$$C(p) = \min_{d_i \leq p} C(p - d_i) + 1$$

Least change for 20 cents is minimum of:

- least change for 20-10 = 10 cents
- least change for 20-5 = 15 cents
- least change for 20-1 = 19 cents

Making Change

In general, suppose a country has denominations:

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

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

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

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

So, we'll check all possibilities.

But we don't know what actual coin is in the optimal solution.

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

Making Change

In general, suppose a country has denominations:

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

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

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

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

So, we'll check all possibilities.

But we don't know what actual coin is in the optimal solution.

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

Making Change – Recursive

```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Making Change – Recursive

```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

Making Change – Recursive

```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for

Making Change – Recursive

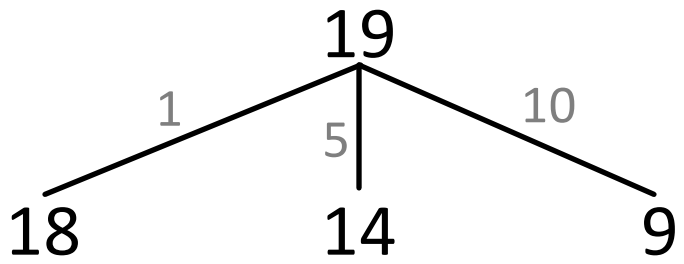
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

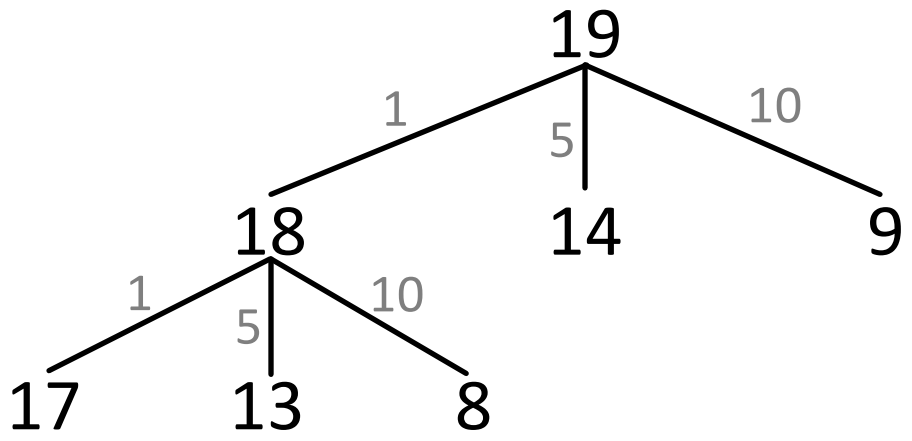
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

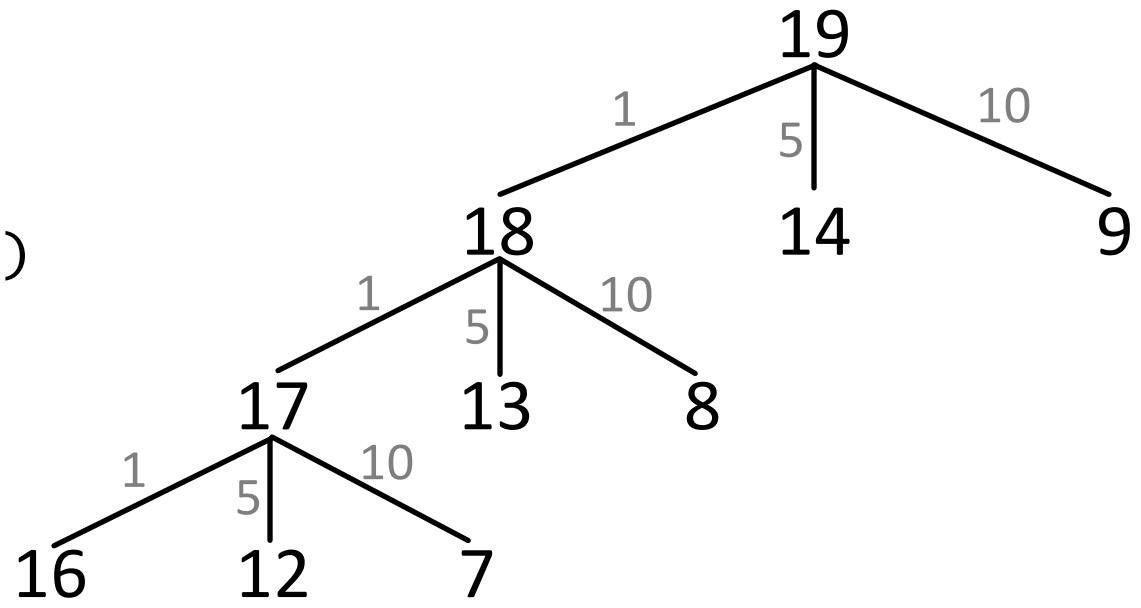
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

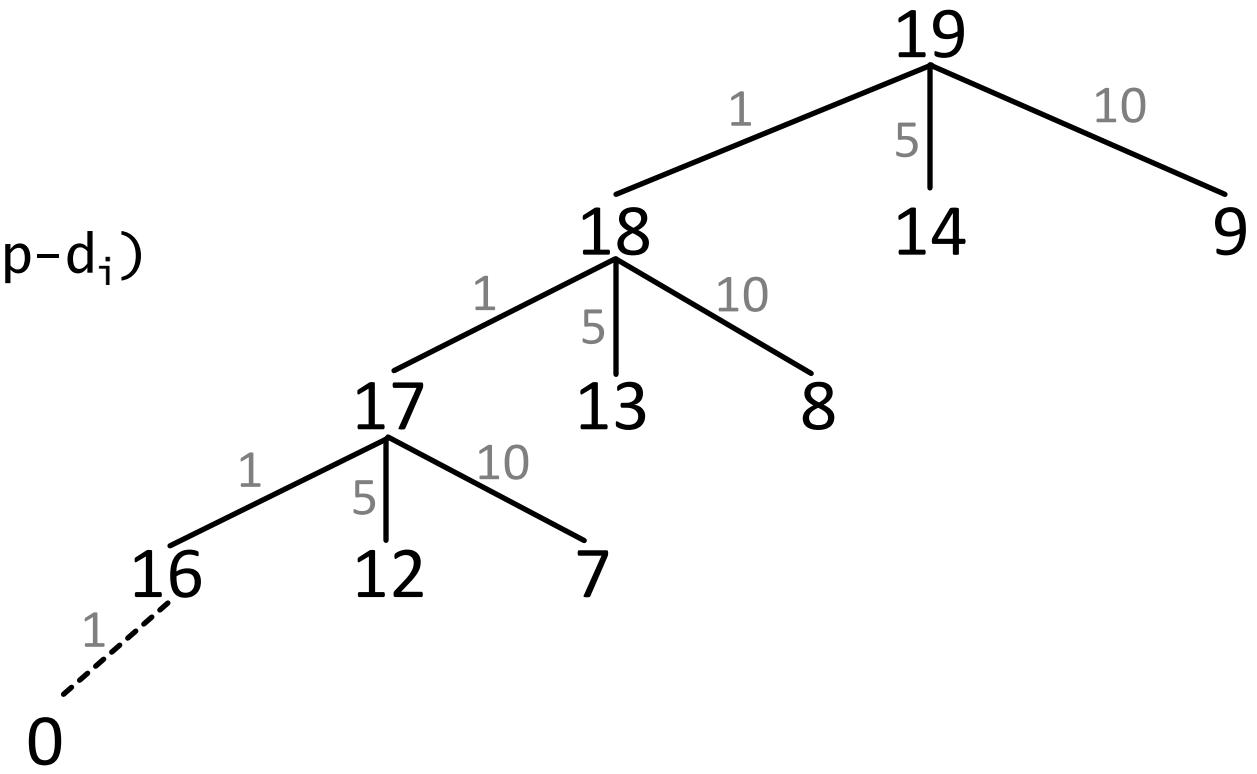
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

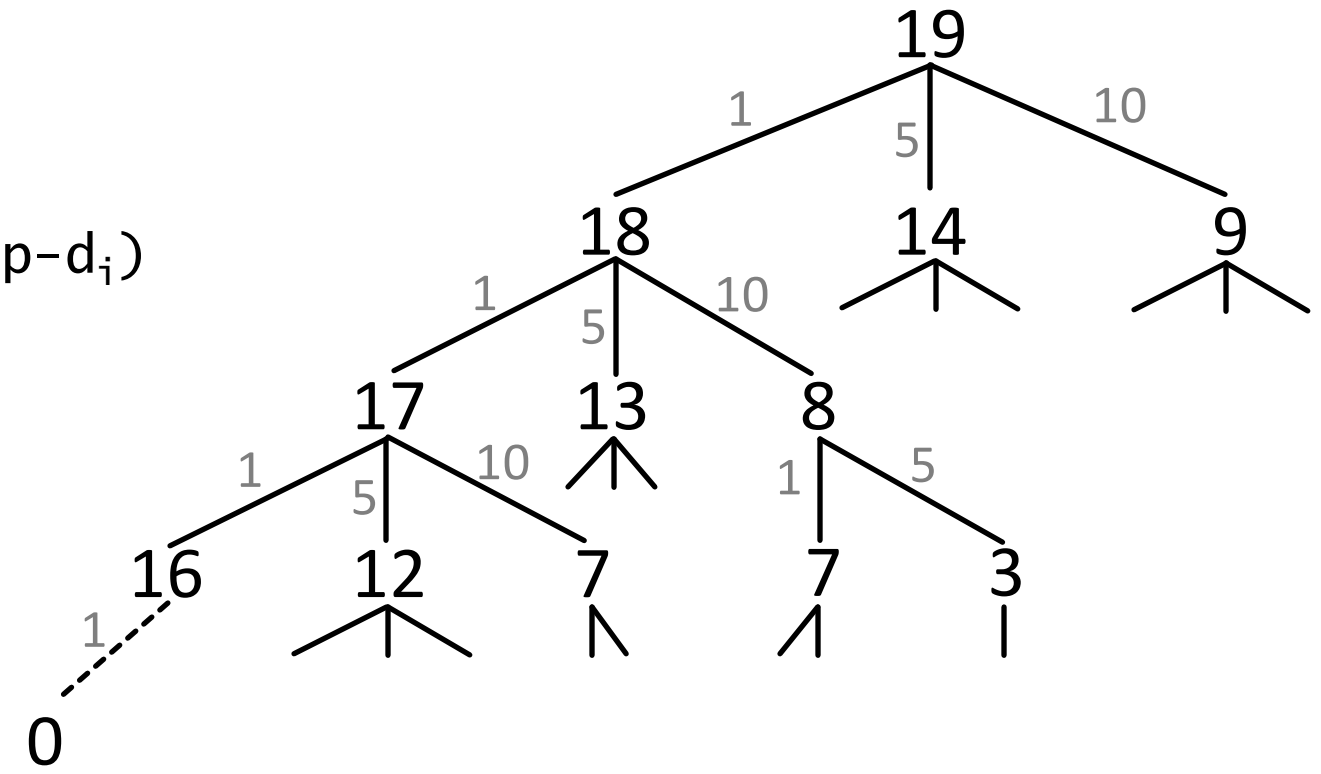
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

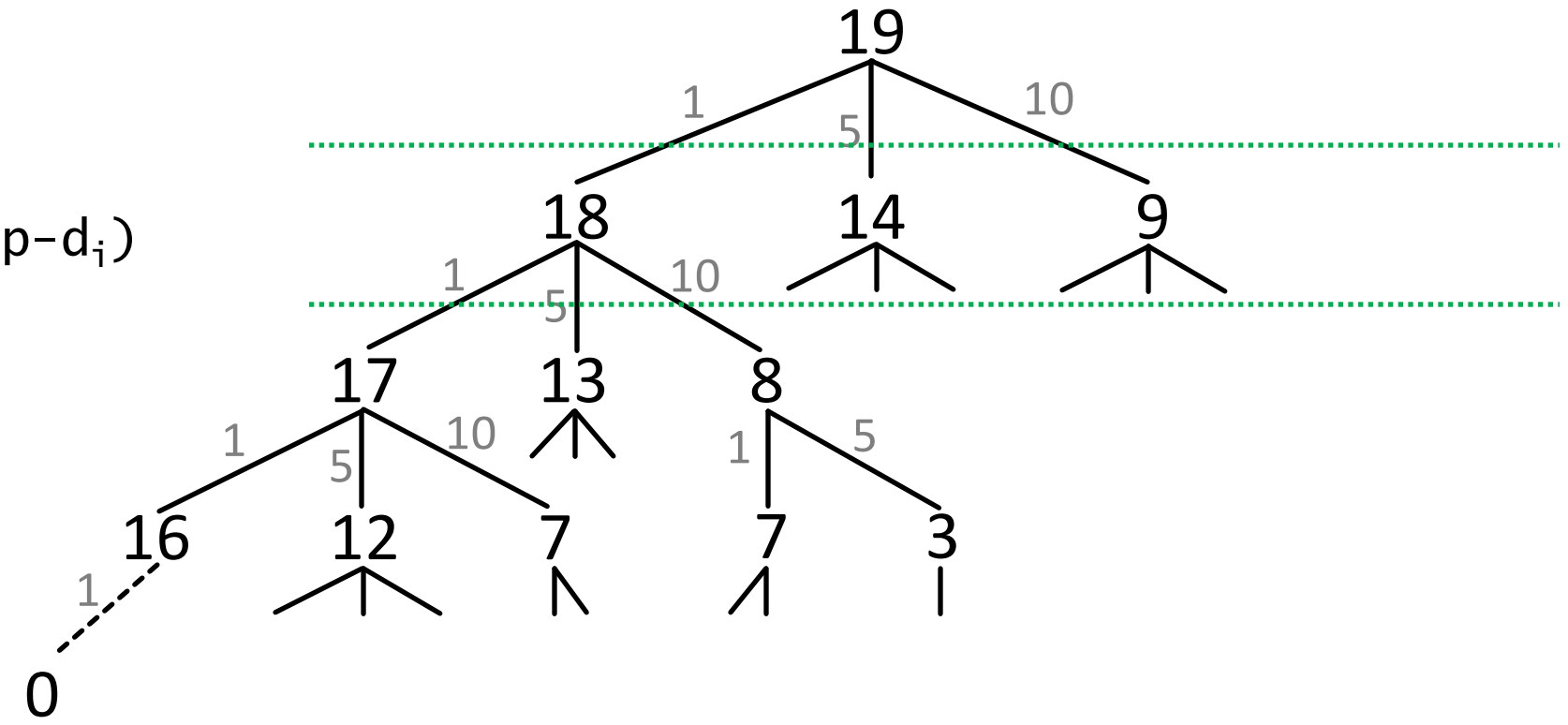
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

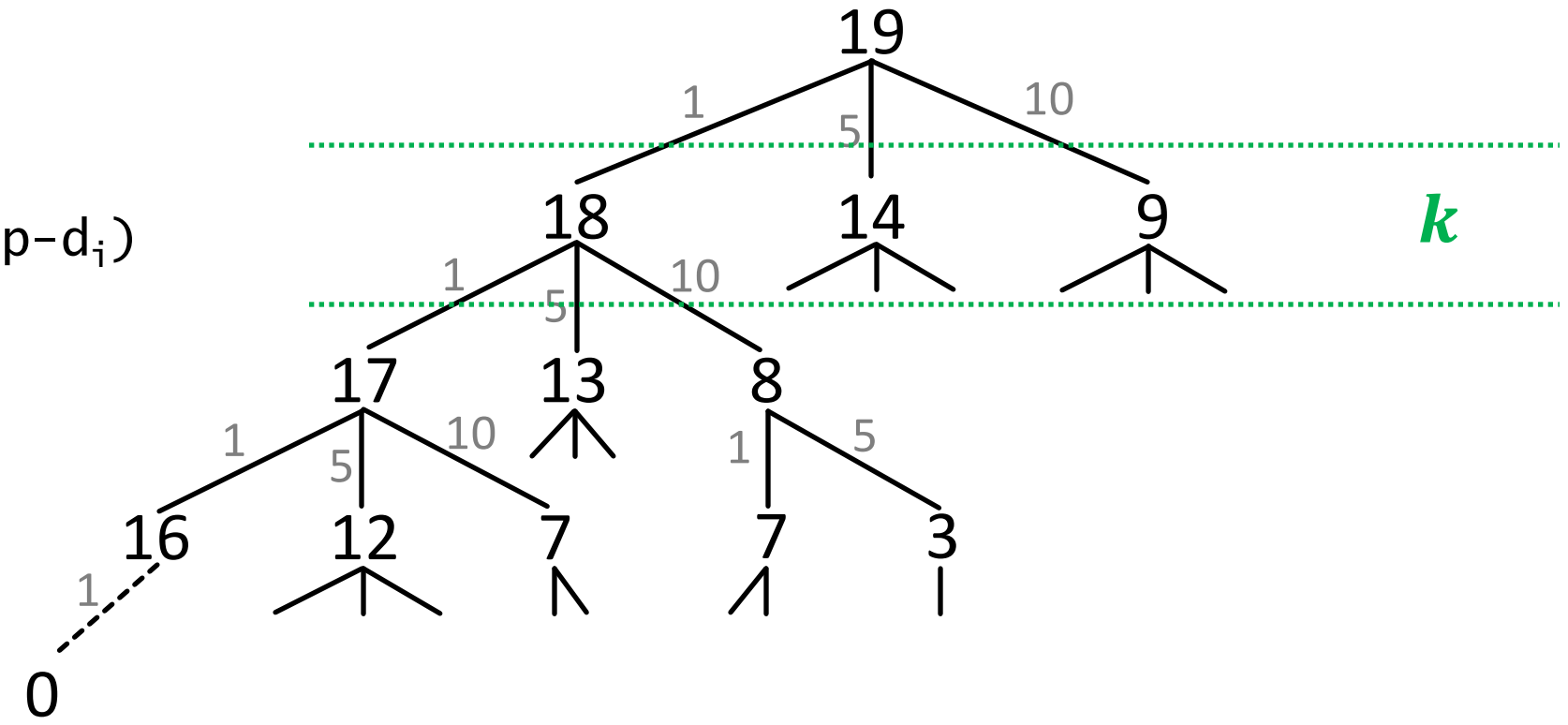
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

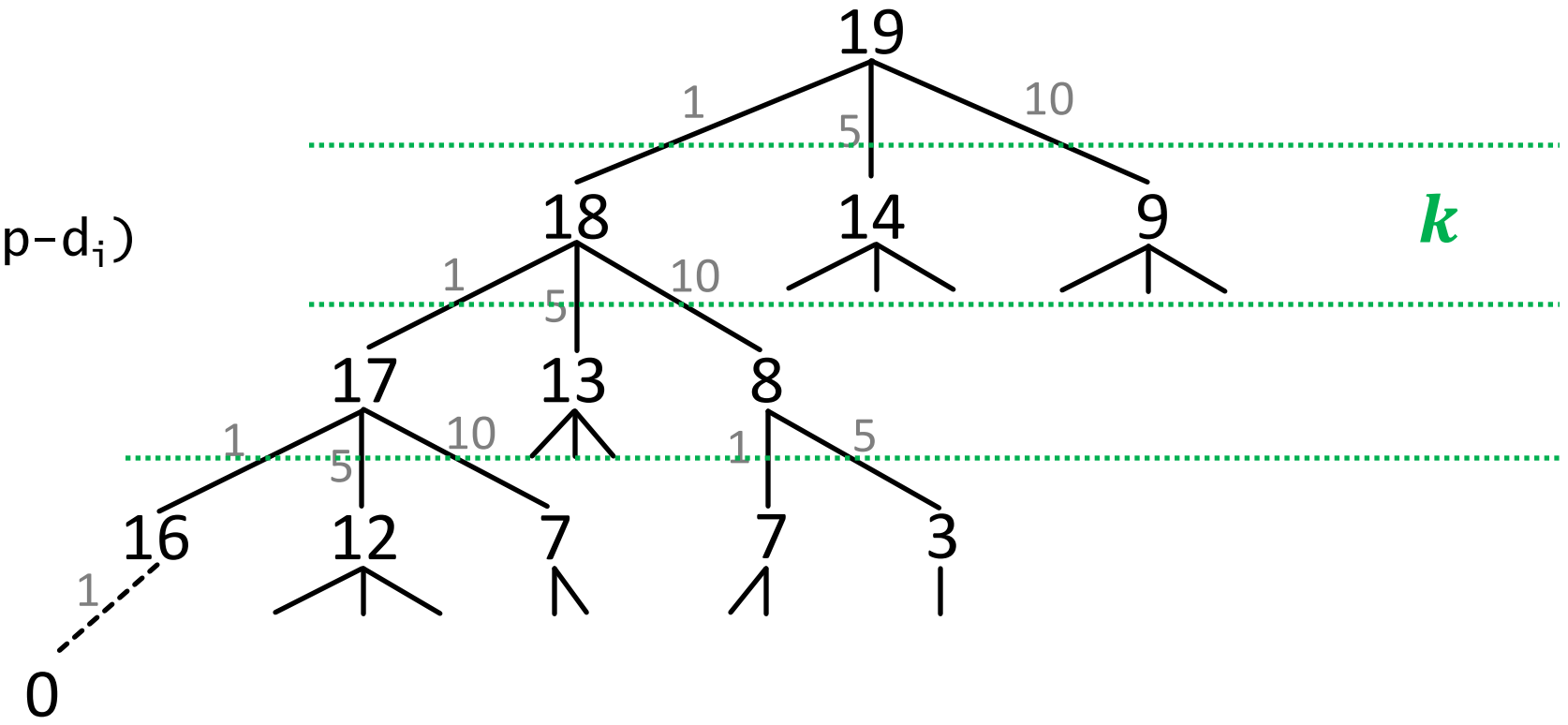
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

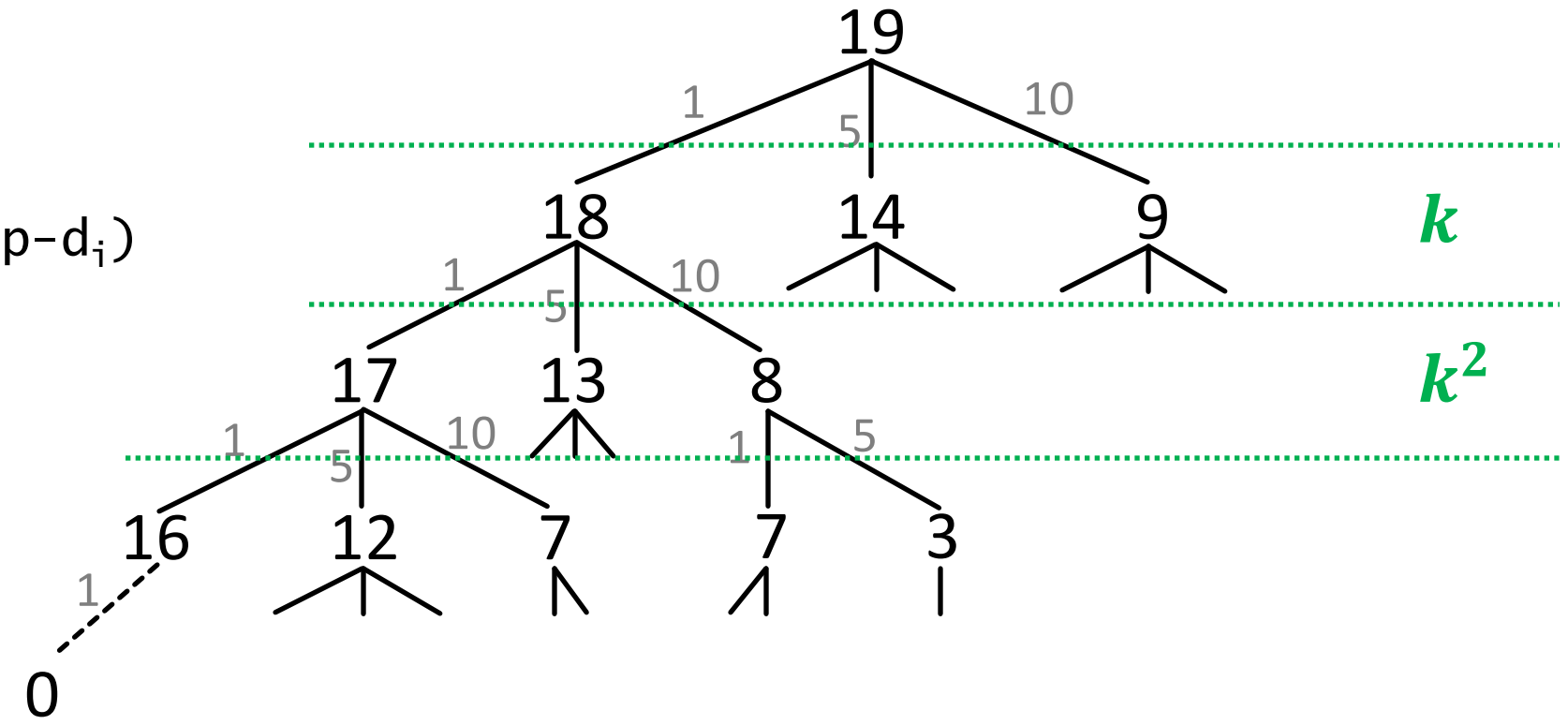
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

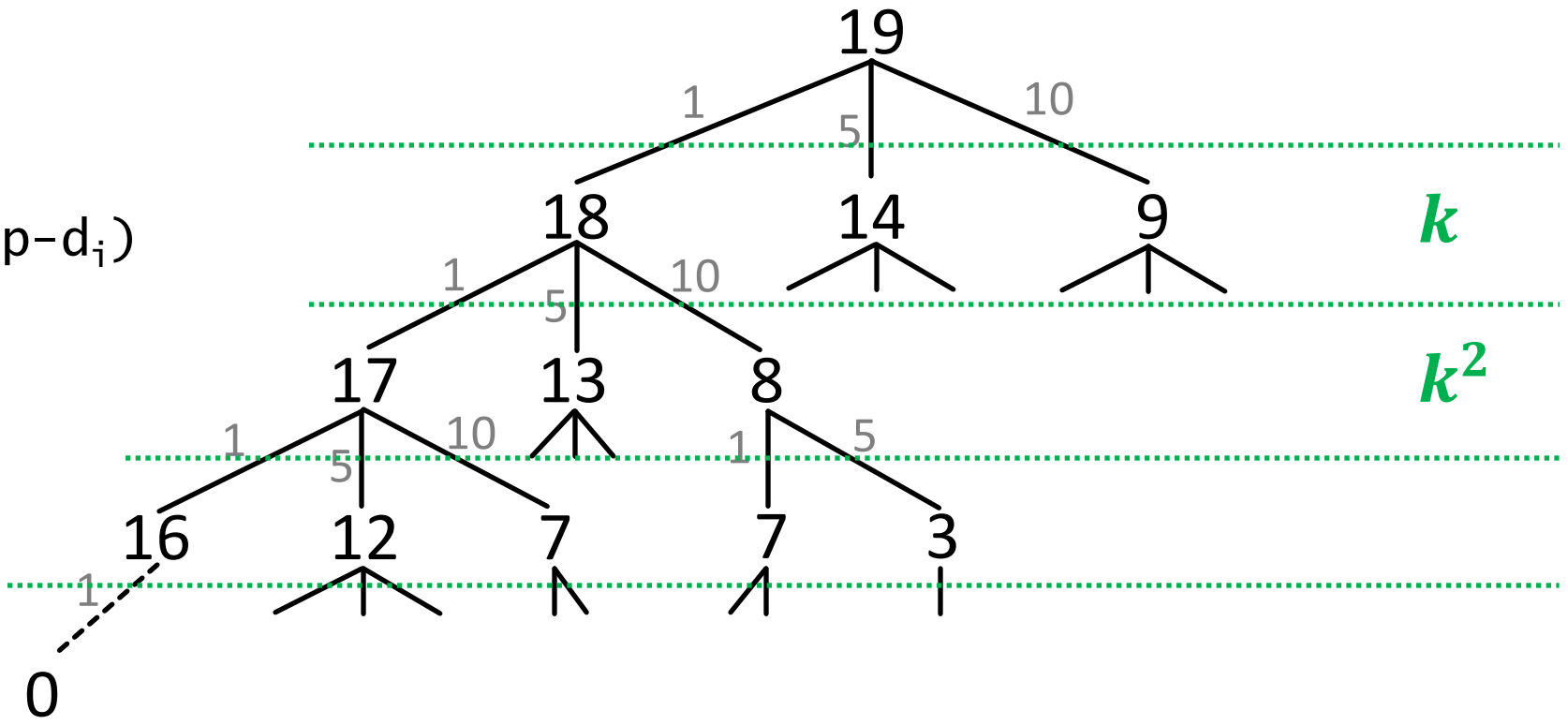
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

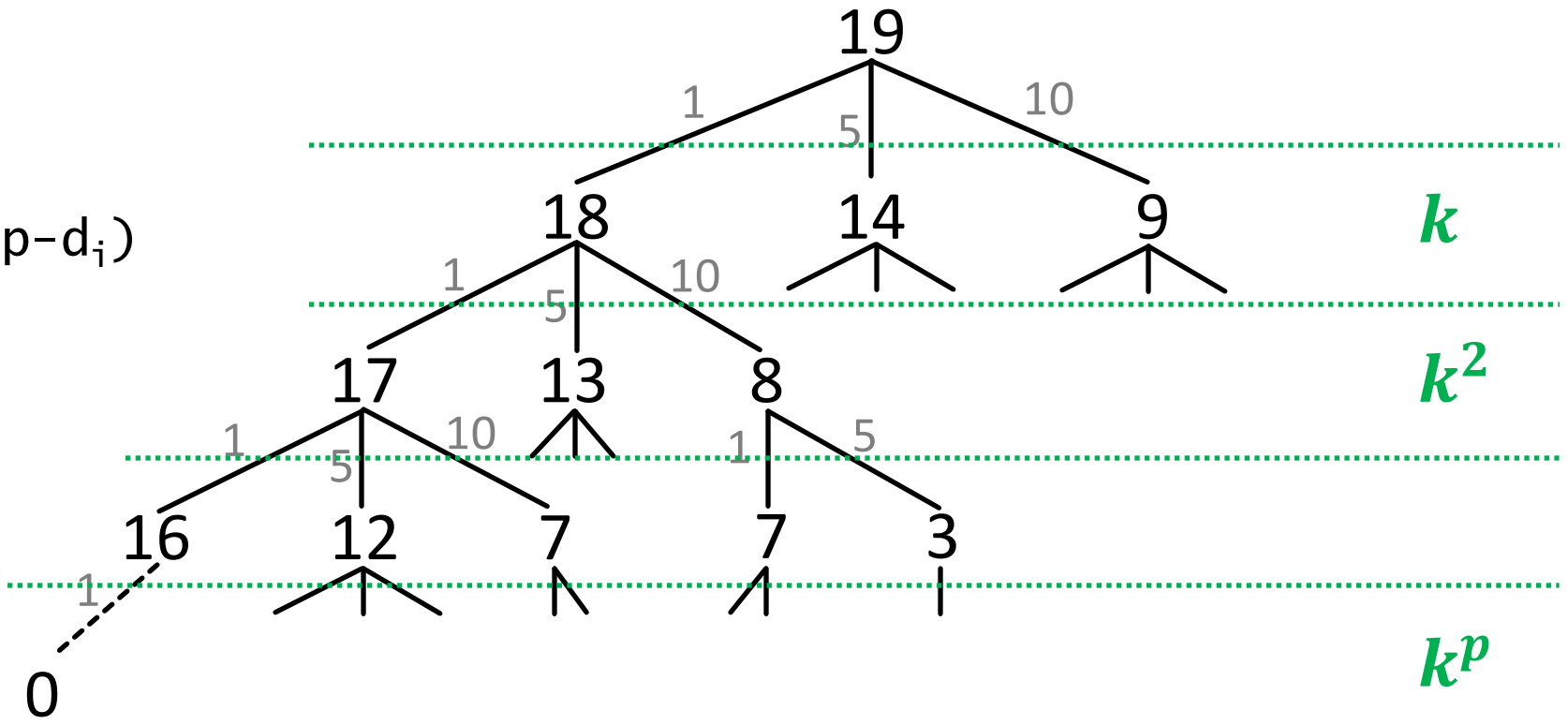
Running time?

$O(k^p)$

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

k = # denominations

p = value to make change for



Making Change – Recursive

```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

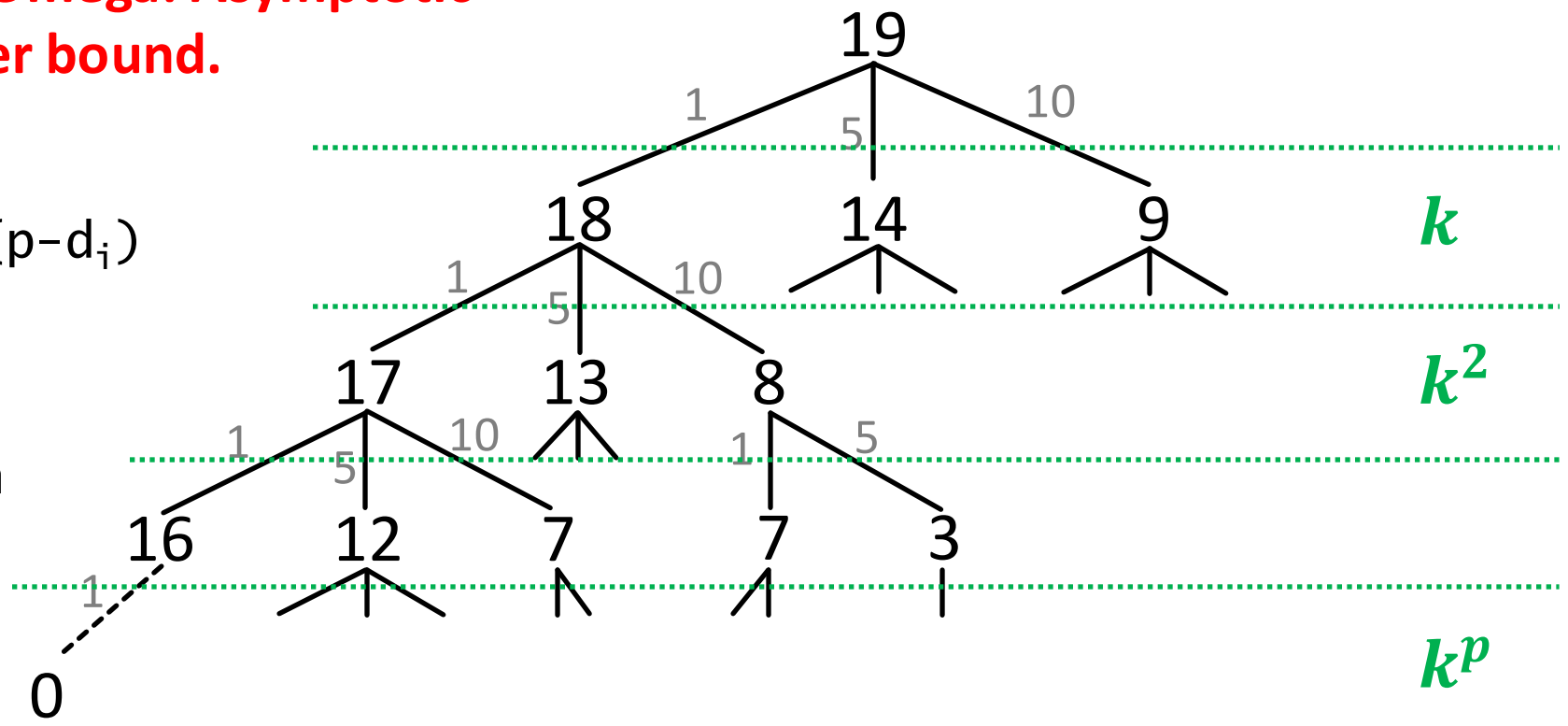
$O(k^p) \in \Omega(\text{Bad})$

Big-Omega: Asymptotic lower bound.

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

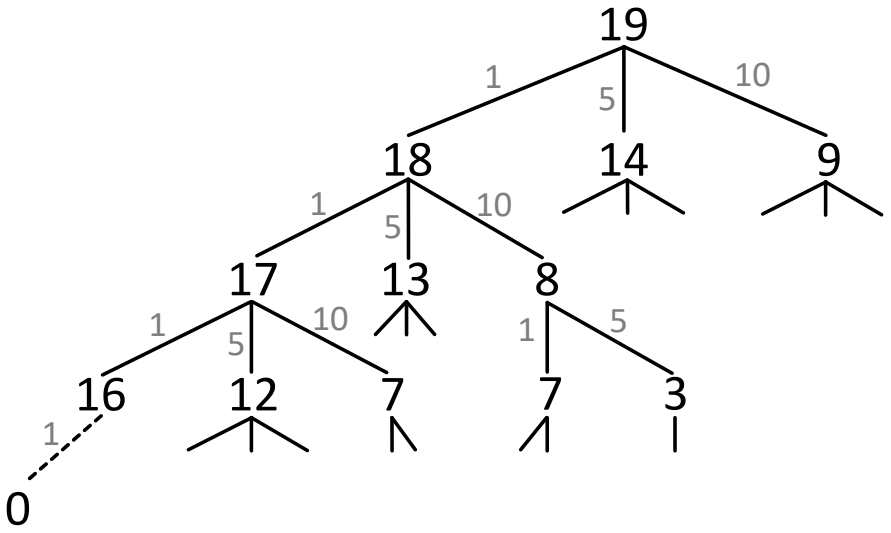
k = # denominations

p = value to make change for



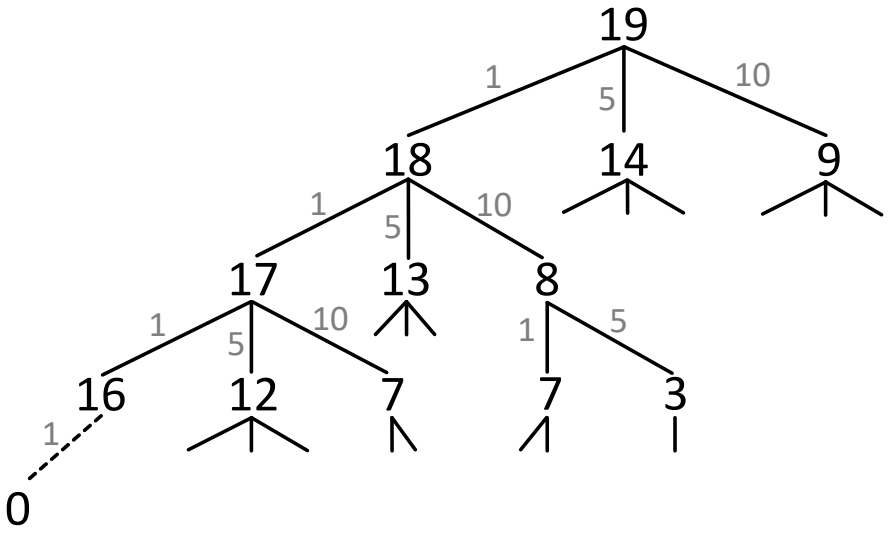
Making Change – Dynamic Programing

How do I make change for $0.33?



Making Change – Dynamic Programing

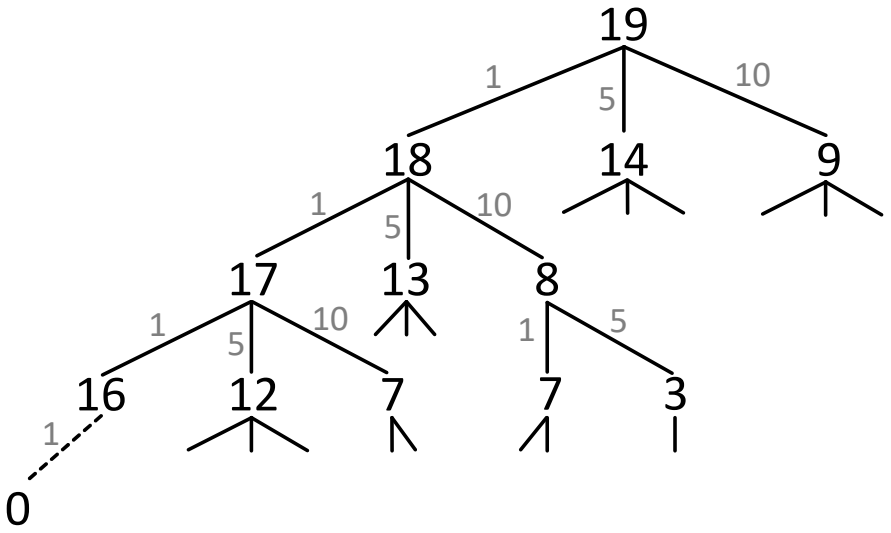
How do I make change for $0.33?



If I know the best way to make change for $0.01 - $0.32, how can I figure out the best way to make change for $0.33?

Making Change – Dynamic Programing

How do I make change for $0.33?

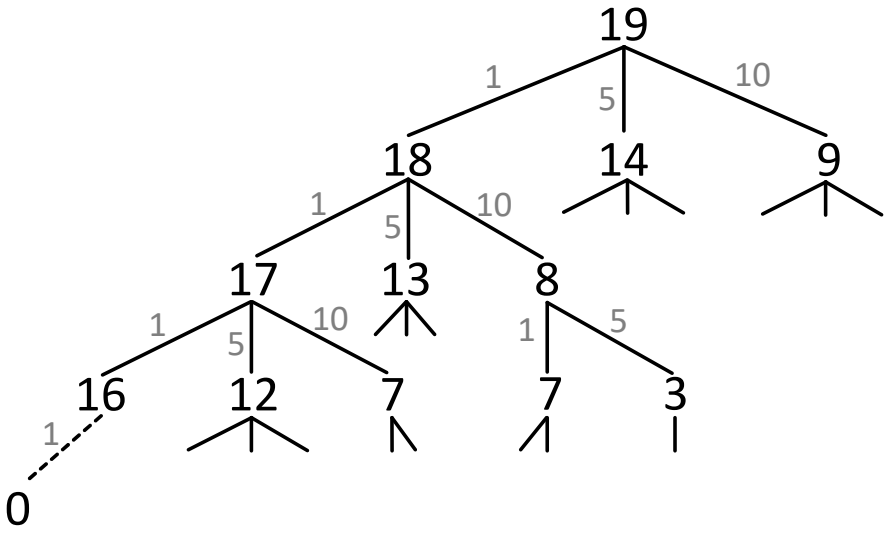


If I know the best way to make change for $0.01 - $0.32, how can I figure out the best way to make change for $0.33?

$$\text{change}(\$0.33) = \min \begin{cases} \text{change}(\$0.08) + 1 \\ \text{change}(\$0.23) + 1 \\ \text{change}(\$0.28) + 1 \\ \text{change}(\$0.32) + 1 \end{cases}$$

Making Change – Dynamic Programing

How do I make change for $0.33?



If I know the best way to make change for $0.01 - $0.32, how can I figure out the best way to make change for $0.33?

$$\text{change}(\$0.33) = \min \begin{cases} \text{change}(\$0.08) + 1 \\ \text{change}(\$0.23) + 1 \\ \text{change}(\$0.28) + 1 \\ \text{change}(\$0.32) + 1 \end{cases}$$

Ground up vs Top down

Sub problems vs Explicit recursion

Saving your work vs Doing the same things multiple times

Making Change – Dynamic Programing

Plan (for value p):

1. Calculate optimal change for each value less than p.
2. Save those values.
3. Apply formula to find optimal change for p.

If I know the best way to make change for \$0.01 - \$0.32, how can I figure out the best way to make change for \$0.33?

$$\text{change}(\$0.33) = \min \begin{cases} \text{change}(\$0.08) + 1 \\ \text{change}(\$0.23) + 1 \\ \text{change}(\$0.28) + 1 \\ \text{change}(\$0.32) + 1 \end{cases}$$

Making Change – Dynamic Programing

```
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]
```

Making Change – Dynamic Programing

```
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]
```

array holding optimal values for all values $\leq p$.

Making Change – Dynamic Programing

```
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]
```

array holding optimal values for all values $\leq p$.

find optimal change amounts starting with 1.

Making Change – Dynamic Programing

```
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]
```

array holding optimal values for all values $\leq p$.

find optimal change amounts starting with 1.

for each denomination value.

Making Change – Dynamic Programing

```
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]
```

array holding optimal values for all values $\leq p$.

find optimal change amounts starting with 1.

for each denomination value.

check optimal value of $m - d_i$

Making Change – Dynamic Programing

```
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]
```

Suppose we are filling this out for $m = 13$.

Chng[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	?	?	?	?	?	?	?	?

Making Change – Dynamic Programing

```
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]
```

Suppose we are filling this out for $m = 13$.

$d_i = 10$:

$d_i = 5$:

$d_i = 1$:

Chng[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	?	?	?	?	?	?	?	?

Making Change – Dynamic Programing

```
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]
```

Suppose we are filling this out for $m = 13$.

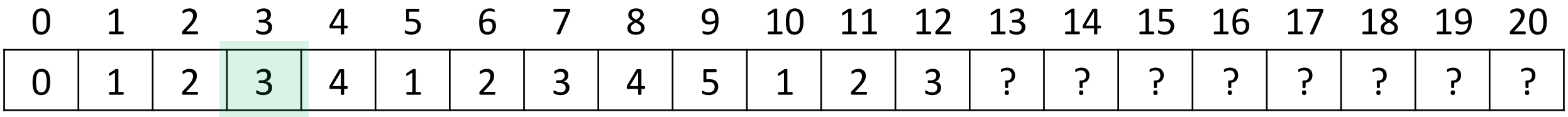
$d_i = 10$:

Chng[13−10]+1 = 4

$d_i = 5$:

$d_i = 1$:

Chng[]



Making Change – Dynamic Programing

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

Suppose we are filling this out for $m = 13$.

$d_i = 10$:
Chng[13−10]+1 = 4

$d_i = 5$:
Chng[13−5]+1 = 5

$d_i = 1$:

Chng[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	?	?	?	?	?	?	?	?

Making Change – Dynamic Programing

```
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]
```

Suppose we are filling this out for $m = 13$.

$d_i = 10$:
Chng[13−10]+1 = 4

$d_i = 5$:
Chng[13−5]+1 = 5

$d_i = 1$:
Chng[13−1]+1 = 4

Chng[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	?	?	?	?	?	?	?	?

Making Change – Dynamic Programing

```
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]
```

Suppose we are filling this out for $m = 13$.

$d_i = 10$:
Chng[13−10]+1 = 4

$d_i = 5$:
Chng[13−5]+1 = 5

$d_i = 1$:
Chng[13−1]+1 = 4

Chng[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	**4**	?	?	?	?	?	?	?

Making Change – Dynamic Programing

```
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]
```

Chng[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	4	5	2	3	4	5	6	2

Making Change – Dynamic Programing

```
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]
```

Running time?

k = # denominations

p = value to make change for

Making Change – Dynamic Programing

```
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]
```

Running time?

Outer for loop $\in O(p)$

Inner for loop $\in O(k)$

Total $\in O(pk)$

k = # denominations

p = value to make change for

Making Change – Dynamic Programing

```
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]
```

Running time?

- Size of array ∈ $O(p)$
- Checks per spot ∈ $O(k)$
- Time per check ∈ $O(1)$
- Total ∈ $O(pk)$

k = # denominations

p = value to make change for

Chng[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	2	3	4	1	2	3	4	5	1	2	3	4	5	2	3	4	5	6	2

Making Change – Dynamic Programing

```
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]
```

What if we want the actual coin values for the optimal solution?

Making Change – Dynamic Programing

```
changeDP(p)
   Chng[0,...,p] = [0,...,0]
   lastCoin[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]
```

What if we want the actual coin values for the optimal solution?

Making Change – Dynamic Programing

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

What if we want the actual coin values for the optimal solution?

Making Change – Dynamic Programing

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

What if we want the actual coin values for the optimal solution?

Making Change – Dynamic Programing

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

What if we want the actual coin values for the optimal solution?

Making Change – Dynamic Programing

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

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

Making Change – Dynamic Programing

```
remain = 0
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

remain = 13

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 =

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

remain = 13

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

remain = 3

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

remain = 3

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10 + 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

remain = 2

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10 + 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

remain = 2

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10 + 1 + 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

remain = 1

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10 + 1 + 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

remain = 1

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10 + 1 + 1 + 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10

Making Change – Dynamic Programing

remain = 0

```
remain = p
Set<Integer> coins
while remain > 0
    coin = lastCoin[remain]
    coins.add(coin)
    remain = remain - coin
```

lastCoin is an array of the final coins added to the optimal solutions.

How do we get all of the coins in the solution?

13 = 10 + 1 + 1 + 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Chng[]:	0	1	2	3	4	1	2	3	4	5	1	2	3	4
lastCoin[]:	0	1	1	1	1	5	1	5	5	1	10	10	1	10