Dynamic Programming
CSCI 432

Strassen's Algorithm

$$log_b a = log_2 7 \cong 2.81 > 2 = d$$
$$\Rightarrow T(n) = O(n^{2.81})$$

$$P_6 \qquad P_1 + P_2$$
$$P_1 + P_5 - P_3 - P_7 \Big]$$

$$(A + D)(E + H)$$
$$(B - D)(G + H)$$
$$(A - C)(E + F)$$

th

=

=

History of Matrix Multiplication Algorithms

- $O(n^3)$ – 1812 (Matrix multiplication defined)
- $O(n^{2.81})$ – 1969 ← **Strassen's Algorithm**
- $O(n^{2.3755})$ – 1990
- $O(n^{2.3737})$ – 2010
- $O(n^{2.3729})$ – 2013
- $O(n^{2.3728639})$ – 2014
- $O(n^{2.3728596})$ – 2020
- $O(n^{2.371866})$ – 2022
- $O(n^{2.371339})$ – 2024
- $O(n^{2.371177})$ – 2026

Lower bound: $O(n^2)$. Need to look at all elements of both matrices.

$O(n^{2.81})$ vs $O(n^3)$?

- **1 second vs 2.6 seconds**
- **1 hour vs 4.5 hours**
- **1 day vs 5.5 days**
- **3 days vs 18 days**

Strassen's Algorithm

$$log_b a = log_2 7 \cong 2.81 > 2 = d$$
$$\Rightarrow T(n) = O(n^{2.81})$$

$$P_6 \qquad P_1 + P_2$$
$$P_1 + P_5 - P_3 - P_7 \Big]$$

$$(A + D)(E + H)$$
$$(B - D)(G + H)$$
$$(A - C)(E + F)$$

th

=

=

History of Matrix Multiplication Algorithms

- $O(n^3)$ – 1812 (Matrix multiplication defined)
- $O(n^{2.81})$ – 1969 ← **Strassen's Algorithm**
- $O(n^{2.3755})$ – 1990
- $O(n^{2.3737})$ – 2010
- $O(n^{2.3729})$ – 2013
- $O(n^{2.3728639})$ – 2014
- $O(n^{2.3728596})$ – 2020
- $O(n^{2.371866})$ – 2022
- $O(n^{2.371339})$ – 2024
- $O(n^{2.371177})$ – 2026
- $O(n^{2.25})$ – 2026

Lower bound: $O(n^2)$. Need to look at all elements of both matrices.

$O(n^{2.81})$ vs $O(n^3)$?

- **1 second vs 2.6 seconds**
- **1 hour vs 4.5 hours**
- **1 day vs 5.5 days**
- **3 days vs 18 days**

Dynamic Programing

Trick-or-treat optimization.



Dynamic Programing

Trick-or-treating at the red house, blue house, and green house are the fewest stops you need to fill your 25-pound capacity sack.



Dynamic Programing

Trick-or-treating at the red house, blue house, and green house are the fewest stops you need to fill your 25-pound capacity sack. The blue house gives you 5 pounds of candy. What can you conclude?



Dynamic Programing

Trick-or-treating at the red house, blue house, and green house are the fewest stops you need to fill your 25-pound capacity sack. The blue house gives you 5 pounds of candy. What can you conclude?

The red house and the green house are the fewest stops you need to get 20+ pounds of candy.



Dynamic Programing

Trick-or-treating at the red house, blue house, and green house are the fewest stops you need to fill your 25-pound capacity sack. The blue house gives you 5 pounds of candy. What can you conclude?

The red house and the green house are the fewest stops you need to get 20+ pounds of candy.

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

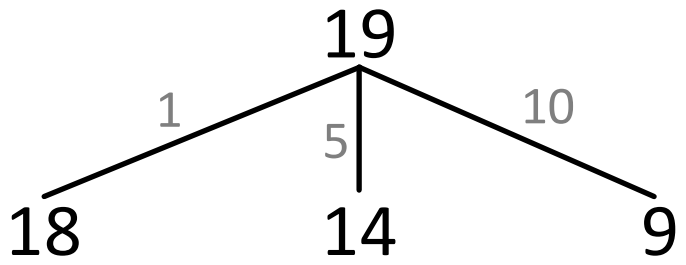
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

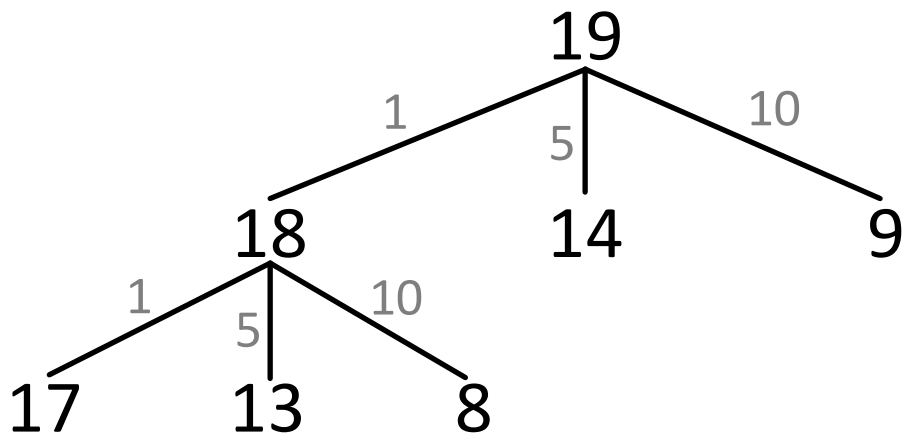
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

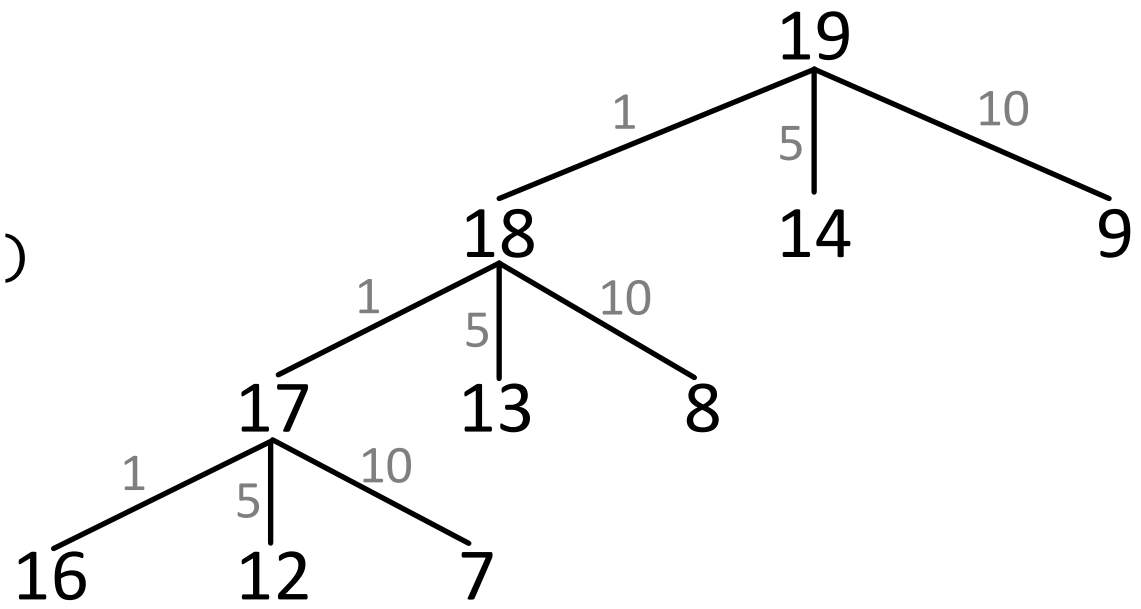
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

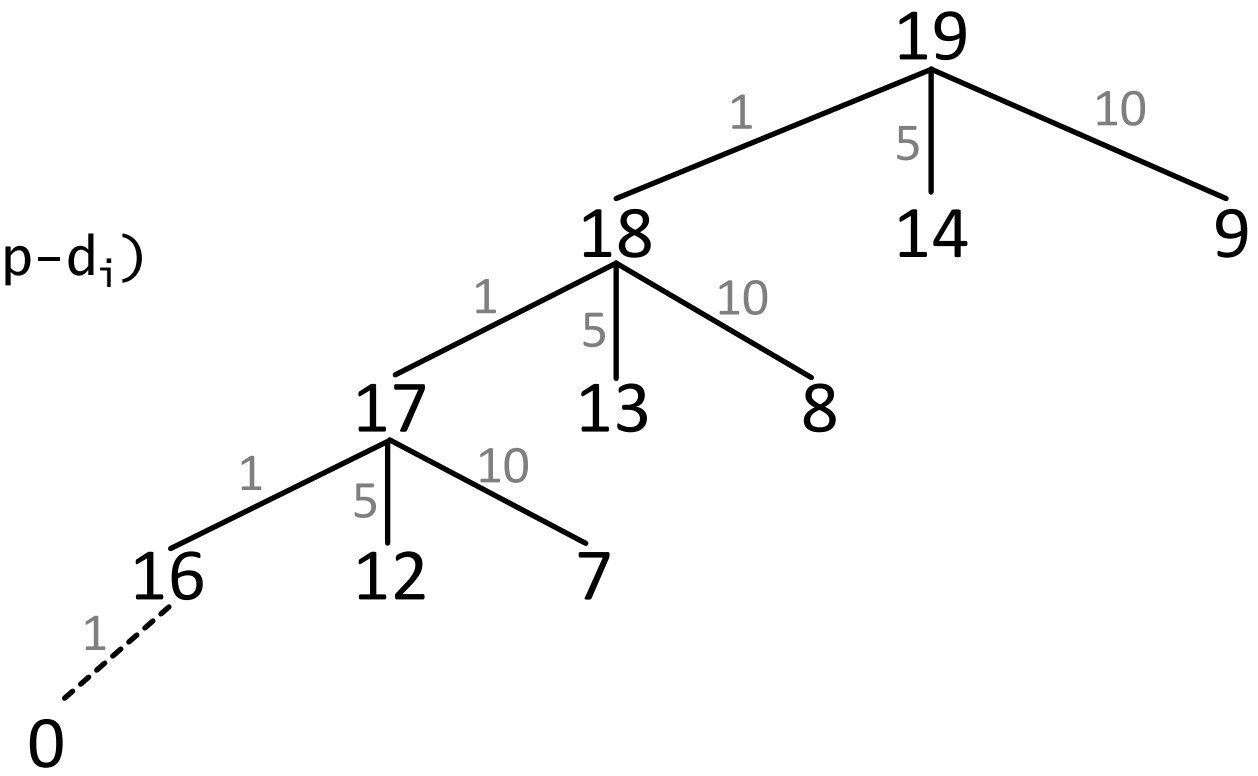
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

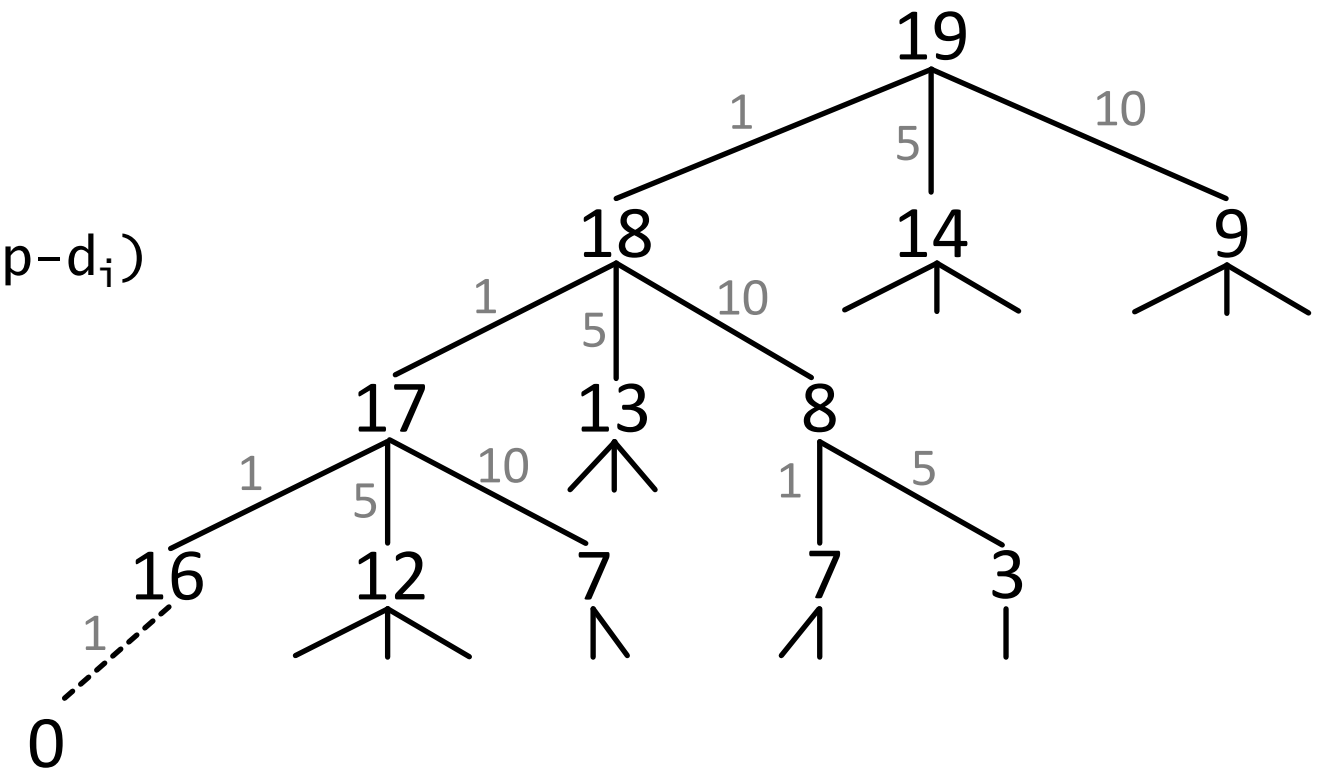
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

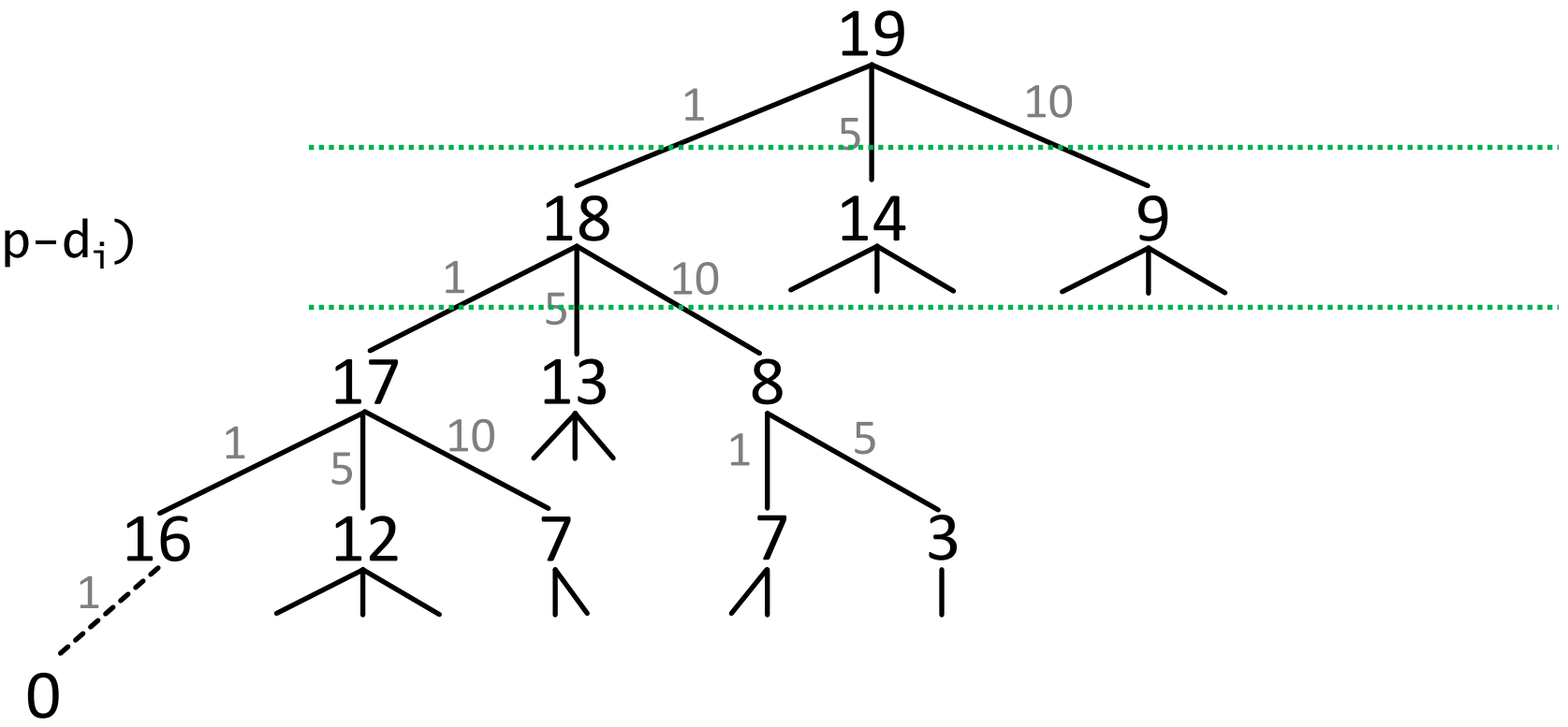
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

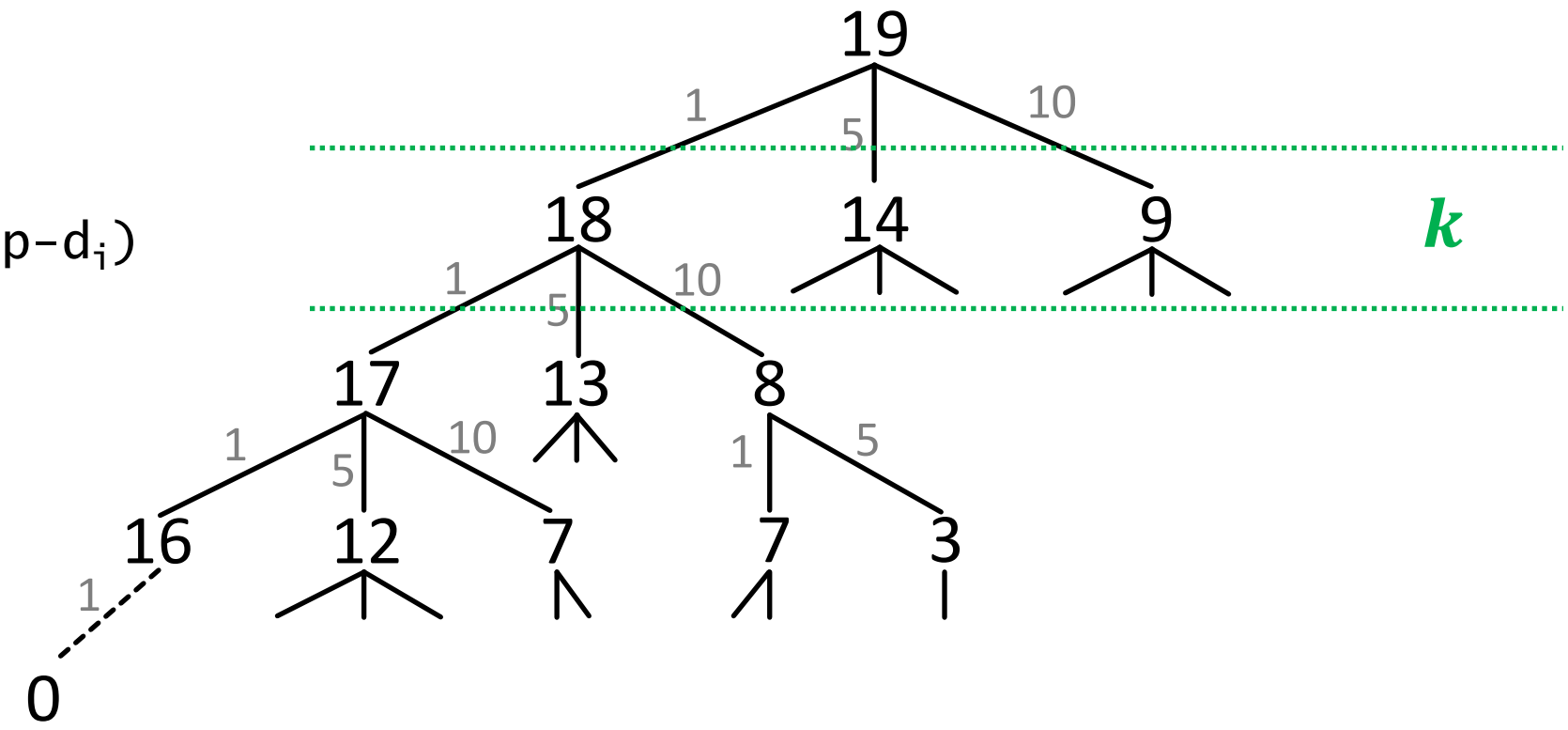
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

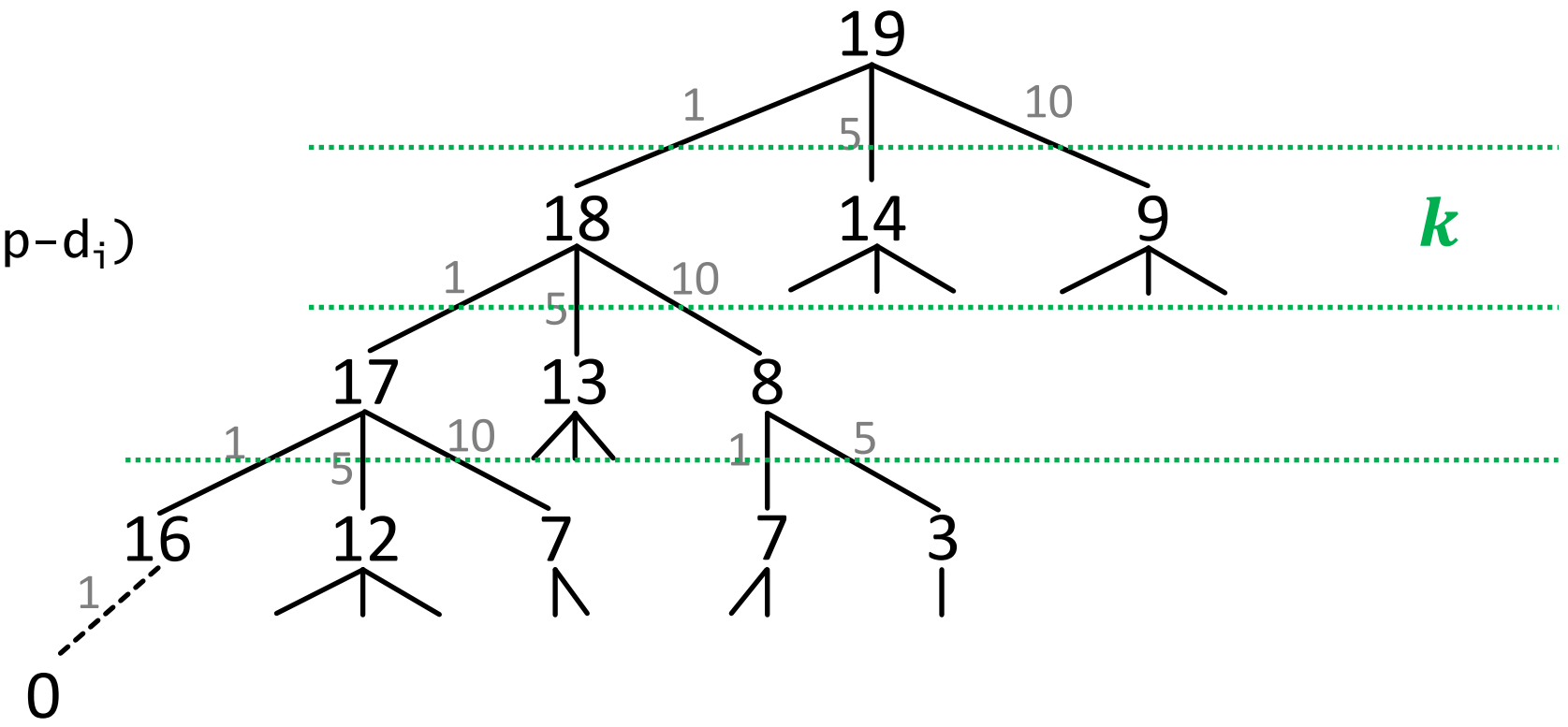
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

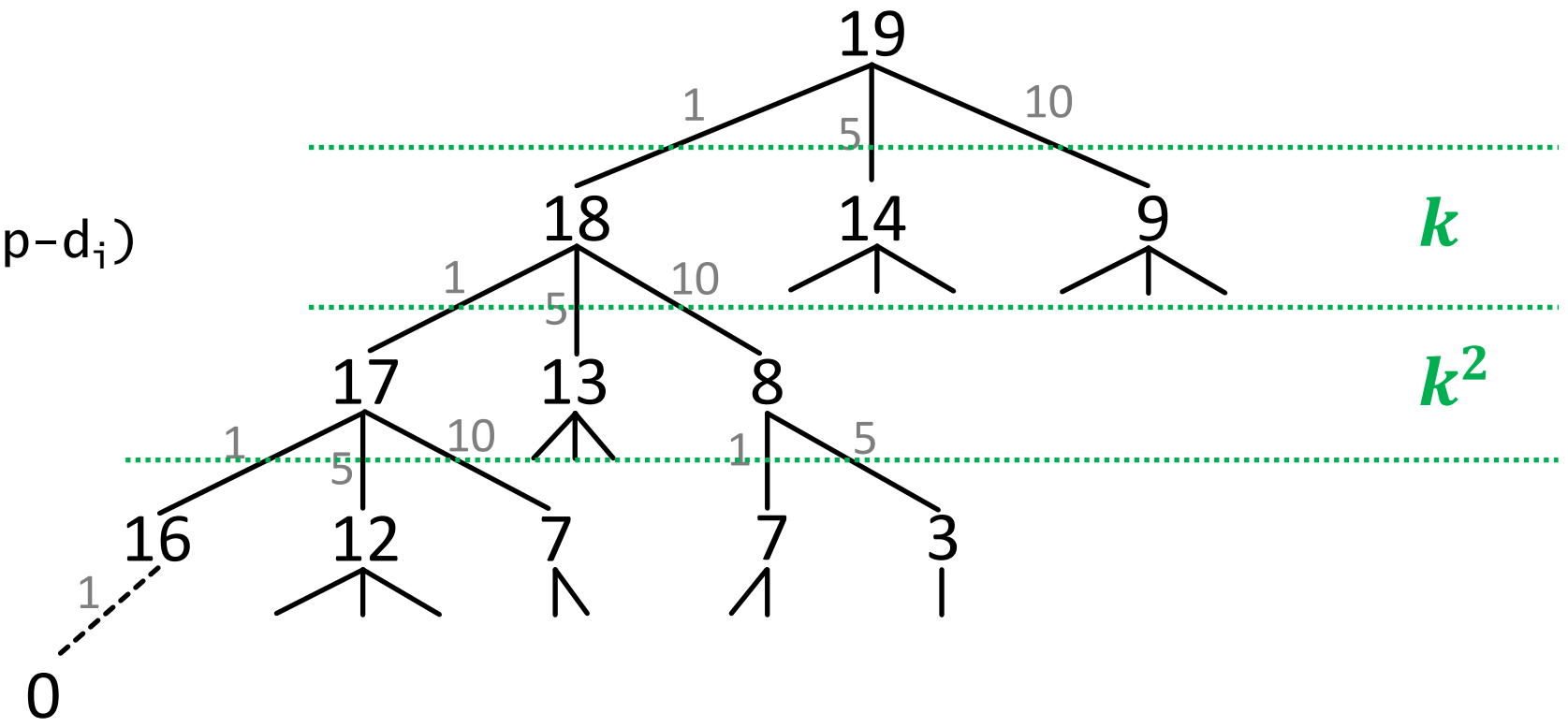
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

Running time?

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

k = # denominations

p = value to make change for



Making Change – Recursive

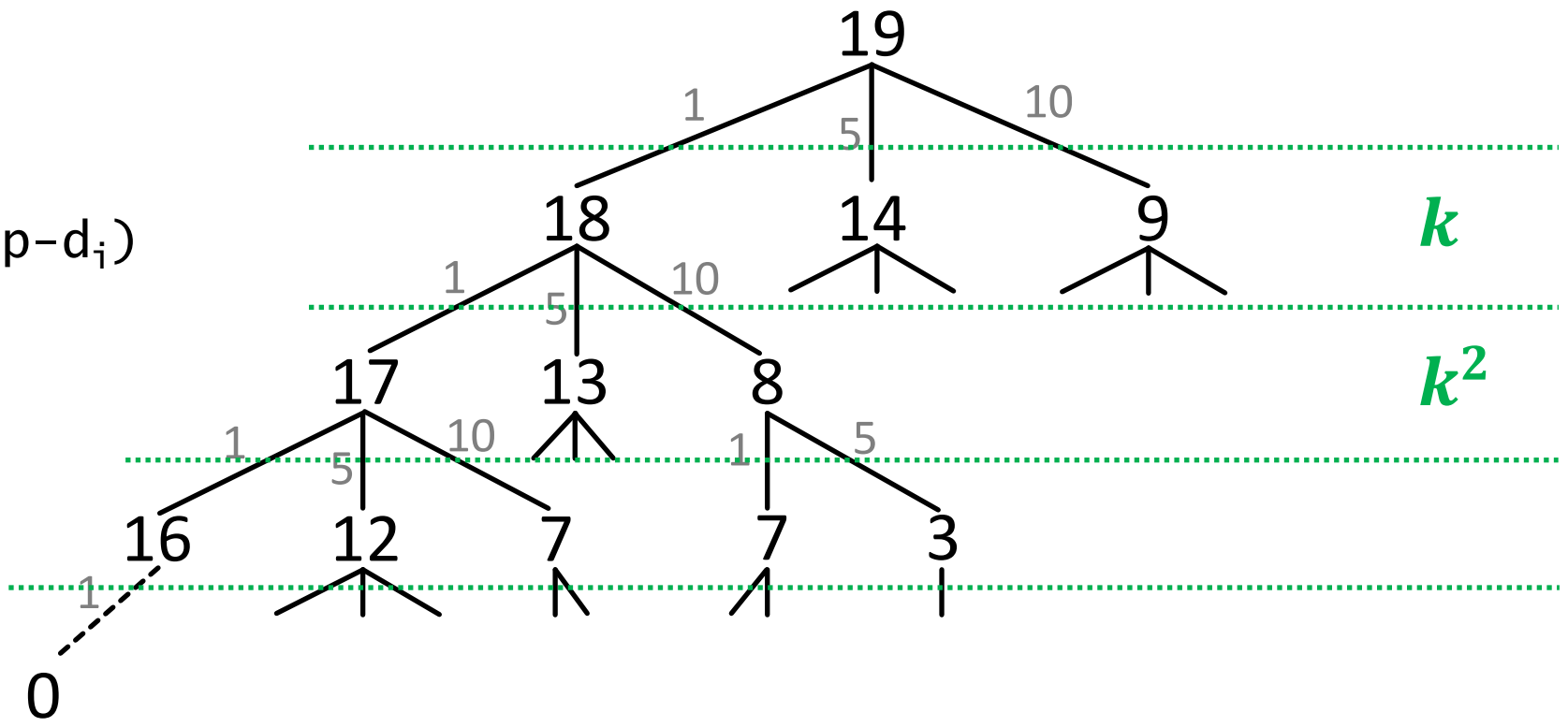
```
change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p-d_i)
            if a < min
                min = a
        return 1 + min
```

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
```

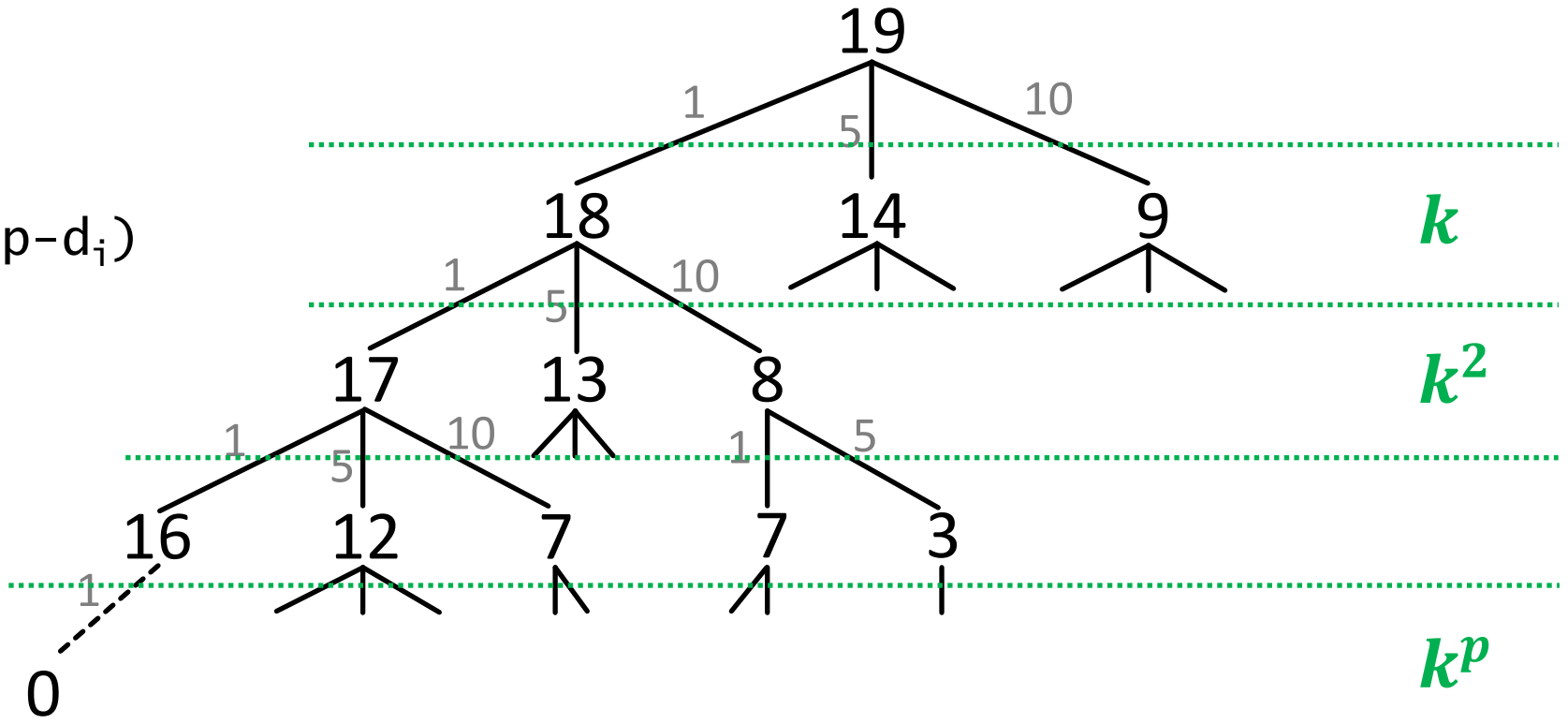
Running time?

$O(k^p)$

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?

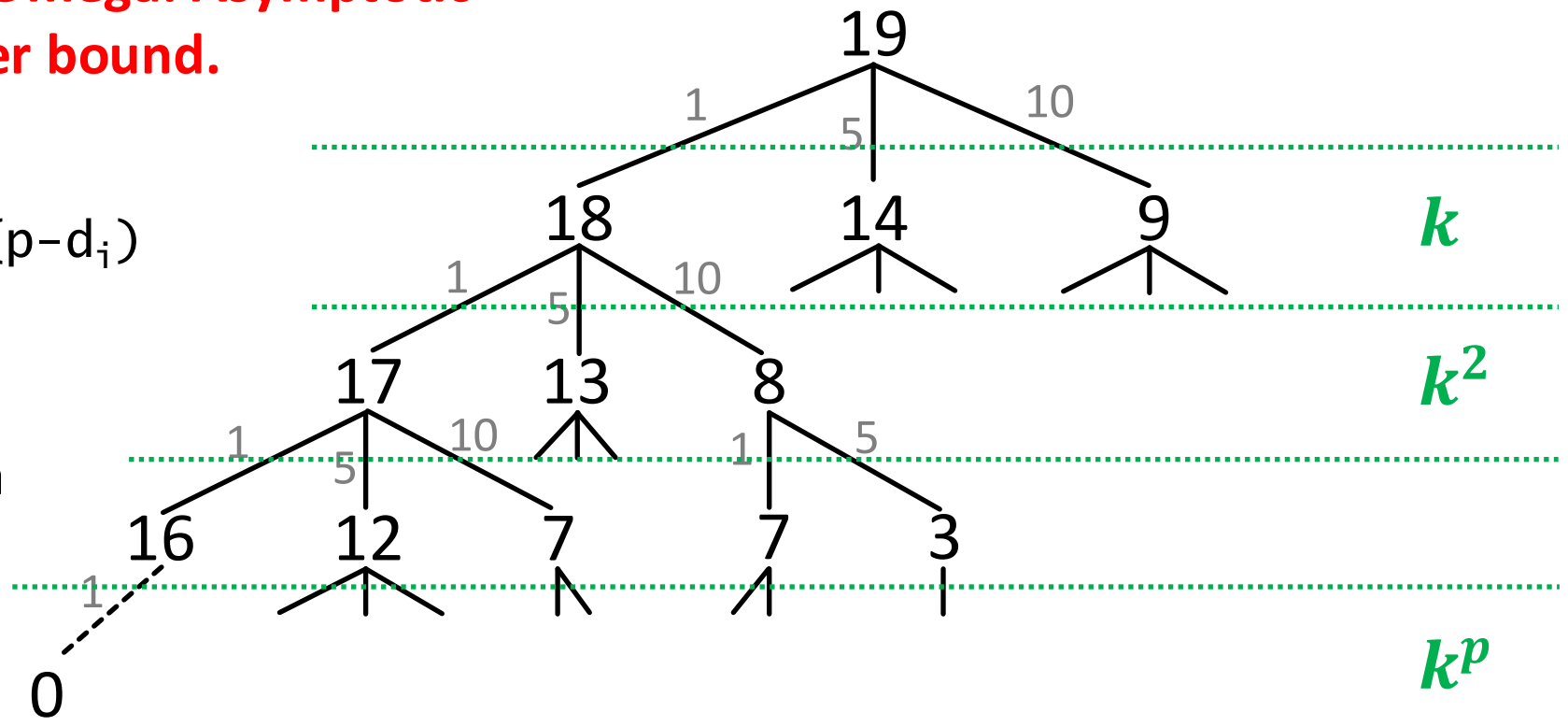
$O(k^p) \in \Omega(\text{Bad})$

Big-Omega: Asymptotic lower bound.

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

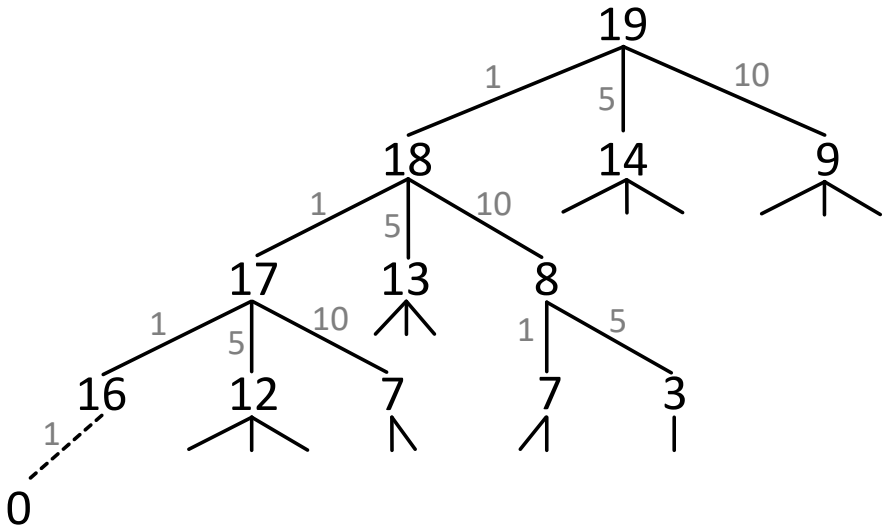
k = # denominations

p = value to make change for



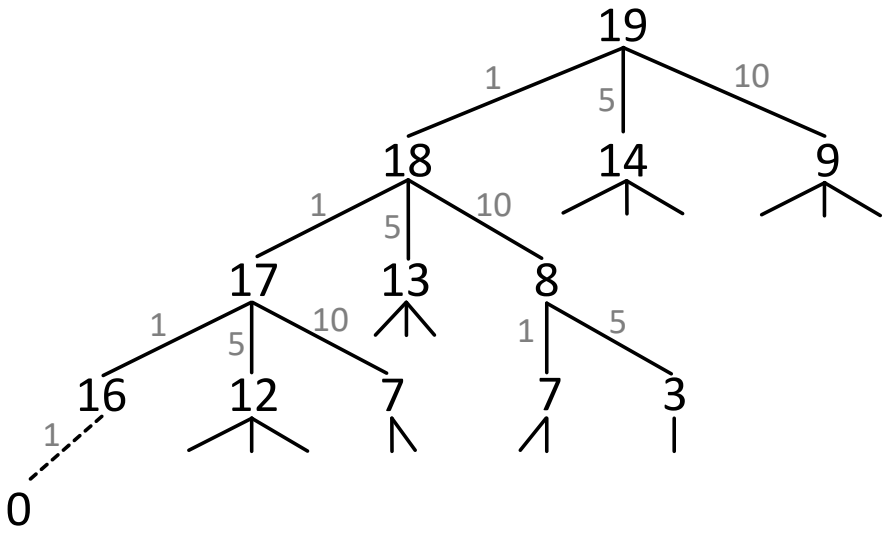
Making Change – Dynamic Programing

How do I make change for $0.33?



Making Change – Dynamic Programing

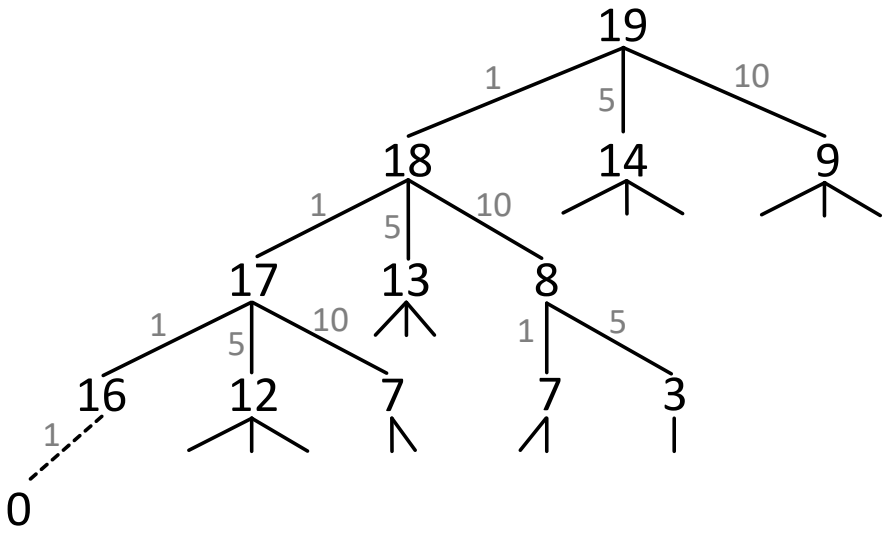
How do I make change for $0.33?



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?

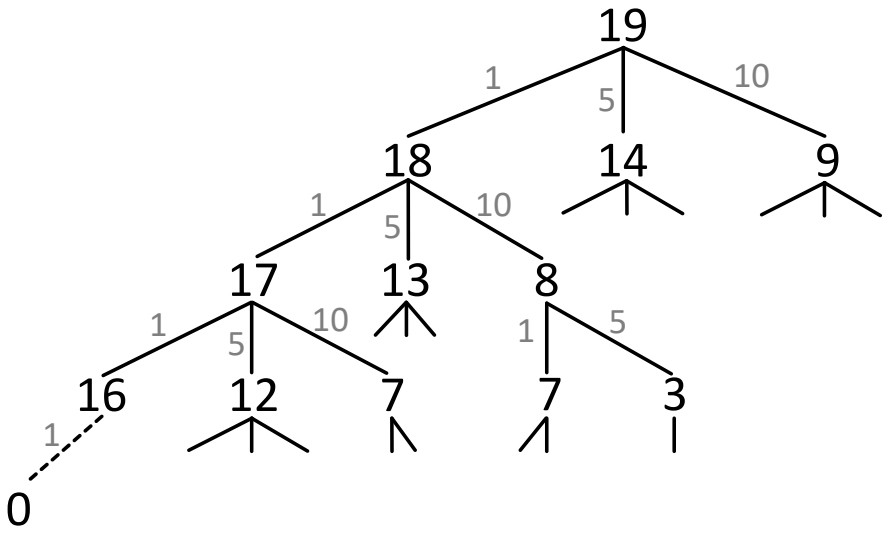


If I know the best way to make change for $0.01 - $0.32, how can I figure out the best way to make change for $0.33?

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

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?

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