Dynamic Programming
CSCI 532
Making Change

\( C(p) \) – minimum number of coins to make \( p \) cents.
\( x \) – value (e.g. $0.25) of a coin used in the optimal solution.

\[
C(p) = 1 + C(p - x).
\]

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

Least change for 20 cents = 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 - 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

```python
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 = \infty
        for d_i \leq 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

\[
\text{change}(p) \\
\text{if } p == 0 \\
\quad \text{return } 0 \\
\text{else} \\
\quad \text{min} = \infty \\
\quad \text{for } d_i \leq p \\
\quad \quad a = \text{change}(p-d_i) \\
\quad \quad \text{if } a < \text{min} \\
\quad \quad \quad \text{min} = a \\
\quad \text{return } 1 + \text{min}
\]

Make $0.19$ with $0.01$, $0.05$, $0.10$
\[k = \# \text{ denominations}\]
\[p = \text{value to make change for}\]

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

Make $0.19$ with $0.01, 0.05, 0.10$
Making Change - Recursive

\[
\text{change}(p) = \begin{cases} 
0 & \text{if } p = 0 \\
\min & \text{else} \\
\infty & \text{for } d_i \leq p \\
a = \text{change}(p - d_i) & \text{if } a < \min \\
\min = a & \text{return } 1 + \min \\
\end{cases}
\]

Running time?

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

\[
k = \# \text{ denominations} \\
p = \text{value to make change for}
\]
Making Change - Recursive

change(p)
    if p == 0
        return 0
    else
        min = ∞
        for dᵢ ≤ p
            a = change(p-dᵢ)
            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

```python
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 $0.01$, $0.05$, $0.10$

$k = \#\text{ denominations}$

$p = \text{value to make change for}$

$\text{Change: } 0.01, 0.05, 0.10$
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

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

$k = \#$ denominations
$p = \text{value to make change for}$
Making Change - Recursive

```python
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
```

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

\[ k = \text{# denominations} \]
\[ p = \text{value to make change for} \]
Making Change - Recursive

\[
\text{change}(p) =
\begin{cases} 
0 & \text{if } p = 0 \\
\min & \text{else}
\end{cases}
\]

\[
\min = \infty
\]

\[
\text{for } d_i \leq p
\]

\[
a = \text{change}(p-d_i)
\]

\[
\text{if } a < \min \\
\min = a
\]

\[
\text{return } 1 + \min
\]

Running time?

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

\[
k = \# \text{denominations}
\]

\[
p = \text{value to make change for}
\]

\[
\begin{array}{c}
k = \#	ext{denominations} \\
p = \text{value to make change for}
\end{array}
\]
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

Make $0.19 with $0.01, $0.05, $0.10
k = # denominations
p = value to make change for
Making Change - Recursive

\[
\text{change}(p) = \left\{ \begin{array}{ll}
0 & \text{if } p = 0 \\
& \text{return } 0 \\
\infty & \text{else} \\
& \text{min } = \infty \\
& \text{for each } d_i \leq p \\
& \quad a = \text{change}(p - d_i) \\
& \quad \text{if } a < \text{min} \\
& \quad \quad \text{min } = a \\
& \text{return } 1 + \text{min}
\end{array} \right.
\]

Running time?

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

\[
k = \# \text{ denominations} \\
p = \text{value to make change for}
\]
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

\(k^p\)
Making Change – Dynamic Programming

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 Programming

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?

$$\text{least}(\$0.33) = \min \left\{ \begin{array}{l}
\text{least}(\$0.08) + 1 \\
\text{least}(\$0.23) + 1 \\
\text{least}(\$0.28) + 1 \\
\text{least}(\$0.32) + 1 \\
\end{array} \right. $$
Making Change – Dynamic Programming

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?

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

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

What’s the difference?
Ground up vs top down. Sub problems vs explicit recursion.

\[
C(p) – \text{minimum number of coins to make } p \text{ cents.}
\]

\[
x – \text{value (e.g. $0.25) of a coin used in the optimal solution.}
\]
Making Change – Dynamic Programming

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?

\[ \text{best}($0.33) = \min \left\{ \begin{array}{l} \text{best}($0.08) + 1 \\ \text{best}($0.23) + 1 \\ \text{best}($0.28) + 1 \\ \text{best}($0.32) + 1 \end{array} \right\} \]

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 $p$. 
Making Change – Dynamic Programming

changeDP(p)

Chng[0,...,p] = [0,...,0]

for m = 1 to p

min = \(\infty\)

for \(d_i \leq m\)

if \(\text{Chng}[m - d_i] + 1 < \text{min}\)

min = \(\text{Chng}[m - d_i] + 1\)

Chng[m] = min

return Chng[p]

array holding optimal values for all values \(\leq p\).
Making Change – Dynamic Programming

\[
\text{changeDP}(p) \\
\text{Chng}[0, \ldots, p] = [0, \ldots, 0] \\
\text{for } m = 1 \text{ to } p \\
\quad \text{min} = \infty \\
\quad \text{for } d_i \leq m \\
\quad \quad \text{if } \text{Chng}[m - d_i] + 1 < \text{min} \\
\quad \quad \quad \text{min} = \text{Chng}[m - d_i] + 1 \\
\quad \text{Chng}[m] = \text{min} \\
\text{return } \text{Chng}[p]
\]

array holding optimal values for all values \( \leq p \).

find optimal change amounts starting with 1.
Making Change – Dynamic Programming

```plaintext
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 ≤ p.
Making Change – Dynamic Programming

changeDP(p)

Chng[0,...,p] = [0,...,0]

for m = 1 to p

    min = \infty

    for d_i \leq 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 = \infty
        for d_i \leq m
            if Chng[m - d_i] + 1 < min
                min = Chng[m - d_i] + 1
        Chng[m] = min
    return Chng[p]
Making Change – Dynamic Programming

\( \text{changeDP}(p) \)

\[ \text{Chng}[0, ..., p] = [0, ..., 0] \]

\[ \text{for } m = 1 \text{ to } p \]

\[ \text{min} = \infty \]

\[ \text{for } d_i \leq m \]

\[ \text{if } \text{Chng}[m - d_i] + 1 < \text{min} \]

\[ \text{min} = \text{Chng}[m - d_i] + 1 \]

\[ \text{Chng}[m] = \text{min} \]

\[ \text{return } \text{Chng}[p] \]

Running time?

Outer for loop \( \in O(p) \)

Inner for loop \( \in O(k) \)

Total \( \in O(pk) \)
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.
Rod Cutting

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>profit</td>
<td>$1</td>
<td>$5</td>
<td>$8</td>
<td>$9</td>
</tr>
</tbody>
</table>
Rod Cutting

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>profit</td>
<td>$1</td>
<td>$5</td>
<td>$8</td>
<td>$9</td>
</tr>
</tbody>
</table>

profit = $9
### Rod Cutting

<table>
<thead>
<tr>
<th>Length</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1</td>
</tr>
<tr>
<td>2</td>
<td>$5</td>
</tr>
<tr>
<td>3</td>
<td>$8</td>
</tr>
<tr>
<td>4</td>
<td>$9</td>
</tr>
</tbody>
</table>

**Profit = $9**

**Profit = $9**
Rod Cutting

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>profit</td>
<td>$1</td>
<td>$5</td>
<td>$8</td>
<td>$9</td>
</tr>
</tbody>
</table>

profit = $9

profit = $9

profit = $9
Rod Cutting

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>profit</td>
<td>$1</td>
<td>$5</td>
<td>$8</td>
<td>$9</td>
</tr>
</tbody>
</table>

Given a rod of length $n$ and a table of prices, determine the maximum profit obtainable by cutting the rod up and selling the pieces.
Given a rod of length \( n \) and a table of prices, determine the maximum profit obtainable by cutting the rod up and selling the pieces.
Rod Cutting

Can we use dynamic programming?
Does the rod cutting problem have optimal substructure?
Does the rod cutting problem have optimal substructure?

I.e. Does an optimal solution to an instance imply an optimal solution to a smaller instance?
Rod Cutting

Does the rod cutting problem have optimal substructure?

Yes! Given an optimal partition for $n$, every subset of that partition must also be optimal!
Rod Cutting

\[ O_n = \text{optimal profit from partitioning rod of length } n. \]

\[ p_i = \text{profit for rod of length } i. \]
Rod Cutting

$O_n =$ optimal profit from partitioning rod of length $n$.
$p_i =$ profit for rod of length $i$.

$O_n =$ ?

How can we recursively calculate $O_n$ from smaller solutions, leveraging the optimal substructure?
Rod Cutting

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>profit</td>
<td>$1</td>
<td>$5</td>
<td>$8</td>
<td>$9</td>
<td>...</td>
<td>$?</td>
</tr>
</tbody>
</table>

\[ O_n = \text{optimal profit from partitioning rod of length } n. \]
\[ p_i = \text{profit for rod of length } i. \]

\[ O_{100} = p_3 + O_{97} \]

How can we recursively calculate \( O_n \) from smaller solutions, leveraging the optimal substructure?
Rod Cutting

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>profit</td>
<td>$1</td>
<td>$5</td>
<td>$8</td>
<td>$9</td>
<td>...</td>
<td>$?</td>
</tr>
</tbody>
</table>

$O_n$ = optimal profit from partitioning rod of length $n$.

$p_i$ = profit for rod of length $i$.

$$O_n = ?$$

How can we recursively calculate $O_n$ from smaller solutions, leveraging the optimal substructure?
Rod Cutting

<table>
<thead>
<tr>
<th>length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>profit</td>
<td>$1</td>
<td>$5</td>
<td>$8</td>
<td>$9</td>
<td>...</td>
<td>$?</td>
</tr>
</tbody>
</table>

$O_n = \text{optimal profit from partitioning rod of length } n.$

$p_i = \text{profit for rod of length } i.$

$$O_n = \max_{1 \leq i \leq n} (p_i + O_{n-i})$$
Rod Cutting

Algorithm?
Rod Cutting

\[
\text{partition}(n, p) \\
\quad r[0, \ldots, n] = [0, \ldots, 0] \\
\quad \text{for } m = 1 \text{ to } n \\
\quad \quad \max = 0 \\
\quad \quad \text{for length } l \leq m \\
\quad \quad \quad \text{if } r[m - 1] + p_1 > \max \\
\quad \quad \quad \quad \max = r[m - 1] + p_1 \\
\quad \quad r[m] = \max \\
\quad \text{return } r[n]
\]
Rod Cutting

partition(n, p)
    r[0,...,n] = [0,...,0]
    for m = 1 to n
        max = 0
        for length l ≤ m
            if r[m - l] + p_l > max
                max = r[m - l] + p_l
        r[m] = max
    return r[n]

Running Time?
Rod Cutting

partition(n, p)
    r[0,...,n] = [0,...,0]
    for m = 1 to n
        max = 0
        for length l ≤ m
            if r[m - l] + p_l > max
                max = r[m - l] + p_l
        r[m] = max
    return r[n]

Running Time? $O(n^2)$
Matrix-Chain Multiplication

\[
\begin{align*}
\text{A} & \quad \text{X} & \quad \text{B} & \quad = & \quad \text{C} \\
10 \times 100 & & 100 \times 5 & & 10 \times 5 \\
\end{align*}
\]
Matrix-Chain Multiplication

\[ A \times B = C \]

Where:
- \( A \) is a 10 x 100 matrix.
- \( B \) is a 100 x 5 matrix.
- \( C \) is a 10 x 5 matrix.
Matrix-Chain Multiplication

\[ A \times B = C \]

Number of scalar multiplications?
Matrix-Chain Multiplication

\[
\begin{align*}
A \times B &= C \\
10 \times 100 &\quad \times \quad 100 \times 5 &\quad = &\quad 10 \times 5 \\
\text{Number of scalar multiplications?} &\quad 10 \times 5 \times 100 = 5000
\end{align*}
\]
Matrix-Chain Multiplication

\[ A \times B = C \]

Number of scalar multiplications?
\[ 10 \times 5 \times 100 = 5000 \]
Matrix-Chain Multiplication

Number of scalar multiplications?

$$10 \times 5 \times 100 = 5000$$

$$n \times p \times m$$
Matrix-Chain Multiplication

\[ A \times B \times C = D \]

- \( A \): \(10 \times 100\)
- \( B \): \(100 \times 5\)
- \( C \): \(5 \times 50\)
- \( D \): \(10 \times 50\)
Matrix-Chain Multiplication

\[
\begin{align*}
A \times (B \times C) & : \\
(A \times B) \times C & : \\
A \times (B \times C) & :
\end{align*}
\]
Matrix-Chain Multiplication

\[ A \times B \times C = \]

\[ 10 \times 100 \times 5 \times 50 \times 10 \times 50 \]

- \((A \times B) \times C : 10 \times 5 \times 100\)
- \(A \times (B \times C) : 100 \times 5 \times 50\)
Matrix-Chain Multiplication

\[(A \times B) \times C : \]
\[10 \times 5 \times 100 \]
\[+10 \times 50 \times 5\]

\[A \times (B \times C) : \]
Matrix-Chain Multiplication

\[
\begin{align*}
A \times B \times C &= \\
(\text{A x B)} \times \text{C} : &
\quad 10 \times 5 \times 100 \\
&\quad + 10 \times 50 \times 5 \\
&\quad = 7500 \\
A \times (\text{B x C}) : &
\end{align*}
\]
Matrix-Chain Multiplication

\[(A \times B) \times C : \quad 10 \times 5 \times 100 \quad + \quad 10 \times 50 \times 5 \quad = \quad 7500\]

\[A \times (B \times C) : \quad 100 \times 50 \times 5 \quad + \quad 10 \times 50 \times 100 \quad = \quad 75000\]
Matrix-Chain Multiplication

Can we use dynamic programming?
Matrix-Chain Multiplication

Optimal substructure?
Matrix-Chain Multiplication

An optimal parenthesization must end with multiplying two matrices. The parenthesizations that led to those two matrices must also be optimal.
Optimal substructure?

An optimal parenthesization must end with multiplying two matrices. The parenthesizations that led to those two matrices must also be optimal.
Matrix-Chain Multiplication

Algorithmic Game Plan?
Matrix-Chain Multiplication

Algorithmic Game Plan?
Check all possible splits within full matrix multiplication. Select minimum. Need to work in bottom up a fashion to have already calculated the optimal sub problems.
Matrix-Chain Multiplication

Algorithmic Game Plan?
Check all possible splits within full matrix multiplication. Select minimum. Need to work in bottom up a fashion to have already calculated the optimal sub problems.
Matrix-Chain Multiplication

Algorithmic Game Plan?

Check all possible splits within full matrix multiplication. Select minimum. Need to work in bottom up a fashion to have already calculated the optimal sub problems.
Matrix-Chain Multiplication

Algorithmic Game Plan?
Check all possible splits within full matrix multiplication.
Select minimum. Need to work in bottom up a fashion to have already calculated the optimal sub problems.
Matrix-Chain Multiplication

Algorithmic Game Plan?
Check all possible splits within full matrix multiplication. Select minimum. Need to work in bottom up a fashion to have already calculated the optimal sub problems.
Matrix-Chain Multiplication

Algorithmic Game Plan?

Check all possible splits within full matrix multiplication. Select minimum. Need to work in bottom up a fashion to have already calculated the optimal sub problems.
Matrix-Chain Multiplication

Algorithmic Game Plan?
Check all possible splits within full matrix multiplication. Select minimum. Need to work in bottom up a fashion to have already calculated the optimal sub problems.
Algorithmic Game Plan?
Check all possible splits within full matrix multiplication. Select minimum. Need to work in bottom up a fashion to have already calculated the optimal sub problems.
Algorithmic Game Plan?
Check all possible splits within full matrix multiplication. Select minimum. Need to work in bottom up a fashion to have already calculated the optimal sub problems.
Matrix-Chain Multiplication

Algorithmic Game Plan?
Check all possible splits within full matrix multiplication. Select minimum. Need to work in bottom up a fashion to have already calculated the optimal sub problems.