Dynamic Programming
Ch. 15
This Course

• Sorting
  – A necessary tool for managing and navigating a variety of datasets.
  – A great context in which to learn computer science
    • Analyze Complexity
    • Prove Correctness

• Data Structures
  – Explore variety of structures beyond arrays and linked lists.
  – How you store data affects how easily you can manage and navigate it.

• Optimization Problems
  – Learn sophisticated methods that are alternates to exponential brute force approach.

• Graph Problems
  – Applicable to wide range of problems (many seemingly unrelated problems map to a graph problem).
  – Another great context in which to learn computer science.
Dynamic Programming

no relation to dynamic memory allocation

• Solution for Optimization (maximize or minimize value).

• Addresses issues of exponential growth typical of brute force.

• Takes advantage of optimal substructure and overlapping subproblems.

• Efficiency gains from reusing solutions to subproblems.

• Reuses solutions by top-down memoizing (storing subproblem solutions) or building a solution from the bottom-up.
You Have a Job Interview
An Optimization Problem

- Brute Force
- Greedy
- Constraint Satisfaction
- Decision Tree
- Informed Search

How will you rate these?
Decision Spaces
One Solution but Not the “Greedy” One

anomalous solution
follow the gradient
local maxima
Decision Space

Maximization Problem

(The solution is the one with the highest possible score.)

Greedy

best *local* part of best *global*

Z-axis
Score of shirt/suit combo

Y-axis
1000 suits, sorted by rank

X-axis
1000 shirts, sorted by rank

THE solution (i.e. max score)
Decision Spaces
Multiple Solutions

all viable, equally good, solutions

all viable, equally good solutions
Optimal Substructure

The optimal solution contains within it optimal solutions to subproblems.

What is an optimal solution (maximization)?

\[ S(X_i) : \text{Value of solution to } X \text{ that includes choice } i \]

Optimal = \( \max_{\text{over all } i's} (S(X_i)) \)

What are the set of solutions?

\[ S(X_i) = \text{value(choice i) + } S(X_{1i}) + S(X_{2i}) \]

where \( X_{1i} \) and \( X_{2i} \) are the subproblems resulting from choice \( i \)

Optimal Substructure

\[ \max_{\text{over all } i} (S(X_i)) \text{ includes } \max(S(X_{1i})) \text{ and } \max(S(X_{2i})) \]
What is an optimal solution (of maximization)?

$S(X_i) = \text{a solution to problem } X \text{ (not the solution)}$

Optimal = $\text{MAX}_{\text{over all } i} (S(X_i))$

How do you find THE solution?

Calculate value of each choice of $i$ and select the Max.

$T(n) = \Omega(2^n)$

Overlapping Subproblems

$X(1..2) = X(1..1) + X(2..2)$

$X(1..n) = X(1..1) + X(2..n)$

$X(1..2) = X(1..1) + X(2..2)$

$X(1..n) = X(1..2) + X(3..n)$

$X(2..n) = X(2..3) + X(3..n)$

$X(1..n) = X(1..n-1) + X(n..n)$
Decision Tree Visualization
(Make the Cut or Don’t)

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

Left: made cut $i$
Right: did not make cut $i$

Leaves reflect every possible combination of cut/no cut.
Subproblem Visualization
(Size of Subproblems)

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

$r_n = \max (p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1)$

for any rod the length of $n$, there are $n-1$ subproblem configurations
### Subproblem Visualization

(Length of the 1st Cut)

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

\[
r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})
\]

\[
r_n = \max (p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1)
\]
Exponential Recursive Rod Cutting

CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = -∞
4  for i = 1 to n
5      q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q

Taking Advantage of Optimal Substructure

The optimal solution of a subproblem (n-i) is part of the optimal solution of problem n.

The oracle tells you that the first cut at length i will lead to optimal solution.

Use PROOF BY CONTRADICTION to show that global optimal solution based on choice i contains optimal solution for length n-i.
Top-Down (Recursive)
Memoized Rod Cutting

MEMOIZED-CUT-ROD(p, n)
1 let r[0 .. n] be a new array
2 for i = 0 to n
3 \( r[i] = -\infty \)
4 return MEMOIZED-CUT-ROD-AUX(p, n, r)

MEMOIZED-CUT-ROD-AUX(p, n, r)
1 if \( r[n] \geq 0 \)
2 return \( r[n] \)
3 if \( n == 0 \)
4 \( q = 0 \)
5 else \( q = -\infty \)
6 for i = 1 to n
7 \( q = \max(q, p[i] + \text{MEMOIZED-CUT-ROD-AUX}(p, n - i, r)) \)
8 \( r[n] = q \)
9 return q

CUT-ROD(p, n)
1 if \( n == 0 \)
2 return 0
3 \( q = -\infty \)
4 for i = 1 to n
5 \( q = \max(q, p[i] + \text{CUT-ROD}(p, n - i)) \)
6 return q

Taking Advantage of Overlapping Subproblems
Bottom-Up (Iterative)
Rod Cutting

**BOTTOM-UP-CUT-ROD**(p, n)
1 let r[0 .. n] be a new array
2 r[0] = 0
3 for j = 1 to n
4 q = -∞
5 for i = 1 to j
6 q = max(q, p[i] + r[j - i])
7 r[j] = q
8 return r[n]

**MEMOIZED-CUT-ROD-AUX**(p, n)
1 if r[n] ≥ 0
2 return r[n]
3 if n == 0
4 q = 0
5 else q = -∞
6 for i = 1 to n
7 q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
8 r[n] = q
9 return q
EXTENDED-BOTTOM-UP-CUT-ROD\((p, n)\)
1 let \(r[0 \ldots n]\) and \(s[0 \ldots n]\) be new arrays
2 \(r[0] = 0\)
3 \textbf{for } j = 1 \textbf{ to } n
4 \hspace{1cm} q = -\infty
5 \hspace{1cm} \textbf{for } i = 1 \textbf{ to } j
6 \hspace{1.5cm} \textbf{if } q < p[i] + r[j - i]
7 \hspace{2cm} q = p[i] + r[j - i]
8 \hspace{1.5cm} s[j] = i
9 \hspace{1cm} r[j] = q
10 \textbf{return } r\text{ and } s

BOTTOM-UP-CUT-ROD\((p, n)\)
1 let \(r[0 \ldots n]\) be a new array
2 \(r[0] = 0\)
3 \textbf{for } j = 1 \textbf{ to } n
4 \hspace{1cm} q = -\infty
5 \hspace{1cm} \textbf{for } i = 1 \textbf{ to } j
6 \hspace{1.5cm} q = \max(q, p[i] + r[j - i])
7 \hspace{1.5cm} r[j] = q
8 \textbf{return } r[n]

PRINT-CUT-ROD-SOLUTION\((p, n)\)
1 \((r, s) = \text{EXTENDED-BOTTOM-UP-CUT-ROD}(p, n)\)
2 \textbf{while } n > 0
3 \hspace{1cm} \text{print } s[n]
4 \hspace{1cm} n = n - s[n]
Applying Dynamic Programming

1. Characterize optimal structure using optimal substructure.
   – Proof by Contradiction of Optimal Substructure

2. Recursively define the solution.
   – Define value/cost in terms of value/cost subproblems.
   – Define the stopping or base condition.

3. Compute value of optimal solution.
   – Define algorithm to compute value/cost of optimal solution.

   – Modify algorithm to save solution and print out.
Matrix Multiplication
A Minimization Problem

```
MATRIX-MULTIPLY(A, B)
1 if A.columns ≠ B.rows
2 error "incompatible dimensions"
3 else let C be a new A.rows × B.columns matrix
4   for i = 1 to A.rows
5     for j = 1 to B.columns
6       c_{ij} = 0
7     for k = 1 to A.columns
8       c_{ij} = c_{ij} + a_{ik} · b_{kj}
9   return C
```

\[ A_1A_2A_3...A_n \]

\[
P(n) = \begin{cases} 
1 & \text{if } n = 1, \\
\sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2.
\end{cases}
\]

P(n) = number of parenthesizations with n matrices
Characterize Optimal Substructure

\[ A_1A_2A_3\ldots A_n \]

Just like cutting the rod: consider a split at all locations.

For any split, 2 subproblems arise.

The optimal solution must contain the optimal solution to the subproblems.

If the optimal solution contains choice i, then the optimal solutions to \( A_1..A_i \) and \( A_{i+1}..A_n \) are part of the optimal solution.
Define Recursive Solution

\[ A_i = p_{i-1} \times p_i \text{ matrix} \]
\[ A_i A_j = p_{i-1} \times p_j \text{ matrix with } p_{i-1} p_i p_j \text{ operations} \]
MATRIX-CHAIN-ORDER(p)

1 \( n = p\.length - 1 \)
2 let \( m[1..n, 1..n] \) and \( s[1..n-1, 2..n] \) be new tables
3 for \( i = 1 \) to \( n \)
4 \( m[i, i] = 0 \)
5 for \( l = 2 \) to \( n \)  // \( l \) is the chain length
6 \( \phantom{m[i, i]} = \infty \)
7 \( m[i, j] = \infty \)
8 for \( k = i \) to \( j - 1 \)
9 \( q = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j \)
10 if \( q < m[i, j] \)
11 \( m[i, j] = q \)
12 \( s[i, j] = k \)
13 return \( m \) and \( s \)
MATRIX-CHAIN-ORDER(p)
1  n = p.length - 1
2  let m[1 .. n, 1 .. n] and s[1 .. n - 1, 2 .. n] be new tables
3  for i = 1 to n
4     m[i, i] = 0
5  for l = 2 to n  // l is the chain length
6     for i = 1 to n - l + 1
7        j = i + l - 1
8        m[i, j] = ∞
9        for k = i to j - 1
10           q = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j
11           if q < m[i, j]
12              m[i, j] = q
13              s[i, j] = k
14  return m and s

PRINT-OPTIMAL-PARENS(s, i, j)
1  if i == j
2     print "A"_i;
3  else print "("
4     PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5     PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6     print ")"
Theorem 15.1: (Optimal substructure of an LCS)

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of $X$ and $Y$.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and $Z_{k-1}$ is an LCS of $X_{m-1}$ and $Y_{n-1}$.

2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that $Z$ is an LCS of $X_{m-1}$ and $Y$.

3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that $Z$ is an LCS of $X$ and $Y_{n-1}$. 

If optimal solution includes choice $i$ then optimal solutions to resulting subproblems must be part of optimal solution.
If optimal solution includes choice i then optimal solutions to resulting subproblems must be part of optimal solution

Theorem 15.1: (Optimal substructure of an LCS)
Let \( X = \langle x_1, x_2, \ldots, x_m \rangle \) and \( Y = \langle y_1, y_2, \ldots, y_n \rangle \) be sequences, and let \( Z = \langle z_1, z_2, \ldots, z_k \rangle \) be any LCS of \( X \) and \( Y \).

1. If \( x_m = y_n \), then \( z_k = x_m = y_n \) and \( Z_{k-1} \) is an LCS of \( X_{m-1} \) and \( Y_{n-1} \).

2. If \( x_m \neq y_n \), then \( z_k \neq x_m \) implies that \( Z \) is an LCS of \( X_{m-1} \) and \( Y \).

3. If \( x_m \neq y_n \), then \( z_k \neq y_n \) implies that \( Z \) is an LCS of \( X \) and \( Y_{n-1} \).
Theorem 15.1: (Optimal substructure of an LCS)

Let \( X = \langle x_1, x_2, \ldots, x_m \rangle \) and \( Y = \langle y_1, y_2, \ldots, y_n \rangle \) be sequences, and let \( Z = \langle z_1, z_2, \ldots, z_k \rangle \) be any LCS of \( X \) and \( Y \).

1. If \( x_m = y_n \), then \( z_k = x_m = y_n \) and \( Z_{k-1} \) is an LCS of \( X_{m-1} \) and \( Y_{n-1} \).

2. If \( x_m \neq y_n \), then \( z_k \neq x_m \) implies that \( Z \) is an LCS of \( X_{m-1} \) and \( Y \).

3. If \( x_m \neq y_n \), then \( z_k \neq y_n \) implies that \( Z \) is an LCS of \( X \) and \( Y_{n-1} \).

\[
c[i, j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0, \\
\max(c[i - 1, j - 1], c[i - 1, j]) & \text{if } i, j > 0 \text{ and } x_i = y_j, \\
\max(c[i, j - 1], c[i - 1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j.
\end{cases}
\]
**LCS : Bottom-Up with Solution**

**LCS-LENGTH(X, Y)**

1. \( m = X.\text{length} \)
2. \( n = Y.\text{length} \)
3. let \( b[1..m, 1..n] \) and \( c[0..m, 0..n] \) be new tables
4. for \( i = 1 \) to \( m \)
5. \( c[i, 0] = 0 \)
6. for \( j = 0 \) to \( n \)
7. \( c[0, j] = 0 \)
8. for \( i = 1 \) to \( m \)
9. for \( j = 1 \) to \( n \)
10. if \( x_i == y_j \)
11. \( c[i, j] = c[i - 1, j - 1] + 1 \)
12. \( b[i, j] = "\downarrow" \)
13. elseif \( c[i - 1, j] \geq c[i, j - 1] \)
14. \( c[i, j] = c[i - 1, j] \)
15. \( b[i, j] = "\uparrow" \)
16. else \( c[i, j] = c[i, j - 1] \)
17. \( b[i, j] = "\leftarrow" \)
18. return \( c \) and \( b \)

![LCS-bottom-up-diagram](image-url)
Dynamic Programming

no relation to dynamic memory allocation

• Solution for Optimization (maximize or minimize value).

• Addresses issues of exponential growth typical of brute force.

• Takes advantage of optimal substructure and overlapping subproblems.

• Efficiency gains from reusing solutions to subproblems.

• Reuses solutions by top-down memoizing (storing subproblem solutions) or building a solution from the bottom-up.
• Convince yourself brute force is exponential.
• Define and prove optimal substructure.
  – Proof by Contradiction
• Demonstrate overlapping subproblems.
  – Evident in a recursive definition of the problem.
• Develop Algorithm for Cost of Solution (and solution)
• Analyze Algorithm for Growth