Dynamic Programming (DP)

- Refresher
- Alternating Coin Game
- Optimal Binary Search Trees
- Parsing for Context Free Grammars

**Refresher:**

DP $\approx$ Clever Brute-Force

- non-oblivious: does not re-solve subproblems encountered before

Applicable when:

i. Guessing part of solution to problem, allows breaking its solution into solving subproblems.

ii. The same is true for every subproblem arising recursively.

iii. The total number of subproblems arising recursively is polynomial.

iv. The #guesses per subproblem is polynomial.

v. The time needed to find problem's solution from subproblem solutions is polynomial.
Example 1: Alternating Coin Game

- Row of $n$ coins of values $v_1, v_2, \ldots, v_n$, where $n$ is even.
- Players play in turns.
- In each turn, a player removes either the first or the last coin, removes it permanently and receives its value.

E.g., $3 \ 5 \ 9 \ 2 \ 4 \ 7$

<table>
<thead>
<tr>
<th>Player</th>
<th>Coins</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>7</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>5</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
</tr>
<tr>
<td>P1</td>
<td>4</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
</tr>
</tbody>
</table>

Player 1: $\$16$
Player 2: $\$14$
Player 1 wins
**Question:** Given \( v_1, v_2, \ldots, v_n \) can player 1 win no matter how well player 2 plays?

\( v(i, j) \): value player 1 can guarantee to himself if remaining coins are \( v_i, v_{i+1}, \ldots, v_j \)

\[ v(i, j) = \max ( v_i + v(i+1, j), v_j + v(i, j-1) ) \]

\[ Q: \] who plays if remaining coins are \( v_i, \ldots, v_j \)?

\[ A: \]
- if \( j-i \) is even, player 2
- \( j-i \) is odd, player 1

\[ Q_2: \] how relate \( v(i, j) \) to subproblems?

\[ A: \]
- two cases:
  - if \( j-i \) is odd (player 1 plays)
    - guess whether to pick coin \( v_i \) or coin \( v_j \)
    - choose the one to maximize \( v(i, j) \)
(2) \( j-i \) is even (player 2 plays)

- guess whether he picks \( v_i \) or \( v_j \)
- choose the one to minimize \( v(i,j) \)

\[
v(i,j) = \min\left( v(i+1,j), v(i,j-1) \right)
\]

(1) \[
V(i,j) = \begin{cases} 
\max\left( v_i + v(i+1,j), v_j + v(i,j-1) \right), & \text{if } j-i \text{ is odd} \\
\min\left( v(i+1,j), v(i,j-1) \right), & \text{if } j-i \text{ is even}
\end{cases}
\]

Runtime = \( O(n^2) \times O(1) \)

\[\uparrow\]

\# subproblems \quad \text{guess + combination work per subproblem}

Memoizing vs Iterating

(1) **Memoizing**: Implement recursive algorithm for (1), using look-up table \( L(i,j) \).

- If during execution \( L(i,j)=\text{null} \), subproblem
\( v(i,j) \) has not been solved before.

- So if need \( L(i,j) \) and \( L(i,j) = \text{null} \), then compute \( L(i,j) \) and store it in look-up table when done (so that it's used in future).

\[ \begin{align*}
\text{(2) Iterating:} & \quad \text{Harder, need to understand sub-problem dependencies and solve sub-problems in order which never requires solutions to subproblems not solved before.} \\
& \quad \text{in our case solve in increasing } j - i.
\end{align*} \]

**Example 2:** Optimal Binary Search Trees

Input: keys \( k_1, k_2, \ldots, k_n \), where \( k_1 < k_2 < \cdots < k_n \)

search probabilities \( p_1, \ldots, p_n \)

Goal: store keys in BST to minimize expected search cost, namely

\[
\text{minimize } \sum_{i=1}^{n} p_i \cdot (\text{depth}(k_i) + 1)
\]
Real-world Application: English → French dictionary
- want more frequent words to have faster look-up time

Algorithm 1: Enumerate all possible BSTs on $k_1, \ldots, k_n$

For each binary tree w/ $n$ nodes, unique way to fill the nodes w/ keys $k_1, \ldots, k_n$

But how many binary trees w/ $n$ nodes?
"Catalan numbers" \( \frac{1}{n+1} \binom{2n}{n} \sim \frac{4^n}{n^{3/2} \sqrt{\pi}} \)

Exponentially many!

Algorithm 2:
- Place key \( k_i \) with highest search probability \( p_i \) at root of tree
- Then recursively solve problem for left keys \( k_1, \ldots, k_{i-1} \) and right keys \( k_{i+1}, \ldots, k_n \)

Q: Correct?

A: No, here is a counter-example:

\[ \text{cost} = 10 + 2 \cdot 1 + 2 \cdot 9 + 3 \cdot 8 = 54 \]

\[ \text{Cost} = 9 + 2 \cdot 10 + 2 \cdot 8 + 3 \cdot 4 = 48 \]
**Algorithm 3: Dynamic Programming**

Q: What guess would allow me to split into subproblems?

A: Which key to put in the root

\[ e(i, j) = \text{cost of optimal BST containing keys } k_i, k_{i+1}, \ldots, k_j \]

\[ e(i, j) = \begin{cases} 
  p_i & \text{if } i = j \\
  \min_{i \leq r \leq j} \left( e(i, r-1) + e(r+1, j) + \sum_{z=i}^{j} p_z \right) & \text{o.w.}
\end{cases} \]

Run-time:

\[ O(n^2) \times O(n) \]

#subproblems work per subproblem

accounts for \( p_r \) of the root node, as well as the increase in depth by 1 of all keys in subtrees of \( k_r \)
Example 3: Parsing Context Free Grammars

\[ \begin{align*}
\text{INTEGER} & \rightarrow \text{SIGN DIGITS} \\
\text{SIGN} & \rightarrow + 1 - 1 E \rightarrow \text{empty string} \\
\text{DIGITS} & \rightarrow \text{DIGIT} | \text{DIGIT DIGITS} \\
\text{DIGIT} & \rightarrow 0 1 2 3 4 5 6 7 8 9 \\
\text{EXPR} & \rightarrow \text{EXPR} \ast \text{EXPR} \\
& \quad | \text{EXPR} + \text{EXPR} \\
& \quad | ( \text{EXPR} ) | \text{TUPLE} | \text{INTEGER} \\
\text{TUPLE} & \rightarrow ( ) | ( \text{EXPRS} ) \\
\text{EXPRS} & \rightarrow \text{EXPR} | \text{EXPR}, \text{EXPRS} \\
\end{align*} \]

Terminal symbols \( \Sigma = \{ +, -, 0, 1, \ldots, 9, \ast, (, ), , \} \)

Non-terminal symbols \( \mathcal{N} = \{ \text{INTEGER}, \text{SIGN}, \text{DIGITS}, \text{DIGIT}, \text{EXPR}, \text{EXPRS}, \text{TUPLE} \} \)

Special starting non-terminal symbol: \( \text{EXPR} \)

Rules \( R \): see above

Problem: given string \( s \in \Sigma^* \), decide whether it can be generated by the CFG\( \) starting w/ the starting non-terminal symbol.
e.g. \((5 + 3, (7 * (42)))\)
**DP subproblem**

$$\pi(i,j,X)$$: 0/1 depending on whether string $$S[i:j]$$ can be generated starting from $$X$$.

**guessing**: what rule $$r: X \rightarrow X_1 \ldots X_m$$ to apply and at which index we each $$X_k$$ starts (note that $$w_1 = 1$$ is forced).

**combining**:

$$\pi(i,j,X) = 1 \iff$$

for some guess $$(X \rightarrow X_1 \ldots X_m, w_2, \ldots, w_m)$$ it holds that: $$\pi(w_1, w_{k+1}, X_e) = 1$$, $$\forall e = 1, \ldots, m-1$$

**base cases**:

$$\pi(i,j, \text{terminal}) = \begin{cases} 1, & \text{if } j = i+1 \land S[i] = \text{terminal symbol} \\ 0, & \text{otherwise} \end{cases}$$

$$\pi(i,j, \varepsilon) = \begin{cases} 1, & \text{if } j = i \\ 0, & \text{otherwise} \end{cases}$$

$$\pi(i,j, \text{anything}) = 0 \text{ if } j < i$$

**Interested in** \(\pi(0, |S|, \text{non-terminal})\).
Runtime:

\# subproblems: \leq |S|^2 \times |N|

\# guesses for subproblem \pi(i,j,x): \leq (\# \text{rules}) \times |S|^{m-1}

work per guess: \O(m)

\Rightarrow \text{runtime} \sum_{(i,j,x)} (\# \text{rules}) \times |S|^{m-1} \times \O(m) \times \# \text{rules}

= \O\left(|S|^2 |N| |CFG| |S|^{m-1} \right)

= \O\left(|S|^{m+1} |N| |CFG| \right)

- fine if RHS's are bounded (small m)
- if not, convert to Chomsky-Normal-Form
  all RHS's are of the form nonterm nonterm or terminal

Challenge: DP that runs in \O(|S|^2 |N| |CFG|) without resorting to Chomsky Normal Form?