Lecture 6 Dynamic Programming

- Longest Palindromic Subsequence
- Optimal Binary Search Trees
- Alternating Coin Game
**Longest Palindromic Subsequence**

**Def** A palindrome is a string that is unchanged when reversed.

**Examples**
- radar, civic, deified, a, kayak
- restriction enzymes
- Haydn's Symphony #47

**Given** string \( x[1..n] \) (\( n \geq 1 \))

**Find** longest palindrome that is a subsequence

**Example**
- given "Character"

(answer always has length \( \geq 1 \))
Idea:

Let \( L(i,j) \) denote length of longest palindromic subsequence of \( X[i...j] \) \((i \leq j)\)

Compute \( L(i,j) \):

\[
\begin{align*}
&\text{if } i == j \quad \text{return } 1 \\
&\text{if } X[i] \neq X[j] \\
&\quad \quad \text{return } \max(L(i+1,j), L(i, j-1)) \\
&\text{if } X[i] = X[j] \\
&\quad \quad \text{if } i+1 == j \\
&\quad \quad \quad \text{return } 2 \\
&\quad \quad \text{return } 2 + L(i+1, j-1)
\end{align*}
\]

Problem: runs in exponential time!

e.g. if all \( X[i] \) distinct symbols (no palindromes)

run time on inputs of length \( n \)

\[
T(n) = \begin{cases} 
1 & n=1 \\
2T(n-1) & n>1 
\end{cases}
\]

\[
= 2^{n-1}
\]

↑ # of subproblems solved by program.
How many subproblems? only one for each $i,j$ pair $i \leq j$

so only $\binom{n}{2} = \Theta(n^2)$ distinct subproblems.

Solve each subproblem only once!

New runtime

$$\Theta(n^2) \cdot \Theta(1) = \Theta(n^2)$$

$\# \text{ subproblems}$ $\quad \text{Time to solve subproblem}$

$\quad \text{GIVEN solutions to smaller subproblems}$

Implementation

1. "Bottom up" - solve subproblems in order of increasing size $j-i$

so all smaller problems solved first

2. "Memoize" - "hash" inputs/outputs for $L$ into 2D array by $(i,j)$

check to see if already solved before resolving
Features of DP problems:

- recursive solution exponential
- polynomial number of subproblems
- memoization or bottom up gives poly runtime

Optimal Binary Search Trees

Given: keys $K_1, K_2, \ldots, K_n$ $K_1 < K_2 < \ldots < K_n$ (wlog $K_i = i$)
weights $w_1, w_2, \ldots, w_n$ (search probabilities)

Find: BST $T$ that minimizes

$$\sum_{i=1}^{n} w_i \cdot \left( \text{depth}_T (K_i) + 1 \right)$$

for convenience only, doesn't affect output.

Example: $w_i = p_i = \text{prob of searching for key } i$
then minimizing search cost!

```
  2
 /|
/  |
1  3
   /|
   /  |
  4  0
```
How many possibilities?

# trees on \( n \) keys \( \approx \Theta \left( \frac{4^n}{n^{3/2}} \right) \) exponential!

- \( n=2 \)
  - Cost: \( w_1 + 2w_2 \) and \( w_2 + 2w_1 \)

- \( n=3 \)
  - Cost: \( 3w_1 + 2w_2 + w_3 \), \( 2w_1 + 3w_2 + w_3 \), \( w_1 + 3w_2 + 2w_3 \), \( w_1 + 2w_2 + 3w_3 \)

Idea:

let \( w(i,j) = w_i + w_{i+1} + \ldots + w_j \)

\( e(i,j) \) = cost of opt BST on \( k_i \ldots k_j \)

we want \( e(i,n) \)

Is greedy possible?

1. pick \( k_r \) in some greedy manner?
   e.g. \( \max w_r \)
2. solve subproblems

we don't know how to do this!
DP Strategy: Guess all roots + pick best

\[ e(i,j) = \begin{cases} w_i & \text{if } i = j \\ \min_{i \leq r \leq j} \left( e(i,r-1) + e(r+1,j) + w(i,j) \right) & \text{else} \end{cases} \]

Why?

\[ \text{total cost: } \sum_{l=1}^{j} e(i,j) = W(i,j) \]

Complexity

\[ \Theta(n^2) \cdot \Theta(n) = \Theta(n^3) \]
Alternating Coin Game

Given: $n$ coin values $V_1, \ldots, V_n$ $n$ even

In each turn, a player selects 1st or last coin from the row, removes it permanently & receives value of coin. (2 player game)

Question: Can 1st player always win?

Try: 4  42  39  17  25  6

Strategy

1) compare $V_1 + V_3 + \ldots + V_{n-1}$ vs. $V_2 + V_4 + \ldots + V_n$ & pick greater

2) only pick from chosen subset

Can you do this?

YES

$V(i,j) = \max$ value we can win if it is our turn & only coins $V_i, \ldots, V_j$ remain

e.g., $V(i,i)$ just pick $i$

$V(i,i+1)$ pick max of 2
\[ v_{i,j} = \max \left\{ \begin{array}{l} \text{range becomes } (i+1,j) \vdash v_i \\text{pick } v_i \\ \text{range becomes } (i,j-1) \vdash v_{i,j} \\text{pick } v_j \end{array} \right\} \]

\* \( V(i+1,j) \) subproblem with \textbf{opponent going 1st!} \[ \Rightarrow \text{we are guaranteed min } \min\{v_{i+1,j-1}, v_{i+2,j}\} \]

So \[ v_{i,j} = \max \left\{ \min \left\{ V(i+1,j-1) + v_i, V(i+2,j) \right\} \right\} \]

\[ \min \left\{ \begin{array}{l} V(i,j-2) + v_{i,j} \\ V(i+1,j-1) \end{array} \right\} \]

\textbf{Complexity?} \[ \Theta(n^2), \quad \Theta(1) = \Theta(n^2) \]

\[ \uparrow \text{number of subproblems} \quad \uparrow \text{time per subproblem} \]