Dynamic Programming
An Alg Design Technique

Thousands of Applications
Kai-Fu Lee's Speech Alg
Genome Seg
Clark-Bryant Hardware checking

We will do
1) Classic Ex
2) Not so Classic

Your solutions should have a special form.
Optimal Binary Search Trees

Input:

<table>
<thead>
<tr>
<th>Keys</th>
<th>K₁</th>
<th>K₂</th>
<th>...</th>
<th>Kₙ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freq</td>
<td>p₁</td>
<td>p₂</td>
<td></td>
<td>pₙ</td>
</tr>
<tr>
<td>Prob</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Goal: BST s.t.

Cost Search (Kᵢ) = Depth(Kᵢ) i.e. Depth of root = 1

Expected Cost = Σᵢ₌₁ⁿ pᵢ * Depth(Kᵢ)

BST with min expected cost.

Naive Alg: Try all possible trees

Side Question: How many trees?

We will use Dyn Prog.
**Step 1**  Give a definition of subproblems being solved

\[ T(n) = \# \text{BST with } n \text{ nodes} \]

**Step 2**  Give a recurrence over subproblems Include base cases.

\[ T(1) = 1 \quad T(2) = 2 \quad \text{or } T(0) = 1 \]

\[ T(n+1) = \sum_{i=0}^{n} T(i) \cdot T(n-i) \]

**Step 3**  Prove recurrence correct by induction.

\[ \text{eg Let } S_n = \text{set of all } n \text{ node BSTs.} \]

Partition \( S_n \) by \# nodes in left subtree.

\[ S^0_n, S^1_n, \ldots, S^{n-1}_n \]

Claim \( |S^i_n| = T(i) \cdot T(n-i) \) by induction
Step 4: Determine run time.

Subprob $T(1), \cdots, T(n)$

$C_i \equiv$ cost to compute $T(i)$ from $T(1), \cdots, T(i-1)$

$C_i = O(i)$

$\therefore$ Total Cost = $O(n^2)$
**Opt-BST**

**Inputs:** \( P_1, ..., P_n \)

**Trick 1** Compute expect cost (find tree later)

**Trick 2** Solve for more than asked

eg Solve \( C_{ij} = \text{expect cost for opt for } P_i, ..., P_j \) \( i \leq j \) (Step 1)

**Step 2** Recurrence relation

**Base Case** \( C_{ij} = P_i \) if \( i = j \) \( (C_{ij} = 0 \text{ if } i < j) \)

Define \( W_{ij} = P_i + ... + P_j \)

\[
C_{ij} = \min_{i \leq t \leq j} \{ C_{it-1} + C_{t+1:j} \} + W_{ij} \quad (*)
\]
Step 2: Correctness

Let $T = \text{opt tree subject to condition root } k_t$.

$T = T_{i+1} \cup T_{j+1}$

$T_L = T_{i+1}$

$T_R = T_{j+1}$

$\text{Cost}(T) = p_t + \sum_{l=1}^{t-1} p_l (\text{Depth}_T(k_l)) + \sum_{l=t+1}^{i} p_l (\text{Depth}_L(k_l))$

$= p_t + \sum_{l=t}^{i} p_l + \sum_{l=t+1}^{i} p_l \text{Depth}_T(k_l) + \sum_{l=t+1}^{j} p_l \text{Depth}_R(k_l)$

$W_{ij} + \text{Cost}(T_L) + \text{Cost}(T_R)$
Claim \( \text{Opt}_{ij} \leq C_{ij} \)

we could return a tree with \( \text{cost} = C_{ij} \)

the root will be \( K_4 \) for \( t \) min \((*)\)

Claim \( C_{ij} \leq \text{Opt}_{ij} \)

definition of \( i\)-1

Let \( T_{ij} \) be opt tree

Suppose root in \( K_4 \)

\( C_{ij} \leq C_{i+1} + C_{i+1} + W_{ij} \)

\( \leq \text{Opt}_{i+1} + \text{Opt}_{i+1} + W_{ij} \)

\( = \text{Cost}(T_{ij}) = \text{Opt}_{ij} \)
Recurrance as code (memorization)

Procedure C(i,j)
if k<j return 0
else if i=j return P_j
else if hash(i,j)≠0 return hash(i,j)
else
    hash(i,j) = \min_{i \leq t < j} \{ C(i, t-1) + C(t+1, j) \} + W_{ij}
return hash(i,j)

Timing
- Table size = O(n^2)
- Cost per entry = O(n)
- Total = O(n^3)
**Table Method**

\[ p = .2, .1, .7 \]

<table>
<thead>
<tr>
<th></th>
<th>1.4</th>
<th>.9</th>
<th>.7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.4</td>
<td>.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ .4 = .3 + \min \{ .1, .2, .3 \} \]
\[ .9 = .8 + \min \{ .1, .73 \} \]
\[ 1.4 = 1 + \min \{ .9 + 0, .2 + .7, 0 + .4 \} \]

---

Can we do better? Yes \( O(n^2) \) time

**Def** \( r(i, j) = \) index of root of an opt tree

**Claim** \[ r(i, j) \leq r(i', j) \leq r(t+1, j) \]