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 one.

Your solutions should have a special form.

4 Steps
Optimal Binary Search Trees

Input:

<table>
<thead>
<tr>
<th>Keys</th>
<th>K1</th>
<th>K2</th>
<th>\ldots</th>
<th>Kn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freq</td>
<td>P1</td>
<td>P2</td>
<td>\ldots</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) \leq Depth of root = 1

Expected Cost = \sum_{i=1}^{n} P_i \cdot \text{Depth}(K_i)

BST with min expected cost.

Naive Alg.: Try all possible trees

Side Question: How many trees (Binary) BT?

We will use Dyn Prog to compute # tree
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.

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

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

Step 3. Prove recurrence correct by induction.

Let \( S_n \) be the set of all \( n \) node BSTs.

Partition \( S_n \) by number of nodes in left subtree.

\( S_n^0, S_n^1, \ldots, S_n^{n-1} \)

Claim: \( |S_n^i| = T(i) \cdot T(n-i-1) \) by induction.
Step 4 Determine run time.

\text{E.g.} \text{ Sub-probs } T(1), \rightarrow T(n)

\[ C_i \equiv \text{cost to compute } T(i) \text{ from } T(1), \rightarrow T(i-1) \]

\[ c_i = O(i) \]

\[ \text{Total Cost} = \sum_{i=1}^{n} c_i = O(n^3) \]
Finding Opt-BST

Input: $P_1, \ldots, P_n$

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

**Trick 2** Solve for more than asked for

eg Solve $C_{ij} = \text{expected cost for opt for } P_i \ldots P_j$ $i \leq i$

(Step 1) Note: $\sum_{i=x}^{n} P_i < 1$

Step 2 Recurrence relation

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

Def $W_{ij} = P_i + \ldots + P_j$

$C_{ij} = \min\{C_{i, t-1} + C_{t+1, j}\} + W_{ij}$ $(*)$
Step 2: Correctness

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

$$T = \begin{cases} k_t \\ T_{l_{i+1}} \\ T_{r_{i+1}} \end{cases}$$

$$T_L = T_{i+1}$$

$$T_R = T_{r_{i+1}}$$

$$\text{Cost}(T) = P_k + \sum_{l=i}^{i+1} P_l (\text{Depth}_{T_L}(k_l)) + \sum_{l=i+1}^{j} P_l (\text{Depth}_{T_R}(k_l))$$

$$= P_k + \sum_{l \neq t} P_l + \sum_{l \neq t} \text{Depth}_{T_L}(k_l) + \sum_{l \neq t} \text{Depth}_{T_R}(k_l)$$

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

we could return a tree with cost \( = C_{ij} \)
the root will be \( K_t \) for \( t \in \text{min} \) \( x \)

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

by induction of \( j-i \)

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

Suppose root in \( K_{t_0} \)

\( C_{ij} \leq C_{i_{t-1}} + C_{t+1} + W_{ij} \)

\( \leq \text{Opt}_{i_{t-1}} + \text{Opt}_{t+1} + W_{ij} \)

\( = \text{Cost}(T_{ij}) = \text{Opt}_{ij} \)
Recurrence as code (memoization)

Procedure C(i,j)
if j < i return 0
elsif i = j return Pj
elsif 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

\[ p = 0.2, 0.1, 0.7 \]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
2 & 3 & 9 & 7 & 0 \\
1 & 4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[ 0.4 = 0.3 + \min \{ 0.1, 0.2, 3 \} \]
\[ 0.9 = 0.8 + \min \{ 0.1, 0.7, 3 \} \]
\[ 1.4 = 1 + \min \{ 0.9+0, 2+0.7, 0+0.4 \} \]

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

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

**Claim**: \( r(i, j-1) \leq r(i, j) \leq r(i+1, j) \)