Design and Analysis of Algorithms

CSE 5311

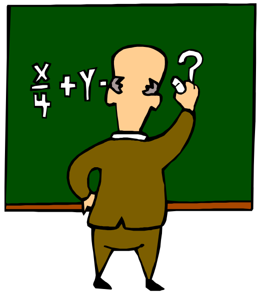
Lecture 15 Dynamic Programming

Junzhou Huang, Ph.D.

Department of Computer Science and Engineering

The General Dynamic Programming Technique

- Applies to a problem that at first seems to require a lot of time (possibly exponential), provided we have:
    - **Subproblem optimality:** the global optimum value can be defined in terms of optimal subproblems
    - **Subproblem overlap:** the subproblems are not independent, but instead they overlap (hence, should be constructed bottom-up).



Recalling: Steps in Dynamic Programming

1. Characterize structure of an optimal solution.
2. Define value of optimal solution recursively.
3. Compute optimal solution values either top-down with caching or bottom-up in a table.
4. Construct an optimal solution from computed values.

Optimal Binary Search Trees

- **Problem**
  - Given sequence $K = k_1 < k_2 < \cdots < k_n$ of n sorted keys, with a search probability p_i for each key k_i.
  - Want to build a binary search tree (BST) with minimum expected search cost.
  - Actual cost = # of items examined.
  - For key k_i, cost = $\text{depth}_T(k_i)+1$, where $\text{depth}_T(k_i)$ = depth of k_i in BST T .

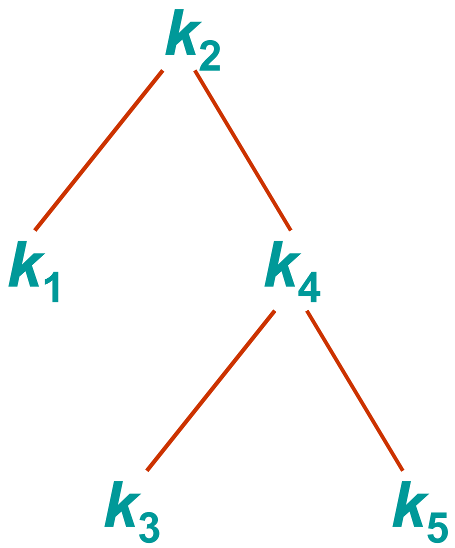
Expected Search Cost

$$
\begin{aligned}
&E[\text{search cost in } T] \\
&= \sum_{i=1}^{n} (\text{depth}_T(k_i) + 1) \cdot p_i \\
&= \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i + \sum_{i=1}^{n} p_i \\
&= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i \qquad \textbf{(15.16)}
\end{aligned}
$$

Sum of probabilities is 1.

Example

- Consider 5 keys with these search probabilities: $p_1 = 0.25, p_2 = 0.2, p_3 = 0.05, p_4 = 0.2, p_5 = 0.3.$

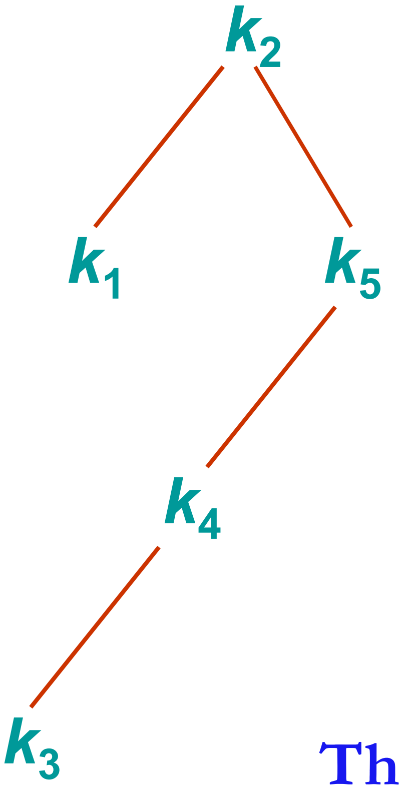


i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	0.25
2	0	0
3	2	0.1
4	1	0.2
5	2	0.6
		1.15

Therefore, E[search cost] = 2.15.

Example

- $p_1 = 0.25, p_2 = 0.2, p_3 = 0.05, p_4 = 0.2, p_5 = 0.3.$



i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	0.25
2	0	0
3	3	0.15
4	2	0.4
5	1	0.3
		1.10

Therefore, E[search cost] = 2.10.

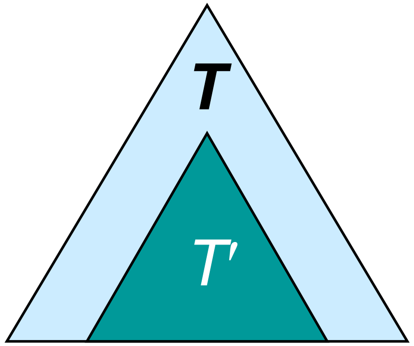
This tree turns out to be optimal for this set of keys.

Example

- **Observations:**
  - Optimal BST may not have smallest height.
  - Optimal BST may not have highest-probability key at root.
- Build by exhaustive checking?
  - Construct each n-node BST.
  - For each,
    assign keys and compute expected search cost.
  - But there are $\Omega(4^n/n^{3/2})$ different BSTs with n nodes.

Optimal Substructure

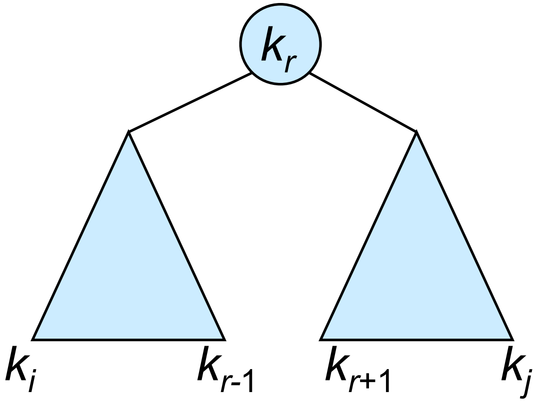
- Any subtree of a BST contains keys in a contiguous range k_i, ..., k_j for some $1 \leq i \leq j \leq n$.



- If T is an optimal BST and
  T contains subtree T' with keys k_i, ... ,k_j ,
  then T' must be an optimal BST for keys k_i, ..., k_j.

Optimal Substructure

- One of the keys in $k_i, \ldots, k_j$, say k_r, where $i \leq r \leq j$, must be the root of an optimal subtree for these keys.
- Left subtree of k_r contains $k_i, \ldots, k_{r-1}$.
- Right subtree of k_r contains $k_r+1, \ldots, k_j$.



- To find an optimal BST:
  - Examine all candidate roots k_r , for $i \leq r \leq j$
  - Determine all optimal BSTs containing $k_i, \ldots, k_{r-1}$ and containing $k_{r+1}, \ldots, k_j$

Recursive Solution

- Find optimal BST for $k_i, \ldots, k_j$, where $i \geq 1, j \leq n, j \geq i-1$. When $j = i-1$, the tree is empty.
- Define $e[i, j]$ = expected search cost of optimal BST for $k_i, \ldots, k_j$.

- If $j = i-1$, then $e[i, j] = 0$.
- If $j \geq i$,
  - Select a root k_r, for some $i \leq r \leq j$.
  - Recursively make an optimal BSTs
    - for $k_i, .., k_{r-1}$ as the left subtree, and
    - for $k_{r+1}, .., k_j$ as the right subtree.

Recursive Solution

- When the OPT subtree becomes a subtree of a node:
  - Depth of every node in OPT subtree goes up by 1.
  - Expected search cost increases by

$$\boxed{w(i, j) = \sum_{l=i}^{j} p_l}$$

from (15.16)

- If k_r is the root of an optimal BST for $k_i,..,k_j$:
  - $e[i, j] = p_r + (e[i, r-1] + w(i, r-1)) + (e[r+1, j] + w(r+1, j))$

$$= e[i, r-1] + e[r+1, j] + w(i, j).$$

(because $w(i, j)=w(i,r-1) + p_r + w(r + 1,$

- But, we don't know k_r. Hence,

$$\boxed{e[i, j] = \begin{cases} 0 & \text{if } j = i-1 \\ \min\limits_{i \le r \le j}\{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}}$$

Computing an Optimal Solution

For each subproblem (i,j), store:

- expected search cost in a table $e[1..n+1, 0..n]$
  - Will use only entries $e[i, j]$, where $j \geq i-1$.
- $\text{root}[i, j]$ = root of subtree with keys $k_i,..,k_j$, for $1 \leq i \leq j \leq n$.
- $w[1..n+1, 0..n]$ = sum of probabilities
  - $w[i, i-1] = 0$ for $1 \leq i \leq n$.
  - $w[i, j] = w[i, j\text{-}1] + p_j$ for $1 \leq i \leq j \leq n$.

Pseudo-code

```
OPTIMAL-BST($p, q, n$)
for $i \leftarrow 1$ to $n + 1$
    do $e[i, i-1] \leftarrow 0$
       $w[i, i-1] \leftarrow 0$
for $l \leftarrow 1$ to $n$
    do for $i \leftarrow 1$ to $n-l+1$
        do $j \leftarrow i + l-1$
           $e[i, j] \leftarrow \infty$
           $w[i, j] \leftarrow w[i, j-1] + p_j$
           for $r \leftarrow i$ to $j$
               do $t \leftarrow e[i, r-1] + e[r+1, j] + w[i, j]$
                  if $t < e[i, j]$
                      then $e[i, j] \leftarrow t$
                           $root[i, j] \leftarrow r$
return $e$ and $root$
```

Consider all trees with l keys.

Fix the first key.

Fix the last key

Determine the root of the optimal (sub)tree

Time: $O(n^3)$

Optimal Substructure

- Show that a solution to a problem consists of making a choice, which leaves one or more subproblems to solve.
- Suppose that you are given this last choice that leads to an optimal solution.
- Given this choice, determine which subproblems arise and how to characterize the resulting space of subproblems.
- Show that the solutions to the subproblems used within the optimal solution must themselves be optimal. Usually use cut-and-paste.
- Need to ensure that a wide enough range of choices and subproblems are considered.

Optimal Substructure

- Optimal substructure varies across problem domains:
  - 1. *How many subproblems* are used in an optimal solution.
  - 2. *How many choices* in determining which subproblem(s) to use.
- Informally, running time depends on (# of subproblems overall) × (# of choices).
- How many subproblems and choices do the examples considered contain?
- Dynamic programming uses optimal substructure **bottom up**.
  - *First* find optimal solutions to subproblems.
  - *Then* choose which to use in optimal solution to the problem.

Optimal Substucture

- Does optimal substructure apply to all optimization problems? No.
- Applies to determining the shortest path but NOT the longest simple path of an unweighted directed graph.
- Why?
  - Shortest path has independent subproblems.
  - Solution to one subproblem does not affect solution to another subproblem of the same problem.
  - Subproblems are not independent in longest simple path.
    - Solution to one subproblem affects the solutions to other subproblems

Overlapping Subproblems

- The space of subproblems must be “small”.
- The total number of distinct subproblems is a polynomial in the input size.
  - A recursive algorithm is exponential because it solves the same problems repeatedly.
  - If divide-and-conquer is applicable, then each problem solved will be brand new.