

Advanced Algorithms Analysis and Design

By

Nazir Ahmad Zafar

Lecture No 23

Longest Common Subsequence (Dynamic Algorithm)

Optimal Binary Search Trees

Theorem: Optimal Substructure of an LCS

- If $X = (x_1, x_2, \dots, x_m)$, and $Y = (y_1, y_2, \dots, y_n)$ be sequences and let us suppose that $Z = (z_1, z_2, \dots, z_k)$ be a longest common sub-sequence of X and Y
 1. if $x_m = y_n$, then $z_k = x_m$ and Z_{k-1} is LCS of X_{m-1}, Y_{n-1} .
 2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is LCS of X_{m-1} and Y
 3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$c(i, j) = \begin{cases} 0 & \text{if } i = 0 \text{ OR } j = 0 \\ c(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(c(i-1, j), c(i, j-1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Example

Problem

If $X = \langle A, B, C, B, D, A, B \rangle$, and $Y = \langle B, D, C, A, B, A \rangle$ are two sequences then compute a maximum-length common subsequence of X and Y .

Solution:

- Let $c(i, j) = \text{length of LCS of } X_i \text{ and } Y_j$, now we have to compute $c(7, 6)$.
- The recursive mathematical formula computing LCS is given below

$$c(i, j) = \begin{cases} 0 & \text{if } i = 0 \text{ OR } j = 0 \\ c(i - 1, j - 1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(c(i - 1, j), c(i, j - 1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Example

If $X = \langle A, B, C, B, D, A, B \rangle$, $Y = \langle B, D, C, A, B, A \rangle$

- $c(1, 1) = \max(c(0, 1), c(1, 0)) = \max(0, 0) = 0$
 $b[1, 1] = \leftarrow$
- $c(1, 2) = \max(c(0, 2), c(1, 1)) = \max(0, 0) = 0$
 $b[1, 2] = \leftarrow$
- $c(1, 3) = \max(c(0, 3), c(1, 2)) = \max(0, 0) = 0$
 $b[1, 3] = \leftarrow$
- $c(1, 4) = c(0, 3) + 1 = 0 + 1 = 1$; $b[1, 4] = \nwarrow$
- $c(1, 5) = \max(c(0, 5), c(1, 4)) = \max(0, 1) = 1$
 $b[1, 5] = \leftarrow$
- $c(1, 6) = c(0, 5) + 1 = 0 + 1 = 1$; $b[1, 6] = \square$

Example

If $X = \langle A, B, C, B, D, A, B \rangle$, $Y = \langle B, D, C, A, B, A \rangle$

- $c(2, 1) = c(1, 0) + 1 = 0 + 1 = 1$; $b[2, 1] = \square$
- $c(2, 2) = \max(c(1, 2), c(2, 1)) = \max(0, 1) = 1$
 $b[2, 2] = \leftarrow$
- $c(2, 3) = \max(c(1, 3), c(2, 2)) = \max(0, 1) = 1$
 $b[2, 3] = \leftarrow$
- $c(2, 4) = \max(c(1, 4), c(2, 3)) = \max(1, 1) = 1$
 $b[2, 4] = \leftarrow$
- $c(2, 5) = c(1, 4) + 1 = 1 + 1 = 2$; $b[2, 5] = \square$
- $c(2, 6) = \max(c(1, 6), c(2, 5)) = \max(1, 2) = 2$
 $b[2, 6] = \leftarrow$

Example

If $X = \langle A, B, C, B, D, A, B \rangle$, $Y = \langle B, D, C, A, B, A \rangle$

- $c(3, 1) = \max(c(2, 1), c(3, 0)) = \max(1, 0) = 1$
 $b[3, 1] = \leftarrow$
- $c(3, 2) = \max(c(2, 2), c(3, 1)) = \max(1, 1) = 1$
 $b[3, 2] = \leftarrow$
- $c(3, 3) = c(2, 2) + 1 = 1 + 1 = 2$; $b[3, 3] = \square$
- $c(3, 4) = \max(c(2, 4), c(3, 3)) = \max(1, 2) = 2$
 $b[3, 4] = \leftarrow$
- $c(3, 5) = \max(c(2, 5), c(3, 4)) = \max(2, 2) = 2$
 $b[3, 5] = \leftarrow$
- $c(3, 6) = \max(c(2, 6), c(3, 5)) = \max(2, 2) = 2$
 $b[2, 6] = \leftarrow$

Example

If $X = \langle A, B, C, B, D, A, B \rangle$, $Y = \langle B, D, C, A, B, A \rangle$

- $c(4, 1) = c(3, 0) + 1 = 0 + 1 = 1$; $b[4, 1] = \square$
- $c(4, 2) = \max(c(3, 2), c(4, 1)) = \max(1, 1) = 1$
 $b[4, 2] = \leftarrow$
- $c(4, 3) = \max(c(3, 3), c(4, 2)) = \max(2, 1) = 2$
 $b[4, 3] = \uparrow$
- $c(4, 4) = \max(c(3, 4), c(4, 3)) = \max(2, 2) = 2$
 $b[4, 4] = \leftarrow$
- $c(4, 5) = c(3, 4) + 1 = 2 + 1 = 3$; $b[4, 5] = \square$
- $c(4, 6) = \max(c(3, 6), c(4, 5)) = \max(2, 3) = 3$
 $b[4, 6] = \leftarrow$

Example

If $X = \langle A, B, C, B, D, A, B \rangle$, $Y = \langle B, D, C, A, B, A \rangle$

- $c(5, 1) = \max(c(4, 1), c(5, 0)) = \max(1, 0) = 1$
 $b[5, 1] = \uparrow$
- $c(5, 2) = c(4, 1) + 1 = 1 + 1 = 2$; $b[5, 2] = \square$
- $c(5, 3) = \max(c(4, 3), c(5, 2)) = \max(2, 2) = 2$
 $b[5, 3] = \leftarrow$
- $c(5, 4) = \max(c(4, 4), c(5, 3)) = \max(2, 2) = 2$
 $b[5, 4] = \leftarrow$
- $c(5, 5) = \max(c(4, 5), c(5, 4)) = \max(3, 2) = 3$
 $b[5, 5] = \uparrow$
- $c(5, 6) = \max(c(4, 6), c(5, 5)) = \max(3, 3) = 3$

Example

If $X = \langle A, B, C, B, D, A, B \rangle$, $Y = \langle B, D, C, A, B, A \rangle$

- $c(6, 1) = \max(c(5, 1), c(6, 0)) = \max(1, 0) = 1$
 $b[6, 1] = \uparrow$
- $c(6, 2) = \max(c(5, 2), c(6, 1)) = \max(2, 1) = 2$
 $b[6, 1] = \uparrow$
- $c(6, 3) = \max(c(5, 3), c(6, 2)) = \max(2, 2) = 2$
 $b[6, 3] = \leftarrow$
- $c(6, 4) = c(5, 3) + 1 = 2 + 1 = 3$; $b[6, 4] = \square$
- $c(6, 5) = \max(c(5, 5), c(6, 4)) = \max(2, 3) = 3$
 $b[6, 5] = \leftarrow$
- $c(6, 6) = c(5, 5) + 1 = 3 + 1 = 4$; $b[6, 6] = \square$

Example

If $X = \langle A, B, C, B, D, A, B \rangle$, $Y = \langle B, D, C, A, B, A \rangle$

- $c(7, 1) = c(6, 0) + 1 = 0 + 1 = 1$; $b[7, 1] = \square$
- $c(7, 2) = \max(c(6, 2), c(7, 1)) = \max(2, 1) = 2$
 $b[7, 2] = \uparrow$
- $c(7, 3) = \max(c(6, 3), c(7, 2)) = \max(2, 2) = 2$
 $b[7, 3] = \leftarrow$
- $c(7, 4) = \max(c(6, 4), c(7, 3)) = \max(3, 2) = 3$
 $b[7, 4] = \uparrow$
- $c(7, 5) = c(6, 4) + 1 = 3 + 1 = 4$; $b[7, 5] = \square$
- $c(7, 6) = \max(c(6, 6), c(7, 5)) = \max(4, 4) = 4$
 $b[7, 6] = \leftarrow$

Results

j	0	1	2	3	4	5	6
i	y_j	B	D	C	A	B	A
0 x_i	0	0	0	0	0	0	0
1 A	0	0	0	0	1	1	1
2 B	0	1	1	1	1	2	2
3 C	0	1	1	2	2	2	2
4 B	0	1	1	2	2	3	3
5 D	0	1	2	2	2	3	3
6 A	0	1	2	2	3	3	4
7 B	0	1	2	2	3	4	4

$$c(i, j) = \begin{cases} 0 & \text{if } i = 0 \text{ OR } j = 0 \\ c(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(c(i-1, j), c(i, j-1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

j	0	1	2	3	4	5	6
i	y_j	B	D	C	A	B	A
0 x_i	•	•	•	•	•	•	•
1 A	•	←	←	←	↖	←	↑
2 B	•	↖	←	←	←	↖	←
3 C	•	↑	←	↖	←	←	←
4 B	•	↖	←	↑	←	↖	←
5 D	•	↑	↖	←	←	↑	←
6 A	•	↑	↑	←	↖	←	↑
7 B	•	↖	↑	←	↑	↖	←

if $i = 0$ OR $j = 0$

if $i, j > 0$ and $x_i = y_j$

if $i, j > 0$ and $x_i \neq y_j$

Computable Tables

<i>i</i>	<i>j</i>	0	1	2	3	4	5	6
<i>i</i>	<i>y_j</i>	B	D	C	A	B	A	
0	<i>x_i</i>	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

$$c(i, j) = \begin{cases} 0 & \text{if } i = 0 \text{ OR } j = 0 \\ c(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(c(i-1, j), c(i, j-1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

<i>i</i>	<i>j</i>	0	1	2	3	4	5	6
<i>i</i>	<i>y_j</i>	B	D	C	A	B	A	
0	<i>x_i</i>	•	•	•	•	•	•	•
1	A	•	←	←	←	↖	←	↖
2	B	•	↖	←	←	←	↖	←
3	C	•	↑	←	↖	←	←	←
4	B	•	□	←	↑	←	↖	←
5	D	•	↑	↖	↖	←	↑	←
6	A	•	↑	↑	←	↖	←	↖
7	B	•	□	↑	←	↑	□	↖

if $i = 0$ OR $j = 0$

if $i, j > 0$ and $x_i = y_j$

if $i, j > 0$ and $x_i \neq y_j$

Computable Tables

<i>i</i>	<i>j</i>	0	1	2	3	4	5	6
<i>i</i>	<i>y_j</i>	B	D	C	A	B	A	
0	<i>x_i</i>	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

$$c(i, j) = \begin{cases} 0 & \text{if } i=0 \text{ OR } j=0 \\ c(i-1, j-1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(c(i-1, j), c(i, j-1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

<i>i</i>	<i>j</i>	0	1	2	3	4	5	6
<i>i</i>	<i>y_j</i>	B	D	C	A	B	A	
0	<i>x_i</i>	•	•	•	•	•	•	•
1	A	•	←	←	←	↖	←	↗
2	B	•	↖	←	←	←	↖	←
3	C	•	↑	←	↗	←	←	←
4	B	•	↖	←	↑	←	↖	←
5	D	•	↑	↖	←	←	↑	←
6	A	•	↑	↑	←	↖	←	↖
7	B	•	↖	↑	←	↑	↖	←

if $i=0$ OR $j=0$

if $i, j > 0$ and $x_i = y_j$

if $i, j > 0$ and $x_i \neq y_j$

Computable Tables

i	j	0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

- Table size: $O(n.m)$
- Every entry takes $O(1)$ time to compute.
- The algorithm takes $O(n.m)$ time and space.
- The space complexity can be reduced to $2 \cdot \min(m, n) + O(1)$.

Longest Common Subsequence Algorithm

$$\begin{aligned} c[i, j] &= c(i-1, j-1) + 1 && \text{if } x_i = y_j; \\ c[i, j] &= \max(c(i-1, j), c(i, j-1)) && \text{if } x_i \neq y_j; \\ c[i, j] &= 0 && \text{if } (i = 0) \text{ or } (j = 0). \end{aligned}$$

function LCS(X, Y)

- 1 m \leftarrow length [X]
- 2 n \leftarrow length [Y]
- 3 **for** i \leftarrow 1 **to** m
- 4 **do** c[i, 0] \leftarrow 0;
- 5 **for** j \leftarrow 1 **to** n
- 6 **do** c[0, j] \leftarrow 0;

Longest Common Subsequence Algorithm

$$\begin{aligned} c[i, j] &= c(i-1, j-1) + 1 && \text{if } x_i = y_j; \\ c[i, j] &= \max(c(i-1, j), c(i, j-1)) && \text{if } x_i \neq y_j; \\ c[i, j] &= 0 && \text{if } (i = 0) \text{ or } (j = 0). \end{aligned}$$

```
7. for i ← 1 to m
8     do for j ← 1 to n
9         do if (xi == yj)
10            then c[i, j] ← c[i-1, j-1] + 1
11            b[i, j] ← “↖”
12        else if c[i-1, j] ≥ c[i, j-1]
13            then c[i, j] ← c[i-1, j]
14            b[i, j] ← “↑”
15        else c[i, j] ← c[i, j-1]
16            b[i, j] ← “←”
17 Return c and b;
```

Construction of Longest Common Subsequence

$$\begin{aligned} c[i, j] &= c(i-1, j-1) && \text{if } x_i = y_j; \\ c[i, j] &= \min(c(i-1, j), c(i, j-1)) && \text{if } x_i \neq y_j; \\ c[i, j] &= 0 && \text{if } (i = 0) \text{ or } (j = 0). \end{aligned}$$

procedure PrintLCS(b, X, i, j)

1. **if** ($i == 0$) or ($j == 0$)
2. **then return**
3. **if** $b[i, j] == “\nwarrow”$
4. **then** PrintLCS(b, X, $i-1, j-1$)
5. Print x_i
6. **else if** $b[i, j] \leftarrow “\uparrow”$
7. **then** PrintLCS(b, X, $i-1, j$)
8. **else** PrintLCS(b, X, $i, j-1$)

Relationship with shortest common super-sequence

- Shortest common super-sequence problem is closely related to longest common subsequence problem

Shortest common super-sequence

- Given two sequences:

$X = < x_1, \dots, x_m >$ and

$Y = < y_1, \dots, y_n >$

- A sequence $U = < u_1, \dots, u_k >$ is a common super-sequence of X and Y if U is a super-sequence of both X and Y .
- The shortest common supersequence (scs) is a common supersequence of minimal length.

Relationship with shortest common super-sequence

Problem Statement

- The two sequences X and Y are given and task is to find a shortest possible common supersequence.
- Shortest common supersequence is not unique.
- Easy to make SCS from LCS for 2 input sequences.

Example,

- $X[1..m] = \text{abcbdab}$
 $Y[1..n] = \text{bdcaba}$
 $\text{LCS} = Z[1..r] = \text{bcba}$
- Insert non-lcs symbols preserving order, we get
 $\text{SCS} = U[1..t] = \text{abdcabdab.}$

Optimal Binary Search Trees

Binary Search Trees

- Binary search tree (BST) is a binary data structure which has the following properties:
 - Each node has a value.
 - An order is defined on these values.
 - Left sub-tree of node contains values less than node value
 - Right sub-tree of a node contains only values greater than or equal to the node's value.

Optimal Binary Search Trees

Example: A translator from English to, say, Urdu.

- Use a binary search tree to store all the words in our dictionary, together with their translations.
- The word “the” is much more likely to be looked up than the word “ring”
- So we would like to make the search time for the word “the” very short, possibly at the expense of increasing the search time for the word “ring.”

Problem Statement: We are given a probability distribution that determines, for every key in the tree, the likelihood that we search for this key.

- The objective is to minimize the expected search time of the tree.

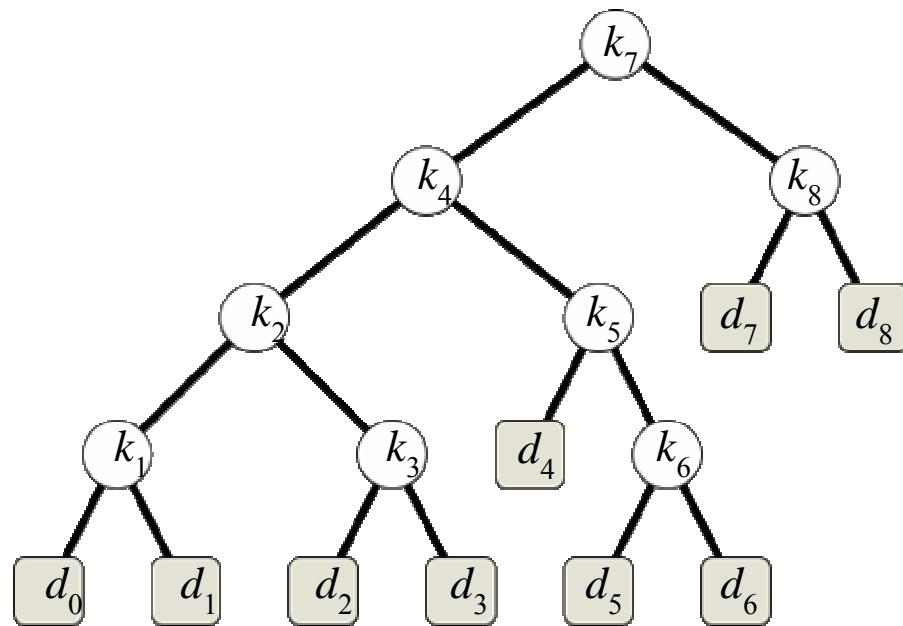
Optimal Binary Search Trees

- We need is known as an optimal binary search tree.
- Formally, we are given a sequence $K = \langle k_1, k_2, \dots, k_n \rangle$ of n distinct keys in sorted order i.e. $k_1 < k_2 < \dots < k_n$, and we wish to build a binary search tree from these keys.
- For each k_i , we have a probability p_i that search is for k_i .
- Some searches may not be successful, so we have $n + 1$ “dummy keys” $d_0, d_1, d_2, \dots, d_n$ representing values not in K
- In particular
 - d_0 = represents all values less than k_1 ,
 - d_n = represents all values greater than k_n , and
 - d_i = represents all values between k_i and k_{i+1} ,
 $\forall i = 1, 2, \dots, n - 1$

Optimal Binary Search Trees

- For each dummy key d_i , we have a probability q_i that a search will correspond to d_i .
- Each k_i is an internal node, each dummy key d_i is a leaf.
- Every search is either successful (finding some key k_i) or failure (finding some dummy key d_i), and so we have

$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1$$



Total Cost: Optimal Binary Search Trees

Total cost =

$$\begin{aligned} & \sum_{i=1}^n p_i (\text{depth}(k_i) + 1) + \sum_{i=0}^n q_i (\text{depth}(d_i) + 1) \\ &= \sum_{i=1}^n p_i \cdot \text{depth}(k_i) + \sum_{i=1}^n p_i + \sum_{i=0}^n q_i \text{depth}(d_i) + \sum_{i=0}^n q_i \end{aligned}$$

Since we know that : $\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1$

Hence Total Cost

$$= \sum_{i=1}^n p_i \cdot \text{depth}(k_i) + \sum_{i=0}^n q_i \text{depth}(d_i) + 1$$

Example

Let

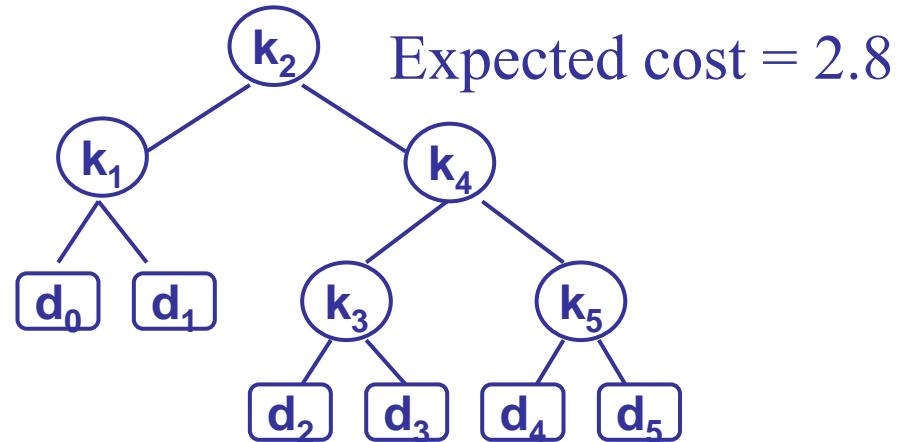
p_i = probability of searching k_i

q_i = probability representing d_i

i	0	1	2	3	4	5
p_i	0.15	0.10	0.05	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

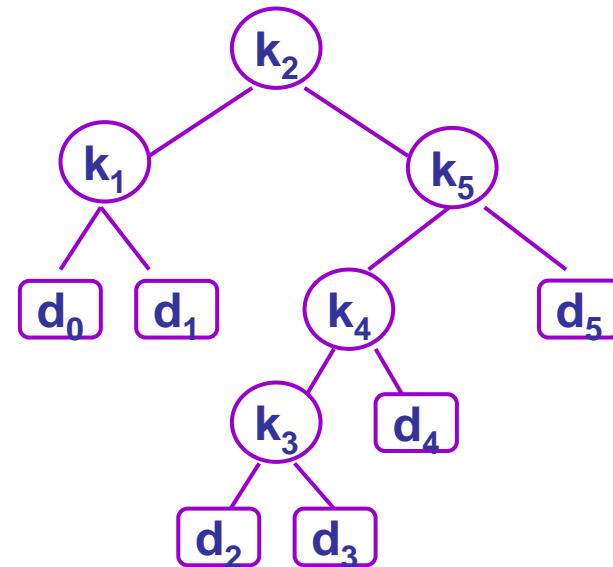
Define cost of a search is as number of nodes examined in a search

Cost of a key = depth of the key + 1



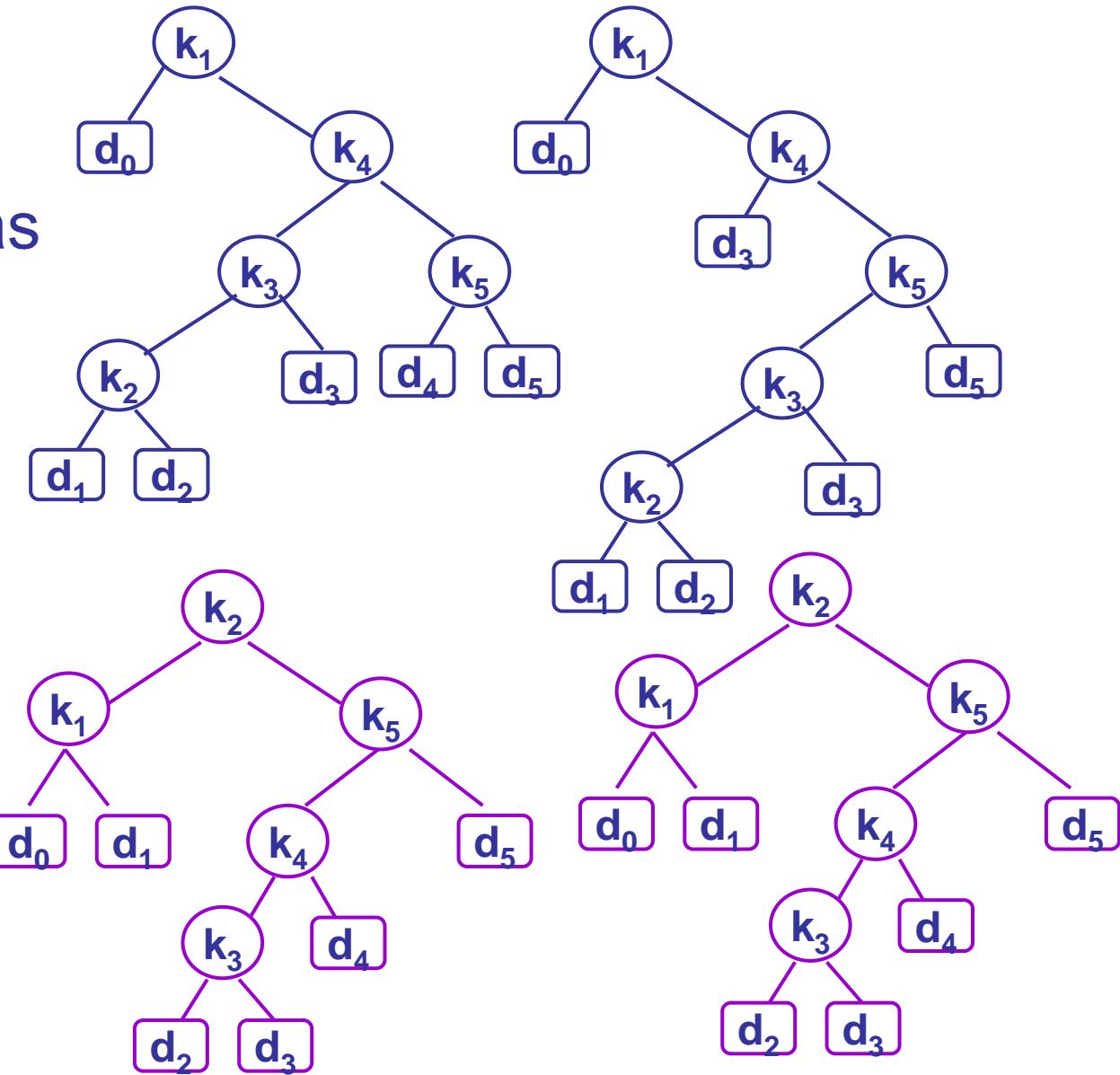
Expected cost = 2.8

Expected cost = 2.75



Brute Force Solution

- Total number of binary trees will be exponential as in case of chain matrix problem
- Brute force approach is not economical



Brute Force Solution

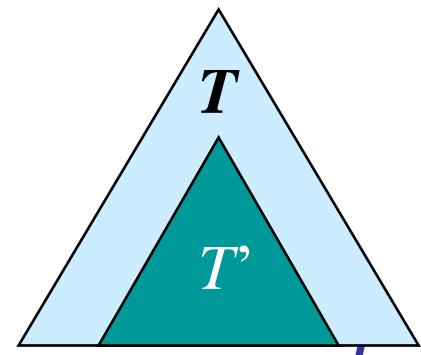
- **Observations:**
 - Optimal BST may not have smallest height.
 - Optimal BST may not have highest-probability key at root.
- Build by exhaustive checking?
 - Construct each n -node BST.
 - For each, assign keys and compute expected search cost.
 - But there are $\Omega(4^n/n^{3/2})$ different BSTs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

Constructing Dynamic Programming

Optimal Substructure

Observation:

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



Lemma

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

Proof:

- Cut and paste method.

Limitations of Dynamic Programming

- Dynamic programming can be applied to any problem that observes the principle of optimality.
- Generally, it means that partial solutions can be optimally extended with regard to the state after the partial solution instead of the partial solution itself.
- The biggest limitation using dynamic programming is number of partial solutions we must keep track of
- For all examples we have seen, partial solutions can be described by stopping places in the input.
- This is because combinatorial objects e.g. strings, numerical sequences, and polygons etc., all have an implicit order defined upon their elements.

Limitations of Dynamic Programming

- This order cannot be changed without completely changing the original problem.
- Once order fixed, there are relatively few possible stopping places, and we get an efficient algorithms.
- If objects are not firmly ordered then we have an exponential number of possible partial solutions
- And we get an infeasible amount of memory resulting an infeasible solution.