Optimal Binary Search Trees

- We are given a sequence $K = <k_1, k_2, \ldots k_n>$ of n distinct keys in sorted order so that $k_1 < k_2 < \ldots k_n$. We wish to build a BST from these keys.

- For each key k_i we have a probability p_i that a search will be for k_i. Some searches may be for values not in K, and so we have $n + 1$ "dummy keys" $d_0, d_1, d_2, \ldots d_n$ 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 for $i = 1, 2, \ldots n - 1$, the dummy key d_i represents all values between k_i and k_{i+1}.

- For each dummy key d_i, we have a probability q_i that a search will correspond to d_i. (Refer to Slide #13). Each key k_i is an internal node, and each dummy key d_i is a leaf.

- Every search is either successful (finding some key k_i) or unsuccessful (finding some dummy key d_i) and so we have $(p_1 + p_2 + \ldots + p_n) + (q_0 + q_1 + q_2 + \ldots q_n) = 1$

- Because we have probabilities of searches for each key and each dummy key, we can determine the expected cost of a search in a given binary search tree T.

- Let us assume that the actual cost of a search is the number of nodes examined, i.e., the depth of the node found by the search in T, plus 1. Then the expected cost of a search in T is

$$\text{E[search cost in } T] = \sum_{i=1}^{n} (\text{depth}_T(k_i) + 1) * p_i + \sum_{i=0}^{n} (\text{depth}_T(d_i) + 1) * q_i$$

$$\begin{aligned} &= (p_1 + p_2 + \ldots + p_n) + \sum_{i=1}^{n} \text{depth}_T(k_i) * p_i + \\ &\quad (q_0 + q_1 + q_2 + \ldots q_n) + \sum_{i=0}^{n} \text{depth}_T(d_i) * q_i \qquad (1) \\ &= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) * p_i + \sum_{i=0}^{n} \text{depth}_T(d_i) * q_i \end{aligned}$$

where depth_T denotes a node's depth in the tree T.

- For a given set of probabilities, our goal is to construct a BST whose expected search cost is smallest. We call such a tree an ***optimal binary search tree***.

- As with matrix –chain multiplication, exhaustive search of all possibilities fails to yield an efficient algorithm. This is because, the number of binary trees with n nodes is $\Omega(4^n / n^{3/2})$ and so there are exponential number of binary search trees that we would have to examine in an exhaustive search. So we will solve this problem with dynamic programming.

- Consider any subtree of a BST. It must contain the keys in contiguous order $k_i, \ldots k_j$ for some $1 \le i \le j \le n$. In addition, a subtree that contains keys $k_i, \ldots k_j$ must also have as its leaves the dummy keys $d_{i-1}, \ldots d_j$

- Here is the optima substructure: if an optimal BST has a subtree T' containing keys $k_i, \ldots k_j$, then this subtree T' **must** be optimal as well for the subproblem with keys $k_i, \ldots k_j$ and dummy keys $d_{i-1}, \ldots d_j$.

- We need to use the optimal substructure to show that we can construct an optimal solution to the problem from optimal solutions to subproblems.

- Given keys $k_i, \ldots k_j$, one of these keys say k_r ($i \le r \le j$), will be the root of an optimal subtree containing these keys.
    - The left subtree of the root k_r will contain the keys $k_i, k_{i+1}, \ldots k_{r-1}$ (and dummy keys $d_{i-1}, d_i, \ldots d_{r-1}$)
    - The right subtree will contain the keys $k_{r+1}, \ldots k_j$ (and dummy keys $d_r, d_{r+1} \ldots d_j$).

- As long as we examine all the candidate roots k_r where $i \le r \le j$, and we determine all optimal binary search trees containing $k_i, \ldots k_{r-1}$ and those containing $k_{r+1}, \ldots k_j$, we are guaranteed that we will find an optimal BST.

- **Now we are ready to define the value of an optimal solution recursively**. We pick our subproblem domain as finding the optimal BST containing the keys $k_i, \ldots k_j$ where $i \ge 1$, $j \le n$, and $j \ge i - 1$. Note that when $j = i - 1$, there are no actual keys; we have just the dummy key d_{i-1}

- Define $e[i, j]$ as the expected cost of searching an optimal BST containing the keys $k_i, \ldots k_j$.

- We wish to compute $e[1,n]$

    - The easy case occurs when $j = i - 1$. Then we have just the dummy key d_{i-1} and the expected cost is $e[i, i-1] = q_{i-1}$

    - When $j \ge i$, we need to select a root k_r, from among $k_i, \ldots k_j$, and then make an optimal BST with keys $k_i, \ldots k_{r-1}$ as its left subtree and an optimal BST with keys $k_{r+1}, \ldots k_j$ as its right subtree.

    - What is the expected search cost of a subtree when it becomes a subtree of a node? **The depth of each node in the subtree increases by 1.**

    - By (1), the expected search cost of this subtree increases by sum of all probabilities in the subtree.

- For a subtree with keys $k_i, \ldots k_j$, let us denote this sum of probabilities as:

$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

- Thus if k_r is the root of an optimal subtree containing the keys $k_i, \ldots k_j$, we have

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

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

- So we can rewrite $e[i, j]$ as

$$e[i, j] = e[i, r-1] + e[r+1, j] + w(i, j) \qquad (2)$$

- (2) assumes that we know which node k_r to use as the root. We choose the root that gives the lowest expected search cost, giving us our final recursive formulation:

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

<u>Example:</u> Construct an optimal BST with keys k_1, k_2, k_3, k_4 and k_5 and $k_1 < k_2 < k_3 < k_4 < k_5$ with the following probabilities:
$(p_1, p_2, p_3, p_4, p_5) = (0.15, 0.10, 0.05, 0.10, 0.20)$

$(q_0, q_1, q_2, q_3, q_4, q_5) = (0.05, 0.10, 0.05, 0.05, 0.05, 0.10)$

Table with values for *w*

$i \rightarrow$

$j \downarrow$	1	2	3	4	5	6
5						
4						
3						
2						
1						
0						

Table with values for *e* $i \rightarrow$

$i \downarrow$	1	2	3	4	5	6
5						
4						
3						
2						
1						
0						

Root table

$i \rightarrow$

$i \downarrow$	1	2	3	4	5
5					
4					
3					
2					
1					