Optimal Binary Search Trees

We are given a sequence $K = \langle k_1, k_2, \ldots, k_n \rangle$ of n distinct keys in sorted order ($k_1 < k_2 < \cdots < k_n$). For each key k_i, we have a probability p_i that a search will be for k_i.

$d_0, k_1, d_1, k_2, d_2, k_3, \ldots, k_{n-1}, d_{n-1}, k_n, d_n$.

We have $n+1$ dummy keys $d_0, d_1, \ldots, d_n$ representing values not in K. d_0 represents all values less than k_1, d_n represents all values greater than k_n, and for $i = 1, 2, \ldots, n-1$, 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.

Every search is either successful (finding some key k_i) or unsuccessful (finding some dummy key d_i) $\Rightarrow$

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

For a given set of probabilities, we wish to construct a Binary Search Tree whose expected search cost is smallest. We call such a tree an Optimal Binary Search Tree.

The expected cost of a search in a BST T is given by:

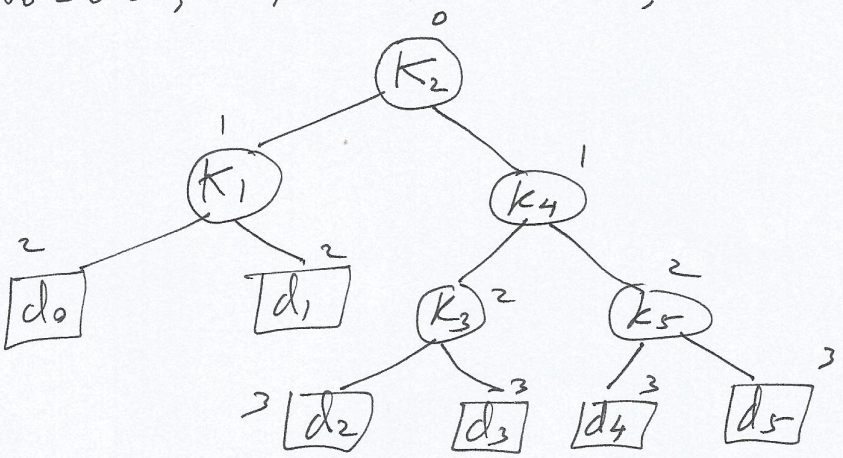
$$E[\text{search cost in } T] = \sum_{i=1}^{n} (\text{depth}_T(k_i)+1) \cdot p_i + \sum_{i=0}^{n} (\text{depth}_T(d_i)+1) q_i$$

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

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

Example : $p_1 = 0.15,\ p_2 = 0.10,\ p_3 = 0.05,\ p_4 = 0.10,\ p_5 = 0.20$

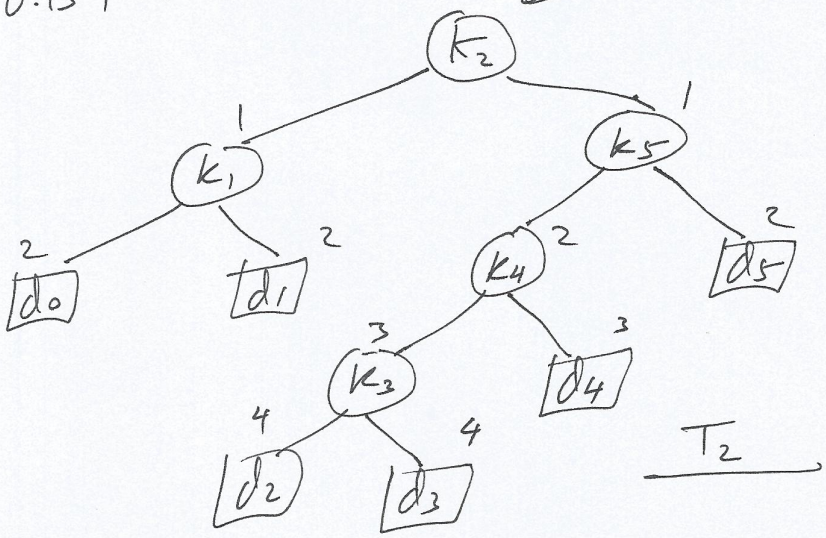
$q_0 = 0.05,\ q_1 = 0.10,\ q_2 = 0.05,\ q_3 = 0.05,\ q_4 = 0.05,\ q_5 = 0.10$



T_1

$$E[T_1] = 1 + 0\times 0.10 + 1\times 0.15 + 1\times 0.10 + 2\times 0.05 + 2\times 0.20 + 2\times 0.05 + 2\times 0.10 + 3\times 0.05 + 3\times 0.05 + 3\times 0.05 + 3\times 0.10$$

$$= 1 + 0.15 + 0.10 + 0.10 + 0.40 + 0.10 + 0.20 + 0.15 + 0.15 + 0.15 + 0.30 = \boxed{2.80}$$



T_2

$$E[T_2] = 1 + 0\times 0.10 + 1\times 0.15 + 1\times 0.20 + 2\times 0.10 + 3\times 0.05 + 2\times 0.05 + 2\times 0.10 + 2\times 0.10 + 3\times 0.05 + 4\times 0.05 + 4\times 0.05$$

$$= 1 + 0.15 + 0.20 + 0.20 + 0.15 + 0.10 + 0.20 + 0.20 + 0.15 + 0.20 + 0.20$$

$$= \boxed{2.75}$$

Step 1: optimal Substructure: Consider any non-leaf rooted subtree of a BST. It must contain keys in a contiguous range $k_i, \ldots, k_j$, 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 $d_{i-1}, \ldots d_j$. We also have leaf-rooted subtrees: $d_0, d_1, \ldots, d_n$. If an optimal BST T 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 $d_{i-1}, \ldots, d_j$.

Step 2: Recursive Solution: We define $e[i,j]$ as the expected cost of searching an optimal BST containing the keys $k_i, \ldots k_j$ for $1 \le i \le j \le n$. Our objective is to compute $e[1,n]$.

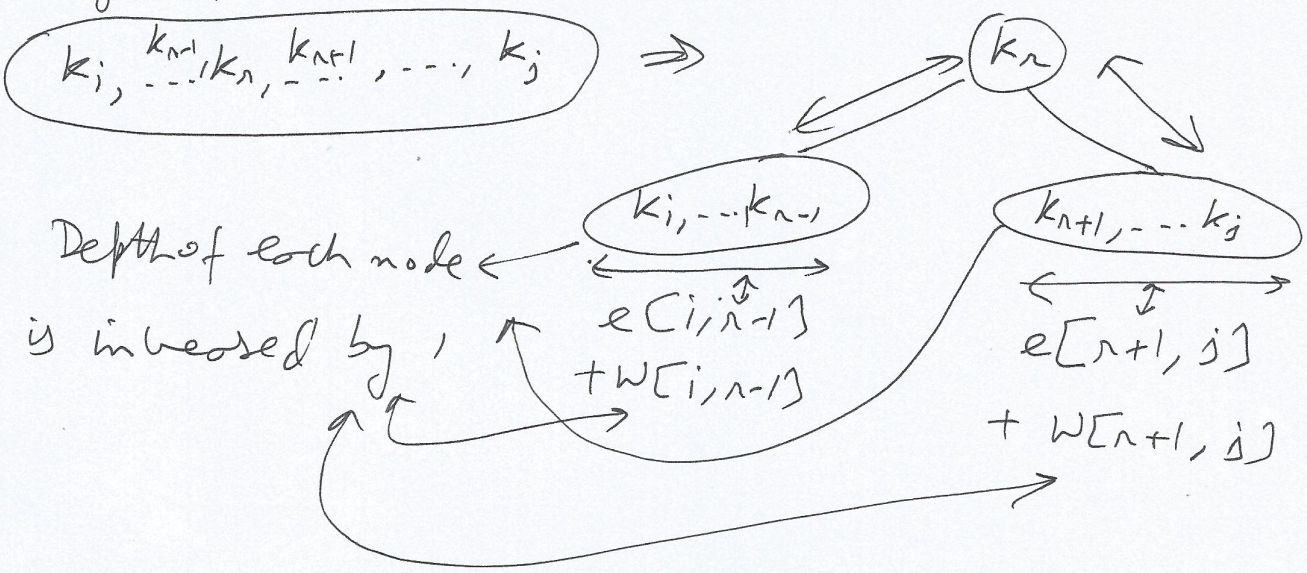
For $1 \le i \le n+1$, we define $e[i, i-1] = q_{i-1} =$ cost of the BST containing only the dummy key d_{i-1} (the smallest subproblem).

Similar to the definition of $e[i,j]$, we define $w[i,j]$ as the sum of probabilities of all nodes in the BST having $k_i, \ldots, k_j$, and $d_{i-1}, \ldots d_j$: for $1 \le i \le j \le n$

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

and for $1 \le i \le n+1$, we define $w[i, i-1] = q_{i-1}$.

If k_r is the root of an optimal subtree containing keys $k_i, \ldots k_j$:



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

we also have: $w[i,j] = w[i,r-1] + p_r + w[r+1,j]$

$$\Rightarrow e[i,j] = e[i,r-1] + e[r+1,j] + w[i,j]$$

We have a choice for node k_r. We will choose the best possible node k_r:

$$e[i,j] = q_{i-1} \quad \text{if } j = i-1$$

$$e[i,j] = \min_{i \le r \le j} \{ e[i,r-1] + e[r+1,j] + w[i,j] \} \text{ if } i \le j$$

Step 3 : Computing the expected search cost of an optimal BST :

```
Optimal-BST(p, q, n)
1   let e[1..n+1, 0..n], w[1..n+1, 0..n], root[1..n, 1..n]
    be new tables
2   for i = 1 to n+1
3       e[i, i-1] ← q_{i-1}
4       w[i, i-1] ← q_{i-1}
5   for l = 1 to n
6       for i = 1 to n-l+1
7           j ← i+l-1
8           e[i,j] ← ∞
9           w[i,j] ← w[i,j-1] + p_j + q_j
10          for r = i to j
11              t ← e[i,r-1] + e[r+1,j] + w[i,j]
12              if t < e[i,j]
13                  e[i,j] ← t
14                  root[i,j] ← r
15  return e and root
```

Time complexity of optimal-BST is $\Theta(n^3)$

Space complexity of optimal-BST is $\Theta(n^2)$

Step 4 : Making the optimal BST from root table :

$k_1, k_2 \ldots, k_{root[1,n]-1}, k_{root[1,n]}, k_{root[1,n]+1}, \ldots, k_n$

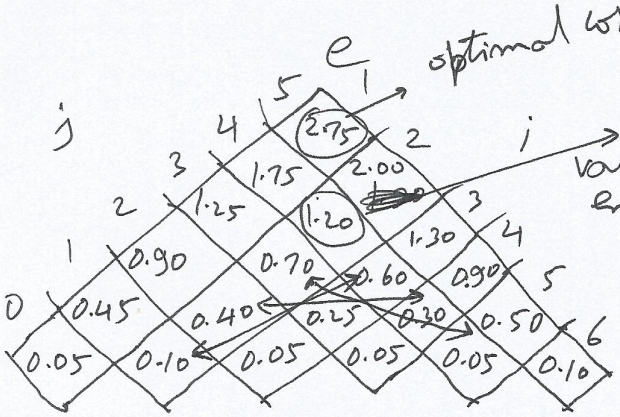
$\Downarrow$

$k_{root[1,n]}$

$k_1, k_2, \ldots, k_{root[1,n]-1}$ → Recursively find opt BST

$k_{root[1,n]+1}, \ldots k_n$ → Recursively find opt BST

Example :



For computing this value use the values at the end of arrows. Same pattern is used for all computations

For computing this take the value from tail of array. Same pattern is used in all computations.