DYNAMIC PROGRAMMING: OPTIMAL BINARY SEARCH TREE

Instructor: Dr Tarunpreet Bhatia
Assistant Professor, CSED
TIET

Cost of search in BST

Input: keys[] = {10, 12, 16}

If given no. of Nodes are n, then.

- Different No. of BST= Catalan(n) = $\frac{{}^{2n}C_n}{(n+1)} = \frac{(2n)!}{(n+1)!\, n!}$
- Different No. of Binary Trees are = n! * Catalan(n)

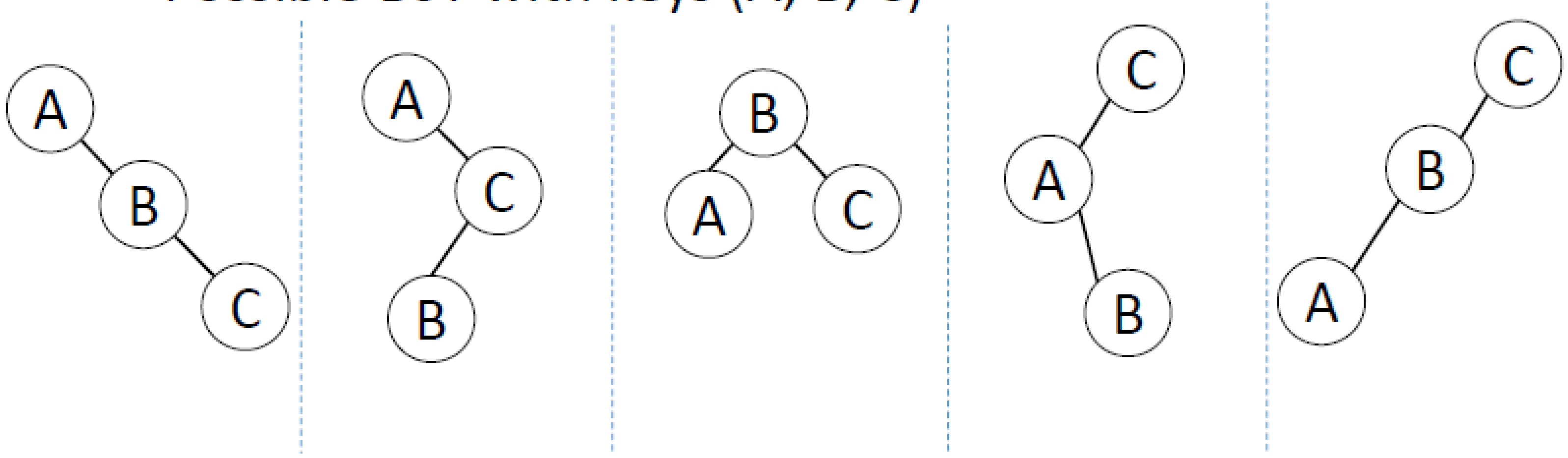
Cost of search in BST

Input: keys[] = {10, 12, 16}

freq[] = {3, 2, 5}

Possible BSTs with 3 keys

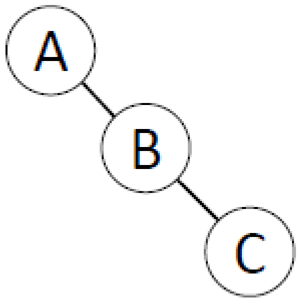
- Possible BST with keys (A, B, C)



	A	B	C
Prob.	0.6	0.3	0.1

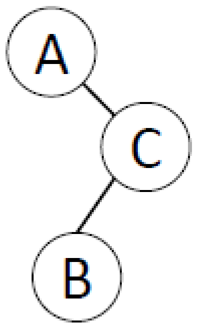
Possible BSTs with 3 keys

- Possible BST with keys (A, B, C)



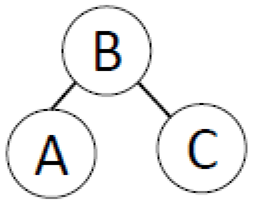
	A	B	C
prob.	0.6	0.3	0.1
comp	1	2	3
weighted prob	0.6	0.6	0.3

1.5



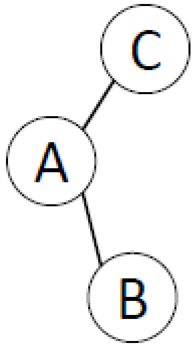
	A	B	C
prob.	0.6	0.3	0.1
comp	1	3	2
weighted prob	0.6	0.9	0.2

1.7



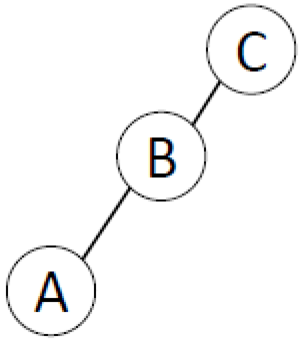
	A	B	C
prob.	0.6	0.3	0.1
comp	2	1	2
weighted prob	1.2	0.3	0.2

1.7



	A	B	C
prob.	0.6	0.3	0.1
comp	2	3	1
weighted prob	1.2	0.9	0.1

2.2



	A	B	C
prob.	0.6	0.3	0.1
comp	3	2	1
weighted prob	1.8	0.6	0.1

2.5

OBST

- OBST is one special kind of advanced tree.
- It focus on how to reduce the cost of the search of the BST.
- It may not have the lowest height !
- Two variants: One with successful search only and other with successful and unsuccessful search.

OBST

- It has n keys (representation $k_1,k_2,\dots,k_n$) in sorted order (so that $k_1<k_2<\dots<k_n$), and we wish to build a BST from these keys with minimum expected cost. For each k_i ,we have a probability p_i that a search will be for k_i.
- In contrast of, some searches may be for values not in k_i, and so we also have n+1 “dummy keys” $d_0,d_1,\dots,d_n$ representating not in k_i.
- In particular, d_0 represents all values less than k_1, and d_n represents all values greater than k_n, and for $i=1,2,\dots,n-1$, the dummy key d_i represents all values between k_i and k_{i+1}.
- The dummy keys are leaves (external nodes), and the data keys mean internal nodes.

Example

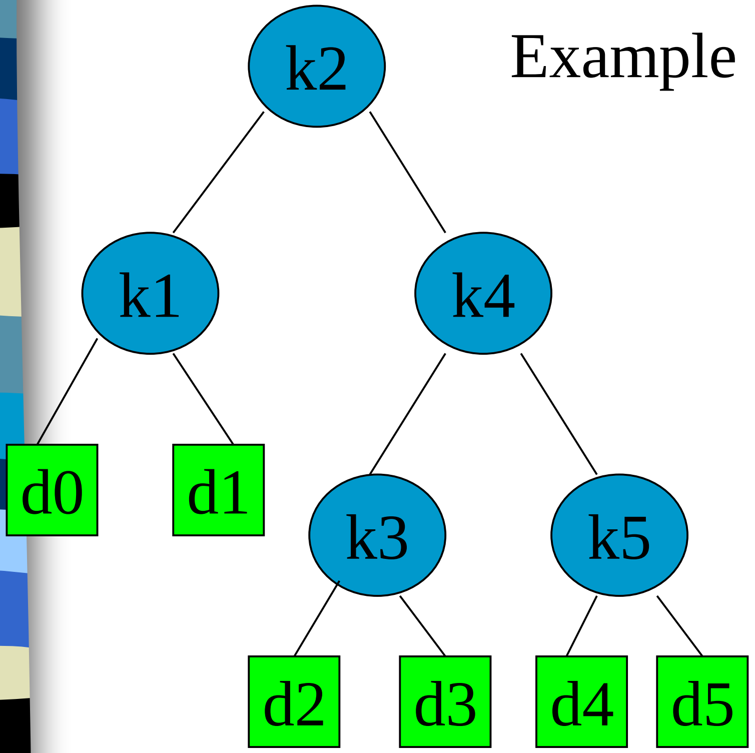


Figure (a)

i	0	1	2	3	4	5
pi		0.15	0.10	0.05	0.10	0.20
qi	0.05	0.10	0.05	0.05	0.05	0.10

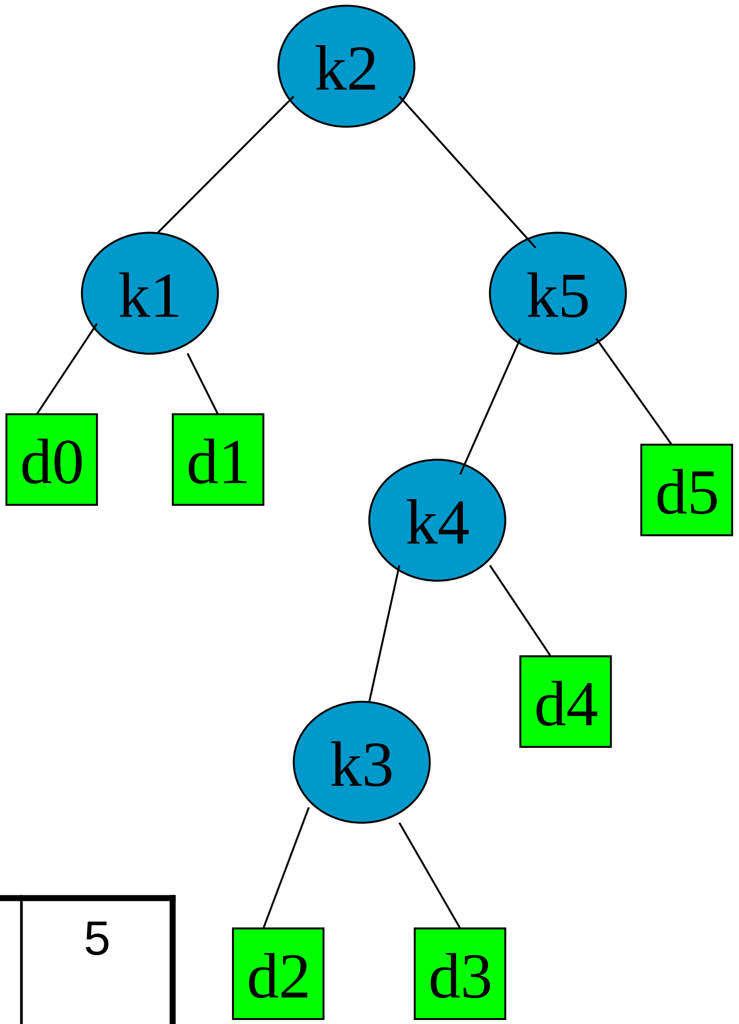


Figure (b)

- By Figure (a), we can calculate the expected search cost node by node:

Node#	Depth	probability	cost
k1	1	0.15	0.30
k2	0	0.10	0.10
k3	2	0.05	0.15
k4	1	0.10	0.20
K5	2	0.20	0.60
d0	2	0.05	0.15
d1	3	0.10	0.30
d2	3	0.05	0.20
d3	3	0.05	0.20
d4	3	0.05	0.20
d5	3	0.10	0.40

Cost=

Probability

*

(Depth+1)

Example

- And the total cost = (0.30 + 0.10 + 0.15 + 0.20 + 0.60 + 0.15 + 0.30 + 0.20 + 0.20 + 0.20 + 0.40) = 2.80
- So Figure (a) costs 2.80 ,on another, the Figure (b) costs 2.75, and that tree is really optimal.
- We can see the height of (b) is more than (a) , and the key k5 has the greatest search probability of any key, yet the root of the OBST shown is k2.(The lowest expected cost of any BST with k5 at the root is 2.85)

Finding a Recursive Solution

Critical Observation: If an optimal BST *T* has a subtree *T′* containing keys $k_i, \ldots, k_j$, then this **must be** the optimal BST for the subproblem with keys $k_i, \ldots, k_j$ and dummy keys $d_{i-1}, d_i, \ldots, d_j$.

- Given keys $k_i, \ldots, k_j$, one of these keys, say k_r (I ≦r ≦j), will be the root of an optimal subtree containing these keys. The left subtree of the root k_r will contain the keys ($k_i, \ldots, k_{r-1}$) and the dummy keys($d_{i-1}, \ldots, d_{r-1}$), and the right subtree will contain the keys ($k_{r+1}, \ldots, k_j$) and the dummy keys ($d_r, \ldots, d_j$). As long as we examine all candidate roots k_r, where i ≦r ≦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 OBST.

Finding a Recursive Solution

- The easy case occurs when j=i-1. Then we have just the dummy key d_{i-1}. The expected search cost is $e[i,i-1]= q_{i-1}$.
- When j ≥ 1, we need to select a root k_r from among $k_i,\ldots,k_j$ and then make an OBST with keys $k_i,\ldots,k_{r-1}$ its left subtree and an OBST with keys $k_{r+1},\ldots,k_j$ its right subtree. By the time, what happens to the expected search cost of a subtree when it becomes a subtree of a node? The answer is that the depth of each node in the subtree increases by 1.
- By the second statement, the excepted search cost of this subtree increases by the sum of all the 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)

Finding a Recursive Solution

- $e[i, j]$ is the expected cost of searching an optimal BST containing the keys $k_i, \ldots, k_j$. Ultimately, we wish to compute $e[1, n]$.
- For subtree with keys $k_i, \ldots, k_j$ denote

  $w(i, j) = \sum_{x=i..j} p_x + \sum_{x=i-1..j} q_x$
- If k_r is the root of the optimal subtree containing $k_i, \ldots, k_j$, we have

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

  which is equivalent to

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

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

Must add because the depth of each node in the left and right subtrees increases by one in the big tree.

Finding a Recursive Solution

- We will need one other table for efficiency. Rather than compute the value of w(i,j) from scratch, every time we are computing e[i,j] ----- we store these values in a table w[1..n+1,0..n].
- For the base case, we compute $w[i,i-1] = q_{i-1}$ for $1 \le i \le n$.
- For $j \ge i$, we compute:

  $w[i, j] = w[i, j-1] + p_j + q_j$

OBST (Successful and Unsuccessful Search)

- It needs 3 tables to record probabilities, cost, and root.
  - e[1…n+1, 0…n] : Cost
  - w[1…n+1, 0…n] : Sum of probability
  - root[1…n, 1…n] : Used to construct OBST

Recursive formula

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

```
OPTIMAL-BST($p, q, n$)
for $i \leftarrow 1$ to $n + 1$
        do $e[i, i - 1] \leftarrow q_{i-1}$
           $w[i, i - 1] \leftarrow q_{i-1}$
for $l \leftarrow 1$ to $n$
        do for $i \leftarrow 1$ to $n - l + 1$
                do $j \leftarrow i + l - 1$
                   $e[i, j] \leftarrow \infty$
                   $w[i, j] \leftarrow w[i, j - 1] + p_j + q_j$
                   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 $root$
```

Time Complexity: $O(n^3)$

Example

How do we organize a binary search tree so as to minimize the number of nodes visited in all searches, given that we know how often each key occurs?

i	0	1	2	3
pi		1	2	4
qi	1	1	1	2

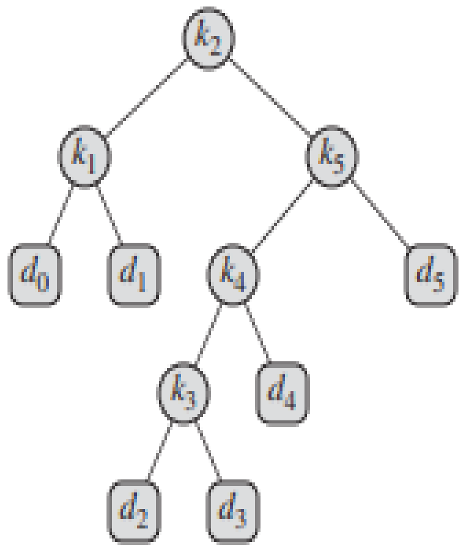
Example

i	0	1	2	3
pi		1	2	4
qi	1	1	1	2

CONSTRUCT-OPTIMAL-BST

- Write pseudocode for the procedure CONSTRUCT-OPTIMAL-BST(*root*) which, given the table *root*, outputs the structure of an optimal binary search tree.
- For the example, your procedure should print out the structure corresponding to the OBST shown below.

k_2 is the root
k_1 is the left child of k_2
d_0 is the left child of k_1
d_1 is the right child of k_1
k_5 is the right child of k_2
k_4 is the left child of k_5
k_3 is the left child of k_4
d_2 is the left child of k_3
d_3 is the right child of k_3
d_4 is the right child of k_4
d_5 is the right child of k_5



CONSTRUCT-OPTIMAL-BST

We need to do is just a pre-order tree walk of the OBST.

```
$CONSTRUCT-OPTIMAL-BST(root)$
    $n \leftarrow root.length$
    $r \leftarrow root[1][n]$
    `print` $k_r$ is the root
    $CONSTRUCT-OPTIMAL-BST-AUX(root, 1, r-1, r, left)$
    $CONSTRUCT-OPTIMAL-BST-AUX(root, r+1, n, r, right)$

$CONSTRUCT-OPTIMAL-BST-AUX(root, i, j, p, s)$
    `if` $i \leq j$
        $r \leftarrow root[i][j]$
        `print` $k_r$ is the $\langle s \rangle$ child of $k_p$
        $CONSTRUCT-OPTIMAL-BST-AUX(root, i, r-1, r, left)$
        $CONSTRUCT-OPTIMAL-BST-AUX(root, r+1, j, r, right)$
    `else`
        print $d_j$ is the $\langle s \rangle$ child of $k_p$
```

Try this!

Input: keys[] = {10, 20, 30, 40}

freq[] = {4, 2, 6, 3}

Try This!

i	0	1	2	3	4	5	6	7
pi		0.04	0.06	0.08	0.02	0.10	0.12	0.14
qi	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05