Optimal Binary Search Trees

Kuan-Yu Chen (陳冠宇)

2019/05/01 @ TR-310-1, NTUST

Review

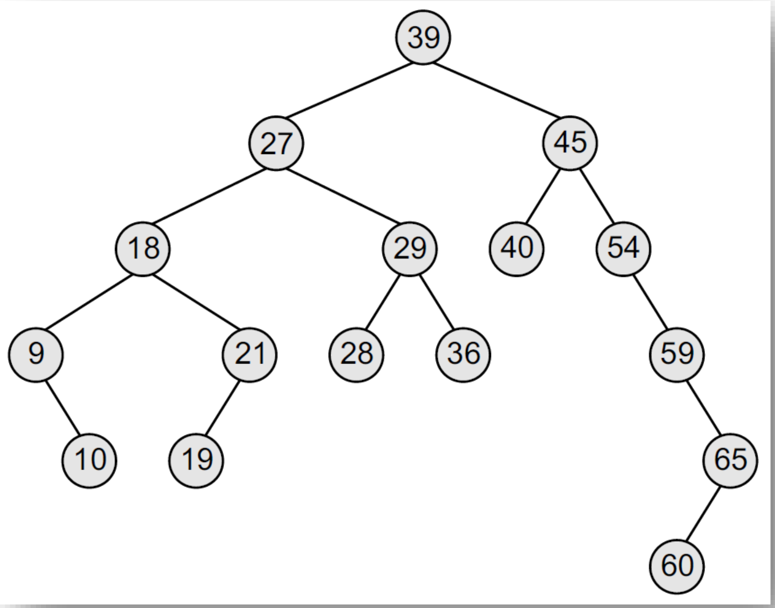
- Given two sequences X and Y, we say that a sequence Z is a ***common subsequence*** of X and Y if Z is a subsequence of both X and Y
  - $X = \{A, B, C, B, D, A, B\}$
  - $Y = \{B, D, C, A, B, A\}$
  - $Z = \{B, C, A\}$ is a common subsequence of both X and Y

  - It is worthy to note that $\{B, C, B, A\}$ is also a common subsequence of both X and Y

  - Since X and Y have no common subsequence of length 5 or greater, thus $\{B, C, B, A\}$ is an **longest-common-subsequence** of both X and Y

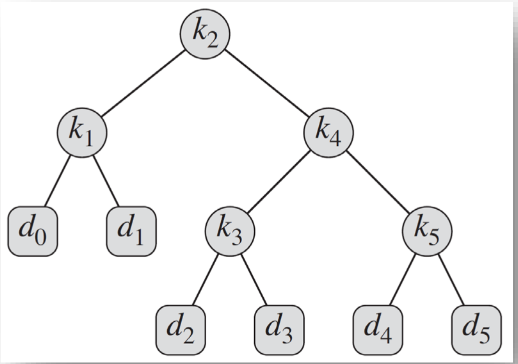
Optimal Binary Search Trees.

- A binary search tree, also known as an **ordered binary tree**, is a variant of binary trees in which the nodes are arranged in an order
  - All the nodes in the **left sub-tree** have a value **less** than that of the root node
  - All the nodes in the **right sub-tree** have a value either **equal to or greater** than the root node



Optimal Binary Search Trees..

- Formally, given a sequence $K = \{k_1, k_2, \dots, k_n\}$ of n distinct keys in sorted order
  - That is $k_1 < k_2 < \cdots < k_n$
  - For each key k_i, we have a probability p_i that a search will be
  - It should also be mentioned that some searches may be for values not in K
    - We assume that there are $n + 1$ dummy keys, $\{d_0, d_1, \dots, d_n\}$
    - d_0 represents all values less than k_1
    - d_n represents all values greater than k_n
    - 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 be



  - Every search is either successful (finding some key k_i) or unsuccessful (finding some dummy key d_i)

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

Optimal Binary Search Trees…

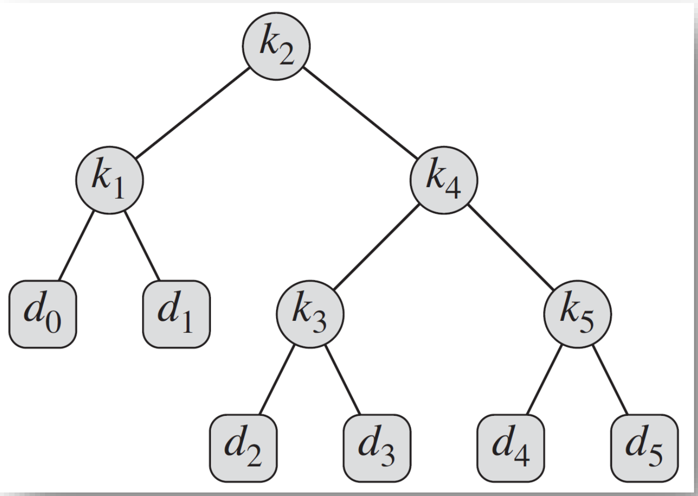
- Because we have probabilities of searches for each key and each dummy key, we can determine the expected cost of a search in a given binary search tree T
  - Let's assume that the actual cost of a search equals the number of nodes examined
    - The depth of the node found by the search in T, plus 1

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

Example – 1

- For a given binary search tree and its search probability table, we can calculate the expected cost of the tree

$$\mathrm{E}[\text{search cost in } T] = \sum_{i=1}^{n} (\mathrm{depth}_T(k_i) + 1) \times p_i + \sum_{i=0}^{n} (\mathrm{depth}_T(d_i) + 1) \times q_i = 2.8$$



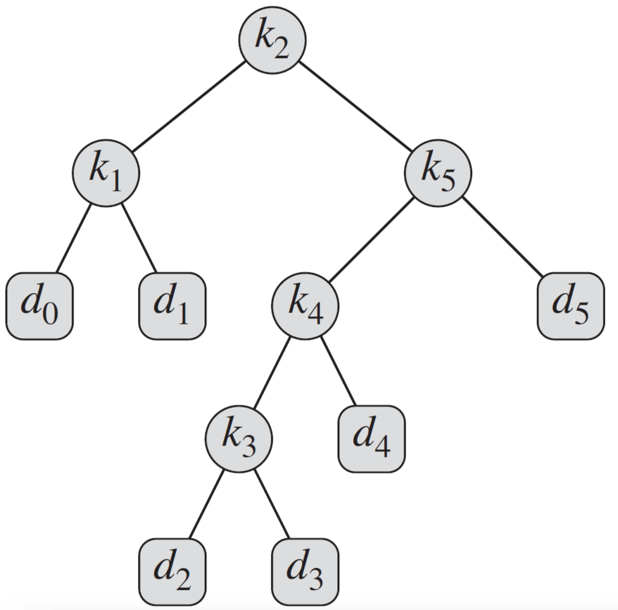
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			2.80

Example – 2

- For a given binary search tree and its search probability table, we can calculate the expected cost of the tree

$$\mathrm{E}[\text{search cost in } T] = \sum_{i=1}^{n} (\mathrm{depth}_T(k_i) + 1) \times p_i + \sum_{i=0}^{n} (\mathrm{depth}_T(d_i) + 1) \times q_i = 2.75$$



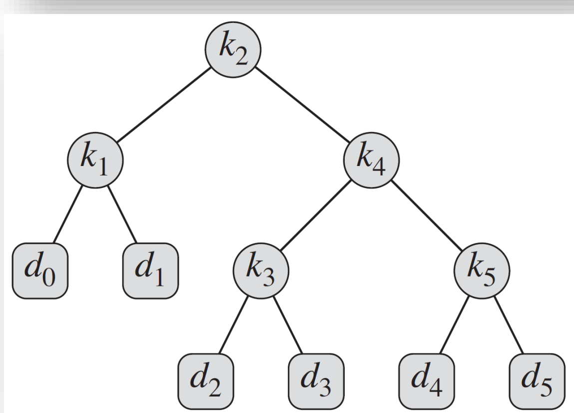
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

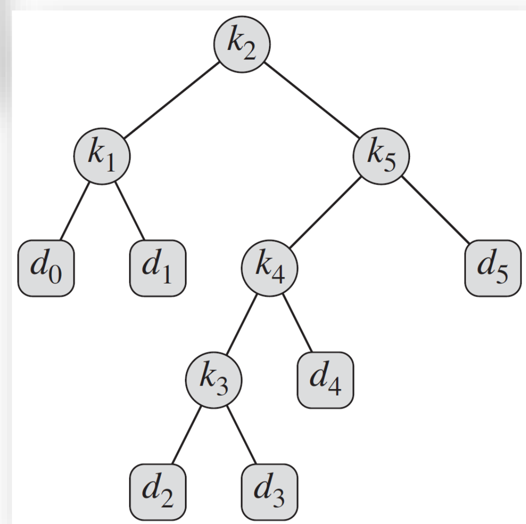
node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	3	0.05	0.2
k_4	2	0.10	0.3
k_5	1	0.20	0.4
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	4	0.05	0.25
d_3	4	0.05	0.25
d_4	3	0.05	0.20
d_5	2	0.10	0.3
Total			2.75

Optimal Binary Search Trees….

- From the two examples, we have several observations:
  - An optimal binary search tree is not necessarily a tree whose overall height is smallest
  - We **DONOT** necessarily construct an optimal binary search tree by always putting the key with the greatest probability at the root
    - The lowest expected cost of any binary search tree with k_5 at the root is 2.85

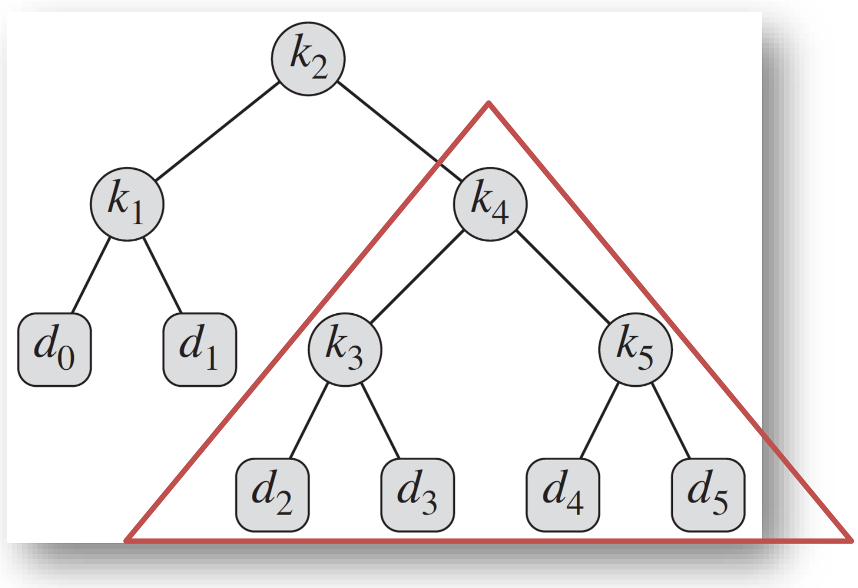
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10





DP for Optimal BSTs.

- Consider any subtree of a binary search tree
  - It must contain keys in a contiguous range $k_i, \dots, k_j$, where $1 \leq i \leq j \leq n$, and dummy keys $d_{i-1}, \dots, d_j$
  - Suppose that if k_r is the root of this subtree, the subtree is optimal
    - The left subtree contains $k_i, \dots, k_{r-1}$ and $d_{i-1}, \dots, d_{r-1}$
    - The right subtree contains $k_{r+1}, \dots, k_j$ and $d_r, \dots, d_j$

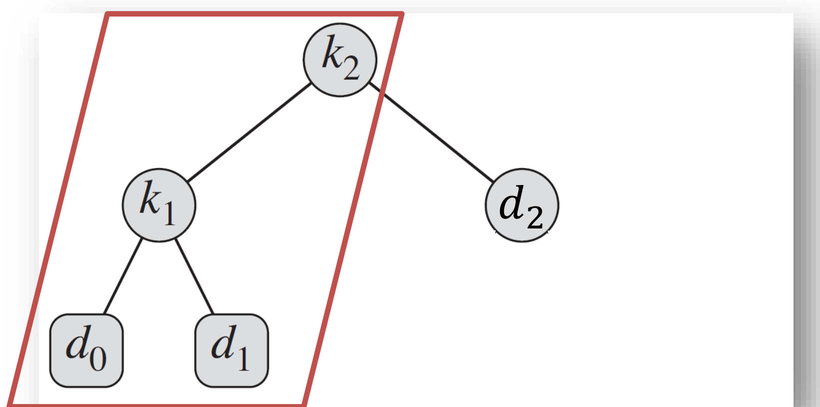


DP for Optimal BSTs..

– There is one detail worth noting about "empty" subtrees
  - Suppose that in a subtree with keys $k_i, \ldots, k_j$, we select k_i as the root

    The left subtree of k_i has no actual keys (but still has a dummy key d_{i-1})
  - Symmetrically, if we select k_j as the root

    The right subtree contains no actual keys, but it does contain the dummy key d_j



DP for Optimal BSTs…

- For a subtree with keys $k_i, \ldots, k_j$, where $1 \le i \le j \le n$, and dummy keys $d_{i-1}, \ldots, d_j$
  - We denote this sum of probabilities as

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

  - We use $e[i,j]$ to store the expected search cost for subtree $k_i, \ldots, k_j$
  - If k_r is the root of an optimal subtree containing keys $k_i, \ldots, k_j$

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

  - Thus the final recursive formulation is

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

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

DP for Optimal BSTs….

- We have three tables for efficiency
  - $e[1..n+1, 0..n]$ is used to store the expected cost
  - $root[1..n, 1..n]$ is used to store the root of each subtree
  - $w[1..n+1, 0..n]$ is used to store the accumulated probability

```
OPTIMAL-BST(p, q, n)
let e[1..n+1, 0..n], w[1..n+1, 0..n],
            and root[1..n, 1..n] be new tables
for i = 1 to n+1
    e[i, i-1] = q_{i-1}
    w[i, i-1] = q_{i-1}
for l = 1 to n
    for i = 1 to n-l+1
        j = i+l-1
        e[i, j] = ∞
        w[i, j] = w[i, j-1] + p_j + q_j
        for r = i to j
            t = e[i, r-1] + e[r+1, j] + w[i, j]
            if t < e[i, j]
                e[i, j] = t
                root[i, j] = r
return e and root
```

DP for Optimal BSTs.....

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

```
OPTIMAL-BST(p, q, n)
let e[1..n+1, 0..n], w[1..n+1, 0..n],
        and root[1..n, 1..n] be new tables
for i = 1 to n + 1
    e[i, i-1] = q_{i-1}
    w[i, i-1] = q_{i-1}
for l = 1 to n
    for i = 1 to n - l + 1
        j = i + l - 1
        e[i, j] = ∞
        w[i, j] = w[i, j-1] + p_j + q_j
        for r = i to j
            t = e[i, r-1] + e[r+1, j] + w[i, j]
            if t < e[i, j]
                e[i, j] = t
                root[i, j] = r
return e and root
```

Questions?

kychen@mail.ntust.edu.tw