Chapter 15
Dynamic Programming

动态规划（2）

15.4 最长公共子序列

一个应用背景：**DNA比对（自学）**

DNA（Deoxyribonucleic Acid，脱氧核糖核酸）是染色体的主要组成成分，由**腺嘌呤**(**A**denine)、**鸟嘌呤**(**G**uanine)、**胞嘧啶**(**C**ytosine)、**胸腺嘧啶**(**T**hymine)等四种**碱基分子**组成。

一个DNA由成千上万的碱基排列组成。

DNA的双螺旋结构

在生物学领域，用碱基名字的单词（**A**denine、**G**uanine、**C**ytosine、**T**hymine）的首字母**A**、**C**、**G**、**T**来代表这四种碱基。这样一条**DNA**上的**碱基排列**就可用**由A、C、G、T四个字母组成的字符串**表示。

如：两个有机体的DNA（碱基排列）分别表示为

S_1=ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

S_2=GTCGTTCGGAATGCCGTTGCTCTGTAAA

一个应用：可以通过比较两个有机体的**DNA**来确定这两个有机体有多么"相似"。这在生物学上叫做"**DNA比对**"。

- **DNA比对和字符串比较：**

当把DNA表示成由A、C、G、T四个字母组成的字符串后，比较两个有机体DNA的相似性就可以看作是**比较两个由字符A、C、G、T组成的字符串的相似性**。

- **怎么度量这种相似性?**

方法一：统计将其中一个字符串转换成另一个字符串所需的工作量。这个工作量越小越相似。

（参见**编辑距离问题**（*Edit distance*）**习题15-5**）。

方法二：在两个字符串（S_1、S_2）中找一个称为**最长公共子序列**的公共子串（S_3），使得 S_3 中的字符以**相同的顺序、但不一定连续**的方式出现在 S_1 和 S_2 中。并以 S_3 的长度度量相似性：S_3 越长表示 S_1 和 S_2 越相似。

如上面给出的两个DNA串，它们的具有上述性质的最长公共子串是：S_3 = GTCGTCGGAAGCCGGCCGAA

S_1 = ACCG**GTCG**AG**T**G**CG**C**G**G**AAGCCGGCCGAA**

S_2 = **GTCGT**T**CGGAA**T**GCCG**TT**GC**T**C**T**G**T**AA**A

怎么找这种公共子串呢?

S_3 以**相同的顺序、但不一定连续**的方式出现在S_1 和 S_2 中

——这就是**最长公共子序列问题**。

1、最长公共子序列问题的定义

(1) 子序列

给定两个序列 $X=<x_1, x_2, ..., x_n>$ 和序列 $Z=<z_1, z_2, ..., z_m>$，若存在 X 的一个**严格递增下标序列** $< i_1, i_2, ..., i_k >$，使得对所有 $j=1, 2, ..., k$，有 $x_{i_j} = z_j$，则称 Z 是 X 的**子序列**。

—— 即子序列 Z 中的字符以一种**“非连续”**、但**“顺序”**的方式依次出现在**“母串”** X 中。

如：$X=<A, B, C, B, D, A, B>$，$Z=<B, C, D, B>$，

则：Z 是 X 的一个子序列。

相应的下标序列为 $<2, 3, 5, 7>$。

(2) 公共子序列

对给定的两个序列 X 和 Y，若序列 Z 既是 X 的子序列，也是 Y 的子序列，则称 Z 是 X 和 Y 的**公共子序列**。

如，若再有 $Y = <\mathbf{B}, D, \mathbf{C}, \mathbf{A}, B, A>$，

$X = < A, \mathbf{B}, \mathbf{C}, B, D, \mathbf{A}, B >$

则序列 $<\mathbf{B}, \mathbf{C}, A>$ 是 X 和 Y 的一个公共子序列。

(3) 最长公共子序列

两个序列的**最长的公共子序列**称为它们的**最长公共子序列**。

如上面的 X 和 Y，$<\mathbf{B}, \mathbf{C}, A>$ 是 X 和 Y 的一个公共子序列，但不是它们的最长公共子序列。最长公共子序列是 $< \mathbf{B}, \mathbf{C}, \mathbf{B}, \mathbf{A} >$。

$X = < A, \mathbf{B}, \mathbf{C}, \mathbf{B}, D, \mathbf{A}, B >$
$Y = < \mathbf{B}, D, \mathbf{C}, A, \mathbf{B}, \mathbf{A}>$

求两个序列的**最长公共子序列**的问题就称为**最长公共子序列问题**，简称为**LCS问题**（*Longest-Common-Subsequence Question*）。

如何求两个序列的LCS呢?

一个可能的计算过程可以这样设计:

置 $i=1\rightarrow m$，$j=1\rightarrow n$，依次比较 X 和 Y 的当前**字符对** (x_i, y_j)：

- 如果 $x_i=y_j$，则 x_i (y_j) 至少会出现在一个公共子序列中；
- 如果 $x_i\neq y_j$，再比较其它字符对。

从而有可能的处理方式：

```
for i =1 to m
    for j =1 to n
        if xi==yj ……
        else ……
```

这里要解决的问题是：如何用**这样的字符**构造出**“更长”**的公共子序列。

前缀：对给定的序列 $X = <x_1, x_2, ..., x_m>$，对于 $i = 0, 1, ..., m$ ，定义 X 的**第 i 个前缀**为 $X_i = <x_1, x_2, ..., x_i>$，即 X 的前 i 个元素构成的“**子序列**”。

如，$X = <A, B, C, B, D, A, B>$，则

$$X_4 = <A, B, C, B>$$

显然，$X_m = X$，

$$X_0 = \Phi$$

$$X_1 = <x_1>$$

$$X_2 = <x_1, x_2>$$

$$X_3 = <x_1, x_2, x_3>$$

……

对计算过程：

```
for i =1 to m
    for j =1 to n
        if x_i==y_j ……
        else ……
```

当前位置：

（1）i 和 j 分别定义了 X 的前缀 X_i 和 Y 的前缀 Y_j。

（2）此处相当于求前缀 X_i 和 Y_j 的 $LCS(X_i, Y_j)$。

（3）此处**“更长”**的公共子序列可以**在前面的某个*LCS*的后面附加 x_i 得到**。

（4）此处求出的 $LCS(X_i, Y_j)$ 也将是后续计算的基础：如果后面又发现新的**“相等字符”**，可以基于此求更长的公共子序列。

（5）这个过程持续进行，当计算到 $LCS(X_m, Y_n)$ 就求出整个问题的解了。

```
for i =1 to m
    for j =1 to n
        if x_i==y_j ……
        else ……
```

随着 i 从 1 至 m、j 从 1 至 n 的变化，该过程的主体就是逐步计算所有可能的**前缀对** (X_i, Y_j) 的LCS。这样的前缀对有：

(X_1, Y_1)、(X_1, Y_2)、(X_1, Y_3)、……、

(X_2, Y_1)、(X_2, Y_2)、(X_2, Y_3)、……、

(X_3, Y_1)、(X_3, Y_2)、(X_3, Y_3)、……、

……… (X_m, Y_n)。

这些前缀对就规定了一系列的子问题，按序依次计算。

当求出 (X_m, Y_n) 的LCS后，过程结束。

可形成这样一个构造LCS的过程：

从**长度为 0 的空公共子序列**开始，在每找到一对相等的 x_i 和 y_j（即 $x_i = y_j$）时，就将 x_i 附加到**以前找到的一个最长公共子序列**之后，使公共子序列的长度**不断变长**而最终构造出来的。

如何找到 **"以前找到的一个最长公共子序列"** ？

—— 它就是前面一个 **"尽可能长的前缀对"** 之间的 *LCS*。

```
for i =1 to m
    for j =1 to n
        if xi==yj ……
        else ……
```

对当前位置（*i*, *j*）而言，这样的 **"尽可能长的前缀对"** **就是**（$X_{i\text{-}1}$, $Y_{j\text{-}1}$）。

记 $Z_{a,b}$ 是 X_a 和 Y_b 的LCS，即 $Z_{a,b} = LCS(X_a, Y_b)$。

则， $Z_{0,0} = LCS(X_0, Y_0)$

$Z_{1,2} = LCS(X_1, Y_2)$

……

$Z_{m,n} = LCS(X_m, Y_n) = LCS(X, Y)$

根据上述分析，对于位置（i, j），

则若 $x_i = y_j$，就在 $Z_{i\text{-}1,\, j\text{-}1}$ (即$X_{i\text{-}1}$ 和 $Y_{j\text{-}1}$ 的LCS) 后面附加 x_i (也或者是 y_j）来构造 (X_i, Y_j) 的LCS：

即：当 $x_i = y_j$ 时，$Z_{i,j} = Z_{i\text{-}1,\, j\text{-}1} + \langle x_i \rangle$

如图，当 $x_i = y_j$ 时，$Z_{i,j} = Z_{i-1,\, j-1} + \langle x_i \rangle$

$$x_1, x_2, \ldots, x_{i-1}, x_i, \ldots\ldots, x_m$$

$$Z_{i,j} = \left|\longleftarrow Z_{i-1,\, j-1} \longrightarrow\right| + x_i$$

$$y_1, y_2, \ldots, y_{j-1}, y_j, \ldots\ldots, y_n$$

注：$Z_{i-1,\, j-1}$ 就是子问题（求X_{i-1}和Y_{j-1}的LCS）的解。因此 $Z_{i,j}$ 就是在较小问题解的基础上进一步构造出来的较大子问题的解。

而如果 $x_i \neq y_j$，则 x_i 和 y_j 对基于 $Z_{i-1, j-1}$ 构造**更长的公共子序列**就没有贡献。这意味着：

截止到当前位置 (i, j)，能够获得的最长公共子序列 $Z_{i, j}$ 仅等于所有“**前面的子问题**”中能找到的最长公共子序列。

而这里能找到可能的最长公共子序列的“**前面的子问题**”仅有两个：

- 一是 X_i 和 Y_{j-1} 的LCS，即 $Z_{i, j-1}$。
- 二是 X_{i-1} 和 Y_j 的LCS，即 $Z_{i-1, j}$。

均是这样情形下“尽可能长的前缀对”之间的“尽可能长的*LCS*”。

那么，$Z_{i, j}$ 将 **“继承”** 于 $Z_{i, j-1}$ 和 $Z_{i-1, j}$ 二者中**较大**的一个，

即：$Z_{i, j} = max\,(Z_{i, j-1}, Z_{i-1, j})$。

$x_1, x_2, \ldots, x_i, \ldots\ldots, x_m$

$y_1, y_2, \ldots, y_{j-1}, y_j, \ldots\ldots, y_n$

$$Z_{i,j} = max \left(\begin{array}{c} \longleftarrow Z_{i,j-1} \longrightarrow \\ \\ \longleftarrow Z_{i-1,j-1} \longrightarrow \end{array} \right)$$

$x_1, x_2, \ldots, x_{i-1}, x_i, \ldots\ldots, x_m$

$y_1, y_2, \ldots, y_j, \ldots\ldots, y_n$

当 $x_i \neq y_j$ 时，$Z_{i,j} = max\,(Z_{i,j-1}, Z_{i-1,j})$

综上所述，可得一个“**递推计算**”过程：

对当前的 i 和 j，X_i 和 Y_j 的LCS有以下递推关系式：

$$Z_{i,j} = \begin{cases} Z_{i-1,j-1} + <x_i> & x_i = y_j \\ \max\{Z_{i,j-1}, Z_{i-1,j}\} & x_i \neq y_j \end{cases}$$

上述推导的正确性由定理15.2给出：

定理15.2 设有序列 $X = <x_1, x_2, ..., x_m>$ 和 $Y = <y_1, y_2, ..., y_n>$，并设序列 $Z = <z_1, z_2, ..., z_k>$ 为 X 和 Y 的任意一个**LCS**。

（1）若 $x_m = y_n$，则 $z_k = x_m = y_n$，且 Z_{k-1} 是 X_{m-1} 和 Y_{n-1} 的一个LCS。

（2）若 $x_m \neq y_n$，则 $z_k \neq x_m$ 蕴含 Z 是 X_{m-1} 和 Y 的一个LCS。

（3）若 $x_m \neq y_n$，则 $z_k \neq y_n$ 蕴含 Z 是 X 和 Y_{n-1} 的一个LCS。

证明:

① **对于(1)**（即若 $x_m = y_n$，则 $z_k = x_m = y_n$，且 Z_{k-1} 是 X_{m-1} 和 Y_{n-1} 的一个LCS）：

如果 $z_k \neq x_m$，既然 $x_m = y_n$，则可以**添加** x_m（或 y_n）到 Z 中而得到一个新的子序列 Z'，Z' 的长度 = Z 的长度 + 1，且 Z' 也是 X 和 Y 的公共子序列，这与 Z 是 X 和 Y 的最长公共子序列相矛盾。故只能有 $z_k = x_m = y_n$。

另外，若 X_{m-1} 和 Y_{n-1} 有一个**长度大于 k-1** 的公共子序列 W，则**将 x_m（y_n）添加到 W 后面**就会产生一个 X 和 Y 的长度大于 k 的公共子序列 Z'，这又与 Z 是 X 和 Y 的最长公共子序列的假设相矛盾，故 Z_{k-1} 必是 X_{m-1} 和 Y_{n-1} 的一个LCS。

② **对于(2)**（即若 $x_m \neq y_n$，则 $z_k \neq x_m$ 蕴含 Z 是 X_{m-1} 和 Y 的一个LCS）

因为 $x_m \neq y_n$ 且 $z_k \neq x_m$，所以 Z 是 $LCS(X, Y)$ 的同时，也是 X_{m-1} 和 Y 的一个公共子序列。是不是 X_{m-1} 和 Y 的 LCS 呢?

由于 X_{m-1} 和 Y 的公共子序列也必是 X_m 和 Y 的公共子序列。故，若假设 X_{m-1} 和 Y 有一个**长度大于** k 的公共子序列 W，则 W 同时也将是 X_m 和 Y 的一个公共子序列，这与 Z 是 X 和 Y 的LCS相矛盾。

所以假设不成立。故 Z 是 X_{m-1} 和Y 的一个LCS。

③ **对于(3)**（即若 $x_m \neq y_n$，则 $z_k \neq y_n$ 蕴含 Z 是 X 和 Y_{n-1} 的一个LCS）

同（2），证略。

3、LCS问题的最优子结构性

定理15.2也说明了LCS问题满足最优子结构性：

求X_{m-1}和Y_{n-1}的LCS、求X_{m-1}和Y_n的LCS以及求X_m和Y_{n-1}的LCS，都可视为求X_m和Y_n的LCS问题的**子问题**。

由定理可知，如果Z_k是X_m和Y_n的最优解（最长公共子序列），则

- Z_k的子序列Z_{k-1}也必是子问题X_{m-1}和Y_{n-1}的最优解（$z_k = x_m = y_n$时）；
- 或者，其自身就是子问题X_{m-1}和Y_n的最优解（$x_m \neq y_n$且$z_k \neq x_m$时）或是子问题X_m和Y_{n-1}的最优解（$x_m \neq y_n$且$z_k \neq y_n$时）。

即，若Z_k是最优的，则其内部子序列也是最优的，所以**LCS问题具有最优子结构性质**。

$$Z_{i,j}=\begin{cases} Z_{i-1,j-1}+<x_i> & x_i=y_j \\ \max\{Z_{i,j-1}, Z_{i-1,j}\} & x_i\neq y_j \end{cases}$$

4、LCS 问题的状态转移方程

定义 $c[i, j]$ 为 X_i和 Y_j的LCS的**长度**，根据定理15.2可得 $c[i, j]$ 的递推计算公式：

$$c[i,j]=\begin{cases} 0 & \text{如果} i=0 \text{或} j=0 \\ c[i-1,j-1]+1 & \text{如果} i,j>0 \text{且} x_i=y_j \\ \max(c[i,j-1],c[i-1,j]) & \text{如果} i,j>0 \text{且} x_i\neq y_j \end{cases}$$

具体含义如下：

（1）边界条件： $i=0$ 或 $j=0$，此时代表 X_i或 Y_j中至少有一个为空序列，所以二者不存在“公共子序列”，二者的LCS也就是**Φ**，所以此时 $c[i,j]=0$。

$$c[i,j]=\begin{cases}0 & \text{如果}i=0\text{或}j=0\\ c[i-1,j-1]+1 & \text{如果}i,j>0\text{且}x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果}i,j>0\text{且}x_i\neq y_j\end{cases}$$

含义（续）：

（2）若 $x_i = y_j$，根据前面的分析，此时 X_i 和 Y_j 的LCS是在 $X_{i\text{-}1}$ 和 $Y_{j\text{-}1}$ 的LCS之后附加字符 x_i（y_j）得到的，所以：

$$c[i, j] = c[i\text{-}1, j\text{-}1] + 1\,;$$

（3）若 $x_i \neq y_j$，X_i 和 Y_j 的LCS等于 LCS($X_{i\text{-}1}$, Y_j) 和 LCS(X_i, $Y_{j\text{-}1}$) 中**长的那一个**。所以

$$c[i, j] = max(c[i\text{-}1, j], c[i, j\text{-}1])。$$

5、算法的设计

$$c[i,j]=\begin{cases}0 & 如果 i=0 或 j=0\\ c[i-1,j-1]+1 & 如果 i,j>0 且 x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & 如果 i,j>0 且 x_i\neq y_j\end{cases}$$

（1）递推计算

$c[i\text{-}1, j\text{-}1]$、$c[i, j\text{-}1]$、$c[i\text{-}1, j]$ 是三个子问题 LCS($X_{i\text{-}1}$, $Y_{j\text{-}1}$)、LCS(X_i ,$Y_{j\text{-}1}$)、LCS($X_{i\text{-}1}$,Y_j) 的解，而 $c[i, j]$ 是在这三个子问题解的基础上进一步计算得到的。所以这个过程需要完成一系列 $c[i, j]$ 的计算，$1\le i \le m$，$1\le j \le n$，共 mn 个。

递推计算：先计算**较小子问题**的解（$c[i\text{-}1, j\text{-}1]$、$c[i, j\text{-}1]$、$c[i\text{-}1, j]$），再计算较大子问题的解（$c[i, j]$），直至$c[m, n]$结束。

（2）造表：

这里 c **就是一个表——二维数组**，记录上述过程中计算出的所有 $c[i, j]$ 的值，$0 \le i \le m, 0 \le j \le n$。

$$c[i,j]=\begin{cases}0 & \text{如果} i=0 \text{或} j=0\\ c[i-1,j-1]+1 & \text{如果} i,j>0 \text{且} x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果} i,j>0 \text{且} x_i\neq y_j\end{cases}$$

- ***b* 表:**

$c[i,j]$ 是 X_i 和 Y_j 的LCS**的长度**， 如何求**LCS的字符序列?**

分析 $c[i,j]$ 取值的来源，**有三个**：

- 由 $c[i\text{-}1, j\text{-}1]$ 加1 得到
- 或直接取 $c[i, j\text{-}1]$
- 或直接取 $c[i\text{-}1, j]$

——对于LCS问题，取哪个值就是**决策**。

为此定义另外一个表：$b[1..m,\ 1..n]$，**记录** $c[i,\ j]$ **的来源**，并在完成 $c[m, n]$ 计算后，利用 b 求出 X_m 和 Y_n 的LCS的字符序列。

（3）过程LCS-LENGTH(X, Y) 实现 c 表和 b 表的具体计算。

$$c[i,j]=\begin{cases}0 & \text{如果}i=0\text{或}j=0\\ c[i-1,j-1]+1 & \text{如果}i,j>0\text{且}x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果}i,j>0\text{且}x_i\neq y_j\end{cases}$$

LCS-LENGTH(*X*, *Y*)

```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i - 1, j - 1]+1
            b[i, j] = "↖"
        elseif c[i - 1, j] ≥ c[i, j -1]
            c[i, j] = c[i - 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j - 1]
            b[i, j] = "←"
return c and b
```

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

*b*表中的“**箭头**”记录了*c*[*i*, *j*] 的来源

LCS-LENGTH的时间复杂度为 ***O(mn)***

例，计算$X = <A, B, C, B, D, A, B>$和$Y = <B, D, C, A, B, A>$的LCS。

LCS-LENGTH的执行过程用下面的**表格**表示。

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

说明:

- 行表示 X，**行下标**是 i。
  列表示 Y，**列下标**是 j；。
- 方格的内容是 $c[i, j]$ 的值以及$b[i, j]$ 的箭头。
- 第一行：所有 $c[0, j]$ 的值，$0 \le j \le n$。
  由于$i = 0$，所以所有 $c[0,j]$ 都等于0。
- 第一列：所有 $c[i, 0]$ 的值，$0 \le i \le m$。
  由于$j = 0$，所以所有 $c[i,0]$ 都等于0。

分步计算如下

LCS-LENGTH(X,Y)

$m = X.length$

$n = Y.length$

let **b[1 . .m, 1..n]** and **c[0..m,0..n]** be new tables

for $i = 1$ **to** m

  $\mathbf{c[}\boldsymbol{i}\mathbf{, 0] = 0}$

for $j = 0$ **to** n

  $\mathbf{c[0,}\ \boldsymbol{j}\mathbf{] = 0}$

$$c[i,j] = \begin{cases} 0 & \text{如果} i = 0 \text{或} j = 0 \\ c[i-1, j-1] + 1 & \text{如果} i, j > 0 \text{且} x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{如果} i, j > 0 \text{且} x_i \neq y_j \end{cases}$$

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

第一行、第一列：

- 第一行是所有 $c[0, j]$ 的值，$0 \le j \le n$。都等于0。
- 第一列是所有 $c[i, 0]$ 的值，$0 \le i \le m$。都等于0。

```
            c[i,j] = c[i - 1, j - 1]+1
            b[i, j] = "↖"
        elseif c[i - 1, j] ≥ c[i, j -1]
            c[i, j] = c[i - 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j - 1]
            b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & \text{如果} i=0 \text{或} j=0\\ c[i-1,j-1]+1 & \text{如果} i,j>0 \text{且} x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果} i,j>0 \text{且} x_i\neq y_j\end{cases}$$

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

第二行：i =1，计算所有的 $c[1, j]$ 。

① $c[1,1]$

因为 $x_1 \neq y_1$，所以有：

$$c[1,1] = max\{c[1,0], c[0,1]\}$$

$$= max\{0, 0\}$$

$$= 0$$

$b[1,1]$ = '↑'

注：在表格中，相对 $c[i, j]$，

- $c[i\text{-}1, j]$ 是其上方元素（上一行、同列的元素）
- $c[i, j\text{-}1]$ 是其左侧元素（同行、左一列的元素）
- $c[i\text{-}1, j\text{-}1]$ 是其左上角元素（上一行且左一列的元素）

```
        c[i,j] = c[i - 1, j - 1]+1
        b[i, j] = "↖"
    elseif c[i - 1, j] ≥ c[i, j -1]
        c[i, j] = c[i - 1, j]
        b[i, j] = "↑"
    else c[i, j] = c[i, j - 1]
        b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & \text{如果} i=0 \text{或} j=0\\ c[i-1,j-1]+1 & \text{如果} i,j>0 \text{且} x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果} i,j>0 \text{且} x_i\neq y_j\end{cases}$$

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

② $c[1,2]$

因为 $x_1 \neq y_2$，所以有：

$$c[1,2] = max\{c[1,1], c[0,2]\}$$
$$= max\{0, 0\}$$
$$= 0$$

$b[1,2]$ = '↑'

```
            c[i,j] = c[i - 1, j - 1]+1
            b[i, j] = "↖"
        elseif c[i - 1, j] ≥ c[i, j -1]
            c[i, j] = c[i - 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j - 1]
            b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & \text{如果} i=0 \text{或} j=0\\ c[i-1,j-1]+1 & \text{如果} i,j>0 \text{且} x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果} i,j>0 \text{且} x_i\neq y_j\end{cases}$$

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			
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

③ $c[1,3]$

因为 $x_1 \neq y_3$，所以有:

$c[1,3] = max\{c[1,2], c[0,3]\}$

$= max\{0, 0\}$

$= 0$

$b[1,3] =$ '↑'

```
            c[i,j] = c[i - 1, j - 1]+1
            b[i, j] = "↖"
      elseif c[i - 1, j] ≥ c[i, j -1]
            c[i, j] = c[i - 1, j]
            b[i, j] = "↑"
      else c[i, j] = c[i, j - 1]
            b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & \text{如果} i=0 \text{或} j=0\\ c[i-1,j-1]+1 & \text{如果} i,j>0 \text{且} x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果} i,j>0 \text{且} x_i\neq y_j\end{cases}$$

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

④ $c[1,4]$

因为 $x_1 = y_4$，所以有：

$c[1,4] = c[0,3]+1$

$= 0+1$

$= 1$

$b[1,4] =$ '↖'

```
        c[i,j] = c[i - 1, j - 1]+1
        b[i, j] = "↖"
    elseif c[i - 1, j] ≥ c[i, j -1]
        c[i, j] = c[i - 1, j]
        b[i, j] = "↑"
    else c[i, j] = c[i, j - 1]
        b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & \text{如果}\ i=0\ \text{或}\ j=0\\ c[i-1,j-1]+1 & \text{如果}\ i,j>0\ \text{且}\ x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果}\ i,j>0\ \text{且}\ x_i\neq y_j\end{cases}$$

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

⑤ $c[1,5]$

因为 $x_1 \neq y_5$，所以有：

$$c[1,5]=max\{c[1,4],\ c[0,5]\}$$

$$=max\{1,\ 0\}$$

$$=\ 0$$

$$b[1,5]=\ '\leftarrow'$$

```
            c[i,j] = c[i - 1, j - 1]+1
            b[i, j] = "↖"
        elseif c[i - 1, j] ≥ c[i, j -1]
            c[i, j] = c[i - 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j - 1]
            b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & \text{如果} i=0 \text{或} j=0\\ c[i-1,j-1]+1 & \text{如果} i,j>0 \text{且} x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果} i,j>0 \text{且} x_i\neq y_j\end{cases}$$

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						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

⑥ $c[1,6]$

因为 $x_1=y_6$，所以有：

$c[1,6]=c[0,5]+1$

$=0+1$

$=1$

$b[1,6]=$ '↖'

第一行计算完毕

```
        c[i,j] = c[i - 1, j - 1]+1
        b[i, j] = "↖"
    elseif c[i - 1, j] ≥ c[i, j -1]
        c[i, j] = c[i - 1, j]
        b[i, j] = "↑"
    else c[i, j] = c[i, j - 1]
        b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & \text{如果}\, i=0\,\text{或}\, j=0\\ c[i-1,j-1]+1 & \text{如果}\, i,j>0\,\text{且}\, x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果}\, i,j>0\,\text{且}\, x_i\neq y_j\end{cases}$$

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					
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

第三行：i=2，计算所有的 $c[2, j]$。

① $c[2,1]$

因为 $x_2 = y_1$，所以有：

$c[2,1] = c[1,0]+1$

$= 0 + 1$

$= 1$

$b[2,1] =$ '↖'

```
            c[i,j] = c[i - 1, j - 1]+1
            b[i, j] = "↖"
        elseif c[i - 1, j] ≥ c[i, j -1]
            c[i, j] = c[i - 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j - 1]
            b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & \text{如果} i=0 \text{或} j=0\\ c[i-1,j-1]+1 & \text{如果} i,j>0 \text{且} x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果} i,j>0 \text{且} x_i\neq y_j\end{cases}$$

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

② $c[2,2]$

因为 $x_2\neq y_2$，所以有：

$c[2,2] = max\{c[2,1], c[1,2]\}$

$= max\{1, 0\}$

$= 1$

$b[2,2] =$ '←'

```
        c[i,j] = c[i - 1, j - 1]+1
        b[i, j] = "↖"
    elseif c[i - 1, j] ≥ c[i, j -1]
        c[i, j] = c[i - 1, j]
        b[i, j] = "↑"
    else c[i, j] = c[i, j - 1]
        b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & \text{如果} i=0 \text{或} j=0\\ c[i-1,j-1]+1 & \text{如果} i,j>0 \text{且} x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果} i,j>0 \text{且} x_i\neq y_j\end{cases}$$

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			
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

③ $c[2,3]$

因为 $x_2\neq y_3$，所以有：

$$c[2,3]=max\{c[2,2],c[1,3]\}$$
$$=max\{1,0\}$$
$$=1$$

$b[2,3]=$ '←'

```
        c[i,j] = c[i - 1, j - 1]+1
        b[i, j] = "↖"
    elseif c[i - 1, j] ≥ c[i, j -1]
        c[i, j] = c[i - 1, j]
        b[i, j] = "↑"
    else c[i, j] = c[i, j - 1]
        b[i, j] = "←"
```

$$c[i,j] = \begin{cases} 0 & \text{如果} i = 0 \text{或} j = 0 \\ c[i-1, j-1] + 1 & \text{如果} i, j > 0 \text{且} x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{如果} i, j > 0 \text{且} x_i \neq y_j \end{cases}$$

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		
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

④ $c[2,4]$

因为 $x_2 \neq y_4$，所以有：

$c[2,4] = max\{c[2,3], c[1,4]\}$

$= max\{1, 1\}$

$= 1$

$b[2,4] =$ '↑'

```
            c[i,j] = c[i - 1, j - 1]+1
            b[i, j] = "↖"
      elseif c[i - 1, j] ≥ c[i, j -1]
            c[i, j] = c[i - 1, j]
            b[i, j] = "↑"
      else c[i, j] = c[i, j - 1]
            b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & 如果 i=0 或 j=0\\ c[i-1,j-1]+1 & 如果 i,j>0 且 x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & 如果 i,j>0 且 x_i\neq y_j\end{cases}$$

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	
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

⑤ $c[2,5]$

因为 $x_2 = y_5$，所以有：

$c[2,5] = c[1,4]+1$

$= 1+1$

$= 2$

$b[2,5] =$ '↖'

```
        c[i,j] = c[i - 1, j - 1]+1
        b[i, j] = "↖"
    elseif c[i - 1, j] ≥ c[i, j -1]
        c[i, j] = c[i - 1, j]
        b[i, j] = "↑"
    else c[i, j] = c[i, j - 1]
        b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & \text{如果} i=0 \text{或} j=0\\ c[i-1,j-1]+1 & \text{如果} i,j>0 \text{且} x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果} i,j>0 \text{且} x_i\neq y_j\end{cases}$$

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						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

⑥ $c[2,6]$

因为 $x_2 \neq y_6$，所以有：

$$c[2,6] = max\{c[2,5], c[1,6]\}$$

$$= max\{2, 1\}$$

$$= 2$$

$b[2,6]$ = '←'

第二行计算完毕

```
        c[i,j] = c[i - 1, j - 1]+1
        b[i, j] = "↖"
    elseif c[i - 1, j] ≥ c[i, j -1]
        c[i, j] = c[i - 1, j]
        b[i, j] = "↑"
    else c[i, j] = c[i, j - 1]
        b[i, j] = "←"
```

$$c[i,j]=\begin{cases}0 & \text{如果}i=0\text{或}j=0\\ c[i-1,j-1]+1 & \text{如果}i,j>0\text{且}x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果}i,j>0\text{且}x_i\neq y_j\end{cases}$$

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

继续完成所有$c[i,j]$的计算

最后的$c[7, 6]$=4，说明 X 和 Y 的 LCS 包含**4个字符**

6、利用 *b* 表构造 LCS 的字符序列

基本策略：

当 $c[m, n]$ 计算完成后，从右下角的 $[m, n]$ 坐标处开始：

① 分析当前表项 $b[i, j]$ 的 **"箭头"**，获取LCS中可能的字符；

② 沿 $b[i, j]$ 箭头的指向**向左、向上**推导，直至到达**左上方**的边界处结束。

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

这一过程中记录下来的字符就构成了一个LCS序列，具体如下：

开始时置 $i \leftarrow m$， $j \leftarrow n$，然后分析 $b[i, j]$ 取值，分两种情况处理：

① 若 $b[i, j]=$ "↖"：根据计算过程，此时意味着 $x_i = y_j$，是将 x_i 附加到 $X_{i\text{-}1}$ 和$Y_{j\text{-}1}$的LCS后面构造出一个关于 X_i和 Y_j的新LCS。所以，

- 此时的 x_i **将是LCS中一个字符**，记录下来。
- 然后置 $i \leftarrow i\text{-}1$， $j \leftarrow j\text{-}1$，根据箭头指向继续对左上方表项处理。

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

如$b[6,6]$，作为**从后向前**递推遇到的第一个"↖"，此处的字符"A"是 X 和 Y 的LCS的**最后一个字符**。

② 若$b[i, j]$= “←” 或 $b[i, j]$= “↑” ：

此时意味着 $x_i \neq y_j$， x_i 和 y_j 对构造新LCS没有贡献，所以，

- **x_i 和 y_j 均不记入LCS序列。**
- 若 $b[i, j]$="←"， 仅置 $j \leftarrow j-1$ **向左**对**左侧**表项继续处理；
- 若$b[i, j]$= "↑"， 仅置 $i \leftarrow i-1$ **向上**对**上方**表项继续处理。

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

如$b[7,6]$， **“↑”意味着此处的LCS直接继承于上方的子问题**，所以仅置$j \leftarrow j-1$，转而处理**上方表项**$b[6,6]$即可。

重复 ①、②，直到 $i = 0$ 或 $j = 0$ 结束

过程实现：

```
PRINT-LCS (b, X, i, j)
    if i == 0 or j== 0
        return
    if b[i, j]== "↖"
        PRINT-LCS(b, X, i - 1, j -1)    //先通过递归处理前面的子序列
        print x_i    //再输出字符x_i，以使得输出内容是正序的
    elseif  b[i, j]== "↑"
        PRINT-LCS (b, X, i - 1, j)    //向上搜索
    else PRINT-LCS(b, X, i, j-1)    //向左搜索
```

初次调用形式：PRINT-LCS(b, X, m, n)

时间分析：由于每一次循环都使 i 或 j 减1，当 $i = 0$ 或 $j = 0$ 时算法结束，所以PRINT-LCS的时间复杂度为 $O(m+n)$。

```
PRINT-LCS (b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i - 1, j - 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS (b, X, i - 1, j)
else PRINT-LCS(b, X, i, j-1)
```

	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

PRINT-LCS(*b*,*X*,7,6) 的执行过程

初始调用： PRINT-LCS(*b*,*X*,7,6)　　**结束**

第1级递归：PRINT-LCS(*b*, *X*, 6, 6)

第2级递归：PRINT-LCS(*b*, *X*, 5, 5)　　返回后输出　**print A**

第3级递归：PRINT-LCS(*b*, *X*, 4, 5)

第4级递归：PRINT-LCS(*b*, *X*, 3, 4)　　返回后输出　**print B**

第5级递归：PRINT-LCS(*b*, *X*, 3, 3)

第6级递归：PRINT-LCS(*b*, *X*, 2, 2)　　返回后输出　**print C**

第7级递归：PRINT-LCS(*b*, *X*, 2, 1)

第8级递归：PRINT-LCS(*b*, *X*, 1, 0)　　返回后输出　**print B**

开始返回

7、算法的改进

（1）可以**去掉表** b，直接基于 c 求LCS。

思考：如何做到这一点？有何改进？

（2）由于算法中计算 $c[i,j]$ 时仅需**两行的数据**：正在被计算的一行和上面的一行。所以可以仅保存这样的两行数据即可，而不用保存所有mn个数据，从而表 c 只包含 $2*min(m, n)$项，另需O(1)的额外空间，即可完成整个计算。

思考：如何做到这一点？

$$c[i,j]=\begin{cases}0 & \text{如果}i=0\text{或}j=0\\ c[i-1,j-1]+1 & \text{如果}i,j>0\text{且}x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果}i,j>0\text{且}x_i\neq y_j\end{cases}$$

15.5 最优二叉搜索树

1、问题的引入

（1）**二叉搜索树**（又称为**二分检索树、二叉排序树**等）

二叉搜索树是一种“**有序**”二元树，它或者为空；或者每个结点都含有一个可以比较大小的数据元素（***key*** 元素），并有：

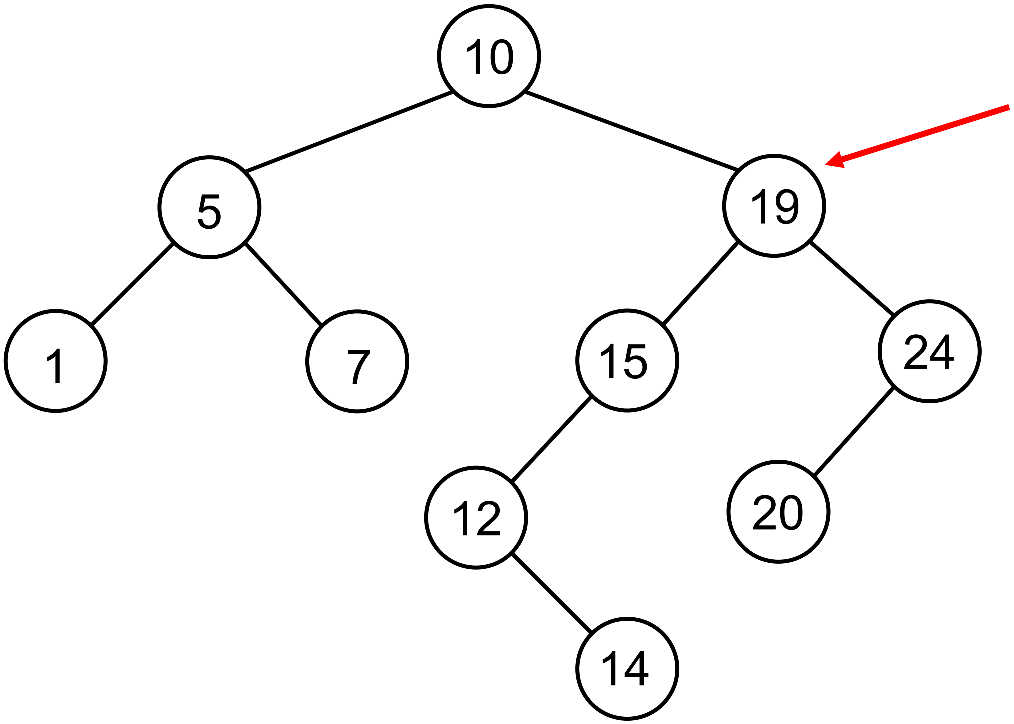
- 左子树所有结点的元素比根结点中的元素小；
- 右子树所有结点的元素比根结点中的元素大；
- 左子树和右子树也是二叉搜索树（**递归定义**）。

注：这里假设**所有元素互异。**

实际应用：二叉搜索树通常是为了**查找运算**而构造的一种**查找结构**。结点代表**数据记录**，结点中可以比较大小的数据元素是数据记录的“**关键字**”。

为方便起见，通常**以关键字代表数据记录**，因此也将一棵二叉搜索树 T 称为是关于**一组关键字 K** 的二叉搜索树，其中 $K=(k_1, k_2, \ldots, k_n)$。

如：一棵由10个关键字构造的**二叉搜索树**如下。



结点内的数字代表结点**关键字的值**。

基本结构特征：

- **"有序"**
- **左小右大**

回顾：对一棵**二叉搜索树**进行**中序遍历**，可以得到关键字的**递增有序序列**，如：**1 5 7 10 12 14 15 19 20 24**

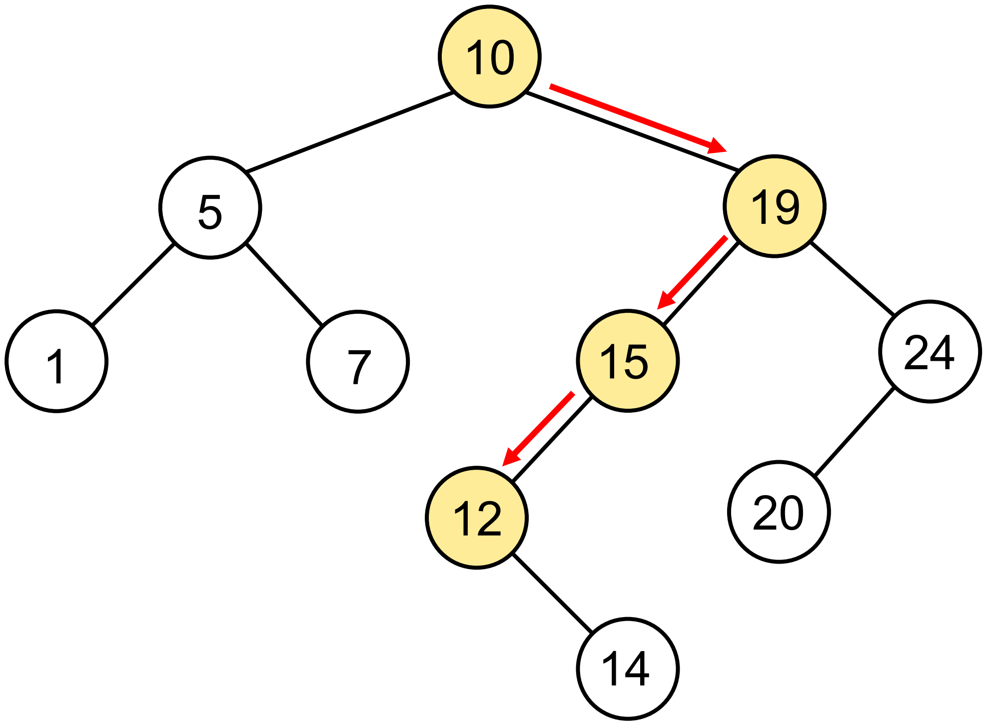
（2）二叉搜索树的基本查找算法

SEARCH(*T*, *k*)

// 在二叉搜索树 *T* 中**查找** *k*。

```
t ← T                      // t 是当前待查结点，将 T 赋给 t，从根结点 T 开始查找
while t ≠ Φ do             // t = Φ 表示查找失败
    case
        : k < t.key : t ← t.lchild      //若 k 小于t，下一步在 t 的左子树中继续查找//
        : k = t.key : return t          //若 k 等于t，返回查找结果
        : k > t.key : t ← t.rchild      //若 k 小于t，下一步在 t 的右子树中继续查找//
    endcase
repeat
end SEARCH
```

如：在上述二叉搜索树中查找 $k = \mathbf{12}$。



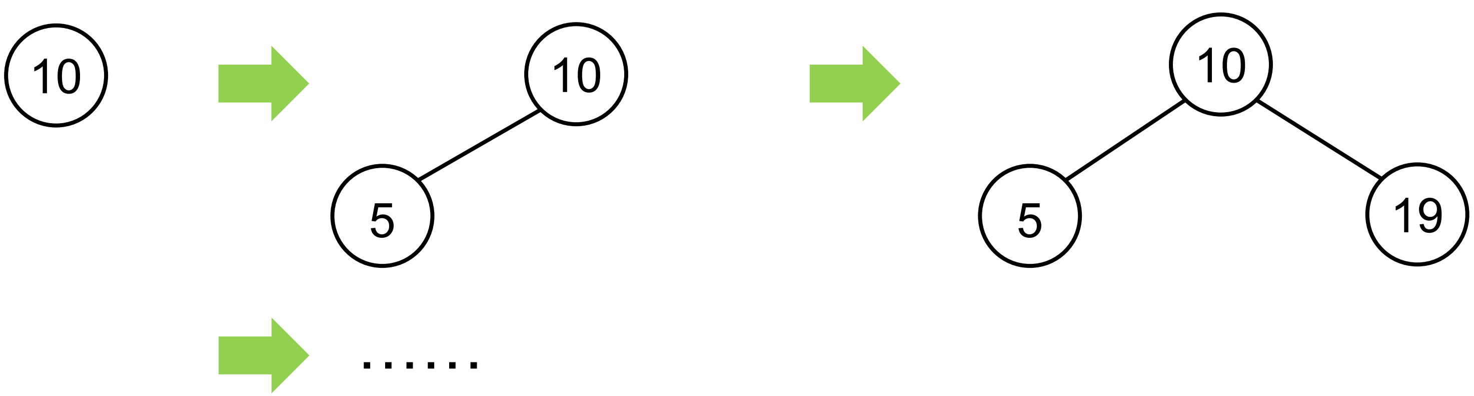
- 从根结点开始，依次和**10**、**19**、**15**、**12**四个结点（关键字）比较，**查找成功**。

（3）二叉搜索树的生成

对于给定的一个关键字集合 $K = (k_1, k_2, \ldots, k_n)$，构造一棵关于 K 的二叉搜索树的一个**朴素算法**是：

以 k_1 为第一个结点并作为二叉搜索树的**根结点** T，然后将其它关键字依次插入到 T 中：若 k_i 小于等于 T，则将 k_i 插入到 T 的左子树中，反之插入到 T 的右子树中；在子树中也按同样的规则找插入点。

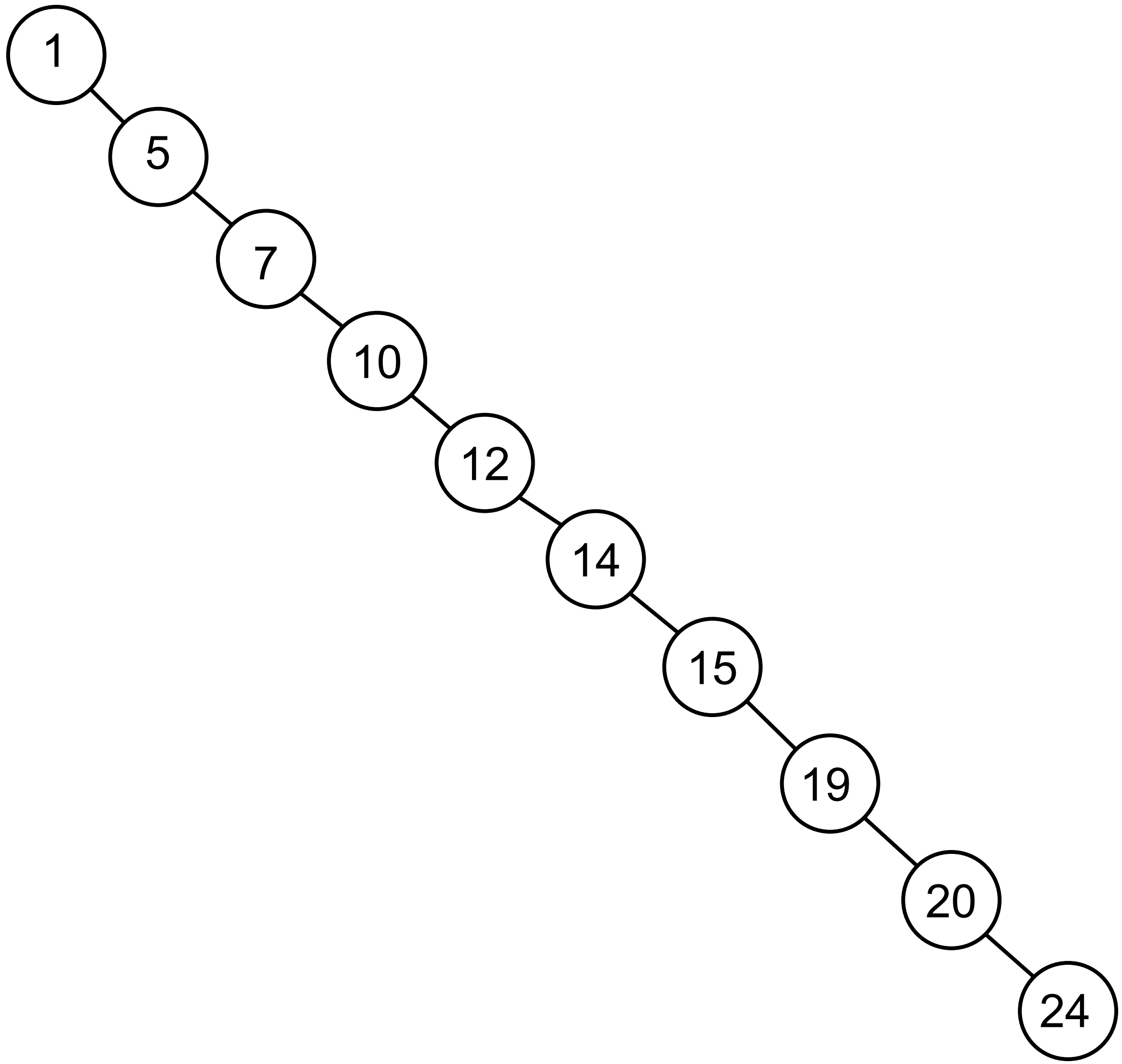
如关键字集合（10, 5, 19, 1, 7, 15, 24, 12, 20, 14）：



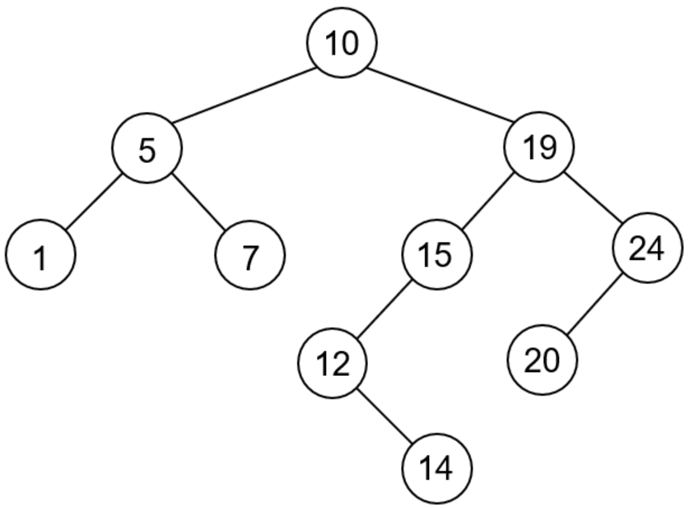
……

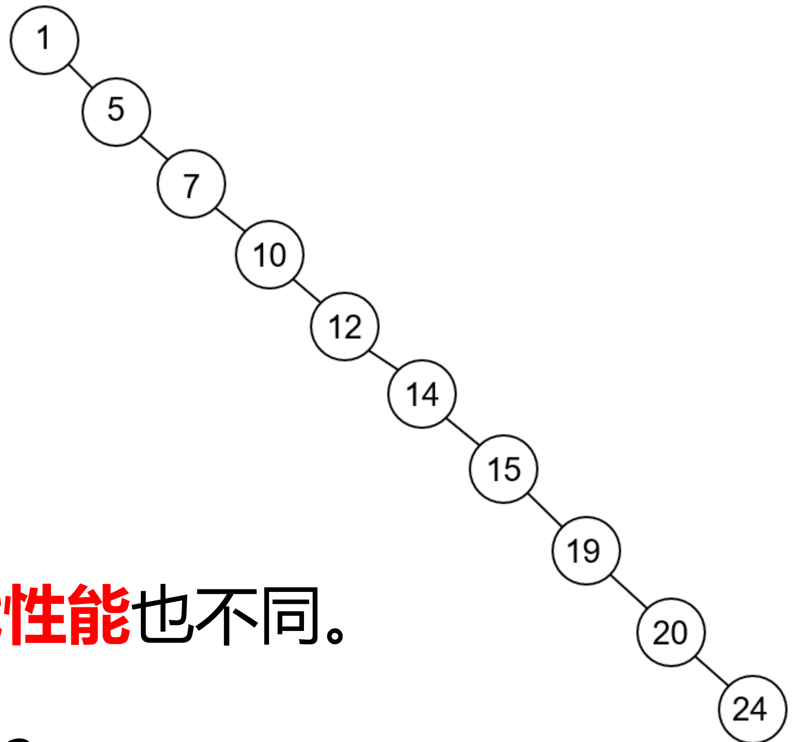
注：如果**关键字集合有序**，则上述**朴素算法**将生成一棵**斜树**。

如：对序列 <1,5,7,10,12,14,15,19,20,24>得到的二叉搜索树如下：



对同一个关键字集合，以不同的关键字作为**树根**或**子树的根**，会得到形态不同的二叉搜索树，如：





另外，不同的二叉搜索树，**查找性能**也不同。

如何评价一棵二叉搜索树的性能?

哪种形态的二叉搜索树**性能最好**?

（4）二叉搜索树的性能评估

不失一般性，后续讨论中假设**初始关键字集合**是一个**有序序列**，即 $K = <k_1, k_2, \ldots, k_n>$ **且** $k_1 < k_2 < \ldots < k_n$。对 K 构造的二分搜索树记为 T 。

根据搜索算法，在 T 中查找一个指定关键字 k 的过程是从树根开始，按照“**左小右大**”的方式依次与一些结点进行比较，直到找到或找不到。

如果找到，则称为**成功查找**，否则称为**失败查找**。

显然，**成功查找的情况有 n 种**，是 k 恰好等于 n 个关键字其中之一的情形。而**失败查找的情况有 n+1 种**。

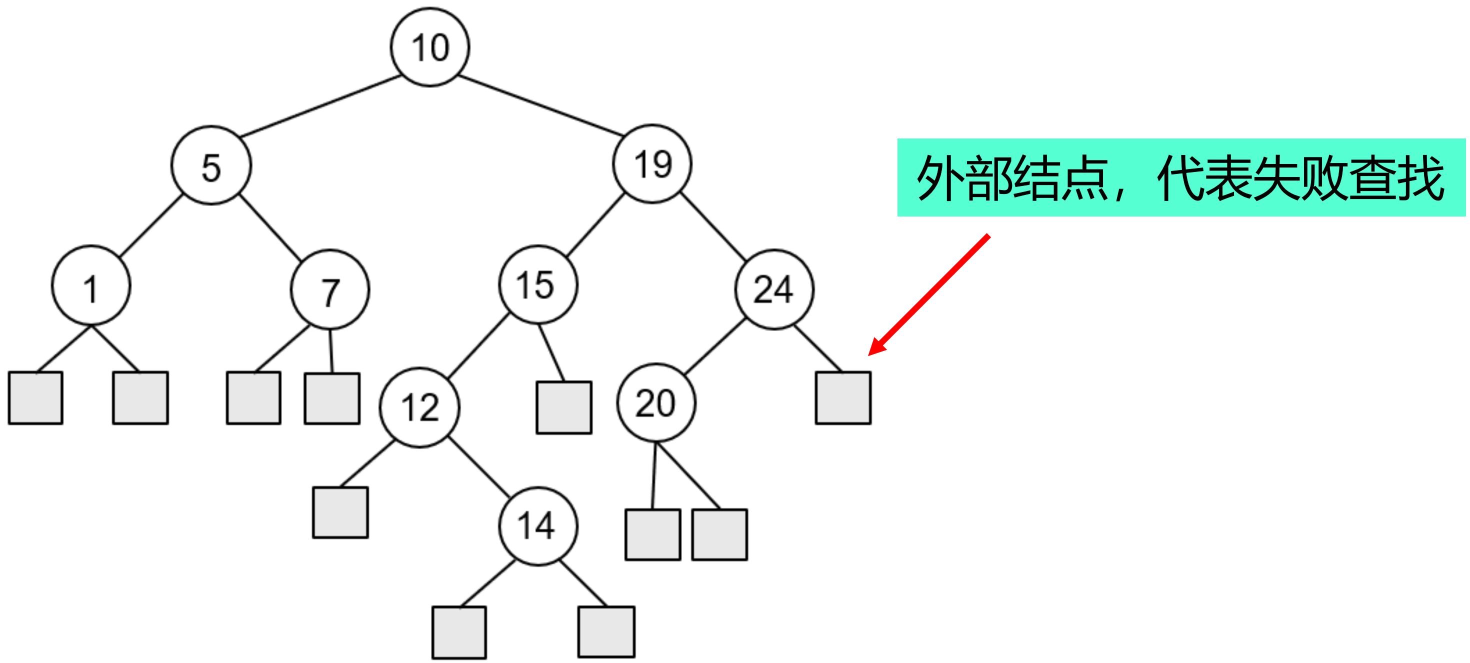
- **不成功查找是出现以下情形的时候：**

$k < k_1$、或者 $k > k_n$、或者 $k_i < k < k_{i+1}$（$1 \le i < n$）。

即 k 落在**比最小关键字小**或**比最大关键字大**或**恰好处于两个相邻的关键字之间**的**区间**。这样的区间共有 $n+1$ 个，所以失败查找的情形共有 $n+1$ 种。

注：失败查找的情形不是指 k 取哪个具体值，而是用"**取值区间**"标识，这样的区间有 $n+1$ 个。

为了在二叉搜索树中表示不成功查找，在原二叉搜索树中增加**外部结点**，如图所示。

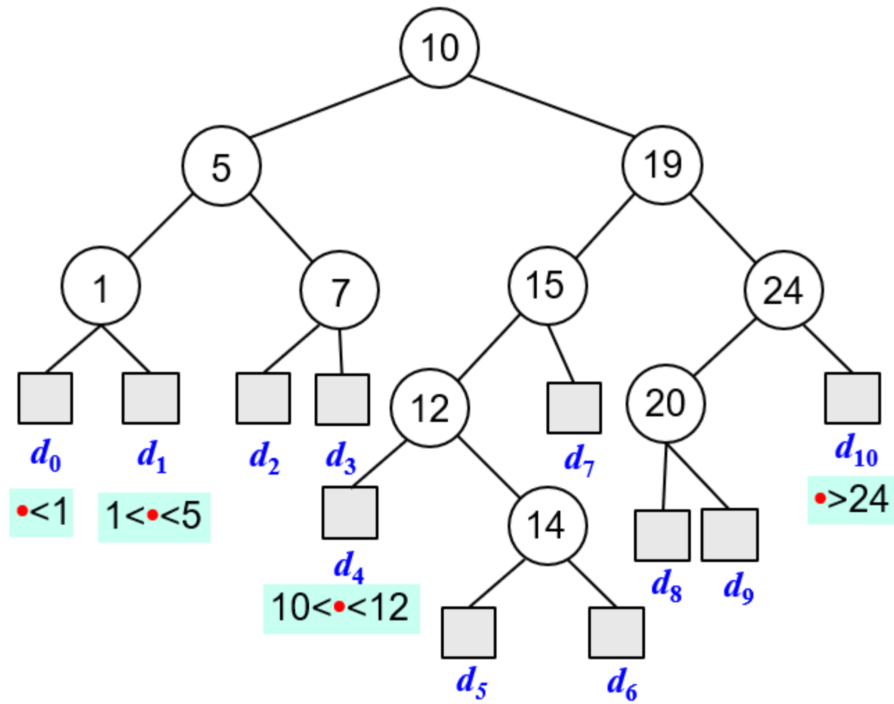


在改造后的二叉树中，外部结点都是**叶子结点**，处于原树中**有空左/右子树的结点**下方，是这些结点的儿子结点，恰好有 n+1 个，分别对应 n+1 个失败查找的区间。

回顾二叉树的基本性质：有n个内结点的二叉树中恰好有n+1个外部结点。

伪关键字：为外部结点引入**伪关键字**，记为 d_i，$0 \le i \le n$。其中，d_0 表示“$\bullet < k_1$”的区间，d_n 表示“$\bullet > k_n$”的区间，其余的 d_i 分别表示“$k_i < \bullet < k_{i+1}$”的各个区间。

如：



为描述方便，这里我们也称当“$k = d_i$”时查找失败。

(注：这里“$k = d_i$”仅代表 k 落入 d_i 代表的区间，并不是真正的值相等)

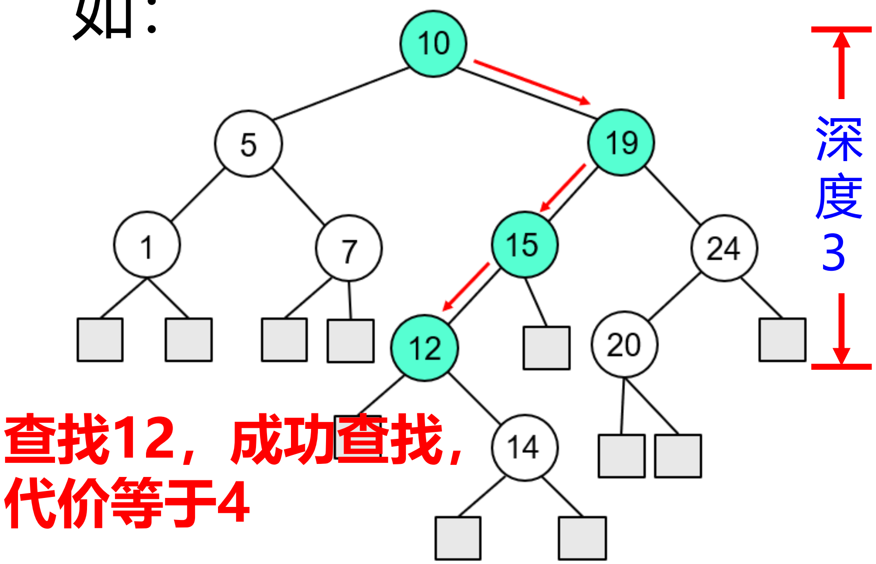
综上所述，有

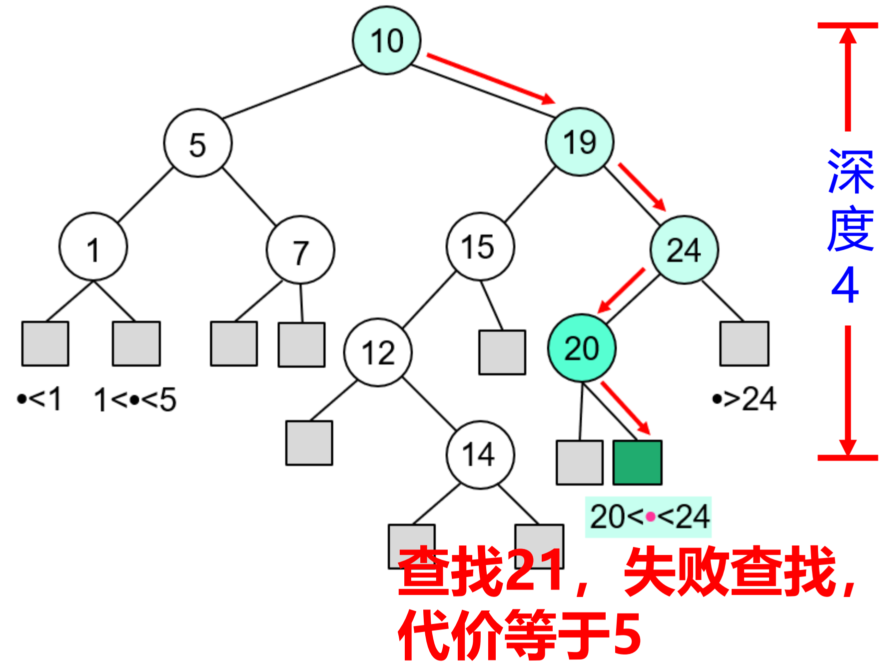
- **内结点**：关键字 k_i 对应的结点，代表**成功查找**的情况， 用圆形框表示，共有 n 个；
- **外结点**：伪关键字 d_i 对应的结点，代表**失败查找**的情况， 用矩形框表示，共有 $n+1$ 个。
- 总共有 $2n+1$ 个结点。

二叉搜索树的性能定义

- **结点的查找代价：** **查找结点 *i* 的代价**定义为一次查找中**访问到的结点数**，等于结点在树中的**深度+1**。

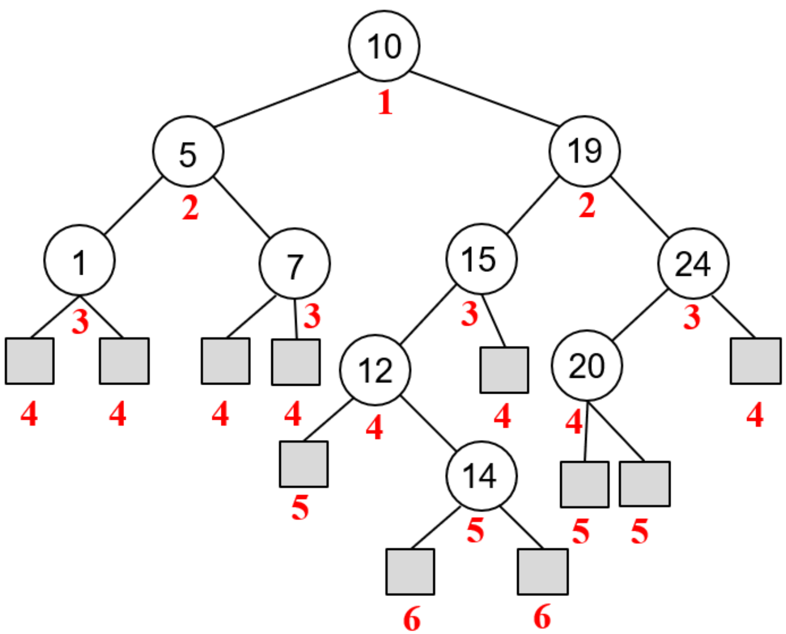
如：





结点的深度：**根结点**的深度为 0，其余结点的深度等于其**父结点的深度**+1。记为 $depth_T(i)$。

- 成功查找的最好情况：**根结点**，代价为 $O(1)$。
- 成功查找的最坏情况：离根最远的内结点。
- 成功查找的平均情况：$\sum_{1\le i\le n}(depth_T(k_i)+1)\,/n$ **(算术平均)**
- 失败查找的最好情况：离根最近的外部结点。
- 失败查找的最坏情况：离根最远的外部结点。
- 失败查找的平均情况：$\sum_{0\le i\le n}(depth_T(d_i)+1)\,/(n+1)$ **(算术平均)**



成功查找的最小代价：1

成功查找的最大代价：5

成功查找的平均代价：3

失败查找的最小代价：4

失败查找的最大代价：6

失败查找的平均代价：4.64

◆ 整棵树的性能评价

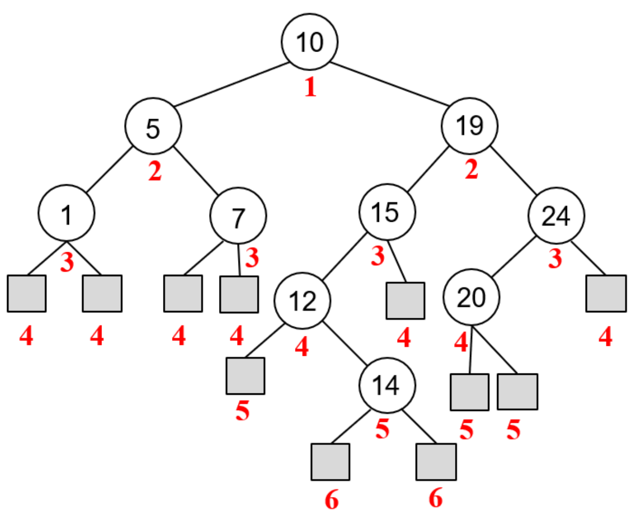
一棵**二叉搜索树的查找性能**用树中**所有结点（包括内结点和外结点）的平均查找代价**表示。有两种情况：

- **等概率查询** —— 所有结点有**相同的被查找概率**（$1/(2n+1)$）。

此时，一般计算结点的**算术平均查找代价**，即：

$$\left(\sum_{1\le i\le n}(depth_T(k_i)+1)+\sum_{0\le i\le n}(depth_T(d_i)+1)\right)\Big/(2n+1)$$

如：



算术平均查找代价

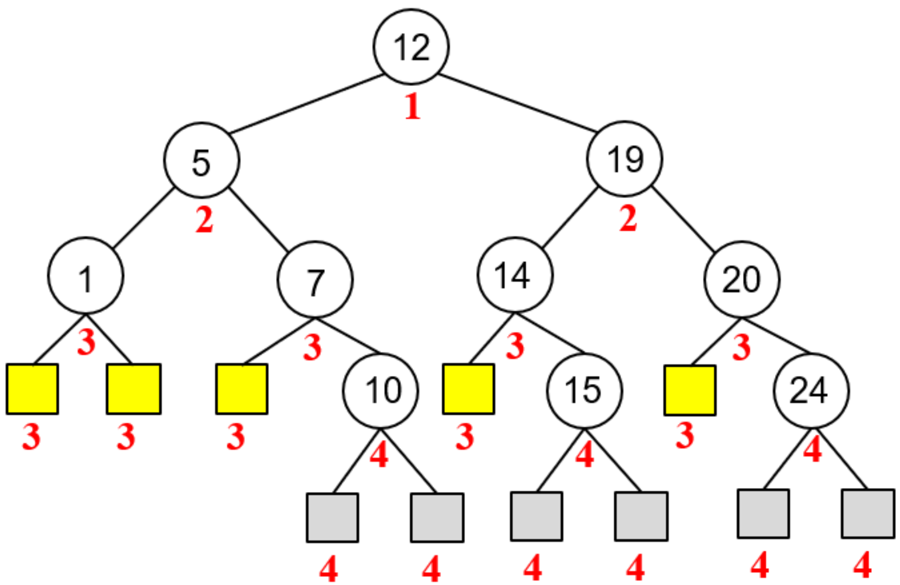
= (1+2+2+3+3+3+3+4+4+5 +

4+4+4+4+4+4+5+5+5+6+6) / 21

=3.86

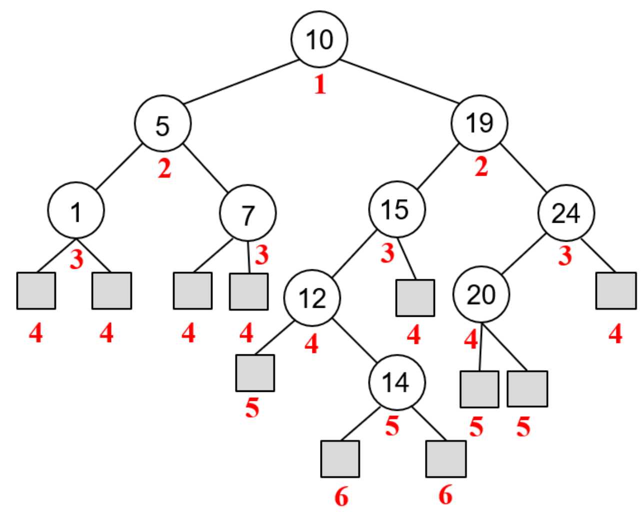
注：对于关键字序列 $K = <k_1, k_2, \ldots, k_n>$（$k_1 < k_2 < \ldots < k_n$），它的**算术平均查找代价最小**的二叉搜索树是一棵 “**近似**” **二分查找树**形态的二叉搜索树。

如对关键字序列<1, 5, 7, 10, 12, 14, 15, 19, 20, 24>。



平均查询代价最小=3.24

对比：3.86



不等概率查询：

——**每个结点被查找的概率不一样。**

一个应用场景：据统计，英文字母中 e 是最常用的字母，而 z 是最不常用的字母。如果以**英文字母**为关键字构造二叉搜索树，并对一篇文章中出现的**字符**进行查找，则 e 结点是最经常被查找的结点，而 z 结点就是最不经常被查找的结点。

那么如何构造一棵**关于英文字母的二叉搜索树**，才能使这种场景下二叉搜索树的性能最好呢?

直觉：对一个结点而言，**离根越近，查找代价越小**。因此常用的关键字应离根尽量近，而不常用的关键字可以离根远一些。

这样被频繁查找的关键字每次查找的代价都尽量小，而查找不频繁的关键字代价高一点也不会带来大影响，“**综合起来**”整棵树的查找性能就好。

但**是不是以查找频率最高的关键字当作树根就行?**

——因还有其它元素，要综合考虑各种情况，简单地以查找频率高的关键字当作树根并不一定能获得最好性能（树的“左小右大”的形态也不能这样简单维持）。（**自行举例并观察思考**）

在有**概率**的情况下，怎么构造二叉搜索树才能获得最优性能呢?

2、最优二叉搜索树

考虑带有概率的查找：

设已知关键字序列 $K = <k_1, k_2, \ldots, k_n>$：

- **已排序**，不失一般性，设有 $k_1 < k_2 < \ldots < k_n$ 。
- 已知每个 k_i 的查找概率，记为 p_i，$1 \le i \le n$，**成功查找概率**；

  以及 $d_0 \sim d_n$ 的查找概率，记为 q_i，$0 \le i \le n$，**失败查找概率**。
- 综合考虑所有结点的查找情况，故有

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

—— 以此来构造一棵“**性能最优**”的二叉搜索树。

二叉搜索树的期望查找代价：

在引入**概率**的前提下，二叉搜索树的性能用其**期望搜索代价**表示，即：

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

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

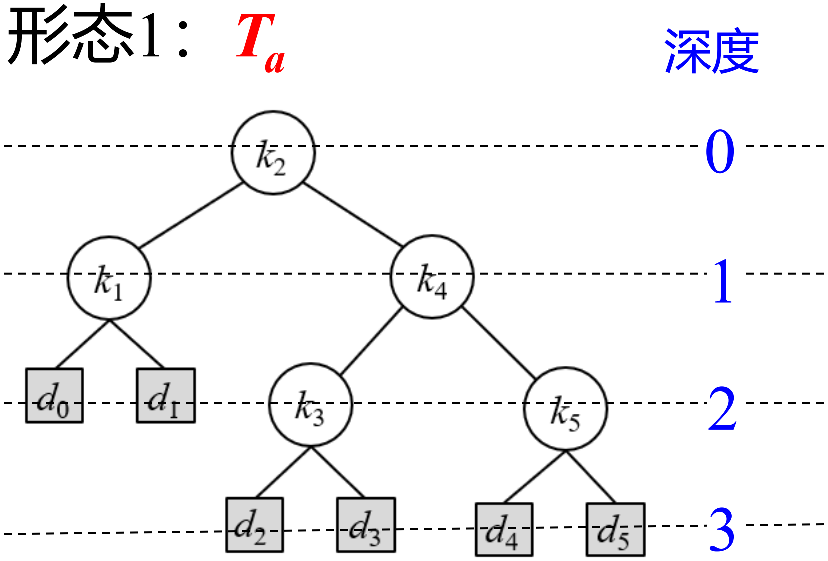
$$\boxed{\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1\ .}$$

例：设有 $n = 5$ 个关键字的集合，每个 k_i 的概率 p_i 和 d_i 的概率 q_i 如表所示：

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

基于该集合构造一棵二叉搜索树。

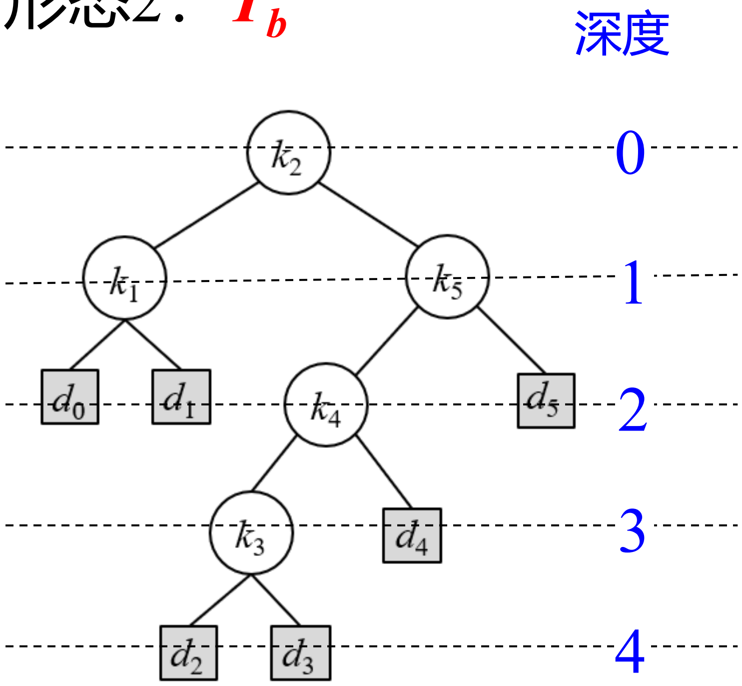
形态1：T_a



$$
\begin{aligned}
E[T_a] &= (1+1)*0.15 + (0+1)*0.10 \\
&+ (2+1)*0.05 + (1+1)*0.10 \\
&+ (2+1)*0.20 + (2+1)*0.05 \\
&+ (2+1)*0.10 + (3+1)*0.05 \\
&+ (3+1)*0.05 + (3+1)*0.05 \\
&+ (3+1)*0.10 \\
&= \mathbf{2.80}
\end{aligned}
$$

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

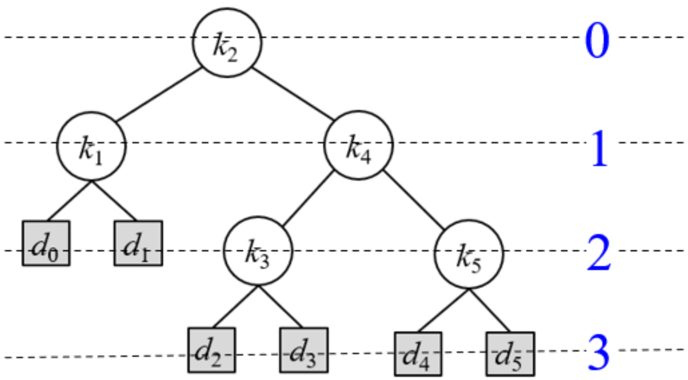
形态2：T_b



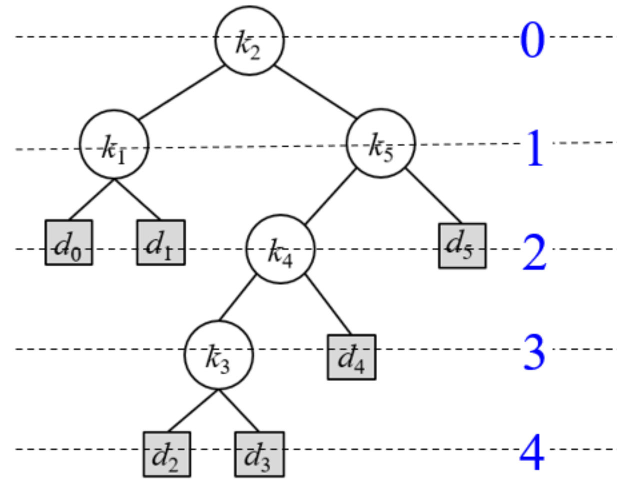
$$
\begin{aligned}
E[T_b] &= (1+1)*0.15 + (0+1)*0.10 \\
&+ (3+1)*0.05 + (2+1)*0.10 \\
&+ (1+1)*0.20 + (2+1)*0.05 \\
&+ (2+1)*0.10 + (4+1)*0.05 \\
&+ (4+1)*0.05 + (3+1)*0.05 \\
&+ (2+1)*0.10 \\
&= \mathbf{2.75}
\end{aligned}
$$

注：对一个关键字集合的不同形态的二叉树，**结点的概率都是一样的**，区别仅在于形态不同造成结点在树中的**深度不一样**。

形态1：T_a 深度



形态2：T_b 深度



T_a的期望代价为**2.80**

T_b的期望代价为**2.75（最优）**

- 即使对同一个关键字集合，不同形态的二叉搜索树的期望搜索代价也有不同。
- 看起来“**深**”的树，期望代价未必高。

—— “**精心设计**”才能构造一棵性能最优的二叉搜索树。

最优二叉搜索树：对于给定的关键字及其概率集合，**期望搜索代价最小**的二叉搜索树称为该关键字及其概率集合下的**最优二叉搜索树**。

怎么构造一个关键字集合的最优二叉搜索树呢?

- 树根不一定是概率最高的关键字；
- 树也不一定是最矮的树；
- 但该树的**期望搜索代价必须是最小的**。

可以枚举，但求解效率低下

3、用动态规划策略构造最优二叉搜索树

（1）证明最优二叉搜索树的最优子结构

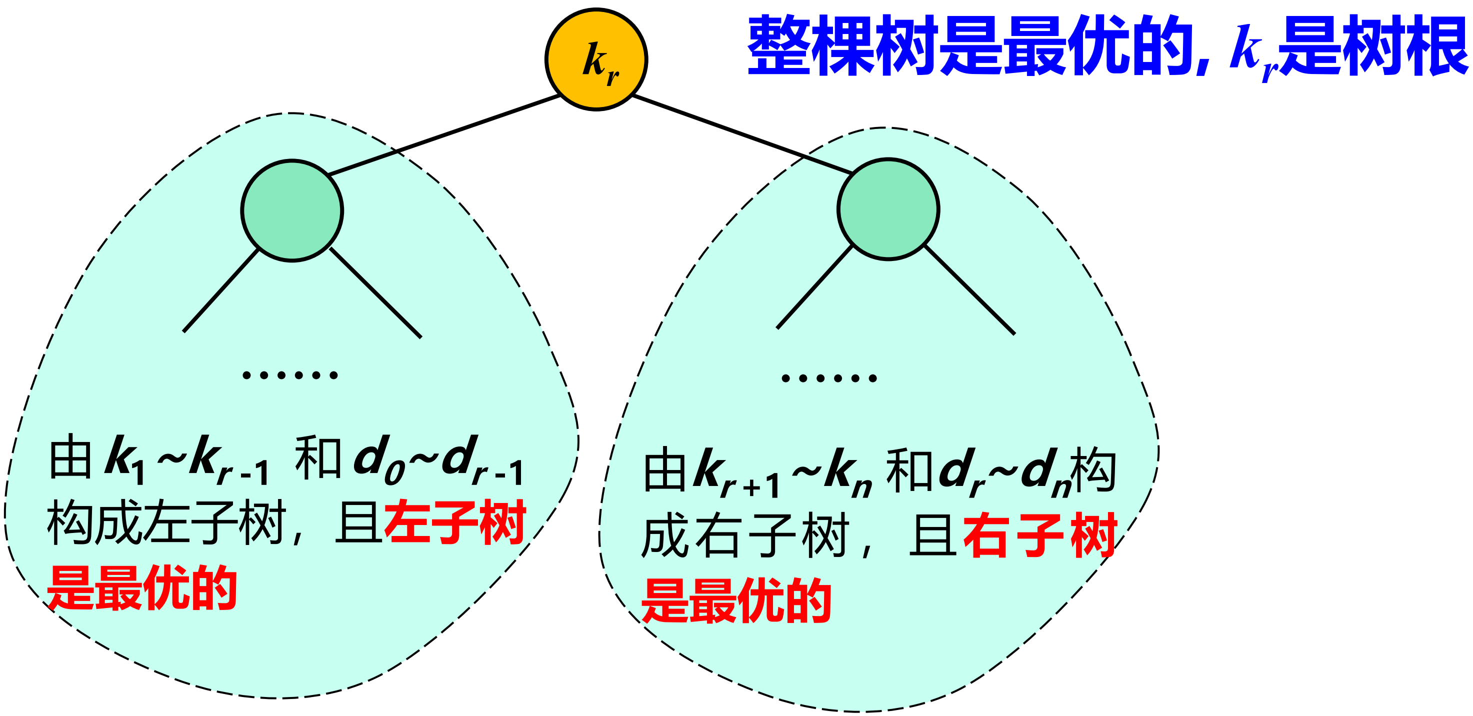
“最优二叉搜索树的最优子结构”形式化描述为：如果 T 是一棵关于关键字 $k_1, \ldots, k_n$ 和伪关键字 $d_0, \ldots, d_n$ 的**最优二叉搜索树**，则 T 中的一棵包含关键字 $k_i, \ldots, k_j$ 的子树 T' 必然是关于关键字 $k_i, \ldots, k_j$ 及伪关键字 $d_{i-1}, \ldots, d_j$ 子问题的一棵最优二叉搜索子树。

证明：用**剪切-粘贴法**证明

对关键字 $k_i, \ldots, k_j$ 和伪关键字 $d_{i-1}, \ldots, d_j$，如果存在子树 T''，其期望搜索代价比 T' 更低，那么将 T' 从 T 中删除，并将 T'' 粘贴到相应位置上，则**可得到一棵比 T 期望搜索代价更低的二叉搜索树**，与 T 是最优的假设矛盾，所以 T' 必定是最优二叉搜索子树。

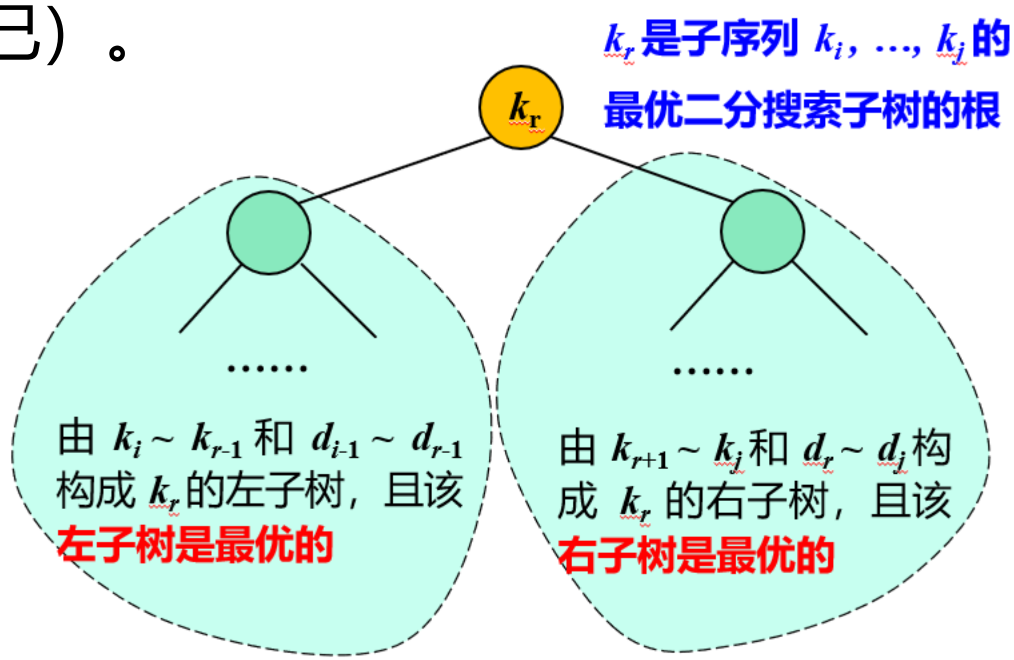
(2) 构造状态转移方程

分析： 根据**最优子结构性**，**任何最优二叉搜索树都是由最优二叉搜索子树**组合而成的。即若该树是一棵最优二叉搜索树，则根的左、右子树必是最优二叉搜索子树。根和左右子树的根连接起来后构成完整的最优二叉搜索树。如下所示：



上述最优子结构性质可以推广到任意子树范围内：

对于任意一个关键字子序列 k_i，…，k_j（相应地，该范围内的伪关键字是 d_{i-1}，…，d_j），若其最优二叉搜索子树以关键字 k_r 为根（$i \le r \le j$），则 k_r 的左子树中包含的关键字是 k_i，…，k_{r-1} 及伪关键字 d_{i-1}，…，d_{r-1}，右子树中包含的关键字是 k_{r+1}，…，k_j 及伪关键字 d_r，…，d_j。并且 k_r **的左子树和右子树也必是最优二叉搜索子树**（更小一点而已）。



如何构造一个子序列 $k_i, \ldots, k_j$ 的最优二叉搜索子树（包括最终原始序列 $k_1, \ldots, k_n$ 的最优二叉搜索树）呢?

构造二叉树的关键是找**树根**。但由于每个关键字都有可能是树/子树的根（注：元素大小关系是确定的，但赋予不同的 p_i 和 d_i，实际构造时，每个结点都有可能是根），所以树/子树的根的位置不是固定的，必须根据实际已知条件计算。

一个策略是：

将 $k_i, \ldots, k_j$ 中的每个关键字都当作一次子树根，尝试构造一棵"**以该关键字为根的最优二叉搜索树**"（共 $j-i+1$ 棵），然后比较它们的期望代价，最后找出期望代价最小的那棵，就是子序列 $k_i, \ldots, k_j$ 最终的最优二叉搜索子树，并且其根也找到了。

计算的实施：

为此，定义 $e[i, j]$ 为包含关键字 $k_i, \ldots, k_j$ 和伪关键字 $d_{i-1}, \ldots, d_j$ 的**最优二叉搜索子树的期望搜索代价**。

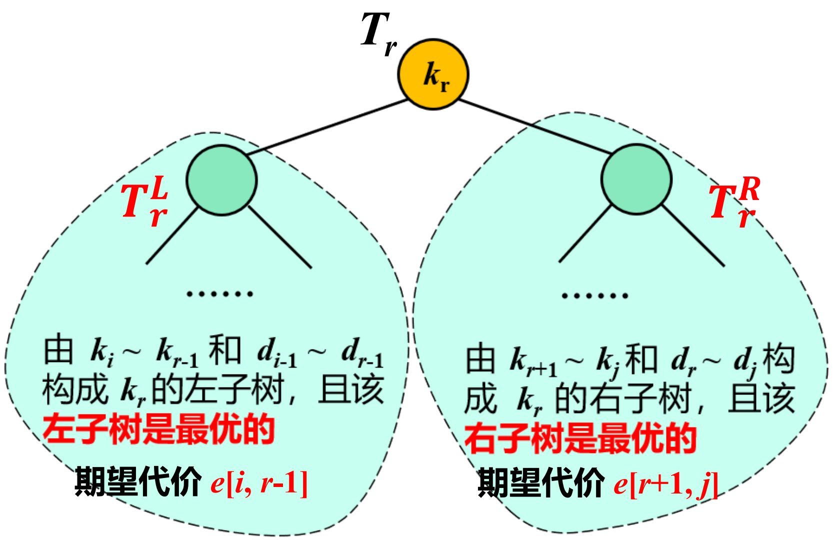
（注：$e[1, n]$ 就是原始问题、相对关键字 $k_1, \ldots, k_n$ 的最优二叉搜索树的期望搜索代价）

分析：

如前所述，若尝试以其中的关键字 k_r $(i \le r \le j)$ 为根构造一棵 **"以关键字 k_r 为根的最优二叉搜索子树"**，记为 T_r，则：

- T_r 的左子树：记为T_r^L，包含关键字 $k_i, \ldots, k_{r-1}$ 及伪关键字 $d_{i-1}, \ldots, d_{r-1}$；
- T_r 的右子树：记为T_r^R，包含关键字 $k_{r+1}, \ldots, k_j$ 及伪关键字 $d_r, \ldots, d_j$。

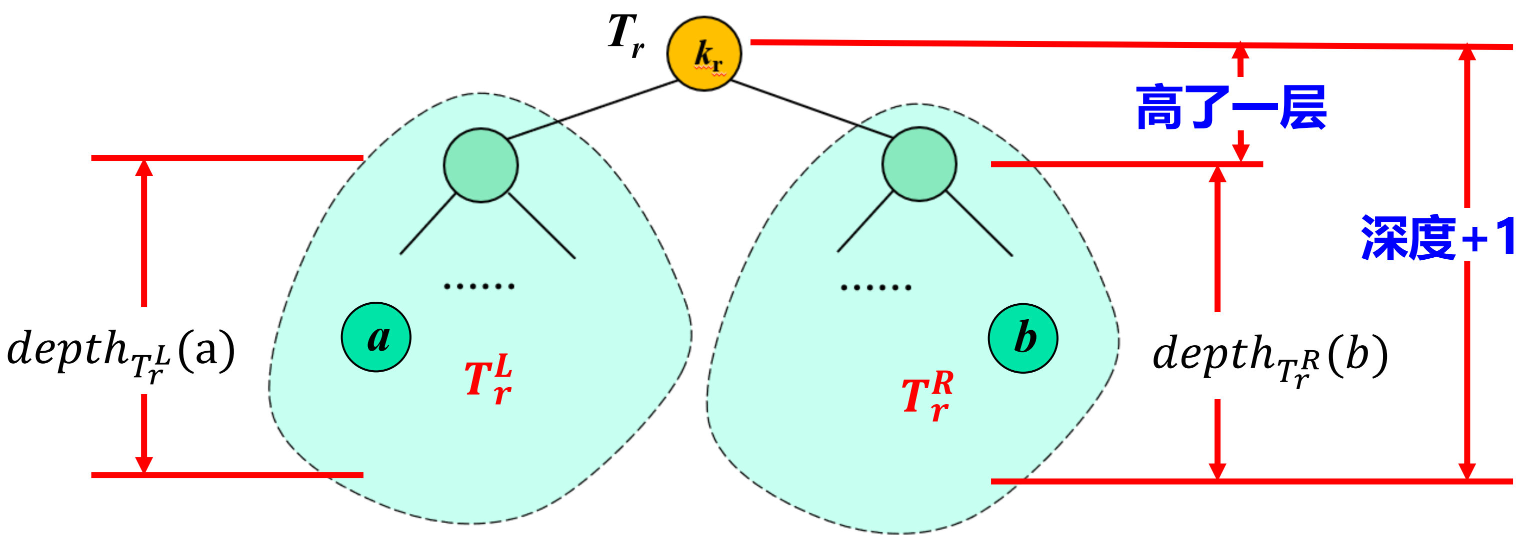
并且根据二叉搜索的**最优子结构性**，要想使 T_r 的期望搜索代价最小，其左、右子树 T_r^L **和** T_r^R **都必须是最优二叉搜索子树**。



尝试以 k_r 为根构造一棵期望代价小的二叉搜索子树

把 T_r^L 和 T_r^R 都看作是**更小的子问题**，它们“**如果可以**”在此之前求解出来，则 T_r^L 期望搜索代价就是 $e[i, r-1]$， T_r^R 的期望搜索代价就是 $e[r+1, j]$。

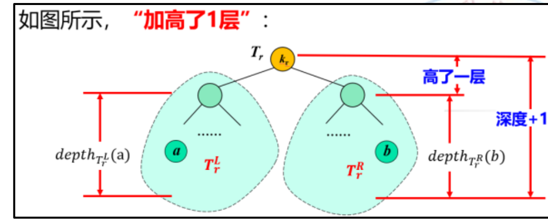
再如图所示，T_r 是在 T_r^L 和 T_r^R 的基础上，上面加了一个根结点 k_r 后得到的——**树“加高了1层”**：



- 对于 T_r^L 中的一个结点 a，若在 T_r^L 中深度为 $depth_{T_r^L}(a)$，则它在 T_r 中的深度 $depth_{T_r}(a)= depth_{T_r^L}(a) + 1$。同理，

- 对于 T_r^R 中的一个结点 b，若在 T_r^R 中深度为$depth_{T_r^R}(b)$，则它在 T_r 中的深度 $depth_{T_r}(b)= depth_{T_r^R}(b) + 1$。

则 T_r 的期望代价和 T_r^L 及 T_r^R 的期望代价之间就有如下关系：



$$E[T_r] = e[i,j] = \sum_{l=i}^{j}(depth_{T_r}(k_l)+1)\cdot p_l + \sum_{l=i-1}^{j}(depth_{T_r}(d_l)+1)\cdot q_l$$

$$= \sum_{l=i}^{r-1} depth_{T_r}(k_l)\cdot p_l + 1\cdot p_r + \sum_{l=r+1}^{j} depth_{T_r}(k_l)\cdot p_l + \sum_{l=i}^{r-1} p_l + \sum_{l=r+1}^{j} p_l$$

$$+ \sum_{l=i-1}^{r-1} depth_{T_r}(d_l)\cdot q_l + \sum_{l=r}^{j} depth_{T_r}(d_l)\cdot q_l + \sum_{l=i-1}^{r-1} q_l + \sum_{l=r}^{j} q_l$$

左子树

$$= \boxed{\sum_{l=i}^{r-1}(depth_{T_r^L}(k_l)+1)\cdot p_l + \sum_{l=i-1}^{r-1}(depth_{T_r^L}(d_l)+1)\cdot q_l}$$

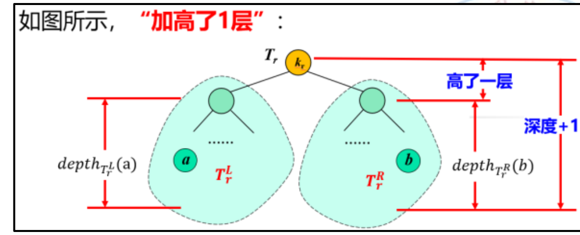
右子树

$$+ \boxed{\sum_{l=r+1}^{j}(depth_{T_r^R}(k_l)+1)\cdot p_l + \sum_{l=r}^{j}(depth_{T_r^R}(d_l)+1)\cdot q_l} + w(i,j)$$

$$= E(T_r^L) + E(T_r^R) + w(i,j)$$

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

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



即： $\color{red}{e[i,j]} = \color{red}{e[i,r-1]} + \color{red}{e[r+1,j]} + \color{blue}{w(i,j)}$

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

注：这样的 r 要在 $[i,\ j]$ 范围内“**测试**”从 i 到 j 的所有取值，以找出“**最小**”期望代价——最优的二分搜索树子树。所以最终得到以下递推关系式。

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

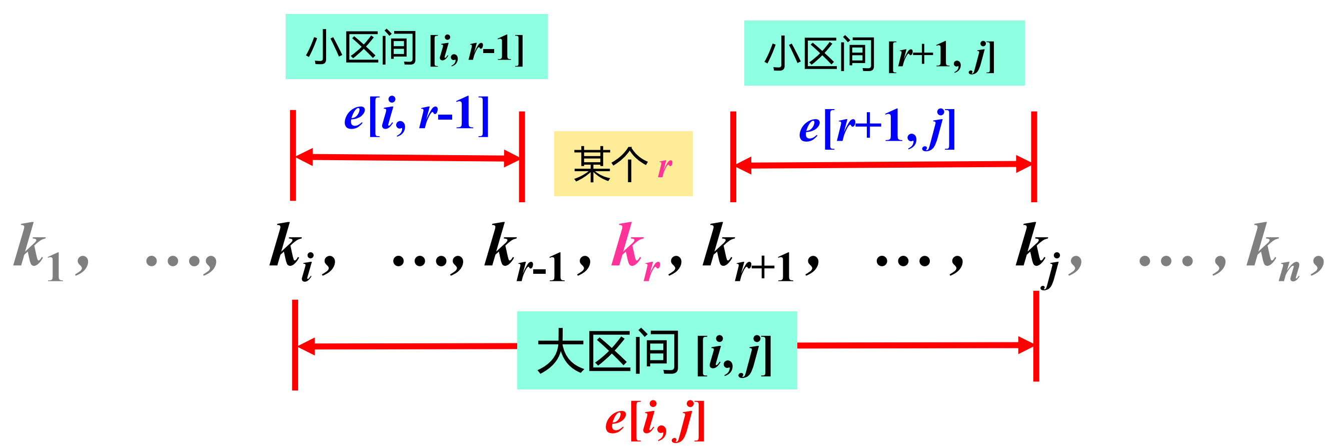
——这就是最优二分搜索树问题的**状态转移方程**。

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

说明：

(1) 计算的顺序：先计算小区间、再计算大区间。

- $e[i, j]$：$[i, j]$ 代表一个区间长度是 $l=j–i+1$ **“大区间”**，
  $e[i, j]$ 是该**大区间**的期望代价。
- r： 由于 $i \le r \le j$，所以 $[i, r\text{-}1]$ 和 $[r+1, j]$ 都是长度小于 l 的
  “小区间”。$e[i, r\text{-}1]$ 和 $e[r+1, j]$ 是这些**小区间**的期望代价。



也就是要先算出**小区间的 e 值**，才能计算**大区间的 e 值**。所以**计算顺序**是**先计算小区间、再计算大区间**。

并且，r 会取到 $i \sim j$ 的每个值，所以随着 r 的不同取值，$[i, r-1]$ 、$[r+1, j]$ 事实上具体代表了长度从 $0 \sim l-1$ 的**众多小区间**。

具体有：$[i, i]$、 $[i, i+1]$、$[i, i+2]$、 …、 $[i, j-1]$，以及

$[i, j]$、 $[i+1, j]$、 …、 $[j-1, j]$、$[j, j]$，

甚至 $[i, i-1]$、 $[j+1, j]$ 这样**长度为 “0”** 的区间。

这些小区间都要被先计算出来——因为在 $e[i, j]$ 的计算中，它们都可能会被用到。

事实上，由于 $1 \le i \le j \le n$，所以随着 i、j 的不同取值，区间表示 $[i, j]$ 将涵盖**区间长度为 $0 \sim n$ 的所有区间。**

其中，

- 区间长度为1的子区间有 n 个：[1, 1]、[2, 2]、[3, 3]、…、$[n, n]$；
- 区间长度为2的子区间有 n-1 个：[1, 2]、[2, 3]、[3, 4]、…、$[n\text{-}1, n]$；
- 区间长度为3的子区间有 n-2 个：[1, 3]、[2, 4]、[3, 5]、…、$[n\text{-}2, n]$；

……

- 区间长度为 n-1 的子区间有 2 个：$[1, n\text{-}1]$、$[2, n]$；
- 区间长度为 n 的子区间有 1 个：$[1, n]$。
- 还有n+1个**长度为“0”**的区间：[1, 0]、[2, 1]、[3, 2]、…、$[n+1, n]$；

它们都要计算、都会用到

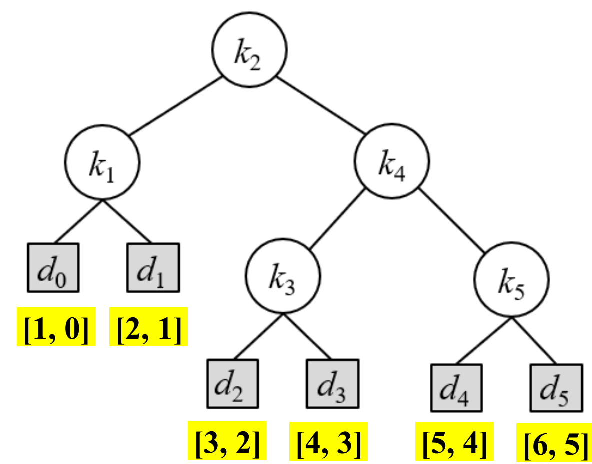
而且是按照区间长度从小到大依次计算，直到 $e[1,n]$

（2）长度为 0 的区间

长度为 0 的区间共有n+1个：[1, 0]、[2, 1] 、 …、 [*n*+1, *n*]。

事实上，“**长度为 0 的区间**”代表“**空**”**关键字子序列**，即不包含任何**有效关键字**的子序列，它们恰好与 n+1 个**失败查找区间**相对应，代表**失败查找**的情形。

故长度为 0 的区间可视为仅包含一个**伪关键字** d_i（$0 \le i \le n$）的子区间，对应一棵“**空**”**二叉搜索子树**，它们的期望代价就是相应的**失败查找的概率** $q_{i\text{-}1}$。



这也恰是**递推关系式边界条件**的含义：

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

综上所述，整个计算过程比较“**繁琐**”：

为了最终计算出 $e[1, n]$，需要从**长度等于0最小区间**开始，依次计算出所有区间的 e 值。这包括：

- n+1个长度为 0 的子区间：有 $e[1,0]$、$e[2,1]$、…,、$e[n+1, n]$

  根据（2）的分析，这些区间的 e 值可以直接给出：

  $e[1,0] = q_0$, $e[2,1] = q_1$, …, $e[n+1, n] = q_n$。

- 然后，**for** $l = 1$ **to** n，计算所有的 $e[i,j]$，其中 $j = i + l - 1$。即各种长度区间的 e 值。
- 至最后，$l = n$ 时，计算出 $e[1,n]$，过程结束。

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

记录子树的根：

$e[i, j]$ 是关键字子序列 $k_i, \ldots, k_j$ 的**最优二叉搜索子树**的最优期望搜索代价。显然 ***min*** 表达式中使 $e[i, j]$ 取得最小值的 r 就是期望获得的**最优二叉搜索子树的根** —— **对应的关键字是** k_r。

为此定义 $root[i, j]$，记录这个能使 $e[i, j]$ 取得最小值的 r。

root的作用： 在求出 $e[1, n]$ 后，$root[1, n]$ 就是最终最优二叉搜索树的根，然后利用 $root$ 的记录**反推**，就可构造出整棵最优二叉搜索树。

造表：

需要定义三个表（都是二维数组）：

① $e[1..n+1, 0..n]$：用于记录 $e[i, j]$ 的值。

② $root[1..n, 1..n]$：用于记录所有最优二叉搜索子树的**根结点**。

其中 $root[1, n]$ 是最终最优二叉搜索树的根。

③ $w[1..n+1, 0..n]$：用于保存子树的结点概率之和。

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

且有递推关系：$w[i, j] = w[i, j\text{-}1] + p_j + q_j$

这样，每个$w[i, j]$的计算时间仅为 $\Theta(1)$

综上所述，下面给出过程 **OPTIMAL-BST**，利用概率列表 p 和 q，对 n 个关键字计算最优二叉搜索树的表 e 和 $root$。

OPTIMAL-BST(*p*, *q*, *n*) // 利用概率表 *p* 和 *q* 计算 *n* 个关键字的最优二叉搜索树的表 *e* 和 *root*

注：关键字按 $k_1 < k_2 < \ldots < k_n$ 有序

```
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   // 构造区间[i, j]
        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      // 找最小的 e[i, j]
                root[i, j] = r   // 记录根
return e and root
```

边界值

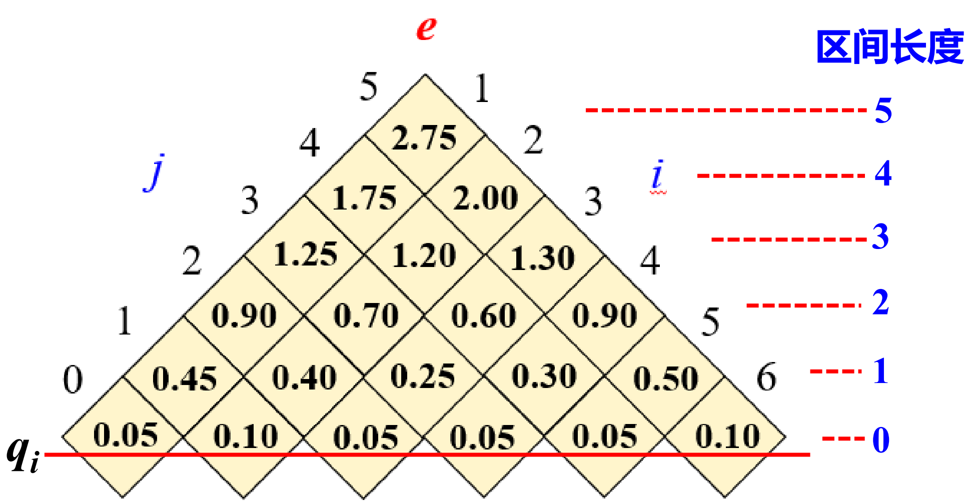
对所有长度的区间计算 *e* 和 *root*

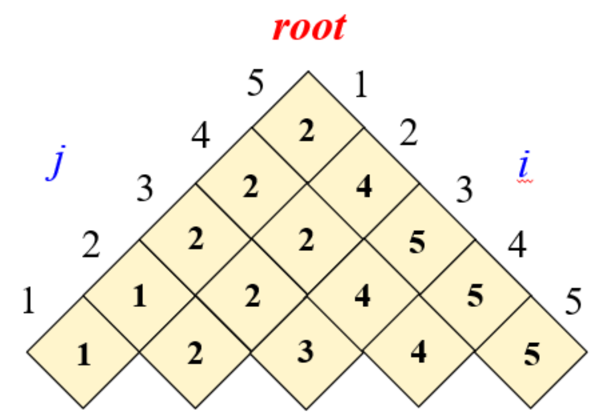
该过程是一个三重循环结构，时间复杂度为 $\Theta(n^3)$

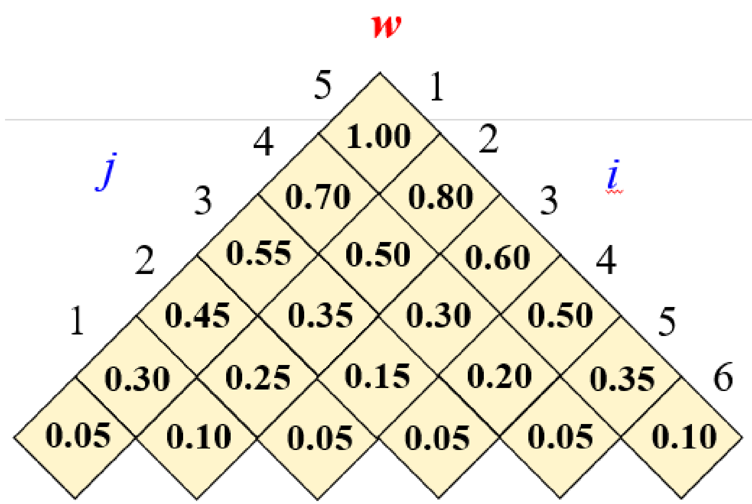
例：n=5，求关于

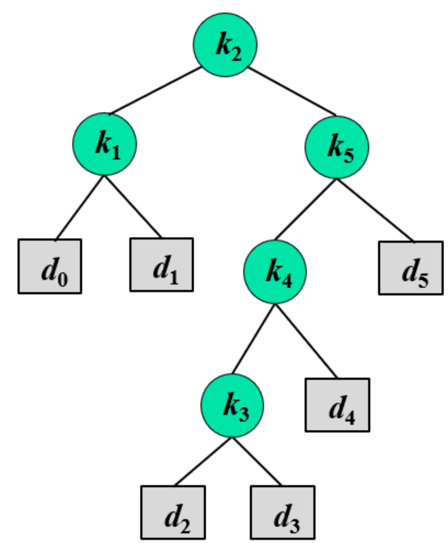
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

的最优二分搜索树





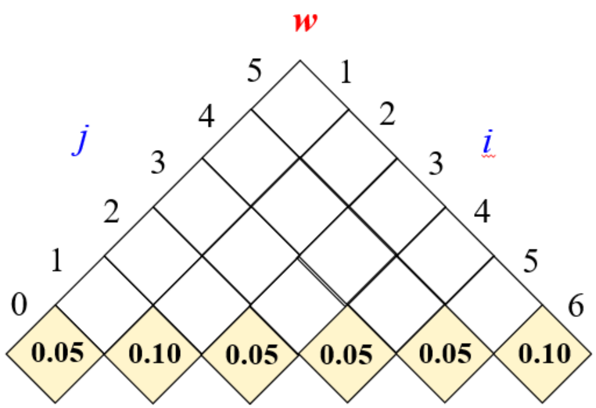


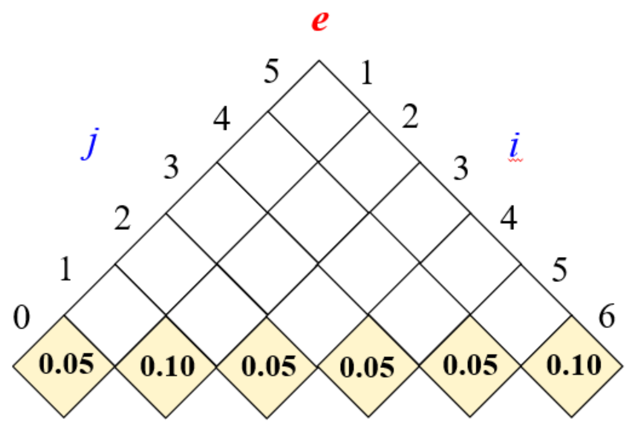


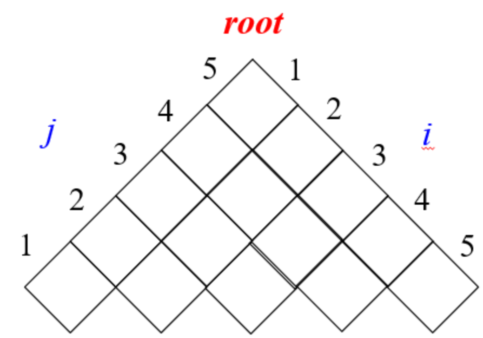
具体计算过程：

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

① 区间长度=0： [1,0]、 [2,1]、 [3,2]、 [4,3]、 [5,4]、 [6,5] 六个







$w[1,0]=q_0$

$w[2,1]=q_1$

$w[3,2]=q_2$

$w[4,3]=q_3$

$w[5,4]=q_4$

$w[5,6]=q_5$

$e[1,0]=q_0$

$e[2,1]=q_1$

$e[3,2]=q_2$

$e[4,3]=q_3$

$e[5,4]=q_4$

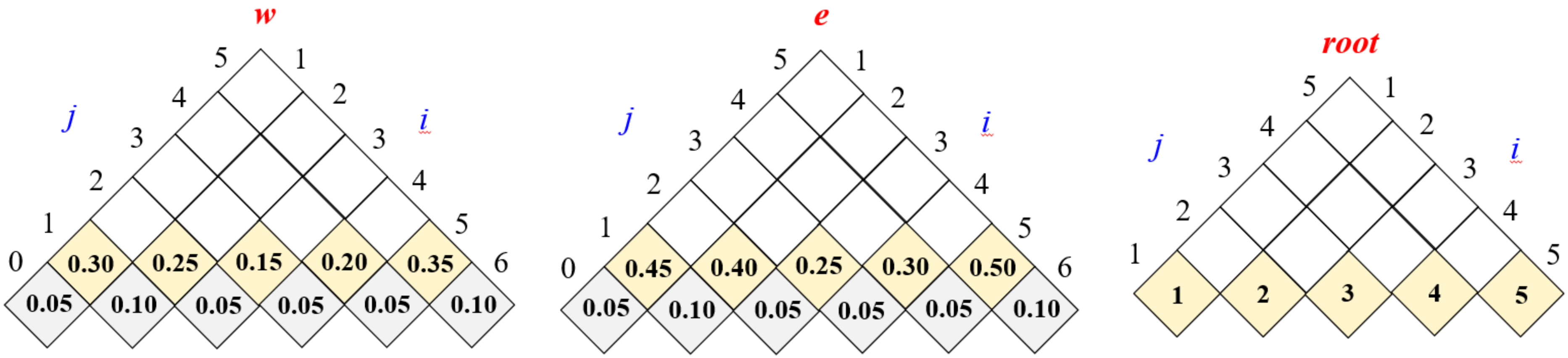
$e[5,6]=q_5$

```
for i = 1 to n+1
        e[i, i-1] = q_{i-1}
        w[i, i-1] = q_{i-1}
```

具体计算过程：

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

② 区间长度=1： [1,1]、 [2,2]、 [3,3]、 [4,4]、 [5,5] **五个**



$$w[1,1] = w[1,0] + p_1 + q_1$$
$$= 0.05+0.15+0.10$$
$$= 0.30$$

$w[2,2]=0.25$

$w[3,3]=0.15$

$w[4,4]=0.20$

$w[5,5]=0.35$

$$e[1,1] = \min\{e[1,0]+e[2,1]\} + w[1,1]$$
$$= \min\{0.05+0.10\} + 0.30 \quad r=1$$
$$= 0.45$$

$e[2,2]=0.40$

$e[3,3]=0.25$

$e[4,4]=0.30$

$e[5,5]=0.50$

$root[1,1]=1$

$root[2,2]=2$

$root[3,3]=3$

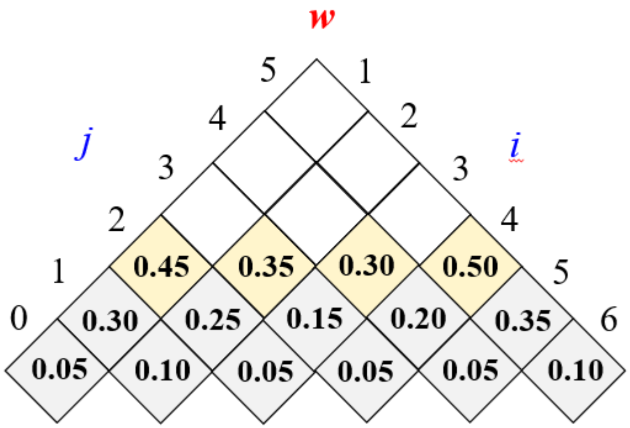
$root[4,4]=4$

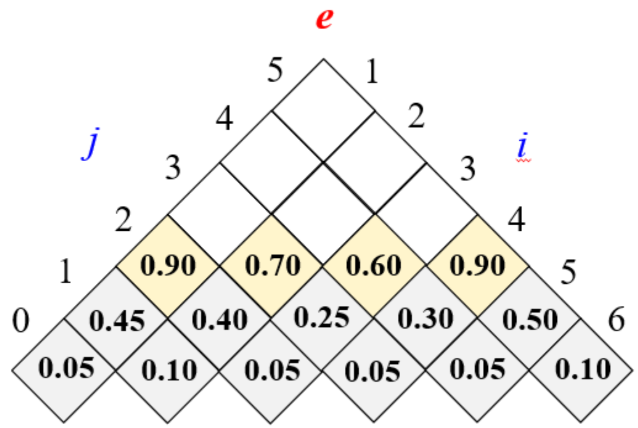
$root[5,5]=5$

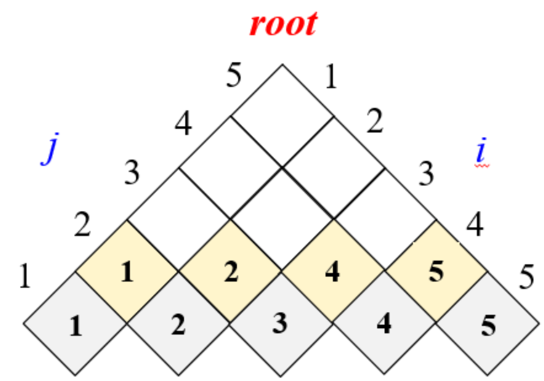
具体计算过程：

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

③ 区间长度=2： [1,2]、 [2,3]、 [3,4]、 [4,5] 四个







$$w[1,2] = w[1,1] + p_2 + q_2$$
$$= 0.30+0.10+0.05$$
$$= 0.45$$

$w[2,3] = 0.35$

$w[3,4] = 0.30$

$w[4,5] = 0.50$

$$e[1,2] = \min\{\, e[1,0]+e[2,2],\ \ r=1$$
$$e[1,1]+e[3,2]\,\} + w[1,2]\ \ r=2$$
$$= \min\{\, 0.05+0.40,$$
$$0.45+0.05\,\} + 0.45$$
$$= 0.90$$

$e[2,3]=0.70$

$e[3,4]=0.60$

$e[4,5]=0.90$

$root[1,2]=1$

$root[2,3]=2$

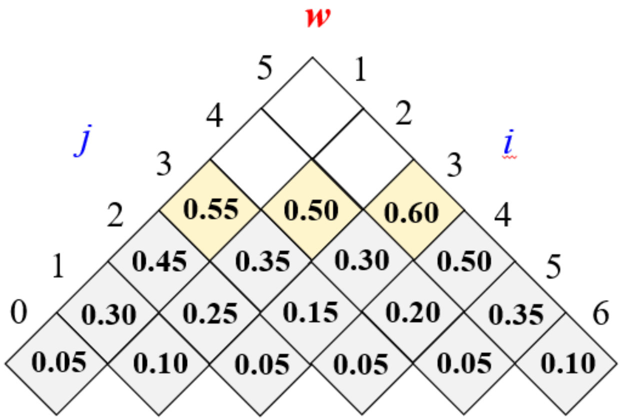
$root[3,4]=4$

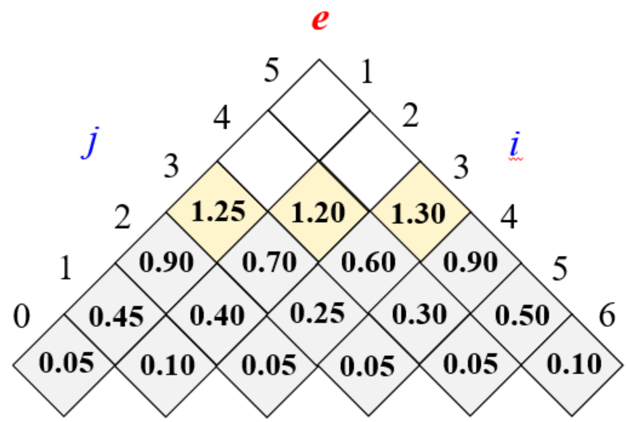
$root[4,5]=5$

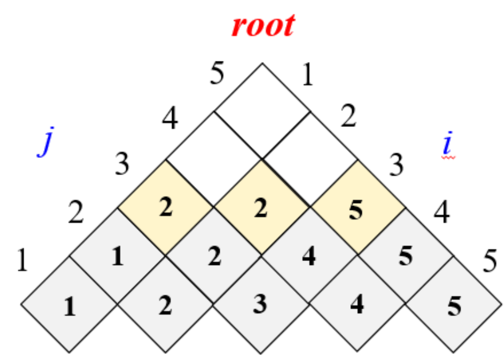
具体计算过程：

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

④ 区间长度=3： [1,3]、 [2,4]、 [3,5] 三个







$w[1,3] = w[1,2] + p_3 + q_3$

$= 0.45+0.05+0.05$

$= 0.55$

$w[2,4] = 0.50$

$w[3,5] = 0.60$

$e[1,3] = \min\{ e[1,0] + e[2,3],$ $r = 1$

$e[1,1] + e[3,3],$ $r = 2$

$e[1,2] + e[4,3]\} + w[1,3]$ $r = 3$

$= \min\{ 0.05 + 0.70,$

$0.45 + 0.25,$

$0.90 + 0.05 \} + 0.55$

$= 1.25$

$e[2,4]=1.20$

$e[3,5]=1.30$

$root[1,3]=2$

$root[2,4]=2$

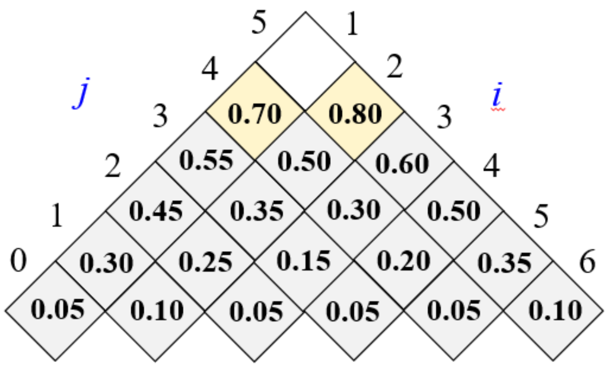
$root[3,5]=5$

具体计算过程：

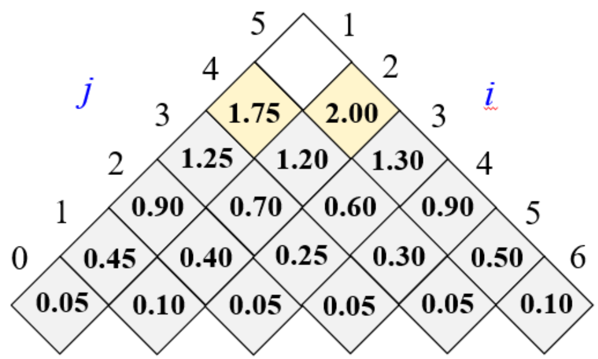
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

⑤ 区间长度=4： [1,4]、 [2,5] 二个

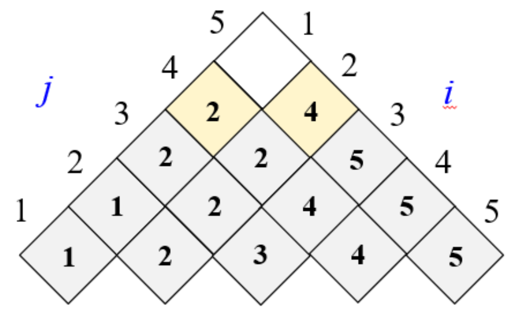
w



e



root



$w[1,4] = w[1,3] + p_4 + q_4$

$= 0.55+0.10+0.05$

$= 0.70$

$w[2,5] = 0.80$

$e[1,4] = \min\{ e[1,0] + e[2,4],$ $r = 1$

$e[1,1] + e[3,4],$ $r = 2$

$e[1,2] + e[4,4],$ $r = 3$

$e[1,3] + e[5,4]\} + w[1,4]$

$= \min\{ 0.05 + 1.20,$ $r = 4$

$0.45 + 0.60,$

$0.90 + 0.30,$

$1.25 + 0.05 \} + 0.70$

$= 1.75$

$e[2,5]=2.00$

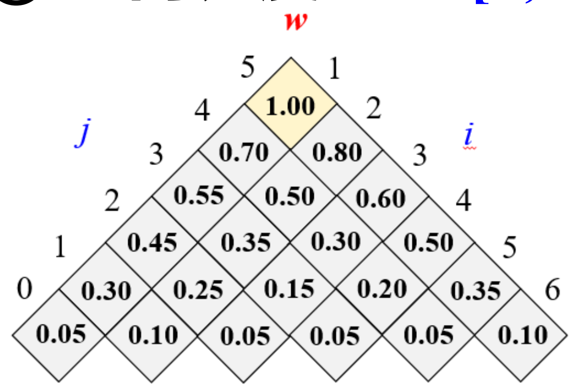
$root[1,4]=2$

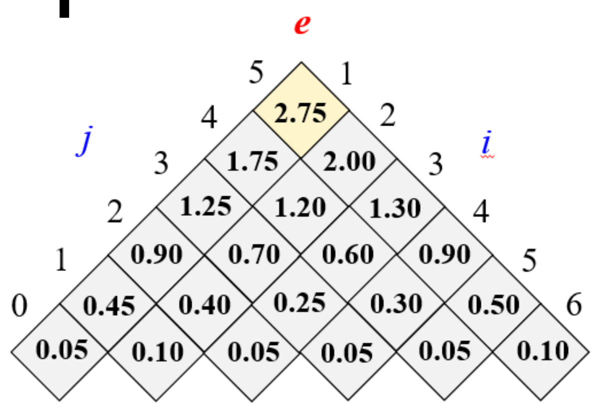
$root[2,5]=4$

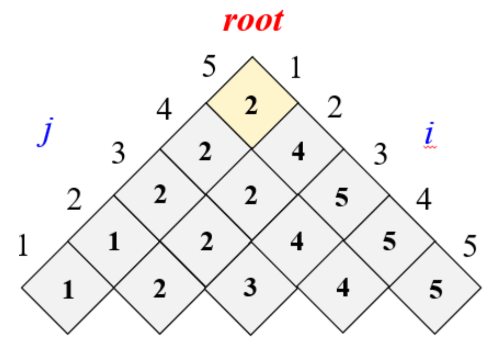
具体计算过程：

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

⑥ 区间长度=5： [1,5] 一个







$w[1,5] = w[1,4] + p_5 + q_5$

$= 0.70+0.20+0.10$

$= 1.00$

$e[1,5] = \min\{\, e[1,0] + e[2,5],$ $r = 1$

$e[1,1] + e[3,5],$ $r = 2$

$e[1,2] + e[4,5],$ $r = 3$

$e[1,3] + e[5,5],$ $r = 4$

$e[1,4] + e[6,5]\} + w[1,5]$ $r = 5$

$= \min\{\, 0.05 + 2.00,$

$0.45 + 1.30,$

$0.90 + 0.90,$

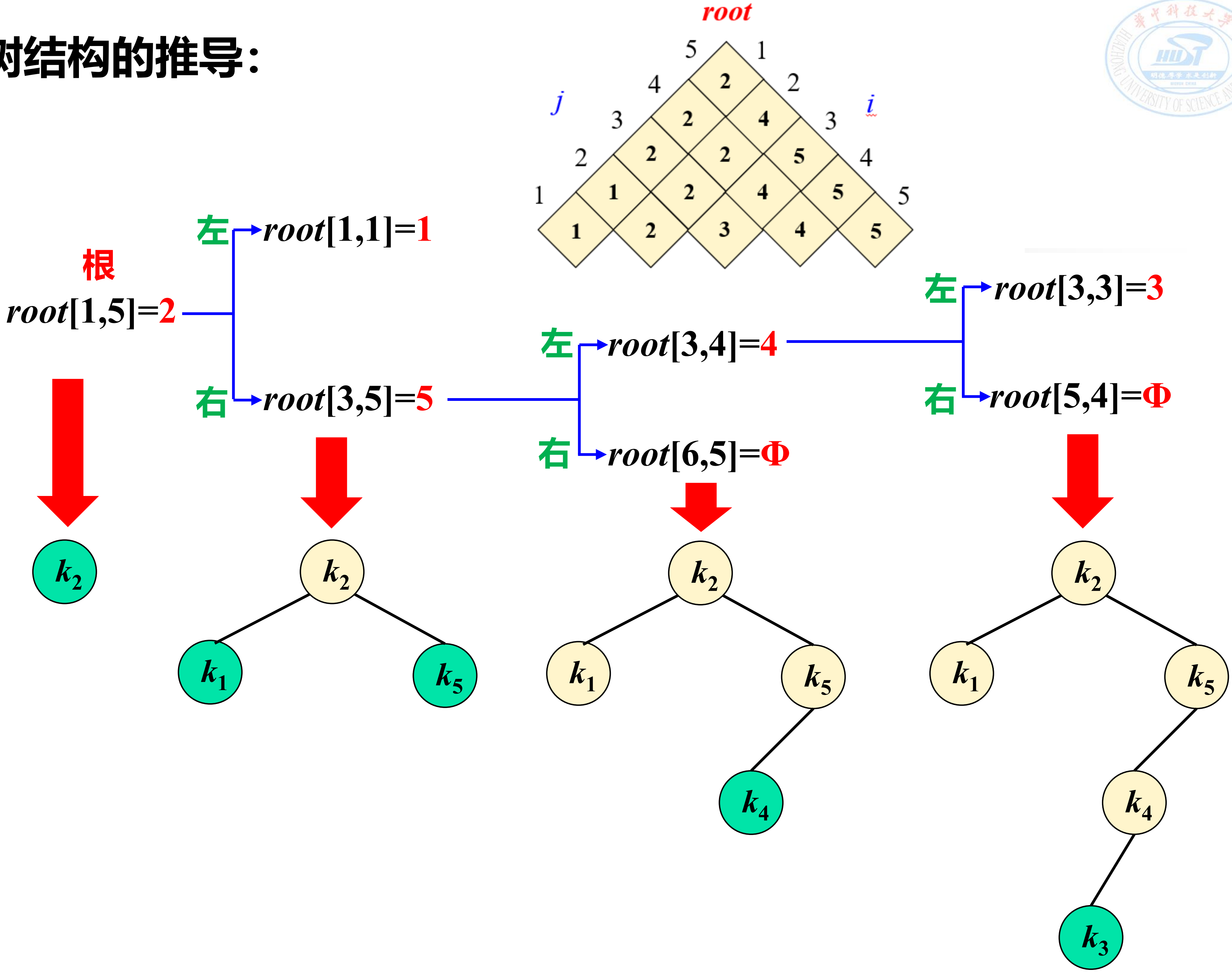
$1.25 + 0.50,$

$1.75 + 0.10\} + 1.00$

$= 2.75$

$root[1,5]=2$

树结构的推导:



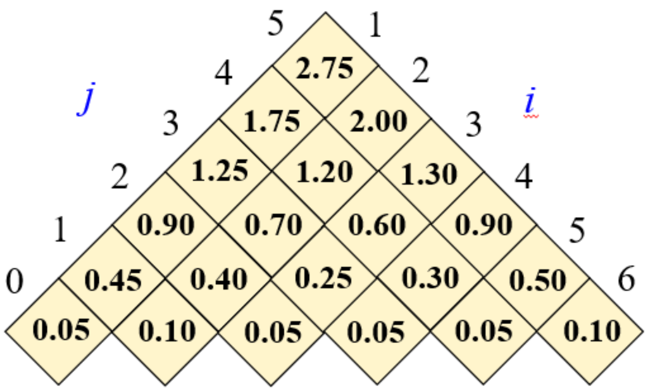
root
j
i
根
root[1,5]=2
左 root[1,1]=1
右 root[3,5]=5
左 root[3,4]=4
右 root[6,5]=Φ
左 root[3,3]=3
右 root[5,4]=Φ
k_1
k_2
k_3
k_4
k_5

课堂练习

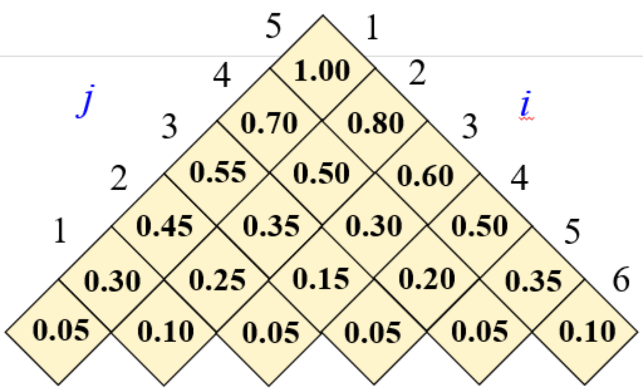
n=5，

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

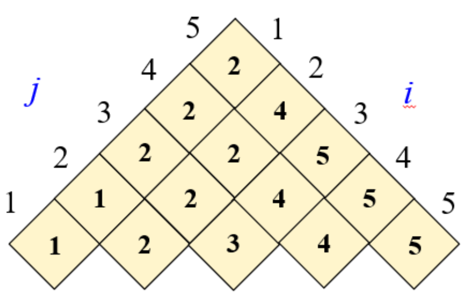
e



w



root



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

$$w[i,j] = w[i,j\text{-}1] + p_j + q_j$$

计算

(1) $w[2,3]$

$e[2,3]$

$root[2,3]$

(2) $w[2,5]$

$e[2,5]$

$root[2,5]$

本章作业：

- 计算题：**15.2-1**、**15.4-1**、**15.5-2**
- 算法设计题：**15.1-3**、**15.9**、**15-11**
- 证明题：
  - **15.2-5**
  - **15.3-6**：提示 基于以下数据讨论第二问。

	r_{ij}	$j=1$	2	3	4
	1	1	2	5/2	6
	2	1/2	1	3/2	3
i	3	2/5	2/3	1	3
	4	1/6	1/3	1/3	1

Let $c_1 = 2$ and $c_2 = c_3 = 3$.