15 Dynamic Programming

songyou@buaa.edu.cn

15 Dynamic Programming

- Assembly Lines Scheduling
- Steel Rod Cutting
- **Matrix-Chain Multiplication (15.2)**
  矩阵链相乘，或矩阵连乘问题
- **Characteristics(Elements) of dynamic programming (15.3)**
- **Longest Common Subsequence (15.4)**
- **Optimal binary search trees (15.5)**
  最优二叉搜索树

15.2 Matrix-chain multiplication (MCM)

- Given a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices to be multiplied, and we wish to compute the product　　n 个矩阵相乘，称为‘矩阵连乘’（或矩阵链乘法），如何求积？

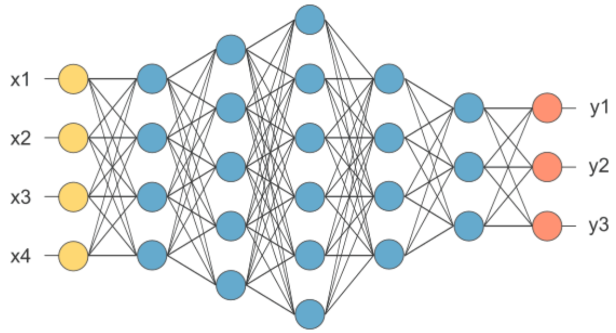
$$A_1A_2A_3A_4 \tag{15.10}$$

$$(A_1\,(A_2\,(A_3\,A_4)))\,,\quad (A_1\,((A_2\,A_3)\,A_4))\,,\quad ((A_1\,A_2)\,(A_3\,A_4)),\quad \ldots\ldots$$

- A product of matrices is fully parenthesized if it is either a single matrix or the product of two fully parenthesized matrix products, surrounded by parentheses. Example, A_1 , $(A_1((A_2A_3)A_4))$, $(A_1((A_2A_3)(A_4A_5)))$.
矩阵连乘全括号：仅有一个矩阵，或者两个“矩阵连乘全括号”的乘积且外层包括一个括号，如:

$$(\,A_1\,(\,(A_2\,A_3)\,A_4)\,)$$

这是嵌套的矩阵对，它给出了 矩阵连乘的一种求解顺序，也简称“矩阵全括号”。



- We can evaluate (15.10) using the standard algorithm for multiplying pairs of matrices as a subroutine once we have parenthesized it.　　“矩阵全括号”给出后，就可以用两个矩阵相乘的标准算法作为子程序来计算式 (15.10)。

Example: Multiplication of two matrices（矩阵相乘）

two $n \times n$ matrices A and B, Complexity($C = A \times B$) = ?

Standard method

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{pmatrix} \cdots\cdots\ .\ .. \\ \cdots\cdots c_{ij} .. \\ \cdots \\ \cdots\cdots\ .\ .. \end{pmatrix} = \begin{pmatrix} \cdots\cdots\cdots \\ \#\#\ldots\# \\ \cdots \\ \cdots\cdots\cdots \end{pmatrix} * \begin{pmatrix} \cdots\cdots\#.. \\ \cdots\cdots\#.. \\ \cdots \\ \cdots\cdots\#.. \end{pmatrix}$$

MATRIX-MULTIPLY(A, B)
  for $i \leftarrow 1$ to n
      for $j \leftarrow 1$ to n
          $C[i, j] \leftarrow 0$
          for $k \leftarrow 1$ to n
              $C[i, j] \leftarrow C[i, j] + A[i, k] \cdot B[k, j]$
  return C

Complexity:
$O(n^3)$ multiplications
and additions.
$T(n) = O(n^3)$.

15.2 Matrix-chain multiplication (MCM)

- Given a sequence (chain) <A_1, A_2, ..., A_n> of n matrices to be multiplied, and we wish to compute the product

$$A_1 A_2 A_3 A_4 \qquad (15.10)$$

- Matrix multiplication is associative, so all parenthesizations yield the same product. For example, if the chain of matrices is <A_1, A_2, A_3, A_4>, the product $A_1A_2A_3A_4$ can be fully parenthesized in five distinct ways:

矩阵连乘满足结合律，因此对所有加括号的方式，矩阵连乘的积相同。例如…

$(A_1(A_2(A_3A_4)))$, $(A_1((A_2A_3)A_4))$, $((A_1A_2)(A_3A_4))$,
$((A_1(A_2A_3))A_4)$, $(((A_1A_2)A_3)A_4)$.

15.2 Matrix-chain multiplication (MCM)

The way we parenthesize a chain of matrices can have a dramatic impact on the cost of evaluating the product.

采用不同的加括号方式，可导致差异极大的乘法开销

$(A_1(A_2(A_3A_4)))$,

$(A_1((A_2A_3)A_4))$,

$((A_1A_2)(A_3A_4))$,

$((A_1(A_2A_3))A_4)$,

$(((A_1A_2)A_3)A_4)$.

15.2 Matrix-chain multiplication (MCM)

- First, consider the cost of multiplying two matrices.
- Two matrices A and B can be multiplied only if they are compatible: columns of A = rows of B. 仅当矩阵A和B相容时，A和B能相乘
  - If A is $p \times q$, B is $q \times r$, then C is $p \times r$.
  - The time to compute C is dominated by the number of scalar multiplications in line 7, which is pqr. 两个矩阵相乘，标量乘法的次数是 pqr

$$A * B \Rightarrow C$$

$$\begin{pmatrix} \text{..........} \\ \#\#\ldots\# \\ \text{..........} \\ \text{..........} \end{pmatrix}_{p*q} * \begin{pmatrix} \text{.......}\#\text{..} \\ \text{.......}\#\text{..} \\ \cdots \\ \text{.......}\#\text{..} \end{pmatrix}_{q*r} \Rightarrow \begin{pmatrix} \text{........ . ..} \\ \text{........}c_{ij}\text{..} \\ \cdots \\ \text{........ . ..} \end{pmatrix}_{p*r}$$

```
MATRIX-MULTIPLY(A, B)
1  if columns[A] ≠ rows[B]
2      return "error: incompatible dimensions"
3  else for i ← 1 to rows[A]      // p is row[A]
4           for j ← 1 to columns[B] // r is columns[B]
5               C[i, j] ← 0
6               for k ← 1 to columns[A] // q is columns[A]
7                   C[i, j] ← C[i, j] + A[i, k]·B[k, j]
8  return C
```

15.2 Matrix-chain multiplication (MCM)

- For $A_{p\times q}$, $B_{q\times r}$, $C=AB$ is $p\times r$. The # of scalar multiplications is pqr. 标量乘法的次数是 pqr
- 考虑一个简单问题，三个矩阵连乘 $<A_1, A_2, A_3>$,
  设 A_1: 10×100; A_2: 100×5; A_3: 5×50
  - 当 $A = ((A_1A_2)A_3)$,
    (a) $C = A_1A_2$, 标量乘法的次数是: $10 \cdot 100 \cdot 5 = 5{,}000$, $C_{10\times 5}$
    (b) $A = CA_3$, 标量乘法的次数是: $10 \cdot 5 \cdot 50 = 2{,}500$, $A_{10\times 50}$
    因此，标量乘法的次数是: for a total of 7,500 .
  - 当 $A = (A_1(A_2A_3))$,
    (a) $C_{100\times 50} = A_2A_3$, 标量乘法的次数是: $100 \cdot 5 \cdot 50 = 25{,}000$,
    (b) $A_{10\times 50} = A_1C$, 标量乘法的次数是: $10 \cdot 100 \cdot 50 = 50{,}000$,
    因此，标量乘法的次数是: for a total of 75,000 .
  - The first case is 10 times faster than the second. 运算效率十倍之差！

$7{,}500 \ll 75{,}000$

15.2 Matrix-chain multiplication (MCM)

$A_1 A_2 A_3 A_4 A_5$: $(A_1 A_2 A_3)(A_4 A_5)$? $(A_1 A_2)(A_3 A_4 A_5)$? ……

$A_1 (A_2 A_3)$? $(A_1 A_2) A_3$?

- **MCM problem** : Given a chain $< A_1, A_2, \ldots, A_n >$, $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1 A_2 \ldots A_n$ in a way that minimizes the number of scalar multiplications.

  n 个矩阵 $< A_1, A_2, \ldots, A_n >$, $i = 1, 2, \ldots, n$, A_i 的维数为 $p_{i-1} \times p_i$, 给出矩阵连乘的一种加括号方式，使得标量乘法的次数最小（少）。

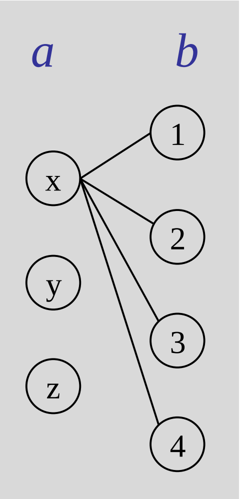
Counting the number of parenthesizations（有多少种全括号方式）

$$A_1\,A_2\,A_3\,A_4\,A_5$$

$$(\,(\underline{A_1\,A_2\,A_3}\,)\,(A_4\,A_5)\,)? \qquad (A_1\,A_2)\,(A_3\,A_4\,A_5)? \quad \ldots\ldots$$

$$(\,A_1\,(A_2\,A_3)\,)? \quad ((A_1\,A_2)\,A_3\,)?$$

$$(\ \underline{(\,A_i\,(A_{i+1}\ldots)\,(\ldots)\ldots A_k\,)}\ \underline{(\,A_{k+1}\ldots\ A_{j-1}\,A_j)}\ \ )$$



- 暴力穷举（枚举所有的全括号方式）（全括号：矩阵连乘加括号）
- $P(n)$: n 个矩阵连乘，有 $P(n)$ 种全括号方式
    - $n = 1$, 显然，$P(n) = 1$.
    - $n \geq 2$,

$$P(n)=\begin{cases}1 & , \quad \text{if } n=1,\\ \sum_{k=1}^{n-1}P(k)P(n-k), & \text{if } n\geq 2.\end{cases} \qquad (15.11)$$

- 式 (15.11) 的解的计算时间是 $\Omega(2^n)$? (guess, then prove), a poor strategy.

Counting the number of parenthesizations（有多少种全括号方式）

$$A_1\,A_2\,A_3\,A_4\,A_5$$

$$(\,(\underline{A_1\,A_2\,A_3})\,(A_4\,A_5)\,)? \qquad (A_1\,A_2)\,(A_3\,A_4\,A_5)? \quad \ldots\ldots$$

$$(\,A_1\,(A_2\,A_3)\,)? \qquad ((A_1\,A_2)\,A_3\,)?$$

$$(\;\underline{(\,A_i\,(A_{i+1}\ldots)\,(\ldots)\ldots A_k\,)}\;\underline{(\,A_{k+1}\ldots\;A_{j-1}\,A_j)}\;)$$

$$P(n)=\begin{cases}1 & ,\text{ if } n=1,\\ \sum_{k=1}^{n-1}P(k)P(n-k), & \text{if } n\ge 2.\end{cases} \qquad (15.11)$$

Running time is $\Omega(2^n)$

Dynamic Programming to solve MCM:

Four Steps

Step 1: The structure of an optimal parenthesization （最优全括号的结构）

$((\underline{A_1 A_2 A_3})(A_4 A_5))$? $(A_1 A_2)(A_3 A_4 A_5)$? ……

$(A_1 (A_2 A_3))$? $((A_1 A_2) A_3)$?

$$(\ \underline{(A_i (A_{i+1} \ldots)(\ldots)\ldots A_k)}\ \underline{(A_{k+1} \ldots\ A_{j-1} A_j)}\ )$$

寻找最优子结构…

$A_{i..j}\ (i \le j)$: 矩阵连乘 $A_i A_{i+1} \ldots A_k A_{k+1} \ldots A_j$

- $i < j$, nontrivial, any parenthesization of the product $A_i A_{i+1} \ldots A_j$ must split the product between A_k and A_{k+1} for some integer k in the range $i \le k < j$.
  非平凡情况下，全括号 $A_i A_{i+1} \ldots A_j$ 必定在位置 k 把问题分成两个部分，如下
- $A_{i..k} \cdot A_{k+1..j} = A_{i\,..\,j}$

全括号 $A_i A_{i+1} \ldots A_j$ 的计算代价（标量乘法），cost($A_{i\,..\,j}$)

cost($A_{i\,..\,j}$) = cost($A_{i\,..\,k}$) ＋ cost($A_{k+1\,..\,j}$) ＋ the cost of multiplying $A_{i..k} \cdot A_{k+1..j}$

Step 1: The structure of an optimal parenthesization （最优全括号的结构）

最优子结构

- 最优全括号 $A_{i..j}$ 在位置 A_k 和 A_{k+1} 处把问题分成两个部分 $A_{i..k}$ 和 $A_{k+1..j}$ 之积

$$\Big(\ \underline{(A_i(A_{i+1}\ldots)(\ldots)\ldots A_k)}\ \underline{(A_{k+1}\ldots\ A_{j-1}A_j)}\ \Big)$$

- The parenthesization of the "prefix" subchain $A_iA_{i+1}\ldots A_k$ within this optimal parenthesization of $A_iA_{i+1}\ldots A_j$ must be an optimal parenthesization of $A_iA_{i+1}\ldots A_k$?

$A_{i..j}$ 的最优全括号中的 $A_{i..k}$ 的全括号必定是 $A_{i..k}$ 的最优全括号

$$\underbrace{\Big(\ \underbrace{(A_i(A_{i+1}\ldots)(\ldots)\ldots A_k)}_{M}\ (A_{k+1}\ldots\ A_{j-1}A_j)\ \Big)}_{X}$$

Proof

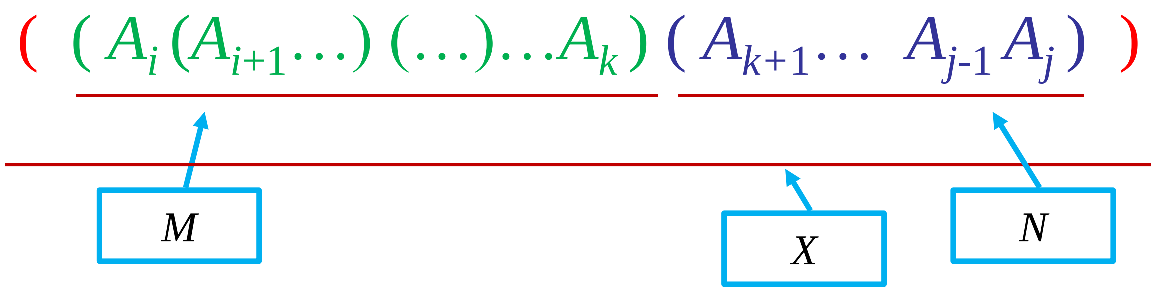
如果 X 最优，则 M 最优

最优子结构：$A_{i..j}$ 的最优全括号 X 中的 $A_{i..k}$ 的全括号M 必定是 $A_{i..k}$ 的最优全括号。

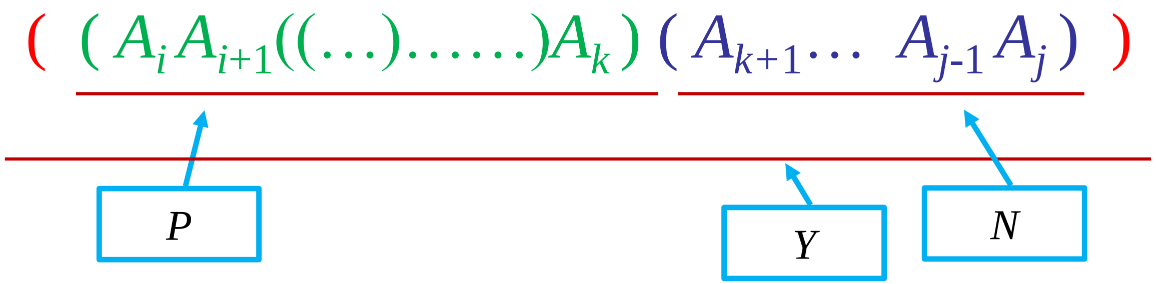
Proof

对子问题 $A_{i..j}$，设存在另一种最优全括号形式 P，即，P 的标量乘法比 M 还少，显然，Y 对应的全括号所需要的标量乘法次数比 X 少，跟 X 是最优全括号矛盾。

同理，适用于全括号 N



X 最优 ➔ M 最优



Step 1: The structure of an optimal parenthesization （最优全括号的结构）

$$\underbrace{(\ \underbrace{(A_i\,(A_{i+1}\ldots)\,(\ldots)\ldots A_k)}_{M}\,(A_{k+1}\ldots\ A_{j-1}\,A_j)\ )}_{X}$$

- 根据最优子结构，可以从用 M 来构造 X（M 是 $A_{i..k}$ 的最优解，X 是 $A_{i..j}$ 的最优解）

- X 的求解过程：.
  - 分割：一个最优解 X 在某个 k 分割 $A_{i..j}$ 为 $A_{i..k}$ 和 $A_{k+1..j}$
  - 求子问题 $A_{i..j}$ 的最优解 M
  - 合并：从 M 构造 X

Step 1: The structure of an optimal parenthesization （最优全括号的结构）

$$A_1\,A_2\,A_3\,A_4\,A_5$$

$$(\underline{(A_1\,A_2\,A_3)}(A_4\,A_5))? \qquad (A_1\,A_2)(A_3\,A_4\,A_5)? \quad \ldots\ldots$$

$$(A_1\,(A_2\,A_3))? \quad ((A_1\,A_2)\,A_3)?$$

$$(\ \underline{(A_i\,(A_{i+1}\ldots)\,(\ldots)\ldots A_k)}\ \underline{(A_{k+1}\ldots\ A_{j-1}\,A_j)}\ )$$

We must consider all possible places so that we are sure of having examined the optimal one.

需要考虑所有分割位置 k 以确保最优解是其中之一

Step 2: A recursive solution（递归解）

$$\underline{(\ \underline{(A_i (A_{i+1}\ldots)(\ldots)\ldots A_k)}\ (A_{k+1}\ldots\ A_{j-1} A_j)\ )}$$

- Define the cost of an optimal solution recursively in terms of the optimal solutions to subproblems.
  根据子问题的最优解可以递归地定义原问题的最优解

- Subproblems $A_{i..j}$: determining the minimum cost of a parenthesization of $A_i A_{i+1} \ldots A_j$ for $1 \le i \le j \le n$.

  Not $A_1 A_2 \ldots A_j$, Why? 为什么子问题定义为 $A_{i..j}$ 而不是 $A_{1..j}$

Step 2: A recursive solution（递归解）

$$\Big(\underbrace{(A_i (A_{i+1} \ldots)(\ldots) \ldots A_k)}_{M} (A_{k+1} \ldots A_{j-1} A_j) \Big)$$

X

$m[i, j] = |X|$: the minimum # of scalar multiplications to compute $A_{i..j}$;
the cost of a cheapest way to compute $A_{1..n}$ is $m[1, n]$.

$m[i, j]$ ：$A_{i..j}$ 的最优全括号的标量乘法次数。

$m[1, n]$: $A_{1..n}$ 的最优全括号的标量乘法次数，即，原问题最优值。

- $i = j$, $A_{i..i} = A_i$，一个矩阵，没有乘法，显然, $m[i, i] = 0$, $i = 1, 2, \ldots, n$.
- $i < j$?

Step 2: A recursive solution（递归解）

$$\underline{(\ \underline{(A_i (A_{i+1} \ldots) (\ldots) \ldots A_k)} (A_{k+1} \ldots\ A_{j-1} A_j)\ )}$$

$m[i, j]$ ：$A_{i..j}$ 的最优全括号的标量乘法次数。

当 $i < j$, 令最优的分割位置是 A_k 和 A_{k+1} 之间， $i \le k < j$.

因此 ，

$$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$$

A_i 的维数为 $p_{i-1} \times p_i$, 因此 $A_{i..k}$ 的维数为 $p_{i-1} \times p_k$, $A_{k+1..j}$ 的维数为 $p_k \times p_j$.

Step 2: A recursive solution（递归解）

$$\underline{(\ \underline{(A_i(A_{i+1}\dots)(\dots)\dots A_k)}\ (A_{k+1}\dots\ A_{j-1}A_j)\ )}$$

$$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$$

- 该递归方程基于已知 k，而 k 是未知的变量，取值区间为 $i \le k < j$.
- 为了解的完备性，需要遍历所有的 k，因此有

$$m[i,j]=\begin{cases}0 & \text{, if } i=j, \\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & \text{if } i<j.\end{cases} \quad (15.12)$$

$m[i, j]$:子问题 $A_{i..j}$ 的最优全括号的标量乘法次数。

Step 2: A recursive solution（递归解）

$$\underline{(\ \underline{(A_i(A_{i+1}\ldots)(\ldots)\ldots A_k)}(A_{k+1}\ldots\ A_{j-1}A_j)\ )}$$

$$m[i,j]=\begin{cases}0 & , \quad \text{if } i=j, \\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & \quad \text{if } i<j.\end{cases} \qquad (15.12)$$

$m[i, j]$:子问题 $A_{i..j}$ 的最优全括号的标量乘法次数。

构造最优解：

式(15.12)求得的 $m[i, j]$ 是最优值。

定义 $s[i, j]$ 用于存储值 k，表示子问题 $A_{i..j}$ 的最优全括号时的分割位置，以便用于构造最优解（找到每个最优加括号的位置）。

Step 3: Computing the optimal costs（计算最优全括号的乘法次数）

$$m[i,j]=\begin{cases}0, & \text{if } i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & \text{if } i<j.\end{cases} \tag{15.12}$$

$m[1, n]$ 表示原问题 $A_1A_2\ldots A_n$ 的解.

式 (15.12) 的递归算法：

```
RE-MCM(p, i, j)
if i == j
      return 0
m[i, j] ← ∞
for k ← i to j-1
      q ← RE-MCM(p, i, k) + RE-MCM(p, k+1, j) + p_{i-1} p_k p_j
      if q < m[i, j]
            m[i, j] ← q
return m[i, j]
```

Running time?

Step 3: Computing the optimal costs（计算最优全括号的乘法次数）

$$(\underline{(A_i (A_{i+1}\ldots)(\ldots)\ldots A_k)} (A_{k+1}\ldots A_{j-1} A_j))$$

$$m[i,j] = \begin{cases} 0 & , \quad \text{if } i = j, \\ \min\limits_{i \le k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\}, & \text{if } i < j. \end{cases} \qquad (15.12)$$

Recursion, Extremely slow! 直接递归，极慢！

# of subproblems: one problem for each choice of i and j satisfying $1 \le i \le j \le n$?

矩阵连乘的所有子问题个数为？

$$C_n^2 + C_n^1 = \Theta(n^2)$$

$$1 \le i = j \le n,\ C_n^1 = n$$

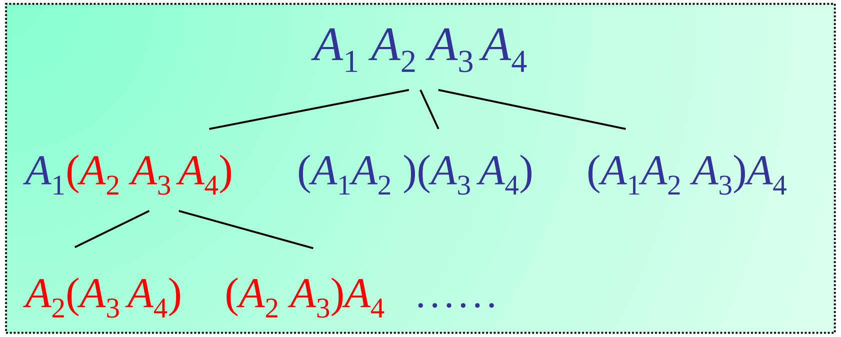
$$1 \le i < j \le n,\ C_n^2 = n(n-1)/2$$

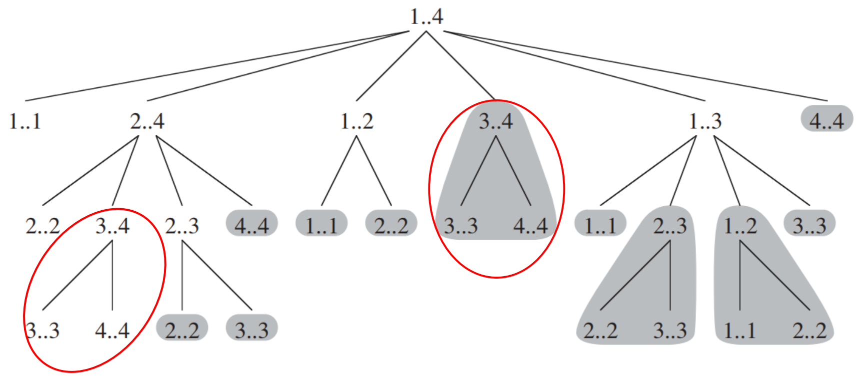
Step 3: Computing the optimal costs（计算最优全括号的乘法次数）

$$\underline{(\ \underline{(A_i (A_{i+1}\ldots)(\ldots)\ldots A_k)}\ (A_{k+1}\ldots\ A_{j-1} A_j)\ )}$$

$$m[i,j]=\begin{cases}0 & ,\quad \text{if } i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & \quad \text{if } i<j.\end{cases} \qquad (15.12)$$

- A recursive algorithm may encounter each subproblem many times in different branches of its recursion tree. 递归计算时，递归过程会重复计算相同的子问题
- Overlapping subproblems: the second hallmark of the applicability of dynamic programming. 重叠子问题，动态规划法的第二个重要特点





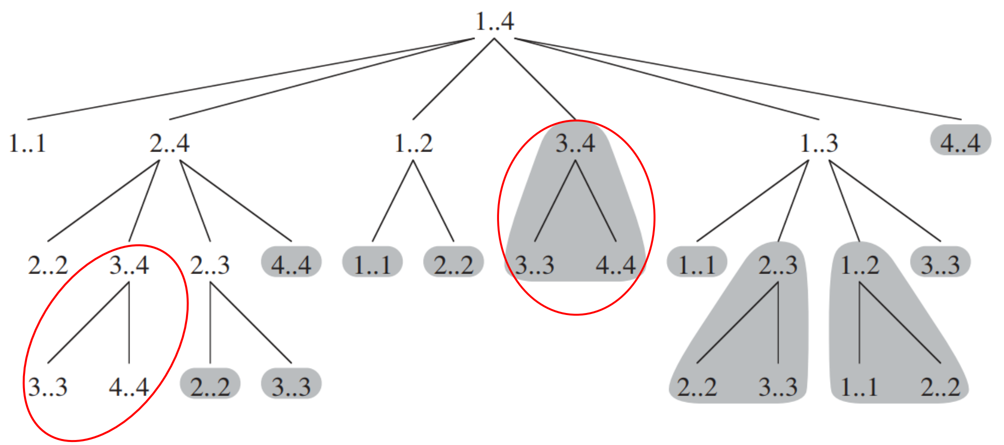
Step 3: Computing the optimal costs（计算最优全括号的乘法次数）

$$\underline{(\ \underline{(A_i(A_{i+1}\ldots)(\ldots)\ldots A_k)}\ (A_{k+1}\ldots\ A_{j\text{-}1}A_j)\ )}$$

$$m[i,j]=\begin{cases}0 & ,\quad \text{if } i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & \quad \text{if } i<j.\end{cases} \qquad (15.12)$$

- # of subproblems: one problem for each choice of i and j satisfying $1 \le i \le j \le n$, or $C_n^2 + C_n^1 = \frac{n(n-1)}{2} + n = \frac{1}{2}(n^2+n) = \Theta(n^2)$ in all. 子问题总数为 $\Theta(n^2)$

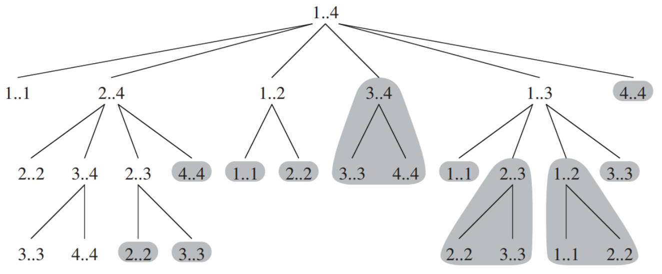
- Instead of recursive method, computing the optimal cost by using a tabular, bottom-up approach.
不用递归方法，而采用列表方式、自底向上的方法计算最优解



Step 3：Computing the optimal costs（计算最优全括号的乘法次数）

$$\underline{(\ \underline{(A_i (A_{i+1}\dots)(\dots)\dots A_k)}\ (A_{k+1}\dots\ A_{j-1} A_j)\ )}$$

$$m[i,j]=\begin{cases}0, & \text{if } i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_k p_j\}, & \text{if } i<j.\end{cases} \qquad (15.12)$$



问题：矩阵 A_i，维数 $p_{i-1} \times p_i$

输入：$p = \langle p_0, p_1, \dots, p_n \rangle$.

程序：表格（数组）$m_{n,n}$ 存储每一个子问题的最优值 $m[i, j]$;

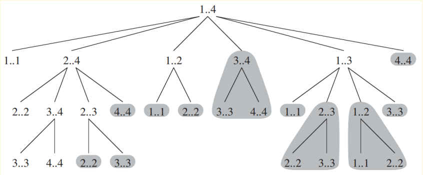
辅助表格 $s_{n,n}$，其中 $s[i, j]$ 记录求 $m[i, j]$ 时的最优分割位置 k.

```
MCM-DP(p)
n ← length[p] – 1
for i ← 1 to n
    m[i, i] ← 0
for l ← 2 to n                // l is the chain length.
    for i ← 1 to n - l + 1
        j ← i + l – 1
        m[i, j] ← ∞
        for k ← i to j - 1
            q ← m[i, k]+m[k+1, j]+p_{i-1}p_k p_j
            if q < m[i, j]
                m[i, j] ← q
                s[i, j] ← k
return m and s
```

Step 3: Computing the optimal costs（计算最优全括号的乘法次数）

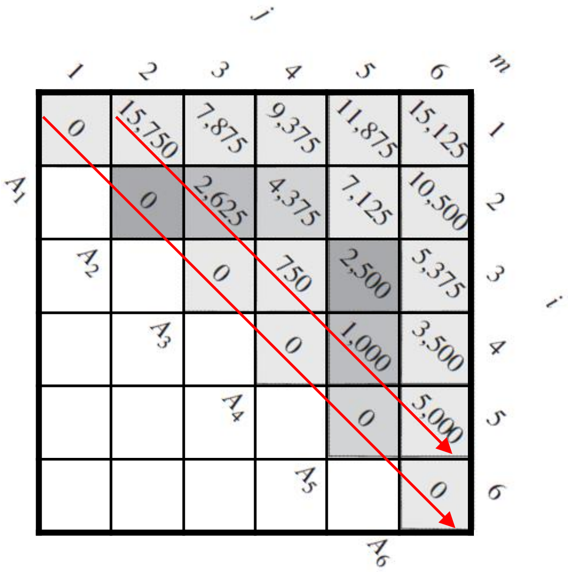
$$\Big(\ \underline{(A_i(A_{i+1}\ldots)(\ldots)\ldots A_k)}\ (A_{k+1}\ldots\ A_{j-1}A_j)\ \Big)$$

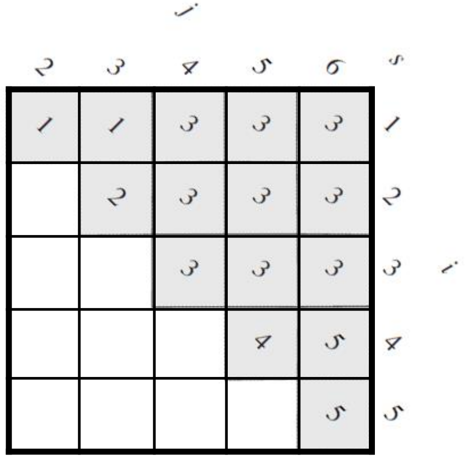
$$m[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min\limits_{i \le k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\}, & \text{if } i < j. \end{cases} \quad (15.12)$$



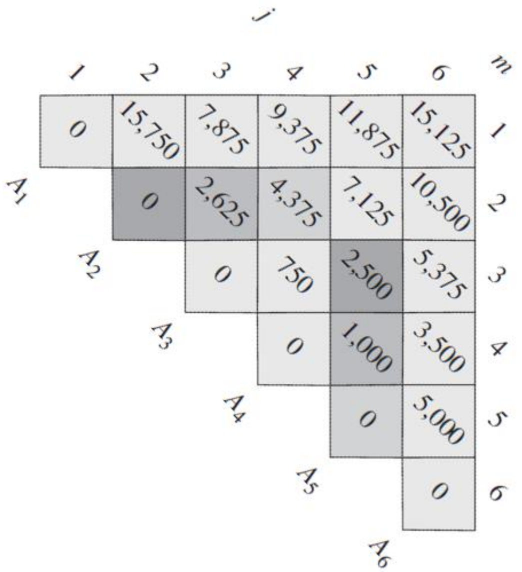

```
MCM-DP(p)
n ← length[p] – 1
for i ← 1 to n
      m[i, i] ← 0
for l ← 2 to n                    // l is the chain length.
      for i ← 1 to n - l + 1
            j ← i + l – 1
            m[i, j] ← ∞
            for k ← i to j - 1
                  q ← m[i, k]+m[k+1, j]+p_{i-1}p_k p_j
                  if q < m[i, j]
                        m[i, j] ← q
                        s[i, j] ← k
return m and s
```

A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

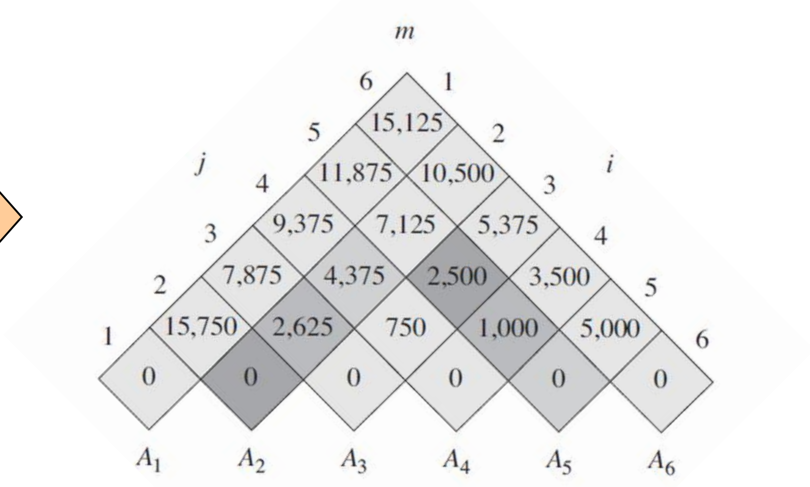


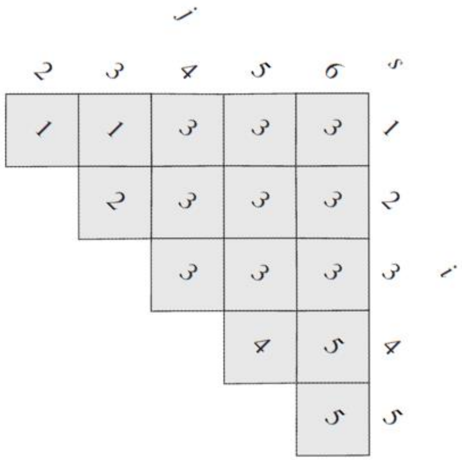


Step 3：Computing the optimal costs（计算最优全括号的乘法次数）

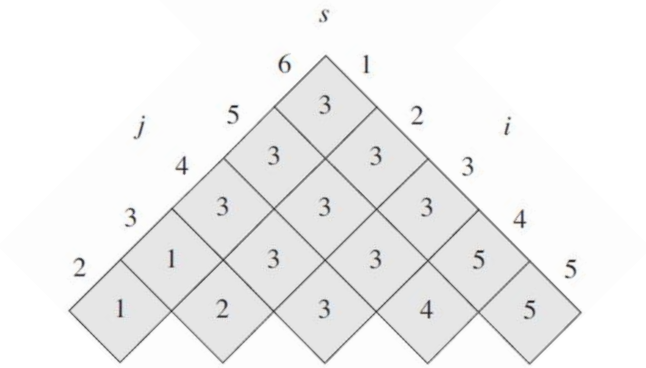


rotate 45^0





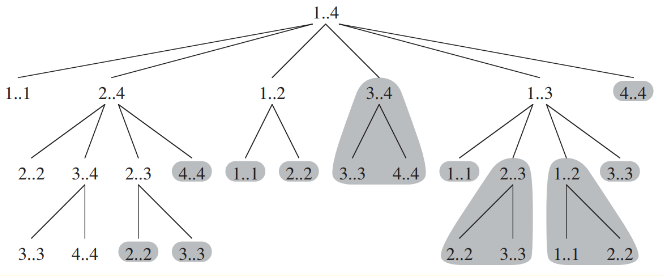
rotate 45^0



Step 3: Computing the optimal costs（计算最优全括号的乘法次数）

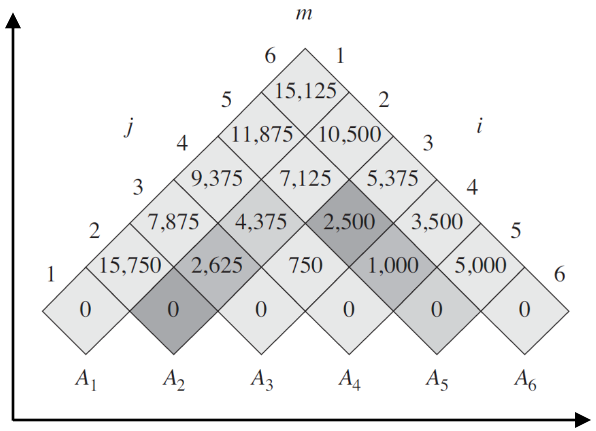
$$\Big(\ \underline{\big(A_i\,(A_{i+1}\ldots)\,(\ldots)\ldots A_k\big)}\ \big(A_{k+1}\ldots\ A_{j-1}\,A_j\big)\ \Big)$$

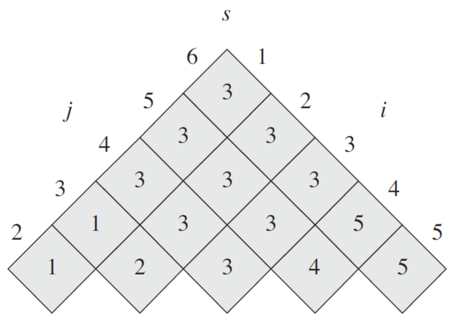
$$m[i,j]=\begin{cases}0 & ,\quad \text{if } i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & \quad \text{if } i<j.\end{cases} \qquad (15.12)$$




```
MCM-DP(p)
n ← length[p] – 1
for i ← 1 to n
      m[i, i] ← 0
for l ← 2 to n        // l is the chain length.
      for i ← 1 to n - l + 1
            j ← i + l – 1
            m[i, j] ← ∞
            for k ← i to j - 1
                  q ← m[i, k]+m[k+1, j]+p_{i-1}p_k p_j
                  if q < m[i, j]
                        m[i, j] ← q
                        s[i, j] ← k
return m and s
```

A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25





Step 3: Computing the optimal costs（计算最优全括号的乘法次数）

```
MCM-DP(p)
n ← length[p] – 1
for i ← 1 to n
    m[i, i] ← 0
for l ← 2 to n      // l is the chain length.
    for i ← 1 to n - l + 1
        j ← i + l – 1
        m[i, j] ← ∞
        for k ← i to j - 1
            q ← m[i, k]+m[k+1, j]+p_{i-1} p_k p_j
            if q < m[i, j]
                m[i, j] ← q
                s[i, j] ← k
return m and s
```

The running time?

Space requirement?

Step 3: Computing the optimal costs（计算最优全括号的乘法次数）

```
MCM-DP(p)
n ← length[p] – 1
for i ← 1 to n
      m[i, i] ← 0
for l ← 2 to n                          // l : n-1 times
      for i ← 1 to n - l + 1           // i : n-l+1 times
            j ← i + l – 1
            m[i, j] ← ∞
            for k ← i to j - 1         // k : j-i=l-1 times
                  q ← m[i, k]+m[k+1, j]+p_{i-1}p_k p_j
                  if q < m[i, j]
                        m[i, j] ← q
                        s[i, j] ← k
return m and s
```

Exercise:

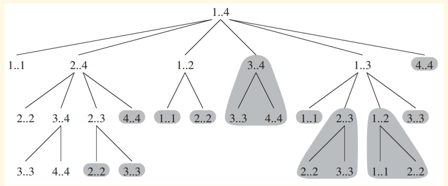
$$T(n) = \sum_{l=2}^{n} (n - l + 1)(l - 1)$$

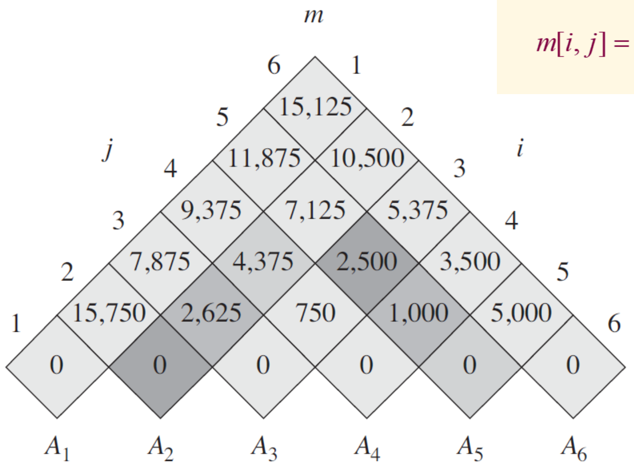
The running time?

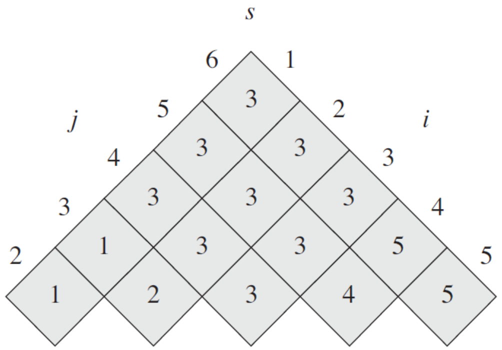
Step 4: Constructing an optimal solution （求最优解）

$$\underline{(\ \underline{(A_i(A_{i+1}\ldots)(\ldots)\ldots A_k)}\ (A_{k+1}\ldots\ A_{j-1}A_j)\ )}$$

$$m[i,j]=\begin{cases}0 & \text{if } i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & \text{if } i<j.\end{cases} \qquad (15.12)$$





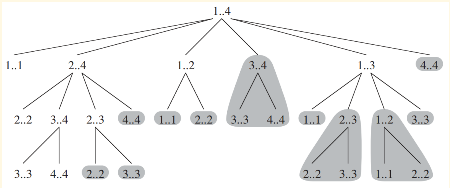


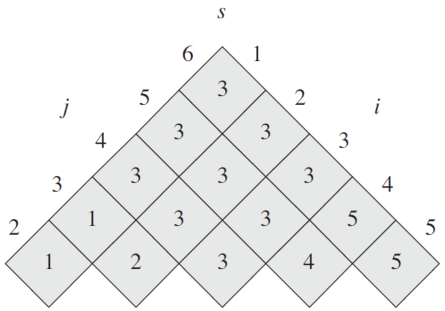
- MCM-DP determines the optimal number $m[i, j]$, but does not directly show how to multiply the matrices.
  算法MCM-DP 给出了如何求最优全括号的乘法次数 $m[i, j]$，但对于按什么顺序来相乘各矩阵，没有给出具体方法 (giving optimal value, no optimal solution)
- Constructing an optimal solution from table $s[1..\ n\text{-}1,\ 2..\ n]$.
  利用辅助数组 s 来构造最优解

Step 4: Constructing an optimal solution （求最优解）

$$(\ (A_i(A_{i+1}\ldots)(\ldots)\ldots A_k)\ (A_{k+1}\ldots\ A_{j-1}A_j)\ )$$

$$m[i,j]=\begin{cases}0 & \text{if } i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & \text{if } i<j.\end{cases} \qquad (15.12)$$





- Each entry $s[i, j]$ records the value of k such that the optimal parenthesization of $A_iA_{i+1}\cdots A_j$ splits the product between A_k and A_{k+1}. Thus, the final matrix multiplication in computing $A_{1..n}$ optimally is $A_{1..s[1,n]}A_{s[1,n]+1..n}$. $s[i, j]$ 记录值 k，表示在矩阵连乘 $A_iA_{i+1}\cdots A_j$ 的最优全括号中，分割点位于 A_k 和 A_{k+1} 之间。因此，矩阵连乘 $A_{1..n}$ 的最优分割方式为 $(A_1\ A_2 \ldots A_{s[1,n]})(A_{s[1,n]+1}\cdots\ A_n)$.
  - 矩阵连乘的括号可以递归计算，

    $s[1, s[1, n]]$ 计算 splits $A_{1..s[1,\ n]}$ 的分割位置

    $s[s[1, n] + 1, n]$ 计算 splits $A_{s[1,\ n]+1..n}$ 的分割位置
- PRINT-OPTIMAL-PARENS(s, i, j)

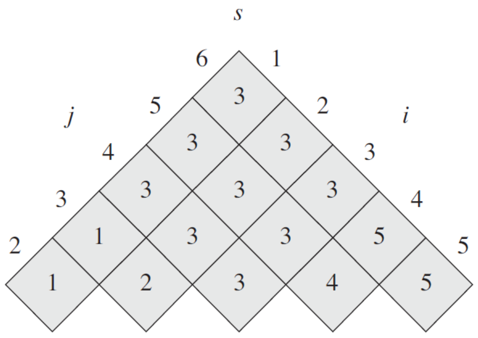
Step 4: Constructing an optimal solution （求最优解）

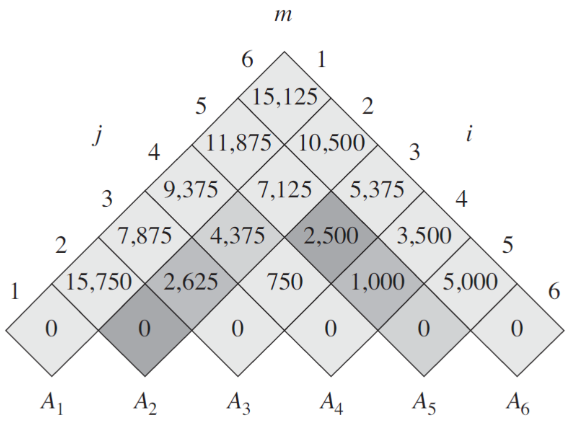
PRINT-OPTIMAL-PARENS(s, i, j) printing an optimal parenthesization of $<A_i, A_{i+1}, \ldots, A_j>$ recursively, given the s table. The initial call i=1, j=n.

求得辅助矩阵 s 后，如下递归算法输出最优加括号

$A_1A_2A_3A_4A_5A_6$

A_1 30×35
A_2 35×15
A_3 15×5
A_4 5×10
A_5 10×20
A_6 20×25






```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
    print "A"_i
else
    print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j]+1, j)
    print ")"
```

How to work?

$((A_1(A_2A_3))\ ((A_4A_5)A_6))$

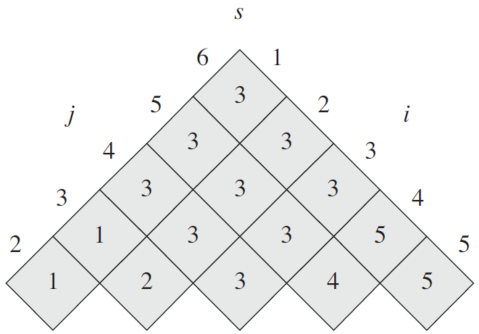
Step 4: Constructing an optimal solution （求最优解）

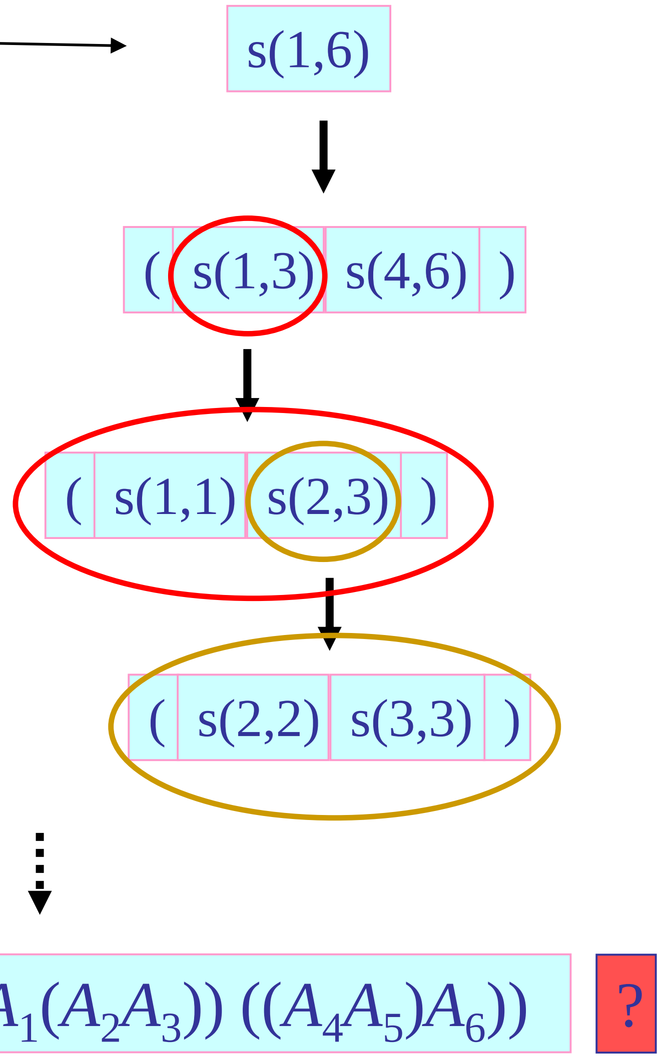
```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
    print "A"_i
else
    print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j]+1, j)
    print ")"
```

$A_1\ A_2\ A_3\ A_4\ A_5\ A_6$: $((A_1\ A_2\ A_3)\ (A_4\ A_5)\ A_6)$?

$A_1\ (A_2\ A_3)$? $(A_1\ A_2)\ A_3$?

A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25





15 Dynamic Programming

- Assembly Lines Scheduling
- Steel Rod Cutting
- Matrix-Chain Multiplication (15.2)
  矩阵链相乘，或矩阵连乘问题
- Characteristics(Elements) of dynamic programming (15.3)
- Longest Common Subsequence (15.4)
- **Optimal binary search trees (15.5)**
  最优二叉搜索树

15.5 Optimal binary search trees

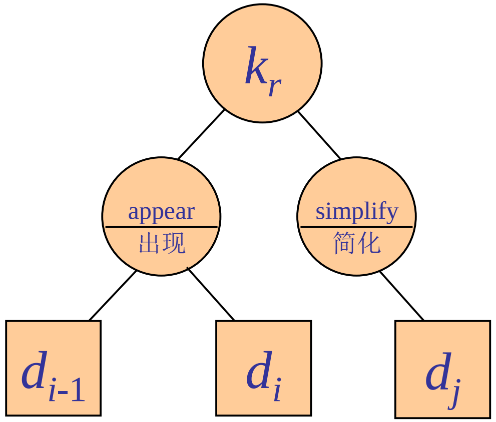
输入法的词库选择

15.5 Optimal binary search trees

Design a program to translate text from English to Chinese （翻译软件的词库的字典顺序如何设计）

15.5 Optimal binary search trees

Design a program to translate text from English to Chinese



生词表

字段1	字段2
aggregate	综合，总体
amortized	分摊，平摊
arbitrary	任意的，武断的
auxiliary	辅助的
binomial	二项的，二项式的
bog	沼泽，陷于泥沼
…	…
design	设计
…	…

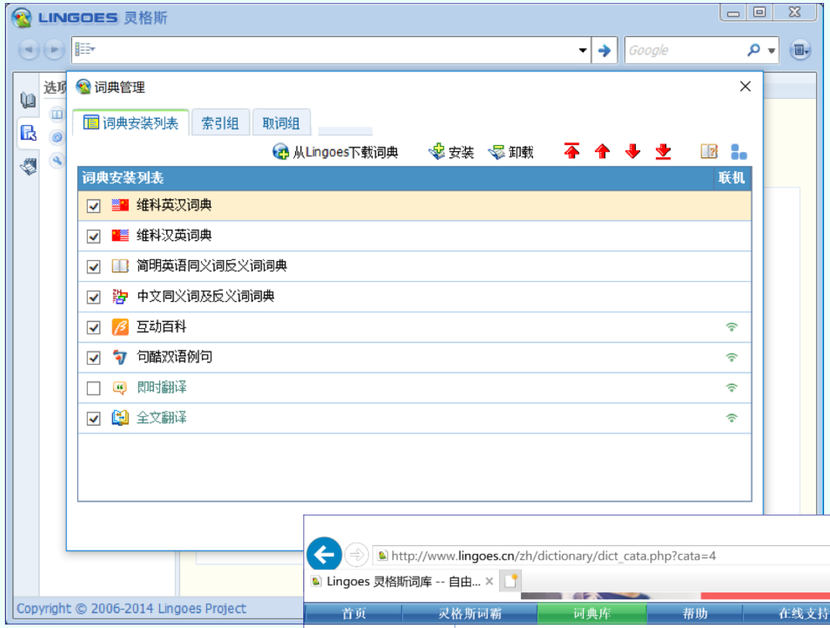
15.5 Optimal binary search trees

an $O(n)$ search time per occurrence by using any linear table operation

设计为线性表，每个单词查找的效率为 $O(n)$

生词表	
字段1	字段2
aggregate	综合，总体
amortized	分摊，平摊
arbitrary	任意的，武断的
auxiliary	辅助的
binomial	二项的，二项式的
bog	沼泽，陷于泥沼
…	…
design	设计
…	…

15.5 Optimal binary search trees



算法是软件的灵魂

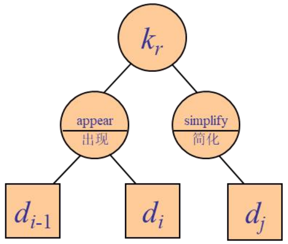
算法是企业生产力

15.5 Optimal binary search trees

- lookup operations: build a binary search tress (BST) with
  - n English words as keys
  - Chinses equivalents as satellite data

  为了高效查找，需要构建一棵二叉搜索树，树的每一个节点是一个单词，包括关键字（比如英语）及其从属数据（比如中文）。

  二叉搜索树，二叉排序树：二叉树，一个节点大于其左子树的所有节点，小于其右子树的所有节点。

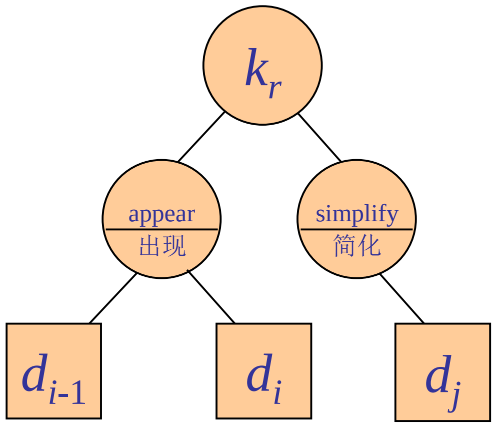


生词表

字段1	字段2
aggregate	综合，总体
amortized	分摊，平摊
arbitrary	任意的，武断的
auxiliary	辅助的
binomial	二项的，二项式的
bog	沼泽，陷于泥沼
...	...
design	设计
...	...

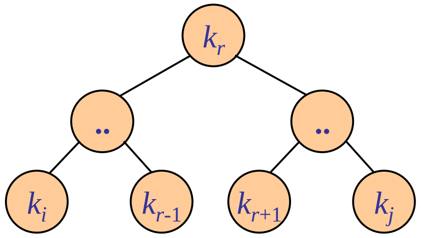
- Because we will search the tree for each individual word in the text, we want the total time spent searching to be as low as possible.
  对于文本中出现的每个单词，都需要搜索该二叉树，如何设计搜索树，使得总的搜索次数最少？
- an $O(\lg n)$ search time per occurrence by using any balanced BST.
  对于任何一个单词的搜索，使用平衡二分搜索法的时间为$O(\lg n)$.

15.5 Optimal binary search trees



A balanced BST…

平衡二叉搜索树



生词表	
字段1	字段2
aggregate	综合，总体
amortized	分摊，平摊
arbitrary	任意的，武断的
auxiliary	辅助的
binomial	二项的，二项式的
bog	沼泽，陷于泥沼
…	…

However, Words appear with different frequencies…?

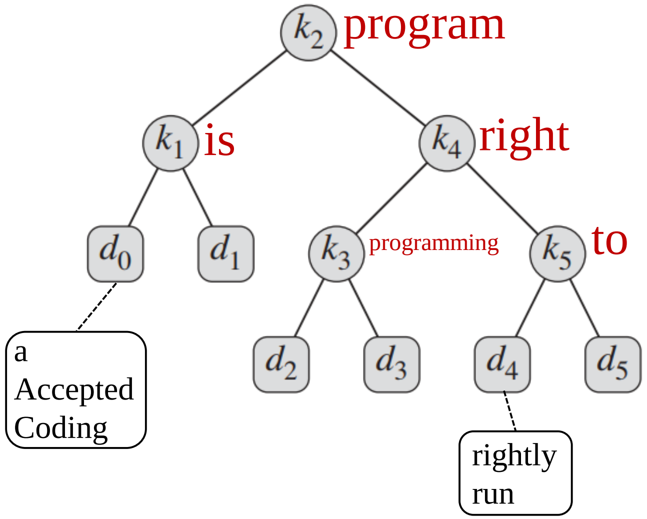
每个单词在文本中出现的频率（频次）不同，该如何设计二叉搜索树？

An example

给定一颗 BST T，对文本中的所有单词在树中进行搜索，访问的节点总数称为期望的搜索代价 (expected cost of a search in T)

$$\begin{aligned} \mathrm{E}[\text{search cost in } T] &= \sum_{i=1}^{n}(\mathrm{depth}_T(k_i)+1)\cdot p_i + \sum_{i=0}^{n}(\mathrm{depth}_T(d_i)+1)\cdot q_i \\ &= 1+\sum_{i=1}^{n}\mathrm{depth}_T(k_i)\cdot p_i + \sum_{i=0}^{n}\mathrm{depth}_T(d_i)\cdot q_i\ , \qquad (15.16) \end{aligned}$$

文本示例：Accepted Coding programming is rightly to program a right program to run a right program



node	word	depth	times 单词出现次数	probability 单词出现频次 p_i	contribution
d_0	a	2	2	4/15	12/15
	Accepted		1		
	Coding		1		
k_1	is : 是	1	1	1/15	2/15
k_2	program : 编程	0	3	3/15	3/15
k_3	programming : 编程	2	1	1/15	3/15
k_4	right : 正确	1	2	2/15	4/15
d_4	rightly	3	1	2/15	8/15
	run		1		
k_5	to : 去	2	2	2/15	6/15
d_{other}			0		
total					38/15

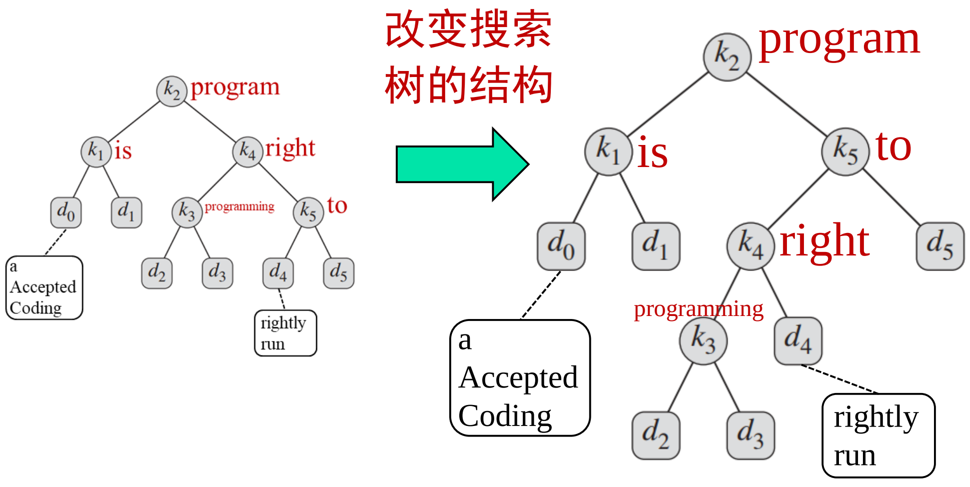
对示例问题，构建二叉搜索树（字典） BST T，其中红色的单词为字典中能查到的，黑色单词为字典中查不到的，则用搜索树 T 对文本进行全文搜索的搜索代价为 38/15。树不一样，搜索代价不同。

An example

给定一颗 BST T，对文本中的所有单词在树中进行搜索，访问的节点总数称为期望的搜索代价 (expected cost of a search in T)

$$\begin{aligned}\mathrm{E}[\text{search cost in } T] &= \sum_{i=1}^{n}(\mathrm{depth}_T(k_i)+1)\cdot p_i + \sum_{i=0}^{n}(\mathrm{depth}_T(d_i)+1)\cdot q_i \\ &= 1+\sum_{i=1}^{n}\mathrm{depth}_T(k_i)\cdot p_i + \sum_{i=0}^{n}\mathrm{depth}_T(d_i)\cdot q_i\ , \qquad (15.16)\end{aligned}$$

文本示例：Accepted Coding programming is rightly to program a right program to run a right program



node	word	depth	times 单词出现次数	probability 单词出现频次 p_i	contribution
d_0	a	2	2	4/15	12/15
	Accepted		1		
	Coding		1		
k_1	is : 是	1	1	1/15	2/15
k_2	program : 编程	0	3	3/15	3/15
k_3	programming : 编程	3	1	1/15	4/15
k_4	right : 正确	2	2	2/15	6/15
d_4	rightly	3	1	2/15	8/15
	run		1		
k_5	to : 去	1	2	2/15	4/15
d_{other}			0		
total					39/15

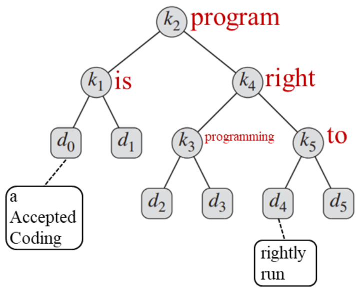
搜索树的结构改变，对文本的搜索代价变化。

An example

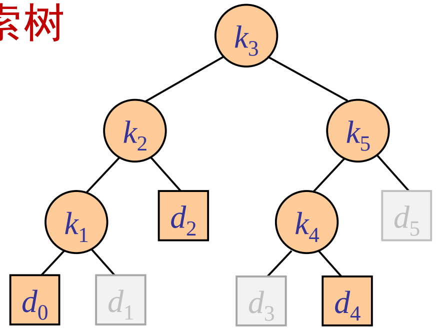
给定一颗 BST T，对文本中的所有单词在树中进行搜索，访问的节点总数称为期望的搜索代价 (expected cost of a search in T)

$$\begin{aligned}\mathrm{E}[\text{search cost in } T] &= \sum_{i=1}^{n}(\mathrm{depth}_T(k_i)+1)\cdot p_i + \sum_{i=0}^{n}(\mathrm{depth}_T(d_i)+1)\cdot q_i \\ &= 1+\sum_{i=1}^{n}\mathrm{depth}_T(k_i)\cdot p_i + \sum_{i=0}^{n}\mathrm{depth}_T(d_i)\cdot q_i\ , \qquad (15.16)\end{aligned}$$

文本示例：Accepted Coding programming is rightly to program a right program to run a right program



使用平衡二叉搜索树



node	word	depth	times 单词出现次数	probability 单词出现频次 p_i	contribution
d_0	a	3	2	4/15	16/15
	Accepted		1		
	Coding		1		
k_1	is : 是	2	1	1/15	3/15
k_2	program : 编程	1	3	3/15	6/15
k_3	programming : 编程	0	1	1/15	1/15
k_4	right : 正确	2	2	2/15	6/15
d_4	rightly	3	1	2/15	8/15
	run		1		
k_5	to : 去	1	2	2/15	4/15
d_{other}			0		
total					44/15

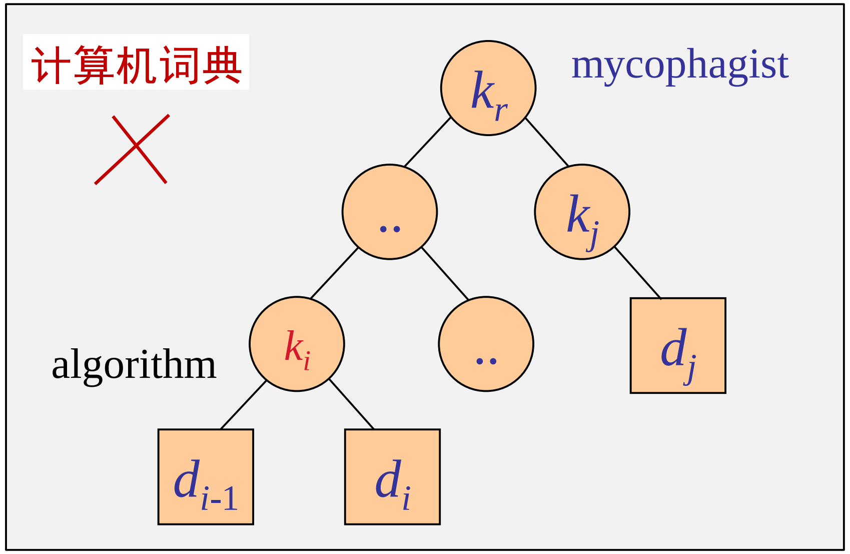
使用平衡二叉树搜索树，搜索代价为 44/15

怎样构建搜索树，使得搜索代价最小？

15.5 Optimal binary search trees

However, Words appear with different frequencies…?

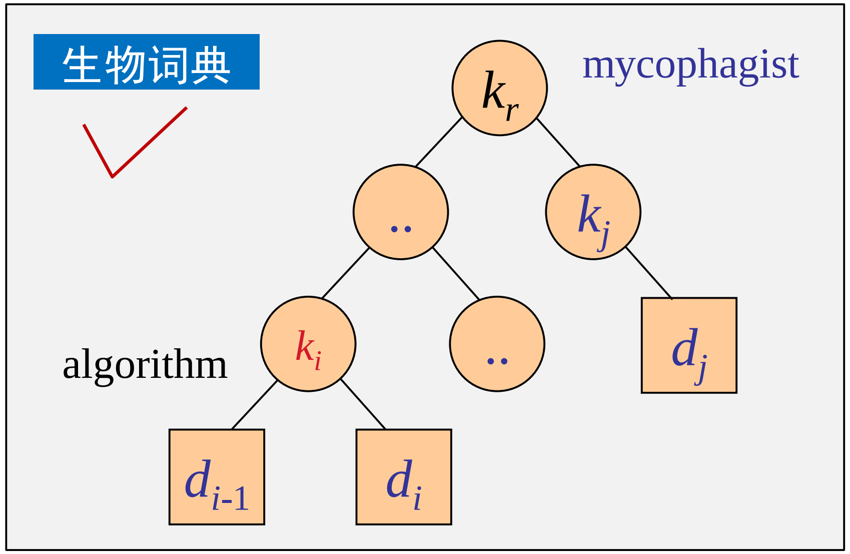
每个单词在文本中出现的频率（频次）不同，该如何设计二叉搜索树？

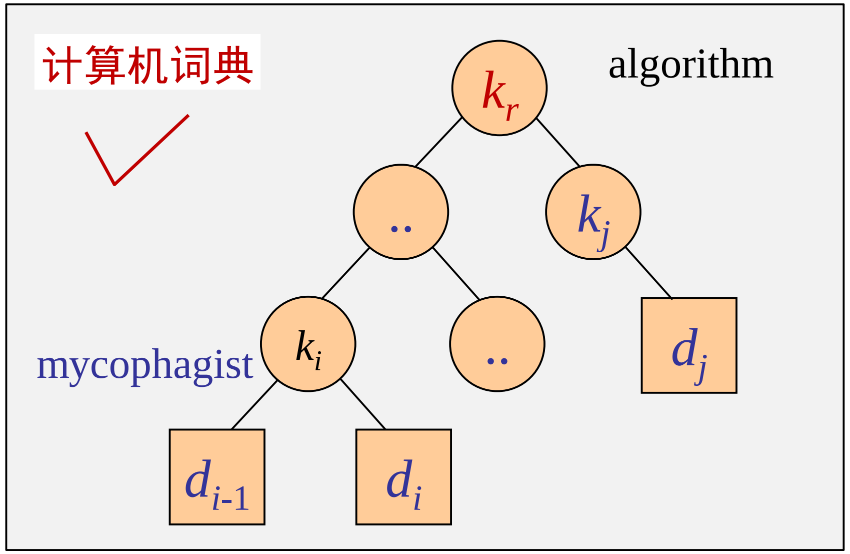


- It may be:
  "algorithm" (frequently used) appears far from the root;
  "mycophagist" (rarely used, 食菌者) appears near the root.
- Such an organization would slow down the translation, since # of nodes visited when searching for a key in a BST is 1+depth .

在翻译时，一个单词（节点）在搜索树BST中每次被访问时，需要访问 1+depth 次。对一个计算机类的词典，如果把出现频率高的单词，如"algorithm"，放在远离搜索树树根的节点处，把出现频率低的单词，如"mycophagist"，放在靠近树根的节点处，这种设计会让翻译效率很低（翻译速度很慢）。

15.5 Optimal binary search trees



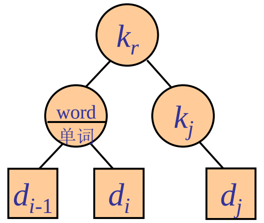


- Words appear with different frequencies
- It may be: “algorithm” (frequently used) appears far from the root; “mycophagist” (rarely used, 食菌者) appears near the root.
- Such an organization would slow down the translation, since # of nodes visited when searching for a key in a BST is 1+depth .
- We want words that occur frequently in the text to be placed nearer the root.

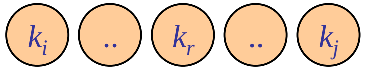
文本中出现频率高的单词，应该放在BST中靠近树根处

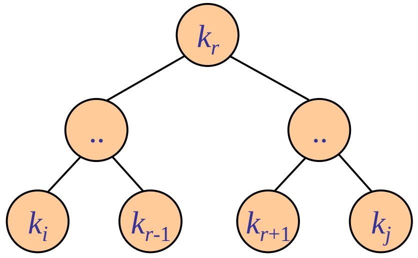
15.5 Optimal binary search trees

- Moreover, there may be words in the text for which there is no Chinese translation, and such words might not appear in the BST at all.
文本中有些英语单词没有对应的汉语译文，即这些英语单词不出现在二叉搜索树BST的"词典"中（置于BST的树叶）



- How do we organize a BST so as to minimize the number of nodes visited in all searches, given that we know how often each word occurs?
设已知每个单词出现的概率，如何设计一颗二叉搜索树BST，使得在对文本中每个单词的所有搜索中，被访问的节点的总数最少？



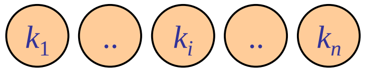


15.5 Optimal binary search trees

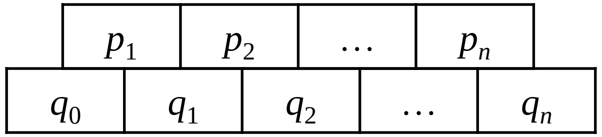
BST: Given a sequence $K = <k_1, k_2, ..., k_n>$ of n distinct keys in sorted order ($k_1 < k_2 < \cdots < k_n$), how to build a BST?

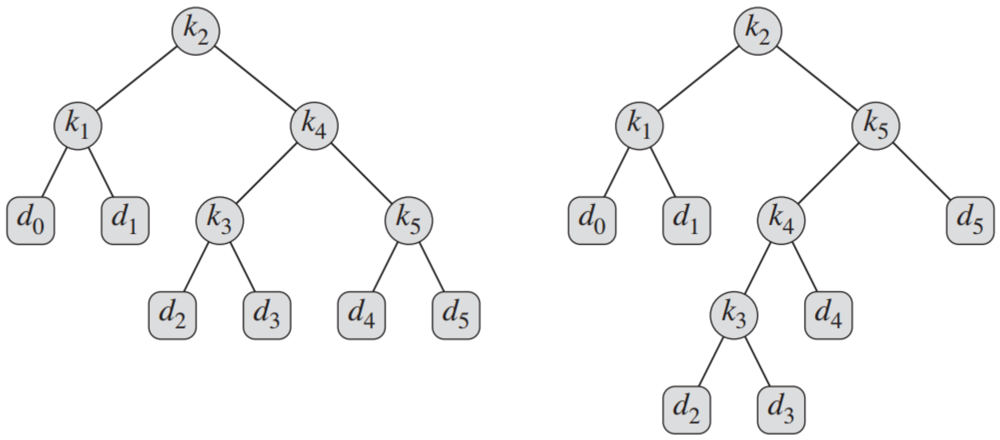
已知 n 个关键字，其值的集合为 $K = <k_1, k_2, ..., k_n>$，($k_1 < k_2 < \cdots < k_n$)，如何构建一颗二叉搜索树

- 关键字 k_i 在文本中的搜索概率（频率）为 p_i，（文本总共有 M 个单词，k_i 出现了 I 次，则 $p_i = I / M$）
- 文本中有些单词在 K 中不存在，称其为虚关键字，有 n+1 类这样的单词，记为 $<d_0, d_1, ..., d_n>$，其出现频率分别为 $q_0, q_1, ..., q_n$，其中，d_0 对应字典序小于 k_1 的一类单词；d_n 对应字典序大于 k_n 的一类单词；对 $1 \le i \le n\text{-}1$，$d_i : k_i < d_i < k_{i+1}$
- BST如图所示，每个关键字 k_i 是内节点，虚关键字 d_i 是树叶



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



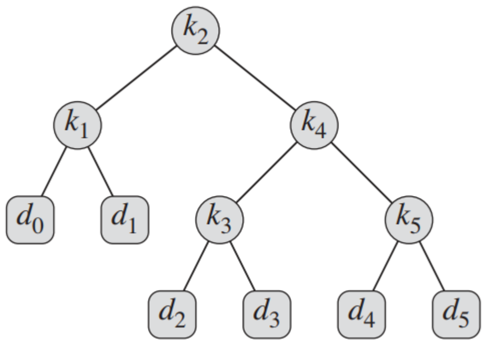


15.5 Optimal binary search trees

- 在BST中查找文本的每个单词，有两种情况：能找到（内节点），不能找到（树叶节点）

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1 \qquad (15.15)$$

- 给定一颗 BST T，对文本中的所有单词在树中进行搜索，访问的节点总数称为期望的搜索代价 (expected cost of a search in T)



depth_T 表示一个节点在树 T 中的高度（深度），树根高度为0

$$\begin{aligned} \text{E[search cost in } T] &= \sum_{i=1}^{n} (\text{depth}_T(k_i)+1) \cdot p_i + \sum_{i=0}^{n} (\text{depth}_T(d_i)+1) \cdot q_i \\ &= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \text{depth}_T(d_i) \cdot q_i \,, \qquad (15.16) \end{aligned}$$

15.5 Optimal binary search trees

对 BST T，

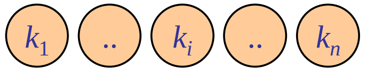
期望的搜索代价

E[search cost in T]

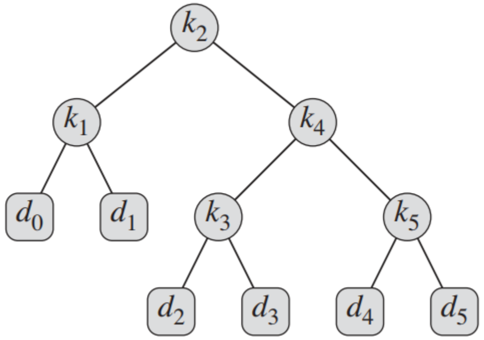
$$= \sum_{i=1}^{n} (\text{depth}_T(k_i)+1)\cdot p_i + \sum_{i=0}^{n} (\text{depth}_T(d_i)+1)\cdot q_i$$

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

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



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



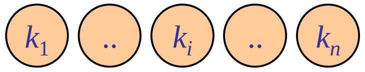
问题 → BST T → 搜索代价 E[T]

15.5 Optimal binary search trees

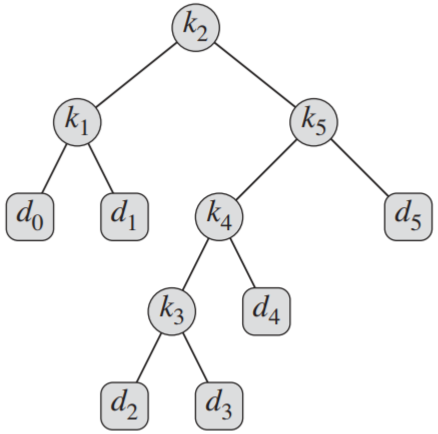
对 BST T，

期望的搜索代价

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



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

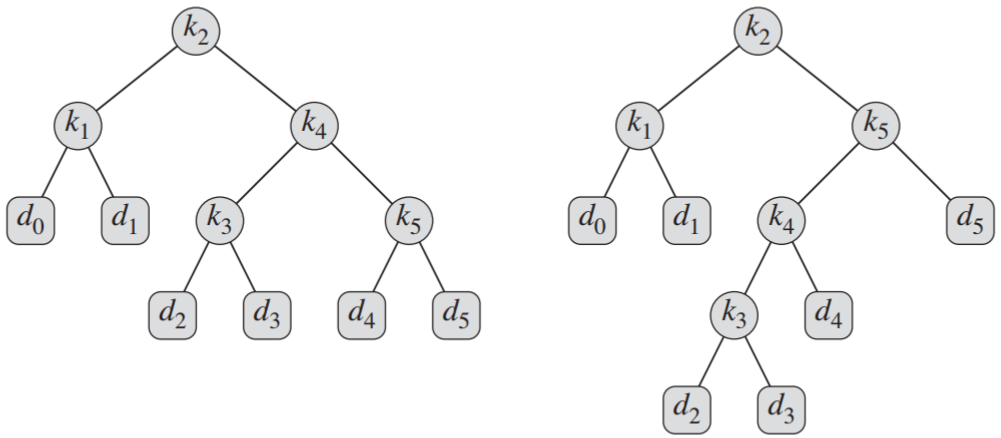


node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	3	0.05	0.20
k_4	2	0.10	0.30
k_5	1	0.20	0.40
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	4	0.05	0.25
d_3	4	0.05	0.25
d_4	3	0.05	0.20
d_5	2	0.10	0.30
Total			2.75

问题 → BST T → 搜索代价 E[T]

15.5 Optimal binary search trees

- Optimal BST 最优二叉搜索树：输入各个关键字的概率集，期望搜索代价最小的树



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

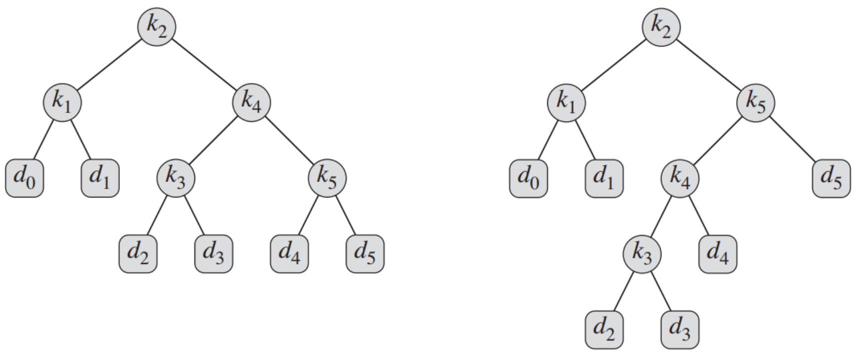
- 如何构造最优二叉搜索树 OBST？

Intuitively, the overall height is smallest; the key with the greatest probability at the root.

直观上看，树的高度应最小；出现概率（频率）最大的关键字应作为树根。

15.5 Optimal binary search trees

- Optimal BST 最优二叉搜索树：输入各个关键字的概率集，期望搜索代价最小的树



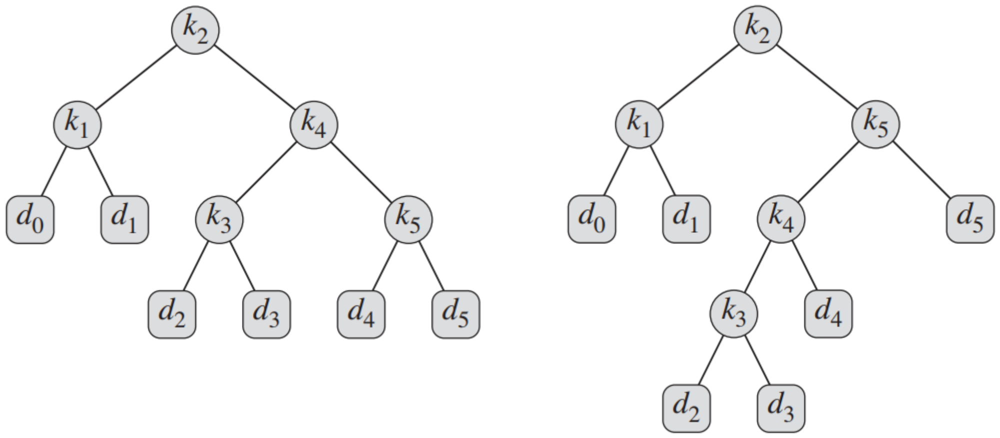
(a) cost: 2.80 (b) cost: 2.75

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

- 图(b)是一颗OBST，其搜索代价为 2.75
  - An Optimal BST is not necessarily a tree whose overall height is smallest.
    不一定要求树的高度最小
  - Nor can we necessarily construct an Optimal BST by always putting the key with the greatest probability at the root. (The lowest expected cost of any BST with k_5 (the greatest probability) at the root is 2.85.)
    不一定将概率最大的 key 放在树根，如…

15.5 Optimal binary search trees

- Exhaustive checking of all possibilities fails to yield an efficient algorithm.
  - ALS, RodCut, MCM



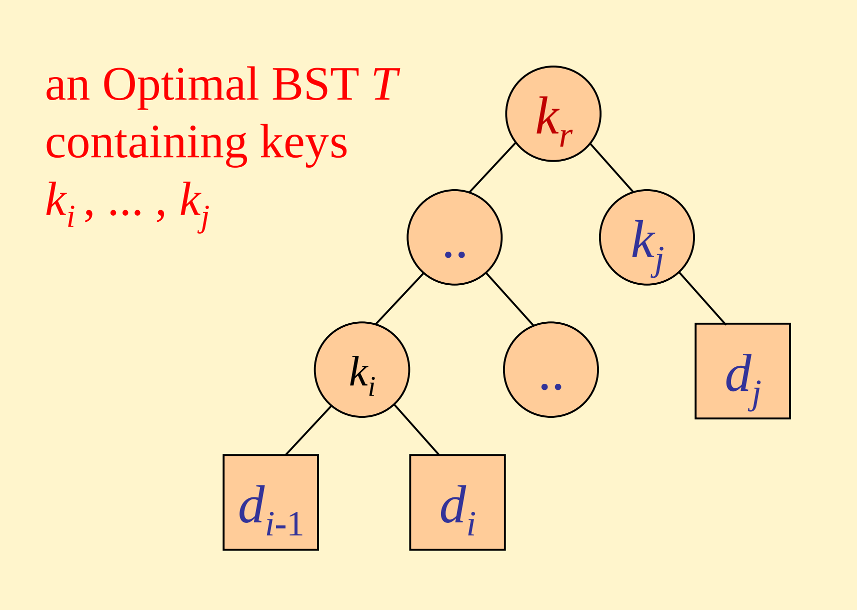
- The # of BST with n nodes is $\Omega(4^n/n^{3/2})$ 〔Problem 12-4〕.

- Not surprisingly, we will solve this problem with dynamic programming.

Step 1: The structure of an Optimal BST（最优子结构）

- Start with an observation about subtrees.
- Consider any subtree of a BST
  - It must contain keys in a contiguous range k_i , ..., k_j , for some $1 \le i \le j \le n$.
  - In addition, the subtree must also have as its leaves the dummy keys d_{i-1} , ..., d_j .
- Optimal substructure?

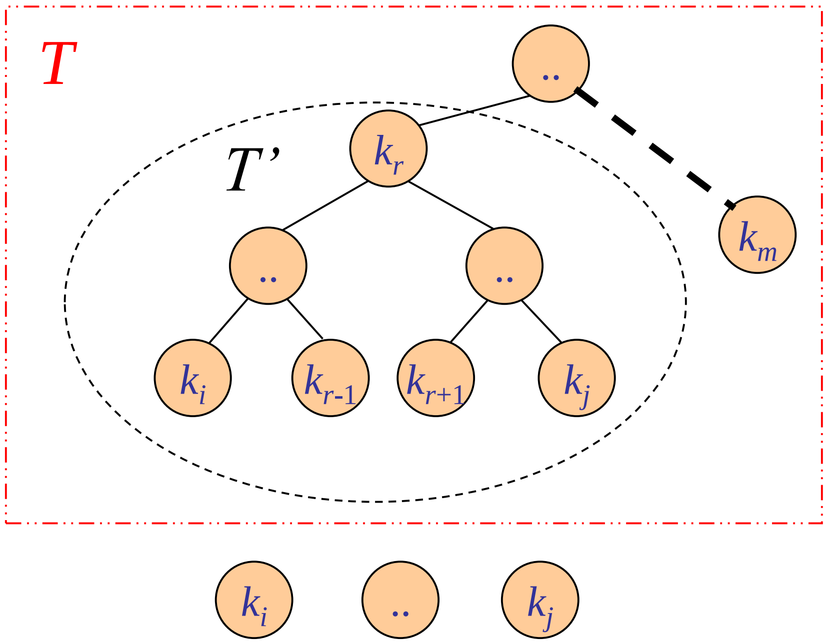
考虑包括节点 k_i , ..., k_j , $1 \le i \le j \le n$,
的一颗 BST T，是否有最优子结构？



Step 1: The structure of an Optimal BST（最优子结构）

Optimal substructure: If an Optimal BST T has a subtree T' containing keys k_i, ..., k_j , then this subtree T' must be optimal as well for the subproblem with keys k_i , ..., k_j and dummy keys d_{i-1} , ..., d_j .

最优子结构：设 T' 为 OBST T 的一个子树， T' 包含keys k_i, ..., k_j , 那么 T' 是子问题〔关于keys k_i, ..., k_j 和dummy keys d_{i-1} , ..., d_j〕的 OBST



T : search tree of k_i , ... , k_m

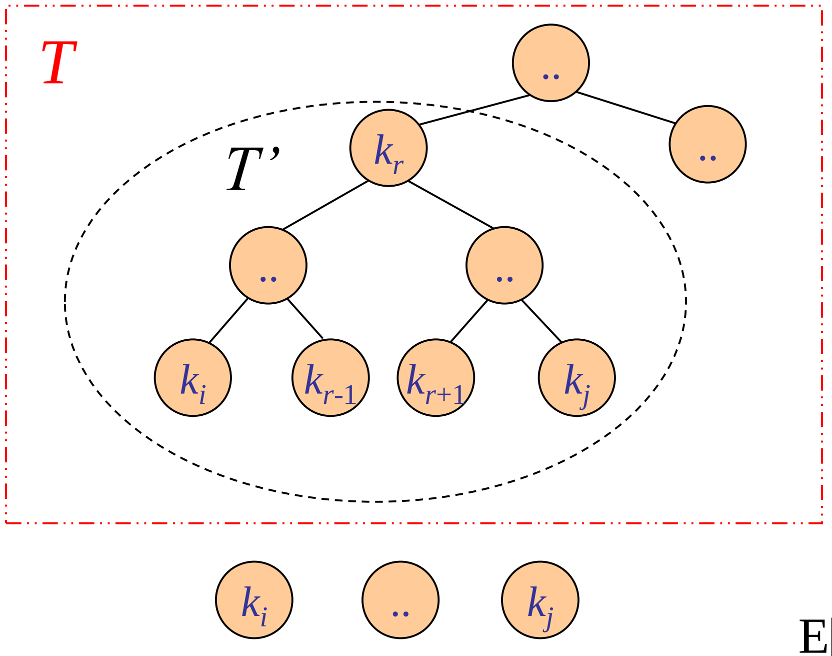
T' : search tree of k_i , ... , k_j

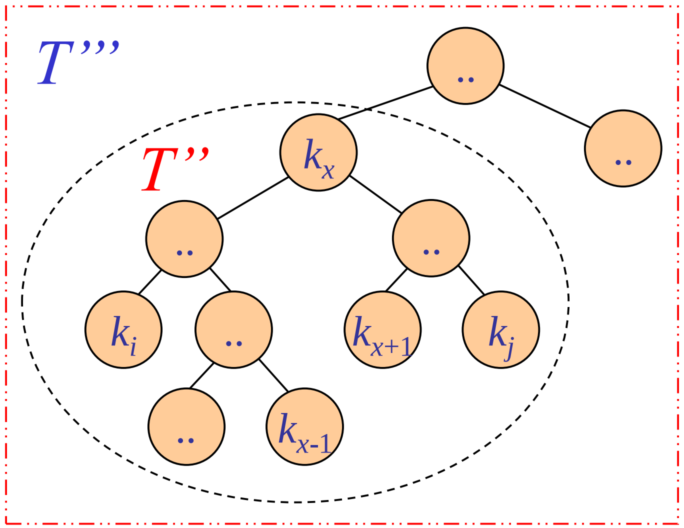
T is OBST ➔ T' is OBST

Step 1: The structure of an Optimal BST（最优子结构）

Idea of Proof: Cut-and-paste.　证明思想：剪切粘贴法

设 BST T 的搜索代价最小，T' 是 T 的搜索子树。如果 T 的一颗搜索子树 T'' 的搜索代价比 T' 的搜索代价更小，在搜索树 T 中把 T' 换成 T''，得到一颗新的搜索树 T'''，其搜索代价比 T 更小，与假设矛盾。

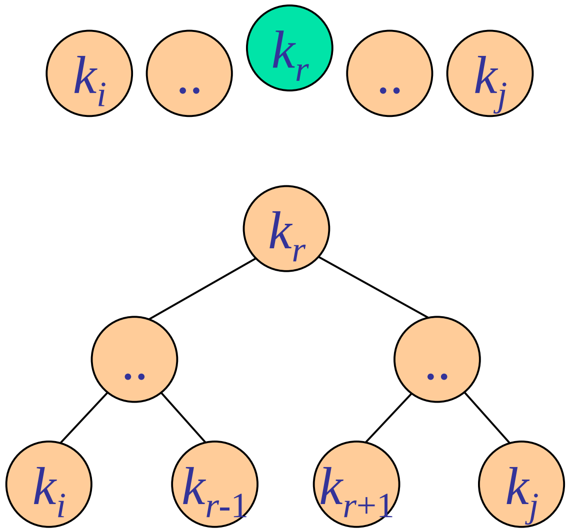




$$E[\text{cost}(T)] = 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \text{depth}_T(d_i) \cdot q_i$$

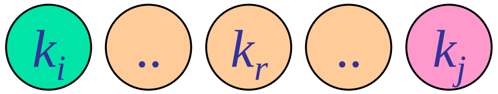
Step 1: The structure of an Optimal BST（最优子结构）

- 通过子问题的最优解构造原问题的最优解
- 给定 keys $k_i, \ldots, k_j$, 设某个 k_r $(i \le r \le j)$ 是最优搜索子树的根，子问题包括由 $k_i, \ldots, k_{r-1}$ 构成的搜索子树，和由 $k_{r+1}, \ldots, k_j$ 构成的子树
- 需要检验所有的 k_r, $i \le r \le j$, 并求相应子树的 OBST，然后求出原问题的 OBST

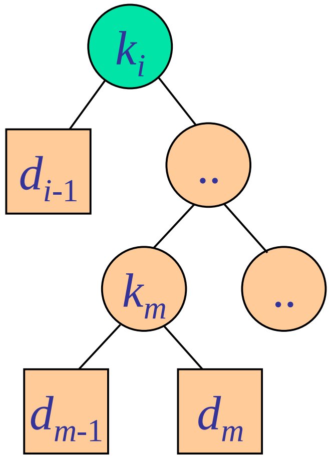


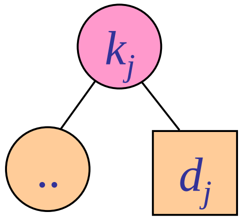
Step 1：The structure of an Optimal BST（最优子结构）

- 特例，空子树：不包括任何 key



- 给定 keys k_i , ..., k_j ,
  - 如果 k_i 是搜索子树的根，其左子树没有 key，此时，值小于 k_i 的值的所有 dummy keys 记为 $d_{i\text{-}1}$ ，表示左子树。
  - k_j 是搜索子树的根，同理，其右子树记为 d_j ，其值大于 k_j 的值。

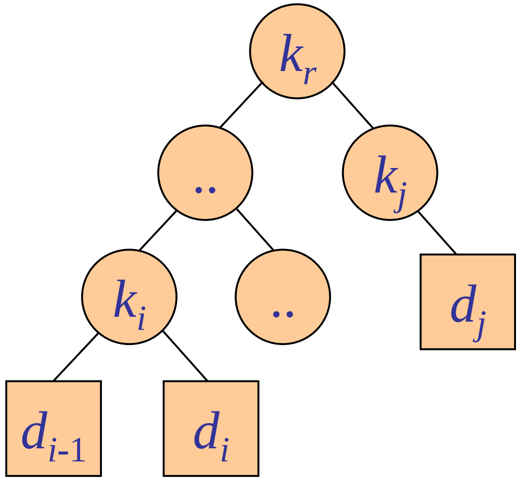


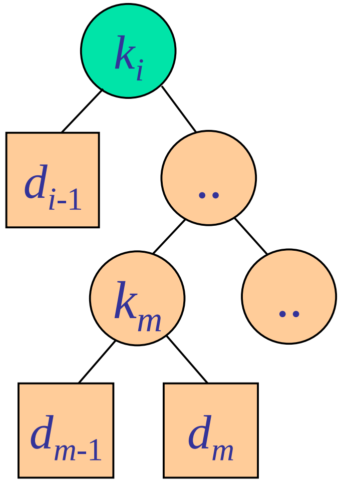


Step 2: A recursive solution（递归解）

子问题：给定 keys k_i, ..., k_j, $i \geq 1$, $j \leq n$, and $j \geq i$-1，设求一颗 OBST（当 $j = i$-1，OBST 只有一个节点，即 dummy key d_{i-1}.）

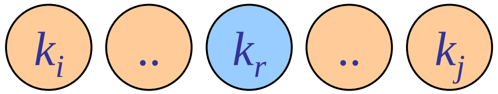
- $e[i, j]$: 一颗 OBST 的搜索代价，**最优值**
- 原问题为 $e[1, n]$
- 当 $j = i$-1， OBST 只有一个节点 d_{i-1} , $e[i, i\text{-}1] = q_{i-1}$
- 当 $j \geq i$?

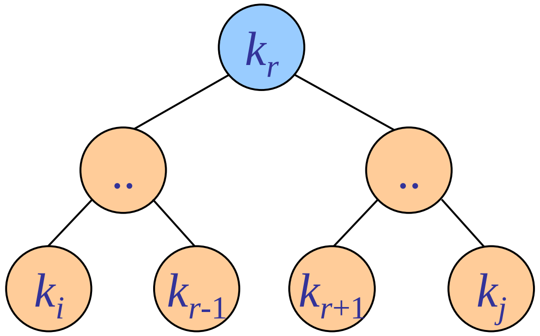




Step 2: A recursive solution（递归解）

当 $j \geq i$, 选树根节点 k_r 子，求子问题：包括 keys k_i , ..., k_{r-1}的OBST，作为其左子树；包括 keys k_{r-1} , ..., k_j 的OBST，作为其右子树。

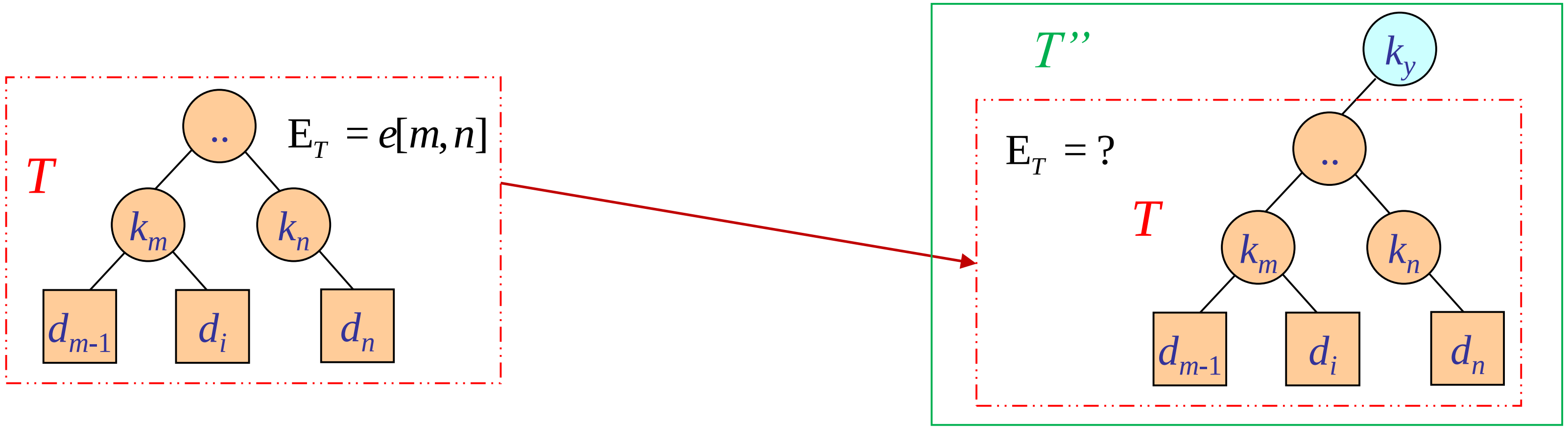




Step 2: A recursive solution（递归解）

当一个搜索子树 T 成为另一个节点 k_y 的子树时（如图），对 T 的搜索代价如何变化？

- 子树 T 的每一个节点在 T'' 中深度增加 1，其搜索代价增加值为其所有节点的概率和，如式：



所有节点的概率和：

$$w[i,j] = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l \qquad (15.17)$$

$$E_T = \sum_{x=m}^{n} (\text{depth}(k_x)+1+1)\cdot p_i + \sum_{x=m\text{-}1}^{n} (\text{depth}(d_x)+1+1)\cdot q_x$$

$$= \sum_{x=m}^{n} (\text{depth}(k_x)+1)\cdot p_i + \sum_{x=m-1}^{n} (\text{depth}(d_x)+1)\cdot q_x + \sum_{x=m}^{n} p_i + \sum_{x=m-1}^{n} q_x$$

$$= e[m,n] + w[m,n]$$

增量为 $w[m, n]$

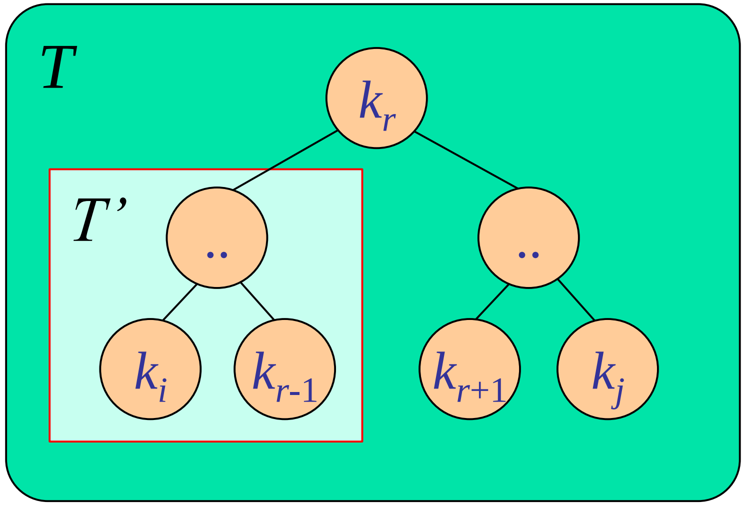
Step 2: A recursive solution（递归解）

OBST T 与 OBS-subTree T' 的关系：

如果 k_r 是 keys k_i , … , k_j 的一颗OBST 的树根，则

$\boxed{e[i, j]}$ = p_r+ ($\boxed{e[i, r\text{-}1]}$ + $w[i, r\text{-}1]$)

+ ($e[r+1, j]$ + $w[r+1, j]$) ?



因为 $w[i, j] = w[i, r\text{-}1] + p_r + w[r+1, j]$

$$\left(w[i, r-1] = \sum_{l=i}^{r-1} p_l + \sum_{l=i-1}^{r-1} q_l \quad , \quad w[r+1, j] = \sum_{l=r+1}^{j} p_l + \sum_{l=r}^{j} q_l \right)$$

所以

$$e[i, j] = e[i, r\text{-}1] + e[r+1, j] + w[i, j] \qquad (15.18)$$

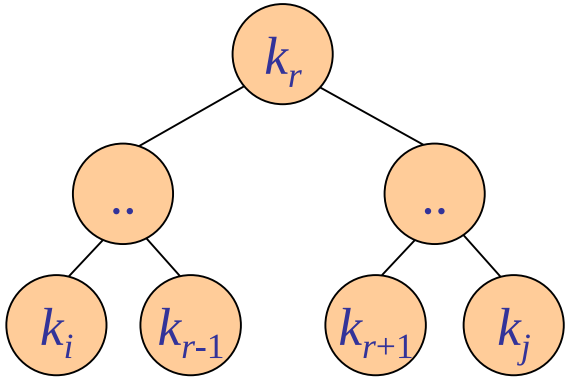
$e[i, j]$: 问题的最优值；$e[i, r\text{-}1]$， $e[r+1, j]$，子问题的最优值。

递归方程 (15.18) 假定已知节点 k_r ，但事实上节点未知，怎么办？

Step 2: A recursive solution（递归解）

- 遍历所有的 k_r

$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i-1, \\ \min\limits_{i \le r \le j}\{e[i, r-1] + e[r+1, j] + w[i, j]\} & \text{if } i \le j. \end{cases} \quad (15.19)$$

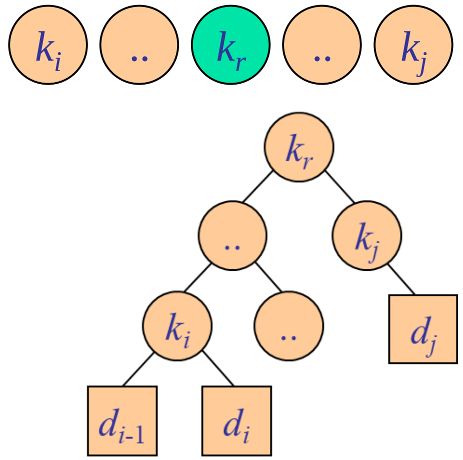


- 定义辅助数组 $root[i, j] = r$， 用于记录一颗 OBST 的树根（记录OBST的结构），此时 k_r 是树根。

Step 3: Computing the expected search cost （最优值）

$$A_i \,..\, A_k A_{k+1} \,..\, A_j$$

$$m[i,j] = \begin{cases} 0 & \text{, if } i = j, \\ \min\limits_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \}, & \text{if } i < j. \end{cases} \qquad (15.12)$$

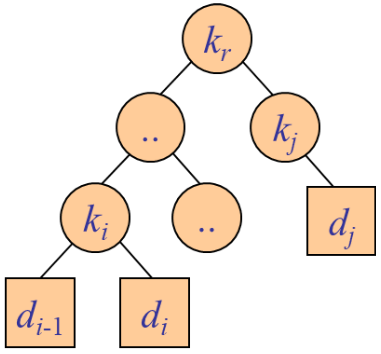


$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1, \\ \min\limits_{i \le r \le j} \{ e[i,r-1] + e[r+1,j] + w[i,j] \} & \text{if } i \le j. \end{cases} \qquad (15.19)$$

- Similarity: OBST and matrix-chain multiplication.
- A direct, recursive implementation would be as inefficient ?
- Store the $e[i, j]$ values in a table $e[1..\, n+1, 0..\, n]$.
  - The first index runs to n+1, in order to have a subtree containing only d_n , need to compute and store $e[n+1, n]$. The second index starts from 0, in order to have a subtree containing only d_0 , need to compute and store $e[1, 0]$. 数组的索引范围略有区别
- $root[i, j]$, recording the root of the subtree containing keys k_i , ..., k_j .

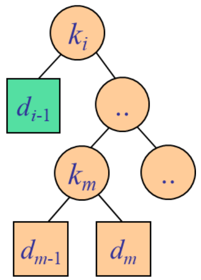
Step 3: Computing the expected search cost （最优值）

继续优化 $e[i, j\,] = e[i, r\text{-}1] + e[r{+}1, j] + w[i, j]$ (15.18)



- 输入：每个 key 出现的频率

	p_i	p_{i+1}	…	$p_{j\text{-}1}$	p_j
$q_{i\text{-}1}$	q_i	q_{i+1}	…	$q_{j\text{-}1}$	q_j



- 在式(15.18)中，每次计算 $e[i, j]$ 时，无需 $O(n)$ 遍历计算 $w[i, j]$，…
- 基本情况，只有一个假节点，$w[i, i\text{-}1] = q_{i\text{-}1}$ for $1 \le i \le n$.
- 当 $j \ge i$,

$$w[i, j] = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l = w[i, j-1] + p_j + q_j \qquad (15.20)$$

- 计算每个 $w[i, j]$，从 $O(n)$ 从优化为 $O(1)$

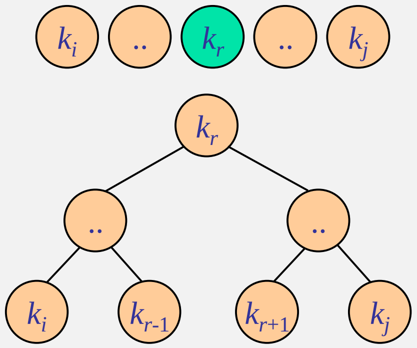
	p_i	p_{i+1}	…	$p_{j\text{-}1}$	p_j
$q_{i\text{-}1}$	q_i	q_{i+1}	…	$q_{j\text{-}1}$	q_j

Step 3: Computing the expected search cost （最优值）

$$e[i,j]=\begin{cases} q_{i-1} & \text{if } j=i-1,\\ \min\limits_{i\le r\le j}\{e[i,r-1]+e[r+1,j]+w[i,j]\} & \text{if } i\le j.\end{cases} \quad (15.19)$$

$$w[i,j]=\sum_{l=i}^{j} p_l+\sum_{l=i-1}^{j} q_l = w[i,j-1]+p_j+q_j \quad (15.20)$$

```
OBST(p, q, n)
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
```



VS

$$(\ \underline{(A_i(A_{i+1}\ldots)(\ldots)\ldots A_k)}\ (A_{k+1}\ldots\ A_{j-1}A_j)\ )$$

$$m[i,j]=\begin{cases} 0 & , \quad \text{if } i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & \text{if } i<j.\end{cases} \quad (15.12)$$

```
MCM-DP(p)
n ← length[p] – 1
for i ← 1 to n
    m[i, i] ← 0
for l ← 2 to n            // l is the chain length.
    for i ← 1 to n - l + 1
        j ← i + l – 1
        m[i, j] ← ∞
        for k ← i to j - 1
            q ← m[i, k]+m[k+1, j]+p_{i-1}p_kp_j
            if q < m[i, j]
                m[i, j] ← q
                s[i, j] ← k
return m and s
```

Step 3: Computing the expected search cost （最优值）

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1, \\ \min\limits_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + w[i,j]\} & \text{if } i \le j. \end{cases} \quad (15.19)$$

$$w[i,j] = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l = w[i,j-1] + p_j + q_j \quad (15.20)$$

```
OBST(p, q, n)
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        // 求 l 个元素的 Opti-BST
    for i ← 1 to n-l+1      // ?1
        j ← i+l-1           // ?2
        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
```

?1

$e[i, j]$:

l 个元素的 Opti-BST 的 cost

$i = 1, \ j = l,$

$i = 2, \ j = l+1,$

…

$i = x, \ j = n,$

$n-x+1 = l \ \ => \ x = n-l+1$

?2

$j-i+1 = l \ \ ==> \ j = i+l-1$

Step 3: Computing the expected search cost （最优值）

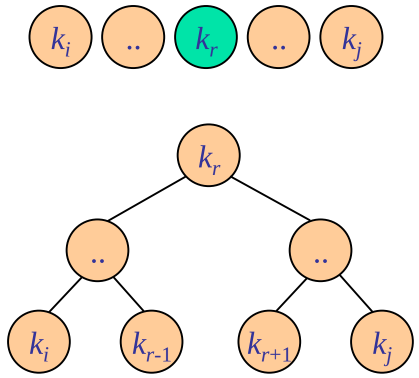
$$e[i,j]=\begin{cases} q_{i-1} & \text{if } j=i-1, \\ \min\limits_{i\le r\le j}\{e[i,r-1]+e[r+1,j]+w(i,j)\} & \text{if } i\le j. \end{cases}$$

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

```
OBST(p, q, n)
1  for i ← 1 to n+1
2       e[i, i-1] ← q_{i-1}
3       w[i, i-1] ← q_{i-1}
4  for l ← 1 to n        // 求 l 个元素的Opti-BST
5       for i ← 1 to n-l+1
6            j ← i+l-1
7            e[i, j] ← ∞
8            w[i, j] ← w[i, j-1]+p_j+q_j
9            for r ← i to j
10                t ← e[i, r-1]+e[r+1, j]+w[i, j]
11                if t < e[i, j]
12                     e[i, j] ← t
13                     root[i, j] ← r
14 return e and root
```

Innermost for loop, in lines 9–13, tries each candidate index r to determine which key k_r to use as the root of an OBST containing keys k_i , ..., k_j.

对包含 k_i , ..., k_j 的最优 BST，遍历每一个 k_r 作为树根，…



Step 3: Computing the expected search cost （最优值）

$$e[i,j]=\begin{cases} q_{i-1} & \text{if } j=i-1,\\ \min\limits_{i\le r\le j}\{e[i,r-1]+e[r+1,j]+w[i,j]\} & \text{if } i\le j.\end{cases} \tag{15.19}$$

$$w[i,j]=\sum_{l=i}^{j} p_l+\sum_{l=i-1}^{j} q_l = w[i,j-1]+p_j+q_j \tag{15.20}$$

```
OBST(p, q, n)
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      // 求 l 个元素的Opti-BST
      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
```

e (i \ j)	0	1	2	3	4	5
1	0.05	0.45	0.90	1.25	1.75	2.75
2		0.10	0.40	0.70	1.20	2.00
3			0.05	0.25	0.60	1.30
4				0.05	0.30	0.90
5					0.05	0.50
6						0.10

w (i \ j)	0	1	2	3	4	5
1	0.05	0.30	0.45	0.55	0.70	1.00
2		0.10	0.25	0.35	0.50	0.80
3			0.05	0.15	0.30	0.60
4				0.05	0.20	0.50
5					0.05	0.35
6						0.10

root (i \ j)	1	2	3	4	5
1	1	1	2	2	2
2		2	2	2	4
3			3	4	5
4				4	5
5					5

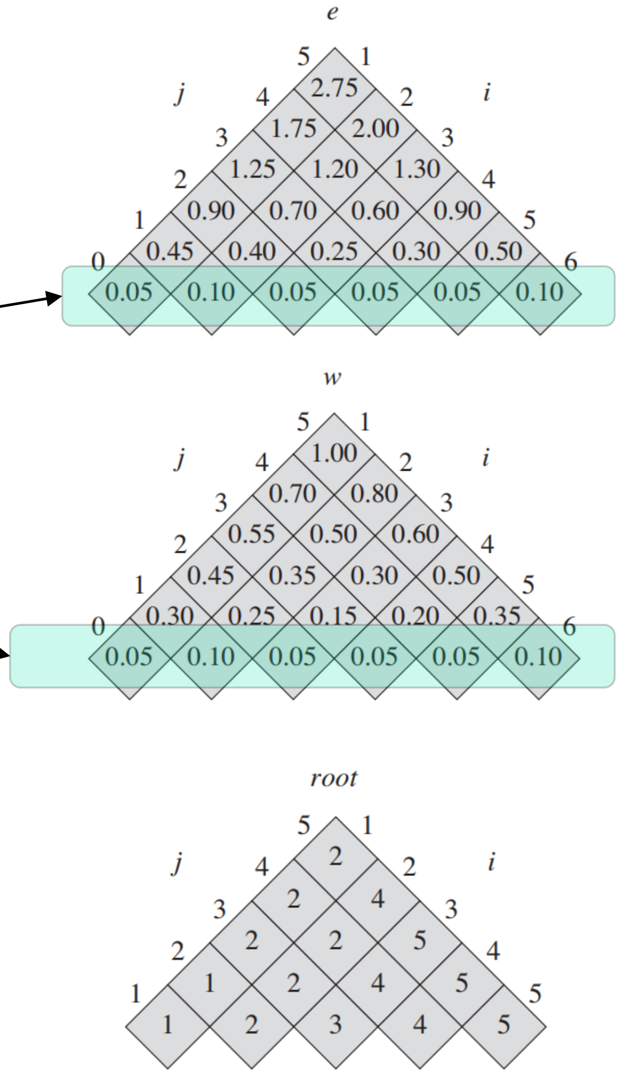
Step 3: Computing the expected search cost （最优值）

$$e[i,j]=\begin{cases} q_{i-1} & \text{if } j=i-1,\\ \min\limits_{i\le r\le j}\{e[i,r-1]+e[r+1,j]+w[i,j]\} & \text{if } i\le j.\end{cases} \qquad (15.19)$$

$$w[i,j]=\sum_{l=i}^{j} p_l+\sum_{l=i-1}^{j} q_l = w[i,j-1]+p_j+q_j \qquad (15.20)$$

OBST(p, q, n)

```
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        // 求 l 个元素的Opti-BST
    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
```



Step 3: Computing the expected search cost （最优值）

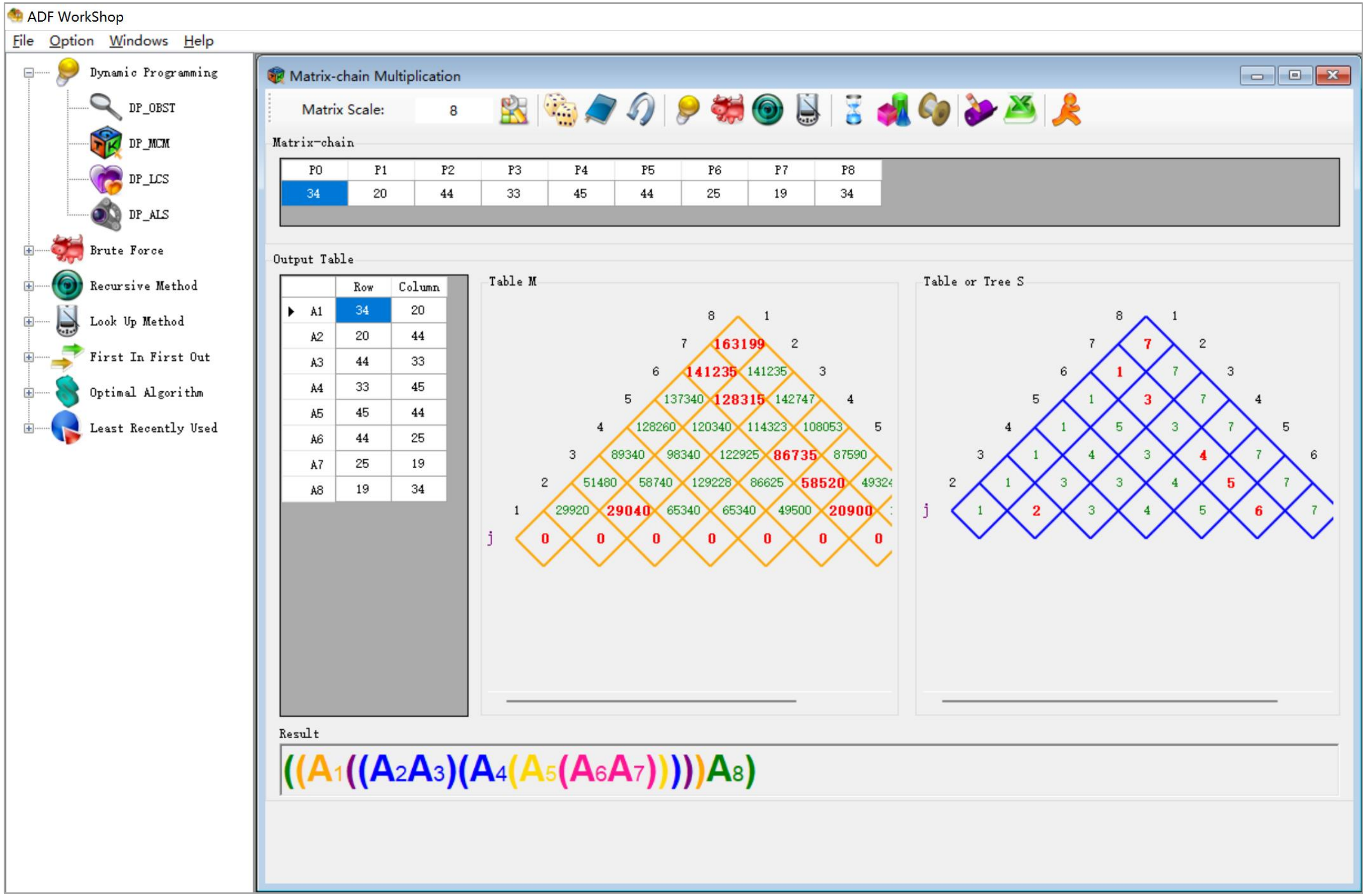
$$e[i,j]=\begin{cases} q_{i-1} & \text{if } j=i-1, \\ \min\limits_{i\le r\le j}\{e[i,r-1]+e[r+1,j]+w[i,j]\} & \text{if } i\le j. \end{cases} \tag{15.19}$$

$$w[i,j]=\sum_{l=i}^{j}p_l+\sum_{l=i-1}^{j}q_l=w[i,j-1]+p_j+q_j \tag{15.20}$$

```
OBST(p, q, n)
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      // 求 l 个元素的Opti-BST
      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
```

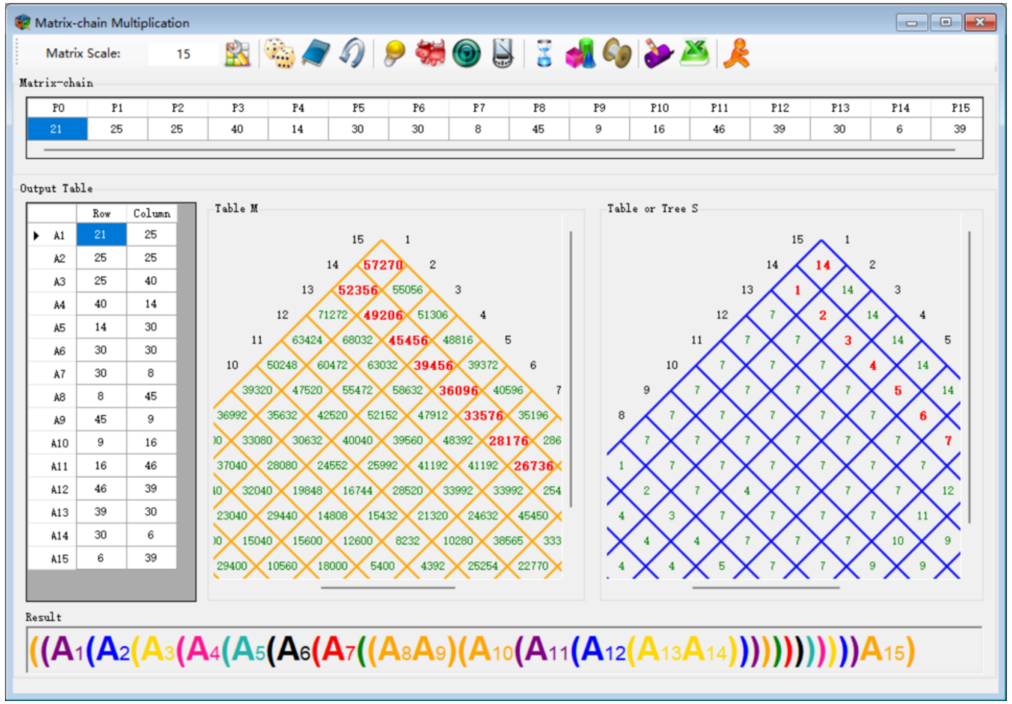
Running times ?

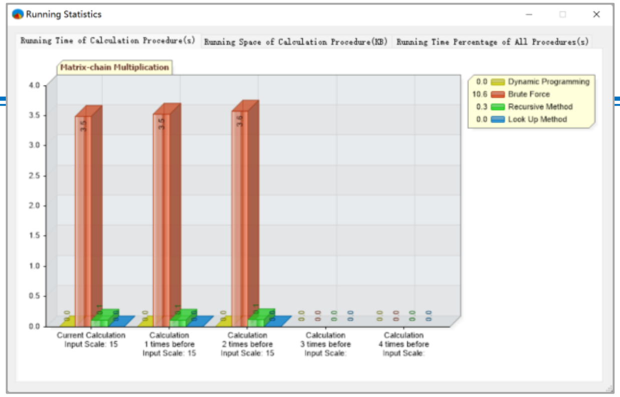
Exercises ADF_WorkShop (video-demo)

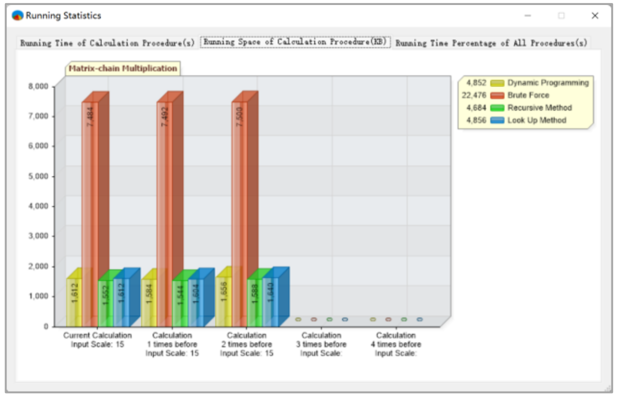


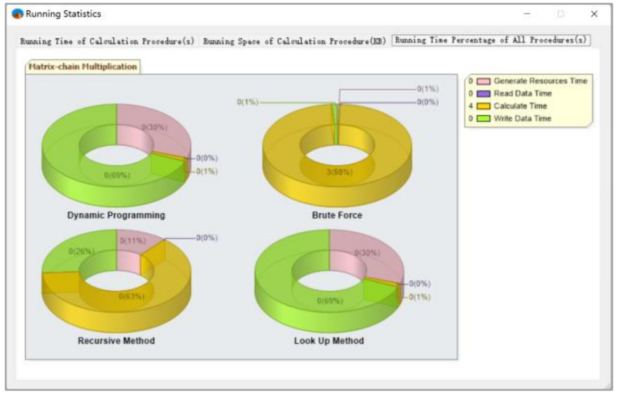
Exercises

ADF_WorkShop



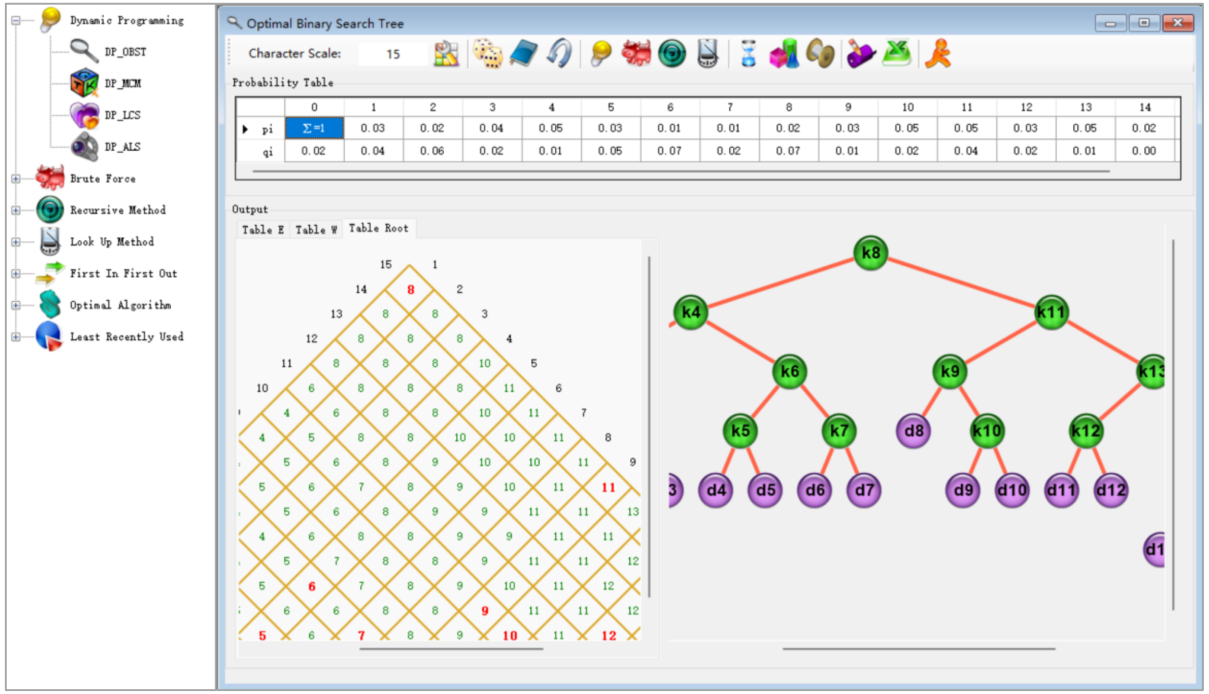


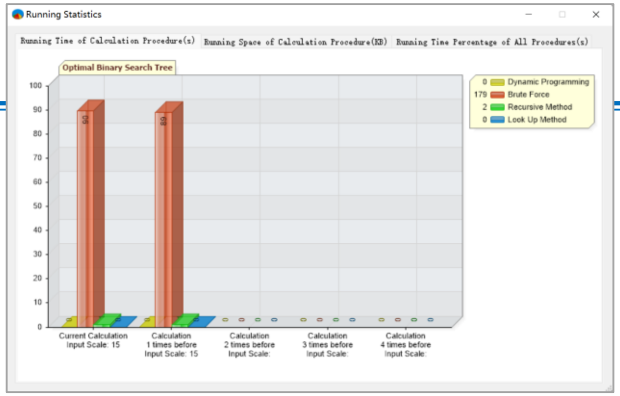


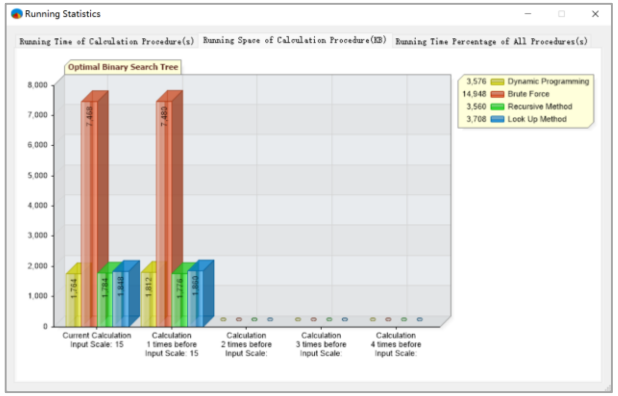


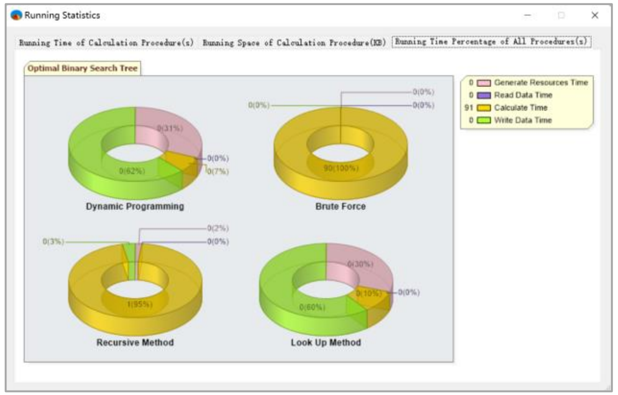
Exercises

ADF_WorkShop





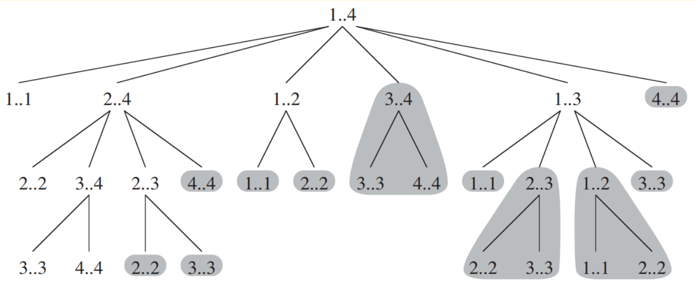




Exercise-1

$$(\ (A_i(A_{i+1}\ldots)(\ldots)\ldots A_k)\ (A_{k+1}\ldots\ A_{j-1}A_j)\ )$$

$$m[i,j]=\begin{cases}0 & \text{, if } i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & \text{if } i<j.\end{cases} \qquad (15.12)$$

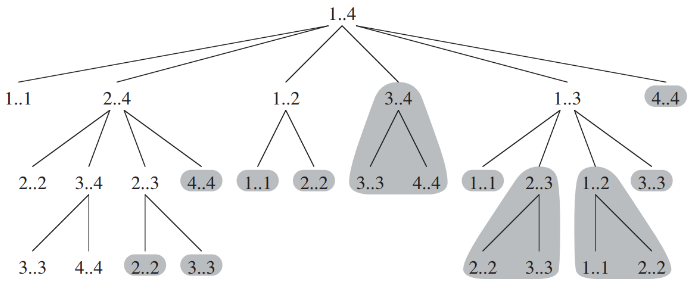


# of subproblems: one problem for each choice of i and j satisfying $1 \le i \le j \le n$? （所有子问题总数为 ?）

Exercise-2

$$\Big(\ \big(A_i\,(A_{i+1}\ldots)\,(\ldots)\ldots A_k\big)\ \big(A_{k+1}\ldots\ A_{j-1}\,A_j\big)\ \Big)$$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min\limits_{i \le k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\}, & \text{if } i < j. \end{cases} \quad (15.12)$$



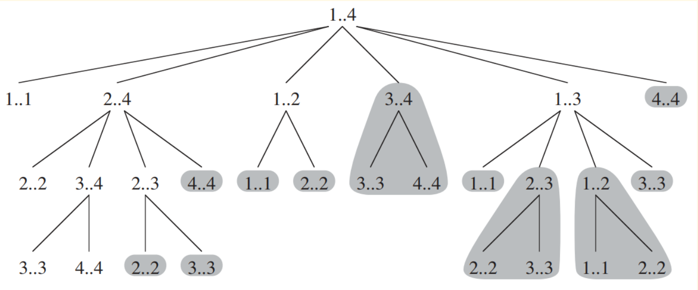

```
RE-MCM(p, i, j)
if i == j
      return 0
m[i, j] ← ∞
for k ← i to j-1
      q ← RE-MCM(p, i, k) + RE-MCM(p, k+1, j) + p_{i-1} p_k p_j
      if q < m[i, j]
            m[i, j] ← q
return m[i, j]
```

Running time ?

Exercise-3

$$(\ (A_i\,(A_{i+1}\ldots)\,(\ldots)\ldots A_k\,)\,(\,A_{k+1}\ldots\ A_{j\text{-}1}\,A_j)\ )$$

$$m[i,j]=\begin{cases}0 & ,\quad \text{if } i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & \text{if } i<j.\end{cases} \qquad (15.12)$$



Dynamic Programming:

top-down with memoization?

采用自顶向下递归的方式进行计算（带备忘录），算法如何写？

Exercise-4

$$e[i,j]=\begin{cases} q_{i-1} & \text{if } j=i-1,\\ \min\limits_{i\le r\le j}\{e[i,r-1]+e[r+1,j]+w[i,j]\} & \text{if } i\le j.\end{cases} \qquad (15.19)$$

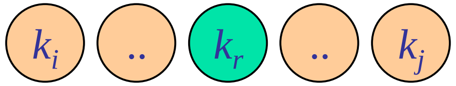
$$w[i,j]=\sum_{l=i}^{j} p_l+\sum_{l=i-1}^{j} q_l = w[i,j-1]+p_j+q_j \qquad (15.20)$$

对OBST问题，基于式(15.19)和(15.20)，

直接的递归算法如何写？

带备忘录(Memoization)的递归算法如何写？

各自的计算时间(Running time)是什么？



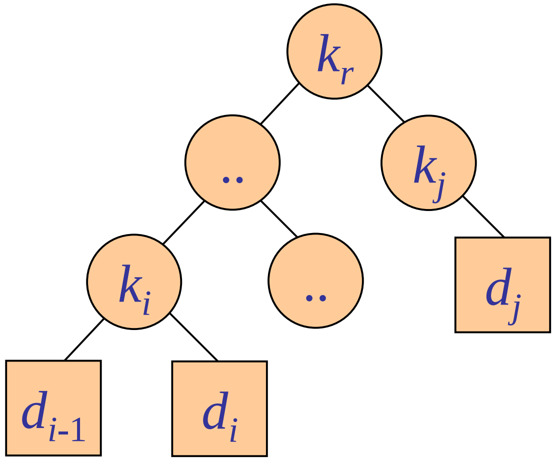
n	0	1	2	3	..
p_n		..	..	..	..
q_n	..	..	..	..	..

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

E[search cost in T]

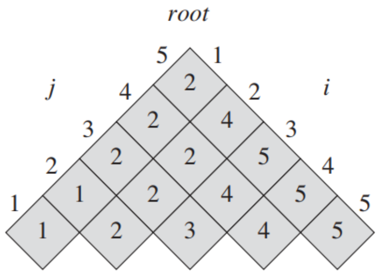
$$=\sum_{i=1}^{n}(\text{depth}_T(k_i)+1)\cdot p_i+\sum_{i=0}^{n}(\text{depth}_T(d_i)+1)\cdot q_i$$

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

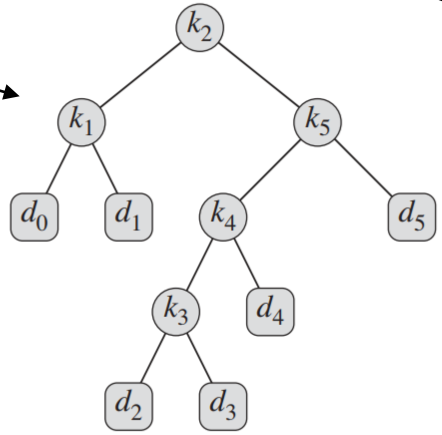


Exercise-5

15.5-1 根据所求出的root，写成一个构造OBST结构的伪代码CONSTRUCT-OBST(root)，以文本形式输出这种结构。



对应的OBST为




```
OBST(p, q, n)
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
```

$k2$ is the root
$k1$ is the left child of $k2$
$d0$ is the left child of $k1$
$d1$ is the right child of $k1$
$k5$ is the right child of $k2$
$k4$ is the left child of $k5$
$k3$ is the left child of $k4$
$d2$ is the left child of $k3$
$d3$ is the right child of $k3$
$d4$ is the right child of $k4$
$d5$ is the right child of $k5$

Exercise-6

$A_1 A_2 A_3 A_4 A_5$: $(A_1 A_2 A_3)(A_4 A_5)$? $(A_1 A_2)(A_3 A_4 A_5)$? ……

$A_1 (A_2 A_3)$? $(A_1 A_2) A_3$?

$$\left(\underline{(A_i (A_{i+1} \ldots)(\ldots)\ldots A_k)} \; \underline{(A_{k+1} \ldots A_{j-1} A_j)} \right)$$

- Brute force: exhaustively checking all possible parenthesizations.
  $P(n)$: the # of alternative parenthesizations of n matrices.
  矩阵连乘时，暴力穷举所有可能的加括号方式
  令$P(n)$表示 n 个矩阵连乘时所有可能的全括号方式的个数
- What is the solution of $P(n)$?

Project

根据一本专业书籍（如《算法导论》），建设一个翻译软件中计算机类词库（字典）的OBST。说明：只考虑英语单词作为关键字。

求解思路：

1. 统计书籍里有多少个单词 M
2. 按字母序，第 i $(1 \le i \le n)$ 个单词在书中出现了 k_i 次 $(k_1 + k_2 + \ldots + k_n = M)$，其词频 $p_i = k_i / M$
3. 根据词频表，构建OBST

思考：对比一般的平衡二叉搜索树 BBST（用中间点作为树根），比较 BBST 与 OBST 的搜索代价。