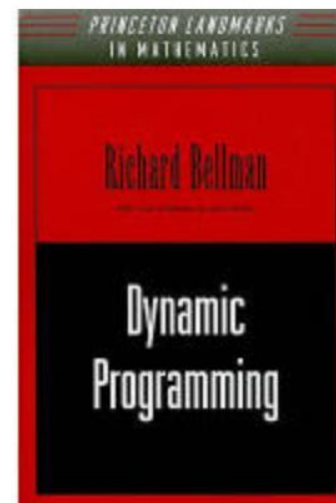


# 15 Dynamic Programming

# 15 Dynamic Programming

**Richard Bellman.  
Dynamic Programming.  
Princeton University  
Press, 1957.**



¥411.00

预订 [Dynamic Programming](#) 原版新书

Google 学术搜索

dynamic programming

文章

找到约 3,570,000 条结果 (用时0.04秒)

时间不限

2021以来

2020以来

2017以来

自定义范围...

[Dynamic programming](#)

[R Bellman - Science, 1966 - science.sciencemag.org](#)

Little has been done in the study of these intriguing questions, and I do not wish to give the impression that any extensive set of ideas exists that could be called a "theory." What is quite surprising, as far as the histories of science and philosophy are concerned, is that the major impetus for the fantastic growth of interest in brain processes, both psychological and ...

☆ 被引用次数: [26841](#) 相关文章 所有 50 个版本

# 15 Dynamic Programming

- ✓ Scheduling two automobile assembly lines
- ✓ Steel rod cutting (15.1)
- ✓ **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)**

## 15.2 Matrix-chain multiplication (MCM)

- Given a sequence (chain)  $\langle A_1, A_2, \dots, A_n \rangle$  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’，如何求积？

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

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

- We can evaluate (15.10) using the standard algorithm for multiplying pairs of matrices as a subroutine once we have parenthesized it.
- 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,  $A_1, (A_1((A_2 A_3) A_4)), (A_1((A_2 A_3)(A_4 A_5)))$ .  
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：

$$\left( A_1 \left( \left( A_2 A_3 \right) A_4 \right) \right)$$


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”。

## Example: Multiplication of two matrices (矩阵相乘)

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

Standard method

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

$$\begin{pmatrix} \dots\dots\dots \\ \dots\dots c_{ij} \dots \\ \dots \\ \dots\dots\dots \end{pmatrix} = \begin{pmatrix} \dots\dots\dots \\ \#\#\dots\# \\ \dots \\ \dots\dots\dots \end{pmatrix} * \begin{pmatrix} \dots\dots\#\dots \\ \dots\dots\#\dots \\ \dots \\ \dots\dots\#\dots \end{pmatrix}$$

**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

- Given a sequence (chain)  $\langle A_1, A_2, \dots, A_n \rangle$  of  $n$  matrices to be multiplied, and we wish to compute the product

$$A_1 A_2 A_3 A_4. \quad (15.10)$$

- Matrix multiplication is **associative**, so all parenthesizations yield the **same product**. For example, if the chain of matrices is  $\langle A_1, A_2, A_3, A_4 \rangle$ , the product  $A_1 A_2 A_3 A_4$  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...

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

## 15.2 Matrix-chain multiplication

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

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

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

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

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

$$( ( (A_1 A_2) A_3) A_4) .$$

## 15.2 Matrix-chain multiplication

- 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  **$pqr$** .

$$A * B \Rightarrow C$$

$$\begin{pmatrix} \dots\dots\dots \\ \#\#\dots\# \\ \dots\dots\dots \\ \dots\dots\dots \end{pmatrix}_{p \times q} * \begin{pmatrix} \dots\dots\#\dots \\ \dots\dots\#\dots \\ \dots \\ \dots\dots\#\dots \end{pmatrix}_{q \times r} \Rightarrow \begin{pmatrix} \dots\dots\dots \\ \dots\dots c_{ij} \dots \\ \dots \\ \dots\dots\dots \end{pmatrix}_{p \times r}$$

**MATRIX-MULTIPLY( $A, B$ )**

```
1 if  $columns[A] \neq rows[B]$ 
2   then return “error: incompatible dimensions”
3 else for  $i \leftarrow 1$  to  $rows[A]$  //  $p$  is  $rows[A]$ 
4       for  $j \leftarrow 1$  to  $columns[B]$  //  $r$  is  $columns[B]$ 
5            $C[i,j] \leftarrow 0$ 
6           for  $k \leftarrow 1$  to  $columns[A]$  //  $q$  is  $columns[A]$ 
7                $C[i,j] \leftarrow C[i,j] + A[i,k] \cdot B[k,j]$ 
8 return  $C$ 
```

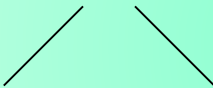


## 15.2 Matrix-chain multiplication

- For  $A_{p \times q}, B_{q \times r}, C=AB$  is  $p \times r$ . The # of **scalar multiplications** is  **$pqr$** .
- Consider the problem of a chain  $\langle A_1, A_2, A_3 \rangle$ ,  
Suppose that  $A_1: 10 \times 100; A_2: 100 \times 5; A_3: 5 \times 50$ 
  - ♦ If  $A = ((A_1 A_2) A_3)$ ,
    - a)  $C = A_1 A_2$ , # of multiplications  $10 \cdot 100 \cdot 5 = \mathbf{5000}$ ,  $C_{10 \times 5}$
    - b)  $A = C A_3$ , # of multiplications  $10 \cdot 5 \cdot 50 = \mathbf{2500}$ ,  $A_{10 \times 50}$   
then, # of scalar multiplications, for a total of **7500**.
  - ♦ If  $A = (A_1 (A_2 A_3))$ ,
    - a)  $C_{100 \times 50} = A_2 A_3$ , # of multiplications  $100 \cdot 5 \cdot 50 = \mathbf{25,000}$ ,
    - b)  $A_{10 \times 50} = A_1 C$ , # of multiplications  $10 \cdot 100 \cdot 50 = \mathbf{50,000}$ ,  
then, # of scalar multiplications, for a total of **75,000**.
  - ♦ The first case is **10 times faster** than the second.

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

## 15.2 Matrix-chain multiplication

$$A_1 A_2 A_3 A_4 A_5 : \quad (A_1 A_2 A_3) (A_4 A_5)? \quad (A_1 A_2) (A_3 A_4 A_5)? \quad \dots$$

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

- **Matrix-chain multiplication problem** : Given a chain  $\langle A_1, A_2, \dots, A_n \rangle$ ,  $i=1, 2, \dots, n$ , matrix  $A_i$  has dimension  $p_{i-1} \times p_i$ , **fully parenthesize** the product  $A_1 A_2 \dots A_n$  in a way that **minimizes** the number of scalar multiplications.
- In the problem, we are not actually multiplying matrices. Our goal is only **to determine an order** for multiplying matrices that has the lowest cost.

## Counting the number of parenthesizations

$$A_1 A_2 A_3 A_4 A_5$$

$$\begin{array}{c} ((\underline{A_1 A_2 A_3}) (A_4 A_5)) ? \quad (A_1 A_2) (A_3 A_4 A_5) ? \quad \dots\dots \\ \swarrow \quad \searrow \\ (A_1 (A_2 A_3)) ? \quad ((A_1 A_2) A_3) ? \end{array}$$

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

- Brute force, exhaustively checking all possible parenthesizations.
- $P(n)$ : the # of alternative parenthesizations of  $n$  matrices.  $P(n)$ 种全括号方式
  - ♦  $n=1$ , one matrix, one way to parenthesize the matrix product.

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

- The solution to (15.11) is  $\Omega(2^n)$  (guess, then prove), a poor strategy.

# Counting the number of parenthesizations

$A_1 A_2 A_3 A_4 A_5$

$((A_1 A_2 A_3) (A_4 A_5))?$        $(A_1 A_2) (A_3 A_4 A_5)?$       .....  
 $\swarrow \quad \searrow$   
 $(A_1 (A_2 A_3))?$      $((A_1 A_2) A_3)?$

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

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

## 4 Recurrences

### Algorithms analysis

### E Zexal的二叉树 (签到)

时间限制: 1000ms 内存限制: 65536kb

通过率: 200/209 (95.69%) 正确率: 200/206 (97.08%)

#### 题目

知识点: 树, 数论, dp, 递归 (都可以做)

上学期我们学习了二叉树, 也都知道3个结点的二叉树有5种, 现在你二叉树的结点个数n, 要你输出不同形态二叉树的种数。

#### 输入

第一个数为一个整数n(n <= 30)

#### 输出

对于每组数据, 输出一行, 不同形态二叉树的种数。

#### 输入样例

3

#### 输出样例

5

### recursion

$h(n) = h(0)*h(n-1)$   
 $+ h(1)*h(n-2) + \dots$   
 $+ h(n-1)h(0)$

### 另类递归式:

$h(n) = ((4*n-2)/(n+1))*h(n-1)$

### 该递推关系的解为:

$h(n) = C(2n, n)/(n+1)$

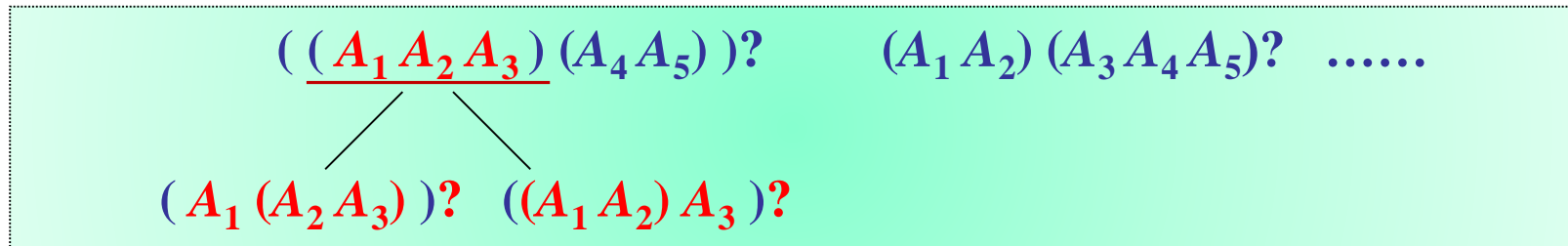
其中, 规定 $h(0) = 1$

catalan数, 卡特兰数, 是一个常出现在各种计数问题中的数列, 以比利时的数学家欧仁-查理-卡特兰命名

# **Dynamic Programming to solve MCM:**

## **Four Steps**

## Step 1: The structure of an optimal parenthesization



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

Find the **optimal substructure**

$A_{i..j}$  (where  $i \leq j$ ) : the product  $A_i A_{i+1} \dots A_k A_{k+1} \dots A_j$

- ♦  $i < j$ , **nontrivial**, any parenthesization of the product  $A_i A_{i+1} \dots A_j$  must **split** the product between  $A_k$  and  $A_{k+1}$  for some integer  $k$  in the range  $i \leq k < j$ .
- ♦ First compute  $A_{i..k}$ , and  $A_{k+1..j}$ , then  $A_{i..k} \cdot A_{k+1..j} = A_{i..j}$

The cost of this parenthesization  $A_i A_{i+1} \dots A_j$

= the cost of computing the matrix  $A_{i..k}$  + the cost of computing  $A_{k+1..j}$   
 + the cost of multiplying  $A_{i..k} \cdot A_{k+1..j}$

# Step 1: The structure of an optimal parenthesization

## The optimal substructure

- Suppose that an optimal parenthesization of  $A_i A_{i+1} \dots A_j$  splits the product between  $A_k$  and  $A_{k+1}$ .

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

设矩阵连乘的最佳全括号将矩阵连乘分成  $A_{i..k}$  和  $A_{k+1..j}$  两部分之积

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

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

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

**Proof**



如果  $X$  最优，则  $M$  最优

## Step 1: The structure of an optimal parenthesization

### The optimal substructure

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

Proof Optimal parenthesization  $A_{i..j} = (A_i \dots A_k) (A_{k+1} \dots A_j) = M \cdot N$ , parenthesization  $M = (A_i A_{i+1} \dots A_k)$  in  $A_{i..j}$  above.

If there were a less costly way to parenthesize  $A_i \dots A_k = P$ , substituting  $M$  with  $P$ , that is  $P \cdot N$  would produce another parenthesization of  $A_{i..j}$  whose cost was lower than  $M \cdot N$ .

A contradiction.

- ◆ A **similar** observation holds for  $A_{k+1} A_{k+2} \dots A_j$



## Step 1: The structure of an optimal parenthesization

$$\frac{((A_i(A_{i+1}\dots)(\dots)\dots A_k)(A_{k+1}\dots A_{j-1}A_j))}{\begin{array}{cc} \boxed{M} & \boxed{X} \end{array}}$$

- Construct an optimal solution to the problem  $X$  from optimal solutions to subproblems  $M$  based on optimal substructure.  
from  $M$  to  $X$
- Any solution to the matrix-chain multiplication requires us to **split** the product, and any optimal solution contains within it optimal solutions to subproblem instances.
  - ◆ **Split** the problem into two subproblems (optimally parenthesizing  $A_i A_{i+1} \dots A_k$  and  $A_{k+1} A_{k+2} \dots A_j$ );
  - ◆ **find** optimal solutions to subproblem  $M$ ;
  - ◆ **combine** these optimal subproblem solutions ( from  $M$  to  $X$  )

## Step 1: The structure of an optimal parenthesization

$$A_1 A_2 A_3 A_4 A_5$$

$$\begin{array}{c} ((\underline{A_1 A_2 A_3}) (A_4 A_5)) ? \quad (A_1 A_2) (A_3 A_4 A_5) ? \quad \dots\dots \\ \swarrow \quad \searrow \\ (A_1 (A_2 A_3)) ? \quad ((A_1 A_2) A_3) ? \end{array}$$

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

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以确保最优解是其中之一

## Step 2: A recursive solution

---

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

- 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  $A_iA_{i+1}\dots A_j$  for  $1 \leq i \leq j \leq n$ .

Not  $A_1A_2\dots A_j$ , Why?

## Step 2: A recursive solution

$$\underbrace{\left( \left( A_i (A_{i+1} \dots) (\dots) \dots A_k \right) \right)}_M \left( A_{k+1} \dots A_{j-1} A_j \right)_X$$

$m[i, j] = |X| :$

the minimum # of scalar multiplications to compute  $A_{i..j}$  ;

the cost of a cheapest way to compute  $A_{1..n}$  is  $m[1, n]$ .

- ◆ If  $i = j$ , one matrix  $A_{i..i} = A_i$ , no scalar multiplications.  
Thus,  $m[i, i] = 0$  for  $i = 1, 2, \dots, n$ .
- ◆ When  $i < j$  ?

## Step 2: A recursive solution

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

$m[i, j]$  :

When  $i < j$ , assuming that the optimal parenthesization **splits**  $A_i A_{i+1} \dots A_j$  **between**  $A_k$  **and**  $A_{k+1}$ , where  $i \leq k < j$ .

Then,

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



$m[i, j]$  = the minimum cost for computing  $A_{i..k}$   
+ the minimum cost for computing  $A_{k+1..j}$   
+ the cost of multiplying  $A_{i..k}$  and  $A_{k+1..j}$

$A_i$  has dimensions  $p_{i-1} \times p_i$ , then  $A_{i..k}$  has  $p_{i-1} \times p_k$ ,  $A_{k+1..j}$  has  $p_k \times p_j$ .

## Step 2: A recursive solution

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

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

- This recursive equation assumes that we know the value of  $k$ , which we actually do not know.
- Only  $j-i$  possible values for  $k$ , namely  $k = i, i+1, \dots, j-1$ .
- Checking them all to find the best  $k$ , we have

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

- The  $m[i, j]$  give the costs of optimal solutions to subproblems.

## Step 2: A recursive solution

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

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

### Construct an optimal solution :

Define  $s[i, j]$  to be a value of  $k$  at which we can split the product  $A_i A_{i+1} \dots A_j$  to obtain an optimal parenthesization. That is,  $s[i, j]$  equals a value  $k$  such that  $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$ .

### Step 3: Computing the optimal costs

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

**recursive** algorithm based on recurrence (15.12)?

$m[1, n]$  for multiplying  $A_1A_2\ldots A_n$  .

```
RE-MCM(p, i, j)
1  if i = j
2    return 0
3  m[i, j] ← ∞
4  for k ← i to j-1
5    q ← RE-MCM(p, i, k) + RE-MCM(p, k+1, j) + pi-1pkpj
6    if q < m[i, j]
7      m[i, j] ← q
8  return m[i, j]
```

**Running time?**



## Step 3: Computing the optimal costs

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

RE-MCM( $p, i, j$ )	$T(n)$
1 <b>if</b> $i = j$	1
2 <b>return</b> 0	1
3 $m[i, j] \leftarrow \infty$	1
4 <b>for</b> $k \leftarrow i$ <b>to</b> $j-1$	1
5 $q \leftarrow$ RE-MCM( $p, i, k$ )	$T(k)$
$+ \text{RE-MCM}(p, k+1, j)$	$T(n-k)$
$+ p_{i-1}p_kp_j$	1
6 <b>if</b> $q < m[i, j]$	1
7 $m[i, j] \leftarrow q$	1
8 <b>return</b> $m[i, j]$	1

$$\begin{aligned} T(n) &\geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \\ &= 2 \sum_{i=1}^{n-1} T(i) + n \geq 3^n \end{aligned}$$

**Proof**     **let**  $T(i) \geq 3^i$  ,

**then**  $T(n) \geq 2 \sum_{i=1}^{n-1} T(i) + n$

$$\begin{aligned} &\geq 2 \times (3^1 + 3^2 + \dots + 3^{n-1}) + n \\ &\geq 2 \times 3 \frac{3^{n-1} - 1}{3 - 1} + n \\ &= 3^n - 3 + n \geq 3^n \end{aligned}$$

This algorithm takes **exponential** time, which is **no better than the brute-force method** of checking each way of parenthesizing the product. **Why?**

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

$\Omega(2^n)$

## Step 3: Computing the optimal costs

$$\begin{aligned}
 & \underline{\underline{((A_i(A_{i+1}\dots)(\dots)\dots A_k)(A_{k+1}\dots A_{j-1}A_j))}} \\
 m[i, j] = & \begin{cases} 0, & \text{if } i = j, \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\}, & \text{if } i < j. \end{cases}
 \end{aligned} \tag{15.12}$$

**Recursion, Extremely slow!**

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

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

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

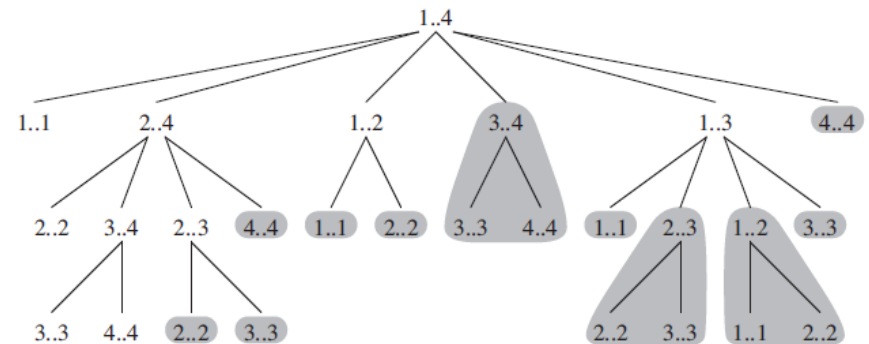
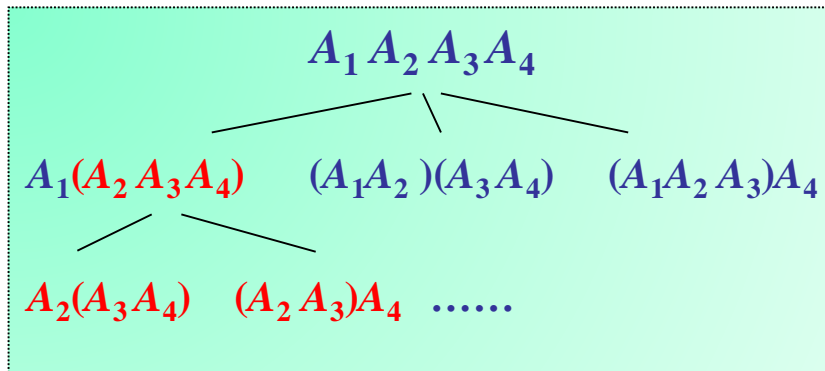
$$1 \leq i < j \leq n, C_n^2 = n(n-1)/2$$

## Step 3: Computing the optimal costs

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

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

- A recursive algorithm may encounter each subproblem **many times** in different branches of its recursion tree.
- **Overlapping subproblems:** the second hallmark of the applicability of dynamic programming.



## Step 3: Computing the optimal costs

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

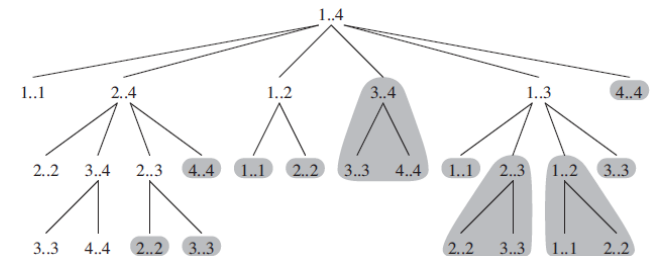
$$m[i, j] = \begin{cases} 0 & , \text{ if } i = j, \\ \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}, & \text{ if } i < j. \end{cases} \quad (15.12)$$

- # of subproblems: one problem for each choice of  $i$  and  $j$

satisfying  $1 \leq i \leq j \leq n$ , or  $\binom{n}{2} + n = \Theta(n^2)$  in all. (子问题个数为 $\Theta(n^2)$ )

- Instead of recursive method, computing the optimal cost by using a tabular, bottom-up approach.

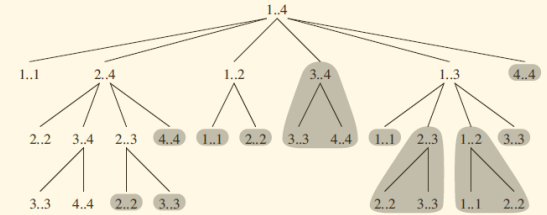
不用递归方法，而采用列表方式、  
自底向上的方法计算最优解



## Step 3: Computing the optimal costs

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

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



**$A_i$  : dimensions  $p_{i-1} \times p_i$**

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

**Procedure:** Table  $m[1..n, 1..n]$   
storing the  $m[i, j]$  costs;

**Auxiliary table  $s[1..n, 1..n]$   
recording which index of  $k$   
achieved the optimal cost in  
computing  $m[i, j]$ .**

计算  $m[i,j]$  时, 用辅助表项(table entry)  $s[i,j]$  来记录最佳位置  $k$  的值

MCM-DP( $p$ )
$$1 \quad n \leftarrow \text{length}[p] - 1$$

```

2 for  $i \leftarrow 1$  to  $n$ 

```

```

3       $m[i, i] \leftarrow 0$ 

```

```
4 for  $l \leftarrow 2$  to  $n$            //  $l$  is the chain length.
```

```

5   for  $i \leftarrow 1$  to  $n - l + 1$ 

```

6  $j \leftarrow i + l - 1$ 

```

7       $m[i,j] \leftarrow \infty$ 

```

8       **for**  $k \leftarrow i$  **to**  $j - 1$

$$9 \quad q \leftarrow m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$$
10                   **if**  $q < m[i, j]$ 11  $m[i, j] \leftarrow q$ 

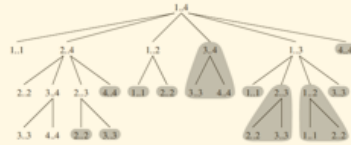
12                     $s[i,j] \leftarrow k$

```
13 return  $m$  and  $s$ 
```

# Step 3: Computing the optimal costs

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

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



	<i>j</i>						
	1	2	3	4	5	6	<i>m</i>
<i>i</i>	1	0	15,750	7,875	9,375	11,875	15,125
	2		0	2,625	4,375	7,125	10,500
	3	<i>A</i> <sub>2</sub>		0	750	2,500	5,375
	4		<i>A</i> <sub>3</sub>		0	1,000	3,500
	5			<i>A</i> <sub>4</sub>		0	5,000
	6				<i>A</i> <sub>5</sub>		0
						<i>A</i> <sub>6</sub>	

MCM-DP(*p*)

```

1  n ← length[p] − 1
2  for i ← 1 to n
3      m[i, i] ← 0
4  for l ← 2 to n           // l is the chain length.
5      for i ← 1 to n − l + 1
6          j ← i + l − 1
7          m[i, j] ← ∞
8          for k ← i to j − 1
9              q ← m[i, k] + m[k + 1, j] + pi−1 pk pj
10             if q < m[i, j]
11                 m[i, j] ← q
12                 s[i, j] ← k
13 return m and s
```

*A*<sub>1</sub> 30 × 35  
*A*<sub>2</sub> 35 × 15  
*A*<sub>3</sub> 15 × 5  
*A*<sub>4</sub> 5 × 10  
*A*<sub>5</sub> 10 × 20  
*A*<sub>6</sub> 20 × 25

	<i>j</i>					
	2	3	4	5	6	<i>s</i>
<i>i</i>	1	1	3	3	3	1
			2	3	3	2
				3	3	3
					4	4
						5

## Step 3: Computing the optimal costs

	<i>j</i>						
	1	2	3	4	5	6	<i>m</i>
<i>i</i>	0	15,750	7,875	9,375	11,875	15,125	1
		0	2,625	4,375	7,125	10,500	2
			0	750	2,500	5,375	3
				0	1,000	3,500	4
					0	5,000	5
						0	6

rotate 45°

						<i>m</i>	
						6	1
						5	2
						4	3
						3	4
						2	5
						1	6
<i>j</i>	15,750	7,875	9,375	11,875	15,125		
	0	2,625	4,375	7,125	10,500		
		0	750	2,500	5,375		
			0	1,000	3,500		
				0	5,000		
					0		

	<i>j</i>					
	2	3	4	5	6	<i>s</i>
<i>i</i>	1	1	3	3	3	1
		2	3	3	3	2
			3	3	3	3
				4	5	4
					5	5

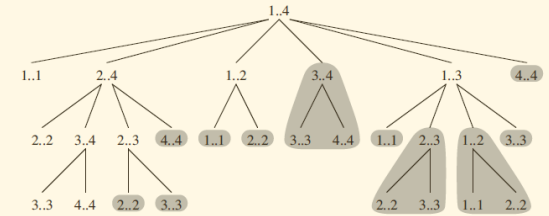
rotate 45°

						<i>s</i>	
						6	1
						5	2
						4	3
						3	4
						2	5
						1	
<i>j</i>	1	2	3	3	3		
		1	3	3	3		
			2	3	3		
				4	5		
					5		

## Step 3: Computing the optimal costs

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

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



MCM-DP( $p$ )

1  $n \leftarrow \text{length}[p] - 1$

2 **for**  $i \leftarrow 1$  **to**  $n$

3  $m[i, i] \leftarrow 0$

4 **for**  $l \leftarrow 2$  **to**  $n$  //  $l$  is the chain length.

5 **for**  $i \leftarrow 1$  **to**  $n - l + 1$

6  $j \leftarrow i + l - 1$

7  $m[i, j] \leftarrow \infty$

8 **for**  $k \leftarrow i$  **to**  $j - 1$

9  $q \leftarrow m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$

10 **if**  $q < m[i, j]$

11  $m[i, j] \leftarrow q$

12  $s[i, j] \leftarrow k$

13 **return**  $m$  and  $s$

$A_1$   $30 \times 35$

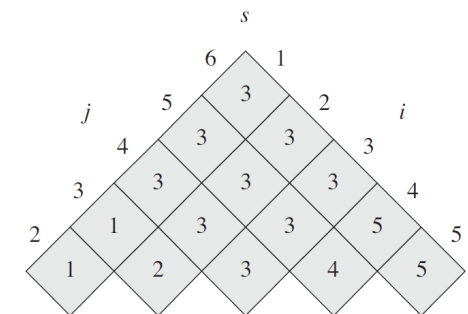
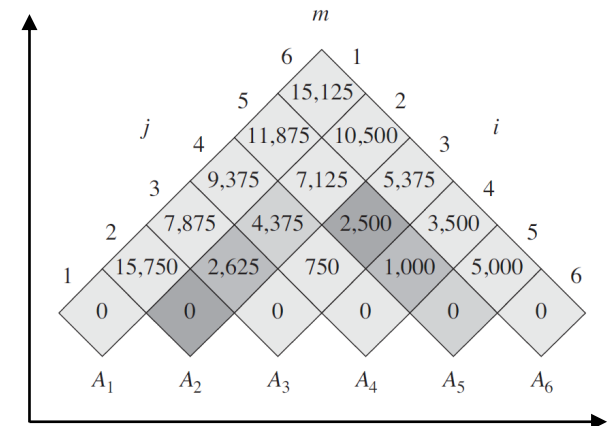
$A_2$   $35 \times 15$

$A_3$   $15 \times 5$

$A_4$   $5 \times 10$

$A_5$   $10 \times 20$

$A_6$   $20 \times 25$





## Step 3: Computing the optimal costs

```
MCM-DP(p)
1 n ← length[p] − 1
2 for i ← 1 to n
3   m[i, i] ← 0
4 for l ← 2 to n    // l is the chain length.
5   for i ← 1 to n − l + 1
6     j ← i + l − 1
7     m[i, j] ← ∞
8     for k ← i to j − 1
9       q ← m[i, k] + m[k + 1, j] + pi−1pkpj
10      if q < m[i, j]
11        m[i, j] ← q
12        s[i, j] ← k
13 return m and s
```

The running time?

Space requirement?

Time :  $T(n) = O(n^3)$

Space:  $S(n) = \Theta(n^2)$

## Step 3: Computing the optimal costs

MCM-DP( $p$ )

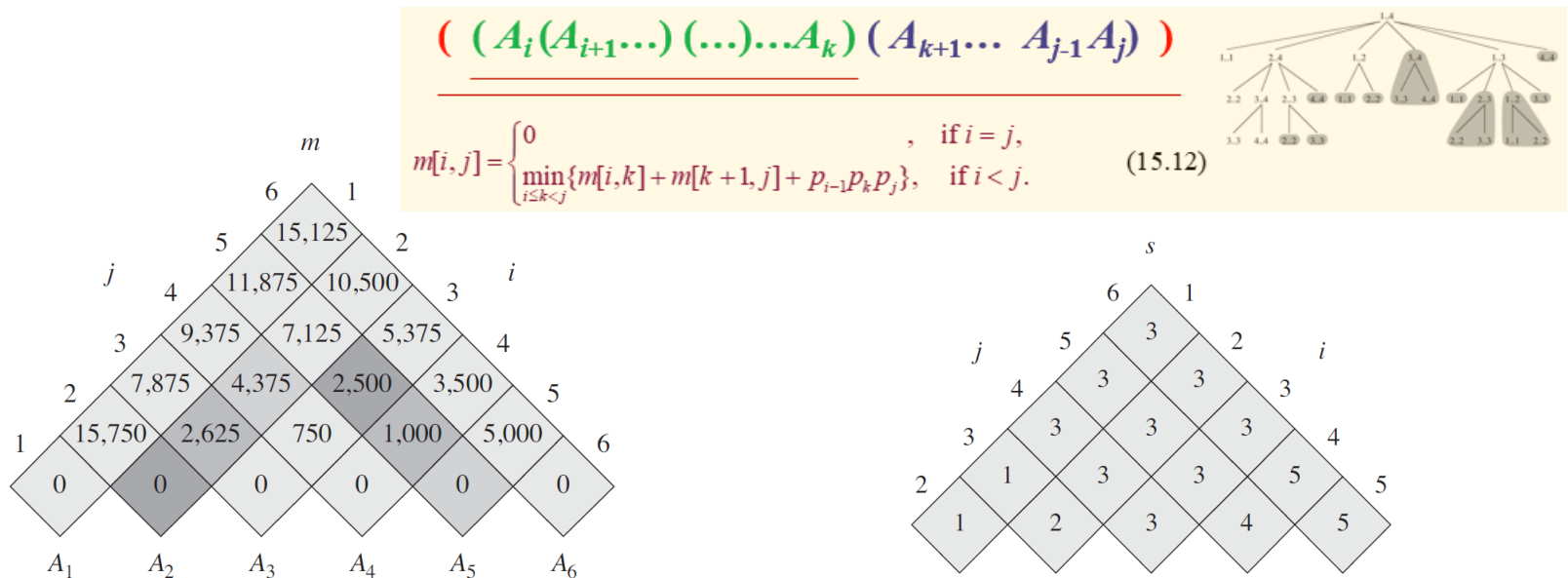
```
1  $n \leftarrow \text{length}[p] - 1$ 
2 for  $i \leftarrow 1$  to  $n$ 
3    $m[i, i] \leftarrow 0$ 
4 for  $l \leftarrow 2$  to  $n$            //  $l : n-1$  times
5   for  $i \leftarrow 1$  to  $n - l + 1$  //  $i : n-l+1$  times
6      $j \leftarrow i + l - 1$ 
7      $m[i, j] \leftarrow \infty$ 
8     for  $k \leftarrow i$  to  $j - 1$  //  $k : j-i=l-1$  times
9        $q \leftarrow m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$ 
10      if  $q < m[i, j]$ 
11         $m[i, j] \leftarrow q$ 
12         $s[i, j] \leftarrow k$ 
13 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

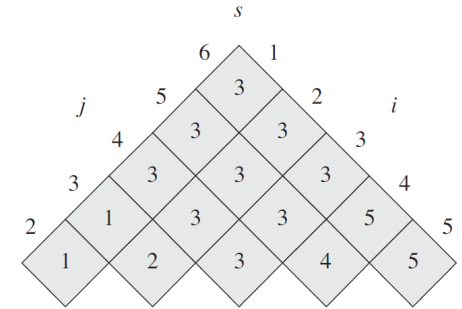
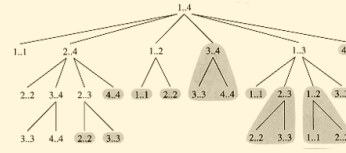


- MCM-DP determines the optimal number  $m[i, j]$ , but **does not directly** show how to multiply the matrices.**  
 算法MCM-DP 给出了如何求解最佳乘法次数  $m[i, j]$ , 但对于按什么顺序来相乘各矩阵, 没有给出具体方法
- Constructing an optimal solution from table  $s[1.. n-1, 2.. n]$ .**

## Step 4: Constructing an optimal solution

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

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



- Each entry  $s[i, j]$  records the value of  $k$  such that the optimal parenthesization of  $A_i A_{i+1} \cdots A_j$  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  $A_i A_{i+1} \cdots A_j$  的最佳全括号中，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  $(A_1 A_2 \cdots A_{s[1,n]})(A_{s[1,n]+1} \cdots A_n)$ 。

- The matrix multiplications can be computed recursively,

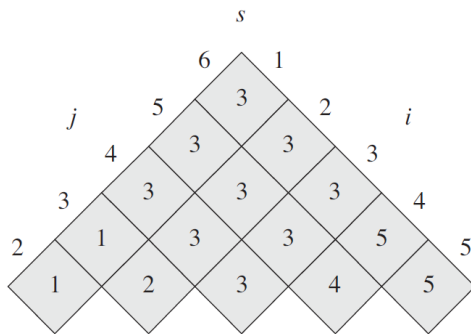
$$s[1, s[1, n]] \rightarrow \text{splits } A_{1..s[1, n]}$$

$$s[s[1, n] + 1, n] \rightarrow \text{splits } A_{s[1, n]+1..n}$$

- PRINT-OPTIMAL-PARENS( $s, i, j$ )

## Step 4: Constructing an optimal solution

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



$A_1$   $30 \times 35$   
 $A_2$   $35 \times 15$   
 $A_3$   $15 \times 5$   
 $A_4$   $5 \times 10$   
 $A_5$   $10 \times 20$   
 $A_6$   $20 \times 25$

```
PRINT-OPTIMAL-PARENS( $s, i, j$ )
1  if  $i == j$ 
2    print " $A$ ";
3  else
4    print "("
5    PRINT-OPTIMAL-PARENS( $s, i, s[i, j]$ )
6    PRINT-OPTIMAL-PARENS( $s, s[i, j]+1, j$ )
7    print ")"
```



**How to work?**

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

## Step 4: Constructing an optimal solution

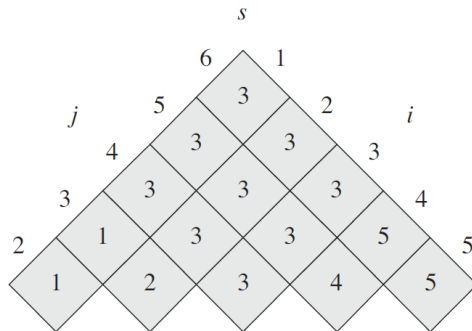
```

PRINT-OPTIMAL-PARENS( $s, i, j$ )
1  if  $i == j$ 
2    print " $A$ ";
3  else
4    print "("
5    PRINT-OPTIMAL-PARENS( $s, i, s[i, j]$ )
6    PRINT-OPTIMAL-PARENS( $s, s[i, j]+1, j$ )
7    print ")"
    
```

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

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

$A_1$   $30 \times 35$   
 $A_2$   $35 \times 15$   
 $A_3$   $15 \times 5$   
 $A_4$   $5 \times 10$   
 $A_5$   $10 \times 20$   
 $A_6$   $20 \times 25$



$s(1,6)$

$(s(1,3) s(4,6))$

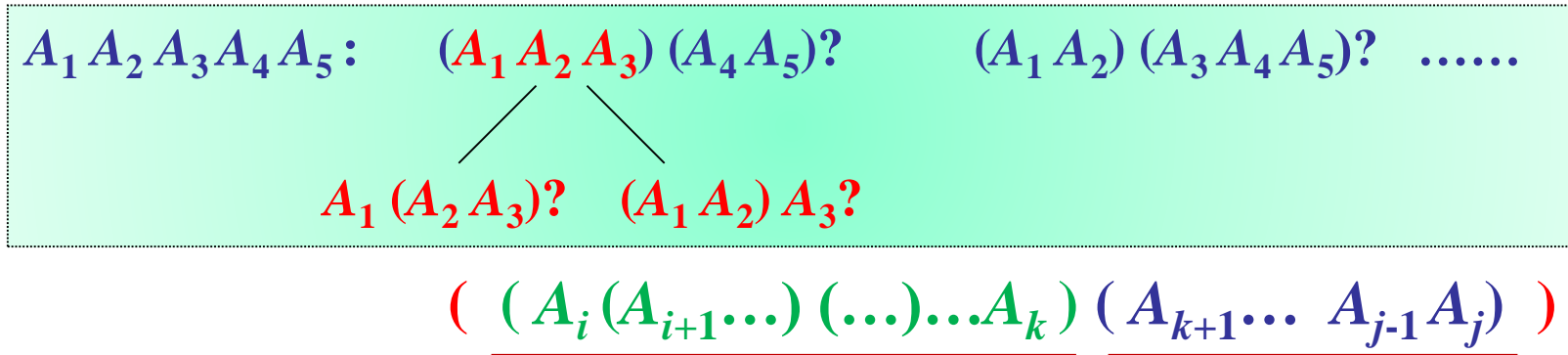
$(s(1,1) s(2,3))$

$(s(2,2) s(3,3))$

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

?

# Exercise-1 (in class)

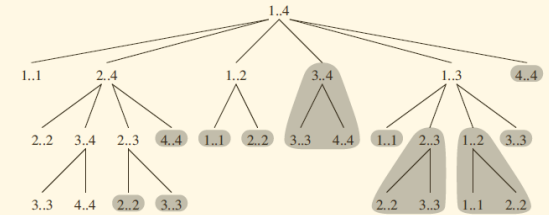


- Brute force: exhaustively checking all possible parenthesizations.
- $P(n)$ : the # of alternative parenthesizations of  $n$  matrices.  
 令  $P(n)$  表示  $n$  个矩阵连乘时所有可能的全括号方式的个数
- What is the solution of  $P(n)$ ?

## Exercise-2 (in class)

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

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



RE-MCM( $p, i, j$ )

```

1  if  $i = j$ 
2    return 0
3   $m[i, j] \leftarrow \infty$ 
4  for  $k \leftarrow i$  to  $j-1$ 
5     $q \leftarrow \text{RE-MCM}(p, i, k) + \text{RE-MCM}(p, k+1, j) + p_{i-1} p_k p_j$ 
6    if  $q < m[i, j]$ 
7       $m[i, j] \leftarrow q$ 
8  return  $m[i, j]$ 
    
```

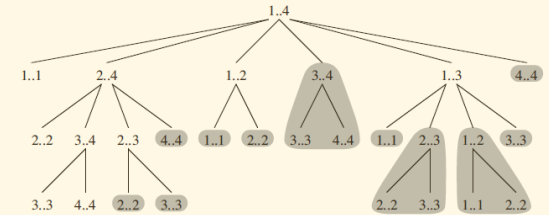
**Running time ?**



# Exercise-3

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

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

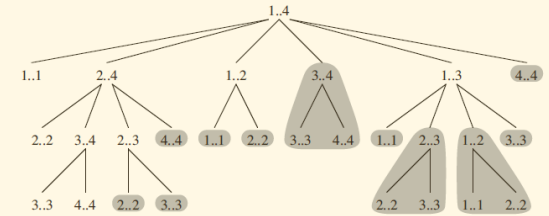


# of subproblems: one problem for each choice of  $i$  and  $j$  satisfying  $1 \leq i \leq j \leq n$  ? (所有子问题个数为 ?)

## Exercise-4

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

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



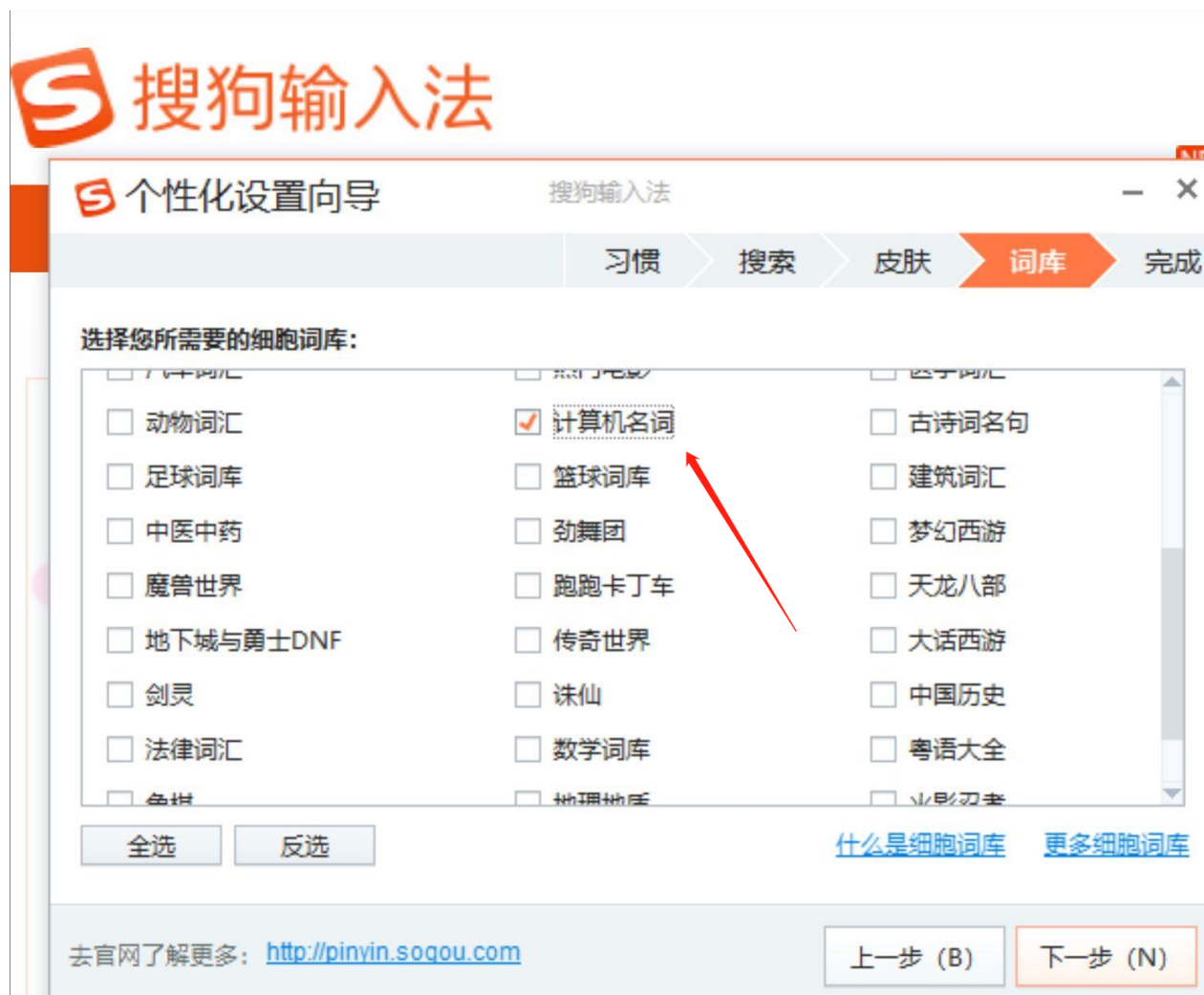
# Dynamic Programming: top-down with memoization?

# 15 Dynamic Programming

- ✓ Scheduling two automobile assembly lines
- ✓ Steel rod cutting (15.1)
- ✓ **Matrix-chain multiplication (15.2)**
- ✓ 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)

译 百度翻译

https://fanyi.baidu.com/

Baidu 翻译

百度wifi翻译机 热 人工翻译 下载翻译插件 下载翻译app 136\*\*\*\*\*73

检测到英语

⇒

中文

翻译

人工翻译

🔍

⚙️

×

Suppose that we are designing a program to translate text from English to French. For each occurrence of each English word in the text, we need to look up its French equivalent. We could perform these lookup operations by building a

🔊 ☆

假设我们正在设计一个程序来把文本从英语翻译成法语。对于每一个出现在文本中的每个英语单词，我们需要查找它的法语等价物。我们可以通过构建一个二进制搜索树来执行这些查找操作，其中n个英语单词作为关键字，它们的法语等价物作为卫星数据。

🔊 📄 报错 拼音 双语对照

1↓ 排序

重点词汇

重点词汇

[Suppose that](#) 假如

[designing](#) 狡诈的, 狡猾的, 诡计多端的; 设计; 设计, ...

[translate](#) 翻译; 转化; 解释; 被翻译

➤

百度翻译APP

各大应用商店  
精品推荐

# 15.5 Optimal binary search trees

## Design a program to translate text from English to Chinese

检测到英语

⇌

中文

翻译

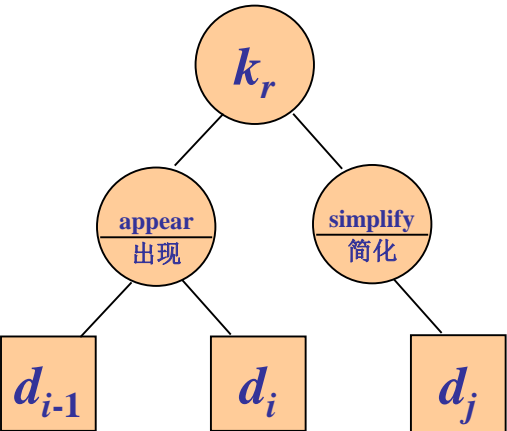
人工翻译

×

Suppose that we are designing a program to translate text from English to French. For each occurrence of each English word in the text, we need to look up its French equivalent. We could perform these lookup operations by building a

假设我们正在设计一个程序来把文本从英语翻译成法语。对于每一个出现在文本中的每个英语单词，我们需要查找它的法语等价物。我们可以通过构建一个二进制搜索树来执行这些查找操作，其中n个英语单词作为关键字，它们的法语等价物作为卫星数据。

报错 拼音 ☐ 双语对照 ☐



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
...	...
suppose	假设
...	...

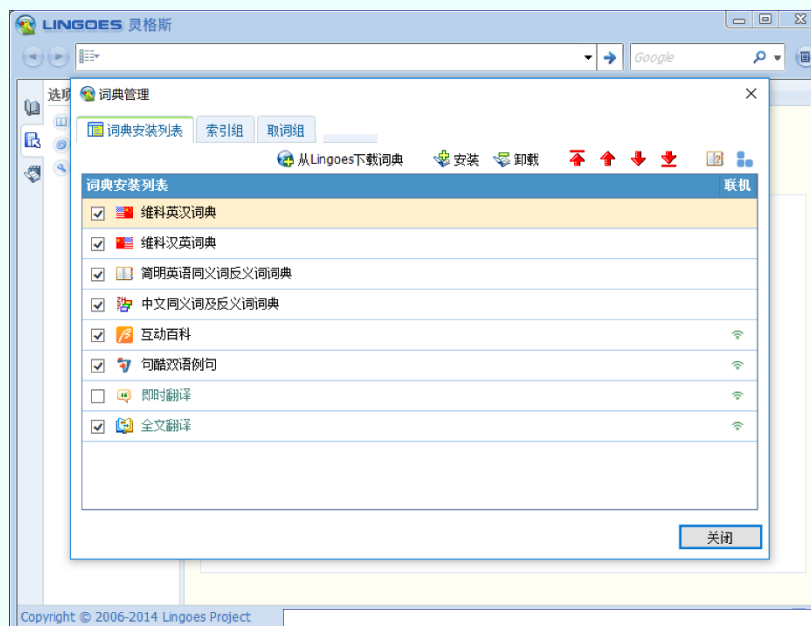
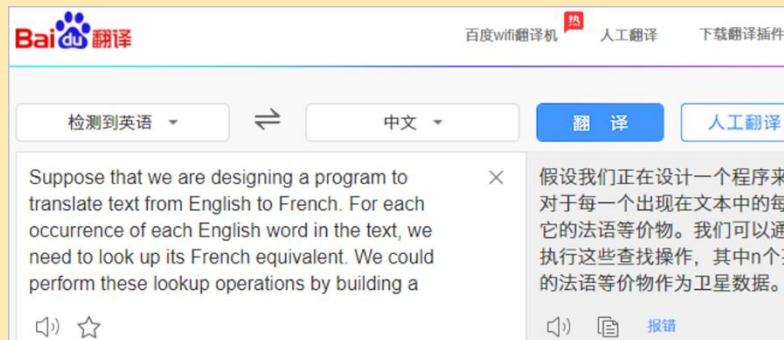
# 15.5 Optimal binary search trees

- Design a program to translate text from English to Chinese
- an  $O(n)$  search time per occurrence by using any linear table operation



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
...	...
suppose	假设
...	...

# 15.5 Optimal binary search trees



算法是软件的灵魂

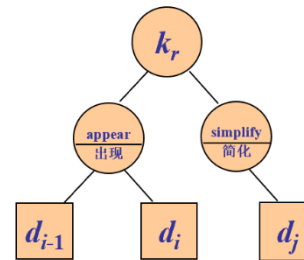
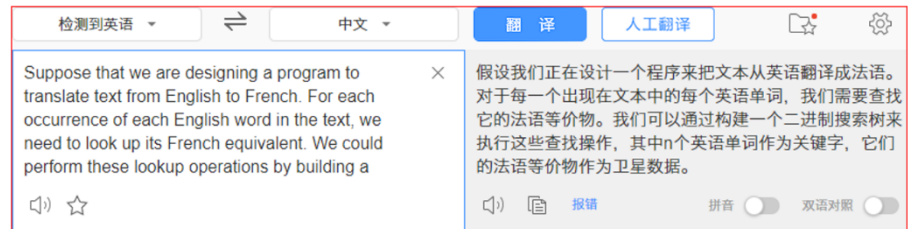
算法是企业生产力





# 15.5 Optimal binary search trees

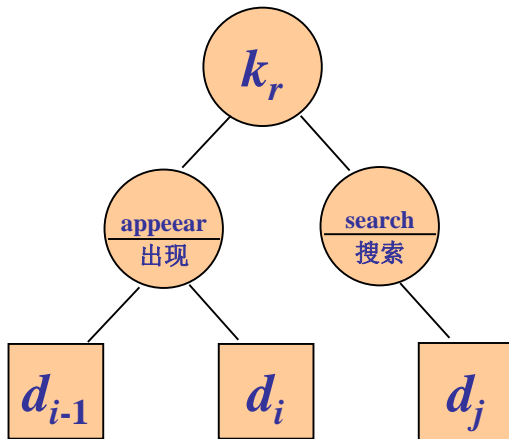
- lookup operations:  
build a **binary search tree** (BST) with
  - ◆  $n$  English words as keys
  - ◆ Chinese equivalents as satellite data 从属数据



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
...	...
suppose	假设
...	...

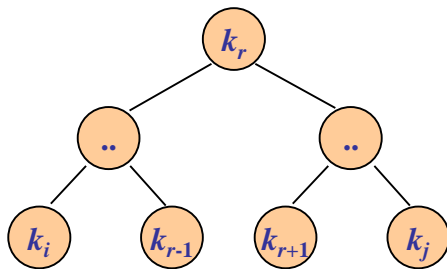
- Because we **will search the tree for each individual word in the text**, we want the total time spent searching to be as low as possible. 对于课文中出现的每个单词，都需要搜索该二叉树，如何使得总的搜索次数最少？
- an  $O(\lg n)$  search time per occurrence by using **any balanced BST**. 对于任何一个单词的搜索，使用二分搜索法的时间为  $O(\lg n)$  .

## 15.5 Optimal binary search trees



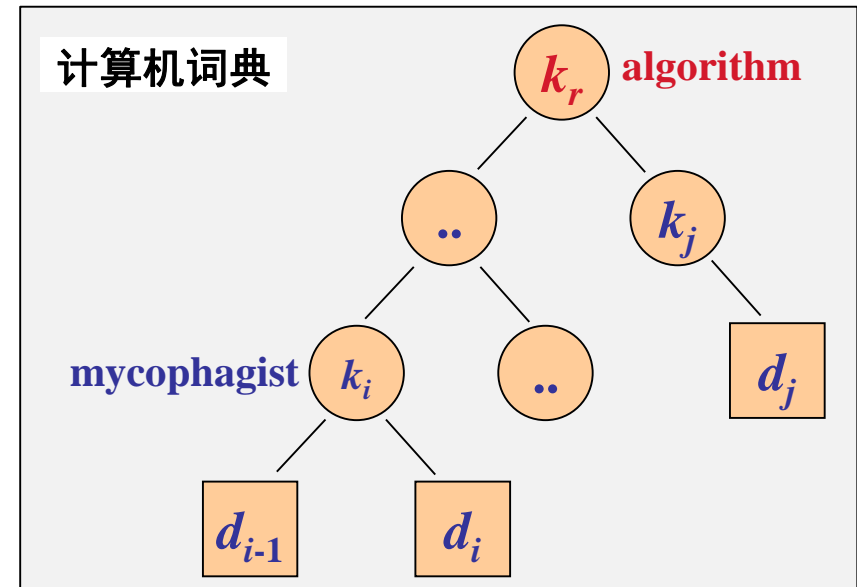
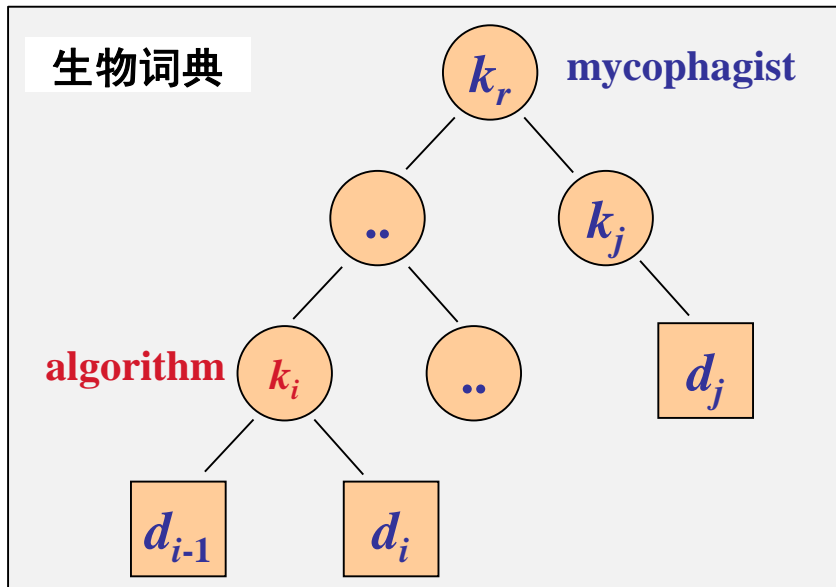
生词表	
字段1	字段2
aggregate	综合，总体
amortized	分摊，平摊
arbitrary	任意的，武断的
auxiliary	辅助的
binomial	二项的，二项式的
bog	沼泽，陷于泥沼
...	...

**A balanced BST...**



**However, Words appear with different frequencies...?**

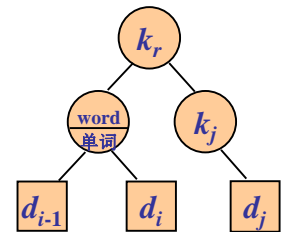
## 15.5 Optimal binary search trees



- 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  $1 + \text{depth}$
- We want words that occur frequently in the text to be placed nearer the root

## 15.5 Optimal binary search trees

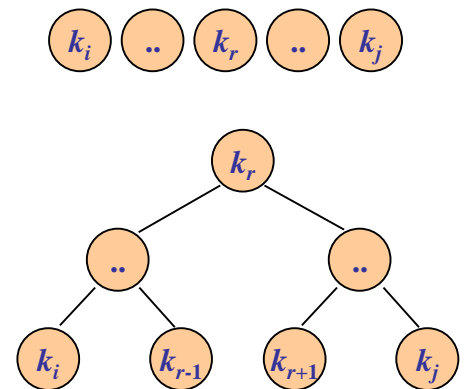
- Need the total time spent searching to be as low as possible.
- We want words that occur frequently in the text to be placed nearer the root.
- Moreover, there may be words in the text for which there is **no Chinese translation**, and such words might not appear in the BST at all.



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“词典”中

- 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?

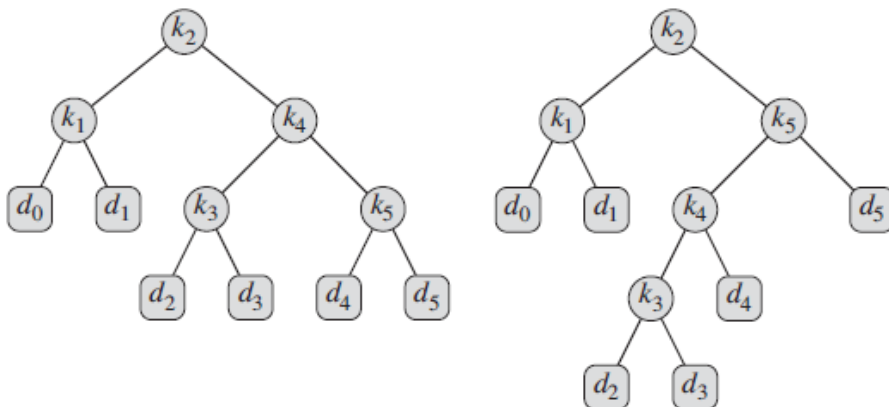
设已知每个单词出现的概率，如何组织一颗二叉搜索树，使得在所有搜索中，被访问的节点的总数最少？



## 15.5 Optimal binary search trees

**BST:** Given a sequence  $K = \langle k_1, k_2, \dots, k_n \rangle$  of  $n$  distinct keys in sorted order ( $k_1 < k_2 < \dots < k_n$ ), how to build a BST?

- For key  $k_i$ , search probability  $p_i$  (it can also be the number  $I$  of occurrence of  $k_i$  in the text, whose number of total words is  $M$ , then  $p_i = I / M$ .)
- Some values not in  $K$ ,  $n+1$  "dummy keys"  $d_0, d_1, \dots, d_n$
- $d_0$  represents all values  $< k_1$ ;  $d_n$  represents all values  $> k_n$
- for  $1 \leq i \leq n-1$ ,  $d_i : k_i < d_i < k_{i+1}$ , search probability  $q_i$
- Fig, each key  $k_i$ , an internal node;  $d_i$ , a leaf



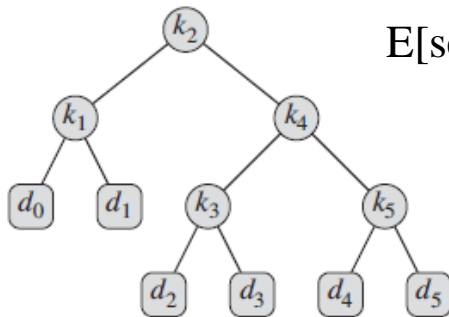
$i$	0	1	2	3	4	5
$p_i$		0.15	0.10	0.05	0.10	0.20
$q_i$	0.05	0.10	0.05	0.05	0.05	0.10

## 15.5 Optimal binary search trees

- Every search is either successful or unsuccessful, we have

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

- We have probabilities of searches for each (dummy) key, we can determine the expected cost of a search in a given BST  $T$ .
- Assume that the actual cost of a search is the number of nodes examined. Then **the expected cost of a search in  $T$  is**



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

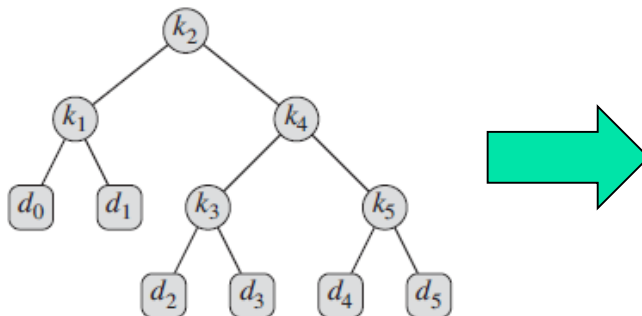
where  $\text{depth}_T$  denotes a node's depth in the tree  $T$ . (树根高度为0)

## 15.5 Optimal binary search trees

$E[\text{search cost in } T]$

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

In Figure 15.7(a), we can calculate the expected search cost **node by node**:



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

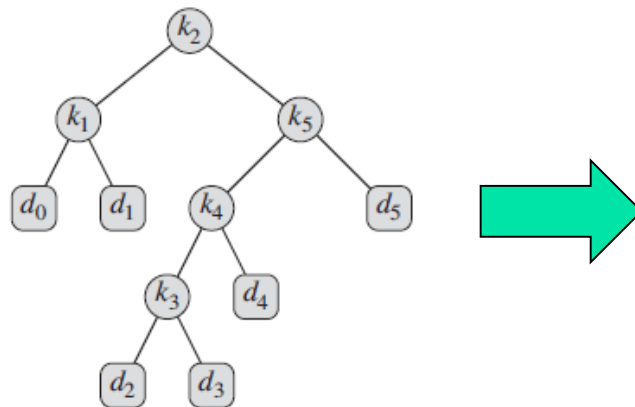
## 15.5 Optimal binary search trees

E[search cost in  $T$ ]

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

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

In Figure 15.7(b)

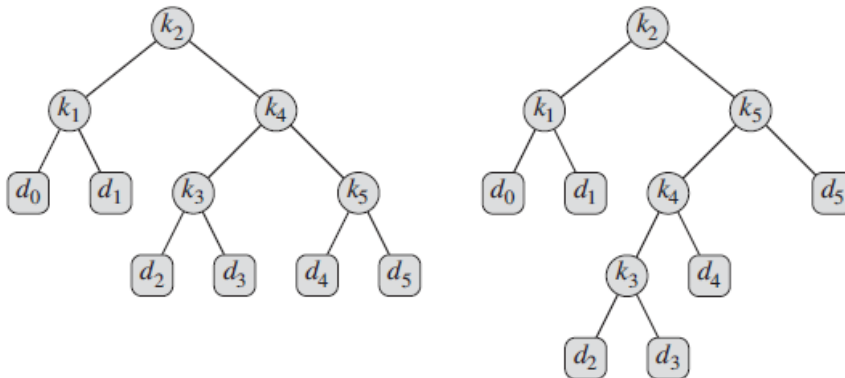


node	depth	probability	contribution
$k_1$	1	0.15	0.30
$k_2$	0	0.10	0.10
$k_3$	3	0.05	0.20
$k_4$	2	0.10	0.30
$k_5$	1	0.20	0.40
$d_0$	2	0.05	0.15
$d_1$	2	0.10	0.30
$d_2$	4	0.05	0.25
$d_3$	4	0.05	0.25
$d_4$	3	0.05	0.20
$d_5$	2	0.10	0.30
Total			2.75



## 15.5 Optimal binary search trees

- **Optimal BST** : for a given set of probabilities, our goal is to construct a BST whose expected search cost is the **smallest**.



$i$	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

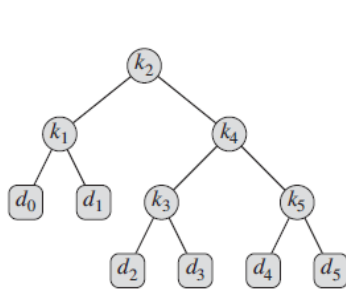
- How to build an OBST?

**Intuitively,**

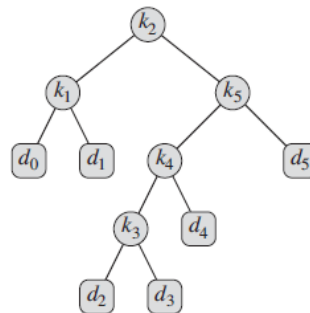
- ◆ the overall height is smallest
- ◆ the key with the greatest probability at the root

## 15.5 Optimal binary search trees

- **Optimal BST** : for a given set of probabilities, our goal is to construct a BST whose expected search cost is the **smallest**.



(a) cost: 2.80



(b) cost: 2.75

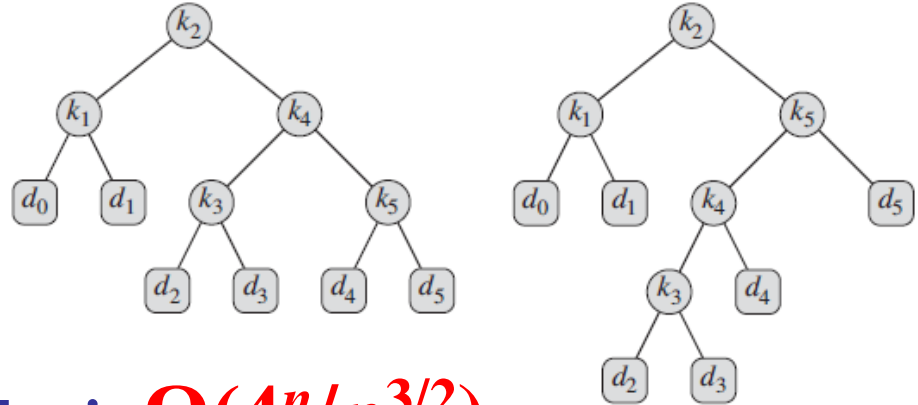
$i$	0	1	2	3	4	5
$p_i$		0.15	0.10	0.05	0.10	0.20
$q_i$	0.05	0.10	0.05	0.05	0.05	0.10

- Figure 15.7(b) shows an Optimal BST's expected cost is 2.75
  - ◆ An Optimal BST is not necessarily a tree whose overall height is smallest.  
不一定要要求树的高度最小
  - ◆ 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.

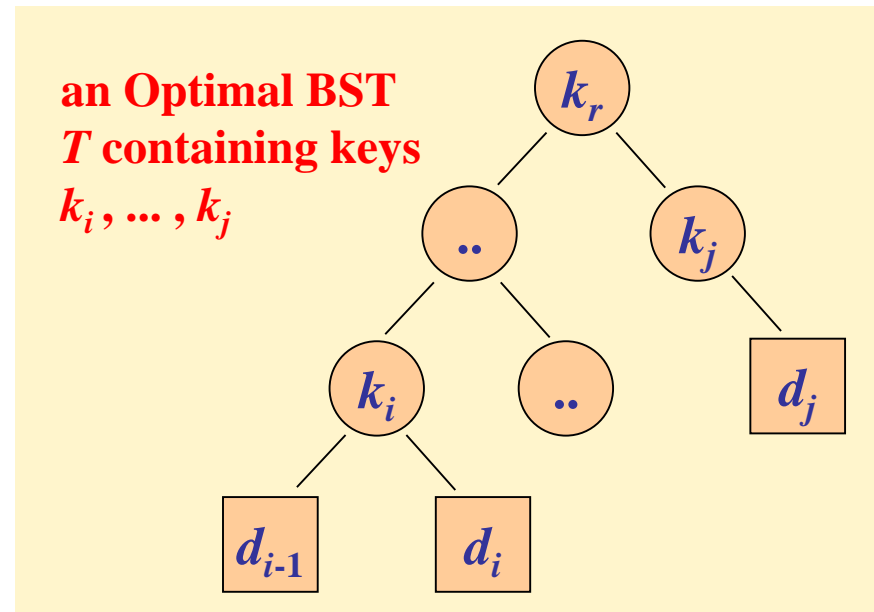
- ◆ ALS, RodCut, MCM



- 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

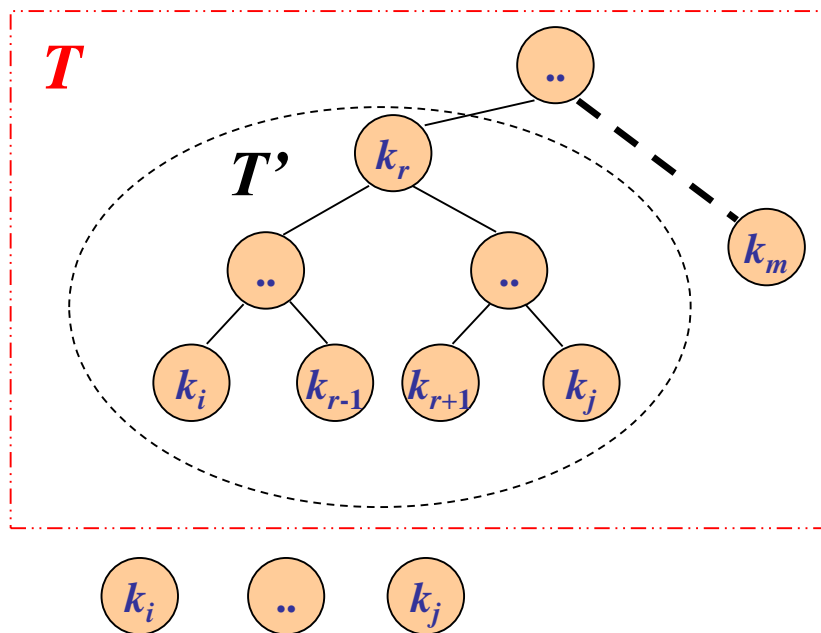
- Start with an observation about subtrees.
- Consider any **subtree** of a BST
  - ◆ It must contain keys in a contiguous range  $k_i, \dots, k_j$ , for some  $1 \leq i \leq j \leq n$ .
  - ◆ In addition, the subtree must also have as its leaves the dummy keys  $d_{i-1}, \dots, d_j$ .
- Optimal substructure?



## Step 1: The structure of an Optimal BST

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

设  $T'$  为最优BST  $T$  的一个子树,  $T'$  包含keys  $k_i, \dots, k_j$ , 那么  $T'$  是子问题〔关于 keys  $k_i, \dots, k_j$  和dummy keys  $d_{i-1}, \dots, d_j$ 〕的最优BST



$T$  : search tree of  $k_i, \dots, k_m$

$T'$  : search tree of  $k_i, \dots, k_j$

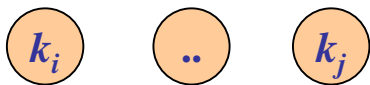
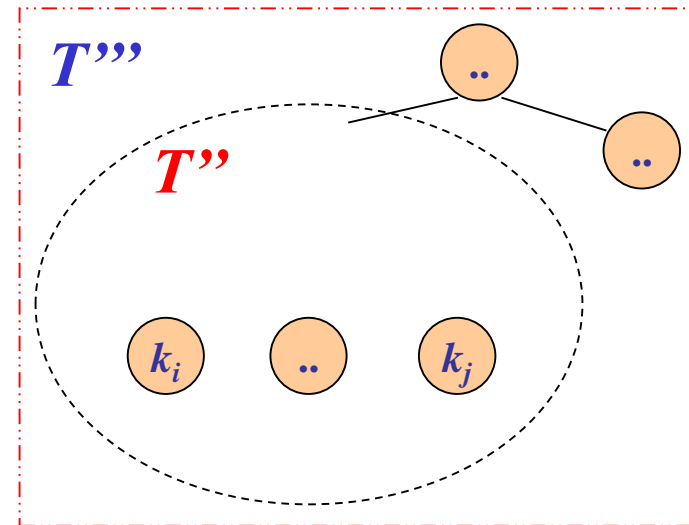
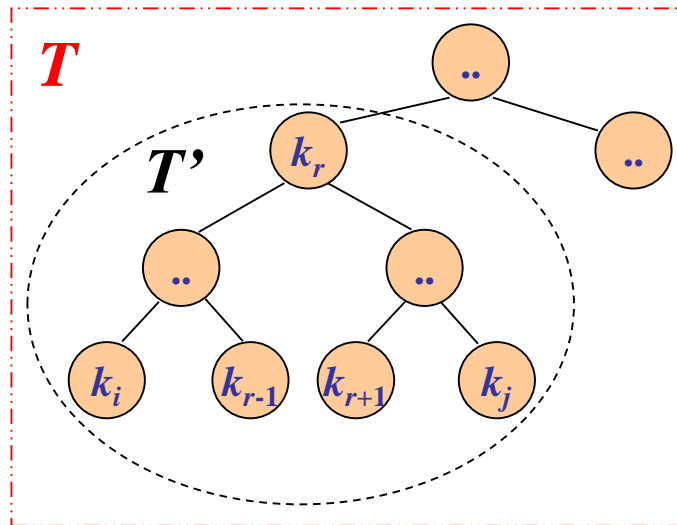
Optimal BST  $T$

➔ Optimal BST  $T'$

## Step 1: The structure of an Optimal BST

Idea of Proof: Cut-and-paste argument applies.

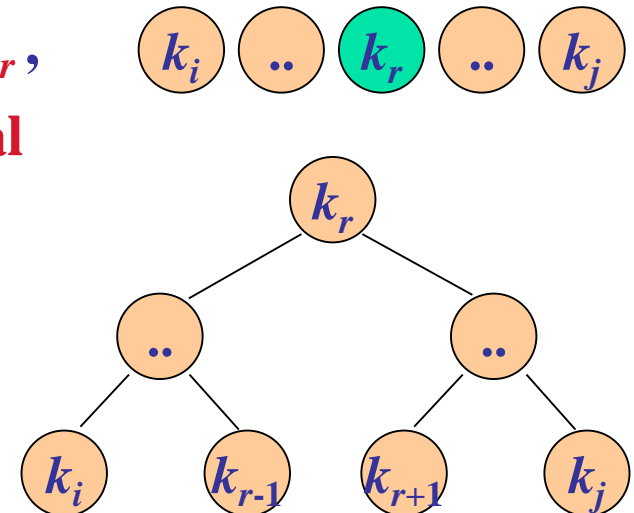
If there were a subtree  $T''$  whose expected cost is lower than that of  $T'$ , then we could cut  $T'$  out of  $T$  and paste in  $T''$ , resulting in a binary search tree of lower expected cost than  $T$ , thus contradicting the optimality of  $T$ .



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

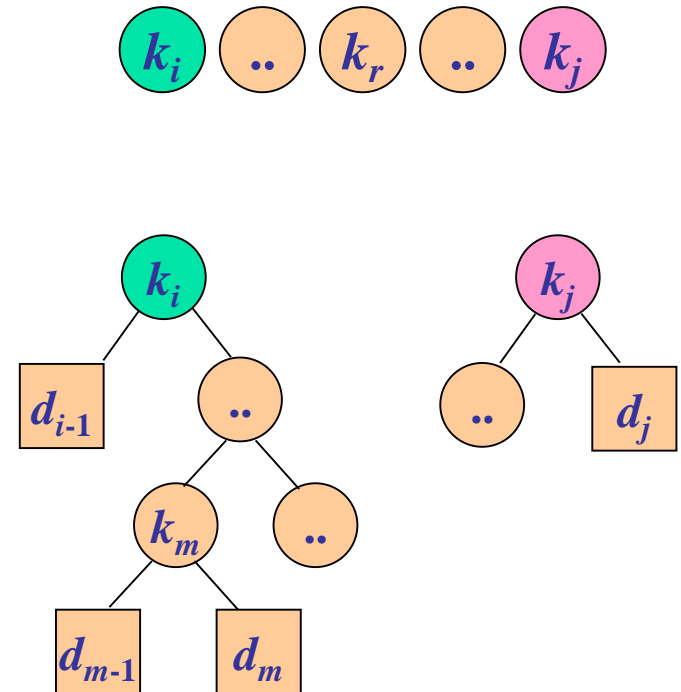
## Step 1: The structure of an Optimal BST

- Using the optimal substructure, we can construct an optimal solution to the problem from optimal solutions to subproblems.
- Given keys  $k_i, \dots, k_j$ , one of these keys, say  $k_r$  ( $i \leq r \leq j$ ), will be the root of an optimal subtree.
  - ♦ The left subtree of the root  $k_r$  will contain the keys  $k_i, \dots, k_{r-1}$  (and dummy keys  $d_{i-1}, \dots, d_{r-1}$ ); the right subtree will contain the keys  $k_{r+1}, \dots, k_j$  (and dummy keys  $d_r, \dots, d_j$ ).
- As long as we **examine all candidate roots  $k_r$** , where  $i \leq r \leq j$ , and we **determine all optimal BST containing  $k_i, \dots, k_{r-1}$  and those containing  $k_{r+1}, \dots, k_j$** , we will find an Optimal BST.



## Step 1: The structure of an Optimal BST

- A detail, “empty” subtrees
- Suppose that in a subtree with keys  $k_i, \dots, k_j$ ,
  - ◆ We select  $k_i$  as the root, left subtree of  $k_i$  contains no keys. Bear in mind, however, that subtrees also contain dummy keys  $d_{i-1}$ .
  - ◆ Symmetrically, if we select  $k_j$  as the root, right subtree of  $k_j$  contains the keys  $k_{j+1}, \dots, k_j$ ; this right subtree contains no actual keys, but it does contain the dummy key  $d_j$ .

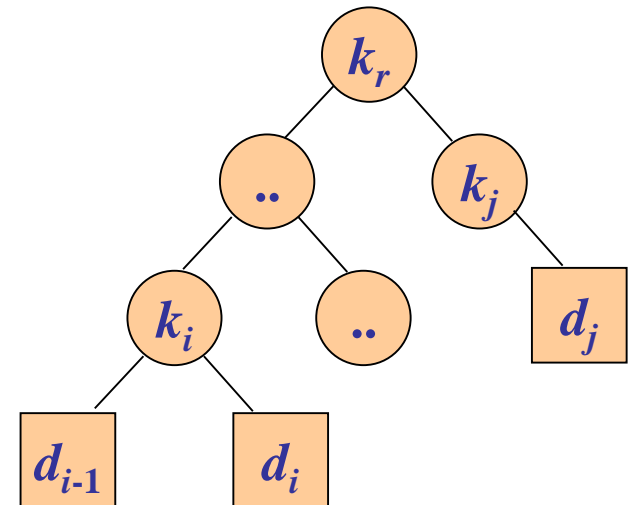
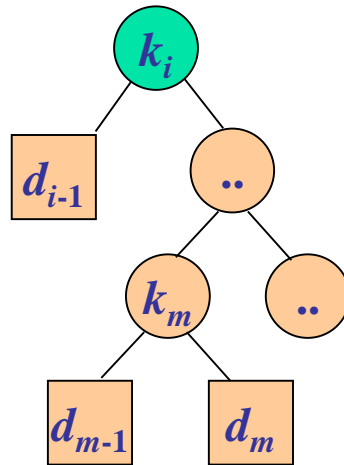




## Step 2: A recursive solution

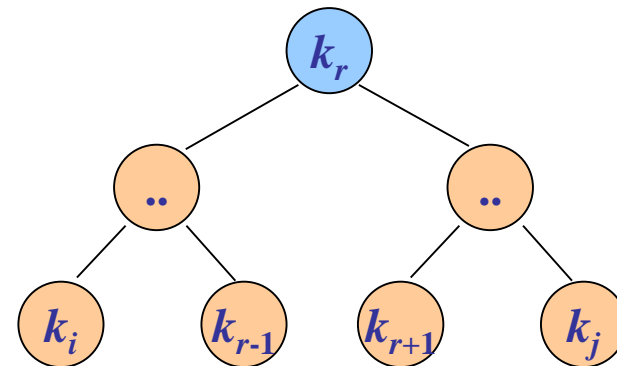
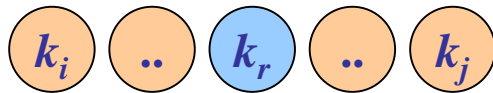
Subproblem : finding an Optimal BST containing the keys  $k_i, \dots, k_j$ , where  $i \geq 1, j \leq n$ , and  $j \geq i-1$ . (when  $j = i-1$ , there are no actual keys, we have just the dummy key  $d_{i-1}$ .)

- **$e[i, j]$**  : the expected cost of searching an Optimal BST containing the keys  $k_i, \dots, k_j$ .
- Ultimately, wish to compute  $e[1, n]$ .
- when  $j = i-1$ , only  $d_{i-1}$ ,  **$e[i, i-1] = q_{i-1}$** .
- When  $j \geq i$ ?



## Step 2: A recursive solution

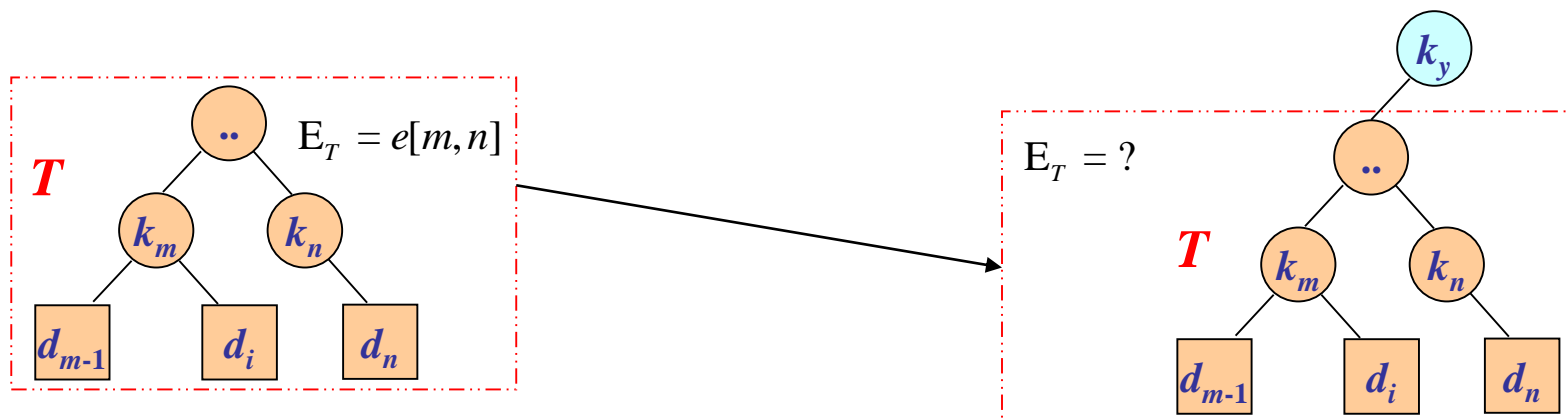
When  $j \geq i$ , **select a root  $k_r$**  from among  $k_i, \dots, k_j$ , and then make an Optimal BST with keys  $k_i, \dots, k_{r-1}$  its **left subtree** and an Optimal BST with keys  $k_{r+1}, \dots, k_j$  its **right subtree**.



## Step 2: A recursive solution

What happens to the expected search cost of a subtree  $T$  when it becomes a subtree of a node?

- ◆ The depth of each node in the subtree increases by 1, the expected search cost of this subtree increases by **the sum of all the probabilities** in the subtree.



**the sum of all the probabilities:**

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

$$\begin{aligned} E_T &= \sum_{x=m}^n (\text{depth}(k_x) + 1 + 1) \cdot p_i + \sum_{x=m-1}^n (\text{depth}(d_x) + 1 + 1) \cdot q_x \\ &= \sum_{x=m}^n (\text{depth}(k_x) + 1) \cdot p_i + \sum_{x=m-1}^n (\text{depth}(d_x) + 1) \cdot q_x + \sum_{x=m}^n p_i + \sum_{x=m-1}^n q_x \\ &= e[m, n] + w[m, n] \quad \text{增量为 } w[m, n] \end{aligned}$$

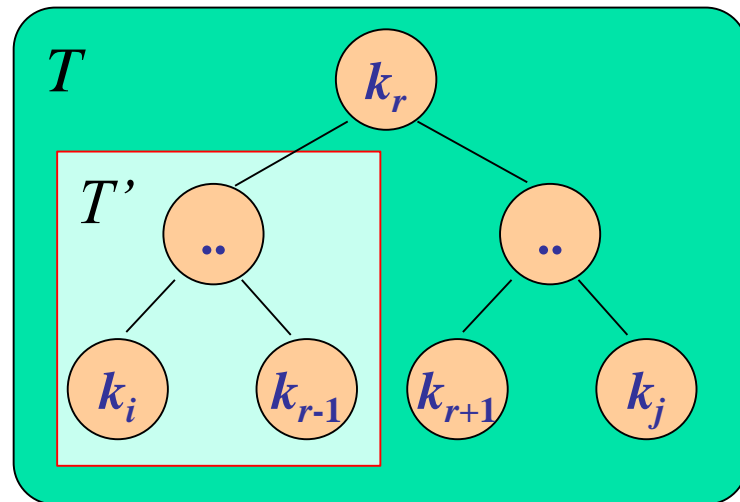
## Step 2: A recursive solution

OBST  $T$  与 OBS-subTree  $T'$  的关系:

if  $k_r$  is the root of an OBST

containing keys  $k_i, \dots, k_j$ , we have

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



Noting that  $w[i, j] = w[i, r-1] + p_r + w[r+1, j]$

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

We rewrite  $e[i, j]$  as

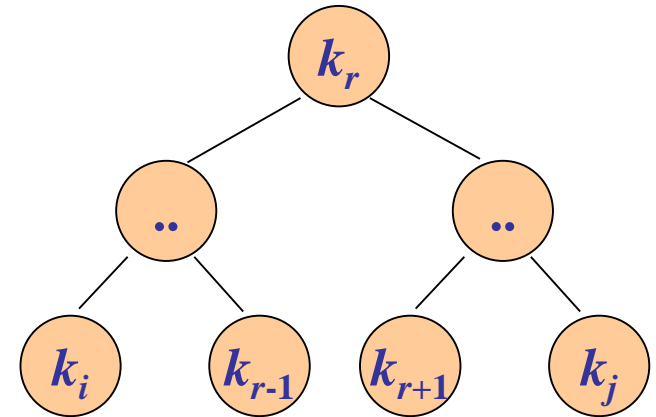
$$e[i, j] = e[i, r-1] + e[r+1, j] + w[i, j] \quad (15.18)$$

The recursive equation (15.18) assumes that we know which node  $k_r$  to use as the root, which we do not know.

## Step 2: A recursive solution

- Choose  $k_r$  as the root that gives the lowest expected search cost, giving us the final recursive formulation of an OBST cost  $e[i, j]$ :

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

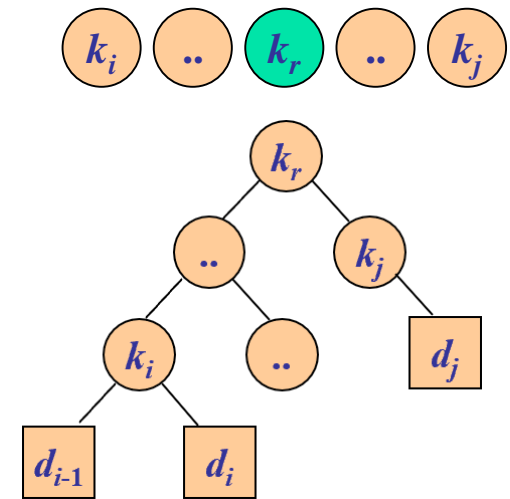


- To help us keep track of the structure of Optimal BST, define  **$root[i, j]$** , for  $1 \leq i \leq j \leq n$ , to be the index  $r$  for which  $k_r$  is the root of an Optimal BST containing keys  $k_i, \dots, k_j$ .

## Step 3: Computing the expected search cost

$$A_i \dots A_k A_{k+1} \dots A_j$$

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



- **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  $e[1, 0]$  .
- **root** $[i, j]$  , recording the root of the subtree containing keys  $k_i$  , ...,  $k_j$  .

## Step 3: Computing the expected search cost

- Other table for efficiency.

$$e[i, j] = e[i, r-1] + e[r+1, j] + w[i, j] \quad (15.18)$$

- Rather than compute the value of  $w[i, j]$  every time we are computing  $e[i, j]$ —which would take  $\Theta(j-i)$  additions—we store these values in a table  $w[1.. n+1, 0.. n]$ .

(无需每次计算  $e[i, j]$  时都计算  $w[i, j]$  , ... )

- For the base case, we compute  $w[i, i-1] = q_{i-1}$  for  $1 \leq i \leq n$ .
- For  $j \geq i$ ,
$$w[i, j] = \sum_{l=i}^j p_l + \sum_{l=i-1}^j q_l = w[i, j-1] + p_j + q_j \quad (15.20)$$
- Thus, compute the  $\Theta(n^2)$  values of  $w[i, j]$  in  $\Theta(1)$  time each.
- Inputs: the probabilities  $p_1, \dots, p_n$  and  $q_0, \dots, q_n$  and the size  $n$

## Step 3: Computing the expected search cost

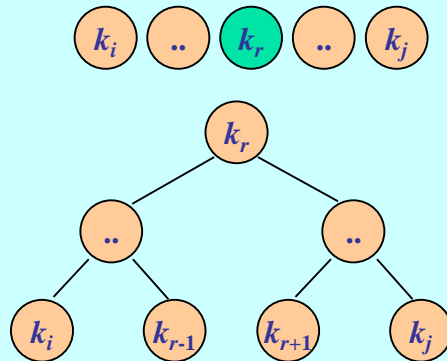
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i-1, \\ \min_{i \leq r \leq j} \{e[i, r-1] + e[r+1, j] + w[i, j]\} & \text{if } i \leq j. \end{cases} \quad (15.19)$$

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

OBST( $p, q, n$ )

```

1 for  $i \leftarrow 1$  to  $n+1$ 
2    $e[i, i-1] \leftarrow q_{i-1}$ 
3    $w[i, i-1] \leftarrow q_{i-1}$ 
4 for  $l \leftarrow 1$  to  $n$ 
5   for  $i \leftarrow 1$  to  $n-l+1$ 
6      $j \leftarrow i+l-1$ 
7      $e[i, j] \leftarrow \infty$ 
8      $w[i, j] \leftarrow w[i, j-1] + p_j + q_j$ 
9     for  $r \leftarrow i$  to  $j$ 
10       $t \leftarrow e[i, r-1] + e[r+1, j] + w[i, j]$ 
11      if  $t < e[i, j]$ 
12         $e[i, j] \leftarrow t$ 
13         $root[i, j] \leftarrow r$ 
14 return  $e$  and  $root$ 
```



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

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

VS

MCM-DP( $p$ )

```

1  $n \leftarrow \text{length}[p] - 1$ 
2 for  $i \leftarrow 1$  to  $n$ 
3    $m[i, i] \leftarrow 0$ 
4 for  $l \leftarrow 2$  to  $n$  //  $l$  is the chain length.
5   for  $i \leftarrow 1$  to  $n - l + 1$ 
6      $j \leftarrow i + l - 1$ 
7      $m[i, j] \leftarrow \infty$ 
8     for  $k \leftarrow i$  to  $j - 1$ 
9        $q \leftarrow m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$ 
10      if  $q < m[i, j]$ 
11         $m[i, j] \leftarrow q$ 
12         $s[i, j] \leftarrow k$ 
13 return  $m$  and  $s$ 
```



## Step 3: Computing the expected search cost

OBST( $p, q, n$ )

```
1 for  $i \leftarrow 1$  to  $n+1$ 
2    $e[i, i-1] \leftarrow q_{i-1}$ 
3    $w[i, i-1] \leftarrow q_{i-1}$ 
4 for  $l \leftarrow 1$  to  $n$ 
5   for  $i \leftarrow 1$  to  $n-l+1$  // ?1
6      $j \leftarrow i+l-1$  // ?2
7      $e[i, j] \leftarrow \infty$ 
8      $w[i, j] \leftarrow w[i, j-1] + p_j + q_j$ 
9     for  $r \leftarrow i$  to  $j$ 
10       $t \leftarrow e[i, r-1] + e[r+1, j] + w[i, j]$ 
11      if  $t < e[i, j]$ 
12         $e[i, j] \leftarrow t$ 
13         $root[i, j] \leftarrow r$ 
14 return  $e$  and  $root$ 
```

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

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

**?1**

$e[i, j]$ :

$l$  个元素的 Opti-BST 的 cost

$i = 1, j = l,$

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

...

$i = x, j = n,$

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

**?2**

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

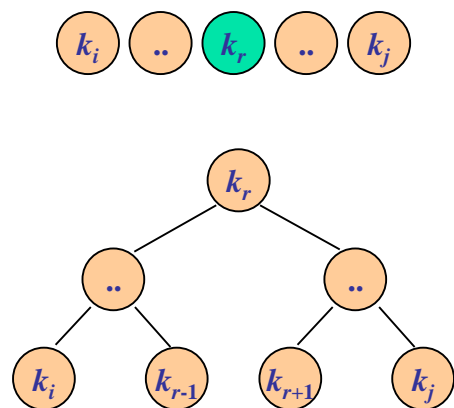
## Step 3: Computing the expected search cost

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

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

```
OBST( $p, q, n$ )
1  for  $i \leftarrow 1$  to  $n+1$ 
2     $e[i, i-1] \leftarrow q_{i-1}$ 
3     $w[i, i-1] \leftarrow q_{i-1}$ 
4  for  $l \leftarrow 1$  to  $n$  // 求  $l$  个元素的 Opti-BST
5    for  $i \leftarrow 1$  to  $n-l+1$ 
6       $j \leftarrow i+l-1$ 
7       $e[i, j] \leftarrow \infty$ 
8       $w[i, j] \leftarrow w[i, j-1] + p_j + q_j$ 
9      for  $r \leftarrow i$  to  $j$ 
10          $t \leftarrow e[i, r-1] + e[r+1, j] + w[i, j]$ 
11         if  $t < e[i, j]$ 
12              $e[i, j] \leftarrow t$ 
13              $root[i, j] \leftarrow r$ 
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  $k_i, \dots, k_j$ . 对包含  $k_i, \dots, k_j$  的最优 BST，遍历每一个  $k_r$  作为树根，...



## Step 3: Computing the expected search cost

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

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

OBST( $p, q, n$ )

```

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

		$j$						
		0	1	2	3	4	5	$e$
$i$	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	

		$j$						
		0	1	2	3	4	5	$w$
$i$	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	

		$j$					
		1	2	3	4	5	$root$
$i$	1	1	1	2	2	2	1
	2		2	2	2	4	2
	3			3	4	5	3
	4				4	5	4
	5					5	5

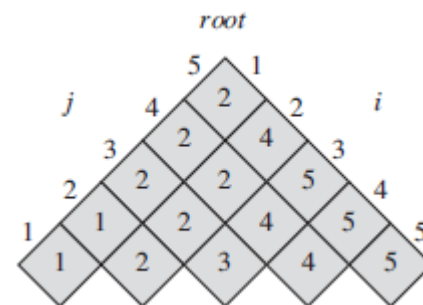
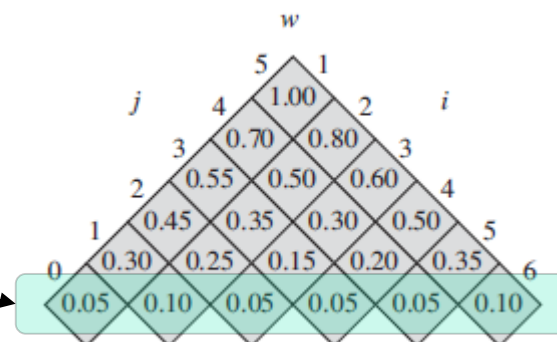
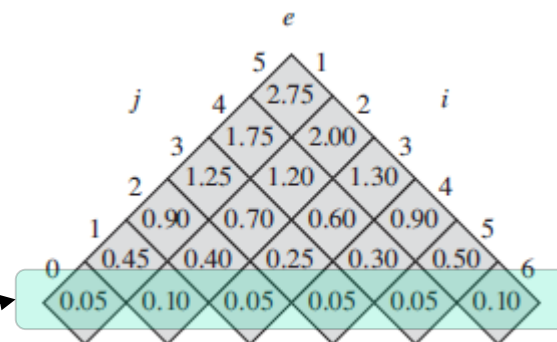
## Step 3: Computing the expected search cost

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

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

```

OBST(p, q, n)
1  for i ← 1 to n+1
2    e[i, i-1] ← qi-1
3    w[i, i-1] ← qi-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] + pj + qj
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
    
```



## Step 3: Computing the expected search cost

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

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

```
OBST(p, q, n)
1  for i ← 1 to n+1
2    e[i, i-1] ← qi-1
3    w[i, i-1] ← qi-1
4  for l ← 1 to n
5    for i ← 1 to n-l+1 // n-l+1 times
6      j ← i+l-1
7      e[i, j] ← ∞
8      w[i, j] ← w[i, j-1] + pj + qj
9      for r ← i to j // j-i+1 = i+l-1-i+1 = l
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
```

Running times ?

## Exercise-5 (in class)

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

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

**Based on equations (15.19) and (15.20) from the OBST,**

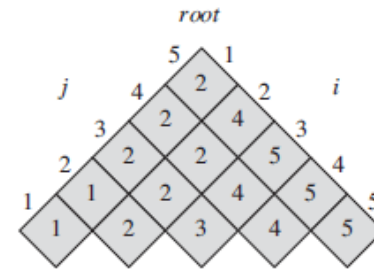
**What is the raw Recursive algorithm?**

**What is the Recursive algorithm with Memoization?**

**Running time?**

## Exercise-6

**15.5-1** Write pseudocode for the procedure **CONSTRUCT-OPTIMAL-BST(root)** which, given the table *root*, outputs the structure of an optimal binary search tree. For the example in Figure 15.8, your procedure should print out the



*k*2 is the root  
*k*1 is the left child of *k*2  
*d*0 is the left child of *k*1  
*d*1 is the right child of *k*1  
*k*5 is the right child of *k*2  
*k*4 is the left child of *k*5  
*k*3 is the left child of *k*4  
*d*2 is the left child of *k*3  
*d*3 is the right child of *k*3  
*d*4 is the right child of *k*4  
*d*5 is the right child of *k*5

# Solution of Exercise-6

**15.5-1**

```
A(r, i, j)
1  if i=1 && j=n
2    r[i, j] is the root
3  if i=j
4    print "d"i-1 "is the left child of k"i
5    print "d"i "is the right child of k"i
6  else if r[i, j]=i
7    print "d"i-1 "is the left child of k"i
8    print "k"r[i+1, j] "is the right child of k"r[i, j]
9    A(r, i+1, j)
10 else if r[i, j]=j
11   print "k"r[i, j-1] "is the left child of k"r[i, j]
12   A(r, i, j-1)
13   print "d"j "is the right child of k"j
14 else // i<r[i, j]<j
15   print "k"r[i, r[i, j]-1] "is the left child of k"r[i, j]
16   A(r, i, r[i, j]-1)
17   print "k"r[r[i, j]+1, j] "is the right child of k"r[i, j]
18   A(r, r[i, j]+1, j)
```



# Big Exercises

---

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  $i$  ( $1 \leq i \leq n$ ) 个单词在书中出现了  $k_i$  次  
( $k_1 + k_2 + \dots + k_n = M$ )，其词频  $p_i = k_i / M$
3. 根据词频表，构建OBST

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

# Eggs



## D 生命之水

时间限制: 1000ms 内存限制: 65536kb

通过率: 162/200 (81.00%) 正确率: 162/493 (32.86%)

### 题目描述

经过你辛勤的浇灌和修剪，你的绿藤长势极好，甚至分出了  $n$  条绿枝，但最近绿藤的生长停滞了，似乎遇到了瓶颈。

你偶然间得到了生命之水，传说生命之水可以滋养万物，提供生命之力，于是你想到了你心爱的绿藤。

每当你使用生命之水滋养一条绿枝时，绿藤可以获得  $x_i$  的成长增益值，但是你不能同时滋养相邻的两条绿枝，这会触发生命之水的副作用，使绿藤失去生机。

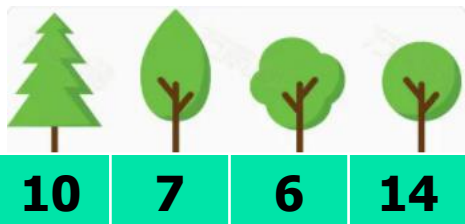
现在，你想知道在不触发副作用的情况下绿藤可以获得的成长增益值最大是多少？

### 输入格式

第一行包含一个整数  $t$ ，表示测试数据的数量，接下来  $t$  组测试数据。

每组测试数据包含两行。

第一行包含一个正整数  $n$  ( $2 \leq n \leq 10^5$ )，含义如上。



# Eggs

## D 生命之水

输入

2

3

1 8 2

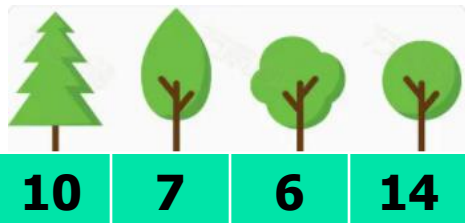
4

10 7 6 14

输出

8

24



方法一:

$n$ 颗小树的成长值记为  $a_1, a_2, a_3, \dots, a_n$

浇灌第 $i \sim j$ 颗小树获得的最大值为 $m[i, j]$ , 若浇灌 $a_k$ , 有

$m[i, j] = \max(m[i, k-2] + a_k + m[k+2, j]), i \leq k \leq j$

$m[i, j] = a_i, i=j$  时,

$m[i, j] = 0, i > j$  时.

按题意,  
空间复杂度超了  
计算时时间复杂度也超

方法二:

$m[j]$ 表示浇灌 $1 \sim j$ 颗小树获得的最大值, 则

$m[j] = \max(m[j-2] + a_j, m[j-1])$

$m[1] = a_1,$

$m[2] = \max(a_1, a_2).$

符合题意

# Exercises

---

## ADF WorkShop