

طراحی الگوریتم ها

هفته دوم اسفند

جلسه ۳، ۴ و ۵

ملکی مجد

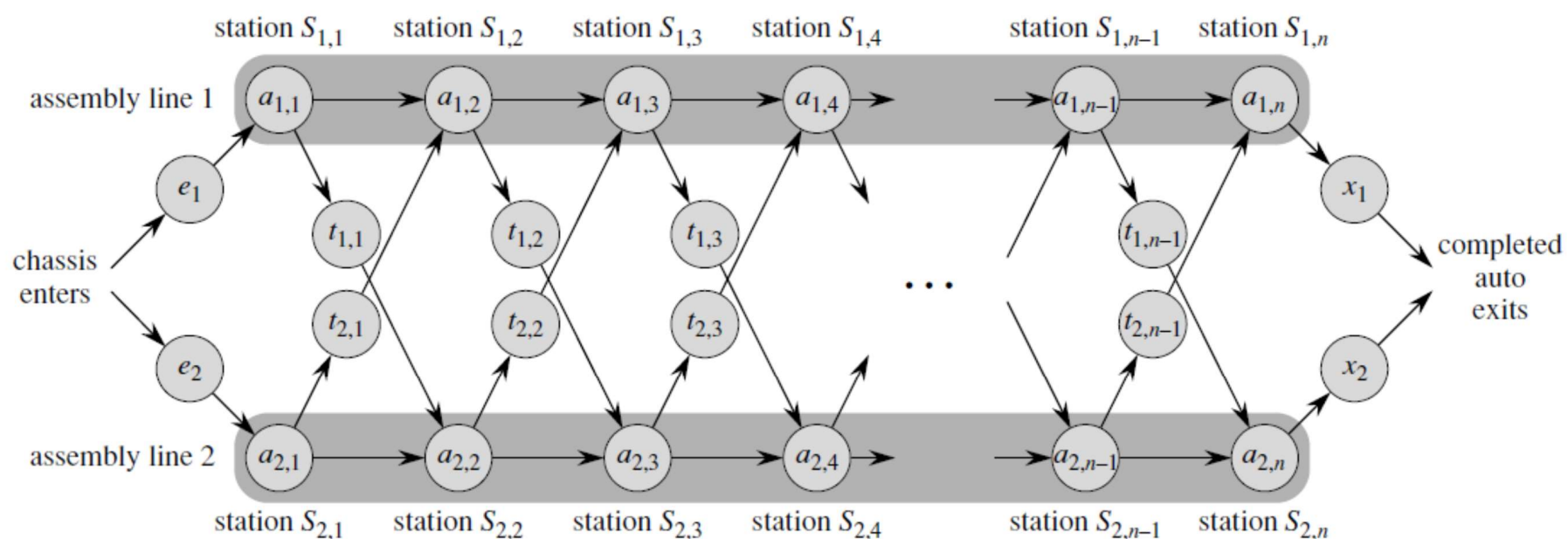
مباحث این هفته

- برنامه نویسی پویا
- مساله خط تولید
- ضرب ماتریسی
- بزرگترین زیر رشته مشترک
- برش میله
- درخت جستجوی دودویی بهینه

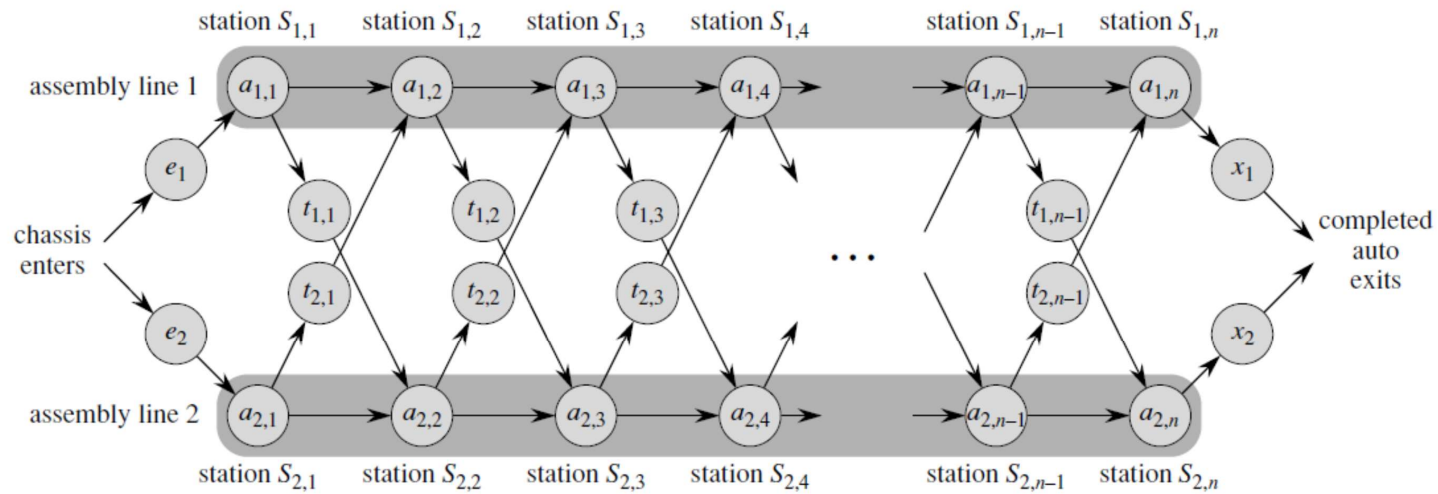
مبحث برنامه نویسی پویا از فصل ۱۵ کتاب CLRS تدریس می شود.

- برنامه نویسی پویا یک روش حل مساله است
- کاربرد آن برای وقتی است که در حل مساله با روش بخش مساله به مساله های کوچک تر، زیر مساله های مشابه داشته باشیم.
- در روش DP هر زیرمساله یک بار حل می شود و نتیجه آن در یک جدول ذخیره می شود
- به طور معمول برای مساله های بهینه سازی استفاده می شود

زمانبندی خط تولید Assembly-line scheduling



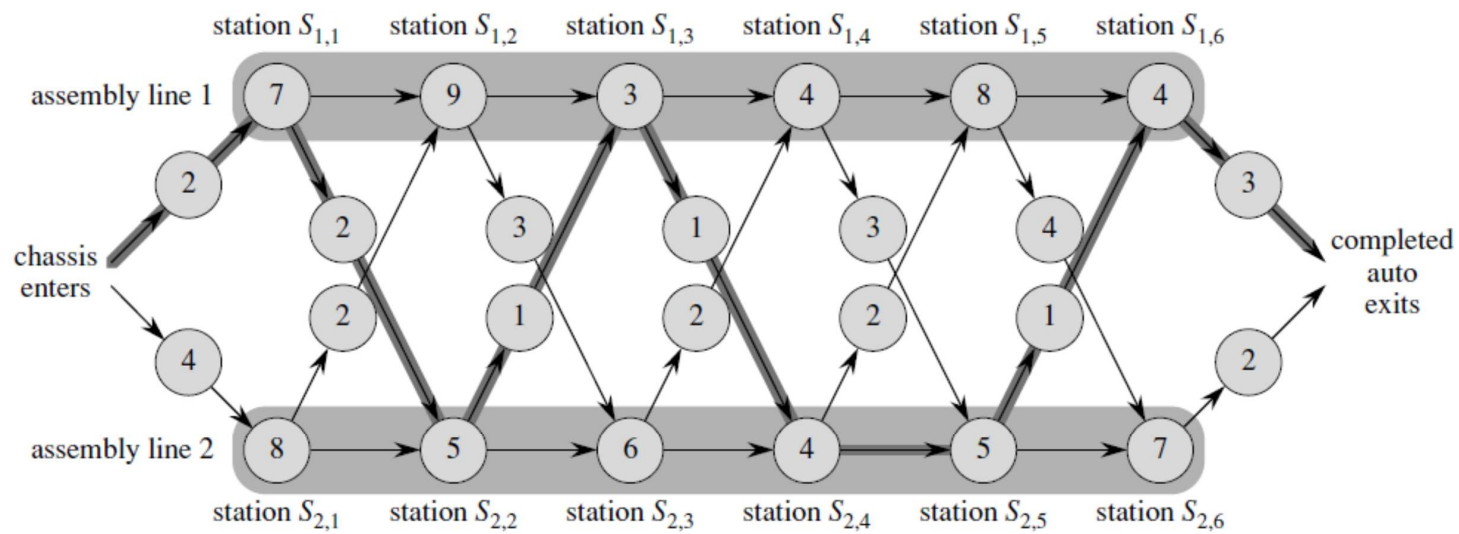
زمانبندی خط تولید Assembly-line scheduling



$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1, \\ \min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}) & \text{if } j \geq 2 \end{cases}$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1, \\ \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}) & \text{if } j \geq 2 \end{cases}$$

مثال



(a)

| | | | | | | |
|----------|----|----|----|----|----|----|
| j | 1 | 2 | 3 | 4 | 5 | 6 |
| $f_1[j]$ | 9 | 18 | 20 | 24 | 32 | 35 |
| $f_2[j]$ | 12 | 16 | 22 | 25 | 30 | 37 |

$f^* = 38$

| | | | | | |
|----------|---|---|---|---|---|
| j | 2 | 3 | 4 | 5 | 6 |
| $l_1[j]$ | 1 | 2 | 1 | 1 | 2 |
| $l_2[j]$ | 1 | 2 | 1 | 2 | 2 |

$l^* = 1$

(b)

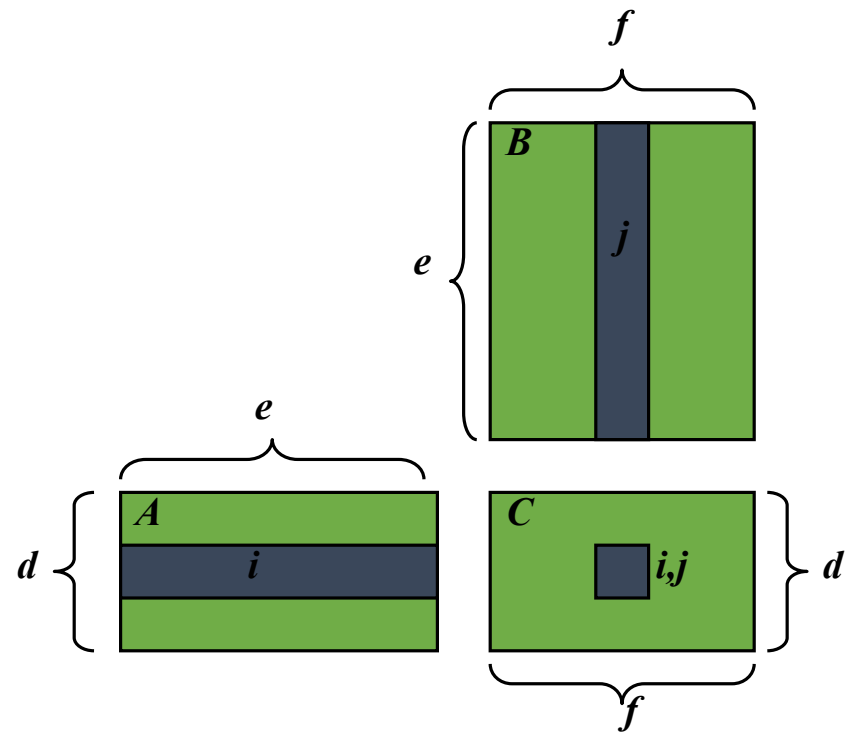
FASTEST-WAY(a, t, e, x, n)

```
1   $f_1[1] \leftarrow e_1 + a_{1,1}$ 
2   $f_2[1] \leftarrow e_2 + a_{2,1}$ 
3  for  $j \leftarrow 2$  to  $n$ 
4      do if  $f_1[j-1] + a_{1,j} \leq f_2[j-1] + t_{2,j-1} + a_{1,j}$ 
5          then  $f_1[j] \leftarrow f_1[j-1] + a_{1,j}$ 
6               $l_1[j] \leftarrow 1$ 
7          else  $f_1[j] \leftarrow f_2[j-1] + t_{2,j-1} + a_{1,j}$ 
8               $l_1[j] \leftarrow 2$ 
9      if  $f_2[j-1] + a_{2,j} \leq f_1[j-1] + t_{1,j-1} + a_{2,j}$ 
10         then  $f_2[j] \leftarrow f_2[j-1] + a_{2,j}$ 
11              $l_2[j] \leftarrow 2$ 
12         else  $f_2[j] \leftarrow f_1[j-1] + t_{1,j-1} + a_{2,j}$ 
13              $l_2[j] \leftarrow 1$ 
14 if  $f_1[n] + x_1 \leq f_2[n] + x_2$ 
15     then  $f^* = f_1[n] + x_1$ 
16          $l^* = 1$ 
17     else  $f^* = f_2[n] + x_2$ 
18          $l^* = 2$ 
```

ضرب ماتریسی

- $C = A_{d \cdot e} * B_{e \cdot f}$
- time = $O(d \cdot e \cdot f)$

$$C[i, j] = \sum_{k=0}^{e-1} A[i, k] * B[k, j]$$



ضرب زنجیره ماتریس Matrix-chain multiplication

- ترتیب انجام ضرب ماتریسی با پرانتزگذاری مشخص می شود
- ابعاد ماتریس ها در زنجیره ضرب باید همخوانی داشته باشد
- نمونه ای از پرانتزگذاری ضرب ۴ ماتریس

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

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

$$((A_1A_2)(A_3A_4))$$

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

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

ضرب زنجیره ماتریس Matrix-chain multiplication

• پرانتزگذاری بر تعداد ضرب های نهایی تاثیر می گذارد

- A1 with dimensions 10×100
- A2 with dimensions 100×5
- A3 with dimensions 5×50
- $((A1 \ A2)A3) \rightarrow (10.100.5)+(10.5.50)=7500$
- $(A1(A2 \ A3)) \rightarrow (100.5.50)+(10.100.50)=75000$

ضرب زنجیره ماتریس Matrix-chain multiplication
(تعداد حالت های مختلف پرانتزگذاری)

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

Catalan numbers, which grows as $\Omega(4^n / n^{3/2})$

ضرب زنجیره ماتریس Matrix-chain multiplication

- ساختار پرانتزگذاری بهینه:
- برای محاسبه ضرب ماتریس های $A_i A_{i+1} \dots A_j$ ابتدا برای یک k ضرب $A_i \dots A_{k-1}$ و $A_k \dots A_j$ محاسبه می شوند
- $A_i \dots A_{k-1}$ و $A_k \dots A_j$ باید به صورت بهینه پرانتز گذاری شده باشند
- چرا؟ (قاعده cut and paste)

ضرب زنجیره ماتریس Matrix-chain multiplication

- رابطه بازگشتی (محاسبه ضرب ماتریس های $A_i A_{i+1} \cdots A_j$)

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

ضرب زنجیره ماتریس Matrix-chain multiplication

محاسبه مقدار بهینه

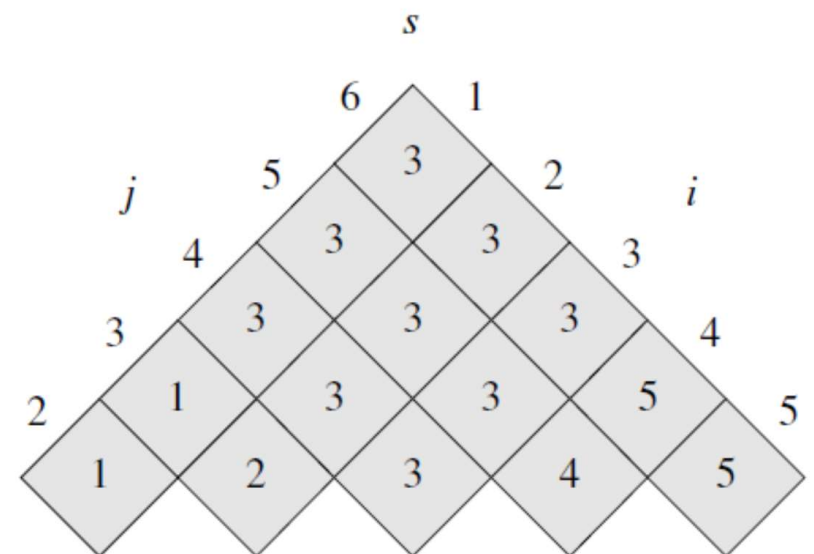
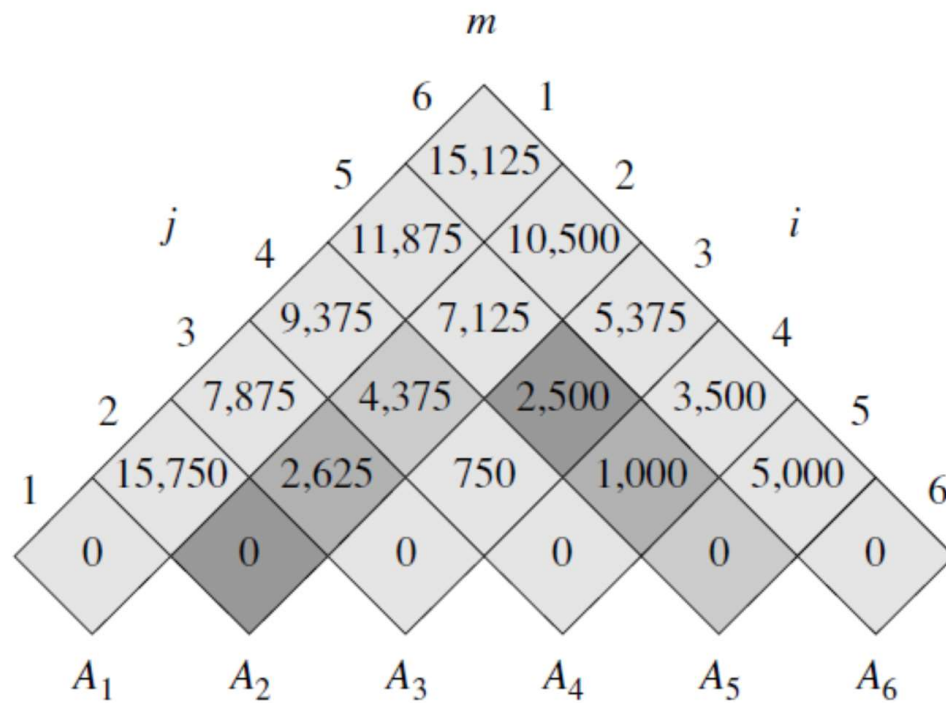
MATRIX-CHAIN-ORDER(p)

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

تعیین مقدار $j \leq k \leq i$ برای تقسیم زنجیره ماتریس های $A_i \dots A_j$

ضرب زنجیره ماتریس Matrix-chain multiplication

مثال



| matrix | dimension |
|--------|----------------|
| A_1 | 30×35 |
| A_2 | 35×15 |
| A_3 | 15×5 |
| A_4 | 5×10 |
| A_5 | 10×20 |
| A_6 | 20×25 |

چاپ پرانتز گذاری ضرب ماتریس ها

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

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


ضرب زنجیره ماتریس Matrix-chain multiplication محاسبه مقدار بهینه

- تعداد ضرب ها را محاسبه می کنیم (نه اینکه مقدار ضرب را محاسبه کنیم)
- الگوریتم از مرتبه زمانی $O(n^3)$ است و فضای مورد نیاز $O(n^2)$ است

When Dynamic Programming applies?

- Optimal substructure
 - Proof: cut and paste!
 - how many subproblems + how many choices
- Overlapping subproblems
 - the total number of distinct subproblems is a polynomial in the input size.

Longest common subsequence

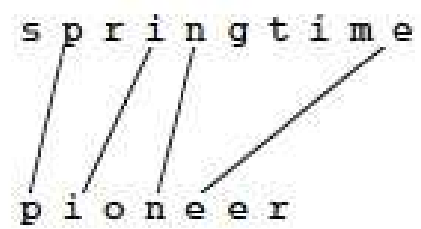
- $S1 = \text{ACCGGTCGAGTGCGCGGAAGCCGGCCGAA}$
- $S2 = \text{GTCGTTCGGAATGCCGTTGCTCTGTAAA}$

GTCGTCGGAAGCCGGCCGAA

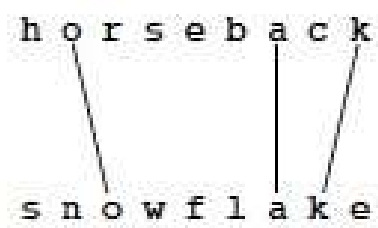
ACC**GGTCG**AG**TGCG**CGGA**AAGCCGGCCGAA**

GTCGTTCGGAATGCCGTTGCTCTGTAAA

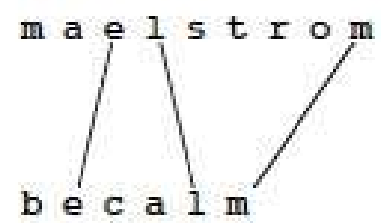
s p r i n g t i m e
p i o n e e r



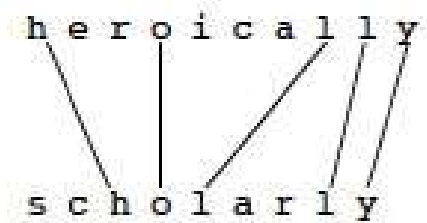
h o r s e b a c k
s n o w f l a k e



m a e l s t r o m
b e c a l m



h e r o i c a l l y
s c h o l a r l y



Longest common subsequence

Characterizing a longest common subsequence

Theorem 15.1 (Optimal substructure of an LCS)

Let $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y .

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1} .
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y .
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1} .

Longest common subsequence

A recursive solution

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 , \\ c[i - 1, j - 1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j , \\ \max(c[i, j - 1], c[i - 1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j . \end{cases}$$

Longest common subsequence

Computing the length of an LCS

```
LCS-LENGTH( $X, Y$ )
1   $m \leftarrow \text{length}[X]$ 
2   $n \leftarrow \text{length}[Y]$ 
3  for  $i \leftarrow 1$  to  $m$ 
4      do  $c[i, 0] \leftarrow 0$ 
5  for  $j \leftarrow 0$  to  $n$ 
6      do  $c[0, j] \leftarrow 0$ 
7  for  $i \leftarrow 1$  to  $m$ 
8      do for  $j \leftarrow 1$  to  $n$ 
9          do if  $x_i = y_j$ 
10             then  $c[i, j] \leftarrow c[i - 1, j - 1] + 1$ 
11                  $b[i, j] \leftarrow \nwarrow$ 
12             else if  $c[i - 1, j] \geq c[i, j - 1]$ 
13                 then  $c[i, j] \leftarrow c[i - 1, j]$ 
14                      $b[i, j] \leftarrow \uparrow$ 
15                 else  $c[i, j] \leftarrow c[i, j - 1]$ 
16                      $b[i, j] \leftarrow \leftarrow$ 
17  return  $c$  and  $b$ 
```

Longest common subsequence

Computing the length of an LCS

```
PRINT-LCS( $b, X, i, j$ )
1  if  $i = 0$  or  $j = 0$ 
2      then return
3  if  $b[i, j] = \nwarrow$ 
4      then PRINT-LCS( $b, X, i - 1, j - 1$ )
5          print  $x_i$ 
6  elseif  $b[i, j] = \uparrow$ 
7      then PRINT-LCS( $b, X, i - 1, j$ )
8  else PRINT-LCS( $b, X, i, j - 1$ )
```


جدول b و c را بکشید.

- Use Dynamic programming to find the LCS of two following strings:
- Str1 = "humans";
- Str2 "chimpanzees"



longest common subsequence problem for multiple string

- For the general case of an arbitrary number of input sequences, the problem is NP-hard
- the dynamic programming approach gives a solution in

$$O(N \prod_{i=1}^N n_i)$$

Rod cutting

- بخش ۱۵.۱ کتاب CLRS را مطالعه کنید
- از مسایل ساده و ابتدایی مبحث برنامه نویسی پویا است.
- هدف تمرین خودخوانی یک مبحث است.
- انتهای جلسه ۵شنبه در صورت نیاز این مبحث رفع اشکال می شود.

Think together 😊

- پیدا کردن بزرگ ترین رشته از دو سری یکی (palindrome) ؟

Look-up time

Optimal Binary Search Trees

- 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
- **total time** spent searching to be as low as possible
 - ensure an $O(\lg n)$ search time per occurrence by using a red-black tree

Why not a a red-black tree?

- case that a frequently used word such as “the” appears far from the root while a rarely used word such as “mycophagist” appears near the root
 - slow down the translation
- > the number of nodes visited when searching for a key in a binary search tree equals one plus the depth of the node containing the key
- Want words that occur frequently in the text to be placed nearer the root

What we have?

- Words with different frequently
- some words in the text might have no French translation,

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

Optimal Binary Search Trees

formal problem definition

given a sequence $K = \langle k_1, k_2, \dots, k_n \rangle$ of n distinct keys in sorted order
 $k_1 < k_2 < \dots < k_n$

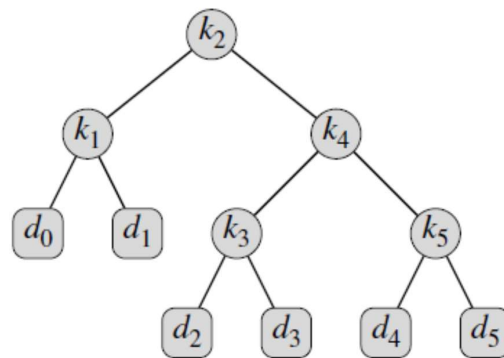
$d_0, d_1, d_2, \dots, d_n$ representing values not in K

d_i represents all values between k_i and k_{i+1} .

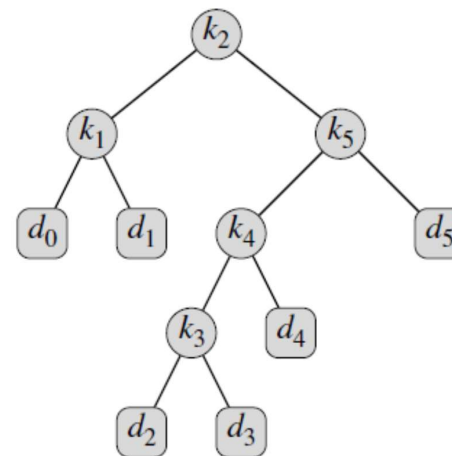
$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1$$

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

Optimal Binary Search Trees example



expected search cost 2.80.



expected search 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 |

Optimal Binary Search Trees

The structure of an optimal binary search tree (1)

range k_i, \dots, k_j , for some $1 \leq i \leq j \leq n$.

a subtree keys k_i, \dots, k_j
leaves dummy keys d_{i-1}, \dots, d_j

- if an optimal BS tree T has a subtree T' containing keys i to j
 - then this subtree T' must be optimal as well
 - cut-and-paste argument applies

Optimal Binary Search Trees

The structure of an optimal binary search tree (2)

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

left subtree

⋮

right subtree

k_i, \dots, k_{r-1} (and dummy keys d_{i-1}, \dots, d_{r-1}) keys k_{r+1}, \dots, k_j (and dummy keys d_r, \dots, d_j)

k_i 's left subtree contains the keys k_i, \dots, k_{i-1}

keys k_i, \dots, k_{i-1} has no actual keys but does contain the single dummy key d_{i-1} .

Optimal Binary Search Trees

A recursive solution (1)

- $e[1, n]$
- easy case occurs when $j = i - 1$.

Optimal Binary Search Trees

A recursive solution (2)

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

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

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

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

Optimal Binary Search Trees

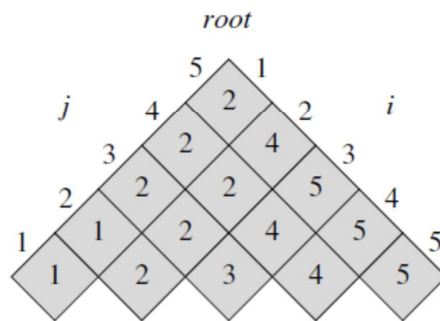
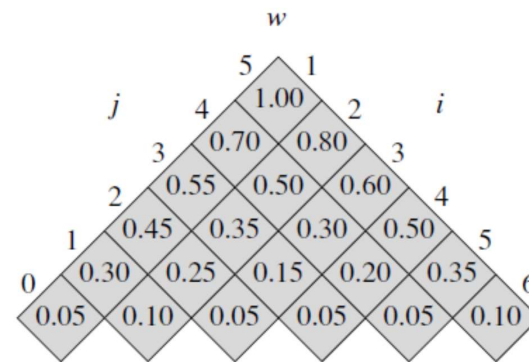
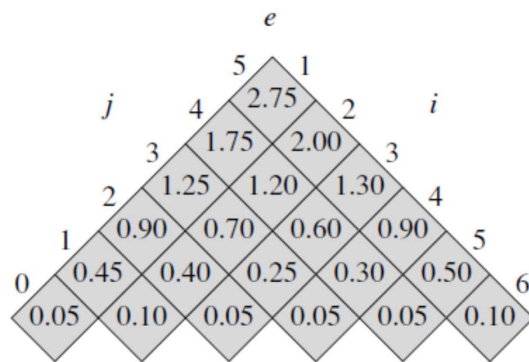
Computing the expected search cost of an optimal binary search tree

OPTIMAL-BST(p, q, n)

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


Optimal Binary Search Trees

example



| <i>i</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 |

رفع اشكال مساله rod cutting