طراحی الگوریتم ها

جلسه ۲، ۳، ۴ و ۵

ملکی مجد

سوال هایی مبحث برنامه نویسی پویا

- برنامه نویسی پویا
  - مساله خط تولید (جلسه دوم)
  - ضرب ماتریسی (جلسه سوم)
  - بزرگترین زیر رشته مشترک (جلسه چهارم)
  - برش میله
  - درخت جستجوی دودویی بهینه

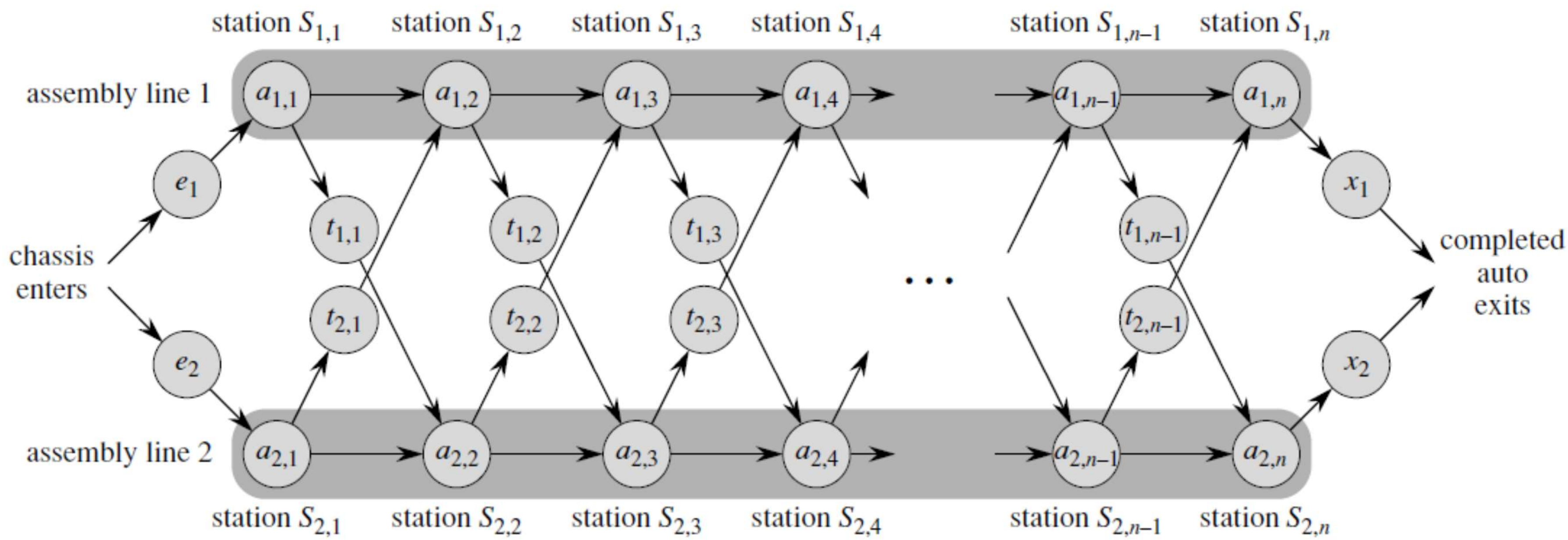
مبحث برنامه نویسی پویا از فصل ۱۵ کتاب CLRS تدریس می شود.

- 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، زیرمساله 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.

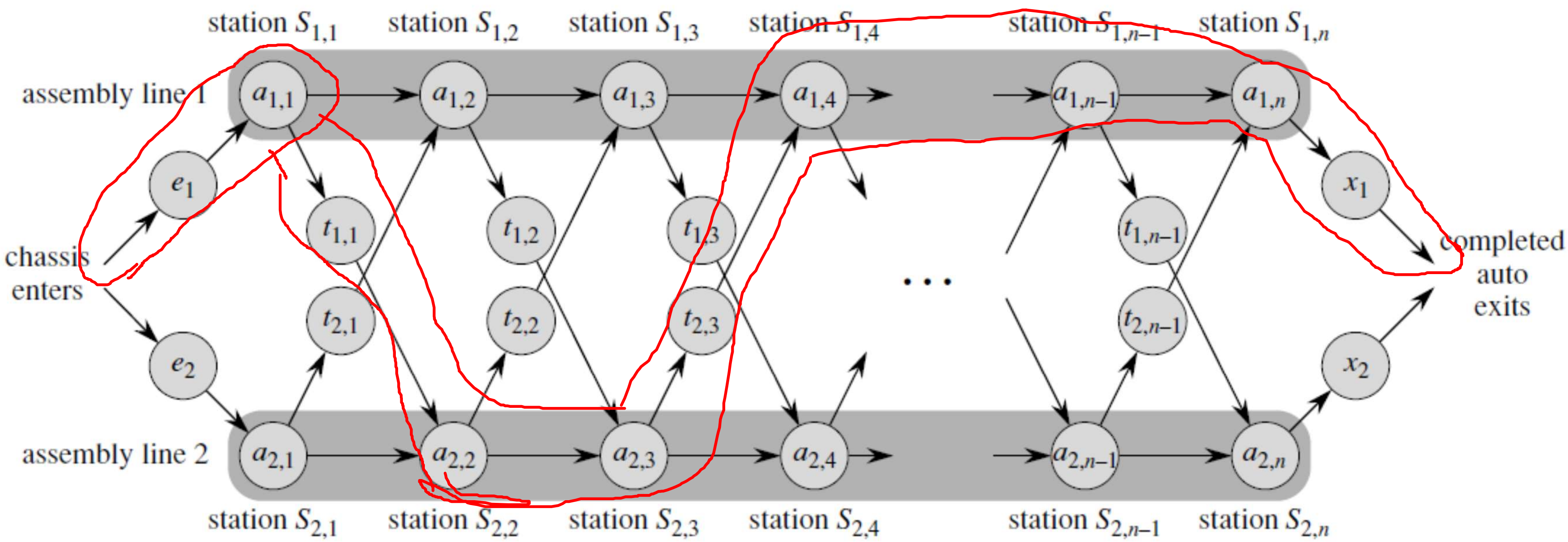
When Dynamic Programming applies?

- 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.

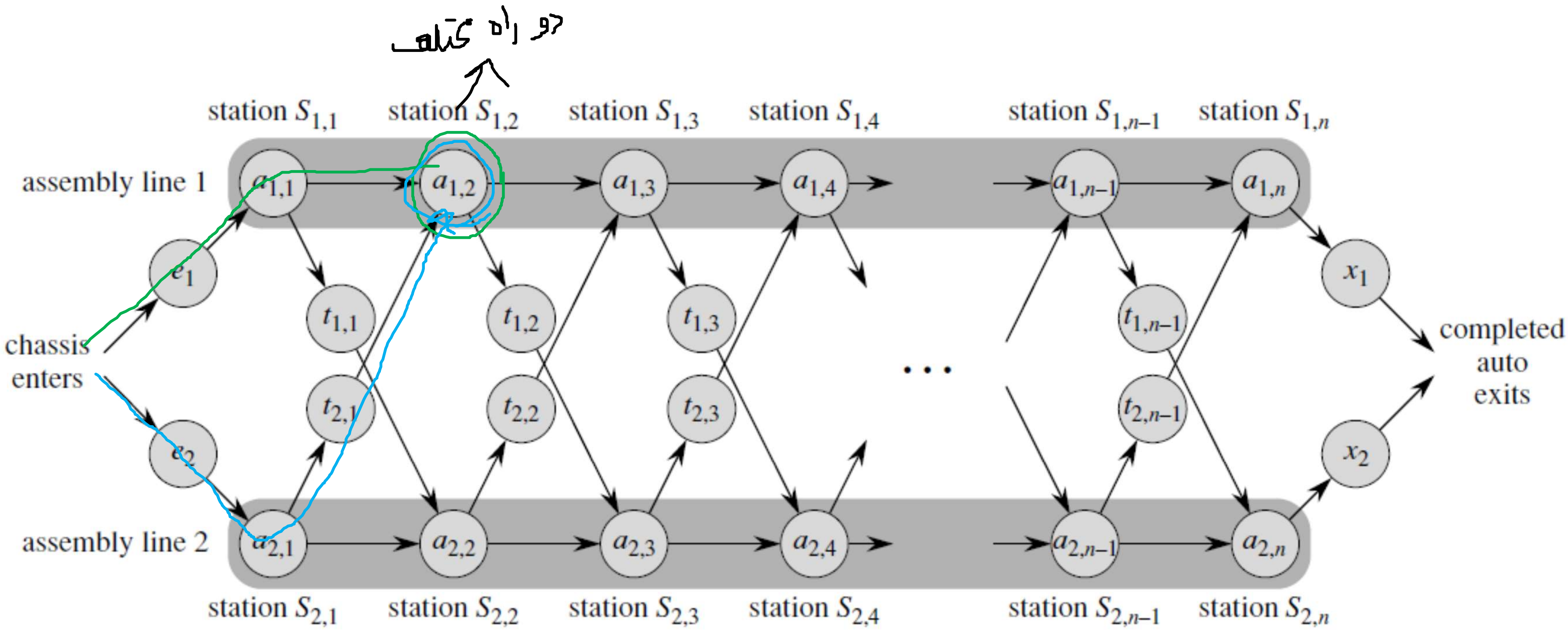
Assembly-line scheduling زمانبندی خط تولید



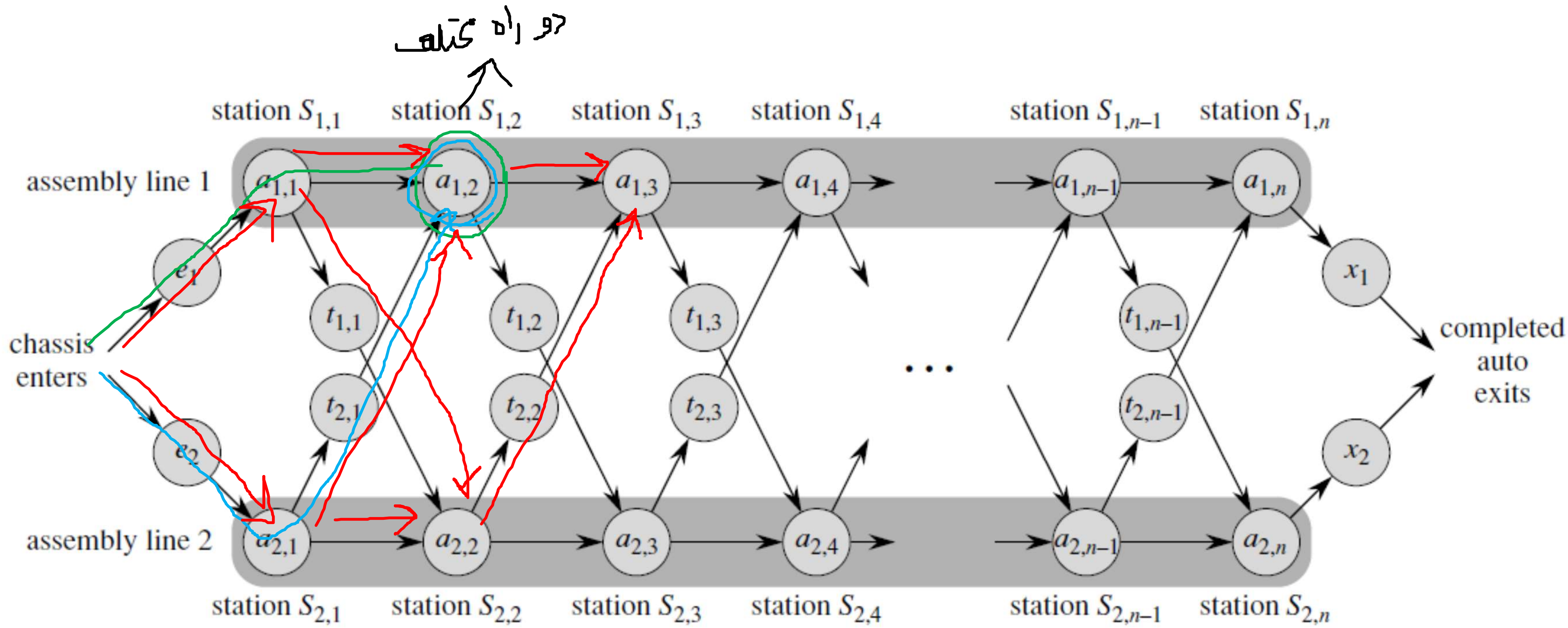
Assembly-line scheduling زمانبندی خط تولید



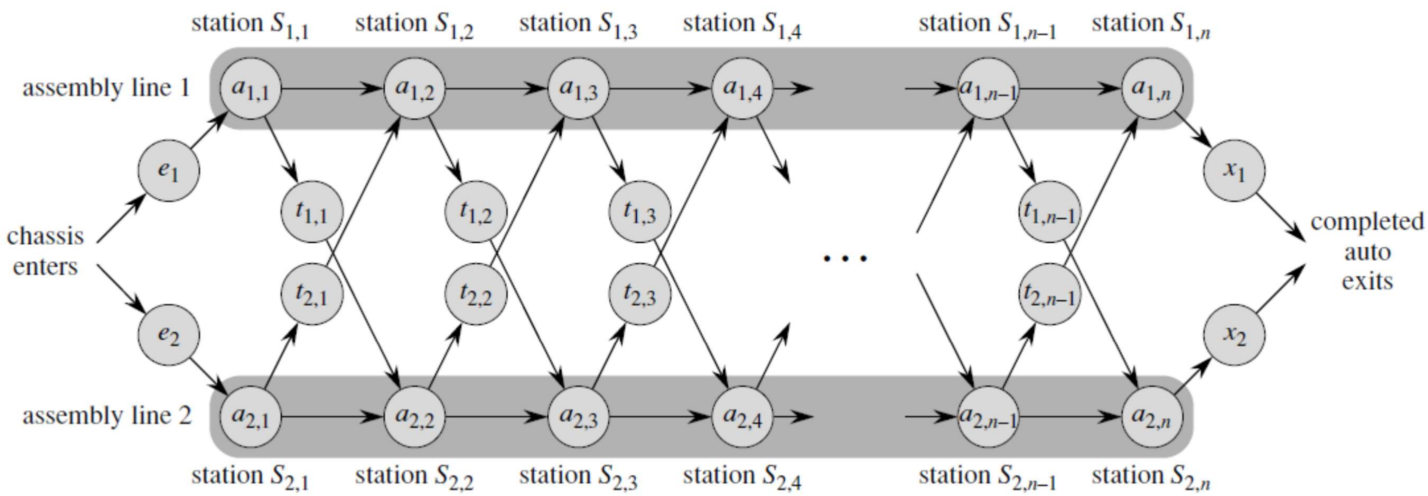
Assembly-line scheduling زمانبندی خط تولید



Assembly-line scheduling زمانبندی خط تولید



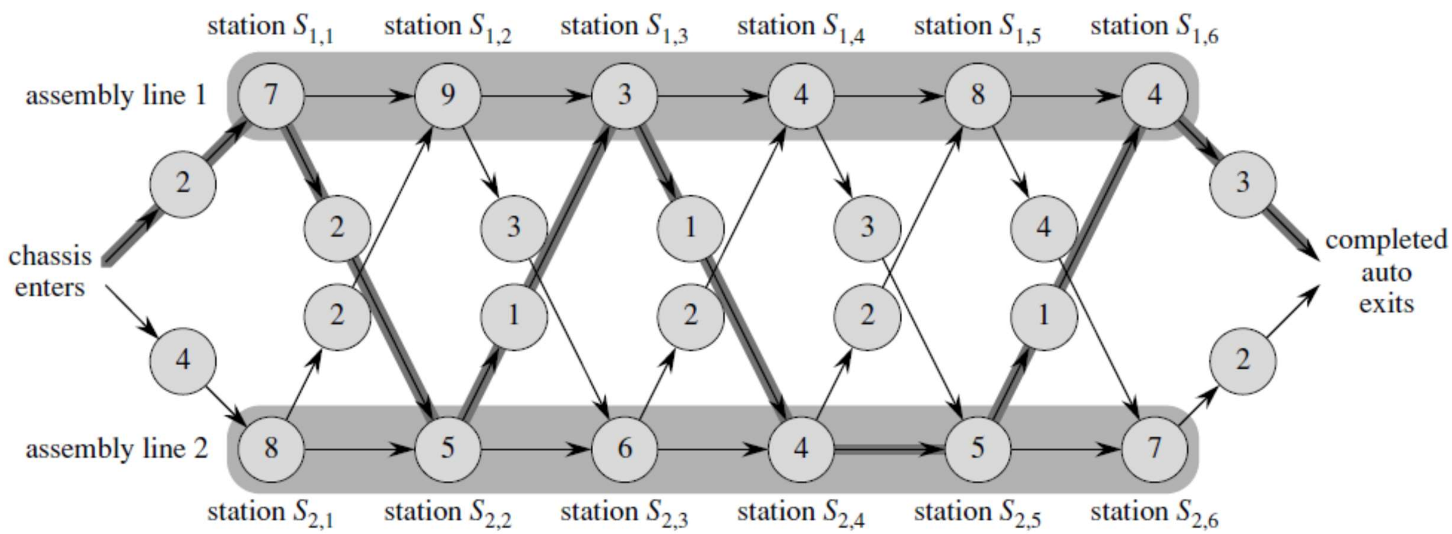
Assembly-line scheduling زمانبندی خط تولید



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

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

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)

$f_1[1] \leftarrow e_1 + a_{1,1}$
$f_2[1] \leftarrow e_2 + a_{2,1}$
for $j \leftarrow 2$ to n
    do if $f_1[j-1] + a_{1,j} \leq f_2[j-1] + t_{2,j-1} + a_{1,j}$
        then $f_1[j] \leftarrow f_1[j-1] + a_{1,j}$
            $l_1[j] \leftarrow 1$
        else $f_1[j] \leftarrow f_2[j-1] + t_{2,j-1} + a_{1,j}$
            $l_1[j] \leftarrow 2$
      if $f_2[j-1] + a_{2,j} \leq f_1[j-1] + t_{1,j-1} + a_{2,j}$
        then $f_2[j] \leftarrow f_2[j-1] + a_{2,j}$
            $l_2[j] \leftarrow 2$
        else $f_2[j] \leftarrow f_1[j-1] + t_{1,j-1} + a_{2,j}$
            $l_2[j] \leftarrow 1$
if $f_1[n] + x_1 \leq f_2[n] + x_2$
  then $f^* = f_1[n] + x_1$
      $l^* = 1$
  else $f^* = f_2[n] + x_2$
      $l^* = 2$

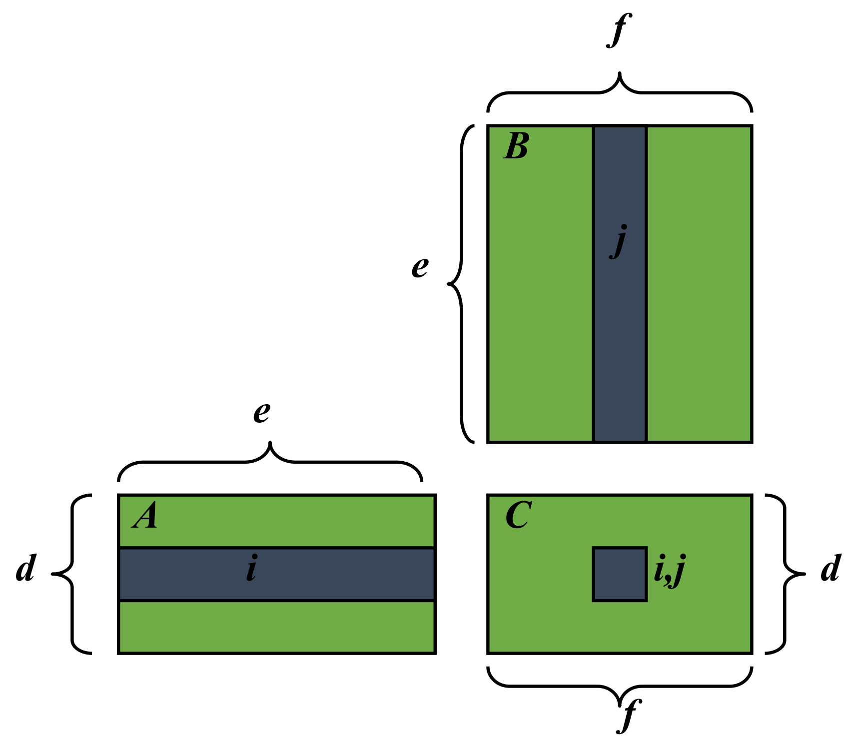
- الگوریتم چاپ مسیر بهینه برای پیمایش خط تولید را بنویسید؟ (از شبه کد استفاده کنید)

یاداوری - ضرب ماتریسی

میخواهیم ماتریس A را در ماتریس B ضرب کنیم.
ضرب و جمع دو عدد ساده را از O(1) در نظر می گیریم
تعداد ضرب های ساده مورد نیاز O(d.e.f) است

- $C = \boldsymbol{A}_{d\,.\,e} * \boldsymbol{B}_{e\,.\,f}$
- 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

مساله مشخص کردن ترتیب ضرب ها برای کمینه کردن تعداد ضرب های ساده مورد نیاز
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

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

Matrix-chain multiplication ضرب زنجیره ماتریس

- سه ماتریس نمونه برای ضرب در هم:

- A1 with dimensions 10 × 100
- A2 with dimensions 100 × 5
- A3 with dimensions 5 × 50

هزینه دو پرانتز گذاری متفاوت: (پرانتز گذاری بر تعداد ضرب های نهایی تاثیر می گذارد)

- ((A1 A2)A3) -> (10.100.5)+(10.5.50)=7500
- (A1(A2 A3)) -> (100.5.50)+(10.100.50)=75000

Matrix-chain multiplication ضرب زنجیره ماتریس

تعداد حالت های مختلف پرانتزگزاری:

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

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

Matrix-chain multiplication ضرب زنجیره ماتریس

- ساختار پرانتزگذاری بهینه:
  - برای محاسبه ضرب ماتریسی $A_i A_{i+1} \ldots A_j$
  ابتدا برای یک k نتیجه دو عبارت $A_i \ldots A_{k-1}$ و $A_k \ldots A_j$ محاسبه می شود

❖ $A_i \ldots A_{k-1}$ و $A_k \ldots A_j$ باید به صورت بهینه پرانتز گذاری شده باشند
❖چرا؟ (قاعده cut and paste)

Matrix-chain multiplication ضرب زنجیره ماتریس

- رابطه بازگشتی برای محاسبه هزینه بهینه ضرب ماتریسی $A_i \ A_{i+1} \cdots A_j$
  - در m[i , j] تعداد ضرب های مورد نیاز برای ضرب ماتریس های i تا j را ذخیره می کنیم
  - ابعاد ماتریس i-اُم برابر با p_{i-1} و p_i است

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

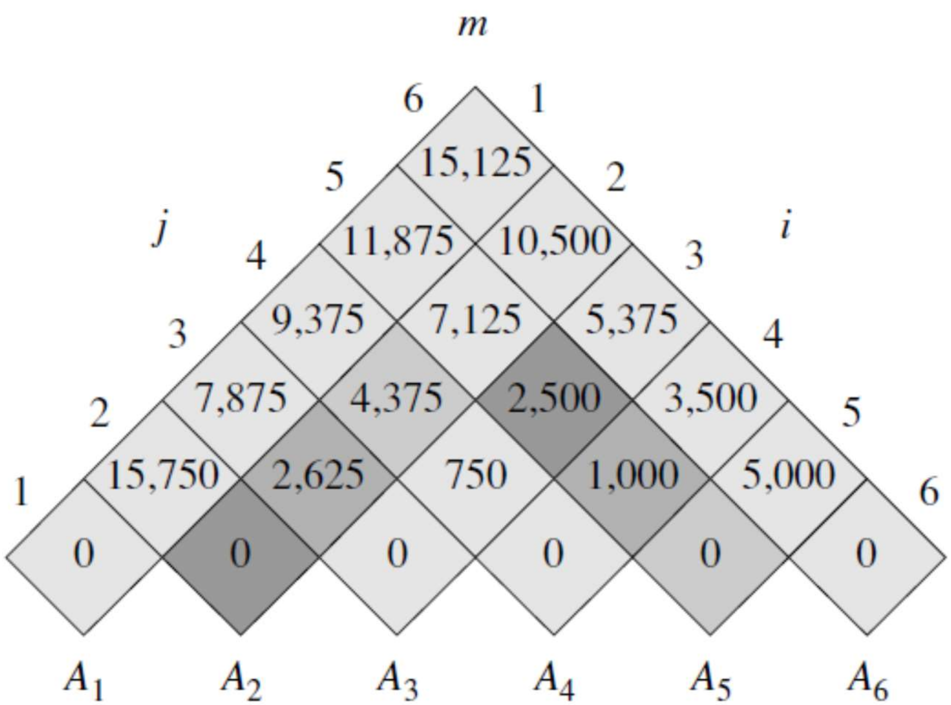
$$m[i, j] = \begin{cases} 0 & \text{if } i = j \,, \\ \min\limits_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1} p_k p_j\} & \text{if } i < j \,. \end{cases}$$

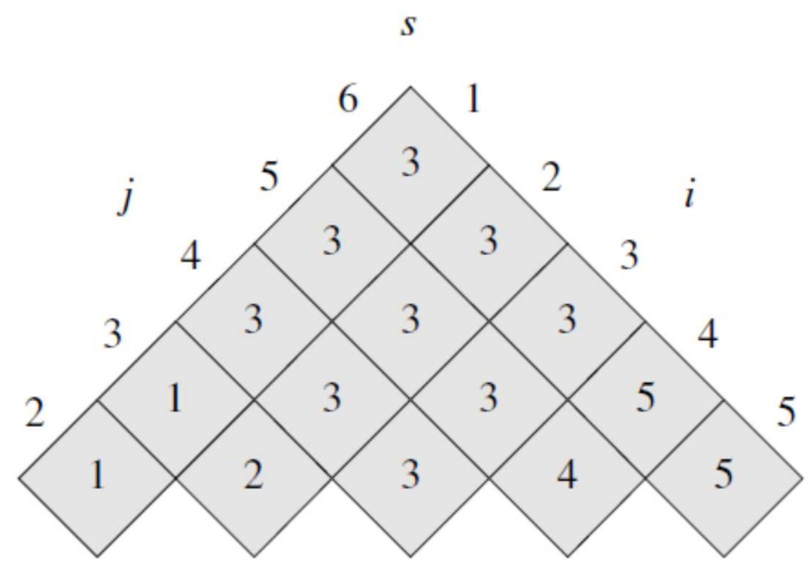
Matrix-chain multiplication ضرب زنجیره ماتریس
محاسبه مقدار بهینه

```
MATRIX-CHAIN-ORDER(p)
n ← length[p] − 1
for i ← 1 to n
    do m[i, i] ← 0
for l ← 2 to n                    ▷ l is the chain length.
    do for i ← 1 to n − l + 1
           do j ← i + l − 1
              m[i, j] ← ∞
              for k ← i to j − 1
                  do q ← m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
                     if q < m[i, j]
                        then m[i, j] ← q
                             s[i, j] ← k
return m and s
```

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

Matrix-chain multiplication ضرب زنجیره ماتریس
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)
if i = j
    then print "A"_i
    else print "("
         PRINT-OPTIMAL-PARENS(s, i, s[i, j])
         PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
         print ")"
```

Matrix-chain multiplication ضرب زنجیره ماتریس
محاسبه مقدار بهینه

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

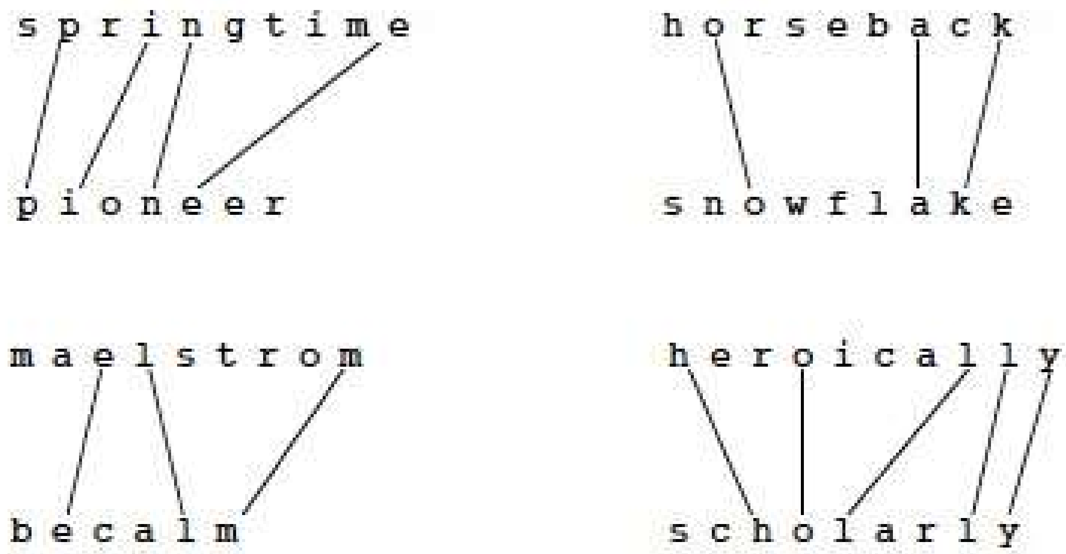
Longest common subsequence

- $S1$ = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA
- $S2$ = GTCGTTCGGAATGCCGTTGCTCTGTAAA

GTCGTCGGAAGCCGGCCGAA

ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

GTCGTTCGGAATGCCGTTGCTCTGTAAA


springtime
pioneer
horseback
snowflake
maelstrom
becalm
heroically
scholarly

Longest common subsequence

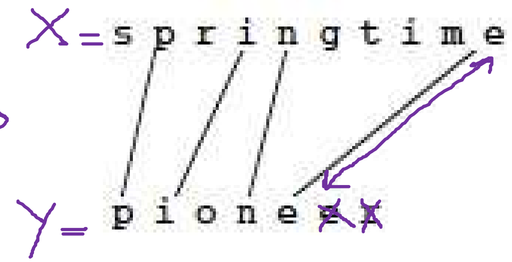
Characterizing a longest common subsequence

Theorem 15.1 (Optimal substructure of an LCS)
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ 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

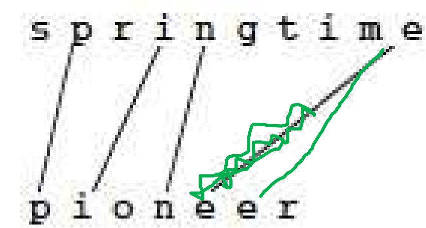
Characterizing a longest common subsequence



Theorem 15.1 (Optimal substructure of an LCS)

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

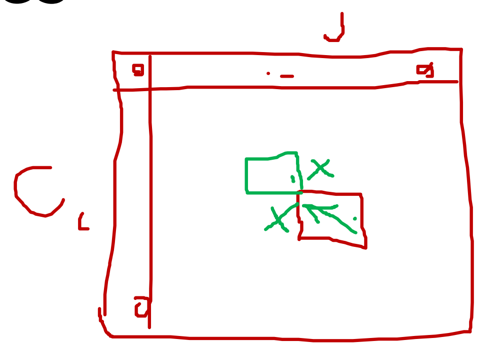


Longest common subsequence
A recursive solution

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

Longest common subsequence

Computing the length of an LCS




```
LCS-LENGTH(X, Y)
m ← length[X]
n ← length[Y]
for i ← 1 to m
        do c[i, 0] ← 0
for j ← 0 to n
        do c[0, j] ← 0
for i ← 1 to m
        do for j ← 1 to n
                do if x_i = y_j
                      then c[i, j] ← c[i − 1, j − 1] + 1
                           b[i, j] ← "↖"
                      else if c[i − 1, j] ≥ c[i, j − 1]
                              then c[i, j] ← c[i − 1, j]
                                   b[i, j] ← "↑"
                              else c[i, j] ← c[i, j − 1]
                                   b[i, j] ← "←"
return c and b
```

Longest common subsequence

Computing the length of an LCS

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

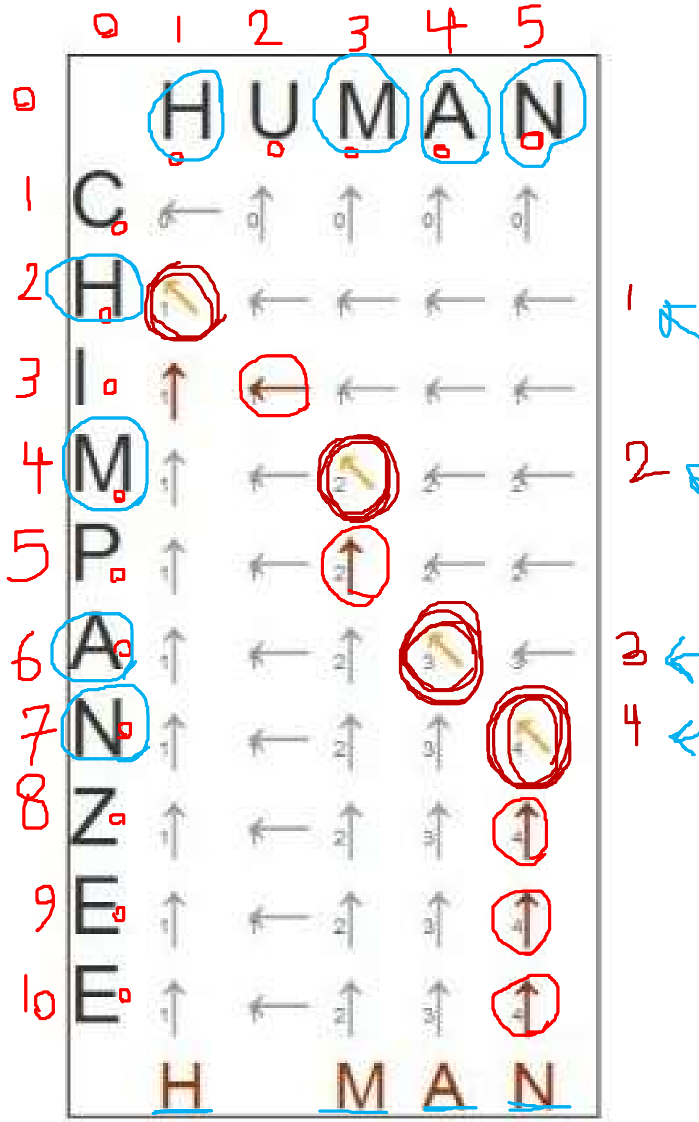
LCS-LENGTH(X, Y)

```
m ← length[X]
n ← length[Y]
for i ← 1 to m
    do c[i, 0] ← 0
for j ← 0 to n
    do c[0, j] ← 0
for i ← 1 to m
    do for j ← 1 to n
        do if x_i = y_j
            then c[i, j] ← c[i − 1, j − 1] + 1
                 b[i, j] ← "↖"
            else if c[i − 1, j] ≥ c[i, j − 1]
                then c[i, j] ← c[i − 1, j]
                     b[i, j] ← "↑"
                else c[i, j] ← c[i, j − 1]
                     b[i, j] ← "←"
return c and b
```

این قسمت حساب شده است

```
LCS-LENGTH(X, Y)
  m ← length[X]
  n ← length[Y]
  for i ← 1 to m
       do c[i, 0] ← 0
4 for j ← 0 to n
       do c[0, j] ← 0
3 for i ← 1 to m
       do for j ← 1 to n
              do if x_i = y_j
                    then c[i, j] ← c[i − 1, j − 1] + 1
                         b[i, j] ← "↖"
                    else if c[i − 1, j] ≥ c[i, j − 1]
                           then c[i, j] ← c[i − 1, j]
                                b[i, j] ← "↑"
                           else c[i, j] ← c[i, j − 1]
                                b[i, j] ← "←"
  return c and b
```

این قسمت حساب شده است




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

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.
  - در صورت نیاز این مبحث رفع اشکال می شود.

Think together ☺

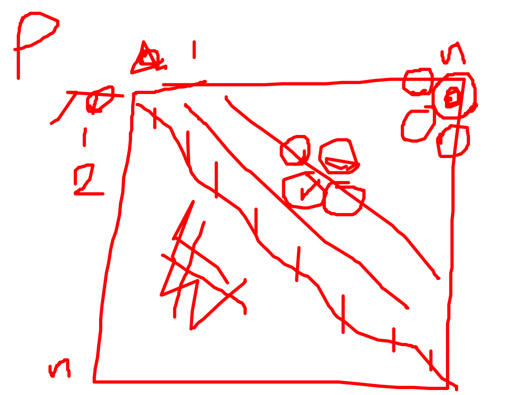
- پیدا کردن طولانی ترین زیر ترتیب از دو سر یکی (palindrome) ؟

Think together ☺

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

$x_1 \quad x_2 \quad x_3 \quad x_{n-1} \quad x_n$

$$P[i,j] \begin{cases} (1)\ i>j & 0 \\ (3)\ = & 2 + P[i+1, j-1] \\ (4)\ \neq & \max\{P[i+1,j],\ P[i,j-1]\} \\ (2)\ i=j & \end{cases}$$



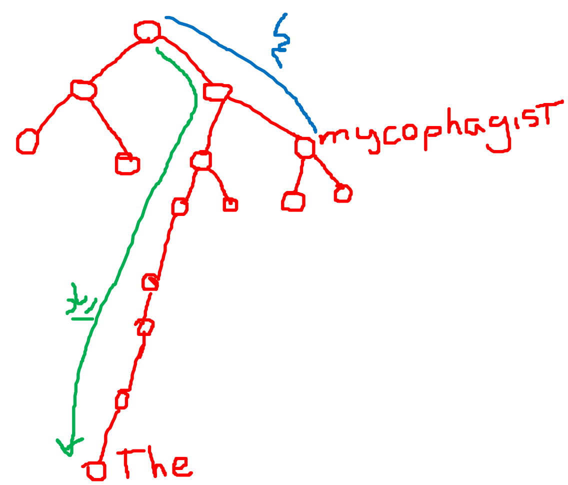
Look-up time
Optimal Binary Search Trees

- 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 $K = \langle k_1, k_2, \ldots, k_n \rangle$ of n distinct keys in sorted order

$$k_1 < k_2 < \cdots < k_n$$

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

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

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

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

Optimal Binary Search Trees

formal problem definition

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

$k_1 < k_2 < \cdots < k_n$

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

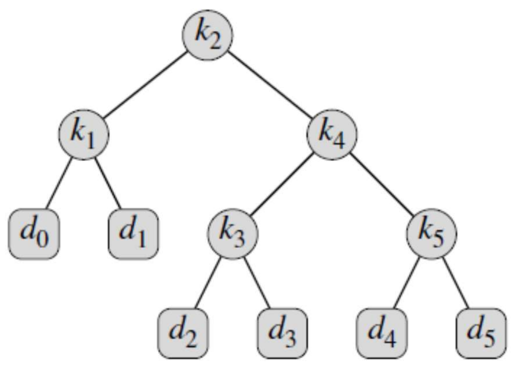
d_i represents all values between k_i and k_{i+1}

thF, They, ... The Thy

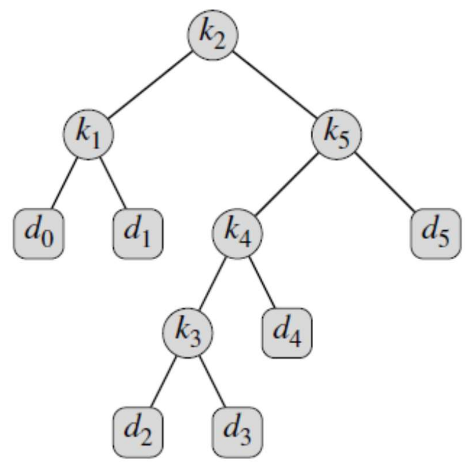
$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

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

Optimal Binary Search Trees example



expected search cost 2.80.

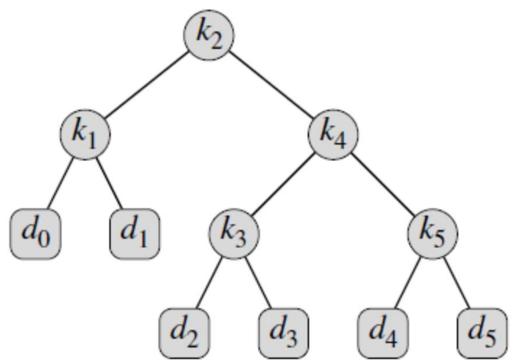


expected search cost 2.75.

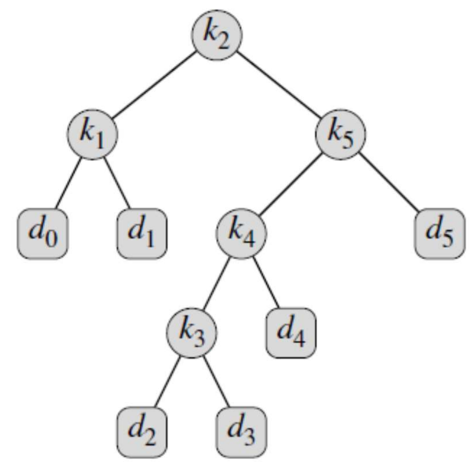
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees example

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



expected search cost 2.80.



expected search cost 2.75.

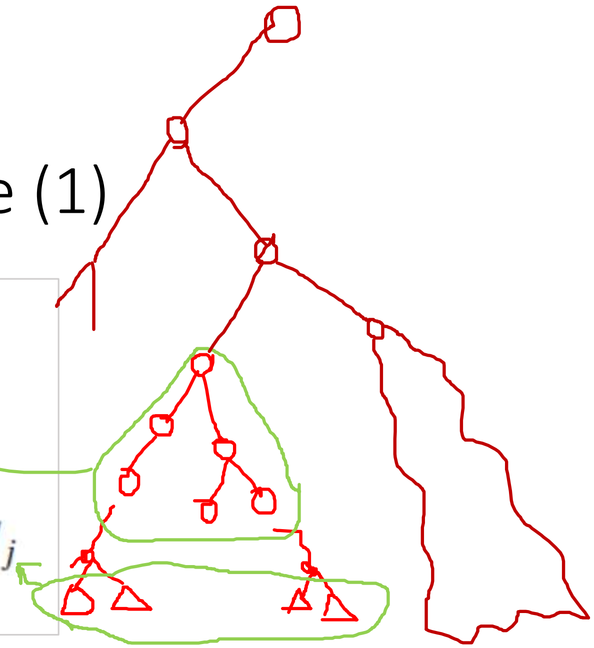
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal Binary Search Trees

The structure of an optimal binary search tree (1)

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

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



- 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)

$k_i, \ldots, k_j$, one of these keys, say k_r ($i \le r \le j$), will be the root of an optimal subtree

K_r

left subtree
$k_i, \ldots, k_{r-1}$ (and dummy keys $d_{i-1}, \ldots, d_{r-1}$)

right subtree
keys $k_{r+1}, \ldots, k_j$ (and dummy keys $d_r, \ldots, d_j$)

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

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

Optimal Binary Search Trees

A recursive solution (1)

- e[i , j]

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

Optimal Binary Search Trees

A recursive solution (2)

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

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

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

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

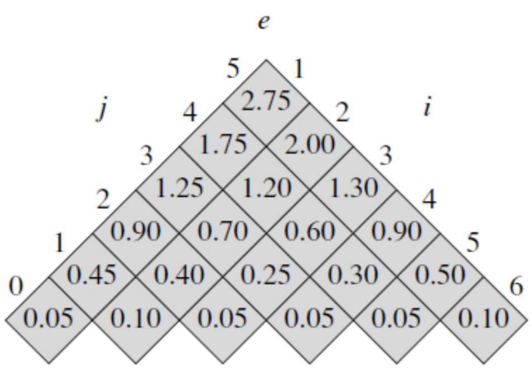
Optimal Binary Search Trees

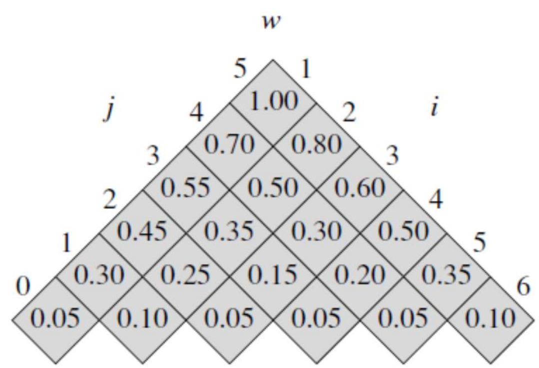
Computing the expected search cost of an optimal binary search tree

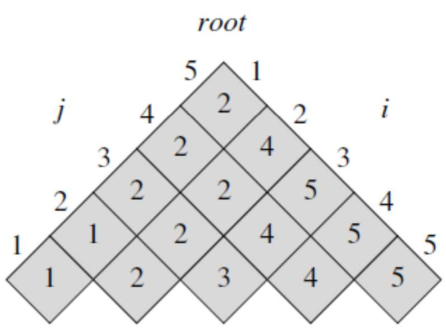
```
OPTIMAL-BST(p, q, n)
for i ← 1 to n + 1
      do e[i, i − 1] ← q_{i−1}
         w[i, i − 1] ← q_{i−1}
for l ← 1 to n
      do for i ← 1 to n − l + 1
               do j ← i + l − 1
                  e[i, j] ← ∞
                  w[i, j] ← w[i, j − 1] + p_j + q_j
                  for r ← i to j
                      do t ← e[i, r − 1] + e[r + 1, j] + w[i, j]
                         if t < e[i, j]
                           then e[i, j] ← t
                                root[i, j] ← r
return e and root
```

Optimal Binary Search Trees

example



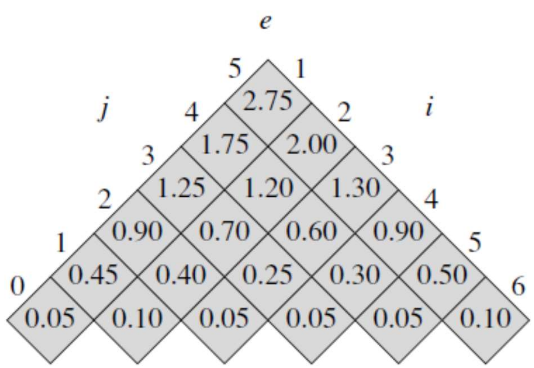


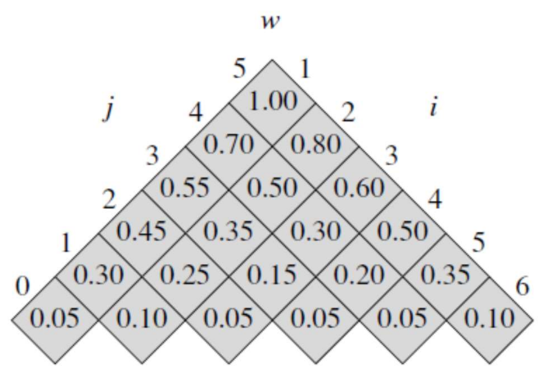


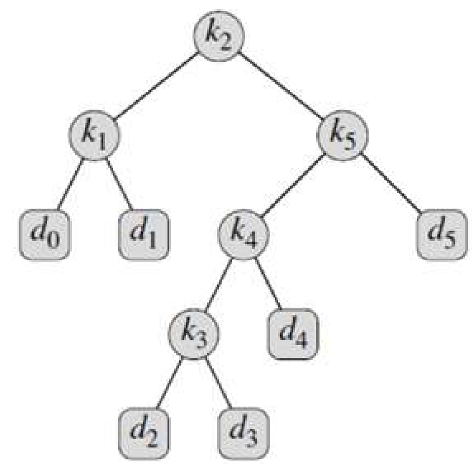
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

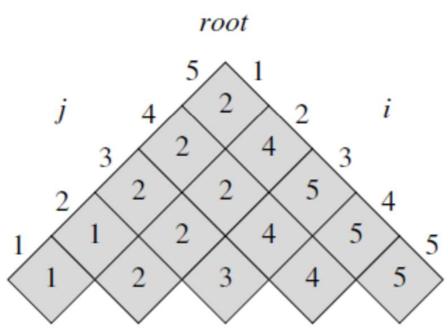
Optimal Binary Search Trees

example









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