

# طراحی الگوریتم‌ها – برنامه نویسی پویا

Introduction to Algorithm

مهدی جوانمردی

۱۴۰۱

# مرور جلسه قبل

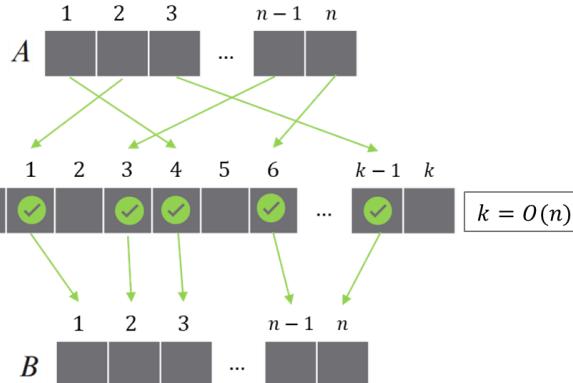
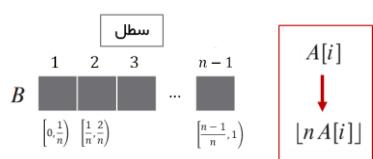
## BUCKET SORT مرتبسازی سطلي

$n$ -element array  $A$

$$0 \leq A[i] < 1$$

توزيع نرمال در بازه  $[0,1]$

$B[0..n - 1]$



## RADIX SORT مرتبسازی خطى

اطلاعات مزاد: اعداد دارای  $d$  رقم و هر رقم  $k$  مقدار متفاوت

329	720	720	329	RADIX-SORT( $A, d$ )
457	355	329	355	1 for $i = 1$ to $d$
657	436	436	436	2 use a stable sort to sort array $A$ on digit $i$
839	457	839	457	
436	657	355	657	
720	329	457	720	
355	839	657	839	

• زمان اجرا:  $\theta(d(n + k))$

• اگر  $d = O(1)$  و  $k = O(n)$  باشد زمان اجرای radix sort  $\theta(n)$  خواهد بود.

## BUCKET SORT تحليل زمانی

BUCKET-SORT( $A$ )

- let  $B[0..n - 1]$  be a new array
- $n = A.length$
- for  $i = 0$  to  $n - 1$ 
  - make  $B[i]$  an empty list
- for  $i = 1$  to  $n$ 
  - insert  $A[i]$  into list  $B[\lfloor nA[i] \rfloor]$
- for  $i = 0$  to  $n - 1$ 
  - sort list  $B[i]$  with insertion sort
- concatenate the lists  $B[0], B[1], \dots, B[n - 1]$  together in order

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

متغیر تصادفي نشانگر تعداد المانها در سطل  $B_i$

$$= \Theta(n) + \sum_{i=0}^{n-1} O\left(2 - \frac{1}{n}\right) = \Theta(n)$$

## RADIX SORT تحليل زمانی

• اگر  $n$  عدد  $b$  بیتی داشته باشیم، برای هر عدد مثبت دلخواه  $r \leq b$  خواهیم داشت:

زمان اجرای radix sort برای این اعداد  $\theta((b/r)(n + 2^r))$  خواهد بود

به شرطی که مرتبسازی پایدار استفاده شده  $\theta(n + k)$  باشد

برای اعداد  $n$  و  $b$  تعیین  $r \leq b$  بگونهای که زمان اجرای  $(b/r)(n + 2^r)$  را کمینه کند

حالت اول:  $r = b$  خواهیم داشت:  $\theta(n)$  برای  $r = b$  داشت:  $b < \lceil \lg n \rceil$

حالت دوم:  $r = \lceil \lg n \rceil$  خواهیم داشت:  $\theta(bn/\lg n)$  برای  $r = \lceil \lg n \rceil$  داشت:  $b \geq \lceil \lg n \rceil$

در صورتی که  $b = O(\lg n)$  باشد با انتخاب  $r = \lceil \lg n \rceil$  زمان اجری radix sort برابر  $O(n)$  خواهد بود.

# فصل ۱۵ کتاب

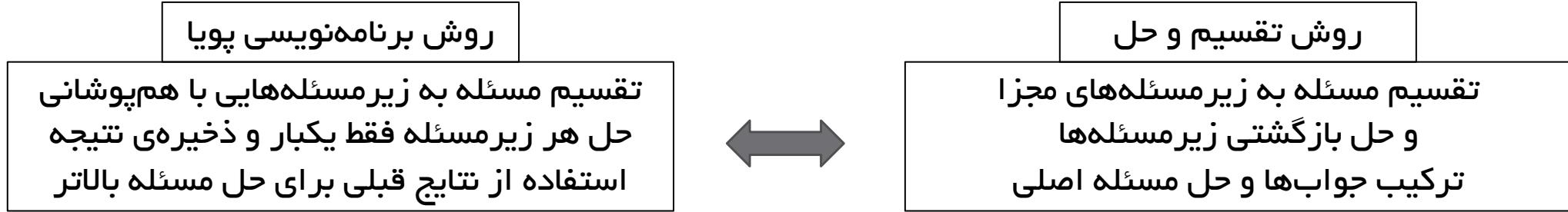
- برنامه‌نویسی پویا
- مسئله برش میله
- ضرب زنجیره‌ای ماتریس
- المان‌های برنامه‌نویسی پویا
- طولانی‌تر زیر رشته مشترک
- درخت دودویی جستجو بهینه

## 15 Dynamic Programming 359

- 15.1 Rod cutting 360
- 15.2 Matrix-chain multiplication 370
- 15.3 Elements of dynamic programming 378
- 15.4 Longest common subsequence 390
- 15.5 Optimal binary search trees 397

# برنامه نویسی پویا Dynamic Programming

- یک روش برای حل مسئله با ترکیب حل زیرمسئله‌ها
- مشابه روش‌های تقسیم و حل



- استفاده برنامه‌نویسی پویا در مسئله‌های بهینه سازی
- بهینه سازی: راه حل‌های متعددی برای حل مسئله دارد و ما جواب بهینه را میخواهیم
- هر جواب یک مقداری دارد و جواب بهینه ماکزیمم یا مینیمم مقدار را داشته باشد

# مراحل برنامه‌نویسی پویا

پایه‌های برنامه‌نویسی پویا



- .1 مشخص کردن ساختار یک جواب بهینه
- .2 تعیین مقدار برای یک جواب بهینه بصورت بازگشتی
- .3 محاسبه مقدار یک جواب بهینه، معمولاً بصورت پایین به بالا
- .4 جواب بهینه را از روی اطلاعات محاسبه شده بدست بیاور

اگر فقط مقدار بهینه را میخواهیم و نه خود جواب بهینه، میتوان از مرحله چهارم صرف نظر کرد  
برای بدست آوردن جواب بهینه میبایست برخی اطلاعات مزاد در مراحل قبلی نگهداری شوند

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

# مسئله برش میله

- برش یک میله بلند به قطعات کوچکتر به گونه ای که قیمت فروش آن بیشینیه شود!

• خروجی ها:

- بیشترین درآمد قابل حصول  $r_n$
- طول برش های مورد نیاز

• ورودی ها:

- طول میله اولیه  $n$
- قیمت فروش بر حسب طول  $p_i$
- قیمت برش رایگان

length $i$	1	2	3	4	5	6	7	8	9	10
price $p_i$	1	5	8	9	10	17	17	20	24	30

جدول قیمت فروش بر حسب طول

$$n = 4 \longrightarrow p_2 + p_2 = 5 + 5 = 10$$

مثال

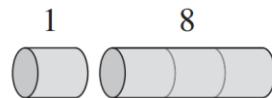
# مسئله برش میله

length $i$	1	2	3	4	5	6	7	8	9	10
price $p_i$	1	5	8	9	10	17	17	20	24	30

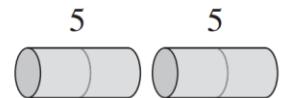
جدول قیمت فروش بر حسب طول



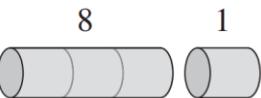
(a)



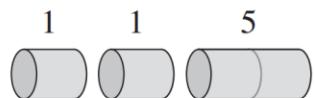
(b)



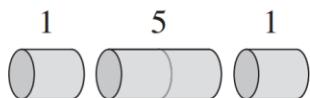
(c)



(d)



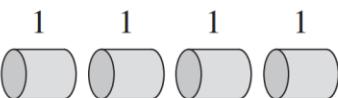
(e)



(f)



(g)



(h)

حالت‌های ممکن برای برش  
میله به طول ۴

فرمول

$$(a + b)^n = \sum_{k=0}^n \binom{n}{k} a^k b^{n-k}$$

partition function  $e^{\pi\sqrt{2n/3}} / 4n\sqrt{3}$

- تعداد حالات ممکن برای برش یک میله به طول  $n$ ؟

# جواب بهینه با مسئله برش میله

- اگر جواب بهینه میله را به  $k$  قسمت تقسیم کند، در این صورت برای برخی  $n$  خواهیم داشت:

$$n = i_1 + i_2 + \cdots + i_k$$

$$r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$$

بگونه‌ییکه در آمد حاصله بیشینه باشد:

length $i$	1	2	3	4	5	6	7	8	9	10
price $p_i$	1	5	8	9	10	17	17	20	24	30

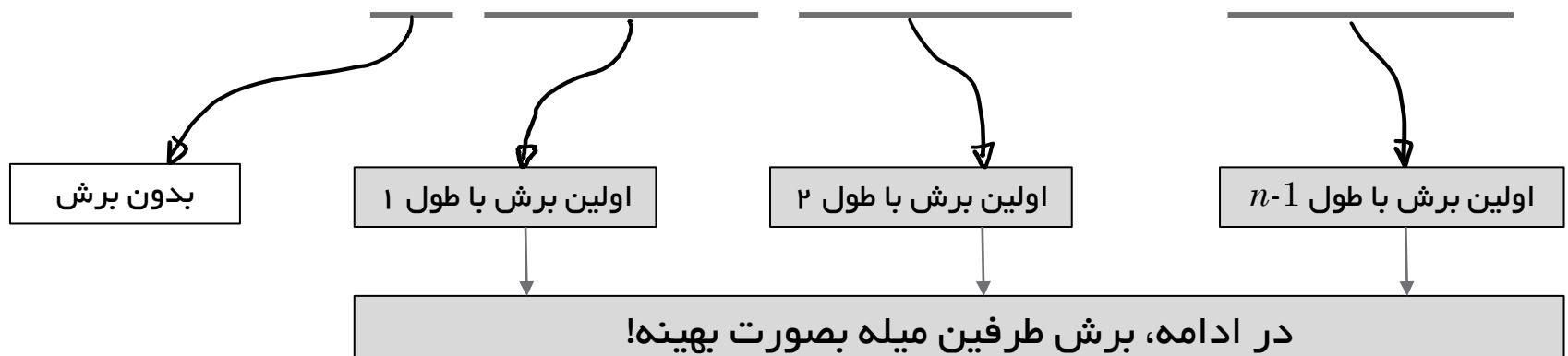
جدول قیمت فروش بر حسب طول

- |   |   |
|---|---|
| $r_1 = 1$ from solution $1 = 1$ (no cuts) , | $r_6 = 17$ from solution $6 = 6$ (no cuts) ,              |
| $r_2 = 5$ from solution $2 = 2$ (no cuts) , | $r_7 = 18$ from solution $7 = 1 + 6$ or $7 = 2 + 2 + 3$ , |
| $r_3 = 8$ from solution $3 = 3$ (no cuts) , | $r_8 = 22$ from solution $8 = 2 + 6$ ,                    |
| $r_4 = 10$ from solution $4 = 2 + 2$ ,      | $r_9 = 25$ from solution $9 = 3 + 6$ ,                    |
| $r_5 = 13$ from solution $5 = 2 + 3$ ,      | $r_{10} = 30$ from solution $10 = 10$ (no cuts) .         |

# به دست آوردن ساختار جواب بھینه

- بیشترین در آمد حاصل از برش یک میله به طول  $n$  ( $r_n$ ) با فرض داشتن بیشترین درآمد حاصل از برش میله‌های کوچکتر ( $r_1, r_2, \dots, r_{n-1}$ ):

$$r_n = \max (p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1)$$



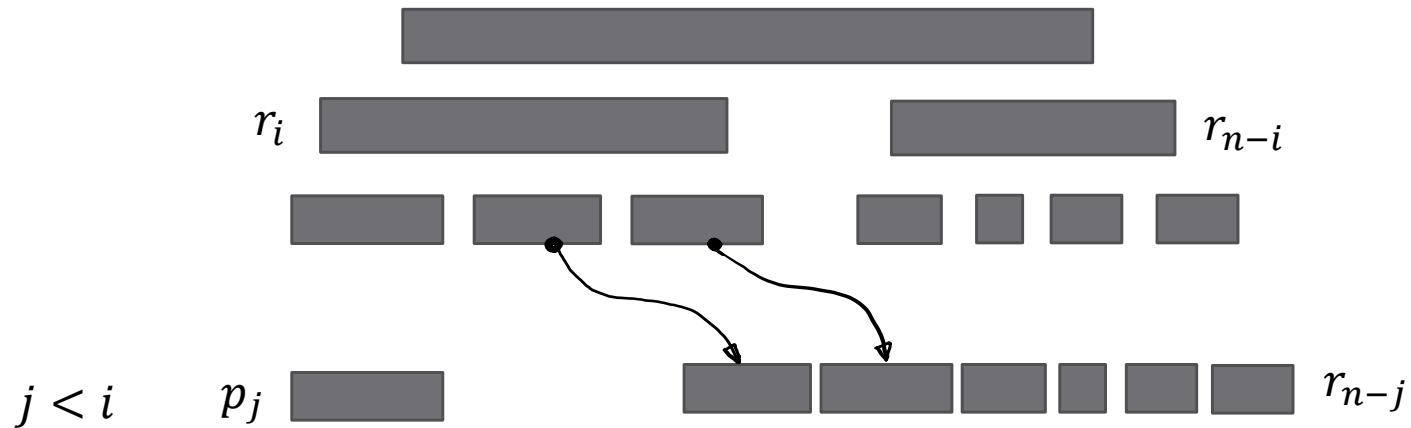
Optimal substructure  
زیرساختار بھینه:

جواب بھینه مسئله را  
می‌توان از جواب بھینه  
زیرمسئله‌ها به دست آورد

# ساختار کلی جواب بهینه

- بیشترین در آمد حاصل از برش یک میله به طول  $n$  ( $r_n$ ) با فرض داشتن بیشترین درآمد حاصل از برش میله‌های کوچکتر ( $r_1, r_2, \dots, r_{n-1}$ )

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1)$$



کلی حالت‌های تکراری خواهیم داشت.. برش فقط یک طرف در ادامه همه حالات را ایجاد میکند!

$$r_n = \max_{1 < i < n} (p_i + r_{n-i})$$

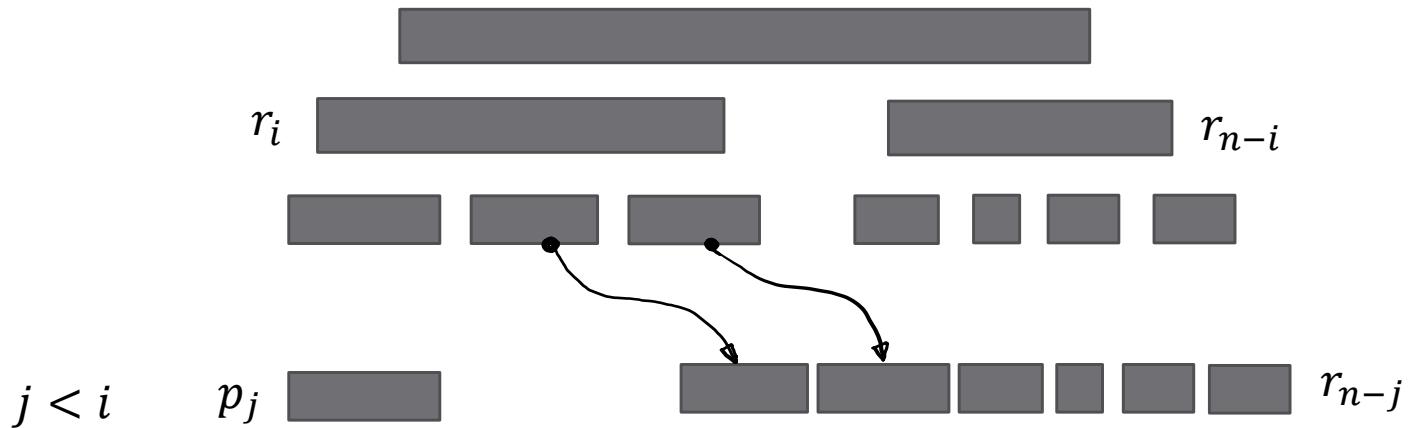
ساختار کلی جواب بهینه

کمی ساده‌تر

# ساختار کلی جواب بهینه

- بیشترین در آمد حاصل از برش یک میله به طول  $n$  ( $r_n$ ) با فرض داشتن بیشترین درآمد حاصل از برش میله‌های کوچکتر ( $r_1, r_2, \dots, r_{n-1}$ )

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1)$$



کلی حالت‌های تکراری خواهیم داشت.. برش فقط یک طرف در ادامه همه حالات را ایجاد میکند!

ساختار کلی جواب بهینه

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

کمی ساده‌تر

# حل بالا به پایین مسئله برش میله

ساختار کلی جواب بهینه

$$r_n = \max_{1 < i < n} (p_i + r_{n-i})$$

- حل بالا به پایین مسئله بصورت بازگشتی

Recursive top-down approach

CUT-ROD( $p, n$ )

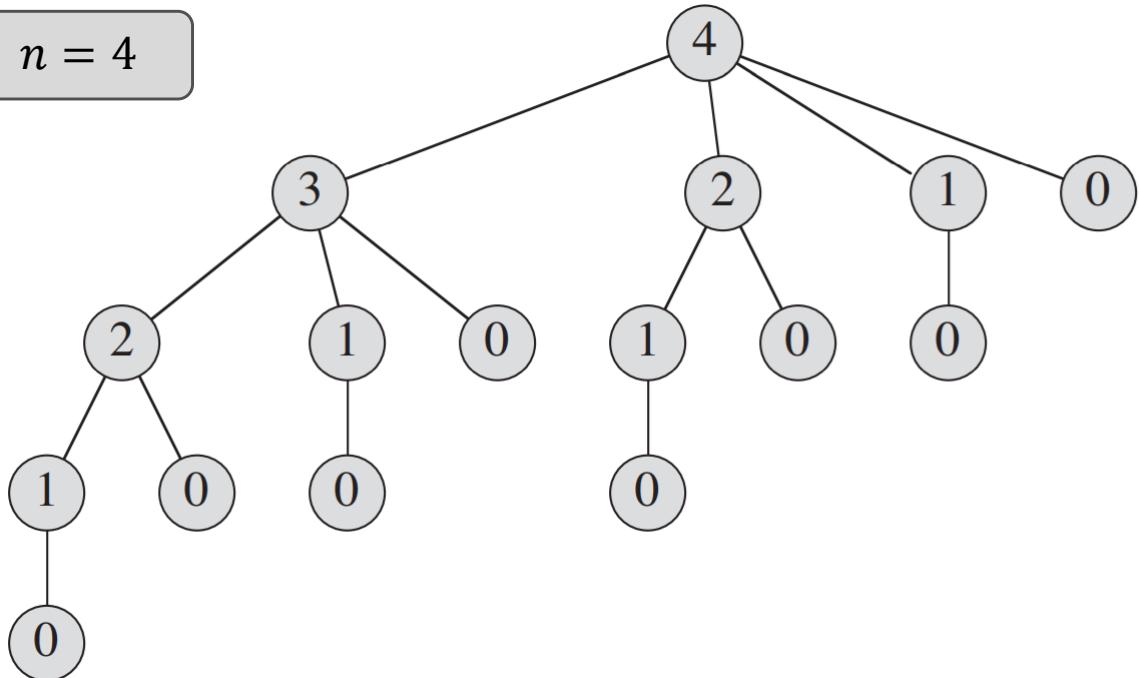
```

1 if  $n == 0$ 
2     return 0
3  $q = -\infty$ 
4 for  $i = 1$  to  $n$ 
5      $q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))$ 
6 return  $q$ 
```

بسیار زمانبر است! چرا؟؟

# تحلیل زمان اجرا حل بالا به پایین مسئله برش میله

$n = 4$



CUT-ROD( $p, n$ )

```

1 if  $n == 0$ 
2   return 0
3  $q = -\infty$ 
4 for  $i = 1$  to  $n$ 
5    $q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))$ 
6 return  $q$ 
  
```

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j)$$

$$T(n) = 2^n$$

# برنامه نویسی پویا و مسئله برش میله

- ایده اصلی: کاری کنیم که هر زیرمسئله فقط یکبار حل شود!
- اگر در ادامه به زیرمسئله تکراری نیاز داریم از جواب قبلی استفاده کنیم نه محاسبه مجدد نیاز به حافظه اضافی برای نگهداری جواب‌های محاسبه شده (time-memory trade-off)

## • روش‌های پیاده‌سازی

### ۱. حل بالا به پایین با حفظ کردن

ساختار کلی مشابه روش قبل

قبل از حل هر تابع بازگشتی چک می‌کند  
که آیا قبل از محاسبه شده یا نه

اگر محاسبه شده فقط استفاده می‌کند  
در غیر اینصورت محاسبه و ذخیره می‌کند

### ۲. حل پایین به بالا

حل مسئله از کوچک به بزرگ  
شروع از کوچکترین زیرمسئله

برای حل هر زیرمسئله مقادیر کوچکتر آن  
قبل از محاسبه و ذخیره شده

حل مسئله بزرگتر با مقادیر کوچکتر

# برنامه نویسی پویا و مسئله برش میله

MEMOIZED-CUT-ROD( $p, n$ )

```

1 let  $r[0..n]$  be a new array
2 for  $i = 0$  to  $n$ 
3    $r[i] = -\infty$ 
4 return MEMOIZED-CUT-ROD-AUX( $p, n, r$ )

```

MEMOIZED-CUT-ROD-AUX( $p, n, r$ )

```

1 if  $r[n] \geq 0$ 
2   return  $r[n]$ 
3 if  $n == 0$ 
4    $q = 0$ 
5 else  $q = -\infty$ 
6 for  $i = 1$  to  $n$ 
7    $q = \max(q, p[i] + \text{MEMOIZED-CUT-ROD-AUX}(p, n - i, r))$ 
8  $r[n] = q$ 
9 return  $q$ 

```

۱. حل بالا به پایین با حفظ کردن

ساختار کلی مشابه روش قبل

قبل از حل هر تابع بازگشتی چک میکند  
که آیا قبل از محاسبه شده یا نه

اگر محاسبه شده فقط استفاده میکند  
در غیر اینصورت محاسبه و ذخیره میکند

# برنامه نویسی پویا و مسئله برش میله

BOTTOM-UP-CUT-ROD( $p, n$ )

```

1 let  $r[0..n]$  be a new array
2  $r[0] = 0$ 
3 for  $j = 1$  to  $n$ 
4      $q = -\infty$ 
5     for  $i = 1$  to  $j$ 
6          $q = \max(q, p[i] + r[j - i])$ 
7      $r[j] = q$ 
8 return  $r[n]$ 
```

۲. حل پایین به بالا

حل مسئله از کوچک به بزرگ  
شروع از کوچکترین زیرمسئله

برای حل هر زیرمسئله مقادیر کوچکتر آن  
قبلا محاسبه و ذخیره شده

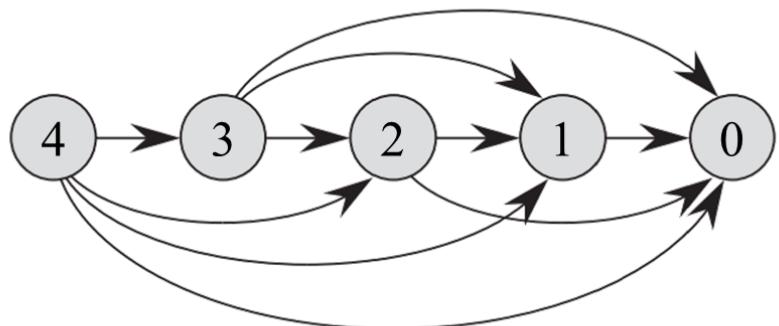
حل مسئله بزرگتر با مقادیر کوچکتر

- تحلیل زمانی هر دو روش: به دلیل حلقه تو در تو  $\Theta(n^2)$
- روش پایین به بالا دارای ثابت کوچکتر و در نتیجه سریع‌تر

# درخت زیرمسئله‌ها

- برای حل مسائل بصورت برنامه‌نویسی پویا باید رابطه بین زیرمسئله‌ها را بدانیم
- درخت زیرمسئله‌ها میتواند اطلاعات کافی در این زمینه به ما بدهد

$n = 4$   
درخت زیرمسئله‌ها برای 4



- یال جهتدار از گره  $x$  به گره  $y$  یعنی:  
حل بهینه  $x$  نیازمند جواب بهینه  $y$
- این گراف فشرده شده گراف حل بالا به پایین بازگشتی
- در روش پایین به بالا، گره  $y$  که از  $x$  به آن یالی وجود دارد  
باید قبل از حل گره  $x$  حتماً حل شده باشد

# بازسازی جواب بهینه

- تا الان: محاسبه بیشترین درآمد ممکن!
- اما روش بدست آوردن بیشترین درآمد؟ ← لیستی از طول قطعات

EXTENDED-BOTTOM-UP-CUT-ROD( $p, n$ )

```

1 let  $r[0..n]$  and  $s[0..n]$  be new arrays
2  $r[0] = 0$ 
3 for  $j = 1$  to  $n$ 
4      $q = -\infty$ 
5     for  $i = 1$  to  $j$ 
6         if  $q < p[i] + r[j-i]$ 
7              $q = p[i] + r[j-i]$ 
8              $s[j] = i$ 
9      $r[j] = q$ 
10 return  $r$  and  $s$ 
```

• نیازمند ذخیره‌سازی نه تنها **مقدار بهینه**  
بلکه **انتخاب بهینه**!

• **انتخاب بهینه:**

طول قطعه حاصل از اولین برش در حالت بهینه



# بازسازی جواب بهینه

- تا الان: محاسبه بیشترین درآمد ممکن!
- اما روش بدست آوردن بیشترین درآمد؟ ← لیستی از طول قطعات

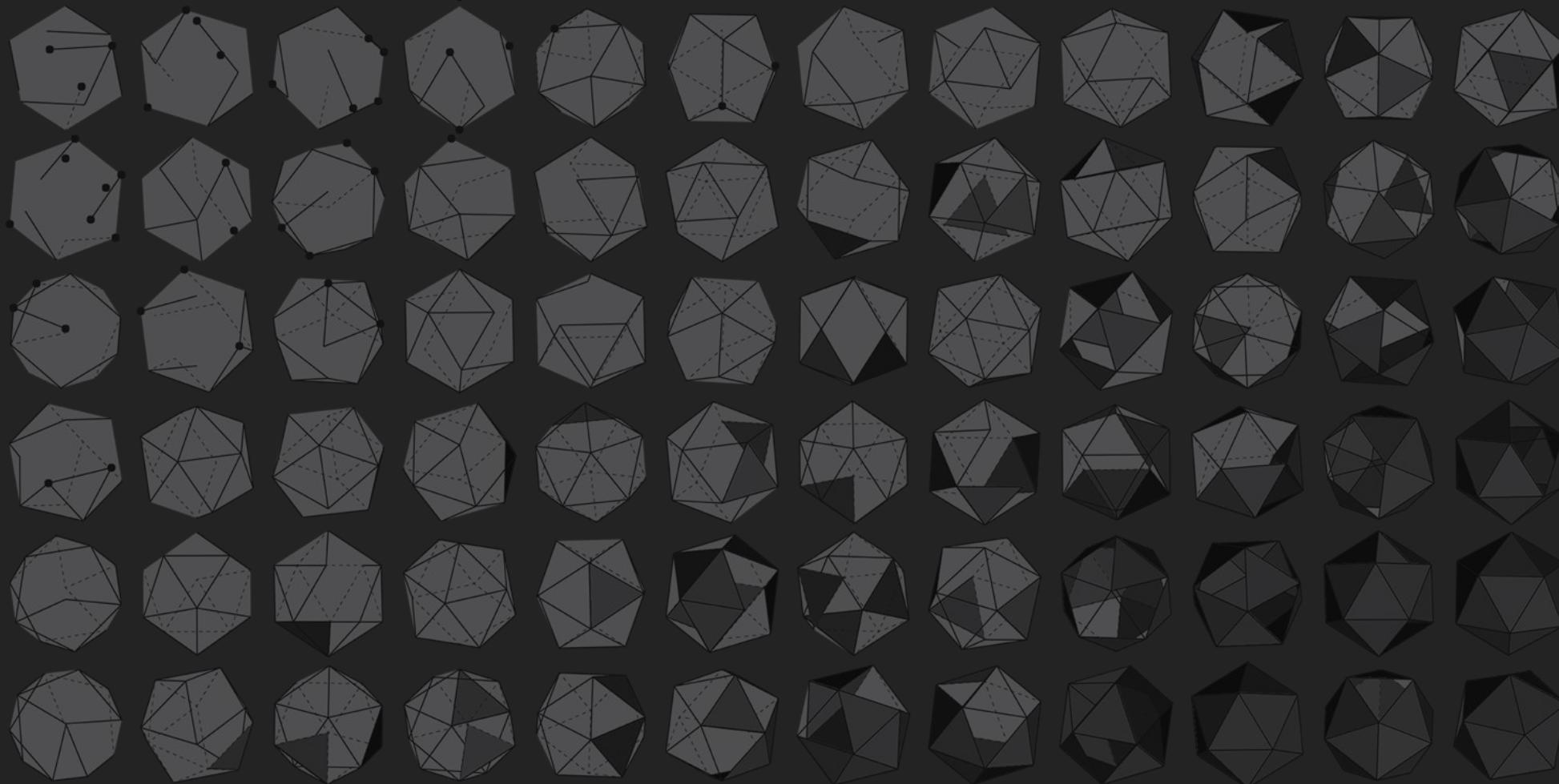
**PRINT-CUT-ROD-SOLUTION( $p, n$ )**

```

1   ( $r, s$ ) = EXTENDED-BOTTOM-UP-CUT-ROD( $p, n$ )
2   while  $n > 0$ 
3       print  $s[n]$ 
4        $n = n - s[n]$ 

```

$i$	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10



# ضرب زنجیره‌ای ماتریس‌ها

# مرور مطالب قبل

## Rod cutting مسئله برش میله

برش یک میله بلند به قطعات کوچکتر به گونه ای که قیمت فروش آن بیشینه شود!

### خروجی ها

- بیشترین درآمد قابل حصول  $r_n$
- طول میله اولیه  $n$
- طول برش های مورد نیاز

### ورودی ها

- قیمت فروش بر حسب طول  $p_i$
- قیمت برش رایگان

length $i$	1	2	3	4	5	6	7	8	9	10
price $p_i$	1	5	8	9	10	17	17	20	24	30

$$n = 4 \longrightarrow p_2 + p_2 = 5 + 5 = 10 \quad \text{مثال}$$

## حل پایین به بالا و پویا مسئله برش میله

BOTTOM-UP-CUT-ROD( $p, n$ )

```

1 let  $r[0..n]$  be a new array
2  $r[0] = 0$ 
3 for  $j = 1$  to  $n$ 
4    $q = -\infty$ 
5   for  $i = 1$  to  $j$ 
6      $q = \max(q, p[i] + r[j-i])$ 
7      $r[j] = q$ 
8 return  $r[n]$ 

```

حل مسئله از کوچک به بزرگ  
شروع از کوچکترین زیرمسئله

برای حل هر زیرمسئله مقادیر کوچکتر آن  
قبلما محاسبه و ذخیره شده

حل مسئله بزرگتر با مقادیر کوچکتر

## Dynamic programming

### روش برنامه نویسی پویا

تقسیم مسئله به زیرمسئله هایی با همپوشانی  
 حل هر زیرمسئله فقط یکبار و ذخیره نتیجه  
 استفاده از نتایج قبلی برای حل مسئله بالاتر



### روش تقسیم و حل

تقسیم مسئله به زیرمسئله هایی مجزا  
 و حل بازگشتی زیرمسئله ها  
 ترکیب جوابها و حل مسئله اصلی

### مراحل برنامه نویسی پویا

- مشخص کردن ساختار یک جواب بهینه
- تعیین مقدار برای یک جواب بهینه بصورت بازگشتی
- محاسبه مقدار یک جواب بهینه، معمولا بصورت پایین به بالا
- جواب بهینه را از روی اطلاعات محاسبه شده بدست بیاور

اگر فقط مقدار بهینه را میخواهیم و نه خود جواب بهینه، میتوان از مرحله چهارم صرف نظر کرد  
برای بدست آوردن جواب بهینه میبایست برخی اطلاعات مازاد در مراحل قبلی نگهداری شوند

# مروی بر ماتریس و ضرب زنجیره‌ای ماتریس‌ها

ماتریس و نحو نمایش آن:

$A =$

$$\begin{bmatrix} a[1,1] & a[1,2] & \cdots & a[1,m-1] & a[1,m] \\ a[2,1] & a[2,2] & \cdots & a[2,m-1] & a[2,m] \\ \vdots & \vdots & & \vdots & \vdots \\ a[n,1] & a[n,2] & \cdots & a[n,m-1] & a[n,m] \end{bmatrix}$$

ماتریس  $n \times m$  بنام  $A = [a[i,j]]$  یک آرایه دو بعدی بصورت با  $n$  سطر و  $m$  ستون است

ضرب ماتریس‌ها:

$$c[i,j] = \sum_{k=1}^q a[i,k]b[k,j]$$

ضرب ماتریسی  $q \times r$  و  $p \times q$  دو ماتریس  $C = AB$

که در آن  $1 \leq j \leq r$  و  $1 \leq i \leq p$

$$A = \begin{bmatrix} 1 & 8 & 9 \\ 7 & 6 & -1 \\ 5 & 5 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 8 \\ 7 & 6 \\ 5 & 5 \end{bmatrix} \quad C = AB = \begin{bmatrix} 102 & 101 \\ 44 & 87 \\ 70 & 100 \end{bmatrix}$$

• مسئله بعدی برنامه‌نویسی پویا ← ضرب زنجیره‌ای ماتریس‌ها

$A_1 A_2 \cdots A_n$

خروجی

$\langle A_1, A_2, \dots, A_n \rangle$

ورودی

# ضرب دو ماتریس در یکدیگر

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

$$c[i, j] = \sum_{k=1}^q a[i, k]b[k, j]$$

- ضرب ماتریسی  $C = AB$  دو ماتریس  $q \times r$  و  $p \times q$

MATRIX-MULTIPLY( $A, B$ )

```

1  if  $A.columns \neq B.rows$ 
2      error “incompatible dimensions”
3  else let  $C$  be a new  $A.rows \times B.columns$  matrix
4      for  $i = 1$  to  $A.rows$ 
5          for  $j = 1$  to  $B.columns$ 
6               $c_{ij} = 0$ 
7              for  $k = 1$  to  $A.columns$ 
8                   $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
9      return  $C$ 

```

تعداد ضرب مورد نیاز:  $r \times q \times p$

# ضرب زنجیرهای ماتریس‌ها

$$A_1 A_2 A_3 \cdots A_{s-1} A_s$$

$$= A_1(A_2(A_3 \cdots (A_{s-1} A_s)))$$

$$A_1 A_2 A_3 = (A_1 A_2) A_3 = A_1 (A_2 A_3)$$

- ضرب زنجیرهای ماتریس‌ها

- ویژگی شرکت پذیری ضرب ماتریس

- تعداد کل ضرب مورد نیاز بستگی به انتخاب پراتز گذاری دارد!

ضرب ماتریسی مورد نظر

$A_1 A_2 A_3 A_4$

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

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

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

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

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

انواع پراتز گذاری ممکن

# تعداد کل ضربها در ضرب زنجیره‌ای ماتریس‌ها

$$A_{i..j} = A_i A_{i+1} \cdots A_j$$

- در ضرب ماتریسی تعداد کل ضربها بستگی به انتخاب پراتز گذاری دارد!

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

- $A_{34} = A_3A_4$  , 250 mults, result is 5 by 1
- $A_{24} = A_2A_{34}$  , 500 mults, result is 100 by 1
- $A_{14} = A_1A_{24}$  , 1000 mults, result is 10 by 1
- Total is 1750**

3  $((A_1A_2)(A_3A_4))$

- $A_{12} = A_1A_2$  , 5000 mults, result is 10 by 5
- $A_{34} = A_3A_4$  , 250 mults, result is 5 by 1
- $A_{14} = A_{12}A_{34}$  , 50 mults, result is 10 by 1
- Total is 5300**

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

- $A_{12} = A_1A_2$  , 5000 mults, result is 10 by 5
- $A_{13} = A_{12}A_3$  , 2500 mults, result is 10 by 50
- $A_{14} = A_{13}A_4$  , 500 mults, results is 10 by 1
- Total is 8000**

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

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

3  $((A_1A_2)(A_3A_4))$

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

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

$A_1: 10 \times 100$

$A_2: 100 \times 5$

$A_3: 5 \times 50$

$A_4: 50 \times 1$

# ضرب زنجیره‌ای ماتریس‌ها بصورت بهینه!

صورت مسئله:

با فرض در اختیار داشتن زنجیره  $n$  تایی از ماتریس‌ها  $\langle A_1, A_2, \dots, A_n \rangle$   
که در آن ابعاد هر یک از  $A_i$  برابر باشد با  $p_{i-1} \times p_i$

انتخاب یک پراتزگذاری کامل برای ضرب زنجیره‌ای ماتریس‌های فوق بصورتیکه  
به کمترین تعداد ضرب نیاز داشته باشد

- در این مسئله در واقع با ضرب ماتریس‌ها کاری نداریم و هدف تعیین کم‌هزینه‌ترین ترتیب ممکن برای ضرب ماتریس‌ها است!
- طبعاً هزینه پیدا کردن روش بهینه باید خیلی کمتر از آورده حاصل از ضرب به روش بهینه باشد!

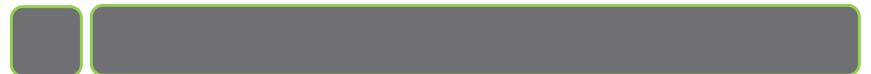
# تعداد پرانتز گذاری های ممکن

$P(n)$ : تعداد پرانتز گذاری ممکن برای  $n$  ماتریس



$A_1A_2 \dots A_n$

تعداد حالت های پرانتز گذاری



$A_1(A_2 \dots A_n)$

$P(1)P(n - 1)$



$(A_1A_2)(A_3 \dots A_n)$

$P(2)P(n - 2)$

:



$(A_1 \dots A_{n-2})(A_{n-1}A_n)$

$P(n - 2)P(2)$



$(A_1 \dots A_{n-1}) A_n$

$P(n - 1)P(1)$

$$27 \quad P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \end{cases}$$

$\longrightarrow \Omega(2^n)$

روش **X brute-force**

# برنامه‌نویسی پویا و حل ضرب زنجیره‌ای ماتریس‌ها

پایه‌های برنامه‌نویسی پویا



28

- .1 مشخص کردن ساختار یک جواب بهینه
  - .2 تعیین مقدار برای یک جواب بهینه بصورت بازگشتی
  - .3 محاسبه مقدار یک جواب بهینه، معمولاً بصورت پایین به بالا
  - .4 جواب بهینه را از روی اطلاعات محاسبه شده بدست بیاور
- 
- .1 تعیین زیرساختار پراتزگذاری بهینه
  - .2 راه حل بازگشتی برای پراتزگذاری بهینه
  - .3 محاسبه هزینه (تعداد ضرب) یک پراتزگذاری بهینه بصورت پایین به بالا
  - .4 جواب بهینه را از روی اطلاعات محاسبه شده بدست بیاور

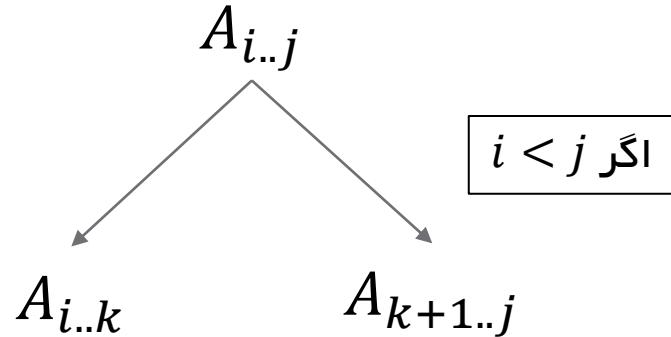
## ۱. تعیین زیرساختار پراتزگذاری بهینه



تعیین ساختار پراتزگذاری بهینه و استفاده از آن برای یافتن بهترین پاسخ

$$A_{i..j} = A_i A_{i+1} \cdots A_j$$

در صورتی که  $j < i$  باشد برای پرانتزگذاری  $A_{i..j} = A_i A_{i+1} \cdots A_j$  باید آن را بین دو ماتریس  $A_k$  و  $A_{k+1..j}$  به دو قسمت  $A_{i..k}$  و  $A_{k+1..j}$  تقسیم کنیم و خواهیم داشت  $i \leq k < j$ .



هزینه محاسبه  $A_{i..}$  با این روش پراتزگذاری برابر است با

- هزینه محاسبه ماتریس  $A_{i..k}$
  - هزینه محاسبه ماتریس  $A_{k+1..j}$
  - هزینه ضرب این دو

ساختار بهینه: اگر پرانتزگذاری بهینه  $A_{i..j}$  آن را به  $A_{k+1..j}$  و  $A_{i..k}$  تقسیم کرده باشد، می‌بایست پرانتزگذاری  $A_{k+1..j}$  و  $A_{i..k}$  نیز به خودی خود بهینه باشند

## ۲. راه حل بازگشتی برای پراتزگذاری بهینه

- استفاده از زیرساختار بهینه برای تعریف راه حل بازگشتی برای پراتزگذاری
- $A_{i..j} = A_i A_{i+1} \cdots A_j$ : کمترین تعداد ضرب مورد نیاز برای محاسبه  $m[i, j]$

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

- $m[i, j]$  معرف هزینه بهینه است اما انتخاب بهینه را به ما نمی‌دهد
- $s[i, j]$ : عدد  $k$  انتخاب شده که به موجب آن هزینه بهینه می‌شود
- می‌توان برای فرمول فوق یک برنامه بازگشتی نوشت

زمان اجرای نمایی



## ۳. محاسبه هزینه یک پراتزگذاری بهینه

- تعداد زیرمسئلهای  $m[i, j]$  محدود است: با توجه به  $1 \leq i \leq j \leq n$  داریم
- به جای راه حل بازگشتی، ارائه راه حل پایین به بالا

$$l = j - i + 1 : m[i, j]$$

برای حل زیرمسئلهایی به طول  $l$  صرفا نیاز به زیرمسئلهایی با طول  $l < i < j$  داریم

Bottom-up approach

شروع به حل از زیرمسئلهای بطول ۱، ۲، ۳، ...،  $n$

$$A_1 A_2 \dots A_n$$

$$A_i : p_{i-1} \times p_i \text{ for } i = 1, 2, \dots, n$$

$$p = \langle p_0, p_1, \dots, p_n \rangle$$

$$p.length = n + 1$$

$$m[1..n, 1..n]$$

جدول هزینه‌ها

$$s[1..n-1, 2..n]$$

جدول انتخاب‌ها

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

## ۳. محاسبه هزینه یک پرانتزگذاری بهینه

- تعداد زیرمسئلهای  $m[i, j]$  محدود است: با توجه به  $1 \leq i \leq j \leq n$  داریم
- به جای راه حل بازگشتی، ارائه راه حل پایین به بالا

MATRIX-CHAIN-ORDER( $p$ )

```

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

$$A_i : p_{i-1} \times p_i \text{ for } i = 1, 2, \dots, n$$

$$p = \langle p_0, p_1, \dots, p_n \rangle$$

$m[1..n, 1..n]$	جدول هزینه‌ها
-----------------	---------------

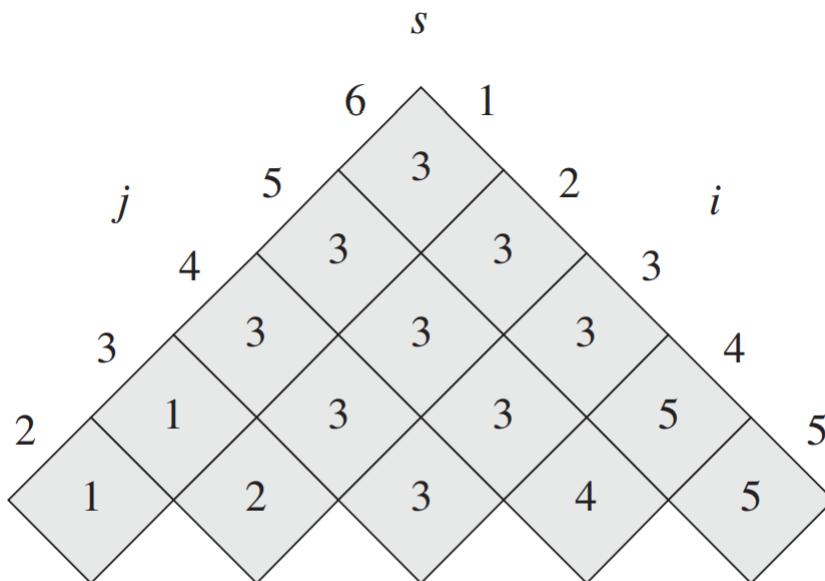
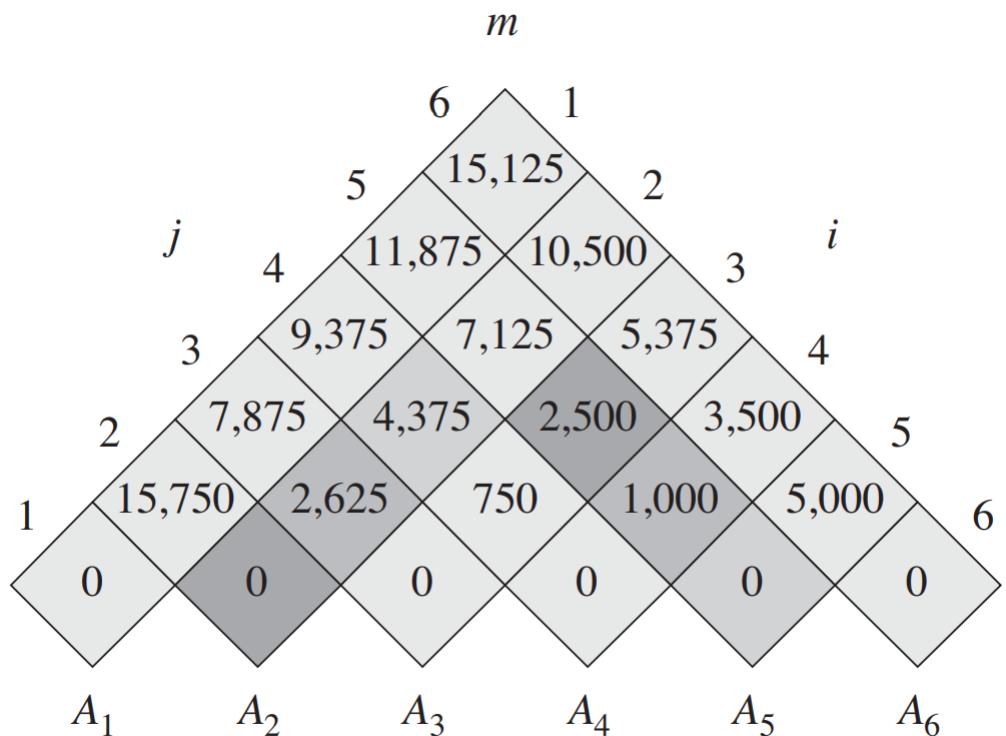
$s[1..n - 1, 2..n]$	جدول انتخاب‌ها
---------------------	----------------

$l = j - i + 1 : m[i, j]$
---------------------------

برای حل زیرمسئلهایی به طول $l$ صرفا نیاز به زیرمسئلهایی با طول $l < i < j$ داریم
---

## ۳. محاسبہ ہزینہ یک پراتر گذاری بھینہ

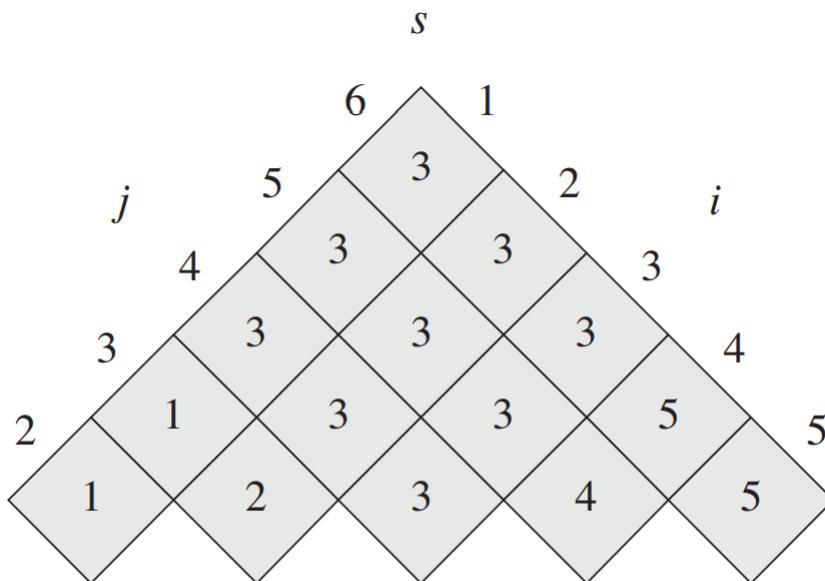
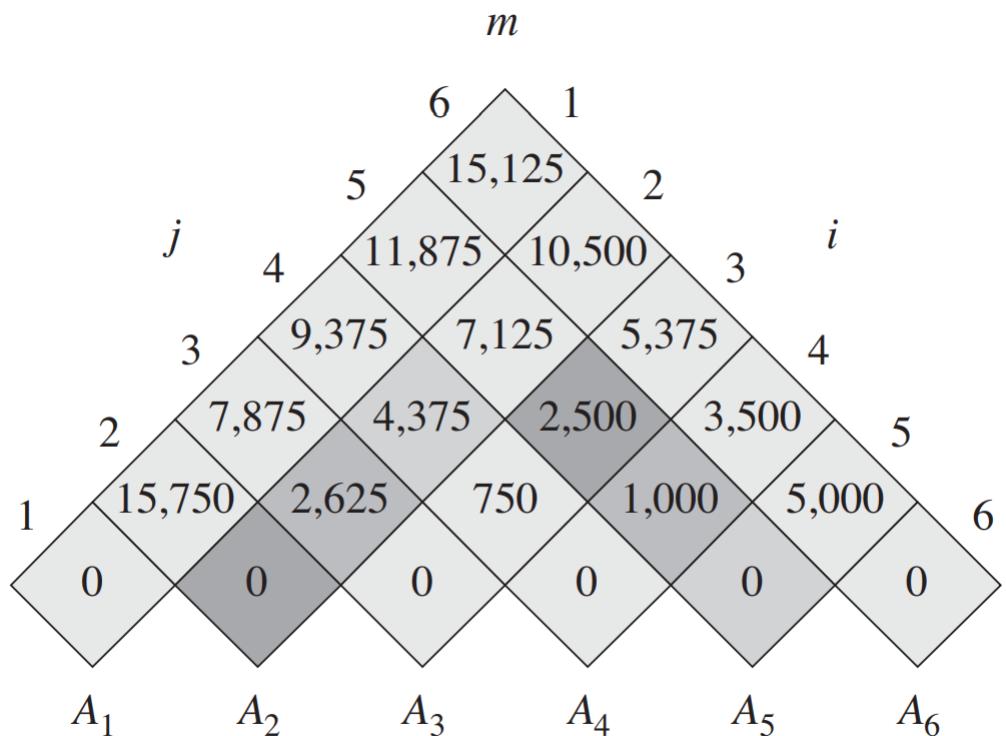
matrix	$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$
dimension	$30 \times 35$	$35 \times 15$	$15 \times 5$	$5 \times 10$	$10 \times 20$	$20 \times 25$



$$m[2, 5] = \min \begin{cases} m[2, 2] + m[3, 5] + p_1 p_2 p_5 &= 0 + 2500 + 35 \cdot 15 \cdot 20 = 13,000, \\ m[2, 3] + m[4, 5] + p_1 p_3 p_5 &= 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125, \\ m[2, 4] + m[5, 5] + p_1 p_4 p_5 &= 4375 + 0 + 35 \cdot 10 \cdot 20 = 11,375 \end{cases}$$

## ۳. محاسبہ ہزینہ یک پراتر گذاری بھینہ

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



$$m[2, 5] = \min \begin{cases} m[2, 2] + m[3, 5] + p_1 p_2 p_5 &= 0 + 2500 + 35 \cdot 15 \cdot 20 = 13,000, \\ m[2, 3] + m[4, 5] + p_1 p_3 p_5 &= 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125, \\ m[2, 4] + m[5, 5] + p_1 p_4 p_5 &= 4375 + 0 + 35 \cdot 10 \cdot 20 = 11,375 \end{cases}$$

## ۴. بازسازی جواب بهینه را از روی اطلاعات

- شبیه کد بازسازی جواب بهینه از روی جدول  $S$

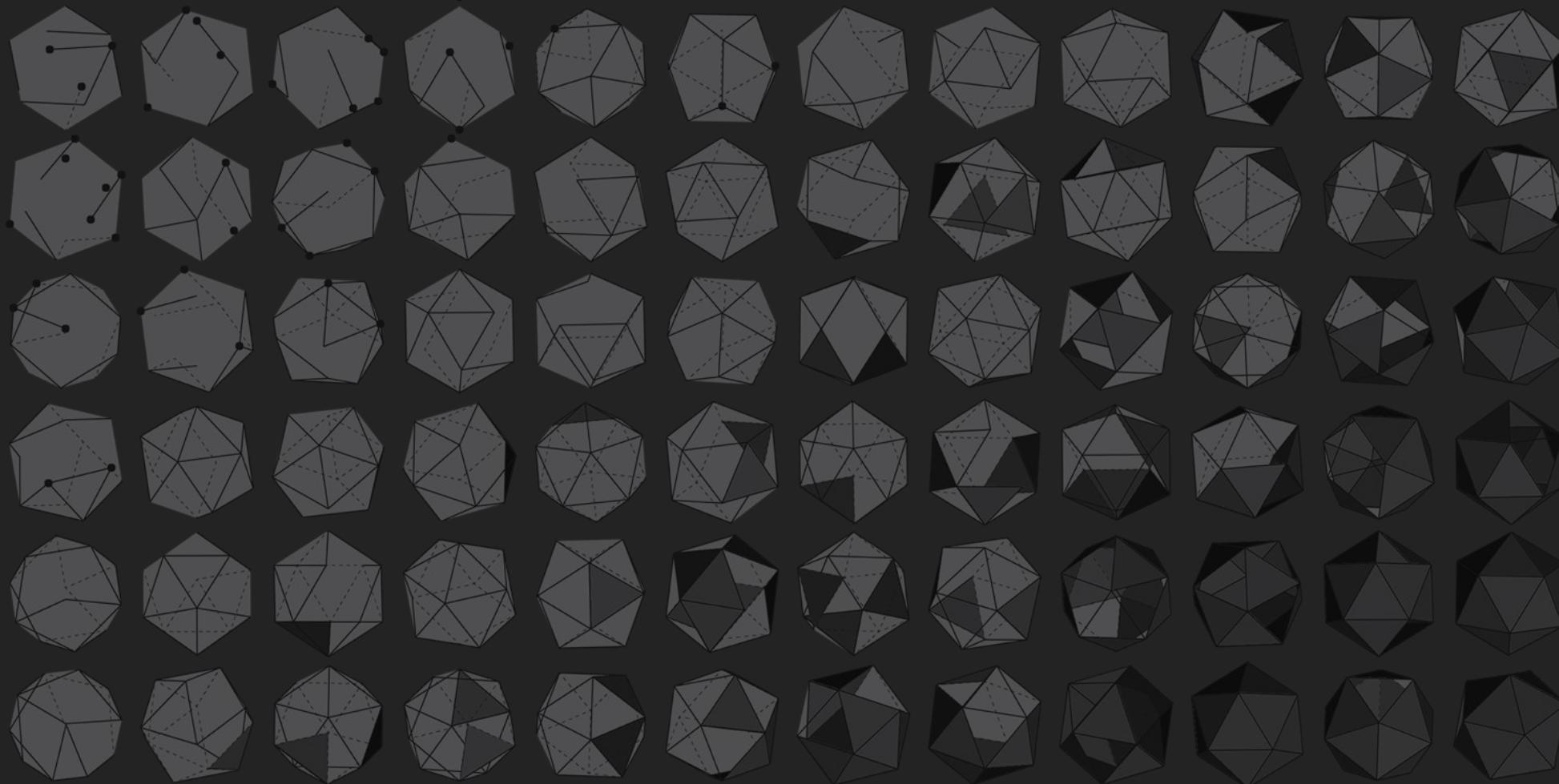
**PRINT-OPTIMAL-PARENS**( $s, i, j$ )

```

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

```

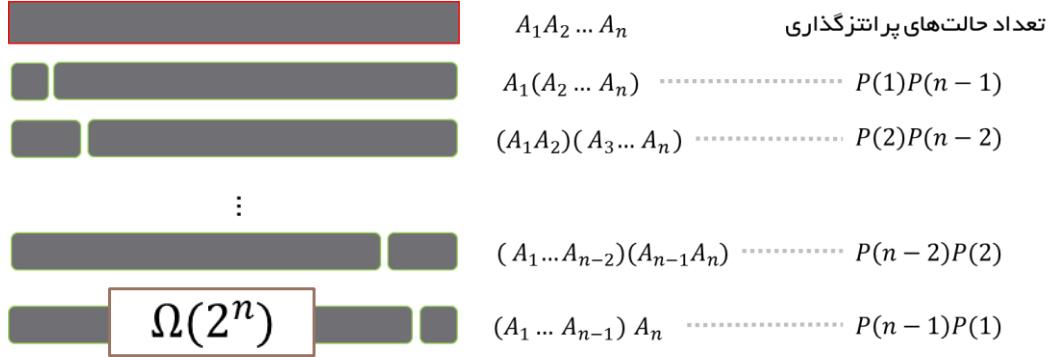
**PRINT-OPTIMAL-PARENS**( $s, 1, n$ )



# LCS طولانی‌ترین زیردنباله مشترک

# مرور مطالب قبل

## یافتن پرانتزگذاری با تعداد ضرب بهینه



## حل پایین به بالا و پویا پرانتزگذاری بهینه

$l = j - i + 1 : m[i, j]$  طول زیرمسئله

برای حل زیرمسئله‌هایی به طول  $l$  صرفاً نیاز به زیرمسئله‌هایی با طول  $l < i < j$  داریم

Bottom-up approach

شروع به حل از زیرمسئله‌های بطول ۲، ۳، ۴، ...،  $n$

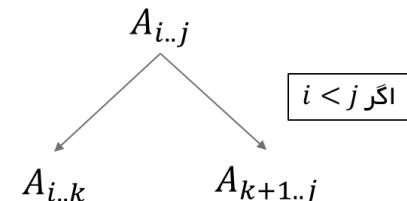
$m[1 \dots n, 1 \dots n]$

جدول هزینه‌ها

$s[1 \dots n - 1, 2 \dots n]$

جدول انتخاب‌ها

$$A_{i..j} = A_i A_{i+1} \dots A_j$$



- هزینه محاسبه ماتریس  $A_{i..k}$
- هزینه محاسبه ماتریس  $A_{k+1..j}$
- هزینه ضرب این دو

## ضرب ماتریسی

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

ضرب ماتریسی  $C = AB$  دو ماتریس  $q \times r$  و  $p \times q$

```

MATRIX-MULTIPLY( $A, B$ )
1 if  $A.columns \neq B.rows$ 
2 error "incompatible dimensions"
3 else let  $C$  be a new  $A.rows \times B.columns$  matrix
4 for  $i = 1$  to  $A.rows$ 
5   for  $j = 1$  to  $B.columns$ 
6      $c_{ij} = 0$ 
7     for  $k = 1$  to  $A.columns$ 
8        $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
9 return  $C$ 

```

تعداد ضرب مورد نیاز:  $p \times q \times r$

## ضرب زنجیره‌ای ماتریس‌ها و پرانتزگذاری

$$\begin{aligned}
(A_1(A_2(A_3 A_4))) &= A_1 A_2 A_3 \cdots A_{s-1} A_s \\
(A_1((A_2 A_3) A_4)) &= A_1(A_2(A_3 \cdots (A_{s-1} A_s))) \\
((A_1 A_2)(A_3 A_4)) &\\
((A_1(A_2 A_3)) A_4) &\\
(((A_1 A_2) A_3) A_4) &
\end{aligned}$$

انواع پرانتزگذاری ممکن

تعداد کل ضرب مورد نیاز بستگی به انتخاب پرانتزگذاری دارد!

# چه زمانی می‌توان از برنامه‌نویسی پویا استفاده کرد؟

- تا اینجای درس دو مثال از برنامه‌نویسی پویا دیدیم:
  - برش میله
  - ضرب ماتریسی
- سوال مهم و تکراری:

برای حل یک مسئله بهینه‌سازی به روش پویا آن مسئله باید چه ویژگی‌هایی داشته باشد؟

(optimum substructure) (زیرساختار بهینه)

جواب بهینه یک مسئله از جواب‌های بهینه زیرمسئله‌ها به دست آید

(overlapping subproblems) (زیرمسئله‌های با همپوشانی)

فضای جستجوی زیرمسئله‌ها باید کوچک باشد و روش بازگشتی بارها یک زیرمسئله را حل کند

# مراحل عمومی کشف زیر ساختار بهینه

۱) راه حل مسئله شامل یک انتخاب باشد و آن انتخاب یک یا چند زیرمسئله جدید ایجاد کند

در این مرحله با یک انتخاب مناسب تا می‌توانید زیرمسئله‌های ساده ایجاد کنید و اگر نشد آن را تعمیم دهید

۲) فرض می‌کنیم که راه حل بهینه مسئله را در اختیار داریم (روش آن مهم نیست)

۳) با در اختیار داشتن انتخاب مرحله اول تعیین می‌کنیم چه زیرمسئله‌هایی متنج می‌شود

الف) چه تعداد زیرمسئله ایجاد می‌شود

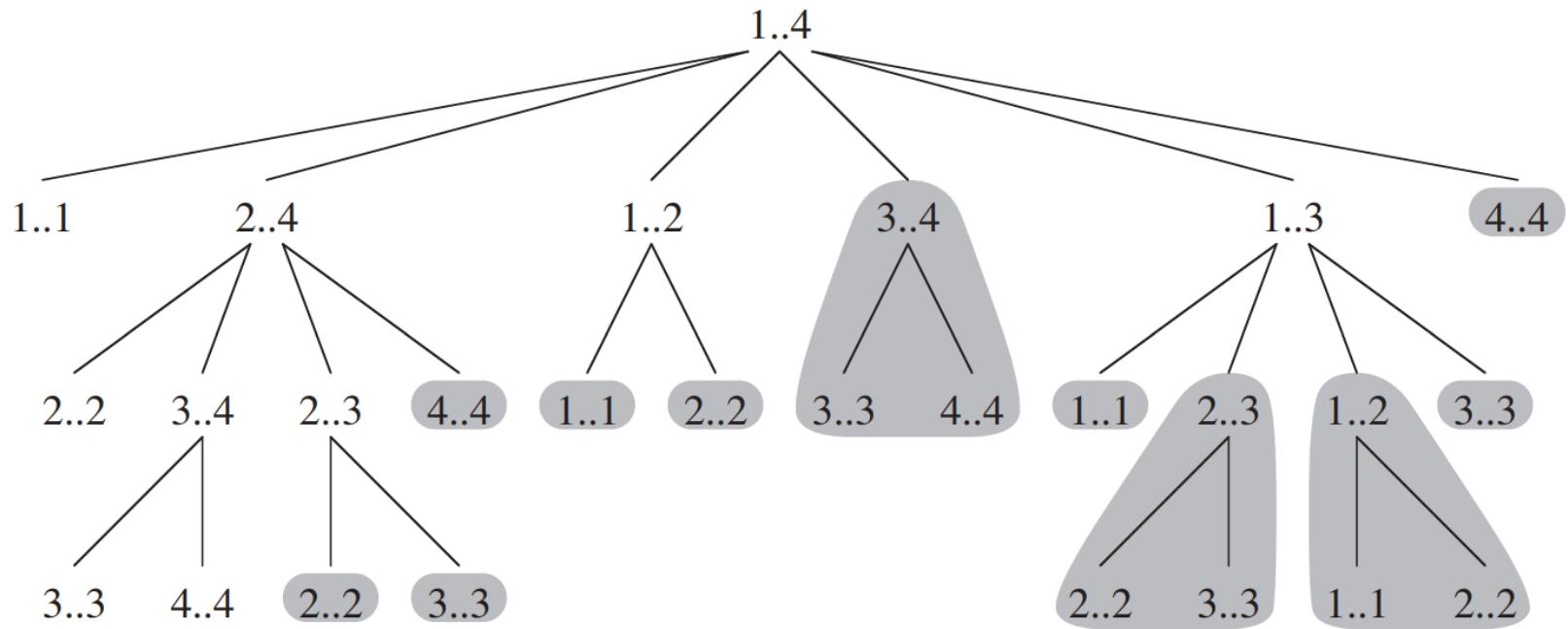
ب) چه تعداد انتخاب با باید امتحان کردن تا به جواب بهینه دست یافت؟

۴) نشان میدهیم برای دستیابی به راه حل بهینه میبایست جواب زیرمسئله‌ها نیز بهینه باشد

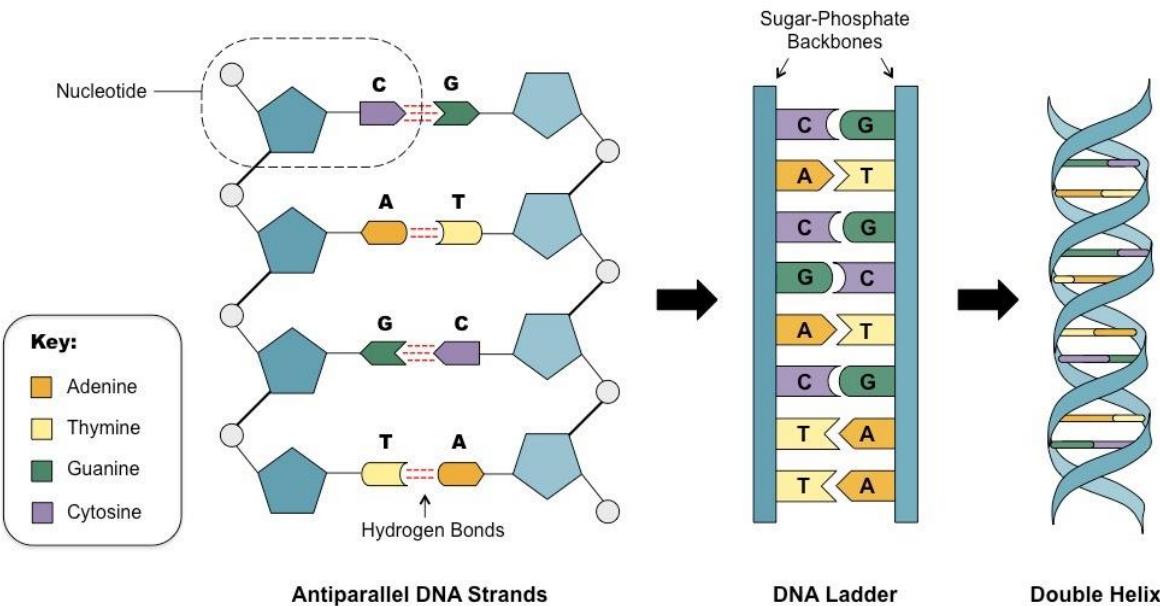
روش: فرض می‌کنیم جواب زیرمسئله‌ها بهینه نیستند و نتیجه می‌گیریم جواب مسئله نیز بهینه نیست

# زیرمسئلهای با همپوشانی

- عموماً تعداد زیرمسئلهای از مرتبه چندجمله‌ای نسبت به  $n$  بخوردارند
- اما وقتی حل به روش بازگشتی نمایی می‌شود ← یعنی هر زیرمسئله چندبار حل شده!
- پس زیرمسئلهای با همپوشانی داریم



# LCS طولانی‌ترین زیردنباله مشترک



- مثال استخراج شباخت بین DNA

adenine, guanine, cytosine, and thymine  
 $\{A, C, G, T\}$

$S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA$   
 $S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA$

GTCGTCGGAAGCCGGCCGAA

# طولانی‌ترین زیردنباله مشترک LCS

$X = \langle x_1, x_2, \dots, x_m \rangle \quad x_i \in S$

$Z = \langle z_1, z_2, \dots, z_k \rangle \quad z_j \in S$

- تعریف علمی زیردنباله:

وجود دارد اندیس‌هایی  $i_1 < i_2 < \dots < i_k$  بطوریکه

برای تمام  $a$  در محدوده  $1 \leq a \leq k$  داشته باشیم  $x_{i_a} = z_a$

- تعریف زیر دنباله مشترک:

فرض کنید  $X$  و  $Y$  دو دنباله از مجموعه  $S$  باشند. در اینصورت دنباله  $Z$  در صورتی یک زیردنباله مشترک است که داشته باشیم:

۱) زیردنباله  $X$  باشد      ۲) زیردنباله  $Y$  باشد

# LCS برنامه‌نویسی پویا و حل

پایه‌های برنامه‌نویسی پویا



- .1 مشخص کردن ساختار یک جواب بهینه
- .2 تعیین مقدار برای یک جواب بهینه بصورت بازگشتی
- .3 محاسبه مقدار یک جواب بهینه، معمولاً بصورت پایین به بالا
- .4 جواب بهینه را از روی اطلاعات محاسبه شده بدست بیاور

# ۱. تعیین ساختار جواب بهینه LCS

$$X = \langle x_1, x_2, \dots, x_m \rangle \quad x \in S$$

$$Y = \langle y_1, y_2, \dots, y_n \rangle \quad y \in S$$

$$Z = \langle z_1, z_2, \dots, z_k \rangle \quad z \in S$$

دنباله‌های ورودی

LCS دنباله‌های ورودی

آیا ویژگی زیرساختار بهینه وجود دارد؟

- راه حل brute-force

- یافتن همه زیردنباله‌های  $X$

- چک کردن آن با  $Y$

- نگهداری بلندترین مشترک‌ها

- تعداد زیردنباله‌های  $X$ :  $2^m$

۱) راه حل مسئله شامل یک انتخاب باشد و آن انتخاب یک یا چند زیرمسئله جدید ایجاد کند

۲) فرض میکنیم که راه حل بهینه مسئله را در اختیار داریم (روش آن مهم نیست)

۳) با در اختیار داشتن انتخاب مرحله اول تعیین میکنیم چه زیرمسئله‌هایی متنج میشود

۴) نشان میدهیم برای دستیابی به راه حل بهینه میبایست جواب زیرمسئله‌ها نیز بهینه باشد

# LCS . ۱ . تعیین ساختار جواب بهینه

$$X = \langle x_1, x_2, \dots, x_m \rangle \quad x \in S$$

$$Y = \langle y_1, y_2, \dots, y_n \rangle \quad y \in S$$

$$Z = \langle z_1, z_2, \dots, z_k \rangle \quad z \in S$$

پیشوند  $i$

$$X = \langle x_1, x_2, \dots, x_m \rangle$$

$$X_i = \langle x_1, x_2, \dots, x_i \rangle$$

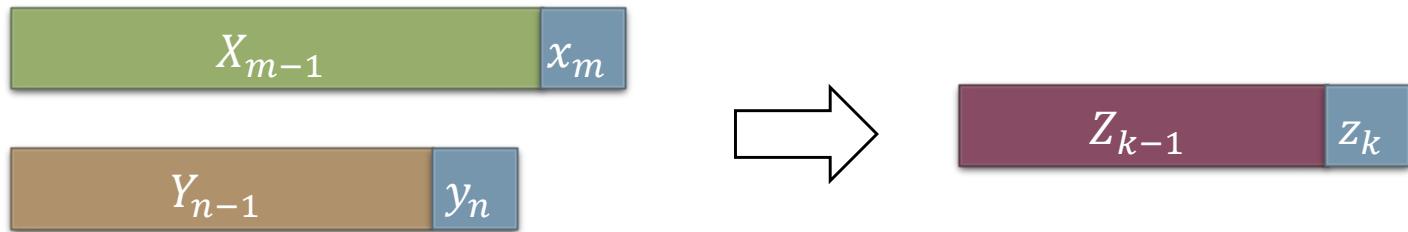
$$X = \langle A, B, C, B, D, A, B \rangle$$

$$X_4 = \langle A, B, C, B \rangle$$

$$X_0 = \langle \rangle$$

آیا ویژگی زیرساختار بهینه وجود دارد؟

- ۱) راه حل مسئله شامل یک انتخاب باشد و آن انتخاب یک یا چند زیرمسئله جدید ایجاد کند
- ۲) فرض میکنیم که راه حل بهینه مسئله را در اختیار داریم (روش آن مهم نیست)
- ۳) با در اختیار داشتن انتخاب مرحله اول تعیین میکنیم چه زیرمسئلهایی منتج میشود
- ۴) نشان میدهیم برای دستیابی به راه حل بهینه میباشد جواب زیرمسئلهها نیز بهینه باشد



- If  $x_m = y_n$ , then  $z_k = x_m = y_n$  and  $Z_{k-1}$  is an LCS of  $X_{m-1}$  and  $Y_{n-1}$ .

# ۱. تعیین ساختار جواب بهینه LCS

$$X = \langle x_1, x_2, \dots, x_m \rangle \quad x \in S$$

$$Y = \langle y_1, y_2, \dots, y_n \rangle \quad y \in S$$

$$Z = \langle z_1, z_2, \dots, z_k \rangle \quad z \in S$$

X پیشوند i

$$X = \langle x_1, x_2, \dots, x_m \rangle$$

$$X_i = \langle x_1, x_2, \dots, x_i \rangle$$

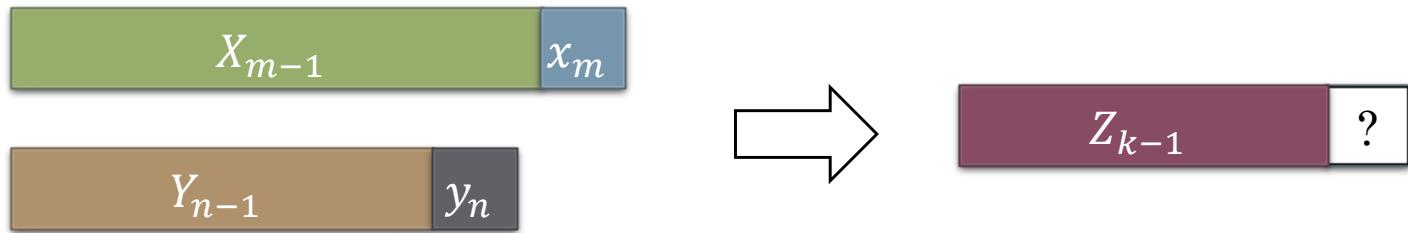
$$X = \langle A, B, C, B, D, A, B \rangle$$

$$X_4 = \langle A, B, C, B \rangle$$

$$X_0 = \langle \rangle$$

آیا ویژگی زیرساختار بهینه وجود دارد؟

- ۱) راه حل مسئله شامل یک انتخاب باشد و آن انتخاب یک یا چند زیرمسئله جدید ایجاد کند
- ۲) فرض میکنیم که راه حل بهینه مسئله را در اختیار داریم (روش آن مهم نیست)
- ۳) با در اختیار داشتن انتخاب مرحله اول تعیین میکنیم چه زیرمسئلهایی منتج میشود
- ۴) نشان میدهیم برای دستیابی به راه حل بهینه میباشد جواب زیرمسئلهها نیز بهینه باشد



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  $Y_{n-1}$ .

# LCS . تعیین ساختار جواب بهینه

$$X = \langle x_1, x_2, \dots, x_m \rangle \quad x \in S$$

$$Y = \langle y_1, y_2, \dots, y_n \rangle \quad y \in S$$

$$Z = \langle z_1, z_2, \dots, z_k \rangle \quad z \in S$$

آیا ویژگی زیرساختار بهینه وجود دارد؟

- ۱) راه حل مسئله شامل یک انتخاب باشد و آن انتخاب یک یا چند زیرمسئله جدید ایجاد کند
  - ۲) فرض میکنیم که راه حل بهینه مسئله را در اختیار داریم (روش آن مهم نیست)
  - ۳) با در اختیار داشتن انتخاب مرحله اول تعیین میکنیم چه زیرمسئلهایی منتج میشود
  - ۴) نشان میدهیم برای دستیابی به راه حل بهینه میباشد جواب زیرمسئلهها نیز بهینه باشد
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  $Y_{n-1}$ .

# ۲. تعیین یک جواب بازگشتی برای LCS

$$X = \langle x_1, x_2, \dots, x_m \rangle \quad x \in S$$

$$Y = \langle y_1, y_2, \dots, y_n \rangle \quad y \in S$$

$$Z = \langle z_1, z_2, \dots, z_k \rangle \quad z \in S$$

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  $Y_{n-1}$ .

$c[i, j]$  the length of an LCS of the sequences  $X_i$  and  $Y_j$

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

## ۳. محاسبه جواب بهینه پایین به بال!

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

فضای جستجوی زیرمسئله‌ها

$$O(2^{\min(m,n)}) \leq O(T) \leq O(2^{n+m}) \quad T(m, n) = T(m, n - 1) + T(m - 1, n)$$

فضای جستجوی زیرمسئله‌ها

$$\Theta(mn)$$

# ۳. محاسبه جواب بهینه پایین به بال!

LCS-LENGTH( $X, Y$ )

```

1    $m = X.length$ 
2    $n = Y.length$ 
3   let  $b[1..m, 1..n]$  and  $c[0..m, 0..n]$  be new tables
4   for  $i = 1$  to  $m$ 
5      $c[i, 0] = 0$ 
6   for  $j = 0$  to  $n$ 
7      $c[0, j] = 0$ 
8   for  $i = 1$  to  $m$ 
9     for  $j = 1$  to  $n$ 
10    if  $x_i == y_j$ 
11       $c[i, j] = c[i - 1, j - 1] + 1$ 
12       $b[i, j] = “↖”$ 
13    elseif  $c[i - 1, j] \geq c[i, j - 1]$ 
14       $c[i, j] = c[i - 1, j]$ 
15       $b[i, j] = “↑”$ 
16    else  $c[i, j] = c[i, j - 1]$ 
17       $b[i, j] = “←”$ 
18   return  $c$  and  $b$ 
```

$$b[i, j] : \langle \text{“←”} \quad \text{“↑”} \quad \text{“↖”} \rangle$$

(3)      (2)      (1)

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  $Y_{n-1}$ .

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

# ۱۴. بازسازی جواب بهینه!

PRINT-LCS( $b, X, i, j$ )

```

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

```

$b[i, j] : \langle \leftarrow \uparrow \nwarrow \rangle$

(3)      (2)      (1)

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  $Y_{n-1}$ .

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

# مثال

LCS-LENGTH( $X, Y$ )

```

1   $m = X.length$ 
2   $n = Y.length$ 
3  let  $b[1..m, 1..n]$  and  $c[0..m, 0..n]$  be new tables
4  for  $i = 1$  to  $m$ 
5     $c[i, 0] = 0$ 
6  for  $j = 0$  to  $n$ 
7     $c[0, j] = 0$ 
8  for  $i = 1$  to  $m$ 
9    for  $j = 1$  to  $n$ 
10   if  $x_i == y_j$ 
11      $c[i, j] = c[i - 1, j - 1] + 1$ 
12      $b[i, j] = "\nwarrow"$ 
13   elseif  $c[i - 1, j] \geq c[i, j - 1]$ 
14      $c[i, j] = c[i - 1, j]$ 
15      $b[i, j] = "\uparrow"$ 
16   else  $c[i, j] = c[i, j - 1]$ 
17      $b[i, j] = "\leftarrow"$ 
18 return  $c$  and  $b$ 
```

PRINT-LCS( $b, X, i, j$ )

```

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

$$X = \langle A, B, C, B, D, A, B \rangle$$

$$Y = \langle B, D, C, A, B, A \rangle$$

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

# تحليل زمان اجرا

LCS-LENGTH( $X, Y$ )

```

1    $m = X.length$ 
2    $n = Y.length$ 
3   let  $b[1..m, 1..n]$  and  $c[0..m, 0..n]$  be new tables
4   for  $i = 1$  to  $m$ 
5      $c[i, 0] = 0$ 
6   for  $j = 0$  to  $n$ 
7      $c[0, j] = 0$ 
8   for  $i = 1$  to  $m$ 
9     for  $j = 1$  to  $n$ 
10    if  $x_i == y_j$ 
11       $c[i, j] = c[i - 1, j - 1] + 1$ 
12       $b[i, j] = "\nwarrow"$ 
13    elseif  $c[i - 1, j] \geq c[i, j - 1]$ 
14       $c[i, j] = c[i - 1, j]$ 
15       $b[i, j] = "\uparrow"$ 
16    else  $c[i, j] = c[i, j - 1]$ 
17       $b[i, j] = "\leftarrow"$ 
18 return  $c$  and  $b$ 

```

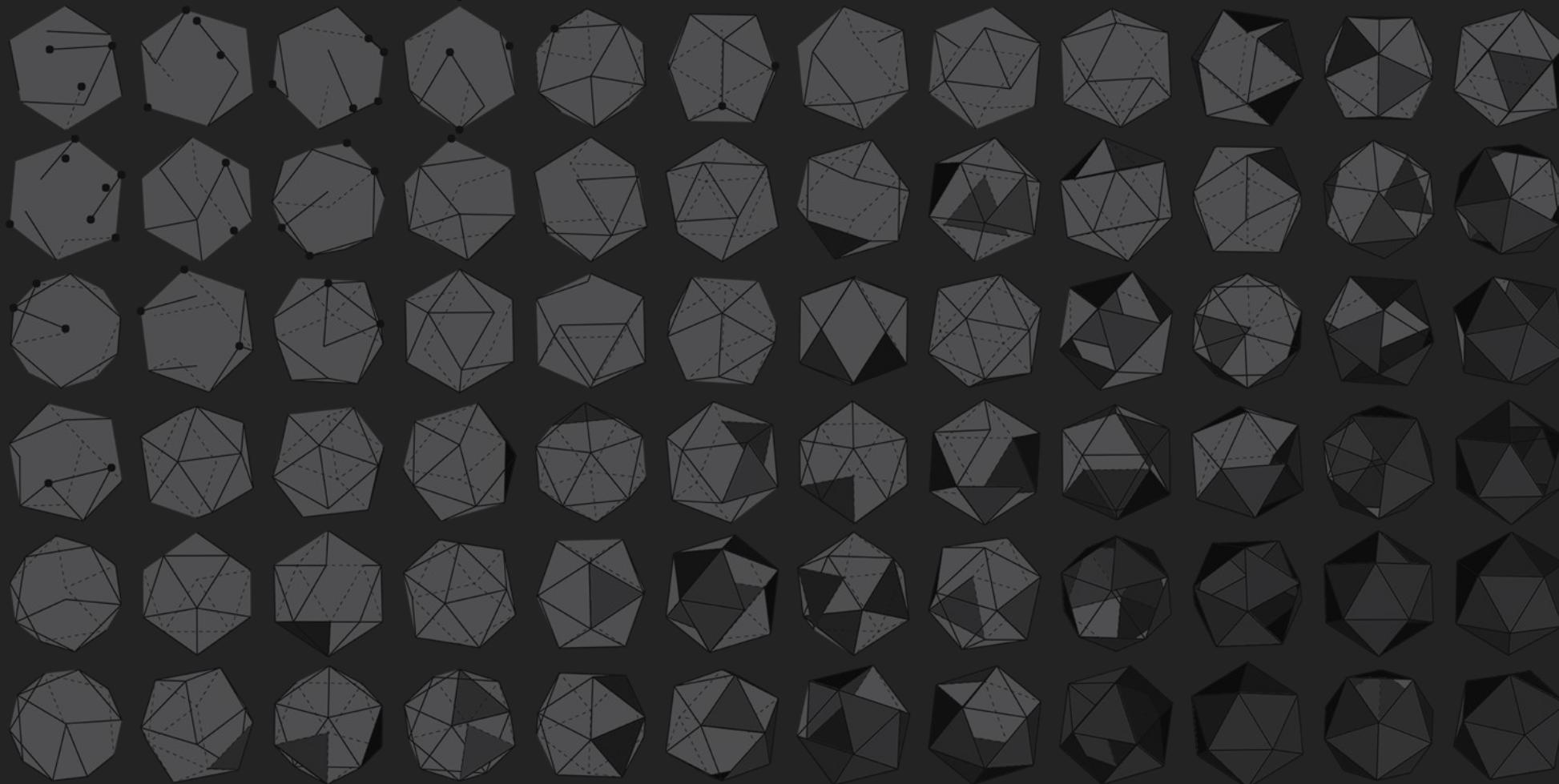
PRINT-LCS( $b, X, i, j$ )

```

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

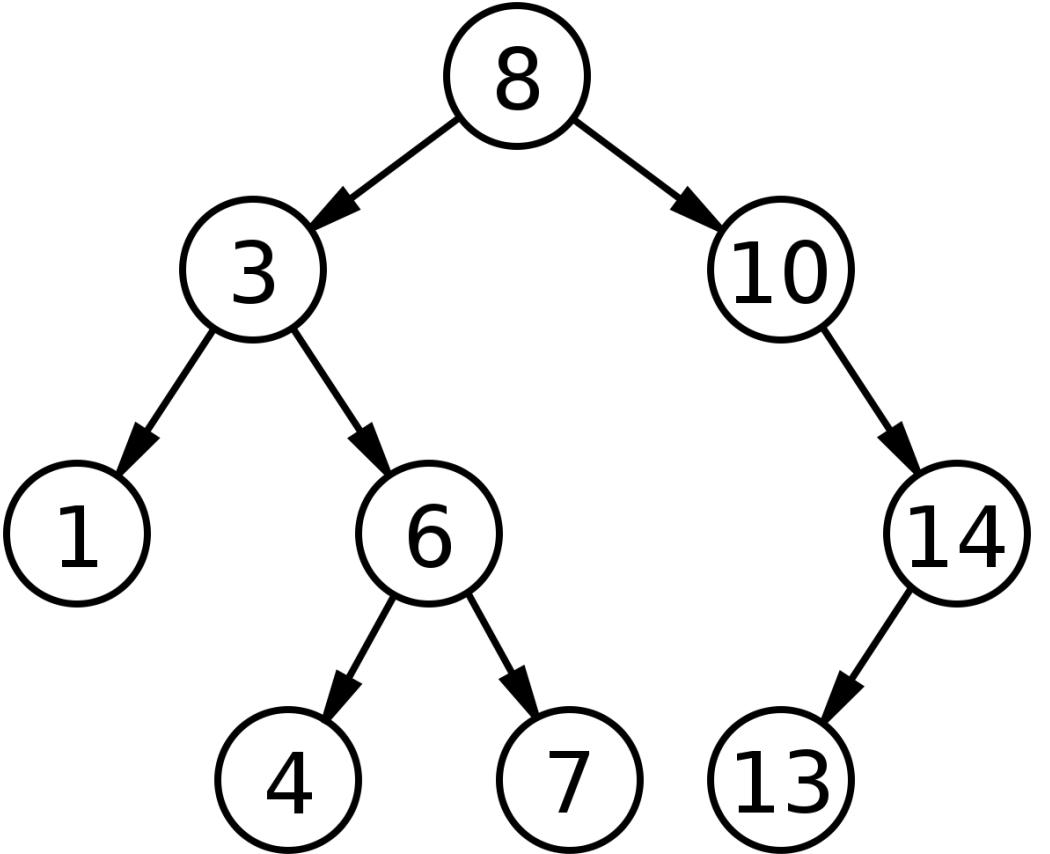
```

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

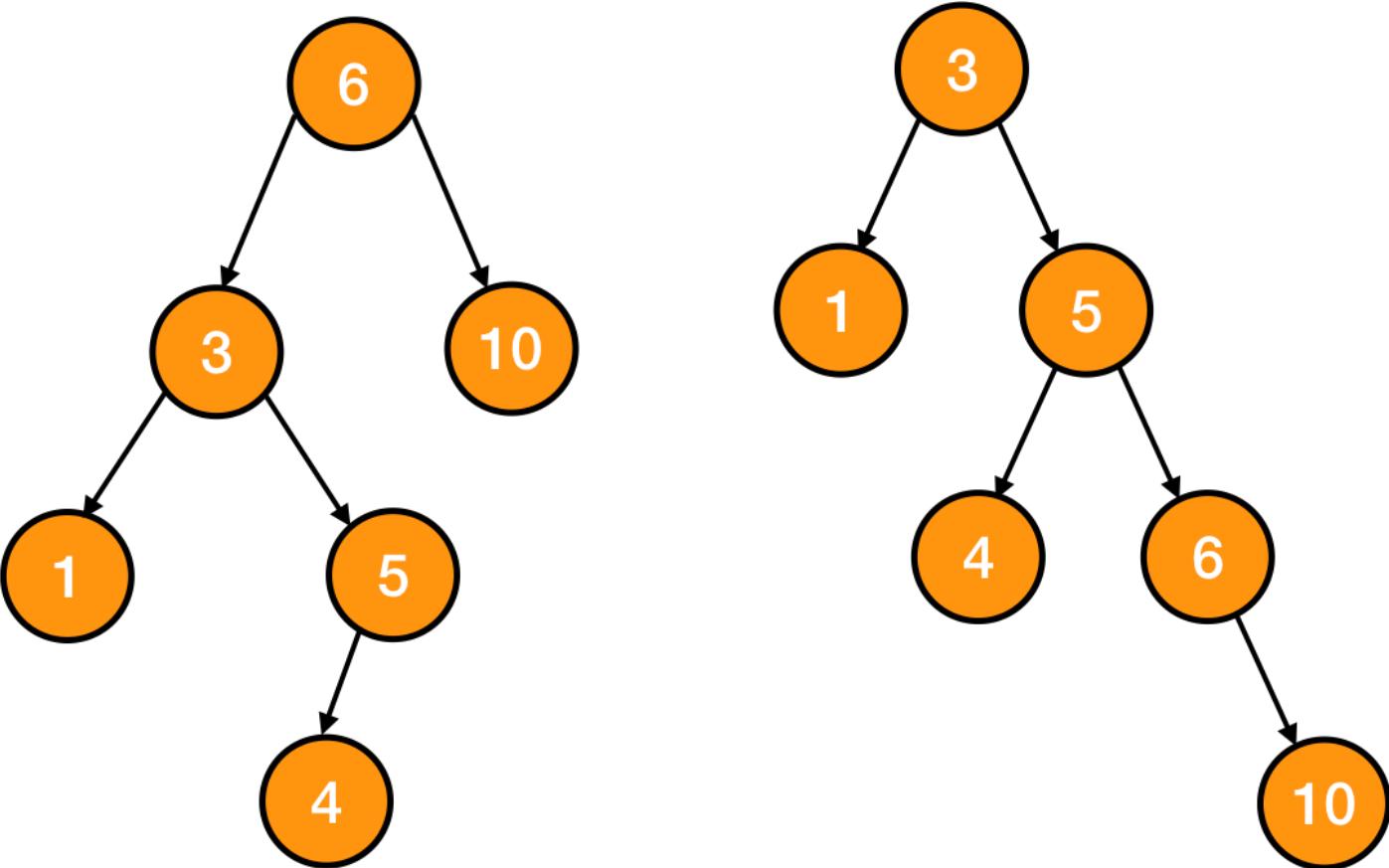


# درخت دودویی جستجو BST

# درخت دودویی جستجوی



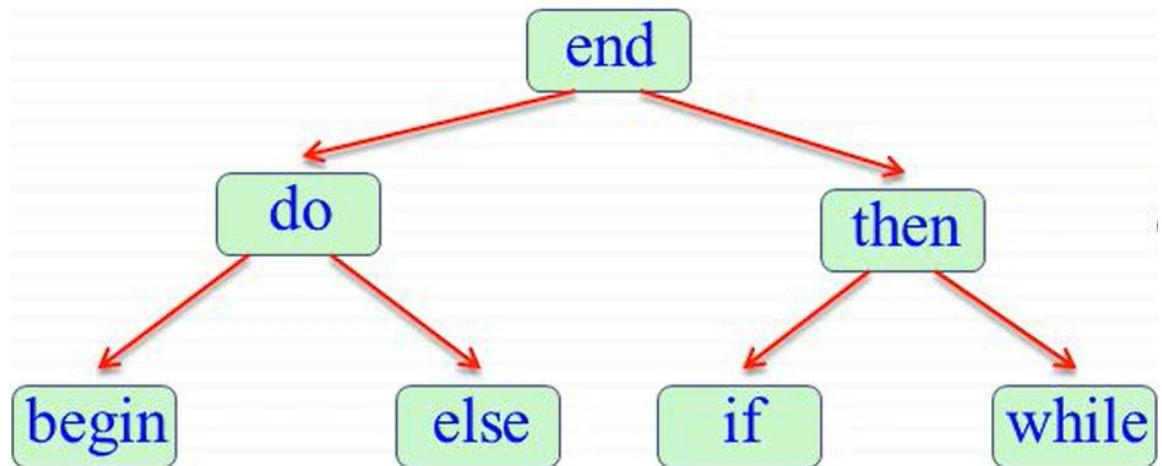
# درخت دودویی جستجوی



Two Binary Search Trees Representing Same Set

# مثال دیکشنری انگلیسی به فرانسه با BST

- مثال درخت دودویی جستجوی: ترجمه انگلیسی به فرانسوی
- ایجاد درخت دودویی جستجو با جفت لغت‌های انگلیسی و ترجمه فرانسه
  - لغت انگلیسی: **key**
  - ترجمه فرانسوی لغت **key**: **satellite data**

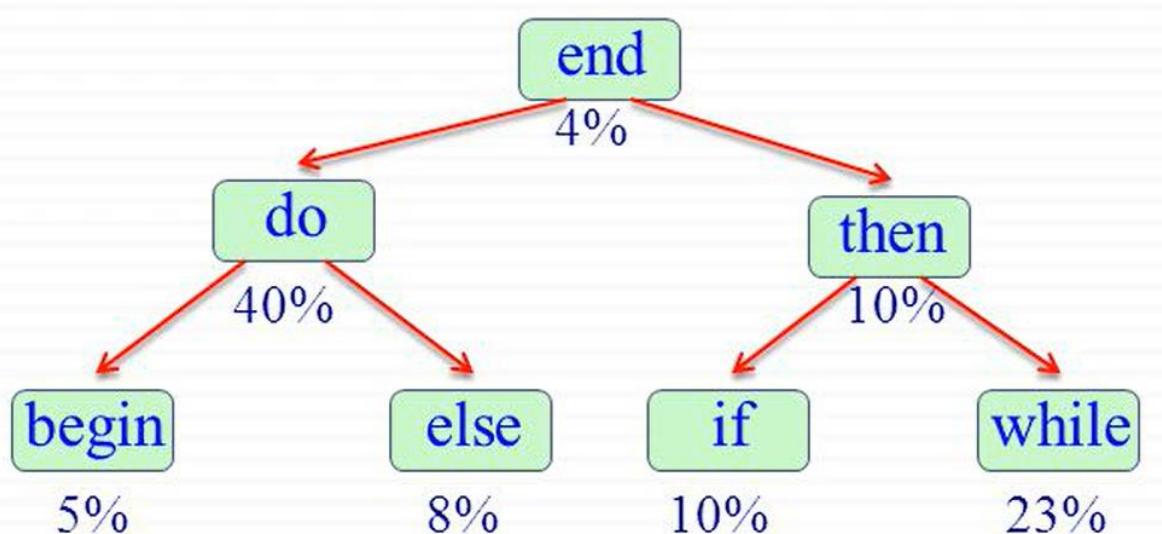


A	B	C	D	E	F	G	H	I
J	K	L	M	N	O	P	Q	
R	S	T	U	V	W	X	Y	Z

# هزینه کل جستجو در دیکشنری

- فرض کنید که درصد رخداد لغت‌های انگلیسی در متن را بدانیم

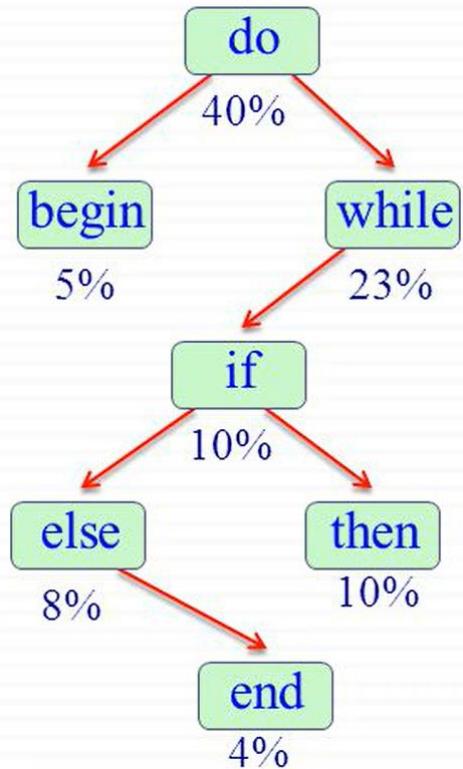
<u>begin</u>	<u>do</u>	<u>else</u>	<u>end</u>	<u>if</u>	<u>then</u>	<u>while</u>
5%	40%	8%	4%	10%	10%	23%



$$Total\ cost = \sum_{i \in English\ words} [(depth(i) + 1) \times freq(i)] = 1 \times .04 + 2 \times .4 + 2 \times .1 + 3 \times .05 + 3 \times .08 + 3 \times .1 + 3 \times .23 = 2.42$$

# درخت جستجوی دودویی بهینه

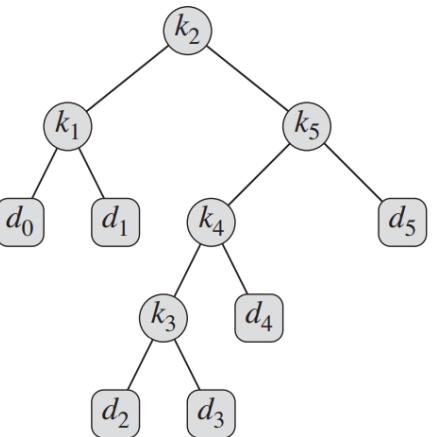
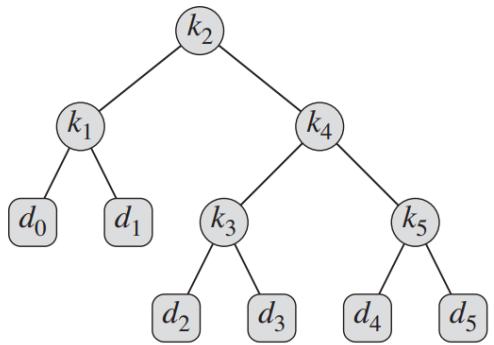
- $k_1 < k_2 < \dots < k_n$      $K = \langle k_1, k_2, \dots, k_n \rangle$
- ورودی: عدد  $key$  مجزا مرتب شده صعودی
  - برای هر  $k_i$ , احتمال  $p_i$  معرف احتمال جستجوی  $key$
  - خروجی: درخت جستجوی دودویی بهینه با کمترین هزینه جستجوی کل



$$\begin{aligned} \text{Total cost} &= 1 \times .4 + 2 \times .05 + 2 \times .23 + 3 \times .1 + 4 \times .08 + 4 \times .1 + 5 \times .04 \\ &= 2.18 \end{aligned}$$

# درخت جستجوی دودویی بهینه

- ورودی:  $n$  عدد  $key$  مجزا مرتب شده صعودی  $K = \langle k_1, k_2, \dots, k_n \rangle$
- برای هر  $k_i$ , احتمال  $p_i$  معرف احتمال جستجوی  $key$
- خروجی: درخت جستجوی دودویی بهینه با کمترین هزینه جستجوی کل
- برخی جستجوها شاید در مجموعه  $K$  وجود نداشته باشد
- نیاز به  $n + 1$  کلید بعنوان *dummy key*
- معرف همه  $key$ هایی که بین  $k_i$  و  $k_{i+1}$  قراردارد



$$K = \langle k_1, k_2, \dots, k_n \rangle$$

$$D = \langle d_0, d_1, d_2, \dots, d_n \rangle$$

# درخت جستجوی دودویی بهینه

- ورودی:  $n$  عدد  $key$  مجزا مرتب شده صعودی
  - برای هر  $k_i$ ، احتمال  $p_i$  معرف احتمال جستجوی  $key$
  - $D = \langle d_0, d_1, d_2, \dots, d_n \rangle$  dummy key عدد  $n + 1$
  - برای هر  $d_i$ ، احتمال  $q_i$  معرف احتمال جستجوی dummy key
- خروجی: درخت جستجوی دودویی بهینه با کمترین هزینه جستجوی کل

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

# تعیین BST بهینه به روش برنامه‌نویسی پویا

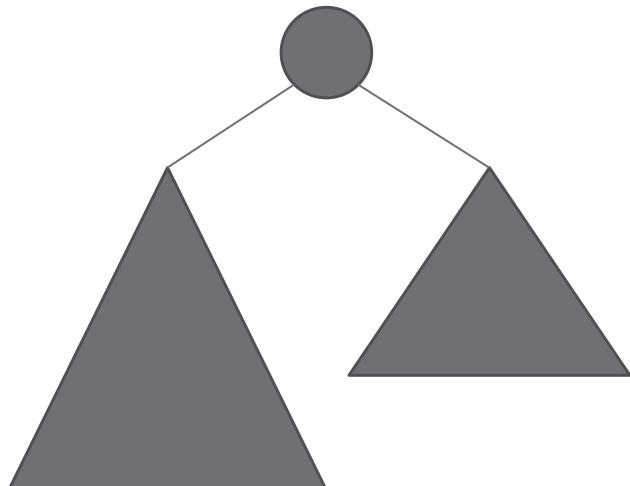
$$k_i, \dots, k_j \quad 1 \leq i \leq j \leq n$$

- تعداد همه BST های ممکن با  $n$  گره:  $\Omega(4^n / n^{3/2})$

## ۱) تعیین ساختار بهینه

- هر زیردرختی در BST می‌بایست شامل  $key$  های پشتسرهم باشد

- درختی با  $key$  های  $d_{i-1}, \dots, d_j$  حتماً شامل  $key$  های  $k_i, \dots, k_j$  نیز هست



- ۱) انتخاب: تعیین ریشه درخت بهینه  $k_r$
- ۲) در اختیار داشتن جواب برای زیردرخت‌های بهینه
- ۳) تعیین زیرمسئله‌ها
- ۴) اثبات زیرساختار بهینه

# تعیین BST بهینه به روش برنامه‌نویسی پویا

## ۲) تعیین مقدار یک جواب بهینه بصورت بازگشتی

$k_i, \dots, k_j \quad i \geq 1, j \leq n$ , and  $j \geq i - 1$ .

$e[i, j]$

امید هزینه جستجو در درخت بهینه شامل  $k_i, \dots, k_j$

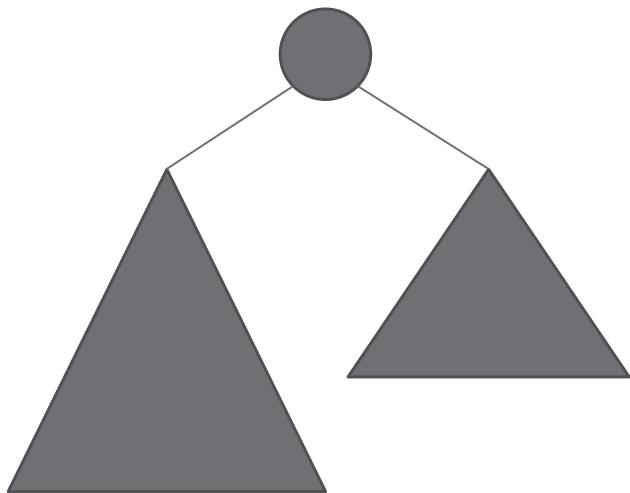
$e[1, n]$

امید هزینه جستجو در درخت بهینه شامل همه  $key$ ها

$j = i - 1$

هیچ  $key$  ای در درخت وجود ندارد!  
(ولی یک dummy وجود دارد)

$e[i, i - 1] = q_{i-1}$



$$e[i, r - 1] = 1 + \sum_{a=i}^{r-1} \text{depth}_T(k_a) \cdot p_a + \sum_{a=i-1}^{r-1} \text{depth}_T(d_a) \cdot q_a$$

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

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

# تعیین BST بهینه به روش برنامه‌نویسی پویا

## ۲) تعیین مقدار یک جواب بهینه بصورت بازگشتی

$e[i, j]$

$k_i, \dots, k_j$  امید هزینه جستجو در درخت بهینه شامل

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

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

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

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

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

# تعیین BST بهینه به روش برنامه‌نویسی پویا

۳) محاسبه کمترین هزینه بصورت پایین به بالا

OPTIMAL-BST( $p, q, n$ )

```

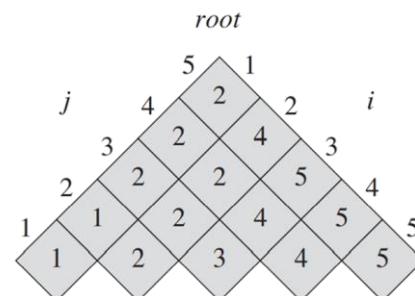
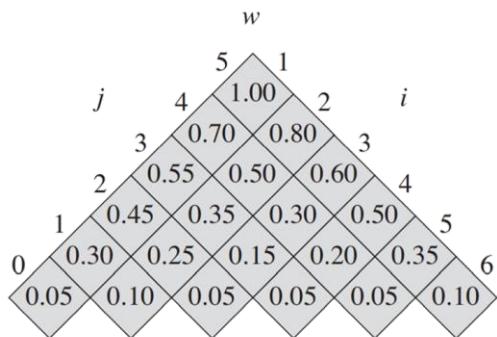
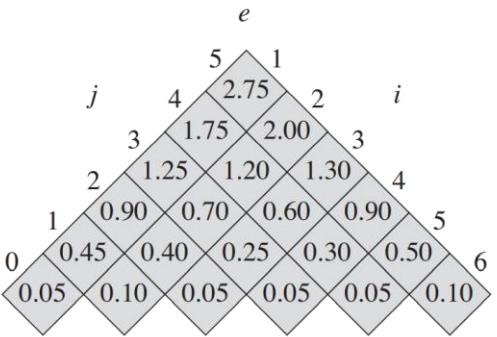
1 let  $e[1..n + 1, 0..n]$ ,  $w[1..n + 1, 0..n]$ ,
    and  $root[1..n, 1..n]$  be new tables
2 for  $i = 1$  to  $n + 1$ 
3      $e[i, i - 1] = q_{i-1}$ 
4      $w[i, i - 1] = q_{i-1}$ 
5 for  $l = 1$  to  $n$ 
6     for  $i = 1$  to  $n - l + 1$ 
7          $j = i + l - 1$ 
8          $e[i, j] = \infty$ 
9          $w[i, j] = w[i, j - 1] + p_j + q_j$ 
10        for  $r = i$  to  $j$ 
11             $t = e[i, r - 1] + e[r + 1, j] + w[i, j]$ 
12            if  $t < e[i, j]$ 
13                 $e[i, j] = t$ 
14                 $root[i, j] = r$ 
15 return  $e$  and  $root$ 
```

# تعیین BST بهینه به روش برنامه‌نویسی پویا

OPTIMAL-BST( $p, q, n$ )

```

1 let  $e[1..n + 1, 0..n]$ ,  $w[1..n + 1, 0..n]$ ,
   and  $root[1..n, 1..n]$  be new tables
2 for  $i = 1$  to  $n + 1$ 
3    $e[i, i - 1] = q_{i-1}$ 
4    $w[i, i - 1] = q_{i-1}$ 
5 for  $l = 1$  to  $n$ 
6   for  $i = 1$  to  $n - l + 1$ 
7      $j = i + l - 1$ 
8      $e[i, j] = \infty$ 
9      $w[i, j] = w[i, j - 1] + p_j + q_j$ 
10    for  $r = i$  to  $j$ 
11       $t = e[i, r - 1] + e[r + 1, j] + w[i, j]$ 
12      if  $t < e[i, j]$ 
13         $e[i, j] = t$ 
14         $root[i, j] = r$ 
15 return  $e$  and  $root$ 
```



node	probability
$k_1$	0.15
$k_2$	0.10
$k_3$	0.05
$k_4$	0.10
$k_5$	0.20
$d_0$	0.05
$d_1$	0.10
$d_2$	0.05
$d_3$	0.05
$d_4$	0.05
$d_5$	0.10