

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



طراحی و تحلیل الگوریتم‌ها

مبحث پنجم

# روش‌های طراحی الگوریتم

روش برنامه‌ریزی پویا

**Methods of Algorithm Design: Dynamic Programming**

کاظم فولادی

دانشکده مهندسی برق و کامپیوتر

دانشگاه تهران

<http://courses.fouladi.ir/algorithm>

## محاسبه‌ی تابع فیبوناچی: روش تقسیم و غلبه

$$f(n) = f(n - 1) + f(n - 2)$$

$$f(0) = 0$$

$$f(1) = 1$$

FIB-NUMBER( $n$ )

**if**  $n \geq 0$  **and**  $n \leq 1$  **then**

**return**  $n$

**else if**  $n > 1$  **then**

**return** FIB-NUMBER( $n - 1$ ) + FIB-NUMBER( $n - 2$ )

$$T(n) \in \Theta(\alpha^n), \alpha > 1$$

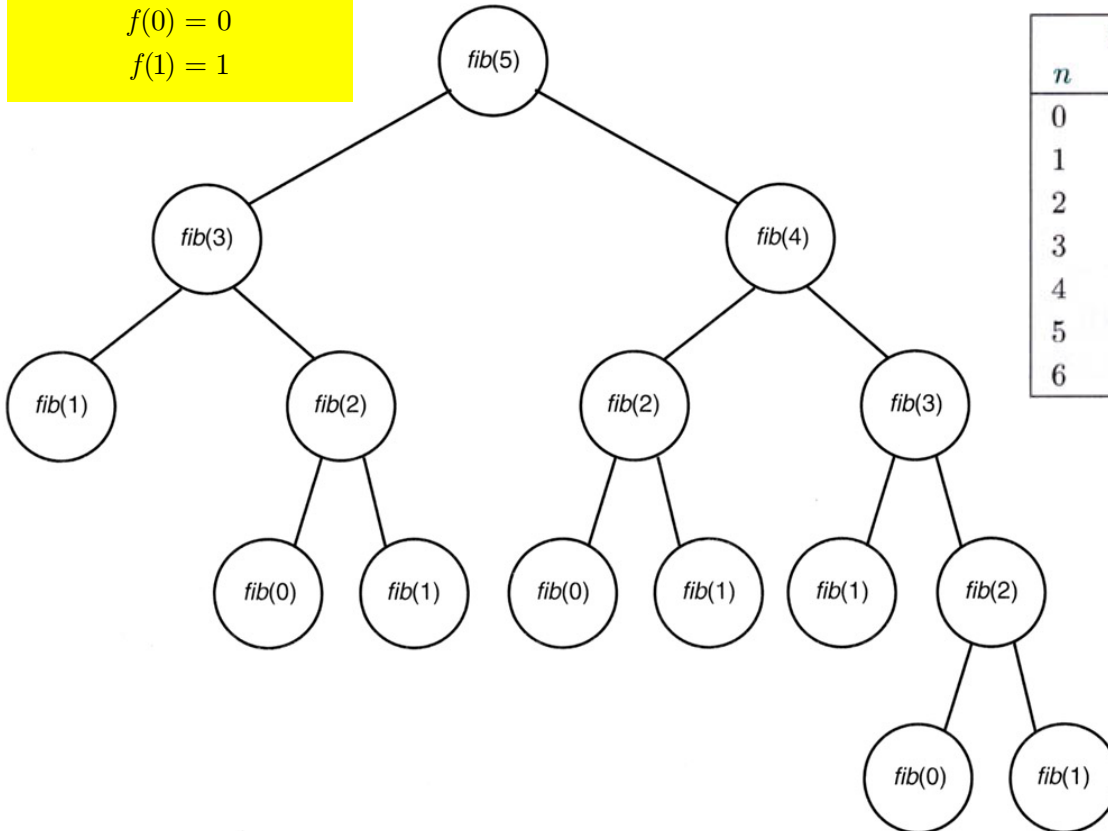
روش تقسیم و غلبه به دلیل وجود همپوشانی زیاد  
بین دو زیرمسئله، کارآمد نیست!

## محاسبه‌ی تابع فیبوناچی: روش تقسیم و غلبه (مثال)

$$f(n) = f(n - 1) + f(n - 2)$$

$$f(0) = 0$$

$$f(1) = 1$$



| $n$ | Number of Terms Computed |
|-----|--------------------------|
| 0   | 1                        |
| 1   | 1                        |
| 2   | 3                        |
| 3   | 5                        |
| 4   | 9                        |
| 5   | 15                       |
| 6   | 25                       |

## محاسبه‌ی تابع فیبوناچی: روش برنامه‌ریزی پویا

$$f(n) = f(n - 1) + f(n - 2)$$

$$f(0) = 0$$

$$f(1) = 1$$

```

FIB-NUMBER(n)
  F ← array[0..n]
  F[0] ← 0
  F[1] ← 1
  if n > 1 then
    for i ← 2 to n do
      F[i] ← F[i - 1] + F[i - 2]
  return F[n]

```

از یک جدول برای نگهداری راه‌حل نمونه‌های کوچکتر مسئله استفاده می‌شود تا از محاسبه‌ی تکراری آنها اجتناب شود.

$$T(n) \in \Theta(n)$$

$$S(n) \in \Theta(n)$$

## محاسبه‌ی تابع فیبوناچی: روش برنامه‌ریزی پویا با بهبود فضای مصرفی

$$f(n) = f(n - 1) + f(n - 2)$$

$$f(0) = 0$$

$$f(1) = 1$$

FIB-NUMBER-IMPROVED( $n$ )

$F_1 \leftarrow 0$

$F_2 \leftarrow 1$

**if**  $n = 0$  **then**

$F \leftarrow 0$

**else if**  $n = 1$  **then**

$F \leftarrow 1$

**else if**  $n > 1$  **then**

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

$F \leftarrow F_1 + F_2$

$F_1 \leftarrow F_2$

$F_2 \leftarrow F$

**return**  $F$

$T(n) \in \Theta(n)$

$S(n) \in \Theta(1)$

از آنجا که برای محاسبه‌ی مقدار تابع فقط به دو مقدار قبلی آن نیاز است، به جای آرایه از سه متغیر استفاده می‌کنیم.

## برای حل یک مسئله با روش برنامه‌ریزی پویا:

- یافتن یک فرمول بازگشتی که راه حل مسئله برای یک نمونه را از روی راه حل نمونه‌های کوچکتر همان مسئله مشخص می‌کند.
- حل نمونه‌ی مسئله به صورت **پایین به بالا** با شروع از نمونه‌های کوچکتر.

مسائلی با این روش قابل حل هستند که **اصل بهینگی** در مورد آنها صادق باشد: راه حل بهینه برای هر نمونه از راه حل بهینه‌ی نمونه‌های کوچکتر حاصل می‌شود.

منظور از برنامه‌ریزی، استفاده از جدول‌بندی در حل مسئله است. اصطلاح «پویا» از کاربرد این روش در سیستم‌های کنترل دینامیکی آمده است (Bellman, 1959).

## محاسبه‌ی ضریب دو جمله‌ای

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}, 0 \leq k \leq n \quad \text{تعریف:}$$

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k}, & 0 < k < n \\ 1, & k = 0, k = n \end{cases}$$

فرمول  
بازگشتی:

## محاسبه‌ی ضریب دو جمله‌ای: روش تقسیم و غلبه

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & , 0 < k < n \\ 1 & , k = 0, k = n \end{cases}$$

**BINOM**( $n, k$ )

**if**  $k = 0$  **or**  $k = n$  **then**

**return** 1

**else**

**return** **BINOM**( $n - 1, k - 1$ ) + **BINOM**( $n - 1, k$ )

$$T(n) = 2 \binom{n}{k} - 1$$

زمان اجرا: بر حسب تعداد عمل جمع  
اثبات: با استقرای ریاضی



## محاسبه‌ی ضریب دو جمله‌ای: روش برنامه‌ریزی پویا

از آرایه‌ی دوبعدی  $B[0..n,0..k]$  استفاده می‌کنیم که  $B[i,j] := \binom{i}{j}$

$$B[i,j] = \begin{cases} B[i-1,j] + B[i-1,j-1] & , 0 < j < i \\ 1 & , j = 0, j = i \end{cases}$$

**BINOM-DP**( $n,k$ )

$B \leftarrow \text{array}[0..n,0..k]$

**for**  $i \leftarrow 0$  **to**  $n$  **do**

**for**  $j \leftarrow 0$  **to**  $\min(i, k)$  **do**

**if**  $j = 0$  **or**  $j = i$  **then**

$B[i,j] \leftarrow 1$

**else**

$B[i,j] \leftarrow B[i-1,j-1] + B[i-1,j]$

**return**  $B[n,k]$

|   | 0 | 1 | 2 | 3 | 4 | $j$ | $k$ |
|---|---|---|---|---|---|-----|-----|
| 0 | 1 |   |   |   |   |     |     |
| 1 | 1 | 1 |   |   |   |     |     |
| 2 | 1 | 2 | 1 |   |   |     |     |
| 3 | 1 | 3 | 3 | 1 |   |     |     |
| 4 | 1 | 4 | 6 | 4 | 1 |     |     |

$B[i-1][j-1]$     $B[i-1][j]$

↓   ↓

→  $B[i][j]$

$T(n) \in \Theta(nk)$

$S(n) \in \Theta(nk)$

زمان اجرا (بر حسب تعداد عمل جمع):  
فضای مصرفی (جدول):

## محاسبه‌ی ضریب دو جمله‌ای: روش برنامه‌ریزی پویا

- $B[0,0] = 1$

**Compute row 1:**

- $B[1,0] = 1$
- $B[1,1] = 1$

**Compute row 2:**

- $B[2,0] = 1$
- $B[2,1] = B[1,0] + B[1,1] = 1+1 = 2$
- $B[2,2] = 1$

**Compute row 3:**

- $B[3,0] = 1$
- $B[3,1] = B[2,0] + B[2,1] = 1+2 = 3$
- $B[3,2] = B[2,1] + B[2,2] = 2+1 = 3$

**Compute row 4:**

- $B[4,0] = 1$
- $B[4,1] = B[3,0] + B[3,1] = 1+3 = 4$
- $B[4,2] = B[3,1] + B[3,2] = 3+3 = 6$

- مثال  $B[4,2] := \binom{4}{2}$

|   | 0 | 1 | 2 | 3 | 4 | $j$ | $k$ |
|---|---|---|---|---|---|-----|-----|
| 0 | 1 |   |   |   |   |     |     |
| 1 | 1 | 1 |   |   |   |     |     |
| 2 | 1 | 2 | 1 |   |   |     |     |
| 3 | 1 | 3 | 3 | 1 |   |     |     |
| 4 | 1 | 4 | 6 | 4 | 1 |     |     |

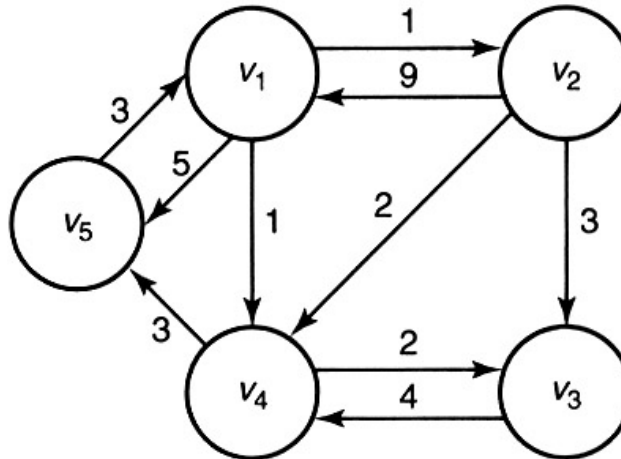
$i$   
 $n$

$$\begin{array}{c}
 B[i-1][j-1] \quad B[i-1][j] \\
 \downarrow \quad \downarrow \\
 \rightarrow B[i][j]
 \end{array}$$

## یافتن کوتاهترین مسیر بین هر دو رأس یک گراف

FLOYD-WARSHALL ALGORITHM

هدف: یافتن کوتاهترین مسیر بین هر دو رأس دلخواه از یک گراف جهت‌دار وزن‌دار

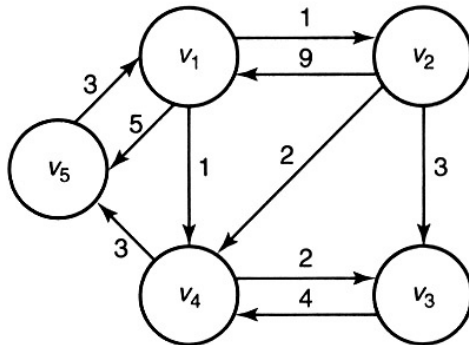


## یافتن کوتاه‌ترین مسیر بین هر دو رأس یک گراف

## FLOYD-WARSHALL ALGORITHM

هدف: یافتن کوتاه‌ترین مسیر بین هر دو رأس دلخواه از یک گراف جهت‌دار وزن‌دار  
 ورودی: ماتریس وزن‌های گراف

$$W[i, j] = \begin{cases} \text{weight on edge} & , (v_i, v_j) \in E(G) \\ \infty & , (v_i, v_j) \notin E(G) \\ 0 & , i = j \end{cases}$$

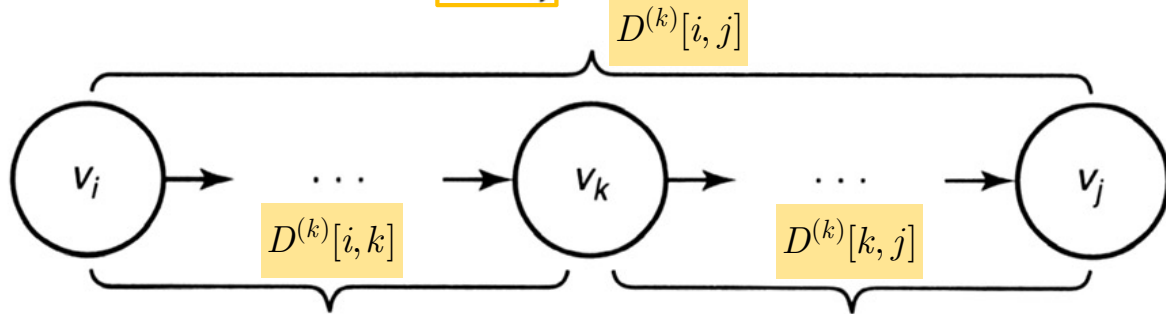


|   | 1        | 2        | 3        | 4        | 5        |
|---|----------|----------|----------|----------|----------|
| 1 | 0        | 1        | $\infty$ | 1        | 5        |
| 2 | 9        | 0        | 3        | 2        | $\infty$ |
| 3 | $\infty$ | $\infty$ | 0        | 4        | $\infty$ |
| 4 | $\infty$ | $\infty$ | 2        | 0        | 3        |
| 5 | 3        | $\infty$ | $\infty$ | $\infty$ | 0        |

W

## یافتن کوتاه‌ترین مسیر بین هر دو رأس یک گراف

A shortest path from  $v_i$  to  $v_j$  using only vertices in  $\{v_1, v_2, \dots, v_k\}$



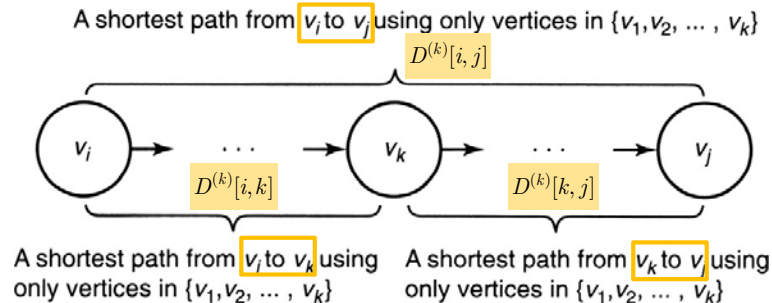
A shortest path from  $v_i$  to  $v_k$  using only vertices in  $\{v_1, v_2, \dots, v_k\}$

A shortest path from  $v_k$  to  $v_j$  using only vertices in  $\{v_1, v_2, \dots, v_k\}$

فرمول بازگشتی: کوتاه‌ترین مسیر بین دو رأس

$$D^{(k)}[i, j] = \min_{1 \leq k \leq n} \{D^{(k-1)}[i, j], D^{(k-1)}[i, k] + D^{(k-1)}[k, j]\}$$

## یافتن کوتاه‌ترین مسیر بین هر دو رأس یک گراف



فرمول بازگشتی: کوتاه‌ترین مسیر بین دو رأس

$$D^{(k)}[i, j] = \min_{1 \leq k \leq n} \{D^{(k-1)}[i, j], D^{(k-1)}[i, k] + D^{(k-1)}[k, j]\}$$

$$D^{(0)} \rightarrow D^{(1)} \rightarrow D^{(2)} \rightarrow \dots \rightarrow D^{(n-1)} \rightarrow D^{(n)}$$

$$\uparrow \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \uparrow$$

$$W \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad D$$

## یافتن کوتاه‌ترین مسیر بین هر دو رأس یک گراف

هدف: یافتن کوتاه‌ترین مسیر بین دو رأس دلخواه از یک گراف جهت‌دار وزن‌دار

ورودی: ماتریس وزن‌های گراف:  $W$

خروجی: ماتریس کوتاه‌ترین فاصله‌ها:  $D$

$$D^{(k)}[i, j] = \min_{1 \leq k \leq n} \{D^{(k-1)}[i, j], D^{(k-1)}[i, k] + D^{(k-1)}[k, j]\}$$

فرمول بازگشتی:  
کوتاه‌ترین مسیر بین دو رأس

$$D[i, j] \leftarrow \min_{1 \leq k \leq n} \{D[i, j], D[i, k] + D[k, j]\}$$

می‌توان کل محاسبات را با استفاده از یک آرایه‌ی  $D$  انجام داد زیرا مقادیر سطر  $k$  و ستون  $k$  در تکرار  $k$ -ام حلقه تغییر نمی‌کند:

SHORTEST-PATH( $W, n$ )

$D \leftarrow W$

for  $k \leftarrow 1$  to  $n$  do

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

for  $j \leftarrow 1$  to  $n$  do

$D[i, j] \leftarrow \min(D[i, j], D[i, k] + D[k, j])$

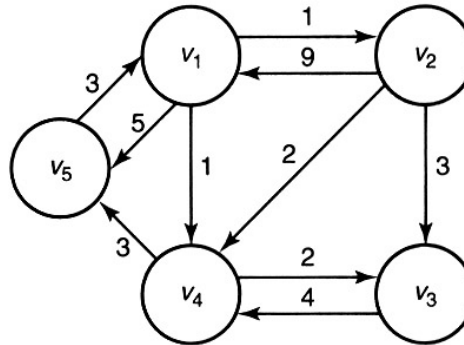
return  $D$

$$T(n) \in \Theta(n^3)$$

$$S(n) \in \Theta(n^2)$$

زمان اجرا:  
فضای مورد نیاز:

## یافتن کوتاهترین مسیر بین هر دو رأس یک گراف: مثال



|   | 1        | 2        | 3        | 4        | 5        |
|---|----------|----------|----------|----------|----------|
| 1 | 0        | 1        | $\infty$ | 1        | 5        |
| 2 | 9        | 0        | 3        | 2        | $\infty$ |
| 3 | $\infty$ | $\infty$ | 0        | 4        | $\infty$ |
| 4 | $\infty$ | $\infty$ | 2        | 0        | 3        |
| 5 | 3        | $\infty$ | $\infty$ | $\infty$ | 0        |

W

|   | 1  | 2  | 3 | 4 | 5 |
|---|----|----|---|---|---|
| 1 | 0  | 1  | 3 | 1 | 4 |
| 2 | 8  | 0  | 3 | 2 | 5 |
| 3 | 10 | 11 | 0 | 4 | 7 |
| 4 | 6  | 7  | 2 | 0 | 3 |
| 5 | 3  | 4  | 6 | 4 | 0 |

D



## یافتن کوتاه‌ترین مسیر بین هر دو رأس یک گراف

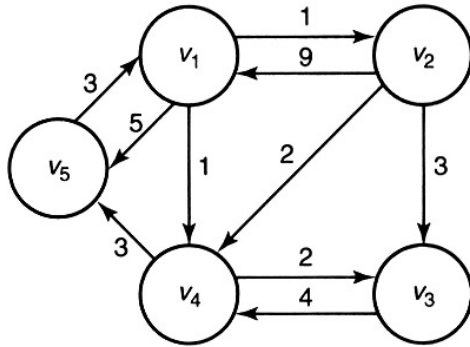
چاپ کوتاه‌ترین مسیر بین هر دو رأس گراف

هدف: یافتن کوتاه‌ترین مسیر بین دو رأس دلخواه از یک گراف جهت‌دار وزن‌دار

ورودی: ماتریس وزن‌های گراف:  $W$ خروجی: ماتریس کوتاه‌ترین فاصله‌ها:  $D$  و خود کوتاه‌ترین مسیرSHORTEST-PATH( $W, n$ ) $P \leftarrow \text{array}[1..n, 1..n]$  $P \leftarrow [0]$  $D \leftarrow W$ **for**  $k \leftarrow 1$  **to**  $n$  **do**    **for**  $i \leftarrow 1$  **to**  $n$  **do**        **for**  $j \leftarrow 1$  **to**  $n$  **do**            **if**  $D[i, j] > D[i, k] + D[k, j]$  **then**                 $D[i, j] \leftarrow D[i, k] + D[k, j]$                  $P[i, j] \leftarrow k$ **return**  $D$ PATH( $P, i, j$ )**if**  $P[i, j] \neq 0$  **then**    PATH( $P, i, P[i, j]$ )    **print**  $P[i, j]$     PATH( $P, P[i, j], j$ )بزرگترین اندیس رئوس میانی در کوتاه‌ترین مسیر از  $i$  تا  $j$  است.

## یافتن کوتاه‌ترین مسیر بین هر دو رأس یک گراف: مثال

چاپ کوتاه‌ترین مسیر بین هر دو رأس گراف



|   | 1        | 2        | 3        | 4        | 5        |
|---|----------|----------|----------|----------|----------|
| 1 | 0        | 1        | $\infty$ | 1        | 5        |
| 2 | 9        | 0        | 3        | 2        | $\infty$ |
| 3 | $\infty$ | $\infty$ | 0        | 4        | $\infty$ |
| 4 | $\infty$ | $\infty$ | 2        | 0        | 3        |
| 5 | 3        | $\infty$ | $\infty$ | $\infty$ | 0        |

W

|   | 1  | 2  | 3 | 4 | 5 |
|---|----|----|---|---|---|
| 1 | 0  | 1  | 3 | 1 | 4 |
| 2 | 8  | 0  | 3 | 2 | 5 |
| 3 | 10 | 11 | 0 | 4 | 7 |
| 4 | 6  | 7  | 2 | 0 | 3 |
| 5 | 3  | 4  | 6 | 4 | 0 |

D

|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 4 | 0 | 4 |
| 2 | 5 | 0 | 0 | 0 | 4 |
| 3 | 5 | 5 | 0 | 0 | 4 |
| 4 | 5 | 5 | 0 | 0 | 0 |
| 5 | 0 | 1 | 4 | 1 | 0 |

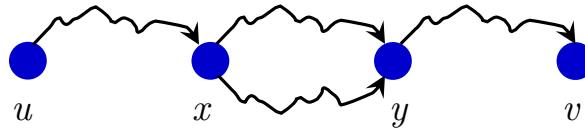
P

 $\text{PATH}(P, i, j)$ 
**if**  $P[i, j] \neq 0$  **then**
 $\text{PATH}(P, i, P[i, j])$ 
**print**  $P[i, j]$ 
 $\text{PATH}(P, P[i, j], j)$

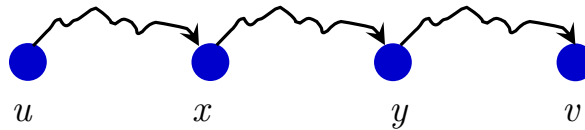
## اصل بهینگی

## اصل بهینگی:

راه حل بهینه برای هر نمونه از راه حل بهینه‌ی نمونه‌های کوچکتر حاصل می‌شود.



اگر این کوتاه‌ترین مسیر بین  $u$  و  $v$  باشد:

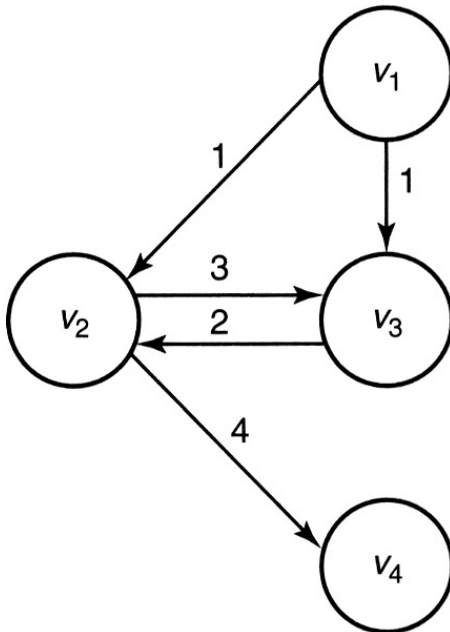


آن‌گاه این کوتاه‌ترین مسیر بین  $x$  و  $y$  است:



## مسئله‌ی طولانی‌ترین مسیر ساده و اصل بهینگی

اگر اصل بهینگی در مورد مسئله‌ای برقرار نباشد، نمی‌توان روش برنامه‌نویسی پویا را برای آن به کار برد.



In Figure the optimal (longest) simple path from  $v_1$  to  $v_4$  is  $[v_1, v_3, v_2, v_4]$ . However, the subpath  $[v_1, v_3]$  is not an optimal (longest) path from  $v_1$  to  $v_3$  because

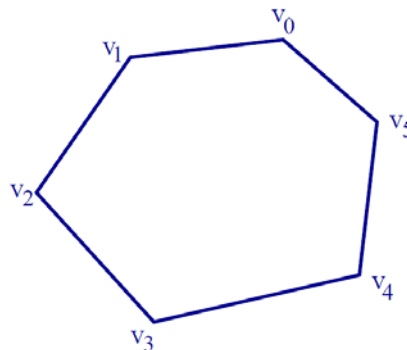
$$\text{length}[v_1, v_3] = 1 \quad \text{and} \quad \text{length}[v_1, v_2, v_3] = 4.$$

Therefore, the principle of optimality does not apply. The reason for this is that the optimal paths from  $v_1$  to  $v_3$  and from  $v_3$  to  $v_4$  cannot be strung together to give an optimal path from  $v_1$  to  $v_4$ .

## مثلتبندی بهینه‌ی یک چندضلعی محدب

CONVEX OPTIMAL POLYGON TRIANGULATION

- چندضلعی: یک منحنی بسته متشکل از پاره‌خطها (اضلاع)
- رأس: محل برخورد دو ضلع
- چندضلعی محدب: یک چندضلعی که پاره‌خط مابین هر دو نقطه‌ی دلخواه آن کاملاً درون همان چندضلعی قرار گیرد.
- ترتیب رئوس را بر خلاف جهت حرکت عقربه‌های ساعت می‌گیریم:
- **Edges are  $v_i v_{i+1}$ , where  $v_n = v_0$**
- قطر: پاره خط اتصال‌دهنده‌ی دو رأس غیرمجاور

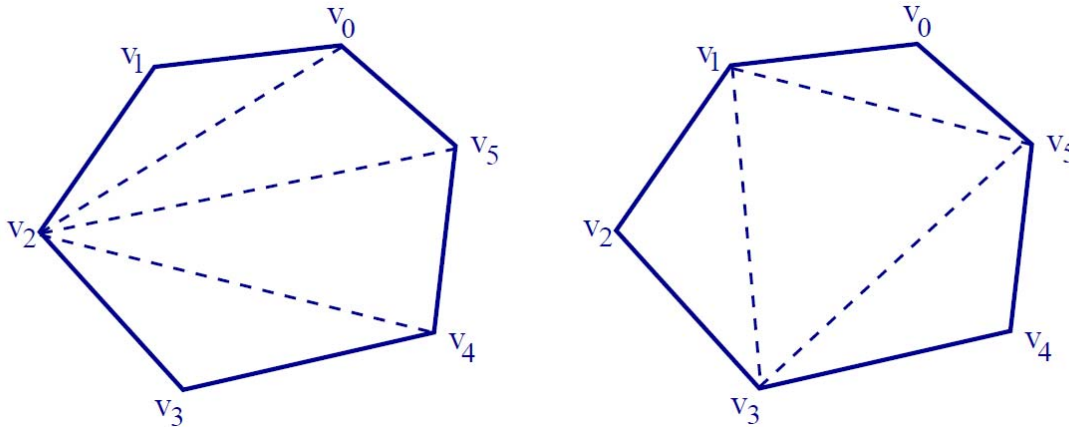


## مثلث‌بندی بهینه‌ی یک چندضلعی محدب

## CONVEX OPTIMAL POLYGON TRIANGULATION

## مثلث‌بندی:

تعیین مجموعه‌ای از قطر‌ها که چندضلعی را به مثلث‌های ناهمپوشان تقسیم می‌کند.



$n$  رأس  
 $n-3$  قطر  
 $n-2$  مثلث

## مثلث‌بندی بهینه:

یک چندضلعی  $P$  و تابع وزن  $W$  داده شده است.  $P = (v_0, \dots, v_{n-1})$  می‌خواهیم یک مثلث‌بندی با می‌نیمم وزن مجموع را بیابیم.

## مثلث‌بندی بهینه‌ی یک چندضلعی محدب: تابع وزن

یک نمونه تابع وزن:

$$w(\Delta v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i|$$

## مثلث‌بندی بهینه‌ی یک چندضلعی محدب: تابع وزن

یک نمونه تابع وزن:  $w(\Delta v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i|$

$m[i, j]$  وزن بهینه‌ی مثلث‌سازی زیر چندضلعی  $(v_i, v_{i+1}, \dots, v_j)$

فرمول وزن یک مثلث‌بندی بهینه:

$$m[i, j] = m[i, k] + m[k, j] + w(\Delta v_i v_j v_k)$$

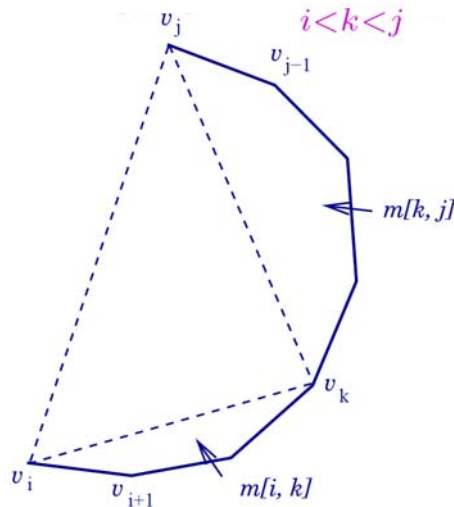


## مثلث‌بندی بهینه‌ی یک چندضلعی محدب

## فرمول بازگشتی

برای یافتن وزن بهینه‌ی مثلث‌سازی برای یک زیرچندضلعی  $(v_i, v_{i+1}, \dots, v_j)$

$$m[i, j] = \min_{i < k < j} \{m[i, k] + m[k, j] + w(\Delta v_i v_j v_k)\}$$



## مثلث‌بندی بهینه‌ی یک چندضلعی محدب: شبه‌کد

$$m[i, j] = \min_{i < k < j} \{m[i, k] + m[k, j] + w(\Delta v_i v_j v_k)\}$$

برای یافتن وزن بهینه‌ی مثلث‌سازی برای یک زیرچندضلعی  $(v_i, v_{i+1}, \dots, v_j)$

فرمول بازگشتی

TRIANGULATION( $P, n$ )

$m \leftarrow \text{array}[1..n, 1..n]$

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

$m[i, i+1] \leftarrow 0$

**for**  $d \leftarrow 1$  **to**  $n-1$  **do**

**for**  $i \leftarrow 1$  **to**  $n-d$  **do**

$j \leftarrow i + d$

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

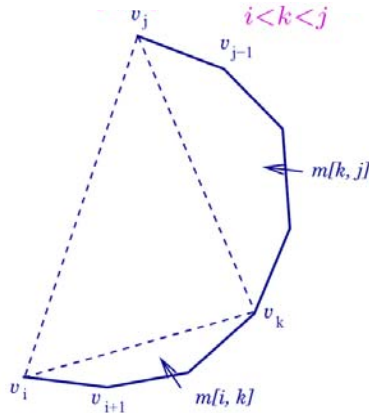
**for**  $k \leftarrow i+1$  **to**  $j-1$  **do**

$q \leftarrow m[i, k] + m[k, j] + w(P, [i, j, k])$

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

$m[i, j] \leftarrow q$

**return**  $m[1, n]$



$$T(n) \in \Theta(n^3)$$

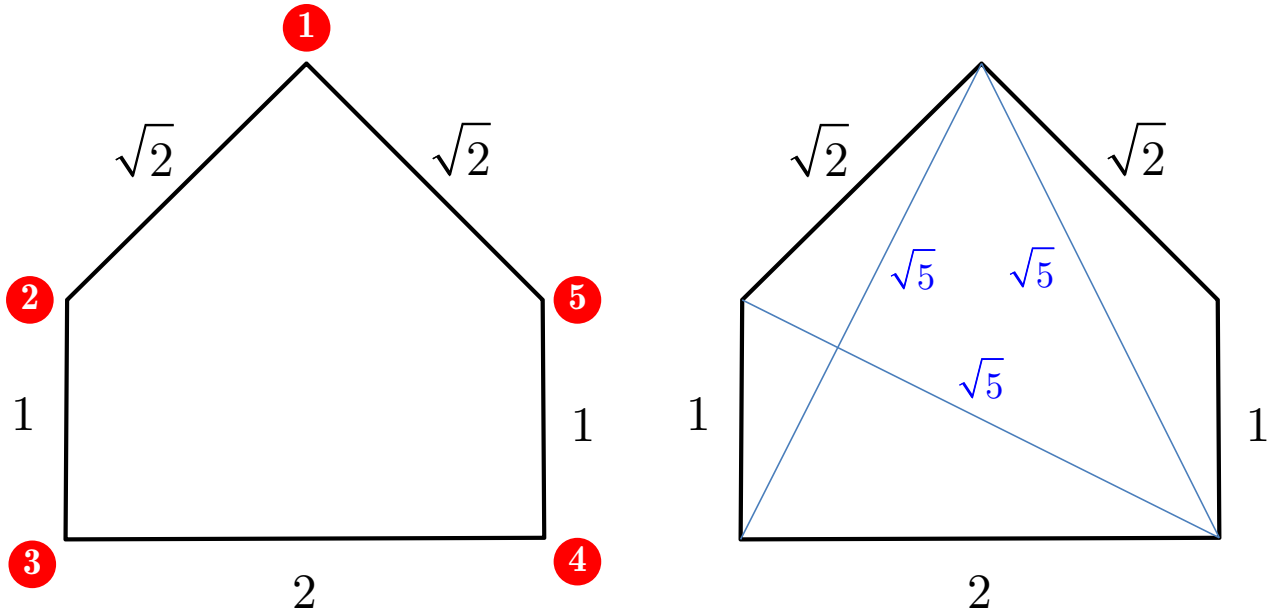
$$S(n) \in \Theta(n^2)$$

زمان اجرا:

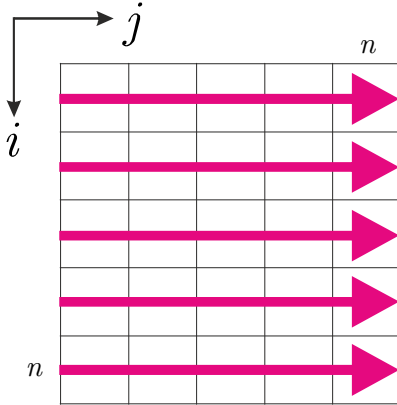
فضای مورد نیاز:

## مثث بندی بهینه ی یک چندضلعی محدب: مثال

مثث بندی با کمترین وزن را برای چندضلعی زیر بیابید:



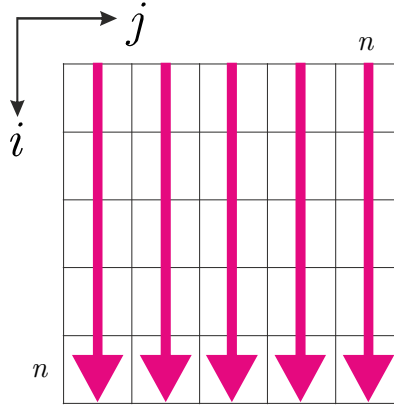
## ترتیب پر کردن جدول



سطری

Row-major

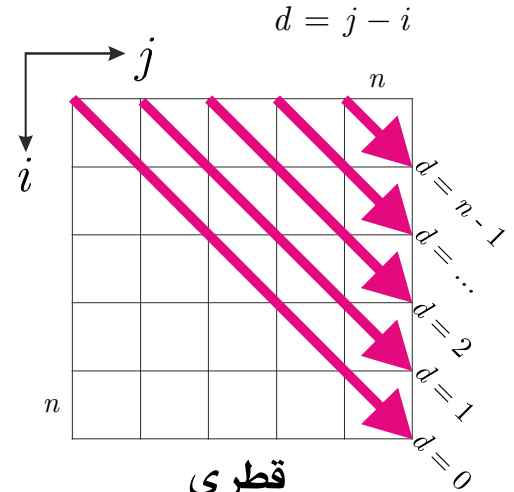
```
for i ← 1 to n do
  for j ← 1 to n do
```



ستونی

Column-major

```
for j ← 1 to n do
  for i ← 1 to n do
```



قطری

Diagonal-major

```
for d ← 0 to n-1 do
  for i ← 1 to n-d do
    j ← i + d
```

## ضرب زنجیره‌ای ماتریس‌ها

MATRIX CHAIN PRODUCT

فرض کنید می‌خواهیم حاصلضرب دو ماتریس زیر را به دست آوریم:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 & 9 & 1 \\ 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 29 & 35 & 41 & 38 \\ 74 & 89 & 104 & 83 \end{bmatrix}$$

برای یافتن عنصر سطر اول و ستون اول حاصلضرب داریم:

$$\underbrace{1 \times 7 + 2 \times 2 + 3 \times 6}_{3 \text{ multiplications}}$$

چون حاصلضرب  $2 \times 4$  عنصر دارد، مجموعاً  $2 \times 4 \times 3$  ضرب ساده لازم است:

$$2 \times 4 \times 3 = 24.$$

## ضرب زنجیره‌ای ماتریس‌ها

به طور کلی در حاصلضرب دو ماتریس با روش استاندارد

$$\begin{bmatrix} A & & \\ & \ddots & \\ & & \end{bmatrix}_{i \times j} \begin{bmatrix} B & & \\ & \ddots & \\ & & \end{bmatrix}_{j \times k}$$

$$i \times j \times k$$

ضرب ساده لازم است

## ضرب زنجیره‌ای ماتریس‌ها

اگر بیش‌تر از دو ماتریس برای ضرب داشته باشیم،  
 آن‌ها را می‌توان به حالت‌های مختلفی در هم ضرب کرد  
 (خاصیت شرکت‌پذیری ضرب ماتریسی).

$$A(B(CD))$$

$$(AB)(CD)$$

$$A((BC)D)$$

$$((AB)C)D$$

$$(A(BC))D$$

|               |          |               |          |                |          |               |
|---------------|----------|---------------|----------|----------------|----------|---------------|
| $A$           | $\times$ | $B$           | $\times$ | $C$            | $\times$ | $D$           |
| $20 \times 2$ |          | $2 \times 30$ |          | $30 \times 12$ |          | $12 \times 8$ |

کدام ترتیب کم‌ترین تعداد ضرب‌ها را دارد؟

## ضرب زنجیره‌ای ماتریس‌ها

$$\begin{array}{cccc}
 A & \times & B & \times & C & \times & D \\
 20 \times 2 & & 2 \times 30 & & 30 \times 12 & & 12 \times 8
 \end{array}$$

کدام ترتیب کم‌ترین تعداد ضرب‌ها را دارد؟

$$\begin{array}{ll}
 A(B(CD)) & 30 \times 12 \times 8 + 2 \times 30 \times 8 + 20 \times 2 \times 8 = 3680 \\
 (AB)(CD) & 20 \times 2 \times 30 + 30 \times 12 \times 8 + 20 \times 30 \times 8 = 8880 \\
 A((BC)D) & 2 \times 30 \times 12 + 2 \times 12 \times 8 + 20 \times 2 \times 8 = \boxed{1232} \\
 ((AB)C)D & 20 \times 2 \times 30 + 20 \times 30 \times 12 + 20 \times 12 \times 8 = 10320 \\
 (A(BC))D & 2 \times 30 \times 12 + 20 \times 2 \times 12 + 20 \times 12 \times 8 = 3120
 \end{array}$$

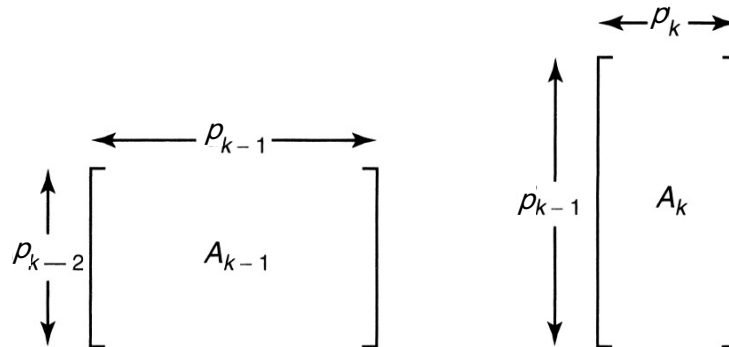


## ضرب زنجیره‌ای ماتریس‌ها

$$A_1 \times A_2 \times A_3 \times \cdots \times A_n$$

زنجیره‌ای از  $n$  ماتریس داریم که باید در هم ضرب شوند.  
**هدف:** یافتن ترتیبی از ماتریس‌ها که کمترین تعداد ضرب را داشته باشد.  
 ابعاد ماتریس‌ها در آرایه‌ی  $p$  داده می‌شود.

$$p = [p_0 \quad p_1 \quad p_2 \quad \cdots \quad p_n]$$



## ضرب زنجیره‌ای ماتریس‌ها

فرض می‌کنیم:

$m[i, j]$  حداقل تعداد ضرب‌ها در پرانتزگذاری بهینه‌ی ضرب  $A_i \dots A_j$  باشد.  
در این صورت  
 $m[1, n]$  جواب مسئله خواهد بود.

اگر جواب بهینه در نقطه‌ی  $k$  شکسته شود، داریم:

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

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

## ضرب زنجیره‌ای ماتریس‌ها

رابطه‌ی بازگشتی:

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

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

## ضرب زنجیره‌ای ماتریس‌ها

شبه‌ک

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

رابطه‌ی بازگشتی:

MATRIX-CHAIN-ORDER( $p$ ) $n \leftarrow \text{length}(p) - 1$  $m \leftarrow \text{array}[1..n, 1..n]$ **for**  $i \leftarrow 1$  **to**  $n$  **do** $m[i, i] \leftarrow 0$ **for**  $d \leftarrow 1$  **to**  $n - 1$  **do****for**  $i \leftarrow 1$  **to**  $n - d$  **do** $j \leftarrow i + d$  $m[i, j] \leftarrow \infty$ **for**  $k \leftarrow i$  **to**  $j - 1$  **do** $q \leftarrow m[i, k] + m[k + 1, j] + p[i - 1] * p[k] * p[j]$ **if**  $q < m[i, j]$  **then** $m[i, j] \leftarrow q$  $s[i, j] \leftarrow k$ **return**  $m[1, n]$ چاپ ترتیب بهینه PRINT-OPTIMAL-PARENS( $s, i, j$ )1 **if**  $i = j$ 2 **then** print " $A$ "; <sub>$i$</sub> 3 **else** print "("4 PRINT-OPTIMAL-PARENS( $s, i, s[i, j]$ )5 PRINT-OPTIMAL-PARENS( $s, s[i, j] + 1, j$ )

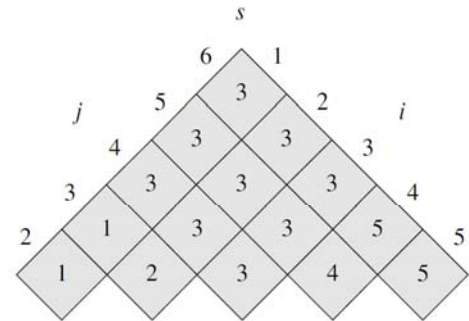
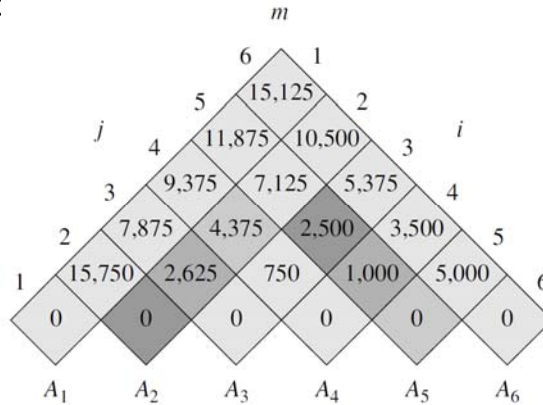
6 print ")"

زمان اجرا:  $T(n) \in \Theta(n^3)$ فضای مورد نیاز:  $S(n) \in \Theta(n^2)$

## ضرب زنجیره‌ای ماتریس‌ها

مثال

matrix    dimension

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

$$m[2, 5] = \min \begin{cases} m[2, 2] + m[3, 5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000, \\ m[2, 3] + m[4, 5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125, \\ m[2, 4] + m[5, 5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{cases}$$

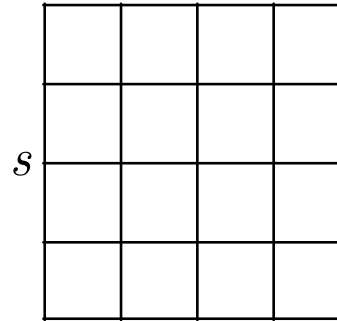
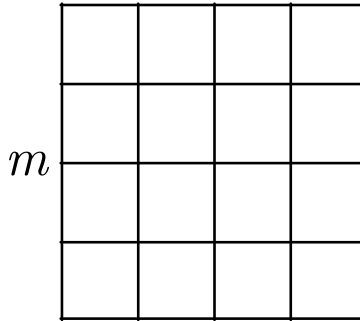
$$= 7125.$$

## ضرب زنجیره‌ای ماتریس‌ها

مثال

$$\begin{array}{cccc}
 A_1 & \times & A_2 & \times & A_3 & \times & A_4 \\
 100 \times 1 & & 1 \times 100 & & 100 \times 1 & & 1 \times 100
 \end{array}$$

$$p = [p_0 \quad p_1 \quad p_2 \quad p_3 \quad p_4] = [100 \quad 1 \quad 100 \quad 1 \quad 100]$$



## تکنیک به‌خاطر سپاری

### MEMOIZATION

### تکنیک به‌خاطر سپاری:

نوعی پیاده‌سازی برنامه‌ریزی پویا که از روش بالا به پایین بهره‌برداری می‌کند.

- ساختار کنترلی برای پر کردن جدول، مشابه روش بازگشتی است.
- هر زیرمسئله که حل شد، مقادیر آن در جدول ذخیره می‌شود.
- جدول در ابتدا با مقادیر اولیه، مقداردهی می‌شود.

## ضرب زنجیره‌ای ماتریس‌ها با تکنیک به‌خاطر سپاری

MEMOIZED-MATRIX-CHAIN( $p$ )

```

1   $n \leftarrow \text{length}[p] - 1$ 
2  for  $i \leftarrow 1$  to  $n$ 
3      do for  $j \leftarrow i$  to  $n$ 
4          do  $m[i, j] \leftarrow \infty$ 
5  return LOOKUP-CHAIN( $p, 1, n$ )

```

LOOKUP-CHAIN( $p, i, j$ )

```

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

```



## طولانی‌ترین زیر دنباله‌ی مشترک

LONGEST COMMON SUBSEQUENCE (LCS)

دنباله‌ی زیر را در نظر می‌گیریم:

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

یک زیر دنباله از یک دنباله، دنباله‌ی حاصل از حذف صفر یا بیشتر عنصر از آن است.

مثال:

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

$$Z_1 = \langle A, C, B, D, B \rangle \text{ زیر دنباله}$$

$$Z_2 = \langle A, B, D \rangle \text{ زیر دنباله}$$

$$Z_3 = \langle C, D, A \rangle \text{ زیر دنباله}$$

یک دنباله به طول  $n$  حداکثر  $2^n$  زیر دنباله دارد.

## طولانی‌ترین زیر دنباله‌ی مشترک

LONGEST COMMON SUBSEQUENCE (LCS)

دو دنباله‌ی زیر را در نظر می‌گیریم:

$$X = \langle x_1, x_2, \dots, x_m \rangle \quad Y = \langle y_1, y_2, \dots, y_n \rangle$$

یک زیر دنباله‌ی مشترک دو دنباله، دنباله‌ای است که زیر دنباله‌ی هر دو باشد.

مثال:

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

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

$$Z_1 = \langle B, C, A \rangle \text{ زیر دنباله‌ی مشترک}$$

$$Z_2 = \langle A, B \rangle \text{ زیر دنباله‌ی مشترک}$$

$$Z_3 = \langle B, D, A, B \rangle \text{ طولانی‌ترین زیر دنباله‌ی مشترک}$$

## طولانی‌ترین زیر دنباله‌ی مشترک

LONGEST COMMON SUBSEQUENCE (LCS)

دو دنباله‌ی زیر را در نظر می‌گیریم:

$$X = \langle x_1, x_2, \dots, x_m \rangle \quad Y = \langle y_1, y_2, \dots, y_n \rangle$$

هدف: یافتن طولانی‌ترین زیر دنباله‌ی مشترک دو دنباله است.

## طولانی‌ترین زیر دنباله‌ی مشترک: ساختار

LONGEST COMMON SUBSEQUENCE (LCS)

$X = \langle x_1, x_2, \dots, x_m \rangle$        $Y = \langle y_1, y_2, \dots, y_n \rangle$       دو دنباله‌ی

$X_i = \langle x_1, x_2, \dots, x_i \rangle$        $Y_i = \langle y_1, y_2, \dots, y_i \rangle$       اگر

$Z = \langle z_1, z_2, \dots, z_k \rangle$       و طولانی‌ترین زیر دنباله‌ی مشترک  $X$  و  $Y$ ، را  $Z$  بنامیم،

$Z_i = \langle z_1, z_2, \dots, z_i \rangle$

آن‌گاه:

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

## طولانی‌ترین زیر دنباله‌ی مشترک: فرمول بازگشتی

LONGEST COMMON SUBSEQUENCE (LCS)

تعریف می‌کنیم:

$$c[i, j] = |LCS(X_i, Y_j)|$$

در این صورت فرمول بازگشتی زیر از ساختار LCS به دست می‌آید:

$$c[i, j] = \left\{ \begin{array}{ll} 0 & \text{if } i \text{ or } j = 0 \\ c[i - 1, j - 1] + 1 & \text{if } x_i = y_j \\ \max\{c[i, j - 1], c[i - 1, j]\} & \text{if } x_i \neq y_j \end{array} \right\}$$

$$X = \langle x_1, x_2, \dots, x_m \rangle \quad Y = \langle y_1, y_2, \dots, y_n \rangle$$

## طولانی‌ترین زیر دنباله‌ی مشترک: شبه‌کد

LONGEST COMMON SUBSEQUENCE (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                  $b[i, j] \leftarrow \text{“}\searrow\text{”}$ 
12             else if  $c[i - 1, j] \geq c[i, j - 1]$ 
13                 then  $c[i, j] \leftarrow c[i - 1, j]$ 
14                      $b[i, j] \leftarrow \text{“}\uparrow\text{”}$ 
15             else  $c[i, j] \leftarrow c[i, j - 1]$ 
16                  $b[i, j] \leftarrow \text{“}\leftarrow\text{”}$ 
17  return  $c$  and  $b$ 

```

|                          |                 |
|--------------------------|-----------------|
| $T(m, n) \in \Theta(mn)$ | زمان اجرا:      |
| $S(m, n) \in \Theta(mn)$ | فضای مورد نیاز: |

## طولانی‌ترین زیر دنباله‌ی مشترک: شبه‌کد

چاپ طولانی‌ترین زیر دنباله‌ی مشترک

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

```

1  if  $i = 0$  or  $j = 0$ 
2    then return
3  if  $b[i, j] = "\searrow"$ 
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( $b, X, i, j - 1$ )

```

|     |       | $j$ | 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   | $A$   | 0   | ↑   | ↑   | ↑   | ↖   | 1   | ←   | 1 |
| 1   | $B$   | 0   | ↖   | 1   | ←   | 1   | ↑   | ↖   | 2 |
| 2   | $C$   | 0   | ↑   | ↑   | ↑   | ↖   | 2   | ←   | 2 |
| 3   | $A$   | 0   | 1   | 1   | ↑   | ↑   | ↑   | ↑   | ↑ |
| 4   | $B$   | 0   | ↖   | 1   | ↑   | ↑   | ↑   | ↖   | 3 |
| 5   | $D$   | 0   | ↑   | ↖   | 2   | 2   | 2   | ↑   | ↑ |
| 6   | $A$   | 0   | ↑   | ↑   | ↑   | ↑   | ↖   | 3   | ↖ |
| 7   | $B$   | 0   | ↖   | 1   | ↑   | ↑   | ↑   | ↖   | 4 |

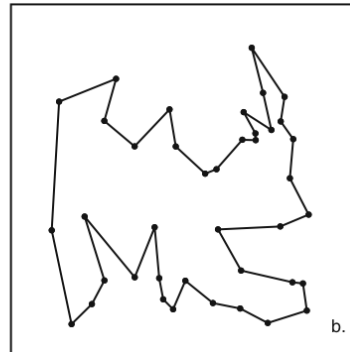
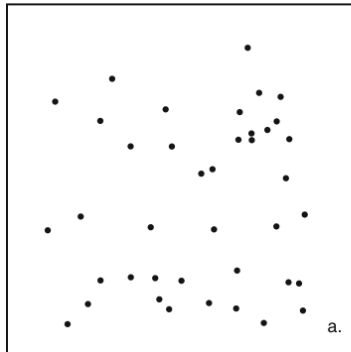
مثال

## مسئله‌ی فروشنده‌ی دوره‌گرد

تعریف مسئله

TRAVELLING SALESMAN PROBLEM (TSP)

یک فروشنده‌ی دوره‌گرد می‌خواهد برای فروش کالاهايش  $n$  شهر را یکی پس از دیگری ببیند و دوباره به شهر مبدأ بازگردد؛ با این شرط که طول مسیری که طی می‌کند، حداقل باشد.





## مسئله‌ی فروشنده‌ی دوره‌گرد

تعریف انتزاعی مسئله

TRAVELLING SALESMAN PROBLEM (TSP)یافتن تور بهینه در یک گراف کامل با  $n$  رأس

مسیری در گراف از یک رأس به خودش که از تمام رئوس گراف عبور کند.

**تور (دور همیلتونی)**  
*Tour (Hamiltonian Cycle)*

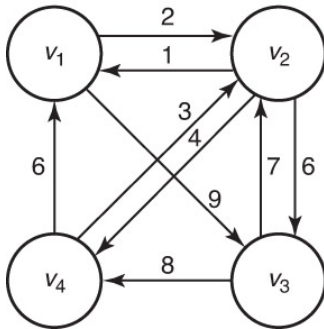
توری با کوتاه‌ترین طول (طول مسیر = مجموع وزن یال‌های مسیر)

**تور بهینه**  
*Optimal Tour*

## مسئله‌ی فروشنده‌ی دوره‌گرد

مثال

## TRAVELLING SALESMAN PROBLEM (TSP)



|   | 1        | 2 | 3        | 4        |
|---|----------|---|----------|----------|
| 1 | 0        | 2 | 9        | $\infty$ |
| 2 | 1        | 0 | 6        | 4        |
| 3 | $\infty$ | 7 | 0        | 8        |
| 4 | 6        | 3 | $\infty$ | 0        |

$$\text{length}[v_1, v_2, v_3, v_4, v_1] = 22$$

$$\text{length}[v_1, v_3, v_2, v_4, v_1] = 26$$

$$\text{length}[v_1, v_3, v_4, v_2, v_1] = 21$$

$$\Downarrow$$

$$[v_1, v_3, v_4, v_2, v_1] \quad \text{تور بهینه:}$$

## مسئله‌ی فروشنده‌ی دوره‌گرد

الگوریتم ناشیانه

TRAVELLING SALESMAN PROBLEM (TSP)یافتن تور بهینه در یک گراف کامل با  $n$  رأس

تمام تورهای یک گراف کامل با  $n$  رأس را بررسی می‌کنیم و کوتاه‌ترین آنها را برمی‌گردانیم.

زمان اجرا: متناسب با تعداد تورهای یک گراف کامل با  $n$  رأس

$$(n - 1)(n - 2) \cdots (1) = (n - 1)! \in \Theta(n!)$$

## مسئله‌ی فروشنده‌ی دوره‌گرد

اصل بهینگی

TRAVELLING SALESMAN PROBLEM (TSP)یافتن تور بهینه در یک گراف کامل با  $n$  رأس

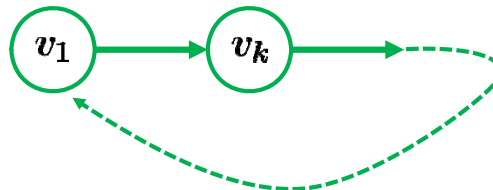
اگر  $v_k$  نخستین رأس پس از  $v_1$  روی هر تور بهینه باشد،  
 زیرمسیر آن تور از  $v_k$  به  $v_1$ ،  
 کوتاه‌ترین مسیر از  $v_k$  به  $v_1$  است  
 که از هر کدام از رئوس دیگر فقط یک بار می‌گذرد.



اصل بهینگی برقرار است.



می‌توان از روش برنامه‌ریزی پویا برای حل مسئله استفاده کرد.



## مسئله‌ی فروشنده‌ی دوره‌گرد

فرمول‌بندی مسئله

TRAVELLING SALESMAN PROBLEM (TSP)

|                 |   |             |
|-----------------|---|-------------|
|                 | مجموعه‌ی رأس‌های گراف   | $V$         |
| $A \subseteq V$ | یک زیرمجموعه از رأس‌های گراف  | $A$         |
|                 | طول کوتاه‌ترین مسیر از $v_i$ به $v_1$ که از هر رأس $A$ دقیقاً یک بار می‌گذرد. | $D[v_i, A]$ |

$$D[v_i, A] = \min_{v_j \in A} (W[i, j] + D[v_j, A - \{v_j\}]), \quad A \neq \emptyset, i \neq 1, v_i \notin A$$

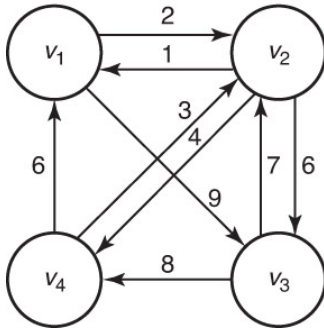
$$D[v_i, \emptyset] = W[i, 1]$$

$$\text{minimum tour length} = D[v_1, V - \{v_1\}]$$

## مسئله‌ی فروشنده‌ی دوره‌گرد

فرمول‌بندی مسئله: مثال

## TRAVELLING SALESMAN PROBLEM (TSP)



$$V = \{v_1, v_2, v_3, v_4\}$$

|   | 1        | 2 | 3        | 4        |
|---|----------|---|----------|----------|
| 1 | 0        | 2 | 9        | $\infty$ |
| 2 | 1        | 0 | 6        | 4        |
| 3 | $\infty$ | 7 | 0        | 8        |
| 4 | 6        | 3 | $\infty$ | 0        |

$$A = \{v_3\}$$

$$\Downarrow$$

$$\begin{aligned} D[v_2, A] &= \text{length}[v_2, v_3, v_1] \\ &= \infty \end{aligned}$$

$$A = \{v_3, v_4\}$$

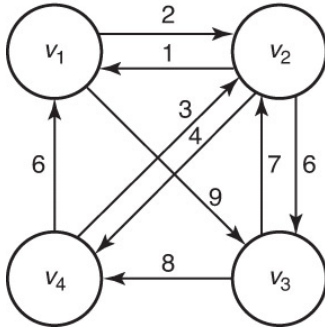
$$\Downarrow$$

$$\begin{aligned} D[v_2, A] &= \min \left\{ \begin{array}{l} \text{length}[v_2, v_3, v_4, v_1] \\ \text{length}[v_2, v_4, v_3, v_1] \end{array} \right\} \\ &= \min(20, \infty) \\ &= 20 \end{aligned}$$

## مسئله‌ی فروشنده‌ی دوره‌گرد

مثال (۱ از ۴)

## TRAVELLING SALESMAN PROBLEM (TSP)

ابتدا برای زیرمجموعه‌های  $k = 0$  عضوی از رئوس (غیر از مبدأ)

$$D[v_2, \emptyset] = 1$$

$$D[v_3, \emptyset] = \infty$$

$$D[v_4, \emptyset] = 6$$

سپس برای زیرمجموعه‌های  $k = 1$  عضوی از رئوس (غیر از مبدأ)

$$\begin{aligned} D[v_3, \{v_2\}] &= \min_{j:v_j \in \{v_2\}} (W[3, j] + D[v_j, \{v_2\} - \{v_j\}]) \\ &= W[3, 2] + D[v_2, \emptyset] \\ &= 7 + 1 = 8 \end{aligned}$$

به طور مشابه:

$$D[v_4, \{v_2\}] = 3 + 1 = 4$$

$$D[v_2, \{v_3\}] = 6 + \infty = \infty$$

$$D[v_4, \{v_3\}] = \infty + \infty = \infty$$

$$D[v_2, \{v_4\}] = 4 + 6 = 10$$

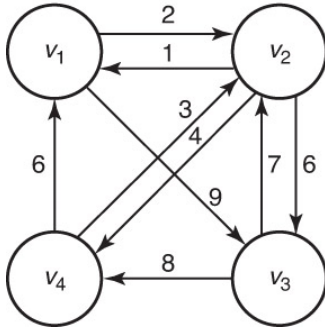
$$D[v_3, \{v_4\}] = 8 + 6 = 14$$

|   | 1        | 2 | 3        | 4        |
|---|----------|---|----------|----------|
| 1 | 0        | 2 | 9        | $\infty$ |
| 2 | 1        | 0 | 6        | 4        |
| 3 | $\infty$ | 7 | 0        | 8        |
| 4 | 6        | 3 | $\infty$ | 0        |

## مسئله فروشندهی دوره‌گرد

مثال (۲ از ۴)

## TRAVELLING SALESMAN PROBLEM (TSP)



سپس برای زیرمجموعه‌های  $k = 2$  عضو از رئوس (غیر از مبدأ)

$$\begin{aligned}
 D[v_4, \{v_2, v_3\}] &= \min_{j: v_j \in \{v_2, v_3\}} (W[4, j] + D[v_j, \{v_2, v_3\} - \{v_j\}]) \\
 &= \min \left\{ \begin{array}{l} W[4, 2] + D[v_2, \{v_3\}] \\ W[4, 3] + D[v_3, \{v_2\}] \end{array} \right\} \\
 &= \min(3 + \infty, \infty + 8) = \infty
 \end{aligned}$$

به طور مشابه:

$$\begin{aligned}
 D[v_3, \{v_2, v_4\}] &= \min(7 + 10, 8 + 4) = 12 \\
 D[v_2, \{v_3, v_4\}] &= \min(6 + 14, 4 + \infty) = 20
 \end{aligned}$$

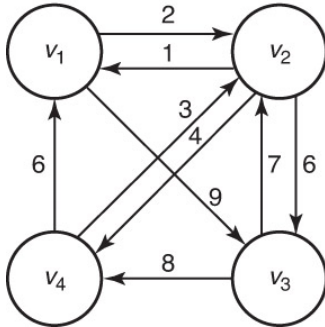
|   | 1        | 2 | 3        | 4        |
|---|----------|---|----------|----------|
| 1 | 0        | 2 | 9        | $\infty$ |
| 2 | 1        | 0 | 6        | 4        |
| 3 | $\infty$ | 7 | 0        | 8        |
| 4 | 6        | 3 | $\infty$ | 0        |



## مسئله فروشندهی دوره‌گرد

مثال (۳ از ۴)

## TRAVELLING SALESMAN PROBLEM (TSP)



سپس برای زیرمجموعه‌های  $k = 3$  عضوی از رئوس (غیر از مبدأ)

$$\begin{aligned}
 D[v_1, \{v_2, v_3, v_4\}] &= \min_{j: v_j \in \{v_2, v_3, v_4\}} (W[1, j] + D[v_j, \{v_2, v_3, v_4\} - \{v_j\}]) \\
 &= \min \left\{ \begin{array}{l} W[1, 2] + D[v_2, \{v_3, v_4\}] \\ W[1, 3] + D[v_3, \{v_2, v_4\}] \\ W[1, 4] + D[v_4, \{v_2, v_3\}] \end{array} \right\} \\
 &= \min(2 + 20, 9 + 12, \infty + \infty) = 21 \quad \text{طول تور بهینه:}
 \end{aligned}$$

|   |          |   |          |          |
|---|----------|---|----------|----------|
|   | 1        | 2 | 3        | 4        |
| 1 | 0        | 2 | 9        | $\infty$ |
| 2 | 1        | 0 | 6        | 4        |
| 3 | $\infty$ | 7 | 0        | 8        |
| 4 | 6        | 3 | $\infty$ | 0        |

← ————— زیرمجموعه‌های  $V - \{v_1\}$  ————— →

$$D =$$

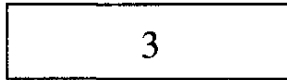
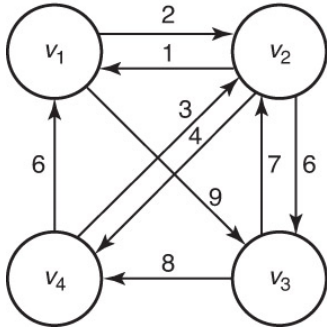
|       | $\{\emptyset\}$ | $\{v_2\}$ | $\{v_3\}$ | $\{v_4\}$ | $\{v_2, v_3\}$ | $\{v_2, v_4\}$ | $\{v_3, v_4\}$ | $\{v_2, v_3, v_4\}$ |
|-------|-----------------|-----------|-----------|-----------|----------------|----------------|----------------|---------------------|
| $v_1$ |                 |           |           |           |                |                |                | 21                  |
| $v_2$ | 1               |           | $\infty$  | 10        |                |                | 20             |                     |
| $v_3$ | $\infty$        | 8         |           | 14        |                | 12             |                |                     |
| $v_4$ | 6               | 4         | $\infty$  |           | $\infty$       |                |                |                     |

→ جهت پر شدن جدول

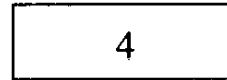
مسئله فروشندهی دوره‌گرد

مثال (۴ از ۴)

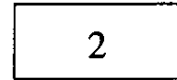
TRAVELLING SALESMAN PROBLEM (TSP)



$$P[1, \{v_2, v_3, v_4\}]$$



$$P[3, \{v_2, v_4\}]$$



$$P[4, \{v_2\}]$$

$$\begin{aligned} \text{اندیس اولین رأس میانی} &= P[1][\{v_2, v_3, v_4\}] = 3 \\ &\downarrow \\ \text{اندیس دومین رأس میانی} &= P[3][\{v_2, v_4\}] = 4 \\ &\downarrow \\ \text{اندیس سومین رأس میانی} &= P[4][\{v_2\}] = 2 \end{aligned}$$

تور بهینه:

$$[v_1, v_3, v_4, v_2, v_1].$$

$$D =$$

|                | {∅} | {v <sub>2</sub> } | {v <sub>3</sub> } | {v <sub>4</sub> } | {v <sub>2</sub> , v <sub>3</sub> } | {v <sub>2</sub> , v <sub>4</sub> } | {v <sub>3</sub> , v <sub>4</sub> } | {v <sub>2</sub> , v <sub>3</sub> , v <sub>4</sub> } |
|----------------|-----|-------------------|-------------------|-------------------|------------------------------------|------------------------------------|------------------------------------|---|
| v <sub>1</sub> |     |                   |                   |                   |                                    |                                    |                                    | 21  |
| v <sub>2</sub> | 1   |                   | ∞                 | 10                |                                    |                                    | 20                                 |   |
| v <sub>3</sub> | ∞   | 8                 |                   | 14                |                                    | 12                                 |                                    |   |
| v <sub>4</sub> | 6   | 4                 | ∞                 |                   | ∞                                  |                                    |                                    |   |

## مسئله‌ی فروشنده‌ی دوره‌گرد

شبه‌کد

TRAVELLING SALESMAN PROBLEM (TSP)TRAVELLING-SALESMAN-PROBLEM( $W, n$ ) $D \leftarrow \text{array}[1..n, \mathcal{P}(V - \{v_1\})]$  $P \leftarrow \text{array}[1..n, \mathcal{P}(V - \{v_1\})]$ **for**  $i \leftarrow 2$  **to**  $n$  **do** $D[i, \emptyset] \leftarrow W[i, 1]$ **for**  $k \leftarrow 1$  **to**  $n - 2$  **do****for** all subsets  $A \subseteq V - \{v_1\}$  containing  $k$  vertices **do****for**  $i$  such that  $i \neq 1$  and  $v_i \notin A$  **do** $D[i, A] \leftarrow \min_{v_j \in A} (W[i, j] + D[v_j, A - \{v_j\}])$  $P[i, A] \leftarrow \text{value of } j \text{ that gave the minimum}$  $D[v_1, V - \{v_1\}] \leftarrow \min_{2 \leq j \leq n} (W[1, j] + D[v_j, V - \{v_1\}])$  $P[v_1, V - \{v_1\}] \leftarrow \text{value of } j \text{ that gave the minimum}$  $\text{minlength} \leftarrow D[v_1, V - \{v_1\}]$ **return**  $\text{minlength}$

## مسئله‌ی فروشنده‌ی دوره‌گرد

## تحلیل زمان اجرا

## TRAVELLING SALESMAN PROBLEM (TSP)

TRAVELLING-SALESMAN-PROBLEM( $W, n$ ) $D \leftarrow \text{array}[1..n, \mathcal{P}(V - \{v_1\})]$  $P \leftarrow \text{array}[1..n, \mathcal{P}(V - \{v_1\})]$ **for**  $i \leftarrow 2$  **to**  $n$  **do** $D[i, \emptyset] \leftarrow W[i, 1]$ **for**  $k \leftarrow 1$  **to**  $n - 2$  **do****for all** subsets  $A \subseteq V - \{v_1\}$  containing  $k$  vertices **do****for**  $i$  such that  $i \neq 1$  and  $v_i \notin A$  **do** $D[i, A] \leftarrow \min_{v_j \in A} (W[i, j] + D[v_j, A - \{v_j\}])$  $P[i, A] \leftarrow \text{value of } j \text{ that gave the minimum}$  $D[v_1, V - \{v_1\}] \leftarrow \min_{2 \leq j \leq n} (W[1, j] + D[v_j, V - \{v_1\}])$  $P[v_1, V - \{v_1\}] \leftarrow \text{value of } j \text{ that gave the minimum}$  $\text{minlength} \leftarrow D[v_1, V - \{v_1\}]$ **return**  $\text{minlength}$ 

حلقه اول

یک مجموع برای  $k$

$$\sum_{k=1}^{n-2} \dots$$

زمان مصرفی اصلی

حلقه دوم

به تعداد زیرمجموعه‌های شامل  $k$  رأس:

$$\binom{n-1}{k}$$

حلقه سوم

بررسی  $n - 1 - k$  رأس  
برای هر کدام از این رأس‌ها  $k$  عمل جمع

برحسب عمل اصلی: جمع

## مسئله‌ی فروشنده‌ی دوره‌گرد

تحلیل زمان اجرا: ادامه

## TRAVELLING SALESMAN PROBLEM (TSP)

TRAVELLING-SALESMAN-PROBLEM( $W, n$ ) $D \leftarrow \text{array}[1..n, \mathcal{P}(V - \{v_1\})]$  $P \leftarrow \text{array}[1..n, \mathcal{P}(V - \{v_1\})]$ **for**  $i \leftarrow 2$  **to**  $n$  **do** $D[i, \emptyset] \leftarrow W[i, 1]$ 

زمان مصرفی اصلی

**for**  $k \leftarrow 1$  **to**  $n - 2$  **do****for all** subsets  $A \subseteq V - \{v_1\}$  containing  $k$  vertices **do**  
**for**  $i$  such that  $i \neq 1$  and  $v_i \notin A$  **do** $D[i, A] \leftarrow \min_{v_j \in A} (W[i, j] + D[v_j, A - \{v_j\}])$  $P[i, A] \leftarrow \text{value of } j \text{ that gave the minimum}$  $D[v_1, V - \{v_1\}] \leftarrow \min_{2 \leq j \leq n} (W[1, j] + D[v_j, V - \{v_1\}])$  $P[v_1, V - \{v_1\}] \leftarrow \text{value of } j \text{ that gave the minimum}$  $\text{minlength} \leftarrow D[v_1, V - \{v_1\}]$ **return**  $\text{minlength}$ 

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

$$\Rightarrow T(n) \in \Theta(n^2 2^n)$$

دو فرمول مورد استفاده:

$$(n-1-k) \binom{n-1}{k} = (n-1) \binom{n-2}{k}$$

$$\sum_{k=1}^n k \binom{n}{k} = n 2^{n-1}$$

## مسئله‌ی فروشنده‌ی دوره‌گرد

تحلیل فضای مصرفی

## TRAVELLING SALESMAN PROBLEM (TSP)

TRAVELLING-SALESMAN-PROBLEM( $W, n$ )

```

D ← array[1..n, P(V - {v1})]
P ← array[1..n, P(V - {v1})]

```

فضای مصرفی اصلی

```

for i ← 2 to n do

```

```

    D[i, ∅] ← W[i, 1]

```

```

for k ← 1 to n - 2 do

```

```

    for all subsets A ⊆ V - {v1} containing k vertices do
        for i such that i ≠ 1 and vi ∉ A do

```

```

            D[i, A] ← minvj ∈ A (W[i, j] + D[vj, A - {vj}] )

```

```

            P[i, A] ← value of j that gave the minimum

```

```

D[v1, V - {v1}] ← min2 ≤ j ≤ n (W[1, j] + D[vj, V - {v1}] )

```

```

P[v1, V - {v1}] ← value of j that gave the minimum

```

```

minlength ← D[v1, V - {v1}]

```

```

return minlength

```

مجموعه‌ی  $V - \{v_1\}$  دارای  $n - 1$  عضو و  $2^{n-1}$  زیرمجموعه است.

$$\begin{aligned}
 S(n) &= 2 \times n 2^{n-1} \\
 &= n 2^n
 \end{aligned}$$

$$\Rightarrow S(n) \in \Theta(n 2^n)$$

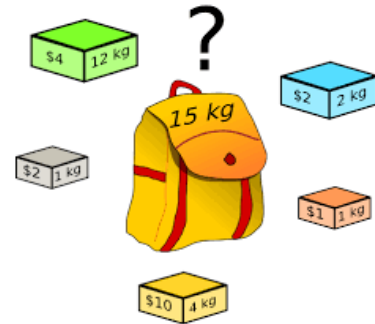
## مسئله‌ی کوله‌پشتی 0/1

0/1- KNAPSACK

$n$  قطعه با وزن‌ها و ارزش‌های مختلف داریم.  
یک کوله‌پشتی با ظرفیت  $W$  موجود است.

می‌خواهیم قطعه‌هایی را انتخاب کنیم که

- کوله‌پشتی تا حد امکان پر شود، و
- مجموع ارزش کل ظرف ماکزیمم شود.



## مسئله‌ی کوله‌پشتی 0/1

0/1-KNAPSACK

$$S = \{item_1, item_2, \dots, item_n\}$$

$$w_i = weight(item_i)$$

$$p_i = value(item_i)$$

$W$  = total weight of knapsack

select  $A \subseteq S$

maximize  $\sum_{item_i \in A} p_i x_i$

subject to  $\sum_{item_i \in A} w_i x_i \leq W$

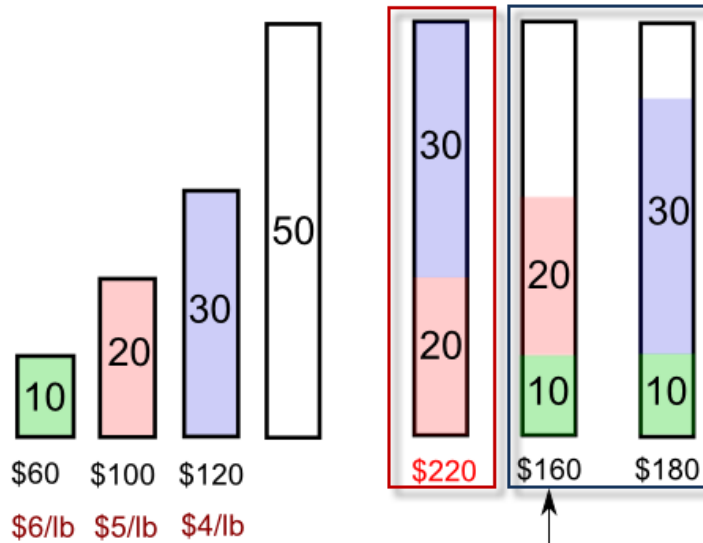
where  $x_i \in \{0, 1\}$ ,  $1 \leq i \leq n$



# مسئله کوله پشتی 0/1

مثال

## 0/1- KNAPSACK



Greedy algorithm does not work for 0-1 knapsack problem

## مسئله‌ی کوله‌پشتی 0/1

الگوریتم ناشیانه

### 0/1-KNAPSACK

## در مسئله‌ی کوله‌پشتی 0/1

همه‌ی زیرمجموعه‌های  $n$  قطعه را بررسی می‌کنیم  
و آنکه بیشترین منفعت را دارد (بدون تجاوز از قید حداکثر وزن کوله‌پشتی) برمی‌گردانیم.

زمان اجرا: متناسب با تعداد زیرمجموعه‌های یک مجموعه‌ی  $n$  عضوی

$$\Theta(2^n)$$

## مسئله‌ی کوله‌پشتی 0/1

اصل بهینگی

0/1-KNAPSACK

## در مسئله‌ی کوله‌پشتی 0/1

فرض می‌کنیم  $A$  یک زیرمجموعه‌ی بهینه از  $n$  قطعه باشد.  
دو حالت وجود دارد:

- $item_n \notin A$ : در این صورت  $A$  مساوی است با زیرمجموعه‌ای بهینه از  $n - 1$  قطعه‌ی نخست.
- $item_n \in A$ : در این صورت منفعت کل حاصل از قطعات موجود در  $A$  مساوی است

با ارزش قطعه‌ی  $n$  به اضافه‌ی منفعت بهینه‌ای است که از زیرمجموعه‌ای از  $n - 1$  قطعه‌ی نخست به دست می‌آید (با رعایت این قید که وزن کل آنها نباید از  $W - w_n$  بیشتر شود).



اصل بهینگی برقرار است.



می‌توان از روش برنامه‌ریزی پویا برای حل مسئله استفاده کرد.

## مسئله‌ی کوله‌پشتی 0/1

فرمول‌بندی مسئله

## 0/1-KNAPSACK

|   |           |
|---|-----------|
| حداکثر گنجایش وزنی کوله‌پشتی  | $W$       |
| تعداد قطعه‌ها   | $n$       |
| ارزش قطعه $i$ ام  | $p_i$     |
| وزن قطعه $i$ ام   | $w_i$     |
| منفعت بهینه‌ی حاصل از انتخاب $i$ قطعه‌ی نخست (با رعایت قید عدم تجاوز وزن کل آنها از $W$ ) | $P[i, w]$ |

$$i > 0, \quad w > 0$$

$$P[i, w] = \begin{cases} \max(P[i-1, w], p_i + P[i-1, w-w_i]) & , w_i \leq w \\ P[i-1, w] & , w_i > w \end{cases}$$

$$P[i, 0] = P[0, w] = 0$$

$$\text{حداکثر منفعت} = P[n, W]$$

$$P[0..n, 0..W]$$

## مسئله‌ی کوله‌پشتی 0/1

شبه‌کد

0/1-KNAPSACK

$$P[i, w] = \begin{cases} \max(P[i-1, w], p_i + P[i-1, w - w_i]) & , w_i \leq w \\ P[i-1, w] & , w_i > w \end{cases}$$

$$P[i, 0] = P[0, w] = 0$$

0/1-KNAPSACK( $p, w, W$ ) $n \leftarrow \text{length}(p)$  $P \leftarrow \text{array}[0..n, 0..W]$ **for**  $w \leftarrow 0$  **to**  $W$  **do** $P[0, w] \leftarrow 0$ **for**  $i \leftarrow 0$  **to**  $n$  **do** $P[i, 0] \leftarrow 0$ **for**  $i \leftarrow 1$  **to**  $n$  **do****for**  $w \leftarrow W$  **downto**  $w_i$  **do****if**  $P[i-1, w - w_i] + p_i > P[i-1, w]$  **then** $P[i, w] \leftarrow P[i-1, w - w_i] + p_i$ **else**  $P[i, w] \leftarrow P[i-1, w]$ **return**  $P[n, W]$ حداکثر منفعت =  $P[n, W]$

## مسئله‌ی کوله‌پشتی 0/1

تحلیل زمان اجرا

0/1-KNAPSACK

```

0/1-KNAPSACK( $p, w, W$ )
   $n \leftarrow \text{length}(p)$ 
   $P \leftarrow \text{array}[0..n, 0..W]$ 
  for  $w \leftarrow 0$  to  $W$  do
     $P[0, w] \leftarrow 0$ 
  for  $i \leftarrow 0$  to  $n$  do
     $P[i, 0] \leftarrow 0$ 
  for  $i \leftarrow 1$  to  $n$  do
    for  $w \leftarrow W$  downto  $w_i$  do
      if  $P[i - 1, w - w_i] + p_i > P[i - 1, w]$  then
         $P[i, w] \leftarrow P[i - 1, w - w_i] + p_i$ 
      else  $P[i, w] \leftarrow P[i - 1, w]$ 
  return  $P[n, W]$ 

```

تعداد عناصری که محاسبه می‌شوند:  $nW \in \Theta(nW)$

چون رابطه‌ای بین  $n$  و  $W$  وجود ندارد، نمی‌توان فرض کرد که این فرمول خطی است.  
 (برای یک  $n$  خاص می‌توان  $W$  به دلخواه بزرگ داشت، مثلاً:  $W = n!$  که در این صورت زمان الگوریتم حاصل از الگوریتم ناشیانه بدتر است.)