# برنامه‌ریزی پویا

سید ناصر رضوی n.razavi@tabrizu.ac.ir

۱۳۹۵

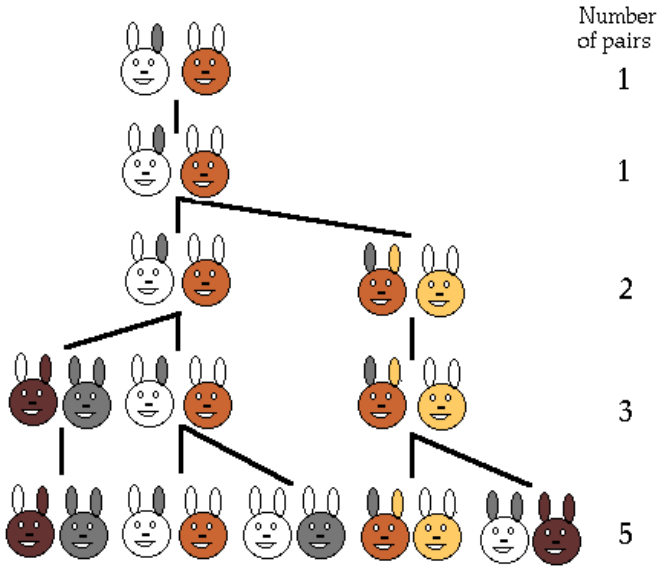
# فهرست مطالب

# اعداد فیبوناچی



«فرض کنیم خرگوش‌هایی وجود دارند که هر جفت (یک نر و یک ماده) از آنها که به سن یک ماهگی رسیده باشند به ازای هر ماه که از زندگی‌شان سپری شود یک جفت خرگوش به دنیا می‌آورند که آنها هم از همین قاعده پیروی می‌کنند. حال اگر فرض کنیم این خرگوشها هرگز نمی‌میرند و در آغاز یک جفت از این نوع خرگوش در اختیار داشته باشیم که به تازگی متولد شده‌اند، حساب کنید پس از **n** ماه چند جفت از این خرگوش‌ها خواهیم داشت.»

لئوناردو فیبوناچی
۱۲۵۰ – ۱۱۷۰



# اعداد فیبوناچی



- اعداد فیبوناچی.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n \geq 2 \end{cases}$$

```
public static long F(int n)
{
   if (n == 0) return 0;
   if (n == 1) return 1;
   return F(n - 1) + F(n - 2);
}
```

# اعداد فیبوناچی

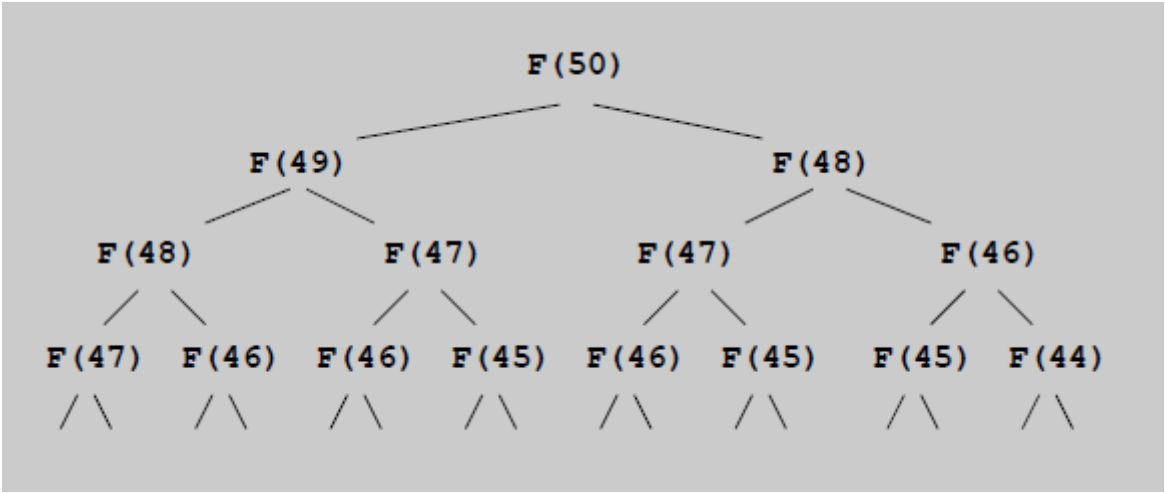


- س. آیا راه‌حل بازگشتی برای محاسبه‌ی **F(50)** کارا است؟

```
public static long F(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    return F(n - 1) + F(n - 2);
}
```

- ج. خیر، این کد به طرز شگفت‌انگیزی ناکارامد است.

**F(50)**: ۱ بار فراخوانی می شود
**F(49)**: ۱ بار فراخوانی می شود
**F(48)**: ۲ بار فراخوانی می شود
**F(47)**: ۳ بار فراخوانی می شود
**F(46)**: ۵ بار فراخوانی می شود
**F(45)**: ۸ بار فراخوانی می شود
...
**F(1)**: ۱۲۵۷۶۲۶۹۰۲۵ بار فراخوانی می شود



# اعداد فیبوناچی



- س. آیا روش سریع‌تری برای محاسبه‌ی **F(50)** وجود دارد؟
- ج. بله، کد زیر این کار را تنها با استفاده از ۴۹ عمل جمع انجام می‌دهد.

```
public static long F(int n)
{
    if (n == 0) return 0;
    long[] F = new long[n + 1];
    F[0] = 0;
    F[1] = 1;
    for (int i = 2; i <= n; i++)
        F[i] = f[i - 1] + F[i - 2];
    return F[n];
}
```

**F(5)**

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

- روش طراحی الگوریتم. روش بالا برای محاسبه‌ی اعداد فیبوناچی، روش برنامه‌ریزی پویا نام دارد.

# برنامه‌ریزی پویا: معرفی



- حوزه‌ها.
  - زیست رایانش
  - نظریه‌ی کنترل
  - نظریه‌ی اطلاعات
  - تحقیق در عملیات
  - علوم کامپیوتر: گرافیک، هوش مصنوعی، کامپایلرها و ...
- برخی از الگوریتم‌های مشهور.
  - الگوریتم به کار رفته در یونیکس برای مقایسه‌ی دو فایل
  - الگوریتم اسمیت – واترمن برای ترازبندی رشته‌های ژنتیکی
  - الگوریتم بلمن – فورد برای یافتن کوتاه‌ترین مسیرها
  - الگوریتم CYK برای تجزیه‌ی گرامرهای مستقل از متن

# علت ناکارآمدی الگوریتم‌های تقسیم و حل



- پس از تقسیم ...
  - نمونه‌های کوچک‌تر غیر مرتبط هستند [مرتب سازی ادغامی]
  - نمونه‌های کوچک‌تر مرتبط هستند [فیبوناچی]
    - حل زیرنمونه‌های مشترک به طور مکرر!

- برنامه‌ریزی پویا.
  - رویکرد پایین به بالا: حل نمونه‌ها از کوچک به بزرگ
  - ذخیره‌ی راه‌حل نمونه‌های کوچک‌تر حل شده در یک جدول

# ضریب دو جمله‌ای: روش اول



- ضریب دو جمله‌ای.

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

- محاسبه‌ی ضریب دو جمله‌ای. [روش اول]

بسیار بزرگ

$$\binom{100}{2} = \frac{100!}{2!\,(100-2)!} = 4950$$

- مشکل. نیاز به محاسبه‌ی اعداد بسیار بزرگ حتی زمانی که مقدار ضریب کوچک است.

# ضریب دو جمله‌ای: روش دوم



- ضریب دو جمله‌ای. [تعریف بازگشتی]

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

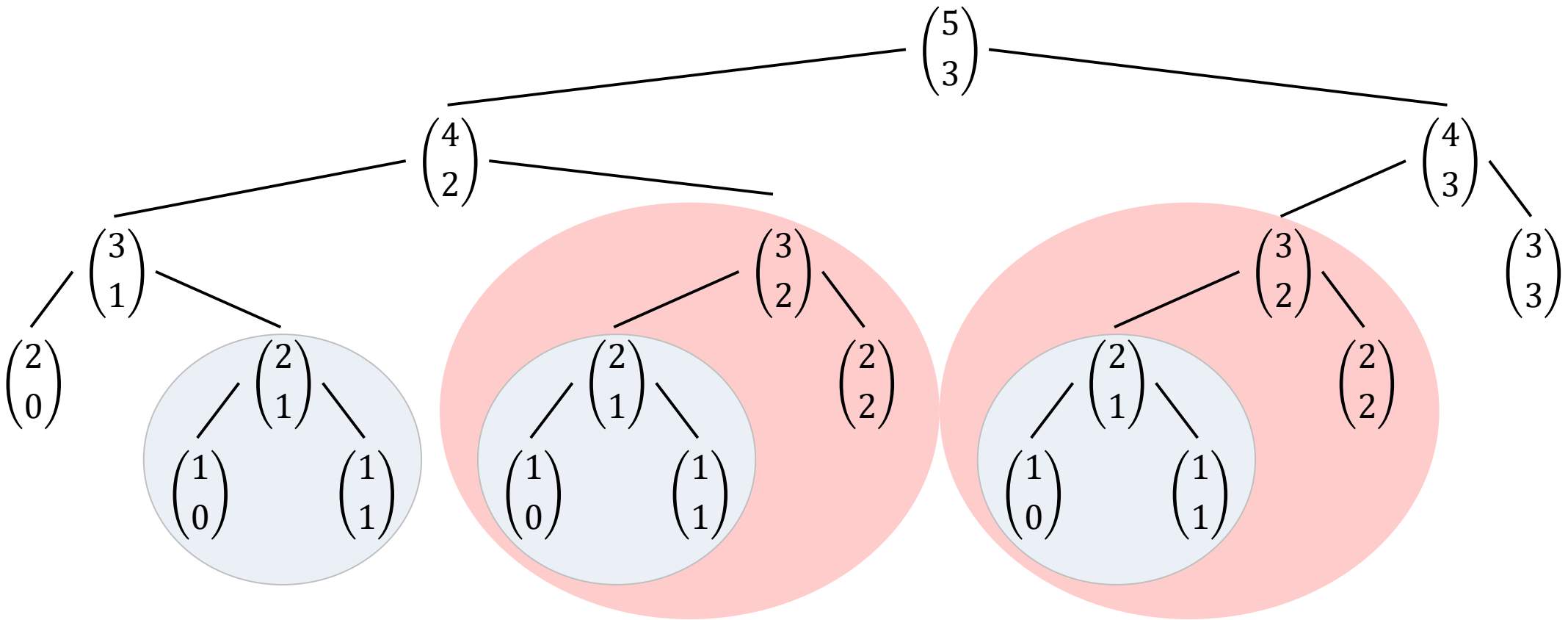
- محاسبه‌ی ضریب دو جمله‌ای. [الگوریتم بازگشتی]

```
public static int bin(int n, int k)
{
   if (k == 0 || k == n) return 1;
   return bin(n - 1, k - 1) + bin(n - 1, k);
}
```

# ضریب دوجمله ای: روش دوم



- مثال. محاسبه‌ی $\binom{5}{3}$



# ضریب دوجمله‌ای: روش سوم



- برنامه‌ریزی پویا.
  - استفاده از ماتریس B به منظور ذخیره‌ی راه‌حل نمونه‌ها.
- مراحل.
  - ارائه‌ی یک خاصیت بازگشتی به منظور محاسبه‌ی راه‌حل یک نمونه با توجه به راه‌حل نمونه‌های کوچک‌تر.

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

  - حل نمونه‌ها از پایین به بالا با استفاده از رابطه‌ی بازگشتی و ذخیره‌ی راه‌حل‌ها در ماتریس B

# ضریب دو جمله‌ای: روش سوم



- مثال. محاسبه‌ی $\binom{5}{3}$

| B | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 1 | | | |
| 1 | 1 | 1 | | |
| 2 | 1 | 2 | 1 | |
| 3 | 1 | 3 | 3 | 1 |
| 4 | 1 | 4 | 6 | 4 |
| 5 | 1 | 5 | 10 | 10 |

خروجی ←

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

# ضریب دو جمله‌ای: روش سوم



- محاسبه‌ی ضریب دو جمله‌ای.

```
public static int bin2(int n, int k)
{
   for (i = 0; i <= n; i++)
      for (j = 0; j <= Math.min(i, k); j++)
         if (j == 0 || j == i)
            B[i][j] = 1;
         else
            B[i][j] = B[i-1][j-1] + B[i-1][j];

   return B[n][k];
}
```

- پیچیدگی زمانی.

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

# برنامه‌ریزی پویا



- مراحل.
  - ارائه‌ی یک خاصیت بازگشتی به منظور حل یک نمونه با توجه به راه‌حل نمونه‌های کوچک‌تر؛
  - حل نمونه‌ها از پایین به بالا و ذخیره‌ی راه‌حل‌ها در یک جدول.

| B | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 1 | | | |
| 1 | 1 | 1 | | |
| 2 | 1 | 2 | 1 | |
| 3 | 1 | 3 | 3 | 1 |
| 4 | 1 | 4 | 6 | 4 |
| 5 | 1 | 5 | 10 | 10 |

# کوتاه‌ترین مسیرها بین همه‌ی زوج رئوس

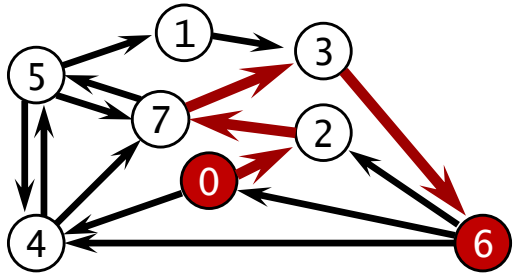
# کوتاه‌ترین مسیرها در گراف‌های جهت‌دار و وزن‌دار



- مسئله. با داشتن یک گراف جهت‌دار وزن‌دار، کوتاه‌ترین مسیر بین دو رأس **s** و **t** را پیدا کن.

گراف جهت‌دار وزن‌دار

```
4->5  0.35
5->4  0.35
4->7  0.37
5->7  0.28
7->5  0.28
5->1  0.32
0->4  0.38
0->2  0.26
7->3  0.39
1->3  0.29
2->7  0.34
6->2  0.40
3->6  0.52
6->0  0.58
6->4  0.93
```

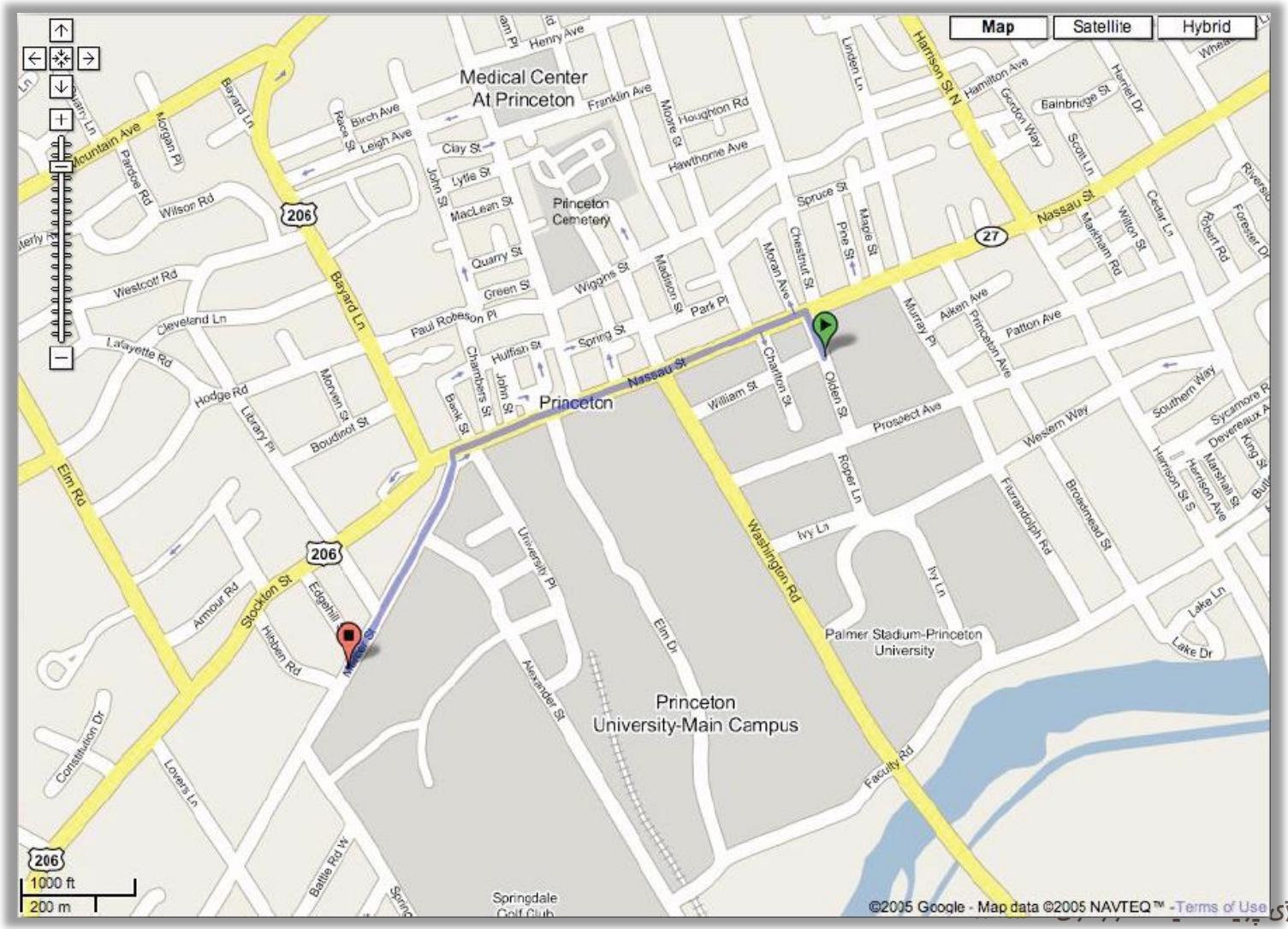



کوتاه‌ترین مسیر از 0 به 6

```
0->2  0.26
2->7  0.34
7->3  0.39
3->6  0.52
```

# نقشه‌های گوگل





# ناوبری خودرو: سامانه‌ی مکان‌یابی جهانی



# کاربردهای کوتاه‌ترین مسیر

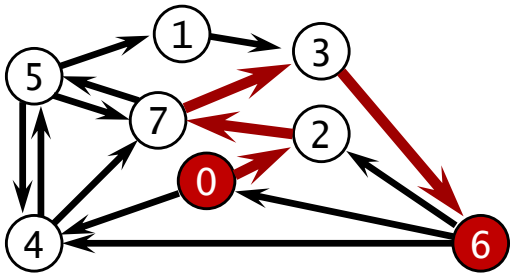


- ناوبری خودرو
- ناوبری روبات
- روش مسیر بحرانی
- مسیریابی در نقشه
- برنامه‌ریزی ترافیک شهری
- مسیریابی پیام‌های مخابراتی
- پروتکل‌های مسیریابی شبکه
- تغییر هوشمندانه‌ی اندازه تصویر
- سودجویی در معاملات ارزی
- و بسیاری از مسائل دیگر ...

http://en.wikipedia.org/wiki/Seam_carving

# گونه‌های مختلف مسئله کوتاه‌ترین مسیر



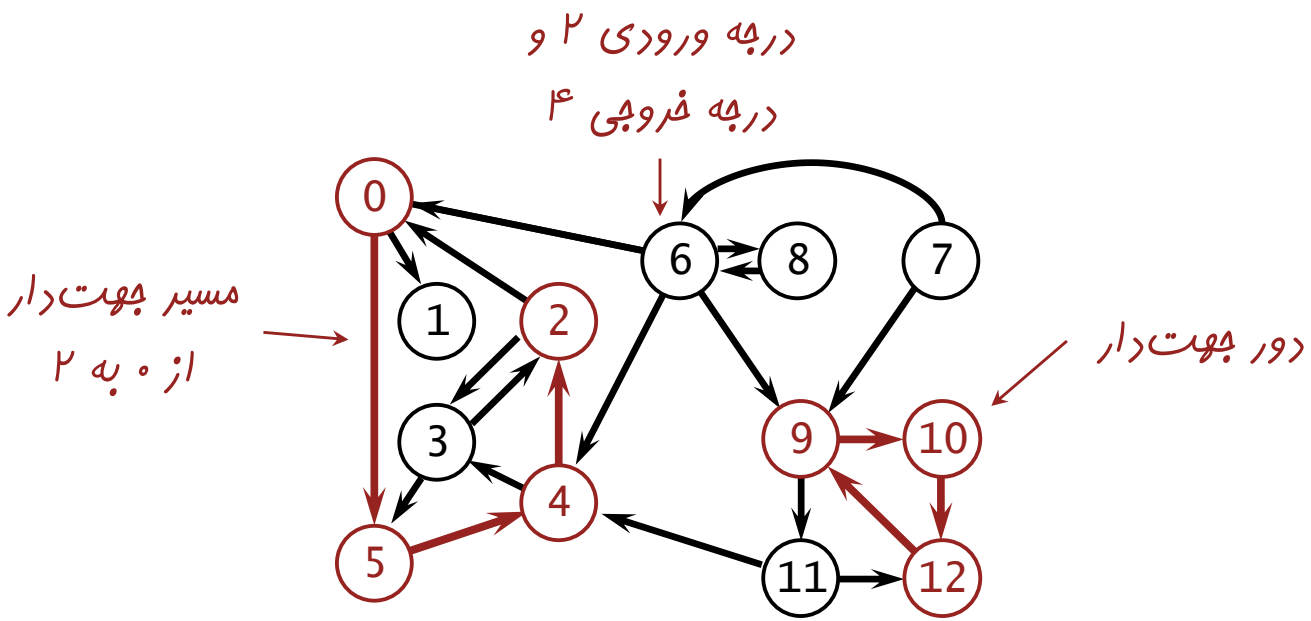



- کدام رئوس؟
  - تک مبدأ-تک مقصد: کوتاه‌ترین مسیر از یک رأس به یک رأس دیگر
  - تک مبدأ: کوتاه‌ترین مسیر از یک رأس به تمام رئوس دیگر
  - همه‌ی زوج رئوس: کوتاه‌ترین مسیرها بین همه‌ی زوج رئوس
- محدودیت بر روی وزن یالها؟
  - وزنهای غیرمنفی
  - وزنهای دلخواه
  - وزنهای اقلیدسی
- دور؟
  - بدون «دور جهت‌دار»
  - بدون «دور منفی»

# الگوریتم فلوید برای محاسبه کوتاه‌ترین مسیرها



- اصطلاحات.
  - گراف: رئوس، یالها
  - گراف جهت‌دار
  - گراف وزن‌دار
  - مسیر، مسیر ساده، دور
  - گراف دور دار، گراف بدون دور
  - طول یک مسیر




یک گراف جهت‌دار و وزن‌دار

# مسائل بهینه‌سازی



- مسئله‌ی بهینه‌سازی.
  - می‌تواند بیش از یک راه‌حل کاندیدا وجود داشته باشد.
  - هر راه‌حل کاندیدا دارای یک مقدار است.
  - هدف پیدا کردن یک راه‌حل کاندیدا با بهترین مقدار (مقدار بهینه) است.
- الگوریتم کورکورانه.
  - الگوریتمی که برای یافتن راه‌حل بهینه، تمامی راه‌حل‌های ممکن را بررسی می‌کند.
- الگوریتم کورکورانه برای مسئله‌ی کوتاه‌ترین مسیر.
  - دارای پیچیدگی زمانی به صورت فاکتوریل!

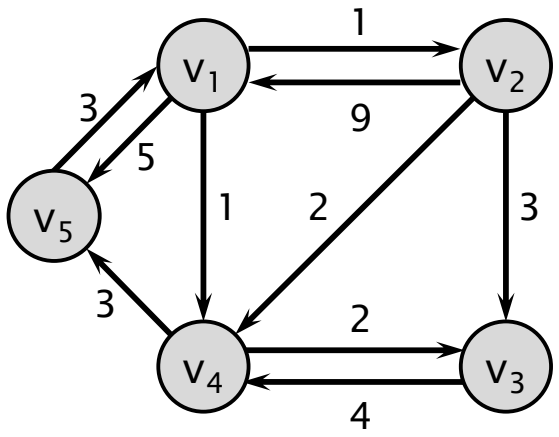
$(n-2) \times (n-1) \times \cdots \times 2 \times 1 = (n-2)!$ حداقل تعداد مسیرهای ممکن میان دو رأس دلخواه

# نمایش گراف



- ماتریس وزن.

$$W[i][j] = \begin{cases} w_{ij} & \text{اگر یالی بین } v_i \text{ و } v_j \text{ وجود داشته باشد} \\ \infty & \text{اگر یالی بین } v_i \text{ و } v_j \text{ وجود نداشته باشد} \\ 0 & \text{اگر } i = j \text{ باشد} \end{cases}$$




| W | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | 0 | 3 | 2 | ∞ |
| 3 | ∞ | ∞ | 0 | 4 | ∞ |
| 4 | ∞ | ∞ | 2 | 0 | 3 |
| 5 | 3 | ∞ | ∞ | ∞ | 0 |

| D | 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 |

# الگوریتم فلوید



- الگوریتم. محاسبه‌ی دنباله‌ای از $n+1$ ماتریس $D^{(k)}$ به ازای $0 \le k \le n$

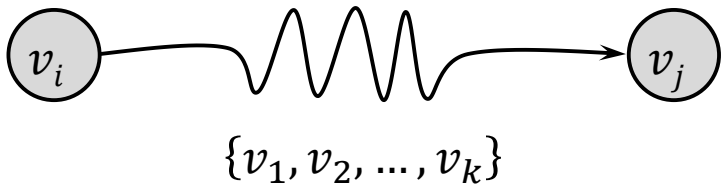
$$D^{(0)} \to D^{(1)} \to D^{(2)} \to \cdots \to D^{(k)} \to \cdots \to D^{(n)}$$

- $D^{(k)}[i][j]$ : طول کوتاه‌ترین مسیر از رأس $v_i$ به رأس $v_j$ که تنها از رئوس موجود در مجموعه‌ی $\{v_1, v_2, \ldots, v_k\}$ به عنوان رئوس میانی استفاده می‌کند.

- دو حالت خاص.

$$D^{(0)} = W$$

$$D^{(n)} = D$$



# برنامه‌نویسی پویا برای مسئله کوتاه‌ترین مسیرها



- هدف. یافتن روشی به منظور محاسبه‌ی $D = D^{(n)}$ از روی $W = D^{(0)}$
- مراحل.
  - ارائه‌ی یک خاصیت بازگشتی به منظور محاسبه‌ی $D^{(k)}$ از $D^{(k-1)}$
  - حل مسئله به روش پایین به بالا به ازای $k = 1$ تا $k = n$ و محاسبه‌ی دنباله زیر:

$$D^{(0)} \rightarrow D^{(1)} \rightarrow D^{(2)} \rightarrow \cdots \rightarrow D^{(k)} \rightarrow \cdots \rightarrow D^{(n)}$$

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

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

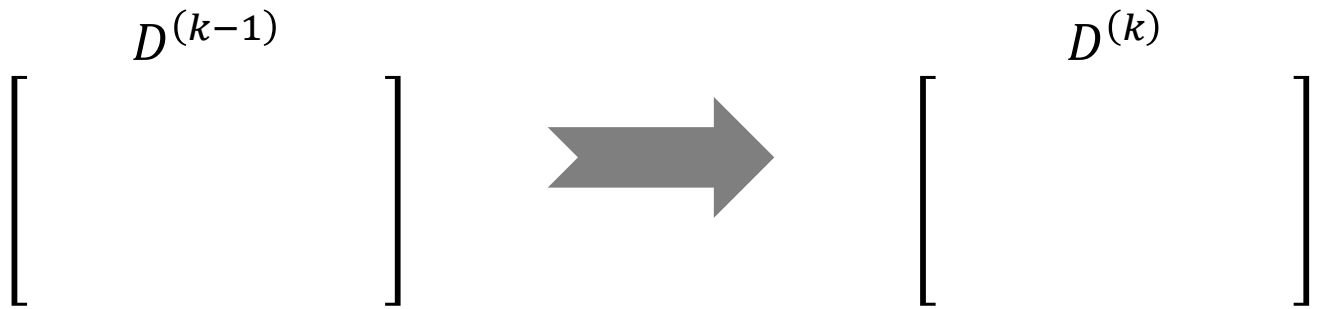
# الگوریتم فلوید: شبه کد



$D^{(0)} = W$

**for** $(\text{int } k = 1; k \leq n; k++)$

    compute $D^{(k)}$ from $D^{(k-1)}$



# محاسبه‌ی $D^{(k)}$ از $D^{(k-1)}$



- حالت ۱. حداقل یک نمونه از کوتاه‌ترین مسیرها از $v_i$ به $v_j$ که فقط از رئوس موجود در مجموعه‌ی $\{v_1, v_2, \ldots, v_k\}$ به عنوان رئوس میانی استفاده می‌کند، از $v_k$ عبور نکند. در این صورت:

$$D^{(k)}[i][j] = D^{(k-1)}[i][j]$$

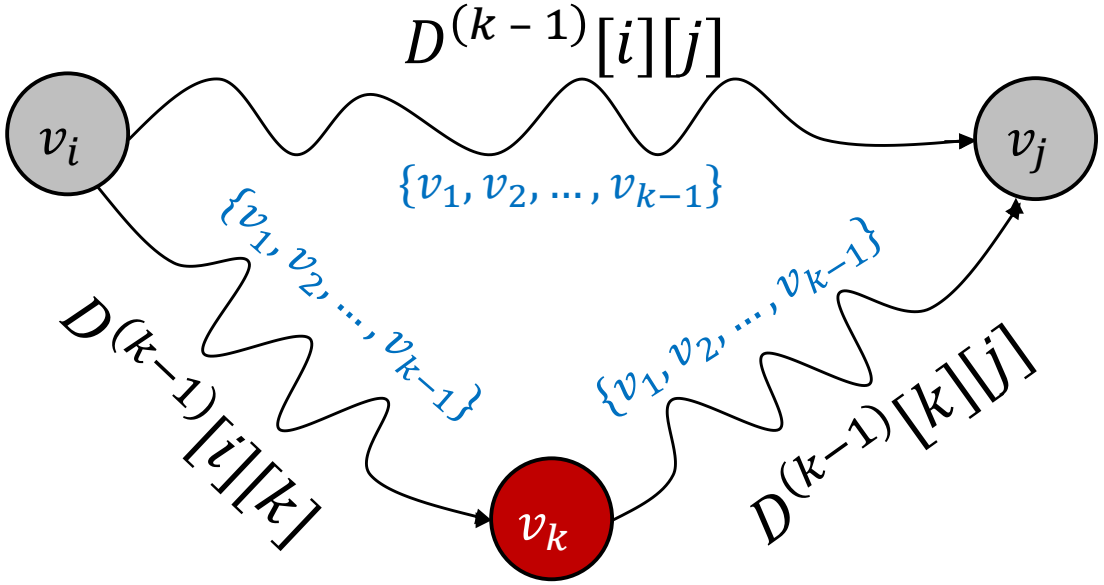
- حالت ۲. تمام نمونه‌های کوتاه‌ترین مسیر از $v_i$ به $v_j$ که فقط از رئوس موجود در مجموعه‌ی $\{v_1, v_2, \ldots, v_k\}$ به عنوان رئوس میانی استفاده می‌کنند، از $v_k$ عبور کنند. در این صورت:

$$D^{(k)}[i][j] = \min\left(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j]\right)$$

# ارائه خاصیت بازگشتی: محاسبه $D^{(k)}$ از روی $D^{(k-1)}$



$$D^{(k-1)} \longrightarrow D^{(k)}$$




$$D^{(k)}[i][j] = minimum(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$

# الگوریتم فلوید: شبه کد



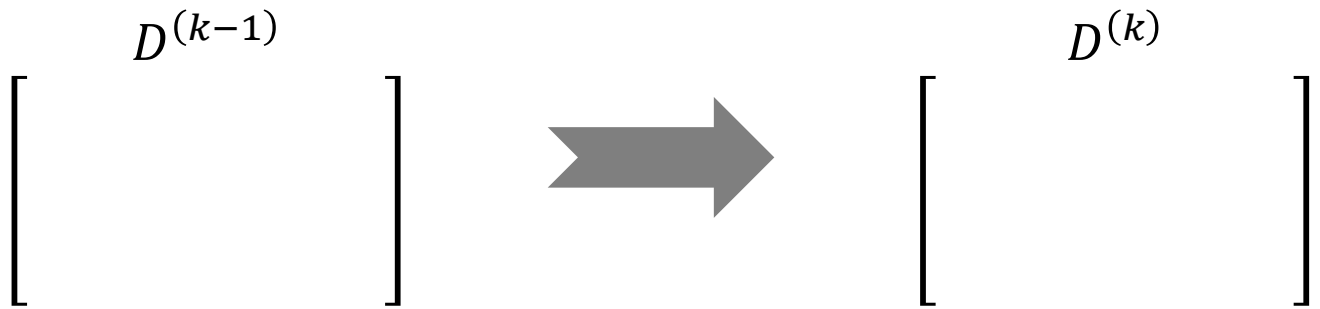
$D^{(0)} = W$

**for** $(\text{int } k = 1; k \leq n; k++)$

  **for** $(\text{int } i = 1; i \leq n; i++)$

    **for** $(\text{int } j = 1; j \leq n; j++)$

      $D^{(k)}[i][j] = \min\left\{D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j]\right\}$



# الگوریتم فلوید: شبه کد



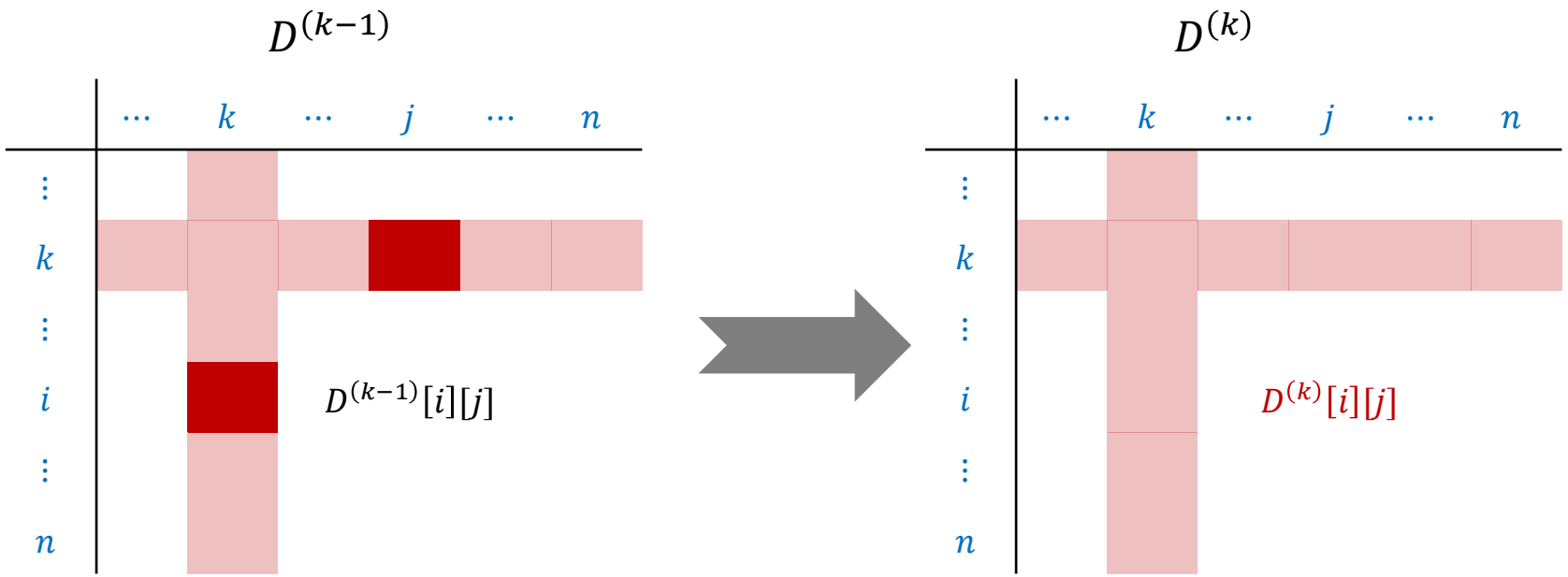
$D^{(0)} = W$

**for** $(\text{int } k = 1; k \leq n; k++)$

  **for** $(\text{int } i = 1; i \leq n; i++)$

    **for** $(\text{int } j = 1; j \leq n; j++)$

      $D^{(k)}[i][j] = \min\left\{D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j]\right\}$



# مثال



$$D^{(k)}[i][j] = minimum(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$

$D^{(0)}$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | 0 | 3 | 2 | ∞ |
| 3 | ∞ | ∞ | 0 | 4 | ∞ |
| 4 | ∞ | ∞ | 2 | 0 | 3 |
| 5 | 3 | ∞ | ∞ | ∞ | 0 |

$k$

$D^{(1)}$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | | | | |
| 3 | ∞ | | | | |
| 4 | ∞ | | | | |
| 5 | 3 | | | | |

$$D^{(1)}[1][1] = \min\left(D^{(0)}[1][1], D^{(0)}[1][1] + D^{(0)}[1][1]\right) = \min(0, 0+0) = 0$$
$$D^{(1)}[1][2] = \min\left(D^{(0)}[1][2], D^{(0)}[1][1] + D^{(0)}[1][2]\right) = \min(1, 0+1) = 1$$
$$D^{(1)}[1][3] = \min\left(D^{(0)}[1][3], D^{(0)}[1][1] + D^{(0)}[1][3]\right) = \min(\infty, 0+\infty) = 0$$
$$D^{(1)}[1][4] = \min\left(D^{(0)}[1][4], D^{(0)}[1][1] + D^{(0)}[1][4]\right) = \min(1, 0+1) = 1$$
$$D^{(1)}[1][5] = \min\left(D^{(0)}[1][5], D^{(0)}[1][1] + D^{(0)}[1][5]\right) = \min(5, 0+5) = 5$$

# مثال



$$D^{(k)}[i][j] = minimum(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$

$D^{(0)}$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | 0 | 3 | 2 | ∞ |
| 3 | ∞ | ∞ | 0 | 4 | ∞ |
| 4 | ∞ | ∞ | 2 | 0 | 3 |
| 5 | 3 | ∞ | ∞ | ∞ | 0 |

$D^{(1)}$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | 0 | 3 | 2 | 14 |
| 3 | ∞ | | | | |
| 4 | ∞ | | | | |
| 5 | 3 | | | | |

$$D^{(1)}[2][1] = \min\left(D^{(0)}[2][1], D^{(0)}[2][1] + D^{(0)}[1][1]\right) = \min(9, 9+0) = 9$$
$$D^{(1)}[2][2] = \min\left(D^{(0)}[2][2], D^{(0)}[2][1] + D^{(0)}[1][2]\right) = \min(0, 1+9) = 0$$
$$D^{(1)}[2][3] = \min\left(D^{(0)}[2][3], D^{(0)}[2][1] + D^{(0)}[1][3]\right) = \min(3, 9+\infty) = 3$$
$$D^{(1)}[2][4] = \min\left(D^{(0)}[2][4], D^{(0)}[2][1] + D^{(0)}[1][4]\right) = \min(2, 9+1) = 2$$
$$D^{(1)}[2][5] = \min\left(D^{(0)}[2][5], D^{(0)}[2][1] + D^{(0)}[1][5]\right) = \min(\infty, 9+5) = 14$$

# مثال



$$D^{(k)}[i][j] = minimum(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$

$D^{(0)}$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | 0 | 3 | 2 | ∞ |
| 3 | ∞ | ∞ | 0 | 4 | ∞ |
| 4 | ∞ | ∞ | 2 | 0 | 3 |
| 5 | 3 | ∞ | ∞ | ∞ | 0 |

$D^{(1)}$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | 0 | 3 | 2 | 14 |
| 3 | ∞ | ∞ | 0 | 4 | ∞ |
| 4 | ∞ | | | | |
| 5 | 3 | | | | |

$$D^{(1)}[3][1] = \min\left(D^{(0)}[3][1], D^{(0)}[3][1] + D^{(0)}[1][1]\right) = \min(\infty, \infty + 0) = \infty$$
$$D^{(1)}[3][2] = \min\left(D^{(0)}[3][2], D^{(0)}[3][1] + D^{(0)}[1][2]\right) = \min(\infty, \infty + 1) = \infty$$
$$D^{(1)}[3][3] = \min\left(D^{(0)}[3][3], D^{(0)}[3][1] + D^{(0)}[1][3]\right) = \min(0, \infty + \infty) = 0$$
$$D^{(1)}[3][4] = \min\left(D^{(0)}[3][4], D^{(0)}[3][1] + D^{(0)}[1][4]\right) = \min(4, \infty + 1) = 4$$
$$D^{(1)}[3][5] = \min\left(D^{(0)}[3][5], D^{(0)}[3][1] + D^{(0)}[1][5]\right) = \min(\infty, \infty + 5) = \infty$$

# مثال



$$D^{(k)}[i][j] = minimum(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$

$D^{(0)}$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | 0 | 3 | 2 | ∞ |
| 3 | ∞ | ∞ | 0 | 4 | ∞ |
| 4 | ∞ | ∞ | 2 | 0 | 3 |
| 5 | 3 | ∞ | ∞ | ∞ | 0 |

$D^{(1)}$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | 0 | 3 | 2 | 14 |
| 3 | ∞ | ∞ | 0 | 4 | ∞ |
| 4 | ∞ | ∞ | 2 | 0 | 3 |
| 5 | 3 | | | | |

$$D^{(1)}[4][1] = \min\left(D^{(0)}[4][1], D^{(0)}[4][1] + D^{(0)}[1][1]\right) = \min(\infty, \infty + 0) = \infty$$
$$D^{(1)}[4][2] = \min\left(D^{(0)}[4][2], D^{(0)}[4][1] + D^{(0)}[1][2]\right) = \min(\infty, \infty + 1) = \infty$$
$$D^{(1)}[4][3] = \min\left(D^{(0)}[4][3], D^{(0)}[4][1] + D^{(0)}[1][3]\right) = \min(2, \infty + \infty) = 2$$
$$D^{(1)}[4][4] = \min\left(D^{(0)}[4][4], D^{(0)}[4][1] + D^{(0)}[1][4]\right) = \min(0, \infty + 1) = 0$$
$$D^{(1)}[4][5] = \min\left(D^{(0)}[4][5], D^{(0)}[4][1] + D^{(0)}[1][5]\right) = \min(3, \infty + 5) = 3$$

# مثال



$$D^{(k)}[i][j] = minimum(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$

$D^{(0)}$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | 0 | 3 | 2 | ∞ |
| 3 | ∞ | ∞ | 0 | 4 | ∞ |
| 4 | ∞ | ∞ | 2 | 0 | 3 |
| 5 | 3 | ∞ | ∞ | ∞ | 0 |

$D^{(1)}$

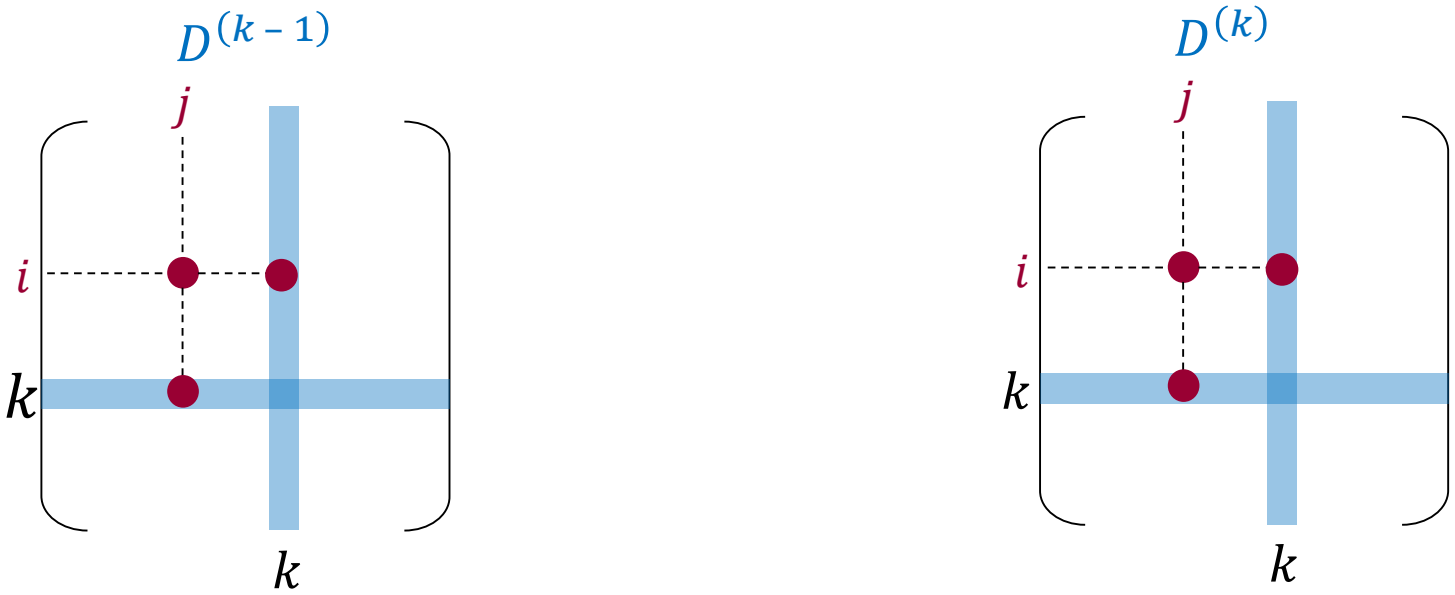
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | ∞ | 1 | 5 |
| 2 | 9 | 0 | 3 | 2 | 14 |
| 3 | ∞ | ∞ | 0 | 4 | ∞ |
| 4 | ∞ | ∞ | 2 | 0 | 3 |
| 5 | 3 | 4 | ∞ | 4 | 0 |

$$D^{(1)}[5][1] = \min(D^{(0)}[5][1], D^{(0)}[5][1] + D^{(0)}[1][1]) = \min(3, 3+0) = 3$$
$$D^{(1)}[5][2] = \min(D^{(0)}[5][2], D^{(0)}[5][1] + D^{(0)}[1][2]) = \min(\infty, 3+1) = 4$$
$$D^{(1)}[5][3] = \min(D^{(0)}[5][3], D^{(0)}[5][1] + D^{(0)}[1][3]) = \min(\infty, 3+\infty) = \infty$$
$$D^{(1)}[5][4] = \min(D^{(0)}[5][4], D^{(0)}[5][1] + D^{(0)}[1][4]) = \min(\infty, 3+1) = 4$$
$$D^{(1)}[5][5] = \min(D^{(0)}[5][5], D^{(0)}[5][1] + D^{(0)}[1][5]) = \min(0, 3+5) = 0$$

# الگوریتم فلوید: کاهش مصرف حافظه



$$D^{(k)}[i][j] = minimum(D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j])$$




$$D^{(k)}[k][j] = minimum(D^{(k-1)}[k][j], \cancel{D^{(k-1)}[k][k]} + D^{(k-1)}[k][j])$$

$$D^{(k)}[i][k] = minimum(D^{(k-1)}[i][k], D^{(k-1)}[i][k] + \cancel{D^{(k-1)}[k][k]})$$

# الگوریتم فلوید: شبه کد



$D = W$

**for** $(\text{int } k = 1; k \leq n; k++)$

  **for** $(\text{int } i = 1; i \leq n; i++)$

    **for** $(\text{int } j = 1; j \leq n; j++)$

      $D[i][j] = \min\{D[i][j], D[i][k] + D[k][j]\}$

- تحلیل. پیچیدگی زمانی

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

# الگوریتم فلوید: محاسبه‌ی خود کوتاه‌ترین مسیرها



```
for (int i = 1; i ≤ n; i++)
    for (int j = 1; j ≤ n; j++)
        P[i][j] = 0

D = W
for (int k = 1; k ≤ n; k++)
    for (int i = 1; i ≤ n; i++)
        for (int j = 1; j ≤ n; j++)
            if (D[i][k] + D[k][j] < D[i][j]) {
                D[i][j] = D[i][k] + D[k][j]
                P[i][j] = k
            }
```

# محاسبه‌ی کوتاه‌ترین مسیرها

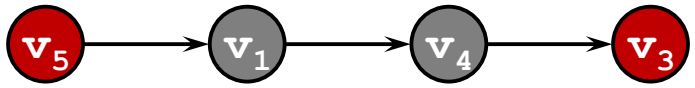


- محاسبه‌ی کوتاه‌ترین مسیر بین هر دو رأس دلخواه از روی ماتریس $P$.

```
public static void path(int i, int j) {
     if (P[i][j] == 0) return;
     path(i, P[i][j]);
     StdOut.print(P[i][j] + " ");
     path(P[i][j], j);
}
```

| P | 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 |

- مثال. محاسبه‌ی کوتاه‌ترین مسیر بین $v_5$ و $v_3$



# برنامه‌نویسی پویا و مسائل بهینه‌سازی



- مراحل حل مسائل بهینه‌سازی.
  - ارائه‌ی یک خاصیت بازگشتی برای محاسبه‌ی راه‌حل نمونه‌های بزرگ‌تر از نمونه‌های کوچک‌تر.
  - محاسبه‌ی مقدار راه‌حل بهینه به روش پایین به بالا.
  - محاسبه‌ی خود راه‌حل بهینه به روش بالا به پایین.

- توجه. به منظور حل یک مسئله‌ی بهینه‌سازی با استفاده از برنامه‌ریزی پویا، باید اصل بهینگی در آن مسئله برقرار باشد.

# اصل بهینگی



- اصل بهینگی. راه‌حل بهینه‌ی یک نمونه از مسئله، همواره شامل راه‌حل‌های بهینه برای تمامی زیرنمونه‌ها است.
  - به بیان دیگر، اگر یک نمونه از مسئله به صورت بهینه حل شده باشد، تمامی زیرنمونه‌های موجود در آن نیز باید به صورت بهینه حل شده باشند.

- این اصل اطمینان می‌دهد که راه‌حل بهینه‌ی یک نمونه از مسئله می‌تواند با ترکیب راه‌حل بهینه‌ی زیرنمونه‌ها حاصل شود.

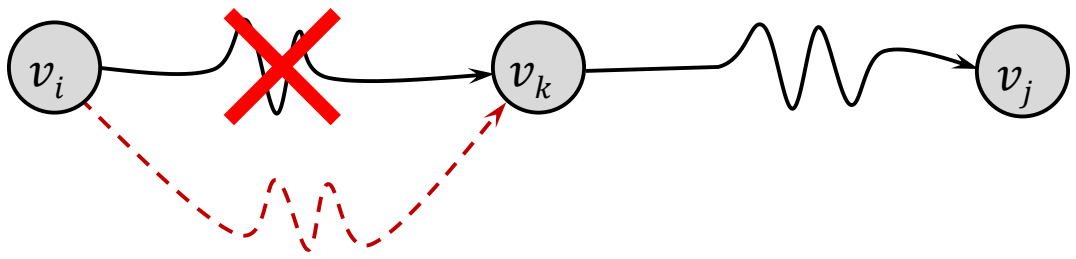
- بنابراین، قبل از استفاده از برنامه‌ریزی پویا برای به دست آوردن راه‌حل، لازم است نشان دهیم اصل بهینگی برقرار است.

# اصل بهینگی: استدلال ببُر و بچسبان



- استدلال ببُر و بچسبان.

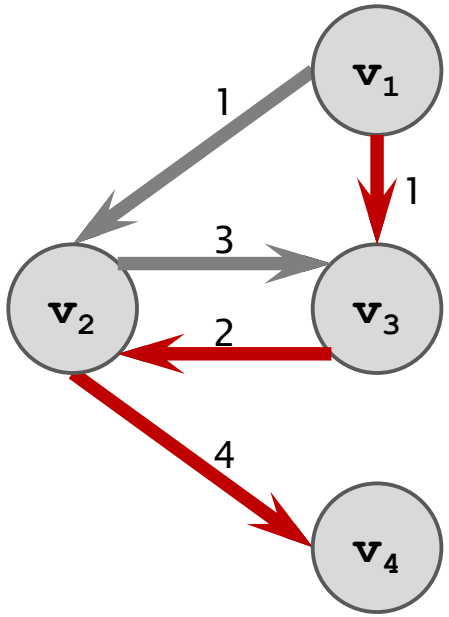
- مثال. مسئله‌ی یافتن کوتاه‌ترین مسیرها.
  - فرض کنید کوتاه‌ترین مسیر بین رئوس $v_i$ و $v_j$ از رأس $v_k$ به عنوان یک رأس میانی می‌گذرد.
  - باید نشان دهیم مسیر $v_i \rightsquigarrow v_k$ و همچنین مسیر $v_k \rightsquigarrow v_j$، کوتاه‌ترین مسیرند.



# اصل بهینگی: استدلال ببُر و بچسبان



- مثال. مسئله‌ی یافتن طولانی‌ترین مسیرها




  - طولانی‌ترین مسیر از $v_1$ به $v_4$

$$v_1 \to v_3 \to v_2 \to v_4$$

  - طولانی‌ترین مسیر از $v_1$ به $v_3$

$$v_1 \to v_2 \to v_3$$

- بنابراین، در این مسئله اصل بهینگی برقرار نیست.

# ترتیب بهینه برای ضرب زنجیری ماتریس‌ها

# ضرب زنجیری ماتریس‌ها



- مسئله. $n$ ماتریس به صورت زیر داده شده است. می‌خواهیم ترتیب بهینه به منظور ضرب کردن این ماتریس‌ها را پیدا کنیم، به گونه‌ای که بتوانیم این ماتریس‌ها را با حداقل تعداد ضرب‌های ممکن، در یکدیگر ضرب کنیم.

$$A_1\,(d_0 \times d_1),\qquad A_2\,(d_1 \times d_2),\qquad \cdots,\qquad A_k\,(d_{k-1} \times d_k),\qquad \cdots, A_n\,(d_{n-1} \times d_n)$$

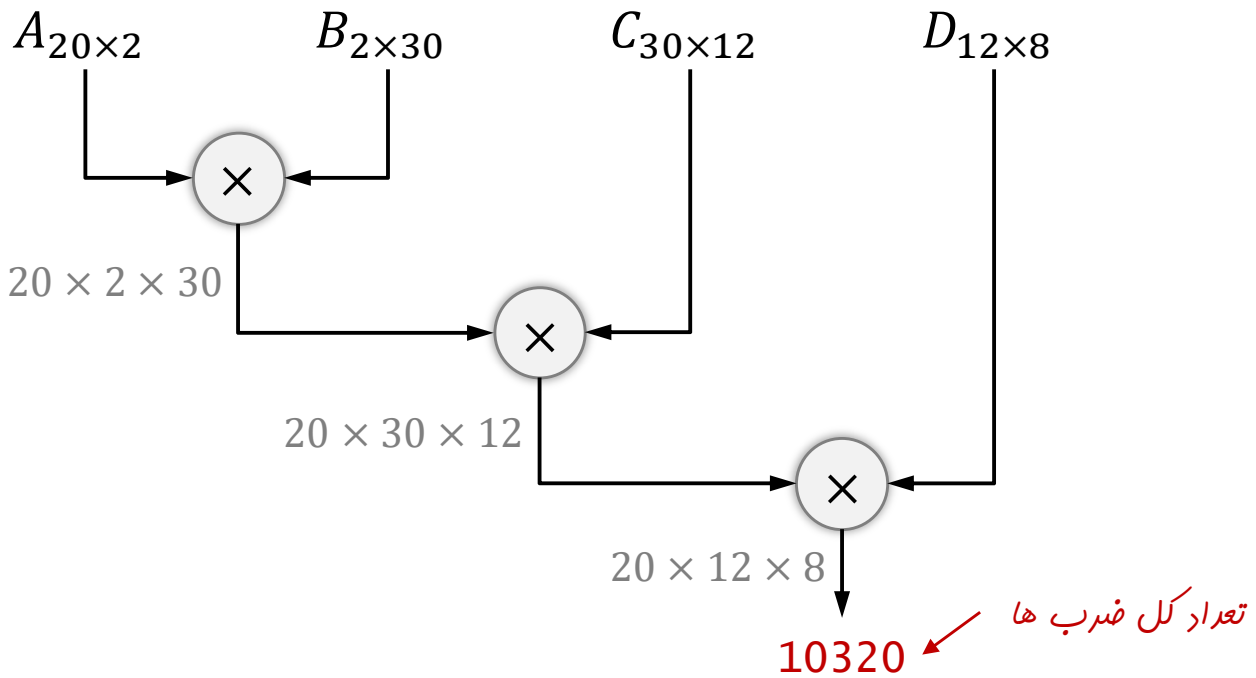
# مثال



- س. بهترین ترتیب برای ضرب ماتریس‌های زیر کدام است؟

$$A_{20\times2} \times B_{2\times30} \times C_{30\times12} \times D_{12\times8}$$

- روش ۱.



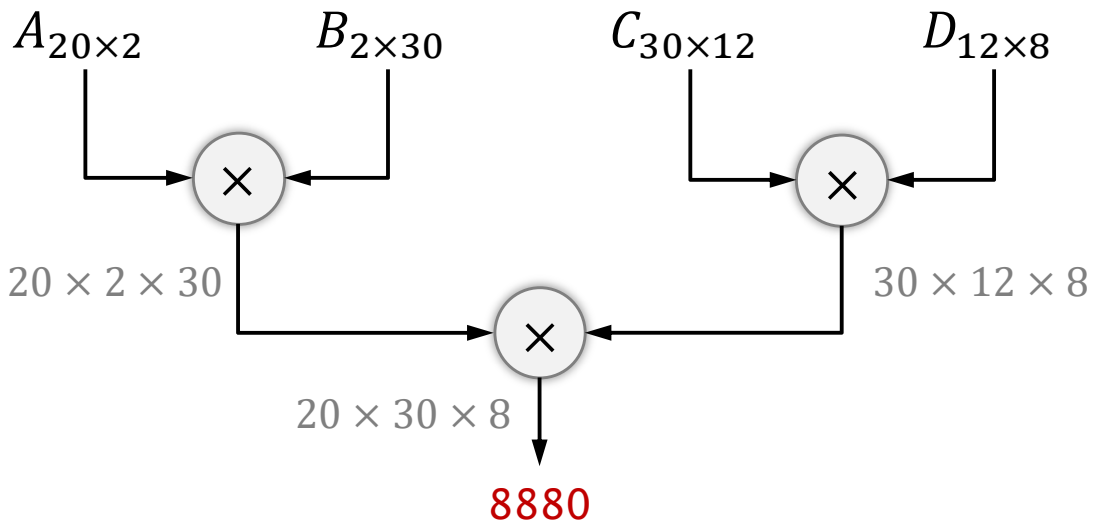
# مثال



- س. بهترین ترتیب برای ضرب ماتریس‌های زیر کدام است؟

$$A_{20\times2} \times B_{2\times30} \times C_{30\times12} \times D_{12\times8}$$

- روش ۲.



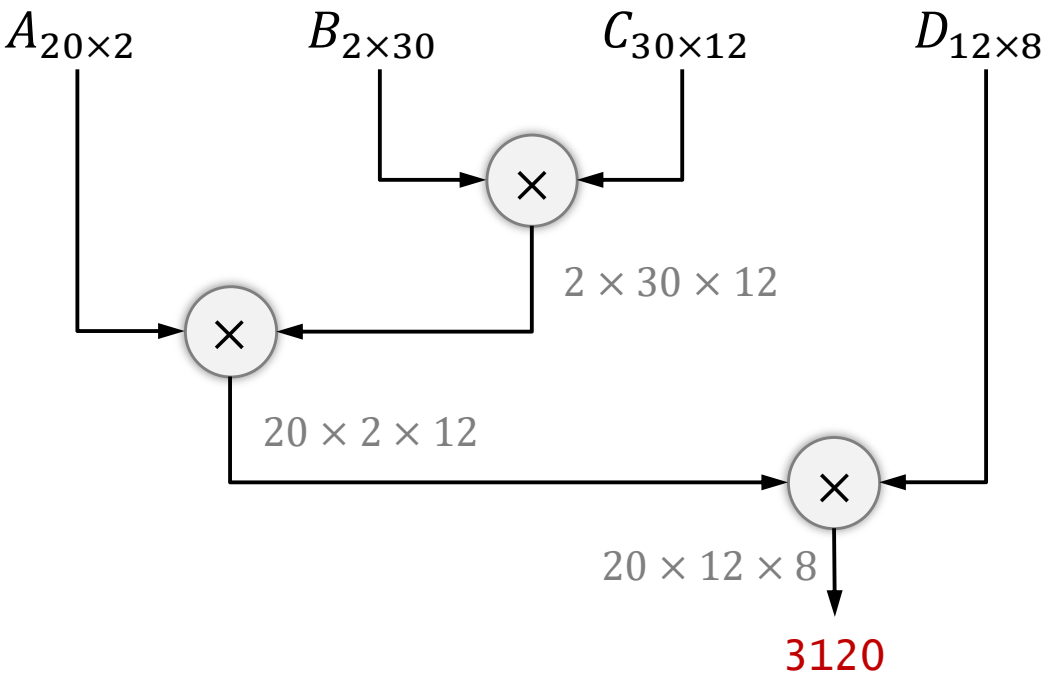
# مثال



- س. بهترین ترتیب برای ضرب ماتریس‌های زیر کدام است؟

$$A_{20\times2} \times B_{2\times30} \times C_{30\times12} \times D_{12\times8}$$

- روش ۳.



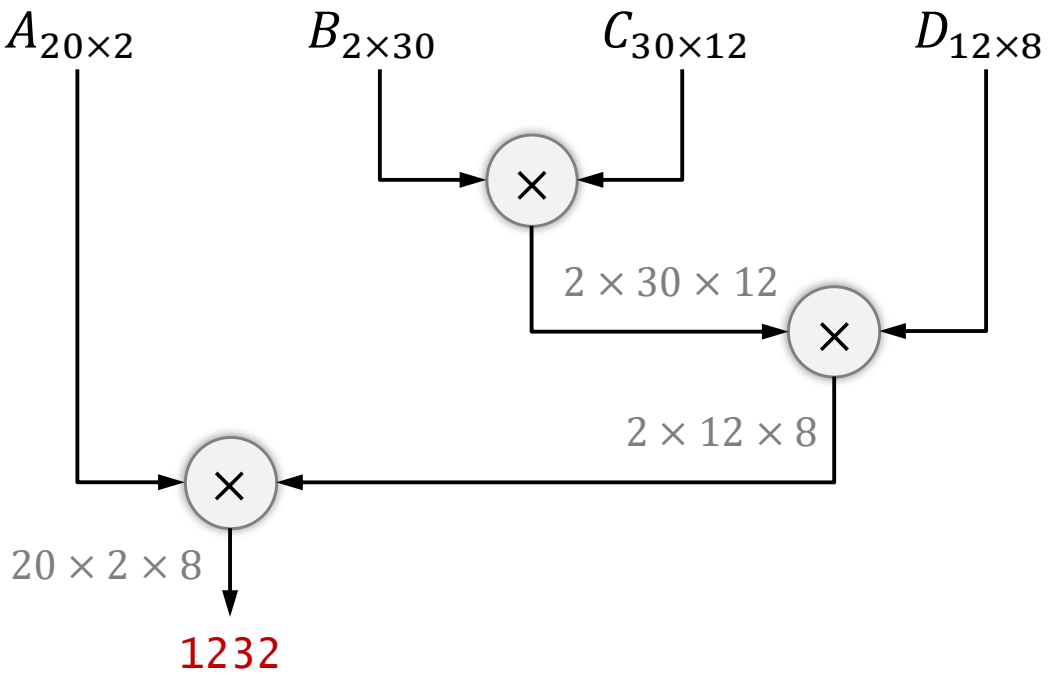
# مثال



- س. بهترین ترتیب برای ضرب ماتریس‌های زیر کدام است؟

$$A_{20\times2} \times B_{2\times30} \times C_{30\times12} \times D_{12\times8}$$

- روش ۴.



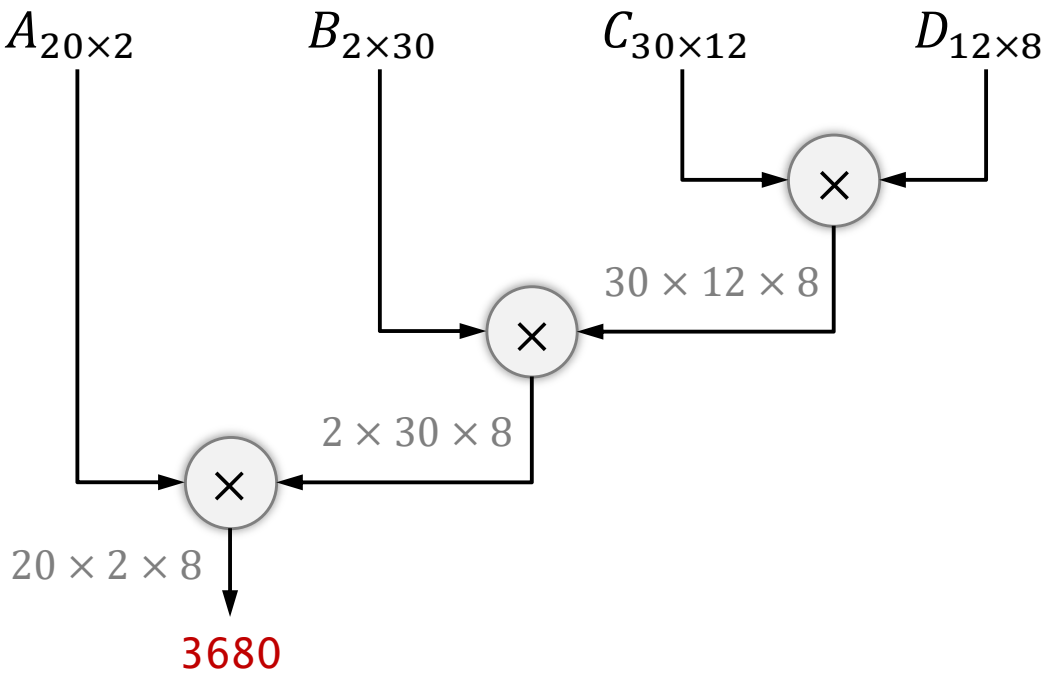
# مثال



- س. بهترین ترتیب برای ضرب ماتریس‌های زیر کدام است؟

$$A_{20\times2} \times B_{2\times30} \times C_{30\times12} \times D_{12\times8}$$

- روش ۵.



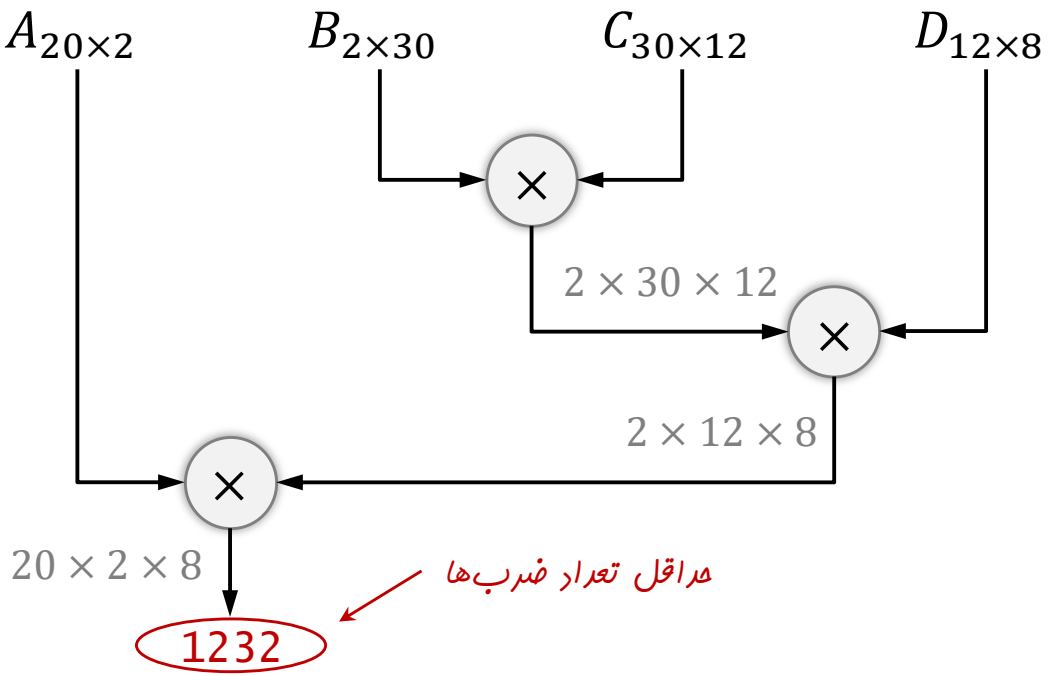
# مثال: پاسخ بهینه



- س. بهترین ترتیب برای ضرب ماتریس‌های زیر کدام است؟

$$A_{20\times2} \times B_{2\times30} \times C_{30\times12} \times D_{12\times8}$$

- روش ۴.




ترتیب بهینه

$$(A_{20\times2} \times ((B_{2\times30} \times C_{30\times12}) \times D_{12\times8}))$$

# الگوریتم کورکورانه



- الگوریتم کورکورانه. الگوریتمی که با بررسی همه‌ی راه‌حل‌های ممکن در یک مسئله‌ی بهینه‌سازی، راه‌حل بهینه را پیدا می‌کند.
- پیچیدگی الگوریتم کورکورانه. حداقل برابر است با تعداد راه‌حل‌های ممکن.
- س. تعداد روش‌های ممکن برای ضرب $n$ ماتریس کدام است؟
- تعریف. $T(n)$ = تعداد روش‌های ممکن برای ضرب $n$ ماتریس

$T(1) = 1, \quad T(2) = 1, \quad T(3) = 2, \quad T(4) = 5, \quad T(5) = 14,$

$T(n) = ?$

# تعداد روش های ممکن برای ضرب n ماتریس



- س. تعداد روش‌های ممکن برای ضرب $n$ ماتریس کدام است؟
- تعریف. $T(n)$ = تعداد روش‌های ممکن برای ضرب $n$ ماتریس

$$(A_1 A_2 \cdots A_k)(A_{k+1} \cdots A_{n-1} A_n)$$

$T(k)$ $\quad$ $T(n-k)$

$$T(n) = \sum_{k=1}^{n-1} T(k) \cdot T(n-k)$$

$$= \frac{1}{n}\binom{2n-2}{n-1}$$

جمله $n-1$ ام در سری اعداد کاتالان

# دنباله‌ی اعداد کاتالان



- دنباله‌ی اعداد کاتالان.

$$T(n) = \frac{1}{n+1}\binom{2n}{n}$$

$$T(n) \approx \frac{4^n}{\sqrt{\pi n^3}} \in O\left(\frac{4^n}{n^{1.5}}\right)$$

| $N$ | $T(N)$ |
|---|---|
| 10 | 16796 |
| 20 | 6564120420 |
| 30 | 3814986502092304 |
| 40 | 2.6221270422764923e+21 |
| 50 | 1.9782616577561612e+27 |
| 60 | 1.5838509645961200e+33 |
| 70 | 1.3214221084202819e+39 |
| 80 | 1.1363595779473356e+45 |
| 90 | 1.0001346008003541e+51 |
| 100 | 8.9651994709013100e+56 |

# الگوریتم برنامه ریزی پویا: یادآوری



- مراحل حل یک مسئله‌ی بهینه‌سازی به وسیله‌ی $DP$.
  - ارائه‌ی یک خاصیت بازگشتی؛
  - محاسبه و ذخیره‌ی مقدار راه‌حل نمونه‌های بزرگ‌تر با داشتن راه‌حل نمونه‌های کوچک‌تر؛
  - محاسبه‌ی خود راه‌حل بهینه [با استفاده از یک الگوریتم تقسیم و حل]

- تعریف. $M[i][j]$

حداقل تعداد ضرب‌های لازم برای ضرب ماتریس‌های $A_i$ تا $A_j$ به ازای $i \leq j$

$M[i][i] = 0$

- مقدار راه‌حل بهینه.

$M[1][n]$

# مرحله ۱) ارائه‌ی یک خاصیت بازگشتی



- محاسبه‌ی $M[i][j]$:

$$\underbrace{(A_i A_2 \cdots A_k)}_{d_{i-1} \times d_i \quad \cdots \quad d_{k-1} \times d_k}\underbrace{(A_{k+1} \cdots A_{j-1} A_j)}_{d_k \times d_{k+1} \quad \cdots \quad d_{j-1} \times d_j} \qquad k = i, \ldots, j-1$$

- حداقل تعداد ضرب‌های لازم برای ضرب $A_i$ تا $A_k$: $M[i][k]$
- حداقل تعداد ضرب‌های لازم برای ضرب $A_{k+1}$ تا $A_j$: $M[k+1][j]$
- تعداد ضرب‌های لازم برای ضرب زیرمسئله‌ی اول در زیرمسئله‌ی دوم: $d_{i-1} \times d_k \times d_j$

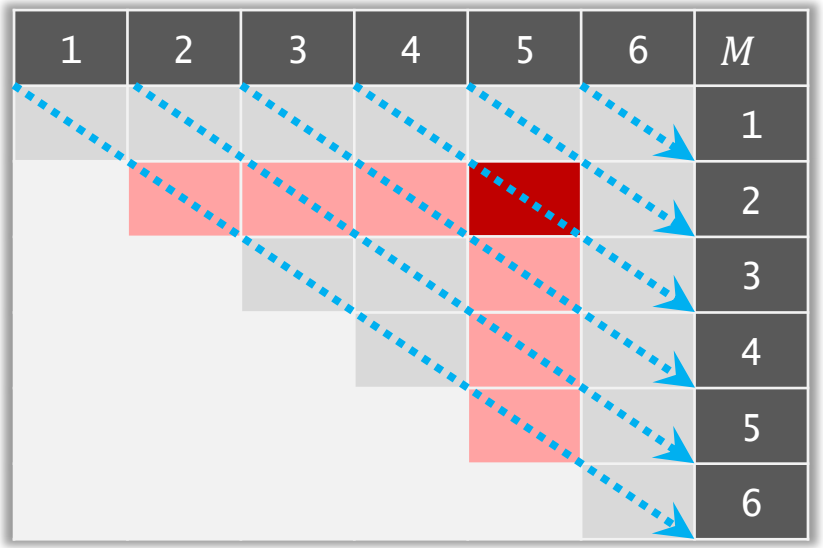
$$M[i][j] = \left(M[i][k] + M[k+1][j] + d_{i-1} \times d_k \times d_j\right)$$

# مرحله ۲) حل نمونه‌ها از کوچک به بزرگ



- مثال. $n = 6$

$$M[i][j] = \min_{i \le k \le j-1} \left( M[i][k] + M[k+1][j] + d_{i-1} \times d_k \times d_j \right)$$




$$M[2][5]$$

$$= \min_{2 \le k \le 5-1} (M[2][k] + M[k+1][5] + d_1 \times d_k \times d_5)$$

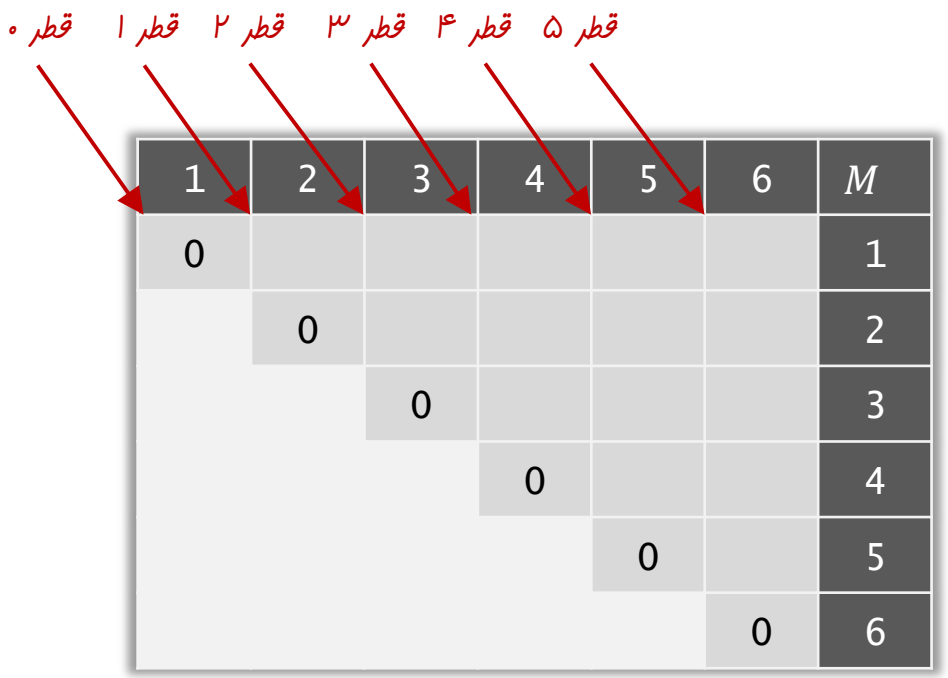
$$= \min \begin{pmatrix} M[2][2] + M[3][5] + d_1 \times d_2 \times d_5, \\ M[2][3] + M[4][5] + d_1 \times d_3 \times d_5, \\ M[2][4] + M[5][5] + d_1 \times d_4 \times d_5 \end{pmatrix}$$

# مرحله ۲) حل نمونه‌ها از کوچک به بزرگ



- مثال. $n = 6$

$$M[i][j] = \min_{i \le k \le j-1} \left( M[i][k] + M[k+1][j] + d_{i-1} \times d_k \times d_j \right)$$



# مثال (n = 4)



- محاسبه‌ی قطر صفر.

$d_0 = 10, d_1 = 20, d_2 = 50, d_3 = 1, d_4 = 100$

```
for (int i = 1; i <= n; i++)
     M[i][i] = 0;
```

$M[1][1] = 0$

$M[2][2] = 0$

$M[3][3] = 0$

$M[4][4] = 0$

| 1 | 2 | 3 | 4 | $M$ |
|---|---|---|---|---|
| 0 | | | | 1 |
| | 0 | | | 2 |
| | | 0 | | 3 |
| | | | 0 | 4 |

# مثال (n = 4)



- محاسبه‌ی قطر ۱.

$$d_0 = 10, d_1 = 20, d_2 = 50, d_3 = 1, d_4 = 100$$

$$M[i][j] = \min_{i \le k \le j-1} \left( M[i][k] + M[k+1][j] + d_{i-1} \times d_k \times d_j \right)$$

$$M[1][2] = \min_{1 \le k \le 1} (M[1][1] + M[2][2] + d_0 d_1 d_2)$$

| 1 | 2 | 3 | 4 | $M$ |
|---|---|---|---|---|
| 0 | 10k | | | 1 |
| | 0 | | | 2 |
| | | 0 | | 3 |
| | | | 0 | 4 |

# مثال (n = 4)



- محاسبه‌ی قطر ۱.

$$d_0 = 10, d_1 = 20, d_2 = 50, d_3 = 1, d_4 = 100$$

$$M[i][j] = \min_{i \le k \le j-1} \left( M[i][k] + M[k+1][j] + d_{i-1} \times d_k \times d_j \right)$$

$$M[2][3] = \min_{2 \le k \le 2} (M[2][2] + M[3][3] + d_1 d_2 d_3)$$

| 1 | 2 | 3 | 4 | $M$ |
|---|---|---|---|---|
| 0 | 10k | | | 1 |
| | 0 | 1k | | 2 |
| | | 0 | | 3 |
| | | | 0 | 4 |

# مثال (n = 4)



- محاسبه‌ی قطر ۱.

$$d_0 = 10, d_1 = 20, d_2 = 50, d_3 = 1, d_4 = 100$$

$$M[i][j] = \min_{i \le k \le j-1} \left( M[i][k] + M[k+1][j] + d_{i-1} \times d_k \times d_j \right)$$

$$M[3][4] = \min_{2 \le k \le 2} (M[3][3] + M[4][4] + d_2 d_3 d_4)$$

| 1 | 2 | 3 | 4 | $M$ |
|---|---|---|---|---|
| 0 | 10k | | | 1 |
| | 0 | 1k | | 2 |
| | | 0 | 5k | 3 |
| | | | 0 | 4 |

# مثال (n = 4)



- محاسبه‌ی قطر ۲.

$$d_0 = 10, d_1 = 20, d_2 = 50, d_3 = 1, d_4 = 100$$

$$M[i][j] = \min_{i \le k \le j-1} \left( M[i][k] + M[k+1][j] + d_{i-1} \times d_k \times d_j \right)$$

$$M[1][3] = \min_{1 \le k \le 2} (M[1][k] + M[k+1][2] + d_0 d_k d_3)$$

$$= \min \begin{pmatrix} M[1][1] + M[2][3] + d_0 d_1 d_3, \\ M[1][2] + M[3][3] + d_0 d_2 d_3 \end{pmatrix}$$

| 1 | 2 | 3 | 4 | $M$ |
|---|---|---|---|---|
| 0 | 10k | 1.2k | | 1 |
| | 0 | 1k | | 2 |
| | | 0 | 5k | 3 |
| | | | 0 | 4 |

# مثال (n = 4)



- محاسبه‌ی قطر ۲.

$$d_0 = 10, d_1 = 20, d_2 = 50, d_3 = 1, d_4 = 100$$

$$M[i][j] = \min_{i \le k \le j-1} \left( M[i][k] + M[k+1][j] + d_{i-1} \times d_k \times d_j \right)$$

| 1 | 2 | 3 | 4 | $M$ |
|---|---|---|---|---|
| 0 | 10k | 1.2k | | 1 |
| | 0 | 1k | 3k | 2 |
| | | 0 | 5k | 3 |
| | | | 0 | 4 |

$$M[2][4] = \min_{2 \le k \le 3} (M[2][k] + M[k+1][4] + d_1 d_k d_4)$$

$$= \min \begin{pmatrix} M[2][2] + M[3][4] + d_1 d_2 d_4, \\ M[2][3] + M[4][4] + d_1 d_3 d_4 \end{pmatrix}$$

# مثال (n = 4)



- محاسبه‌ی قطر ۳.

$$d_0 = 10, d_1 = 20, d_2 = 50, d_3 = 1, d_4 = 100$$

$$M[i][j] = \min_{i \le k \le j-1} \left( M[i][k] + M[k+1][j] + d_{i-1} \times d_k \times d_j \right)$$

$$M[1][4] = \min_{1 \le k \le 3} (M[1][k] + M[k+1][4] + d_0 d_k d_4)$$

$$= \min \begin{pmatrix} M[1][1] + M[2][4] + d_0 d_1 d_4, \\ M[1][2] + M[3][4] + d_0 d_2 d_4, \\ M[1][3] + M[4][4] + d_0 d_3 d_4 \end{pmatrix}$$

| 1 | 2 | 3 | 4 | $M$ |
|---|---|---|---|---|
| 0 | 10k | 1.2k | 2.2k | 1 |
| | 0 | 1k | 3k | 2 |
| | | 0 | 5k | 3 |
| | | | 0 | 4 |

# الگوریتم محاسبه‌ی ترتیب بهینه



- محاسبه‌ی عناصر قطر اصلی.

```
for (int i = 1; i <= n; i++)
     M[i][i] = 0;
```

- محاسبه‌ی عناصر قطر ۱ تا $n - 1$.

```
for (int d = 1; d <= n - 1; d++)
     Compute diagonal d
```

- برگرداندن مقدار $M[1][n]$

```
return M[1][n];
```

# محاسبه‌ی عناصر قطر d



- قطر $d$.
  - تعداد عناصر قطر $d$ برابر است با $n - d$
  - شماره ستون عنصر واقع در سطر $i$ از قطر $d$ برابر است با $i + d$

تعداد عناصر قطر $d$

```
for (int i = 1; i <= n - d; i++)
{
    j = i + d;                                  شماره ستون عنصر سطر i و قطر d
    M[i][j] = min_{i<=k<j} (M[i][k] + M[k+1][j] + d[i-1]*d[k]*d[j]);
    P[i][j] = the value of k which gives the minimum;
}
```

# الگوریتم محاسبه‌ی ترتیب بهینه



```
public int min_mult (int n, int[] d, int[][] P)
{
   int[][] M = new int[1..n][1..n];

   for (int i = 1; i <= n; i++)                          محاسبه قطر اصلی
       M[i][i] = 0;


   for (int d = 1; d <= n - 1; d++) {

       for (int i = 1; i <= n - d; i++)                  محاسبه قطر d
       {
          j = i + d;
          M[i][j] = min_{i<=k<j} (M[i][k] + M[k+1][j] + d[i-1]*d[k]*d[j]);
          P[i][j] = the value of k which gives the minimum;
       }

   }
   return M[1][n];                                       برگرداندن مقدار راه حل بهینه
}
```

# الگوریتم محاسبه‌ی ترتیب بهینه



- تحلیل پیچیدگی زمانی.
  - اندازه‌ی ورودی: تعداد ماتریس‌هایی که باید ضرب شوند.
  - عمل اصلی: دستورالعمل قرار گرفته در درون داخلی‌ترین حلقه.

$$T(n) = \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} \sum_{k=i}^{j-1} 1 = \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} j - i = \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} d = \sum_{d=1}^{n-1} d(n-d) = \frac{n(n-1)(n+1)}{6} \sim \frac{1}{6} n^3$$

# مرحله ۳) محاسبه‌ی راه‌حل بهینه



- محاسبه‌ی ترتیب بهینه برای ضرب ماتریس‌ها.

$A_1$ $A_2$ $A_3$ $A_4$ $A_5$ $A_6$

| 1 | 2 | 3 | 4 | 5 | 6 | P |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 0 | 2 | 3 | 4 | 5 | 2 |
| | | 0 | 3 | 4 | 5 | 3 |
| | | | 0 | 4 | 5 | 4 |
| | | | | 0 | 5 | 5 |
| | | | | | 0 | 6 |

P[1][6] = 1 $A_1\ (A_2\ A_3\ A_4\ A_5\ A_6)$

P[2][6] = 5 $A_1((A_2\ A_3\ A_4\ A_5)A_6)$

P[2][5] = 4 $A_1(((A_2\ A_3\ A_4)A_5)A_6)$

P[2][4] = 4 $A_1((((A_2\ A_3)A_4)A_5)A_6)$

ترتیب بهینه

# تراز بندی بهینه‌ی رشته‌ها

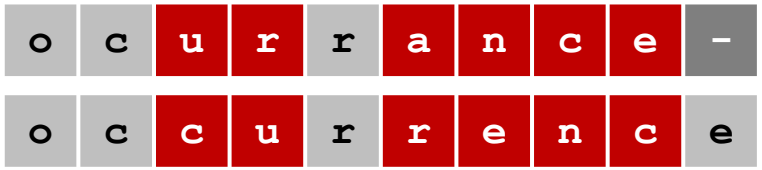
# شباهت دو رشته



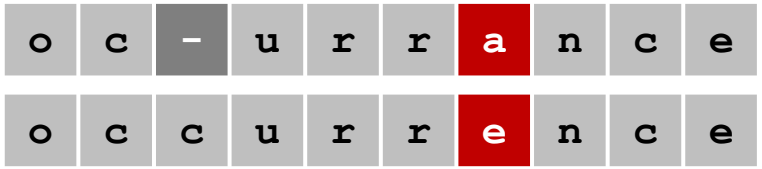
- س. دو رشته‌ی زیر چقدر به یکدیگر شبیه هستند؟

ocurrance

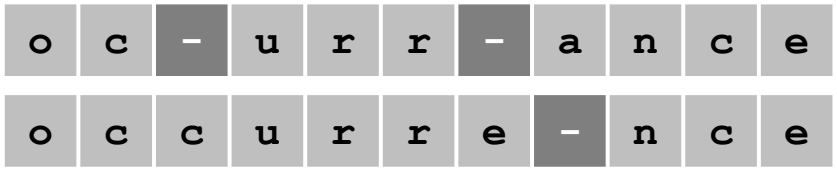
occurrence




۶ عدم تطابق، ۱ فاصله




۱ عدم تطابق، ۱ فاصله




۰ عدم تطابق، ۳ فاصله

# فاصله ویرایش



- کاربردها.
  - تشخیص گفتار
  - زیست‌شناسی محاسباتی
- فاصله ویرایش.
  - جریمه به ازای هر فاصله $\delta$، جریمه به ازای هر عدم تطابق $\alpha_{pq}$
  - هزینه = مجموع جریمه‌های مربوط به فاصله‌ها و عدم تطابق‌ها

| - | C | T | G | A | C | C | T | A | C | C | T |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C | C | T | G | A | C | - | T | A | C | A | T |

$$2\delta + \alpha_{CA}$$

# ترازبندی رشته‌ها



- هدف. یافتن ترازبندی با کمترین هزینه برای دو رشته‌ی X و Y.

- $X = x_1\ x_2\ \dots\ x_m$
- $Y = y_1\ y_2\ \dots\ y_m$

- تعریف. $OPT(i, j)$ = حداقل هزینه به منظور تراز کردن دو زیررشته‌ی زیر:

- $x_1\ x_2\ \dots\ x_i$
- $y_1\ y_2\ \dots\ y_j$

# ترازبندی رشته‌ها



- حالت ۱) تطبیق دادن $x_i$ و $y_j$
  - پرداخت هزینه‌ی تطبیق $x_i$ و $y_j$ به علاوه حداقل هزینه برای تراز کردن دو رشته‌ی زیر:
    - $x_1\ x_2\ \dots\ x_{i-1}$
    - $y_1\ y_2\ \dots\ y_{j-1}$
- حالت ۲ الف) عدم تطبیق $x_i$
  - پرداخت هزینه‌ی عدم تطبیق $x_i$ به علاوه حداقل هزینه برای تراز کردن دو رشته‌ی زیر:
    - $x_1\ x_2\ \dots\ x_{i-1}$
    - $y_1\ y_2\ \dots\ y_j$
- حالت ۲ ب) عدم تطبیق $y_j$
  - پرداخت هزینه‌ی عدم تطبیق $y_j$ به علاوه حداقل هزینه برای تراز کردن دو رشته‌ی زیر:
    - $x_1\ x_2\ \dots\ x_i$
    - $y_1\ y_2\ \dots\ y_{j-1}$

$$OPT(i,j) = \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases}$$

# ترازبندی رشته‌ها: برنامه‌ریزی پویا



```
int Sequence-Alignment(String X, string Y, int m, int n, int delta, int[][] alpha)
{
     for (i = 0; i <= m; i++)
          M[i][0] = delta * i;
     for (j = 0; i <= n; j++)
          M[0][j] = delta * j;
     for (i = 0; i <= m; i++)
          for (j = 0; j <= n; j++)
               M[i][j] = min(alpha[x_i][y_j]  + M[i-1][j-1],
                             delta            + M[i-1][j  ],
                             delta            + M[i  ][j-1]);
     return M[m][n];
}
```

$$OPT(i,j) = \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases}$$

# ترازبندی رشته‌ها: برنامه‌ریزی پویا



- تحلیل. پیچیدگی زمانی و حافظه
  - $T(m,n) \sim mn$
  - $S(m,n) \sim mn$
- کلمات انگلیسی.
  - $m,n \leq 10$
- رشته‌های ژنتیکی.
  - $m = n = 100{,}000$
  - زمان: ده میلیارد عمل
  - حافظه: ده گیگابایت

# ترازبندی رشته‌ها: مصرف حافظه خطی



- س. آیا می‌توان از مصرف حافظه‌ی درجه دوم اجتناب نمود؟

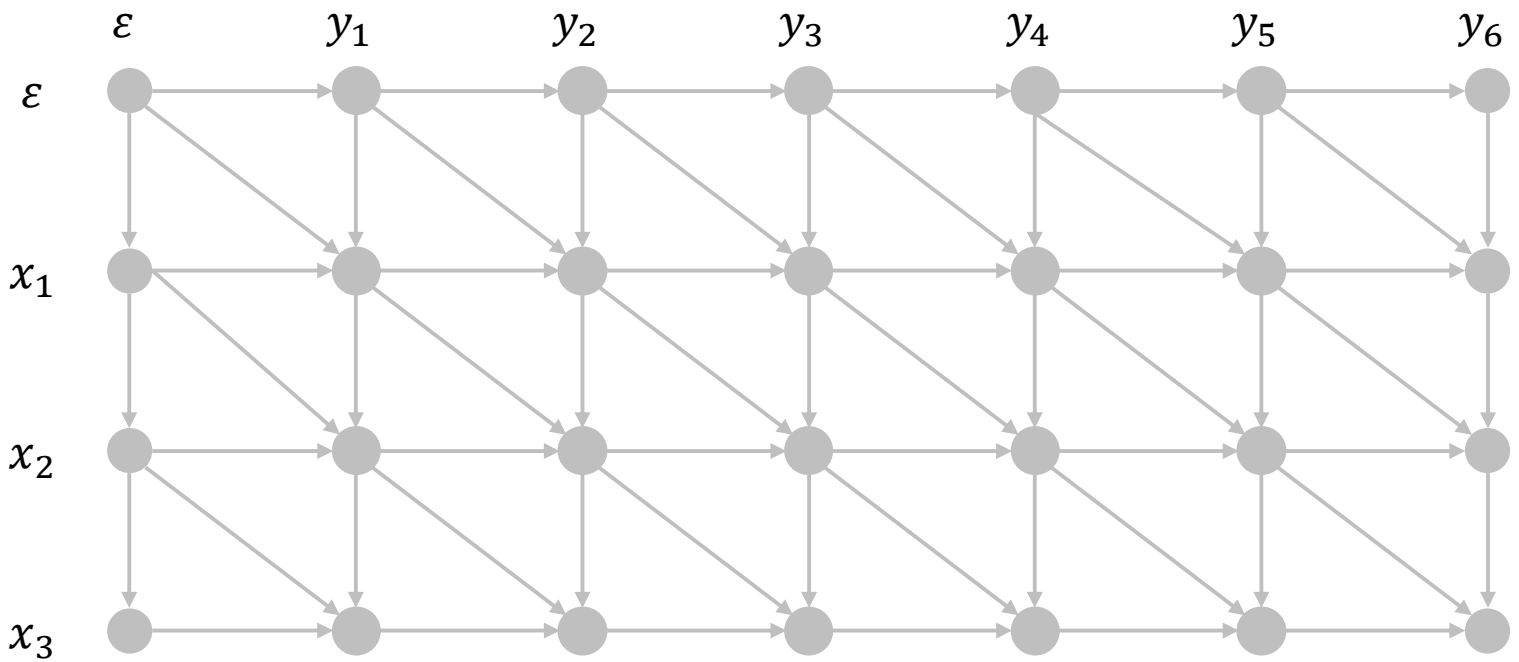
- ج. بله، مقدار راه‌حل بهینه را می‌توان در زمان $O(mn)$ و حافظه‌ی $O(m+n)$ محاسبه نمود.
  - عناصر سطر $i$ تنها از روی عناصر سطر $i-1$ قابل محاسبه هستند.
  - اما در این صورت، هیچ روش ساده‌ای برای محاسبه خود راه‌حل بهینه وجود ندارد.

- قضیه. [هیرشبرگ، ۱۹۷۵] ترازبندی بهینه در زمان $O(mn)$ و حافظه‌ی $O(m+n)$
  - یک ترکیب هوشمندانه از روش تقسیم و حل و برنامه‌ریزی پویا!

# ترازبندی رشته‌ها: مصرف حافظه خطی



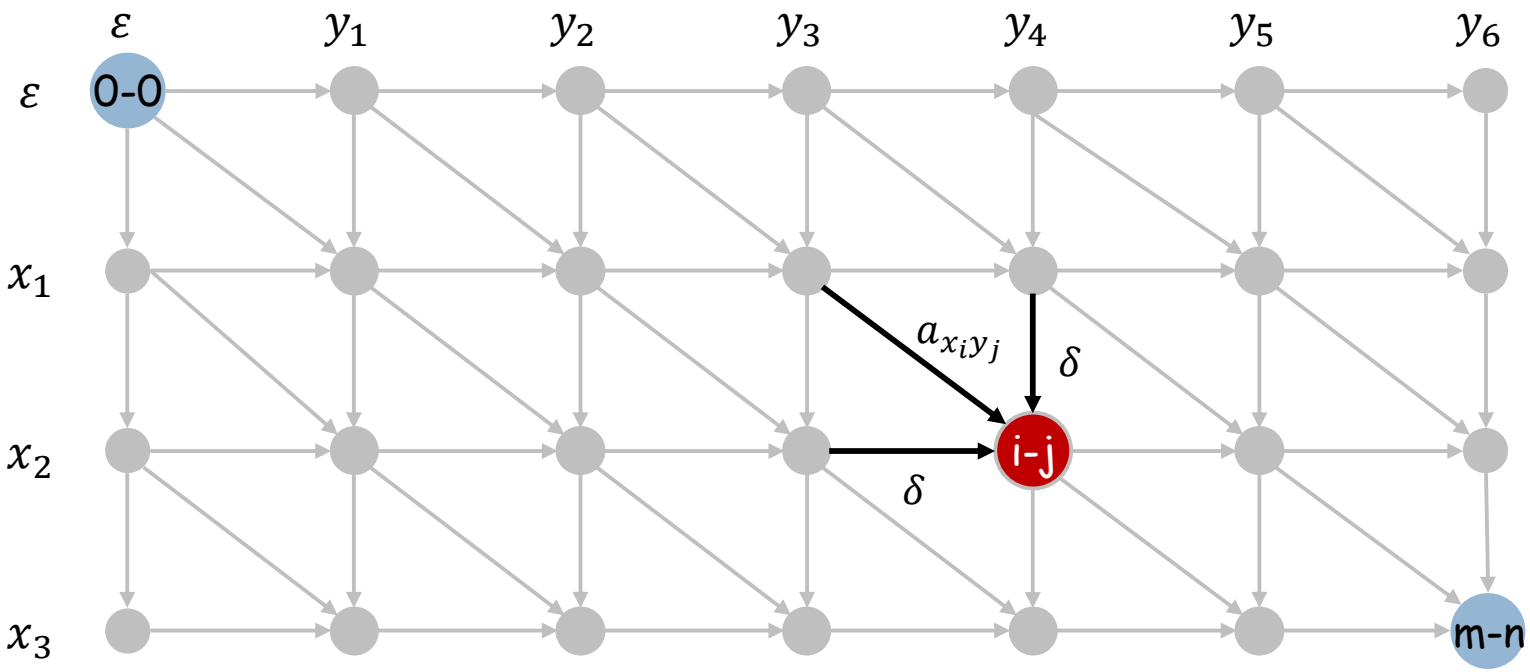
- گراف فاصله ویرایش.
  - فرض کنید `f(i, j)` برابر با کوتاه‌ترین مسیر از رأس `(0, 0)` به رأس `(i, j)` باشد.
  - مشاهده: `f(i, j) = OPT(i, j)`



# ترازبندی رشته‌ها: مصرف حافظه خطی



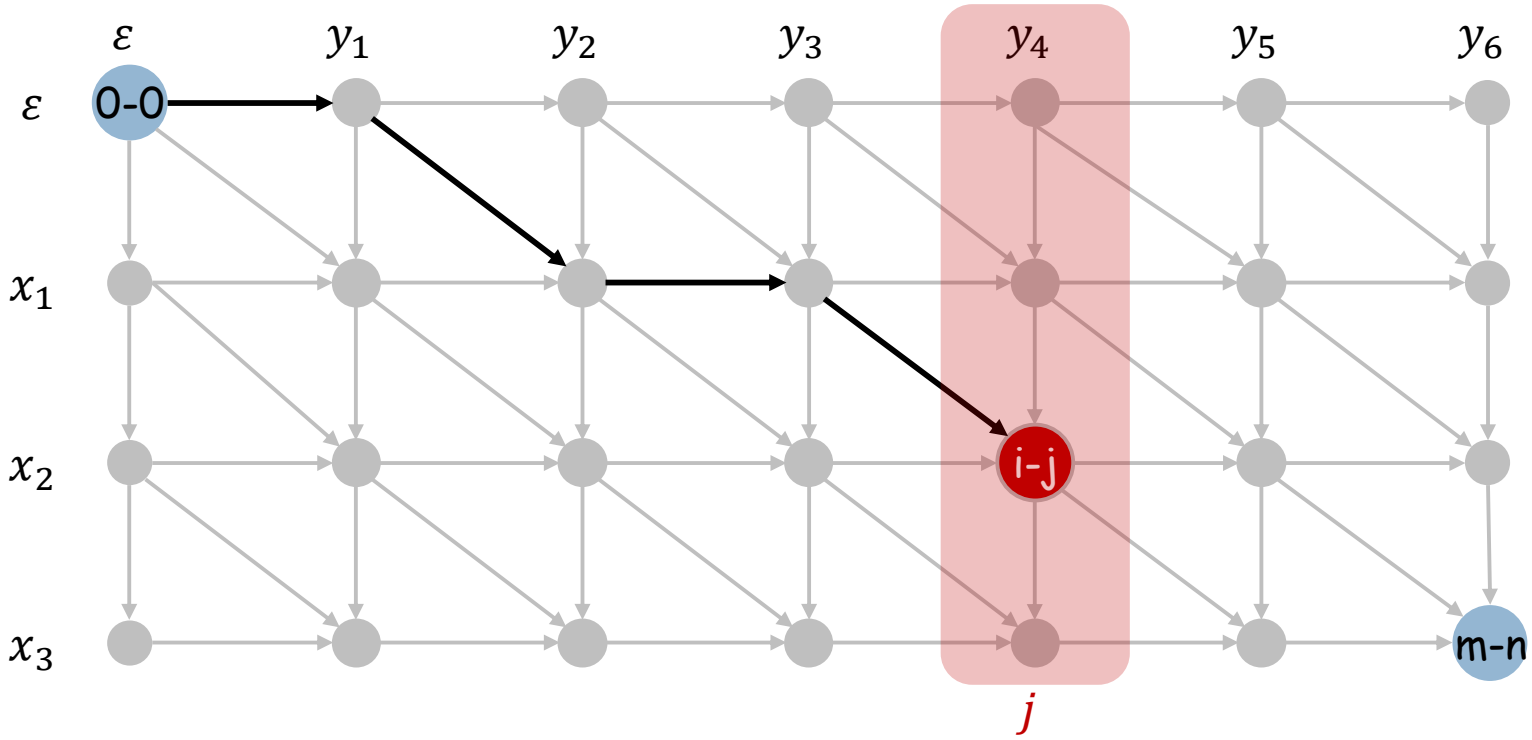
- گراف فاصله ویرایش.
  - فرض کنید `f(i, j)` برابر با کوتاه‌ترین مسیر از رأس `(0, 0)` به رأس `(i, j)` باشد.
  - مشاهده: `f(i, j) = OPT(i, j)`



# ترازبندی رشته‌ها: مصرف حافظه خطی



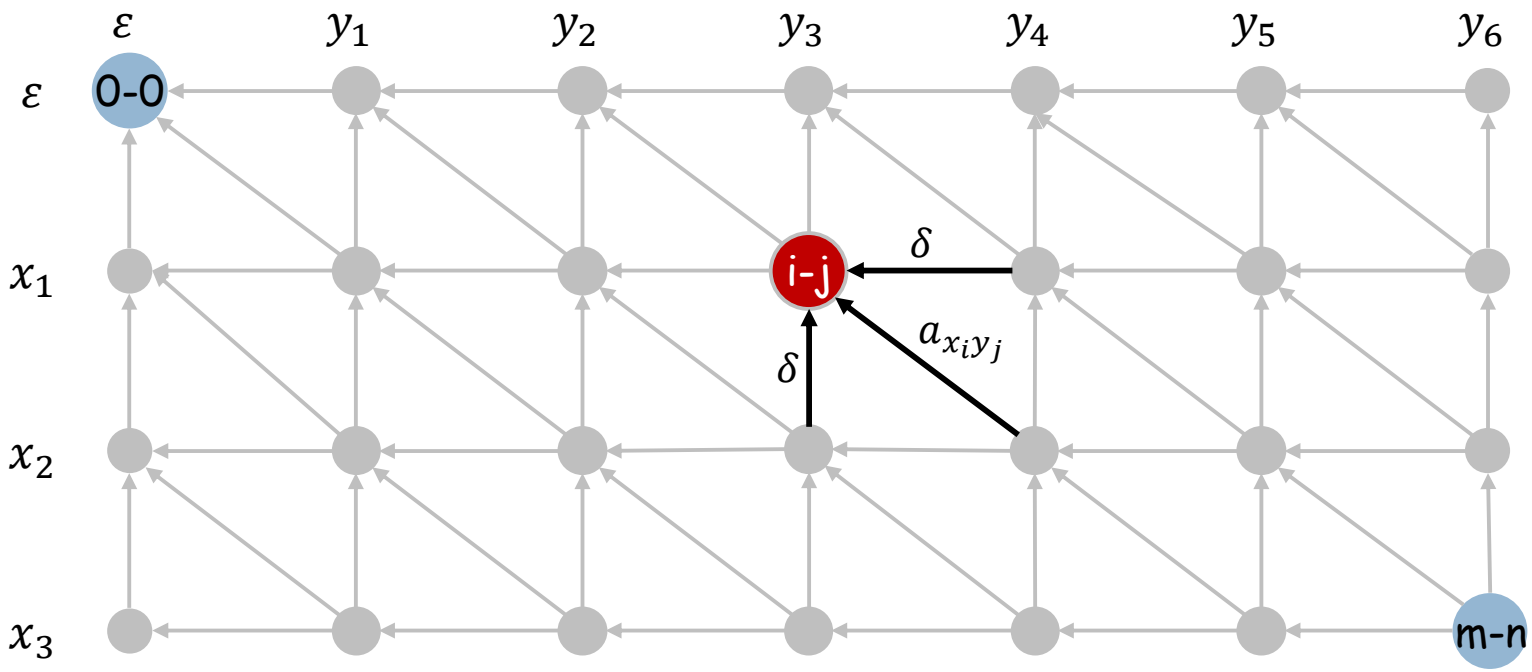
- گراف فاصله ویرایش.
  - فرض کنید `f(i, j)` برابر با کوتاه‌ترین مسیر از رأس `(0, 0)` به رأس `(i, j)` باشد.
  - می‌توان `f(., j)` را برای هر `j` در زمان `O(mn)` و حافظه‌ی `O(m+n)` محاسبه نمود.



# ترازبندی رشته‌ها: مصرف حافظه خطی



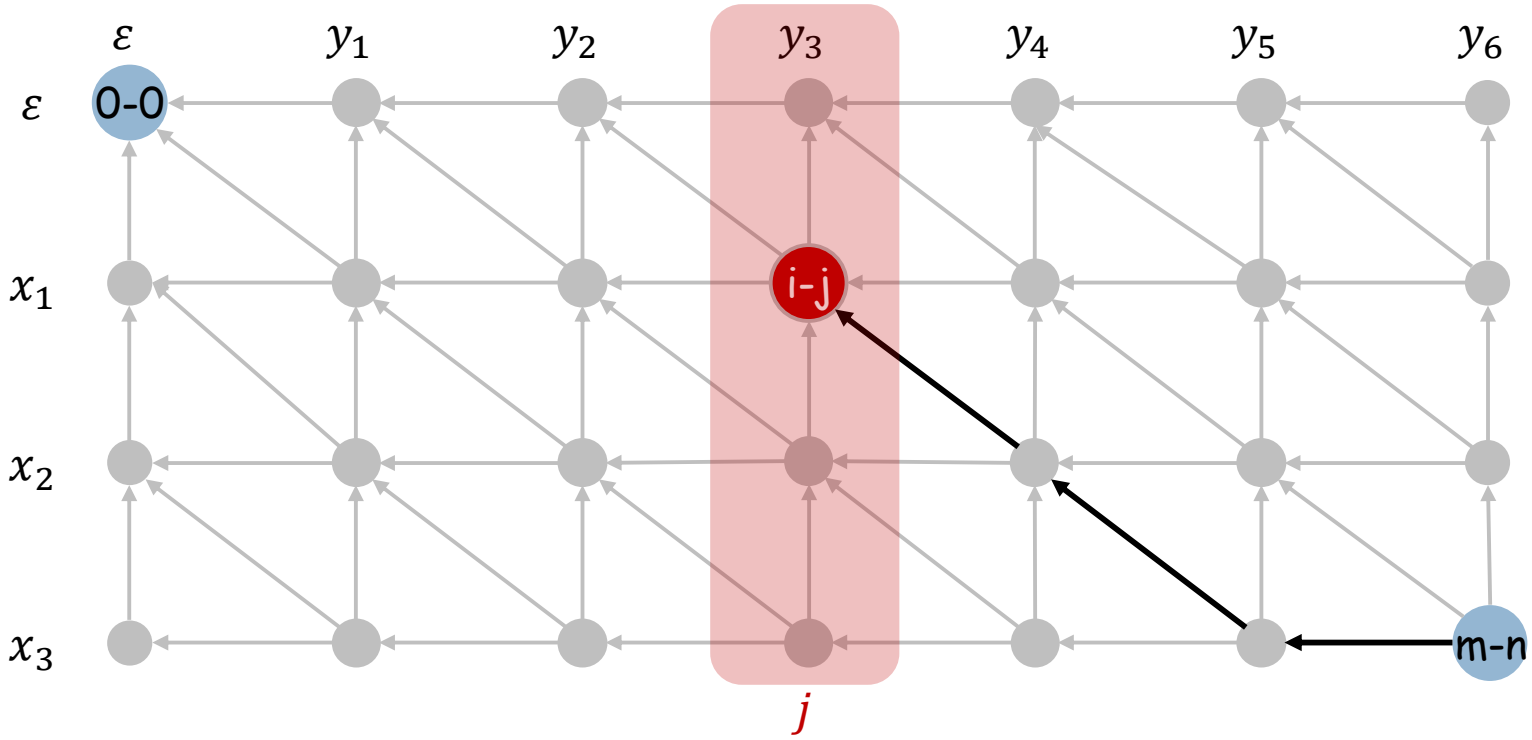
- گراف فاصله ویرایش.
  - فرض کنید `g(i, j)` برابر با کوتاه‌ترین مسیر از رأس `(i, j)` به رأس `(m, n)` باشد.
  - می‌توان `g(i, j)` را با معکوس کردن جهت همه‌ی یالها و تعویض نقش `(0, 0)` با `(m, n)` محاسبه نمود.



# ترازبندی رشته‌ها: مصرف حافظه خطی



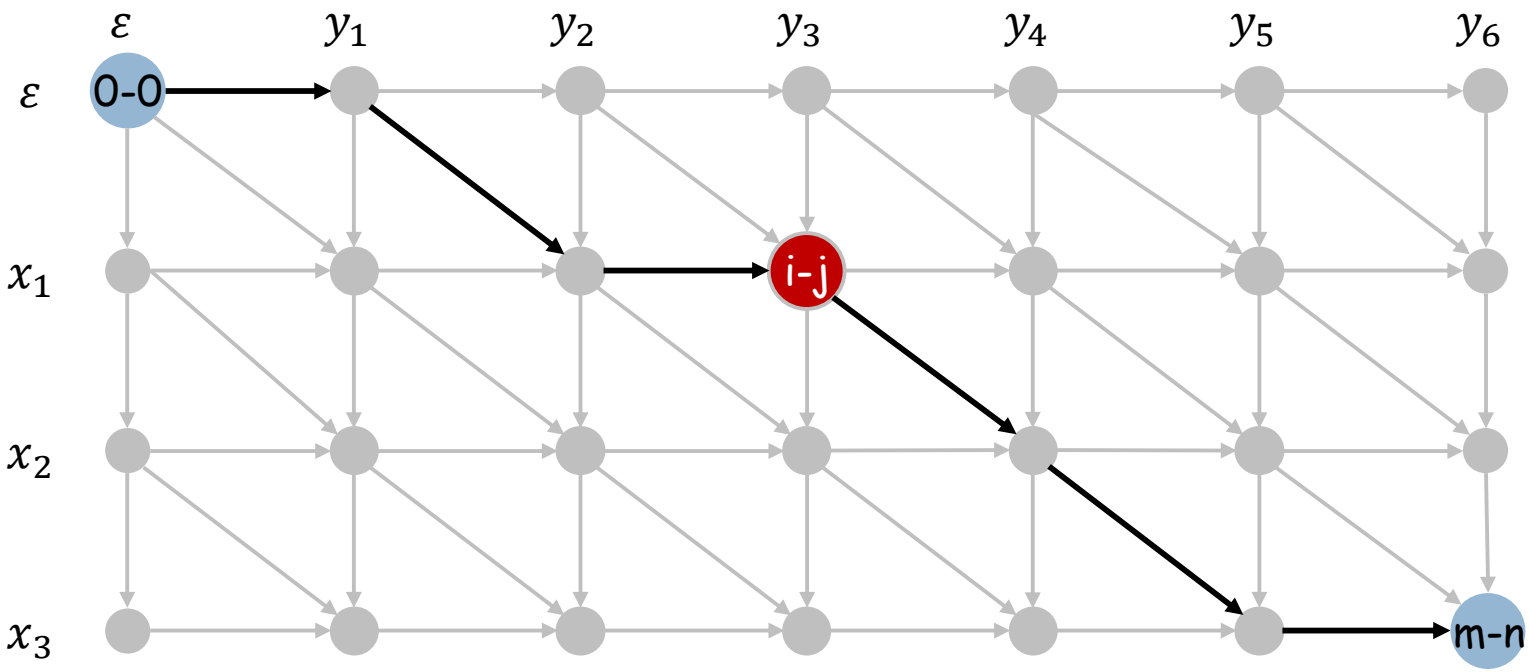
- گراف فاصله ویرایش.
  - فرض کنید `g(i, j)` برابر با کوتاه‌ترین مسیر از رأس `(i, j)` به رأس `(m, n)` باشد.
  - می‌توان `g(., j)` را برای هر `j` در زمان `O(mn)` و حافظه‌ی `O(m+n)` محاسبه نمود.



# ترازبندی رشته‌ها: مصرف حافظه خطی



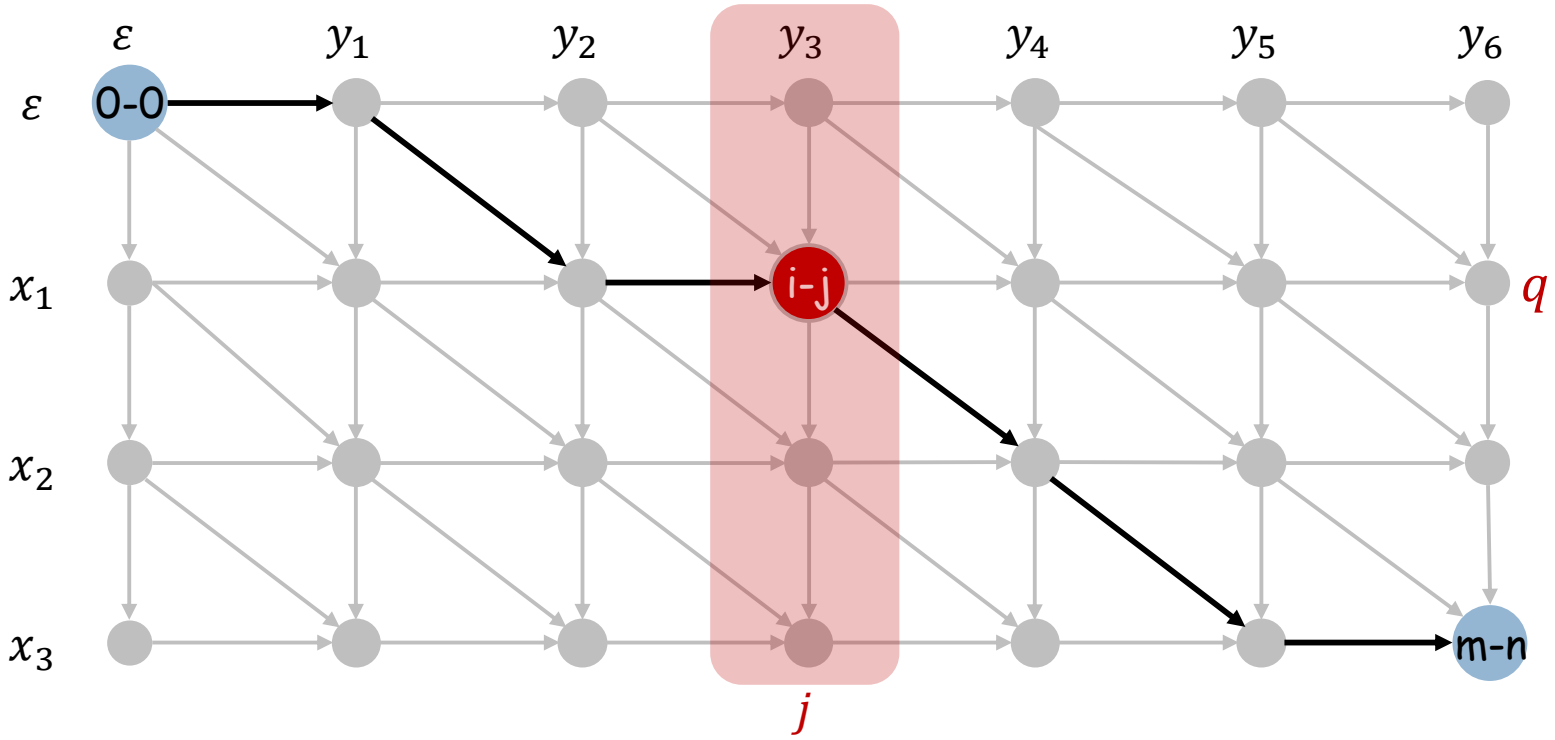
- مشاهده ۱.
  - طول کوتاه‌ترین مسیر که از رأس `(i, j)` می‌گذرد برابر است با `f(i, j) + g(i, j)`



# ترازبندی رشته‌ها: مصرف حافظه خطی



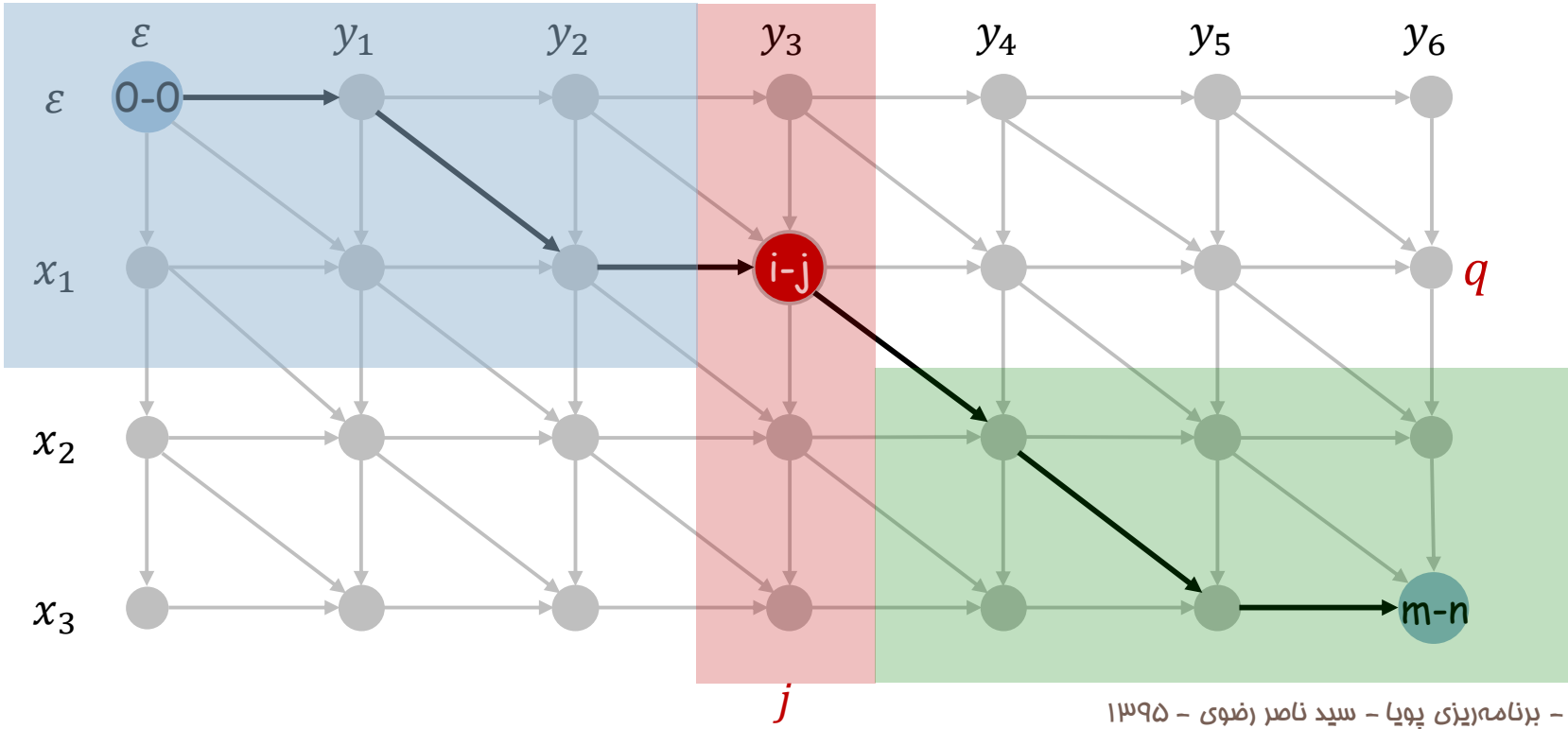
- مشاهده ۲.
  - فرض کنید q اندیسی باشد که `f(q, n/2) + g(q, n/2)` را کمینه می‌کند.
  - در این صورت کوتاه‌ترین مسیر از `(0, 0)` به `(m, n)` از q می‌گذرد.



# ترازبندی رشته‌ها: مصرف حافظه خطی



- تقسیم. با DP اندیس q را پیدا کنید به گونه‌ای که `f(q, n/2) + g(q, n/2)` را کمینه می‌کند.
- دو زیررشته‌ی $x_q$ و $y_{n/2}$ را تراز کنید.
- حل. به صورت بازگشتی ترازبندی بهینه را در هر یک از دو زیرمسئله محاسبه کنید.



# ترازبندی رشته‌ها: مصرف حافظه‌ی خطی



- قضیه. فرض کنید `T(m, n)` حداکثر زمان اجرای این الگوریتم بر روی دو رشته به طول‌های `m` و `n` باشد. در این صورت

`T(m, n) = O(mn)`

- اثبات. [با استقرا بر روی `n`]
  - زمان `O(mn)` به منظور محاسبه‌ی `f(., n/2)` و `g(., n/2)` و یافتن اندیس `q`.
  - زمان `T(n/2, q) + T(m-q, n/2)` برای حل دو زیرمسئله به صورت بازگشتی.
  - ثابت `c` را به گونه‌ای انتخاب کنید که:

$$T(m,2) \le cm$$
$$T(2,n) \le cn$$
$$T(m,n) \le cmn + T(q,n/2) + T(m-q,n/2)$$

  - حالت‌های پایه. `m = 2` یا `n = 2`
  - فرضیه استقرا: `T(m, n) <= 2cmn`

$$\begin{aligned} T(m,n) &\le T(q,n/2) + T(m-q,n/2) + cmn \\ &\le 2cq\ n/2 + 2c(m-q)\ n/2 + cmn \\ &= cqn + cmn - cqn + cmn \\ &= 2cmn \end{aligned}$$