# فصل چهارم
# رویکرد برنامه سازی پویا

**مقدمه: (مهم)**

- به یک رابطه بازگشتی نیاز است که حالات بزرگتر را بر حسب حالات کوچکتر نشان دهد (مشابه تقسیم و حل).
- ابتدا نمونه‌های کوچکتر، حل و نتایج آن‌ها در حافظه در قالب آرایه و . . . ذخیره می‌شود (در موقع نیاز به جای محاسبه دوباره فقط بازیابی می‌شوند)
- یک روش پائین به بالا (Down-Top) است. (بر خلاف روش تقسیم و حل)
- در مسائلی که نمونه‌های کوچکتر با هم ارتباط دارند، از روش پویا استفاده می‌شود (در این مسائل روش تقسیم و حل هزینه زیادی دارد):
- به طور مثال در مسئله فیبوناچی که جملات به هم وابسته‌اند (جمله پنجم به جملات سوم و چهارم نیاز دارد و هر کدام از این جملات نیز به جمله دوم، نیاز دارند. لذا $fib(2)$ بایستی چندین مرتبه محاسبه شود. در روش تقسیم و حل به هر جمله، هر زمان که نیاز باشد، دوباره محاسبه می‌شود ولی در روش پویا یک جمله، یک بار محاسبه و بارها استفاده (بازیابی) می‌شود)
- در مسائلی که نمونه‌های کوچکتر با هم ارتباط ندارند، بهتر است از روش تقسیم و حل استفاده شود. (مانند مرتب سازی ادغام که نمونه‌های کوچکتر با هم ارتباطی ندارند)

## روش پویا (مثال بالا به پائین و پائین به بالا)

**حل رابطه بازگشتی** $T(n)=T(n-1)+3$ **با حالت خاص** $T(1)=1$

مفهوم $T(n)=T(n-1)+3$:

هزینه اجرای یک الگوریتم با ورودی n برابر است با هزینه اجرای الگوریتم با n-1 ورودی به علاوه ۳

**بالا به پایین:**

$$\begin{aligned} T(n) &= T(n-1)+3=(T(n-2)+3)+3=T(n-2)+2\times 3 \\ &= T(n-3)+3\times 3=T(n-4)+4\times 3=\dots \\ &= T(n-(n-1))+(n-1)\times 3=T(1)+(n-1)\times 3=1+(n-1)\times 3 \end{aligned}$$

$$T(n)=\theta(n)$$

**پائین به بالا:**

$$T(1)=1$$
$$T(2)=T(1)+3=1+3=4$$
$$T(3)=T(2)+3=4+3=7$$
$$T(4)=T(3)+3=7+3=10$$
$$T(5)=T(4)+3=10+3=13$$

پس از روند بالا نتیجه میگیریم که

$$T(n)=3n-2$$

پس:

$$T(n)=\theta(n)$$

## الگوریتم‌های نمونه روش پویا

- سری فیبوناچی
- ضرب زنجیره‌ای ماتریس‌ها (حذف)
- الگوریتم فلوید (حذف)
- درخت جستجوی دودویی
- فروشنده دوره‌گرد (حذف)
- ضریب دوجمله‌ای

### سری فیبوناچی

#### با روش تقسیم و حل

الگوریتم:

```
int fib (int n)
{
        if (n <= 1)
                return n;
        else
                return fib (n - 1) + fib (n - 2);
}
```

$$T(n) \approx 2T(n-2) = O\left(2^{\frac{n}{2}}\right), \quad a \neq 1$$

هزینه بسیار بالاست. (در فصل ۱ ارائه شد)

#### با روش پویا

الگوریتم با آرایه

| | | |
|---|---|---|
| int fib (int n) | fib: نام الگوریتم<br>n: شماره جمله مورد نظر در سری فیبوناچی | |
| { | | |
| int f[0 . . n], i; | | |
| f[0] = 0; | | |
| if (n > 0) | | |
| { | | |
| f[1] = 1; | | |
| for (i=2; i <= n; i++) | | این حلقه جملات دوم تا n را محاسبه و در آرایه مربوطه قرار میدهد |
| { | | |
| f[i] = f[i-1] + f[i-2]; | | |
| } | | |
| } | | |
| return f[n]; | جمله nام را برمیگرداند | |
| } | | |

الگوریتم (بدون آرایه: برای صرفه جویی در حافظه)

| | |
|---|---|
| int fib (int n) | |
| { | |
| int f1, f2, f; | |
| f1 = 0; | |
| f2 = 1; | |
| if (n == 0) | |
| { | |
| return f1; | |
| } | |
| else if (n == 1) | |
| { | |
| return f2; | |
| } | |
| for (i == 2; i <= n; i++) | |
| { | |
| f = f1 + f2; | |
| f1 = f2; | |
| f2 = f; | |
| } | |
| retrun f; | جمله n ام برگردانده می‌شود |
| } | |

تذکر:

الگوریتم دوم، نسبت به الگوریتم اول، از نظر پیچیدگی زمانی، هیچ تفاوتی ندارد. تفاوت آن فقط صرفه جویی در حافظه است.

در الگوریتم دوم، فقط ۳ مکان حافظه مصرف می‌شود. (f1, f2, f) در حالیکه در الگوریتم اول n مکان مصرف می‌شود.

**مرتبه اجرایی با روش پویا برای سری فیبوناچی**

$$T(n)=O(n)$$

## الگوریتم ضرب دو جمله‌ای نیوتن

**تعریف ریاضی**

$$\binom{n}{k}=\frac{n!}{k!(n-k)!} \qquad for \quad 0\le k\le n$$

**تعریف بازگشتی**

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

**روش تقسیم و حل**

int: نوع مقدار برگشتی

bin: نام الگوریتم

دو پارامتر n و k دارد.

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

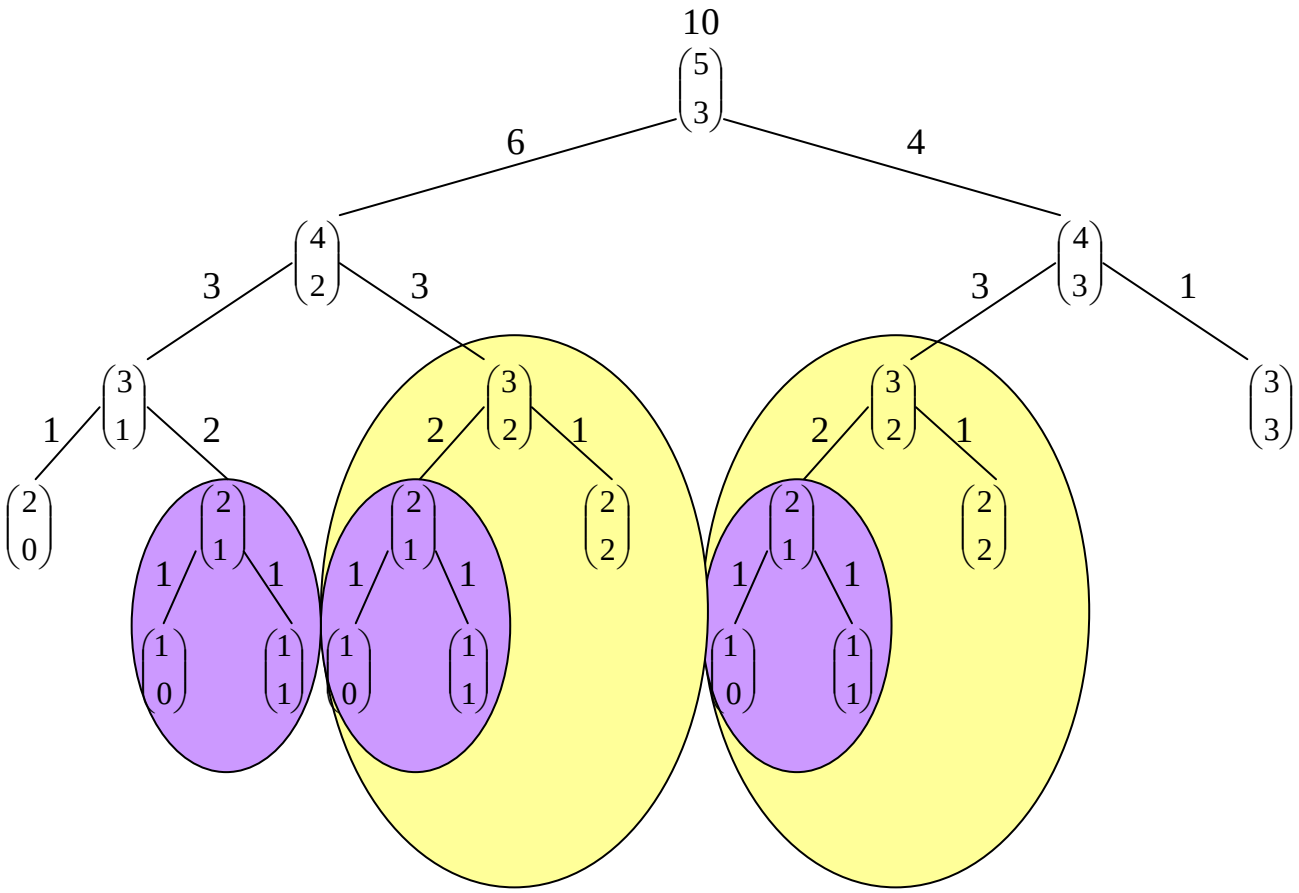
**هزینه با این روش**

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

این هزینه وحشتناک است، زیرا هزینه ۲ برابر منهای ۱ می‌شود.

**مثال**

محاسبه $C(5,3)$




همانطور که از شکل مشخص است، علت افزایش هزینه به صورت نمایی، وجود گره‌های تکراری در محاسبات می‌باشد. به عنوان مثال $\binom{2}{1}$ ، ۳ بار و $\binom{3}{2}$ ، ۲ بار تکرار شده است. راه حل استفاده از روش برنامه سازی پویا است.

**روش برنامه نویسی پویا**

استفاده از یک آرایه (ماتریس) $B$ به منظور ذخیره کردن ضرایب

مراحل:

بنا نهادن یک خاصیت بازگشتی

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

حل یک نمونه مسئله به روش پایین به بالا با محاسبه نمودن سطرهای B به طور متوالی با شروع از سطر اول

**پیاده‌سازی و مثال**

آرایه B

| | 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 | | | | | | | $B[i-1][j-1]$ | $B[i-1][j]$ |
| n | | | | | | | | + $B[\overbrace{i}^{n}][\overbrace{j}^{k}]$ |

محاسبه $B[4][2]$:

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

نکته:

هر گاه مقدار i از j کوچکتر شود جواب آرایه B برابر صفر خواهد بود.

هر گاه مقدار j برابر صفر باشد جواب آرایه B برابر یک خواهد بود.

هر گاه مقدار i با j برابر باشد جواب آرایه B برابر یک خواهد بود.

**الگوریتم مسئله ضریب دو جمله‌ای با استفاده از روش برنامه سازی پویا**

| | |
|---|---|
| int bin2 (int n, int k) | تعداد سطرها n و تعداد ستون‌ها k |
| index i, j; | |
| int B[0..n] [0..k]; | تعداد سطرهای آرایه، تعداد ستون‌های آرایه |
| for (i = 0; i <= n; i++) | |
| for (j = 0; j <= minimum (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**

تعداد گذرها از حلقه j به ازاء هر مقدار از i

| i | 0 | 1 | 2 | 3 | . . . | k | k + 1 | . . . | n |
|---|---|---|---|---|---|---|---|---|---|
| تعداد گذرها | 1 | 2 | 3 | 4 | . . . | k + 1 | k + 1 | . . . | k + 1 |

$$1+2+3+4+\ldots+k+\underbrace{(k+1)+(k+1)+\ldots+(k+1)}_{\text{مرتبه } n-k+1}=\frac{k(k+1)}{2}+(n-k+1)(k+1)=\frac{(2n-k+2)(k+1)}{2}\in\theta(nk)$$

پس:

$$T(n)=\theta(nk)$$

با فرض $n\cong k$ پس: $T(n)=\theta(n^2)$ زیرا ۲ تا حلقه تو در تو داریم.