# برنامه نویسی پویا

## ایده کلی روش برنامه‌نویسی پویا

روش برنامه‌نویسی پویا مشابه روش تقسیم و حل، مساله با استفاده از نمونه‌های کوچکتر، حل‌می‌شود. با این تفاوت که نخست، نمونه‌های کوچکتر حل‌شده و نتایج برای استفاده‌های بعدی، ذخیره‌می‌شود. برای این منظور، برنامه‌نویسی پویا شامل دو گام اصلی است:

۱- ایجاد رابطه (ویژگی) بازگشتی

۲- حل نمونه مساله از پایین به بالا(جزء به کل)

☜ **پرسش:** واژه برنامه‌نویسی به چه چیزی اشاره‌دارد؟

The term ***Dynamic Programming*** comes from **Control Theory**, not computer science.
Programming refers to the use of **tables (arrays)** to construct a solution. Used extensively in "Operation Research" given in the Math dept.

## ویژگی‌های روش برنامه‌نویسی پویا

- In dynamic programming we usually reduce *time* by increasing the amount of *space*
- We solve the problem by solving sub-problems of increasing size and saving each optimal solution in a table (usually).
- The table is then used for finding the optimal solution to larger problems.
- Time is saved since each sub-problem is solved only once.

**مثال:** سری فیبوناچی (Fibonacci's Series)

T(n) = T(n-1) + T(n-2) + 1 if n >= 2
T(2) =1, T(1) = T(0) = 0

**روش بازگشتی** (روش تقسیم و حل)

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

**Alg. 5-1**

$T(n) = O(2^n)$ , $T(n) = \Omega(2^{n/2})$

*Builds a table with the first n Fibonacci numbers.*

**روش برنامه‌نویسی پویا** (تکراری)

```
int fib(n)
{
        A[0] = 0
        A[1] = 1
        for i ← 2 to n
                do A[ i ] = A [i -1] + A[i -2 ]
   return A[n];
}
```

**Alg. 5-2**

☜ **پرسش:** پیچیدگی زمانی و مکانی الگوریتم روش برنامه‌نویسی پویا برای مساله فیبوناچی چیست؟ با استفاده از روش معادله مشخصه، مرتبه زمانی الگوریتم بازگشتی برای مساله فیبوناچی، محاسبه کنید.

**مثال**: ضرایب دوجمله‌ای(Binomial Coefficient)

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

$$(x+y)^1 = x+y$$

$$(x+y)^2 = x^2 + 2xy + y^2$$

$$(x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$$

$$(x+y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$$

.....

$$C_k^n = \binom{n}{k} = \frac{n!}{k!(n-k)!} \quad \text{if} \quad 0 < k < n$$

**فرم بازگشتی**

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

## راه حل برنامه‌نویسی پویا (الگوریتم تکراری)

**1-** Establish a recursive property.
Use a matrix B of n+1 rows, k+1 columns where

$$B[n,k] = \binom{n}{k}$$

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

**2-** Solve an instance of the problem in a *bottom-up* fashion by computing the rows in B.

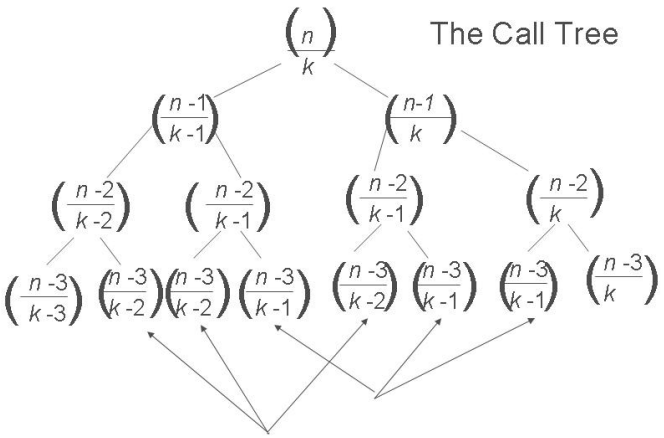
```
int bin2 (int  n, int  k)
{
  index i, j ;
   int  B[0..n][0..k];
   for ( i = 0; i ≤ n; i++)
    {
        for ( j = 0; j ≤ min(i, k) ; j++)
         {
             if ( j= 0 || j=i )
                B[i][j] = 1;
             else
                B[i][j] = B[i–1][j–1] + B[i–1][j];
         }
    }
   return B[n][k] ;
}
```

**Alg. 5-4**

## راه حل تقسیم و حل (الگوریتم بازگشتی)

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

**Alg. 5-3**




**شکل۵-۱:** درخت فراخوانی برای روش تقسیم و حل

## آنالیز الگوریتم برای راه حل پویا برای مساله محاسبه ضرائب دو جمله‌ای

$$T(n) = 1+2+3+\cdots+k+\underbrace{(k+1)+(k+1)+\cdots+(k+1)}_{n-k+1 \text{ times}}$$
$$= \frac{k(k+1)}{2} + (k+1)(n-k+1)$$
$$= \frac{(2n-k+2)(k+1)}{2} \in \Theta(nk)$$

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1

## الگوریتم فلوید (Floyd's Algorithm for Shortest Paths)

### گذری بر تئوری گراف (یادآوری از درس ساختمان داده‌ها)

**Vertex:** *represented by a circle in a graph.*
**Edge:** *a line from one vertex to another.*
**Graph:** *denoted as G(V,E).*
**Directed Graph:** *the edges have a direction.*
**Weighted Graph:** *each edge has a weight associated with it.*

**همسایگی**

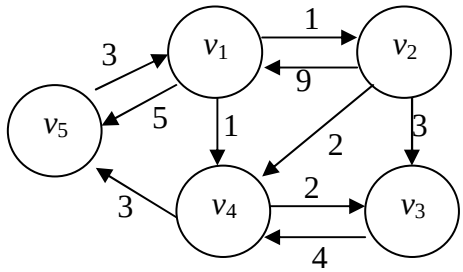
$\mathbf{v_i}$ *is said to be* ***adjacent*** *to* $v_j$ *if there is an edge from* $\mathbf{v_i}$ *to* $\mathbf{v_j}$.

**ماتریس همسایگی** (Adjacent Matrix)

*A graph can be represented by the* **Adjacent Matrix**, *W[i][j].*

$$W[i][j] = \begin{cases} \textit{weight on edge}, & \text{if there is an edge from } v_i \text{ to } v_j \\ \infty, & \text{if there is no edge from } v_i \text{ to } v_j \\ 0, & \text{if } i = j. \end{cases}$$

**مثال**




**شکل۵-۲:** گراف مثال

$$W = \begin{bmatrix} 0 & 1 & \infty & 1 & 5 \\ 9 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$

### بخش‌های اصلی الگوریتم فلوید

**Problem:** All sources Shortest paths.
**Inputs:** A weighted, directed graph with n vertices and adjacent matrix W.
**Output:** A two-dimensional array D.

### هدف اصلی الگوریتم فلوید

**Shortest Paths problem** *is an optimization problem: a candidate solution is a path from one vertex to another, the value is the length of the path, and the optimal value is the minimum of these lengths.*

$$D^{(k)}[i][j] = \text{length of a shortest path from } v_i \text{ to } v_j \text{ using only vertices in the set } \{v_1, v_2, \cdots, v_k\} \text{ as intermediate vertices.}$$

```
void floyd (int n, const number W[][], number D[][] )
 {
   index  i, j, k ;
    D = W;
    for ( k = 1; k <= n; k++)
        for ( I = 1; I <= n; i++)
            for ( j = 1; j <= n; j++)
              D [i][j]= min{D [i][j], D [i][k]+ D [k][j]};
 }
```

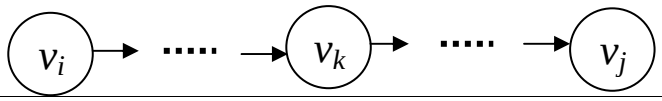
**Alg. 5-5**

**پیچیدگی زمانی** $T(n) = n^3 \in \Theta(n^3)$

## مراحل اصلی الگوریتم فلوید(برنامه‌نویسی‌پویا)

**1.** Establish a **recursive property** with which we can compute $D^{(k)}$ from $D^{(k-1)}$.
**2.** Solve an instance of the problem in a **bottom-up** fashion by representing the process for $k = 1$ to $n$.

$$D^{(k)}[i][j] = \min\left\{ \underbrace{D^{(k-1)}[i][j]}_{\text{Case 1}},\ \underbrace{D^{(k-1)}[i][k] + D^{(k-1)}[k][j]}_{\text{Case 2}} \right\}$$




## مراحل مختلف الگوریتم فلوید روی مثال

$$D^{(0)} = \begin{bmatrix} 0 & 1 & \infty & 1 & 5 \\ 9 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix} \quad D^{(1)} = \begin{bmatrix} 0 & 1 & \infty & 1 & 5 \\ 9 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 4 & \infty & 4 & 0 \end{bmatrix} \quad D^{(2)} = \begin{bmatrix} 0 & 1 & 4 & 1 & 5 \\ 9 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 4 & 7 & 4 & 0 \end{bmatrix}$$

$$D^{(3)} = \begin{bmatrix} 0 & 1 & 4 & 1 & 5 \\ 9 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 4 & 7 & 4 & 0 \end{bmatrix} \quad D^{(4)} = \begin{bmatrix} 0 & 1 & 3 & 1 & 4 \\ 9 & 0 & 3 & 2 & 5 \\ \infty & \infty & 0 & 4 & 7 \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 4 & 6 & 4 & 0 \end{bmatrix} \quad D^{(5)} = \begin{bmatrix} 0 & 1 & 3 & 1 & 4 \\ 8 & 0 & 3 & 2 & 5 \\ 10 & 11 & 0 & 4 & 7 \\ 6 & 7 & 2 & 0 & 3 \\ 3 & 4 & 6 & 4 & 0 \end{bmatrix}$$

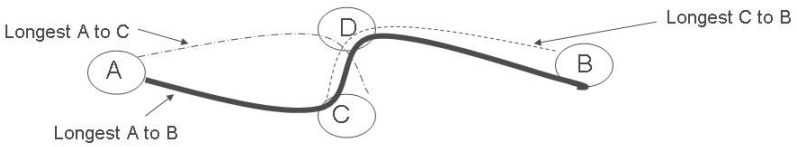
## اصل بهینگی(Principle of Optimality)

- اصل بهینگی روی یک مساله اعمال می‌شود و نه روی یک الگوریتم.
- تعداد زیادی از مسائل بهینه‌سازی، این اصل را برآورده‌می کنند.

**Principle of Optimization**
The *principle of optimality* is said to apply in a problem if an optimal solution to an instance of a problem always contains optimal solutions to all sub-instances.

**Dynamic Programming Algorithm**
1- Establish a recursive property that gives the optimal solution to an instance of the problem.
2- Compute the value of an optimal solution in a bottom-up fashion.
3- Construct an optimal solution in a bottom-up fashion.

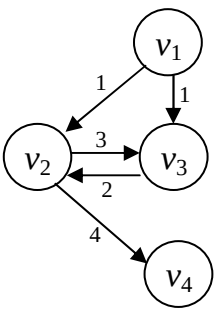
**پرسش:** آیا اصل بهینگی برای هر مساله بهینه‌سازی قابل اعمال است؟




**شکل۵-۳:** یافتن طولانی ترین مسیر بین دو نقطه

**مثال:** طولانی‌ترین مسیر از $v_1$ به $v_4$ در گراف شکل۵-۴، چیست؟ آیا در این مساله، اصل بهینگی برقرار است؟ چرا؟

$v_1$ to $v_4$ : $[v_1, v_3, v_2, v_4]$
$v_1$ to $v_3$ : $[v_1, v_2, v_3]$




**شکل۵-۴:** گراف مثال

## تمرین‌ها

**تمرین۵-۱:** مرتبه پیچیدگی زمانی الگوریتم۵-۳ (Alg. 5-3) را محاسبه کنید.