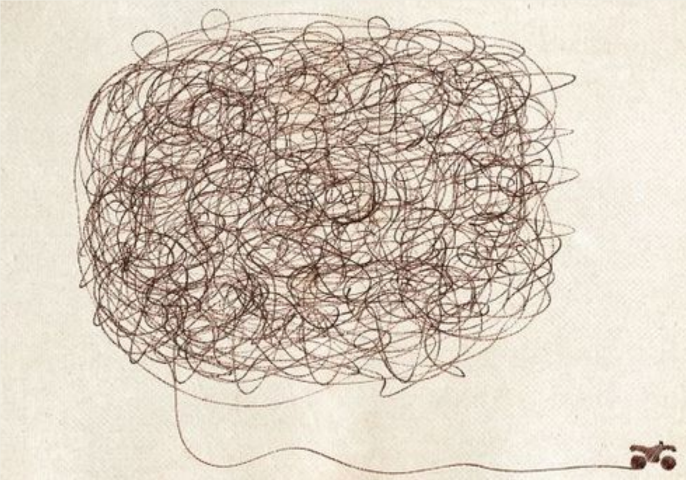
本节主题：

算法分析实例

算法复杂性分析方法

- 时间复杂度的度量
  - $T(n) = O(f(n))$
  - n：问题规模
  - $f(n)$：基本运算次数的量级
- 步骤
  - 确定基本运算
  - 构造$f(n)$
- 提示
  - 只需要确定量级，不必精确计算



例1：方阵相加

- 分析求两个n阶方阵的相加（C=A+B）的算法的时间复杂度

```
#define M   20   //定义最大的方阶
void matrixadd(int n,int A[M][M],int B[M][M],int C[M][M])
{
    int i,j;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            C[i][j]=A[i][j]+B[i][j];
}
```

$$A=\begin{bmatrix}1&2&3\\4&5&5\\7&8&9\end{bmatrix}\qquad B=\begin{bmatrix}4&2&1\\7&5&3\\9&8&6\end{bmatrix}$$

$$C=\begin{bmatrix}5&4&4\\11&10&8\\16&16&15\end{bmatrix}$$

- 基本运算：C[i][j]=A[i][j]+B[i][j]
- 基本运算的的频度：

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}1=\sum_{i=0}^{n-1}n=n\sum_{i=0}^{n-1}1=n*n=n^2$$

- 时间复杂度：$T(n)=O(n^2)$

例2：计数

```
int fun(int n)
{
   int i,j,k,s;
   s=0;
   for (i=0; i<=n; i++)
      for (j=0; j<=i; j++)
         for (k=0; k<=j; k++)
            s++;
   return(s);
}
```

- 基本运算： s++
- 基本运算的的频度： $\sum_{i=0}^{n}\sum_{j=0}^{i}\sum_{k=0}^{j}1$
- 时间复杂度： $T(n)=O(n^3)$

例3：递归算法的复杂度

```
void fun(int a[],int n,int k)
{
   int i;
   if (k==n-1)
      for (i=0; i<n; i++)
         printf("%d\n",a[i]);
   else
   {
      for (i=k; i<n; i++)
         a[i]=a[i]+i*i;
      fun(a,n,k+1);
   }
}
```

调用fun(a,n,0)

设fun(a,n,k)的执行时间为$T_1(n,k)$，则fun(a,n,0)的执行时间为$T(n)=T_1(n,0)$

$$T_1(n,k)=\begin{cases} n & \text{当}k=n-1\text{时} \\ (n-k)+T_1(n,k+1) & \text{其他情况} \end{cases}$$

则：$T(n)$

$= T_1(n,0)$

$= n + T_1(n,1)$

$= n + (n-1) + T_1(n,2)$

$= \ldots = n + (n-1) + \ldots + 2 + T_1(n, n-1)$

$$=\frac{(n+2)(n-1)}{2}+n=\frac{n^2}{2}+\frac{3n}{2}-1$$

$=O(n^2)$

算法空间复杂度分析

- 空间复杂度是对一个算法在运行过程中临时占用的存储空间大小的量度
- 空间复杂度一般也作为问题规模n的函数，以数量级形式给出，记作：S(n) = O(g(n))
  - 若所需额外空间相对于输入数据量来说是常数，则称此算法为原地工作——O(1)
  - 若所需存储量依赖于特定的输入，则通常按最坏情况考虑。
- 例

```
#define M   20   //定义最大的方阶
void matrixadd(int n,int A[M][M],int B[M][M],int C[M][M])
{
   int i,j;
   for (i=0; i<n; i++)
      for (j=0; j<n; j++)
         C[i][j]=A[i][j]+B[i][j];
}
```

解：

算法中临时变量的个数与问题规模n无关，所以空间复杂度为O(1)。