**1、用计算机求解问题的步骤：**

1、问题分析 2、数学模型建立 3、算法设计与选择 4、算法指标 5、算法分析 6、算法实现 7、程序调试 8、结果整理文档编制

**2、算法定义：**算法是指在解决问题时，按照某种机械步骤一定可以得到问题结果的处理过程

**3、算法的三要素**

1、操作 2、控制结构 3、数据结构

**算法具有以下 5 个属性：**

有穷性：一个算法必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。

确定性：算法中每一条指令必须有确切的含义。不存在二义性。只有一个入口和一个出口

可行性：一个算法是可行的就是算法描述的操作是可以通过已经实现的基本运算执行有限次来实现的。

输入：一个算法有零个或多个输入，这些输入取自于某个特定对象的集合。

输出：一个算法有一个或多个输出，这些输出同输入有着某些特定关系的量。

**算法设计的质量指标：**

正确性：算法应满足具体问题的需求；

可读性：算法应该好读，以有利于读者对程序的理解；

健壮性：算法应具有容错处理，当输入为非法数据时，算法应对其作出反应，而不是产生莫名其妙的输出结果。

效率与存储量需求：效率指的是算法执行的时间；存储量需求指算法执行过程中所需要的最大存储空间。一般这两者与问题的规模有关。

**经常采用的算法主要有迭代法、分而治之法、贪婪法、动态规划法、回溯法、分支限界法**

**迭代法**

**基本思想：**迭代法也称“辗转法”，是一种不断用变量的旧值递推出新值的解决问题的方法。

**解题步骤：**1、确定迭代模型。根据问题描述，分析得出前一个（或几个）值与其下一个值的迭代关系数学模型。

2、建立迭代关系式。迭代关系式就是一个直接或间接地不断由旧值递推出新值的表达式，存储新值的变量称为迭代变量

3、对迭代过程进行控制。确定在什么时候结束迭代过程，这是编写迭代程序必须考虑的问题。不能让迭代过程无休止地重复执行下去。迭代过程的控制通常可分为两种情况：一种是所需的迭代次数是个确定的值，可以计算出来；另一种是所需的迭代次数无法确定。对于前一种情况，可以构建一个固定次数的循环来实现对迭代过程的控制；对于后一种情况，需要进一步分析出用来结束迭代过程的条件。

编写计算斐波那契（Fibonacci）数列的第 n 项函数 fib（n）。

写成递归函数有：

```
int fib(int n)
{ if (n==0) return 0;
if (n==1) return 1;
if (n>1) return fib(n-1)+fib(n-2);
}
```

一个饲养场引进一只刚出生的新品种兔子，这种兔子从出生的下一个月开始，每月新生一只兔子，新生的兔子也如此繁殖。如果所有的兔子都不死去，问到第 12 个月时，该饲养场共有兔子多少只？

```
Main()
{int I,a=1,b=1;
Print(a,b);
For(i=1;i<=10;i++)
{
C=a+b;
Print(c);
A=b; B=c; }}
```

**分而治之法**

**1、基本思想**

将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。

分治法所能解决的问题一般具有以下几个特征：

（1）该问题的规模缩小到一定的程度就可以容易地解决；

（2）该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质；

（3）利用该问题分解出的子问题的解可以合并为该问题的解；

（4）该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子子问题。

**3、分治法的基本步骤**

（1）分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题；

（2）解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题；

（3）合并：将各个子问题的解合并为原问题的解。

**贪婪法**

**基本思想：**以逐步的局部最优，达到最终的全局最优。无后效性

**【问题】　背包问题**

问题描述：有不同价值、不同重量的物品 n 件，求从这 n 件物品中选取一部分物品的选择方案，使选中物品的总重量不超过指定的限制重量，但选中物品的价值之和最大。

```
#include<stdio.h>
void main()
{
    int m,n,i,j,w[50],p[50],pl[50],b[50],s=0,max;
    printf("输入背包容量 m,物品种类 n :");
    scanf("%d %d",&m,&n);
    for(i=1;i<=n;i=i+1)
    {
        printf("输入物品的重量 W 和价值 P :");
        scanf("%d %d",&w[i],&p[i]);
        pl[i]=p[i];
        s=s+w[i]; }
    if(s<=m)
    {printf("whole choose\n");
    //return; }

    for(i=1;i<=n;i=i+1)
    {
        max=1;
        for(j=2;j<=n;j=j+1)
        if(pl[j]/w[j]>pl[max]/w[max])
        max=j;
        pl[max]=0;
        b[i]=max;}
    for(i=1,s=0;s<m && i<=n;i=i+1)
    s=s+w[b[i]];
    if(s!=m)
    w[b[i-1]]=m-w[b[i-1]];
    for(j=1;j<=i-1;j=j+1)
    printf("choose weight %d\n",w[b[j]]);}
```

**}**

**}**

**动态规划**

**基本思想：**把求解的问题分成许多阶段或多个子问题，然后按顺序求解各个子问题。前一个子问题的解为后一个子问题的求解提供了有用的信息。在求解任何一子问题时，列出各种可能的局部解，通过决策保留那些有可能达到最优的局部解，丢弃其他局部解，依次解决各子问题，最后一个子问题就是问题的解。

**基本步骤**

（1）划分阶段：按照问题的时间或空间特征，把问题分为若干个阶段。注意这若干个阶段一定要是有序的或者是可排序的（即无后向性），否则问题就无法用动态规划求解。

（2）选择状态：将问题发展到各个阶段时所处于的各种客观情况用不同的状态表示出来。当然，状态的选择要满足无后效性。

（3）确定决策并写出状态转移方程：之所以把这两步放在一起，是因为决策和状态转移有着天然的联系，状态转移就是根据上一阶段的状态和决策来导出本阶段的状态。所以，如果我们确定了决策，状态转移方程也就写出来了。但事实上，我们常常是反过来做，根据相邻两段的各状态之间的关系来确定决策。

标准动态规划的基本框架

**1. 对 $f_{n+1}(x_{n+1})$初始化； {边界条件}**

**for k:=n downto 1 do**

**for 每一个 $x_k \in X_k$ do**

**for 每一个 $u_k \in U_k(x_k)$ do**

**begin**

**$f_k(x_k)$:=一个极值; {∞或－∞}**

**$x_{k+1}:=T_k(x_k,u_k)$; {状态转移方程}**

**$t:=\varphi(f_{k+1}(x_{k+1}),v_k(x_k,u_k))$; {基本方程(9)式}**

**if t 比 $f_k(x_k)$更优 then $f_k(x_k):=t$; {计算 $f_k(x_k)$的最优值}**

**end;**

**t:=一个极值; {∞或－∞}**

**for 每一个 $x_1 \in X_1$ do**

**if $f_1(x_1)$比 t 更优 then $t:=f_1(x_1)$; {按照 10 式求出最优指标}**

**输出 t;**

**回溯法**

**基本思想：**按照深度优先策略，从根结点出发搜索解空间。算法搜索至解空间的任一结点时总是先判断该结点是否问题的约束条件。如果满足进入该子树，继续按深度优先的策略搜索。否则，不去搜索以该结点为根的子树，而是逐层向其祖先结点回溯。其实回溯法就是对隐式图的深度优先搜索算法

**基本步骤：**

1、 确定问题的解空间：应用回溯法时，首先应明确定义问题的解的空间。问题的解空间应至少包含问题的一个解。

2、 确定结点的扩展规则

3、 搜索解空间：从开始结点出发，以深度优先的方式搜索整个解空间。

【问题】 n 皇后问题

**分支限界法**

**基本思想：** 分支限界法是由“分支”和“限界”两部分组成。“分支”策略体现在对问题空间是按广度优先的策略进行搜索“限界”策略是为了加速搜索速度面采用启发信息剪枝策略。