吴鉴鹰总结的单片机常用算法

算法（Algorithm）：计算机解题的基本思想方法和步骤。

算法的描述：是对要解决一个问题或要完成一项任务所采取的方法和步骤的描述，包括需要什么数据（输入什么数据、输出什么结果）、采用什么结构、使用什么语句以及如何安排这些语句等。通常使用自然语言、结构化流程图、伪代码等来描述算法。

一、计数、求和、求阶乘等简单算法

此类问题都要使用循环，要注意根据问题确定循环变量的初值、终值或结束条件，更要注意用来表示计数、和、阶乘的变量的初值。

例：用随机函数产生 100 个[0，99]范围内的随机整数，统计个位上的数字分别为 1，2，3，4，5，6，7，8，9，0 的数的个数并打印出来。

本题使用数组来处理，用数组 a[100]存放产生的确 100 个随机整数，数组 x[10]来存放个位上的数字分别为 1，2，3，4，5，6，7，8，9，0 的数的个数。即个位是 1 的个数存放在 x[1]中，个位是 2 的个数存放在 x[2]中，……个位是 0 的个数存放在 x[10]。

```
void  main()
{

int  a[101],x[11],i,p;
for(i=0;i<=11;i++)
x[i]=0;
for(i=1;i<=100;i++)
{
```

```
a[i]=rand()  %  100;
printf("%4d",a[i]);

if(i%10==0)printf("\n");
}
for(i=1;i<=100;i++)
{
p="a"[i]%10;
if(p==0)  p="10";
x[p]=x[p]+1;
}
for(i=1;i<=10;i++)
{
p="i";
if(i==10)  p="0";
printf("%d,%d\n",p,x[i]);
}
printf("\n");
}
```

二、求两个整数的最大公约数、最小公倍数

分析：求最大公约数的算法思想：(最小公倍数=两个整数之积/最大公约数)

(1) 对于已知两数 m，n，使得 m>n；

(2) m 除以 n 得余数 r；

(3) 若 r=0，则 n 为求得的最大公约数，算法结束；否则执行(4)；

(4) m←n，n←r，再重复执行(2)。

例如: 求 m="14" ,n=6 的最大公约数. m n r

14 6 2

6 2 0

```
void main()
{ int nm,r,n,m,t;
printf("please input two numbers:\n");
scanf("%d,%d",&m,&n);
nm=n*m;
if (m<n)
{ t="n"; n="m"; m="t"; }
r=m%n;
while (r!=0)
{ m="n"; n="r"; r="m"%n; }
printf("最大公约数:%d\n",n);
printf("最小公倍数:%d\n",nm/n);
}
```

三、判断素数

只能被 1 或本身整除的数称为素数 基本思想：把 m 作为被除数，将 2—IN T（ ）作为除数，如果都除不尽，m 就是素数，否则就不是。（可用以下程序

段实现）

```
void  main()
{  int  m,i,k;
printf("please  input  a  number:\n");
scanf("%d",&m);
k=sqrt(m);
for(i=2;i<k;i++)
if(m%i==0)  break;
if(i>=k)
printf("该数是素数");
else
printf("该数不是素数");
}
```

将其写成一函数,若为素数返回 1，不是则返回 0

```
int  prime(  m%)
{int  i,k;
k=sqrt(m);
for(i=2;i<k;i++)
if(m%i==0)  return  0;
return  1;
}
```

四、验证哥德巴赫猜想

（任意一个大于等于 6 的偶数都可以分解为两个素数之和）

基本思想：n 为大于等于 6 的任一偶数，可分解为 n1 和 n2 两个数，分别检查 n 1 和 n2 是否为素数，如都是，则为一组解。如 n1 不是素数，就不必再检查 n2

是否素数。先从 n1=3 开始，检验 n1 和 n2（n2=N-n1）是否素数。然后使 n1+2 再检验 n1、n2 是否素数，… 直到 n1=n/2 为止。

利用上面的 prime 函数，验证哥德巴赫猜想的程序代码如下：

```
#include "math.h"
int prime(int m)
{ int i,k;
k=sqrt(m);
for(i=2;i<k;i++)
if(m%i==0) break;
if(i>=k)
return 1;
else
return 0;
}

main()
{ int x,i;
printf("please input a even number(>=6):\n");
scanf("%d",&x);
if (x<6||x%2!=0)
printf("data error!\n");
else
for(i=2;i<=x/2;i++)
if (prime(i)&&prime(x-i))
{
printf("%d+%d\n",i,x-i);
```

```
printf("验证成功!");
break;
}
}
```


五、排序问题

1．选择法排序（升序）

基本思想：

1）对有 n 个数的序列（存放在数组 a(n)中），从中选出最小的数，与第 1 个数交换位置；

2）除第 1 个数外，其余 n-1 个数中选最小的数，与第 2 个数交换位置；

3）依次类推，选择了 n-1 次后，这个数列已按升序排列。

程序代码如下：

```
void main()
{ int i,j,imin,s,a[10];
printf("\n input 10 numbers:\n");

for(i=0;i<10;i++)

scanf("%d",&a[i]);
for(i=0;i<9;i++)
{ imin="i";
for(j=i+1;j<10;j++)
if(a[imin]>a[j]) imin="j";
```

```
if(i!=imin)
{s=a[i];  a[i]=a[imin];  a[imin]=s;  }
printf("%d\n",a[i]);
}
}
```

2．冒泡法排序（升序）

基本思想：(将相邻两个数比较，小的调到前头)

1）有 n 个数（存放在数组 a(n)中），第一趟将每相邻两个数比较，小的调到前头，经 n-1 次两两相邻比较后，最大的数已“沉底”，放在最后一个位置，小数上升“浮起”；

2）第二趟对余下的 n-1 个数（最大的数已“沉底”）按上法比较，经 n-2 次两两相邻比较后得次大的数；

3）依次类推，n 个数共进行 n-1 趟比较，在第 j 趟中要进行 n-j 次两两比较。

程序段如下

```
void  main()
{  int  a[10];
int  i,j,t;
printf("input  10  numbers\n");
for(i=0;i<10;i++)
scanf("%d",&a[i]);
printf("\n");

for(j=0;j<=8;j++)
for(i=0;i<9-j;i++)
if(a[i]>a[i+1])
{t=a[i];a[i]=a[i+1];a[i+1]=t;}
printf("the  sorted  numbers:\n");
```

```
for(i=0;i<10;i++)
printf("%d\n",a[i]);
}
```

3．合并法排序（将两个有序数组 A、B 合并成另一个有序的数组 C，升序）

基本思想：

1）先在 A、B 数组中各取第一个元素进行比较，将小的元素放入 C 数组；

2)取小的元素所在数组的下一个元素与另一数组中上次比较后较大的元素比较，重复上述比较过程，直到某个数组被先排完；

3）将另一个数组剩余元素抄入 C 数组，合并排序完成。

程序段如下：

```
void main()
{

int a[10],b[10],c[20],i,ia,ib,ic;
printf("please input the first array:\n");
for(i=0;i<10;i++)
scanf("%d",&a[i]);
for(i=0;i<10;i++)
scanf("%d",&b[i]);
printf("\n");
ia=0;ib=0;ic=0;
while(ia<10&&ib<10)
```

```
{ if(a[ia]<b[ib])

{ c[ic]=a[ia];ia++;}
else
{ c[ic]=b[ib];ib++;}
ic++;
}
while(ia<=9)
{ c[ic]=a[ia];
ia++;ic++;
}
while(ib<=9)
{ c[ic]=b[ib];
b++;ic++;
}
for(i=0;i<20;i++)
printf("%d\n",c[i]);
}
```


六、查找问题

顺序查找法（在一列数中查找某数 x）

基本思想：一列数放在数组 a[1]---a[n]中，待查找的数放在 x 中，把 x 与 a 数组中的元素从头到尾一一进行比较查找。用变量 p 表示 a 数组元素下标，p 初值为 1，使 x 与 a[p]比较，如果 x 不等于 a[p]，则使 p=p+1，不断重复这个过程；一旦 x 等于 a[p]则退出循环；另外，如果 p 大于数组长度，循环也应该停止。（这个过程可由下语句实现）

```
void  main()
```

```
{ int a[10],p,x,i;
printf("please input the array:\n");

for(i=0;i<10;i++)
scanf("%d",&a[i]);
printf("please input the number you want find:\n");
scanf("%d",&x);
printf("\n");
p=0;
while(x!=a[p]&&p<10)
p++;
if(p>=10)
printf("the number is not found!\n");
else
printf("the number is found the no%d!\n",p);
}
```

思考：将上面程序改写一查找函数 Find，若找到则返回下标值，找不到返回-1

②基本思想：一列数放在数组 a[1]---a[n]中，待查找的关键值为 key，把 key 与 a 数组中的元素从头到尾一一进行比较查找，若相同，查找成功，若找不到，则查找失败。(查找子过程如下。index：存放找到元素的下标。)

```
void main()
{ int a[10],index,x,i;
printf("please input the array:\n");
for(i=0;i<10;i++)
scanf("%d",&a[i]);
printf("please input the number you want find:\n");
scanf("%d",&x);
printf("\n");
```

```
index=-1;
for(i=0;i<10;i++)
if(x==a[i])

{ index="i"; break;
}
if(index==-1)
printf("the number is not found!\n");
else
printf("the number is found the no%d!\n",index);
}
```

七、二分法

在一个数组中，知道一个数值，想确定他在数组中的位置下标，如数组：A[5] = {1，2，6，7，9};我知道其中的值为 6，那么他的下标位置就是 3。

```
int Dichotomy(int *ucData, int long, int num)
{
    int iDataLow    = 0 ;
    int iDataHigh = num - 1;
    int iDataMIDDLE;
    while (iDataLow <= iDataHigh)
  {
        iDataMIDDLE = (iDataHigh + iDataLow)/2;
        i f (ucData[iDataMIDDLE] > long)
        {
            iDataHigh = iDataMIDDLE - 1 ;
        }
        else if (ucData[iDataMIDDLE] < long)
  {
    iDataLow = iDataMIDDLE + 1 ;
  }   else{
    return iDataMIDDLE ;
```

```
  }
}
}
```

八、限幅滤波法

对于随机干扰 ，限幅滤波是一种有效的方法；

基本方法：比较相邻 n 和 n - 1 时刻的两个采样值 y(n)和 y(n － 1)，根据经验确定两次采样允许的最大偏差。如果两次采样值的差值超过最大偏差范围 ,认为发生可随机干扰 ,并认为后一次采样值 y(n)为非法值 ,应予删除 ,删除 y(n)后 ，可用 y(n － 1) 代替 y(n)；若未超过所允许的最大偏差范围 ,则认为本次采样值有效。

下面是限幅滤波程序：(A 值可根据实际情况调整，value 为有效值 ,new_value 为当前采样值滤波程序返回有效的实际值)

```
#define A 10
char value;
char filter()
{   char new_value;
    new_value = get_ad();
    if ( ( new_value - value > A ) || ( value - new_value > A ))   return value;
    return new_value;
}
```

九、中位值滤波法

中位值滤波法能有效克服偶然因素引起的波动或采样不稳定引起的误码等脉冲干扰；

对温度 液位等缓慢变化的被测参数用此法能收到良好的滤波效果 ,但是对于流量压力等快速变化的参数一般不宜采用中位值滤波法；

基本方法：对某一被测参数连续采样 n 次(一般 n 取奇数) ,然后再把采样值按大小排列 ,取中间值为本次采样值。

下面是中位值滤波程序：

```
#define N   11
char filter()
{  char value_buf[N]，  count,i,j,temp;
   for ( count=0;count<N;count++)
   {  value_buf[count] = get_ad();    delay();   }
   for (j=0;j<N-1;j++)
   {  for (i=0;i<N-j;i++)
```

```
        {  if ( value_buf[i]>value_buf[i+1] )
             {temp = value_buf[i]; value_buf[i] = value_buf[i+1]; value_buf[i+1] =
temp;  }
        }
   }
   return value_buf[(N-1)/2];
}
```

十.算术平均滤波法

算术平均滤波法适用于对一般的具有随机干扰的信号进行滤波。这种信号的特点

是信号本身在某一数值范围附近上下波动 ,如测量流量、 液位；

基本方法：按输入的 N 个采样数据 ,寻找这样一个 Y ,使得 Y 与各个采样值之间的偏差的平方和最小。

编写算术平均滤波法程序时严格注意:

一.为了加快数据测量的速度 ,可采用先测量数据 存放在存储器中 ,测完 N 点后 ,再对 N 个数据进行平均值计算;

二.选取适当的数据格式 ,也就是说采用定点数还是采用浮点数。其程序

如下所示：

```
#define N 12
char filter()
{int   sum = 0，count;
      for ( count=0;count<N;count++)
      {  sum+=get_ad();    delay();}
return (char)(sum/N);
}
```

十一、递推平均滤波法

基本方法：采用队列作为测量数据存储器， 设队列的长度为 N ,每进行一次测量 ,把测量结果放于队尾 ,而扔掉原来队首的一个数据 ,这样在队列中始终就有 N 个 “最新” 的数据。当计算平均值时 ,只要把队列中的 N 个数据进行算数平均 ,就可得到新的算数平均值。这样每进行一次测量 ,就可得到一个新的算术平均值。

```
#define N 12
```

```
char value_buf[N]，i=0;
char filter()
{ char count; int    sum=0;
  value_buf[i++] = get_ad();
 if ( i == N )     i = 0;
 for ( count=0;count<N;count++)
        sum = value_buf[count];
 return (char)(sum/N);
}
```

十二、一阶滞后滤波法

优点：对周期性干扰具有良好的抑制作用,适用于波动频率较高的场合；

缺点：相位滞后，灵敏度低.滞后程度取决于 a 值大小.不能消除滤波频率高于采样频率的 1/2 的干扰信号。程序如下：

```
#define a 50
char value;
char filter()


{ char    new_value;
   new_value = get_ad();
   return (100-a)*value + a*new_value;
}
```

十三、PID 控制算法

在过程控制中，按偏差的比例（P）、积分（I）和微分（D）进行控制的 PID 控制器（亦称 PID 调节器）是应用最为广泛的一种自动控制器;

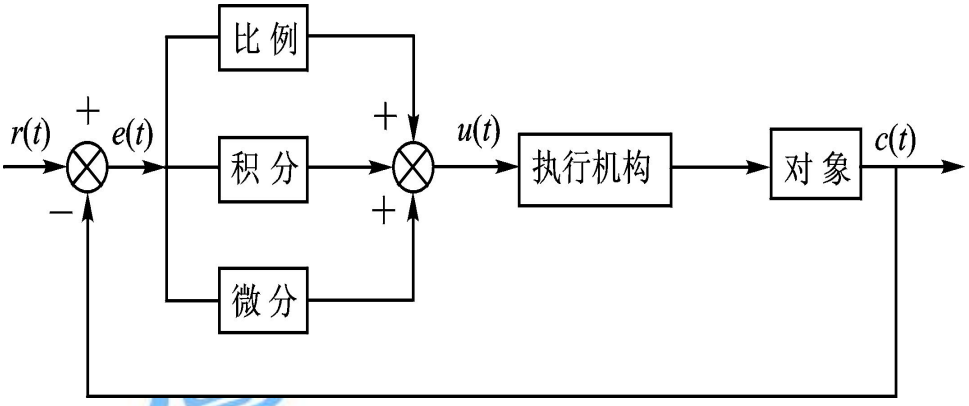
对于过程控制的典型对象——“一阶滞后＋纯滞后”与“二阶滞后＋纯滞后”的控制对象，PID 控制器是一种最优控制;

PID 调节规律是连续系统动态品质校正的一种有效方法，它的参数整定方式简

便，结构改变灵活（PI、PD、…）。

一　模拟 PID 调节器

吴鉴鹰总结的单片机常用算法



模拟 PID 控制系统原理框图

PID 调节器各校正环节的作用：

比例环节：即时成比例地反应控制系统的偏差信号 e(t)，偏差一旦产生，调节器立即产生控制作用以减小偏差；

积分环节：主要用于消除静差，提高系统的无差度。积分时间常数 TI 越大，积分作用越弱，反之则越强；

微分环节：能反应偏差信号的变化趋势(变化速率)，并能在偏差信号的值变得太大之前，在系统中引入一个有效的早期修正信号，从而加快系统的动作速度，减小调节时间。

PID 调节器是一种线性调节器，它将给定值 r(t)与实际输出值 c(t)的偏差的比例(P)、积分(I)、微分(D)通过线性组合构成控制量，对控制对象进行控制。

PID 调节器的微分方程 【其中 $e(t)=r(t)-c(t)$】

$$u(t)=K_P\left[e(t)+\frac{1}{T_I}\int_0^t e(t)dt+T_D\frac{de(t)}{dt}\right]$$

PID 调节器的传递函数

$$D(S)=\frac{U(S)}{E(S)}=K_P\left[1+\frac{1}{T_I S}+T_D S\right]$$

二　数字 PID 控制器

模拟 PID 控制规律的离散化形式

吴鉴鹰总结的单片机常用算法

模拟形式	离散化形式
$e(t)=r(t)-c(t)$	$e(n)=r(n)-c(n)$
$\frac{de(t)}{dT}$	$\frac{e(n)-e(n-1)}{T}$
$\int_0^t e(t)dt$	$\sum_{i=0}^{n} e(i)T = T\sum_{i=0}^{n} e(i)$

数字 PID 控制器的差分方程：

$$u(n)=K_P\left\{e(n)+\frac{T}{T_I}\sum_{i=0}^{n}e(i)+\frac{T_D}{T}[e(n)-e(n-1)]\right\}+u_0$$

$$=u_P(n)+u_I(n)+u_D(n)+u_0$$

式中：$u_P(n)=K_Pe(n)$，为比例项；

$u_I(n)=K_P\frac{T}{T_I}\sum_{i=0}^{n}e(i)$，为积分项；

$u_D(n)=K_P\frac{T_D}{T}[e(n)-e(n-1)]$，为微分项。

程序片段如下：

```
#include <reg52.h>
#include <string.h>
typedef struct PID {
double SetPoint;        // 设定目标 Desired value
double Proportion;      // 比例常数 Proportional Const
double Integral;        // 积分常数 Integral Const
double Derivative;      // 微分常数 Derivative Const
double LastError;       // Error[-1]

double PrevError;       // Error[-2]
double SumError;     // Sums of Errors
} PID;

主程序：
double sensor (void)
{
return 100.0; }
void actuator(double rDelta)
{}
void main(void)
{
PID sPID;
double rOut;
double rIn;
PIDInit ( &sPID );
sPID.Proportion = 0.5；
sPID.Derivative = 0.0;
sPID.SetPoint = 100.0;

for (;;) {
rIn = sensor ();
rOut = PIDCalc ( &sPID,rIn );
actuator ( rOut );
}
}
```

十四、开根号算法

单片机开平方的快速算法

吴鉴鹰总结的单片机常用算法

因为工作的需要，要在单片机上实现开根号的操作。目前开平方的方法大部分是用牛顿迭代法。我在查了一些资料以后找到了一个比牛顿迭代法更加快速的方法。不敢独享，介绍给大家，希望会有些帮助。

1.原理

因为排版的原因，用 pow(X,Y)表示 X 的 Y 次幂，用 B[0]，B[1]，...，B[m-1]表示一个序列，其中[x]为下标。

假设：

B[x],b[x]都是二进制序列,取值 0 或 1。

M = B[m-1]*pow(2,m-1) + B[m-2]*pow(2,m-2) + ... + B[1]*pow(2,1) + B[0]*pow(2,0)

N = b[n-1]*pow(2,n-1) + b[n-2]*pow(2,n-2) + ... + b[1]*pow(2,1) + n[0]*pow(2,0)

pow(N,2) = M

(1) N 的最高位 b[n-1]可以根据 M 的最高位 B[m-1]直接求得。

设 m 已知,因为 pow(2, m-1) <= M <= pow(2, m)，所以 pow(2, (m-1)/2) <= N <= pow(2, m/2)

如果 m 是奇数，设 m=2*k+1,

那么 pow(2,k) <= N < pow(2, 1/2+k) < pow(2, k+1),

n-1=k, n=k+1=(m+1)/2

如果 m 是偶数，设 m=2k,

那么 pow(2,k) > N >= pow(2, k-1/2) > pow(2, k-1),

n-1=k-1,n=k=m/2

所以 b[n-1]完全由 B[m-1]决定。

余数 M[1] = M - b[n-1]*pow(2, 2*n-2)

(2) N 的次高位 b[n-2]可以采用试探法来确定。

因为 b[n-1]=1，假设 b[n-2]=1，则 pow(b[n-1]*pow(2,n-1) + b[n-1]*pow(2,n-2), 2) = b[n-1]*pow(2,2*n-2) + (b[n-1]*pow(2,2*n-2) + b[n-2]*pow(2,2*n-4)),

然后比较余数 M[1]是否大于等于 (pow(2,2)*b[n-1] + b[n-2]) * pow(2,2*n-4)。这种比较只须根据 B[m-1]、B[m-2]、...、B[2*n-4]便可做出判断，其余低位不做比较。

若 M[1] >= (pow(2,2)*b[n-1] + b[n-2]) * pow(2,2*n-4)，则假设有效，b[n-2] = 1；

余数 M[2] = M[1] - pow(pow(2,n-1)*b[n-1] + pow(2,n-2)*b[n-2], 2) = M[1] - (pow(2,2)+1)*pow(2,2*n-4)；

若 M[1] < (pow(2,2)*b[n-1] + b[n-2]) * pow(2,2*n-4)，则假设无效，b[n-2] = 0；余数 M[2] = M[1]。

(3) 同理，可以从高位到低位逐位求出 M 的平方根 N 的各位。

使用这种算法计算 32 位数的平方根时最多只须比较 16 次，而且每次比较时不必把 M 的各位逐一比较，尤其是开始时比较的位数很少，所以消耗的时间远低于牛顿迭代法。

3. 实现代码

这里给出实现 32 位无符号整数开方得到 16 位无符号整数的 C 语言代码。

```
/*****************************************/
/*Function: 开根号处理                    */

/*入口参数：被开方数，长整型              */
/*出口参数：开方结果，整型                */
/*****************************************/
unsigned int sqrt_16(unsigned long M)
{
    unsigned int N, i;
    unsigned long tmp, ttp;     // 结果、循环计数
    if (M == 0)                 // 被开方数，开方结果也为 0
        return 0;

    N = 0;

    tmp = (M >> 30);            // 获取最高位：B[m-1]
    M <<= 2;
    if (tmp > 1)                // 最高位为 1
    {
        N ++;                   // 结果当前位为 1，否则为默认的 0
        tmp -= N;
    }

    for (i=15; i>0; i--)        // 求剩余的 15 位
    {
        N <<= 1;                // 左移一位

        tmp <<= 2;
        tmp += (M >> 30);       // 假设
```

```
            ttp = N;
            ttp = (ttp<<1)+1;

            M <<= 2;
            if (tmp >= ttp)          // 假设成立
            {
                  tmp -= ttp;
                  N ++;
            }

      }

      return N;
}
```