C算法总结

第一章 数学问题

第一节 特殊数

一.因子

概念：在数学中，因子是指能够整除给定数的数

```
输出n的所有因子
int n, i;
scanf("%d", &n);
for (i = 1; i <= n; i++)
    if (n % i == 0)
        printf("%d\t", i);
```

二.真因子

概念：在数学中，真因子是指一个自然数除自身以外的因子

```
输出n的所有真因子
#include <stdio.h>
int main()
{
    int n, i;
    scanf("%d", &n);
    for (i = 1; i < n; i++) //这里注意和因子区分开
        if (n % i == 0)
            printf("%d\t", i);
    return 0;
}
```

对于一个数n，可以证明在大于n/2的部分不会出现n的真因子，因此循环部分也可改为：

```
for (i = 1; i <= n / 2; i++) //注意是<=
    if (n % i == 0)
        printf("%d\t", i);
```

三.完全数（完美数）

概念：完全数指的是一个正整数，其所有真因子（即除了自身以外的因子）的和等于这个数本身

```
输出 [1,1000]内的所有完全数
int i, j, s;
for (i = 1; i <= 1000; i++)
{
    //真因子之和
    s = 0;
    for (j = 1; j < i; j++)
        if (i % j == 0)
            s += j;
    //判断
    if (s == i)
        printf("%d\t", j);
}
```

四.亲密数对

概念：a的真因子和等于b,b的真因子和等于a

```
输出3000以内的所有亲密数
int a, i, b, n;
for (a = 1; a < 3000; a++)
{
    for (b = 0, i = 1; i < a; i++)
        if (a % i == 0)
            b += i;
    for (n = 0, i = 1; i < b; i++)
        if (b % i == 0)
            n += i;
    if (n == a && a < b)//看题目输出，不重复则加上a < b 条件
        printf("%d\t%d\n", a, b);
}
```

五.素数（质数）

概念：素数（质数）是指在大于1的自然数中，除了1和它本身以外，不能被其他自然数整除的数。换句话说，一个大于1的自然数如果只有两个正因数（1和它本身），则这个数就是素数。

合数：在大于1的整数中，除了能被1和它本身整除外，还能被其他整数整除的数。

简而言之，正整数不是质数就是合数。

最基础的素数判断写法,也就是根据定义来写，写法如下：

判断一个数n是否为素数

```
#include <stdio.h>
int main()
{
    int n, i;
    scanf("%d", &n);
    for (i = 2; i < n; i++)
        if (n % i == 0)
            break;
    if (i == n) printf("prime");
    else printf("not prime");
    return 0;
}
```

素数判断也可以有优化方式，具体有两种：

优化1：对半优化

```
#include <stdio.h>
#include <math.h>
int main()
{
    int n, i;
    scanf("%d", &n);
    for (i = 2; i <=n / 2; i++)
        if (n % i == 0)
            break;
    if (i > sqrt(n)) printf("prime");
    else printf("not prime");
    return 0;
}
```

优化2：开根号优化

优化2模版一

```
#include <stdio.h>
#include <math.h>//别忘记数学函数头文件
int main()
{
    int n, i;
    scanf("%d", &n);
    for (i = 2; i <= sqrt(n); i++)
        if (n % i == 0)
            break;
    if (i > sqrt(n)) printf("prime");
    else printf("not prime");
    return 0;
}
```

优化2模版二

```c
#include <stdio.h>
#include <math.h>
int main()
{
    int n, i;
    scanf("%d", &n);
    for (i = 2; i * i <= n; i++)
        if (n % i == 0)
            break;
    if (i * i > n) printf("prime");
    else printf("not prime");
    return 0;
}
```

说明1：优化1和优化2模版一在C语言考试中较为常见，优化2模版二 C++中较为常见，最好都记住

说明2：执行速度排行（最快到最慢）：

优化2 >优化1>无优化

说明3：如果不想在循环外用循环变量和待判断的数相比较来判断是否是素数，也可以用旗帜法，我来演示一下：

```c
#include <stdio.h>
int main()
{
    int n, i, f = 1;
    scanf("%d", &n);
    for (i = 2; i < n; i++)
        if (n % i == 0)
        {
            f = 0;
            break; //判断不是素数了没必要再往下做，直接break掉
        }
        if (f) printf("prime");
    else printf("not prime");
    return 0;
}
```

六.水仙花数

概念：三位正整数，其各位三次方之和等于本身

水仙花数模版一

```c
输出所有的水仙花数
#include <stdio.h>
int main()
{
    int n, m, sum, t;
    for (n = 100; n <= 999; n++)
    {
        sum = 0, m = n;
        while (m)
        {
```

```
            t = m % 10;
            sum += t * t * t;
            m /= 10;
        }
        if (sum == n)
            printf("%d\t", n);
    }
    return 0;
}
```

水仙花数模版二

```
输出所有的水仙花数
#include <stdio.h>
int main() {
    int x, y, z;
    for (x = 1; x <= 9; x++)
        for (y = 0; y <= 9; y++)
            for (z = 0; z <= 9; z++)
                if(x*x*x+y*y*y+z*z*z==x*100+y*10+z)
                    printf("%d\t",x*100+y*10+z);
    return 0;
}
```

说明：模版一在判断完一个数是水仙花数后可以继续对这个数做其他操作，实用性更广，而模版二只适合输出所有的水仙花数。因此，我更推荐模版一。

七.阿姆斯特朗数

概念：简而言之就是未知位数，也未知次方数的水仙花数。

```
输出1~2000的所有阿姆斯特朗数
注意是几次方,这里以三次方为例，具体看题目
#include <stdio.h>
int main()
{
    int n, m, s, t;
    for (n = 1; n <= 2000; n++)
    {
        m = n, s = 0;
        while (m)
        {
            t = m % 10;
            s += t * t * t;
            m /= 10;
        }
        if (s == n)
            printf("%d\t", n);
    }
    return 0;
}
```

八.同构数

概念：一个数出现在它的平方的右端

同构数模版一（学生工作页）

```
输出[1,1000]内的所有同构数
int x, d;
for (x = 1; x <= 1000; x++)
{
    d = x * x;
    if(d%10==x||d%100==x||d%1000==x)
        printf("%d,%d\n", x, d);
}
```

同构数模版二（练习册上）

```
int x, d, m;
for (x = 1; x <= 1000; x++)
{
    d = x * x;
    m = 10;
    if (x > 10) m = 100;
    if (x > 100) m = 1000;
    if (d % m == x)
        printf("%d\n", x);
}
```

九.完全平方数

概念： 一个数是另一个数的平方，那么这个数就是完全平方数。

```
输出1~1000之间的所有完全平方数
#include <stdio.h>
#include <math.h>//数学函数头文件
int main()
{
    int x, d;
    for (x = 1; x <= 1000; x++)
    {
        d = (int)sqrt(x);//这里使用强制类型转换
        if (d * d == x)
            printf("%d %d\n", x, d);
    }
    return 0;
}
```

第二节 分解质因数

概念：分解质因数就是把一个合数分解成若干个质因数的乘积的形式。

例如，28=2×2×7，2和7就是28的质因数。

分解方法：一般先从较小的质数试着分解，一直除到商为质数为止。

分解质因数模版一

```
键盘输入一个整数n，对其进行质因数分解
#include <stdio.h>
int main()
{
    int n, i;
    scanf("%d", &n);
    printf("%d=", n);
    for (i = 2; i <= n; i++)//从最小素数2开始分解
        while (i != n)//可能分解出多个质数i，因此用while循环
            if (n % i == 0)
            {
                printf("%d*", i);
                n /= i;
            }
            else
                break;
    printf("%d", n);
    return 0;
}//下方有解释
```

单看这个程序，肯定有人会有疑问:“外循环是for (i = 2; i <= n; i++)难道不会把n分成像4，6这样的合数吗？”其实不会，而且遇到合数时不会执行内循环对n进行分解，请仔细想一想n是否有机会分解为这些合数？以合数4举例，4是2的倍数，如果n可以整除4，那么n一定可以整除2，由于程序顺序执行，4就先被分解成了两个2相乘的形式；再看6, 6也一定可以分成2和3相乘的形式。这里和大家补充个小知识：素数的n(n>1且n为正整数)倍一定是合数（埃氏筛主要思想，下一节会讲到），这也就可以解释通为什么程序不会将n分解为合数了。

其实还可以对上面的代码进行精简，我先介绍一个等价的模版：

```
while (1)
    if (表达式成立)
    {
        循环体;
    }
    else
        break;
```

```
while (表达式成立)
{
    循环体;
}
```

以上两种写法等价，因此分解质因数的代码可以写成：

分解质因数模版二

```
#include <stdio.h>
int main()
{
    int n, i;
    scanf("%d", &n);
    printf("%d=", n);
```

```
    for (i = 2; i <= n; i++)
        while (n != i && n % i == 0)
        {
            printf("%d*", i);
            n /= i;
        }
        printf("%d", n);
    return 0;
}
```

可能又有人会想：“为什么要把内循环写得那么抽象，感觉一行while(n%i==0)就够了。”程序之所以这样写是因为要处理格式的问题，如果写成一行，那么程序的末尾会多一个“”号，会变成“72=22233”不符合输出规范，所以需要特殊处理，让i!=n时候做内循环，只要一判断到相等，就立马跳出到循环外输出最后一个分解的数。其实也可以把最后一个要输出的数放在内循环中处理，因为多了一个“*”号，所以只需要在循环外退格一下即可，可以参考以下代码：

分解质因数模版三

```
#include <stdio.h>
int main()
{
    int n, i;
    scanf("%d", &n);
    printf("%d=", n);
    for (i = 2; i <= n; i++)
        while (n % i == 0)
        {
            printf("%d*", i);
            n /= i;
        }
        printf("\b ");//  \b 和 空格
    return 0;
}
```

第三节 素数筛法

概念：素数筛法是一种用于筛选出一定范围内所有素数的方法。最常用的素数筛法是埃拉托斯特尼筛法（Sieve of Eratosthenes），简称埃氏筛，它由希腊数学家埃拉托斯特尼提出。

埃氏筛基本思想：埃拉托斯特尼筛法的基本思想是从最小的素数2开始，将其所有的倍数标记为合数，然后找到下一个未被标记的数（即下一个素数3），再将其所有的倍数标记为合数，如此循环，直到遍历完所有小于等于n的整数。未被标记的数即为素数。

由于需要对素数进行标记，习惯来说我们可以定义一个全局标记数组来标记素数

```
#define N 1000
int flag[N + 1] = {1, 1, 0}; //flag[i]为0表示i是素数，为1表示合数
```

N是上边界，也就是输出的所有素数一定小于等于N。因为可以取到N，所以开标记数组flag时需要开N+1空间，使得下标范围在[0,N]的所有数都可以标记到。因为0,1都不是素数，因此可以先对标记数组初始化为int flag[N + 1] = {1, 1, 0};

为了方便大家在头脑中建立数学模型，这里我用n = 12来模拟一下整个过程：

flag下标	初始化	i=2	i=3	i=5	i=7	i=11
0	1	1	1	1	1	1
1	1	1	1	1	1	1
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	1	1	1	1	1
5	0	0	0	0	0	0
6	0	1	1	1	1	1
7	0	0	0	0	0	0
8	0	1	1	1	1	1
9	0	0	1	1	1	1
10	0	1	1	1	1	1
11	0	0	0	0	0	0
12	0	1	1	1	1	1

解释：如果做到被标记为1的数（如flag[4]=1）即合数，是直接跳过它的，不对它的倍数进行标记；只有执行到被标记为0的数（如flag[3]=0）即质数，才应该对它的倍数进行标记。

根据我们模拟的过程来写代码：

素数筛法模版一（技能手册）

```
void is_prime()
{
    int i, j;
    for (i = 2; i <= N; i++)//遍历[0,N]所有数
        if (!flag[i])//如果是素数（0）
            for (j = 2; i * j<= N; j++)//在素数的j倍没有超过
                flag[j * i] = 1;//N范围情况下将其标记为合数（1）
            }
```

素数筛法模版二（信息学奥赛指导用书）

```
void is_prime()
{
    int i, j;
    for (i = 2; i <= N; i++)
        if (!flag[i])
            for (j = 2 * i; j <= N; j += i)//这样内层下标好写点
                flag[j] = 1;
}
```

一般素数筛算法最好写成自定义函数，在主函数开始时直接调用即可

主函数中：

```
输出[0,N]范围内的所有素数
int main()
{
    is_prime(); //直接调用函数即可
    int i, k = 0;
    for (i = 0; i <= N; i++)
        if (!flag[i])
        {
            printf("%d\t", i);
            if ((k + 1) % 6 == 0)
                printf("\n");
            k++;
        }
        return 0;
}
```

其实还有一种更加高效的筛素数算法叫做欧拉筛（线性筛），这种算法可以避免一个合数被多个素数同时筛去的情况，举个例子，埃氏筛中2和3都会把6筛掉，因为6既是2的倍数同时也是3的倍数，而欧拉筛不会出现这种情况，具体怎么实现的可以上网查阅了解，这里只给出代码，不过多赘述。

```
#include <stdio.h>
const int n = 1000;
int vis[n];
int a[n], cnt;
void is_prime()
{
    for (int i = 2; i <= n; i++)
    {
        if (!vis[i])
            a[cnt++] = i;
        for (int j = 0; j < cnt && a[j] * i <= n; j++)
            {       //i代表的是倍数,不可以和上if结合
                vis[a[j] * i] = 1;
                if (i % a[j] == 0)
                    break;
            }
        }
    }
    int main()
{
    is_prime();
    for (int i = 0; i < cnt; i++)
    {
        printf("%d\t", a[i]);
        if ((i + 1) % 5 == 0)
            printf("\n");
    }
    return 0;
}
```

拥有梦想，不如脚踏实地，一步一个脚印向前走

恭喜你，所有有关数学类的C语言算法已经学完了。

第二章 数组变换

第一节 矩阵问题

一.矩阵转置

概念：行变列，列变行；

举两个例子：

方阵				**转换后**		
1	2	3	—>	1	4	7
4	5	6		2	5	8
7	8	9		3	6	9
非方阵				转换后		
1	2	3	—>	1	4	
4	5	6		2	5	
				3	6	

代码实现代码如下：方阵转置

```
将a方阵转置
void zhuanzhi(int a[N][N])//方阵转置只需要对自身进行操作即可
{
    int i, j, t;
    for (i = 0; i < N; i++)
        for (j = 0; j < i; j++)
        {
            t = a[i][j];
            a[i][j] = a[j][i];
            a[j][i] = t;
        }
}
```

非方阵转置

```
将a矩阵转置
void zhuanzhi(int a[N][M], int b[M][N])//非方阵转置需要辅助数组
{
    int i, j, t;
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            b[j][i] = a[i][j];
    //其实 a[i][j] = b[j][i]也行
}
```

说明：第二种非方阵转置同样也适用于方阵

二.矩阵旋转

1.顺时针转90度

字面意思...

举一个例子：

					旋转后数组	
原数组	1	2	3	—>	4	1
	4	5	6		5	2
					6	3
转置后	1	4				
	2	5				
	3	6				

从原数组直接变成旋转后的数组，用笔在纸上模拟简直轻而易举，但是怎样用C语言来模拟这一整个过程呢?

(原数组转置后的样子是参考图，方便理解)

顺时针旋转90度模版一

由上图不难发现旋转后的数组就是将原数组转置后，再将其列逆序。

那么我们先仿照上面总结出的过程来写一段程序：

```
将a矩阵顺时针旋转90°给c数组（通过b数组）
void xuanzhuan(int a[N][M], int b[M][N], int c[M][N])
{
    int i, j, t;
    //第一步：将a数组转置给b数组
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            b[j][i] = a[i][j];
    //第二步：将b数组列逆序
    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            c[i][N - 1 - j] = b[i][j];
}
```

注释：也可以不用c数组，直接对b数组列进行交换从进行逆序，但是为了便于理解旋转的具体过程，这里就用复制的方法。

这里附上用交换来写的程序段：

顺时针旋转90度模板二

```
void xuanzhuan(int a[N][M], int b[M][N])
{
    int i, j, t;
    //第一步：将a数组转置给b数组
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            b[j][i] = a[i][j];
    //第二步：将b数组列改变
    for (i = 0; i < M; i++)
        for (j = 0; j < N / 2; j++)
        {
            t = b[i][j];
            b[i][j] = b[i][N-1-j];
            b[i][N-1-j] = t;
        }
    }
```

“先将原数组转置后，再将其列逆序”自己写编程题时候便于理解可以使用，但是程序改错和填空可不会出得这么繁琐。其实，旋转还有更精简的写法。

我们可以想一想，模版一中的第一步与第二步是否可以结合在一起写？当然是可以的。

可以参照下方图来理解

						旋转后数组b	
	原数组a	1	2	3	—>	4	1
		4	5	6		5	2
						6	3
	a数组转置后	1	4				
		2	5				

						旋转后数组b	
		3	6				

由于对下标的使用较为复杂，这里新定义一种方法“下标法”。本章从这里将一直使用“下标法”来解决数组下标类问题。

“下标法”：定义原数组为a（2行3列），旋转后的数组为b数组（3行2列）；假设a数组中某一个元素下标位(i,j)，如果我知道了a数组(i,j)对应b数组中的什么下标，再将a数组(i,j)对应的元素复制过去，那么问题就解决了。

第一步：将(i,j)转置，得(j,i)。（对应模版一第一步）

第二步：将(j,i)列下标进行逆序，得(j,2-1-i)。（对应模版一第二步）

综上，a数组中(i,j)下标对应b数组(j,2-1-i)下标；现在只需要将a数组(i,j)下标对应元素复制到b数组(j,2-1-i)下标位置上即可b[j][2-1-i]=a[i][j];

若a数组有N行M列，b数组有M行N列，则b[j][N-1-i]=a[i][j];

完整代码如下：

```
void xuanzhuan(int a[N][M], int b[M][N])
{
    int i, j, t;
    //转置逆序两步合一步
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            b[j][N-1-i] = a[i][j];
}
```

顺时针旋转90度模板三

假设a数组某个元素下标为(i,j)代码会写的，那假设b数组某个元素下标为(i,j)呢?

其实一样的，只是模拟的顺序从a数组推到b数组 改变到 从b数组推到a数组。

大家可以借助下方图来理解

					旋转后数组b	
原数组a	1	2	3		4	1
	4	5	6		5	2
					6	3
					b数组转置后	
				4	5	6
				1	2	3

如图，发现只需要先转置再将其行改变即可。

定义原数组为a（2行3列），旋转后的数组为b数组（3行2列）；假设b数组中某个元素下标为(i,j)。

第一步：(i,j)转置为(j,i)。

第二步：(j,i)改变行为(2-1-j,i)。

最后进行复制b[i][j]=a[2-1-j][i];

完整代码如下：

```
void xuanzhuan (int a[N][M], int b[M][N])
{
    int i, j, t;
    //转置逆序两步合一步
    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            b[i][j] = a[N-1-j][i];
}
```

2.逆时针旋转90度

旋转过程如下：

						旋转后数组b	
	原数组a	1	2	3		3	6
		4	5	6		2	5
						1	4
	a数组转置后	1	4				
		2	5				
		3	6				

同理顺时针旋转的过程，这里直接附上以元素组来写的代码：

```
void xuanzhuan (int a[N][M], int b[M][N])
{
    int i, j, t;
    //转置逆序两步合一步
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            b[M-1-j][i] = a[i][j];
}
```

3.旋转180度（不分顺逆时针）

字面意思....

						旋转后数组b	
	原数组a	1	2			10	9
		3	4			8	7

						旋转后数组b	
		5	6			6	5
		7	8			4	3
		9	10			1	2

由于旋转180度是不分顺时针还是逆时针的，因此我们可以通过两次顺时针旋转90度来推导旋转180度的下标。

假设原数组a（2行3列）某个元素下标为(i,j)。

第一次顺时针旋转90度：(i,j)转置得到(j,i)，(j,i)列逆序得到(j,2-1-i)；

第二次顺时针旋转90度：(j,2-1-i)转置得到(2-1-i,j)，(2-1-i,j)列逆序得到

(2-1-i,3-1-j)。

如果不理解这里的2-1-和3-1-是为什么，可以在草稿纸上模拟一下整个过程。

```
void xuanzhuan (int a[N][M], int b[N][M])
{
    int i, j, t;
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            b[N-1-i][M-1-j] = a[i][j];
}
```

第二节 一维/二维复制

一.原数组为二维数组，赋值给一维数组

1.按行转

0,0	0,1		
1,0	1,1		
2,0	2,1		

上图是一个三行两列的矩阵（行，列形式），现在想将按行转成一个一维数组

二维	0,0	0,1	1,0	1,1	2,0	2,1
一维	0	1	2	3	4	5

注：本小节图示都是下标，不是值！

发现，二维数组中某个元素在一维数组中的位置就是这个元素在二维数组中先从左到右再从上到下数前面有多少个元素。

以二维数组中（2,1）为例，在其之前有2行，每行有2列，本行之前有1个元素，因此，（2,1）之前共有2*2+1=5个元素，对应一维数组中下标为5的位置。

总结出规律：一维数组下标=之前元素个数=行标*列数+列标

下面推导一般情况：

假设二维数组为a[M][N],一维数组为b[M*N*]；*二维数组a中某个元素下标为（i，j），那么按照刚才总结出的结论，一维数组中的下标为*iN+j(行标*列数+列标)即b[*iN+j]=a[i][j];

```
void fuzhi(int a[M][N], int b[M * N])
{
    int i, j, t;
    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            b[i * N + j] = a[i][j];
}
```

2.按列转

0,0	**0,0**	**0,1**	**0,1**			
1,0	1,0	1,1	1,1			
2,0	2,0	2,1	2,1			
按列转成一维数组						
二维	0,0	1,0	2,0	0,1	1,1	2,1
一维	0	1	2	3	4	5

按列转与按行转不同，按行转是先从左到右再从上到下数，而按列转是先从上到下再从左到右数，正好相反。

同样以二维数组中（2,1）为例，在其之前有1列，每列有3列，本列之前有2个元素，因此，（2,1）之前共有1*3+2个元素，对应一维数组中下标为5的位置。

总结出规律：一维数组下标=之前元素个数=列标*行数+行标

下面推导一般情况：

假设二维数组为a[M][N],一维数组为b[M*N*]；*二维数组a中某个元素下标为（i，j），那么按照刚才总结出的结论，一维数组中的下标为*jM+i（列标*行数+行标）即b[*jM+i]=a[i][j];

```
void fuzhi(int a[M][N], int b[M * N])
{
    int i, j, t;
    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            b[j * M + i] = a[i][j];
}
```

二.原数组为一维数组，赋值给二维数组

1.按行转

二维数组是3行2列，一维数组3*2=6个元素。

二维	0,0	0,1	1,0	1,1	2,0	2,1
一维	0	1	2	3	4	5

二维数组行标0,0,1,1,2,2变化，一维数组下标0,1,2,3,4,5，发现符合除法运算规则，又因为每两个为一组，所以是除以2，即一维数组下标/2=二维数组行标；同理，二维数组列标0,1,0,1,0,1变化，符合取余运算，又因为两个一组，所以是取余2，即一维数组下标%2=二维数组列标。而2又是二维数组列标，由此可以得出按行转公式：

二维数组行标 = 一维数组下标 / 列数

二维数组列标 = 一维数组小标 % 列数

假设原一维数组为a[M*N],二维数组为b[M][N];a数组中某个下标为i，则对应在b数组中的下标为(i/N,i%N)，即b[i / N][i % N] = a[i];

```
void fuzhi(int a[M * N], int b[M][N])
{
    int i, j, t;
    for (i = 0; i < M * N; i++)
        b[i / N][i % N] = a[i];
}
```

2.按列转

二维数组是3行2列，一维数组3*2=6个元素。

二维	0,0	1,0	2,0	0,1	1,1	2,1
一维	0	1	2	3	4	5

观察图示不难推出结论：

和按行转正好相反

二维数组行标 = 一维数组下标 % 行数

二维数组列标 = 一维数组小标 / 行数

假设原一维数组为a[M*N],二维数组为b[M][N];a数组中某个下标为i，则对应在b数组中的下标为(i%M,i/M)，即b[i % M][i / M] = a[i];

```
void fuzhi(int a[M * N], int b[M][N])
{
    int i, j, t;
    for (i = 0; i < M * N; i++)
        b[i % M][i / M] = a[i];
}
```

三.变种写法

看下方的代码，说出它的功能

```
void fuzhi(int a[M * N], int b[M][N])
{
    int i, j, t;
    for (i = 0; i < M * N; i++)
        a[i] = b[i/N][i%N];
}
```

相信大家看到这串代码肯定有很多疑惑，这串代码的功能其实是将二维数组按行赋值给一维数组。

像这样变种的写法还有，比如

```
void fuzhi(int a[M * N], int b[M][N])
{
    int i, j, t;
    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            b[i][j] = a[i * N + j];
}
```

这串代码的功能是将一维数组按行赋值给二维数组。

由于一维二维复制将赋值号两边正过来写和反过来写只是维度的不同，因此

我们可以总结出一个应对这些复制类问题读程题的方法：

1.关注是按行转还是按列转：可以通过第一第二大点总结的公式来判断；

2.看赋值号左边是一位还是二维：如果是一维那就是二维转一维，如果是二维那就是一维转二维。

如果是自己写编程题或者是程序填空或改错，要注意循环如果是二维那么复制时候二维数组的下标一定是循环的下标（纯一维二维复制）。

知道了做这类问题的方法，我们来实操一下；说出下面代码的功能

```
void fuzhi(int a[M * N], int b[M][N])
{
    int i, j, t;
    for (i = 0; i < M * N; i++)
        a[i] = b[i % M][i / M];
}
```

这串代码的功能是讲二维数组b中的元素按列复制给一维数组a。

简单的纯复制类问题看完了，下面我们来看个难的

```
void fuzhi(int a[6], int b[2][3])
{
    int i, j, t;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 2; j++)
            b[j][3 - 1 - i] = a[i * 2 + j];
}
```

i * 2 + j可以看出来是按行转（不要看形参数组下标，看循环下标范围）。

（j，3-1-i）可以看出来是顺时针旋转90度（看形参数组下标）。

由此可以推出，这串代码的功能是将一维数组a中的元素按行转给二维数组b后再将二维数组b顺时针旋转90度。这里只是将按行转和旋转结合起来写了。

每个人都在奔向自己的目标，不要和别人的脚步纠缠，保持自己的节奏。

恭喜你，所有有关于数组变换类的C语言算法已经学完了。

第三章 去重及排名问题

第一节 单纯去重

一.有序or无序纯去重

1.保留法去重

概念：在保存过得所有元素中查找当前元素是否重复出现，如果出现过就不存，没有出现过就存。

```
int del(int a[], int n)
{
    int i, j, cnt = 0;
    for (i = 0; i < n; i++)
    {
        //保存过得元素当中查找是否有相同
        for (j = 0; j < cnt; j++)
            if (a[j] == a[i])
                break;
        //没有相同则存储当前元素
        if (j == cnt)
            a[cnt++] = a[i];
    }
    return cnt;
}
```

2.删除法去重

概念：从当前元素之后一个元素开始，只要找到重复，就是用移位删除。

```
int del(int a[], int n)
{
    int i, j, k;
    for (i = 0; i < n - 1; i++)
        for (j = i + 1; j < n; j++)
            if (a[j] == a[i])
            {
                for (k = j + 1; k < n; k++)
                    a[k - 1] = a[k];
                n--;//删一个元素就要把数组长度减一
                j--;//防止漏删，最好减内层循环变量，效率更高
            }
    return n;
}
```

以上两种去重方法对于有序和无序都有效。

二.有序

1.相邻比较法去重模版

在有序数组中，将当前元素与保存过得最后一个元素比较，如果相同那就不保存，都则就保存。（有些类似保留法）

相邻比较法模版一

```
int del(int a[], int n)
{
    int i, j, cnt = 1;
    for (i = 1; i < n; i++)
        if (a[i] != a[cnt - 1])
            a[cnt++]  = a[i];
    return cnt;
}
```

相邻比较法模版二

```
int del(int a[], int n)
{
    int i, j, cnt = 0;
    for (i = 1; i < n; i++)
        if (a[i] != a[cnt])
            a[++cnt]  = a[i];
    return cnt + 1;
}
```

第二节 计数去重

一.数组元素有序或无序计数去重

方法：删除法计数去重

概念：从当前元素之后一个元素开始，只要找到重复，就是用移位删除并且将其数量加一。

```
int del_count(int a[], int n, int count[])
{
    int i, j, k;
    for (i = 0; i < n; i++)
    {
        count[i] = 1;//自己本身也算一个
        for (j = i + 1; j < n; j++)
            if (a[j] == a[i])
            {
                count[i]++;
                for (k = j + 1; k < n; k++)
                    a[k - 1] = a[k];
                n--;
                j--;
            }
    }
    return n;
}
```

二．数组元素有序去重

方法：模块化计数去重模版

概念：当前元素和后一个元素比较，如果相同则数量加一，继续往后走；如果不同，则保存刚刚这个元素和这个元素出现的数量。

```
int del_count(int a[], int n, int count[])
{
    int i, j, cnt = 0, c;
    for (i = 0; i < n; i++)
    {
        c = 1;
        while (i + 1 < n && a[i] == a[i + 1])
            {//加上i+1<n防止下标越界和出现最后一个元素单独存在情况
c++;
                i++;
            }
            a[cnt] = a[i];
        count[cnt] = c;
        cnt++;
    }
    return cnt;
}
```

如果内循环没有i+1<n这个条件，1.会使下标越界2.处理不了像1,1,1,2,2,3这种最后一个元素和倒数第二个元素不同的情况。加上这个条件之后，如果倒数第二个元素等于最后一个元素,那么程序执行到倒数第二个时候计数，然后循环到最后一个元素时候内循环终止并进行统计最后一个元素出现的数量；如果倒数第二个元素不等于最后一个元素，那么程序执行到倒数第二个元素时候不会计数，直接统计倒数第二个元素出现的数量，然后程序执行到最后一个元素时不做内循环，直接统计最后一个元素

出现了一次。

第三节 排名问题

概念：如果是升序排名，就找出小于这个元素的有多少个；如果是降序排名，那就找出大于这个元素的有多少个。类似excel中的rank函数。

如果相同元素排名相同，程序如下

```
void rank(int a[], int n, int c[])
{
    int i, j, cnt;
    for (i = 0; i < n; i++)
    {
        cnt = 1;
        for (j = 0; j < n; j++)
            if (a[j] < a[i])
                cnt++;
        c[i] = cnt;
    }
}
```

如果题目明确了相同元素后一个比前一个排名靠后（排名各不相同），那么程序如下

```
void rank(int a[], int n, int c[])
{
    int i, j, cnt;
    for (i = 0; i < n; i++)
    {
        cnt = 1;
        for (j = 0; j < n; j++)
            if (a[j] < a[i] || (j < i && a[j] == a[i]))
                cnt++;
        c[i] = cnt;
    }
}
```

注：这种方式会在后面章节讲到的按排名排序中被运用到。

相信自己的力量，你拥有足够的勇气和智慧战胜人生的挑战。

恭喜你，所有有关于去重及排名类的C语言算法已经学完了

第四章 二分查找

第一节 无重复元素二分查找

概念：在有序数组中查找一个指定元素出现的位置，无重复元素。

对于无重复元素的数组进行二分查找，只需要找到其位置直接返回即可。

注意：二分查找算法有一个非常重要也是自己写编程题时候非常容易疏忽的前提：数组元素有序。

无重复元素二分查找模板（升序）

```
int find(int a[], int n, int x)
{
    int l = 0, r = n - 1, mid;
    while (l <= r)
    {
        mid = (l + r) / 2; //注释
        if (a[mid] == x)
            return mid;
        else if (a[mid] < x)
            l = mid + 1;
        else
            r = mid - 1;
    }
    return -1;
}
```

注释：补充一个小知识，部分mid=(l+r)/2可以替换成mid = l + (r - l) / 2，两者可以展开证明相等，而第二种比第一种更好的地方在于可以防止两数相加发生溢出。

第二节 有重复元素二分查找

概念：在有序数组中查找一个指定元素出现的位置，有重复元素。

有重复元素二分查找顾名思义就是可能会有多个待查找元素。那么到底要找哪一个的位置就需要自己根据题目描述自己做出判断。

一．查找第一个大于等于待查找元素的位置

下标	**0**	**1**	**2**	**3**	**4**	**5**
值	1	1	2	2	4	5

如果查找2，由于2出现了两次，用第一节讲的查找无重复元素的位置的方法就行不通了。如果查找第一个大于等于2的位置，就会得到下标2。如果查找3，没有找到3，就会找比3大的第一个元素位置，即下标4。

有重复元素二分查找第一个大于等于待查找元素位置模板（升序）

```
int find(int a[], int n, int x)
{
    int l = 0, r = n - 1, mid;
    while (l <= r)
    {
        mid = l + (r - l) / 2;
        if (a[mid] >= x)
            r = mid - 1;
        else
            l = mid + 1;
    }
    return l;
}
```

如果想吃透这个算法的运行过程，需要用纸张模拟一下整个过程，这里就不展示了。

算法分析：如果找到待查找元素，那么就会返回第一个大于等于待查找元素的下标；

如果没有找到待查找元素，就会返回比它大的第一个元素的下标。

二．查找插入点

插入	**6**					
值	1	3	6	7	9	
结果	1	3	6	6	7	9
插入	4					
值	1	3	6	7	9	
结果	1	3	4	6	7	9

插入	**6**					
插入	0					
值	1	3	6	7	9	
结果	0	1	3	6	7	9
插入	10					
值	1	3	6	7	9	
结果	1	3	6	7	9	10

观察可以发现，第一个大于待查找元素的位置就是插入点。有了第一点找第一个大于等于待查找元素的位置模板作铺垫，这里查找第一个大于待查找元素位置模板就可以直接给出了。

有重复元素二分查找第一个大于待查找元素位置模板（升序）

```
int find(int a[], int n, int x)
{
    int l = 0, r = n - 1, mid;
    while (l <= r)
    {
        mid = l + (r - l) / 2;
        if (a[mid] > x)//只有这里和第一点不同，变成了大于
            r = mid - 1;
        else
            l = mid + 1;
    }
    return l;//左边界l就是插入点
}
```

插入点(返回的值l)找到了，相信大家元素移位插入的算法已经背的滚瓜烂熟了。

二分查找插入点模板

```
void find(int a[], int n, int x)//x是待插入元素，n是原数组元素个数
{
    int l = 0, r = n - 1, mid, i;
    while (l <= r)
    {
        mid = l + (r - l) / 2;
        if (a[mid] > x)
            r = mid - 1;
        else
            l = mid + 1;
    }
    for (i = n - 1; i >= l; i--)
        a[i + 1] = a[i];
    a[i + 1] = x;
}
```

拓展：使用二分查找法来写插入法排序。

二分查找插入点的插入法排序模板

```
void sort(int a[], int n)
{
    int i, j, l, r, mid, x;
    for (i = 1; i < n; i++)
    {
        l = 0, r = i - 1, x = a[i];
        while (l <= r)
        {
            mid = l + (r - l) / 2;
            if (a[mid] > x)
                r = mid - 1;
            else
                l = mid + 1;
        }
        //此时左边界l就是插入点
        for (j = i - 1; j >= l; j--)
            a[j + 1] = a[j];
        a[j + 1] = x;
    }
}
```

当然，二分的魅力远不止此。请思考：实数是否可以二分呢?

提前告诉你答案，其实是可以的。由于实数二分精度问题，通常不会选择将l和r进行比较，一般使用一个极小值eps作为比较标准，如果答案小于这个极小值，说明满足条件。如果想要了解的话可以上洛谷学习一下，这里就不多讲了。

第五章 进阶排序

考虑到C语言的局限性，以及算法的深度，本章节均使用C++语言编写，如有理解上的问题，可以询问AI(C/C++最大的改变只有输入输出)，同时会涉及到一些有关时间复杂度和空间复杂度内容，这些内容如果不打竞赛的话不需要了解，到大学之后老师会教的（卷王当我没说）。

第一节 shell希尔排序（快速插入排序）

基本思想：由于插入排序在元素本来就有序情况下速度非常快，所以思考能否先将序列先粗糙排序然后在进行插入排序。假设序列长度为n，以升序为例。

1：每次选择一个间距step，第一次是step = n/2，以后每排完一轮将step /= 2。

2：枚举同一间距的不同起点k，例如step=2,k=0时1,3,5和step=2,k=1时2,4,6间距相同但起点不同。

3：插入排序。

时间复杂度：$O(n^{1.3}) \sim O(n^{1.5})$

空间复杂度：$O(1)$

```
#include <iostream>
using namespace std;
const int N = 1e5 + 10; //限制数量不找过1e5+10个
int a[N], n;
int main()
{
```

```
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int step = n / 2; step >= 1; step /= 2)//控制间距
    {
        for (int k = 0; k < step; k++)//遍历不同起点
        {
            for (int i = step + k; i <= n; i += step)//对每个分组进行插入排序
            {
                int x = a[i], j = i - step;
                while (j >= step && x < a[j])
                {
                    a[j + step] = a[j];
                    j -= step;
                }
                a[j + step] = x;
            }
        }
    }
    for (int i = 1; i <= n; i++)
        cout << a[i] << ' ';
    return 0;
}
```

第二节 快速排序

基本思想：用到了分治、递归、双指针的思想，以从小到大排序为例。

1：找到一个基准值x，通常是中间数、第一个数、最后一个数或是随机位置的数。

2：将比x小的数放在x左边，比x大的数放在x的右边。（双指针算法）

3：以x为中心拆分成左右两个序列，对于左序列和右序列，分别不断重复1、2两个步骤，直到序列中的数不可再分位置。（分治、递归算法）

时间复杂度：$O(n \log n)$

空间复杂度：$O(n \log n) \sim O(n)$

```
#include <iostream>
using namespace std;
const int N = 1e5 + 10; //限制数量不找过1e5+10个
int a[N], n;
void quick_sort(int *a, int l, int r)
{
    if (l >= r) //如果序列只有一个数就不需要再排序了
        return;
    int i = l, j = r, mid = a[(l + r) / 2];//找中间数作为基准数-->1
    //双指针-->2
    while (i <= j)//小于等于是为了让两个递归区间正好相邻不重复
    {
        while (a[i] < mid) i++;//对于[2, 2, 2, 2, 2]
        while (a[j] > mid) j--;//如果加上等号会导致ij指针越界
        if (i <= j)
        {
            swap(a[i], a[j]);
```

```
            i++, j--; //不增加会导致死循环，因为ij指针会一直停在这
        }
    }
    //递归两个区间-->3
    if (l < j) quick_sort(a, l, j); //判断条件可以不要，因为函数一开始有终止条件
    if (i < r) quick_sort(a, i, r);
}
int main()
{
    ios::sync_with_stdio(0);//关闭流同步，使得输入输出更快
    cin.tie(0);
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    quick_sort(a, 1, n);//调用快速排序函数
    for (int i = 1; i <= n; i++)
        cout << a[i] << ' ';
    return 0;
}
```

第三节 归并排序

基本思想：类似快速排序，用到了分治、递归、双指针的思想，以从小到大排序为例：

1：划分区间，将一个区间化为两部分， 直到序列长度为1。（递归， 分治）

2：每两个子序列进行归并，使其成为有序序列。不断这个操作，使得整个序列有序。（双指针）

时间复杂度：$O(n \log n)$

空间复杂度：$O(n)$

```
#include <iostream>
using namespace std;
const int N = 1e5 + 10; //限制数量不找过1e5+10个
int n;
int a[N], b[N];
void merge_sort(int *a, int l, int r)
{
    if (l >= r) //如果序列只有一个数就不需要再排序了
        return;
    //划分区间-->1
    int mid = l + (r - l) / 2;
    merge_sort(a, l, mid);
    merge_sort(a, mid + 1, r);
    int i = l, j = mid + 1, k = 0;
    //归并部分-->2
    while (i <= mid && j <= r)
        if (a[i] < a[j])
            b[++k] = a[i++];
        else
            b[++k] = a[j++];
    while (i <= mid)
        b[++k] = a[i++];
    while (j <= r)
        b[++k] = a[j++];
```

```
    for (int i = 1; i <= k; i++)
        a[l + i - 1] = b[i];
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    merge_sort(a, 1, n);
    for (int i = 1; i <= n; i++)
        cout << a[i] << ' ';
    return 0;
}
```

作者：

- 南京浦口中等专业学校 综高22计算机 周泓霖