

**ACM算法模板总结**

**班 级 : 计算机091**

**学 号 : 10921018**

**姓 名 : 鲍 杭**

|  |  |  |
| --- | --- | --- |
| 文件状态：  [√] 草稿  [ ] 正式发布  [ ] 正在修改 | 文件标识： | ACM算法模板总结 |
| 当前版本： | 1.0 |
| 作 者： | 鲍杭编 |
| 完成日期： | 2011-4-3 |

目录

[数论 1](#_Toc291168862)

[高精度算法[1] 1](#_Toc291168863)

[高精度加法 2](#_Toc291168864)

[高精度减法 2](#_Toc291168865)

[高精度乘法 4](#_Toc291168866)

[整型常量阶乘 6](#_Toc291168867)

[整型常量阶乘和 6](#_Toc291168868)

[高精度的乘方 7](#_Toc291168869)

[高精度除法 7](#_Toc291168870)

[带符号的数据运算 9](#_Toc291168871)

[广度优先搜索大数应用 11](#_Toc291168872)

[基于比较的排序算法 12](#_Toc291168873)

[插入排序 12](#_Toc291168874)

[选择排序 13](#_Toc291168875)

[冒泡排序 13](#_Toc291168876)

[堆排序 13](#_Toc291168877)

[归并排序 14](#_Toc291168878)

[快速排序 15](#_Toc291168879)

[希尔排序 16](#_Toc291168880)

[同余运算 16](#_Toc291168881)

[模线性方程问题 16](#_Toc291168882)

[计算a^n mod k 19](#_Toc291168883)

[阶乘最后非0位 20](#_Toc291168884)

[素数篇 20](#_Toc291168885)

[线性筛法 21](#_Toc291168886)

[费马小定理 22](#_Toc291168887)

[MR素性测试 22](#_Toc291168888)

[欧拉函数问题 22](#_Toc291168889)

[巴赫猜想 23](#_Toc291168890)

[最大公约数 24](#_Toc291168891)

[最小公倍数 25](#_Toc291168892)

[不定方程求解 25](#_Toc291168893)

[概率因子分解Pollard 算法 25](#_Toc291168894)

[数值计算[2] 26](#_Toc291168895)

[定积分计算(Romberg) 26](#_Toc291168896)

[多项式求根(牛顿法) 27](#_Toc291168897)

[周期性方程(追赶法) 29](#_Toc291168898)

[组合论[3] 30](#_Toc291168899)

[组合公式 30](#_Toc291168900)

[全组合排列 30](#_Toc291168901)

[非重复组合排列 32](#_Toc291168902)

[类循环组合排列 34](#_Toc291168903)

[普通选择性组合排列 35](#_Toc291168904)

[生成全子集组合排列 36](#_Toc291168905)

[非重复生成全子集组合排列 37](#_Toc291168906)

[一般组合 39](#_Toc291168907)

[排列组合生成 39](#_Toc291168908)

[生成gray码 41](#_Toc291168909)

[置换(polya) 42](#_Toc291168910)

[字典序全排列 42](#_Toc291168911)

[字典序组合 43](#_Toc291168912)

[计算几何 43](#_Toc291168913)

[对于任意多边形求面积 43](#_Toc291168914)

[判断两线是否相交 44](#_Toc291168915)

[二分查找模板 44](#_Toc291168916)

[二分图匹配模板 45](#_Toc291168917)

[动态规划 46](#_Toc291168918)

[求局部最大和模板 46](#_Toc291168919)

[字符串 46](#_Toc291168920)

[字符计数应用 46](#_Toc291168921)

[最大子序列算法 47](#_Toc291168922)

[求最长公供子串（DP算法） 48](#_Toc291168923)

[数据结构 49](#_Toc291168924)

[图论 49](#_Toc291168925)

[入度和出度表的转化 49](#_Toc291168926)

[生成树 52](#_Toc291168927)

[最小生成树 56](#_Toc291168928)

[关键路径 59](#_Toc291168929)

[迷宫算法的队列实现 64](#_Toc291168930)

[最短路径 67](#_Toc291168931)

[栈的相关操作 75](#_Toc291168932)

[一元多项式计算 78](#_Toc291168933)

[单链表的简单操作 84](#_Toc291168934)

[快速排序 89](#_Toc291168935)

[哈夫曼编码 90](#_Toc291168936)

[队列的简单操作 94](#_Toc291168937)

[KMP模式匹配算法 98](#_Toc291168938)

[常用的数学公式[4] 100](#_Toc291168939)

[中国剩余定理 100](#_Toc291168940)

[基本几何公式 100](#_Toc291168941)

[欧拉公式 101](#_Toc291168942)

[划分问题： 102](#_Toc291168943)

[Stirling公式 102](#_Toc291168944)

[皮克定理 103](#_Toc291168945)

[卡特兰数 103](#_Toc291168946)

[原理: 103](#_Toc291168947)

[错排公式 103](#_Toc291168948)

[等比数列 104](#_Toc291168949)

[等差数列 104](#_Toc291168950)

[二次函数 104](#_Toc291168951)

[二次方程 105](#_Toc291168952)

[约瑟夫环 105](#_Toc291168953)

[多边形面积 105](#_Toc291168954)

[均值不等式的简介 105](#_Toc291168955)

[均值不等式的变形 106](#_Toc291168956)

[Lucas 定理 107](#_Toc291168957)

[斐波那契数列 107](#_Toc291168958)

[欧拉函数 108](#_Toc291168959)

[蚂蚁爬绳 109](#_Toc291168960)

[(a/b)%m 109](#_Toc291168961)

[泰勒公式 109](#_Toc291168962)

[乘法与因式分解公式 109](#_Toc291168963)

[三角不等式 110](#_Toc291168964)

[某些数列的前n项和 110](#_Toc291168965)

[二项式展开公式 110](#_Toc291168966)

[三角函数公式 110](#_Toc291168967)

[常用系统函数 112](#_Toc291168968)

[将字符串转化为数字 112](#_Toc291168969)

[系统函数排序 112](#_Toc291168970)

# 数论

## 高精度算法[1]

#include<stdio.h>

#include<string.h>

char c[2000];//全局变量，存储大数运算的结果

char arr[1000];//高精度除以高精度的余数

long z=0;//高精度除以低精度的余数

int Judge(char ch[])

{//判断字符串ch是否全为0，若全为0，返回0，否则返回1

int i,k;

k=strlen(ch);

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

if(ch[i]!='0')

return 0;

return 1;

}

int Compare(char a[],char b[])

{//比较字符串的大小，方法不同于strcmp函数，类似于整型常量的比较

int lena,lenb,i;

lena=strlen(a);

lenb=strlen(b);

if(lena<lenb)

return -1;

else if(lena>lenb)

return 1;

else

{

if(strcmp(a,b)==0)

return 0;

else

{

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

{

if(a[i]>b[i])

return 1;

if(a[i]<b[i])

return -1;

}

return 0;

}

}

}

### 高精度加法

/\*算法：先确定a和b中的最大位数k，然后依照由低至高位的顺序进行加法运算。注意进位，若高位有进位，则c的长度为k+1。\*/

void BigNumberAdd(char a1[],char b1[])

{

int i,j,k,lena,lenb;

int a[1000]={0},b[1000]={0},d[1000]={0};

lena=strlen(a1);

lenb=strlen(b1);

//将加数与被加数化为整型数组，并且该数组的其他位为

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

a[i]=a1[lena-i-1]-'0';

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

b[i]=b1[lenb-1-i]-'0';

//当数组除了加数和被加数以外的整型数组元素均为0时，无需考虑lena和lenb的大小

k=lena>lenb?lena:lenb;

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

{

d[i]=a[i]+b[i]+d[i];

d[i+1]=d[i+1]+d[i]/10;

d[i]=d[i]%10;

}

//若高位进

while(d[k])

k++;

while(!d[k-1])

k--;//001+0003=4

//将整型数组逆着转变并赋值给c字符型数组

for(j=0;j<k;j++)

c[j]=d[k-j-1]+'0';

if(Judge(c))//若全为0，则只输出一个

strcpy(c,"0");

}

### 高精度减法

/\*算法：依照由低位至高位的顺序进行减法运算。在每次位运算中，若出现不够减的情况，则向高位借位。在进行了la的减法后，若最高位为0，则a的长度减。若A、B大小未知，则需先判断大小。\*/

void BigNumberSub(char a1[],char b1[])

{//a1为被减数，b1为减数

int lena,lenb,i,j,k,flag;

int a[1000]={0},b[1000]={0},d[1000]={0};

lena=strlen(a1);

lenb=strlen(b1);

if(Compare(a1,b1)>=0)

{//若被减数大于等于减数

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

a[i]=a1[lena-1-i]-'0';

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

b[i]=b1[lenb-1-i]-'0';

flag=0;//结果正的标志

}

else

{//若被减数小于减数

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

a[i]=b1[lenb-1-i]-'0';

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

b[i]=a1[lena-1-i]-'0';

flag=1;//结果负的标志

}

k=lena>lenb?lena:lenb;

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

{//大数减小数

if(a[i]<b[i])

{//若被减数不够减，向高位借一位

a[i+1]--;

d[i]=a[i]-b[i]+10;

}

else

d[i]=a[i]-b[i];

}

//若较高位已为0，并且不止1位时

while(!d[i-1])

{

k--;

i--;

}

//根据flag，输出有无"-"

if(!flag)

{

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

{//将结果转化为字符逆着赋给数组c

if(!i&&!d[k-i-1])//若差的第一个字母为0，则马上跳过

continue;

c[i]=d[k-i-1]+'0';

}

}

else

{

c[0]='-';

for(i=1;i<=k;i++)

{//将结果转化为字符逆着赋给数组c

if(i==1&&!d[k-i])//若差的第一个字母为0，则马上跳过

continue;

c[i]=d[k-i]+'0';//注意d的下标，不是k-i-1

}

}

if(Judge(c))//若差全为0，则只输出一个

strcpy(c,"0");

}

### 高精度乘法

**高精度乘以低精度**

/\*算法：将多位数存入数组，低位在前、高位在后，然后用一位数去乘数组的各位，考虑进位，最后按正常顺序输出\*/

void BigNumMultiSmall(char a1[],int b1)

{

int i,j,t;

int a[2000]={0};

//将字符串转化为整型数组，并逆置

t=strlen(a1);

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

a[i]=a1[t-1-i]-'0';

//整型数组的每个元素乘以b1，然后对其进行求余，整除，使其只有一位数

a[0]=a[0]\*b1;

for(i=1;i<t;i++)

{

a[i]\*=b1;

a[i]+=a[i-1]/10;

a[i-1]=a[i-1]%10;

}

while(a[i-1]>9)

{//若最后一个元素大于9

a[i]=a[i-1]/10;

a[i-1]=a[i-1]%10;

i++;

}

//将得到的整型数组逆置赋给字符串

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

c[j]=a[i-j-1]+'0';

if(Judge(c))//若积全为0，则只输出一个0

strcpy(c,"0");

}

**高精度乘以高精度**

void BigNumMultiBig(char a1[],char b1[])

{

int i,j,k,lena,lenb;

int a[1000]={0},b[1000]={0},d[2000]={0};

//将字符串转化为整型数组，并逆置

lena=strlen(a1);

lenb=strlen(b1);

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

a[i]=a1[lena-i-1]-'0';

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

b[i]=b1[lenb-i-1]-'0';

//计算乘数从低位到高位以此乘以被乘数的低位到高位

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

for(j=0;j<lenb;j++)

{

d[i+j]=d[i+j]+a[i]\*b[j];

d[i+j+1]+=d[i+j]/10;

d[i+j]=d[i+j]%10;

}

//根据高位是否为判断整型数组的位数

k=lena+lenb;

while(!d[k-1])

k--;

//积转化为字符型

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

c[i]=d[k-1-i]+'0';

if(Judge(c))//若积全为0，则只输出一个1

strcpy(c,"0");

}

### 整型常量阶乘

void BigNumFact(int x)

{

int i,k,m=0,a[1000]={0};

a[0]=1;

for(;x;x--)

{//m为在求阶乘过程中a的元素个数

for(k=i=0;i<=m;i++)

{

k=k+a[i]\*x;//数组各个元素均乘以x(x递减)，以完成阶乘的运算

a[i]=k%10;

k/=10;

}

while(k)

{

a[++m]=k%10;

k/=10;

}

}

//阶乘的结果转化为字符型

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

c[i]=a[m-i]+'0';

if(Judge(c))//若结果全为，则只输出一个

strcpy(c,"0");

}

### 整型常量阶乘和

void BigNumFactAdd(int t) //可以改进，减少算法时间复杂度

{

int i;

char sum[2000],d[2000];

//对字符串进行初始化

memset(d,0,sizeof(d));

memset(sum,0,sizeof(sum));

//分别求出相应i的阶乘然后相加

for(i=t;i>0;i--)

{

BigNumFact(i); //调用大数乘法。可以减少时间复杂度

strcpy(d,c);

memset(c,0,sizeof(c));

BigNumberAdd(d,sum);

strcpy(sum,c);

memset(c,0,sizeof(c));

}

strcpy(c,sum);//将结果赋值给全局变量，进行输出

}

### 高精度的乘方

void BigNumInvol(char a1[],int b1) //幂为整型常量

{

int i;

char temp[1000];

strcpy(temp,a1);//注意乘方是自己乘自己，而不是结果乘结果

for(i=2;i<b1;i++)

{

BigNumMultiBig(a1,temp);

strcpy(temp,c);

memset(c,0,sizeof(c));//将c清空，防止出现错误

}

//进行最后一次乘法

BigNumMultiBig(a1,temp);

if(Judge(c))//若结果全为0，则只输出一个0

strcpy(c,"0");

}

### 高精度除法

**高精度除以低精度，只产生余数**

int BigNumDividSmall(char a1[],int b1)

{

if(!b1)

return 0;

int i,j,k,flag=0,a[1000]={0};

char b[2000];

memset(b,0,sizeof(b));

k=strlen(a1);

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

a[i]=a1[i]-'0';

z=0;

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

{

z=a[i]+z\*10;

b[i]=z/b1+'0';

z=z%b1;

}

i=j=0;

while(b[i++]=='0');

for(i=i-1;i<k;i++)

c[j++]=b[i];

return 1;

}

**高精度除以高精度，只产生余数**

void BigNumDividBig(char a1[],char b1[])

{

char a[1000],b[1000],time[1000];

int lena1,lentime,i,j,k,flag=0;

memset(arr,0,sizeof(arr));

//若被除数小于除数，则商为，余数为被除数

if(Compare(a1,b1)<0)

strcpy(arr,a1);

//若两数相等，则商为，余数为

else if(!Compare(a1,b1))

c[0]='1';

//若被除数大于除数

else

{

j=lentime=0;

lena1=strlen(a1);

memset(b,0,sizeof(b));

memset(time,0,sizeof(time));

for(i=0;i<lena1;i++)

{//计算得到被除数的前几位,得到整型数组形式的商

//time的一个元素表示一次相除的商

b[j++]=a1[i];

flag=0;

while(Compare(b,b1)>=0)

{

BigNumberSub(b,b1);

strcpy(b,c);

memset(c,0,sizeof(c));

time[lentime]++;

flag=1;//控制time的元素的位置

}

if(flag)//将商转换为字符

time[lentime]+='0';

else//当被除数前几位小于除数，商补

time[lentime]='0';

if(!strcmp(b,"0"))//若b为‘’

j=0;

else//继续在b的后面加值

j=strlen(b);

lentime++;

}

k=0;

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

if(time[i]!='0')

break;//找到time数组中第一个不为的位置

for(j=i;j<lentime;j++)

c[k++]=time[j];

strcpy(arr,b);

}

if(Judge(c))

strcpy(c,"0");

if(Judge(arr))

strcpy(arr,"0");

}

### 带符号的数据运算

2  
$1,234,567.89  
$9,876,543.21

#include<string.h>

#include<stdio.h>

int main()

{

char s[10001][30],t[30];

long a,sum;

int i,j,k,m,n;

while(scanf("%d",&n)&&n)

{

sum=0;

getchar();

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

{

gets(s[i]);

m=strlen(s[i]);

a=0;

for(j=1;j<m;j++)

{

if(s[i][j]!=','&&s[i][j]!='.')

a=a\*10+(s[i][j]-'0');

}

sum+=a;

}

if(sum<100)

{

t[0]=(sum%10-0)+'0';

t[1]=((sum%100-sum%10)/10+'0');

t[2]='.';

t[3]='0';

j=4;

}

else

{

t[0]=(sum%10-0)+'0';

t[1]=((sum%100-sum%10)/10+'0');

sum/=100;

t[2]='.';

k=0;

j=3;

while(sum)

{

if(k!=3)

{

t[j]=sum%10+'0';

sum/=10;

j++;

k++;

}

else

{

t[j]=',';

k=0;

j++;

}

}

}

printf("$");

for(i=j-1;i>=0;i--)

printf("%c",t[i]);

printf("\n");

}

}

## 广度优先搜索大数应用

/\*\*\*\*\*求一个数m它的各个数位只由0或1构成，并且能被输入的整数m整除\*\*\*\*\*\*\*/

#include<stdio.h>

struct node

{

int a,b; ///a存0、1,b 存余数

int pre; //前一点

}queue[200];

void output(int k)

{

if(queue[k].pre>=0)

{

output(queue[k].pre);

}

printf("%d",queue[k].a);

}

int main()

{

int n,r,i,j,L;

queue[0].a=1;

queue[0].b=1;

queue[0].pre=-1;

while(scanf("%d",&n)!=EOF) //除数

{

if(n==0)

return 0;

char used[200]={0};

used[1]=1;

L=1; //r

if(n>1)

{

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

{

for(j=0;j<2;j++)//确定只有0或1,广度优先搜索思维

{

r=(queue[i].b\*10+j)%n;

if(!used[r])

{

queue[L].a=j;

queue[L].b=r; //用余数继续乘10+0或+1循环乘、取余直到整除

queue[L].pre=i;

used[r]=1;

L++; //用来控制输出0 1序列的,构造是从右往左,输出时从左往右(L-1开始)

}

if(r==0)

break;

}

if(r==0)

break;

}

}

output(L-1);

printf("\n");

}

return 0;

}

## 基于比较的排序算法

#include <iostream>

using namespace std;

#define SWAP(i,j) {int t=(i);(i)=(j);(j)=t;}

### 插入排序

void InsertSort(int\*a,int len) //算法主体函数

{

for (int i=1;i<len;i++)

{

int j=i,x=a[i];

while (j && a[j-1]>x)

a[j]=a[j-1],j--;

a[j]=x;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 选择排序

void SelectSort(int\*a,int len) //算法主体函数

{

for (int i=1,j,k;i<len;i++)

{

for (j=i,k=i-1;j<len;j++)

if (a[j]<a[k])

k=j;

if (k>=i)

SWAP(a[i-1],a[k]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 冒泡排序

void BubbletSort(int\*a,int len) //算法主体函数

{

for (bool bSwap=true;bSwap;len--)

{

bSwap=false;

for (int j=1;j<len;j++)

if (a[j-1]>a[j])

{

SWAP(a[j-1],a[j]);

bSwap=true;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 堆排序

//堆调整

void HeapAdjust(int \*a,int root,int len)

{

int child,x=a[root];

while (child=root<<1|1,child<len)

{

if (child<len-1 && a[child]<a[child+1])

child++;

if (x<a[child])

{

a[root]=a[child];

root=child;

}

else

break;

}

a[root]=x;

}

//堆排序

void HeapSort(int\*a,int len) //算法主体函数

{

for (int i=len/2-1;i>=0;i--)

HeapAdjust(a,i,len);

while (--len)

{

SWAP(a[0],a[len]);

HeapAdjust(a,0,len);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 归并排序

//归并

void Merge(int\*a,int len1,int len2)

{

int \*a1=new int[len1+1],\*a2=new int[len2+1],len=len1+len2;

for (int i=0;i<len1;i++)

a1[i]=a[i];

for (int i=0;i<len2;i++)

a2[i]=a[len1+i];

a1[len1]=a2[len2]=INT\_MAX;

for (int i=0,j=0,k=0;k<len;k++)

if (a1[i]<a2[j])

a[k]=a1[i++];

else

a[k]=a2[j++];

delete[] a1;delete[] a2;

}

//归并排序

void MergeSort(int\*a,int len) //算法主体函数

{

if (len>1)

{

int c=len/2;

MergeSort(a,c);

MergeSort(a+c,len-c);

Merge(a,c,len-c);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 快速排序

//划分

int Partition(int\*a,int len)

{

int x=a[--len],i=-1;

for (int j=0;j<len;j++)

if (a[j]<x)

{

i++;

SWAP(a[i],a[j]);

}

SWAP(a[i+1],a[len]);

return i+1;

}

//快速排序

void QuickSort(int\*a,int len) //算法主体函数

{

if (len > 0)

{

int q=Partition(a,len);

if (q<len-q)

{

QuickSort(a,q-1);

QuickSort(a+q+1,len-q-1);

}

else

{

QuickSort(a+q+1,len-q-1);

QuickSort(a,q-1);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*插入式希尔排序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 希尔排序

//希尔插入

void ShellInsert(int\*a,int inc,int len)

{

for (int i=inc;i<len;i+=inc)

{

int j=i,x=a[i];

while (j>0 && a[j-inc]>x)

a[j]=a[j-inc],j-=inc;

a[j]=x;

}

}

//插入式希尔排序

void ShellSort(int\*a,int len) //算法主体函数

{

int inc=len;

do

{

inc=inc/3+1;

for(int s=0;s<inc;s++)

ShellInsert(a-s,inc,len+s);

}

while (inc>1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## **同余运算**

(1）if a≡b(Mod m) and c≡d(Mod m) , then a+c≡b+d(Mod m)

(2）if a≡b(Mod m) and c≡d(Mod m) , then ac≡bd(Mod m)

(3）if a≡b(Mod m) k>0 , then ak≡bk(Mod mk)

(4）if a≡b(Mod m) and d|a,d|b,d|m , then a/d≡b/d(Mod m/d)

(5）if a≡b(Mod m) and d|m , then a≡b(Mod d)

(6）if ca≡cb(Mod m) and d=gcd(c,m) , then a≡b(Mod m/d)

若a≡0(Mod m)，称a被m整除

## 模线性方程问题

/\*

求最小的 N，使得 A + C \* N == B (mod 2^k)。也就是求模线性方程 C \* N == B - A (mod 2^k) 的最小解,N<4^k

推论1：方程ax=b(mod n)对于未知量x有解，当且仅当gcd(a,n) | b。

推论2：方程ax=b(mod n)或者对模n有d个不同的解，其中d=gcd(a,n)，或者无解。

定理1：设d=gcd(a,n)，假定对整数x和y满足d=ax+by(比如用扩展Euclid算法求出的一组解)。如果d | b，则方程ax=b(modn)有一个解x0满足x0=x\*(b/d) mod n 。特别的设e=x0+n，方程ax=b(mod n)的最小整数解x1=e mod(n/d)，最大整数解x2=x1+(d-1)\*(n/d)。

定理2：假设方程ax=b(mod n)有解，且x0是方程的任意一个解，则该方程对模n恰有d个不同的解(d=gcd(a,n))，分别为：xi=x0+i\*(n/d) mod n 。

以上定理的具体证明见《算法导论》

\*/

#include<iostream>

#include<cmath>

using namespace std;

#define mod(a,b) ((a)%(b)+(b))%(b) //把负数转化成为正数

typedef long long int64;

int64 ext\_euclid(int64 a,int64 b,int64 &x,int64 &y)

{

int64 t,d;

if (b==0)

{

x=1;

y=0;

return a;

}

d=ext\_euclid(b,mod(a,b),x,y);

t=x;

x=y;

y=t-a/b\*y;

return d;

}

void modular\_linear\_equation\_solver(int64 a,int64 b,int64 n)

{

int64 d;

int64 x,y;

d=ext\_euclid(a,n,x,y);

if(mod(b,d)>0)

cout<<"FOREVER"<<endl;

else cout<<mod((x\*(b/d)),n/d)<<endl;

}

int main()

{

int64 a,b,c,n;

int k;

while(cin>>a>>b>>c>>k)

{

if(!a&&!b&&!c&&!k)

break;

n=pow(2.0,k );

modular\_linear\_equation\_solver(c,b-a,n);

}

return 0;

}

方法二：

#ifdef WIN32

typedef \_\_int64 i64;

#else

typedef long long i64;

#endif

//扩展Euclid求解gcd(a,b)=ax+by

int ext\_gcd(int a,int b,int& x,int& y){

int t,ret;

if (!b){

x=1,y=0;

return a;

}

ret=ext\_gcd(b,a%b,x,y);

t=x,x=y,y=t-a/b\*y;

return ret;

}

//计算m^a, O(loga), 本身没什么用, 注意这个按位处理的方法 :-P

int exponent(int m,int a){

int ret=1;

for (;a;a>>=1,m\*=m)

if (a&1)

ret\*=m;

return ret;

}

//计算幂取模a^b mod n, O(logb)

int modular\_exponent(int a,int b,int n){ //a^b mod n

int ret=1;

for (;b;b>>=1,a=(int)((i64)a)\*a%n)

if (b&1)

ret=(int)((i64)ret)\*a%n;

return ret;

}

//求解模线性方程ax=b (mod n)

//返回解的个数,解保存在sol[]中

//要求n>0,解的范围0..n-1

int modular\_linear(int a,int b,int n,int\* sol){

int d,e,x,y,i;

d=ext\_gcd(a,n,x,y);

if (b%d)

return 0;

e=(x\*(b/d)%n+n)%n;

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

sol[i]=(e+i\*(n/d))%n;

return d;

}

//求解模线性方程组(中国余数定理)

// x = b[0] (mod w[0])

// x = b[1] (mod w[1])

// ...

// x = b[k-1] (mod w[k-1])

//要求w[i]>0,w[i]与w[j]互质,解的范围1..n,n=w[0]\*w[1]\*...\*w[k-1]

int modular\_linear\_system(int b[],int w[],int k){

int d,x,y,a=0,m,n=1,i;

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

n\*=w[i];

for (i=0;i<k;i++){

m=n/w[i];

d=ext\_gcd(w[i],m,x,y);

a=(a+y\*m\*b[i])%n;

}

return (a+n)%n;

}

## 计算a^n mod k

**//k<40000**

int powmod( int a, int n, int k )

{

int d = 1;

for (a %= k; n > 0; n >>= 1)

{

if(n&1)

d=(d\*a)%k;

a=(a\*a)%k;

}

return d;

}

## 阶乘最后非0位

//求阶乘最后非零位,复杂度O(nlogn)

//返回该位,n以字符串方式传入

#include <string.h>

#define MAXN 10000

int lastdigit(char\* buf){

const int mod[20]={1,1,2,6,4,2,2,4,2,8,4,4,8,4,6,8,8,6,8,2};

int len=strlen(buf),a[MAXN],i,c,ret=1;

if (len==1)

return mod[buf[0]-'0'];

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

a[i]=buf[len-1-i]-'0';

for (;len;len-=!a[len-1]){

ret=ret\*mod[a[1]%2\*10+a[0]]%5;

for (c=0,i=len-1;i>=0;i--)

c=c\*10+a[i],a[i]=c/5,c%=5;

}

return ret+ret%2\*5;

}

## 素数篇

除了1和此整数自身外，没法被其他自然数整除的数，又称质数。

方法一：

void prim()

{

long i,j,k;

prime[0]=2;

prime[1]=3;

jue[2]=true;

jue[3]=true;

jue[5]=true;

for(i=5,k=2;i<999983;i+=2)

{

for(j=0;prime[j]\*prime[j]<=i&&i%prime[j];)

{

j++;

if(prime[j]\*prime[j]>i)

{

prime[k++]=i;

jue[i]=true;

}

}

}

}

方法二：

#include <string.h>

bool used[21000];

void getprimes()

{

memset(used,true,sizeof(used));

used[0]=false;

used[1]=false;

for(int i=2;i<=21000;i++)

{

if(used[i])

for(int j=2\*i;j<=21000;j+=i)

used[j]=false;

}

}

### 线性筛法

const int UPBOUND = 100000;

int a[UPBOUND];

int p[UPBOUND],pn;

//号称线性的筛素数算法，实际性能确实不错

//p[]={2,3,5,7,...},pn为小于UPBOUND的素数个数

//若i是合数a[i]为i的最小因子，若i是素数a[i]=0

void primefilter()

{

int i, j;

for(i = 2; i < UPBOUND; ++i)

{

if(!(a[i])) p[pn++] = i;

for(j = 0; (j<pn && i\*p[j]<UPBOUND && (p[j]<=a[i]||a[i]==0)); ++j)

a[i\*p[j]] = p[j];

}

}

void prim() //建立素数表

{

long long i,j,k;

prime[0]=2;

prime[1]=3;

for(i=5,k=2;i<47000;i+=2)

{

for(j=1;prime[j]\*prime[j]<=i&&i%prime[j];j++);

if(prime[j]\*prime[j]>i)

prime[k++]=i;

}

}

void getprimes() //构建素数表

{

int i;

primes[1]=1;

for(i=2;i<=32;i++)

{

if(!primes[i])

for(int j=2\*i;j<=32;j+=i)

primes[j]=1;

}

}

### 费马小定理

若p为素数，a为正整数，那么一定有a^p≡a(Mod p)

### MR素性测试

给定整数a，若满足a^p≡a(Mod p)，p为一个素数，则a很可能也是素数

## 欧拉函数问题

#include<stdio.h>

int main()

{

long n,i,k;

double r;

while(scanf("%ld",&n)&&n)

{

r=n;

i=2;

while(n>1)

{

k=0;

while(n%i==0)

{

n/=i;

k=1;

}

if(k)

r\*=1.0-1.0/i;

if(i==2)

i++;

else

i=i+2;

}

printf("%.0lf\n",r);

}

return 0;

}

## 巴赫猜想

/\*

哥德巴赫猜想:每一个数最多可由3个素数相加表示，任意一个大于4的偶数必定可以表示为两奇素数之和

\*/

#include<stdio.h>

int cnt,n,tmp;

int ff[10001],f[10001],b[10001],primes[3000];

void prime()

{

int i,j,t;

for(i=0;i<10001;i++)

ff[i]=0;

for(i=2;i<=100;i++)

{

if(ff[i])

continue;

j=i\*i;

while(j<=10000){

ff[j]=1;

j+=i;

}

}

cnt=0;

for(i=2;i<=10000;i++)

if(ff[i]==0)

primes[cnt++]=i;

for(i=0;i<10001;i++)

f[i]=-1;

f[0]=0;

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

{

t=primes[i];

for(j=t;j<=3\*t;j++)

{

/////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*和包含t的数中，最小为t，最大为t+t+t

if(j>10000)

break;

if(f[j-t]!=-1&&(f[j]==-1||f[j]>f[j-t]+1))

{

f[j]=f[j-t]+1;

b[j]=t;

}

}

}

}

int main()

{

prime();

while(scanf("%d",&n)!=EOF)

{

if(n<=1)

printf("0\n");

else

{

if(ff[n]==0)

{

printf("1\n");

printf("%d\n",n);

continue;

}

else

{

printf("%d\n",f[n]);

while(n>0)

{

printf("%d",b[n]);

n-=b[n];

if(n==0)

printf("\n");

else

printf(" ");

}

}

}

}

return 0;

}

## 最大公约数

int gcd(int a , int b) //递归实现

{

return (b? gcd(b , a%b ) : a);

}

int gcd(int a , int b) //非递归实现

{

while (b){ int t = a % b; a = b; b = t; }

return a;

}

## 最小公倍数

lcm(a,b) = a/gcd(a,b)\*b

## 不定方程求解

二元线性不定方程：形如ax + by = c

不定方程求解程序：对于方程ax + by = gcd(a, b)

void euclid\_gcd(int a , int b , int & d , int & x , int & y)

{

if (!b){ d = a; x = 1; y = 0; }

else {

euclid\_gcd(b , a%b , d , y , x );

y -= x\*(a/b);

}

}

## 概率因子分解Pollard 算法

typedef \_\_int64 Long; //或自定义大整数类（必须重载运算符）

Long Pollard(Long n)

{

Long i=1, x=rand()%n, y=x, k=2;

while (1)

{

++i;

x = (x\*x-1) % n;

Long d = gcd(y-x, n);

if (d>1 && d<n) return d;

if (i==k)

{

y = x; k \*= 2;

}

}

return -1;

}

## 数值计算[2]

### 定积分计算(Romberg)

/\* Romberg求定积分

输入：积分区间[a,b]，被积函数f(x,y,z)

输出：积分结果

f(x,y,z)示例：

double f0( double x, double l, double t )

{

return sqrt(1.0+l\*l\*t\*t\*cos(t\*x)\*cos(t\*x));

}

\*/

double Integral(double a, double b, double (\*f)(double x, double y, double z), double eps,

double l, double t)

double Romberg (double a, double b, double (\*f)(double x, double y, double z), double eps,

double l, double t)

{

#define MAX\_N 1000

int i, j, temp2, min;

double h, R[2][MAX\_N], temp4;

for (i=0; i<MAX\_N; i++) {

R[0][i] = 0.0;

R[1][i] = 0.0;

}

h = b-a;

min = (int)(log(h\*10.0)/log(2.0)); //h should be at most 0.1

R[0][0] = ((\*f)(a, l, t)+(\*f)(b, l, t))\*h\*0.50;

i = 1;

temp2 = 1;

while (i<MAX\_N){

i++;

R[1][0] = 0.0;

for (j=1; j<=temp2; j++)

R[1][0] += (\*f)(a+h\*((double)j-0.50), l, t);

R[1][0] = (R[0][0] + h\*R[1][0])\*0.50;

temp4 = 4.0;

for (j=1; j<i; j++) {

R[1][j] = R[1][j-1] + (R[1][j-1]-R[0][j-1])/(temp4-1.0);

temp4 \*= 4.0;

}

if ((fabs(R[1][i-1]-R[0][i-2])<eps)&&(i>min))

return R[1][i-1];

h \*= 0.50;

temp2 \*= 2;

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

R[0][j] = R[1][j];

}

return R[1][MAX\_N-1];

}

double Integral(double a, double b, double (\*f)(double x, double y, double z), double eps,

double l, double t)

{

#define pi 3.1415926535897932

int n;

double R, p, res;

n = (int)(floor)(b \* t \* 0.50 / pi);

p = 2.0 \* pi / t;

res = b - (double)n \* p;

if (n)

R = Romberg (a, p, f0, eps/(double)n, l, t);

R = R \* (double)n + Romberg( 0.0, res, f0, eps, l, t );

return R/100.0;

}

### 多项式求根(牛顿法)

/\* 牛顿法解多项式的根

输入：多项式系数c[]，多项式度数n，求在[a,b]间的根

输出：根

要求保证[a,b]间有根

\*/

double fabs( double x )

{

return (x<0)? -x : x;

}

double f(int m, double c[], double x)

{

int i;

double p = c[m];

for (i=m; i>0; i--)

p = p\*x + c[i-1];

return p;

}

int newton(double x0, double \*r,

double c[], double cp[], int n,

double a, double b, double eps)

{

int MAX\_ITERATION = 1000;

int i = 1;

double x1, x2, fp, eps2 = eps/10.0;

x1 = x0;

while (i < MAX\_ITERATION) {

x2 = f(n, c, x1);

fp = f(n-1, cp, x1);

if ((fabs(fp)<0.000000001) && (fabs(x2)>1.0))

return 0;

x2 = x1 - x2/fp;

if (fabs(x1-x2)<eps2) {

if (x2<a || x2>b)

return 0;

\*r = x2;

return 1;

}

x1 = x2;

i++;

}

return 0;

}

double Polynomial\_Root(double c[], int n, double a, double b, double eps)

{

double \*cp;

int i;

double root;

cp = (double \*)calloc(n, sizeof(double));

for (i=n-1; i>=0; i--) {

cp[i] = (i+1)\*c[i+1];

}

if (a>b) {

root = a; a = b; b = root;

}

if ((!newton(a, &root, c, cp, n, a, b, eps)) &&

(!newton(b, &root, c, cp, n, a, b, eps)))

newton((a+b)\*0.5, &root, c, cp, n, a, b, eps);

free(cp);

if (fabs(root)<eps)

return fabs(root);

else

return root;

}

### 周期性方程(追赶法)

/\* 追赶法解周期性方程

周期性方程定义：| a1 b1 c1 ... | = x1

| a2 b2 c2 ... | = x2

| ... | \* X = ...

| cn-1 ... an-1 bn-1 | = xn-1

| bn cn an | = xn

输入：a[],b[],c[],x[]

输出：求解结果X在x[]中

\*/

void run()

{

c[0] /= b[0]; a[0] /= b[0]; x[0] /= b[0];

for (int i = 1; i < N - 1; i ++) {

double temp = b[i] - a[i] \* c[i - 1];

c[i] /= temp;

x[i] = (x[i] - a[i] \* x[i - 1]) / temp;

a[i] = -a[i] \* a[i - 1] / temp;

}

a[N - 2] = -a[N - 2] - c[N - 2];

for (int i = N - 3; i >= 0; i --) {

a[i] = -a[i] - c[i] \* a[i + 1];

x[i] -= c[i] \* x[i + 1];

}

x[N - 1] -= (c[N - 1] \* x[0] + a[N - 1] \* x[N - 2]);

x[N - 1] /= (c[N - 1] \* a[0] + a[N - 1] \* a[N - 2] + b[N - 1]);

for (int i = N - 2; i >= 0; i --)

x[i] += a[i] \* x[N - 1];

}

# 组合论[3]

### 组合公式

1. C(m,n)=C(m,m-n)

2. C(m,n)=C(m-1,n)+C(m-1,n-1)

derangement D(n) = n!(1 - 1/1! + 1/2! - 1/3! + ... + (-1)^n/n!)

= (n-1)(D(n-2) - D(n-1))

Q(n) = D(n) + D(n-1)

求和公式,k = 1..n

1. sum( k ) = n(n+1)/2

2. sum( 2k-1 ) = n^2

3. sum( k^2 ) = n(n+1)(2n+1)/6

4. sum( (2k-1)^2 ) = n(4n^2-1)/3

5. sum( k^3 ) = (n(n+1)/2)^2

6. sum( (2k-1)^3 ) = n^2(2n^2-1)

7. sum( k^4 ) = n(n+1)(2n+1)(3n^2+3n-1)/30

8. sum( k^5 ) = n^2(n+1)^2(2n^2+2n-1)/12

9. sum( k(k+1) ) = n(n+1)(n+2)/3

10. sum( k(k+1)(k+2) ) = n(n+1)(n+2)(n+3)/4

12. sum( k(k+1)(k+2)(k+3) ) = n(n+1)(n+2)(n+3)(n+4)/5

### 全组合排列

方法一：

#define MAX\_N 10

int n; //共n个数

int rcd[MAX\_N]; //记录每个位置填的数

int used[MAX\_N]; //标记数是否用过

int num[MAX\_N]; //存放输入的n个数

void full\_permutation(int L)

{

int i;

if (L == n+1)

{ //输出

return;

}

for (i=1; i<=n; i++)//枚举所有的数，循环从0开始

if (!used[i]) { //num[i]没有使用过

used[i] = 1; //标记为已使用

rcd[L] = i; //在l位置放上该数

full\_permutation(L+1); //填下一个位置

used[i] = 0; //清标记

}

}

方法二：

#include <iostream>

using namespace std;

int x[11]={0,1,2,3,4,5,6,7,8,9,10};

void permute(int t)//全排列

{

if(t>n)

output();

else

{

for(int i = t; i <= n; i++)

{

swap(x[t],x[i]);

permute(t+1);

swap(x[t],x[i]);

}

}

}

**测试函数**

Sample Input

3

1 2 3

Sample Output

123

132

213

231

312

321

//Program

#include<stdio.h>

#include<string.h>

const int maxn=11;

int n;

int used[maxn];//标记数组

int mat[maxn];//存储数组

int num[maxn];//输出数组

void solve(int l)

{

if(l>=n)

{

for(int i=0;i<n;++i)printf("%d", num[i]);

puts("");

return;

}

for(int i=0;i<n;++i)

{

if(!used[i])

{

used[i]=1;

num[l]=mat[i];

solve(l+1);

used[i]=0;

}

}

}

int main()

{

while(scanf("%d", &n)!=EOF)

{

for(int i=0;i<n;++i)scanf("%d", mat+i);

memset(used,0,sizeof(used));

solve(0);

}

return 0;

}

### 非重复组合排列

**含重复数字时，生成不重复组合排列**

Sample Input

4

1 2 2 3

Sample Output

1223

1232

1322

2123

2132

2213

2231

2312

2321

3122

3212

3221

//Program

#include<stdio.h>

const int maxn=10;

int n,var;

int Index;

int used[maxn],mat[maxn],num[maxn];

void push(int varNum){ //压栈

for(int i=0;i<Index;++i){

if(mat[i]==varNum){

++used[i];

return;

}

}

mat[Index]=varNum;

++used[Index++];

}

void solve(int l){ //求解

if(l>=n){

for(int i=0;i<n;++i)printf("%d", num[i]);

puts("");

return;

}

for(int i=0;i<Index;++i){

if(used[i]){

used[i]--;

num[l]=mat[i];

solve(l+1);

used[i]++;

}

}

}

int main(){

while(scanf("%d", &n)!=EOF){

Index=0;

for(int i=0;i<n;++i){

scanf("%d", &var);

push(var);

}

solve(0);

}

return 0;

}

### 类循环组合排列

Sample Input：

4 2

Sample Output

0000

0001

0010

0011

0100

0101

0110

0111

1000

1001

1010

1011

1100

1101

1110

1111

//Program

#include<stdio.h>

int n,m;

int mat[10];

void solve(int l)

{

if(l>=n)

{

for(int i=0;i<n;++i)

printf("%d", mat[i]);

puts("");

return;

}

for(int i=0;i<m;++i)

{

mat[l]=i;

solve(l+1);

}

}

int main()

{

while(scanf("%d%d", &n, &m)!=EOF)

{

solve(0);

}

return 0;

}

### 普通选择性组合排列

Sample Input

5 3

1 2 3 4 5

Sample Output

123

124

125

134

135

145

234

235

245

345

//Program

#include<stdio.h>

const int maxn=10;

int totalN,selectM;

int mat[maxn];//存储数组

int num[maxn];//输出数组

void solve(int startVar,int selectVar)

{

if(selectVar>=selectM)

{

for(int i=0;i<selectM;++i)printf("%d", num[i]);

puts("");

return;

}

for(int i=startVar;i<totalN;++i)

{

num[selectVar]=mat[i];

solve(i+1,selectVar+1);

}

}

int main()

{

while(scanf("%d%d", &totalN, &selectM)!=EOF)

{

for(int i=0;i<totalN;++i)scanf("%d", mat+i);

solve(0,0);

}

return 0;

}

### 生成全子集组合排列

Sample Input

4

1 2 3 4

Sample Output

1

12

123

1234

124

13

134

14

2

23

234

24

3

34

4

//Program

#include<stdio.h>

const int maxn=10;

int n;

int mat[maxn];

int num[maxn];

void solve(int cur\_totalVar,int nextVar)

{

for(int i=0;i<cur\_totalVar;++i)printf("%d", num[i]);

if(cur\_totalVar)puts("");

for(int i=nextVar;i<n;++i)

{

num[cur\_totalVar]=mat[i];

solve(cur\_totalVar+1,i+1);

}

}

int main(){

while(scanf("%d", &n)!=EOF)

{

for(int i=0;i<n;++i)scanf("%d",mat+i);

solve(0,0);

}

return 0;

}

**//注意：倘若需要输出空集（也即输出一个换行），可做如下修改**

**//在函数solve()中，将if(cur\_totalVar)puts(""); 改为puts("");**

### 非重复生成全子集组合排列

**含重复数字时，生成不重复全子集组合排列**

Sample Input

4

1 2 2 3

Sample Output

1

12

122

1223

123

13

2

22

223

23

3

//Program

#include<stdio.h>

const int maxn=10;

int n,var;

int Index;

int used[maxn],mat[maxn],num[maxn];

void push(int varNum)

{ //压栈

for(int i=0;i<Index;++i)

{

if(mat[i]==varNum)

{

++used[i];

return;

}

}

mat[Index]=varNum;

++used[Index++];

}

void solve(int l,int p)

{ //求解

for(int i=0;i<l;++i)printf("%d", num[i]);

if(l)puts("");

for(int i=p;i<Index;++i)

{

if(used[i])

{

used[i]--;

num[l]=mat[i];

solve(l+1,i);

used[i]++;

}

}

}

int main()

{

while(scanf("%d", &n)!=EOF)

{

Index=0;

for(int i=0;i<n;++i)

{

scanf("%d", &var);

push(var);

}

solve(0,0);

}

return 0;

}

//注意：倘若需要输出空集（也即输出一个换行），可做如下修改

//在函数solve()中，将if(l)puts(""); 改为puts("");

### 一般组合

select\_combination ( int Step, int Pos )

{

// 已经确定Step-1个数字, 第Step步的数为X[Pos,…,n]的任一个

if ( Step == m+1 )

{ // 已确定数字数目为m时，

Output(); // 输出一种组合

return ;

} for (i=Pos; i<=n; i++)

{

A[Step] = X[i];

select\_combination ( Step + 1 , Pos + 1 );

}

}

**一般组合的优化**

select\_combination ( int Step, int Pos )

{

// 已经确定Step-1个数字, 第Step步的数为X[Pos,…,n]的任一个

if ( Step == m+1 )

{ // 已确定数字数目为m时，

Output(); // 输出一种组合

return ;

} for (i=Pos;i<=n-m+Step; i++)

{

A[Step] = X[i];

select\_combination ( Step + 1 , Pos + 1 );

}

}

总结：递归思想在搜索技术中有着广泛的应用，应当熟练掌握和学习这种思想，对求解这一类搜索题目大有裨益。

### 排列组合生成

//gen\_perm产生字典序排列P(n,m)

//gen\_comb产生字典序组合C(n,m)

//gen\_perm\_swap产生相邻位对换全排列P(n,n)

//产生元素用1..n表示

//dummy为产生后调用的函数,传入a[]和n,a[0]..a[n-1]为一次产生的结果

#define MAXN 100

int count;

#include <iostream.h>

void dummy(int\* a,int n){

int i;

cout<<count++<<": ";

for (i=0;i<n-1;i++)

cout<<a[i]<<' ';

cout<<a[n-1]<<endl;

}

void \_gen\_perm(int\* a,int n,int m,int l,int\* temp,int\* tag){

int i;

if (l==m)

dummy(temp,m);

else

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

if (!tag[i]){

temp[l]=a[i],tag[i]=1;

\_gen\_perm(a,n,m,l+1,temp,tag);

tag[i]=0;

}

}

void gen\_perm(int n,int m){

int a[MAXN],temp[MAXN],tag[MAXN]={0},i;

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

a[i]=i+1;

\_gen\_perm(a,n,m,0,temp,tag);

}

void \_gen\_comb(int\* a,int s,int e,int m,int& count,int\* temp){

int i;

if (!m)

dummy(temp,count);

else

for (i=s;i<=e-m+1;i++){

temp[count++]=a[i];

\_gen\_comb(a,i+1,e,m-1,count,temp);

count--;

}

}

void gen\_comb(int n,int m){

int a[MAXN],temp[MAXN],count=0,i;

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

a[i]=i+1;

\_gen\_comb(a,0,n-1,m,count,temp);

}

void \_gen\_perm\_swap(int\* a,int n,int l,int\* pos,int\* dir){

int i,p1,p2,t;

if (l==n)

dummy(a,n);

else{

\_gen\_perm\_swap(a,n,l+1,pos,dir);

for (i=0;i<l;i++){

p2=(p1=pos[l])+dir[l];

t=a[p1],a[p1]=a[p2],a[p2]=t;

pos[a[p1]-1]=p1,pos[a[p2]-1]=p2;

\_gen\_perm\_swap(a,n,l+1,pos,dir);

}

dir[l]=-dir[l];

}

}

void gen\_perm\_swap(int n){

int a[MAXN],pos[MAXN],dir[MAXN],i;

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

a[i]=i+1,pos[i]=i,dir[i]=-1;

\_gen\_perm\_swap(a,n,0,pos,dir);

}

### 生成gray码

//生成reflected gray code

//每次调用gray取得下一个码

//000...000是第一个码,100...000是最后一个码

void gray(int n,int \*code){

int t=0,i;

for (i=0;i<n;t+=code[i++]);

if (t&1)

for (n--;!code[n];n--);

code[n-1]=1-code[n-1];

}

### 置换(polya)

//求置换的循环节,polya原理

//perm[0..n-1]为0..n-1的一个置换(排列)

//返回置换最小周期,num返回循环节个数

#define MAXN 1000

int gcd(int a,int b){

return b?gcd(b,a%b):a;

}

int polya(int\* perm,int n,int& num){

int i,j,p,v[MAXN]={0},ret=1;

for (num=i=0;i<n;i++)

if (!v[i]){

for (num++,j=0,p=i;!v[p=perm[p]];j++)

v[p]=1;

ret\*=j/gcd(ret,j);

}

return ret;

}

### 字典序全排列

//字典序全排列与序号的转换

int perm2num(int n,int \*p){

int i,j,ret=0,k=1;

for (i=n-2;i>=0;k\*=n-(i--))

for (j=i+1;j<n;j++)

if (p[j]<p[i])

ret+=k;

return ret;

}

void num2perm(int n,int \*p,int t){

int i,j;

for (i=n-1;i>=0;i--)

p[i]=t%(n-i),t/=n-i;

for (i=n-1;i;i--)

for (j=i-1;j>=0;j--)

if (p[j]<=p[i])

p[i]++;

}

### 字典序组合

//字典序组合与序号的转换

//comb为组合数C(n,m),必要时换成大数,注意处理C(n,m)=0|n<m

int comb(int n,int m){

int ret=1,i;

m=m<(n-m)?m:(n-m);

for (i=n-m+1;i<=n;ret\*=(i++));

for (i=1;i<=m;ret/=(i++));

return m<0?0:ret;

}

int comb2num(int n,int m,int \*c){

int ret=comb(n,m),i;

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

ret-=comb(n-c[i],m-i);

return ret;

}

void num2comb(int n,int m,int\* c,int t){

int i,j=1,k;

for (i=0;i<m;c[i++]=j++)

for (;t>(k=comb(n-j,m-i-1));t-=k,j++);

}

# 计算几何

## 对于任意多边形求面积

对于平面内的任意一个多边行,我们要求其面积,可以采用这样一来的思想方法:

//万涛的理解:多边形每相邻两点与原点形成的矢量面积(叉积)之和.

double av(double v)

{

if(v<0)

v=-v;

return v;

}

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

{

c=x[i]\*y[(i+1)%n]

d=y[i]\*x[(i+1)%n];

area+=c;

area-=d;

printf("c=%.2lf\n",c-d);

}

area=av(area)/2;

## 判断两线是否相交

//判断点在直线的上或下及左或右側.具体是采用点斜式求出一条线段所在直线的方程,然后将点代如求值进行判断

参照网站:计算几何算法概览

double DirectionV3(double x1, double y1, double x2, double y2, double x3, double y3){

return x1\*y3+x2\*y1+x3\*y2-x1\*y2-x2\*y3-x3\*y1;

}

bool Intersect(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)

{

依据:两条线段不相交,必定是存在其中一条线段的两端点在在另一条线段所在直线的同側.所以要分别考虑两种情况.

if(DirectionV3(x1, y1, x2, y2, x3, y3)\*DirectionV3(x1, y1, x2, y2, x4, y4)>0)//一条线段的两点全在另一条线段的同一側

return false;

if(DirectionV3(x3, y3, x4, y4, x1, y1)\*DirectionV3(x3, y3, x4, y4, x2, y2)>0)//

return false;

return true;

}

一。最佳原理（DP思想的精华）：

假设为了解决某一优化问题，需要依次作出N个决策D1,D2,....,DN.如若这个决策序列是最优的，对于任何一个整数K,1<K<N，不论前面K个决策是怎样的，以后的最优决策只取决于由前面决策所确定的当前状态，即以后的决策DK+1,DK+2,...DN也是最优的。

## 二分查找模板

//This is a binary-search mode .

int Bsearch(int b,int num[],int n)

{

int mid,l=0,r=n-1;

while(l<=r)

{

mid=(r+l)/2;

if(b<num[min]) r=mid-1; //less than it.

else if(b>num[min]) l=mid+1; //bigger than it.

else return min; //Just find it and return.

}

return -1; //fail to find.

}

## 二分图匹配模板

//匈牙利算法实现

int Bipartite(bool vc [][MAX],int nv1,int nv2) {

int i, j, x, n;

int q[MAX], prev[MAX], qs, qe;

int vm1[MAX], vm2[MAX];

n = 0;

for( i = 0; i < nv1; i++ ) vm1 = -1;

for( i = 0; i < nv2; i++ ) vm2 = -1;

for( i = 0; i < nv1; i++ ) {

for( j = 0; j < nv2; j++ ) prev[j] = -2;

qs = qe = 0;

for( j = 0; j < nv2; j++ ) if( vc[j] ) {

prev[j] = -1;

q[qe++] = j;

}

while( qs < qe ) {

x = q[qs];

if( vm2[x] == -1 ) break;

qs++;

for( j = 0; j < nv2; j++ ) if( prev[j] == -2 && vc[vm2[x]][j] ) {

prev[j] = x;

q[qe++] = j;

}

}

if( qs == qe ) continue;

while( prev[x] > -1 ) {

vm1[vm2[prev[x]]] = x;

vm2[x] = vm2[prev[x]];

x = prev[x];

}

vm2[x] = i;

vm1 = x;

n++;

}

return n;

}

# 动态规划

## 求局部最大和模板

定义两个变量max,change,对于一维数组a[]中的连续最大和,我们可以采取以下的循环完成.

/\*\*\*\*\*\*\*\*\*\*\*\*\*/

change=0;

max=a[1];

for (i=2;i<=n;i++) //'n' is the lenth of array a[].

{

if (change+a[ i ]>0)

change+=a[ i ]; //"change" is changable

else

{

change=0;

if(max<a[ i ])max=a[ i ];

continue;

}

if (max<change) max=change;

b[ i ]=max;// 如果我们要求1~n 的数的所有的局部最大和.

# 字符串

## 字符计数应用

判断两个字符串是否含有相同种类和个数的字符，两个字符串的长度一致

int judge(int len) //判断两个字符串是否含有相同种类和个数的字符

{

int i;

char ch1[26]={0},ch2[26]={0};

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

{

ch1[in[i]-'a']=ch1[in[i]-'a']+1; //统计第一个字符串中不同种类字符串的个数和种类

ch2[ou[i]-'a']=ch2[ou[i]-'a']+1;

}

if(strcmp(ch1,ch2)==0) //判断函数

return 1;

return 0;

}

## 最大子序列算法

#include<stdio.h>

int main()

{

int h[30000];

int n,i,j,max;

while(scanf("%d",&n)!=EOF)

{

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

scanf("%d",&h[i]);

int cont[30000]={0};

for(i=n-1;i>=0;i--)

{

for(j=n-1,max=0;j>i;j--)

{

if(h[i]>=h[j]&&max<cont[j])

max=cont[j];

}

cont[i]=max+1;

}

for(i=1,max=0;i<n;i++)

{

if(cont[max]<cont[i])

max=i;

}

printf("The number is:%d\n%d ",cont[max],h[max]);

for(i=max;i<n;i++)

{

if(cont[i]==(cont[max]-1)&&h[max]>h[i])

{

printf("%d ",h[i]);

max=i;

}

}

printf("\n");

}

return 0;

}

/\*

8

389 207 155 300 299 170 158 65

9

1 389 207 155 300 299 170 158 65

8

207 155 300 299 170 158 65 389

8

65 207 155 300 389 299 170 158

\*/

## 求最长公供子串（DP算法）

int Longest common strings(int L [ ] [ MAX ],char a [ ],char b [ ],int m,int n)

{

int i=1;j;

for(i=0;i<m;i++) L [ i ] [ 0 ]=0;

for(i=0;i<n;i++) L [ 0 ] [ i ]=0;

while(i<=m)

{

j=1;

while(j<=n)

{

if(a [ i ] ==b [ j ]) L[ i ] [ j ]= L [ i-1 ] [ j-1 ] +1;

else L[ i ] [ j ]=L[ i-1 ] [ j ] > L[ i ] [ j-1 ] ? L[ i-1 ] [ j ] : L[ i ] [ j-1 ];

j++;

}

}

return L [ m ] [ n ];

}

# 数据结构

## 图论

### 入度和出度表的转化

/\*

5 12

ABCDE

A D 1

A B 1

B E 1

B C 1

B A 1

C E 1

C D 1

C B 1

D C 1

D A 1

E C 1

E B 1

\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct ArcNode

{

int adjvex;

int arclength;

struct ArcNode \*nextarc;

}ArcNode;

typedef struct Vnode

{

char ch;

ArcNode \*firstarc;

}Vnode,AdjList[20];

typedef struct

{

AdjList vertices;

int vexnum,arcnum;

}ALGraph;

int LocateVex(ALGraph G, char v)

{

int i;

for(i=0;i<G.vexnum;i++)

{

if(G.vertices[i].ch==v)

return i;

}

// return -1;

}

void GreateDG(ALGraph &G)

{

int i,j,ar;

char v1,v2;

ArcNode \*arcnode;

printf("请输入顶点数和弧数:");

scanf("%d%d",&G.vexnum,&G.arcnum);

getchar();

printf("请输入各个顶点:\n");

for(i=0;i<G.vexnum;i++)

{

scanf("%c",&G.vertices[i].ch);

G.vertices[i].firstarc=NULL;

}

getchar();

for(i=0;i<G.arcnum;i++)

{

printf("请输入联通的两个顶点以及他们之间的弧度:");

scanf("%c %c %d",&v1,&v2,&ar);

getchar();

arcnode=(ArcNode \*)malloc(sizeof(ArcNode));

j=LocateVex(G,v2);

arcnode->adjvex=j;

arcnode->arclength=ar;

j=LocateVex(G,v1);

arcnode->nextarc=G.vertices[j].firstarc;

G.vertices[j].firstarc=arcnode;

}

}

void changer(ALGraph G,ALGraph &G2)

{

int i;

ArcNode \*p,\*q;

for(i=0;i<G.vexnum;i++)

{

G2.vertices[i].ch=G.vertices[i].ch;

G2.vertices[i].firstarc=NULL;

}

G2.arcnum=G.arcnum;

G2.vexnum=G.vexnum;

for(i=0;i<G.vexnum;i++)

{

p=G.vertices[i].firstarc;

while(p)

{

q=p->nextarc;

G.vertices[i].firstarc=q;

p->nextarc=G2.vertices[p->adjvex].firstarc;

G2.vertices[p->adjvex].firstarc=p;

p->adjvex=i;

p=q;

}

}

}

void print(ALGraph G)

{

ArcNode \*p;

int i;

for(i=0;i<G.vexnum;i++)

{

printf("%c->",G.vertices[i].ch);

p=G.vertices[i].firstarc;

while(p)

{

printf("%c->",G.vertices[p->adjvex]);

p=p->nextarc;

}

printf("NULL\n");

}

}

void main()

{

ALGraph G,G2;

GreateDG(G);

printf("生成的出度表为:\n");

print(G);

changer(G,G2);

printf("生成的入度表为:\n");

print(G2);

}

### 生成树

#include<stdio.h>

#include<string.h>

#include<malloc.h>

#include<stdlib.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define OVERFLOW 0

//#define MAX\_NAME 3 // 顶点字符串的最大长度+1

#define MAX\_VERTEX\_NUM 20 // 最大顶点个数

//enum GraphKind{DG,DN,UDG,UDN}; // {有向图,有向网,无向图,无向网}

typedef int InfoType; // 相关信息类型

typedef char VertexType[MAX\_VERTEX\_NUM]; // 顶点类型

//typedef char VertexTyped[MAX\_NAME];

typedef VertexType TElemType; // 定义树的元素类型为图的顶点类型

typedef int InfoType; // 权值类型

bool visited[MAX\_VERTEX\_NUM]; // 访问标志数组(全局量)

void(\*VisitFunc)(char\* v); // 函数变量(全局量)

//--------树的二叉链表(孩子—兄弟)存储表示-------------------------------

typedef struct CSNode

{

TElemType data;

CSNode \*firstchild,\*nextsibling;

}CSNode,\*CSTree;

//------------------图的邻接表存储表示--------------------------

typedef struct ArcNode

{

int adjvex; // 该弧所指向的顶点的位置(指示与该顶点邻接的点在图中的位置)

ArcNode \*nextarc; // 指向下一条弧的指针

InfoType \*info; // 该弧相关信息的指针

}ArcNode; //(边的信息结点，图的边集合模型)表结点

typedef struct VNode

{

VertexType data; // 顶点信息

ArcNode \*firstarc; // 指向第一条依附该顶点的弧的指针

}VNode,AdjList[MAX\_VERTEX\_NUM]; // 表头结点

typedef struct ALGraph

{

AdjList vertices; //图的点集合模型

int vexnum,arcnum; // 图的当前顶点数和弧数

}ALGraph;

void PreOrderTraverse(CSTree T,void(\*Visit)(TElemType))

{ // 先根遍历孩子—兄弟二叉链表结构的树T

if(T)

{

Visit(T->data); // 先访问根结点

PreOrderTraverse(T->firstchild,Visit); // 再先根遍历长子子树

PreOrderTraverse(T->nextsibling,Visit); // 最后先根遍历下一个兄弟子树

}

}

int LocateVex(ALGraph G,VertexType u)

{// 操作结果：若G中存在顶点u，则返回该顶点在图中位置；否则返回-1

int i;

for(i=0;i<G.vexnum;++i)

if(strcmp(u,G.vertices[i].data)==0)

return i;

return -1;

}

void CreateGraph(ALGraph &G)

{ // 采用邻接表存储结构，构造有向图

int i,j,k;

VertexType va,vb; // 连接边或弧的2顶点

ArcNode \*s;

printf("请输入图的顶点数,边数: ");

scanf("%d,%d",&G.vexnum,&G.arcnum);

printf("请输入%d个顶点的值:\n",G.vexnum);

for(i=0;i<G.vexnum;++i) // 构造顶点向量

{

scanf("%s",G.vertices[i].data);

G.vertices[i].firstarc=NULL; // 初始化与该顶点有关的出弧链表

}

printf("请输入每条弧(边)的弧尾和弧头以及权值(以空格作为间隔):\n");

for(k=0;k<G.arcnum;++k) // 构造相关弧链表

{

scanf("%s%s",va,vb);

i=LocateVex(G,va); // 弧尾

j=LocateVex(G,vb); // 弧头

s=(ArcNode\*)malloc(sizeof(ArcNode));

//s->info=(int \*)malloc(sizeof(int)); // 动态生成存放权值的空间

//\*(s->info)=w;

s->adjvex=j;

s->nextarc=G.vertices[i].firstarc;

G.vertices[i].firstarc=s;

}

}

int FirstAdjVex(ALGraph G,VertexType v)

{

//返回v的第一个邻接顶点的序号。若顶点在G中没有邻接顶点，则返回-1

ArcNode \*p;

int v1;

v1=LocateVex(G,v); // v1为顶点v在图G中的序号

p=G.vertices[v1].firstarc;

if(p)

return p->adjvex;

else

return -1;

}

int NextAdjVex(ALGraph G, VertexType v, VertexType w)

{

//返回v的(相对于w的)下一个邻接点

ArcNode \*p;

int v1,w1;

v1=LocateVex(G,v);

w1=LocateVex(G,w);

p=G.vertices[v1].firstarc;

while(p!=NULL)

{

if(visited[p->adjvex]!=TRUE&&p->adjvex!=w1)

return p->adjvex;

p=p->nextarc;

}

return -1;

}//NextAdjVex

void DFSTree(ALGraph G,int v,CSTree &T)

{

// 从第v个顶点出发深度优先遍历图G，建立以T为根的生成树。算法7.8

bool first=TRUE;

int w;

CSTree p,q;

visited[v]=TRUE;

for(w=FirstAdjVex(G,G.vertices[v].data);w>=0;w=NextAdjVex(G,G.vertices[v].data,G.vertices[w].data)) //w依次为v的邻接顶点

{

if(!visited[w]) // w顶点不曾被访问

{

p=(CSTree)malloc(sizeof(CSNode)); // 分配孩子结点

strcpy(p->data,G.vertices[w].data);

p->firstchild=NULL;

p->nextsibling=NULL;

if(first)

{

// w是v的第一个未被访问的邻接顶点

T->firstchild=p;

first=FALSE; // 是根的第一个孩子结点

}

else // w是v的其它未被访问的邻接顶点

q->nextsibling=p; //是上一邻接顶点的兄弟姐妹结点(第1次不通过此处，以后q已赋值)

q=p;

DFSTree(G,w,q); // 从第w个顶点出发深度优先遍历图G，建立子生成树q

}

}

}

void visit(char \*i)

{

printf("%s ",i);

}

void Display(ALGraph G)

{

// 输出图的邻接矩阵G

int i;

ArcNode \*p;

/\*switch(G.kind)

{

case DG: printf("有向图\n");

break;

case DN: printf("有向网\n");

break;

case UDG:printf("无向图\n");

break;

case UDN:printf("无向网\n");

}\*/

printf("%d个顶点：\n",G.vexnum);

for(i=0;i<G.vexnum;++i)

printf("%s ",G.vertices[i].data);

printf("\n%d条弧(边):\n",G.arcnum);

for(i=0;i<G.vexnum;i++)

{

p=G.vertices[i].firstarc;

while(p)

{

printf("%s→%s ",G.vertices[i].data,G.vertices[p->adjvex].data);

p=p->nextarc;

}

printf("\n");

}

}

void main()

{

ALGraph G;

CSTree t;

CreateGraph(G); // 构造无向图G

Display(G); // 输出无向图g

t=(CSTree)malloc(sizeof(CSNode));

t->firstchild=NULL;

t->nextsibling=NULL;

strcpy(t->data,G.vertices[0].data);

DFSTree(G,0,t);

//DFSForest(g,t); // 建立无向图g的深度优先生成森林的孩子—兄弟链表t

printf("先序遍历生成树：\n");

PreOrderTraverse(t,visit); // 先序遍历生成森林的孩子—兄弟链表t

printf("\n");

}

### 最小生成树

/\*

6 10

ABCDEF

A B 6

A C 1

A D 5

B C 5

B E 3

C D 5

C E 6

C F 4

D F 2

E F 6

\*/

#include<stdio.h>

#include<string.h>

typedef struct

{

char ch[100];

int arc[100][100];

int vexnum,arcnum;

}MGraph;

typedef struct

{

char adjvex;

int lowcost;

}Closedge[100];//记录从顶点集U到V-U的代价最小的边的辅助数组定义

int LocateVex(MGraph G, char v)

{

int i;

for(i=0;i<G.vexnum;i++)

{

if(G.ch[i]==v)

return i;

}

return -1;

}

int CreateUND(MGraph &G)

{

int i,j,w,k1,k2;

char v1,v2;

printf("请输入定点数和狐数:");

scanf("%d %d",&G.vexnum,&G.arcnum);

getchar();

printf("请输入各个顶点:\n");

for(i=0;i<G.vexnum;i++)

scanf("%c",&G.ch[i]);

getchar();

for(i=0;i<G.vexnum;i++)

{

for(j=0;j<G.vexnum;j++)

G.arc[i][j]=10000;

}

for(j=0;j<G.arcnum;j++)

{

printf("请输入联通的结点以及之间的弧长:");

scanf("%c %c %d",&v1,&v2,&w); //注意输入格式

getchar();

k1=LocateVex(G,v1);

k2=LocateVex(G,v2);

if(k1>=0&&k2>=0)

{

G.arc[k1][k2]=w;

G.arc[k2][k1]=w;

}

else

{

printf("输入有误！\n");

return -1;

}

}

return 1;

}

int minimum(Closedge cl,MGraph G)// 求closedge.lowcost的最小正值

{

int i,k,min=10000;

for(i=0;i<G.vexnum;i++)

{

if(cl[i].lowcost>0)

{

if(min>cl[i].lowcost)

{

min=cl[i].lowcost;

k=i;

}

}

}

return k;

}

void MiniSpanTree\_PRIM(MGraph G,char v)

{

int i,j,k;

Closedge closedge;

k=LocateVex(G,v);

for(j=0;j<G.vexnum;j++)

{

if(j!=k)

{

closedge[j].lowcost=G.arc[k][j];

closedge[j].adjvex=v;

}

}

closedge[k].lowcost=0;

for(i=1;i<G.vexnum;i++)

{

k=minimum(closedge,G);

printf("%c %c\n",closedge[k].adjvex,G.ch[k]);//输出生成树的边

closedge[k].lowcost=0;

for(j=0;j<G.vexnum;++j)

{

if(G.arc[k][j]<closedge[j].lowcost)

{

closedge[j].lowcost=G.arc[k][j];

closedge[j].adjvex=G.ch[k];

}

}

}

}

void main()

{

MGraph G;

if(CreateUND(G))

{

MiniSpanTree\_PRIM(G,G.ch[0]);

}

}

### 关键路径

/\*

9 11

v1 v2 v3 v4 v5 v6 v7 v8 v9

v1 v2 6

v1 v3 4

v1 v4 5

v2 v5 1

v3 v5 1

v4 v6 2

v5 v7 9

v5 v8 1

v6 v8 4

v7 v9 2

v8 v9 4

\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAX\_VEXTEX\_NUM 20

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 10

#define OK 1

#define ERROR 0

typedef int ElemType;

int ve[MAX\_VEXTEX\_NUM];

int vl[MAX\_VEXTEX\_NUM];

//////////////////////////////

typedef struct ArcNode

{

int adjvex;

struct ArcNode \*nextarc;

int weight;

}ArcNode;

typedef struct VNode

{

char ver\_name[10];

ArcNode \*firstarc;

}VNode,AdjList[MAX\_VEXTEX\_NUM];

typedef struct

{

AdjList vertices;

int vexnum, arcnum;

}ALGraph;

typedef struct //构建栈

{

ElemType \*base;

ElemType \*top;

int stacksize;

}SqStack;

//////////////////////////////

void InitStack(SqStack &S)//初始化栈

{

S.base=(ElemType \*)malloc(STACK\_INIT\_SIZE\*sizeof(ElemType));

if(!S.base) exit(ERROR);

S.top=S.base;

S.stacksize=STACK\_INIT\_SIZE;

}

//////////////////////////////

void Pop(SqStack &S,int &e)

{

if(S.top==S.base)

printf("ERROR");

S.top--;

e=\*S.top;

}

//////////////////////////////

void Push(SqStack &S,ElemType e)//进栈操作

{

if(S.top-S.base>=S.stacksize)

{

S.base = (ElemType \*)realloc(S.base,(S.stacksize+STACKINCREMENT)\*sizeof(ElemType));

if(!S.base)

exit(ERROR);

S.top=S.base+S.stacksize;

S.stacksize+=STACKINCREMENT;

}

\*S.top++=e;

}

//////////////////////////////

int StackEmpty(SqStack &S)//判断栈是否为空

{

if(S.top==S.base)

return OK;

else

return ERROR;

}

//////////////////////////////

int Location(ALGraph G,char ch[10])//得到当前顶点的下表

{

int i;

for(i=0;i<G.vexnum;i++)

{

if(!strcmp(ch,G.vertices[i].ver\_name))

return i;

}

return -1;

}

//////////////////////////////

void CreatGraph(ALGraph &G) //构建图

{

int m,n,i,weight;

char sh[10],ch[10]; //sh表示弧头定点名，ch表示弧尾顶点名

ArcNode \*p;

printf("请输入顶点事件数和活动数:");

scanf("%d%d",&G.vexnum,&G.arcnum);

printf("请输入%d个顶点,用空格隔开:",G.vexnum);

for(i=0;i<G.vexnum;i++)

{

getchar();

scanf("%s",G.vertices[i].ver\_name);

G.vertices[i].firstarc=NULL;

}

for(i=0;i<G.arcnum; i++) //输入存在的活动的点集合及活动所需时间

{

printf("\n请输入存在活动的两个顶点的序号及活动所需时间:");

scanf("%s%s%d",sh,ch,&weight);

getchar();

while(Location(G,sh)==-1||Location(G,sh)==-1)

{

printf("输入的顶点序号不正确 请重新输入:");

scanf("%s%s%d",sh,ch,&weight);

}

p=(ArcNode\*)malloc(sizeof(ArcNode));

if(p==NULL)

{

printf("memory allocation failed");

exit(1);

}

m=Location(G,ch);

p->adjvex=m;

p->weight=weight;

n=Location(G,sh);

p->nextarc=G.vertices[n].firstarc;

G.vertices[n].firstarc=p;

}

}

//////////////////////////////

void FindInDegree(ALGraph G, int indegree[])//求入度操作

{

int i;

for(i=0;i<G.vexnum;i++)

indegree[i]=0;

for(i=0;i<G.vexnum;i++)

{

while(G.vertices[i].firstarc)

{

indegree[G.vertices[i].firstarc->adjvex]++;

G.vertices[i].firstarc=G.vertices[i].firstarc->nextarc;

}

}

}

//////////////////////////////

void TopologicalOrder(ALGraph G,SqStack &T)

{

SqStack S;

ArcNode \*p;

int indegree[MAX\_VEXTEX\_NUM]={0};

int i,count,k,j;

FindInDegree(G,indegree);

InitStack(S);

for(i=0;i<G.vexnum;i++)

if(!indegree[i])

Push(S,i);

InitStack(T);

count=0;

for(i=0;i<G.vexnum;i++)

ve[i]=0;

while(S.base!=S.top)

{

Pop(S,j);

Push(T,j);

count++;

for(p=G.vertices[j].firstarc;p;p=p->nextarc)

{

k=p->adjvex;

if(!(--indegree[k]))

Push(S,k);

if(ve[j]+p->weight>ve[k])

ve[k]=ve[j]+p->weight;

}

}

if(count<G.vexnum)

printf("ERROR");

}

//////////////////////////////

void CriticalPath(ALGraph G)

{

SqStack T;

int i,j,k,ee,el;

char tag;

ArcNode \*p;

TopologicalOrder(G,T);

for(i=0;i<G.vexnum;i++)

vl[i]=ve[i];

while(T.base!=T.top)

{

for(Pop(T,j),p=G.vertices[j].firstarc;p;p=p->nextarc)

{

k=p->adjvex;

if(vl[j]+p->weight<vl[k])

vl[j]=vl[k]-p->weight;

}

}

for(j=0;j<G.vexnum;j++)

{

printf("事件%s最早发生时间为:%d，最晚发生时间为:%d。\n",G.vertices[j].ver\_name,ve[j],vl[j]);

for(p=G.vertices[j].firstarc;p;p=p->nextarc)

{

k=p->adjvex;

ee=ve[j];

el=vl[k]-p->weight;

tag=(ee==el)?'\*':'#';

printf("活动%s->%s是否为关键路径%c(\*表示是，#表示否)\n",G.vertices[j].ver\_name,G.vertices[k].ver\_name,tag);

}

}

}

//////////////////////////////

void main()

{

ALGraph G;

CreatGraph(G);

CriticalPath(G);

getchar();

}

### 迷宫算法的队列实现

#include<stdio.h>

#include<stdlib.h>

typedef struct LNode

{

int i;

int j;

int cont;

}LNode;

void MazePath(int num[7][7])

{

LNode \*L,\*newbase;

int use[7][7];

int t,p,cont[100],k=0;

int leng=100,inleng=20;

L=(LNode \*)malloc(leng\*sizeof(LNode));

L->i=1;

L->j=1;

L->cont=0;

for(t=0;t<7;t++)

{

for(p=0;p<7;p++)

use[t][p]=0;

}

for(t=0,p=1;;t++)

{

if(use[(L+t)->i][(L+t)->j]==0)

{

if(num[(L+t)->i-1][(L+t)->j]==0)

{

(L+p)->i=(L+t)->i-1;

(L+p)->j=(L+t)->j;

(L+p)->cont=t;

if((L+p)->i==5&&(L+p)->j==5)

break;

p++;

}

if(p==100)

{

newbase=(LNode \*)realloc(L,(leng+inleng)\*sizeof(LNode));

L=newbase;

leng=leng+inleng;

}

if(num[(L+t)->i+1][(L+t)->j]==0)

{

(L+p)->i=(L+t)->i+1;

(L+p)->j=(L+t)->j;

(L+p)->cont=t;

if((L+p)->i==5&&(L+p)->j==5)

break;

p++;

}

if(p==100)

{

newbase=(LNode \*)realloc(L,(leng+inleng)\*sizeof(LNode));

L=newbase;

leng=leng+inleng;

}

if(num[(L+t)->i][(L+t)->j-1]==0)

{

(L+p)->i=(L+t)->i;

(L+p)->j=(L+t)->j-1;

(L+p)->cont=t;

if((L+p)->i==5&&(L+p)->j==5)

break;

p++;

}

if(p==100)

{

newbase=(LNode \*)realloc(L,(leng+inleng)\*sizeof(LNode));

L=newbase;

leng=leng+inleng;

}

if(num[(L+t)->i][(L+t)->j+1]==0)

{

(L+p)->i=(L+t)->i;

(L+p)->j=(L+t)->j+1;

(L+p)->cont=t;

if((L+p)->i==5&&(L+p)->j==5)

break;

p++;

}

if(p==100)

{

newbase=(LNode \*)realloc(L,(leng+inleng)\*sizeof(LNode));

L=newbase;

leng=leng+inleng;

}

}

use[(L+t)->i][(L+t)->j]=1;

}

t=0;

while(p!=0)

{

cont[t]=p;

t++;

p=(L+p)->cont;

}

printf("(0, 0)\n");

while(t--)

{

printf("(%d, %d)\n",(L+cont[t])->i-1,(L+cont[t])->j-1);

}

}

int main()

{

int num[7][7];

int i,j;

for(i=0,j=0;j<7;j++)

num[i][j]=1;

for(i=6,j=0;j<7;j++)

num[i][j]=1;

for(i=1,j=0;i<6;i++)

num[i][j]=1;

for(i=1,j=6;i<6;i++)

num[i][j]=1;

for(i=1;i<6;i++)

for(j=1;j<6;j++)

scanf("%d",&num[i][j]);

MazePath(num);

return 0;

}

### 最短路径

#include<stdio.h>

void chushihua(int G[][100],int n) //初始化函数，将图的矩阵进行初始化

{

int i,j;

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

for(j=0;j<n;j++)

G[i][j]=999;

}

char gouzao(int G[][100],int m)//构造图的矩阵

{

int i,j;

char ch;

printf("请输入%d条弧的弧头 弧尾，权值，例如:V0 V2 30:\n",m);

while(m--)

{

getchar();

scanf("%c%d%\*c",&ch,&i);

scanf("%c%d%\*c",&ch,&j);

scanf("%d",&G[i][j]);

}

return ch;

}

void shuchutu(int G[][100],int n,char ch)//将图矩阵输出

{

int i,j;

printf(" ");

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

printf("%3c%d",ch,i);

printf("\n");

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

{

for(j=0;j<n;j++)

{

if(j==0)

printf("%c%d",ch,i);

printf("%4d",G[i][j]);

}

printf("\n");

}

}

int shortest(int G[][100],int n,int a[],int b[])

{

int i,j,k,t,min;

b[0]=0;

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

{

a[i]=G[0][i];

}

for(i=0,k=0;i<n;i++)

{

if(a[i]!=999)

{

k=1;

min=a[i];

}

}

if(k==0)

{

printf("源点与所有点都不连通!\n");

return 0;

}

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

{

if(min>a[i])

{

min=a[i];

b[1]=i;

}

}

for(k=1;k<n;k++)

{

min=G[b[k]][0];

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

{

if(min>G[b[k]][i])

{

min=G[b[k]][i];

b[k+1]=i;

}

}

if(min==999)

{

for(j=0;j<n;j++)

{

for(i=0,t=0;i<=k;i++)

if(j==b[i])

t=1;

if(t==0)

{

b[k+1]=j;

}

}

}

if(min+a[b[k]]<a[b[k+1]])

a[b[k+1]]=min+a[b[k]];

min=999;

for(j=0;j<n;j++)

{

for(i=0,t=0;i<=k;i++)

if(j==b[i])

t=1;

if(t==0)

{

if(min>a[j])

{

min=a[j];

b[k+1]=j;

}

}

}

}

return 1;

}

void shuchu(int n,int a[],int b[],char ch)

{

int i;

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

{

if(a[b[i]]!=999)

{

printf("%c0到%c%d的最短路径为:%d\n",ch,ch,b[i],a[b[i]]);

}

else

printf("%c0与%c%d不连通!\n",ch,ch,b[i]);

}

}

void main()

{

int n,m;

char ch;

int G[100][100];

int a[100];

int b[100];

printf("请输入顶点数n和弧数m:");

scanf("%d%d",&n,&m);

chushihua(G,n);

ch=gouzao(G,m);

shuchutu(G,n,ch);

shortest(G,n,a,b);

shuchu(n,a,b,ch);

}

//以下是:ZOJ----1372题的代码//

#include<iostream>

using namespace std;

int main()

{

int p,r,i,rode,count,j,route[51][51],record[51],min,d,stack[51],top;

int a,b,c;

while(cin>>p&&p)

{

cin>>r;

for(i=1;i<=p;i++)

for(j=1;j<=p&&j!=i;j++)

{

route[i][j]=1000;

route[j][i]=1000;

}

memset(record,0,sizeof(record));

for(i=1;i<=r;i++)

{

cin>>a>>b>>c;

if(route[a][b]>c||route[b][a]>c)

{

route[a][b]=c;

route[b][a]=c;

}

}

/\*以下是求最段路径的prim算法:(将那些加入树上的点看成静态,每次对于那些不在树上的点,通过与树上的点比较得到最短的路径,在由所有类似这样未为树上的点中在找出一条最最短的路径,并将其对应的点加入数中来.\*/

top=1;

stack[top++]=1;

record[1]=1;

count=0;

rode=0;

while(count!=p-1)//循还p-1次

{

count++;

c=0;

for(i=1;i<=p;i++)

{

if(!record[i])

{

c++;

a=1001;

for(j=1;j<top;j++)

{

if(a>route[i][stack[j]])

a=route[i][stack[j]];

}

if(c==1)

{

min=a;

d=i;

}

else if(c>1)

{

if(a<min)

{

min=a;

d=i;

}

}

}

}

record[d]=1;

stack[top++]=d;

rode+=min;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

}

cout<<rode<<endl;

}

return 0;

}

//////////////////////////////////////////////////////////////////

以下算法为更优化的代码

http://cs.tju.edu.cn/acm/show\_problem.php?pid=1411

[CODE]:

#include<iostream>

#include<cmath>

#include<iomanip>

using namespace std;

#define MAX 501

struct node

{

double x,y;

}outpost[MAX];

double outdis[MAX][MAX];

int cmp(const void\*a,const void\*b)

{

double c=\*(double\*)a-\*(double\*)b;

if(c>0)return 1;

else if(c==0)return 0;

return -1;

}

void mintree(int n,int s)

{

int record[MAX],i,j,sign,sign1;

double min,mindis[MAX],D;

memset(record,0,sizeof(record));

record[0]=1;

min=1000000000;

for(i=1;i<n;i++)

{

if(min>outdis[0][i])

{

min=outdis[0][i];

sign=i;

}

mindis[i]=outdis[0][i];

}

record[sign]=1;

j=1;

mindis[0]=0;

while(j<n)

{

min=1000000000;

for(i=1;i<n;i++)

{

if(!record[i])

{

if(mindis[i]>outdis[sign][i])

{

mindis[i]=outdis[sign][i];

}

if(min>mindis[i])

{

min=mindis[i];

sign1=i;

}

}

}

record[sign1]=1;

sign=sign1;

j++;

}

qsort(mindis,n,sizeof(mindis[0]),cmp);

j=n-1;

s--;

while(s>0)

{

j--;

s--;

}

D=mindis[j];

cout<<fixed<<showpoint<<setprecision(2)<<D<<endl;

}

int main()

{

int m,i,j,n,s;

double num;

cin>>m;

while(m--)

{

cin>>s>>n;

for(i=0;i<n;i++)cin>>outpost[i].x>>outpost[i].y;

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

for(j=i+1;j<n;j++)

{

num=sqrt((outpost[i].x-outpost[j].x)\*(outpost[i].x-outpost[j].x)+

(outpost[i].y-outpost[j].y)\*(outpost[i].y-outpost[j].y));

outdis[i][j]=num;

outdis[j][i]=num;

}

mintree(n,s);

}

return 0;

}

///////////////////////////////////////////////////////////

/\*

min spanning tree.

\*/

int prim(int n,int dis[max][],father[max])

{

int record[max],i,j,weight[max],m,num,w;

memset(record,0,sizeof(record));

record[0]=1;

m=0;

father[0]=0;

for(i=0;i<n;i++)weight[i]=999999;

weight[0]=0;

w=0;

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

{

for(j=0;j<n;j++)

{

if(!record[j])

{

if(weight[j]>dis[m][j])

{

weight=dis[m][j];

father[j]=m;

}

}

}

num=999999;

for(j=0;j<n;j++)

{

if(!record[j])

{

if(num>weight[j])

{

num=weight[j];

m=j;

}

}

}

record[m]=1;

w+=weight[m];

}

return w;

}

## 栈的相关操作

#define daxiao 1000

#define zhenjia 100

#include<stdio.h>

#include<stdlib.h>

typedef struct

{

int \*top;

int \*base;

int sise;

}zhan;

void gouzhao(zhan &s)//构造一个空栈

{

s.base=(int \*)malloc(daxiao \* sizeof(int));//申请一个存储空间

if(!s.base)

printf("wrong!\n");

s.top=s.base;

s.sise=daxiao;

}

void panduan(zhan &s)//判断栈是否为空

{

if(s.top==s.base)

printf("True!\n");

else

printf("False!\n");

}

void shuru(zhan &s)//在栈中输入元素

{

int i,j;

if((s.top-s.base)>daxiao)//判断此时的存储空间是否充足，不足的话增加存储空间

{

s.base=(int \*)malloc((daxiao+zhenjia) \* sizeof(int));

s.top=s.base+daxiao;

}

printf("How many number do you want to input \n");

scanf("%d",&j);

printf("input what number do you want to:\n");

for(i=0;i<j;i++,s.top++)//将输入的元素储存在相应的地址中

{

scanf("%d",s.top);

}

printf("The number int it are:\n");

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

printf("%5d",\*(s.base+i));//输出个栈内的元素

printf("\n");

}

int changdu(zhan &s) //测量栈的长度

{

int i;

i=s.top-s.base;

printf("There are %d numbers in it!\n",i);

return(i);

}

void shuchu(zhan &s)//输出栈顶元素

{

if(s.top==s.base)

printf("ERROR!\n");

else

printf("The head number is %d\n",\*(s.top-1));

}

int charu(int i,zhan &s)//插入一个新的栈顶元素

{

if((s.top-s.base)>daxiao)//判断此时的存储空间是否充足，不足的话增加存储空间

{

s.base=(int \*)malloc((daxiao+zhenjia) \* sizeof(int));

s.top=s.base+daxiao;

}

printf("input what number do you want to:\n");

scanf("%d",s.top); //在栈顶存入元素

s.top++; //栈顶上移

i++;

printf("The head is:\n");

printf("%d\n",\*(s.top-1));

return(i);

}

int shanchu(int i,zhan &s)//删除顶元素并返回其值

{

printf("The delete number is %d\n",\*(s.top-1));

s.top--;

i--;

return(i);

}

void vist(int \*q) //输出函数

{

printf("%5d",\*q);

}

void fangwen(zhan &s)//访问函数，对栈从栈尾元素逐一进行访问，并调用Vist函数输出

{

int \*p=s.base;

printf("Now it is starting visting:\n");

for(;p!=s.top;p++)

vist(p);

printf("\n");

}

void zhikong(zhan &s)//将栈置空

{

s.top=s.base;

}

void xiaohui(zhan &s) //将栈销毁

{

free(s.base);

s.base=s.top=NULL;

s.sise=0;

}

void main()

{

int k;

zhan s;

gouzhao(s);

shuru(s);

k=changdu(s);

shuchu(s);

k=charu(k,s);

k=shanchu(k,s);

fangwen(s);

zhikong(s);

xiaohui(s);

}

## 一元多项式计算

/\*

3

2 3

3 4

5 7

4

3 3

-3 4

4 6

5 7

\*/

#include<stdio.h>

#include<stdlib.h>

#define ERROR 0

#define OK 1

typedef struct polynode{

float coef; //系数

int expn; //指数

struct polynode \*next;

}polynode,\*polylist;

int InitList(polylist &L) //构造一个空的头结点

{

L=(polylist)malloc(sizeof(polynode));

if(!L)

return ERROR;

L->next=NULL;

return OK;

}

int comper(polylist P,polylist Q)

{

return (P->expn-Q->expn);

}

void create(polylist L,int n) //创建多项式

{

int i;

polylist P,Q;

Q=L;

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

{

P=(polylist)malloc(sizeof(polynode));

printf("请输入本项的系数和指数:");

scanf("%f%d",&P->coef,&P->expn);

Q->next=P;

Q=P;

}

Q->next=NULL;

}

void display(polylist L) //显示链表内容

{

polylist P;

P=L->next;

if(P==NULL)

{

printf("这是一张空表!");

}

while(P)

{

if(P!=L->next)

{

if(P->coef>0)

printf("+");

}

printf("%.2fx(%d)",P->coef,P->expn);

P=P->next;

}

printf("\n");

}

void InsLast(polylist &R,float e1,int e2)//尾插入法

{

polylist newnode;

newnode=(polylist)malloc(sizeof(polynode));

newnode->coef=e1;

newnode->expn=e2;

R->next=newnode;

R=newnode;

}

void add(polylist La,polylist Lb,polylist &Lc)//多项式加法

{

polylist P,Q,R;

P=La->next;

Q=Lb->next;

R=Lc;

int t;

while(P&&Q)

{

t=comper(P,Q);

if(t<0)

{

InsLast(R,P->coef,P->expn);

P=P->next;

}

if(t==0)

{

if((P->coef+Q->coef)!=0)

InsLast(R,P->coef+Q->coef,P->expn);

P=P->next;

Q=Q->next;

}

if(t>0)

{

InsLast(R,Q->coef,Q->expn);

Q=Q->next;

}

}

while(P)

{

InsLast(R,P->coef,P->expn);

P=P->next;

}

while(Q)

{

InsLast(R,Q->coef,Q->expn);

Q=Q->next;

}

R->next=NULL;

}

void subtract(polylist La,polylist Lb,polylist &Ld) //多项式减法

{

polylist P,Q,R;

P=La->next;

Q=Lb->next;

R=Ld;

int t;

while(P&&Q)

{

t=comper(P,Q);

if(t<0)

{

InsLast(R,P->coef,P->expn);

P=P->next;

}

if(t==0)

{

if((P->coef-Q->coef)!=0)

InsLast(R,P->coef-Q->coef,P->expn);

P=P->next;

Q=Q->next;

}

if(t>0)

{

InsLast(R,-Q->coef,Q->expn);

Q=Q->next;

}

}

while(P)

{

InsLast(R,P->coef,P->expn);

P=P->next;

}

while(Q)

{

InsLast(R,-Q->coef,Q->expn);

Q=Q->next;

}

R->next=NULL;

}

void addother(polylist L,polylist &Le)//乘法两个多项式合并

{

polylist P,Q,R,temp;

P=L->next;

Q=Le->next;

R=Le;

int t;

while(P&&Q)

{

t=comper(P,Q);

if(t<0)

{

R->next=P;

R=R->next;

P=P->next;

}

if(t==0)

{

temp=P;

P=P->next;

free(temp);

temp=Q;

Q=Q->next;

free(temp);

}

if(t>0)

{

R->next=Q;

R=R->next;

Q=Q->next;

}

}

if(P)

R->next=P;

}

void Multi(polylist La,polylist Lb,polylist L,polylist &Le)//乘法计算

{

polylist P,Q,R;

P=La->next;

while(P)

{

R=L;

Q=Lb->next;

while(Q)

{

InsLast(R,P->coef\*Q->coef,P->expn+Q->expn);

Q=Q->next;

}

R->next=NULL;

addother(L,Le);

L->next=NULL;

P=P->next;

}

}

void main()

{

polylist La,Lb,Lc,Ld,L,Le;

int n;

printf("请输入多项式La的项数:");

scanf("%d",&n);

InitList(La);

create(La,n);

printf("多项式La:");

display(La);

printf("请输入多项式Lb的项数:");

scanf("%d",&n);

InitList(Lb);

create(Lb,n);

printf("多项式Lb:");

display(Lb);

InitList(Lc);

printf("进行加法计算\n");

add(La,Lb,Lc);

printf("加法计算结果Lc:");

display(Lc);

InitList(Ld);

printf("进行减法计算\n");

subtract(La,Lb,Ld);

printf("减法计算结果Ld:");

display(Ld);

printf("进行乘法计算\n");

InitList(L);

InitList(Le);

Multi(La,Lb,L,Le);

display(Le);

printf("程序运行结束!\n");

}

## 单链表的简单操作

#include<stdio.h>

#include<stdlib.h>

#define LIST\_INIT\_SIZE 100 //线性表存储空间的初始分配量

#define LIST\_INCREMENT 10 //线性表存储空间的分配增量

#define OK 1

#define ERROR 0

typedef int Status;

struct SqList

{

int \*elem; //储存空间的基地址

int length; //当前长度

int listsize; //当前存储容量

};

Status compar(int a,int b)

{

if(a==b)

return OK;

else

return ERROR;

}

int vist(int \*p)

{

printf("%d ",\*p);

return 0;

}

Status InitList(SqList &L) //构造一个空的线性表

{

L.elem=(int \*)malloc(LIST\_INIT\_SIZE\*(sizeof(int)));

if(!L.elem)

return ERROR; //地址分配失败

L.length=0; //初始化长度

L.listsize=LIST\_INIT\_SIZE; //初始化存储容量

return OK;

}

void DestroyList(SqList &L) //销毁线性表L

{

free(L.elem);

L.elem=NULL;

L.length=0;

L.listsize=0;

}

void ClearList(SqList &L) //将L重置为空表

{

L.length=0;

}

Status ListEmpty(SqList L) //判断L表是否为空表

{

if(L.length==0)

return OK;

return ERROR;

}

int ListLength(SqList L) //返回L用数据元素的个数

{

return L.length;

}

Status GetElem(SqList L,int i,int &e) //用e返回L中第i个值

{

if(i<1||i>L.length)

return ERROR;

e=\*(L.elem+i-1);

return OK;

}

Status LocateElem(SqList L,int e,Status (\*compar)(int,int))//返回L中第一个与e满足compar()关系的数据，若无返回0

{

int \*p;

p=L.elem;

int i=1;

while(i<=L.length&&!compar(\*p++,e))

{

i++;

}

if(i>L.length)

return ERROR;

else

return i;

}

Status PriorElem(SqList L,int cur\_e,int &per\_e) //若L中存在cur\_e且不是第一个，则直接用per\_e返回它的前驱

{

int \*p=L.elem;

int i=1;

while(\*p!=cur\_e&&i<=L.length)

{

p++;

i++;

}

if(i==1)

return ERROR;

per\_e=\*(--p);

return OK;

}

Status NextElem(SqList L,int cur\_e,int &next\_e) //若L中存在cur\_e且不是最后一个，则直接用next\_e返回它的后继

{

int \*p=L.elem;

int i=1;

while(\*p!=cur\_e&&i<=L.length)

{

p++;

i++;

}

if(i==L.length)

return ERROR;

next\_e=\*(++p);

return OK;

}

Status ListInsert(SqList &L,int i,int e) //在L中第i个位置处插入数据e

{

int \*p,\*q;

if(i>L.length+1||i<1)

return ERROR;

if(L.length==L.listsize)

{

int \*newbase;

if(!(newbase=(int \*)realloc(L.elem,(L.listsize+LIST\_INCREMENT)\*sizeof(int))))

return ERROR;

L.elem=newbase;

L.listsize=L.listsize+LIST\_INCREMENT;

}

p=L.elem+i-1;

q=L.elem+L.length-1;

for(;q>=p;q--)

{

\*(q+1)=\*q;

}

\*p=e;

L.length++;

return OK;

}

Status ListDelete(SqList &L,int i,int &e) //删除L中的第i个位置的数据，并用e返回

{

int \*p,\*q;

if(i<1||i>L.length)

return ERROR;

p=L.elem+i-1;

e=\*p;

q=L.elem+L.length-1;

for(;p<q;p++)

\*p=\*(p+1);

L.length--;

return OK;

}

Status ListTraveres(SqList L,int (\*vist)(int \*)) //一次对数据中的每个元素调用vist()函数

{

int \*p;

p=L.elem;

int i;

for(i=1;i<=L.length;i++)

vist(p++);

return OK;

}

void main()

{

SqList L;

int e;

Status i;

InitList(L); //测试初始化函数

printf("初始化后：L.elem=%u L.length=%d L.listsize=%d\n",L.elem,L.length,L.listsize);

for(i=1;i<=5;i++)

ListInsert(L,1,i); //测试插入函数

printf("在表头插入1到5后:L=");

ListTraveres(L,\*vist); //调用访问函数

printf("\n");

printf("第一次插入数值后：L.elem=%u L.length=%d L.listsize=%d\n",L.elem,L.length,L.listsize);

i=ListEmpty(L); //测试判断顺序表是否为空的函数

if(i==1)

printf("L为空!\n");

else

printf("L不为空!\n");

/\*\*\*看看清空后表L是否为空\*\*\*\*\*\*\*\*/

ClearList(L); //调用清空函数

printf("清空后：L.elem=%u L.length=%d L.listsize=%d\n",L.elem,L.length,L.listsize);

i=ListEmpty(L); //测试判断顺序表是否为空的函数

if(i==1)

printf("L为空!\n");

else

printf("L不为空!\n");

/\*\*\*重新调用插入函数，给表L赋值\*/

for(i=1;i<=10;i++)

ListInsert(L,1,i); //测试插入函数

printf("在表头插入1到10后:L=");

ListTraveres(L,\*vist); //调用访问函数

printf("\n");

/\*\*\*\*插入操作\*\*\*\*\*\*\*/

printf("第二次插入数值后：L.elem=%u L.length=%d L.listsize=%d\n",L.elem,L.length,L.listsize);

printf("下面我们要调用输入函数，请输入你要插入的位置i:");

scanf("%d",&i);

printf("请输入你要插入的数值e:");

scanf("%d",&e);

ListInsert(L,i,e); //测试插入函数

printf("在表i处插入e后:L=");

ListTraveres(L,\*vist); //调用访问函数

printf("\n");

printf("L.elem=%u L.length=%d L.listsize=%d\n",L.elem,L.length,L.listsize);

/\*\*\*\*\*调用GetElem函数\*\*\*\*\*\*\*/

printf("下面我们要调用得到元素函数，请输入你要得到的元素序数i:");

scanf("%d",&i);

GetElem(L,i,e);

printf("第%d个元素为: %d\n",i,e);

/\*\*\*\*\*调用这个啥啥叫不出名字的函数\*\*\*\*\*/

printf("输入你想得到的位置的元素e:");

scanf("%d",&e);

i=LocateElem(L,e,\*compar);

if(i==0)

printf("L中没有这个元素!\n");

else

printf("它在L中的位置为%d个与元素序位!\n",i);

/\*\*\*\*\*\*\*\*\*\*\*\*\*测试前驱\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("输入你需要的查找前驱的数据e:");

scanf("%d",&e);

if(PriorElem(L,e,i))

printf("e的直接前驱为:%d\n",i);

else

printf("e没有直接前驱!\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*测试后驱函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("输入你需要的查找后继的数据e:");

scanf("%d",&e);

if(NextElem(L,e,i))

printf("e的直接后继为:%d\n",i);

else

printf("e没有直接后继!\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*求表长\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

i=ListLength(L);

printf("表长为:%d\n",i);

/\*\*\*\*\*\*\*\*\*\*\*\*调用删除函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("请输入你要删除的数值的序位:");

scanf("%d",&i);

ListDelete(L,i,e);

printf("删除的元素为:%d\n",e);

printf("删除后的表L为:\n");

ListTraveres(L,\*vist);

printf("\n");

printf("元素删除后:L.elem=%u L.length=%d L.listsize=%d\n",L.elem,L.length,L.listsize);

/\*\*\*\*\*\*\*\*\*\*\*调用销毁函数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DestroyList(L);

printf("表销毁后:L.elem=%u L.length=%d L.listsize=%d\n",L.elem,L.length,L.listsize);

}

## 快速排序

#include<stdio.h>

#define MAXSIZE 100

typedef struct

{

int r[MAXSIZE+1];

int length;

}SqList;

int Partition(SqList &L,int low,int high)

{

L.r[0]=L.r[low];

while(low<high)

{

while(low<high&&L.r[high]>L.r[0])

high--;

L.r[low]=L.r[high];

while(low<high&&L.r[low]<L.r[0])

low++;

L.r[high]=L.r[low];

}

L.r[low]=L.r[0];

return low;

}

void QSort(SqList &L,int low,int high)

{

int k;

if(low<high)

{

k=Partition(L,low,high);

QSort(L,low,k-1);

QSort(L,k+1,high);

}

}

int main()

{

int i,n;

SqList L;

printf("请输入需要排序的数据个数n:");

scanf("%d",&n);

L.length=n;

printf("请输入需要排序的%d个数据:",n);

for(i=1;i<=n;i++)

scanf("%d",&L.r[i]);

QSort(L,1,L.length);

for(i=1;i<=n;i++)

printf("%4d",L.r[i]);

printf("\n");

return 0;

}

## 哈夫曼编码

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct

{

char ch;

int weight;

int parent,lchild,rchild;

}HTNode,\*HuffmanTree;

typedef char \*\*HuffmanCode;

void Select(HuffmanTree HT,int n,int &s1,int &s2) //选择两个权重最小的无父亲结点，且较小的权的下标对s1

{

int i;

for(i=1;i<=n;i++)

{

if(HT[i].parent==0)

{

s1=i;

break;

}

}

for(i=s1+1;i<=n;i++)

{

if((HT[i].parent==0)&&(HT[i].weight<HT[s1].weight))

s1=i;

}

for(i=1;i<=n;i++)

{

if(i!=s1&&HT[i].parent==0)

{

s2=i;

break;

}

}

for(i=s2+1;i<=n;i++)

{

if(i!=s1)

{

if((HT[i].parent==0)&&(HT[i].weight<HT[s2].weight))

s2=i;

}

}

}

void CreateHuffmanTree(HuffmanTree &HT,int w[],char ch[],int n)//构建哈夫曼树

{

int i,m,s1,s2;

m=2\*n-1;

HT=(HuffmanTree)malloc((m+1)\*sizeof(HTNode));

for(i=1;i<=n;i++)

{

HT[i].ch=ch[i-1];

HT[i].weight=w[i-1];

HT[i].parent=0;

HT[i].rchild=0;

HT[i].lchild=0;

}

for(i=n+1;i<=m;i++)

{

HT[i].ch='\n';

HT[i].weight=0;

HT[i].parent=0;

HT[i].rchild=0;

HT[i].lchild=0;

}

for(i=n+1;i<=m;i++)

{

Select(HT,i-1,s1,s2);

HT[s1].parent=i;

HT[s2].parent=i;

HT[i].lchild=s1;

HT[i].rchild=s2;

HT[i].weight=HT[s1].weight+HT[s2].weight;

}

}

void HTCodeing(HuffmanTree HT,HuffmanCode &HC,int n)

{

int i,start;

int f;

int c;

char \*cd;

HC=(HuffmanCode)malloc((n)\*sizeof(char \*));

cd=(char\*)malloc(n\*sizeof(char));

cd[n-1]='\0';

for(i=1;i<=n;i++)

{

start=n-1;

for(c=i,f=HT[i].parent;f!=0;c=f,f=HT[f].parent)

{

if(HT[f].lchild==c)

cd[--start]='0';

else

cd[--start]='1';

}

HC[i-1]=(char \*)malloc((n-start)\*sizeof(char));

strcpy(HC[i-1],&cd[start]);

}

free(cd);

}

void PrintfCode(HuffmanCode HC,int n,char ch[])

{

int i;

printf("编码为:\n");

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

printf("%c %s\n",ch[i],HC[i]);

}

double AverageLength(HuffmanCode HC,HuffmanTree HT,int n)

{

int i,sum=0,len;

double length=0.0;

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

{

len=strlen(HC[i]);

length+=len\*HT[i+1].weight;

sum+=HT[i+1].weight;

}

length=length/sum;

return length;

}

void DeCode(HuffmanTree HT,int n)

{

int i;

char ch;

i=2\*n-1;

while(scanf("%c",&ch)&&ch!='#')

{

if(ch=='0')

i=HT[i].lchild;

else

i=HT[i].rchild;

if(HT[i].lchild==0)

{

printf("%c",HT[i].ch);

i=2\*n-1;

}

}

if((HT[i].lchild!=0)&&(i!=2\*n-1))

printf("\n未能完全解码\n");

printf("\n");

}

void main()

{

int n,i;

char arrch[20];

int arrweigth[20];

double avlength;

HuffmanTree HT;

HuffmanCode HC;

printf("请输入的需要的字母数:");

scanf("%d",&n);

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

{

getchar();

printf("输入字母和权重:");

scanf("%c%d",&arrch[i],&arrweigth[i]);

}

getchar();

CreateHuffmanTree(HT,arrweigth,arrch,n);

HTCodeing(HT,HC,n);

PrintfCode(HC,n,arrch);

avlength=AverageLength(HC,HT,n);

printf("平均编码长度为：%.2lf\n",avlength);

printf("请输入需要解码的数据:");

DeCode(HT,n);

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

free(HC[i]);

free(HC);

free(HT);

}

## 队列的简单操作

#include<stdio.h>

#include<stdlib.h>

typedef struct QNode

{

int data;

struct QNode \*next;

}\*QueuePtr;

typedef struct

{

QueuePtr front; //队头指针

QueuePtr rear;//队尾指针

}LinkQueue;

void InitQueue(LinkQueue &Q)//构造一个空队列

{

Q.front=Q.rear=(QueuePtr)malloc(sizeof(QNode));

if(!Q.front)

printf("Wrong!\n");

Q.front->next=NULL;

}

void shuru(LinkQueue &Q)//在队列空间中存入元素

{

int i,n,a;

QNode \*p,\*r;

printf("please input how many number do you want to input :\n");

scanf("%d",&n);

printf("please input the number you want to input:\n");

r=Q.front;

for(i=1;i<=n;i++)

{

scanf("%d",&a);

p=(QueuePtr)malloc(sizeof(QNode));

p->data=a;

r->next=p;

r=p;

Q.rear->next=p;

Q.rear=p;

}

p->next=NULL;

r=Q.front;

for(i=1;i<=n;i++)

{

p=r->next;

printf("%d->",p->data);

r=p;

}

printf("NULL\n");

}

void DesrotyQueue(LinkQueue &Q)//销毁队列

{

Q.rear=Q.front->next;

free(Q.front);

Q.front=Q.rear;

}

void ClearQueue(LinkQueue &Q)//清空队列

{

QNode \*p,\*r;

Q.rear=Q.front;

r=Q.front;

p=r->next;

for(;r!=NULL;)

{

r=p->next;

free(p);

p=r;

}

}

void QueueEmpty(LinkQueue &Q)//判断队列是否为空，为空则输出True

{

if(Q.front->next==NULL)

printf("True!\n");

else

printf("False!\n");

}

void QueueLength(LinkQueue &Q)//输出队列的长度

{

int i;

QNode \*r;

r=Q.front;

for(i=0;r->next!=NULL;)

i++;

printf("There are %d numbers!\n",i);

}

void GetHead(LinkQueue &Q)//返回队列的头元素

{

QNode \*p;

p=Q.front->next;

printf("The head is %d\n",p->data);

}

void EnQueue(LinkQueue &Q)//插入新的队列尾元素

{

QNode \*p;

int a;

p=(QueuePtr)malloc(sizeof(QNode));

printf("please input the number :\n");

scanf("%d",&a);

p->data=a;

Q.rear->next=p;

Q.rear=p;

p->next=NULL;

printf("The last number is %d now\n",a);

}

void DeQueue(LinkQueue &Q)//删除队列的头元素

{

QNode \*r;

r=Q.front->next;

Q.front->next=r->next;

printf("%d is delete\n",r->data);

free(r);

}

void vist(QNode \*p)//定义VIsit函数

{

printf("%4d",\*p);

}

void QueueTraverse(LinkQueue &Q)//从队头到队尾对么个函数调用函数Vist

{

QNode \*q;

q=Q.front;

for(;q->next!=NULL;)

{

q=q->next;

vist(q);

}

printf("\n");

}

void main()

{

LinkQueue Q;

InitQueue(Q);

QueueEmpty(Q);

shuru(Q);

GetHead(Q);

EnQueue(Q);

DeQueue(Q);

QueueTraverse(Q);

QueueLength(Q);

ClearQueue(Q);

DesrotyQueue(Q);

}

## KMP模式匹配算法

#include<stdio.h>

#include<string.h>

#define MAX 255

typedef char SString[MAX+1];

SString T,S;

int next[MAX],pos;

void get\_nex(SString T,int next[])

{

int i,j;

i=1;

j=0;

next[1]=0;

while(i<T[0])

{

if(j==0||T[i]==T[j])

{

i++;

j++;

next[i]=j;

}

else

j=next[j];

}

}

int Index\_KMP(SString S,SString T,int pos,int next[])

{

int i=1;

int j=1;

while(i<=S[0]&&j<=T[0])

{

if(j==0||S[i]==T[j])

{

i++;

j++;

}

else

j=next[j];

}

if(j>T[0])

{

pos=i-T[0];

return pos;

}

else

{

pos=0;

return pos;

}

}

void main()

{

int i,k;

gets(S);

k=strlen(S);

for(i=k;i>0;i--)

{

S[i]=S[i-1];

}

S[0]=k;

gets(T);

k=strlen(T);

for(i=k;i>0;i--)

{

T[i]=T[i-1];

}

T[0]=k;

printf("%d\n",k);

get\_nex(T,next);

pos=Index\_KMP(S,T,pos,next);

for(i=1;i<=T[0];i++)

printf("%d ",next[i]);

printf("\n");

if(pos!=0)

printf("pos=%d\n",pos);

else

printf("Wrong!\n");

}

# 常用的数学公式[4]

## 中国剩余定理

一般形式：设m = m1 ，… ，mk 为两两互素的正整数，m＝m1，…mk ，m＝mi

Mi，i＝1，2，… ，k 。则同余式组x≡b1(modm1)，…，x≡bk(modmk)的解为

x≡M'1M1b1＋…＋M'kMkbk （modm）。式中M'iMi≡1 （modmi），i＝1，2，…，k 。

完全数：又称完美数或完备数，是一些特殊的自然数：

它所有的真因子（即除了自身以外的约数）的和（即因子函数），恰好等于它本身。

偶完全数公式：2^(n-1)\*(2^n-1) ，当且仅当2^n-1为素数，又称梅森素数。

2^n-1为素数的必要条件为n是素数。

## 基本几何公式

圆柱体的体积公式：体积=底面积×高 ，如果用h代表圆柱体的高，则圆柱＝S底×h

长方体的体积公式：体积=长×宽×高

如果用a、b、c分别表示长方体的长、宽、高则

长方体体积公式为：V长=abc

正方体的体积公式：体积＝棱长×棱长×棱长．

如果用a表示正方体的棱长，则

正方体的体积公式为V正＝a·a·a＝a³

锥体的体积=底面面积×高÷3 V 圆锥＝S底×h÷3

台体体积公式:V=[ S上+√(S上S下)+S下]h÷3

圆台体积公式:V=(R²+Rr+r²)hπ÷3

球缺体积公式＝πh²(3R-h)÷3

球体积公式：V＝4πR³/3

棱柱体积公式：V＝S底面×h＝S直截面×l （l为侧棱长,h为高)

棱台体积：V=〔S1＋S2＋开根号（S1\*S2）〕／3\*h

注：V：体积；S1：上表面积；S2：下表面积；h：高。

几何体的表面积计算公式

圆柱体:

表面积:2πRr+2πRh 体积:πRRh (R为圆柱体上下底圆半径,h为圆柱体高) 圆锥体:

表面积:πRR+πR[(hh+RR)的平方根] 体积: πRRh/3 (r为圆锥体低圆半径,h为其高, 平面图形

名称 符号 周长C和面积S

正方形 a—边长 C＝4a S＝a2 长方形 a和b－边长 C＝2(a+b) S＝ab 三角形 a,b,c－三边长h－a边上的高s－周长的一半A,B,C－内角其中

s＝(a+b+c)/2 S＝ah/2＝ab/2·sinC ＝[s(s-a)(s-b)(s-c)]1/2＝a2sinBsinC/(2sinA) 四边形 d,D－对角线长α－对角线夹角 S＝dD/2·sinα 平行四边形 a,b－边长h－a边的高α－两边夹角 S＝ah＝absinα 菱形 a－边长α－夹角D－长对角线长d－短对角线长 S＝Dd/2＝a2sinα 梯形 a和b－上、下底长h－高m－中位线长 S＝(a+b)h/2＝mh 圆 r－半径 d－直径 C＝πd＝2πr S＝πr2＝πd2/4 扇形 r—扇形半径 a—圆心角度数 C＝2r＋2πr×(a/360) S＝πr2×(a/360) 弓形 l－弧长 S＝r2/2·(πα/180-sinα)

b－弦长 ＝r2arccos[(r-h)/r] - (r-h)(2rh-h2)1/2

h－矢高 ＝παr2/360 - b/2·[r2-(b/2)2]1/2

r－半径 ＝r(l-b)/2 + bh/2

α－圆心角的度数 ≈2bh/3 圆环 R－外圆半径 S＝π(R2-r2)

r－内圆半径 ＝π(D2-d2)/4

D－外圆直径

d－内圆直径 椭圆 D－长轴 S＝πDd/4

d－短轴

递推问题：三角形l=l+i\*(i+1)/2+i/2\*(i-i/2)

## 欧拉公式

（1）分式里的欧拉公式

a＾r/(a-b)(a-c)+b＾r/(b-c)(b-a)+c＾r/(c-a)(c-b)

当r=0,1时式子的值为0 当r=2时值为1

当r=3时值为a+b+c

（2）三角形中的欧拉公式

设R为三角形外接圆半径，r为内切圆半径，d为外心到内心的距离，则： d＾2=R＾2-2Rr

（3）拓扑学里的欧拉公式

V+F-E=X(P)，V是多面体P的顶点个数，F是多面体P的面数，E是多面体P的棱的条数,X(P)是多面体P的欧拉示性数。

（4）初等数论里的欧拉公式

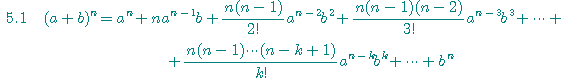
欧拉函数，在数论，对正整数n，欧拉函数是少于或等于n的数中与n互质的数的数目。

此函数以其首名研究者欧拉命名，它又称为Euler's totient function、φ函数、欧拉商数等。

φ函数的值Euler函数通式：φ(x)=x(1-1/p1)(1-1/p2)(1-1/p3)(1-1/p4)…..(1-1/pn),

其中p1, p2……pn为x的所有质因数，x是不为0的整数。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **三角不等式** | | |a+b|≤|a|+|b| | | | | | | |a-b|≤|a|+|b| | | | |a|≤b<=>-b≤a≤b | | |
|  | | | | | |  | | |  | | |
| |a-b|≥|a|-|b| | | | | | | -|a|≤a≤|a| | | |  | | |
| **一元二次方程的解** | | -b+√(b2-4ac)/2a | | | | | | -b-√(b2-4ac)/2a | | |  | |  |
| **根与系数的关系** | | | | X1+X2=-b/a | | | | | X1\*X2=c/a | | | 注：韦达定理 | |
| **数列前n项和** | 1+2+3+4+5+6+7+8+9+…+n=n(n+1)/2 | | | | | 1+3+5+7+9+11+13+15+…+(2n-1)=n2 | | | | | | | |
| 2+4+6+8+10+12+14+…+(2n)=n(n+1) | | | | | 12+22+32+42+52+62+72+82+…+n2=n(n+1)(2n+1)/6 | | | | | | | |
| 13+23+33+43+53+63+…n3=n2(n+1)2/4 | | | | | 1\*2+2\*3+3\*4+4\*5+5\*6+6\*7+…+n(n+1)=n(n+1)(n+2)/3 | | | | | | | |
| **常用数学公式表:解析几何公式** | | | | | | | | | | | | | |
| **圆的标准方程** | | | (x-a)2+(y-b)2=r2 | | | | 注：（a,b）是圆心坐标 | | | | | | |
| **圆的一般方程** | | | x2+y2+Dx+Ey+F=0 | | | | 注：D2+E2-4F>0 | | | | | | |
| **抛物线标准方程** | | | y2=2px | | y2=-2px | | x2=2py | | | x2=-2py | | | |
| **常用数学公式表:几何图形公式** | | | | | | | | | | | | | |
| **直棱柱侧面积** | | | S=c\*h | | | | **斜棱柱侧面积** | | | S=c'\*h | | | |
| **正棱锥侧面积** | | | S=1/2c\*h' | | | | **正棱台侧面积** | | | S=1/2(c+c')h' | | | |
| **圆台侧面积** | | | S=1/2(c+c')l=pi(R+r)l | | | | **球的表面积** | | | S=4pi\*r2 | | | |
| **圆柱侧面积** | | | S=c\*h=2pi\*h | | | | **圆锥侧面积** | | | S=1/2\*c\*l=pi\*r\*l | | | |
| **弧长公式** | | | l=a\*r (a是圆心角的弧度数r>0) | | | | **扇形面积公式** | | | s=1/2\*l\*r | | | |
| **锥体体积公式** | | | V=1/3\*S\*H | | | | **圆锥体体积公式** | | | V=1/3\*pi\*r2h | | | |
| **柱体体积公式** | | | V=s\*h | | | | **圆柱体** | | | V=pi\*r2h | | | |
| **斜棱柱体积** | | | V=S'L (S'是直截面面积，L是侧棱长) | | | | 注：pi=acos(-1.0); | | | | | | |



## 划分问题：

1. n个点最多把直线分成C(n,0)+C(n,1)份；

2、n条直线最多把平面分成C(n,0)+C(n,1)+C(n,2)份；   
3、n个平面最多把空间分成C(n,0)+C(n,1)+C(n,2)+C(n,3)=(n³+5n+6)/6份；   
4、n个空间最多把“时空”分成C(n,0)+C(n,1)+C(n,2)+C(n,3)+C(n,4)份.

## Stirling公式

lim(n→∞) √(2πn) \* (n/e)^n  = n!

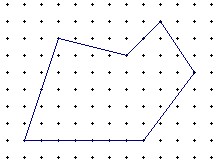
也就是说当n很大的时候,n!与√(2πn) \* (n/e) ^ n的值十分接近，这就是Stirling公式.(2πn) ^0.5× n^ n × e^(-n) =n!;

n! 约等于 sqrt(2\*pi\*n)\*(n/e)^n

## 皮克定理

一个多边形的顶点如果全是格点，这多边形就叫做格点多边形。有趣的是，这种格点多边形的面积计算起来很方便，只要数一下图形边线上的点的数目及图内的点的数目，就可用公式算出。

　　这个公式是皮克(Pick)在1899年给出的，被称为“皮克定理”，这是一个实用而有趣的定理。



给定顶点坐标均是整点（或正方形格点）的简单多边形，皮克定理说明了其面积S和内部格点数目a、边上格点数目b的关系：

S=a+ b/2 - 1。

(其中a表示多边形内部的点数,b表示多边形边界上的点数,S表示多边形的面积)

## 卡特兰数

### 原理:

令h(1)＝1,h(0)=1，catalan

数满足递归式：

　　h(n)= h(0)\*h(n-1)+h(1)\*h(n-2) + ... + h(n-1)h(0) (其中n>=2)

另类递归式：

h(n) = h(n-1)\*(4\*n-2)/(n+1);

　　该递推关系的解为：

　　h(n)=C(2n,n)/(n+1) (n=1,2,3,...)

　　卡特兰数的应用

（实质上都是递归等式的应用）

## 错排公式

当n个编号元素放在n个编号位置,元素编号与位置编号各不对应的方法数用M(n)表示,那么M(n-1)就表示n-1个编号元素放在n-1个编号位置,各不对应的方法数,其它类推.

第一步,把第n个元素放在一个位置,比如位置k,一共有n-1种方法;

第二步,放编号为k的元素,这时有两种情况.1,把它放到位置n,那么,对于剩下的n-2个元素,就有M(n-2)种方法;2,不把它放到位置n,这时,对于这n-1个元素,有M(n-1)种方法;

综上得到递推公式：

M(n)=(n-1)[M(n-2)+M(n-1)] 特殊地，M(1)=0,M(2)=1

通项公式：

M(n)=n![(-1)^2/2!+…+(-1)^(n-1)/(n-1)!+(-1)^n/n!]

优美的式子：

Dn=[n!/e+0.5],[x]为取整函数．

公式证明较简单．观察一般书上的公式，可以发现e-1的前项与之相同，然后作比较可得/Dn-n!e-1/<1/(n+1)<0.5,于是就得到这个简单而优美的公式（此仅供参考）

## 等比数列

(1) [等比数列](http://baike.baidu.com/view/62282.htm)：a (n+1)/an=q (n∈N)。

(2) 通项公式：an=a1×q^(n-1)；

推广式：an=am×q^(n-m)；

(3) 求和公式：Sn=n\*a1 (q=1)

Sn=a1(1-q^n)/(1-q) =(a1-an\*q)/(1-q) (q≠1) (q为比值，n为项数）

(4)性质：

①若 m、n、p、q∈N，且m＋n=p＋q，则am\*an=ap\*aq；

②在等比数列中，依次每 k项之和仍成等比数列.

③若m、n、q∈N，且m+n=2q，则am\*an=aq^2

(5)"G是a、b的等比中项""G^2=ab（G ≠ 0）".

(6)在等比数列中，首项a1与公比q都不为零.

注意：上述公式中an表示等比数列的第n项。

## 等差数列

1. Sn=n(a1+an)/2
2. Sn=a1\*n+n(n-1)d/2   
    an=a1+(n-1)d

Sn＝（a1＋an）\*n/2

项数＝（末项-首项）÷公差＋1

A1=2\*S/n-an

an=2\*S/n-a1

an=a1+（n-1）\*d;

性质：

若 m、n、p、q∈N

①若m＋n=p＋q，则am+an=ap+aq

②若m+n=2q，则am+an=2aq

注意：上述公式中an表示等差数列的第n项。

## 二次函数

**定义与定义表达式**

1：一般式：

y=ax^2;+bx+c (a≠0，a、b、c为常数)

对称轴为直线*x = -b/2a，*顶点坐标(-b/2a,(4ac-b^2)/4a)。

2：[顶点式](http://baike.baidu.com/view/1895614.htm)：

y=a(x-h）^2+k 或 y=a(x+m)^2+k

(两个式子实质一样,但初中课本上都是第一个式子)（若给出抛物线的顶点坐标或对称轴与最值，通常可设顶点式）

**3：**[**交点式**](http://baike.baidu.com/view/2974353.htm)**（与x轴）**：

y=a(x-x1)(x-x2)

（若给出抛物线与x轴的交点及对称轴与x轴的交点距离或其他一的条件，通常可设交点式）

重要概念：

（a，b，c为常数，a≠0，且a决定函数的开口方向，a>0时，开口方向向上，a<0时，开口方向向下。a的绝对值还可以决定开口大小,a的绝对值越大开口就越小,a的绝对值越小开口就越大。）

y＝ax2＋bx＋c＝image001＝image002  
＝image003＝image004

## 二次方程

a\*x+b\*y+c=0;

当△<0 ，方程无解；

当△=0 ，x1=x2= -b/(2\*a)；

当△>0 ，x1= [-b-sqrt(b\*b-4\*a\*c)]/(2\*a)，x2=[-b+ sqrt(b\*b-4\*a\*c)]/(2\*a)。

## 约瑟夫环

令f表示i个人玩游戏报m退出最后胜利者的编号，最后的结果自然是f[n].

递推公式:

　　f[1]=0;

　　f=(f[i-1]+m)%i; (i>1)

　有了这个公式，我们要做的就是从1-n顺序算出f的数值，最后结果是f[n]。因为实际生活中编号总是从1开始，我们输出f[n]+1由于是逐级递推，不需要保存每个f，程序也是异常简单：

## 多边形面积

点顺序给出

S=0.5\*abs(x1\*y2-y1\*x2+x2\*y3-y2\*x3+...+xn\*y1-yn\*x1)

## 均值不等式的简介

概念：

1、调和平均数：

Hn=n/(1/a1+1/a2+...+1/an)

2、几何平均数：

Gn=(a1a2...an)^(1/n)=n次√(a1\*a2\*a3\*...\*an)

3、算术平均数：

An=(a1+a2+...+an)/n

1. 平方平均数：  
   Qn=√ [(a1^2+a2^2+...+an^2)/n]
2. 这四种平均数满足:

Hn≤Gn≤An≤Qn

a1、a2、… 、an∈R +，当且仅当a1=a2= … =an时取“=”号

均值不等式的一般形式：

设函数D(r)=[（a1^r+a2^r+...an^r）/n]^(1/r) (当r不等于0时);

(a1a2...an)^(1/n)(当r=0时）（即D(0)=(a1a2...an)^(1/n)）

则有：当r<s时，D(r)≤D(s)

注意到Hn≤Gn≤An≤Qn仅是上述不等式的特殊情形，即D(-1)≤D(0)≤D(1)≤D(2)

## 均值不等式的变形

　　(1)对实数a,b，有a^2+b^2≥2ab (当且仅当a=b时取“=”号)，a^2+b^2>0>-2ab

　　(2)对非负实数a,b，有a+b≥2√(a\*b)≥0，即(a+b)/2≥√(a\*b)≥0

　　(3)对负实数a,b，有a+b<0<2√(a\*b)

　　(4)对实数a,b，有a(a-b)≥b(a-b)

　　(5)对非负数a,b，有a^2+b^2≥2ab≥0

　　(6)对非负数a,b，有a^2+b^2 ≥1/2\*(a+b)^2≥ab

　　(7)对非负数a,b,c，有a^2+b^2+c^2≥1/3\*(a+b+c)^2

　　(8)对非负数a,b,c，有a^2+b^2+c^2≥ab+bc+ac

　　(9)对非负数a,b，有a^2+ab+b^2≥3/4\*(a+b)^2

(10)对实数a,b,c，有(a+b+c)/3>=(abc)^(1/3)

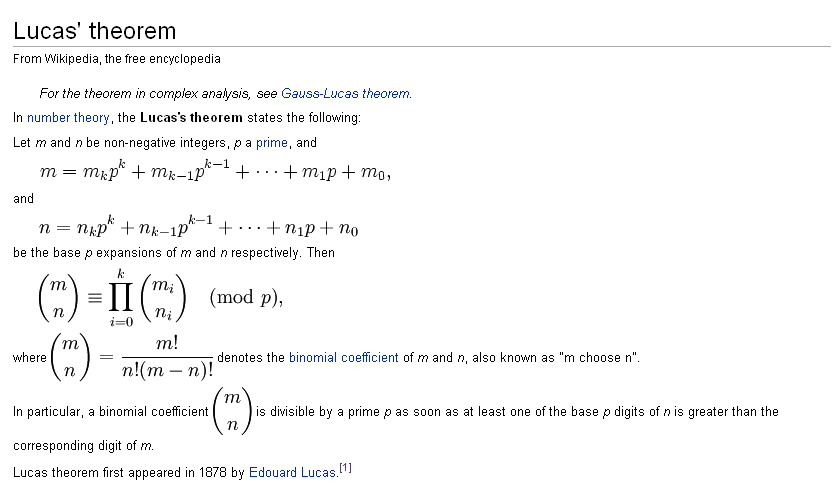
(11) a^3+b^3+c^3>=3abc,a、b、c都是正数。

扩展：若有y=x1\*x2\*x3.....Xn 且x1+x2+x3+...+Xn=常数P,则Y的最大值为((x1+x2+x3+.....+Xn)/n)^n

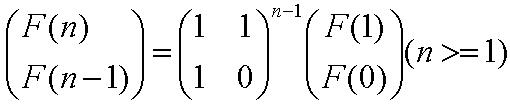
1、| |a|-|b| |≤|a-b|≤|a|+|b| 2、| |a|-|b| |≤|a+b|≤|a|+|b|

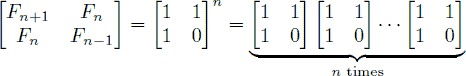
设a1,a2,…an；b1,b2…bn均是实数，且a1≥a2≥a3≥…≥an,b1≥b2≥b3≥…≥bn;则有a1b1+a2b2+…+anbn（顺序和）≥a1b2+a2b1+a3b3+…+aibj+…+anbm（乱序和）≥a1bn+a2bn-1+a3bn-2+…+anb1（逆序和）,仅当a1=a2=a3=…an,b1=b2=b3=…=bn时等号成立。

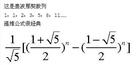
## Lucas 定理



## 斐波那契数列







|  |
| --- |
| 欧拉函数 实际使用的时候是用下面这个式子：  在程序中利用欧拉函数如下性质，可以快速求出欧拉函数的值(a为N的质因素) (1) 若(N%a==0 && (N/a)%a==0) 则有:E(N)=E(N/a)\*a; (2) 若(N%a==0 && (N/a)%a!=0) 则有:E(N)=E(N/a)\*(a-1);  欧拉函数的值  \varphi(1)=1（小于等于1的正整数中唯一和1互质的数就是1本身）。  若*n*是[质数](http://zh.wikipedia.org/zh-cn/%E8%B3%AA%E6%95%B8)*p*的*k*次[幂](http://zh.wikipedia.org/zh-cn/%E5%86%AA)，\varphi(n)=p^k-p^{k-1}=(p-1)p^{k-1}，因为除了*p*的[倍数](http://zh.wikipedia.org/zh-cn/%E5%80%8D%E6%95%B8)外，其他数都跟*n*互质。  欧拉函数是[积性函数](http://zh.wikipedia.org/zh-cn/%E7%A9%8D%E6%80%A7%E5%87%BD%E6%95%B8)，即是说若*m*,*n*互质，\varphi(mn)=\varphi(m)\varphi(n)。证明：设*A*, *B*, *C*是跟*m*, *n*, *mn*互质的数的集，据[中国剩余定理](http://zh.wikipedia.org/zh-cn/%E4%B8%AD%E5%9C%8B%E5%89%A9%E9%A4%98%E5%AE%9A%E7%90%86)，A \times B和*C*可建立[双射](http://zh.wikipedia.org/zh-cn/%E5%8F%8C%E5%B0%84)(一一对应)的关系。因此\varphi(n)的值使用[算术基本定理](http://zh.wikipedia.org/zh-cn/%E7%AE%97%E8%A1%93%E5%9F%BA%E6%9C%AC%E5%AE%9A%E7%90%86)便知，  若n = p_1^{k_1} p_2^{k_2} \cdots p_r^{k_r}  则\varphi(n) = \prod_{p\mid n} p^{\alpha_p-1}(p-1) = n\prod_{p|n}\left(1-\frac{1}{p}\right)。  例如\varphi(72)=\varphi(2^3\times3^2)=2^{3-1}(2-1)\times3^{2-1}(3-1)=2^2\times1\times3\times2=24  先看（2）式，N%a==0 && (N/a)%a!=0，因为a是质数，而N/a不能被a整除，说明N/a与a互质，由上面的\varphi(mn)=\varphi(m)\varphi(n)可知E(N)=E(N/a)\*E(a);而E(a)=a-1 （因为按照欧拉函数定义**欧拉函数\varphi(n)**是小于或等于*n*的正整数中与*n*[互质](http://zh.wikipedia.org/zh-cn/%E4%BA%92%E8%B3%AA)的数的数目，a为质数，从1到a-1都和a互质，所以E(a)=a-1）,故E(N)=E(N/a)\*(a-1);  再看（1）式，由于a是N/a的质因数，不能像（2）一样用上面的积性函数的公式。那看这个式子\varphi(n) = \prod_{p\mid n} p^{\alpha_p-1}(p-1) = n\prod_{p|n}\left(1-\frac{1}{p}\right)，假设E(n)可以表示成a^k\*(a-1)\*C，C为后面的部分，则E(n/a)可以表示成a^(k-1)\*(a-1)\*C，比较两式的差别可以得到E(n)是E(n/a)的a倍，即E(N)=E(N/a)\*a。  其实，（2）式也可以像（1）式一样证明，因为N/a与a互质，说明N的质因数有且只有一个a，则E(n)可表示为(a-1)\*C，E(N/a)可表示为C，比较两式差别可以得到E(N)=E(N/a)\*(a-1). |

## 蚂蚁爬绳

一绳长L米，一蚂蚁从绳的一端爬向另一端，速度为每秒v m/s，同时，绳子以每秒u 米的速度均匀伸长，问：蚂蚁能否达到绳的另一端？如能，需多长时间？如不能，请说明理由。（假设绳子质量无限好，蚂蚁寿命无限长）

T=[e(u/v)-1]\*L/u;

## (a/b)%m

背景：a是b的倍数  
1.如果m是质数，很简单，直接用扩展的欧几里德求b关于m的逆元

对于 a/b%m = ans, 求 ans。  
a = a%m, b = b%m  
ans = (a % m)\*(x % m) % m  （x为b的逆元）  
求逆元则利用扩展欧几里德：  
对于 b\*x = 1(mod m)  
可以求b\*x + m\*y = 1的解（ 用extennd\_Euclid(b, m, x, y) )!  
然后把 x 映射到 [0,m)区间，带入上式， 即得解。

2.如果m不是质数，把m质数分解成质数p1,p2,……,pk的积  
  然后把a分解成a1\*a2，其中a1的质因数只能在p[]中，a2与p[]中的所有质数都互质，即a2与m互质  
  同理把b分解成b1\*b2，其中b1的质因数只能在p[]中，b2与p[]中的所有质数都互质，即b2与m互质  
3.现在问题变成了(a1\*a2)/(b1\*b2)%m，即(a1/b1)%m\*(a2/b2)%m。  
  问题分解成了两个问题:  
  对于a1/b1%m，可以化为:

(p1^m1\*p2^m2\*……\*pk^mk)/(p1^n1\*p2^n2\*……\*pk^nk)%m， 即:p1^(m1-n1)\*p2^(m2-n2)\*……\*pk^(mk-nk)%m  
对于a2/b2%m，b2与m互质，则可以直接求出b2关于m的逆元化为a2\*b2^(-1)%m.  
4.于是，问题解决，时间复杂度约为O(sqrt(m) + log(m))

## 泰勒公式

}B]T}R[RV~$(HGOCFQE}92F

## 乘法与因式分解公式

1.2 formula1

formula2

1.4  formula3

## 三角不等式

2.1 formula4 2.2 formula5 2.3 formula6

2.4 formula7 2.6 formula8

## 某些数列的前n项和

formulaC 4.2 formulaD

4.3 formulaEformulaF

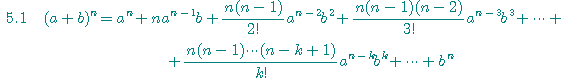
formulaG

formulaH

4.7  formulaI

formulaJ

## 二项式展开公式



## 三角函数公式

1  两角和公式

formulaN

formulaO

2  倍角公式

6.5 formulaP

6.6 formulaQ

formulaR

formulaS

3  半角公式

formulaT

formulaU

formulaV

formulaW

4  和差化积

formulaX

formulaY

formulaZ

formulb0

formulb1

formulb2

formulb3

formulb4

formulb5

formulb6

# 常用系统函数

## 将字符串转化为数字

头文件：#include<stdlib.h>

相关函数：atof，atol，atrtod，strtol，strtoul

int atoi(const char \*nptr);

atoi()与使用strtol(nptr，(char\*\*)NULL，10)；结果相同。

a=strtol(ch,NULL,16);运行结果，将合法的16进制字符串转化为10进制

atoi()会扫描参数nptr字符串，跳过前面的空格字符，直到遇上数字或正负符号才开始做转换，而再遇到非数字或字符串结束时('')才结束转换，并将结果返回。  
long int strtol(const char \*nptr,char \*\*endptr,int base);  
strtol()会将参数nptr字符串根据参数base来转换成长整型数。参数base范围从2至36，或0。参数base代表采用的进制方式，如 base值为10则采用10进制，若base值为16则采用16进制等。当base值为0时则是采用10进制做转换，但遇到如'0x'前置字符则会使用 16进制做转换。一开始strtol()会扫描参数nptr字符串，跳过前面的空格字符，直到遇上数字或正负符号才开始做转换，再遇到非数字或字符串结束时('')结束转换，并将结果返回。若参数endptr不为NULL，则会将遇到不合条件而终止的nptr中的字符指针由endptr返回。

## 系统函数排序

#include <algorithm>  
using namespace std;

void sort(a,a+n); //a，数组首地址，n数组长度，默认从小到大排序

//void sort( iterator start, iterator end, StrictWeakOrdering cmp );

bool compare(int a,int b)

{

return a<b; //升序排列，如果改为return a>b，则为降序

}

void sort(a,a+n, compare);

函数名 功能描述

sort 对给定区间所有元素进行排序

stable\_sort 对给定区间所有元素进行稳定排序

partial\_sort 对给定区间所有元素部分排序

partial\_sort\_copy 对给定区间复制并排序

nth\_element 找出给定区间的某个位置对应的元素

is\_sorted 判断一个区间是否已经排好序

partition 使得符合某个条件的元素放在前面

stable\_partition 相对稳定的使得符合某个条件的元素放在前面

2）更进一步，让这种操作更加能适应变化。也就是说，能给比较函数一个参数，用来指示是按升序还是按降序排,这回轮到函数对象出场了。

为了描述方便，我先定义一个枚举类型EnumComp用来表示升序和降序。很简单：

enum Enumcomp{ASC,DESC};

然后开始用一个类来描述这个函数对象。它会根据它的参数来决定是采用“<”还是“>”。

class compare

{

private:

Enumcomp comp;

public:

compare(Enumcomp c):comp(c) {};

bool operator () (int num1,int num2)

{

switch(comp)

{

case ASC:

return num1<num2;

case DESC:

return num1>num2;

}

}

};

接下来使用 sort(begin,end,compare(ASC)实现升序，sort(begin,end,compare(DESC)实现降序。

主函数为：

int main()

{

int a[20]={2,4,1,23,5,76,0,43,24,65},i;

for(i=0;i<20;i++)

cout<<a[i]<<endl;

sort(a,a+20,compare(DESC));

for(i=0;i<20;i++)

cout<<a[i]<<endl;

return 0;

}

3)其实对于这么简单的任务（类型支持“<”、“>”等比较运算符），完全没必要自己写一个类出来。标准库里已经有现成的了，就在functional里，include进来就行了。functional提供了一堆基于模板的比较函数对象。它们是（看名字就知道意思了）：equal\_to<Type>、not\_equal\_to<Type>、greater<Type>、greater\_equal<Type>、less<Type>、less\_equal<Type>。对于这个问题来说，greater和less就足够了，直接拿过来用：

升序：sort(begin,end,less<data-type>());

降序：sort(begin,end,greater<data-type>()).

int \_tmain(int argc, \_TCHAR\* argv[])

{

int a[20]={2,4,1,23,5,76,0,43,24,65},i;

for(i=0;i<20;i++)

cout<<a[i]<<endl;

sort(a,a+20,greater<int>());

for(i=0;i<20;i++)

cout<<a[i]<<endl;

return 0;

}

4)既然有迭代器，如果是string 就可以使用反向迭代器来完成逆序排列，程序如下：

int main()

{

string str("cvicses");

string s(str.rbegin(),str.rend());

cout << s <<endl;

return 0;

}

qsort():

原型:

\_CRTIMP void \_\_cdecl qsort (void\*, size\_t, size\_t,int (\*)(const void\*, const void\*));

解释: qsort ( 数组名 ，元素个数，元素占用的空间(sizeof)，比较函数)

比较函数是一个自己写的函数 遵循 int com(const void \*a,const void \*b) 的格式。

当a b关系为 > < = 时，分别返回正值 负值 零 （或者相反）。

使用a b 时要强制转换类型，从void \* 转换回应有的类型后，进行操作。

数组下标从零开始,个数为N, 下标0-(n-1)。

实例：

int compare(const void \*a,const void \*b)

{

return \*(int\*)b-\*(int\*)a;

}

int main()

{

int a[20]={2,4,1,23,5,76,0,43,24,65},i;

for(i=0;i<20;i++)

cout<<a[i]<<endl;

qsort((void \*)a,20,sizeof(int),compare);

for(i=0;i<20;i++)

cout<<a[i]<<endl;

return 0;

}

二维数组排序举例：

#include <iostream>

#include <algorithm>

#include <ctime>

using namespace std;

bool cmp(int \*p,int \*q)

{

if(p[0]==q[0])

{

if(p[1]==q[1])

{

return p[2]<q[2];

}

else return p[1]<q[1];

}

else return p[0]<q[0];

}

int main()

{

srand(time(0));

int i;

//int a[1000][3];

int \*\*a=new int\*[1000]; // 这a里¤? 如¨?果?用®? inta[1000][3] 则¨°会¨¢出?错ä¨ª。¡ê

for(i=0;i<1000;++i)

{

a[i]=new int[3];

a[i][0]=rand()%1000;

a[i][1]=rand()%1000;

a[i][2]=rand()%1000;

}

sort(a,a+1000,cmp);

cout<<"After sort"<<endl;

for(i=0;i<1000;++i)

{

printf("%d\t%d\t%d\n",a[i][0],a[i][1],a[i][2]);

}

return 0;

}

结构体形式举例：

#include <stdio.h>

#include <algorithm>

using namespace std;

struct catfood{

int j;

int f;

};

bool cam(catfood a, catfood b){

double s=a.j\*1.0/a.f, s1=b.j\*1.0/b.f;

return s>s1;

}

double b[1003];

catfood s[1003];

int main(){

int n, m;

while(scanf("%d%d", &m, &n)!=EOF)

{

if(m==-1) break;

for(int i=0;i<n;i++)

{

scanf("%d%d", &s[i].j, &s[i].f);

b[i]=0;

}

sort(s, s+n, cam);

double c=m, sum=0;

int i;

for(i=0; i<n; i++){

if(s[i].f>c) break;

b[i]=1;

c-=s[i].f;

sum+=s[i].j;

}

if(i<n) sum+=c\*s[i].j/s[i].f;

printf("%.3lf\n", sum);

}

return 0;

}

Switch(表达式)

{

Case 常量：语句；（break）

Default :

}

Strcpy(数组名，字符串) //复制（包括”\0”）;

Strcat(数组名1，数组名2)//将数组二连接到数组1后面

Strcmp(字符串1，字符串2)//比较两个字符串相等返回0；前大返回正，后大返回负

**主要参考**

1. 蓝燕的高精度算法模板；
2. 浙大牛人所做算法模板，网上下载所得；
3. 谭立栋所提供的材料整理所得
4. 队友网络下载所得；