**ACM模板**

|  |  |  |  |
| --- | --- | --- | --- |
| **数学问题：** |  |  |  |
| [1.精度计算——大数阶乘](file:///E:\ACM\ACM常用代码.mht) | 2.精度计算——乘法（大数乘小数） | [3.精度计算——乘法（大数乘大数）](file:///E:\ACM\ACM常用代码.mht) | [4.精度计算——加法](file:///E:\ACM\ACM常用代码.mht) |
| [5.精度计算——减法](file:///E:\ACM\ACM常用代码.mht) | [6.任意进制转换](file:///E:\ACM\ACM常用代码.mht) | [7.最大公约数、最小公倍数](file:///E:\ACM\ACM常用代码.mht) | [8.组合序列](file:///E:\ACM\ACM常用代码.mht) |
| [9.快速傅立叶变换（FFT）](file:///E:\ACM\ACM常用代码.mht) | [10.Ronberg算法计算积分](file:///E:\ACM\ACM常用代码.mht) | [11.行列式计算](file:///E:\ACM\ACM常用代码.mht) | [12.求排列组合数](file:///E:\ACM\ACM常用代码.mht) |
|  |  |  |  |
| **字符串处理：** |  |  |  |
| [1.字符串替换](file:///E:\ACM\ACM常用代码.mht) | [2.字符串查找](file:///E:\ACM\ACM常用代码.mht) | [3.字符串截取](file:///E:\ACM\ACM常用代码.mht) |  |
|  |  |  |  |
| **计算几何：** |  |  |  |
| [1.叉乘法求任意多边形面积](file:///E:\ACM\ACM常用代码.mht) | [2.求三角形面积](file:///E:\ACM\ACM常用代码.mht) | [3.两矢量间角度](file:///E:\ACM\ACM常用代码.mht) | [4.两点距离（2D、3D）](file:///E:\ACM\ACM常用代码.mht) |
| [5.射向法判断点是否在多边形内部](file:///E:\ACM\ACM常用代码.mht) | [6.判断点是否在线段上](file:///E:\ACM\ACM常用代码.mht) | [7.判断两线段是否相交](file:///E:\ACM\ACM常用代码.mht) | [8.判断线段与直线是否相交](file:///E:\ACM\ACM常用代码.mht) |
| [9.点到线段最短距离](file:///E:\ACM\ACM常用代码.mht) | [10.求两直线的交点](file:///E:\ACM\ACM常用代码.mht) | [11.判断一个封闭图形是凹集还是凸集](file:///E:\ACM\ACM常用代码.mht) | [12.Graham扫描法寻找凸包](file:///E:\ACM\ACM常用代码.mht) |
|  |  |  |  |
| **数论：** |  |  |  |
| [1.x的二进制长度](file:///E:\ACM\ACM常用代码.mht) | [2.返回x的二进制表示中从低到高的第i位](file:///E:\ACM\ACM常用代码.mht) | [3.模取幂运算](file:///E:\ACM\ACM常用代码.mht) | [4.求解模线性方程](file:///E:\ACM\ACM常用代码.mht) |
| [5.求解模线性方程组(中国余数定理)](file:///E:\ACM\ACM常用代码.mht) | [6.筛法素数产生器](file:///E:\ACM\ACM常用代码.mht) | [7.判断一个数是否素数](file:///E:\ACM\ACM常用代码.mht) |  |
|  |  |  |  |
| **图论：** |  |  |  |
| [1.Prim算法求最小生成树](file:///E:\ACM\ACM常用代码.mht) | [2.Dijkstra算法求单源最短路径](file:///E:\ACM\ACM常用代码.mht) | [3.Bellman-ford算法求单源最短路径](file:///E:\ACM\ACM常用代码.mht) | [4.Floyd算法求每对节点间最短路径](file:///E:\ACM\ACM常用代码.mht) |
|  |  |  |  |
| **排序/查找：** |  |  |  |
| [1.快速排序](file:///E:\ACM\ACM常用代码.mht) | [2.希尔排序](file:///E:\ACM\ACM常用代码.mht) | [3.选择法排序](file:///E:\ACM\ACM常用代码.mht) | [4.二分查找](file:///E:\ACM\ACM常用代码.mht) |
|  |  |  |  |
| **数据结构：** |  |  |  |
| [1.顺序队列](file:///E:\ACM\ACM常用代码.mht) | [2.顺序栈](file:///E:\ACM\ACM常用代码.mht) | [3.链表](file:///E:\ACM\ACM常用代码.mht) | [4.链栈](file:///E:\ACM\ACM常用代码.mht) |
| [5.二叉树](file:///E:\ACM\ACM常用代码.mht) |  |  |  |

**一、数学问题**

1.精度计算——大数阶乘

|  |  |
| --- | --- |
| 语法：int result=factorial(int n); | |
| 参数： | |
| n： | n 的阶乘 |
| 返回值： | 阶乘结果的位数 |
| 注意： |  |
|  | 本程序直接输出n!的结果，需要返回结果请保留long a[] |
|  | 需要 math.h |
| 源程序： |  |
|  | int factorial(int n) { long a[10000]; int i,j,l,c,m=0,w;  a[0]=1;  for(i=1;i<=n;i++)     {      c=0;      for(j=0;j<=m;j++)         {          a[j]=a[j]\*i+c;          c=a[j]/10000;          a[j]=a[j]%10000;      }      if(c>0) {m++;a[m]=c;}  }   w=m\*4+log10(a[m])+1; printf("\n%ld",a[m]);  for(i=m-1;i>=0;i--) printf("%4.4ld",a[i]); return w; } |

2.精度计算——乘法（大数乘小数）

|  |  |
| --- | --- |
| 语法：mult(char c[],char t[],int m); | |
| 参数： | |
| c[]： | 被乘数，用字符串表示，位数不限 |
| t[]： | 结果，用字符串表示 |
| m： | 乘数，限定10以内 |
| 返回值： | Null |
| 注意： |  |
|  | 需要 string.h |
| 源程序： |  |
|  | void mult(char c[],char t[],int m) {     int i,l,k,flag,add=0;     char s[100];     l=strlen(c);     for (i=0;i<l;i++)         s[l-i-1]=c[i]-'0';      for (i=0;i<l;i++)            {            k=s[i]\*m+add;            if (k>=10) {s[i]=k%10;add=k/10;flag=1;} else {s[i]=k;flag=0;add=0;}            }     if (flag) {l=i+1;s[i]=add;} else l=i;      for (i=0;i<l;i++)         t[l-1-i]=s[i]+'0';     t[l]='\0'; } |

3.精度计算——乘法（大数乘大数）

|  |  |
| --- | --- |
| 语法：mult(char a[],char b[],char s[]); | |
| 参数： | |
| a[]： | 被乘数，用字符串表示，位数不限 |
| b[]： | 乘数，用字符串表示，位数不限 |
| t[]： | 结果，用字符串表示 |
| 返回值： | Null |
| 注意： |  |
|  | 空间复杂度为 o(n^2) |
|  | 需要 string.h |
| 源程序： |  |
|  | void mult(char a[],char b[],char s[]) {     int i,j,k=0,alen,blen,sum=0,res[65][65]={0},flag=0;     char result[65];     alen=strlen(a);blen=strlen(b);      for (i=0;i<alen;i++)     for (j=0;j<blen;j++) res[i][j]=(a[i]-'0')\*(b[j]-'0');      for (i=alen-1;i>=0;i--)         {             for (j=blen-1;j>=0;j--) sum=sum+res[i+blen-j-1][j];             result[k]=sum%10;             k=k+1;             sum=sum/10;         }      for (i=blen-2;i>=0;i--)         {             for (j=0;j<=i;j++) sum=sum+res[i-j][j];             result[k]=sum%10;             k=k+1;             sum=sum/10;         }     if (sum!=0) {result[k]=sum;k=k+1;}      for (i=0;i<k;i++) result[i]+='0';     for (i=k-1;i>=0;i--) s[i]=result[k-1-i];     s[k]='\0';      while(1)         {         if (strlen(s)!=strlen(a)&&s[0]=='0')              strcpy(s,s+1);         else             break;         } } |

4.精度计算——加法

|  |  |
| --- | --- |
| 语法：add(char a[],char b[],char s[]); | |
| 参数： | |
| a[]： | 被乘数，用字符串表示，位数不限 |
| b[]： | 乘数，用字符串表示，位数不限 |
| t[]： | 结果，用字符串表示 |
| 返回值： | null |
| 注意： |  |
|  | 空间复杂度为 o(n^2) |
|  | 需要 string.h |
| 源程序： |  |
|  | void add(char a[],char b[],char back[]) {     int i,j,k,up,x,y,z,l;     char \*c;     if (strlen(a)>strlen(b)) l=strlen(a)+2; else l=strlen(b)+2;     c=(char \*) malloc(l\*sizeof(char));     i=strlen(a)-1;     j=strlen(b)-1;     k=0;up=0;     while(i>=0||j>=0)         {             if(i<0) x='0'; else x=a[i];             if(j<0) y='0'; else y=b[j];             z=x-'0'+y-'0';             if(up) z+=1;             if(z>9) {up=1;z%=10;} else up=0;             c[k++]=z+'0';             i--;j--;         }     if(up) c[k++]='1';     i=0;     c[k]='\0';     for(k-=1;k>=0;k--)         back[i++]=c[k];     back[i]='\0'; } |

5.精度计算——减法

|  |  |
| --- | --- |
| 语法：sub(char s1[],char s2[],char t[]); | |
| 参数： | |
| s1[]： | 被减数，用字符串表示，位数不限 |
| s2[]： | 减数，用字符串表示，位数不限 |
| t[]： | 结果，用字符串表示 |
| 返回值： | null |
| 注意： |  |
|  | 默认s1>=s2，程序未处理负数情况 |
|  | 需要 string.h |
| 源程序： |  |
|  | void sub(char s1[],char s2[],char t[]) {     int i,l2,l1,k;     l2=strlen(s2);l1=strlen(s1);     t[l1]='\0';l1--;     for (i=l2-1;i>=0;i--,l1--)         {         if (s1[l1]-s2[i]>=0)              t[l1]=s1[l1]-s2[i]+'0';         else             {             t[l1]=10+s1[l1]-s2[i]+'0';             s1[l1-1]=s1[l1-1]-1;             }         }     k=l1;     while(s1[k]<0) {s1[k]+=10;s1[k-1]-=1;k--;}     while(l1>=0) {t[l1]=s1[l1];l1--;} loop:     if (t[0]=='0')          {         l1=strlen(s1);         for (i=0;i<l1-1;i++) t[i]=t[i+1];         t[l1-1]='\0';         goto loop;         }     if (strlen(t)==0) {t[0]='0';t[1]='\0';} } |

6.任意进制转换

|  |  |
| --- | --- |
| 语法：conversion(char s1[],char s2[],long d1,long d2); | |
| 参数： | |
| s[]： | 原进制数字，用字符串表示 |
| s2[]： | 转换结果，用字符串表示 |
| d1： | 原进制数 |
| d2： | 需要转换到的进制数 |
| 返回值： | null |
| 注意： |  |
|  | 高于9的位数用大写'A'～'Z'表示，2～16位进制通过验证 |
| 源程序： |  |
|  | void conversion(char s[],char s2[],long d1,long d2) {     long i,j,t,num;     char c;     num=0;     for (i=0;s[i]!='\0';i++)         {         if (s[i]<='9'&&s[i]>='0') t=s[i]-'0'; else t=s[i]-'A'+10;         num=num\*d1+t;         }     i=0;     while(1)         {         t=num%d2;         if (t<=9) s2[i]=t+'0'; else s2[i]=t+'A'-10;         num/=d2;         if (num==0) break;         i++;         }     for (j=0;j<i/2;j++)         {c=s2[j];s2[j]=s[i-j];s2[i-j]=c;}     s2[i+1]='\0'; } |

7.最大公约数、最小公倍数

|  |  |
| --- | --- |
| 语法：resulet=hcf(int a,int b)、result=lcd(int a,int b) | |
| 参数： | |
| a： | int a，求最大公约数或最小公倍数 |
| b： | int b，求最大公约数或最小公倍数 |
| 返回值： | 返回最大公约数（hcf）或最小公倍数（lcd） |
| 注意： |  |
|  | lcd 需要连同 hcf 使用 |
| 源程序： |  |
|  | int hcf(int a,int b) {     int r=0;     while(b!=0)         {         r=a%b;         a=b;         b=r;         }     return(a); }  lcd(int u,int v,int h) {     return(u\*v/h); } |

8.组合序列

|  |  |
| --- | --- |
| 语法：m\_of\_n(int m, int n1, int m1, int\* a, int head) | |
| 参数： | |
| m： | 组合数C的上参数 |
| n1： | 组合数C的下参数 |
| m1： | 组合数C的上参数，递归之用 |
| \*a： | 1～n的整数序列数组 |
| head： | 头指针 |
| 返回值： | null |
| 注意： |  |
|  | \*a需要自行产生 |
|  | 初始调用时，m=m1、head=0 |
|  | 调用例子：求C(m,n)序列：m\_of\_n(m,n,m,a,0); |
| 源程序： |  |
|  | void m\_of\_n(int m, int n1, int m1, int\* a, int head)  {      int i,t;      if(m1<0 || m1>n1) return;      if(m1==n1)          {          for(i=0;i<m;i++) cout<<a[i]<<' '; // 输出序列          cout<<'\n';          return;          }      m\_of\_n(m,n1-1,m1,a,head); // 递归调用      t=a[head];a[head]=a[n1-1+head];a[n1-1+head]=t;     m\_of\_n(m,n1-1,m1-1,a,head+1); // 再次递归调用      t=a[head];a[head]=a[n1-1+head];a[n1-1+head]=t; } |

9.快速傅立叶变换（FFT）

|  |  |
| --- | --- |
| 语法：kkfft(double pr[],double pi[],int n,int k,double fr[],double fi[],int l,int il); | |
| 参数： | |
| pr[n]： | 输入的实部 |
| pi[n]： | 数入的虚部 |
| n，k： | 满足n=2^k |
| fr[n]： | 输出的实部 |
| fi[n]： | 输出的虚部 |
| l： | 逻辑开关，0 FFT，1 ifFT |
| il： | 逻辑开关，0 输出按实部/虚部；1 输出按模/幅角 |
| 返回值： | null |
| 注意： |  |
|  | 需要 math.h |
| 源程序： |  |
|  | void kkfft(pr,pi,n,k,fr,fi,l,il)  int n,k,l,il;  double pr[],pi[],fr[],fi[];  {     int it,m,is,i,j,nv,l0;      double p,q,s,vr,vi,poddr,poddi;      for (it=0; it<=n-1; it++)          {          m=it; is=0;          for (i=0; i<=k-1; i++)              {j=m/2; is=2\*is+(m-2\*j); m=j;}         fr[it]=pr[is]; fi[it]=pi[is];          }      pr[0]=1.0; pi[0]=0.0;      p=6.283185306/(1.0\*n);      pr[1]=cos(p); pi[1]=-sin(p);      if (l!=0) pi[1]=-pi[1];      for (i=2; i<=n-1; i++)          {        p=pr[i-1]\*pr[1];        q=pi[i-1]\*pi[1];          s=(pr[i-1]+pi[i-1])\*(pr[1]+pi[1]);          pr[i]=p-q; pi[i]=s-p-q;          }      for (it=0; it<=n-2; it=it+2)          {        vr=fr[it]; vi=fi[it];          fr[it]=vr+fr[it+1]; fi[it]=vi+fi[it+1];          fr[it+1]=vr-fr[it+1]; fi[it+1]=vi-fi[it+1];          }      m=n/2; nv=2;      for (l0=k-2; l0>=0; l0--)          {         m=m/2; nv=2\*nv;          for (it=0; it<=(m-1)\*nv; it=it+nv)              for (j=0; j<=(nv/2)-1; j++)                  {                p=pr[m\*j]\*fr[it+j+nv/2];                  q=pi[m\*j]\*fi[it+j+nv/2];                  s=pr[m\*j]+pi[m\*j];                  s=s\*(fr[it+j+nv/2]+fi[it+j+nv/2]);                  poddr=p-q; poddi=s-p-q;                  fr[it+j+nv/2]=fr[it+j]-poddr;                  fi[it+j+nv/2]=fi[it+j]-poddi;                  fr[it+j]=fr[it+j]+poddr;                  fi[it+j]=fi[it+j]+poddi;                  }          }      if (l!=0)          for (i=0; i<=n-1; i++)              {            fr[i]=fr[i]/(1.0\*n);              fi[i]=fi[i]/(1.0\*n);              }      if (il!=0)              for (i=0; i<=n-1; i++)              {            pr[i]=sqrt(fr[i]\*fr[i]+fi[i]\*fi[i]);              if (fabs(fr[i])<0.000001\*fabs(fi[i]))                  {                if ((fi[i]\*fr[i])>0) pi[i]=90.0;                  else pi[i]=-90.0;                  }              else                  pi[i]=atan(fi[i]/fr[i])\*360.0/6.283185306;              }      return;  } |

10.Ronberg算法计算积分

|  |  |
| --- | --- |
| 语法：result=integral(double a,double b); | |
| 参数： | |
| a： | 积分上限 |
| b： | 积分下限 |
| function f： | 积分函数 |
| 返回值： | f在（a,b）之间的积分值 |
| 注意： |  |
|  | function f(x)需要自行修改，程序中用的是sina(x)/x |
|  | 需要 math.h |
|  | 默认精度要求是1e-5 |
| 源程序： |  |
|  | double f(double x) {      return sin(x)/x; //在这里插入被积函数  }  double integral(double a,double b)  {      double h=b-a;      double t1=(1+f(b))\*h/2.0;     int k=1;      double r1,r2,s1,s2,c1,c2,t2;  loop:      double s=0.0;      double x=a+h/2.0;      while(x<b)          {          s+=f(x);          x+=h;          }      t2=(t1+h\*s)/2.0;     s2=t2+(t2-t1)/3.0;     if(k==1)       {          k++;h/=2.0;t1=t2;s1=s2;         goto loop;          }      c2=s2+(s2-s1)/15.0;      if(k==2){          c1=c2;k++;h/=2.0;          t1=t2;s1=s2;          goto loop;          }      r2=c2+(c2-c1)/63.0;      if(k==3){          r1=r2; c1=c2;k++;          h/=2.0;          t1=t2;s1=s2;         goto loop;          }      while(fabs(1-r1/r2)>1e-5){          r1=r2;c1=c2;k++;         h/=2.0;          t1=t2;s1=s2;          goto loop;          }      return r2; } |

11.行列式计算

|  |  |
| --- | --- |
| 语法：result=js(int s[][],int n) | |
| 参数： | |
| s[][]： | 行列式存储数组 |
| n： | 行列式维数，递归用 |
| 返回值： | 行列式值 |
| 注意： |  |
|  | 函数中常数N为行列式维度，需自行定义 |
| 源程序： |  |
|  | int js(s,n)  int s[][N],n;  {     int z,j,k,r,total=0;      int b[N][N];/\*b[N][N]用于存放，在矩阵s[N][N]中元素s[0]的余子式\*/      if(n>2)         {         for(z=0;z<n;z++)              {             for(j=0;j<n-1;j++)                   for(k=0;k<n-1;k++)                          if(k>=z) b[j][k]=s[j+1][k+1];  else b[j][k]=s[j+1][k];              if(z%2==0) r=s[0][z]\*js(b,n-1); /\*递归调用\*/              else r=(-1)\*s[0][z]\*js(b,n-1);              total=total+r;              }          }      else if(n==2)        total=s[0][0]\*s[1][1]-s[0][1]\*s[1][0];      return total;  } |

12.求排列组合数

|  |  |
| --- | --- |
| 语法：result=P(long n,long m); / result=long C(long n,long m); | |
| 参数： | |
| m： | 排列组合的上系数 |
| n： | 排列组合的下系数 |
| 返回值： | 排列组合数 |
| 注意： |  |
|  | 符合数学规则：m<＝n |
| 源程序： |  |
|  | long P(long n,long m) {     long p=1;     while(m!=0)         {p\*=n;n--;m--;}     return p; }  long C(long n,long m) {     long i,c=1;     i=m;     while(i!=0)         {c\*=n;n--;i--;}     while(m!=0)         {c/=m;m--;}     return c; } |

**二、字符串处理**

1.字符串替换

|  |  |
| --- | --- |
| 语法：replace(char str[],char key[],char swap[]); | |
| 参数： | |
| str[]： | 在此源字符串进行替换操作 |
| key[]： | 被替换的字符串，不能为空串 |
| swap[]： | 替换的字符串，可以为空串，为空串表示在源字符中删除key[] |
| 返回值： | null |
| 注意： |  |
|  | 默认str[]长度小于1000，如否，重新设定设定tmp大小 |
|  | 需要 string.h |
| 源程序： |  |
|  | void replace(char str[],char key[],char swap[]) {     int l1,l2,l3,i,j,flag;     char tmp[1000];     l1=strlen(str);     l2=strlen(key);     l3=strlen(swap);     for (i=0;i<=l1-l2;i++)         {         flag=1;         for (j=0;j<l2;j++)             if (str[i+j]!=key[j]) {flag=0;break;}         if (flag)             {             strcpy(tmp,str);             strcpy(&tmp[i],swap);             strcpy(&tmp[i+l3],&str[i+l2]);             strcpy(str,tmp);             i+=l3-1;             l1=strlen(str);             }         } } |

2.字符串查找

|  |  |
| --- | --- |
| 语法：result=strfind(char str[],char key[]); | |
| 参数： | |
| str[]： | 在此源字符串进行查找操作 |
| key[]： | 被查找的字符串，不能为空串 |
| 返回值： | 如果查找成功，返回key在str中第一次出现的位置，否则返回-1 |
| 注意： |  |
|  | 需要 string.h |
| 源程序： |  |
|  | int strfind(char str[],char key[]) {     int l1,l2,i,j,flag;     l1=strlen(str);     l2=strlen(key);     for (i=0;i<=l1-l2;i++)         {         flag=1;         for (j=0;j<l2;j++)             if (str[i+j]!=key[j]) {flag=0;break;}         if (flag) return i;         }     return -1; } |

3.字符串截取

|  |  |
| --- | --- |
| 语法：mid(char str[],int start,int len,char strback[]) | |
| 参数： | |
| str[]： | 操作的目标字符串 |
| start： | 从第start个字符串开始，截取长度为len的字符 |
| len： | 从第start个字符串开始，截取长度为len的字符 |
| strback[]： | 截取的到的字符 |
| 返回值： | 0：超出字符串长度，截取失败；1：截取成功 |
| 注意： |  |
|  | 需要 string.h |
| 源程序： |  |
|  | int mid(char str[],int start,int len,char strback[]) {     int l,i,k=0;     l=strlen(str);     if (start+len>l) return 0;     for (i=start;i<start+len;i++)         strback[k++]=str[i];     strback[k]='\0';     return 1; } |

**三、计算几何**

1.叉乘法求任意多边形面积

|  |  |
| --- | --- |
| 语法：result=polygonarea(Point \*polygon,int N); | |
| 参数： | |
| \*polygon： | 多变形顶点数组 |
| N： | 多边形顶点数目 |
| 返回值： | 多边形面积 |
| 注意： |  |
|  | 支持任意多边形，凹、凸皆可 |
|  | 多边形顶点输入时按顺时针顺序排列 |
| 源程序： |  |
|  | typedef struct {     double x,y; } Point;  double polygonarea(Point \*polygon,int N) {     int i,j;     double area = 0;      for (i=0;i<N;i++) {         j = (i + 1) % N;         area += polygon[i].x \* polygon[j].y;         area -= polygon[i].y \* polygon[j].x;         }      area /= 2;     return(area < 0 ? -area : area); } |

2.求三角形面积

|  |  |
| --- | --- |
| 语法：result=area3(float x1,float y1,float x2,float y2,float x3,float y3); | |
| 参数： | |
| x1～3： | 三角形3个顶点x坐标 |
| y1～3： | 三角形3个顶点y坐标 |
| 返回值： | 三角形面积 |
| 注意： |  |
|  | 需要 math.h |
| 源程序： |  |
|  | float area3(float x1,float y1,float x2,float y2,float x3,float y3) {     float a,b,c,p,s;     a=sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2));     b=sqrt((x1-x3)\*(x1-x3)+(y1-y3)\*(y1-y3));     c=sqrt((x3-x2)\*(x3-x2)+(y3-y2)\*(y3-y2));     p=(a+b+c)/2;     s=sqrt(p\*(p-a)\*(p-b)\*(p-c));     return s; } |

3.两矢量间角度

|  |  |
| --- | --- |
| 语法：result=angle(double x1, double y1, double x2, double y2); | |
| 参数： | |
| x/y1～2： | 两矢量的坐标 |
| 返回值： | 两的角度矢量 |
| 注意： |  |
|  | 返回角度为弧度制，并且以逆时针方向为正方向 |
|  | 需要 math.h |
| 源程序： |  |
|  | #define PI 3.1415926  double angle(double x1, double y1, double x2, double y2) {     double dtheta,theta1,theta2;      theta1 = atan2(y1,x1);     theta2 = atan2(y2,x2);     dtheta = theta2 - theta1;     while (dtheta > PI)         dtheta -= PI\*2;     while (dtheta < -PI)         dtheta += PI\*2;      return(dtheta); } |

4.两点距离（2D、3D）

|  |  |
| --- | --- |
| 语法：result=distance\_2d(float x1,float x2,float y1,float y2); | |
| 参数： | |
| x/y/z1～2： | 各点的x、y、z坐标 |
| 返回值： | 两点之间的距离 |
| 注意： |  |
|  | 需要 math.h |
| 源程序： |  |
|  | float distance\_2d(float x1,float x2,float y1,float y2)  {     return(sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2))); }   float distance\_3d(float x1,float x2,float y1,float y2,float z1,float z2) {     return(sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2)+(z1-z2)\*(z1-z2))); } |

5.射向法判断点是否在多边形内部

|  |  |
| --- | --- |
| 语法：result=insidepolygon(Point \*polygon,int N,Point p); | |
| 参数： | |
| \*polygon： | 多边形顶点数组 |
| N： | 多边形顶点个数 |
| p： | 被判断点 |
| 返回值： | 0：点在多边形内部；1：点在多边形外部 |
| 注意： |  |
|  | 若p点在多边形顶点或者边上，返回值不确定，需另行判断 |
|  | 需要 math.h |
| 源程序： |  |
|  | #define MIN(x,y) (x < y ? x : y) #define MAX(x,y) (x > y ? x : y)  typedef struct {     double x,y; } Point;  int insidepolygon(Point \*polygon,int N,Point p) {     int counter = 0;     int i;     double xinters;     Point p1,p2;      p1 = polygon[0];     for (i=1;i<=N;i++) {         p2 = polygon[i % N];         if (p.y > MIN(p1.y,p2.y)) {             if (p.y <= MAX(p1.y,p2.y)) {                 if (p.x <= MAX(p1.x,p2.x)) {                     if (p1.y != p2.y) {                         xinters = (p.y-p1.y)\*(p2.x-p1.x)/(p2.y-p1.y)+p1.x;                         if (p1.x == p2.x || p.x <= xinters)                             counter++;                         }                     }                 }             }             p1 = p2;         }      if (counter % 2 == 0)         return(OUTSIDE);     else         return(INSIDE); } |

6.判断点是否在线段上

|  |  |
| --- | --- |
| 语法：result=Pointonline(Point p1,Point p2,Point p); | |
| 参数： | |
| p1、p2： | 线段的两个端点 |
| p： | 被判断点 |
| 返回值： | 0：点在不在线段上；1：点在线段上 |
| 注意： |  |
|  | 若p线段端点上返回1 |
|  | 需要 math.h |
| 源程序： |  |
|  | #define MIN(x,y) (x < y ? x : y) #define MAX(x,y) (x > y ? x : y)  typedef struct { double x,y; } Point;  int FC(double x1,double x2) {     if (x1-x2<0.000002&&x1-x2>-0.000002) return 1; else return 0; }   int Pointonline(Point p1,Point p2,Point p) {     double x1,y1,x2,y2;     x1=p.x-p1.x;     x2=p2.x-p1.x;     y1=p.y-p1.y;     y2=p2.y-p1.y;     if (FC(x1\*y2-x2\*y1,0)==0) return 0;     if ((MIN(p1.x,p2.x)<=p.x&&p.x<=MAX(p1.x,p2.x))&&             (MIN(p1.y,p2.y)<=p.y&&p.y<=MAX(p1.y,p2.y)))         return 1; else return 0; } |

7.判断两线段是否相交

|  |  |
| --- | --- |
| 语法：result=sectintersect(Point p1,Point p2,Point p3,Point p4); | |
| 参数： | |
| p1～4： | 两条线段的四个端点 |
| 返回值： | 0：两线段不相交；1：两线段相交；2两线段首尾相接 |
| 注意： |  |
|  | p1!=p2;p3!=p4; |
| 源程序： |  |
|  | #define MIN(x,y) (x < y ? x : y) #define MAX(x,y) (x > y ? x : y)  typedef struct {     double x,y; } Point;  int lineintersect(Point p1,Point p2,Point p3,Point p4) {     Point tp1,tp2,tp3;     if ((p1.x==p3.x&&p1.y==p3.y)||(p1.x==p4.x&&p1.y==p4.y)||(p2.x==p3.x&&p2.y==p3.y)||(p2.x==p4.x&&p2.y==p4.y))         return 2; //快速排斥试验     if ((MIN(p1.x,p2.x)<p3.x&&p3.x<MAX(p1.x,p2.x)&&MIN(p1.y,p2.y)<p3.y<MAX(p1.y,p2.y))||             (MIN(p1.x,p2.x)<p4.x&&p3.x<MAX(p1.x,p2.x)&&MIN(p1.y,p2.y)<p3.y<MAX(p1.y,p2.y)))         ;else return 0;  //跨立试验     tp1.x=p1.x-p3.x;     tp1.y=p1.y-p3.y;     tp2.x=p4.x-p3.x;     tp2.y=p4.y-p3.y;     tp3.x=p2.x-p3.x;     tp3.y=p2.y-p3.y;     if ((tp1.x\*tp2.y-tp1.y\*tp2.x)\*(tp2.x\*tp3.y-tp2.y\*tp3.x)>=0) return 1; else return 0; } |

8.判断线段与直线是否相交

|  |  |
| --- | --- |
| 语法：result=lineintersect(Point p1,Point p2,Point p3,Point p4); | |
| 参数： | |
| p1、p2： | 线段的两个端点 |
| p3、p4： | 直线上的两个点 |
| 返回值： | 0：线段直线不相交；1：线段和直线相交 |
| 注意： |  |
|  | 如线段在直线上，返回 1 |
| 源程序： |  |
|  | typedef struct {     double x,y; } Point;  int lineintersect(Point p1,Point p2,Point p3,Point p4) {     Point tp1,tp2,tp3;     tp1.x=p1.x-p3.x;     tp1.y=p1.y-p3.y;     tp2.x=p4.x-p3.x;     tp2.y=p4.y-p3.y;     tp3.x=p2.x-p3.x;     tp3.y=p2.y-p3.y;     if ((tp1.x\*tp2.y-tp1.y\*tp2.x)\*(tp2.x\*tp3.y-tp2.y\*tp3.x)>=0) return 1; else return 0; } |

9.点到线段最短距离

|  |  |
| --- | --- |
| 语法：result=mindistance(Point p1,Point p2,Point q); | |
| 参数： | |
| p1、p2： | 线段的两个端点 |
| q： | 判断点 |
| 返回值： | 点q到线段p1p2的距离 |
| 注意： |  |
|  | 需要 math.h |
| 源程序： |  |
|  | #define MIN(x,y) (x < y ? x : y) #define MAX(x,y) (x > y ? x : y)  typedef struct {     double x,y; } Point;  double mindistance(Point p1,Point p2,Point q) {     int flag=1;     double k;     Point s;     if (p1.x==p2.x) {s.x=p1.x;s.y=q.y;flag=0;}     if (p1.y==p2.y) {s.x=q.x;s.y=p1.y;flag=0;}     if (flag)         {         k=(p2.y-p1.y)/(p2.x-p1.x);         s.x=(k\*k\*p1.x+k\*(q.y-p1.y)+q.x)/(k\*k+1);         s.y=k\*(s.x-p1.x)+p1.y;         }     if (MIN(p1.x,p2.x)<=s.x&&s.x<=MAX(p1.x,p2.x))         return sqrt((q.x-s.x)\*(q.x-s.x)+(q.y-s.y)\*(q.y-s.y));     else         return MIN(sqrt((q.x-p1.x)\*(q.x-p1.x)+(q.y-p1.y)\*(q.y-p1.y)),sqrt((q.x-p2.x)\*(q.x-p2.x)+(q.y-p2.y)\*(q.y-p2.y))); } |

10.求两直线的交点

|  |  |
| --- | --- |
| 语法：result=mindistance(Point p1,Point p2,Point q); | |
| 参数： | |
| p1～p4： | 直线上不相同的两点 |
| \*p： | 通过指针返回结果 |
| 返回值： | 1：两直线相交；2：两直线平行 |
| 注意： |  |
|  | 如需要判断两线段交点，检验k和对应k1（注释中）的值是否在0～1之间，用在0～1之间的那个求交点 |
| 源程序： |  |
|  | typedef struct {    double x,y; } Point;  int linecorss(Point p1,Point p2,Point p3,Point p4,Point \*p) {   double k;  //同一直线  if ((p4.x-p3.x)\*(p1.y-p3.y)-(p4.y-p3.y)\*(p1.x-p3.x)==0&&  (p2.x-p1.x)\*(p1.y-p3.y)-(p2.y-p1.y)\*(p1.x-p3.x)==0) return 2;  //平行，不同一直线  if ((p4.y-p3.y)\*(p2.x-p1.x)-(p4.x-p3.x)\*(p2.y-p1.y)==0) return 0;  k=((p4.x-p3.x)\*(p1.y-p3.y)-(p4.y-p3.y)\*(p1.x-p3.x))/((p4.y-p3.y)\*(p2.x-p1.x)-(p4.x-p3.x)\*(p2.y-p1.y));  //k1=((p2.x-p1.x)\*(p1.y-p3.y)-(p2.y-p1.y)\*(p1.x-p3.x))/((p4.y-p3.y)\*(p2.x-p1.x)-(p4.x-p3.x)\*(p2.y-p1.y));  (\*p).x=p1.x+k\*(p2.x-p1.x);  (\*p).y=p1.y+k\*(p2.y-p1.y);  return 1;//有交点} |

11.判断一个封闭图形是凹集还是凸集

|  |  |
| --- | --- |
| 语法：result=convex(Point \*p,int n); | |
| 参数： | |
| \*p： | 封闭曲线顶点数组 |
| n： | 封闭曲线顶点个数 |
| 返回值： | 1：凸集；-1：凹集；0：曲线不符合要求无法计算 |
| 注意： |  |
|  | 默认曲线为简单曲线：无交叉、无圈 |
| 源程序： |  |
|  | typedef struct {     double x,y; } Point;  int convex(Point \*p,int n) {     int i,j,k;     int flag = 0;     double z;      if (n < 3)         return(0);      for (i=0;i<n;i++) {         j = (i + 1) % n;         k = (i + 2) % n;         z = (p[j].x - p[i].x) \* (p[k].y - p[j].y);         z -= (p[j].y - p[i].y) \* (p[k].x - p[j].x);         if (z < 0)             flag |= 1;         else if (z > 0)             flag |= 2;         if (flag == 3)             return －1; //CONCAVE         }     if (flag != 0)         return 1; //CONVEX     else     return 0; } |

12.Graham扫描法寻找凸包

|  |  |
| --- | --- |
| 语法：Graham\_scan(Point PointSet[],Point ch[],int n,int &len); | |
| 参数： | |
| PointSet[]： | 输入的点集 |
| ch[]： | 输出的凸包上的点集，按照逆时针方向排列 |
| n： | PointSet中的点的数目 |
| len： | 输出的凸包上的点的个数 |
| 返回值： | null |
| 源程序： |  |
|  | struct Point{     float x,y; };  float multiply(Point p1,Point p2,Point p0) {     return((p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y));  }  float distance(Point p1,Point p2) {     return(sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y)));  }  void Graham\_scan(Point PointSet[],Point ch[],int n,int &len) {     int i,j,k=0,top=2;     Point tmp;     for(i=1;i<n;i++)     if ((PointSet[i].y<PointSet[k].y)||((PointSet[i].y==PointSet[k].y)&&(PointSet[i].x<PointSet[k].x)))     k=i;     tmp=PointSet[0];     PointSet[0]=PointSet[k];     PointSet[k]=tmp;      for (i=1;i<n-1;i++)         {         k=i;         for (j=i+1;j<n;j++)             if ( (multiply(PointSet[j],PointSet[k],PointSet[0])>0) ||                      ((multiply(PointSet[j],PointSet[k],PointSet[0])==0)                          &&(distance(PointSet[0],PointSet[j])<distance(PointSet[0],PointSet[k])))   )                 k=j;         tmp=PointSet[i];         PointSet[i]=PointSet[k];         PointSet[k]=tmp;         }     ch[0]=PointSet[0];     ch[1]=PointSet[1];     ch[2]=PointSet[2];      for (i=3;i<n;i++)         {         while (multiply(PointSet[i],ch[top],ch[top-1])>=0) top--;         ch[++top]=PointSet[i];         }     len=top+1; } |

**四、数论**

1.x的二进制长度

|  |  |
| --- | --- |
| 语法：result=BitLength(int x); | |
| 参数： | |
| x： | 测长的x |
| 返回值： | x的二进制长度 |
| 源程序： |  |
|  | int BitLength(int x) {     int d = 0;     while (x > 0) {         x >>= 1;         d++;     }     return d; } |

2.返回x的二进制表示中从低到高的第i位

|  |  |
| --- | --- |
| 语法：result=BitAt(int x, int i); | |
| 参数： | |
| x： | 十进制 x |
| i： | 要求二进制的第i位 |
| 返回值： | 返回x的二进制表示中从低到高的第i位 |
| 注意： |  |
|  | 最低位为第一位 |
| 源程序： |  |
|  | int BitAt(int x, int i) {     return ( x & (1 << (i-1)) ); } |

3.模取幂运算

|  |  |
| --- | --- |
| 语法：result=Modular\_Expoent(int a,int b,int n); | |
| 参数： | |
| a、b、n： | a^b mod n 的对应参数 |
| 返回值： | a^b mod n 的值 |
| 注意： |  |
|  | 需要BitLength和BitAt |
| 源程序： |  |
|  | int Modular\_Expoent(int a,int b,int n) {     int i, y=1;     for (i = BitLength(b); i > 0; i--)         {          y = (y\*y)%n;         if (BitAt(b,i) > 0)          y = (y\*a)%n;         }     return y; } |

4.求解模线性方程

|  |  |
| --- | --- |
| 语法：result＝modular\_equation(int a,int b,int n); | |
| 参数： | |
| a、b、n： | ax=b (mod n) 的对应参数 |
| 返回值： | 方程的解 |
| 源程序： |  |
|  | int ext\_euclid(int a,int b,int &x,int &y)  //求gcd(a,b)=ax+by {     int t,d;     if (b==0) {x=1;y=0;return a;}     d=ext\_euclid(b,a %b,x,y);     t=x;     x=y;     y=t-a/b\*y;     return d; }  void modular\_equation(int a,int b,int n) {     int e,i,d;     int x,y;     d=ext\_euclid(a,n,x,y);     if (b%d>0)        printf("No answer!\n");     else         {        e=(x\*(b/d))%n;         for (i=0;i<d;i++)             printf("The %dth answer is : %ld\n",i+1,(e+i\*(n/d))%n);          } } |

5.求解模线性方程组(中国余数定理)

|  |  |
| --- | --- |
| 语法：result=Modular\_Expoent(int a,int b,int n); | |
| 参数： | |
| B[]、W[]： | a=B[] (mod W[]) 的对应参数 |
| 返回值： | a 的值 |
| 注意： |  |
|  | 其中W[],B[]已知，W[i]>0且W[i]与W[j]互质, 求a |
| 源程序： |  |
|  | int ext\_euclid(int a,int b,int &x,int &y)  //求gcd(a,b)=ax+by {     int t,d;     if (b==0) {x=1;y=0;return a;}     d=ext\_euclid(b,a %b,x,y);     t=x;     x=y;     y=t-a/b\*y;     return d; }   int China(int B[],int W[],int k) {    int i;     int d,x,y,a=0,m,n=1;     for (i=0;i<k;i++)         n\*=W[i];     for (i=0;i<k;i++)        {        m=n/W[i];         d=ext\_euclid(W[i],m,x,y);         a=(a+y\*m\*B[i])%n;         }     if (a>0) return a;     else return(a+n); } |

6.筛法素数产生器

|  |  |
| --- | --- |
| 语法：result=prime(int a[],int n); | |
| 参数： | |
| a[]： | 用于返回素数的数组 |
| n： | 产生n以内的素数，按升序放入a[]中 |
| 返回值： | n以内素数的个数 |
| 注意： |  |
|  | 其中W[],B[]已知，W[i]>0且W[i]与W[j]互质, 求a |
| 源程序： |  |
|  | int prime(int a[],int n) {     int i,j,k,x,num,\*b;     n++;     n/=2;     b=(int \*)malloc(sizeof(int)\*(n+1)\*2);     a[0]=2;a[1]=3;num=2;     for(i=1;i<=2\*n;i++)         b[i]=0;     for(i=3;i<=n;i+=3)         for(j=0;j<2;j++)             {             x=2\*(i+j)-1;             while(b[x]==0)                 {                 a[num++]=x;                 for(k=x;k<=2\*n;k+=x)                     b[k]=1;                 }             }     return num; } |

7.判断一个数是否素数

|  |  |
| --- | --- |
| 语法：result=comp(int n); | |
| 参数： | |
| n： | 判断n是否素数 |
| 返回值： | 素数返回1，否则返回0 |
| 源程序： |  |
|  | int comp(int n) {    int i,flag=1;     for (i=2;i<=sqrt(n);i++)     if (n%i==0) {flag=0;break;}     if (flag==1) return 1; else return 0; } |

**五、图论**

1.Prim算法求最小生成树

|  |  |
| --- | --- |
| 语法：prim(Graph G,int vcount,int father[]); | |
| 参数： | |
| G： | 图，用邻接矩阵表示 |
| vcount： | 表示图的顶点个数 |
| father[]： | 用来记录每个节点的父节点 |
| 返回值： | null |
| 注意： |  |
|  | 常数max\_vertexes为图最大节点数 |
|  | 常数infinity为无穷大 |
| 源程序： |  |
|  | #define infinity 1000000 #define max\_vertexes 5   typedef int Graph[max\_vertexes][max\_vertexes];  void prim(Graph G,int vcount,int father[]) {     int i,j,k;     int lowcost[max\_vertexes],closeset[max\_vertexes],used[max\_vertexes];     for (i=0;i<vcount;i++)         {         lowcost[i]=G[0][i];         closeset[i]=0;          used[i]=0;         father[i]=-1;          }     used[0]=1;      for (i=1;i<vcount;i++)         {         j=0;         while (used[j]) j++;         for (k=0;k<vcount;k++)             if ((!used[k])&&(lowcost[k]<lowcost[j])) j=k;         father[j]=closeset[j];          used[j]=1;         for (k=0;k<vcount;k++)             if (!used[k]&&(G[j][k]<lowcost[k]))                 { lowcost[k]=G[j][k];                 closeset[k]=j; }         } } |

2.Dijkstra算法求单源最短路径

|  |  |
| --- | --- |
| 语法：result=Dijkstra(Graph G,int n,int s,int t, int path[]); | |
| 参数： | |
| G： | 图，用邻接矩阵表示 |
| n： | 图的顶点个数 |
| s： | 开始节点 |
| t： | 目标节点 |
| path[]： | 用于返回由开始节点到目标节点的路径 |
| 返回值： | 最短路径长度 |
| 注意： |  |
|  | 输入的图的权必须非负 |
|  | 顶点标号从0开始 |
|  | 用如下方法打印路径：     i=t;     while (i!=s)         {         printf("%d<--",i+1);         i=path[i];         }     printf("%d\n",s+1); |
| 源程序： |  |
|  | int Dijkstra(Graph G,int n,int s,int t, int path[]) {     int i,j,w,minc,d[max\_vertexes],mark[max\_vertexes];     for (i=0;i<n;i++) mark[i]=0;     for (i=0;i<n;i++)         { d[i]=G[s][i];         path[i]=s; }     mark[s]=1;path[s]=0;d[s]=0;     for (i=1;i<n;i++)         {        minc=infinity;         w=0;         for (j=0;j<n;j++)             if ((mark[j]==0)&&(minc>=d[j])) {minc=d[j];w=j;}         mark[w]=1;         for (j=0;j<n;j++)         if ((mark[j]==0)&&(G[w][j]!=infinity)&&(d[j]>d[w]+G[w][j]))             { d[j]=d[w]+G[w][j];             path[j]=w; }         }     return d[t]; } |

3.Bellman-ford算法求单源最短路径

|  |  |
| --- | --- |
| 语法：result=Bellman\_ford(Graph G,int n,int s,int t,int path[],int success); | |
| 参数： | |
| G： | 图，用邻接矩阵表示 |
| n： | 图的顶点个数 |
| s： | 开始节点 |
| t： | 目标节点 |
| path[]： | 用于返回由开始节点到目标节点的路径 |
| success： | 函数是否执行成功 |
| 返回值： | 最短路径长度 |
| 注意： |  |
|  | 输入的图的权可以为负，如果存在一个从源点可达的权为负的回路则success=0 |
|  | 顶点标号从0开始 |
|  | 用如下方法打印路径：     i=t;     while (i!=s)         {         printf("%d<--",i+1);         i=path[i];         }     printf("%d\n",s+1); |
| 源程序： |  |
|  | int Bellman\_ford(Graph G,int n,int s,int t,int path[],int success) {     int i,j,k,d[max\_vertexes];     for (i=0;i<n;i++) {d[i]=infinity;path[i]=0;}     d[s]=0;     for (k=1;k<n;k++)         for (i=0;i<n;i++)             for (j=0;j<n;j++)                 if (d[j]>d[i]+G[i][j]) {d[j]=d[i]+G[i][j];path[j]=i;}     success=0;     for (i=0;i<n;i++)         for (j=0;j<n;j++)             if (d[j]>d[i]+G[i][j]) return 0;     success=1;     return d[t]; } |

4.Floyd-Warshall算法求每对节点间最短路径

|  |  |
| --- | --- |
| 语法：Floyd\_Washall(Graph G,int n,Graph D,Graph P); | |
| 参数： | |
| G： | 图，用邻接矩阵表示 |
| n： | 图的顶点个数 |
| D： | D[i,j]表示从i到j的最短距离 |
| P： | P[i,j]表示从i到j的最短路径上j 的父节点 |
| 返回值： | null |
| 源程序： |  |
|  | void Floyd\_Washall(Graph G,int n,Graph D,Graph P) {     int i,j,k;     for (i=0;i<n;i++)         for (j=0;j<n;j++)             { D[i][j]=G[i][j];                 P[i][j]=i; }     for (i=0;i<n;i++) { D[i][i]=0;P[i][i]=0; }     for (k=0;k<n;k++)         for (i=0;i<n;i++)             for (j=0;j<n;j++)                 if (D[i][j]>D[i][k]+D[k][j])                     { D[i][j]=D[i][k]+D[k][j];                         P[i][j]=P[k][j]; } } |

**六、排序/查找**

1.快速排序

|  |  |
| --- | --- |
| 语法：quicksort(int l,int r,int b[]); | |
| 参数： | |
| l： | 排序上界，开始时l=0 |
| r： | 排序下界，开始时r=数组元素个数 |
| b[]： | 被排序的元素 |
| 返回值： | null |
| 注意： |  |
|  | 输出升序序列 |
| 源程序： |  |
|  | void quicksort(int l,int r,int b[]) {     int i,j,x;     if(l>=r) return;     i=l;     j=r;     x=b[i];     while(i!=j)         {         while(b[j]>x&&j>i) j--;         if(i<j)             {             b[i]=b[j];             i++;             }         while(b[i]<x&&j>i)i++;             if(i<j)                 {                 b[j]=b[i];                 j--;                 }         }     b[i]=x;     quicksort(l,j-1,b);     quicksort(i+1,r,b); } |

2.希尔排序

|  |  |
| --- | --- |
| 语法：shellsort(int a[],int n); | |
| 参数： | |
| n： | 数组元素个数 |
| a[]： | 待排序数组 |
| 返回值： | null |
| 注意： |  |
|  | 输出升序序列 |
| 源程序： |  |
|  | void shellsort(int a[],int n) {     int i,j,g;     int temp,k;     g=n/2;     while(g!=0)         {         for(i=g+1;i<=n;i++)             {             temp=a[i];             j=i-g;             while(j>0)                 {                 k=j+g;                 if(a[j]<=a[k])                     j=0;                 else                     {                     temp=a[j];a[j]=a[k];a[k]=temp;                     }                 j=j-g;                 }             }         g=g/2;         } } |

3.选择法排序

|  |  |
| --- | --- |
| 语法：sort(int t[],int n); | |
| 参数： | |
| t[]： | 待排序数组 |
| n： | 数组t[]元素的个数 |
| 返回值： | null |
| 注意： |  |
|  | 输出升序序列 |
|  | 小规模排序用 |
| 源程序： |  |
|  | void sort(int t[],int n) {    int i,j,k,temp;     for (i=0;i<n;i++)         {         k=i;         for (j=i;j<n;j++) if (t[j]<t[k]) k=j;         temp=t[i];t[i]=t[k];t[k]=temp;         } } |

4.二分查找

|  |  |
| --- | --- |
| 语法：result=search\_bin(int \*t,int k); | |
| 参数： | |
| t[]： | 待查找数组 |
| k： | 查找关键字 |
| 返回值： | 如果k在t[]中存在，输出i：t[i]=k，否则输出－1 |
| 注意： |  |
|  | 要求查找数组是有序升序序列 |
| 源程序： |  |
|  | int search\_bin(int \*t,int k) {     int low=1,high=10,mid;     while (low<=high)         {         mid=(low+high)/2;         if (k==t[mid]) return mid;         else if (k<t[mid]) high=mid-1;         else low=mid+1;         }     return -1; } |

**七、数据结构**

1.顺序队列

|  |  |
| --- | --- |
| 源程序： |  |
|  | #define maxsize 100 typedef struct {     int data[maxsize];     int front;     int rear; } sqqueue;  int sqinit(sqqueue \*p) //队列初始化 {     p->front=0;     p->rear=0;     return 1; }  int enqueue(sqqueue \*q, int e) //入队 {     if((q->rear+1)%maxsize==q->front)         return 0;     else         q->data[q->rear]=e;     q->rear=(q->rear+1)%maxsize;     return 1; }  int dequeue(sqqueue \*q) //出队 {     int e;     if (q->front==q->rear)         return 0;     e=q->data[q->front];     q->front=(q->front+1)%maxsize;     return e; }  int empty(sqqueue \*q)  //判空 {     int v;     if (q->front==q->rear)         v=1;     else         v=0;      return v;  }  int gethead(sqqueue \*q)  //取得头元素 {     int e;     if (q->front==q->rear)          e=-1;     else         e=q->data[q->front];     return e; }  void display(sqqueue \*q) //显示所有元素 {     int s;     s=q->front;     printf("the sequeue is display:\n");     if (q->front==q->rear)         printf("the sequeue is empty!");     else         {         while(s<q->rear)             {             printf("->%d", q->data[s]);             s=(s+1)%maxsize;             }      printf("\n"); } }  main(sqqueue \*head)  //函数使用样例 {     int n,i,m,x,y,select,xq;     printf("create a empty sequeue\n");     sqinit(head);     printf("please input the sequeue length:\n");      scanf("%d",&n);     for (i=0;i<n;i++)         {         printf("please input a sequeue value:\n");         scanf("%d",&m);         enqueue(head,m);        }     printf("head->rear:%d\n",head->rear);     printf("head->front:%d\n",head->front);     display(head);     printf("select 1 \*\*\*\* enqueue() \n");     printf("select 2 \*\*\*\* dequeue() \n");     printf("select 3 \*\*\*\* empty () \n");     printf("select 4 \*\*\*\* gethead() \n");     printf("select 5 \*\*\*\* display() \n");     printf("please select (1--5):");     scanf("%d",&select);     switch(select)         {         case 1:             {              printf("please input a value :\n ");             scanf("%d",&x);             enqueue(head,x);             display(head);             break;             }         case 2:             {             dequeue(head);             display(head);             break;             }         case 3:             {         if(empty(head))             printf("the sequeue is empty");         else             printf("the sequeue is full");             }         case 4:             {             y=gethead(head);             printf("output head value:%d\n",y);             break;             }         case 5:             {             display(head);             break;             }         }     } } |

2.顺序栈

|  |  |
| --- | --- |
| 源程序： |  |
|  | #define m 100 typedef struct {     int stack[m];     int top; } stackstru;  init(stackstru \*s) /\*装入栈\*/ {     s->top=0;     return 1; }  int push(stackstru \*s,int x) /\*入栈操作\*/ {     if (s->top==m)         printf("the stack is overflow!\n");     else         {         s->top=s->top+1;         s->stack[s->top]=x;         } }  void display(stackstru \*s) /\*显示栈所有数据\*/ {     if(s->top==0)         printf("the stack is empty!\n");     else         {         while(s->top!=0)             {             printf("%d->",s->stack[s->top]);             s->top=s->top-1;             }         } }  int pop(stackstru \*s) /\*出栈操作并返回被删除的那个记录\*/ {     int y;     if(s->top==0)         printf("the stack is empty!\n");     else         {         y=s->stack[s->top];         s->top=s->top-1;         return y;         } }  int gettop(stackstru \*s) /\*得到栈顶数\*/ {      int e;     if(s->top==0)         return 0;     else          e=s->stack[s->top];     return e; }  main(stackstru \*p) //函数使用演示 {     int n,i,k,h,x1,x2,select;     printf("create a empty stack!\n");     init(p);     printf("input a stack length:\n");     scanf("%d",&n);     for(i=0;i<n;i++)         {         printf("input a stack value:\n");         scanf("%d",&k);         push(p,k);         }     printf("select 1:display()\n");     printf("select 2:push()\n");     printf("select 3:pop()\n");     printf("select 4:gettop()\n");     printf("input a your select(1-4):\n");     scanf("%d",&select);     switch(select)         {         case 1:             {             display(p);             break;             }         case 2:             {             printf("input a push a value:\n");             scanf("%d",&h);             push(p,h);             display(p);             break;             }         case 3:             {             x1=pop(p);             printf("x1->%d\n",x1);             display(p);             break;             }         case 4:             {             x2=gettop(p);             printf("x2->%d",x2);             break;             }         } } |

3.链表

|  |  |
| --- | --- |
| 源程序： |  |
|  | # define null 0   typedef char ElemType; /\* 字符型数据\*/   typedef struct LNode {     ElemType data;     struct LNode \*next; };  setnull(struct LNode \*\*p); int length (struct LNode \*\*p); ElemType get(struct LNode \*\*p,int i); void insert(struct LNode \*\*p,ElemType x,int i); int delete(struct LNode \*\*p,int i); void display(struct LNode \*\*p);  main() {     struct LNode \*head,\*q; /\*定义静态变量\*/     int select,x1,x2,x3,x4;     int i,n;      int m,g;     char e,y;       head=setnull(&head); /\*建议链表并设置为空表\*/     printf("请输入数据长度: ");     scanf("%d",&n);     for(i=1;i<n;i++);         {         printf("将数据插入到单链表中: ");         scanf("%d",&y);         insert(&head,y,i);} /\*插入数据到链表\*/         display(&head); /\*显示链表所有数据\*/          printf("select 1 求长度 length()\n");         printf("select 2 取结点 get()\n");         printf("select 3 求值查找 locate()\n");         printf("select 4 删除结点 delete()\n");         printf("input your select: ");         scanf("%d",&select);          switch(select)             {             case 1:                 {                 x1=length(&head);                 printf("输出单链表的长度%d ",x1);                 display(&head);                 }break;             case 2:                 {                 printf("请输入要取得结点: ");                 scanf("%d",&m);                 x2=get(&head,m);                 printf(x2);                 display(&head);                 }break;          case 3:                 {                 printf("请输入要查找的数据: ");                 scanf("%d",&e);                 x3=locate(&head,e);                 printf(x3);                 display(&head);                 }break;          case 4:                 {                 printf("请输入要删除的结点: ");                 scanf("%d",&g);                 x4=delete(&head,g);                 printf(x4);                 display(&head);                 }break;             }         } }  setnull(struct LNode \*\*p) {     \*p=null; }  int length (struct LNode \*\*p) {     int n=0;     struct LNode \*q=\*p;     while (q!=null)         {         n++;         q=q->next;         }     return(n); }  ElemType get(struct LNode \*\*p,int i) {     int j=1;     struct LNode \*q=\*p;     while (j<i&&q!=null)         {         q=q->next;         j++;         }         if(q!=null)             return(q->data);         else             printf("位置参数不正确!\n"); }  int locate(struct LNode \*\*p,ElemType x)     {     int n=0;     struct LNode \*q=\*p;     while (q!=null&&q->data!=x)         {         q=q->next;         n++;         }     if(q==null)         return(-1);     else         return(n+1); }  void insert(struct LNode \*\*p,ElemType x,int i)     {     int j=1;     struct LNode \*s,\*q;     s=(struct LNode \*)malloc(sizeof(struct LNode));     s->data=x;     q=\*p;     if(i==1)         {         s->next=q;         p=s;         }     else         {         while(j<i-1&&q->next!=null)             {             q=q->next;             j++;             }         if(j==i-1)             {             s->next=q->next;             q->next=s;             }         else              printf("位置参数不正确!\n");         }  }  int delete(struct LNode \*\*p,int i) {     int j=1;     struct LNode \*q=\*p,\*t;     if(i==1)         {         t=q;         \*p=q->next;         }     else         {         while(j<i-1&&q->next!=null)             {             q=q->next;             j++;             }         if(q->next!=null&&j==i-1)             {             t=q->next;             q->next=t->next;             }         else              printf("位置参数不正确!\n");         }     if(t=null)      free(t); }  void display(struct LNode \*\*p)     {      struct LNode \*q;     q=\*p;     printf("单链表显示: ");     if(q==null)         printf("链表为空!");     else if (q->next==null)         printf("%c\n",q->data);     else         {         while(q->next!=null)             {             printf("%c->",q->data);             q=q->next;             }         printf("%c",q->data);     }     printf("\n"); } |

4.链栈

|  |  |
| --- | --- |
| 源程序： |  |
|  | # define null 0   typedef struct stacknode {     int data;     struct stacknode \*next; } stacklink; typedef struct {     stacklink \*top;     int stacksize;     }stackk;  initlink(stackk \*s) {    s->top=(stacklink \*)malloc(sizeof(stacklink));     s->top->data=0;     s->top->next=null; }   int poplink(stackk \*s) {    stackk \*p;int v;     if(s->top->next==null) printf("the stackis empty\n");     else         {         v=s->top->next->data;          p=s->top->next;          s->top=s->top->next;         }      free(p);     return v; } }  int pushlink(stackk \*s,int x) {    stackk \*p;     p=(stacklink \*)malloc(sizeof(stacklink));     p->data=x;     p->next=s->top->next;     s->top->next=p; }  int gettop(stackk \*s) {    int e;     if(s==null) printf("the stack is empty!\n");     e=s->top->next->data;     return e; }  display(stackk \*s) {    stackk \*p;     p=s->top->next;     printf("display the stacklink:\n");     if (s->top=null) printf("the stacklink is empty!\n");     else        {        while(p)             {            printf("->%d",p->data);             p=p->next;            }         } }  main(stacklink \*p) {    int n,k,i,select,h,x1,x2;     printf("create a empty stacklink!\n");     initlink(p);     printf("input a stacklink length:\n");     scanf("%d",&n);     for (i=1;i<=n;i++)         {printf("input a stacklink value:\n");     scanf("%d",&k);     pushlink(p,k);         }     printf("select 1:display()\n");     printf("select 2:pushlink()\n");     printf("select 3:poplink()\n");     printf("select 4:gettop()\n");     printf("input a your select(1-4):\n");     scanf("%d",&select);     switch(select)         {case 1:              {display(p);break;}         case 2:            {printf("input a push a value :\n");             scanf("%d",&h);             pushlink(p,h);             display(p);             break;}         case 3:            {x1=poplink(p);printf("x1->%d\n",x1);             display(p);             break;}         case 4:            {x2=gettop(p);printf("x2->%d",x2);             break;}         } } |

5.二叉树

|  |  |
| --- | --- |
| 源程序： |  |
|  | typedef struct bitnode {     char data;     struct bitnode \*lchild, \*rchild; }bitnode, \*bitree;  void createbitree(t,n) bitnode \*\* t; int \*n; {     char x;     bitnode \*q;     \*n=\*n+1;     printf("\n Input %d DATA:",\*n);     x=getchar();     if(x!='\n') getchar();     if (x=='\n')         return;     q=(bitnode\*)malloc(sizeof(bitnode));     q->data=x;     q->lchild=NULL;     q->rchild=NULL;     \*t=q;     printf(" This Address is: %o, Data is: %c,\n Left Pointer is: %o, Right Pointer is: %o",q,q->data,q->lchild,q->rchild);     createbitree(&q->lchild,n);     createbitree(&q->rchild,n);     return; }  void visit(e) bitnode \*e; {     printf(" Address: %o, Data: %c, Left Pointer: %o, Right Pointer: %o\n",e,e->data,e->lchild,e->rchild); }  void preordertraverse(t) bitnode \*t; {     if(t)         {         visit(t);         preordertraverse(t->lchild);         preordertraverse(t->rchild);         return ;         }    else      return ; }  void countleaf(t,c) bitnode \*t; int \*c; {     if(t!=NULL)         {         if (t->lchild==NULL && t->rchild==NULL)         {\*c=\*c+1;         }     countleaf(t->lchild,c);     countleaf(t->rchild,c); } return; }  int treehigh(t) bitnode \*t; {    int lh,rh,h;     if(t==NULL)         h=0;     else        {         lh=treehigh(t->lchild);         rh=treehigh(t->rchild);         h=(lh>rh ? lh:rh)+1;         }     return h; }  main() {     bitnode \*t; int count=0;     int n=0;    printf("\n Please input TREE Data:\n");     createbitree(&t,&n);    printf("\n This is TREE struct: \n");     preordertraverse(t);    countleaf(t,&count);     printf("\n This TREE has %d leaves ",count);    printf(" , High of The TREE is: %d\n",treehigh(t)); } |

**1、回溯算法**

实验目的：熟练掌握回溯算法

实验内容：回溯算法的几种形式

a) 用回溯算法搜索子集树的一般模式

void search(int m)

{

if(m>n) //递归结束条件

output(); //相应的处理(输出结果)

else

{

a[m]=0; //设置状态：0表示不要该物品

search(m+1); //递归搜索：继续确定下一个物品

a[m]=1; //设置状态：1表示要该物品

search(m+1); //递归搜索：继续确定下一个物品

}

}

b) 用回溯算法搜索子集树的一般模式

void search(int m)

{

if(m>n) //递归结束条件

output(); //相应的处理(输出结果)

else

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

{

swap(m,i); //交换a[m]和a[i]

if()

if(canplace(m)) //如果m处可放置

search(m+1); //搜索下一层

swpa(m,i); //交换a[m]和a[i](换回来)

}

}

6． 素数环问题

把从1到20这20个数摆成一个环，要求相邻的两个数的和是一个素数。

分析：用回溯算法，考察所有可能的排列。

程序如下：

#include <stdio.h>

#include <math.h>

void search(int);

void init(); //初始化

void printresult(); //打印结果

int isprime(int); //判断该数是否是素数

void swap(int,int); //交换a[m]和a[i]

int a[21]; //a数组存放素数环

int main()

{

init();

search(2); //递归搜索

}

int isprime(int num)

{

int i,k;

k=sqrt(num);

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

if(num%i==0)

return(0);

return(1);

}

void printresult()

{

int i;

for(i=1;i<=20;i++)

printf("%3d",a[i]);

printf("\n");

}

void search(int m)

{

int i;

if(m>20) //当已经搜索到叶结点时

{

if(isprime(a[1]+a[20])) //如果a[1]+a[20]也是素数

printresult(); //输出当前解

return;

}

else

{

for(i=m;i<=20;i++) //(排列树)

{

swap(m,i); //交换a[m]和a[i]

if(isprime(a[m-1]+a[m])) //判断a[m-1]+a[m]是否是素数

search(m+1); //递归搜索下一个位置

swap(m,i); //把a[m]和a[i]换回来

}

}

}

void swap(int m, int i)

{

int t;

t=a[m];

a[m]=a[i];

a[i]=t;

}

void init()

{

int i;

for(i=0;i<21;i++)

a[i]=i;

}

**2.计算矩阵连乘积**

在科学计算中经常要计算矩阵的乘积。矩阵A和B可乘的条件是矩阵A的列数等于矩阵B的行数。若A是一个p×q的矩阵，B是一个q×r的矩阵，则其乘积C=AB是一个p×r的矩阵。由该公式知计算C=AB总共需要pqr次的数乘。其标准计算公式为：

现在的问题是，给定n个矩阵{A1,A2,…,An}。其中Ai与Ai+1是可乘的，i=1,2,…,n-1。要求计算出这n个矩阵的连乘积A1A2…An。

递归公式：

程序如下：

#include<stdio.h>

int main()

{

int p[101],i,j,k,r,t,n;

int m[101][101]; //为了跟讲解时保持一致数组从1开始

int s[101][101]; //记录从第i到第j个矩阵连乘的断开位置

scanf("%d",&n);

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

scanf("%d",&p[i]); //读入p[i]的值(注意：p[0]到p[n]共n+1项)

for(i=1;i<=n;i++) //初始化m[i][i]=0

m[i][i]=0;

for(r=1;r<n;r++) //r为i、j相差的值

for(i=1;i<n;i++) //i为行

{

j=i+r; //j为列

m[i][j]=m[i+1][j]+p[i-1]\*p[i]\*p[j]; //给m[i][j]赋初值

s[i][j]=i;

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

{

t=m[i][k]+m[k+1][j]+p[i-1]\*p[k]\*p[j];

if(t<m[i][j])

{

m[i][j]=t; //m[i][j]取最小值

s[i][j]=k;

}

}

}

printf("%d",m[1][n]);

}

**3． 迷宫问题**

给一个20×20的迷宫、起点坐标和终点坐标，问从起点是否能到达终点。

输入数据：’.’表示空格；’X’表示墙。

程序如下：

#include <stdio.h>

#include <math.h>

void search(int,int);

int canplace(int,int);

void readdata(); //读入数据

void printresult(); //打印结果

int a[20][20]; //a数组存放迷宫

int s,t;

int main()

{

int row, col;

readdata();

row=s/20;

col=s%20;

search(row,col); //递归搜索

printresult();

}

void search(int row, int col)

{

int r,c;

a[row][col]=1;

r=row; //左

c=col-1;

if(canplace(r,c)) //判断(r,c)位置是否已经走过

search(r,c); //递归搜索(r,c)

r=row+1; //下

c=col;

if(canplace(r,c)) //判断(r,c)位置是否已经走过

search(r,c); //递归搜索(r,c)

r=row; //右

c=col+1;

if(canplace(r,c)) //判断(r,c)位置是否已经走过

search(r,c); //递归搜索(r,c)

r=row-1; //上

c=col;

if(canplace(r,c)) //判断(r,c)位置是否已经走过

search(r,c); //递归搜索(r,c)

}

void printresult()

{

int i,j;

for(i=0;i<20;i++)

{

for(j=0;j<20;j++)

printf("%3d",a[i][j]);

printf("\n");

}

}

void readdata()

{

int i,j;

for(i=0;i<20;i++)

{

for(j=0;j<20;j++)

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

}

}

int canplace(int row, int col)

{

if(row>=0&&row<20&&col>=0&&col<20&&a[row][col]==0)

return 1;

else

return 0;

}

**实验三：搜索算法**

实验目的：熟练掌握搜索算法

**实验内容：广度优先搜索**

搜索算法的一般模式：

void search()

{

closed表初始化为空;

open表初始化为空;

起点加入到open表

while( open表非空 )

{

取open表中的一个结点u;

从open表中删除u;

u进入closed表;

for( 对扩展结点u得到的每个新结点vi )

{

if(vi是目标结点)

输出结果并返回;

if vi 的状态与closed表和open表中的结点的状态都不相同

vi进入open表;

}

}

}

搜索算法关键要解决好状态判重的问题，这样可省略closed表，一般模式可改为：

void search()

{

open表初始化为空;

起点加入到open表;

while( open表非空 )

{

取open表中的一个结点u;

从open表中删除u;

for( 对扩展结点u得到的每个新结点vi )

{

if(vi是目标结点)

输出结果并返回;

If(notused(vi))

vi进入open表;

}

}

}

**常用头文件**

#include <algorithm>

#include <bitset>

#include <cassert>

#include <cmath>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <functional>

#include <iomanip>

#include <iostream>

#include <list>

#include <map>

#include <queue>

#include <set>

#include <sstream>

#include <stack>

#include <string>

#include <vector>

using namespace std;

typedef long long LL;

#define rep(i,b) for(int i=0; i<(b); i++)

#define foreach(i,a) for(\_\_typeof((a).begin()) i=a.begin(); i!=(a).end(); ++i)

// 任意整数的读入外挂

template <typename T> inline int read(T& x) {

int c, sign = 1; x = 0;

#define S ((c=getchar())>='0'&&c<='9')

while (!(S || c == '-')) if (c == EOF) return c;

if (c == '-') sign = -1; else x = x\*10 + c - '0';

while (S) x = x\*10 + c - '0';

x \*= sign;

return 1;

}

int main() {

//ios::sync\_with\_stdio(false);

#ifndef ONLINE\_JUDGE

freopen("in.txt", "r", stdin);

#endif

return 0;

}

judging 您的程序正在被评测

Queuing 请稍等，评测系统正在测评其他人的程序，请不要过度频繁地刷新。

Accept 恭喜你，您的程序正确通过了该测试点，取得了相应的分数。

Compile Error 您的程序不能通过编译，您可以点击链接查看错误原因。请不要包含stdafx.h头文件。

Wrong Answer 您的程序输出结果与答案不符。

Time Limit Exceed 您的程序在规定时间内没有结束。

Runtime Error 您的程序发生了访问违规，即访问了不属于自己的内存。最常见的原因是数组下标越界。

Memory Limit Exceed 您程序的内存使用超过了题目的规定。

Restrict Function Call 您的程序使用了在线评测系统禁止的功能，如读写文件，运行其他程序，访问网络之类的。

Presentation Error 您的程序是基本正确的，只是输出不符合题目要求的格式，通常是行尾有多余空格，没有空行或空行太多。

Stack Overflow 您的程序堆栈溢出了。通常是您的程序陷入了无穷递归，或递归嵌套层数过多。

ACM常用术语2007-07-30 0:59算法常用术语英中对照

Approximate String Matching 模糊匹配

Arbitrary Precision Arithmetic 高精度计算

Bandwidth Reduction 带宽压缩

Bin Packing 装箱问题

Calendrical Calculations 日期

Clique 最大团

Combinatorial Problems 组合问题

Computational Geometry 计算几何

Connected Components 连通分支

Constrained and Unconstrained Optimization 最值问题

Convex Hull 凸包

Cryptography 密码

Data Structures 基本数据结构

Determinants and Permanents 行列式

Dictionaries 字典

Discrete Fourier Transform 离散Fourier变换

Drawing Graphs Nicely 图的描绘

Drawing Trees 树的描绘

Edge and Vertex Connectivity 割边/割点

Edge Coloring 边染色

Eulerian Cycle / Chinese Postman Euler回路/中国邮路

Factoring and Primality Testing 因子分解/质数判定

Feedback Edge/Vertex Set 最大无环子图

Finite State Machine Minimization 有穷自动机简化

Generating Graphs 图的生成

Generating Partitions 划分生成

Generating Permutations 排列生成

Generating Subsets 子集生成

Graph Data Structures 图

Graph Isomorphism 同构

Graph Partition 图的划分

Graph Problems — hard 图论-NP问题

Graph Problems — polynomial 图论-多项式算法

Hamiltonian Cycle Hamilton回路

Independent Set 独立集

Intersection Detection 碰撞测试

Job Scheduling 工程安排

Kd-Trees 线段树

Knapsack Problem 背包问题

Linear Programming 线性规划

Longest Common Substring 最长公共子串

Maintaining Line Arrangements 平面分割

Matching 匹配

Matrix Multiplication 矩阵乘法

Medial-Axis Transformation 中轴变换

Median and Selection 中位数

Minimum Spanning Tree 最小生成树

Minkowski Sum Minkowski和

Motion Planning 运动规划

Nearest Neighbor Search 最近点对查询

Network Flow 网络流

Numerical Problems 数值问题

Planarity Detection and Embedding 平面性检测和嵌入

Point Location 位置查询

Polygon Partitioning 多边形分割

Priority Queues 优先队列

Random Number Generation 随机数生成

Range Search 范围查询

rate of convergence 收敛速度

robustness 鲁棒性

Satisfiability 可满足性

Searching 查找

Set and String Problems 集合与串的问题

Set Cover 集合覆盖

Set Data Structures 集合

Set Packing 集合配置

Shape Similarity 相似多边形

Shortest Common Superstring 最短公共父串

Shortest Path 最短路径

Simplifying Polygons 多边形化简

Solving Linear Equations 线性方程组

Sorting 排序

Steiner Tree Steiner树

String Matching 模式匹配

Text Compression 压缩

Topological Sorting 拓扑排序

Transitive Closure and Reduction 传递闭包

Traveling Salesman Problem 旅行商问题

Triangulation 三角剖分

Vertex Coloring 点染色

Vertex Cover 点覆盖

Voronoi Diagrams Voronoi图

Calculate 计算 Contain 包含 two integers两个整数

negative integer 负整数 algebraic integer 代数整数

odd integer 奇整数 positive integer 正整数

consist of 由…组成

Per line 每行

For each case, output SUM(n) in one line 对于每组输入数据，输出一行

followed by跟随，其后，其次

You may assume the result will be in the range of 32-bit signed integer 你可以认为32位整数足以保存结果 Assume 假设, 臆断, 猜想

in the range of 在…范围内;在射程内

1. given data 已知数据

一.基本算法:

(1)枚举. (poj1753,poj2965)

(2)贪心(poj1328,poj2109,poj2586)

(3)递归和分治法.

(4)递推.

(5)构造法.(poj3295)

(6)模拟法.(poj1068,poj2632,poj1573,poj2993,poj2996)

二.图算法:

(1)图的深度优先遍历和广度优先遍历.

(2)最短路径算法(dijkstra,bellman-ford,floyd,heap+dijkstra)

(poj1860,poj3259,poj1062,poj2253,poj1125,poj2240)

(3)最小生成树算法(prim,kruskal)

(poj1789,poj2485,poj1258,poj3026)

(4)拓扑排序 (poj1094)

(5)二分图的最大匹配 (匈牙利算法) (poj3041,poj3020)

(6)最大流的增广路算法(KM算法). (poj1459,poj3436)

三.数据结构.

(1)串 (poj1035,poj3080,poj1936)

(2)排序(快排、归并排(与逆序数有关)、堆排) (poj2388,poj2299)

(3)简单并查集的应用.

(4)哈希表和二分查找等高效查找法(数的Hash,串的Hash)

(poj3349,poj3274,POJ2151,poj1840,poj2002,poj2503)

(5)哈夫曼树(poj3253)

(6)堆

(7)trie树(静态建树、动态建树) (poj2513)

四.简单搜索

(1)深度优先搜索 (poj2488,poj3083,poj3009,poj1321,poj2251)

(2)广度优先搜索(poj3278,poj1426,poj3126,poj3087.poj3414)

(3)简单搜索技巧和剪枝(poj2531,poj1416,poj2676,1129)

五.动态规划

(1)背包问题. (poj1837,poj1276)

(2)型如下表的简单DP(可参考lrj的书 page149):

1.E[j]=opt{D[i]+w(i,j)} (poj3267,poj1836,poj1260,poj2533)

2.E[i,j]=opt{D[i-1,j]+xi,D[i,j-1]+yj,D[i-1][j-1]+zij} (最长公共子序列)

(poj3176,poj1080,poj1159)

3.C[i,j]=w[i,j]+opt{C[i,k-1]+C[k,j]}.(最优二分检索树问题)

六.数学

(1)组合数学:

1.加法原理和乘法原理.

2.排列组合.

3.递推关系.

(POJ3252,poj1850,poj1019,poj1942)

(2)数论.

1.素数与整除问题

2.进制位.

3.同余模运算.

(poj2635, poj3292,poj1845,poj2115)

(3)计算方法.

1.二分法求解单调函数相关知识.(poj3273,poj3258,poj1905,poj3122)

七.计算几何学.

(1)几何公式.

(2)叉积和点积的运用(如线段相交的判定,点到线段的距离等). (poj2031,poj1039)

(3)多边型的简单算法(求面积)和相关判定(点在多边型内,多边型是否相交)

(poj1408,poj1584)

(4)凸包. (poj2187,poj1113)

中级:

一.基本算法:

(1)C++的标准模版库的应用. (poj3096,poj3007)

(2)较为复杂的模拟题的训练(poj3393,poj1472,poj3371,poj1027,poj2706)

二.图算法:

(1)差分约束系统的建立和求解. (poj1201,poj2983)

(2)最小费用最大流(poj2516,poj2516,poj2195)

(3)双连通分量(poj2942)

(4)强连通分支及其缩点.(poj2186)

(5)图的割边和割点(poj3352)

(6)最小割模型、网络流规约(poj3308, )

三.数据结构 .

(1)线段树. (poj2528,poj2828,poj2777,poj2886,poj2750)

(2)静态二叉检索树. (poj2482,poj2352)

(3)树状树组(poj1195,poj3321)

(4)RMQ. (poj3264,poj3368)

(5)并查集的高级应用. (poj1703,2492)

(6)KMP算法. (poj1961,poj2406)

四.搜索

(1)最优化剪枝和可行性剪枝

(2)搜索的技巧和优化 (poj3411,poj1724)

(3)记忆化搜索(poj3373,poj1691)

五.动态规划

(1)较为复杂的动态规划(如动态规划解特别的施行商问题等)

(poj1191,poj1054,poj3280,poj2029,poj2948,poj1925,poj3034)

(2)记录状态的动态规划. (POJ3254,poj2411,poj1185)

(3)树型动态规划(poj2057,poj1947,poj2486,poj3140)

六.数学

(1)组合数学:

1.容斥原理.

2.抽屉原理.

3.置换群与Polya定理(poj1286,poj2409,poj3270,poj1026).

4.递推关系和母函数.

(2)数学.

1.高斯消元法(poj2947,poj1487, poj2065,poj1166,poj1222)

2.概率问题. (poj3071,poj3440)

3.GCD、扩展的欧几里德(中国剩余定理) (poj3101)

(3)计算方法.

1.0 /1分数规划. (poj2976)

2.三分法求解单峰(单谷)的极值.

3.矩阵法(poj3150,poj3422,poj3070)

4.迭代逼近(poj3301)

(4)随机化算法(poj3318,poj2454)

(5)杂题.

(poj1870,poj3296,poj3286,poj1095)

七.计算几何学.

(1)坐标离散化.

(2)扫描线算法(例如求矩形的面积和周长并,常和线段树或堆一起使用).

(poj1765,poj1177,poj1151,poj3277,poj2280,poj3004)

(3)多边形的内核(半平面交)(poj3130,poj3335)

(4)几何工具的综合应用.(poj1819,poj1066,poj2043,poj3227,poj2165 ,poj3429)

高级:

一.基本算法要求:

(1)代码快速写成,精简但不失风格

(poj2525,poj1684,poj1421,poj1048,poj2050,poj3306)

(2)保证正确性和高效性. poj3434

二.图算法:

(1)度限制最小生成树和第K最短路. (poj1639)

(2)最短路,最小生成树,二分图,最大流问题的相关理论(主要是模型建立和求解)

(poj3155, poj2112,poj1966,poj3281,poj1087,poj2289,poj3216,poj2446

(3)最优比率生成树. (poj2728)

(4)最小树形图(poj3164)

(5)次小生成树.

(6)无向图、有向图的最小环

三.数据结构.

(1)trie图的建立和应用. (poj2778)

(2)LCA和RMQ问题(LCA(最近公共祖先问题) 有离线算法(并查集+dfs) 和 在线算法

(RMQ+dfs)).(poj1330)

(3)双端队列和它的应用(维护一个单调的队列,常常在动态规划中起到优化状态转移的

目的). (poj2823)

(4)左偏树(可合并堆).

(5)后缀树(非常有用的数据结构,也是赛区考题的热点).

(poj3415,poj3294)

四.搜索

(1)较麻烦的搜索题目训练(poj1069,poj3322,poj1475,poj1924,poj2049,poj3426)

(2)广搜的状态优化:利用M进制数存储状态、转化为串用hash表判重、按位压缩存储状态、双向广搜、A\*算法. (poj1768,poj1184,poj1872,poj1324,poj2046,poj1482)

(3)深搜的优化:尽量用位运算、一定要加剪枝、函数参数尽可能少、层数不易过大、可以考虑双向搜索或者是轮换搜索、IDA\*算法. (poj3131,poj2870,poj2286)

五.动态规划

(1)需要用数据结构优化的动态规划.

(poj2754,poj3378,poj3017)

(2)四边形不等式理论.

(3)较难的状态DP(poj3133)

六.数学

(1)组合数学.

1.MoBius反演(poj2888,poj2154)

2.偏序关系理论.

(2)博奕论.

1.极大极小过程(poj3317,poj1085)

2.Nim问题.

七.计算几何学.

(1)半平面求交(poj3384,poj2540)

(2)可视图的建立(poj2966)

(3)点集最小圆覆盖.

(4)对踵点(poj2079)