The Code Libraries For XJTU ACM,RpBomb

Edit By $\LaTeX\,2_\varepsilon$

by ChenKun, XJTU

XJTU'ACM RpBomb

November 9, 2007

Contents

1 High Precision **1**
- 1.1 BigInteger . . . 1
- 1.2 High Precision Div . . . 4

2 Data Structure **5**
- 2.1 Disjoint Set . . . 5
- 2.2 Range Minimum Query)(return the min element) . . . 5
- 2.3 Range Minimum Query(return the min element's pos) . . . 6
- 2.4 Leftist Tree . . . 7
- 2.5 Integerval Tree(common class) . . . 8
- 2.6 Heap-Dijkstra . . . 9
- 2.7 Binary Indexed Tree . . . 11
- 2.8 Splay Tree-Dynamic Array . . . 11
- 2.9 Interval Tree-Dynamic Ranking . . . 12
- 2.10 A Data Structure to Representations hexahedron . . . 12

3 Strings **15**
- 3.1 KMP Match . . . 15
- 3.2 Trie tree . . . 16
- 3.3 Trie Graph . . . 17
- 3.4 O(n) Suffix Array . . . 18

4 Graph Theory **23**
- 4.1 Strongly Connected Components . . . 23
- 4.2 Lowest Common Ancestor(Online,RMQ-LCA) . . . 25
- 4.3 Bipartite Maximum Match(DFS) . . . 28
- 4.4 Bipartite Optimized Match(KM Algorithm) . . . 29
- 4.5 General Maximum Match . . . 30
- 4.6 Maximum Flow(Matrix) . . . 33
- 4.7 Maximum Flow(LinkList) . . . 34
- 4.8 Maximum Flow(LinkList) . . . 35
- 4.9 Minimum Cost Maximum Flow . . . 36
- 4.10 Minimum Arborescence . . . 36
- 4.11 K minimum span tree . . . 40
- 4.12 K minimum span tree . . . 43

5 Computational Geometry 45
5.1 Geometry 2D 45
5.2 Geometry 3D 82
5.3 Convex Hull(Graham Algorithm) 82
5.4 binary div to determin whether all points is lay the same side of a line 84

6 Mathematics 91
6.1 Basic Number Theory 91
6.2 Gaussian Elimination 94
6.3 Romberg Numberical Integration 94

Chapter 1

High Precision

1.1 BigInteger

```
//writen by chenkun

#include <cstdio>
#include <cstring>

inline __int64 max(__int64 a,__int64 b) { return a>b?a:b; }
struct Bignum {
    static const __int64 MAXUNIT=1000000000;
    //数组中每个元素能表示的最大数字+1
    static const int MAXLEN=9;
    //数组每个元素能表示的最大位数
    static const int MAXDIGITS=180;
    //bignum所能表示的最大位数
    static const int MAXROOMS=MAXDIGITS/MAXLEN; //数组大小
    __int64 v[MAXROOMS];
    int len;
    bool positive;
    Bignum(__int64 init=0) {
        memset(v,0,sizeof v);
        v[0]=init;
        len=1;
        if(init>=0) positive=true; else positive=false;
    }
    Bignum(const char* str);
    Bignum operator+(const Bignum& rhs);
    Bignum& operator+=(const Bignum& rhs);
    Bignum operator*(const Bignum& rhs);
    bool operator<(const Bignum& rhs);
    bool operator==(const Bignum& rhs);
    bool operator>(const Bignum& rhs);
    void print();
};
```

```
Bignum::Bignum(const char* str) {
    __int64 t;
    char *s=(char*)((void*)str);
    if(s[0]=='-') {
        positive=false;
        s++;
    }else{
        positive=true;
    }
    if(s[0]=='+') s++;
    int l=strlen(s)/MAXLEN;
    if(strlen(s)%MAXLEN!=0) l++;
    len=l;
    t=0;
    for(int i=0;i<strlen(s)%MAXLEN;i++) {
        t=t*10+s[i]-'0';
    }
    v[len-1]=t;
    int start=len-1;
    if(strlen(s)%MAXLEN!=0) start--;
    for(int i=strlen(s)%MAXLEN;i<strlen(s);i+=MAXLEN) {
        t=0;
        for(int j=i;j<i+MAXLEN;j++) {
            t=t*10+s[j]-'0';
        }
        v[start--]=t;
    }
}

Bignum Bignum::operator +(const Bignum &rhs) {
    Bignum ret;
    int maxlen=max(len,rhs.len);
    __int64 r=0;
    for(int i=0;i<maxlen;i++) {
        ret.v[i]=v[i]+rhs.v[i]+r;
        if(ret.v[i]>=MAXUNIT) {
            r=ret.v[i]/MAXUNIT;
            ret.v[i]%=MAXUNIT;
        }else r=0;
    }
    ret.len=maxlen;
    if(r>0) ret.v[ret.len++]=r;
    return ret;
}

Bignum& Bignum::operator +=(const Bignum& rhs) {
    int maxlen=max(len,rhs.len);
    __int64 r=0;
    for(int i=0;i<maxlen;i++) {
```

```
        v[i]=v[i]+rhs.v[i]+r;
        if(v[i]>=MAXUNIT) {
            r=v[i]/MAXUNIT;
            v[i]%=MAXUNIT;
        }else r=0;
    }
    len=maxlen;
    if(r>0) v[len++]=r;
    return *this;
}

Bignum Bignum::operator *(const Bignum& rhs) {
    Bignum ret;
    __int64 r;
    for(int i=0;i<rhs.len;i++) {
        r=0;
        for(int j=0;j<len;j++) {
            ret.v[i+j]+=(rhs.v[i]*v[j]+r);
            if(ret.v[i+j]>=MAXUNIT) {
                r=ret.v[i+j]/MAXUNIT;
                ret.v[i+j]%=MAXUNIT;
            } else r=0;
        }
        if(r>0) ret.v[len+i]+=r;
    }
    ret.len=rhs.len+len;
    while(ret.v[ret.len-1]==0) ret.len--;
    if(ret.len==0) ret.len=1;
    return ret;
}


bool Bignum::operator <(const Bignum& rhs) {
    if(len>rhs.len) return false;
    if(len<rhs.len) return true;
    for(int i=len-1;i>=0;i--) {
        if(v[i]>rhs.v[i]) return false;
        if(v[i]<rhs.v[i]) return true;
    }
    return false;
}

bool Bignum::operator >(const Bignum& rhs) {
    if(len>rhs.len) return true;
    if(len<rhs.len) return false;
    for(int i=len-1;i>=0;i--) {
        if(v[i]>rhs.v[i]) return true;
        if(v[i]<rhs.v[i]) return false;
    }
    return false;
```

```
}

bool Bignum::operator ==(const Bignum& rhs) {
    if(len!=rhs.len) return false;
    for(int i=len-1;i>=0;i--) {
        if(v[i]!=rhs.v[i]) return false;
    }
    return true;
}

void Bignum::print() {
    char buf[15];
    if(!positive) putchar('-');
    printf("%I64d",v[len-1]);
    for(int i=len-2;i>=0;i--) {
        sprintf(buf,"%I64d",v[i]);
        int len=strlen(buf);
        if(len<MAXLEN)
        for(int j=0;j<MAXLEN-len;j++) putchar('0');
        printf("%I64d",v[i]);
    }
    putchar('\n');
}
```

1.2 High Precision Div

Chapter 2

Data Structure

2.1 Disjoint Set

```
int p[MAXN+1],rank[MAXN+1];

inline void makeset(int a) {
    p[a]=a;
    rank[a]=0;
}


int findset(int a) {
    if(a!=p[a]) p[a]=findset(p[a]);
    return p[a];
}

void unionset(int a,int b) {
    if(a==b) return ;
    if(rank[a]>rank[b]) {
        p[b]=a;
    }else {
        p[a]=b;
        if(rank[a]==rank[b])
            rank[b]++;
    }
}
```

2.2 Range Minimum Query)(return the min element)

```
int a[MAXLEN];
int r[20][MAXLEN];
//rmq预处理数组r[j][i]表示a(i...i+2powj-1)中的最小值
int n;        //a数组的长度
```

```
void rmq() {
    for(int i=0;i<n;i++)
        r[0][i]=a[i];
    for(int k=1;(1<<k)<=n;k++) {
        for(int i=0;i<n;i++) {
            r[k][i]=r[k-1][i];
            if(i+(1<<k-1)<n&&r[k-1][i+(1<<k-1)]<r[k][i])
                r[k][i]=r[k-1][i+(1<<k-1)];
        }
    }
}

//取得a数组在(x..y)之间的最小值
//int k=0;
//while((1<<k)<=(y-x)) k++;
//k--;
//return min(r[k][x+1],r[k][y-(1<<k)+1]);
```

2.3 Range Minimum Query(return the min element's pos)

```
const int MAX=1<<17;
//MAX is a pow of 2 and just bigger than n
int key[MAX];           //存放数据[0..n)
int n;                  //数据的个数
int pp[MAX*2];          //rmq数组

//¿号则返回最小值小标,¡则返回最大值小标
inline bool RMQcmp(int i,int j) {
        return key[i]>key[j];
}

//预处理，建树
void RMQCreate() {
        int f=MAX, t=MAX+n-1,i;
        for(i=0;i<n;i++)
                pp[i+MAX]=i;
        for(;f<t;f/=2,t/=2) {
                for(i=f;i<t;i+=2) {
                        if(!RMQcmp(pp[i],pp[i+1]))
                                pp[i/2]=pp[i];
                        else
                                pp[i/2]=pp[i+1];
                }
                if(!(t&1)) pp[t/2]=pp[t];
        }
}
```

```
//返回key[ss,ee]中的最小（大）值下标
int RMQFind(int ss,int ee) {
        int k,f,t;
        ss+=MAX,ee+=MAX;
        k=!RMQcmp(pp[ss],pp[ee])?pp[ss]:pp[ee];
        for(f=ss,t=ee;f<t;f/=2,t/=2) {
                if(!(f&1)&&f+1<t&&!RMQcmp(pp[f+1],k))
                        k=pp[f+1];
                if((t&1)&&t-1>f&&!RMQcmp(pp[t-1],k))
                        k=pp[t-1];
        }
        return k;
}
```

2.4 Leftist Tree

```
//左偏树writen by chenkun
int key[MAXN+1];              //节点的键值
int left[MAXN+1],right[MAXN+1],p[MAXN+1],dist[MAXN+1];
//左偏树上每个节点的属性
//left,right表示左儿子和右儿子
//p表示父亲节点
//dist表示距离，使得左偏树能快速合并

inline void swap(int& a,int& b) {
        int t=a;
        a=b;
        b=t;
}

//init: 初始化
//读入key[],初始化key[]=right[]=p[]=dist[]=0

//合并以a为根，以b为根的2个堆,返回新堆的根节点
int merge(int a,int b) {
        if(a==0) return b;
        if(b==0) return a;
        //如果是小堆，则改为if(key[b]¡key[a])
        if(key[b]>key[a]) swap(a,b);
        int c=merge(right[a],b);
        right[a]=c; p[c]=a;
        if(dist[right[a]]>dist[left[a]]) {
                int la=left[a];
                int ra=right[a];
                left[a]=ra;
                right[a]=la;
        }
        if(right[a]==0)
                dist[a]=0;
        else
```

```
                dist[a]=dist[right[a]]+1;
        return a;
}

//在以a为根的左偏树中插入节点x,返回根节点
int insert(int x,int a) {
        p[x]=left[x]=right[x]=dist[x]=0;
        return merge(a,x);
}

//返回节点a所在的左偏树的根节点,即返回最小值或最大值
int getroot(int a) {
        if(p[a]==0) return a;
        else return getroot(p[a]);
}

//删除节点a所在的左偏树的根节点，返回新的堆的根节点
int deletemin(int a) {
        int r=getroot(a);
        p[left[r]]=p[right[r]]=0;
        return merge(left[r],right[r]);
}

/*
//最一般的删除操作，可以在左偏树中删除任意一个已知的节点x
void delete(int x) {
        int q=p[x];
        int pp=merge(left[x],right[x]);
        p[x]=left[x]=right[x]=dist[x]=0;
        p[pp]=q;
        if(q!=0&&left[q]==x)
                left[q]=p;
        if(q!=0&&right[q]==x)
                right[q]=p;
        while(q) {
                if(dist[left[q]]<dist[right[q]])
                        swap(left[q],right[q]);
                if(dist[right[q]]+1==dist[q]) return;
                dist[q]=dist[right[q]]+1;
                pp=q;
                q=p[pp];
        }
}
*/
```

2.5 Integerval Tree(common class)

```
//线段树的通用构造方法writen by chenkun
//MAXN为线段上点的最大个数
int len=1;                    //用来记录线段个数的变量
```

```
struct segtree {
        int l,r,sum;      //l,r表示左右端点的值
        segtree *lc,*rc;
        void con( int x, int y); //在(x,y)范围内建树
        void ins( int x,int y);  //在(x,y)范围内进行插入
        void cal( int , int );
}tr[MAXN*2],*rt = &tr[0];

void segtree::con(int x,int y) {
        l = x;
        r = y;
        sum=0;
        if( x == y ){ lc = rc = 0; return; }
        int mid = ( x + y ) >> 1;
        lc = &tr[len++];
        rc = &tr[len++];
        lc -> con( x , mid );
        rc -> con( mid+1 , y );
}

void segtree::ins(int x,int y) {}

void segtree::cal(int x,int y) {}
```

2.6 Heap-Dijkstra

```
//堆，另有一个用这个堆实现的Dijkstra
// *push 在某个位置插入
// *pop删除最小值
// *top取最小值

//堆的大小，也即途中顶点数
const int HEAP_SIZE=100;
template <class COST_TYPE>
class Heap {
public:
        //保存堆，元素值为堆中该位置代表的途中顶点编号
        int data[HEAP_SIZE];
        int size;       //以用堆的大小
        int index[HEAP_SIZE];//保存图中的顶点对应于堆中的编号
        COST_TYPE cost[HEAP_SIZE];
        //cost[i]表示堆中data[i]位置的值

        void shift_up(int i) {
                int j;
                while(i>0) {
                   j=(i-1)/2;
                   if(cost[data[i]]<cost[data[j]]) {
                     swap(index[data[i]],index[data[j]]);
                     swap(data[i],data[j]);
```

```
                i=j;
            }else{
                break;
            }
        }
}

void shift_down(int i) {
        int j,k;
        while(2*i+1<size) {
            j=2*i+1;
            k=j+1;
            if((k<size)&&
            (cost[data[k]]<cost[data[j]])&&
            (cost[data[k]]<cost[data[i]])) {
                swap(index[data[k]],index[data[i]]);
                swap(data[k],data[i]);
                i=k;
            }else if(cost[data[j]]<cost[data[i]]) {
                swap(index[data[j]],index[data[i]]);
                swap(data[j],data[i]);
            }else{ break; }
        }
}

void init() {
        size=0;
        memset(index,-1,sizeof index);
        memset(cost,-1,sizeof cost);
}
bool empty() { return size==0; }
int pop() {
        int res=data[0];
        data[0]=data[size-1];
        index[data[0]]=0;
        size--;
        shift_down(0);
        return res;
}
int top() { return data[0]; }
void push(int x,COST_TYPE c) {
        if(index[x]==-1) {
                cost[x]=c;
                data[size]=x;
                index[x]=size;
                size++;
                shift_up(index[x]);
        }else if(c<cost[x]) {
                cost[x]=c;
                shift_up(index[x]);
```

```
                        shift_down(index[x]);
                }
        }
};

int Dijkstra(int G[20][20],int n,int s,int t) {
        Heap<int> heap;
        heap.init();
        heap.push(s,0);
        while(!heap.empty()) {
                int u=heap.pop();
                if(u==t) {
                  return heap.cost[t];
                }
                for(int i=0;i<n;i++) {
                  if(G[u][i]>=0) {
                   heap.push(i,heap.cost[u]+G[u][i]);
                  }
                }
        }
        return -1;
}
```

2.7 Binary Indexed Tree

```
//树状数组
//c数组下标从1开始,也可以计算sum(0)
inline int Lowbit(int t) {
    return t & ( t ^ ( t - 1 ) );
}
//在pos位置加上num
void plus(int pos , int num) {
    while(pos <= n) {
                c[pos] += num;
                pos += Lowbit(pos);
    }
}
//返回(1..end)区间所有数的和
int sum(int end) {
    if(end==0) return 0;
    int sum = 0;
    while(end > 0) {
                sum += c[end];
                end -= Lowbit(end);
    }
    return sum;
}
```

2.8 Splay Tree-Dynamic Array

2.9 Interval Tree-Dynamic Ranking

```
//线段树求区间中排序后的第k个数
// *构造函数建树
// *find函数查找出区间中比给定数值小的元素个数
struct IntervalTree {
  int A,B,Mid,N;
  int *Data;
  IntervalTree *Left,*Right;

  IntervalTree(int x,int y) {
    A=x;B=y;Mid=(x+y)>>1;N=y-x+1;
    Left=Right=NULL;
    if(x==y) {
      Data=new int[N];
      Data[0]=List[x];
      return;
    }
    Left=new IntervalTree(A,Mid);
    Right=new IntervalTree(Mid+1,B);
    Data=new int[N];
    memcpy(Data,Left->Data,sizeof(int)*Left->N);
    memcpy(Data+Left->N,Right->Data,sizeof(int)*Right->N);
    sort(Data,Data+N);
  }
  int Rank(int x,int y,int number) {
    if(x>B||y<A) return 0;
    if(x<=A&&B<=y) {
      return lower_bound(Data,Data+N,number)-Data;
    }
    return Left->Rank(x,y,number)+Right->Rank(x,y,number);
  }
};
```

2.10 A Data Structure to Representations hexahedron

```
//前面，后面，上面，下面，左面，右面一次编号为3,1,2,0,4,5
// 前
// —
// —
//左¡—¿右
// —
// 后
int back[6]={2,3,0,1,5,4};   //记录每个面的对面的编号
int myleft[6][6];
//myleft[i][j]表示当下面，前面的编号分别为i,j是，左边的编号

void pre() {
```

```
        myleft[0][3]=4;
        myleft[0][1]=5;
        myleft[0][5]=3;
        myleft[0][4]=1;
        myleft[3][0]=5;
        myleft[3][2]=4;
        myleft[3][4]=0;
        myleft[3][5]=2;
        myleft[2][3]=5;
        myleft[2][1]=4;
        myleft[2][4]=3;
        myleft[2][5]=1;
        myleft[1][2]=5;
        myleft[1][0]=4;
        myleft[1][4]=2;
        myleft[1][5]=0;
        myleft[4][3]=2;
        myleft[4][1]=0;
        myleft[4][2]=1;
        myleft[4][0]=3;
        myleft[5][3]=0;
        myleft[5][1]=2;
        myleft[5][2]=3;
        myleft[5][0]=1;
}
```

Chapter 3

Strings

3.1 KMP Match

主串：s[]，模板串：t[]。
KMP函数返回t第一次在s中出现的位置(start from 0)。
若s不包含t，则返回-1。
调用KMP函数之前要调用一次getnext函数

```
#include<stdio.h>
#include<string.h>
#define MAXs 10000
#define MAXt 1000

int next[MAXt];

void getnext(char t[])
{
    int i=0,j=-1,size=strlen(t);;
    next[0]=-1;
    while(i<size)
    {
        if(j==-1 || t[i]==t[j]) {i++; j++; next[i]=j;}
        else j=next[j];
    }
}

int KMP(char s[],char t[],int pos)
{
    int i=pos,j=0;
    int s1=strlen(s),s2=strlen(t);
    while(i<s1 && j<s2)
    {
        if(j==-1 || s[i]==t[j]) {i++; j++;}
        else j=next[j];
    }
    if(j>=s2) return i-s2;
```

```
    else return -1;
}

int main()
{
    char s[MAXs],t[MAXt];
    int pos;
    while(scanf("%s %s",s,t)==2)
    {
        getnext(t);
        pos=KMP(s,t,0);
        if(pos>=0)
        printf("t first appears in s at %d\n",pos);
        else printf("t is not included in s\n");
    }
    return 0;
}
```

3.2 Trie tree

```
//字典树
//在危险节点上用邻接表保存引起该危险的单词序号
const int MAXNODE=100000;
int child[MAXNODE][26];
bool danger[MAXNODE];
int nodes;
//邻接表
int head[MAXNODE],q[MAXNODE],next[MAXNODE],tot;
void build_trie() {
        int i,j;
        nodes=1;
        static char words[10];
        for(i=0;i<m;i++) {
                scanf("%s\n",words);
                p=1;
                for(j=0;j<strlen(words);j++) {
                        int d=words[j]-'a';
                        if(d=='?'-'a') d=26;
                        if(d=='*'-'a') d=27;
                        if(child[p][d]==0) {
                                nodes++;
                                child[p][d]=nodes;
                        }
                        p=child[p][d];
                }
                danger[p]=true;
                if(head[p]==0) {
                        tot++;
                        head[p]=tot;
                        q[tot]=i;
```

```
                        } else {
                                int t=head[p];
                                while(next[t]) t=next[t];
                                tot++;
                                next[t]=tot;
                                q[tot]=i;
                        }
                }
}
```

3.3 Trie Graph

```
//Trie图，进行多串匹配
const int maxn=3000; //节点的最大数目
const int maxchar=27;//字符集中元素的个数

int child[maxn+1][27];
bool danger[maxn+1];
int suffix[maxn+1],q[maxn+1];
int nodes,f,r,p;

//读入模式串，建立trie树
void build_trie() {
        int m;
        nodes=1;
        string words;
        cin>>m;
        for(int i=0;i<m;i++) {
                cin>>words;
                p=1;
                for(int j=0;j<words.size();j++) {
                        int d=words[j]-'a';
                        if(child[p][d]==0) {
                                nodes++;
                                child[p][d]=nodes;
                        }
                        p=child[p][d];
                        if(danger[p]) break;
                }
                danger[p]=true;
        }
}
//根据以建好的trie树建立trie图
void build_graph() {
        f=r=0;
        for(int i=0;i<26;i++) {
                if(child[1][i]==0) {
                        child[1][i]=1;
                }else {
                        r++;
```

```
                        q[r]=child[1][i];
                        suffix[child[1][i]]=1;
                    }
            }
            while(f<r) {
              f++;
              danger[q[f]]=danger[q[f]]||
                           danger[suffix[q[f]]];
              if(!danger[q[f]]) {
                for(int i=0;i<26;i++) {
                  if(child[q[f]][i]==0)
                    child[q[f]][i]=child[suffix[q[f]]][i];
                  else {
                    r++;
                    q[r]=child[q[f]][i];
                    suffix[q[r]]=child[suffix[q[f]]][i];
                  }
                }
              }
            }
        }
}
```

3.4 O(n) Suffix Array

```
#define MAXLEN 200100
int n;  //字符串长度
int SA[MAXLEN],rank[MAXLEN],h[MAXLEN],height[MAXLEN];
int r[20][MAXLEN];          //rmq预处理数组
int str[MAXLEN];            //字符串

inline bool leq(int a1,int a2,int b1,int b2) {
  return(a1 < b1 || a1 == b1 && a2 <= b2);
}
inline bool leq(int a1,int a2,int a3,int b1,int b2,int b3)
{
  return(a1 < b1 || a1 == b1 && leq(a2,a3, b2,b3));
}

static void radixPass(int* a,int* b,int* r,int n,int K) {
  int* c = new int[K + 1];
  for (int i = 0;  i <= K;  i++) c[i] = 0;
  for (int i = 0;  i < n;  i++) c[r[a[i]]]++;
  for (int i = 0, sum = 0;  i <= K;  i++) {
     int t = c[i];  c[i] = sum;  sum += t;
  }
  for (int i = 0;  i < n;  i++) b[c[r[a[i]]]++] = a[i];
  delete [] c;
}

void suffixArray(int* s, int* SA, int n, int K) {
```

```
int n0=(n+2)/3, n1=(n+1)/3, n2=n/3, n02=n0+n2;
int* s12=new int[n02 + 3];
s12[n02]=s12[n02+1]=s12[n02+2]=0;
int* SA12=new int[n02 + 3];
SA12[n02]=SA12[n02+1]=SA12[n02+2]=0;
int* s0=new int[n0];
int* SA0=new int[n0];

for (int i=0,j=0;i<n+(n0-n1);i++) if(i%3!=0) s12[j++]=i;

radixPass(s12 , SA12, s+2, n02, K);
radixPass(SA12, s12 , s+1, n02, K);
radixPass(s12 , SA12, s  , n02, K);

int name = 0, c0 = -1, c1 = -1, c2 = -1;
for(int i=0;i<n02;i++) {
if(s[SA12[i]]!=c0||s[SA12[i]+1]!=c1||s[SA12[i]+2]!=c2){
  name++;c0=s[SA12[i]];c1=s[SA12[i]+1];c2=s[SA12[i]+2];
}
if(SA12[i]%3==1){s12[SA12[i]/3]=name;} // left half
else{s12[SA12[i]/3 + n0]=name;} // right half
}

if (name < n02) {
  suffixArray(s12, SA12, n02,name);
  for (int i=0;i<n02;i++) s12[SA12[i]] = i + 1;
} else
  for(int i=0;i<n02;i++) SA12[s12[i]-1] = i;

for (int i=0,j=0;i<n02;i++) if(SA12[i]<n0)
    s0[j++]=3*SA12[i];
radixPass(s0, SA0, s, n0, K);

for(int p=0,t=n0-n1,k=0;k<n;k++) {
#define GetI() (SA12[t]<n0?SA12[t]*3+1:(SA12[t]-n0)*3+2)
  int i = GetI();
  int j = SA0[p];
  if (SA12[t]<n0?
    leq(s[i],s12[SA12[t] + n0],s[j],s12[j/3]):
    leq(s[i],s[i+1],s12[SA12[t]-n0+1],
        s[j],s[j+1],s12[j/3+n0]))
  {
    SA[k]=i;t++;
    if (t == n02) {
      for(k++;p<n0;p++,k++) SA[k]=SA0[p];
    }
  } else {
    SA[k]=j;p++;
    if (p == n0)  {
      for(k++;t<n02;t++,k++) SA[k]=GetI();
```

```
        }
      }
    }
    delete[] s12;delete[] SA12;delete[] SA0;delete[] s0;
}

void lcs() {
    for(int i=0;i<n;i++) rank[SA[i]]=i;
    for(int i=0;i<n;i++) {
      if(rank[i]==0) {
        h[i]=0;
        continue;
      }
      int j=rank[i]-1,k=rank[i],s;
      if(i==0||h[i-1]<=1)
          s=0;
      else
          s=h[i-1]-1;
      for(;SA[k]+s<n&&SA[j]+s<n;s++)
          if(str[SA[k]+s]!=str[SA[j]+s]) break;
       h[i]=s;
     }
     for(int i=0;i<n;i++)
        height[rank[i]]=h[i];
}

void rmq() {
    for(int i=0;i<n;i++) r[0][i]=height[i];
    for(int k=1;(1<<k)<=n;k++) {
      for(int i=0;i<n;i++) {
          r[k][i]=r[k-1][i];
          if(i+(1<<k-1)<n&&r[k-1][i+(1<<k-1)]<r[k][i])
            r[k][i]=r[k-1][i+(1<<k-1)];
      }
    }
}

int asklcs(int x,int y) {
    if(x>y) swap(x,y);
    int k=0;
    while((1<<k)<=(y-x)) k++;
    k--;
    return min(r[k][x+1],r[k][y-(1<<k)+1]);
}


int main() {
    //读入str,长度为n,令SA[i]=str[i]
    str[n]=str[n+1]=str[n+2]=SA[n]=SA[n+1]=SA[n+2]=0;
    suffixArray(str,SA,n,b);        //其中b为str数组中最大的数
```

```
    lcs();
    rmq();
    //asklcs(i,j) //求(i..j)的最长公共前缀
}
```

Chapter 4

Graph Theory

4.1 Strongly Connected Components

```
#include <stdio.h>
#include <string.h>
#define G_size 100000
#define V_size 11000

typedef struct Graph
{
    int id;
    int next;
} Graph;

typedef struct Edge
{
    int s, e;
} Edge;

Edge E[G_size];
Graph GA[G_size], GT[G_size];
int N, M;           //点数，边数
int G_end;
int order[V_size],id[V_size],vis[V_size];
int cnt;

void Insert(int s, int e) //建立原图和逆图
{
    int p;
    p = s;
    while (GA[p].next)
        p = GA[p].next;
    GA[G_end].id = e;
    GA[p].next = G_end;
```

```
    p = e;
    while (GT[p].next)
        p = GT[p].next;
    GT[G_end].id = s;
    GT[p].next = G_end;

    G_end++;
}

void DFST(int x) //对逆图进行搜索
{
    int p, q;
    vis[x] = 1;
    p = GT[x].next;
    while (p)
    {
        q = GT[p].id;
        if (!vis[q])
            DFST(q);
        p = GT[p].next;
    }
    order[cnt++] = x;
}

void DFSA(int x) //对原图进行搜索
{
    int p, q;
    vis[x] = 1;
    id[x] = cnt;
    p = GA[x].next;
    while (p)
    {
        q = GA[p].id;
        if (!vis[q])
            DFSA(q);
        p = GA[p].next;
    }
}

void Solve() //主要过程
{
    int s, e;
    int i;

    memset(GA, 0, sizeof(GA));
    memset(GT, 0, sizeof(GT));
    memset(E, 0, sizeof(E));
    G_end = N + 1;

    for (i = 0; i < M; i++)
```

```
    {
        scanf("%d %d", &s, &e);
        E[i].s = s - 1;
        E[i].e = e - 1;
        Insert(s - 1, e - 1);
    }

    memset(vis, 0, sizeof(vis));
    cnt = 0;
    for (i = 0; i < N; i++)
    {
        if (!vis[i])
        {
            DFST(i);
        }
    }

    memset(vis, 0, sizeof(vis));
    cnt = 0;
    for (i = N - 1; i >= 0; i--)
    {
        if (!vis[order[i]])
        {
            DFSA(order[i]);
            cnt++;
        }
    }
    //id[v]表示v点所在强连同分量的编号
}
```

4.2 Lowest Common Ancestor(Online,RMQ-LCA)

```
/*rmq-lca方法求最小公共祖先
  writen by chenkun
*/
#include <cstdio>
#include <cstring>

//树中节点最大数
const int MAXN=40000;
//使2 pow MAX大于MAXN即可
const int MAX=1<<17;

//下面为rmq数组
int key[MAX],seq[MAX];
int n;  //特殊用途，中序遍历的计数
int pp[MAX*2];

int nn,m;//顶点数，边数nn-1==m
//下面为邻接表
```

```
int head[MAXN+1],next[2*MAXN+10],
    q[2*MAXN+10],cost[2*MAXN+10],tot;
int first[MAXN+1];
bool vis[MAXN+1];

inline void swap(int& a,int& b) {
        int t=a;a=b;b=t;
}
//加入边a-b
void insert(int a,int b) {
        int t;
        tot++;
        if(head[a]==0)
                head[a]=tot;
        else {
                t=head[a];
                head[a]=tot;
                next[tot]=t;
        }
        q[tot]=b;
        tot++;
        if(head[b]==0)
                head[b]=tot;
        else {
                t=head[b];
                head[b]=tot;
                next[tot]=t;
        }
        q[tot]=a;
}

void init() {
        int a,b,len;
        char c;
        scanf("%d %d",&nn,&m);
        for(int i=0;i<m;i++) {
                scanf("%d %d",&a,&b);
                insert(a,b);
        }
}

//根据邻接表dfs构造树
void build_tree(int u,int level) {
        vis[u]=true;
        key[n]=level;
        seq[n]=u;
        first[u]=n;
        n++;
        int h=head[u];
        while(h) {
```

```
                if(!vis[q[h]]) {
                        build_tree(q[h],level+1);
                        key[n]=level;
                        seq[n]=u;
                        n++;
                }
                h=next[h];
        }

}
inline bool RMQcmp(int i,int j) {
        return key[i]>key[j];
}

void RMQCreate() {
        int f=MAX,t=MAX+n-1,i;
        for(i=0;i<n;i++)
                pp[i+MAX]=i;
        for(;f<t;f/=2,t/=2) {
                for(i=f;i<t;i+=2) {
                        if(!RMQcmp(pp[i],pp[i+1]))
                                pp[i/2]=pp[i];
                        else
                                pp[i/2]=pp[i+1];
                }
                if(!(t&1)) pp[t/2]=pp[t];
        }
}

int RMQFind(int ss,int ee) {
        int k,f,t;
        ss+=MAX,ee+=MAX;
        k=!RMQcmp(pp[ss],pp[ee])?pp[ss]:pp[ee];
        for(f=ss,t=ee;f<t;f/=2,t/=2) {
                if(!(f&1)&&f+1<t&&!RMQcmp(pp[f+1],k))
                        k=pp[f+1];
                if((t&1)&&t-1>f&&!RMQcmp(pp[t-1],k))
                        k=pp[t-1];
        }
        return k;
}

int main() {
        int cas,s,e,fs,fe,lca;
        init();
        build_tree(1,0);
        RMQCreate();
        //查询s点和e点的lca
        scanf("%d %d",&s,&e);
        fs=first[s],fe=first[e];
```

```
        if(fs>fe) swap(fs,fe);
        lca=seq[RMQFind(fs,fe)];
        return 0;
}
```

4.3 Bipartite Maximum Match(DFS)

```
const int maxm=200; //右图最大顶点数
const int maxn=200; //左图最大顶点数
bool g[maxn][maxm]; //邻接矩阵
int match[maxm];
bool us[maxm];

//dfs寻找增广路
bool go(int v) {
  int i;
  for(i=0;i<m;i++) {
    if(us[i]) continue;
    if(g[v][i]) {
        if(match[i]==-1) {
          us[i]=1;
          match[i]=v;
          return true;
        }
    }
  }
  for(i=0;i<m;i++) {
    if(us[i]) continue;
    if(g[v][i]) {
        us[i]=1;
        if(go(match[i])) {
          match[i]=v;
          return true;
        }
    }
  }
  return false;
}
//返回最大匹配数
int matcher() {
  int i,res=0;
  memset(match,255,sizeof match);
  for(i=0;i<n;i++) {
    memset(us,0,sizeof us);
    if(go(i)) ++res;
  }
  return res;
}
```

4.4 Bipartite Optimized Match(KM Algorithm)

```
/*
Hungary 算法求二部图的最优匹配
输入: C–二部图的利润矩阵,C[x][y]¿=0表示x和y匹配的利润
 若x和y之间没有边则C[x][y]=0
 nx–二部图的节点集合X中的元素数目
 nu–二部图的节点集合Y中的元素数目
输出: X,Y–最优匹配X,Y集合众节点所匹配的节点的id,-1表示
 该节点没有被匹配,若C[i][X[i]]=0,则最终结果可以删除
 这一匹配,因为有无这一匹配对最大利润没有影响
注意:输入必须保证C[x][y]¿=0
*/
typedef int Graph[100][100];
typedef int Path[100];
void optmatch(Graph C,int nx,int ny,Path X,Path Y) {
        Path Lx,Ly,Q,prev;
        int i,j,k,s,head,tail;
        //要保证Y中的节点数目比X中的多
        if(ny<nx) ny=nx;
        for(i=0;i<nx;i++) {
                Lx[i]=Ly[i]=0;
                for(j=0;j<ny;j++)
                        Lx[i]=max(Lx[i],C[i][j]);
        }
        memset(X,-1,sizeof Path);
        memset(Y,-1,sizeof Path);
        i=0;
        while(i<nx) {
          memset(prev,-1,sizeof Path);
          for(Q[0]=i,head=0,tail=1;head<tail&&X[i]<0;head++) {
            s=Q[head];
            for(j=0;j<ny&&X[i]<0;j++) {
                if(Lx[s]+Ly[j]>C[s][j]||prev[j]>=0) continue;
                Q[tail++]=Y[j];
                prev[j]=s;
                if(Y[j]<0) {
                  while(j>=0) {
                        s=prev[j];
                        Y[j]=s;
                        k=X[s];
                        X[s]=j;
                        j=k;
                  }
                }
            }
          }
          if(X[i]>=0) {i++;}
          else{
```

```
                        k=2147483647;
                        for(head=0;head<tail;head++) {
                            s=Q[head];
                            for(j=0;j<ny;j++) {
                                if(prev[j]==-1) {
                                    k=min(k,Lx[s]+Ly[j]-C[s][j]);
                                }
                            }
                        }
                        for(j=0;j<tail;j++) Lx[Q[j]]-=k;
                        for(j=0;j<ny;j++)
                            if(prev[j]>=0) Ly[j]+=k;
                }
            }
}
```

4.5 General Maximum Match

```
/*
Method:Maimum Cardinality Mathcing Problem in General Graph
By Edmonds Blossom-Contraction Algorithm
Detail:
Augmenting Path Theorem:Iff there is no augmenting path,the
matching is maximal.An augmenting path is an alternating path
which is started and ended with unmatched nodes;

Use BFS to find augmenting path.
If there is an alternating cycle(must have an odd number of nodes),
this cycle called "blossom" must be contracted as a new node

Notice to maintain the father pointers of the nodes in blossoms.
*/
#include <iostream>
#include <algorithm>
using namespace std;
#define SET0(x) memset(x,0,sizeof(x))
#define SET1(x) memset(x,0xff,sizeof(x))

const int MAXN=250;
typedef int Graph[MAXN][MAXN];
typedef int Path[MAXN];

int n,head,tail,start,finish,newbase;
Graph g;Path mat;Path Q,inQ,inP,inB,father,base;
/*QUeue,inQueue,inPath,in Blossom,*/
void createGraph() {
        int u,v;
        SET0(g);
        scanf("%d",&n);
        while(2==scanf("%d%d",&u,&v))
```

```
                g[u][v]=g[v][u]=true;
}
inline void push(int u) {Q[tail++]=u;inQ[u]=true;}
inline int pop() {return Q[head++];}

int findCommonAncestor(int u,int v) {
        SET0(inP);
        while(1) {
                u=base[u];
                inP[u]=true;
                if(u==start) break;
                u=father[mat[u]];
        }
        while(1) {
                v=base[v];
                if(inP[v]) break;
                v=father[mat[v]];
        }
        return v;
}
void resetTrace(int u) {
        int v;
        while(base[u]!=newbase) {
                v=mat[u];
                inB[base[u]]=inB[base[v]]=true;
                u=father[v];
                if(base[u]!=newbase) father[u]=v;
        }
}

void blossomContract(int u,int v) {
        newbase=findCommonAncestor(u,v);
        SET0(inB);
        resetTrace(u);resetTrace(v);
        if(base[u]!=newbase) father[u]=v;
        if(base[v]!=newbase) father[v]=u;
        for(u=1;u<=n;u++)
                if(inB[base[u]]){
                        base[u]=newbase;
                        if(!inQ[u]) push(u);
                }
}
void findAugmentingPath() {
        int u,v;
        SET0(inQ);SET0(father);
        for(u=1;u<=n;u++) base[u]=u;
        tail=head=1;
        push(start);
        finish=0;
        while(head<tail) {
```

```
                u=pop();
                for(v=1;v<=n;v++)
                        if(g[u][v]&&base[u]!=base[v]&&mat[u]!=v)
                                if(v==start||mat[v]>0&&father[mat[v]]>0)
                                        blossomContract(u,v);
                                else if(father[v]==0) {
                                        father[v]=u;
                                        if(mat[v]>0) push(mat[v]);
                                        else {finish=v;return;}
                                }
        }
}
void augmentPath() {
        int u,v,w;
        u=finish;
        while(u>0) {
                v=father[u];
                w=mat[v];
                mat[v]=u;
                mat[u]=v;
                u=w;
        }
}
void edmonds() {
        int u;
        SET0(mat);
        for(u=1;u<=n;u++)
                if(mat[u]==0) {
                        start=u;
                        findAugmentingPath();
                        if(finish>0) augmentPath();
                }
}
void printMatch() {
        int u;
        int count=0;
        for(u=1;u<=n;u++)
                if(mat[u]>0) count++;
        printf(”%dn”,count);
        for(u=1;u<=n;u++)
                if(u<mat[u])
                        printf(”%d %d\n”,u,mat[u]);
}
int main() {
        createGraph();
        edmonds();
        printMatch();
        return 0;
}
```

4.6 Maximum Flow(Matrix)

```
//方法，将流网络读入矩阵co后(co[i][j]=-co[j][i]),
//while(bfs());即可得到最大流
//最大流的结果存放在flow[][]中
//writen by chenkun
const int maxn=110; //顶点最大数

struct tnode {
  int pre,now,mincost;
  tnode() {}
  tnode(int P,int N,int C):
    pre(P),now(N),mincost(C) {}
};

int n,m,s,t;      //s,t分别为源点，汇点

int flow[maxn][maxn];
int co[maxn][maxn];
tnode queue[maxn];
bool vis[maxn];

//bfs求增广路并进行增流
bool bfs() {
  int f=0,r=0,h,nowf;
  tnode tt;
  memset(vis,false,sizeof vis);
  queue[r++]=tnode(-1,s,9999999);
  vis[s]=true;
  while(f<r) {
    tt=queue[f];
    if(tt.now==t) break;
    for(int i=0;i<=n+1;i++) {
        nowf=co[tt.now][i]-flow[tt.now][i];
        if (!vis[i]&&nowf>0) {
          queue[r++]=tnode(f,i,min(tt.mincost,nowf));
          vis[i]=true;
        }
    }
    f++;
   }
   if(tt.now!=t) return false;
   int minp=tt.mincost;
   nowf=tt.pre;
   while(nowf!=-1) {
     flow[queue[nowf].now][tt.now]+=minp;
     flow[tt.now][queue[nowf].now]
       =-flow[queue[nowf].now][tt.now];
     tt=queue[nowf];
```

```
        nowf=tt.pre;
    }
    return true;
}
```

4.7 Maximum Flow(LinkList)

```
//链表最大流
//使用方法:while(Find_Path())Update()
//即可得到最大流的数值在Flow中
#include <cstdio>
#include <vector>
using namespace std;

const int MaxN=100,MaxM=1000,INF=(1<<30);
struct Node{
        int Data,C,F,Op;
};
vector<int> G[MaxN];
Node E[MaxM];

int S,T,M; //S,T为源点和汇点,M边数
int Flow;
int Q[MaxN],vf[MaxN],pre[MaxN];

void Add(int u,int v,int C) {
        E[M].Data=v;
        E[M].C=C;E[M].F=0;E[M].Op=M+1;
        G[u].push_back(M);M++;
        E[M].Data=u;
        E[M].C=0;E[M].F=0;E[M].Op=M-1;
        G[v].push_back(M);
        M++;
}
inline int fmin(int u,int v) {return (u<v)?u:v;}
bool Find_Path() {
        int F=0,R=0,u,v,j;
        memset(pre,-1,sizeof pre);
        Q[0]=S;pre[S]=-2;vf[S]=INF;
        while(F<=R) {
                u=Q[F++];
                for(j=0;j<G[u].size();j++) {
                        v=G[u][j];
                        if(E[v].C>E[v].F&&pre[E[v].Data]==-1) {
                                Q[++R]=E[v].Data;
                                pre[E[v].Data]=E[v].Op;
                                vf[E[v].Data]=fmin(E[v].C-E[v].F,vf[u]);
                                if(E[v].Data==T) return true;
                        }
                }
```

```
        }
        return false;
}
void Update() {
        int cnt=T;
        while(pre[cnt]!=-2) {
                E[pre[cnt]].F-=vf[T];
                E[E[pre[cnt]].Op].F+=vf[T];
                cnt=E[pre[cnt]].Data;
        }
        Flow+=vf[T];
}
```

4.8 Maximum Flow(LinkList)

```
const int maxn=100; //顶点最大数

struct tnode {
        int pre,now,mincost;
        tnode() {}
        tnode(int P,int N,int C):pre(P),now(N),mincost(C) {}
};

int n,m,s,t;     //s,t分别为源点和汇点
int flow[maxn+1][maxn+1]; //流量
tnode queue[maxlen];
bool vis[maxn+1];

//邻接表,c[]为容量
int head[maxn],next[maxn*maxn],q[maxn],c[maxn],tot;

//bfs求增广路并进行增流
bool bfs() {
        int f=0,r=0,h,nowf;
        tnode tt;
        memset(vis,false,sizeof vis);
        queue[r++]=tnode(-1,s,9999999);
        vis[s]=true;
        while(f<r) {
                tt=queue[f];
                h=head[tt.now];
                if(tt.now==t) break;
                while(h) {
                        nowf=c[h]-flow[tt.now][q[h]];
                        if(!vis[q[h]]&&nowf>0) {
                          queue[r++]=tnode(f,q[h],min(tt.mincost,nowf));
                          vis[q[h]]=true;
                        }
                        h=next[h];
                }
```

```
                f++;
            }
            if(tt.now!=t) return false;
            int minp=tt.mincost;
            nowf=tt.pre;
            while(nowf!=-1) {
                flow[queue[nowf].now][tt.now]+=minp;
                flow[tt.now][queue[nowf].now]=-flow[queue[nowf].now][tt.now];
                tt=queue[nowf];
                nowf=tt.pre;
            }
            return true;
}
```

4.9 Minimum Cost Maximum Flow

```
#include <iostream>
using namespace std;
#define MAXN 100
#define inf 1.0e10
int min_cost_max_flow(int n,int mat[][MAXN],int cost[][MAXN],
  int source,int sink,int flow[][MAXN],int& netcost) {
  int pre[MAXN],min[MAXN],d[MAXN],i,j,t,tag;
  if(source==sink) return inf;
  for(i=0;i<n;i++)
    for(j=0;j<n;flow[i][j++]=0);
  for(netcost=0;;) {
    for(i=0;i<n;i++) pre[i]=0,min[i]=inf;
    for(pre[source]=source+1,min[source]=0,d[source]=inf,tag=1;tag;)
      for(tag=t=0;t<n;t++)
          if(d[t]) for(i=0;i<n;i++)
              if(j=mat[t][i]-flow[t][i]&&min[t]+cost[t][i]<min[i])
              tag=1,min[i]=min[t]+cost[t][i],pre[i]=t+1,d[i]=d[t]<j?d[t]:j;
              else if(j=flow[i][t]&&min[t]-cost[i][t]<min[i])
              tag=1,min[i]=min[t]-cost[i][t],pre[i]=-t-1,d[i]=d[t]<j?d[t]:j
    if(!pre[sink]) break;
    for(netcost+=min[sink]*d[i=sink];i!=source;)
        if(pre[i]>0)
          flow[pre[i]-1][i]+=d[sink],i=pre[i]-1;
        else
          flow[i][-pre[i]-1]-=d[sink],i=-pre[i]-1;
        }
    for(j=i=0;j<n;j+=flow[source][i++]);
    return j;
}
```

4.10 Minimum Arborescence

```
//最小树形图，邻接表writen by chenkun
```

```
#define NOEDGE 999999
const unsigned int maxm=40000;          //最大边数
const int maxn=1000;                    //最大点数
bool vis[maxn];
int N,M;                                //顶点数，边数
int head[maxn],next[maxm],q[maxm],tot;    //正向边的邻接表存储
int head2[maxn],next2[maxm],q2[maxm],tot2;   //反向边的邻接表
存储
int G[maxn][maxn];                      //邻接矩阵
int res;                                //最优值

void dfs(int v){
        vis[v]=true;
        int i=head[v];
        while(i) {
                if((!vis[q[i]]))
                        dfs(q[i]);
                i=next[i];
        }
}

//是否存在可行解
bool possible(){
        memset(vis,0,sizeof(vis));
        dfs(0);
        for(int i=1;i<N;++i)
                if(!vis[i])
                        return false;
        return true;
}

//求最小树形图
int pre[maxn];
bool del[maxn];

//求最小树形图，最后结果在res中（可以输出方案)
void solve(){
  memset(del,0,sizeof(del));
  for(;;){
    int i;
    for(i=1;i<N;i++) {
        if(del[i]) continue;
        pre[i]=i;
        G[i][i]=NOEDGE;
        int j=head2[i];
        while(j) {
                if(!del[q2[j]]) {
                        if(G[pre[i]][i]>G[q2[j]][i])
                                pre[i]=q2[j];
                }
```

```
                j=next2[j];
            }
        }
        for(i=1;i<N;i++) {
            if(del[i]) continue;
            int j=i;
            memset(vis,0,sizeof vis);
            //寻找环
            while(!vis[j]&&j!=0) {
                vis[j]=true;
                j=pre[j];
            }
            //如果没有环，则退出
            if(j==0) continue;
            i=j;
            res+=G[pre[i]][i];
            //删除环
            for(j=pre[i];j!=i;j=pre[j]) {
                res+=G[pre[j]][j];
                del[j]=true;
            }
            j=head2[i];
            //更新相应的边
            while(j) {
                if(!del[q2[j]]) {
                    G[q2[j]][i]-=G[pre[i]][i];
                }
                j=next2[j];
            }
            for(j=pre[i];j!=i;j=pre[j]) {
                int k=head[j];
                while(k) {
                    if(q[k]!=0&&!del[q[k]]&&k!=i) {
                        if(G[i][q[k]]>G[j][q[k]]) {
                            //在邻接表中增加边
                            if(G[i][q[k]]==NOEDGE) {
                                tot++;tot2++;
                                int ttt=head[i];
                                head[i]=tot;
                                next[tot]=ttt;
                                q[tot]=q[k];
                                ttt=head2[q[k]];
                                head2[q[k]]=tot2;
                                next2[tot2]=ttt;
                                q2[tot2]=i;
                            }
                            G[i][q[k]]=G[j][q[k]];
                        }
                    }
                    k=next[k];
```

```
                }
                k=head2[j];
                while(k) {
                        if(!del[q2[k]]) {
                                if(G[q2[k]][i]>G[q2[k]][j]-G[pre[j]][j]) {
                                        if(G[q2[k]][i]==NOEDGE) {
                                                tot++; tot2++;
                                                int ttt=head[q2[k]];
                                                head[q2[k]]=tot;
                                                next[tot]=ttt;
                                                q[tot]=i;
                                                ttt=head2[i];
                                                head2[i]=tot2;
                                                next2[tot2]=ttt;
                                                q2[tot2]=q2[k];
                                        }
                                        G[q2[k]][i]=G[q2[k]][j]-G[pre[j]][j];
                                }
                        }
                        k=next2[k];
                }
        }
        for(j=pre[i];j!=i;j=pre[j]) {
                del[j]=true;
        }
        break;
    }
    if(i>=N) {
        for(int k=1;k<N;k++) {
                if(del[k]) continue;
                res+=G[pre[k]][k];
        }
        break;
    }
  }
}

void init() {
        tot=tot2=0;
        int a,b,c;
        scanf("%d %d",&N,&M);
        memset(head,0,sizeof head);
        memset(next,0,sizeof next);
        memset(head2,0,sizeof head2);
        memset(next2,0,sizeof next2);
        for(int i=0;i<N;i++)
                for(int j=i;j<N;j++)
                        G[i][j]=G[j][i]=NOEDGE;
        for(int i=0;i<M;i++) {
                scanf("%d %d %d",&a,&b,&c);
```

```
                    tot++;
                    tot2++;
                    if(head[a]==0) {
                            head[a]=tot;
                            q[tot]=b;
                    } else {
                            int j=head[a];
                            while(next[j]) j=next[j];
                            next[j]=tot;
                            q[tot]=b;
                    }
                    if(head2[b]==0) {
                            head2[b]=tot2;
                            q2[tot2]=a;
                    } else {
                            int j=head2[b];
                            while(next2[j]) j=next2[j];
                            next2[j]=tot2;
                            q2[tot2]=a;
                    }
                    G[a][b]=c;
            }
}
```

4.11 K minimum span tree

```
//K限度生成树writen by chenkun
const int maxn=50;
//G原图G2去掉限度节点后的图
int G[maxn][maxn],G2[maxn][maxn];
//label对G2进行编号,统一连同分量的节点标号相同;tot[i]表示标号为i的节点
个数
int label[maxn],tot[maxn];
//u:dfs编号时的标记;G3:最小生成树
bool u[maxn],G3[maxn][maxn],b[maxn],u2[maxn];
//id表示当前强连同分量的标号值
int k,root,m,minid,maxid,id,maxdel,me1,me2;

//dfs求强连同分量并标号
void dfs(int v) {
        u[v]=true;
        tot[id]++;
        label[v]=id;
        for(int i=minid;i<=maxid;i++)
                if(G2[v][i]>0&&!u[i]) {
                        dfs(i);
                }
}

//计算从root-¿v出发的圈中除root¡-¿v外的边的最大值maxdel
```

```
//me1,me2记录最大边的端点
bool circlemax(int v,int j) {
        u2[v]=true;
        for(int i=minid;i<=maxid;i++) {
                if(!G3[v][i]) continue;
                if(u2[i]) continue;
                if(i==root) {
                        maxdel=j;
                        return true;
                }
                if(j<G[v][i]) {
                        me1=v;me2=i;
                }
                if(circlemax(i,max(j,G[v][i]))) return true;
        }
        return false;
}

//对k限度生成树进行一次换边操作
int addmstedge() {
        int bestdel=-1,bestret=999999,me1b,me2b;
        int addv;
        for(int i=minid;i<=maxid;i++) {
                if(G[root][i]==0) continue;
                if(G3[root][i]==true) continue;
                memset(u2,false,sizeof u2);
                circlemax(i,0);
                if(G[root][i]-maxdel<bestret) {
                        bestret=G[root][i]-maxdel;
                        addv=i;
                        me1b=me1,me2b=me2;
                }
        }
        G3[root][addv]=G3[addv][root]=true;
        G3[me1b][me2b]=G3[me2b][me1b]=false;

        if(bestret==999999) return 0;else return bestret;
}

//求连同分量并标号
void make_cc() {
        for(int i=minid;i<=maxid;i++) {
                for(int j=minid;j<=maxid;j++) {
                        G2[i][j]=G[i][j];
                        if(i==root||j==root) G2[i][j]=0;
                }
        }
        memset(u,false,sizeof u);
        memset(tot,0,sizeof tot);
        id=0;u[root]=true;
```

```
        for(int i=minid;i<=maxid;i++) {
                if(i!=root)
                        if(!u[i]) {
                                id++;
                                dfs(i);
                        }
        }
}

//计算标号为id的连同分量的最小生成树
int calc_mst(int id) {
        int sp,ep,sum=0;
        for(int i=minid;i<=maxid;i++) {
                if(label[i]==id) {
                        b[i]=true;
                        break;
                }
        }
        for(int i=1;i<tot[id];i++) {
                int min=999999;
                for(int j=minid;j<=maxid;j++) {
                        if(label[j]!=id) continue;
                        if(!b[j]) continue;
                        for(int kk=minid;kk<=maxid;kk++) {
                                if(label[kk]!=id) continue;
                                if(b[kk]) continue;
                                if(G2[j][kk]==0) continue;
                                if(G2[j][kk]<min) {
                                        min=G2[j][kk];
                                        sp=j;
                                        ep=kk;
                                }
                        }
                }
                sum+=min;
                b[ep]=true;
                G3[sp][ep]=G3[ep][sp]=true;
        }
        return sum;
}

int solve() {
        int sum=0;
        make_cc();
        memset(b,false,sizeof b);
        memset(G3,false,sizeof G3);
        for(int i=1;i<=id;i++)
                sum+=calc_mst(i);
        for(int i=1;i<=id;i++) {
                int min=999999;
```

```
                int nowid;
                for(int j=minid;j<=maxid;j++) {
                        if(label[j]!=i) continue;
                        if(G[root][j]==0) continue;
                        if(G[root][j]<min) {
                                min=G[root][j];
                                nowid=j;
                        }
                }
                sum+=min;
                G3[root][nowid]=G3[nowid][root]=true;
        }
        int bestans=sum,ret=bestans;
        for(int nowx=id;nowx<k;nowx++) {
                bestans=bestans+addmstedge();
                ret=min(bestans,ret);
        }
        return ret;
}
```

4.12 K minimum span tree

```
//最优比率生成树快速迭代writen by chenkun
double l[maxn+1][maxn+1],G[maxn+1][maxn+1];//l每条边的花费
int c[maxn+1][maxn+1];                     //c每条边的收益

double prim(double r) {
        double ret=0;
        memset(G,0,sizeof G);
        for(int i=1;i<=n;i++) {
                for(int j=i+1;j<=n;j++) {
                  G[i][j]=G[j][i]=c[i][j]-r*l[i][j];
                }
        }
        lc=0;
        rc=0;
        memset(u,false,sizeof u);
        u[1]=true;
        for(int i=2;i<=n;i++) {
                a[i]=G[1][i];
                cb[i]=c[1][i];
                lb[i]=l[1][i];
        }
        for(int i=2;i<=n;i++) {
                int id;
                double mind=999999999;
                for(int j=1;j<=n;j++) {
                        if(!u[j]&&a[j]<mind) {
                                id=j;
                                mind=a[j];
```

```
                }
            }
            u[id]=true;
            ret+=mind;
            lc+=cb[id];
            rc+=lb[id];
            for(int j=1;j<=n;j++) {
                if(!u[j]&&G[id][j]<a[j]) {
                    a[j]=G[id][j];
                    cb[j]=c[id][j];
                    lb[j]=l[id][j];
                }
            }
        }
        return ret;
}

double solve() {
        double r=30,rb=0;
        while(fabs(prim(r))>EPS&&fabs(r-rb)>EPS) {
            rb=r;
            r=lc/rc;        //迭代r=ax/bx
        }
        return r;
}
```

Chapter 5

Computational Geometry

5.1 Geometry 2D

```
//geom2d.cpp
//二维计算几何代码库

//By starfish(2003)
//Enhanced by phoenixinter(2004,2005)

#include<stdio.h>
#include<math.h>
#include<algorithm>
using namespace std;

#define INF 1e10       //无穷大
#define EPS 1e-8       //计算精度
#define PI acos(-1.0)
#define MAX_N 100      //多边形的最多顶点数目

/*=================================
            数据结构定义
==================================*/

struct Point
{
    double x,y;
    Point(double x0=0,double y0=0):x(x0),y(y0){}
};

struct Line
{
    Point p1,p2;
};
```

```
/*=========================================
                浮点数处理
=========================================*/
#define abs(x) ((x)>=0?x:-(x))
#define min(x,y) ((x)<(y)?(x):(y))
#define max(x,y) ((x)>(y)?(x):(y))
#define eq(x,y) (fabs((x)-(y))<EPS)
#define leq(x,y) ((x)<=(y)+EPS)
#define geq(x,y) ((x)+EPS>=(y))
#define zero(x) (((x)>0?(x):-(x))<EPS)        //判定x是否为[-EPS,EPS]
#define _sign(x) ((x)>EPS?1:((x)<-EPS?2:0)) //判定x的符号，返
回1,2,0.
/*
 注意：
 如果是一个很小的负的浮点数，
 保留有效位数输出的时候会出现-0.000这样的形式，
 前面多了一个负号，这就会导致错误！！！！！！
 因此在输出浮点数时，一定要输出fix(x)!
*/
#define fix(x) (fabs(x)<EPS?0:x)

/*=========================================
               矢量基本操作
=========================================*/
bool operator<(Point p1,Point p2)
{
    if(!eq(p1.x,p2.x)) return p1.x<p2.x;
    else return p1.y<p2.y;
}

//计算p1-p2
Point operator-(Point p1,Point p2)
{
    return Point(p1.x-p2.x,p1.y-p2.y);
}

//计算叉积p1*p2
double operator*(Point p1,Point p2)
{
    return (p1.x*p2.y-p1.y*p2.x);
}

//计算叉积(p1-p0)*(p2-p0)
double times(Point p0,Point p1,Point p2)
{
    return (p1.x-p0.x)*(p2.y-p0.y)-(p1.y-p0.y)*(p2.x-p0.x);
}

//计算点积p1.p2
double operator&(Point p1,Point p2)
```

```
{
    return (p1.x*p2.x+p1.y*p2.y);
}

//求矢量u的模
inline double norm(Point u)
{
    return sqrt(u.x*u.x+u.y*u.y);
}

/*
 旋转矢量
 输入：p 被旋转的矢量
  angle 旋转角度，用弧度表示，
  ¿0表示逆时针旋转，
  ¡0表示顺时针旋转
 输出：旋转后得到的矢量
 调用：无
*/
Point Rotate(Point p,double angle)
{
    Point res;
    res.x=p.x*cos(angle)-p.y*sin(angle);
    res.y=p.x*sin(angle)+p.y*cos(angle);
    return res;
}

/*
   矢量V以P为顶点逆时针旋转angle并放大scale倍(By phoenixinter)
*/
Point Rotate(Point v,Point p,double angle,double scale)
{
    Point ret=p;
    v.x-=p.x;v.y-=p.y;
    p.x=scale*cos(angle);
    p.y=scale*sin(angle);
        ret.x+=v.x*p.x-v.y*p.y;
        ret.y+=v.x*p.y+v.y*p.x;
        return ret;
}

/*
  求一条直线的倾角，直线用浮点数表示
  (By phoenixinter)
  输出：直线(x1,y1)-¿(x2,y2)的倾角（严格来说是向量的）
*/
double GetAngle(Point p1,Point p2)
{
    double dx=p2.x-p1.x,dy=p2.y-p1.y;
        double theta,pi=acos(-1.0);
```

```
        if(dx==0)
    {
                if(dy>0) return 90.0;
                else return  270.0;
        }
        else
    {
                theta=atan((double)dy/(double)dx)*180.0/pi; //弧
度化为角度
                if(dx<0) theta+=180;
                else if(dy<0) theta+=360;
                return theta;
        }
}

//求定比分点的坐标(By phoenixinter)
//输入：两个点p1,p2,定比k
//输出：定比分点的坐标
Point dingbipoint(Point p1,Point p2,double k)
{
    Point p;
    p.x=(p1.x+p2.x*k)/(1+k);
    p.y=(p1.y+p2.y*k)/(1+k);
        return p;
}

//二维向量的垂直运算(Perp dot)
//将向量(0,0)-¿V逆时针旋转90度，并保持长度不变。
Point perp(Point v)
{
    Point p;
    p.x=-v.y;p.y=v.x;
    return p;
}

//计算两点之间距离
inline double Dis(Point p1,Point p2)
{
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}

/*========================================
               点线关系
==========================================*/
/*
  判断三点是否共线
  输入：三个点P1,P2,P3
  输出：true/false;
*/
bool dots_inline(Point p1,Point p2,Point p3)
```

```
{
    return zero(times(p3,p1,p2));
}

/*
  P0是否在向量P1P2的左边
 ¿0 说明在左边
  =0 说明在P1P2这条直线上
 ¡0 说明在P1P2右边
*/

double isLeft(Point P0,Point P1,Point P2)
{
    return times(P0,P1,P2);
}

/*
  计算点p到直线L的距离
  调用：无
*/
double Dis2Line(Point p,Line L)
{
    Point a,b;
    a.x=p.x-L.p1.x;a.y=p.y-L.p1.y;
    b.x=L.p2.x-L.p1.x;b.y=L.p2.y-L.p1.y;
    return fabs(a.x*b.y-a.y*b.x)/sqrt(b.x*b.x+b.y*b.y);
}

double ptoline(Point P,Line L)
{
    return fabs(times(L.p2,P,L.p1))/Dis(L.p1,L.p2);
}

/*
  计算点p到直线L的最近点
  调用：无
*/
Point Npt2Line(Point p,Line L)
{
    Point res;
    double a,b,t;
    a=L.p2.x-L.p1.x;b=L.p2.y-L.p1.y;
    t=((p.x-L.p1.x)*a+(p.y-L.p1.y)*b)/(a*a+b*b);
    res.x=L.p1.x+a*t;
    res.y=L.p1.y+b*t;
    return res;
}

/*
  判断点p是否在直线L上
```

```
  调用：无
*/
bool OnLine(Point p,Line L)
{
    double res;
    res=(L.p2.x-L.p1.x)*(p.y-L.p1.y)-(L.p2.y-L.p1.y)*(p.x-L.p1.x);
    return fabs(res)<EPS;
}

/*
  计算点p与直线L的相对关系
  输出：
    0 - 点p在直线L上
    1 - 点p在直线L左侧
    2 - 点p在直线L右侧
  调用；无
*/
int Relation(Point p,Line L)
{
    double res;
    res=(L.p2.x-L.p1.x)*(p.y-L.p1.y)-(L.p2.y-L.p1.y)*(p.x-L.p1.x);
    if(fabs(res)<EPS)
        return 0;
    else
        return(res>0)?1:2;
}

/*
  求点p关于直线L的对称点
  调用：无
*/
Point SymPoint(Point p,Line L)
{
    Point res;
    double a,b,t;
    a=L.p2.x-L.p1.x;b=L.p2.y-L.p1.y;
    t=((p.x-L.p1.x)*a+(p.y-L.p1.y)*b)/(a*a + b*b);
    res.x=2*L.p1.x+2*a*t-p.x;
    res.y=2*L.p1.y+2*b*t-p.y;
    return res;
}

/*
  判断点p是否在线段L上
  调用：无
*/
bool OnLineSeg(Point p,Line L)
{
    double r;
    r=(L.p2.x-L.p1.x)*(p.y-L.p1.y)-(L.p2.y-L.p1.y)*(p.x-L.p1.x);
```

```
    return (fabs(r)<EPS&&(p.x-L.p1.x)*(p.x-L.p2.x)<=EPS
            &&(p.y-L.p1.y)*(p.y-L.p2.y)<=EPS);
}

//判点是否在线段上,包括端点(By phoenixinter)
int dot_online_in(Point p,Line L)
{
    return zero(times(L.p2,p,L.p1))&&(L.p1.x-p.x)*(L.p2.x-p.x)<EPS
    &&(L.p1.y-p.y)*(L.p2.y-p.y)<EPS;
}

//判点是否在线段上,不包括端点(By phoenixinter)
int dot_online_ex(Point p,Line L)
{
    return dot_online_in(p,L)&&(!zero(p.x-L.p1.x)||!zero(p.y-L.p1.y))
        &&(!zero(p.x-L.p2.x)||!zero(p.y-L.p2.y));
}

//判两点在线段同侧,点在线段上返回0(By phoenixinter)
int same_side(Point p1,Point p2,Line L)
{
    return times(L.p2,L.p1,p1)*times(L.p2,L.p1,p2)>EPS;
}

//判两点在线段异侧,点在线段上返回0(By phoenixinter)
int opposite_side(Point p1,Point p2,Line L)
{
    return times(L.p2,L.p1,p1)*times(L.p2,L.p1,p2)<-EPS;
}

/*
  计算点p到线段L的最近点
  调用：无
*/
Point Npt2LineSeg(Point p,Line L)
{
    Point res;
    double a,b,t,d1,d2;
    a=L.p2.x-L.p1.x;b=L.p2.y-L.p1.y;
    t=((p.x-L.p1.x)*a+(p.y-L.p1.y)*b)/(a*a+b*b);
    if(geq(t,0)&&leq(t,1))
    {
        res.x=L.p1.x+a*t;
        res.y=L.p1.y+b*t;
    }
    else
    {
        d1=(p.x-L.p1.x)*(p.x-L.p1.x)+(p.y-L.p1.y)*(p.y-L.p1.y);
        d2=(p.x-L.p2.x)*(p.x-L.p2.x)+(p.y-L.p2.y)*(p.y-L.p2.y);
        if(d1<d2)
```

```
            res=L.p1;
        else
            res=L.p2;
    }
    return res;
}

/*
  计算点p到线段L的距离(By phoenixinter)
  如果p到直线L的垂点在线段上，那么返回点P到直线L的距离
  否则返回P到线段两个端点较近一个的距离
*/
double ptoseg(Point p,Line L)
{
    Point t=p;
    t.x+=L.p1.y-L.p2.y;
    t.y+=L.p2.x-L.p1.x;
    if(times(p,L.p1,t)*times(p,L.p2,t)>EPS)
        return Dis(p,L.p1)<Dis(p,L.p2)?Dis(p,L.p1):Dis(p,L.p2);
    return fabs(times(L.p2,p,L.p1))/Dis(L.p1,L.p2);
}

//点到线段最近距离的平方
double SquaredDistance(Point Y,Line S)
{
    Point D=S.p2-S.p1;
    Point YmP0=Y-S.p1;
    double t=D&YmP0;
    if(t<=0)
        return YmP0&YmP0;
    double DdD=D&D;
    if(t>=DdD)
    {
        Point YmP1=Y-S.p2;
        return YmP1&YmP1;
    }
    return YmP0&YmP0-t*t/DdD;
}

/*==========================================
            线线关系
==============================================*/
/*
  直线方程：两点式-¿标准式
  输入：P1,P2两个点
  输出：ax+by+c=0的直线方程的(a,b,c)系数
*/
void convert(Point p1,Point p2,double& a,double& b,double& c)
{
    a=p1.y-p2.y;
```

```
    b=p2.x-p1.x;
    c=p1.x*p2.y-p2.x*p1.y;
}

/*
  判断两条直线L1,L2是否相交
  调用：无
*/
bool LineIntersect(Line L1,Line L2)
{
    double res;
    res=(L1.p1.x-L1.p2.x)*(L2.p1.y-L2.p2.y)-
        (L1.p1.y-L1.p2.y)*(L2.p1.x-L2.p2.x);
    return fabs(res)>EPS;
}

//判断两条直线是否垂直(By phoenixinter)
bool perpendicular(Line u,Line v)
{
    return zero((u.p1.x-u.p2.x)*(v.p1.x-v.p2.x)+
           (u.p1.y-u.p2.y)*(v.p1.y-v.p2.y));
}

/*
  判断两条线段L1,L2是否相交
  调用：函数times
  说明：快速排斥实验不仅仅是为了提高效率，更是必不可少的!
*/
bool LineSegIntersect(Line L1,Line L2)
{
    return(geq(max(L1.p1.x,L1.p2.x),min(L2.p1.x,L2.p2.x))
    &&geq(max(L2.p1.x,L2.p2.x),min(L1.p1.x,L1.p2.x))
    &&geq(max(L1.p1.y,L1.p2.y),min(L2.p1.y,L2.p2.y))
    &&geq(max(L2.p1.y,L2.p2.y),min(L1.p1.y,L1.p2.y))
    &&times(L1.p1,L2.p1,L1.p2)*times(L1.p1,L2.p2,L1.p2)<=EPS
    &&times(L2.p1,L1.p1,L2.p2)*times(L2.p1,L1.p2,L2.p2)<=EPS);
}

/*
  SRbGa书上的线段相交(By phoenixinter)
  通过测试:zju 1010,
*/
int dblcmp(double d)
{
    if(fabs(d)<EPS)
        return 0;
    return (d>0)?1:-1;
}

double det(double x1,double y1,double x2,double y2)
```

```
{
    return x1*y2-x2*y1;
}

double cross(Point a,Point b,Point c)
{
    return det(b.x-a.x,b.y-a.y,c.x-a.x,c.y-a.y);
}

double dotdet(double x1,double y1,double x2,double y2)
{
    return x1*x2+y1*y2;
}

double dot(Point a,Point b,Point c)
{
    return dotdet(b.x-a.x,b.y-a.y,c.x-a.x,c.y-a.y);
}

int betweencmp(Point a,Point b,Point c)
{
    return dblcmp(dot(a,b,c));
}

//0 no intersection, 1 proper intersection, 2 improper intersection
//p - point of intersection
//判断线段(a,b),(c,d)是否相交
int segcross(Point a,Point b,Point c,Point d,Point& p)
{
    double s1,s2,s3,s4;
    int d1,d2,d3,d4;
    d1=dblcmp(s1=cross(a,b,c));
    d2=dblcmp(s2=cross(a,b,d));
    d3=dblcmp(s3=cross(c,d,a));
    d4=dblcmp(s4=cross(c,d,b));
    if((d1^d2)==-2&&(d3^d4)==-2)
    {
        p.x=(c.x*s2-d.x*s1)/(s2-s1);
        p.y=(c.y*s2-d.y*s1)/(s2-s1);
        return 1;
    }
    if(d1==0&&betweencmp(c,a,b)<=0||
       d2==0&&betweencmp(d,a,b)<=0||
       d3==0&&betweencmp(a,c,d)<=0||
       d4==0&&betweencmp(b,c,d)<=0)
       return 2;
    return 0;
}

// intersect2D_2Segments(): the intersection of 2 finite 2D segments
```

```
//    Input:  two finite segments S1 and S2
//    Output: *I0 = intersect point (when it exists)
//            *I1 = endpoint of intersect segment [I0,I1] (when it exists)
//    Return: 0=disjoint (no intersect)
//            1=intersect in unique point I0
//            2=overlap in segment from I0 to I1
/*
int intersect2D_Segments( Segment S1, Segment S2, Point* I0, Point* I1 )
{
    Vector    u = S1.P1 - S1.P0;
    Vector    v = S2.P1 - S2.P0;
    Vector    w = S1.P0 - S2.P0;
    float     D = perp(u,v);

    // test if they are parallel (includes either being a point)
    if (fabs(D) < SMALL_NUM) {          // S1 and S2 are parallel
        if (perp(u,w) != 0 || perp(v,w) != 0) {
            return 0;                   // they are NOT collinear
        }
        // they are collinear or degenerate
        // check if they are degenerate points
        float du = dot(u,u);
        float dv = dot(v,v);
        if (du==0 && dv==0) {           // both segments are points
            if (S1.P0 != S2.P0)         // they are distinct points
                return 0;
            *I0 = S1.P0;                // they are the same point
            return 1;
        }
        if (du==0) {                    // S1 is a single point
            if (inSegment(S1.P0, S2) == 0)  // but is not in S2
                return 0;
            *I0 = S1.P0;
            return 1;
        }
        if (dv==0) {                    // S2 a single point
            if (inSegment(S2.P0, S1) == 0)  // but is not in S1
                return 0;
            *I0 = S2.P0;
            return 1;
        }
        // they are collinear segments - get overlap (or not)
        float t0, t1;                   // endpoints of S1 in eqn for S2
        Vector w2 = S1.P1 - S2.P0;
        if (v.x != 0) {
                t0 = w.x / v.x;
                t1 = w2.x / v.x;
        }
        else {
                t0 = w.y / v.y;
```

```
                t1 = w2.y / v.y;
        }
        if (t0 > t1) {                    // must have t0 smaller than t1
                float t=t0; t0=t1; t1=t;      // swap if not
        }
        if (t0 > 1 || t1 < 0) {
            return 0;      // NO overlap
        }
        t0 = t0<0? 0 : t0;                // clip to min 0
        t1 = t1>1? 1 : t1;                // clip to max 1
        if (t0 == t1) {                   // intersect is a point
            *I0 = S2.P0 + t0 * v;
            return 1;
        }

        // they overlap in a valid subsegment
        *I0 = S2.P0 + t0 * v;
        *I1 = S2.P0 + t1 * v;
        return 2;
    }

    // the segments are skew and may intersect in a point
    // get the intersect parameter for S1
    float     sI = perp(v,w) / D;
    if (sI < 0 || sI > 1)                 // no intersect with S1
        return 0;

    // get the intersect parameter for S2
    float     tI = perp(u,w) / D;
    if (tI < 0 || tI > 1)                 // no intersect with S2
        return 0;

    *I0 = S1.P0 + sI * u;                 // compute S1 intersect point
    return 1;
}
//===================================================================
*/

/*
  计算两条直线的交点
  输入：L1, L2 – 两条直线
  输出：P – 两直线的交点
    返回值说明了两条直线的位置关系
   1 – 共线
   2 – 平行
   0 – 相交
  调用：宏eq
*/
int CalCrossPoint(Line L1,Line L2,Point& P)
{
```

```
    double a1,b1,c1,a2,b2,c2;
    a1=L1.p2.y-L1.p1.y;b1=L1.p1.x-L1.p2.x;
    c1=L1.p2.x*L1.p1.y-L1.p1.x*L1.p2.y;
    a2=L2.p2.y-L2.p1.y;b2=L2.p1.x-L2.p2.x;
    c2=L2.p2.x*L2.p1.y-L2.p1.x*L2.p2.y;
    if(eq(a1*b2,b1*a2))
    {
        if(eq(a1*c2,a2*c1)&&eq(b1*c2,b2*c1))
            return 1;    //共线
        else
            return 2;    //平行
    }
    else
    {
        P.x=(b2*c1-b1*c2)/(a2*b1-a1*b2);
        P.y=(a1*c2-a2*c1)/(a2*b1-a1*b2);
        return 0;        //相交
    }
}

/*
  求两直线的夹角
  输出：0 PI之间的弧度
   调用：函数norm
*/
double Angle(Line L1,Line L2)
{
    Point u,v;
    u.x=L1.p1.x-L1.p2.x;u.y=L1.p1.y-L1.p2.y;
    v.x=L2.p1.x-L2.p2.x;v.y=L2.p1.y-L2.p2.y;
    return acos((u.x*v.x+u.y*v.y)/(norm(u)*norm(v)));
}

/*
  计算线段L1到线段L2的最短距离
   调用：函数LineSegIntersect, Npt2LineSeg, Dis
*/
double MinDis(Line L1,Line L2)
{
    double d1,d2,d3,d4;
    if(LineSegIntersect(L1,L2))
        return 0;
    else
    {
        d1=Dis(Npt2LineSeg(L1.p1,L2),L1.p1);
        d2=Dis(Npt2LineSeg(L1.p2,L2),L1.p2);
        d3=Dis(Npt2LineSeg(L2.p1,L1),L2.p1);
        d4=Dis(Npt2LineSeg(L2.p2,L1),L2.p2);
        return min(min(d1,d2),min(d3,d4));
    }
```

```
}

//计算两个移动物体的最近距离及其时间

struct Track
{
    Point P0;          //物体的初始位置
    Point v;           //物体移动的方向向量
};

#define dot1(u,v)    ((u).x*(v).x+(u).y*(v).y)

double cpa_time(Track Tr1,Track Tr2)
{
    Point dv=Tr1.v-Tr2.v;
    double dv2=dot1(dv,dv);
    if(dv2<EPS)
        return 0.0;
    Point w0=Tr1.P0-Tr2.P0;
    double cpatime=-dot1(w0,dv)/dv2;
    return cpatime;
}

double cpa_distance(Track Tr1,Track Tr2)
{
    double ctime=cpa_time(Tr1,Tr2);
    Point P1,P2;
    P1.x=Tr1.P0.x+(ctime*Tr1.v.x);
    P1.y=Tr1.P0.y+(ctime*Tr1.v.y);
    P2.x=Tr2.P0.x+(ctime*Tr2.v.x);
    P2.y=Tr2.P0.y+(ctime*Tr2.v.y);
    return Dis(P1,P2);
}

/*==================================================
             三角形
==================================================*/
struct Triangle
{
    Point P0,P1,P2;
};

/*
  计算三角形面积
  输入：a，b，c是三角形的三个顶点
  输出：三角形面积，面积正负按照右手旋规则确定
  调用：无
*/
double Area(Point a,Point b,Point c)
{
```

```
    return((b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x))/2.0;
}

/*
  计算三角形面积
 输入：a，b，c是三角形的三条边
  输出：三角形面积
  调用：无
*/
double Area(double a,double b,double c)
{
    double s=(a+b+c)/2.0;
    return sqrt(s*(s-a)*(s-b)*(s-c));
}

/*
 判断点p是否在三角形ABC内
  输出：0 - 点p在三角形外
   1 - 点p在三角形内
   2 - 点p在三角形边界上
  调用：函数OnLineSeg, Relation
*/
int InTriangle(Point p,Point A,Point B,Point C)
{
    Point center;
    Line side[3];
    int i,rel;
    center.x=(A.x+B.x+C.x)/3.0;center.y=(A.y+B.y+C.y)/3.0;
    side[0].p1=A;side[0].p2=B;
    side[1].p1=B;side[1].p2=C;
    side[2].p1=C;side[2].p2=A;
    rel=Relation(center,side[0]);
    for(i=0;i<3;i++)
    {
        if(OnLineSeg(p,side[i]))
            return 2;           //点p在三角形边界上
        else if(Relation(p,side[i])!=rel)
            return 0;           //点p在三角形外
    }
    return 1;
}

/*
  计算点到三角形的最近距离
*/
double SquaredDistance(Point Y,Triangle T)
{
    Point D0=T.P1-T.P0,D1=T.P2-T.P0,Delta=Y-T.P0;
    double a00=D0&D0,a01=D0&D1,a11=D1&D1;
    double b0=D0&Delta,b1=D1&Delta;
```

```
double n0=a11*b0-a01*b1;
double n1=a00*b1-a01*b0;
double d=a00*a11-a01*a01;
if(n0+n1<=d)
{
    if(n0>=0)
    {
        if(n1>=0)
        {//点在三角形内,region 0
            return 0;
        }
        else
        {//region 5
            double c=Delta&Delta;
            if(b0>0)
            {
                if(b0<a00)
                    return c-b0*b0/a00;
                else
                    return a00-2*b0+c;
            }
            else
                return c;
        }
    }
    else if(n1>=0)
    {//region 3
        double c=Delta&Delta;
        if(b1>0)
        {
            if(b1<a11)
                return c-b1*b1/a11;
            else
                return a11-2*b1+c;
        }
        else return c;
    }
    else
    {//region 4
        double c=Delta&Delta;
        if(b0<a00)
        {
            if(b0>0)
                return c-b0*b0/a00;
            else
            {
                if(b1<a11)
                {
                    if(b1>0)
                        return c-b1*b1/a11;
```

```
                        else
                            return c;
                    }
                    else
                        return a11-2*b1+c;
                }
            }
            else
                return a00-2*b0+c;
        }
    }
    else if(n0<0)
    {//region 2
        double c=Delta&Delta;
        if(b1>0)
        {
            if(b1<a11)
                return c-b1*b1/a11;
            else
            {
                double n=a11-a01+b0-b1,d=a00-2*a01+a11;
                if(n>0)
                {
                    if(n<d)
                        return (a11-2*b1+c)-n*n/d;
                    else
                        return a00-2*b0+c;
                }
                else
                    return a11-2*b1+c;
            }
        }
        else
            return c;
    }
    else if(n1<0)
    {//region 6
        double c=Delta&Delta;
        if(b0>0)
        {
            if(b0<a00)
                return c-b0*b0/a00;
            else
            {
                double n=a11-a01+b0-b1,d=a00-2*a01+a11;
                if(n>0)
                {
                    if(n<d)
                        return(a11-2*b1+c)-n*n/d;
                    else
```

```
                    return a00-2*b0+c;
                }
                else
                    return a11-2*b1+c;
            }
        }
        else return c;
    }
    else
    {//region 1
        double c=Delta&Delta;
        double n=a11-a01+b0-b1,d=a00-2*a01+a11;
        if(n>0)
        {
            if(n<d)
                return (a11-2*b1+c)-n*n/d;
            else
                return a00-2*b0+c;
        }
        else
            return a11-2*b1+c;
    }
}

/*
Another Version:
float SquaredDistance (Point Y, Triangle T)
{
// T has vertices V0, V1, V2
// t0 = n0/d0 = Dot(Y - V0, V1 - V0) / Dot(V1 - V0, V1 - V0)
Point D0 = Y - V0, E0 = V1 - V0;
float n0 = Dot(D0, E0);
// t1 = n1/d1 = Dot(Y - V1, V2 - V1) / Dot(V2 - V1, V2 - V1)
Point D1 = Y - V1, E1 = V2 - V1;
float n1 = Dot(D1, E1);
if (n0 <= 0 and n1 <= 0) // closest point is V1
return Dot(D1, D1); // RETURN 0
// t2 = n2/d2 = Dot(Y - V2, V0 - V2) / Dot(V0 - V2, V0 - V2);
Point D2 = Y - V2, E2 = V0 - V2;
float n2 = Dot(D2, E2);
if (n1 <= 0 and n2 == 0) // closest point is V2
return Dot(D2, D2); // RETURN 1
if (n0 <= 0 and n2 <= 0) // closest point is V0
return Dot(D0, D0); // RETURN 2
// D0 = Y - V0 = V0 + c1 * (V1 - V0) + c2 * (V2 - V0) = V0 + c1
// * E1 - c2 * E2 for
// c0 + c1 + c2 = 1, c0 = m0 / d, c1 = m1 / d, c2 = m2 / d
float e00 = Dot(E0, E0), e02 = Dot(E0, E2), e22 = Dot(E2, E2);
float d = e02 * e02 - e00 * e22;
float a = Dot(D0, E2);
```

```
float m1 = e02 * a - e22 * n0;
float m0, m2;
Point D;
if (d > 0) {
if (m1 < 0) { // closest point is V2 + t2 * E2
t2 = n2 / e22;
D = Y - (V2 + t2 * E2);
return Dot(D, D); // RETURN 3a
}
m2 = e00 * a - e02 * n0;
if (m2 < 0) { // closest point is V0 + t0 * E0
t0 = n0 / e00;
D = Y - (V0 + t0 * E0);
return Dot(D, D); // RETURN 4a
}
m0 = d - m1 - m2;
if (m0 < 0) { // closest point is V1 + t1 * E1
t1 = n1/Dot(E1, E1);
D = Y - (V1 + t1 * E1);
return Dot(D, D); // RETURN 5a
}
} else {
if (m1 > 0) { // closest point is V2 + t2 * E2
t2 = n2 / e22;
D = Y - (V2 + t2 * E2);
return Dot(D, D); // RETURN 3b
}
m2 = e00 * a - e02 * n0;
if (m2 > 0) { // closest point is V0 + t0 * E0
t0 = n0 / e00;
D = Y - (V0 + t0 * E0);
return Dot(D, D); // RETURN 4b
}
m0 = d - m1 - m2;
if (m0 > 0) { // closest point is V1 + t1 * E1
t1 = n1 / Dot(E1, E1);
D = Y - (V1 + t1 * E1);
return Dot(D, D); // RETURN 5b
}
}
return 0; // Y is inside triangle, RETURN 6
}
*/

/*==================================================
                    多边形
==================================================*/
/*
  计算多边形面积
  输入：poly 多边形顶点数组
```

```
  n 多边形顶点数目
  输出：多边形的面积，正负按照右手旋规则确定
  调用：无
*/
double Area(Point poly[],int n)
{
    double res=0;
    if(n<3) return 0;
    for(int i=0;i<n;i++)
    {
        res+=poly[i].x*poly[(i+1)%n].y;
        res-=poly[i].y*poly[(i+1)%n].x;
    }
    return (res/2.0);
}

/*
  计算多边形的重心，适用于任意简单多边形
  输入的多边形顶点数目必须大于0
  该算法可以一边读入多边形的顶点一边计算重心
  调用：无
*/
Point Orthocenter(Point poly[],int n)
{
    Point p,p0,p1,p2,p3;
    double m,m0;
    p1=poly[0];p2=poly[1];p.x=p.y=m=0;
    for(int i=2;i<n;i++)
    {
        p3=poly[i];
        p0.x=(p1.x+p2.x+p3.x)/3.0;
        p0.y=(p1.y+p2.y+p3.y)/3.0;
        m0=p1.x*p2.y+p2.x*p3.y+p3.x*p1.y-p1.y*p2.x-p2.y*p3.x-p3.y*p1.x;
        if(fabs(m+m0)<EPS)
            m0+=EPS;      // 为了防止除0溢出，对m0做一点点修正
        p.x=(m*p.x+m0*p0.x)/(m+m0);
        p.y=(m*p.y+m0*p0.y)/(m+m0);
        m+=m0;
        p2=p3;
    }
    return p;
}

/*
    计算多边形的重心，采用行列式方法计算
    误差会小一些
    By phoenixinter
*/
Point Orthocenter1(Point poly[], int n)
{
```

```
        Point p;
        p.x=p.y=0;
        for(int i=0;i<n;i++)
        {
          p.x+=(poly[i].x+poly[(i+1)%n].x)*(poly[i].x*poly[(i+1)%n].y-
                poly[(i+1)%n].x*poly[i].y);
          p.y+=(poly[i].y+poly[(i+1)%n].y)*(poly[i].x*poly[(i+1)%n].y-
                poly[(i+1)%n].x*poly[i].y);
        }
        p.x/=(6*Area(poly,n));
        p.y/=(6*Area(poly,n));
        return p;
}

/*
  判断多边形是否是凸的
   调用：函数Relation
*/
bool IsConvex(Point poly[],int n)
{
    int i,rel;
    Line side;
    if(n<3) return false;
    side.p1=poly[0];side.p2=poly[1];
    rel=Relation(poly[2],side);
    for(i=1;i<n;i++)
    {
        side.p1=poly[i];
        side.p2=poly[(i+1)%n];
        if(Relation(poly[(i+2)%n],side)!= rel) return false;
    }
    return true;
}

/*
   判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线
   (By phoenixinter)
*/
int is_convex(int n,Point p[])
{
    int i,s[3]={1,1,1};
    for(i=0;i<n&&s[1]|s[2];i++)
        s[_sign(times(p[i],p[(i+1)%n],p[(i+2)%n]))]=0;
    return s[1]|s[2];
}

/*
  判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线
   (By phoenixinter)
*/
```

```
int is_convex_v2(int n,Point p[])
{
        int i,s[3]={1,1,1};
        for(i=0;i<n&&s[0]&&s[1]|s[2];i++)
            s[_sign(times(p[i],p[(i+1)%n],p[(i+2)%n]))]=0;
        return s[0]&&s[1]|s[2];
}

/*
  判点是否在凸多边形内(By phoenixinter)
  顶点按顺时针或逆时针给出,在多边形边上返回0
*/
int inside_convex_v2(Point q,int n,Point p[])
{
    int i,s[3]={1,1,1};
    for(i=0;i<n&&s[0]&&s[1]|s[2];i++)
        s[_sign(times(p[i],p[(i+1)%n],q))]=0;
    return s[0]&&s[1]|s[2];
}

/*
  判断点p是否在凸多边形poly内
  poly的顶点数目要大于等于3
  输出：0 - 点p在poly外
    1 - 点p在poly内
    2 - 点p在poly边界上
  调用：函数OnLineSeg, Relation
*/
int InConvex(Point p,Point poly[],int n)
{
    Point q;
    Line side;
    int i;
    q.x=q.y=0;
    for(i=0;i<n;i++)
    {
        q.x+=poly[i].x;
        q.y+=poly[i].y;
    }
    q.x=1.0*q.x/n;q.y=1.0*q.y/n;
    for(i=0;i<n;i++)
    {
        side.p1=poly[i];
        side.p2=poly[(i+1)%n];
        if(OnLineSeg(p,side))
            return 2;   //点p在poly边界上
        else if(Relation(p,side)!= Relation(q,side))
            return 0;   //点p在poly外
    }
    return 1;  //点p在poly内
```

```
}

/*
  判断点是否在任意简单多边形内(射线法)
  输出：0 – 点在poly外
   1 – 点在poly内
   2 – 点在poly边界上
  调用：函数eq，OnLineSeg，LineSegIntersect
  测试通过：ZJU 1081
*/
int InPolygon(Point p,Point poly[],int n)
{
    int i,c;
    Line ray,side;
    c=0;
    ray.p1=p;ray.p2.y=p.y;ray.p2.x=-INF;
    for(i=0;i<n;i++)
    {
        side.p1=poly[i];
        side.p2=poly[(i+1)%n];
        if(OnLineSeg(p,side))
            return 2;    //点在poly边界上
        if(eq(side.p1.y,side.p2.y))   //如果side平行x轴则不作考
虑
            continue;
        if (OnLineSeg(side.p1,ray))
        {
            if(side.p1.y>side.p2.y) c++;
        }
        else if(OnLineSeg(side.p2,ray))
        {
            if(side.p2.y>side.p1.y) c++;
        }
        else if(LineSegIntersect(ray,side))
            c++;
    }
    return((c%2==1)?1:0);     //1:点在poly内;0:点在poly外
}

/*
  判断线段是否在任意简单多边形内
  调用：宏eq，函数InPolygon, OnLineSeg, LineSegIntersect
    Point的¡ 操作符
*/
bool InPolygon(Line L,Point poly[],int n)
{
    bool res;
    int i,m;
    Point p,pts[MAX_N];
    Line side;
```

```
    if(!InPolygon(L.p1,poly,n)||!InPolygon(L.p2,poly,n))
        return false;
    m=0;
    for(i=0;i<n;i++)
    {
        side.p1=poly[i];side.p2=poly[(i+1)%n];
        if(OnLineSeg(L.p1,side))
            pts[m++]=L.p1;
        else if(OnLineSeg(L.p2,side))
            pts[m++]=L.p2;
        else if(OnLineSeg(side.p1,L))
            pts[m++]=side.p1;
        else if(OnLineSeg(side.p2,L))
            pts[m++]=side.p2;
        else if(LineSegIntersect(side,L))
            return false;
    }
    sort(&pts[0],&pts[m]);    //对交点进行排序
    for(i=1;i<m;i++)
        if(!eq(pts[i-1].x,pts[i].x)||!eq(pts[i-1].y,pts[i].y))
        {
            p.x=(pts[i-1].x+pts[i].x)/2.0;
            p.y=(pts[i-1].y+pts[i].y)/2.0;
            if(!InPolygon(p,poly,n))
                return false;
        }
    return true;
}

/*
  判断线段是否在任意简单多边形外
   调用：宏eq，函数InPolygon, OnLineSeg, LineSegIntersect
     Point的¡ 操作符
*/
bool OutPolygon(Line L,Point poly[],int n)
{
    bool res;
    int i,m;
    Point p,pts[MAX_N];
    Line side;
    if(InPolygon(L.p1,poly,n)==1||InPolygon(L.p2,poly,n)==1)
        return false;
    m=0;
    for(i=0;i<n;i++)
    {
        side.p1=poly[i];side.p2=poly[(i+1)%n];
        if(OnLineSeg(L.p1,side)&&OnLineSeg(L.p2,side)) return true;
        if(LineSegIntersect(side,L)) return false;
    }
    for(i=0;i<n;i++)
```

```
    {
        side.p1=poly[i]; side.p2=poly[(i+1)%n];
        if(OnLineSeg(L.p1,side))
            pts[m++]=L.p1;
        else if(OnLineSeg(L.p2,side))
            pts[m++]=L.p2;
        else if(OnLineSeg(side.p1,L))
            pts[m++]=side.p1;
        else if(OnLineSeg(side.p2,L))
            pts[m++]=side.p2;
    }
    sort(&pts[0],&pts[m]);
    for(i=1;i<m;i++)
    {
        if(!eq(pts[i-1].x,pts[i].x)||!eq(pts[i-1].y,pts[i].y))
        {
            p.x=(pts[i-1].x+pts[i].x)/2.0;
            p.y=(pts[i-1].y+pts[i].y)/2.0;
            if(InPolygon(p,poly,n)==1)
                return false;
        }
    }
    return true;
}

/*
  用有向直线line切割凸多边形
  复杂度；O(n)
  输入：line 用来切割凸多边形的有向直线
  poly 凸多边形顶点数组，顶点必须按照逆时针排列
  n 凸多边形的顶点数目
  输出：result[1] 切割后line的左侧部分
   result[2] 切割后line的右侧部分
   result[0] 没有用到，只是作为辅助存储空间
   m[0..2] m[i]是result[i]中顶点的数目
     返回值切口的长度，如果返回值为0,说明未做切割
     当未作切割时，如果多边形在该直线的左侧，
   则result[1]等于该多边形，否则result[2]等于该多边形
  注意：
  1. 被切割的多边形一定要是凸多边形，顶点按照逆时针排列
  2. 可利用这个函数来求多边形的核，初始的核设为一个很大的矩形，
   然后依次用多边形的每条边去割
  调用：函数Relation, CalCrossPoint, Dis
*/
double CutConvex(Line line,Point poly[],int n,
                 Point result[3][MAX_N],int m[3])
{
    Point pts[3],p;
    Line side;
    int i,cur,pre,npt;
```

```
    m[0]=m[1]=m[2]=npt=0;
    if(n==0)    return 0;
    pre=cur=Relation(poly[0],line);
    for(i=0;i<n;i++)
    {
        cur=Relation(poly[(i+1)%n],line);
        if(cur==pre)
            result[cur][m[cur]++]=poly[(i+1)%n];
        else
        {
            side.p1=poly[i];side.p2=poly[(i+1)%n];
            CalCrossPoint(side,line,p);
            pts[npt++]=p;
            result[pre][m[pre]++]=p;
            result[cur][m[cur]++]=p;
            result[cur][m[cur]++]=poly[(i+1)%n];
            pre=cur;
        }
    }
    sort(&pts[0],&pts[npt]);
    if(npt<2)
        return 0;
    else
        return Dis(pts[0],pts[npt-1]);
}

/*
  寻找一个点到多边形的切点
  输入：点P(多边形外部的一个点)
   n(多边形的顶点数)
   V(多边形的顶点坐标)
  输出：int rtan,ltan
   分别表示最右边和最左边的切点的坐标的index
*/

// tests for polygon vertex ordering relative to a fixed point P
#define above(P,Vi,Vj)  (isLeft(P,Vi,Vj) > 0)   // true if Vi is above Vj
#define below(P,Vi,Vj)  (isLeft(P,Vi,Vj) < 0)   // true if Vi is below Vj

//复杂度O(n)
void tangent_PointPoly(Point P,int n,Point V[],int& rtan,int& ltan)
{
    double eprev,enext;
    int i;
    rtan=ltan=0;
    eprev=isLeft(V[0],V[1],P);
    for(i=1;i<n;i++)
    {
        enext=isLeft(V[i],V[(i+1)%n],P);
        if(eprev<=0&&enext>0)
```

```
        {
            if(!below(P,V[i],V[rtan]))
                rtan=i;
        }
        else if(eprev>0&&enext<=0)
        {
            if(!above(P,V[i],V[ltan]))
                ltan=i;
        }
        eprev=enext;
    }
    return;
}

int Rtangent_PointPolyC(Point P,int n,Point V[]);
int Ltangent_PointPolyC(Point P,int n,Point V[]);

void tangent_PointPolyC(Point P,int n,Point V[],int& rtan,int& ltan)
{
    rtan=Rtangent_PointPolyC(P,n,V);
    ltan=Ltangent_PointPolyC(P,n,V);
}

//返回：最右边切点坐标的index
int Rtangent_PointPolyC(Point P,int n,Point V[])
{
    int a,b,c,upA,dnC;
    if(below(P,V[1],V[0])&&!above(P,V[n-1],V[0]))
        return 0;
    for(a=0,b=n;;)
    {
        c=(a+b)/2;
        dnC=below(P,V[c+1],V[c]);
        if(dnC&&!above(P,V[c-1],V[c]))
            return c;
        upA=above(P,V[a+1],V[a]);
        if(upA)
        {
            if(dnC)
                b=c;
            else
            {
                if(above(P,V[a],V[c]))
                    b=c;
                else
                    a=c;
            }
        }
        else
        {
```

```
                if(!dnC)
                    a=c;
                else
                {
                    if(below(P,V[a],V[c]))
                        b=c;
                    else
                        a=c;
                }
            }
        }
}

//返回：最左边切点坐标的index
int Ltangent_PointPolyC(Point P,int n,Point V[])
{
    int a,b,c,dnA,dnC;
    if(above(P,V[n-1],V[0])&&!below(P,V[1],V[0]))
        return 0;
    for(a=0,b=n;;)
    {
        c=(a+b)/2;
        dnC=below(P,V[c+1],V[c]);
        if(above(P,V[c-1],V[c])&&!dnC)
            return c;
        dnA=below(P,V[a+1],V[a]);
        if(dnA)
        {
            if(!dnC)
                b=c;
            else
            {
                if(below(P,V[a],V[c]))
                    b=c;
                else
                    a=c;
            }
        }
        else
        {
            if(dnC)
                a=c;
            else
            {
                if(above(P,V[a],V[c]))
                    b=c;
                else
                    a=c;
            }
        }
```

```
    }
}

// RLtangent_PolyPolyC(): get the RL tangent between two convex polygons
//    Input:  m = number of vertices in polygon 1
//            V = array of vertices for convex polygon 1 with V[m]=V[0]
//            n = number of vertices in polygon 2
//            W = array of vertices for convex polygon 2 with W[n]=W[0]
//    Output: *t1 = index of tangent point V[t1] for polygon 1
//            *t2 = index of tangent point W[t2] for polygon 2
void RLtangent_PolyPolyC( int m, Point* V, int n, Point* W, int* t1, int* t2 )
{
    int ix1, ix2;       // search indices for polygons 1 and 2

    // first get the initial vertex on each polygon
    ix1 = Rtangent_PointPolyC(W[0], m,V);    // right tangent from W[0] to V
    ix2 = Ltangent_PointPolyC(V[ix1], n,W); // left tangent from V[ix1] to W

    // ping-pong linear search until it stabilizes
    int done = false;                     // flag when done
    while (done==false) {
        done = true;                      // assume done until...
        while (isLeft(W[ix2], V[ix1], V[ix1+1]) <= 0){
            ++ix1;                        // get Rtangent from W[ix2] to V
        }
        while (isLeft(V[ix1], W[ix2], W[ix2-1]) >= 0){
            --ix2;                        // get Ltangent from V[ix1] to W
            done = false;                 // not done if had to adjust this
        }
    }
    *t1 = ix1;
    *t2 = ix2;
    return;
}

/*==========================================
               矩形
==========================================*/
//表示矩形，左下角坐标是(llx,lly),右上角坐标是(urx,ury)
struct Rect
{
    int llx,lly,urx,ury;
};

/*
  判断两个矩形是否相交
  相邻不算相交
*/
bool intersect(Rect r1,Rect r2)
{
```

```
    return(max(r1.llx,r2.llx)<min(r1.urx,r2.urx)&&
            max(r1.lly,r2.lly)<min(r1.ury,r2.ury));
}

/*
  判断点p是否在矩形内
  调用：函数geq
  通过测试：PKU 1468
*/
bool inrect(Point p,Rect rect)
{
    return(geq(p.x,rect.llx)&&leq(p.x,rect.urx)
        &&geq(p.y,rect.lly)&&leq(p.y,rect.ury));
}

/*
  用矩形b切割矩形a
  输出：out[4]中保存切割a后得到的新矩形
   a 本身将会被改变
  返回值:如果矩形a,b不相交，返回0
   如果矩形b完全覆盖矩形a,返回-1
   否则返回切割后得到的新矩形的数目
*/
int CutRect(Rect& a,Rect b,Rect out[4])
{
    if(b.urx<a.llx||b.llx>=a.urx)    return 0;
    if(b.ury<a.lly||b.lly>=a.ury)    return 0;
    if(b.llx<=a.llx&&b.urx>=a.urx&&b.lly<=a.lly&&b.ury>=a.ury)
        return -1;
    int n=0;
    if(b.llx>a.llx)
    {
        out[n]=a;out[n].urx=b.llx;
        n++;a.llx=b.llx;
    }
    if(b.urx<a.urx)
    {
        out[n]=a;out[n].llx=b.urx;
        n++;a.urx=b.urx;
    }
    if(b.lly>a.lly)
    {
        out[n]=a;out[n].ury=b.lly;
        n++;a.lly=b.lly;
    }
    if(b.ury<a.ury)
    {
        out[n]=a;out[n].lly=b.ury;
        n++;a.ury=b.ury;
    }
```

```
    return n;
}

//用长宽表示矩形,w,h分别表示宽度和高度
struct Rect2
{
    double w,h;
};

/*
  判断矩形r2是否可以放置在矩形r1内
  r1和r2可以任意地旋转
  调用：无
*/
bool IsContain(Rect2 r1,Rect2 r2)
{
    double cross,alpha;
    if(r1.h>r1.w) swap(r1.h,r1.w);
    if(r2.h>r2.w) swap(r2.h,r2.w);
    if(leq(r2.h,r1.h)&&leq(r2.w,r1.w))
        return true;
    cross=sqrt(r2.w*r2.w+r2.h*r2.h);
    //注意，现在r1.h肯定大于cross
    alpha=asin(r1.h/cross)-asin(r2.h/cross);
    if(alpha>0&&2.0*alpha<PI)
    {
        if(r2.w*cos(alpha)+r2.h*sin(alpha)<=r1.w+EPS)
            return true;
    }
    swap(r2.h,r2.w);
    alpha=asin(r1.h/cross)-asin(r2.h/cross);
    if(alpha>0&&2.0*alpha<PI)
    {
        if(r2.w*cos(alpha)+r2.h*sin(alpha)<=r1.w+EPS)
            return true;
    }
    return false;
}

/*==================================================
                圆(By phoenixinter)
==================================================*/
struct Circle
{
    Point c;
    double r;
};

/*
  判线段和圆相交, 包括端点和相切
```

```
*/
bool intersect(Circle C,Line L)
{
    Point c=C.c;
    double t1=Dis(c,L.p1)-C.r,t2=Dis(c,L.p2)-C.r;
    if(t1<EPS||t2<EPS)
        return t1>-EPS||t2>-EPS;
    Point t=c;
    t.x+=L.p1.y-L.p2.y;
    t.y+=L.p2.x-L.p1.x;
    return times(t,L.p1,c)*times(t,L.p2,c)<EPS&&Dis2Line(c,L)-C.r<EPS;
}

double mario(double a, double b, double c)
{
    return acos(.5*(a*a+b*b-c*c)/(a*b));
}


//计算两个圆相交部分的面积
//通过测试：PKU 2546
double commonarea(Circle a,Circle b)
{
    double d=Dis(a.c,b.c),a1,a2,a3;
    if(fabs(a.r-b.r)>=d)
    {
        if(a.r<b.r)
            return PI*a.r*a.r;
        else
            return PI*b.r*b.r;
    }
    else if(a.r+b.r<=d)
        return 0;
    else
    {
        a1=mario(a.r,d,b.r);
                a2=mario(b.r,d,a.r);
                a3=mario(a.r,b.r,d);
                double ans=(a1*a.r*a.r+a2*b.r*b.r-a.r*b.r*sin(a3));
                return ans;
    }
}

/*
  计算三个点所组成的三角形的外接圆
  center是圆心的坐标，r是外接圆的半径
  通过测试：PKU 2242
*/
void outercircle(Point p1,Point p2,Point p3,Point& center,double& r)
{
```

```
        double a,b,c,d,e,f,g;
        a=p2.x-p1.x;b=p2.y-p1.y;
        c=p3.x-p1.x;d=p3.y-p1.y;
        e=a*(p1.x+p2.x)+b*(p1.y+p2.y);
        f=c*(p1.x+p3.x)+d*(p1.y+p3.y);
        g=2.0*(a*(p3.y-p2.y)-b*(p3.x-p2.x));
        center.x=(d*e-b*f)/g;center.y=(a*f-c*e)/g;
        r=sqrt((p1.x-center.x)*(p1.x-center.x)+
               (p1.y-center.y)*(p1.y-center.y));
}

/*
  输入两个点和半径，输出两个可能的圆心
  如果有圆心，返回true
  else return false
*/
bool centre(Point a,Point b,double r,Point& p1,Point& p2)
{
    double rise,run,theta,chordlen,perplen,tantheta,tantheta1;
    Point temp;
    temp.x=a.x-b.x;temp.y=a.y-b.y;
    chordlen=sqrt(temp.x*temp.x+temp.y*temp.y);
    if(chordlen>2*r)
        return false;
    tantheta=sqrt(4*r*r-chordlen*chordlen)/chordlen;
    run=(a.x-b.x)/2;rise=(a.y-b.y)/2;
    p1.x=(a.x+b.x)/2+rise*tantheta;p1.y=(a.y+b.y)/2-run*tantheta;
    p2.x=(a.x+b.x)/2-rise*tantheta;p2.y=(a.y+b.y)/2+run*tantheta;
    return true;
}

/*
  计算两个圆外公切线的交点
  如果不存在，返回false
  通过测试：ZJU 1199
*/
double sqr(double x)
{
    return x*x;
}

bool outertangent(Circle a,Circle b,Point& p)
{
    if(leq(sqr(a.c.x-b.c.x)+sqr(a.c.y-b.c.y),(a.r-b.r)*(a.r-b.r))
    ||eq(a.r,b.r))
        return false;
    p.x=(b.c.x*a.r-a.c.x*b.r)/(a.r-b.r);
    p.y=(b.c.y*a.r-a.c.y*b.r)/(a.r-b.r);
    return true;
}
```

```
/*==================================================
                圆(By freezy)
==================================================*/


/*求点于圆切线的切点
    poi : 点
    cc : 圆
    输出：result1,result2 为两个切点
*/
void TangentPoint_PC(Point poi,Circle cc,Point& result1,Point& result2){
        double line=sqrt((poi.x-cc.c.x)*(poi.x-cc.c.x)+
                         (poi.y-cc.c.y)*(poi.y-cc.c.y));
        double angel=acos(cc.r/line);
        Point unitvector,lin;
        lin.x=poi.x-cc.c.x;
        lin.y=poi.y-cc.c.y;
        unitvector.x=lin.x/sqrt(lin.x*lin.x+lin.y*lin.y)*cc.r;
        unitvector.y=lin.y/sqrt(lin.x*lin.x+lin.y*lin.y)*cc.r;
        result1=Rotate(unitvector,-angel);
        result2=Rotate(unitvector,angel);
        result1.x+=cc.c.x;
        result1.y+=cc.c.y;
        result2.x+=cc.c.x;
        result2.y+=cc.c.y;
        return;
}

/* 求两圆的外公切线的切点
    两圆：c1,c2;
    输出四个点：
    p1,p2,p3,p3
    其中p1,p2在c1上，p3,p4在c2上。
    而且p1,p3为一条外公切线
      p2,p4为一条外公切线
*/
void TangentPoint_CC_out(Circle c1,Circle c2,Point &p1,Point &p2,Point &p3
        double line=sqrt((c1.c.x-c2.c.x)*(c1.c.x-c2.c.x)+
                    (c1.c.y-c2.c.y)*(c1.c.y-c2.c.y));
        double angel1=acos((fabs(c1.r-c2.r)/line));
        Point unitvector1,unitvector2,lin1,lin2;
        if (c1.r>=c2.r){
          lin1.x=c2.c.x-c1.c.x;
          lin1.y=c2.c.y-c1.c.y;
          unitvector1.x=lin1.x/sqrt(lin1.x*lin1.x+lin1.y*lin1.y)*c1.r;
          unitvector1.y=lin1.y/sqrt(lin1.x*lin1.x+lin1.y*lin1.y)*c1.r;
          lin2.x=c1.c.x-c2.c.x;
          lin2.y=c1.c.y-c2.c.y;
          unitvector2.x=lin2.x/sqrt(lin2.x*lin2.x+lin2.y*lin2.y)*c2.r;
```

```
        unitvector2.y=lin2.y/sqrt(lin2.x*lin2.x+lin2.y*lin2.y)*c2.r;
        p1=Rotate(unitvector1,angel1);
        p2=Rotate(unitvector1,-angel1);
        p3=Rotate(unitvector2,-(PI-angel1));
        p4=Rotate(unitvector2,(PI-angel1));
        p1.x+=c1.c.x;p1.y+=c1.c.y;
        p2.x+=c1.c.x;p2.y+=c1.c.y;
        p3.x+=c2.c.x;p3.y+=c2.c.y;
        p4.x+=c2.c.x;p4.y+=c2.c.y;
    }

    else {
        lin2.x=c1.c.x-c2.c.x;
        lin2.y=c1.c.y-c2.c.y;
        unitvector2.x=lin2.x/sqrt(lin2.x*lin2.x+lin2.y*lin2.y)*c2.r;
        unitvector2.y=lin2.y/sqrt(lin2.x*lin2.x+lin2.y*lin2.y)*c2.r;
        p3=Rotate(unitvector2,angel1);
        p4=Rotate(unitvector2,-angel1);
        lin1.x=c2.c.x-c1.c.x;
        lin1.y=c2.c.y-c1.c.y;
        unitvector1.x=lin1.x/sqrt(lin1.x*lin1.x+lin1.y*lin1.y)*c1.r;
        unitvector1.y=lin1.y/sqrt(lin1.x*lin1.x+lin1.y*lin1.y)*c1.r;
        p1=Rotate(unitvector1,-(PI-angel1));
        p2=Rotate(unitvector1,(PI-angel1));
        p1.x+=c1.c.x;p1.y+=c1.c.y;
        p2.x+=c1.c.x;p2.y+=c1.c.y;
        p3.x+=c2.c.x;p3.y+=c2.c.y;
        p4.x+=c2.c.x;p4.y+=c2.c.y;
    }
    return ;
}

/*
  求两圆的内公切线的切点
  两圆：c1,c2;
  输出：p1,p2,p3,p4;
  其中p1,p2在c1上，p3,p4在c2上；
  且p1,p3为一条内公切线
   p2,p4为一条内公切线
*/
void TangentPoint_CC_in(Circle c1,Circle c2,Point &p1,Point &p2,Point &p3,Point &p4
    double line=sqrt((c1.c.x-c2.c.x)*(c1.c.x-c2.c.x)+
                (c1.c.y-c2.c.y)*(c1.c.y-c2.c.y));
    double angel1=acos((c1.r+c2.r)/line);
    Point unitvector1,unitvector2,lin1,lin2;

    lin1.x=c2.c.x-c1.c.x;
    lin1.y=c2.c.y-c1.c.y;
    unitvector1.x=lin1.x/sqrt(lin1.x*lin1.x+lin1.y*lin1.y)*c1.r;
    unitvector1.y=lin1.y/sqrt(lin1.x*lin1.x+lin1.y*lin1.y)*c1.r;
```

```
        lin2.x=c1.c.x-c2.c.x;
        lin2.y=c1.c.y-c2.c.y;
        unitvector2.x=lin2.x/sqrt(lin2.x*lin2.x+lin2.y*lin2.y)*c2.r;
        unitvector2.y=lin2.y/sqrt(lin2.x*lin2.x+lin2.y*lin2.y)*c2.r;

        p1=Rotate(unitvector1,angel1);
        p2=Rotate(unitvector1,-angel1);

        p3=Rotate(unitvector2,angel1);
        p4=Rotate(unitvector2,-angel1);

        p1.x+=c1.c.x;p1.y+=c1.c.y;
        p2.x+=c1.c.x;p2.y+=c1.c.y;
        p3.x+=c2.c.x;p3.y+=c2.c.y;
        p4.x+=c2.c.x;p4.y+=c2.c.y;

        return ;
}

/*
判断点是否在圆内
返回值：点p在圆内(包括边界)时，返回true
用途：因为圆为凸集，所以判断点集，折线，多边形是否在圆内时，只需要逐
一判断点是否在圆内即可。
*/
//(未测试)

bool point_in_circle(Point o,double r,Point p)
{
    double d2=(p.x-o.x)*(p.x-o.x)+(p.y-o.y)*(p.y-o.y);
    double r2=r*r;
    return d2<r2||fabs(d2-r2)<EPS;
}

/*
    用途：求不共线的三点确定一个圆
    输入：三个点p1,p2,p3
    返回值：如果三点共线，返回false；反之，返回true。圆心由q返回，半
径由r返回
*/
//(未测试)
bool cocircle(Point p1,Point p2,Point p3,Point &q,double &r)
{
    double a1,a2,a3;
    a1 = Dis(p1,p2);
    a2 = Dis(p2,p3);
    a3 = Dis(p1,p3);
    double s = Area(a1, a2, a3);
```

```
    if (fabs(s) < EPS) return false;

    r = a1 * a2 * a3 / ( 4 * s );

    Point p;
    p.x = p2.x - p1.x;
    p.y = p2.y - p1.y;

    double theda = acos(a1 / 2 / r);
    q = Rotate(p,theda);
    return true;
}


/*
     判断圆于线段是否相交
      输入：C,L
     输出：true表示相交
       flase表示不相交
*/

bool intersect1(Circle C,Line L)
{
    Point c=C.c;
    double t1=Dis(c,L.p1)-C.r,t2=Dis(c,L.p2)-C.r;
    if(t1<EPS||t2<EPS)
        return t1>-EPS||t2>-EPS;
    Point t=c;
    t.x+=L.p1.y-L.p2.y;
    t.y+=L.p2.x-L.p1.x;
    return times(t,L.p1,c)*times(t,L.p2,c)<EPS&&Dis2Line(c,L)-C.r<EPS;
}

/*

/*
    求两圆的相交面积
*/
double commonarea1(Circle a,Circle b)
{
    double d=Dis(a.c,b.c),a1,a2,a3;
    if(fabs(a.r-b.r)>=d)
    {
        if(a.r<b.r)
            return PI*a.r*a.r;
        else
            return PI*b.r*b.r;
    }
    else if(a.r+b.r<=d)
        return 0;
```

```
    else
    {
        a1=mario(a.r,d,b.r);
                a2=mario(b.r,d,a.r);
                a3=mario(a.r,b.r,d);
                double ans=(a1*a.r*a.r+a2*b.r*b.r-a.r*b.r*sin(a3));
                return ans;
    }
}

/*
    求三角形内切圆圆心
    输入：p1,p2,p3;
    输出: q,r;
*/

void Inscribed_circle(Point p1, Point p2, Point p3, Point &q , double &r)
{
    double a1,a2,a3;
    a1 = Dis(p1,p2);
    a2 = Dis(p1,p3);
    a3 = Dis(p2,p3);
    r = 2 * fabs(Area(p1,p2,p3)) / (a1 + a2 + a3);
    double theda = (mario(a1,a2,a3)) * 0.5;
    double Len = r / sin(theda);
    Line l;
    l.p1 = p1;
    l.p2 = p2;
    if (Relation(p3,l) == 2) theda = - theda;
    Point temp;
    temp.x = p2.x - p1.x;
    temp.y = p2.y - p1.y;
    temp = Rotate(temp,theda);
    temp.x = temp.x / a1 * Len;
    temp.y = temp.y / a1 * Len;
    temp.x+=p1.x;temp.y+=p1.y;
    q = temp;
};
```

5.2 Geometry 3D

```
See hyh's paper
```

5.3 Convex Hull(Graham Algorithm)

```
//Graham-Scan
//计算平面点集的凸包
///通过测试:
//PKU 2178,ZJU 1453,PKU 2595
```

```
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;

#define EPS 1.0e-8
#define MAX_N 100
#define eq(x,y) (fabs((x)-(y))<EPS)
#define geq(x,y) (x+EPS>=y)

struct Point {
        double x,y,angle,dis;
};
bool operator<(const Point& p1,const Point& p2) {
        if(eq(p1.angle,p2.angle))
                return p1.dis<p2.dis;
        return p1.angle<p2.angle;
}

//计算叉积(p1-p0)*(p2-p0)
double times(Point p0,Point p1,Point p2) {
        return((p1.x-p0.x)*(p2.y-p0.y)-(p1.y-p0.y)*(p2.x-p0.x));
}
/*
Graham扫描求点集凸包
输入:pts 点集数组
 n 点集大小
  输出: con 凸包上的顶点数组，按逆时针排序
   m 凸包上的顶点数目
*/
void Graham(Point pts[],int n,Point con[],int& m) {
        int i,k;
        m=0;
        if(n<3) {
                con[0]=pts[0];
                con[1]=pts[1];
                return;
        }
        /*选取pts中y坐标最小的pts[k],
          如果这样的点有多个，则取最左边的一个
        */
        k=0;
        for(i=1;i<n;i++) {
                if(eq(pts[i].y,pts[k].y)) {
                        if(pts[i].x<=pts[k].x) k=i;
                }else if(pts[i].y<pts[k].y) k=i;
        }
        swap(pts[0],pts[k]);
        for(i=1;i<n;i++) {
```

```
                pts[i].angle=atan2(pts[i].y-pts[0].y,pts[i].x-pts[0].x);
                pts[i].dis=(pts[i].x-pts[0].x)*(pts[i].x-pts[0].x)+
                        (pts[i].y-pts[0].y)*(pts[i].y-pts[0].y);
        }
        sort(pts+1,pts+n);
        con[m++]=pts[0];
        for(i=1;i<n;i++) {
                //如果有极角相同的点，只去相对于pts[0]最远的一个
                if(i+1<n&&eq(pts[i].angle,pts[i+1].angle))
                        continue;
                if(m>=3)
                        while(geq(times(con[m-2],pts[i],con[m-1]),0)) m--;
                con[m++]=pts[i];
        }
}

int main() {
        int i,n,m;
        Point pts[MAX_N],con[MAX_N];
        scanf("%d",&n);
        for(i=0;i<n;i++)
                scanf("%lf %lf",&pts[i].x,&pts[i].y);
        Graham(pts,n,con,m);
        for(i=0;i<m;i++)
                printf("%lf %lf\n",con[i].x,con[i].y);
        return 0;
}
```

5.4 binary div to determin whether all points is lay the same side of a line

```
//二分快速判断平面上的点集是否在一条直线的同一侧
//writen by chenkun
//通过测试:pku 1912
//输入p: 平面点集
// 每次查询的直线(p1,p2)
// 调用solve(0,m-1)返回结果,m为点集凸包顶点数
// true:点集在直线的同一侧
// false:否则
#include "stdio.h"
#include "stdlib.h"
#include "math.h"
#define MAXN 100000
#define EPS 1.0e-6

struct point
{
    double x, y;
} p[MAXN+1],h[MAXN+1],h2[MAXN+1],p1,p2;
```

```
typedef struct point point;
typedef int (*compfn)(const void*,const void*);
int compare(point *a, point *b) {
  if((a->x-b->x<-EPS)||((fabs(a->x-b->x)<=EPS)&&a->y-b->y<-EPS))
    return -1;
  if((a->x-b->x>EPS)||((fabs(a->x-b->x)<=EPS)&&a->y-b->y>EPS))
    return 1;
  return 0;
}
double distance(const point& p1,const point& p2) {
   return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double multiply(const point& sp,const point& ep,const point& op) {
   return((sp.x-op.x)*(ep.y-op.y)-(ep.x-op.x)*(sp.y-op.y));
}

int partition(point a[],int p,int r) {
        int i=p,j=r+1,k;
        double ang,dis;
        point R,S;
        k=(p+r)/2;//防止快排退化
        R=a[p];
        a[p]=a[k];
        a[k]=R;
        R=a[p];
        dis=distance(R,a[0]);
        while(1) {
                while(1) {
                        ++i;
                        if(i>r) {
                                i=r;
                                break;
                        }
                        ang=multiply(R,a[i],a[0]);
                        if(ang>0)
                                break;
                        else if(ang==0) {
                                if(distance(a[i],a[0])>dis)
                                break;
                        }
                }
                while(1) {
                        --j;
                        if(j<p) {
                                j=p;
                                break;
                        }
                        ang=multiply(R,a[j],a[0]);
                        if(ang<0)
```

```
                                break;
                        else if(ang==0) {
                                if(distance(a[j],a[0])<dis)
                                        break;
                        }
                }
                if(i>=j)break;
                S=a[i];
                a[i]=a[j];
                a[j]=S;
        }
        a[p]=a[j];
        a[j]=R;
        return j;
}

void anglesort(point a[],int p,int r) {
   if(p<r) {
      int q=partition(a,p,r);
      anglesort(a,p,q-1);
      anglesort(a,q+1,r);
   }
}

void Graham_scan(point PointSet[],point ch[],int n,int &len) {
      int i,k=0,top=2;
      point tmp;
      //选取PointSet中y坐标最小的点PointSet[k],
      //如果这样的点有多个.则取最左边的一个
      for(i=1;i<n;i++)
            if(PointSet[i].x<PointSet[k].x||
            (PointSet[i].x==PointSet[k].x)&&(PointSet[i].y<PointSet[k].y))
               k=i;
      tmp=PointSet[0];
      PointSet[0]=PointSet[k];
      PointSet[k]=tmp; //现在PointSet中y坐标最小的点在PointSet[0]
      /* 对顶点按照相对PointSet[0]的极角从小到大进行排序，极角相同
         的按照距离PointSet[0]从近到远进行排序 */
      anglesort(PointSet,1,n-1);
      if(n<3) {
        len=n;
        for(int i=0;i<n;i++) ch[i]=PointSet[i];
        return ;
      }

      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;
}

double cross(point a,point b, point c) {
    return ((b.x-a.x)*(c.y-b.y)-(c.x-b.x)*(b.y-a.y));
}

//计算直线(rp1,rp2)与(p1,p2)的交点res
void cross_point(point rp1,point rp2,point& res) {
        if(fabs(rp1.x-rp2.x)<=EPS) {
                res.x=rp1.x;
                res.y=p1.y+(p2.y-p1.y)*(res.x-p1.x)/(p2.x-p1.x);
                return;
        }
        if(fabs(p1.x-p2.x)<=EPS) {
                res.x=p1.x;
                res.y=rp1.y+(rp2.y-rp1.y)*(res.x-rp1.x)/(rp2.x-rp1.x);
                return;
        }
        double a=p1.y;
        double b=(p2.y-p1.y)/(p2.x-p1.x);
        double c=rp1.y;
        double d=(rp2.y-rp1.y)/(rp2.x-rp1.x);
        res.x=(a-c-b*p1.x+d*rp1.x)/(d-b);
        res.y=a+b*(res.x-p1.x);
}

bool solve(int l,int r) {
        point cro,p3;
        double e;
        int mid=(l+r)>>1;
        if(l>r) return false;
        if(fabs((h[mid].x-h[1].x)*(p2.y-p1.y)-
                (p2.x-p1.x)*(h[mid].y-h[1].y))<=EPS) {
                e=cross(p1,h[1],h[mid]);
                if(l==r&&fabs(e)>EPS) return false;
                if(fabs(e)<=EPS) {
                        return true;
                }else if(e<0) {
                        return solve(l,mid-1);
                }else {
                        return solve(mid+1,r);
                }
        }
        cross_point(h[1],h[mid],cro);
        if(cro.x>=h[1].x&&cro.x<=h[mid].x&&
        ((cro.y>=h[1].y&&cro.y<=h[mid].y)||
        (cro.y>=h[mid].y&&cro.y<=h[1].y))){
```

```
                return true;
        }
        if(cro.x>h[mid].x) {
                p3.x=h[mid].x+p2.x-cro.x;
                p3.y=h[mid].y+p2.y-cro.y;
                double c1=cross(h[1],h[mid],p3);
                double c2=cross(p3,h[mid],h[mid+1]);
                if(c1*c2>=0) {
                        return solve(mid+1,r);
                }
                c1=cross(h[mid-1],h[mid],p3);
                c2=cross(p3,h[mid],h[1]);
                if(c1*c2>=0) {
                        return solve(l,mid-1);
                }
                return false;
        }else{
                p3.x=h[1].x+p2.x-cro.x;
                p3.y=h[1].y+p2.y-cro.y;
                double c1=cross(h[mid],h[1],p3);
                double c2=cross(p3,h[1],h[(int)h[0].x-1]);
                if(c1*c2>=0) {
                        return solve(mid+1,r);
                }
                c1=cross(h[2],h[1],p3);
                c2=cross(p3,h[1],h[mid]);
                if(c1*c2>=0) {
                        return solve(l,mid-1);
                }
                return false;
        }
}

int main() {
        int n;
        int i,len;
        scanf("%d",&n);
        for (i=0;i<n;i++) {
                scanf("%lf %lf",&p[i].x,&p[i].y);
        }
        Graham_scan(p,h2,n,len);
        h[0].x=len+1;
        h[1].x=h2[0].x;
        h[1].y=h2[0].y;
        for(int i=len-1;i>=1;i--) h[len+1-i].x=h2[i].x,h[len+1-i].y=h2[i].y
        while(scanf("%lf %lf %lf %lf",&p1.x,&p1.y,&p2.x,&p2.y)!=EOF) {
                if(solve(2,(int)h[0].x-1)) {
                        printf("BAD\n");
                }else{
                        printf("GOOD\n");
```

```
                    }
        }
        return 0;
}
```

Chapter 6

Mathematics

6.1 Basic Number Theory

```
//扩展欧几里德
//使得m*x+n*y=1,返回m,n的最大公约数
long long exgcd(long long m,long long & x,long long n,long long & y)
{       long long x1,y1,x0,y0;
        x0=1;y0=0;
        x1=0;y1=1;
          long long r=(m%n+n)%n;
        long long q=(m-r)/n;
        x=0;y=1;
        while(r)
        {   x=x0-q*x1;y=y0-q*y1;  x0=x1;y0=y1;
             x1=x;y1=y;
             m=n;n=r;r=m%n;
            q=(m-r)/n;
        }
        return n;
}

//快速取模mod exp,calculates a pow b mod n
int ModExp(int a,int b,int n) {
        int c=1,d=a;
        while(b) {
                if(b&1) c=(c*d)%n;
                d=(d*d)%n;
                b>>1;
        }
        return c;
}

//求解模线性方程ax=b(mod n),n¿0
void modular_linear_equation(long long a,long long b,long long n) {
        long long e,d,x,y;
```

```
        int i;
        d=exgcd(a,x,n,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 %lldn",i+1,(e+i*(n/d))%n);
        }
}

/*中国余数订立，求解模线性方程组
a=B[1](mod W[1])
a=B[2](mod W[2])
........
a=B[n](mod W[n])
其中W,B已知,W[i]¿0且W[i]与W[j]互质，求a
*/
int linear_modular_equation_system(long long B[],long long W[],int k)
{
        int i;
        long long 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=exgcd(W[i],x,m,y);
                a=(a+y*m*B[i])%n;
        }
        if(a>0) return a;
        else return a+n;
}

/*==========================================================
                  二分求矩阵冥
==========================================================*/
//list[0][][]:单位矩阵
//list[1][][]:结果矩阵,初始为单位矩阵
void matrixmul(int mode) {        //mode 0:乘单位矩阵1:自乘
        for(int i=1;i<=nodes;i++)
                for(int j=1;j<=nodes;j++)
                        temp[i][j]=0;
        for(int i=1;i<=nodes;i++)
          for(int j=1;j<=nodes;j++)
            for(int k=1;k<=nodes;k++)
                temp[i][j]+=graph[1][i][k]*graph[mode][k][j];
        for(int i=1;i<=nodes;i++)
          for(int j=1;j<=nodes;j++)
                graph[1][i][j]=temp[i][j]%100000;
}

void binarymul(int n) {
```

```
        if(n==1) return;
        binarymul(n/2);
        matrixmul(1);
        if(n%2==1) matrixmul(0);
}

//求矩阵的n次方:binarymul(n);

/*=============================================
            素数测试随机化算法
==============================================*/
//调用isprim(n,s)返回结果，true为素数
//n为要判断的数,s为随机测试次数
long long a,N;

long long mimod(long long n) {
        if(n==1) return a%N;
        long long t=mimod(n/2);
        long long r=(t*t)%N;
        if(n%2==1) r=(r*a)%N;
        return r;
}

bool isprim(long long n,int s) {
        N=n;
        for(int i=0;i<s;i++) {
                a=rand()%(n-1)+1;
                if(mimod(n-1)!=1)
                        return false;
        }
        return true;
}

/*=============================================
            数的因子分解
==============================================*/
void PrimeFactor(long long n) {
    int i,j;
        long long res=1;

    if (n < 2) {
        return n;
    }
    i = 2;
    while (i <= (int) (sqrt ((double) (n)))) {
        j = 0;
        while(n % i == 0) {
            n /= i;
            j ++;
        }
```

```
        if(j){
                        //因子i出现j次
        }
        i ++;
    }
         if (n!= 1) {
        //最后还有个因子n
    }
}
```

6.2 Gaussian Elimination

```
see hyh's paper
```

6.3 Romberg Numberical Integration

```
double F(double t) {
        return (R/(f*cost(t)+h));
}

double romberg(double a,double b,double eps){
        vector<double> R;
        int k=-1;
        double r=0.5*(b-a)*(F(a)+F(b));
        R.push_back(r);
        do{
                k+=1;
                r=0.0;
                for(int i=0;i<pow(2.0,k);i++)
                        r+=F(a+(b-a)*(i+0.5)/pow(2.0,k));
                r*=(b-a)/pow(2.0,k+1);
                r+=0.5*R[k];
                R.push_back(r);
                for(int m=0;m<=k;m++)
                  R[k-m]=(pow(4.0,m+1)*R[k+1-m]-R[k-m])/(pow(4.0,m+1)-1);
        }while(fabs(R[0]-R[1])>eps);
        return R[0];
}
```