Contents

1. Grafos.

1.1. Busquedas.

1.1.1. *DFS y BFS.*

```
int parent[MAX];
int seen[MAX];
```

```
bool BFS(int s, int t) {
  queue <int> q;
  memset(seen, 0, sizeof(seen));
  parent[s] = -1;
  seen[s] = 1;
  q.push( s );
  while(!q.empty())
  {
    s = q.front(); q.pop();
    if (s == t) break;
    for (int i=0; i<n; i++)
      if (!seen[i] && C[s][i] > 0)
          parent[i] = s,
        q.push( i );

  }
  return seen[t] != 0;
}
```

1.1.2. *Deteccion de puntos de articulacion (Articulation Points).*

```
int tim;
bool artic[MV];
int d[MV];
int low[MV];
int seen[MV];
int parent[MV];
void dfs(int x){
    seen[x]=1;
    low[x]=d[x]=tim++;

    for(int i=0;i<deg[x];i++)
        if(!seen[ady[x][i]]){
            parent[ady[x][i]]=x;
            dfs(ady[x][i]);
            if(low[x]>low[ady[x][i]])
                low[x]=low[ady[x][i]];
            if(low[ady[x][i]]>=d[x])
                artic[x]=true;
        }else if(ady[x][i]!=parent[x])
            low[x]<?=d[ady[x][i]];
}
void dfs_f(int n){
    memset(artic,0,sizeof(artic));
    memset(seen,0,sizeof(seen));
    memset(parent,-1,sizeof(parent));
    tim=0;

    for(int i=0;i<n;i++)
         if(!seen[i]){
             seen[i]=1;
             low[i]=d[i]=tim++;
             int nh=0;
             for(int j=0;j<deg[i];j++)
                 if(!seen[ady[i][j]]){
                     nh++;
                     parent[ady[i][j]]=i;
                     dfs(ady[i][j]);
                 }
             if(nh>=2) artic[i]=true;
         }
}
```

1.1.3. *Deteccion de puentes (Bridges).*

```
void dfs(int x){
    seen[x]=1;
    low[x]=d[x]=tim++;

    for(int i=0;i<deg[x];i++)
        if(!seen[ady[x][i]]){
            parent[ady[x][i]]=x;
            dfs(ady[x][i]);
            if(low[x]>low[ady[x][i]])
                 low[x]=low[ady[x][i]];
            if(low[ady[x][i]]==d[ady[x][i]]){
                //x - ady[x][i] es un puente
            }else if(ady[x][i]!=parent[x])
                low[x]<?=d[ady[x][i]];
}
void dfs_f(int n){
    memset(seen,0,sizeof(seen));
    memset(parent,-1,sizeof(parent));
    tim=0;

    for(int i=0;i<n;i++)
        if(!seen[i])   dfs(i);
}
```

1.1.4. *Ciclo de Euler.* Existencia del ciclo de Euler. Un grafo tiene un ciclo de Euler si y solo si (i) esta conectado y (2) todos sus vertices tienen grado par.

Existencia de un camino de Euler. Un grafo tiene un camino de Euler si y solo si (i) esta conectado y (2) exactamente 2 de sus vertices tienen grado impar, los cuales constituyen el inicio y el fin del camino.

Algoritmo para encontrar el camino de Euler. Encontrar ciclos de vertices disjuntos e irlos uniendo.

```
////////////////////////////
int tour(int x){
    int w,v=x;
    bool stilledges=true;

    while(stilledges){
        stilledges=false;
        for(int i=0;i<MB;i++)
           if(mat[v][i])
               {w=i;
                stilledges=true;
                break;}
        if(stilledges){
            S.push(v);
            mat[v][w]--;
            mat[w][v]--;
            v=w;
        }
    }
    return v;
}
//x contiene el vertice por el cual empieza el ciclo
void find_euler(int x){
    int v=x;
    bool f=true; //solo se usa para imprimir bien

    while(tour(v)==v&&!S.empty()){
       v=S.top();S.pop();
       if(f){
           cout<<x+1<<' '<<v+1<<endl;
           cout<<v+1<<' ';
       }else{
           cout<<v+1<<endl;
           if(!S.empty()) cout<<v+1<<' ';
       }
       f=false;
    }
}
```

1.1.5. *Determinar si un grafo es bipartito.* Determinar si un grafo es bipartito es equivalente a determinar si el grafo puede ser coloreado con 2 colores, de tal forma que no haya dos vertices compartiendo el mismo color, lo cual a su vez es equivalente a determinar si el grafo no tiene un ciclo de longitud impar.

```
memset(col,-1,sizeof(col));
if(dfs(0,0)) //entonces es bipartito
bool dfs(int x,int color){
        col[x]=(color+1)%2;
        for(int i=0;i<deg[x];i++)
                if(col[adj[x][i]]==-1){
                        if(!dfs(adj[x][i],col[x])) return false;
                }else if(col[adj[x][i]]==col[x]) return false;
        return true;
}
```

1.2. DAGs.

1.2.1. *Ordenamiento Topologico.*

```
int indeg[MAXV]; //contiene el numero de aristas que entran al vertice
queue<int> q;
for(int i=0;i<n;i++) if(!indeg[i]) q.push(i);

while(!q.empty()){
    int x=q.front(); q.pop();
    //Aqui poner codigo para procesar el vertice (x)
    for(int i=0;i<deg[x];i++){
        indeg[adj[x][i]]--;
        if(!indeg[adj[x][i]]) q.push(adj[x][i]);
    }
}
```

1.3. Arbol de expansion Minima.

1.3.1. *Algoritmo de Prim.* Cambiar ciclo de relajacion de Dijkstra por:

```
for(int i=0;i<deg[u];i++){
    int v=adj[u][i];
    if(w[u][v]<d[v]){
        d[v]=w[u][v];
        parent[v]=u;
    }
}
```

1.3.2. *Algoritmo de Kruskal.*

```
for(int i=0;i<n;i++) make_set(i);
sort(lista.begin(),lista.end(),cmp); // cmp (peso a)<(peso b)
suma = 0;
for(int i = 0; i < lista.size();i++)
    if(find_set(lista[i].u)!= find_set(lista[i].v)) {
        suma+=lista[i].w;
        union_set(lista[i].u,lista[i].v);
    }
```

1.3.3. *Segundo arbol de expansion minima.*

1.4. Caminos mas cortos.

1.4.1. *Bellman-Ford (Pesos negativos, no ciclos negativos).*

1.4.2. *Dijkstra (Pesos positivos).*

```
#define INF 0x3f3f3f3f
int d[MAXV];     //distancias
int parent[MAXV];  //antecesor
void dijkstra(int s, int n){
    memset(d,0x3f,sizeof(d)); //d[i]=INF
    memset(seen,0,sizeof(seen));
    d[s]=0;
    parent[s]=-1;
    while(1){
        int u=-1,dmin=INF;
        for(int i=0;i<n;i++)
            if(!seen[i]&&d[i]<dmin)
                {dmin=d[i]; u=i;}
        if(u==-1) break;
        seen[u]=1;
        //Se utilizan listas de adyacencia
        for(int i=0;i<deg[u];i++){
            int v=adj[u][i];
            if(d[u]+w[u][v]<d[v]){
                d[v]=d[u]+w[u][v];
                parent[v]=u;
            }
        }
    }
}
//Posibles adiciones=cola de prioridad, imprimir ca
```

1.4.3. *Floyd-Warshall (Pesos negativos, no ciclos negativos).*

```
void imprime(int r, int d)
{
  if(p[r][d]==-1)
  {
    printf ("%d",d);
    return;
  }
  imprime(r,p[r][d]);
  printf (" %d",d);
}
for(int i = 0; i<n;i++)
 for(int j = 0; j<n;j++)
  if( d[i][j] != 0 && d[i][j]!=INF )
    p[i][j] = i;
  else
   p[i][j] = -1;
```

```
for(int k = 0; k<n;k++)
 for(int i = 0; i<n;i++)
  for(int j = 0; j<n;j++)
    if(d[i][k]+d[k][j] < d[i][j])
    {
      d[i][j] = d[i][k]+d[k][j];
      p[i][j] = p[k][j];
    }
//Modificacion min max (frogger)
//minimum necessary edge weight
//over all possible paths
if(i!=k&&j!=k)
    d[i][j]=min(d[i][j],
    max(d[i][k],d[k][j]));
//Modificacion max min (turista)
if(i!=k&&j!=k)
    d[i][j]=max(d[i][j],
    min(d[i][k],d[k][j]));
```

1.5. Flujos en Redes (Network Flow).

1.5.1. *Bipartite Matching.*

```
//Numero de nodos a la izquierda
#define M 50
//Numero de nodos a la derecha
#define N 50
//graph[i][j]=1,si hay una arista de i a j
bool graph[M][N];
bool seen[N];
//Contienen -1 si no hay matching
int matchL[M], matchR[N];
```

```
int n,m;
bool bpm( int u )
{
    for(int v=0;v<n;v++) if(graph[u][v])
    {
        if(seen[v]) continue;
        seen[v] = true;
        if(matchR[v]<0||bpm(matchR[v]))
        {
            matchL[u] = v;
```

```
            matchR[v] = u;
            return true;
        }
    }
    return false;
}
//Ejemplo de uso
int main(){
    while(cin>>m>>n){
        memset(graph,0,sizeof(graph));
        for(int i=0;i<m;i++)
            for(int j=0;j<n;j++)
                cin>>graph[i][j];
        memset( matchL, -1, sizeof( matchL ) );
        memset( matchR, -1, sizeof( matchR ) );
        int cnt = 0;

        for(int i = 0; i < m; i++ ){
            memset( seen, 0, sizeof( seen ) );
            if( bpm( i ) ) cnt++;
        }
        cout<<cnt<<endl;
    }
    return 0;
}
```

1.5.2. *MaxFlow (Edmond-Karps).*

```
#define MV 100
//Numero maximo de aristas saliendo de un vertice
#define INF 0x3f3f3f3f
int c[MV][MV];
int f[MV][MV];
int adj[MV][MV];
int deg[MV];
int parent[MV];
int seen[MV];
int bfs_edmond(int s,int t){
   //inicializar busqueda
   memset(seen,0,sizeof(seen));
   parent[s]=-1;
   seen[s]=1;
   //Hacer BFS
   queue<int> q;
   q.push(s);
   int x; bool found=false;
   while(!q.empty()&&!found){
      x=q.front(); q.pop();
      for(int i=0;i<deg[x]&&!found;i++)
          if(!seen[adj[x][i]]&&
          (c[x][adj[x][i]]-f[x][adj[x][i]])>0){
             parent[adj[x][i]]=x;
             seen[adj[x][i]]=1;
             if(adj[x][i]==t)
                found=true;
             q.push(adj[x][i]);
          }
   }
   if(!found) {return 0; }
   //Obtener el maximo volumen que puede ser enviado
   //a traves del camino encontrado
   int res=INF;
   x=t;
   while(parent[x]!=-1){
      res<?=(c[parent[x]][x]-f[parent[x]][x]);
      x=parent[x];
   }
   return res;
}
void augment_path(int s,int t,int v){
   int x=t;
   while(x!=s){
       f[parent[x]][x]+=v;
       f[x][parent[x]]-=v;
       x=parent[x];
   }
}
Uso: llenar matriz de adyacencia, establecer flujo
a cero,y llenar capacidades (soporta grafos no dirigidos),
y a continuacion:
int v,tot=0;
while((v=bfs_edmond(s,t))) {tot+=v; augment_path(s,t,v);}
en tot, queda el flujo maximo que se puede enviar de s a t
```

1.5.3. *s-t Minimum Cut.* Max Flow Min Cut Teorema. El valor de un flujo maximo es igual a la capacidad de un corte minimo.

1.5.4. *All pairs-Minimum Cut.* Se puede encontrar el minimo corte de un grafo fijando un vertice y corriendo n-1 flujos a partir de ese vertice a los demas, y tomando el minimo de estos.

Otra forma sin aplicar flujos, es mediante el algoritmo de Stoer-Wagner, el cual es el siguiente:

```
// Maximum number of vertices in the graph
#define NN 256
// Maximum edge weight
//(MAXW * NN * NN must fit into an int)
#define MAXW 1000
// Adjacency matrix and some internal arrays
int g[NN][NN], v[NN], w[NN], na[NN];
bool a[NN];
int minCut( int n )
{
   // init the remaining vertex set
   for( int i = 0; i < n; i++ ) v[i] = i;
   // run Stoer-Wagner
   int best = MAXW * n * n;
   while( n > 1 )
   {
      // initialize the set A and vertex weights
      a[v[0]] = true;
      for( int i = 1; i < n; i++ ){
            a[v[i]] = false;
            na[i - 1] = i;
            w[i] = g[v[0]][v[i]];
      }
      // add the other vertices
      int prev = v[0];
```

```
    for( int i = 1; i < n; i++ ){
        // find the most tightly connected non-A vertex
        int zj = -1;
        for( int j = 1; j < n; j++ )
            if(!a[v[j]]&&(zj<0||w[j]>w[zj]))
               zj = j;
        // add it to A
        a[v[zj]] = true;
        // last vertex?
        if( i == n - 1 ){
           // remember the cut weight
           best <?= w[zj];
           // merge prev and v[zj]
           for( int i = 0; i < n; i++ )
                                        g[v[i]][prev] = g[prev][v[i]]
                                                      += g[v[zj]][v[i]];
           v[zj] = v[--n];
           break;
        }
        prev = v[zj];
        // update the weights of its neighbours
        for(int j=1;j<n;j++) if(!a[v[j]])
           w[j] += g[v[zj]][v[j]];
     }
  }
  return best;
 }
```

Forma de uso: llenar matriz de adyacencia g y establecer n al numero de vertices del grafo.

1.5.5. *MinCost MaxFlow (Tambien MaxCost MaxFlow).*

```
using namespace std;
#define MV 250         //Numero de vertices de la red
int adj[MV][MV];       //Lista de adyacencia
int deg[MV];           //Grado de cada vertice
int f[MV][MV];         //flujos de las aristas
int cap[MV][MV];       //capacidad  de las aristas
double cost[MV][MV];   //costos de las aristas (tipo depende del prob.)
double d[MV];          //Vector distancia (Dijkstra)
int par[MV];           //Vector padre (Dijkstra)
int seen[MV];          //Vector seen(Djikstra)
double pi[MV];         //funcion de etiquetado para los nodos
//#define INF  1000000000000000000LL    //(long long)
#define INFD    1e9                     //(double)
#define INF  0x3f3f3f3f               //(int)
//------------------------------------------------
bool djikstra(int s,int t,int n){
        for(int i=0;i<n;i++) d[i]=INFD;
        memset(par,-1,sizeof(par));
        memset(seen,0,sizeof(seen));
        par[s]=s;
        d[s]=0;

        while(1){
                int u=-1; double mmin=INFD;
                for(int i=0;i<n;i++) if(!seen[i]&&d[i]<mmin){
                        mmin=d[i];
                        u=i;
                }
                if(u==-1) break;

                seen[u]=1;
                for(int i=0;i<deg[u];i++){
                        int v=adj[u][i];
                        if(seen[v]) continue;

                        //checar si hay flujo de u a v
                        if(f[u][v]<cap[u][v]&&d[v]>d[u]+(pi[u]+cost[u][v]-pi[v])){
                                d[v]=d[u]+(pi[u]+cost[u][v]-pi[v]);
                                par[v]=u;
                        }
                }
        }

        for(int i=0;i<n;i++) if(pi[i]<INFD) pi[i]+=d[i];
        return par[t]>=0;
}
```

```
//------------------------------------------------------
void init_pi(int s,int n){      //Bellman-Ford para maxcost-maxflow
   for(int i=0;i<n;i++) pi[i]=INFD;
   pi[s]=0;

   for(int i=0;i<n-1;i++)
    for(int j=0;j<n;j++)
     for(int k=0;k<deg[j];k++)
        if((f[j][adj[j][k]]<cap[j][adj[j][k]])&&
           (pi[adj[j][k]]>pi[j]+cost[j][adj[j][k]]))
                   pi[adj[j][k]]=pi[j]+cost[j][adj[j][k]];
}
//------------------------------------------------------
int mcmf(int s,int t,int n, double &fcost,bool mincost){
        memset(f,0,sizeof(f));
        if(mincost)    //Si es mincost entonces funcion de etiquetado=0
                for(int i=0;i<n;i++) pi[i]=0;
        else           //Si es maxcost entonces inicializar con Bellman-Ford
                init_pi(s,n);

        int flow=0; fcost=0;
        while(djikstra(s,t,n)){
                //Obtener el cuello de botella
                int bot=INF;
                int v=t,u;
                while(v!=s){
                        u=par[v];
                        bot<?=(cap[u][v]-f[u][v]);
                        v=u;
                }

                //Actualizar el flujo y el costo
                v=t;
                while(v!=s){
                        u=par[v];
                        f[u][v]+=bot; f[v][u]-=bot;
                        fcost+=(bot*cost[u][v]);
                        v=u;
                }
                flow+=bot;
        }
        return flow;
}
//Ejemplo de uso
int main(){
       memset(deg,0,sizeof(deg));    //establecer el grado a 0
       int source=0,sink=n;
       //Leer el grafo y almacenar valores para capacidad y costo
       adj[source][deg[source]++]=i;
       cap[source][nodo]=cca;
       cost[source][nodo]=20;
       // Asi como tambien crear la arista que va al reves
       adj[nodo][deg[nodo]++]=source;
       cap[nodo][source]=0;
       cost[nodo][source]=-20;

       bool bmincost=true; //si se quiere maxcost, establecer false
       double fcost;       //Valor del costo
       int flow=mcmf(source,sink,n+1,fcost,bmincost);
}
```

Notas: - Si se quiere obtener el costo maximo, entonces antes de realizar el algoritmo se deben negar los costos.

-El algoritmo anterior asume que si $(u, v) \in E$, entonces $(v, u) \notin E$, por lo que si se quieren representar grafos no dirigidos, se debe dividir cada vertice en dos nuevos vertices, el primero al que se conectaran todas las aristas que entran al vertice, al segundo se conectaran todas las aristas que salen del vertice, y ademas se unen estos dos nuevos vertices con una arista dirigida del primero al segundo, con costo de 0 y capacidad infinita.

1.5.6. *Stable Marriage Problem.*

1.6. Otros Problemas.

1.6.1. *Minimum Vertex Cover.* Un vertex cover de un grafo no dirigido G=(V,E), es un subconjunto V' de V que contiene al menos una de las terminaciones de una arista, el problema denominado minimum vertex cover tree busca el vertex cover con el minimo numero de elementos, en grafos en general es un problema NP.

Grafos Bipartitos (Teorema de Konig). En un grafo bipartito G=(V,E), la maxima cardinalidad de cualquier matching iguala la minima cardinalidad de cualquier vertex cover de G. Es decir, si se quiere encontrar el minimum vertex cover de cualquier grafo es suficiente con encontrar el maximum matching del grafo.

1.7. Teoria de Grafos.

Formula de Euler.

$V - E + F = 2$

Numero de arboles diferentes etiquetados.

$NAr = sn^{n-s-1}$, s numero de componentes conexas.

2. Programacion Dinamica.

2.1. LCS..

```
#define MATCH 1
#define L 2
#define U 3
#define M 500
int len[M][M],p[M][M];
///////// Obtener longitud de LCS
int lcs(char X[],char Y[]) {
  int m=strlen(X);
  int n=strlen(Y);
  for (int i=1;i<=m;i++) len[i][0]=0;
  for (int j=0;j<=n;j++) len[0][j]=0;
  for (int i=1;i<=m;i++)
    for (int j=1;j<=n;j++) {
      if (X[i-1]==Y[j-1]) {
        len[i][j]=len[i-1][j-1]+1;
        p[i][j]=MATCH; /* match, incrementar */
      }
      else if (c[i-1][j]>=c[i][j-1]) {
        len[i][j]=len[i-1][j];
        p[i][j]=R; /* de arriba */
      }
      else {
        len[i][j]=len[i][j-1];
        p[i][j]=L; /* de la izquierda */
      }
    }
  return len[m][n];
}
////////////Imprimir LCS
void print_lcs(int m,int n){
        if(m==0||n==0) return;
        if(p[m][n]==MATCH) {
                print_lcs(m-1,n-1);
                cout<<arrm[m]<<' ';
        }else if(p[m][n]==L) print_lcs(m,n-1);
        else print_lcs(m-1,n);
}
```

2.2. LIS (nlogk).

2.3. Multiplicacion optima de matrices.

2.4. Edit Distance.

```
#define MATCH 0
#define SUBST 1
#define DELETE 2
#define INSERT 3
#define ML 85 //Maxima longitud de las cadenas
int parent[ML][ML];
int cost[ML][ML];
char s[85],t[85];
//Edit distance para transformar s a t.
int ed_distance(void){
    int n=strlen(s);
    int m=strlen(t);

    for(int i=0;i<=n;i++) {cost[0][i]=i;
                           parent[0][i]=DELETE;}
    for(int i=0;i<=m;i++) {cost[i][0]=i;
                           parent[i][0]=INSERT;}
    parent[0][0]=-1;

    for(int i=1;i<=m;i++)
        for(int j=1;j<=n;j++){
```

```
            if(s[j-1]==t[i-1]){
                cost[i][j]=cost[i-1][j-1];
                parent[i][j]=MATCH;
            }else{
                cost[i][j]=cost[i-1][j-1]+1;
                parent[i][j]=SUBST;
            }
            if(cost[i][j-1]+1<cost[i][j]){
                cost[i][j]=cost[i][j-1]+1;
                parent[i][j]=DELETE;
            }
            if(cost[i-1][j]+1<cost[i][j]){
                cost[i][j]=cost[i-1][j]+1;
                parent[i][j]=INSERT;
            }
        }
    return cost[m][n];
}
```

```
//Inicializar nin=1,off=0,m=strlen(t),n=strlen(s)
void print_edit(int m,int n){
     if(parent[m][n]!=-1){
          switch(parent[m][n]){
              case MATCH:  print_edit(m-1,n-1); break;
              case SUBST:  print_edit(m-1,n-1);
                  printf("%d Replace %d,%c\n",
                          nin++,n+off,t[m-1]); break;
              case DELETE: print_edit(m,n-1);
                  printf("%d Delete %d\n",nin++,
                          n+off); off--; break;
              case INSERT: print_edit(m-1,n);
                  printf("%d Insert %d,%c\n",nin++,
                          n+off+1,t[m-1]); off++; break;
          }
     }
}
```

2.5. Coin Change.

```
long nway[MAX+1];
int coin[5] = {20,25,10,5,1};
void main() {
  int i, j, n, v = 5,c;
  nway[0] = 1;
  for(i = 0; i < v; i++) {
      c = coin[i];
      for(j = c; j <= n;j++)
       nway[j] += nway[j-c];
    }
  solucion en nway[n]
}
```

2.6. Knapsack 0-1 (Mochila).

```
#define MAXC    1000 //Maxima capacidad
#define MAXO    200  //Numero maximo de objetos
int vi[MAXO];        //Valores de los objetos
int wi[MAXO];        //Pesos de los objetos
int v[MAXO][MAXC];   //Mochila
//Mochila con DP, regresa valor maximo
int knap(int n,int c){
   for(int i=0;i<=n;i++) v[i][0]=0;
   for(int i=0;i<=c;i++) v[0][i]=0;

   for(int i=1;i<=n;i++)
      for(int j=1;j<=c;j+=1){
         v[i][j]=v[i-1][j];
         if(j-wi[i-1]>=0)
            v[i][j]>?=(vi[i-1]+v[i-1][j-wi[i-1]]);
      }
   return v[n][c];
}
//Imprimir solucion.
void print_knap(int i,int c){
   if(i>0&&c>0){
      if(v[i][c]==v[i-1][c]) //no se encuentra
         print_knap(i-1,c);
      else{                  //si se encuentra
         print_knap(i-1,c-wi[i-1]);
         //Imprimir dato sobre el objeto
         totw+=wi[i-1];
      }
   }
}
```

2.7. KMP (String Matching).

2.8. Problema del cartero chino (Chinese Postman Problem).

```
int best[1<<14];
int floyd[25][25];
int lodd[20];
int deg[25];
int ngen;
int solve(int x){
    if(best[x]==-1){
        best[x]=INF;
        for(int i=0;i<ngen;i++)
            for(int j=i+1;j<ngen;j++)
```

```
                if((x>>i)%2&&(x>>j)%2)
                    best[x]<?=(floyd[lodd[i]][lodd[j]]+solve(x-(1<<i)-(1<<j)));
    }
    return best[x];
}
int main(){
    int n,e;
    while(scanf("%d",&n)==1&&n){
        memset(best,-1,sizeof(best));
        memset(deg,0,sizeof(deg));
        best[0]=0;
        scanf("%d",&e);
        memset(floyd,0x3f,sizeof(floyd));
        int res=0;

        for(int i=0;i<e;i++){
            int a,b,c;
            scanf("%d %d %d",&a,&b,&c); a--; b--;
            res+=c;
            deg[a]++;
            deg[b]++;
            floyd[a][b]<?=c;
            floyd[b][a]<?=c;
        }

        for(int k=0;k<n;k++) for(int i=0;i<n;i++) for(int j=0;j<n;j++)
            floyd[i][j]<?=(floyd[i][k]+floyd[k][j]);

        int n2=0;
        for(int i=0;i<n;i++)
           if(deg[i]%2) lodd[n2++]=i;
        ngen=n2;

        printf("%d\n",res+solve((1<<n2)-1));
   }
   return 0;
}
```

3. Estructuras de datos.

3.1. Hashing (para cadenas).

```
// primo cercano al tamaño la tabla
const int NHASH = 29989;
const int MULT = 31;
typedef struct node *nodeptr;
typedef struct node {
   char *word;
   int count;
   nodeptr next;
}node;
nodeptr bin[NHASH];
unsigned int hash(char *p) {
   unsigned int h = 0;
   for(; *p;p++)
      h = MULT * h + *p;
   return (h % NHASH);
}
void incword(char *s) {
   unsigned int h = hash(s);
   nodeptr p;
   for(p = bin[h]; p != NULL; p = p->next)
      if(strcmp(s,p->word)==0) {
            (p->count)++;
            return;
         }
   p = (nodeptr)malloc(sizeof(node));
   p->count = 1;
   p->word = (char *)malloc(strlen(s)+1);
   strcpy(p->word,s );
   p->next = bin[h];
   bin[h] = p;
}
void inicializa() {
   for(int i = 0; i < NHASH; i++)
      bin[i]=NULL;
}
void imprime() {
 nodeptr p;
 for(int i = 0; i < NHASH; i++)
    for( p = bin[i]; p!=NULL;p = p->next)
       printf("%s %d\n",p->word,p->count);
}
// Para ejecutar hay que inicilazar
// y luego insertar cada palabra en incword
```

3.2. Estructura Union-Find.

```
int p[MAX],rank[MAX];
void make_set(int x){
     p[x] = x;
     rank[x]=0;
}
void link(int x, int y) {
    if(rank[x]>rank[y])
        p[y] = x;
    else {
        p[x] = y;
        if(rank[x]==rank[y])
            rank[y] = rank[y] + 1;
    }
}
int find_set(int x) {
    if(x != p[x])
         p[x] = find_set(p[x]);
    return p[x];
}
void union_set(int x, int y) {
    link(find_set(x),find_set(y));
}
```

3.3. Funcion para suma de dos strings.

```
string suma(string a, string b)
{
   int l = 1 + (a.length() > b.length() ?
               a.length() : b.length());
   string c(l, '0');
   a = string(l - a.length(), '0') + a;
   b = string(l - b.length(), '0') + b;
   int ac = 0, sum = 0;
   for (int i=l-1; i>=0; i--) {
      sum = a[i] + b[i] - '0'  - '0' + ac;
      c[i] = (sum % 10) + '0';
      ac = sum / 10;
   }
   while(c.length() > 0 && c[0] == '0')
       c.erase( c.begin() );
   return c;
}
```

3.4. Funcion para multiplicacion de dos strings.

```
string mult(string a, string b)
{
   string m, n, r = "0";
   int d, ac, prod;
   if (a.length() > b.length()) m = a, n = b;
   else m = b, n = a;
   while( n.length() > 0 ) {
      d = n[n.length()-1] - '0';
      string aux = "0" + m;
      ac = 0;
      for (int i=aux.length()-1; i>=0; i--) {
         prod = d * (aux[i] - '0') + ac;
         aux[i] = (prod % 10) + '0';
         ac = prod / 10;
      }
      r = suma( r, aux );
      m = m + "0",
      n.erase( n.end()-1 );
   }
   return r;
}
```

3.5. BigInteger.

3.5.1. *C++.*

3.5.2. *Java.*

```
//Performing Bitwise Operations with BigInteger
// Create via a string
BigInteger bi1 = new BigInteger("1234567890123456890");
// Create via a long
  BigInteger bi2 = BigInteger.valueOf(123L);
bi1 = bi1.add(bi2);
bi1 = bi1.multiply(bi2);
bi1 = bi1.subtract(bi2);
bi1 = bi1.divide(bi2);
bi1 = bi1.negate();
int exponent = 2;
bi1 = bi1.pow(exponent);
//Parsing and Formatting a Big Integer into Binary, Octal, and Hexadecimal
BigInteger bi = new BigInteger("1023");
// Parse and format to binary
bi = new BigInteger("1111111111", 2); // 1023
String s = bi.toString(2);            // 1111111111
// Parse and format to octal
bi = new BigInteger("1777", 8);       // 1023
s = bi.toString(8);                   // 1777
// Parse and format to decimal
bi = new BigInteger("1023");          // 1023
s = bi.toString();                      // 1023
// Parse and format to hexadecimal
```

```
bi = new BigInteger("3ff", 16);       // 1023
s = bi.toString(16);                  // 3ff
// Parse and format to arbitrary radix <= Character.MAX_RADIX
int radix = 32;
bi = new BigInteger("vv", radix);     // 1023
s = bi.toString(radix);               // vv
Arrays.sort(A);
```

4. Busqueda y Ordenamiento.

4.1. MergeSort.

```
void merge(int p, int q, int r) {
    int i,j;
    int n1 = q -p + 1;
    int n2 = r -q;
    int L[n1+1],R[n2+1];
    memcpy(L,A+p,n1*sizeof(int));
    memcpy(R,A+q+1,n2*sizeof(int));
    L[n1]=0xFFFFFFF;
    R[n2]=0xFFFFFFF;
    i = 0; j = 0;
    for(int k = p; k <= r;k++)
        if(L[i]<=R[j])
            A[k]=L[i++];
        else{
            conta += n1-i;
            A[k] = R[j++];
        }
}
void mergesort(int p, int r) {
    int q;
    if(p<r) {
         q = (p+r)/2;
         mergesort(p,q);
         mergesort(q+1,r);
         merge(p,q,r);
    }
}
```

4.2. Busqueda Binaria.

```
//Regresa -1 si no se encuentra el elemento a buscar
//de otra forma regresa el indice en el que se encuentra
int binsearch(int arr[],int t,int n){
        int l=0,u=n-1,m;

        while(1){
                if(l>u) return -1;
                m=(l+u)/2;
                if(arr[m]==t) return m;
                else if(arr[m]<t) l=m+1;
                else u=m-1;
        }
}
```

5. Geometria.

5.1. Geometria Clasica.

5.1.1. *Punto de Interseccion (Segmento - Segmento, Segmento - Linea, Linea - Linea).*

```
bool SegSegInt(point a,point b,point c,point d,point p){
    double s,t,num,denom;
    denom=a[X]*double(d[Y]-c[Y])+b[X]*double(c[Y]-d[Y])+
          d[X]*double(b[Y]-a[Y])+c[X]*double(a[Y]-b[Y]);

    //Paralelos
    if(denom==0.0) return false;

    num=a[X]*double(d[Y]-c[Y])+c[X]*double(a[Y]-d[Y])+
        d[X]*double(c[Y]-a[Y]);

    s=num/denom;

    num=-(a[X]*double(c[Y]-b[Y])+b[X]*double(a[Y]-c[Y])+
        c[X]*double(b[Y]-a[Y]));

    t=num/denom;

    p[X]=a[X]+s*(b[X]-a[X]);
```

```
    p[Y]=a[Y]+s*(b[Y]-a[Y]);

    return (0.0<=s&&s<=1.0&&0.0<=t&&t<=1.0);
}
```

El codigo anterior funciona para la interseccion de dos segmentos (a,b) y (c,d), para interseccion de segmento linea modificar la ultima condicion por:

```
return (0.0<=s&&s<=1.0);
```

Y para la interseccion linea-linea, smplemente regresar true.

5.1.2. *Distancia mas cercana de un punto a una linea.*

Linea dada en forma de dos puntos (x0,y0) y (x1,y1) y el punto (x,y). La distancia es con signo,
$d(P,L) = \frac{(y_0-y_1)x+(x_1-x_0)y+(x_0y_1-x_1y_0)}{\sqrt{(x_1-x_0)^2+(y_1-y_0)^2}}$
Linea implicita de la forma f(x,y)=ax+by+c=0.
$d(P,L) = \frac{ax+by+c}{\sqrt{a^2+b^2}}$

5.1.3. *Interseccion de Rectangulos.*

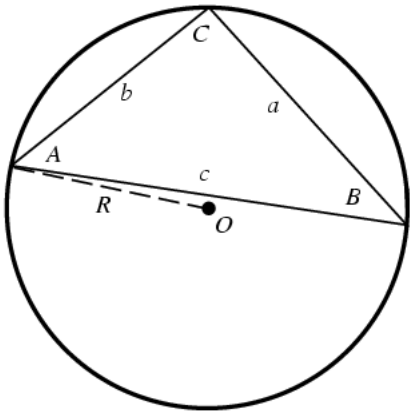
```
//left lower(xi,yi), right upper(xf,yf)
struct rect{
        int xi,xf,yi,yf;
};
bool inter_rect(rect &a,rect &b,rect &c){
        c.xi=max(a.xi,b.xi);
        c.xf=min(a.xf,b.xf);
        c.yi=max(a.yi,b.yi);
        c.yf=min(a.yf,b.yf);
        if(c.xi<=c.xf&&c.yi<=c.yf) return true;
        return false;
}
```

5.1.4. *Calculo del area total de un conjunto de rectangulos.*

```
double get_Area_Rect(int n){
    set<double> sx;
    set<double> sy;
    for(int i=0;i<n;i++) {
        sx.insert(R[i].xi); sx.insert(R[i].xf);
        sy.insert(R[i].yi); sy.insert(R[i].yf);
    }

    vector<double> vx(sx.begin(),sx.end());
    vector<double> vy(sy.begin(),sy.end());
     double res=0.0;
    for(int i=0;i<nx-1;i++){
        for(int j=0;j<ny-1;j++){
            bool inrect=false;
            for(int k=0;k<n&&!inrect;k++)
                if(R[k].xi<=vx[i]&&vx[i+1]<=R[k].xf&&
                   R[k].yi<=vy[j]&&vy[j+1]<=R[k].yf)
                       inrect=true;
            if(inrect) res+=(vx[i+1]-vx[i])*(vy[j+1]-vy[j]);
        }
    }
    return res;
}
```

5.1.5. *Circulo a traves de tres puntos.*



5.1.6. *Resolucion de Triangulos.*

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2 * R$$
$$a^2 = b^2 + c^2 - 2bc\cos A \quad \Big| \quad b^2 = a^2 + c^2 - 2ac\cos B$$
$$c^2 = a^2 + b^2 - 2ab\cos C$$

5.1.7. *Formulas utiles.* **Distancia Geodesica entre dos puntos (en la superficie de una esfera).**

```
haversine(x)=(1.0-cos(x))/2.0
a = haversine(lat2 - lat1)
b = cos(lat1) * cos(lat2) * haversine(lon2 - lon1)
c = 2 * atan2(sqrt(a + b), sqrt(1 - a - b))
d = R * c
```

donde R, es el radio de la esfera (6378 km para la tierra), Importante: convertir las latitudes y longitudes a radianes antes de aplicar la Formula. Longitud (este a oeste, 0 a 360°). Latitud Norte (+90°) a sur (-90°).

Radio del circulo inscrito en un traingulo de lados a,b,c.

$r = \frac{\sqrt{s(s-a)(s-b)(s-c)}}{s}$ donde s es el semiperimetro.

Radio del circulo que pasa por un triangulo de lados a,b,c.

$r = \frac{abc}{4\sqrt{s(s-a)(s-b)(s-c)}}$

Matriz de rotacion de puntos. Para rotar un punto (x,y) en θ grados, en el sentido counterclockwise con centro en el punto (0,0), multiplicar el punto por la siguiente matriz.

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Formula de Heron , Area de triangulo a,b,c.

$A = \sqrt{s(s-a)(s-b)(s-c)}$

5.2. Geometria Computacional.

5.2.1. *Area de triangulo y funciones Left(), Collinear, Right (CounterClockwise Order), Dot.*

```
typedef TipoPunto point[2];
#define X 0
#define Y 1
//Nota: regresa 2 veces el area del triangulo
//formado por los puntos a,b y c
TipoPunto Area2(point a,point b,point c){
    return a[X]*(b[Y]-c[Y])-a[Y]*(b[X]-c[X])+(b[X]*c[Y]-b[Y]*c[X]);
}
bool Left(point a,point b,point c){
     return Area2(a,b,c)>0;
}
bool Lefton(point a,point b,point c){
     return Area2(a,b,c)>=0;
}
bool Collinear(point a,point b,point c){
     return Area2(a,b,c)==0;
}
bool Right(point a,point b,point c){
     return Area2(a,b,c)<0;
}
```

```
TipoPunto Dot(point a,point b){
     return a[X]*b[X]+a[Y]*b[Y];
}
```

5.2.2. *Interseccion de segmentos (booleana).*

```
//Interseccion propia (interseccion en el interior de los segmentos)
bool int_prop(point a,point b,point c,point d){
        //Eliminar casos impropios
        if(Collinear(a,b,c)||Collinear(a,b,d)||
           Collinear(c,d,a)||Collinear(c,d,b))
                    return false;

        return ((Left(a,b,c))xor(Left(a,b,d)))&&
               ((Left(c,d,a))xor(Left(c,d,b)));
}
//Checar si un punto (c) esta en el interior de un segmento (a,b)
bool between(point a,point b,point c){
        if(!Collinear(a,b,c)) return false;

        if(a[X]!=b[X]) return (a[X]<=c[X]&&c[X]<=b[X])||(a[X]>=c[X]&&c[X]>=b[X]);
        else return (a[Y]<=c[Y]&&c[Y]<=b[Y])||(a[Y]>=c[Y]&&c[Y]>=b[Y]);
}
//Funcion que checa si dos segmentos se intersectan
bool intersect(point a,point b,point c,point d){
        if(int_prop(a,b,c,d)) return true;
        else if(between(a,b,c)||between(a,b,d)||
                        between(c,d,a)||between(c,d,b))
                               return true;
        return false;
}
```

5.2.3. *Distancia mas cercana de un punto a un segmento.*

```
double dist_point_to_segment( point p, point a,point b,point pclose)
{
    double v[2]={b[X]-a[X],b[Y]-a[Y]};
    double w[2]={p[X]-a[X],p[Y]-a[Y]};
    double c1 = Dot(w,v);
    if ( c1 <= 0 ){
        pclose[X]=a[X];
        pclose[Y]=a[Y];
        return dist(p,a);
    }
    double c2 = Dot(v,v);
    if ( c2 <= c1 ){
        pclose[X]=b[X];
        pclose[Y]=b[Y];
        return dist(p,b);
    }
    double t = c1 / c2;
    pclose[X]=a[X]+t*v[X];
    pclose[Y]=a[Y]+t*v[Y];
    return dist(p,pclose);
}
```

5.2.4. *Area de Poligono.*

```
double area(polygon P,int n){
    double total=0.0;
    int i,j;
    for(i=0;i<n;i++){
        j=(i+1)%n;
        total+=P[i][X]*P[j][Y]-P[j][X]*P[i][Y];
    }
    return fabs(total)/2.0;
}
```

Nota: "total" es positivo si los vertices estan ordenados counterclockwise, y negativo si clockwise

5.2.5. *Area de Poligono 3D..*

5.2.6. *Punto en Poligono.*

```
bool InPoly(point &q,polygon P,int n){
    int rcross,lcross,i1; rcross=lcross=0;
    bool rstrad,lstrad;
    double x;
    for(int i=0;i<n;i++){
        //El punto es un vertice
        if(P[i][X]==q[X]&&P[i][Y]==q[Y])
            return true;
        i1=(i-1+n)%n;
        rstrad=(P[i][Y]>q[Y])!=(P[i1][Y]>q[Y]);
        lstrad=(P[i][Y]<q[Y])!=(P[i1][Y]<q[Y]);
        if(rstrad||lstrad){
            x=(q[Y]*(P[i][X]-P[i1][X])
                    -P[i1][Y]*P[i][X]
                    +P[i1][X]*P[i][Y])/
                    (P[i][Y]-P[i1][Y]);
            if(rstrad&&x>q[X]) rcross++;
            if(rstrad&&x<q[X]) lcross++;
        }
    }
    //El punto esta en una arista
    if((rcross%2)!=(lcross%2)) return true;

    //Estrictamente interior
    if((rcross%2)==1) return true;
    else return false;
}
```

5.2.7. *Cerco Convexo (Algoritmo de Graham).*

```
typedef int TipoPunto ;
typedef TipoPunto point[DIM];
struct p{
    int vnum;
    point v;
    bool del;
} P[MAXP];
p* pila[MAXP];
void Swap(int i, int j){
     p temp; memcpy(&temp,&P[i],sizeof(p));
     memcpy(&P[i],&P[j],sizeof(p));
     memcpy(&P[j],&temp,sizeof(p));
}
//Encontrar punto mas abajo a la derecha
void find_lowest(int n){
    int imin=0;
    for(int i=1;i<n;i++)
        if((P[i].v[Y]<P[imin].v[Y])||
        (P[i].v[Y]==P[imin].v[Y]
         &&P[i].v[X]>P[imin].v[X]))
             imin=i;
    Swap(0,imin);
}
//Funcion de comparacion
int comp(const void *aa,const void *bb){
    p *a=(p *)aa;
    p *b=(p *)bb;
    TipoPunto ar=Area2(P[0].v,a->v,b->v);
    if(ar>0){
          return -1;
    }else if(ar<0){
          return 1;
    }else{
        TipoPunto dx0,dx1,dy0,dy1;
        dx0=a->v[X]-P[0].v[X];
        dx1=b->v[X]-P[0].v[X];
        dy0=a->v[Y]-P[0].v[Y];
        dy1=b->v[Y]-P[0].v[Y];
        if(dx0<0) dx0=-1*dx0;
        if(dx1<0) dx1=-1*dx1;
        if(dy0<0) dy0=-1*dy0;
        if(dx1<0) dy1=-1*dy1;

        TipoPunto dx=dx0-dx1;
        TipoPunto dy=dy0-dy1;

        if(dx<0||dy<0) {a->del=true; return -1;}
        if(dx>0||dy>0) {b->del=true; return 1;}
        if(a->vnum>b->vnum) b->del=true;
        else a->del=true;
        return 0;
    }
}
//Borrar Repetidos
int Squash(int n){
    int i,j; i=j=0;
    for(;j<n;j++)
        if(!P[j].del){
            memcpy(&P[i],&P[j],sizeof(p));
            i++;
        }
    return i;
}
int Graham(int n){
    find_lowest(n);
    qsort(&P[1],n-1,sizeof(p),comp);
    n=Squash(n);

    int t=0;
    pila[t++]=&P[0];
    pila[t++]=&P[1];

    int i=2;
    while(i<n)
        if(Left(pila[t-2]->v,pila[t-1]->v,P[i].v)){
            pila[t++]=&P[i];
            i++;
        }else t--;
    return t;
}
```

5.2.8. *Triangulacion de poligonos (Van Gogh).*

5.2.9. *Poligonos Lattice y Teorema de Pick.*

```
for(int i = 0; i < verts.size(); i++) {
    j = (i+1)%verts.size();
    dx = abs(verts[j].first-verts[i].first);
    dy = abs(verts[j].second-verts[i].second);
    tot += gcd(dy,dx);
  }
Tot contiene el número de puntos en la frontera del poligono
A(P) = I(P) + B(P)/2 - 1
```

5.2.10. *Par mas cercano de un conjunto de puntos.*

```
typedef double TipoPunto;
#define INF 1e50
struct point{
   TipoPunto x,y;
   int id;
}X[10020],Y[10020];
bool lr[10005];
bool sortx(const point  &a,const point &b){
    return a.x<b.x;
}
bool sorty(const point  &a,const point &b){
    return a.y<b.y;
}
TipoPunto dist(point &a,point &b){
   TipoPunto dx=a.x-b.x;
   TipoPunto dy=a.y-b.y;
   return dx*dx+dy*dy;
}
TipoPunto solve(point X[],point Y[],int t){
   if(t==2)
      return dist(X[0],X[1]);

   if(t==3){
      TipoPunto res=INF;
      res<?=dist(X[0],X[1]);
      res<?=dist(X[0],X[2]);
      res<?=dist(X[1],X[2]);
      return res;
   }

   point Xl[t/2+1],Xr[t/2+1],
         Yl[t/2+1],Yr[t/2+1];

   memset(lr,0,sizeof(lr));
   int cl,cr;
   cl=0;cr=0;

   for(int i=0;i<t;i++)
      if(i<(t+1)/2)
         Xl[cl++]=X[i];
      else {
         Xr[cr++]=X[i];
         lr[X[i].id]=true;
      }
   cl=0; cr=0;
   for(int i=0;i<t;i++)
      if(!lr[Y[i].id])
            Yl[cl++]=Y[i];
      else
            Yr[cr++]=Y[i];

   TipoPunto dl=solve(Xl,Yl,cl);
   TipoPunto dr=solve(Xr,Yr,cr);
   TipoPunto d=min(dl,dr);

   point YP[t];

   double med=(X[t/2-1].x
              +X[t/2].x)/2;

   int ty=0;
   for(int i=0;i<t;i++){
      double v=fabs(Y[i].x-med);
      if(v*v<=d)
         YP[ty++]=Y[i];
   }

   for(int i=0;i<ty;i++)
      for(int j=i+1;j<ty&&j<i+7;j++)
         d<?=dist(YP[i],YP[j]);

   return d;
}
int main(){
     int n;
     while(scanf("%d",&n)==1&&n){
     for(int i=0;i<n;i++) {
         scanf("%lf %lf",&X[i].x,&X[i].y);
         Y[i].x=X[i].x; Y[i].y=X[i].y;
         X[i].id=Y[i].id=i;
     }
     if(n==1)
        {printf("INFINITY\n"); continue;}
     sort(X,X+n,sortx);
     sort(Y,Y+n,sorty);

     double dmin=sqrt(solve(X,Y,n));
     if(dmin<10000.0) printf("%.4lf",dmin);
     else printf("INFINITY");
     printf("\n");
     }
  return 0;
}
```

5.2.11. *Mínimo rectángulo encapsulador.* Primero se tiene que calcular el ConvexHull. Y se usará los datos de la pila resultante.

```
double smallestBoundingRectangle (int n)
{
    /// Se toma cada elemento de la pila
    double mmin = 0;
```

```
    double flag2 = true;
    double inc = 1;
    double a = 0, b = 90; /// Topes para la primera iteración
    while (inc>=1e-12)
    {
        double area  = 0;
        double ta,tb;
        int flag = true;
        for (double teta=a; teta<=b;teta+=inc)
        {
            double angle = (PI*teta)/180.0;
            point temp;
            double x1,x2,y1,y2;
            bool first=true;
            for (int i=0;i<n;i++)
            {
                 temp[X] = pila[i]->v[X]*cos(angle) - pila[i]->v[Y]*sin(angle);
                 temp[Y] = pila[i]->v[X]*sin(angle) + pila[i]->v[Y]*cos(angle);
                 if (first)
                 {
                        x1 = temp[X];x2 = temp[X];y1 = temp[Y];y2 = temp[Y];
                        first = !first;
                 }
                 x1<?=temp[X];
                 x2>?=temp[X];
                 y1<?=temp[Y];
                 y2>?=temp[Y];
            }
            if (flag)
            {
              area = (x2-x1) * (y2-y1);
              ta = teta - inc;
              tb = teta + inc;
              flag = !flag;
            }
            if ((x2-x1) * (y2-y1) < area)
            {
                area = (x2-x1) * (y2-y1);
                ta = teta - inc;
                tb = teta + inc;
            }
        }
        a = ta;
        b = tb;
        if (flag2)
        {
            mmin = area;flag2= !flag2;
        }
        mmin<?=area;
        inc/=10;
    }
    return mmin;
}
```

5.2.12. *Distancia más cercana entre 2 polígonos.*

```
#define MAXP 105
#define X 0
#define Y 1
#define DIM 2
#define INF 1E18;
#define MAXV 30
typedef double TipoPunto;
typedef TipoPunto point[DIM];
const double PI = 2*acos(0);
struct poly{
```

```
    int vnum;
    point v;
    bool del;
};
poly P[MAXV][MAXP];
int nP[MAXV];
double adj[MAXV][MAXV];
TipoPunto dist(point a,point b){
    TipoPunto dx=a[X]-b[X];
    TipoPunto dy=a[Y]-b[Y];
    return sqrt(dx*dx+dy*dy);
}
TipoPunto Dot(point a,point b){
    return a[X]*b[X]+a[Y]*b[Y];
}
double dist_point_to_segment( point p, point a,point b,point pclose)
{
    TipoPunto v[2]={b[X]-a[X],b[Y]-a[Y]};
    TipoPunto w[2]={p[X]-a[X],p[Y]-a[Y]};
    TipoPunto c1 = Dot(w,v);
    if ( c1 <= 0 ){
        pclose[X]=a[X];
        pclose[Y]=a[Y];
        return dist(p,a);
    }
    double c2 = Dot(v,v);
    if ( c2 <= c1 ){
        pclose[X]=b[X];
        pclose[Y]=b[Y];
        return dist(p,b);
    }
    double t = c1 / c2;
    pclose[X]=a[X]+t*v[X];
    pclose[Y]=a[Y]+t*v[Y];
    return dist(p,pclose);
}
/// i y j son los poligonos del arreglo de poligonos P
/// el arreglo nP lleva el número de puntos
double minimumDistancePolygons(int a, int b)
{
    double minima = INF;
    point temp;
    for (int i=0;i<nP[a];i++)
    {
        for (int j=0;j<nP[b];j++)
        {
             // Condicion acorde al problema
            if ((b==0||b==1) && j==nP[b]-1)
              break;
            minima<?= dist_point_to_segment (P[a][i].v,P[b][j].v,P[b][(j+1)%nP[b]].v,temp);
        }
    }
    for (int i=0;i<nP[b];i++)
    {
        for (int j=0;j<nP[a];j++)
        {             // Condicion acorde al problema
            if ((a==0||a==1) && j==nP[a]-1) break;
            minima<?= dist_point_to_segment (P[b][i].v,P[a][j].v,P[a][(j+1)%nP[a]].v,temp);
       }
    }
    return minima;
}
```

6. Teoria de Numeros.

6.1. Aritmetica Modular.

6.1.1. *Modulo y division de un entero grande sobre un long long (s/n o s%n).*

```
string coc;
long long modu=0;
for(int i=0;i<s.size();i++){
    modu=(modu*10+s[i]-'0');
    coc+=((modu/n)+'0');
    modu%=n;
}
int ind=0;
while(ind<s.size()&&coc[ind]=='0') ind++;
coc=coc.substr(ind);
if(coc.empty()) coc="0";
```

6.2. Exponenciacion rapida.

```
int cuadrado(int n) { return n * n; }
int exponenciacion_rapida(int x, int n){
  if (n == 0)
    return 1;
  else if (n % 2 == 0)
    return cuadrado(exponenciacion_rapida(x,n/2));
  else
    return x*exponenciacion_rapida(x,n-1);
}
```

6.3. Factorizacion prima de un entero.

```
map<int,int> fact_primo(int x){
        map<int,int> res;
        while(x%2==0) {x/=2; res[2]++;}
        int c=3;
        while(c<=sqrt(double(x))+1)
                if(x%c==0) {x/=c; res[c]++;}
                else c+=2;
        if(x>1) res[x]++;
        return res;
}
```

6.4. GCD Extendido.

```
int gcd (int p, int q, int *x, int *y)
{
    int x1,y1;
    long g;
    if (q>p) return (gcd(q,p,y,x));
    if (q==0){
        *x = 1;
        *y = 0;
        return p;
     }
    g=gcd (q,p%q,&x1,&y1);
    *x = y1;
    *y = (x1 - (p/q)*y1);
    return g;
}
```

6.5. Criba de Eratostenes.

6.5.1. *En un rango.*

```
void sieve(int L,int U) {
    int i,j,d;
    d=U-L+1;
    bool *flag=new bool[d];
    for (i=0;i<d;i++)
```

```
        flag[i]=true;
    for (i=(L%2!=0);i<d;i+=2)
        flag[i]=false;
    for (i=3;i<=sqrt(U);i+=2) {
        if (i>L && !flag[i-L])
            continue;
        j=L/i*i;
        if (j<L) j+=i;
        if (j==i) j+=i;
        j-=L;
        for (;j<d;j+=i)
           flag[j]=false;
    }
    if (L<=1) flag[1-L]=false;
    if (L<=2) flag[2-L]=true;
    /* output the result */
    for (i=0;i<d;i++) if(flag[i])
         cout << (L+i) << " ";
    cout << endl;
}
```

6.6. Funcion Phi de Euler (Numero de primos relativos a un numero).

6.6.1. *De un solo numero.*

```
int phi_euler(int x){
        map<int,int> m=fact_primo(x);
        map<int,int>::iterator it;
        int res=x;
        for(it=m.begin();it!=m.end();it++)
                {res/=(it->first); res*=(it->first-1);}
        return res;
}
```

6.6.2. *En un rango determinado.*

```
int euler[M];
char prime[M];
int phi_euler2(int x){
    memset(prime,-1,sizeof(prime));
    criba[0]=criba[1]=1;
    for(int i=0;i<M;i++)
        euler[i]=i;
    for(int i=0;i<M/2;i++)
        if(prime[i])
            for(int j=i+i;j<M;j+=i){
                prime[j]=0;
                euler[j]/=i;
                euler[j]*=(i-1);
            }
}
```

6.7. Formulas y teoremas utiles.

6.7.1. *Numero de divisores de un entero n.* n es el numero con su factorizacion por primos. nd es el numero de divisores.

$n = p_1^{\alpha_1} p_2^{\alpha_2} ... p_n^{\alpha_n}$

$nd = (\alpha_1 + 1)(\alpha_2 + 1)...(\alpha_n + 1)$

6.7.2. *Fracciones continuadas.*

6.7.3. *Ternas Pitagoricas.* Las soluciones primitivas positivas de $x^2 + y^2 = z^2$ con y par son $x = r^2 - s^2$, $y = 2rs$, $z = r^2 + s^2$ donde r y s son enteros arbitrarios de paridad opuesta con r $>$ s $>$ 0 y (r,s) = 1.

6.7.4. *Division de factorial sobre un primo.* $f(n,p) = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + ...$

6.7.5. *Teorema chino del Residuo.* Sea el sistema de congruencias.
$x \equiv a_1 mod m_1$, $x \equiv a_2 mod m_2$, . . . , $x \equiv a_k mod m_k$ donde los m's son coprimos entre si.
el sistema tiene como solucion:
$t = a_1 x_1 t_1 + a_2 x_2 t_2 + ... + a_k x_k t_k$
donde
$t_i = \frac{m_1 m_2 ... m_k}{m_i}$
y x_i es un numero tal que $1 <= x_i < m_i$ y $t_i x_i \equiv 1 mod m_i$

6.7.6. *Simbolo de Legendre.* Para todo a tal que (a,m)=1, a recibe el nombre de residuo cuadratico modulo m si la congruencia $x^2 \equiv amodm$ tiene una solucion. Si no tiene solucion, entonces a se llama no residuo cuadratico modulo m.

Si p denota un primo impar y (a,p) = 1, el simbolo de Legendre $\left(\frac{a}{p}\right)$se define como 1 si a es un residuo cuadratico, -1 si a es no residuo cudratico modulo p. Regresa 0 si p divide a a.

Nota: si p == 2, siempre existe un residuo cuadrático.
$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} modp$

6.7.7. *Conjetura de Golbach.* Todo numero par puede ser escrito como la suma de dos numeros primos (a excepcion del 2).

6.7.8. *Primeras cifras de n a la k.*

```
x = k * log10(n)
mantisa(x) = x - floor(x)
signif = pow(10.0, mantisa(x)) * 100  // para obtener las 3 más significativas
```

7. Combinatoria.

7.1. **Combinaciones.**

7.1.1. *C(n,k).*

```
void div_by_gcd(long &a, long &b){
    long g = gcd(a,b);
    a /= g;
    b /= g;
}
long C(int n, int k){
    long num = 1;
    long den = 1;
    long tomult,todiv;
    if(k > n/2)
        k = n-k;
    for(int i = k;i;i--){
        tomult = n-k+i;
        todiv = i;
        div_by_gcd(tomult,todiv);
        div_by_gcd(num,todiv);
        div_by_gcd(tomult,den);
        num *= tomult;
        den * = todiv;
    }
    return num/den;
}
```

7.1.2. *Triangulo de Pascal (DP)..*

```
//m contiene la fila maxima que se quiere calcular
void pascal(int m){
        C[0][0]=1;
        for(int i=1;i<=m;i++){
                C[i][0]=C[i][i]=1;
                for(int j=1;j<i;j++)
                        C[i][j]=C[i-1][j-1]+C[i-1][j];
        }
```

```
}
```

7.2. Resolucion de Recurrencias Lineales usando Exponenciacion de una matriz (QMatrix).

7.2.1. *Fibonacci Exponencial.*

```
          n
| 0 1 | = | fn-2 fn-1 |
| 1 1 |   | fn-1 fn   |
```

7.3. Conteo.

7.3.1. *Combinaciones con repeticiones.* El numero de combinaciones de n objetos tomados de r en r, con repeticiones, es $C(n+r-1, r)$. Lo cual es equivalente a:

(1) El numero de soluciones enteras de la ecuacion
$x_1 + x_2 + ... + x_n = r$ con $x_i \geq 0$

(2) El numero de formas en que r objetos identicos se pueden distribuir entre n recipientes distintos.

7.4. Formulas utiles.

7.4.1. *Recurrencia de Catalan (y algunos de sus posibles significados).* $C_0 = 1$, $C_{n+1} = \frac{2(2n+1)}{n+2} C_n$

Primeros numeros de Catalan

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790

7.5. Generacion Combinatoria.

7.5.1. *KLexSubset.*

```
int T[30]; int U[30];
for(int j=1;j<=n;j++) T[j]=j; //Init
int KSubsetLexSucc(int k, int n){
    memcpy(U,T,sizeof(T));
    int i = k;
    while((i>=1)&&(T[i]==n-k+i))
        i--;
    if(i==0)  return 0;
    for(int j = i;j <=k;j++)
        U[j]=T[i]+1+j-i;
    memcpy(T,U,sizeof(U));
    return 1;
}
// El subconjunto que sigue queda en
// las primeras k posiciones de T
```

8. Otros.

Archivos en Java.

```
import java.io.*;
import java.util.*;
class test {
  public static void main (String [] args) throws IOException {
    // Use BufferedReader rather than RandomAccessFile; it's much faster
    BufferedReader f = new BufferedReader(new FileReader("test.in"));
                                          // input file name goes above
    PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter("test.out")));
    // Use StringTokenizer vs. readLine/split -- lots faster
    StringTokenizer st = new StringTokenizer(f.readLine());
                                                  // Get line, break into tokens
    int i1 = Integer.parseInt(st.nextToken());    // first integer
    int i2 = Integer.parseInt(st.nextToken());    // second integer
    out.println(i1+i2);                           // output result
    out.close();                                  // close the output file
    System.exit(0);                               // don't omit this!
  }
}
// Para leer de la entrada estandar usar
```

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Plantilla de Codigo.

```
#include <iostream>
#include <algorithm>
#include <cctype>
#include <cmath>
#include <cstdlib>
#include <string>
#include <cstring>
#include <cstdio>
#include <iomanip>
#include <map>
#include <vector>
#include <queue>
#include <stack>
#include <set>
#include <sstream>
#include <utility>
using namespace std;
int main(){
    freopen("a.in","r",stdin);
    freopen("a.out","w",stdout);

    return 0;
}
```

Codigo Miscelaneo.

GCD

```
int GCD(int a,int b) {if(a%b==0) return b; else return GCD(b,a%b);}
```

```
Probar si un año es bisiesto
bool leap(int y) {
    return y % 4 == 0 && (y % 100 != 0 || y % 400 == 0);}
```

```
PI
PI = 2*acos(0);
```
