# Implementación en ISyS de medidas para la caracterización de redes complejas.

Autor: T.S.U. Hebert Avendaño.

Tutor: Dr. Kay Tucci

Universidad de Los Andes

Mérida - Venezuela

2010

# Objetivo

Diseñar e implementar el módulo de ISyS para el cálculo de algunas cantidades estadísticas que permitan caracterizar a las redes complejas.

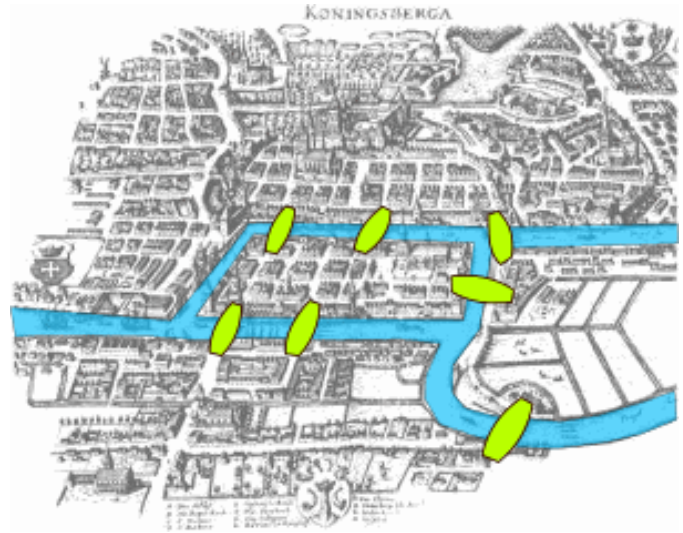
- Seleccionar las medidas a implementar.
- Implementar en ISyS las medidas para la caracterización de redes complejas.
- Calcular las diferentes medidas para caracterizar las redes complejas ya implementadas en ISyS.
- Comparar los resultados obtenidos con los resultados teóricos y experimentales.
- Sistematizar la información haciendo uso de las normas de sintaxis establecidas para lograr la estabilidad y cohesión con los módulos ya existentes en ISyS.

# Índice

1) Grafos.
2) Representación de grafos.
3) Diseño e implementación.
4) Resultados.
5) Conclusiones y Recomendaciones.
6) Bibliografía.

# Leonard Euler y los 7 puentes

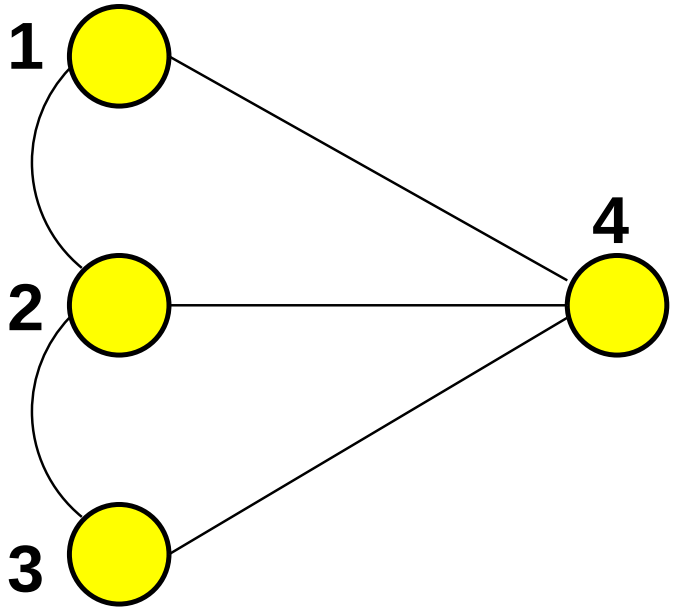
$(1707 - 1783)$

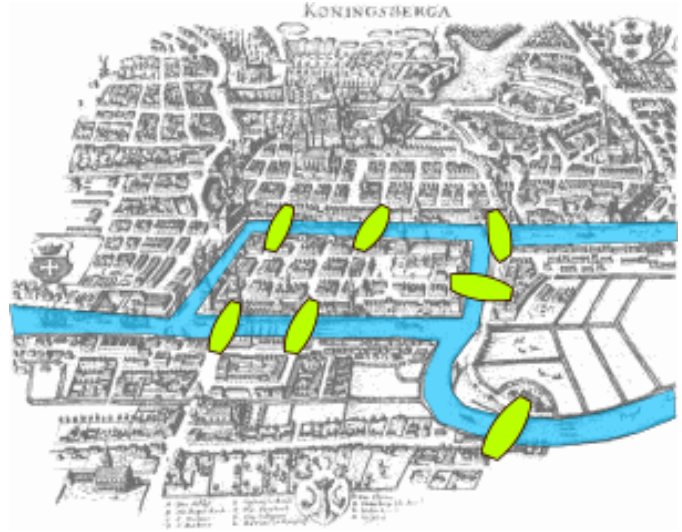



Konigsberg

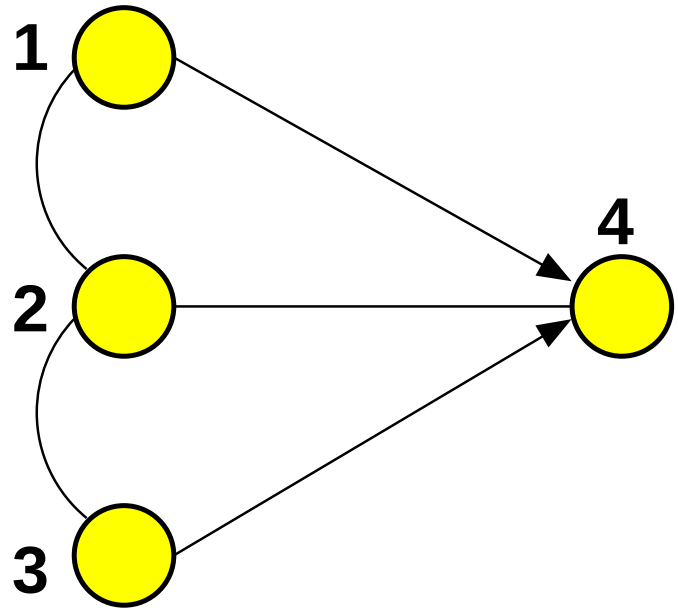
# Grafos

## Grafo Simple



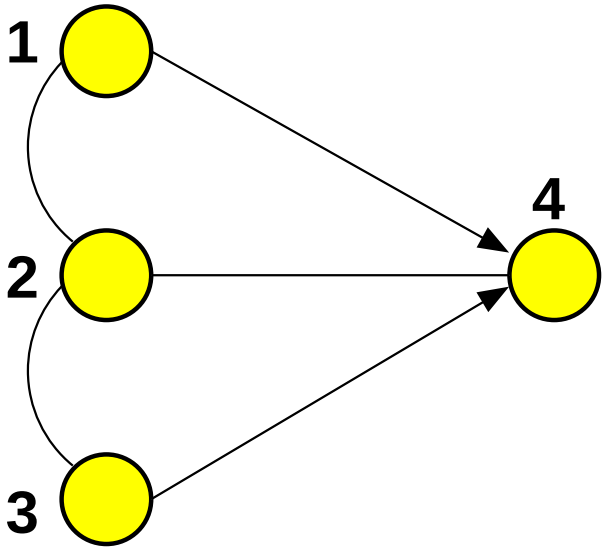



# Grafo Dirigido



# Representación de Grafos

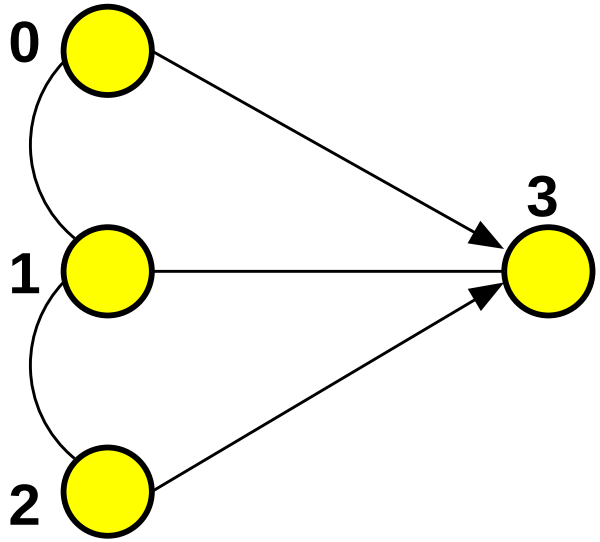
## Matriz de Adyacencia

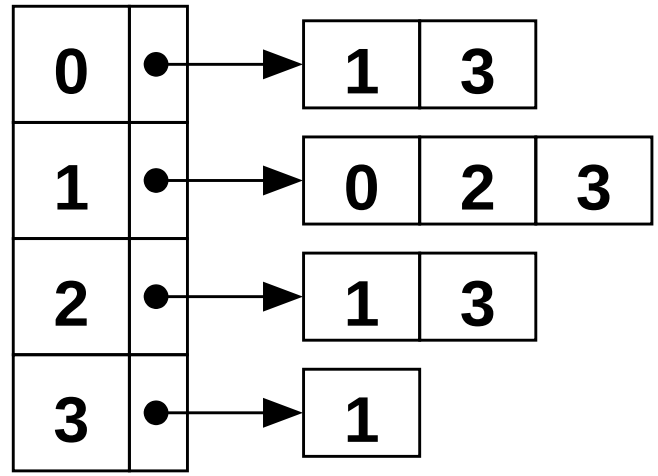



$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

¿Cómo ISyS almacena los datos?

# Conjunto de Vecinos




**node[0].n_nei = 2**

**node[0].nb[1] = 3**

**node[1].nb[1] = 2**

**node[3].nb[0] = 1**

# Diseño e Implementación

## Medidas

| | | |
|---|---|---|
| Grados entrantes. | Distancia promedio. | Vulnerabilidad sin el nodo $i$. |
| Grados salientes. | Eficiencia. | Vulnerabilidad máxima. |
| Distribución de grados salientes. | Eficiencia sin el nodo $i$. | Coeficiente de agrupamiento. |
| Distribución de grados entrantes. | Media armónica. | Espectro de Autovalores. |
| Entropía. | | |

# Normas de Sintaxis

```
/* ------------------------------------------------------
A_. NOMBRE
        Avedistance
B_. SINOPSIS
        "hebert.h"
        double Avedistance(void)
        Ademas utiliza el la funciones:
        ...
C_. DESCRIPCION
        Esta funcion calcula la el promedio de las distancias
        mas cortas entre cada par de vertices.
D_. VALOR QUE DEVUELVE
        En caso de exito, retorna un valor de tipo double con el
        ...
*/
extern double Avedistance(void);
```

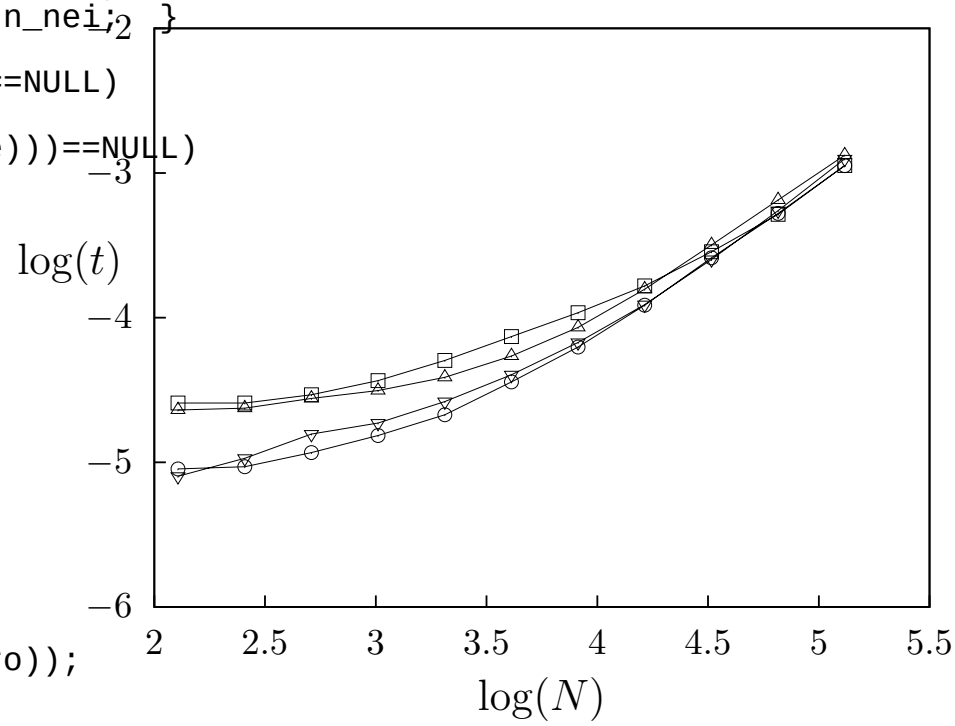
# Normas de Sintaxis

```
PROGRAM=          demo
SRC=              salida.c
ISyS_HOME=        /opt/ISyS
X11_DIR=          /usr/lib
LATTICE =         $(ISyS_HOME)/lib/Lattices/smallworld.o
DYNAMIC =         ISyS_HOME)/lib/TestLattice/scan.o
NETDYNAMIC =      $(ISyS_HOME)/lib/NetDynamics/static.o
XDISPLAY =        $(ISyS_HOME)/lib/Display/nopgplot.o \
                  $(ISyS_HOME)/lib/Display/color.o
STATISTIC =       ../lib/Avedistance.o
CF_OBJS =         $(LATTICE) \
                  $(DYNAMIC) \
                  $(NETDYNAMIC) \
                  $(STATISTIC) \
                  $(XDISPLAY)
CC=               cc
F77=              gfortran
CFLAGS=           -O2 -W -Wall -Wtraditional
FFLAGS=           -O2 -Wall
CF_INCL=          -I../include
CF_LIBS=          -llapack
```

# Implementacion: Grados

```
void entropy(){
  long n;   double entro,ff,*w; long double s = 0;
  long double ss = 0;
long kMin = nn,kMax = 0, *h, nHisto;
  for(n=0; n<nn; n++){
    if(node[n].n_nei > kMax) kMax = node[n].n_nei;
    if(node[n].n_nei < kMin) kMin = node[n].n_nei; }
  nHisto = kMax-kMin+1;
  if((h=(long*)calloc(nHisto,sizeof(long)))==NULL)
      error_message(2);
  if((w=(double*)calloc(nHisto,sizeof(double)))==NULL)
      error_message(2);
  for(n=0; n<nHisto; n++) {
    h[n] = 0;   w[n] = 0; }
  for(n=0; n<nn; n++){
    s += node[n].n_nei;
    ss += node[n].n_nei*node[n].n_nei;
    h[node[n].n_nei-kMin]++;  }
  ff = 0;
  for(n=0; n<nHisto; n++){
    ff += ((double)h[n])/nn;
    w[n] = ((double)h[n])/nn; }
  entro = 0;
  for(n=0; n<nHisto; n++){
     if(w[n] > 0){
entro += w[n]*log10(w[n]);  }  }
  printf(" Entropia \t%lf \n\n",(-1.0)*(entro));
}
```

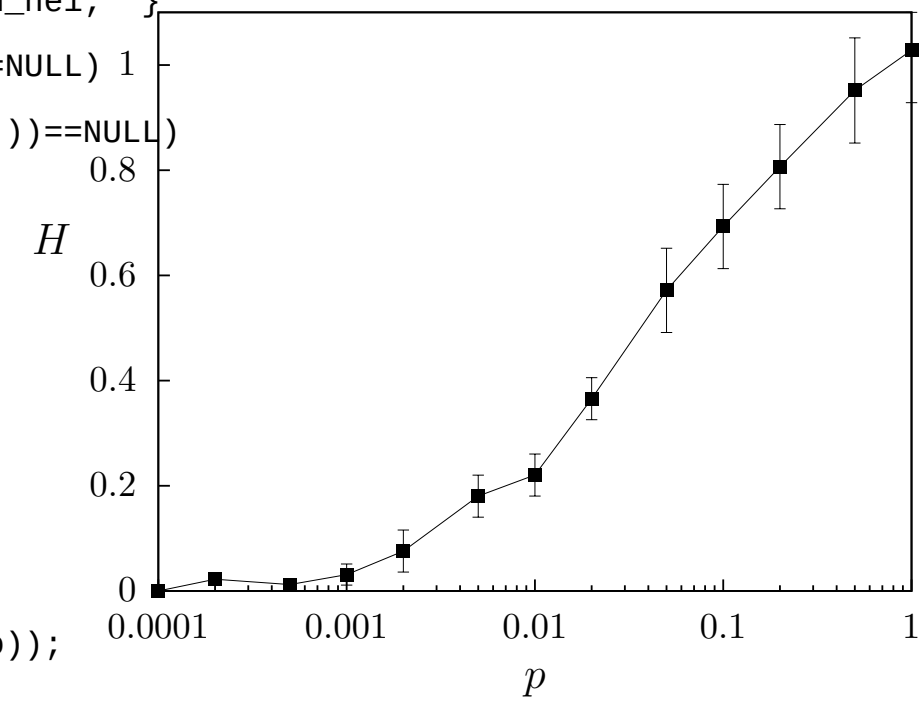
$$H = -\sum_k P_k \log(P_k)$$



# Implementación: Grados

```
void entropy(){
  long n;   double entro,ff,*w; long double s = 0;
  long double ss = 0;
long kMin = nn,kMax = 0, *h, nHisto;
  for(n=0; n<nn; n++){
    if(node[n].n_nei > kMax) kMax = node[n].n_nei;
    if(node[n].n_nei < kMin) kMin = node[n].n_nei;  }
  nHisto = kMax-kMin+1;
  if((h=(long*)calloc(nHisto,sizeof(long)))==NULL)
      error_message(2);
  if((w=(double*)calloc(nHisto,sizeof(double)))==NULL)
      error_message(2);
  for(n=0; n<nHisto; n++) {
    h[n] = 0;   w[n] = 0; }
  for(n=0; n<nn; n++){
    s += node[n].n_nei;
    ss += node[n].n_nei*node[n].n_nei;
    h[node[n].n_nei-kMin]++;  }
  ff = 0;
  for(n=0; n<nHisto; n++){
    ff += ((double)h[n])/nn;
    w[n] = ((double)h[n])/nn; }
  entro = 0;
  for(n=0; n<nHisto; n++){
     if(w[n] > 0){
entro += w[n]*log10(w[n]);  }  }
  printf(" Entropia \t%lf \n\n",(-1.0)*(entro));
}
```

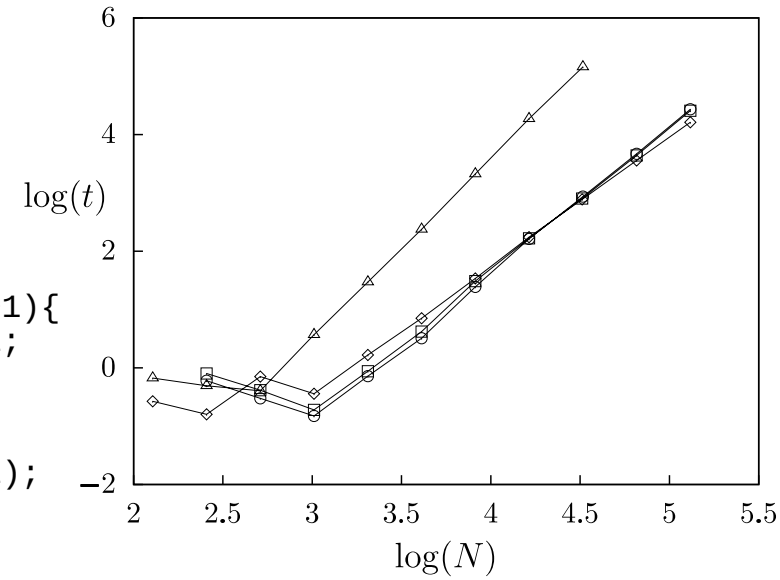
$$H = -\sum_k P_k \log(P_k)$$



# Implementación: Distancia

```
double Avedistance(void)
{
  double acum;   long *length, n, m, k, neiNode;
  char changes;
  if((length=(long*)calloc(nn,sizeof(long)))==NULL)
      error_message(2);
  acum=0;
  for(n=0; n<nn; n++){
    for(m=0; m<nn; m++)
      length[m] = nn+1;
    length[n] = 0;
    do{
      changes = false;
      for(m=0; m<nn; m++){
        if(length[m] <= nn){
          for(k=0; k<node[m].n_nei; k++){
            neiNode = node[m].nb[k];
            if(length[neiNode] > length[m]+1){
              length[neiNode] = length[m]+1;
              changes = true;}  }  } }
    }while(changes);
    for(m=0; m<nn; m++){
      if(length[m] > nn){ warning_message(1);
return nn+1;}
        acum += length[m];}  }
  free(length);
  return (float)acum/(nn*(nn-1));
}
```

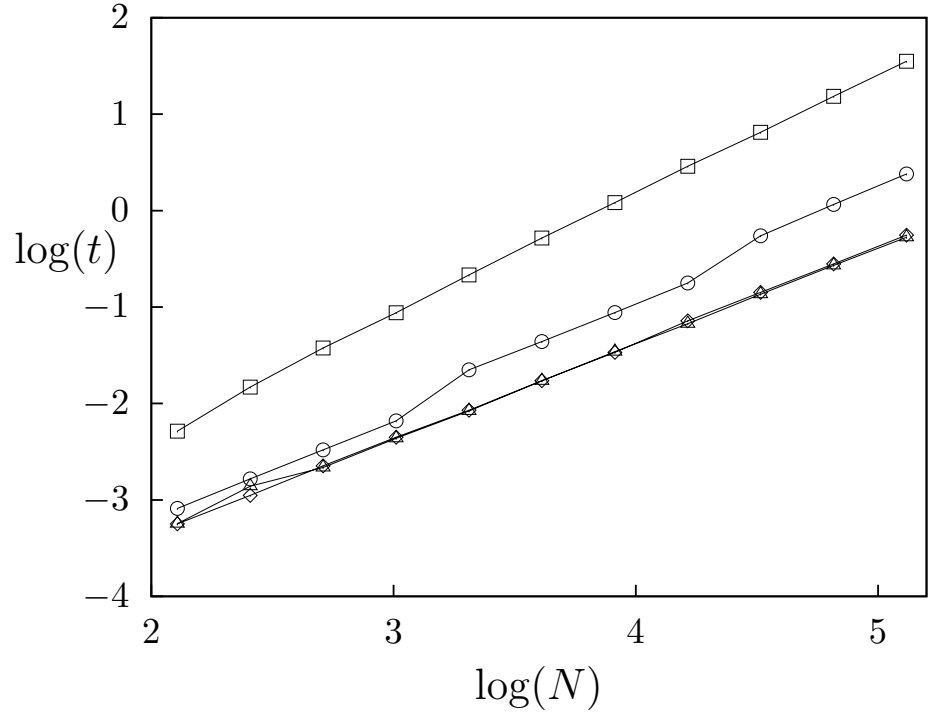
$$l = \frac{1}{N(N-1)} \sum_{i \neq j} d_{ij}$$



# Implementación: Agrupamiento

```
double Newcluster(void)
{
  long    i, j, k, n, m, li, ki;
  float c;
  j=0; ki=0; c=0;
  for(i=0; i<nn; i++){
     li=0;
     ki=node[i].n_nei;
     if(ki>1){
     for(n=0; n<ki; n++){
        j=node[i].nb[n];
        if(j!=i)
          for(m=0; m<ki; m++){
            k=node[i].nb[m];
            if ((k!=j)&&(k!=j)){
              if(isNeighborOf(k,j))
                li++;
              if(isNeighborOf(j,k))
                li++;   }  }  }
    c += li/(ki*2*(ki-1.0));  }   }
    c /= nn;
    return c;
}
```

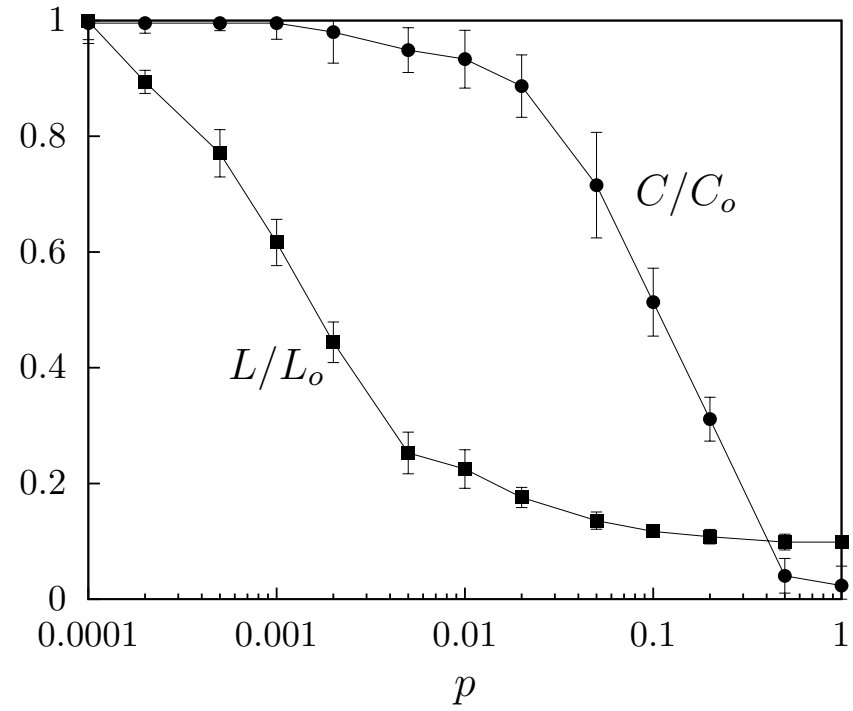
$$C_i = \frac{2 * l_i}{k_i * (k_i - 1)}$$



# Imp: Distancia y Agrupamiento

```
double Avedistance(void)
{
  double acum;   long *length, n, m, k, neiNode;
  char changes;
  if((length=(long*)calloc(nn,sizeof(long)))==NULL)
      error_message(2);
  acum=0;
  for(n=0; n<nn; n++){
    for(m=0; m<nn; m++)
      length[m] = nn+1;
    length[n] = 0;
    do{
      changes = false;
      for(m=0; m<nn; m++){
        if(length[m] <= nn){
          for(k=0; k<node[m].n_nei; k++){
            neiNode = node[m].nb[k];
            if(length[neiNode] > length[m]+1){
              length[neiNode] = length[m]+1;
              changes = true;}  }  } }
    }while(changes);
    for(m=0; m<nn; m++){
      if(length[m] > nn){ warning_message(1);
return nn+1;}
        acum += length[m];}  }
  free(length);
  return (float)acum/(nn*(nn-1));
}
```

$$l = \frac{1}{N(N-1)} \sum_{i \neq j} d_{ij}$$

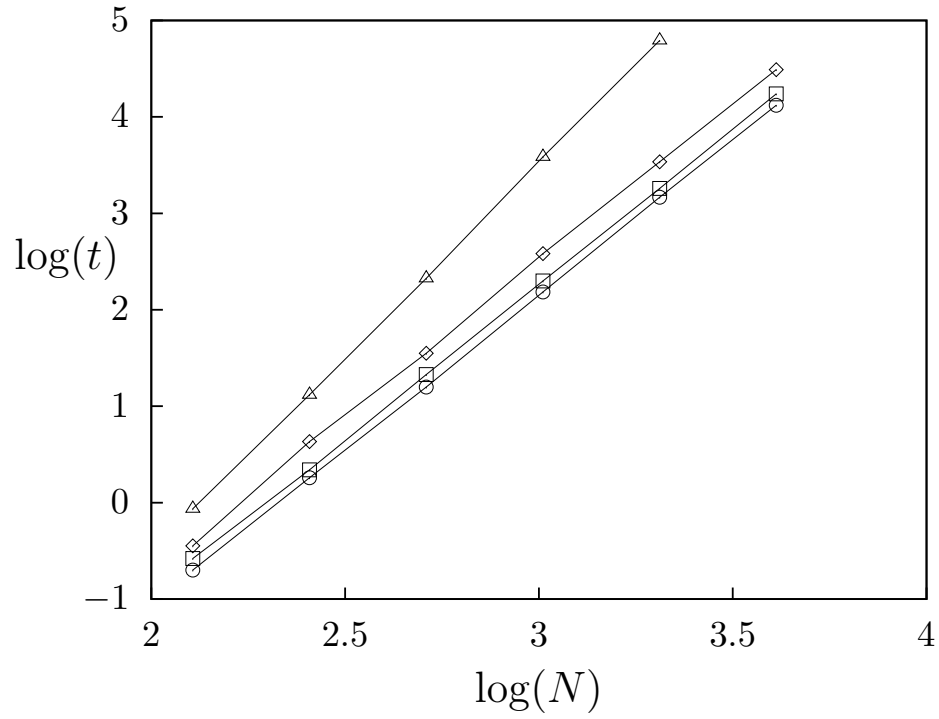


# Implementación: Vulnerabilidad

```
void vulneravility(double* v,double* q)
{
   double  eg;
   long   i;
   eg = globalefficiency();
   for(i=0; i<nn; i++){
      v[i]=((eg-q[i])/eg);
   }
   for (i=0; i<nn; i++){
       if (q[i]<0)
    v[i]=0;
   }
}

double MaxVulneravility(double* v)
{
  int i;
  double vMax = v[0];
  for (i=1; i<nn; i++)
    if(v[i]>vMax)
      vMax=v[i];
  return vMax;
}
```

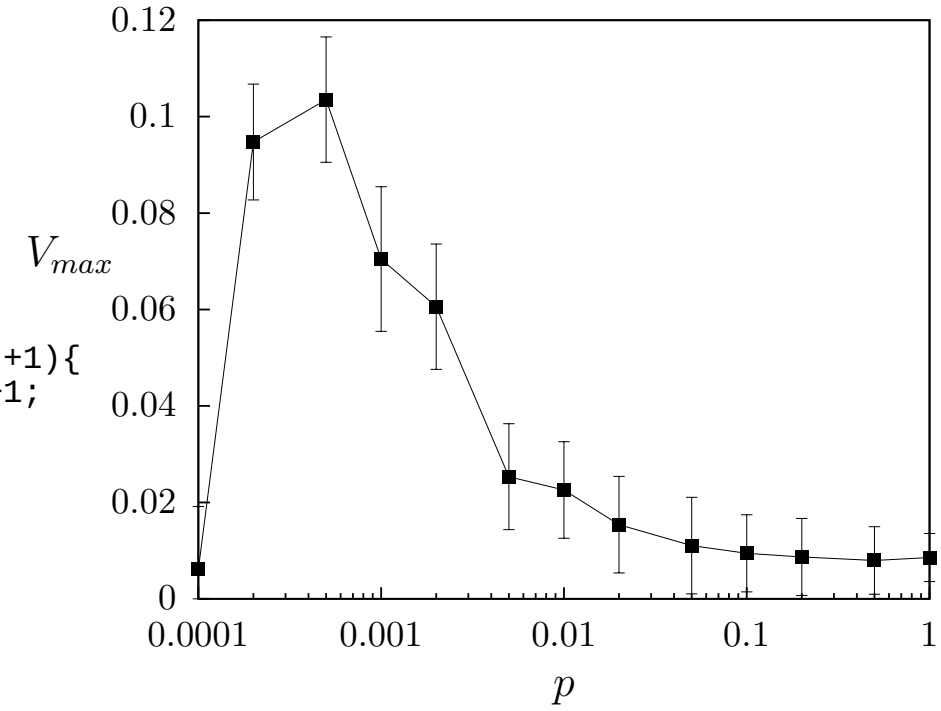
$$\langle V \rangle = max_i V_i$$



# Implementación: Vulnerabilidad

```
double globalefficiency()
{
 double acum,r;     long *length, n, m, k, neiNode;
 char changes;
  if((length = (long*)calloc(nn,sizeof(long))) == NULL)
    error_message(2);
  acum=0;
  areNeighbors(0,1);
  for(n=0; n<nn; n++){
    for(m=0; m<nn; m++)
      length[m] = nn+1;
    length[n] = 0;
    do{ changes = false;
        for(m=0; m<nn; m++){
          if(length[m] <= nn){
            for(k=0; k<node[m].n_nei; k++){
              neiNode = node[m].nb[k];
              if(length[neiNode] > length[m]+1){
                length[neiNode] = length[m]+1;
                changes = true;}   }   }   }
    }while(changes);
    for(m=0; m<nn; m++){
      if(length[m] > nn){warning_message(1);
    return nn+1;}
if ((n!=m)&&(length[m]!=0))
      acum += ((1.0)/length[m]); }   }
  free(length);
  r= acum/(nn*(nn-1));
  return (r);
}
```

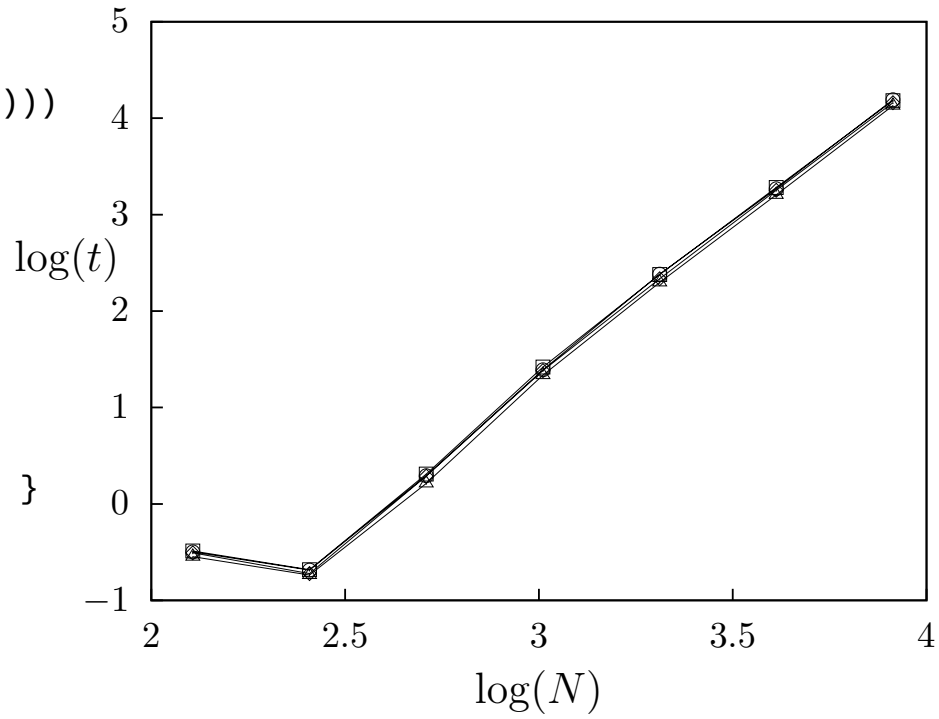
$$\langle V \rangle = max_i V_i$$



# Implementación: Autovalores

$$\rho(\lambda) = \frac{1}{N}\sum_i \delta(\lambda - \lambda_i)$$

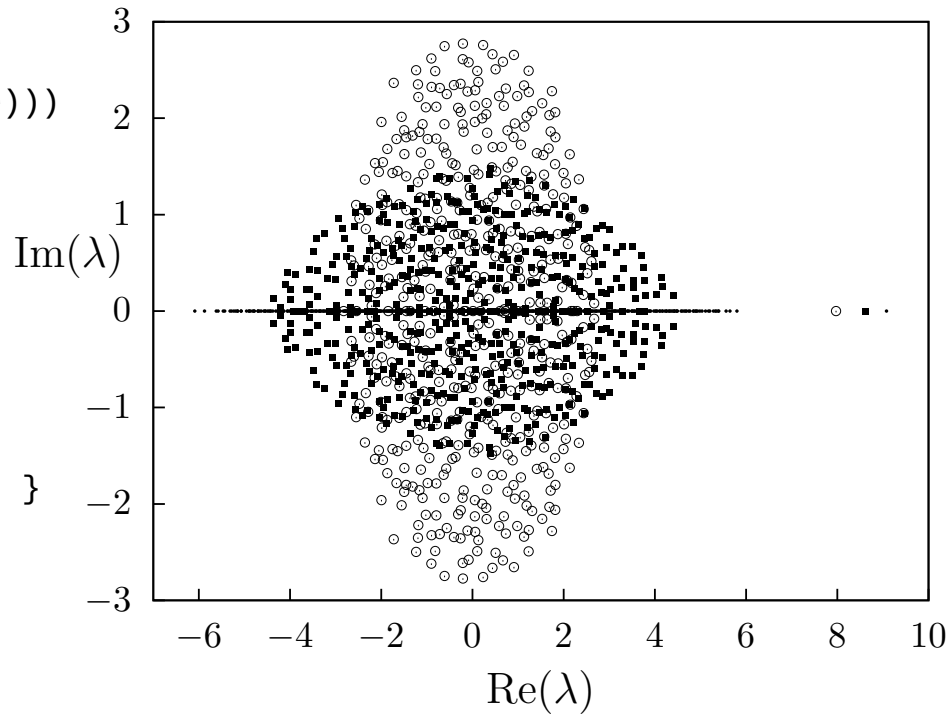
```
autovalores(){
  double A[nn][nn], lr[nn], li[nn], DUMMY[1][nn], WORK[nn];
  double* AT;
/* for transformed matrix */
  int i, j, ok, c1, c2, c3;
  char c4;
  if((AT=(double*)calloc(nn*nn,sizeof(double)))
    ==NULL)
    error_message(2);
  for (i=0; i<nn; i++){
    for(j=0; j<nn; j++){
      AT[(j+nn*i)]=isNeighborOf(i,j); }
}
  c1=nn;    c2=3*nn;    c3=1; c4='N';
  dgeev_(&c4,&c4,&c1,AT,&c1,lr,li,DUMMY,&c3,
         DUMMY,&c3,WORK,&c2,&ok);
  if (ok==0){
    for (i=0; i<nn; i++){
      printf("%lf\t%lf\n", lr[i],li[i]);  }  }
  else printf("An error occured");
  free(AT);
}
```



# Implementación: Autovalores

$$\rho(\lambda) = \frac{1}{N} \sum_i \delta(\lambda - \lambda_i)$$

```
autovalores(){
  double A[nn][nn], lr[nn], li[nn], DUMMY[1][nn], WORK[nn];
  double* AT;
/* for transformed matrix */
  int i, j, ok, c1, c2, c3;
  char c4;
  if((AT=(double*)calloc(nn*nn,sizeof(double)))
    ==NULL)
    error_message(2);
  for (i=0; i<nn; i++){
    for(j=0; j<nn; j++){
      AT[(j+nn*i)]=isNeighborOf(i,j); }
}
  c1=nn;    c2=3*nn;    c3=1; c4='N';
  dgeev_(&c4,&c4,&c1,AT,&c1,lr,li,DUMMY,&c3,
         DUMMY,&c3,WORK,&c2,&ok);
  if (ok==0){
    for (i=0; i<nn; i++){
      printf("%lf\t%lf\n", lr[i],li[i]);  }  }
  else printf("An error occured");
  free(AT);
}
```



# Conclusiones y Recomendaciones

Se seleccionaron las medidas: Grado de la red, la distribución de probabilidades de los grados entrantes y salientes, la entropía de la distribución de grado, la logitud característica, la Eficiencia, la eficiencia sin el nodo $i$, la vulnerabilidad sin el nodo $i$, la Vulnerabilidad máxima, el coeficiente de agrupamiento para grafos simples y dirigidos y el espectro de autovalores de la red.

Luego se calcularon para las diferentes medidas ya implementadas en ISyS los tiempos de ejecución para diferentes tamaños de red estableciendo generalemente una relación funcional de una ley de potencia $t \sim N^{\gamma}$ entre el tamaño de la red y el tiempo requerido por la implementación de la medida.

# Bibliografía

## Referencias


[1] Tucci, K.*Procesos dinámicos espaciotemporales en redes inhomogéneas.* Tesis Doctoral, Doctorado en Física Fundamental, Universidad de Los Andes. Mérida. Venezuela. (2002)

[2] P. Erdös, A. Rényi, *On random graphs. I.*, Publicationes Mathematicae **6**, 290-297 (1959).

[3] L. da F. Costa, F.A. Rodriguez, G. Travieso, P.R.Villas Boas, *Characterization of Complex Networks: A Survey of measurements*,
Instituto de Física de São Carlos, Universidade de São Paulo, Física, SP. Brazil, 16 August (2006).

[4] R. Johnsonbaugh, *Matemáticas Discretas*, Prentice Hall, México (1997).

[5] K.H.Rosen. *Matemática Discreta y sus aplicaciones.* (5ta Edicion) McGraw-Hill Interamericana Editores, 2004.

# Bibliografía

## Referencias


[1] V. Latora, M. Marchiori *Efficient behavior of small-world network.* Physics Review Letters, 87:198701,(2001).

[2] V. Gol'dshtein, G. A. Koganov, G.I. Surdutovich. *Vulneravility and hierarchy of complex networks.* cond-mat/0409298, (2004)

[3] V. Latora M. Marchiori *Vulnerability and protection of critical infrastructure.* Physics Review E, 71:015103R,(2005).

[4] A.-L. Barabási, R. Albert. (1997). *Emegence of scaling in random networks.* Science, 286:509-5123.

[5] R. Albert, A. L. Barabási, *Statistical mechanics of complex networks*, Rev. Mod. Phys. **74**, 47-97 (2002).

# Bibliografía

## Referencias


[1] M. A. Porter, J.P. Onnela, P. J. Mucha *Communities in Networks*, Not. Amer. Math. Soc., **56**, 1082-1097 (2009)

[2] E. Lieberman, C. Hauert, M.A. Nowak, *Evolutionary dynamics on graphs. Nature*, **433**, 312-316 (2005).

[3] M. E. J. Newman *The structure and function of complex networks*, SIAM Review **45**, 167-256 (2003).

[4] S.N. Dorogovtsev, J.F.F. Mendes, *Evolution of networks*, Advances in Physics **51**, 1079-1187 (2002).

[5] *UNAM, 2006*
http://www.fis.unam.mx/ max/English/notasredes.pdf