UNIVERSIDAD NACIONAL DE HURLINGHAM

Estructuras de Datos

Profesor

Sergio Gonzalez

UNIVERSIDAD NACIONAL DE HURLINGHAM

Unidad 8: Mix

Profesor

Sergio Gonzalez

Mix de temas

- Diccionarios
- Listas nativas
- map, filter y reduce
- Grafos

TDA Diccionario

- Colección de pares clave – valor
- Tabla o arreglo asociativo
- Hashing (Busqueda rapida)
- Se usa una clave para acceder a un valor
- Claves unicas
- Diccionario tradicional:
  - Clave: Palabra
  - Valor: Definicion

Diccionarios en Python

- Estructura dinamica mutable no ordenada
- La clave puede ser un entero, un float, un string, una tupla o cualquier inmutable y el valor pueden ser cualquier cosa (una lista por ejemplo)

productos = {} o productos = dict()

```
productos={"manzanas":39, "peras":32, "lechuga":17}
```

```
dic = dict(zip('abcd',[1,2,3,4]))

dic →   {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
```

```
productos[nombre]=precio
```

Diccionarios en Python

```
paises={"argentina":40000000, "españa":46000000, "brasil":190000000, "uruguay": 3400000}
```

```
def imprimir(paises):
    for clave in paises:
        print(clave, paises[clave])
```

```
>>> caballeros = {'gallahad': 'el puro', 'robin': 'el valiente'}
>>> for k, v in caballeros.items():
...     print(k, v)
...
gallahad el puro
robin el valiente
```

```
dic =  {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
keys= dic.keys()

keys→ [‘a’,’b’,’c’,’d’]
```

```
values= dic.values()


values→ [1,2,3,4]
```

Diccionarios en Python

```
dic = {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
valor = dic.get('b')

valor → 2
```

```
valor = dic.pop('b')

valor → 2
dic → {'a' : 1, 'c' : 3 , 'd' : 4}
```

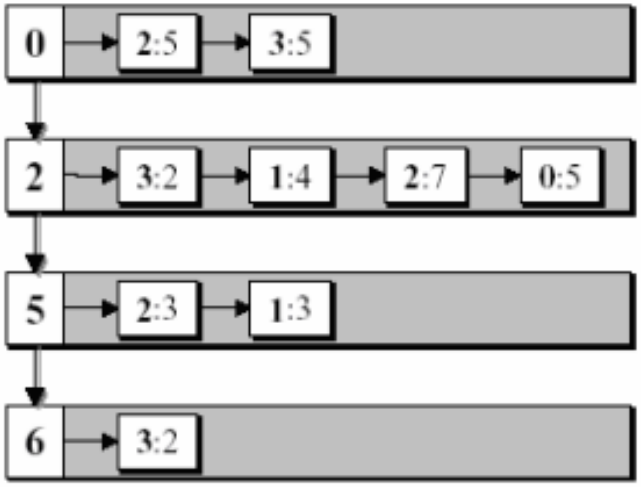
```
dic1 = dic.copy()

dic1 → {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
```

Una aplicación: Matrices esparcidas

- Muchos ceros
- Guardamos solo posiciones y valores

$$\begin{pmatrix} 0 & 0 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 4 & 7 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \end{pmatrix}$$



Listas nativas en Python

- Implementacion en Python de listas enlazadas
- Estructura de datos mas usada
- Iterable
- Mutable
- Dinamica

Creacion:

lista = [] o lista = list()

lista = [1,2,3,4] o lista = list((1,2,3,4))

Insertar valores:

lista.append(valor)

lista.insert(posicion,valor)

lista[indice] = valor

Borrar:

Lista.remove(valor)

lista.pop(posicion)

Direccionamiento:

Posicion inicial = 0

lista[posicion]

lista[inicio:fin+1]

Operaciones:

len(lista)

lista2 = lista.copy()

Listas nativas en Python

Iterar lista

```
for valor in lista:
    print(valor)

for index in range(len(lista)):
    print(lista[index])
```

Buscar en lista

```
if valor in lista:
    print(valor)
    print(lista.index(valor))
```

```
lista + lista2
min(lista)
max(lista)
```

Map, filter y reduce

- Paradigma de programacion funcional
- Funciones que reciben a otras funciones por parametros
- Con map, filter y reduce, podemos aplicar funciones a los elementos de una lista sin iteraciones

Map

Toma una funcion y un iterable y retorna un iterable con la funcio aplicada a cada elemento

```
#Ejemplo del operador Map
def add_five(x):
    return x + 5

nums = [11, 25, 34, 100, 23]
result = list(map(add_five, nums))
print(result)

#Resultado:
[16, 30, 39, 105, 28]
```

Filter

Filtra los elementos de una lista cuando la funcion devuelve True para el elemento

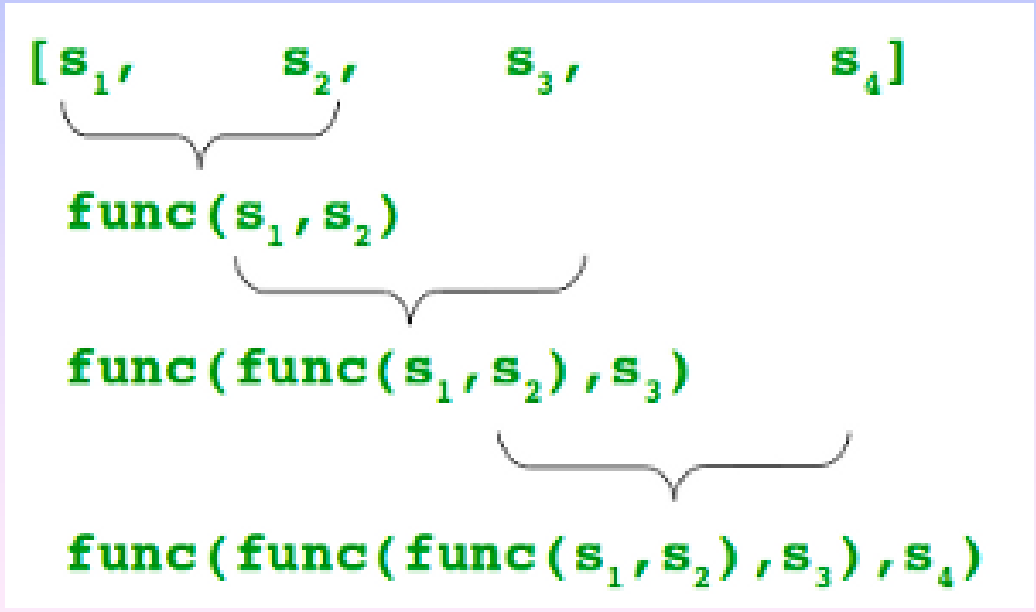
```
>>> def esPar(x):
...     return x%2 == 0
...
>>> l = [1,2,3,4,5]
>>> lfilt = list(filter(esPar,l))
>>> lfilt
[2, 4]
>>>
```

Reduce

Reduce los elementos de una lista a un unico elemento aplicando la funcion reductora

```
>>> def suma(x,y):
...     return x+y
...
>>> l = [1,2,3,4,5]
>>> reduce(suma, l)
15
```

```
>>> def esMayor(x,y):
...     res = x
...     if y > x:
...             res = y
...     return res
...
>>> l = [10,25,13,14,5]
>>> reduce(esMayor, l)
25
```



Operador Lambda

Crear funciones sin nombre, que se pueden desechar luego de usarlas, se suelen usar en conjunto con map, filter y reduce

```
#Función lambda que devuelve la suma de sus dos argumentos:
f = lambda x, y : x + y
f(2 + 2)

#Resultado:
4
```

Operador Lambda

```
#Ejemplo del operador Map y Lambda
nums = [11, 25, 34, 100, 23]
result = list(map(lambda x:x+5, nums))
print(result)

#Resultado:
[16, 30, 39, 105, 28]
```

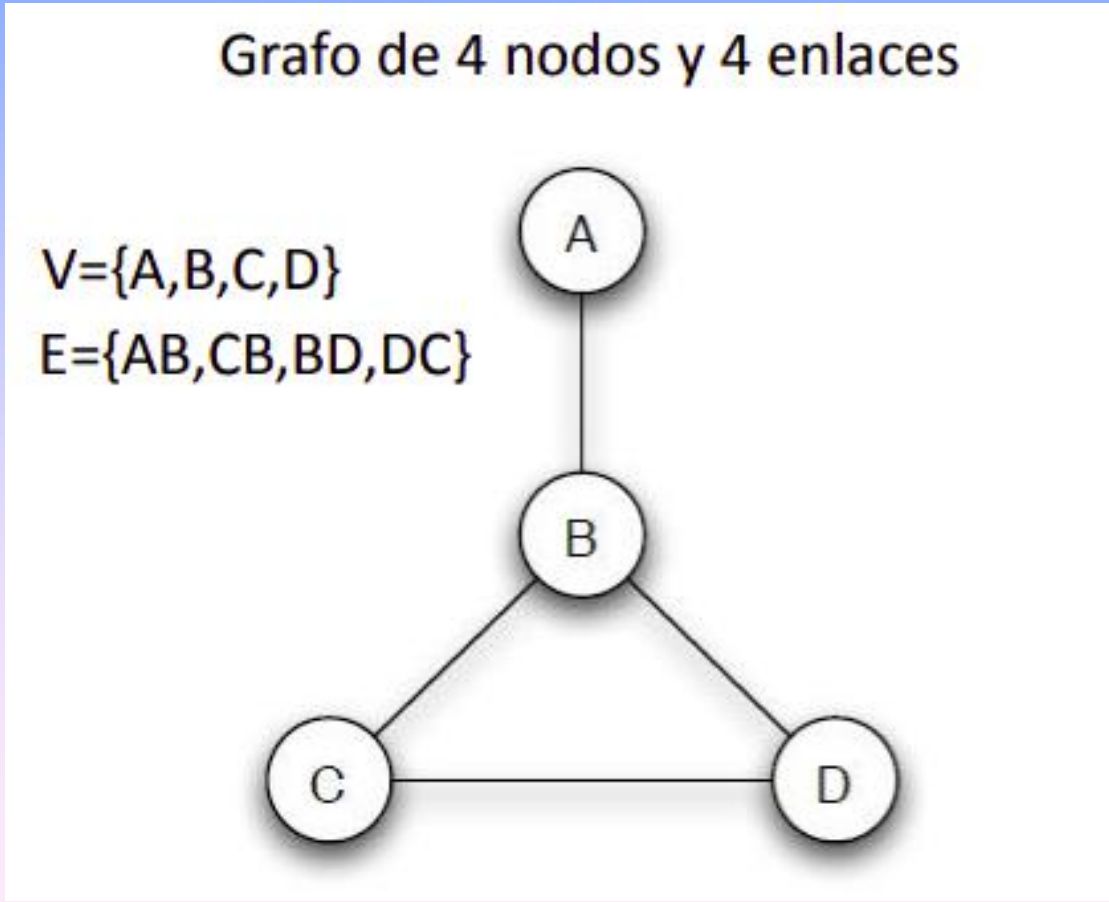
```
#Usando el operador Filter
nums = [0, 2, 5, 8, 10, 23, 31, 35, 36, 47, 50, 77, 93]
result = filter(lambda x: x % 2 == 0, nums)
print(result)

#Resultado:
[2, 8, 10, 36, 50]
```

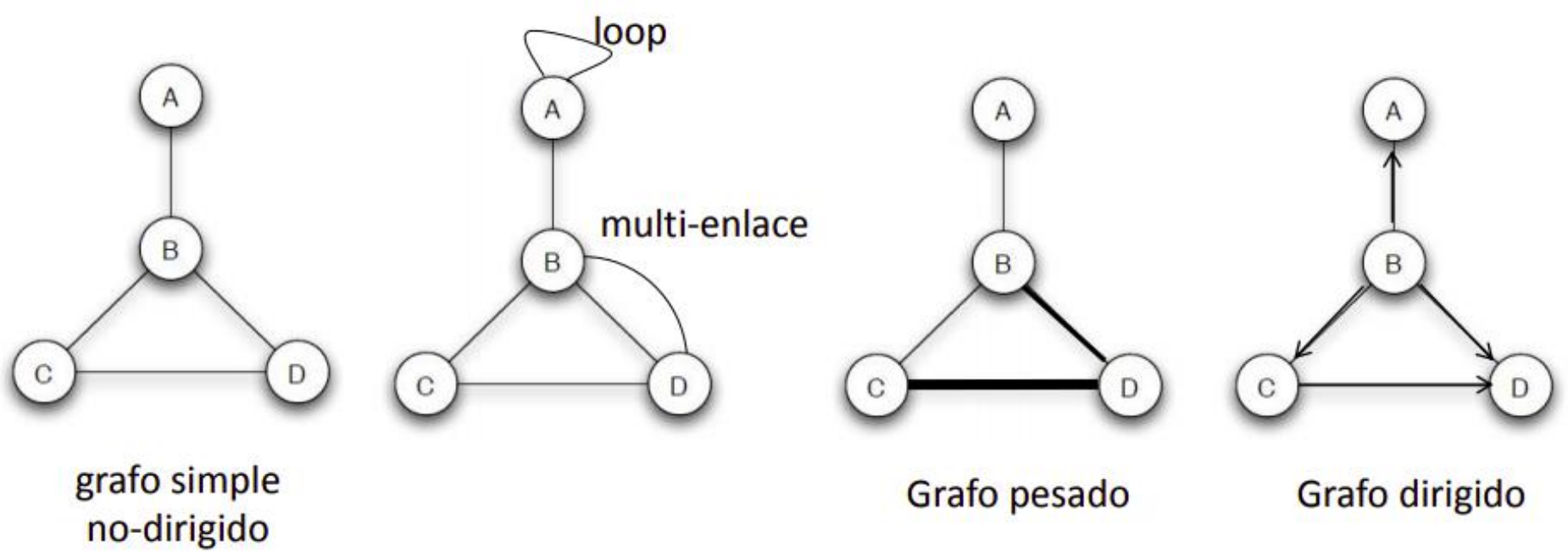
Grafos

- Estructura dinamica/estatica no lineal
- Conjunto de nodos (vertices) conectados por un conjunto de enlaces (aristas)
- Matematicamente:
  - Vertices = $V = \{v_i\}$
  - Aristas = $E = \{a_i\}$ = Pares de vertices

Grafos

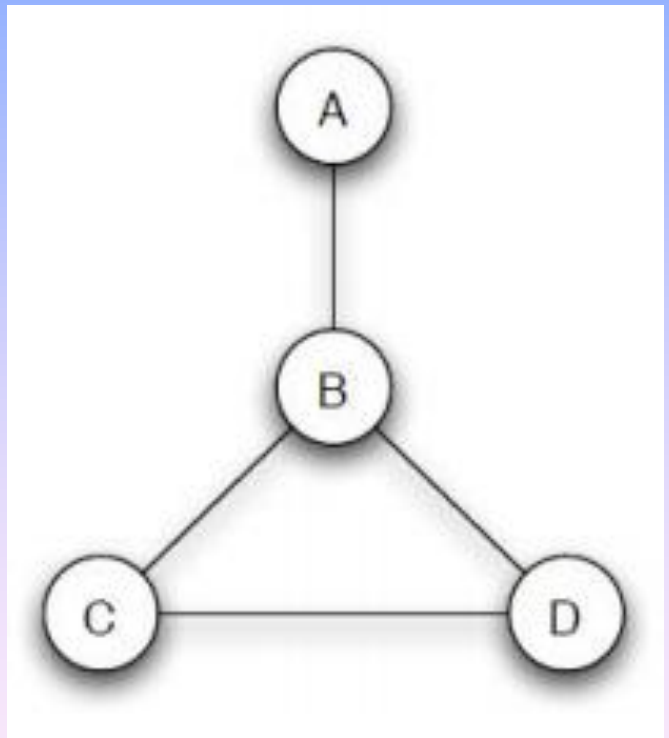


Algunos tipos de grafos



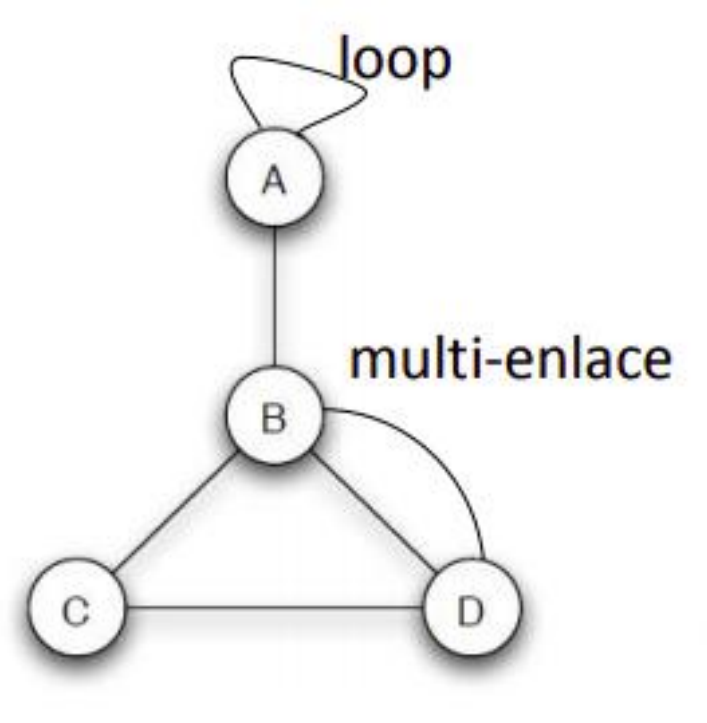
Grafos simples no dirigidos

- *G(V,E)*
- Aristas pares no ordenados de vertices distintos
- No importa el orden
- Vertices adyacentes (vecinos)



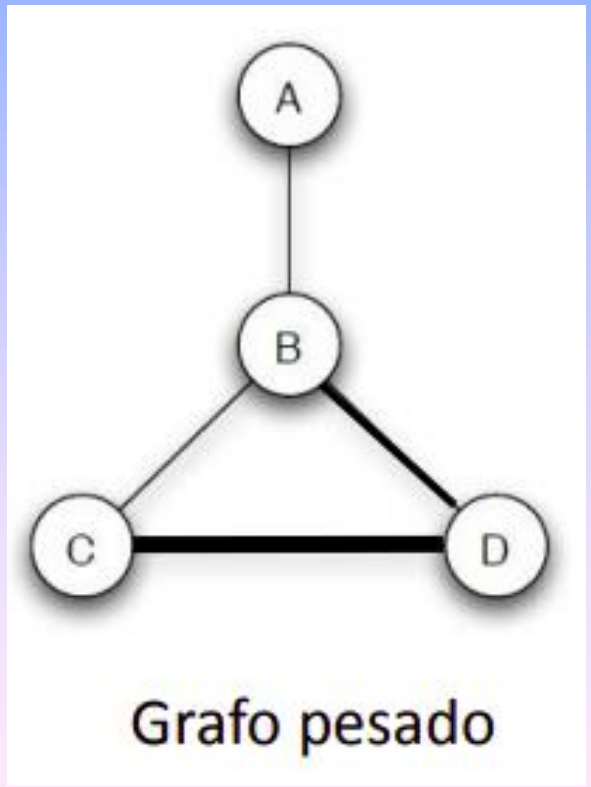
Grafos multiples no dirigidos

- *G(V,E)*
- Aristas pares no ordenados de vertices que pueden ser no distintos
- Permite loops y enlaces multiples



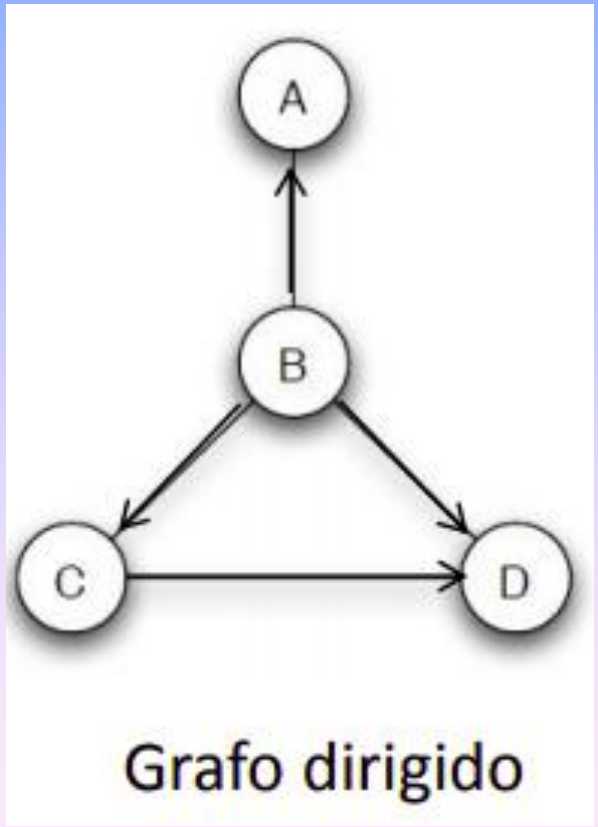
Grafos pesados

- *G(V,E)*
- Aristas pares no ordenados de vertices
- Funcion de peso (*w*)



Grafos dirigidos

- *G(V,E)*
- Aristas pares ordenados de vertices distintos
- Cabeza y cola de arista
- Vertices predecesor y sucesor

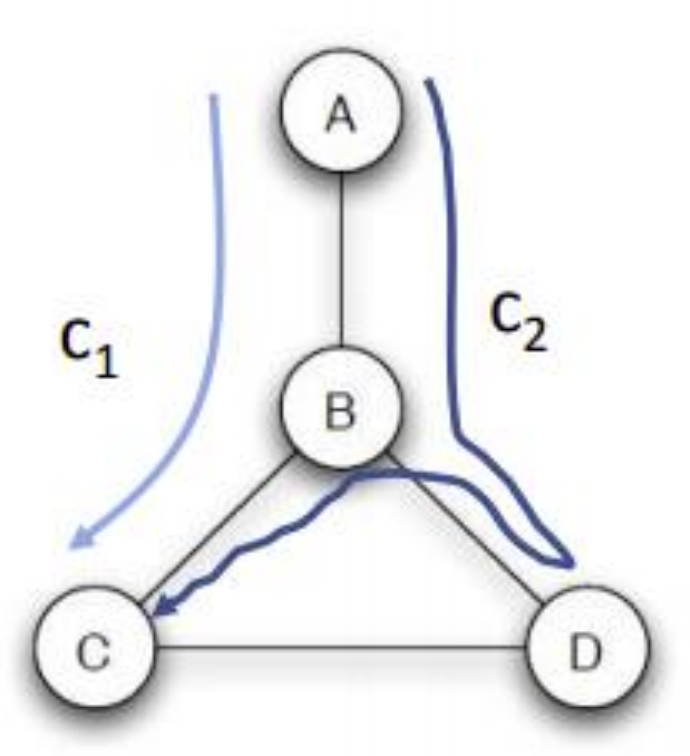


Grafo dirigido

Caminos y ciclos

- Camino: Secuencia de vertices, tales que vertices consecutivos son adyacentes
- Camino simple: Camino que no repite vertices
- Longitud de camino: Cantidad de aristas (distancia topologica entre vertices)
- Ciclo: Camino que no repite vertices, solo el primero y el ultimo

Caminos y ciclos



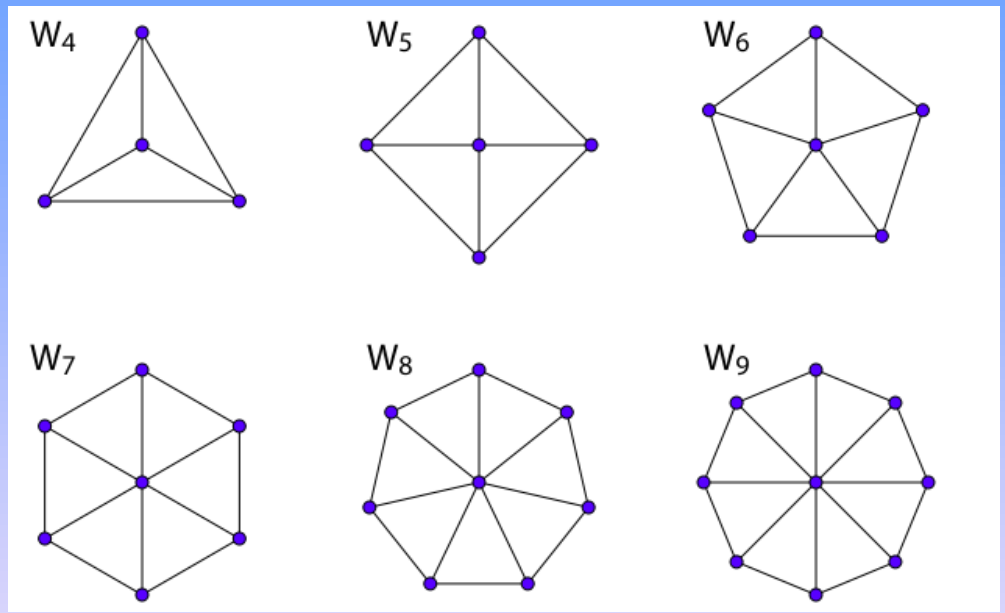
c1={A,B,C}

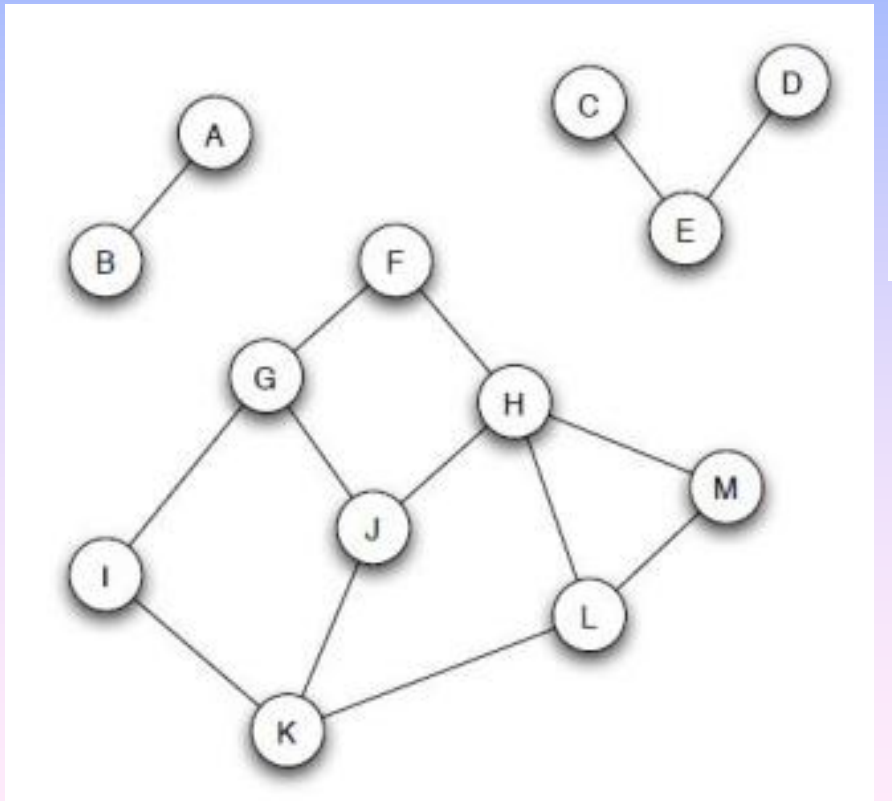
c2={A,B,D,B,C}

c3={B,C,D,B}

Conectividad

- No siempre es posible encontrar un camino entre cualquier par de vertices
- Grafo conexo: Existe un camino entre cualquier par de vertices
- Componente: Subgrafo conexo
- Componente gigante: Componente mas grande
- Grafo completo: Todos vertices vecinos
- Grado de vertice: Numero de vecinos


W4
W5
W6
W7
W8
W9


A
B
C
D
E
F
G
H
I
J
K
L
M

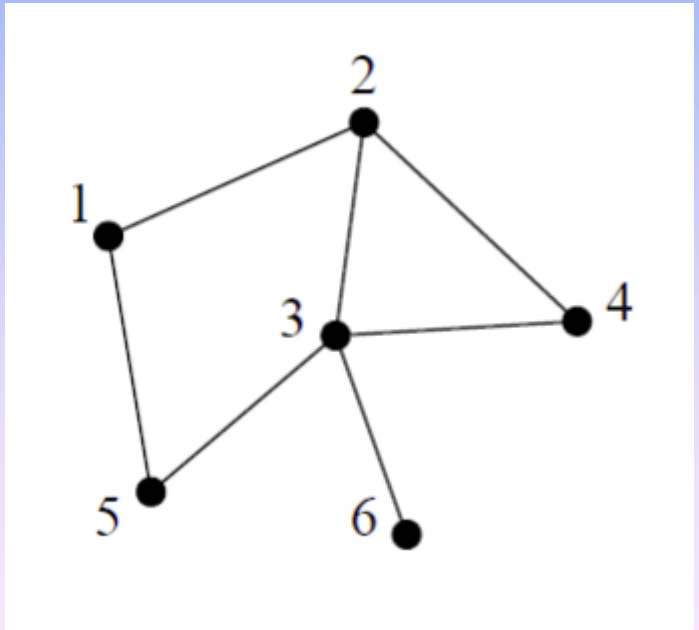
Representacion de grafos simples

- Lista de aristas
- Listas enlazadas
- Matriz de adyacencia
- Diccionarios
- …

Representacion de grafos simples

Lista de enlaces

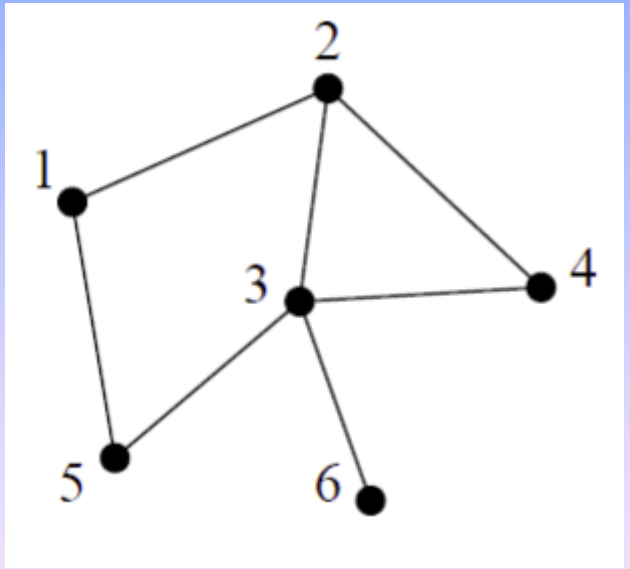
(1,2)
(1,5)
(2,3)
(2,4)
(3,4)
(3,5)
(3,6)



Representacion de grafos simples

Matriz de adyacencia

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



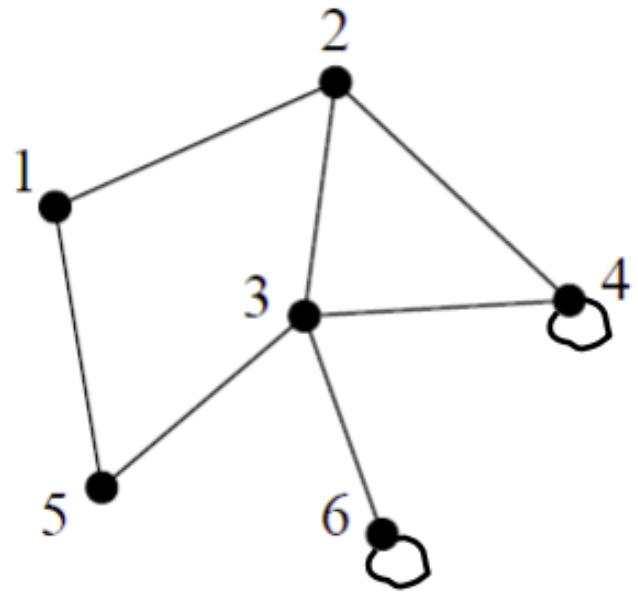
A_{ij}= 1 si existe enlace entre nodos *i* y *j*
0 si no

A es una matriz **simétrica.**

Representacion de grafos con loops

Matriz de adyacencia

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 2 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 \end{pmatrix}$$



A_{ij}= 1 si existe enlace entre nodos *i* y *j*
0 si no

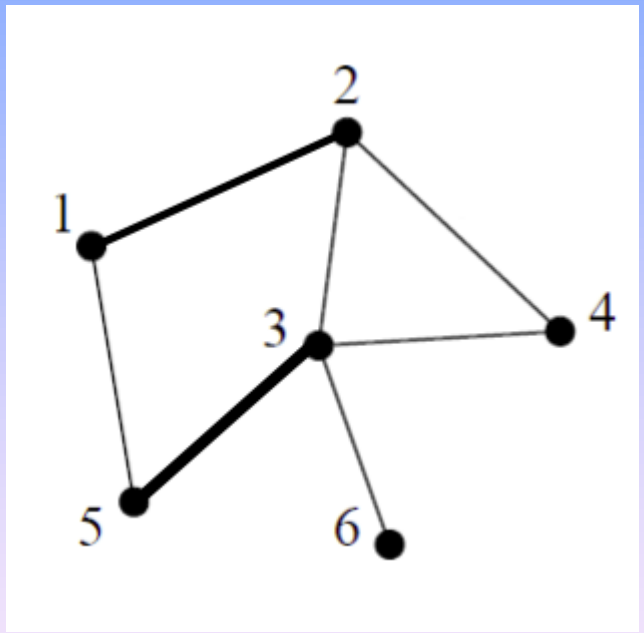
Los ***loops*** se corresponden con
$A_{ii} = 2$

A es una matriz **simétrica.**

Representacion de grafos pesados

Matriz de adyacencia

$$\mathbf{A} = \begin{pmatrix} 0 & 2 & 0 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 3 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$



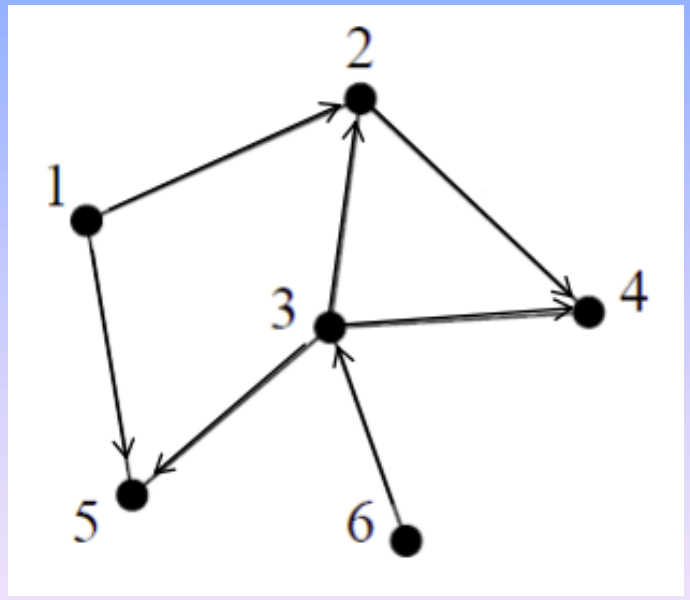
A_{ij}= w_{ij} si existe enlace entre nodos *i* y *j*
0 si no

A es una matriz **simétrica.**

Representacion de grafos dirigidos

Matriz de adyacencia

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

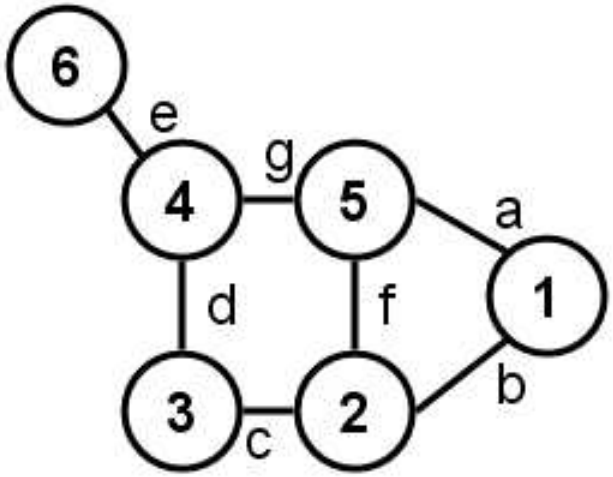


A_{ij}= 1 si existe enlace que **llega a** *i* **desde** *j*
0 si no

A no es una matriz **simétrica.**

- Matrices de adyacencia poseen muchos ceros
  - Muchas propiedades del grafo se pueden calcular usando la matriz
- Matrices de incidencia
  - Aristas como columnas y vertices como filas

a	b	c	d	e	f	g	
1	1	0	0	0	0	0	1
0	1	1	0	0	1	0	2
0	0	1	1	0	0	0	3
0	0	0	1	1	0	1	4
1	0	0	0	0	1	1	5
0	0	0	0	1	0	0	6



- Campo de estudio matematico
  - Recorridos
  - Propiedades emergentes
    - Surgen de la complejidad del sistema
  - Clustering
  - Propiedades topologicas

- Modelos de cualquier tipo de red
  - Sistemas complejos
  - Analisis usando propiedades del grafo
  - Ciencias sociales
  - Biologia
  - Transporte
  - Aprendizaje automatico, Data mining
  - Roles topologicos: Puentes, Hubs
  - ….

Ejemplo: Mundo pequeño

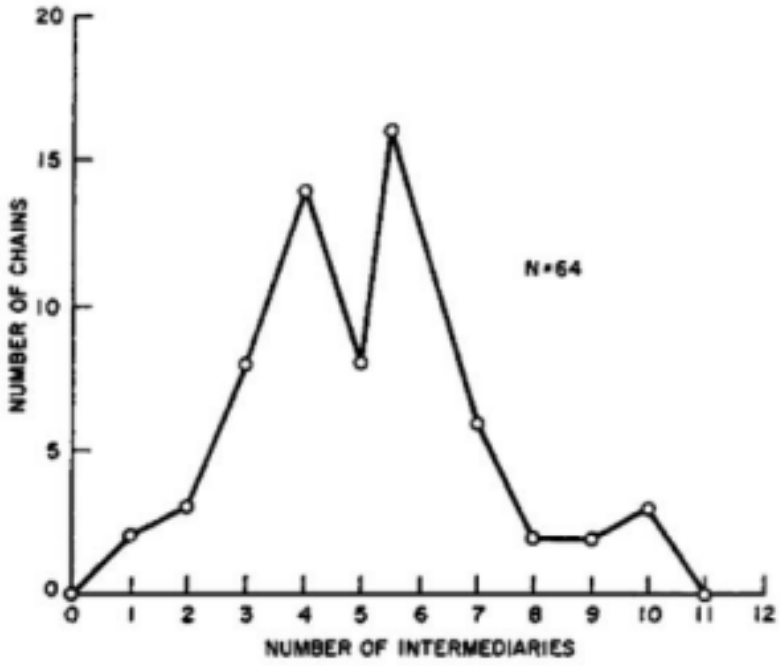
- Problema en redes de personas: A conoce a B
- Existen caminos inesperadamente cortos entre dos personas cualquiera

Hipótesis: estamos conectados **globalmente** por medio de **cadenas cortas** de conocidos

Experimento:

I. Selección aleatoria de 296 personas de la costa oeste.
II. Tarea: hacer llegar una carta a una dada persona (corredor de bolsa de direcci{on conocida en Boston)
III. Sólo podían entregar la carta a alguien que conocieran, y encomendarle II y III

- ✓ 64/296 cadenas llegaron a destino
- ✓ Mediana de intermediarios: 6

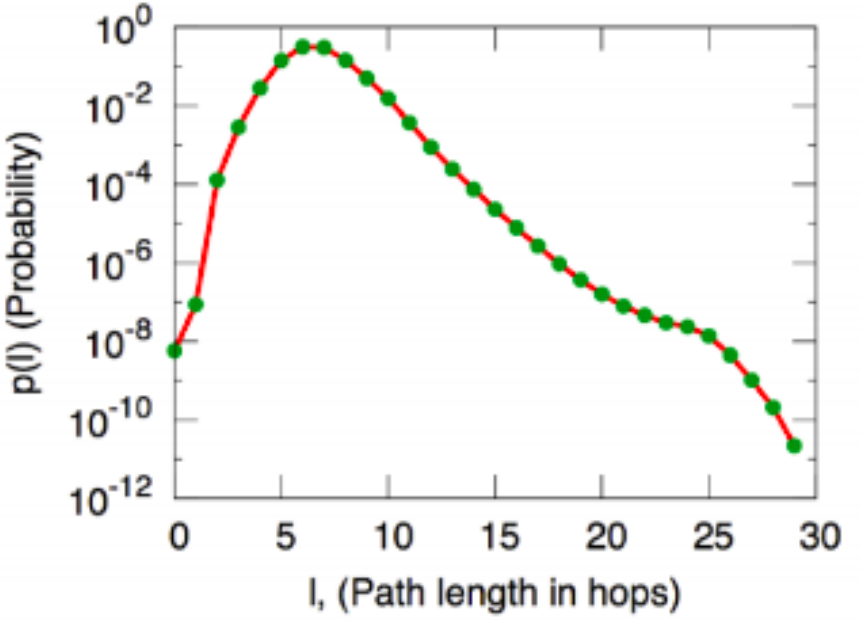


Notas:

- Conclusión sobre prop de la red usando estimación via *trazadores*
- 6 es mucho o poco?

Stanley Milgram (1967) "The small world problem" Psychology Today

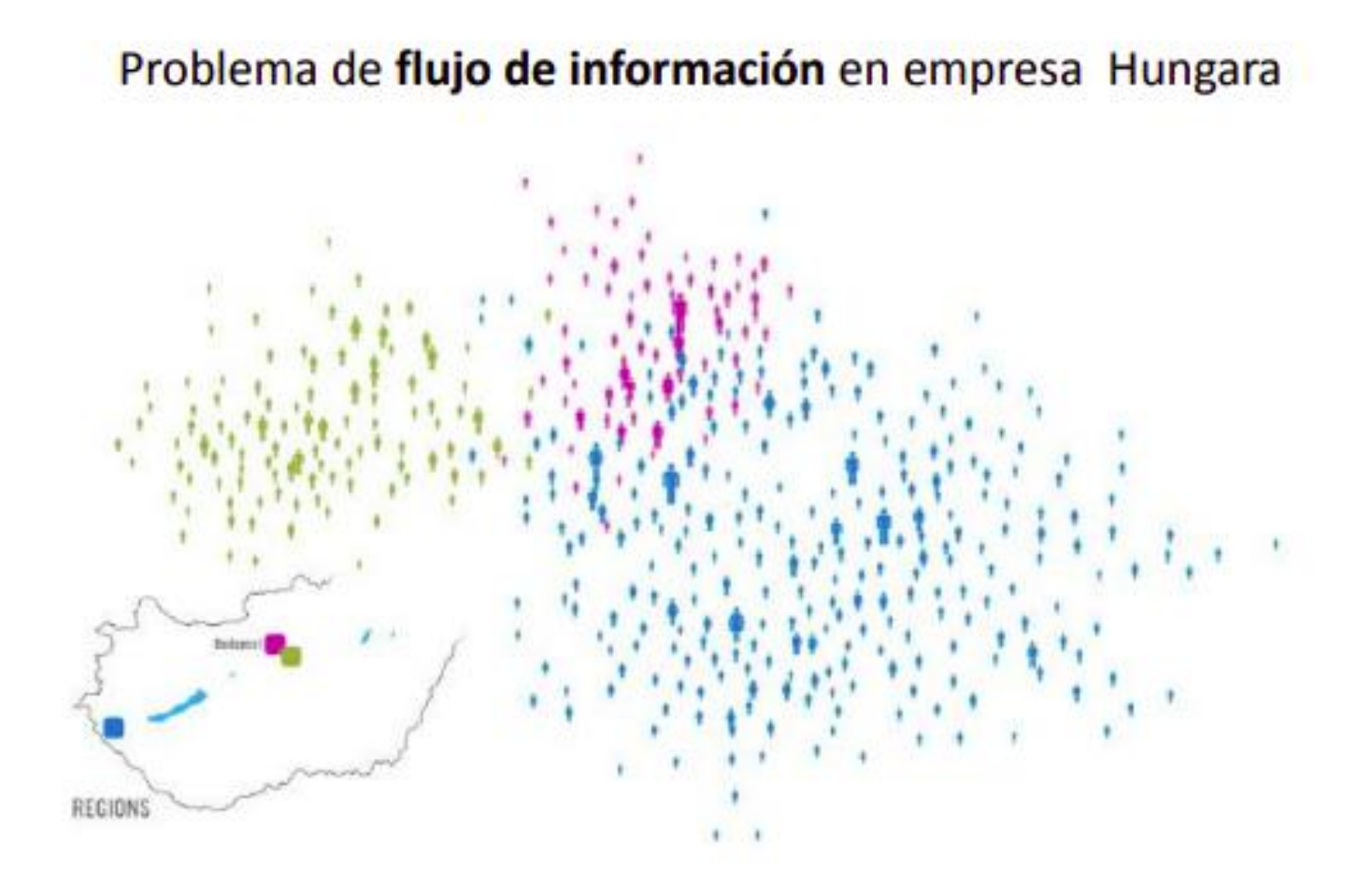
- Analisis de 240.000.000 de cuentas de Microsoft Instant Messenger activas, durante un mes.
- Enlaces entre usuarios que intercambian mensajes



- ✓ Sampling: BFS para 1000 usuarios
- ✓ Estimación un poco más global, aunque aún sesgada (personas tecnoalfabetizadas)
- ✓ Existencia de componente gigante que engloba a la mayor parte de la red
- ✓ <d>~6.6 (!)

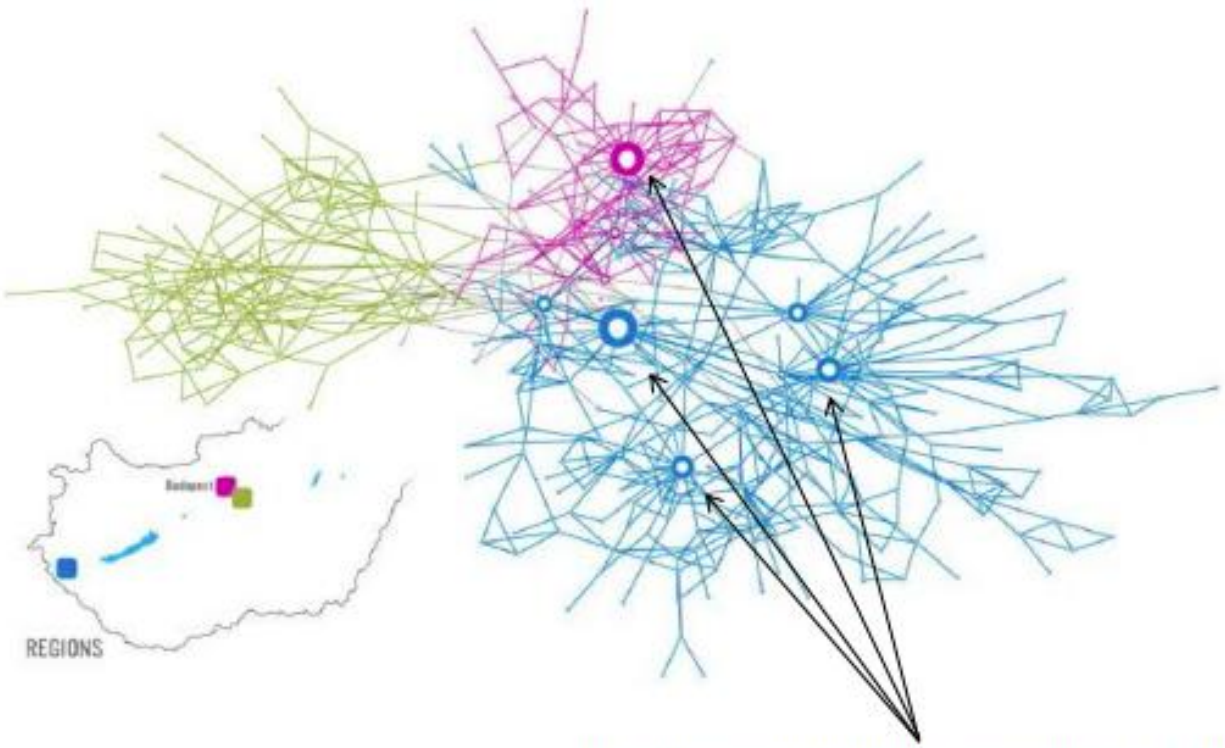
Leskovec 2007 "Planetary-scale views of instant-messagging network"

Ejemplo: Management



Ejemplo: Management

Mapeo de resultados a **lenguaje de grafos**: *¿a quién consultas para tomar decisiones?*



Identificación de individuos de alta influencia

Ejemplo: Management

Red de influencia del empleado más conectado (1eros y 2dos vecinos)

- Empleado de Seguridad y problematicas ambientales
- Visita regularmente diferentes instalaciones
- Conectado con mucha gente, pero no con management (!)
- *Gossip center*