7

1º iMAT - Algorítmica

Tema 7: Estrategias y algoritmos avanzados.

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

Algoritmo voraz

Algoritmo voraz (I)

- La solución se va construyendo con ‘las mejores opciones’ en cada momento.
- En cada iteración se escoge el elemento que ofrece la mejor solución inmediata
- La solución local ‘óptima’ en cada momento conduce a una solución óptima global.

Algoritmo voraz (I)

- Producir un resultado “óptimo” a partir de un conjunto de opciones candidatas
  - Optimizar cierta medida o función objetivo
  - Cumpliendo ciertas restricciones
- Hipótesis de partida:
  - Al menos existe una solución óptima, si hay varias equivalentes
- Características de este enfoque:
  - La solución se va formando poco a poco (incrementalmente)
    - Se elige la mejor forma de incrementar la solución parcial eligiendo la mejor opción candidata
    - Una vez incrementada (mejorada) la solución parcial, no se puede modificar eliminando elementos de la misma

Algoritmo voraz (II)

- Esquema general

```
sol = Voraz (candidatos)

sol = { };

1 while (candidatos ≠ { } AND NoEsSolucion(sol) )

    mejor = Seleccionar (candidatos);
    candidatos = candidatos – {mejor};

    2 if EsValida (sol ∪ {mejor} )
            sol = sol ∪ {mejor} ;
    end 2

end 1
Fin
```

Algoritmo voraz: Problema del cambio

- Dado un importe, desglosarlo en el menor número de billetes y monedas posible
- Útil en una máquina que tenga que devolver cambio (por ej. una expendedora de billetes de metro)

Algoritmo voraz: Problema del cambio

- Desglose de un importe
  - Dado un importe, desglosarlo en el menor número de billetes y monedas posible
  - Útil en una máquina que tenga que devolver cambio (por ej. una expendedora de billetes de metro)
- Se plantea utilizando un par de vectores:

- El vector de valores de monedas está ordenado de + a -
- Se supone que el importe se puede desglosar perfectamente

val: (índices 1 … N)

1					N
50	20	10	5	2	1

importe: 123

Desglose del importe según las monedas con los valores val, minimizando el número de monedas

Devuelve el vector de cantidades (con el número de monedas de cada tipo)

cant: (índices 1 … N)

1					N
2	1	0	0	1	1

Algoritmo voraz: ejemplo (II)

- El desglose del dinero se puede plantear como un algoritmo voraz ya que:
  - Conjunto de candidatos
    - Cada una de las monedas disponibles
  - Solución posible
    - Conjunto de monedas tal que sumen el importe a desglosar
  - Condición de factibilidad (EsValida)
    - En todo momento la suma de las monedas tiene que ser menor o igual al importe
  - Función de selección
    - Elegir, mientras sea posible, la moneda de valor mayor de entre las candidatas
  - Función objetivo
    - Minimizar el número de monedas

Algoritmo voraz: ejemplo (III)

- Desglose de un importe, tres posibles

```
cant = DesgloseVersion1 (importe, val, N)

cant = ceros (N); % inicializa a ceros
i=1;
1 while (i ≤ N & importe > 0)
   2  if (val(i) ≤ importe)
        cant(i) = cant(i) + 1;
        importe = importe - val(i);
    else 2
        i = i + 1;
    end 2
end 1
Fin
```

```
cant = DesgloseVersion2 (importe, val, N)

cant = ceros (N); % inicializa a ceros
i=1;
1 while (i ≤ N & importe > 0)
   2  while (val(i) ≤ importe)
        cant(i) = cant(i) + 1;
        importe = importe - val(i);
    end 2
    i = i + 1;
end 1
Fin
```

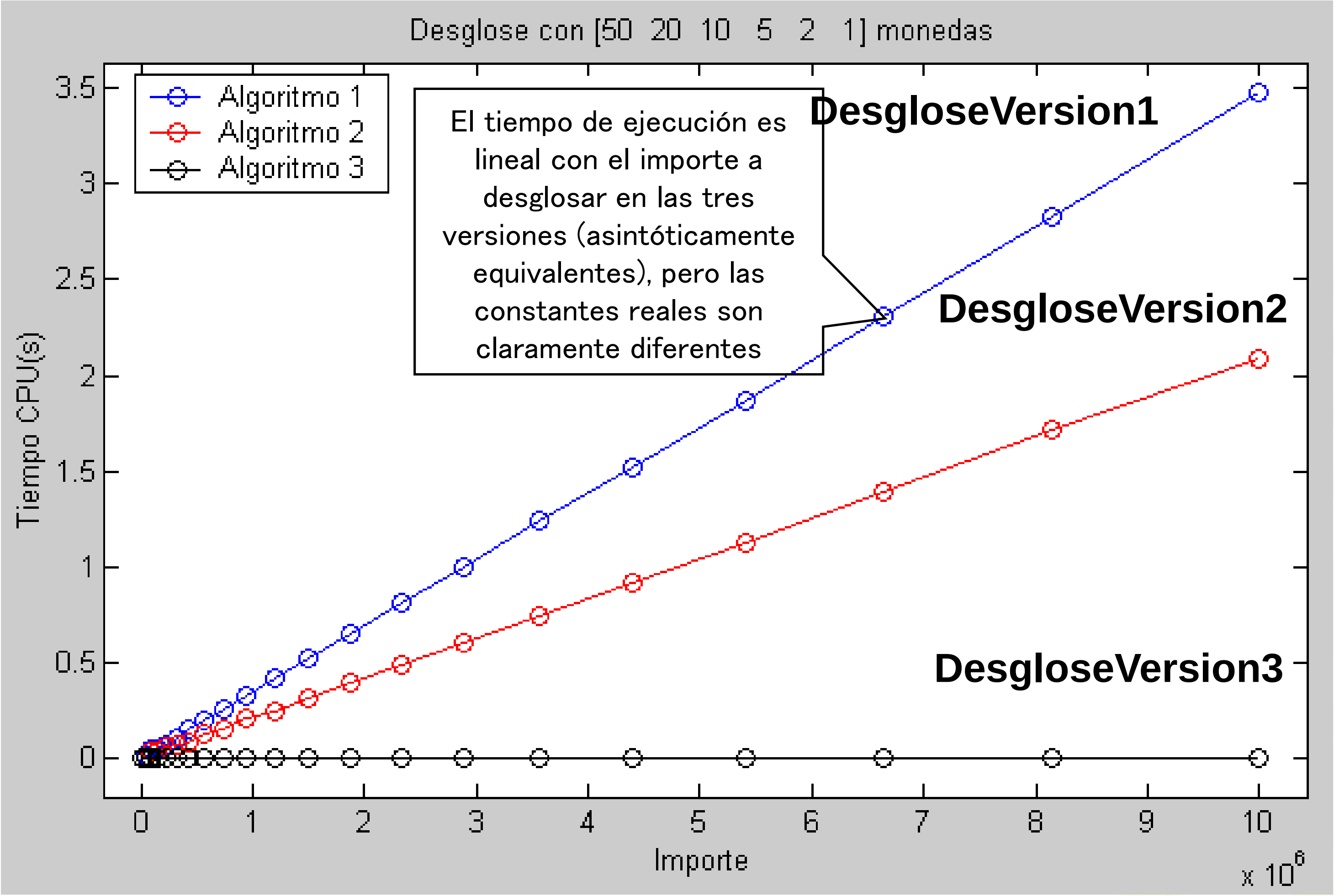
```
cant = DesgloseVersion3 (importe, val, N)

cant = ceros (N); % inicializa a ceros
i=1;
1 while (i ≤ N & importe > 0)
    cant(i) = floor (importe / val(i));
    importe = mod (importe, val(i));
    i = i + 1;
end 1
Fin
```

- ¿Son teóricamente equivalentes en cuanto a tiempo de ejecución?
- Desde el punto de vista práctico ¿existirán diferencias?

Algoritmo voraz: ejemplo (IV)

- Desglose de un importe



Algoritmo voraz: ejemplo (II)

- Problema desglose de actividades:
  - Dado un conjunto de actividades N, con sus horas de inicio y finalización, seleccionar el número máximo de actividades que puede realizar una persona, teniendo en cuenta que una persona sólo puede estar en una actividad en un tiempo determinado.

Actividad	A1	A2	A3	A4	A5	A6
H.Inicio	0	3	1	5	5	8
H. Fin	6	4	2	8	7	9

Algoritmo voraz: ejemplo (II)

- Problema desglose de actividades:

Actividad	A1	A2	A3	A4	A5	A6
H.Inicio	0	3	1	5	5	8
H. Fin	6	4	2	8	7	9

Si comenzamos por A0, sólo podemos hacer 2 actividades

Actividad	A3	A2	A1	A5	A4	A6
H.Inicio	1	3	0	5	5	8
H. Fin	2	4	6	7	8	9

Ordenadas por H. Fin

Algoritmo voraz: ejemplo (III)

Problema de la mochila

- Es un problema de optimización combinatoria, es decir, que busca la mejor solución entre un conjunto finito de posibles soluciones a un problema.

Algoritmo voraz: ejemplo (III)

Problema de la mochila

- Introducir un número de objetos en la mochila de tal manera que el peso no se excede y el valor es el máximo.

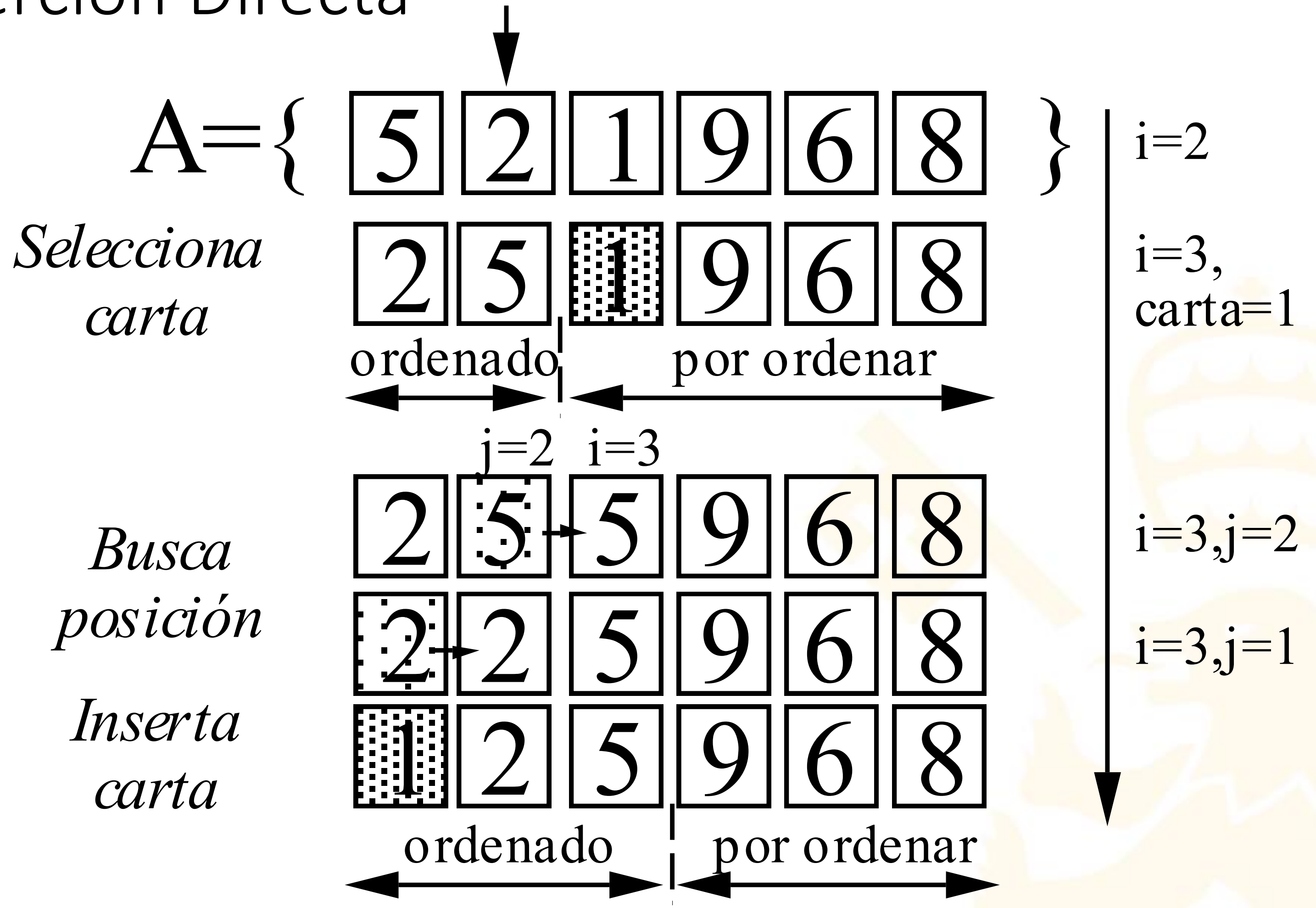
Algoritmo voraz: ejemplo (III)

Problema de la mochila – pasos a seguir

1. Calcular la densidad o beneficio por unidad de medida de cada elemento
2. Ordenar los elementos en base a este beneficio
3. Introducirlos en la mochila siempre que el peso lo permita
4. Añadir el siguiente elemento tan fraccionado como se pueda.

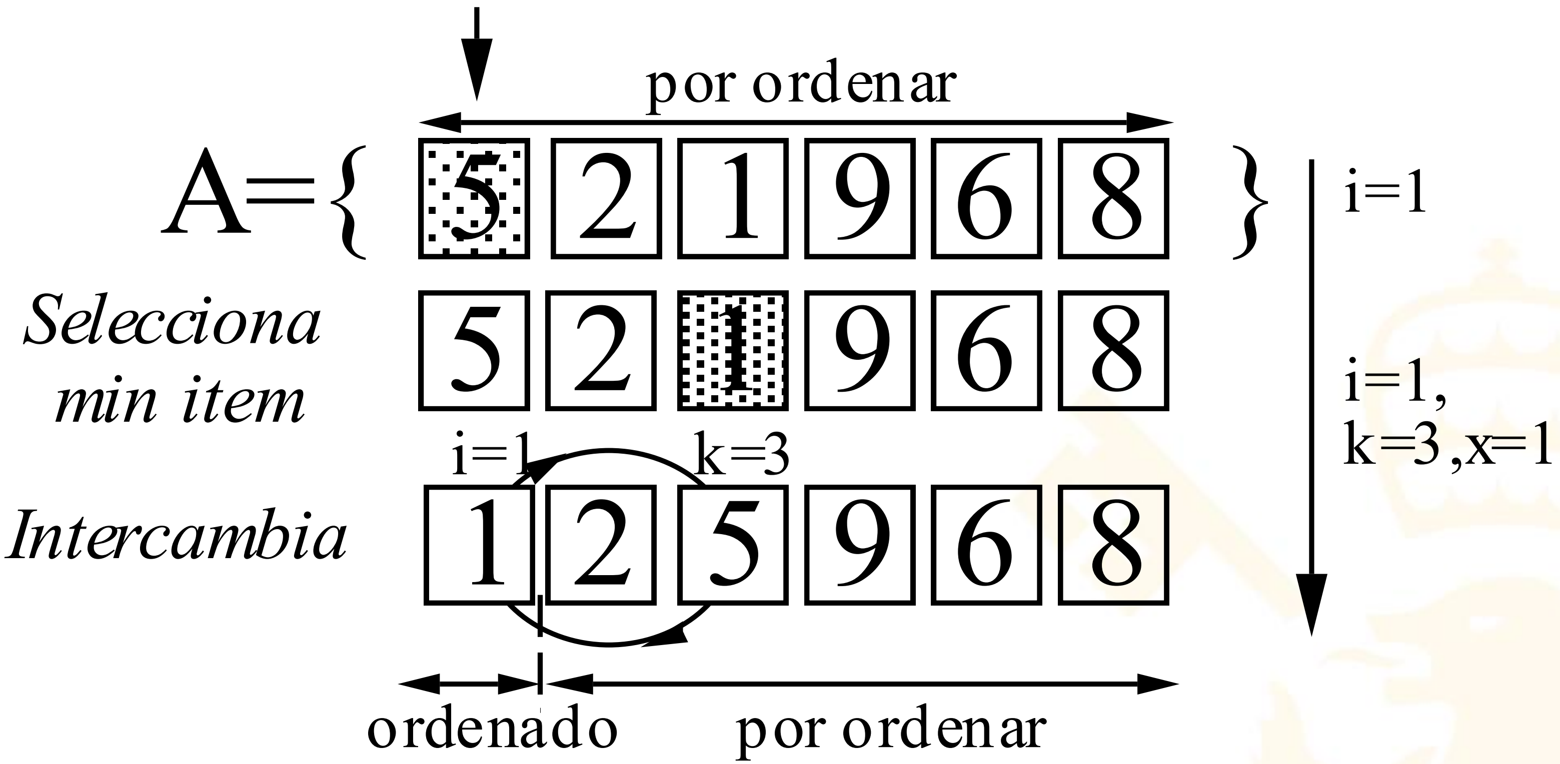
Algoritmo voraz:

- Inserción Directa



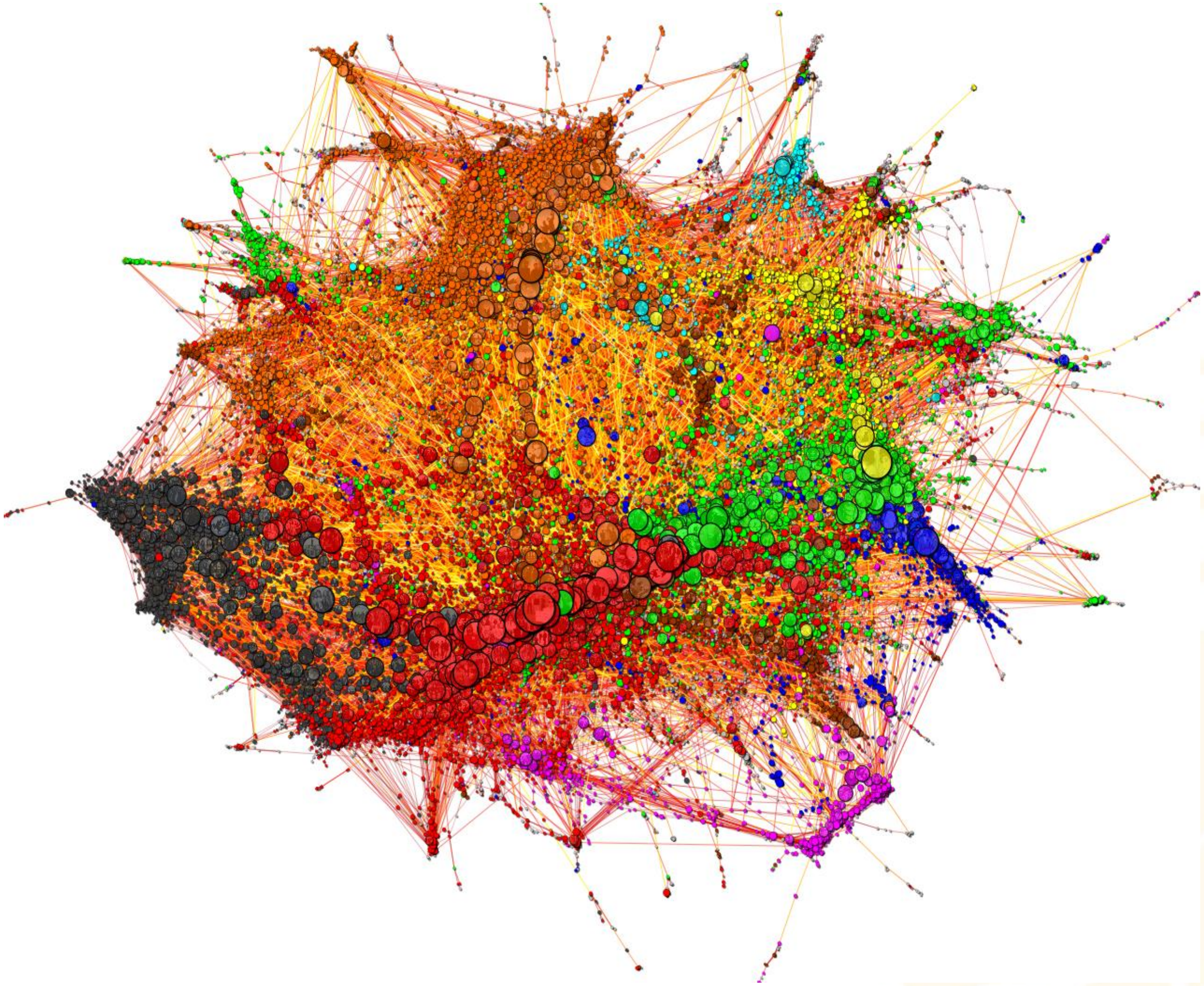
Algoritmo voraz:

- Selección Directa

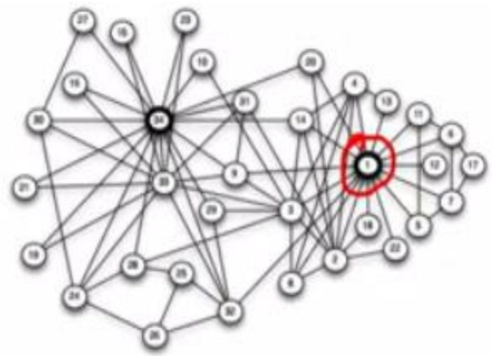


Algoritmo voraz:

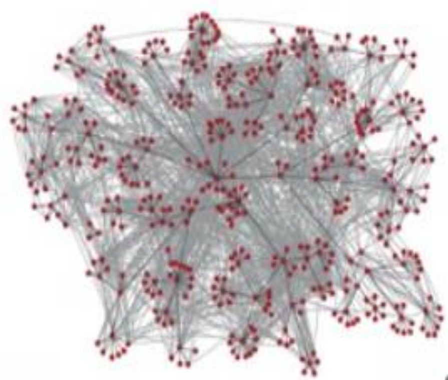
- Grafos



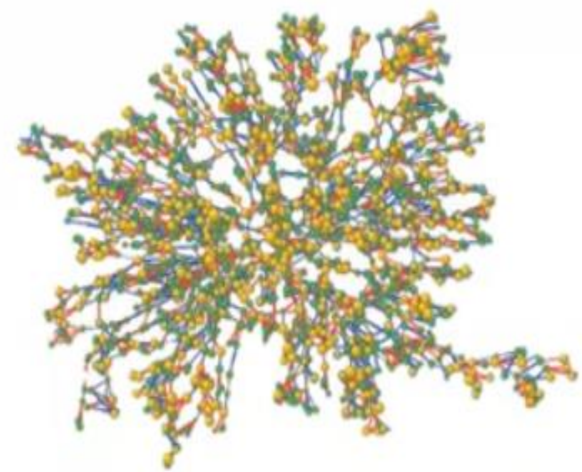
Grafos y caminos: Definiciones (I)



Friendship network in a 34-person karate club
[Zachary 1977]



E-mail communication network
among 436 HP employees [Adamic & Adar 2005]



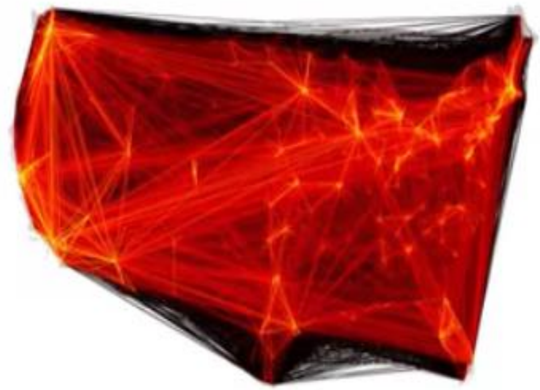
Network of friendship, marital tie, and
family tie among 2200 people
[Christakis & Fowler 2007]

Grafos y caminos: Definiciones (I)

Transportation and Mobility Networks

Network of direct flights around the world
[Bio.Diaspora]

Ann Arbor bus transportation network



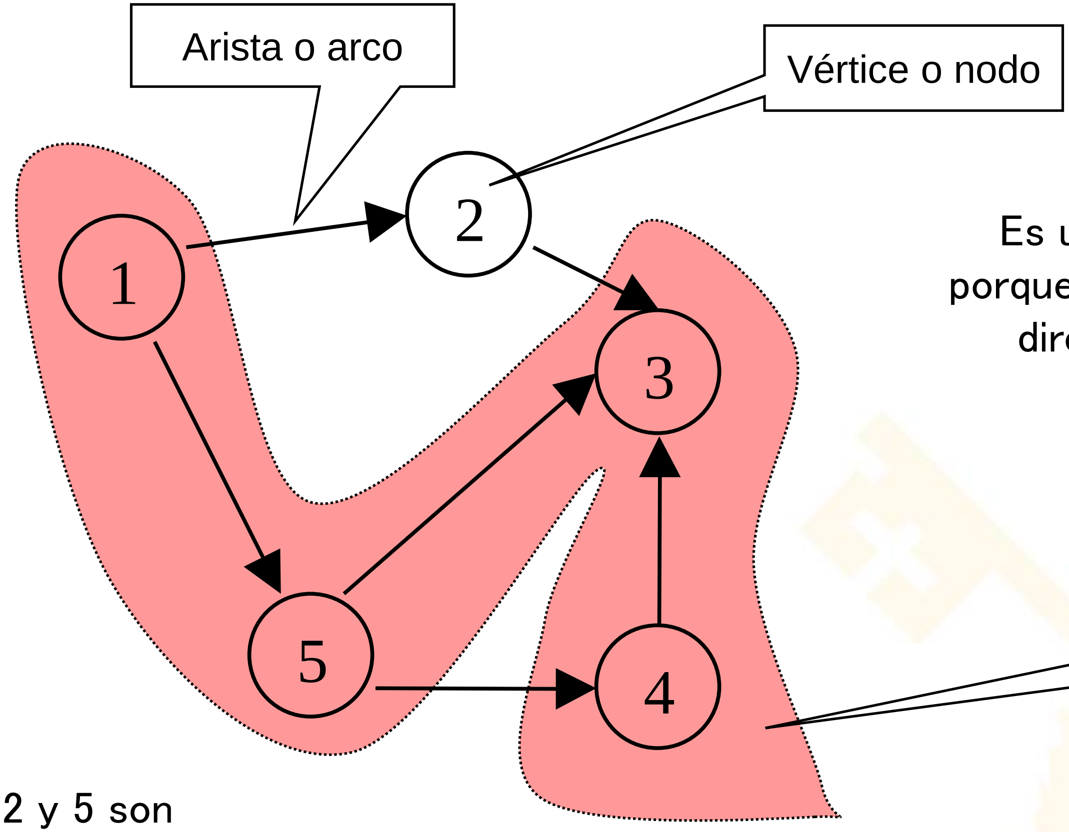
Human mobility network based
on location of dollar bills (Where's George)
[Thiemann et al. 2010]

Grafos y caminos: Definiciones (I)

- Grafo: G(V, A)
  - Conjunto de vértices (nodos) conectados a través de un conjunto de aristas (arcos)
  - Las aristas pueden tener un coste o peso asociado
  - Digrafo: grafo dirigido (aparecen flechas, aristas con dirección)
- Camino:
  - Secuencia de vértices conectados por aristas
  - Longitud sin pesos del camino: número de aristas
  - Longitud con pesos del camino: suma del coste de las aristas de ese camino
- Otros conceptos:
  - Los vértices adyacentes de un nodo son aquellos conectados mediante una única arista

Grafos y caminos: Definiciones (II)

- Ejemplo:



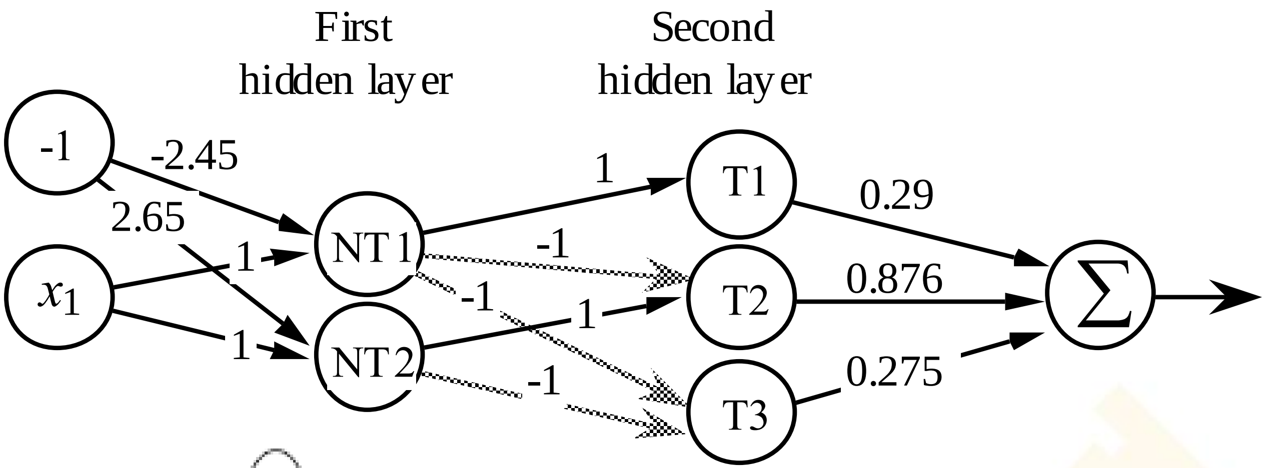
Es un grafo dirigido porque las aristas tienen dirección (flecha)

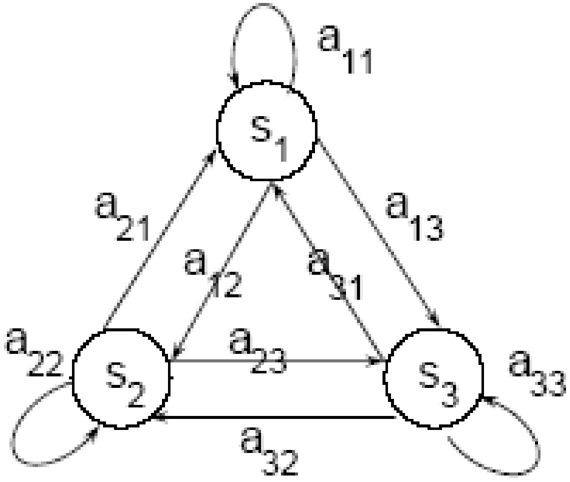
Los nodos 2 y 5 son vértices adyacentes del nodo 1 (vecinos)

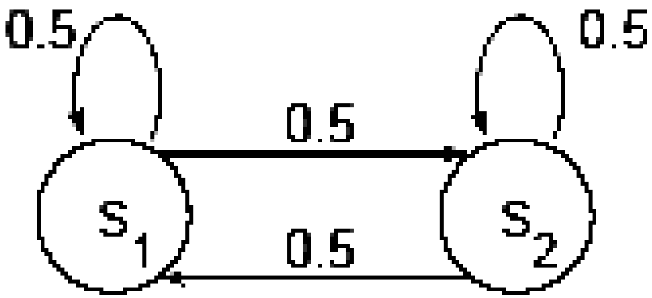
Grado de entrada del nodo 5: 1
Grado de salida del nodo 5: 2

Grafos y caminos: Ejemplos reales

- Ejemplo: MLP, modelos de markov ….



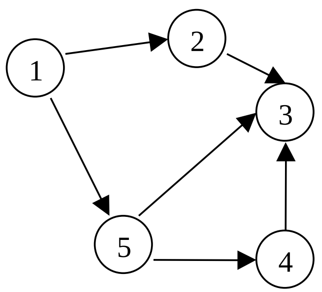




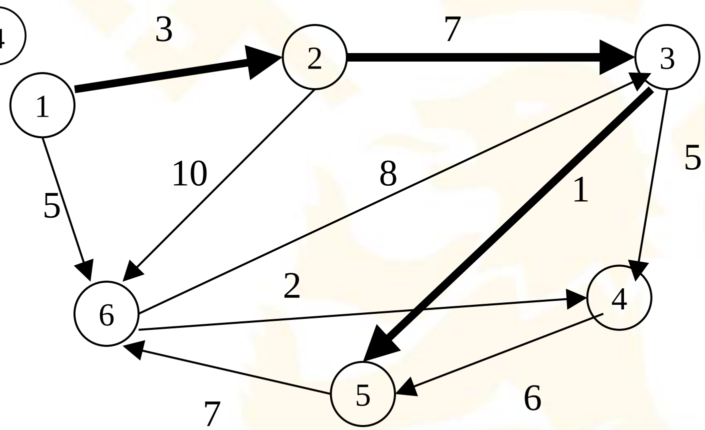
Grafos y caminos: Representación (I)

- Matriz de adyacencia:
  - Matriz bidimensional para representar grafos densos
    - Grafo sin pesos: Cada elemento representa la existencia (0/1) de la arista entre dos nodos
    - Grafo con pesos: Cada elemento representa el coste o peso de la arista entre dos nodo. Si no hay arista se guarda un infinito lógico
  - El espacio consumido es cuadrático en el número de vértices

	1	2	3	4	5
1	0	1	0	0	1
2	0	0	1	0	0
3	0	0	0	0	0
4	0	0	1	0	0
5	0	0	1	1	0

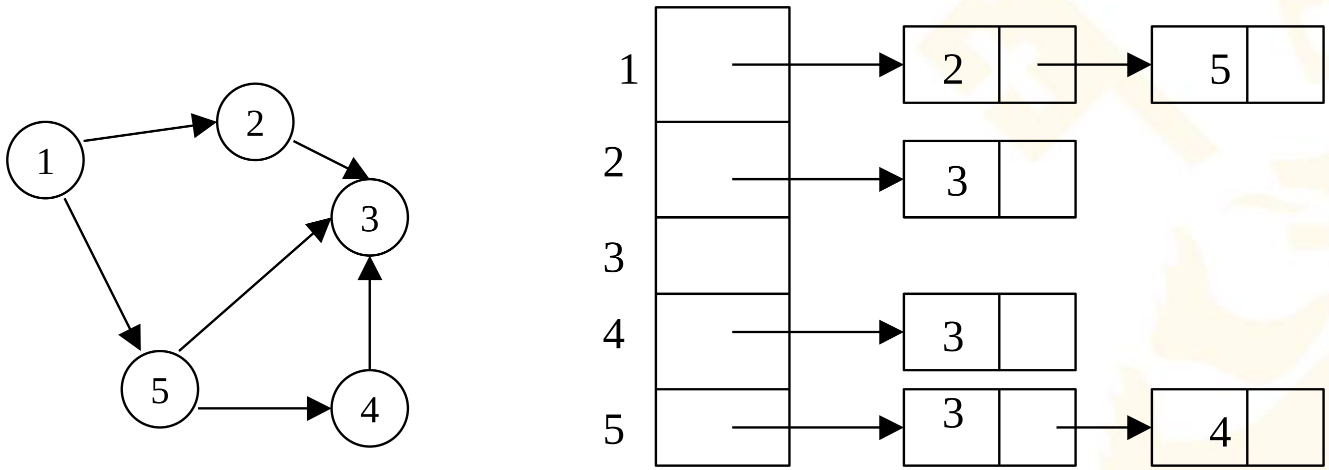


	1	2	3	4	5	6
1	0	3	∞	∞	∞	5
2	∞	0	7	∞	∞	10
3	∞	∞	0	5	1	∞
4	∞	∞	∞	0	6	∞
5	∞	∞	∞	∞	0	7
6	∞	∞	8	2	∞	0



Grafos y caminos: Representación (II)

- Lista de adyacencia:
  - Vector de listas enlazadas para representar grafos dispersos
  - Para cada vértice se tiene una lista con todos los nodos adyacentes
    - El orden de los nodos en las listas no es importante, se inserta por la cabeza
    - Si es un grafo con pesos también se guarda el peso de la arista en los elementos de las listas
  - El espacio consumido es lineal en el número de vértices



Grafos y caminos: Representación (III)

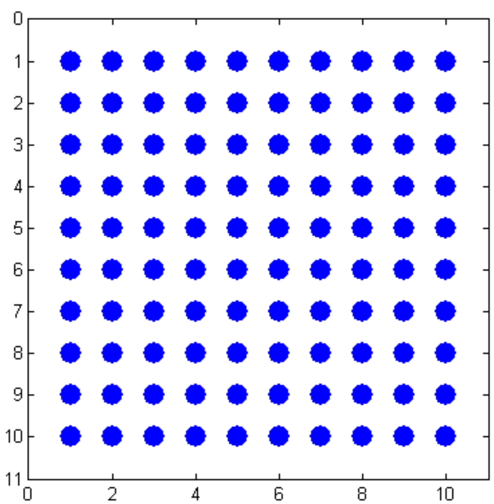
- Internamente los vértices no se gestionan directamente con su nombre, se les asigna un "número interno"
- Información adicional
  - Normalmente los nodos de un grafo suelen tener un **nombre asociado**
  - Pueden tener otros valores asociados como las coordenadas x.y si se representa gráficamente, el color, etc
- La Matriz de adyacencia no permite almacenar esa información de los vértices (la lista de adyacencia sí)
  - Si se utiliza la matriz de adyacencia es necesario tener un diccionario para guardar la información adicional de los vértices, así como la relación entre esa información y los números internos (típicamente mediante una tabla hash)
  - Sin embargo, aunque consuma más memoria en grafos dispersos y no permita almacenar la información adicional, los algoritmos son más sencillos y rápidos

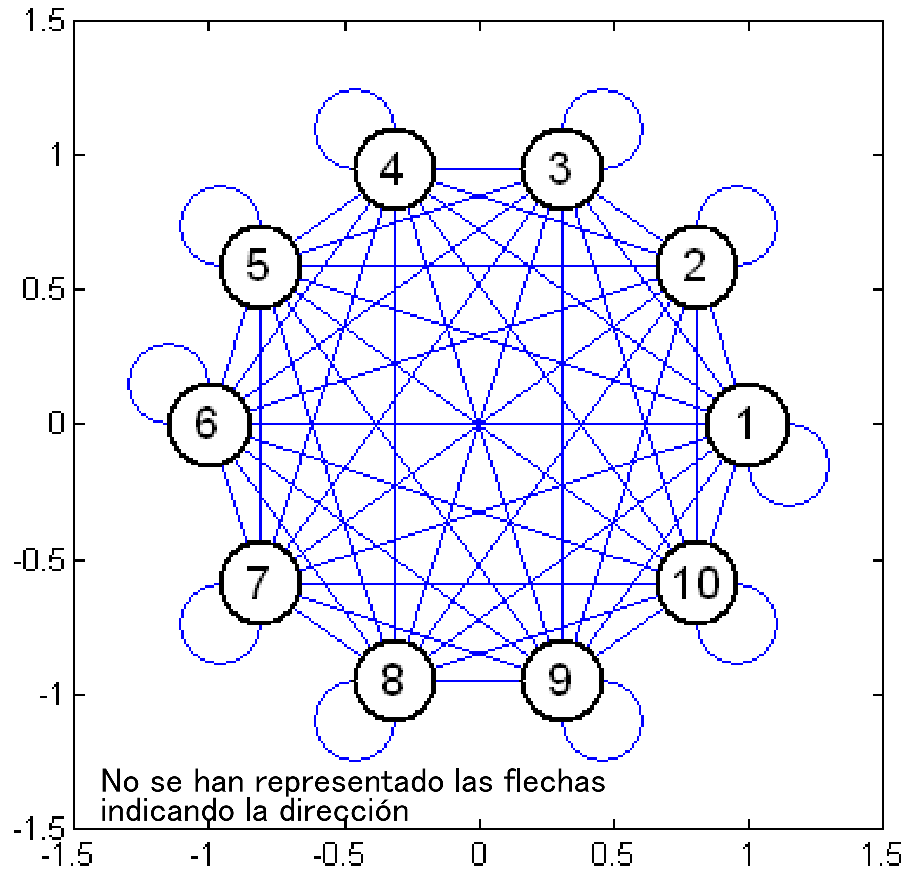
Grafos y caminos: Ejemplos (I)

- Grafo completo (número de aristas máximo posible) N*(N-1)/2:

Bytes (matriz normal): 800
Bytes (matriz dispersa): 1244 (+55%)

		2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	1

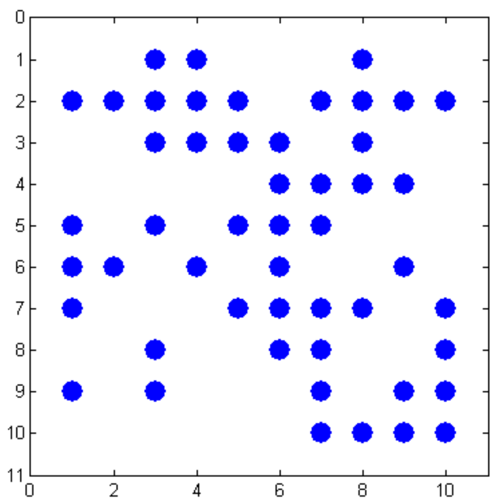




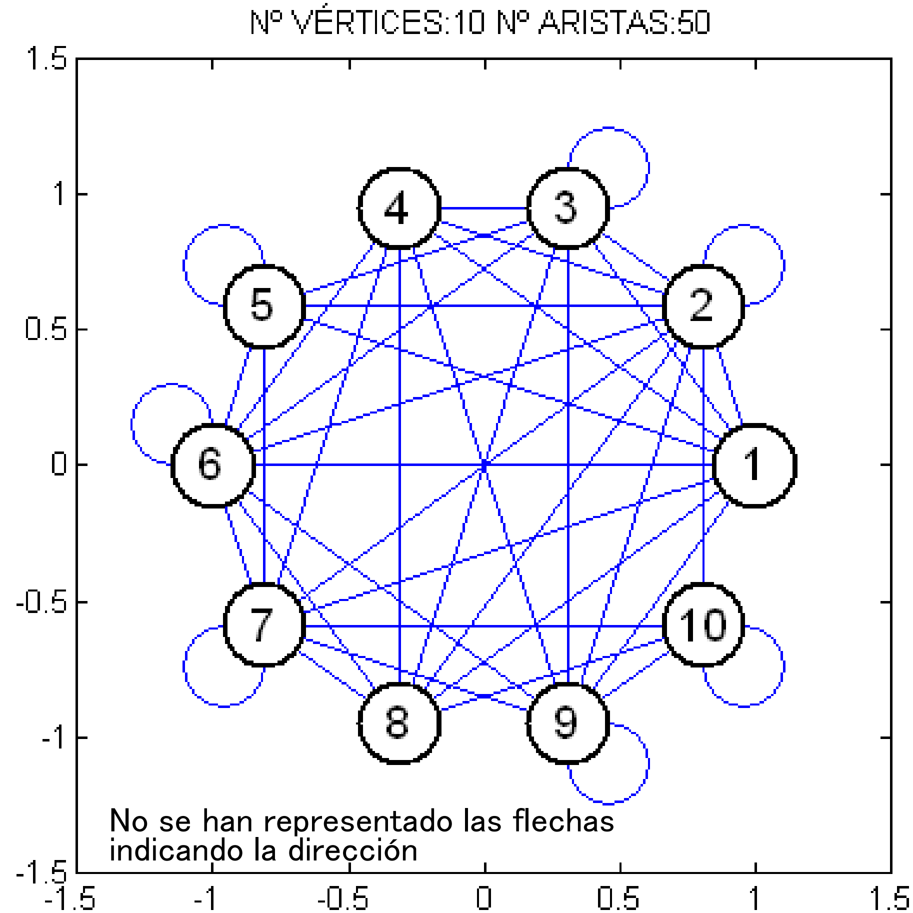
Grafos y caminos: Ejemplos (II)

- Grafo con la mitad del número de aristas máximo posible:

		2	3	4	5	6	7	8	9	10
1	0	0	1	1	0	0	0	1	0	0
2	1	1	1	1	1	0	1	1	1	1
3	0	0	1	1	1	1	0	1	0	0
4	0	0	0	0	0	1	1	1	1	0
5	1	0	1	0	1	1	1	0	0	0
6	1	1	0	1	0	1	0	0	1	0
7	1	0	0	0	1	1	1	1	0	1
8	0	0	1	0	0	1	1	0	0	1
9	1	0	1	0	0	0	1	0	1	1
10	0	0	0	0	0	0	1	1	1	1



Bytes (matriz normal): 800
Bytes (matriz dispersa): 644 (−19%)

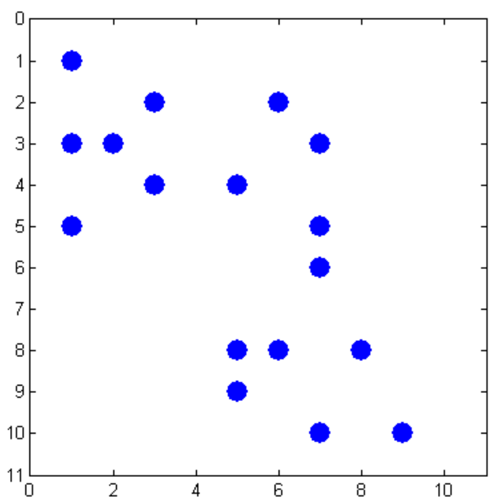


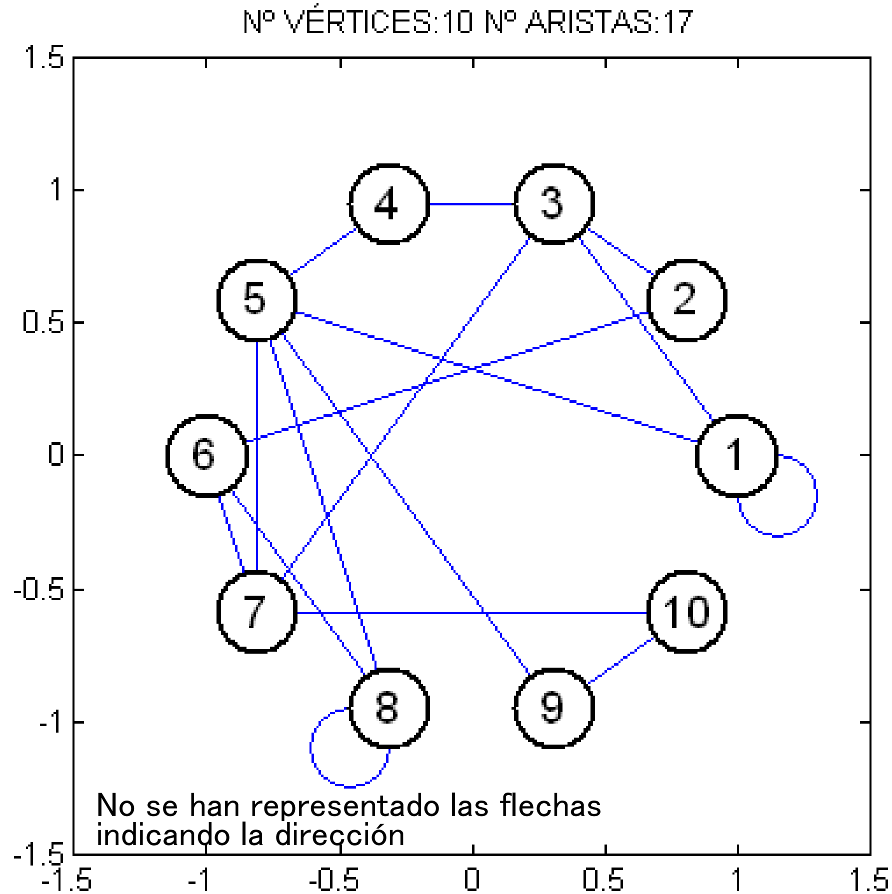
Grafos y caminos: Ejemplos (III)

- Grafo disperso, con un 17% del número de aristas máximo posible:

Bytes (matriz normal): 800
Bytes (matriz dispersa): 248 (−68%)

		2	3	4	5	6	7	8	9	10
1	1	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	1	0	0	0	0
3	1	1	0	0	0	0	1	0	0	0
4	0	0	1	0	1	0	0	0	0	0
5	1	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	1	0	1	0	0
9	0	0	0	0	1	0	0	0	0	0
10	0	0	0	0	0	0	1	0	1	0



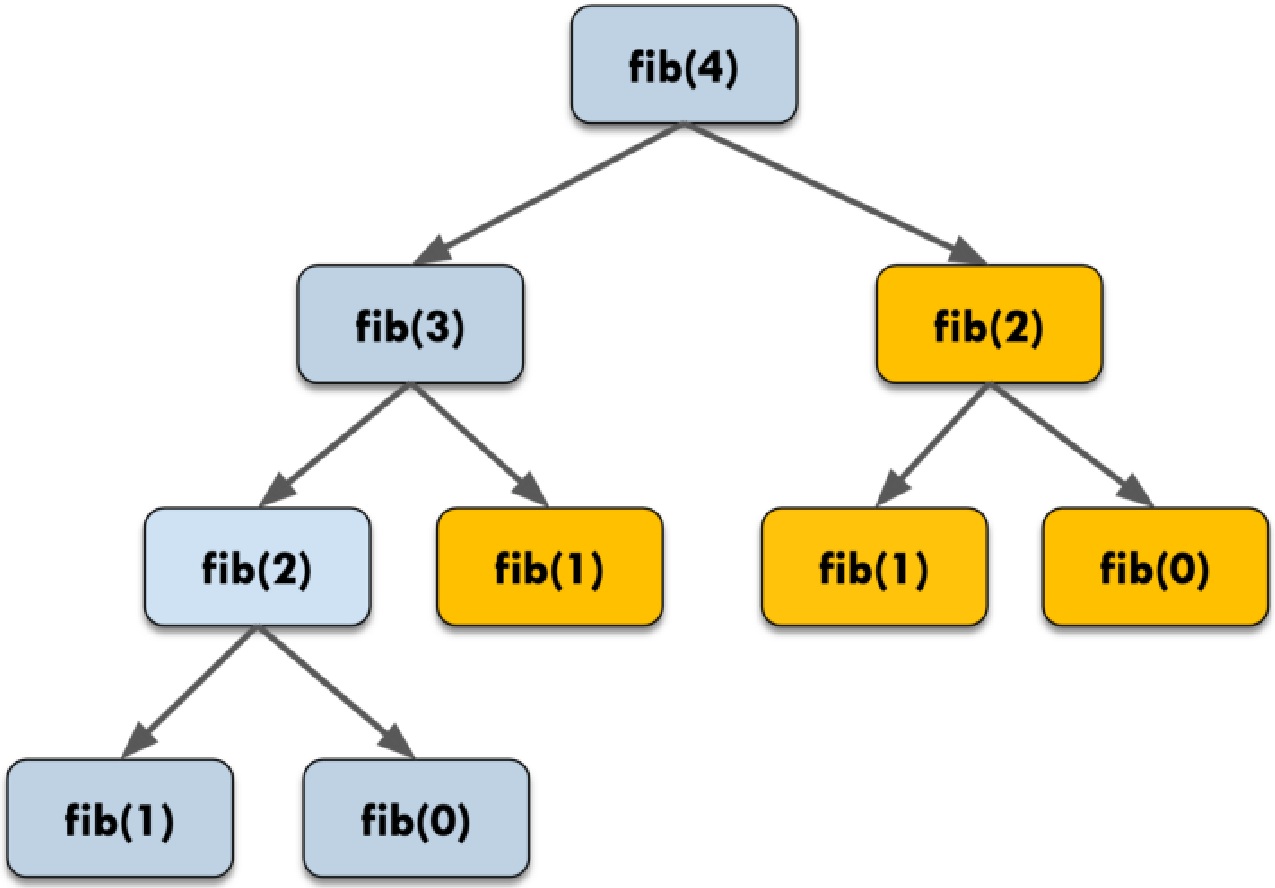


Programación dinámica

- Técnica que permite dividir un problema de optimización en subproblemas y aplicando el concepto de que la solución óptima del problema global es la solución óptima de cada subproblema.
- Optimización del paradigma divide y vencerás
- ‘Recuperamos’ la solución de otros problemas y la aplicamos, no la volvemos a calcular.
- $1+1+1+1+1+1+1=7$
- $\underbrace{1+1+1+1+1+1+1}+2= 9$

Programación dinámica

- Estructura óptima (fib(n)=fib(n-1)+fib(n-2))
- Sobreposición de problemas



Programación dinámica

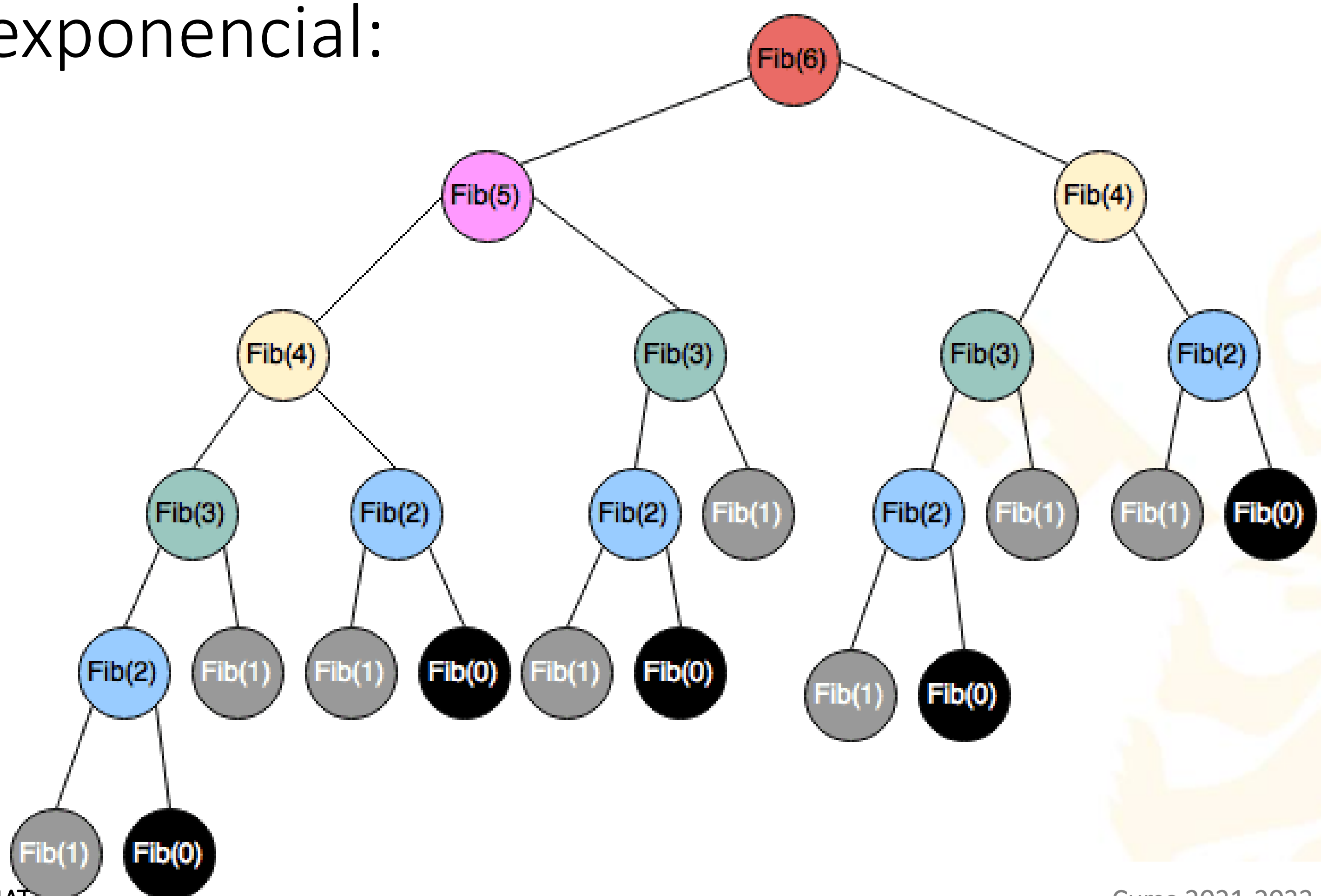
- Técnicas de memorización:
- Se trata de ir resolviendo recursivamente subproblemas e ir memorizando los resultados para utilizarlos en los siguientes subproblemas.
- Estructura top-down con memorización
- Ej: Si tenemos la serie de Fibonacci:
- 0,1,1,2,3,5,8,13,21,34,55….

```python
def Fibonacci(n):
    if n<1 :
        return "ERROR"
    elif n==1:
        return 0
    elif n==2:
        return 1
    else:
        return Fibonacci(n-1)+Fibonacci(n-2)
```

→ Divide y vencerás

Programación dinámica

- Si mostramos el árbol de recursión, aparecen 3 veces Fib(3), 5 veces Fib(2)…complejidad exponencial:



Programación dinámica

- Si mostramos el árbol de recursión, aparecen 3 veces Fib(3), 5 veces Fib(2)…complejidad exponencial:

```
import time
start=time.time()
Fibonacci(40)
end=time.time()
print(end-start)
```

```
55.022984981536865
```

```
def Fibonaccim(n,memo):
    if n<1 :
        return "ERROR"
    if n==1:
        return 0
    if n==2:
        return 1
    if not n in memo:
        memo[n]= Fibonaccim(n-1,memo)+Fibonaccim(n-2,memo)
    return memo[n]
```

```
import time
start=time.time()
dicti={}
print(Fibonaccim(150,dicti))
end=time.time()
print(end-start)
```

```
6161314747715278029583501626149
0.0009973049163818836
```

Programación dinámica

- Tabulación ej.: fib(n)=fib(n-1)+fib(n-2)

F1	F2	F3	F4	F5	F6
0	1				F4+F5

F1	F2	F3	F4	F5	F6
0	1			F3+F4	F4+F5

F1	F2	F3	F4	F5	F6
0	1		F2+F3	F3+F4	F4+F5

F1	F2	F3	F4	F5	F6
0	1	F1+F2	F2+F3	F3+F4	F4+F5

F1	F2	F3	F4	F5	F6
0	1	1	F2+F3	F3+F4	F4+F5

F1	F2	F3	F4	F5	F6
0	1	1	2	F3+F4	F4+F5

Programación dinámica

- Tabulación:
  - El planteamiento es el inverso, resolvemos los problemas de abajo a arriba
  - No tiene recursividad y resolvemos los problemas ‘pequeños’ hasta que por agregación de subproblemas resolvemos el problema principal.
  - Se resuelve rellenando una table, y en base a los resultados que obtengamos podremos resolver el problema global.

Programación dinámica

- Tabulación ej.: fib(n)=fib(n-1)+fib(n-2)

F1	F2	F3	F4	F5	F6
0	1	F1+F2			

F1	F2	F3	F4	F5	F6
0	1	1			

F1	F2	F3	F4	F5	F6
0	1	1	F2+F3		

F1	F2	F3	F4	F5	F6
0	1	1	2		

F1	F2	F3	F4	F5	F6
0	1	1	2	F3+F4	

F1	F2	F3	F4	F5	F6
0	1	1	2	3	

Programación dinámica

- Ejercicio: Dado un número N, encontrar las formas de expresarlo como sumas de 1, 3 y 4:
- N=5 → 6 formas: (4,1), (1,4), (3,1,1),(1,3,1),(1,1,3),(1,1,1,1,1)
- N=4 → 4 formas
- Realizar la versión recursiva, y luego ambas versiones, con memorización y con tabulación.

Programación dinámica

- Ejercicio: Dado un número N, encontrar las formas de expresarlo como sumas de 1, 3 y 4:
  - N=5 → 6 formas: (4,1), (1,4), (3,1,1),(1,3,1),(1,1,3),(1,1,1,1,1)
  - N=4 → 4 formas

Número factorizado

```python
# Versión recursiva:
def NumFactor(N):
    if N in (0,1,2):
        return 1
    if N == 3 : return 2
    else:
        rec1= NumFactor(N-1)
        rec2= NumFactor(N-3)
        rec3= NumFactor(N-4)
    return rec1+rec2+rec3

print(NumFactor(4))
```

```
4
```

Programación dinámica

- Ejercicio: Dado un número N, encontrar las formas de expresarlo como sumas de 1, 3 y 4:
  - N=5 → 6 formas: (4,1), (1,4), (3,1,1),(1,3,1),(1,1,3),(1,1,1,1,1)
  - N=4 → 4 formas
  - **CON MEMORIZACIÓN:**

```python
# Versión memorización:
def NumFactorM(N,dic):
    if N in (0,1,2):
        return 1
    if N == 3 : return 2

    else:
        if N not in dic:
            rec1= NumFactorM(N-1,dic)
            rec2= NumFactorM(N-3,dic)
            rec3= NumFactorM(N-4,dic)
            dic[N]=rec1+rec2+rec3
        return dic[N]



dic={}
print(NumFactorM(8,dic))
```

25

Programación dinámica

- Ejercicio: Dado un número N, encontrar las formas de expresarlo como sumas de 1, 3 y 4:
- N=5 → 6 formas: (4,1), (1,4), (3,1,1),(1,3,1),(1,1,3),(1,1,1,1,1)
- N=4 → 4 formas
- **CON TABULACIÓN:**

```
# Versión Tabulada:
def NumFactorTab(N):
    tb=[1,1,1,2]
    for i in range (4, N+1):
        tb.append(tb[i-1]+tb[i-3]+tb[i-4])

    return tb[N]

print(NumFactorTab(8))
```

```
25
```

Algoritmos Heurísticos y Algoritmos aproximados

- Algoritmo heurístico:
- Procedimiento que puede producir una solución no muy alejada de la óptima, o la óptima si tenemos suerte!
- Algoritmo aproximado:
- Procedimiento que proporciona una solución aproximada, que si bien no es la óptima, se puede 'medir' lo cerca que está.
- Los algoritmos heurísticos y aproximados no garantizan encontrar la solución óptima.
- Los algoritmos aproximados establecen una cota de error.

Algoritmos Heurísticos y Algoritmos aproximados

- Ejemplo Algoritmo heurístico:
- https://www.youtube.com/watch?v=L-DebhVbYyI

- Método de Montecarlo:
- https://www.youtube.com/watch?v=WJjDr67frtM