

Grafos (II)



Carlos Delgado Kloos
Ingeniería Telemática
Univ. Carlos III de Madrid

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 1

Definición

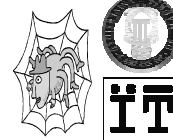


- Un **grafo** $G=(V,E)$ consiste en
 - un conjunto V de **nod**os (**vértices**) y
 - un conjunto E de **aristas** (**arcos**)
- Cada arista es un par (v,w) , con $v,w \in V$
- Si el par está ordenado, tenemos un **grafo dirigido**
- Se puede asociar a las aristas una tercera componente: **coste** (**peso**)

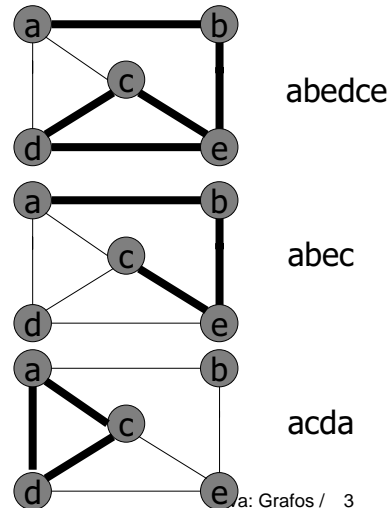
Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 2

Terminología



- **Camino:** secuencia de vértices, tal que dos vértices consecutivos son adyacentes
- **Camino simple:** aquel que no tiene vértices repetidos
- **Ciclo:** camino simple, excepto que el último vértice es el primero



Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 3

Problema del camino mínimo



- *Encontrar el camino más corto desde un vértice O a cualquier otro vértice*
- Consideramos grafos dirigidos
- Variantes:
 - Sin pesos
 - Con pesos positivos
 - Con pesos (positivos y) negativos
- Caso especial
 - Grafos Dirigidos Acíclicos



Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 4

Camino mínimo sin pesos

①



IT

- *Encontrar el camino más corto (medido por el número de aristas) desde un vértice O a cualquier otro vértice*
- Estrategia: Búsqueda por anchura

Copyright © 2001 cdk@it.uc3m.es

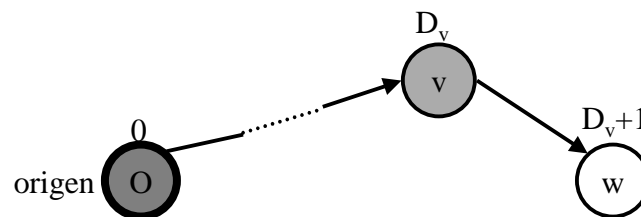
Java: Grafos / 5

Cálculo de D_w

①



IT



Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 6

Algoritmo

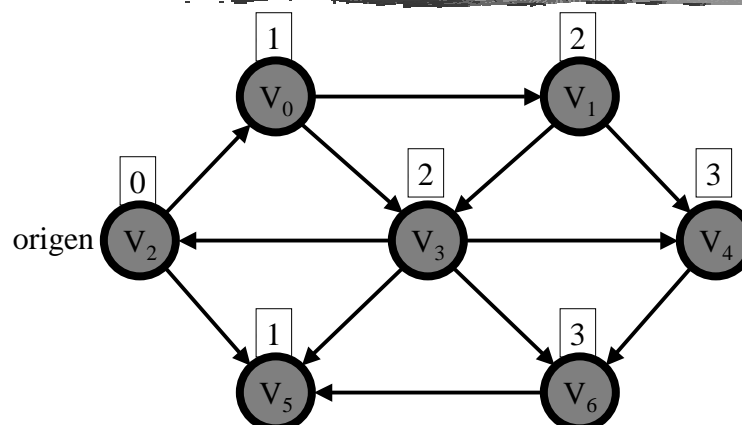


- Avanzar por niveles a partir del Origen asignando distancias según se avanza.
- Inicialmente, es $D_w = \infty$. Al visitar w se reduce al valor correcto $D_w = D_v + 1$.
- Desde cada v , visitamos a todos los nodos adyacentes a v (lista de adyacencia).
- Después nos fijamos en otro nodo u tal que $D_u = D_v$, y, si no existe, $D_u = D_v + 1$.

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 7

Algoritmo



Copyright © 2001 cdk@it.uc3m.es


Java: Grafos / 8

Implementación

①



IT

```
private void sinPesos (int nodoOrig){
    int v,w;
    Cola q=new ColaVec();
    limpiarDatos();
    tabla[nodoOrig].dist=0;
    q.insertar(new Integer(nodoOrig));
    try {...} 
    catch(DesbordamientoInferior e){}
}
```

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 9

Implementación

①



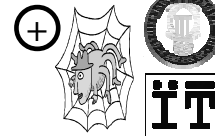
IT

```
try {
    while (!q.esVacia()) {
        v=((Integer) q.quitarPrimero()).intValue();
        ListaIter p=new
            ListaEnlazadaIter(tabla[v].ady);
        for(;p.estaDentro();p.avanzar()) {
            w=((Arista) p.recuperar()).dest;
            if (tabla[w].dist==INFINITO) {
                tabla[w].dist=tabla[v].dist+1;
                tabla[w].ant=v;
                q.insertar(new Integer(w)); }
        }
    }
}
```

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 10

Camino mínimo con pesos positivos

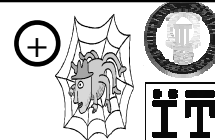


- *Encontrar el camino más corto (medido con su coste total) desde un vértice O a cualquier otro vértice*
- *El coste de cada arista es no negativo.*
- Estrategia: Algoritmo de Dijkstra

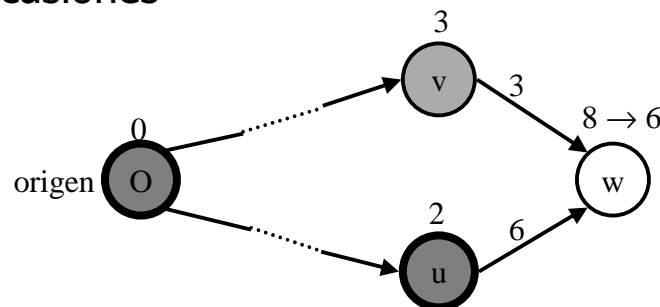
Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 11

Cálculo de D_w



- $D_w = D_v + c_{v,w}$
- D_w posiblemente se modifique en varias ocasiones



Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 12

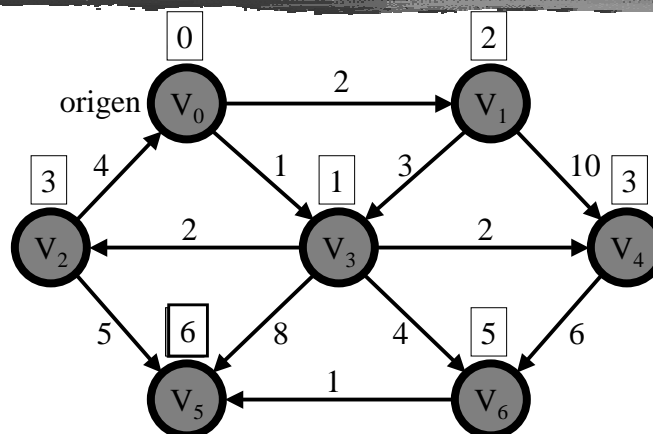
¿Cuál es el siguiente v?

- Utilizamos una cola de prioridad
- Encontramos el siguiente v, eliminando repetidamente el vértice de mínima distancia hasta que se obtenga un vértice no visitado.

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 13

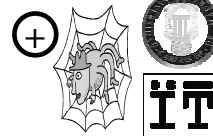
Algoritmo



Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 14

Camino

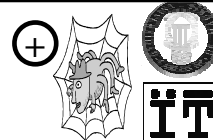


```
class Camino implements Comparable {
    int dest;
    int coste;
    static Camino infNeg=new Camino();
    Camino() {this(0);}
    Camino(int d) {this(d,0);}
    Camino(int d, int c) {dest=d;coste=c;}
    public boolean menorQue(Comparable lder) {
        return coste<((Camino)lder).coste;}
    public int compara (Comparable lder) {
        return coste<((Camino)lder).coste ? -1 :
            coste>((Camino)lder).coste ? 1 : 0;}
}
```

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 15

Implementación

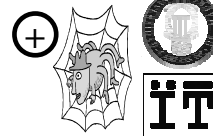


```
private boolean dijkstra(int nodoOrig){
    int v,w;
    ColaPrioridad cp=
        new MonticuloBinario(Camino.infNeg);
    Camino vrec; limpiarDatos();
    tabla[nodoOrig].dist=0;
    cp.insertar(new Camino(nodoOrig,0));
    try {...} ■■■■■—————▶
    catch(DesbordamientoInferior e){};
    return true;
}
```

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 16

Implementación



```
try {
    for (int nodosVistos=0;
        nodosVistos<numVertices;
        nodosVistos++) {
        do {if (cp.esVacia()) return true
            vrec=(Camino) cp.eliminarMin(); }
        while (tabla[vrec.dest].extra != 0);
        v=vrec.dest;
        tabla[v].extra=1;
        ListaIter p=
            new ListaEnlazadaIter(tabla[v].ady); ...
    }
```

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 17

Implementación

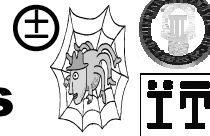


```
try {
    ...
    for (; p.estaDentro(); p.avanzar()) {
        w=((Arista)p.recuperar()).dest;
        int cvw=((Arista)p.recuperar()).coste;
        if (cvw<0) return false;
        if (tabla[w].dist>tabla[v].dist+cvw){
            tabla[w].dist=tabla[v].dist+cvw;
            tabla[w].ant=v;
            cp.insertar(new Camino(w,tabla[w].dist));
        }
    }
}
```

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 18

Camino mínimo con pesos (pos. y) negativos

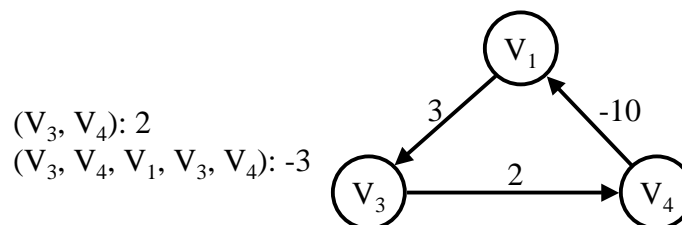
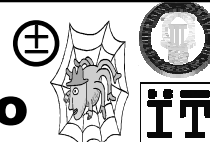


- *Encontrar el camino más corto (medido con su coste total) desde un vértice O a cualquier otro vértice*
- *El coste de cada arista puede ser positivo o negativo.*

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 19

Ciclos de coste negativo

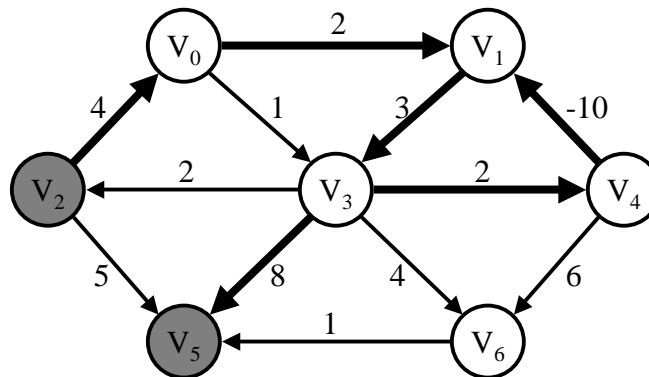


- ¡Coste no definido!
- También afecta a nodos fuera del ciclo

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 20

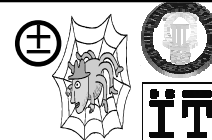
Ejemplo




Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 21

Implementación



```
private boolean negat(int nodoOrig){  
    int v,w;  
    Cola q=new ColaVec();  
    int cvw; limpiarDatos();  
    tabla[nodoOrig].dist=0;  
    q.insertar(new Integer(nodoOrig));  
    tabla[nodoOrig].extra++;  
    try {...}   
    catch(DesbordamientoInferior e){};  
    return true;  
}
```

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 22

Implementación



```
try {
    while (!q.esVacia()) {
        v=((Integer) q.quitarPrimero()).intValue();

        if(tabla[v].extra++>2*numVertices)
            return false; //existe ciclo

        ListaIter p=new
            ListaEnlazadaIter(tabla[v].ady);

        ...
    }
}
```

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 23

Implementación

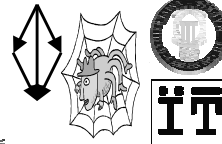


```
try {
    for(;p.estaDentro();p.avanzar()) {
        w=((Arista) p.recuperar()).dest;
        cvw=((Arista) p.recuperar()).coste;
        if (tabla[w].dist>tabla[v].dist+cvw){
            tabla[w].dist=tabla[v].dist+cvw;
            tabla[w].ant=v;
            if (tabla[w].extra++%2==0) {
                q.insertar(new Integer(w)); }
            else tabla[w].extra++;
        }
    }
}
```

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 24

DAGs

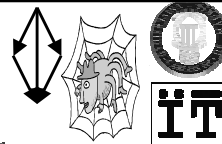


- DAG=Directed Acyclic Graph
- GDA=Grafo Dirigido Acíclico
 - Grafo dirigido sin ciclos

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 25

Orden topológico

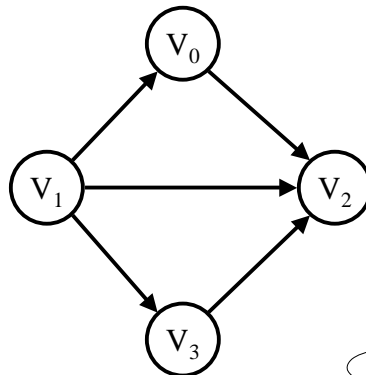
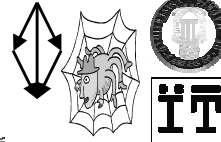


- Un orden topológico ordena los vértices de un GDA de tal forma que si hay un **camino de v a w**, w aparece **después** de v en el orden.
- Un GDA tiene **al menos un** orden topológico
- Un GDA **puede** tener **varios** órdenes topológicos

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 26

Orden topológico



(V_1, V_0, V_3, V_2)

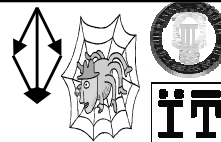
(V_1, V_3, V_0, V_2)

dos vértices consecutivos
no tienen que ser adyacentes

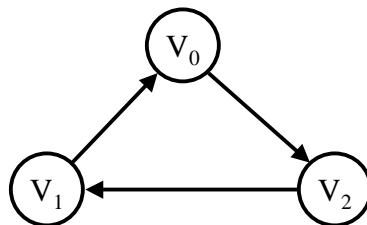
Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 27

Orden topológico



- Un **grafo con ciclos** no se puede ordenar topológicamente.
- Para dos vértices v y w en un ciclo, hay un camino de v a w y uno de w a v .

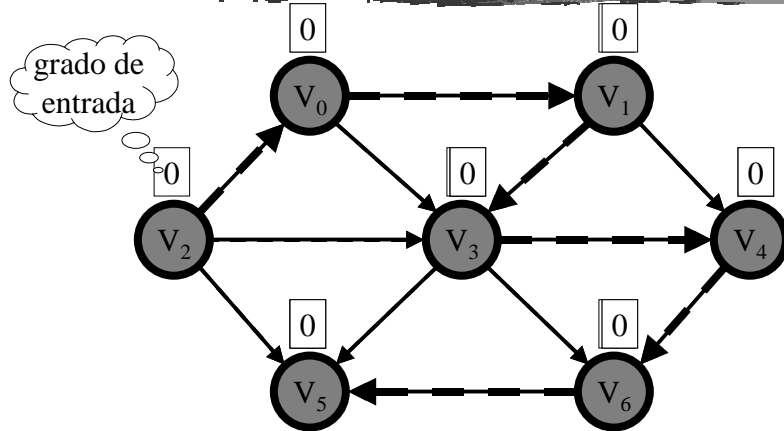


$(V_2, V_1) \not\subseteq (V_1, V_0, V_2)$

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 28

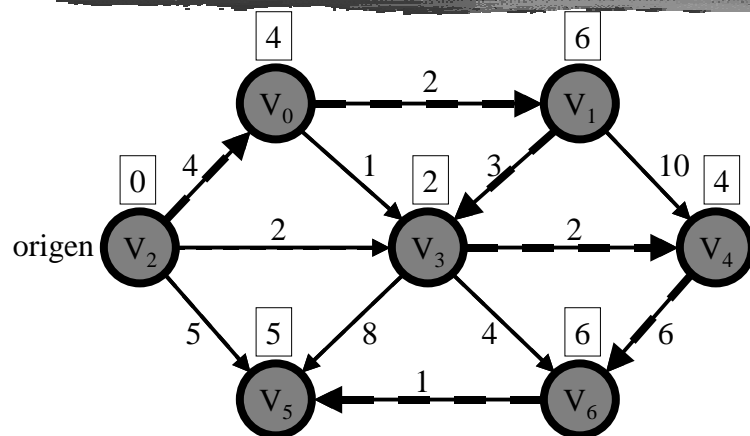
Algoritmo



Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 29

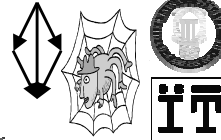
Camino mínimo



Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 30

Implementación

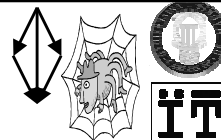


```
private boolean aciclico(int nodoOrig){
    int v,w,iteraciones=0;
    Cola q=new ColaVec();
    limpiarDatos();
    tabla[nodoOrig].dist=0;
    try { ■ ■ ■ ■ ■ }
    catch (DesbordamientoInferior e) {}
    return iteraciones==numVertices;
}
```

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 31

Implementación



```
for (v=0; v<numVertices; v++)
    ListaIter p=
        new ListaEnlazadaIter(tabla[v].ady);
    for (; p.estaDentro(); p.avanzar())
        tabla[(Arista)p.recuperar()].dest.extra++;
}

try {
    for (v=0; v<numVertices; v++)
        if (tabla[v].extra==0)
            q.insertar(new Integer(v));
    ...
}
```

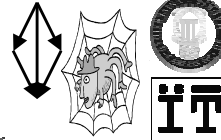
cálculo de los
grados de entrada

inserción de
vértices de
grado 0 en
la cola

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 32

Implementación

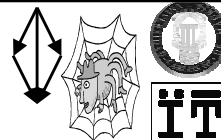


```
try {  
    ...  
    for (iteraciones=0;  
        !q.esVacia();  
        iteraciones++) {  
        v=((Integer) q.quitarPrimero()).intValue;  
        ListaIter p=  
            new ListaEnlazadaIter(tabla[v].ady);  
        ...  
    }  
}
```

Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 33

Implementación

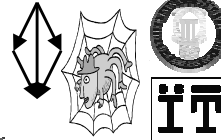


```
try {  
    ...  
    for (; p.estaDentro(); p.avanzar()) {  
        w=((Arista)p.recuperar()).dest;  
        if (--tabla[w].extra==0)  
            q.insertar(new Integer(w));  
        if (tabla[v].dist==INFINITO) continue;  
        int cvw=((Arista)p.recuperar()).coste;  
        if (tabla[w].dist>tabla[v].dist+cvw) {  
            tabla[w].dist=tabla[v].dist+cvw;  
            tabla[w].ant=v;  
        }  
    }  
}
```

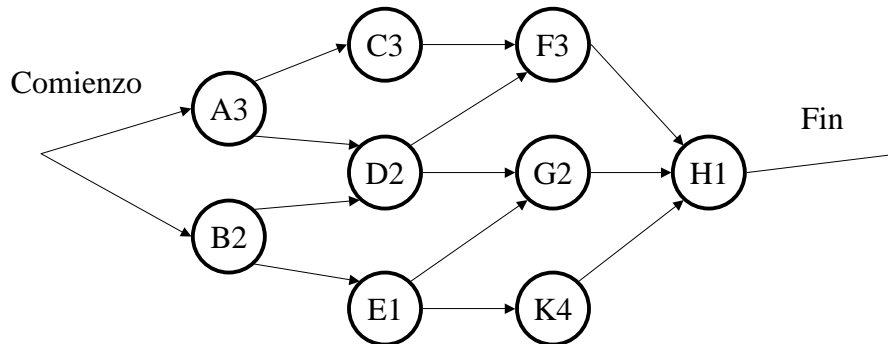
Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 34

Análisis de caminos críticos



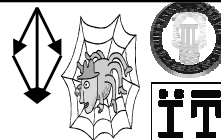
■ Grafo de actividades



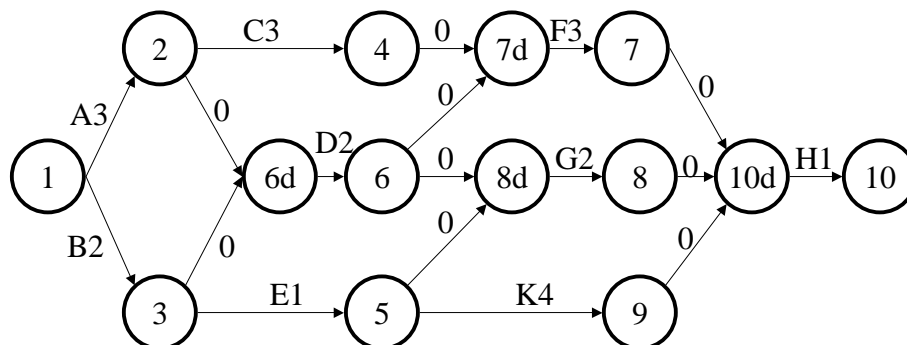
Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 35

Análisis de caminos críticos



■ Grafo de eventos



Copyright © 2001 cdk@it.uc3m.es

Java: Grafos / 36