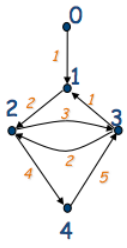
Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Tema 13. Algoritmos voraces en Grafos

Algorítmica y Complejidad

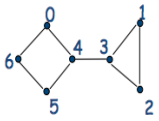
1

Grafos



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

Grafo Dirigido (GD)



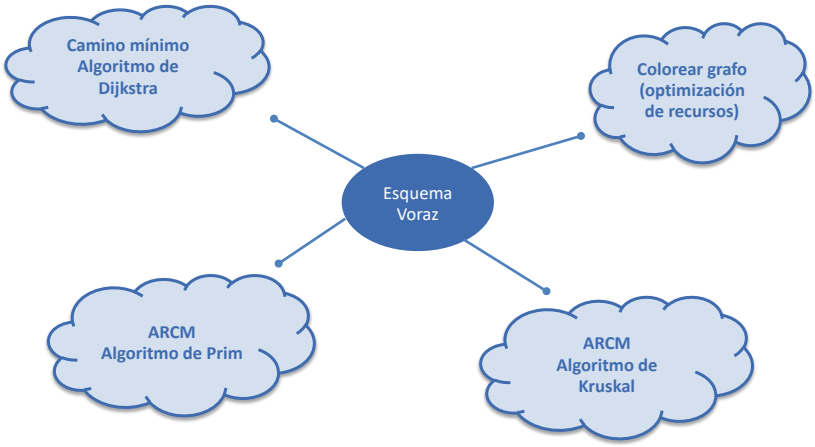
	0	1	2	3	4	5	6
0	f	f	f	f	v	f	v
1	f	f	v	v	f	f	f
2	f	v	f	v	f	f	f
3	f	v	v	f	v	f	f
4	v	f	f	v	f	v	f
5	f	f	f	f	v	f	v
6	v	f	f	f	f	v	f

Grafo NO dirigido (GN)

2

Algoritmo de Dijkstra (camino mínimo)

Algunos problemas planteados



1. Dijkstra

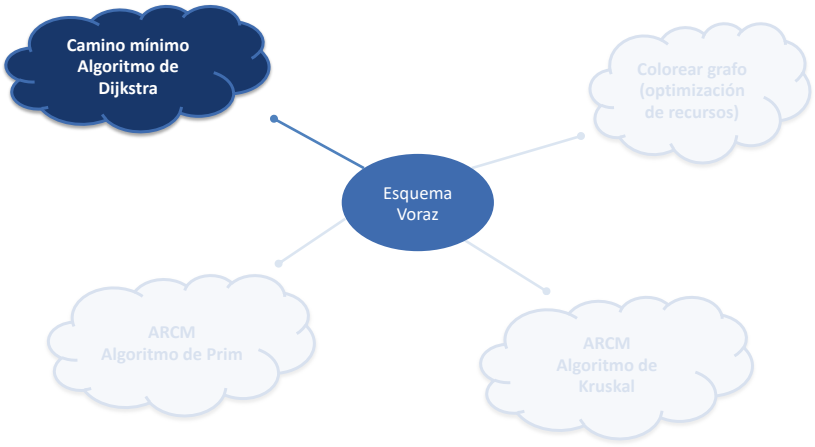
1 2 3 4

3

3

Algoritmo de Dijkstra (camino mínimo)

Algunos problemas planteados



1. Dijkstra

1 2 3 4

4

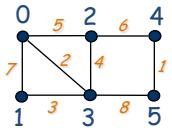
4

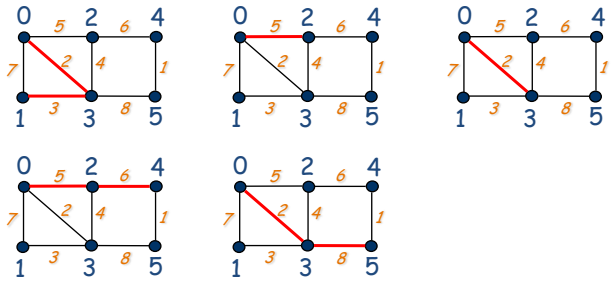
Algoritmo de Dijkstra (camino mínimo)

Enunciado

- **Dijkstra**: *Buscar los caminos mínimos entre un vértice **origen** y el resto de vértices del grafo (costes positivos)*

CM_Dijkstra(0, ...)





Algoritmo voraz $\Theta(N^2)$

El enfoque voraz planteado por Dijkstra se basa en que la selección local del vértice *No Visitado* cuyo coste (calculado hasta el momento) sea menor nos llevará a encontrar la solución global de caminos mínimos desde origen hasta el resto de vértices

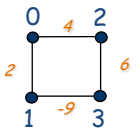
5

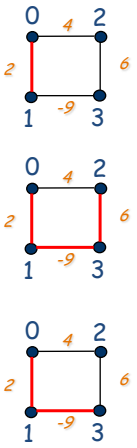
Algoritmo de Dijkstra (camino mínimo)

Enunciado

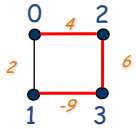
- **Costes negativos? No funciona**

CM_Dijkstra(0, ...)





?



Al seleccionar siempre el nodo con menor distancia (de los No Visitados) pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista negativa (*para los vértices visitados el algoritmo no considera rutas alternativas posteriores*)

6

Algoritmo de Dijkstra (camino mínimo)

Implementación

```
void Dijkstra(int origen, int[][]grafo, double[] menorCoste,
                          int[] verticeMasCercano){

     boolean[] visitados = new boolean[grafo.length];
     …
}
```

	0	1	2	3	4
0	∞	1	∞	∞	∞
1	1	∞	1	7	∞
2	∞	1	∞	2	∞
3	∞	7	2	∞	5
4	∞	∞	∞	5	∞

verticeMasCercano

Vértice visitado más cercano a N-1

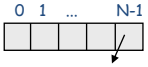
(vértice previo a N-1 en el camino encontrado de origen a N-1)

menorCoste

Coste del menor camino calculado desde origen a N-1

(pasando sólo por vértices ya visitados)

visitados



true si ya se ha obtenido un camino mínimo de origen a N-1

(el vértice ya ha sido seleccionado en alguna iteración del algoritmo voraz)

7

Algoritmo de Dijkstra (camino mínimo)

Implementación

inicializarDijkstra(0, …)

```
private static final int SIN_PREVIO = -1;
private static final int SIN_ARISTA = Integer.MAX.VALUE;

void inicializarDijkstra(int origen, int[][] grafo, double[] menorCoste,
                         boolean[] visitados, int[] verticeMasCercano){
     for (int i=0; i<grafo.length; i++){
        visitados[i]= false;
        menorCoste[i]= grafo[origen][i];
        if (grafo[origen][i]!= SIN_ARISTA)
           verticeMasCercano[i] = origen;
        else
           verticeMasCercano[i] = SIN_PREVIO;
     }
     visitados[origen] = true;
  }
}
```

	0	1	2	3	4
0	∞	1	∞	∞	∞
1	1	∞	1	7	∞
2	∞	1	∞	2	∞
3	∞	7	2	∞	5
4	∞	∞	∞	5	∞

verticeMasCercano

0	1	...	4
-1	0	-1	-1	-1

menorCoste

0	1	...	4
∞	1	∞	∞	∞

visitados

0	1	...	4
t	f	f	f	f

origen (vértice 0)

8

Algoritmo de Dijkstra (camino mínimo)

Implementación

```
int seleccionarVertice(int[][] grafo, double[] menorCoste, boolean[] visitados){

        int vertice=0;
        double menor;
        menor = Integer.MAX_VALUE;
        for (int i=0; i<grafo.length; i++)
            if (!visitados[i] && (menorCoste[i]<menor)){
                menor = menorCoste[i];
                vertice = i;
            }
        return vertice;
    }
```


9

Algoritmo de Dijkstra (camino mínimo)

Implementación

```
void Dijkstra(int origen, int[][] grafo, double[] menorCoste,
              int[] verticeMasCercano){

      boolean[] visitados = new boolean[grafo.length];
      int vertice;
      inicializarDijkstra(origen, grafo, menorCoste, visitados, verticeMasCercano);
      for (int i=1; i<grafo.length; i++){
          vertice = seleccionarVertice(grafo, menorCoste, visitados);
          visitados[vertice] = true;
          for (int j=0; j<grafo.length; j++)
              if (!visitados[j])
                  if (menorCoste[j] > (menorCoste[vertice] + grafo[vertice][j])){
                      menorCoste[j] = menorCoste[vertice] + grafo[vertice][j];
                      verticeMasCercano[j] = vertice;
                  }
      }
  }
```

Algoritmo voraz **$\Theta(N^2)$**

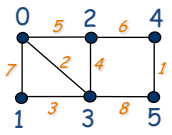
10

Algoritmo de Dijkstra (camino mínimo)

Seguimiento

Dijkstra(0, grafo, menorCoste, verticeMasCercano)

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



visitados

0	1	2	3	4	5
†	f	f	f	f	f

menorCoste

0	1	2	3	4	5
∞	7	5	2	∞	∞

verticeMasCercano

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

i=1; vertice:=seleccionarVertice(...); (3 *)*

visitados

0	1	2	3	4	5
†	f	f	†	f	f

menorCoste

0	1	2	3	4	5
∞	5	5	2	∞	10

verticeMasCercano

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

i=2; vertice:=seleccionarVertice(...); (1 *)*

visitados

0	1	2	3	4	5
†	†	f	†	f	f

menorCoste

0	1	2	3	4	5
∞	5	5	2	∞	10

verticeMasCercano

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

i=3; vertice:=seleccionarVertice(...); (2 *)*

visitados

0	1	2	3	4	5
†	†	†	†	f	f

menorCoste

0	1	2	3	4	5
∞	5	5	2	11	10

verticeMasCercano

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

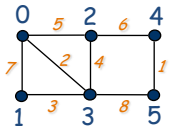
11

Algoritmo de Dijkstra (camino mínimo)

Seguimiento

Dijkstra(0, grafo, menorCoste, verticeMasCercano)

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



i=4; vertice:=seleccionarVertice(...); (5 *)*

visitados

0	1	2	3	4	5
†	†	†	†	f	†

menorCoste

0	1	2	3	4	5
∞	5	5	2	11	10

verticeMasCercano

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

i=5; vertice:=seleccionarVertice(...); (4 *)*

visitados

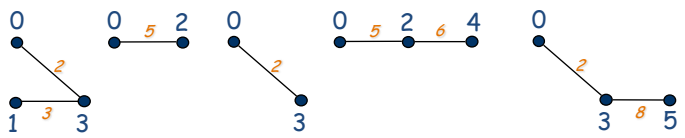
0	1	2	3	4	5
†	†	†	†	†	†

menorCoste

0	1	2	3	4	5
∞	5	5	2	11	10

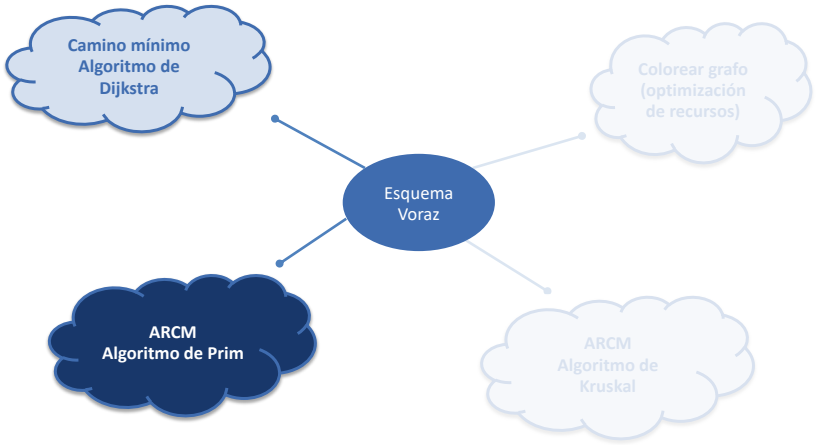
verticeMasCercano

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



12

Algunos problemas planteados

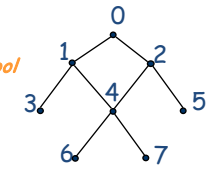


13

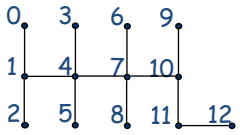
Conceptos

- **Árbol:**
  - Un **árbol** es un GN conexo y acíclico (un **grafo acíclico** es aquel que no tiene ciclos)

NO es árbol



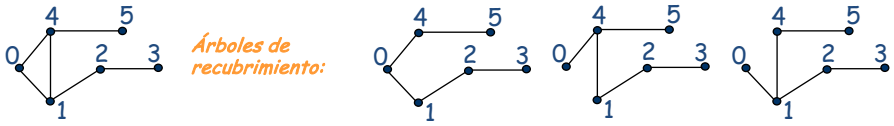
Es árbol



14

Conceptos

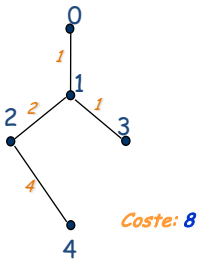
- **Árbol de recubrimiento:**
  - Sea G=<V,A> un grafo GN, y sea G_A=<V',A'> un árbol. Si V=V' y A'⊆A entonces decimos que G_A es un **árbol de recubrimiento** de G (spanning tree).



15

Conceptos

- **Coste del árbol:**
  - Sea G=<V,A,f> un árbol valorado. Denominamos **coste del árbol** a la suma de los costes de todas sus aristas o arcos:

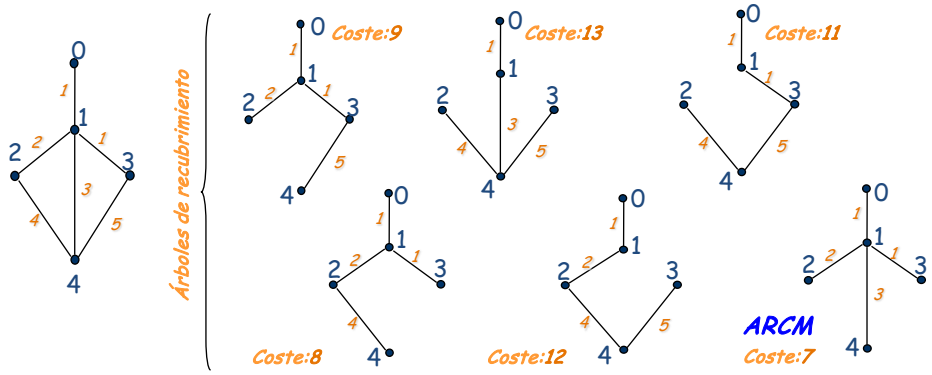


$$\sum_{\forall a \in A} f(a)$$

16

Conceptos

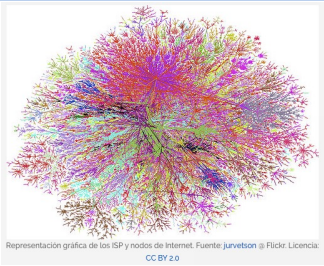
- **Árbol de Recubrimiento de Coste Mínimo (ARCM):**
  - Sea G un grafo (GD). Llamamos **árbol de recubrimiento de coste mínimo** de G al árbol de recubrimiento de G cuyo coste sea menor. Un grafo puede tener varios ARCM.

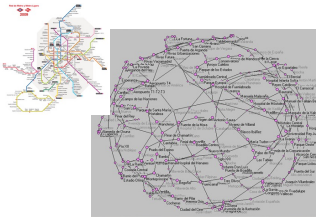


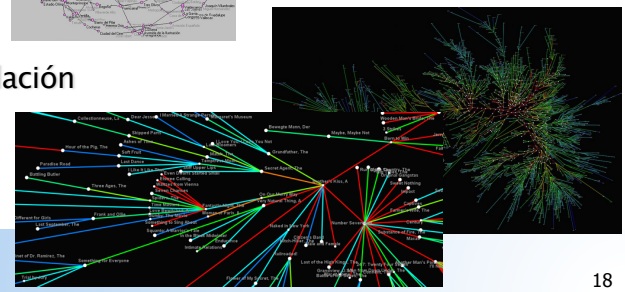
17

Conceptos

- **Ejemplos:**
  - Línea de comunicaciones
  - Red de metro
  - Sistema de recomendación



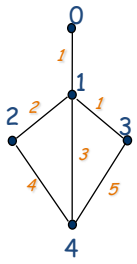




18

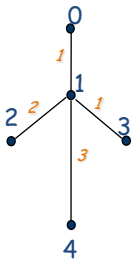
Conceptos

- **Propiedad MST (Minimum-cost Spanning Tree):**
  - Sea G=<V,A,f> un GN conexo valorado y sea U un subconjunto estricto de V (U⊂V, U≠V). Si $a=(u,v)$ ∈ A es una arista tal que u ∈ U y v ∈ V-U y además no existe ninguna otra arista con un extremo en U y otro fuera de U (en V-U) cuyo coste sea menor que el de la arista a, entonces existe algún ARCM que contiene a la arista a.



V = {0,1,2,3,4}
U = {0,1,2,3}
V-U = {4}
a = (1,4)

Coste:7
ARCM



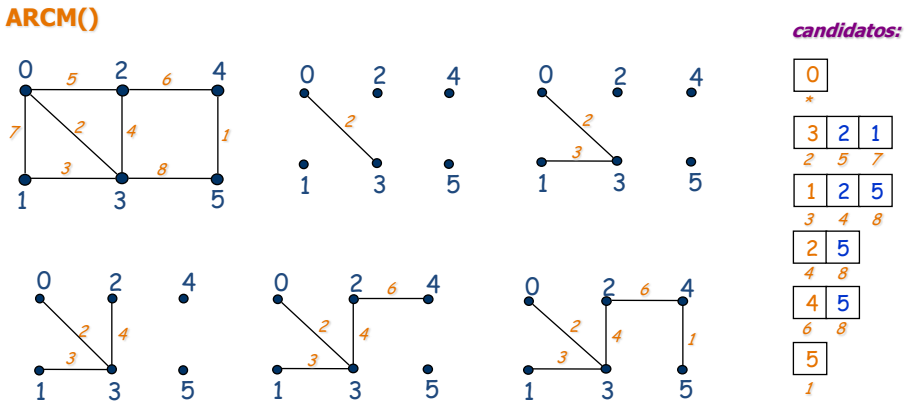
Enunciado

- **ARCM (GN conexos valorados). algoritmo de Prim :**
  - Esquema de búsqueda de vértices no visitados. En cada iteración del algoritmo voraz se selecciona uno de los vértices todavía no visitados.
  - Selección de siguiente vértice a visitar basado en la propiedad MST (Minimum-cost Spanning Tree).
  - De entre los vértices candidatos (los no visitados, adyacentes a alguno de los visitados) se selecciona el vértice cuya distancia o coste a cualquiera de los vértices ya visitados sea la menor.

Algoritmo de Prim (ARCM) Grafo No dirigido (GN).

Implementación

- **ARCM: esquema general del algoritmo de Prim (GN). Seguimiento**

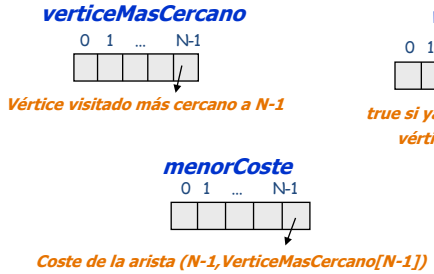


21

Algoritmo de Prim (ARCM) Grafo No dirigido (GN).

Implementación

```
ArrayList<Arista> Prim(int[][] grafo){
   int[] menorCoste = new int[grafo.length];
   int[] verticeMasCercano = new int[grafo.length];
   boolean[] visitados = new boolean[grafo.length];
        …
}
```




```
public class Arista {

    private int origen;
    private int destino;
    private int coste;
    public Arista(int origen,
                  int destino,
                  int coste){
        this.origen = origen;
        this.destino = destino;
        this.coste=coste;
    }

     /* getters y setters */
}
```

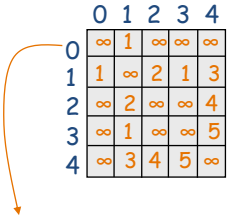

22

Algoritmo de Prim (ARCM) Grafo No dirigido (GN).

Implementación

Primer vértice visitado: 0

```
void inicializar(int[][] grafo, int[] menorCoste,
                 int[] verticeMasCercano, boolean[] visitados){

   visitados[0] = true;

   for (int i=1; i<numVertices; i++){
       menorCoste[i] = grafo[0][i];
       verticeMasCercano[i] = 0;
       visitados[i] = false;
   }
}
```



visitados

0	1	…	4
t	f	f	f	f

verticeMasCercano

0	1	…	4
?	0	0	0	0

menorCoste

0	1	…	4
?	1	∞	∞	∞

23

Algoritmo de Prim (ARCM) Grafo No dirigido (GN).

Implementación

```
int seleccionarVertice(int[] menorCoste, boolean[] visitados){

    int vertice = 0;
    double minimo = Integer.MAX_VALUE;
    for (int i=1; i<menorCoste.length; i++)
        if (!visitados[i] && menorCoste[i]<minimo) {
            minimo = menorCoste[i];
            vertice = i;
        }
    return vertice;
}
```

Θ(N)

24

Algoritmo de Prim (ARCM) Grafo No dirigido (GN).

Implementación

```
void actualizarEstructuras(int[][] grafo, int vertice, int[] menorCoste,
                           int[] verticeMasCercano, boolean[] visitados){

    visitados[vertice]=true;
    for (int i=1; i<grafo.length; i++){
        if ( (!visitados[i]) && (grafo[vertice][i] < menorCoste[i]) ){
            verticeMasCercano[i] = vertice;
            menorCoste[i] = grafo[vertice][i];
        }
    }
}
```

Θ(N)

25

Algoritmo de Prim (ARCM) Grafo No dirigido (GN).

Implementación

```
ArrayList<Arista> Prim(int[][] grafo){
  int[] menorCoste = new int[grafo.length];
  int[] verticeMasCercano = new int[grafo.length];
  boolean[] visitados = new boolean[grafo.length];

  int vertice;
  ArrayList<Arista> arcm = new ArrayList<Arista>();

  inicializar(grafo, menorCoste, verticeMasCercano, visitados);
  for (int v=1; v<grafo.length; v++){
     vertice = seleccionarVertice(menorCoste, visitados);
     arcm.add(new Arista(verticeMasCercano[vertice], vertice, menorCoste[vertice]));
     actualizarEstructuras(grafo, vertice, menorCoste, verticeMasCercano, visitados);
  }
  return arcm;
}
```

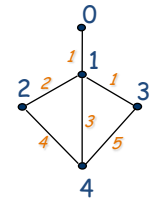
Algoritmo voraz **Θ(N^2)**

26

Algoritmo de Prim (ARCM) Grafo No dirigido (GN).

Seguimiento

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



visitados

0	1	2	3	4
t	f	f	f	f

verticeMasCercano

0	1	2	3	4
?	0	0	0	0

menorCoste

0	1	2	3	4
?	1	∞	∞	∞

v=1; vertice = seleccionarVertice(menorCoste, visitados) **vertice: 1**

ARCM (0,1)

actualizarEstructuras(grafo, 1, menorCoste, verticeMasCercano, visitados)

visitados

0	1	2	3	4
t	t	f	f	f

verticeMasCercano

0	1	2	3	4
?	0	1	1	1

menorCoste

0	1	2	3	4
?	1	2	1	3

v=2; vertice = seleccionarVertice(menorCoste, visitados) **vertice: 3**

ARCM (0,1) (1,3)

actualizarEstructuras(grafo, 3, menorCoste, verticeMasCercano, visitados)

visitados

0	1	2	3	4
t	t	f	t	f

verticeMasCercano

0	1	2	3	4
?	0	1	1	1

menorCoste

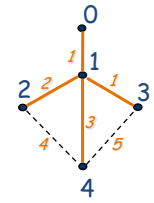
0	1	2	3	4
?	1	2	1	3

27

Algoritmo de Prim (ARCM) Grafo No dirigido (GN).

Seguimiento

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



visitados

0	1	2	3	4
t	t	f	t	f

verticeMasCercano

0	1	2	3	4
?	0	1	1	1

menorCoste

0	1	2	3	4
?	1	2	1	3

v=3; vertice = seleccionarVertice(menorCoste, visitados) **vertice: 2**

ARCM (0,1) (1,3) (1,2)

actualizarEstructuras(grafo, 2, menorCoste, verticeMasCercano, visitados)

visitados

0	1	2	3	4
t	t	t	t	f

verticeMasCercano

0	1	2	3	4
?	0	1	1	1

menorCoste

0	1	2	3	4
?	1	2	1	3

v=4; vertice = seleccionarVertice(menorCoste, visitados) **vertice: 4**

ARCM (0,1) (1,3) (1,2) (1,4)

actualizarEstructuras(grafo, 4, menorCoste, verticeMasCercano, visitados)

visitados

0	1	2	3	4
t	t	t	t	t

verticeMasCercano

0	1	2	3	4
?	0	1	1	1

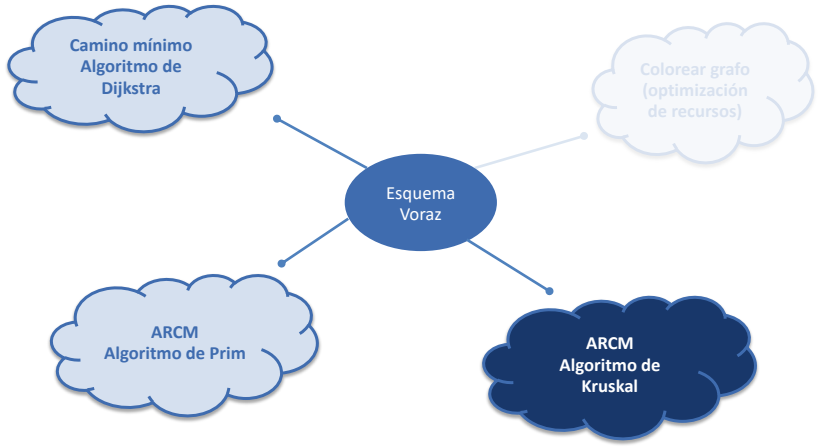
menorCoste

0	1	2	3	4
?	1	2	1	3

28

Algoritmo de Kruskal (ARCM) Grafo No dirigido (GN).

Algunos problemas planteados



29

Algoritmo de Kruskal (ARCM) Grafo No dirigido (GN).

Implementación

```
int[][] Kruskal(int[][] grafo){
    int numAristasARCM;
    Arista arista;
    int[][] gcopia = copia(grafo);  // copia del grafo original
    int[][] arcm = new int[grafo.length][grafo.length];  //arcm sin aristas
    for (int i=0; i<arcm.length; i++)
        for (int j=0; j<arcm.length; j++)
            arcm[i][j]=SIN_ARISTA;
    numAristasARCM = 0;
    do{
        // selecciona la arista de menor peso y la elimina de gcopia
        arista = seleccionarYEliminarAristaMenor(gcopia);
        if (!existeCamino(arcm, arista.getOrigen(), arista.getDestino())){
            arcm[arista.getOrigen()][arista.getDestino()]=arista.getCoste();
            arcm[arista.getDestino()][arista.getOrigen()]=arista.getCoste();
            numAristasARCM++;
        }
    } while (numAristasARCM < grafo.length-1);
    return arcm;
}
```

Algoritmo voraz

peor caso Θ(N³)
caso medio Θ(N²)

30

Algoritmo de Kruskal (ARCM) Grafo No dirigido (GN).

Implementación

```
int[][] copia(int[][] grafo){
    int[][] gcopia = new int[grafo.length][grafo.length];
    for (int i=0; i<grafo.length; i++)
        for (int j=i; j<grafo.length; j++)    gcopia[i][j]=grafo[i][j];
    return gcopia;
}

Arista seleccionarYEliminarAristaMenor(int[][] grafo){
    int menorPeso = Integer.MAX_VALUE;
    int o=0,d=0;
    // como es un GN basta con recorrer la diagonal superior
    for (int i=0; i<grafo.length; i++)
        for (int j=i; j<grafo.length; j++)
            if ((grafo[i][j]!=SIN_ARISTA)&&(grafo[i][j] < menorPeso)){
                menorPeso = grafo[i][j];    o = i;    d = j; // selecciona la menor
            }
    int peso = grafo[o][d];
    grafo[o][d] = SIN_ARISTA;// elimina arista de origen a destino
    grafo[d][o] = SIN_ARISTA;// elimina arista de destino a origen
    return new Arista(o,d,peso);
}
```


31

Algoritmo de Kruskal (ARCM) Grafo No dirigido (GN).

Implementación

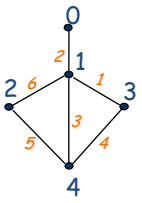
```
boolean existeCamino(int[][] grafo, int origen, int destino){
   boolean[] visitados = new boolean[grafo.length];
   for (int v=0;v<grafo.length;v++) visitados[v] = false;

   return existeCamino(grafo, origen, destino, visitados);
}

boolean existeCamino(int[][] grafo, int origen, int destino, boolean[] visitados){
   boolean existe = false;
   visitados[origen] = true;
   int ady = 0;
   while ((ady<grafo.length) && !existe){
      if (grafo[origen][ady] != SIN_ARISTA)
        if (ady != destino){
           if (!visitados[ady]) existe=existeCamino(grafo, ady, destino, visitados);
        }
        else existe = true;
      ady++;
   }
   return existe;
}
```


32

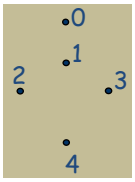
Algoritmo de Kruskal (ARCM) Grafo No dirigido (GN).

Seguimiento

	0	1	2	3	4
0	∞	2	∞	∞	∞
1	2	∞	6	1	3
2	∞	6	∞	∞	5
3	∞	1	∞	∞	4
4	∞	3	5	4	∞

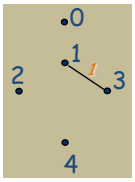


NumAristasARCM = 0 *ARCM* [ ] [ ] [ ] [ ]



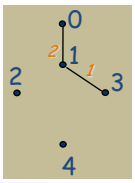
EscogerMenorArista(G, Arista) → *Arco: <1,3>*
NOT ExisteCamino(1,3) en ARCM

NumAristasARCM = 1 *ARCM* [<1,3>] [ ] [ ] [ ]



EscogerMenorArista(G, Arista) → *Arco: <0,1>*
NOT ExisteCamino(0,1) en ARCM

NumAristasARCM = 2 *ARCM* [<1,3>] [<0,1>] [ ] [ ]

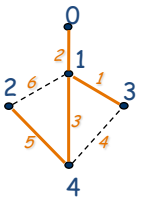


33

Algoritmo de Kruskal (ARCM) Grafo No dirigido (GN).

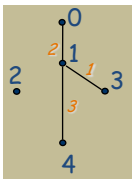
Seguimiento

	0	1	2	3	4
0	∞	2	∞	∞	∞
1	2	∞	6	1	3
2	∞	6	∞	∞	5
3	∞	1	∞	∞	4
4	∞	3	5	4	∞



EscogerMenorArista(G, Arista) → *Arco: <1,4>*
NOT ExisteCamino(1,4) en ARCM

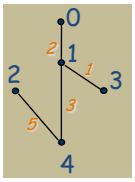
NumAristasARCM = 3 *ARCM* [<1,3>] [<0,1>] [<1,4>] [ ]



EscogerMenorArista(G, Arista) → *Arco: <3,4>*
ExisteCamino(3,4) en ARCM

EscogerMenorArista(G, Arista) → *Arco: <2,4>*
NOT ExisteCamino(2,4) en ARCM

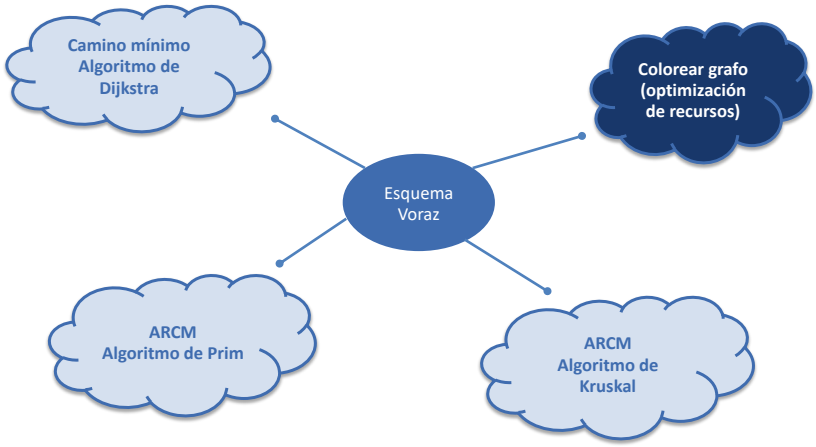
NumAristasARCM = 4 *ARCM* [<1,3>] [<0,1>] [<1,4>] [<2,4>]



34

Colorear un grafo (GN)

Algunos problemas planteados



35

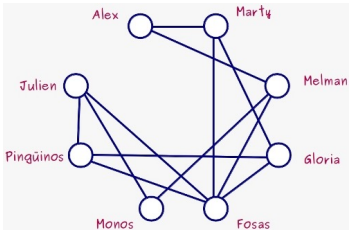
Colorear un grafo (GN)

Enunciado

- **Coloración de los vértices de un grafo**:
  - Ejemplos:
    - Optimización de recursos: queremos trasladar a los animales del zoo de Central Park al mundo salvaje. Para su traslado necesitamos utilizar jaulas (el menor número posible, porque cada jaula es muy cara y estamos en crisis). El problema reside en que no podemos transportar en la misma jaula a animales incompatibles (depredador y víctima).

      Cada animal se modela como un vértice del grafo y cada pareja de animales incompatibles como una arista entre los vértices que los representan (*fuente*: *Naukas.com; Clara Grima*).

ALEX	MARTY	MELMAN	GLORIA	FOSAS	MONOS	PINGÜINOS	JULIEN
MARTY	ALEX	ALEX	MARTY	MARTY	MELMAN	GLORIA	FOSAS
MELMAN	GLORIA	FOSAS	FOSAS	GLORIA	JULIEN	FOSAS	MONOS
	FOSAS	MONOS	PINGÜINOS	MELMAN		JULIEN	PINGÜINOS
				JULIEN			
				PINGÜINOS			



36

Colorear un grafo (GN)

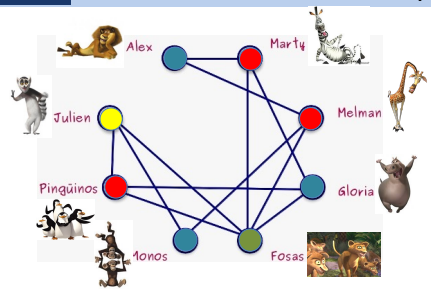
Implementación

```
int[] numeroCromaticoVoraz(int[][] grafo){
   int[] colores = new int[grafo.length];
   for (int i=0; i<grafo.length; i++)  colores[i] = -1; // SIN_COLOR
   int maxColor=0, c, vVecino;
   boolean incompatible, encontrado;
   for (int v=0; v<grafo.length; v++){ // selecciona vertice v
      c=0;   encontrado = false;
      while(c<=maxColor && !encontrado){
         vVecino=0; incompatible = false;
         while ((vVecino<grafo.length) && (!incompatible)) {
             if ((grafo[v][vVecino] != SIN_ARISTA) && colores[vVecino] == c)
                 incompatible = true;
             vVecino++;
         }
         if (incompatible) c++;
         else {  colores[v]=c;   encontrado=true;  }
      }
      if (!encontrado){   maxColor++;    colores[v] = maxColor;   }
    }
   return colores;
}
```

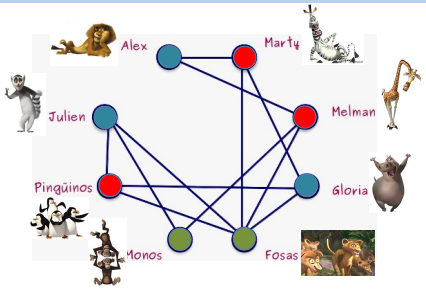

37

Colorear un grafo (GN)

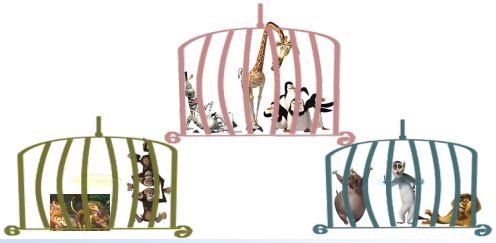
Implementación



Número cromático (voraces): 4 jaulas



Número cromático (selección óptima): 3 jaulas



38