



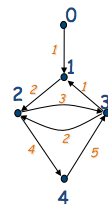
Tema 10. Backtracking en grafos

Algorítmica y Complejidad

1

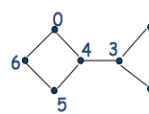
Backtracking y selección óptima en grafos

Grafos



Grafo Dirigido (GD)

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



Grafo NO dirigido (GN)

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

Recorrido (profundidad o amplitud): $\Theta(V)$

(Estructura de datos. Primer curso. 2º cuatrimestre)

- GD y GN ¿es conexo?
- GD y GN obtener un Árbol de Recubrimiento
- ¿Es un GN un árbol?
- Componente conexa con más vértices en GD y GN

Combinación de vértices y aristas del grafo que cumplen una condición:

- ARCM en GN (**voraz** -> Prim y Kruskal)
- Camino mínimo (Dijkstra -**voraz**- y Floyd -**dinámico**-)
- Si no hay más remedio que buscar
Backtracking o **Selección óptima**

2

Backtracking y selección óptima en grafos

Algunos problemas planteados

```
graph TD; A([Esquema backtracking]) --> B([Camino simple de coste exacto X]); A --> C([ARCM en GD]); A --> D([Colorear grafo]);
```

Camino coste X

1

2

3

3

3

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

```
graph TD; A([Esquema backtracking]) --> B([Camino simple de coste exacto X]); A --> C([ARCM en GD]); A --> D([Colorear grafo]);
```

Camino coste X

1

2

3

4

4

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

```

boolean caminoCosteExacto(int[][] grafo, int origen, int destino,
    double coste, Stack<Integer> camino) {

```

Vértice actual en el camino que se está construyendo (inicialmente origen)
 Coste restante del camino que se está construyendo (inicialmente coste total)
 Vértices visitados en el camino que se está construyendo

visitados: 0 1 2 3 4
 F F F F F

```

void caminoExactoBack(int[][] grafo, int origen, int destino, double coste, boolean[] visitados,
    Stack<Integer> camino, Booleano exito)

```

Pila de vértices ordenados del camino que se está construyendo (inicialmente con vértice origen)
 Candidatos: todos los vértices
 Aceptable: existe arista de origen a candidato, candidato no visitado y coste arista no supera coste restante

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

Camino coste X

1 — 2 — 3

7

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

```

boolean caminoCosteExacto(int[][] grafo, int origen, int destino,
    double coste, Stack<Integer> camino) {
    Booleano exito = new Booleano(false);
    boolean[] visitados = new boolean[grafo.length];
    for (int i = 0; i < grafo.length; i++) visitados[i] = false;
    visitados[origen] = origen != destino; // para permitir buscar caminos simples con origen=destino
    camino.push(origen);
    caminoExactoBack(grafo, origen, destino, coste, visitados, camino, exito);
    return exito.getValor();
}

void caminoExactoBack(int[][] grafo, int origen, int destino, double coste, boolean[] visitados,
    Stack<Integer> camino, Booleano exito)

```

Candidatos: todos los vértices
 Aceptable: existe arista de origen a candidato, candidato no visitado y coste arista no supera coste restante

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

Camino coste X

1 — 2 — 3

8

Camino simple de coste exacto X (GD y GN)

```

void caminoExactoBack(int[][] grafo, int origen, int destino, double coste,
    boolean[] visitados, Stack<Integer> camino, Booleano exito) {
    if ((coste == 0) && (origen==destino))
        exito.setValor(true);
    else {
        int v = 0;
        while (!exito.getValor() && (v != grafo.length)){
            if ((!visitados[v]) && (grafo[origen][v] != Integer.MAX_VALUE))
                if (grafo[origen][v] <= coste) {
                    visitados[v] = true;
                    coste = coste - grafo[origen][v];
                    camino.push(v);
                    caminoExactoBack(grafo, v, destino, coste, visitados, camino, exito);
                    if (!exito.getValor()) {
                        visitados[v] = false;
                        coste = coste + grafo[origen][v];
                        camino.pop();
                    }
                }
            v++;
        }
    }
}

```

Camino coste X



9

9

Camino simple de coste exacto X (GD y GN)

```

void caminoExactoBack(int[][] grafo, int origen, int destino, double coste,
    boolean[] visitados, Stack<Integer> camino, Booleano exito) {
    if ((coste == 0) && (origen==destino))
        exito.setValor(true);
    else {
        int v = 0;
        while (!exito.getValor() && (v != grafo.length)){
            if ((!visitados[v]) && (grafo[origen][v] != Integer.MAX_VALUE))
                if (grafo[origen][v] <= coste) {
                    visitados[v] = true;
                    coste = coste - grafo[origen][v];
                    camino.push(v);
                    caminoExactoBack(grafo, v, destino, coste, visitados, camino, exito);
                    if (!exito.getValor()) {
                        visitados[v] = false;
                        coste = coste + grafo[origen][v];
                        camino.pop();
                    }
                }
            v++;
        }
    }
}

```

Es solución?

Camino coste X



10

10

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

```

void caminoExactoBack(int[][] grafo, int origen, int destino, double coste,
    boolean[] visitados, Stack<Integer> camino, Booleano exito) {
    if ((coste == 0) && (origen==destino))
        exito.setValor(true);
    else {
        int v = 0;
        while (!exito.getValor() && (v != grafo.length)){
            if ((!visitados[v] && (grafo[origen][v] != Integer.MAX_VALUE))
                if (grafo[origen][v] <= coste) {
                    visitados[v] = true;
                    coste = coste - grafo[origen][v];
                    camino.push(v);
                    caminoExactoBack(grafo, v, destino, coste, visitados, camino, exito);
                    if (!exito.getValor()) {
                        visitados[v] = false;
                        coste = coste + grafo[origen][v];
                        camino.pop();
                    }
                }
            v++;
        }
    }
}

```

*Candidatos:
todos los vértices*

Camino coste X

11

11

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

```

void caminoExactoBack(int[][] grafo, int origen, int destino, double coste,
    boolean[] visitados, Stack<Integer> camino, Booleano exito) {
    if ((coste == 0) && (origen==destino))
        exito.setValor(true);
    else {
        int v = 0;
        while (!exito.getValor() && (v != grafo.length)){
            if ((!visitados[v] && (grafo[origen][v] != Integer.MAX_VALUE))
                if (grafo[origen][v] <= coste) {
                    visitados[v] = true;
                    coste = coste - grafo[origen][v];
                    camino.push(v);
                    caminoExactoBack(grafo, v, destino, coste, visitados, camino, exito);
                    if (!exito.getValor()) {
                        visitados[v] = false;
                        coste = coste + grafo[origen][v];
                        camino.pop();
                    }
                }
            v++;
        }
    }
}

```

Aceptable?

Camino coste X

12

12

Camino simple de coste exacto X (GD y GN)

```

void caminoExactoBack(int[][] grafo, int origen, int destino, double coste,
    boolean[] visitados, Stack<Integer> camino, Booleano exito) {
    if ((coste == 0) && (origen==destino))
        exito.setValor(true);
    else {
        int v = 0;
        while (!exito.getValor() && (v != grafo.length)){
            if ((!visitados[v] && (grafo[origen][v] != Integer.MAX_VALUE))
                if (grafo[origen][v] <= coste) {
                    visitados[v] = true;
                    coste = coste - grafo[origen][v];
                    camino.push(v);
                    caminoExactoBack(grafo, v, destino, coste, visitados, camino, exito);
                    if (!exito.getValor()) {
                        visitados[v] = false;
                        coste = coste + grafo[origen][v];
                        camino.pop();
                    }
                }
            v++;
        }
    }
}

```

Anotar

Camino coste X



13

13

Camino simple de coste exacto X (GD y GN)

```

void caminoExactoBack(int[][] grafo, int origen, int destino, double coste,
    boolean[] visitados, Stack<Integer> camino, Booleano exito) {
    if ((coste == 0) && (origen==destino))
        exito.setValor(true);
    else {
        int v = 0;
        while (!exito.getValor() && (v != grafo.length)){
            if ((!visitados[v] && (grafo[origen][v] != Integer.MAX_VALUE))
                if (grafo[origen][v] <= coste) {
                    visitados[v] = true;
                    coste = coste - grafo[origen][v];
                    camino.push(v);
                    caminoExactoBack(grafo, v, destino, coste, visitados, camino, exito);
                    if (!exito.getValor()) {
                        visitados[v] = false;
                        coste = coste + grafo[origen][v];
                        camino.pop();
                    }
                }
            v++;
        }
    }
}

```

Avance en el espacio de búsqueda

Camino coste X



14

14

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

```

void caminoExactoBack(int[][] grafo, int origen, int destino, double coste,
    boolean[] visitados, Stack<Integer> camino, Booleano exito) {
    if ((coste == 0) && (origen==destino))
        exito.setValor(true);
    else {
        int v = 0;
        while (!exito.getValor() && (v != grafo.length)){
            if ((!visitados[v]) && (grafo[origen][v] != Integer.MAX_VALUE))
                if (grafo[origen][v] <= coste) {
                    visitados[v] = true;
                    coste = coste - grafo[origen][v];
                    camino.push(v);
                    caminoExactoBack(grafo, v, destino, coste, visitados, camino, exito);
                    if (!exito.getValor()) {
                        visitados[v] = false;
                        coste = coste + grafo[origen][v];
                        camino.pop();
                    }
                }
            v++;
        }
    }
}

```

Camino coste X

Si no se ha encontrado solución: desanotar

15

15

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

camino

0

visitados

T F F F F

exito

false

Camino coste X

16

16

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

origen = 0
destino = 4
coste = 12

origen = 1
destino = 4
coste = 12

camino

1
0

visitados

0	1	2	3	4
T	T	F	F	F

exito

false

Camino coste X

1 — 2 — 3

17

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

origen = 0
destino = 4
coste = 12

origen = 1
destino = 4
coste = 12

camino

1
0

visitados

0	1	2	3	4
T	T	F	F	F

exito

false

Camino coste X

1 — 2 — 3

18

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

origen = 0
destino = 4
coste = 12

origen = 1
destino = 4
coste = 11

origen = 2
destino = 4
coste = 11

	0	1	2	3	4
0	T	T	T	F	F

camino

2
1
0

exitos

false

Camino coste X

1

2

3

19

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

origen = 0
destino = 4
coste = 12

origen = 1
destino = 4
coste = 11

origen = 2
destino = 4
coste = 11

	0	1	2	3	4
0	T	T	T	F	F

camino

2
1
0

exitos

false

Camino coste X

1

2

3

20

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

camino

3
2
1
0

visitados

0	1	2	3	4
T	T	T	T	F

exito

false

Camino coste X

1
2
3

21

21

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

camino

3
2
1
0

visitados

0	1	2	3	4
T	T	T	T	F

exito

false

Camino coste X

1
2
3

22

22

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

camino

4
3
2
1
0

visitados

0	1	2	3	4
T	T	T	T	T

exito

false

Camino coste X

1 2 3

23

23

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

camino

4
3
2
1
0

visitados

0	1	2	3	4
T	T	T	T	T

exito

false

Camino coste X

1 2 3

24

24

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

origen = 0

destino = 4

coste = 12

origen = 1

destino = 4

coste = 11

origen = 2

destino = 4

coste = 9

origen = 3

destino = 4

coste = 9

camino

3
2
1
0

visitados

0	1	2	3	4
T	T	T	T	F

exito

false

Camino coste X

1

2

3

25

25

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

origen = 0

destino = 4

coste = 12

origen = 1

destino = 4

coste = 11

origen = 2

destino = 4

coste = 11

camino

2
1
0

visitados

0	1	2	3	4
T	T	T	F	F

exito

false

Camino coste X

1

2

3

26

26

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

origen = 0

destino = 4

coste = 12

origen = 1

destino = 4

coste = 12

camino

1

0

visitados

	0	1	2	3	4
0	T	T	F	F	F

exito

false

Camino coste X

1

2

3

27

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

origen = 0

destino = 4

coste = 12

origen = 1

destino = 4

coste = 5

origen = 3

destino = 4

coste = 5

camino

3

1

0

visitados

	0	1	2	3	4
0	T	T	F	T	F

exito

false

Camino coste X

1

2

3

28

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

origen = 0
destino = 4
coste = 12

origen = 1
destino = 4
coste = 5

origen = 3
destino = 4
coste = 5

camino

3
1
0

visitados

0	1	2	3	4
T	T	F	T	F

exito

false

Camino coste X

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

origen = 0
destino = 4
coste = 12

origen = 1
destino = 4
coste = 5

origen = 3
destino = 4
coste = 3

origen = 2
destino = 4
coste = 3

camino

2
3
1
0

visitados

0	1	2	3	4
T	T	T	T	F

exito

false

Camino coste X

Backtracking y selección óptima en grafos

Camino simple de coste exacto X (GD y GN)

caminoCosteExacto (grafo, 0, 4, 13, camino)

caminoExactoBack(grafo, 0, 4, 13, visitados, camino, exito)

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

origen = 0
destino = 4
coste = 12

origen = 1
destino = 4
coste = 5

origen = 3
destino = 4
coste = 0

origen = 4
destino = 4
coste = 0

camino

4
3
1
0

visitados

0	1	2	3	4
T	T	F	T	T

exito

true

Camino coste X

1 — 2 — 3

33

33

Backtracking y selección óptima en grafos

Árbol de Recubrimiento de Coste Mínimo en GD valorado

Camino simple de coste exacto X

Esquema backtracking

ARCM en GD

Colorear grafo

ARCM en GD

1 — 2 — 3

34

34

Backtracking y selección óptima en grafos

Árbol de Recubrimiento de Coste Mínimo en GD valorado

Árbol:

- Un **árbol** es un GN conexo y acíclico (un **grafo acíclico** es aquel que no tiene ciclos)
- Un **árbol** es un GD conexo (no fuertemente) en el que un vértice tiene grado de entrada 0 (al que denominamos **raíz**) y todos los demás grado de entrada 1.

Árbol de recubrimiento:

- Sea $G = \langle V, A \rangle$ un grafo (GD o GN), y sea $G_A = \langle V', A' \rangle$ un árbol. Si $V = V'$ y $A' \subseteq A$ entonces decimos que G_A es un **árbol de recubrimiento** de G (spanning tree).

Coste del árbol:

- Sea $G = \langle V, A, f \rangle$ un árbol valorado. Denominamos **coste del árbol** a la suma de los costes de todas sus aristas o arcos.

Árbol de Recubrimiento de Coste Mínimo (ARCM):

- Sea G un grafo (GN o 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.

Propiedad MST (Minimum-cost Spanning Tree):

- Sea $G = \langle V, A, f \rangle$ un **GN conexo valorado** y sea U un subconjunto estricto de V ($U \subset V$, $U \neq V$). Si $a = (u, v) \in A$ es una arista tal que $u \in U$ y $v \in V - U$ 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 .

ARCM en GD

1

2

3

35

Backtracking y selección óptima en grafos

Árbol de Recubrimiento de Coste Mínimo en GD valorado

0 1 2 3 4

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

0 1 2 3 4

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

Tema 13. Voraces en grafos

Esquema voraz

Prim y Kruskal

ARCM

Selección óptima

ARCM en GD

1

2

3

36

18

Árbol de Recubrimiento de Coste Mínimo en GD valorado

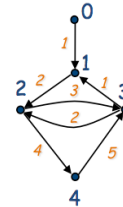
```

boolean esArbol(int[][] grafo){
    int gradoEnt0 = 0 , gradoEnt, v = 0;
    boolean ok = esGrafoConexo(grafo);
    while ((v < grafo.length) && ok){
        gradoEnt = gradoEntrada(grafo, v);
        ok = gradoEnt <= 1;
        if (gradoEnt==0) gradoEnt0++;
        v++;
    }
    return ((gradoEnt0 == 1) && ok);
}

int gradoEntrada(int[][] grafo, int vertice){
    int grado = 0;
    for (int ady=0; ady<grafo.length; ady++)
        if (grafo[ady][vertice]!=Integer.MAX_VALUE) grado++;
    return grado;
}

```

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



ARCM en GD



37

37

Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

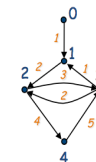
boolean esGrafoConexo(int[][] grafo){
    boolean[] visitados = new boolean[grafo.length];
    for (int i=0; i<grafo.length; i++) visitados[i]=false;
    recorrerCadena(grafo, 0, visitados);
    return todosVisitados(visitados);
}

void recorrerCadena(int[][] grafo, int origen, boolean[] visitados) {
    visitados[origen] = true;
    for (int ady = 0; ady < grafo.length; ady++)
        if (!visitados[ady])
            if (grafo[origen][ady]!=Integer.MAX_VALUE || grafo[ady][origen]!=Integer.MAX_VALUE)
                recorrerCadena(grafo, ady, visitados);
}

boolean todosVisitados(boolean[] visitados){
    boolean ok=true; int i=0;
    while ((i<visitados.length) && ok){ ok = visitados[i]; i++; }
    return ok;
}

```

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



ARCM en GD



38

38

Backtracking y selección óptima en grafos
Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

int[][] ARCM(int[][] grafo)

void ARCM_GD(int[][] grafo, int[][] subgrafo, int Naristas, int coste,
int[][] ARCM, Entero mejorCoste)

```

Grafo que se está construyendo (inicialmente con N° vértices = grafo original y sin aristas)

Número de aristas que tiene el subgrafo (inicialmente = 0)

Coste del subgrafo (inicialmente= 0)

Grafo que contiene el ARCM encontrado hasta el momento (inicialmente con N° vértices = grafo original y sin aristas)

Coste de ARCM (inicialmente= MAX_VALUE)

Candidatos: todas las aristas

Aceptable: existe arista en el grafo y no está incorporada en subgrafo

1

2

3

ARCM en GD

39

39

Backtracking y selección óptima en grafos
Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

int[][] ARCM(int[][] grafo){
    int[][] ARCM = new int[grafo.length][grafo.length];
    int[][] subgrafo = new int[grafo.length][grafo.length];
    for (int i=0; i<subgrafo.length; i++)
        for (int j=0; j<grafo.length; j++)
            subgrafo[i][j]=Integer.MAX_VALUE;
    Entero mejorCoste = new Entero(Integer.MAX_VALUE);
    int coste = 0;
    ARCM_GD(grafo, subgrafo, 0, coste, ARCM, mejorCoste);
    if (mejorCoste.getValor()!=Integer.MAX_VALUE) return ARCM;
    else return null;
}

void ARCM_GD(int[][] grafo, int[][] subgrafo, int Naristas, int coste,
int[][] ARCM, Entero mejorCoste)

```

Candidatos: todas las aristas

Aceptable: existe arista en el grafo y no está incorporada en subgrafo

1

2

3

ARCM en GD

40

40

Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

void ARCM_GD(int[][] grafo, int[][] subgrafo, int Naristas, int coste,
             int[][] ARCM, Entero mejorCoste){
    if (Naristas == grafo.length-1) {
        if (esArbol(subgrafo))
            if (mejorCoste.getValor() > coste) {
                copia(subgrafo, ARCM);
                mejorCoste.setValor(coste);
            }
    } else
        for (int i = 0; i < grafo.length; i++)
            for (int j = 0; j < grafo.length; j++)
                if ((grafo[i][j] != Integer.MAX_VALUE) && (subgrafo[i][j] == Integer.MAX_VALUE)){
                    subgrafo[i][j] = grafo[i][j];
                    coste = coste + grafo[i][j];
                    Naristas = Naristas + 1;
                    if (coste < mejorCoste.getValor())
                        ARCM_GD(grafo, subgrafo, Naristas, coste, ARCM, mejorCoste);
                    subgrafo[i][j] = Integer.MAX_VALUE;
                    coste = coste - grafo[i][j];
                    Naristas = Naristas - 1;
                }
    }
}

```

ARCM en GD



41

41

Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

void ARCM_GD(int[][] grafo, int[][] subgrafo, int Naristas, int coste,
             int[][] ARCM, Entero mejorCoste){
    if (Naristas == grafo.length-1) {
        if (esArbol(subgrafo)) Es solución
            if (mejorCoste.getValor() > coste) {
                copia(subgrafo, ARCM);
                mejorCoste.setValor(coste);
            }
    } else
        for (int i = 0; i < grafo.length; i++)
            for (int j = 0; j < grafo.length; j++)
                if ((grafo[i][j] != Integer.MAX_VALUE) && (subgrafo[i][j] == Integer.MAX_VALUE)){
                    subgrafo[i][j] = grafo[i][j];
                    coste = coste + grafo[i][j];
                    Naristas = Naristas + 1;
                    if (coste < mejorCoste.getValor())
                        ARCM_GD(grafo, subgrafo, Naristas, coste, ARCM, mejorCoste);
                    subgrafo[i][j] = Integer.MAX_VALUE;
                    coste = coste - grafo[i][j];
                    Naristas = Naristas - 1;
                }
    }
}

```

ARCM en GD



42

42

Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

void ARCM_GD(int[][] grafo, int[][] subgrafo, int Naristas, int coste,
             int[][] ARCM, Entero mejorCoste){
    if (Naristas == grafo.length-1) {
        if (esArbol(subgrafo))
            if (mejorCoste.getValor() > coste) {
                copia(subgrafo, ARCM);
                mejorCoste.setValor(coste);
            }
        }else
            for (int i = 0; i < grafo.length; i++)
                for (int j = 0; j < grafo.length; j++)
                    if ((grafo[i][j] != Integer.MAX_VALUE) && (subgrafo[i][j] == Integer.MAX_VALUE)){
                        subgrafo[i][j] = grafo[i][j];
                        coste = coste + grafo[i][j];
                        Naristas = Naristas + 1;
                        if (coste < mejorCoste.getValor())
                            ARCM_GD(grafo, subgrafo, Naristas, coste, ARCM, mejorCoste);
                        subgrafo[i][j] = Integer.MAX_VALUE;
                        coste = coste - grafo[i][j];
                        Naristas = Naristas - 1;
                    }
    }
}

```

*Es solución
mejor*

ARCM en GD



43

43

Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

void ARCM_GD(int[][] grafo, int[][] subgrafo, int Naristas, int coste,
             int[][] ARCM, Entero mejorCoste){
    if (Naristas == grafo.length-1) {
        if (esArbol(subgrafo))
            if (mejorCoste.getValor() > coste) {
                copia(subgrafo, ARCM);
                mejorCoste.setValor(coste);
            }
        }else
            for (int i = 0; i < grafo.length; i++)
                for (int j = 0; j < grafo.length; j++)
                    if ((grafo[i][j] != Integer.MAX_VALUE) && (subgrafo[i][j] == Integer.MAX_VALUE)){
                        subgrafo[i][j] = grafo[i][j];
                        coste = coste + grafo[i][j];
                        Naristas = Naristas + 1;
                        if (coste < mejorCoste.getValor())
                            ARCM_GD(grafo, subgrafo, Naristas, coste, ARCM, mejorCoste);
                        subgrafo[i][j] = Integer.MAX_VALUE;
                        coste = coste - grafo[i][j];
                        Naristas = Naristas - 1;
                    }
    }
}

```

*Candidatos: todas las
aristas del grafo*

ARCM en GD



44

44

Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

void ARCM_GD(int[][] grafo, int[][] subgrafo, int Naristas, int coste,
             int[][] ARCM, Entero mejorCoste){
    if (Naristas == grafo.length-1) {
        if (esArbol(subgrafo))
            if (mejorCoste.getValor() > coste) {
                copia(subgrafo, ARCM);
                mejorCoste.setValor(coste);
            }
    }else
        for (int i = 0; i < grafo.length; i++)
            for (int j = 0; j < grafo.length; j++)
                if ((grafo[i][j]!=Integer.MAX_VALUE)&&(subgrafo[i][j]==Integer.MAX_VALUE)){
                    subgrafo[i][j] = grafo[i][j];
                    coste = coste + grafo[i][j];
                    Naristas = Naristas + 1;
                    if (coste < mejorCoste.getValor())
                        ARCM_GD(grafo, subgrafo, Naristas, coste, ARCM, mejorCoste);
                    subgrafo[i][j] = Integer.MAX_VALUE;
                    coste = coste - grafo[i][j];
                    Naristas = Naristas - 1;
                }
    }
}

```

*Aceptable: existe la
arista en el grafo y no
está en subgrafo*

ARCM en GD



45

45

Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

void ARCM_GD(int[][] grafo, int[][] subgrafo, int Naristas, int coste,
             int[][] ARCM, Entero mejorCoste){
    if (Naristas == grafo.length-1) {
        if (esArbol(subgrafo))
            if (mejorCoste.getValor() > coste) {
                copia(subgrafo, ARCM);
                mejorCoste.setValor(coste);
            }
    }else
        for (int i = 0; i < grafo.length; i++)
            for (int j = 0; j < grafo.length; j++)
                if ((grafo[i][j]!=Integer.MAX_VALUE)&&(subgrafo[i][j]==Integer.MAX_VALUE)){
                    subgrafo[i][j] = grafo[i][j];
                    coste = coste + grafo[i][j];
                    Naristas = Naristas + 1;
                    if (coste < mejorCoste.getValor())
                        ARCM_GD(grafo, subgrafo, Naristas, coste, ARCM, mejorCoste);
                    subgrafo[i][j] = Integer.MAX_VALUE;
                    coste = coste - grafo[i][j];
                    Naristas = Naristas - 1;
                }
    }
}

```

Anotar

ARCM en GD



46

46

Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

void ARCM_GD(int[][] grafo, int[][] subgrafo, int Naristas, int coste,
             int[][] ARCM, Entero mejorCoste){
    if (Naristas == grafo.length-1) {
        if (esArbol(subgrafo))
            if (mejorCoste.getValor() > coste) {
                copia(subgrafo, ARCM);
                mejorCoste.setValor(coste);
            }
    }else
        for (int i = 0; i < grafo.length; i++)
            for (int j = 0; j < grafo.length; j++)
                if ((grafo[i][j]!=Integer.MAX_VALUE)&&(subgrafo[i][j]==Integer.MAX_VALUE)){
                    subgrafo[i][j] = grafo[i][j];
                    coste = coste + grafo[i][j];
                    Naristas = Naristas + 1;
                    Avance en el espacio de búsqueda (poda)
                    if (coste < mejorCoste.getValor())
                        ARCM_GD(grafo, subgrafo, Naristas, coste, ARCM, mejorCoste);
                    subgrafo[i][j] = Integer.MAX_VALUE;
                    coste = coste - grafo[i][j];
                    Naristas = Naristas - 1;
                }
    }
}

```

ARCM en GD



47

47

Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

void ARCM_GD(int[][] grafo, int[][] subgrafo, int Naristas, int coste,
             int[][] ARCM, Entero mejorCoste){
    if (Naristas == grafo.length-1) {
        if (esArbol(subgrafo))
            if (mejorCoste.getValor() > coste) {
                copia(subgrafo, ARCM);
                mejorCoste.setValor(coste);
            }
    }else
        for (int i = 0; i < grafo.length; i++)
            for (int j = 0; j < grafo.length; j++)
                if ((grafo[i][j]!=Integer.MAX_VALUE)&&(subgrafo[i][j]==Integer.MAX_VALUE)){
                    subgrafo[i][j] = grafo[i][j];
                    coste = coste + grafo[i][j];
                    Naristas = Naristas + 1;
                    if (coste < mejorCoste.getValor())
                        ARCM_GD(grafo, subgrafo, Naristas, coste, ARCM, mejorCoste);
                    subgrafo[i][j] = Integer.MAX_VALUE;
                    coste = coste - grafo[i][j];
                    Naristas = Naristas - 1;
                }
    }
}

```

ARCM en GD



48

48

Backtracking y selección óptima en grafos

Árbol de Recubrimiento de Coste Mínimo en GD valorado

ARCM_GD(grafo, subgrafo, 0, coste, ARCM, mejorCoste)

Diagram illustrating the ARCM_GD algorithm. The search tree shows nodes representing partial solutions (sets of edges). The root node is (0,1). The tree branches out to (0,1), (1,2), (2,3), and (2,4). Further branches lead to nodes like (0,1), (1,2), (2,3), (2,4), (3,1), (3,2), and (4,3). Some branches are pruned (marked with blue X's). The bottom row shows small graphs with their costs and whether they are trees. One graph is highlighted with a red border and labeled 'Igual a mejorCoste'.

ARCM en GD

49

49

Backtracking y selección óptima en grafos

Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

int[][] ARCM_optimizado(int[][] grafo){
    int[][] ARCM = new int[grafo.length][grafo.length];
    int[][] subgrafo = new int[grafo.length][grafo.length];
    for (int i=0; i<subgrafo.length; i++)
        for (int j=0; j<grafo.length; j++)
            subgrafo[i][j]=Integer.MAX_VALUE;
    Entero mejorCoste = new Entero(Integer.MAX_VALUE);
    int coste = 0;
    ARCM_GD_optimizado(grafo, subgrafo, 0, 0, 0, coste, ARCM, mejorCoste);
    if (mejorCoste.getValor()!=Integer.MAX_VALUE) return ARCM;
    else return null;
}

```

ARCM en GD

50

50

Árbol de Recubrimiento de Coste Mínimo en GD valorado

```

void ARCM_GD_optimizado(int[][] grafo, int[][] subgrafo, int Naristas, int coste,
                        int x, int y, int[][] ARCM, Entero mejorCoste){
    if (Naristas == grafo.length-1) {
        if (esArbol(subgrafo))
            if (mejorCoste.getValor() > coste) { copia(subgrafo, ARCM); mejorCoste.setValor(coste); }
        } else
            for (int i = x; i < grafo.length; i++)
                for (int j = y; j < grafo.length; j++)
                    if ((grafo[i][j] != Integer.MAX_VALUE)) {
                        subgrafo[i][j] = grafo[i][j];
                        coste = coste + grafo[i][j];
                        Naristas = Naristas + 1;
                        if (coste < mejorCoste.getValor()) {
                            int nx=i, ny=j+1;
                            if (ny==grafo.length){ ny=0; nx++; }
                            ARCM_GD_optimizado(grafo, subgrafo, Naristas, coste, nx, ny, ARCM, mejorCoste);
                        }
                        subgrafo[i][j] = Integer.MAX_VALUE;
                        coste = coste - grafo[i][j];
                        Naristas = Naristas-1;
                    }
    }
}

```

ARCM en GD

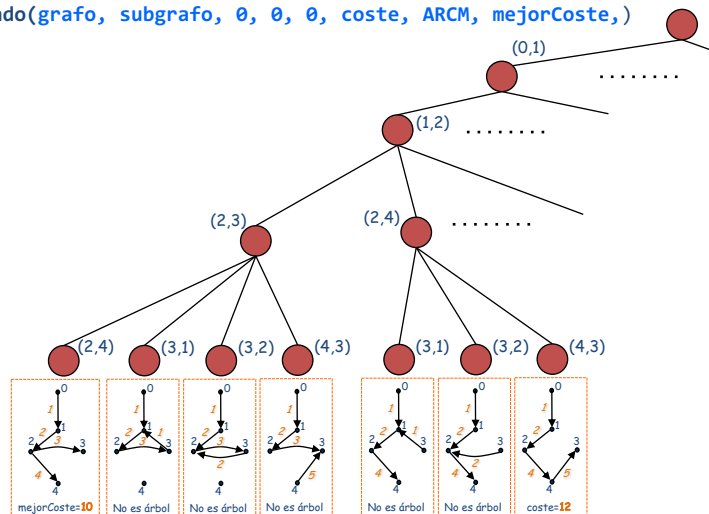
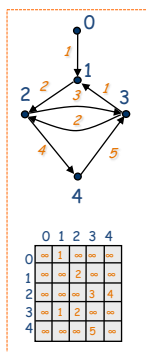


51

51

Árbol de Recubrimiento de Coste Mínimo en GD valorado

ARCM_GD_optimizado(grafo, subgrafo, 0, 0, 0, coste, ARCM, mejorCoste,)

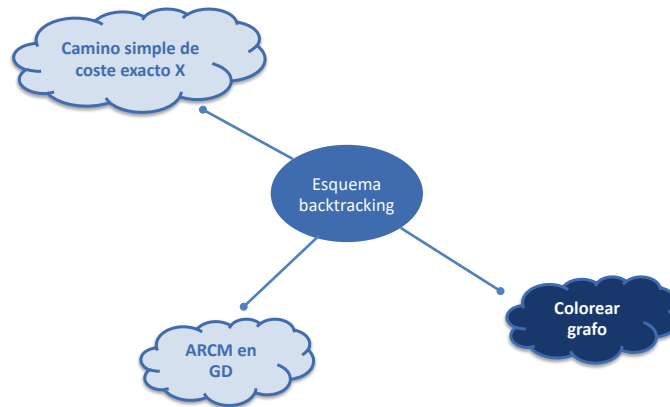


ARCM en GD



52

52



53

53

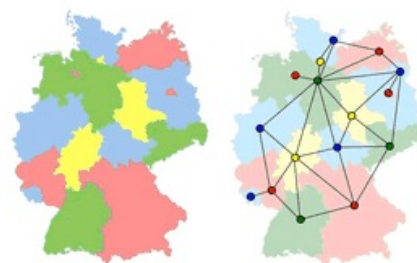
• Coloración de los vértices de un grafo:

- Caso especial de etiquetado de grafos donde la etiqueta es un color.
- Objetivo: asignar un color a cada vértice del grafo, de forma que ningún vértice adyacente comparta el mismo color, utilizando el mínimo número de colores posibles (número cromático).

– Ejemplos:

- Dado un mapa, ¿pueden pintarse sus regiones de tal forma que no haya dos regiones adyacentes de igual color y se emplee el menor número de colores posible?

Cada región se modela como un vértice y si dos regiones son adyacentes sus correspondientes vértices estarán conectados con una arista.



54


54

Colorear un grafo (GN)

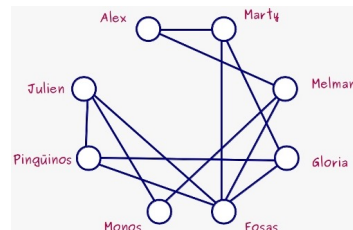
• Coloración de los vértices de un grafo:– Ejemplos:

- 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 (*fuelle: Clara Grima; <http://naukas.com/2012/05/23/por-que-solo-cuatro-colores/>*).



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	JULIEN	PINGÜINOS
				JULIEN			
				PINGÜINOS			



Colorear grafo



55

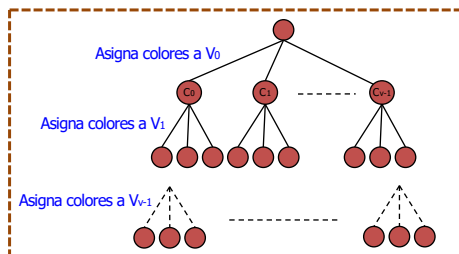
55

Colorear un grafo (GN)

Selección óptima

Número Cromático

Tema 13. Voraces en grafos

Esquema voraz
(solución óptima?)

colores

0	1	2	3	4	v-1
-1	-1	-1	-1	-1	-1

mejorColores

0	1	2	3	4	v-1
0	1	2	3	4	v-1

```
void numeroCromatico_SelOp(int[][] grafo, int v, int[] colores, int[] mejorColores)
```

Colorear grafo



56

56

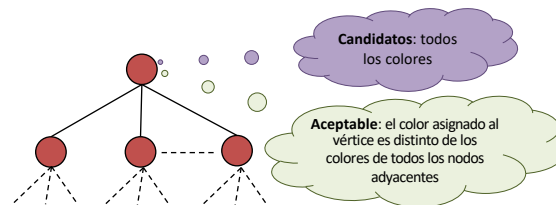
Colorear un grafo (GN)

```

int[] numeroCromatico(int[][] grafo){
    int[] colores = new int[grafo.length];
    int[] mejorColores = new int[grafo.length];
    for (int i=0; i<grafo.length; i++){
        colores[i] = -1; // SIN_COLOR
        mejorColores[i] = i; // Inicializa al peor valor (V colores distintos)
    }
    numeroCromatico_SelOp(grafo, 0, colores, mejorColores);
    return mejorColores;
}

void numeroCromatico_SelOp(int[][] grafo, int v, int[] colores, int[] mejorColores)

```



57

57

Colorear un grafo (GN)

```

void numeroCromatico_SelOp(int[][] grafo, int v, int[] colores, int[] mejorColores){
    if (v == grafo.length) {
        if (coloresDistintos(mejorColores) > coloresDistintos(colores)) {
            for (int i = 0; i < colores.length; i++)
                mejorColores[i] = colores[i];
        }
    } else
        for (int c=0; c<grafo.length; c++)
            if (aceptable(grafo, v, c, colores)) {
                colores[v] = c;
                v=v+1;
                numeroCromatico_SelOp(grafo, v, colores, mejorColores);
                v=v-1;
                colores[v]=-1;
            }
}

```



58

58

Colorear un grafo (GN)

```

boolean acceptable(int[][] grafo, int v, int c, int[] colores){
    boolean ok = true;
    int j=0;
    while ((j<v) && ok){
        if (grafo[v][j] != Integer.MAX_VALUE)    ok = (colores[j] != c);
        j++;
    }
    return ok;
}

int coloresDistintos(int[] colores){
    int c=0;
    boolean[] hayColor = new boolean[colores.length];
    for (int i=0; i< colores.length; i++)    hayColor[i] = false;
    for (int i=0; i<colores.length; i++)    hayColor[colores[i]] = true;
    for (int j=0; j<colores.length; j++)
        if (hayColor[j]) c++;
    return c;
}

```

Colorear grafo

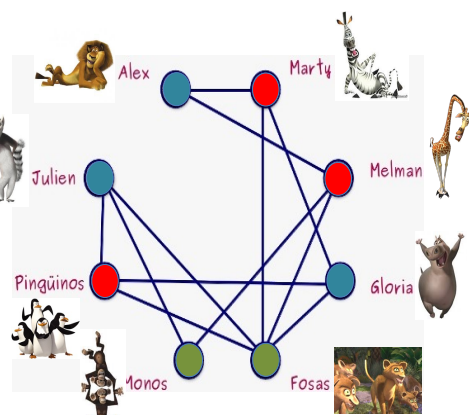


59

59

Colorear un grafo (GN)

Número cromático: 3 jaulas



Colorear grafo



60

60