

**Ejercicio 1.** Modificar el algoritmo DFS para que determine si el grafo de entrada es o no es conexo.

$\text{DFS}(G):$ <ol style="list-style-type: none"> <li>1: for <math>v \in V(G):</math></li> <li>2:   visitado(<math>v</math>) = false</li> <li>3: for <math>v \in V(G):</math></li> <li>4:   if not visitado(<math>v</math>):</li> <li>5:     EXPLORAR(<math>G, v</math>)</li> </ol>	}	$\text{DFS-IS-CONNECTED}(G):$ <ol style="list-style-type: none"> <li>1: for <math>v \in V(G)</math></li> <li>2: visitado(<math>v</math>) = false</li> <li>3: flag = 0</li> <li>4: for <math>v \in V(G):</math></li> <li>5: if not visited(<math>v</math>):</li> <li>6:   EXPLORAR(<math>G, v</math>)</li> <li>7:   flag += 1</li> <li>8: return flag == 1</li> </ol>
---	---	--

**Ejercicio 2.** Modificar el algoritmo DFS para que determine la cantidad de componentes del grafo.

 $\text{DFS-IS-CONNECTED}(G):$ 

- 1: for  $v \in V(G)$
- 2: visitado( $v$ ) = false
- 3: count = 0
- 4: for  $v \in V(G):$
- 5: if not visited( $v$ ):
- 6:   EXPLORAR( $G, v$ )
- 7:   flag += 1
- 8: return count

**Ejercicio 3.** Modificar el algoritmo DFS para que llene un arreglo *componente*, indexado por los vértices del grafo, tal que  $\text{componente}[v] = \text{componente}[u]$  si  $u$  y  $v$  están en el mismo componente de  $G$ .

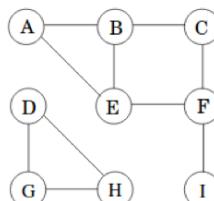
 $\text{DFS-COMPONENTS}(G)$ 

- 1: for  $v \in V(G)$
- 2: visitado( $v$ ) = false
- 3: component = 0
- 4: for  $v \in V(G):$
- 5: if not visited( $v$ ):
- 6:   EXPLORAR( $G, v, \text{component}$ )

 $\text{EXPLORAR-COMPONENTS}(G, v, \text{component})$ 

- 1: visitado( $v$ ) = true
- 2: components[v] = component
- 3: for  $uv \in E(G)$
- 4: if not visitado( $u$ )
- 5:   EXPLORAR-COMPONENTS( $G, u, \text{component}$ )

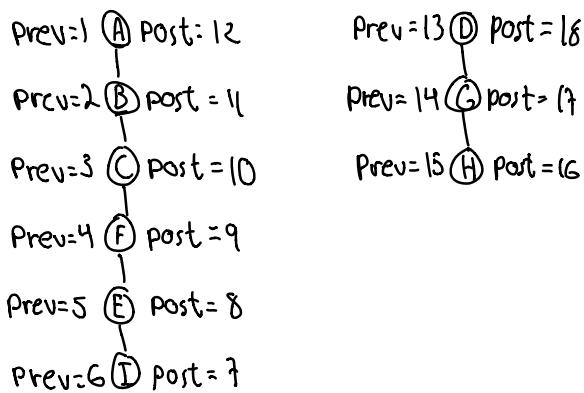
**Ejercicio 4.** Ejecute paso a paso el algoritmo DFS en el siguiente grafo. Siempre que haya una elección de vértices debe elegir aquel vértice que es alfabéticamente menor. Debe indicar la posición pre y post de cada vértice al finalizar el algoritmo como fue visto en clase.



```

EXPLORAR( $G, v$ )
1: visitado( $v$ ) = true
2: Pre[ $v$ ] = clock
3: clock = clock + 1
4: for  $w \in V(G)$ :
5:   if not visitado( $w$ )
6:     EXPLORAR( $G, w$ )
7:   Post[ $w$ ] = clock
8:   clock = clock + 1

```



**Ejercicio 5.** Modificar el algoritmo BFS para que, reciba un grafo  $G$  y dos vértices  $s$  y  $t$ , y devuelva un camino mínimo entre  $s$  y  $t$  si existe. Si no existe camino debe mostrar un mensaje.

```

BFS( $G, s$ )
1 for  $v \in V(G, s, t)$ 
2 dist( $v$ ) =  $\infty$ 
3 dist( $s$ ) = 0
4 Prev( $s$ ) =  $\emptyset$ 
5 Q = { $s$ }
6 while  $Q \neq \emptyset$ 
7    $u = \text{DESENCOLAR}(Q)$ 
8   for  $uv \in E(G)$ 
9     if dist( $v$ ) =  $\infty$ 
10    dist( $v$ ) = dist( $u$ ) + 1
11    Prev( $v$ ) =  $u$ 
12    if  $v = t$ 
13      return GET-PATH( $v$ )
14      ENCOLAR(Q,  $v$ )
15 return "There is no path between  $s$  and  $t$ "

```

```

GET-PATH( $v$ )
1 let  $S$  be a stack
2 while  $v \neq \emptyset$ 
3   PUSH( $S, v$ )
4    $v = \text{prev}(v)$ 
5 return  $S$ 

```

**Ejercicio 6.** Modificar el algoritmo BFS para que, reciba un grafo conexo  $G$  y verifique si el grafo es bicolorable. Un grafo es *bicolorable* si existe una asignación de dos colores a los vértices del grafo tal que dos vértices adyacentes no tienen el mismo color.

```

BFS-COLOR( $G, s, c$ )
1 for  $v \in V(G)$ 
2 dist( $v$ ) =  $\infty$ 
3 dist( $s$ ) = 0
4 col( $s$ ) =  $c$ 
5 Q = { $s$ }
6 while  $Q \neq \emptyset$ 
7    $u = \text{DESENCOLAR}(Q)$ 
8    $c = 1 - c$ 
9   for  $uv \in E(G)$ 
10    if dist( $v$ ) =  $\infty$ 
11      col( $v$ ) =  $c$ 

```

```

12     dist(v)=dist(u)+1
13     else if col(u)=col(v)
14         return false
15 return true

```

**Ejercicio 7.** El *diámetro* de un grafo simple (no dirigido y sin pesos en las aristas) es la distancia máxima entre cualquier par de vértices. Utilizando BFS, encuentre un algoritmo con complejidad  $O(n(m + n))$  para encontrar el diámetro en un grafo con  $n$  vértices y  $m$  aristas.

```

BFS-DIÁMETRO(G)
1 diámetro=-∞
2 for s ∈ V(G)
3     for v ∈ V(G)
4         dist(v)=∞
5     dist(s)=0
6     Q={s}
7     while Q ≠ ∅
8         u=DESENCOLAR(Q)
9         for uv ∈ E(G)
10            if dist(v)=∞
11                dist(v)=dist(u)+1
12                ENCOLAR(Q,v)
13            if dist(v)>diámetro
14                diámetro=dist(v)
15 return diámetro

```

**Ejercicio 8.** Podemos representar un campo minado por una matriz de números enteros, donde el carácter 0 representa un lugar sin mina, el carácter 1 representa un lugar minado y el carácter 2 representa un lugar al cual no se puede acceder.

- (a) Suponga que tenemos un campo minado con una única mina. Deseamos saber cuales son las posibilidades de poder toparse con dicha mina al moverse a partir de una posición (los movimientos no pueden ser diagonales) Haga un algoritmo que calcule cual es el menor número de pasos que una persona debe moverse para toparse con dicha mina.

Por ejemplo, si la matriz es

$$\begin{bmatrix} 0 & 2 & 0 \\ 0 & 2 & 2 \\ 0 & 1 & 0 \end{bmatrix}$$

su algoritmo deberá devolver una matriz

$$\begin{bmatrix} 3 & -1 & \infty \\ 2 & -1 & -1 \\ 1 & 0 & 1 \end{bmatrix}$$

Siendo la matriz  $n \times n$ , su algoritmo deberá consumir tiempo  $O(n^2)$ .

- (b) Suponga que tenemos un campo minado con **varias** minas. Deseamos saber cuales son las posibilidades de poder toparse con alguna de esas minas al moverse a partir de una posición (los movimientos no pueden ser diagonales) Haga un algoritmo que calcule cual es el menor número de pasos que una persona debe moverse para toparse con **alguna** mina.

Por ejemplo, si la matriz es

$$\begin{bmatrix} 0 & 2 & 0 & 1 \\ 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

su algoritmo deberá devolver una matriz

$$\begin{bmatrix} \infty & -1 & 1 & 0 \\ -1 & -1 & 2 & 1 \\ 2 & 1 & 2 & 2 \\ 1 & 0 & 1 & 2 \end{bmatrix}$$

Siendo la matriz  $n \times n$ , su algoritmo **deberá consumir tiempo**  $O(n^2)$ .

(a) ALGORITMO ( $M$ )  
 1  $n = M.\text{rows}$   
 2 for  $i = 1$  to  $n$

  |  $\text{fill\_Answer}(M, i, j)$  ASUMIR QUE  $A$  ES UNA MATRIZ  $n \times n$  CON VALORES  $\infty$   
 |  $Q = \{(i, j)\}$   
 | 2  $\text{currentDist} = 1$

```

3   for j=1 to n
4     if M[i][j]=1
5       Fin-Answer(M,i,j)
6       return
7
8   while Q ≠ φ
9     (v,l)=DESENCOLAR(Q)
10    if Is-Inside-Bounds(k,l+1)
11      if M[k,l+1]=0
12        ENCOLAR(Q,(k,l+1))
13        A[k,l+1]=currentDist
14      else if M[k,l+1]=2
15        A[k,l+1]=-1
16
17    // HACER LO MISMO CON LOS OREOS LADOS
18    currentDist+=1
19
20  return A

```

5) ALGORITMO(M)

```

1  for i=1 to n
2    for j=1 to n
3      if M[i,j]=1
4        DFS(M,i,j,0)
5      else if M[i,j]=2
6        A[i,j]=-1
7
8  return A

```

DFS(M,i,j,val) || Asumir que A tiene sus valores seteados a  $\infty$

```

1  if Inside-Bounds(i,j+1) and M[i,j]≠{1,2}
2    A[i,j+1]=min{A[i,j+1],val+1}
3
4 // REPETIR LO MISMO PARA LOS OREOS LADOS

```

**Ejercicio 9.** Dado un árbol  $T$  (grafo conexo sin circuitos, no dirigido y sin pesos en las aristas), un vértice  $s$  y un conjunto  $R \subseteq V(T)$ , decimos que una hoja  $v$  (vértice de grado uno diferente de  $s$ ) de  $T$  es  $k$ -atingible desde  $s$ , si el (único) camino desde  $s$  hacia  $v$  contiene como máximo  $k$  vértices en  $R$ . Diseñe un algoritmo eficiente ( $O(|V(G)|)$ ) que recibe un árbol  $G$ , un vértice  $s$  en  $G$ , un conjunto  $R \subseteq V(G)$ , un entero  $k$  y encuentra todas las hojas  $k$ -atingibles desde  $s$  en  $G$ .

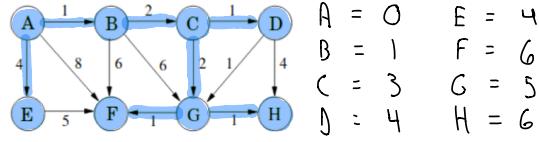
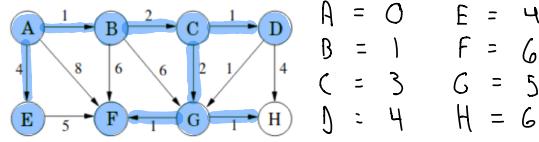
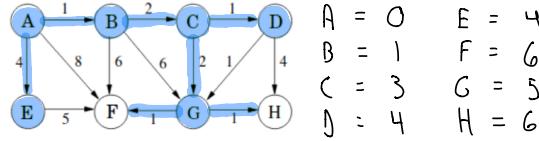
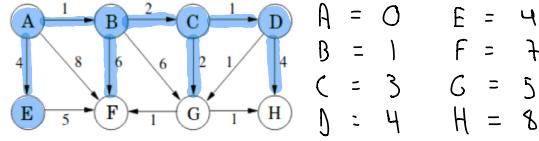
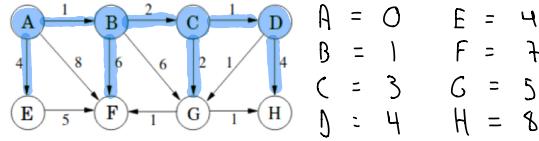
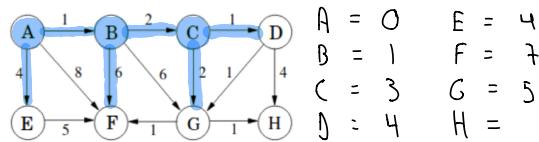
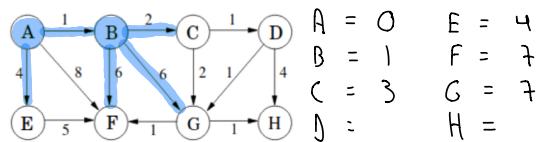
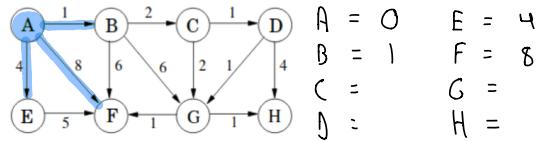
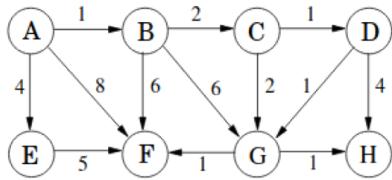
ALGORITMO( $G, s, R, k$ )

```

1  for v ∈ V(G)
2    dist(v)=∞
3  dist(s)=0
4  Q={s}
5  A={}
6  while Q ≠ φ
7    u=DESENCOLAR(Q)
8    for uv ∈ E(G)
9      if dist(v)=∞
10        if v ∈ R
11          dist(v)=dist(u)+1
12        else
13          dist(v)=dist(u)
14        if dist(v)≤k
15          if v is leaf
16            A=A ∪ {v}
17        else
18          ENQUEUE(Q,u)

```

**Ejercicio 10.** Muestre el valor de dist en cada iteración del algoritmo de Dijkstra, a partir del nodo A, y suponiendo que los vecinos de un vértice son procesados en orden alfabético



**Ejercicio 11.** Suponga que cambiamos la línea 4 del algoritmo de Dijkstra (implementación con fila de prioridades) como sigue:

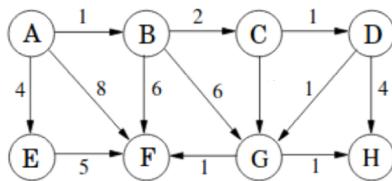
```
while  $|Q| > 1$ 
```

Eso causa que la condición se ejecute  $|V(G)| - 1$  veces en lugar de  $|V(G)|$  veces. ¿El algoritmo continúa correcto? Justifique.

```

DIJKSTRA( $G, s$ )
1 for  $v \in V(G)$ 
2    $dist(v) = \infty$ 
3  $dist(s) = 0$ 
4  $Q = V(G)$ 
5 while  $Q \neq \emptyset$ 
6    $u = DESARROLLAR(Q)$ 
7   for  $uv \in E(G)$ 
8     if  $dist(u) + l_{uv} < dist(v)$ 
9        $dist(v) = dist(u) + l_{uv}$ 
```

SEAN  $v$  EL VÉRTICE QUE QUEDA CUANDO  $|Q|=1$ . EL ALGORITMO DE DIJKSTRA EN LAS ITERACIONES ANTERIORES HA ENCONTRADO LA DISTANCIA MÁS CORTA DESDE  $s$  HASTA TODOS LOS VÉRTICES  $V(G)$ . POR TANTO, EL ALGORITMO SIGUE SIENDO CORRECTO.



**Ejercicio 12.** Diseñe (haga un pseudocódigo) de un algoritmo que hace lo siguiente:

Recibe: Un grafo no dirigido  $G$  con longitudes no negativas  $\ell$  en las aristas y una arista  $e \in E(G)$

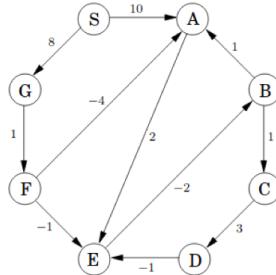
Devuelve: La longitud de un circuito (ciclo sin repeticiones de vértices ni aristas) mínimo que contiene  $e$

Su algoritmo deberá consumir tiempo  $O(|V(G)|^2)$ .

ALGORITMO( $G, e$ )

**Ejercicio 13.** Muestre el valor de dist en cada iteración del algoritmo Bellman-Ford, suponiendo que se revisan las aristas en el siguiente orden:

$AE, BA, BC, CD, DE, EB, FA, FE, GF, SA, SG$

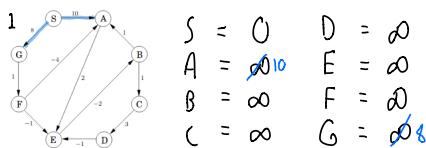


BELLMAN-FORD ( $G, s$ )

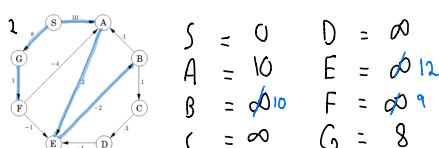
```

1 for  $v \in V(G)$ 
2    $dist(v) = \infty$ 
3    $dist(s) = 0$ 
4 for  $i = 1$  to  $|V(G)|$ 
5   for  $uv \in E(G)$ 
6      $dist(v) = \min\{dist(v), dist(u) + l_{uv}\}$ 

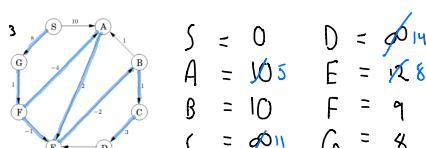
```



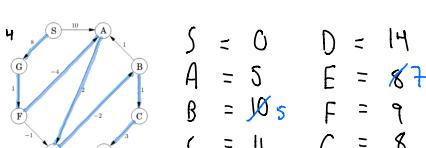
$S = 0 \quad D = \infty$   
 $A = 10 \quad E = \infty$   
 $B = \infty \quad F = \infty$   
 $C = \infty \quad G = 8$



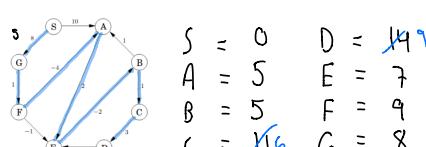
$S = 0 \quad D = \infty$   
 $A = 10 \quad E = 12$   
 $B = 10 \quad F = 9$   
 $C = \infty \quad G = 8$



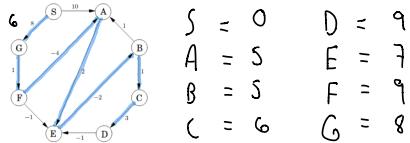
$S = 0 \quad D = 14$   
 $A = 10 \quad E = 12$   
 $B = 10 \quad F = 9$   
 $C = 11 \quad G = 8$



$S = 0 \quad D = 14$   
 $A = 5 \quad E = 7$   
 $B = 5 \quad F = 9$   
 $C = 11 \quad G = 8$



$S = 0 \quad D = 14$   
 $A = 5 \quad E = 7$   
 $B = 5 \quad F = 9$   
 $C = 11 \quad G = 8$



$$\begin{array}{ll} S = 0 & D = 9 \\ A = 5 & E = 7 \\ B = 5 & F = 9 \\ C = 6 & G = 8 \end{array}$$

~~AE, BA, BC, CD, DE, EB, FA, FE, GF, SA, SG~~

**Ejercicio 14.** Podemos modificar el algoritmo de Bellman-Ford para que termine en menos iteraciones si detecta que no han habido modificaciones en las distancias.

(a) Diseñe (haga pseudocódigo) que incluya esa modificación

(b) ¿El número de iteraciones, luego de incluir la modificación sugerida en (a), podría cambiar para una misma entrada si es que el orden de revisión de las aristas cambia?

(a) BELLMAN-FORD( $G, s$ )

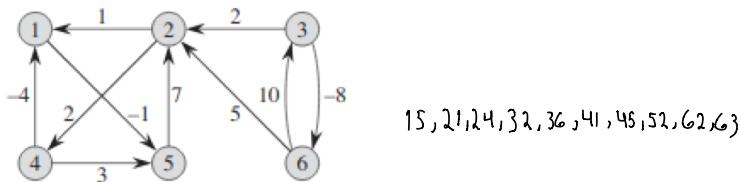
```

1 for  $v \in V(G)$ 
2    $dist(v) = \infty$ 
3    $dist(s) = 0$ 
4 for  $i=1$  to  $|V(G)|-1$ 
5   flag = false
6   for  $uv \in E(G)$ 
7     if  $dist(u) + l_{uv} < dist(v)$ 
8       flag = true
9        $dist(v) = dist(u) + l_{uv}$ 
10  if not flag
11    break

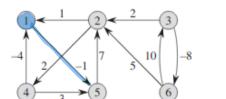
```

(b) PODRÍA AUMENTAR, DISMINUIR O MANTENERSE IGUAL.

**Ejercicio 15.** Ejecute los algoritmos vistos en clase para calcular caminos mínimos entre todos los pares en el siguiente grafo dirigido. Muestre los resultados parciales (iteración por iteración).

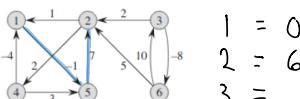


Dijkstra

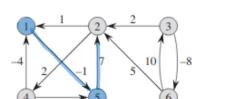


$$\begin{array}{ll} 1 = 0 & 4 = \\ 2 = & 5 = -1 \\ 3 = & 6 = \end{array}$$

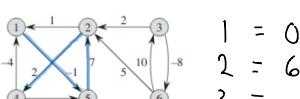
BELLMAN-FORD PARA ①



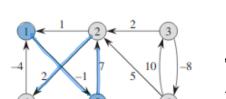
$$\begin{array}{ll} 1 = 0 & 4 = \\ 2 = 6 & 5 = -1 \\ 3 = & 6 = \end{array}$$



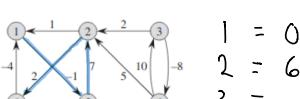
$$\begin{array}{ll} 1 = 0 & 4 = \\ 2 = 6 & 5 = -1 \\ 3 = & 6 = \end{array}$$



$$\begin{array}{ll} 1 = 0 & 4 = 8 \\ 2 = 6 & 5 = -1 \\ 3 = & 6 = \end{array}$$



$$\begin{array}{ll} 1 = 0 & 4 = 8 \\ 2 = 6 & 5 = -1 \\ 3 = & 6 = \end{array}$$



$$\begin{array}{ll} 1 = 0 & 4 = 8 \\ 2 = 6 & 5 = -1 \\ 3 = & 6 = \end{array}$$

15, 21, 24, 31, 36, 41, 45, 52, 62, 63

## PESOS NEGATIVOS

**Ejercicio 16.** Indique como modificar los algoritmos vistos en clase para calcular caminos mínimos entre todos los pares, para que también devuelvan una matriz de predecesores. En esta matriz, la posición  $(i, j)$  guarda el predecesor de  $j$  en un camino mínimo de  $i$  hacia  $j$ .

<pre> 1 DISTANCE-BETWEEN-EVERY-PAIR(G) 2   for v ∈ V(G) 3     Dijkstra(G, s) 4     Bellman-Ford(G, s)       </pre>	<pre> 1 DISTANCE-BETWEEN-EVERY-PAIR(G, s) 2   for v ∈ V(G) 3     dist(v) = ∞ 4   dist(s) = 0 5   MATRIZ Dijkstra[s, s] = ∅ 6   while Q ≠ ∅ 7     u = DESENCOLAR(Q) 8     for uv ∈ E(G) 9       if dist(u) + luv &lt; dist(v) 10      dist(v) = dist(u) + luv 11      MATRIZ Dijkstra[s, v] = u 12      DECREASE-KEY(Q, v)       </pre>	<pre> Bellman-Ford(G, s) 1 for v ∈ V(G) 2 dist(v) = ∞ 3 dist(s) = 0 4 for i=1 to  V(G)  5   for uv ∈ E(G) 6     if dist(u) + luv &lt; dist(v) 7       dist(v) = dist(u) + luv 8       MATRIZ BF[s, v] = u       </pre>
--	--	--

```

EXPLORAR(V):
1: visitado(v)=true
2: for uv ∈ E(G)
3:   if not visitado(u)
4:     EXPLORAR(u)

```

```

DFS(G)
1: for v ∈ V(G)
2:   visitado(v)=false
3:   for v ∈ V(G)
4:     if not visitado(v)
5:       EXPLORAR(G,v)

```

DFS-Bicolor(G)

```

1 for v ∈ V(G)
2 visitado(v)=true
3 for v ∈ V(G)
4   if not visitado(v)
5     if not COLOREAR(G,v,0)
6       return falso
7 return Verdadero

```

COLOREAR(G, v, c)

```

1 col(v)=c
2 for uv ∈ E(G)
3   if not visitado(u)
4     if not COLOREAR(G,u,1-c)
5       return false
6   else if col(u)=col(v)
7     return false
8 return true

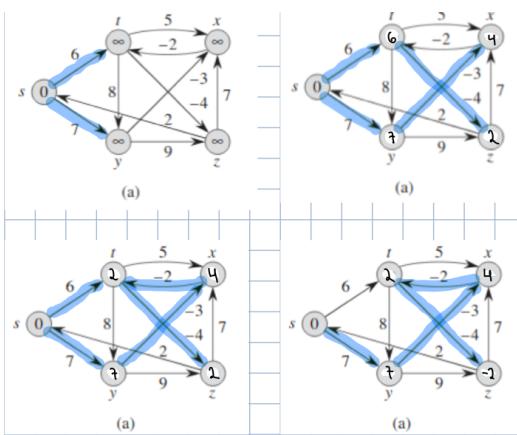
```

Dijkstra-FP(G,s)

```

1 for v ∈ V(G)
2 dist(v)=∞
3 dist(s)=0
4 Q=V(G)
5 while Q≠∅
6   v=EXTRACT-MIN(Q)
7   for vz ∈ E(G)
8     if dist(v)+lvz<dist(z)
9       dist(z)=dist(v)+lvz
10      DECREASE-KEY(Q,z)

```



```

BFS(G,s)
1: for v ∈ V(G)
2:   dist(v)=∞
3: dist(s)=0
4: Q={s}
5: while Q≠∅
6:   u = DESENCOLAR(Q)
7:   for uv ∈ E(G)
8:     if dist(v)=∞
9:       Encolar(Q,v)
10:      dist(v)=dist(u)+1

```

DISTANCIAS(G,s)

```

1 for v ∈ V(G)
2 dist(v)=∞
3 dist(s)=0
4 R=∅
5 while R≠V(G)
6   Sea v ∈ V(G)\R tal que
7     dist(v)=min{dist(w): w ∈ V(G)\R}
8   R=R ∪ {v}
9   for vz ∈ E(G)
10    if dist(v)+lvz<dist(z)
11      dist(z)=dist(v)+lvz

```

BELLMAN-FORD(G,s)

```

1 for v ∈ V(G)
2 dist(v)=∞
3 dist(s)=0
4 for i=1 to |V(G)|-1
5   for uv ∈ E(G)
6     dist(v)=min{dist(v), dist(u)+luv}

```

SIMULACIÓN DE DIJKSTRA

$A=0$	$D=\infty$
$B=4$	$E=\infty$
$C=2$	
$A=0$	$D=6$
$B=3$	$E=7$
$C=2$	
$A=0$	$D=5$
$B=3$	$E=6$
$C=2$	
$A=0$	$D=5$
$B=3$	$E=6$
$C=2$	