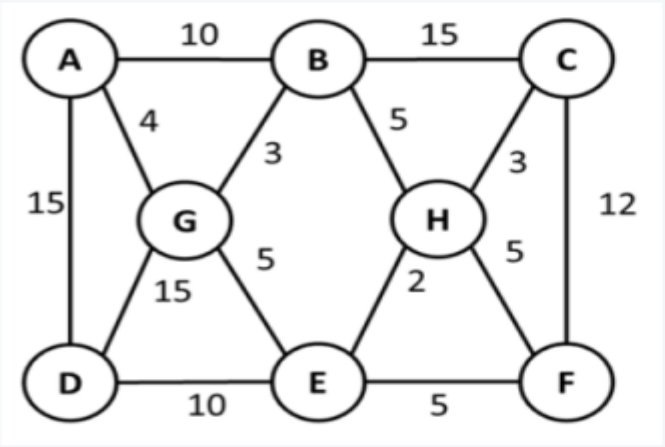
Aplicando el algoritmo de KRUSKAL al siguiente grafo,



Seleccione una:

a. la tercer arista que se elige es la GE

b. un vez elegida la cuarta arista, el costo computado hasta ese momento es 12

c. la segunda arista que se elige es la EH

d. una vez elegidas cinco aristas, la cantidad de componentes conexos es 4

A-

B-

C- No es posible, EH tiene el costo mínimo del grafo, va a ser la primer arista elegida

D- Imposible. No es posible tener 5 aristas y 4 componentes separados.(Minimo para tener 4 componentes es 8 vertices. Si tenes 8 vertices tenes 4 aristas. Cuando tenes los 8 vertices (total) por algoritmo de Kruskal tienes que tener 7 aristas (n-1))

Respuesta: CORRECTA B

HE - 2

HC - 3

GB - 3

AG - 4

TOTAL: 12

HF - 5

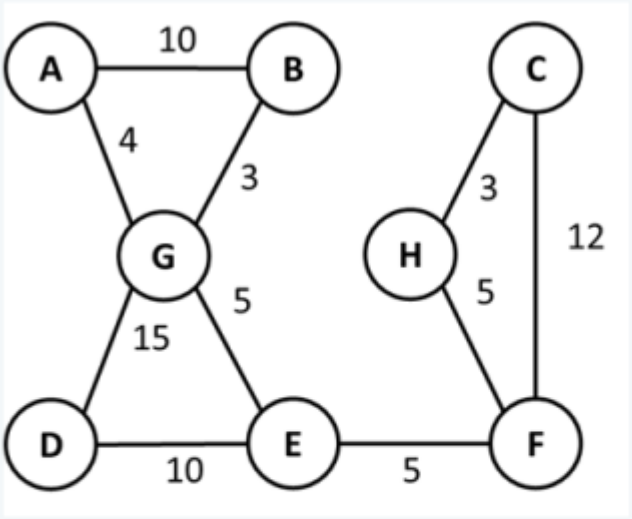
GE - 5

ED - 10

```
Método TGrafo Kruskal ;
F conjunto de aristas; // T: (V, F)

Comienzo
  F.Vaciar;
  Repetir
     elegir una arista de costo mínimo tal que
          no esté   en F  ni haya sido elegida;
     Si la arista no conecta dos vértices del
mismo    componente  entonces agregarla a F;
  hasta que  todos los vértices estén en un sólo
          componente;
fin;
```

Buscando puntos de articulación en este grafo,



Seleccione una:

a. se llega a la conclusión de que el grafo es biconexo

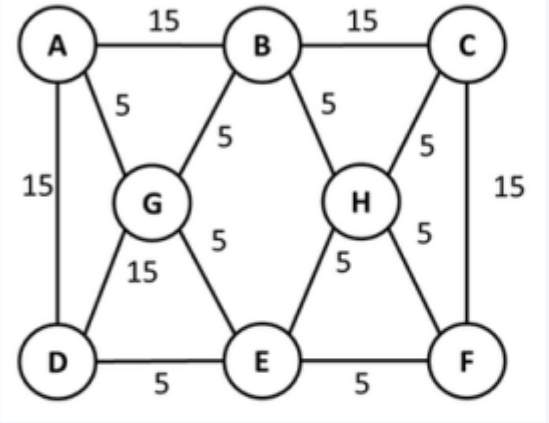
b. se encuentran dos

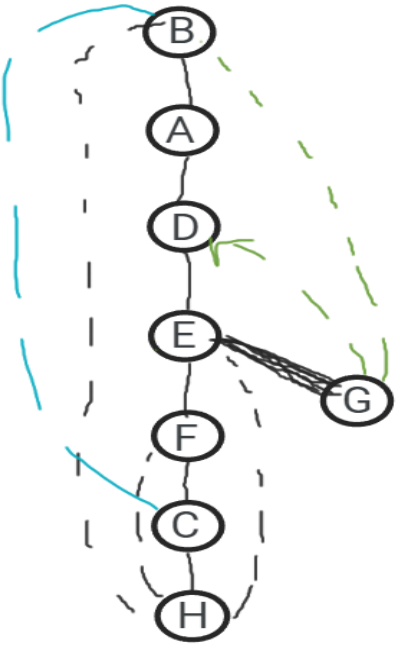
c. se encuentran tres

d. no se encuentra ninguno

Respuesta:
C - Si se eliminan (G, E o F) se rompe el grafo.

Si se aplica una búsqueda en **profundidad** a partir del vértice "B" y tomando alfabéticamente el orden de los adyacentes.





Seleccione una:

a. el arco GE es cruzado y el arco BH es de avance

b. el arco HB es de retroceso y el arco FE es de árbol

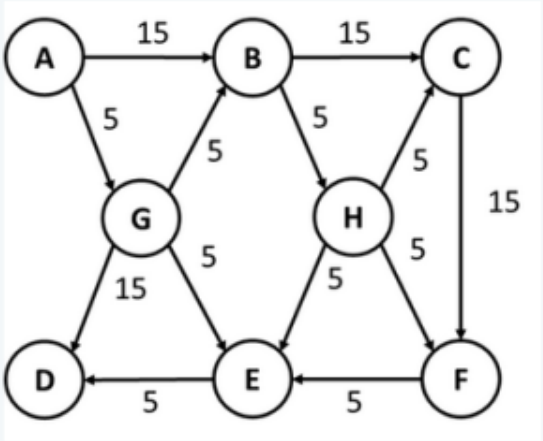
c. el arco BC es de árbol y el arco GB es de retroceso

d. el arco BH es cruzado y el arco HF es de árbol

Quitar mi elección

A- No es posible. No existen arcos cruzados en **BPF NO DIRIGIDO**
B- Correcta
C- GB efectivamente es de retroceso pero como es orden de acceso alfabético el arco de árbol desde B va hacia A
D- No es posible. No existen arcos cruzados en BPF

Suponiendo que el siguiente grafo dirigido acíclico representa un proyecto que tiene como inicio el vértice D y como fin el vértice A (ya tiene las aristas invertidas), un orden posible para cumplir con las restricciones y completar el proyecto es:



Seleccione una:

a. D, G, E, F, C, H, B, A

b. D, E, F, C, B, H, G, A

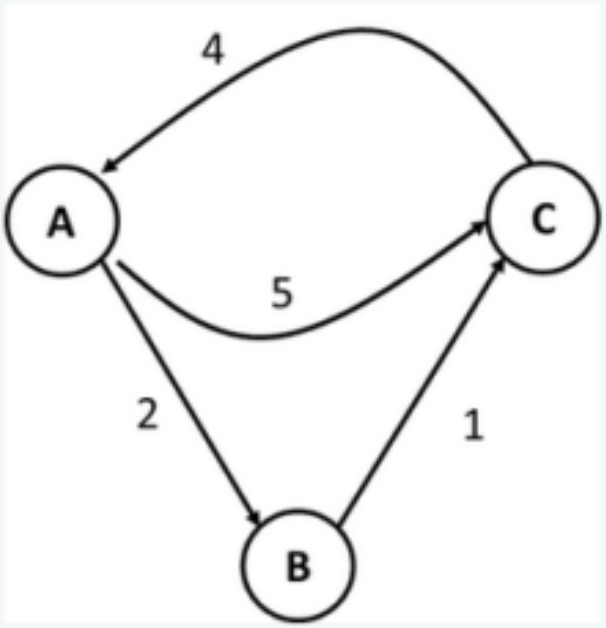
c. D, E, F, C, H, B, G, A

d. D, F, E, C, B, H, G, A

Respuesta:
Es recorrer el grafo con las aristas invertidas
A- NO. G ->E no puede ser ya que las aristas estan invertidas
B- NO. H no conecta con G
C- Respuesta Correcta
D- NO. D no conecta con F

Vamos recorriendo los vértices hasta llegar a D, algun camino nos va a llevar a D

En el grafo de la figura, el centro es



Seleccione una:

a. A o B indistintamente

b. C

c. A

d. B

Respuesta:

A-

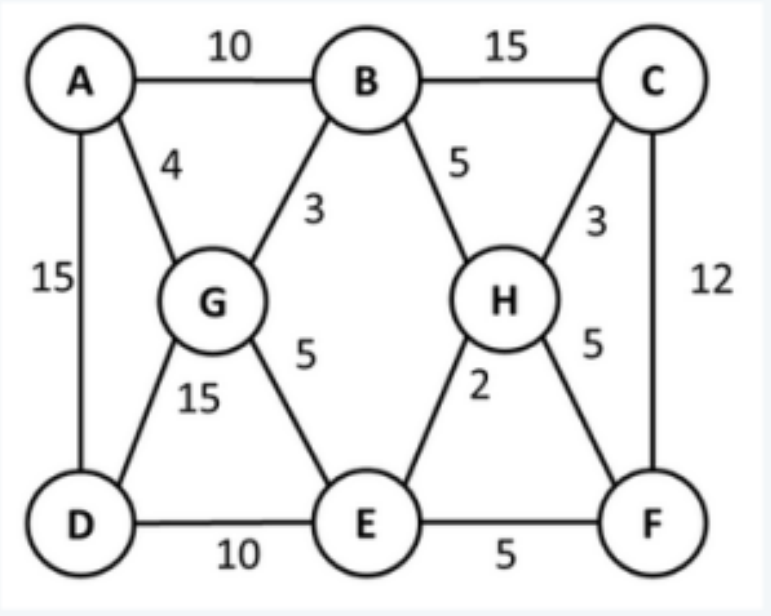
B-

C- Correcta (5)

D-

	A	B	C
A	**0**	**2**	**5**
B	**inf**	**0**	**1**
C	**4**	**inf**	**0**
EXCENTRI	**inf**	**inf**	**5**

Aplicando el algoritmo de PRIM a partir el vértice A al siguiente grafo.



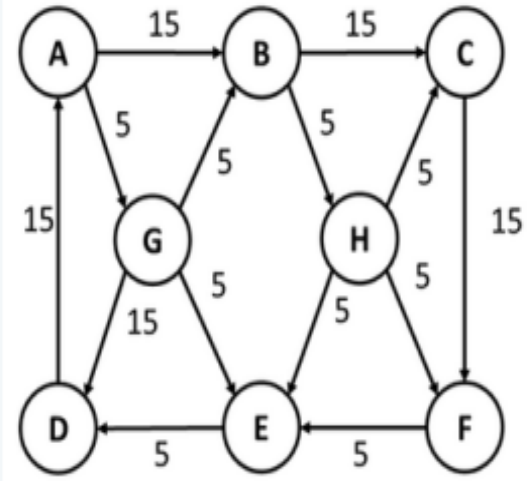
Método TGRAFO. Prim (conjunto de aristas T);
U: conjunto de vértices;
u, v: vértice;
// el TGRAFO representado por un conjun
de vértices V y un conjunto de Aristas A
COMIENZO
T. Vaciar;
U.Agregar (1);
MIENTRAS U <> V **hacer**
elegir una arista (u,v) *de costo mínim*
tal que u *está en* U *y* v *está en* V-U;
T.agregar (u,v);
U.agregar(v);
FIN MIENTRAS
FIN;

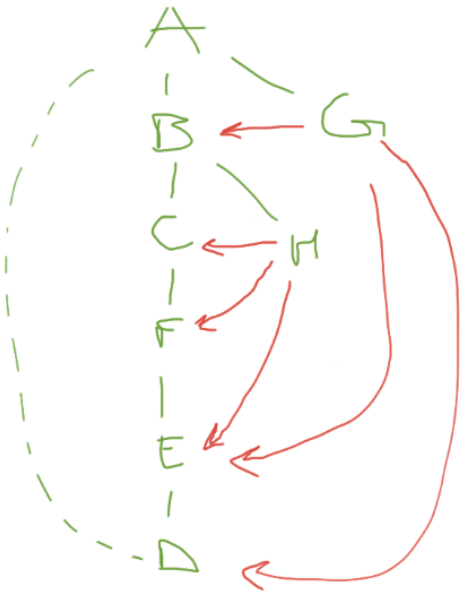
Seleccione una:

a. una vez elegidas cinco aristas, la cantidad de componentes conexos es 4
b. la segunda arista que se elige es la EH
c. un vez elegida la cuarta arista, el costo computado hasta ese momento es 12
d. la tercer arista que se elige puede ser la GE

A- Imposible. No es posible tener 5 aristas y 4 componentes separados.(Minimo para tener 4 componentes es 8 vertices. Si tenes 8 vertices tenes 4 aristas. Cuando tenes los 8 vertices (total) por algoritmo de Kruskal tienes que tener 7 aristas (n-1)).
B- Comienza desde A entonces el primer arco es A-G, el segundo será G-B. Para que haya una chance de que suceda, E o H debería ser adyacente a A.
C- Al momento de seleccionar 3 aristas ya tenemos un costo costo computado hasta el momento de 12. Elegir una nueva arista es al remilpedo
D- Puede ser perfectamente GE, o BH ya que ambas aristas tienen costo minimo (5) y u esta en U y v está en VU CORRECTA

Si se aplica una búsqueda en **profundidad** a partir del vértice "A" y tomando alfabéticamente el orden de los adyacentes.





Seleccione una:

a. el arco BC es de árbol y el arco GB es de retroceso

b. el arco BH es cruzado y el arco HF es de árbol

c. el arco GE es cruzado y el arco BH es de avance

d. el arco DA es de retroceso y el arco FE es de árbol

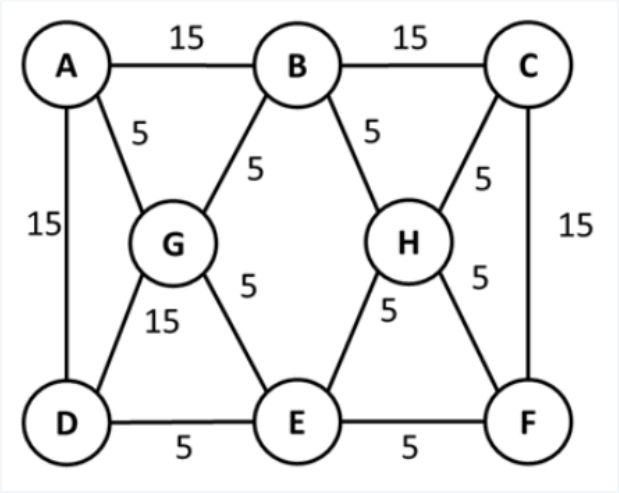
Quitar mi elección

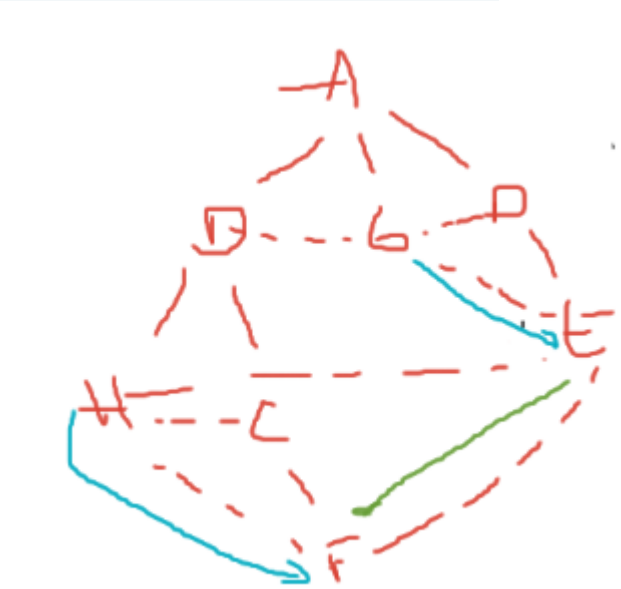
Respuesta:
A- GB es cruzado
B- HF no es de árbol
C- Si bien GE es cruzado, BH es de árbol.
D- Correcta

Diferencia Arco Cruzado y de retroceso.

Si de un vértice voy a otro ya visitado: Si es padre directo (mismo subárbol) es de retroceso
Si no es un ancestro directo (está en otro subárbol) es cruzado.

Si se aplica una búsqueda en **amplitud** a partir del vértice "B" y tomando alfabéticamente el orden de los adyacentes,





Seleccione una:

a. el arco DA es de retroceso y el arco FE es cruzado

b. el arco BC es de árbol y el arco GB es de retroceso

c. el arco GE es cruzado y el arco BH es de avance

d. el arco BH es de árbol y el arco HF es cruzado

A- Falso DA es de árbol
B- FALSO GB es cruzado
C- BH es de árbol
D- VERDADERO HF es cruzado

Avance-> pertenecen al mismo subarbol pero no son directos
Cruzado-> diferentes subarboles

Se define **excentricidad** de un vértice de un grafo como

Seleccione una:

a. La menor de todas las distancias de menor costo que llegan a él desde todos los otros vértices.

b. La mayor de todas las distancias de menor costo que llegan a él desde todos los otros vértices.

c. La menor de todas las distancias de mayor costo que llegan a él desde todos los otros vértices.

d. La mayor de todas las distancias de mayor costo que llegan a él desde todos los otros vértices.

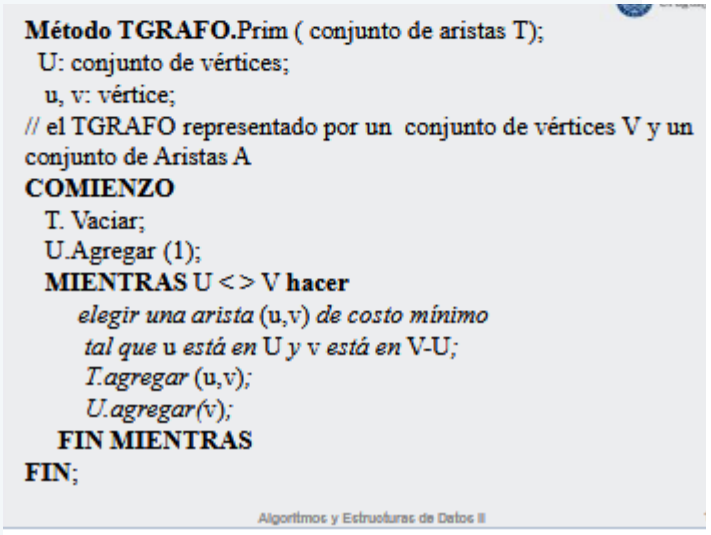
Quitar mi elección

CORRECTA: b) **La mayor de todas las distancias de menor costo que llegan a él desde todos los otros vértices.**

El algoritmo de PRIM para obtener un árbol abarcador de costo mínimo está incompleto. La sentencia que falta es

```
Método TGRAFO.Prim ( conjunto de aristas T);
  U: conjunto de vértices;
  u, v: vértice;

COMIENZO
  T. Vaciar;
  U.Agregar (1);
  MIENTRAS U < > V hacer
     sentencia que falta
      T.agregar (u,v);
      U.agregar(v);
   FIN MIENTRAS
FIN;
```



Método TGRAFO.Prim (conjunto de aristas T);
U: conjunto de vértices;
u, v: vértice;
// el TGRAFO representado por un conjunto de vértices V y un conjunto de Aristas A
COMIENZO
T. Vaciar;
U.Agregar (1);
MIENTRAS U < > V hacer
elegir una arista (u,v) *de costo mínimo*
tal que u *está en* U *y* v *está en* V-U;
T.agregar (u,v);
U.agregar(v);
FIN MIENTRAS
FIN;

Algoritmos y Estructuras de Datos II

Seleccione una:

- a. elegir una arista (u,v) de costo mínimo tal que u está en U y v está en V-U;
- b. elegir dos vértices u y v de costo mínimo tal que u está en U y v está en V-U;
- c. elegir un vértice v de costo mínimo tal que está en V-U;
- d. elegir una arista (u,v) de costo mínimo tal que u está en U y v está en V;

Quitar mi elección

RESPUESTA: (A)

El algoritmo para recuperar los caminos en la matriz "P" obtenida de la aplicación del método de Floyd se escribe con las sentencias en cualquier orden. El orden correcto es:

procedure camino (i, j : enteros)
Comienzo
1 Si k = 0 entonces
2 escribir (k);
3 salir
4 camino (i,k);
5 finsi;
6 camino (k,j);
7 k = P[i,j];
Fin

```
procedure camino ( i, j: integer
var k: integer;
begin
    k := P[i, j];
    if k = 0 then salir;
    camino(i, k);
    imprimir(k);
    camino(k, j)
end; { camino }
```

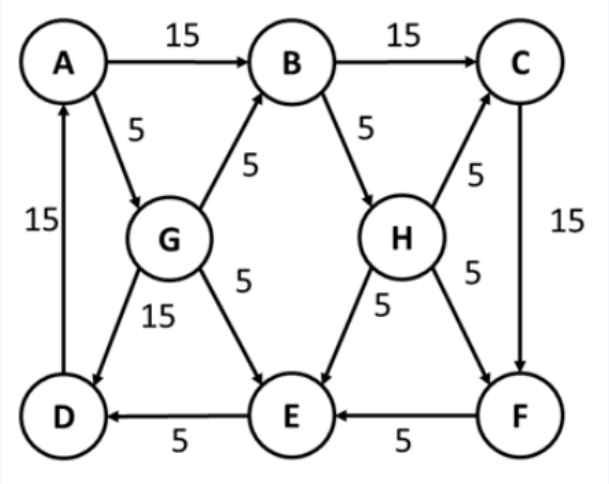
1
2
3

Seleccione una:

a. 1, 7, 6, 5, 4, 2, 3
b. 7, 1, 5, 3, 4, 6, 2
c. 1, 4, 6, 5, 7, 2, 3
d. 7, 1, 3, 5, 4, 2, 6

procedure camino (i, j : enteros)
Comienzo
7 k = P[i,j];
1 Si k = 0 entonces
3 salir
5 finsi;
4 camino (i,k);
2 escribir (k);
6 camino (k,j);
Fin

Si se aplica una búsqueda en **amplitud** a partir del vértice "A" y tomando alfabéticamente el orden de los adyacentes,



Seleccione una:

a. el arco BC es de árbol y el arco GB es de retroceso

b. el arco DA es de retroceso y el arco FE es cruzado

c. el arco GE es cruzado y el arco BH es de avance

d. el arco BH es cruzado y el arco HF es de árbol

a) **GB es cruzado (SON DE SUB-ARBOLES DIFERENTES)**
b) **CORRECTO**
c) **GE es de árbol porque es hijo de G**
d) **BH es de árbol**

```
bea (TAristas aristas)
COMIENZO
    TCola cola = nueva TCola
    this.visitar
    cola.agregar(this)
    mientras <sentencias que faltan> hacer
        v = cola.quitar
        para cada w adyacente a v hacer
            Si no visitado w entonces
                w.visitar
                aristas.agregar((v,w))
                <sentencias que faltan>
            finsi
        fin para cada
    fin mientras
FIN
```

Seleccione una:

- a. no (cola.vacia) y cola.agregar(w)
- b. no (cola.vacia) y w.bea (aristas)
- c. cola != aristas y cola.agregar(w)
- d. cola != aristas y w.bea (aristas)

Algoritmo de búsqueda en amplitud.

- Método Tvertice ***bea*** : String
- //{bea visita todos los vértices conectados a ***v*** usando búsqueda en amplitud.}

- C: ColaDeVértices;
- x,y : Vértice;
- tempstr : String

- **COM**
- Visitar();
- C.Insertar (this);
- tempstr <- tempstr + etiqueta
- **mientras no** vacía(C) **hacer**
- x ← C.eliminar () // en x queda el elemento frente de la cola
- **para** cada vértice ***y*** adyacente a ***x*** **hacer**
- **Si no** y.Visitado() **entonces**
- y.Visitar();
- C.insertar (y);
- tempstr <- tempstr + y.etiqueta
- **fin si**
- **fin para cada**;
- **fin mientras**;
- Devolver tempstr
- **FIN**; {bea}

```
hayCiclo(TCamino camino)
COMIENZO
    this.visitar;
    camino.agregar(this)
    para cada v adyacente de this hacer
```

```java
@Override
public boolean tieneCiclo(LinkedList<Comparable> verticesVisitados) {
    this.setVisitado(true);
    verticesVisitados.add(this.getEtiqueta());
    for (TAdyacencia adyacencia : this.getAdyacentes()) {
```

Seleccione una:

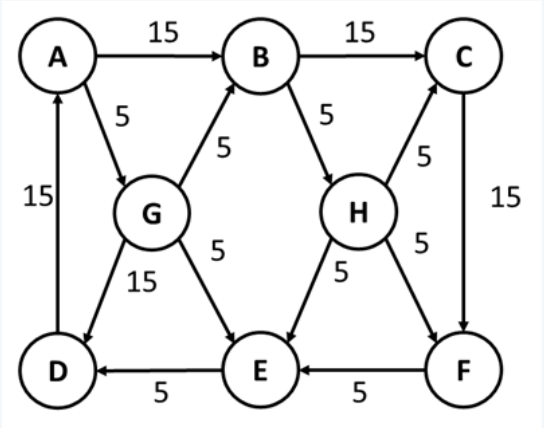
a. Si Camino.contiene(v) entonces
imprimir("hay ciclo")
finsi
y
Camino.quitar(this)

b. imprimir("hay ciclo")
y
Camino.quitar(this)

c. imprimir("hay ciclo")
y
this.desvisitar

d. Si Camino.contiene(v) entonces
imprimir("hay ciclo")
finsi
y
Camino.quitar(v)

Si se aplica el método de DIJKSTRA para hallar los caminos de menor costo a partir del vértice "A" , al finalizar la primera iteración (en la que se elige al primer vértice destino) del algoritmo:



Función Dijkstra
COM
Inicializar ***S, D***
S = {1};
para i = 2 a n hacer D[i] = C[1,i] *//(el valor inicial, infinito si //no hay camino directo)*
Mientras ***V <> S*** **hacer**
Elegir ***w*** **perteneciente a** ***V-S,*** tal que la distancia ***D[w]*** sea un **mínimo**
Agregar ***w*** a ***S***
ParaCada ***v*** perteneciente a ***V-S*** hacer
D[v] = min (D[v], D[w]+ costo(w,v)
FinMientras;
FIN {Dijkstra}

Seleccione una:

a. la distancia computada hasta el momento para llegar a C es 25.

b. no se sabe aún si hay un camino para llegar a D

c. el vector de predecesores aún no se ha modificado.

d. se sabe que el próximo vértice a elegir puede estar entre el E o el B indistintamente.