Apuntes elaborados por: Aaron Asencio, Eduardo Quevedo, Raquel López
Revisado por: Javier Miranda el ¿???

Tema 9: Grafos

Los grafos no son más que la versión general de un árbol, es decir, cualquier nodo de un grafo puede apuntar a cualquier otro nodo de éste (incluso a él mismo).

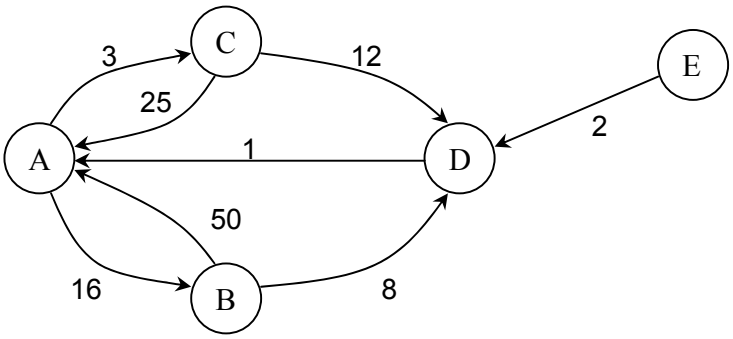
Este tipo de estructuras de datos tienen una característica que lo diferencia de las estructuras que hemos visto hasta ahora: los grafos se usan para almacenar datos que están relacionados de alguna manera (relaciones de parentesco, puestos de trabajo, ...); por esta razón se puede decir que los grafos representan la estructura real de un problema.

En lo que a ingeniería de telecomunicaciones se refiere, los grafos son una importante herramienta de trabajo, pues se utilizan tanto para diseño de circuitos como para calcular la mejor ruta de comunicación en Internet.

Terminología de grafos:

- **Vértice** : Nodo.
- **Enlace** : Conexión entre dos vértices (nodos).
- **Adyacencia** : Se dice que dos vértices son adyacentes si entre ellos hay un enlace directo.
- **Vecindad** : Conjunto de vértices adyacentes a otro.
- **Camino** : Conjunto de vértices que hay que recorrer para llegar desde un nodo origen hasta un nodo destino.
- **Grafo conectado** : Aquél que tiene camino directo entre todos los nodos.
- **Grafo dirigido** : Aquél cuyos enlaces son unidireccionales e indican hacia donde están dirigidos.
- **Gafo con pesos** : Aquél cuyos enlaces tienen asociado un valor. En general en este tipo de grafos no suele tener sentido que un nodo se apunte a sí mismo porque el coste de este enlace sería nulo.

Representación en memoria de un grafo:



Al igual que ocurría con el árbol, también hay dos formas de representar un grafo en memoria:

- ✓ **Matricial:** Usamos una *matriz cuadrada de boolean* en la que las filas representan los nodos origen, y las columnas, los nodos destinos. De esta forma, cada intersección entre fila y columna contiene un valor booleano que indica si hay o no conexión entre los nodos a los que se refiere. Si se trata de un grafo con pesos, en lugar de usar valores booleanos, usaremos los propios pesos de cada enlace y en caso de que no exista conexión entre dos nodos, rellenaremos esa casilla con un valor que represente un coste ∞, es decir, con el valor Natural'Last. A esta matriz se le llama *Matriz de Adyacencia*.

  Ejemplo: Representación matricial del grafo anterior:

  - Si no tuviera pesos:

	A	B	C	D	E
A	0	1	1	0	0
B	1	0	0	1	0
C	1	0	0	1	0
D	1	0	0	0	0
E	0	0	0	1	0

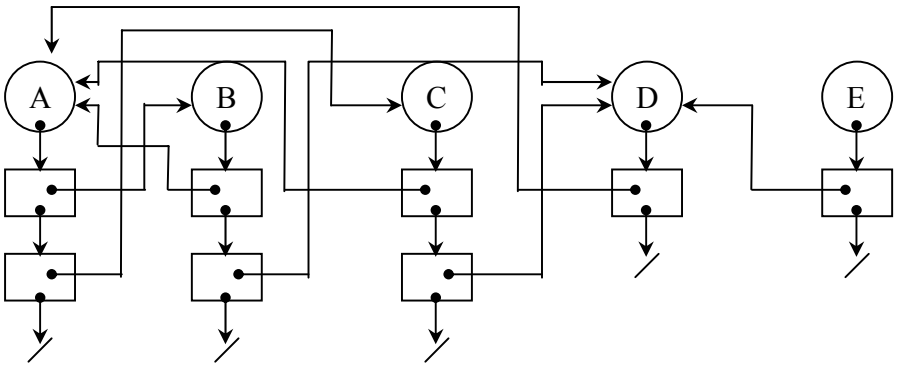
  - Teniendo en cuenta los pesos:

	A	B	C	D	E
A	0	16	3	∞	∞
B	50	0	∞	8	∞
C	25	∞	0	12	∞
D	1	∞	∞	0	∞
E	∞	∞	∞	2	0

  OBSERVACIÓN: En caso de que el grafo sea no dirigido, la matriz resultante es simétrica respecto a la diagonal principal.

  NOTA: La representación de un grafo mediante una matriz sólo es válida cuando el número de nodos del grafo es fijo.

- ✓ **Dinámica:** Usamos *listas dinámicas*. De esta manera, cada nodo tiene asociado una lista de punteros hacia los nodos a los que está conectado:



Por simplicidad, nosotros vamos a representar los grafos por el método matricial.

Recorrido de un grafo:

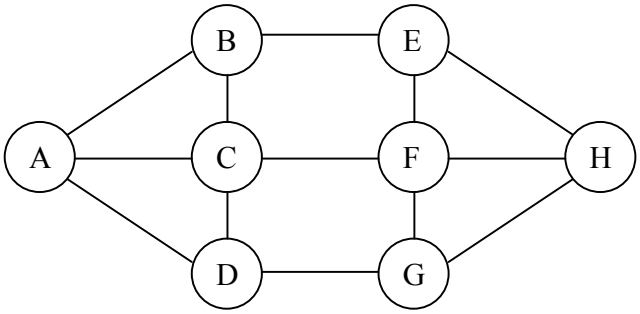
Lo que queremos conseguir es pasar por todos los nodos de un grafo evitando tratarlos dos veces (por ejemplo, si se están escribiendo en pantalla el contenido de todos los nodos del grafo hay que evitar que un nodo se escriba dos veces). Para ello es importante usar o bien un *array de boolean* que represente todos los nodos del grafo, o bien añadir un *campo "Visitado" de tipo boolean* dentro de cada nodo. De esta forma, cuando pasemos la primera vez por un nodo, pondremos su campo "Visitado" a True. Esto es necesario para saber si ya hemos pasado por un nodo y así poder evitar pasar de nuevo por él, pues de no saber si lo hemos visitado o no, el algoritmo podría entrar en un bucle infinito saltando siempre entre los dos mismos nodos.

Hay dos tipos de recorridos de un grafo, y pueden ser implementados tanto de forma recursiva como de forma iterativa. Estos recorridos son:

- **En profundidad**, que consiste en alejarse todo lo posible del nodo origen para después empezar a visitar los nodos restantes a la vuelta.

- **A lo ancho** (o **por nivel**), que consiste en visitar primero los nodos vecinos del origen, luego los vecinos de éstos, y así sucesivamente.

Veamos como puede implementarse de forma iterativa cada uno de estos dos recorridos:



Recorrido en profundidad: En este tipo de recorrido hay que usar una pila para ir almacenando los nodos que nos falta visitar. Vamos a ver como se comportaría este algoritmo si queremos recorrer el grafo anterior partiendo de "A":

Usamos un array para ir marcando los nodos por los que hemos pasado:

	A	B	C	D	E	F	G	H
Visitados	F	F	F	F	F	F	F	F

Usamos un puntero "Actual" para ir moviéndonos por el grafo.

Empieza el algoritmo:

1. Actual := "A". Marcamos "A" como visitado.

	A	B	C	D	E	F	G	H
Visitados	T	F	F	F	F	F	F	F

2. Marcamos todos los vecinos no marcados de "A" como visitados y los metemos en la pila.

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	F	F	F	F

Pila
D
C
B

3. Hacemos un pop de la pila y avanzamos "Actual" hacia ese puntero, es decir, hacia "D".

4. Marcamos todos los vecinos no marcados de "D" como visitados y los metemos en la pila.

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	F	F	T	F

Pila
G
C
B

5. Hacemos un pop de la pila y avanzamos "Actual" hacia ese puntero, es decir, hacia "G".

6. Marcamos todos los vecinos no marcados de “G” como visitados y los metemos en la pila.

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	F	T	T	T

Pila

H
F
C
B

7. Hacemos un pop de la pila y avanzamos “Actual” hacia ese puntero, es decir, hacia “H”.

8. Marcamos todos los vecinos no marcados de “H” como visitados y los metemos en la pila.

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	T	T	T	T

Pila

E
F
C
B

9. Hacemos un pop de la pila y avanzamos “Actual” hacia ese puntero, es decir, hacia “E”.

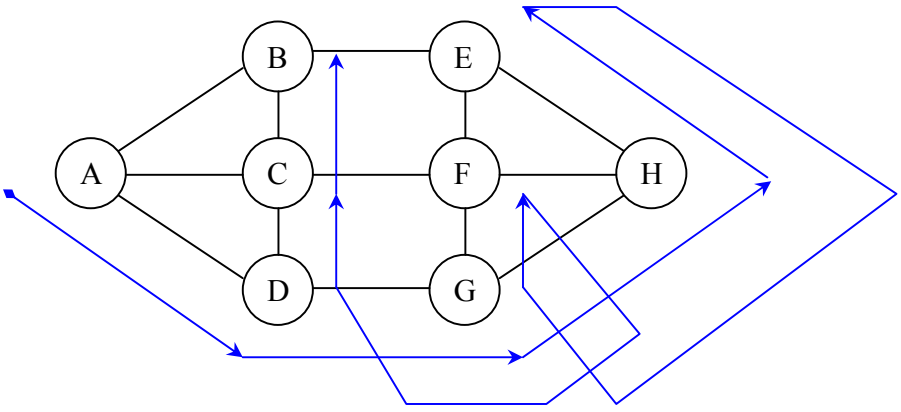
10. “E” no tiene vecinos no visitados. Hacemos un pop de la pila y avanzamos “Actual” hacia ese puntero, es decir, hacia “F”.

11. “F” no tiene vecinos no visitados. Hacemos un pop de la pila y avanzamos “Actual” hacia ese puntero, es decir, hacia “C”.

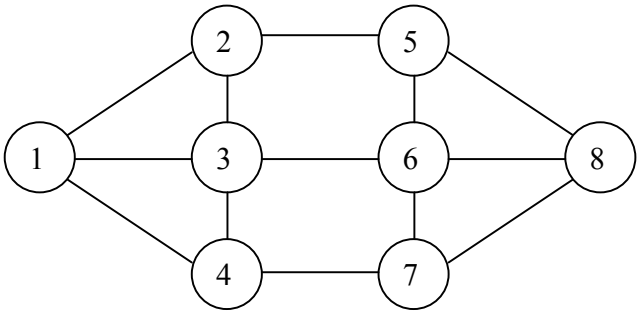
12. “C” no tiene vecinos no visitados. Hacemos un pop de la pila y avanzamos “Actual” hacia ese puntero, es decir, hacia “B”.

13. “B” no tiene vecinos no visitados. La pila está vacía → FIN del algoritmo.

El camino que hemos hecho es el siguiente: A, D, G, H, E, F, C, B.



Ahora analizamos la correspondiente implementación en Ada. Para facilitar su explicación utilizamos un grafo de números enteros, una pila y un array de Insertados, además de los procedimientos Push, Pop, PilaVacia y Vaciar_Pila explicados en clase:




```
type  T_Insertados is array (1 .. 8) of Boolean;
procedure Push ( P : in out T_Pila; Dato : in Integer);
procedure Pop  (P : in out T_Pila; Dato : out Integer);
function PilaVacia (P : in T_Pila) return Boolean;
procedure Vaciar_Pila (P : in out T_Pila);

procedure Recorrrer_Ancho (A : in T_Matriz; Nodo_Comienzo : in Positive) is
    V : T_Insertados := (others => false);
    P : T_Pila;
    Actual : Positive ;
begin
    Vaciar_Pila (P);
    Push (P, Nodo_Comienzo) ;
    V (Nodo_Comienzo) := True ;
    while not PilaVacia (P) loop
            Pop (P, Actual) ;
            Text_IO.Put_Line (Actual) ;
            for I in A'Range(2) loop
                    if (A(Actual,i) < Integer'Last) and not V(i) then
                            Push (P, i);
                            V(i) := True;
                    end if;
            end loop;
end Recorrer_Ancho;
```

Recorrido a lo ancho: La implementación de este recorrido es exactamente igual que la del recorrido en profundidad, con la única diferencia de que en lugar de una pila, usamos una cola. Veamos como se comportaría este algoritmo si queremos recorrer el grafo anterior partiendo de "A":

Al igual que antes necesitamos un array de "Visitados":

	A	B	C	D	E	F	G	H
Visitados	F	F	F	F	F	F	F	F

Usamos un puntero "Actual" para ir moviéndonos por el grafo.

Empieza el algoritmo:

1. Actual := "A". Marcamos "A" como visitado.

	A	B	C	D	E	F	G	H
Visitados	T	F	F	F	F	F	F	F

2. Marcamos todos los vecinos no marcados de "A" como visitados y los metemos en la cola.

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	F	F	F	F

Cola
D
C
B

3. Sacamos de la cola y avanzamos "Actual" hacia ese puntero, es decir, hacia "B".

4. Marcamos todos los vecinos no marcados de "B" como visitados y los metemos en la cola.

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	T	F	F	F

Cola
E
D
C

5. Sacamos de la cola y avanzamos "Actual" hacia ese puntero, es decir, hacia "C".

6. Marcamos todos los vecinos no marcados de “C” como visitados y los metemos en la cola.

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	T	T	F	F

Cola

F
E
D

7. Sacamos de la cola y avanzamos “Actual” hacia ese puntero, es decir, hacia “D”.

8. Marcamos todos los vecinos no marcados de “D” como visitados y los metemos en la cola.

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	T	T	T	F

Cola

G
F
E

9. Sacamos de la cola y avanzamos “Actual” hacia ese puntero, es decir, hacia “E”.

10. Marcamos todos los vecinos no marcados de “E” como visitados y los metemos en la cola.

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	T	T	T	T

Cola

H
G
F

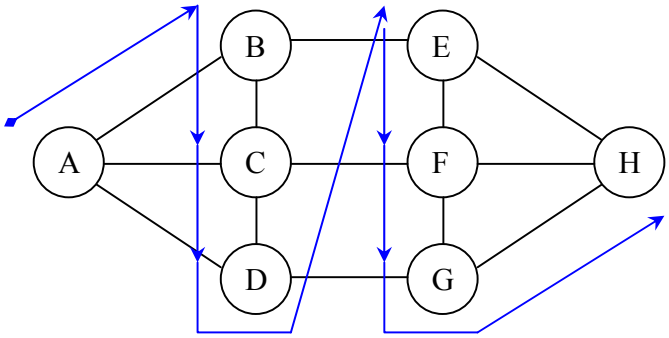
11. Sacamos de la cola y avanzamos “Actual” hacia ese puntero, es decir, hacia “F”.

12. “F” no tiene vecinos no visitados. Sacamos de la cola y avanzamos “Actual” hacia ese puntero, es decir, hacia “G”.

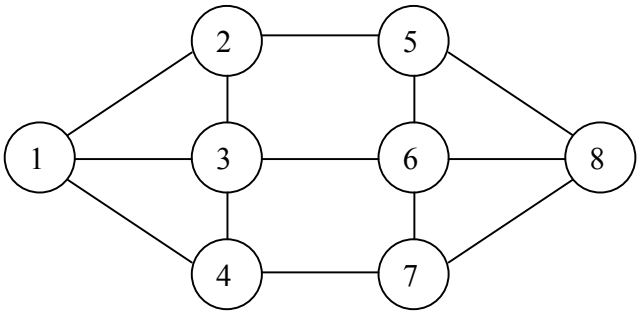
13. “G” no tiene vecinos no visitados. Sacamos de la cola y avanzamos “Actual” hacia ese puntero, es decir, hacia “H”.

14. “H” no tiene vecinos no visitados. La cola está vacía → FIN del algoritmo.

El camino que hemos hecho es el siguiente: A, B, C, D, E, F, G, H.



Ahora analizamos la correspondiente implementación en Ada. Para facilitar su explicación utilizamos un grafo de números enteros, una cola y un array de Insertados, además de los procedimientos Insertar, Extraer, ColaVacia y Vaciar_Cola explicados en clase:



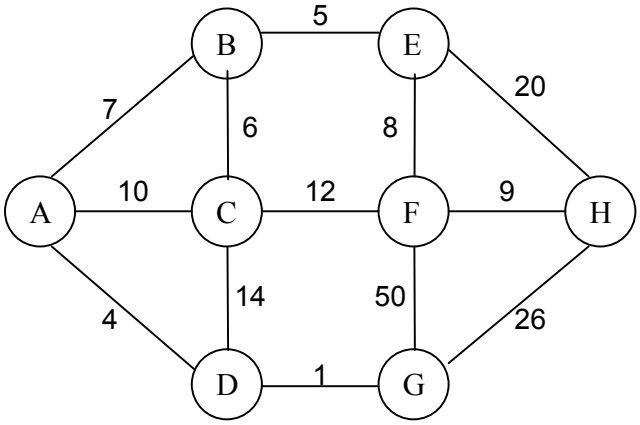

```
type  T_Insertados is array (1 .. 8) of Boolean;
procedure Insertar (C : in out T_Cola; Dato : in Integer);
procedure Extraer ( C : in out T_Cola; Dato : out Integer);
function ColaVacia (C : in T_Cola ) return Boolean;
procedure Vaciar_Cola ( C : in out T_Cola);


procedure Recorrer_Ancho ( A : in T_Matriz ; Nodo_Comienzo) is
    V : T_Insertados;
    C : T_Cola;
    Actual : Positive;
begin
    Vaciar_Cola ( C);
    Insertar ( C, Nodo_Comienzo);
    V (Nodo_Comeinzo) : True;
```

```
        while not ColaVacia ( C )  loop
                Extraer (C, Actual);
                Text_IO.Put_Line (Actual);
                for I in A'Range (2) loop
                        if  (A(Actual,i) < Integer'Last) and not V(i) then
                                Insertar (C,i);
                                V(i) := True;
                        end if;
                end loop;
        end loop;
end Recorrer_Ancho;
```

<u>Algoritmo de Dijkstra para calcular cual es el primer paso que hay que dar si queremos recorrer el camino de menor coste en un grafo con pesos</u>:

Vamos a calcular el coste y la ruta para ir a cada nodo del siguiente grafo a partir del nodo "A":



La matriz que representa este grafo es:

	A	B	C	D	E	F	G	H
A	0	7	10	4	∞	∞	∞	∞
B	7	0	6	∞	5	∞	∞	∞
C	10	6	0	14	∞	12	∞	∞
D	4	∞	14	0	∞	∞	1	∞
E	∞	5	∞	∞	0	8	∞	20
F	∞	∞	12	∞	8	0	50	9
G	∞	∞	∞	1	∞	50	0	26
H	∞	∞	∞	∞	20	9	26	0

Para hacer este algoritmo nos hará falta utilizar lo siguiente:

- Un array “Visitados” de Boolean:

	A	B	C	D	E	F	G	H
Visitados	F	F	F	F	F	F	F	F

- Un puntero “Actual” para ir recorriendo el grafo.

- Un array “Costes” de Natural en el que iremos guardando una estimación de los costes para llegar a cada nodo. Este array se inicia a Natural’Last, es decir, a ∞, para que al comparar costes cualquier camino sea menor que éste:

	A	B	C	D	E	F	G	H
Costes	∞	∞	∞	∞	∞	∞	∞	∞

- Un array “Ruta” en la que recordaremos cual es el primer paso que hay que dar para llegar a cada nodo por el camino de menor peso.

Empezamos el algoritmo:

1. Actual := “A”. Marcamos “A” como visitado.

	A	B	C	D	E	F	G	H
Visitados	T	F	F	F	F	F	F	F

2. Copiamos la fila “A” de la matriz en el array de costes y rellenamos las casillas correspondientes del array de ruta con las direcciones de los nodos vecinos de “A”:

	A*	B	C	D	E	F	G	H
Costes	∞	7	10	4	∞	∞	∞	∞
Ruta	∞	B	C	D	∞	∞	∞	∞

3. En el array de costes, elegimos el nodo de menor coste de los que todavía no han sido visitados, es decir, el “D” (Actual = “D”), y lo marcamos como visitado:

	A	B	C	D	E	F	G	H
Visitados	T	F	F	T	F	F	F	F

4. Vamos estimando el coste para llegar desde “D” hasta cada uno de sus nodos vecinos no marcados como visitados:

   - Coste para llegar a “C” = 4 + 14 = 18. Comparamos este valor con el coste actual para llegar a “C” que hay en el array de costes: 18 > 10 $\Rightarrow$ Lo dejamos como está.
   - Coste para llegar a “G” = 4 + 1 = 5. Comparamos con el valor que hay en el array de costes: 5 < ∞ $\Rightarrow$ Guardamos el nuevo valor en el array de costes y copiamos el camino que usamos para llegar a “D” en la casilla correspondiente al nodo “G” del array de rutas:

	A*	B	C	D*	E	F	G	H
Costes	∞	7	10	4	∞	∞	5	∞
Ruta	∞	B	C	D	∞	∞	D	∞

5. En el array de costes, elegimos el nodo de menor coste de los que todavía no han sido visitados, es decir, el “G” (Actual = “G”), y lo marcamos como visitado:

	A	B	C	D	E	F	G	H
Visitados	T	F	F	T	F	F	T	F

6. Vamos estimando el coste para llegar desde “G” hasta cada uno de sus nodos vecinos no marcados como visitados:

   - Coste para llegar a “F” = 5 + 50 = 55. Comparamos este valor con el coste actual para llegar a “F” que hay en el array de costes: 55 < ∞ $\Rightarrow$ Guardamos el nuevo valor en el array de costes y copiamos el camino que usamos para llegar a “G” en la casilla correspondiente al nodo “F” del array de rutas:

	A*	B	C	D*	E	F	G*	H
Costes	∞	7	10	4	∞	55	5	∞
Ruta	∞	B	C	D	∞	D	D	∞

   - Coste para llegar a “H” = 5 + 26 = 31. Comparamos este valor con el coste actual para llegar a “H” que hay en el array de costes: 31 < ∞ $\Rightarrow$ Guardamos el nuevo valor en el array de costes y copiamos el camino que usamos para llegar a “G” en la casilla correspondiente al nodo “H” del array de rutas:

	A*	B	C	D*	E	F	G*	H
Costes	∞	7	10	4	∞	55	5	31
Ruta	∞	B	C	D	∞	D	D	D

7. En el array de costes, elegimos el nodo de menor coste de los que todavía no han sido visitados, es decir, el “B” (Actual = “B”), y lo marcamos como visitado:

	A	B	C	D	E	F	G	H
Visitados	T	T	F	T	F	F	T	F

8. Vamos estimando el coste para llegar desde “B” hasta cada uno de sus nodos vecinos no marcados como visitados:

   - Coste para llegar a “C” = 7 + 6 = 13. Comparamos este valor con el coste actual para llegar a “C” que hay en el array de costes: 13 > 10 $\Rightarrow$ Lo dejamos como está.
   - Coste para llegar a “E” = 7 + 5 = 12. Comparamos este valor con el coste actual para llegar a “E” que hay en el array de costes: 12 < ∞ $\Rightarrow$ Guardamos el nuevo valor en el array de costes y copiamos el camino que usamos para llegar a “B” en la casilla correspondiente al nodo “E” del array de rutas:

	A*	B*	C	D*	E	F	G*	H
Costes	∞	7	10	4	12	55	5	31
Ruta	∞	B	C	D	B	D	D	D

9. En el array de costes, elegimos el nodo de menor coste de los que todavía no han sido visitados, es decir, el “C” (Actual = “C”), y lo marcamos como visitado:

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	F	F	T	F

10. Vamos estimando el coste para llegar desde “C” hasta cada uno de sus nodos vecinos no marcados como visitados:

    - Coste para llegar a “F” = 10 + 12 = 22. Comparamos este valor con el coste actual para llegar a “C” que hay en el array de costes: 22 < 55 $\Rightarrow$ Guardamos el nuevo valor en el array de costes y copiamos el camino que usamos para llegar a “C” en la casilla correspondiente al nodo “F” del array de rutas:

	A*	B*	C*	D*	E	F	G*	H
Costes	∞	7	10	4	12	22	5	31
Ruta	∞	B	C	D	B	C	D	D

11. En el array de costes, elegimos el nodo de menor coste de los que todavía no han sido visitados, es decir, el “E” (Actual = “E”), y lo marcamos como visitado:

	A	B	C	D	E	F	G	H

Visitados	T	T	T	T	T	F	T	F

12. Vamos estimando el coste para llegar desde “E” hasta cada uno de sus nodos vecinos no marcados como visitados:

    - Coste para llegar a “F” = 12 + 8 = 20. Comparamos este valor con el coste actual para llegar a “F” que hay en el array de costes: 20 < 22 ⇒ Guardamos el nuevo valor en el array de costes y copiamos el camino que usamos para llegar a “E” en la casilla correspondiente al nodo “F” del array de rutas:

	A*	B *	C*	D*	E*	F	G*	H
Costes	∞	7	10	4	12	20	5	31
Ruta	∞	B	C	D	B	B	D	D

    - Coste para llegar a “H” = 12 + 20 = 32. Comparamos este valor con el coste actual para llegar a “H” que hay en el array de costes: 32 < 31 ⇒ Lo dejamos como está.

13. En el array de costes, elegimos el nodo de menor coste de los que todavía no han sido visitados, es decir, el “F” (Actual = “F”), y lo marcamos como visitado:

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	T	T	T	F

14. Vamos estimando el coste para llegar desde “F” hasta cada uno de sus nodos vecinos no marcados como visitados:

    - Coste para llegar a “H” = 20 + 9 = 29. Comparamos este valor con el coste actual para llegar a “H” que hay en el array de costes: 29 < 31 ⇒ Guardamos el nuevo valor en el array de costes y copiamos el camino que usamos para llegar a “F” en la casilla correspondiente al nodo “H” del array de rutas:

	A*	B *	C*	D*	E*	F*	G*	H
Costes	∞	7	10	4	12	20	5	29
Ruta	∞	B	C	D	B	B	D	B

15. En el array de costes, elegimos el nodo de menor coste de los que todavía no han sido visitados, es decir, el “H” (Actual = “H”), y lo marcamos como visitado:

	A	B	C	D	E	F	G	H
Visitados	T	T	T	T	T	T	T	T

16. Vamos estimando el coste para llegar desde “H” hasta cada uno de sus nodos vecinos no marcados como visitados ⇒ Todos los vecinos de “H” ya han sido visitados ⇒ FIN del algoritmo.

El algoritmo devuelve el array Ruta, que nos dice el primer salto que hay que hacer a partir del nodo origen, en nuestro caso el “A”, para llegar al nodo destino con el menor coste posible. El siguiente salto se decide en el segundo nodo, que tendrá también una tabla análoga a la del nodo “A”, y así sucesivamente hasta llegar a nuestro nodo destino.

Ejercicios propuestos:

1. Dado un grafo sin pesos y con enlaces bidireccionales hacer un procedimiento que busque el camino para llegar desde un nodo origen cualquiera hasta otro nodo destino.

   Planteamiento:

   Primero hay que decidir si hacemos el procedimiento recursivo o iterativo. En principio, si no nos dicen nada, es generalmente más fácil e intuitivo hacerlo recursivo.

   Ahora, hay que ver que tipo de recorrido vamos a hacer, en profundidad o a lo ancho: como el recorrido en profundidad utiliza una pila, es mejor utilizar éste ya que así en la pila tendremos guardado el camino exacto que hemos hecho.

   Pasos a seguir en el algoritmo:

   1. Hacemos un recorrido a partir del nodo origen hasta encontrar el nodo destino.
   2. Copiamos el contenido de la pila de nodos por los que hemos pasado en una pila auxiliar.
   3. Continuamos el recorrido. Si encontramos otro camino que sea menor que el camino guardado en la pila auxiliar, copiamos el nuevo camino en la pila auxiliar.
   4. Volvemos al paso 3.

2. Dado un grafo con pesos y con enlaces unidireccionales, hacer un procedimiento que busque el camino de menor coste para llegar desde un nodo origen cualquiera hasta otro nodo destino.