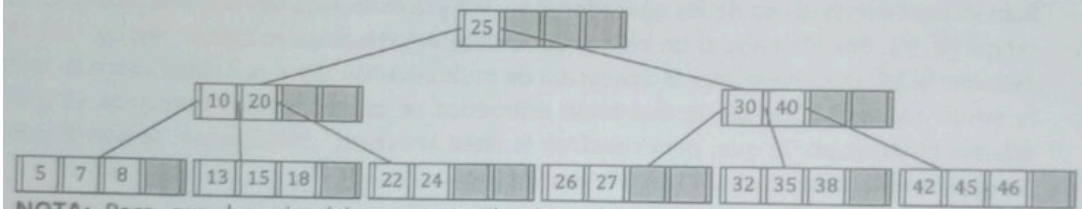
Ejercicio 1

1.2: **[1.5 puntos]** Dado el árbol B de la figura adjunta, respresentar el árbol resultante después de la siguiente secuencia de borrados: 25 - 45 - 24 - 38 - 32 - 8 - 27 - 46 - 13 - 42 - 5 - 22 - 18 - 26 - 7 - 35 - 15.



NOTA: Para que los ejercicios sean evaluados con la máxima calificación, es necesario representar los *pasos intermedios relevantes*. Presentar únicamente el resultado final equivale a una calificación de cero en el ejercicio correspondiente.

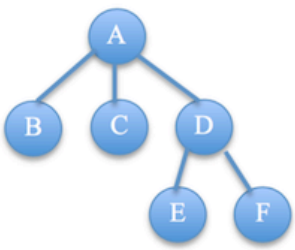
1.3: **[1 punto]** Una agencia de viajes quiere almacenar en una estructura de datos eficiente la información de vuelos con origen en Madrid y destino en algunas de las ciudades más importantes del mundo. En concreto, quiere determinar qué ciudades están a una distancia (en tiempo) menor que un umbral dado. Determine la estructura de datos más adecuada para resolver este problema y represente paso a paso cómo se iría conformado dicha estructura si se asume que la información se introducen en el orden dado por la tabla.

Nº Horas	Destino	Tipo Avión	Nº Pasajeros
2	Barcelona	Boening 727	125
5	Munich	Boeing727	125
10	Tokio	Airbus 340	300
13	Singapure	Airbus 340	300
1	Valencia	Boeing 727	125
12	Pelin	Airbus 340	300
16	Melbourne	Airbus 340	300
14	Tasmania	Airbus 340	300
15	Honolulu	Airbus 340	300

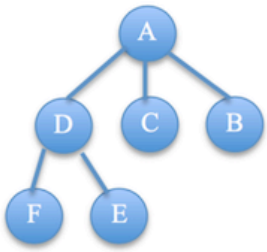
Ejercicio 2 [3 puntos].

Aumentar la funcionalidad de los árboles n-arios (`LinkedTree`) añadiendo los siguientes métodos:

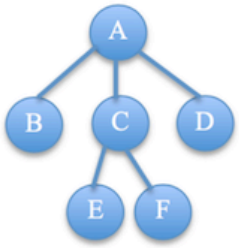
a) **[1.5 puntos]** Método que determine si dos árboles son iguales. Para ello, debe comprobar por niveles si los elementos son iguales (independientemente del orden). Por ejemplo, dado el árbol T1, el método implementado debería determinar que es igual a T2 y distinto a T3.



Árbol T1

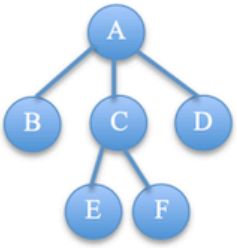


Árbol T2

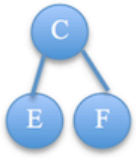


Árbol T3

b) **[1.5 puntos]** Método que reciba el `Position` de un nodo y devuelva (en una copia) el subárbol enraizado en dicho nodo. Por ejemplo, dado el árbol T1 de la figura, si se recibe como argumento de entrada el position del nodo `C`, devolvería el subárbol T2.



Árbol T1



Subárbol T2

Ejercicio 3

Implementar los conocidos algoritmos de Prim y Kruskal estudiados durante la asignatura. Para ello, se deberá utilizar la clase genérica GraphUtils incluida en el proyecto. Se deberá respetar rigurosamente la genericidad que se ha establecido en la clase mediante los parámetros genéricos de la misma.

a) Implementar un método que, dado un grafo, devuelva un Iterable de arcos siguiendo al algoritmo de Kruskal. La cabecera para dicho método debe ser la siguiente:

```
public Iterable <Edge <Integer>> getKruskal(Graph <V,Integer> g)
```

b) Implementar un método que, dado un grafo, devuelva un Iterable de arcos siguiendo al algoritmo de Prim. La cabecera para dicho método debe ser la siguiente:

```
public Iterable <Edge <Integer>> getPrim(Graph <V,Integer> g)
```

c) Implementar un programa de pruebas que instancie un grafo de un tipo concreto y pruebe los métodos anteriores.