Práctica 2: Mapas y diccionarios

Normas:

- Cada estudiante debe realizar los ejercicios de manera individual, aunque pueden compartir información oral. Existe un detector anticopias.
- Este enunciado va acompañado de un fichero ZIP que contiene el código necesario para hacer la práctica. Los test que acompañan a las prácticas son orientativos.

Ejercicio 1: Implementar un diccionario

Implementar la clase `MyDictionary` que implementa la interfaz `Dictionary`. Esta clase comenzará con un *bucket* de 20 posiciones y cuando su factor de carga sea de 0,75 duplicará su tamaño y reasignará las entradas.

Ejercicio 2: contador de colisiones

Partiendo de tablas de tamaño 15000, modificar el código de tabla hash para poder rellenar la siguiente tabla:

	Número de colisiones N=10000	Número de colisiones N=1000000
Prueba lineal		
Prueba cuadrática		
Hashing doble		

Ejercicio 3: Organiza el viaje

Un viajero despistado llega a nosotros con un montón de billetes de avión. Todos ellos tienen un origen y un destino, pero sin fecha. Nuestro trabajo es ordenarlos para poder indicar a nuestro viajero cuál será el itinerario de su viaje, pero además debemos hacerlo con el coste de complejidad más bajo posible. Por ejemplo, los billetes son:

Origen		Destino
New York	→	Boston
Los Angeles	→	Las Vegas
Orlando	→	New York
Las Vegas	→	Orlando

En este caso el itinerario de nuestro viajero será:
“Los Angeles” → “Las Vegas” → “Orlando” → “New York” → “Boston”
Asumiremos que los billetes que nos dan no son cíclicos (no se puede pasar 2 veces por el mismo destino), si esto ocurre no hay origen y se debe generar una excepción. Además, existe un billete desde cada ciudad exceptuando el destino final.
Nos proporcionarán una lista de pares, origen destino, con todos los billetes.

Se pide:

a) Elegir una estructura de datos adecuada para resolver el problema e inicializarla. Para ello el constructor de la clase `Organize` recibirá una lista de pares de ciudades, siendo el primer elemento del par el origen y el segundo el destino.

b) Implementar el método `itineratio()` que devuelve la lista de ciudades que representa el itinerario que debe seguir el viajero. En caso de que no exista origen (por ejemplo si hay un ciclo) se debe lanzar una excepción indicando que no hay un origen (véanse los test).