Ejercicios de clase

TEMA 6 – Grafos y Estructuras de Partición (UF-Sets)

EJERCICIO 1

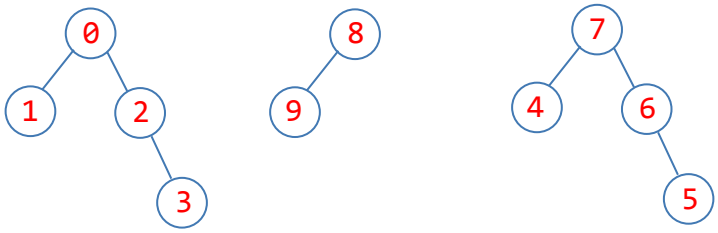
Sea la siguiente representación de un UF-Set:

0	1	2	3	4	5	6	7	8	9
-3	0	0	2	7	6	7	-3	-2	8

a) Dibujar el bosque de árboles que contiene.
b) Teniendo en cuenta que se implementa la fusión por rango y la compresión de caminos, indicar cómo irá evolucionando dicha representación tras la ejecución de las instrucciones de la siguiente tabla. **Nota:** al unir dos árboles con la misma altura (o rango), el primer árbol deberá colgar del segundo.

Solución

a) Hay que dibujar los árboles correspondientes a las clases que hay representadas en el vector del UF-Set. Como debéis recordar, los valores negativos indican que es un elemento raíz, y el valor es el rango del árbol. Las casillas que no son raíz tienen como contenido el índice de su padre. Por ejemplo, el 0 es raíz, el 1 tiene como padre al 0 el 2 tiene como padre al 0, el 3 tiene como padre el 2, etc...



b) Veamos cada operación.

- Al hacer `find(3)` se busca el padre de 3, que es el 2, y el padre de 2 que es el 0. Como se usa Compresión de caminos todos los elementos que se han seguido en ese camino se cuelgan directamente del 0 (que es la raíz), por lo que cambia el padre del 3, que pasa a ser el 0. Nótese que el rango del 0 sigue siendo -3, porque no se puede saber la altura que queda después de este proceso (en el ejemplo sí porque es muy simple, pero el algoritmo no contempla que se cambie el valor del rango), y se deja el que estaba.
- Al hacer `find(5)` se hace el recorrido hasta la raíz, pasando por el 6. Luego todos los elementos de ese camino se cuelgan directamente de la raíz (el 7), en este caso sólo el 5. (De nuevo no se cambia el rango).
- Al hacer `union(0,7)` se mira el rango del 0 y el 7. Como son iguales se cuelga el primero (o sea el 0) del segundo (o sea el 7), tal como indica la Nota. Ahora sí que se modifica el rango, ya que es seguro que si unimos dos árboles iguales el rango aumenta en 1. Por eso aparece el -4.
- Al hacer `union(7,8)` miran los rangos correspondientes. Como 7 tiene menor rango (-2) que 8 (-4) entonces se cuelga el 7 del 8. No se modifica el Rango, ya que no cambia nada al colgar un árbol pequeño de uno grande.

	0	1	2	3	4	5	6	7	8	9
find(3)	-3	0	0	**0**	7	6	7	-3	-2	8
find(5)	-3	0	0	0	7	**7**	7	-3	-2	8
union(0, 7)	**7**	0	0	0	7	7	7	**-4**	-2	8
union(7, 8)	7	0	0	0	7	7	7	-4	**7**	8

EJERCICIO 2

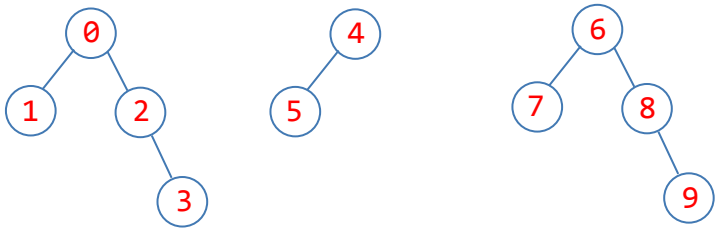
Sea la siguiente representación de un UF-Set:

0	1	2	3	4	5	6	7	8	9
-3	0	0	2	-2	4	-3	6	6	8

a) Dibujar el bosque de árboles que contiene.
b) Teniendo en cuenta que se implementa la fusión por rango y la compresión de caminos, indicar cómo irá evolucionando dicha representación tras la ejecución de las instrucciones de la siguiente tabla. **Nota:** al unir dos árboles con la misma altura (o rango), el primer árbol deberá colgar del segundo.

Solución

Similar a la del ejercicio anterior.



	0	1	2	3	4	5	6	7	8	9
find(9)	-3	0	0	2	-2	4	-3	6	6	**6**
union(4, 6)	-3	0	0	2	**6**	4	-3	6	6	6
union(0, 6)	**6**	0	0	2	6	4	**-4**	6	6	6

EJERCICIO 3

Diseña un método en la clase *ForestUFSet* que muestre por pantalla los identificadores de cada uno de los conjuntos que hay en el *UF-Set*.

Solución

Como en la representación del UF-Set el elemento identificador de la clase (la raíz) es el valor negativo de la altura estimada del árbol, entonces sólo hay que buscar los elementos cuyo contenido es negativo.

```
public void mostrarIdentificadores() {
  for (int i = 0; i < talla; i++)
    if (elArray[i] < 0)
      System.out.println("" + i);
}
```

EJERCICIO 4

Diseña un método en la clase *ForestUFSet* que muestre por pantalla los elementos del conjunto al que pertenece un elemento dado *x*.

Solución

Para resolver este problema lo primero es conocer la clase a que pertenece *x*. Luego hay que recorrer todos los elementos y preguntar si pertenecen a la misma clase de *x*.

```
public void mostrarConjunto(int x) {
  int raiz = find(x);
  for (int i = 0; i < talla; i++)
    if (find(i) == raiz)
      System.out.println("" + i);
}
```

EJERCICIO 5

Diseña un método en la clase *ForestUFSet* que devuelva el número de elementos que habría en el conjunto resultante de unir los conjuntos a los que pertenecen dos elementos dados *x* e *y*.

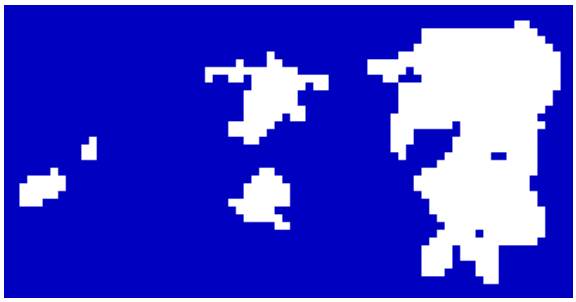
Solución

Es parecido al de antes. Se averigua la clase a la que pertenece x y la clase a la que pertenece y. Luego se exploran todos los elementos y si pertenecen a una de las dos clases se suman.

```
public void tallaUnion(int x, int y) {
  int raizX = find(x), raizY = find(y), n = 0;
  for (int i = 0; i < talla; i++) {
    int aux = find(i);
    if (aux == raizX || aux == raizY)
      n++;
  }
  return n;
}
```

EJERCICIO 6

Se dispone de una matriz de tipo *boolean* que representa un mapa de un archipiélago. Un valor *true* en una posición (x,y) indica que hay tierra en ese punto, mientras que un valor *false* indica que hay mar.



Ejemplo: Los valores *true* y *false* se representan mediante los colores blanco y azul, respectivamente.

Implementa un método con el siguiente perfil que permita calcular el número de islas que hay en el archipiélago:

```
public static int numIslas(boolean[][] mapa);
```

Solución

La idea es la siguiente: Se va a crear un UFset en que los elementos que pertenecen a una isla está en la misma clase. Una vez hecho esto sólo hay que contar cuantas clases hay.

- Para crear el UFset lo que se hace es inicializarlo, de modo que cada elemento pertenece a su propia clase `m=new ForestUFset (talla)`. El UFset asocia una elemento a cada casilla del mapa, pero como ha de ser un vector, lo que se hace a lo largo del algoritmo es "linealizar" la matriz, es decir, poner una fila detrás de otra y así tenemos un vector compuesto por la concatenación de las filas de la matriz.
- Durante el recorrido de la matriz del mapa (los dos bucles `for`) lo que se hace es comparar cada casilla (x,y) con la contigua de la derecha (x+1,y) y la de abajo (x,y+1), siempre comprobando que no nos salimos de la matriz. Lo que se analiza en cada casilla es que si mapa[x,y] es tierra y mapa[x+1,y] también es tierra (o al revés, ambos son agua) entonces se unen en la misma clase. Lo mismo con [x,y] y [x,y+1].
- Luego, si es tuviéramos dentro de la clase sólo habría que contar los elementos Raiz, como los problemas anteriores, pero como no tenemos acceso a la representación del UFset sólo podemos usar los métodos visibles. Por eso para calcular el número de islas hay que recorrer de nuevo toda la matriz, preguntar si estamos en una casilla Tierra, y si es así, averiguar cúal es si raíz. Si esa raíz es la primera vez que la vemos incrementamos el contador de inslas y la marcamos como visitada, ya que cuando vayamos a otra casilla de la misma isla no debemos incrementar el contador.

```
// Convierte las coordenadas del mapa en un numero entero
private static int posInt(int x, int y, int ancho) {
  return y * ancho + x;
}

// Une los conjuntos a los que pertenecen dos elementos dados
private static void unir(int x, int y, UFSet m) {
  int raizX = m.find(x);
  int raizY = m.find(y);
  if (raizX != raizY) m.merge(raizX, raizY);
}
```

```
public static int numIslas(boolean[][] mapa) {
  int ancho = mapa.length, alto = mapa[0].length;
  int talla = ancho * alto, res = 0;
  UFSet m = new ForestUFSet(talla);
  // Creamos las clases de equivalencia
  for (int x = 0; x < ancho; x++) {
    for (int y = 0; y < alto; y++) {
      if (x + 1 < ancho && mapa[x][y] == mapa[x + 1][y])
        unir(posInt(x, y, ancho), posInt(x + 1, y, ancho));
      if (y + 1 < alto && mapa[x][y] == mapa[x][y + 1])
        unir(posInt(x, y, ancho), posInt(x, y + 1, ancho));
    }
  }
  // Calculamos el número de islas (conjuntos de tierra)
  boolean[] visitado = new boolean[talla];
  for (int x = 0; x < ancho; x++) {
    for (int y = 0; y < alto; y++) {
      if (mapa[x][y]) {          // Tierra
        int raiz = m.find(posInt(x, y));
        if (!visitado[raiz]) {
          visitado[raiz] = true;
          res++;
        }
      }
    }
  }
  return res;
}
```