DAVID CARRERAS CASTAÑ, 1ºGEI

INFORME PRÁCTICA I

Cypher Encryption System

ÍNDICE

Función allPairs:

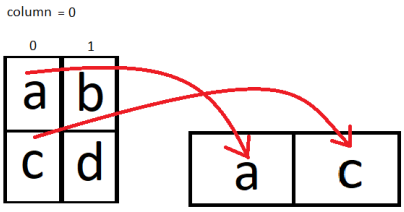
- En primer lugar he comprobado que el array no apunte a null para evitar errores y comprobaciones innecesarias.
- En cada una de las posiciones, he comprobado si alguna de las filas apunta a null. En caso de no apuntar a null compruebo que la fila tenga exactamente dos caracteres en cada columna. Si una de las dos condiciones se cumple tras haber hecho esta comprobación para cada una de las posiciones, significa que no todas las filas del array tienen una pareja de caracteres.

Otras posibilidades:

- Otra alternativa hubiese sido ir comprobando que cada columna tenga tantos caracteres como filas, pero quizás hubiese sido un poco menos intuitivo, ya que estamos comprobando parejas de carácteres.

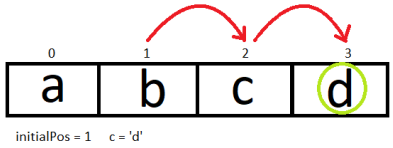
Función getColumn:

- Primeramente, si el número de filas es cero, directamente retornamos un array vacío, para evitar comprobaciones innecesarias que se puedan realizar posteriormente.
- Posición a posición copiamos del array 2d la columna que nos pasa como int la función y la copiamos en el array que retornaremos como resultado.



Función containsInSuffix:

- Para resolver este problema, compruebo que la posición inicial “initialPos” no sea mayor que la longitud total de los caracteres de chars, ya que esto significaría comprobar en una posición fuera del índice del array.
- Destacar que en el bucle en lugar de comenzar en la posición 0, empiezo a comprobar a partir de initialPos si el carácter almacenado en esa posición se corresponde con el “c”.




```
public boolean containsInSuffix(char[] chars, int initialPos, char c) {
        boolean containsInSuffix = false;
        if (chars.length > initialPos) {
            for (int i = initialPos; i < chars.length; i++) {
                if (c == chars[i]) {
```

```
                containsInSuffix = true;
            }
        }
    }
    return containsInSuffix; }
```

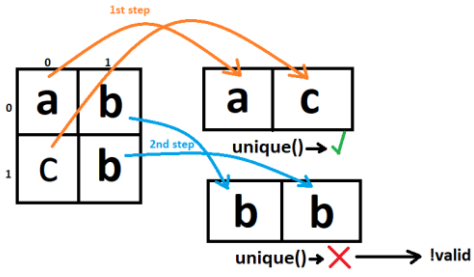
Otras posibilidades:
Una alternativa hubiese sido empezar el bucle desde la última posición hasta initialPos, habiendo comprobado antes que la longitud del array fuese mayor que initialPos.

Función unique:

- Este caso lo he resuelto pasándole a la función unique la posición siguiente (initialPos) a la posición en la que se encuentra el carácter buscado, de tal manera que con una sola llamada a unique ya sé si ese carácter se repite en el array chars.

Función isValid:

- En primer lugar, con la llamada a allPairs comprobamos que la clave de cifrado no sea null (está implementado en la función allPairs) y a su vez que cada fila del array contenga dos caracteres, y si esto no se cumple directamente key no es válida.
- Tras eso comprobaré que el número de filas no sea cero, ya que de lo contrario el proceso de comprobar que no haya caracteres repetidos en cada columna nos devolvería un error ya que no habría elementos que comprobar si están repetidos .
- Una vez comprobado eso, implemento un bucle anidado que itera primero las columnas y después las filas, donde he tenido que crear un array “column” porque la función unique requiere de un array de una dimensión y no de dos dimensiones como es el caso de “key”.
- Copio los caracteres de la primera columna en “column” para comprobar aprovechando la función unique si se repiten los caracteres en alguna columna.




```
public boolean isValid(char[][] key) {
    boolean valid = allPairs(key);
    if (valid && key.length != 0) {
        for (int j = 0; j < key[0].length; j++) {
            char[] column = new char[key[0].length];
```

```
            for (int i = 0; i < key.length; i++) {
                column[i] = key[i][j];
                if (!unique(column)) {
                    valid = false;
                }
            }
        }
```

Otras posibilidades:

Otra alternativa hubiese sido comprobar antes de si todas las filas tienen parejas de caracteres, si hay más filas que 0, ya que así nos evitaríamos comprobar antes las parejas de caracteres.

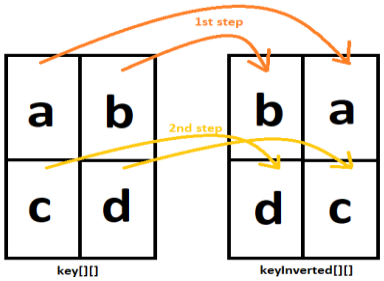
Tests que añadí:

Para comprobar que funcionaba con más columnas y con filas que solo contenían un carácter.

```
assertEquals("isValid9", false, isValid(new char[][]{{'a', 'b'}, {'c', 'd'},{'f'}}));
assertEquals("isValid10", false, isValid(new char[][]{{'a', 'b'}, {},{'c', 'd'},{'f'}}));
```

Función invert:

- En primer lugar, compruebo que las filas de key no sean 0, ya que si lo son no es posible invertir las columnas, y por tanto ya retorno un array vacío.
- Por consiguiente, lo que he hecho ha sido crear otro array 2d con las mismas dimensiones que key y con un bucle he copiado el carácter en la columna opuesta al original por filas hasta llegar a la longitud de filas final.



Función createKey:

- En primer lugar, compruebo cuál de las dos strings tiene el menor tamaño llamando a una función que he creado “minStrLength”, ya que con este sabemos el número de claves que vamos a crear.
- Si no hay ninguna clave directamente esquivo todo el proceso de creación de la clave y retorno un array 2d vacío, ya que no es posible crear la clave.
- Si finalmente no se ha dado la situación anterior, entonces voy a crear un bucle que va a iterar tantas veces como parejas hemos calculado que habrá “numberOfPairs”. Dentro de este bucle simplemente iré copiando los caracteres de la string de la izquierda en la columna izquierda del array 2d y los de la derecha en la columna 1.
- Recalcar que cuando creo el array 2d directamente le asigno dos columnas, una para el String left y otra para el String right.

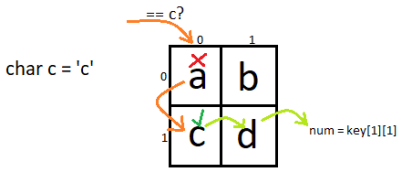
Otras posibilidades:
En lugar de crear una función alternativa para saber cual de las dos cadenas de caracteres tiene la menor longitud, podría haber usado el método Math.min que directamente realiza estos cálculos.

Tests que añadí:
Para comprobar que era capaz de crear keys de más de dos filas:

```
assertEquals("createKey5", new char[][]{{'a', 'c'}, {'b', 'd'},{'c','c'}},
createKey("abcd", "cdc"));
```

Función encodeChar:

- Para cada una de las filas miraremos si el carácter que queremos codificar se encuentra en el patrón para cifrar la clave (key[][]), y si es así retornaremos el valor “codificado” de la segunda columna.
- En caso de que tras recorrer todas las filas no encontrásemos codificación para el carácter buscado o que el número de filas fuese 0, retornaríamos directamente el valor -1.



Función encodeText:

- En esta función he creado un bucle que por tantos caracteres como tiene la String, llamo a la función que he explicado previamente para que me codifique el carácter con la clave key, y en caso de no encontrarse el carácter en key que me retorne -1. De esta forma evito volver a programar la codificación del carácter que sería repetir código innecesariamente.
- Tras esto, si finalmente es posible codificar el carácter realizo una conversión explícita a char del int que me devuelve la codificación del carácter y lo guardo en el array que finalmente retornare como el String que indica la cabecera de la función.

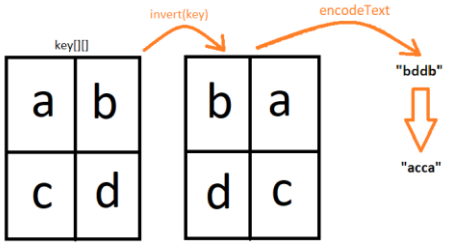
Otras posibilidades:

Una alternativa sería haber comprobado si algún carácter si repetía(con la función unique) no haría falta hacer todo el proceso de codificación, sino que simplemente copiando la codificación de la primera conversión de ese mismo carácter sería suficiente

Función decodeText:

- Como en esta función se exigía usar funciones ya creadas, me he limitado a eso y simplemente he usado la función invert para que el array de key me quedase justo al revés, y de esta forma poder llamar a la función encodeText para decodificarlo (ya que ahora la clave pasaría a estar al revés y por tanto en lugar de codificar, estaríamos haciendo el proceso inverso).

```
public String decodeText(char[][] key, String encodedText) {
	return new String(encodeText(invert(key), encodedText));
		}
```



Tests que añadí:

Para comprobar que funcionaba con keys de más columnas
assertEquals("decodeText3", "accacx", decodeText(new char[][]{{'a', 'b'}, {'c', 'd'},{'x','c'}}, "bddbdc"));

Conclusión y cosas que me han costado más:

De todo el proyecto, lo que más me ha costado ha sido arreglar las funciones y simplificar al máximo teniendo en cuenta que se tenía que hacer un código agradable de leer. Por otro lado, crear el informe ha sido otra de las partes que más me ha costado, pues resumir brevemente el funcionamiento de las funciones sin que se escape ningún detalle que pueda parecer relevante ha sido para mí una tarea costosa.