Desarrollo de Aplicaciones Multiplataforma
Examen de Programación
3ª Evaluación - 24 de mayo de 2023

Ejercicio 1 (5 puntos)

Una empresa de clonación se dedica a clonar animales en peligro de extinción de diferentes especies. Se pide crear un programa Java que clasifique secuencias de ADN de mejor a peor antes de llevar a cabo un proceso de clonado. En la entrada estándar se leerán varias secuencias correspondientes a una misma especie según las especificaciones siguientes:

- En la primera línea se recibe la longitud de las secuencias de ADN de la especie.
- En sucesivas líneas se reciben secuencias de diferentes ejemplares, a razón de una por línea. Una secuencia de ADN es una secuencia de unos y ceros separados por el carácter `'!'`. La cantidad conjunta de unos y ceros debe coincidir con la longitud especificada en la primera línea. En caso contrario, la secuencia leída será desechada mostrando un mensaje de error en la consola.
- En la última línea se recibe la cadena `"clasificar"`.

Para determinar si una secuencia es mejor que otra, se tendrán en cuenta los criterios siguientes:

1. Es mejor la que tenga la subsecuencia de unos de mayor longitud.
2. Si dos secuencias coinciden en el criterio anterior, se considerará mejor aquella en la que la subsecuencia esté situada más a la izquierda.
3. Si dos secuencias coinciden en los dos criterios anteriores, se considerará mejor aquella que contenga más unos.

Para resolver el problema, cada secuencia de ADN se representará como un objeto de la clase `SecuenciaADN` con los atributos y métodos siguientes:

- Array de caracteres que almacena la secuencia de unos y ceros (sin el carácter `'!'`).
- Longitud de la mayor subsecuencia de unos.
- Índice de comienzo de la mayor subsecuencia de unos situada más a la izquierda.
- Cantidad de unos.
- Constructor que recibe en un `String` una línea leída de la entrada estándar que contiene una secuencia de ADN y la procesa para inicializar los atributos de instancia.
- Setters y/o Getters.
- Redefinición de los métodos heredados que corresponda.

El programa leerá cada secuencia de ADN, la representará como un objeto de tipo `SecuenciaADN` y la almacenará en un `TreeSet`. Dentro de esta colección, las secuencias se ordenarán de mejor a peor según el orden natural que se defina en la clase `SecuenciaADN`.

Una vez leídas y almacenadas todas las secuencias de ADN y recibida la orden `"clasificar"`, se mostrarán por pantalla ordenadas de mejor a peor, cada una en una línea con el formato que se muestra en los ejemplos siguientes:

ENTRADA	SALIDA
5 1!0!1!1!0 0!1!1!0!0 clasificar	[0, 1, 1, 0, 0], subsecuencia=2, índice=1, unos=2 [1, 0, 1, 1, 0], subsecuencia=2, índice=2, unos=3
4 1!1!0!1 1!0!0!1 1!1!0!0 clasificar	[1, 1, 0, 1], subsecuencia=2, índice=0, unos=3 [1, 1, 0, 0], subsecuencia=2, índice=0, unos=2 [1, 0, 0, 1], subsecuencia=1, índice=0, unos=2

Ejercicio 2 (2 puntos)

Escribe en una clase llamada `Ejercicio2` un método estático llamado `explosionEnCadena` que reciba una cadena y retorne una versión transformada de la misma según las especificaciones siguientes:

- La cadena original contendrá únicamente caracteres del alfabeto latino (de la 'a' a la 'z'), dígitos del '0' al '9' y el símbolo '>' que llamaremos explosión.
- Cada explosión siempre va seguida de al menos un dígito.
- El dígito que sigue a una explosión (si hay varios, sólo el primero) indica un número de caracteres después del símbolo '>' que se eliminarán de la cadena original (incluido el propio dígito).
- Si se encuentra otra explosión antes de la eliminación del total de caracteres especificado en una explosión anterior, el resto se acumulará a la nueva explosión.
- El carácter explosión nunca se elimina.

Pon a prueba el método `explosionEnCadena` incluyendo en la clase Ejercicio2 el método `main` siguiente (el programa finaliza pulsando ctrl+z):

```
public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      while (scanner.hasNext())
         System.out.println(explosionEnCadena(scanner.next()));
      scanner.close();
}
```

Ejemplos:

ENTRADA	SALIDA
abv>1>1>2>2asdasd	abv>>>>dasd
pepe>2ses>1un>2xprogramador>4C++Java	pepe>es>un>programador>Java

Ejercicio 3 (3 puntos)

Un administrador de sistemas necesita ayuda para gestionar la seguridad de sus servidores. Se trata de localizar a los autores de una nueva amenaza que se basa en inundar el servidor con mensajes. Estos usuarios son difíciles de detectar ya que cambian su dirección IP con regularidad.

Se pide realizar un programa Java que rastree los registros almacenados en los servidores en busca de este tipo de usuarios. Para ello se usará un mapa (`Map<String, Map<String, Integer>>`) para almacenar, por cada usuario, todas las direcciones IP desde las que ha enviado mensajes y la cantidad de mensajes enviados desde cada una. Los registros del servidor se recibirán a través de la entrada estándar, cada uno en una línea de texto con el formato:

```
IP=dirección_IP message='texto del mensaje' user=nombre_usuario
```

Se usará una expresión regular para extraer los datos significativos de cada registro, teniendo en cuenta que el texto del mensaje se encuentra entre comillas simples y ni la IP ni el nombre de usuario contendrán espacios en blanco.

Cuando se reciba en la entrada estándar una línea con el texto "fin", se mostrará en la consola la información de cada usuario en orden alfabético con el formato siguiente:

```
nombre_usuario: número_de_ips / número_total_de_mensajes
IP1 -> número_de_mensajes
IP2 -> número_de_mensajes
       ⋮
```

Ejemplo:

ENTRADA	SALIDA
IP=151.23.130.56 message='Hey son' user=mother IP=151.23.156.40 message='Hi mom!' user=child0 IP=151.23.156.41 message='Hi from me too' user=child1 IP=192.23.30.42 message='spam' user=destroyer IP=192.23.35.37 message='spam' user=destroyer IP=192.23.50.40 message='' user=yetAnotherUsername IP=192.23.50.40 message='comment' user=yetAnotherUsername IP=192.23.35.37 message='spam' usuario=destroyer IP=197.111.155.99 message='Hello' user=unknown end	child0: 1 / 1 151.23.156.40 -> 1 child1: 1 / 1 151.23.156.41 -> 1 destroyer: 2 / 3 192.23.30.42 -> 1 192.23.35.37 -> 2 mother: 1 / 1 ⋮