Prácticas 2 y 4

Programando en Java

1. Objetivos

El objetivo de estas prácticas es familiarizarse con la programación en Java, adquiriendo las habilidades mínimas necesarias para hacer programas que procesen datos de entrada recibidos por la entrada estándar o mediante la lectura de ficheros, realicen cálculos y muestren resultados por la salida estándar o los guarden en ficheros. Para ello se propone realizar programas que permitan calcular fórmulas que resultarán útiles como herramienta en la resolución de ejercicios y problemas, así como llevar a cabo procesamientos sencillos de datos para el análisis de tráfico en redes.

Estas sesiones de prácticas no se evalúan, por tanto debes aprovechar el tiempo para afianzar los conocimientos de programación en Java impartidos en las clases de teoría y consultar todas aquellas dudas que puedan surgirte.

2. Calculadora de la función B de Erlang

La función B de Erlang se utiliza en un sistema de conmutación telefónica con *n* canales que lo unen a otro conmutador para calcular la probabilidad de que una llamada encuentre todas las líneas ocupadas. El significado de sus parámetros, así como su uso para resolver problemas se explicarán en clases de teoría. En esta práctica nos centraremos sólo en calcular su valor.

La función B de Erlang o *B(a,k)* se define como:

$$B(a,k) = \frac{\frac{a^k}{k!}}{\sum_{i=0}^{k} \frac{a^i}{i!}}$$

Los parámetros de entrada son lo que llamaremos la intensidad de tráfico y que en la fórmula siguiente se representa como *a*, que es un numero real entre 0 e infinito; y el número de canales disponibles que llamaremos *k* y será un número entero mayor que 0.

2.1 Especificación

Realiza un programa en Java que cumpla con la siguiente especificación:

Uso:

```
java ErlangB <a> <k>
```

Descripción:

Calcula la función *B(a,k)* imprimiendo el resultado por pantalla.

Los parámetros `<a>` y `<k>` son obligatorios. En caso de que no se indiquen el programa imprimirá por pantalla un error y terminará la ejecución del mismo.

El resultado será una única línea de texto con un número real que represente el valor esperado. Debe tener al menos 4 decimales

Ejemplos:

```
$ java ErlangB 3.0 4
0.206107

$ java ErlangB 100.0 15
0.851718
```

2.2. Pruebas y depuración

Se pretende que esta herramienta pueda utilizarse en la resolución de problemas del tema 5. Para ello verifica que funciona en el rango de variables de interés, para 1, 5, 10 y 20 canales, al aumentar la carga debes obtener mayores probabilidades de bloqueo. Comprueba al menos que funcionan todos estos antes de continuar:

```
B(0.25,1)= 0.200000     B(0.5,1)= 0.333333      B(0.75,1)= 0.428571
B(1.0,1)= 0.500000      B(1.25,1)= 0.555556

B(1.25,5)= 0.007300     B(2.5,5)= 0.069731      B(3.75,5)= 0.176617
B(5.0,5)= 0.284868      B(6.25,5)= 0.377503

B(2.5,10)= 0.000216     B(5.0,10)= 0.018385     B(7.5,10)= 0.099544
B(10.0,10)= 0.214582    B(12.5,10)= 0.321951

B(5.0,20)= 0.000000     B(10.0,20)= 0.001869    B(15.0,20)= 0.045593
B(20.0,20)= 0.158892    B(25.0,20)= 0.279890
```

Prueba también para valores de *k* mayores donde es más probable que la función tenga problemas. Utiliza tu programa para calcular:

```
B(10.0,21)= 0.000889    B(20.0,21)= 0.131436    B(20.0,22)= 0.106734

B(20.0,23)= 0.084930    B(100.0,21)= 0.792577   B(150.0,100)= 0.345373
```

Si observas algún problema, ten en cuenta que la función *B(a,k)* tal y como está definida sólo puede devolver números entre 0 y 1 (son probabilidades). Si te fijas en la fórmula, para que devuelva un número negativo o mayor que 1 es que algo está mal hecho, porque el numerador siempre es menor que el denominador y todo es positivo. Así que si esto ocurre, depura el programa y descubre qué es lo que sucede.

2.3. Eligiendo el valor óptimo de *k* en la B de Erlang

En ocasiones, necesitarás obtener el número de canales *k* para una intensidad de tráfico *a* y que la probabilidad de bloqueo *p* sea cierto valor deseado. Para eso necesitaremos calcular el inverso de la función *B(a,k)*, es decir dado un valor *p* y otro de *a*, obtener el *k* que cumple que *B(a,k)=p*. Esto se puede hacer sabiendo que la función *B(a,k)* es decreciente en *k*, es decir, *B(a,k) > B(a,k+1)*.

Se pide realizar un programa que a partir de un valor de *a* y un valor de *p* objetivo encuentre el menor valor de *k* que cumpla que *B(a,k) < p*.

Uso:

```
java ErlangBFindK <a> <p>
```

Descripción:

Calcula el primer valor de *k* que cumple que *B(a,k) < p*

La salida será una línea mostrando el valor de *B(a,k)* menor que `<p>`, con el formato que se ve en los ejemplos.

Ejemplos:

```
$ java ErlangBFindK 7.0 0.01
B(7.000000,14)= 0.007135

$ java ErlangBFindK 15.0 0.05
B(15.000000,20)= 0.045593
```

2.4. Graficando la B de Erlang

Para dibujar la B de Erlang queremos calcular valores de un rango de parámetros de entrada e imprimirlos en un fichero. Construye un nuevo programa con la siguiente funcionalidad:

Uso:

```
java ErlangBPlot <k> <ai> <af> <na> <file>
```

Descripción:

Calcula la función *B(a,k)* en un rango de valores de `<a>` desde `<ai>` hasta `<af>` con `<na>` valores intermedios. Escribe los resultados por pantalla y en el fichero `<file>` como filas de tres valores separados en cada fila por un espacio. Los tres valores serán respectivamente: *a k B(a,k)*

Ejemplo:

```
$ java ErlangBPlot 4 0.0 6.0 10 values
0.000000 4 0.000000
0.600000 4 0.002965
1.200000 4 0.026226
1.800000 4 0.075033
2.400000 4 0.138706
3.000000 4 0.206107
3.600000 4 0.270685
4.200000 4 0.329628
4.800000 4 0.382206
5.400000 4 0.428650
6.000000 4 0.469565

$ cat values
0.000000 4 0.000000
0.600000 4 0.002965
1.200000 4 0.026226
1.800000 4 0.075033
2.400000 4 0.138706
3.000000 4 0.206107
3.600000 4 0.270685
4.200000 4 0.329628
4.800000 4 0.382206
5.400000 4 0.428650
6.000000 4 0.469565
```

Utiliza esto para graficar la función B de Erlang, por ejemplo usando el programa `gnuplot` de la siguiente manera:

```
$ gnuplot
gnuplot> plot "values" using 1:3 title "ErlangB" with linespoints ps 3
```

3. Procesando la salida de una captura

El objetivo es utilizar la programación en Java para analizar los resultados obtenidos con los programas `Wireshark/tshark` y ser capaz de realizar medidas a partir de una captura, programando tu propia herramienta que obtenga información sobre una traza de tráfico de red.

Aprende a utilizar el comando `tshark` para obtener un fichero de texto con los tiempos y tamaños de las tramas Ethernet que se hayan visto en la red de su ordenador. Por ejemplo, ejecuta este comando:

```
$ tshark -i eth0
```

Verás los paquetes que se transmiten por la Ethernet, aquellos que puede leer el interfaz `eth0`. El formato con el que se imprimen en pantalla sin embargo es un poco difícil de leer para un programa. Queremos usar el comando `tshark` para que proporcione una lista de las tramas Ethernet que se han visto en la red pero de una forma mas fácil de procesar. Vamos a usar una serie de opciones al llamar a `tshark`:

- `-T` nos permite decir que sólo queremos que imprima algunos campos
- `-E` nos deja especificar opciones, por ejemplo que los campos estén separados por espacio
- `-e` vamos eligiendo los campos que queremos que aparezcan

Por ejemplo, podemos mostrar el tiempo en el que se envió y el tamaño de cada paquete, separados por espacios con el siguiente comando:

```
$ tshark -i eth0 -T fields -E separator=/s -e frame.time_relative -e
frame.len
Capturing on eth0
0.000000000 74
0.000146000 74
0.000174000 66
...
```

Si queremos almacenar los resultados para utilizarlos en la siguiente fase puedes dirigir la salida a un fichero:

```
$ tshark -i eth0 -T fields -E separator=/s -e frame.time_relative -e
frame.len > captura.txt
```

Para detener la captura use *Control-C* o bien puedes indicar cómo especificar un número máximo de paquetes capturados (busca en la ayuda de `tshark` esta opción).

También puede interesar guardar la captura completa para poder estudiarla con `Wireshark`. En ese caso los datos de interés los puedes sacar directamente de la captura:

```
$ tshark -i eth0 -w captura.pcap
Capturing on eth0

(detener captura con Control-C o especificando número de paquetes)

$

$ tshark -r captura.pcap -T fields -E separator=/s -e frame.time_relative
-e frame.len > captura.txt
```

3.1. Lectura de tamaños

Realiza un programa que lea los tamaños de las tramas del fichero de captura (segunda columna) y realice operaciones con ellos. Construye un nuevo programa con la siguiente funcionalidad:

Uso:

```
java ReadSizes <nombrecaptura> <numeroreferencia>
```

Descripción:

Lee el fichero `<nombrecaptura>` que contiene en cada línea el tiempo en el que se ha realizado la captura (número decimal positivo, primera columna) y el tamaño de la trama (número entero positivo, segunda columna) separados por un espacio, la analizará y mostrará una línea como la siguiente (indicando la opción adecuada):

```
Leído tamaño trama [x] es [menor|mayor|igual] que [numeroreferencia]
```

Ejemplo:

```
$ cat captura.txt
0.000000000 74
0.000146000 74
0.000174000 66
0.000178000 128
0.000184000 74
0.000189000 256
...

$ java ReadSizes captura.txt 128
Leido tamaño trama [74] es menor que [128]
Leido tamaño trama [74] es menor que [128]
Leido tamaño trama [66] es menor que [128]
Leido tamaño trama [128] es igual que [128]
Leido tamaño trama [74] es menor que [128]
Leido tamaño trama [256] es mayor que [128]
...
```

3.2. Calculando el histograma de los tamaños

En esta parte se pretende realizar un programa que lea un fichero de captura como el anterior, que en la segunda columna contiene el tamaño de las tramas. Se acepta que el programa acabe con un error si hay algún número negativo o algo que no sea un número en la segunda columna. El programa proporcionará el histograma analizando cuántas veces aparece cada tamaño en el fichero. El programa aceptará como argumento un número máximo a partir del cual no nos interesa saber las repeticiones individuales. El funcionamiento del programa debe cumplir la siguiente especificación:

Uso:

```
java SizeHist <nombrecaptura> <maximonumero>
```

Descripción:

Calcula el histograma de los tamaños de las tramas de la captura indicada por `<nombrecaptura>`. El histograma es el número de repeticiones de cada número individual. Solo nos interesa saber las repeticiones de los números menores que `<maximonumero>`. Para los valores que no están dentro del rango nos vale con saber el número global. El fichero de captura puede tener potencialmente un número de líneas ilimitado. La salida será una línea por cada número de interés indicando el número y la cantidad de apariciones en el fichero. El formato de la salida debe ser el que se ve en el ejemplo.

Ejemplos:

```
$ cat captura.txt
0.000000000 74
0.000146000 74
0.000174000 66
0.000178000 128
0.000184000 74
0.000189000 256

$ java SizeHist captura.txt 128
0 : 0 veces
1 : 0 veces
...
65 : 0 veces
66 : 1 veces
67 : 0 veces
...
74 : 3 veces
...
128 : 1 veces
# otros : 1 veces
# total : 6
```

Para calcular el histograma no es aceptable leer todos los números del fichero y después hacer los cálculos, ya que el fichero podría tener millones de números. Se trata de leer cada línea, utilizar la información de esa línea y olvidarse de ese número. Es un requisito del programa que sea capaz de analizar un fichero de 1.000.000 de líneas. Si intenta leerlos todos fallará.

Se recomienda utilizar el número de referencia para decidir cuántos contadores necesitamos, crear los contadores necesarios y para cada número leído actualizar los contadores correspondientes. Una vez leídos todos los números se imprimirá el resultado.

Puede redirigir la salida de su programa a un fichero y hacer el gráfico del histograma mediante `gnuplot` de la siguiente manera:

```
$ java SizeHist captura.txt 128 > hist

$ gnuplot
gnuplot> set xlabel "tamaños (B)"
gnuplot> set ylabel "número de veces"
gnuplot> plot "hist" using 1:3 title "histograma" with impulses
```

3.3. Analizando el throughput

En esta parte se pretende estudiar el throughput de la captura. Realiza un programa que lea un fichero de captura como el anterior y muestre la cantidad total de bytes que ve en cada intervalo de tiempo de duración indicada:

Uso:

```
java TimeThr <nombrecaptura> <duración del intervalo>
```

Descripción:

Lee el fichero `<nombrecaptura>` que contiene en cada línea la información de una trama Ethernet observada en la red por orden de aparición. Cada línea tendrá el formato:

```
<tiempo> <tamaño de la trama Ethernet>
```

El programa agrupará las tramas que aparezcan en cada intervalo de duración indicada mostrando para cada intervalo que pase: el tiempo final del intervalo, la cantidad de bytes observada en el mismo, la cantidad de bytes totales observada hasta ese momento y el throughput en el intervalo (en bps).

Ejemplo:

```
$ cat captura.txt
0.000000000 60
0.006421000 60
0.010704000 124
0.187527000 60
0.201154000 60
0.261155000 60
```

```
0.261158000 60
0.353060000 149
0.392433000 221
0.392436000 221
0.421082000 60
0.446327000 60
0.479638000 60
0.492356000 60
0.588422000 92
...

$ java TimeThr captura.txt 0.1
0.100 244 244 19520.0
0.200 60 304 4800.0
0.300 180 484 14400.0
0.400 591 1075 47280.0
0.500 240 1315 19200.0
...
```

Asegúrate de que el programa funciona correctamente. Para ello comprueba el funcionamiento comparando los resultados obtenidos con tu programa y con los resultados que obtiene `wireshark` analizando el mismo fichero de captura .pcap.

Para dibujar los ficheros que estás haciendo utiliza la utilidad `gnuplot`. Abre un terminal y ve al directorio donde está el fichero de resultados. Utiliza los siguientes comandos:

```
$ java TimeThr captura.txt 0.1 > dat   # calculamos los throughputs en dat

$ gnuplot
gnuplot> set xlabel "tiempo (s)"
gnuplot> set ylabel "bytes recibidos"
gnuplot> plot "dat" using 1:2 title "Th" with lines
```

O puedes dibujar el tráfico acumulado con la tercera columna

```
gnuplot> plot "dat" using 1:3 title "Th" with lines
```

3.4. Filtrando el tráfico

A continuación modifica la captura de `tshark` para que en cada línea añada dos nuevas columnas al final con el Ethertype y la dirección MAC destino de cada trama. Amplía tu programa `TimeThr` para que acepte dos nuevos argumentos opcionales tras los anteriores:

Uso:

```
java TimeThrv2 <nombrecaptura> <duración del intervalo> [ethertype] [MACdst]
```

Descripción:

Lee el fichero `<nombrecaptura>` con líneas que contienen la información de una trama Ethernet observada en la red por orden de aparición. Cada línea tendrá el formato:

```
<tiempo> <tamaño de la trama Ethernet> <ethertype> <MACdestino>
```

El programa agrupará las tramas que aparezcan en cada intervalo de duración indicada mostrando para cada intervalo que pase: el tiempo final del intervalo, la cantidad de bytes observada en el mismo, la cantidad de bytes totales observada hasta ese momento y el throughput en el intervalo (en bps).

El parámetro `[ethertype]` es opcional así que puede no existir, en caso de indicarse quiere decir que solo se contarán aquellas tramas con ese valor de Ethertype.

En los casos en que exista se puede indicar * para decir "cualquier Ethertype".

El parámetro `[MACdst]` sólo puede existir si existe también `[ethertype]` e indica que se cuenten solo aquellas tramas que se dirijan a esa dirección MAC.

Comprueba el funcionamiento del programa haciendo una captura durante la cual haga tráfico con diferentes direcciones MAC (por ejemplo `ping` a máquinas del laboratorio) y comparando las series temporales de tu programa filtrando por diversas direcciones MAC con las que le construye `wireshark`.