DISEÑO DE UN SIMULADOR POR EVENTOS DISCRETOS USANDO EL LENGUAJE DE PROPOSITO GENERAL PASCAL

En esta sección veremos con detalles los distintos elementos de un simulador por eventos discretos, los cuales son completamente transparentes al usuario cuando se usan lenguajes de simulación. Se construirá un programa que permite simular una red con nodos o taquillas de capacidad múltiple[1] y tiempos entre llegadas y de servicio exponenciales. La red puede ser un sistema abierto (pueden haber llegadas del exterior) y por lo tanto se definen dos tipos de nodos. Los nodos tipo *llegada* tienen dos parámetros: el tiempo promedio entre llegadas de los clientes que llegan del exterior y el tiempo promedio de servicio. Los nodos tipo *otro* solo necesitan del tiempo promedio de servicio ya que los clientes que reciben llegan de otros nodos y no del exterior. Cada nodo también necesita información sobre su capacidad, de sus nodos sucesores (a los cuales los clientes se pueden dirigir al salir del nodo) y la distribución de probabilidad respectiva. Si un nodo no tiene sucesores entonces los clientes salientes de este nodo abandonan el sistema. Las colas que se pueden formar en cada nodo siguen una disciplina FIFO (first in first out - primero en llegar primero en salir). A continuación se da un ejemplo de una red.

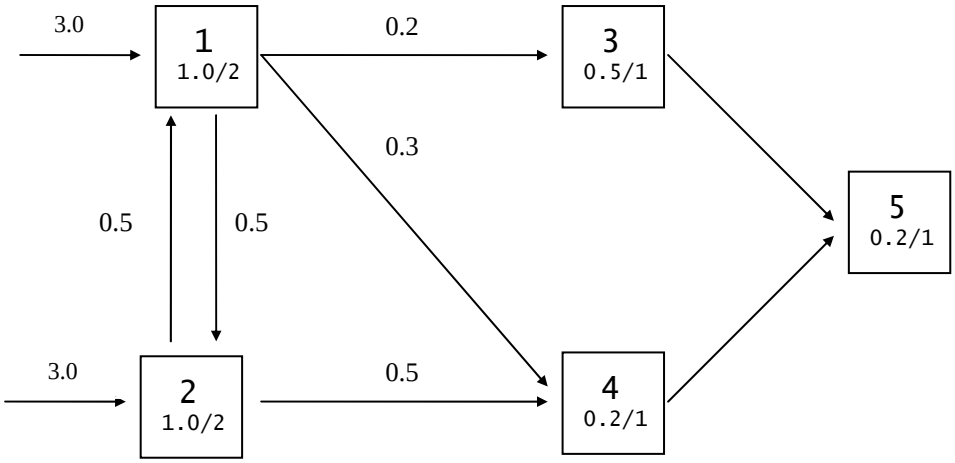


Figura 1. Ejemplo de una red.

La Figura 1 muestra una red con cinco nodos (numerados 1,2,...,5) con tiempos de servicio 1.0, 1.0, 0.5, 0.2, 0.2 y capacidades 2, 2, 1, 1, 1 respectivamente. Los nodos 1 y 2 son de *llegada* ya que reciben clientes provenientes del exterior a razón de 1 cada 3.0 unidades de tiempo[2]. Los nodos 3,4, y 5 son de tipo *otro*. El nodo 1 tiene como nodos sucesores el 2, 3, y 4 con probabilidades 0.5, 0.2 y 0.3 respectivamente. Para el nodos 2 los sucesores son el nodo 1 y 4 con la misma probabilidad (0.5). Los nodos 3 y 4 tienen como

[1] Pueden atender mas de un cliente en un momento dado.

[2] Las unidades dependen del sistema. Pueden ser segundos, minutos, horas, etc.

sucesor el nodo 5 con probabilidad 1. El nodo 5 no tiene sucesores y los clientes que completan servicio en este nodo abandonan el sistema.

La definición de la red a simular y el tiempo de simulación se harán usando un archivo. Esto evita el tedioso proceso de tener que dar la definición de la red manualmente cada vez que se desea simular. Además cualquier cambio que se quiera hacer en esta definición se puede hacer fácilmente editando el archivo.

Para estas redes existen básicamente dos tipos de eventos: *llegadas* y *salidas*. Como un nodo en particular puede tener tanto *llegadas externas* como *internas* (provenientes de otros nodos), también se hace necesario hacer esta distinción. Una llegada externa implica la generación de una próxima llegada externa, mientras que una llegada interna no involucra la generación de una próxima llegada. Por lo tanto, un evento debe contener la siguiente información: tiempo de ocurrencia, nodo de ocurrencia y el tipo de evento (*llegada*: *interna o externa*, *salida*).

En términos generales la dinámica del el simulador es la siguiente (básicamente el `lazo` descrito en I-14):

Leer archivos	
Inicializar el manejador de eventos	
Inicializar variables - Entre otras hacer *Tiempo_simulado* = 0	
Obtener el próximo evento a simular y actualizar *Tiempo_simulado*	en este caso el primer evento
Repita mientras *Tiempo_simulado* ≤ *Tiempo_de_simulación*	comienzo del lazo
Para el nodo en donde ocurre el evento haga	
Si el evento es una llegada entonces	
Si *utilización* = *capacidad* entonces	nodo sin disponibilidad
incrementa en uno el número de clientes en cola	
de lo contrario	cliente atendido de inmediato
genera la salida del cliente	
incrementa en uno la *utilización*	
Si la llegada fue externa entonces	
genera la próxima llegada externa	
de lo contrario	no es llegada sino salida
Si hay nodos sucesores entonces	si no, abandona el sistema
determinar a que nodo ira el cliente y generar dicha llegada	
Si hay clientes en cola entonces	se fue uno, entra otro
decrementar en uno el número de clientes en cola	
generar salida del cliente	
de lo contrario	sin cola – queda un cupo
decrementa en uno la *utilización*	
Obtener el próximo evento a simular y actualizar *Tiempo_simulado*	busca próximo evento
Fin del repita mientras	fin del lazo
Imprimir estadísticas	

Figura 2. El simulador.

I. EL MANEJADOR DE EVENTOS

Se usara una lista encadenada simple para implementar el manejador de eventos, la cual estará ordenada cronológicamente por el tiempo de ocurrencia de los eventos (ascendentemente). Esta lista la llamaremos *Lista de Eventos Futuros* (LEP) ya que consta de eventos que han sido programados pero que no han ocurrido. Cada evento estará representado por un registro que incluye 4 campos: tiempo de ocurrencia, nodo de ocurrencia, tipo de evento y apuntador al siguiente evento (ver Figura 3).

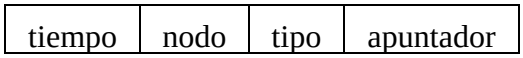


Figura 3. Representación de un evento.

Específicamente, la definición en Pascal de un registro que representa un evento esta dada por:

```
type
  tipo_ = (Llegada_i, Llegada_e, Salida, Ultimo);
  apt_evento = ^evento_;
  evento_ = record
    tiempo : real;
    nodo : integer;
    tipo : tipo_;
    proximo : apt_evento;
  end;
```

Figura 4. Definición en Pascal de un registro de evento.

Con el fin de facilitar la inserción de nuevos eventos en la LEP, se usaran dos eventos (eventos marca) que marcan el comienzo y el fin de la lista. Estos son eventos artificiales que no corresponden a actividades en la red y solo evitan tener que considerar como casos especiales la inserción de un evento al comienzo o al final de la lista (ver Figura 5). El evento marca al comienzo (*Primero*) no requiere inicialización de sus campos excepto por el apuntador. El evento marca final (*Ultimo*) tiene como tiempo de ocurrencia el máximo valor real permitido (ya que nunca debe ocurrir) y el apuntador se inicializa con *nil*. Los campos *nodo* y *tipo* en realidad no requieren inicialización pero se colocan en 0 y *Ultimo* para fines de claridad. Cuando se define la red, la numeración de los nodos comienza desde 1.

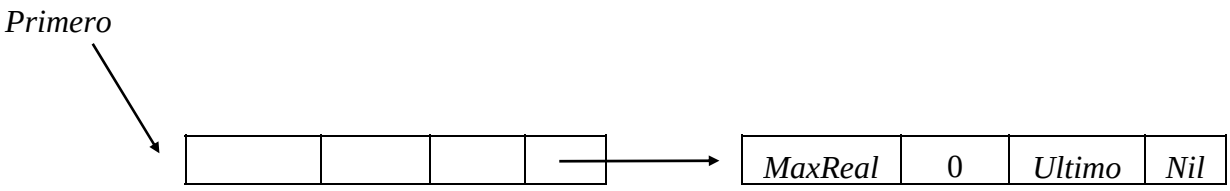


Figura 5. Inicialmente la LEP no tiene eventos reales y esta formada solo por los dos eventos marca.

Es importante destacar que hay otras formas de implementar las listas encadenadas simples y que también hay otras opciones para implementar el manejador de eventos como por ejemplo: arboles splay, arboles binarios, heaps, etc., que pueden ser mas eficientes.

II. RUTINAS DE MANEJO DE LA LEP

1. Inicialización

Esta rutina genera los dos eventos marca y también los primeros eventos (llegadas). Las variables *Primero* y *evento* son apuntadores a registros *tipo evento* y estan definidas como variables globales. *Max_Real* esta definido como una constante con valor $1.7x10^{38}$, que es el máximo valor real en Turbo Pascal 6.0.

```
procedure inicia_LEP;
  begin
    new(evento);                  { crea el evento que marca el fin de la LEP }
    with evento^ do begin         { con tiempo muy grande(∞)nunca sera procesado }
      tiempo := Max_Real;
      nodo := 0;
      tipo := Ultimo;
      proximo := nil;
    end;
    new(primero);                 { crea apuntador al comienzo de la LEP y   }
    primero^.proximo := evento;   { se encadena con el evento marca final    }
  end;
```

Figura 6. Rutina de inicializacion de la LEP.

2. Inserción

Cada vez que se genera un evento, este debe ser insertado cronológicamente en la LEP. Para esto se utiliza la siguiente rutina:

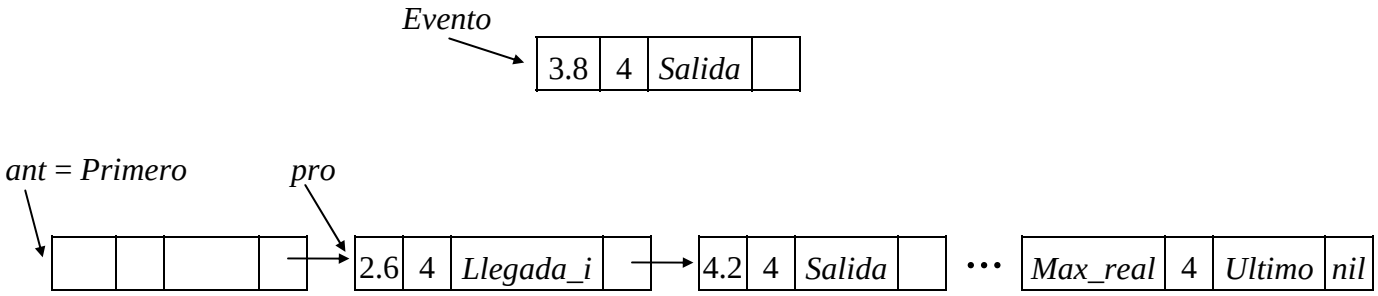
```
procedure inserta(var evento : apt_evento);
  var ant, pro : apt_evento;
  begin
    ant := primero;
    pro := primero^.proximo;
    while evento^.tiempo >= pro^.tiempo do begin
      ant := pro;
      pro := pro^.proximo;
    end;
    ant^.proximo := evento;
    evento^.proximo := pro;
  end;
```

Figura 7. Rutina para insertar eventos en la LEP.

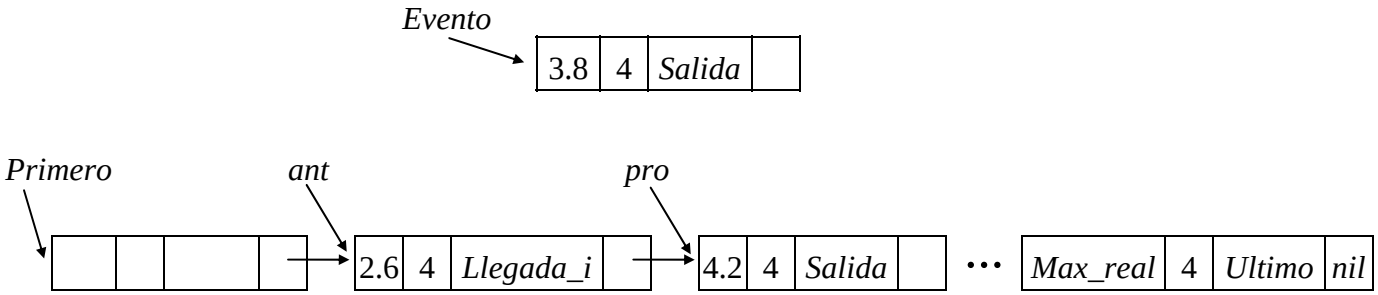
La rutina *inserta* usa dos apuntadores (*ant* y *pro*) los cuales inicialmente apuntan al primer y segundo registro de la LEP respectivamente. Recordemos que el primer registro de la LEP es un evento marca y que

todo evento que se inserte en la LEP debe ir ubicado entre los dos eventos marca. Estos dos apuntadores se van desplazando hasta encontrar el sitio en donde va el nuevo evento. Una vez localizada la ubicación del nuevo evento, se actualizan los apuntadores respectivamente (ver Figura 8).

a) Inicialmente:



b) Avanza *ant* y *pro:*



c) Localizada ubicación del evento a insertar - actualiza apuntadores:

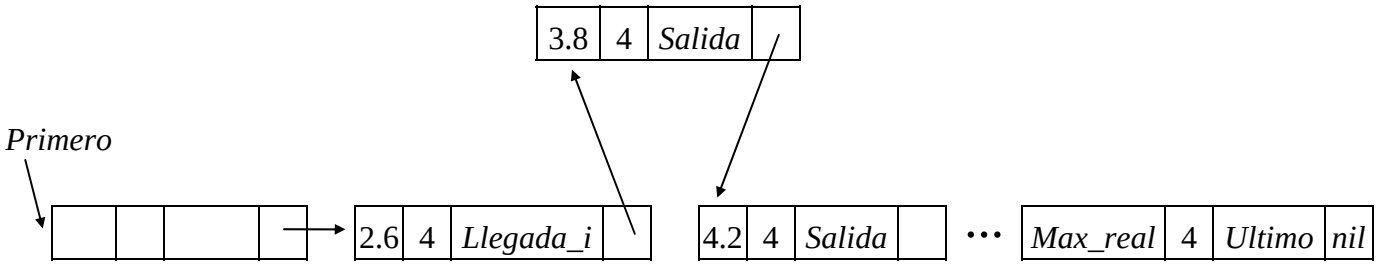


Figura 8. Ejemplo de la inserción de un evento en la LEP.

3. Obtención del próximo evento a simular.

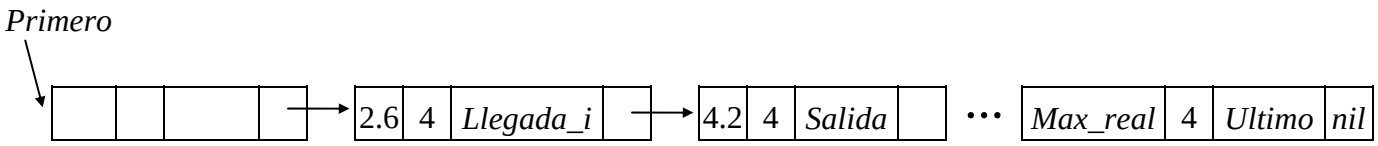
Después de simular un evento, hay que obtener el próximo evento a simular. Este es simplemente el primer evento real en la LEP. Para conseguir este evento usaremos una función que simplemente retorna el apuntador al primer evento real y actualiza el apuntador del evento marca al comienzo de la lista. Nótese que esta función también actualiza el tiempo simulado (ver Figura 9).

```
function prox_evento : apt_evento;
  var evento : apt_evento;
  begin
    evento := primero^.proximo;
    primero^.proximo := evento^.proximo;
    prox_evento := evento;
    tiempo_sim := evento^.tiempo;
  end;
```

Figura 9. Obtencion del proximo evento a simular.

A continuación se da un ejemplo de como funciona esta rutina:

a) Inicialmente:



b) Al retornar de la función:

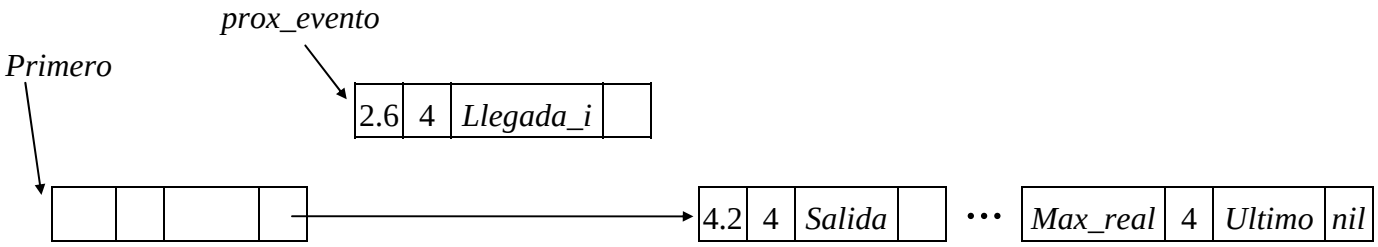


Figura 10. Ejemplo de como trabaja la función *prox_evento*.

III. CALCULO DEL PROXIMO NODO A VISITAR.

Cuando un cliente finaliza el servicio en un nodo y este nodo tiene varios sucesores, hay que determinar cual de los sucesores visitara seguidamente. Para cada nodo de la red se define dos arreglos: uno que indica cuales son los sucesores y el otro contiene las probabilidades respectivas (ver Figura 11).

Las constantes *Max_nodos* y *Max_sucesores* limitan el número de nodos que puede tener la red y el número de sucesores por nodos. Estas constantes son necesarias ya que la red se define como un arreglo de nodos y los sucesores como un arreglo de enteros. Estas constantes se pueden adaptar de acuerdo a las necesidades.

```
const Max_nodos = 20;     { Maximo número de nodos en la red    }
      Max_sucesores = 5;  { Maximo número de sucesores por nodo }
nodo_ = record
  { otras variables que definen el tipo de nodo
    y para almacenar estadisticas }
  suc : array[1..Max_sucesores] of integer;
  prob_suc : array[1..Max_sucesores] of real;
end;
red_ = record
  num_nodos : integer;
  nodos : array[1..Max_nodos] of nodo_;
end;

var red : red_;             { La red                          }
```

Figura 11. Definición de la red y sus nodos.

Cuando se almacenan las probabilidades de ir a cada sucesor, en realidad lo que se almacena es la probabilidad acumulada ya que simplifica los cálculos. Por ejemplo, para el nodo 1 en la Figura 1 los arreglos tendrían los siguientes elementos (las casillas con * no son consideradas ya que solo hay 3 sucesores):

suc:	2	3	4	*	*
prob_suc:	0.5	0.7	1	*	*

Figura 11. Los arreglos *suc* y *prob_suc* para el nodo 1 de la Figura 1.

Para calcular el próximo nodo a visitar se usa el algoritmo siguiente:

```
X:=random;
i:=1;
while prob_suc[i]<X do i:=i+1;
{ el proximo nodo a visitar es suc[i] }
```

Figura 12. Algoritmo para calcular el próximo nodo a visitar.

random es una función de Turbo Pascal que devuelve un valor aleatorio uniforme entre 0 y 1. Nótese que los valores de *i* dependen del valor de *X* (ver Tabla 1) y corresponden a los dados en la Figura 1.

	i	*Probabilidad*
$0 \le X \le 0.5$	1	0.5
$0.5 < X \le 0.7$	2	0.2
$0.7 < X$	3	0.3

Tabla 1. Valores de *i* en función de *X*.

IV. ACUMULACION Y CÁLCULO DE ESTADISTICAS.

Entre las estadísticas que nos interesan para cada nodo están:

- Longitud promedio de cola
- Longitud máxima de cola
- Longitud de la cola al terminar la simulación
- Número de clientes en el nodo al terminar la simulación
- Tiempo promedio de espera en cola
- Ocupación promedio del servicio
- Tiempo en que el servicio estuvo libre
- Número de clientes que llegaron al nodo
- Número de clientes que completaron el servicio en el nodo

A continuación se muestra como calcular el tiempo promedio de espera en cola y la longitud promedio de cola para un nodo de capacidad uno. Los resultados son fácilmente extensibles a redes con múltiples nodos y capacidades, así como a otras estadísticas.

1. Calculo del *tiempo promedio de espera en cola* y de la *longitud promedio de cola*

Supongamos una red con un nodo de capacidad uno y la siguiente secuencia de eventos:

Evento	L_1	L_2	L_3	S_1	L_4	L_5	S_2	S_3	S_4
Tiempo	1	2	3	4	5	6	8	11	13

en donde:

L_i = tiempo de llegada del *i-esimo* cliente

S_i = tiempo de salida del *i-esimo* cliente

Si graficamos los tiempos de espera tenemos:

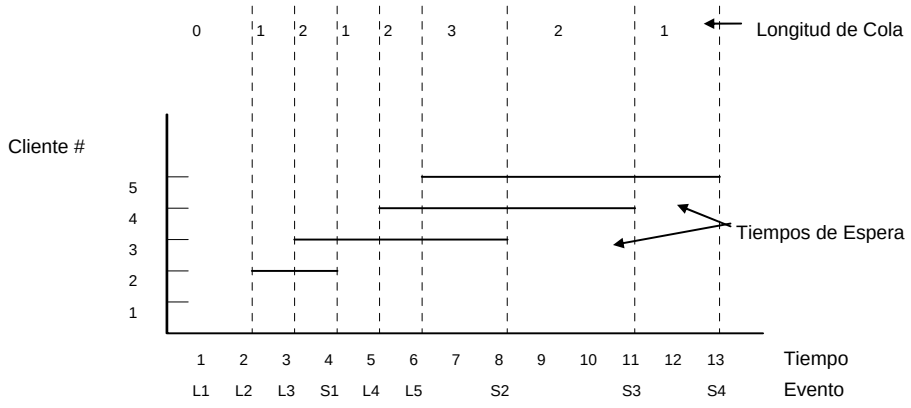


Figura 13. Gráfica de los tiempos de espera y la longitud de cola.

De la gráfica tenemos que el *tiempo total de espera* (*tte*) esta dado por:

$$tte = (S_1 - L_2) + (S_2 - L_3) + (S_3 - L_4) + (S_4 - L_5)$$

$$= (L_3 - L_2).1 + (S_1 - L_3).2 + (L_4 - S_1).1 + (L_5 - L_4).2 + (S_2 - L_5).3 + (S_3 - S_2).2 + (S_4 - S_3).1$$

Si definimos E_i como el *i*-esimo evento y LC_i como la longitud de cola desde la ocurrencia del evento E_{i-1} a la ocurrencia del evento E_i , entonce podemos calcular *tte* usando:

$$tte = \sum_{i=3}^{n} (E_i - E_{i-1}).LC_i$$

y

$$tpe = \frac{tte}{numero \quad de \quad llegadas} \qquad\qquad lpc = \frac{tte}{tiempo \quad total}$$

2. Resultados analíticos para un servicio con media entre llegadas y tiempo de servicio exponencial

Para una red con un nodo de capacidad uno, se pueden obtener los siguientes resultados analíticos que usaremos posteriormente para verificar que el simulador funciona correctamente. Estos resultados se extienden a cadenas de nodos (uno detrás del otro) con los mismos tiempos de servicio y los cuales tendrán todos el mismo λ .

Sea λ la tasa de llegadas (clientes/unidad de tiempo) y μ la tasa de servicio (clientes por unidad de tiempo), y sea $\rho = \lambda / \mu$ la intensidad de tráfico, entonces:

Promedio de clientes en el sistema $L = \frac{\rho}{1-\rho}$

Tiempo promedio de permanencia en el sistema $w = \frac{1/\mu}{1-\rho}$

Tiempo promedio de espera en cola $w_Q = \rho \frac{1/\mu}{1-\rho}$

Longitud promedio de cola $L_Q = \frac{\rho^2}{1-\rho}$

Condición de estabilidad: $\lambda < \mu$ o $\rho < 1$

La probabilidad de tener 0 (cero) clientes en el sistema viene dada por: $P_0 = 1 - \rho$; por lo tanto, si simulamos por T unidades de tiempo el servicio permanece vacío aproximadamente $T\rho_0 = T(1-\rho)$ unidades de tiempo.

Ejemplo:

Si en promedio los clientes llegan cada 3 segundos y tardan 2 segundos en ser servidos (se asumen distribuciones exponenciales para los promedios), entonces:

$$\lambda = 1/3 \qquad \mu = 1/2 \qquad \rho = \frac{1/3}{1/2} = 2/3$$

$$w_q = \frac{2}{3}\frac{1}{\frac{1/2}{1/3}} = 4 \qquad L_Q = \frac{(2/3)^2}{1/3} = 4/3 = 1.333$$

Si T = 50000, entonces el servicio permanece vacío 50000(1/3) = 16666.67 unidades de tiempo en promedio.

Si en vez de capacidad uno el nodo tiene capacidad multiple c (note que el problema anterior se convierte en un caso particular de este cuando c = 1) tenemos:

La intensidad de tráfico $\rho = \lambda / c\mu$

Probabilidad de que el sistema este vacio $P_0 = \left\{ \left[\sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!} \right] + \left[(c\rho)^c \left(\frac{1}{c!} \right) \frac{1}{1-\rho} \right] \right\}^{-1}$

Promedio de clientes en el sistema $L = c\rho + \frac{(c\rho)^{c+1} P_0}{c(c!)(1-\rho)^2}$

Tiempo promedio de permanencia en el sistema $w = \frac{L}{\lambda}$

Tiempo promedio de espera $w_Q = w - \frac{1}{\mu}$

Longitud promedio de cola = $L_Q = \lambda\, w_Q$

Condición de estabilidad: $\lambda < c\mu$ o $\rho < 1$

Ejemplo:

Con capacidad dos, si en promedio los clientes llegan cada 3 segundos y tardan 4 segundos en ser servidos (se asumen distribuciones exponenciales para los promedios), entonces:

$$\lambda = 1/3 \qquad \mu = 1/4 \qquad c = 2 \qquad \rho = \frac{1/3}{2\,1/4} = 2/3$$

$$P_0 = 0.20 \qquad w_q = 3.20 \qquad L_Q = 1.07$$

Si T = 50000, entonces el servicio permanece vacío 50000(0.20) = 10000.00 unidades de tiempo en promedio.

V. EL SIMULADOR

1. El código completo del simulador (Delphi 5)

```
{  1}  program redes;
{  2}  (*
{  3}    Creado por: Herbert Hoeger
{  4}
{  5}    Este programa permite simular una red con nodos de capacidad multiple y
{  6}    tiempos exponenciales. La red es un sistema abierto, o cerrado y se
{  7}    definen dos tipos de nodos. Los nodos tipo 'llegada' tienen dos parametros:
{  8}    el tiempo promedio entre llegadas de los clientes que llegan del exterior,
{  9}    y el tiempo promedio de servicio. Los nodos tipo 'otro' solo necesitan del
{ 10}    tiempo promedio de servicio ya que los clientes que reciben salen de otros
{ 11}    nodos y no llegan del exterior.
{ 12}    Cada nodo tambien necesita la informacion de los nodos sucesores (a los
{ 13}    cuales los clientes se pueden dirigir al salir del nodo) y la distribucion
{ 14}    de probabilidades respectiva. Si un nodo no tiene sucesores entonces los
{ 15}    clientes que salientes abandonan el sistema.
{ 16}    La definicion de la red es leida de un archivo texto (ver procedimiento
{ 17}    lee_red mas adelante) con extension '.def' por defecto. Los resultados
{ 18}    van a un archivo que por defecto tiene el mismo nombre extension '.out'
{ 19}  *)
{ 20}
{ 21}  uses Sysutils;
{ 22}
{ 23}  const Max_nodos = 20;                  { Maximo numero de nodos en la red }
{ 24}        Max_sucesores = 5;               { Maximo numero de sucesores por nodo }
{ 25}        Max_Real = 1.7e38;               { Maximo numero real }
{ 26}
{ 27}  type  tipo_ = (Llegada, Llegada_i, Llegada_e, Salida, Otro, Ultimo);
{ 28}                                         { sirve para eventos y nodos }
{ 29}
{ 30}        nodo_ = record
{ 31}          tipo_n : tipo_;
{ 32}          llegaron, servidos, t_llegadas, t_servicio, t_total_esp,
{ 33}          t_even_prev, t_vacio, util_pond : real;
{ 34}          capacidad, utilizacion, cola, cola_ini, cola_max, num_suc : longint;
{ 35}          suc : array[1..Max_sucesores] of integer;
{ 36}          prob_suc : array[1..Max_sucesores] of real;
{ 37}        end;
{ 38}  (*
{ 39}        En orden: tipo del nodo, contador de los clientes que han llegado al
{ 40}        nodo, los que han sido servidos, tiempo promedio entre llegadas, tiempo
{ 41}        promedio de servicio, tiempo total de espera, tiempo del evento previo,
{ 42}        tiempo vacio, utilizacion ponderada, capacidad, longitud de cola,
{ 43}        utilizacion, longitud de cola maxima, numero de nodos sucesores,
{ 44}        los sucesores, la distribucion de probabilidad para los sucesores.
{ 45}  *)
{ 46}        apt_evento = ^evento_;
{ 47}
{ 48}        evento_ = record
{ 49}          tiempo : real;
{ 50}          nodo : integer;
{ 51}          tipo : tipo_;
{ 52}          proximo : apt_evento;
{ 53}        end;
{ 54}  (*
{ 55}        En orden: tiempo de ocurrencia del evento, nodo de ocurrencia del
{ 56}        evento, tipo de evento, y apuntador al proximo evento en la Lista
{ 57}        de Eventos Pendientes (LEP).
```

```
{ 58}  *)
{ 59}        red_ = record
{ 60}          num_nodos : integer;
{ 61}          nodos : array[1..Max_nodos] of nodo_;
{ 62}        end;
{ 63}
{ 64}  var red : red_;                          { La red                          }
{ 65}      TSIM : real;                         { Tiempo de simulacion            }
{ 66}      i, nodo_actual : integer;            { Auxiliar, nodo en que ocurrio el  }
{ 67}                                           {   evento                        }
{ 68}      primero, primero_pv,                 { Apuntadores a la cabeza de la LEP }
{ 69}      evento, evento_a : apt_evento;       {   a la cabeza del pull de vacios  }
{ 70}                                           {   al evento que se esta procesando}
{ 71}                                           {   y un evento auxiliar          }
{ 72}      nombre_e, nombre_s : string[12];     { Nombre de los archivos de entrada }
{ 73}                                           {   y salida                      }
{ 74}      tiempo_sim, X : real;                { Tiempo simulado, auxiliar       }
{ 75}      a_entrada, a_salida : text;          { Archivo de entrada y salida     }
{ 76}      t_inicial, t_final, t_dif : comp;    { Tiempo inicial, tiempo final y la }
{ 77}                                           {   diferencia entre anteriores   }
{ 78}
{ 79}  function alea_exp(media : real) : real;
{ 80}  (*
{ 81}    Genera numeros aleatorios con distribucion exponencial de media 'media'
{ 82}  *)
{ 83}  begin
{ 84}    alea_exp := -media*ln(random);
{ 85}  end;
{ 86}
{ 87}  procedure lee_nom_arch;
{ 88}  var i : integer;
{ 89}  begin
{ 90}    writeln;
{ 91}    write('Nombre del archivo de entrada: ');readln(nombre_e);
{ 92}    if Pos('.',nombre_e) = 0 then nombre_e := nombre_e+'.def';
{ 93}    assign(a_entrada,nombre_e);
{ 94}    reset(a_entrada);
{ 95}    writeln;
{ 96}    write('Nombre del archivo de salida (con -> pantalla:) ');readln(nombre_s);
{ 97}    If nombre_s = '' then
{ 98}      nombre_s := Copy(nombre_e,1,Pos('.',nombre_e)) + 'out';
{ 99}    for i := 1 to length(nombre_s) do nombre_s[i] := upcase(nombre_s[i]);
{100}    assign(a_salida,nombre_s);
{101}    rewrite(a_salida);
{102}  end;
{103}
{104}  procedure lee_red;
{105}  (*
{106}    Lee la definicion de la red de un archivo con extension '.red' y la
{107}    almacena en la estructura 'red'
{108}
{109}    La primera linea es el tiempo de simulacion (Real) y el numero inicial de
{110}    clientes en el sistema, que sera repartido uniformemente entre los nodos.
{111}    Luego el archivo debe definir nodo por nodo de la siguiente forma:
{112}       Tipo del nodo: 1 -> Llegada, otro entero -> Otro  (Enteros)
{113}       Si el nodo es de tipo 'Llegada':
{114}         Tiempo entre llegadas, de servicio y capacidad  (Real, Real, Entero)
{115}       Si el nodo es es tipo 'Otro':
{116}         Tiempo de servicio y capacidad  (Real, Entero)
{117}       Numero de sucesores  (Entero - si no hay debe ser 0)
{118}       Sucesores - si los hay  (Enteros)
{119}       Distribucion de probabilidades - si hay sucesores  (Reales)
{120}  *)
{121}  var i, j , num_cli: integer;
```

```
{122}  begin
{123}    readln(a_entrada,TSIM, num_cli);
{124}    i := 0;
{125}    with red do begin
{126}      while not eof(a_entrada) do begin
{127}        i := i + 1;
{128}        with nodos[i] do begin
{129}          readln(a_entrada,j);
{130}          if j=1 then tipo_n := Llegada
{131}          else tipo_n := Otro;
{132}          if tipo_n = Llegada then read(a_entrada,t_llegadas);
{133}          readln(a_entrada,t_servicio,capacidad);
{134}          readln(a_entrada,num_suc);
{135}          if num_suc > 0 then begin
{136}            for j := 1 to num_suc do read(a_entrada,suc[j]);
{137}            readln(a_entrada);
{138}            for j := 1 to num_suc do read(a_entrada,prob_suc[j]);
{139}            readln(a_entrada);
{140}          end;
{141}          { inicializacion del resto de los parametros }
{142}          t_total_esp := 0;  t_even_prev := 0;   llegaron := 0;
{143}          cola := 0;         t_vacio := 0;       cola_max := 0;
{144}          servidos := 0;     util_pond := 0;     utilizacion := 0;
{145}          cola_ini := 0;
{146}        end;
{147}      end;
{148}      num_nodos := i;
{149}    end;
{150}    { Reparte los clientes iniciales uniformemente entre los nodos }
{151}    i := 1;
{152}    for j:=1 to num_cli do begin
{153}      red.nodos[i].cola := red.nodos[i].cola + 1;
{154}      i := i + 1;
{155}      if i > red.num_nodos then i := 1;
{156}    end;
{157}    close(a_entrada);
{158}  end;
{159}
{160}  procedure imprime_red;
{161}  (*
{162}    Imprime la definicion de la red almacenada en la estructura 'red'
{163}  *)
{164}  var i, j : integer;
{165}  begin
{166}    writeln(a_salida);
{167}    writeln(a_salida,'Archivo: ',nombre_e);
{168}    writeln(a_salida);
{169}    with red do
{170}      for i := 1 to num_nodos do
{171}        with nodos[i] do begin
{172}          if tipo_n = Llegada then
{173}            writeln(a_salida,'Nodo: ',i:2,'  Tipo: Llegada')
{174}          else writeln(a_salida,'Nodo: ',i:2,'  Tipo: Otro');
{175}          if tipo_n = Llegada then
{176}            write(a_salida,'T_E_Llegadas: ',t_llegadas:5:2);
{177}          writeln(a_salida,'  T_Servicio: ',t_servicio:5:2,
{178}                           '  Capacidad: ',capacidad:3);
{179}          write(a_salida,'Sucesores: ');
{180}          for j := 1 to num_suc do
{181}            write(a_salida,suc[j],' (',prob_suc[j]:3:2,')   ');
{182}          writeln(a_salida); writeln(a_salida);
{183}       end;
{184}  end;
{185}
```

```
{186}  procedure rec_LEP;
{187}  (*
{188}    Recorre e imprime los eventos en la LEP. Se uso para fines de depuracion
{189}    pero no es necesario para la simulacion.
{190}  *)
{191}  var evento : apt_evento;
{192}  begin
{193}    evento := primero^.proximo;
{194}    while evento <> nil do begin
{195}      with evento^ do begin
{196}        if tipo = Llegada then write('Llegada ')
{197}        else write('Salida ');
{198}        writeln('nodo: ',nodo,'  tiempo: ',tiempo:9:2);
{199}      end;
{200}      evento := evento^.proximo;
{201}    end;
{202}  end;
{203}
{204}  procedure inserta(var evento : apt_evento);
{205}  (*
{206}    Inserta un evento en la lista de eventos pendientes
{207}  *)
{208}  var ant, pro : apt_evento;
{209}  begin
{210}    ant := primero;
{211}    pro := primero^.proximo;
{212}    while evento^.tiempo >= pro^.tiempo do begin
{213}      ant := pro;
{214}      pro := pro^.proximo;
{215}    end;
{216}    ant^.proximo := evento;
{217}    evento^.proximo := pro;
{218}  end;
{219}
{220}  procedure inicia_LEP;
{221}  (*
{222}    Inicializa la Lista de Eventos Pendientes: crea 'primero', 'ultimo' y las
{223}    primeras llegadas.
{224}  *)
{225}  var n_nodo,i,iniciales : integer;
{226}  begin
{227}
{228}    new(evento);                          { crea el evento que marca el fin de }
{229}    with evento^ do begin                 { la LEP - nunca sera procesado      }
{230}      tiempo := Max_Real;;
{231}      nodo := 0;
{232}      tipo := Ultimo;
{233}      proximo := nil;
{234}    end;
{235}
{236}    new(primero);                          { crea apuntador al comienzo de la  }
{237}    primero^.proximo := evento;            { LEP                               }
{238}
{239}    for n_nodo := 1 to red.num_nodos do    { genera las primeras llegadas de   }
{240}      with red.nodos[n_nodo] do begin      { todos los nodos que tienen        }
{241}        if tipo_n = Llegada then begin     { llegadas externas                 }
{242}          new(evento);
{243}          with evento^ do begin
{244}            tiempo := alea_exp(t_llegadas);
{245}            nodo := n_nodo;
{246}            tipo := Llegada_e;
{247}          end;
{248}          inserta(evento);
{249}        end;
```

```
{250}        if cola > 0 then begin             { hay clientes iniciales           }
{251}          cola_ini := cola;
{252}          cola_max := cola;
{253}          llegaron := cola;
{254}          if cola > capacidad then iniciales := capacidad
{255}          else iniciales := cola;
{256}          for i:=1 to iniciales do begin
{257}            cola := cola - 1;
{258}            utilizacion := utilizacion + 1;
{259}            new(evento);
{260}            with evento^ do begin
{261}              tiempo := alea_exp(t_servicio);
{262}              nodo := n_nodo;
{263}              tipo := Salida;
{264}            end;
{265}            inserta(evento);
{266}          end;
{267}        end;
{268}      end;
{269}
{270}    primero_pv := nil;                   { inicializa el pull de vacios      }
{271}
{272}  end;
{273}
{274}  function prox_evento : apt_evento;
{275}  (*
{276}    Retorna el proximo evento a procesar y actualiza el tiempo simulado.
{277}  *)
{278}  var evento : apt_evento;
{279}  begin
{280}    evento := primero^.proximo;
{281}    primero^.proximo := evento^.proximo;
{282}    prox_evento := evento;
{283}    tiempo_sim := evento^.tiempo;
{284}  end;
{285}
{286}  (* Procedimientos para obtener y retornar registros de eventos en el pull
{287}    de eventos vacios *)
{288}
{289}  procedure new_n(var evento : apt_evento);
{290}  begin
{291}    if primero_pv <> nil then begin
{292}      evento := primero_pv;
{293}      primero_pv := evento^.proximo;
{294}    end else new(evento);
{295}  end;
{296}

{297}  procedure dispose_n(var evento : apt_evento);
{298}  begin
{299}    evento^.proximo := primero_pv;
{300}    primero_pv := evento;
{301}  end;
{302}
{303}  begin
{304}    randomize;                                 { Inicializa la semilla de los }
{305}                                               {   numeros aleatorios         }
{306}    lee_nom_arch;
{307}    lee_red;
{308}    imprime_red;
{309}
{310}    t_inicial := TimeStampToMSecs(DateTimeToTimeStamp(Time));  { Tiempo Inicio }
{311}
{312}    tiempo_sim := 0;
```

```
{313}
{314}    inicia_LEP;
{315}
{316}    evento := prox_evento;                          { Obtener el primer evento }
{317}
{318}    while tiempo_sim < TSIM do begin
{319}      with evento^ do begin
{320}        nodo_actual := nodo;
{321}        with red.nodos[nodo_actual] do begin
{322}          t_total_esp := t_total_esp + cola*(tiempo_sim - t_even_prev);
{323}          util_pond := util_pond + utilizacion*(tiempo_sim - t_even_prev);
{324}          if tipo <> Salida then begin              { El evento es una Llegada }
{325}            llegaron := llegaron + 1;
{326}            if utilizacion = capacidad then begin   { Nodo ocupado - a la cola }
{327}              cola := cola + 1;
{328}              if cola > cola_max then cola_max := cola;
{329}            end else begin                         { Nodo vacio - genera salida }
{330}              if utilizacion = 0 then
{331}                t_vacio := t_vacio + (tiempo_sim - t_even_prev);
{332}              new_n(evento_a);
{333}              with evento_a^ do begin
{334}                tiempo := tiempo_sim + alea_exp(t_servicio);
{335}                nodo := nodo_actual;
{336}                tipo := Salida;
{337}              end;
{338}              inserta(evento_a);
{339}              utilizacion := utilizacion + 1;
{340}            end;
{341}            { Si la llegada es externa, genera la proxima llegada; las }
{342}            { variables nodo, tipo y externa son iguales al del evento }
{343}            { procesado    }
{344}            if tipo = Llegada_e then begin
{345}              tiempo := tiempo_sim + alea_exp(t_llegadas);
{346}              inserta(evento);
{347}            end else dispose_n(evento);
{348}          end else begin                    { El evento es una Salida         }
{349}            if num_suc > 0 then begin       { Calcula a donde va el cliente y }
{350}              X := random;                  {   genera la respectiva llegada  }
{351}              i := 1;
{352}              while prob_suc[i] < X do i := i + 1;
{353}              new_n(evento_a);
{354}              with evento_a^ do begin
{355}                tiempo := tiempo_sim;
{356}                nodo := suc[i];
{357}                tipo := Llegada_i;
{358}              end;
{359}              inserta(evento_a);
{360}            end;
{361}            servidos := servidos + 1;
{362}            if cola > 0 then begin
{363}              cola := cola - 1;
{364}              tiempo := tiempo_sim + alea_exp(t_servicio);
{365}              inserta(evento);
{366}            end else begin
{367}              utilizacion := utilizacion - 1;
{368}              dispose_n(evento);
{369}            end;
{370}          end;
{371}          t_even_prev := tiempo_sim;
{372}        end;
{373}      end;
{374}      evento := prox_evento;
{375}    end;
{376}
```

```
{377}    t_final := TimeStampToMSecs(DateTimeToTimeStamp(Time));     { Tiempo Final }
{378}    t_dif := t_final - t_inicial;
{379}
{380}    writeln(a_salida);                                    { Imprimir resultados }
{381}    write(a_salida,'Tiempo de simulacion: ',TSIM:8:0,'   Tiempo de corrida: ');
{382}    writeln(a_salida,t_dif/1000:6:2,'s');
{383}    writeln(a_salida);
{384}    writeln(a_salida,
{385}    'Nod Llega  Servi  EnSer LCol  LColM ColI  TProEsp LProCol  T Vacio  OcupPro');
{386}    writeln(a_salida,
{387}    '=== ====== ====== ===== ===== ===== ===== ======= ======= ========= =======');
{388}    for i:=1 to red.num_nodos do
{389}      with red.nodos[i] do begin
{390}        if utilizacion = 0 then t_vacio := t_vacio + (TSIM - t_even_prev)
{391}        else util_pond := util_pond + utilizacion*(TSIM - t_even_prev);
{392}        t_total_esp := t_total_esp + cola*(TSIM - t_even_prev);
{393}        writeln(a_salida,
{394}                ' ',i:2,' ',llegaron:6:0,' ',servidos:6:0,' ',utilizacion:5,' ',
{395}               cola:5,' ',cola_max:5,' ',cola_ini:5,' ',
{396}               t_total_esp/llegaron:7:1,' ',t_total_esp/TSIM:7:1,' ',
{397}               t_vacio:9:2,' ',util_pond/TSIM:7:2);
{398}      end;
{399}    if nombre_s = 'CON' then readln;
{400}    close(a_salida);
{401}  end.
```

Nótese que de las 401 líneas que tiene el código, unas 104 corresponden a comentarios y líneas en blanco para fines de claridad y entendimiento, y otras 17 a la rutina *rec_LEP* (líneas 186 a 202) que no es necesaria. Esto deja unas 280 líneas de código útil.

2. Ejemplo de un archivo de definición de una red.

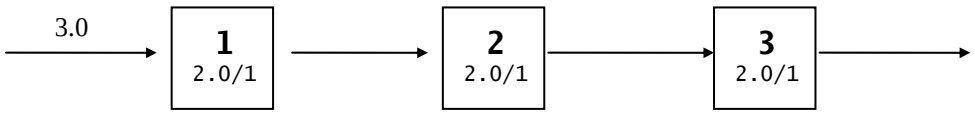
```
1000.0 0        Tiempo de simulación y número inicial de clientes
1               NODO 1  (llegada)
3.0 1.0 1       tiempo entre llegada, de servicio y capacidad
3               tres sucesores
2 3 4           nodos sucesores: 2, 3 y 4
0.5 0.7 1.0     probabilidades acumuladas
1               NODO 2  (llegada)
3.0 1.0 1       tiempo entre llegada, de servicio y capacidad
2               dos sucesores
1 4             sucesores: 1 y 4
0.5 1.0         probabilidades acumuladas
2               NODO 3  (otro)
0.5 1           tiempo de servicio y capacidad
1               un sucesor
5               nodo 5 es el sucesor
1.0             probabilidad
2               NODO 4  (otro)
0.5 1           tiempo de servicio y capacidad
1               un sucesor
5               nodo 5 es el sucesor
1.0             probabilidad
2               NODO 5  (otro)
0.2 1           tiempo de servicio y capacidad
0               no hay sucesores
```

Figura 14. Archivo de definición para la red dada en la Figura 1 con capacidades iguales a 1.

Para la red dada en la Figura 1 con capacidades iguales a 1, el archivo de definición esta dado en la Figura 14. Es de hacer notar que por la forma como el simulador lee estos archivos, los comentarios que se incluyen en el mismo son perfectamente validos. Lo único que hay que tener presente es que debe haber un espacio entre el ultimo dígito y el comienzo del comentario.La primera línea corresponde al tiempo de simulación y al número inicial de clientes (necesario en caso de que la red sea cerrada). Luego se va definiendo nodo por nodo. La numeración de los nodos es automática y es de acuerdo al orden en que se definen. Ver las líneas 104 a 158 del código para entender como se define cada nodo.

3. Resultados del simulador para 3 redes.

a) Red 1:



Al correr el simulador con la definición de esta red, se obtienen los resultados dados a continuación. Nótese que los resultados corresponden aproximadamente con los resultados analíticos presentados en la sección IV-2.

Definición leída y presentada por el simulador:

```
Archivo: red.def

Nodo:  1  Tipo: Llegada
T_E_Llegadas:  3.00  T_Servicio:  2.00  Capacidad:   1
Sucesores: 2 (1.00)

Nodo:  2  Tipo: Otro
  T_Servicio:  2.00  Capacidad:   1
Sucesores: 3 (1.00)

Nodo:  3  Tipo: Otro
  T_Servicio:  2.00  Capacidad:   1
Sucesores:
```

Estadísticas presentadas por el simulador:

```
Tiempo de simulacion:    50000   Tiempo de corrida: 0h 0m 10.6s

Nod Llega  Servi  EnSer LCol  LColM ColI  TProEsp LProCol  T Vacio  OcupPro
=== ====== ====== ===== ===== ===== ===== ======= ======= ========= =======
  1  16551  16551     0     0    19     0     4.0     1.3  16821.38    0.66
  2  16551  16551     0     0    20     0     3.9     1.3  17005.63    0.66
  3  16551  16548     1     2    21     0     3.7     1.2  17070.88    0.66
```

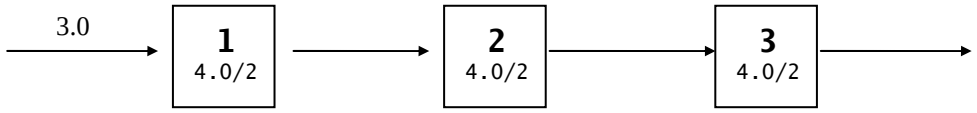
Estadísticas presentadas por el simulador para la misma red pero otra corrida:

```
Tiempo de simulacion:    50000   Tiempo de corrida: 0h 0m 10.32s

Nod Llega  Servi  EnSer LCol  LColM ColI  TProEsp LProCol  T Vacio  OcupPro
```

```
=== ====== ====== ===== ===== ===== ===== ======= ======= ========= =======
  1  16938  16938     0     0    17     0     4.0     1.4  16119.57    0.68
  2  16938  16937     1     0    18     0     4.2     1.4  16241.51    0.68
  3  16937  16935     1     1    21     0     4.2     1.4  16148.52    0.68
```

b) Red 2:



Estadísticas presentadas por el simulador (la definición es similar a la de la Red 1):

```
Tiempo de simulacion:    50000   Tiempo de corrida: 0h 0m 10.33s

Nod Llega  Servi  EnSer LCol  LColM ColI  TProEsp LProCol  T Vacio  OcupPro
=== ====== ====== ===== ===== ===== ===== ======= ======= ========= =======
  1  16387  16387     0     0    19     0     3.0     1.0  10288.10    1.32
  2  16387  16385     2     0    20     0     2.9     0.9  10498.18    1.30
  3  16385  16385     0     0    18     0     2.8     0.9  10817.02    1.29
```

b) Red 3:

Esta red corresponde a la dada en la Figura 1 pero con capacidades iguales a 1.

Definición leída y presentada por el simulador:

```
Archivo: red3.def

Nodo:  1  Tipo: Llegada
T_E_Llegadas:  3.00  T_Servicio:  1.00  Capacidad:   1
Sucesores: 2 (0.50)   3 (0.70)   4 (1.00)

Nodo:  2  Tipo: Llegada
T_E_Llegadas:  3.00  T_Servicio:  1.00  Capacidad:   1
Sucesores: 1 (0.50)   4 (1.00)

Nodo:  3  Tipo: Otro
  T_Servicio:  0.50  Capacidad:   1
Sucesores: 5 (1.00)

Nodo:  4  Tipo: Otro
  T_Servicio:  0.50  Capacidad:   1
Sucesores: 5 (1.00)

Nodo:  5  Tipo: Otro
  T_Servicio:  0.20  Capacidad:   1
Sucesores:
```

Estadísticas presentadas por el simulador:

```
Tiempo de simulacion:    50000   Tiempo de corrida: 0h 0m 27.80s

Nod Llega  Servi  EnSer LCol  LColM ColI  TProEsp LProCol  T Vacio  OcupPro
=== ====== ====== ===== ===== ===== ===== ======= ======= ========= =======
  1  33590  33587     1     2    42     0     2.2     1.5  16207.60    0.68
  2  33376  33376     0     0    21     0     2.0     1.3  16651.53    0.67
  3   6689   6689     0     0     3     0     0.0     0.0  46623.76    0.07
  4  26685  26685     0     0     8     0     0.2     0.1  36560.03    0.27
  5  33374  33374     0     0     4     0     0.0     0.0  43331.88    0.13
```