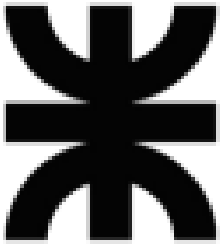

UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL ROSARIO

"SIMULACIÓN: SISTEMAS DE COLAS"

Integrantes:

Demaria, María Ayelén (ayedemaria@gmail.com)

Robaglio, Magdalena (robaglio.magdalena@gmail.com)

Docente: Juan Torres

Comisión: 403

Año: 2019

CONTENIDO

INTRODUCCIÓN

El fenómeno de la espera no es una experiencia que se limite sólo a los humanos: los trabajos esperan a ser procesados en una máquina, los aviones vuelan en círculo hasta que la torre de control les da permiso de aterrizar y los automóviles se detienen ante la luz roja de los semáforos. Desafortunadamente no se puede eliminar la espera sin incurrir en gastos desmesurados.

El estudio de las líneas de espera trata de cuantificar el fenómeno de esperar formando colas, mediante medidas representativas de eficiencia, como la longitud promedio de la cola, el tiempo promedio de espera en ella, y la utilización promedio de las instalaciones.

DESARROLLO

ELEMENTOS DE UN MODELO DE COLA

Los actores principales en una línea de espera o cola son el cliente y el servidor. Al llegar, los clientes pueden recibir servicio de inmediato, o esperar en una cola o línea de espera, si el servidor está ocupado. Cuando finaliza el servicio para un cliente, en forma automática se le cede a un cliente que espera, si lo hay, de la cola. Si la cola está vacía, el servidor se vuelve inactivo hasta que llega un cliente nuevo.

Desde el punto de vista del análisis de las colas, el proceso de llegada se representa con el tiempo entre llegadas, de los clientes sucesivos, y el servicio se describe con el tiempo de servicio por cada cliente. Por lo general, los tiempos entre llegadas y de servicio pueden ser probabilísticos o determinísticos.

El tamaño de la cola puede ser finito o infinito. El orden en el que se seleccionan los clientes de una cola, es un factor importante en el análisis de los modelos de colas.

La forma más común es la de primero en llegar, primero en servirse (FIFO, del inglés first-in, first-out). Entre otras están último en llegar, primero en servirse (LIFO, last-in, first-out), y de dar servicio en orden aleatorio (random). También, los clientes se pueden seleccionar en la cola con base en cierto orden de prioridad.

El comportamiento de los clientes en espera juega un papel en el análisis de las líneas de espera. Los clientes "humanos" se pueden saltar de una cola a otra, tratando de reducir la espera. También pueden rehusar totalmente a la cola por haber esperado demasiado.

El diseño de la instalación de servicio puede comprender servidores en paralelo. También, los servidores pueden ordenarse en serie o bien pueden formar una red.

La fuente donde se generan los clientes puede ser finita o infinita. Una fuente finita limita a los clientes que llegan al servicio. También, una fuente infinita es abundante por siempre.

Las variaciones de los elementos de un caso de colas dan lugar a diversos modelos de colas.

SISTEMA CON UN SERVIDOR

En el presente trabajo utilizaremos el modelo para el caso en que hay un solo servidor (c = 1), aunque hay dos modelos con un servidor: en el primer modelo no se establece límite para la cantidad máxima en el sistema, y en el segundo se supone un límite finito del sistema.

Las llegadas suceden con la frecuencia de clientes por unidad de tiempo (λ), y la tasa de servicio es clientes por unidad de tiempo (μ).

Utilizamos la notación de Kendall para resumir las características del sistema: **M/M/1**.

NOTACIÓN DE KENDALL

Notación de Kendall: A / B / C / D / E / F

- **A**: la distribución de llegada.
- **B**: la distribución de servicio.
  A y B pueden ser M, D o G:
  - **M**: Distribución de Markov. La tasa de arribos es una variable de Poisson, el tiempo entre arribos es Exponencial.
  - **D**: Distribución determinística (un valor fijo).
  - **G**: General, es decir cualquier distribución de probabilidad, menos Poisson o Exponencial.

- **C**: entero positivo que denota el número servidores en paralelo.
- **D**: Cantidad máxima de clientes permitidos en el sistema. Si esta capacidad es superada se rechaza el arribo de un nuevo cliente.
- **E**: Política de atención de la cola:
  - **FIFO** (first in first out).
  - **LIFO** (last in first out).
  - **SIRO** (service in random order).
  - **Prioridad**.
- **F**: Tamaño de la población que ingresa al sistema. Infinita o un valor numérico.

MEDIDAS DE RENDIMIENTO

El estudio de sistemas de cola trata de cuantificar el fenómeno de esperar formando colas, mediante medidas representativas de eficiencia, como la longitud promedio de la cola, el tiempo promedio de espera en ella, y la utilización promedio de las instalaciones.

Si bien estas son las medidas de rendimiento más habituales, y son las que desarrollamos y aplicamos en el presente trabajo, al estudiar un sistema se puede estudiar también el valor mínimo, el valor máximo o la distribución de probabilidad (comportamiento) de la variable.

Los contadores estadísticos son las variables auxiliares dentro del algoritmo de simulación que permiten calcular estas medidas de rendimiento.

Demora promedio por cliente

Es la demora en cola que en promedio esperan los clientes. Es una medida que indica desde el punto de vista del cliente qué tan bien se comporta el sistema prestándoles un servicio.

Se calcula como la sumatoria de las demoras individuales por cliente dividido la cantidad de clientes que completaron demora. En general se calcula para cada cola que haya en el sistema.

La demora estimada que emplea un cliente esperando en cola es:

$$W_q = \frac{\lambda}{\mu\left(\mu - \lambda\right)} \tag{1}$$

Proporción del tiempo en que el servidor permanece ocupado

Esta variable evalúa al sistema desde el punto de vista de maximizar la utilización de los recursos, sobre los que se ha invertido.

Se calcula como la sumatoria de los tiempos de servicio incurridos en atención de clientes, dividido por el tiempo final de la simulación o de la corrida; y para cada servidor en el sistema.

Se puede ver como el área bajo B(t) dividida por el tiempo de la simulación. B(t) es una función que asume el valor 1 desde el instante en que el servidor está ocupado y 0 cuando está desocupado.

La proporción de utilización teórica del servidor está dada por la siguiente fórmula:

$$\rho = \frac{\lambda}{\mu} \tag{2}$$

Número promedio en el tiempo de clientes en cola

Esta variable evalúa al sistema desde el punto de vista de la atención al cliente. Representa el tamaño promedio de la cola a lo largo de toda la simulación.

Se calcula a partir del área bajo Q(t) dividido por el tiempo final de la simulación o de la corrida.

Q(t) es una función que indica la cantidad de clientes en cola en el tiempo t.; y se implementa en el algoritmo como una variable que se incrementa en uno cuando entra un cliente en cola, y se decrementa en uno cuando se quita a un cliente de la cola.

El número estimado de clientes que esperan ser atendidos es:

$$L_q = \frac{\lambda^2}{\mu\,(\mu - \lambda)} \tag{3}$$

RUTINAS DE CONTROL

Rutina de inicialización

Contendrá la definición e inicialización de:

• Una estructura de datos para mantener la información de los clientes que están en cola.

• Una estructura de datos para mantener información de los eventos a agendar a futuro y procesar. La llamaremos Lista de Eventos.

• Los contadores estadísticos necesarios para cada variable de respuesta:

• Demora promedio por cliente.

• Acumulador de demoras individuales.

• Cantidad de clientes que completaron demora.

• Número promedio en el tiempo de clientes en cola

• Área bajo Q(t)

• Tiempo final de la simulación (Reloj al ejecutar Reporte)

• Proporción que el servidor permanece ocupado

• Proporción que el servidor permanece ocupado

• Acumulador de tiempos de servicio

• Una variable que contiene el instante del tiempo de la simulación

• Una variable que contiene el instante del tiempo del último evento

Rutina de Tiempos

Contendrá:

• La selección del próximo evento a procesar, como el más cercano en el tiempo.

• La actualización de la variable de tipo de evento al tipo de evento a ocurrir.

• La actualización de la variable Tiempo del último evento.

• La actualización de la variable Reloj con el instante en que ocurrirá el próximo evento.

Rutina de arribos / partidas

En la definición de la lógica de las rutinas que procesan eventos se debe tener en cuenta los siguientes criterios:

• Por cada flecha llena (desencadenamiento obligatorio) que sale del evento en cuestión, se debe agendar a futuro el evento indicado como destino de la flecha.

• Por cada flecha punteada (desencadenamiento condicional) que sale del evento, se debe evaluar la condición (con un SI en pseudocódigo).

- Si es verdadera, generar a futuro el evento indicado como destino de la flecha.
- Si no lo es no generar el evento indicado como destino de la flecha.

En cada camino de la lógica evaluar lo siguiente:

• Qué variables de estado se deben actualizar.

• Por cada variable de respuesta, qué contadores estadísticos se deben actualizar.

• El orden en el que se actualizan estas variables deben asegurar el correcto cálculo de los contadores estadísticos relacionados a las variables de respuesta.

Rutina de Reportes

Se debe aplicar los siguientes criterios en la definición de la lógica de esta rutina:

• Dado que cada vez que se ejecuta esta rutina se obtiene una observación de cada variable de respuesta, es necesario guardar estas observaciones de manera que puedan ser utilizadas en el análisis de datos.

• Se debe reportar solamente aquellas variables de respuesta que han sido identificadas en el paso 1 de una experiencia de simulación.

• La cantidad de observaciones a generar va a depender del nivel de precisión con el que estarán estimados los valores reales de las variables. Dicha precisión estará dada por el nivel de error tolerado que será definido teniendo en cuenta el orden de magnitud de los valores que asume la variable de respuesta.

SIMULACIÓN

Realizamos la simulación de un sistema de colas M/M/1 utilizando el lenguaje de programación Python.

Para comenzar, realizamos la simulación de un único sistema M/M/1 tomando las siguientes variables de entrada:

- Tasa de llegadas (λ): 2.
- Tasa de servicio (μ): 3.
- Número máximo de clientes: 200.

Y obtuvimos los siguientes resultados:

Medida	**Valores simulación**	**Valores teóricos**
Número promedio de clientes en cola	1.283	1.333
Demora promedio en cola	0.676	0.666
Utilización promedio del servidor	0.667	0.666

Cuadro 1: Tabla comparativa.

Como puede notarse, los valores calculados durante la simulación son muy similares a los calculados teóricamente.

A continuación se muestra una gráfica generada en tiempo real que muestra en la izquierda la gráfica de la función Q(t) (número de clientes en cola) y a la derecha B(t) (1: servidor utilizado; 0: servidor libre).

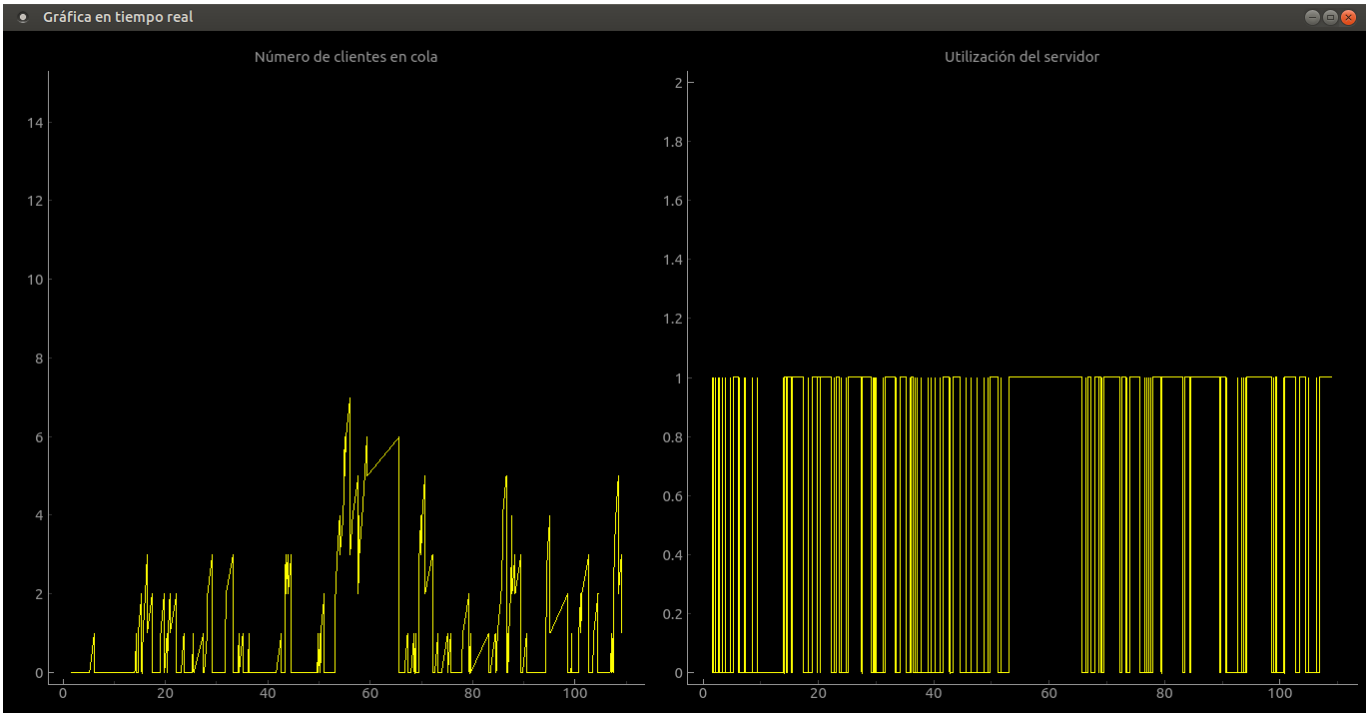

Figura 1: Gráfica en tiempo real de Q(t) a la izquierda, B(t) a la derecha.

Luego, realizamos la misma simulación pero esta vez lo repetimos n=1000 veces. A continuación se grafican los resultados obtenidos junto con los valores teóricos correspondientes:

Figura 2: Número promedio de clientes en cola.

Figura 3: Utilización promedio del servidor.

Figura 4: Número promedio de demora en cola.

ANEXOS

Código Python

```
# -* - coding: utf-8 -* -
import numpy as np
import matplotlib.pyplot as plt
import statistics


# Variables globales
CANT_TIPOS_EVENTOS = 2# Defimos número de tipos de eventos (usamos 2: arribos y llegadas)
# Criterio de estabilidad MM1: la tasa de servicio debe ser mayor que la tasa de llegada
TIEMPO_MEDIO_LLEGADA = 2.0# tiempo medio de llegada * * lambda
TIEMPO_MEDIO_SERVICIO = 3.0# tiempo medio de servicio * * MU
TOTAL_CLIENTES = 200# número total de clientes cuyas demoras serán observadas
TIEMPO = 0.0# Reloj de simulación
ESTADO = 0 # 0: si el servidor está ocioso - 1: si el servidor está ocupado
ANCC=0.0 # área debajo de la función número de clientes en cola
NCC=0 # número de clientes en cola
TIEMPO_ULT_EV=0.0 # tiempo del último evento que cambió el número en cola
NUM_CLIENTES = 0 # número de clientes que completaron sus demoras
ARREGLO_PROX_EV = np.zeros([CANT_TIPOS_EVENTOS+1]) # arreglo que contiene el tiempo del próximo evento
I en la posición ARREGLO_PROX_EV[I]
TIEMPO_TOT_DEMORAS=0.0 # tiempo total de los clientes que completaron sus demoras
TIEMPO_PROX_EV = 0.0 # tiempo de ocurrencia del próximo evento a ocurrir
TIPO_PROX_EV = 0 # tipo de evento (1: ARRIBOS o 2: PARTIDAS) del próximo evento que va a ocurrir
ARREGLO_TIEMPOS_ARRIBO = np.zeros([TOTAL_CLIENTES+1]) # tiempo de arribo del cliente I que está esperando en cola
TIEMPO_SERV_ACUM = 0.0


# Subrutina init
def init():
    global TIEMPO, ESTADO, NCC, TIEMPO_ULT_EV, NUM_CLIENTES, TIEMPO_TOT_DEMORAS, ANCC,
ARREGLO_PROX_EV, TIEMPO_PROX_EV, TIPO_PROX_EV, ARREGLO_TIEMPOS_ARRIBO,TIEMPO_SERV_ACUM
    # inicializamos.0 el reloj de simulación
    TIEMPO = 0.0


    # inicializamos variables de estado
    ESTADO = 0 # 0: si el servidor está ocioso - 1: si el servidor está ocupado
    NCC=0 # número de clientes en cola
    TIEMPO_ULT_EV=0.0 # tiempo del último evento que cambió el número en cola


    # inicializamos contadores estadísticos
    NUM_CLIENTES = 0 # número de clientes que completaron sus demoras
    TIEMPO_TOT_DEMORAS=0.0 # tiempo total de los clientes que completaron sus demoras
    ANCC=0.0 # área debajo de la función número de clientes en cola


    ARREGLO_PROX_EV = np.zeros([CANT_TIPOS_EVENTOS+1]) # arreglo que contiene el tiempo del próximo
evento I en la posición ARREGLO_PROX_EV[I]
    TIEMPO_PROX_EV = 0.0 # tiempo de ocurrencia del próximo evento a ocurrir
    TIPO_PROX_EV = 0 # tipo de evento (1: ARRIBOS o 2: PARTIDAS) del próximo evento que va a ocurrir
    ARREGLO_TIEMPOS_ARRIBO = np.zeros([TOTAL_CLIENTES+1]) # tiempo de arribo del cliente I que está
esperando en cola
    TIEMPO_SERV_ACUM = 0.0


    # inicializamos lista de eventos. Como no hay clientes en cola, se define el tiempo de la próxima salida en infinito.
    ARREGLO_PROX_EV[1] = TIEMPO + np.random.exponential(1/TIEMPO_MEDIO_LLEGADA)# stats.expon(TIEMPO
# tiempo actual + valor generado exponencialmente con lambda = tiempo medio de llegada
    ARREGLO_PROX_EV[2] = 10.0* * 30 # Lo seteamos en infinito


    TIEMPO_SERV_ACUM = 0.0


    return None
```

```
    # Subrutina timing
    def timing():
        global TIEMPO,ARREGLO_PROX_EV, TIPO_PROX_EV, TIEMPO_PROX_EV, CANT_TIPOS_EVENTOS
        TIEMPO_PROX_EV = 10.0* * 29 # tiempo de ocurrencia del próximo evento a ocurrir
        TIPO_PROX_EV = 0

        # determinamos el tipo de evento del próximo evento que va a ocurrir
        for i in range(1,CANT_TIPOS_EVENTOS+1):
        if ARREGLO_PROX_EV[i] <TIEMPO_PROX_EV:
                TIEMPO_PROX_EV = ARREGLO_PROX_EV[i]
                TIPO_PROX_EV = i

        # veo que la lista de eventos no esté vacia
        if TIPO_PROX_EV >0:
        TIEMPO = ARREGLO_PROX_EV[TIPO_PROX_EV]

        # si la lista de eventos está vacía, fin de simulación
        else:
        print('Lista de eventos vacía. Fin de la simulación')

        return None

    # Subrutina arribos
    def arrive():
        global ESTADO,TIEMPO_TOT_DEMORAS,NUM_CLIENTES,ARREGLO_TIEMPOS_ARRIBO,ARREGLO_PROX_
        # programamos el próximo arribo
        ARREGLO_PROX_EV[1] = TIEMPO + np.random.exponential(1/TIEMPO_MEDIO_LLEGADA)# stats.expon(TIEMPO
# tiempo actual + valor generado exponencialmente con lambda = tiempo medio de llegada

        # Vemos el estado del servidor, si está vacío (=0) comienza el servicio el cliente que arribó
        if ESTADO == 1: # servidor ocupado, actualizamos área debajo de la función número de clientes en cola
        ANCC += NCC* (TIEMPO-TIEMPO_ULT_EV)
        TIEMPO_ULT_EV = TIEMPO

        # agregamos uno al número de clientes en cola
        NCC += 1
        # verificamos condición de máximos clientes en cola TOTAL_CLIENTES
        if NCC <= TOTAL_CLIENTES:
                # ARREGLO_TIEMPOS_ARRIBO: tiempo de arribo del cliente I que está esperando en cola
                ARREGLO_TIEMPOS_ARRIBO[NCC] = TIEMPO
        else:
                print('Se alcanzó el límite de clientes a observar')
        else:
        # servidor ocioso, cliente tiene demora nula
        DEMORA = 0.0

        # cambiamos estado del servidor a ocupado
        ESTADO = 1

        TIEMPO_TOT_DEMORAS += DEMORA

        # agregamos uno al número de clientes que completaron su demora
        NUM_CLIENTES += 1

        # generamos la salida
        ARREGLO_PROX_EV[2] = TIEMPO + np.random.exponential(1/TIEMPO_MEDIO_SERVICIO)# stats.expon(TIEMPO
# tiempo actual + valor generado exponencialmente con lambda = tiempo medio de servicio

        TIEMPO_SERV_ACUM += (ARREGLO_PROX_EV[2] - TIEMPO)

        return None

    # Subrutina partidas
    def depart():
```

```
        global NCC, ESTADO, ANCC, TIEMPO, TIEMPO_ULT_EV, ARREGLO_TIEMPOS_ARRIBO, TIEMPO_TOT_DEMO
ARREGLO_PROX_EV, DEMORA, NUM_CLIENTES, TIEMPO_MEDIO_SERVICIO,TIEMPO_SERV_ACUM
        # si la cola está vacía, cambiamos el estado del servidor a ocioso
        # y seteamos el tiempo del próximo evento de partida en infinito
        if NCC >0:
        # la cola no está vacía
        # actualizamos el área debajo de la función de números de clientes en cola
        ANCC += NCC* (TIEMPO-TIEMPO_ULT_EV)
        TIEMPO_ULT_EV = TIEMPO

        # restamos uno del número de clientes en cola
        NCC -= 1# NCC

        # calculamos la demora del cliente que está comenzando el servicio
        DEMORA = TIEMPO - ARREGLO_TIEMPOS_ARRIBO[1]# Para mi esto está mal, en lugar de 1 debería ser NCC

        TIEMPO_TOT_DEMORAS += DEMORA
        # agregamos uno al número de clientes que completaron su demora
        NUM_CLIENTES += 1
        # calculamos la partida
        ARREGLO_PROX_EV[2] = TIEMPO + np.random.exponential(1/TIEMPO_MEDIO_SERVICIO)# stats.expon(TIEMPO
# tiempo actual + valor generado exponencialmente con lambda = tiempo medio de servicio
        TIEMPO_SERV_ACUM += (ARREGLO_PROX_EV[2] - TIEMPO)

        # si la cola no está vacía, mover cada cliente de la cola en una posición
        if NCC != 0:
                for i in range(1, NCC + 1):
                j = i + 1
                ARREGLO_TIEMPOS_ARRIBO[i]=ARREGLO_TIEMPOS_ARRIBO[j]

        else:
        # marcamos el servidor como libre
        ESTADO = 0
        ARREGLO_PROX_EV[2] = 10.0* * 30 # Lo seteamos en infinito

        return None

    # Subrutina reportes
    def report():
        global TIEMPO_MEDIO_LLEGADA, TIEMPO_MEDIO_SERVICIO, TOTAL_CLIENTES, NUM_CLIENTES,
ANCC, TIEMPO_TOT_DEMORAS, TIEMPO, TIEMPO_SERV_ACUM
        # mostramos encabezado y parámetros de entrada
        # print("Sistema de cola simple")
        # print("Tiempo medio entre arribos:", TIEMPO_MEDIO_LLEGADA,' minutos')# , '. Tasa de llegadas:', 1/TIEM-
PO_MEDIO_LLEGADA)
        # print("Tiempo medio de servicio:", TIEMPO_MEDIO_SERVICIO,' minutos')# , '. Tasa de servicio:', 1/TIEM-
PO_MEDIO_SERVICIO)
        # print("Número máximo de clientes:", TOTAL_CLIENTES)

        AVGNCC = ANCC/TIEMPO
        # print("Número promedio de clientes en cola", AVGNCC)

        AVGDEL = TIEMPO_TOT_DEMORAS/NUM_CLIENTES
        # print("Demora promedio en cola:",AVGDEL,' minutos')

        AVGUTSERV = TIEMPO_SERV_ACUM/TIEMPO
        # print(Ütilización promedio del servidor:",AVGUTSERV)

        return AVGNCC, AVGDEL, AVGUTSERV

    # Programa principal
    # l:lambda; mu:mu
    def main_program(l, mu):
        # print(.^entramos a main program")
```

```
        global NUM_CLIENTES, TOTAL_CLIENTES, TIPO_PROX_EV,TIEMPO_PROX_EV, TIEMPO_MEDIO_LLEGADA
TIEMPO_MEDIO_SERVICIO

        TIEMPO_MEDIO_LLEGADA = l

        TIEMPO_MEDIO_SERVICIO = mu

        reporte = ()
        # iniciamos la simulación, llamamos subrutina init
        init()

        # si la simulación terminó, llamamos la rutina de reportes y fin de la simulación
        while NUM_CLIENTES <= TOTAL_CLIENTES:# no terminó simulación
        # determinamos próximo evento, llamamos subrutina timing
        timing()

        # vemos qué tipo de evento es el próximo
        if TIPO_PROX_EV == 1:
                # llamamos la rutina de arribos de eventos
                arrive()
        else:
                # llamamos la rutina de partidas
                depart()

        else:# terminó simulación
        # llamamos a la subrutina de reportes
        reporte = report()

        return reporte

    if __name__ == '__main__':
        clientes_en_cola = []
        demora_en_cola = []
        utilizacion_servidor = []

        # Valores teóricos
        promedio_clientes_en_cola = (TIEMPO_MEDIO_LLEGADA**2)/(TIEMPO_MEDIO_SERVICIO*(TIEMPO_MEDIO
TIEMPO_MEDIO_LLEGADA))
        promedio_demora_en_cola = TIEMPO_MEDIO_LLEGADA/(TIEMPO_MEDIO_SERVICIO*(TIEMPO_MEDIO_SER
TIEMPO_MEDIO_LLEGADA))
        promedio_utilizacion_servidor = TIEMPO_MEDIO_LLEGADA/TIEMPO_MEDIO_SERVICIO

        n = 1000
        l = 2 # lamda
        mu = 3
        for i in range(n):
        rta = main_program(l,mu)
        clientes_en_cola.append(rta[0])
        demora_en_cola.append(rta[1])
        utilizacion_servidor.append(rta[2])

        lista_clientes_en_cola = []
        lista_demora_en_cola = []
        lista_utilizacion_servidor = []

        for i in range(n):
        clientes_en_cola_i = statistics.mean(clientes_en_cola[:i+1])
        lista_clientes_en_cola.append([i,clientes_en_cola_i])
        demora_promedio_i = statistics.mean(demora_en_cola[:i+1])
        lista_demora_en_cola.append([i,demora_promedio_i])
        utilizacion_servidor_i = statistics.mean(utilizacion_servidor[:i+1])
        lista_utilizacion_servidor.append([i,utilizacion_servidor_i])

        plt.title('Número promedio de clientes en cola') # Colocamos el título
```

```
x1, y1 = zip(* [m for m in lista_clientes_en_cola])
p1 = plt.plot(x1, y1,'ro-', markersize=0.5, lw=0.5,color='g')
# lambda=5 mu =8
# x2, y2 = zip(* [m for m in lista_clientes_en_cola_2])
# p2 = plt.plot(x2, y2,'ro-', markersize=0.5, lw=0.5,color='r')
# plt.legend((p1[0], p2[0]), ('lambda=2; mu=3', 'lambda=5; mu=9'))
plt.plot([promedio_clientes_en_cola for i in range(n)], linestyle='dashed', color='blue')
plt.grid(True)
plt.show()

x, y = zip(* [m for m in lista_demora_en_cola])
plt.title('Número promedio de demora en cola') # Colocamos el título
plt.plot(x, y,'ro-', markersize=0.5, lw=0.5,color='g')
plt.plot([promedio_demora_en_cola for i in range(n)], linestyle='dashed', color='blue')
plt.grid(True)
plt.show()

x, y = zip(* [m for m in lista_utilizacion_servidor])
plt.title('Utilización promedio del servidor') # Colocamos el título
plt.plot(x, y,'ro-', markersize=0.5, lw=0.5,color='g')
plt.plot([promedio_utilizacion_servidor for i in range(n)], linestyle='dashed', color='blue')
plt.grid(True)
plt.show()
```

Librería gráficos en tiempo real:

```
from pyqtgraph.Qt import QtGui, QtCore
import pyqtgraph as pg

# Número de clientes en cola
app = QtGui.QApplication([])
win = pg.GraphicsWindow(title="Gráfica en tiempo real")

p = win.addPlot(title="Número de clientes en cola")
curva = p.plot(pen='y')
p.setRange(yRange=[0, 15])

p2 = win.addPlot(title=Ütilización del servidor")
curva2 = p2.plot(pen='y')
p2.setRange(yRange=[0, 2])

def Update():
  global curva, reloj_sim, num_cli_sim, servidor_sim

  # Actualizamos reloj_sim y num_cli_sim

  curva.setData(reloj_sim, num_cli_sim)
  curva2.setData(reloj_sim, servidor_sim)

  QtGui.QApplication.processEvents()

pg.QtGui.QApplication.exec_()
```

BIBLIOGRAFÍA

- Simulation, modeling and analysis - Law, Kelton.
- Investigación de Operaciones - 7ma edición - Hamdy A. Taha.
- Presentaciones de la cátedra.