Girish Kumar
C1151988

CM1103 Coursework

Report

CM1103 Coursework – Girish Kumar (C1151988)

16th December 2011

Question 1

1. Consider a bank with a single teller and queue. The manager would like to investigate the improvement in quality of service that would arise through reducing the average time taken to service individual customers.

   (a) Implement step 2 of the guidance notes in a function `nextTime` which takes an argument *mean* and returns a corresponding random interarrival time. Verify through simulation (using this function) that for a large number of samples, the technique in step 2 above gives an average interarrival time of α for $\alpha = 1, 10, 50$ and 100 minutes.

Answer: Source Code

```
import math
import random

random.seed()

def nextTime(mean):
    return -mean * math.log(1 - random.random())

def question1(alpha, maxi):
    avg,suma = 0,0
    for i in range(maxi):
        suma = suma + nextTime(alpha)
    avg = suma/maxi
    print avg
```

OUTPUT

```
>>> ================================ RESTART ================================
>>>
>>> question1(1,100000)
1.00334400876
>>> question1(10,10000000)
10.000247363
>>> question1(50,10000000)
50.0070256224
>>> question1(100,100000000)
100.009821252
>>>
```

(b) Implement the algorithm from step 4 of the guidance notes, and simulate a single queue with a single teller over an 8-hour day in order to calculate the *maximum queue length* for the cases:

- $\alpha = 10, \beta = 4$
- $\alpha = 10, \beta = 2$
- $\alpha = 5, \beta = 4$
- $\alpha = 5, \beta = 2$

Your results should be averaged over a number of simulations to account for random variations.

Answer: Source Code

```
import math
import random

random.seed()

def nextTime(mean):
    return -mean * math.log(1 - random.random())

def question2(alpha, beta, maxi):
    sumq = 0.0
    for i in range(maxi):
        queuelist = []
        ta, ts, c, Q = nextTime(alpha), nextTime(beta), 0, 1

        # 8 hours = 480 minutes
        while (c <= 480):
            if (ta < ts):
                ts -= ta
                c += ta
                Q += 1
                ta = nextTime(alpha)
            else:
                ta -= ts
                c += ts
                Q -= 1
                ts = nextTime(beta)
            while (Q == 0):
                c += ta
                Q += 1
                ta = nextTime(alpha)
            queuelist.append(Q)
        #print average sum of max(queuelist)
        sumq = sumq + max(queuelist)
    print sumq/maxi
```

OUTPUT

```
>>> ================================ RESTART ================================
>>>
>>> question2(10,4,10000)
4.2954
>>> question2(10,2,10000)
3.0079
>>> question2(5,4,10000)
10.6455
>>> question2(5,2,10000)
5.0713
>>>
```

Comparing two cases with multiple reputations:

	Alpha	Beta	Reputations	Avg. max queue length
CASE I	10	4	100	4.22
	10	4	1000	4.373
	10	4	10000	4.3515
	10	4	100000	4.32942
CASE II	5	2	100	5.14
	5	2	1000	5.069
	5	2	10000	5.0444
	5	2	100000	5.047483

*Since we are using random.random() function it is clearly shown that the more reputations we have the more closer to the shorter range of the answer

Question 2

2. Consider a second branch of the bank that has two classes of customer; *business* and *consumer*. Assume there is a single distribution of interarrival times for all customers, with mean $\alpha = 5$, but that business customers make up a quarter of the custom. The mean service time for consumers is 3 minutes.

   (a) Modify the algorithm appropriately, and compare using simulation:

   - Two queues (one for each teller), where each new customer joins the shortest queue when they arrive.
   - Two queues, one reserved for business customers and one for consumers.

   for the cases where the mean service time for business customers is 5, 10 and 20 minutes.

Answer: Source Code

```
import os
import sys

import math
import random

random.seed()

# arrival time basicially setted constant to 5(=alpha) (as given in question)
def arrivalTime():
    mean=5
    return -mean * math.log(1 - random.random())

# serving time for Consumer Class: (as default delared as 3 for consumers)
def serveTime1():
    return -3 * math.log(1 - random.random())

# serving time for Business Class: (mean is passed as argument)
def serveTime2(mean):
    return -mean * math.log(1 - random.random())
```

```
def main(bus,maxi):
    sumq1 = 0.0
    sumq2 = 0.0
    for i in range(maxi):
        queues1 = []
        queues2 = []
        fq=[]
        ta = arrivalTime()
        ts1 = serveTime1()
        ts2 = serveTime2(bus)
        c = 0
        Q1 = 1
        Q2 = 1

        while (c <= 480):
            # if time arrive is less for service time 1 and 2
            if (ta < ts1 and ta < ts2): # 2 BUSY CASE
                # if two tellers are busy so join the queue
                ts1 = ts1 - ta
            ts2 = ts2 - ta
            c = c + ta
            ta = arrivalTime()
            if (Q1 < Q2):
                Q1 = Q1 + 1
            else:
                Q2 = Q2 + 1
          # if NOT
          else:
              # time arrive > service time 1 and < service time 2
              if (ta > ts1 and ta < ts2): # 1 FREE and 1 BUSY CASE
                  ta = ta - ts1
                  c = c + ts1
                  Q1 = Q1 - 1
                  ts1 = serveTime1()
              # or time arrive < service time 1 and > service time 2
              elif (ta < ts1 and ta > ts2): # 1 BUSY and 1 FREE CASE
                  ta = ta - ts2
                  c = c + ts2
                  Q2 = Q2 - 1
                  ts2 = serveTime2(bus)
              # if not for above two cases. and checks whether service time for
              # 1 is greater than 2 and either cases
              else:
                  if (ts1 > ts2):
                      ta = ta - ts2
                      c = c + ts2
                      ts2 = serveTime2(bus)
                  else:
                      ta = ta - ts1
                      c = c + ts2
                      ts1 = serveTime1()
                  Q1 = Q1 - 1
                  Q2 = Q2 - 1

          # finally checks wheter the queue is empty
          if (Q1 == 0):
              # stops two queues to a new customer arrival
              if (Q2 > 0):
                  c = c + ta
                  ta = arrivalTime()
```

```
            Q1 = Q1 + 1
        if (Q2 == 0):
            c = c + ta
            Q2 = Q2 + 1
            ta = arrivalTime()

        queues1.append(Q1)
        queues2.append(Q2)
    fq.append(max(queues1))
    fq.append(max(queues2))
    sumq1 = sumq1 + min(fq)
print sumq1/maxi
```

OUTPUT

```
>>> ================================ RESTART ================================
>>>
>>> main(5,10000)
3.1898
>>> main(10,10000)
3.7399
>>> main(20,10000)
4.3
>>>
```

(b) Show what happens if the following modification was made to the process. If the business queue is empty, the business teller will call over and deal with the next consumer customer.

Answer: Source Code

```
import os
import sys

import math
import random

random.seed()

# arrival time basicially setted constant to 10
def arrivalTime():
    mean=10
    return -mean * math.log(1 - random.random())

# serving time for Consumer Class: (as default delared as 6 for consumers)
def serveTime():
    return -6 * math.log(1 - random.random())
```

```
def main(maxi):
    sumq = 0.0
    for i in range(maxi):
        queues = []
        fq=[]
        ta = arrivalTime()
        ts1 = serveTime()
        ts2 = serveTime()
        c = 0
        Q = 1

        while (c <= 480):
            # if time arrive is less for service time 1 and 2
            if (ta < ts1 and ta < ts2):
                # if two tellers are busy so join the queue
                ts1 = ts1 - ta
                ts2 = ts2 - ta
                c = c + ta
                ta = arrivalTime()
                Q = Q + 1
            # if NOT
            else:

              # time arrive > service time 1
              # one teller is free to do service
               if (ta > ts1):
                  if (ts1 < ts2):
                     ta = ta - ts1
                     c = c + ts1
                  else:
                     ta = ta - ts2
                     c = c + ts2
                  ts1 = serveTime()
                  Q = Q - 1
                # or time arrive > service time 2
               if (ta > ts2):
                   if (ts1 < ts2):
                      ta = ta - ts1
                      c = c + ts1
                   else:
                      ta = ta - ts2
                      c = c + ts2
                   ts2 = serveTime()
                   Q = Q - 1
```

```
            # finally checks wheter the queue is empty
            while (Q <= 0):
                c = c + ta
                Q = Q + 1
                ta = arrivalTime()

            queues.append(Q)
        sumq = sumq + max(queues)
    print sumq/maxi
```

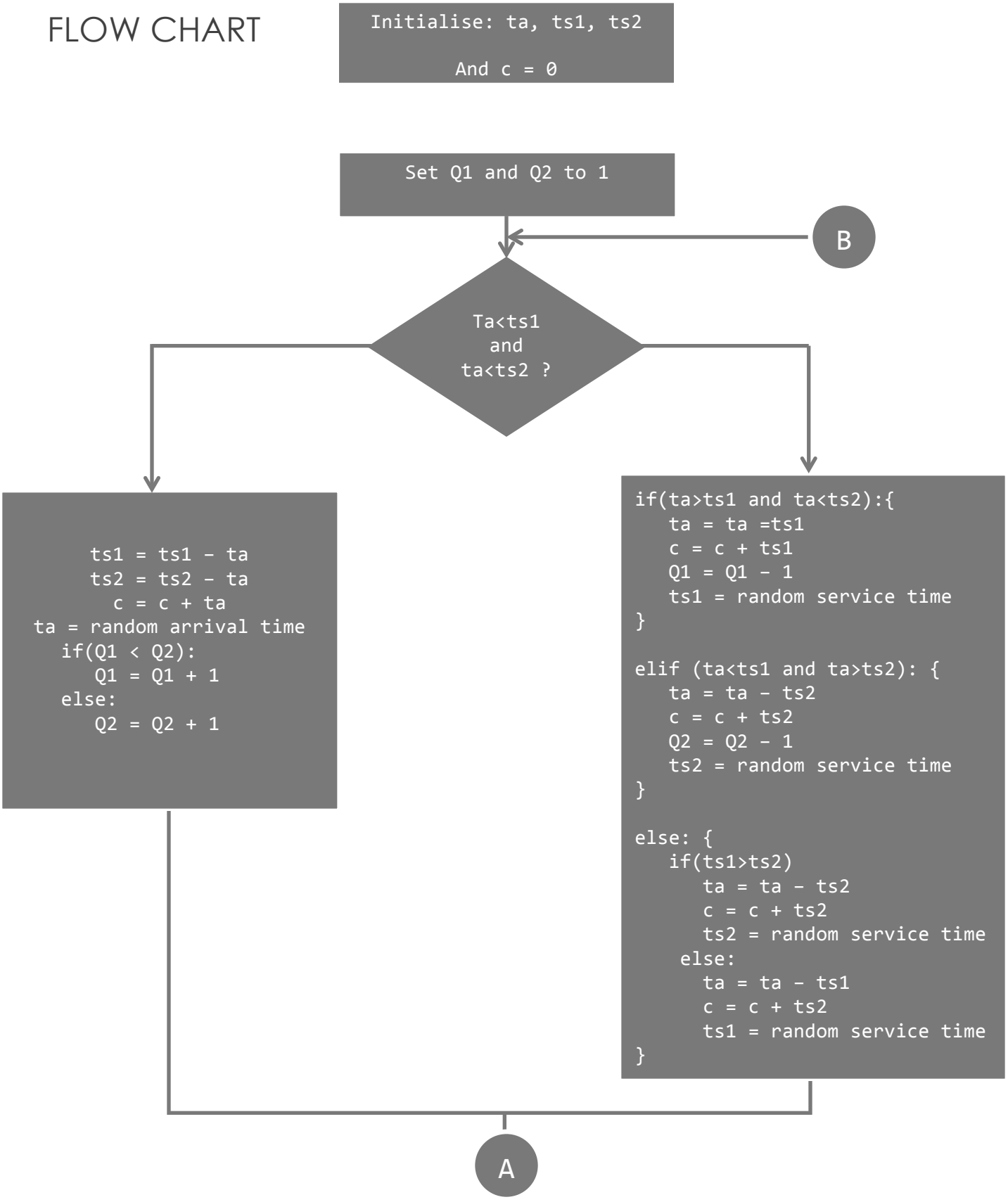
OUTPUT

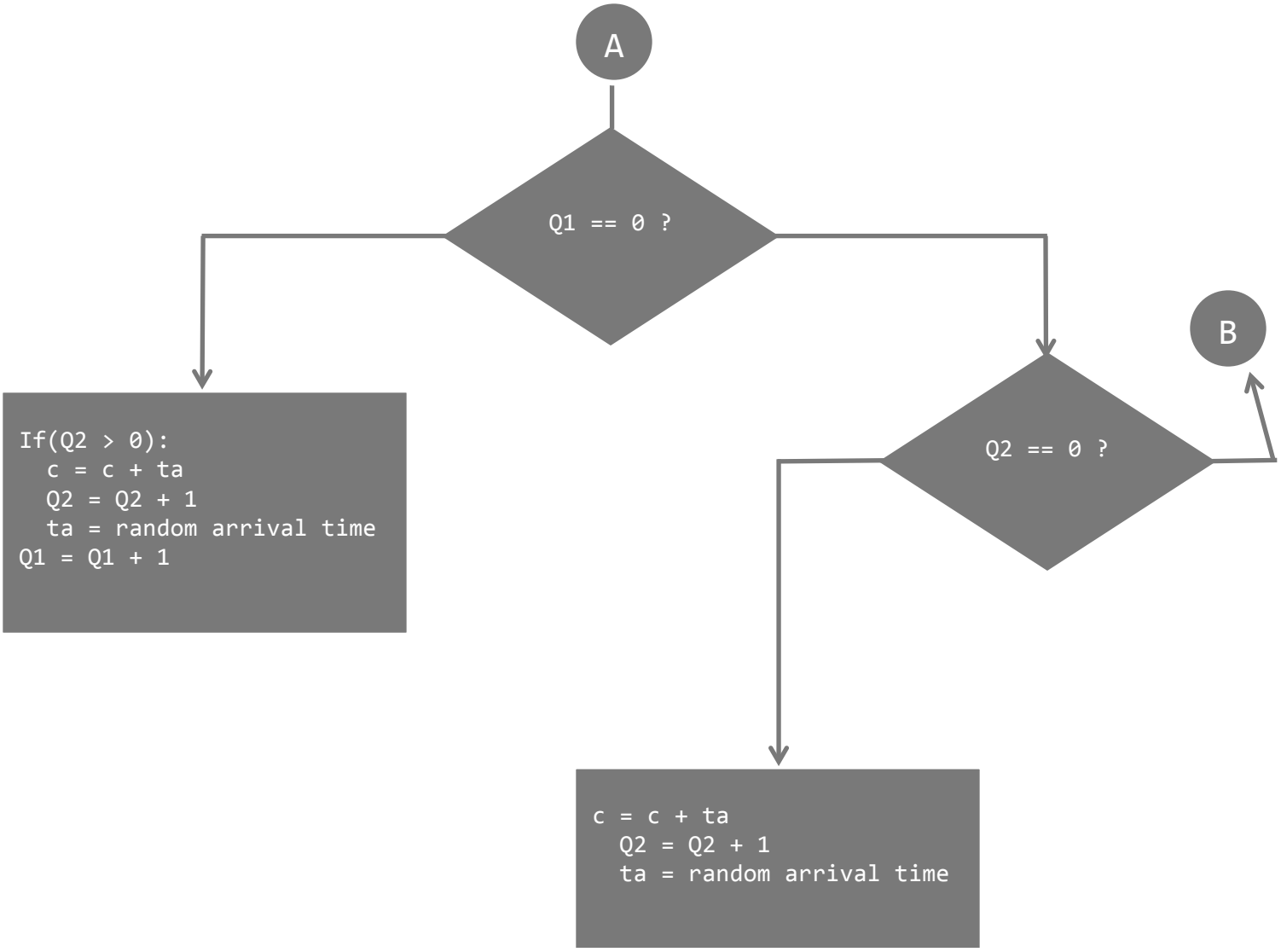
```
>>> ================================ RESTART ================================
>>>
>>> main(10000)
4.0601
>>> main(10000)
4.0539
>>>
```

(c) How does the *mean* waiting time vary in the above scenarios?

In the case two (one queue and two teller) we first check for whether teller machine is free. If the 1st teller is not free – check whether the 2nd teller is free, else we will increase Q by 1 but in the case one (two teller and two queues) we first look for the shortest queue rather than availability of teller machine. In the program point of view the mean waiting time varies for the two cases since the approach of the solving is different for each cases.

FLOW CHART


Initialise: ta, ts1, ts2
And c = 0
Set Q1 and Q2 to 1
B
Ta<ts1 and ta<ts2 ?
ts1 = ts1 – ta
ts2 = ts2 – ta
c = c + ta
ta = random arrival time
if(Q1 < Q2):
Q1 = Q1 + 1
else:
Q2 = Q2 + 1
if(ta>ts1 and ta<ts2):{
ta = ta =ts1
c = c + ts1
Q1 = Q1 – 1
ts1 = random service time
}
elif (ta<ts1 and ta>ts2): {
ta = ta – ts2
c = c + ts2
Q2 = Q2 – 1
ts2 = random service time
}
else: {
if(ts1>ts2)
ta = ta – ts2
c = c + ts2
ts2 = random service time
else:
ta = ta – ts1
c = c + ts2
ts1 = random service time
}
A


A
Q1 == 0 ?
If(Q2 > 0):
c = c + ta
Q2 = Q2 + 1
ta = random arrival time
Q1 = Q1 + 1
B
Q2 == 0 ?
c = c + ta
Q2 = Q2 + 1
ta = random arrival time