เอกสารประกอบการบรรยาย

Data Structure

Queue

Queue

เนื้อหา

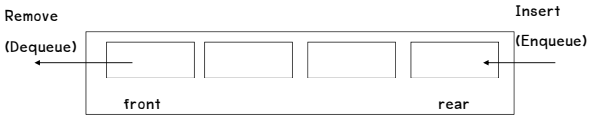
- โครงสร้างข้อมูลแบบคิว
- การทำงานของคิว
- การแทนที่ข้อมูลของคิว
- การประยุกต์ใช้คิว

Queue

คิว (Queue) เป็นโครงสร้างข้อมูลแบบเชิงเส้นหรือลิเนียร์ลิสต์ซึ่งการเพิ่มข้อมูลจะกระทำที่ปลายข้างหนึ่งซึ่งเรียกว่าส่วนท้ายหรือเรียร์ (rear) และการนำข้อมูลออกจะกระทำที่ปลายอีกข้างหนึ่งซึ่งเรียกว่า ส่วนหน้าหรือฟรอนต์ (front)

ลักษณะการทำงานของคิวเป็นลักษณะของการเข้าก่อน ออกก่อนหรือที่เรียกว่า FIFO (First In First Out)

Queue(Cont.)



ลักษณะสำคัญของการดำเนินการในโครงสร้างข้อมูลแบบคิว คือ การนำเข้าข้อมูลด้วยการต่อท้าย(Enqueue) และการดึงข้อมูลออกในส่วนหัว(Dequeue)

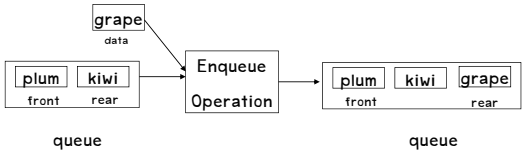
Enqueue

การทำงานของคิว

การใส่สมาชิกตัวใหม่ลงในคิว เรียกว่า Enqueue ซึ่งมีรูปแบบคือ

enqueue (queue, newElement)

หมายถึง การใส่ข้อมูล newElement ลงไปที่ส่วนเรียร์ของคิว



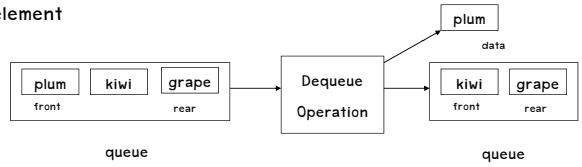
แสดงการเพิ่มข้อมูลเข้าไปในคิว

Dequeue

การนำสมาชิกออกจากคิว เรียกว่า Dequeue ซึ่งมีรูปแบบคือ

dequeue (queue, element)

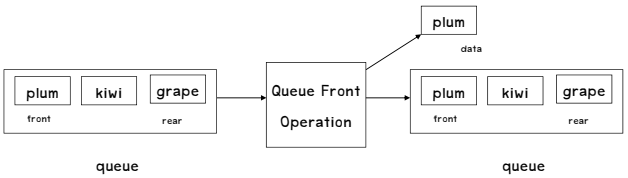
หมายถึง การนำออกจากส่วนหน้าของคิวและให้ข้อมูลนั้นกับ element



แสดงการนำข้อมูลออกจากคิว

Front

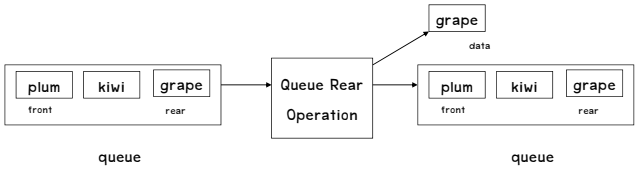
การนำข้อมูลที่อยู่ตอนต้นของคิวมาแสดงจะ เรียกว่า Queue Front แต่จะไม่ทำการเอาข้อมูลออกจากคิว



แสดงการนำข้อมูลตอนต้นของคิวมาแสดง

Rear

การนำข้อมูลที่อยู่ตอนท้ายของคิวมาแสดงจะเรียกว่า Queue Rear แต่จะไม่ทำการเพิ่มข้อมูลเข้าไปในคิว

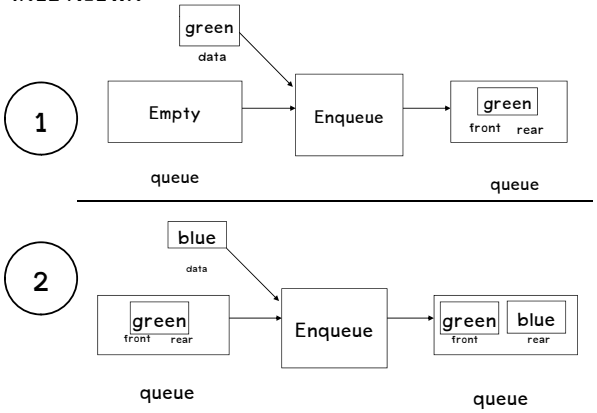


แสดงการนำข้อมูลตอนท้ายของคิวมาแสดง

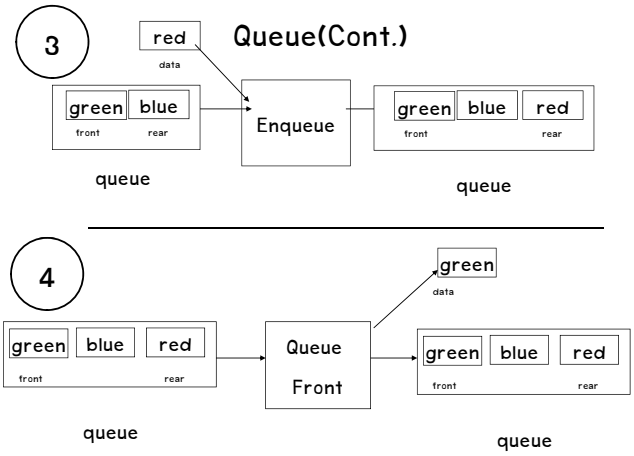
ตัวอย่าง การนำข้อมูล 3 ข้อมูล green, blue, red เข้าสู่คิว และนำออกตามการดำเนินการ จะได้ขั้นตอนดังนี้

1. Enqueue(Q,green)
2. Enqueue(Q,blue)
3. Enqueue(red)
4. Front(Q)
5. Rear(Q)
6. Dequeue(Q,green)
7. Dequeue(Q,blue)
8. Dequeue(Q,red)

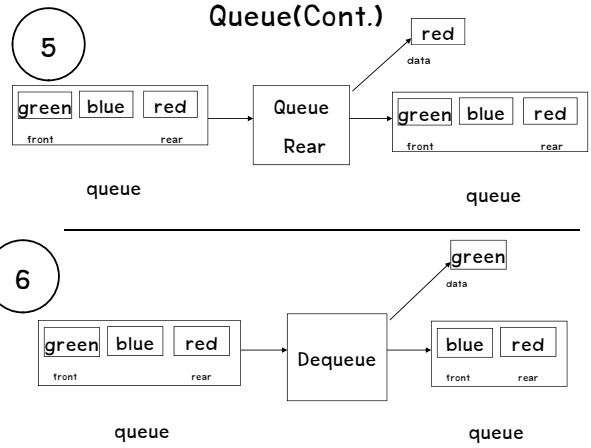
ตัวอย่างของคิว



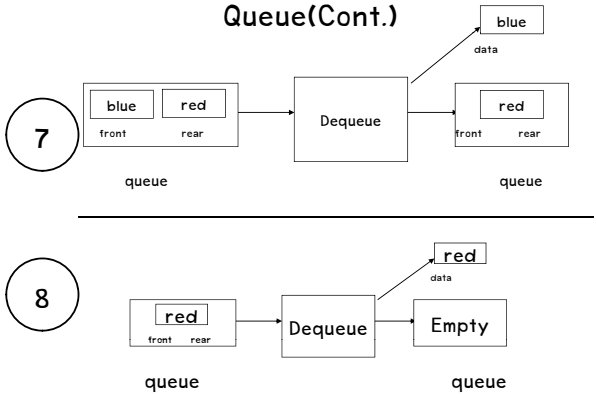
Queue(Cont.)



Queue(Cont.)



Queue(Cont.)



Queue (Cont.)

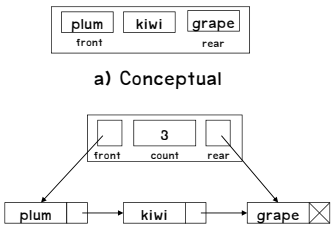
การแทนที่ข้อมูลของคิว

การแทนที่ข้อมูลของคิว สามารถทำได้ 2 วิธี คือ

1. การแทนที่ข้อมูลของคิวแบบลิงค์ลิสต์
2. การแทนที่ข้อมูลของคิวแบบอะเรย์

Queue (Cont.)

การแทนที่ข้อมูลของคิวแบบลิงค์ลิสต์

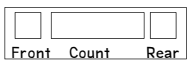


Queue (Cont.)

การแทนที่ข้อมูลของสแตกแบบลิงค์ลิสต์ จะประกอบไปด้วย 2 ส่วน คือ

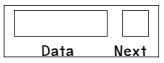
1. Head Node จะประกอบไปด้วย 3 ส่วนคือ พอยเตอร์จำนวน 2 ตัว คือ Front และ rear กับจำนวนสมาชิกในคิว
2. Data Node จะประกอบไปด้วยข้อมูล (Data) และพอยเตอร์ที่ชี้ไปยังข้อมูลตัวถัดไป

Queue (Cont.)



Head Node

```
queueHead
   front    <node pointer>
   count    <integer>
   rear     <node pointer>
end queueHead
```



Data Node

```
node
   data     <data type>
   next     <node pointer>
end node
```


Queue (Cont.)

การดำเนินการเกี่ยวกับคิว

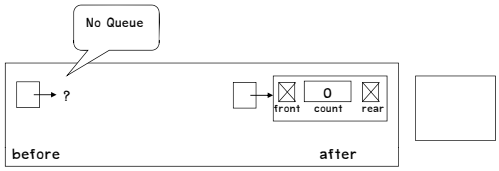
การดำเนินการเกี่ยวกับคิว ได้แก่

1. Create Queue
2. Enqueue
3. Dequeue
4. Queue Front
5. Queue Rear
6. Empty Queue
7. Full Queue
8. Queue Count
9. Destroy Queue

Queue (Cont.)

1. Create Queue

จัดสรรหน่วยความจำให้แก่ Head Node และให้ค่า pointer ทั้ง 2 ตัวมีค่าเป็น null และจำนวนสมาชิกเป็น0



Queue (Cont.)

Algorithm CreateQueue

Pre Nothing

Post Head has been allocated and initialized

Return Head's address if successful, null if voerflow

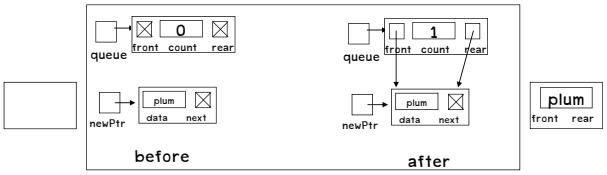
1. if (memory available)
   1 allocate (newPrt)
   2 newPtr->front = null pointer
   3 newPtr->rear = null pointer
   4 newPtr->count = 0
   5 return newPtr
2. Else
   1 return null pointer

End CreateQueue

Queue (Cont.)

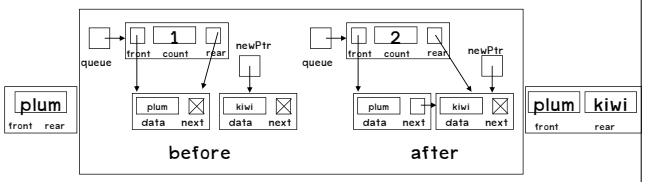
2. Enqueue

การเพิ่มข้อมูลเข้าไปในคิว



กรณีที่ไม่มีข้อมูลอยู่ในคิว

Queue (Cont.)



กรณีที่มีข้อมูลอยู่ในคิว

Algorithm EnQueue

Pre Queue has been create

Post Item data have been inserted

Return Boolean; True: if successful, False if overflow

---

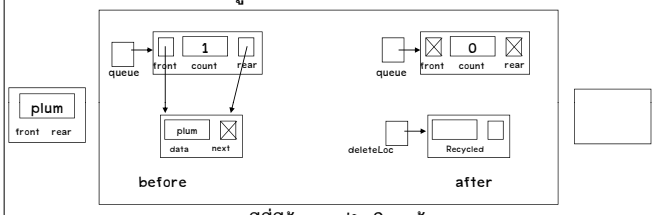
1. if (queue full)
   1. return false
2. else
   1 allocate(newPtr)
   2 newPtr->data = item
   3 newPtr->next = null pointer
   4 if (queue->count zero)
      1 queue->front = newPtr
   5 else
      1 queue->rear->next = newPtr
   6 queue->rear = newPtr
   7 queue->count = queue->count+1
   8 return true

End EnQueue

Queue (Cont.)

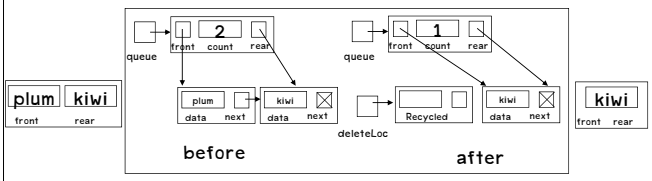
3. Dequeue

การนำข้อมูลออกจากคิว



กรณีที่มีข้อมูลอยู่ในคิว 1 ข้อมูล

Queue (Cont.)



กรณีที่มีข้อมูลอยู่ในคิวมากกว่า 1 ข้อมูล

Algorithm DeQueue

Pre Queue has been create

Post Data at front of queue returned to user through item and front element deleted and recycled

Return Boolean; True: if successful, False if underflow

---

1. if (queue->count is 0)
   1 return false
2. else
   1 item = queue->front->data
   2 deleteLoc = queue->front
   3 if (queue->count is 1)
      1 queue->rear = null pointer
   4 queue->front = queue->front->next
   5 queue->count = queue->count-1
   6 recycle(deleteLoc)
   7 return true

End Dequeue

Queue (Cont.)

4. Queue Front

เป็นการนำข้อมูลที่อยู่ส่วนต้นของคิวมาแสดง

Algorithm QueueFront

Pre Queue is a pointer to an initialized queue

Post Data pass back to caller

Return Boolean; True: successful, False if underflow

Queue (Cont.)

5. Queue Rear

เป็นการนำข้อมูลที่อยู่ส่วนท้ายของคิวมาแสดง

Algorithm QueueRear

Pre Queue is a pointer to an initialized queue

Post Data pass back to caller

Return Boolean; True: successful, False if underflow

Queue (Cont.)

6. Empty Queue

เป็นการตรวจสอบว่าคิวว่างหรือไม่

Algorithm EmptyQueue

Pre Queue is a pointer to a queue head node

Return Boolean; True: if empty, False if queue has data

1. Return (queue->count equal 0)

End EmptyQueue

Queue (Cont.)

7. Full Queue

เป็นการตรวจสอบว่าคิวเต็มหรือไม่

Algorithm FullQueue

Pre Queue is a pointer to a queue head node

Return Boolean; True: if full, False if room for another node

1. allocate (tempPtr)
2. if (allocation successful)
   1 release (tempPtr)
   2 return false
3. else
   1 return true

End FullQueue

Queue (Cont.)

8. Queue Count

เป็นการนับจำนวนสมาชิกที่อยู่ในคิว

Algorithm QueueCount

Pre Queue is a pointer to the queue head node

Return Queue count

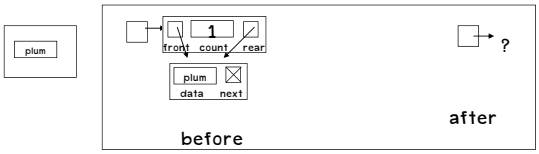
1. Return (queue->count)

End QueueCount

Queue (Cont.)

9. Destroy Queue

เป็นการลบข้อมูลทั้งหมดที่อยู่ในคิว



Queue (Cont.)

Algorithm DestroyQueue

Pre Queue is valid queue

Post All data have been deleted and recycled

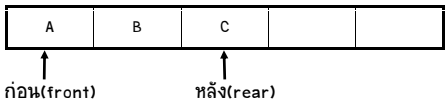
Return null pointer

1. pWalker = queue->front
2. Loop(pWalker not null)
   1 deletePtr = pWalker
   2 pWalker = pWalker->next
   3 recycle (deletePtr)
   4 recycle (queue)
   5 return null pointer

End DestroyCount

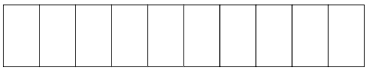
การแทนที่ข้อมูลของคิวแบบอะเรย์

- การแทนโครงสร้างคิวด้วยอาร์เรย์จะทำให้สามารถกำหนดจำนวนของการจองพื้นที่บนหน่วยความจำได้
- การแทนคิวด้วยอาร์เรย์นั้นจะต้องมีการระบุจำนวนสูงสุดของพื้นที่เก็บข้อมูล(Maxsize) พร้อมกับกำหนดพอยเตอร์ขึ้นมาอีก 2 ตัว คือ Front และ Rear เพื่อใช้ในการชี้ค่าข้อมูลที่อยู่ส่วนหัวและส่วนท้าย



การแทนที่ข้อมูลของคิวแบบอะเรย์

ในการใส่สมาชิกลงในคิวจะต้องตรวจสอบก่อนว่าคิวเต็มหรือไม่

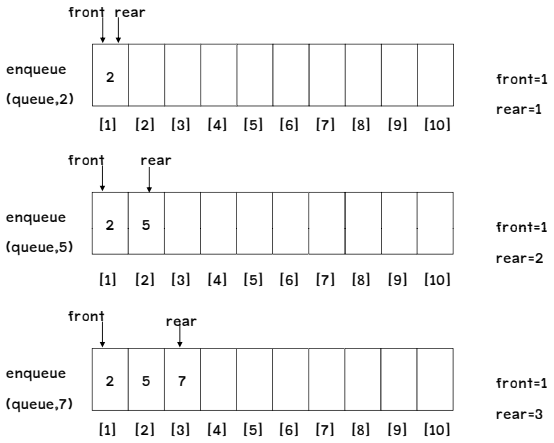


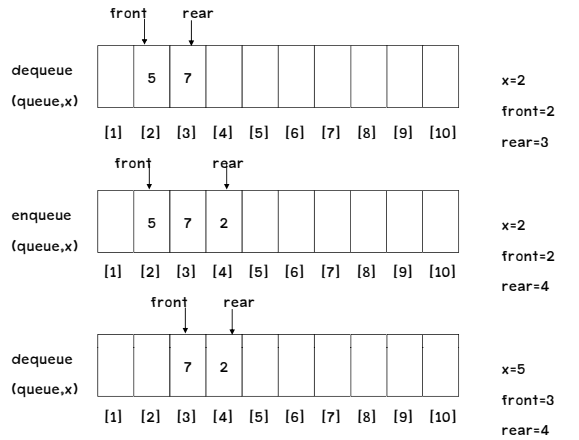
คิวว่าง

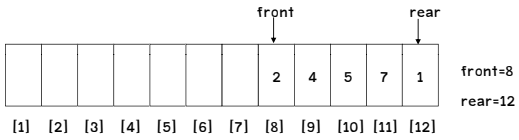
front=rear=null

การนำข้อมูลเข้าสู่คิว จะไม่สามารถนำเข้าในขณะที่คิวเต็มหรือไม่มีที่ว่าง ถ้าพยายามนำเข้าจะทำให้เกิดความผิดพลาดที่เรียกว่า overflow

การนำข้อมูลออกจากคิว จะไม่สามารถนำอะไรออกจากคิวที่ว่างเปล่าได้ ถ้าพยายามจะทำให้เกิดความผิดพลาดที่เรียกว่า underflow

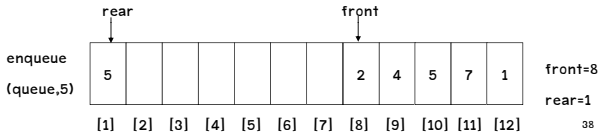






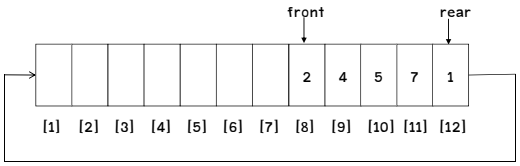
จากตัวอย่าง จะเห็นได้ว่าอาจจะมีปัญหาในการนำเข้าข้อมูล ในกรณีที่คิวเต็ม แต่สภาพความเป็นจริงแล้ว front ไม่ได้อยู่ในช่องแรกของคิว จะไม่สามารถนำที่ว่างในส่วนหน้ามาใช้ได้อีก

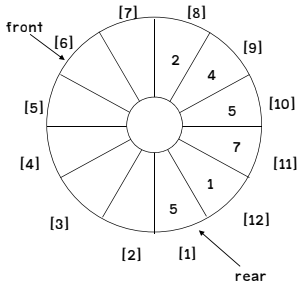
วิธีการแก้ปัญหาดังกล่าว จะใช้วิธีการกำหนดให้คิวเป็นวงกลม(Circular Queue) ซึ่งคิวช่องสุดท้ายนั้นต่อกับคิวช่องแรกสุด



Circular Queue

- เป็นการออกแบบ Queue ขึ้นมาเพื่อใช้ในการแก้ปัญหาในกรณีที่ยังมีพื้นที่เหลือในการเก็บข้อมูลในส่วนหน้า แต่ส่วนท้าย rear=maxsize ซึ่งสามารถ Enqueue ข้อมูลเข้ามายังคิวได้จนกว่าคิวจะเต็มจริงๆ





- ลักษณะโครงสร้างของคิววงกลมจะมีการนำเอาคิวส่วนท้ายมาเชื่อมต่อกับคิวส่วนหัวและเมื่อมีการ enqueue ข้อมูลเข้าจะดำเนินการต่อไปในส่วนของ rear โดยมีทิศทางในลักษณะของการเดินตามเข็มนาฬิกา
- เมื่อมีการ enqueue ข้อมูลเข้ามาอีกก็จะเพิ่มต่อท้ายไปและเซตพอยเตอร์ ชี้ไปยังตำแหน่ง Q[2] และหากมีการ dequeue ข้อมูลก็จะกระทำ ณ ตำแหน่ง Front ทำให้สามารถใช้งานพื้นที่ว่างบนคิวได้

Queue (Cont.)

ในกรณีที่เป็นคิวแบบวงกลม คิวจะเต็มก็ต่อเมื่อมีการเพิ่มข้อมูลเข้าไปในคิวเรื่อย ๆ จนกระทั่ง rear มีค่าน้อยกว่า front อยู่หนึ่งค่า คือ rear = front - 1

Queue (Cont.)

การประยุกต์ใช้คิว

คิวถูกประยุกต์ใช้มากในการจำลองระบบงานธุรกิจ เช่น การให้บริการลูกค้า ต้องวิเคราะห์จำนวนลูกค้าในคิวที่เหมาะสมว่าควรเป็นจำนวนเท่าใด เพื่อให้ลูกค้าเสียเวลาน้อยที่สุด

ในด้านคอมพิวเตอร์ ได้นำคิวเข้ามาใช้ คือ ในระบบปฏิบัติการ (Operation System) ในเรื่องของคิวของงานที่เข้ามาทำงาน(ขอใช้ CPU) จะจัดให้งานที่เข้ามาได้ทำงานตามความสำคัญ

การทำบัฟเฟอร์(Buffering)

- การทำบัฟเฟอร์จะใช้ในกรณีที่อัตราความเร็วในการทำงานของอุปกรณ์คอมพิวเตอร์มีการทำงานด้วยความเร็วที่แตกต่างกัน เช่น การทำงานของเครื่องพิมพ์กับ CPU ซึ่งถือว่ามีการทำงานในอัตราเร็วที่แตกต่างกันมาก แต่เมื่อต้องการส่งผ่านข้อมูลหากัน CPU ซึ่งมีการทำงานด้วยความเร็วจะทำการเก็บการประมวลผลส่งไปยังบัฟเฟอร์ก่อน เมื่อทำงานใดเสร็จบัฟเฟอร์ก็จะส่งต่อการทำงานให้เครื่องพิมพ์ทำงานจนกว่าจะหมดข้อมูลในบัฟเฟอร์สำหรับการทำงานของ CPU และการทำงานของเครื่องพิมพ์นั้น

แบบฝึกหัด

1. ให้นศ.ยกตัวอย่างลำดับคิวที่มีการจัดลำดับความสำคัญที่พบเห็นในชีวิตประจำวัน พร้อมอธิบายถึงสาเหตุที่ได้รับการจัดลำดับก่อน(ห้ามซ้ำกัน)
2. คิวขนาด 7 ค่ามีรายการต่อไปนี้

   front = 2 , rear = 5

   Queue[] = {null, null, “ขนุน”, “ส้มโอ”,”แตงโม”, “ฝรั่ง”, null}

   จงแสดงผลของคิวแบบวงกลม โดยมีค่าเริ่มต้นต่ำแหน่งที่ 0 รวมทั้ง front และ rear เมื่อมีการกระทำต่อไปนี้

   ก. เพิ่ม “มะม่วง” ข. นำ 2 รายการออกจากคิว

   ค. เพิ่ม “ทุเรียน” ง.เพิ่ม “องุ่น”

   จ. นำ 2 รายการออกจากคิว ฉ. เพิ่ม “ละมุด”