Chương 2_3:

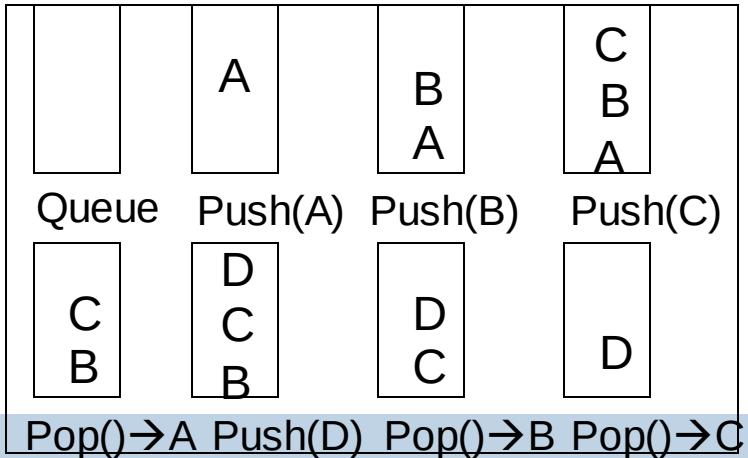
QUEUE – HÀNG ĐỢI

Nội dung

- Hàng đợi (Queue)
  - Khái niệm Queue
  - Các thao tác trên Queue
  - Hiện thực Queue
  - Ứng dụng Queue

Queue - Khái niệm

- Queue là một danh sách mà các đối tượng được **thêm vào** ở một đầu của danh sách và **lấy ra** ở một đầu kia của danh sách. (A *queue is also a list of elements with insertions permitted at one end and deletions permitted from the other end)*
- Việc thêm một đối tượng luôn diễn ra ở **cuối** Queue và việc lấy ra một đối tượng luôn diễn ra ở **đầu** Queue
- Vì thế, thao tác trên Queue được thực hiện theo cơ chế FIFO (First In First Out - *Vào trước ra trước*)



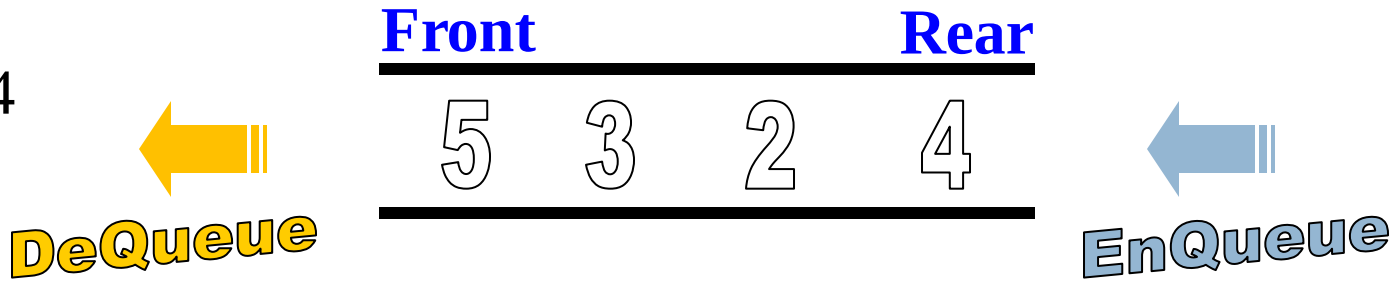
Queue - Khái niệm

Imaging

Queue – Các thao tác

- Queue hỗ trợ 2 thao tác chính:
  - **AddQueue**(): **Thêm** đối tượng vào cuối (rear) Queue
  - **RemoveQueue**(): **Lấy** đối tượng ở đầu (front) Queue
- Ví dụ:

  5 3 2 - - 4



- Queue còn hỗ trợ các thao tác:
  - **isEmpty**(): Kiểm tra xem Queue có rỗng không
  - **Front**(): Trả về giá trị phần tử nằm ở đầu Queue mà không hủy nó. Nếu Queue rỗng thì lỗi sẽ xảy ra

Queue – Hiện thực Queue
(Implementation of a Queue)

Mảng 1 chiều

Kích thước queue khi quá thiếu, lúc quá thừa

AddQueue/Remove Queue khá phức tạp

Danh sách LK

Cấp phát động!

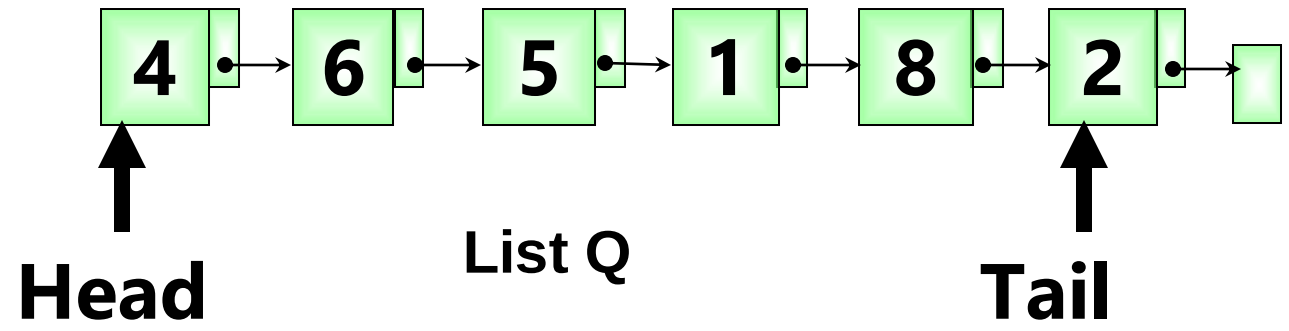
AddQueue/Remove Queue khá dễ dàng

Cài đặt Queue

- Dùng mảng 1 chiều


```
Data   S [N];
int    f,r;
```

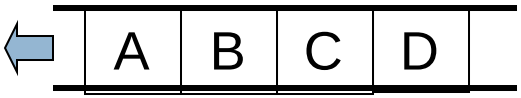
- Dùng danh sách liên kết đơn



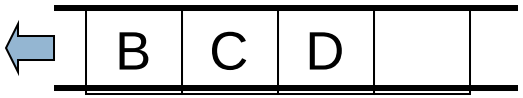
* Thêm và hủy Khác phía

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

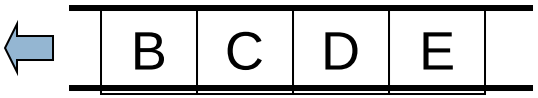
- Hai cách hiện thực:
  - **Khi lấy một phần tử ra thì đồng thời dời các ô phía sau nó lên một vị trí:**



Ban đầu

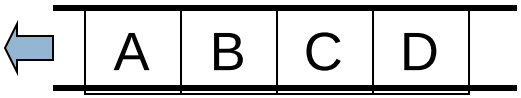


Lấy ra 1 phần tử:
dời tất cả về trước để
trống chỗ thêm vào

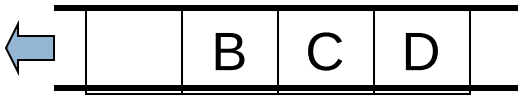


Thêm vào 1 phần tử

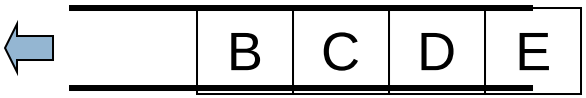
  - **Khi lấy một phần tử ra thì không dời ô lên (xoay vòng):**



Ban đầu



Lấy ra 1 phần tử

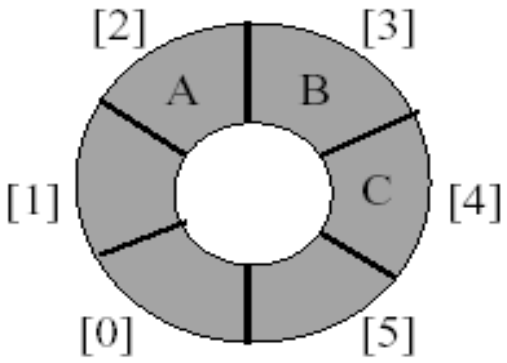


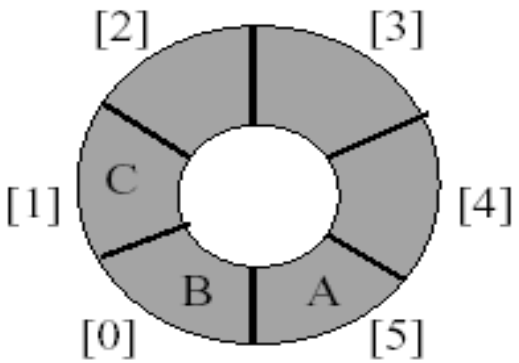
Thêm vào 1 phần tử

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

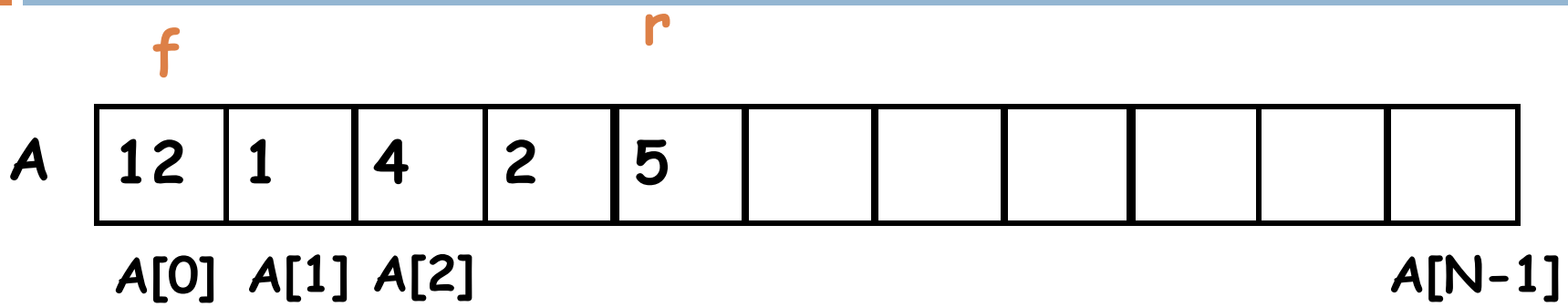
- Trạng thái Queue lúc bình thường:

- Trạng thái Queue lúc xoay vòng:





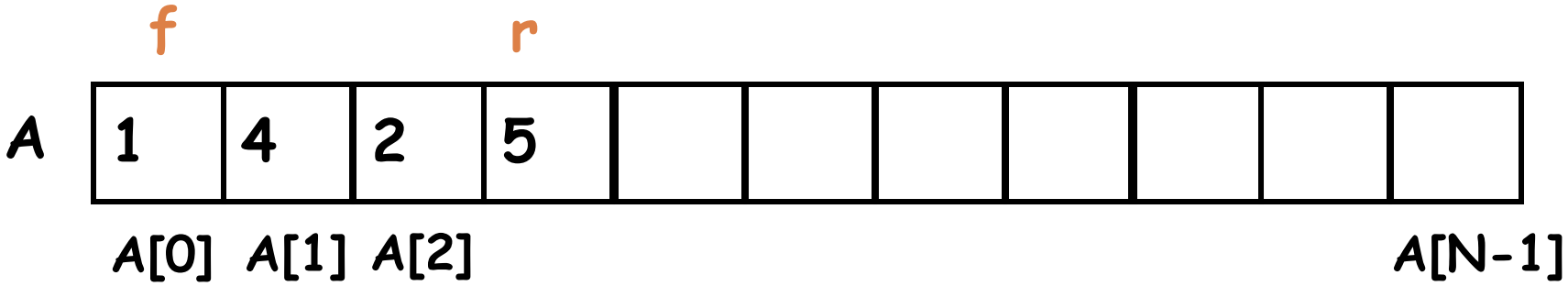
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)

Cách dùng mảng 1

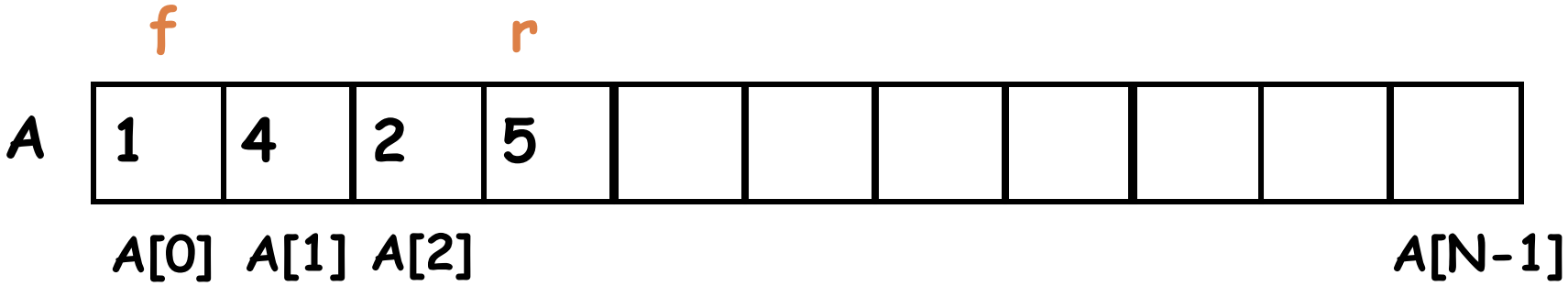
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)

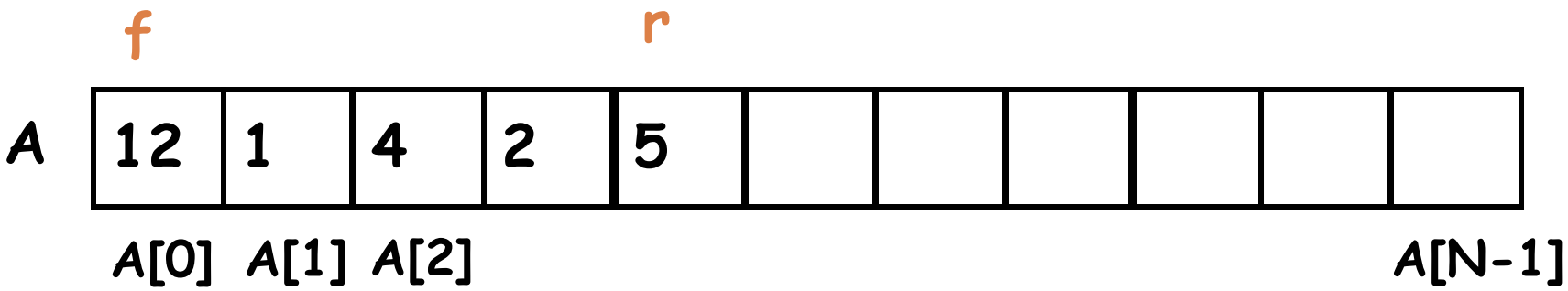
Cách dùng mảng 1

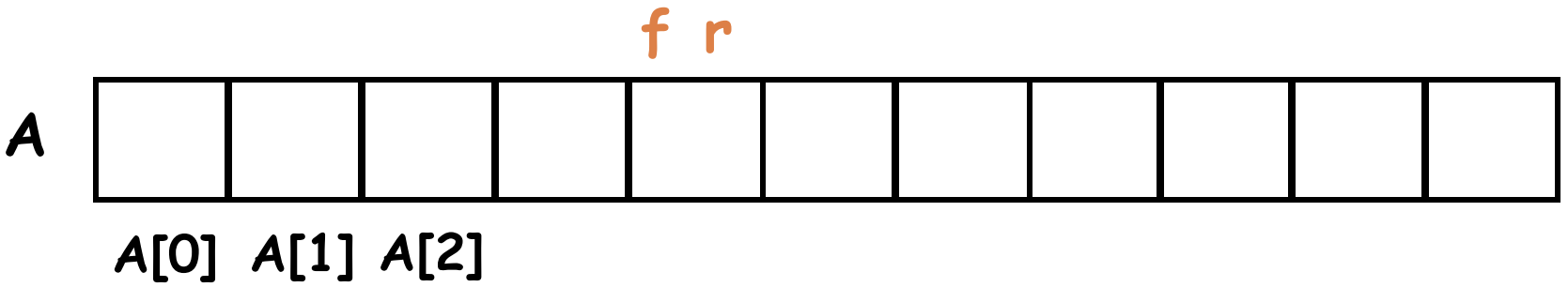
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



Cách dùng mảng 1

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

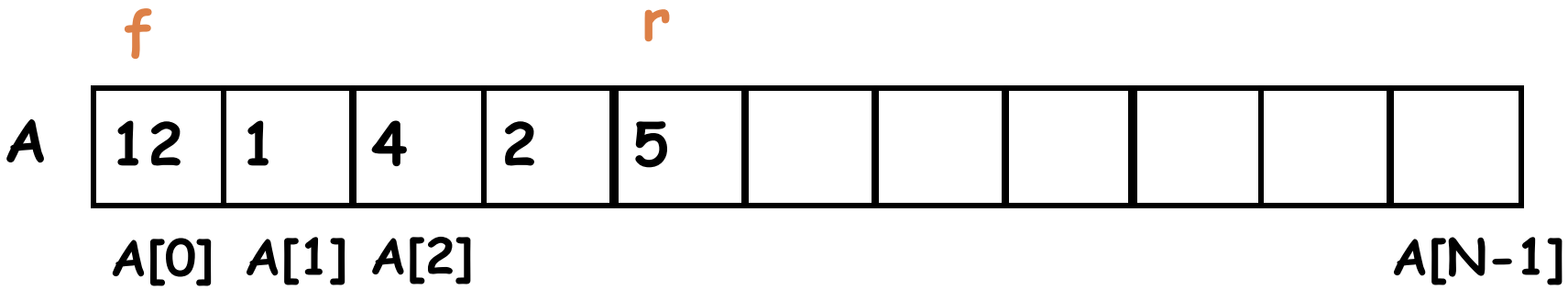




Cách dùng mảng 2

Empty queue f=r

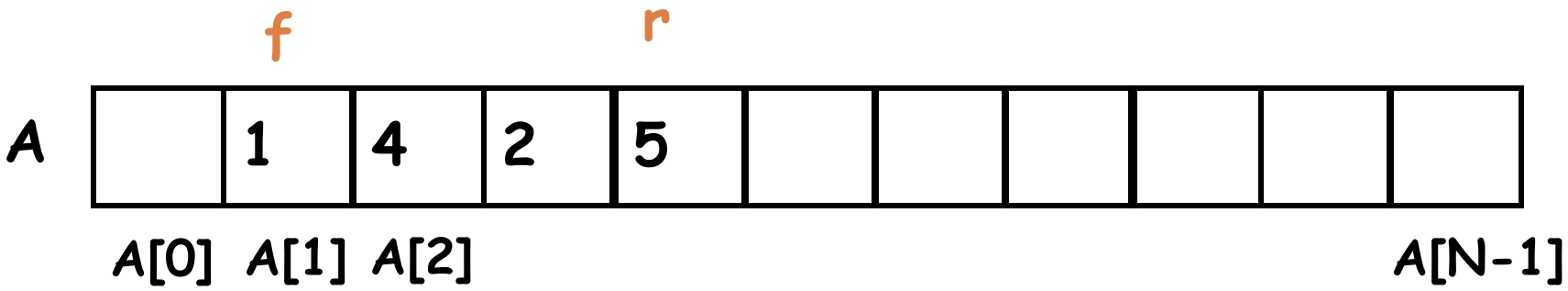
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)

Cách dùng mảng 2

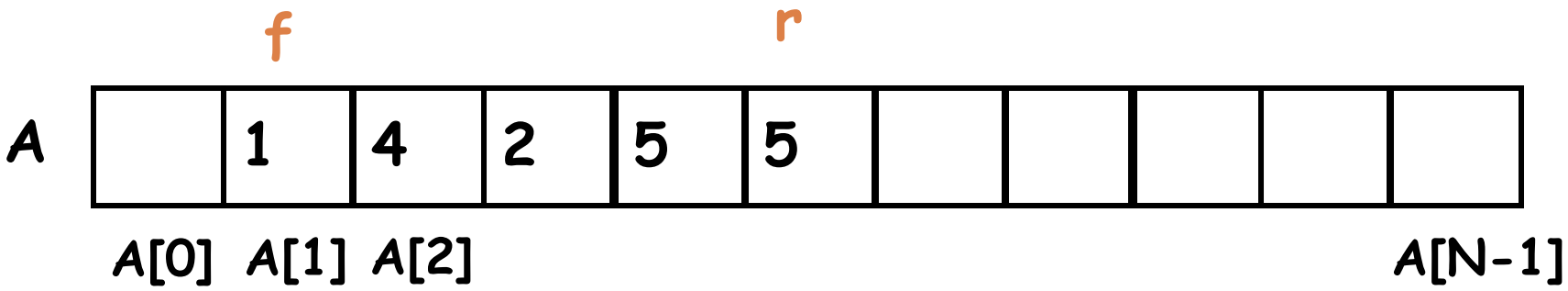
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)
EnQueue(5,Q)

Cách dùng mảng 2

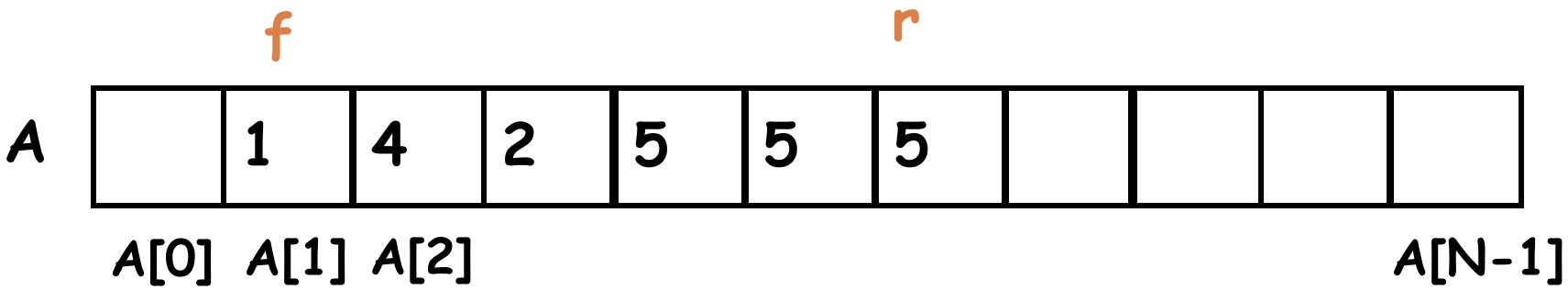
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)
EnQueue(5,Q)
EnQueue(5,Q)

Cách dùng mảng 2

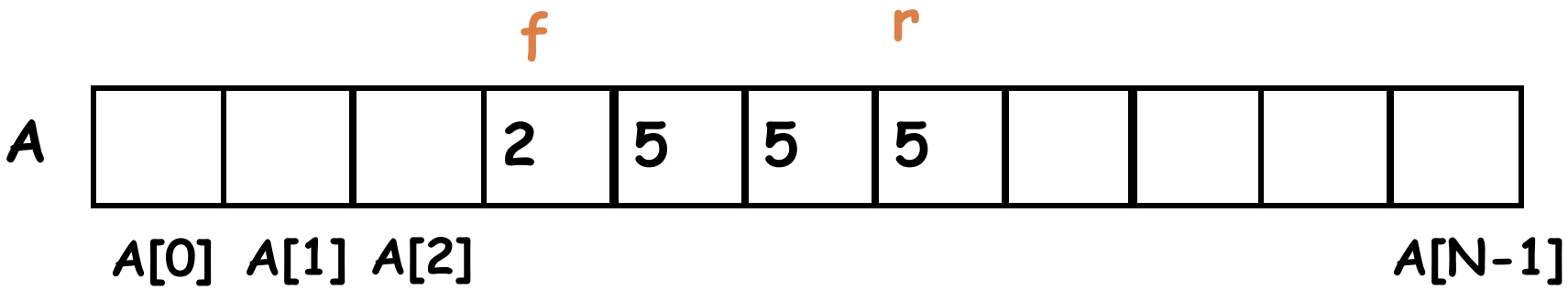
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)
EnQueue(5,Q)
EnQueue(5,Q)
DeQueue(Q)
DeQueue(Q)

Cách dùng mảng 2

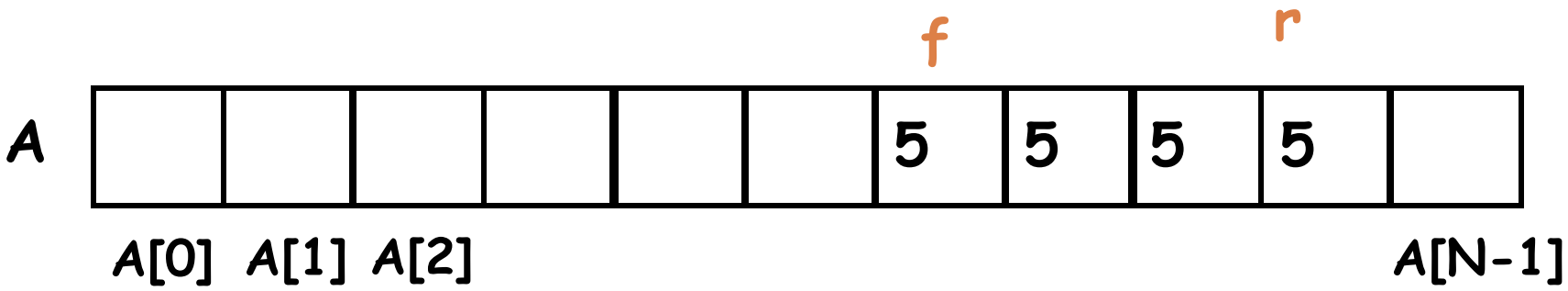
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)
EnQueue(5,Q)
EnQueue(5,Q)
DeQueue(Q)
DeQueue(Q)
DeQueue(Q), EnQueue(5,Q), DeQueue(Q), EnQueue(5,Q),…….

Cách dùng mảng 2

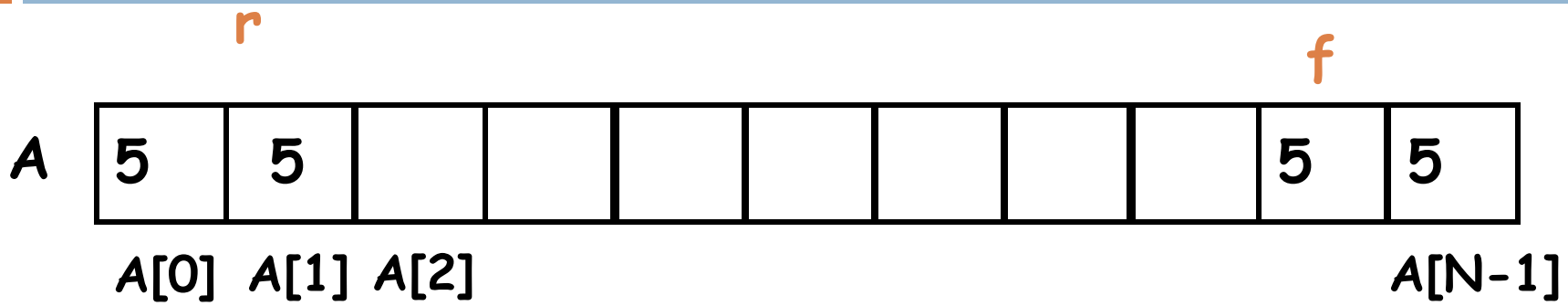
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)
EnQueue(5,Q)
EnQueue(5,Q)
DeQueue(Q)
DeQueue(Q)
DeQueue(Q), EnQueue(5,Q), DeQueue(Q), EnQueue(5,Q),…….

Cách dùng mảng 2

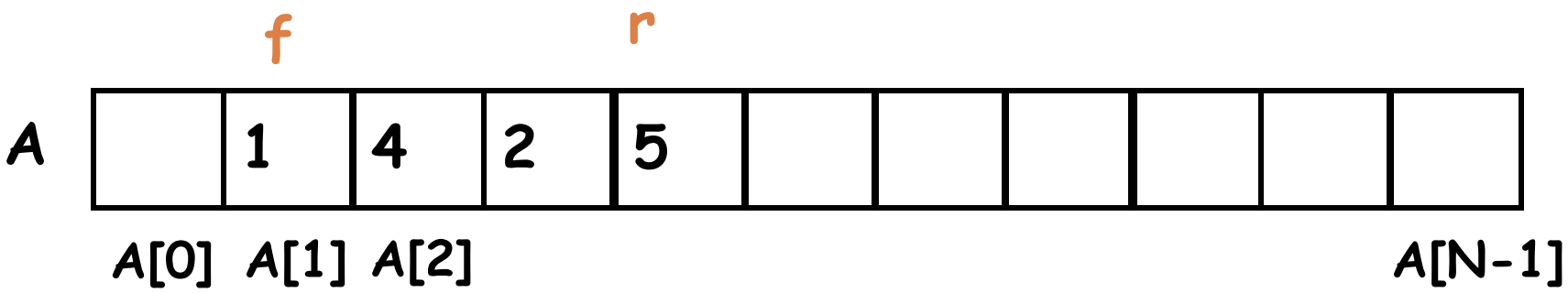
Dùng mảng vòng



DeQueue(Q), EnQueue(5,Q), DeQueue(Q), EnQueue(5,Q),…….

Cách dùng mảng 2

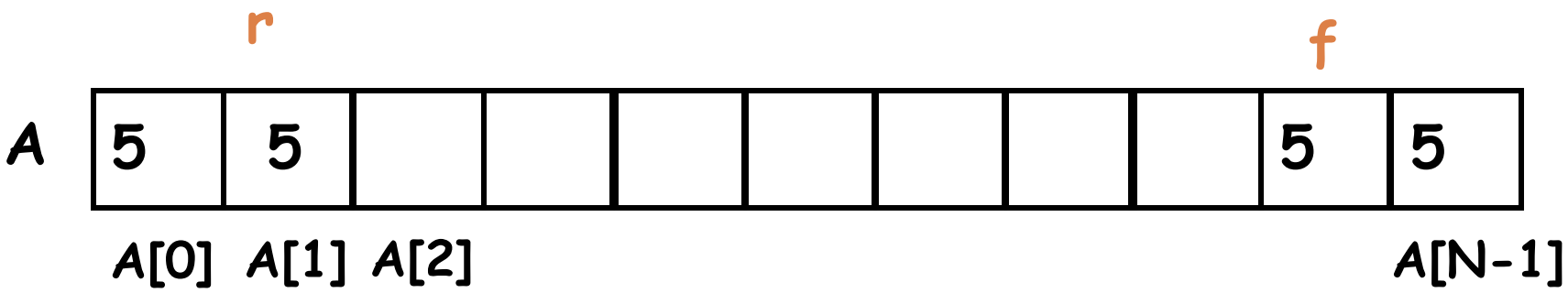
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)




```
size(Q):  if (r >= f) then return (r-f)
          else return N-(f-r)
```

Cách dùng mảng 2

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



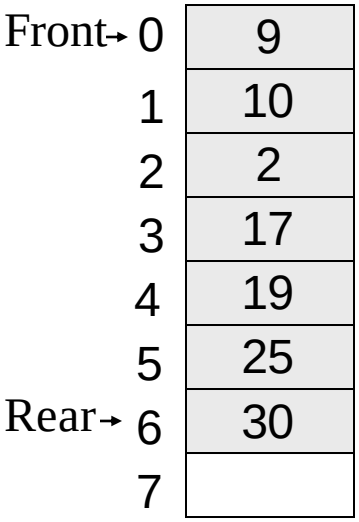

```
size(Q):  if (r >= f) then return (r-f)
          else return N-(f-r)
```

Cách dùng mảng 2

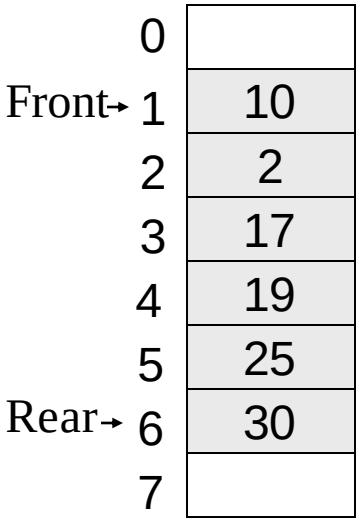
CÀI ĐẶT HÀNG ĐỢI

a. Cài đặt hàng đợi bằng mảng:

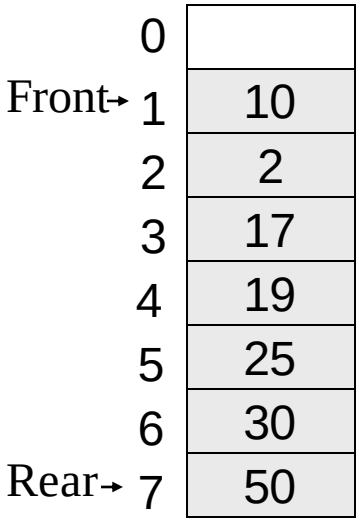
- Giả sử một mảng a[8] chứa các phần tử của hàng đợi.



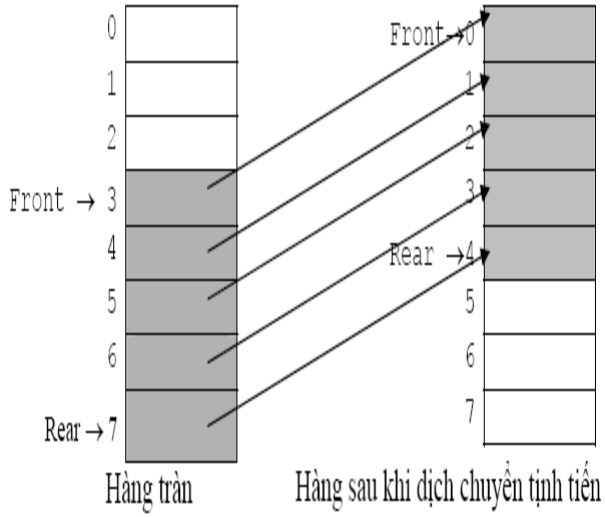
Các phần tử của hàng được đưa vào mảng a[8]



Xóa một phần tử Front tăng lên 1



Thêm một phần tử Rear tăng lên 1



- Nếu thêm tiếp một phần tử thì hàng bị tràn.

- Nên phải tịnh tiến trước khi thêm vào hàng

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Nhận xét:
  - Khi `front = rear` thì queue có thể đầy hoặc rỗng
  - Không thể phân biệt được queue đầy hoặc rỗng trong trường hợp này

Operation	Return Value	first ← Q ← last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue( )	5	[3]
Q.is_empty( )	False	[3]
Q.dequeue( )	3	[ ]
Q.is_empty( )	True	[ ]
Q.dequeue( )	"error"	[ ]
Q.enqueue(7)	–	[7]
Q.enqueue(9)	–	[7, 9]
Q.first( )	7	[7, 9]
Q.enqueue(4)	–	[7, 9, 4]
len(Q)	3	[7, 9, 4]
Q.dequeue( )	7	[9, 4]

CÁC PHÉP TOÁN TRÊN HÀNG ĐỢI BẰNG PYTHON

- **Q.enqueue(e):** Thêm phần tử e vào cuối hàng đợi Q.
- **Q.dequeue():** Loại bỏ và trả về phần tử đầu tiên từ hàng đợi Q; lỗi xảy ra nếu hàng đợi trống.
- **Q.first():** Trả về một tham chiếu đến phần tử ở đầu hàng đợi Q mà không xóa nó; lỗi xảy ra nếu hàng đợi trống.
- **Q.is_empty():** Trả về True nếu hàng đợi Q không chứa phần tử nào.
- **len(Q):** Trả về số phần tử có trong hàng đợi Q;

CÁC PHÉP TOÁN TRÊN HÀNG ĐỢI BẰNG PYTHON

- **_data:** là một tham chiếu đến một thể hiện danh sách có giá trị cố định.
- **_size:** là số nguyên biểu thị số hiện tại của các phần tử được lưu trữ trong hàng đợi (ngược lại với độ dài của danh sách dữ liệu).
- **_front:** là số nguyên biểu thị chỉ số trong dữ liệu của phần tử đầu tiên của hàng đợi (giả sử hàng đợi không trống).

```python
class ArrayQueue:
    '''
    FIFO Queu implementation using a python list as underlying storage
    '''
    DEFAULT_CAPACITY = 5 #moderate capacity for all new queues
    def __init__(self):
        '''Create an empty queue'''
        self._data =[None] * ArrayQueue.DEFAULT_CAPACITY
        self._size =0
        self._front =0
    #def
    def __len__(self):
        #return the number of elements in the queue
        return self._size
    #def
    def is_empty(self):
        #Return True if the queue is empty
        return self._size == 0
    #def
    def first(self):
        '''
        Return (but do not remove) the element at the front of the queue
        Raise Empty Exception if the queue is empty
        '''
        if self.is_empty():
            raise Empty ('Queue is Empty')
        return self._data[self._front]
    #def
```

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

```
29      def dequeue(self):
30          '''
31          Remove and return the first element of the queue
32          raise Empty exception if the queue is empty
33          '''
34          if self.is_empty():
35              raise Empty('Queue is Empty')
36          answer = self._data[self._front]
37          self._data[self._front] = None # help garbage collection
38          self._front = (self._front+1)%len(self._data) #circular indexing
39          self._size -=1 #reduce the queue size
40          return answer
41      #def
42      def enqueue(self,e):
43          '''Add an element to the back of queue '''
44          if self._size == len(self._data):
45              self._resize(2*len(self._data)) #double the array size
46          avail =(self._front +self._size) % len(self._data)
47          self._data[avail] =e
48          self._size +=1
49      #def
```

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

```
50      def _resize(self,cap):
51          #resize to a new list of capacity >=len(self)
52          old =self._data
53          self._data = [None]*cap
54          walk =self._front
55          for k in range(self._size):#only consider existing element
56              self._data[k] = old[walk] #intentionally shift indices
57              walk = (1+walk) % len(old) #use old size as modulus
58          self._front = 0 #front has been realigned
59      #def
60      def __str__(self):
61          #string representation of the queue
62          return '<'+''.join(str(self._data))+'<'
63  #class
```

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

```
64  ##########################
65  if __name__ =='__main__':
66      Q =ArrayQueue()
67      Q.enqueue(5)
68      Q.enqueue(7)
69      Q.enqueue(9)
70      Q.enqueue(2)
71      Q.enqueue(6)
72      Q.enqueue(4)
73      Q.enqueue(1)
74      Q.enqueue(0)
75      print('============Demo============')
76      print('Q: ',Q)
77      print('Queue Lenght:', len(Q))
78      print('Remove last item: ',Q.dequeue())
79      print('Remove last item: ',Q.dequeue())
80      print('Q: ',Q)
81      print('Queue Lenght:', len(Q))
```

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

SHRINKING THE UNDERLYING ARRAY

```
29      def dequeue(self):
30          '''
31          Remove and return the first element of the queue
32          raise Empty exception if the queue is empty
33          '''
34          if self.is_empty():
35              raise Empty('Queue is Empty')
36          answer = self._data[self._front]
37          self._data[self._front] = None # help garbage collection
38          self._front = (self._front+1)%len(self._data) #circular indexing
39          self._size -=1 #reduce the queue size
40          if 0<self._size <len(self._data)//4: #shrink the array by half
41              self._resize(len(self._data)//2) #when queue size 1/4
42          return answer #total array capacity
43      #def
```

Hiện
(Impleme

```
67  if __name__ =='__main__':
68      Q =ArrayQueue()
69      Q.enqueue(5)
70      Q.enqueue(7)
71      Q.enqueue(9)
72      Q.enqueue(2)
73      Q.enqueue(6)
74      Q.enqueue(4)
75      Q.enqueue(1)
76      Q.enqueue(0)
77      print('==========Demo==========')
78      print('Q: ',Q)
79      print('Queue Lenght:', len(Q))
80      print('Remove last item: ',Q.dequeue())
81      print('Remove last item: ',Q.dequeue())
82      print('Q: ',Q)
83      print('Queue Lenght:', len(Q))
84      print('Remove last item: ',Q.dequeue())
85      print('Remove last item: ',Q.dequeue())
86      print('Remove last item: ',Q.dequeue())
87      print('Remove last item: ',Q.dequeue())
88      print('Q: ',Q)
89      print('Queue Lenght:', len(Q))
90      print('Remove last item: ',Q.dequeue())
91      print('Q: ',Q)
```

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

Độ phức tạp của thuật toán

Operation	Running Time
Q.enqueue(e)	$O(1)^*$
Q.dequeue( )	$O(1)^*$
Q.first( )	$O(1)$
Q.is_empty( )	$O(1)$
len(Q)	$O(1)$

*amortized

Hiện thực Queue dùng danh sách liên kết

(Implementation of a Queue using Array)

Tạo tập tin DSLKQueue.PY:

- Nhập vào từ modun DSLK lớp DSLienKet
- Định nghĩa lớp HangDoi gồm các phương thức:
  1. **__init__(self):** khởi tạo
  2. **__str(self)__:** Đổi sang kiểu chuỗi
  3. **is_empty(self):** Kiểm tra hàng đợi (danh sách) rỗng
  4. **xep_hang(self, gia_tri):** Thêm 1 phần tử vào hàng
  5. **ra_hang(self):** Lấy 1 phần tử ra khỏi hàng

Viết đoạn mã thực thi tạo 1 hàng đợi. Lần lượt thêm các giá trị từ 1 đến 5 vào hàng đợi qua phương thức **xep_hang**. Lần lượt lấy và xuất ra màn hình các giá trị từ hàng đợi qua phương thức **ra_hang**.

Hiện thực Queue dùng danh sách

Dùng LIST – Không hiệu quả

(Implementation of a Queue using Array)

```
class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, item):
        """Thêm phần tử vào cuối hàng đợi"""
        self.queue.append(item)

    def dequeue(self):
        """Lấy phần tử đầu hàng đợi"""
        if not self.is_empty():
            return self.queue.pop(0)  # O(n)
        return None
```

```
    def is_empty(self):
        """Kiểm tra hàng đợi có rỗng không"""
        return len(self.queue) == 0

    def size(self):
        """Trả về số phần tử trong hàng đợi"""
        return len(self.queue)

# Test hàng đợi
q = Queue()
q.enqueue(1)
q.enqueue(2)
q.enqueue(3)
print(q.dequeue())  # Output: 1
print(q.queue)      # Output: [2, 3]
```

.pop(0) không hiệu quả vì nó có độ phức tạp **O(n)**.

Hiện thực Queue dùng danh sách
Dùng collections.deque (Hiệu quả)

(Implementation of a Queue using Array)

```python
from collections import deque

class Queue:
    def __init__(self):
        self.queue = deque()

    def enqueue(self, item):
        self.queue.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.queue.popleft()  # O(1)
        return None

    def is_empty(self):
        return len(self.queue) == 0

    def size(self):
        return len(self.queue)

# Test hàng đợi
q = Queue()
q.enqueue(1)
q.enqueue(2)
q.enqueue(3)
print(q.dequeue())  # Output: 1
print(q.queue)      # Output: deque([2, 3])
```

deque từ thư viện collections có độ phức tạp **O(1)** cho cả enqueue() và dequeue()

Hiện thực Queue dùng danh sách

(Implementation of a Queue using Array)

Dùng queue.Queue (Dành cho lập trình đa luồng)

Thư viện queue có thể dùng trong **đa luồng (multithreading)** vì có cơ chế **lock**

```
from queue import Queue


q = Queue()
q.put(1)
q.put(2)
q.put(3)


print(q.get())  # Output: 1
print(q.qsize())  # Output: 2
```

CÁC PHÉP TOÁN TRÊN HÀNG ĐỢI BẰNG PYTHON

- So sánh các cách

Cách cài đặt	enqueue (Thêm)	dequeue (Xóa)	Ưu điểm	Nhược điểm
List	O(1)	O(n)	Đơn giản	Không hiệu quả
deque	O(1)	O(1)	Hiệu quả, nhanh	Không hỗ trợ đa luồng
queue.Queue	O(1)	O(1)	Hỗ trợ đa luồng	Phải import thư viện

CÁC PHÉP TOÁN TRÊN HÀNG ĐỢI BẰNG PYTHON
DOUBLE – ENDED QUEUES

- A deque (pronounced as “deck”) or a double-ended queue is a queue-like data structure that supports insertion and deletion at both the front and the back of the queue.
- The deque abstract data type is more general than both the stack and the queue ADTs.
- Example: a restaurant using a queue to maintain a waitlist
  - Occasionally, the first person might be removed from the queue only to find that atable was not available; typically, the restaurant will re-insert the person at the firstposition in the queue.
  - It may also be that a customer at the end of the queue may grow impatient and leave the restaurant.

CÁC PHÉP TOÁN TRÊN HÀNG ĐỢI BẰNG PYTHON
DOUBLE – ENDED QUEUES

- **D.add_first(e):** Thêm phần tử e vào trước deque D.
- **D.add_last(e):** Thêm phần tử e vào sau deque D.
- **D.delete_first():** Loại bỏ và trả về phần tử đầu tiên của deque D; lỗi xảy ra nếu deque trống.
- **D.delete_last():** Loại bỏ và trả về phần tử cuối cùng của deque D; lỗi xảy ra nếu deque trống.
- **D.first():** Trả về (nhưng không xóa) phần tử đầu tiên của deque D; lỗi xảy ra nếu deque trống.
- **D.last():** Trả về (nhưng không loại bỏ) phần tử cuối cùng của deque D; lỗi xảy ra nếu deque trống.
- **D.is_empty():** Trả về True nếu deque D không chứa phần tử nào.
- **len(D):** Trả về số phần tử trong deque D;

CÁC PHÉP TOÁN TRÊN NGĂN XẾP BẰNG PYTHON

DOUBLE – ENDED QUEUES

Operation	Return Value	Deque
D.add_last(5)	–	[5]
D.add_first(3)	–	[3, 5]
D.add_first(7)	–	[7, 3, 5]
D.first( )	7	[7, 3, 5]
D.delete_last( )	5	[7, 3]
len(D)	2	[7, 3]
D.delete_last( )	3	[7]
D.delete_last( )	7	[ ]
D.add_first(6)	–	[6]
D.last( )	6	[6]
D.add_first(8)	–	[8, 6]
D.is_empty( )	False	[8, 6]
D.last( )	6	[8, 6]

CÁC PHÉP TOÁN TRÊN NGĂN XẾP BẰNG PYTHON
DOUBLE – ENDED QUEUES

Our Deque ADT	collections.deque	Description
len(D)	len(D)	number of elements
D.add_first( )	D.appendleft( )	add to beginning
D.add_last( )	D.append( )	add to end
D.delete_first( )	D.popleft( )	remove from beginning
D.delete_last( )	D.pop( )	remove from end
D.first( )	D[0]	access first element
D.last( )	D[–1]	access last element
	D[j]	access arbitrary entry by index
	D[j] = val	modify arbitrary entry by index
	D.clear( )	clear all contents
	D.rotate(k)	circularly shift rightward k steps
	D.remove(e)	remove first matching element
	D.count(e)	count number of matches for e

```
1  import collections
2  D = collections.deque()
3  D.appendleft(5)
4  D.appendleft(6)
5  D.appendleft(10)
6  D.appendleft(2)
7  D.appendleft(3)
8  D.appendleft(7)
9  print('Deque D: ',D)
10 print('Length: ',len(D))
11 D.rotate(5) # circularly shift rightward k step
12 print('Deque D: ',D)
13 D.popleft()
14 D.pop()
15 print('Deque D: ',D)
16 print('Length: ',len(D))
```

Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

Nhận xét:

- Các thao tác trên Queue biểu diễn bằng danh sách liên kết làm việc với chi phí O(1)
- Nếu không quản lý phần tử cuối xâu, thao tác **RemoveQueue** sẽ có độ phức tạp O(n)

Queue - Ứng dụng

- Queue có thể được sử dụng trong một số bài toán:
  - Bài toán "sản xuất và tiêu thụ" (ứng dụng trong các hệ điều hành song song)
  - Bộ đệm (ví dụ: Nhấn phím $\Rightarrow$ Bộ đệm $\Rightarrow$ CPU xử lý)
  - Xử lý các lệnh trong máy tính (ứng dụng trong HĐH, trình biên dịch), hàng đợi các tiến trình chờ được xử lý, ….

CÁC PHÉP TOÁN TRÊN NGĂN XẾP BẰNG C

- **Các phép toán cơ bản trên hàng**
  - **CREATE_QUEUE(Q)** khởi tạo một hàng rỗng.
  - **FRONT(Q)** hàm trả về phần tử đầu tiên của hàng Q.
  - **ENQUEUE(x,Q)** thêm phần tử x vào cuối hàng Q.
  - **DEQUEUE(Q)** xoá phần tử tại đầu của hàng Q.
  - **EMPTY_QUEUE(Q)** hàm kiểm tra hàng rỗng.
  - **FULL_QUEUE(Q)** kiểm tra hàng đầy.

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Để khai báo một Queue, ta cần khai báo:
  - 1 mảng một chiều list,
  - 2 số nguyên front, rear cho biết chỉ số của đầu và cuối của hàng đợi,
  - hằng số N cho biết kích thước tối đa của Queue
- Hàng đợi có thể được khai báo cụ thể như sau:

```
struct Queue
{
    DataType  list[N];
    int   front, rear;
};
```

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Các hàm cần cài đặt:
- `Init(Queue &q)`
- `isEmpty(Queue q)`
- `EnQueue(Queue &q, DataType x)`
- `DeQueue(Queue &q)`
- `Front(Queue q)`
- Do khi cài đặt bằng mảng một chiều, Queue bị giới hạn kích thước nên cần xây dựng thêm một thao tác phụ:
  - **isFull**(): Kiểm tra xem Queue có đầy chưa

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- **Khởi tạo Queue:**

```
void    Init(Queue &q)
{
      q.front = q.rear = 0;
}
```

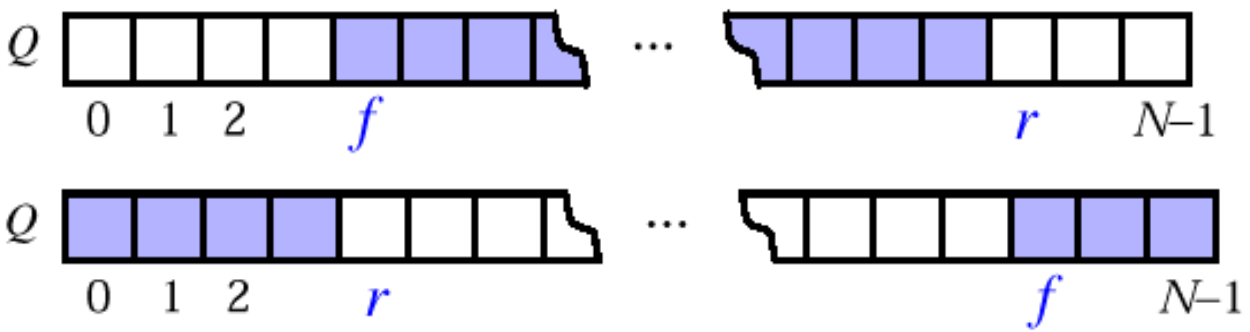
- **Kiểm tra xem Queue có rỗng không:**

```
int isEmpty(Queue q)
{
      if ( q.front==q.rear && q.rear==0 )
            return 1;
      if (q.front == q.rear) return 1;
      return 0;
}
```

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- **Kiểm tra xem Queue có đầy hay không:**

```
int isFull(Queue q)
{
   if (q.front == q.rear) return 1;
   return 0;
}
```



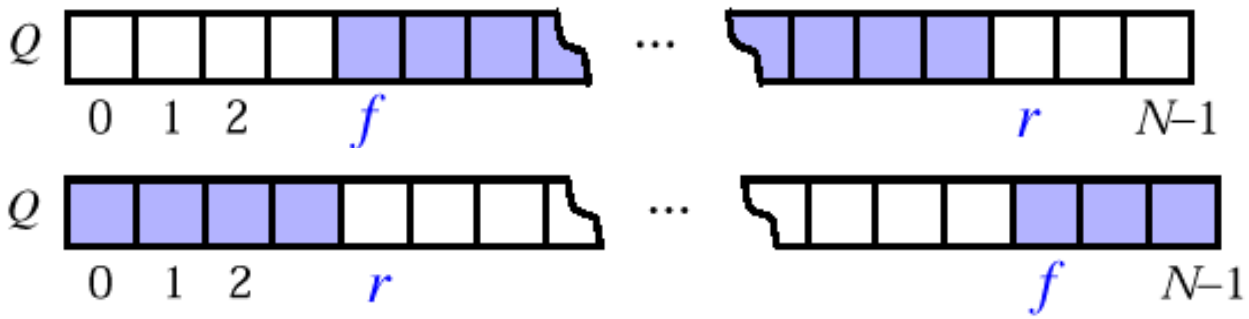
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Giải quyết trường hợp điều kiện Queue đầy hoặc rỗng:
  1. Không để Queue đầy
     - Tăng kích thước mảng khi thêm mà không còn chỗ
  2. Định nghĩa thêm 1 biến để tính số phần tử hiện hành trong Queue (NumElements)
     - Mỗi khi thêm 1 pt vào Queue thì NumElements++
     - Mỗi khi lấy 1 pt khỏi Queue thì NumElements—
     - Queue rỗng khi (`front = rear` và `NumElements=0`)
     - Queue đầy khi (`front = rear` và `NumElements!=0`)

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Thêm một phần tử x vào cuối Queue:

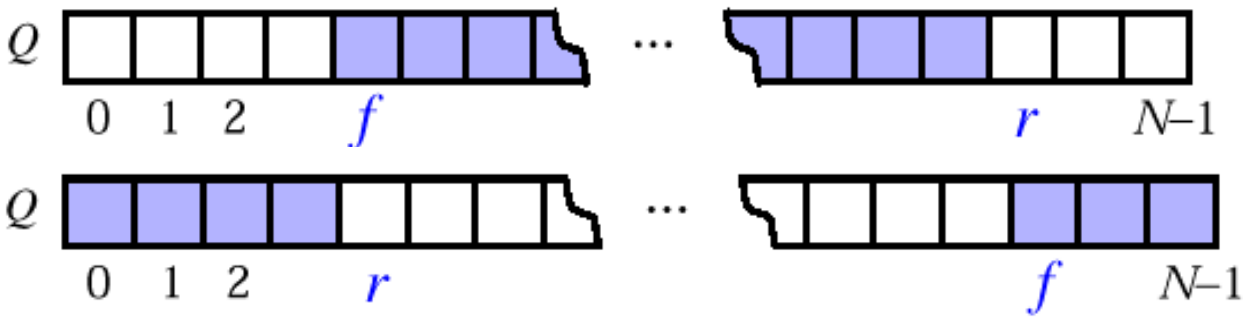
```
int  EnQueue(Queue &q, DataType x)
{
     if (isFull(q))
          return 0; // không thêm được vì Queue đầy
     q.list[q.rear] = x;
     q.rear++;
     if (q.rear==N)        q.rear=0;
     return 1;
}
```



Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- **Lấy phần tử ra khỏi Queue:**

```
DataType  DeQueue(Queue &q)
{
      if (isEmpty(q)){
            cout<<"Queue rong";
            return 0;}
      DataType t = q.list[q.front];
      q.front++;
      if (q.front==N) q.front = 0;
      return t;
}
```



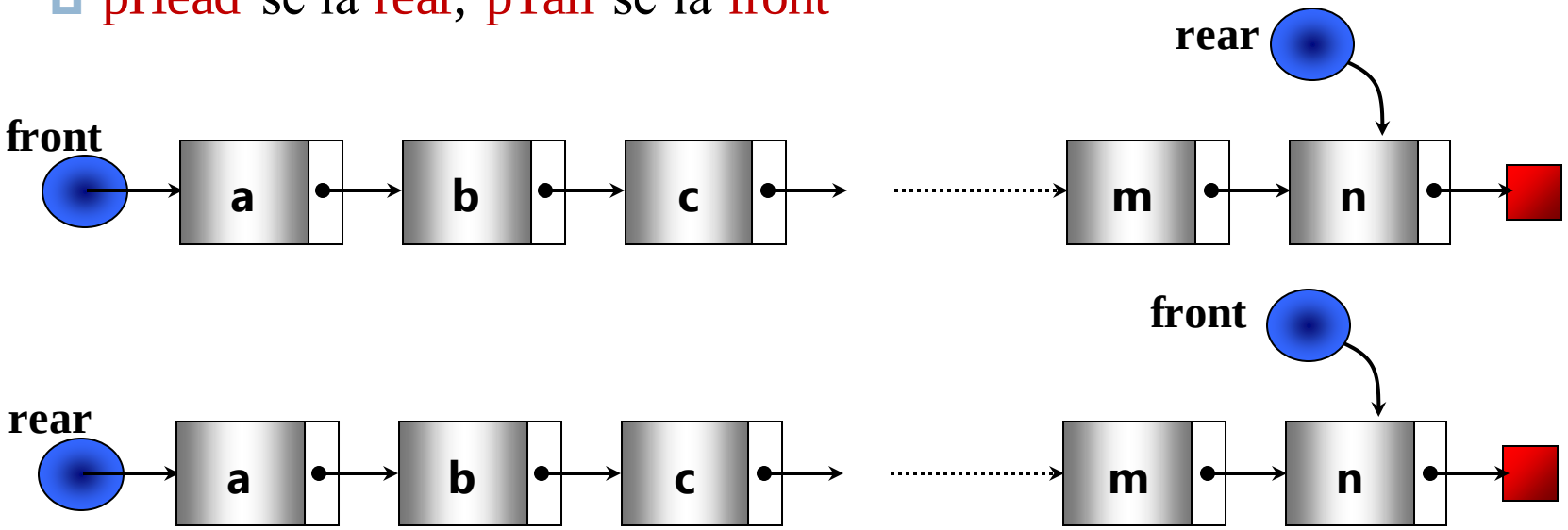
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Xem thông tin của phần tử ở đầu Queue:

```
DataType Front(Queue q)
{
   if (isEmpty(q))
   {
        cout<<"Queue rong";
        return 0;
   }
   return q.list[q.front];
}
```

Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

- Có thể biểu diễn Queue bằng cách sử dụng DSLK đơn
- Có 2 lựa chọn (cách nào tốt nhất?):
  - pHead sẽ là front, pTail sẽ là rear
  - pHead sẽ là rear, pTail sẽ là front



Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

- Khai báo các cấu trúc:

```
struct Node
{
    DataType data;
    Node *pNext;
};
struct Queue
{
    Node *front, *rear;
};
```

Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

- Khởi tạo Queue rỗng:

```
void Init(Queue &q)
{
    q.front = q.rear = NULL;
}
```

- Kiểm tra hàng đợi rỗng :

```
int  isEmpty(Queue &q)
{
    if ( q.front==NULL )
        return 1;
    else
        return 0;
}
```

Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

- **Thêm một phần tử p vào cuối Queue:**

```
int AddQueue(Queue &q, DataType x)
{
   Node *p = new Node;
   if (p==NULL) return 0; //Khong du bo nho
   p->pNext = NULL;
   p->data = x;
   if (q.front==NULL) // TH Queue rỗng
      q.front = q.rear = p;
   else
   {
      q.rear->pNext = p;
      q.rear = p;
   }
   return 1;
}
```

Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

- **Lấy phần tử ra khỏi Queue:**

```
DataType RemoveQueue(Queue &q)
{
    if (isEmpty(q)) {
        cout<<"Queue rong";return 0;
    }
    Node *p = q.front;
    DataType x = p->data;
    q.front = q.front->pNext;
    if ( q.front==NULL ) q.rear = NULL;
    delete p;
    return x;
}
```

Hiện thực Queue dùng mảng
(Implementation of a Queue using Linked List)

- Xem thông tin của phần tử ở đầu Queue:

```
DataType Front(Queue q)
{
   if (isEmpty(q))
   {
        cout<<"Queue rong";
        return 0;
   }
   return q.front->data;
}
```