

Chương 2_3:

QUEUE – HÀNG ĐỢI

Nội dung

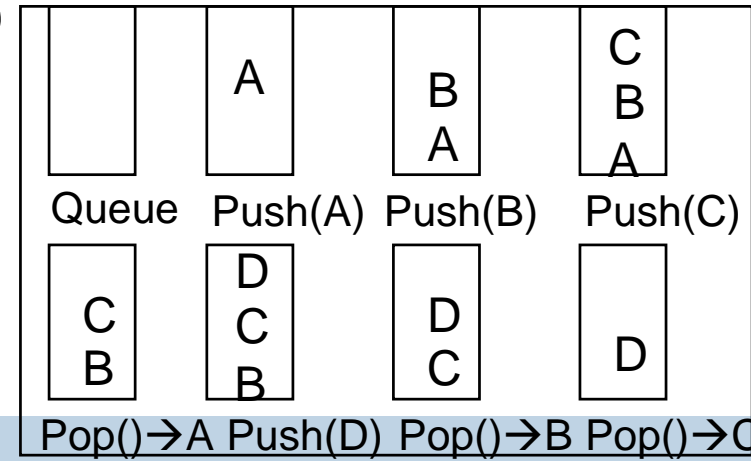
2

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

Queue - Khái niệm

3

- 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

4

Imaging

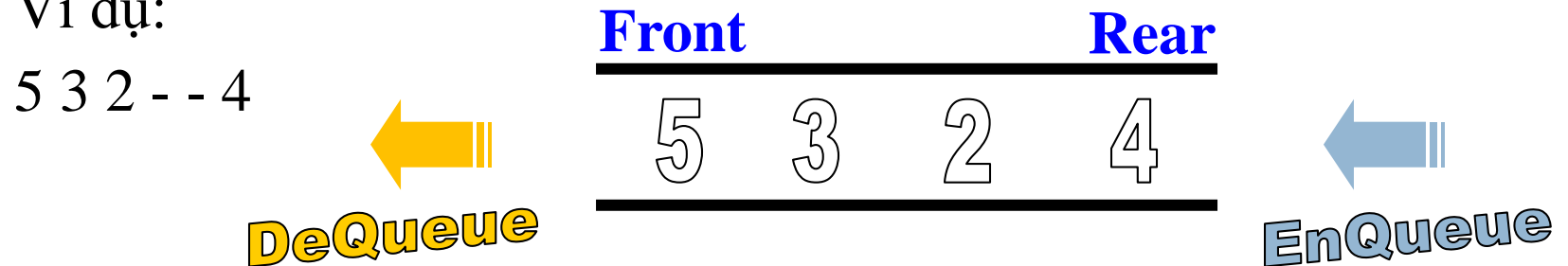


Queue – Các thao tác

5

- 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:



- 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

CÁC PHÉP TOÁN TRÊN NGĂN XẾP

- 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)** xóa phần tử tại đầu của hàng Q.
 - ▣ **EMPTY_QUEUE(Q)** hàm kiểm tra hàng rỗng.
 - ▣ **FULL_QUEUE(Q)** kiểm tra hàng đầy.

Queue – Hiện thực Queue

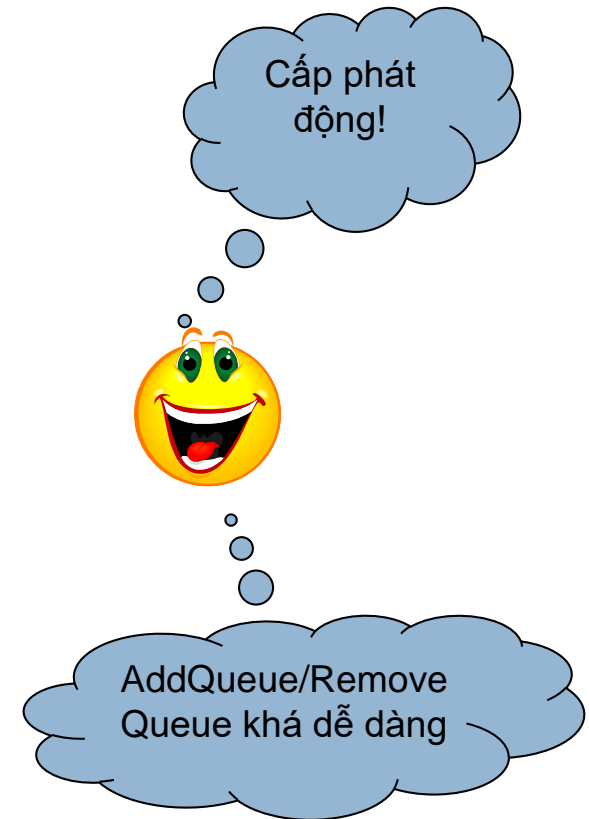
(Implementation of a Queue)

7

Mảng 1 chiều



Danh sách LK



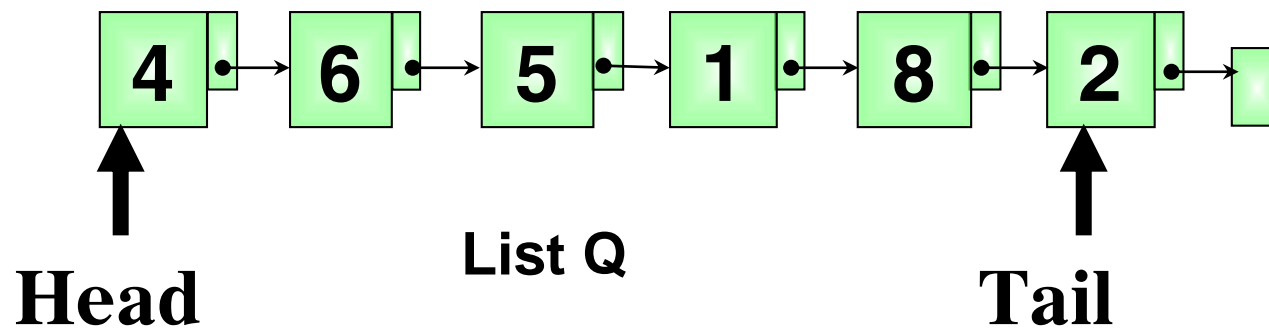
Cài đặt Queue

- 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)

9

□ Hai cách hiện thực:

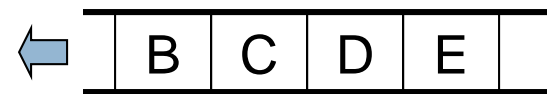
▣ **Khi lấy một phần tử ra thì đồng thời dời các ô phía sau nó lên một vị trí:**



Ban đầu



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



Thêm vào 1 phần tử

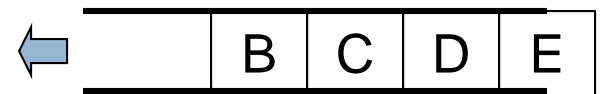
▣ **Khi lấy một phần tử ra thì không dời ô lên (xoay vòng):**



Ban đầu



Lấy ra 1 phần tử



Thêm vào 1 phần tử

Hiện thực Queue dùng mảng

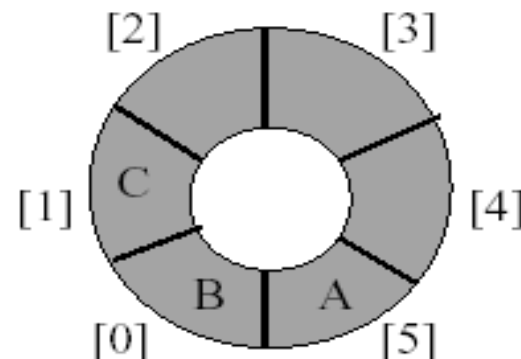
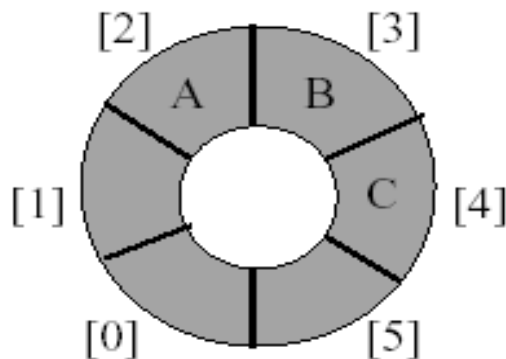
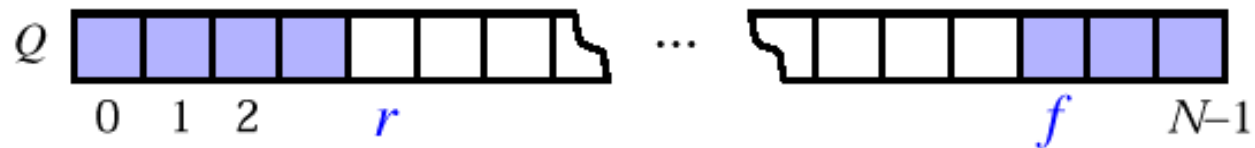
(Implementation of a Queue using Array)

10

- Trạng thái Queue lúc bình thường:



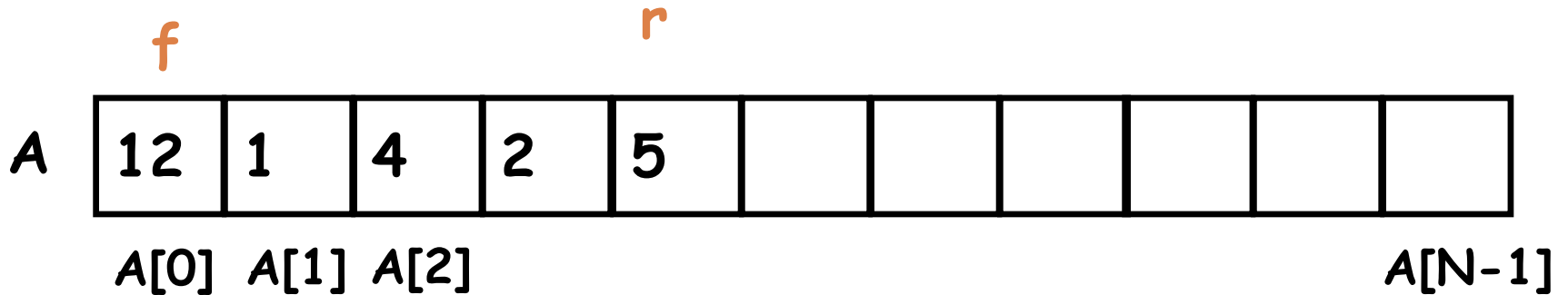
- Trạng thái Queue lúc xoay vòng:



Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

11



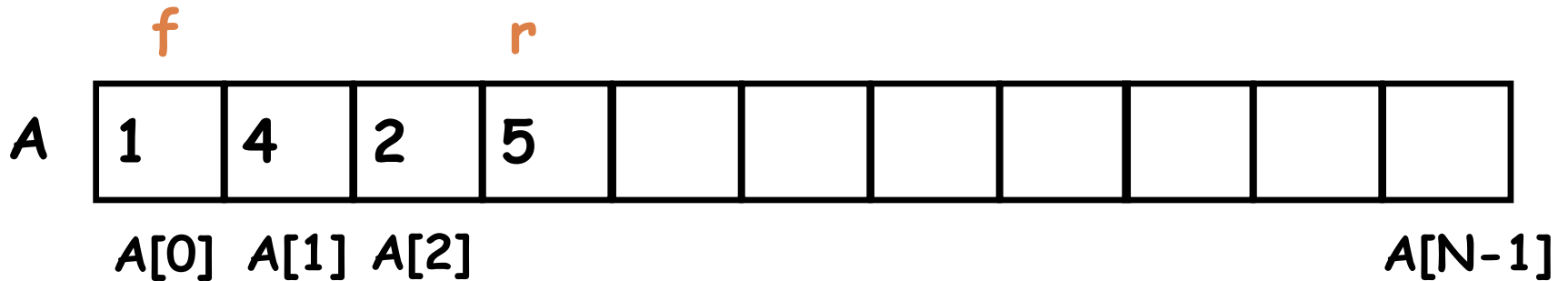
DeQueue(Q)

Cách dùng mảng 1

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

12



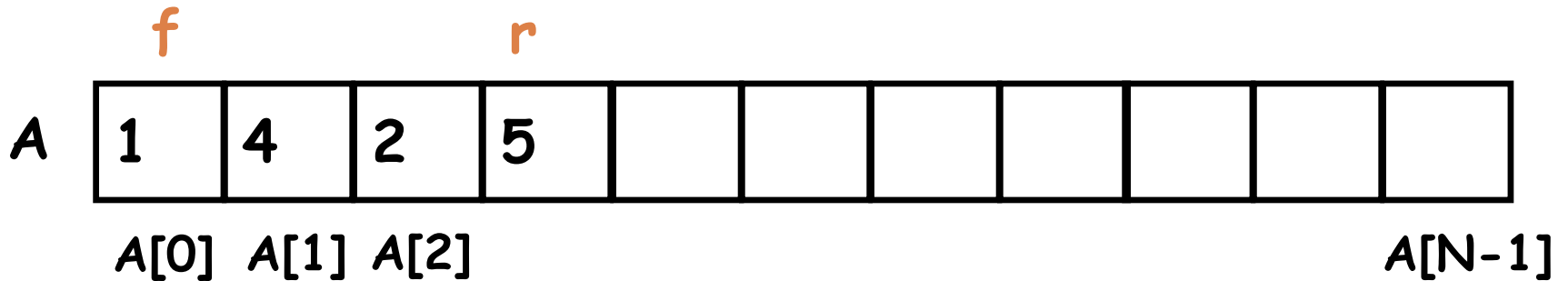
DeQueue(Q)

Cách dùng mảng 1

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

13

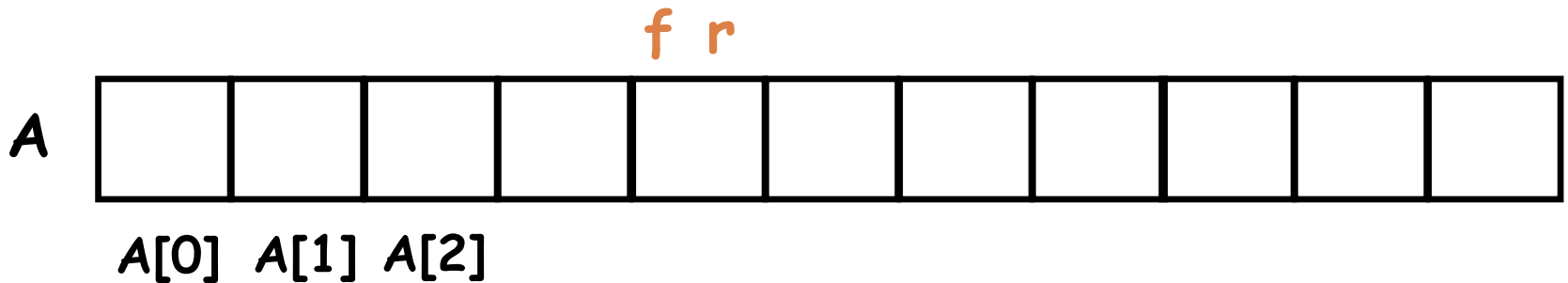
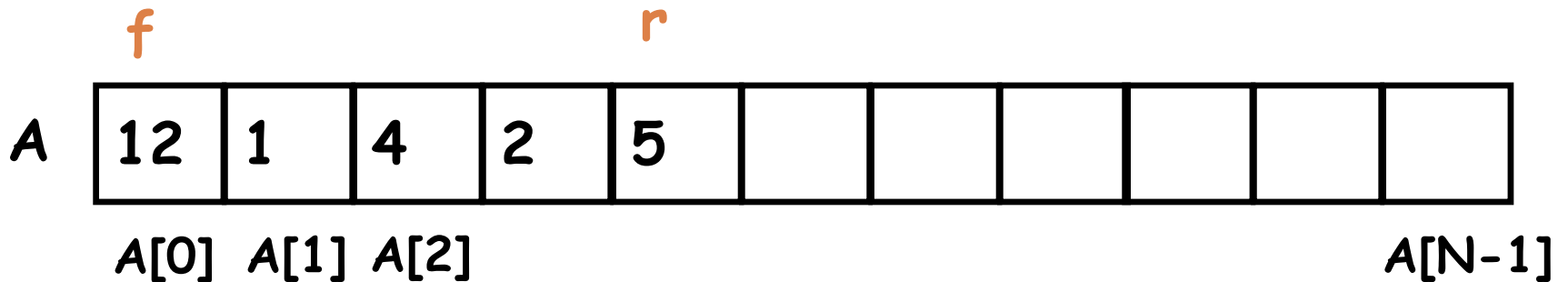


Cách dùng mảng 1

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

14



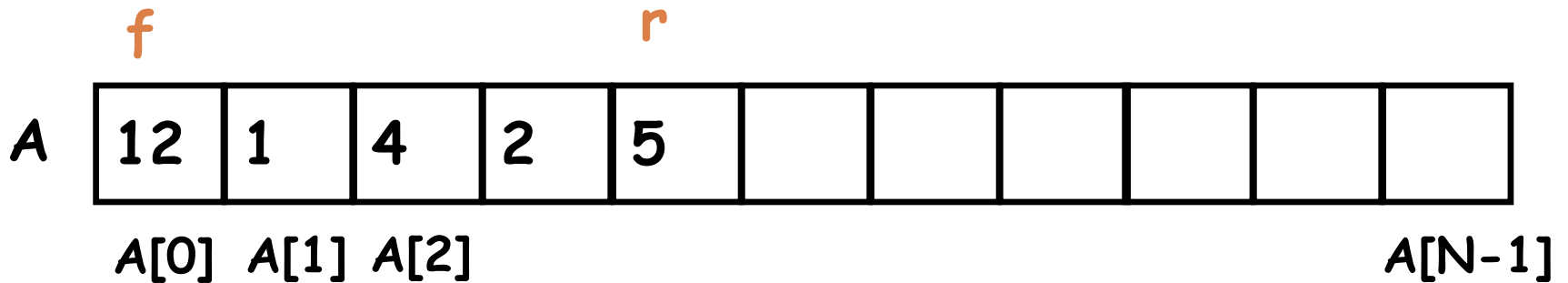
Cách dùng mảng 2

Empty queue $f=r$

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

15



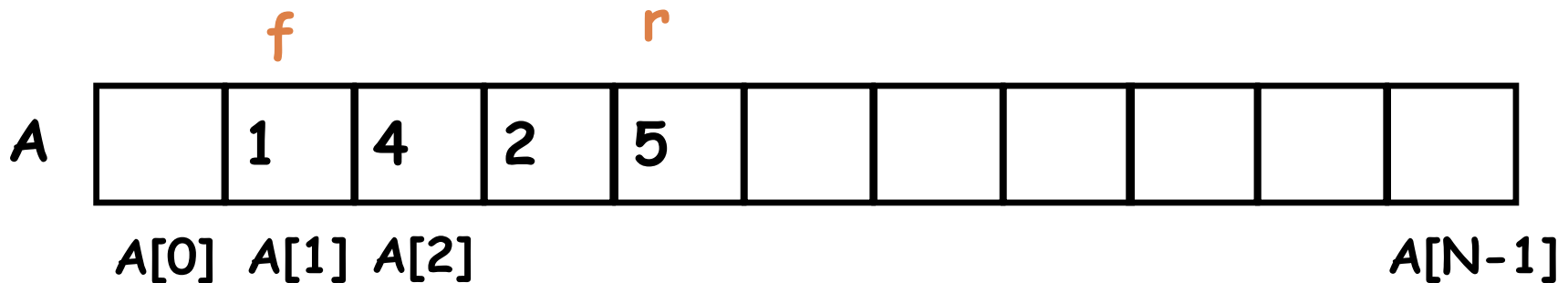
DeQueue(Q)

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

16



DeQueue(Q)

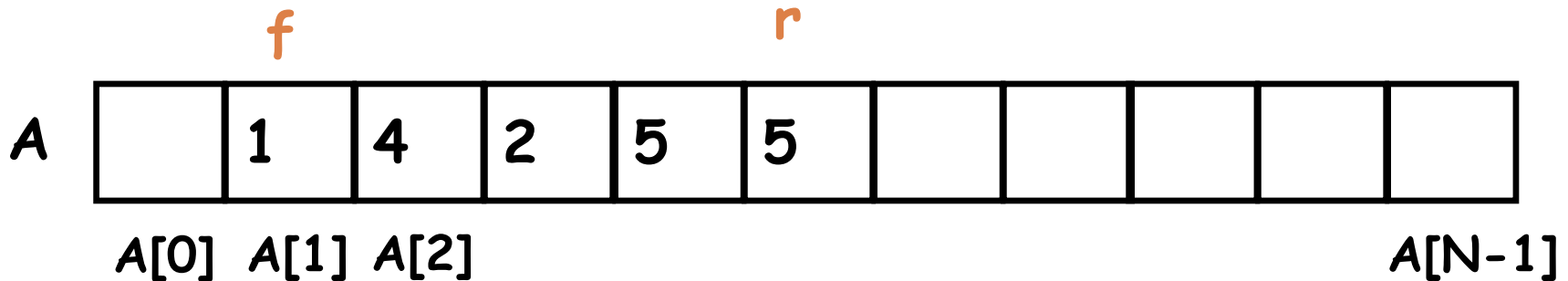
EnQueue(5, Q)

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

17



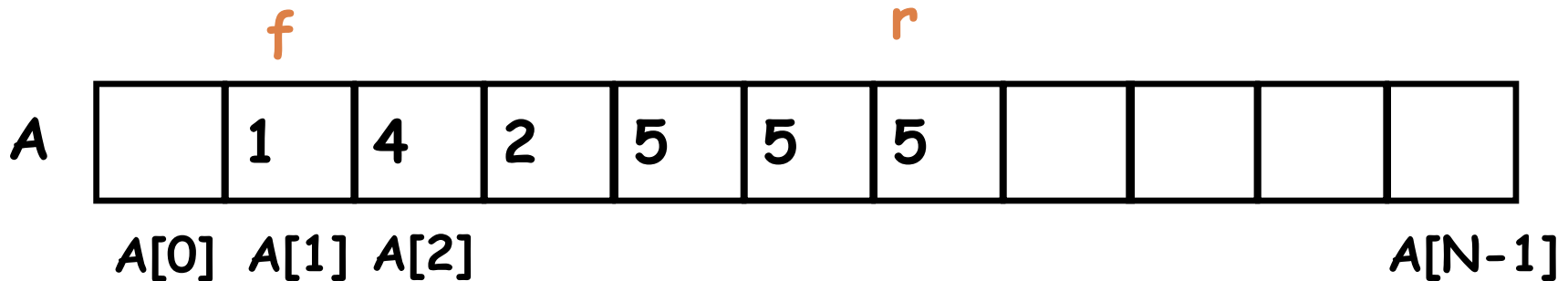
DeQueue(Q)
EnQueue(5, Q)
EnQueue(5, Q)

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

18



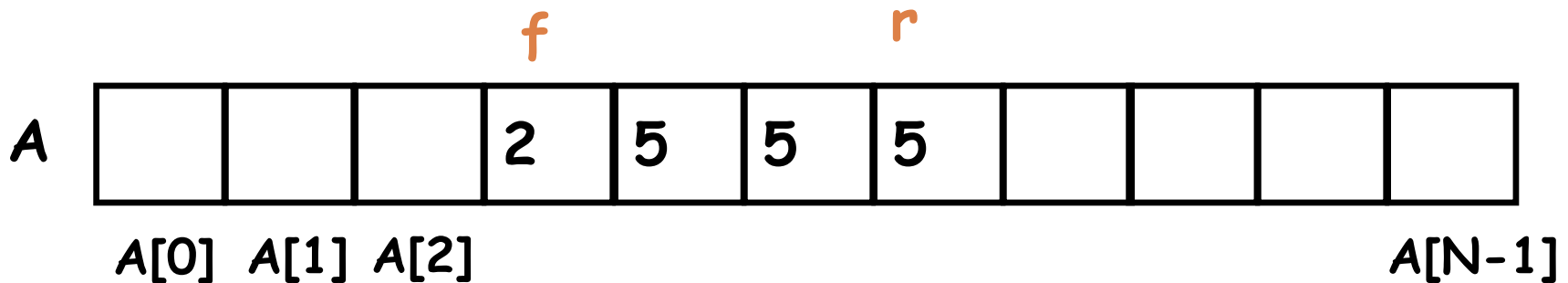
DeQueue(Q)
EnQueue(5, Q)
EnQueue(5, Q)
DeQueue(Q)
DeQueue(Q)

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

19



DeQueue(Q)

EnQueue(5, Q)

EnQueue(5, Q)

DeQueue(Q)

DeQueue(Q)

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

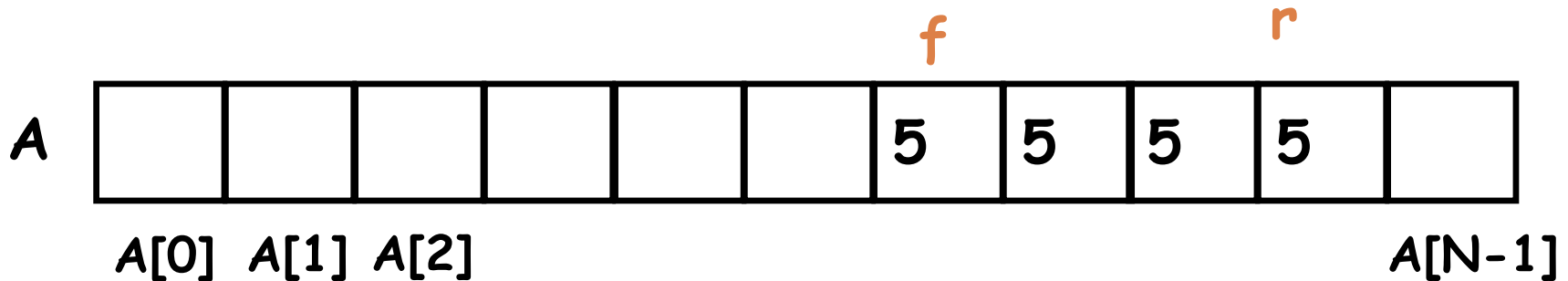
EnQueue(5, Q),

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

20



DeQueue(Q)

EnQueue(5, Q)

EnQueue(5, Q)

DeQueue(Q)

DeQueue(Q)

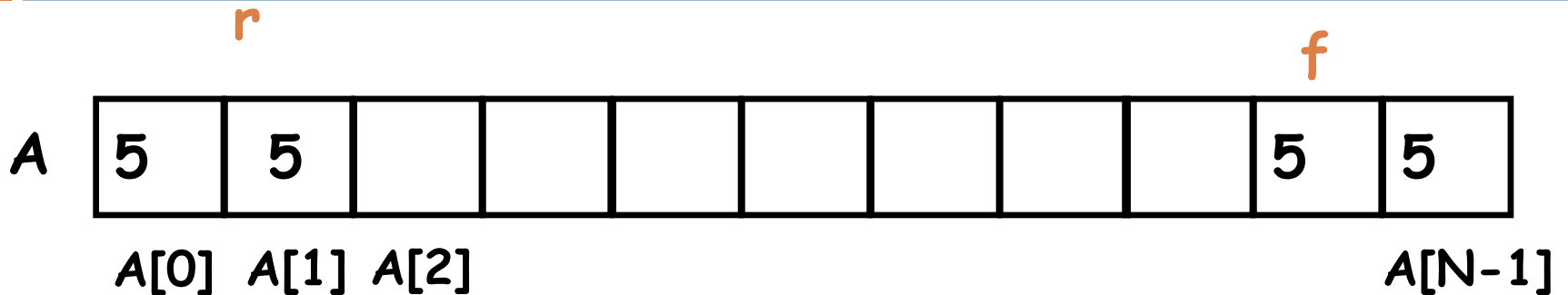
DeQueue(Q), EnQueue(5, Q), DeQueue(Q),

EnQueue(5, Q),

Cách dùng mảng 2

Dùng mảng vòng

21



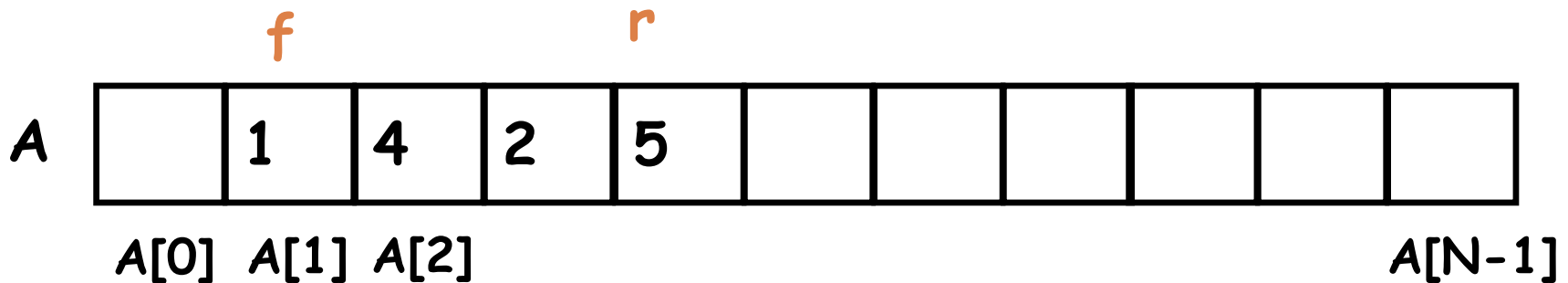
DeQueue(Q), EnQueue(5,Q), DeQueue(Q),
EnQueue(5,Q),.....

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

23



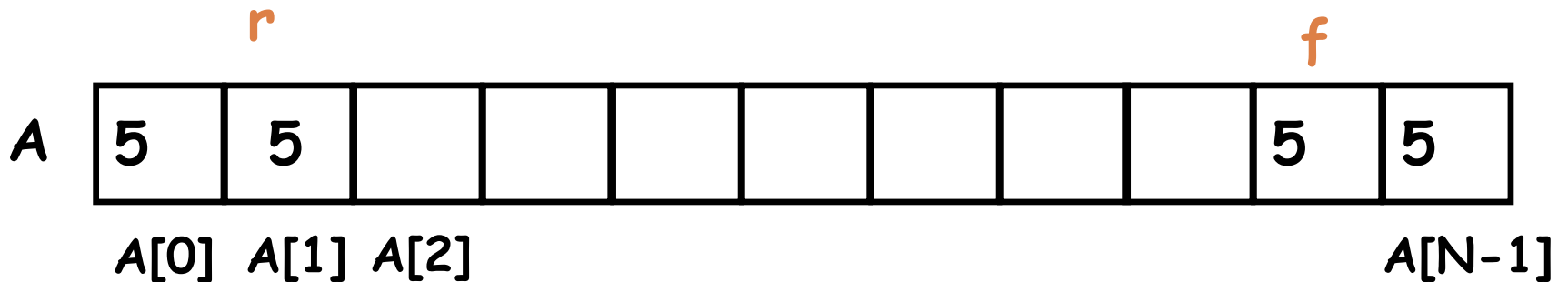
size(Q): if $(r \geq f)$ then return $(r - f)$
else return $N - (f - r)$

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

24



size(Q): if $(r \geq f)$ 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.

Front→ 0	9
1	10
2	2
3	17
4	19
5	25
Rear→ 6	30
7	

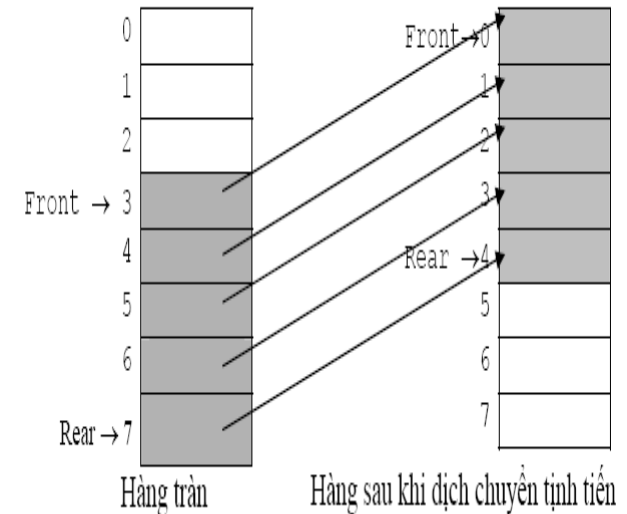
Các phần tử của hàng đợi được đưa vào mảng $a[8]$

Front→ 0	
1	10
2	2
3	17
4	19
5	25
Rear→ 6	30
7	

Xóa một phần tử
Front tăng lên 1

Front→ 0	
1	10
2	2
3	17
4	19
5	25
6	30
Rear→ 7	50

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)

26

- 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 \leftarrow Q \leftarrow 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 NGĂN XẾP 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 NGĂN XẾP 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).

```

1  class ArrayQueue:
2      ...
3      FIFO Queue implementation using a python list as underlying storage
4      ...
5      DEFAULT_CAPACITY = 5 #moderate capacity for all new queues
6      def __init__(self):
7          '''Create an empty queue'''
8          self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
9          self._size = 0
10         self._front = 0
11     #def
12     def __len__(self):
13         #return the number of elements in the queue
14         return self._size
15     #def
16     def is_empty(self):
17         #Return True if the queue is empty
18         return self._size == 0
19     #def
20     def first(self):
21         ...
22         Return (but do not remove) the element at the front of the queue
23         Raise Empty Exception if the queue is empty
24         ...
25         if self.is_empty():
26             raise Empty ('Queue is Empty')
27         return self._data[self._front]
28     #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

(Implement)

```
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 Stack và Queue dùng danh sách liên kết

(Implementation of a Queue using Array)

Định nghĩa lớp DSLienKet trong file DSLK.PY mô tả 1 danh sách liên kết gồm các phương thức:

- 1. `__init__(self)`:** khởi tạo
- 2. `__str__(self)`:** Đổi sang kiểu chuỗi
- 3. `them_dau(self, gia_tri)`:** Thêm phần tử vào đầu danh sách
- 4. `them_duoi(self, gia_tri)`:** Thêm phần tử vào cuối danh sách
- 5. `lay_dau(self)`:** Lấy giá trị phần tử đầu danh sách
- 6. `xoa_dau(self)`:** Xoá phần tử đầu danh sách

Hiện thực Stack và Queue dùng danh sách liên kết

(Implementation of a Queue using Array)

Tạo tập tin DSLKStack.PY:

- Nhập vào từ modul DSLK lớp DSLienKet
- Định nghĩa lớp NganXep gồm các phương thức:
 1. **__init__(self)**: khởi tạo
 2. **__str(self)__**: Đổi sang kiểu chuỗi
 3. **is_rong(self)**: Kiểm tra stack(danh sách) rỗng
 4. **Day_vao(self, gia_tri)**: Đẩy 1 phần tử vào đầu danh sách
 5. **lay_ra(self)**: Lấy 1 phần tử ra khỏi danh sách

Viết đoạn mã thực thi tạo 1 ngăn xếp. Lần lượt thêm các giá trị từ 1 đến 5 vào ngăn xếp qua phương thức **day_vao**. Lần lượt lấy và xuất ra màn hình các giá trị từ ngăn xếp qua phương thức **lay_ra**.

Hiện thực Stack và Queue dùng danh sách liên kết

(Implementation of a Queue using Array)

Tạo tập tin DSLKQueue.PY:

- Nhập vào từ modul 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**.

CÁC PHÉP TOÁN TRÊN NGĂN XẾP 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 a table was not available; typically, the restaurant will re-insert the person at the first position 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 NGĂN XẾP 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))
```

CÁC PHÉP TOÁN TRÊN NGĂN XẾP BẢNG 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)** xóa 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)

44

- 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)

45

- 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)

46

□ 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;
}
```

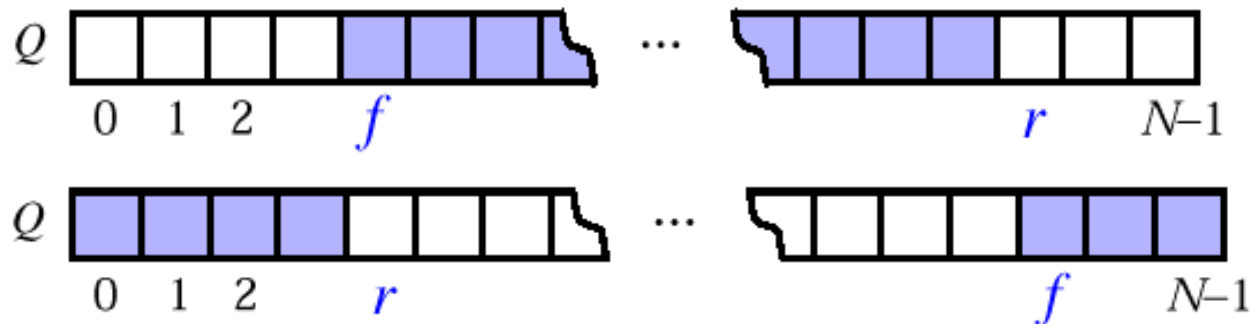
Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

47

- 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)

48

- 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`)

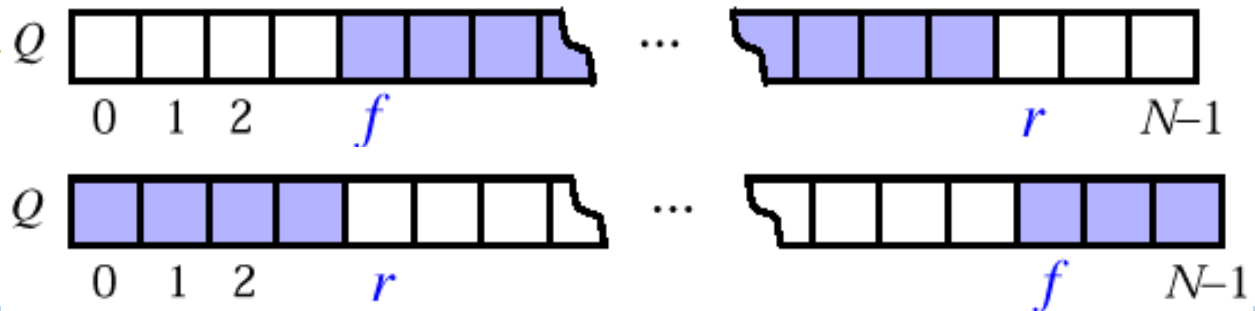
Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

49

- 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;
}
```



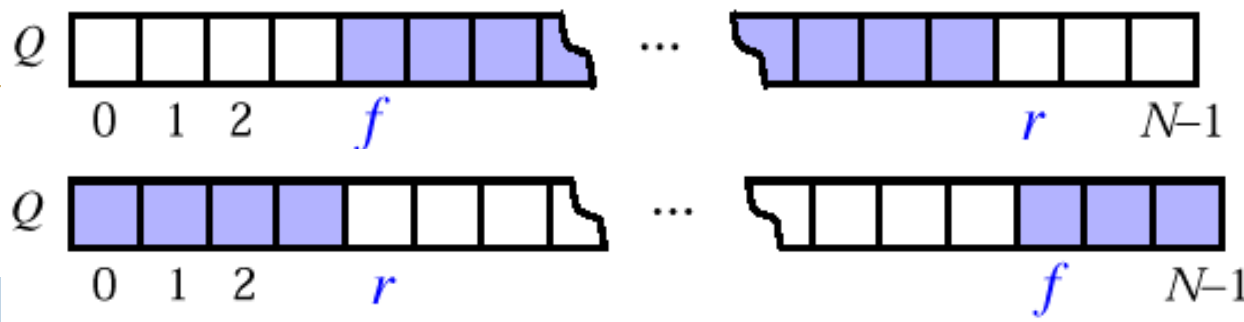
Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

50

□ 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:

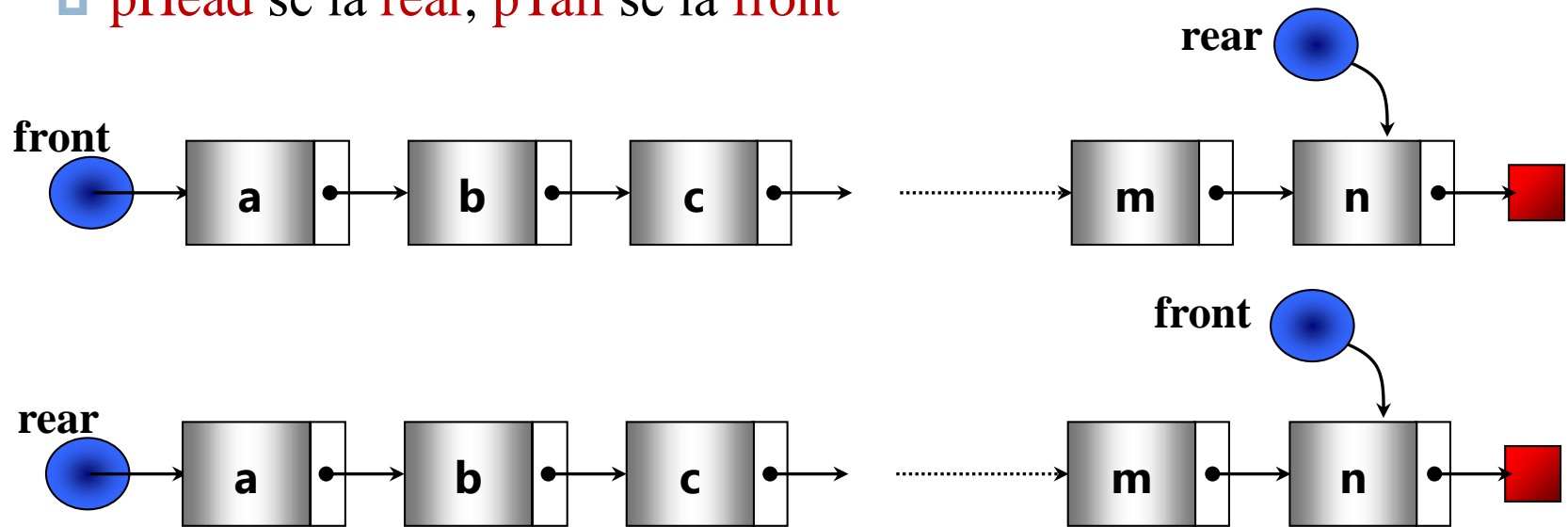
```
DataType Front (Queue q)
{
    if (isEmpty(q))
    {
        cout<<"Queue rong";
        return 0;
    }
    return q.list[q.front];
}
```

Hiện thực Queue dùng DSLK

(Implementation of a Queue using Linked List)

52

- 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)

53

- 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)

54

- 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)

55

- Thêm một phần tử p vào cuối Queue:

```
int AddQueue (Queue &q, DataType x)
{
    Node *p = new Node;
    if (p==NULL) return 0; //Không đủ bộ nhớ
    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)

56

□ 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;
}
```


Hiện thực Queue dùng DSLK

(Implementation of a Queue using Linked List)

58

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

59

- 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,