

Chương 4: NGĂN XẾP – HÀNG ĐỢI (Stack - Queue)

Nội dung

2

- Ngăn xếp (**Stack**)
 - Khái niệm Stack
 - Các thao tác trên Stack
 - Hiện thực Stack
 - Ứng dụng của Stack
- Hàng đợi

Stack - Khái niệm

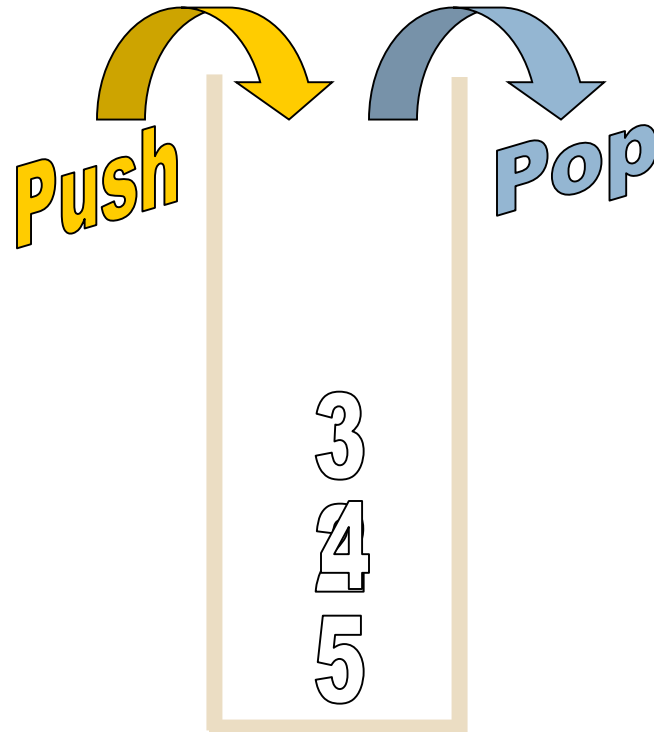
3

- Stack là một danh sách mà các đối tượng được **thêm vào** và **lấy ra** chỉ ở một đầu của danh sách (A stack is simply a list of elements with insertions and deletions permitted at one end)
- Vì thế, việc thêm một đối tượng vào Stack hoặc lấy một đối tượng ra khỏi Stack được thực hiện theo cơ chế LIFO (Last In First Out - *Vào sau ra trước*)
- Các đối tượng có thể được thêm vào Stack bất kỳ lúc nào nhưng chỉ có đối tượng **thêm vào sau cùng** mới được phép lấy ra khỏi Stack

Stack – Các thao tác

4

- Stack hỗ trợ 2 thao tác chính:
 - **“Push”**: Thao tác **thêm** 1 đối tượng vào Stack
 - **“Pop”**: Thao tác **lấy** 1 đối tượng ra khỏi Stack
- Ví dụ:
5 3 2 - - 4



Stack – Các thao tác

5

- Stack cũng hỗ trợ một số thao tác khác:
 - ▣ **isEmpty()**: Kiểm tra xem Stack có rỗng không
 - ▣ **Top()**: Trả về giá trị của phần tử nằm ở đầu Stack mà không hủy nó khỏi Stack. Nếu Stack rỗng thì lỗi sẽ xảy ra

Stack – Hiện thực Stack

(Implementation of a Stack)

6

Mảng 1 chiều



Danh sách LK



Hiện thực Stack dùng mảng

(Implementation of a Stack using Array)

7

- Có thể tạo một Stack bằng cách khai báo một **mảng 1 chiều** với kích thước tối đa là N (ví dụ: $N = 1000$)
 - ▣ Stack có thể chứa tối đa N phần tử đánh số từ 0 đến $N-1$
- Phần tử nằm ở đỉnh Stack sẽ có chỉ số là **top** (\rightarrow lúc đó trong Stack đang chứa $top+1$ phần tử)
- Như vậy, để khai báo một Stack, ta cần một **mảng 1 chiều** list, và 1 biến số nguyên **top** cho biết chỉ số của đỉnh Stack:

```
struct Stack {  
    DataType list[N];  
    int top;  
};
```



Hiện thực Stack dùng mảng

(Implementation of a Stack using Array)

8

- Lệnh **top** = 0 sẽ tạo ra một Stack S rỗng
- Giá trị của **top** sẽ cho biết số phần tử hiện hành có trong Stack
- Khi cài đặt bằng mảng 1 chiều, Stack bị giới hạn kích thước nên cần xây dựng thêm một thao tác phụ cho Stack:
 - ▣ **isFull()**: Kiểm tra xem Stack có đầy chưa, vì khi Stack đầy, việc gọi đến hàm Push() sẽ phát sinh ra lỗi



Hiện thực Stack dùng mảng

(Implementation of a Stack using Array)

9

□ Khởi tạo Stack:

```
void Init (Stack &s)
{
    s.top = 0;
}
```



Hiện thực Stack dùng mảng

(Implementation of a Stack using Array)

10

- Kiểm tra Stack rỗng hay không:

```
int isEmpty(Stack s)
{
    if (s.top==0)
        return 1; // stack rỗng
    else
        return 0;
}
```



Hiện thực Stack dùng mảng

(Implementation of a Stack using Array)

11

- Kiểm tra Stack đầy hay không:

```
int isFull(Stack s)
{
    if (s.top >= N)
        return 1;
    else
        return 0;
}
```



Hiện thực Stack dùng mảng

(Implementation of a Stack using Array)

12

- Thêm một phần tử x vào Stack

```
void Push (Stack &s, DataType x)
{
    if (!isFull(s)) // stack chưa đầy
    {
        s.list[s.top]=x;
        s.top++;
    }
}
```



Hiện thực Stack dùng mảng

(Implementation of a Stack using Array)

13

- Trích thông tin và huỷ phần tử ở đỉnh Stack

```
DataType Pop (Stack &s)
```

```
{
```

```
    DataType x;
```

```
    if (!Empty(s)) // stack khác rỗng
```

```
    {
```

```
        s.top--;
```

```
        x = s.list[s.top];
```

```
    }
```

```
    return x;
```

```
}
```



Hiện thực Stack dùng mảng

(Implementation of a Stack using Array)

14

Nhận xét:

- ▣ Các thao tác trên đều làm việc với chi phí $O(1)$
- ▣ Việc cài đặt Stack thông qua mảng một chiều đơn giản và khá hiệu quả
- ▣ Tuy nhiên, hạn chế lớn nhất của phương án cài đặt này là giới hạn về kích thước của Stack (N)
 - Giá trị của N có thể quá nhỏ so với nhu cầu thực tế hoặc quá lớn sẽ làm lãng phí bộ nhớ

Hiện thực Stack dùng DSLK

(Implementation of a Stack using Linked List)

15

- Có thể tạo một Stack bằng cách sử dụng một danh sách liên kết đơn (DSLK)
- Khai báo các cấu trúc:

```
struct Node
{
    DataType data;
    Node *pNext;
};

struct Stack
{
    Node *top;
};
```

Hiện thực Stack dùng DSLK

(Implementation of a Stack using Linked List)

16

□ Khởi tạo Stack:

```
void Init(Stack &t)
{
    t.top = NULL;
}
```


Hiện thực Stack dùng DSLK

(Implementation of a Stack using Linked List)

17

- Kiểm tra xem Stack có rỗng không:

```
int  isEmpty (Stack t)
{
    return  t.top == NULL ? 1 : 0;
}
```

Hiện thực Stack dùng DSLK

(Implementation of a Stack using Linked List)

18

- Thêm một phần tử x vào Stack:

```
void Push (Stack &t, DataType x)
{
    Node *p = new Node;
    if (p==NULL) { cout<<"Không đủ bộ nhớ"; return; }
    p->data = x;
    p->pNext = NULL;
    Thêm phần tử vào đầu danh sách
    if (t.top==NULL) // if (Empty(l))
        t.top = p;
    else{
        p->pNext = t.top;
        t.top = p;
    }
}
```

Hiện thực Stack dùng DSLK

(Implementation of a Stack Using Linked List)

19

- Trích thông tin và hủy phần tử ở đỉnh Stack:

```
DataType Pop (Stack &t)
{
    if (t.top==NULL) {
        cout<<"Stack rỗng"; return NULLDATA; }
    DataType x;
    Node *p = t.top;
    t.top = t.top->pNext;
    p->pNext = NULL;
    x = p->data;
    delete p;
    return x;
}
```

Lấy và xóa phần tử ở đầu danh sách

Stack - Ứng dụng

20

- Stack thích hợp lưu trữ các loại dữ liệu mà trình tự truy xuất ngược với trình tự lưu trữ
- Một số ứng dụng của Stack:
 - ▣ Trong trình biên dịch (thông dịch), khi thực hiện các thủ tục, Stack được sử dụng để lưu môi trường của các thủ tục
 - ▣ Lưu dữ liệu khi giải một số bài toán của lý thuyết đồ thị (như tìm đường đi)
 - ▣ Khử đệ qui
 - ▣ Ứng dụng trong các bài toán tính toán biểu thức
 - ▣ ...

Stack - Ứng dụng

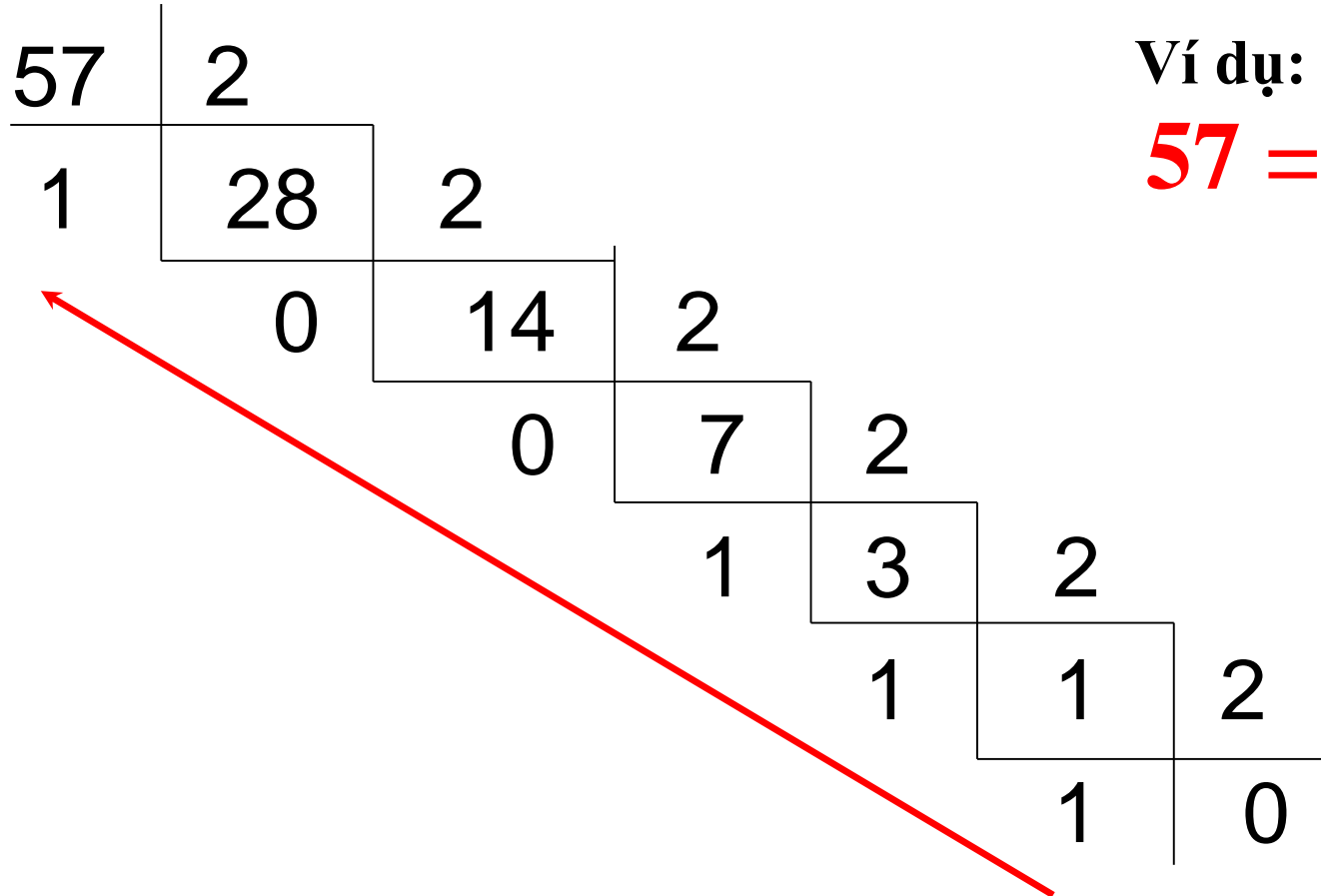
21

Ví dụ: thủ tục Quick_Sort dùng Stack để khử đệ qui:

- Bước 1. $l=1; r=n$;
- Bước 2. Chọn phần tử giữa $x=a[(l+r) / 2]$
- Bước 3. Phân hoạch (l, r) thành $(l1, r1)$ và $(l2, r2)$ bằng cách xét:
 - ▣ y thuộc $(l1, r1)$ nếu $y \leq x$
 - ▣ y thuộc $(l2, r2)$ ngược lại
- Bước 4. Nếu phân hoạch $(l2, r2)$ có nhiều hơn 1 phần tử thì thực hiện:
 - ▣ Cất $(l2, r2)$ vào Stack
 - ▣ Nếu $(l1, r1)$ có nhiều hơn 1 phần tử thì thực hiện:
 - $l = l1$
 - $r = r1$
 - Quay lên bước 2
 - ▣ Ngược lại
 - Lấy (l, r) ra khỏi Stack, nếu Stack khác rỗng thì quay lên bước 2, ngược lại thì dừng

Stack - Ứng dụng

Bài tập: đổi số từ cơ số 10 sang cơ số x



Ví dụ: $57 = ???_2$

$$57 = 111001_2$$

```

void main()
{
    Stack s;
    int coso, so, sodu;
    Init(s);
    // Nhập số cần chuyển vào so ...
    // Nhập cơ số cần chuyển vào coso...
    while (so != 0)
    {
        sodu = so % coso;
        Push (s, sodu); // push so du vao stack
        so = so/coso;
    }
    cout<<"Kết quả: ";
    while (!isEmpty(s))
        cout<<Pop(s); // pop so du ra khoi stack
}

```

Stack - Ứng dụng

24

- Thuật toán Ba Lan ngược
(Reverse Polish Notation – RPN)
 - ▣ Định nghĩa RPN:
 - Biểu thức toán học trong đó các toán tử được viết sau toán hạng và không dùng dấu ngoặc
 - ▣ Phát minh bởi Jan Lukasiewics một nhà khoa học Ba Lan vào những năm 1950

RPN

Infix	:	toán tử viết giữa toán hạng
Postfix (RPN):		toán tử viết sau toán hạng
Prefix	:	toán tử viết trước toán hạng

Examples:

INFIX

RPN (POSTFIX)

PREFIX

A + B

A B +

+ A B

A * B + C

A B * C +

+ * A B C

A * (B + C)

A B C + *

* A + B C

A - (B - (C - D))

A B C D - - -

- A - B - C D

A - B - C - D

A B - C - D -

- - - A B C D

Lượng giá biểu thức RPN

Kỹ thuật ***gạch dưới***:

1. Duyệt từ trái sang phải của biểu thức cho đến khi gặp toán tử.
2. Gạch dưới 2 toán hạng ngay trước toán tử và kết hợp chúng bằng toán tử trên
3. Lặp đi lặp lại cho đến hết biểu thức.

Ví dụ $2*((3+4)-(5-6))$

2 3 4 + 5 6 - - *

→ 2 3 4 + 5 6 - - *

→ 2 **7** 5 6 - - *

→ 2 7 5 6 - - *

→ 2 7 **-1** - *

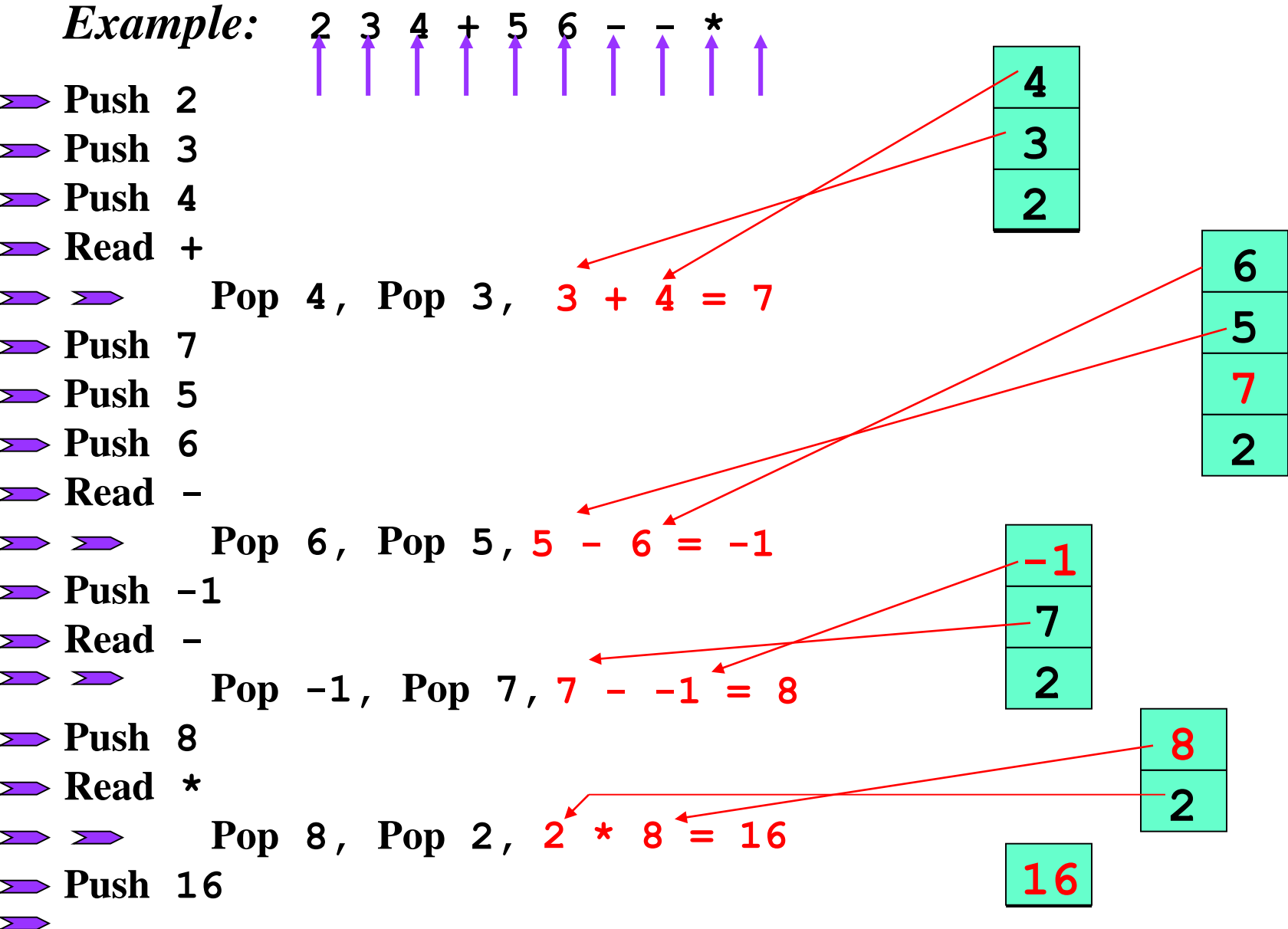
→ 2 7 -1 - * → 2 **8** * → 2 8 * → **16**

Thuật toán tính giá trị

1. Khởi tạo Stack rỗng (*chứa hằng hoặc biến*).
2. Lặp cho đến khi kết thúc biểu thức:
 - Đọc 01 phần tử của biểu thức (*hằng, biến, phép toán*).
 - Nếu phần tử là hằng hay biến: đưa vào Stack.
 - Ngược lại:
 - Lấy ra 02 phần tử của Stack.
 - Áp dụng phép toán cho 02 phần tử vừa lấy ra.
 - Đưa kết quả vào Stack.
3. Giá trị của biểu thức chính là phần tử cuối cùng của Stack.

$2 * ((3 + 4) - (5 - 6))$

Example:



Chuyển infix thành postfix

1. Khởi tạo Stack rỗng (chứa các phép toán).
2. Lặp cho đến khi kết thúc biểu thức:

Đọc 01 phần tử của biểu thức

(01 phần tử có thể là hằng, biến, phép toán, “)”

hay “(”).

Nếu phần tử là:

2.1 “(” : đưa vào Stack.

2.2 “)” : lấy các phần tử của Stack ra
cho đến khi gặp “(” trong Stack.

Chuyển infix thành postfix

2.3 Một phép toán: $+$ $-$ $*$ $/$

Nếu **Stack rỗng**: đưa vào Stack.

Nếu Stack khác rỗng và **phép toán có độ ưu tiên cao hơn phần tử ở đầu Stack**: đưa vào Stack.

Nếu Stack khác rỗng và phép toán có **độ ưu tiên thấp hơn hoặc bằng phần tử ở đầu Stack**:

- lấy phần tử từ Stack ra;
- sau đó lặp lại việc so sánh với phần tử ở đầu Stack.

Chuyển infix thành postfix

2.4 Hằng hoặc biến: đưa vào kết quả.

3. Lấy hết tất cả các phần tử của Stack ra.

Độ ưu tiên

<input type="checkbox"/> + , _	1
<input type="checkbox"/> * , /	2
<input type="checkbox"/> ^	3

Example: $(A+B*C) / (D-(E-F))$

➤ Push (
➤ Display A
➤ Push +
➤ Display B
➤ Push *
➤ Display C
➤ Read)
➤ Pop *, Display *,
➤ Pop +, Display +, Pop (
➤ Push /
➤ Push (
➤ Display D
➤ Push -
➤ Push (
➤ Display E
➤ Push -
➤ Display F
➤ Read)
➤ Pop -, Display -, Pop (
➤ Read)
➤ Pop -, Display -, Pop (
➤ Pop /, Display /

Output

A

AB

ABC

ABC*

ABC*+

ABC*+D

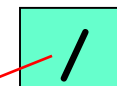
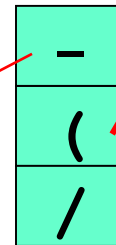
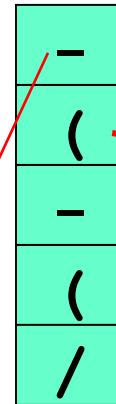
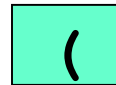
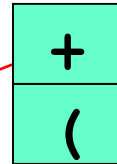
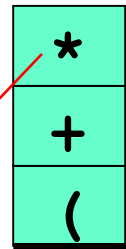
ABC*+DE

ABC*+DEF

ABC*+DEF-

ABC*+DEF--

ABC*+DEF--/



Ví dụ

$A + (B * C - (D / E ^ F) * G) * H$

$S = [];$

$KQ = ""$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H \quad \text{A} + (B * C - (D / E ^ F) * G) * H$$

S=[;];

KQ="A"

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

S=[+({};

KQ=ABC

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$$S = [+ (\{ \})];$$

$$KQ = ABC *$$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$$S = [+(-)];$$

$$KQ = ABC *$$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$S = [+(- (];$

$KQ = ABC * D$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$$S = I + (C / B);$$

$$KQ = ABC * D$$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$S = [+(-(/];$

$KQ = ABC * DE$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$$S = [+ (((/)] ; ;$$

$$KQ = ABC * DE$$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$S = [+(-(/^)];$

$KQ = ABC * DEF$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$$S = [+ (- (/ ^))] ;$$

$$KQ = ABC * DEF ^ /$$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$S = [+(-*);$

$KQ = ABC * DEF ^ /$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$$S = [+(-*);$$

$$KQ = \cancel{A}BC^* \cancel{DEF}^{\wedge} // G$$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

S=[+{()*},];

KQ=ABC*DEF^/G*-

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

S=[+,*],;

KQ=ABC*DEF^/G*-

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$S = [+ *];$

$KQ = ABC * DEF ^ / G * - H$

Ví dụ

$$A + (B * C - (D / E ^ F) * G) * H$$

$S = \{+, *, \}$;

~~$KQ = A * B * C * D * E * F * G * H * +$~~

Nội dung

51

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

Queue - Khái niệm

52

- 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 vào Queue luôn diễn ra ở cuối Queue và việc **lấy** một đối tượng ra khỏi Queue luôn diễn ra ở đầu Queue
- Vì thế, việc thêm một đối tượng vào Queue hoặc lấy một đối tượng ra khỏi Queue được thực hiện theo cơ chế FIFO (First In First Out - *Vào trước ra trước*)

Queue - Khái niệm

53

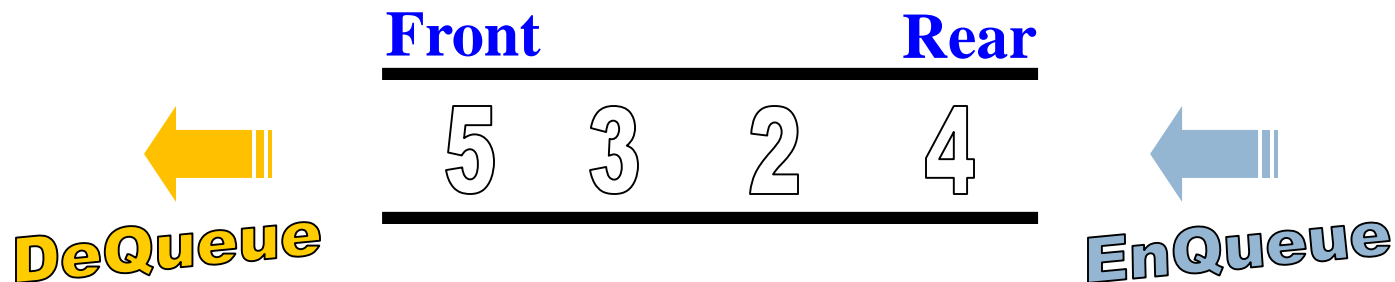
Imaging



Queue – Các thao tác

54

- Hàng đợi hỗ trợ các thao tác:
 - ▣ **EnQueue()**: Thêm đối tượng vào cuối (rear) Queue
 - ▣ **DeQueue()**: Lấy đối tượng ở đầu (front) Queue ra khỏi Queue
- Ví dụ:
5 3 2 - - 4



Queue – Các thao tác

55

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

Queue – Hiện thực Queue

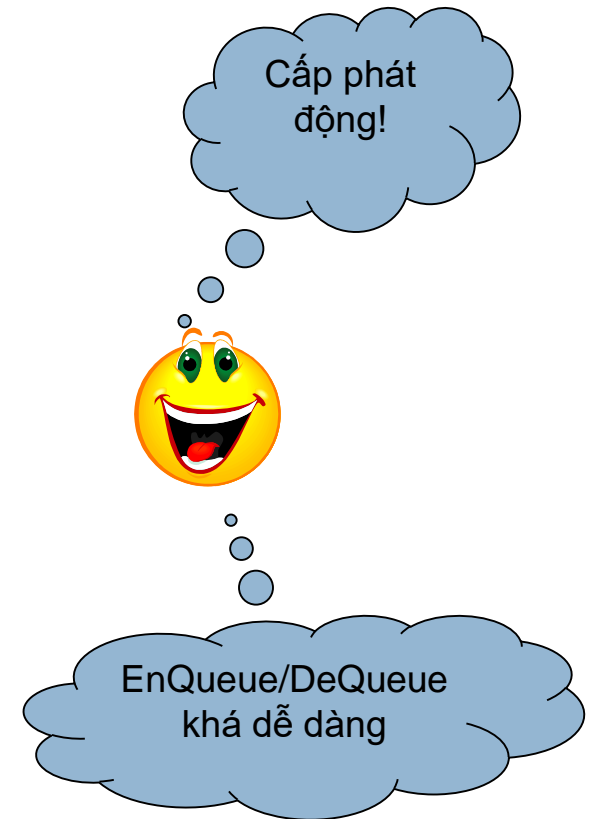
(Implementation of a Queue)

56

Mảng 1 chiều



Danh sách LK



Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

57

- Có thể tạo một Queue bằng cách sử dụng một mảng 1 chiều theo kiểu xoay vòng (coi phần tử a_{n-1} kề với phần tử a_0)
⇒ Hàng đợi chứa tối đa N phần tử
- Phần tử ở đầu hàng đợi sẽ có chỉ số front
- Phần tử ở cuối hàng đợi sẽ có chỉ số rear
- The limitation of an array implementation is that the queue cannot grow and shrink dynamically as per the requirement

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

58

- Dùng mảng: Có xu hướng dời về cuối mảng
- Hai cách hiện thực:

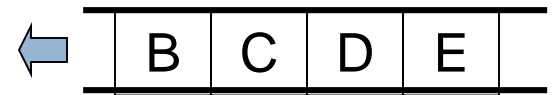
▣ **Khi lấy một phần tử ra thì đồng thời dời ô lên một vị trí:**



Ban đầu

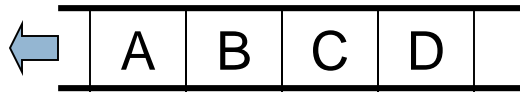


Lấy ra 1 phần tử:
dời tất cả về trước để
trống chỗ thêm vào

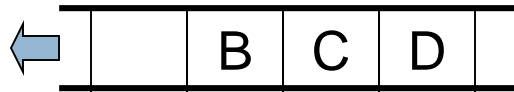


Thêm vào 1 phần tử

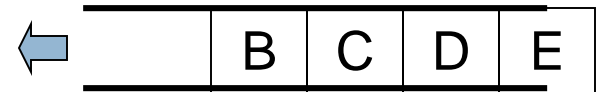
▣ **Khi lấy một phần tử ra thì không dời ô lên:**



Ban đầu



Lấy ra 1 phần tử



Thêm vào 1 phần tử

Hiện thực Queue dùng mảng

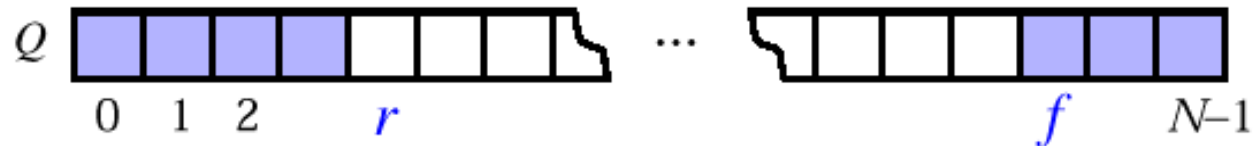
(Implementation of a Queue using Array)

59

- Trạng thái Queue lúc bình thường:



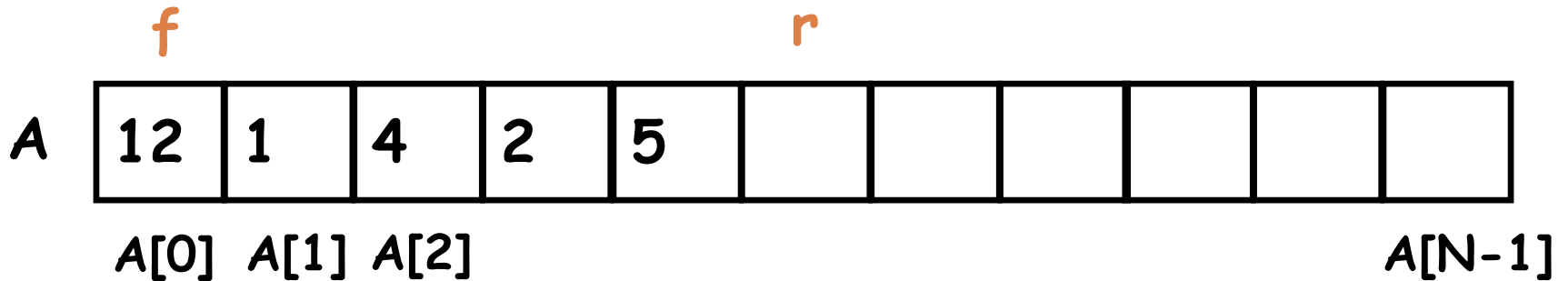
- Trạng thái Queue lúc xoay vòng:



Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

60



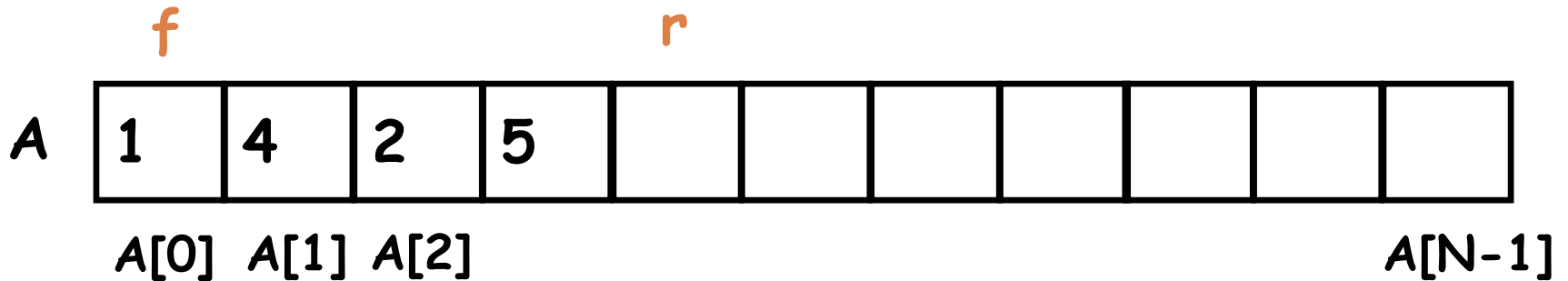
DeQueue(Q)

Cách dùng mảng 1

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

61



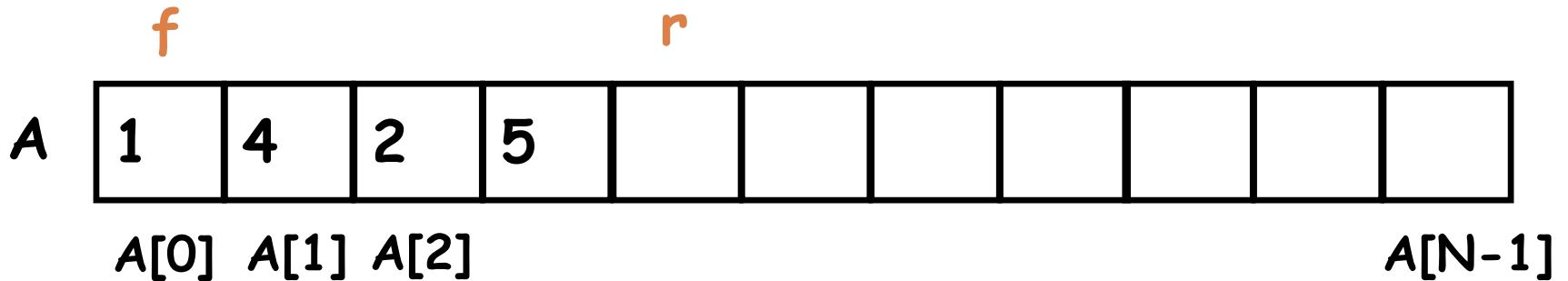
DeQueue(Q)

Cách dùng mảng 1

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

62

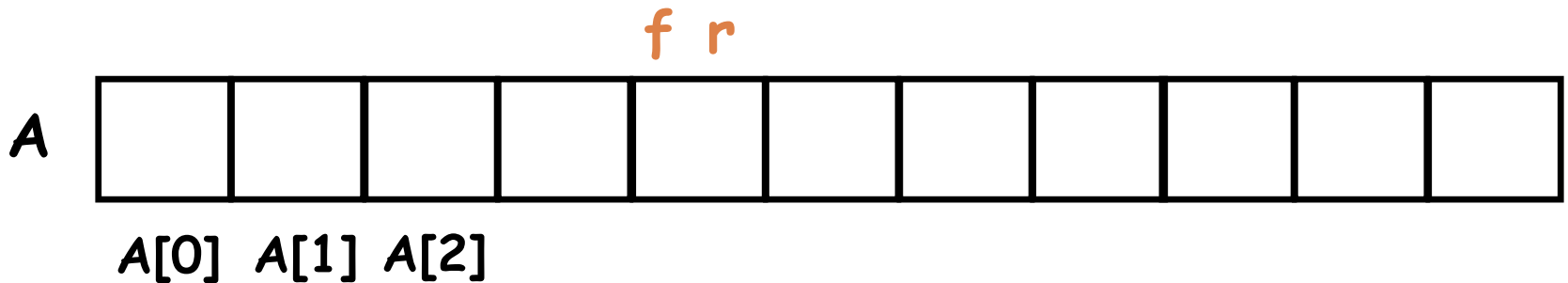
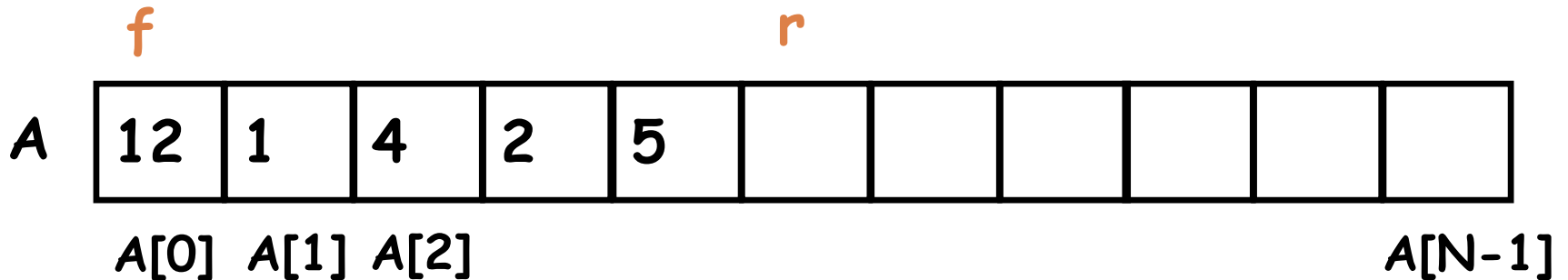


Cách dùng mảng 1

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

63



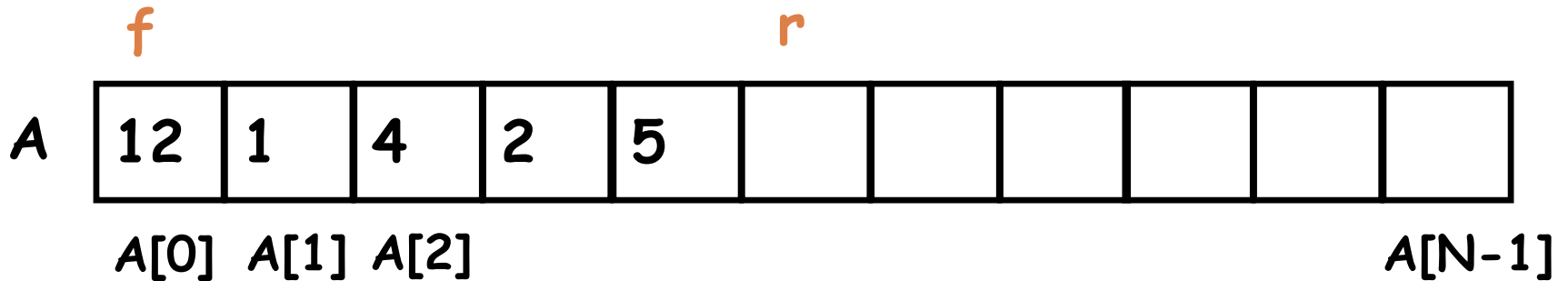
Cách dùng mảng 2

Empty queue $f=r$

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

64



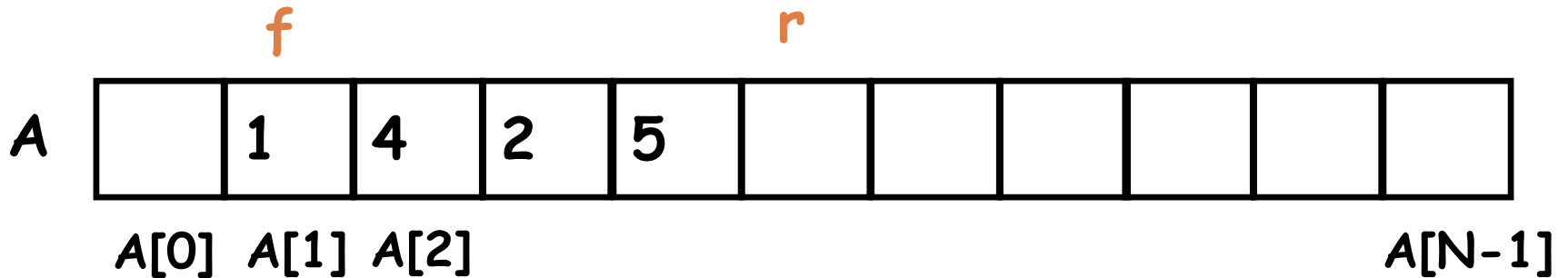
DeQueue(Q)

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

65



DeQueue(Q)

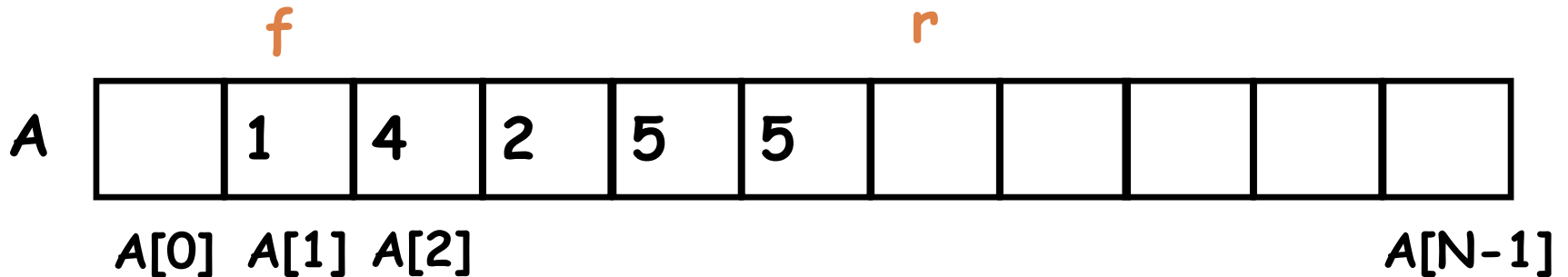
EnQueue(5, Q)

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

66



DeQueue(Q)

EnQueue(5, Q)

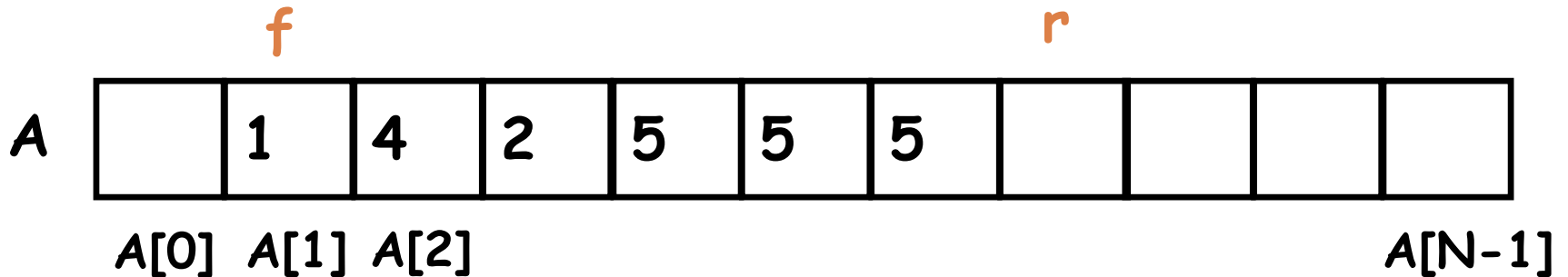
EnQueue(5, Q)

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

67



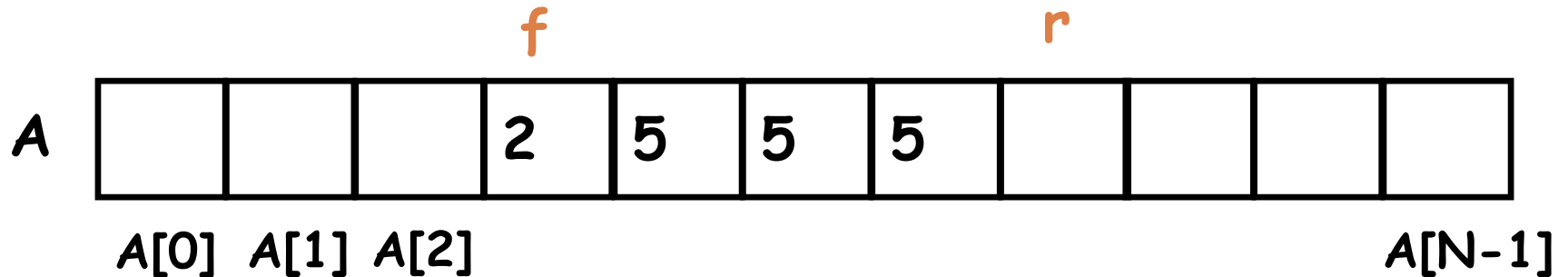
DeQueue(Q)
EnQueue(5, Q)
EnQueue(5, Q)
DeQueue(Q)
DeQueue(Q)

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

68



DeQueue(Q)

EnQueue(5, Q)

EnQueue(5, Q)

DeQueue(Q)

DeQueue(Q)

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

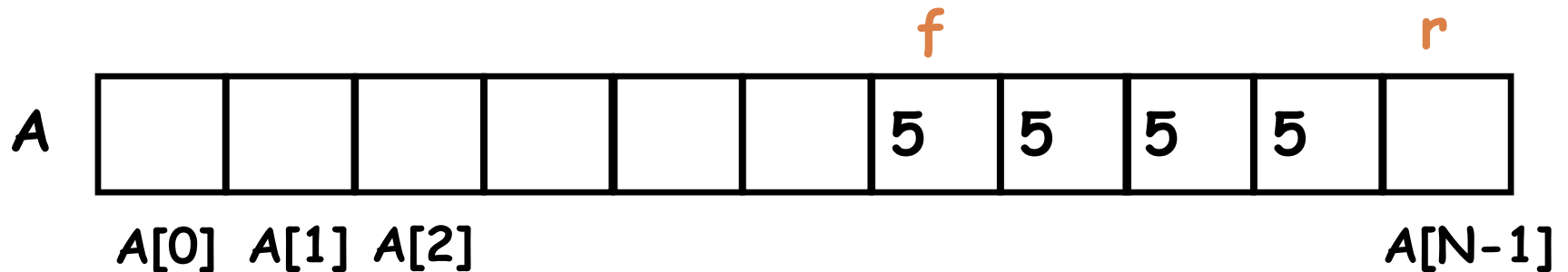
EnQueue(5, Q),

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

69



DeQueue(Q)

EnQueue(5, Q)

EnQueue(5, Q)

DeQueue(Q)

DeQueue(Q)

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

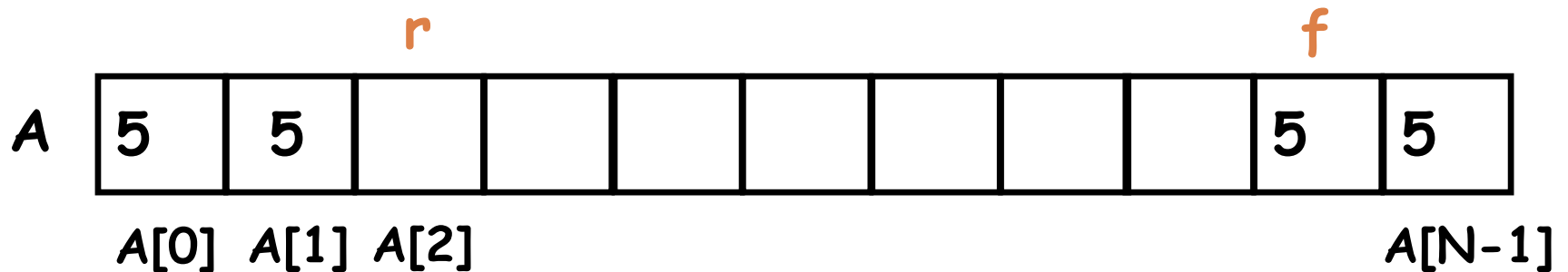
EnQueue(5, Q),

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

70



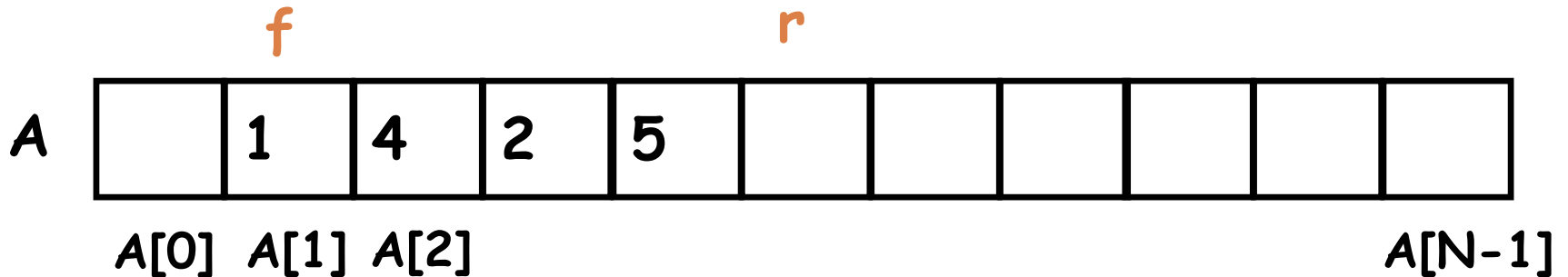
DeQueue(Q), EnQueue(5,Q), DeQueue(Q),
EnQueue(5,Q),.....

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

72



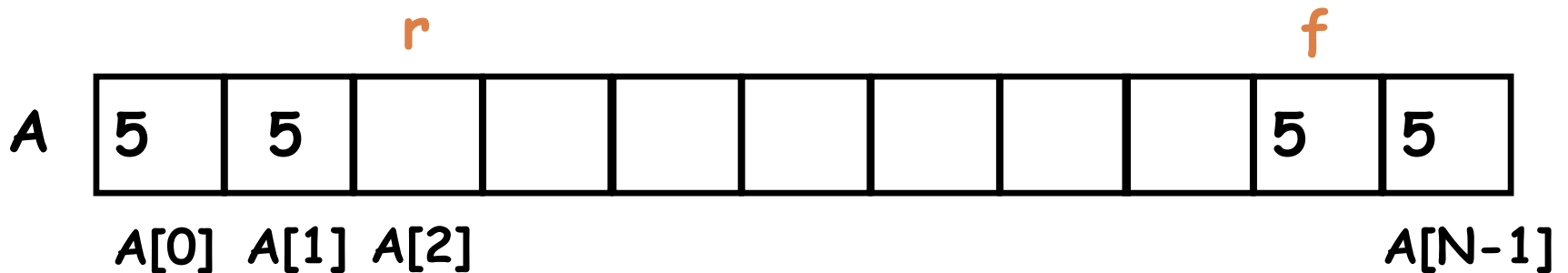
size(Q): if $(r \geq f)$ then return $(r - f)$
else return $N - (f - r)$

Cách dùng mảng 2

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

73



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)

74

- Để khai báo một Queue, ta cần khai báo:
 - ▣ một mảng một chiều list,
 - ▣ hai biến 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
{
    int    front, rear;
    DataType list[N];
};
```

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

75

- Do khi cài đặt bằng mảng một chiều, hàng đợi bị giới hạn kích thước nên cần xây dựng thêm một thao tác phụ cho hàng đợi:
 - ▣ **isFull()**: Kiểm tra xem hàng đợi có đầy chưa

Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

76

□ 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 == 0)
        return 1;
    if (q.front == q.rear)
        return 1;
    return 0;
}
```

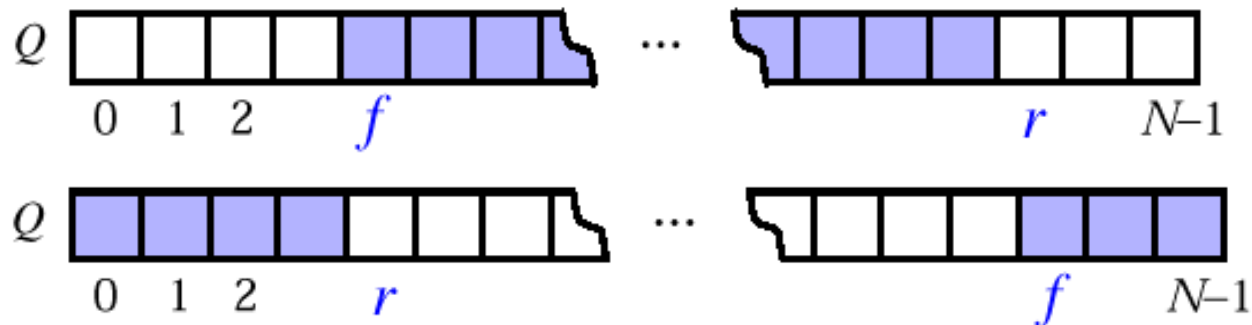
Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

77

- Kiểm tra hàng đợi đầy hay không:

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



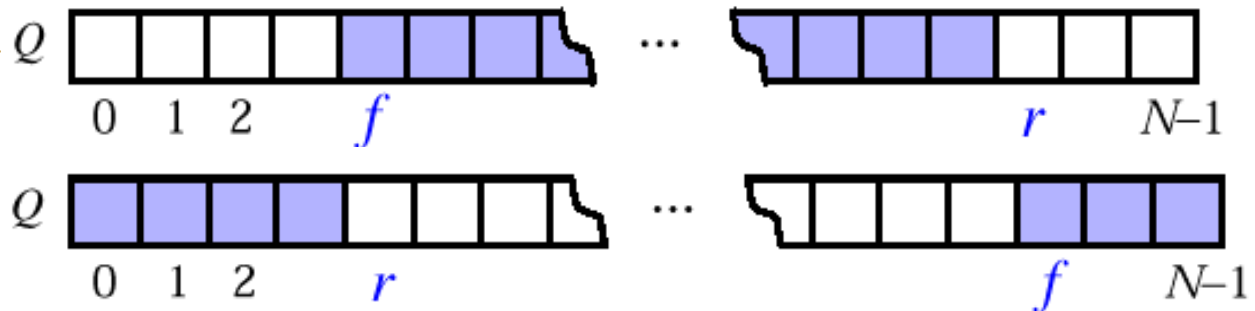
Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

78

- 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;
}
```



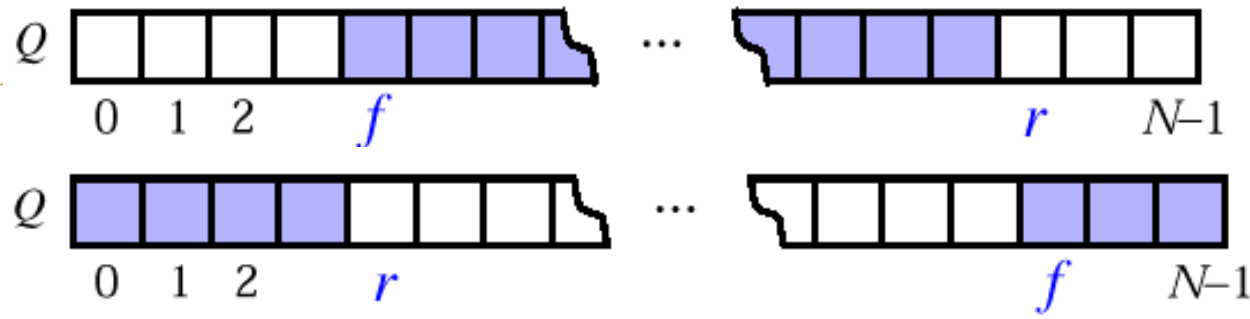
Hiện thực Queue dùng mảng

(Implementation of a Queue using Array)

79

- Trích, huỷ phần tử ở đầu hàng đợi:

```
DataType DeQueue (Queue &q)
{
    if (Empty(q)) return NULLDATA;
    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:

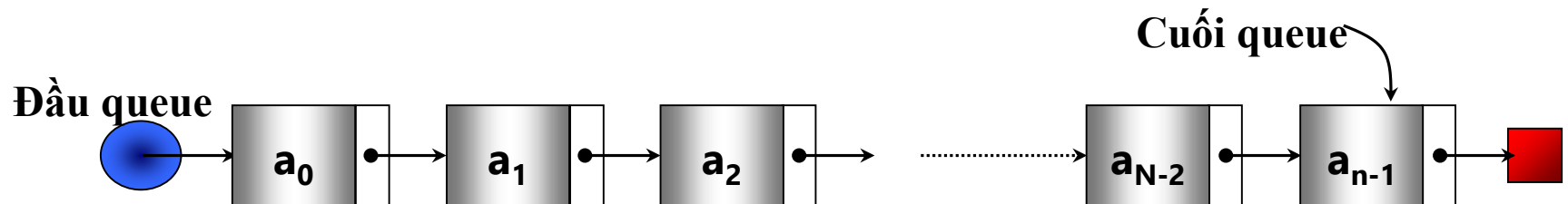
```
DataType      Front (Queue q)
{
    if (isEmpty(q)) return NULLDATA;
    return q.list[q.front];
}
```

Hiện thực Queue dùng DSLK

(Implementation of a Queue using Linked List)

81

- Có thể tạo một hàng đợi sử dụng một DSLK đơn
- Phần tử đầu DSLK (phead) sẽ là phần tử đầu Queue (front), phần tử cuối DSLK (ptail) sẽ là phần tử cuối Queue (rear)



Hiện thực Queue dùng DSLK

(Implementation of a Queue using Linked List)

82

- 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)

83

- 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)

84

- Thêm một phần tử p vào cuối Queue:

```
int EnQueue (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)

85

- Trích và huỷ phần tử ở đầu Queue:

```
DataType DeQueue (Queue &q)
{
    if (isEmpty(q)) return NULLDATA;
    Node *p = q.front;
    q.front = q.front->pNext;
    p->pNext = NULL;
    if (q.front==NULL) q.rear = NULL;
    DataType x = p->data;
    delete p;
    return x;
}
```

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)) return NULLDATA;
    return q.front->data;
}
```

Hiện thực Queue dùng DSLK

(Implementation of a Queue using Linked List)

87

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 **Enqueue** sẽ có độ phức tạp $O(n)$

Queue - Ứng dụng

88

- 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,