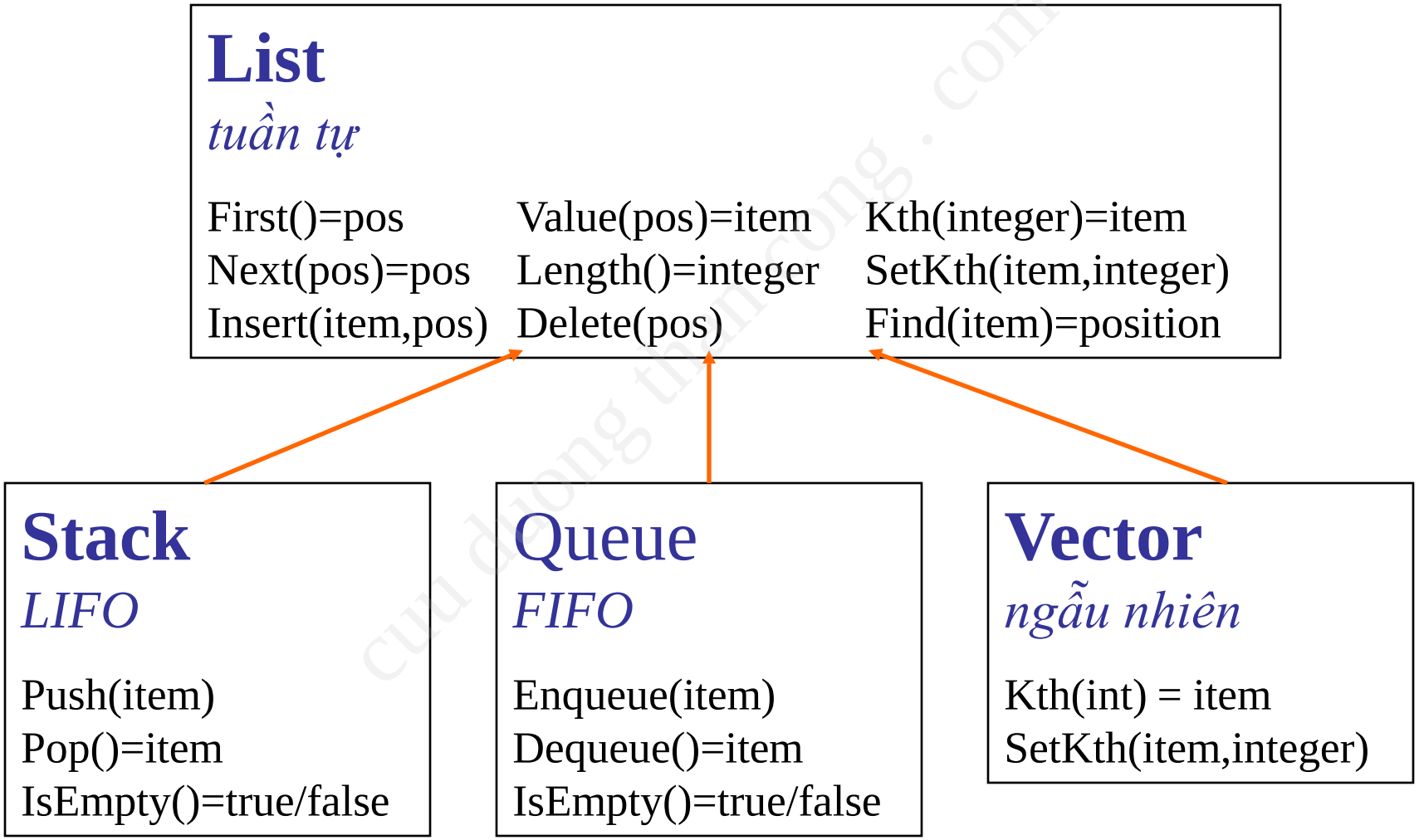
Ngăn xếp - Hàng đợi

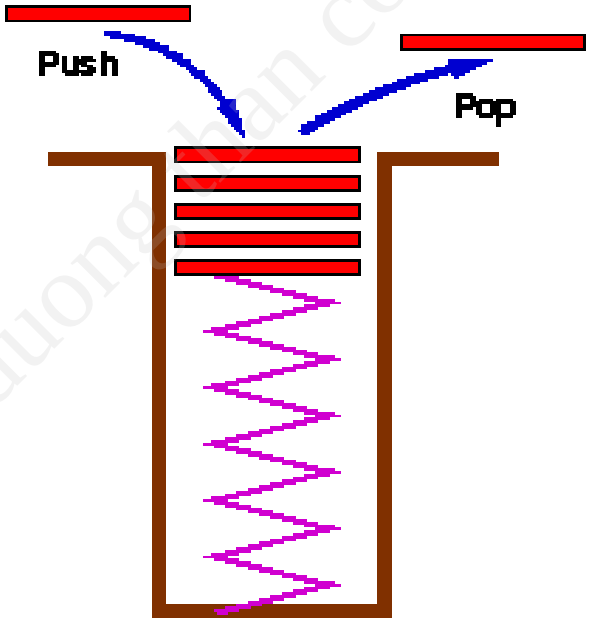
(Stack – Queue)

Phân cấp đặc biệt hoá



Stack

Stack



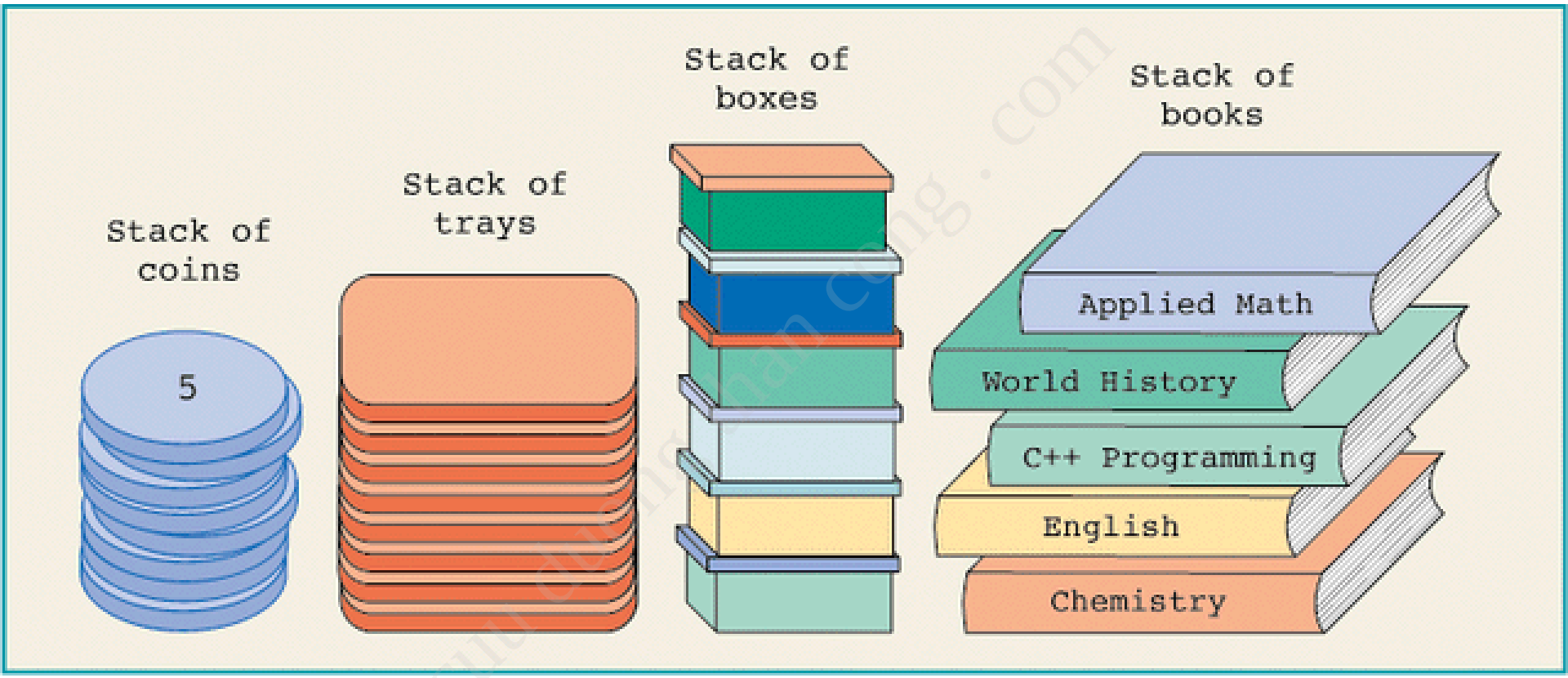


Figure 18-1 Various types of stacks

Stack

Stack là một vật chứa (container) các đối tượng làm việc theo cơ chế LIFO (*Last In First Out*) ⇒ Việc thêm một đối tượng vào stack hoặc lấy một đối tượng ra khỏi stack được thực hiện theo cơ chế "*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 là một CTDL trừu tượng (ADT) tuyến tính hỗ trợ 2 thao tác chính:

- **Push**(o): Thêm đối tượng o vào đầu stack
- **Pop**(): Lấy đối tượng ở đầu stack ra khỏi stack và trả về giá trị của nó. Nếu stack rỗng thì lỗi sẽ xảy ra.

Stack

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 Nếu stack rỗng thì lỗi sẽ xảy ra.

Ví dụ

push

push

push

Ví dụ

push

push

push

pop

push

Last in,
First out.

Stack

Stack	List
Push	InsertHead
Pop	PickHead
IsEmpty	IsEmpty

Stack – Cài đặt

```
typedef struct Stack
{
   List L;
}

void Push(Stack &S, data x)
{
        InsertHead(S.L,x);
}
```

Stack – Cài đặt

```
Node* Pop(Stack &S)
{
        return RemoveHead(S.L);
}

Data Top(Stack S)
{
        return S.L.pHead->info;
}
```

Hiện thực Stack dùng mảng

Có thể tạo một stack bằng cách khai báo một mảng 1 chiều với kích thước tối đa là N

Stack có thể chứa tối đa N phần tử đánh số từ 0 đến N-1.

Phần tử nằm ở đầu stack sẽ có chỉ số t (lúc đó trong stack đang chứa t+1 phần tử)

Hiện thực Stack dùng mảng

Để khai báo một stack, ta cần một mảng 1 chiều S, biến nguyên t cho biết chỉ số của đầu stack và hằng số N cho biết kích thước tối đa của

```
typedef struct Stack{
    Data  D [N];
    int     t;
    }
```

Hiện thực Stack dùng mảng

Lệnh t = 0 sẽ tạo ra một stack S rỗng.

Giá trị của t sẽ cho biết số phần tử hiện hành có trong

Khi cài đặt bằng mảng 1 chiều, stack có kích thước tối đa nên cần xây dựng thêm một thao tác phụ cho stack:

- **IsFull**(): Kiểm tra xem stack có đầy chưa.
- Khi stack đầy, việc gọi đến hàm push() sẽ phát sinh ra lỗi.

Hiện thực Stack dùng mảng

Kiểm tra stack rỗng hay không

```
char IsEmpty(Stack S)
{  if(S.t == 0) // stack rỗng
     return 1;
   else
     return 0;
}
```

Kiểm tra stack đầy hay không

```
char IsFull(Stack S)
{  if(S.t >= N) // stack đầy
     return 1;
   else
     return 0;
}
```

Hiện thực Stack dùng mảng

Thêm một phần tử x vào stack S

```
void Push(Stack S,Data x)
{  if(S.t < N) // stack chưa đầy
   {S.D[t] = x; S.t++; }
   else puts("Stack đầy");
}
```

Trích thông tin và huỷ phần tử ở đỉnh stack S

```
Data Pop(Stack S)
{  Data    x;
   if(S.t > 0) // stack khác rỗng
   { S.t--; x = S.D[t]; return x;}
   else puts("Stack rỗng")
}
```

Hiện thực Stack dùng mảng

Xem thông tin của phần tử ở đỉnh stack S

```
Data Top()
{ Data   x;
  if(t > 0) // stack khác rỗng
  { x = S.D[S.t-1];
    return   x;
  }
  else  puts("Stack rỗng")
}
```

Hiện thực Stack dùng mảng

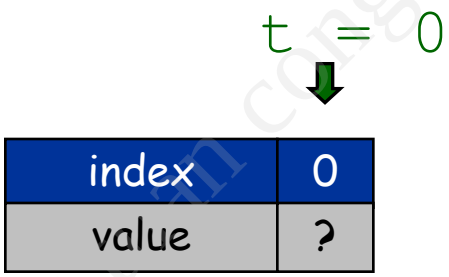
Nhận xét:

- 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.

```
Stack stack = new Stack();
push("A");
push("B");
push("C");
push("D");
push("E");
push("F");
cout<<pop();
cout<<pop();
push("G");
cout<<pop();
cout<<pop();
cout<<pop();
```

Ví dụ

- `Stack stack = new Stack ();`



Ví dụ

- `Stack stack = new Stack();`
- `push("A");`

`t = 1`

index	0
value	A

Ví dụ

- `Stack stack = new Stack();`
- `push("A");`
- `push("B");`

`t = 2`

index	0	1
value	A	B

Ví dụ

- `Stack stack = new Stack();`
- `push("A");`
- `push("B");`
- `push("C");`

`t = 3`

index	0	1	2	3
value	A	B	C	?

Ví dụ

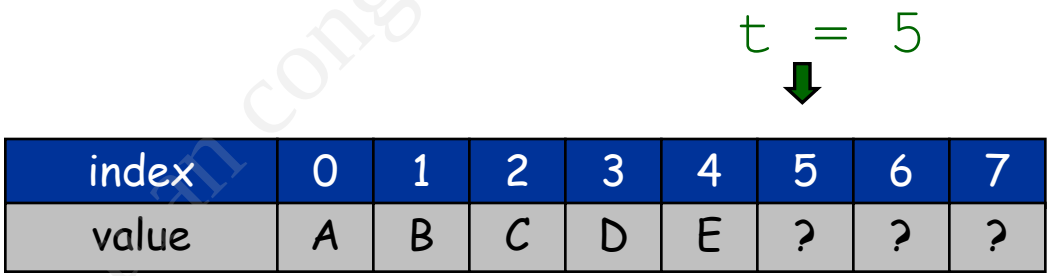
- `Stack stack = new Stack();`
- `push("A");`
- `push("B");`
- `push("C");`
- `push("D");`

`t = 4`

index	0	1	2	3
value	A	B	C	D

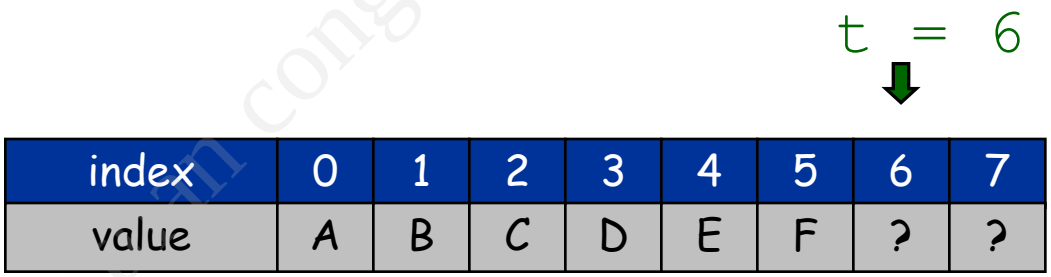
Ví dụ

- `Stack stack = new Stack();`
- `push("A");`
- `push("B");`
- `push("C");`
- `push("D");`
- `push("E");`



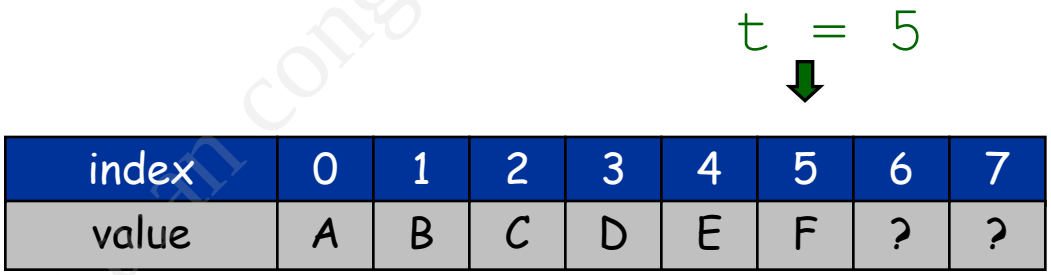
Ví dụ

- `Stack stack = new Stack();`
- `push("A");`
- `push("B");`
- `push("C");`
- `push("D");`
- `push("E");`
- `push("F");`



Ví dụ

- `Stack stack = new Stack();`
- `push("A");`
- `push("B");`
- `push("C");`
- `push("D");`
- `push("E");`
- `push("F");`
- `cout<<pop();`



Ví dụ

- `Stack stack = new Stack();`
- `push("A");`
- `push("B");`
- `push("C");`
- `push("D");`
- `push("E");`
- `push("F");`
- `cout<<pop();`
- `cout<<pop();`

`t = 4`

index	0	1	2	3	4	5	6	7
value	A	B	C	D	E	F	?	?

F
E

Ví dụ

- `Stack stack = new Stack();`
- `push("A");`
- `push("B");`
- `push("C");`
- `push("D");`
- `push("E");`
- `push("F");`
- `cout<<pop();`
- `cout<<pop();`
- `push("G");`

t = 5

index	0	1	2	3	4	5	6	7
value	A	B	C	D	G	F	?	?

```
F
E
```

Ví dụ

- `Stack stack = new Stack();`
- `push("A");`
- `push("B");`
- `push("C");`
- `push("D");`
- `push("E");`
- `push("F");`
- `cout<<pop();`
- `cout<<pop();`
- `push("G");`
- `cout<<pop();`

`t = 4`

index	0	1	2	3	4	5	6	7
value	A	B	C	D	G	F	?	?

F
E
G

Ví dụ

- `Stack stack = new Stack();`
- `push("A");`
- `push("B");`
- `push("C");`
- `push("D");`
- `push("E");`
- `push("F");`
- `cout<<pop();`
- `cout<<pop();`
- `push("G");`
- `cout<<pop();`
- `cout<<pop();`

t = 3

index	0	1	2	3	4	5	6	7
value	A	B	C	D	G	F	?	?

F
E
G
D

Ví dụ

- `Stack stack = new Stack();`
- `push("A");`
- `push("B");`
- `push("C");`
- `push("D");`
- `push("E");`
- `push("F");`
- `cout<<pop();`
- `cout<<pop();`
- `push("G");`
- `cout<<pop();`
- `cout<<pop();`
- `cout<<pop();`

`t = 2`

index	0	1	2	3	4	5	6	7
value	A	B	C	D	G	F	?	?

```
F
E
G
D
C
```

Ứng dụng của Stack

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 của Stack

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

1. l:=1; r:=n;
2. Chọn phần tử giữa x:=a[(l+r) div 2];
3. Phân hoạch (l,r) thành (l1,r1) và (l2,r2) bằng cách xét:
   - y thuộc (l1,r1) nếu y≤x;
   - y thuộc (l2,r2) ngược lại;

Ứng dụng của Stack

4. Nếu phân hoạch (l2,r2) có nhiều hơn 1 phần tử thực hiện:
   - Cất (l2,r2) vào Stack;
   - Nếu (l1,r1) có nhiều hơn 1 phần tử thực hiện:

     l=l1;

     r=r1;

     Goto 2;
   - Ngược lại

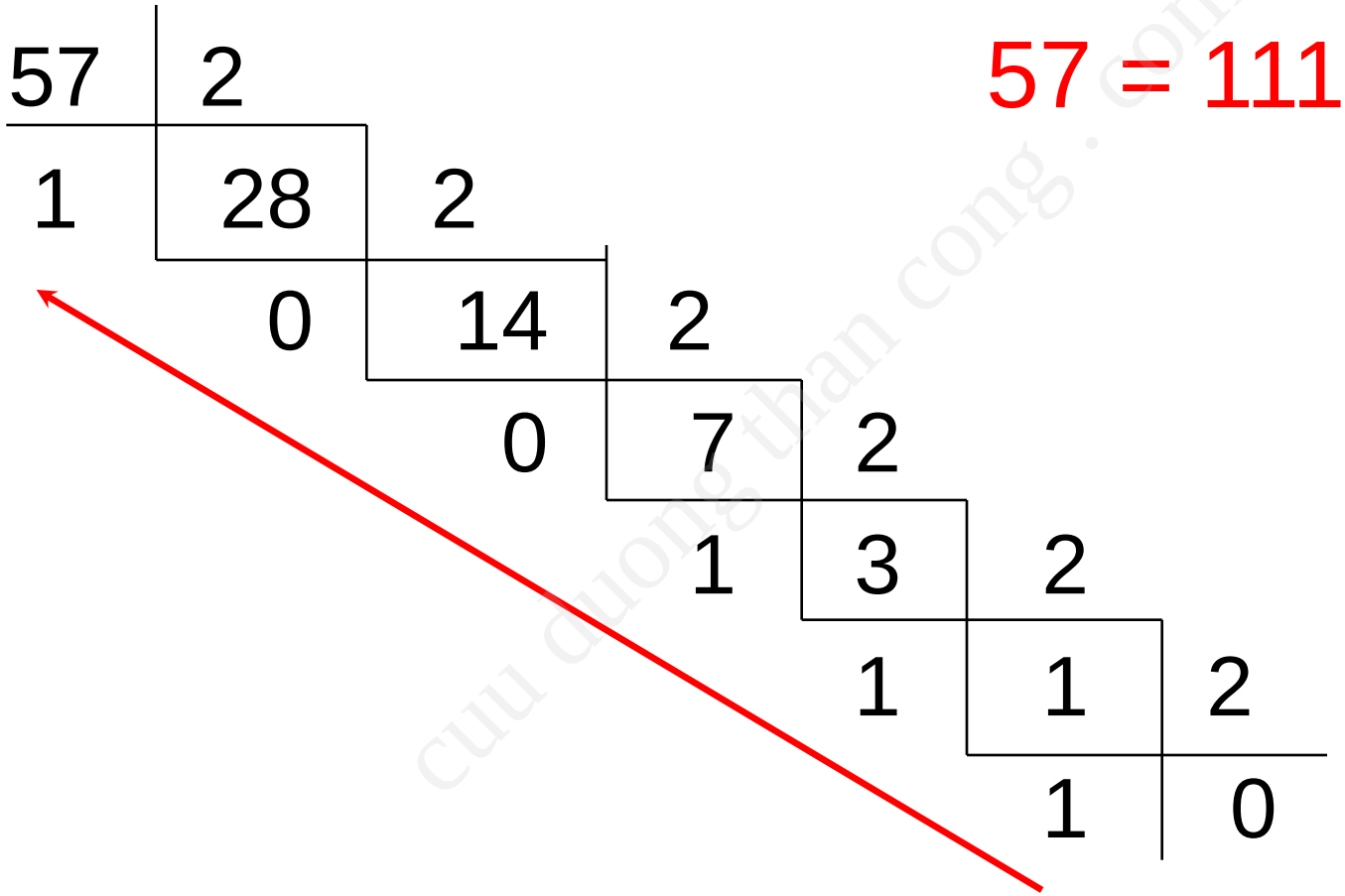
     Lấy (l,r) ra khỏi Stack nếu Stack khác rỗng và Goto 2;

     Nếu không dừng;

Đổi cơ số (10 sang x)

- $57 = ???_2$

Đổi cơ số



$57 = 111001_2$

Đổi cơ số

```
void main()
{
    stack s;
    int coso, so, sodu;
    init(s);
    while (so != 0)
     {
        sodu = so % coso;
        push(s, sodu);  // push so du vao stack
        so = so / coso;
     }
     printf("Kết quả: ");
     while(!empty(s))
        cout<<pop(s);  // pop so du ra khoi stack
}
```

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 nhưng 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 - *

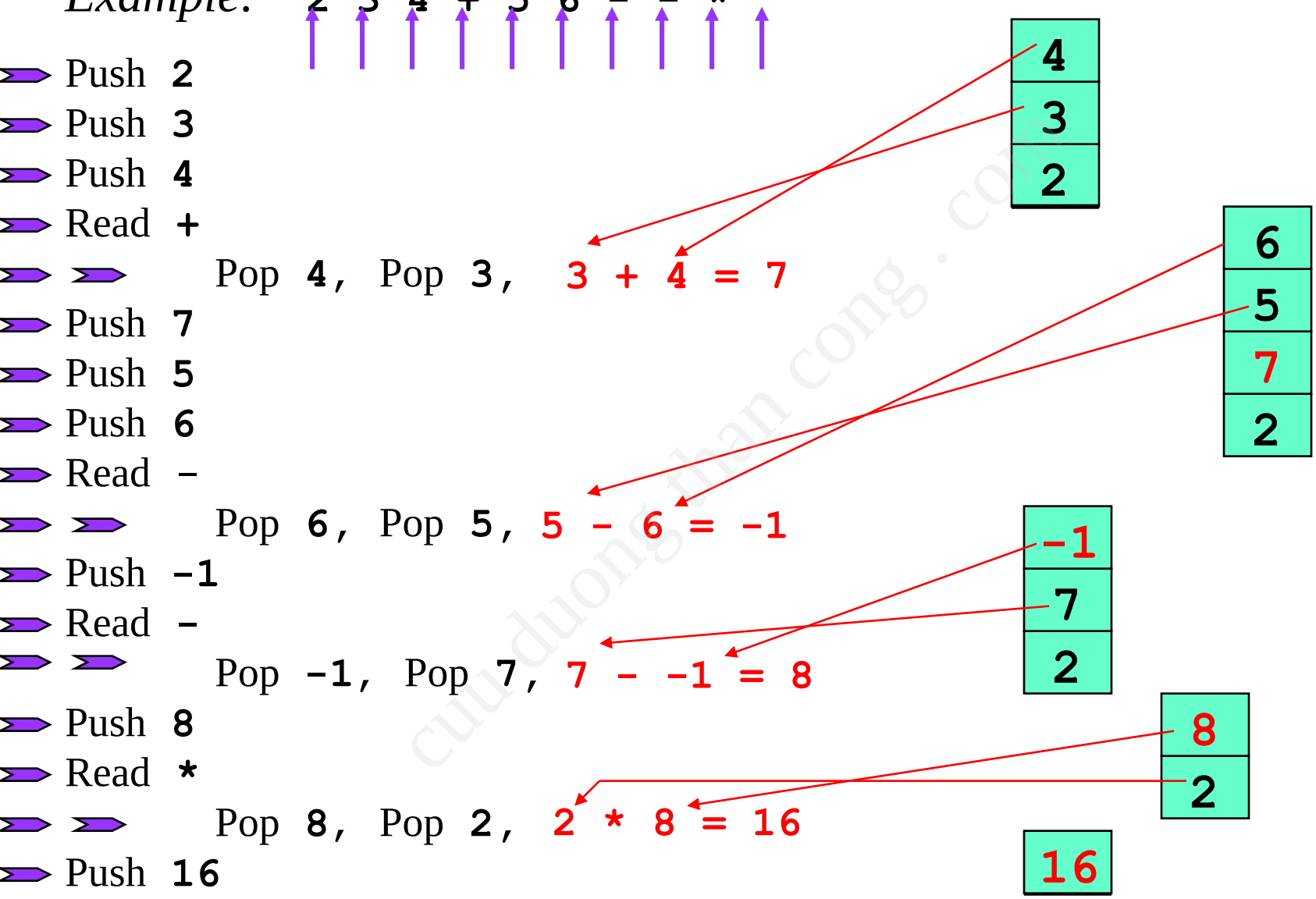
→ 2 7 -1 - * → 2 8 * → 2 8 * → 16

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: `2 3 4 + 5 6 - - *`



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.

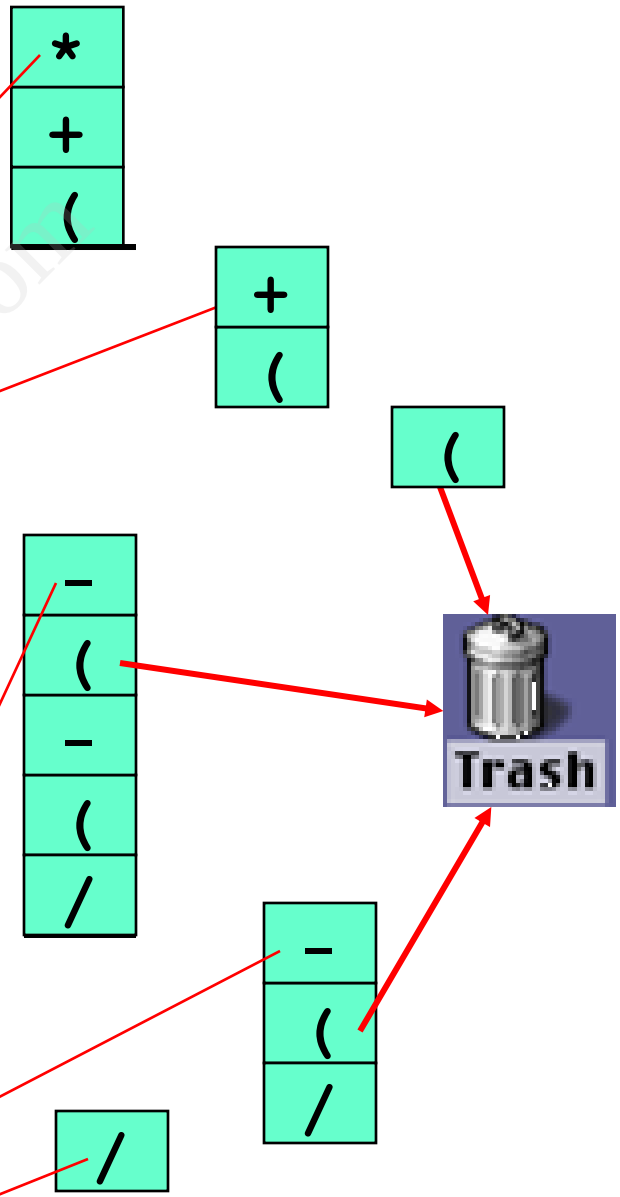
3. Lấy hết tất cả các phần tử của Stack ra.

Độ ưu tiên

- + , _ 1
- *, / 2
- ^ 3

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

	Output
Push **(**	
Display **A**	**A**
Push **+**	
Display **B**	**AB**
Push *	
Display **C**	**ABC**
Read **)**	
Pop *****, Display *****,	**ABC***
Pop **+**, Display **+**, Pop **(**	**ABC*+**
Push **/**	
Push **(**	
Display **D**	**ABC*+D**
Push **–**	
Push **(**	
Display **E**	**ABC*+DE**
Push **–**	
Display **F**	**ABC*+DEF**
Read **)**	
Pop –, Display –, Pop (	**ABC*+DEF–**
Read **)**	
Pop –, Display –, Pop (	**ABC*+DEF––**
Pop /, Display /	**ABC*+DEF––/**



Ví dụ

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

S=[];

KQ=“”

Ví dụ

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

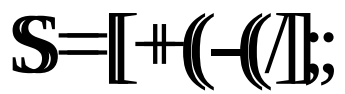
A + (B*C - (D/E^F) * G) * H

S=[+(-(];

KQ=ABC*D

Ví dụ

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



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=[+(-(/休];

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=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*-

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=ABC*DEF^/G*-H*+

Bài tập

3 ? 4 ? 5 ? 6 = 17

3 + 4 ? 5 ? 6 = 17

3 + 4 * 5 – 6 = 17

Hàng đợi (Queue)

Hàng đợi (Queue)

Hàng đợi là một vật chứa (container) các đối tượng làm việc theo cơ chế FIFO (*First In First Out*) ⇒ việc thêm một đối tượng vào hàng đợi hoặc lấy một đối tượng ra khỏi hàng đợi được thực hiện theo cơ chế "*Vào trước ra trước*".

Các đối tượng có thể được thêm vào hàng đợi bất kỳ lúc nào nhưng chỉ có đối tượng thêm vào đầu tiên mới được phép lấy ra khỏi hàng đợi.

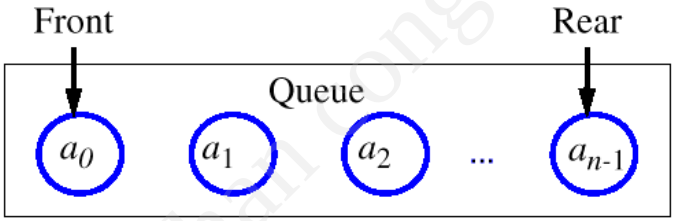
Hàng đợi (Queue)

"**Enqueue**":Thao tác thêm một đối tượng vào hàng đợi

"**Dequeue**":Thao tác lấy một đối tượng ra khỏi hàng đợi.

Hàng đợi (Queue)

Việc thêm một đối tượng vào hàng đợi luôn diễn ra ở cuối hàng đợi và một phần tử luôn được lấy ra từ đầu hàng đợi.

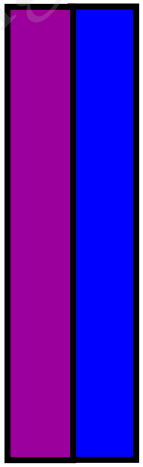


Trong tin học, CTDL hàng đợi có nhiều ứng dụng: tổ chức lưu vết các quá trình tìm kiếm theo chiều rộng và quay lui, vét cạn, tổ chức quản lý và phân phối tiến trình trong các hệ điều hành, tổ chức bộ đệm bàn phím, ...

Ví dụ

EnQueue

EnQueue



Ví dụ

EnQueue

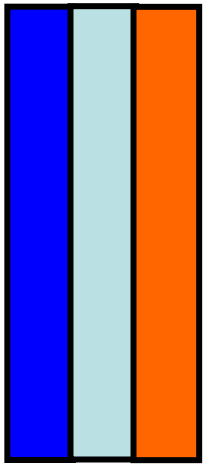
EnQueue

DeQueue

EnQueue

EnQueue

First in,
First out (FIFO).



Hàng đợi (Queue)

Hàng đợi là một CTDL trừu tượng (ADT) tuyến tính

Hàng đợi hỗ trợ các thao tác:

- **EnQueue**(o):Thêm đối tượng o vào cuối hàng đợi
- **DeQueue**(): Lấy đối tượng ở đầu queue ra khỏi hàng đợi và trả về giá trị của nó. Nếu hàng đợi rỗng thì lỗi sẽ xảy ra.
- **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

Queue	List
EnQueue	InsertTail
DeQueue	RemoveHead
IsEmpty	IsEmpty

Queue – Cài đặt

```
typedef struct Queue
{
   List L;
}

void EnQueue(Queue &Q, data x)
{
        InsertTail(Q.L,x);
}
```

Queue – Cài đặt

```
Node* DeQueue(Queue &Q)
{
        return RemoveHead(Q.L);
}

Node* Front(Queue &Q)
{
        return Q.L.pHead->info;
}
```

Biểu diễn Queue dùng mảng

Có thể tạo một hàng đợi bằng cách sử dụng một mảng 1 chiều với kích thước tối đa là N (ví dụ, N=1000) theo kiểu xoay vòng (coi phần tử a_{n-1} kề với phần tử a_0) $\Rightarrow$ Hàng đợi chứa tối đa N phần tử.

Phần tử ở đầu hàng đợi (front element) sẽ có chỉ số f.

Phần tử ở cuối hàng đợi (rear element) sẽ có chỉ số r .

Biểu diễn Queue dùng mảng

Để khai báo một hàng đợi, ta cần:

- một mảng một chiều Q,
- hai biến nguyên f, r cho biết chỉ số của đầu và cuối của hàng đợi
- hằng số N cho biết kích thước tối đa của hàng đợi.

Biểu diễn Queue dùng mảng

Ngoài ra, khi dùng mảng biểu diễn hàng đợi, cần dùng một giá trị đặc biệt, ký hiệu là NULLDATA, để gán cho những ô còn trống trên hàng đợi. Giá trị này là một giá trị nằm ngoài miền xác định của dữ liệu lưu trong hàng đợi..

Biểu diễn Queue dùng mảng

Hàng đợi có thể được khai báo cụ thể như sau:

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

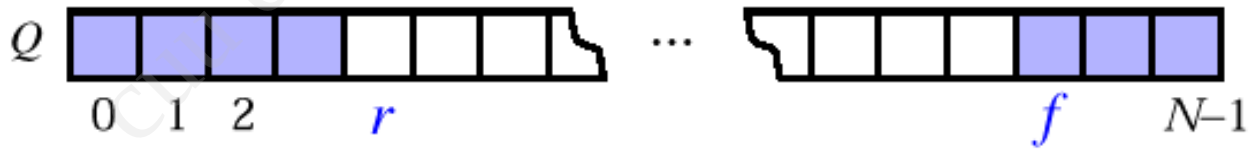
Do khi cài đặt bằng mảng một chiều, hàng đợi có kích thước tối đa nên cần xây dựng thêm một thao tác phụ cho hàng đợi:

- **IsFull**(): Kiểm tra xem hàng đợi có đầy chưa.

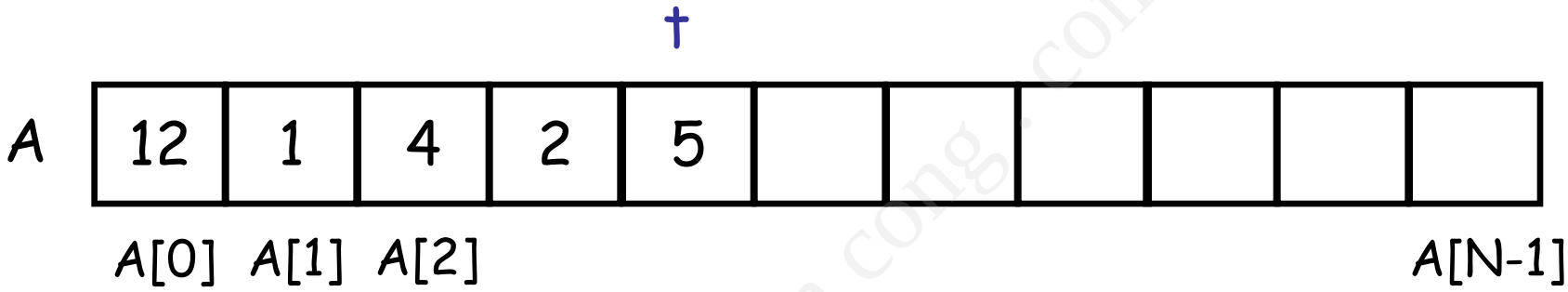
Biểu diễn Queue dùng mảng

Trạng thái hàng đợi lúc bình thường:

Trạng thái hàng đợi lúc xoay vòng:

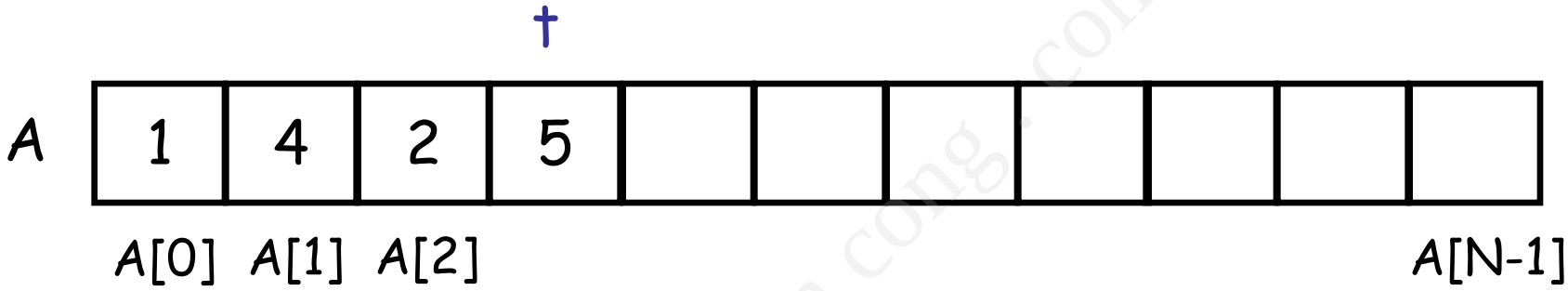


Cài đặt dùng mảng



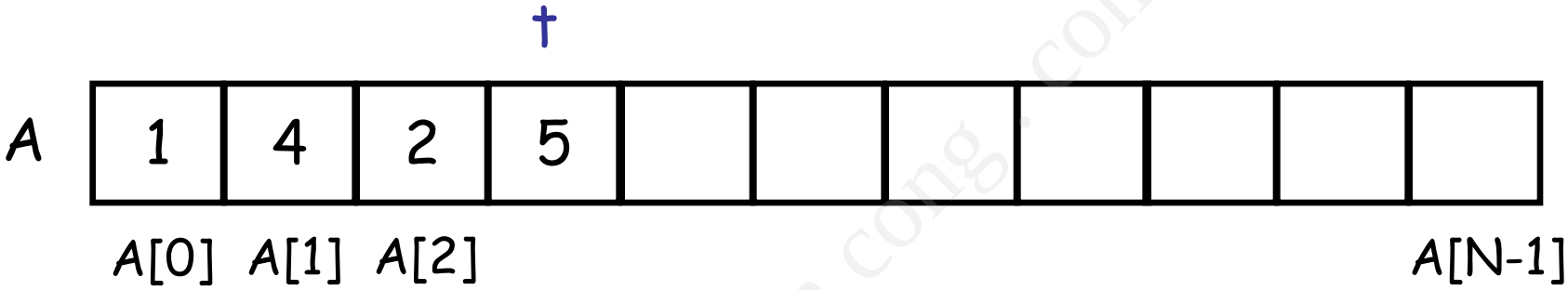
DeQueue(Q)

Cài đặt dùng mảng

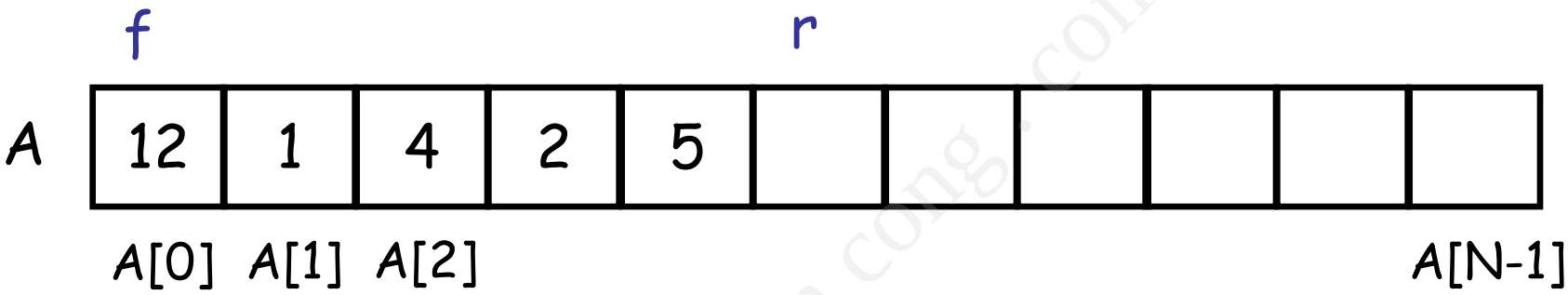


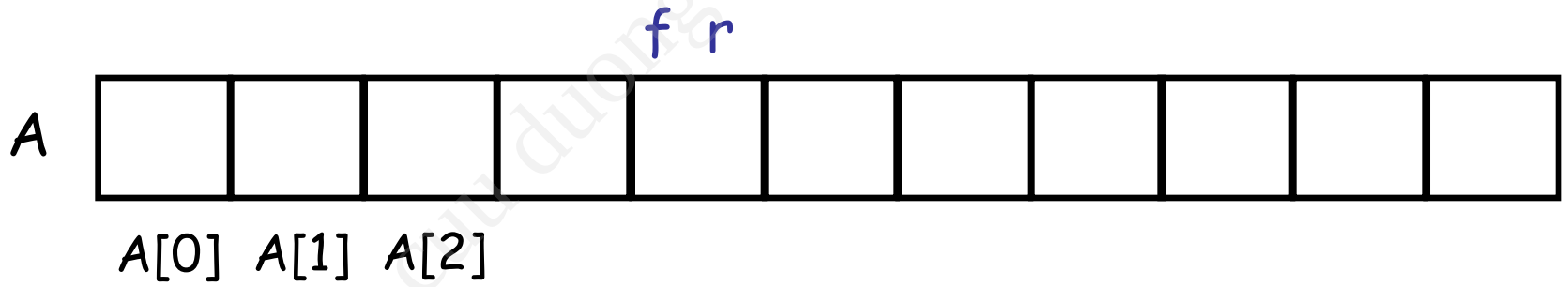
DeQueue(Q)

Cài đặt dùng mảng



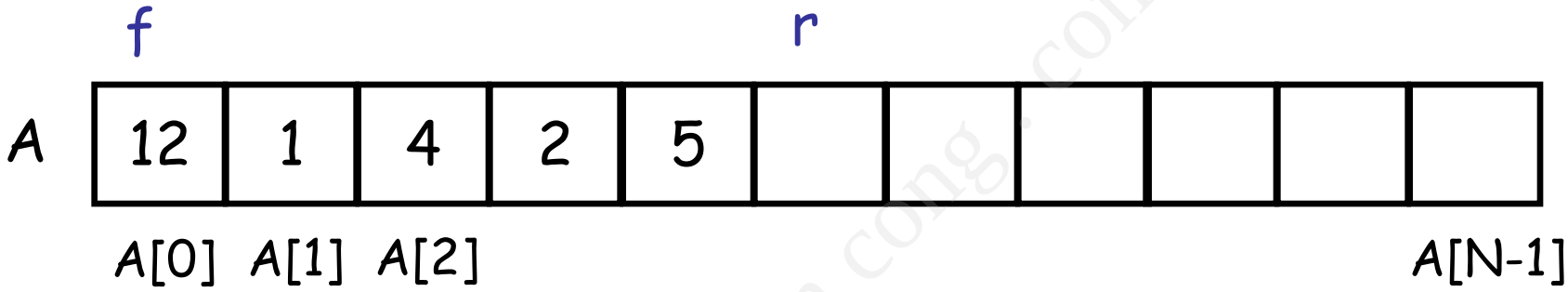
Cài đặt dùng mảng





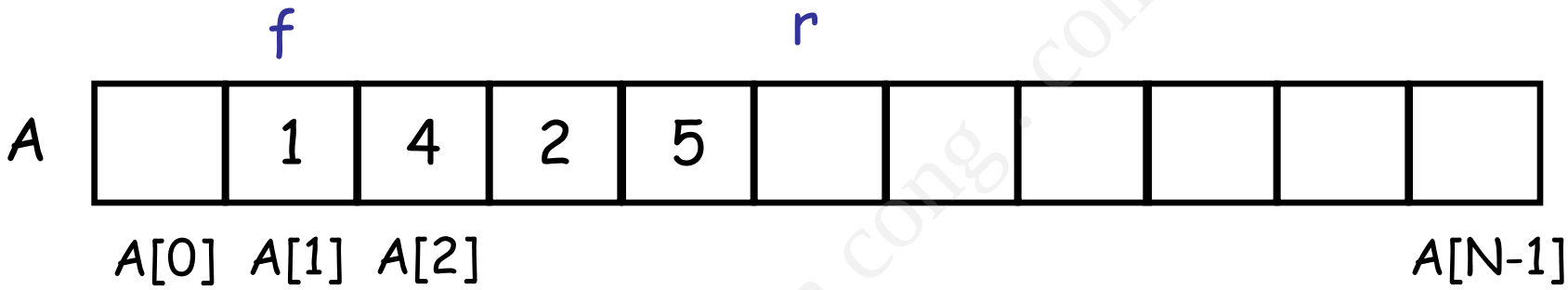
Empty queue f=r

Cài đặt dùng mảng



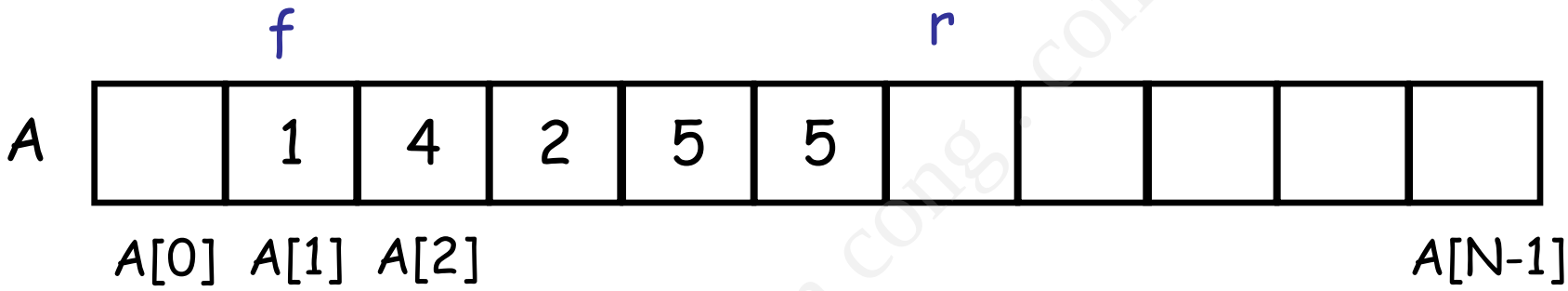
DeQueue(Q)

Cài đặt dùng mảng



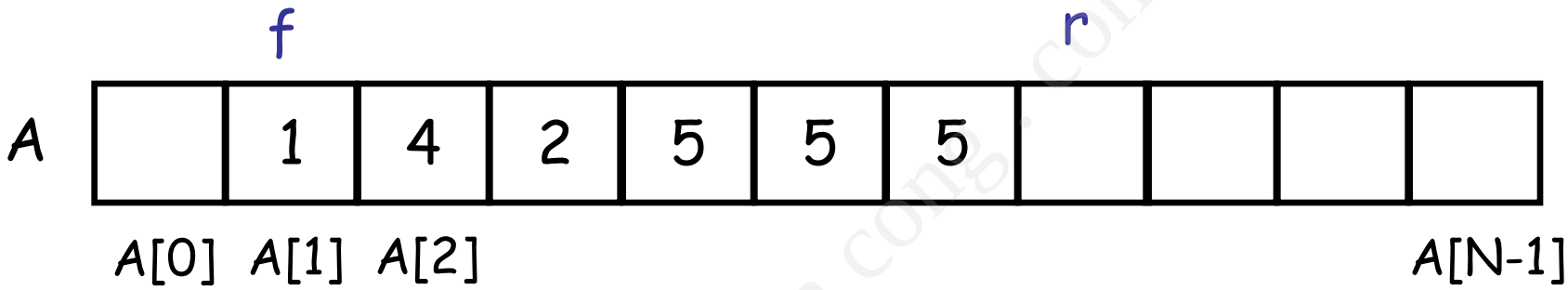
DeQue
EnQueue(5,Q)

Cài đặt dùng mảng



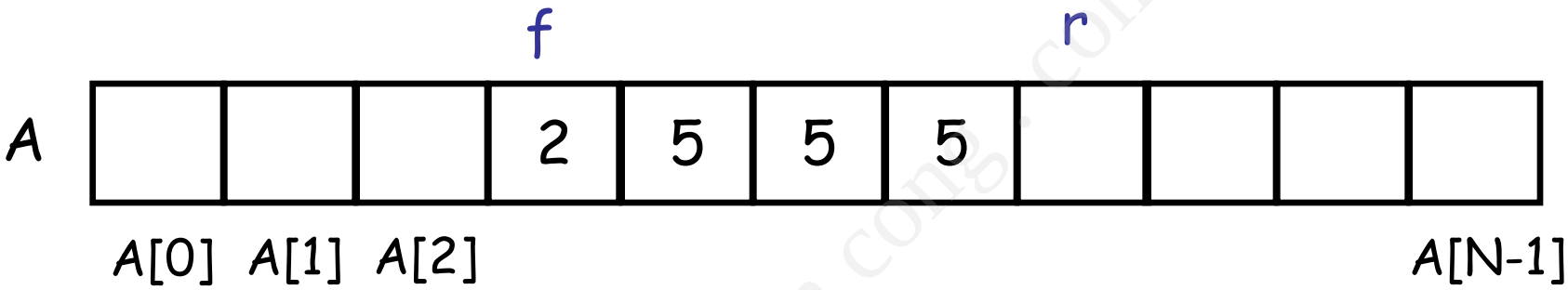
DeQue
EnQueue(5,Q)
EnQueue(5,Q)

Cài đặt dùng mảng



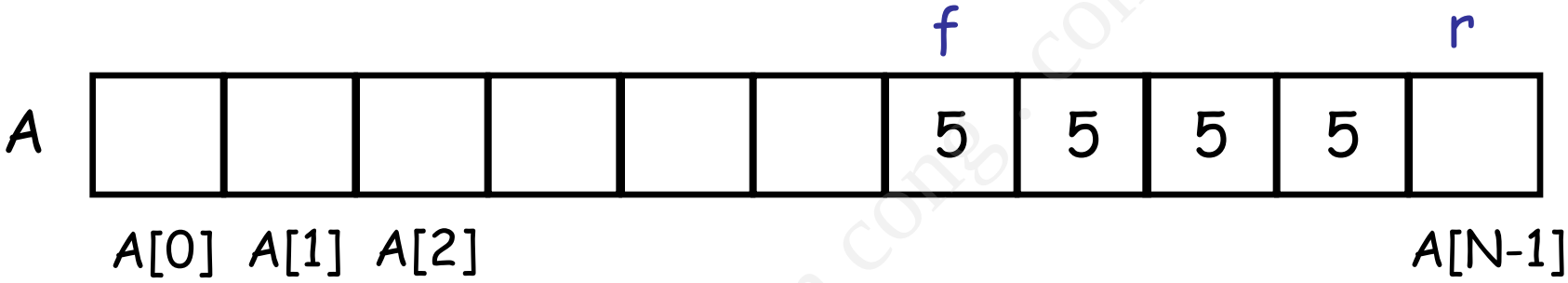
DeQue
EnQueue(5,Q)
EnQueue(5,Q)
DeQue
DeQue

Cài đặt dùng mảng



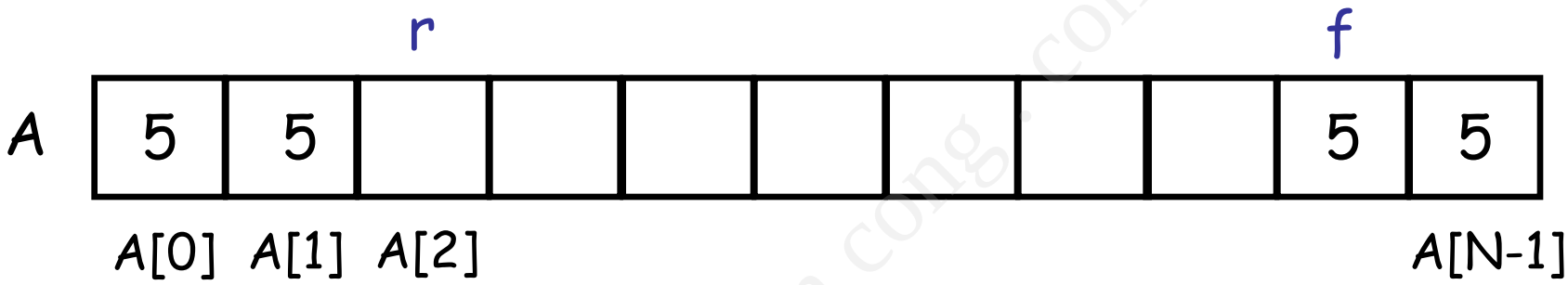
DeQue
EnQueue(5,Q)
EnQueue(5,Q)
DeQue
DeQue
DeQueue(Q), EnQueue(5,Q), DeQueue(Q),
EnQueue(5,Q),.....

Cài đặt dùng mảng



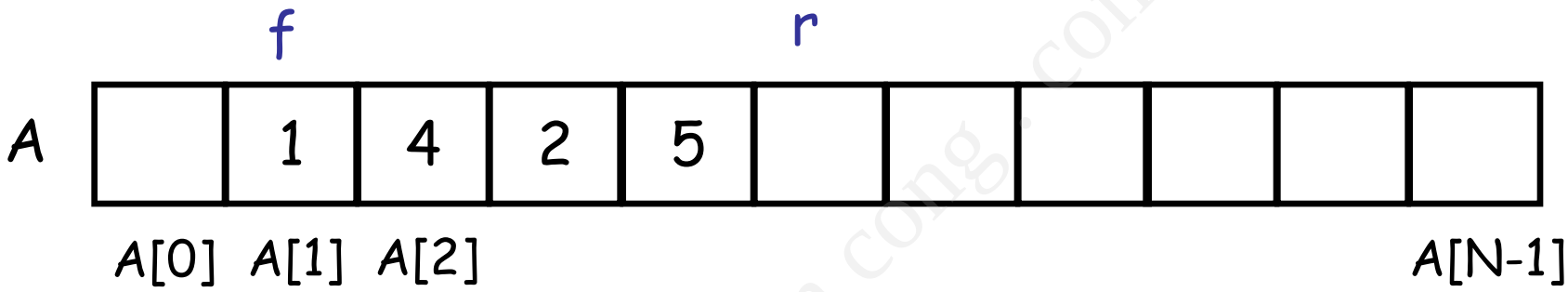
DeQue
EnQueue(5,Q)
EnQueue(5,Q)
DeQue
DeQue
DeQueue(Q), EnQueue(5,Q), DeQueue(Q), EnQueue(5,Q),.....

Dùng mảng vòng



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

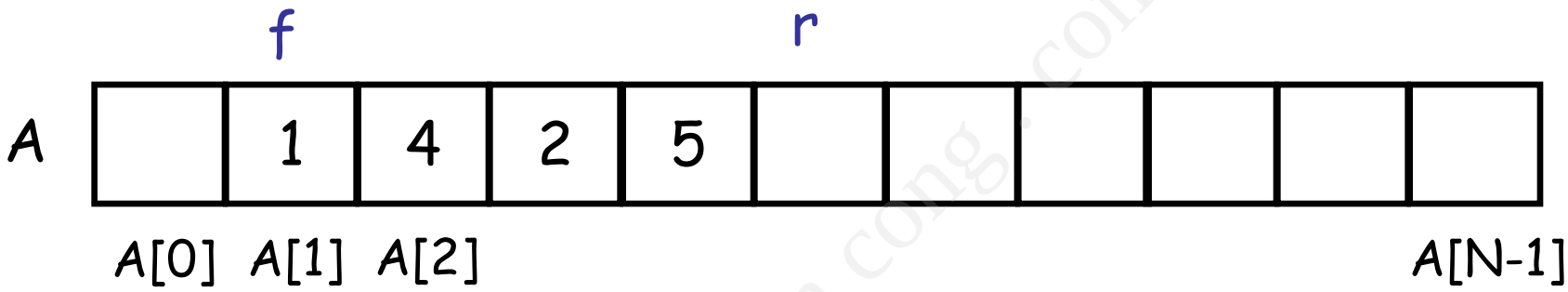
Thao tác



empty(Q): return (f = r)

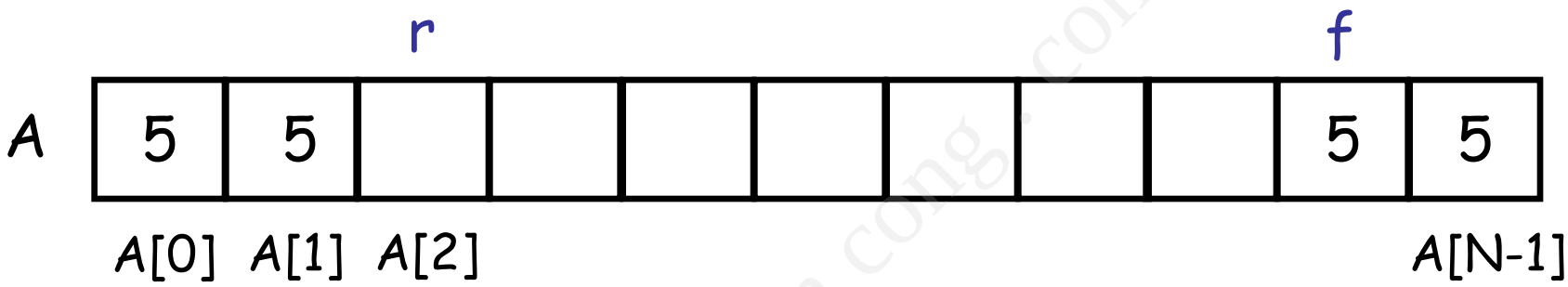
Front(Q): if empty(Q) then error
else return A[f]

Thao tác



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

Thao tác



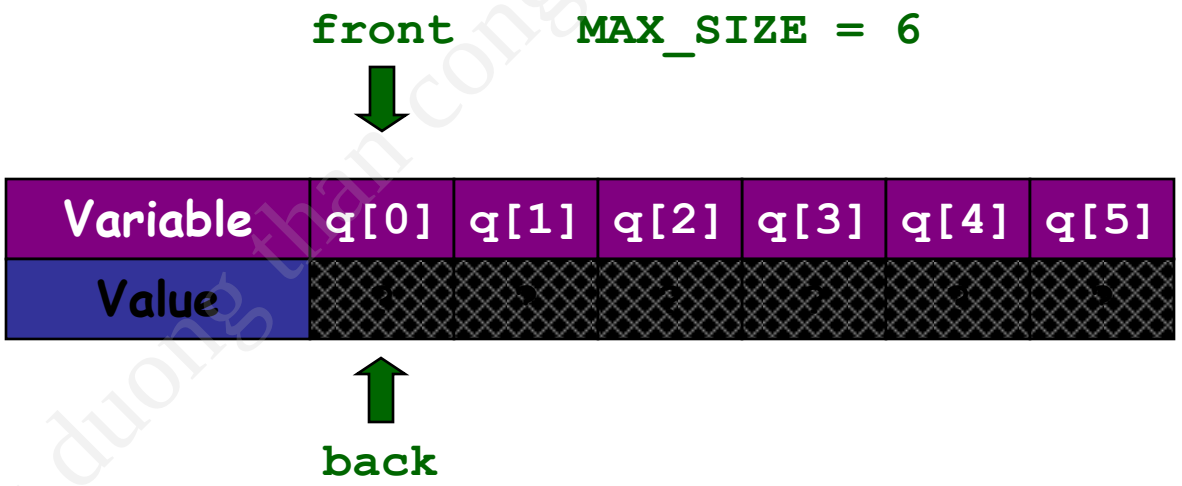
size(Q): if (r >= f) then return (r-f)
else return N-(f-r)

Queue Implementation with Arrays

- `QUEUEinit();`

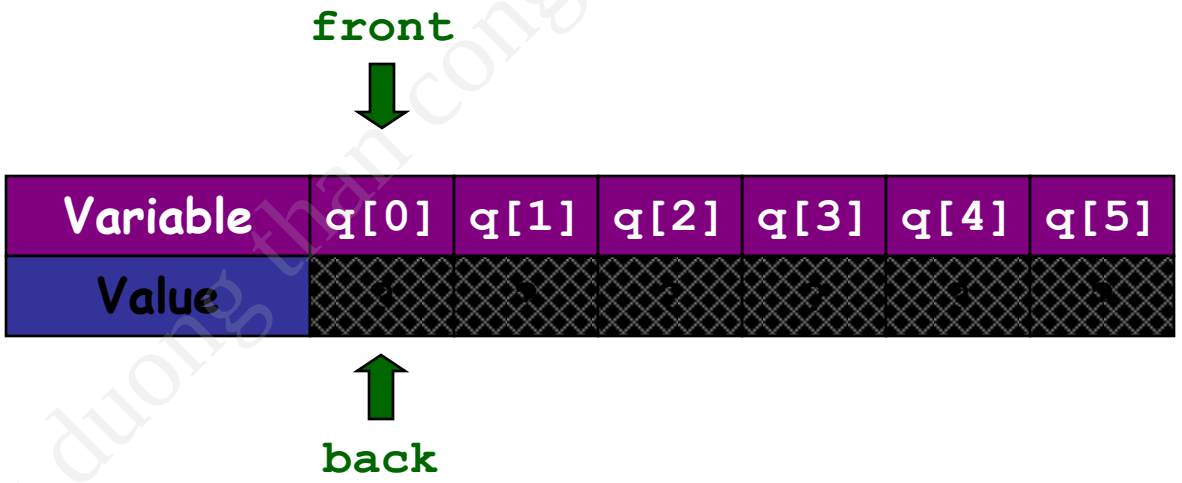
Queue Implementation with Arrays

- `QUEUEinit();`



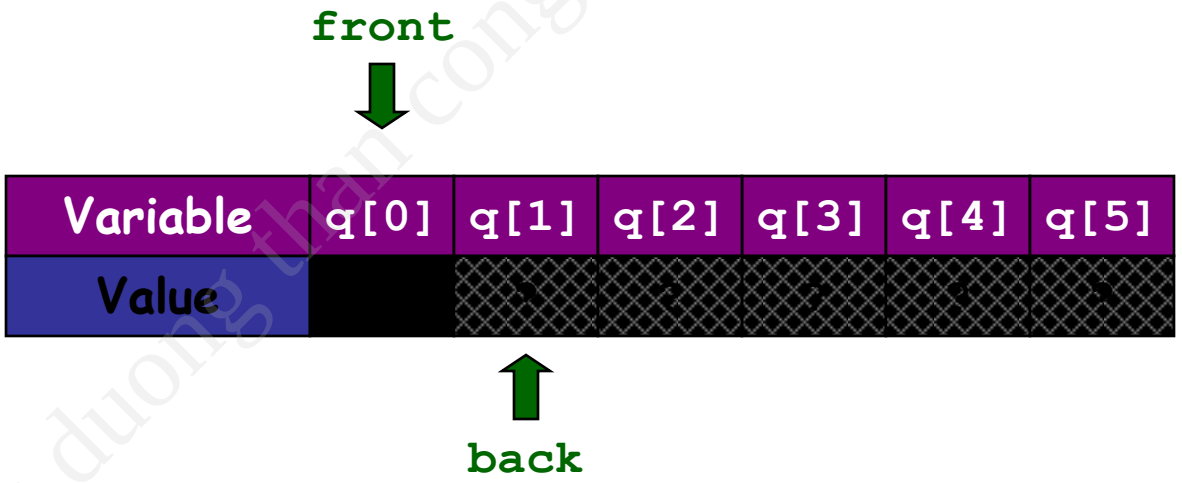
Queue Implementation with Arrays

- `QUEUEinit();`
- `QUEUEput('A');`



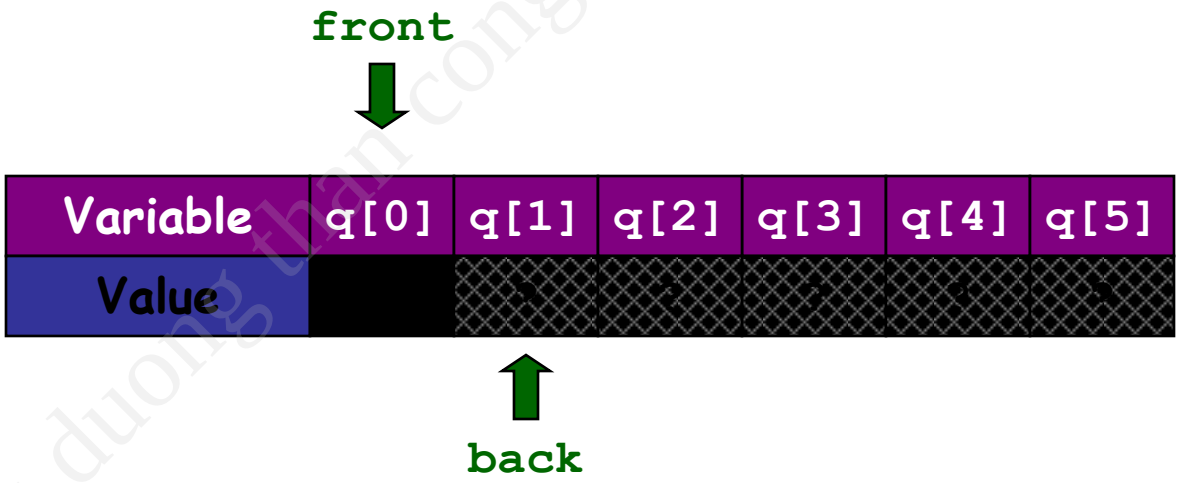
Queue Implementation with Arrays

- `QUEUEinit();`
- `QUEUEput('A');`



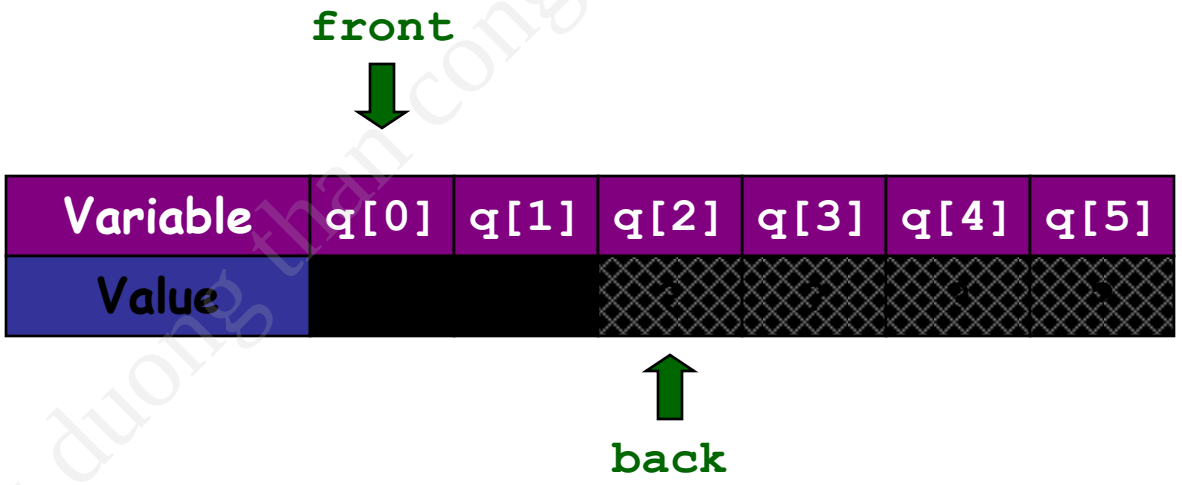
Queue Implementation with Arrays

- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`



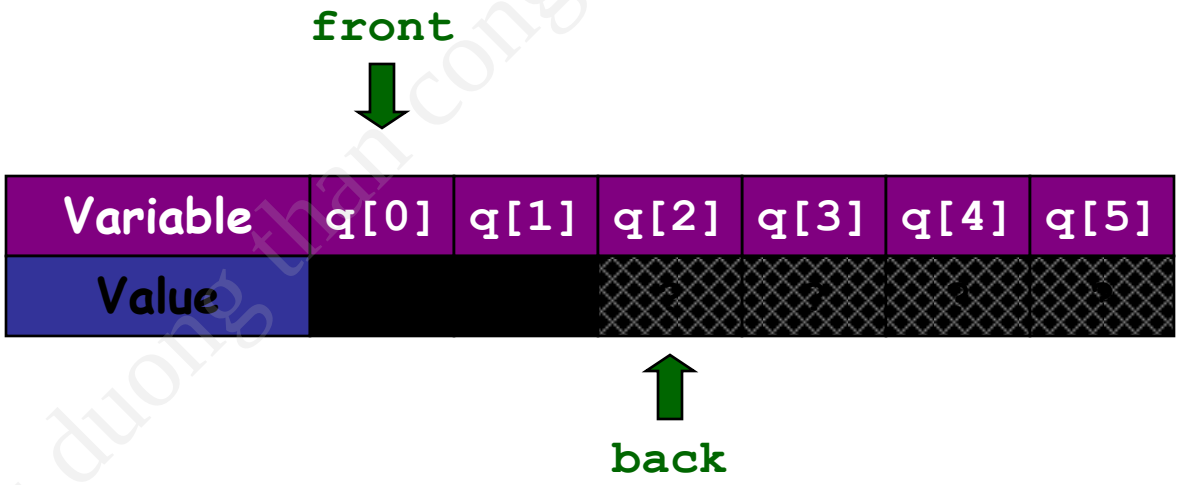
Queue Implementation with Arrays

- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`



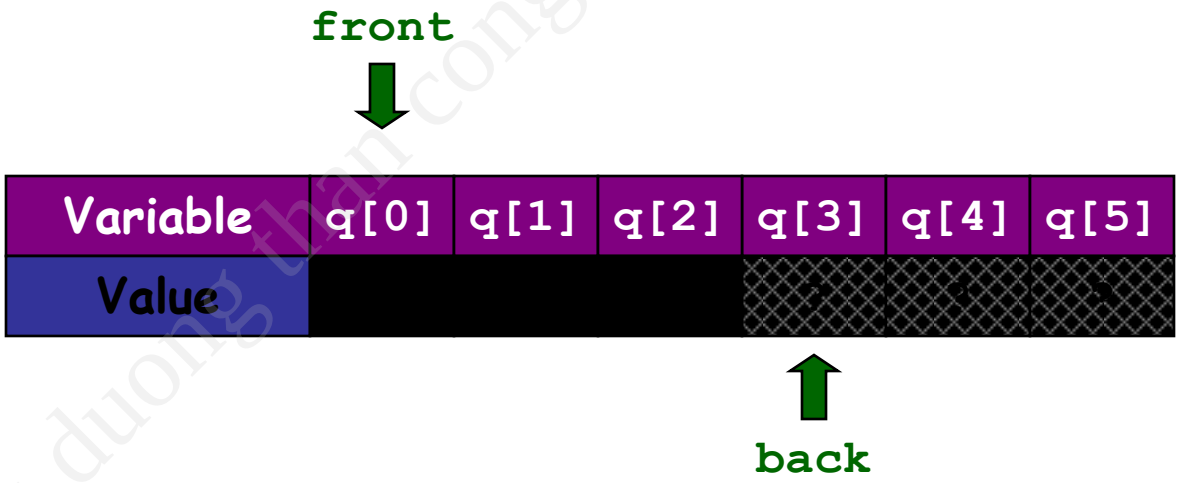
Queue Implementation with Arrays

- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`



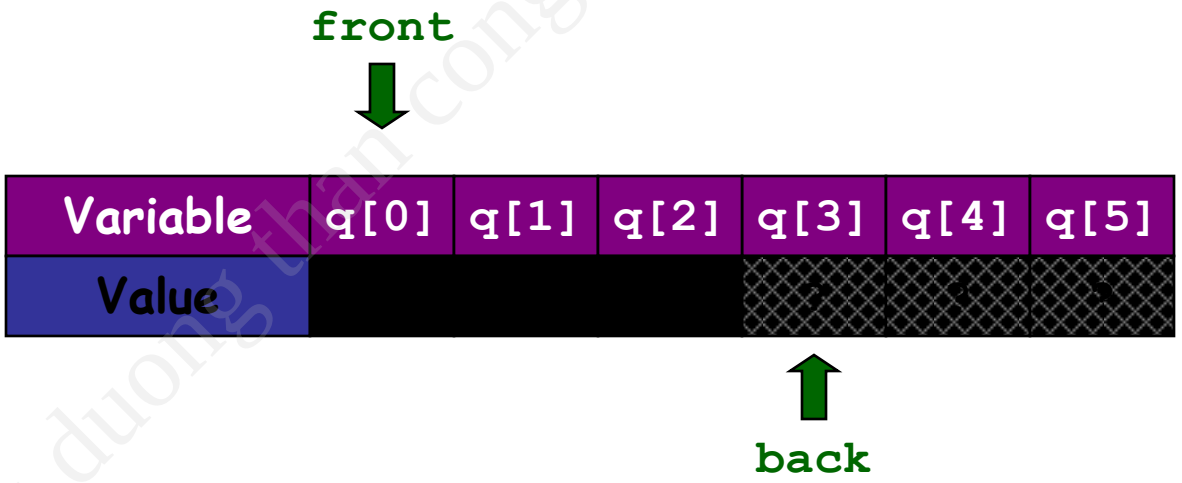
Queue Implementation with Arrays

- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`



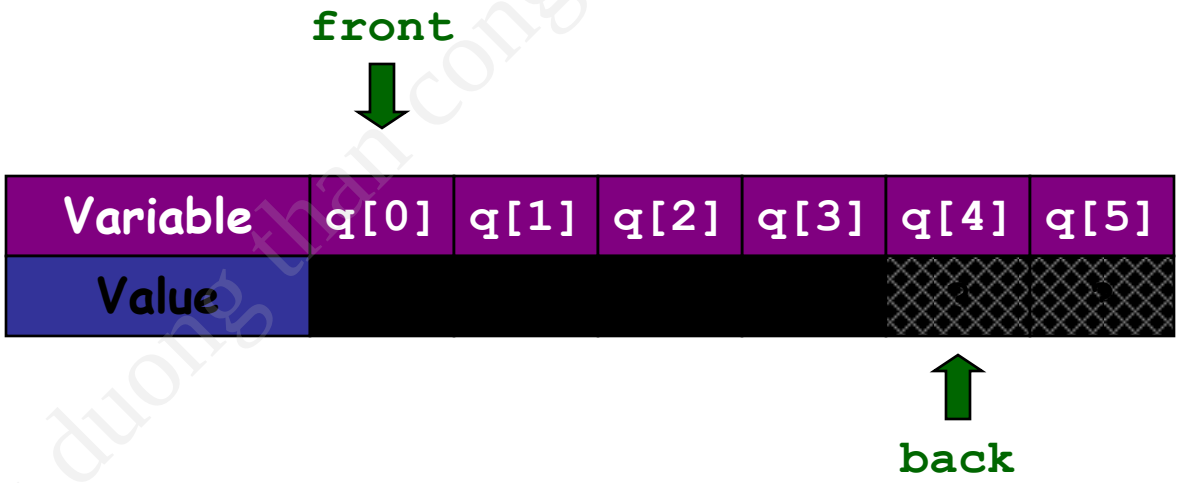
Queue Implementation with Arrays

- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`



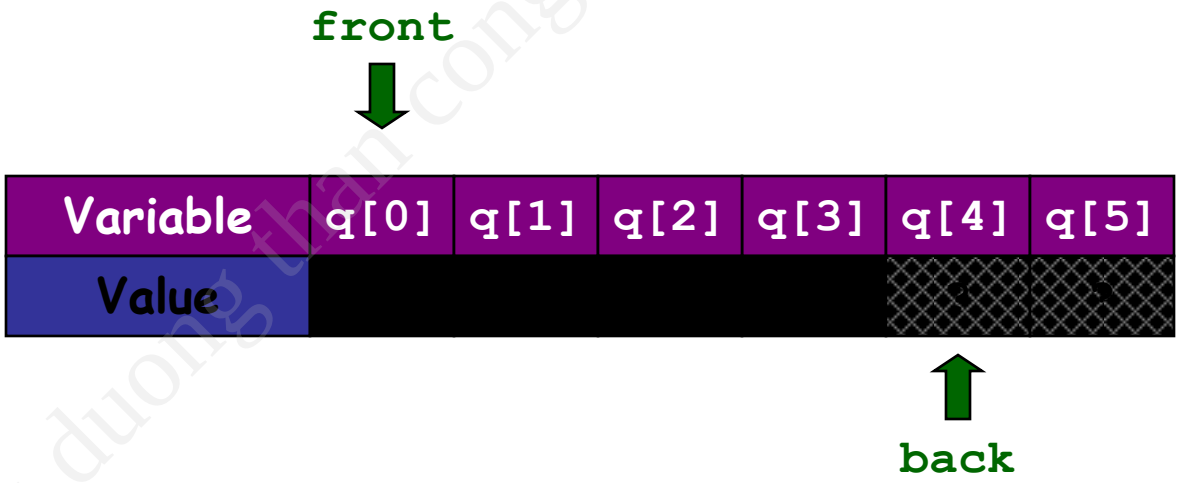
Queue Implementation with Arrays

- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`



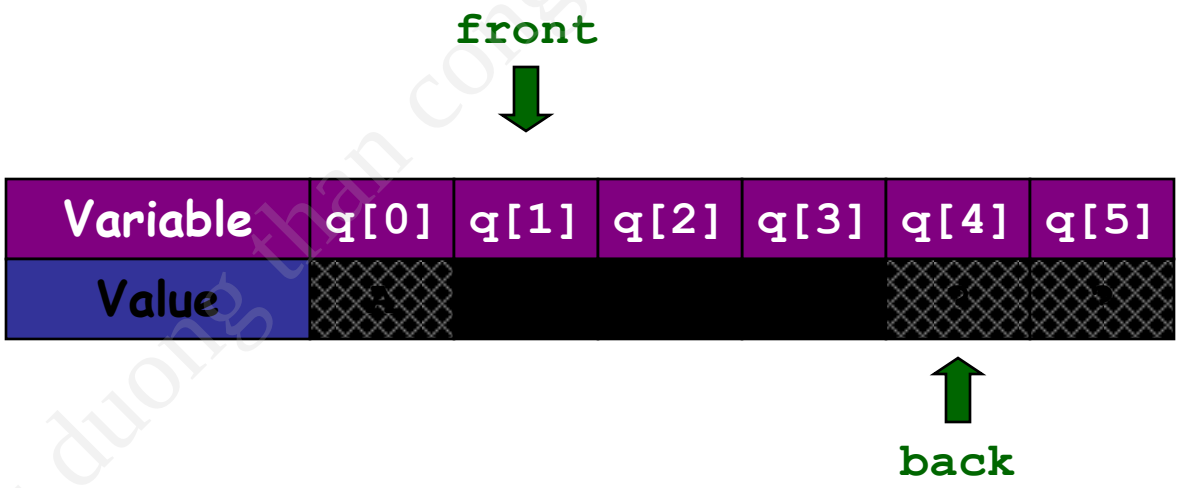
Queue Implementation with Arrays

- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`



Queue Implementation with Arrays

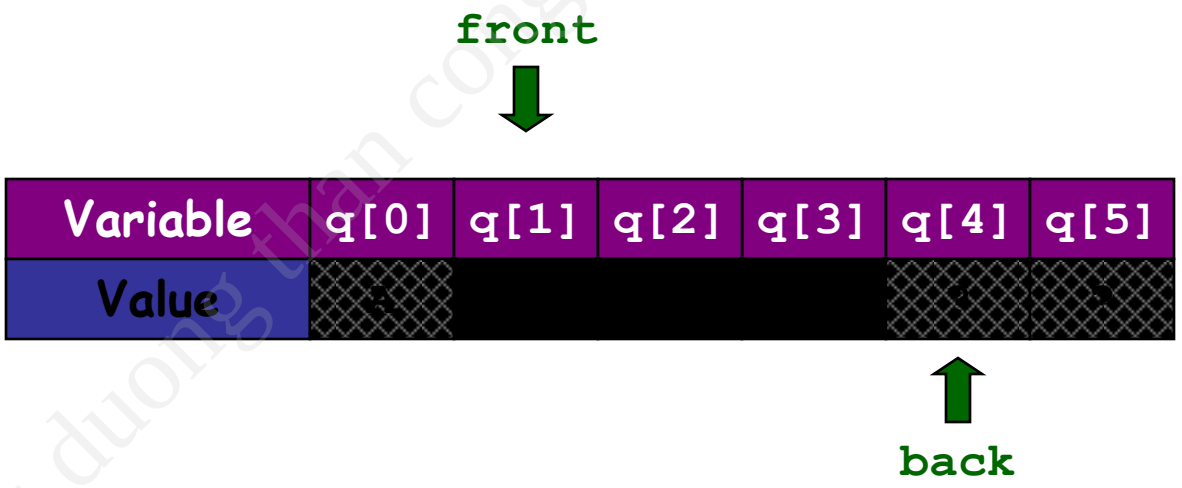
- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEpu`
  `t('E');`




```
Items dequeued: A
```

Queue Implementation with Arrays

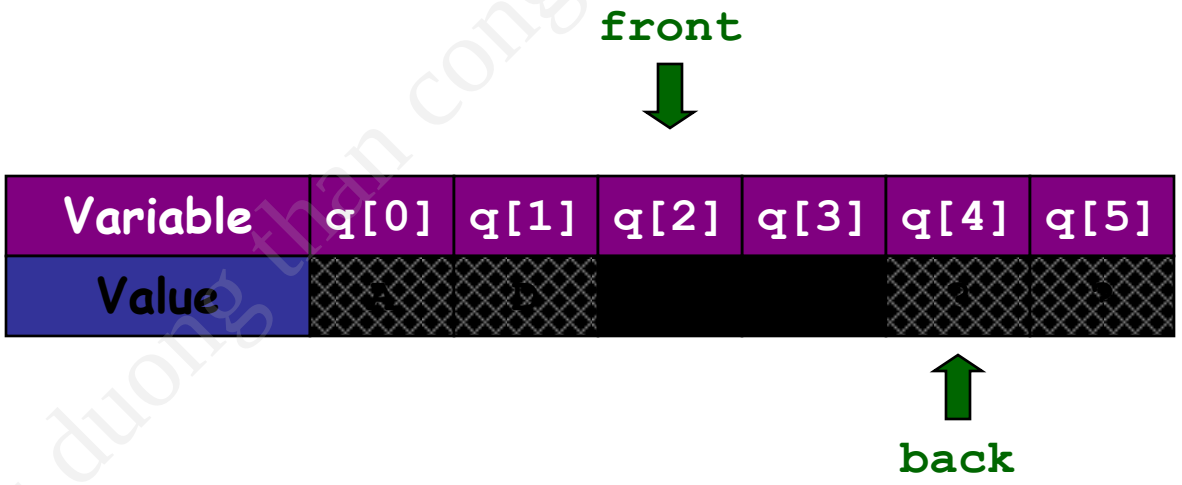
- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`




```
Items dequeued: A
```

Queue Implementation with Arrays

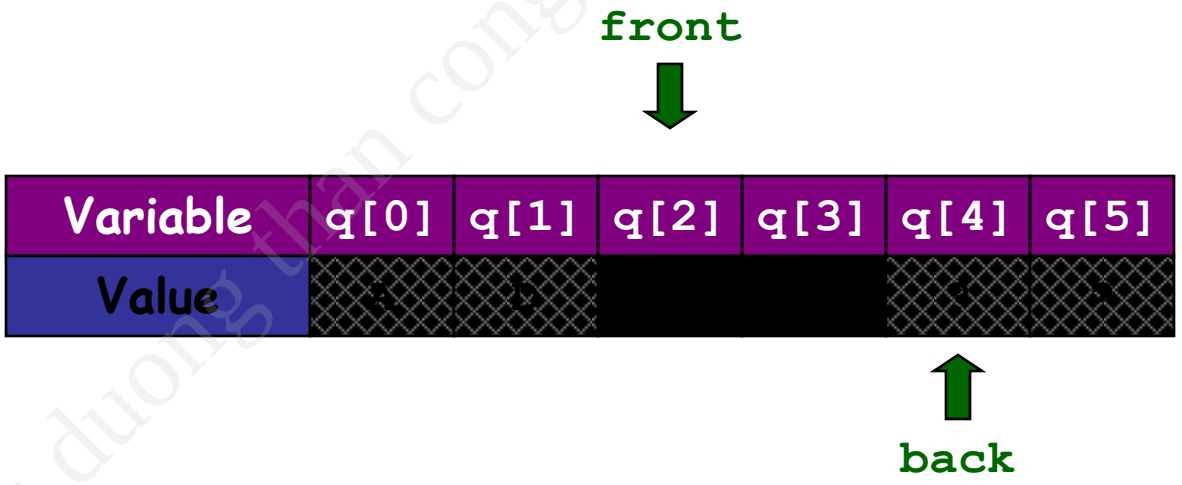
- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`




```
Items dequeued: A D
```

Queue Implementation with Arrays

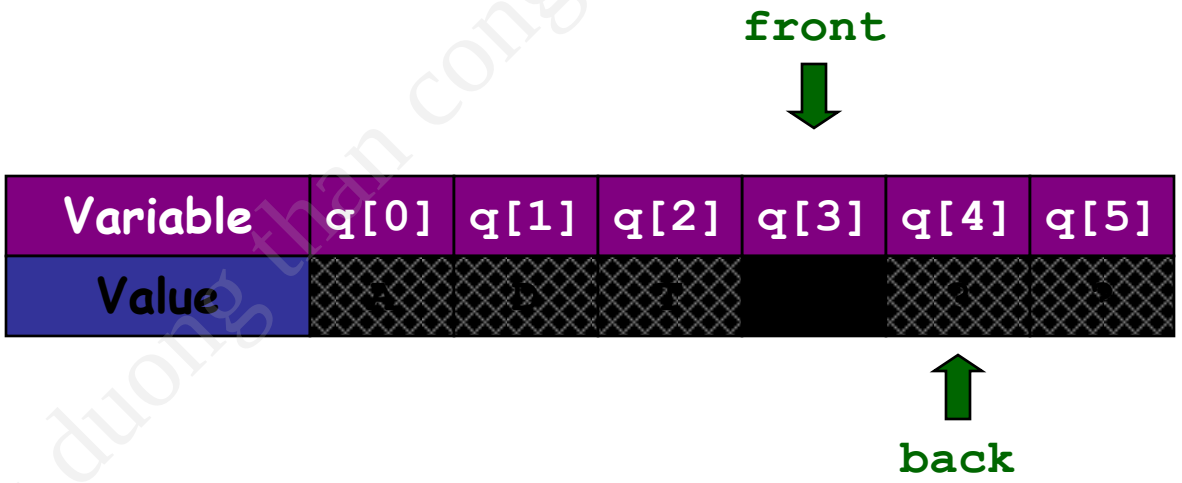
- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`




```
Items dequeued: A D
```

Queue Implementation with Arrays

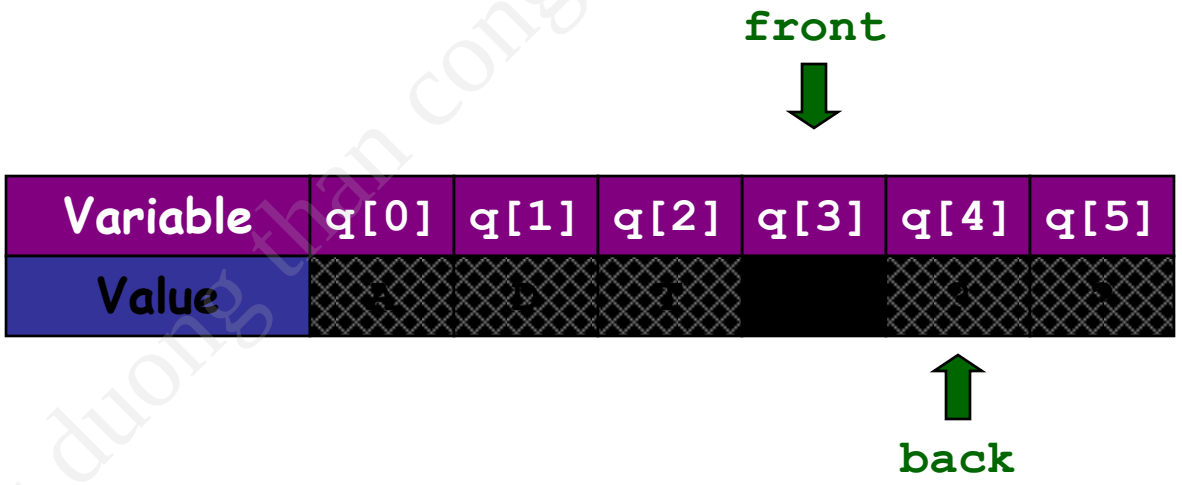
- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`




```
Items dequeued: A D T
```

Queue Implementation with Arrays

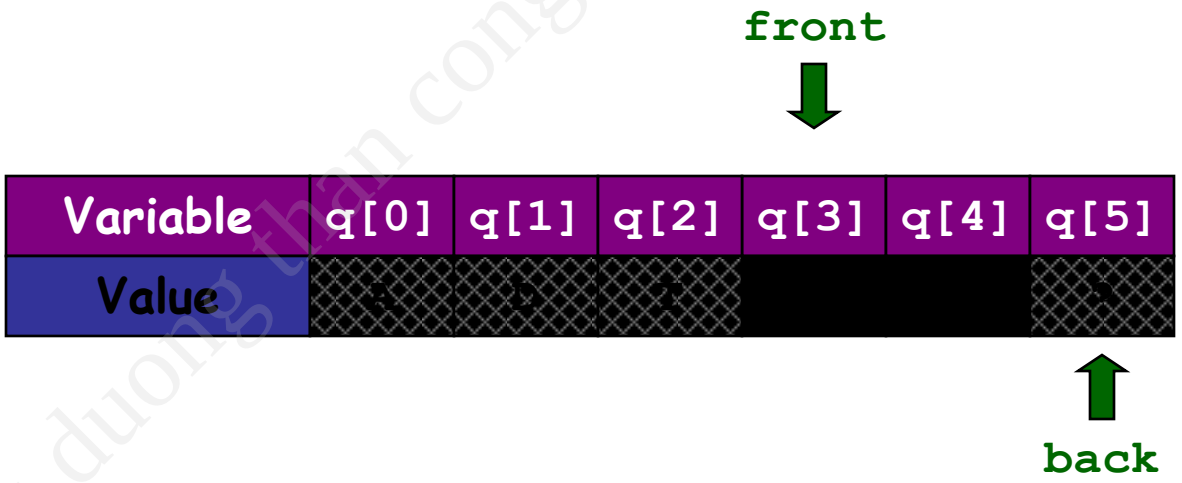
- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEpu`
  `t('E');`




```
Items dequeued: A D T
```

Queue Implementation with Arrays

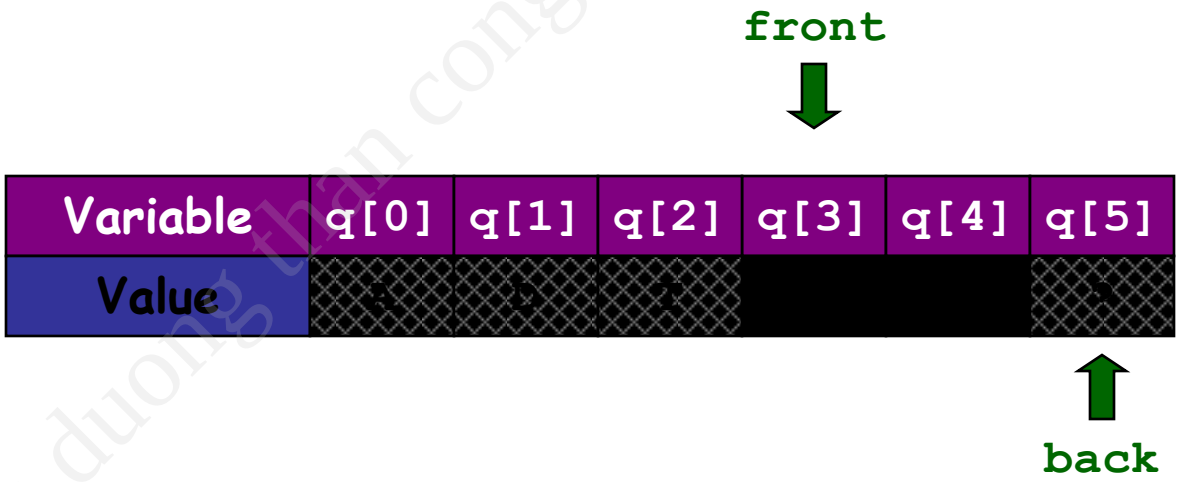
- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`




```
Items dequeued: A D T
```

Queue Implementation with Arrays

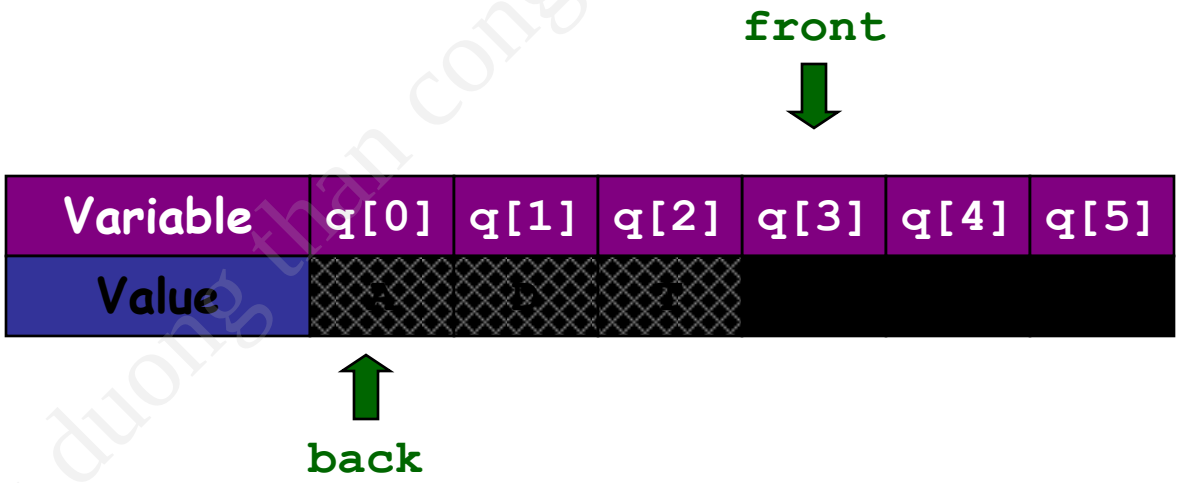
- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`




```
Items dequeued: A D T
```

Queue Implementation with Arrays

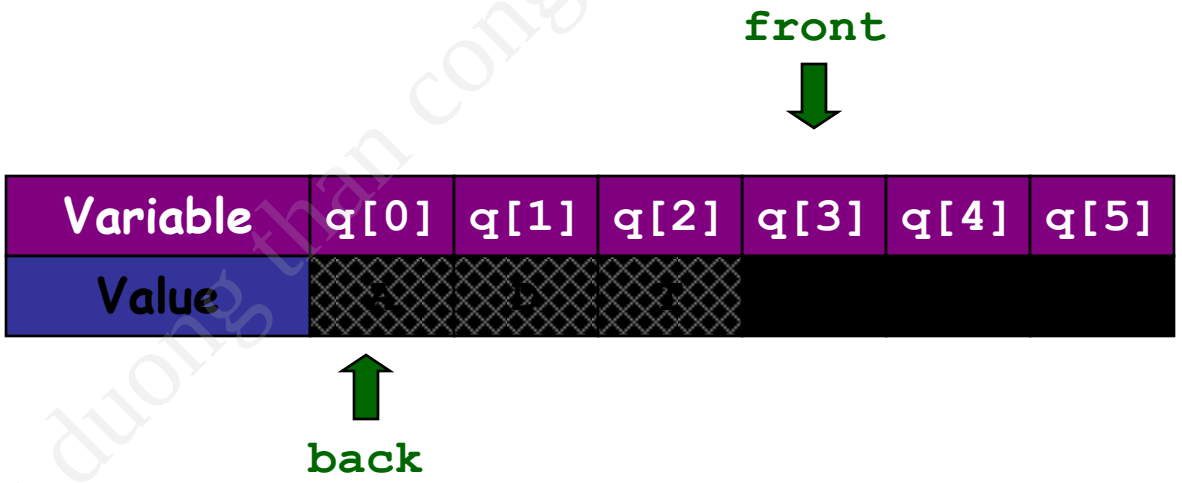
- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`




```
Items dequeued: A D T
```

Queue Implementation with Arrays

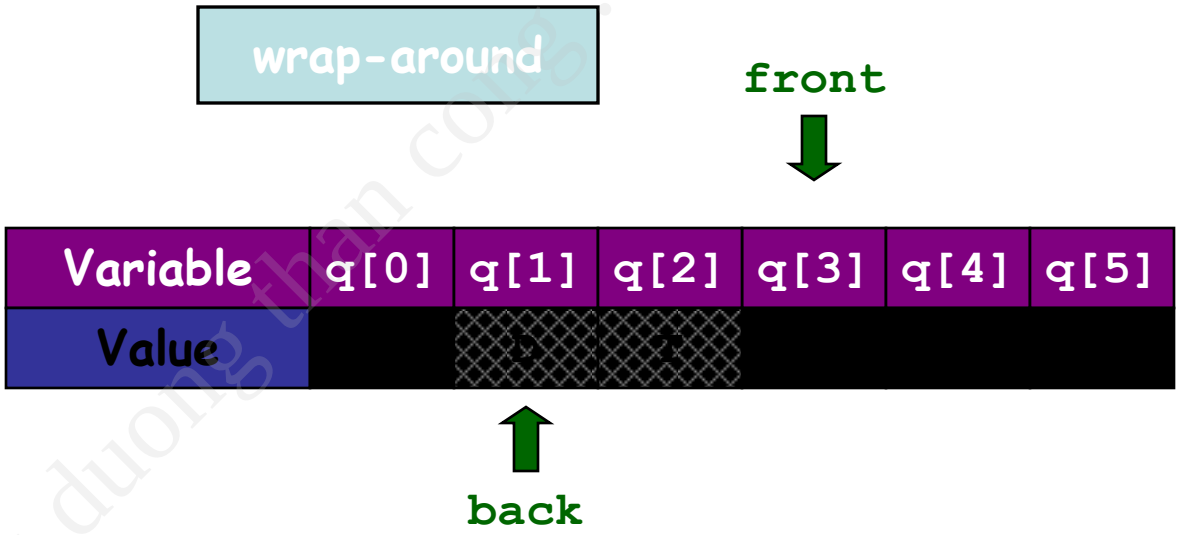
- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`




```
Items dequeued: A D T
```

Queue Implementation with Arrays

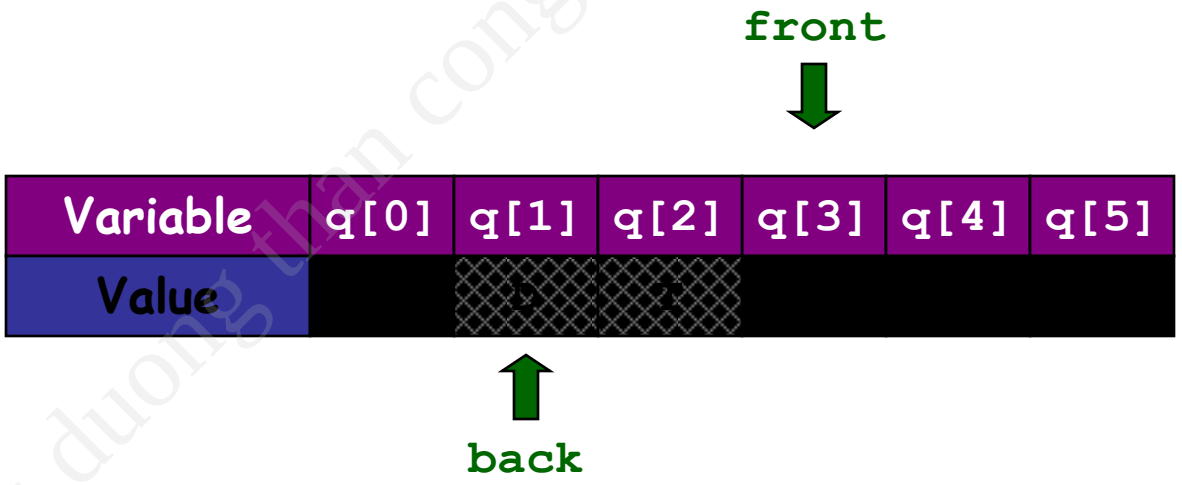
- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`




```
Items dequeued: A D T
```

Queue Implementation with Arrays

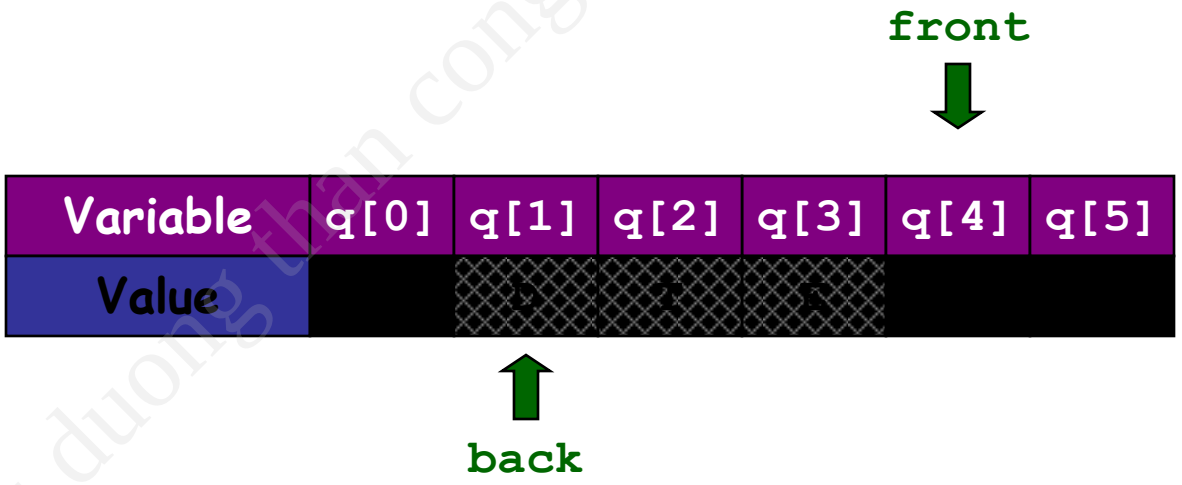
- QUEUEinit();
- QUEUEput('A');
- QUEUEput('D');
- QUEUEput('T');
- QUEUEput('E');




```
Items dequeued: A D T
```

Queue Implementation with Arrays

- `QUEUEinit();`
- `QUEUEput('A');`
- `QUEUEput('D');`
- `QUEUEput('T');`
- `QUEUEput('E');`




```
Items dequeued: A D T E
```

Biểu diễn Queue dùng mảng

Tạo hàng đợi rỗng

```
void   InitQueue()
{
   f = r = 0;
   for(int i = 0; i < N; i++)
      Q[i] = NULLDATA;
}
```

Biểu diễn Queue dùng mảng

Kiểm tra hàng đợi rỗng hay không

```
char  IsEmpty()
{
   return (Q[f] == NULLDATA);
}
```

Kiểm tra hàng đợi đầy hay không

```
char IsFull()
{
  return (Q[r] != NULLDATA);
}
```

Biểu diễn Queue dùng mảng

Thêm một phần tử x vào cuối hàng đợi Q

```
char      EnQueue(Data x)
{
   if(IsFull()) return -1; //Queue  đầy
   Q[r++] = x;
   if(r == N)     // xoay vòng
      r = 0;
}
```

Biểu diễn Queue dùng mảng

Trích, huỷ phần tử ở đầu hàng đợi Q

```
Data      DeQueue()
{  Data x;
   if(IsEmpty()) return NULLDATA; //Queue  rỗng
   x = Q[f]; Q[f++] = NULLDATA;
   if(f == N)        f = 0; // xoay vòng
   return x;
}
```

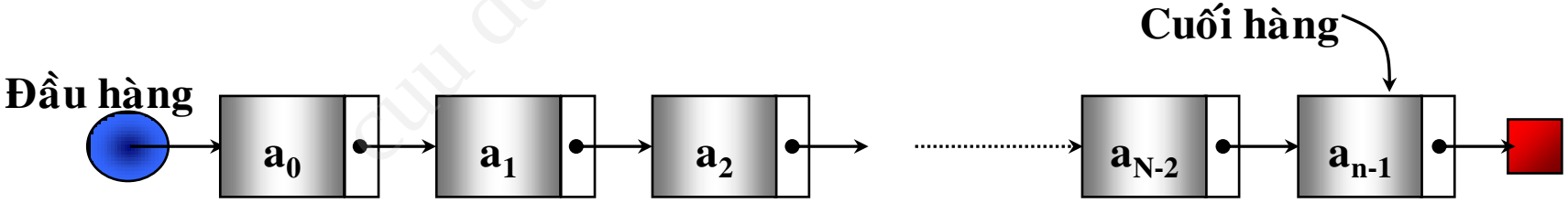
Xem thông tin của phần tử ở đầu hàng đợi Q

```
Data      Front()
{
   if(IsEmpty()) return NULLDATA; //Queue  rỗng
   return Q[f];
}
```

Biểu diễn hàng đợi dùng danh sách liên kết

Có thể tạo một hàng đợi sử dụng một DSLK đơn.

Phần tử đầu DSKL (head) sẽ là phần tử đầu hàng đợi, phần tử cuối DSKL (tail) sẽ là phần tử cuối hàng đợi.



Ứng dụng của hàng đợi

Hàng đợi 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 ⇒ Bộ đệm ⇒ 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, ...

Hàng đợi hai đầu (double-ended queue)

Hàng đợi hai đầu (gọi tắt là Deque) là một vật chứa các đối tượng mà việc thêm hoặc hủy một đối tượng được thực hiện ở cả 2 đầu của nó.

Hàng đợi hai đầu (double-ended queue)

Deque là một CTDL trừu tượng (ADT) hỗ trợ các thao tác chính sau:

- **InsertFirst**(e): Thêm đối tượng e vào đầu deque
- **InsertLast**(e): Thêm đối tượng e vào cuối deque
- **RemoveFirst**(): Lấy đối tượng ở đầu deque ra khỏi deque và trả về giá trị của nó.
- **RemoveLast**(): Lấy đối tượng ở cuối deque ra khỏi deque và trả về giá trị của nó.

Hàng đợi hai đầu (double-ended queue)

Ngoài ra, deque cũng hỗ trợ các thao tác sau:

- **IsEmpty**(): Kiểm tra xem deque có rỗng không.
- **First**(): Trả về giá trị của phần tử nằm ở đầu deque mà không hủy nó.
- **Last**(): Trả về giá trị của phần tử nằm ở cuối deque mà không hủy nó.

Dùng deque để cài đặt stack và queue

Dùng Deque để cài đặt Stack

STT	Stack	Deque
1	Push	InsertLast
2	Pop	RemoveLast
3	Top	Last
4	IsEmpty	IsEmpty

Dùng Deque để cài đặt Queue

STT	Queue	Deque
1	Enqueue	InsertLast
2	Dequeue	RemoveFist
3	Front	First
4	IsEmpty	IsEmpty

Cài đặt deque

Do đặc tính truy xuất hai đầu của deque, việc xây dựng CTDL biểu diễn nó phải phù hợp.

Có thể cài đặt CTDL deque bằng danh sách liên kết đơn. Tuy nhiên, khi đó thao tác RemoveLast hủy phần tử ở cuối deque sẽ tốn chi phí O(n). Điều này làm giảm hiệu quả của CTDL.

Thích hợp nhất để cài đặt deque là dùng danh sách liên kết kép. Tất cả các thao tác trên deque khi đó sẽ chỉ tốn chi phí O(1).

Danh sách liên kết có thứ tự (Odered List)

Trong hàng đợi có độ ưu tiên, mỗi phần tử được gán cho một độ ưu tiên.

Hàng đợi có độ ưu tiên cũng giống như hàng đợi bình thường ở thao tác lấy một phần tử khỏi hàng đợi (lấy ở đầu queue) nhưng khác ở thao tác thêm vào. Thay vì thêm vào ở cuối queue, việc thêm vào trong hàng đợi có độ ưu tiên phải bảo đảm phần tử có độ ưu tiên cao đứng trước, phần tử có độ ưu tiên thấp đứng sau.

Danh sách liên kết có thứ tự (Odered List)

Hàng đợi có độ ưu tiên có nhiều ứng dụng, ví dụ như dùng quản lý hàng đợi các tiến trình chờ được xử lý trong các hệ điều hành đa nhiệm.

Bài tập

- **Câu 1**

như sau:

A = { m, a, t, g, p, b, a, p, h, o, b, o, n, r, n, o, i, a, g, p, h, g, o, n}

-

ng

  - o ST2.
  - o ST2.

- o ST1
- o ST2.
- n:
  - o QUE
  - o QUE
- o ST1
- o QUE.
- o QUE
- o QUE.
- i

o ST1

- A = { m, a, t, g, p, b, a, p, h, o, b, o, n, r, n, o, i, a, g, p, h, g, o, n}
- ST1=[m,a,t,g,p];
- ST2=[];
- QUE=[];

o ST2

- A = { m, a, t, g, p, b, a, p, h, o, b, o, n, r, n, o, i, a, g, p, h, g, o, n}
- ST1=[m,a,t,g,p];
- ST2=[b,a,p,h,o,b,o,n,r];
- QUE=[];

o ST1

- A = { m, a, t, g, p, b, a, p, h, o, b, o, n, r, n, o, i, a, g, p, h, g, o, n}

- ST1=[m,a,t,g,p];
- ST2=[b,a,p,h,o,b,o,n,r];
- QUE=[];

o ST2

o ST1

o ST1

o ST2

n:

y **01**
y 01

a **ST1**
a **ST2**

o QUE
o **QUE**

o ST1

o QUE

o QUE

o QUE.

o QUE.

Đa thức

Biểu diễn bằng mảng

A[i] chứa hệ số của x^{i-1} :

$5 + 2x + 3x^2$	(5 2 3)
$7 + 8x$	(7 8)
$3 + x^2$	(3 0 2)

$4 + 3x^{2001}$

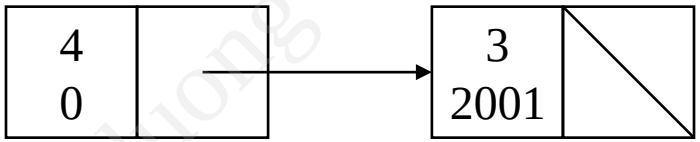
(**4** 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
…………………………………………………………………… . . 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 **3**)

Biểu diễn bằng danh sách liên kết

$4 + 3x^{2001}$

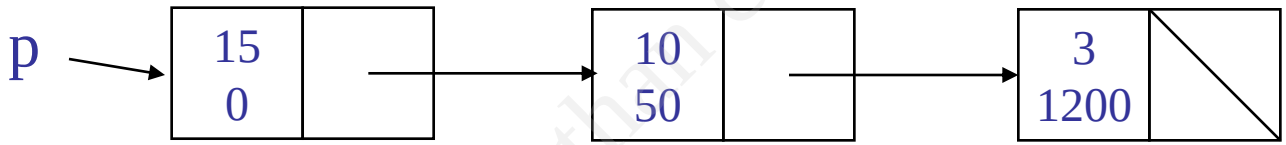
(<4 0> <2001 3>)



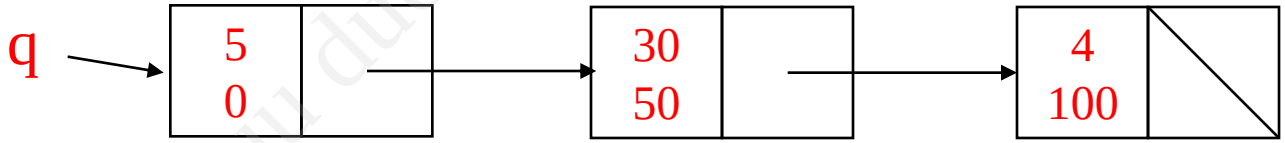
Phép cộng?

Complexity?

$15+10x^{50}+3x^{1200}$



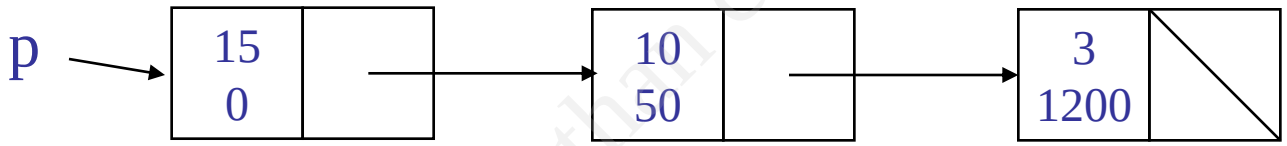
$5+30x^{50}+4x^{100}$



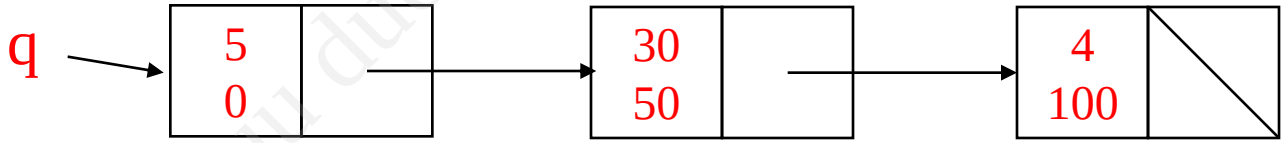
Cộng 2 đa thức

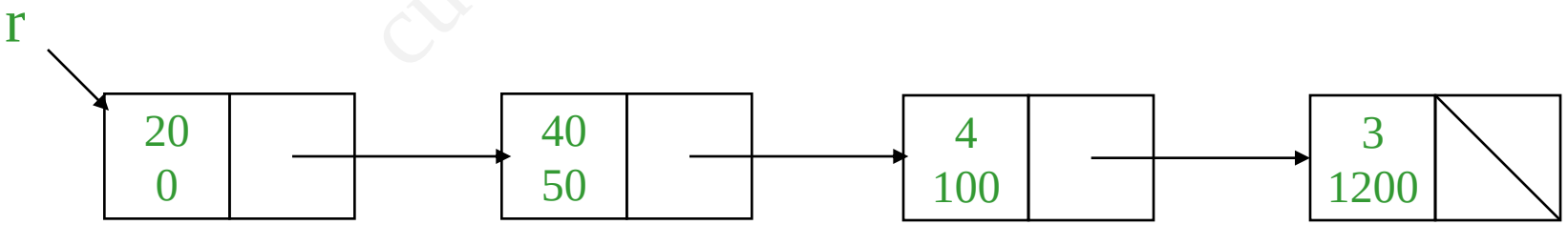
- Độ phức tạp: O(n+m)

$15+10x^{50}+3x^{1200}$



$5+30x^{50}+4x^{100}$





Ma trận

18	0	33	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	99	0	0
0	0	0	0	0	0
0	0	0	0	0	27