Cấu trúc dữ liệu và giải thuật

CÁC CẤU TRÚC DỮ LIỆU CƠ BẢN

Giảng viên:
Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến-
Vũ Thanh Hưng

Nội dung trình bày

- Danh sách liên kết
- Ngăn xếp
- Hàng đợi

Ngăn xếp - stack

Nội dung

- Giới thiệu
- Các thao tác cơ bản
- Ký pháp Ba Lan

Giới thiệu

- Một số hình ảnh thông dụng:

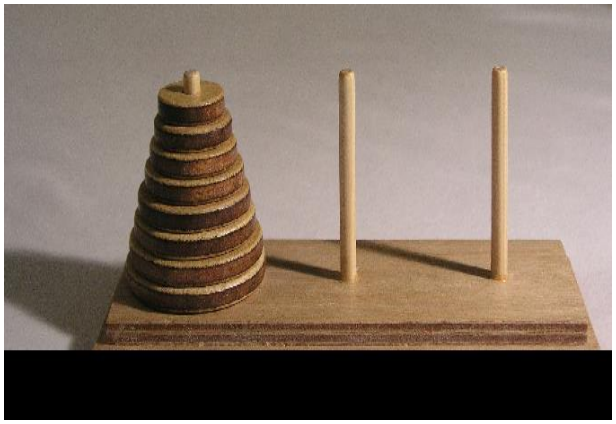
  - Một chồng sách vở ở trên bàn

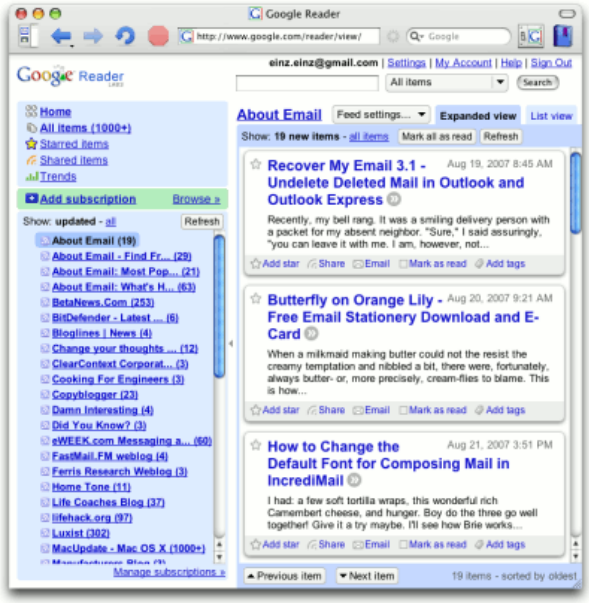
  - Một chồng đĩa
  - Các viên đạn trong băng đạn.

Nhận xét gì từ các tính chất của các tập hợp trên?

Tính chất vào sau ra trước

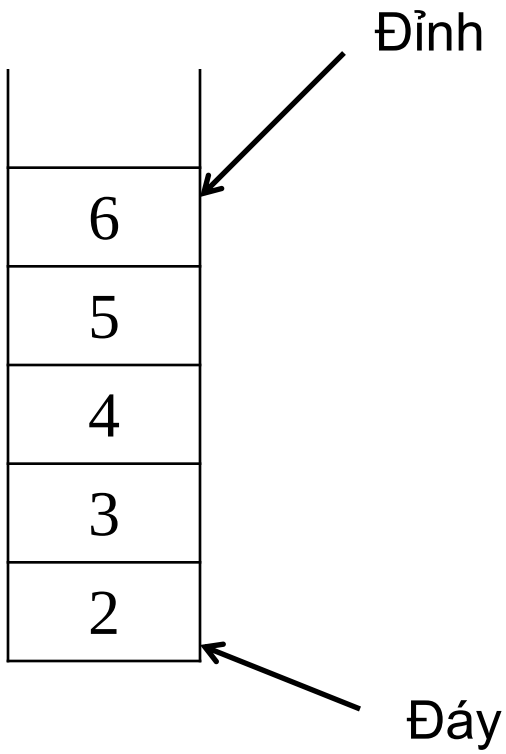
- Một tập hợp với có tính chất vào sau ra trước hữu ích trong một số bài toán:
  - Bài toán tháp Hà Nội
  - Chuyển biểu thức trung tố thành hậu tố
  - Ứng dụng đọc tin tức (Google Reader, facebook, vnexpress v.v…)





Giới thiệu

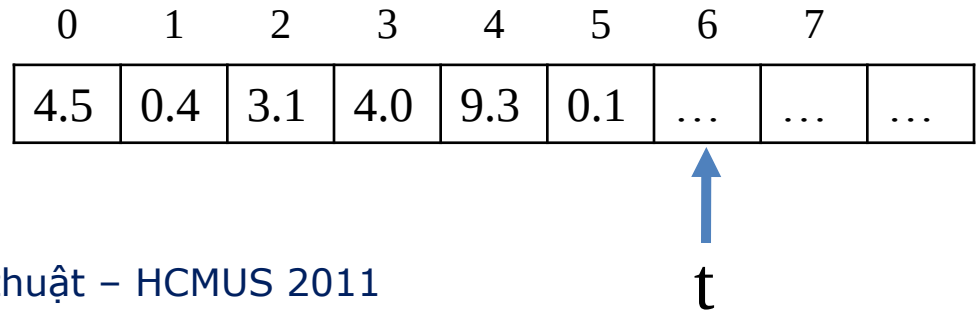
- Định nghĩa:
  - Ngăn xếp là **tập hợp** các đối tượng **làm việc theo cơ chế** "vào sau ra trước" (**L**ast **I**n **F**irst **O**ut)
  - Đối tượng có thể được thêm vào bất kì lúc nào, nhưng chỉ có ***đối tượng vào sau cùng mới được phép lấy ra khỏi ngăn xếp.***



Lưu trữ ngăn xếp

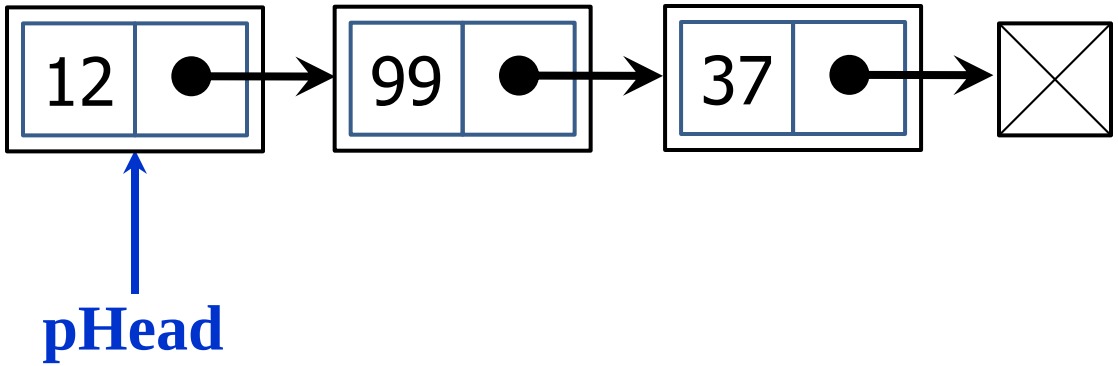
- Lưu trữ bằng mảng
  - Khai báo mảng 1 chiều với kích thước tối đa N.
  - t là địa chỉ của đỉnh của ngăn xếp → t sẽ thay đổi khi ngăn xếp hoạt động.
    - *Ngăn xếp rỗng thì giá trị của t là 0*
  - Tạo ngăn xếp S và quản lý ngăn xếp bằng biến t:

```
Data S[N];
int t;
```



Lưu trữ ngăn xếp

- Lưu trữ bằng DSLK:
  - Dùng con trỏ pHead lưu địa chỉ của đỉnh ngăn xếp
  - Ngăn xếp rỗng khi pHead = NULL

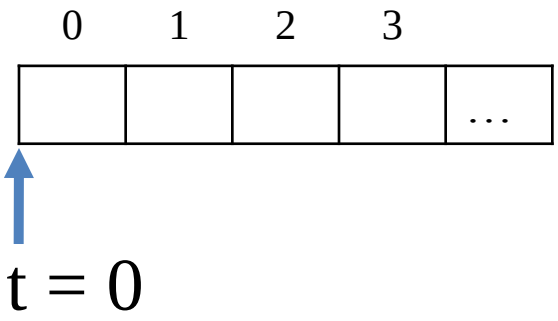


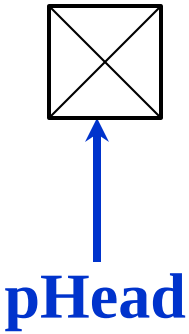
Các thao tác trên ngăn xếp

- Các thao tác cơ bản:
  - Push: Thêm 1 phần tử vào ngăn xếp
  - Pop: Lấy 1 phần tử ra khỏi ngăn xếp
- Các thao tác khác:
  - Kiểm tra ngăn xếp đầy: isfull()
  - Kiểm tra ngăn xếp rỗng : isempty()
  - Lấy thông tin của phần tử đầu ngăn xếp: peep()

Kiểm tra ngăn xếp rỗng: isempty()

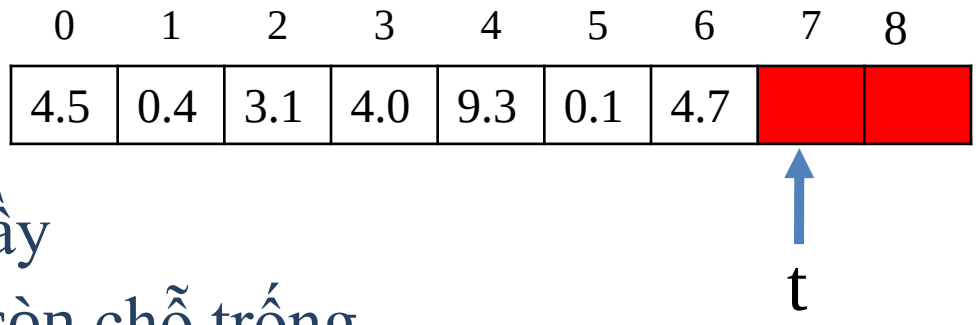
- Input:
- Output:
  - TRUE nếu ngăn xếp rỗng
  - FALSE nếu ngăn xếp không rỗng
- Ngăn xếp rỗng:
  - Mảng: số lượng phần tử mảng là 0
  - DSLK: pHead = NULL





Kiểm tra ngăn xếp đầy: isfull()

- Input:



- Output:
  - TRUE nếu ngăn xếp đầy
  - FALSE nếu ngăn xếp còn chỗ trống

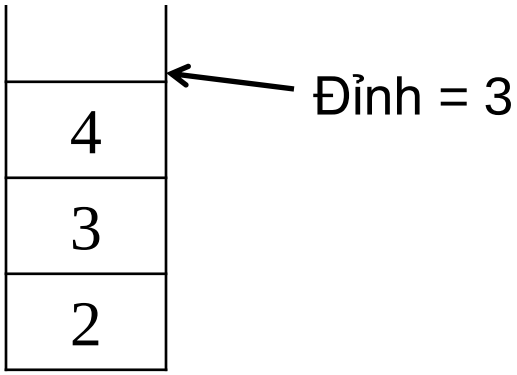
- Ngăn xếp đầy:
  - Mảng: đã lưu hết các phần tử mảng
  - DSLK: không cấp phát được vùng nhớ mới cho ngăn xếp

Thêm phần tử vào ngăn xếp: push()

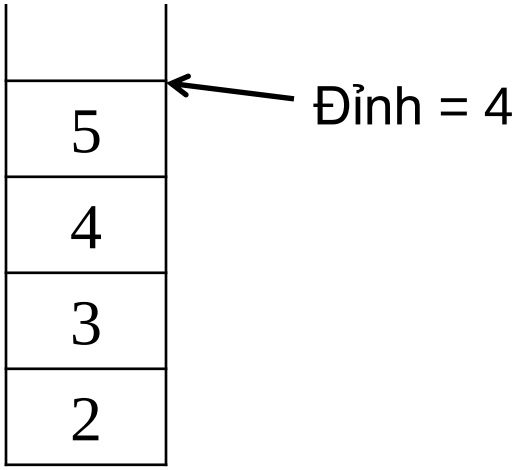
- Input: phần tử cần thêm
- Output:
- Giải thuật:
  - Kiểm tra ngăn xếp có đầy không?
  - Nếu không
    - Bổ sung phần tử mới vào
    - Cập nhật địa chỉ của con trỏ đến đỉnh ngăn xếp

Thêm phần tử vào ngăn xếp: push()

- Ví dụ:



Ngăn xếp ban đầu



Ngăn xếp sau khi thêm push(5)

Code

- Cho một ngăn xếp được lưu trữ dưới dạng mảng số nguyên a có tối đa N phần tử.
- Viết đoan code mô tả thao tác push()

```
int a[1000];
int v;
//Giả sử a = {5, 4, 7}
 //viết thao tác push tại đây.
//Sau khi push 8 và a = {5, 4, 7, 8}
 void push(int a[], int N, int t, int value)
 {
 …
 }
```

Lấy phần tử ra khỏi ngăn xếp: pop()

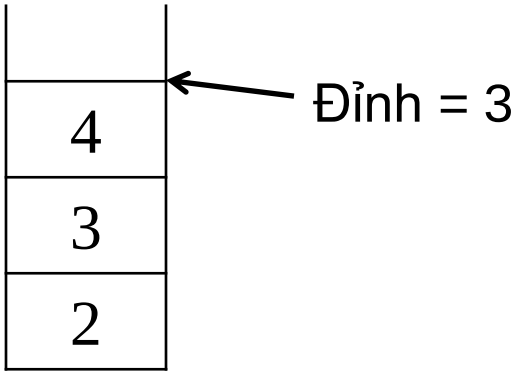
- Input:
- Output: giá trị của đối tượng đầu ngăn xếp

- Thuật toán:
  - Kiểm tra ngăn xếp rỗng hay không?
  - Nếu không:
    - Cập nhật địa chỉ của con trỏ đến đỉnh ngăn xếp

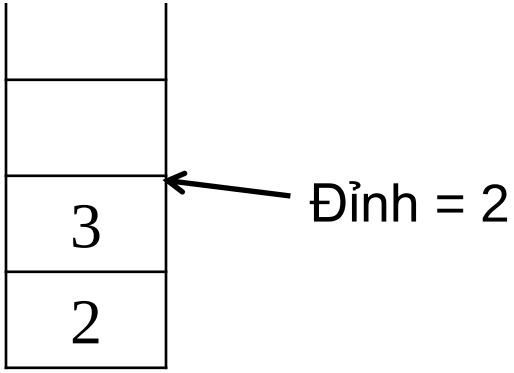
    → Xóa phần tử ở đỉnh khỏi ngăn xếp
    - Trả về giá trị của phần tử ở đỉnh

Lấy phần tử ra khỏi ngăn xếp: pop()

- Ví dụ:



Ngăn xếp ban đầu



Ngăn xếp sau khi pop()

return 4;

Code

- Cho một ngăn xếp được lưu trữ dưới dạng mảng số nguyên a có n phần tử.
- Viết đoan code mô tả thao tác pop() trả về giá trị và lưu trong biến v

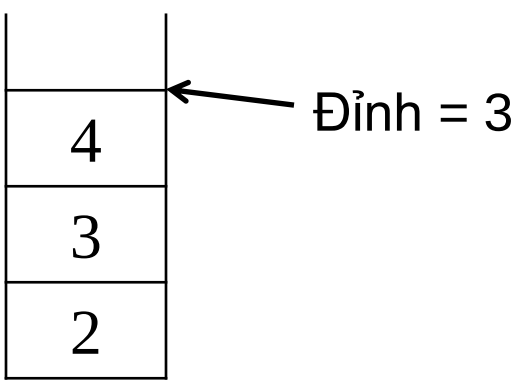
```
int a[1000];
int v;
//Giả sử a = {5, 4, 7}
//viết thao tác pop tại đây
//Sau khi pop, v = 7 và a = {5, 4}
int pop(int a[], int &t)
{
}
```

Lấy thông tin đỉnh ngăn xếp: peep()

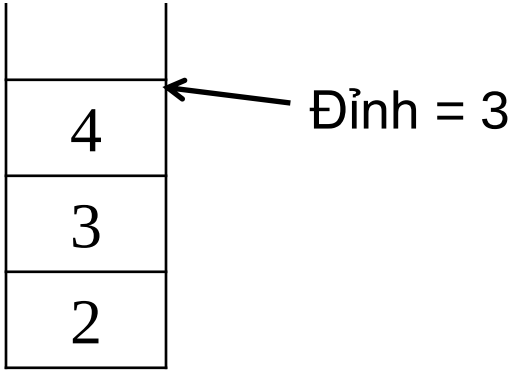
- Chỉ lấy giá trị của phần tử đầu mà *không hủy nó* khỏi ngăn xếp.
- Input:
- Output: giá trị tại đỉnh ngăn xếp
- Giải thuật:
  - Kiểm tra xem ngăn xếp có rỗng không?
  - Trả về giá trị của phần tử ở đỉnh ngăn xếp

Lấy thông tin đỉnh ngăn xếp: peep()

- Ví dụ



Ngăn xếp ban đầu



Ngăn xếp sau khi gettop()

return 4;

Code

- Cho một ngăn xếp được lưu trữ dưới dạng mảng số nguyên a có n phần tử.
- Viết đoạn code mô tả thao tác peep() trả về giá trị và lưu trong biến v

```
int a[1000];
int v;
//Giả sử a = {5, 4, 7}
//viết thao tác peep tại đây
//Sau khi peep, v = 7 và a = {5, 4, 7}
int peep(int a[], int t)
{
}
```

Bài tập

- Cho biết nội dung của stack sau khi thực hiện các thao tác trong dãy (từ trái sang phải):

  EAS*Y**QUE***ST***I*ON

  Mỗi chữ cái hoặc dấu * tương ứng một thao tác trên stack trong đó:

  - Một chữ cái tượng trưng cho thao tác thêm chữ cái đó vào stack
  - Dấu * tượng trưng cho thao tác lấy nội dung một phần tử trong stack ra rồi in lên màn hình.

  Cho biết kết quả xuất ra màn hình sau khi hoàn tất chuỗi trên?

Bài tập

- Cho biết nội dung của stack và giá trị của các biến sau khi thực hiện liên tiếp các thao tác sau:
- (Ban đầu các biến được khởi tạo: A= 5, B = 3, C= 7)

a. Tạo stack

b. push A

c. push C*C

d. pop rồi lưu trữ vào biến B

e. push B+A

f. pop rồi lưu trữ vào biến A

g. pop rồi lưu trữ vào biến B

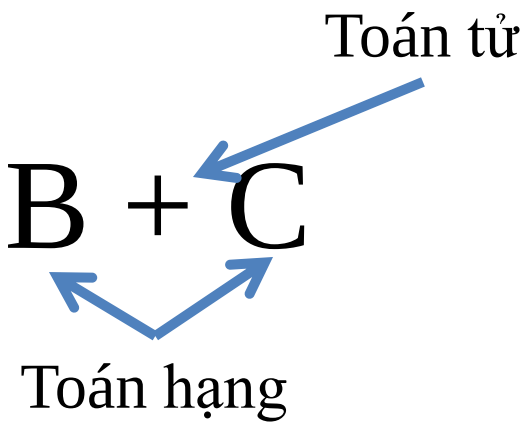
Ký pháp Ba Lan

- Biểu thức dạng *trung tố*: dấu của các phép toán hai ngôi luôn được *đặt giữa* 2 toán hạng
  - Ví dụ: A + B * C A + B * C - D

    (A+B) * C (A + B)* (C – D)

⇒ Qui định thứ tự ưu tiên của các phép toán

⇒ Dùng dấu ngoặc để phân biệt thứ tự thực hiện.



Ký pháp Ba Lan

- Biểu thức dạng *tiền tố*:

Trung tố	Tiền tố
A + B	+ A B
(A+B)*C	*+ A B C
(A + B)* (C – D)	* + A B – C D

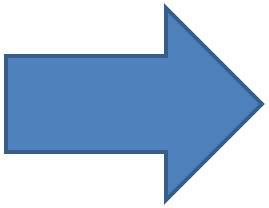
- Biểu thức dạng *hậu tố*:

Trung tố	Hậu tố
A + B	A B +
(A+B)*C	A B + C *
(A + B)* (C – D)	A B + C D - *

Không cần thiết phải dùng dấu ngoặc

Chuyển trung tố → hậu tố

- 3 + 4
- 3 * 4 + 2
- 4 + 3 * 2
- (5 + 3) * 2
- 4 + (7 – 4) * 5



- 3 4 +
- 3 4 * 2 +
- 4 3 2 * +
- 5 3 + 2 *
- 4 7 4 –5 * +

Nguyên tắc tính toán biểu thức hậu tố

- 3 4 + → 7
- 3 4 * 2 + → 14
- 4 3 2 * + → 10
- 5 3 2 * + →11
- 4 7 4 –5 * + →19

Tại sao cần chuyển sang dạng hậu tố

- So với dạng trung tố, dạng hậu tố có ưu điểm
  - Đơn giản: không cần dùng dấu ngoặc
  - Dễ tính toán

  → Thích hợp cho máy tính

Thuật toán chuyển trung tố → hậu tố

- Nguyên tắc:
- Duyệt qua các token
  - Dấu (: push vào ngăn xếp
  - Toán hạng: Xuất ra Q
  - Toán tử:
    - Nếu độ ưu tiên (token) < toán tử đỉnh S
      - Pop(S) và xuất ra Q
      - Push(S, token)
    - Ngược lại
      - Push(S, token)
  - Dấu): pop tất cả ra khỏi stack cho tới khi gặp dấu (

Chuyển trung tố sang hậu tố

- Mã giả: P là biểu thức trung tố ban đầu, Q là biểu thức kết quả dạng hậu tố

```
0.1 push('(');
0.2 Thêm ')' vào P
while (chưa hết biểu thức P)
{
   1. đọc 1 kí tự x trong P (từ trái qua phải)
   2. if (x là toán hạng)              //TOÁN HẠNG
                Thêm x vào Q
   3. if (x là dấu ngoặc mở)           //DẤU (
                push(x);
```

Chuyển trung tố sang hậu tố

- Mã giả:

```
4. if (x là toán tử)                    //TOÁN TỬ
        4.1 while( thứ tự ưu tiên tại đỉnh >= x)
            4.1.1 w = pop();
            4.1.2 Thêm w vào Q
        4.2 push(x);
5. if (x là dấu ngoặc đóng)   //DẤU )
        5.1 while( chưa gặp ngoặc mở)
            4.1.1 w = pop();
            4.1.2 Thêm w vào Q
        5.2 pop();//đẩy ngoặc mở ra khỏi ngăn xếp
}
```

Chuyển trung tố sang hậu tố

- Ví dụ 1: P = (A + B) * (C - (D + A))

Chuyển trung tố sang hậu tố


```
0.1 push('(');
0.2 Thêm ')' vào P
while (chưa hết biểu thức P)
{
   1. đọc 1 kí tự x trong P (từ trái qua phải)
   2. if (x là toán hạng)                    //TOÁN HẠNG
                Thêm x vào Q
   3. if (x là dấu ngoặc mở)                 //DẤU (
                push(x);
   4. if (x là toán tử)                      //TOÁN TỬ
                4.1 while( thứ tự ưu tiên tại đỉnh >= x)
                        4.1.1 w = pop();
                        4.1.2 Thêm w vào Q
                4.2 push(x);
   5. if (x là dấu ngoặc đóng)               //DẤU )
                5.1 while( chưa gặp ngoặc mở)
                        4.1.1 w = pop();
                        4.1.2 Thêm w vào Q
                5.2 pop();//đẩy ngoặc mở ra khỏi ngăn xếp
}
```

Chuyển trung tố sang hậu tố

- Ví dụ 2: đổi biểu thức trung tố

  P = A + (B * C – (D / E ^ F) * G) * H

  sang biểu thức dạng hậu tố

Ứng dụng khác của ngăn xếp

- Lượng giá biểu thức hậu tố
- Trong trình biên dịch, ngăn xếp được sử dụng để lưu môi trường các thủ tục.
- Dùng trong một số bài toán của lý thuyết đồ thị.
- Dùng biến đổi cơ số
- Khử đệ qui đuôi.

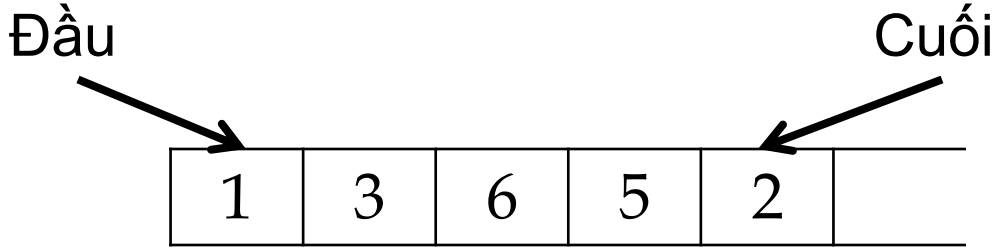
Hàng đợi - Queue

Nội dung

- Giới thiệu
- Các thao tác cơ bản
- Ứng dụng

Giới thiệu

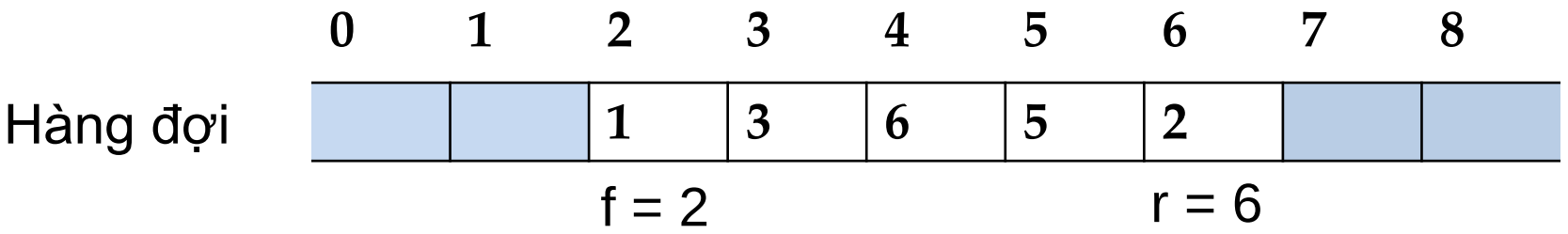
- Giới thiệu:
  - Là tập hợp các đối tượng làm việc theo qui tắc *vào trước ra trước* (FIFO).
  - Các đối tượng có thể được thêm vào hàng đợi bất kì lúc nào nhưng chỉ có đối tượng *thêm vào đầu tiên* mới được *lấy ra khỏi hàng đợi*



  - Việc thêm vào diễn ra ở cuối, việc lấy ra diễn ra ở đầu

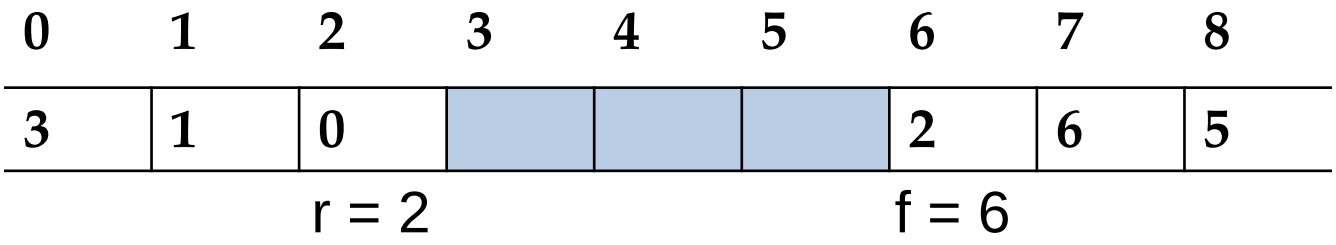
Lưu trữ hàng đợi

- Lưu trữ bằng mảng:
  - Khai báo mảng 1 chiều với kích thước tối đa N.
  - f là địa chỉ của phần tử nằm ở đầu, r là địa chỉ của phần tử nằm ở cuối



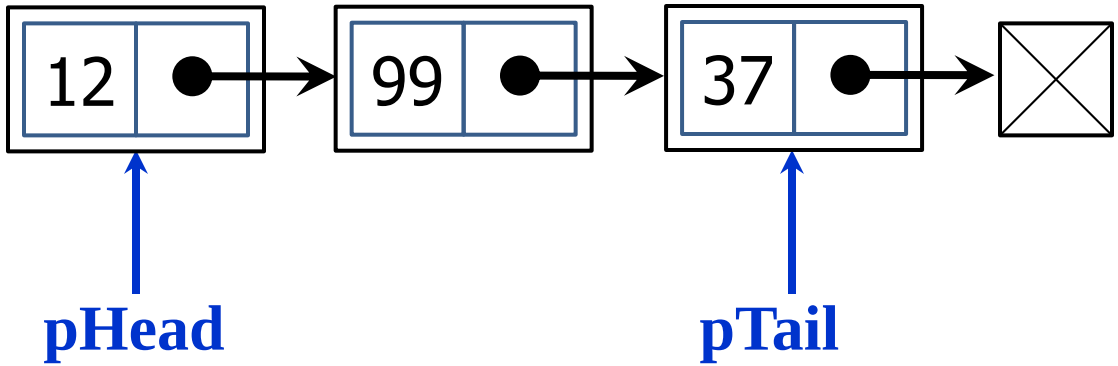
Lưu trữ hàng đợi

- Lưu trữ bằng mảng:
  - Các phần tử của hàng đợi sẽ di chuyển khắp các ô nhớ → coi không gian dành cho hàng đợi theo dạng xoay vòng.
  - Hàng đợi khi xoay vòng:



Lưu trữ hàng đợi

- Lưu trữ hàng đợi bằng danh sách liên kết đơn
  - Phần tử đầu DSLK sẽ là phần tử đầu hàng đợi.
  - Phần tử cuối DSLK sẽ là phần tử cuối hàng đợi

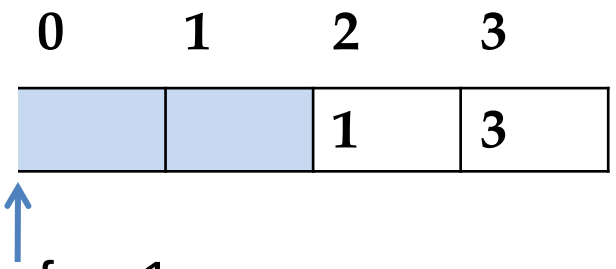


Các thao tác trên hàng đợi

- Thao tác cơ bản:
  - Enqueue: Thêm 1 đối tượng vào cuối hàng đợi
  - Dequeue: Lấy đối tượng ở đầu ra khỏi hàng đợi
- Thao tác khác:
  - Kiểm tra hàng đợi rỗng: isempty()
  - Kiểm tra hàng đợi đầy: isfull()
  - Lấy thông tin của đối tượng ở đầu hàng đợi: peep()

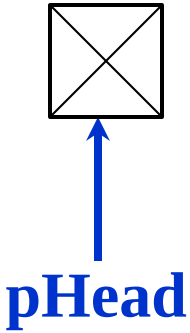
Kiểm tra hàng đợi rỗng: isempty()

- Input:



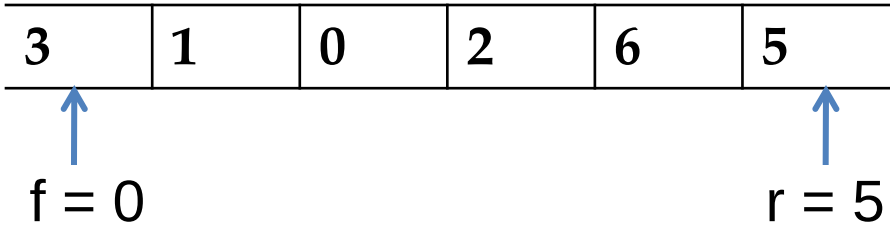
- Output:
  - TRUE nếu hàng đợi rỗng
  - FALSE nếu hàng đợi không rỗng

- Hàng đợi rỗng:
  - Mảng: ô nhớ đầu tiên không chứa dữ liệu
  - DSLK: pHead = NULL

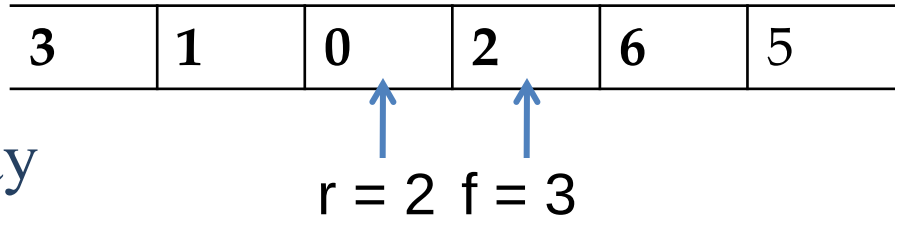


Kiểm tra hàng đợi đầy: isfull()

- Input:



- Output:
  - TRUE nếu hàng đợi đầy
  - FALSE nếu hàng đợi không đầy



- Hàng đợi đầy:
  - Mảng: ô nhớ cuối hàng đợi đã chứa dữ liệu
  - DSLK: không cấp phát được vùng nhớ cho phần tử mới

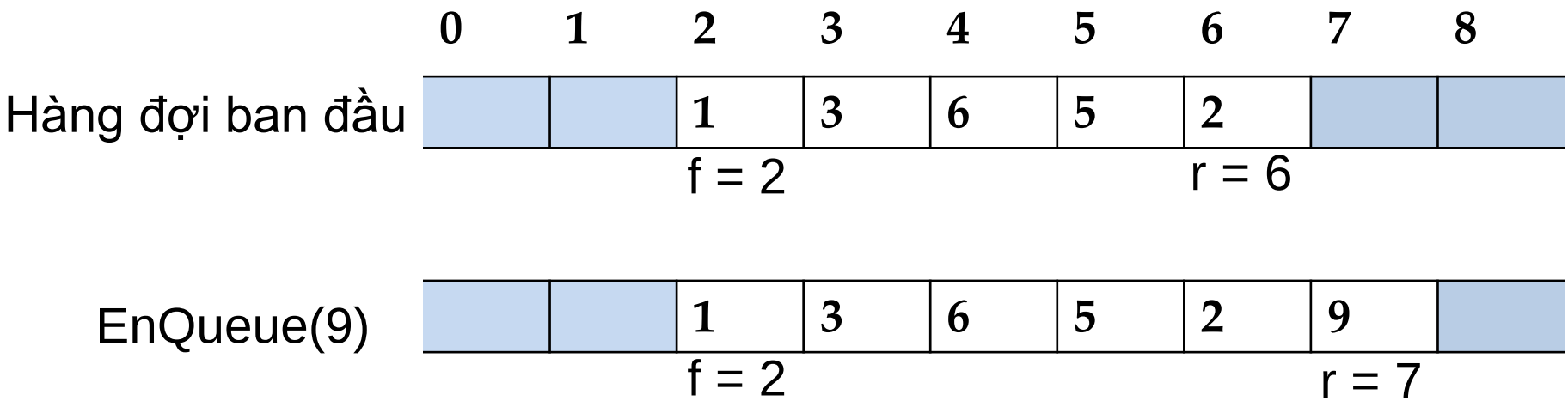
Thêm phần tử vào cuối hàng đợi

- Input: giá trị cần thêm
- Output:

- Giải thuật thêm phần tử (EnQueue)
  - Kiểm tra hàng đợi đã đầy chưa?
  - *Trong trường hợp lưu trữ bằng mảng*: kiểm tra điều kiện xoay vòng.
  - Thêm phần tử vào cuối hàng đợi
  - Cập nhật địa chỉ phần tử cuối hàng đợi

Thêm phần tử vào cuối hàng đợi

- Ví dụ:



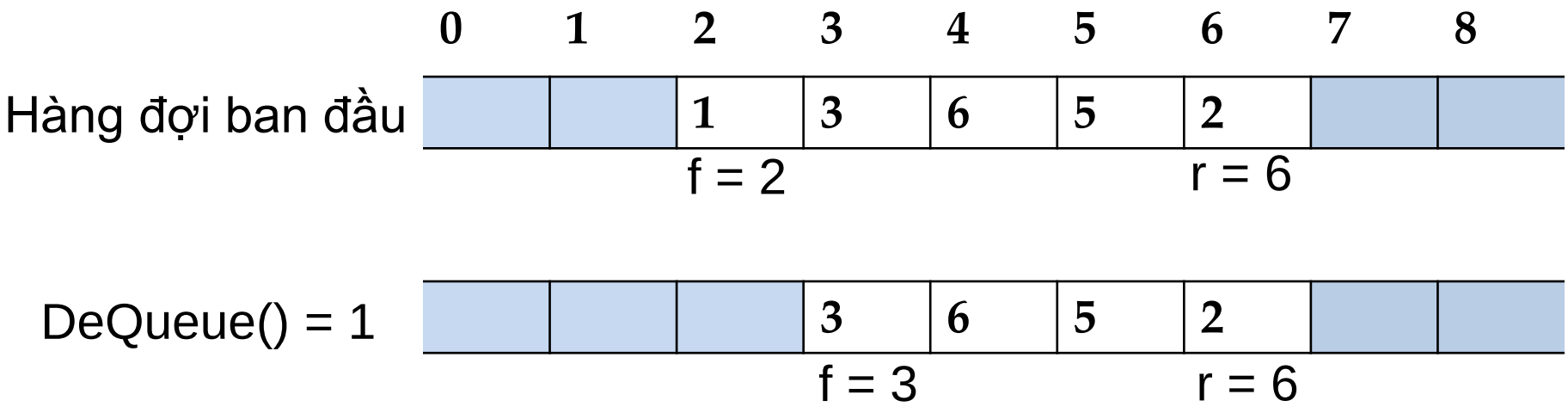
Lấy phần tử đầu ra khỏi hàng đợi

- Input:
- Output: giá trị của phần tử đầu hàng đợi

- Giải thuật lấy phần tử ở đầu (DeQueue)
  - Kiểm tra hàng đợi có rỗng không?
  - Xóa phần tử đầu ra khỏi hàng đợi
  - Cập nhật địa chỉ phần tử đầu hàng đợi
  - *Trong trường hợp lưu trữ bằng mảng:* kiểm tra điều kiện xoay vòng

Lấy phần tử đầu ra khỏi hàng đợi: peep()

- Ví dụ:



Lấy thông tin đầu hàng đợi

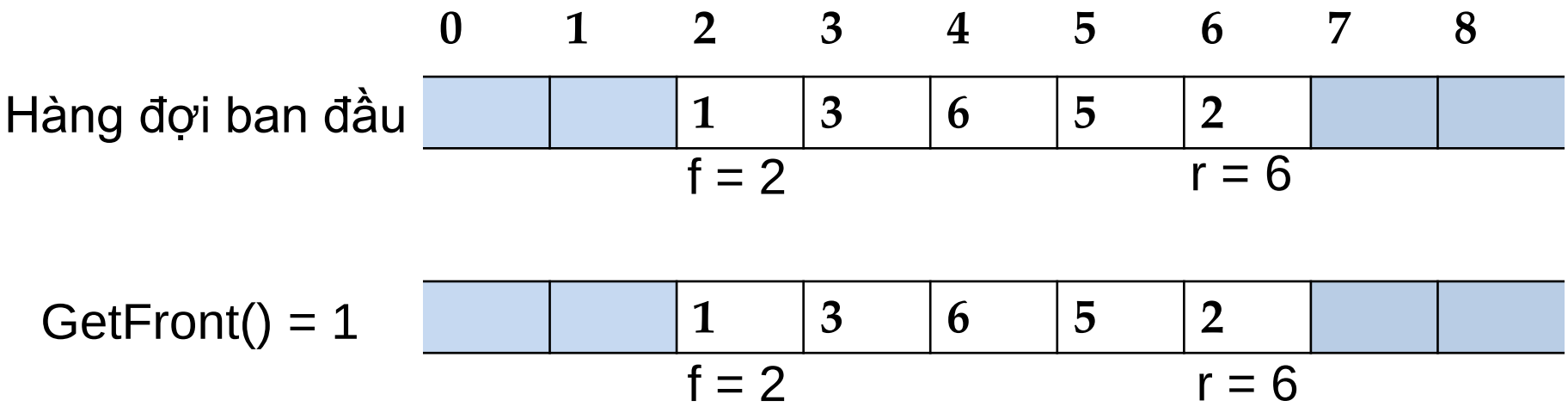
- Chỉ lấy thông tin của đối tượng đầu hàng đợi mà *không hủy* đối tượng khỏi hàng đợi.

- Input: hàng đợi
- Output: giá trị của đối tượng đầu hàng đợi

- Giải thuật:
  - Kiểm tra hàng đợi rỗng?
  - Trả về giá trị của phần tử đầu hàng đợi

Lấy thông tin đầu hàng đợi

- Ví dụ:



Ứng dụng của hàng đợi

- Bộ đệm bàn phím của máy tính
- Xử lý các lệnh trong máy tính: hàng đợi thông điệp trong Windows, hàng đợi tiến trình …
- Thường dùng trong các hệ mô phỏng.

Code

- Cho trước hàng đợi q cài đặt bằng mảng xoay vòng có thể chứa tối đa n phần tử
- Thông tin vị trí đầu và cuối hàng đợi được mô tả bởi f và r
  - Viết hàm :

void Enqueue(int q[], int N, int f, int r, int value)

  - Viết hàm :

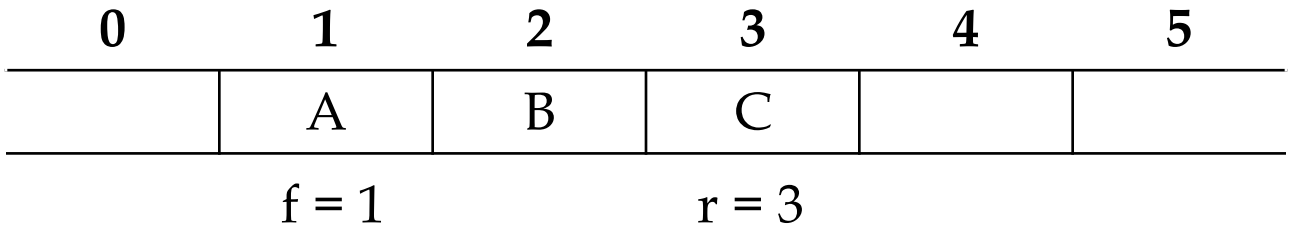
int Dequeue(int q[], int N, int f, int r)

  - Viết hàm :

int Peep(int q[], int N, int f, int r)

Bài tập

Cho hàng đợi ban đầu như sau: (hàng đợi có tối đa 6 phần tử)



Vẽ tình trạng của hàng đợi, cho biết giá trị f, r tương ứng với mỗi lần thực hiện thao tác sau:

a. Bổ sung E vào hàng đợi
b. Loại 2 phần tử khỏi hàng đợi
c. Bổ sung I, J, K vào hàng đợi
d. Loại 2 phần tử khỏi hàng đợi
e. Bổ sung O vào hàng đợi
f. Loại 2 phần tử khỏi hàng đợi

Hỏi và Đáp