Bài 5.4: Hàng đợi vòng

- ✓ Định nghĩa và đặc điểm
- ✓ Ứng dụng của hàng đợi vòng
- ✓ Các hành động đặc trưng
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

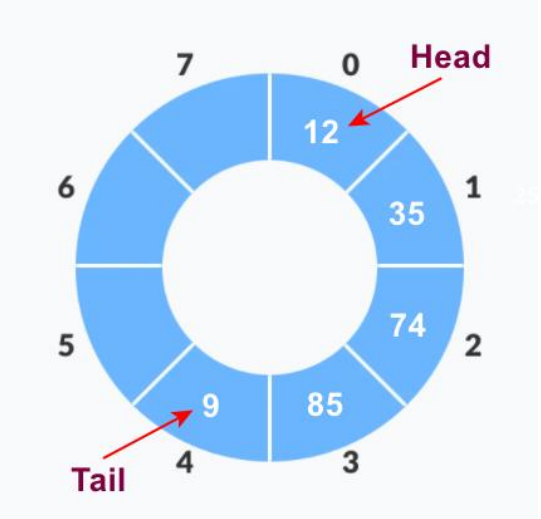
Định nghĩa và đặc điểm

- Circular queue – hàng đợi vòng là một biến thể của hàng đợi.
- Đây là một cấu trúc dữ liệu tuyến tính, trong đó phần tử cuối queue trỏ đến phần tử đầu queue tạo ra một vòng khép kín.
- Hàng đợi vòng dùng để giải quyết các hạn chế của hàng đợi thông thường triển khai bằng mảng.
- Hàng đợi vòng tuân thủ quy tắc FIFO.
- Hàng đợi vòng còn được gọi là bộ đệm vòng – ring buffer.

Ứng dụng

- Hàng đợi vòng được ứng dụng trong quản lý cấp phát bộ nhớ.
- Hàng đợi vòng được ứng dụng trong lập lịch CPU.
- Hệ thống đèn giao thông cũng ứng dụng hàng đợi vòng: các màu của đèn giao thông nhảy theo một vòng khép kín.

Ví dụ



Các hành động

- **push(data)** – thêm phần tử mới vào cuối queue.
- **pop()** – xóa phần tử ở đầu queue.
- **front()** – lấy phần tử đầu queue.
- **isFull()** – kiểm tra xem queue đã đầy chưa.
- **isEmpty()** – kiểm tra xem queue có rỗng không.
- **size()** – trả về số phần tử hiện có của queue.

Các hành động

Tạo queue vòng với các thành phần:

- Biến headIndex lưu vị trí của phần tử đầu hàng đợi.
- Biến tailIndex lưu vị trí của phần tử cuối hàng đợi.
- Biến data kiểu ArrayList<E> để lưu các phần tử của hàng đợi.
- Biến currentElement để lưu số phần tử hiện tại của hàng đợi.
- Biến capacity để lưu số phần tử tối đa có thể chứa của hàng đợi.

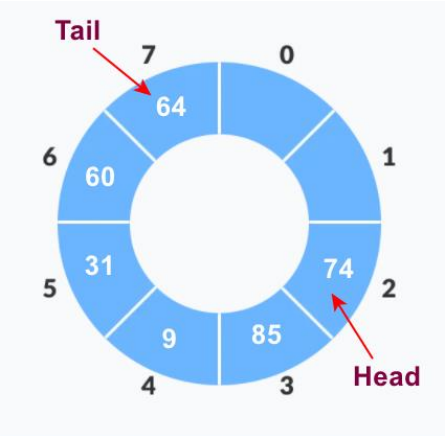
Kiểm tra rỗng, đầy

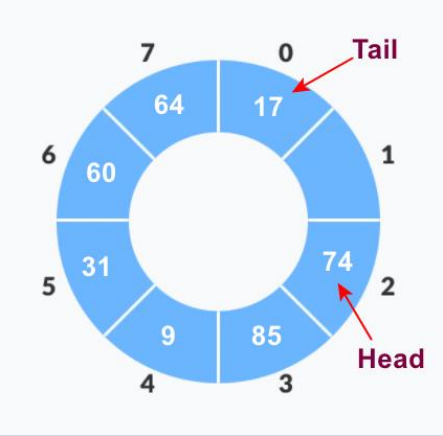
- Hàng đợi rỗng khi currentElement = 0 hoặc headIndex = -1.
- Hàng đợi đầy khi currentElement = capacity hoặc headIndex = tailIndex + 1. Hoặc headIndex = 0 và tailIndex = capacity-1.

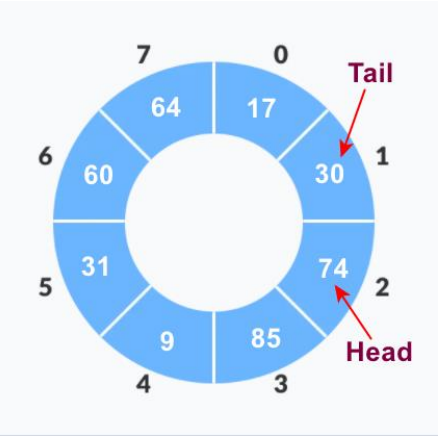
```
// kiểm tra rỗng
bool isEmpty() {
    return currentElement == 0;
}
// kiểm tra đầy
bool isFull() {
    return currentElement == capacity;
}
```

Thêm phần tử vào queue

- Nếu queue đầy, thông báo thêm phần tử thất bại, return false.
- Nếu queue rỗng: headIndex = 0;
- tailIndex = (tailIndex + 1) % capacity;
- Gán data[tailIndex] = e;
- Tăng currentElement: currentElement++;





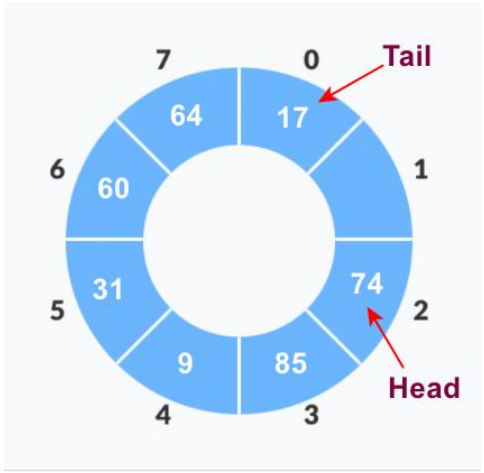


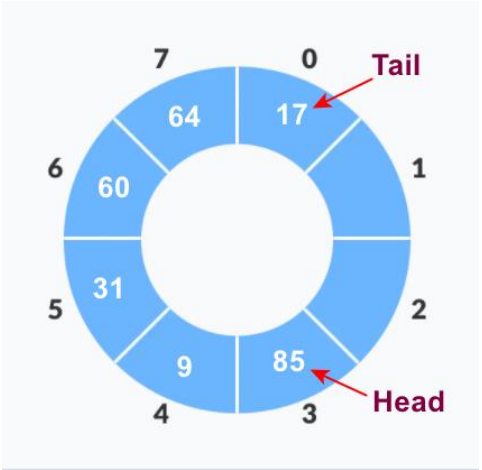
Thêm phần tử vào queue

```
// thêm mới một phần tử vào hàng đợi
bool push(T e) {
    if (isFull()) {
        cout << "Queue day. Them that bai.\n";
        return false;
    }
    if (isEmpty()) {
        headIndex = 0;
    }
    tailIndex = (tailIndex + 1) % capacity;
    data[tailIndex] = e;
    currentElement++;
    return true;
}
```

Xóa phần tử khỏi queue

- Nếu queue rỗng, văng ngoại lệ hoặc trả về null.
- lưu lại giá trị tại vị trí headIndex.
- Nếu queue chỉ có 1 phần tử, gán headIndex = tailIndex = -1. Nếu không rỗng, headIndex = (headIndex + 1) % capacity;
- Gán headIndex = (headIndex + 1) % SIZE;
- Giảm currentElement: currentElement--;
- Trả về phần tử vừa xóa.





Xóa phần tử khỏi queue

```
// xóa phần tử đầu queue
T pop() {
    if (isEmpty()) {
        throw exception("Queue rong.");
    }
    else {
        T front = data[headIndex];
        currentElement--;
        if (headIndex == tailIndex) {
            headIndex = tailIndex = -1;
        }
        else {
            headIndex = (headIndex++) % capacity;
        }
        return front;
    }
}
```

Lấy phần tử đầu queue

- Nếu queue rỗng, văng ngoại lệ với thông điệp queue rỗng hoặc trả về null.
- Ngược lại, trả về giá trị của node head.

```
// lấy phần tử đầu hàng đợi
T front() {
    if (!isEmpty()) {
        return data[headIndex];
    }
    else {
        throw exception("Queue rong.");
    }
}
```

Hiển thị các phần tử

```
// hiển thị các phần tử trong queue
void showNodes() {
    if (isEmpty()) {
        cout << "Queue rong." << endl;
    }
    else {
        cout << "Cac phan tu trong queue: ";
        for (int i = headIndex; i != tailIndex; i = (i + 1) % capacity) {
            cout << data[i] << " ";
        }
        cout << data[tailIndex] << endl;
    }
}
```

Nội dung tiếp theo

Hàng đợi ưu tiên