Bài 4.4: Hàng đợi vòng

- ✓ Định nghĩa và đặc điểm
- ✓ Ứng dụng của hàng đợi vòng
- ✓ Các hành động đặc trưng

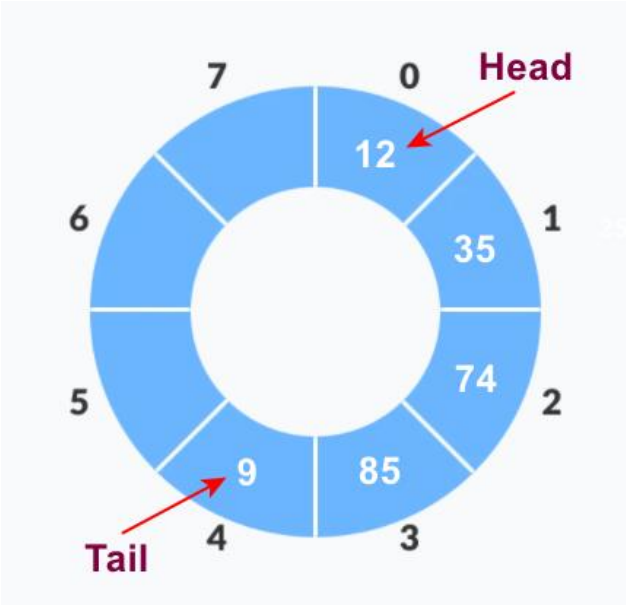
Định nghĩa và đặc điểm

- 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 ứng dụng trong quản lý cấp phát bộ nhớ.
- Hàng đợi vòng 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.

Ví dụ



Các hành động

- enqueue(e): thêm phần tử mới vào queue.
- dequeue(): xóa và trả về phần tử đầu queue.
- peek(): lấy phần tử đầu queue nhưng không xóa.
- isEmpty(): kiểm tra queue có rỗng hay không.
- isFull(): kiểm tra queue có đầy không.
- size(): trả về số lượng phần tử hiện có trong queue.
- display(): hiển thị các phần tử trong queue.

Tạo hàng đợi vòng

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.

```
public class CircularQueue <E> {
    private int headIndex; // chỉ số phần tử đầu hàng đợi vòng
    private int tailIndex; // chỉ số phần tử cuối hàng đợi vòng
    private int capacity;  // khả năng lưu trữ tối đa của hàng đợi vòng
    private int currentElement; // số phần tử hiện thời của hàng đợi vòng
    private ArrayList<E> data; // mảng dùng để lưu trữ các phần tử của hàng đợi vòng
    //...
}
```

Kiểm tra queue rỗng

- Hàng đợi vòng rỗng khi headIndex == -1
- Hoặc currentElement == 0.

```
public boolean isEmpty() {
    return currentElement == 0;
}
```

Kiểm tra queue đầy

- Hàng đợi vòng đầy khi headIndex == 0 && tailIndex == capacity – 1.
- Hoặc headIndex == tailIndex + 1.
- Hoặc currentElement == capacity.

```
public boolean isFull() {
    return currentElement == capacity;
}
```

Thêm phần tử e 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 e vào queue

```
public boolean enqueue(E e) { // phương thức thêm mới phần tử vào queue vòng
    if (isFull()) {
        System.out.println("Queue đầy.");
        return false;
    } else {
        currentElement++; // tăng số phần tử hiện có trong queue lên 1
        if (headIndex == -1) {
            headIndex++;
        }
        tailIndex = (tailIndex + 1) % capacity; // cập nhật chỉ số phần tử cuối
        if (data.size() == capacity) { // nếu arraylist đã đạt đủ số phần tử tối đa của queue
            data.set(tailIndex, e); // gán giá trị cho vị trí tailIndex
        } else { // nếu chưa đạt đủ max phần tử của queue
            data.add(e); // thêm phần tử mới vào arraylist
        }
        return true;
    }
}
```

Xóa phần tử khỏi queue

- Nếu queue rỗng, thông báo queue rỗng và return null.
- Nếu queue chỉ có 1 phần tử, ta gán headIndex = tailIndex = -1.
- headIndex = (headIndex + 1) % SIZE;
- Giảm currentElement: currentElement--;

Xóa phần tử khỏi queue

```java
public E dequeue() { // phương thức xóa và trả về phần tử đầu queue vòng
    if (isEmpty()) {
        System.out.println("Queue rỗng.");
        return null;
    } else {
        E front = data.get(headIndex); // lấy giá trị phần tử đầu
        currentElement--; // giảm số phần tử hiện có của queue
        if (headIndex == tailIndex) { // nếu queue chỉ có 1 phần tử
            headIndex = tailIndex = -1; // reset về queue rỗng
        } else {
            headIndex = (headIndex + 1) % capacity; // update chỉ số kế tiếp
        }
        return front; // trả về kết quả
    }
}
```

Lấy phần tử đầu queue

- Nếu queue rỗng, return null. Nếu không, return phần tử tại vị trí headIndex.

```
public E peek() { // phương thức lấy phần tử đầu queue vòng
    if (!isEmpty()) {
        return data.get(headIndex);
    }
    return null;
}
```

Hiển thị các phần tử trong queue

- Nếu queue rỗng, hiện thông báo và kết thúc.
- Cho biến k từ headIndex đến tailIndex in ra từng phần tử.

```
public void display() { // phương thức hiển thị các phần tử trong queue
    if (isEmpty()) {
        System.out.println("Queue rỗng.");
    } else {
        int i;
        System.out.println("Phần tử head: " + data.get(headIndex)); // hiển thị độc lập head
        System.out.println("Thứ tự các phần tử trong queue: ");
        for (i = headIndex; i != tailIndex; i = (i + 1) % capacity) {
            System.out.print(data.get(i) + " "); // in danh sách các phần tử khác tail
        }
        System.out.println(data.get(i)); // in phần tử tail
        System.out.println("Phần tử tail: " + data.get(tailIndex)); // hiển thị độc lập tail
    }
}
```

Nội dung tiếp theo

Hàng đợi ưu tiên