Bài 6.5: Tạo hàng đợi ưu tiên dùng heap

- ✓ Các thao tác của priority queue
- ✓ Triển khai hàng đợi ưu tiên
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

Các thao tác của hàng đợi

- add(e): thêm phần tử mới vào queue theo mức ưu tiên từ cao đến thấp.
- remove(): xóa và trả về phần tử có mức ưu tiên cao nhất ở 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(áp dụng với triển khai bằng mảng).
- size(): trả về số lượng phần tử hiện có trong queue.
- search(e): tìm xem phần tử e có trong queue không.
- Bài này ta sẽ sử dụng max heap với quy ước phần tử có thứ tự ưu tiên cao hơn đứng đầu hàng đợi.

Triển khai hàng đợi ưu tiên

➢ Ta sửa đổi lớp Heap để tạo PriorityQueue:

```
public class PriorityQueue<E extends Comparable<E>> {
    private Node[] data;       // mảng chứa các node của hàng đợi
    private final int MAX_SIZE; // số phần tử tối đa
    private int currentSize; // số phần tử hiện thời

    public PriorityQueue(int size) {
        MAX_SIZE = size;
        currentSize = 0;
        data = new Node[MAX_SIZE];
    }

    static class Node<E extends Comparable<E>> {
        private E value;
        private int priority;

        public Node(E e, int priority) {
            this.value = e; // giá trị của mỗi node
            this.priority = priority; // thứ tự ưu tiên
        }

        public E getValue() { return value; }

        public int getPriority() { return priority; }
    }
```

Thêm node vào queue

➢ Ta truyền vào giá trị node e và thứ tự ưu tiên của nó:

```
/**
 * Phương thức thêm phần tử mới vào hàng đợi ưu tiên
 * triển khai bằng heap.
 *
 * @param e        giá trị phần tử cần thêm
 * @param priority thứ tự ưu tiên của phần tử cần thêm
 * @return true nếu thêm thành công
 */
public boolean add(E e, int priority) {
    if (!isFull()) { // nếu queue chưa đầy
        Node<E> node = new Node<>(e, priority); // tạo node mới
        data[currentSize] = node; // gán giá trị node vào heap
        siftUp(currentSize); // sàng lên
        currentSize++; // tăng số lượng phần tử của queue
        return true; // thêm thành công
    } else {
        System.out.println("Queue đầy!");
        return false; // thêm thất bại
    }
}
```

Kiểm tra rỗng, đầy, lấy size()

- Sau đây là mã thật thao tác kiểm tra rỗng, đầy, lấy kích thước của hàng đợi ưu tiên:

```
// kiểm tra đầy
public boolean isFull() { return currentSize == MAX_SIZE; }

// kiểm tra rỗng
public boolean isEmpty() { return currentSize == 0; }

// kiểm tra kích thước hàng đợi
public int size() { return currentSize; }
```

Xóa phần tử đầu queue

- Sau đây là mã thật thao tác pop:

```
/**
 * Phương thức xóa và trả về phần tử đầu hàng đợi
 *
 * @return phần tử bị xóa hoặc null nếu xóa thất bại
 */
public Node<E> pop() {
    if (!isEmpty()) {
        Node<E> removedNode = data[0]; // lưu lại phần tử cần xóa
        data[0] = data[currentSize - 1]; // tráo đổi giá trị
        data[currentSize - 1] = null; // hủy liên kết node cuối
        currentSize--; // giảm số phần tử của hàng đợi
        siftDown( index: 0); // sàng xuống
        return removedNode; // xóa thành công
    } else {
        return null; // xóa thất bại
    }
}
```

Lấy phần tử đầu queue

- Sau đây là mã thật thao tác peek:

```
/**
 * Phương thức lấy phần tử đầu hàng đợi
 *
 * @return giá trị đầu hàng đợi hoặc null nếu hàng đợi rỗng
 */
public Node<E> peek() {
    if (!isEmpty()) {
        return data[0];
    } else {
        return null;
    }
}
```

Tìm kiếm phần tử trong queue

➢ Sau đây là mã thật thao tác search:

```
/**
 * Phương thức tìm kiếm node trong hàng đợi
 *
 * @param e node cần tìm
 * @return chỉ số của phần tử e trong hàng đợi
 * hoặc -1 nếu e không tồn tại trong queue
 */
public int search(E e) {
    for (int i = 0; i < currentSize; i++) {
        if (data[i].getValue().compareTo(e) == 0) {
            return i;
        }
    }
    return -1; // không tìm thấy node e trong heap
}
```

Nội dung tiếp theo

Thuật toán sắp xếp heap sort