Bài 7.5: Hàng đợi ưu tiên vs 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

- push(e): thêm phần tử mới vào queue theo mức ưu tiên từ cao đến thấp.
- pop(): xóa và trả về phần tử có mức ưu tiên cao nhất ở đầu queue.
- front(): lấy phần tử đầu queue nhưng không xóa.
- back(): lấy phần tử cuối 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.

Tạo node

```
template<class T> class Node {
public:
    T value;
    int priority;
    // hàm khởi tạo
    Node(T value = T(), int priority = 0) {
        this->value = value;
        this->priority = priority;
    }
    // nạp chồng toán tử so sánh <
    bool operator < (const Node<T>& other) {
        return priority < other.priority;
    }
    // nạp chồng toán tử so sánh >
    bool operator > (const Node<T>& other) {
        return priority > other.priority;
    }
    // nạp chồng toán tử so sánh ==
    bool operator == (const Node<T>& other) {
        return value == other.value;
    }
};
```

Tạo hàng đợi ưu tiên

```
template<class T> class PriorityQueue {
    Node<T>* data;
    int currentSize;
    int capacity;
public:
    PriorityQueue(int cap = 10) { ... }
    // thêm node mới vào hàng đợi
    bool push(T value, int priority) { ... }
    // hàm sàng lên
    void siftUp(int index) { ... }
    // hàm hiển thị
    void showElements() { ... }
    // hàm lấy kích thước hiện thời của hàng đợi
    int size() { ... }
    // hàm kiểm tra hàng đợi rỗng
    bool isEmpty() { ... }
    // hàm kiểm tra hàng đợi đầy
    bool isFull() { ... }
    // hàm xóa giá trị value trong heap
    Node<T> pop() { ... }
    // hàm lấy ra giá trị đầu hàng đợi
    Node<T> front() { ... }
    // hàm lấy ra giá trị cuối hàng đợi
    Node<T> back() { ... }
    // hàm sàng xuống
    void siftDown(int index) { ... }
    // hàm tìm một node
    int search(T key) { ... }
    // hàm hủy
    ~PriorityQueue() { ... }
};
```

Thêm node vào queue

- Ta truyền vào giá trị value và thứ tự ưu tiên priority của nó:

```
// thêm node mới vào hàng đợi
bool push(T value, int priority) {
    if (currentSize < capacity) {
        data[currentSize] = Node(value, priority);
        siftUp(currentSize);
        currentSize++;
        return true;
    }
    else {
        return false;
    }
}
```

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:

```
// hàm lấy kích thước hiện thời của hàng đợi
int size() {
    return currentSize;
}
// hàm kiểm tra hàng đợi rỗng
bool isEmpty() {
    return currentSize == 0;
}
// hàm kiểm tra hàng đợi đầy
bool isFull() {
    return currentSize == capacity;
}
```

Xóa phần tử đầu queue

- Sau đây là mã thật thao tác pop:

```
// hàm xóa giá trị value trong heap
Node<T> pop() {
    if (!isEmpty()) {
        Node<T> tmp = data[0];
        currentSize--;
        siftDown(0);
        return tmp;
    }
    else {
        throw exception("Queue rong.");
    }
}
```

Lấy phần tử đầu, cuối queue

- Sau đây là mã thật thao tác front, back:

```
// hàm lấy ra giá trị đầu hàng đợi
Node<T> front() {
    if (!isEmpty()) {
        return data[0];
    }
    else {
        throw exception("Queue rong.");
    }
}
// hàm lấy ra giá trị cuối hàng đợi
Node<T> back() {
    if (!isEmpty()) {
        return data[currentSize - 1];
    }
    else {
        throw exception("Queue rong.");
    }
}
```

Tìm kiếm phần tử trong queue

- Sau đây là mã thật thao tác search:

```
// hàm tìm một node
int search(T key) {
    for (int i = 0; i < currentSize; i++)
    {
        if (data[i] == key) { // nếu tìm thấy
            return i; // trả về vị trí của node đó
        }
    }
    return -1; // mặc định không tìm thấy
}
```

Nội dung tiếp theo

Thuật toán sắp xếp heap sort