Bài 5.5: Hàng đợi ưu tiên

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

Định nghĩa và đặc điểm

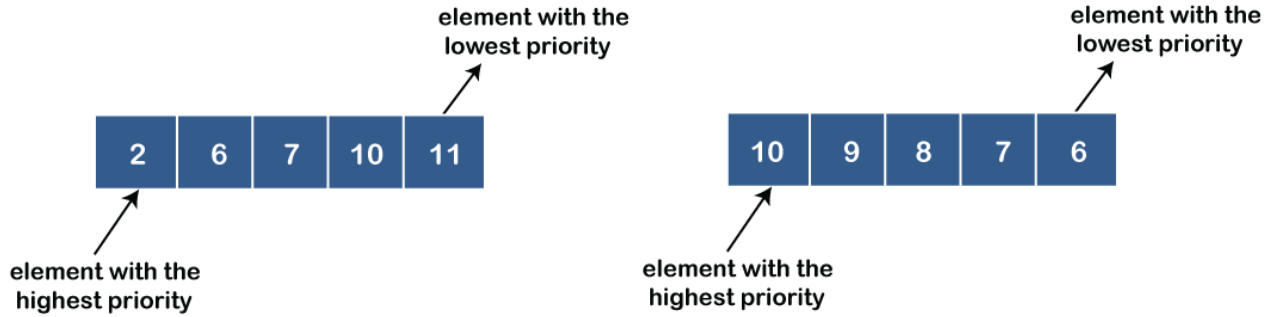
- Hàng đợi ưu tiên(priority queue) là biến thể của hàng đợi trong đó mỗi phần tử sẽ có mức ưu tiên khác nhau. Phần tử có mức ưu tiên cao nhất sẽ được đưa lên đầu queue và được lấy ra đầu tiên. Phần tử có mức ưu tiên nhỏ hơn sẽ được lấy ra sau.
- Hàng đợi ưu tiên chỉ hỗ trợ các phần tử có thể so sánh với nhau, tức là các phần tử trong hàng đợi sẽ được sắp xếp theo trật tự tăng dần hoặc giảm dần.
- Đặc trưng 1: mỗi phần tử sẽ có một mức ưu tiên gắn liền với nó.
- Đặc trưng 2: một phần tử có mức ưu tiên cao hơn sẽ bị xóa khỏi hàng đợi trước các phần tử có mức ưu tiên thấp hơn.
- Đặc trưng 3: nếu 2 phần tử có cùng mức ưu tiên chúng sẽ được sắp xếp theo thứ tự FIFO.

Ứng dụng

- Thuật toán tìm đường đi ngắn nhất: Dijkstra.
- Thuật toán Prim.
- Nén dữ liệu.
- Thuật toán tìm kiếm A* trong trí tuệ nhân tạo.
- Heap sort.
- Trong hệ điều hành: cân bằng tải, xử lý ngắt

Phân loại

- Có 2 loại hàng đợi ưu tiên:
- Loại 1: hàng đợi ưu tiên có các phần tử được sắp xếp theo thứ tự tăng dần giá trị các phần tử theo tiêu chí nào đó. Trong hàng đợi kiểu này, phần tử có giá trị nhỏ hơn sẽ có mức ưu tiên cao hơn.



- Loại 2: hàng đợi ưu tiên có các phần tử được sắp xếp theo thứ tự giảm dần giá trị các phần tử theo tiêu chí nào đó. Trong hàng đợi kiểu này, các phần tử có giá trị lớn hơn sẽ có mức ưu tiên cao hơn.

Các hành động

- 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.
- top(): lấy phần tử đầu queue nhưng không xóa.
- isEmpty(): kiểm tra queue có rỗng hay không.
- size(): trả về số lượng phần tử hiện có trong queue.

Triển khai hàng đợi ưu tiên

- Sử dụng mảng.
- Sử dụng danh sách liên kết.
- Sử dụng heap.
- Sử dụng cây nhị phân tìm kiếm.
- Trong bài này ta sẽ triển khai hàng đợi ưu tiên sử dụng danh sách liên kết.

Tạo hàng đợi ưu tiên

```
template<class T> class Node { // lớp mô tả thông tin một node
public:
    T data; // phần dữ liệu
    int priority; // thứ tự ưu tiên
    Node<T>* next; // con trỏ next

    Node(T data, int priority) { ... }
};

template<class T> class PriorityQueue { // lớp queue template
private:
    Node<T>* head; // node đầu queue
    Node<T>* tail; // node cuối queue
    int currentSize; // số phần tử hiện có trong queue
public:
    // hàm khởi tạo
    PriorityQueue() { ... }
    // kiểm tra queue rỗng
    bool isEmpty() { ... }
    // thêm phần tử mới vào vị trí phù hợp trong queue
    void push(T data, int priority) { ... }
    // xóa và trả về phần tử top
    T pop() { ... }
    // trả về phần tử đầu queue
    T front() { ... }
    // trả về phần tử cuối queue
    T back() { ... }
    // kích thước queue
    int size() { ... }
};
```

Lớp Node bổ sung priority

```cpp
template<class T> class Node { // lớp mô tả thông tin một node
public:
    T data; // phần dữ liệu
    int priority; // thứ tự ưu tiên
    Node<T>* next; // con trỏ next

    Node(T data, int priority) {
        this->priority = priority; // mặc định thứ tự ưu tiên của node là 0
        this->next = nullptr; // khởi tạo con trỏ next
        this->data = data; // gán dữ liệu cho node hiện tại
    }
};
```

Các hành động

```
// hàm khởi tạo
PriorityQueue() {
    currentSize = 0;
    head = nullptr;
    tail = nullptr;
}
// kiểm tra queue rỗng
bool isEmpty() {
    return head == nullptr;
}
// kích thước queue
int size() {
    return currentSize;
}
```

Thêm mới 1 phần tử

```
// thêm phần tử mới vào vị trí phù hợp trong queue
void push(T data, int priority) {
    Node<T>* p = new Node<T>(data, priority);
    if (isEmpty()) {
        head = tail = p;
    }
    else if(priority > head->priority) { // chèn đầu queue
        p->next = head;
        head = p;
    }
    else { // tìm node r đứng trước node p
        Node<T>* r = head;
        for (Node<T>* q = head->next; q != nullptr; q = q->next) {
            if (q->priority < priority) {
                break;
            }
            r = q;
        }
        p->next = r->next;
        r->next = p;
        if (r == tail) { // nếu r là node tail cũ
            tail = p; // cập nhật tail mới
        }
    }
    currentSize++; // tăng số phần tử hiện có
}
```

Xóa node đầu hàng đợi

```
// xóa và trả về phần tử top
T pop() {
    if (!isEmpty()) {
        Node<T>* p = head;
        head = head->next;
        currentSize--;
        return p->data;
    }
    else {
        throw exception("Queue rong.");
    }
}
```

Lấy phần tử đầu, cuối queue

```
// trả về phần tử đầu queue
T front() {
    if (!isEmpty()) {
        return head->data;
    }
    else {
        throw exception("Queue rong.");
    }
}
// trả về phần tử cuối queue
T back() {
    if (!isEmpty()) {
        return tail->data;
    }
    else {
        throw exception("Queue rong.");
    }
}
```

Nội dung tiếp theo

Tìm hiểu thư viện <queue>