VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Cấu Trúc Dữ Liệu và Giải Thuật (DSA)

DSA1 - HK242

Queue

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 1/2025

Mục lục

1 Khái niệm

Hàng đợi (Queue)

Queue là một cấu trúc dữ liệu tuân theo nguyên tắc **First In First Out (FIFO)**. Điều này có nghĩa là phần tử đầu tiên được thêm vào *queue* sẽ là phần tử đầu tiên được xóa.

Queue thường được sử dụng trong các trường hợp:

- Quản lý tác vụ hoặc xử lý công việc theo trình tự, chẳng hạn như trong hàng chờ in ấn.
- Lập lịch CPU hoặc các thuật toán lập lịch khác.
- Thực hiện thuật toán duyệt đồ thị **BFS (Breadth-First Search)**.
- Lưu trữ và quản lý dữ liệu trong các ứng dụng mạng như hàng đợi gói tin.

Dữ liệu triển khai: *Queue* thường được triển khai bằng:

- **Mảng (Array):** Đơn giản và hiệu quả với kích thước cố định, nhưng cần xử lý vòng lặp khi sử dụng mảng vòng (circular array).
- **Danh sách liên kết (Linked List):** Linh hoạt với kích thước động, cho phép dễ dàng thêm và xóa phần tử.

Các thao tác cơ bản với độ phức tạp $O(1)$:

- **Enqueue:** Thêm một phần tử vào cuối hàng đợi.
- **Dequeue:** Xóa phần tử ở đầu hàng đợi.
- **Peek/Front:** Trả về phần tử ở đầu hàng đợi mà không xóa nó.
- **IsEmpty:** Kiểm tra xem hàng đợi có trống không.
- **Size:** Trả về số lượng phần tử trong hàng đợi.

```cpp
// Abstract class for a Queue
template <typename T>
class AbstractQueue {
public:
    // Destructor
    virtual ~AbstractQueue() {}

    // Add an element to the end of the queue
    virtual void enqueue(const T& value) = 0;

    // Remove the front element from the queue
    virtual void dequeue() = 0;

    // Get the front element of the queue without removing it
    virtual T front() const = 0;

    // Check if the queue is empty
    virtual bool isEmpty() const = 0;

    // Get the number of elements in the queue
    virtual size_t size() const = 0;
};
```

2 Hiện thực Queue bằng Array

Trong hiện thực bằng **Array**, queue được lưu trữ trong một mảng với các thao tác thêm (**enqueue**), xóa (**dequeue**), và truy cập phần tử đầu (**front**) được thực hiện với độ phức tạp trung bình **O(1)**. Queue sẽ sử dụng kỹ thuật mảng vòng (**circular array**) để quản lý hiệu quả không gian.

```
template <typename T>
class ArrayQueue : AbstractQueue<T>
{
private:
    T* array;    //! hàng đợi
    int frontIdx; //! chỉ số của phần tử đầu
    int rearIdx;  //! chỉ số của phần tử cuối
    int Size;     //! số lượng phần tử hiện tại
    const int MAXSIZE; //! kích thước tối đa

public:
    // Constructor
    ArrayQueue(int maxSize = 100)
        : MAXSIZE(maxSize), frontIdx(0), rearIdx(-1), Size(0) {
        array = new T[MAXSIZE]; // Cấp phát bộ nhớ cho mảng
    }

    // Destructor
    ~ArrayQueue() {
        delete[] array; // Giải phóng bộ nhớ
    }

    // function extend AbstractQueue
};
```

Giải thích

- **T* array;** Mảng động dùng để lưu trữ các phần tử trong queue.
- **int frontIdx;** Chỉ số của phần tử đầu trong queue.
- **int rearIdx;** Chỉ số của phần tử cuối trong queue.
- **int Size;** Số lượng phần tử hiện tại trong queue.
- **const int MAXSIZE;** Kích thước tối đa của queue, được khởi tạo thông qua constructor.
- **Constructor ArrayQueue(int maxSize = 100)** cấp phát bộ nhớ cho mảng **array** với kích thước tối đa là **maxSize**. Các giá trị **frontIdx = 0**, **rearIdx = -1**, và **Size = 0**.
- **Destructor ArrayQueue()** giải phóng bộ nhớ đã cấp phát cho **array**.

Các hàm được thừa kế từ class AbstractQueue

1. `enqueue(const T& value)`

```
// Thêm phần tử vào cuối queue
void enqueue(const T& value) override {
    if (Size >= MAXSIZE) {
        throw std::overflow_error("Queue overflow: Cannot enqueue, queue is
        ↪ full.");
    }
    rearIdx = (rearIdx + 1) % MAXSIZE; // Vòng qua mảng
```

```
    array[rearIdx] = value; // Thêm phần tử
    Size++; // Tăng kích thước
}
```

- **Chức năng:** Thêm một phần tử vào cuối queue.
- **Cách hoạt động:**
  - Kiểm tra nếu queue đầy $(Size \geq MAXSIZE)$, ném ngoại lệ `std::overflow_error`.
  - Tính chỉ số tiếp theo cho **rearIdx** bằng công thức vòng $(rearIdx + 1)\%MAXSIZE$.
  - Đặt giá trị tại vị trí `rearIdx` và tăng `Size`.

2. `dequeue()`

```
// Xóa phần tử ở đầu queue
void dequeue() override {
    if (isEmpty()) {
        throw std::underflow_error("Queue underflow: Cannot dequeue, queue is
        ↪ empty.");
    }
    frontIdx = (frontIdx + 1) % MAXSIZE; // Vòng qua mảng
    Size--; // Giảm kích thước
}
```

- **Chức năng:** Xóa phần tử ở đầu queue.
- **Cách hoạt động:**
  - Kiểm tra nếu queue rỗng $(Size == 0)$, ném ngoại lệ `std::underflow_error`.
  - Tính chỉ số tiếp theo cho **frontIdx** bằng công thức vòng $(frontIdx + 1)\%MAXSIZE$.
  - Giảm `Size`.

3. `front() const`

```
// Trả về phần tử ở đầu queue mà không xóa nó
T front() const override {
    if (isEmpty()) {
        throw std::underflow_error("Queue underflow: Cannot access front, queue
        ↪ is empty.");
    }
    return array[frontIdx]; // Trả về phần tử đầu
}
```

- **Chức năng:** Trả về phần tử ở đầu queue mà không xóa nó.
- **Cách hoạt động:**
  - Kiểm tra nếu queue rỗng, ném ngoại lệ `std::underflow_error`.
  - Trả về `array[frontIdx]`.

4. `isEmpty() const`

```cpp
// Kiểm tra queue có rỗng không
bool isEmpty() const override {
    return Size == 0; // Queue rỗng khi Size = 0
}
```

- **Chức năng:** Kiểm tra xem queue có rỗng không.
- **Cách hoạt động:**
  - Trả về `true` nếu `Size == 0`, ngược lại trả về `false`.

5. `size() const`

```cpp
// Trả về số lượng phần tử trong queue
size_t size() const override {
    return Size; // Trả về số phần tử hiện tại trong queue
}
```

- **Chức năng:** Trả về số lượng phần tử hiện tại trong queue.
- **Cách hoạt động:**
  - Đơn giản trả về giá trị `Size`.

3 Hiện thực Queue bằng Linked List

Trong hiện thực bằng **Linked List**, queue được lưu trữ bằng danh sách liên kết với mỗi phần tử là một `Node`. Việc thêm (**enqueue**) vào cuối, xóa (**dequeue**) ở đầu, và truy cập phần tử đầu (**front**) đều được thực hiện với độ phức tạp **O(1)**.

```cpp
template <typename T>
class LinkedListQueue
{
private:
    struct Node {
        T data; //! Dữ liệu của phần tử trong queue
        Node* next; //! Con trỏ đến phần tử tiếp theo

        Node(T value, Node* nextNode = nullptr)
            : data(value), next(nextNode) {}
    };

    Node* head; //! Con trỏ đến phần tử đầu queue
    Node* tail; //! Con trỏ đến phần tử cuối queue
    int Size; //! Kích thước hiện tại của queue

public:
    // Constructor
    LinkedListQueue() : head(nullptr), tail(nullptr), Size(0) {}

    // Destructor
    ~LinkedListQueue() {
        while (!isEmpty()) {
            dequeue();
```

```
        }
    }

    // function for Queue operations
};
```

Giải thích

- **struct Node:** Cấu trúc `Node` chứa dữ liệu và con trỏ trỏ đến phần tử tiếp theo.
- **Node* head;** Con trỏ trỏ đến phần tử đầu tiên trong queue.
- **Node* tail;** Con trỏ trỏ đến phần tử cuối cùng trong queue.
- **int Size;** Số lượng phần tử hiện tại trong queue.
- **Constructor LinkedListQueue():** Khởi tạo queue rỗng với `head = nullptr`, `tail = nullptr`, và `Size = 0`.
- **Destructor ~LinkedListQueue():** Giải phóng toàn bộ bộ nhớ khi queue bị hủy.

Các hàm chính trong Queue

1. `enqueue(const T& value)`

```
// Thêm phần tử vào cuối queue
void enqueue(const T& value) {
    Node* newNode = new Node(value); // Tạo Node mới
    if (isEmpty()) {
        head = tail = newNode; // Cả head và tail đều trỏ đến Node mới
    } else {
        tail->next = newNode; // Liên kết Node mới vào cuối queue
        tail = newNode; // Cập nhật tail
    }
    Size++;
}
```

   - **Chức năng:** Thêm phần tử vào cuối queue.
   - **Cách hoạt động:**
     - Tạo một `Node` mới chứa `value`.
     - Nếu queue rỗng, cả `head` và `tail` trỏ đến Node mới.
     - Nếu không, liên kết Node mới vào cuối queue và cập nhật `tail`.
     - Tăng `Size`.

2. `dequeue()`

```
// Xóa phần tử ở đầu queue
void dequeue() {
    if (isEmpty()) {
        throw std::underflow_error("Queue underflow: Cannot dequeue, queue is
        → empty.");
    }
    Node* temp = head; // Lưu con trỏ của phần tử hiện tại
    head = head->next; // Cập nhật head đến phần tử tiếp theo
    if (!head) { // Nếu queue trống sau khi xóa
```

```
        tail = nullptr;
    }
    delete temp; // Giải phóng bộ nhớ của phần tử cũ
    Size--;
}
```

- **Chức năng:** Xóa phần tử ở đầu queue.
- **Cách hoạt động:**
  - Nếu queue rỗng, ném ngoại lệ `std::underflow_error`.
  - Lưu con trỏ của `head` hiện tại.
  - Cập nhật `head` để trỏ đến phần tử tiếp theo.
  - Nếu `head` trống, đặt `tail = nullptr`.
  - Giải phóng bộ nhớ của phần tử cũ và giảm `Size`.

3. `front() const`

```
// Trả về phần tử ở đầu queue mà không xóa nó
T front() const {
    if (isEmpty()) {
        throw std::underflow_error("Queue underflow: Cannot access front, queue
        ↪  is empty.");
    }
    return head->data; // Trả về dữ liệu của phần tử đầu
}
```

- **Chức năng:** Trả về phần tử ở đầu queue mà không xóa nó.
- **Cách hoạt động:**
  - Nếu queue rỗng, ném ngoại lệ `std::underflow_error`.
  - Trả về giá trị của `head->data`.

4. `isEmpty() const`

```
// Kiểm tra queue có rỗng không
bool isEmpty() const {
    return head == nullptr; // Queue rỗng khi head là nullptr
}
```

- **Chức năng:** Kiểm tra xem queue có rỗng không.
- **Cách hoạt động:**
  - Trả về `true` nếu `head == nullptr`, ngược lại trả về `false`.

5. `size() const`

```
// Trả về số lượng phần tử trong queue
size_t size() const {
    return Size; // Trả về số phần tử hiện tại trong queue
}
```

- **Chức năng:** Trả về số lượng phần tử hiện tại trong queue.
- **Cách hoạt động:**
  - Đơn giản trả về giá trị `Size`.

4 So sánh Queue bằng Linked List và Array

Tiêu chí	**ArrayQueue**	**LinkedListQueue**
Bộ nhớ	Cần cấp phát trước (giới hạn kích thước)	Linh hoạt, không giới hạn số phần tử
Độ phức tạp của enqueue/dequeue	$O(1)$	$O(1)$
Tốc độ thực thi	Nhanh hơn do không cần cấp phát động	Chậm hơn do cấp phát động
Truy cập phần tử bất kỳ	$O(1)$ (dùng chỉ số)	$O(n)$ (duyệt qua danh sách)

Bảng 1: *So sánh Queue bằng Array và Linked List*