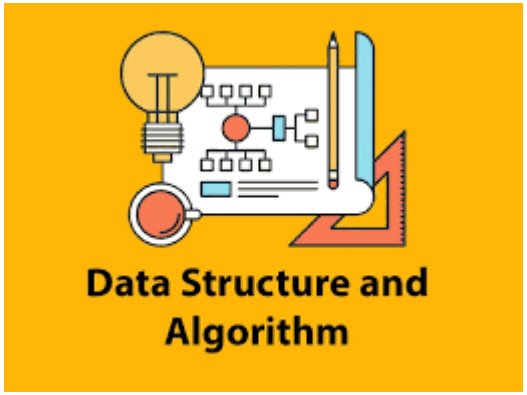
CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

NGÀY 3.1: HÀNG ĐỢI - QUEUE



Giảng viên: Th.S Bùi Văn Kiên

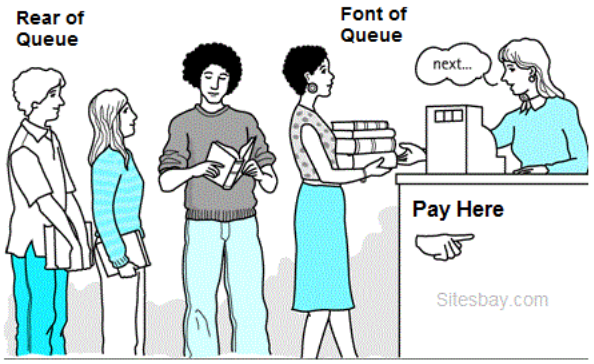
NỘI DUNG

- Khái niệm
  - Ví dụ với thư viện chuẩn
  - Triển khai với mảng
- Các bài toán ứng dụng:
  - Loang BFS

1.1 Khái niệm

- Hàng đợi là một danh sách có thứ tự của các phần tử. Hàng đợi có hai đầu, các phép toán được thực hiện như sau:
  - Thêm vào cuối.
  - Lấy ra ở đầu.

- Tính chất: vào trước ra trước (FIFO: First In, First Out).



Real Life Example of Queue : Library Counter

1.1 Khái niệm

Trừu tượng hóa cấu trúc hàng đợi

- Đặc tả dữ liệu

A = (a_0, a_1, …, a_{n-1}) trong đó

a_{n-1} là cuối hàng đợi (rear)

a_0 là đầu hàng đợi (front)

Queue Data Structure

- Các thao tác:
  - Kiểm tra hàng đợi có rỗng hay không: isEmpty()
  - Thêm phần tử x vào cuối hàng đợi: push(x) - enqueue
  - Trả về phần tử ở đầu hàng đợi: front()
  - Loại phần tử ở đầu hàng đợi : pop() - dequeue
  - Đếm số phần tử của hàng đợi : size()

1.1 Queue của thư viện chuẩn STL

- #include <queue>

```
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<int> Q;
    Q.push(10);
    Q.push(20);
    Q.push(30);
    Q.push(40);
    Q.pop();

    while(!Q.empty()) {
        cout << Q.front() << " ";
        Q.pop();
    }
    cout << endl;

    return 0;
}
```

Output = 20 30 40

1.2 Biểu diễn queue bằng mảng

- Khai báo struct:

```
#define MAX 10005

struct Queue {
    int front, rear, size, capacity;
    int arr[MAX];
    Queue(){}
    Queue(int _capacity){
        capacity = _capacity;
        front = size = 0;
        rear = -1;
    }
```

Queue Data Structure

1.2 Biểu diễn queue bằng mảng

- Thao tác 1: Kiểm tra tính rỗng của queue

```
int isEmpty(){
    return (size == 0);
}
```

- Thao tác 2: Kiểm tra tính đầy của queue

```
int isFull(){
    return (size == capacity);
}
```

1.2 Biểu diễn queue bằng mảng

- Thao tác 3: Trả về phần tử đầu tiên của hàng đợi

```
int getFront(){
    if (isEmpty())
        return -1;
    return arr[front];
}
```

- Thao tác 4: Trả về phần tử cuối cùng của hàng đợi

```
int getRear(){
    if (isEmpty())
        return -1;
    return arr[rear];
}
```

1.2 Biểu diễn queue bằng mảng

- Thao tác 5: Đưa dữ liệu vào cuối hàng đợi

```
void push(int item){
    if (isFull())
        return;
    rear = rear + 1;
    arr[rear] = item;
    size = size + 1;
    cout << item << " push to queue\n";
}
```

- Thao tác 6: Đẩy dữ liệu ra khỏi đầu hàng đợi

```
void pop(){
    if (isEmpty()) return;
    int item = arr[front];
    front = (front + 1);
    size = size - 1;
    cout << "Pop from queue " << item << endl;
}
```

1.2 Biểu diễn queue bằng mảng

Ví dụ:

Init Queue, front = 0, rear = -1	queue<int> Q
front = 0, rear = 0, arr[0] = 10	Q.push(10)
front = 0, rear = 1, arr[1] = 20	Q.push(20)
front = 0, rear = 2, arr[2] = 30	Q.push(30)
front = 0, rear = 3, arr[3] = 40	Q.push(40)
front = 1, rear = 3,	Q.pop()
front = 1, rear = 4, arr[4] = 50	Q.push(50)

2. Thuật toán loang BFS

- Kỹ thuật "loang" BFS
  - Bước 1: Đưa trạng thái xuất phát vào hàng đợi
  - Bước 2: Lặp đến khi hàng đợi rỗng hoặc gặp trạng thái đích
    - Xét trạng thái S ở đầu hàng đợi
    - Loại bỏ S ra khỏi hàng
    - Đưa các trạng thái đến được từ S vào hàng đợi

  (chú ý: nếu trạng thái đã từng có trong hàng đợi trước đó thì không thể đưa vào lần 2)
  - Bước 3: Kết luận kết quả
- Kỹ thuật BFS sẽ cho số bước chuyển trạng thái từ xuất phát đến đích là ít nhất.
- Vấn đề: đánh dấu trạng thái đã đi qua?

2. Thuật toán loang BFS

Có 3 trường hợp:

- Trường hợp 1: Các trạng thái hoàn toàn phân biệt => Không cần đánh dấu
  - Các bài toán BFS nhị phân
- Trường hợp 2: Các trạng thái có thể đánh dấu bằng mảng (một chiều hoặc hai chiều).
  - Ví dụ: BFS trên đồ thị
- Trường hợp 3: Các trạng thái phức tạp hoặc cần tìm kiếm nhanh => Sử dụng mã hóa + đánh dấu bằng set hoặc map.

QUESTIONS & ANSWERS