**Bài 9: Queue - Khái niệm cơ bản - Sử dụng thư viện chuẩn STL cho C/C++**

**Đăng bởi: Admin | Lượt xem: 4658 | Chuyên mục: C/C++**

Queue - Hàng đợi là  một cấu trúc dữ liệu dùng để lưu giữ các đối tượng theo cơ chế **FIFO** (viết tắt từ tiếng Anh: *First In First Out*), nghĩa là “vào trước ra trước”.

A picture containing text, transport, car

Description automatically generated

Khác với ngăn xếp, hàng đợi là mở ở cả hai đầu. Một đầu luôn luôn được sử dụng để chèn dữ liệu vào (hay còn gọi là sắp vào hàng) và đầu kia được sử dụng để xóa dữ liệu (rời hàng). Cấu trúc dữ liệu hàng đợi tuân theo phương pháp First-In-First-Out, tức là dữ liệu được nhập vào đầu tiên sẽ được truy cập đầu tiên.

Trong cấu trúc hàng đợi(queue), ta chỉ có thể thêm các phần tử vào một đầu của queue(giả sử là cuối), và cũng chỉ có thể xóa phần tử ở đầu còn lại của queue(tạm gọi là đầu). Như vậy, ở một đầu không thể xảy ra hai hành động thêm và xóa đồng thời.

Trong đời sống thực chúng ta có rất nhiều ví dụ về hàng đợi, chẳng hạn như hàng xe ô tô trên đường một chiều (đặc biệt là khi tắc xe), trong đó xe nào vào đầu tiên sẽ thoát ra đầu tiên. Một vài ví dụ khác là xếp hàng học sinh, xếp hàng mua vé, …

**I . Các hoạt động cơ bản trên cấu trúc dữ liệu hàng đợi :**

 hoạt động trên cấu trúc dữ liệu hàng đợi có thể liên quan tới việc khởi tạo hàng đợi, sử dụng dữ liệu trên hàng đợi và sau đó là xóa dữ liệu khỏi bộ nhớ.

Để cài đặt được Queue, chúng ta sẽ cần sử dụng các biến như sau:

* *queue[]*: Một mảng một chiều mô phỏng cho hàng đợi
* *arraySize:* Số lượng phần tử tối đa có thể lưu trữ trong queue[]
* *front:* Chỉ số của phần tử ở đang đầu queue. Nó sẽ là chỉ số của phần tử sẽ bị xóa ở lần tiếp theo
* *rear:*Chỉ số của phần tử tiếp theo sẽ được thêm vào cuối của queue

Danh sách dưới đây là một số hoạt động cơ bản có thể thực hiện trên cấu trúc dữ liệu hàng đợi:

**1 . Enqueue – Thêm vào cuối hàng đợi :**

Nếu hàng đợi chưa đầy, chúng ta sẽ thêm phần tử cần thêm vào cuối(*rear*) của hàng đợi. Ngược lại, thông báo lỗi.

Các bạn lưu ý, *rear*là chỉ số của phần tử sẽ được thêm ở lần tiếp theo. Do đó, thêm xong rồi ta mới tăng *rear*lên 1 đơn vị. Giá trị *rear*cần thay đổi nên được truyền theo tham chiếu.

**void** **Enqueue**(**int** queue[], **int** element, **int**& rear, **int** arraySize) {

**if**(rear == arraySize) // Queue is full

printf("OverFlow\n");

**else**{

queue[rear] = element; // Add the element to the back

rear++;

}

}

**2. Dequeue – Xóa khỏi đầu hàng đợi :**

Nếu hàng đợi có ít nhất 1 phần tử, chúng ta sẽ tiến hành xóa bỏ phần tử ở đầu của hàng đợi bằng cách tăng front lên 1 giá trị.

Ở đây, *front*đang là chỉ số của phần tử sẽ bị xóa rồi. Cho nên chỉ cần tăng là xong, đó cũng là lý do ta cần truyền tham số *front*sử dụng tham chiếu.

**void** **Dequeue**(**int** queue[], **int**& front, **int** rear) {

**if**(front == rear) // Queue is empty

printf("UnderFlow\n");

**else** {

queue[front] = 0; // Delete the front element

front++;

}

}

Tới đây chắc nhiều bạn thắc mắc tại sao lại tăng *front*mà không phải là giảm. Các bạn xem giải thích ở hình sau nhé:

A picture containing diagram

Description automatically generated

**3. Front – Lấy giá trị ở đầu hàng đợi :**

Hàm này sẽ lấy và trả về giá trị của phần tử đang ở đầu hàng đợi

**int** **Front**(**int** queue[], **int** front) {

**return** queue[front];

}

**4. Các hàm hỗ trợ khác :**

Hàm lấy kích thước của Hàng đợi, nói cách khác là số lượng phần tử đang có trên hàng đợi

**int** **Size**(**int** front, **int** rear) {

**return** (rear - front);

}

//Hàm kiểm tra queue rỗng

**bool** **IsEmpty**(**int** front, **int** rear) {

**return** (front == rear);

}

A screenshot of a computer

Description automatically generated with medium confidence

**II . Hoạt động enqueue trong cấu trúc dữ liệu hàng đợi**

Bởi vì cấu trúc dữ liệu hàng đợi duy trì hai con trỏ dữ liệu: front và rear, do đó các hoạt động của loại cấu trúc dữ liệu này là khá phức tạp khi so sánh với cấu trúc dữ liệu ngăn xếp.

Dưới đây là các bước để enqueue (chèn) dữ liệu vào trong hàng đợi:

* Kiểm tra xem hàng đợi là có đầy không.
* Nếu hàng đợi là đầy, tiến trình bị lỗi và bị thoát.
* Nếu hàng đợi không đầy, tăng con trỏ rear để trỏ tới vị trí bộ nhớ trống tiếp theo.
* Thêm phần tử dữ liệu vào vị trí con trỏ rear đang trỏ tới trong hàng đợi.
* Trả về success.

### Bài 10: Queue - Bài tập cơ bản - Sử dụng thư viện chuẩn STL cho C/C++

**Đăng bởi: Admin | Lượt xem: 2788 | Chuyên mục: C/C++**

## I . Ứng dụng của hàng đợi :

Để thấy được vai trò của cấu trúc dữ liệu *queue*cũng như hiểu hơn về nó. Chúng ta sẽ đi vào một bài toán cụ thể như sau:

Bạn có một chuỗi ký tự. Hãy lấy ký tự ở đầu của chuỗi và thêm nó vào cuối chuỗi. Hãy cho tôi thấy sự thay đỗi của chuỗi sau khi thực hiện hành động trên N lần.

**Ý tưởng**: Chuỗi ký tự trên có thể xem xét như là một hàng đợi. Tại mỗi bước, chúng ta sẽ dequeue(xóa) phần tử ở đầu chuỗi và thực hiện enqueue(thêm) phần tử đó cuối chuỗi. Lặp lại N lần bước công việc này, chúng ta sẽ có câu trả lời.

#**include** <iostream>

#**include** <cstdio>

**using** **namespace** std;

**void** **Enqueue**(**char** queue[], **char** element, **int**& rear, **int** arraySize) {

**if**(rear == arraySize) // Queue is full

printf("OverFlow\n");

**else** {

queue[rear] = element; // Add the element to the back

rear++;

}

}

**void** **Dequeue**(**char** queue[], **int**& front, **int** rear) {

**if**(front == rear) // Queue is empty

printf("UnderFlow\n");

**else** {

queue[front] = 0; // Delete the front element

front++;

}

}

**char** **Front**(**char** queue[], **int** front) {

**return** queue[front];

}

**int** **main**() {

**char** queue[20] = {'a', 'b', 'c', 'd'};

**int** front = 0, rear = 4;

**int** arraySize = 20; // Size of the array

**int** N = 3; // Number of steps

**char** ch;

**for**(**int** i = 0;i < N;++i) {

ch = Front(queue, front);

Enqueue(queue, ch, rear, arraySize);

Dequeue(queue, front, rear);

}

**for**(**int** i = front;i < rear;++i)

printf("%c", queue[i]);

printf("\n");

**return** 0;

}

## II . Các biến thể của Queue ;

##### 1 . Hàng đợi 2 đầu :

Trong hàng đợi tiêu chuẩn phía trên, dữ liệu được thêm ở cuối và xóa đi ở đầu của hàng đợi. Nhưng với*Double-ended*queue, dữ liệu có thể thêm hoặc xóa ở cả đầu(front) và cuối(rear) của hàng đợi.

Nào, hãy cùng tôi đi cài đặt các hàm cho các chức năng của queue 2 đầu nào.

**Thêm dữ liệu vào cuối queue:**

**void** **InsertAtBack**(**int** queue[], **int** element, **int** &rear, **int** array\_size){

**if**(rear == array\_size)

printf("Overflow\n");

**else**{

queue[rear] = element;

rear = rear + 1;

}

}

**Xóa dữ liệu ở cuối queue :**

**void** **DeleteFromBack**(**int** queue[], **int** &rear, **int** front){

**if**(front == rear)

printf("Underflow\n");

**else**{

rear = rear - 1;

queue[rear] = 0;

}

}

**Thêm dữ liệu vào đầu queue :**

**void** **InsertAtFront**(**int** queue[], **int** &rear, **int** &front, **int** element, **int** array\_size){

**if**(rear == array\_size)

printf("Overflow\n");

**else**{

**for**(**int** i=rear; i>front; i--)

queue[i] = queue[i-1];

queue[front] = element;

rear = rear+1;

}

}

**Xóa dữ liệu ở đầu queue :**

**void** **DeleteAtFront**(**int** queue[], **int** &front, **int** &rear){

**if**(front == rear)

printf("Underflow\n");

**else**{

queue[front] = 0;

front = front + 1;

}

}

**Lấy giá trị ở đầu queue :**

**int** **GetFront**(**int** queue[], **int** front){

**return** queue[front];

}

**Lấy giá trị ở cuối queue :**

**int** **GetRear**(**int** queue[], **int** rear){

**return** queue[rear-1];

}

Các hàm chức năng khác như **Size**và **IsEmpty**giống với hàng đợi tiêu chuẩn.

##### 2. Hàng đợi vòng :

Hàng đợi vòng là một cải tiến của hàng đợi tiêu chuẩn. Trong hàng đợi tiêu chuẩn, khi một phần tử bị xóa khỏi hàng đợi, các vị trí bị xóa đó sẽ không được tái sử dụng. Hàng đợi vòng sinh ra để khắc phục sự lãng phí này.

Timeline

Description automatically generated

Trong khi thêm phần tử vào hàng đợi vòng, nếu chỉ số *rear* đã ở vị trí cuối của mảng, khi đó bạn vẫn có thể thêm vào hàng đợi bằng cách thêm vào phía đầu của mảng(đó là các vị trí ở đầu mảng đã bị xóa đi và chưa được dùng).

Để cài đặt hàng đợi vòng, ngoài các biến sử dụng trong hàng đợi tiêu chuẩn, chúng ta cần thêm một biến khác nữa để lưu số lượng phần tử đang có trong hàng đợi, đặt nó là *count*

Bây giờ chúng ta cùng đi viết các hàm cho hàng đợi vòng nhé.

###### Enqueue :

**void** **Enqueue**(**int** queue[], **int** element, **int**& rear, **int** arraySize, **int**& count) {

**if**(count == arraySize) // Queue is full

printf(“OverFlow\n”);

**else**{

queue[rear] = element;

rear = (rear + 1)%arraySize;

count = count + 1;

}

}

Mình giải thích tại sao lại là *(rear + 1)%arraySize* : Giả sử khi*rear = arraySize - 1*, khi đó với hàng đợi tiêu chuẩn bạn sẽ không thể thêm nữa. Nhưng với hàng đợi vòng, ta sẽ thêm chừng nào*count != arraySize. arraySize* là số phần tử tối đa có thể có, do vậy, *count != arraySize* nghĩa là còn ô trống để insert vào *queue*. Chỉ số được insert vào *queue*như sau(giả sử*arraySize = 3* nhé):

* Nếu *rear*= 0, giá trị *rear*thực là (0 + 1) % 3 = 1
* Nếu *rear*= 1, giá trị *rear*thực là (1 + 1) % 3 = 2
* Nếu *rear*= 2, giá trị *rear*thực là (2 + 1) % 3 = 0

Các bạn nên nhớ: *rear*là chỉ số của phần tử sẽ được insert ở lần tiếp theo. Mỗi khi ***dequeue***, *count*sẽ phải giảm đi 1. Ngược lại, nếu ***enqueue***thành công, count sẽ tăng lên 1.

Giải thích này cũng đúng với *front*trong hàm **dequeue**dưới đây.

###### Dequeue

**void** **Dequeue**(**int** queue[], **int**& front, **int** rear, **int**& count) {

**if**(count == 0) // Queue is empty

printf(“UnderFlow\n”);

**else** {

queue[front] = 0; // Delete the front element

front = (front + 1)%arraySize;

count = count - 1;

}

}

###### Front

**int** **Front**(**int** queue[], **int** front) {

**return** queue[front];

}

###### Size :

**int** **Size**(**int** count) {

**return** count;

}

###### IsEmpty :

**bool** **IsEmpty**(**int** count) {

**return** (count == 0);

}