Bài 9: Stack - Queue

Nguyễn Hoàng Anh

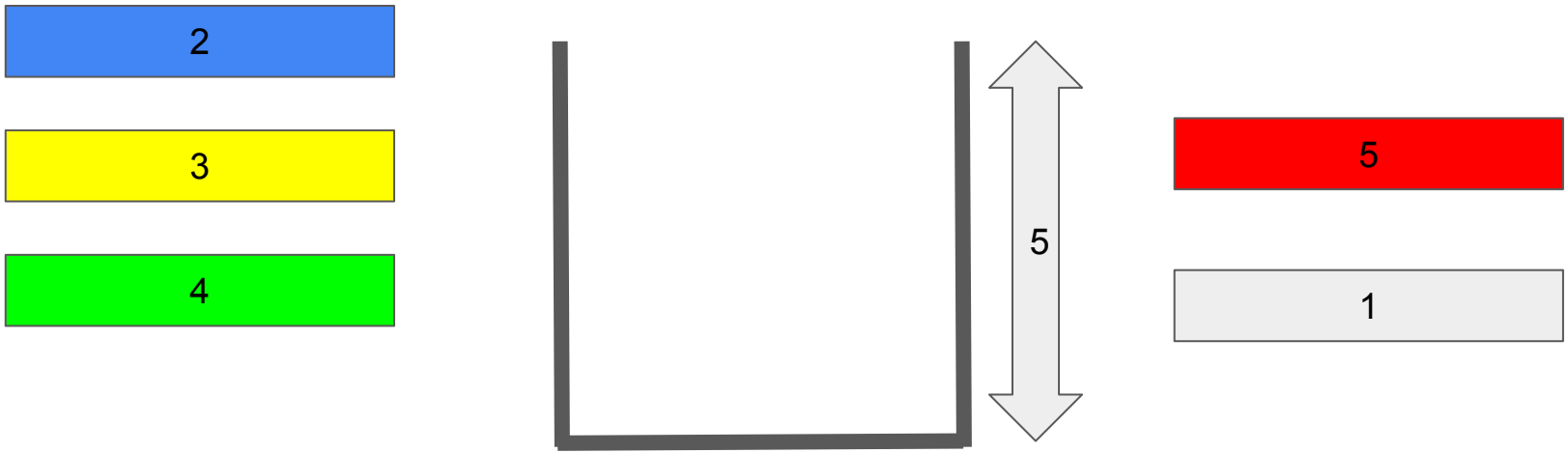
Cấu trúc dữ liệu

Cấu trúc dữ liệu là cách **tổ chức**, và **lưu trữ** dữ liệu để chúng có thể được truy cập và **sử dụng** một cách hiệu quả, đóng vai trò quan trọng trong việc giải quyết các bài toán và tối ưu hóa thuật toán, vì nó ảnh hưởng trực tiếp đến tốc độ thực thi và tính phức tạp của chương trình.

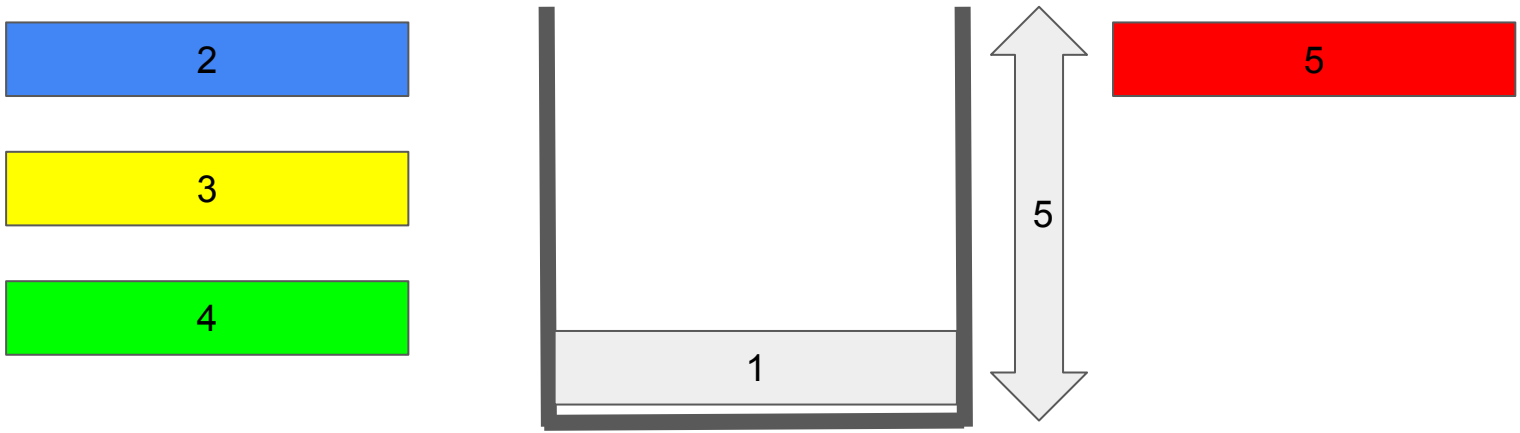
Cấu trúc dữ liệu được phân làm 2 loại chính:

- Cấu trúc dữ liệu tuyến tính (Linear Data Structure): **mảng** (Array), **ngăn xếp** (Stack), **hàng đợi** (Queue), **danh sách liên kết** (Linked List).
- Cấu trúc dữ liệu phi tuyến tính (Non-linear Data Structure): đồ thị (Graphs), **cây** (Trees).

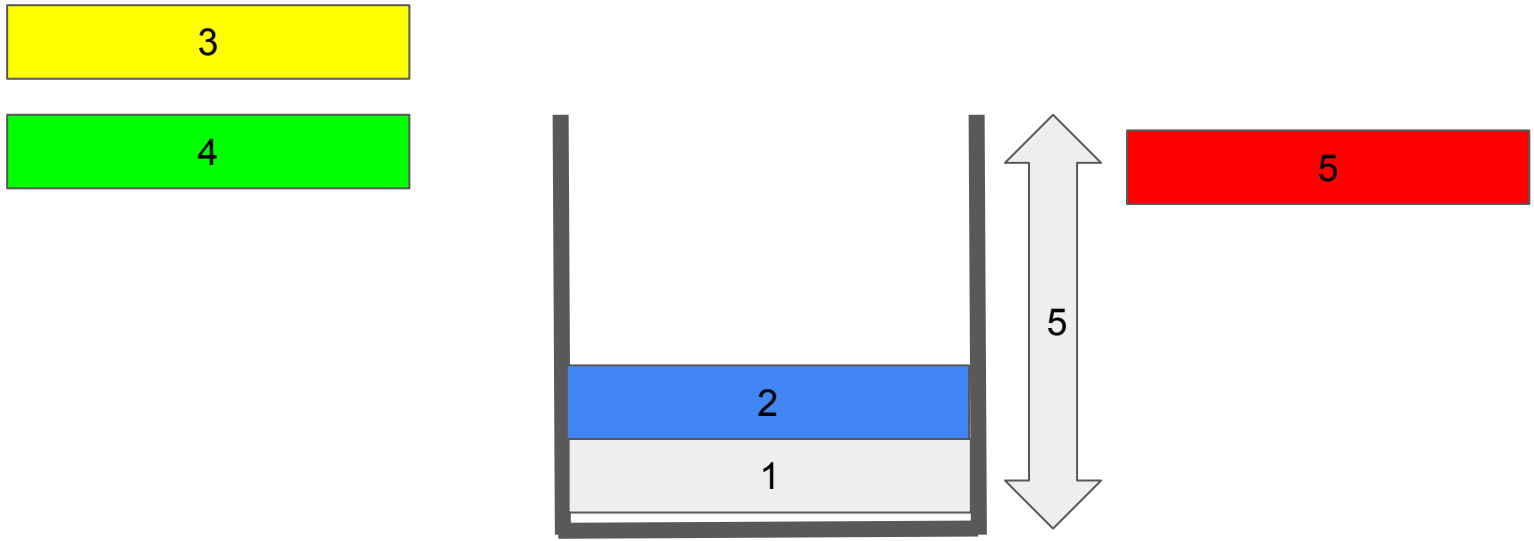
Stack



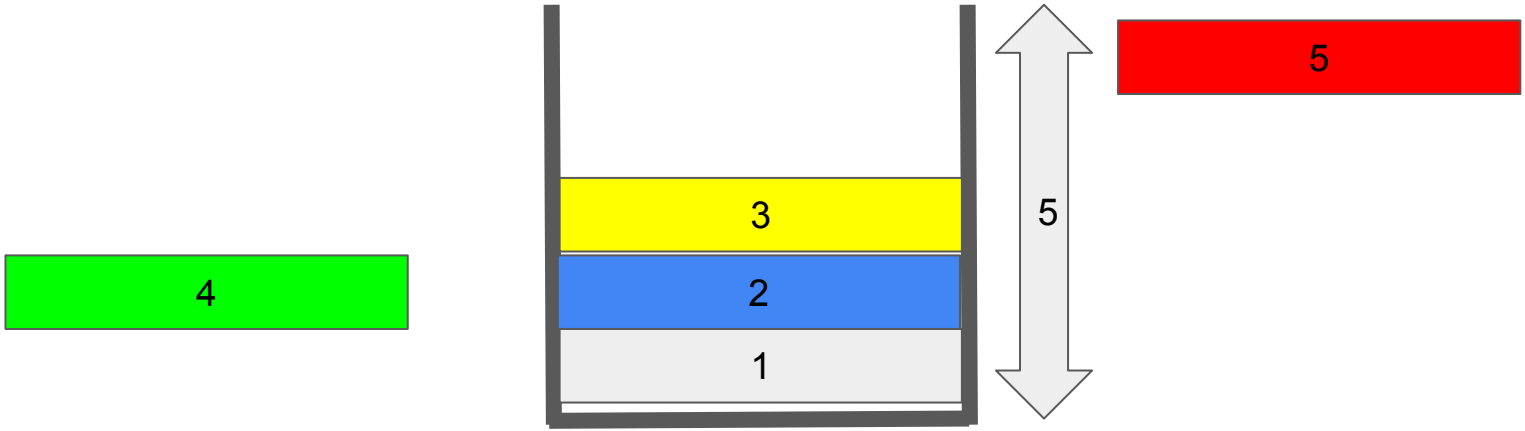
Stack



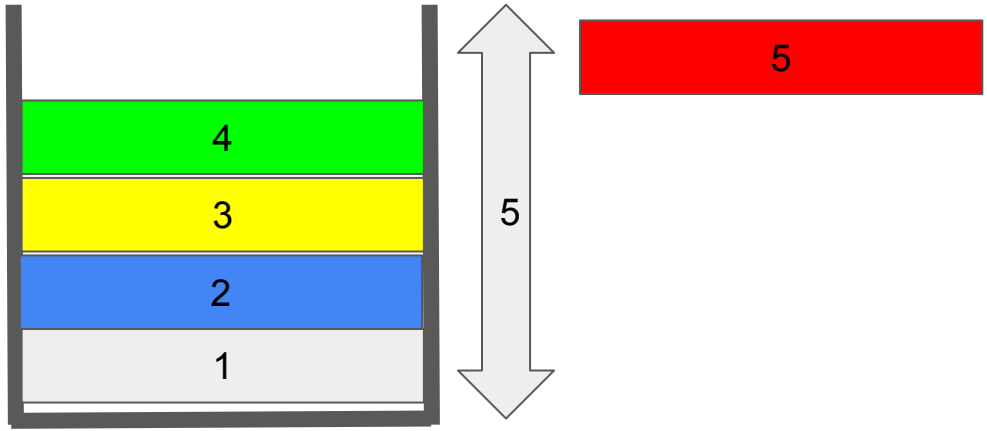
Stack



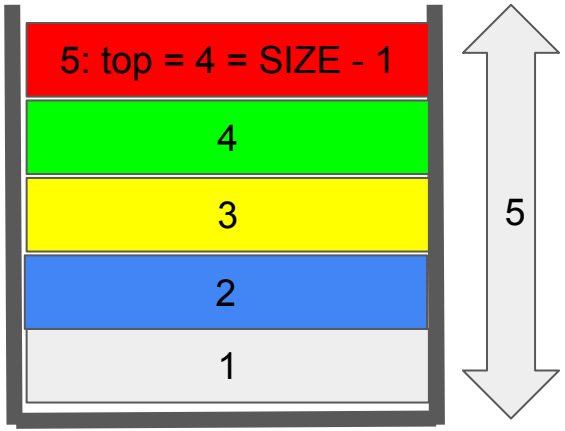
Stack



Stack



Stack



Stack

Stack (ngăn xếp) là một cấu trúc dữ liệu tuân theo nguyên tắc "**Last In**, **First Out**" (LIFO), nghĩa là phần tử cuối cùng được thêm vào stack sẽ là phần tử đầu tiên được lấy ra.

Các thao tác cơ bản trên stack bao gồm:

- **"push"** để thêm một phần tử vào **đỉnh** của stack
- **"pop"** để xóa một phần tử ở **đỉnh** stack.
- "**peek/top**" để lấy giá trị của phần tử ở **đỉnh stack**.
- Kiểm tra Stack đầy: top = size - 1
- Kiểm tra Stack rỗng: top = -1

push → top++

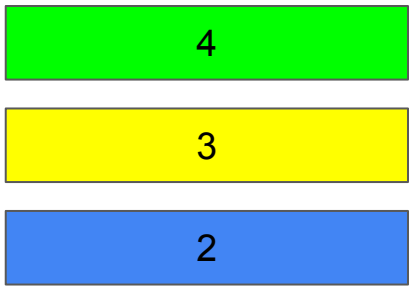
pop → top--

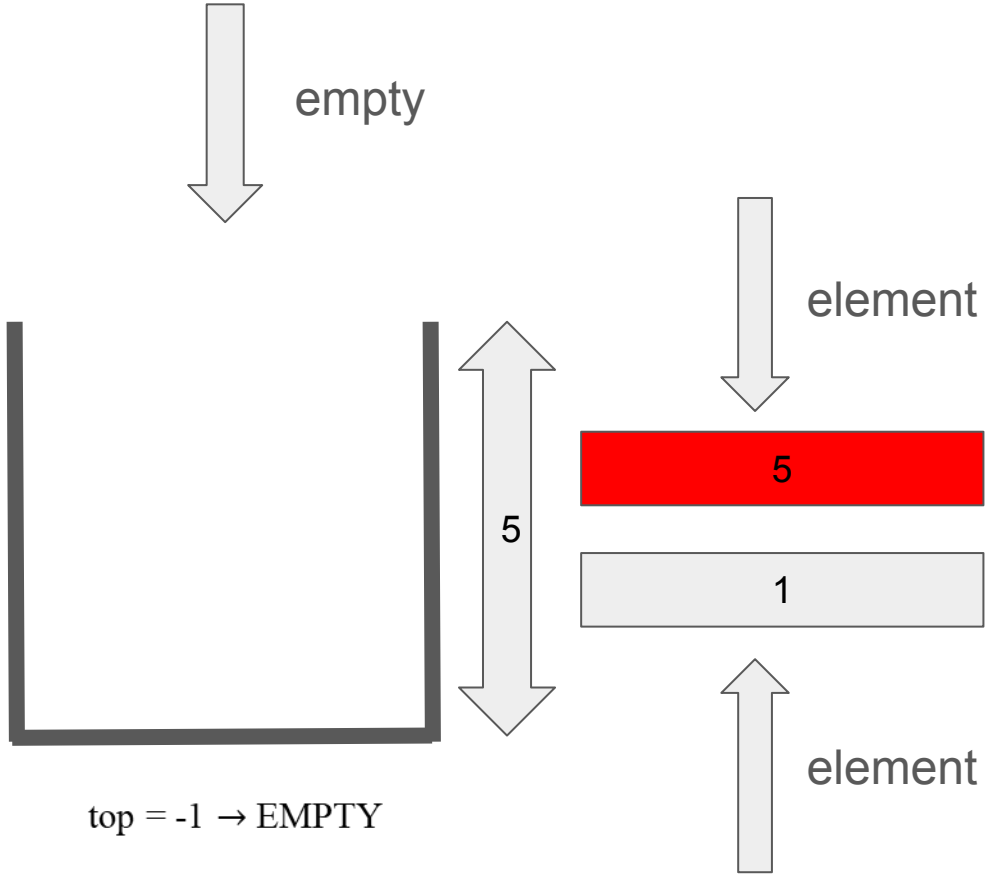
top (max) = size – 1 -> FULL

top = -1 -> EMPTY

Stack

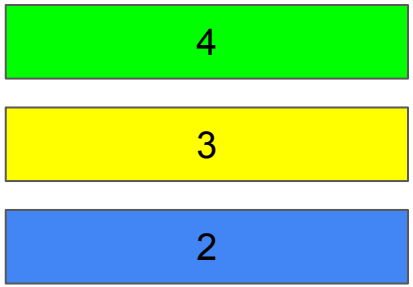
push element 1

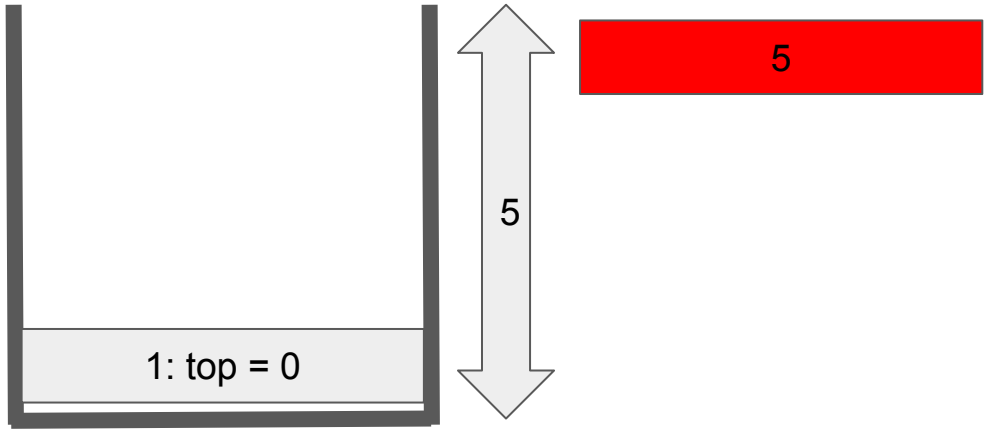




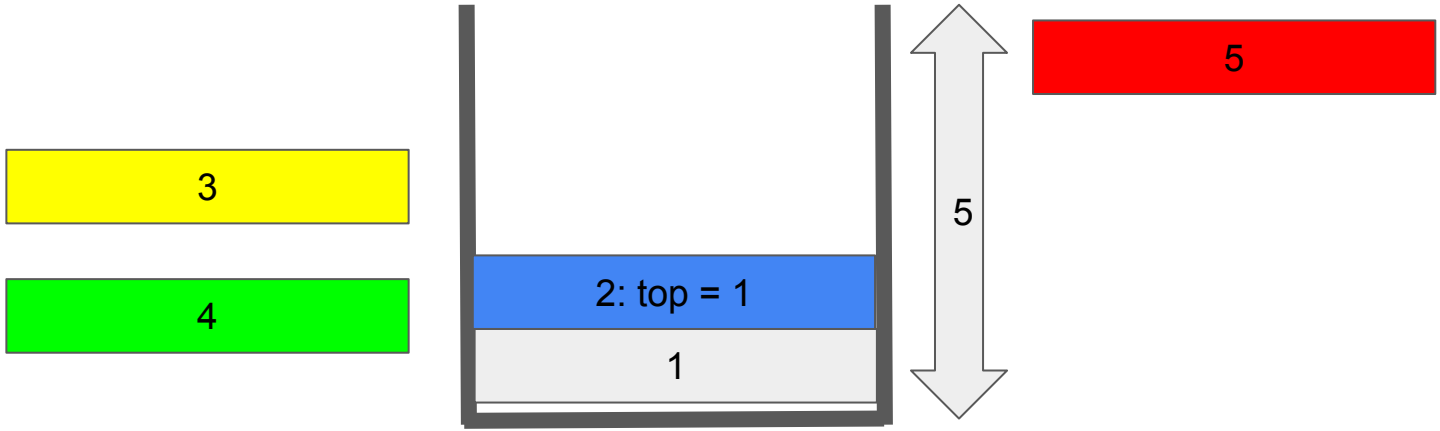
Stack

push element 2



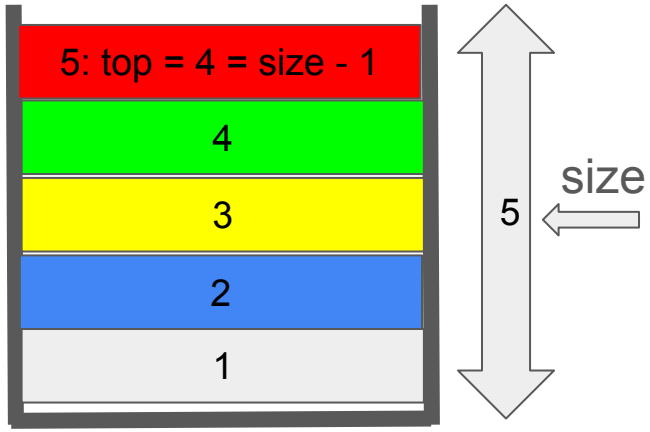


Stack



Stack

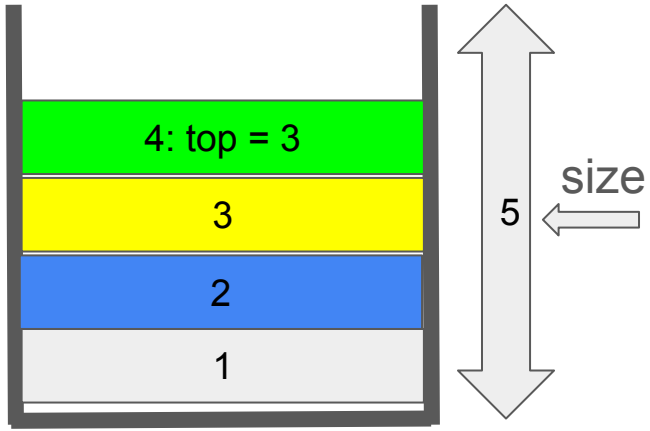
push element 5



top = size – 1 → FULL

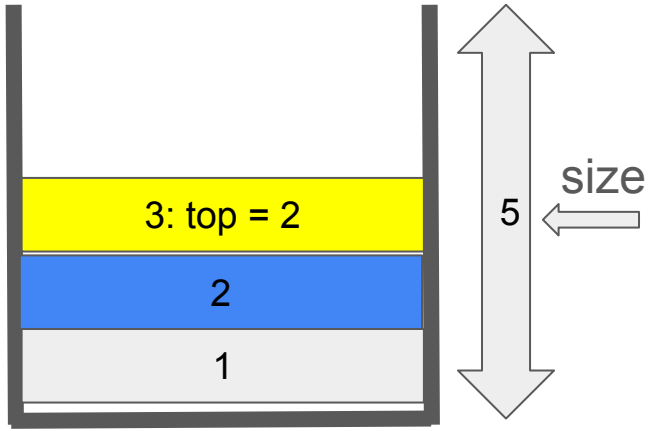
Stack

pop element 5



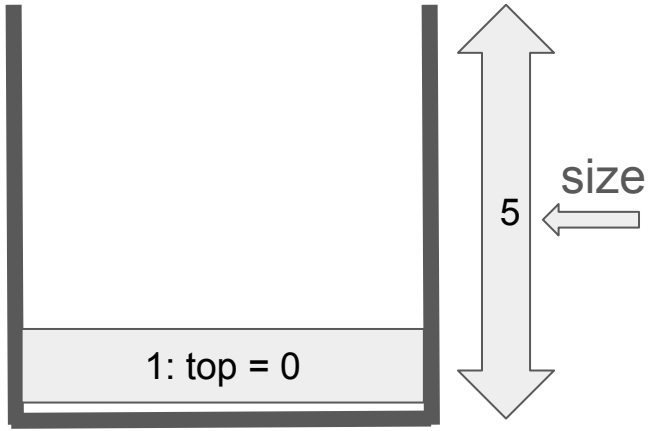
Stack

pop element 4



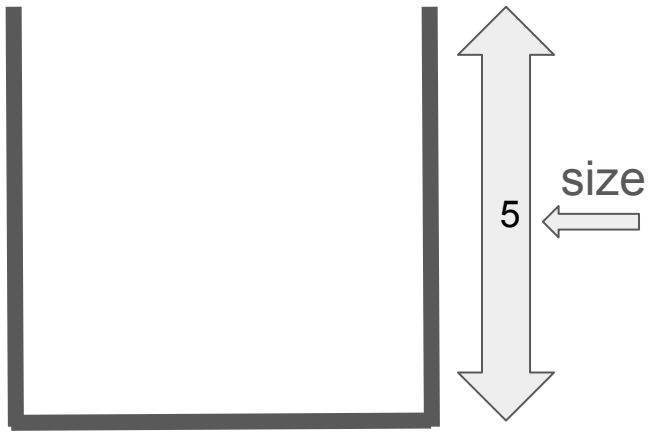
Stack

pop element 2



Stack

pop element 1



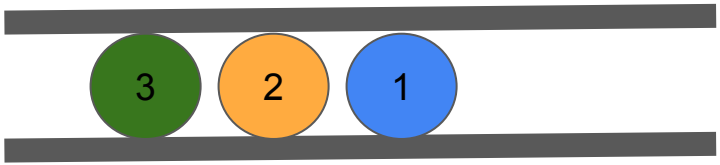
top = -1

Stack

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

typedef struct
{
    int *item;  // mảng lưu trữ giá trị các phần tử
    int size;   // số lượng phần tử tối đa của Stack
    int top;    // chỉ số lấy giá trị ở đỉnh Stack
} Stack;


#define STACK_EMPTY -1

// khởi tạo stack
void init(Stack *stack, int newSize)
{
    stack->size = newSize;
```

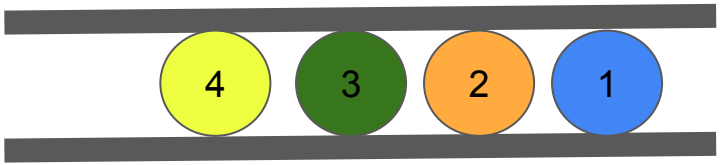
Queue

Queue

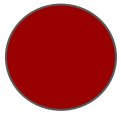


bán
cf

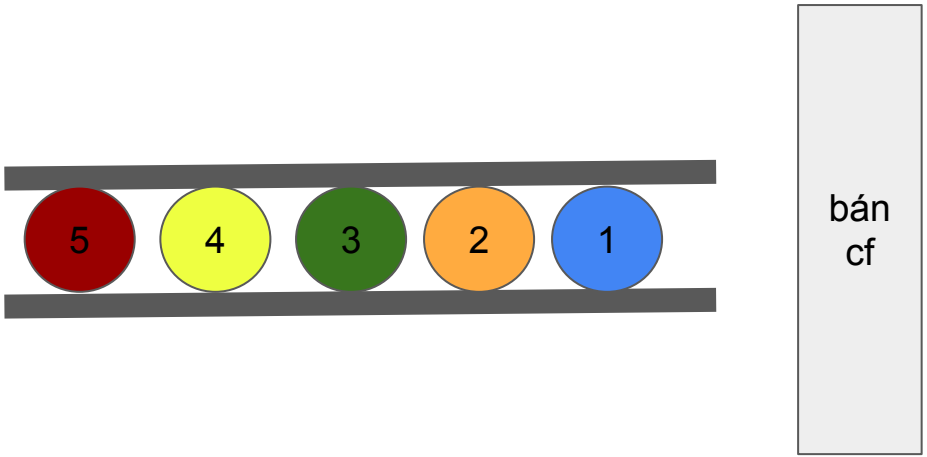
Queue



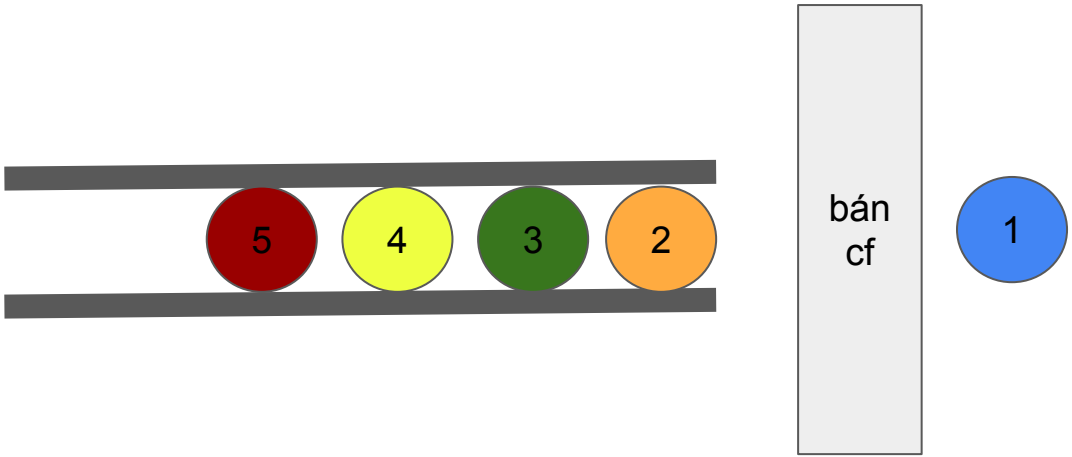
bán
cf



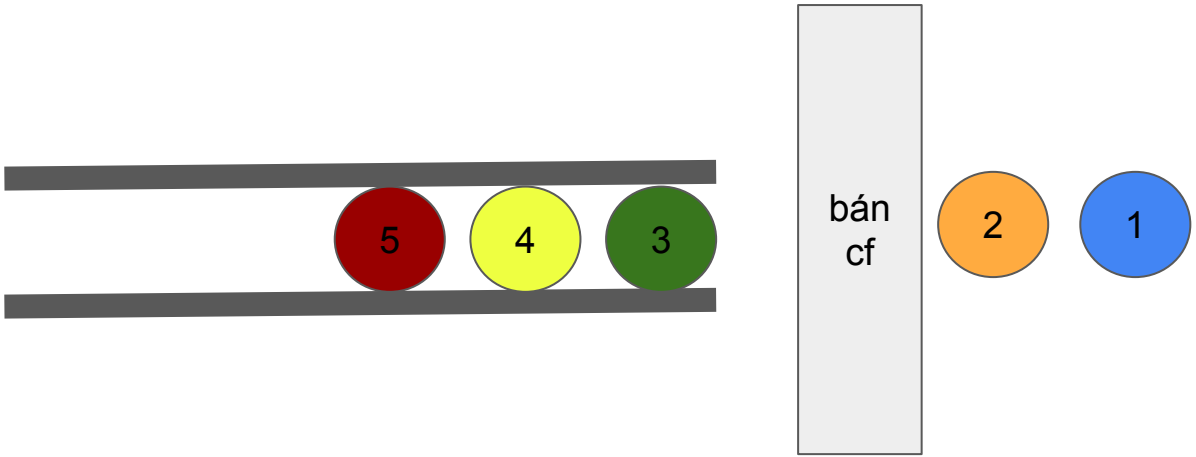
Queue



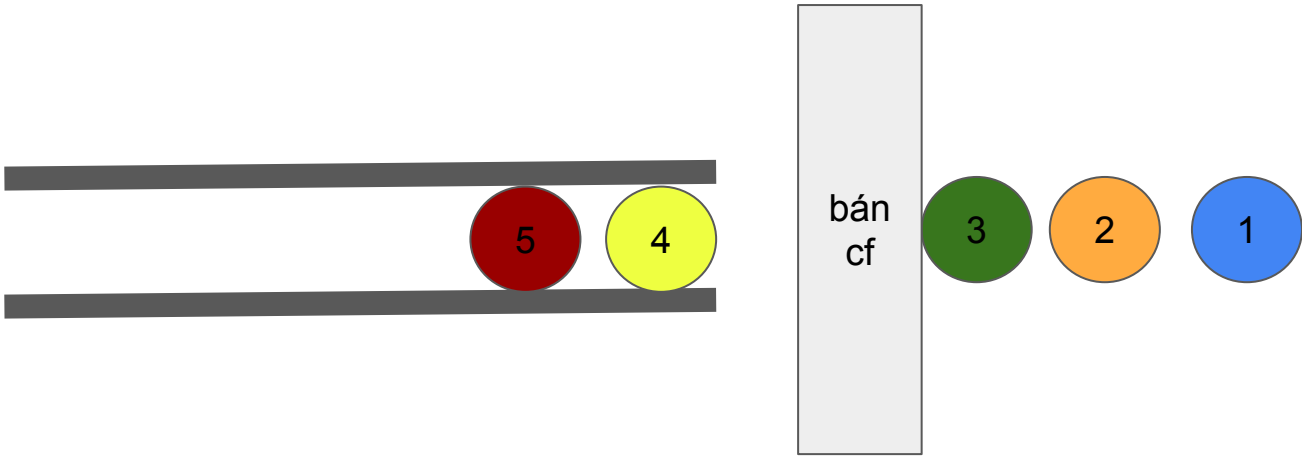
Queue



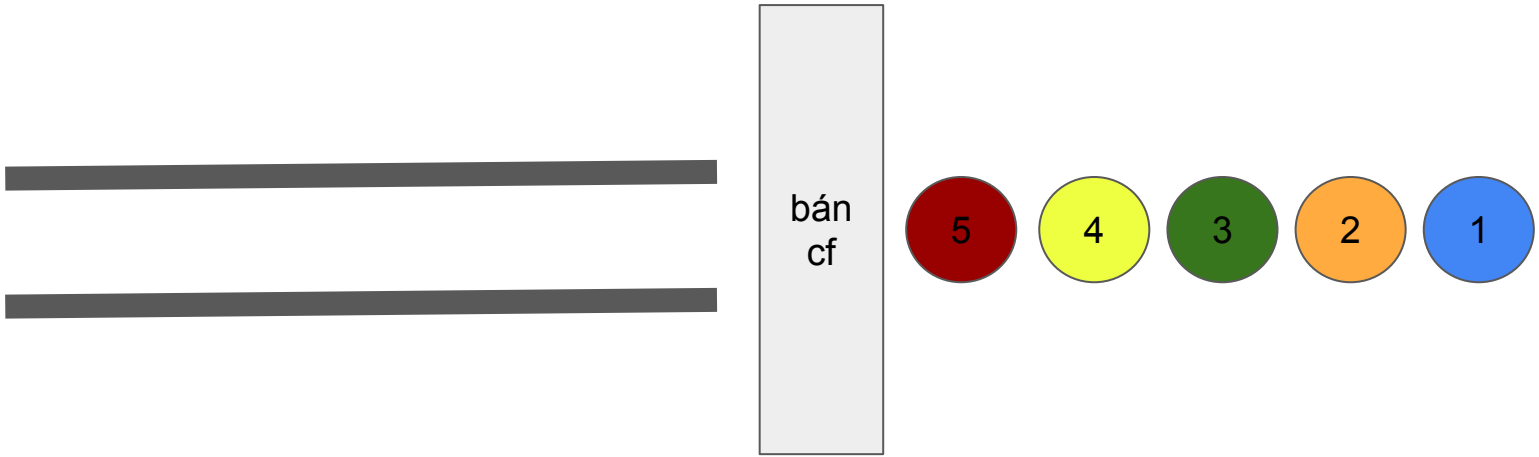
Queue



Queue



Queue



Queue

Queue là một cấu trúc dữ liệu tuân theo nguyên tắc "**First In, First Out**" (FIFO), nghĩa là phần tử đầu tiên được thêm vào hàng đợi sẽ là phần tử đầu tiên được lấy ra.

Các thao tác cơ bản trên hàng đợi bao gồm:

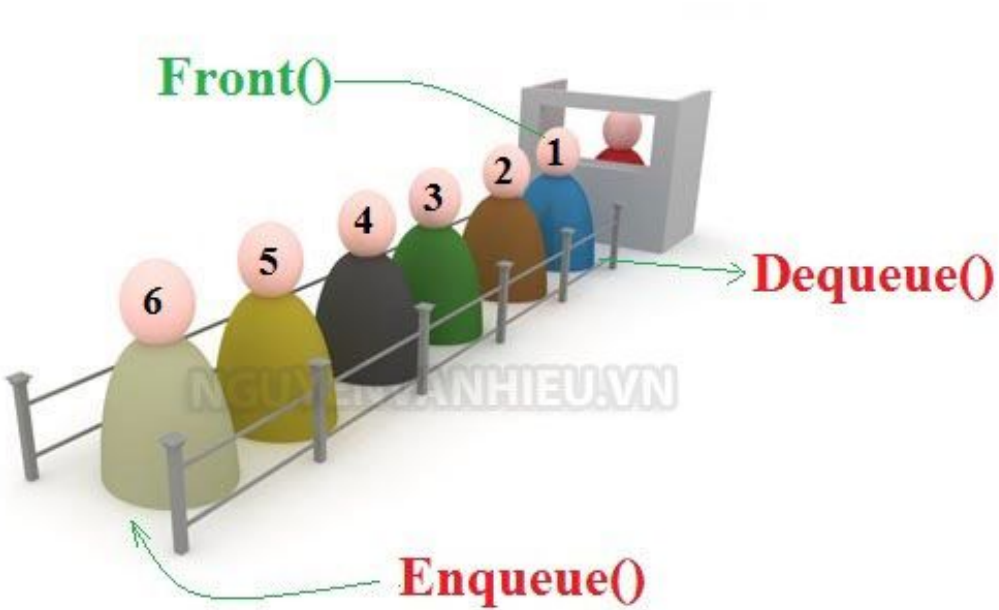
- "enqueue" (thêm phần tử vào **cuối** hàng đợi)
- "dequeue" (lấy phần tử từ **đầu** hàng đợi).
- "front" để lấy giá trị của phần tử đứng đầu hàng đợi.
- "rear" để lấy giá trị của phần tử đứng cuối hàng đợi.
- Kiểm tra hàng đợi đầy/rỗng.

Linear Queue

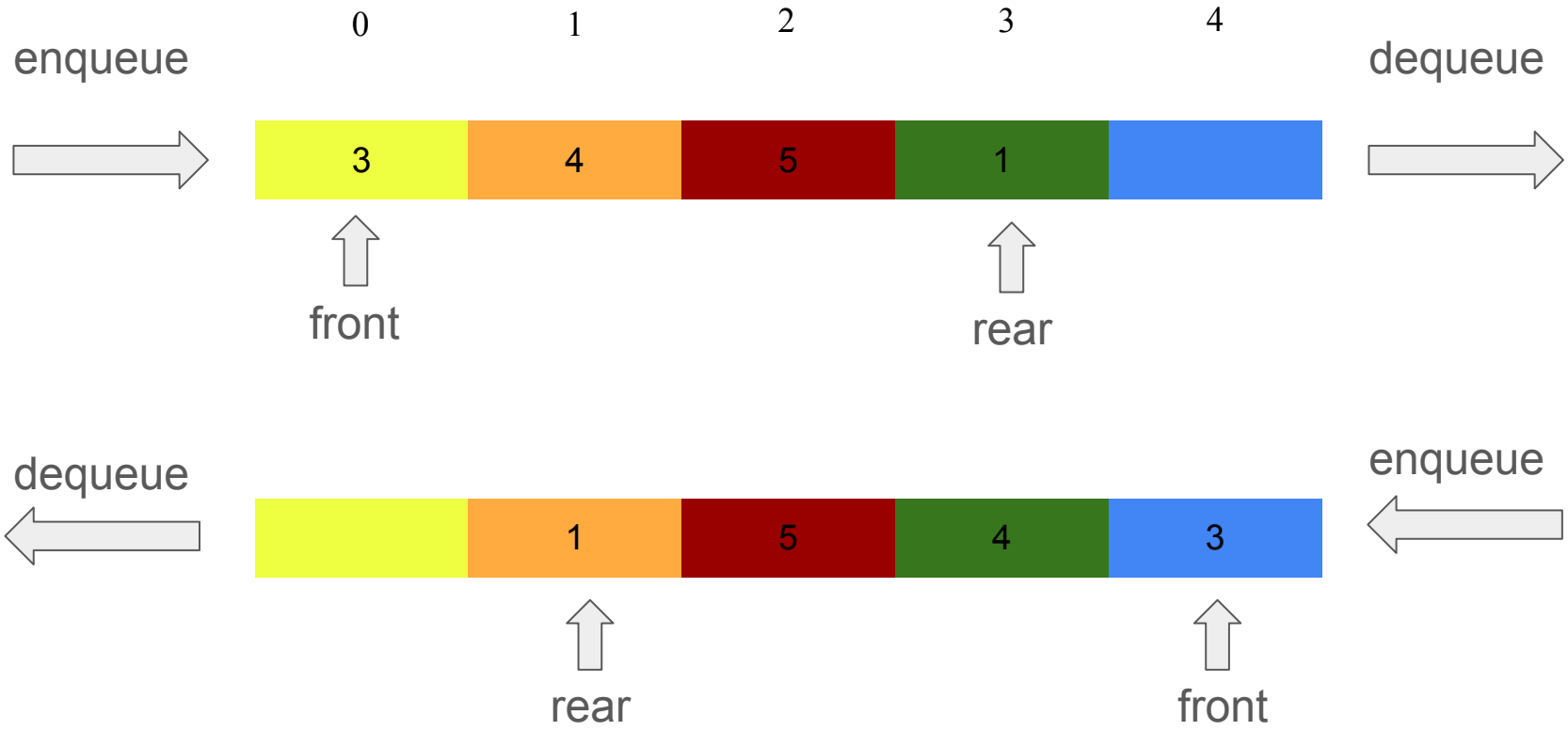
Circular Queue

Priority Queue

Queue



Queue


0
1
2
3
4
enqueue
dequeue
3
4
5
1
front
rear
dequeue
enqueue
1
5
4
3
rear
front

Linear Queue

front = -1

rear = -1

Linear Queue

enqueue: 3

front = 0

rear = 0

Linear Queue

enqueue: 4

front = 0

rear = 1

Linear Queue

enqueue: 2

front = 0

rear = 2

Linear Queue

enqueue: 5

front = 0

rear = 3

Linear Queue

enqueue → rear++

rear = size – 1 → Queue Full

enqueue: 7

front = 0

rear = 4

Linear Queue

dequeue: 3

front = 1

rear = 4

Linear Queue

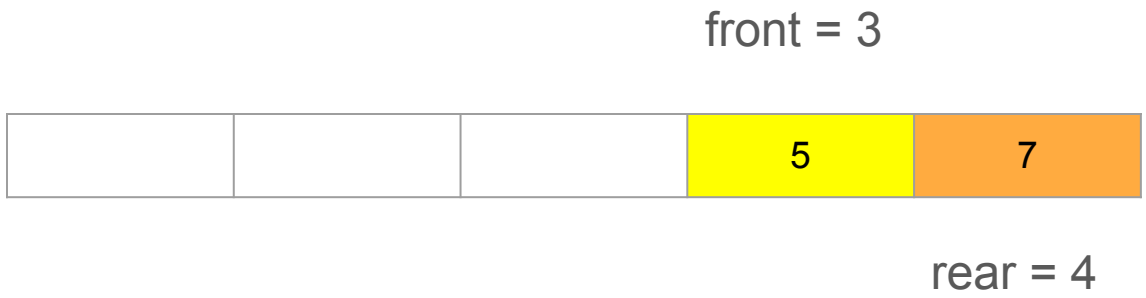
dequeue: 4

front = 2

rear = 4

Linear Queue

dequeue: 2



Linear Queue

dequeue: 5

front = 4

rear = 4

Linear Queue

dequeue → front ++

front == -1 hoặc front > rear → Queue Empty

dequeue: 7

front = 5

rear = 4

front = -1

rear = -1

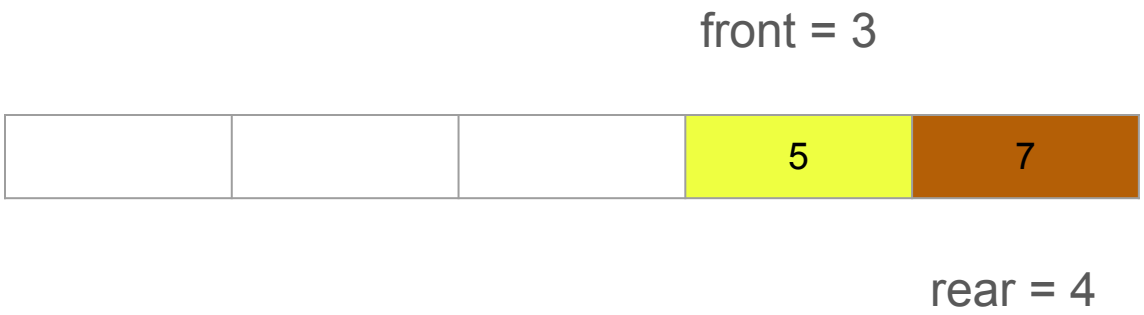
Linear Queue

- Trong Linear Queue, nếu ‘rear’ đã đạt tới max, thì queue sẽ được coi là đầy và không thể thêm phần tử mới, ngay cả khi phía trước còn khoảng trống do các phần tử đã bị xóa.
- Chỉ có thể thêm phần tử mới khi đã dequeue toàn bộ các phần tử hiện có (tức là queue rỗng hoàn toàn và front được reset về vị trí ban đầu).

Linear Queue

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct
{
    int *item;  // mảng lưu trữ giá trị các phần tử
    int size;   // số lượng phần tử tối đa có thể đưa vào
    int front;  // chỉ số của phần tử đầu hàng đợi
    int rear;   // chỉ số của phần tử cuối hàng đợi
} Queue;

// khởi tạo hàng đợi
void initialize(Queue *queue, int size)
{
```

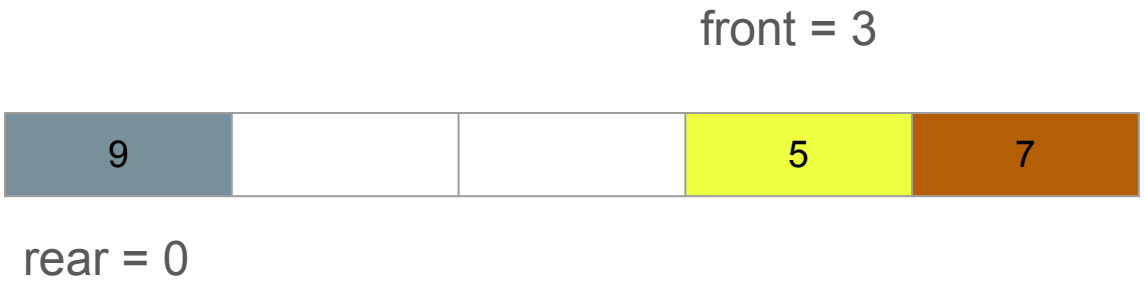
Circular Queue



Khi rear đạt tới size - 1 và không còn chỗ trống từ phía cuối, nếu front đã di chuyển (nghĩa là đã có các phần tử được dequeue), rear có thể "quay vòng" về vị trí 0 để tận dụng khoảng trống.

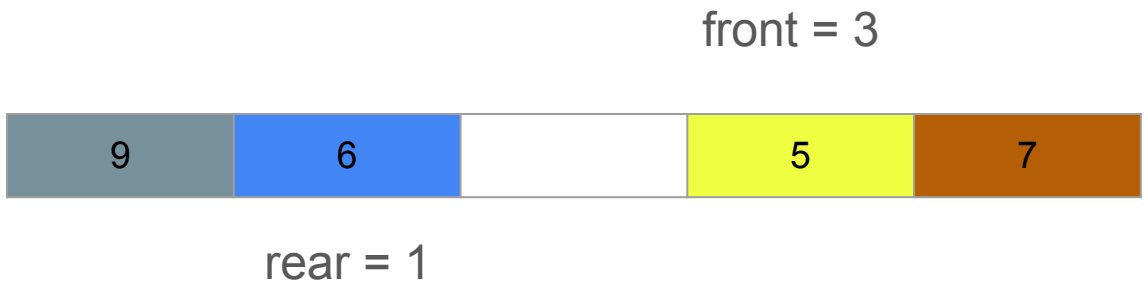
Circular Queue

enqueue: 9



Circular Queue

enqueue: 6

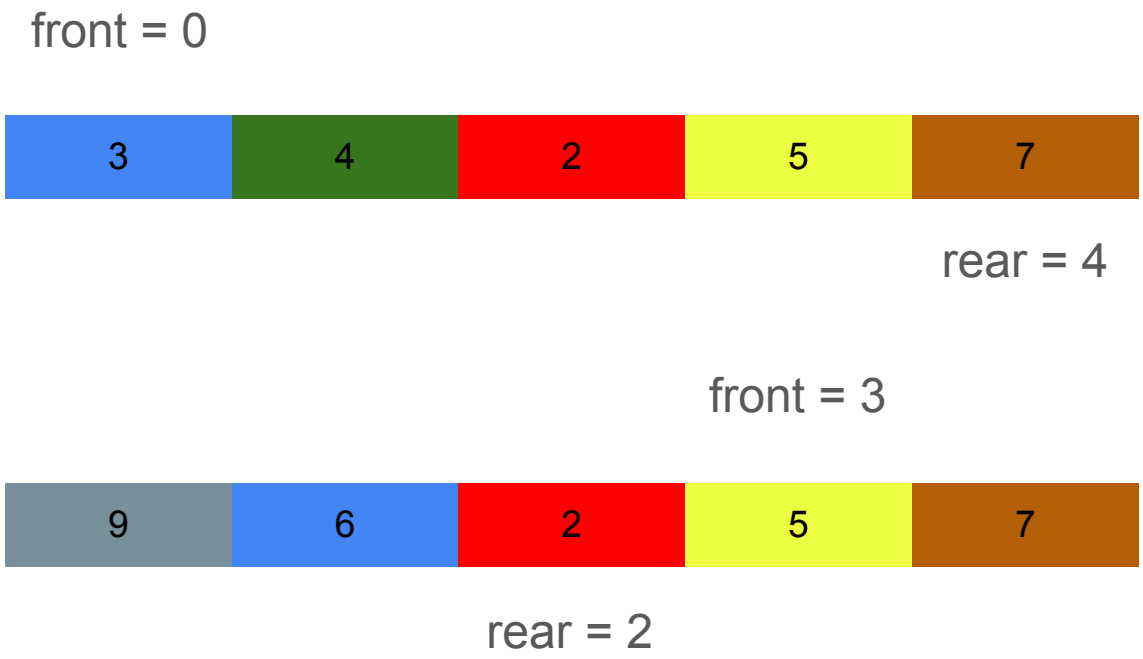


Circular Queue

enqueue: 2

Circular Queue

queue full: **front == (rear + 1) % SIZE**



Circular Queue

queue empty: **front == -1**

front = -1

rear = -1

Circular Queue

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct
{
    int *items; // mảng lưu trữ giá trị các phần tử
    int size;   // kích thước của hàng đợi
    int front;  // chỉ số phần tử đầu hàng đợi
    int rear;   // chỉ số phần tử cuối hàng đợi
} Queue;

// khởi tạo hàng đợi
void queue_Init(Queue *queue, int size)
{
    queue->items = (int*)malloc(size * sizeof(int));
```