Stack và Queue

DATA STRUCTURES AND ALGORITHMS

Các cấu trúc đặc biệt của danh sách đơn

- Stack (ngăn xếp): Là 1 vật chứa các đối tượng làm việc theo cơ chế LIFO (Last In First Out), tức việc thêm 1 đối tượng vào Stack hoặc lấy 1 đối tượng ra khỏi Stack được thực hiện theo cơ chế "vào sau ra trước"
- Queue (hàng đợi): Là 1 vật chứa các đối tượng làm việc theo cơ chế FIFO (First In First Out), tức việc thêm 1 đối tượng vào hàng đợi hay lấy 1 đối tượng ra khỏi hàng đợi thực hiện theo cơ chế "vào trước ra trước".

Ứng dụng Stack và Queue

- Stack:
  - Trình biên dịch
  - Khử đệ qui đuôi
  - Lưu vết các quá trình quay lui, vét cạn
- Queue:
  - Tổ chức lưu vết các quá trình tìm kiếm theo chiều rộng, và quay lui vét cạn
  - Tổ chức quản lý và phân phối tiến trình trong các hệ điều hành.
  - Tổ chức bộ đệm bàn phím, …

Các thao tác trên Stack

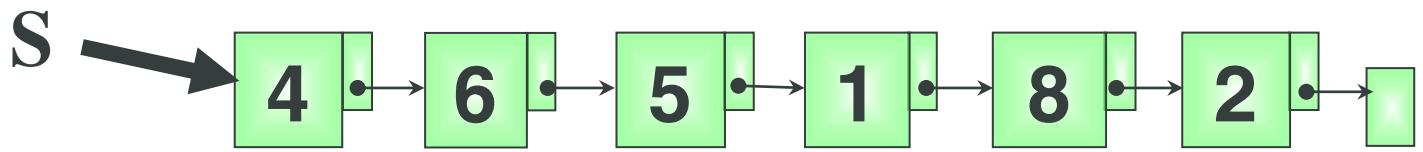
- Push(o): Thêm đối tượng o vào Stack
- Pop(): Lấy đối tượng từ Stack
- isEmpty(): Kiểm tra Stack có rỗng hay không
- isFull(): Kiểm tra Stack có đầy hay không
- Top(): Trả về giá trị của phần tử nằm đầu Stack mà không hủy nó khỏi Stack.

Cài đặt Stack

- Dùng mảng 1 chiều

Data S [N];
int t;

- Dùng danh sách liên kết đơn



List S

- Chú ý: Thêm và hủy cùng phía

Cài Stack bằng mảng 1 chiều

- Cấu trúc dữ liệu của Stack

```
struct Stack {
    int a[MAX];
    int t;
};
```

- Khởi tạo Stack

```
void CreateStack(Stack &s) {
    s.t = -1;
}
```

Kiểm tra tính rỗng và đầy của Stack

```
bool isEmpty(Stack s) { //Stack có rỗng hay không
    if (s.t == -1)
        return 1;
    return 0;
}

bool isFull(Stack s) { //Kiểm tra Stack có đầy hay không
    if (s.t >= MAX)
        return 1;
    return 0;
}
```

Thêm 1 phần tử vào Stack

```
bool Push(Stack &s, int x) {
    if (isFull(s) == 0)
        s.a[++s.t] = x;
        return 1;
    }
    return 0;
}
```

Lấy 1 phần tử từ Stack

```
int Pop(Stack &s, int &x) {
    if (isEmpty(s) == 0) {
        x = s.a[s.t--];
        return 1;
    }
    return 0;
}
```

Cài Stack bằng danh sách liên kết

- Kiểm tra tính rỗng của Stack

```
int isEmpty(LIST &s) {
    if (s.pHead == NULL)//Stack rong
          return 1;
    return 0;
}
```

Thêm 1 phần tử vào Stack

```
void Push(LIST &s, NODE *p) { // AddHead
    if (s.pHead == NULL) {
        s.pHead = p;
        s.pTail = p;
    }
    else {
        p->pNext = s.pHead;
        s.pHead = p;
    }
}
```

Lấy 1 phần tử từ Stack

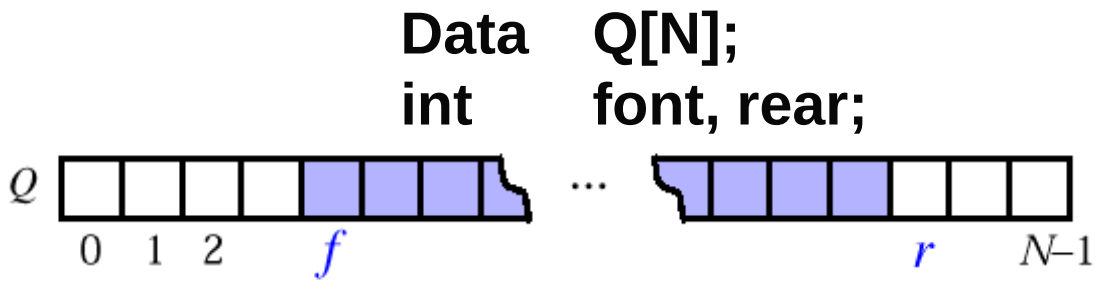
```
bool Pop(LIST &s, int &x) {
    NODE *p;
    if (isEmpty(s) != 1) {
        if (s.pHead != NULL) {
            p = s.pHead;
            x = p->info;
            s.pHead = s.pHead->pNext;
            if (s.pHead == NULL)
                s.pTail = NULL;
            return 1;
        }
    }
    return 0;
}
```

Các thao tác trên Queue

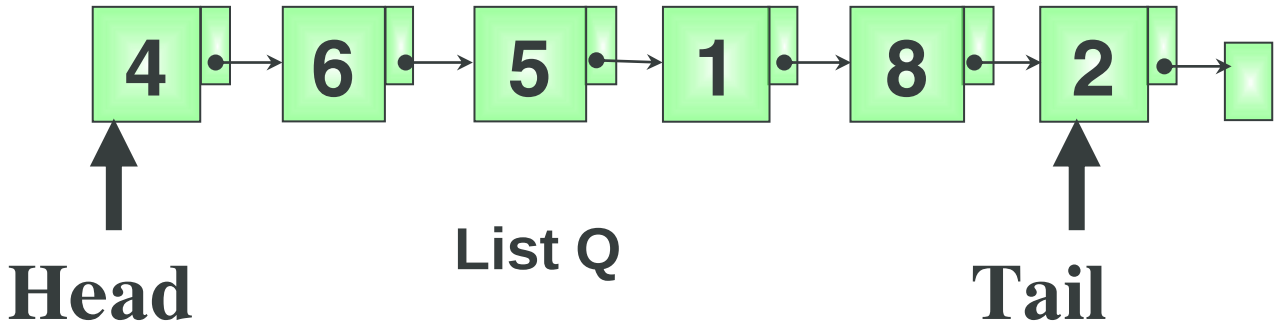
- EnQueue(O): Thêm đối tượng O vào cuối hàng đợi.
- DeQueue(): Lấy đối tượng ở đầu hàng đợi
- isEmpty(): Kiểm tra xem hàng đợi có rỗng hay không?
- Front(): Trả về giá trị của phần tử nằm đầu hàng đợi mà không hủy nó.

Cài đặt Queue

- Dùng mảng 1 chiều



- Dùng danh sách liên kết đơn



- Chú ý: **Thêm và hủy Khác phía**

Cài đặt Queue bằng mảng 1 chiều

- *Cấu trúc dữ liệu:*

```
typedef struct tagQueue {
    int a[MAX];
    int Front; //chỉ số của phần tử đầu trong Queue
    int Rear; //chỉ số của phần tử cuối trong Queue
} Queue;
```

- *Khởi tạo Queue rỗng*

```
void CreateQueue(Queue &q) {
    q.Front = -1;
    q.Rear = -1;
}
```

Kiểm tra tính rỗng và đầy của Queue

```
bool isEmpty(Queue q) { // Queue có rỗng?
  if (q.Front == -1)
     return 1;
  return 0;
}

bool isFull(Queue q) { // Kiểm tra Queue có đầy?
  if (q.Rear - q.Front + 1 == MAX)
     return 1;
  return 0;
}
```

Thêm 1 phần tử vào Queue

```
void EnQueue(Queue &q, int x) {
    int f, r;
    if (isFull(q)) //queue bị đầy => không thêm được nữa
        printf("queue day roi khong the them vao duoc nua");
    else {
        if (q.Front == -1) {
            q.Front = 0;
            q.Rear = -1;
        }
        if (q.Rear == MAX - 1) { //Queue đầy ảo
            f = q.Front;
            r = q.Rear;
            for (int i = f; i <= r; i++)
                q.a[i - f] = q.a[i];
            q.Front = 0;
            q.Rear = r - f;
        }
        q.Rear++;
        q.a[q.Rear] = x;
    }
}
```

Lấy 1 phần tử từ Queue

```
bool DeQueue(Queue &q, int &x) {
    if (isEmpty(q)==0) { //queue khong rong
        x = q.a[q.Front];
        q.Front++;
        if (q.Front>q.Rear) { //truong hop co mot phan tu
          q.Front = -1;
          q.Rear = -1;
        }
        return 1;
    }
    //queue trong
    printf("Queue rong");
    return 0;
}
```

Cài đặt Queue bằng List

- Kiểm tra Queue có rỗng?

```
bool isEmpty(LIST &Q) {
    if (Q.pHead == NULL) //Queue rỗng
          return 1;
    return 0;
}
```

Thêm 1 phần tử vào Queue

```
void EnQueue(LIST &Q, NODE *p) {
    if (Q.pHead == NULL) {
        Q.pHead = p;
        Q.pTail = p;
    }
    else {
        Q.pTail->pNext = p;
        Q.pTail = p;
    }
}
```

Lấy 1 phần tử từ Queue

```
int DeQueue(LIST &Q, int &x) {
    NODE *p;
    if (isEmpty(Q) != 1) {
        if (Q.pHead != NULL) {
            p = Q.pHead;
            x = p->info;
            Q.pHead = Q.pHead->pNext;
            if (Q.pHead == NULL)
                Q.pTail = NULL;
            return 1;
        }
    }
    return 0;
}
```

Chúc các em học tốt!