Cấu trúc dữ liệu và thuật toán

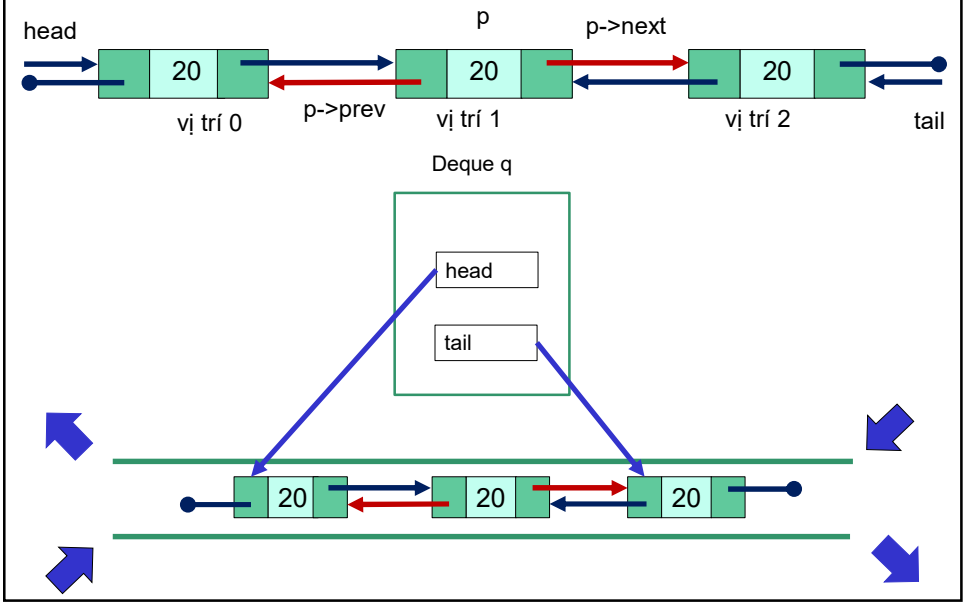
PGS. TS. Phạm Tuấn Minh
Trường Công nghệ Thông tin, Đại học Phenikaa
minh.phamtuan@phenikaa-uni.edu.vn
https://sites.google.com/site/phamtuanminh/

Chương 2: Mảng, danh sách liên kết, ngăn xếp, hàng đợi

Cấu trúc dữ liệu hàng đợi hai đầu

- Ví dụ lập lịch của bộ vi xử lý
- Hàng đợi hai đầu (double-ended queue, hay deque)
- Các thao tác trên hàng đợi hai đầu
  - createDeque(): Khởi tạo một hàng đợi hai đầu rỗng
  - isEmpty(q): Kiểm tra hàng đợi hai đầu q có rỗng không
  - printDeque(q): Liệt kê các phần tử trong hàng đợi hai đầu
  - pushFront(q, x): Thêm x vào đầu hàng đợi hai đầu
  - pushRear(q, x): Thêm x vào cuối hàng đợi hai đầu
  - popFront(q): Lấy phần tử ở đầu hàng đợi hai đầu
  - popRear(q): Lấy phần tử ở cuối hàng đợi hai đầu
  - peekFront(q): Xem phần tử ở đầu hàng đợi hai đầu
  - peekRear(q): Xem phần tử ở cuối hàng đợi hai đầu

Khai báo hàng đợi hai đầu



Khai báo hàng đợi hai đầu

```
typedef struct _dlistnode{
        int num;
        struct _dlistnode *next;
    struct _dlistnode *prev;
} DListNode;
typedef struct Deque {
   DListNode* head;
   DListNode* tail;
} Deque;
```

Tạo nút mới

```
DListNode* createNode(int data) {
   DListNode* newNode = (DListNode*)malloc(sizeof(DListNode));
   if (newNode == NULL) {
      printf("Memory allocation failed.\n");
      return NULL;
   }
   newNode->data = data;
   newNode->next = NULL;
   newNode->prev = NULL;
   return newNode;
}
```

Tạo hàng đợi hai đầu

```
Deque* createDeque() {
    Deque* deque = (Deque*)malloc(sizeof(Deque));
    if (deque == NULL) {
        printf("Memory allocation failed.\n");
        return NULL;
    }
    deque->head = NULL;
    deque->tail = NULL;
    return deque;
}
```

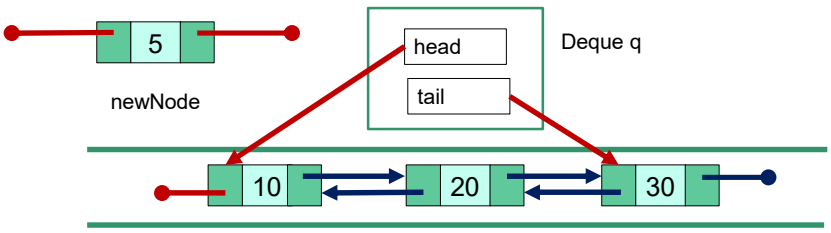
Kiểm tra có phần tử không

```
int isEmpty(Deque* deque) {
    return deque->head == NULL;
}
```

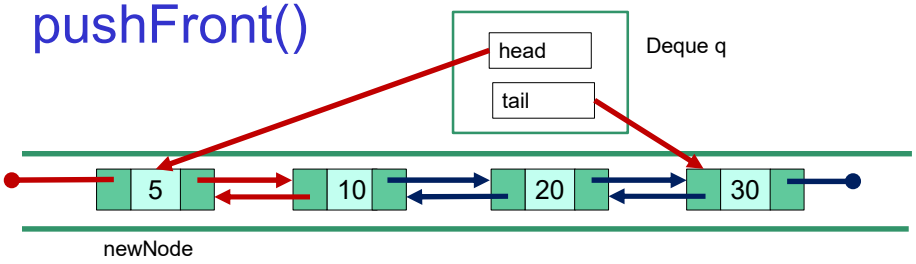
Liệt kê các phần tử

```
void printDeque(Deque* deque) {
    if (isEmpty(deque)) {
        printf("Deque is empty.\n");
        return;
    }
    DListNode* current = deque->head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

pushFront()

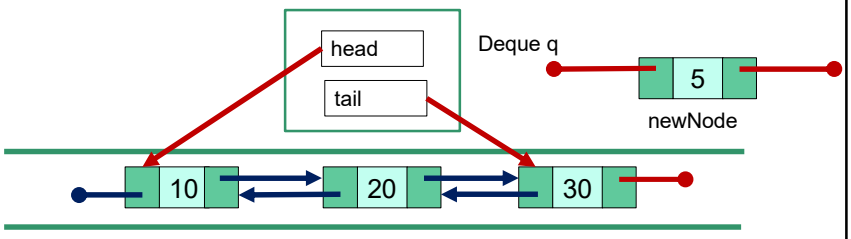


pushFront()

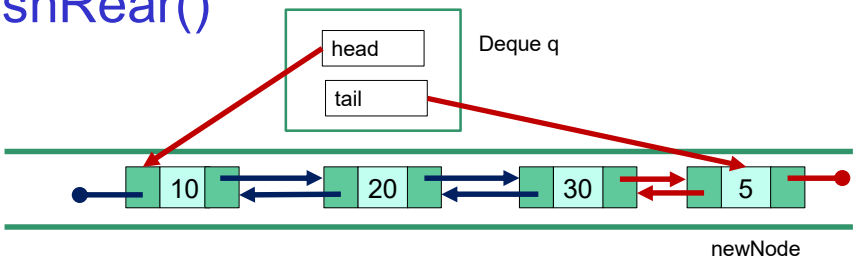



```
void pushFront(Deque* deque, int data) {
    DListNode* newNode = createNode(data);
    if (isEmpty(deque)) {
        deque->head = newNode;
        deque->tail = newNode;
    } else {
        newNode->next = deque->head;
        deque->head->prev = newNode;
        deque->head = newNode;
    }
}
```

pushRear()

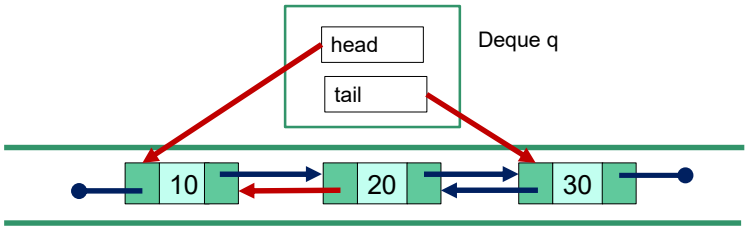


pushRear()

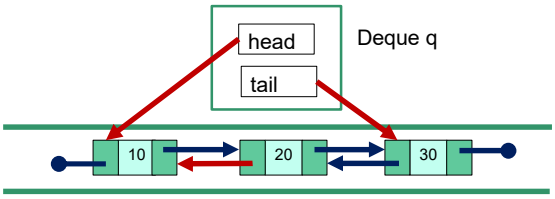



```
void pushRear(Deque* deque, int data) {
    DListNode* newNode = createNode(data);
    if (isEmpty(deque)) {
        deque->head = newNode;
        deque->tail = newNode;
    } else {
        newNode->prev = deque->tail;
        deque->tail->next = newNode;
        deque->tail = newNode;
    }
}
```

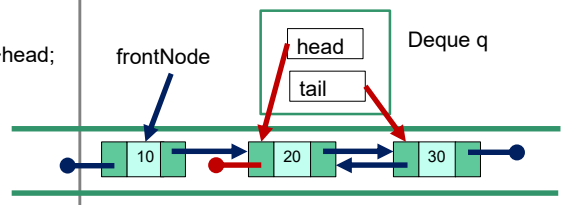
popFront()



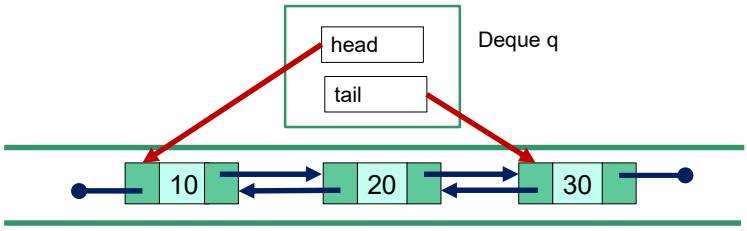
popFront()



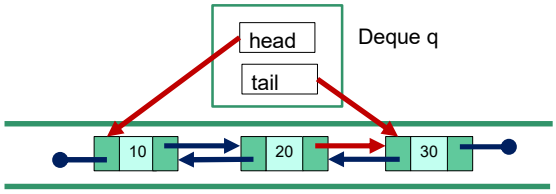

```
int popFront(Deque* deque) {
    if (isEmpty(deque)) {
        printf("Deque is empty.\n");
        return EXIT_FAILURE;
    }
    DListNode* frontNode = deque->head;
    int data = frontNode->data;
    deque->head = frontNode->next;
    if (deque->head != NULL) {
        deque->head->prev = NULL;
    } else {
        deque->tail = NULL;
    }
    free(frontNode);
    return data;
}
```



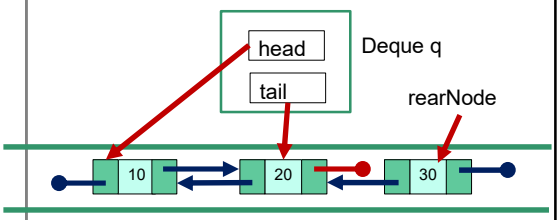
popRear()



popRear()




```
int popRear(Deque* deque) {
    if (isEmpty(deque)) {
        printf("Deque is empty.\n");
        return EXIT_FAILURE;
    }
    DListNode* rearNode = deque->tail;
    int data = rearNode->data;
    deque->tail = rearNode->prev;
    if (deque->tail != NULL) {
        deque->tail->next = NULL;
    } else {
        deque->head = NULL;
    }
    free(rearNode);
    return data;
}
```



peekFront()

```
int peekFront(Deque* deque) {
    if (isEmpty(deque)) {
        printf("Deque is empty.\n");
        return EXIT_FAILURE;
    }
return deque->head->data;
}
```

peekRear()

```
int peekRear(Deque* deque) {
    if (isEmpty(deque)) {
        printf("Deque is empty.\n");
        return EXIT_FAILURE;
    }
    return deque->tail->data;
}
```

Độ phức tạp thời gian tính

- printDeque(q): O(n)
- createDeque(): O(1)
- isEmpty(q): O(1)
- pushFront(q, x): O(1)
- pushRear(q, x): O(1)
- popFront(q): O(1)
- popRear(q): O(1)
- peekFront(q): O(1)
- peekRear(q): O(1)

Cấu trúc dữ liệu và giải thuật

- Nội dung bài giảng được biên soạn bởi PGS. TS. Phạm Tuấn Minh.

1-21