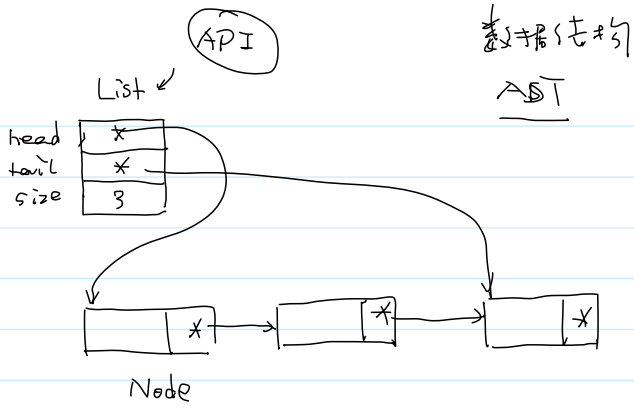
作业讲解

2024年5月4日 9:31

ad

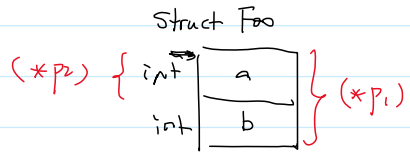
```c
typedef struct node {
    E data;
    struct node* next;
} Node;

typedef struct {
    Node* head;
    Node* tail;
    int size;
} List;
```



面试题：（基础）

直接操作结点



struct Foo foo = {1, 2};

struct Foo * p1 = &foo;

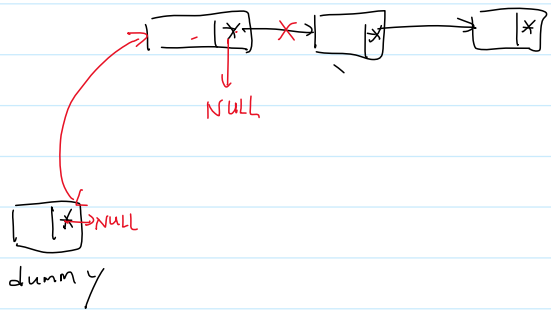
int * p2 = &foo.a;

int * p;

整数：short、int、long、long long

unsigned short、unsigned int、unsigned long、unsigned long long.

```c
p = head;
while(p){
    q = p->next;
    p->next = dummyHead.next;
    dummyHead.next = p;
    p = q;
}
```



栈的应用

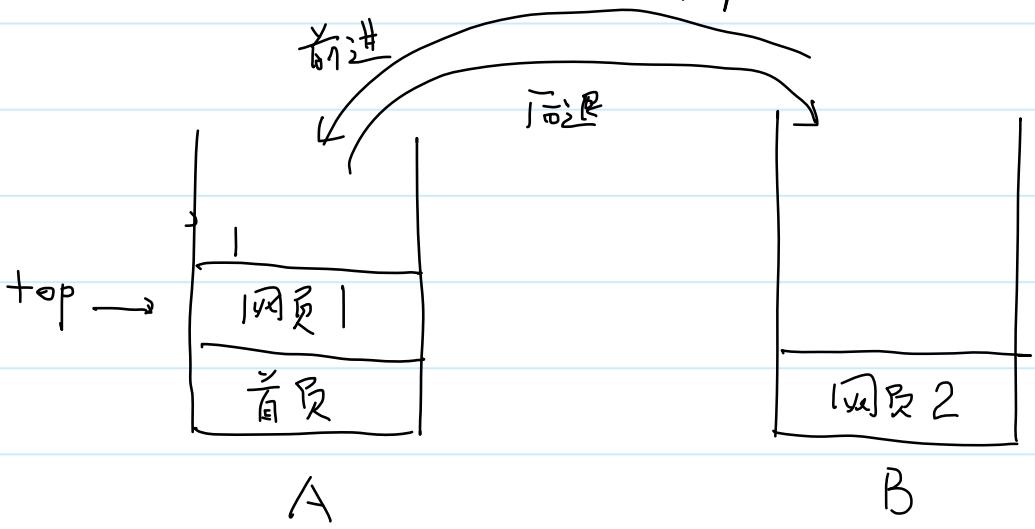
2024年5月4日 9:53

(3) 用栈来记录轨迹

a. 浏览器的前进/后退.

HTTP：无状态的协议.

↳ 每一次请求是独立.



b. 深度优先搜索

c. 回溯

队列

2024年5月4日 10:14

1. 模型：受限的线性表，一端添加元素，另一端删除元素

FIFO

图1 队列示意图

2. 基本操作

入队列 queue_push

出队列 queue_pop

查看队头 queue_peek

判空 queue_empty

判满 queue_full

3. 实现（动态数组）

a. 队头=0，用 rear 标识队尾 rear：下一个该要添加的位置

Capacity

入队列：O(1)

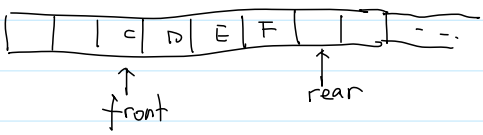
出队列：O(n)

查看队头：O(1)

判空：rear == 0

判满：rear == Capacity

b. 用 front 标识队头，用 rear 标识队尾 rear：下一个该要添加的位置



Capacity

入队列：O(1)

出队列：O(1)

出队列：O(1)

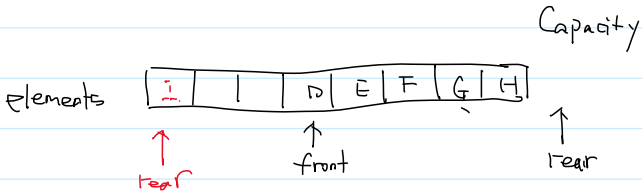
查看队头：O(1)

判空：rear - front == 0

判满：rear == Capacity → ① 浪费内存空间

rear - front == Capacity ② 需要大量移动元素

C、"循环数组"

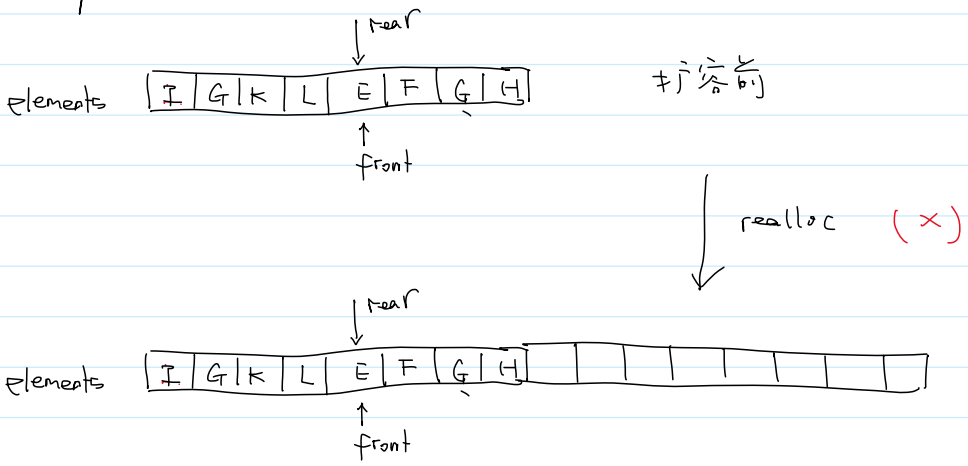


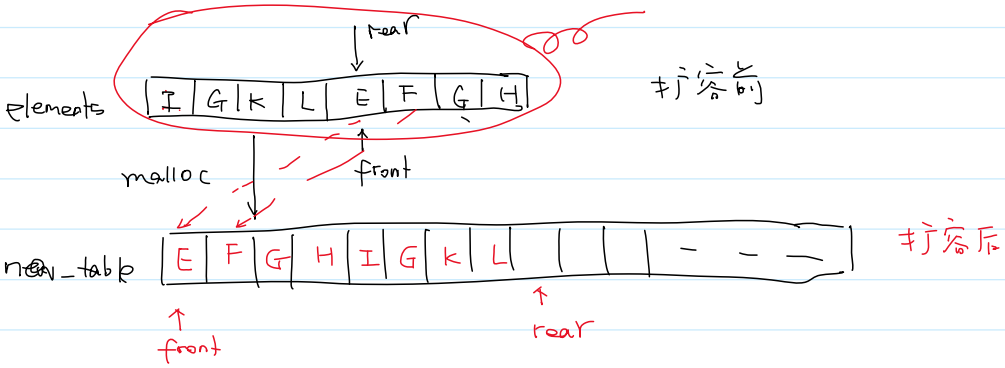
rear = (rear+1) % Capacity

入队列：O(1)

```
if ( -- ) {
    grow Capacity();   ⇒ 扩容.
}
elements[rear] = val;
rear = (rear+1) % Capacity.
```

扩容





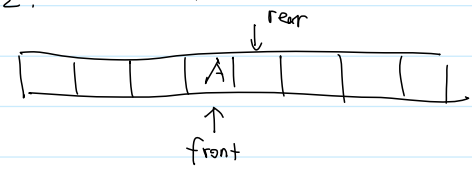
2. 出队列 O(1)

front = (front + 1) % capacity;

3. 查看队头

elements[front]

4. 判空. rear == front size == 0

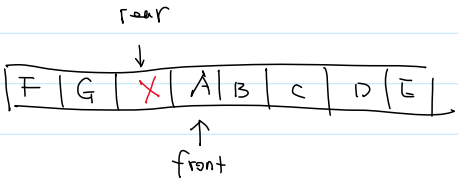


5. 判满. rear == front size == capacity

size

① (rear + 1) % capacity == front

② 添加 size 属性.



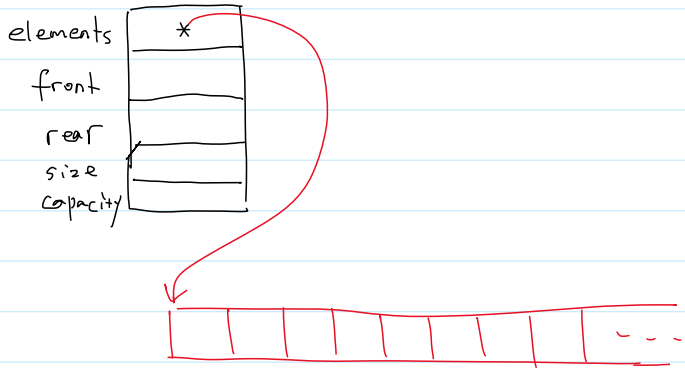

```c
// Queue.h
typedef int E;

typedef struct {
    E* elements;
    int front;
    int rear;
    int size;
    int capacity;
} Queue;

// API
Queue* queue_create();
void queue_destroy(Queue* q);

void queue_push(Queue* q, E val);
E queue_pop(Queue* q);
E queue_peek(Queue* q);

bool queue_empty(Queue* q);
```



#4. 应用.

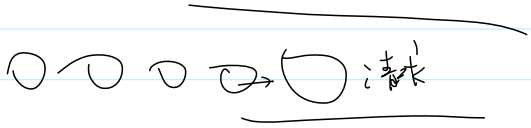
① 缓冲 (FIFO, 分平峰)

有界队列

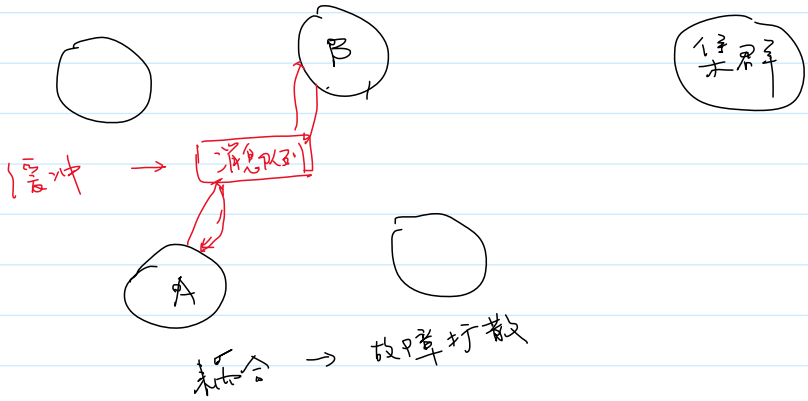
Out Of Memory

OOM

系统进程.



消息队列 (中间件)



② 广度优先搜索 （三度好友） 队列