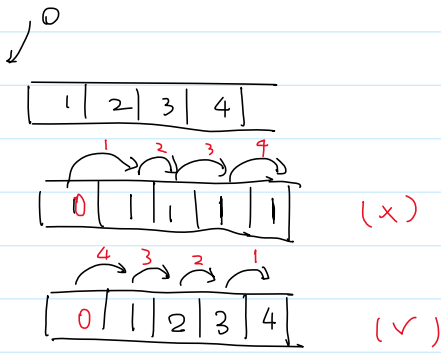
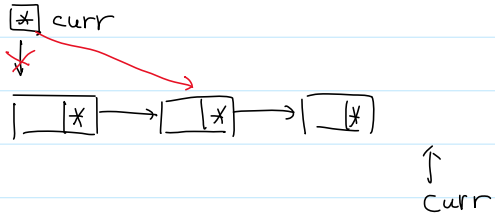
作业讲解

2024年5月3日 10:20

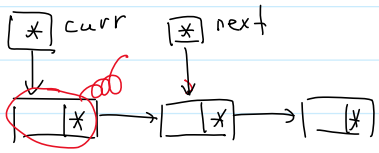
```
// 在数组最前面添加元素，所有元素依次后移
void push_front(Vector* v, E val) {
        if (v->size == v->capacity) {
                grow_capacity(v);
        }
        // 将所有元素后移一位
        for (int i = v->size; i > 0; i--) {
                v->elements[i] = v->elements[i - 1];
        }
        // 将元素添加到第一个位置
        v->elements[0] = val;
        v->size++;
}
```




```
void display_list(Node* list) {
        Node* curr = list;
        while (curr != NULL) {
                printf("%d ", curr->data);
                curr = curr->next;
        }
        printf("\n");
}
```




```
void free_list(Node* list) {
        Node* curr = list;
        while (curr != NULL) {
                Node* next = curr->next;
                free(curr);
                curr = next;
        }
}
```

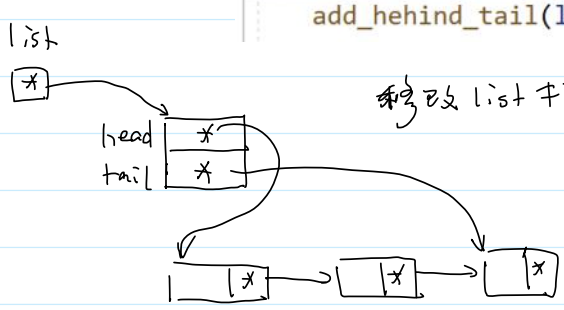


free(curr);
curr = curr->next; ⇒ use after free

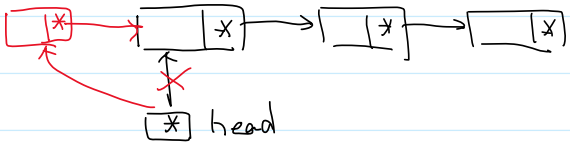
单链表

2024年5月3日 10:53

```c
int main(void) {
    List* list = list_create();

    // add_before_head(list, 1);
    // add_before_head(list, 2);
    // add_before_head(list, 3);
    // add_before_head(list, 4);    // 4 --> 3 --> 2 --> 1

    add_behind_tail(list, 1);
    add_behind_tail(list, 2);
    add_behind_tail(list, 3);
    add_behind_tail(list, 4);   // 1 --> 2 --> 3 --> 4
```

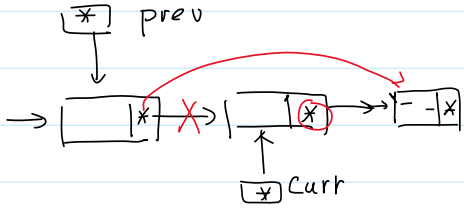


修改 list 指向对象，传 list



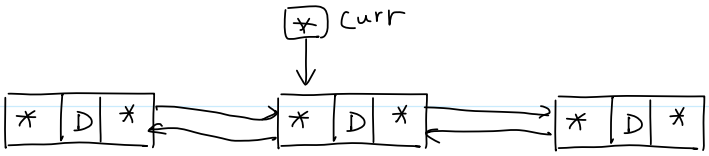
修改 head 的值（指向），传 &head

```c
prev->next = curr->next;
```



双链表

2024年5月3日 11:27



模型

基本操作：

增：在某个结点前面添加0

删：删除当前结点

查：a. 根据索引查找值。

单链表：O(n)，平均遍历 $\frac{n}{2}$

双链表：O(n)，平均遍历 $\frac{n}{4}$

b. 查找与特定值相等的结点

(1) 无序 O(n)

(2) 有序（多次查次）

单链表：

双链表：效果更佳

static Node* last; 记录上一次查找的结点

c. 查找前驱结点 O(1)

遍历。

正向

逆向。

空间和时间。

空间换时间：缓存 缓冲。

时间换空间：压缩、交换区（swap area）

常见的面试题

2024年5月3日 14:26

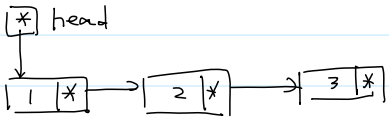
1. 求链表中间结点的值（876. 链表的中间结点）。

```
struct ListNode* middleNode(struct ListNode* head);
Example 1:
    输入：1 --> 2 --> 3
    输出：2
Example 2:
    输入：1 --> 2 --> 3 --> 4
    输出：3
```

数组：

n为奇：arr[$\frac{n}{2}$]

n为偶：arr[$\frac{n}{2}$]



思路1：

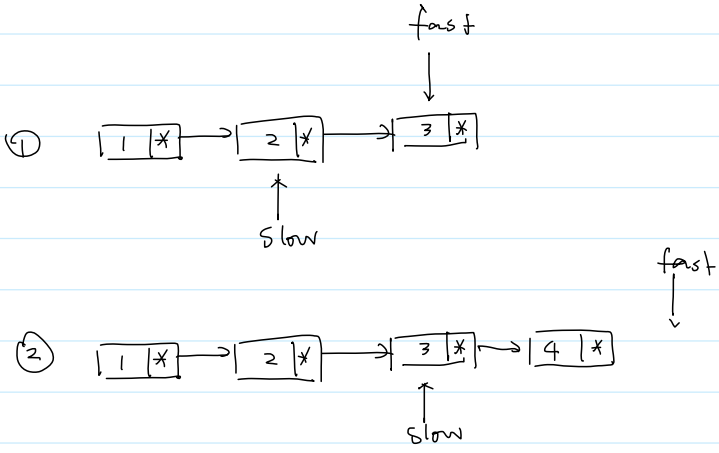
1、遍历链表，求链表的长度n。

2、求下标为$\frac{n}{2}$的结点

时间：O(n)

空间：O(1)

思路2：快慢指针



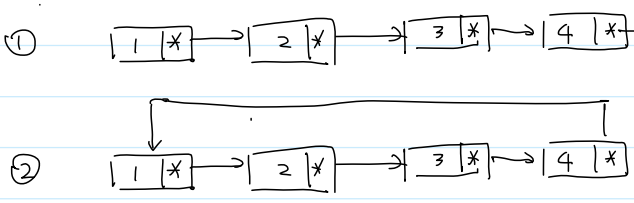
什么时候fast到达了末尾？

fast->next ==NULL || fast == NULL (X) 矩路原则

fast == NULL || fast->next == NULL (√)

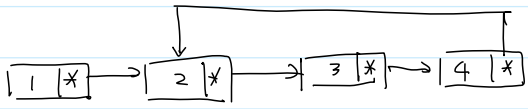
2. 判断单链表是否有环？（141. 环形链表）

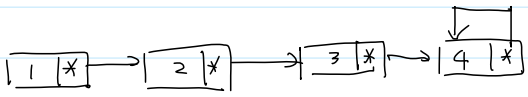
```
bool hasCycle(struct ListNode *head);
```

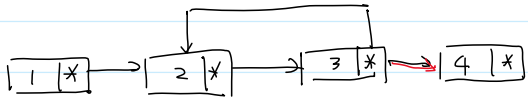


(false)

(true)

③  (true)

④  (true)

⑤  (不存在)

思路1. 迷雾森林

travelled: 已遍历结点的指针.

遍历结点, ① 判断当前结点 curr, 是否在 travelled 集合.

是, true

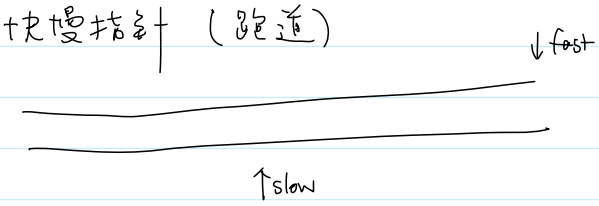
否, 将 curr 结点添加到 travelled 集合, curr = curr->next;

② curr == NULL, false.

时间: 和集合 travelled 的查找算法相关, 如果使用哈希表, O(n)

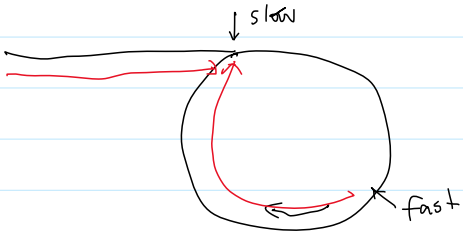
空间, O(n)

② 快慢指针 (跑道)



fast 一定会到达尾,

fast 和 slow 不会再一次相遇. (无环)



fast 和 slow 一定会再一次相遇. (有环)

时间: O(n)

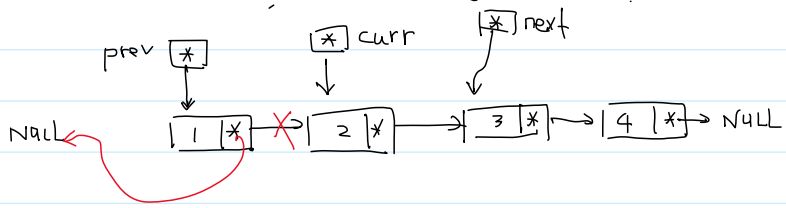
空间, O(1)

3. 反转单链表 (206. 反转链表)

```
struct ListNode* reverseList(struct ListNode* head);
Example 1:
    输入: 1 --> 2 --> 3
    输出: 3 --> 2 --> 1
Example 2:
    输入: 1
    输出: 1
```

思路1：头插法

遍历链表，依次反转每一个结点。




```
Node* next = curr->next;
curr->next = prev;
prev = curr;
curr = next;
```
反转一个结点

采用头插法，将 curr 结点插入到 prev 链表最前面。

时间：O(n)

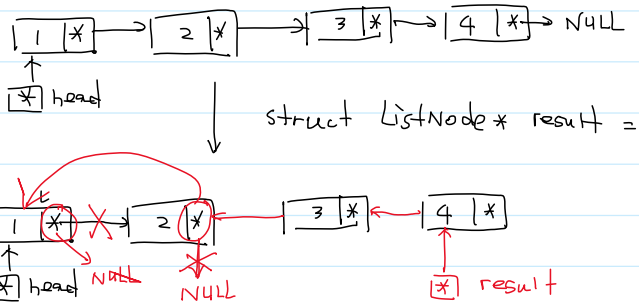
空间：O(1)

思路2：递归

边界条件：head == NULL || head->next == NULL

递归关键：

在逆转后 n-1 个结点的情况下，如何反转第一个结点。



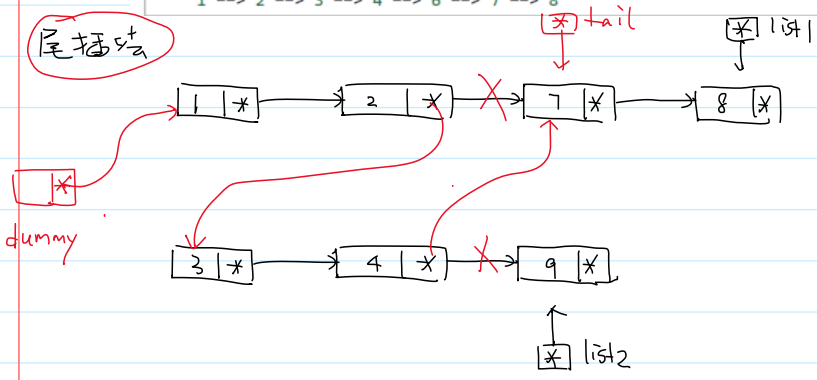

```
head->next->next = head;
head->next = NULL;
```

时间：O(n)

空间：O(n) → 栈的深度

4. 合并两条有序的单向链表，使得合并后的链表也是有序的（要求：不能额外申请堆内存空间）。（21. 合并两个有序链表）

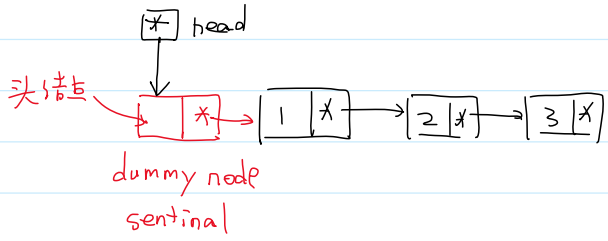
```
struct ListNode* mergeTwoLists(struct ListNode* list1, struct ListNode* list2);
Example1:
    输入：
    1 --> 3 --> 5
    2 --> 4 --> 6
    输出：
    1 --> 2 --> 3 --> 4 --> 5 --> 6
Example2:
    输入：
    1 --> 2 --> 3 --> 4
    6 --> 7 --> 8
    输出：
    1 --> 2 --> 3 --> 4 --> 6 --> 7 --> 8
```



tail → next = list1
tail = list1
list1 = list1 → next

tail → next = list2;
tail = list2;
list2 = list2 → next;

dummy node：哑结点（简化链表的操作）

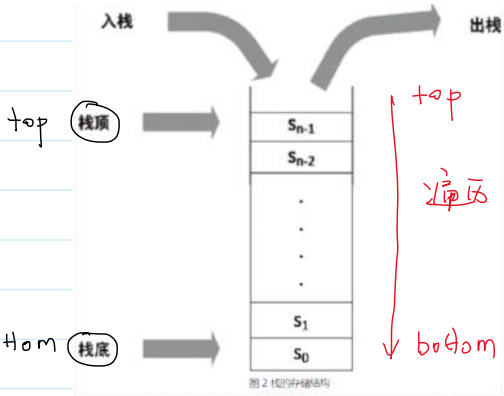


栈

2024年5月3日 16:56

一、模型：栈是操作受限的线性表（数组、链表），在一端添加，在同一端删除元素

受限



特性：LIFO

Q：为什么需要栈这种数据结构

1. 安全
2. 可读性强
3. 和现实生活中的东西是对应的

二、基本操作

添加：push

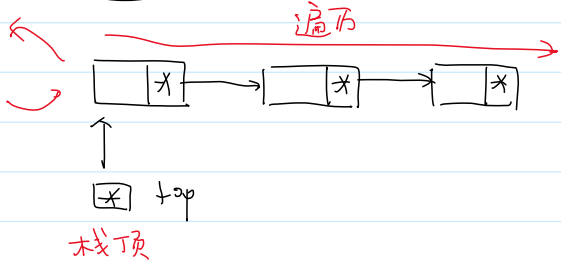
删除：pop

查找：peek

判空：empty → 遍历

三、实现（链表）

1. 用单链表还是双向链表？

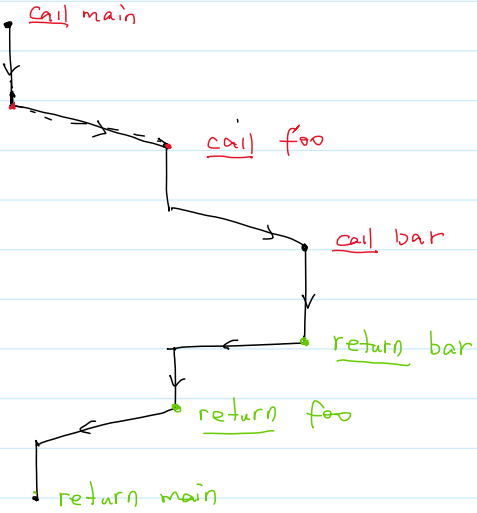


```c
// Stack.h
#include <stdbool.h>

typedef int E;

typedef struct node {
    E val;
    struct node* next;
} Node;

typedef struct {
    Node* top;
    int size;
} Stack;

// API
Stack* stack_create(void);
void stack_destroy(Stack* s);

void stack_push(Stack* s, E val);
E stack_pop(Stack* s);
E stack_peek(Stack* s);

bool stack_empty(Stack* s);
```

四. 应用

(1) 特点：LIFO

函数调用栈



call：入栈

return：出栈

LIFO

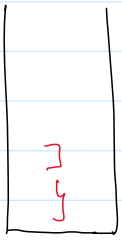
b. 符号匹配问题：( ) 、[ ] 、{ } 、Aa 、Bb ...

LIFO 匹配：{[( )]}、( )[]{}、([])

不匹配：{[( ])}、{( )[]}、([{])

左括号：入栈

右括号：出栈

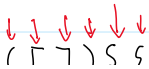


遍历字符串

1. 遇到左括号，将对应的右括号入栈；
2. 遇到右括号，出栈，判断是否和遇到符号相等

否：不匹配

是：匹配

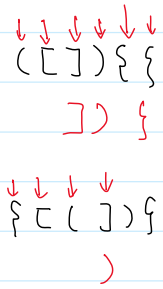


是：匹配

遍历完之后，判断栈是否为空，

是，匹配

否，不匹配



(2) 栈还可以表示优先级

↳ 单调栈

表达式求值.

中缀表达式：a + b * c / d ⇒ 优先级.

3 1 2

1 + 3 * 4 / 2

后缀表达式：1 3 4 * 2 / + ⇒ 没有优先级：运算符出现的顺序，就是实际执行的顺序.

a. 如何计算后缀表达式

操作数栈

① 遇到操作数：入栈

② 遇到运算符：

连续出栈两个操作数

计算

将结果入栈

1 3 4 * 2 / +

3 * 4 = 12

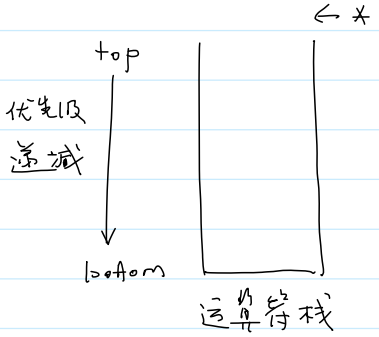
12 / 2 = 6

1 + 6 = 7

b 如何将中缀表达式转换成后缀表达式

1 + 3 * 4 / 2 - 7 * 8 ⇒ 优先级

3 1 2 5 4



top

优先级递减

bottom

运算符栈

① 遇到操作数，直接输出

② 遇到运算符：

循环出栈优先级大于等于它的运算符

入栈

③ 依次出栈，将运算符拼接在末尾.

1 2 3 4 5

1 3 4 * 2 / + 7 8 * -