

数据结构与算法链表及其应用

陈宇琪

2020 年 3 月 27 日

摘要

主要内容包括 list, linked-queue, linked-stack。

理论课作业包括：

- 选择题和简答题第一小问（在超星上提交）
- 编程题第一小问（在超星上提交，请注意各种异常情况，链接节点定义已经给出，请大家不要修改！）
- EOJ 上 4.2 链表去重

实践课作业包括：

- EOJ 上 2.1 队列的实现和 4.1 约瑟夫环问题（请大家完成过程中用规范的链表来完成，不允许使用任何 STL 工具）
- 超星实践作业模块中的两道题目（其中一道题也是约瑟夫环问题，这次请大家尝试使用 STL 的 list 在完成）

注意：简答题第二题如果有兴趣的同学可以在超星上写下自己的想法。

提交代码要求：提交完整可运行的代码，同时考虑代码的鲁棒性和异常情况！

ddl: 2020-04-05

目录

1	链表 Linked structure	2
1.1	节点定义 Node	2
1.2	链表插入 Insert	2
1.3	链表删除 Delete	3
1.4	Linked Stack	3
1.5	Linked Queue	3
1.6	其他链表结构	3
1.7	Linked Strcture 时间复杂度分析	4
2	选择题	4
3	简答题	4
4	编程练习	4
5	附录：奇思妙想	5

1 链表 Linked structure

A linked structure is made up of nodes, each containing both the information that is to be stored as an entry of the structure and a pointer telling where to find the next node in the structure.

1.1 节点定义 Node

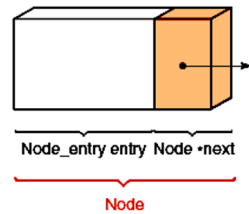


图 1: Node definition

1.2 链表插入 Insert

第一种情况 在第一个结点前插入

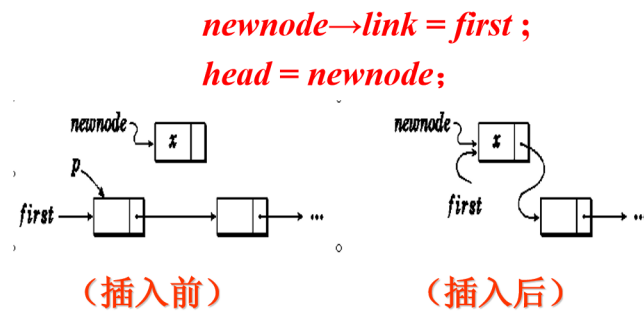


图 2: Insert 1

第二种情况 在链表中间插入

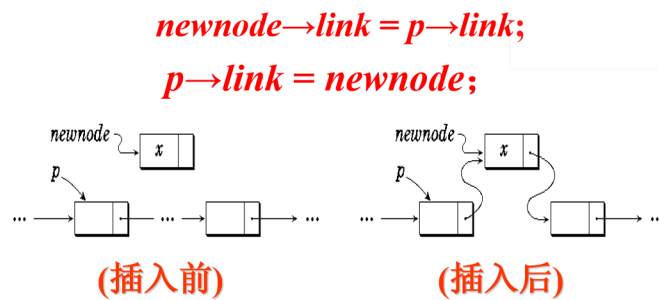


图 3: Insert 2

问题

- 1、是否需要考虑在链表尾部插入的特殊情况? (可以不需要考虑)

2、写代码是否需要考虑什么？（如果插入的位置大于链表的长度，需要处理这种异常）

1.3 链表删除 Delete

类似 insert 的过程，需要考虑两种情况，同时需要处理各种异常情况。

特别注意：在 delete 的时候需要将删除的节点对应的内存空间释放掉，以避免内存泄漏。

1.4 Linked Stack

Linked Stack 可以看成简单的链表，它只支持在链表的头部插入和删除数据。

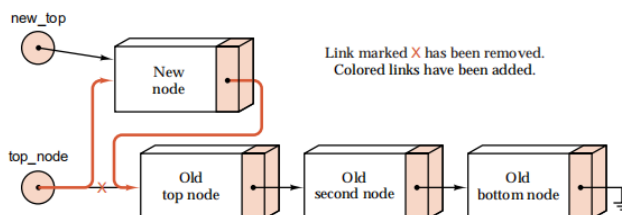


图 4: Linked Stack-Push

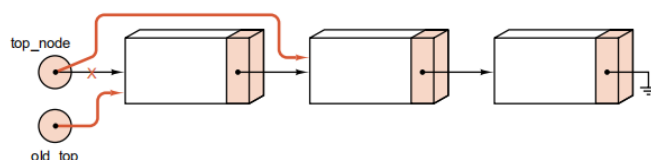


图 5: Linked Stack-Pop

1.5 Linked Queue

区别于 Linked Stack, Linked Queue 为了使得每个操作的复杂度都是 $O(1)$ ，需要动态维护 head 和 tail 两个指针。

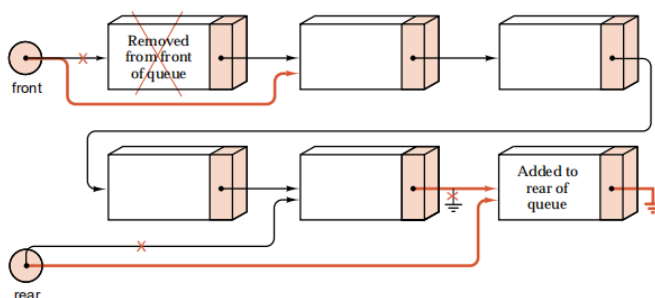


图 6: Linked Queue-Push

1.6 其他链表结构

单向链表，双向链表，循环单链表，循环双链表。

1.7 Linked Strcture 时间复杂度分析

表 1: Linked Strcture 时间复杂度分析

Data Structure	Function	Time Complexity	Expect Time Complexity
Linked List	Insert	$\Omega(1), O(N)$	$O(N)$
Linked List	Delete	$\Omega(1), O(N)$	$O(N)$
Linked Stack	Push	$O(1)$	$O(1)$
Linked Stack	Pop	$O(1)$	$O(1)$
Linked Stack	Top	$O(1)$	$O(1)$
Linked Queue	Push	$O(1)$	$O(1)$
Linked Queue	Pop	$O(1)$	$O(1)$
Linked Queue	Front	$O(1)$	$O(1)$

2 选择题

1、链式栈结点为: (data,link), top 指向栈顶. 若想摘除栈顶结点, 并将删除结点的值保存到 x 中, 则应执行操作 ()。

A. x=top->data;top=top->link; B. top=top->link;x=top->link;

C. x=top;top=top->link; D. x=top->link;

2、最大容量为 n 的循环队列, 队尾指针是 rear, 队头是 front, 则队空的条件是 ()。

A. (rear+1)%n==front B.rear==front C. rear+1==front D. (rear-1)%n==front

3、用链接方式存储的队列, 在进行删除运算时 ()。

A. 仅修改头指针 B. 仅修改尾指针 C. 头、尾指针都要修改 D. 头、尾指针可能都要修改

3 简答题

1、在编程练习中实现了约瑟夫环问题, 那么能不能用循环单链表实现约瑟夫环问题? 有没有什么解决方法? 在哪些情况下可能会存在异常? 在这些异常情况下又应该怎么处理?

*2、在“附录: 奇思妙想”中介绍了我胡思乱想的数据结构, 那么它的优点是什么? 缺点是什么呢? 这个缺点是不是致命的呢? 这样的做法在什么情况下可能有效? 请作简要的说明。

4 编程练习

链表节点定义:

Listing 1: node.cpp

```
#include <iostream.h>
typedef int Node_entry;

struct Node {
    // data members
    Node_entry entry;
    Node *next;
    // constructors
    Node( );
    Node(Node_entry item, Node *add_on = NULL);
```

```
};
```

1、(完整代码)使用单链表实现“删除链表的倒数第 N 个节点”。

问题描述: 给定一个链表, 删除链表的倒数第 N 个节点, 并且返回链表的头节点。

例如: 链表 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$, $N = 2$, 则输出 $1 \rightarrow 2 \rightarrow 3 \rightarrow 5$ 。

注意: 你的算法只能使用常数的额外空间 (除了输入的链表以外的空间); 只能遍历一次链表。

请大家注意以下说明:

- 函数原型 `Node * remove_back_nth_element(Node *head, int N);`
- 你只需要提交这个函数就可以了。
- 假设你不知道链表的长度 (也就是说如果你想知道链表的长度, 就已经遍历了一次链表)。
- 在评分中会考察程序的正确性, 对异常的处理, 以及是否只遍历了一次链表。

*2、使用单链表实现“ k 个一组翻转链表”。

问题描述: 给你一个链表, 每 k 个节点一组进行翻转, 请你返回翻转后的链表。如果节点总数不是 k 的整数倍, 那么请将最后剩余的节点保持原有顺序。

例如: 链表 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$, $k = 2$, 则输出 $2 \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 5$ 。

注意: 你的算法只能使用常数的额外空间 (除了输入的链表以外的空间); 同时你的算法不能只是单纯的改变节点的内部值, 而是需要实际进行节点交换。

备注: 由于题目较难, 所以不要求大家提交, 有兴趣同学可以前往 <https://leetcode-cn.com/problems/reverse-nodes-in-k-group/> 提交。

3、完成 EOJ 上习题 2.1, 4.1-4.2, 其中 2.1 和 4.1 将作为实践课作业, 请认真完成。

5 附录: 奇思妙想

在 1.7 的分析中, 我们可以看到链表在复杂度上的表现并不是十分理想, 那么我们是不是可以优化链表的性能?

以下算法来自我的胡思乱想:

假设我们有一个数据结构, 我们定义一个 `DataStruct` 的类, 其类定义可以写成:

Listing 2: `ds.cpp`

```
#include <bits/stdc++.h>
using namespace std;

template<class T>
class DataStruct
{
public:
    DataStruct();
    T get(int rank);
    void add(const T& data);
    void remove(const T& data);

private:
    ...
};
```

支持以下三个操作：

- get rank 返回数据结构中第 rank 大的元素
- remove data 在数据结构中删除数据为 data 的节点
- add data 在数据结构中插入数据为 data 的节点

其中每个函数的复杂度均为 $O(\log(N))$ 。

有了这个数据结构那么我们可以这样实现链表的插入，假设当前链表中有 K 个元素，插入位置为 pos：

- $K = 0$ ：查找 rank=1 的元素，分配当前元素的编号为 rank=1 的元素的编号-1。
- $0 < K < pos$ ：查找 rank=pos,rank=pos+1 的元素，分配当前元素的编号为两个元素的编号的平均。
- $K = pos$ ：查找 rank=pos 的元素，分配当前元素的编号为 rank=pos 的元素的编号 +1。
- $K > pos$ ：插入不合法。

其思想为对每一个数据分配一个唯一的编号，根据编号查找数据结构中的第 pos 大的元素。

删除和查找时候，只要 pos 合法，我们就可以找到第 pos 大的元素，然后删除这个元素。

于是每个操作的复杂度都是 $O(\log(N))$ ，其中编号我们可以用 double 类型。

当然，这个所谓的数据结构是基于 splay tree 实现的名次数（rank tree）。

Listing 3: mylist.cpp

```
#include <bits/stdc++.h>
using namespace std;

template<class T>
class myList
{
public:
    myList();
    void insert(const T &data,int pos)
    T get(int pos);
    void remove(int pos);

private:
    int cnt;
    DataStruct<double,T> ds;
};
```