GeekBand 极客班

互联网人才 + 油站!

GeekBand 极客班

www.geekband.com

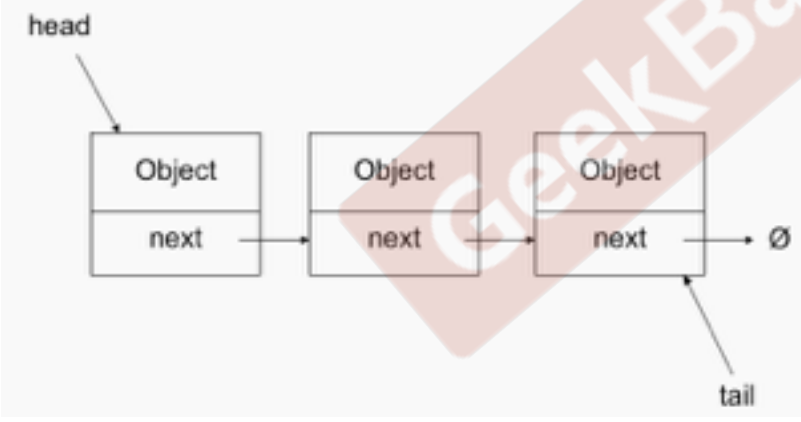
2. 链表

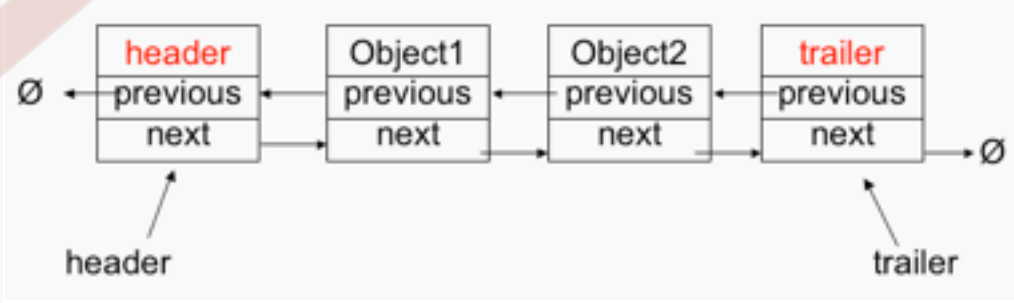
大纲

1. 链表介绍
2. 基本操作
3. Dummy Node
4. 追赶指针技巧
5. 例题分析

链表介绍

单向链表(singly linked list)，每个节点有一个next指针指向后一个节点，
还有一个成员变量用以储存数值；
双向链表(Doubly Linked List)，还有一个prev指针指向前一个节点。
Search: O(n), Del, Add: O(1)

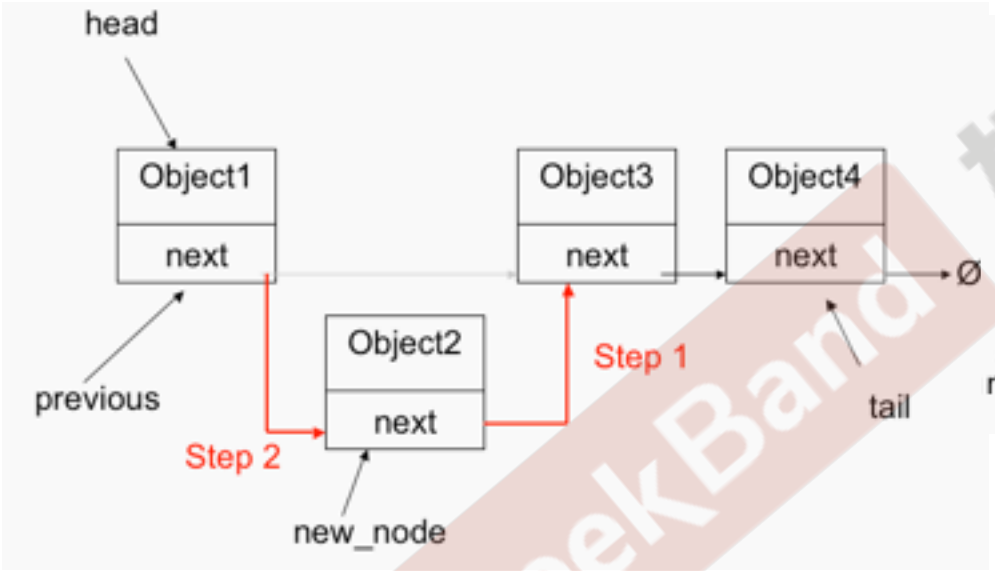


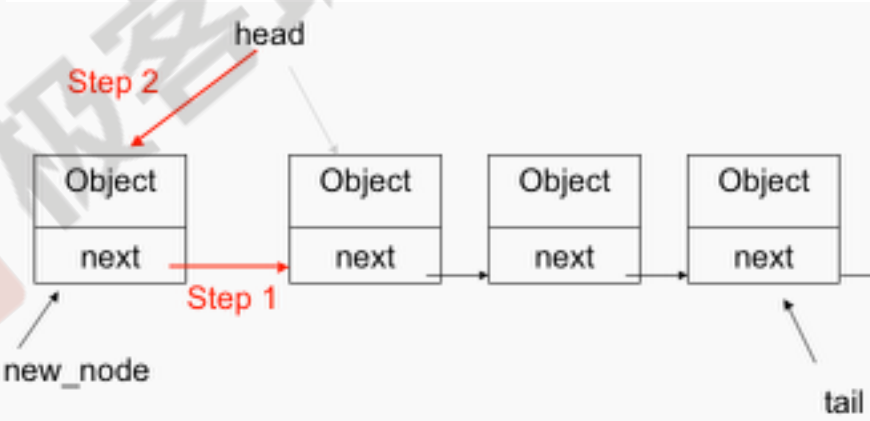


基本操作：查找

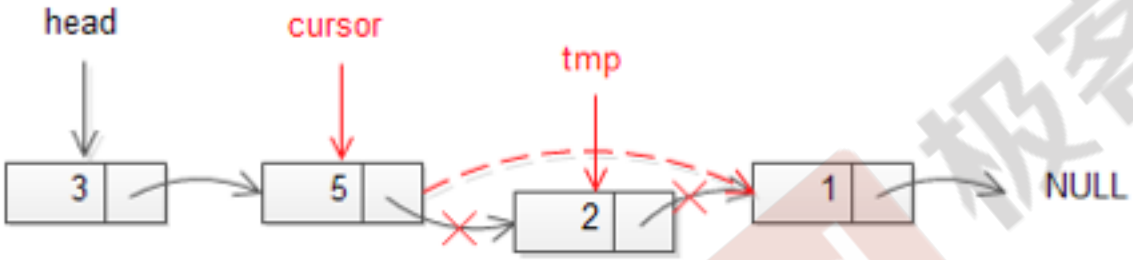
```
Node* LinkedList::search(int val) {
    Node* pNode = _pHead;

    /* traverse the list */
    while (pNode != NULL) {
        /* Target! */
        if(pNode->_value == val) {
            return pNode;
        }
        /* move to the next one */
        pNode = pNode->_pNext;
    }
    return NULL;
}
```

基本操作：增加





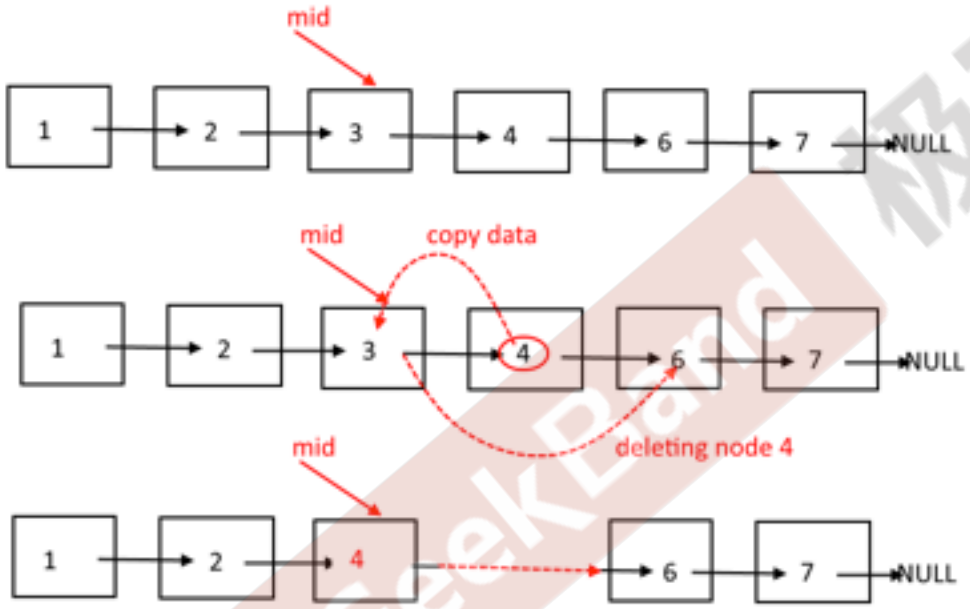
基本操作：删除




```
void delNode(ListNode *prev) {
  ListNode *curr = prev->next;
  prev->next = curr->next;    // 删除curr节点只会使prev节点的next受到影响
  delete curr;    // 清理trash指针
}
```

注：操作Linked List时务必注意边界条件：curr == head, curr == tail 或者 curr == NULL

直接删除当前节点



Remove Duplicates from Sorted List

Given a sorted linked list, delete all duplicates such that each element appear only once. For example, Given 1->1->2, return 1->2. Given 1->1->2->3->3, return 1->2->3.

Dummy Node技巧

考虑：
a. 哪个节点的next指针会受到影响，则需要修正该指针
b. 如果待删除节点是动态开辟的内存空间，则需要释放这部分空间(C/C++)

利用dummy node是一个非常好用的trick：只要涉及操作head节点，当头节点操作不确定的时候，不妨创建dummy node:

```
ListNode *dummy = new ListNode(0);
dummy->next = head;
```

Remove Duplicates from Sorted List II

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

For example,
Given 1->2->3->3->4->4->5, return 1->2->5.
Given 1->1->1->2->3, return 2->3.

Partition List

Given a linked list and a value x, write a function to reorder this list such that all nodes less than x come before the nodes greater than or equal to x.

解题分析：将list分成两部分，但两部分的head节点连是不是null都不确定。但总是可以创建两个dummy节点然后在此基础上append，这样就不用处理边界条件了。

追赶指针技巧

对于寻找list某个特定位置的问题，不妨用两个变量chaser与runner，以不同的速度遍历list，找到目标位置： ListNode *chaser = head, *runner = head。并且可以用一个简单的小test case来验证（例如长度为4和5的list）

Middle Point

Find the middle point of linked list.

解题分析： 寻找特定位置，runner以两倍速前进，chaser 一倍速，当runner到达tail时，chaser即为所求解。

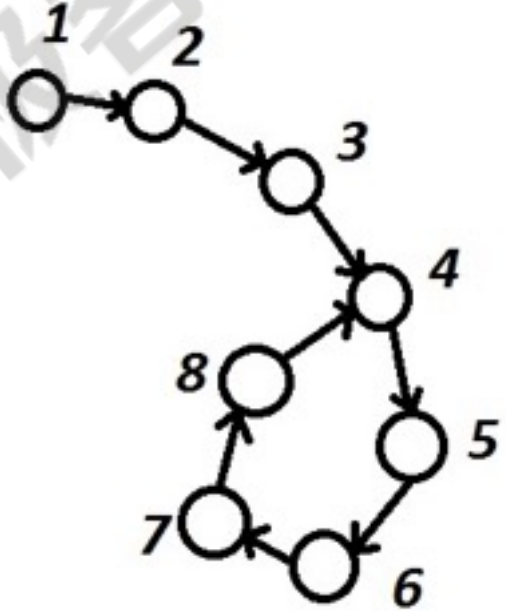
kth to Last element

Find the kth to last element of a singly linked list

解题分析：之前类似。只是runner与chaser以相同倍速前进，但runner提前k步出发

如何判断一个单链表中有环?

Given a linked list, determine if it has a cycle in it.



Circular List Node

Given a circular linked list, return the node at the beginning of the loop

解题分析：寻找某个特定位置，用runner technique。Runner以两倍速度遍历，假定有loop，那么runner与chaser一定能在某点相遇。相遇后，再让chaser从head出发再次追赶runner，第二次相遇的节点为loop开始的位置。

判断两个单链表是否有交点?

先判断两个链表是否有环，如果一个有环一个没环，肯定不相交；如果两个都没有环，判断两个列表的尾部是否相等；如果两个都有环，判断一个链表上的Z点是否在另一个链表上。

如何找到第一个相交的节点?

求出两个链表的长度L1,L2（如果有环，则将Y点当做尾节点来算），假设L1<L2，用两个指针分别从两个链表的头部开始走，长度为L2的链表先走(L2-L1，然后两个一起走，直到二者相遇。

Rotate List

Given a list, rotate the list to the right by k places, where k is non-negative.

模式识别

1.在遍历Linked list时，注意每次循环内只处理一个或一对节点。核心的节点只处理当前这一个，否则很容易出现重复处理的问题。

Reverse Linked List

Reverse the linked list and return the new head.

循环遍历linked-list, 每次只处理当前指针的next 变量。

非递归 vs 递归

模式识别

2. Swap Node 问题

交换两个节点，不存在删除的话，两个节点的prev节点的next指针，以及这两个节点的next指针，会受到影响。总是可以

a. 先交换两个prev节点的next指针的值；

b. 再交换这两个节点的next指针的值。

无论这两个节点的相对位置和绝对位置如何，以上的处理方式总是成立

Swap Adjacent Nodes

Given a linked list, swap every two adjacent nodes and return its head.

模式识别

3. 同时处理两个linked list的问题，循环的条件一般可以用 while(l1 && l2) ,再处理剩下非NULL 的list。这样的话，边界情况特殊处理，常规情况常规处理。

Add List Sum

Given two linked lists, each element of the lists is a integer. Write a function to return a new list, which is the "sum" of the given two lists.
Part a. Given input (7->1->6) + (5->9->2), output 2->1->9.
Part b. Given input (6->1->7) + (2->9->5), output 9->1->2.

解题分析：对于a，靠前节点的解不依赖靠后节点，因此顺序遍历求解即可。对于b，靠前节点的解依赖于靠后节点（进位），因此必须用递归或栈处理。并且，subproblem返回的结果，可以是一个自定义的结构（进位 + sub-list）。当然，也可以reverse List之后再用a的解法求解。

Merge Two Sorted List

Merge two sorted linked lists and return it as a new list.

Merge K Sorted List

```
1:    ListNode *mergeKLists(vector<ListNode *> &lists) {
2:      if(lists.size() == 0) return NULL;
3:      ListNode *p = lists[0];
4:      for(int i =1; i< lists.size(); i++)
5:      {
6:        p = merge2Lists(p, lists[i]);
7:      }
8:      return p;
9:    }
```

Better Solution?

HEAP

1. Create a heap to store ListNode*, which should sort list nodes in the ascending order or node values.
2. Insert the first node(head) of each list into the heap, so that we store all the k entries in the heap.
3. Get the top element in heap and add in to the merged list.
4. If the top element of heap is the last node in a list, pop it and go to step 3.
5. If the top element of heap has followers, pop it and push its following node into heap.
6. Go to step 3 until the heap is empty.

模式识别

4.如果对靠前节点的处理必须在靠后节点之后，即倒序访问问题，则用recursion，或者等效地，stack来解决。

例题

Traverse the linked list reversely.

```
void traverse(ListNode *head) {
  if (head == NULL)
    return;
  traverse(head->next);
  visit(head);
}
```

基础训练

1. Insert a Node in Sorted List
2. Remove a Node from Linked List
3. Reverse a Linked List
4. Merge Two Linked Lists
5. Find the Middle of a Linked List

工具箱

对C++,

Doubly linked list的实现类是std::list<T>.

常用iterator： begin(), end(), rbegin(), rend().

常用函数:
empty(), size(), push_back(T value), pop_back(T value);
erase(iterator pos), insert(iterator pos, T value);

对于Java,

Doubly linked list 的实现类是 LinkedList<E>

常用函数:
add(E e), add(int index, E element), remove(int index),
addAll(Collection<? Extends E> c), get(int index),

Reorder List

Given a singly linked list L: L0→L1→…→Ln-1→Ln,
reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

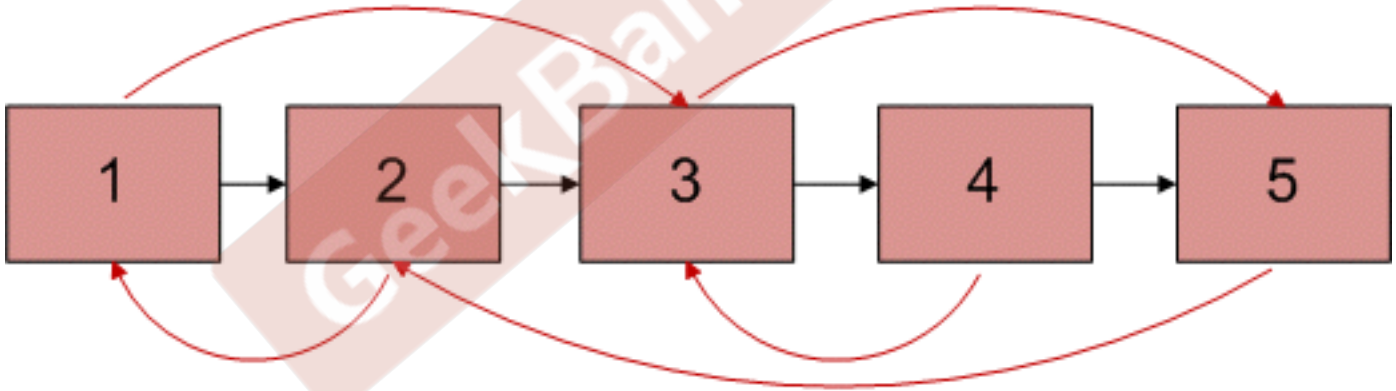
You must do this in-place without altering the nodes' values.

Example
For example,
Given 1->2->3->4->null, reorder it to 1->4->2->3->null.

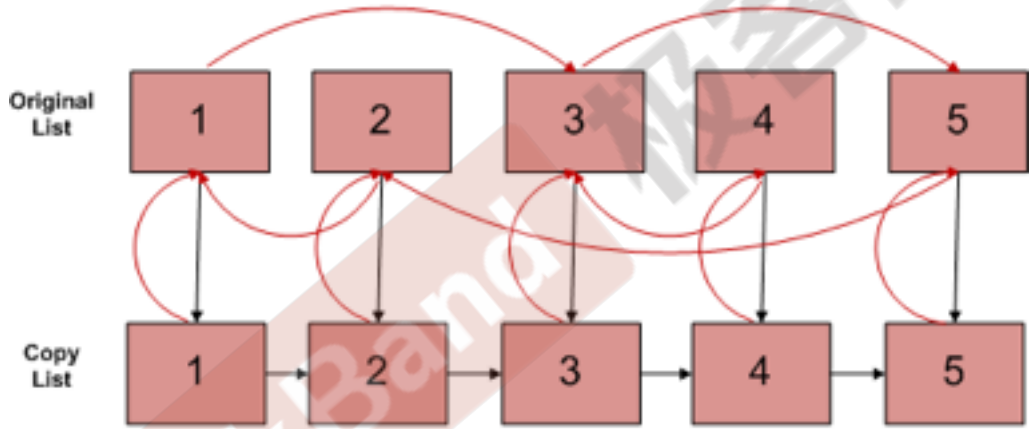
Clone a linked list with next and random pointer

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.



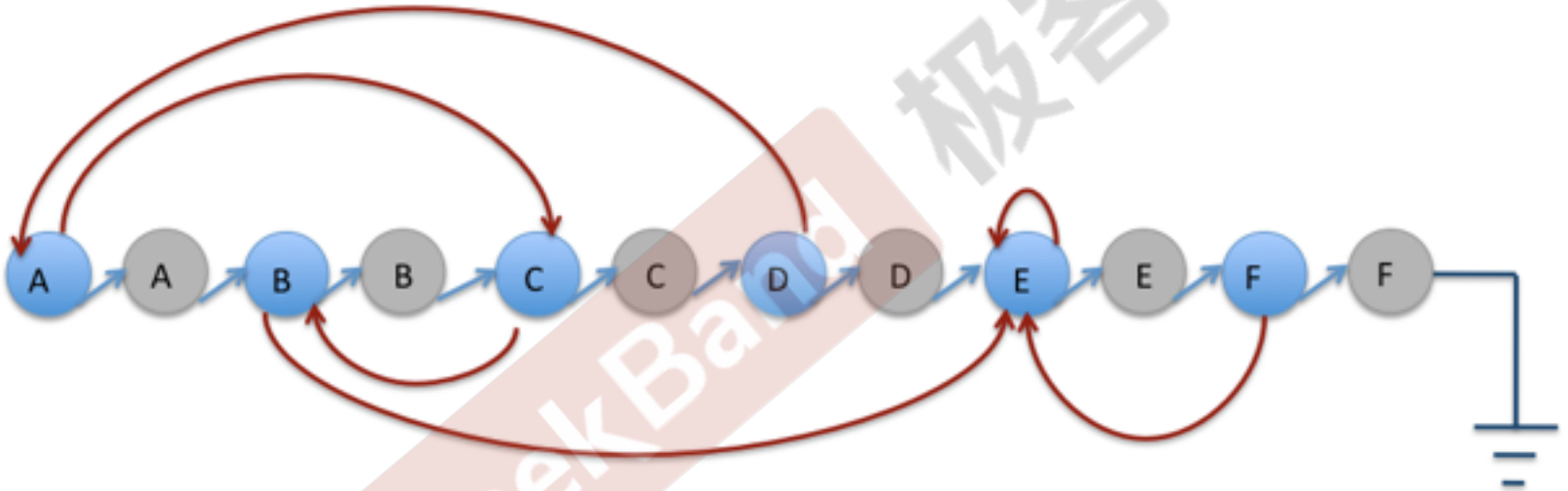
Using HashMap



Time Complexity: O(n)

Auxiliary Space: O(n)

Better Solution?



node->next->random = node->random->next;

Homework

Remove Duplicates from Unsorted List

Write a removeDuplicates() function which takes a list and deletes any duplicate nodes from the list. The list is not sorted.

For example if the linked list is 12->11->12->21->41->43->21, then removeDuplicates() should convert the list to 12->11->21->41->43.

If temporary buffer is not allowed, how to solve it?

Reverse a linked list

Reverse a linked list from position m to n.

Note
Given m, n satisfy the following condition: 1 ≤ m ≤ n ≤ length of list.

Example
Given 1->2->3->4->5->NULL, m = 2 and n = 4, return 1->4->3->2->5->NULL.

Challenge
Reverse it in-place and in one-pass

Palindrome List

Given a singly linked list of characters, write a function that returns true if the given list is palindrome, else false.