report

week 4

程雨歌 12307110079

2016 年 3 月 26 日

1 reverseList.c

1.1 函数说明

输入 指向一个链表头指针的指针。

输出 无输出，将指向的链表反向。

实现方法 现将链表头指向 NULL，然后从原链表头节点开始，依次将节点运用**头插法**插入 head，直到链表结束即可。

每一次插入的具体步骤为:

1. 记录待插入节点 p 的下一节点 t;
2. 将待插入节点 p 的 next 指针指向当前的头节点 head 的下一节点;（此时已插入该节点 p）
3. 将头节点 head 指向刚刚插入的节点 p;
4. 将之前记录的下一节点 t 设为待插入节点 p

1.2 复杂度

时间复杂度 $\mathcal{O}(n)$

空间复杂度 $\mathcal{O}(1)$

1.3 边界情况

当链表为空时，不进入头插法循环，故直接返回 head;

当链表只有一个节点时，先将头节点指向 NULL，再将该节点指向头节点的 next（即 NULL），然后头节点指向该节点;

1.4 程序运行结果

```
Case 1:
The original linked list:
null
The reversed linked list:
null

Case 2:
The original linked list:
44
The reversed linked list:
44

Case 3:
The original linked list:
58->30
The reversed linked list:
30->58

Case 4:
The original linked list:
45->10->14->2->97->2->79->98
The reversed linked list:
98->79->2->97->2->14->10->45

sh: pause: command not found
```

2 isPalindrome.c

2.1 函数说明

输入 指向一个链表头指针的指针。

输出 一个布尔值，即该链表是否为回文串。

实现方法

1. 先用半速移动法标记链表的中点节点；
2. 将链表截断，并将后半截链表反向（运用第一题的方法）；
3. 依次对比两个半链表，若直到短的结束两者都完全一致，则返回 true；否则返回 false。

2.2 复杂度

时间复杂度 $\mathcal{O}(n)$

空间复杂度 $\mathcal{O}(1)$

2.3 边界情况

若输入为空链表，两个头节点的 next 都为 NULL，第三步中不进行比较，返回 true。

若输入为奇数长度的回文串，中间节点被分在（反向后的）后半截链表的末尾，因此在进行比较时当在前半截链表上的 t 到达末尾，比较结束，无需比较中间节点，也不会出现调用空节点数值。

注意，在对后半截链表进行反转前，新声明了一个无意义的节点，以符合头节点的新约定。

2.4 程序运行结果

```
11->19->29->34->16->98->48->48->98->16->34->29->19->11
Your judgement: True
Correct!

Start testing...

Your code runs well!

sh: pause: command not found
```

3 Argue

助教您好!

第一周作业我的评分为**A-**。附注为:“没有考虑 numsSize = 0 的情况”，这也许与实际情况不相符。

不知道能否请您重新评估我的成绩?

我虽然没有做专门的条件判断，但其实是考虑到的，当 numsSize = 0 时，函数所有循环都不会进行，自然结果就是返回默认值 false（containsDuplicate）或不进行任何操作（moveZeroes）。您可以查看当时我提交到邮箱里的代码，或者审阅我复制到文末的当时的代码。

抱歉给您添麻烦了！谢谢!

Code 1 – containsDuplicate.c

```c
// File encoding with UTF-8
// Please compile with -std=c99
//
// Week 1 Assignment 1
// 2016.02.29
// 程雨歌 12307110079
// Aim: to find if the given array contains any duplicates.
// Algorithm: quickSort the array, then check if there are identical adjacent elements.

#include <stdio.h>
#include <stdlib.h> // for function malloc
#include <string.h> // for function memcpy
#include <stdbool.h> // for type Bool

void quickSort(int*, int, int);
int roughSort(int*, int, int);
bool containsDuplicate(int*, int); // key function of this assignment
```

```c
void verbose(int*, int);

int main(int argc, const char* argv[])
{
    int a[] = {7, 12, 1, -2, 0, 15, 4, 11, 9};
    verbose(a, 9);
    int b[] = {9, 12, 1, -4, 0, 15, 4, 11, 9};
    verbose(b, 9);
    verbose(b, 9);
    return 0;
}

void quickSort(int* a, int l, int r)
{
    int m; // location of the pivot of roughSort
    if (l < r)
    {
        m = roughSort(a, l, r); // divide and conquer
        quickSort(a, l, m - 1);
        quickSort(a, m + 1, r);
    }
}

int roughSort(int* a, int l, int r) // bifurcate, increasingly
{
    int pivot, i, j, t;
    pivot = a[l];
    i = l; j = r + 1;
    while(1)
    {
        while(a[i] <= pivot && i <= r) i++;
        while(a[j] > pivot && j >= l) j--;
        if ( i >= j ) break;
        t = a[i]; a[i] = a[j]; a[j] = t;
    }
    t = a[l]; a[l] = a[j]; a[j] = t; // swap pivot to middle
    return j;
}

bool containsDuplicate(int* nums, int numsSize)
{
    int* a = malloc(numsSize * sizeof(int)); // get a copy to keep given array
    untouched
    memcpy(a, nums, numsSize * sizeof(int));
    quickSort(a, 0, numsSize - 1);
    int i;
    for (i = 0; i < numsSize - 2; i++)
        if (a[i] == a[i + 1])
        {
            free(a);
            return true;
        }
    free(a);
    return false;
}

void verbose(int* nums, int numsSize)
{
    int i;
    printf("\n{");
    for(i = 0; i < numsSize - 1; i++)
        printf("%d, ", nums[i]);
```

```
    printf("%d}\n", nums[i]);
    if (containsDuplicate(nums, numsSize))
        printf("CONTAINS duplicate(s).\n");
    else
        printf("contains NO duplicates.\n");
}
```

Code 2 – moveZeroes.c

```
// File encoding with UTF-8
//
// Week 1 Assignment 2
// 2016.02.29
// 程雨歌 12307110079
// Aim: to move all 0's to the end of the given array while maintaining the relative order of the non-zero elements.
// Algorithm: move the non-zero elements to front and queue.

#include <stdio.h>

void moveZeroes(int*, int);

int main(int argc, const char* argv[])
{
    int a[] = {0, 1, 0, 3, 12, -4, 0, 0, 89};
    int i;
    printf("\nBefore:\n");
    for (i = 0; i < 9; i++) printf("%d ", a[i]);
    moveZeroes(a, 9);
    printf("\nAfter:\n");
    for (i = 0; i < 9; i++) printf("%d ", a[i]);
    printf("\n");
    return 0;
}

void moveZeroes(int* nums, int numsSize)
{
    int i, j;
    for (j = 0; j < numsSize && nums[j] != 0; j++);
    for (i = j + 1; i < numsSize; i++)
        if (nums[i] != 0)
        {
            nums[j] = nums[i];
            nums[i] = 0;
            j++;
        }
}
```