数据结构

姓名：杨瑞灵 学号：2252941

1.给定两个单向链表的头指针,判定它们是否相交(为简化问题，假设两个链表均不带环)

（1）请画出链表相交示意图。

- 如图，链表相交只能是第二种情况，因为链表只有一个next指针只能指向一个节点。

（2）给出算法思想，并分析时间复杂度。

- 我的第一想法是：既然他们相交后就会一直相同，只需要比较最后一个节点的地址是否相同。
  - 相同：一定相交
  - 不同：一定没有相交
- 只需要遍历两个链表，假设第一个链表长度是n_1，第二个是n_2
  - 时间复杂度：$O(n_1+n_2)$
  - 空间复杂度：$O(1)$

（3）如何求出两个单链表的第一个交点。

- 在上道题的基础上作出改进，要知道第一个交点，需要从后往前遍历直到遇到不一样地址的节点
  - 因为是单向链表，要从后往前遍历，就必须再开两段连续的空间，data *a[n]和data *b[n]来存储两个链表的指针。
  - 先比对最后的指针，如果相同再依次向前比对，直到遇到第一个不同的地址，那么最后一个相同的就是第一个交点的地址
  - 时间复杂度：$O(n_1+n_2)$
  - 空间复杂度：$O(n_1+n_2)$
- 也有其他同学用第二题的方法进行了解答。先变成环再用龟兔指针

2. 如何判断一个单链表内是否有环?若有环,找到入环的第一个节点

2.1 题目分析：为了保证安全性，肯定不能修改节点里的数据，那么要么用指针，要么新开别的空间。

2.2 问题解答：我们小组讨论了三种不同的解法分别如下

2.2.1 暴力对比

- 外层循环设置P1指针遍历节点。每次经过一个节点记录经过的节点数，并且内层循环重新设置P2指针，从head开始依次遍历到P1，比较两次经过的节点个数
- 由于在最坏的情况下，每一个节点都需要和前面所有节点的地址进行比对，因此
  - 时间复杂度：$O(n^2)$
  - 空间复杂度：$O(1)$

```
int isLoop(LinkList head)
{
    LNode *ptr = head;
    if (ptr == NULL) return ERROR;
    int n = 0;
    for (ptr = head; ptr != NULL; ptr = ptr->next)
    {
        n++;
        int m;
        for(LNode *q = head; q != ptr; q = q->next)
            m++;
        if(n != m)
            return true;
    }
    return false;
}
```

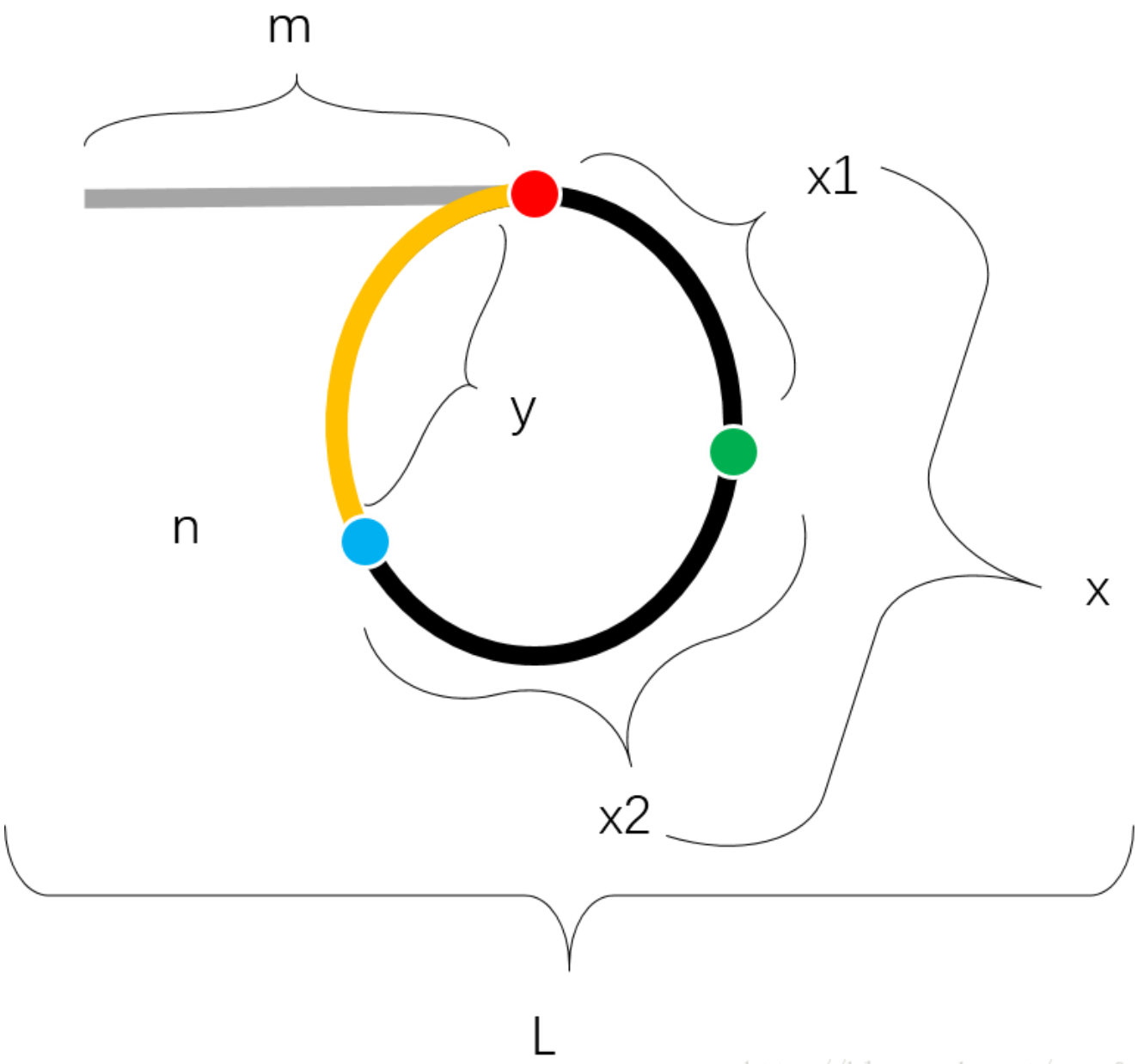
2.2.2 二叉树/容器map set

- 链表中的节点的地址总是不相同的，则可以在遍历过程中将遍历过的节点存储到其他容器中
- 在这个容器中遍历，看是否此地址已经在容器中
  - 如果在则说明这个地址已经被访问过，则原链表是有环的
  - 如果不再就把当前节点的地址存到这个容器中，继续遍历下一个节点
- 复杂度分析
  - 时间复杂度：用二叉树进行构造，每一个节点的最大查找时间是$O(logn)$，由于有n个节点，总的查找时间是 $O(nlogn)$
  - 空间复杂度:由于有另外存储的容器，则会有容器大小的空间开销$O(n)$

```cpp
#include <set>
bool isLoop(LinkList head)
{
    LinkNode* ptr = head;
    set<ListNode*> s;
    while (ptr != NULL)
    {
        if (s.count(ptr)) return true;
        s.insert(ptr);
        ptr = ptr->next;
    }
    return false;
}
```

2.2.3 快慢指针法(龟兔指针？？？)

- 创建两指针指向链表头节点，开始遍历。遍历时快指针 P1 向下移动两个节点，慢指针 P2 向下移动一个节点
  - 如果链表中有环，则当两个指针都进入到这个环当中移动后一定会有一时刻两个指针指向一个节点
  - 如果链表中没有环，则两个指针不会相遇
- 找出节点：再设计一个指针 P3 放在头节点处，慢指针 P2 与 P3 同时开始遍历，它们相遇之处就是入环的节点。
  - 原理：需要简单的数学计算



- 上图是一个简单有环单链表的模型图，总长度为L，无环部分长度（图中灰色部分）为m，环的长度（图中黑色加橙色部分）为n，红点为环的入口节点。
- 下面开始证明一个结论：

$$m = k * n + y.(k\text{为正整数})$$

  - 当快慢指针相遇时，假设慢指针走了t步，那么快指针一定走了2t步。
  - 当慢指针进入以后，快指针一定会在慢指针走完一圈之前追上它。因为最坏的情况是慢指针进入环时快指针刚好落后它一整圈，这样慢指针走完一圈快指针刚好追上慢指针。
  - 快慢指针相遇时，慢指针走过的距离是：

$$t = m + x$$

  - 当慢指针刚进入环入口时，快指针走过的的距离是：

$$t1 = m + k * n + x1.(k为正整数)$$

- 当快慢指针相遇时，快指针又走过了：

$$t2 = n + x2$$

- 所以快慢指针相遇时，快指针一共走过了：

$$\begin{gathered} t1 + t2 \\ = m + k * n + x1 + n + x2 \\ = m + (k + 1) * n + x \\ = m + k * n + x.(k为正整数) \end{gathered}$$

- 而相遇时快指针走过的距离是慢指针的两倍。

$$\begin{gathered} 2 * t = t1 + t2 \\ 2(m + x) = m + k * n + x \\ m = k * n - x \\ m = (k - 1) * n + (n - x) \\ m = k * n + y.(k为正整数) \end{gathered}$$

- 这个结论说明，从头节点到环入口的距离等于快慢指针相遇处继续走到环入口（图中橙色部分）的距离加上环长度的整倍数。
- 如果有一个人从链表头节点开始每次移动一个节点地往后走，另一个人从快慢指针相遇处（图中蓝色点）以同样的速度往前走，结果就是，两人相遇在环的入口节点处。

- 复杂度分析
  - 时间复杂度：O(n) 主要取决于环的大小（若速度差能改变同时取决于速度差），其决定了相遇时间
  - 空间复杂度：指针操作，除指针空间外不会产生额外开销

```
bool isLoop(LinkList head)
{
ListNode* slowptr, fastptr;
slowptr = fastptr = head;
while (fastptr != NULL && fastptr->next != NULL)
{
    slowptr = slowptr->next;
    fastptr = fastptr->next;
//in case of overflow && the linkedlist ends where NULL can be reached
    if (fastptr == NULL || fastptr->next == NULL)
        return false;
    fastptr = fastptr->next;
    if (slowptr == fastptr)
        return true;
}
return false;
}
```

> 参考连接：https://blog.csdn.net/puss0/article/details/78462375

3.医院看病排队管理

3.1 问题分析：

- 这是一个多链表的问题，主要解决的是病人插入不同列表的情况，可能是队尾也可能是开头，且如何分配节点也是问题。

3.2 解决方法：

3.1.1 想法一

- 使用队列的数据结构
- 对于病情不同程度的一系列病人建立不同优先级的队列
- 病人结构体和队列

```
struct Patient
{
Patient() : next(NULL){}
int pri;
string name;
Patient* next;
};
struct waitQueue()
{
waitQueue() : head(NULL), rear(NULL){}
Patient* head;
Patient* rear;
};
vector<waitQueue> sys;
int N = 5; //优先级个数
for (int i = 1; i <= N; i++)
{
Patient* pa = (Patient*)malloc(sizeof(Patient));
sys.push_back(pa);
}
```

- 入队

```
int n; //病人个数
//链表插入
void insert(waitQueue& wq, Patient* pa)
{
pa->next = NULL;
rear->next = pa;
rear = pa;
}
for (int i = 1; i <= n; i++)
{
Patient* pa = (Patient*)malloc(sizeof(Patient));
scanf("%d%s", &pa->pri, &pa->name);
insert(sys[pa->pri - 1], pa);
}
```