

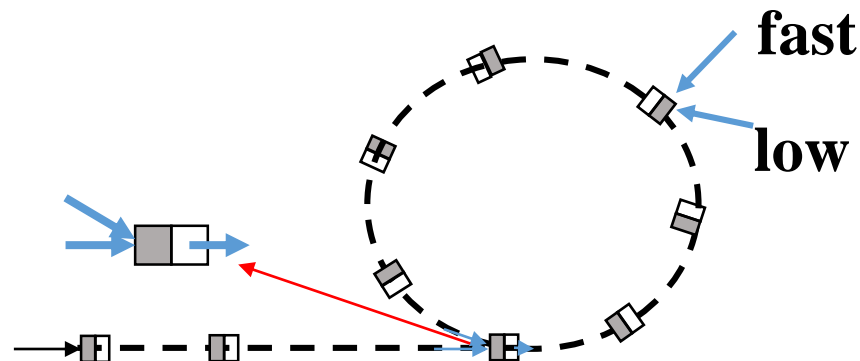
【B-1】线性链表环路问题

问题1：如何判断单链表中是否存在环？

设一快一慢两个指针（Node *fast, *low）同时从链表起点开始遍历，快指针每次移动长度为2，慢指针则为1。

若无环，开始遍历之后fast不可能与low重合，fast或fast->next领先于low到达表尾；

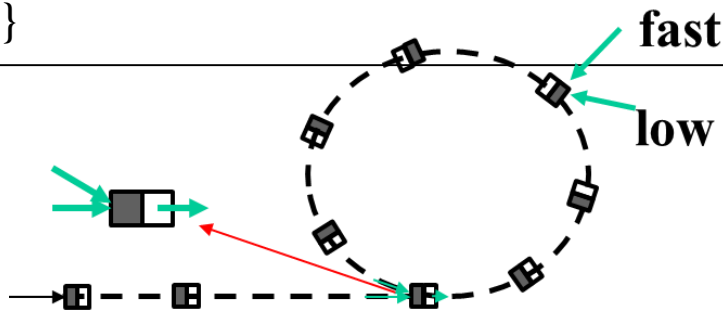
若有环，则fast必然不迟于low先进入环，且由于fast移动步长为2，low移动步长为1，则在low进入环后继续绕环遍历一周之前fast必然能与low重合（且必然是第一次重合）。



```

bool IfCircle(Node* head, Node* &encounter)
{
    Node *fast = head, *slow = head;
    while(fast && fast->next)
    {
        fast = fast->next->next;
        slow = slow->next;
        if(fast == slow)
        {
            encounter = fast;
            return true;
        }
    }
    encounter = NULL;
    return false;
}

```



算法2: 设两个指针p, q, 初始化指向头。p以步长2的速度向前跑, q的步长是1。这样, 如果链表不存在环, p和q肯定不会相遇。如果存在环, p和q一定会相遇。(就像两个速度不同的汽车在一个环上跑绝对会相遇)。复杂度 $O(n)$

```

int any_ring(node_t *head)
{
    node_t *p, *q;
    for (p = q = head; p; p = p->next, q = q->next)
    {
        p = p->next;
        if (!p) break;
        if (p == q) return 1; //yes
    }
    return 0; //fail find
}

```



问题2：若存在环，如何找到环的入口点？

设链起点到环入口点间的距离为 x ，环入口点到 $fast$ 与 low 重合点的距离为 y ，又设在 $fast$ 与 low 重合时 $fast$ 已绕环 n 周 ($n > 0$)，且此时 low 移动总长度为 s ， $fast$ 移动总长度为 $2s$ ，设环的长度为 r 。则：

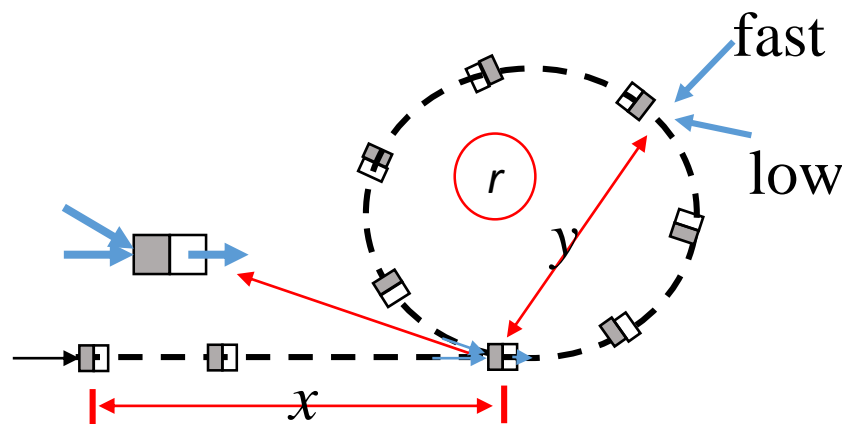
$$s = x + y \quad (1)$$

$$s + nr = 2s, \quad n > 0 \quad (2)$$

由(2)式得 $s = nr$

代入(1)式得


$$x = nr - y \quad (3)$$



现设一指针 $p1$ 从链表起点处开始遍历，指针 $p2$ 从 $encounter$ 处开始遍历， $p1$ 和 $p2$ 移动步长均为 1。当 $p1$ 移动 x 步即到达环的入口点，由(3)式可知，此时 $p2$ 也已移动 x 步即 $nr - y$ 步。由于 $p2$ 是从 $encounter$ 处开始移动，故 $p2$ 移动 nr 步是移回到了 $encounter$ 处，再退 y 步则是到了环的入口点。也即，当 $p1$ 移动 x 步第一次到达环的入口点时， $p2$ 也恰好到达了该入口点。

算法1

```
Node* findEntry1(Node* head, Node* encounter)
{
    Node *p1 = head, *p2 = encounter;
    while(p1 != p2)
    {
        p1 = p1->next;
        p2 = p2->next;
    }
    return p1;
}
```




p扫描的步长为1, q扫描的步长为2。它们的相遇点为图中meet处(环上)。设头指针head到入口点entry之间的距离是K. 当q入环的时候, p已经领先了q为: $d = K \% n$ (n为环的周长)。

meet相对entry的距离(行进方向)为x:
 $(n-d)+x = 2x$ (p行进的路程是q的两倍), 解得 $x = n-d$, 那么当p和q在meet处相遇时, 从head处再发出一个步长为1的指针r, r和q会在entry处相遇! 如算法2。

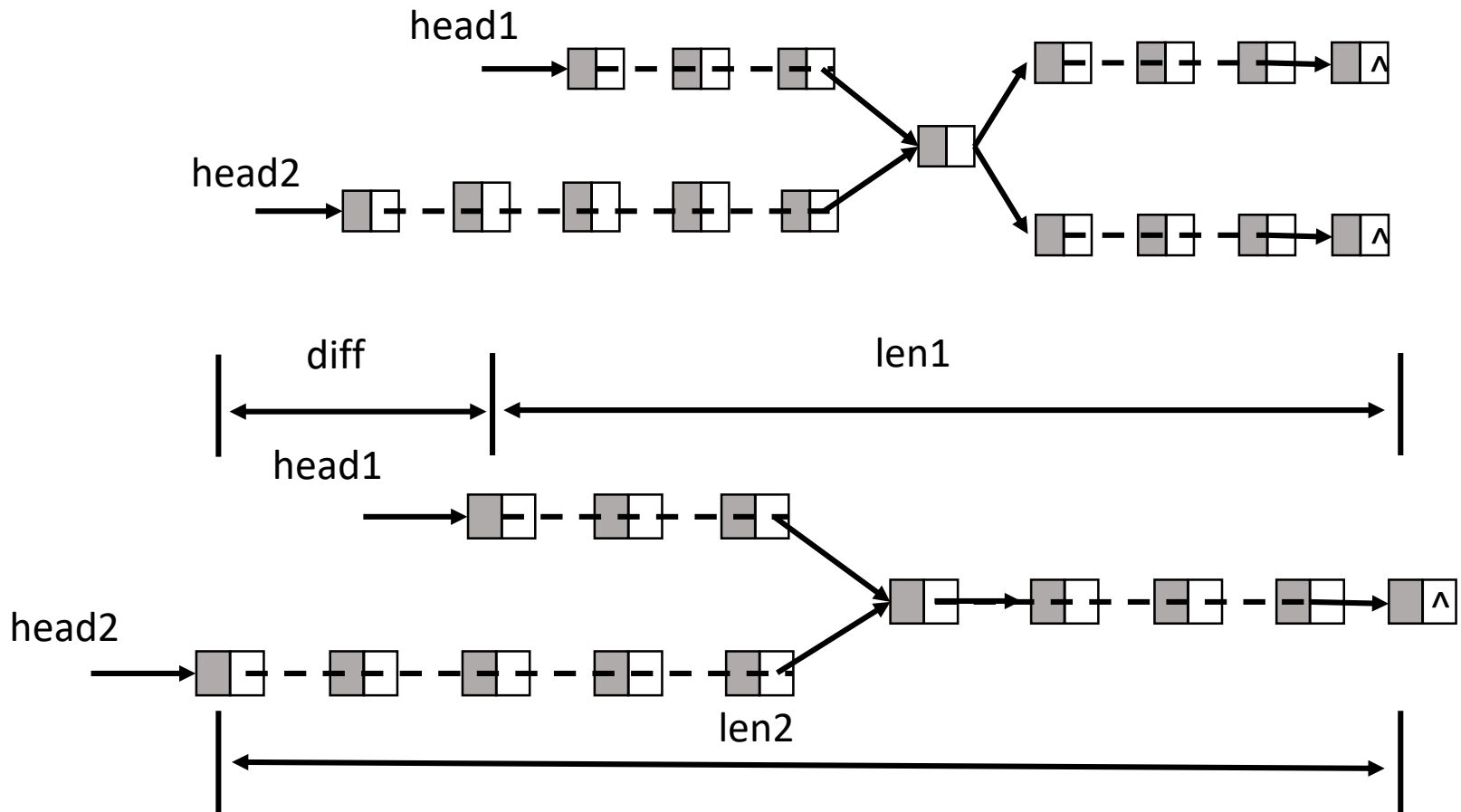
算法2:

初始化三个指针p, q, r全部指向head。然后p以2的速度行进, q以1的速度行进。当p和q相遇的时候, 发出r指针并以1的速度行进, 当p和r相遇返回这个结点。复杂度 $O(n)$

```
node_t *find_entry(node_t *head)
{
    node_t *p, *q, *r;
    for (p = q = head; p; p = p->next, q = q->next)
    {
        p = p->next;
        if (!p) break;
        if (p == q) break;
    }
    if (!p) return 0; //no ring in list
    for (r = head, q = q->next; q != r; r = r->next, q = q->next);
    return r;
}
```



【B-2】线性链表交叉问题



算法1:



```
Node *IfCross(node *head1, node *head2)
{
    node *p1, *p2; int len1=0;int len2=0;int diff = 0;
    if(NULL == head1 || NULL == head2)
        return NULL;    //有为空的链表，不相交
    p1 = head1;
    p2 = head2;
    while(NULL != p1->next)
    {
        p1 = p1->next;
        len1++;
    }
    while(NULL != p2->next)
    {
        p2 = p2->next;
        len2++;
    }

    if(p1 != p2) //最后一个节点不相同,返回NULL
        return NULL;
```

```
diff = abs(len1 - len2);
if(len1 > len2)
{
    p1 = head1;
    p2 = head2;
}
else
{
    p1 = head2;
    p2 = head1;
}
for(int i=0; i<diff; i++)
    p1 = p1->next;
while(p1 != p2)
{
    p1 = p1->next;
    p2 = p2->next;
}
return p1;    //相交入口点
}
```

算法2：判断两个单链表是否相交

两个指针遍历这两个链表,如果他们的尾结点相同,则必定相交.复杂度 $O(m+n)$

```
int is_intersect(node_t *a, node_t *b)
{
    if (!a || !b) return -1;    //a or b is NULL
    for (; a->next; a = a->next);
    for (; b->next; b = b->next);
    return a == b ? 1 : 0; //return 1 for yes, 0 for no
}
```



假设两个链表a,b.a比b长k个结点($k \geq 0$), 那么当a_ptr,b_ptr两个指针同时分别遍历a,b的时候, 必然b_ptr先到达结尾(NULL),而此时a_ptr落后a的尾巴k个结点。

如果此时再从a的头发出一个指针t,继续和a_ptr 一起走,当a_ptr达到结尾(NULL)时,t恰好走了k个结点.此时从b的头发出一个指针s, s和t一起走,因为a比b长k个结点,所以,t和s会一起到达交点。

算法3:

p,q分别遍历链表a,b,假设q先到达NULL,此时从a的头发出一个指针t,当p到达NULL时,从b的头发出s,当s==t的时候即交点。

```
node_t *intersect_point(node_t *a, node_t *b)
{
    //当a,b不相交,函数返回0,否则返回相交结点指针
    node_t *p, *q, *k, *t, *s;
    for (p = a, q = b; p && q; p = p->next, q = q->next);
    k = (p == 0)?q: p;    //k record the pointer not NULL
    t = (p == 0)?b: a;    //if p arrive at tail first, t = b ; else p = a
    s = (p == 0)?a: b;
    for (; k; k = k->next, t = t->next);
    for (; t != s; t = t->next, s = s->next);
    return t;
}
```

