# 16单链表小结：腾讯面试题

## 静态链表的删除操作

我们的故事还没结束，小C看到小A和2B这样非法的勾当，内心觉得很不爽，一句话也不说就离开了队伍。。。。。。

我们先在图上实践一下，然后研究代码：

代码：ListDelete.c

/\* 删除在L中的第i个数据元素 \*/

Status ListDelete(StaticLinkList L, int i)

{

int j, k;

if( i<1 || i>ListLength(L) )

{

return ERROR;

}

k = MAX\_SIZE - 1;

for( j=1; j <= i-1; j++ )

{

k = L[k].cur; // k1 = 1, k2 = 5

}

j = L[k].cur; // j = 2

L[k].cur = L[j].cur;

Free\_SLL(L, j);

return OK;

}

/\* 将下标为k的空闲结点回收到备用链表 \*/

void Free\_SLL(StaticLinkList space, int k)

{

space[k].cur = space[0].cur;

space[0].cur = k;

}

/\* 返回L中数据元素个数 \*/

int ListLength(StaticLinkList L)

{

int j = 0;

int i = L[MAXSIZE-1].cur;

while(i)

{

i = L[i].cur;

j++;

}

return j;

}

## 静态链表优缺点总结

优点：

* + 在插入和删除操作时，只需要修改游标，不需要移动元素，从而改进了在顺序存储结构中的插入和删除操作需要移动大量元素的缺点。

缺点：

* + 没有解决连续存储分配（数组）带来的表长难以确定的问题。
  + 失去了顺序存储结构随机存取的特性。

总的来说，静态链表其实是为了给没有指针的编程语言设计的一种实现单链表功能的方法。

尽管我们可以用单链表就不用静态链表了，但这样的思考方式是非常巧妙的，应该理解其思想，以备不时之需。

## 单链表小结腾讯面试题

题目：快速找到未知长度单链表的中间节点。

既然是面试题就一定有普通方法和高级方法，而高级方法无疑会让面试官大大加分！

普通的方法很简单，首先遍历一遍单链表以确定单链表的长度L。然后再次从头节点出发循环L/2次找到单链表的中间节点。

算法复杂度为：O(L+L/2)=O(3L/2)。

能否再优化一下这个时间复杂度呢？

有一个很巧妙的方法：利用快慢指针！(两个指针同时行进，一个快每次行进两个位置，一个慢每次行进一个位置，当快指针指向单链表的最后则慢指针所指向的即为中间节点)

利用快慢指针原理：设置两个指针\*search、\*mid都指向单链表的头节点。其中\* search的移动速度是\*mid的2倍。当\*search指向末尾节点的时候，mid正好就在中间了。这也是标尺的思想。

我们来看下代码实现：GetMidNode.c

Status GetMidNode(LinkList L, ElemType \*e)

{

LinkList search, mid;

mid = search = L;

while (search->next != NULL)

{

//search移动的速度是 mid 的2倍

if (search->next->next != NULL)

{

search = search->next->next;

mid = mid->next;

}

else

{

search = search->next;

}

}

\*e = mid->data;

return OK;

}

课后作业：写一个完整的程序，实现随机生成20个元素的链表（尾插法或头插法任意），用我们刚才学到的方法快速查找中间结点的值并显示。

程序演示：GetMidNode2.exe

#include "stdio.h"

#define OK 1

#define ERROR 0

#define TRUE 1

#define FALSE 0

typedef int Status; /\* Status是函数的类型,其值是函数结果状态代码，如OK等 \*/

typedef int ElemType; /\* ElemType类型根据实际情况而定，这里假设为int \*/

typedef struct Node

{

ElemType data;

struct Node \*next;

}Node;

typedef struct Node \*LinkList; /\* 定义LinkList \*/

Status visit(ElemType c)

{

printf("%d ",c);

return OK;

}

/\* 初始化顺序线性表 \*/

Status InitList(LinkList \*L)

{

\*L=(LinkList)malloc(sizeof(Node)); /\* 产生头结点,并使L指向此头结点 \*/

if(!(\*L)) /\* 存储分配失败 \*/

{

return ERROR;

}

(\*L)->next=NULL; /\* 指针域为空 \*/

return OK;

}

/\* 初始条件：顺序线性表L已存在。操作结果：返回L中数据元素个数 \*/

int ListLength(LinkList L)

{

int i=0;

LinkList p=L->next; /\* p指向第一个结点 \*/

while(p)

{

i++;

p=p->next;

}

return i;

}

/\* 初始条件：顺序线性表L已存在 \*/

/\* 操作结果：依次对L的每个数据元素输出 \*/

Status ListTraverse(LinkList L)

{

LinkList p=L->next;

while(p)

{

visit(p->data);

p = p->next;

}

printf("\n");

return OK;

}

/\* 随机产生n个元素的值，建立带表头结点的单链线性表L（尾插法） \*/

void CreateListTail(LinkList \*L, int n)

{

LinkList p,r;

int i;

srand(time(0)); /\* 初始化随机数种子 \*/

\*L = (LinkList)malloc(sizeof(Node)); /\* L为整个线性表 \*/

r=\*L; /\* r为指向尾部的结点 \*/

for (i=0; i < n; i++)

{

p = (Node \*)malloc(sizeof(Node)); /\* 生成新结点 \*/

p->data = rand()%100+1; /\* 随机生成100以内的数字 \*/

r->next=p; /\* 将表尾终端结点的指针指向新结点 \*/

r = p; /\* 将当前的新结点定义为表尾终端结点 \*/

}

r->next = NULL; /\* 表示当前链表结束 \*/

// 创建有环链表

//r->next = p;

}

Status GetMidNode(LinkList L, ElemType \*e)

{

LinkList search, mid;

mid = search = L;

while (search->next != NULL)

{

//search移动的速度是 mid 的2倍

if (search->next->next != NULL)

{

search = search->next->next;

mid = mid->next;

}

else

{

search = search->next;

}

}

\*e = mid->data;

return OK;

}

int main()

{

LinkList L;

Status i;

char opp;

ElemType e;

int find;

int tmp;

i=InitList(&L);

printf("初始化L后：ListLength(L)=%d\n",ListLength(L));

printf("\n1.查看链表 \n2.创建链表（尾插法） \n3.链表长度 \n4.中间结点值 \n0.退出 \n请选择你的操作：\n");

while(opp != '0')

{

scanf("%c",&opp);

switch(opp)

{

case '1':

ListTraverse(L);

printf("\n");

break;

case '2':

CreateListTail(&L,20);

printf("整体创建L的元素(尾插法)：\n");

ListTraverse(L);

printf("\n");

break;

case '3':

//clearList(pHead); //清空链表

printf("ListLength(L)=%d \n",ListLength(L));

printf("\n");

break;

case '4':

//GetNthNodeFromBack(L,find,&e);

GetMidNode(L, &e);

printf("链表中间结点的值为：%d\n", e);

//ListTraverse(L);

printf("\n");

break;

case '0':

exit(0);

}

}

}

# 17线性表12

## 循环（取余）链表

循环，顾名思义就是：绕。

打个比方，就是从前山上有座庙，庙里有个老和尚和一个小和尚，有一天老和尚对小和尚说“从前山上有座庙，庙里有个老和尚和一个小和尚，有一天老和尚对小和尚说“从前 。。。。。。

对于单链表，由于每个结点只存储了向后的指针，到了尾部标识就停止了向后链的操作。也就是说，按照这样的方式，只能索引后继结点不能索引前驱结点。

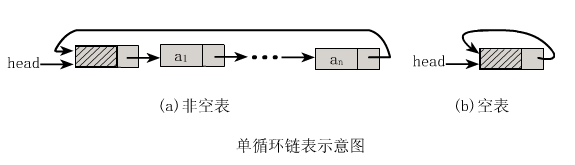
这会带来什么问题呢？

例如不从头结点出发，就无法访问到全部结点。

事实上要解决这个问题也并不麻烦，只需要将单链表中终端结点的指针端由空指针改为指向头结点，问题就结了。

将单链表中终端结点的指针端由空指针改为指向头结点，就使整个单链表形成一个环，这种头尾相接的单链表成为单循环链表，简称循环链表。

No pic you say a J8环节：



注：这里并不是说循环链表一定要有头结点。

其实循环链表的单链表的主要差异就在于循环的判断空链表的条件上，原来判断head->next是否为null，现在则是head->next是否等于head。

回到刚才的问题，由于终端结点用尾指针rear指示，则查找终端结点是O(1)，而开始结点是rear->next->next，当然也是O(1)。

代码实现讲解，由于很多朋友要求小甲鱼把代码讲解一下，咱讲就讲吧~\_~

初始化部分：ds\_init.c

插入部分：ds\_insert.c

删除部分：ds\_delete.c

返回结点所在位置：ds\_search.c

**ds\_init.c**

/\*初始化循环链表\*/

void ds\_init(node \*\*pNode)

{

int item;

node \*temp;

node \*target;

printf("输入结点的值，输入0完成初始化\n");

while(1)

{

scanf("%d", &item);

fflush(stdin); //清除缓冲区

if(item == 0)

return;

if((\*pNode) == NULL)

{ /\*循环链表中只有一个结点\*/

\*pNode = (node\*)malloc(sizeof(struct CLinkList));

if(!(\*pNode))

exit(0);

(\*pNode)->data = item;

(\*pNode)->next = \*pNode;

}

else

{

/\*找到next指向第一个结点的结点\*/

for(target = (\*pNode); target->next != (\*pNode); target = target->next)

;

/\*生成一个新的结点\*/

temp = (node \*)malloc(sizeof(struct CLinkList));

if(!temp)

exit(0);

temp->data = item;

temp->next = \*pNode;

target->next = temp;

}

}

}

**ds\_insert.c**

/\*链表存储结构的定义\*/

typedef struct CLinkList

{

int data;

struct CLinkList \*next;

}node;

/\*插入结点\*/

/\*参数：链表的第一个结点，插入的位置\*/

void ds\_insert(node \*\*pNode , int i)

{

node \*temp;

node \*target;

node \*p;

int item;

int j = 1;

printf("输入要插入结点的值:");

scanf("%d", &item);

if(i == 1)

{ //新插入的结点作为第一个结点

temp = (node \*)malloc(sizeof(struct CLinkList));

if(!temp)

exit(0);

temp->data = item;

/\*寻找到最后一个结点\*/

for(target = (\*pNode); target->next != (\*pNode); target = target->next)

;

temp->next = (\*pNode);

target->next = temp;

\*pNode = temp;

}

else

{

target = \*pNode;

for( ; j < (i-1); ++j )

{

target = target->next;

}

// target指向第三个元素的

temp = (node \*)malloc(sizeof(struct CLinkList));

if(!temp)

exit(0);

temp->data = item;

p = target->next;

target->next = temp;

temp->next = p;

}

}

**ds\_search.c**

/\*返回结点所在位置\*/

int ds\_search(node \*pNode, int elem)

{

node \*target;

int i = 1;

for(target = pNode; target->data != elem && target->next != pNode; ++i)

{

target = target->next;

}

if(target->next == pNode) /\*表中不存在该元素\*/

return 0;

else

return i;

}

**ds\_delete.c**

/\*删除结点\*/

void ds\_delete(node \*\*pNode, int i)

{

node \*target;

node \*temp;

int j = 1;

if(i == 1)

{ //删除的是第一个结点

/\*找到最后一个结点\*/

for(target = \*pNode; target->next != \*pNode;target = target->next)

;

temp = \*pNode;

\*pNode = (\*pNode)->next;

target->next = \*pNode;

free(temp);

}

else

{

target = \*pNode;

for( ; j < i-1; ++j)

{

target = target->next;

}

temp = target->next;

target->next = temp->next;

free(temp);

}

}

GirLL.c

#include <stdio.h>

#include <stdlib.h>

/\*链表存储结构的定义\*/

typedef struct CLinkList

{

int data;

struct CLinkList \*next;

}node;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 操作 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*初始化循环链表\*/

void ds\_init(node \*\*pNode)

{

int item;

node \*temp;

node \*target;

printf("输入结点的值，输入0完成初始化\n");

while(1)

{

scanf("%d", &item);

fflush(stdin);

if(item == 0)

return;

if((\*pNode) == NULL)

{ /\*循环链表中只有一个结点\*/

\*pNode = (node\*)malloc(sizeof(struct CLinkList));

if(!(\*pNode))

exit(0);

(\*pNode)->data = item;

(\*pNode)->next = \*pNode;

}

else

{

/\*找到next指向第一个结点的结点\*/

for(target = (\*pNode); target->next != (\*pNode); target = target->next)

;

/\*生成一个新的结点\*/

temp = (node \*)malloc(sizeof(struct CLinkList));

if(!temp)

exit(0);

temp->data = item;

temp->next = \*pNode;

target->next = temp;

}

}

}

/\*插入结点\*/

/\*参数：链表的第一个结点，插入的位置\*/

void ds\_insert(node \*\*pNode , int i)

{

node \*temp;

node \*target;

node \*p;

int item;

int j = 1;

printf("输入要插入结点的值:");

scanf("%d", &item);

if(i == 1)

{ //新插入的结点作为第一个结点

temp = (node \*)malloc(sizeof(struct CLinkList));

if(!temp)

exit(0);

temp ->data = item;

/\*寻找到最后一个结点\*/

for(target = (\*pNode); target->next != (\*pNode); target = target->next)

;

temp->next = (\*pNode);

target->next = temp;

\*pNode = temp;

}

else

{

target = \*pNode;

for( ; j < (i-1); ++j )

{

target=target->next;

}

temp = (node \*)malloc(sizeof(struct CLinkList));

if(!temp)

exit(0);

temp ->data = item;

p = target->next;

target->next = temp;

temp->next = p;

}

}

/\*删除结点\*/

void ds\_delete(node \*\*pNode, int i)

{

node \*target;

node \*temp;

int j = 1;

if(i == 1)

{ //删除的是第一个结点

/\*找到最后一个结点\*/

for(target = \*pNode; target->next != \*pNode;target = target->next)

;

temp = \*pNode;

\*pNode = (\*pNode)->next;

target->next = \*pNode;

free(temp);

}

else

{

target = \*pNode;

for( ; j < i-1; ++j )

{

target = target->next;

}

temp = target->next;

target->next = temp->next;

free(temp);

}

}

/\*返回结点所在位置\*/

int ds\_search(node \*pNode, int elem)

{

node \*target;

int i = 1;

for(target = pNode; target->data != elem && target->next != pNode; ++i)

{

target = target->next;

}

if(target->next == pNode) /\*表中不存在该元素\*/

return 0;

else

return i;

}

/\*遍历\*/

void ds\_traverse(node \*pNode)

{

node \*temp;

temp = pNode;

printf("\*\*\*\*\*\*\*\*\*\*\*链表中的元素\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

do

{

printf("%4d ", temp->data);

}while((temp = temp->next) != pNode);

printf("\n");

}

int main()

{

node \*pHead = NULL;

char opp;

int find;

printf("------------------- 广 告 位 招 租 --------------------\n\n");

printf("欢迎来到鱼C论坛和大家一起交流互助：http://bbs.fishc.com\n\n");

printf("喜欢鱼C就加入成为终身VIP会员吧：http://fishc.taobao.com\n\n");

printf("-------------------------------------------------------\n\n");

printf("1.初始化链表 \n\n2.插入结点 \n\n3.删除结点 \n\n4.返回结点位置 \n\n5.遍历链表 \n\n0.退出 \n\n请选择你的操作：");

while(opp != '0')

{

scanf("%c", &opp);

switch(opp)

{

case '1':

ds\_init(&pHead);

printf("\n");

ds\_traverse(pHead);

break;

case '2':

printf("输入需要插入结点的位置？");

scanf("%d", &find);

ds\_insert(&pHead, find);

printf("在位置%d插入值后：\n", find);

ds\_traverse(pHead);

printf("\n");

break;

case '3':

printf("输入需要删除的结点位置？");

scanf("%d", &find);

ds\_delete(&pHead, find);

printf("删除第%d个结点后:\n", find);

ds\_traverse(pHead);

printf("\n");

break;

case '4':

printf("你要查找倒数第几个结点的值？");

scanf("%d", &find);

printf("元素%d所在位置：%d\n", find, ds\_search(pHead, find));

//ListTraverse(L);

printf("\n");

break;

case '5':

ds\_traverse(pHead);

printf("\n");

break;

case '0':

exit(0);

}

}

return 0;

}

# 18约瑟夫问题

## 约瑟夫问题

据说著名犹太历史学家 Josephus有过以下的故事：在罗马人占领乔塔帕特后，39个犹太人与Josephus及他的朋友躲到一个洞中，39个犹太人决定宁愿死也不要被敌人抓到，于是决定了一个自杀方式，41个人排成一个圆圈，由第1个人开始报数，每报数到第3人该人就必须自杀，然后再由下一个重新报数，直到所有人都自杀身亡为止。

然而Josephus和他的朋友并不想遵从，Josephus要他的朋友先假装遵从，他将朋友与自己安排在第16个与第31个位置，于是逃过了这场死亡游戏。

## 理论付诸实践

小甲鱼：理论为什么能够付诸实践？

热心鱼油：你TMD在说啥\*&（\*%@……！

小甲鱼：我的意思是约瑟夫问题跟我们讲的循环链表有啥关系？

某女鱼油：它们都带套！

小甲鱼：真聪明，亲一个^\_^

小甲鱼：对的，约瑟夫问题里边41个人是围成一个圆圈，我们的循环链表也是一个圆圈，所以可以模拟并让计算机运行告诉我们结果！

小甲鱼：理论为什么能够付诸实践？

热心鱼油：你TMD在说啥\*&（\*%@……！

小甲鱼：我的意思是约瑟夫问题跟我们讲的循环链表有啥关系？

某女鱼油：它们都带套！

小甲鱼：真聪明，亲一个^\_^

小甲鱼：对的，约瑟夫问题里边41个人是围成一个圆圈，我们的循环链表也是一个圆圈，所以可以模拟并让计算机运行告诉我们结果！

**Josephus.c**

//n个人围圈报数，报m出列，最后剩下的是几号？

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int data;

struct node \*next;

}node;

node \*create(int n)

{

node \*p = NULL, \*head;

head = (node\*)malloc(sizeof (node ));

p = head;

node \*s;

int i = 1;

if( 0 != n )

{

while( i <= n )

{

s = (node \*)malloc(sizeof (node));

s->data = i++; // 为循环链表初始化，第一个结点为1，第二个结点为2。

p->next = s;

p = s;

}

s->next = head->next;

}

free(head);

return s->next ;

}

int main()

{

int n = 41;

int m = 3;

int i;

node \*p = create(n);

node \*temp;

m %= n; // m在这里是等于2

while (p != p->next )

{

for (i = 1; i < m-1; i++)

{

p = p->next ;

}

printf("%d->", p->next->data );

temp = p->next ; //删除第m个节点

p->next = temp->next ;

free(temp);

p = p->next ;

}

printf("%d\n", p->data );

return 0;

}

**answer.c**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_NODE\_NUM 100

#define TRUE 1U

#define FALSE 0U

typedef struct NodeType

{

int id;

int cipher;

struct NodeType \*next;

} NodeType;

/\* 创建单向循环链表 \*/

static void CreaList(NodeType \*\*, const int);

/\* 运行"约瑟夫环"问题 \*/

static void StatGame(NodeType \*\*, int);

/\* 打印循环链表 \*/

static void PrntList(const NodeType \*);

/\* 得到一个结点 \*/

static NodeType \*GetNode(const int, const int);

/\* 测试链表是否为空, 空为TRUE，非空为FALSE \*/

static unsigned EmptyList(const NodeType \*);

int main(void)

{

int n, m;

NodeType \*pHead = NULL;

while (1)

{

printf("请输入人数n（最多%d个）: ", MAX\_NODE\_NUM);

scanf("%d", &n);

printf("和初始密码m: ");

scanf("%d", &m);

if (n > MAX\_NODE\_NUM)

{

printf("人数太多，请重新输入！\n");

continue;

}

else

break;

}

CreaList(&pHead, n);

printf("\n------------ 循环链表原始打印 -------------\n");

PrntList(pHead);

printf("\n-------------删除出队情况打印 -------------\n");

StatGame(&pHead, m);

}

static void CreaList(NodeType \*\*ppHead, const int n)

{

int i, iCipher;

NodeType \*pNew, \*pCur;

for (i = 1; i <= n; i++)

{

printf("输入第%d个人的密码: ", i);

scanf("%d", &iCipher);

pNew = GetNode(i, iCipher);

if (\*ppHead == NULL)

{

\*ppHead = pCur = pNew;

pCur->next = \*ppHead;

}

else

{

pNew->next = pCur->next;

pCur->next = pNew;

pCur = pNew;

}

}

printf("完成单向循环链表的创建!\n");

}

static void StatGame(NodeType \*\*ppHead, int iCipher)

{

int iCounter, iFlag = 1;

NodeType \*pPrv, \*pCur, \*pDel;

pPrv = pCur = \*ppHead;

/\* 将pPrv初始为指向尾结点，为删除作好准备 \*/

while (pPrv->next != \*ppHead)

pPrv = pPrv->next;

while (iFlag)

{

for (iCounter = 1; iCounter < iCipher; iCounter++)

{

pPrv = pCur;

pCur = pCur->next;

}

if (pPrv == pCur)

iFlag = 0;

pDel = pCur; /\* 删除pCur指向的结点，即有人出列 \*/

pPrv->next = pCur->next;

pCur = pCur->next;

iCipher = pDel->cipher;

printf("第%d个人出列, 密码: %d\n", pDel->id, pDel->cipher);

free(pDel);

}

\*ppHead = NULL;

getchar();

}

static void PrntList(const NodeType \*pHead)

{

const NodeType \*pCur = pHead;

if (EmptyList(pHead))

return;

do

{

printf("第%d个人, 密码: %d\n", pCur->id, pCur->cipher);

pCur = pCur->next;

}

while (pCur != pHead);

getchar();

}

static NodeType \*GetNode(const int iId, const int iCipher)

{

NodeType \*pNew;

pNew = (NodeType \*)malloc(sizeof(NodeType));

if(!pNew)

{

printf("Error, the memory is not enough!\n");

exit(-1);

}

pNew->id = iId;

pNew->cipher = iCipher;

pNew->next = NULL;

return pNew;

}

static unsigned EmptyList(const NodeType \*pHead)

{

if(!pHead)

{

printf("The list is empty!\n");

return TRUE;

}

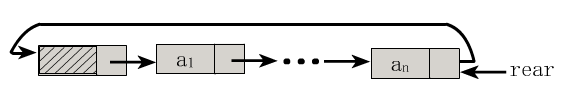
return FALSE;

}

# 19线性表14

## 循环链表的特点

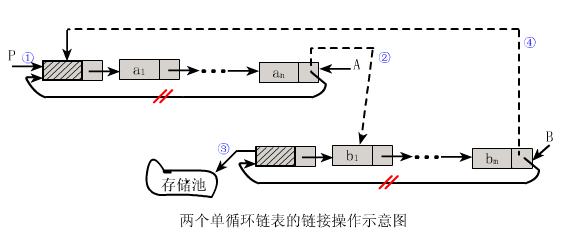
* 回顾一下，在单链表中，我们有了头结点时，我们可以用O(1)的时间访问第一个结点，但对于要访问最后一个结点，我们必须要挨个向下索引，所以需要O(n)的时间。
* 大家猜的没错，如果用上今天我们学习到的循环链表的特点，用O(1)的时间就可以由链表指针访问到最后一个结点。
* 不过我们需要改造一下现有的循环链表，我们不用头指针，而是用指向终端结点的尾指针来表示循环链表，此时查找开始结点和终端结点都很方便了，如图：



* 那么按照这个逻辑的话，判断是否为空链表的条件应该如何呢？
  + 就是判断rear是否等于rear->next，大家猜对了吗？
* 循环链表的特点是无须增加存储量，仅对链接方式稍作改变，即可使得表处理更加方便灵活。

## 一道例题

* **题目：**实现将两个线性表（a1，a2，…，an）和（b1，b2，…，bm）连接成一个线性表（a1，…，an，b1，…bm）的运算。
* **分析：** 
  + 若在单链表或头指针表示的单循环表上做这种链接操作，都需要遍历第一个链表，找到结点an，然后将结点b1链到an的后面，其执行时间是O(n)。
  + 若在尾指针表示的单循环链表上实现，则只需修改指针，无须遍历，其执行时间是O(1)。



稍作思考看代码实现：cennect.c

//假设A，B为非空循环链表的尾指针

LinkList Connect(LinkList A,LinkList B)

{

LinkList p = A->next; //保存A表的头结点位置

A->next = B->next->next; //B表的开始结点链接到A表尾

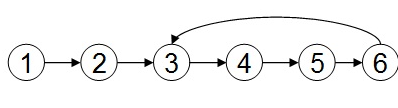
free(B->next); //释放B表的头结点,初学者容易忘记

B->next = p;

return B; //返回新循环链表的尾指针

}

有环的定义是，链表的尾节点指向了链表中的某个节点。



那么要判断单链表中是否有环，主要有以下两种方法。

方法一：使用p、q两个指针，p总是向前走，但q每次都从头开始走，对于每个节点，看p走的步数是否和q一样。如图，当p从6走到3时，用了6步，此时若q从head出发，则只需两步就到3，因而步数不等，出现矛盾，存在环(到了同位置但是步数不等则存在环)。

方法二：快慢指针，使用p、q两个指针，p每次向前走一步，q每次向前走两步，若在某个时候p == q，则存在环。

**Loop.c**

#include "stdio.h"

#define OK 1

#define ERROR 0

#define TRUE 1

#define FALSE 0

typedef int Status;/\* Status是函数的类型,其值是函数结果状态代码，如OK等 \*/

typedef int ElemType;/\* ElemType类型根据实际情况而定，这里假设为int \*/

typedef struct Node

{

ElemType data;

struct Node \*next;

}Node, \*LinkList;

/\* 初始化带头结点的空链表 \*/

Status InitList(LinkList \*L)

{

\*L = (LinkList)malloc(sizeof(Node)); /\* 产生头结点,并使L指向此头结点 \*/

if(!(\*L)) /\* 存储分配失败 \*/

return ERROR;

(\*L)->next=NULL; /\* 指针域为空 \*/

return OK;

}

/\* 初始条件：顺序线性表L已存在。操作结果：返回L中数据元素个数 \*/

int ListLength(LinkList L)

{

int i=0;

LinkList p=L->next; /\* p指向第一个结点 \*/

while(p)

{

i++;

p=p->next;

}

return i;

}

/\* 随机产生n个元素的值，建立带表头结点的单链线性表L（头插法） \*/

void CreateListHead(LinkList \*L, int n)

{

LinkList p;

int i;

srand(time(0)); /\* 初始化随机数种子 \*/

\*L = (LinkList)malloc(sizeof(Node));

(\*L)->next = NULL; /\* 建立一个带头结点的单链表 \*/

for (i=0; i < n; i++)

{

p = (LinkList)malloc(sizeof(Node)); /\* 生成新结点 \*/

p->data = rand()%100+1; /\* 随机生成100以内的数字 \*/

p->next = (\*L)->next;

(\*L)->next = p; /\* 插入到表头 \*/

}

}

/\* 随机产生n个元素的值，建立带表头结点的单链线性表L（尾插法） \*/

void CreateListTail(LinkList \*L, int n)

{

LinkList p,r;

int i;

srand(time(0)); /\* 初始化随机数种子 \*/

\*L = (LinkList)malloc(sizeof(Node)); /\* L为整个线性表 \*/

r = \*L; /\* r为指向尾部的结点 \*/

for (i=0; i < n; i++)

{

p = (Node \*)malloc(sizeof(Node)); /\* 生成新结点 \*/

p->data = rand()%100+1; /\* 随机生成100以内的数字 \*/

r->next=p; /\* 将表尾终端结点的指针指向新结点 \*/

r = p; /\* 将当前的新结点定义为表尾终端结点 \*/

}

r->next = (\*L)->next->next;

}

// 比较步数的方法

int HasLoop1(LinkList L)

{

LinkList cur1 = L; // 定义结点 cur1

int pos1 = 0; // cur1 的步数

while(cur1)

{ // cur1 结点存在

LinkList cur2 = L; // 定义结点 cur2

int pos2 = 0; // cur2 的步数

while(cur2)

{ // cur2 结点不为空

if(cur2 == cur1)

{ // 当cur1与cur2到达相同结点时

if(pos1 == pos2) // 走过的步数一样

break; // 说明没有环

else // 否则

{

printf("环的位置在第%d个结点处。\n\n", pos2);

return 1; // 有环并返回1

}

}

cur2 = cur2->next; // 如果没发现环，继续下一个结点

pos2++; // cur2 步数自增

}

cur1 = cur1->next; // cur1继续向后一个结点

pos1++; // cur1 步数自增

}

return 0;

}

// 利用快慢指针的方法

int HasLoop2(LinkList L)

{

int step1 = 1;

int step2 = 2;

LinkList p = L;

LinkList q = L;

while (p != NULL && q != NULL && q->next != NULL)

{

p = p->next;

if (q->next != NULL)

q = q->next->next;

printf("p:%d, q:%d \n", p->data, q->data);

if (p == q)

return 1;

}

return 0;

}

int main()

{

LinkList L;

Status i;

char opp;

ElemType e;

int find;

int tmp;

i = InitList(&L);

printf("初始化L后：ListLength(L)=%d\n",ListLength(L));

printf("\n1.创建有环链表（尾插法） \n2.创建无环链表（头插法） \n3.判断链表是否有环 \n0.退出 \n\n请选择你的操作：\n");

while(opp != '0')

{

scanf("%c",&opp);

switch(opp)

{

case '1':

CreateListTail(&L, 10);

printf("成功创建有环L(尾插法)\n");

printf("\n");

break;

case '2':

CreateListHead(&L, 10);

printf("成功创建无环L(头插法)\n");

printf("\n");

break;

case '3':

printf("方法一: \n\n");

if( HasLoop1(L) )

{

printf("结论：链表有环\n\n\n");

}

else

{

printf("结论：链表无环\n\n\n");

}

printf("方法二：\n\n");

if( HasLoop2(L) )

{

printf("结论：链表有环\n\n\n");

}

else

{

printf("结论：链表无环\n\n\n");

}

printf("\n");

break;

case '0':

exit(0);

}

}

}

# 20魔术师发牌问题

## 题外话

今天小甲鱼看到到微博有朋友在问，这个《数据结构和算法》系列课程有木有JAVA版本的？

因为这个问题之前也有一些朋友问过，所以咱在这里统一说下哈。

这个你要这么想，我们怀着民族情结看日本AV的时候，我们固然是完全不知道她们在所啥的，这点你要承认，但是我们对于师生，地铁，公车，办公室等的剧情确实了然指掌，完全可以忽略语言的障碍！那么，编程为什么不可以呢？！

## 魔术师发牌问题

问题描述：魔术师利用一副牌中的13张黑牌，预先将他们排好后叠放在一起，牌面朝下。对观众说：“我不看牌，只数数就可以猜到每张牌是什么，我大声数数，你们听，不信？现场演示。”魔术师将最上面的那张牌数为1，把他翻过来正好是黑桃A，将黑桃A放在桌子上，第二次数1,2，将第一张牌放在这些牌的下面，将第二张牌翻过来，正好是黑桃2，也将它放在桌子上这样依次进行将13张牌全部翻出，准确无误。

问题：牌的开始顺序是如何安排的？

请利用循环链表来解决：Magician.c

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 8 | 2 | 5 | 10 | 3 | 12 | 11 | 9 | 4 | 7 | 6 | 13 |

#include <stdio.h>

#include <stdlib.h>

#define CardNumber 13

typedef struct node

{

int data;

struct node \*next;

}sqlist, \*linklist;

linklist CreateLinkList()

{

linklist head = NULL;

linklist s, r;

int i;

r = head;

for(i=1; i <= CardNumber; i++)

{

s = (linklist)malloc(sizeof(sqlist));

s->data = 0;

if(head == NULL)

head = s;

else

r->next = s;

r = s;

}

r->next = head;

return head;

}

// 发牌顺序计算

void Magician(linklist head)

{

linklist p;

int j;

int Countnumber = 2;

p = head;

p->data = 1; //第一张牌放1

while(1)

{

for(j=0; j < Countnumber; j++)

{

p = p->next;

if(p->data != 0) //该位置有牌的话,则下一个位置

{

p->next;

j--;

}

}

if(p->data == 0)

{

p->data = Countnumber;

Countnumber ++;

if(Countnumber == 14)

break;

}

}

}

// 销毁工作

void DestoryList(linklist\* list)

j

}

int main()

{

linklist p;

int i;

p = CreateLinkList();

Magician(p);

printf("按如下顺序排列：\n");

for (i=0; i < CardNumber; i++)

{

printf("黑桃%d ", p->data);

p = p->next;

}

DestoryList(&p);

return 0;

}

## 拉丁方阵问题

* 拉丁方阵是一种n×n的方阵，方阵中恰有n种不同的元素，每种元素恰有n个，并且每种元素在一行和一列中 恰好出现一次。著名数学家和物理学家欧拉使用拉丁字母来作为拉丁方阵里元素的符号，拉丁方阵因此而得名。
* 例如下图是一个3×3的拉丁方阵：

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 2 | 3 | 1 |
| 3 | 1 | 2 |

* 请利用循环链表来解决：latin.c