**数据结构实验报告**

姓名：郭宇瑶 学号：U201717031 班级：软工1704班

**实验三：教材3.10Josephus（约瑟夫环）问题、多项式乘法问题的求解**

1. 问题描述
2. 约瑟夫环问题

N人围成一个圈，每经过K个人就消灭一个人，谁将是最后的幸存者? 。用游标方式的循环链表的方式实现Josephus(n, K)问题的求解过程

1. 多项式乘法

用链表表示多项式，分别在对指数排序和不排序的情况下，写出求两个给定多项式的乘法的函数。其计算复杂程度分别是多少？

1. 问题分析与算法设计
2. 约瑟夫环问题
3. 利用游标方式的循环链表模拟整个过程：定义结构体，成员为本元素编号、上一个元素的索引、下一个元素的索引，节点的数值部分存储整数1至n。通过迭代法进行循环，当p节点被删除时，将p赋值为-1，更改p前节点的下游标和p后节点的上游标。每一重循环删除一个节点，并更新当前节点的索引。当当前节点的索引和当前节点的下游标相同时，意味着数组中只剩下一个元素，即为留下的最后一个元素。
4. 利用递推公式J(n,k) = (J(n−1,k) + k)%n （if n > 1）, J(1,k) = 0 ，

递归或者迭代求出最后一个元素。

1. 多项式乘法

构造两个链表p1、p2，每个链表节点中存放多项式其中一项的系数和次数，以及下一节点的地址。再构造一个总链表p，代表p1、p2相乘后得到的多项式。总链表每一项的系数为对应p1、p2节点系数之积，每一项的次数为对应p1、p2节点次数之和。用两重循环遍历p1、p2,按照规定的计算方法将相乘所得的指数和次数存入总链表p中。更新总链表时先遍历以及建立的总链表p，看看其中是否以及存在同次项，若存在，更改同次项的系数；若不存在重新开辟一个节点表示该项。

1. 实验实现过程
2. 约瑟夫环问题

法1：

1. 定义结构体

Typedef struct

{

int number;

int lastIndex;//上游标

int nextIndex;//下游标

}listNode;

1. 初始化循环链表：数字、上一个元素的索引、下一个元素的索引。对于最后一个元素和第一个元素进行相应的判断和处理。
2. 模拟整个过程：对于给定的n和k，每当找到第k个元素，将本元素赋值为-1，更新前后节点的游标；当最后一个元素的上下游标相等时标志着结束。

法2：利用递推公式，（if n > 1）J(n,k) = (J(n−1,k) + k)%n , J(1,k) = 0 ，递归调用即可。

1. 多项式乘法

（1）定义结构体

typedef struct

{

float coef;//系数

int expn;//指数

}term;

typedef struct Lnode

{

term data;

Lnode \*next;

}\*Link, \*Linklist;

（2）初始化两个链表

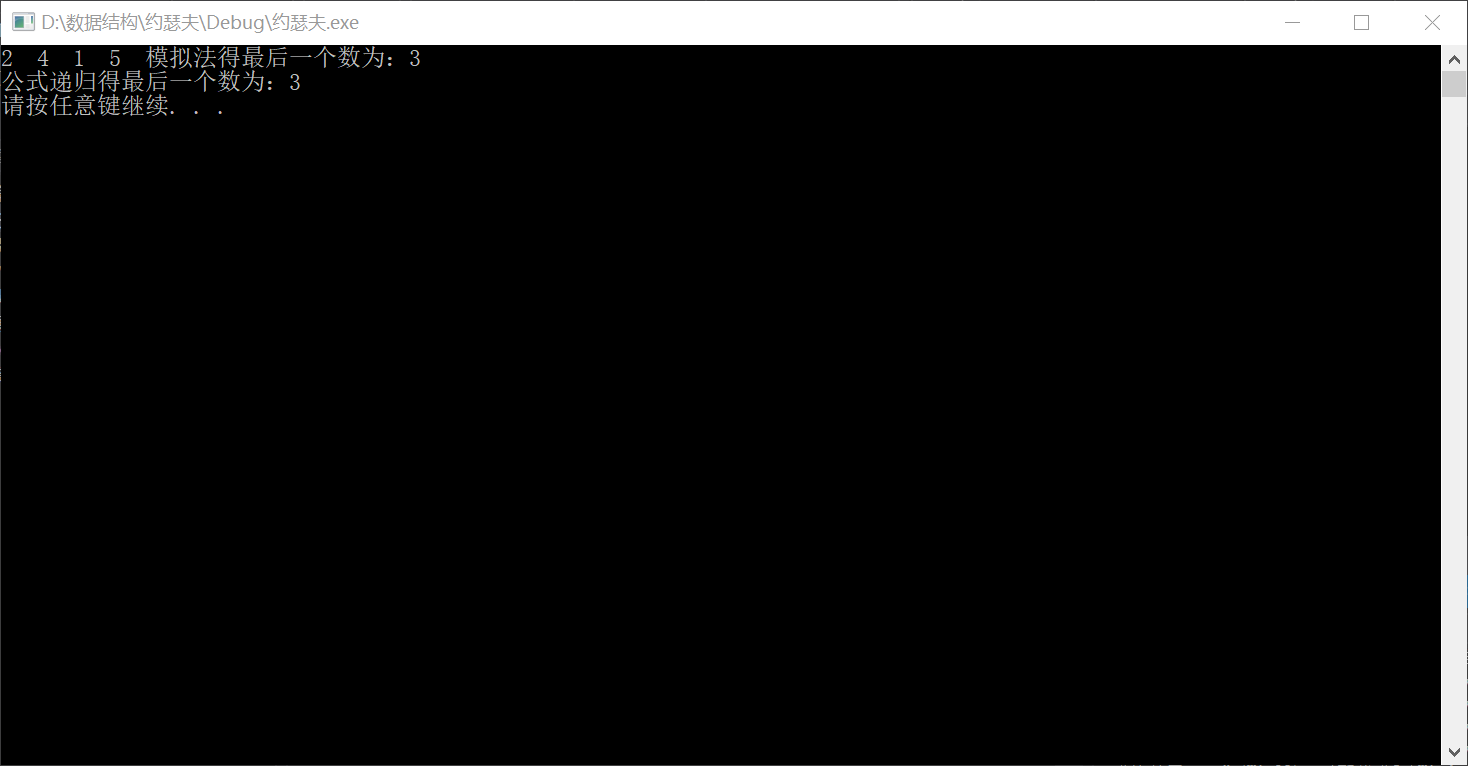
（3）两个链表相乘：对于每一项，P.degree = P1->degree + P2->degree;

遍历 P1, P2 ，如果能找到P.degree则系数直接相加并存储在对应位置；否则 insert一个新的节点来存储相乘的这一项

1. 打印
2. 输入不同的项数，分别为排序和未排序的多项式，循环10000次打印时间
3. 测试结果与分析

1.

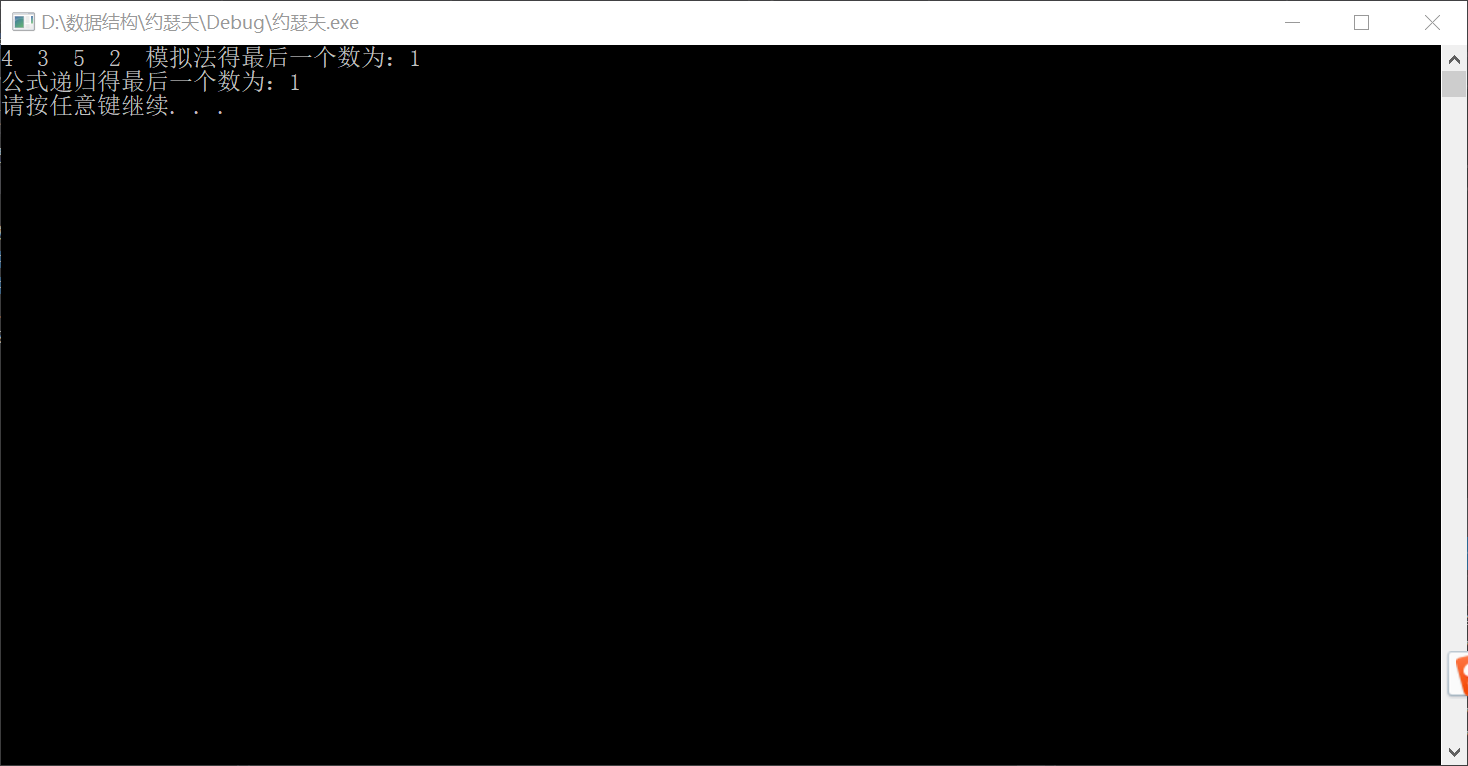
N=5，M=1



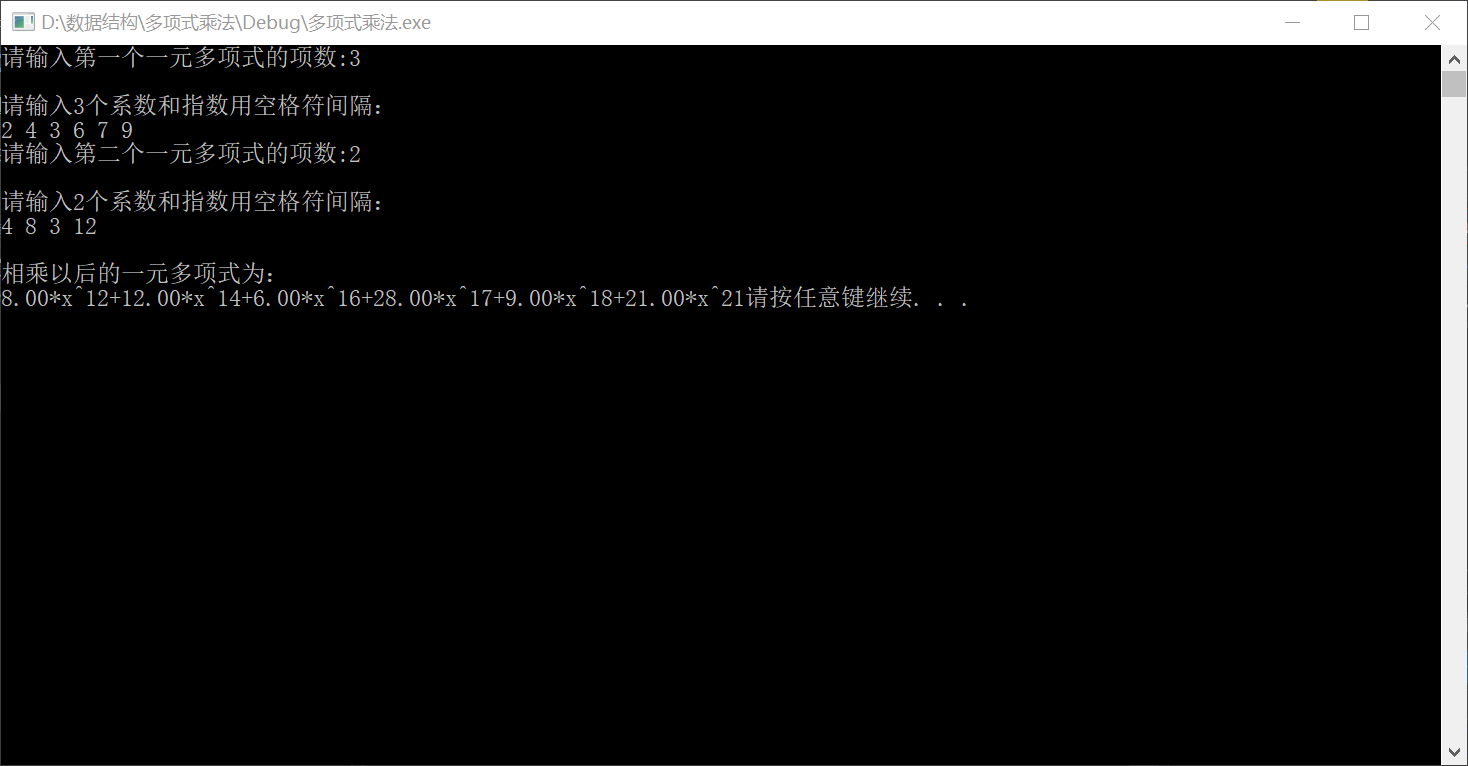
N=5，M=2



N=5，M=3



2.多项式乘法



求时间复杂度：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 未排序 | | | | | |
| n | 2 | 4 | 6 | 8 | 8 |
| m | 3 | 4 | 6 | 8 | 10 |
| m+n | 5 | 8 | 12 | 16 | 18 |
| t | 34 | 132 | 312 | 1106 | 1188 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 已排序 | | | | | |
| n | 2 | 4 | 6 | 8 | 8 |
| m | 3 | 4 | 6 | 8 | 10 |
| m+n | 5 | 8 | 12 | 16 | 18 |
| 时间 | 3 | 4 | 5 | 7 | 8 |

1. 实验结论
2. 约瑟夫环问题

两种方法结论一致。时间复杂度：

1. 链表模拟法：该算法主要有两重循环，外循环遍历整个结构体数组T(N)=O(N)，内循环寻找后m个存活的节点T(N)=O(M)。总体时间复杂度为T(N)=O(N\*M)；
2. 公式递归法：显然该算法时间复杂度为T(N)=O(N)
3. 多项式乘法
4. 指数未排序的多项式：时间复杂度为O（M²N²）
5. 指数已排序的多项式：时间复杂度为O（MN）

六、代码

1.模拟法

#include <stdio.h>

#include<stdlib.h>

#define LIST\_SPACE 50

void init();

int alloc();

void del(int listHead, int value);

void insert(int listHead, int value);

int josephusProblem(int n, int m);

typedef struct data {

int value;

}Data;

typedef struct node {

Data data;

int next;

int pre;

}Node;

Node list\_arr[LIST\_SPACE];

void init() {

for (int i = 0; i < LIST\_SPACE; i++) {

if (i == LIST\_SPACE - 1) {

list\_arr[i].next = 0;

}

else {

list\_arr[i].next = i + 1;

}

list\_arr[i].pre = i - 1;

}

}

void del(int listHead, int value) {

int deleteIndex = -1, currentIndex = listHead;

while (currentIndex != 0) {

if (list\_arr[currentIndex].data.value == value) {

deleteIndex = currentIndex;

if (list\_arr[deleteIndex].next == 0) {

list\_arr[list\_arr[deleteIndex].pre].next = 0;

}

else {

list\_arr[list\_arr[deleteIndex].pre].next = list\_arr[deleteIndex].next;

list\_arr[list\_arr[deleteIndex].next].pre = list\_arr[deleteIndex].pre;

}

list\_arr[deleteIndex].next = 0;

list\_arr[deleteIndex].pre = -1;

int i = 0;

while (list\_arr[i].next != 0) {

i = list\_arr[i].next;

}

list\_arr[i].next = deleteIndex;

list\_arr[deleteIndex].pre = i;

break;

}

}

}

int alloc() {

if (list\_arr[0].next != 0) {

int allocIndex = list\_arr[0].next;

list\_arr[0].next = list\_arr[allocIndex].next;

list\_arr[allocIndex].pre = -1;

list\_arr[allocIndex].next = -1;

return allocIndex;

}

else {

return -1;

}

}

int Joseph(int n, int m)/\*计算约瑟夫环的递归函数\*/

{

if (n == 1)

return n;

else

{

return (Joseph(n - 1, m) + m) % n;

}

}

int josephusProblem(int n, int m) {

init();int currentIndex=0;

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

int allocIndex = alloc();

int head = -1;

if (allocIndex == -1) {

return -1;

}

else {

if (i == 0) {

head = allocIndex;

currentIndex = head;

}

else {

list\_arr[currentIndex].next = allocIndex;

list\_arr[allocIndex].pre = currentIndex;

list\_arr[allocIndex].next = 0;

}

}

}

return -1;

}

//法2

int Joseph(int n, int m)/\*计算约瑟夫环的递归函数\*/

{

if (n == 1 )

return n;

else

{

return (Joseph(n - 1, m) + m) % n;

}

}

int main()

{

printf("模拟法得最后一个数为：%d\n", josephusProblem(8, 2));

int n, m, x;

//scanf("%d %d", &n, &m);

x = Joseph(8, 2);

printf("公式递归得最后一个数为：%d\n", x);

system("pause");

}

2.多项式乘法

#define null 0

#include "stdio.h"

#include "stdlib.h"

#include "math.h"

#include<time.h>

int op;

typedef struct

{

float coef;//系数

int expn;//指数

}term;

typedef struct Lnode

{

term data;

Lnode \*next;

}\*Link, \*Linklist;

int cmp(term a, term b)//将指数从小到大排序

{

if (a.expn == b.expn)

return 0;

else

return (a.expn - b.expn) / abs(a.expn - b.expn);

}

void Orderinsert(Linklist &L, term e, int(\*comp)(term, term))//直接将输入的项数和指数放入链表（没有可以合并的项）

{

Link o, p, q;

q = L;

p = q->next;

while (p&&comp(p->data, e) < 0)

{

q = p;

p = p->next;

}

o = (Link)malloc(sizeof(Lnode));

o->data = e;

q->next = o;

o->next = p;

}

void multPoly(Linklist &L1, Linklist &L2, Linklist &L3)

{

L3 = (Link)malloc(sizeof(Lnode));

L3->next = null;

Link S = L1->next;

term s, e;

while (S)

{

s.coef = S->data.coef;

s.expn = S->data.expn;

Link Q = L2->next;

while (Q)

{

e.coef = s.coef \* Q->data.coef;

e.expn = s.expn + Q->data.expn;

//Orderinsert(L3, e, cmp);

Q = Q->next;

}

S = S->next;

}

}

int LocateElem(Linklist L, term e, Link &s, Link &q, int(\*comp)(term, term))//判断是否有可以合并的项

{

Link p;

s = L;

p = s->next;

while (p&&comp(p->data, e) != 0)

{

s = p;

p = p->next;

}

if (!p)

{

s = q = null;

return 0;

}

else

{

q = p;

return 1;

}

}

void Delnext(Linklist &L, Link s)

{

Link q = s->next;

s->next = q->next;

free(q);

}

void Orderinsertmerge(Linklist &L, term e, int(\*compara)(term, term))//将同一个多项式里相同的项合并

{

Link q, s;

if (LocateElem(L, e, s, q, compara))

{

q->data.coef += e.coef;

if (!q->data.coef)

{

Delnext(L, s);

}

}

else

Orderinsert(L, e, compara);

}

void Creatpolyn(Linklist &p, int m)//输入多项式的系数和指数

{

term e;

int i;

p = (Link)malloc(sizeof(Lnode));//给链表开拓空间

p->next = null;

printf("\n请输入%d个系数和指数用空格符间隔：\n", m);

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

{

scanf\_s("%f%d", &e.coef, &e.expn);

Orderinsertmerge(p, e, cmp);//将输入的多项式中可以合并的项合并

}

}

void add(Linklist &La, Linklist Lb)//合并两个多项式

{

Link qb; term b;

qb = Lb->next;

while (qb)

{

b = qb->data;

Orderinsertmerge(La, b, cmp);

qb = qb->next;

}

//销毁链表Lb

}

void copy(Linklist &Lc, Linklist La)

{

Lc = (Link)malloc(sizeof(Lnode));

Lc->next = null;

Link S = La->next;

term e;

while (S)

{

e.coef = S->data.coef;

e.expn = S->data.expn;

Orderinsertmerge(Lc, e, cmp);

S = S->next;

}

}

void printpolyn(Linklist p)//打印出相加后的多项式

{

Link q;

q = p->next;

int t = 0;

while (q)

{

if (t == 0)

{

if (q->data.coef == 1)

printf("x^%d", q->data.expn);

else if (q->data.coef == -1)

printf("-x^%d", q->data.expn);

else if (q->data.coef > 0)

printf("%.2f\*x^%d", q->data.coef, q->data.expn);

else if (q->data.coef < 0)

printf("%.2f\*x^%d", q->data.coef, q->data.expn);

}

else

{

if (q->data.coef == 1)

printf("+x^%d", q->data.expn);

else if (q->data.coef == -1)

printf("-x^%d", q->data.expn);

else if (q->data.coef > 0)

printf("+%.2f\*x^%d", q->data.coef, q->data.expn);

else if (q->data.coef < 0)

printf("%.2f\*x^%d", q->data.coef, q->data.expn);

}

q = q->next;

t++;

}

}

int main()

{

int x;

Linklist L1, L2, L3;

printf("请输入第一个一元多项式的项数:");

scanf\_s("%d", &x);

Creatpolyn(L1, x);

printf("请输入第二个一元多项式的项数:");

scanf\_s("%d", &x);

Creatpolyn(L2, x);

copy(L3, L1);

/\*add(L3, L2);

printf("\n相加以后的一元多项式为：\n");

printpolyn(L3);\*/

time\_t start, end;

start = clock();

for (int i = 1; i < 10000; i++) {

multPoly(L1, L2, L3);

}

end = clock();

printf("\n相乘以后的一元多项式为：\n");

printpolyn(L3);

printf("time:%d", (int)end - start);

system("pause");

return 0;

}