2016级数据结构第三次上机解题报告（助教版）

A:Mdd的链表(II)

这道题考察的就是链表的基本操作，可以先建立链表然后求得所有元素的个数，这样就可以知道所要删除元素的位置，然后删除这个结点即可。不过还有一种，也就是参考代码这种，只遍历一遍的方法，用两个指针来标记位置，这样效率快的多。

参考代码：

#include <iostream>

#include <fstream>

#define in std::cin

#define out std::cout

struct SqList {

int val;

SqList \*next;

SqList(int val=-1): val(val), next(nullptr){}

};

void show(SqList \*head) {

if (head) {

out << head->val;

head = head->next;

}

while (head) {

out << "->" << head->val;

head = head->next;

}

}

void del(SqList \*head) {

while (head) {

SqList\* temp = head;

head = head->next;

delete (temp);

}

}

int main() {

int n, val;

SqList \*list, \*tail;

list = new SqList();

tail = list;

in >> n;

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

in >> val;

SqList \*temp = new SqList(val);

tail->next = temp;

tail = temp;

}

SqList \*ans = list;

while (in >> val) {

SqList \*temp = new SqList(val);

tail->next = temp;

tail = temp;

ans = ans->next;

}

out << ans->next->val << "\n";

ans->next = ans->next->next;

show(list->next);

del(list);

return 0;

}

B:Mdd玩炉石

这道题的核心是反转链表，不过加上了元素个数的限制。所以我们只要加上判断，然后递归实现就行。

参考代码

//mdd的炉石

#include <iostream>

#include <fstream>

using namespace std;

#define in std::cin

#define out std::cout

//ifstream in;

//ofstream out;

struct SqList {

int val;

SqList \*next;

SqList(int x=-1) : val(x), next(nullptr) {}

};

SqList\* reverse(SqList\* head, int k) {

int cnt = 0;

SqList \*tail = head;

while (tail) {

cnt++;

tail = tail->next;

if (cnt == k)

break;

}

if (cnt < k)

return head;

SqList \*new\_head = reverse(tail, k);

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

SqList \*tmp = head->next;

head->next = new\_head;

new\_head = head;

head = tmp;

}

return new\_head;

}

void show(SqList \*head) {

while (head) {

out << head->val << " ";

head = head->next;

}

out << "\n";

}

void del(SqList \*head) {

while (head) {

SqList \*tmp = head;

head = head->next;

delete tmp;

}

}

int main() {

// in.open("hearth\_stone\_in5.txt", std::ios::in);

// out.open("hearth\_stone\_out5.txt", std::ios::out);

int k, val;

in >> k;

SqList \*head = new SqList();

SqList \*tail = head;

while (in >> val) {

SqList \*tmp = new SqList(val);

tail->next = tmp;

tail = tmp;

}

head = reverse(head->next, k);

show(head);

del(head);

return 0;

}

C

思路：

将队员标号0,1,2,…n-1,轻易得出每次得到球的队员标号为f(x)=x\*(x+1)/2(mod n),则若是所有队员都能分到球则f(x)必然为n的完全剩余系，即不存在x!=y，使得f(x)=f(y)，下面来根据n的值来讨论f(x)是否为n的完全剩余系：

首先给出三个显然的结论

1、任何一个偶数都可以表示成b\*2^a的形式

2、(x+y+1)和(x-y)必定一奇一偶

3、若x\*y=2^m，则x=2^m1,y=2^m2(0<=m1,m2<=m)

若f(x)=f(y)(不妨令0 <= y < x < n)，则x\*(x+1)/2-y\*(y+1)/2=0(mod n)，不妨令g(x,y)=x\*(x+1)/2-y\*(y+1)/2=(x+y+1)\*(x-y)/2，下面就n的值来讨论g(x,y)=tn是否有解

1、当n为奇数时

令x=y+2，则2y+3=n,此时显然存在x=(n+1)/2,y=(n-3)/2(n=1,3时显然f(x)不是n的完全剩余系）,使得g(x,y)=n成立，故此时f(x)不是n的完全剩余系

2、当n为偶数且不能被表示成2^a形式时，令n=b\*2^a(a>=1,b>1且b为奇数)

(1)当b<2^(a+1)时,不妨令x+y+1=2^(a+1),x-y=b,解此方程组得x=2^a+(b-1)/2 < b\*2^a,y=2^a-(b+1)/2 >= 0,显然存在合适的x,y使得g(x,y)=n成立

(2)当b>2^(a+1)时,不妨令x+y+1=b,x-y=2^a,解此方程组得x=2^a+(b-1)/2 < b\*2^a,y=(b-1)/2-2^a>=0,显然存在合适的x,y使得g(x,y)=n成立

综(1)(2)可知此时g(x,y)=n有解，即f(x)不是n的完全剩余系

3、当n为偶数且能被表示成2^a形式时

若(x+y+1) \* (x-y) / 2=tn,显然t可以表示成c \* 2^d(c为奇数,d>=0),故原式可转化为(x+y+1)\*(x-y)=c\*2^(a+d+1),由结论2可知x+y+1与x-y一奇一偶，下面分两种情况讨论

(1)x+y+1为奇,x-y为偶，此时x+y+1=c,x-y=2^(a+d+1),由y < x < n=2^a知此方程组无解

(2)x+y+1为偶,x-y为奇，此时x+y+1=2^(a+d+1),x-y=c,仍然由y < x < n=2^a可知x+y+1<=2^(a+1)-3<2^(a+d+1),故此方程组无解

综(1)(2)可知此时g(x,y)=tn无解，即f(x)时n的完全剩余系

综上所述，只有当n=2^a时，f(x)才是n的完全剩余系

代码：

#include <cstdio>

int main()

{

long long n;

while(scanf("%lld",&n)!=EOF)

{

if(!(n&(n-1)))

printf("GzhIsSoHandsome\n");

else

printf("GzhIsHandsome\n");

}

return 0;

}

D ModricWang’s JOSEPHUS Problem

对于约瑟夫问题当前实现方法大概有两种：

一：模拟：

链表模拟：

#include<stdio.h>

#include<malloc.h>

typedef struct List

{

int data;

struct List \*next;

}LinkList;

int main()

{

LinkList \*L,\*r,\*s;

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

r =L;

int n,i;

int k;

scanf("%d%d",&n,&k);

for(i = 1;i<=n;i++) ///尾插法建立循环链表

{

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

s->data = i;

r->next = s;

r= s;

}

r->next =L->next; //让最后一个链表的下一个节点指向开头

LinkList \*p;

p = L->next;

while(p->next != p) //开始模拟（判断条件要注意：因为最后肯定剩下一个人， 所以最后一个数据的next还是他本身）

{

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

{

p = p->next; //每k个数死一个人

}

p->next = p->next->next; //将该节点从链表上删除。

p = p->next;

}

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

return 0;

}

数组模拟：

#include<stdio.h>

int main()

{

int n, k;

scanf("%d%d", &n, &k);

int i;

int a[1001];

int dead = 0; //表示已经死了多少人

int num = 0; //num模拟没有被杀的人的喊数

for (i = 1; i<=n; i++) //开始时每个人都可以报数，为了能得到最后一个人的编号，我们让初始值为i下标

{

a[i] = i;

}

for (i = 1;; i++)

{

if (i > n)

{

i = i%n; //如果大于总人数，我们就从头开始

}

if (a[i] > 0) //如果当前这个人没有死，就报数

num++;

if (k == num && dead != n-1) //如果当前这个人报的数等于k 并且没有已经死亡n-1个人

{

num = 0;

a[i] = 0;

dead++;

}

else if(k == num && dead == n-1) //如果这个人报数等于k，并且已经死了n-1个人，说明当前这个人就是最后的一个活着的了。。

{

printf("%d", a[i]);

break;

}

}

return 0;

}

二、公式法（即递推）：

递推过程：

(1)第一个被删除的数为（m-1）%n;

(2)设第二次的开始数字为k，

做下映射：（即将数字的排列计算还是从0开始）

k--->0

k+1--->1

k+2--->2

--- ---

k-2--->n-2

此时剩下n-1个人 ，假如我们已经知道了n-1个人时，最后胜利者的编号为x，利用映射关系逆推，就可以得出n个人时，胜利者的编号为(x+k)%n（要注意的是这里是按照映射后的序号进行的）

其中k=m%n。

代入

（x+k）%n<=>(x+(m%n))%n<=>(x%n + (m%n)%n)%n<=> (x%n+m%n)%n <=> (x+m)%n

(3)第二个被删除的数为（m-1）%n-1

(4)假设第三轮的开始数字为o，那这n-2个数构成的约瑟夫环为o,o+1,o+2,...,o-3,o-2。

映射

o--->0

o+1--->1

o+2--->2

--- ---

o-2--->n-3

这是一个n-2个人的问题。假设最后胜利者为y，那么n-1个人时，胜利者为(y+o)%(n-1)，其中o等于m%(n-1)。代入可得(y+m)%(n-1)

要得到n-1个人问题的解，只需要得到n-2个人问题的解，倒退下去。只有一个人时，胜利者就是编号0.小面给出递推式：

f(1)=0;

f(i)=(f[i-1]+m)%i;(i>1)

这个公式的思想：

现在假设n=10

0 1 2 3 4 5 6 7 8 9

k=3

第一个人出列后的序列为：

0 1 3 4 5 6 7 8 9

即: 3 4 5 6 7 8 9 0 1（1式）

我们把该式转化为: 0 1 2 3 4 5 6 7 8 (2式)

则你会发现: （(2式)+3）%10则转化为(1式)了

也就是说，我们求出9个人中第9次出环的编号，最后进行上面的转换就能得到10个人第10次出环的编号了

设f(n,k,i)为n个人的环，报数为k，第i个人出环的编号，则f(10,3,10)是我们要的结果

当i=1时， f(n,k,i) = (n+k-1)%n

当i!=1时， f(n,k,i)= ( f(n-1,k,i-1)+k )%n

#include<stdio.h>

int main()

{

int n, m,i,s=0;

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

for(i=2;i<=n;i++)

s=(s+m)%i;

printf("%d", s+1);

return 0;

}

说一下：

for(i=2;i<=n;i++)

s=(s+m)%i;

这个式子：

首先从2开始，因为1个人的时候报的数字的人为0号，结果已经确定了。不需要从i=0开始，要注意的是序列从0开始编号的，所以最后的输出结果也要加1.

s表示的是上一轮的结果，m代表是每多少个人出列一次，i代表当前已经出列了多少个人。

整个式子就是根据上一个的出列数和已经出列的人数来算的。

E：

思路：

单调栈，从头到尾遍历数组。对每一个元素，记录元素大小和一个标记数值，表示他会在第几轮被杀（如果为0则不会被杀），先pop掉栈中所有比当前元素小的元素，如果此时栈为空，则没有人能杀掉现在这个人，如果有pop出的元素，但是栈中还有元素，则当前元素会在pop出的元素都被杀掉后被杀，也就是他的标记数值为pop出的元素的最大标记数值+1，如果没有元素被pop出来，栈中还有元素，那么现在这个人会被栈顶之人在第1轮杀掉，最后把这个元素push进栈（包括元素大小和标记数值）。扫完一遍，所有标记数值的最大值即为答案。

代码：

#include <cstdio>

#include <stack>

using namespace std;

stack<pair<int,int> > st;

int main()

{

int n,x,ans,t;

while(scanf("%d",&n)!=EOF)

{

ans=0;

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

{

scanf("%d",&x);

t=0;

while(!st.empty()&&x>st.top().first)

{

if(t<st.top().second)

t=st.top().second;

st.pop();

}

if(st.empty())

t=-1;

if(t+1>ans)

ans=t+1;

st.push(make\_pair(x,t+1));

}

printf("%d\n",ans);

while(!st.empty())

st.pop();

}

return 0;

}

F ModricWang’s JOSEPHUS Problem II

从第一个好人开始从1进行编号，那么根据要求，在2N个人被处决剩下N个人之前，所有被处决的人的编号都应该是大于N的。如果对于一个具体的数值M，它是答案的必要条件是面对这2N个人，第一个被处决的人的编号大于N，之后，面对这2N-1个人，被处决的人的编号也应该大于N。这个题目不关心具体哪一个人被处决，而只关心被处决的人的位置。因此M的值可以由N+1遍历。然后检验这个M是否满足条件，若不满足，令M取下一个数，重新检验，否则，找到了答案。代码如下：

#include<cstdio>

#include<cstring>

int main() {

int n;

scanf("%d", &n);

for(int m = 2;; ++m) {

bool flag = 1;

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

int a = i;

for(int j = n + 1; j <= n + n; ++j) a = (a + m - 1) % j + 1;

if(a > n) {

flag = 0;

break;

}

}

if(flag) return printf("%d\n", m), 0;

}

}