西安電子科技大學

XIDIAN UNIVERSITY

数据结构上机练习

数据结构第一次上机实验报告

陈德创

19030500217

计算机科学与技术专业

指导教师 王凯东

October 6, 2020

数据结构第一次上机实验报告

陈德创　　19030500217

西安电子科技大学

日期：2020 年 10 月 6 日

目录

1 小组名单

学号	姓名	工作
19030500217	陈德创	完成程序、联系例程、调试、小组讨论

2 约瑟夫环

2.1 需求分析

2.1.1 题目描述

编号为 $1,2,...,n$ 的 n 个人按顺时针方向围坐一圈，每人持有一个密码（正整数）。现在给定一个随机数 $m > 0$，从编号为 1 的人开始，按顺时针方向 1 开始顺序报数，报到 m 时停止。报 m 的人出圈，同时留下他的密码作为新的 m 值，从他在顺时针方向上的下一个人开始，重新从 1 开始报数，如此下去，直至所有的人全部出圈为止。

2.1.2 题目分析

经典的约瑟夫问题，分析如下：

1). 数据结构

因为所有人坐成一圈，所以采用循环链表。

2). 算法梗概

模拟取数过程，即从当前结点（最初为 1 号结点）遍历 m 个结点，删除末位置元素并更新 m 值。如此往复，直至链表为空。

3). 小优化

显然，对于 n 个元素的循环链表来说，前移 m 个元素和前移 $m \mod n$ 个元素并没有什么区别，所以我们每次只用进行 $m \mod n$ 次前移操作就可以了，在密码很大的情况下，可以极大地提高程序运行效率。但是要注意 m 是 n 的倍数的情况，在这种情况下，我们置 $m = n$ 而不是 $m = 0$。

2.2 程序设计

2.2.1 整体概览

```
1  // 链表结点
2  typedef struct Node{
3      int data;    // 密码
4      int poi;     // 编号
5      struct Node* next;  // 下一个
6  } *LNode, Node;
7
```

```
/*
  根据所给数组创建一个循环链表
  a: 数组, n: 数组长度
  return: 创建的循环链表的首元结点
*/
LNode Build(const int* a, int n);

/*
  删除链表从head开始数的第m个元素, 并输出其编号
  head: 链表当前位置, len: 链表元素个数, m: 步数
  return: 所删除结点的密码值
*/
int Remove(LNode& head, int len, int m);
```

2.2.2 具体分析

Build 函数　此函数将会创建一个循环链表。我们先按照创建单链表的方式创建，最后将末尾元素的 *next* 指针指向首元结点即可。注意，在我的具体实现中，返回的指针实质上是指向末尾元素的，这样方便此题目的后续操作。

```
LNode Build(const int* a, int n){
    // 头结点
    LNode head = (LNode) malloc(sizeof(Node));
    LNode now = head;
    for (int i = 1; i <= n; ++i){
        // 新插入的结点信息
        LNode node = (LNode) malloc(sizeof(Node));
        node->data = a[i];
        node->poi = i;
        // 将结点插入到上一结点之后
        now->next = node;
        now = node;
    }
    // 将最后一个结点的下一个指向首元结点, 形成循环
    now->next = head->next;
    // 释放空间
    free(head);
    return now;
}
```

Remove 函数　此函数将会迭代链表 m 次，并删除最后迭代到的元素。由于我们是单链表，所以要删除当前指针指向的元素是有些困难的，因为难以寻找当前结点的前继结点。但是，我们删除当前结点后一个结点是非常方便的。所以我们考虑迭代 $m-1$ 次，然后删除后一个结点。

但是这样当 $m=0$ 的时候会出现一点小问题，但是由于是循环链表，所以当 $m=0$ 时我们置 $m=n$，其中 n 为链表元素个数即可。

```
// 返回密码
int Remove(LNode& head, int len, int m){
  // 这里简化迭代次数，但是直接%len可能会产生0
  // 这样写当m整除len时，会得到n，否则不变
  m = (m - 1 + len) % len + 1;
  // 迭代m-1次
    for (int i = 1; i < m; ++i){
        head = head->next;
  }
  // 删除下一个结点
    LNode tmp = head->next;
  head->next = tmp->next;
  // 输出要删除的结点的编号
  printf("%d ", tmp->poi);
  // 记录更新m
    m = tmp->data;
  free(tmp);
    return m;
}
```

2.3 程序分析

时间复杂度 显然，我们一共要删除 n 次，对于第 i 次删除，我们要迭代 $min(n-i+1,m)$ 个元素。所以总体的时间复杂为 $O(n^2)$。

空间复杂度 我们用链表储存，对于每个花费常量空间，一共有 n 个人，所以总体的空间复杂度为 $O(n)$。

3 多项式加减法

3.1 需求分析

3.1.1 题目描述

给定两个多项式，求解其和与差。多项式的项数为 M，而最高幂次为 N。($1 <= M <= 10, 1 <= N <= 1000000$)

3.1.2 题目分析

注意到 $N <= 1000000$，很小，远在计算机空间所能承受的范围内。所以我们采用桶思想，即我们用数组下表模拟多项式的次数，数组元素代表多项式的系数。这样我们开两个 1000000 的数组就可以表示两个多项式，同样再开两个表示答案。对于多项式的加减法，直接对位相加/相减即可。最后输出时注意一下如果系数不为零就输出。

3.2 程序设计

3.2.1 整体概览

```
// 输入，传入一个数组，会将多项式输入到数组中
inline int in(int* t);
// 输出，传入一个数组，和数组中不为零的元素个数（在主函数中可一并求得）
inline void out(const int* t, int m);
// 主函数，整体调度并且完成多项式的加、减
int main();
```

3.2.2 具体分析

输入 每次读入两个数，我们以次数为下标，系数为元素储存。

```
// 传入一个数组
inline void in(int* t){
  int m;
  scanf("%d", &m);
  // 读入系数和次数
  for (int i = 1, x, y; i <= m; ++i){
    scanf("%d%d", &x, &y);
    // 次数为下标，元素为系数
    t[y] += x;
  }
}
```

主程序 主程序主要是对多项式加减法的处理，注意处理的时候维护一下结果中系数不为零的项数，在答案输出的时候要用得到。

```
  // 储存结果中系数不为零的项数
  int cnt1 = 0, cnt2 = 0;
    for (int i = MAXN - 1; i >= 0; --i){
    // 模拟加减法，记录结果
        r1[i] = a[i] + b[i];
    r2[i] = a[i] - b[i];
    // 记录系数不为零的项数
        cnt1 += (0 || r1[i]);
        cnt2 += (0 || r2[i]);
    }
```

3.3 程序复杂度

时空复杂度都是定死的，为 $O(1000000)$，即题目中给定次数的最大值。

4 总结

本以为有着之前的基础，写起来会很轻松。但是发现很多细节还是不会处理，总想着写的完美一些，以至于一行代码都写不出来。这是病，得改。不过好在最后在认真学习了相关知识后，顺利完成了作业。在这里也很感谢同学的答疑解惑。

总而言之，切勿眼高手低。

附录

A 约瑟夫环源码

```cpp
// 2020_10_04_约瑟夫环.cpp
// Created by RainCurtain on 2020/10/4.
//

#include <cstdio>
#include <cstdlib>

typedef struct Node{
    int data;   // 密码
    int poi;    // 编号
    struct Node* next;  // 下一个
} *LNode, Node;

LNode Build(const int* a, int n){
    // 头结点
    LNode head = (LNode) malloc(sizeof(Node));
    LNode now = head;
    for (int i = 1; i <= n; ++i){
        // 新插入的结点信息
        LNode node = (LNode) malloc(sizeof(Node));
        node->data = a[i];
        node->poi = i;
        // 将结点插入到上一结点之后
        now->next = node;
        now = node;
    }
    // 将最后一个结点的下一个指向首元结点，形成循环
    now->next = head->next;
    // 释放空间
    free(head);
    return now;
}

// 返回密码
int Remove(LNode& head, int len, int m){
    m = (m - 1 + len) % len + 1;
    for (int i = 1; i < m; ++i){
        head = head->next;
    }
    LNode tmp = head->next;
    head->next = tmp->next;
    printf("%d ", tmp->poi);
```

```
43      m = tmp->data;
44      free(tmp);
45      return m;
46  }
47
48  int a[105];
49  int main(){
50      int n, m;
51      scanf("%d%d", &n, &m);
52      for (int i = 1; i <= n; ++i){
53          scanf("%d", a + i);
54      }
55      LNode list = Build(a, n);
56      for (int i = 1; i <= n; ++i){
57          m = Remove(list, n - i + 1, m);
58      }
59      return 0;
60  }
```

B 多项式加减法源码

```cpp
// 2020_10_06_dxsjjf.cpp
// Created by RainCurtain on 2020/10/6.
//

#include <cstdio>
const int MAXN = 1000005;

int a[MAXN], b[MAXN], r1[MAXN], r2[MAXN];

inline void in(int* t){
    int m;
    scanf("%d", &m);
    for (int i = 1, x, y; i <= m; ++i){
        scanf("%d%d", &x, &y);
        t[y] += x;
    }
}

inline void out(const int* t, int m){
    printf("%d ", m);
    for (int i = MAXN - 1; i >= 0; --i) if(t[i]) {
        printf("%d %d ", t[i], i);
    }
    printf("\n");
}

int main(){
    in(a), in(b);
    int cnt1 = 0, cnt2 = 0;
    for (int i = MAXN - 1; i >= 0; --i){
        r1[i] = a[i] + b[i];
        r2[i] = a[i] - b[i];
        cnt1 += (0 || r1[i]);
        cnt2 += (0 || r2[i]);
    }
    out(r1, cnt1);
    out(r2, cnt2);
    return 0;
}
```