春雨惊春 清谷天
夏满芒夏 暑相连
秋处露秋 寒霜降
冬雪雪冬 小大寒

惟楚有材，於斯爲盛

数据结构

- 编写效率更高的程序解决新的设计问题
- 有效考虑时间和空间限制

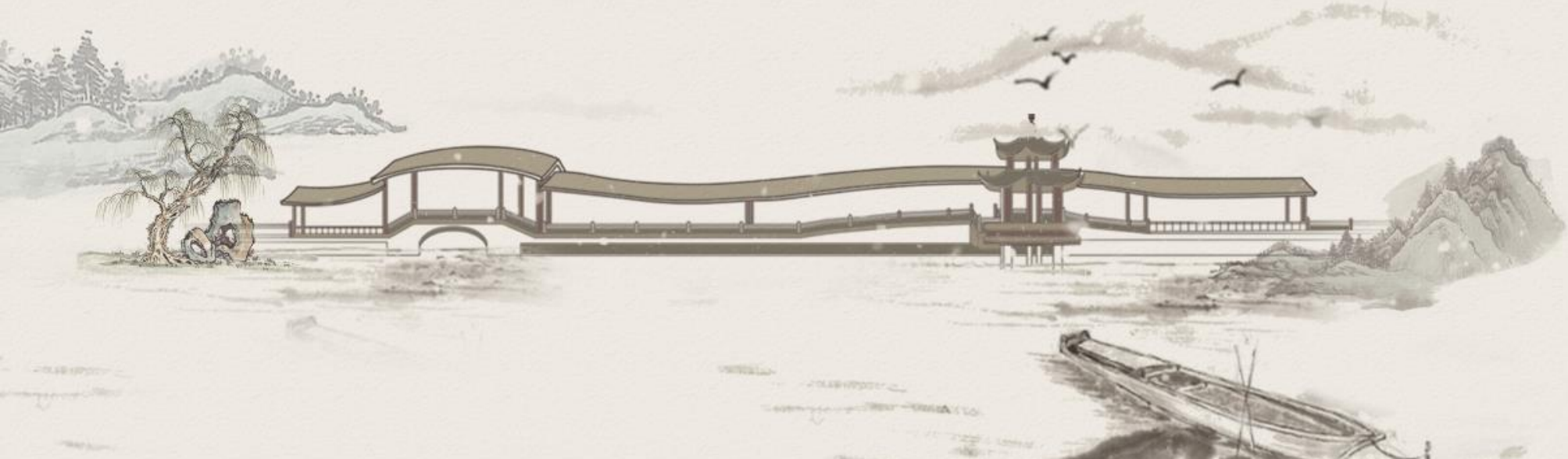
目录

壹 问题引入-约瑟夫问题

贰 分析解决问题(利用数据结构中的线性表进行解决)

叁 数据结构(队列)在解决问题中的作用

肆 队列的简单介绍及应用



01

问题引入-约瑟夫问题

编号为1、2、3…n的n个小朋友按顺时针方向围坐一圈，若从1号小朋友开始按顺时针方向从1顺序报数，报m的小朋友则出列，然后从他在顺时针方向的下一位开始，继续从1开始报数，报m者出列。如此下去，直到所有的小朋友均出列为止。试设计一程序，输出出列小朋友编号的序列。

简单描述

编　号

报　数

指定数字的人列出

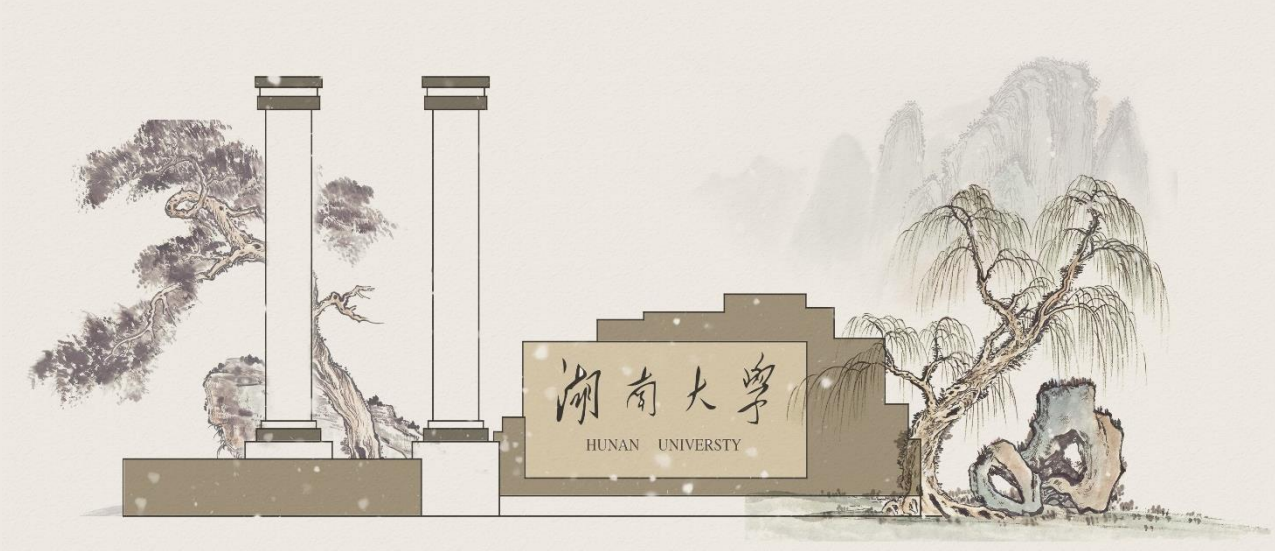
重新报数

出学号同编号输出列的

02

分析问题

考虑时间和空间代价，从算法进行优化



问题分析处理

考虑数据结构是否有某种结构可以解决此问题

每次将报到指定数字的小朋友出列

由于标记和删除的操作，每次都要遍历全部元素，在数据规模较大时，标记或者删除存在一定的问题

最后可以直接按出列编号输出

想到对数组进行标记处理或者删除该元素

综合多方考虑，想到线性表中的队列

队列如何实现

1. 每次报到m的小朋友出列，考虑到在这个结构上我们可以利用队列先进先出的特点，可以这样编写。
2. 每次先判断队列是否为空，若不为空，将队首元素取出，判断是否为m。
3. 如果不为m则将其入队，如果为m则进行输出处理，直到队列为空，即全部小朋友出列。

03 数据结构在起到的作用

- 队列本身先进先出的特点有利于处理这个问题，而在数组处理上需要将每次出列的小朋友的编号进行标记或者是整体进行移动，队列在操作上就体现出方便，快捷的优势。
- 对于规模较大的问题，时间成本更低

代码分析

结构体队列的定义

```
struct Queue0
{
    int elem[MAXQSIZE]; //存储队列元素
    int front;      //队头指针
    int rear;       //队尾指针
} ;
```

约瑟夫问题实现

```
while(Empty_Queue(&sq)!= 1)
{
    Out_Queue(&sq,i);
    if(count==m)
    {
        cout<<i<<" ";
        count = 0;
    }
    else
    {
        In_Queue(&sq,i);
    }
    count++;
}
```

队列操作函数的定义

```
void Init_Queue(Queue0 *sq)
{
    //初始化队列;
    sq->front = sq->rear = 0;
}
int Empty_Queue(Queue0 *sq)
{
    //判断队列空;
    if(sq->front == sq->rear)//队首等于队尾
        return TRUE;
    return FALSE;
}
int In_Queue(Queue0 *sq, int x)
{
    //入队;
    if(((sq->rear + 1)%MAXQSIZE) == sq->front )//队已满，无法入队
        return 0;
    else
    {
        sq->elem[sq->rear]=x;
        sq->rear = (sq->rear + 1) % MAXQSIZE;
        return OK;
    }
}
int  Out_Queue (Queue0 *sq, int &y)
{
    //出队;
    if(Empty_Queue(sq))
        return FALSE;
    y=sq->elem[sq->front];
    sq->front = (sq->front+1)% MAXQSIZE;
    return OK;
}
```

顺序队列的物理实现（代码说明）

```
template <typename E> class AQueue: publicQueue<E> {
private:
  intmaxSize;              // Maximum size of queue 队列最大容量
  intfront;                // Index of front element 头部的索引
  int rear;              //Index of rear element 尾部索引
  E*listArray;             // Array holding queue elements 数组存储队列元素

public:
  AQueue(intsize =100) {  // Constructor 构造函数
// Make list array one position larger for empty slot
//使数组比队列容量多一个位置做为空槽
    maxSize =size+1;
    rear =0;   front = 1;
    listArray =new E[maxSize];
  }
  ~AQueue() {delete [] listArray; } // Destructor 析构函数
```

```
void clear() {rear = 0; front = 1; } // Reinitialize 重新预置
void enqueue(const E& it) {  // Put"it" in queue.   it 入队列
  assert(((rear+2) % maxSize) != front);  // "Queueis full" 队满则断言终止
   rear =(rear+1) % maxSize; // Circularincrement 头部索引在队列范围内循环加 1
   listArray[rear]= it;
  }
E dequeue(){              // Take element out. 一个元素出队列
  assert(length() != 0);          // "Queue is empty" 队空则断言终止
   E it =listArray[front];
   front =(front+1) % maxSize; // Circular increment 尾部索引在队列范围内循环
1 return it;
  }
const E& frontValue() const { // Get front value 获得头部元素
  assert(length() != 0);          //"Queueis empty"队空则断言终止
   returnlistArray[front];
  }
virtual intlength() const      //Return length 返回队列长度
   { return ((rear+maxSize) - front + 1) %maxSize; }
};
```

04 队列的简单介绍及应用

其他应用-广度优先搜索

队列实现广度优先搜索，广度优先搜索，每次扩展的方式向外访问顶点，类似于树的遍历，通过反复取出队首顶点，将该顶点可到达的但未曾加入过队列的顶点全部入队，直到队列为空时遍历结束。

算法如下：（用QUEUE）

(1) 把初始节点S0放入Open表中；

(2) 如果Open表为空，则问题无解，失败

退出；

(3) 把Open表的第一个节点取出放入 Closed表，并记该节点为n；

(4) 考察节点n是否为目标节点。若是，

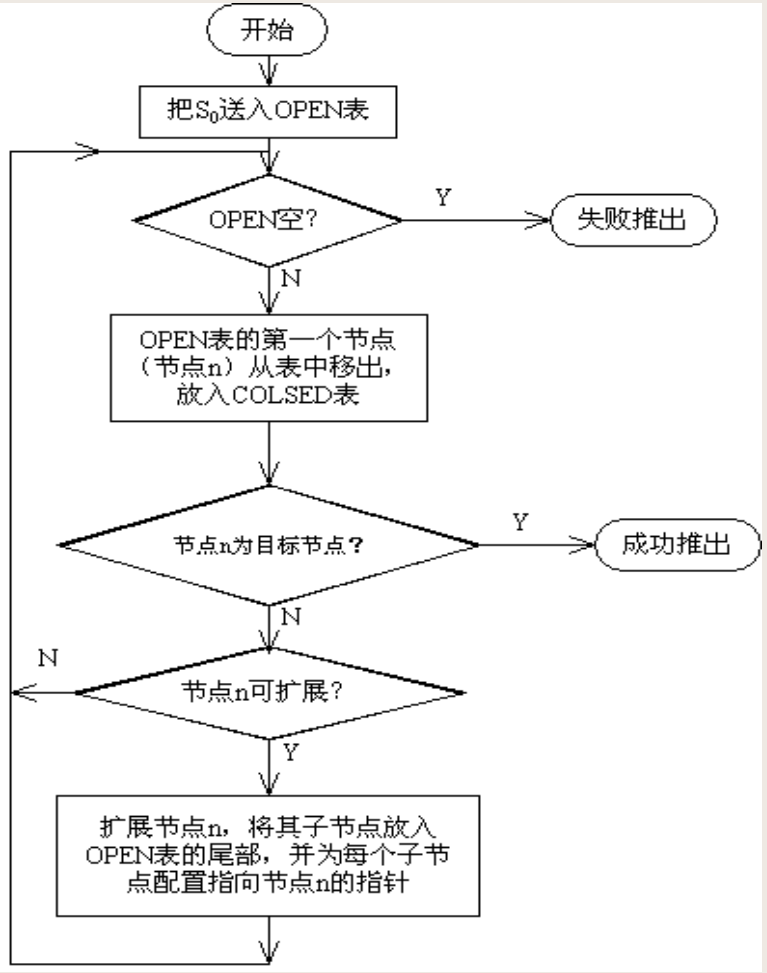
则得到问题的解，成功退出；

(5) 若节点n不可扩展，则转第(2)步；

(6) 扩展节点n，将其不在Closed表和

Open表中的子节点(判重）放入Open表的尾部

，并为每一个子节点设置指向父节点的指针(或记录节点的层次），然后转第(2)步。



总结：

- 数据结构在编写效率更高的程序解决新的设计问题有着很重要的作用，充分考虑时间和空间限制，可以提高程序的效率
- 队列这类数据结构具有先进先出的结构特点，队列元素只能从队尾插入（称为入队操作，enqueue），从队首删除（称为出队操作，dequeue），队列就是按照到达顺序来释放元素的，队列又称为FIFO线性表（First In First Out）.

謝　謝　觀　賞

THANKS FOR WATCHING

参考文献
【1】Clifford A.Shaffer 著 张铭 刘晓丹 等译 。数据结构与算法分析【M】。北京：电子工业出版社，2013：84-85.
【2】胡凡，曾磊 。算法笔记【M】。北京：机械工业出版社，2016：359-367.