桂林电子科技大学

实验 2 栈和队列的基本操作、算法和应用 实验报告

实验名称	栈和队列的基本操作、算法和应用						
院　　系	计算机与信息安全学院		专业	信息安全			
学　　号	2000301708		姓名	蔡响			
实验日期	2021	年	10	月	29	日	

辅导员意见：

成绩　　辅导员签　名

一、实验目的

1. 掌握栈和队列的基本操作
2. 栈和队列的算法设计
3. 栈和队列的应用
4. 递归

二、实验内容

1. 循环队列
2. 进制转换
3. 迷宫-深度策略
4. 迷宫-广度策略
5. 农夫过河-广度策略
6. “聪明的学生”

三、实验环境

在PTA平台进行实验

四、实验要求

根据每个实训的要求完成代码提交和测评

五、实验步骤

描述算法的原理或实现流程（测评完全正确或部分正确的实训）

六、问题记录和实验总结

6-1 循环队列入队出队

读题：本题要求实现队列的顺序存储表示，编写三个入队、出队和取队头的函数

思路：

- 入队：只要队列不满，就入队
- 出队：只要对列不空，就出队
- 去取队头：判断是否为空，为空就打印保存信息，并反回0。否则返回队头。

实现：

主要通过队列中定义队头指针r，队尾指针f，以及Max进行相关的判断实现。

6-2 进制转换（10->16）

读题：本题要求实现十进制到十六进制的转换，用户输入10进制的数，要求输出该数的16进制表示

思路：主要通过栈的方式实现，后入先出，符合正常的运算顺序。先将末尾数转化为16进制，然后向高位进位。

实现：实现采用栈的思想进行入栈与进栈，过程比较简单，不再赘述。对于十进制中10~15对应十六进制的A~F，一般的同学采用的是switch的方式进行。个人认为代码冗余，故采用了ASCALL码的方式进行化简：

```
    if (num < 10 && num >= 0)
        printf("%d", num);
    else
        printf("%c", 55 + num);
```

7-1 迷宫-深度策略

读题：

本题以小老鼠为例子，主要是要求我们写深度搜索迷宫的算法。

思路：

按照题目要求，采用广度搜索的方式进行。

深度优先遍历图的方法是，从图中某顶点v出发：

（1）访问顶点v；

（2）依次从v的未被访问的邻接点出发，对图进行深度优先遍历；直至图中和v有路径 相通的顶点都被访问；

（3）若此时图中尚有顶点未被访问，则从一个未被访问的顶点出发，重新进行深度优先遍历直到图中所有顶点均被访问过为止。

实现：

实现过程与书上略有不同，采用如下的整体架构：

```
Stack Pstack = InitStack();
PassMaze(Pstack, map, step, Begin_End);
PrintStack(Pstack);
```

将走迷宫以及打印迷宫的过程分开，而不是将整个过程通过一个函数实现，增加了代码的可读性，提升了代码的重复利用率，有利于以后的修改。

7-2 迷宫-广度策略

读题：

与上一题相同，要求采用广度的搜索策略

思路：

已知图G=(V,E)和一个源顶点s，宽度优先搜索以一种系统的方式探寻G的边，从而“发现”s所能到达的所有顶点，并计算s到所有这些顶点的距离(最少边数)，该算法同时能生成一棵根为s且包括所有可达顶点的宽度优先树。对从s可达的任意顶点v，宽度优先树中从s到v的路径对应于图G中从s到v的最短路径，即包含最小边数的路径。该算法对有向图和无向图同样适用。
之所以称之为宽度优先算法，是因为算法自始至终一直通过已找到和未找到顶点之间的边界向外扩展，就是说，算法首先搜索和s距离为k的所有顶点，然后再去搜索和S距离为k+l的其他顶点。

实现：

广度策略的实现过程难度较大，主要是要通过添加一个label进行标注，该点的上一个节点位置。因为广度尽可能的搜索路径，所以探测到的路径多，在队列中存储的顺序杂乱，因此需要一个相关的索引进行记录。

农夫过河-广度策略

读题：题目也是源于小时候的问题，要求农夫带着一只狼、一只羊和一棵白菜，到河的北岸。

思路：

用四个二进制位XXXX分别表示农夫、狼、菜和羊四个物品所在的位置。例如1110表示农夫、狼和菜在北岸，菜在南岸。农夫过河问题的初始状态为0000，结束状态为1111。

整个过程采用队列的方式进行存储，采用异或的方式改变状态。

实现：

```
while (!IsNullQueue_seq(moveTo) && (status[15] == -1))
{
    nowstatus = FrontQueue_seq(moveTo);
    DeQueue_seq(moveTo);
    for (movers = 1; movers <= 8; movers <<= 1)

        if ((0 != (nowstatus & 0x08)) == (0 != (nowstatus & movers)))
        //考虑农夫与移动的物品在同一侧
        {
            newstatus = nowstatus ^ (0x08 | movers);
            //如果新状态时安全的且之前没有出现过
            if (IsSafe(newstatus) && (status[newstatus] == -1))
            {
                status[newstatus] = nowstatus;
                EnQueue_seq(moveTo, newstatus);
            }
        }
}

if (status[15] != -1)
{
    for (nowstatus = 15; nowstatus >= 0; nowstatus = status[nowstatus])
    {
        printf("%d ", nowstatus);
        if (nowstatus == 0)
            return;
    }
}
```

聪明的学生

读题：三个学生，互相猜自己头上的数，教授挨个问答，经过6次之后，第三个人得出自己的数

思路：整个思路个人感觉可以采用穷举的方式进行，把自己假设成为其中的某个人，采用穷举的方式进猜测和排除情况。

问题分析：依题可知，每个学生都能知道其他另外两个学生的数字，但不清楚自己数字。假设，我们以 1,2,3，作为例子来分析。A 只有两种情况，一种是（21），另外一种是（2+1），但是 A 自己不能确定是哪一种情况，所以 A 猜不出来。 再来看看 B，两种情况是（1+3）或者（3-1），但是 B 仍然不能确定，最后是 C， 两种情况是（1+2）或者（2-1），但是 C 是可以排除（2-1）这种情况的，因为如 果 C 是（2-1），那么 B 是在看到 A 是 1，C 是 1 的情况下，

B 可以猜出来自己是 2，但是 B 没有猜出来，故而 C 可以排除是（2-1）这种情况，因而 C 只能是（2+1），也就是 3。可以分析总结出最大的数字总会最先猜出来，是因为只有最大的数字 是可以排除相减的情况的，因为如果它是相减得到的话，前面一定有人可以猜出 来。 现在，我们将问题抽象。即 A,B,C 三个学生，他们头上的数字分别为 x1,x2,x3。 从上述结论可知，最大的数总会被最先猜出来。不妨假设，B 是最先猜出来的学 生，即 x2 = x1+x3.而 B 能排除|x1-x3|这种可能性的依据有两个，一是 x1=x3，那 么 B 只能是 x1+x3，因为三个都是正整数。另外一种依据是假设在 x2=|x1-x3|的 前提下，那么在前面的提问中，A 或者 C 已经先猜出来了，而因为他们没有猜出 来，让自己可以确定自己是两数之和，而非两数之差。可以结合上面 1,2,3 去看。 而是 A 先猜出来还是 C 先猜出来，如果 A>C，那么 A 先猜出来，反之，C 会先猜 出来。 那么问题，可以转化为，找出 x1，|x1-x3|,x3 中第一个猜出数字的人所用的 次数。这个问题不断的重复成一个子问题，不断的重复这个过程，直到某个学生 能够猜出这个数字来。

实现：

这样可以归结为如下的一个递归函数

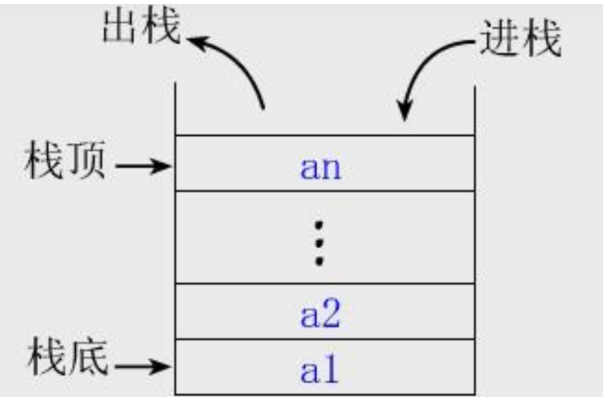
$$Times(I,J,T1,T2,T3)=\begin{cases} T3 & (I=J) \\ Times(I,J,T1,T2,T3) & (I>J) \\ Times(I,J,T1,T2,T3) & (I<J) \end{cases} \tag{1}$$

```
int times(int i,int j,int t1,int t2,int t3)
{
    int k;
    k = i - j;
    if (k == 0)
        return t3;
    else if (k > 0)
        return times(j, i - j, t2, t3, t1) + step(t1, t3);
    else
        return times(i, j - i, t1, t3, t2) + step(t2, t3);
}
```

总结

栈

只允许在一端进行插入或删除操作的线性表。首先，栈是一种线性表，但限定这种线性表只能在某一段进行插入和删除操作。



栈顶（Top）：线性表允许进行插入和删除的一端。

栈底（Bottom）：固定的，不允许进行插入和删除的另一端。

空栈：不含任何元素。

如上图：a1为栈底元素，an为栈顶元素。由于栈只能在栈顶进行插入和删除操作，故进栈次序依次为a1，a2，... ,an而出栈次序为an，...，a2，a1。栈的明显的操作特征为后进先出（Last In First Out，LIFO）,故又称 后进先出的线性表。

栈的基本操作

1）InitStack（&S）：初始化空栈S

2）StackEmpty（S）：判断一个栈是否为空

3）Push（&S，x）：进栈，若栈未满，则将x加入使之成为新栈顶

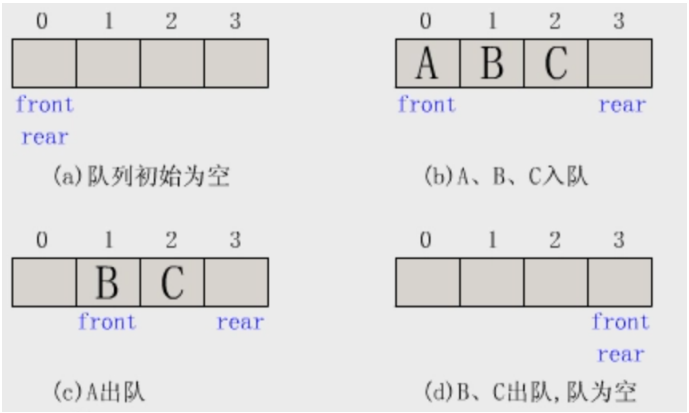
4）Pop（&S，&x）：出栈，若栈非空，则将栈顶元素，并用x返回

5）GetTop(S，&x)：读栈顶元素，若栈顶元素非空，则用x返回栈顶元素

6）DestroyStack(&S)：销毁栈，并释放栈S占用的存储空间

队列

队列是一种特殊的线性表，特殊之处在于它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作，和栈一样，队列是一种操作受限制的线性表。进行插入操作的端称为队尾，进行删除操作的端称为队头。



克服假溢出的方法有两种。一种是将队列中的所有元素均向低地址区移动，显然这种方法是很浪费时间的；另一种方法是将数组存储区看成是一个首尾相接的环形区域。当存放到n地址后，下一个地址就"翻转"为1。在结构上采用这种技巧来存储的队列称为循环队列。
队列和栈一样只允许在断点处插入和删除元素。
循环队的入队算法如下：
1、tail=tail+1；
2、若tail=n+1，则tail=1；
3、若head=tail，即尾指针与头指针重合了，表示元素已装满队列，则作上溢出错处理；
4、否则，Q(tail)=X，结束（X为新入出元素）。
队列和栈一样，有着非常广泛的应用。
注意：（1）有时候队列中还会设置表头结点，就是在队头的前面还有一个结点，这个结点的数据域为空，但是指针域

指向队头元素。

（2）另外，上面的计算还可以利用下面给出的公式cq.rear=(cq.front+1)/max;
当有表头结点时，公式变为cq.rear=(cq.front+1)/（max+1）。

广度搜索

BFS，其英文全称是Breadth First Search。 BFS并不使用经验法则算法。从算法的观点，所有因为展开节点而得到的子节点都会被加进一个先进先出的队列中。一般的实验里，其邻居节点尚未被检验过的节点会被放置在一个被称为 open 的容器中（例如队列或是链表），而被检验过的节点则被放置在被称为 closed 的容器中。

深度搜索

DFS简介——DFS即深度优先搜索算法，属于图的遍历算法中的一种，英文缩写为DFS即Depth First Search.其搜索过程简要来说是对每一个可能的分支路径深入到不能再深入为止，而且每个节点都只会搜索一次。

它的具体思想就如同在家里找钥匙，无论从哪一间房间开始都可以，比如主卧室，然后从房间的一个角开始，将房间内的墙角、床头柜、床上、床下、衣柜里衣柜上、前面的电视柜等挨个寻找，做到不放过任何一个死角，所有的抽屉、储藏柜中全部都找遍，形象比喻就是翻个底朝天，然后再寻找下一间，直到找到为止。对于DFS的理解是：不撞南墙不回头（如果撞了，那么可能回头，可能不回头；如果回头，那么回头的过程叫做回溯）。