班级：21级自动化 6班 姓名： 吴俊达 学号： 210320621

**第三章 栈和队列**

1、设将整数1、2、3、4依次进栈，但只要出栈时栈非空，则可将出栈操作按任何次序夹入其中，请回答下列问题：

（1）若入栈次序为push(1)，pop()，push(2)，push(3)，pop()，pop()，push(4)，pop()，则出栈的数字序列为什么？

（2）能否得到出栈序列423和432？并说明为什么不能得到或如何得到。

（3）请分析1、2、3、4的24种排列中，哪些序列可以通过相应的入出栈得到。

答：(1) 出栈的数字序列为：1,3,2,4.

(2) 可以得到出栈序列432，方法是：push(1)，push(2)，push(3)，push(4)，pop()，pop()，pop()。

不可以得到出栈序列423，因为栈是后进先出的数据结构。第一个输出的是4，说明1、2、3、4已经全部入栈，然后4出栈，此时3在栈顶，一定比2先出栈。

(3) 可以得到的是：

1234【方法是：push(1)，pop()，push(2)，pop()，push(3)，pop()，push(4)，pop()】

1243【方法是：push(1)，pop()，push(2)，pop()，push(3)，push(4)，pop()，pop()】

1324【方法是：push(1)，pop()，push(2)，push(3)，pop()，pop()，push(4)，pop()】

1342【方法是：push(1)，pop()，push(2)，push(3)，pop()，push(4)，pop()，pop()】

1432【方法是：push(1)，pop()，push(2)，push(3)，push(4)，pop()，pop()，pop()】

2134【方法是：push(1)，push(2)，pop()，pop()，push(3)，pop()，push(4)，pop()】

2143【方法是：push(1)，push(2)，pop()，pop()，push(3)，push(4)，pop()，pop()】

2314【方法是：push(1)，push(2)，pop()，push(3)，pop()，pop()，push(4)，pop()】

2341【方法是：push(1)，push(2)，pop()，push(3)，pop()，push(4)，pop()，pop()】

2431【方法是：push(1)，push(2)，pop()，push(3)，push(4)，pop()，pop()，pop()】

3241【方法是：push(1)，push(2)，push(3)，pop()，pop()，push(4)，pop()，pop()】

3214【方法是：push(1)，push(2)，push(3)，pop()，pop()，pop()，push(4)，pop()】

3421【方法是：push(1)，push(2)，push(3)，pop()，push(4)，pop()，pop()，pop()】

4321【方法是：push(1)，push(2)，push(3)，push(4)，pop()，pop()，pop()，pop()】

共14种。

2、链栈中为何不设头指针？

答： 因为链栈是运算受限的单链表，其插入和删除操作仅限制在表头位置上进行，如果加了头结点，要对头结点之后的结点进行操作，反而使算法更复杂。

3、 当利用大小为N的数组顺序存储一个栈时,假定用top==N表示栈空,则向这个栈插入一个元素时,首先应执行( 2 )语句修改top指针.

* 1. top++
  2. top--
  3. top=0
  4. top=N-1

4、假定利用数组a[N]顺序存储一个栈，top表示栈顶指针，top==-1表示栈空，并已知栈未满，当元素X进栈时所执行的操作为( 3 )。

* 1. a[--top]=x
  2. a[top--]=x
  3. a[++top]=x
  4. a[top++]=x

5、判定一个栈s（最多元素数为m0）为空的条件是( 2 )。为满的条件是( 4 )。

* 1. s->top!=0
  2. s->top==0
  3. s->top!=m0
  4. s->top==m0
  5. s->top!=m0-1
  6. s->top==m0-1

6、假定一个链式栈的栈顶指针用top表示，每个结点包含data和next两个域，入栈时所进行的指针操作为( 3 )。

* 1. top->next=top
  2. top=top->data
  3. top=top->next
  4. top->next=top->next->next

4、指出下列程序段的功能是什么？

(1)

void demo1(seqstack \*s){

int I;

arr[64];n=0;

while (!stackempty(s)) arr[n++]=pop(s);

for(I=0;I<n;I++) push(s,arr[I]);

}

程序段的功能是： 将栈倒序排列

(2)

void demo2(seqstack \*s,int m){

seqstack t;

int i;

initstack(t);

while(! Stackempty(s))

if((i=pop(s))!=m) push(t,i);

while(! Stackempty(t)) {

i=pop(t);

push(s,i);

}

}

程序段的功能是： 删除栈s中值为m的元素

7、有字符串次序为-3\*-y-a/y^2,试利用栈排出将次序改变为3y-\*ay^2/--的操作步骤。

（假设用X代表扫描该字符串顺序取一字符进栈的操作，用S代表从栈中取出一个字符到新字符串尾的出栈操作。如abc变为bca，则操作步骤为xxsxss）。

解：操作步骤为XXSXXXSSSXXSXXXXSSSSSS。

8、循环队列的优点是什么？如何判断它的空和满？

答：循环队列的优点：

1）相对于链式队列，使用数组存储，可以很方便地通过下标来访问，操作更为简便；

2）顺序队列中，当队尾指针指向队列存储结构中的最后单元时，如果再继续插入新的元素，则会产生溢出，且当队列发生溢出时，队列存储结构中可能还存在一些空白位置（已被取走数据的元素），即假上溢现象；循环队列通过头尾指针的移动，使得队列的最后一个元素不受限于数组最大上标所指向的位置，而是可以循环存储，提高了空间利用效率，减少了空元素（例如，循环队列发生溢出时，只有一个元素空间未被利用）。

判断循环队列空和满的方法可以有以下三种：

其一是另设一个布尔变量以匹别队列的空和满。

**其二（最常用的方法），是少用一个元素的空间，约定入队前，测试尾指针在循环意义下加1后是否等于头指针（即测试(rear+1)%MaxSize是否等于front），若相等则认为队满（rear始终指向最后一个元素的下一个单元，始终为空）；判断队空的方法是：rear==front时，队为空。**

其三是使用一个计数器记录队列中元素的总数（实际上是队列长度）。

9、设长度为n的链队列用单循环链表表示，若只设头指针，则怎样进行入队和出队操作；若只设尾指针呢？

答：**以下假设这个单循环链表没有头结点。（即头指针对应的结点就存储数据，而不是头指针指向的下一个结点才存储数据）**

**只设头指针：**

入队：通过头指针向后找n个结点到达尾结点，在尾结点后插入新结点。假设尾结点指针为rear，则插入的代码是（设s指向新结点）s->next = front; rear->next = s。

出队：p=front; front = front->next; free(p)

**只设尾指针：**

入队：设s指向新结点。则代码为s->next = rear->next; rear->next = s.（直接在尾结点后插入新结点）

出队：先存储rear->next->next的指针（假设存储在指针p中）；然后通过rear->next找到头结点，释放掉头结点的空间后，再将rear->next指向p即可。

10、 假设Q[11](下标为从0到10)是一个循环队列,初始状态为front=rear=0;画出分别做完下列操作后队列的头尾指针的装填变化情况,若不能入队,请指出其元素,说明理由. (采用少用一个元素空间的方式)

* + d,e,b,g,h入队
  + d,e出队
  + i,j,k,l,m入队
  + b出队
  + n,o,p,q,r 入队

答：如下图所示

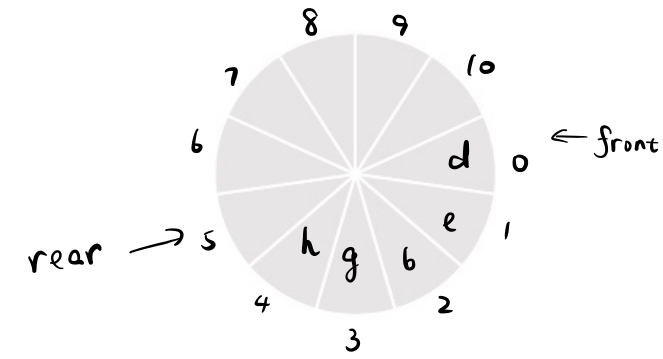
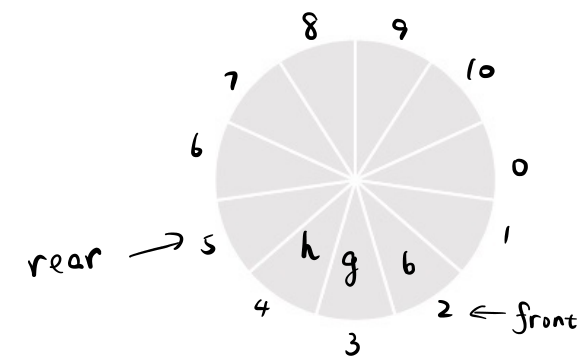
 

图1 图2

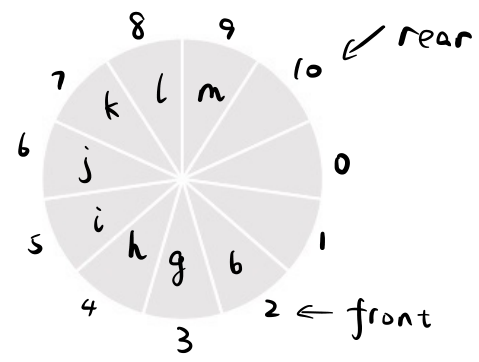
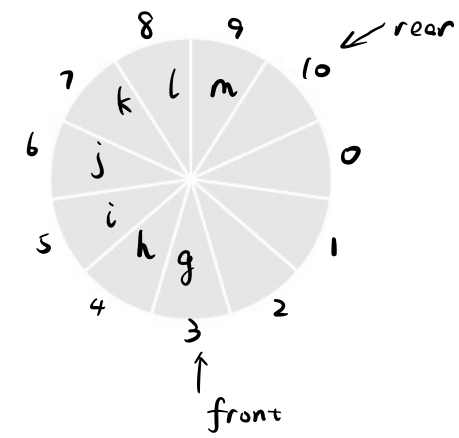
 

图3 图4

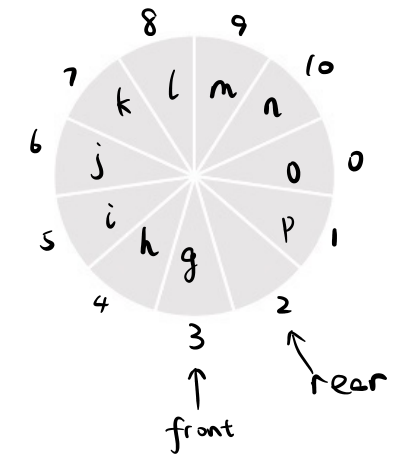


图5

可见，q,r无法入队，因为p入队后，(rear+1)%MaxSize == front，队已满。

11、一个队列的入队序列是1，2，3，4，则队列的输出序列是( 2 )

* 1. 4，3，2，1
  2. 1，2，3，4
  3. 1，4，3，2
  4. 3，2，4，1

12、假定一个顺序循环队列的队首和队尾指针分别用front和rear表示，则判断队空的条件为( 4 )。

* 1. front +1==rear
  2. rear+1==front
  3. front= =0
  4. front= =rear

13、假定一个顺序循环队列存储于数组a[N]中，其队首和队尾指针分别用front和rear表示，则判断队满的条件为( 2 )。

* 1. （rear-1）%N= =front
  2. （rear+1）%N==front
  3. （front-1）%N== rear
  4. （front+1）%N==rear

14、循环队列用数组a[m]存放其元素值，已知其头尾指针分别用front和rear表示，则队列中的元素个数为( 1 )。

* 1. （rear-front+m）%m
  2. rear-front+1
  3. rear-front-1
  4. rear-front

15、假定一个链队列的队首和队尾指针分别用front和rear表示，每个结点包含data和next两个域，出队时所进行的指针操作为( 2 )。

* 1. front=front->date
  2. front=front->next
  3. rear=rear->next
  4. rear=rear->date

16、设栈S和队列Q的初始状态为空，元素e1，e2，e3，e4，e5，e6依次通过栈S，一个元素出栈后即进入队列Q，若出队的顺序为e2，e4，e3，e6，e5，e1，则栈S的容量至少应该为( B )。

A. 2 B. 3

C. 4 D. 5

17、假定用一维数组a[7]顺序存储一个循环队列，队首和队尾指针分别用front和rear表示，当前队列中已有4个元素：12，23，78，60，其中12为队首元素，front的值为3，请画出对应的存储状态。当连续做2次出队运算后，再让15，36，40，50，60元素依次进队，分别画出队、入队后对应的存储状态。

初始状态：

rear front

0 1 2 3 4 5 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 12 | 23 | 78 | 60 |

出队两次后：

rear front

0 1 2 3 4 5 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 78 | 60 |

5个元素入队后：

rear front

0 1 2 3 4 5 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 15 | 36 | 40 | 50 |  | 78 | 60 |

**60 无法进队。**

**第五章 数组和广义表**

1. 假设一个10×10的上三角矩阵A按照列优先顺序压缩存储在一维数组 B中，则B数组的大小应为( B )。

A. 50 B. 55 C. 100 D. 101

1. A是7×4的二维数组，按行优先方式顺序存储，元索A[0][0]的存储地址为1000，若每个元素占2个字节，则A[3][3]的存储地址为( C )。

A. 1026 B. 1028 C. 1030 D. 1032

1. 对稀疏矩阵采用三元组表示法的目的是( D )。

A.便于输入和输出 B.便于进行矩阵运算

C.降低时间复杂度 D.节省存储空间

1. 已知一个6×6的稀疏矩阵如下所示，试写出该矩阵的三元组表示。



**三元组表示为：【下标从0开始】**

1. 一个数组的第一个元素的存储地址是100，每个元素占2存储单元，则第5个元素的存储地址是( B )。

A．105 B.108 C．115 D．118

1. 假设一个8阶的上三角矩阵A按照列优先顺序压缩存储在一维数组B中，则B数组的大小应为\_\_\_36\_\_\_\_\_\_\_\_。
2. 二维数组A[8][9]按行优先顺序存储，若数组元素A[2][3]的存储地址为1087，A[4][7]的存储地址为1153，则每个数组元素占用的存储单元的个数是\_\_\_\_\_3\_\_\_\_\_\_。
3. 二维数组A按行序优先顺序存储，每个数据元素占1个存储单元。若数据元素 A[1][1]的存储地址是420，A[3][3]的存储地址是446，则A[5][5]的存储地址是( C )。

A．470 B. 471 C．472 D．473

1. 稀疏矩阵可以采用\_\_\_\_转化为三元组\_\_\_\_\_\_方法进行压缩存储。
2. 把特殊矩阵A[10][10]的下三角矩阵压缩存储到一个一维数组M中，则A中元素a[4][3]在M中所对应的下标位置是( C )。

A．8 B．12 C．13 D．55

1. 设有二维数组A[8][10]，按行序优先存储，且每个元素占用2个存储单元，若第一个元素的存储起始位置为b，则存储位置为b+20处的元素为\_\_\_A[1][0]\_\_\_\_\_\_\_\_。