第三章 栈，队列和数组

2022年9月30日 星期五 12:52

栈

逻辑结构知识点

栈	• 只允许在一端进行插入或删除操作的线性表
栈顶	• 线性表允许进行插入删除的那一段
栈底	• 固定的，不允许进行插入删除的那一段
操作特性	• 后进先出
数学性质	• n个不同元素进栈，出栈元素的不同排列的个数为 $\frac{1}{n+1}C_{2n}^{n}$（卡特兰数）

存储结构知识点

1、栈的顺序存储结构

顺序栈的定义

- 顺序栈：采用顺序存储的栈称为顺序栈
- 利用一组地址连续的存储单元存放自栈底到栈顶的数据元素

顺序栈的实现

```
#define Maxsize 50
typedef struct
{
    Elemtype data[Maxsize]; //存放栈中元素
    int top;                //栈顶指针
} SqStack;
```

栈顶指针	S.top，初始时设置S.top=-1；栈顶元素：S.data[S.top]		
进栈操作	栈不满时，栈顶指针先加1，再送值到栈顶元素	出栈操作	栈非空时，先栈栈顶元素值，再将栈顶指针减1
栈空条件	S.top==-1	栈满条件	S.top==Maxsize-1；栈长：S.top+1

顺序栈基本运算代码

初始化

```
void InitStack(SqStack &S)
{
    S.top = -1;
}
```

判栈空

```
bool StackEmpty(SqStack S)
{
    if (S.top == -1)
        return true;
    else
        return false;
}
```

进栈

```
bool Push(SqStack &S, ElemType x)
{
    if (S.top == Maxsize - 1)
        return false;
    S.data[++S.top] = x return ture;
}
```

出栈

```
bool Pop(SqStack &S, ElemType x)
{
    if (S.top == -1)
        return false;
    S.data[S.top--] = x return ture;
}
```

读栈顶元素

```
bool GetTop(SqStack S, ElemType x)
{
    if (S.top == -1)
        return false;
    x = S.data[S.top] return ture;
}
```

共享栈

定义	• 利用栈底位置相对不变的特性，可让两个顺序栈共享一个一位数组看空间 • 将两个栈的栈底分别设置在共享空间的两端，两个栈顶向共享空间的中间延伸
特点	• top0=-1时0号栈为空，top1=MaxSize时1号栈为空 • 只有当top1-top0=1时，判断为栈满 • 当0号栈进栈时top0先加1再赋值，1号栈进栈时top1先减1再赋值；出栈时恰好相反
目的	• 更有效地利用存储空间，两个栈的空间相互调节，只有在整个存储空间都被占满时才发生上溢

2、栈的链式存储结构

概念	• 采用链式存储的栈
优点	• 便于多个栈共享存储空间和提高其效率，且不存在栈慢上溢的情况
特点	• 通常采用单链表实现，并规定所有操作都是在单链表的表头进行的 • 这里规定链栈没有头结点，Lhead指向栈顶元素

链式栈的实现

```
typedef struct Linknode
{
    Elemtype data;          //存放栈中元素
    struct Linknode *next; //栈顶指针
} * LiStack;
```

运算/操作

InitStack(&S)	初始化栈	• 初始化一个空栈S
StackEmpty(S)	判断栈是否为空	• 空则返回True
Push(&S,x)	进栈	• 若S未满，则将x加入使之成为新栈顶
Pop(&S,&x)	出栈	• 若S非空，则弹出栈顶元素，并用x返回
GetTop(S,&x)	读栈顶元素	• 若栈非空，则用x返回栈顶元素
DestroyStack(&S)	销毁栈	• 销毁并释放栈S占用的存储空间

应用

括号匹配	• 初始设置一个空栈，顺序读入括号 • 若是右括号，则或者使置于栈顶的最急迫期待得以消解，或者是不合法的情况（括号序列不匹配，退出程序） • 若是左括号，则作为一个新的更急迫的期待压入栈中，自然使原有的在栈中的所有未消解的期待的急迫性下降了一级 • 算法结束时，栈为空，否则括号序列不匹配
表达式求值	通过后缀表示计算表达式值的过程： • 顺序扫描表达式的每一项，然后根据它的类型做出相应操作 • 若该项是操作数，则将其压入栈中 • 若该项是操作符<op>，则连续从栈中退出两个操作数Y和X，形成运算指令X<op>Y，并将计算结果重新压入栈中 • 当表达式的所有项都扫描并处理完毕后，栈顶存放的就是最后的计算结果。
递归	• 在递归调用的过程中，系统为每一层的返回点、局部变量、传入实参等开辟了递归工作栈来进行数据存储 • 递归次数过多容易造成栈溢出等。 • 其效率不高的原因是递归调用过程中包含很多重复的计算 • 但代码简单，容易理解 • 可以将递归算法转换为非递归算法，通常需要借助栈来实现这种转换
进制转换	
迷宫求解	

队列

逻辑结构知识点

队列	• 一种操作首先的线性表，只允许在表的一端进行插入，而在表头的另一端进行删除。
入队或进队	• 向队列中插入元素
出队或离队	• 删除元素
队头	• 允许删除的一端
队尾	• 允许插入的一端
空队列	• 不含任何元素的空表
操作特性	• 先进先出

存储结构知识点

1、队列的顺序存储结构

顺序队列的定义

- 分配一块连续的存储单元存放队列中的元素
- 并附设两个指针
  - 队头指针front指向队头元素
  - 队尾指针rear指向队尾元素的下一个位置。

顺序队列的实现

```
#define MaxSize 50
typedef struct
{
    ElemType data[MaxSize];
    int front, rear;
} SqQueue;
```

初始状态	Q.front==Q.rear==0	队满操作	• Q.rear==MaxSize不能作为队列满的条件 • 只有一个元素仍满足该条件（假溢出）
进队操作	队不满时，先送值到队尾元素，再将队尾指针加1	出队操作	队不空时，先取队头元素值，再将队头指针加1

循环队列的定义

- 把存储队列元素的表从逻辑上视为一个环

循环队列的实现

- 当队首指针Q.front=MaxSize-1后，再前进一个位置就自动到0
- 初始时：Q.front=Q.rear=0
- 队首指针进1：Q.front=（Q.front+1）%Maxsize
- 队尾指针进1：Q.rear=（Q.rear+1）%MaxSize
- 出队入队时：指针都按顺时针方向进1
- 队空和队满的条件：Q.front=Q.rear

如何区分循环队列队空还是对满

	方法一	方法二	方法三
定义	牺牲一个单元来区分队空和队满 入队时少用一个队列单元 约定以"队头指针在队尾指针的下一位置作为队满的标志	类型中增设表示元素个数的数据成员	类型中增设tag数据成员，以区分是队满还是队空
队空条件	Q.front==Q.rear	Q.size==0	tag=0且因删除导致Q.front==Q.rear
队满条件	Q.rear+1）%MaxSize==Q.front	Q.size==MaxSize	tag=0且因插入导致Q.front==Q.rear
元素个数	（Q.rear-Q.front+MaxSize）%MaxSize		

循环队列的操作代码

初始化

```
void InitQueue(SqQueue &Q)
{
    Q.rear = Q.front = 0;
}
```

判队空

```
bool isEmpty(SqQueue Q)
{
    if (Q.rear == Q.front)
        return true;
    else
        return false;
}
```

入队

```
bool EnEmpty(SqQueue &Q, ElemType x)
{
    if ((Q.rear + 1) % MaxSize == Q.front)
        return false;
    Q.data[Q.rear] = x;
    Q.rear = (Q.rear + 1) % MaxSize;
    return true;
}
```

出队

```
bool DeEmpty(SqQueue &Q, ElemType &x)
{
    if (Q.rear == Q.front)
        return false;
    x = Q.data[Q.front];
    Q.front = (Q.front + 1) % MaxSize;
    return true;
}
```

2、队列的链式存储结构

链式队列的定义

- 实质上是一个同时带有队头指针和队尾指针的单链表
- 头指针指向队头节点，尾指针指向队尾节点，即单链表的最后一个节点

链式队列的实现

```
typedef struct LinkNode
{
    ElemType data;
    struct LinkNode *next;
} LinkNode;
typedef struct
{
    LinkNode *front, *rear;
} LinkNode;
```

链式队列的操作代码

初始化

```
void InitQueue(LinkQueue &Q)
{
    Q.front = Q.rear = (LinkNode *)
malloc(sizeof(LinkNode));
    Q.front->next = NULL;
}
```

判队空

```
bool isEmpty(LinkQueue Q)
{
    if (Q.rear == Q.front)
        return true;
    else
        return false;
}
```

入队

```
bool EnEmpty(LinkQueue &Q, ElemType x)
{
    LinkNode *s = (LinkNode *)malloc(sizeof(LinkNode));
    s->data = x;
    s->next = NULL;
    Q.rear->next = s;
    Q.rear = s;
}
```

出队

```
bool DeEmpty(LinkQueue &Q, ElemType &x)
{
    if (Q.rear == Q.front)
        return false;
    LinkNode *p = Q.front->next;
    x = p->data;
    Q.front->next = p->next
    if (Q.rear == p)
        Q.rear = Q.front;
    free(p);
    return true;
}
```

3、双端队列

定义	• 允许两端都可以进行入队和出队操作的队列。 • 将队列的两端分别称为前端和后端
特点	• 其元素的逻辑结构仍是线性结构
分类	输出受限的双端队列：• 允许在一段进行插入和删除，但在另一端只允许插入的双端队列 输入受限的双端队列：• 允许在一段进行插入和删除，但在另一端只允许删除的双端队列

运算/操作

InitQueue(&Q)	初始化队列	• 构造一个空队列Q
QueueEmpty(Q)	判队列为空	• （空则返回True）
EnQueue(&Q,x)	入队	• 若队列未满，将x加入，使之成为新的队尾
DeQueue(&Q,&x)	出队	• 若队列非空，删除表头元素，并用x返回
GetHead(Q,&x)	读队头元素	• 若队列非空，则将队头元素赋值给x

应用

层次遍历	• 需要逐层或逐行处理的问题，解决方法如下👇
	  ◦ 往往是在处理完当前层或当前行时就对下一层或下一行做预处理
	  ◦ 把处理顺序安排好，等到当前层或当前行处理完毕，就可以处理下一层或下一行
	• 使用队列是为了保存下一步的处理顺序
	• 层次遍历二叉树的处理过程👇
	  ◦ 1.根节点入队
	  ◦ 2.若队空（所有节点都已处理完毕），则结束遍历；否则重复3操作
	  ◦ 3.队列中第一个节点入队，并访问之。若其有左孩子，则将左孩子入队；若其有右孩子，则将右孩子入队，返回2

在计算机系统中的应用	1. 解决主机与外部设备之间速度不匹配的问题（如打印机与主机，设置一个打印数据缓冲区） 2. 解决由多用户引起的资源竞争问题（如CPU资源的竞争）

数组和特殊矩阵

本部分重点	• 如何将矩阵更有效地存储在内存中，并能方便地提取矩阵中的元素
数组的定义	• 由n个相同类型的数据元素构成的有限序列
数组的特点	• 数组一旦被定义，其维数和维界就不再改变 • 除结构的初始化和销毁外，数组只会有存取元素和修改元素的操作
数组的数据结构	一个数组的所有元素在内存中占用一段连续的存储空间 对于多维数组，有两种映射方法 按行优先：先行后列，先存储行号较小的元素，行号相等先存储列号较小的元素 按列优先：先列后行，先存储列号较小的元素，列号相等先存储行号较小的元素
特殊矩阵相关概念	压缩存储：为多个值相同的元素只分配一个空间，对零元素不分配存储空间，其目的是节省存储空间 特殊矩阵：具有许多相同矩阵元素或者零元素，并且这些相同矩阵元素或零元素的分布有一定规律性的矩阵 常见的特殊矩阵有对称矩阵、上（下）三角矩阵、对角矩阵等
特殊矩阵的压缩存储方法	• 找到特殊矩阵中值相同的矩阵元素的分布规律 • 把这些呈现规律性分布的、值相同的多个矩阵元素压缩存储到一个存储空间中
稀疏矩阵的定义	• 矩阵中非零元素的个数t，相对矩阵元素的个数s非常少的矩阵 • 将非零元素及其相应的行和列构成一个三元组（行标，列标，值），然后按照某种规律存储这些三元组
叙述矩阵的特点	• 稀疏矩阵压缩存储后便失去了随机存取特性。 • 三元组既可以使用数组存储，也可以采用十字链表法存储