

3.1 栈

栈的基本概念

只允许在一端进行插入或删除操作的线性表 先进后出

结构 栈顶 (Top) : 线性表允许进行插入删除的那一端

栈底 (Bottom) : 固定的, 不允许进行插入和删除的另一端

卡特兰数 n 个不同元素进栈, 出栈元素不同排列的个数为 $\frac{1}{n+1} C_{2n}^n$

栈的基本操作 InitStack (&S): 初始化一个空栈S

StackEmpty (S): 判断一个栈是否为空, 若栈S为空则返回true, 否则返回false Push (&S, x): 进栈, 若栈S未满, 则将x加入使之成为新栈顶

Pop (&S, &x): 出栈, 若栈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S非空, 则用x返回栈顶元素

DestroyStack (&S): 销毁栈, 并释放栈S占用的存储空间 ("&"表示引用调用)

在解答算法题时, 若题干未做出限制, 则可直接使用这些基本的操作函数

栈的顺序存储结构

顺序栈的实现

采用顺序存储的栈称为顺序栈, 它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, 同时附设一个指针(top)指示当前栈顶元素的位置

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, 先取栈顶元素值, 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

```
void InitStack(SqStack &S) {
    S.top = -1; // 初始化栈顶指针
}
```

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

```
bool Push(SqStack &S, ElemType x) {
    if (S.top == MaxSize - 1) // 栈满, 报错
        return false;
    S.data[++S.top] = x; // 指针先加1, 再入栈
    return true;
}
```

```
bool Pop(SqStack &S, ElemType &x) {
    if (S.top == -1) // 栈空, 报错
        return false;
    x = S.data[S.top--]; // 先出栈, 指针再减1
    return true;
}
```

```
bool GetTop(SqStack S, ElemType &x) {
    if (S.top == -1) // 栈空, 报错
        return false;
    x = S.data[S.top]; // * 记录栈顶元素
    return true;
}
```

顺序栈的基本操作

让两个顺序栈共享一个一维数组空间, 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

共享栈 基本原则 两个栈的栈顶指针都指向栈顶元素, top0 = -1 时 0号栈为空, top1 = MaxSize 时 1号栈为空

当两个栈顶指针相邻 (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; 出栈时则刚好相反

存取数据的时间复杂度均为 O(1)

栈的链式存储结构

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

优点 便于多个栈共享存储空间和提高其效率

且不存在栈满上溢的情况

3.2队列（上）

队列的基本概念

- 队列的定义
 - 是一种操作受限的线性表，只允许在表的一端进行插入，而在表的另一端进行删除
 - 向队列中插入元素称为入队或进队；删除元素称为出队或离队
 - 先进先出
- 结构
 - 队头(Front)：允许删除的一端，又称队首
 - 队尾(Rear)：允许插入的一端
 - 空队列：不含任何元素的空表
- 队列基本操作
 - InitQueue (&Q):初始化队列，构造一个空队列Q
 - QueueEmpty (Q):判队列空，若队列Q为空返回true,否则返回false
 - EnQueue (&Q, x):入队，若队列Q未满，将x加入，使之成为新的队尾
 - DeQueue (&Q, &x):出队，若队列Q非空，删除队头元素，并用x返回
 - GetHead(Q, &x):读队头元素，若队列Q非空，则将队头元素赋值给x

队列的顺序存储结构

- 队列的顺序存储
 - 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（具体问题具体分析）
 - 初始状态（队空条件）： $Q.front == Q.rear == 0$
 - 基本操作
 - 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
 - 假溢出
 - 这种溢出并不是真正的溢出,在data数组中依然存在可以存放元素的空位置
- 循环队列
 - 把存储队列元素的表从逻辑上视为一个环，称为循环队列
 - 基本操作
 - 初始时: $Q.front = Q.rear = 0$
 - 队首指针进1: $Q.front = (Q.front + 1) \% \text{MaxSize}$
 - 队尾指针进1: $Q.rear = (Q.rear + 1) \% \text{MaxSize}$
 - 队列长度: $(Q.rear + \text{MaxSize} - Q.front) \% \text{MaxSize}$
 - 队空： $Q.front == Q.rear$
 - 判断条件
 - 牺牲一个单元来区分队空和队满,即"队头指针在队尾指针的下一位置作为队满的标志"
 - 队满条件: $(Q.rear + 1) \% \text{MaxSize} == Q.front$
 - 队空条件: $Q.front == Q.rear$
 - 队列中元素的个数: $(Q.rear - Q.front + \text{MaxSize}) \% \text{MaxSize}$
 - 队满
 - 类型中增设表示元素个数的数据成员。
 - 队空：为 $Q.size == 0$
 - 队满： $Q.size == \text{MaxSize}$
 - 类型中增设 tag 数据成员,以区分是队满还是队空
 - tag 等于0时,若因删除导致 $Q.front == Q.rear$,则为队空
 - tag 等于1时,若因插入导致 $Q.front == Q.rear$,则为队满

3.2 队列（下）

队列的链式存储结构

队列的链式存储

队列的链式表示称为链队列，是一个同时带有队头指针和队尾指针的单链表。

头指针指向队头结点

尾指针指向队尾结点

当 $Q.front == NULL$ 且 $Q.rear == NULL$ 时，链式队列为空

优点

适合于数据元素变动比较大的情形

不存在队列满且产生溢出的问题

双端队列

概述

双端队列是指允许两端都可以进行入队和出队操作的队列

逻辑结构仍是线性结构

将队列的两端分别称为前端和后端，两端都可以入队和出队

分类

输出受限的双端队列：允许在一端进行插入和删除，但在另一端只允许插入的双端队列

输入受限的双端队列：允许在一端进行插入和删除，但在另一端只允许删除的双端队列

3.3~4矩阵的压缩存储（上）

栈的应用

括号匹配

中缀表达式转后缀表达式（手算）

- ① 确定中缀表达式中各个运算符的运算顺序
- ② 选择下一个运算符，按照「左操作数 右操作数 运算符」的方式组合成一个新的操作数
- ③ 如果还有运算符没被处理，就继续 ②

后缀表达式的计算

手算

从左往右扫描，每遇到一个运算符，就让运算符前面最近的两个操作数执行对应运算，合体为一个操作数

- ① 从左往右扫描下一个元素，直到处理完所有元素
- ② 若扫描到操作数则压入栈，并回到①；否则执行③
- ③ 若扫描到运算符，则弹出两个栈顶元素，执行相应运算，运算结果压回栈顶，回到①

机算

中缀表达式转前缀表达式（手算）

- ① 确定中缀表达式中各个运算符的运算顺序
- ② 选择下一个运算符，按照「运算符 左操作数 右操作数」的方式组合成一个新的操作数
- ③ 如果还有运算符没被处理，就继续 ②

表达式求值

前缀表达式的计算

- ① 从右往左扫描下一个元素，直到处理完所有元素
- ② 若扫描到操作数则压入栈，并回到①；否则执行③
- ③ 若扫描到运算符，则弹出两个栈顶元素，执行相应运算，运算结果压回栈顶，回到①

中缀表达式转后缀表达式

手算

- ① 确定中缀表达式中各个运算符的运算顺序
- ② 选择下一个运算符，按照「左操作数 右操作数 运算符」的方式组合成一个新的操作数
- ③ 如果还有运算符没被处理，就继续 ②

“左优先”原则：只要左边的运算符能先计算，就优先算左边的

从左到右处理各个元素，直到末尾。可能遇到三种情况：

机算

- ① 遇到操作数。直接加入后缀表达式。
- ② 遇到界限符。遇到“(“ 直接入栈；遇到”)“ 则依次弹出栈内运算符并加入后缀表达式，直到弹出“(“ 为止。注意：“(“ 不加入后缀表达式。
- ③ 遇到运算符。依次弹出栈中优先级高于或等于当前运算符的所有运算符，并加入后缀表达式，若碰到“(“ 或栈空则停止。之后再当前运算符入栈。

中缀表达式的计算（用栈实现）

- 初始化两个栈，操作数栈和运算符栈
- 若扫描到操作数，压入操作数栈
- 若扫描到运算符或界限符，则按照“中缀转后缀”相同的逻辑压入运算符栈（期间也会弹出运算符，每当弹出一个运算符时，就需要再弹出两个操作数栈的栈顶元素并执行相应运算，运算结果再压回操作数栈）

递归

计算正整数的阶乘 $n!$

- 递归调用时，函数调用栈可称为“递归工作栈”
- 每进入一层递归，就将递归调用所需信息压入栈顶
- 每退出一层递归，就从栈顶弹出相应信息

求斐波那契数列

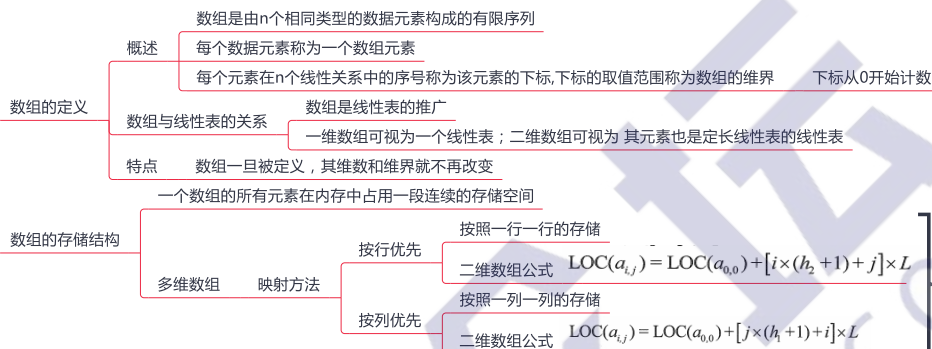
缺点：太多层递归可能会导致栈溢出

队列在计算机系统中的应用

- 解决主机与外部设备之间速度不匹配的问题 利用队列先进先出的性质，实现对打印机与主机速度不匹配的协调功能
- 解决由多用户引起的资源竞争问题 将多个用户排成一个队列，然后利用先进先出的性质分别将CPU分配给不同的用户使用
- 图的广度优先遍历

3.3~4矩阵的压缩存储（下）

特殊矩阵的压缩存储



考试的时候可以根据具体问题现场推演

矩阵的压缩存储



记不住公式的话记住推导过程,考试的时候可以
直接带入试试