INT202W02_数据结构：栈，队列，表

> 数据结构是存储和访问信息的具体方法。
>
> 我们研究不同类型的数据结构，因为对于特定的算法和数据类型，某些数据结构可能其他类型更合适。

栈（Stacks）

一种后进先出（LIFO）的数据结构。只能直接访问最后插入的元素。

抽象数据类型（Abstract Data Type (ADT)），支持以下操作：

- push(Obj) : 把object Obj 压到栈顶
- pop() :移除并返回栈顶的对象，如果空栈返回错误。
- initialize() : 初始化一个栈
- isEmpty() : 如果空栈返回True，反之False
- isFull() : 满栈？返回True，反之False

应用：

通过两个栈产生反转数组

```
ReverseArray(Data: values[])
    // Push the values from the array onto the stack.
    Stack: stack = New Stack
    For i = 0 To <length of values> - 1
        stack.Push(values[i])
    Next i
// Pop the items off the stack into the array.
    For i = 0 To <length of values> - 1
        values[i] = stack.Pop()
    Next i
End ReverseArra
```

算数表达式

栈可以执行算术表达式，若使用后缀表示法=逆波兰表示法（Reverse Polish notation, RPN）。

标准算数表达式是中缀表达式（infix notation）。在中缀表示法中，理解运算符的优先级很重要。例如4 + 3 * 9

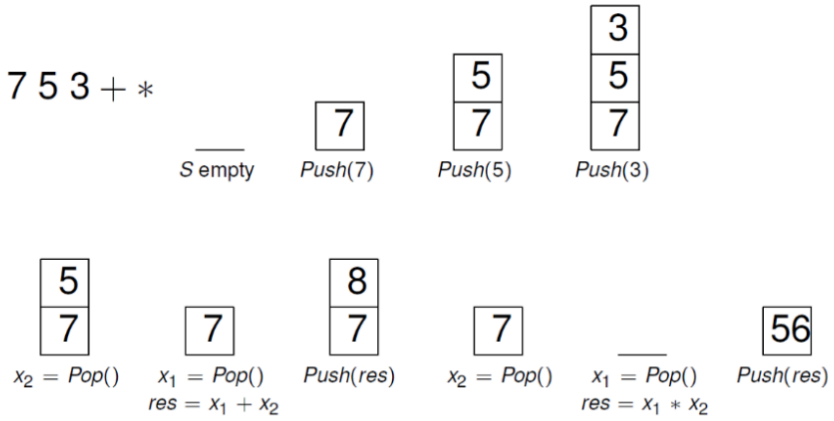
后缀表示法（postfix notation）中，操作数位于算术运算之前，例如 x y +，x y z + * 或 x y + z *。当你遇到一个运算符时，它按照顺序应用于列表中的前两个操作数。

x y + -> x+y

x y z + ∗ -> (y+z)*x

x y + z ∗ -> (x+y)*z

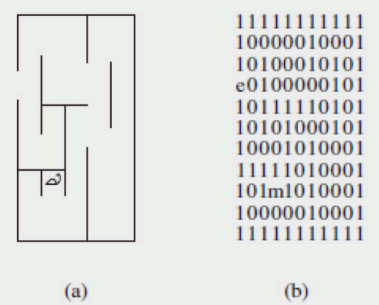
x y w z / − ∗ -> x ∗ (y − w/z) 注意顺序，新弹栈的元素在前面

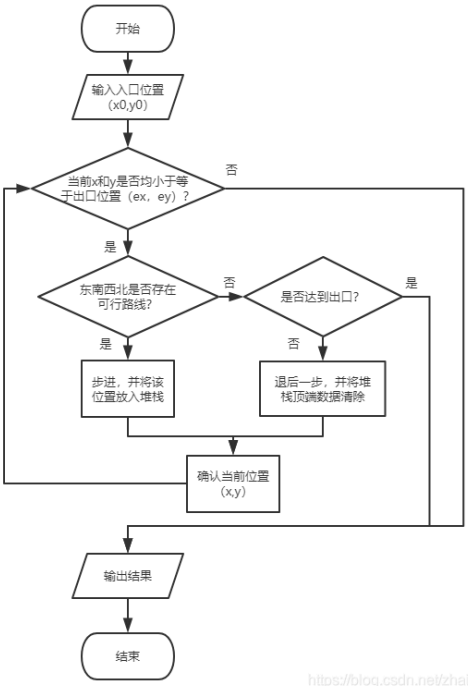


解迷宫

回溯求解法（深度优先）：

假设我们有一个矩阵表示的迷宫，1代表墙壁，0代表通路，e代表出口，m代表小老鼠：






```
exitMaze(){
    //初始化栈，出口位置，入口位置=当前位置
    initialize stack, exitCell, entryCell, currentCell = entryCell;
    //当前位置不是出口时
    while currentCell is not exitCell
        //标记当前位置已访问
        mark currentCell as visited;
        //将当前位置相邻的一个位置压入栈
        push onto the stack the unvisited neighbors of currentCell;
        //若栈为空（没有可能的路线了）则失败
        if stack is empty
            failure;
        //若还有未探索的位置，则弹栈作为新的位置
        else pop off a cell from the stack and make it currentCell;
    //直到是出口，赢了
    success;
}
```

时间复杂度：O(N×M)，最坏情况遍历整个迷宫。

在这里，栈用于储存待探索的位置，我们每次将未访问的相邻位置压入栈以供后续探索，从栈顶弹出一个位置用于探索新位置，若栈为空（没有可能的路线了）则说明迷宫无解。

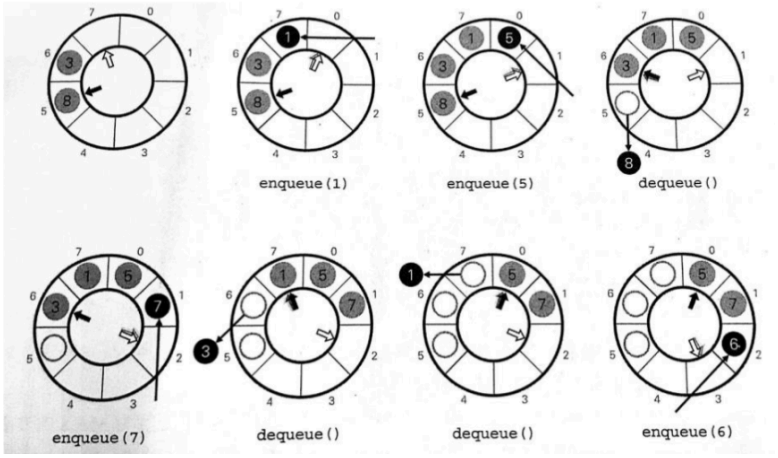
在使用DFS解迷宫时，必须确保不会重复访问已探索的单元格，需要储存访问过的单元（使用集合或者直接修改迷宫）。同时，DFS不保证能找到最短路径（BFS可以）。

队列（Queues）

先进先出（FIFO）的队列结构，对象可以插入队尾，但只能移除队首的元素

- enqueue(Obj): 将对象插入队尾
- dequeue(): 移除并返回队首元素，若队列为空则报错
- size(): 返回队列中对象数量
- isEmpty() : 如果空栈返回True，反之False
- isFull() : 满栈？返回True，反之False
- front(): 仅返回队首元素，若空报错

A circular queue holding the values 3 and 8



可用于多道程序设计（Multiprogramming）

从而实现有限的并行性，允许同时运行多个任务或线程。使用队列以轮询协议分配 CPU 时间给线程。

表（List）

List是项目（item）的集合，每个项目储存在一个节点（node）中，每个节点有一个数据部分和指向下一个元素的指针（双链表还可包含上一个元素的指针，还有跳表等等）。支持引用，更新（插入，删除），搜索。

引用方法（Referring methods）：

- first()：返回第一个元素的位置；如果列表 S 为空，则发生错误。
- last(): 返回最后一个元素的位置；如果列表 S 为空，则发生错误。

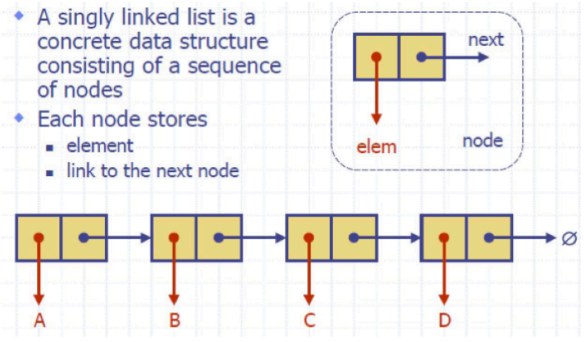
- isFirst(p): 如果元素 p 是列表中的第一个项目返回 true，否则返回 false。
- isLast(p): 如果元素 p 是列表中的最后一个元素返回 true，否则返回 false。
- before(p): 返回 S 中位于位置 p 之前的元素的位置；如果 p 是第一个元素，则报错。
- after(p): 返回 S 中位于位置 p 之后的元素的位置；如果 p 是最后一个元素，则报错。

更新方法（Update methods）：

- replaceElement(p,e): 替换p位置元素 p - position, e - element.
- swapElements(p,q): 交换pq位置元素 p,q - positions.
- insertFirst(e): 首插入元素 e - element.
- insertLast(e): 尾插入元素 e - element.
- insertBefore(p,e): 在p位置前插入元素 p - position, e - element.
- insertAfter(p,e): 在p位置后插入元素 p - position, e - element.
- remove(p): 移除p位置元素 p - position.

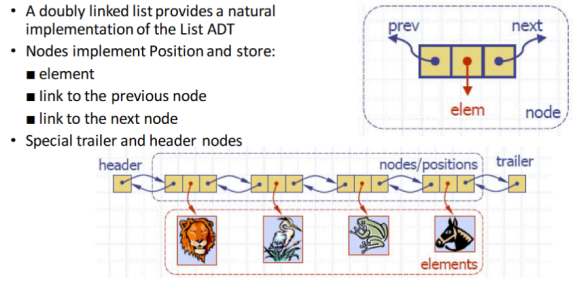
单链表（Singly-linked list）

单链表的每个节点储存一个指向下个节点的指针（或者叫link链接）（最后一个指向null）



双链表（doubly-linked list）

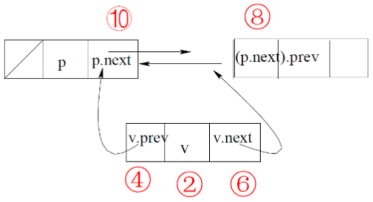
双链表的每个节点储存两个链接：指向下一个和上一个元素。接下来我们关注双链表。



元素插入

Pseudo-code for *insertAfter(p,e)* :

```
INSERTAFTER(p,e)
    //Create a new node v
2   v.element ← e
    //Link v to its predecessor
4   v.prev ← p
    //Link v to its successor
6   v.next ← p.next
    //Link p's old successor to v
8   (p.next).prev ← v
    //Link p to its new successor v
10  p.next ← v
11  return v
```



元素删除

The pseudo-code for *remove(p)*:

```
REMOVE(p)
    //Assign a temporary variable to hold return value
2   t ← p.element
    //Unlink p from list
4   (p.prev).next ← p.next
5   (p.next).prev ← p.prev
    //invalidate p
7   p.prev ← null
8   p.next ← null
9   return t
```