数据结构

第 7 周 – 栈：后进先出原则、实现、应用

Koczka Ferenc 博士 · 埃斯特哈齐·卡罗伊大学 · 2026

复习

- **数组**：直接访问，在末尾插入和删除是 **O(1)**
- **抽象数据类型（ADT）**：说明**能做什么**，而不是**怎么做**（第 1 周）
- **集合**：不同的表示方法，不同的效率

今天的问题

- 什么是栈？它为什么这么重要？
- 怎样用数组实现栈？
- 计算机怎样计算 `(3 + 4) * 2` 这样的表达式？

栈的概念

栈是一种只能访问**最后插入**的元素的数据结构。

LIFO – Last In, First Out（后进先出）：最后进去的最先出来。

生活中的例子

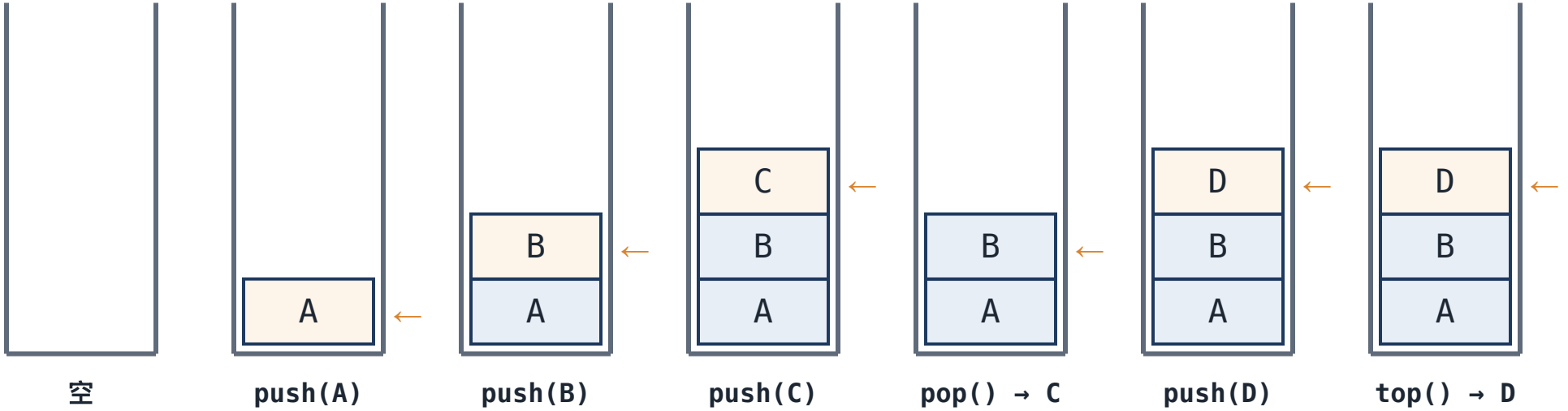
- 一摞盘子
- 一筒薯片
- 浏览器的“后退”按钮
- 文本编辑器中的“撤销”(Ctrl+Z)

性质

元素	同构
关系	线性
访问	**受限** – 只能访问栈顶
大小	静态或动态

栈的操作

栈顶 – 唯一可以插入和删除的位置



栈的操作 – 汇总

操作	含义	步数
Create	建立空栈	O(1)
Push	元素压入栈顶	O(1)
Pop	弹出并返回栈顶元素	O(1)
Top / Peek	读取栈顶元素但不删除	O(1)
IsEmpty? · IsFull?	查询状态	O(1)

非法操作

- **对空栈执行 Pop 或 Top** → **下溢**(stack underflow)
- **向满栈执行 Push** → **上溢**(stack overflow) – 只针对固定大小的栈

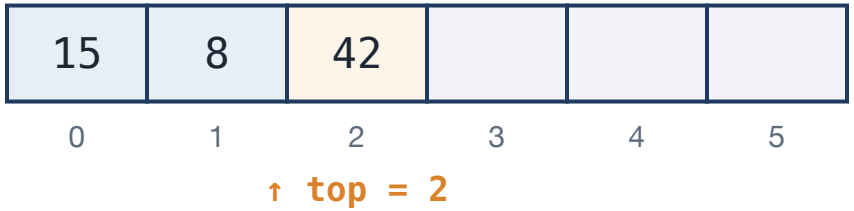
执行操作之前一定要检查 **IsEmpty?** / **IsFull?** – 或者让实现来报告错误（Java：异常，Go：返回 `error`）。

Stack Overflow 这个名字就来源于此：递归过深时，程序的**调用栈**会被填满（见后文和第 12 周）。

用数组实现

数组栈

数组栈：栈底在下标 0，栈顶在下标 top




```
push(x): top ← top + 1; T[top] ← x
pop(): x ← T[top]; top ← top − 1
空：top = −1 满：top = n − 1
```

均为 O(1)

- 栈的**底部**是数组的开头，**栈顶**在下标 `top` 处
- Push 和 Pop 都在数组的**末尾**进行 → 不需要移动元素 → **O(1)**

数组栈 – 伪代码

```
// 数组 T[1..n],top:最后一个元素的下标

Procedure Create
  top ← 0
End procedure

Function IsEmpty
  Return top = 0
End function

Function IsFull
  Return top = n
End function
```

```
Procedure Push(x)
  If IsFull then ERROR("overflow")
  else
    top ← top + 1
    T[top] ← x
  End if
End procedure

Function Pop
  If IsEmpty then ERROR("underflow")
  else
    x ← T[top]
    top ← top − 1
    Return x
  End if
End function
```

伪代码下标从 1 开始，所以空栈时 top = 0;Java/Go 下标从 0 开始，所以空栈时 top = −1。

数组栈 – Java 和 Go

Java

```java
class ArrayStack {
    private final int[] data;
    private int top = -1;

    ArrayStack(int capacity) {
        data = new int[capacity];
    }
    boolean isEmpty() { return top == -1; }
    boolean isFull()  { return top == data.length - 1; }

    void push(int x) {
        if (isFull())
            throw new IllegalStateException("full");
        data[++top] = x;
    }
    int pop() {
        if (isEmpty())
            throw new IllegalStateException("empty");
        return data[top--];
    }
}
```

Go – 泛型，用切片实现

```go
type Stack[T any] struct {
    items []T
}

func (s *Stack[T]) Push(x T) {
    s.items = append(s.items, x)
}

func (s *Stack[T]) Pop() T {
    n := len(s.items) - 1
    x := s.items[n]
    s.items = s.items[:n]
    return x
}

func (s *Stack[T]) IsEmpty() bool {
    return len(s.items) == 0
}
```

切片是动态数组 → 没有“满”的状态。

内置的栈

Java

```java
Deque<Integer> st = new ArrayDeque<>();
st.push(1);
st.push(2);
st.peek();     // 2
st.pop();      // 2
st.isEmpty();  // false
```

推荐使用 **`ArrayDeque`**，而不是旧的 `java.util.Stack` 类（更快）。

Go

```go
st := []int{}
st = append(st, 1)         // push
st = append(st, 2)
top := st[len(st)-1]       // peek
st = st[:len(st)-1]        // pop
empty := len(st) == 0
```

Go 没有单独的栈类型 – **切片**就完全可以胜任。

应用：表达式求值

中缀、前缀、后缀

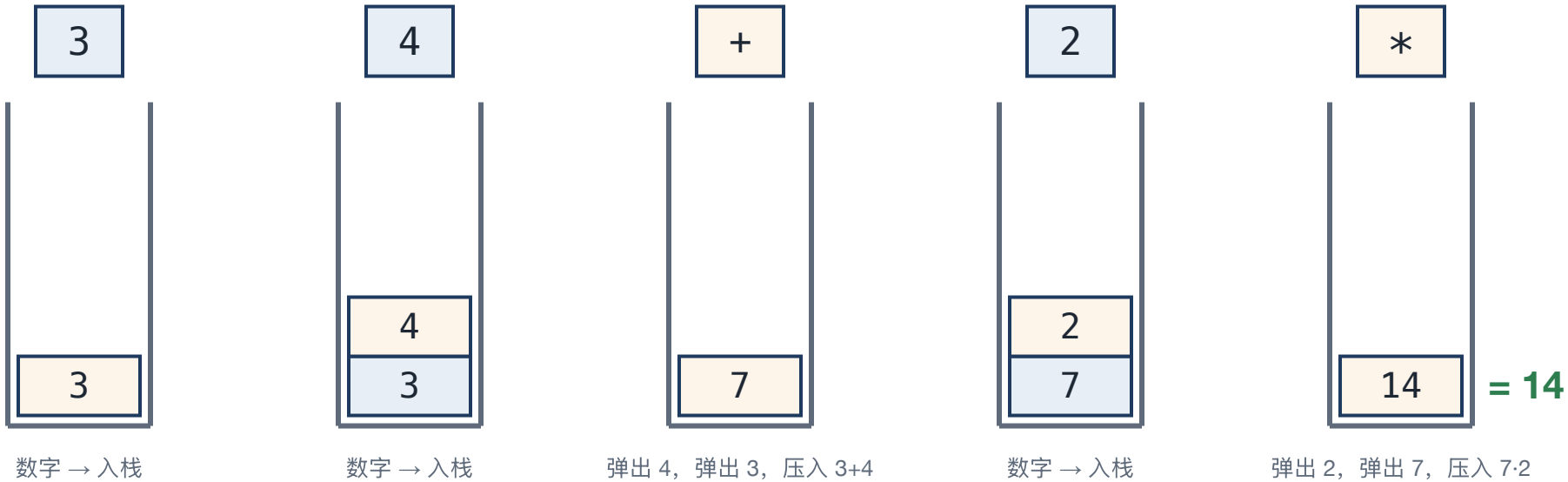
记法	例子	运算符的位置
中缀	`(3 + 4) * 2`	在操作数之间 – 需要括号
前缀（波兰式）	`* + 3 4 2`	在操作数之前
后缀（逆波兰式）	`3 4 + 2 *`	在操作数之后

后缀表达式的优点：

- **不需要括号**，也不需要优先级规则
- 可以从左往右，**一次遍历**，用一个栈完成求值
- 编译器、Java 虚拟机和 HP 计算器都是这样工作的

用栈计算后缀表达式

后缀表达式：3 4 + 2 * (= (3 + 4) · 2)



后缀表达式求值 – 算法

伪代码

```
Function PostfixValue(expression)
  Create
  For every token t from left to right
    If t is a number then
      Push(t)
    else                        // t 是运算符
      b ← Pop                   // 右操作数!
      a ← Pop                   // 左操作数
      Push(a t b)
    End if
  End for
  Return Pop
End function
```

右操作数先出来！ `5 2 -` → b = 2,a = 5 → 5 – 2 = 3

Java

```java
ArrayDeque<Double> st = new ArrayDeque<>();
for (String t : expr.split("\\s+")) {
    if (isOperator(t)) {
        double b = st.pop();
        double a = st.pop();
        st.push(apply(a, b, t));
    } else {
        st.push(Double.parseDouble(t));
    }
}
return st.pop();
```

步数： O(n) – 每个记号处理一次

非法表达式： 对空栈执行 Pop，或者最后剩下的元素不是恰好一个

例：`1 2 + 3 4 * - 9 /`

记号	操作	栈（自底向上）
1	入栈	1
2	入栈	1 2
+	1 + 2	3
3	入栈	3 3
4	入栈	3 3 4
*	3 · 4	3 12
–	3 – 12	–9
9	入栈	–9 9
/	–9 / 9	**–1**

中缀形式：((1 + 2) – 3 · 4) / 9 = –1

中缀转后缀：调度场算法

E. W. Dijkstra（1961），简化版本 – **运算符**进栈：

1. **操作数** → 直接输出
2. `(` → 入栈
3. `)` → 把栈中的元素依次输出，直到遇到第一个 `(`；丢弃 `(`
4. **运算符** → 只要栈顶是**优先级不低于**它的运算符，就把栈顶输出；然后把新运算符入栈
5. 最后，把栈中剩余的元素依次输出

中缀	后缀
`3 + 4 * 2`	`3 4 2 * +`
`(3 + 4) * 2`	`3 4 + 2 *`
`(a + b) * (c - d)`	`a b + c d - *`

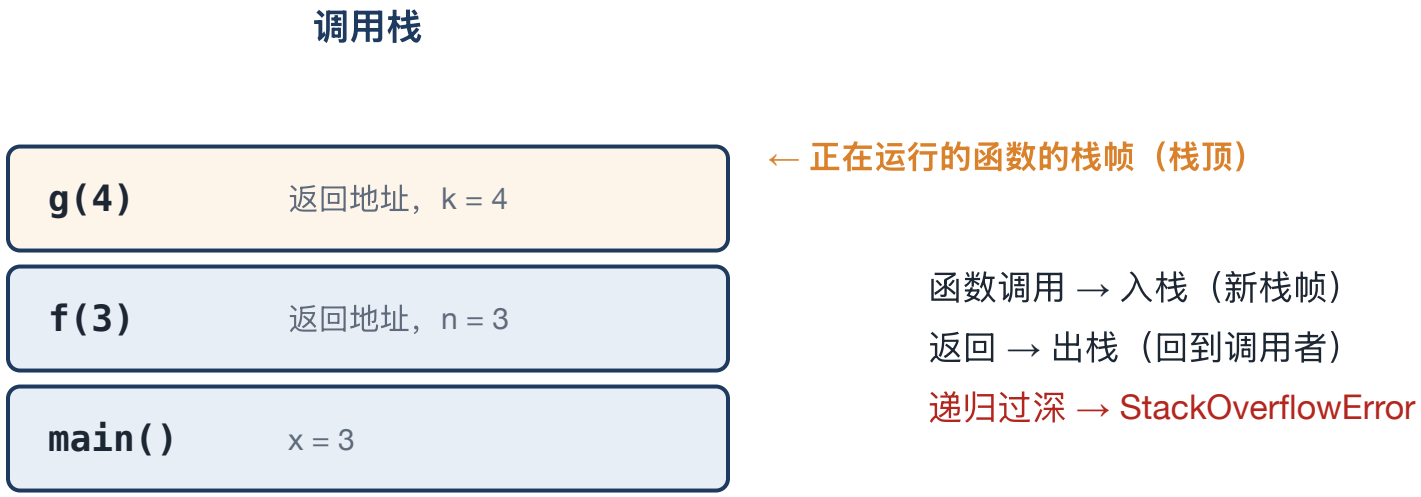
应用：括号匹配检查

括号是否正确？ `{[()()]}` ✓ · `([)]` ✗ · `((a + b)` ✗

```
Function Balanced(S)
  Create
  For every character c
    If c is an opening bracket then Push(c)
    else if c is a closing bracket then
      If IsEmpty then Return false          // 没有对应的左括号
      If Pop is not the pair of c then Return false
    End if
  End for
  Return IsEmpty                            // 还有没闭合的吗？
End function
```

应用：编译器、文本编辑器、HTML/XML 检查 · **O(n)**

应用：调用栈



每次函数调用都会把一个**栈帧**（局部变量、参数、返回地址）压入栈中 – **递归**正是靠它实现的（第 12 周）。

其他应用

- **撤销：**每次修改之前的状态压入栈中
- **浏览器历史：**“后退” = 出栈，打开新页面 = 入栈
- **回溯：**迷宫、数独、八皇后 – 栈中保存目前为止的各个决策
- 树和图的**深度优先遍历**
- **进制转换：**余数逆序输出 → 栈
- **字符串逆序：**每个字符入栈，然后全部出栈

小结

- **栈：**后进先出 – 只能访问栈顶
- 操作：**Push**、**Pop**、**Top**、**IsEmpty?**、**IsFull?** – 都是 **O(1)**
- **数组实现：**在数组末尾使用 `top` 下标；固定大小时可能上溢
- Java: `ArrayDeque` · Go：切片（`append`、`s[:len(s)-1]`）
- **后缀表达式求值：**一次遍历，一个栈，O(n) – 右操作数先出来！
- 应用：括号匹配、撤销、调用栈、回溯

下周：队列 – 先进先出原则、循环队列、双端队列、优先队列。

复习题

1. 输出什么？`push(1); push(2); pop(); push(3); push(4); pop(); pop(); pop();`（pop 输出该元素）
2. 用栈求值，并写出每一步之后栈的状态：`6 2 / 3 1 - *`
3. 转换成后缀形式：`a * (b + c) - d / e`
4. 对 `n = 50`，下面的代码输出什么？一般地，它做了什么？

```
while (n > 0) { s.push(n % 2); n = n / 2; } while (!s.isEmpty())
print(s.pop());
```

5. 判断对错：在数组实现中，Pop 需要 O(n) 步，因为要移动元素。

实践课

✎ 示例程序

程序	Java	Go	演示内容
数组栈	`ArrayStack.java`	`w07/arraystack`	push、pop、上溢/下溢、泛型栈
后缀表达式	`Postfix.java`	`w07/postfix`	求值，打印栈的状态
中缀 → 后缀	`InfixToPostfix.java`	`w07/infixtopostfix`	调度场算法
括号	`Brackets.java`	`w07/brackets`	检查 `()` 、 `[]` 、 `{}`

✎ 练习

1. **二进制形式：**用栈把十进制数转换成二进制！
2. **逆序：**用栈把文本逆序！（在 Go 中，中文文本也要正确！）
3. **最小栈：**实现一个 `min()` 操作也是 O(1) 的栈！
4. **撤销：**模拟文本编辑器的撤销功能！
5. **代码阅读：**字母 = 入栈，`*` = 出栈并输出。输出什么？`D A T * A * S * T R * U C * T`
6. **相邻对：**反复删除相邻的相同字母对！（`"abbaca" → "ca"`）

参考答案：`Exercises07.java`、`w07/exercises`

✎ 课后作业

1. 为后缀表达式求值程序增加 `^`（乘方）和 `%`（取余）运算符！
2. 扩展中缀 → 后缀转换程序，使它支持**多位数**（例如 `12 + 345 * 6`）！
3. 编写程序，用两个栈实现一个**队列**（先进先出）！各操作需要多少步？
4. 编写程序，检查 HTML 文本中的标签（`<b>…</b>`、`<i>…</i>`）是否正确嵌套！