Python 数据结构与算法分析（第三章 基本数据结构）

1. 基本数据结构

- **线性数据结构**：一旦某个元素被添加进来，它与前后元素的相对位置将保持不变，这样的数据集合经常被称为线性数据结构。

  真正区分线性数据结构的是元素的添加方式和移除方式，尤其是添加操作和移除操作发生的位置。
- **栈**：栈有时也被称作“下推栈”。它是有序集合，添加操作和移除操作总发生在同一端，即“顶端”（last-in first-out, LIFO），另一端则被称为“底端”。
- **中序表达式**：运算符出现在操作数的中间，如$A + B$。
- **前序表达式和后序表达式**：所有运算符出现在它所作用的两个操作数之前为前序表达式，后序表达式则相反。如$+AB$; $AB+$。

  对于计算机，中序表达式需要明确以何种顺序进行何种运算，为避免歧义需要使用括号构造完全括号表达式。而对于前序和后序表达式，运算顺序完全由运算符位置决定。故，中序表达式是最不理想的表达式。

  栈数据结构的相关例子包括：括号匹配、十进制数转为任意进制、中序表达式转换为前序及后序表达式等。

- **队列**：队列是有序集合，添加操作发生在“尾部”，移除操作则发生在“头部”（first-in first-out, FIFO）。

  队列数据结构相关例子包括：打印任务队列等。

- **双端队列**：双端队列是与队列类似的有序集合。它有一前、一后两端，元素在其中保持自己的位置。与队列不同的是，双端队列对在哪一端添加和移除元素没有任何限制。新元素既可以被添加到前端，也可以被添加到后端。同理，已有的元素也能从任意一端移除。
- **链表**：一种线性表，但是并不会按线性的顺序存储数据，而是在每一个节点里存到下一个节点的指针。

2. 编程练习

（1）后序表达式

```
def calculate_backward(formula):
    list_temp = []
    for i in formula:
        if i not in ['+', '-', '*', '/']+[str(i) for i in range(10)]:
            assert print('Error! Please check the input.')
        elif i in [str(i) for i in range(10)]:
            list_temp.append(i)
        else:
            if len(list_temp) > 1:
                num_1 = int(list_temp.pop())
                num_2 = int(list_temp.pop())
            else:
                assert print('Error! Please check the input.')
            if i == '+':
                temp_result = num_1 + num_2
            elif i == '-':
                temp_result = num_1 - num_2
            elif i == '*':
                temp_result = num_1 * num_2
            else:
                temp_result = num_1 / num_2
            list_temp.append(temp_result)
    return list_temp.pop()


backward_formula = '12345*+*+'
print('The result of {} is {}.'.format(backward_formula, calculate_backward('12345*+*+')))
```

> The result of 12345*+*+ is 47.

(2) 基数排序器（桶排序）

```
def buck_sort(list_unsort):
    buck_pool = [[] for _ in range(10)]  ## 桶初始化
    bit_compare = 1  ## 比较位
    while len(buck_pool[0]) < len(list_unsort):
        ## 递归，至所有数字最高位被比较
        buck_pool = [[] for _ in range(10)]  ## 清空桶
        ## 基于比较位入桶（比较位==桶编号）
        for i in list_unsort:
            if len(str(i)) >= bit_compare:
                buck_pool[int(str(i)[-bit_compare])].append(i)
            else:
                buck_pool[0].append(i)
        print("比较位：{}；各桶数据：{}".format(bit_compare, buck_pool))
        bit_compare += 1  ## 比较位自增
        list_unsort = sum(buck_pool, [])  ## 合并，入主桶
    return list_unsort


print("桶排序结果：{}".format(buck_sort([1, 354312, 53, 234, 53415, 12343, 33, 23, 43, 2, 53, 876])))
```

> 比较位：1；各桶数据：$[[], [1], [354312, 2], [53, 12343, 33, 23, 43, 53], [234], [53415], [876], [], [], []]$
> 比较位：2；各桶数据：$[[1, 2], [354312, 53415], [23], [33, 234], [12343, 43], [53, 53], [], [876], [], []]$
> 比较位：3；各桶数据：$[[1, 2, 23, 33, 43, 53, 53], [], [234], [354312, 12343], [53415], [], [], [], [876], []]$
> 比较位：4；各桶数据：$[[1, 2, 23, 33, 43, 53, 53, 234, 876], [], [12343], [53415], [354312], [], [], [], [], []]$
> 比较位：5；各桶数据：$[[1, 2, 23, 33, 43, 53, 53, 234, 876], [12343], [], [], [], [53415, 354312], [], [], [], []]$
> 比较位：6；各桶数据：$[[1, 2, 23, 33, 43, 53, 53, 234, 876, 12343, 53415], [], [], [354312], [], [], [], [], [], []]$
> 比较位：7；各桶数据：$[[1, 2, 23, 33, 43, 53, 53, 234, 876, 12343, 53415, 354312], [], [], [], [], [], [], [], [], []]$
> 桶排序结果：$[1, 2, 23, 33, 43, 53, 53, 234, 876, 12343, 53415, 354312]$

3. 参考文献

- Python数据结构与算法分析（第2版）