【栈】空栈压数

题目描述与示例

题目描述

向一个空栈压入**正整数**，每当压入一个整数时，执行以下规则（设: **栈顶至栈底**整数依次编号为 `n1, n2, ..., nx`，其中 `n1` 为最新压入的整数)

1. 如果 `n1 = n2`，则 `n1`、`n2` 全部出栈，压入新数据 `m (m = 2*n1)`
2. 如果 `n1 = n2 + ... + ny`(`y` 的范围为 `[3,x]`)，则 `n1, n2, ..., ny` 全部出栈，压入新数据 `m (m = 2*n1)`。
3. 如果上述规则都不满足，则不做操作。

如：依次向栈压入 `6、1、2、3`，当压入 `2` 时，栈顶至栈底依次为 `[2,1,6]`；当压入 `3` 时，`3 = 2 + 1`，`3、2、1` 全部出栈，重新入栈整数 `6`，此时栈顶至栈底依次为 `[6,6]`；`6 = 6`，两个 `6` 全部出栈，压入 `12`，最终栈中只剩个元素 `12`。

向栈中输入一串数字，请输出应用此规则后栈中最终存留的数字。

输入描述

使用单个空格隔开的正整数的字符串，如 `"5 6 7 8"`，左边的数字先入栈。

- 正整数大小为 `[1, 2^31-1]`。
- 正整数个数为 `[1,1000]`。

输出描述

最终栈中存留的元素值，元素值使用单个空格隔开，如 `"8 7 6 5"`，从左至右依次为**栈顶至栈底**的数字。

示例

输入

```
10 20 50 80 1 1
```

输出

```
2 160
```

解题思路

> 注意，本题最初见于华为2023年校招暑期实习的题目，在OJ上直接搜索**空栈压数**即可搜索得到题目。

由于本题的数据量较小，最多仅为 `1000` 。可以直接使用栈的数据结构，对题目进行模拟即可。

代码

Python

```python
# 题目：【栈】2024E-空栈压数
# 作者：闭着眼睛学数理化
# 算法：栈，模拟
# 代码有看不懂的地方请直接在群上提问


# 检查是否需要弹出若干栈顶元素并进行压缩的函数
def check_stack_top(stack, num):
    global flag_continue_loop
    # 初始化栈顶元素的和为0
    top_sum = 0
```

```
    # 初始化栈顶元素索引为idx，逆序遍历
    idx = len(stack) - 1
    # 进行循环，满足以下两个条件之一，则退出循环：
    # 1. 栈顶元素大于等于num
    # 2. 逆序遍历的索引idx小于0
    while top_sum < num and idx >= 0:
        # 在循环中，
        # 栈顶元素和递增
        # 栈顶元素索引递减
        top_sum += stack[idx]
        idx -= 1
    # 退出循环后，若栈顶元素和top_sum恰好等于num
    if top_sum == num:
        # 需要继续进行循环，故设置flag为True
        flag_continue_loop = True
        # 需要删除之前计入栈顶元素和中的所有元素，用切片操作即可
        # num更新为原来的两倍，返回这两个参数
        return stack[:idx+1], num * 2
    # 否则
    else:
        # 无需继续进行循环，故设置flag为False
        flag_continue_loop = False
        # 返回原先的stack和num
        return stack, num


# 输入待操作的列表nums，从左到右为压栈的顺序
nums = list(map(int, input().split()))

# 初始化一个空栈
stack = list()

# 从左到右，遍历每一个数字
for num in nums:
    # 设置变量flag_continue_loop为True
    # flag_continue_loop的值会在函数check_stack_top()中进行修改
    flag_continue_loop = True
    while flag_continue_loop:
        # 调用check_stack_top()函数，更新stack和num的情况
        stack, num = check_stack_top(stack, num)
    # 空栈的情况实际上也包含在上述函数中了
    # 做完计算，必须把最新得到的num加入栈中
    stack.append(num)


# 题目要求输出从栈顶到栈底的数字
# 故需要逆序输出栈中元素
```

```
print(" ".join(str(num) for num in stack[::-1]))
```

Java

```
import java.util.Scanner;
import java.util.Stack;

public class Main {

    static boolean flagContinueLoop;

    public static int checkStackTop(Stack<Integer> stack, int num) {
        int topSum = 0;
        int idx = stack.size() - 1;

        while (topSum < num && idx >= 0) {
            topSum += stack.get(idx);
            idx--;
        }

        if (topSum == num) {
            flagContinueLoop = true;
            while (stack.size() > idx + 1) {
                stack.pop();
            }
            return num * 2;
        } else {
            flagContinueLoop = false;
            return num;
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Stack<Integer> nums = new Stack<>();

        while (scanner.hasNext()) {
            int num = scanner.nextInt();
            nums.push(num);
        }

        Stack<Integer> stack = new Stack<>();
        for (int num : nums) {
            flagContinueLoop = true;
            while (flagContinueLoop) {
```

```
                num = checkStackTop(stack, num);
            }
            stack.push(num);
        }

        while (!stack.isEmpty()) {
            System.out.print(stack.pop() + " ");
        }
    }
}

```

C++

```
#include <iostream>
#include <vector>
using namespace std;

int checkStackTop(vector<int>& stack, int num, bool& flagContinueLoop) {
    int topSum = 0;
    int idx = stack.size() - 1;

    while (topSum < num && idx >= 0) {
        topSum += stack[idx];
        idx--;
    }

    if (topSum == num) {
        flagContinueLoop = true;
        while (stack.size() > idx + 1) {
            stack.pop_back();
        }
        return num * 2;
    } else {
        flagContinueLoop = false;
        return num;
    }
}

int main() {
    vector<int> nums;
    int num;
    while (cin >> num) {
```

```
        nums.push_back(num);
        if (cin.get() == '\n') break;
    }

    vector<int> stack;

    for (int num : nums) {
        bool flagContinueLoop = true;
        while (flagContinueLoop) {
            num = checkStackTop(stack, num, flagContinueLoop);
        }
        stack.push_back(num);
    }

    for (int i = stack.size() - 1; i >= 0; --i) {
        cout << stack[i] << " ";
    }
    return 0;
}

```

时空复杂度

时间复杂度：`O(N^2)`。对于遍历到的每一个元素`num`，栈中的元素都要检查一次，故最差的时间复杂度为`O(N^2)`

空间复杂度：`O(N)`。栈所占空间。