【门徒计划】第三周刷题代码

栈的基本操作

Leetcode-面试题 03.04-化栈为队

```
class MyQueue {
public:
    stack<int> s1, s2;
    /** Initialize your data structure here. */
    MyQueue() {}

    /** Push element x to the back of queue. */
    void push(int x) {
        s2.push(x);
        return ;
    }

    void transfer() {
        if (!s1.empty()) return ;
        while (!s2.empty()) {
            s1.push(s2.top());
            s2.pop();
        }
        return ;
    }
    /** Removes the element from in front of queue and returns that element. */
    int pop() {
        transfer();
        int ret = s1.top();
        s1.pop();
        return ret;
    }

    /** Get the front element. */
    int peek() {
        transfer();
        return s1.top();
    }

    /** Returns whether the queue is empty. */
    bool empty() {
        return s1.empty() && s2.empty();
    }
};

/**
 * Your MyQueue object will be instantiated and called as such:
 * MyQueue* obj = new MyQueue();
 * obj->push(x);
 * int param_2 = obj->pop();
 * int param_3 = obj->peek();
 * bool param_4 = obj->empty();
 */
```

Leetcode-682-棒球比赛

```
class Solution {
public:
    int calPoints(vector<string>& ops) {
        stack<int> s;
        for (int i = 0; i < ops.size(); i++) {
            if (ops[i] == "+") {
                int a = s.top(); s.pop();
                int b = s.top();
                s.push(a), s.push(a + b);
            } else if (ops[i] == "D") {
                s.push(2 * s.top());
            } else if (ops[i] == "C") {
                s.pop();
            } else {
                s.push(atoi(ops[i].c_str()));
            }
        }
        int sum = 0;
        while (!s.empty()) {
            sum += s.top();
            s.pop();
        }
        return sum;
```

```cpp
    }
};
```

Leetcode-844-比较含退格的字符串

```cpp
class Solution {
public:
    void transform(string S, stack<char> &s) {
        for (int i = 0; i < S.size(); i++) {
            if (S[i] == '#' && !s.empty()) s.pop();
            else if (S[i] != '#') s.push(S[i]);
        }
        return ;
    }
    bool backspaceCompare(string S, string T) {
        stack<char> s;
        stack<char> t;
        transform(S, s);
        transform(T, t);
        if (s.size() - t.size()) return false;
        while (!s.empty()) {
            if (s.top() != t.top()) return false;
            s.pop(), t.pop();
        }
        return true;
    }
};
```

Leetcode-946-验证栈序列

```cpp
class Solution {
public:
    bool validateStackSequences(vector<int>& pushed, vector<int>& popped) {
        stack<int> s;
        for (int i = 0, j = 0; i < popped.size(); i++) {
            while (j < pushed.size() && (s.empty() || s.top() != popped[i])) {
                s.push(pushed[j]);
                j += 1;
            }
            if (s.top() != popped[i]) return false;
            s.pop();
        }
        return true;
    }
};
```

Leetcode-20-有效的括号

```cpp
class Solution {
public:
    bool isValid(string s) {
        stack<char> ss;
        unordered_map<char, char> valid;
        valid[')'] = '(';
        valid[']'] = '[';
        valid['}'] = '{';
        for (int i = 0; i < s.size(); i++) {
            switch (s[i]) {
                case '(':
                case '[':
                case '{': ss.push(s[i]); break;
                case ')':
                case ']':
                case '}': if (ss.empty() || valid[s[i]] != ss.top()) return 
false; ss.pop(); break;
            }
        }
        return ss.empty();
    }
};
```

Leetcode-1021-删除最外层的括号

```cpp
class Solution {
public:
    string removeOuterParentheses(string S) {
        string ret;
        for (int i = 0, pre = 0, cnt = 0; i < S.size(); i++) {
            if (S[i] == '(') cnt += 1;
            else cnt -= 1;
            if (cnt != 0) continue;
            ret += S.substr(pre + 1, i - pre - 1);
            pre = i + 1;
        }
        return ret;
    }
};
```

Leetcode-1249-移除无效的括号

```cpp
class Solution {
public:
    string minRemoveToMakeValid(string s) {
        char *t = new char[s.size() + 1];
        char *ans = new char[s.size() + 1];
        int tlen = 0;
        for (int i = 0, cnt = 0; i < s.size(); i++) {
            if (s[i] == '(' || s[i] != ')') {
                cnt += (s[i] == '(');
                t[tlen++] = s[i];
            } else {
                if (cnt == 0) continue;
                cnt -= 1;
                t[tlen++] = ')';
            }
        }
        ans[tlen] = '\0';
        int ans_head = tlen;
        for (int i = tlen - 1, cnt = 0; i >= 0; i--) {
            if (t[i] == ')' || t[i] != '(') {
                cnt += (t[i] == ')');
                ans[--ans_head] = t[i];
            } else {
                if (cnt == 0) continue;
                cnt -= 1;
                ans[--ans_head] = '(';
            }
        }
        return string(ans + ans_head);
    }
};
```

Leetcode-145-二叉树的后序遍历

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    vector<int> postorderTraversal(TreeNode* root) {
        if (root == nullptr) return vector<int>();
        vector<int> ans;
        stack<TreeNode *> s1; // 递归过程中的局部变量
        stack<int> s2;        // 递归到的程序位置
        s1.push(root);
        s2.push(0);
        while (!s1.empty()) {
            int status = s2.top();
            s2.pop();
            switch (status) {
                case 0: {
                    s2.push(1);
                    if (s1.top()->left != nullptr) {
                        s1.push(s1.top()->left);
                        s2.push(0);
                    }
                } break;
                case 1: {
                    s2.push(2);
                    if (s1.top()->right != nullptr) {
                        s1.push(s1.top()->right);
                        s2.push(0);
                    }
                } break;
                case 2: {
                    ans.push_back(s1.top()->val);
                    s1.pop();
                } break;
            }
        }
        return ans;
    }
};
```

Leetcode-331-验证二叉树的前序序列化

```cpp
class Solution {
public:
    bool isValidSerialization(string preorder) {
        vector<string> s;
        for (int i = 0, j = 0; i < preorder.size(); i = j + 1) {
            j = i;
            while (j < preorder.size() && preorder[j] != ',') ++j;
            s.push_back(preorder.substr(i, j - i));
            int last = s.size() - 1;
            while (s.size() >= 3 && s[last] == "#"
            && s[last - 1] == "#" && s[last - 2] != "#") {
                s[last - 2] = "#";
                s.pop_back();
                s.pop_back();
                last = s.size() - 1;
            }
        }
        return s.size() == 1 && s[0] == "#";
    }
};
```

Leetcode-227-基本计算器II

```
class Solution {
public:
    int level(char c) {
        switch (c) {
            case '@': return -1;
            case '+':
            case '-': return 1;
            case '*':
            case '/': return 2;
        }
        return 0;
    }
    int calc(int a, char op, int b) {
        switch (op) {
            case '+': return a + b;
            case '-': return a - b;
            case '*': return a * b;
            case '/': return a / b;
        }
        return 0;
    }
    int calculate(string s) {
        stack<int> num;
        stack<char> ops;
        s += "@";
        for (int i = 0, n = 0; i < s.size(); i++) {
            if (s[i] == ' ') continue;
            if (level(s[i]) == 0) {
                n = n * 10 + (s[i] - '0');
                continue;
            }
            num.push(n);
            n = 0;
            while (!ops.empty() && level(s[i]) <= level(ops.top())) {
                int b = num.top(); num.pop();
                int a = num.top(); num.pop();
                num.push(calc(a, ops.top(), b));
                ops.pop();
            }
            ops.push(s[i]);
        }
        return num.top();
    }
};
```

Leetcode-636-函数的独占时间

```cpp
class Solution {
public:
    vector<int> exclusiveTime(int n, vector<string>& logs) {
        vector<int> ans(n);
        stack<int> vID;
        for (int i = 0, pre = 0; i < logs.size(); i++) {
            int pos1 = logs[i].find_first_of(":");
            int pos2 = logs[i].find_last_of(":");
            string id_str   = logs[i].substr(0, pos1);
            string status   = logs[i].substr(pos1 + 1, pos2 - pos1 - 1);
            string time_str = logs[i].substr(pos2 + 1, logs[i].size());
            int id = atoi(id_str.c_str());
            int time_stamp = atoi(time_str.c_str());
            if (!vID.empty()) ans[vID.top()] += time_stamp - pre + (status ==
"end");
            pre = time_stamp + (status == "end");
            if (status == "start") vID.push(id);
            else vID.pop();
        }
        return ans;
    }
};
```

Leetcode-1124-表现良好的最长时间段

```cpp
class Solution {
public:
    int longestWPI(vector<int>& hours) {
        unordered_map<int, int> ind;
        unordered_map<int, int> f;
        ind[0] = -1;
        f[0] = 0;
        int cnt = 0, ans = 0;
        for (int i = 0; i < hours.size(); i++) {
            if (hours[i] > 8) cnt += 1;
            else cnt -= 1;
            if (ind.find(cnt) == ind.end()) {
                ind[cnt] = i;
                if (ind.find(cnt - 1) == ind.end()) f[cnt] = 0;
                else f[cnt] = f[cnt - 1] + (i - ind[cnt - 1]);
            }
            if (ind.find(cnt - 1) == ind.end()) continue;
            ans = max(ans, i - ind[cnt - 1] + f[cnt - 1]);
        }
        return ans;
    }
};
```