【门徒计划】第二周刷题代码

链表复习题

Leetcode-86-分隔链表

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        ListNode r1, r2, *p1 = &r1, *p2 = &r2, *p = head, *q;
        while (p) {
            q = p->next;
            if (p->val < x) {
                p->next = p1->next;
                p1->next = p;
                p1 = p;
            } else {
                p->next = p2->next;
                p2->next = p;
                p2 = p;
            }
            p = q;
        }
        p1->next = r2.next;
        return r1.next;
    }
};
```

Leetcode-138-复制带随机指针的链表

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* next;
    Node* random;

    Node(int _val) {
        val = _val;
        next = NULL;
        random = NULL;
    }
};
*/

class Solution {
public:
    Node* copyRandomList(Node* head) {
        if (head == nullptr) return nullptr;
        Node *p = head, *q, *new_head;
        while (p) {
            q = new Node(p->val);
            q->random = p->random;
            q->next = p->next;
            p->next = q;
            p = q->next;
        }
        p = head->next;
        while (p) {
            if (p->random) p->random = p->random->next;
            (p = p->next) && (p = p->next);
        }
        new_head = head->next;
        p = head;
        while (p) {
            q = p->next;
            p->next = q->next;
            if (p->next) q->next = p->next->next;
            p = p->next;
        }
        return new_head;
    }
};
```

队列的封装与使用

Leetcode-622-设计循环队列

```cpp
class MyCircularQueue {
public:
    vector<int> arr;
    int head, tail, cnt;
    MyCircularQueue(int k) : arr(k), head(0), tail(0), cnt(0) {}

    bool enQueue(int value) {
        if (isFull()) return false;
        arr[tail] = value;
        tail = (tail + 1) % arr.size();
        cnt += 1;
        return true;
    }

    bool deQueue() {
        if (isEmpty()) return false;
        head = (head + 1) % arr.size();
        cnt -= 1;
        return true;
    }

    int Front() {
        if (isEmpty()) return -1;
        return arr[head];
    }

    int Rear() {
        if (isEmpty()) return -1;
        return arr[(tail - 1 + arr.size()) % arr.size()];
    }

    bool isEmpty() {
        return cnt == 0;
    }

    bool isFull() {
        return cnt == arr.size();
    }
};

/**
 * Your MyCircularQueue object will be instantiated and called as such:
 * MyCircularQueue* obj = new MyCircularQueue(k);
 * bool param_1 = obj->enQueue(value);
 * bool param_2 = obj->deQueue();
 * int param_3 = obj->Front();
 * int param_4 = obj->Rear();
 * bool param_5 = obj->isEmpty();
 * bool param_6 = obj->isFull();
 */
```

Leetcode-641-设计循环双端队列

```cpp
class MyCircularDeque {
public:
    /** Initialize your data structure here. Set the size of the deque to be k. */
    vector<int> arr;
    int cnt, head, tail;
    MyCircularDeque(int k) : arr(k), head(0), tail(0), cnt(0) {}

    /** Adds an item at the front of Deque. Return true if the operation is successful. */
    bool insertFront(int value) {
        if (isFull()) return false;
        head = head - 1;
        if (head == -1) head = arr.size() - 1;
        arr[head] = value;
        cnt += 1;
        return true;
    }

    /** Adds an item at the rear of Deque. Return true if the operation is successful. */
    bool insertLast(int value) {
        if (isFull()) return false;
        arr[tail] = value;
        tail += 1;
        if (tail == arr.size()) tail = 0;
        cnt += 1;
        return true;
    }

    /** Deletes an item from the front of Deque. Return true if the operation is successful. */
    bool deleteFront() {
        if (isEmpty()) return false;
        head = (head + 1) % arr.size();
        cnt -= 1;
        return true;
    }

    /** Deletes an item from the rear of Deque. Return true if the operation is successful. */
    bool deleteLast() {
        if (isEmpty()) return false;
        tail = (tail - 1 + arr.size()) % arr.size();
        cnt -= 1;
        return true;
    }

    /** Get the front item from the deque. */
    int getFront() {
        if (isEmpty()) return -1;
        return arr[head];
    }

    /** Get the last item from the deque. */
    int getRear() {
        if (isEmpty()) return -1;
        return arr[(tail - 1 + arr.size()) % arr.size()];
    }

    /** Checks whether the circular deque is empty or not. */
    bool isEmpty() {
        return cnt == 0;
    }

    /** Checks whether the circular deque is full or not. */
    bool isFull() {
        return cnt == arr.size();
    }
};

/**
 * Your MyCircularDeque object will be instantiated and called as such:
 * MyCircularDeque* obj = new MyCircularDeque(k);
 * bool param_1 = obj->insertFront(value);
 * bool param_2 = obj->insertLast(value);
 * bool param_3 = obj->deleteFront();
 * bool param_4 = obj->deleteLast();
 * int param_5 = obj->getFront();
 * int param_6 = obj->getRear();
 * bool param_7 = obj->isEmpty();
 * bool param_8 = obj->isFull();
 */
```

Leetcode-1670-设计前中后队列

```cpp
class Node {
public :
    int val;
    Node *next, *pre;
    Node(int val = 0, Node *next = nullptr, Node *pre = nullptr) :
val(val), next(next), pre(pre) {}
    void insert_pre(Node *p) {
        p->pre = pre;
        p->next = this;
        if (this->pre) this->pre->next = p;
        this->pre = p;
        return ;
    }
    void insert_next(Node *p) {
        p->pre = this;
        p->next = this->next;
        if (this->next) this->next->pre = p;
        this->next = p;
        return ;
    }
    void delete_pre() {
        if (this->pre == nullptr) return ;
        Node *p = this->pre;
        this->pre = p->pre;
        if (p->pre) p->pre->next = this;
        delete p;
        return ;
    }
    void delete_next() {
        if (this->next == nullptr) return ;
        Node *p = this->next;
        this->next = p->next;
        if (p->next) p->next->pre = this;
        delete p;
        return ;
    }
};

class Queue {
public :
    Node head, tail;
    int cnt;
    Queue() : cnt(0) {
        head.next = &tail;
        head.pre = nullptr;
        tail.next = nullptr;
        tail.pre = &head;
    }
    void push_back(int val) {
        tail.insert_pre(new Node(val));
        cnt += 1;
        return ;
    }
    void push_front(int val) {
        head.insert_next(new Node(val));
        cnt += 1;
        return ;
    }
    int pop_back() {
        if (isEmpty()) return -1;
        int ret = tail.pre->val;
        tail.delete_pre();
        cnt -= 1;
        return ret;
    }
    int pop_front() {
        if (isEmpty()) return -1;
        int ret = head.next->val;
        head.delete_next();
        cnt -= 1;
        return ret;
    }
    int front() {
        return head.next->val;
    }
    int back() {
        return tail.pre->val;
    }
    bool isEmpty() {
        return head.next == &tail;
    }
    int size() {
        return cnt;
    }
};

class FrontMiddleBackQueue {
public:
    Queue q1, q2;
    FrontMiddleBackQueue() {}

    void pushFront(int val) {
        q1.push_front(val);
        update();
        return ;
    }

    void pushMiddle(int val) {
        if (q1.size() > q2.size()) {
            q2.push_front(q1.back());
            q1.pop_back();
        }
        q1.push_back(val);
        return ;
    }

    void pushBack(int val) {
        q2.push_back(val);
        update();
        return ;
    }

    int popFront() {
        if (isEmpty()) return -1;
        int ret = q1.pop_front();
        update();
        return ret;
    }

    int popMiddle() {
        if (isEmpty()) return -1;
        int ret = q1.pop_back();
        update();
        return ret;
    }

    int popBack() {
        if (isEmpty()) return -1;
        int ret;
        if (q2.isEmpty()) {
            ret = q1.pop_back();
        } else {
            ret = q2.pop_back();
        }
        update();
        return ret;
    }
    bool isEmpty() {
        return q1.size() == 0;
    }
    void update() {
        if (q1.size() < q2.size()) {
            q1.push_back(q2.front());
            q2.pop_front();
        }
        if (q1.size() == q2.size() + 2) {
            q2.push_front(q1.back());
            q1.pop_back();
        }
        return ;
    }
};

/**
 * Your FrontMiddleBackQueue object will be instantiated and called as such:
 * FrontMiddleBackQueue* obj = new FrontMiddleBackQueue();
 * obj->pushFront(val);
 * obj->pushMiddle(val);
 * obj->pushBack(val);
 * int param_4 = obj->popFront();
 * int param_5 = obj->popMiddle();
 * int param_6 = obj->popBack();
 */
```

Leetcode-933-最近的请求次数

```
class RecentCounter {
public:
    queue<int> q;
    RecentCounter() {}

    int ping(int t) {
        q.push(t);
        while (t - q.front() > 3000) q.pop();
        return q.size();
    }
};

/**
 * Your RecentCounter object will be instantiated and called as such:
 * RecentCounter* obj = new RecentCounter();
 * int param_1 = obj->ping(t);
 */
```

智力发散题

Leetcode-17.09-第 k 个数

```cpp
class Solution {
public:
    int getKthMagicNumber(int k) {
        vector<int> arr;
        arr.push_back(1);
        int p3 = 0, p5 = 0, p7 = 0;
        while (arr.size() < k) {
            int ans = 3 * arr[p3];
            ans = min(ans, 5 * arr[p5]);
            ans = min(ans, 7 * arr[p7]);
            if (3 * arr[p3] == ans) p3++;
            if (5 * arr[p5] == ans) p5++;
            if (7 * arr[p7] == ans) p7++;
            arr.push_back(ans);
        }
        return arr[k - 1];
    }
};
```

Leetcode-859-亲密字符串

```cpp
class Solution {
public:
    bool has_repeate(string a) {
        int cnt[26] = {0};
        for (int i = 0; a[i]; i++) {
            cnt[a[i] - 'a'] += 1;
            if (cnt[a[i] - 'a'] == 2) return true;
        }
        return false;
    }
    bool buddyStrings(string a, string b) {
        if (a.size() != b.size()) return false;
        if (a == b) return has_repeate(a);
        int i = 0, j;
        while (a[i] == b[i]) ++i;
        j = i + 1;
        while (j < a.size() && a[j] == b[j]) ++j;
        if (j == a.size()) return false;
        if (a[i] != b[j] || a[j] != b[i]) return false;
        j += 1;
        while (j < a.size()) {
            if (a[j] != b[j]) return false;
            j += 1;
        }
        return true;
    }
};
```

Leetcode-860-柠檬水找零

```cpp
class Solution {
public:
    bool lemonadeChange(vector<int>& bills) {
        int cnt5 = 0, cnt10 = 0;
        for (int i = 0; i < bills.size(); i++) {
            switch (bills[i]) {
                case 5: cnt5 += 1; break;
                case 10: {
                    if (cnt5 == 0) return false;
                    cnt5 -= 1; cnt10 += 1;
                } break;
                case 20: {
                    if (cnt10 && cnt5) {
                        cnt10 -= 1, cnt5 -= 1;
                    } else if (cnt5 >= 3) {
                        cnt5 -= 3;
                    } else {
                        return false;
                    }
                } break;
            }
        }
        return true;
    }
};
```

Leetcode-969-煎饼排序

```cpp
class Solution {
public:
    void reverse(vector<int> &arr, int n, vector<int> &ind) {
        for (int i = 0, j = n - 1; i < j; i++, j--) {
            swap(arr[i], arr[j]);
            ind[arr[i]] = i;
            ind[arr[j]] = j;
        }
        return ;
    }
    vector<int> pancakeSort(vector<int>& arr) {
        vector<int> ind(arr.size() + 1);
        vector<int> ret;
        for (int i = 0; i < arr.size(); i++) ind[arr[i]] = i;
        for (int i = arr.size(); i >= 1; i--) {
            if (ind[i] + 1 != 1) {
                ret.push_back(ind[i] + 1);
                reverse(arr, ind[i] + 1, ind);
            }
            if (i != 1) {
                ret.push_back(i);
                reverse(arr, i, ind);
            }
        }
        return ret;
    }
};
```

Leetcode-621-任务调度器

```
class Solution {
public:
    int leastInterval(vector<char>& tasks, int n) {
        int cnt[26] = {0};
        for (int i = 0; i < tasks.size(); i++) cnt[tasks[i] - 'A'] += 1;
        sort(cnt, cnt + 26);
        int m = 0;
        for (int i = 25; i >= 0 && cnt[i] == cnt[25]; i--, m++) ;
        return max((int)tasks.size(), (cnt[25] - 1) * (n + 1) + m);
    }
};
```