

# 【门徒计划】第二周刷题代码

## 【门徒计划】第二周刷题代码

### 链表复习题

Leetcode-86-分隔链表

Leetcode-138-复制带随机指针的链表

### 队列的封装与使用

Leetcode-622-设计循环队列

Leetcode-641-设计循环双端队列

Leetcode-1670-设计前中后队列

Leetcode-933-最近的请求次数

### 智力发散题

Leetcode-17.09-第 k 个数

Leetcode-859-亲密字符串

Leetcode-860-柠檬水找零

Leetcode-969-煎饼排序

Leetcode-621-任务调度器

## 链表复习题

### Leetcode-86-分隔链表

```
1  /**
2   * Definition for singly-linked list.
3   * struct ListNode {
4   *     int val;
5   *     ListNode *next;
6   *     ListNode() : val(0), next(nullptr) {}
7   *     ListNode(int x) : val(x), next(nullptr) {}
8   *     ListNode(int x, ListNode *next) : val(x), next(next) {}
9   * };
10  */
11  class Solution {
12  public:
13      ListNode* partition(ListNode* head, int x) {
14          ListNode r1, r2, *p1 = &r1, *p2 = &r2, *p = head, *q;
15          while (p) {
16              q = p->next;
17              if (p->val < x) {
18                  p->next = p1->next;
19                  p1->next = p;
20                  p1 = p;
21              } else {
22                  p->next = p2->next;
23                  p2->next = p;
24                  p2 = p;
25              }
26              p = q;
27          }
28          p1->next = r2.next;
29          return r1.next;
30      }
```

## Leetcode-138-复制带随机指针的链表

```
1  /*
2  // Definition for a Node.
3  class Node {
4  public:
5      int val;
6      Node* next;
7      Node* random;
8
9      Node(int _val) {
10         val = _val;
11         next = NULL;
12         random = NULL;
13     }
14 };
15 */
16
17 class Solution {
18 public:
19     Node* copyRandomList(Node* head) {
20         if (head == nullptr) return nullptr;
21         Node *p = head, *q, *new_head;
22         while (p) {
23             q = new Node(p->val);
24             q->random = p->random;
25             q->next = p->next;
26             p->next = q;
27             p = q->next;
28         }
29         p = head->next;
30         while (p) {
31             if (p->random) p->random = p->random->next;
32             (p = p->next) && (p = p->next);
33         }
34         new_head = head->next;
35         p = head;
36         while (p) {
37             q = p->next;
38             p->next = q->next;
39             if (p->next) q->next = p->next->next;
40             p = p->next;
41         }
42         return new_head;
43     }
44 };
```

## 队列的封装与使用

## Leetcode-622-设计循环队列

```
1  class MyCircularQueue {
2  public:
3      vector<int> arr;
4      int head, tail, cnt;
5      MyCircularQueue(int k) : arr(k), head(0), tail(0), cnt(0) {}
6
7      bool enqueue(int value) {
8          if (isFull()) return false;
9          arr[tail] = value;
10         tail = (tail + 1) % arr.size();
11         cnt += 1;
12         return true;
13     }
14
15     bool dequeue() {
16         if (isEmpty()) return false;
17         head = (head + 1) % arr.size();
18         cnt -= 1;
19         return true;
20     }
21
22     int Front() {
23         if (isEmpty()) return -1;
24         return arr[head];
25     }
26
27     int Rear() {
28         if (isEmpty()) return -1;
29         return arr[(tail - 1 + arr.size()) % arr.size()];
30     }
31
32     bool isEmpty() {
33         return cnt == 0;
34     }
35
36     bool isFull() {
37         return cnt == arr.size();
38     }
39 };
40
41 /**
42  * Your MyCircularQueue object will be instantiated and called as such:
43  * MyCircularQueue* obj = new MyCircularQueue(k);
44  * bool param_1 = obj->enqueue(value);
45  * bool param_2 = obj->dequeue();
46  * int param_3 = obj->Front();
47  * int param_4 = obj->Rear();
48  * bool param_5 = obj->isEmpty();
49  * bool param_6 = obj->isFull();
50  */
```

## Leetcode-641-设计循环双端队列

```
1 class MyCircularDeque {
2 public:
3     /** Initialize your data structure here. Set the size of the deque to be
4     k. */
5     vector<int> arr;
6     int cnt, head, tail;
7     MyCircularDeque(int k) : arr(k), head(0), tail(0), cnt(0) {}
8
9     /** Adds an item at the front of Deque. Return true if the operation is
10    successful. */
11    bool insertFront(int value) {
12        if (isFull()) return false;
13        head = head - 1;
14        if (head == -1) head = arr.size() - 1;
15        arr[head] = value;
16        cnt += 1;
17        return true;
18    }
19
20    /** Adds an item at the rear of Deque. Return true if the operation is
21    successful. */
22    bool insertLast(int value) {
23        if (isFull()) return false;
24        arr[tail] = value;
25        tail += 1;
26        if (tail == arr.size()) tail = 0;
27        cnt += 1;
28        return true;
29    }
30
31    /** Deletes an item from the front of Deque. Return true if the
32    operation is successful. */
33    bool deleteFront() {
34        if (isEmpty()) return false;
35        head = (head + 1) % arr.size();
36        cnt -= 1;
37        return true;
38    }
39
40    /** Deletes an item from the rear of Deque. Return true if the operation
41    is successful. */
42    bool deleteLast() {
43        if (isEmpty()) return false;
44        tail = (tail - 1 + arr.size()) % arr.size();
45        cnt -= 1;
46        return true;
47    }
48
49    /** Get the front item from the deque. */
50    int getFront() {
51        if (isEmpty()) return -1;
52        return arr[head];
53    }
54
55    /** Get the last item from the deque. */
```

```

51     int getRear() {
52         if (isEmpty()) return -1;
53         return arr[(tail - 1 + arr.size()) % arr.size()];
54     }
55
56     /** Checks whether the circular deque is empty or not. */
57     bool isEmpty() {
58         return cnt == 0;
59     }
60
61     /** Checks whether the circular deque is full or not. */
62     bool isFull() {
63         return cnt == arr.size();
64     }
65 };
66
67 /**
68  * Your MyCircularDeque object will be instantiated and called as such:
69  * MyCircularDeque* obj = new MyCircularDeque(k);
70  * bool param_1 = obj->insertFront(value);
71  * bool param_2 = obj->insertLast(value);
72  * bool param_3 = obj->deleteFront();
73  * bool param_4 = obj->deleteLast();
74  * int param_5 = obj->getFront();
75  * int param_6 = obj->getRear();
76  * bool param_7 = obj->isEmpty();
77  * bool param_8 = obj->isFull();
78  */

```

## Leetcode-1670-设计前中后队列

```

1  class Node {
2  public :
3      int val;
4      Node *next, *pre;
5      Node(int val = 0, Node *next = nullptr, Node *pre = nullptr) :
        val(val), next(next), pre(pre) {}
6      void insert_pre(Node *p) {
7          p->pre = pre;
8          p->next = this;
9          if (this->pre) this->pre->next = p;
10         this->pre = p;
11         return ;
12     }
13     void insert_next(Node *p) {
14         p->pre = this;
15         p->next = this->next;
16         if (this->next) this->next->pre = p;
17         this->next = p;
18         return ;
19     }
20     void delete_pre() {
21         if (this->pre == nullptr) return ;
22         Node *p = this->pre;
23         this->pre = p->pre;

```

```

24         if (p->pre) p->pre->next = this;
25         delete p;
26         return ;
27     }
28     void delete_next() {
29         if (this->next == nullptr) return ;
30         Node *p = this->next;
31         this->next = p->next;
32         if (p->next) p->next->pre = this;
33         delete p;
34         return ;
35     }
36 };
37
38 class Queue {
39 public :
40     Node head, tail;
41     int cnt;
42     Queue() : cnt(0) {
43         head.next = &tail;
44         head.pre = nullptr;
45         tail.next = nullptr;
46         tail.pre = &head;
47     }
48     void push_back(int val) {
49         tail.insert_pre(new Node(val));
50         cnt += 1;
51         return ;
52     }
53     void push_front(int val) {
54         head.insert_next(new Node(val));
55         cnt += 1;
56         return ;
57     }
58     int pop_back() {
59         if (isEmpty()) return -1;
60         int ret = tail.pre->val;
61         tail.delete_pre();
62         cnt -= 1;
63         return ret;
64     }
65     int pop_front() {
66         if (isEmpty()) return -1;
67         int ret = head.next->val;
68         head.delete_next();
69         cnt -= 1;
70         return ret;
71     }
72     int front() {
73         return head.next->val;
74     }
75     int back() {
76         return tail.pre->val;
77     }
78     bool isEmpty() {
79         return head.next == &tail;
80     }
81     int size() {

```

```
82         return cnt;
83     }
84 };
85
86 class FrontMiddleBackQueue {
87 public:
88     Queue q1, q2;
89     FrontMiddleBackQueue() {}
90
91     void pushFront(int val) {
92         q1.push_front(val);
93         update();
94         return ;
95     }
96
97     void pushMiddle(int val) {
98         if (q1.size() > q2.size()) {
99             q2.push_front(q1.back());
100             q1.pop_back();
101         }
102         q1.push_back(val);
103         return ;
104     }
105
106     void pushBack(int val) {
107         q2.push_back(val);
108         update();
109         return ;
110     }
111
112     int popFront() {
113         if (isEmpty()) return -1;
114         int ret = q1.pop_front();
115         update();
116         return ret;
117     }
118
119     int popMiddle() {
120         if (isEmpty()) return -1;
121         int ret = q1.pop_back();
122         update();
123         return ret;
124     }
125
126     int popBack() {
127         if (isEmpty()) return -1;
128         int ret;
129         if (q2.isEmpty()) {
130             ret = q1.pop_back();
131         } else {
132             ret = q2.pop_back();
133         }
134         update();
135         return ret;
136     }
137     bool isEmpty() {
138         return q1.size() == 0;
139     }
```

```

140     void update() {
141         if (q1.size() < q2.size()) {
142             q1.push_back(q2.front());
143             q2.pop_front();
144         }
145         if (q1.size() == q2.size() + 2) {
146             q2.push_front(q1.back());
147             q1.pop_back();
148         }
149         return ;
150     }
151 };
152
153 /**
154  * Your FrontMiddleBackQueue object will be instantiated and called as
such:
155  * FrontMiddleBackQueue* obj = new FrontMiddleBackQueue();
156  * obj->pushFront(val);
157  * obj->pushMiddle(val);
158  * obj->pushBack(val);
159  * int param_4 = obj->popFront();
160  * int param_5 = obj->popMiddle();
161  * int param_6 = obj->popBack();
162  */

```

## Leetcode-933-最近请求次数

```

1  class RecentCounter {
2  public:
3      queue<int> q;
4      RecentCounter() {}
5
6      int ping(int t) {
7          q.push(t);
8          while (t - q.front() > 3000) q.pop();
9          return q.size();
10     }
11 };
12
13 /**
14  * Your RecentCounter object will be instantiated and called as such:
15  * RecentCounter* obj = new RecentCounter();
16  * int param_1 = obj->ping(t);
17  */

```

## 智力发散题



## Leetcode-17.09-第 k 个数

```
1 class Solution {
2 public:
3     int getKthMagicNumber(int k) {
4         vector<int> arr;
5         arr.push_back(1);
6         int p3 = 0, p5 = 0, p7 = 0;
7         while (arr.size() < k) {
8             int ans = 3 * arr[p3];
9             ans = min(ans, 5 * arr[p5]);
10            ans = min(ans, 7 * arr[p7]);
11            if (3 * arr[p3] == ans) p3++;
12            if (5 * arr[p5] == ans) p5++;
13            if (7 * arr[p7] == ans) p7++;
14            arr.push_back(ans);
15        }
16        return arr[k - 1];
17    }
18};
```

## Leetcode-859-亲密字符串

```
1 class Solution {
2 public:
3     bool has_repeate(string a) {
4         int cnt[26] = {0};
5         for (int i = 0; i < a.size(); i++) {
6             cnt[a[i] - 'a']++;
7             if (cnt[a[i] - 'a'] == 2) return true;
8         }
9         return false;
10    }
11    bool buddyStrings(string a, string b) {
12        if (a.size() != b.size()) return false;
13        if (a == b) return has_repeate(a);
14        int i = 0, j = 0;
15        while (i < a.size() && a[i] == b[i]) i++;
16        j = i + 1;
17        while (j < a.size() && a[j] == b[j]) j++;
18        if (j == a.size()) return false;
19        if (a[i] != b[j] || a[j] != b[i]) return false;
20        j++;
21        while (j < a.size()) {
22            if (a[j] != b[j]) return false;
23            j++;
24        }
25        return true;
26    }
27};
```

## Leetcode-860-柠檬水找零

```
1 class Solution {
2 public:
3     bool lemonadeChange(vector<int>& bills) {
4         int cnt5 = 0, cnt10 = 0;
5         for (int i = 0; i < bills.size(); i++) {
6             switch (bills[i]) {
7                 case 5: cnt5 += 1; break;
8                 case 10: {
9                     if (cnt5 == 0) return false;
10                    cnt5 -= 1; cnt10 += 1;
11                } break;
12                case 20: {
13                    if (cnt10 && cnt5) {
14                        cnt10 -= 1, cnt5 -= 1;
15                    } else if (cnt5 >= 3) {
16                        cnt5 -= 3;
17                    } else {
18                        return false;
19                    }
20                } break;
21            }
22        }
23        return true;
24    }
25};
```

## Leetcode-969-煎饼排序

```
1 class Solution {
2 public:
3     void reverse(vector<int> &arr, int n, vector<int> &ind) {
4         for (int i = 0, j = n - 1; i < j; i++, j--) {
5             swap(arr[i], arr[j]);
6             ind[arr[i]] = i;
7             ind[arr[j]] = j;
8         }
9         return ;
10    }
11    vector<int> pancakeSort(vector<int>& arr) {
12        vector<int> ind(arr.size() + 1);
13        vector<int> ret;
14        for (int i = 0; i < arr.size(); i++) ind[arr[i]] = i;
15        for (int i = arr.size(); i >= 1; i--) {
16            if (ind[i] + 1 != 1) {
17                ret.push_back(ind[i] + 1);
18                reverse(arr, ind[i] + 1, ind);
19            }
20            if (i != 1) {
21                ret.push_back(i);
22                reverse(arr, i, ind);
23            }
24        }
25    }
```

```
25     return ret;
26 }
27 };
```

## Leetcode-621-任务调度器

```
1  class Solution {
2  public:
3      int leastInterval(vector<char>& tasks, int n) {
4          int cnt[26] = {0};
5          for (int i = 0; i < tasks.size(); i++) cnt[tasks[i] - 'A'] += 1;
6          sort(cnt, cnt + 26);
7          int m = 0;
8          for (int i = 25; i >= 0 && cnt[i] == cnt[25]; i--, m++) ;
9          return max((int)tasks.size(), (cnt[25] - 1) * (n + 1) + m);
10     }
11 };
```

