

【门徒计划】线程池与任务队列

【门徒计划】线程池与任务队列

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

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;
15          ListNode *p1=&r1, *p2=&r2;
16          ListNode *p=head, *q;
17
18          while(p){
19              q=p->next;
20              // big ? small
21              if(p->val < x){
22                  p->next = p1->next;
23                  p1->next = p;
24                  p1 = p;
```

```

25         } else {
26             p->next = p2->next;
27             p2->next = p;
28             p2 = p;
29         }
30         p=q;
31     }
32     p1->next = r2.next;
33     return r1.next;
34 }
35 };

```

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){
21             return nullptr;
22         }
23         Node *p = head;
24         Node *new_head;
25         while(p){
26             //copy node
27             Node *q = new Node(p->val);
28             q->random = p->random;
29             q->next = p->next;
30             p->next = q;
31             p = q->next;
32         }
33         // 回到头节点,修正

```

```

34     p = head;
35     while(p) {
36         // p1 -> q1 -> p2 -> q2
37         // q1 : p1->next
38         // q1.random = p1.random.next
39         if(p->random) p->next->random = p->random->next;
40         (p = p->next) && (p = p->next);
41     }
42     // 拆链表
43     new_head = head->next;
44     p = head;
45     while(p){
46         Node* q = p->next;
47         p->next = q->next;
48         if(p->next) q->next = p->next->next;
49         p=p->next;
50     }
51     return new_head;
52 }
53 };

```

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,count;
5
6      MyCircularQueue(int k):arr(k),head(0),tail(0),count(0) {}
7
8      bool enqueue(int value) {
9          if(isFull()){
10             return false;
11         }
12         arr[tail]=value;
13         tail++;
14         //if tail> arr.size()
15         tail = tail % arr.size();
16         count ++;
17         return true;
18     }
19
20     bool dequeue() {
21         if(isEmpty())return false;

```

```

22     head++;
23     head=head%arr.size();
24     count--;
25     return true;
26 }
27
28 int Front() {
29     if(isEmpty())return -1;
30     return arr[head];
31 }
32
33 int Rear() {
34     if(isEmpty())return -1;
35     int real_tail=(tail-1+arr.size()) % arr.size();
36     return arr[real_tail];
37 }
38
39 bool isEmpty() {
40     return (count==0);
41 }
42
43 bool isFull() {
44     //if(count == arr.size())return true;
45     //return false;
46     return count == arr.size();
47 }
48 };
49
50 /**
51  * Your MyCircularQueue object will be instantiated and called as such:
52  * MyCircularQueue* obj = new MyCircularQueue(k);
53  * bool param_1 = obj->enqueue(value);
54  * bool param_2 = obj->dequeue();
55  * int param_3 = obj->front();
56  * int param_4 = obj->rear();
57  * bool param_5 = obj->isEmpty();
58  * bool param_6 = obj->isFull();
59  */
60

```

Leetcode-641-设计循环双端队列

```

1 class MyCircularDeque {
2 public:
3     vector<int> arr;
4     int head,tail,count;
5     MyCircularDeque(int k):arr(k),head(0),tail(0),count(0) {}

```

```
6
7     bool insertFront(int value) {
8         if(isFull())return false;
9         head = (head -1 + arr.size()) % arr.size();
10        arr[head] = value;
11        count++;
12        return true;
13    }
14
15    bool insertLast(int value) {
16        if(isFull()){
17            return false;
18        }
19        arr[tail]=value;
20        tail = (tail+1) % arr.size();
21        count ++;
22        return true;
23    }
24
25    bool deleteFront() {
26        if(isEmpty())return false;
27        head=(head+1)%arr.size();
28        count--;
29        return true;
30    }
31
32    bool deleteLast() {
33        if(isEmpty())return false;
34        tail = (tail-1 + arr.size()) % arr.size();
35        count--;
36        return true;
37    }
38
39    int getFront() {
40        if(isEmpty())return -1;
41        return arr[head];
42    }
43
44    int getRear() {
45        if(isEmpty())return -1;
46        int real_tail=(tail-1+arr.size()) % arr.size();
47        return arr[real_tail];
48    }
49
50    bool isEmpty() {
51        return (count==0);
52    }
53
54    bool isFull() {
```

```

55         //if(count == arr.size())return true;
56         //return false;
57         return count == arr.size();
58     }
59 };
60
61 /**
62  * Your MyCircularDeque object will be instantiated and called as such:
63  * MyCircularDeque* obj = new MyCircularDeque(k);
64  * bool param_1 = obj->insertFront(value);
65  * bool param_2 = obj->insertLast(value);
66  * bool param_3 = obj->deleteFront();
67  * bool param_4 = obj->deleteLast();
68  * int param_5 = obj->getFront();
69  * int param_6 = obj->getRear();
70  * bool param_7 = obj->isEmpty();
71  * bool param_8 = obj->isFull();
72  */

```

Leetcode-1670-设计前中后队列

```

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

```



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

```
75     count--;
76     return ret;
77 }
78 int front(){
79     return head.next->val;
80 }
81 int back(){
82     return tail.pre->val;
83 }
84 bool isEmpty(){
85     return count==0;
86 }
87 int size(){
88     return count;
89 }
90 };
91
92 class FrontMiddleBackQueue {
93 public:
94     Queue q1,q2;
95 public:
96     FrontMiddleBackQueue() {}
97
98     void pushFront(int val) {
99         q1.push_front(val);
100         update();
101     }
102
103     void pushMiddle(int val) {
104         if (q1.size()>q2.size() ){
105             q2.push_front(q1.pop_back());
106         }
107         q1.push_back(val);
108         update();
109     }
110
111     void pushBack(int val) {
112         q2.push_back(val);
113         update();
114     }
115
116     int popFront() {
117         if(isEmpty())return -1;
118         int ret=q1.pop_front();
119         update();
120         return ret;
121     }
122
123     int popMiddle() {
```



```

124         if(isEmpty())return -1;
125         int ret=q1.pop_back();
126         update();
127         return ret;
128     }
129
130     int popBack() {
131         if(isEmpty())return -1;
132         int ret=-1;
133         if(q2.isEmpty()){
134             ret = q1.pop_back();
135         }else{
136             ret = q2.pop_back();
137         }
138         update();
139         return ret;
140     }
141
142     bool isEmpty(){
143         return ((q1.size()==0)&&(q2.size()==0));
144     }
145
146     void update(){
147         if(q1.size()<q2.size()){
148             q1.push_back(q2.pop_front());
149         }
150         if(q1.size()==q2.size()+2){
151             q2.push_front(q1.pop_back());
152         }
153         return;
154     }
155 };
156 /**
157  * Your FrontMiddleBackQueue object will be instantiated and called as such:
158  * FrontMiddleBackQueue* obj = new FrontMiddleBackQueue();
159  * obj->pushFront(val);
160  * obj->pushMiddle(val);
161  * obj->pushBack(val);
162  * int param_4 = obj->popFront();
163  * int param_5 = obj->popMiddle();
164  * int param_6 = obj->popBack();
165  */

```

Leetcode-933-最近的请求次数

```
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){
9              q.pop();
10         }
11         return q.size();
12     }
13 };
14
15 /**
16  * Your RecentCounter object will be instantiated and called as such:
17  * RecentCounter* obj = new RecentCounter();
18  * int param_1 = obj->ping(t);
19  */
```

智力发散题

Leetcode-17.09-第 k 个数

```
1  class Solution {
2  public:
3      vector<int> q;
4      int min(int a,int b ){
5          if(a>b)return b;
6          return a;
7      }
8      int getKthMagicNumber(int k) {
9          q.push_back(1);
10         int head3=0,head5=0,head7=0;
11         while(q.size()<k){
12             int min_ans = q[head3]*3;
13             min_ans = min(min_ans,q[head5]*5);
14             min_ans = min(min_ans,q[head7]*7);
15
16             if(min_ans==q[head3]*3)head3++;
17             if(min_ans==q[head5]*5)head5++;
18             if(min_ans==q[head7]*7)head7++;
19
20             q.push_back(min_ans);
11         }
```

```
21     }
22     return q[k-1];
23 }
24 };
```

Leetcode-859-亲密字符串

```
1  class Solution {
2  public:
3      bool has_repeat(string a,string b){
4          int count[26]={0};
5          for(int i=0;a[i];i++){
6              count[a[i]-'a']++;
7              if(count[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_repeat(a,b);
14         //找两处不一样的地方, 然后, 看是否只有两处, 且, 这两处是字母交换位置
15         int i,j;
16         i=0;
17         while(a[i]==b[i]){
18             i++;
19         }
20         //a[i] <> b[i]
21         j=i+1;
22         if(j>=a.size())return false;
23         while(j<a.size() && a[j]==b[j]){
24             j++;
25         }
26         //只有一处差异
27         if(j==a.size()) return false;
28         //发现两处不一致
29         //不能通过交叉换位获得相等
30         if(a[i]!=b[j]||a[j]!=b[i]){
31             return false;
32         }
33         j++;
34         //有三个以上的差异
35         while(j<a.size()){
36             if(a[j]!=b[j])return false;
37             j++;
38         }
39         return true;
40     }
```

```
41     }
42 };
```

Leetcode-860-柠檬水找零

```
1  class Solution {
2  public:
3      bool lemonadeChange(vector<int>& bills) {
4          int count5=0,count10=0;
5          bool ret=true;
6          for(int i=0;i<bills.size();i++){
7              switch(bills[i]){
8                  case 5:
9                      count5++;
10                     break;
11                 case 10:
12                     if(count5>0){
13                         count5--;
14                         count10++;
15                     }else{
16                         return false;
17                     }
18                     break;
19                 case 20:
20                     if(count10>0&&count5>0){
21                         count10--;
22                         count5--;
23                     }else if(count5>=3){
24                         count5-=3;
25                     }else{
26                         return false;
27                     }
28                     break;
29                 default:
30                     break;
31             }
32         }
33         return ret;
34     }
35 };
```

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      }
10     vector<int> pancakeSort(vector<int>& arr) {
11         vector<int> ind(arr.size()+1);
12         vector<int> ret;
13         for (int i=0;i<arr.size();i++){
14             ind[arr[i]]=i;
15         }
16         for (int i=arr.size();i>=1;i--){
17             if(ind[i]!=0){
18                 ret.push_back(ind[i]+1);
19                 reverse(arr, ind[i]+1, ind);
20             }
21             if(i!=1){
22                 ret.push_back(i);
23                 reverse(arr,i,ind);
24             }
25         }
26         return ret;
27     }
28 };
```