بسمه تعالی

گزارش تمرین ۵ سوالات تحلیلی

ساختمان داده

محسن کربلائی امینی، ۹۸۲۴۲۱۲۸

آذر ۱۴۰۲

سوال ۱:

با استفاده از یک صف که با استفاده از یک لینکدلیست دو طرفه ساخته شده می‌توان این کار را انجام داد. به این صورت که برای اضافه کردن هر عنصر، در لیست `insertFirst` کرده و سپس برای خواندن از لیست آخرین عنصر را خوانده و `next` عنصر قبلی را `null` میکنیم تا عنصر از صف حذف شود.
در نهایت هر جا که خواستیم لیست را خالی میکنیم که در این فانکشن میتوان تعیین کرد که تبدیل به باینری هم انجام گردد.

```
main() {
    Queue q۱ = new Queue();
    n = std_input();
    for (int i=۱ ; i <= n ; i++ ) {
        q۱.add(i);
        if (q۱.isFull == ۱)
            q۱.emptyQueue();
    }
    q۱.emptyQueue();
}
```

سوال ۲:

الف) برای این کار می‌توان به شکل زیر عمل کرد:
دو صف را در نظر میگیریم. در هر بار اضافه کردن، ابتدا عنصر به صف خالی اضافه می‌شود. سپس محتویات صف پر به این صف خالی اضافه می‌شود. بعد از این، صف اول مجدد خالی شده و محتویات آن درون صف اول ریخته می‌شود. به این ترتیب با `remove` از صف پر، آخرین عنصری که به مجموعه اضافه شده، خارج می‌شود.

```
class Stack {
    Queue<Integer> queue۱, queue۲;
    public void push(int value) {
        queue۲.add(value);
        while (!queue۱.isEmpty()) {
            queue۲.add(queue۱.remove());
        }
        queue۱ = queue۲;
        queue۲ = new LinkedList<>();
    }
    public int pop() {
       return queue۱.remove();
    }
}
```

ب) با استفاده از دو پشته می‌توان یک صف را پیاده‌سازی کرد. در این روش فقط در یکی از پشته‌ها عمل push را انجام می‌دهیم. سپس تمامی دیتاهای موجود را pop کرده و به همان ترتیب pop کردن داخل پشته دیگر push میکنیم. به این ترتیب در صورت pop کردن از پشته دوم، First In خارج می‌شود.

```
class Queue {
    private Stack<Integer> stack۱;
    private Stack<Integer> stack۲;
    public void enqueue(int value) {
        stack۱.push(value);
    }
    public int dequeue() {
        if (isEmpty()) {
            throw new IndexOutOfBoundsException("Queue is empty");
        }
        if (stack۲.isEmpty()) {
            while (!stack۱.isEmpty()) {
                stack۲.push(stack۱.pop());
            }
        }
        return stack۲.pop();
    }
}
```

ج)

سوال ۳:

برای حل این مساله یک ساختمان داده جدید با الگو گیری از ساختمان داده لینکدلیست ارائه داده‌ایم. شبه کد زیر نقشه این ساختمان داده را مشخص می‌کند:

```
class Node {
    int data;
    OpNode next;
}
class OpNode {
    char operator;
    Node   next;
    char[] priorities;
}
class Operation {
    Node n۱,n۲;
    OpNode operator;
    Operation(Node n۱,Node n۲, OpNode operator) {
```

```
        this.n1=n1;
        this.n2=n2;
        this.operator=operator;
    }
    public static Node calculate(Operation op) {
        Node result = new Node();
        // do calculations
        return result;
    }
}
class calculatingLinkedList {
    Operation head;
    Node result;
    calculatingLinkedList(Operation head) {
        this.head = head;
    }
    public void insertFirst(Operation newOp) {
        // add ops here
    }
    public Node calculate() {
        Node result;
        // if there are no head.operators return head.n1
        // store operators from each operation,
        // calculate the results of that operation,
        // connect the before and after operations to the results of this
operation
        // if there are no other operations to be connected, just have n1
    }
}
```

سوال ۴:

برای بردن قایق نجات، از ساختمان داده لینکد لیست دایره‌ای دوطرفه استفاده می‌کنیم. درون هر گره مقدار index جایگاه اولیه فرد ذخیره می‌شود. در این ساختمان داده تابعی با عنوان حذف گره خواهیم داشت که بعد از حذف یک گره، گره قبلی خود را به گره بعدی گره حذف شده، متصل می‌کند. در عمل حذف، لازم است در صورتی که گرهی که شروع پیمایش از آن است، حذف می‌شود، اشاره‌گر شروع پیمایش به گره بعدی (از روی جهت)، منتقل شود.

به این ترتیب، پس از مشخص شدن داده‌های اولیه مسئله، در این لینکدلیست از جهت مورد نظر و به تعداد k شروع به پیمایش و حذف گره‌ها میکنیم. این عمل تا جایی که یک گره در لیست باقی بماند ادامه پیدا می‌کند و در نهایت گرهی که باقی مانده باشد، جایگاه برنده را در خود ذخیره کرده است.