8월 스터디 2주차 3조 팀 발표자료

[스택]

17608 막대기

```java
public class Boj_17608 {
    public static void main(String[] args) throws NumberFormatException, IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        Stack<Integer> st = new Stack<>();
        int N = Integer.parseInt(br.readLine());

        for(int i=0; i<N; i++) {
            int num = Integer.parseInt(br.readLine());
            //스택이 비어있지 않고, 판별했을 때 들어갈 막대기보다 들어있는 막대기의 크기가 작으면 뺀다.
            while(!st.isEmpty()&&isVisible(st, num)) {
                st.pop();
            }
            //뺀 뒤에 다시 막대기 집어넣기(집어넣는 막대기보다 큰 막대기들만 스택에 남아있음)
            st.push(num);
        }
        System.out.println(st.size());
    }

    //지금 넣는 막대기보다 스택에 있는 막대기가 작으면 True 반환, 아니면 False
    static boolean isVisible(Stack st, int num) {
        int top = (int) st.peek();
        return num>=top;

    }
}
```

10799 쇠막대기

1. 스택에 인덱스를 입력한 경우

```java
public class IronStick_10799 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

            int cnt = 0;//잘려나간 스틱

            String str = sc.next();//괄호(레이저와 막대기) 입력

            Stack<Integer> stack = new Stack<>();

            for(int i=0;i<str.length();i++) {
                if(str.charAt(i)=='(')
                    stack.push(i);
                else {
                    if(stack.peek()==i-1) {
                        stack.pop();
                        cnt+=stack.size();
                    }
                    else {
                        stack.pop();
                        cnt++;
                    }
                }
            }
            System.out.println(cnt);

    }
}
```

2. 스택에 인덱스와 괄호를 갖는 Node 클래스를 삽입한 경우

```java
public class BOJ_10799_쇠막대기 {

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String input = br.readLine();           // 입력
        List<Node> stack = new ArrayList<>();   // 입력의 괄호를 하나씩 넣을 스택
        int sum = 0;    // 자른 쇠막대기 갯수

        for(int i=0;i<input.length();i++) {
            char ch = input.charAt(i);
            if(ch == '(') {
                stack.add(new Node(i, ch));
            }
            else{   // ch == ')'
                int lastIdx = stack.size()-1;   // 레이저로 잘린 왼쪽 막대기 수
                Node peek = stack.get(lastIdx);
                if(peek.c == '(') { // 쌍이 맞는지 검사하기 위해
                    Node pop = stack.remove(lastIdx);   // 무조건 '(' 와 해당 문자의 위치
                    /* '()((()))' 의 경우처럼 맨처음은 아무것도 자르지 않는 경우 때문에 */
                    if(stack.isEmpty()) {
                        // 쇠막대기 닫는 괄호와 레이저 닫는 괄호를 구분하기 위해서.
                        if(pop.idx != i-1)  sum++;
                        continue;
                    }
                    if(pop.c == '(' && pop.idx == i-1)  // 레이저의 닫는 괄호 만난 경우.
                        sum += lastIdx;
                    else    // 쇠막대기 닫는 괄호 만난 경우.
                        sum++;
                }
            }
        }
        sum += stack.size();    // 스택에 남은 것들을 다 더해줌.
        System.out.println(sum);

    }

    static class Node{
        int idx;
        char c;

        public Node(int idx, char c) {
            super();
            this.idx = idx;
            this.c = c;
        }
    }
}
```

3. 스택에 괄호를 삽입한 경우

```java
public class Q10799 {//쇠막대기

    public static void main(String[] args) throws IOException {
        BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
        String s = bf.readLine();
        Stack<Character> stack = new Stack<>();
        int sum = 0;
        for (int i = 0; i < s.length(); i++) {
            if (i != s.length()- 1) {
                if (s.charAt(i) == '(' && s.charAt(i + 1) == ')') {//레이저이면
                    sum += stack.size(); //레이저로 인해 스택에 있는 막대기수만큼 조각 발생
                    i++; //다음 인자가 ')'임을 아니까 하나 넘어감
                    continue;
                } else {
                    if (s.charAt(i) == '(') { //여는 괄호이면
                        stack.add('('); //스택에 삽입
                    } else { //레이저가 아니고 닫는 기호이면
                        stack.pop();//스택에서'(' 제거
                        sum += 1; // 전체 개수에 막대기 하나 추가
                    }
                }
            } else { // 끝단
                if (s.charAt(i-1) == '(') //레이저이면 추가안함
                    continue;
                else sum+=1; //닫는 기호이면 막대기 하나 추가

            }
        }
        System.out.println(sum);

    }

}
```

2304 창고 다각형

1. 스택을 이용한 경우

```java
public class BOJ_2304_창고다각형 {
    public static void main(String[] args) throws NumberFormatException, IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int N = Integer.parseInt(br.readLine());    // 막대 기둥의 수
        List<Node> list = new ArrayList<Node>();    // 막대의 입력과 x기준으로 정렬하기 위해 만든 리스트
        Stack<Node> stack = new Stack<>();          // 스택
        int Area = 0;   // 총 창고 넓이(출력값)

        // 막대 입력
        for(int i=0;i<N;i++) {
            StringTokenizer st = new StringTokenizer(br.readLine()," ");
            int x = Integer.parseInt(st.nextToken());
            int h = Integer.parseInt(st.nextToken());
            list.add(new Node(x,h));
        }

        Collections.sort(list); // 정렬
        /* 입력 받고 x를 기준으로 정렬까지 완료.*/

        /**로직 시작*/
        // 먼저 스택에 넣고 빼면서 창고 넓이를 일부 구함.
        for (Node cur : list) {
            if(stack.isEmpty()) // 비어 있다면
                stack.push(cur);
            else {              // 스택이 비어 있지 않다면
                Node pop = null;
                while(!stack.isEmpty()) {
                    if(stack.peek().height < cur.height)    // 더 큰 높이가 들어온다면
                        pop = stack.pop();
                    else    // 스택에 있는것보다 더 낮은 크기가 들어온다면 반복문을 나가서 그냥 push만 해줌.
                        break;
                }
                if(stack.isEmpty()) {
                    int localArea = pop.height * (cur.x - pop.x);
                    Area += localArea;
                }
                stack.push(cur);
            }
        }
```

```java
        // 스택에 남아있는 것들을 이용해 남은 창고 넓이를 구함. => 최대 높이 이후의 부분.
        while(stack.size() > 1) {
            Node pop = stack.pop();
            int localArea = pop.height * (pop.x - stack.peek().x);
            Area += localArea;
        }

        // 가장 큰 부분이 남기 때문에 마지막으로 더해줌.
        Area += stack.pop().height;

        // 결과 출력
        System.out.println(Area);
    }
}

class Node implements Comparable<Node>{
    int x;
    int height;

    public Node(int x, int height) {
        super();
        this.x = x;
        this.height = height;
    }

    @Override
    public int compareTo(Node other) {
        // TODO Auto-generated method stub
        if(this.x < other.x)
            return -1;
        else if(this.x > other.x)
            return 1;
        return 0;
    }
}
```

2. 리스트를 이용한 경우

```
public class Q2304 { //창고 다각형

    static class Pillar implements Comparable<Pillar> { //기둥 클래스
        int x; //기둥의 x값
        int height; //기둥의 높이

        public Pillar(int index, int height) {
            super();
            this.x = index;
            this.height = height;
        }

        @Override
        public String toString() { return "Pillar [index=" + x + ", height=" + height + "]"; }

        @Override
        public int compareTo(Pillar o) { //x값 기준 오름차순 정렬
            if (this.x > o.x)
                return 1;
            return -1;
        }

    }

    public static void main(String[] args) throws IOException {

        BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
        int n = Integer.parseInt(bf.readLine());
        List<Pillar> pillars = new ArrayList<>(); //필러의 값들 저장
        int highest = 0; //가장 높은 기둥의 높이
        int highx = 0; //가장 높은 기둥의 x값
        int highIndex=0; //가장 높은 기둥의 순서
        for (int i = 0; i < n; i++) {
            String s = bf.readLine();
            StringTokenizer st = new StringTokenizer(s);
            int x = Integer.parseInt(st.nextToken());
            int height = Integer.parseInt(st.nextToken());
            pillars.add(new Pillar(x, height));
            if (highest <= height) { //최대 높이보다 같거나 높은거 발견하면 갱신
                highest = height;
                highx = x;

            }

        }
```

```
        }
        Collections.sort(pillars); //x값 기준 정렬

        int befHigh = pillars.get(0).height; //이전기둥의 높이
        int befx = pillars.get(0).x; //이전 기둥의 x값
        int sum = 0; //넓이



        for (int i = 1; i <n; i++) {
            if(pillars.get(i).x>highx) { //최댓값을 지나가면
                highIndex=i-1; //바로 전값이 최댓값이므로 저장하고 탈출
                break;
            }
            if (befHigh <= pillars.get(i).height) { // 이전값보다 높이가 크면
                sum += (pillars.get(i).x - befx) * befHigh; //넓이 더해줌
                befHigh = pillars.get(i).height;
                befx= pillars.get(i).x;


            }

            else continue; //이전 값보다 작은 기둥은 무시해도 됨

        }
        sum+=highest; //가장 높은 기둥의 높이만큼 넓이 더해줌
        befx= pillars.get(n-1).x; //이제 반대로 진행
        befHigh = pillars.get(n-1).height;

        for(int i=n-2;i>=highIndex;i--) {


            if (befHigh <= pillars.get(i).height) {
                sum += (befx-pillars.get(i).x) * befHigh;
                befHigh = pillars.get(i).height;
                befx= pillars.get(i).x;


            }

            else continue;
        }
        System.out.println(sum);


    }

}
```

[큐]

15828 라우터 ->모두 유사하게 구현

```java
public class Router_15828 {
    public static void main(String[] args) throws NumberFormatException, IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        Queue<Integer> que = new LinkedList<>();
        int N = Integer.parseInt(br.readLine());

        while(true) {
            int p = Integer.parseInt(br.readLine());
            if(p==0) {
                que.remove();
                continue;
            }// 0
            if(p==-1) {
                if(que.isEmpty()) System.out.println("empty");
                else {
                    int tmp = que.size();
                    for(int i=0;i<tmp;i++) {
                        System.out.print(que.poll()+" ");
                    }
                }
                break;
            }// -1
            if(que.size()==N) {
                continue;
            }// full
            else{
                que.add(p);
            }
        }
    }
}
```

```java
public static void main(String[] args) throws  IOException {//라우터
    // TODO Auto-generated method stub

    BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
    int size = Integer.parseInt(bf.readLine());
    Queue <Integer> que = new LinkedList<>();
    int i=0;
    while(true) {
        i=Integer.parseInt(bf.readLine());
        if(i==-1) break; //-1입력하면 종료
        if(i!=0) { //0이 아닐 때
            if(que.size()<size) //버퍼 크기를 넘지 않았으므로
            que.add(i);//값 추가
            }
        else que.poll(); //0ㅇ일때 값 빼줌
    }
    boolean isEmpty=true;
    while(!que.isEmpty()) {//비어있지 않으면
        System.out.print(que.poll()+" "); //버퍼 내용 출력
        isEmpty=false;
    }
    if(isEmpty) System.out.println("empty"); //비어있다고 출력
}
```

2161 카드1

카드수가 짝수인지 홀수인지를 통해 구분하여 큐에 삽입, 삭제 진행

```java
//백준 2161 카드1
public class BOJ_2161 {
    static int temp;
    static int idx=1;
    public static void main(String[] args) throws NumberFormatException, IOException {
//      Scanner sc = new Scanner(System.in);
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        Queue<Integer> card = new LinkedList<Integer>();
        int N = Integer.parseInt(br.readLine());
        for(int i=1; i<=N; i++) {
            card.add(i);
        }

        while(card.size()!=1) {
            temp=0;
            if(idx%2==1 && !card.isEmpty()) {
                temp = card.poll();
                System.out.print(temp+" ");
                idx= idx+1;
            }
            else if(idx%2==0 && !card.isEmpty()) {
                temp = card.poll();
                card.add(temp);
                idx= idx+1;
            }

        }
        System.out.print(card.peek());


    }
}
```

3190 뱀

덱을 사용하여 뱀이 위치한 좌표 삽입, boolean으로 현재 위치 여부 체크. 이동한 위치에서 사과가 없으면 덱에서 꼬리 삭제. 이동할 때마다 머리 삽입. 재귀를 통해 끝날 때까지 반복.

```
public class Q3190 { //뱀
    static class Pos{ //뱀의 위치 저장
        int y;
        int x;
        Pos(int y,int x){
            this.y=y;
            this.x=x;
        }
    }
    static boolean [][] isPos;//뱀위치
    static int n,time;
    static int [][]arr; //사과 위치
    static char [] commands;//명령어들 모임
    static Deque<Pos> snake;
    public static void main(String[] args) throws  IOException {

        BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
        n= Integer.parseInt(bf.readLine());
        time = Integer.parseInt(bf.readLine()); //사과 입력
        arr = new int[n+1][n+1];
        isPos =new boolean[n+1][n+1];
        commands = new char[100001];
        snake =new  LinkedList<>();
        for(int i=0;i<time;i++) {
            String s = bf.readLine();
            StringTokenizer st = new StringTokenizer(s);
            int y= Integer.parseInt(st.nextToken());
            int x = Integer.parseInt(st.nextToken());
            arr[y][x]=1;    //배열에 사과 위치에 1로 저장
        }
        int moveTime = Integer.parseInt(bf.readLine()); //이동방향 입력 횟수
        for(int i=0;i<moveTime;i++) {
            String s =bf.readLine();
            StringTokenizer st = new StringTokenizer(s);
            int second = Integer.parseInt(st.nextToken()); //시간을 담을 배열에 해당 초가 되에
            char c = st.nextToken().charAt(0); //명령어 입력
            commands[second]=c;
        }
        isPos[1][1]=true; //1,1에 현재 뱀이 있으므로 true
        snake.add(new Pos(1,1));//처음 시작 위치 1,1를 뱀의 위치를 저장하는 deque에 저장
        move(0,1,1,1);


    }
```

```java
    private static void move(int curTime, int direct, int y, int x) {//현재 진행한 초,방향,y,x

        if(commands[curTime]=='D') //해당 초에 명령어 'D'가 있으면 오른쪽으로 90도 회전
            direct+=1;
        else if(commands[curTime]=='L') //해당 초에 명령어 'L'이 있으면 왼쪽으로 90도 회전
            direct-=1;
        if(direct==0) direct = 4;
        else if(direct==5) direct=1;

        int ny=0,nx=0;
        if(direct==1) { // ->
            ny=y; nx=x+1;
        }
        else if(direct==2) {//하
            ny=y+1; nx=x;
        }
        else if(direct==3) { //<-
            ny=y; nx=x-1;
        }
        else if(direct==4) {//상
            ny=y-1; nx=x;
        }

        if(!isIn(ny,nx)) { //배열의 크기 벗어나면 중단하고 시간 출력
            System.out.println(curTime+1); //0으로 시작해서 +1해줌

            return;
        }
        if(isPos[ny][nx]) { //자기 몸에 부딛히면 중단하고 시간 출력
            System.out.println(curTime+1);
            return;
        }
        snake.addFirst(new Pos(ny,nx)); //이동 가능한 범위이면 deq에 해당 위치를 머리에 넣어둠
        if(arr[ny][nx]!=1) { //이동위치에 사과가 있지않으면

            Pos tail =snake.pollLast();//제일 끝인 꼬리를 빼줌
            isPos[tail.y][tail.x] =false;
        }
        isPos[ny][nx]=true; //머리를 넣어줌
        arr[ny][nx]=0;//사과를 먹었을 수 있으니 0으로 바꿔줌
        move(curTime+1,direct,ny,nx); //다음초 호출

    }
    private static boolean isIn(int y, int x) {//배열을 벗어나지 않는지 확인
        if(y>n || x>n || x<=0 || y<=0)
        return false;
        return true;
    }

}
```