문제14 - 표_편집(포기)

태그	
생성 일시	@2024년 8월 15일 오전 12:13

접근 방식

처음에는 해쉬맵을 설정해서 행 번호와 value쌍으로 만들어서 조작해보려고 하였으나, 문제 내에 이름 열을 필요로 하지 않는다는 조건이 붙어서 행 번호만 따로 처리해야겠다고 생각하였음

행 번호만을 따로 저장해놓고 선택된 위치를 가리키는 변수를 만들어서 배열의 각 원소를 조정하면 가능할 것이라고 생각하였음.

자료구조

배열과 리스트 중에서 고민하였음. 인덱스가 삭제될 때 줄어들기 때문에 리스트를 사용하면 좋을 것이라고 생각하였음. 리스트에 행 번호를 저장하고 앞 뒤로 없어져야 계산이 편하다고 생각하였으나 리스트의 크기가 달라지면 마지막에 행 번호를 비교하기가 힘들 것이라 생각해서 배열로 문제를 풀어보려고 생각함.

배열을 0부터 n-1까지의 크기로 선언하고 C라는 문자를 발견할 때마다 배열에 있는 원소를 스택에다가 넣고 기존의 배열에 -1을 대입해줌. Z문자를 만나면 스택에서 빼가지고 배열에다가 다시 넣어줌.

배열로 풀어보다가 C가 연속으로 만났을 때 제거를 두 번 해야하는데 이러면 포인터를 하나씩 감소시켜주어야하는데, 그러면 U, D할 때 카운트를 제대로 맞추기가 힘든 상황이 발생하였음. 따라서 리스트를 통해서 인덱스를 감소시키고 인덱스와 배열에 담긴 값들을 가지고 리스트에 대입시켰으나 포인터를 조작시키는 과정에서 어려움이 발생하였다가 시간복잡도 측면에서는 괜찮았지만, 효율성 테스트에서 탈락함.

결국 AI통해서 답을 내었음.

내 코드

```java
public static String solution(int n, int k, String[] cmd) {
        int choice = k;
        int[] index = new int[n];
        List<Integer> list = new ArrayList<>();
        StringBuilder answer = new StringBuilder();
        ArrayDeque<Fair> stack = new ArrayDeque<>();

        for (int i = 0; i < n; i++) {
            list.add(i);
        }

        for (int i = 0; i < cmd.length; i++) {
            char action = cmd[i].charAt(0);
            if (action == 'U') {
                int upNum = Integer.parseInt(cmd[i].split(" "
                choice -= upNum;
            } else if (action == 'D') {
                int downNum = Integer.parseInt(cmd[i].split("
                choice += downNum;
            } else if (action == 'C') {
                stack.push(new Fair(choice, list.get(choice))
                list.remove(choice);
                if (choice >= list.size()) {
                    choice = list.size() - 1;
                }
            } else if (action == 'Z') {
                Fair x = stack.pop();
                list.add(x.index, x.value);
                if (x.index <= choice) {
                    choice++;
                }

            }
        }

        for (int i = 0; i < n; i++) {
            if (list.contains(i)) {
                answer.append('O');
```

```
            } else {
                answer.append('X');
            }
        }

        return answer.toString();
    }
```

AI 코드

```
public static String solution2(int n, int k, String[] cmd) {
        int[] prev = new int[n];
        int[] next = new int[n];
        for (int i = 0; i < n; i++) {
            prev[i] = i - 1;
            next[i] = i + 1;
        }
        next[n - 1] = -1;

        Stack<Integer> removed = new Stack<>();
        int current = k;

        for (String command : cmd) {
            char action = command.charAt(0);
            if (action == 'U' || action == 'D') {
                int x = Integer.parseInt(command.substring(2)
                if (action == 'U') {
                    while (x-- > 0) {
                        current = prev[current];
                    }
                } else {
                    while (x-- > 0) {
                        current = next[current];
                    }
                }
            } else if (action == 'C') {
```

```
                System.out.println(Arrays.toString(next));
                System.out.println(Arrays.toString(prev));
                System.out.println();

                removed.push(current);
                System.out.println(current);

                if (prev[current] != -1) {
                    next[prev[current]] = next[current];
                }

                if (next[current] != -1) {
                    prev[next[current]] = prev[current];
                }

                current = next[current] != -1 ? next[current]

            } else if (action == 'Z') {
                int restore = removed.pop();
                if (prev[restore] != -1) {
                    next[prev[restore]] = restore;
                }
                if (next[restore] != -1) {
                    prev[next[restore]] = restore;
                }
            }
        }

        char[] result = new char[n];
        Arrays.fill(result, 'X');
        for (int i = 0; i < n; i++) {
            if (prev[i] != -1 || next[i] != -1) {
                result[i] = 'O';
            }
        }
        for (int r : removed) {
            result[r] = 'X';
        }
```

```
        return new String(result);
    }
```

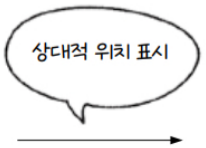
해설

배열을 그대로 사용하는 아이디어를 생각할 수 있다. 하지만 이 방법은 삽입과 삭제가 빈번한 상황에서는 매우 비효율적이다. 따라서 문제에서 반환하는 정보는 cmd가 수행되었을 때 처음과 비교해서 각 행이 삭제되었는지의 여부이다. 여기서 사용하는 방법은 실제 배열을 선언하고 삽입과 삭제 대신, 인덱스만으로 연산하는 것이다.

예를들어서 cmd 명령어를 수행할 때마다 해당 행의 위와 아래에 위치한 행의 번호를 계속 업데이트 하는 것이다. 즉, 실제 배열을 삽입 삭제하는 대신 인접한 행의 정보를 활용하는 것이다.

up이나 down은 각 행을 기준으로 연산이 수행된 후의 위치를 표시하는 것이다.

행 번호	이름
0	무지
1	콘
2	어피치
3	제이지



up	down
-1	1
0	2
1	3
2	4

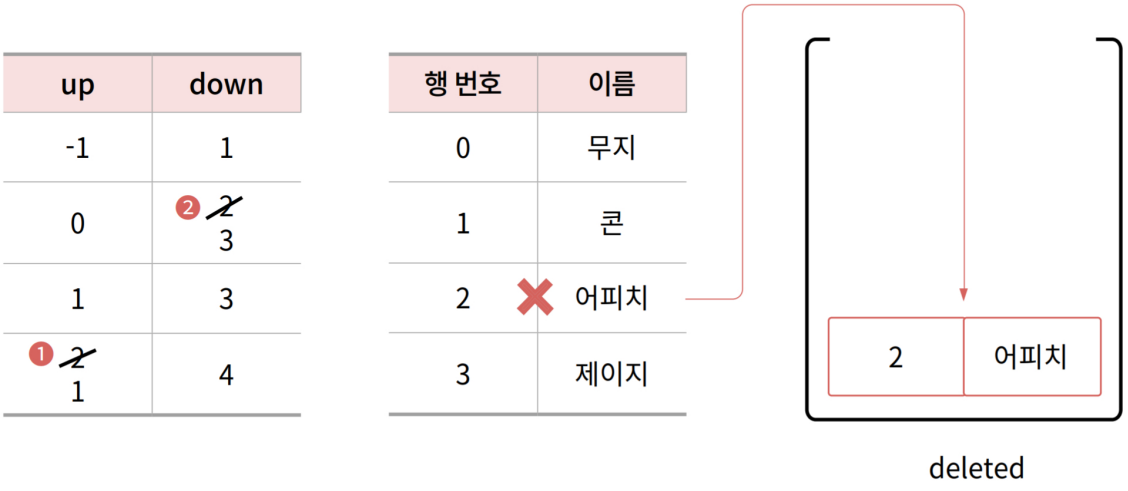
이렇게 하면 실제로 배열 연산을 하지 않으면서 행 삭제나 추가등을 표시할 수 있다. 무지의 위치는 0이기 때문에 앞선 수를 -1로 표현하였고 아래는 1이라고 표현할 수 있다.

콘의 경우는 위에가 0 아래가 2라는 것을 나타낸다.

C를 만났을 때 삭제해야하는데 다음을 고려해야한다.

1. 삭제된 행을 저장하는 방법

2. 마지막 행이 삭제되었을 때, 바로 위 행을 선택하는 방법
3. 배열을 삭제하지 않고도 인덱스를 활용해서 삭제가 된 것처럼 만드는 방법



만약 2번 어피치가 삭제된다고 가정하였을 때 현재위치를 k 라고 하였을때 다음과 같이 변경되어야한다.

1. k 아래에 있는 행의 윗부분은, k의 윗부분이 되어야한다. 즉, 제이지의 위는 콘이 되어야한다.
    a. up[down[k]] = up[k];
2. k의 윗부분에 있는 행의 아랫부분은, k의 아랫부분이 되어야한다. 즉, 콘의 아래는 제이지가 되어야한다.
    a. down[up[k]] = down[k];

- 3인 제이지에서 위로 올라갔을 때 원래 2였던 부분에서 2가 없어졌기 때문에 1로 변경되어야한다.
- 1인 콘에서 아래로 내려날 때 원래 2였던 부분에서 2가 없어졌기 때문에 3으로 변경되어야한다.

이후 스택을 만들어서 삭제 행을 푸시함.

Z를 만났을 때 복구 동작의 핵심은 기존 삭제 위치에 행을 삽입해야 한다는 것이다.

1. 스택에서 최근에 삭제된 행을 pop하고 삭제한 원소를 restore에 보관
2. 행 번호 restore을 기준으로 윗 행의 다음과 아래 행의 이전은 restore가 되어야 함. 즉, down[up[restore]]와 up[down[restore]]가 restore여야 함.

테이블 양 끝에서도 추가 삭제가 가능하기 때문에 양 끝에서도 적용할 수 있어야하지만, 맨 위에는 테이블 상에는 없었다. 따라서 양 끝에 명령어를 수행할 때도 정상적으로 작동할 수 있도록 크기가 큰 배열을 만들어 관리하면 된다.

```
public String solution3(int n, int k, String[] cmd) {
        //삭제된 행의 인덱스를 저장하는 스택
        Stack<Integer> deleted = new Stack<>();

        //각 행을 기준으로 연산에 따른 위치를 표시하기 위한 배열
        int[] up = new int[n + 2];
        int[] down = new int[n + 2];

        for (int i = 0; i < n + 2; i++) {
            up[i] = i - 1;
            down[i] = i + 1;
        }

        //현재 위치를 나타내는 인덱스
        k++;

        //주어진 명령어를 하나씩 처리
        for (String c : cmd) {
            //현재 위치를 삭제하고 그 다음 위치로 이동
            if (c.startsWith("C")) {
                deleted.push(k);
                up[down[k]] = up[k];
                down[up[k]] = down[k];

                if (n < down[k]) {
                    k = up[k];
                } else {
                    k = down[k];
                }
```

```
            }
            //가장 최근에 삭제한 행을 복원
            else if (c.startsWith("Z")) {
                int restore = deleted.pop();
                down[up[restore]] = restore;
                up[down[restore]] = restore;
            } else {
                String[] s = c.split(" ");
                int x = Integer.parseInt(s[1]);

                for (int i = 0; i < x; i++) {
                    k = s[0].equals("U") ? up[k] : down[k];
                }
            }
        }

        //삭제한 행의 위치에 'X'를, 그렇지 않은 행 위치에는 'O' 저장
        char[] answer = new char[n];
        Arrays.fill(answer, 'O');

        for (int i : deleted) {
            answer[i - 1] = 'X';
        }
        return new String(answer);
    }
```