# 스터디 발표

| | |
|---|---|
| 날짜 | @2021년 8월 20일 오후 7:00 (KST) |
| 태그 | #2조 #3주차 #달팽이리스트 #스터디 #에디터 #연결리스트 #팀발표 |

# [8월 3주차 스터디]

## 1. 백준 1406번 에디터

▼ 문제 링크

> 1406번: 에디터
> 한 줄로 된 간단한 에디터를 구현하려고 한다. 이 편집기는 영어 소문자만을 기록할 수 있는 편집기로, 최대 600,000글자까지 입력할 수 있다. 이 편집기에는 '커서'라는 것이 있는데, 커서는 문장의 맨 앞(첫 번째 문자의 왼쪽), 문장의 맨 뒤(마지막 문자의 오른쪽), 또는 문장 중간 임의의 곳(모든 연속된 두 문자 사이)에 위치할 수 있다.
> https://www.acmicpc.net/problem/1406

첫번째 시도 (with LinkedList, System.out.print())

❌ ArrayList와 달리 이전과 다음 요소의 정보를 활용하여 삽입 / 삭제 를 처리할 수 있기에 LinkedList를 이용해서 구현하고자 했다. cursor의 위치를 문장의 맨 뒤로 초기화 하고 각 연산을 진행할 때마다 조건에 맞춰서 해당 cursor의 위치를 바꿔주도록 구현했다. L, D, B, P 명령어 수행시 LinkedList가 저장된 cursor값을 활용해서 해당하는 인덱스를 찾고 삽입 / 삭제가 처리되는 방식으로 구현했다. (문제에서 주어진 명령어의 최대 개수는 50만개라는 걸 고려하지 않은 설계, 최악의 경우 문자열 길이 10만에 대해서 50만의 연산을 수행할 수 도 있다. 50만 * 10만 = 500억 대략 500초 라고 본다면 당연히 시간초과가 발생!)

```
package com.ssafy.linkedlist;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Iterator;
import java.util.LinkedList;

/**
 *
 * @author comkkyu
 * @date 21. 8. 17
 *
 * 백준 1406 - 에디터
 * https://www.acmicpc.net/problem/1406
 *
 */
public class B1406 {

  static LinkedList<Character> list;
  static int cursor, len, M;

  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String str = br.readLine(); // 초기 입력 받은 문자열

    list = new LinkedList<>();

    for (int i = 0; i < str.length(); i++) {
      list.add(str.charAt(i));
    }

    cursor = list.size(); // 명령어 수행전 커서의 위치는 문장의 맨뒤
    len = list.size(); // 초기 문자열의 길이

    M = Integer.parseInt(br.readLine()); // 명령어 개수
```

```
    while (M-- > 0) {
      String[] command = br.readLine().split(" ");

      char com = command[0].charAt(0);

      switch (com) {
      case 'L':
        if (cursor != 0) cursor--;
        break;
      case 'D':
        if (cursor < len) cursor++;
        break;
      case 'B':
        if (cursor != 0) {
          list.remove(--cursor);
          len--;
        }
        break;
      case 'P':
        char c = command[1].charAt(0);
        list.add(cursor, c);
        len++;
        cursor++;
        break;
      }
    }

    Iterator<Character> it = list.iterator();
    while (it.hasNext()) {
      System.out.print(it.next());
    }

    br.close();
  }
}
```

| | | | | | | |
|---|---|---|---|---|---|---|
| comkkyu | 1406 | 시간 초과 | | | Java 11 / 수정 | 1518 B |

두번째 시도 (with Stack, System.out.print())

❌ 인덱스를 이용한 접근방법으로 풀 때 해당문제가 시간초과가 일어나는 걸 확인했다. 주어진 문제에서 처럼 커서를 기준으로 왼쪽, 오른쪽에서 연산이 일어나는 O(1) 의 시간복잡도를 가진 연산이 필요하다. 지금까지 배운 자료구조중 해당 연산을 수행할 수 있는 자료구조가 있는지 먼저 고민해 보았다. LIFO(Last In Last Out) 형식을 가진 Stack을 활용한다면 문제를 해결 할 수 있다. 커서를 기준으로 왼쪽과 오른쪽에 대해서 삽입, 삭제가 이루어지기 때문에 두개의 스택을 활용해서 구현했다. 명령어가 수행되기 전에는 문장의  맨 뒤에 커서가 위치하고 있다고 했으므로 커서를 기준으로 왼쪽 스택에 초기 문자열에 대한 모든 내용이 담겨있고 오른쪽 스택은 비어있는 상태가 된다. top에 대해서 pop 과 push 연산으로 O(1) 의 연산을 할 수 있도록 구현했지만 여전히 시간초과가 나타났다.

```
package com.ssafy.linkedlist;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Stack;
import java.util.StringTokenizer;

/**
 *
 * @author comkkyu
 * @date 21. 8. 17
 *
 * 백준 1406 - 에디터
 * https://www.acmicpc.net/problem/1406
 *
 */
public class B1406_stack {

  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
    StringTokenizer st;
    String str = br.readLine();
    Stack<Character> left = new Stack<>();
    Stack<Character> right = new Stack<>();
    int M = 0; // 명령어 개수
```

```
    for (int i = 0; i < str.length(); i++) {
      left.push(str.charAt(i)); // 처음에는 커서가 맨 뒤이므로 모든 문자들이 커서 왼쪽에 위치함
    }

    M = Integer.parseInt(br.readLine());

    while (M-- > 0) {
      st = new StringTokenizer(br.readLine());
      char command = st.nextToken().charAt(0);

      switch (command) {
      case 'L': // 커서를 왼쪽으로 한 칸 옮김
        // 커서 왼쪽에 있던 내용이 오른쪽으로 옮겨져야 함
        // 커서가 문장의 맨 앞이면 무시됨
        if (!left.isEmpty()) right.push(left.pop());
        break;
      case 'D': // 커서를 오른쪽으로 한 칸 옮김
        // 커서 오른쪽에 있던 내용이 왼쪽으로 옮겨져야 함
        // 커서가 문장의 맨 뒤이면 무시됨
        if (!right.isEmpty()) left.push(right.pop());
        break;
      case 'B': // 커서 왼쪽에 있는 문자를 삭제함
        // 왼쪽 스택 top 에 있는 내용이 커서의 바로 왼쪽에 있는 내용이므로 pop
        // 커서가 문장의 맨 앞이면 무시됨
        if (!left.isEmpty()) left.pop();
        break;
      case 'P': // 다음에 오는 문자를 커서 왼쪽에 추가함
        // 커서 왼쪽 영역인 left 스택의 top에 해당 문자를 추가
        char c = st.nextToken().charAt(0);
        left.push(c);
        break;
      }
    }

    while (!left.isEmpty()) {
      right.push(left.pop());
    }

    while (!right.isEmpty()) {
      System.out.print(right.pop());
    }

    br.close();
  }

}
```

| | | | | | | |
|---|---|---|---|---|---|---|
| comkkyu | 3 1406 | 시간 초과 | | | Java 11 / 수정 | 1376 B |

세번째 시도 (with Stack, BufferedWriter)

👌 연산속도를 줄였는데도 시간초과가 발생! 출력문에서 시간초과가 나는 것 같아서 System.out.print 대신 BufferedWriter를 사용해서 문제를 해결했다. 입력된 데이터가 바로 전달되지 않고 버퍼를 거쳐서 전달되므로 데이터 처리 효율성이 높다. 많은 양의 데이터를 처리할 때 유리하다. BufferedWriter 사용시 버퍼를 잡아 놓았기 때문에 flush(), close()를 반드시 호출해 줘야 한다. 또한 println 처럼 자동개행 기능이 없기 때문에 개행이 필요한 경우 개행문자(\n)를 통해 따로 처리해 줘야 한다.

```
package com.ssafy.linkedlist;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.Stack;
import java.util.StringTokenizer;

/**
 *
 * @author comkkyu
 * @date 21. 8. 17
 *
 * 백준 1406 - 에디터
 * https://www.acmicpc.net/problem/1406
 *
```

```
 */
public class B1406_stack {

  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
    StringTokenizer st;
    String str = br.readLine();
    Stack<Character> left = new Stack<>();
    Stack<Character> right = new Stack<>();
    int M = 0; // 명령어 개수

    for (int i = 0; i < str.length(); i++) {
      left.push(str.charAt(i)); // 처음에는 커서가 맨 뒤이므로 모든 문자들이 커서 왼쪽에 위치함
    }

    M = Integer.parseInt(br.readLine());

    while (M-- > 0) {
      st = new StringTokenizer(br.readLine());
      char command = st.nextToken().charAt(0);

      switch (command) {
      case 'L': // 커서를 왼쪽으로 한 칸 옮김
        // 커서 왼쪽에 있던 내용이 오른쪽으로 옮겨져야 함
        // 커서가 문장의 맨 앞이면 무시됨
        if (!left.isEmpty()) right.push(left.pop());
        break;
      case 'D': // 커서를 오른쪽으로 한 칸 옮김
        // 커서 오른쪽에 있던 내용이 왼쪽으로 옮겨져야 함
        // 커서가 문장의 맨 뒤이면 무시됨
        if (!right.isEmpty()) left.push(right.pop());
        break;
      case 'B': // 커서 왼쪽에 있는 문자를 삭제함
        // 왼쪽 스택 top 에 있는 내용이 커서의 바로 왼쪽에 있는 내용이므로 pop
        // 커서가 문장의 맨 앞이면 무시됨
        if (!left.isEmpty()) left.pop();
        break;
      case 'P': // 다음에 오는 문자를 커서 왼쪽에 추가함
        // 커서 왼쪽 영역인 left 스택의 top에 해당 문자를 추가
        char c = st.nextToken().charAt(0);
        left.push(c);
        break;
      }
    }

    while (!left.isEmpty()) {
      right.push(left.pop());
    }

    while (!right.isEmpty()) {
      bw.write(right.pop());
    }

    bw.flush();
    bw.close();
    br.close();
  }

}
```

| comkkyu | 1406 | 맞았습니다!! | 122240 KB | 708 ms | Java 11 / 수정 | 1557 B |
|---|---|---|---|---|---|---|

## 2. 백준 17827번 달팽이 리스트

▼ 문제 링크

17827번: 달팽이 리스트

문제 영진이는 달팽이를 좋아한다. 달팽이를 너무너무 좋아하기 때문에 특정한 모양의 단방향 연결리스트에 달팽이 리스트라는 이름을 붙여주었다. 일반적인 선형 단방향 연결리스트의 각 노드 번호를 연결된 순서대로 1, 2, ..., N이라 하자. 이때 N번 노드는 아무 노드도 가리키지 않는데, 여기서 N 번 노드가 1번 노드를 제외한 임의의 노드

https://www.acmicpc.net/problem/17827

BAEKJOON ONLINE JUDGE

첫번째 시도 (with LinkedList, Recursive)

❌ 문제에 나와있는 대로 연결리스트로 구현했다. K 값이 노드의 개수 N 보다 작으면 그대로 K를 반환하면 되고 노드 개수와 마지막 연결노드의 번호 V가 동일하다면 K 값이 N 보다 큰 경우에는 자기 자신 cycle 안에서 계속 맴돌고 있으므로 N - 1을 반환한다. K 같은 경우에는 1번 노드 부터 시작해서 한칸 이동한게 K = 1 이기 때문에 K 그대로 반환했지만 N의 경우 인덱스를 맞춰주기 위해서 N - 1을 반환해주었다. K가 노드 개수 N보다 큰 경우에는 cycle 을 돌기 때문에 K 값에서 재귀적으로 cycle의 크기만큼 계속 빼주도록 구현했다. 결과는 시간초과..! 재귀로 풀만한 시간제한의 문제는 아닌 걸로...!!

```
package com.ssafy.linkedlist;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.LinkedList;
import java.util.StringTokenizer;

/**
 *
 * @author comkkyu
 * @date 21. 8. 17
 *
 * 백준 17827 - 달팽이 리스트
 * https://www.acmicpc.net/problem/17827
 *
 */
public class B17827 {

  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    StringTokenizer st = new StringTokenizer(br.readLine());
    StringBuilder sb = new StringBuilder();
    int N = Integer.parseInt(st.nextToken());
    int M = Integer.parseInt(st.nextToken());
    int V = Integer.parseInt(st.nextToken());

    LinkedList<Integer> list = new LinkedList<>();
    st = new StringTokenizer(br.readLine());

    while (st.hasMoreTokens()) {
      list.add(Integer.parseInt(st.nextToken()));
    }

    for (int m = 0; m < M; m++) {
      int k = Integer.parseInt(br.readLine());

      sb.append(list.get(where(N, k, V))+"\n");
    }zx

    System.out.println(sb);
  }

  /**
   * cycle 까지 계산해 실제 노드 위치를 반환하는 메서드
   *
   * @param n 노드 개수
   * @param k 이동 횟수
   * @param v n 번 노드가 가리키는 번호
   * @return k 번 이동한 노드의 인덱스 번호
   */
  private static int where(int n, int k, int v) {
    if (k < n) return k; // cycle 진입 전에는 노드번호 반환
    if (v == n) return n - 1; // 민달팽이 리스트

    return where(n, k - (n - v + 1), v); // 생성되는 cycle의 크기 : n - v + 1
//    return (k - v + 1) % (n - v + 1) + (v - 1);
  }

}
```

| | | | | | | |
|---|---|---|---|---|---|---|
| comkkyu | 17827 | 시간 초과 | | | Java 11 / 수정 | 1496 B |

두번째 시도 (with LinkedList, Mathmatical rules)

❌ 1 ~ N 번까지의 노드 N개가 주어질 때 K 가 N 보다 큰 경우에 대해서 수학적 규칙을 찾아볼 수 있다. V 값을 기준으로 cycle이 구성되고 K가 N 보다 큰경우에는 cycle 안에서 움직인다는 뜻이 되므로 cycle을 몇번돌고 cycle의 몇번째 위치에 있는지 나머지 연산을 통해서 구하면 된다. 최종적으로 cycle이 아닌 노드의 개수를 K 에서 빼주고 cycle 개수에 대해서만 cycle 크기로 나머지 연산을 하면 해당 나머지 값이 cycle의 위치가 된다. 여기서 주의할 점은 cycle 기준으로 첫번째는 0, 두번째는 1, .. 값을 구한거기 때문에 다시 cycle이 아닌 앞에 있는 노드의 개수만큼 더해줘야 최종적으로 검색하고자 하는 index 를 구할 수 있다. 하지만 재귀와 마찬가지로 시간초과..!! ㅜㅜ

```
package com.ssafy.linkedlist;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.LinkedList;
import java.util.StringTokenizer;

/**
 *
 * @author comkkyu
 * @date 21. 8. 17
 *
 * 백준 17827 - 달팽이 리스트
 * https://www.acmicpc.net/problem/17827
 *
 */
public class B17827 {

  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    StringTokenizer st = new StringTokenizer(br.readLine());
    StringBuilder sb = new StringBuilder();
    int N = Integer.parseInt(st.nextToken());
    int M = Integer.parseInt(st.nextToken());
    int V = Integer.parseInt(st.nextToken());

    LinkedList<Integer> list = new LinkedList<>();
    st = new StringTokenizer(br.readLine());

    while (st.hasMoreTokens()) {
      list.add(Integer.parseInt(st.nextToken()));
    }

    for (int m = 0; m < M; m++) {
      int k = Integer.parseInt(br.readLine());

      sb.append(list.get(where(N, k, V))+"\n");
    }

    System.out.println(sb);
  }

  /**
   * cycle 까지 계산해 실제 노드 위치를 반환하는 메서드
   *
   * @param n 노드 개수
   * @param k 이동 횟수
   * @param v n 번 노드가 가리키는 번호
   * @return k 번 이동한 노드의 인덱스 번호
   */
  private static int where(int n, int k, int v) {
    if (k < n) return k; // cycle 진입 전에는 노드번호 반환
    if (v == n) return n - 1; // 민달팽이 리스트

//    return where(n, k - (n - v + 1), v); // 생성되는 cycle의 크기 : n - v + 1
    return (k - v + 1) % (n - v + 1) + (v - 1);
  }

}
```

| comkkyu | 17827 | 시간 초과 | | | Java 11 / 수정 | 1588 B |
|---|---|---|---|---|---|---|

세번째 시도 (with ArrayList, Mathmatical rules)

❌ 결국 최종적인 목적은 구한 index 에 대해서 해당 index 위치에 있는 노드 값을 반환하는 것이기 때문에 순차적인 검색에서 LinkedList 보다 빠르기 때문에 ArrayList로 바꿔서 제출해보니 통과가 되었다. 중간에서 추가 및 삭제 작업을 할 때는 LinkedList가 빠르지만 인덱싱을 활용한 순차적인 접근에는 ArrayList가 빠르므로 상황에 맞게 적절하게 사용하는 능력이 필요하다..!  인덱싱을 이용해서 해결했기 때문에 배열로 구현해도 해결이 될 것 같아 보인다.

```
package com.ssafy.linkedlist;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.StringTokenizer;

/**
 *
 * @author comkkyu
 * @date 21. 8. 17
 *
 * 백준 17827 - 달팽이 리스트
 * https://www.acmicpc.net/problem/17827
 *
 */
public class B17828_ArrayList {

  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    StringTokenizer st = new StringTokenizer(br.readLine());
    StringBuilder sb = new StringBuilder();
    int N = Integer.parseInt(st.nextToken());
    int M = Integer.parseInt(st.nextToken());
    int V = Integer.parseInt(st.nextToken());

    ArrayList<Integer> list = new ArrayList<>();
    st = new StringTokenizer(br.readLine());

    while (st.hasMoreTokens()) {
      list.add(Integer.parseInt(st.nextToken()));
    }

    for (int m = 0; m < M; m++) {
      int k = Integer.parseInt(br.readLine());

      sb.append(list.get(where(N, k, V))+"\n");
    }

    System.out.println(sb);
  }

  /**
   * cycle 까지 계산해 실제 노드 위치를 반환하는 메서드
   *
   * @param n 노드 개수
   * @param k 이동 횟수
   * @param v n 번 노드가 가리키는 번호
   * @return k 번 이동한 노드의 인덱스 번호
   */
  private static int where(int n, int k, int v) {
    if (k < n) return k; // cycle 진입 전에는 노드번호 반환
    if (v == n) return n - 1; // 민달팽이 리스트

//    return where(n, k - (n - v + 1), v); // 생성되는 cycle의 크기 : n - v + 1
    return (k - v + 1) % (n - v + 1) + (v - 1);
  }

}
```

| 17827 | 맞았습니다!! | 76088 KB | 856 ms | Java 11 / 수정 | 1585 B |
|---|---|---|---|---|---|

# 스터디 팀원 풀이 발표

## 1. 풍선 터뜨리기

▼ 문제 링크

2346번: 풍선 터뜨리기

1번부터 N번까지 N개의 풍선이 원형으로 놓여 있고. i번 풍선의 오른쪽에는 i+1번 풍선이 있고, 왼쪽에는 i-1번 풍선이 있다. 단, 1번 풍선의 왼쪽에 N번 풍선이 있고, N번 풍선의 오른쪽에 1번 풍선이 있다. 각 풍선 안에는 종이가 하나 들어있고, 종이에는 -N보다 크거나 같고, N보다 작거나 같은 정수가 하나 적혀있다. 이 풍선들을 다음과

https://www.acmicpc.net/problem/2346

## 김유진님 풀이

```
/**
 * 풍선의 번호와 다음에 터뜨릴 풍선의 위치를 클래스로 만들었으며,
 * 다음에 터트릴 풍선의 위치가 양수인 경우에는 풍선을 터뜨리면서 다음 풍선들의 인덱스가 1씩 감소했기 때문에 다음 풍선의 위치에서 1을 빼고 이동했으며,
 * 음수의 경우에는 인덱스의 변화가 없으므로 그대로 인덱스를 옮겨서 풍선을 터뜨리는 방식으로 구현하였다.
 * 인덱스를 갱신하는 과정에서 음수거나 인덱스를 벗어나는 경우 나머지 연산과 리스트 크기만큼 더하는 연산을 통해서 0~list.size()-1 범위로 갱신하였다.
 * 다만 마지막 과정에서 리스트가 비어 있으므로 0나누기 에러가 발생한다. 이를 막기 위하여 if를 사용하였다.
 */

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.LinkedList;
import java.util.Scanner;

//public class Main {

public class B2346_풍선터뜨리기_1 {
   static LinkedList<balloon> l;
   static int N;

   public static void main(String[] args) throws FileNotFoundException {
      System.setIn(new FileInputStream("2346_input"));
      Scanner sc = new Scanner(System.in);
      N = sc.nextInt();
      l = new LinkedList<balloon>();
      for (int i = 0; i < N; i++)
         l.add(new balloon(i + 1, sc.nextInt())); // 각 풍선들을 리스트에 넣어준다. (풍선 번호, 다음에 터트릴 풍선의 위치)
      int idx = 0; // 처음 풍선 인덱스
      while (!l.isEmpty()) { // 풍선이 없다면 종료
         balloon tmp = l.remove(idx); // 해당 풍선을 터트린다.
         if (tmp.next > 0)
            idx += tmp.next - 1; // 양수인 경우에는 풍선이 터지면서 다른 풍선들의 인덱스가 하나씩 줄었다.
         else
            idx += tmp.next; // 그 외의 경우에는 상관없다.

         if (!l.isEmpty()) {// 위에서 리스트에서 목표 풍선을 제거함
            idx %= l.size(); // 0으로 나누는 경우가 있다., 나누는 이유 : 몫만큼 회전하는건 제자리를 도는거라 의미가 없다.
            if (idx < 0)
               idx += l.size(); // 0보다 작은 경우에는 사이즈 만큼 더해서 0~l.size()-1 인덱스로 재 설정 해준다.
         }

         System.out.print(tmp.num + " "); // 터진 풍선의 번호 출력
      }
      sc.close();
   }

   static class balloon {
      int num; // 풍선 번호
      int next; // 다음에 몇번째 풍선을 터트릴꺼냐?

      public balloon(int num, int next) { // 생성자
         super();
         this.num = num;
         this.next = next;
      }
   }
}
```

## 민선규님 풀이

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Deque;
import java.util.LinkedList;
import java.util.StringTokenizer;

public class Baekjoon_2346 {

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

```
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); // 버퍼 선언, 생성
        int N = Integer.parseInt(br.readLine()); // 풍선의 수 입력받기
        Deque<pair> deque = new LinkedList<>(); //pair 타입 Deque 선언
        StringBuilder sb = new StringBuilder(); // 스트링빌더 생성

        StringTokenizer st = new StringTokenizer(br.readLine()); // 풍선의 정보를 받기 위해서 st 생성

        for (int i = 1; i <= N; i++) {
           deque.add(new pair(i, Integer.parseInt(st.nextToken()))); // 풍선 정보를 deque에 추가할 때 몇번 째 풍선인지와 함께 풍선안의 내용도 같이
        }

        while (!deque.isEmpty()) {

           sb.append(deque.getFirst().index+" "); // 첫번째 풍선 인덱스 저장
           int next = deque.poll().next; // 다음 풍선을 알기 위해서 풍선 내부에 저장되어있는 내용 확인

           if (deque.isEmpty()) // 모든 풍선이 터졌으면 반복문 중지
              break;

           if(next > 0) { // 안에 내용물이 양수
              next -= 1; // 풍선이 터지면서 한칸 씩 당겨지므로 next -1 해주기
              for(int i = 0; i <next ; i ++) { // 남은 넥스트만큼 왼쪽으로 이동하기
                 deque.addLast(deque.pollFirst());
              }
           }

           if(next <= 0) { // 안에 내용물이 음수
              for(int i = 0; i < Math.abs(next) ; i ++) { // 남은 넥스트만큼 오른쪽으로 이동하기
                 deque.addFirst(deque.pollLast());
              }
           }
        }

        System.out.println(sb.toString()); // 최종출력
     }

}

class pair { // 배열안에 2개의 정보를 넣기 위한 pair 클래스
     int index; // 몇 번째 풍선인지
     int next; // 풍선 안에 내용이 무엇인지

     pair(int index, int next) { // pair 생성자
        this.index = index;
        this.next = next;
     }

}
```

❌ 위의 풀이의 경우 메모리초과가 발생함. 팀원들과 해당 코드에 대해서 리뷰하는 시간을 가졌지만 메모리초과에 대해서는 아직 관련 지식이 부족해서 해결하지는 못함. 각자 덱을 사용해서 구현해보는걸로 결론을 내렸다.

---

## 2. 회전하는 큐

▼ 문제 링크

1021번: 회전하는 큐

지민이는 N개의 원소를 포함하고 있는 양방향 순환 큐를 가지고 있다. 지민이는 이 큐에서 몇 개의 원소를 뽑아내려고 한다. 지민이는 이 큐에서 다음과 같은 3가지 연산을 수행할 수 있다. 큐에 처음에 포함되어 있던 수 N이 주어진다. 그리고 지민이가 뽑아내려고 하는 원소의 위치가 주어진다. (이 위치는 가장 처음 큐에서의 위치이다.)

https://www.acmicpc.net/problem/1021

### 김유진님 풀이

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.LinkedList;
import java.util.StringTokenizer;

/**
 * 왼쪽의 이동, 오른쪽이동, 타겟의 인덱스를 반환하는 메서드를 구현하였으며,
 * 목표의 index가 중앙보다 앞에 있는경우 왼쪽으로 돌리고, 중앙보다 뒤에 있는 경우에는 오른쪽으로 돌려서 목표를 꺼내는 방식으로 구현하였다.
 * 우식님은 메서드를 만들어서 index를 찾지 않고 list.indexOf(); 메서드를 이용해서 리스트의 인덱스를 찾았으며,
```

```
 *  deque을 사용해서 구현하는 것이 좀 더 효율적인 사용임을 배울 수 있었다.
 */
public class B1021_회전하는큐_1 {
   static int N, M, cnt;
   static LinkedList<Integer> l;
   static int[] target;

   public static void main(String[] args) throws IOException {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      StringTokenizer st = new StringTokenizer(br.readLine());
      N = Integer.parseInt(st.nextToken());
      M = Integer.parseInt(st.nextToken());
      l = new LinkedList<Integer>();
      target = new int[M];
      st = new StringTokenizer(br.readLine());
      for (int m = 0; m < M; m++) {
         target[m] = Integer.parseInt(st.nextToken()); // 목표를 저장한다.
      }
      for (int n = 1; n <= N; n++) {
         l.add(n); // 1 ~ N 을 입력한다.
      }

      for (int i = 0; i < M; i++) {
         int tmp = target[i]; // 타겟 저장
         while (l.size() != N - M) {
            int idx = search(tmp); // 목표의 인덱스 출력
            if (tmp == l.get(0)) { // 목표와 맨앞의 요소가 같다면
               l.remove(0); // 빼내기
               break; // 그럼 다음 타겟을 노려야 한다. 반복문 종료
            }
            if (idx > l.size() / 2) // 목표의 인덱스가 중앙보다 뒤라면
               Rrotate(); // 목표를 뒤로 빼내는게 빠르다.
            else // 목표의 인덱스가 중앙보다 앞이라면
               Lrotate(); // 목표를 앞으로 빼내는게 빠르다.

            cnt++; // rotate 연산 횟수 체크
         }
      }

      System.out.println(cnt); // rotate 연산 횟수 출력

   }

   static int search(int tar) {
      for (int i = 0; i < l.size(); i++) {
         if (l.get(i) == tar) // 타겟이 몇번에 위치해 있는지
            return i; // 인덱스를 반환
      }
      return -1;
   }

   static void Lrotate() {
      l.addLast(l.remove(0)); // 0 번째 (맨 앞)요소를 빼서 맨 뒤에 집어 넣는다.
   }

   static void Rrotate() {
      l.addFirst(l.remove(l.size() - 1)); // 맨 마지막(맨 뒤) 요소를 빼서 맨 앞에 집어 넣는다.
   }
}
```

## 유우식님 풀이

```
import java.util.LinkedList;
import java.util.Scanner;

public class BOJ_1021 {
   static LinkedList<Integer> list;
   static int N, M;

   public static void main(String[] args) {
      list = new LinkedList<Integer>();
      Scanner sc = new Scanner(System.in);
      int cnt = 0;
      // 리스트 크기
      N = sc.nextInt();
      // 뽑을 숫자 횟수
      M = sc.nextInt();
      // 리스트의 원소 위치 저장할 배열
      int[] loc = new int[M];
      // 리스트에 아무거나 집어넣기
      for (int i = 1; i <= N; i++) {
         list.add(i);
      }
```

```
      for (int i = 0; i < M; i++) {
         loc[i] = sc.nextInt();
      }

      // 순서 찾을 횟수만큼 반복
      for (int i = 0; i < M; i++) {
         int target_idx = list.indexOf(loc[i]);
         int half_idx;
         if (list.size() % 2 == 0) {
            half_idx = list.size() / 2 - 1;

         } else {
            half_idx = list.size() / 2;
         }
         // 원소의 위치가 리스트의 크기의 반보다 작으면
         // 2번연산 ㄱㄱ
         if (target_idx <= half_idx) {
            // 기존의 원소 위치의 값을 임시 저장
            for (int j = 0; j < target_idx; j++) {
               leftspin();
               cnt++;
            }
         }
         // 원소의 위치가 리스트의 크기의 반보다 크면
         // 3번연산 ㄱㄱ
         else {
            for (int j = 0; j < list.size() - target_idx; j++) {
               rightspin();
               cnt++;
            }

         }
         // 연산 후 첫번째 원소 빼기
         list.pollFirst();

      }

      System.out.println(cnt);

   }

   // 2번 연산
   static void leftspin() {
      // 첫번째 원소 저장 및 삭제
      int first = list.pollFirst();
      // 첫번째 원소 마지막 위치에 삽입
      list.addLast(first);
   }

   // 3번 연산
   static void rightspin() {
      // 마지막 원소 저장 및 삭제
      int last = list.pollLast();
      // 마지막원소 첫번째 삽입
      list.addFirst(last);

   }

}
```

## 새롭게 알게된 내용

▼ ListIterator

http://tcpschool.com/java/java_collectionFramework_iterator

▼ indexOf

http://tcpschool.com/java/java_api_string

▼ deque

http://tcpschool.com/java/java_collectionFramework_stackQueue