Chapter 06.
파트별 쪽지시험 #4

Clip 01 | [14426] 접두사 찾기
이분탐색

Clip 02 | [1021] 회전하는 큐
큐, 아이디어

Clip 03 | [15823] 카드 팩 구매하기
이분탐색, 투포인터

Clip 04 | [14597] Quilting (Large)
DP

Clip 05 | [15906] 변신 이동 게임
탐색, 우선순위큐

Ch06. 파트별 쪽지시험 #4

1. [14426] 접두사 찾기

BOJ14426: 접두사 찾기

문제 요약

- N개의 문자열 집합 (1 <= N <= 10,000)
- M개의 문자열 집합 (1 <= M <= 10,000)

- 문자열의 길이는 500 이하

- N의 접두사 중 M과 일치 하는 경우는 몇가지인가?

BOJ14426: 접두사 찾기

문제 분석

- 완전 탐색을 한다면?
  - N * M * {글자길이}
  - 1만 * 1만 * 500 == 500억

- 어떻게 하면 검사를 덜 할 수 있을까?

BOJ14426: 접두사 찾기

문제 분석

- 만약 종이 사전에서 Banana라는 단어를 찾는다고 생각해보자
- 글자를 찾기 위해 사전의 모든 글자를 다 봐야 할까?
- 아니다, 현재 펼친 페이지가 찾고자 하는 단어보다 알파벳순으로 앞인지, 뒤인지를 검사하면 된다

BOJ14426: 접두사 찾기

문제 분석

- 동일한 아이디어를 적용해 보자
- 접두사를 비교하고자 하는 단어를 정렬 해놓고
- 아무 위치의 단어랑 비교해보자
  - 비교한 단어가 사전순으로 더 뒤에 있다면?
    - 찾고자 하는 단어는 더 앞에 있다
  - 비교한 단어가 사전순으로 더 앞에 있다면?
    - 찾고자 하는 단어는 더 뒤에 있다

BOJ14426: 접두사 찾기

문제 분석

시간 복잡도는?

- 10000개의 패턴 단어에 대해서 정렬을 한다
  - O(n log n)
- 대상 단어의 탐색범위가 매 횟수마다 절반으로 줄어든다
  - O(log n) * m * 500

O(n log n) + 500* m * O(log n) == O(n log n)

BOJ14426: 접두사 찾기

문제 분석

- 이제 아무 위치를 아래의 규칙으로 바꿔보자
- 탐색구간 [start, end]
  - 비교 지점 mid: (start + end) /2
- mid 위치의 단어가 사전순으로 앞인지? 뒤인지?
  - 다음 탐색: [start, mid] or [mid + 1, end]

BOJ14426: 접두사 찾기

구현

```
List<String> s = new ArrayList<>();
List<String> p = new ArrayList<>();
for (int i = 0; i < n; i++) {
    s.add(sc.next());
}
for (int i = 0; i < m; i++) {
    p.add(sc.next());
}

s.sort(String::compareTo);
```

입력을 리스트에 받고
찾을 단어를 정렬

BOJ14426: 접두사 찾기

구현

```
for (String str : p) {
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        String target = s.get(mid).substring(0, str.length());
        int compare = target.compareTo(str);
        if (compare == 0){
            cnt++;
            break;
        }
        else if (compare < 0) left = mid + 1;
        else right = mid - 1;
    }
}
```

이분탐색을하며
mid 위치에 있는 단어를 비교

같으면 cnt 증가와 종료

다르면 탐색 범위 재조정

Ch06. 파트별 쪽지시험 #4

2. [1021] 회전하는 큐

BOJ1021: 회전하는 큐

문제 요약

- 1<=N<=50 큐의 크기
- 1<=M<=N 큐에서 꺼낼 원소의 개수

- M개의 큐에서 꺼낼 원소의 초기 위치가 순서대로 입력

- 자료구조는 양방향 큐로 3가지 동작이 있다
  - pop
  - 왼쪽 끝의 원소를 오른쪽 끝으로 이동
  - 오른쪽 끝의 원소를 왼쪽 끝으로 이동->왼쪽 이동

- 원소들을 모두 꺼내는데 필요한 이동의 최소 횟수(pop 제외)출력

BOJ1021: 회전하는 큐

입력 데이터
10 3 2 9 5

출력 데이터
8

문제 요약

1	2	3	4	5	6	7	8	9	10

2	3	4	5	6	7	8	9	10	1

3	4	5	6	7	8	9	10	1

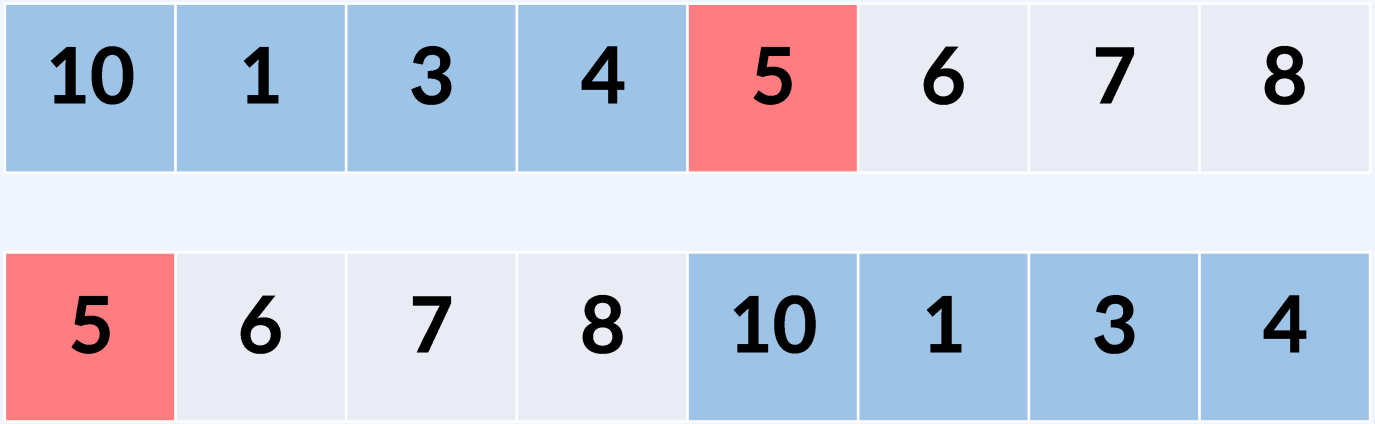
9	10	1	3	4	5	6	7	8

BOJ1021: 회전하는 큐

입력 데이터
10 3 2 9 5

출력 데이터
8

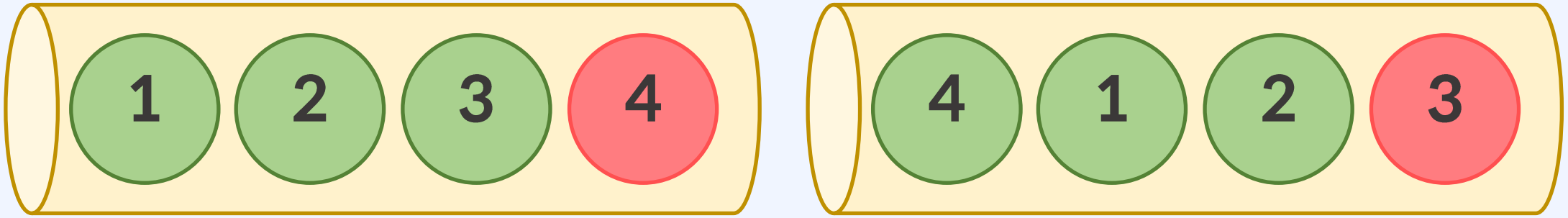
문제 요약



왼 → 오 1회
오 → 왼 3회
오 →왼 or 왼 → 오 4회

BOJ1021: 회전하는 큐

문제 분석



- 오른쪽의 원소를 하나 뽑아서 왼쪽에 넣기?
- 왼쪽의 원소를 N-1개 뽑아서 오른쪽에 넣기

- 무작정 한방향으로 뽑은 뒤, min(N – {뽑은횟수}), {뽑은횟수}) 중에 작은 걸 실제 횟수로 취급하면 된다

BOJ10

구현

```
Queue<Integer> q = new LinkedList<>();
for(int i = 1; i <= n; i++) {
    q.offer(i);
}
int answer = 0;
for(int i = 0; i < m; i++) {
    int cnt = 0;
    while(q.peek() != arr[i]) {
        q.offer(q.poll());
        cnt++;
    }
    answer += Math.min(Math.abs(q.size() - cnt), cnt);
    q.poll();
}
System.out.println(answer);
```

Ch06. 파트별 쪽지시험 #4

3. [15823] 카드 팩 구매하기

BOJ15823: 카드 팩 구매하기

문제 요약

- 카드 팩의 정의
  - 카드들이 서로 인접해 있어야 함
  - 선택한 카드들은 서로 다른 번호를 가지고 있어야 함
  - 카드 팩은 동일한 수의 카드가 들어있어야 함
  - 하나의 카드는 여러 팩에 속할 수 없음

- 카드의 개수: N (1 <= N <= 100,000)
- 카드 식별 번호는 50만 이하의 양의 정수
- 카드 팩의 개수: M (1 <= M <= N)
- M개의 카드 팩을 만들 때
  카드 팩에 넣을 수 있는 카드의 최대 개수 출력

BOJ15823: 카드 팩 구매하기

문제 분석

- 문제에서 생각할 요소가 너무 많다
- M개의 카드 팩을 만들었을 때 팩에 카드가 얼마나 들어가는지?
- 우선 문제를 결정문제로 바꿔서 아래와 같이 생각을 해보자
  - 1개의 카드로 카드 팩 M개를 만들 수 있는가?
  - 2개의 카드로 카드 팩 M개를 만들 수 있는가?
  - 3개의 카드로 카드 팩 M개를 만들 수 있는가?
  - …
  - N개의 카드로 카드 팩 M개를 만들 수 있는가?

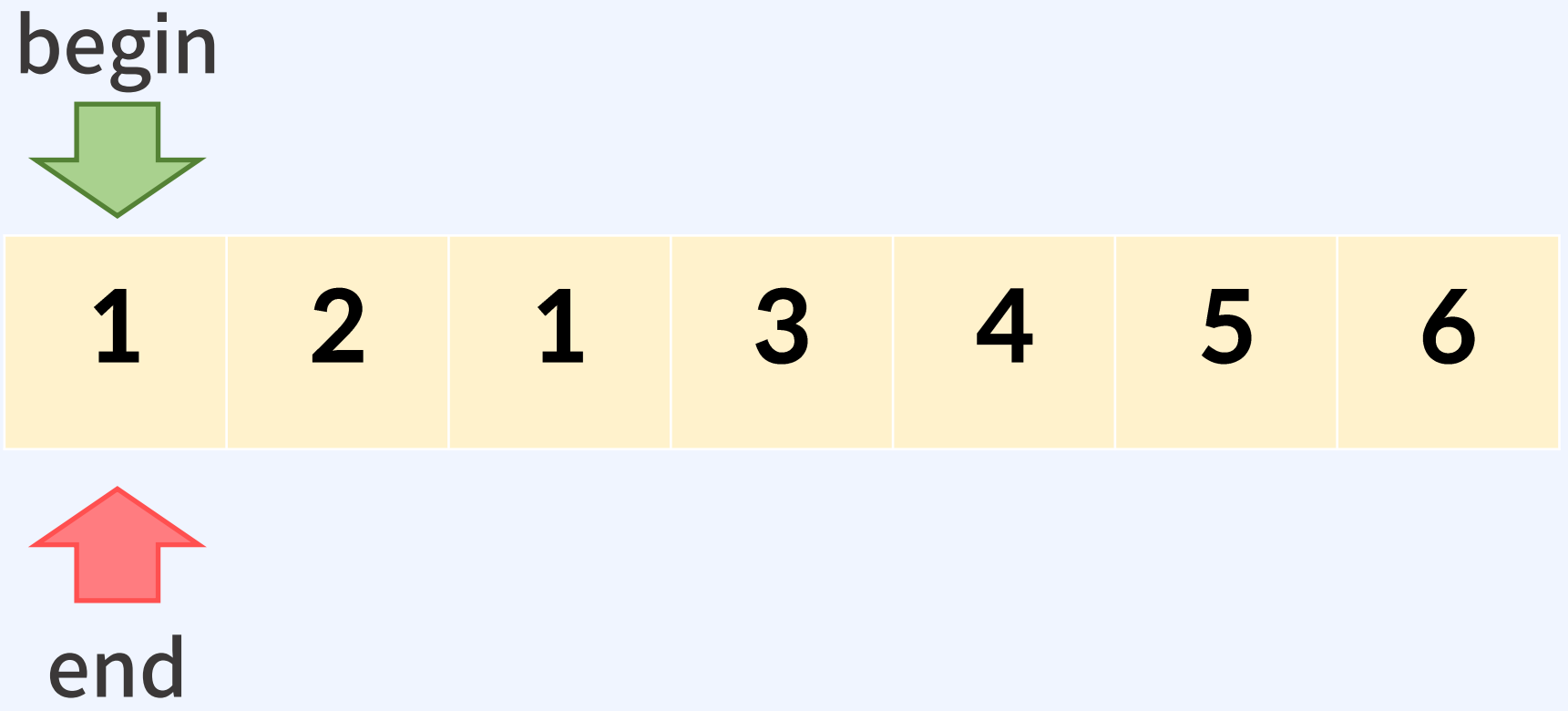
BOJ15823: 카드 팩 구매하기

문제 분석

- 결정 문제를 전부 다 계산한다음 [1, N] 범위에서 가장 큰 값을 고른다
- 그렇다면 우선 카드 팩을 만들 수 있는지 없는지를 판단하는 로직을 구현해보자

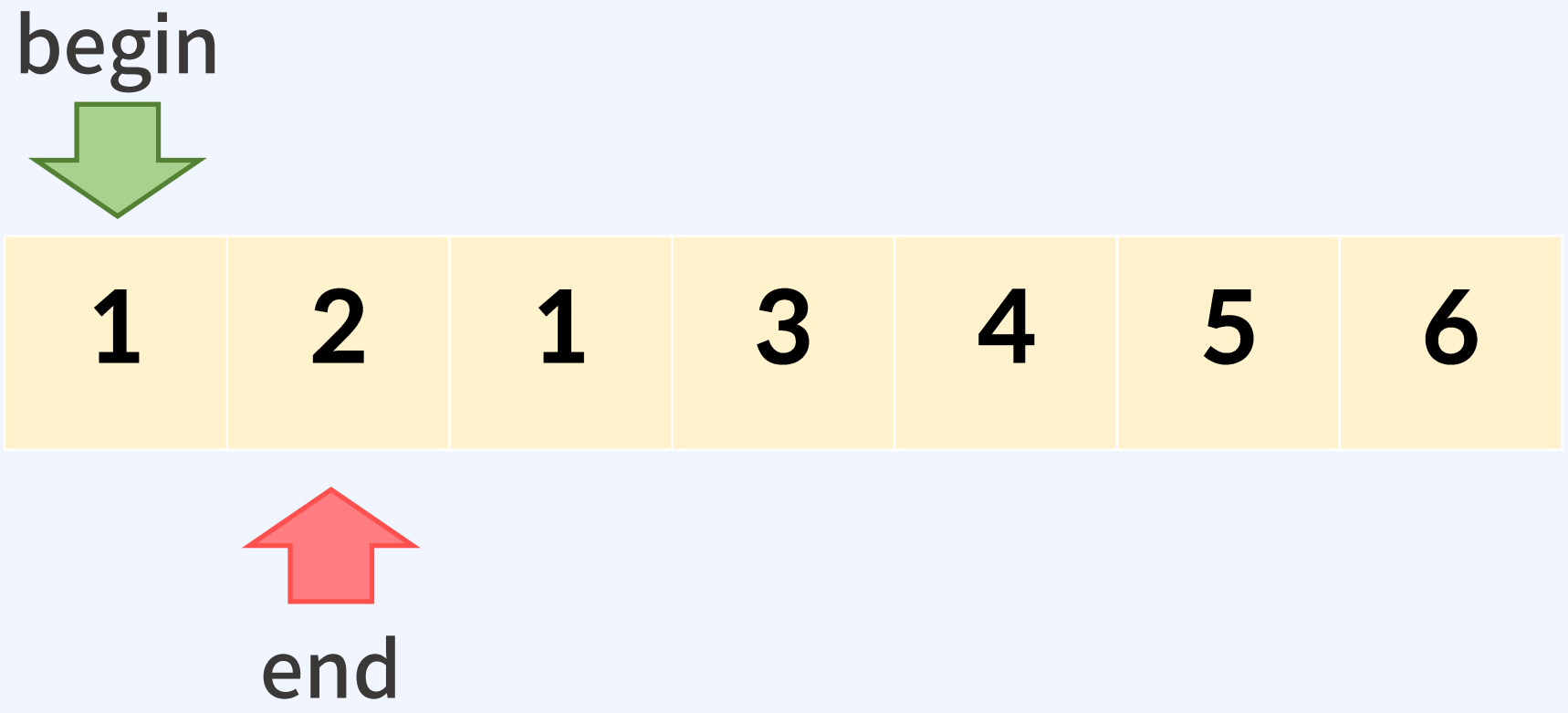
BOJ15823: 카드 팩 구매하기

문제 분석



BOJ15823: 카드 팩 구매하기

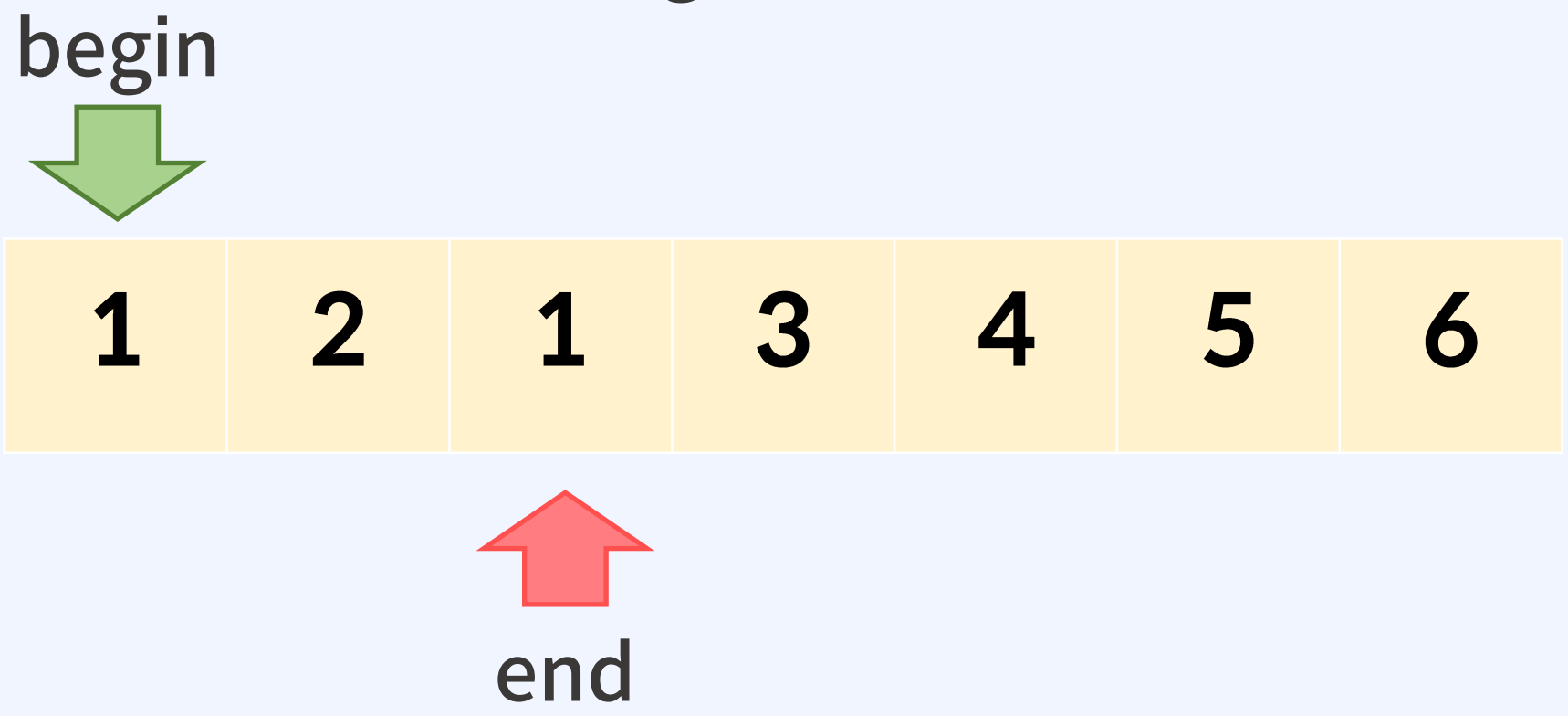
문제 분석



BOJ15823: 카드 팩 구매하기

문제 분석

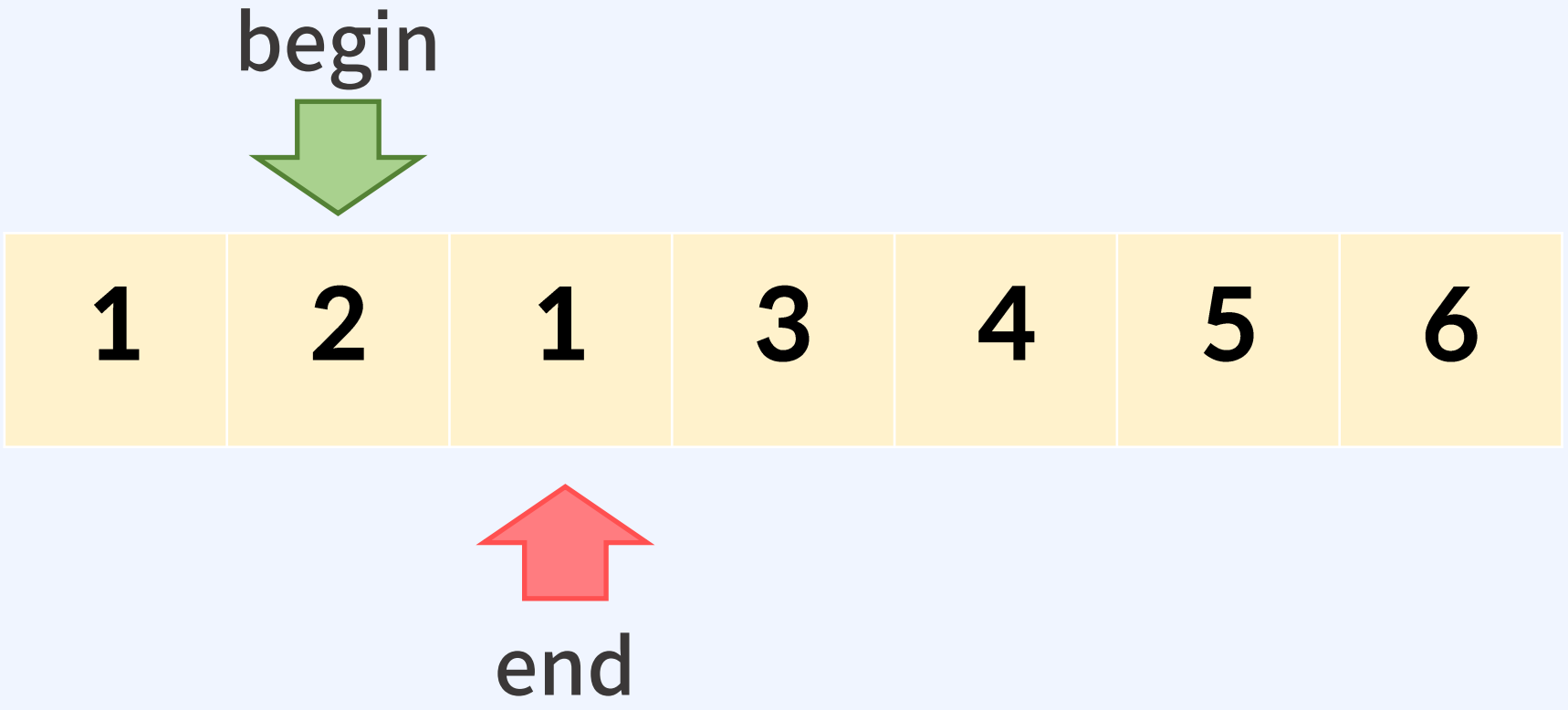
- {end}에 중복되는 수가 나오면?
- {begin}을 이전 중복위치 +1로 이동



BOJ15823: 카드 팩 구매하기

문제 분석

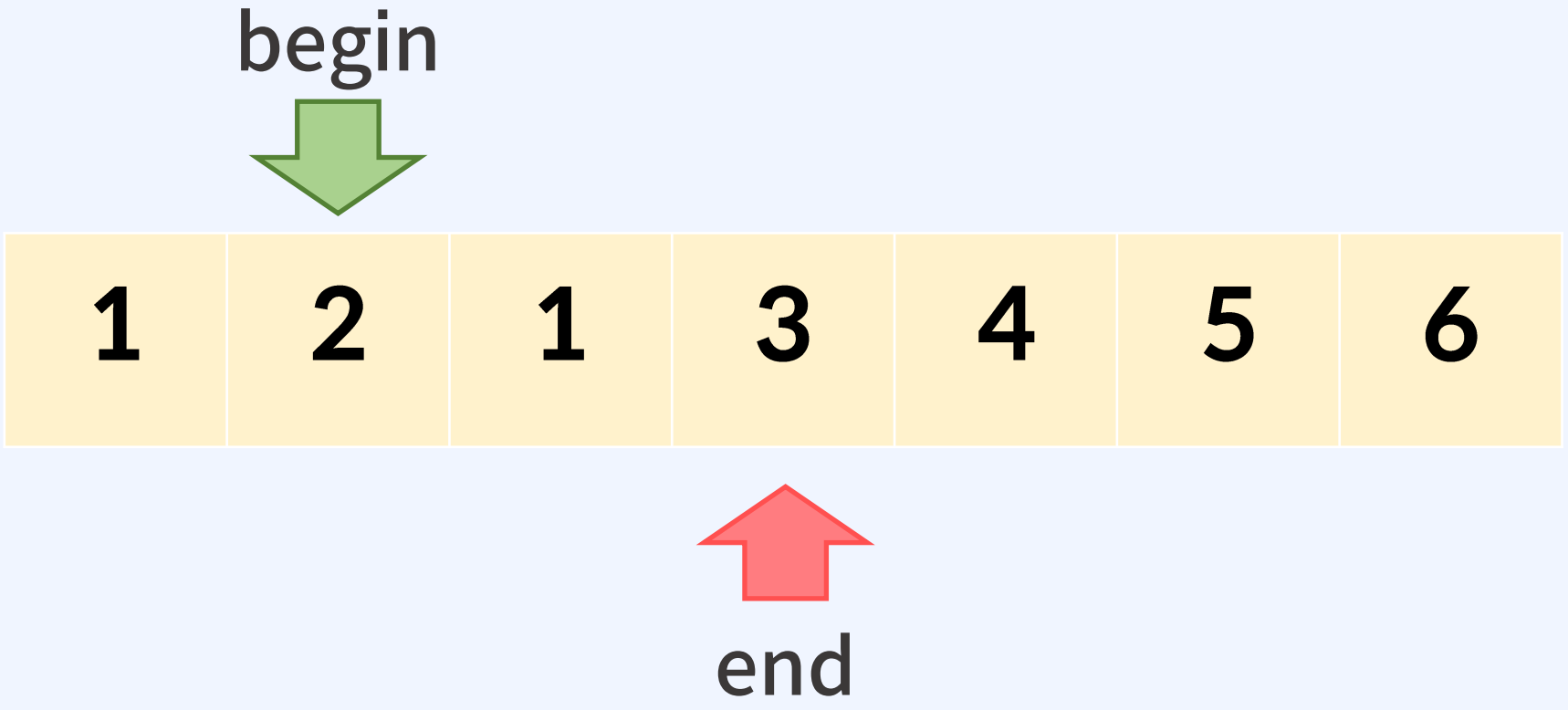
- {end}에 중복되는 수가 나오면?
- {begin}을 이전 중복위치 +1로 이동



BOJ15823: 카드 팩 구매하기

문제 분석

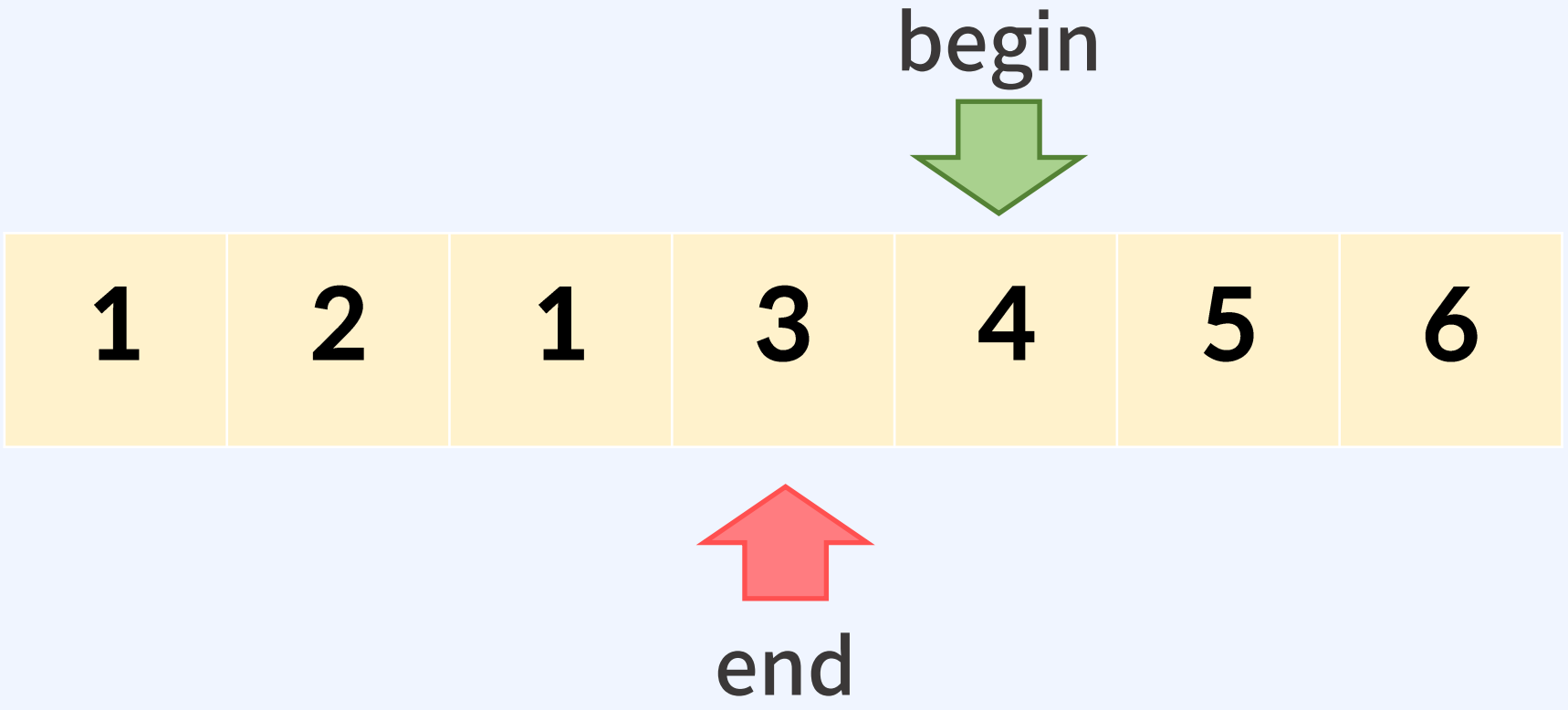
- 원하는 길이만큼 중복이 없다면? (3)
- 개수를 세고, {begin}을 {end} +1 로 이동



BOJ15823: 카드 팩 구매하기

문제 분석

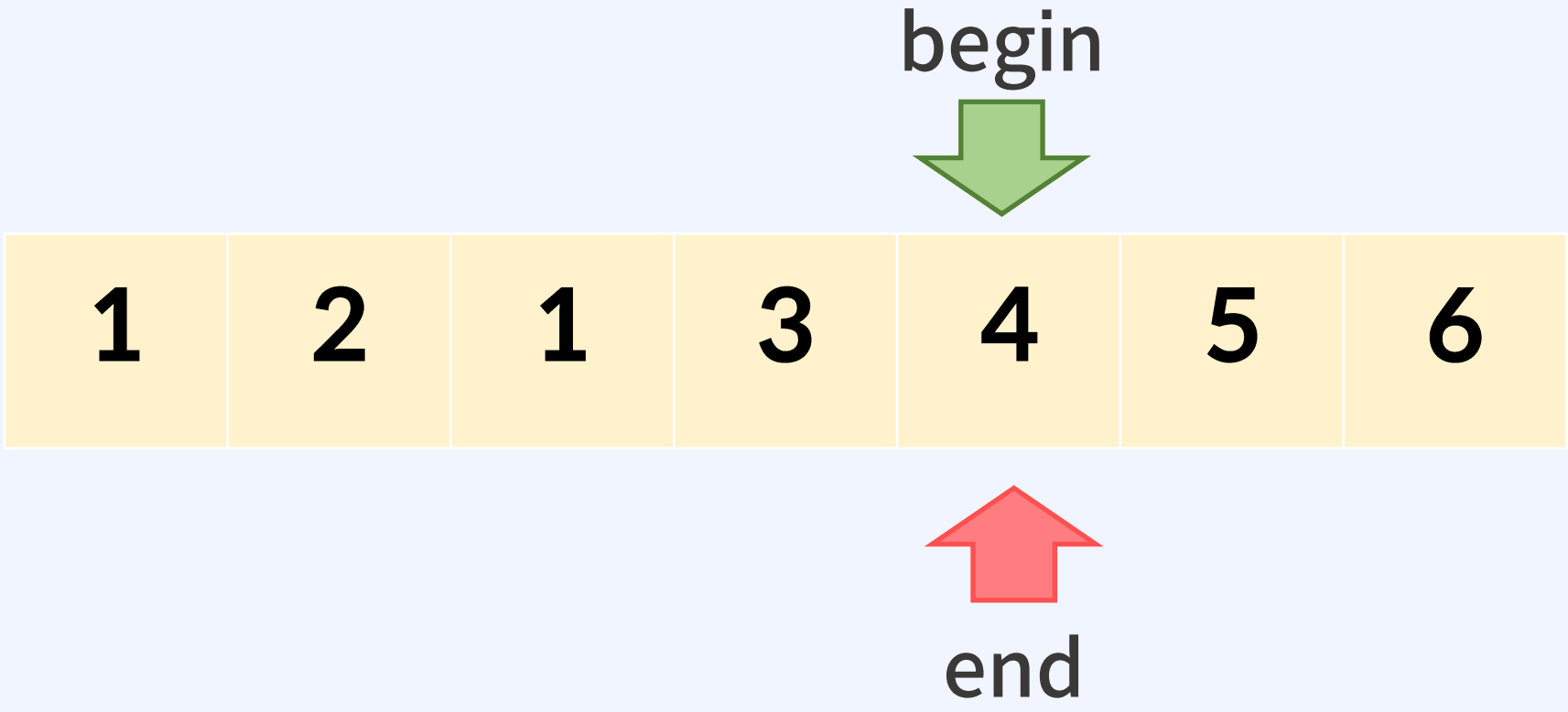
- 원하는 길이만큼 중복이 없다면? (3)
- 개수를 세고, {begin}을 {end} +1 로 이동



BOJ15823: 카드 팩 구매하기

문제 분석

- 원하는 길이만큼 중복이 없다면? (3)
- 개수를 세고, {begin}을 {end} +1 로 이동



BOJ15823: 카드 팩 구매하기

문제 분석

- 투 포인터를 이용한 개수 카운트
- O(n)의 시간 복잡도로 아래 결정문제를 풀 수 있다
- N개의 카드로 카드 팩 M개를 만들 수 있는가?
- 그렇다면 최종 시간 복잡도는? O(n^2)

BOJ15823: 카드 팩 구매하기

문제 분석

- n == 10만 으로, 정해진 시간 안에 문제를 풀 수 없다
- 아래 결정문제에 전부 답해야 하는 걸까?

  - 1개의 카드로 카드 팩 M개를 만들 수 있는가?
  - 2개의 카드로 카드 팩 M개를 만들 수 있는가?
  - 3개의 카드로 카드 팩 M개를 만들 수 있는가?
  - …
  - N개의 카드로 카드 팩 M개를 만들 수 있는가?

BOJ15823: 카드 팩 구매하기

문제 분석

- 만약, 3개의 카드로 카드 팩을 만들 수 있다면?
  - 모든 카드 팩에서 1개를 뺀, 2개로도 만들 수 있다
  - 모든 카드 팩에서 2개를 뺀, 1개로도 만들 수 있다

- 만약 k개의 카드로 카드 팩을 만들 수 있다면?
  - [1, k-1]개의 카드로 카드 팩을 만들 수 있다

BOJ15823: 카드 팩 구매하기

문제 분석

- 만약, 3개의 카드로 카드 팩을 만들 수 있다면?
  - 카드가 1개 늘어난 4개로도 만들 수 없다
  - 카드가 2개 늘어난 5개로도 만들 수 없다

- 만약 k개의 카드로 카드 팩을 만들 수 없다면?
  - [k, n]개의 카드로 카드 팩을 만들 수 있다

BOJ15823: 카드 팩 구매하기

문제 분석

1	2	3	4	5	6
O	O	O	X	X	X

- 정답의 배치에 연속성이 있다 [1, k], [k+1, n]
- 위와 같은 정답배치에는 이분탐색을 적용할 수 있다

BOJ158

구현

```
static boolean decision(int length) {
    int pack = 0;
    int begin = 1; int end = 1;
    Arrays.fill(check, -1);

    while (end <= n) {
        if(check[arr[end]] >= begin) {
            begin = check[arr[end]] + 1;
        }
        check[arr[end]] = end;
        if(end - begin + 1 == length) {
            pack++;
            begin = end + 1;
        }
        if(pack == m) return true;
        end++;
    }
    return false;
}
```

중복 발견 시 처음이 아닌
중복 위치 + 1로 이동

카드 팩 완성 시 카운트
begin을 end+1로 이동

m개의 카드 팩이 만들어지면 true

마지막까지 m개가 안되면 false

BOJ15823: 카드 팩 구매하기

구현

1	2	3	4	5	6
O	O	O	X	X	X

```
static int divide(int start, int end) {
    int mid = (start + end + 1) / 2;
    if(start == mid) return start;
    if(decision(mid)) {
        return divide(mid, end);
    } else {
        return divide(start, mid - 1);
    }
}
```

{mid} 길이로 카드 팩을
m개 만들 수 있다면?

정답은 [mid, end범위]

그렇지 않다면
[start, mid-1] 범위

Ch06. 파트별 쪽지시험 #4

4. [14597] Quilting (Large)

BOJ14597: Quilting (Large)

문제 요약

- 높이가 같은 흑백(Grayscale) 이미지를 좌우로 합침
  - 높이: (1 <= H <= 100), 너비: (1 <= W <= 100)
- 경계선은 전/후 픽셀의 +1, 0, -1 칸만 허용함
- 경계선으로 지정되는 두 이미지의 픽셀의 차이가 최소가 되도록 경계선을 설정
- 부자연도는 차이의 제곱을 더한 값

BOJ14597: Quilting (Large)

문제 분석

- 문제 요구사항: 최소의 부자연스러움
  - d[r][c] = 1행부터 (r, c)까지 경계선을 그었을 때 최소의 부자연 스러움

- 영역만 벗어나지 않는다면, 임의의 열에서 시작하고 임의의 열에서 끝나도 상관이 없다

- 경계 조건만 잘 잡아주고, 동적계획법을 수행하면 된다

BOJ14597: Quilting (Large)

문제 분석

- 초기화
  - 최소 값을 찾아야 하므로
    dp배열 전체에 문제 제약조건상 최대 값을 넣는다
  - 단, 첫번째 행의 연산의 편의를 위해, 0행은 0으로 둔다

BOJ14597: Quilting (Large)

문제 분석

- case1: (r-1, c-1) 에 최소가 들어있는 경우
- case2: (r-1, c) 에 최소가 들어있는 경우
- case3: (r-1, c+1) 에 최소가 들어있는 경우

- 위의 3가지 경우 중 가장 작은 값을 고르고
  (r, c)의 부자연스러움과 합치면 된다
  - (image[0][r][c] – image[1][r][c])^2

- 정답은? h행에서 모든 열 중의 최소 값

BOJ14597: Quilting (Large)

구현

```
for(int r = 1; r <= h; r++) {
    for(int c = 1; c <= w; c++) {
        int case1 = dp[r - 1][c - 1];
        int case2 = dp[r - 1][c];
        int case3 = dp[r - 1][c + 1];
        int diff = image[0][r][c] - image[1][r][c];
        dp[r][c] = Math.min(case1, Math.min(case2, case3)) +
            (diff * diff);
    }
}
```

Ch06. 파트별 쪽지시험 #4

5. [15906] 변신 이동 게임

BOJ15906: 변신 이동

문제 요약

- (1, 1)에서 (r, c)로 이동하는데 걸리는 시간
  - (1 <= r, c <= N <= 500)

- 변신이라는 특수 연산이 포함됨
  - 변신에는 t 시간이 소요됨 (0 <= t <= 500)
  - 가장 가까운 워프 장소 '#' 로 이동하는데 1턴 소모
  - 변신을 푸는데 0턴 소모

BOJ15906: 변신 이동

문제 요약

입력 데이터
3 0 3 3 ... ... #.#

출력 데이터
2

(1, 1) -> (3,1) -> (3, 3) 으로 변신 후 이동하면
cost: 2 로 이동

BOJ15906: 변신 이동

문제 분석

- 워프만 없다면 일반적인 4방향 탐색이다

- 워프를 어떻게 구현해야 할까?
  - 문제의 좌표 사이즈가 크지 않다
  - 현재 좌표에서 상/하/좌/우 이동하며, 워프 좌표를 찾는다
    - 맵의 끝에 도달하면 중단한다

BOJ15906: 변신 이동

문제 분석

- 워프 처리 주의사항
  - 연달아서 워프할 수 있다 (예제 2번)
  - 워프 좌표로 이동할 때,
    이미 변신상태일수도 있다 (중복 계산하면 안된다)

  - 워프 좌표는, 워프 없이 걸어갈 수도 있다

BOJ15906: 변신 이동

문제 분석

- 문제에서 요구하는 출력은 최소 턴 수이다
- 따라서 우선순위 큐를 턴 수를 기준으로 작성하여 다음 좌표들을 넣어두면?
- 적은 턴 수를 가진 좌표들을 우선적으로 처리할 수 있다
- 다익스트라와 유사하게 구현된다

BOJ15906: 변신 이동

구현 – 입력, 초기화

```
for(int i = 1; i <= n; i++) {
    String s = sc.next();
    for(int j = 1; j <= n; j++) {
        char ch = s.charAt(j - 1);
        if(ch == '#') map[i][j] = 1;
        else map[i][j] = 0;
        cost[0][i][j] = Integer.MAX_VALUE;
        cost[1][i][j] = Integer.MAX_VALUE;
    }
}
PriorityQueue<Point> pq = new PriorityQueue<>(
    (a, b) -> a.cost - b.cost
);
```

일반 좌표는 0
워프 좌표는 1로 처리

cost 기준으로 최소힙

BOJ15906: 변신 이동

구현 – 입력, 초기화

```
class Point {
    int r, c, changed, cost;
    Point (int r, int c, int changed, int cost) {
        this.r = r;
        this.c = c;
        this.changed = changed;
        this.cost = cost;
    }
}
```

좌표 정보

변신 상태 여부

해당 좌표까지 소모된 턴 수

BOJ15906: 변신 이동

구현 – 입력, 초기화

```
pq.add(new Point(1, 1, 0, 0));
cost[0][1][1] = 0;
while(!pq.isEmpty()) {
    Point now = pq.poll();
    if(now.r == gr && now.c == gc) {
        System.out.println(now.cost);
        return;
    }
    if(cost[now.changed][now.r][now.c] < now.cost) continue;
```

(1, 1)을 시작 좌표로

도착 좌표에 도달하면 종료

현재 좌표가 유망하지 않으면 스킵

BOJ15906: 변신 이동

구현 – 입력, 초기화

```
for(int i = 0; i < 4; i++) {
    int nr = now.r + dr[i];
    int nc = now.c + dc[i];
    if(!isValid(nr, nc, n)) continue;
    Point next = new Point(nr, nc, 0, now.cost + 1);
    if(cost[0][nr][nc] > next.cost) {
        cost[0][nr][nc] = next.cost;
        pq.add(next);
    }
}
```

일반 4방향 탐색

1턴으로 1칸이동

최적 값을 발견하면 갱신
다음 좌표로 활용

BOJ

구

```
for(int i = 0; i < 4; i++) {
    int nr = now.r + dr[i];
    int nc = now.c + dc[i];
    int nextCost = now.cost + t * (now.changed == 0 ? 1 : 0) + 1;
    while(true) {
        if(!isValid(nr, nc, n)) break;
        if(map[nr][nc] == 1) {
            if (cost[1][nr][nc] > nextCost) {
                cost[1][nr][nc] = nextCost;
                pq.add(new Point(nr, nc, 1, nextCost));
            }
            break;
        }
        nr += dr[i];
        nc += dc[i];
    }
}
```

워프 4방향 탐색

변신이 안 된 경우에만 턴수 소모

워프 좌표를 찾으면
계산한 코스트 정보로 이동
변신 상태는 유지됨

지정된 방향으로 이동하며
워프 좌표를 탐색