# 🚀 백준 20125 문제: 패스워드 검사 알고리즘 (Kotlin)

## 🧠 알고리즘 개념 및 핵심 로직

### 1️⃣ 심장 위치 찾기 (탐색 알고리즘)

- **문제 핵심:** 심장은 '*'이 있는 첫 번째 줄에서 바로 아래 줄의 같은 위치에 있습니다.
- **방법:** 2중 루프를 사용해 가장 먼저 나타나는 '*'의 바로 아래 좌표를 심장으로 정의합니다.

### 2️⃣ 신체 부위 측정 (DFS/BFS 필요 없음)

- **왼팔 & 오른팔 길이:** 심장의 좌우 방향으로 '*' 개수 카운트
- **허리 길이:** 심장 아래로 '*' 개수 카운트
- **다리 길이:** 허리 끝에서 좌우로 각각 아래 방향으로 '*' 개수 카운트

### 3️⃣ 좌표 처리 (주의할 점)

- 입력은 (1, 1)부터 시작하지만, 코틀린 리스트는 0-index 기반입니다.
- 결과 출력 시 **+1 보정** 필요.

---

## 🚀 코틀린 코드 (개선된 버전)

```
fun main() = with(System.`in`.bufferedReader()) {
    val N = readLine().toInt()
    val space = List(N) { readLine().toList() }
    var heart = Pair(0, 0)

    // 1️⃣ 심장 위치 찾기
    outer@ for (i in 0 until N) {
        for (j in 0 until N) {
            if (space[i][j] == '*') {
                heart = Pair(i + 1, j) // 심장은 머리의 바로 아래
                break@outer
            }
        }
    }

    val (hx, hy) = heart

    // 2️⃣ 각 신체 부위 길이 측정
    val leftArm = (hy - 1 downTo 0).count { space[hx][it] == '*' }
    val rightArm = (hy + 1 until N).count { space[hx][it] == '*' }
    val back = (hx + 1 until N).count { space[it][hy] == '*' }

    // 다리 위치 기준
    val leftLeg = (hx + back + 1 until N).count { space[it][hy - 1] == '*'
}
    val rightLeg = (hx + back + 1 until N).count { space[it][hy + 1] ==
'*' }
```

```
    // 3️⃣ 결과 출력
    println("${hx + 1} ${hy + 1}")
    println("$leftArm $rightArm $back $leftLeg $rightLeg")
}
```

---

## 🔑 코틀린 문법 포인트

1. **`outer@`라벨과 `break` 사용**

```
outer@ for (i in 0 until N) {
    for (j in 0 until N) {
        if (space[i][j] == '*') {
            heart = Pair(i + 1, j)
            break@outer // 바깥 루프 탈출
        }
    }
}
```

- 중첩된 루프를 한 번에 탈출하는 방법
- break@label을 사용하여 외부 루프까지 종료 가능

2. **리스트 생성 및 읽기**

```
val space = List(N) { readLine().toList() }
```

- `List(N)`과 `readLine().toList()`를 활용해 입력을 간결하게 처리
- 불변 리스트로 처리하면 코드가 더 안정적

3. **조건부 카운팅 with `count`**

```
val leftArm = (hy - 1 downTo 0).count { space[hx][it] == '*' }
val rightArm = (hy + 1 until N).count { space[hx][it] == '*' }
```

- count 함수는 조건에 맞는 요소의 개수를 바로 반환
- 루프를 간결하게 표현 가능 (가독성 향상)

---

## ✨ 선언형 스타일 (더 깔끔한 코드)

```
fun main() = with(System.`in`.bufferedReader()) {
    val N = readLine().toInt()
    val space = List(N) { readLine().toList() }
```

```
    val heart = space.withIndex().firstNotNullOf { (i, row) ->
        row.indexOf('*').takeIf { it != -1 }?.let { Pair(i + 1, it) }
    }

    val (hx, hy) = heart

    val leftArm = (hy - 1 downTo 0).takeWhile { space[hx][it] == '*'
}.count()
    val rightArm = (hy + 1 until N).takeWhile { space[hx][it] == '*'
}.count()
    val back = (hx + 1 until N).takeWhile { space[it][hy] == '*' }.count()
    val leftLeg = (hx + back + 1 until N).takeWhile { space[it][hy - 1] ==
'*' }.count()
    val rightLeg = (hx + back + 1 until N).takeWhile { space[it][hy + 1]
== '*' }.count()

    println("${hx + 1} ${hy + 1}")
    println("$leftArm $rightArm $back $leftLeg $rightLeg")
}
```

# 🎯 백준 1205번 문제: 랭킹 시스템 구현 (DJMAX 점수 랭킹 문제)

---

## 🚀 문제 개념

- **목표:** 태수의 새로운 점수가 기존 랭킹 리스트에 몇 등으로 들어갈 수 있는지 구하는 문제.
- **랭킹 규칙:**
    1. **비오름차순(내림차순) 정렬**된 리스트에서 높은 점수가 더 좋은 순위를 가짐.
    2. **같은 점수일 경우** 더 높은 순위를 가짐.
    3. **리스트가 꽉 차 있을 경우**, 태수의 점수가 기존 점수보다 높아야 랭킹에 들어갈 수 있음.

---

## 📋 입력 및 출력

### ✅ 입력

- 첫째 줄: `N` (현재 랭킹 리스트의 점수 개수), `score` (태수의 점수), `P` (랭킹 리스트 최대 크기)
- 둘째 줄 (선택적): `N`개의 점수 (내림차순 정렬됨)

### ✅ 출력

- 태수의 새로운 점수 순위
- 랭킹 리스트에 진입하지 못하면 `-1` 출력

---

## ⚡ 알고리즘 개념

### 1️⃣ 예외 처리 (Edge Case)

- **리스트가 비어있는 경우:** 무조건 1등
- **리스트가 꽉 찼고 태수의 점수가 가장 낮은 점수보다 작거나 같은 경우:** 랭킹 진입 불가 → `-1` 출력

---

### 2️⃣ 순위 결정 로직 (Ranking Logic)

- 리스트를 순회하면서 태수의 점수와 비교:
  - **점수가 더 크거나 같으면 해당 순위 결정**
  - **점수가 더 작으면 순위를 계속 증가**

---

### 3️⃣ 리스트가 꽉 찼을 경우 추가 처리

- **순위가 최대 크기 `P`를 초과하면** 랭킹에 진입 불가

---

## 💡 코틀린 코드 (명령형 스타일)

```
import java.io.*

fun main() = with(System.`in`.bufferedReader()) {
    val (N, score, P) = readLine().split(" ").map { it.toInt() }
    val rankingList = if (N > 0) readLine().split(" ").map { it.toInt() }
else listOf()

    if (N == 0) {
        println(1)
        return@with
    }

    if (N == P && score <= rankingList.last()) {
        if (score == rankingList.last()) {
            println(-1)
            return@with
        }
    }

    var rank = 1
    for (i in rankingList.indices) {
        if (score < rankingList[i]) {
            rank = i + 2
        } else {
            break
        }
    }

    if (N == P && rank > P) {
        println(-1)
    } else {
        println(rank)
    }
}
```

## 💡 선언형 스타일로 개선

```
import java.io.*

fun main() = with(System.`in`.bufferedReader()) {
    val (N, score, P) = readLine().split(" ").map { it.toInt() }
    val rankingList = if (N > 0) readLine().split(" ").map { it.toInt() }
else listOf()

    when {
        N == 0 -> println(1)  // 빈 리스트의 경우 무조건 1등
        N == P && score <= rankingList.last() && score ==
rankingList.last() -> println(-1)  // 진입 불가
        else -> {
            val rank = rankingList.indexOfFirst { score >= it }.let { if
(it == -1) N + 1 else it + 1 }
            if (N == P && rank > P) println(-1) else println(rank)
        }
    }
}
```

---

## 💡 코틀린 문법 포인트

### 1. `indexOfFirst` 함수

```
val rank = rankingList.indexOfFirst { score >= it }
```

- 조건을 만족하는 첫 번째 인덱스를 반환
- 만족하는 값이 없으면 `-1` 반환