🏃 백준 9017번 크로스 컨트리 문제 - Kotlin 코드 설명

📋 문제 개요

크로스 컨트리 대회에서 **우승 팀**을 결정하는 프로그램을 작성합니다.

- **팀 구성:** 최대 6명의 선수
- **점수 계산:** 상위 4명의 선수 등수 합산
- **동점 처리:** 5번째 선수의 등수가 더 낮은 팀이 우승

✅ 코드 전체 구조

```
fun main() = with(File("9017_input.txt").bufferedReader()) {
    val T = readLine().toInt()  // 테스트 케이스 수

    repeat(T) {
        val N = readLine().toInt()  // 참가한 선수 수
        val teamsList = readLine().split(" ").map { it.toInt() }  // 각 선수의 팀 번호

        val eligibleList = mutableListOf<Int>()  // 6명 이상인 팀 저장
        val resetList = mutableListOf<Int>()     // 점수 계산을 위한 리스트
        val result = mutableListOf<Pair<Int, MutableList<Int>>>() // 최종 결과 저장

        // 1️⃣ 참가 선수가 6명인 팀 찾기
        for (i in 1..N) {
            val test = teamsList.count { it == i }
            if (test == 6) {
                eligibleList.add(i)
            }
        }

        // 2️⃣ 6명 이상 팀의 선수만 추려서 저장
        for (j in teamsList.indices) {
            for (i in eligibleList) {
                if (teamsList[j] == i) resetList.add(i)
            }
        }

        // 3️⃣ 팀별로 선수의 등수 저장
        val eligibleMap = mutableMapOf<Int, MutableList<Int>>()
        resetList.forEachIndexed { index, team ->
            eligibleMap.getOrPut(team) { mutableListOf() }.add(index + 1)
        }

        // 4️⃣ 팀 점수 계산 (상위 4명의 점수 + 5번째 선수 점수)
        val teamScores = eligibleMap.map { (team: Int, scores:
```

```
MutableList<Int>) ->
            val top4Sum = scores.take(4).sum()  // 상위 4명의 점수 합
            val fifthScore = scores[4]          // 5번째 선수 점수
            Triple(team, top4Sum, fifthScore)   // (팀 번호, 총 점수, 5번째 선수
점수)
        }

        // 5️⃣ 최종 우승 팀 결정
        val winner = teamScores.minWith(compareBy({ it.second }, {
it.third }))!!.first

        println(winner)
    }
}
```

🚀 백준 1244번 스위치 켜고 끄기 문제 - Kotlin 코드 설명

📋 문제 개요

- **스위치:** 1은 켜진 상태, 0은 꺼진 상태
- **학생의 행동:**
    1. **남학생:** 받은 수의 배수 위치의 스위치 상태를 모두 반전
    2. **여학생:** 받은 수를 중심으로 좌우 대칭인 구간을 찾아 상태를 모두 반전
- **출력:** 스위치 상태를 한 줄에 최대 20개씩 출력

✅ 코드 전체 구조

```
fun main() = with(File("1244_input.txt").bufferedReader()) {
    val switches = readLine().toInt()                                  // 스위
치 개수
    val switchState = readLine().split(" ").map { it.toInt()
}.toMutableList()  // 스위치 상태
    val students = readLine().toInt()                                  // 학생
수

    switchState.add(0, 0)  // 인덱스 1부터 맞추기 위해 0 추가

    repeat(students) {
        val (sex, index) = readLine().split(" ").map { it.toInt() }
        var i = 0

        when (sex) {
            1 -> {  // 남학생: 받은 수의 배수 스위치 반전
                i++
                while (index * i <= switchState.size - 1) {
```

```
                        switchState[index * i] = if (switchState[index * i] ==
0) 1 else 0
                        i++
                    }
                }

                2 -> {  // 여학생: 대칭 구간 스위치 반전
                    while (index + i <= switchState.size - 1 && index - i >=
0) {
                        if (index == 1) {
                            switchState[1] = if (switchState[1] == 0) 1 else 0
                            break
                        } else if (switchState[index + i] == switchState[index
- i] && index - i != 0) {
                            val checker = switchState[index + i]
                            switchState[index + i] = if (checker == 0) 1 else
0
                            switchState[index - i] = if (checker == 0) 1 else
0
                            i++
                        } else {
                            break
                        }
                    }
                }
            }
        }

        switchState.remove(0)  // 인덱스 보정 제거
        switchState.chunked(20).forEach { chunk ->
            println(chunk.joinToString(" "))
        }
    }
```

💡 백준 17266번 가로등 설치 문제 - Kotlin 코드 설명

📋 문제 개요

- **목표:** 가로등의 높이를 최소화하여 굴다리의 모든 구간(0 ~ N)을 밝히는 것
- **조건:**
    1. 모든 가로등의 **높이는 동일**해야 하며, **정수**여야 함
    2. 가로등의 높이 H → 왼쪽으로 H, 오른쪽으로 H만큼 밝힘
    3. **최소 높이**를 찾아야 함

✅ 알고리즘 핵심 개념

1️⃣ 이진 탐색 (Binary Search)

- **이유:**
  - 최소 높이를 찾는 문제 → **최적화 문제**
  - `0`부터 `N`까지의 높이 중에서 최소 높이를 빠르게 찾기 위해 **이진 탐색** 사용
- **시간 복잡도:**
  - `O(log N)` (이진 탐색) + `O(M)` (각 높이 검증)
    → **총합:** `O(M * log N)`

✅ 코드 설명

```
fun main() = with(File("17266_input.txt").bufferedReader()) {
    val N = readLine().toInt()              // 굴다리 길이
    val M = readLine().toInt()              // 가로등 개수
    val positions = readLine().split(" ").map { it.toInt() } // 가로등 위치 리스트

    var left = 0
    var right = N
    var result = N

   // 가로등 높이로 굴다리 전체를 비출 수 있는지 확인하는 함수
    fun canLightAll(H: Int): Boolean {
        var prev = 0

        // 1️⃣ 첫 번째 가로등이 시작부터 커버 가능한지 확인
        if (positions[0] - H > 0) return false

        // 2️⃣ 가로등들로 굴다리의 모든 구간이 커버되는지 확인
        for (pos in positions) {
            if (prev < pos - H) return false  // 어두운 구간 발생 시 불가능
            prev = pos + H                    // 다음 가로등이 비출 수 있는 마지막 위치
        }

        // 3️⃣ 굴다리의 끝(N)까지 밝힐 수 있는지 확인
        return prev >= N
    }

    // ✅ 이진 탐색 수행
    while (left <= right) {
        val mid = (left + right) / 2

        if (canLightAll(mid)) {
            result = mid          // 현재 높이로 커버 가능하면 최소 높이 갱신
            right = mid - 1       // 더 작은 높이 시도
        } else {
            left = mid + 1        // 더 큰 높이 필요
        }
    }
```

```
    println(result)                // 최소 높이 출력
}
```

커버 여부 확인 함수: `canLightAll(H)`

```
fun canLightAll(H: Int): Boolean {
    var prev = 0

    if (positions[0] - H > 0) return false

    for (pos in positions) {
        if (prev < pos - H) return false
        prev = pos + H
    }

    return prev >= N
}
```

작동 원리

1. 시작점 커버 확인:
    - `positions[0] - H > 0` -> 시작점(0)을 밝히지 못하면 `false`반환
2. 어두운 구간 체크
    - 이전 가로등(`prev`)이 현재 가로등의 왼쪽 끝(`pos-H`)보다 작다면 -> 어두운 구간 발생
3. 끝점 커버 확인
    - 마지막 가로등이 굴다리 끝(`N`)까지 커버 가능한지 확인

🚗 백준13305번 주유소 문제 - Kotlin 코드 설명

📋 문제 개요

- **목표:** 제일 왼쪽 도시에서 오른쪽 도시로 이동할 때, **최소 비용으로 주유**하기
- **조건:**
    1. **처음 출발 시 반드시 주유 후 출발**
    2. **1km당 1L의 기름 소모**
    3. 각 도시에는 하나의 주유소가 있으며, **리터당 가격이 다름**
    4. **기름통의 용량은 무제한**

✅ 알고리즘 핵심 개념

1️⃣ 그리디 알고리즘 (Greedy Algorithm)

- **핵심 아이디어:**
  현재까지의 최소 기름값으로 최대한 주유하는 것이 최적의 해법이다.
  - 기름값이 더 저렴한 도시를 만날 때까지는 현재 도시에서 필요한 만큼 주유
  - 더 저렴한 가격의 도시가 나오면 그곳에서 다시 주유

✅ Kotlin 코드 구현

```kotlin
fun main() = with(File("13305_input.txt").bufferedReader()) {
    val N = readLine().toInt()
    val distances = readLine().split(" ").map { it.toLong() } // 도로 길이
    val prices = readLine().split(" ").map { it.toLong() }    // 주유소 가격

    var minPrice = prices[0] // 초기 최소 기름값은 첫 번째 도시의 가격
    var totalCost = 0L       // 총 비용

    for (i in 0 until N - 1) {
        // 현재 최소 가격으로 주유
        totalCost += minPrice * distances[i]

        // 더 싼 기름 가격이 나오면 갱신
        if (prices[i + 1] < minPrice) {
            minPrice = prices[i + 1]
        }
    }

    println(totalCost)
}
```

📚 백준 20920 - **영단어 암기는 괴로워** (Kotlin 풀이)

🚀 문제 설명

- **목표:** 주어진 단어들을 특정 우선순위에 따라 정렬하여 출력하기
- **우선순위:**
  1. **자주 나오는 단어**일수록 앞에 배치
  2. **단어의 길이**가 길수록 앞에 배치
  3. **알파벳 사전 순**으로 앞에 있는 단어일수록 앞에 배치
- **조건:** 단어의 길이는 `M` 이상이어야 처리

💡 핵심 아이디어 (알고리즘 개념)

1. **단어 빈도수 계산:**
   `Map`을 활용하여 각 단어의 출현 빈도수 기록

2. **조건에 따른 정렬:**
   `sortedWith`와 `compareByDescending`을 이용해 복합 조건 정렬
3. **출력 최적화:**
   `StringBuilder`로 빠른 결과 출력 처리

✅ Kotlin 코드 구현

```kotlin
import java.io.*

fun main() = with(File("20920_input.txt").bufferedReader()) {
    val (N, M) = readLine().split(" ").map { it.toInt() }

    val dictMap = mutableMapOf<String, Int>()
    val result = StringBuilder()

    repeat(N) {
        val word = readLine()
        if (word.length >= M) {
            dictMap[word] = dictMap.getOrPut(word) { 0 } + 1
        }
    }

    dictMap.toList()
        .sortedWith(compareByDescending<Pair<String, Int>> { it.second }
// 1️⃣ 빈도수 기준 내림차순
            .thenByDescending { it.first.length }                    //
2️⃣ 단어 길이 기준 내림차순
            .thenBy { it.first })                                    //
3️⃣ 알파벳 사전 순 (오름차순)
        .toMap(LinkedHashMap())                                      //
순서 유지
        .forEach { result.append(it.key).append('\n') }

    println(result)
}
```