# Problem 3283: Maximum Number of Moves to Kill All Pawns

## Problem Information

**Difficulty:** Hard
**Acceptance Rate:** 33.25%
**Paid Only:** No
**Tags:** Array, Math, Bit Manipulation, Breadth-First Search, Game Theory, Bitmask

## Problem Description

There is a `50 x 50` chessboard with **one** knight and some pawns on it. You are given two integers `kx` and `ky` where `(kx, ky)` denotes the position of the knight, and a 2D array `positions` where `positions[i] = [xi, yi]` denotes the position of the pawns on the chessboard.

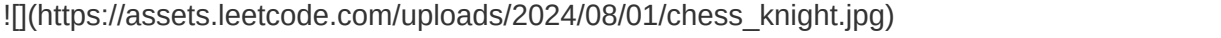
Alice and Bob play a _turn-based_ game, where Alice goes first. In each player's turn:

* The player _selects_ a pawn that still exists on the board and captures it with the knight in the **fewest** possible **moves**. **Note** that the player can select **any** pawn, it **might not** be one that can be captured in the **least** number of moves. * In the process of capturing the _selected_ pawn, the knight **may** pass other pawns **without** capturing them. **Only** the _selected_ pawn can be captured in _this_ turn.

Alice is trying to **maximize** the **sum** of the number of moves made by _both_ players until there are no more pawns on the board, whereas Bob tries to **minimize** them.

Return the **maximum** _total_ number of moves made during the game that Alice can achieve, assuming both players play **optimally**.

Note that in one **move,** a chess knight has eight possible positions it can move to, as illustrated below. Each move is two cells in a cardinal direction, then one cell in an orthogonal direction.

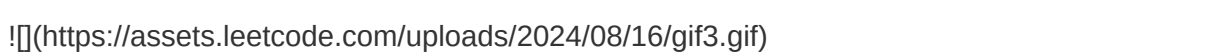



**Example 1:**

**Input:** kx = 1, ky = 1, positions = [[0,0]]

**Output:** 4

**Explanation:**



The knight takes 4 moves to reach the pawn at `(0, 0)`.

**Example 2:**

**Input:** kx = 0, ky = 2, positions = [[1,1],[2,2],[3,3]]

**Output:** 8

**Explanation:**

****

* Alice picks the pawn at `(2, 2)` and captures it in two moves: `(0, 2) -> (1, 4) -> (2, 2)`. * Bob picks the pawn at `(3, 3)` and captures it in two moves: `(2, 2) -> (4, 1) -> (3, 3)`. * Alice picks the pawn at `(1, 1)` and captures it in four moves: `(3, 3) -> (4, 1) -> (2, 2) -> (0, 3) -> (1, 1)`.

**Example 3:**

**Input:** kx = 0, ky = 0, positions = [[1,2],[2,4]]

**Output:** 3

**Explanation:**

* Alice picks the pawn at `(2, 4)` and captures it in two moves: `(0, 0) -> (1, 2) -> (2, 4)`. Note that the pawn at `(1, 2)` is not captured. * Bob picks the pawn at `(1, 2)` and captures it in one move: `(2, 4) -> (1, 2)`.

**Constraints:**

* `0 <= kx, ky <= 49` * `1 <= positions.length <= 15` * `positions[i].length == 2` * `0 <= positions[i][0], positions[i][1] <= 49` * All `positions[i]` are unique. * The input is generated such that `positions[i] != [kx, ky]` for all `0 <= i < positions.length`.

## Code Snippets

**C++:**

```
class Solution {
public:
int maxMoves(int kx, int ky, vector<vector<int>>& positions) {

}
};
```

**Java:**

```
class Solution {
public int maxMoves(int kx, int ky, int[][] positions) {

}
}
```

**Python3:**

```
class Solution:
def maxMoves(self, kx: int, ky: int, positions: List[List[int]]) -> int:
```