Knight’s Path

Description

Given an $n \times n$ chess board, a starting position, and ending position, and a bishop position, compute the shortest path a knight starting at the start position and ending at the end position can take without being threatened by the bishop. If the knight can capture the bishop, then it is free to take any path.

Formally: given an integer pair (x_0, y_0) starting position, an integer pair (x_1, y_1) ending position, and an integer pair (p, q) bishop position, compute the shortest path from the start to the end without being threatened by the bishop, returning -1 if no path exists.

Notes

A knight can move in an L-shape. That is, given some position (p, q) on the board, the knight can move to $(p+1, q+2)$, $(p+2, q+1)$, $(p-1, q+2)$, etc. Here is a picture for better visualization.

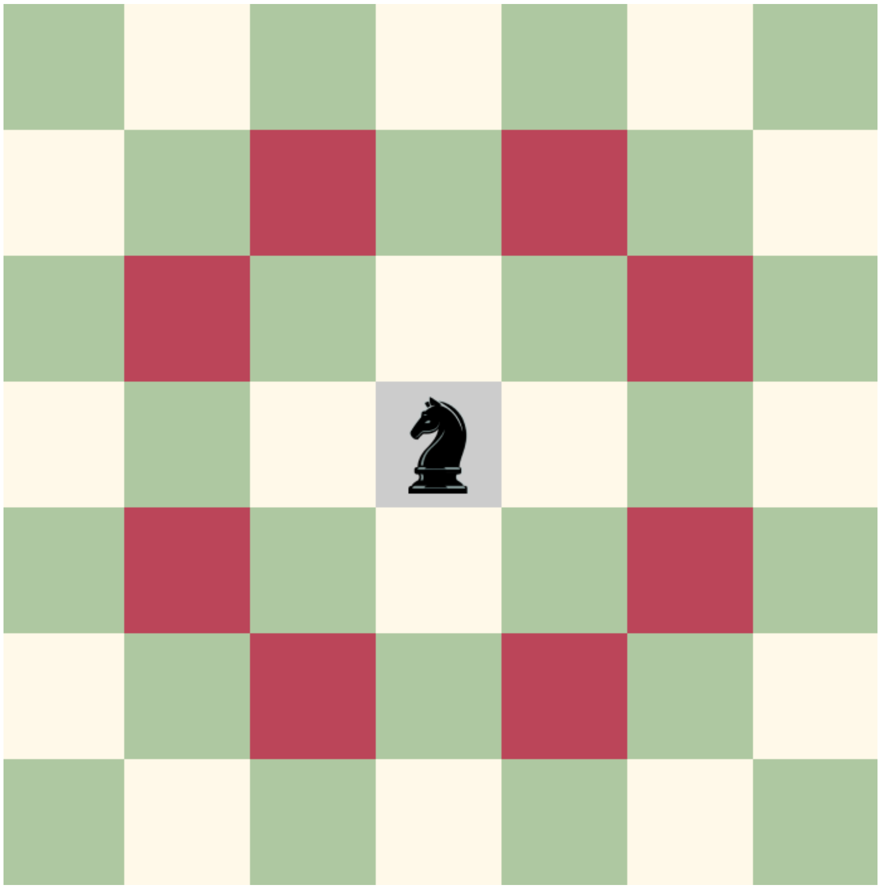

Figure 1: A knight’s possible moves

The shortest path is one with the fewest of such moves. Now, the bishop moves on diagonals. That is, given a position (p, q) on the board, the bishop threatens any position diagonal from (p, q).

Solution

Idea: Treat the chessboard as a graph, and use a known graph traversal algorithm (BFS) to solve!

1. Set up functions to construct nodes, check their validity, and create a list of nodes that the bishop threatens
2. Create a BFS function that accounts for the possible threat of the bishop
3. Wire up our constructions to solve the problem
4. Optimize if time remains

Set Up

```
procedure MAKE-NODE($x, y, d$) return $(x, y, d)$    ▷ A Node is a 3-Tuple with an x coordinate, y coordinate, and integer distance
procedure IS-VALID($x, y, n$)
    if $x < 1$ or $y < 1$ or $x > n$ or $y > n$ then return false
    return true
procedure IS-VALID-BISHOP($x, y, A$)
    if $(x, y)$ in $A$ then return false
    return true
procedure BISHOP-POSITIONS($x, y, n$)
    $A \leftarrow \emptyset$
    for $i \in \{1, 2, \ldots, n\}$ do
        if IS-VALID($x + i, y + i, n$) then
            $A.add(x, y)$
        if IS-VALID($x - i, y + i, n$) then
            $A.add(x, y)$
        if IS-VALID($x + i, y - i, n$) then
            $A.add(x, y)$
        if IS-VALID($x - i, y - i, n$) then
            $A.add(x, y)$
    return $A$
```

Modified BFS

```
procedure SHORTEST-PATH($x_0, y_0, x_1, y_1, n, b, A$) ▷ $x_0, y_0$ is start position, $x_1, y_1$ is end position, $n$ is dimension, and $b$ is a Boolean for if the bishop exists
    $row \leftarrow \{2, 2, -2, -2, 1, 1, -1, -1\}$
    $col \leftarrow \{-1, 1, 1, -1, 2, -2, 2, -2\}$
    $visited \leftarrow \emptyset$
    Queue $Q \leftarrow \{$MAKE-NODE$(x_0, y_0, 0)\}$
    while $Q \neq \emptyset$ do
        $v \leftarrow Q.pop$
        if $v.x = x_1$ and $v.y = y_1$ then return $v.d$
        if $v \notin visited$ then
            $visited.add(v)$
            for $i \in \{1, 2, \dots, 8\}$ do
                $x_{new} \leftarrow v.x + col[i]$
                $y_{new} \leftarrow v.y + row[i]$
                if $b = true$ then
                    if IS-VALID($x_{new}, y_{new}, n$) and IS-VALID-BISHOP($x_{new}, y_{new}, A$) then
                        $Q.add($MAKE-NODE$(x_{new}, y_{new}, v.d + 1))$
                else
                    if IS-VALID($x_{new}, y_{new}, n$) then
                        $Q.add($MAKE-NODE$(x_{new}, y_{new}, v.d + 1))$
    return -1
```

Final

```
procedure KNIGHT'S-PATH($x_0, y_0, x_1, y_1, b_x, b_y, n$)
    $A \leftarrow$ BISHOP-POSITIONS($b_x, b_y, n$)
    to-bishop $\leftarrow$ SHORTEST-PATH($x_0, y_0, b_x, b_y, n, true, A$)
    to-goal $\leftarrow$ SHORTEST-PATH($x_0, y_0, x_1, y_1, n, true, A$)
    if to-bishop $\neq -1$ then
        b-to-goal $\leftarrow$ SHORTEST-PATH($b_x, b_y, x_1, y_1, n, false, A$)
        if b-to-goal $\neq -1$ then
            if to-goal $\neq -1$ then return min(to-bishop + b-to-goal, to-goal)
            else return to-bishop + b-to-goal
        else return to-goal
    else return to-goal
```

Optimizations

While this solution should be able to handle pretty large instances of the problem, what if we get a really, really big one? If the start and end are sufficiently far away, our algorithm may take too long to find a solution (at least, too long for hackerrank). How can we optimize?
Idea: Run a modified BFS from either end, and see if/where they meet!