Liam Murphy

Lecture Outline

Outline

- Start with general 2-3 minute talk about how the abstract algorithms we learn have pretty important uses, even for students (not just researchers)
- Introduce the idea of a technical interview: explain how the skill the companies are looking for is not coding, it's knowledge of algorithms and data structures
- Introduce the problem that we will be studying: the Knight's Path
- (be sure to mention that this is totally *not* a problem from an actual interview)
- Explain that while this big description of a problem seems scary at first glance, it's actually a fairly simple application of a common algorithm we saw in class, the BFS
- Show how to translate the problem from words into a graph problem
- Solve the problem (important: show pseudocode for it then maybe show a working python implementation ?? if I have time to code that up this week)
- Closing remarks about how the hard part is translating the problem from words to an algorithmic problem that is easy to solve

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.

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.

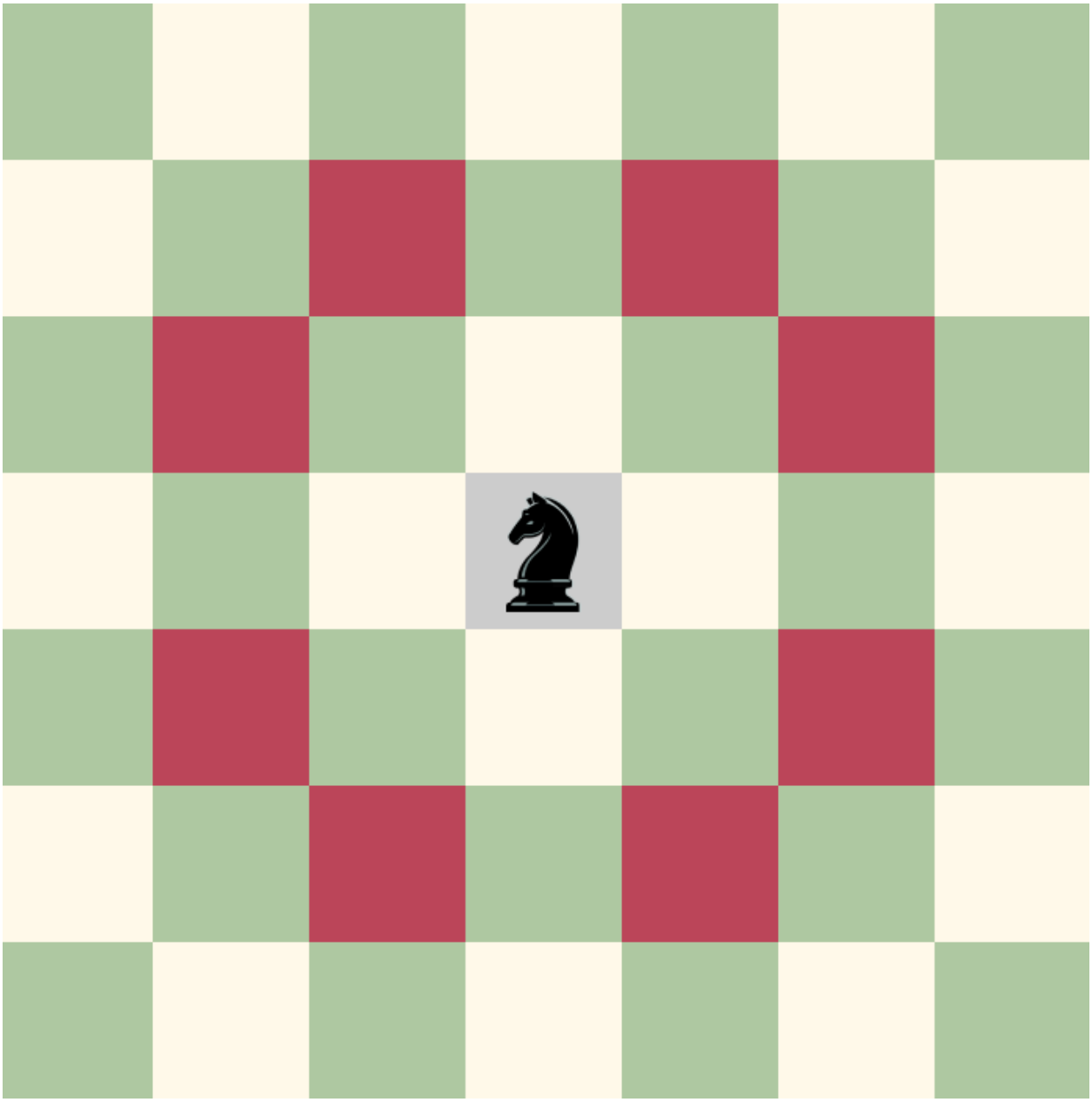

Figure 1: A knight's possible moves

The shortest path refers to the path that costs the knight the least amount of moves. Now, the bishop moves in a diagonal pattern, so when designing our algorithm, we cannot land on any spaces that are diagonal from the bishop, unless we capture (land on top of it) first. With the details now in mind, we're ready to start.

Solution

We want to design an algorithm that treats the chessboard as a graph, then use BFS to find the shortest path (with a few modifications, of course). However, we can't just make the entire chessboard, a graph, because the knight can't move to just any space! Here's the idea: use BFS to determine how many moves it would take to reach the bishop, capture, then move to the ending position (or if this is even possible) and compare it with just avoiding the bishop's threat and going straight to the end position (if this is even possible). Return the length of the shorter path, if either answer reaches the end space.

```
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 ← ∅
    for i ∈ {1, 2, ..., 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
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 ← {2, 2, −2, −2, 1, 1, −1, −1}
    col ← {−1, 1, 1, −1, 2, −2, 2, −2}
    visited ← ∅
    Queue Q ← {MAKE-NODE(x_0, y_0, 0)}
    while Q ≠ ∅ do
        v ← Q.pop
        if v.x = x_1 and v.y = y_1 then return v.d
        if v ∉ visited then
            visited.add(v)
            for i ∈ {1, 2, ..., 8} do
                x_new ← v.x + col[i]
                y_new ← 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
procedure KNIGHT'S-PATH(x_0, y_0, x_1, y_1, b_x, b_y, n)
    A ← BISHOP-POSITIONS(b_x, b_y, n)
    to-bishop ← SHORTEST-PATH(x_0, y_0, b_x, b_y, n, true, A)
    to-goal ← SHORTEST-PATH(x_0, y_0, x_1, y_1, n, true, A)
    if to-bishop ≠ −1 then
        b-to-goal ← SHORTEST-PATH(b_x, b_y, x_1, y_1, n, false, A)
        if b-to-goal ≠ −1 then
            if to-goal ≠ −1 then return min(to-bishop + b-to-goal, to-goal)
            elsereturn to-bishop + b-to-goal
        elsereturn to-goal
    elsereturn to-goal
```