

Knight Moves problem

The **Knight Moves problem** involves finding the minimum number of moves a knight needs to go from a starting position to a target position on a chessboard or 2D grid. Since the knight can move in **8 possible L-shaped directions**, BFS is used to explore all positions level by level. Each cell on the grid is treated as a node, and BFS ensures the shortest number of moves is found because it explores all positions reachable in the current number of moves before moving to positions that require more moves. This method works for any board size and is guaranteed to find the minimum moves if a solution exists.

Advantages

- Finds **minimum moves efficiently**.
- Works for **any 2D grid size**.
- BFS ensures **shortest path in terms of moves**.
- Can be adapted for obstacles or restricted grids.

Limitations

- Uses **extra memory** for the queue and visited array.
- Slower for **very large grids**.
- Cannot handle **weighted moves** (all moves are assumed equal cost).

Steps for Knight Moves Using BFS

1. Initialize

- Create a 2D array **visited** to mark visited cells.
- Create a queue to store positions and moves (**x, y, distance**).

2. Start from the source cell

- Mark the starting position as visited and enqueue it with distance 0.

3. Perform BFS

- While the queue is not empty:
 - Dequeue the current cell ($x, y, distance$).
 - If it's the target, return $distance$.
 - For each of the 8 knight moves:
 - Compute the new position (nx, ny).
 - If (nx, ny) is inside the grid and unvisited:
 - Mark it visited.
 - Enqueue ($nx, ny, distance + 1$).

4. Repeat

- Continue until the queue is empty.
- If the target is never reached, return -1 (unreachable).

Input:

e2 e4

a1 b2

b2 c3

a1 h8

a1 h7

h8 a1

b1 c3

f6 f6

Execution:

Input: e2 e4

1. Read Input: start = "e2", end = "e4"
2. Convert Coordinates:
 - o e2 → column 'e' = 4, row '2' = 1 → (4,1) o e4 → column 'e' = 4, row '4' = 3 → (4,3)
3. Initialize BFS:
 - o Create visited[8][8] = all false o Queue = [(4,1,0)] o visited[4][1] = true
4. Process Queue:
 - o Dequeue (4,1,0)
 - o Generate 8 knight moves: (6,2), (6,0), (5,3), (5,-1), (3,3), (3,-1), (2,2), (2,0) o Valid moves (within board): (6,2), (6,0), (5,3), (3,3), (2,2), (2,0)
 - o Enqueue all valid moves with moves=1
5. Continue BFS:
 - o Dequeue (6,2,1)
 - o Generate moves: (8,3), (8,1), (7,4), (7,0), (5,4), (5,0), (4,3), (4,1) o Found destination (4,3) → return 1 + 1 = 2
6. Output: "To get from e2 to e4 takes 2 knight moves."

Input: a1 b2

1. Read Input: start = "a1", end = "b2"
2. Convert Coordinates: a1 → (0,0), b2 → (1,1)

3. BFS Search: Explores multiple paths through intermediate squares

4. Find Path: $(0,0) \rightarrow (2,1) \rightarrow (1,3) \rightarrow (3,2) \rightarrow (1,1)$ = 4 moves

5. Output: "To get from a1 to b2 takes 4 knight moves."

Input: b2 c3

1. Read Input: start = "b2", end = "c3"

2. Convert Coordinates: b2 → (1,1), c3 → (2,2)

3. BFS Search: Finds path through one intermediate square

4. Find Path: $(1,1) \rightarrow (2,3) \rightarrow (2,2)$ = 2 moves

5. Output: "To get from b2 to c3 takes 2 knight moves."

Input: a1 h8

1. Read Input: start = "a1", end = "h8"

2. Convert Coordinates: a1 → (0,0), h8 → (7,7)

3. BFS Search: Explores longest diagonal path

4. Find Path: Multiple 6-move paths exist across board

5. Output: "To get from a1 to h8 takes 6 knight moves."

Input: a1 h7

1. Read Input: start = "a1", end = "h7"

2. Convert Coordinates: a1 → (0,0), h7 → (7,6)

3. BFS Search: Slightly shorter than h8 path

4. Find Path: Optimal path found in 5 moves

5. Output: "To get from a1 to h7 takes 5 knight moves."

Input: h8 a1

1. Read Input: start = "h8", end = "a1"

2. Convert Coordinates: h8 → (7,7), a1 → (0,0)

3. BFS Search: Reverse of previous path
4. Find Path: Same 6 moves as forward direction
5. Output: "To get from h8 to a1 takes 6 knight moves."

Input: b1 c3

1. Read Input: start = "b1", end = "c3"
2. Convert Coordinates: b1 → (1,0), c3 → (2,2)
3. BFS Search: Direct L-shaped move available
4. Find Path: (1,0)→(2,2) = 1 move
5. Output: "To get from b1 to c3 takes 1 knight moves."

Input: f6 f6

1. Read Input: start = "f6", end = "f6"
2. Convert Coordinates: f6 → (5,5), f6 → (5,5)
3. Special Case: Same start and end position
4. Immediate Return: 0 moves without BFS
5. Output: "To get from f6 to f6 takes 0 knight moves."

Output:

To get from e2 to e4 takes 2 knight moves.

To get from a1 to b2 takes 4 knight moves.

To get from b2 to c3 takes 2 knight moves.

To get from a1 to h8 takes 6 knight moves.

To get from a1 to h7 takes 5 knight moves.

To get from h8 to a1 takes 6 knight moves.

To get from b1 to c3 takes 1 knight moves.

To get from f6 to f6 takes 0 knight moves.

Pseudocode for Knight Moves BFS

KnightMovesBFS(start, target, N, M):

 create visited[N][M], initialize all to false

 create queue Q

 enqueue (start.x, start.y, 0)

 mark visited[start.x][start.y] = true

 while Q is not empty:

 (x, y, dist) = dequeue Q

 if (x, y) == target:

 return dist

 for each move (dx, dy) in [(2,1),(1,2),(-1,2),(-2,1),
 (-2,-1),(-1,-2),(1,-2),(2,-1)]:

 nx = x + dx

 ny = y + dy

 if nx in [0,N) and ny in [0,M) and not visited[nx][ny]:

 mark visited[nx][ny] = true

 enqueue (nx, ny, dist + 1)

 return -1

Time Complexity

- $O(N \times M)$
 - Each cell is visited **at most once**, and each has up to 8 neighbors.
- **Space Complexity:** $O(N \times M)$ for the visited array and BFS queue.

Code:

https://github.com/shanto470/algorith_plm/blob/main/BFS/Knight%20moves/knightMovers.cpp