Problem 2 : KnightMoves

Problem Link:

https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=380

What is the problem?

You are given two squares on a chessboard. A knight must move from the start square to the target square using the minimum number of moves.

This is a shortest path problem on an unweighted graph:

- **Nodes:** 64 squares of the chessboard
- **Edges:** A valid knight move between two squares

Because every move costs the same (1 step), BFS (Breadth-First Search) is the best algorithm.

Problem Statement :

The goal is to find the **minimum number of moves** a knight needs to travel from one square to another on a standard **8×8 chessboard**. The input provides two positions in algebraic notation (e.g., a1, h8). From the starting square, the knight can move in its usual **L-shaped** pattern, giving it up to **eight possible moves**.

Since each knight move has the same cost, the chessboard can be treated as an **unweighted graph** where each square is a node and each legal knight move is an edge. To determine the shortest path from the start to the destination, we apply **Breadth-First Search (BFS)**, which explores squares level by level.

The output is a single number: the **minimum number of knight moves** required to reach the target square. If the start and end squares are the same, the answer is zero.

2) Hint (Your hint for solving the problem):

- A chessboard can be modeled as a graph with 64 nodes.
- A knight's legal move creates an edge between two squares.
- Since every knight move has the same "cost", this becomes a shortest path problem in an unweighted graph.
- The best algorithm for shortest path in an unweighted graph is **Breadth-First Search (BFS)**.
- Using BFS ensures that the first time you reach the target square, you have found the minimum number of moves.

So the key hint is:

Use BFS from the starting square and explore all 8 knight moves until you reach the destination.

3) Solution Approach (Your approach for solving the problem):

Step 1: Represent the chessboard

- Consider the chessboard as an 8×8 grid.
- Each square (r, c) is treated as a **node**.
- Knight moves are the **edges** connecting these nodes.

Step 2: Define knight's 8 possible moves

A knight can move using the following coordinate changes:

(+2, +1), (+2, -1), (-2, +1), (-2, -1),

(+1, +2), (+1, -2), (-1, +2), (-1, -2)

These are stored in two arrays kr[] and kc[] in the code.

Step 3: Set up BFS

- Create a 2D array dist[8][8] to store minimum moves to reach each square.
- Create a color[8][8] or visited[8][8] array to track explored squares.
- Initialize all distances as infinity and visited states as unvisited.

Step 4: BFS traversal

- Put the starting square into a queue and set its distance = 0.
- Repeatedly pop the front of the queue, and generate all valid 8 knight moves.
- If a move leads to a valid unvisited square, push it into the queue and update its distance.
- Continue until the target square is reached.

Step 5: Return result

- The moment BFS reaches the target square, the distance stored is the **minimum number of moves**.
- BFS guarantees correctness because it explores the board level by level.

PSEUDOCODE:

```
FUNCTION bfs(startRow, startCol, endRow, endCol):
    CREATE 8x8 array dist and fill with -1
    CREATE empty queue Q
    SET dist[startRow][startCol] = 0
    PUSH (startRow, startCol) into Q
    WHILE Q is not empty:
        (r, c) = Q.pop()
        IF r = endRow AND c = endCol:
            RETURN dist[r][c]
        FOR i FROM 0 TO 7:
            newR = r + dr[i]
            newC = c + dc[i]
            IF newR and newC are inside board AND dist[newR][newC] = -1:
                dist[newR][newC] = dist[r][c] + 1
                PUSH (newR, newC) into Q
    RETURN -1
```

Implementation Link:

https://github.com/TasnimaSultana/Algo_Lab_Final/blob/main/BFS/KnightMoves/knightmoves.cpp