Problem2: 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 positions on a standard **8×8 chessboard**.
A knight starts at the first position and must reach the second position.

Your task is to compute the **minimum number of knight moves** needed to go from the start square to the end square.

The chessboard uses algebraic notation:

- Example: "a1", "h8", "c3"

Each square corresponds to coordinates on an 8×8 grid.

This is a **shortest path problem**, solved using **Breadth-First Search (BFS)**.

Problem Statement

Given two positions on a chessboard, determine the fewest number of moves a knight requires to travel from the starting square to the ending square.

You will repeatedly receive pairs of positions until EOF.
For each pair, output:

To get from X to Y takes N knight moves.

Hint

- Use **BFS**, because the knight moves through an **unweighted graph**.
- Convert positions like "a1" to integer coordinates (x, y).
  - Column: 'a' → 0, 'b' → 1, … 'h' → 7
  - Row: '1' → 0, '2' → 1, … '8' → 7
- Knight has **8 possible moves**:
- (+2,+1), (+2,-1), (-2,+1), (-2,-1),
- (+1,+2), (+1,-2), (-1,+2), (-1,-2)
- Use a **distance array** or visited array to ensure each square is processed once.
- BFS guarantees you find the **minimum number of moves**.

Solution Approach (Step-by-Step)

1. Read input positions start, end
2. Convert both squares into coordinates (sx, sy) and (ex, ey)
3. Set up a BFS queue
4. Create a 2D array dist[8][8] initialized to -1
5. Push the starting cell with distance 0
6. While queue not empty:
    - Pop a cell
    - If it's the target: answer found
    - Try all 8 knight moves
    - If inside board and unvisited, push into queue with distance +1
7. Print the distance of the target square

Pseudocode

```
function bfs(start, end):
   convert start to (sx, sy)
   convert end to (ex, ey)

   create dist[8][8], fill with -1
   queue Q
   Q.push((sx, sy))
   dist[sx][sy] = 0

   knight_moves = [
      (2,1), (2,-1), (-2,1), (-2,-1),
      (1,2), (1,-2), (-1,2), (-1,-2)
   ]

   while Q not empty:
      (x, y) = Q.pop()
      if x == ex and y == ey:
         return dist[x][y]

      for each move (dx, dy) in knight_moves:
         nx = x + dx
         ny = y + dy
         if nx, ny inside board and dist[nx][ny] == -1:
            dist[nx][ny] = dist[x][y] + 1
            Q.push((nx, ny))
```

Implementation Link :

algorithm-/BFS/KnightMoves/knight.cpp at main · Jannat651/algorithm-