Tiles

A

B

C

D

E

F

G

H

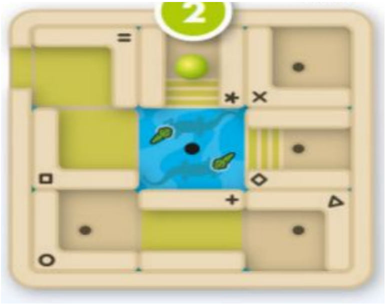
Input Format

1. First line: `T` (number of test cases).
2. Next `T` lines: One 20-character string per test case.

Test Case String (20 Chars)

- **Board (Characters 0-17):**
  - The first 18 characters define the 3x3 board.
  - Read as 9 two-character pairs in row-major order (top-left to bottom-right).
  - Orientation is from 0 to 3, with each number representing degree of clockwise rotation divided by 90 modulo 4.
  - `T0`: Tile `T` with orientation `0`.
  - `--`: Empty tile.
- **Pawn (Characters 18-19):**
  - The last 2 characters, `rc`, are the pawn's 0-indexed starting position.
  - `r` is the row (0-2), `c` is the column (0-2).

Example:



1
A2E3G3B0--D0H2C1F201

Core Logic: A* Search

The solver finds the optimal solution by implementing the A* search algorithm.

- **State (`GameState`)**: A "state" in the search is a complete snapshot of the puzzle, including:
  - The 3x3 board configuration (`BoardConfig`).
  - The pawn's exact position (row, column) and floor (top/ground).
  - The position of the blank (empty) tile.
  - The cost to reach this state (`cost_so_far`).
  - The estimated cost to the goal (`heuristic_cost`).
- **Transitions (Moves)**: From any state, the solver generates all possible next states:
  - **Tile Slides (`find_tile_slides`)**: Generates up to 4 new states by swapping the blank tile with an adjacent tile (unless the pawn is on it). Each slide has a cost of `1`.
  - **Pawn Moves (`find_pawn_moves`)**: This is a complex move. The solver runs a Breadth-First Search (BFS) starting from the pawn's current position to find *all reachable "resting spots"* (tiles with holes or the final exit) in a single turn. A new state is generated for each reachable spot. The cost is the number of steps in the BFS.
- **Heuristic (`calculate_heuristic`)**: To guide the A* search, a heuristic function estimates the remaining cost. It uses:
  - The Manhattan distance of the pawn to the exit `(0, -1)`.
  - Penalties for the blank tile being in a "bad" position (e.g., at `(0,0)`, blocking the exit).
  - Penalty for the tile on `(0,0)`, not having a open top on left.
- **Optimizations (`CompactState`)**: To avoid re-exploring states, the program uses a `min_cost` hash map. A full `GameState` object is too large to hash efficiently, so it's converted to a `CompactState` (using a custom hash of the board and pawn/blank positions) which serves as a fast, unique key. **Hashing and using std::unordered_map**, instead of std::map(which is a red black tree) for visited state reduced the time taken, by a factor of 10-12. Using a 1d array to store board also reduced the time taken by factor of 5. Combining many such small optimizations **reduced the time taken from 250 seconds to 2 seconds** on the toughest test case I had. Now the factor slowing down the code is priority queue, which stores game states, that can be optimized by using some encoded representation of the state.

Running Guide

Add number of test cases and your test cases to the `input.txt` file (one 20-character string per line).

Compile the program:

g++ -O3 -march=native -std=c++23 solver.cpp -o solver

Run the solver:
./solver

Check the results:

`output.txt`: Contains the full, step-by-step solution path.

`error.txt`: Shows the total states explored and the final run time.

Performance & Final Words

- The code efficiently solves even the most complex test cases found online in **under 3 seconds**.
- The heuristic is an *admissible* , which is crucial for A*'s optimality.
    - **Tough Cases:** On test cases with high-cost solutions, the heuristic provides a ~5% reduction in explored states, which is not much because the heuristic estimates are always in single digits compared to the actual costs in 2-3 digit numbers. If there are test cases which have optimal costs in range of 1000s, then I would recommend removing the heuristic usage, as then it would only lead to extra computations.
    - **Easy Cases:** On test cases with elegant or short solutions, the heuristic is highly effective, reducing the number of explored states by **30-60%** almost every time.