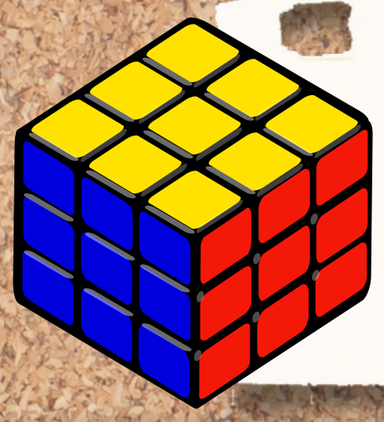
Parallel

RUBIK'S CUBE SOLVER



How the cube moves

Each possible move (U, U', D, D', L, L', R, R', F, F', B, B') represents a 90-degree rotation of a specific face of the cube. ' is the counterclockwise rotation.

Sequential Backtracking Solver

- The solver initializes the moves list, sets a maximum search depth, and prepares a set to track visited states
- Backtracking DFS: For each possible move, apply the move and recursively search the new state.
  - Move Pruning: Avoid reversing the previous move to reduce redundant searches.
  - Backtrack: If the move does not lead to a solution, undo it and try the next move.
- Determines if the search has found a solution, if the search should continue with a higher bound,

Parallel version

- In the initial levels of the search tree (depth ≤ 2), the solver parallelizes the search for each possible move from the current state. After the initial levels (depth > 2), the solver switches to a sequential search to avoid overhead. Quickly identifies promising paths and prunes less promising ones early in the search process.
- Use thread pool to manage the execution of parallel tasks.
- Generates and evaluates each possible move from the current state concurrently

Heuristic

Estimate the minimum number of moves required to solve the cube. The function helps guide the search process more efficiently by focusing on states that appear closer to the solution, reducing the overall number of moves and time needed to find the solution.

What I learned

- heuristic functions can significantly reduce the search space and improve solver performance.
- Backtracking algorithm
- parallelization speeds up the search but it can also lead to overhead that counteracts the benefits

Problems

Cant figure out how to solve the cube more efficiently, the sequential version could only solve till 8/9 moves if more it would just take forever without completing