Don’t need to understand everything

Just need to understand that there’s 6 moves

F, F\_i, L, L\_i, U, U\_i

In the function we are given a start and an end

We are given rubik.i

I is a description of a solved puzzle

Find shortest path from start to end

We do that by applying the moves

Think of start as being a vertex. Call perm\_apply with all of the moves to the configuration

If we call perm.apply(f) then we et a configuration

If we call perm.apply(f\_i) then we get a second

If we call perm.apply(L) etc. etc then we et 6 configurations

Then for the end we call perm.apply on each of the 6 moves too

Then we get configurations

We don’t want to visit configurations that we’ve already seen

We call perm apply and then we check if the resulting state is something we’ve visited before.

Frontier at size 2, then we have 30 vertices because one move is the same as the inverse

Frontier at size 3, then we have 30\*5 vertices

If we haven’t made a move yet then frontier is start, what that really is is rubik.i, my end is rubik.i

First we check is if anything in ends frontier is also in starts frontier. If so then we just return the list of moves needed to get there

In every step we have to check if there’s something in the frontier\_start is also in frontier\_end

The path is the list of moves. Perm\_inverse is what we call to invert the moves for end. It returns a move

The path is moves, start and end are configurations

Have both sides be dictionaries

Key would be configurations, values would be the move that took us there or the depth

Python has sets, and they support intersections