ROBOT VACUUM WORLD ALGORITHM

Vacuum world example(4*4 grid)

R		D	
	##		D
D	##		
		D	

R = Robot starting position)

D= Dirty Cell

#= Obstacle

= Empty Cell

1. State Repressentation
   state = (robot_position, dirty_set)
   robot_position = (row, col)
   dirty_set = set of cells still dirty

2. Initial State
   robot starts at R
   dirty_set contains all D cells

3. Action COST
   Move Up
   Move Down
   Move left
   Move Right
   Clean (only works when current cell is dirty)

4. Goal Test
   dirty_set is empty

Part A. Depth First Search (DFS)

DFS explores one path as deep as possible before backtracking. It is memory efficient but does not guarantee optimal solutions

1. DFS PSEUDOCODE (ITERATIVE)

```
FUNCTION DFS_SEARCH(problem):

  stack ← empty stack
  explored ← empty set

  PUSH (initial_state, empty_path) INTO stack

  nodes_expanded ← 0
  max_frontier_size ← 1

  WHILE stack is not empty:

    max_frontier_size ← max(max_frontier_size, size(stack))

    (state, path) ← POP stack

    IF state in explored:
      CONTINUE

    ADD state to explored
    nodes_expanded ← nodes_expanded + 1

    IF goal_test(state):
      RETURN (path, nodes_expanded, max_frontier_size)

    FOR each action in possible_actions(state):
      next_state ← transition(state, action)

      IF next_state NOT in explored:
        PUSH (next_state, path + action) INTO stack

  RETURN (None, nodes_expanded, max_frontier_size)
```

PART B A* SEARCH

A* is an informed search algorithm that expands the node with the lowest $f(n) = g(n) + h(n)$using a heuristic to guide the search. With an admissible heuristic, it guarantees an optimal solution but uses more memory than DFS and IDA*.

2. A* Pseudocode

```
FUNCTION ASTAR_SEARCH(problem, heuristic):

  frontier ← priority queue ordered by f
  explored ← empty map (state → best_g_cost)

  PUSH initial_state with:
    g = 0
    f = h(initial_state)
    path = empty

  nodes_expanded ← 0
  max_frontier_size ← 1

  WHILE frontier is not empty:

    max_frontier_size ← max(max_frontier_size, size(frontier))

    (state, g, path) ← POP state with lowest f
    nodes_expanded ← nodes_expanded + 1

    IF goal_test(state):
      RETURN (path, nodes_expanded, max_frontier_size)

    IF state in explored AND explored[state] ≤ g:
      CONTINUE

    explored[state] ← g

    FOR each action in possible_actions(state):
      next_state ← transition(state, action)
      new_g ← g + 1
```

```
            new_f ← new_g + heuristic(next_state)

            PUSH next_state with priority new_f
                and path + action

    RETURN (None, nodes_expanded, max_frontier_size)
```

PART C IDA* SEARCH

IDA* combines A*'s heuristic guidance with DFS's low memory usage by repeatedly performing depth-limited searches using increasing f-cost thresholds.

IDA* Pseudocode

```
FUNCTION IDASTAR_SEARCH(problem, heuristic):

    threshold ← heuristic(initial_state)
    nodes_expanded ← 0
    iterations ← 0

    WHILE TRUE:
        iterations ← iterations + 1

        result ← DFS_F_LIMITED(
                    state = initial_state,
                    g = 0,
                    threshold,
                    path = empty
                 )

        IF result is SOLUTION:
            RETURN (solution_path, nodes_expanded, iterations)

        IF result == ∞:
            RETURN None

        threshold ← result
```