Question: Blocksworld Task: Initial State: stack1: D stack2: G C A F stack3: B I E H Goal State: stack1: A F H stack2: C G I stack3: B D E Please provide a series of moves to reach the goal state. Your answer:

SteerLLM Guidance 1: Solve the following question using your coding skills. Suggest python code (in a python coding block) for the user to execute. You can first output your thinking steps with texts and then the final python code. Start the python block with ```python

TaskLLM Answer 1: To solve the Blocksworld task, we need to move blocks from their initial positions to match the goal …<omitted text>…

```python
moves = [
"Move D from stack1 to stack3",
"Move F from stack2 to stack1",
"Move A from stack2 to stack1",
"Move H from stack3 to stack1",
"Move I from stack3 to stack2",
"Move E from stack3 to stack2"
]
print("<<<")
for move in moves:
print(move)
print(">>>")
```

☹ ✗

SteerLLM Guidance 2: To improve the solution, consider implementing a more sophisticated algorithm that can handle the constraints of the Blocksworld task. Use symbolic computing or an efficient search algorithm, such as A* or depth-first search, to explore possible moves and find a valid sequence that achieves the goal state …

TaskLLM Answer 2: To solve the Blocksworld task using a more sophisticated algorithm, we can implement a depth-first search (DFS) approach …<omitted text>…

```python
def is_goal_state(state, goal_state):
return state == goal_state

def get_possible_moves(state):
moves = []
for i, stack in enumerate(state):
if stack: # If the stack is not empty
block = stack[-1] # Get the top block
for j in range(len(state)):
if i != j: # Move to a different stack
new_state = [list(s) for s in state] # Deep copy of the state
new_state[i].pop() # Remove the block from the current stack
new_state[j].append(block) # Add the block to the new stack
moves.append((new_state, f"Move {block} from stack{i+1} to
stack{j+1}"))
return moves
…<omitted code>…
```

☹ ✗

SteerLLM Guidance 3: The current approach using depth-first search (DFS) is a step in the right direction, but it seems to be inefficient, leading to a timeout. Consider optimizing the search algorithm by implementing a more efficient search strategy, such as A* search, which uses heuristics to prioritize moves that are more likely to lead to the goal state. Additionally, ensure that the state representation and move generation *…<omitted code>…*

TaskLLM Answer 3: To optimize the search for a solution in the Blocksworld task, we can implement the A* search algorithm. A* search is more efficient than DFS …<omitted text>…

```python
import heapq

def heuristic(state, goal_state):
misplaced_blocks = 0
for stack, goal_stack in zip(state, goal_state):
for block, goal_block in zip(stack, goal_stack):
if block != goal_block:
misplaced_blocks += 1
return misplaced_blocks

def is_goal_state(state, goal_state):
return state == goal_state

def get_possible_moves(state):
moves = []
for i, stack in enumerate(state):
if stack: # If the stack is not empty
block = stack[-1] # Get the top block
for j in range(len(state)):
if i != j: # Move to a different stack
new_state = [list(s) for s in state] # Deep copy of the state
new_state[i].pop() # Remove the block from the current stack
new_state[j].append(block) # Add the block to the new stack
moves.append((tuple(tuple(s) for s in new_state), f"Move {block} from
stack{i+1} to stack{j+1}"))
return moves

def a_star_search(initial_state, goal_state):
open_set = []
heapq.heappush(open_set, (0, initial_state, []))
visited = set()
…<omitted code>…
```

☺ ✓

SteerLLM Guidance 4: Return Answer