```
# import os
# from swarms import Agent


# from typing import List


# class DepthFirstSearchSwarm:
#     def __init__(self, agents: List[Agent]):
#         self.agents = agents
#         self.visited = set()


#     def dfs(self, agent, task, results):
#         if agent.agent_name in self.visited:
#             return
#         self.visited.add(agent.agent_name)


#         # Execute the agent's task
#         result = agent.run(task)
#         results.append(result)


#         # If agent produces more tasks, continue the DFS
#         if isinstance(result, dict) and "next_tasks" in result:
#             for next_task in result["next_tasks"]:
#                 next_agent = self.get_next_agent()
#                 if next_agent:
#                     self.dfs(next_agent, next_task, results)
#                 else:
```

```
#                 print("No more agents available for further tasks.")

#     def get_next_agent(self):
#         for agent in self.agents:
#             if agent.agent_name not in self.visited:
#                 return agent
#         return None

#     def run(self, task):
#         results = []
#         if self.agents:
#             initial_agent = self.agents[0]
#             self.dfs(initial_agent, task, results)
#         return results


# # Usage example

# # Define agents with their specific roles or capabilities
# agents = [
#     Agent(
#         agent_name="Financial-Analysis-Agent",
#         system_prompt="Perform financial analysis",
#         llm=OpenAIChat(
#             api_key=os.getenv("OPENAI_API_KEY"),
#             model_name="gpt-4o-mini",
#             temperature=0.1,
#         ),
# max_loops=1,
# autosave=True,
#         verbose=True,
#         streaming_on=True,
#         dynamic_temperature_enabled=True,
#         # saved_state_path="finance_agent.json",
#         user_name="swarms_corp",
#         retry_attempts=3,
#         context_length=200000,
#     ),
#     # Add more agents with specific tasks if needed
# ]

# # Initialize the DFS swarm
# dfs_swarm = DepthFirstSearchSwarm(agents)

# # Run the DFS swarm with a task
# task = (
#     "Analyze the financial components of a startup's stock incentive plan."
# )
# results = dfs_swarm.run(task)

# # Print the results
# for idx, result in enumerate(results):
```

```
#     print(f"Result from Agent {idx + 1}: {result}")


# ####################
# import os
# from swarms import Agent


# class DFSSwarm:
#     def __init__(self, agents):
#         self.agents = agents
#         self.visited = set()

#     def dfs(self, agent_index, task, previous_output=None):
#         if agent_index >= len(self.agents):
#             return previous_output

#         agent = self.agents[agent_index]

#         # Use the previous agent's output as input to the current agent
#         if previous_output:
#             task = f"{task}\nPrevious result: {previous_output}"

#         # Run the current agent's task
#         output = agent.run(task)

#         # Add output to visited to avoid redundant work
#         self.visited.add(output)
#         # Recursively call DFS on the next agent
#         return self.dfs(agent_index + 1, task, output)

#     def run(self, task):
#         # Start DFS from the first agent
#         return self.dfs(0, task)



# # Get the OpenAI API key from the environment variable
# api_key = os.getenv("OPENAI_API_KEY")

# # Create an instance of the OpenAIChat class for each agent
# model = OpenAIChat(openai_api_key=api_key, model_name="gpt-4o-mini", temperature=0.1)

# # Initialize multiple agents
# agent1 = Agent(
#     agent_name="Agent-1",
#     system_prompt="Agent 1 prompt description here",
#     llm=model,
# max_loops=1,
# autosave=True,
#     dynamic_temperature_enabled=True,
#     verbose=True,
#     streaming_on=True,
#     user_name="swarms_corp",
# )


# agent2 = Agent(
#     agent_name="Agent-2",
#     system_prompt="Agent 2 prompt description here",
#     llm=model,
# max_loops=1,
# autosave=True,
#     dynamic_temperature_enabled=True,
#     verbose=True,
#     streaming_on=True,
#     user_name="swarms_corp",
# )


# # Add more agents as needed
# # agent3 = ...
# # agent4 = ...


# # Create the swarm with the agents
# dfs_swarm = DFSSwarm(agents=[agent1, agent2])


# # Run the DFS swarm on a task
# result = dfs_swarm.run("Analyze the financial components of a startup's stock incentives.")
# print("Final Result:", result)
```

```
class DFSSwarm:
    def __init__(self, agents):
        self.agents = agents
        self.visited = set()

    def dfs(self, agent_index, task, previous_output=None):
        if agent_index >= len(self.agents):
            return previous_output

        agent = self.agents[agent_index]

        # If there is a previous output, include it in the task for the next agent
        if previous_output:
            task = f"{task}\nPrevious result: {previous_output}"

        # Run the current agent's task and get the output
        output = agent.run(task)

        # Log the output (optional)
        print(f"Agent {agent_index + 1} Output: {output}")

        # Add output to visited to avoid redundant work
        self.visited.add(output)

        # Recursively call DFS on the next agent
        return self.dfs(agent_index + 1, task, output)
```

```
def run(self, task):

    # Start DFS from the first agent and return the final result

    final_result = self.dfs(0, task)

    return final_result
```