Министерство науки и образования Российской Федерации

ФГБОУ ВО «Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Лабораторная работа №3**

Дисциплина: «Системы искусственного интеллекта»

«Базы знаний и онтологии»

Вариант 12

Выполнил студент  
группы ИВТАПбд-41  
Меховников Е.А.

Проверил:  
преподаватель кафедры ВТ  
Хайруллин И. Д.

Ульяновск, 2024

**Цель работы**

В рамках данной лабораторной работы необходимо разработать систему управления для автоматизации заданного технического объекта. Для хранения базы правил и онтологии предметной области предлагается использовать Neo4j. Задание предполагает проектирование логической модели управления, построение и настройку правил управления, а также разработку симулятора для проверки работы системы управления. Для формирования условий срабатывания правил необходимо использовать фуззификацию на основе нечеткой логики.

Также необходимо разработать минимально рабочий симулятор предметной области с дискретным программным управлением.

Необходимо автоматизировать склад, система должна управлять движением роботов для перемещения товаров по складу в зависимости от наличия заказов.

**Ход работы**

Для моделирования системы необходимо создать узлы, представляющие различные категории состояний: StorageCondition (состояние склада), OrderCondition (состояние заказа) и RobotCondition (состояние робота). Каждый из этих узлов будет связан с определенными значениями, описывающими текущее состояние. После этого необходимо создать отношения ACTION, которые будут устанавливать связь между этими узлами и соответствующими действиями, которые система может предпринять..

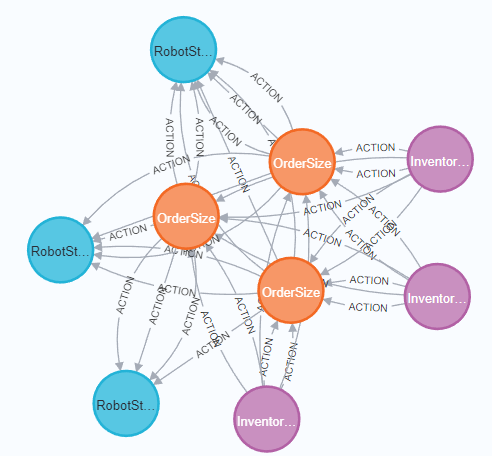


Рис. 1 - Схема в Neo4j

В этой системе получается такая таблица сценариев:

*Таблица 1. Сценариев системы*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Состояние склада (Storage Condition)** | **Тип заказа (Order Condition)** | **Состояние робота (Robot Condition)** | **Действие (Action)** | **Действие робота (Robot Action)** |
| Мало товаров (LowStock) | Маленький заказ (SmallOrder) | Свободен (Idle) | Быстрое пополнение (QuickRestock) | Быстрая сборка (FastPick) |
| Мало товаров (LowStock) | Маленький заказ (SmallOrder) | Свободен (Idle) | Быстрое пополнение (QuickRestock) | Робот не занят (RobotIdle) |
| Мало товаров (LowStock) | Средний заказ (MediumOrder) | Занят (Busy) | Уведомление менеджера (AlertManager) | Уведомление цепочки поставок (NotifySupplyChain) |
| Мало товаров (LowStock) | Средний заказ (MediumOrder) | Занят (Busy) | Уведомление менеджера (AlertManager) | Робот занят (RobotBusy) |
| Мало товаров (LowStock) | Крупный заказ (LargeOrder) | Заряжается (Charging) | Экстренное пополнение (EmergencyRestock) | Экспресс-доставка (UrgentShipping) |
| Мало товаров (LowStock) | Крупный заказ (LargeOrder) | Заряжается (Charging) | Экстренное пополнение (EmergencyRestock) | Робот на зарядке (RobotRecharging) |
| Средний запас (MediumStock) | Маленький заказ (SmallOrder) | Свободен (Idle) | Стандартная операция (StandardOperation) | Обычная сборка (NormalPick) |
| Средний запас (MediumStock) | Маленький заказ (SmallOrder) | Свободен (Idle) | Стандартная операция (StandardOperation) | Робот не занят (RobotIdle) |
| Средний запас (MediumStock) | Средний заказ (MediumOrder) | Занят (Busy) | Оптимизация маршрута (OptimizeRouting) | Эффективная сборка (EfficientPick) |
| Средний запас (MediumStock) | Средний заказ (MediumOrder) | Занят (Busy) | Оптимизация маршрута (OptimizeRouting) | Робот занят (RobotBusy) |
| Средний запас (MediumStock) | Крупный заказ (LargeOrder) | Заряжается (Charging) | Подготовка ресурсов (PrepareResources) | Выделение ресурсов (AllocateResources) |
| Средний запас (MediumStock) | Крупный заказ (LargeOrder) | Заряжается (Charging) | Подготовка ресурсов (PrepareResources) | Робот на зарядке (RobotRecharging) |
| Высокий запас (HighStock) | Маленький заказ (SmallOrder) | Свободен (Idle) | Быстрая доставка (RapidShipping) | Немедленная сборка (ImmediatePick) |
| Высокий запас (HighStock) | Маленький заказ (SmallOrder) | Свободен (Idle) | Быстрая доставка (RapidShipping) | Робот не занят (RobotIdle) |
| Высокий запас (HighStock) | Средний заказ (MediumOrder) | Занят (Busy) | Сбалансированная работа (BalancedWorkload) | Параллельная обработка (ParallelProcessing) |
| Высокий запас (HighStock) | Средний заказ (MediumOrder) | Занят (Busy) | Сбалансированная работа (BalancedWorkload) | Робот занят (RobotBusy) |
| Высокий запас (HighStock) | Крупный заказ (LargeOrder) | Заряжается (Charging) | Максимизация эффективности (MaximizeEfficiency) | Масштабирование работы (ScaleOperation) |
| Высокий запас (HighStock) | Крупный заказ (LargeOrder) | Заряжается (Charging) | Максимизация эффективности (MaximizeEfficiency) | Робот на зарядке (RobotRecharging) |
| Мало товаров (LowStock) | Маленький заказ (SmallOrder) | Заряжается (Charging) | Простой заряд (SimpleCharge) | Медленная сборка (SlowPick) |
| Мало товаров (LowStock) | Маленький заказ (SmallOrder) | Заряжается (Charging) | Простой заряд (SimpleCharge) | Робот на зарядке (RobotRecharging) |
| Мало товаров (LowStock) | Средний заказ (MediumOrder) | Свободен (Idle) | Дополнительное пополнение (AdditionalRestock) | Умеренная сборка (ModeratePick) |
| Мало товаров (LowStock) | Средний заказ (MediumOrder) | Свободен (Idle) | Дополнительное пополнение (AdditionalRestock) | Робот не занят (RobotIdle) |
| Мало товаров (LowStock) | Крупный заказ (LargeOrder) | Занят (Busy) | Критическое пополнение (CriticalRestock) | Экспресс-доставка (ExpressShipping) |
| Мало товаров (LowStock) | Крупный заказ (LargeOrder) | Занят (Busy) | Критическое пополнение (CriticalRestock) | Робот занят (RobotBusy) |
| Средний запас (MediumStock) | Маленький заказ (SmallOrder) | Занят (Busy) | Параллельная сборка (ParallelOperation) | Быстрая сборка (RapidPick) |
| Средний запас (MediumStock) | Маленький заказ (SmallOrder) | Занят (Busy) | Параллельная сборка (ParallelOperation) | Робот занят (RobotBusy) |
| Средний запас (MediumStock) | Средний заказ (MediumOrder) | Свободен (Idle) | Регулярная операция (RegularOperation) | Нормальная доставка (NormalShipping) |
| Средний запас (MediumStock) | Средний заказ (MediumOrder) | Свободен (Idle) | Регулярная операция (RegularOperation) | Робот не занят (RobotIdle) |
| Средний запас (MediumStock) | Крупный заказ (LargeOrder) | Свободен (Idle) | Продуктивное пополнение (ProductiveRestock) | Быстрая доставка (FastShipping) |
| Средний запас (MediumStock) | Крупный заказ (LargeOrder) | Свободен (Idle) | Продуктивное пополнение (ProductiveRestock) | Робот не занят (RobotIdle) |
| Высокий запас (HighStock) | Маленький заказ (SmallOrder) | Занят (Busy) | Оптимизированная доставка (OptimizedShipping) | Сбалансированная сборка (BalancedPick) |
| Высокий запас (HighStock) | Маленький заказ (SmallOrder) | Занят (Busy) | Оптимизированная доставка (OptimizedShipping) | Робот занят (RobotBusy) |
| Высокий запас (HighStock) | Средний заказ (MediumOrder) | Свободен (Idle) | Максимальная скорость (MaximumSpeed) | Быстрая сборка (FastPick) |
| Высокий запас (HighStock) | Средний заказ (MediumOrder) | Свободен (Idle) | Максимальная скорость (MaximumSpeed) | Робот не занят (RobotIdle) |
| Высокий запас (HighStock) | Крупный заказ (LargeOrder) | Свободен (Idle) | Масштабируемая операция (ScalableOperation) | Масштабируемая доставка (LargeScaleShipping) |
| Высокий запас (HighStock) | Крупный заказ (LargeOrder) | Свободен (Idle) | Масштабируемая операция (ScalableOperation) | Робот не занят (RobotIdle) |

Метод get\_action() отвечает за выполнение запроса к базе данных (Листинг 1).

*Листинг 1. Запрос в базу данных для получения действия робота и склада*

|  |
| --- |
| query = (  "MATCH (st:StorageCondition {value: $storage\_condition})-[a:ACTION]->"  "(o:OrderCondition {value: $order\_size})-[b:ACTION]->(r:RobotCondition {value: $robot\_status}) "  "RETURN a.name AS storage\_action, b.name AS robot\_action" ) |

Далее используются функции фуззификаций (fuzzify\_storage\_condition, fuzzify\_order\_size, fuzzify\_robot\_status) для преобразования входных параметров (Листинг 2).

*Листинг 2. Фуззификации*

|  |
| --- |
| def fuzzify\_storage\_condition(self, stock\_level):  """Фуззификация уровня запасов на складе (0-100)"""  if stock\_level <= 33:  return "LowStock"  elif stock\_level <= 66:  return "MediumStock"  else:  return "HighStock"  def fuzzify\_order\_size(self, order\_size):  """Фуззификация размера заказа (0-100)"""  if order\_size <= 33:  return "SmallOrder"  elif order\_size <= 66:  return "MediumOrder"  else:  return "LargeOrder"  def fuzzify\_robot\_status(self, battery\_level):  """Фуззификация статуса робота (0-100)"""  if battery\_level <= 33:  return "Charging"  elif battery\_level <= 66:  return "Idle"  else:  return "Busy" |

Затем выполняется запрос к графу, связывающему узлы, представляющие состояния склада, заказа и робота, через отношения ACTION. Запрос определяет соответствующее действие, и в случае его нахождения это действие возвращается; в противном случае возвращается строка "No action".

Метод simulate() осуществляет пошаговое моделирование, отслеживая изменения состояний склада, заказа и робота. На каждом шаге моделирования вызывается функция get\_action() для определения соответствующего действия, исходя из текущих состояний. В результате этого действия происходит изменение состояний. Для робота предусмотрена отдельная логика, регулирующая изменение уровня заряда в зависимости от его состояния (Листинг 3).

*Листинг 3. Функция simulate()*

|  |
| --- |
| def simulate(self, initial\_storage\_condition, initial\_order\_size, initial\_battery\_level, steps=5):  # Симуляция работы склада и робота на протяжении заданного количества шагов  storage\_condition = self.fuzzify\_storage\_condition(initial\_storage\_condition)  order\_size = self.fuzzify\_order\_size(initial\_order\_size)  robot\_status = self.fuzzify\_robot\_status(initial\_battery\_level)  battery\_level = initial\_battery\_level   for step in range(steps):  # Получаем действия для склада и робота  storage\_action, robot\_action = self.get\_action(storage\_condition, order\_size, robot\_status)   print(  f"Шаг {step+1}: StorageCondition={storage\_condition}, OrderSize={order\_size}, "  f"RobotStatus={robot\_status}, BatteryLevel={battery\_level}, StorageAction={storage\_action}, RobotAction={robot\_action}"  )   # Обновляем статус робота  if robot\_status == "Charging":  battery\_level += np.random.uniform(7.0, 10.0)  elif robot\_status == "Idle":  battery\_level -= np.random.uniform(1.5, 3.0)  elif robot\_status == "Busy":  battery\_level -= np.random.uniform(3.0, 7.0)   # Ограничиваем заряд батареи между 0 и 100  battery\_level = max(0, min(battery\_level, 100))   # Обновляем состояние склада после изменений  storage\_condition = self.fuzzify\_storage\_condition(np.random.randint(0, 101))   # Обновляем размер заказа  order\_size = self.fuzzify\_order\_size(np.random.randint(0, 101)) |

Данный код описывает автоматизированную систему управления складом, где робот самостоятельно выполняет операции по сбору и доставке товаров, руководствуясь текущим состоянием склада, поступившими заказами и уровнем заряда своей батареи.

**Инициализация системы управления**: Программа начинается с создания объекта класса WarehouseRobotControlSystem, который управляет роботом на складе. Для его инициализации необходимы параметры подключения к базе данных или серверу: URI, USER и PASSWORD. Эти параметры обеспечивают доступ системы к данным и настройку соединения с внешними сервисами или базами данных, с которыми она взаимодействует (Листинг 4).

*Листинг 4. Инициализация системы*

|  |
| --- |
| system = WarehouseRobotControlSystem(URI, USER, PASSWORD) |

\***Сценарий 1:** Низкий уровень запасов (20%), небольшой заказ (30 единиц), робот занят (заряд 50%).

\***Сценарий 2:** Средний уровень запасов (50%), крупный заказ (90 единиц), робот свободен (заряд 80%).

\***Сценарий 3:** Высокий уровень запасов (80%), небольшой заказ (30 единиц), робот заряжается (заряд 25%).*Листинг 5. Сценарии системы*

|  |
| --- |
| system.simulate(initial\_storage\_condition=20, initial\_order\_size=30, initial\_battery\_level=50, steps=STEPS)  system.simulate(initial\_storage\_condition=50, initial\_order\_size=90, initial\_battery\_level=80, steps=STEPS)  system.simulate(initial\_storage\_condition=80, initial\_order\_size=30, initial\_battery\_level=25, steps=STEPS) |

Ниже приведен список шагов для каждого сценария, который демонстрирует изменения в действиях роботов на каждом этапе работы в зависимости от условий склада и батареи (Листинг 6).

*Листинг 6. Вывод программы*

|  |
| --- |
| Сценарий 1: Низкий запас, маленький заказ, робот занят  Шаг 1: StorageCondition=LowStock, OrderSize=SmallOrder, RobotStatus=Idle, BatteryLevel=50, StorageAction=QuickRestock, RobotAction=FastPick  Шаг 2: StorageCondition=HighStock, OrderSize=SmallOrder, RobotStatus=Idle, BatteryLevel=47.938189821728955, StorageAction=RapidShipping, RobotAction=FastPick  Шаг 3: StorageCondition=MediumStock, OrderSize=SmallOrder, RobotStatus=Idle, BatteryLevel=45.340198909011846, StorageAction=StandardOperation, RobotAction=FastPick  Шаг 4: StorageCondition=HighStock, OrderSize=LargeOrder, RobotStatus=Idle, BatteryLevel=43.60617094834819, StorageAction=MaximizeEfficiency, RobotAction=FastShipping  Шаг 5: StorageCondition=HighStock, OrderSize=LargeOrder, RobotStatus=Idle, BatteryLevel=42.019045530095894, StorageAction=MaximizeEfficiency, RobotAction=FastShipping  Сценарий 2: Средний запас, крупный заказ, робот свободен  Шаг 1: StorageCondition=MediumStock, OrderSize=LargeOrder, RobotStatus=Busy, BatteryLevel=80, StorageAction=PrepareResources, RobotAction=ExpressShipping  Шаг 2: StorageCondition=HighStock, OrderSize=SmallOrder, RobotStatus=Busy, BatteryLevel=76.7743536838916, StorageAction=RapidShipping, RobotAction=RapidPick  Шаг 3: StorageCondition=MediumStock, OrderSize=MediumOrder, RobotStatus=Busy, BatteryLevel=72.9249972411785, StorageAction=OptimizeRouting, RobotAction=NotifySupplyChain  Шаг 4: StorageCondition=HighStock, OrderSize=MediumOrder, RobotStatus=Busy, BatteryLevel=69.19137920176476, StorageAction=BalancedWorkload, RobotAction=NotifySupplyChain  Шаг 5: StorageCondition=HighStock, OrderSize=MediumOrder, RobotStatus=Busy, BatteryLevel=64.09235347523581, StorageAction=BalancedWorkload, RobotAction=NotifySupplyChain  … |

**Вывод**

В рамках этой лабораторной работы была создана система управления складом, использующая роботов для автоматического перемещения товаров. Система учитывает состояние склада, типы заказов и уровень заряда роботов. Для хранения данных и создания связных правил управления применена база данных Neo4j, что позволило представить логику управления в виде графа. Методы нечеткой логики обеспечивают гибкость в принятии решений и позволяют системе адаптироваться к различным ситуациям. Разработанный симулятор использовался для тестирования различных сценариев работы системы и проверки её корректности в разнообразных условиях.

**Приложение А. Код на Python**

|  |
| --- |
| from neo4j import GraphDatabase import numpy as np  # Подключение к базе данных Neo4j URI = "bolt://localhost:7687" USER = "neo4j" PASSWORD = "admin1234" STEPS = 5 # Количество шагов симуляции np.random.seed(42)  class WarehouseRobotControlSystem:  def \_\_init\_\_(self, uri, user, password):  # Инициализация подключения к базе данных Neo4j  self.driver = GraphDatabase.driver(uri, auth=(user, password))   def close(self):  # Закрытие подключения к базе данных  self.driver.close()   def get\_action(self, storage\_condition, order\_size, robot\_status):  # Запрос в базу данных для получения действия робота и склада  query = (  "MATCH (st:StorageCondition {value: $storage\_condition})-[a:ACTION]->"  "(o:OrderCondition {value: $order\_size})-[b:ACTION]->(r:RobotCondition {value: $robot\_status}) "  "RETURN a.name AS storage\_action, b.name AS robot\_action"  )   with self.driver.session() as session:  result = session.run(  query,  storage\_condition=storage\_condition,  order\_size=order\_size,  robot\_status=robot\_status  )   records = result.data()  if records:  # Выбираем первое совпадение  record = records[0]  return record["storage\_action"], record["robot\_action"]  # Возвращаем дефолтные действия, если не найдены соответствующие в базе  return "No action", "No action"   def fuzzify\_storage\_condition(self, stock\_level):  """Фуззификация уровня запасов на складе (0-100)"""  if stock\_level <= 33:  return "LowStock"  elif stock\_level <= 66:  return "MediumStock"  else:  return "HighStock"   def fuzzify\_order\_size(self, order\_size):  """Фуззификация размера заказа (0-100)"""  if order\_size <= 33:  return "SmallOrder"  elif order\_size <= 66:  return "MediumOrder"  else:  return "LargeOrder"   def fuzzify\_robot\_status(self, battery\_level):  """Фуззификация статуса робота (0-100)"""  if battery\_level <= 33:  return "Charging"  elif battery\_level <= 66:  return "Idle"  else:  return "Busy"   def simulate(self, initial\_storage\_condition, initial\_order\_size, initial\_battery\_level, steps=5):  # Симуляция работы склада и робота на протяжении заданного количества шагов  storage\_condition = self.fuzzify\_storage\_condition(initial\_storage\_condition)  order\_size = self.fuzzify\_order\_size(initial\_order\_size)  robot\_status = self.fuzzify\_robot\_status(initial\_battery\_level)  battery\_level = initial\_battery\_level   for step in range(steps):  # Получаем действия для склада и робота  storage\_action, robot\_action = self.get\_action(storage\_condition, order\_size, robot\_status)   print(  f"Шаг {step+1}: StorageCondition={storage\_condition}, OrderSize={order\_size}, "  f"RobotStatus={robot\_status}, BatteryLevel={battery\_level}, StorageAction={storage\_action}, RobotAction={robot\_action}"  )   # Обновляем статус робота  if robot\_status == "Charging":  battery\_level += np.random.uniform(7.0, 10.0)  elif robot\_status == "Idle":  battery\_level -= np.random.uniform(1.5, 3.0)  elif robot\_status == "Busy":  battery\_level -= np.random.uniform(3.0, 7.0)   # Ограничиваем заряд батареи между 0 и 100  battery\_level = max(0, min(battery\_level, 100))   # Обновляем состояние склада после изменений  storage\_condition = self.fuzzify\_storage\_condition(np.random.randint(0, 101))   # Обновляем размер заказа  order\_size = self.fuzzify\_order\_size(np.random.randint(0, 101))   # Инициализация системы управления if \_\_name\_\_ == "\_\_main\_\_":  system = WarehouseRobotControlSystem(URI, USER, PASSWORD)   try:  print("\nСценарий 1: Низкий запас, маленький заказ, робот занят")  system.simulate(initial\_storage\_condition=20, initial\_order\_size=30, initial\_battery\_level=50, steps=STEPS)   print("\nСценарий 2: Средний запас, крупный заказ, робот свободен")  system.simulate(initial\_storage\_condition=50, initial\_order\_size=90, initial\_battery\_level=80, steps=STEPS)   print("\nСценарий 3: Высокий запас, маленький заказ, робот заряжается")  system.simulate(initial\_storage\_condition=80, initial\_order\_size=30, initial\_battery\_level=25, steps=STEPS)   finally:  system.close() |

**Приложение Б. Скрипт системы на neo4j**

|  |
| --- |
| // Удаляем все существующие данные MATCH (n) DETACH DELETE n;  // Создаем узлы для состояния склада CREATE (st1:StorageCondition {value: 'LowStock', type: 'InventoryStatus'}), // Мало товаров на складе  (st2:StorageCondition {value: 'MediumStock', type: 'InventoryStatus'}), // Среднее количество товаров  (st3:StorageCondition {value: 'HighStock', type: 'InventoryStatus'}) // Высокий уровень запасов WITH st1, st2, st3  // Создаем узлы для типов заказов CREATE (o1:OrderCondition {value: 'SmallOrder', type: 'OrderSize'}), // Маленький заказ  (o2:OrderCondition {value: 'MediumOrder', type: 'OrderSize'}), // Средний заказ  (o3:OrderCondition {value: 'LargeOrder', type: 'OrderSize'}) // Крупный заказ WITH st1, st2, st3, o1, o2, o3  // Создаем узлы для состояния робота CREATE (r1:RobotCondition {value: 'Idle', type: 'RobotStatus'}), // Робот свободен  (r2:RobotCondition {value: 'Busy', type: 'RobotStatus'}), // Робот занят  (r3:RobotCondition {value: 'Charging', type: 'RobotStatus'}) // Робот заряжается WITH st1, st2, st3, o1, o2, o3, r1, r2, r3  // Устанавливаем связи между состояниями склада, заказами и состояниями роботов с соответствующими действиями  // Если мало товаров, маленький заказ и робот свободен -> Быстрая сборка MERGE (st1)-[:ACTION {name: 'QuickRestock'}]->(o1)-[:ACTION {name: 'FastPick'}]->(r1)  // Если мало товаров, средний заказ и робот занят -> Уведомление менеджера MERGE (st1)-[:ACTION {name: 'AlertManager'}]->(o2)-[:ACTION {name: 'NotifySupplyChain'}]->(r2)  // Если мало товаров, крупный заказ и робот заряжается -> Экстренная доставка MERGE (st1)-[:ACTION {name: 'EmergencyRestock'}]->(o3)-[:ACTION {name: 'UrgentShipping'}]->(r3)  // Если средний запас, маленький заказ и робот свободен -> Обычная сборка MERGE (st2)-[:ACTION {name: 'StandardOperation'}]->(o1)-[:ACTION {name: 'NormalPick'}]->(r1)  // Если средний запас, средний заказ и робот занят -> Оптимизация маршрута MERGE (st2)-[:ACTION {name: 'OptimizeRouting'}]->(o2)-[:ACTION {name: 'EfficientPick'}]->(r2)  // Если средний запас, крупный заказ и робот заряжается -> Подготовка ресурсов MERGE (st2)-[:ACTION {name: 'PrepareResources'}]->(o3)-[:ACTION {name: 'AllocateResources'}]->(r3)  // Если высокий запас, маленький заказ и робот свободен -> Быстрая доставка MERGE (st3)-[:ACTION {name: 'RapidShipping'}]->(o1)-[:ACTION {name: 'ImmediatePick'}]->(r1)  // Если высокий запас, средний заказ и робот занят -> Сбалансированная работа MERGE (st3)-[:ACTION {name: 'BalancedWorkload'}]->(o2)-[:ACTION {name: 'ParallelProcessing'}]->(r2)  // Если высокий запас, крупный заказ и робот заряжается -> Максимальная эффективность MERGE (st3)-[:ACTION {name: 'MaximizeEfficiency'}]->(o3)-[:ACTION {name: 'ScaleOperation'}]->(r3)  // Добавляем дополнительные сценарии  // Если мало товаров, маленький заказ и робот заряжается -> Простой заряд MERGE (st1)-[:ACTION {name: 'SimpleCharge'}]->(o1)-[:ACTION {name: 'SlowPick'}]->(r3)  // Если мало товаров, средний заказ и робот свободен -> Дополнительное пополнение MERGE (st1)-[:ACTION {name: 'AdditionalRestock'}]->(o2)-[:ACTION {name: 'ModeratePick'}]->(r1)  // Если мало товаров, крупный заказ и робот занят -> Срочная доставка MERGE (st1)-[:ACTION {name: 'CriticalRestock'}]->(o3)-[:ACTION {name: 'ExpressShipping'}]->(r2)  // Если средний запас, маленький заказ и робот занят -> Параллельная сборка MERGE (st2)-[:ACTION {name: 'ParallelOperation'}]->(o1)-[:ACTION {name: 'RapidPick'}]->(r2)  // Если средний запас, средний заказ и робот свободен -> Обычная доставка MERGE (st2)-[:ACTION {name: 'RegularOperation'}]->(o2)-[:ACTION {name: 'NormalShipping'}]->(r1)  // Если средний запас, крупный заказ и робот свободен -> Продуктивная сборка MERGE (st2)-[:ACTION {name: 'ProductiveRestock'}]->(o3)-[:ACTION {name: 'FastShipping'}]->(r1)  // Если высокий запас, маленький заказ и робот занят -> Оптимизированная доставка MERGE (st3)-[:ACTION {name: 'OptimizedShipping'}]->(o1)-[:ACTION {name: 'BalancedPick'}]->(r2)  // Если высокий запас, средний заказ и робот свободен -> Максимальная скорость MERGE (st3)-[:ACTION {name: 'MaximumSpeed'}]->(o2)-[:ACTION {name: 'FastPick'}]->(r1)  // Если высокий запас, крупный заказ и робот свободен -> Масштабируемая операция MERGE (st3)-[:ACTION {name: 'ScalableOperation'}]->(o3)-[:ACTION {name: 'LargeScaleShipping'}]->(r1); |