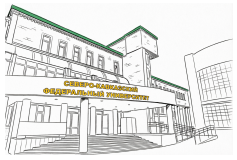

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

Департамент цифровых, робототехнических систем и электроники

Математические модели представления знаний

Методические указания
к выполнению практических работ

Направление подготовки: 09.04.02 «Информационные системы и технологии»

Профиль: «Искусственный интеллект, математическое моделирование и суперкомпьютерные технологии в разработке информационных систем»

Квалификация (степень) выпускника: магистр

Форма обучения: очная

Ставрополь – 2022

УДК 004.8:004.9
ББК 32.81

Аннотация: Методические указания содержат комплект из 16 лабораторных работ по дисциплине «Математические модели представления знаний». Каждая работа включает цель, задачи, теоретический минимум, пошаговый разбор примера, фрагменты конфигурационных файлов и программного кода, индивидуальное задание для самостоятельного выполнения, а также вопросы для самоконтроля. Материал охватывает полный цикл разработки систем, основанных на знаниях: от анализа предметной области и извлечения знаний до формализации в логике предикатов, создания онтологий в Protégé, использования языков RDF/SPARQL и интеграции методов NLP. Рекомендуемый объем — 48 часов лабораторных занятий. Работы выполняются в локальной виртуальной среде с соблюдением норм академической этики и стандартов W3C.
Составитель: к.ф.-м.н. Д.Л. Винокурский
Рецензент: к.ф.м.н. Е.В. Крахоткина

Содержание

1 Введение в математические модели представления знаний: анализ предметной области

Цель: Освоить методы первичного анализа предметной области и выделения ключевых понятий.
Задачи:

- Выбрать предметную область (медицина, образование, техника, право).
- Провести структурный анализ: выделить объекты, атрибуты, отношения, ограничения.
- Составить глоссарий терминов (мин. 25 понятий) с определениями и источниками.
- Построить концептуальную карту в инструменте (CmapTools, Miro, draw.io).

Теоретическое описание: Жизненный цикл системы знаний включает этапы идентификации, концептуализации, формализации и реализации (методология CommonKADS). Анализ предметной области начинается с выделения сущностей и установления связей между ними. Концептуальная модель служит основой для дальнейшей формализации и должна быть независима от конкретной программной реализации. **Разобранный пример:** 1. Выбор области: «Система диагностики неисправностей электродвигателей».
2. Выделение сущностей: Двигатель, Подшипник, Вибрация, Температура, Оператор.
3. Определение связей: «имеет_параметр», «порождает», «требует_обслуживания».
4. Построение глоссария: Вибрация (мм/с) — механическое колебание вала; Норма — < 4.5 мм/с.
5. Визуализация: создание графа понятий с использованием draw.io. **Программный код:**

```
# Example of generating a simple concept graph for analysis
import networkx as nx
import matplotlib.pyplot as plt

G = nx.DiGraph()
G.add_edge("Motor", "Bearing", relation="contains")
G.add_edge("Bearing", "Vibration", relation="generates")
G.add_edge("Vibration", "Diagnosis", relation="determines")

pos = nx.spring_layout(G, seed=42)
nx.draw(G, pos, with_labels=True, node_color="lightblue",
        node_size=2000, font_size=10, arrowsize=15)
plt.show()
```

Индивидуальное задание: Выполните анализ выбранной предметной области. Составьте глоссарий из 30 терминов. Постройте концептуальную карту, выделив не менее 3 уровней иерархии. **Вопросы для самопроверки:**

1. Чем отличаются данные, информация и знания в контексте DIKW-пирамиды?
2. Какие трудности возникают при выделении границ предметной области?
3. Почему концептуальная карта должна быть ациклической на начальных этапах?

2 Методы извлечения знаний: моделирование интервью

Цель: Отработать техники структурированного интервью с экспертом.
Задачи:

- Разработать сценарий интервью для выбранной предметной области (15–20 вопросов).
- Провести имитацию интервью (в парах: «инженер знаний» «эксперт»).
- Зафиксировать ответы, выделить неявные знания и эвристики.
- Преобразовать результаты в структурированную форму (таблица, граф).

Теоретическое описание: Извлечение знаний (Knowledge Acquisition) — критический этап, определяющий качество итоговой системы. Методы включают: свободное интервью, критические инциденты, репертуарные решётки, протоколирование мыслей. Неявные знания часто формулируются как «чувствую, что...», «обычно делаю, если...». Инженер знаний должен уметь задавать уточняющие вопросы и выявлять исключения.
Разобранный пример: 1. Вопрос: «Как вы определяете критический износ подшипника?»
2. Ответ эксперта: «Слушаю гул, если меняется тон — смотрю датчик вибрации».
3. Формализация: Правило: ЕСЛИ тон_шума = «изменён» И вибрация > 4.5 ТО статус = «предкритический».
4. Структурирование: Таблица с колонками [Ситуация, Признак, Значение, Действие, Источник]. **Программный код:**

```
import pandas as pd

# Example of extracted knowledge table structure
knowledge_table = pd.DataFrame([
    {"Condition": "Vibration > 7.0 mm/s", "Action": "Immediate
       shutdown", "Type": "Procedural"},
    {"Condition": "Winding Temp > 90C", "Action": "Reduce load by 30%
       ", "Type": "Heuristic"},
    {"Condition": "Service Interval < 1000 h", "Action": "Replace
       grease", "Type": "Declarative"}
])
print(knowledge_table.to_markdown(index=False))
```

Индивидуальное задание: Составьте сценарий интервью. Проведите сессию с коллегой. Выделите 3 эвристики и 2 исключения. Оформите протокол в виде структурированной таблицы. **Вопросы для самопроверки:**

1. Почему эксперты часто не могут чётко сформулировать свои эвристики?
2. Как отличить факт от субъективного мнения в ходе интервью?
3. Какие приёмы помогают выявить «слепые зоны» в знаниях эксперта?

3 Концептуальное моделирование: ER-диаграммы и онтологии

Цель: Научиться переходить от неформального описания к формальной модели.
Задачи:

- На основе глоссария построить ER-диаграмму (сущности, связи, мощности).
- Выделить иерархии классов и отношения (наследование, агрегация, ассоциация).
- Экспортировать модель в формат, совместимый с инструментами онтологий.
- Проверить целостность связей и типы данных.

Теоретическое описание: ER-моделирование служит мостом между концептуальным анализом и онтологическим проектированием. Сущности становятся классами, связи — объектными свойствами, атрибуты — дата-свойствами. Важно различать классификационные отношения (is-a) и структурные (part-of). На этом этапе фиксируется кардинальность и обязательность полей. **Разобранный пример:** 1. Сущность: «Оборудование» → подклассы: «Двигатель», «Насос».
2. Связь: «Оборудование» —[установлен_в]→ «Цех» (N:1).
3. Атрибуты: мощность (Float, kW), дата_установки (Date).
4. Ограничение: «Насос» должен иметь атрибут «напор» > 0. **Программный код:**

```
# PlantUML generation of ER diagram
@startuml
entity Equipment {
  * id : string
  --
  * power : float
  install_date : date
}
entity Workshop {
  * id : string
  name : string
}
Equipment "N" -- "1" Workshop : installed_in >
@enduml
```

Индивидуальное задание: Преобразуйте концептуальную карту из РР №1 в ER-диаграмму. Добавьте не менее 5 сущностей, 8 связей, 3 иерархии. Подготовьте схему в draw.io или PlantUML. **Вопросы для самопроверки:**

1. В чём принципиальное отличие ER-модели от онтологии?
2. Как правильно моделировать отношения «многие-ко-многим»?
3. Почему важно фиксировать ограничения целостности до этапа формализации?

4 Логика высказываний: формализация правил

Цель: Освоить базовые средства формальной логики для представления знаний.
Задачи:

- Сформулировать 10–15 правил предметной области в естественном языке.
- Перевести правила в формулы логики высказываний ($\rightarrow, \wedge, \vee, \neg$).
- Проверить непротиворечивость набора правил методом таблиц истинности.
- Реализовать проверку в Python (библиотека ‘sympy.logic‘).

Теоретическое описание: Логика высказываний оперирует атомарными утверждениями, принимающими значения Истина/Ложь. Правила вида «ЕСЛИ А И В, ТО С» формализуются как $(A \wedge B) \rightarrow C$. Набор правил образует базу знаний. Проверка на непротиворечивость выполняется через анализ выполнимости (SAT). **Разобранный пример:** 1. Правило: «Если температура > 80 И вибрация > 5, ТО требуется остановка».
2. Формализация: $T \wedge V \rightarrow S$, где T: температура > 80, V: вибрация > 5, S: остановка.
3. Таблица истинности: проверяется, что при $T = 1, V = 1$ значение $S = 1$. **Программный код:**

```
from sympy.logic.boolalg import And, Implies, satisfiable
from sympy import symbols

T, V, S = symbols('T V S')
# Rule 1: If T and V then S
rule1 = Implies(And(T, V), S)
# Fact: T and V are true
rule2 = And(T, V)

kb = And(rule1, rule2)
print("Satisfiable:", satisfiable(kb))
# Check consequence: S must be True
from sympy.logic.inference import valid
print("S is a consequence:", valid(Implies(kb, S)))
```

Индивидуальное задание: Формализуйте 12 правил для выбранной области. Проверьте набор на непротиворечивость и полноту с помощью ‘sympy‘. Найдите и устраните 1 искусственное противоречие. **Вопросы для самопроверки:**

1. Почему логика высказываний не позволяет выразить утверждение «Все насосы имеют подшипники»?
2. Как проверить, является ли правило тавтологией?
3. В чём разница между синтаксическим и семантическим следствием?

5 Логика предикатов первого порядка: построение базы знаний

Цель: Научиться представлять знания с использованием кванторов и предикатов.
Задачи:

- Расширить модель: ввести объекты, предикаты, функции.
- Записать 10+ фактов и правил в форме формул логики предикатов.
- Реализовать простой механизм логического вывода (forward chaining).
- Протестировать на примерах запросов к базе знаний.

Теоретическое описание: Логика предикатов первого порядка (FOL) расширяет логику высказываний кванторами ($\forall, \exists$), предикатами $P(x)$ и функциями $f(x)$. Позволяет описывать свойства классов, отношения между объектами. Механизмы вывода используют унификацию и резолюцию для доказательства гипотез. **Разобранный пример:**
1. Факты: Motor($M1$), HasBearing($M1, B1$), Worn($B1$).
2. Правило: $\forall x, y(\text{Motor}(x) \land \text{HasBearing}(x, y) \land \text{Worn}(y) \rightarrow \text{NeedsMaintenance}(x))$.
3. Вывод: NeedsMaintenance($M1$) — истинно. **Программный код:**

```python
# Simple FOL interpreter based on rules
facts = {
    "Motor": ["M1", "M2"],
    "HasBearing": [("M1", "B1"), ("M2", "B2")],
    "Worn": ["B1"]
}

def check_needs_maintenance(obj):
    if obj in facts["Motor"]:
        for m, b in facts["HasBearing"]:
            if m == obj and b in facts["Worn"]:
                return True
    return False

print("M1 needs maintenance:", check_needs_maintenance("M1"))
```

Индивидуальное задание: Сформулируйте 15 фактов и 5 правил в нотации FOL. Реализуйте механизм вывода. Протестируйте на 3 запросах. **Вопросы для самопроверки:**

1. Почему FOL называется логикой «первого порядка»?
2. Как квантор существования отличается от квантора общности?
3. В чём заключается проблема экспоненциального роста пространства поиска?

6 Семантические сети и фреймы: визуальное моделирование

Цель: Освоить альтернативные логике форматы представления знаний.
Задачи:

- Построить семантическую сеть для предметной области (узлы — понятия, дуги — отношения).
- Реализовать фреймовую модель: слоты, значения, ограничения, методы-демоны.
- Сравнить выразительность и удобство разных форматов.
- Визуализировать результат в виде графа (Graphviz, NetworkX).

Теоретическое описание: Семантические сети представляют знания в виде ориентированного графа. Фреймы структурируют знания по стереотипам: слоты хранят значения, могут иметь значения по умолчанию и процедуры (демоны), вызываемые при изменении данных. Оба подхода интуитивны для моделирования иерархий. **Разобранный пример:** 1. Сеть: [Двигатель] –(is-a)–> [Оборудование].
2. Фрейм: `Frame: Motor`, Slots: `Power: default=5kW`.
3. Демон: при изменении `Temperature` вызывается проверка порога. **Программный код:**

```
class Frame:
    def __init__(self, name, slots=None):
        self.name = name
        self.slots = slots or {}

    def set_slot(self, key, value):
        self.slots[key] = value
        if key == "Temp" and value > 90:
            print(f"ALERT: Temp {value} exceeded!")

motor = Frame("Motor", {"Power": 5.0})
motor.set_slot("Temp", 95)
```

Индивидуальное задание: Спроектируйте фреймовую модель для 5 классов. Добавьте наследование слотов и 2 демона. Визуализируйте сеть. **Вопросы для самопроверки:**

1. Как механизм наследования слотов решает проблему избыточности?
2. В каких случаях семантическая сеть предпочтительнее продукционных правил?
3. Почему демоны могут приводить к непредсказуемому поведению?

7 Введение в Protégé: создание простой онтологии

Цель: Освоить базовый инструментарий разработки онтологий.
Задачи:

- Установить Protégé 5.x, изучить интерфейс.
- Создать онтологию: определить 5–7 классов, иерархию, свойства.
- Добавить экземпляры (индивиды) и заполнить свойства.
- Экспортировать онтологию в RDF/XML и Turtle.

Теоретическое описание: Protégé — стандартная среда разработки онтологий, поддерживающая форматы RDF/OWL. Онтология описывает концепты, их отношения и ограничения. Классы образуют таксономию, свойства связывают индивидов или связывают индивидов с литералами. **Разобранный пример:** 1. Создание класса `Equipment`, подклассов `Motor`, `Sensor`.
2. Объектное свойство `hasPart`: domain `Equipment`, range `Equipment`.
3. Создание индивида `M1`: тип `Motor`, `hasPower 5.0`. **Программный код:**

```
# Convert OWL to Turtle for analysis
riot --output=Turtle ontology.owl > ontology.ttl
# View triples
head -n 20 ontology.ttl
```

Индивидуальное задание: Создайте онтологию (мин. 8 классов, 5 свойств, 10 индивидов). Экспортируйте в RDF/XML и Turtle. Сделайте скриншоты. **Вопросы для самопроверки:**

1. Почему Protégé использует OWL, а не реляционную схему?
2. В чём разница между объектным и дата-свойством?
3. Как метаданные онтологии влияют на её повторное использование?

8 Онтологические ограничения и аксиомы в OWL

Цель: Научиться использовать выразительные средства OWL для точного моделирования.
Задачи:

- Добавить ограничения: `someValuesFrom`, `allValuesFrom`, `cardinality`.
- Сформулировать 3–5 инференциальных правил через аксиомы.
- Запустить классификатор (HermiT/Pellet), проанализировать выводы.
- Исправить выявленные противоречия.

Теоретическое описание: OWL DL основан на описательных логиках и позволяет задавать строгие семантические ограничения. Дизъюнктность (`owl:disjointWith`) запрещает пересечение классов. Reasoner автоматически вычисляет иерархию и находит противоречия. **Разобранный пример:** 1. Ограничение: `Motor SubClassOf hasPower exactly 1 xsd:float`.
2. Дизъюнктность: `Motor owl:disjointWith Sensor`.
3. Запуск HermiT: выявляет, что `M1` классифицируется как `HighRiskMotor`. **Программный код:**

```
from owlready2 import *

onto = get_ontology("http://test.org/onto.owl")
with onto:
    class Equipment(Thing): pass
    class Motor(Equipment): pass
    class Sensor(Equipment): pass
    Motor.disjoint_with = [Sensor]

onto.save("test_owl.owl")
# sync_reasoner() # Requires Java and reasoner installed
```

Индивидуальное задание: Расширьте онтологию: добавьте 4 ограничения, 2 дизъюнктности. Запустите reasoner, исправьте несоответствия. **Вопросы для самопроверки:**

1. Почему использование `allValuesFrom` может привести к неожиданным следствиям?
2. Как reasoner обрабатывает противоречия?
3. В чём разница между `equivalentClass` и `subClassOf`?

9 Языки семантической паутины: RDF и RDFS

Цель: Освоить базовые стандарты представления знаний в вебе.
Задачи:

- Представить онтологию в формате RDF (Turtle).
- Добавить RDFS-схему: `rdfs:subClassOf`, `rdfs:domain`, `rdfs:range`.
- Загрузить данные в графовую БД (Apache Jena Fuseki или GraphDB Free).
- Выполнить 3–5 простых запросов через SPARQL-интерфейс.

Теоретическое описание: RDF представляет знания в виде триплетов (субъект-предикат-объекет). RDFS добавляет таксономию и ограничения свойств. Turtle — компактный сериализатор RDF. Эти форматы обеспечивают интероперабельность. **Разобранный пример:** 1. Триплет: `:M1 rdf:type :Motor .`
2. Свойство: `:hasPower rdfs:domain :Motor ; rdfs:range xsd:float .`
3. Загрузка в Fuseki. **Программный код:**

```
# Example Turtle file
@prefix : <http://example.org/onto#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:Motor a rdfs:Class ; rdfs:subClassOf :Equipment .
:M1 a :Motor ; :hasPower "5.0"^^<http://www.w3.org/2001/XMLSchema#
   float> .
```

Индивидуальное задание: Сконвертируйте онтологию в Turtle. Напишите RDFS-схему. Разверните SPARQL-endpoint, загрузите данные, выполните запросы. **Вопросы для самопроверки:**

1. Почему RDF использует ориентированный граф?
2. Как `rdfs:subClassOf` влияет на вывод в SPARQL?
3. В чём преимущество Turtle перед XML?

10 Язык запросов SPARQL: извлечение знаний из графа

Цель: Научиться формулировать сложные запросы к онтологиям.
Задачи:

- Подключиться к графовой БД из РР №9.
- Написать 5 запросов разной сложности: выбор, фильтрация, агрегация, конструкт.
- Использовать `FILTER`, `OPTIONAL`, `COUNT`, `GROUP BY`.
- Визуализировать результаты (таблицы, графики).

Теоретическое описание: SPARQL — стандартный язык запросов к RDF-данным. Поддерживает проекцию (`SELECT`), конструирование графов (`CONSTRUCT`) и проверку наличия (`ASK`). Фильтрация и агрегация поддерживают аналитику. **Разобранный пример:** 1. Выбор: `SELECT ?m WHERE {?m a :Motor ; :hasPower ?p . FILTER(?p > 4.5)}`.
2. Агрегация: `SELECT (COUNT(?m) as ?total) WHERE {?m a :Motor}`. **Программный код:**

```
# SPARQL query in query.rq file
PREFIX : <http://example.org/onto#>
SELECT ?motor ?power
WHERE {
  ?motor a :Motor ;
         :hasPower ?power .
  FILTER(?power > 4.0)
}
ORDER BY DESC(?power)
LIMIT 10
```

Индивидуальное задание: Напишите 5 SPARQL-запросов. Выполните через SPARQL-интерфейс. Сравните производительность. **Вопросы для самопроверки:**

1. Чем `CONSTRUCT` отличается от `SELECT`?
2. Почему `OPTIONAL` может увеличивать размер результата?
3. Как индексация влияет на скорость `FILTER`?

11 Продукционные системы: разработка правил

Цель: Освоить архитектуру систем, основанных на правилах.
Задачи:

- Сформулировать 15–20 продукций «ЕСЛИ-ТО».
- Реализовать механизм прямого вывода на Python.
- Протестировать на наборе входных фактов (мин. 5 сценариев).
- Добавить объяснение вывода (трассировка).

Теоретическое описание: Продукционная система состоит из базы правил, рабочей памяти и механизма вывода. Forward chaining активирует правила, чьи условия удовлетворены. Конфликты разрешаются стратегиями приоритета. **Разобранный пример:** 1. Правило 1: ЕСЛИ `temp>80` И `vib>5` ТО `alert=True`.
2. Цикл: проверка → срабатывание → добавление факта → повтор. **Программный код:**

```
facts = {"temp": 85, "vib": 6}
rules = [
    {"cond": lambda f: f["temp"] > 80 and f["vib"] > 5, "act": {"
       alert": True}},
    {"cond": lambda f: f.get("alert") is True, "act": {"action": "
       stop"}}
]
trace = []
changed = True
while changed:
    changed = False
    for r in rules:
        if r["cond"](facts):
            facts.update(r["act"])
            trace.append(r["act"])
            changed = True
print("Result:", facts)
```

Индивидуальное задание: Разработайте продукционную систему. Реализуйте forward chaining с трассировкой. Протестируйте на 5 сценариях. **Вопросы для самопроверки:**

1. Почему forward chaining не всегда подходит для диагностики?
2. Как механизм разрешения конфликтов предотвращает зацикливание?
3. В чём разница между рабочей памятью и базой правил?

12 Построение экспертной системы: прототип

Цель: Интегрировать знания, правила и интерфейс в рабочую систему.
Задачи:

- Выбрать узкую задачу (диагностика, рекомендация).
- Разработать архитектуру: БЗ, вывод, UI.
- Реализовать прототип (консольный или веб на Flask/Streamlit).
- Провести юзабилити-тест с 2–3 пользователями.

Теоретическое описание: Экспертная система имитирует рассуждения специалиста. Архитектура включает модуль знаний, механизм вывода, модуль объяснений и интерфейс. Важна объяснимость (XAI). **Разобранный пример:** 1. Задача: «Рекомендация обслуживания двигателя».
2. UI: Streamlit-форма с ползунками.
3. Вывод: класс риска + текст объяснения. **Программный код:**

```
import streamlit as st
st.title("Expert System")
temp = st.slider("Temperature", 50, 120, 75)
if temp > 90:
    st.error("Critical temperature! Stop required.")
elif temp > 80:
    st.warning("Warning: reduce load.")
else:
    st.success("Normal: operation in standard mode.")
```

Индивидуальное задание: Соберите прототип ЭС. Добавьте модуль объяснений. Проведите тестирование, зафиксируйте метрики. **Вопросы для самопроверки:**

1. Почему модуль объяснений критичен для внедрения ЭС?
2. Как масштабировать базу правил без потери производительности?
3. В каких случаях ЭС целесообразнее заменить моделью ML?

13 Механизмы логического вывода: классификация и реализация

Цель: Сравнить стратегии вывода и реализовать одну из них.
Задачи:

- Изучить стратегии: прямой/обратный вывод, резолюция.
- Реализовать алгоритм унификации/резолюции на Python.
- Протестировать на логических задачах.
- Оценить сложность и полноту.

Теоретическое описание: Резолюция преобразует формулы в КНФ и применяет правило вывода. Унификация находит подстановку, делающую термы идентичными. Резолюция полна для логики первого порядка. **Разобранный пример:** 1. Формулы: $P(x) \to Q(x)$, $P(a)$, цель: $Q(a)$.
2. Резолюция: $\neg P(x) \vee Q(x)$ с $P(a) \to Q(a)$.
3. Доказательство через приведение к противоречию. **Программный код:**

```
def unify(term1, term2):
    if term1 == term2: return {}
    if term1.islower(): return {term1: term2}
    if term2.islower(): return {term2: term1}
    return None

clauses = [["!P(x)", "Q(x)"], ["P(a)"], ["!Q(a)"]]
print("Resolution example:", unify("P(a)", "P(x)"))
```

Индивидуальное задание: Реализуйте упрощённый резолюционный доказатель. Протестируйте на 3 задачах. Сравните с forward chaining. **Вопросы для самопроверки:**

1. Почему резолюция требует КНФ?
2. Как стратегия выбора дизъюнктов влияет на скорость?
3. В чём разница между синтаксической и семантической полнотой?

14 Графы знаний: построение и визуализация

Цель: Освоить технологии построения графов знаний из структурированных данных.
Задачи:

- Взять открытый датасет (Wikidata, DBpedia).
- Извлечь сущности и связи, привести к формату RDF.
- Построить граф знаний в Neo4j или NetworkX.
- Реализовать 3–5 аналитических запросов (поиск путей, центральность).

Теоретическое описание: Граф знаний — это RDF-граф, обогащённый онтологией. Neo4j использует Cypher для обходов. Ключевые метрики: центральность, плотность, связные компоненты. **Разобранный пример:** 1. Загрузка CSV связей.
2. Импорт в Neo4j.
3. Запрос: поиск пути между операторами. **Программный код:**

```
import networkx as nx
import pandas as pd

df = pd.read_csv("equipment_relations.csv")
G = nx.from_pandas_edgelist(df, source="From", target="To",
    create_using=nx.DiGraph)
deg = dict(G.degree())
print("Node centrality:", deg)
```

Индивидуальное задание: Загрузите датасет (мин. 50 узлов). Постройте граф, вычислите метрики. Найдите аномальные узла. Визуализируйте. **Вопросы для самопроверки:**

1. Почему графовые БД эффективнее реляционных для глубоких обходов?
2. Как betweenness centrality помогает в анализе устойчивости?
3. В чём разница между Knowledge Graph и семантической сетью?

15 NLP для извлечения знаний: от текста к онтологии

Цель: Применить методы обработки естественного языка для автоматического извлечения знаний.
Задачи:

- Подготовить корпус текстов по предметной области.
- Использовать NLP-инструменты (spaCy) для NER и relation extraction.
- Сформировать черновик онтологии.
- Сравнить с онтологией, созданной вручную.

Теоретическое описание: NER выделяет сущности, RE определяет связи. Кластеризация терминов выявляет тематические группы. Автоматически извлечённые знания требуют валидации экспертом. **Разобранный пример:** 1. Текст: «Двигатель M1 установлен в цехе A».
2. NER: `Motor(M1)`, `Workshop(A)`.
3. RE: `(M1, located_in, A)`. **Программный код:**

```
import spacy
# Load Russian model or English equivalent for testing
nlp = spacy.load("ru_core_news_sm")
text = "Motor M1 installed in Workshop A."
doc = nlp(text)
for ent in doc.ents:
    print(f"{ent.text} -> {ent.label_}")
```

Индивидуальное задание: Обработайте 15 документов. Извлеките NER и RE. Оцените precision/recall. Сравните с ручной онтологией. **Вопросы для самопроверки:**

1. Почему NLP-модели плохо работают в узких технических доменах?
2. Как dependency parsing помогает извлекать отношения?
3. В каких случаях ручное извлечение остаётся предпочтительным?

16 Интеграционный проект: сквозная разработка системы знаний

Цель: Консолидировать компетенции в рамках полного цикла разработки.
Задачи:

- Выбрать прикладную задачу.
- Пройти все этапы: анализ → моделирование → реализация → тестирование.
- Подготовить презентацию и демо-версию.
- Документировать проект.

Теоретическое описание: Проект моделирует реальный процесс разработки knowledge-intensive systems. Требует координации ролей и валидации с экспертом. Итоговая система должна быть объяснимой и расширяемой. **Разобранный пример:** 1. Тема: «Система поддержки принятия решений для IoT-датчиков».
2. Этапы: Сбор данных → OWL онтология → SPARQL запросы → UI Dashboard.
3. Результат: Рабочий прототип с базой знаний и визуализацией. **Программный код:**

```
# Example project structure
mkdir ke_project
cd ke_project
mkdir -p data ontology rules src docs
touch ontology/ontology.owl
touch src/main.py
```

Индивидуальное задание: Реализуйте сквозной проект. Пройдите полный цикл разработки. Подготовьте документацию и презентацию. **Вопросы для самопроверки:**

1. Какие метрики качества используются для оценки систем знаний?
2. Как организовать версионирование онтологий в команде?
3. В чём заключаются основные риски при развёртывании прототипа в production?

Список литературы

[1] Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. СПб.: Питер, 2001. 384 с.

[2] Рассел С., Норвиг П. Искусственный интеллект: современный подход. 4-е изд. М.: Вильямс, 2020. 1400 с.

[3] W3C. OWL 2 Web Ontology Language. URL: `https://www.w3.org/TR/owl2-overview/`.

[4] Bird S., Klein E., Loper E. Natural Language Processing with Python. O'Reilly, 2009.

[5] Musen M.A. The Protégé Project: A Look Back and a Look Forward. AI Matters, 2015.

[6] SPARQL 1.1 Query Language. W3C Recommendation, 2013. URL: `https://www.w3.org/TR/sparql11-query/`.

[7] RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 2014. URL: `https://www.w3.org/TR/rdf11-concepts/`.

[8] Webber J., Jimenez A. Graph Databases. O'Reilly, 2013.

[9] Wielinga B.J. et al. KADS: A Principled Approach to Knowledge-Based System Development. Academic Press, 1992.

[10] Calvanese D. et al. Ontop: Answering SPARQL Queries over Relational Databases. Semantic Web Journal, 2017.