Отчет по лабораторной работе
Тема: «Реализация семантической сети»

1. Введение

1.1. Текстовая формулировка задачи

Целью работы является разработка программного комплекса на языке C++, реализующего модель семантической сети. Программа должна обеспечивать создание структуры сети путем парсинга данных из внешнего файла формата JSON. В памяти объекты и связи между ними должны быть представлены с помощью соответствующих структур данных. Программный интерфейс должен предоставлять базовые функции для взаимодействия с сетью: вывод полной иерархической структуры, а также выполнение поисковых запросов для нахождения объектов и связей.

1.2. Пример кода, решающего задачу

Пример использования основных функций библиотеки представлен в функции ‘main‘. Данный код инициализирует семантическую сеть из файла, выводит ее полную структуру, а затем выполняет два типа поиска: поиск объекта по отношению и поиск отношения по имени целевого объекта.

Листинг 1. Пример основной логики приложения

```
int main() {
  setlocale(LC_ALL, "Russian");

  if (gSemantic.Create())
  {
    for (int i = 0; i < gSemantic.StaticObjects.size(); i++)
    {
      gSemantic.PrintObject(gSemantic.StaticObjects.at(i));
      std::cout << std::endl << "----------------------------"
          << std::endl;;
    }

    auto Founded = gSemantic.FindObjectByRelation(gSemantic.
       GetObjectByName("Petya"), "has_a");

    if (!Founded.empty())
    {
      for (int i = 0; i < Founded.size(); i++)
      {
        std::cout << "Object by relation: " << Founded.at(i)
           ->Name << std::endl;
```

```
        }
      }

      std::string FoundedRelation = gSemantic.
         FindRelationByObject(gSemantic.GetObjectByName("Petya")
         , "hairs");

      std::cout << "Relation by object: " << FoundedRelation.
         c_str() << std::endl;

      gSemantic.Destroy();
    }
    else
    std::cout << "Parse error" << std::endl;

    system("pause");
    return 0;
  };
```

1.3. Концептуальный граф и результат работы

На рисунке 1 представлена концептуальная схема небольшой части семантической сети. На рисунке 2 показан результат работы программы, демонстрирующий вывод данных.

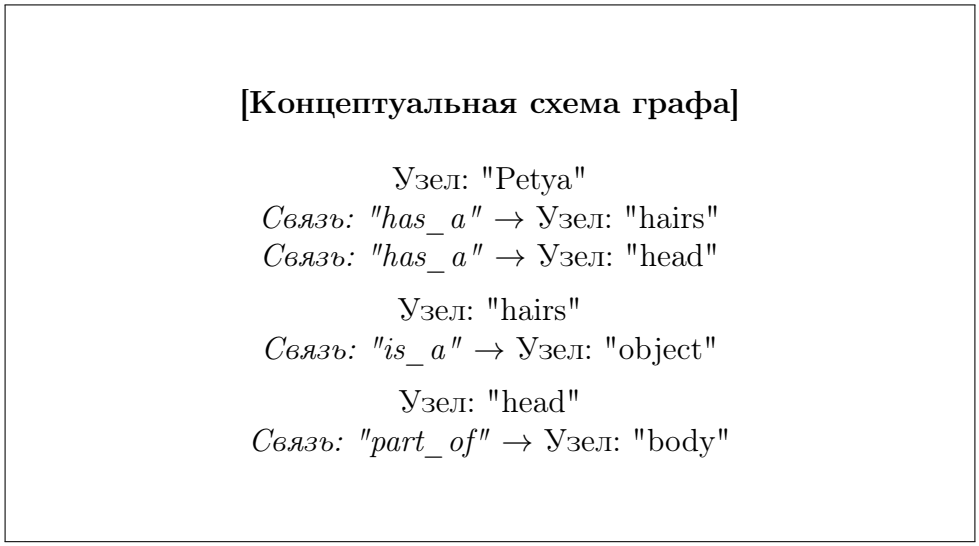

Рис. 1. Концептуальная схема части семантической сети

Рис. 2. Пример вывода программы

2. Ход работы

2.1. Структуры данных

В основе реализации лежит структура ‘cObject‘, которая представляет узел семантической сети. Каждый объект имеет имя (‘std::string Name‘) и набор отношений (‘Relations‘), реализованный через ‘std::map‘. Ключом в ‘map‘ является имя отношения (например, "is_a "has_a"), а значением — вектор указателей (‘std::vector<pObject>‘) на объекты, с которыми установлена данная связь. Это позволяет одному объекту иметь множество связей разных типов с разными объектами.

Листинг 2. Определение структуры объекта

```
    typedef struct cObject {
```

```
    std::string Name;
    std::map<std::string, std::vector<pObject>> Relations;
} *pObject;
```

2.2. Основные компоненты программы

Программа состоит из нескольких ключевых компонентов: парсера JSON и класса для управления семантической сетью.

2.2.1. Парсер JSON

Отвечает за чтение и интерпретацию файла ‘Objects.json‘. Он последовательно создает все объекты, а затем, на втором проходе, устанавливает связи между ними, используя ранее созданные экземпляры. Для работы с JSON используется библиотека ‘nlohmann/json‘.

Листинг 3. Функция парсинга JSON

```
bool cParser::ParseJson(std::vector<pObject> &Objects, std::
    string Path)
{
    std::ifstream InputFile(Path);
    if (!InputFile.is_open()) {
        return false;
    }

    json Data;
    InputFile >> Data;

    for (const auto& NodeJson : Data["objects"]) {
        Objects.push_back(gSemantic.AddObject(NodeJson["name"]));
    }

    for (const auto& RelationJson : Data["relations"]) {
        std::string Relation = RelationJson["relation"];
        pObject Source = gSemantic.GetObjectByName(RelationJson["
            source"], Objects);
        pObject Target = gSemantic.GetObjectByName(RelationJson["
            target"], Objects);
        if (Source && Target)
        gSemantic.AddRelation(Source, Target, Relation);
    }

    return true;
}
```

2.2.2. Класс семантической сети и рекурсивный обход графа

Класс ‘cSemantic‘ инкапсулирует всю логику работы с сетью. Ключевые операции, такие как поиск и вывод, реализованы с помощью рекурсивного обхода графа.

Вывод иерархии. Метод ‘PrintObject‘ рекурсивно обходит связи объекта и выводит их с отступами для наглядного отображения иерархии. Глубина рекурсии отслеживается переменной ‘Cycles‘ для корректного форматирования.

Листинг 4. Рекурсивный вывод структуры сети

```
void cSemantic::PrintObject(pObject Object, pObject
   EntryObject)
{
  std::cout << Ladder(Cycles) << "+Object: " << Object->Name
     << std::endl;

  if(Object->Relations.empty())
  std::cout << std::endl;

  for (auto const& [RelationName, Objects] : Object->
     Relations)
  {
    std::cout << Ladder(Cycles) << "-" << RelationName << ":
       ";
    for (pObject FoundedObject : Objects) {
      std::cout << FoundedObject->Name << " ";
    }

    for (pObject FoundedObject : Objects)
    {
      if (EntryObject == FoundedObject)
      {
        std::cout << std::endl;
        continue;
      }

      if (FoundedObject)
      {
        Cycles++;
        std::cout << std::endl;
        PrintObject(FoundedObject, EntryObject ? EntryObject
           : Object);
      }
    }
  }

  if(Cycles > 0)
```

```
    Cycles--;
  }
```

Поиск по графу. Рассмотрим механизм рекурсии на примере функции ‘FindObjectByRelation‘. Функция принимает на вход текущий объект для проверки (‘Object‘), имя искомой связи (‘Name‘) и ‘EntryObject‘ — узел, из которого был совершен переход на текущий уровень рекурсии.

Принцип работы следующий:

1) **Прямой поиск:** Функция проверяет, есть ли у текущего объекта ‘Object‘ связь с именем ‘Name‘. Если есть, она найдена, и функция возвращает связанные с ней объекты.

2) **Рекурсивный шаг:** Если прямой поиск не дал результатов, функция переходит к обходу в глубину. Она итерирует по всем связям текущего объекта.

3) **Предотвращение зацикливания:** Перед тем как совершить рекурсивный вызов для связанного объекта (‘FoundedObject‘), происходит проверка ‘if (EntryObject == FoundedObject)‘. Это необходимо, чтобы избежать бесконечного цикла в случае двунаправленных связей (например, A $\leftrightarrow$ B). Параметр ‘EntryObject‘ не позволяет функции вернуться в узел, из которого она только что пришла.

4) **Вызов самой себя:** Если проверка на зацикливание пройдена, функция вызывает саму себя для связанного узла: ‘FindObjectByRelation(FoundedObject, Name, Object)‘. Текущий объект ‘Object‘ передается как ‘EntryObject‘ для следующего уровня рекурсии.

Такой подход позволяет обойти всю достижимую из начальной точки часть графа. Аналогичный механизм используется и в других поисковых методах класса.

Листинг 5. Рекурсивный поиск объекта по связи

```
std::vector<pObject> cSemantic::FindObjectByRelation(pObject
    Object, std::string Name, pObject EntryObject)
{
  std::vector<pObject> FoundedObjects;

  if (!Object || Name == "")
  return FoundedObjects;

  for (auto const& [RelationName, Objects] : Object->
      Relations)
  {
    if (RelationName == Name)
    {
      FoundedObjects = Objects;
    }
```

```
        else
        {
          for (pObject FoundedObject : Objects)
          {
            if (EntryObject == FoundedObject)
            {
              continue;
            }

            if (FoundedObject)
            {
              if (!EntryObject)
              FoundedObjects = FindObjectByRelation(FoundedObject
                 , Name, Object);
              else
              FoundedObjects = FindObjectByRelation(FoundedObject
                 , Name, EntryObject);
            }
          }
        }

        return FoundedObjects;
    }
```

3. Заключение

В ходе выполнения работы была успешно реализована программная библиотека для создания и анализа семантической сети. Разработанные структуры данных и рекурсивные алгоритмы позволяют эффективно представлять и обрабатывать объекты и их связи, загруженные из внешнего JSON-файла. Функциональность вывода и поиска, продемонстрированная в приведенных примерах кода и на рисунке 2, подтверждает корректность работы реализованной модели.

Список литературы

[1] Кнут Д.Э. Всё про TEX. — Москва: Изд. Вильямс, 2003 г. 550 с.

[2] Львовский С.М. Набор и верстка в системе LATEX. — 3-е издание, исправленное и дополненное, 2003 г.

[3] Воронцов К.В. LATEX в примерах. 2005 г.