哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：大数据计算基础

课程类型：必修

实验题目：基于Neo4j和DBpedia的知识图谱索引建立

学号：1190201024

姓名：魏宇鹏

目录

第一部分　绪论 **3**

一、 研究背景 **3**

二、 领域综述 **3**

1. 概述 3
2. 子图查询 4

三、 研究问题的挑战 **5**

四、 本文的工作要解决的问题以及方法 **5**

五、 本文的贡献 **6**

六、 文章结构 **6**

第二部分　系统/方法框架 **6**

七、 知识图谱和DBpedia **6**

八、 Neo4j的查找和索引 **9**

第三部分　技术 **10**

九、 Cypher语言 **10**

十、 Neo4j索引概述 **12**

十一、 B树 **13**

第四部分　实验 **14**

十二、 实验设计 **14**

1. 索引结构设计 14

2. 实验流程设计 16

十三、 实验过程 16

1. DBpedia导入Neo4j 16
2. 无索引下的搜索 17
3. 创建索引 18
4. 利用索引进行搜索 18
5. 进行对比，得出结论 18
6. 索引占用空间 19

第五部分 相关工作 19

第六部分 结论 20

第七部分 参考文献 20

第八部分 实验环境 21

第一部分　绪论

一、　研究背景

知识图谱（ Knowledge Graph）的概念由谷歌2012年正式提出，旨在实现更智能的搜索引擎，并且于2013年以后开始在学术界和业界普及。目前，随着智能信息服务应用的不断发展，知识图谱已被广泛应用于智能搜索、智能问答、个性化推荐、情报分析、反欺诈等领域。另外，通过知识图谱能够将Web上的信息、数据以及链接关系聚集为知识，使信息资源更易于计算、理解以及评价，并且形成一套Web语义知识库。知识图谱以其强大的语义处理能力与开放互联能力，可为万维网上的知识互联奠定扎实的基础，使Web 3.0提出的“知识之网”愿景成为了可能。

由于大数据的出现，计算能力的升级，涌现出以知识图谱为代表的一批大数据人工智能时代的产物。知识图谱的查询一直受到学术界和工业界的广泛关注，知识查询的效率与知识图谱的索引密切相关。

二、　领域综述

1.　概述

为了更好地管理和使用知识图谱，一个关键的任务就是对知识图谱进行查询。在知识图谱上的查询需要解决两个基本问题：查询的表达和查询的执行。由于知识图谱本质上也是一种图数据，因此知识图谱上的查询与传统图数据上的查询有着密切的联系；同时由于知识图谱所特有的结构和语义信息，二者又有一定的区别。

为了描述知识图谱上的查询，其关键问题是提升查询的表达能力，即充分表达实际应用中的语义信息和结构约束。国际标准化组织 W3C 提出了针对 RDF 知识图谱的标准化查询语言 SPARQL。作为一种图数据，SPARQL 查询的执行可以通过图匹配的方式实现。然而图匹配的计算复杂度较高，关键于查询的执行和实现，核心问题是如何降低时空开销，同时保证结果的正确性，因此需要设计合理的模型和算法。

2. 子图查询

SPARQL查询可以表达为一个规模较小的查询图，因此可以通过子图查询完成。子图查询是一个NP-hard问题，为了提高计算效率，设计有效的索引结构进行剪枝是非常关键的步骤。目前的索引策略大体上可以分为两类：**全局结构特征索引**和**局部邻居信息索引**。

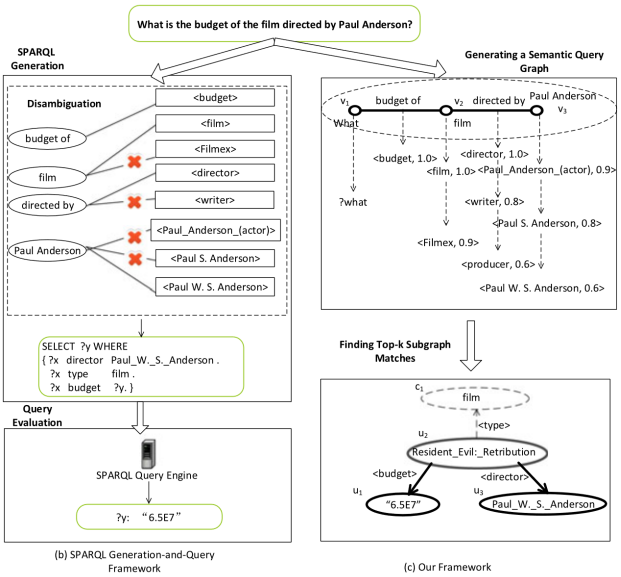

图 1: 自然语言转化子图[2]

1 全局结构特征索引 选取特殊的结构(如路径、子树或子图)作为特征，在离线处理阶段，这些结构特征在图数据库中出现的位置被记录下来，并被有效地组织和存储起来。在线查询阶段在给定了查询图 q 之后，与处理目标图类似，首先提取 q 中相应的结构特征，通过这些结构特征和离线阶段建好的索引结构可以有效筛选候选图。

其中的基本原理是:如果某个特征结果在 q 中出现，但是在某个图 G 中不存在，则可以确定 q 一定不会子图同构于图 G。显然，如果图数据库中有很多目标图，则不必逐一进行比较，根据索引结构就可以筛选出一些图作为候选答案。

例如，GraphGrep 选取所有长度值不大于某个值 maxL 的路径作为结构特征; glndex和 FG-index提出利用频繁子图来构建索引，相对于基于路

径的方法，其优势体现在频繁子图描述了更多的信息，因此具有较强的过滤能力，但其缺点也很明显，即空间开销较大；QuickSI 利用频繁子树作为结构特征，并且设计了有效的搜索顺序。[5]

2 局部邻居信息索引 也就是基于每个节点的邻居信息进行剪枝。其基本原理是，如果子图 q 同构于图 G，那么对于 q 中的每个节点 u，在 G 中存在一个节点v与其匹配，并且 u 的邻居结构也子图同构于 v 的邻居结构；反之，若 u的邻居结构与 v 的邻居结构不满足子图同构关系，则说明 v 不可能是 u 的候选匹配节点。这种方式可以筛选节点 u 的候选匹配节点。

GCoding 将每个节点周围的多跳邻居结构转换成树结构，然后把树结构的信息编码到数值空间，称作顶点签名(Verlex Signature)。SPath 把节点n跳以内的最短路径作为基本索引单元。NOVA 基于每个节点周围邻居的标签分布，设计了一个向量索引。SQBC 设计了一种融合局部邻居信息和结构特征的索引，其中局部邻居信息索引部分基于邻居标签和节点度数为每个节点编码。它采用团(Clique)作为结构特征，这样可以提升过滤能力并加速子图同构判定。

三、 研究问题的挑战

由于需要挖掘（频繁）子结构作为结构特征，基于结构特征的方法并不适用于规模很大的图。此外，基于结构特征的方法大多只用于过滤候选图，而较少用于匹配阶段的子图同构判定。

对于DBpedia知识图谱这样大的数据集而言，采用图的结构特征索引是不现实的。

四、 本文的工作要解决的问题以及方法

本文针对知识图谱规模巨大、查找资源消耗大的问题，给出了建立B树索引的解决方案。首先，本文给出了DBpedia导入Neo4j的一般流程。其次，针对大规模数据查询本文给出了建立索引的一种方式以及进行实验验证B树结构的索引对查询效率的提升。

在下面的部分中给出了一种基于Neo4j的知识图谱索引建立方法，并将简单查询的效率提升约77倍，控制所使用的索引空间大小为原数据库大小的1/4000。

五、 本文的贡献

1 系统总结了知识图谱的子图搜索和索引。

2 呈现了DBpedia导入Neo4j、增添索引等功能，并分析了Neo4j索引。

六、 文章结构

第一部分给出了关于子图查询相关问题的综述。第二部分介绍了本次实验使用的系统，主要是Neo4j和DBpedia。第三部分介绍了使用的相关技术，包括Cyhper和B树。第四部分叙述了实验流程、实验结果和结论。第五部分总结了关于图查询和索引的相关工作。最后给出结论和参考文献。

第二部分 系统/方法框架

七、 知识图谱和DBpedia

图 2: DBpedia部分本体模式一览[1]

DBpedia 是一项众包社区工作，旨在从 Wikipedia 和 Wikidata 中提取结构化信息并使这些信息在 Web 上可用(Lehmann 等，2015 年)。通过提

取维基百科的结构化内容，例如信息框、表格、列表和分类数据，并将它们放入一致的知识库中，可以回答维基百科内部文本搜索引擎不允许的表达性查询，例如：”Give me all Italian musicians from the 18th century”。

DBpedia数据系统包括两个基本部分[1]：模式(schema,内涵部分),包含类别、属性等和数据集(外延部分)包含序数化和定位了的数据集。在知识数据表示模型RDF中，使用IRI标识对象，并使用统一的命名空间http://dbpedia.org/resource/NAME，每个DBpedia实体的名字NAME都导向wiki文章，并由多个属性描述，例如标签、wiki网页链接和描述图片。

Dataset	Description
Mapping-based Types	Contains the *rdf:types* of the instances which have been extracted from the infoboxes
Mapping-based Properties	Contains the actual data values that have been extracted from infoboxes (e.g., 'volume').
Mapping-based Properties (Specific)	Contains properties which have been specialized for a specific class using a specific unit. e.g. the property height is specialized on the class Person using the unit centimetres instead of metres.
Titles	Contains the *rdfs:labels* representing the title of the corresponding Wikipedia page
Short Abstract	Contains the short abstract for each DBpedia resource. It is the value of the *rdfs:comment*property
Extended Abstracts	Contains the long abstract for each DBpedia resource. It is the value of the *dbpedia.org/ontology/abstract* property.
External Links	Contains the wikipedia External links. For example, the books.google.com URL for a DBpedia Book resource.

图 3: DBpedia数据集文件

其中较为重要的是Mapping-based Types和Mapping-based Properties部分.Types文件按如下形式存储rdf三元组:
(subject,rdf:type,object),Properties按(subject property object)存储，该文件当前的版本被称作mapping-based objects.

DBpedia的文件可以在https://downloads.dbpedia.org/repo/dbpedia/ 处获取。其中我们只关注/generic和/mappings部分。在每个目录中，都有.md文件说明该文件的用途，下面略列举如下。

1	generic	主要部分，包括链接和属性
2	mappings	映射关系
3	spotlight	dbpedia工具， 用于在文本中自动注释 DBpedia 资源的提及。
4	text	包括概述、文本中出现的公式和表格
5	transition	目前尚未完全建立
6	wikidata	包含页面、标签、描述等

表 1: dbpedia一级目录

本实验导入的包在dbpedia_files.txt中。 DBpedia的实例通过两个属性：自动生成的id和uri与Neo4j节点建立映射关系，节点之间的关系由Mapping及其他文件以三元组的形式赋予。

图 4: DBpedia属性层级

八、 Neo4j的查找和索引

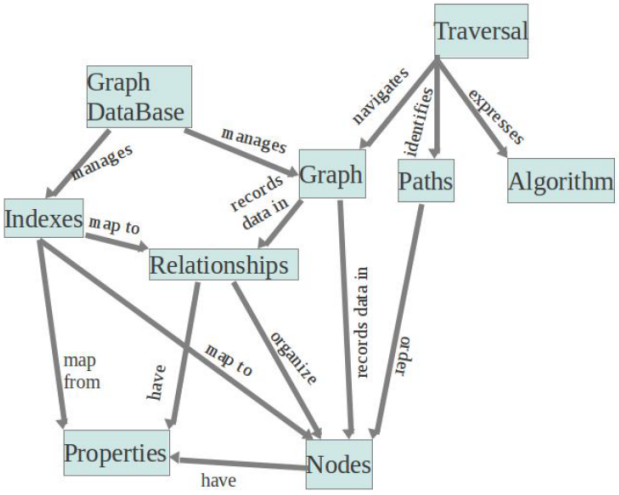

图 5: Neo4j图数据库结构

Neo4j 图数据库，如图 2 所示，能后灵活的管理图及其连接的索引。一次查询过程标识它必须为查询响应标识的节点的顺序，因此它对图遍历进行。建立索引可以提搞查询遍历期间的性能潜力。索引有助于从节点关系映射到节点属性(或相反)。索引查找是为了快速查询响应而导航的最简单的路径。数据库通过 Cypher 查询语言查询。 Cypher 查询遍历从起始节点导航到相关节点，在多维索引的帮助下找到快速响应[4]。

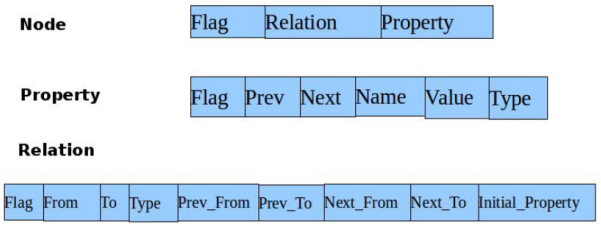

图 6: Neo4j表格式

Neo4j 专门处理优化的图结构数据，由具有元数据信息的节点和节点之间的边链接组成。该图模型由用于查询处理的多维索引技术组成。该索引

框架与存储基表的后端存储服务交互，并为更快的查询处理应用程序提供经济的数据检索接口。 Neo4j 的主要表具体构成了节点、关系和属性，如图 3 所示。节点是实体，其中每个实体都有自己的属性并指定与其他节点的关系。每个节点都有标签，此标签有助于按每个节点的角色和属性对节点进行聚类。关系连接实体并构建节点域模型。每个节点的属性清楚地建立了与该节点相关的属性以及节点处理的元数据。

第三部分　技术

九、　Cypher语言

1	CREATE (nname : key, mapp, mapr)	creation of new node
2	CREATE (n)-[r]->(m)	node n belongs to node m, r relation
3	START n = node(key)	Start from node with specified key value.
4	START n = node : nodeIndexName (key =value)	Query the index with node auto index (Multidimenstional Index)
5	MATCH (n)–(m)	Maps the node n with node m
6	ORDER BY n.property	Sort the result
7	RETURN DISTINCT n	Return unique rows

表 2: 用到的Cypher语言

Neo4j使用Cypher语言进行查询和数据库管理。一个查询语句的实例为：

match (n {uri: 'http://dbpedia.org/resource/Aristotle'})

-[r: rdf__type]-(m)

return n,r,m limit 20 结果是输出20个与亚里士多德有type关系的节点：包括Philosopher，Agent，Person以及相关资源（(wiki页面、图片等)。对于点本身properties的搜索可以通过

match (n uri:'http://dbpedia.org/resource/Socrates')

-[r:ns0__birthPlace] -(m)

return n,r,m limit 25 进行。另外，由于id是唯一的，可以利用where

id(n) = 1447554 搜索。

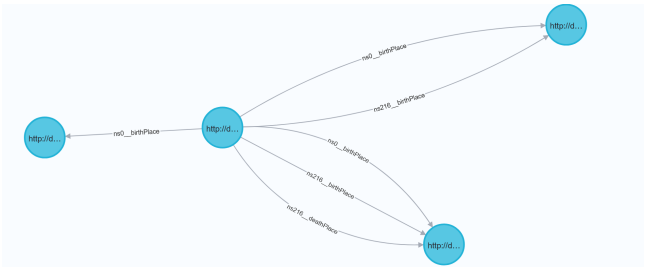

图 7: 苏格拉底出生地的搜索，返回的是Athens，Deme，Alopece

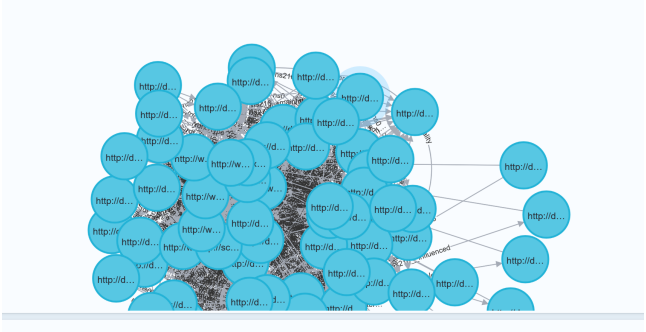

图 8: 对苏格拉底具有关系的搜索

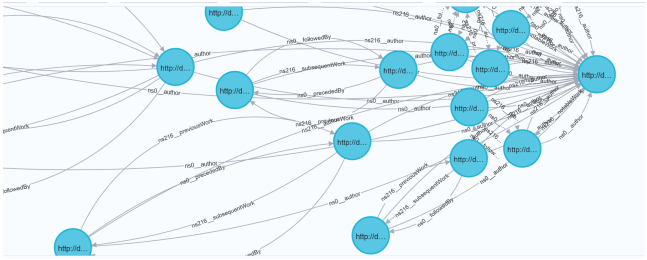

图 9: 与Agatha Christie具有author关系、不超过四层的20个节点

neosemantics (n10s)库 neosemantics (n10s) [新语义库]是一个插件，可以在 Neo4j 中使用 RDF 及其相关内容，如(OWL、RDFS、SKOS 等)。由于知识图谱用IRI组织，其中每个部分都是链接，neosematics提供了接口用以快捷映射RDF为节点和关系。

Py2neo库 Py2neo 是一个客户端库和工具包，用于在 Python 应用程序和命令行中使用 Neo4j。该库支持 Bolt 和 HTTP，并提供高级 API、OGM、管理工具、交互式控制台、Pygments 的 Cypher 词法分析器以及许多其他功能。

十、 Neo4j索引概述

有效地从图中检索信息需要所谓的遍历。图的遍历涉及 "行走"遵循图的元素。遍历是数据检索的一个基本操作，Neo4j通过广度优先遍历。遍历和SQL查询之间的一个主要区别是，遍历是本地化的操作。Neo4j没有全局的邻接索引，而是图中的每个顶点和边都存储了一个与之相连的对象的 "迷你索引",这意味着，图的大小遍历时没有性能上的影响，而且在SQL JOIN语句中进行的昂贵的分组操作也没有必要。值得注意的是，Neo4j中确实存在全局索引，但它们只在试图找到遍历的起点时使用。遍历的起点时才使用。

为了根据顶点的值快速检索顶点，索引是必需的。它们提供了一个它们提供了一个开始遍历的起始点。如果没有索引，确定一个特定的元素是否有一个特定的属性将需要对所有元素进行线性扫描，成本为O(n)，n是集合中元素的数量。或者说查阅索引的成本要低得多，为O(log2n) 。

由于知识图谱数据结构特点(每个实体都有独一无二的uri)，Neo4j在导入时就加入了对于 uri 的索引。实际使用中，由于鲜少对uri直接搜索，我们将不再考虑这部分内容的索引。

Neo4j有多种索引类型可用：b-tree、全文(full-text)、查找(lookup)和文本(text)索引类型。在4.0版本中加入了对关系(relationship)创建的索引的支持。从官方文档得知，Neo4j的(文本)索引支持是基于Lucene的，而Lucene使用FST保存词典。

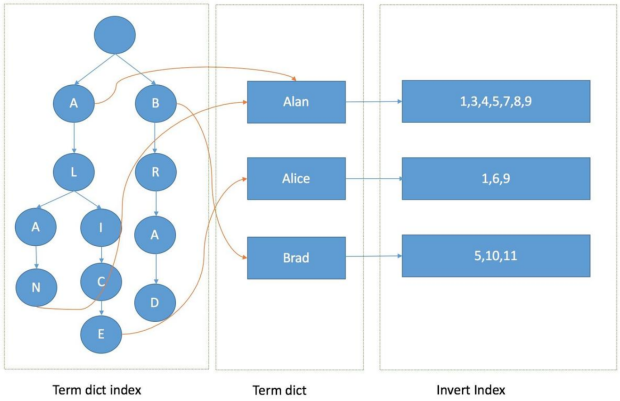

图 10: FST索引结构

官方文档中针对任何给定标签或关系类型在单个属性上创建的索引称为单一属性索引。

针对任何给定标签或关系类型在多个属性上创建的索引称为复合索引。

文本索引 在本节中对文本索引进行大概介绍。与 b-tree索引不同，文本索引是一种单属性索引，并且仅索引具有字符串值的属性。它们专门用于有效处理 ENDS WITH 或 CONTAINS 查询。它们通过 Cypher 使用，并且支持较小的字符串查询集。尽管文本索引确实支持其他文本查询，但只有 ENDS WITH 或 CONTAINS 查询是这种索引类型比 b-tree 索引具有优势的查询。

十一、 B树

概念 B树是平衡树，和平衡二叉树不同的是B树属于多叉树，又名平衡多路查找树(查找路径不只两个)，数据库索引技术里大量使用着B树和B+树的数据结构，其特点：

规则

1 排序方式:所有节点关键字是按递增次序排列，并遵循左小右大原则；

2 子节点数：非叶节点的子节点数>1，且<=M ，M>=2，空树除外（注：M阶代表一个树节点最多有多少个查找路径，M=M路,当M=2则是2叉树,M=3则是3叉）；

3 关键字数：枝节点的关键字数量大于等于ceil(m/2)-1个且小于等于M-1个

4 所有叶子节点均在同一层、叶子节点除了包含了关键字和关键字记录的指针外也有指向其子节点的指针,只不过其指针地址都为null。

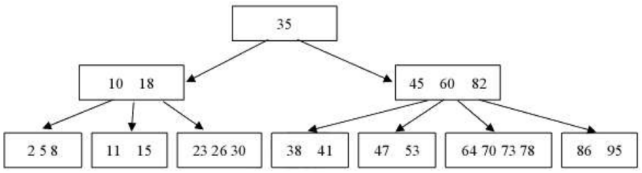

图 11: 一棵5阶B树

B树索引的建立被附在代码部分中，分别是AbstructBTree.java, BTree.java, KeyEntry.java, TreeNode.java。

第四部分　实验

十二、　实验设计

1.　索引结构设计

查询问题　在query2.txt中记录有一些著名作家的名字，共计79人（有少数重复）。

图 12: query部分文件内容

在query1.txt中记录有这些名字的错误版本，目的是为了进行[无法找到]的查询。查询使用py2neo进行，并记录时间。

索引 使用B-Tree结构的索引，索引的数据结构在代码中的几个java文件内。实际使用时，可以利用Neo4j提供的接口建立。针对每个节点的属性ns0__name (DBpedia解释该属性为实体名，具体的人名属于属性ns216__name) 建立索引，并通过:Schema命令查询：
Cypher语言： Match (n:Resource ns0__name:"...") 仍有一个问题，即搜索对象的同一性问题。比如说，我们通常说的《唐璜》的作者George Byron在wikidata里ns0_name(资源名)被记录为The Lord Byron(拜伦勋爵)，如果导入的重定向库中没有建立两者的关系可能会导致查询不到结果.对此改进方法是使用ns217_name(人名)查询，但是这里会发生多一部分的情况。比如Byron被记录为George Gordon Byron。在具体的The Lord Byron词条中，记录其birthName是我们熟知的，但对于很多作家而言流传的都是笔名。

该问题的解决方式之一是使用contains搜索：考虑这个名字的firstname和lastname，如果And(Contains(firstname), Contains(lastname))=true 就认为这个作家被搜索到。(或者，当全部搜索不奏效时使用该策略，这会导致两倍的搜索时间。)

在改进搜索策略中，就能搜索到全部作家信息了。


```
{"ns0__n":"no","ns0__cursign":"","ns0__deathDate":"1976-01-12","ns0__q
":"Agatha Christie","ns0__r":-5,"ns0__s":"Author:Agatha Mary Clarissa
Christie","ns0__viaf":71388952,"ns0__index":"UK","ns0__v":"no","ns0__r
estingPlace":"Church of St Mary, Cholsey, Oxfordshire, England","ns0__
notableworks":"The Murder of Roger Ackroyd","ns0__pseudonym":"Mary Wes
tmacott","ns0__label":"Agatha Christie","ns0__spouses":1930,"ns0__star
tYear":1998,"ns0__value":50,"ns0__align":"right","ns0__source":"Agatha
 Christie","ns0__alt":"Black and white portrait photograph of Christie
 as a middle-aged woman","ns0__birthName":"Agatha Mary Clarissa Miller
","ns217__name":"Agatha Christie","ns216__deathDate":"1976-01-12","ns2
16__birthDate":"1890-09-15","ns0__onlinebooksby":"yes","rdfs__label":"
Agatha Christie","ns0__birthPlace":"Torquay, Devon, England","uri":"ht
tp://dbpedia.org/resource/Agatha_Christie","ns0__wikt":"no","ns0__quot
e":"The lure of the past came up to grab me. To see a dagger slowly ap
pearing, with its gold glint, through the sand was romantic. The caref
ulness of lifting pots and objects from the soil filled me with a long
ing to be an archaeologist myself.","ns0__name":"Agatha Christie","ns0
__author":"no","ns0__width":"30.0","ns0__deathPlace":"Winterbrook Hous
e, Winterbrook, Oxfordshire, England","ns0__birthDate":"1890-09-15","n
s216__pseudonym":"Mary Westmacott","ns0__b":"no","ns216__birthName":"A
gatha Mary Clarissa Miller","ns0__c":":Category:Agatha Christie","ns0_
_signature":"Agatha Christie's signature.png","ns0__by":"yes","ns0__oc
cupation":"Novelist, short story writer, playwright, poet, memoirist",
```


图 13: 关于Agatha Christie的信息

图 14: :Schema命令查询如图

2. 实验流程设计

1 DBpedia导入Neo4j

2 无索引下的搜索

3 创建索引

4 利用索引进行搜索

5 进行对比，得出结论

6 索引占用空间

十三、 实验过程

1. DBpedia导入Neo4j

从 https://downloads.dbpedia.org/repo/dbpedia/ 下载dbpedia资料文件，导入列表：dbpedia_files.txt 列表内容可以调整，但必须包括mapping和infobox。

问题：DBpedia 包含格式错误的 IRI，我已尽力排除它们，但仍有一些可以通过。另外DBpedia 具有类型不一致的多值属性，例如，关于birthDate包括Date格式的数据和Integer类型的(年份)。解决方案之一是本次实验使用的 handleMultival: ”OVERWRITE” 选项。

在linux虚拟机上执行文件夹中的get-neo4j.sh，下载Neo4j 4.3.6及neosemasitic 4.3.0.2版本。然后执行download-dbpedia.sh， 下载depedia_files.txt目录中的包。最后使用import-dbpedia.sh，或者使用 import-packages.py 导入。

导入后，在Neo4j-Browser中即可看到导入了30,383,632个节点以及123,376,700条边.

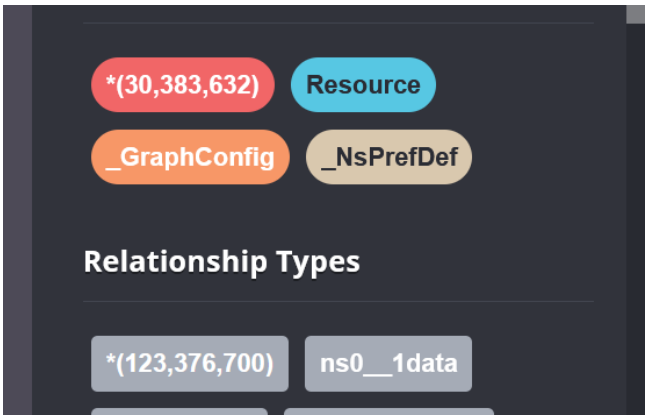

图 15: Neo4j浏览器查看导入结果

2. 无索引下的搜索

执行work1.py进行五次实验，记录如下：

有内容搜索用时(s)	无内容搜索用时(s)	平均(有内容)	平均(无内容)
2705.18	2701.39	34.68	34.63
2691.51	2694.47	34.51	34.54
2685.33	2706.99	34.43	34.7
2684.36	2693.95	34.41	34.54
2679.24	2685.42	34.35	34.43
平均			
2686.124	2696.44	34.47	34.56

表 3: 无索引搜索78条实体用时

可以看出，有、无内容搜索的时间是几乎相同的。这是因为Neo4j默认遍历所有的内容进行搜索，可以通过指定Limit+NUM来限制搜索结果数量改善。其次，从结果上看这5次实验返回的结果都是相同的。在准确率的角度，具有acid属性的Neo4j可以几乎100%返回正确结果(但未必是需要结果)。

3. 创建索引

对ns0__name属性创建B-tree索引。使用:schema命令验证创建是否成功。重启Neo4j，再次执行查询程序。

4. 利用索引进行搜索

执行work2.py进行五次实验，记录如下：

有内容搜索用时(s)	无内容搜索用时(s)	平均(有内容)	平均(无内容)
17.39	15.74	0.223	0.202
20.04	20.44	0.257	0.262
18.87	15.89	0.242	0.204
16.39	15.71	0.21	0.201
15.50	15.88	0.199	0.204
平均			
17.63	16.73	0.225	0.215

表 4: 有索引搜索78条实体用时

直观上，即看出二者相差非常大。下面用表格列出了实验结果的对比：

5. 进行对比，得出结论

效率提升公式 提升率

$$X = \frac{1/T_{IndexMatching} - 1/T_{No-indexMatching}}{1/T_{No-indexMatching}}$$

即

$$\frac{T_{No-indexMatching}}{T_{IndexMatching}} - 1$$

实验次数	有结果效率提升	无结果效率提升
1	66.93	59.08
2	82.15	99.69
3	88.86	78.05
4	81.78	76.39
5	68.82	74.62
平均		
	77.708	77.566

表 5: 索引效率提升

由于方差是比较大的，可以看出该实验结果仍有进一步的探索空间。其中一个原因是因为相比于原先的查询时间来说，使用索引的时间大大减小，从而细微的差别也会引起倍数的改变。

6. 索引占用空间

在Neo4j中利用:schema命令显示所有schema索引。 Neo4j将索引文件储存在database storeNAME schema index 目录当中。查看该文件：

图 16: 索引文件

大小是71264KB，也就是71.2MB。数据大小约为29.2G(ttl文件)，索引:数据约为4101.1,而提升效率约为77倍左右(使用索引查找时间为原先的1/78)。因此指出，知识图谱使用B-tree索引能够很好的提高搜索的效率同时不占用太多空间。

第五部分　相关工作

全局结构特征索引　GraphGrep 选取所有长度值不大于某个值 maxL 的路径作为结构特征; glndex和 FG-index提出利用频繁子图来构建索引，相对于基于路径的方法，其优势体现在频繁子图描述了更多的信息，因此具有较强的过滤能力，但其缺点也很明显，即空间开销较大；QuickSI 利用频繁子树作为结构特征，并且设计了有效的搜索顺序。[5]

局部邻居信息索引 GCoding 将每个节点周围的多跳邻居结构转换成树结构，然后把树结构的信息编码到数值空间，称作顶点签名(Verlex Signature)。SPath 把节点n跳以内的最短路径作为基本索引单元。NOVA 基于每个节点周围邻居的标签分布，设计了一个向量索引。SQBC 设计了一种融合局部邻居信息和结构特征的索引，其中局部邻居信息索引部分基于邻居标签和节点度数为每个节点编码。它采用团(Clique)作为结构特征，这样可以提升过滤能力并加速子图同构判定。

其他图数据库 gStore是由北京大学-王选计算机研究所-数据管理实验室邹磊教授团队研发面向RDF知识图谱的开源图数据库系统。使用不同于传统基于关系数据库的知识图谱数据管理方法，gStore原生基于图数据模型(Native Graph Model)，维持了原始RDF知识图谱的图结构；其数据模型是有标签、有向的多边图，每个顶点对应着一个主体或客体。gStore将面向RDF的SPARQL查询，转换为面向RDF图的子图匹配查询，利用引用文章中提出的基于图结构的索引**VS-tree**来加速查询的性能。[3]

第六部分　结论

本文给出了一种基于Neo4j的知识图谱索引建立方法，并将简单查询的效率提升约77倍，控制所使用的索引空间大小为原数据库大小的1/4000。进一步的工作在于：使用其他结构索引，例如基于FST数据结构的索引，能否起到更好的效果？

第七部分　参考文献

参考文献

[1] Enrico G Caldarola, Antonio Picariello, Antonio M Rinaldi, and Marco Sacco. Exploration and visualization of big graphs. In *Proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 257–264, 2016.

[2] Sen Hu, Lei Zou, Jeffrey Xu Yu, Haixun Wang, and Dongyan Zhao. Answering natural language questions by subgraph matching over knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 30(5):824–837, 2018.

[3] Lei, Chen, M., Tamer, OEzsu, Ruizhe, Huang, Lei, Zou, and Dongyan and. gstore: a graph-based sparql query engine. *Vldb Journal the International Journal of Very Large Data Bases*, 2014.

[4] Anita Brigit Mathew and SM Kumar. An efficient index based query handling model for neo4j. *IJCST*, 3(2):12–18, 2014.

[5] 肖仰华. *知识图谱: 概念与技术*. 电子工业出版社, 2020.

第八部分　实验环境

1 Window11, AMD 5600U, 16G RAM

2 Ubuntu 21.04, 8G RAM

3 Neo4j community, version4.3.6

4 Python 3.8.0