COMS E6111 Advanced Database Systems
Spring 2015

Project 2

Due Date: Friday March 27 at 5 p.m. ET
(same deadline for all students, on-campus and CVN)

Teams

You will carry out this project in **teams of two**. You do not need to notify us now of your team composition. Instead, when you submit your project you should indicate in your submission your team composition, which can be different from that of Project 1. Both students in a team will receive the same grade for Project 2. Please check the Collaboration Policy Web page for important information on what kinds of collaboration are allowed for projects, and how to compute the available number of project grace days for a team.

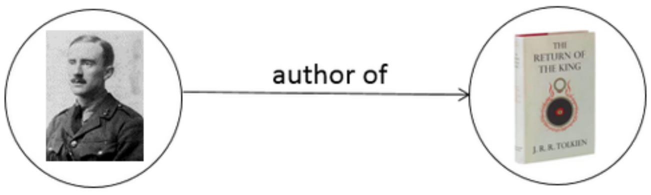
Important Notes:

- Please be considerate and notify your Project 1 teammate immediately if you decide to change teams for Project 2; similarly, do not wait until the day before the project deadline to contact your former teammate just to find out that s/he decided unilaterally to form another group.
- Please follow the same guidelines as in Project 1 to find a teammate if you need to.
- You can do this project by yourself if you so wish. Be aware, however, that you will have to do exactly the same project as two-student teams will.
- For this project, you have a choice: you can write your code in **either Java or Python**.

Background

In this project, you will learn and play with tools and technology that modern search engines are rapidly incorporating to answer user queries more effectively than by just returning a ranked list of webpages.

Specifically, you will be using a ***knowledge graph***, which is a graph encoding important information about the real world and which can be used to answer queries effectively. The vertices of the graph correspond to "entities" (interpreted broadly), while the directed edges of the graph correspond to the "relationships" that semantically connect these entities, if any. For example, both "Lord of the Rings - The Return of the King" and "J. R. R. Tolkien" have corresponding vertices that are connected with the edge "is author of" to represent the fact that J. R. R. Tolkien wrote the "Lord of the Rings" novel.

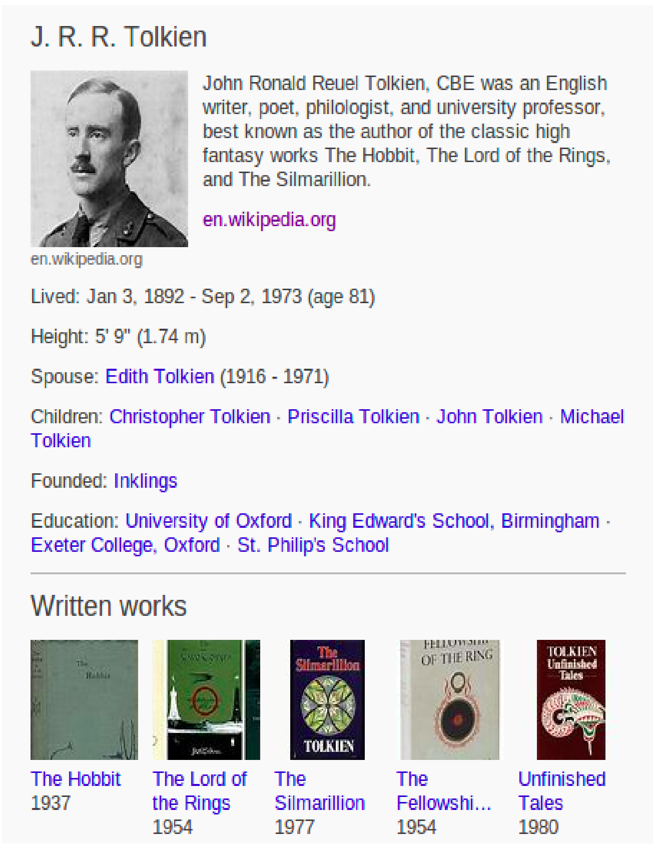


"J. R. R. Tolkien" (left) and "Lord of the Rings - The Return of the King" (right)
as connected in a knowledge graph with an "author of" relationship (image source: Wikipedia).

Search engines exploit knowledge graphs to provide meaningful and often "structured" information in response to user queries. We can exploit knowledge graphs in the context of web search in different ways, including through ***infobox creation*** and ***question answering***, which you will focus on for this project.

Part 1: Infobox Creation

If you search Google or Bing with query [tolkien], on the right side of the result page you will find an appropriate *infobox* for J.R.R. Tolkien, the most likely match for your query. Such an infobox provides structured information (e.g., date of birth, books written, etc.) about the author:


J. R. R. Tolkien

John Ronald Reuel Tolkien, CBE was an English writer, poet, philologist, and university professor, best known as the author of the classic high fantasy works The Hobbit, The Lord of the Rings, and The Silmarillion.

en.wikipedia.org

en.wikipedia.org

Lived: Jan 3, 1892 - Sep 2, 1973 (age 81)

Height: 5' 9" (1.74 m)

Spouse: Edith Tolkien (1916 - 1971)

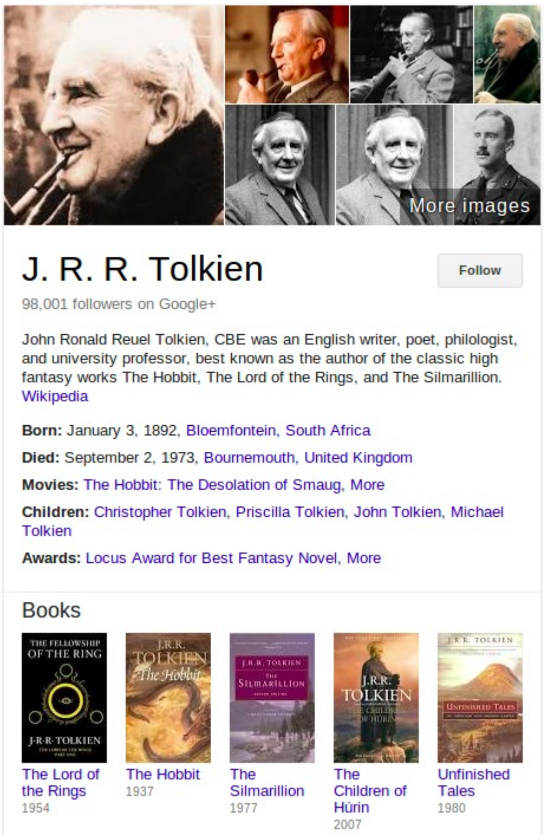
Children: Christopher Tolkien · Priscilla Tolkien · John Tolkien · Michael Tolkien

Founded: Inklings

Education: University of Oxford · King Edward's School, Birmingham · Exeter College, Oxford · St. Philip's School

Written works

The Hobbit 1937 · The Lord of the Rings 1954 · The Silmarillion 1977 · The Fellowshi... 1954 · Unfinished Tales 1980


J. R. R. Tolkien

Follow

98,001 followers on Google+

John Ronald Reuel Tolkien, CBE was an English writer, poet, philologist, and university professor, best known as the author of the classic high fantasy works The Hobbit, The Lord of the Rings, and The Silmarillion. Wikipedia

Born: January 3, 1892, Bloemfontein, South Africa

Died: September 2, 1973, Bournemouth, United Kingdom

Movies: The Hobbit: The Desolation of Smaug, More

Children: Christopher Tolkien, Priscilla Tolkien, John Tolkien, Michael Tolkien

Awards: Locus Award for Best Fantasy Novel, More

Books

The Lord of the Rings 1954 · The Hobbit 1937 · The Silmarillion 1977 · The Children of Húrin 2007 · Unfinished Tales 1980

Infobox returned by Bing for query [tolkien]. *Infobox returned by Google for query [tolkien].*

As Part 1 of the project, you will implement a simple structured search system that receives a keyword query and returns as a result one appropriate infobox, as discussed below, by using the large-scale, state-of-the-art Freebase knowledge graph, at http://www.freebase.com/. To keep the project to a manageable size, we will restrict the kind of queries that you are expected to handle, as well as the kind of output that you should produce, as we specify below.

For Part 1 of the project, your system should:

1. Receive as input a string *q*, representing a user query.
2. Query Freebase through the Freebase Search API with query *q*, to obtain a list of entities ranked according to the relevance to the query, from most relevant to least relevant. For each entity in the ranked list, Freebase returns an associated identifier, referred to as the entity's *mid*. (Freebase also returns 5 additional attributes for the entity that you can ignore, namely, *id*, *lang*, *name*, *notable(id, name)*, and *score*.)

   For example, here are the results of the Freebase Search API for query [tolkien] (we use the square brackets to denote queries; you should not include the brackets when contacting the API), as of February 22, 2015: tolkien.search. [tolkien] is an ambiguous query that could refer to multiple entities, and the Freebase Search API returns as the top match the entity J.R.R. Tolkien, arguably the most likely intended meaning for the query.

   Note that when you use the Freebase Search API you have to "URL-escape" the input query *q* first, to avoid problems with any special characters that the input query might contain. For more information and examples on how to query the Freebase Search API, see the Freebase Usage section below. Other than this URL-escaping, you should not modify query *q* in any way before passing it to the Freebase Search API.
3. Query Freebase through the Freebase Topic API with the *mid*s of the entities returned in step 2, to obtain properties of interest of these entities. Different types of entities (e.g., a person, a sports team) will have different properties of interest associated with them (e.g., birthday for a person, championships for a sports team). The Freebase Topic API returns one or more values of a property named */type/object/type* that indicates the type or types of the corresponding entity and, in turn, this determines the properties of interest associated with the entity.

   For our example, here are the results of the Freebase Topic API for *mid=/m/05qdh*, corresponding to the top entity for query [tolkien], as of February 22, 2015: tolkien.topic. The */type/object/type* property of the results includes the values */people/person* and */book/author*, which indicate that J.R.R. Tolkien was a Person and an Author, respectively, and then the properties of interest for these two types of entities apply.

   For a complete reference on how to read the */type/object/type* property as well as how to interpret the Freebase Topic API results in general, see the Freebase Usage section below.
4. Create and return an infobox for the **first** entity that corresponds to one of our 6 types of interest (see below) using the properties of interest of step 3. Your infobox will consist of the properties that make sense for the specific type or types of the entity at hand, as we discuss below.

   To help perform step 4 above, which is the most challenging step in the infobox creation process, you will consider only 6 high-level types of entities, namely, Person, Author, Actor, BusinessPerson, League, and SportsTeam, each with a specific set of properties of interest, as outlined in the following table:

Type of Entity	Properties of Interest
Person	Name, Birthday, Place of Birth, Death(Place, Date, Cause), Siblings, Spouses, Description

Author	Books(Title), Book About the Author(Title), Influenced, Influenced by
Actor	FilmsParticipated(Film Name, Character)
BusinessPerson	Leadership(From, To, Organization, Role, Title), BoardMember(From, To, Organization, Role, Title), Founded(OrganizationName)
League	Name, Championship, Sport, Slogan, OfficialWebsite, Description, Teams
SportsTeam	Name, Description, Sport, Arena, Championships, Coaches(Name, Position, From, To), Founded, Leagues, Locations, PlayersRoster(Name, Position, Number, From, To)

Types of entities that we will consider for the project and their associated properties of interest.

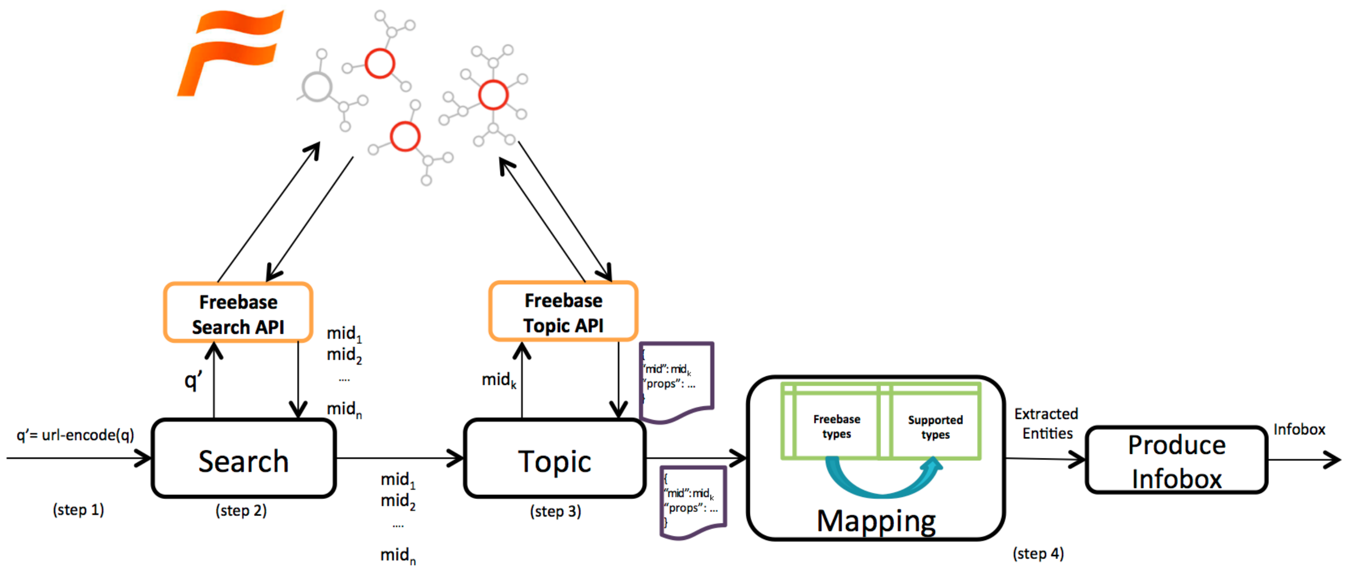
Each Freebase type, as returned by the Freebase Topic API as the value or values of the */type/object/type* property (see step 4 above), gets mapped to one of the 6 high-level types above, as follows:

Freebase Entity Type	Type of Entity for our Project
/people/person	Person
/book/author	Author
/film/actor **/tv/tv_actor**	Actor
/organization/organization_founder **/business/board_member**	BusinessPerson
/sports/sports_league	League
/sports/sports_team **/sports/professional_sports_team**	SportsTeam

Mapping of Freebase entity types to the 6 high-level types that we use in this project for infobox construction.

IMPORTANT NOTE: The top-ranked entities returned by Freebase might correspond to types of entities that we don't cover in this project; in this case you should ignore these entities and move on to the first entity in the rank that corresponds to one of our 6 types. If no such entity exists, then you should not return an infobox for the query, following the behavior of our reference implementation, which we discuss below.

Given the mapping of the types of entities, the main goal of step 4 is to map the properties of the Freebase Topic API result to the properties of interest. For instance, if the mapped type is **Author**, as is the case for J.R.R. Tolkien, you should extract the titles of the books he created, the titles of the books written about this author, the names of the people he influenced, and the names of people he was influenced by (see table above).



Infobox creation pipeline: As step 1, your system receives as input a free-text query and "URL-escapes" it. As step 2, your system sends the escaped query to the Freebase Search API and receives in return the most relevant entities, each with its corresponding mid. As step 3, your system sends the mid for the top-ranked entities to the Freebase Topic API and receives in return a JSON document with the properties for the entity. As step 4, your system uses the JSON document to decide what supported types should be used to build the output infobox, based on the Freebase types listed on the JSON document. Finally, your system builds and returns the appropriate infobox, based on the properties of interest of the supported types for the entity, and the values for such properties as returned by Freebase.

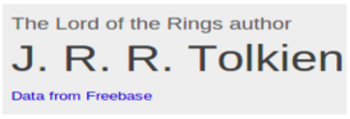
For more information and examples on how to use the Freebase Search and Topic APIs, see the Freebase Usage section below. To help clarify this part of the project further, here is a table with examples of entities, their Freebase *mid*s, their Freebase types, their associated entity types of interest, the results provided by the Freebase Topic API, and the infobox provided by the reference implementation.

Entity	Freebase *mid*	Freebase Type(s)	Entity Type(s)	Topic API Result	Reference Implementation Results
Bill Gates	/m/017nt	/people/person /organization/organization_founder /business/board_member /book/author	Person BusinessPerson Author	Bill Gates.topic	Bill Gates.infobox
Robert Downey Jr.	/m/016z2j	/people/person /tv/tv_actor /film/actor	Person Actor	Robert Downey Jr.topic	Robert Downey Jr.infobox
N.B.A.	/m/05jvx	/sports/sports_league	League	NBA.topic	NBA.infobox
N.Y. Knicks	/m/0jm3v	/sports/sports_team /sports/professional_sports_team	SportsTeam	N.Y. Knicks.topic	N.Y. Knicks.infobox

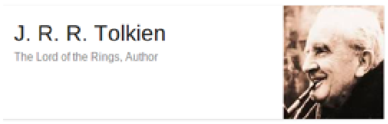
Examples of entities, their Freebase mids, their Freebase types, their types for our project, the results provided by the Freebase Topic API, and the infobox *produced by our reference implementation*.

Part 2: Question Answering

Modern search engines increasingly attempt to provide concrete answers to queries that are identified as questions, rather than (or in addition to) returning a traditional rank of webpages as the query result. For instance, if you issue query/question [Who created the Lord of the Rings?] to Google or Bing, you can expect to receive the actual answer among your search results, as follows:


The Lord of the Rings author
J. R. R. Tolkien
Data from Freebase

Bing's answer to query [Who created the Lord of the Rings?].


J. R. R. Tolkien
The Lord of the Rings, Author

Google's answer to query [Who created the Lord of the Rings?].

For this, search engines need to (a) understand the question type (e.g., who created a book), (b) infer the answer type (*The Lord of the Rings* is a book and thus it can be created by an author, which should be the answer), (c) transform the question into appropriate queries for the search engine's knowledge graph, (d) send the queries to the knowledge graph, (e) receive the results, and (f) present the results to the user in an appropriate form.

As Part 2 of this project, you will implement a simple question-answering system for a very restricted class of questions, namely, **"Who created [X]?"** questions. Examples of this class of question include:

- Who created Lord of the Rings?
- Who created Google?
- Who created Microsoft?
- Who created Romeo and Juliet?

Your system can just focus on strings that match this question pattern, and dismiss other queries just as we do in our reference implementation (see below). After identifying the input as a question of the right type, your system shoud query Freebase to attempt to answer the question. Specifically, the language you should use to generate the queries is called *MetaWeb Query Language* (MQL), a language created by Freebase. To execute your MQL queries against Freebase you will use the Freebase MQLRead API. **An online MQL query editor as well as many MQL queries examples can be found here.** Moreover, for your reference the MQL API guide can be found here. A quick reference on how to use the MQLRead API for this project is in the Using the Freebase MQLRead API section below.

When your system is presented with a [Who created [X]?] query, you should reply with the name of the person who created either X or something that matches X because its name has X as a substring (e.g., Microsoft Corporation should be considered when X=Microsoft, because Microsoft is a substring of Microsoft Corporation). To understand how to interpret "created," note that a business person is a founder of (has created) a company, while an author has written (has created) a set of books. Specifically, for this project you should focus only on the following two types of "creation" relationships:

Creation Relationship	*X*	Answer Type
An author can create a book	Book Title	Author
A businessperson can create an organization	Organization	BusinessPerson

Who can create what?

Your output for an input question should be of the form "*<Name> (as <Answer Type>) created <X_1> [, <X_2>, ... , and <X_n>].*" If your output contains multiple lines, corresponding to multiple "names," then sort the lines alphabetically by *<Name>*.

To produce the output for a question [Who created [X]?], you should build an MQL query to search for created objects (i.e., books and organizations) whose name contains the string X as a substring. For this, your system should enable the "contain" functionality using the ~ MQL operator (see MQL Operators) instead of using exact match. For instance, for the question "Who created Microsoft?" your system should reply, among others, that Bill Gates created "Microsoft Corporation." For you convenience, we provide an MQL query example in the Using the Freebase MQLRead API section.

Notice that a person of certain type (i.e., Author or BusinessPerson) might have created multiple objects whose name contain the string X, and your output should include all of these matching creations (<X_1> [, <X_2>, ... , and <X_n>]) in a single line. For instance, Bill Gates has created "Microsoft Corporation" and "Microsoft Research," so your output should be "Bill Gates (as BusinessPerson) created <Microsoft Corporation> and <Microsoft Research>." Also, note that a person might be both an author and a businessperson with matching created objects in each separate role. In such a case, your output should include two lines for such a person, namely, one line for each role with the corresponding matching created objects.

Freebase Basics

In order to use the Freebase APIs (Search, Topic, and MQLRead) you should create an API key that you will pass as a parameter with your requests to the Freebase APIs. To create an API key, go to the Google API console following this link in order to create a new project, then enable the Freebase API for this project, and finally create an API key. You can create/find your API key through:

Google Console (for the created project) > APIs & auth > Credentials > Public API access

To understand how to connect to Freebase and make proper use of the APIs programmatically, you should visit the Freebase official site. There you can find the developer's guide and references for each API, including examples in Java and Python.

IMPORTANT NOTE: The Freebase Read APIs (i.e., Search, Topic, and MQLRead) have a limit of 100,000 requests per day, which should be more than sufficient for this project. **However**, Freebase also has a limit on the number of requests per second per user, namely, a maximum of 10 requests/second. If you exceed this limit, Freebase will return an appropriate error message. You can change this limit through the Google console.

Freebase Usage

1. **Using the Freebase Search API**

   Your system should use the Freebase Search API to retrieve entities for an input query *q*. The Freebase Search API is a Web Service (i.e., endpoint accessed through a URL) that accepts the query *q* as a URL GET parameter. As we mentioned above, and to avoid problems with special characters that might be present in query *q*, you have to URL-encode the query before submitting it to Freebase. For instance, a query like *Bill Gates* should be converted to *Bill%20Gates*. Both Java and Python provide a handful of options for URL encoding (e.g., Java and Python), which you are encouraged to use.

   Given a URL-encoded query, say, *Bill%20Gates*, you can query the Freebase Search API using the URL *https://www.googleapis.com/freebase/v1/search?query=Bill%20Gates&key=API_KEY,* where API_KEY is your Freebase API key (see Freebase Basics section above). Freebase provides a handful of URL

GET parameter names/values to constraint the back-end Freebase search functionality. Your system should only use the ones mentioned above, namely, *query* and *key*.

In order to send your HTTP request, and similarly to what you did for Project 1, both Java an Python provide a handful of options (e.g., Java and Python). When you submit your HTTP request (and no error occurs), the Search API returns a JSON document. For our Bill Gates example above, here is a fragment of the Search API result (Bill Gates.search), as of February 22, 2015:

```
{
    "cost": 10,
    "cursor": 20,
    "hits": 2546,
    "result": [
        {
            "mid": "/m/017nt",
            "id":"/en/bill_gates",
            "name":"Bill Gates",
            "notable":{
                "name":"Organization leader",
                "id":"/business/board_member"
            },
            "lang":"en",
            "score":816.259216
        },
        {
            "mid": "/m/012mjr",
            "id":"/en/bill_melinda_gates_foundation",
            "name":"Bill & Melinda Gates Foundation",
            "notable":{
                "name":"Foundation",
                "id":"/m/0339xs"
            },
            "lang":"en",
            "score":390.698151
        },
        {
            "mid":"mid": "/m/09q11d",
            "id":"/en/bill_gates_house",
            "name":"Bill Gates's house",
            "notable":{
                "name":"Building",
                "id":"/architecture/building"
            },
            "lang":"en",
            "score":297.354309
        },


        ...more entries here...
    ],
    "status": "200 OK"
}
```

The Search API result contains the high-level properties *cost*, *cursor*, *hits*, *result*, and *status*. From these properties, we are only interested in *result*, and you should ignore the others. The *result* property is an array of objects where each object corresponds to one Freebase entity. As the figure above shows, the result entries are sorted in decreasing order of relevance, reported as the *score* property, for the original query. As discussed in step 3 in Part 1, you need to extract the *mid* values returned in the results. (You will use *mid* to query the Freebase Topic API, as we explain next.) You should ignore all the other properties at the *mid* level, namely, *id*, *lang*, *name*, *and notable*.

2. **Using the Freebase Topic API**

To use the Freebase Topic API in step 3 in Part 1, you should use the *mid* values returned from the Search API. For instance, based on the Search API result above, the first *mid* to consider is *"mid": "/m/017nt."* You can query the Topic API for this *mid* using the URL *https://www.googleapis.com/freebase/v1/topic/m/017nt?key=API_KEY.* As of February 22, 2015, this URL returns the JSON document that you can find here.

At the top level, the Topic API result has two properties, namely, *property* and *id.* We will focus on *property,* because it encapsulates all the information that we need regarding the queried *mid*. All properties, for instance *"/people/person/date_of_birth"*, share a common structure at the top level, with *count*, *values*, and *valuetype* (see examples below). We will **only** use the ***values*** property, which contains relevant values of the "parent" property (in this example, *"/people/person/date_of_birth"*). Specifically, the *values* property is an array of values of the parent property, where each entry might in turn consist of multiple properties. In this project, you should only worry about the *text*, *value*, *id*, and *property* properties in order to extract the values for the parent property. The only case where you should use the *id* property is in order to extract the *"/type/object/type,"* as we discuss below.

Here is an example to illustrate how you can use the *text* property to extract the name of an organization that businessperson Bill Gates founded:

```
"/organization/organization_founder/organizations_founded": {
        "count": 7.0,
        "values": [
                {
                        "creator": "/user/earlye",
                        "id": "/m/012mjr",
                        "lang": "en",
                        "text": "Bill & Melinda Gates Foundation", ⇐ Businessperson.founded[0]
                        "timestamp": "2008-07-13T18:46:07.000Z"
                },
                {
                        "creator": "/user/gardening_bot",
                        "id": "/m/04sv4",
                        "lang": "en",
                        "text": "Microsoft Corporation", ⇐ Businessperson.founded[1]
                        "timestamp": "2010-07-25T05:09:59.000Z"
                },
                {
                        "creator": "/user/gardening_bot",
                        "id": "/m/02fzrq",
                        "lang": "en",
                        "text": "Corbis", ⇐ Businessperson.founded[2]
                        "timestamp": "2010-07-25T05:21:02.000Z"
                },
                {
                        "creator": "/user/gardening_bot",
                        "id": "/m/04q34w5",
                        "lang": "en",
                        "text": "bgC3", ⇐ Businessperson.founded[3]
                        "timestamp": "2010-07-25T05:23:29.000Z"
                },
                {
                        "creator": "/user/fact_farm_wikimapper_intersector",
                        "id": "/m/03cs4bl",
                        "lang": "en",
                        "text": "Cascade Investment", ⇐ Businessperson.founded[4]
                        "timestamp": "2012-12-19T02:18:02.000Z"
                },
                {
                        "creator": "/user/fact_farm",
                        "id": "/m/06rkgg",
                        "lang": "en",
                        "text": "Microsoft Research", ⇐ Businessperson.founded[5]
                        "timestamp": "2012-12-21T17:59:10.000Z"
                },
                {
                        "creator": "/user/fpmdisalvia",
                        "id": "/m/085_73",
                        "lang": "en",
                        "text": "The Global Fund to Fight AIDS, Tuberculosis & Malaria", ⇐ Businessperson.founded[6]
                        "timestamp": "2013-02-27T22:09:52.000Z"
                }
        ],
        "valuetype": "object"
}
```

Then, here is an example to illustrate how you can use the *value* property to extract the date of birth of person Bill Gates (note that you could also use the *text* property here):

```
"/people/person/date_of_birth": {
        "count": 1.0,
        "values": [
                {
                        "creator": "/user/mwcl_musicbrainz",
                        "lang": "",
                        "text": "1955-10-28",
                        "timestamp": "2006-12-10T16:31:50.016Z",
                        "value": "1955-10-28", ⇐ Person.DateOfBirth
                }
        ],
        "valuetype": "datetime"
}
```

A more complex structure pattern involves the values of type *property*. In this example, the value is a compound object (i.e., not just a list of strings or numbers, but rather objects with named properties with lists of strings or numbers.) We use this example to illustrate how you can extract the board memberships of a businessperson:

```
"/business/board_member/organization_board_memberships": {
        "count": 4.0,
        "values": [
                <...1 entry here about Board Membership of Bill Gates @ Bill & Melinda Gates Foundation ...>
                {
                        "creator": "/user/gardening_bot",
                        "id": "/m/02h53n7",
                        "lang": "en",
                        "property": {
                                "/organization/organization_board_membership/from": {
                                        "count": 1.0,
                                        "values": [
                                                {
                                                        "creator": "/user/gardening_bot",
                                                        "lang": "",
                                                        "text": "1981", ⇐ Businessperson.BoardMember[1](From, To, Organization, Rol
                                                        "timestamp": "2010-07-30T00:38:48.000Z",
                                                        "value": "1981"
                                                }
                                        ],
                                        "valuetype": "datetime"
                                },
                                "/organization/organization_board_membership/organization": {
                                        "count": 1.0,
                                        "values": [
                                                {
                                                        "creator": "/user/gardening_bot",
                                                        "id": "/m/04sv4",
                                                        "lang": "en",
                                                        "text": "Microsoft Corporation", ⇐ Businessperson.BoardMember[1](From, To,
                                                        "timestamp": "2010-07-30T00:38:48.000Z"
                                                }
                                        ],
                                        "valuetype": "object"
                                },
                                "/organization/organization_board_membership/role": {
                                        "count": 1.0,
                                        "values": [
                                                {
                                                        "creator": "/user/gmackenz",
                                                        "id": "/m/09d6p2",
                                                        "lang": "en",
                                                        "text": "Chairman",  ⇐ Businessperson.BoardMember[1](From, To, Organization
                                                        "timestamp": "2014-02-05T02:56:38.000Z"
                                                }
                                        ],
                                        "valuetype": "object"
                                },
                                "/organization/organization_board_membership/title": {
                                        "count": 1.0,
                                        "values": [
                                                {
                                                        "creator": "/user/gmackenz",
                                                        "lang": "en",
                                                        "text": "Chairman",  ⇐ Businessperson.BoardMember[1](From, To, Organization
                                                        "timestamp": "2014-02-05T02:55:47.000Z",
                                                        "value": "Chairman"
                                                }
                                        ],
                                        "valuetype": "string"
                                },
                                "/organization/organization_board_membership/to": {
                                        "count": 1.0,
                                        "values": [
                                                {
                                                        "creator": "/user/gmackenz",
                                                        "lang": "",
                                                        "text": "2014-02-04",  ⇐ Businessperson.BoardMember[1](From, To, Organizati
                                                        "timestamp": "2014-02-05T02:54:04.000Z",
                                                        "value": "2014-02-04"
                                                }
                                        ],
                                        "valuetype": "datetime"
                                },
                        },
                        "text": "1981 - Microsoft Corporation - Chairman - Chairman - 2014-02-04 - robert - Organization governorshi
                        "timestamp": "2010-07-30T00:38:48.000Z"
                },
                <...more entries here about Bill Gates Booard Membership ...>
        ],
        "valuetype": "compound"
}
```

For Part 1 of your project, you should analyze the Topic API results and determine the mapping between the properties in the Topic API results and the properties of interest for the entity types Person, Businessperson, Author, Actor, League, and SportsTeam.

3. **Extracting the Types of an Object**

   The only case where you should not use the *text*, *value*, and *property* properties to extract the values of a parent property is when you have to extract the types of an object using the "*/type/object/type*" property. In this case, you are required to use the *id* value only. Note that a Topic API result might contain many "*/type/object/type*" properties. Your system should extract only the **top-level "*/type/object/type*" property**, which is unique (among the top level properties) and contains the type or types of the associated entity, as the *id* value or values. The following example includes the first 3 types associated with entity Bill Gates:

```
"text":"/type/object/type": {
        "count": 31.0,
        "values": [
                {
                        "creator": "/user/earlye",
                        "id": "/book/author",  ⇐ type[0]
                        "lang": "en",
                        "text": "Author",
                        "timestamp": "2008-07-13T18:55:37.000Z"
                },
                {
                        "creator": "/user/earlye",
                        "id": "/organization/organization_founder",⇐ type[1]
                        "lang": "en",
                        "text": "Organization founder",
                        "timestamp": "2008-07-13T18:45:46.000Z"
                },
                {
                        "creator": "/user/mwcl_musicbrainz",
                        "id": "/people/person",  ⇐ type[2]
                        "lang": "en",
                        "text": "Person",
                        "timestamp": "2006-12-10T16:31:50.016Z"
                },
                <...more entries here...>
        ]
}
```

   Given our mapping from Freebase types to our entities of interest for this project, we conclude that Bill Gates is an *Author*, a *BusinessPerson*, and a *Person*. Accordingly, we can proceed and extract the specified properties associated with these entity types (see the *Using the Freebase Topic API* section above).

4. **Using the Freebase MQLRead API**

   The Freebase MQLRead API lets us send MQL queries to Freebase. The MQLRead API is a web service and its URL GET parameter is an MQL query that has to be URL-escaped, as usual. You can query the API using URL *https://www.googleapis.com/freebase/v1/mqlread?query=url_encoded(query)&key=API_KEY*. Again, Freebase provides a handful of URL GET parameter names/values to constraint the back-end Freebase MQLRead functionality. Your system should only use the ones mentioned above, namely, *query* and *key*. In the "*Who created [X]?*" question, your system should identify the people (i.e., Authors or BusinessPersons) whose written books or founded organizations contain the string *X* in their name. To illustrate how to do this, the following example performs this task for question "Who created Mona Lisa?" and focuses on artists who have at least one artwork that contains the string "Mona Lisa." The MQL Query is as follows:

```
[{
        "/visual_art/visual_artist/artworks": [{
                "a:name": null,
                "name~=": "Mona Lisa"
        }],
        "id": null,
        "name": null,
        "type": "/visual_art/visual_artist"
}]
```

   In this MQL query, we ask for the *id* and *name* properties of the entity with "*type*" equal to "*/visual_art/visual_artist*." Moreover, we limit our search to only those artists who have at least one artwork with a name that contains the string "Mona Lisa" ("*name~=*"*: "Mona Lisa*"). Finally, the query will also return the artworks (array) with the names that match the "*contains*" constraint ("*a:name*"*: null*). The "*a:*" identifier before the name is only for disambiguation from the other "name" we have used ("*name~=*") and it has no further semantic meaning. You can run the query and see the results of this MQL query here.

Test Cases

Your submission (see below), which should follow the reference implementation, should include a transcript of the runs of your program for the following queries and questions:

Part 1: Infobox Creation

1. Bill Gates
2. Robert Downey Jr.
3. Jackson
4. NFL
5. NBA
6. NY Knicks
7. Miami Heat

Part 2: Question Answering

1. Who created Google?
2. Who created Lord of the Rings?
3. Who created Microsoft?
4. Who created Romeo and Juliet?

We will check the execution of your program on these cases, as well as on some other queries and questions. Also, note that we will check your program in terms of redundancy of results (you should avoid producing the same infobox or the same answer multiple times). We will also check that your system does not provide infoboxes or answers whenever Freebase does not return any appropriate information. In other words, your system should realize when no appropriate answer should be produced, following how our reference implementation behaves.

Reference Implementation

We created a reference implementation for this project. To run the reference implementation from your CS account, you have three options:

1. **/home/gravano/6111/Html/Proj2/code/run.sh -key <Freebase API key> -q <query> -t <infobox|question>**
2. **/home/gravano/6111/Html/Proj2/code/run.sh -key <Freebase API key> -f <file of queries> -t <infobox|question>**
3. **/home/gravano/6111/Html/Proj2/code/run.sh -key <Freebase API key>**

where:

- **<Freebase API key >** is your Google account key for a project that has enabled Freebase (see above)
- **-q <query>** is your query (either an entity for infobox creation or a question)
- **-f <file of queries>** is a text file with one query per line. Here is the file of queries for the infobox creation test cases above and here is another text file with the questions for the question answering test cases above.
- **-t <infobox|question>** indicates whether you are invoking Part 1 of the project, by specifying -t infobox, or Part 2 of the project, by specifying -t question.

The -q option sends just one query to the reference implementation, while the -f option sends a batch of queries. The third option above, without -q or -f, lets you submit queries to the reference implementation interactively. **Please follow this format closely for your submission.**

You should use the reference implementation **to give you an idea of how your system should behave** in terms of the output that it produces. We will not grade you on the response time or efficiency of your project submission, but **your output should match that of the reference implementation as much as possible**.

What You Should Submit

1. Your well-commented **Java or Python code**, which should follow the **format of our reference implementation** (see above).
2. A **README** file including the following information:

   a) Your name and your partner's name and Columbia UNI

   b) A list of all the files that you are submitting

   c) A clear **description of how to run your program** (note that your project must compile/run under Linux in your CS account)

   d) A clear description of the internal design of your project, including listing the mapping that you used to map from Freebase properties to the entity properties of interest that you return

   f) **Your Freebase API Key** (so we can test your project) as well as the **requests per second per user** that you have set when you configured your Google project (see Freebase Basics section)

   g) Any additional information that you consider significant
3. A **transcript** of the runs of your program on the test cases above.

How to Submit

1. Create a directory named **<your-UNI>-proj2**, where you should replace **<your-UNI>** with the Columbia UNI of **one** teammate (for example, if the teammate's UNI is **abc123**, then the directory should be named **abc123-proj2**).
2. Copy the source code files into the **<your-UNI>-proj2** directory, and include all the other files that are necessary for your program to run.
3. Copy your **README** file as well as your query **transcript** (see above) into the **<your-UNI>-proj2** directory.
4. **Tar** and **gzip** the **<your-UNI>-proj2** directory, to generate **a single file <your-UNI>-proj2.tar.gz**, which is the file that you will submit.
5. Login to Courseworks at https://courseworks.columbia.edu/ and select the site for our class.
6. Select "Assignments."
7. Upload your **<your-UNI>-proj2.tar.gz** file under **"Project 2."**

Fotis Psallidas and Luis Gravano