ENGR 212 – PROGRAMMING PRACTICE
SPRING 2015
MINI PROJECT 3

April 17, 2015

In this project, you will develop a search engine for a digital library of academic papers. Your search engine will allow performing keyword searches, and search results will be ranked in a particular way.

Dataset:

As part of the project, you are provided a paper dataset from physics domain. This dataset contains two types of data as summarized next.

- **Citation data**: Almost all papers cite other papers which are listed in bibliography/ references section of each paper. The file with name, citations.txt, contains paper citation information. Each line of this file contains two paper ids separated by a tab character. The first id on each line is the id of the *citing* paper, and the second one is the id the *cited* paper.

- **Paper Metadata (Title, Abstract, Authors, etc.):** Metadata for each paper is stored in a separate file (one per paper) which is named as *paperid.abs*, and located under *metadata* directory. An example metadata file content is shown below:

```
\\
Paper: hep-th/9201020
From: STELLE@PHYS.TAMU.EDU
Date: Sat, 11 Jan 1992 17:06:09 CST   (30kb)

Title: The Superparticle and the Lorentz Group
Authors: A.S. Galperin and K.S. Stelle
Comments: 33 pages
Journal-ref: Nucl.Phys. B368 (1992) 248-280
\\
  We present a unified group-theoretical framework for superparticle theories.
This explains the origin of the ``twistor-like'' variables that have been used
in trading the superparticle's $\kappa$-symmetry for worldline supersymmetry.
We show that these twistor-like variables naturally parametrise the coset space
${\cal G}/{\cal H}$, where $\cal G$ is the Lorentz group $SO^\uparrow(1,d-1)$
and $\cal H$ is its maximal subgroup. This space is a compact manifold, the
sphere $S^{d-2}$. Our group-theoretical construction gives the proper
covariantisation of a fixed light-cone frame and clarifies the relation between
target-space and worldline supersymmetries.
```

Project Description:

- *Initialization & Indexing:*
  - ✓ Your search engine first needs to be initialized by parsing and loading the paper data from the above described files. You may assume that citation file is located in the same directory as your program, and named as citations.txt, and paper metadata files are stored under the metadata directory (which is also located in your program's directory) as described above (one file per paper).
  - ✓ In order to minimize the searching time, you should build an index. However, this index will be a little different than the one that we saw in the class which was employing a database. Your index will not use a database. Instead, it will be built on a set of Python dictionaries that you are going to design and populate. Here are some dictionary suggestions for your index.
    - wordlocations → key: a word w, value: a dictionary where key is a paper id, and value is the list of locations that w appear in that paper. That is, your dictionary will be organized as follows:

wordlocations[*a_word*] = {paperId1: [loc1, loc2, …, locN], paperId2: [loc1, loc2, …, locM], …, paperIdZ: [loc1, loc2, …, locK]}

Locations are indexes of the words. You may assume that the title and the abstract of a paper are combined into one piece of text. Hence, you do not need to differentiate between title and abstract. The first word in the title will have location 0, and the last word in the abstract will have location (len(title) + len(abstract) - 1).

- citations → key: toId (id of the cited paper), value: list of paper ids which cite the paper in the key. That is, your dictionary will be organized as follows:

  citations[9201020] = [9451040, 9351670, …, 9472345]

- citationcounts → key: fromId (id of the citing paper), value: the number of papers which are cited by the paper in the key.

✓ To be used during ranking, you need to pre-compute PageRank scores of papers based on the citation data. That is, if a paper A cites paper B, you may assume that there is a link from paper A to paper B. Using the PageRank approach that we discussed in the class, implement a new method in your code file that will compute PageRank score of each paper, and store it in a dictionary where key will be paper id, and value will be PageRank score of that paper. You do not have to, but if you like, you may use the PageRank implementation in searchengine.py module (method name: *calculatepagerank*), as a template. However, you need to replace all database-related code in that method with your code that utilizes the above dictionaries (that is, citations and citationcounts). You need call the normalization function from searchengine.py to normalize the PageRank scores at the end (higher is better).

✓ Your program will have a GUI similar to the one shown in Figure 1. The above described initialization steps are started when the user clicks on the button under the search box. This button has the label “Initialize” when your program starts. During the initialization phase, your program should print some status messages showing what it has done so far, and what it is doing right now. Status messages can be printed in the area where search results will be shown (you may use the Text widget) (Please see Figure 1). Please print the same messages on the console/terminal screen as well.

✓ After the initialization process completes, the label of the initialization button should be automatically changed to “Search” (Figure 2). From this point on, this button will be used to initiate a search with the provided keywords in the search box above the button.

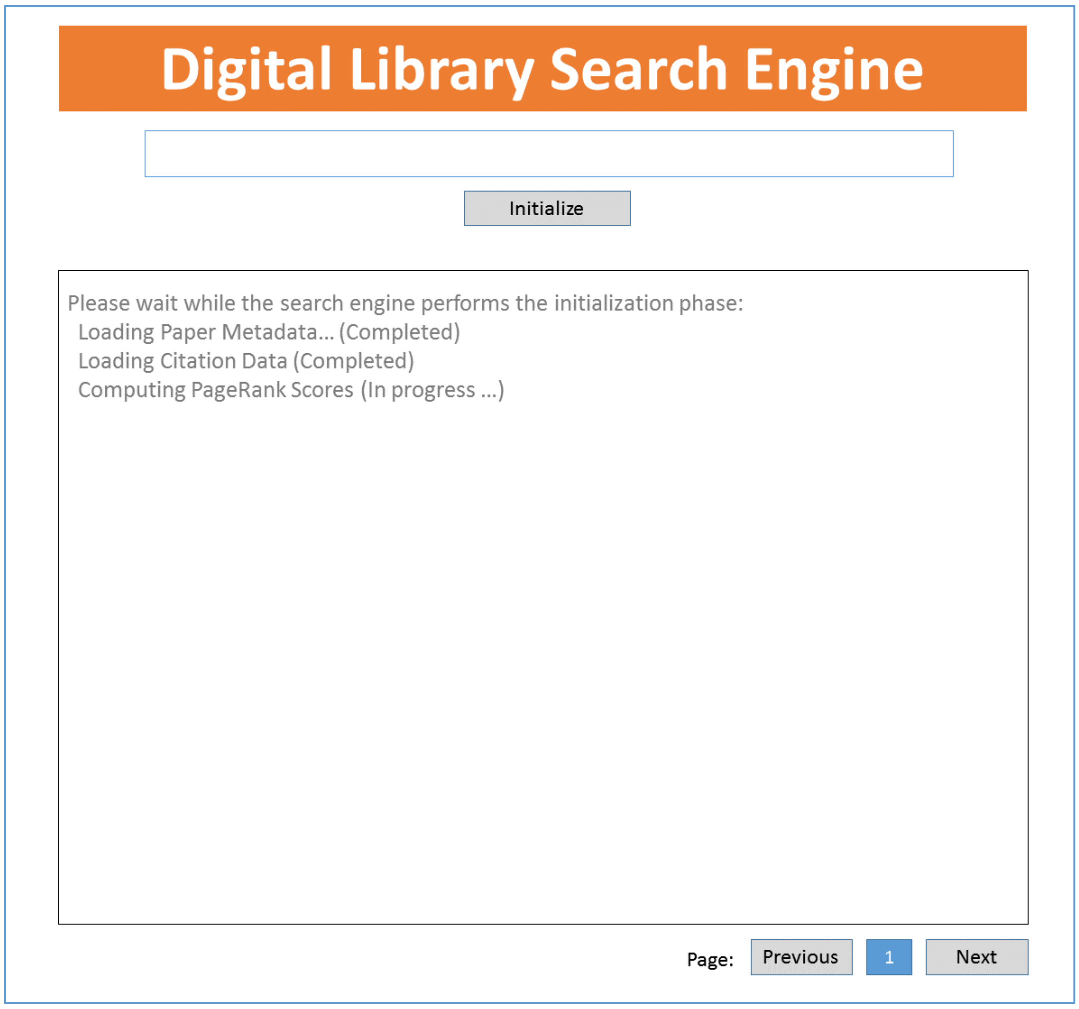
- *Searching:*

✓ Users should be able to search with a single or multiple keywords. In order for a paper to be included in the search results, all of the search keywords should appear in the title and/or abstract of the paper. In the implementation, this requires computing the intersection of dictionaries that store paperIds for each word in your index.

✓ In the search results, each row should contain rank (starting from 1), paper title, and paper score (computed according to the ranking measures described in the next section).

✓ Your program should display the total number of papers in a search result set, and the time it took to complete the search at upper left corner of the search results area (Please see Figure 2).

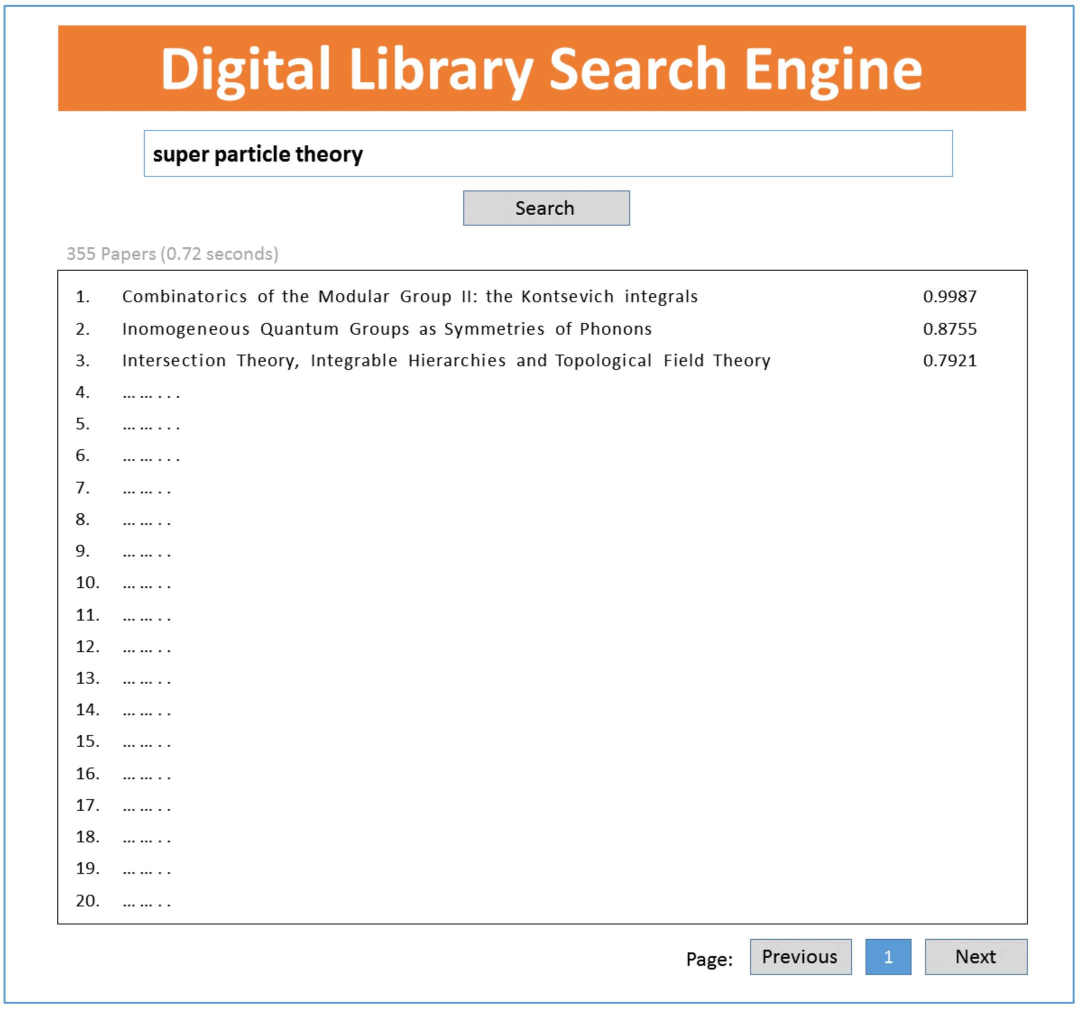
✓ Search results should be shown in a paginated manner similar to Google. That is, you will show 20 search results at a time, and users should be able to navigate to the previous and next page by using buttons located at the bottom of the search result area. Between '*previous*' and '*next*' buttons, you should have a label that shows the current page (starting from 1) (Please see Figure 2).



Please wait while the search engine performs the initialization phase:
Loading Paper Metadata... (Completed)
Loading Citation Data (Completed)
Computing PageRank Scores (In progress ...)

Figure 1

- *Ranking:*
  - ✓ You are going to rank search results by combining a content- and a link-based scoring measure.
    - Content-based measure: As content-based measure, you are going to write a method that will compute a kind of frequency-based score. More specifically, let's say that a user has searched with three keywords, "word1 word2 word3". Assume that a paper p contains 3 occurrences of word1, 2 occurrences of word2, and 5 occurrences of word3. Then, the content-based score of p is 3 x 2 x 5 = 30. You need to call the normalization function from searchengine.py to normalize the scores at the end (higher is better).
    - Link-based measure: You will use the PageRank scores that you computed and stored during the initialization phase at the starting part of the program.
    - In order to combine the above two scores, just sum them up, and use the resulting score to rank the result web pages.



355 Papers (0.72 seconds)

```
1.   Combinatorics of the Modular Group II: the Kontsevich integrals          0.9987
2.   Inomogeneous Quantum Groups as Symmetries of Phonons                     0.8755
3.   Intersection Theory, Integrable Hierarchies and Topological Field Theory 0.7921
4.   … … . . .
5.   … … . . .
6.   … … . . .
7.   … … . .
8.   … … . .
9.   … … . .
10.  … … . .
11.  … … . .
12.  … … . .
13.  … … . .
14.  … … . .
15.  … … . .
16.  … … . .
17.  … … . .
18.  … … . .
19.  … … . .
20.  … … . .
```

Figure 2

Can you provide any further pointers that may be helpful? :

- As for the GUI, you **should use** Tkinter that we have covered last semester (see ENGR 211 last week's slides). If you do not like Tkinter, you may use PyQt (we have not covered that in ENGR 211), but in any case, you are **<u>not</u>** allowed to use a designer or any other GUI module (other than the above ones).
- In order to be able to place widgets as shown in Figures, you should heavily use frames. Please see ENGR 211, last week's slides for creating row, col, and grid frames, and their example uses with Swampy.
- For the output area at the bottom, again, you may use the Text widget.
- You may use *separatewords* method in searchengine.py module to divide paper title and abstract into the individual words.
- You may use os.listdir() method to get the list of paper files under metadata directory during the initialization phase. The following tutorial has an example:

  http://www.tutorialspoint.com/python/os_listdir.htm
- To relabel a button object, you may use the following call:

  my_button.configure(text='Search')

How and when do I submit my project? :

- Projects may be done individually or as a small group of two students. If you are doing it as a group, only **one** of the members should submit the project. File name will tell us group members (Please see the next item for details).

- Submit your own code in a **single** Python file (Do **not** include clusters.py or feeds.py that you import). Name it with your and your partner's first and last names (see below for naming).
    - If your team members are Deniz Barış and Ahmet Çalışkan, then name your code file as deniz_baris_ahmet_caliskan.py (Do **not** use any Turkish characters in file name).
    - If you are doing the project alone, then name it with your name and last name similar to the above naming scheme.
- Do **not** copy/paste code from searchengine.py into your own code file. Anything that you need from searchengine.py should be called with proper dot notation after importing that module.
- Do **not** use any external module other than Swampy, which are not included in standard Python installation.
- Do **not** use Python 3.x. Use Python 2.7.x.
- Submit it online on LMS (Go to the Assignments Tab) by **17:00 on May 1 (Friday)**.

    Late Submission Policy:

    - -20%: Submissions between 17:01 – midnight (00:00) on the due date.
    - -40%: Submissions which are 24 hour late.
    - -50%: Submissions which are 48 hours late.
    - Submission more than 48 hours late will not be accepted.

Grading Criteria? :

- Does it run? (Submissions that do not run will get some partial credit which will not exceed 30% of the overall project grade).
- Does it implement all the features according to the specifications and produce correct results?
- Code organization (Meaningful names, sufficient and appropriate comments, proper organization into functions and classes, clean and understandable, etc.)?
- Interview evaluation.

Have further questions? :

Please contact your TAs (Mehmet Aytimur and Muhammed Esad Unal) if you have further questions. If you need help with anything, **please use the office hours** of your TAs and the instructor to get help. If office hours are not suitable, please **get** an appointment through email before walking in your TAs offices.