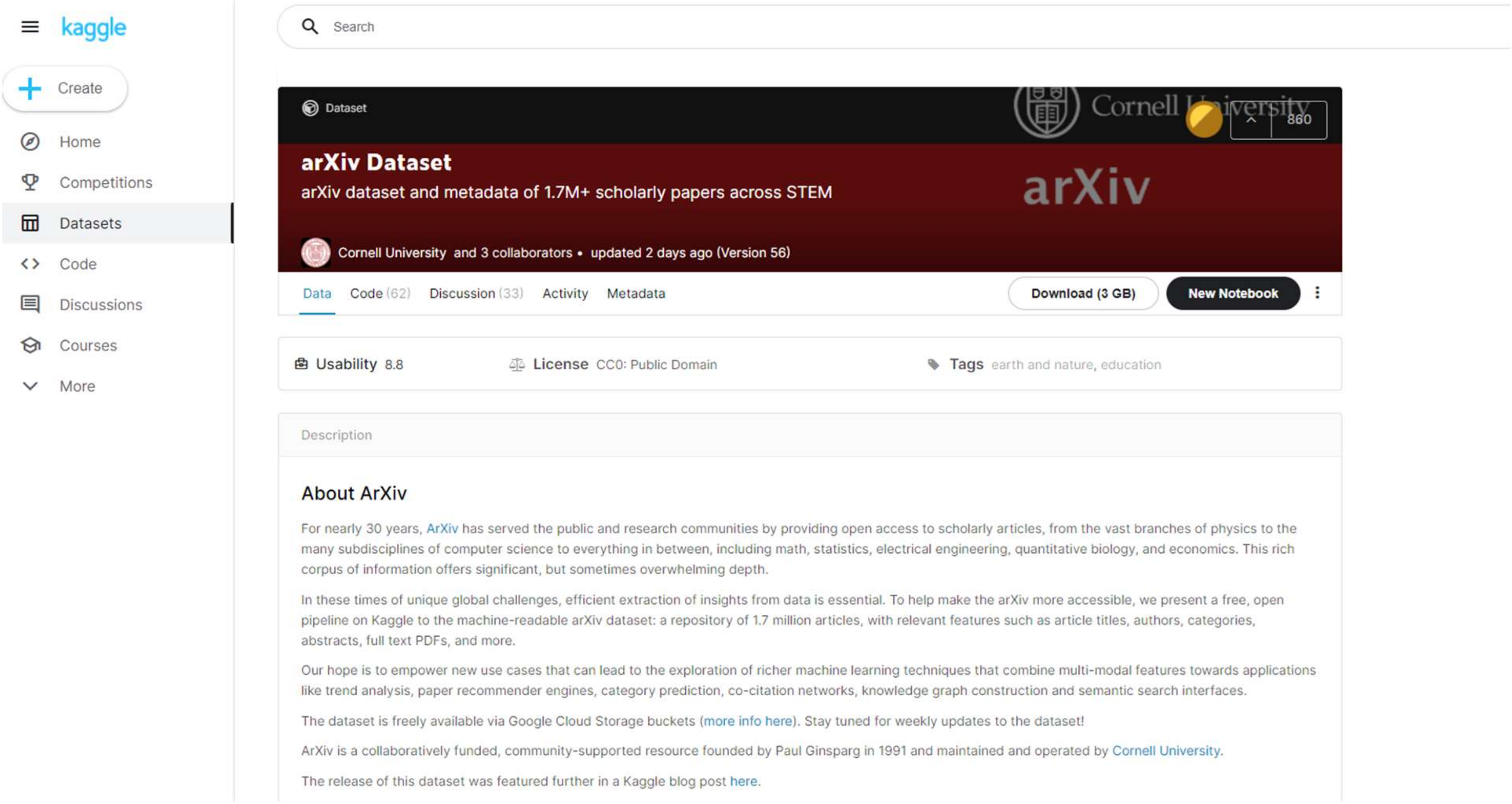
Final Project – Essay Search

Deadline : 2022/1/19 23:59

Intro

- There are many search engine nowadays
- Eg: Google, Yahoo, Baidu… etc.
- In this final project, we need to build a simple essay search engine

Dataset



arXiv Dataset

arXiv dataset and metadata of 1.7M+ scholarly papers across STEM

Cornell University and 3 collaborators • updated 2 days ago (Version 56)

Usability 8.8 License CC0: Public Domain Tags earth and nature, education

Description

About ArXiv

For nearly 30 years, ArXiv has served the public and research communities by providing open access to scholarly articles, from the vast branches of physics to the many subdisciplines of computer science to everything in between, including math, statistics, electrical engineering, quantitative biology, and economics. This rich corpus of information offers significant, but sometimes overwhelming depth.

In these times of unique global challenges, efficient extraction of insights from data is essential. To help make the arXiv more accessible, we present a free, open pipeline on Kaggle to the machine-readable arXiv dataset: a repository of 1.7 million articles, with relevant features such as article titles, authors, categories, abstracts, full text PDFs, and more.

Our hope is to empower new use cases that can lead to the exploration of richer machine learning techniques that combine multi-modal features towards applications like trend analysis, paper recommender engines, category prediction, co-citation networks, knowledge graph construction and semantic search interfaces.

The dataset is freely available via Google Cloud Storage buckets (more info here). Stay tuned for weekly updates to the dataset!

ArXiv is a collaboratively funded, community-supported resource founded by Paul Ginsparg in 1991 and maintained and operated by Cornell University.

The release of this dataset was featured further in a Kaggle blog post here.

Essay Search

- Input
    - A set of txt files (essays) in the given folder path (0.txt, 1.txt, ...)
    - A given txt file containing search queries
    - Output file name
- Output
    - Output a txt file with the given name
- Given a word, our objective is to list the essays that their titles or abstracts contain the word
- We need to consider only the alphabetic words. You can ignore special symbols or digits
- The queries are <u>case insensitive</u>, i.e., we are treating uppercase and lowercase characters the same

Query

- Exact Search: “search-word”
  - Eg: we want to search essay with ***graph***, we use query - “graph”
- Prefix Search: search-word
  - Eg: we want to search essay with prefix ***graph***, we use query - graph
- Suffix Search: *search-word*
  - Eg: we want to search essay with suffix ***graph***, we use query - *graph*
- *And* operator: “+”
  - Eg: we want to search essay with ***graph*** and ***sparsity***,
    we use query – “graph” + “sparsity”
- *Or* operator: “/”
  - Eg: we want to search essay with ***graph*** or ***quantum***,
    we use query – “graph” / “quantum”

Requirements

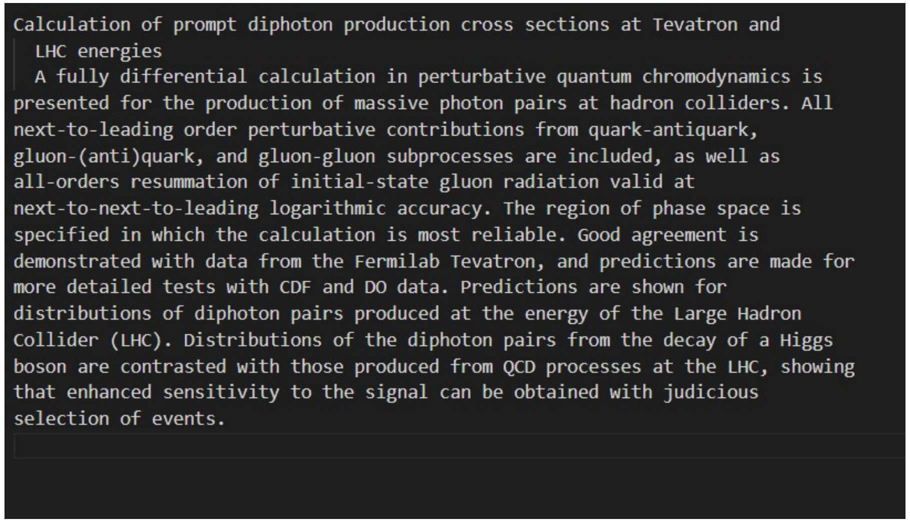
- Implement with C/C++
- Design your own data structure to make search faster
- Strictly follow the input/output formats
- Do not use any string matching library (eg: str.find, …)
- Do not copy/paste others’ codes

Input file – essay file

- ➢ There are a set of essay txt files, named 0.txt, 1.txt, ...
- ➢ Those essay txt files will be put in the given directory

- ➢ Every essay txt file contains two parts

1. Title (the first line)
2. Abstract (the remaining sentences)



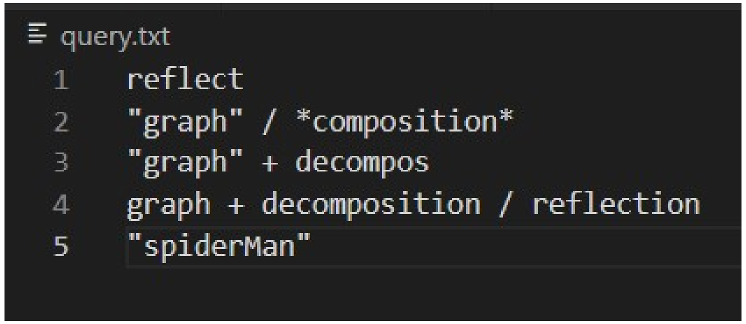
```
Calculation of prompt diphoton production cross sections at Tevatron and
  LHC energies
  A fully differential calculation in perturbative quantum chromodynamics is
presented for the production of massive photon pairs at hadron colliders. All
next-to-leading order perturbative contributions from quark-antiquark,
gluon-(anti)quark, and gluon-gluon subprocesses are included, as well as
all-orders resummation of initial-state gluon radiation valid at
next-to-next-to-leading logarithmic accuracy. The region of phase space is
specified in which the calculation is most reliable. Good agreement is
demonstrated with data from the Fermilab Tevatron, and predictions are made for
more detailed tests with CDF and DO data. Predictions are shown for
distributions of diphoton pairs produced at the energy of the Large Hadron
Collider (LHC). Distributions of the diphoton pairs from the decay of a Higgs
boson are contrasted with those produced from QCD processes at the LHC, showing
that enhanced sensitivity to the signal can be obtained with judicious
selection of events.
```

Input file – query file

- There would be several queries in a query file
- One line represents one query that has to be processed
- The And / Or operator is left associative
- Eg: graph + decomposition / quantum

  = (graph + decomposition) / quantum
- All the queries are valid, i.e., you don’t need to worry about invalid queries

Query example

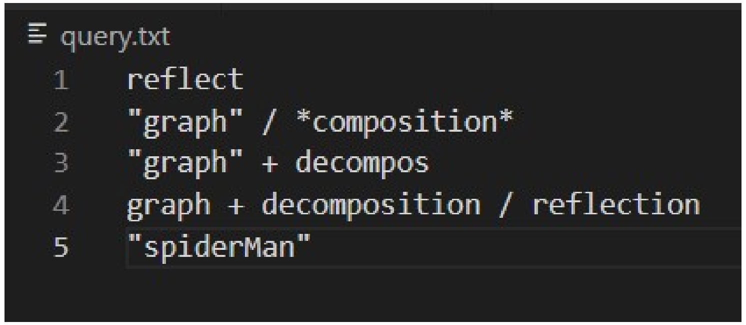


query.txt

```
reflect
"graph" / *composition*
"graph" + decompos
graph + decomposition / reflection
"spiderMan"
```

- First query: **reflect**
  - Find essays that have word with prefix [reflect], eg: reflect, reflection.
- Second query: **“graph” / *composition***
  - Essay set A: Find essays that have exactly the word [graph]
  - Essay set B: Find essays that have words with suffix [composition]
  - A, B set with OR operator -> answer = union of sets A and B
- Third query: **“graph” + decompos**
  - Essay set A : Find essays that have exactly the word [graph]
  - Essay set B : Find essays that have words with prefix [decompos]
  - A, B set with AND operator -> answer = intersection of sets A and B

Query example



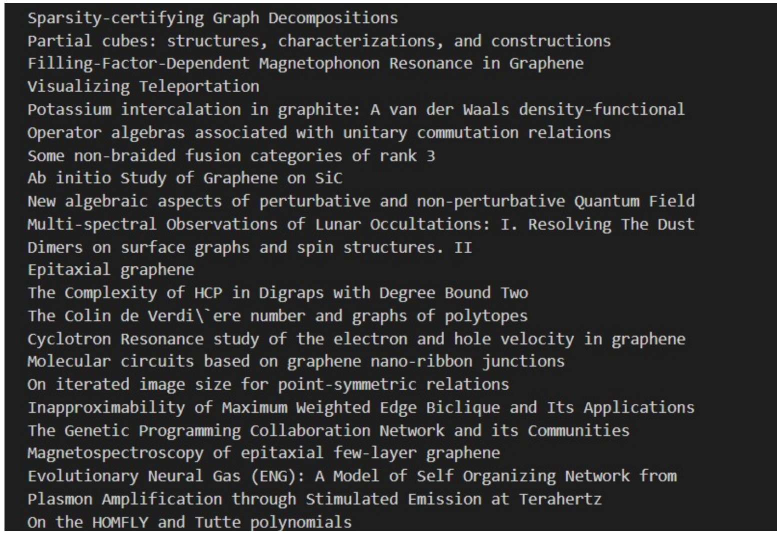
```
reflect
"graph" / *composition*
"graph" + decompos
graph + decomposition / reflection
"spiderMan"
```

- ➢ Fourth query: **graph + decomposition / reflection**
  - ➢ Essay set A: Find essays that have words with prefix [graph]
  - ➢ Essay set B: Find essays that have words with prefix [decomposition]
  - ➢ Essay set C: Find essays that have words with prefix [reflection]
  - ➢ We know that A + B / C = (A + B) / C
  - ➢ Essay set D = intersection of sets A and B
  - ➢ Answer = union of sets D and C
- ➢ Fifth query: **“spiderMan”**
  - ➢ Find essays that have exactly the word [spiderman]
  - ➢ Keep in mind that upper- and lower-case characters are treated the same

Output file format

- Output file name is given as arguments when executing your program
- Output the essay titles of the search result in output file
- Every essay title should be followed with a new line character
- If not found -> print out “Not Found!” (不用印雙引號)
- Output order follows the essay order

  (0.txt, 1.txt, ...)



```
Sparsity-certifying Graph Decompositions
Partial cubes: structures, characterizations, and constructions
Filling-Factor-Dependent Magnetophonon Resonance in Graphene
Visualizing Teleportation
Potassium intercalation in graphite: A van der Waals density-functional
Operator algebras associated with unitary commutation relations
Some non-braided fusion categories of rank 3
Ab initio Study of Graphene on SiC
New algebraic aspects of perturbative and non-perturbative Quantum Field
Multi-spectral Observations of Lunar Occultations: I. Resolving The Dust
Dimers on surface graphs and spin structures. II
Epitaxial graphene
The Complexity of HCP in Digraps with Degree Bound Two
The Colin de Verdi\`ere number and graphs of polytopes
Cyclotron Resonance study of the electron and hole velocity in graphene
Molecular circuits based on graphene nano-ribbon junctions
On iterated image size for point-symmetric relations
Inapproximability of Maximum Weighted Edge Biclique and Its Applications
The Genetic Programming Collaboration Network and its Communities
Magnetospectroscopy of epitaxial few-layer graphene
Evolutionary Neural Gas (ENG): A Model of Self Organizing Network from
Plasmon Amplification through Stimulated Emission at Terahertz
On the HOMFLY and Tutte polynomials
```

Test environment

- CPU: i9-9900k
- RAM: 32GB DDR4
- DISK: 1TB
- GCC version: 7.5.0
  - If you need another version of the compiler, please let us know the reason

Testing

- Your code should take three arguments:
  - input folder path
  - query file path
  - output file name
- Output file name should be:
  - Output file with [output_file_name]
- TA will compile your file as follows
  - **g++ -std=c++17 -o essay-search.exe ./*.cpp -lstdc++fs**
  - If your code need to use other library so that this command cannot compile your code, please specify the compile command you used and state the reason clearly **in the report**
- TA will test your code as follows
  - ./essay_search.exe [input_folder_path] [query_file_path] [output_file_name]
- Time limits
  - Your program would be killed after 4 seconds
  - Brute-force algorithms won’t get through

Scoring

We have a small dataset (1000 files) and a bigger dataset (8000up files)

Exact Search + And / Or Operator (50%)

100% query output correct -> get 50 points

80%~99% query output correct -> get 25 points

less than 80% query output correct -> get 0 points

Suffix / Prefix Search (15%)

100% query output correct -> get 15 points

80%~99% query output correct -> get 7 points

less than 80% query output correct -> get 0 points

Scalability Test: test with more essays and queries (10%)

You get these points when the answers are all correct

We will test your code only if you pass last two test

Speed Test: compete the speed with your classmate (15%)

We will test your code only if you get all the points in above three tests (75 points)
Last 10%: 0 points
Top 50% - Top 90%: 5 points
Top 20% - Top 50%: 10 points
Top 20%: 15 points

Report (10%)

Report

- Your report should contain
  - How you implement your code
  - Challenges you encounter in this project
  - References that give you the idea (github/paper…)

- No more than 2 pages

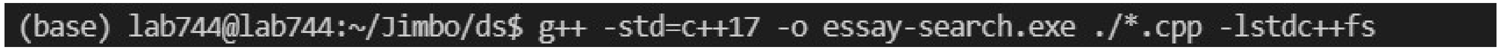
Submission

- Submit your
  - Code
  - Report

- Submit a zip file with filename “[student_id]_project”

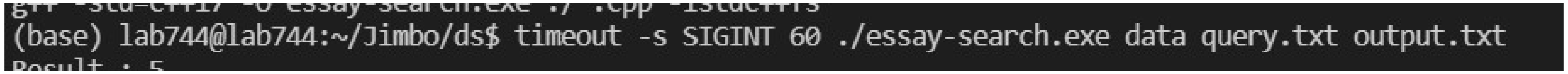
Submission

- After compile command
  g++ -std=c++17 -o essay-search.exe ./*.cpp -lstdc++fs
  a executable file “essay-search.exe” should be created



```
(base) lab744@lab744:~/Jimbo/ds$ g++ -std=c++17 -o essay-search.exe ./*.cpp -lstdc++fs
```

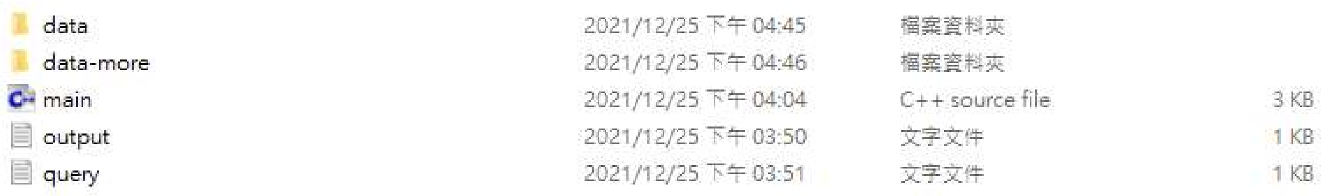
- After execute, [output-file-name] should be created



```
g++ -std=c++17 -o essay-search.exe ./*.cpp -lstdc++fs
(base) lab744@lab744:~/Jimbo/ds$ timeout -s SIGINT 60 ./essay-search.exe data query.txt output.txt
Result : 5
```

File structure

- main.cpp: essay txt parser and some hint
- query.txt: sample input
- output.txt: sample output
- data: sample essay data folder
- data-more: more essay data provide for self testing (1000 files)



data	2021/12/25 下午 04:45	檔案資料夾	
data-more	2021/12/25 下午 04:46	檔案資料夾	
main	2021/12/25 下午 04:04	C++ source file	3 KB
output	2021/12/25 下午 03:50	文字文件	1 KB
query	2021/12/25 下午 03:51	文字文件	1 KB

Given main.cpp & parser

- We have provide some code in main.cpp
- You can use these code for your implementation

  助教提供Parser，如要自行implement，請自行確定與助教提供之parser輸出相同

- Functionality that has been provided

1. Store variable for argv argument
2. Process essay title and content, storing into two vector<string>
3. Utility function for parsing and string split

Note

- You are allowed to use STL
- But don’t use any string matching library function

Implementation

- How can we build data structure that efficiently support searching?
- These are some common structure that we can reference to
- Trie (TA implemented this)
  - reference: https://www.geeksforgeeks.org/trie-insert-and-search
  - reference: https://www.hackerearth.com/practice/data-structures/advanced-data-structures/trie-keyword-tree/tutorial/
- Suffix-Tree
  - reference: https://blog.csdn.net/fjsd155/article/details/80211145
  - reference: https://www.geeksforgeeks.org/ukkonens-suffix-tree-construction-part-1/
- Ternary Search Tree
  - reference: https://www.geeksforgeeks.org/ternary-search-tree/
  - reference: https://www.cs.upc.edu/~ps/downloads/tst/tst.html
- Compressed Trie
  - reference: https://www.geeksforgeeks.org/compressed-tries/