Joogle - CSI4107 Search Engine Project

By: Jacob Danovitch

Foreword

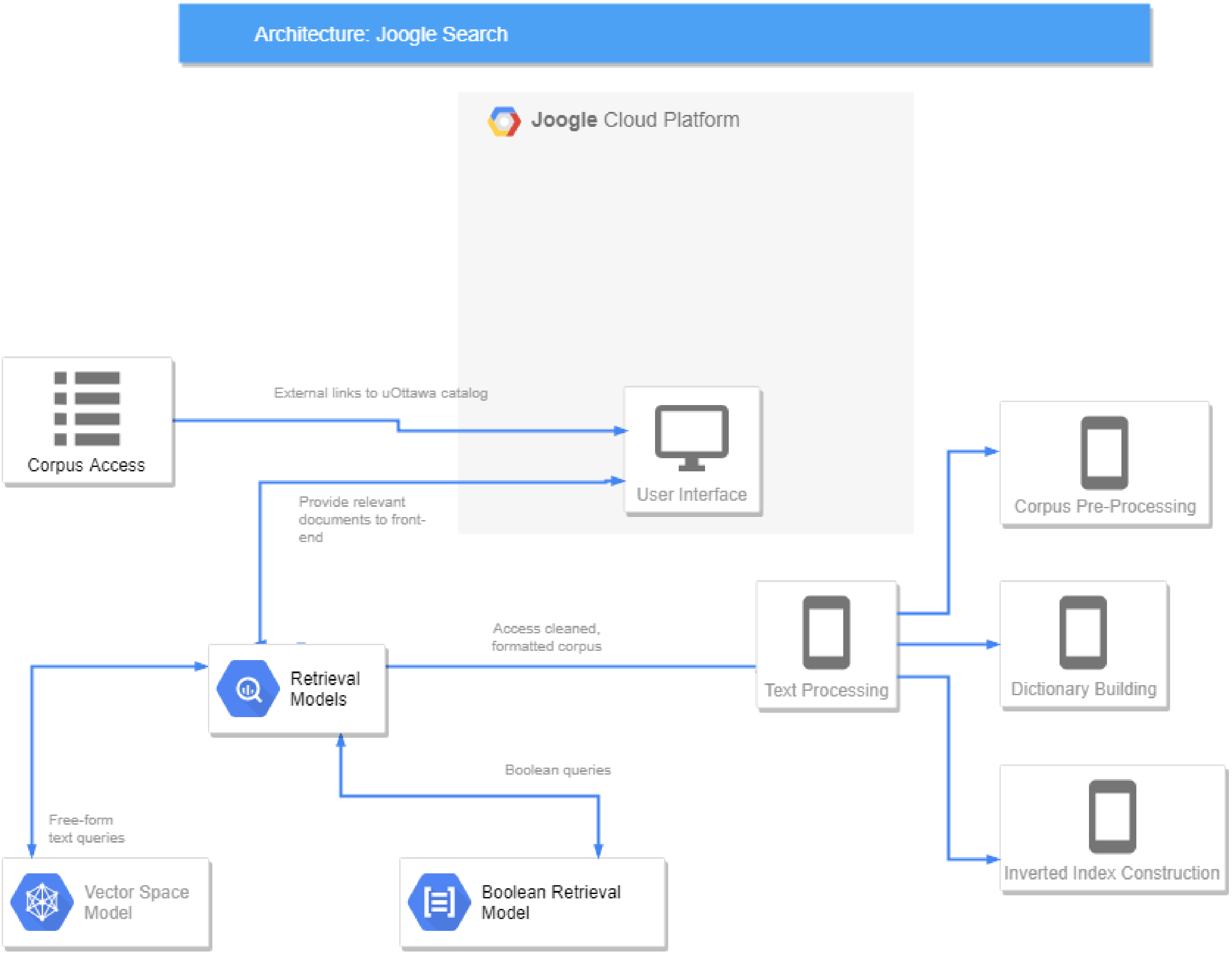
The following is my report for the search engine project. Please note that rather than running my code, it is much easier to go to its website online at https://joogle-csi4107.herokuapp.com (https://joogle-csi4107.herokuapp.com). Instructions are in `README.md`.

This report contains a system architecture diagram, followed by explanations and demonstrations of the modules implemented in my program, and concludes with the demonstration screenshots. The bulk of the length of this report is due to the explanations/demonstrations of the modules, so if these are unnecessary you may skip directly to Demo for the example queries suggested by Professor Barriere.

I recommend viewing the HTML version of this report, or viewing it on NBViewer (https://nbviewer.jupyter.org/github/jacobdanovitch/Joogle/blob/master/Report_JacobDanovitch_CSI4107A1.ipynb) However, a PDF version is also included.

System Architecture

The following diagram was created using [draw.io (https://draw.io)](https://draw.io) and their Google Cloud Platform templates.



Modules - Mandatory

Corpus Pre-Processing

File(s)

- `uottawa_scrape.py`

Description

This module scrapes and parses the uOttawa course catalogue using `requests` and `beautifulsoup4`, saving the data to [`data/catalogue-uottawa-ca.json` (data/catalogue-uottawa-ca.json)](data/catalogue-uottawa-ca.json).

Demonstration

First, the HTML is acquired using the `requests` library, and a `BeautifulSoup` object is created.

In [1]:
```python
from uottawa_scrape import *

html = scrape(to_file=False)
print(html.find("div", {"class": "courseblock"}).text)
```

```
CSI 1306 Computing Concepts for Business (3 units)

Introduction to computer-based problem solving from the perspective of the bu
siness world. Design of algorithms for solving business problems. Basics of c
omputer programming in a modern programming language. Solving business proble
ms using application packages including spreadsheets and databases. Basics of
web design. Collaborative tools. Using open source software.Course Component:
Laboratory, LectureThe courses ITI 1120, GNG 1106, CSI 1306, CSI 1308, CSI 13
90 cannot be combined for credits.
```

The HTML is then parsed using `beautifulsoup4`, such that it is stored in the following format.

In [2]:
```python
output = to_json(html, filename=None)
print(output[0])
```

```
{'id': 0, 'title': 'CSI 1306 Computing Concepts for Business (3 units)', 'bod
y': 'Introduction to computer-based problem solving from the perspective of t
he business world. Design of algorithms for solving business problems. Basics
of computer programming in a modern programming language. Solving business pr
oblems using application packages including spreadsheets and databases. Basic
s of web design. Collaborative tools. Using open source software.'}
```

This format facilitates easily manipulating and rendering the data using other modules.

User Interface

File(s)

- `app.py`
- `templates/`
  - `static/`
    - `css/`
    - `js/`
    - `img/`
  - `index.html`
  - `layouts.html`
  - `about.html`
  - `404.html`

Description

This module is implemented as a small Flask app. The UI is a parody of Google (see acknowledgements section below), allowing users to search the catalogue using either retrieval module.

Demonstration

See Demo below for a full demonstration.

Dictionary building

File(s)

- `build_dictionary.py`
- `construct_index.py`
- `retrieval_model.py`

Description

This module contains utilities used to build the dictionary for the retrieval models. The actual construction of the dictionary takes place in `retrieval_model.py`.

Demonstration

Note that due to the inclusion of the optional phrase query indexing module, said module is also a part of the dictionary building process. This will be expanded upon further in its own section below.

First, we reference the data created from the scraping model as seen above. We narrow the data to focus only on the descriptions (the titles were **not** made part of the dictionary).

In [3]:
```
import json

data = json.load(open("data/catalogue-uottawa-ca.json"))
descriptions = [row['body'] for row in data]
descriptions[0]
```

Out[3]:
```
'Introduction to computer-based problem solving from the perspective of the b
usiness world. Design of algorithms for solving business problems. Basics of
computer programming in a modern programming language. Solving business probl
ems using application packages including spreadsheets and databases. Basics o
f web design. Collaborative tools. Using open source software.'
```

These descriptions are then passed to the `clean` function from `build_dictionary.py`, which performs casing, normalization, lemmatization, stopword removal, and tokenization.

In [4]:
```
from build_dictionary import clean
print(clean(descriptions[0]))
```

```
{'source', 'open', 'perspective', 'language', 'problem', 'programming', 'pack
age', 'including', 'collaborative', 'computer', 'spreadsheet', 'based', 'soft
ware', 'business', 'web', 'tool', 'modern', 'using', 'algorithm', 'design',
'introduction', 'world', 'solving', 'application', 'database', 'basic'}
```

Inverted Index Construction

File(s)

- `construct_index.py`

Description

Demonstration

The descriptions above are passed to the `build_postings` function, which returns both the index and the term dictionary. The index is constructed first, as follows:

In [5]:
```python
from construct_index import build_postings

index, term_dict = build_postings(descriptions)
print(index[0])
```

```
(0, ['source', 'open', 'spreadsheet', 'perspective', 'language', 'problem',
'package', 'programming', 'web', 'modern', 'database', 'problem', 'using', 'a
lgorithm', 'including', 'collaborative', 'computer', 'design', 'introductio
n', 'world', 'solving', 'tool', 'application', 'based', 'software', 'busines
s', 'basic'])
```

Then, the term dictionary is constructed from the index. As it is, of course, a dictionary, retrieval is completed in **constant** $\mathcal{O}(1)$ time.

Its construction is completed in $\mathcal{O}(n \cdot (m+1))$ time, where m is the number of words in the longest document in the corpus. The n documents contained in the `.json` data are iterated over once to construct the index (as seen above) containing n rows. Then, each of the n rows of the index is to construct the term dictionary. At each row, m words are iterated over, incrementally updating its associated count. Pseudocode is presented below.

```python
for (doc_id, words) in index:
    for word in words:
        term_dict[doc_id][word] += 1
```

The outer loop performs n iterations of m steps each for $O(n \cdot m)$ time. Combined with the n iterations of the index construction, we have $\mathcal{O}(n + n \cdot m) = \mathcal{O}(n \cdot (m+1))$ time.

Finally, both are returned.

Note: As an implementation decision, the weights are calculated later, in the `VSM` model itself. This simply made for a cleaner and more organized project structure. The weights could have been included here otherwise.

In [6]:
```python
print(term_dict['algorithm'])
```

```
{0: 1, 1: 1, 2: 1, 3: 1, 4: 1, 9: 2, 17: 1, 21: 1, 31: 2, 34: 1, 37: 1, 39:
1, 49: 1, 51: 1, 52: 1, 56: 1, 67: 1, 72: 1, 74: 1, 75: 1, 76: 1, 77: 1, 79:
1, 85: 1, 88: 1, 89: 1, 94: 1}
```

Corpus Access

File(s)

- same as in User Interface

Description

This module is implemented within the user interface, seen within the search results.

Demonstration

See Demo below for a full demonstration.

Boolean model of information retrieval

File(s)

- `brm.py`

Description

The `BRM` (Boolean Retrieval Model) class operates as a model to query the corpus using boolean searches.

Demonstration

The `boolean.py` library is used for symbolic manipulation. The query is parsed as a logical expression, and then each literal word of the query is cleaned using the previous module.

In [7]:
```
from brm import *

b = BRM()
q = "(computer OR systems) AND (data)"

b.preprocess_query(q)
```

```
wildcards: []
computer OR systems AND data
```

Out[7]:
```
OR(Symbol('computer'), AND(Symbol('systems'), Symbol('data')))
```

For each document, the truth value of each symbol is evaluated as the presence of the word in the document to resolve the expression. Each of the n documents are evaluated, for a query complexity of $\mathcal{O}(n)$.

As the `BRM` is unranked, the `top_n` argument will simply return the first `n` documents found and is only useful for demonstration purposes (as in this instance) or for the UI to paginate the results (not yet implemented).

In [8]:
```
b.query(q, top_n=5)
```

```
wildcards: []
computer OR systems AND data
```

Out[8]:

	title	body
0	CSI 1306 Computing Concepts for Business (3 un...	Introduction to computer-based problem solving...
1	CSI 1308 Introduction to Computing Concepts (3...	Introduction to computer based problem solving...
2	CSI 1390 Introduction to Computers (3 units)	Computing and computers. Problem solving and a...
3	CSI 2101 Discrete Structures (3 units)	Discrete structures as they apply to computer ...
21	CSI 4109 Introduction to Distributed Computing...	Computational models. Communication complexity...

Vector Space Model of information retrieval

File(s)

- `vsm.py`

Description

The `VSM` (Vector Space Model) class operates as a model to query the corpus using $\texttt{tf} - \texttt{idf}$ retrieval.

Demonstration

The $\texttt{tf} - \texttt{idf}$ scores are computed for each w_i, d_j pair upon construction of the model.

In [9]:
```python
from vsm import *

v = VSM()
v.d_w.T.head()
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	...	94	9
source	1.711807	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	...	0.0	0.
open	1.711807	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	...	0.0	0.
spreadsheet	2.012837	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	...	0.0	0.
perspective	1.313867	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	...	0.0	0.
language	0.808717	0.808717	0.0	0.0	0.0	0.808717	0.0	0.0	0.808717	0.0	...	0.0	0.

5 rows × 103 columns

At runtime, the query is pre-processed in the same fashion as the documents were processed orginally.

In [10]:
```python
q = "operating systems"
q = clean(q)
q
```

Out[10]: `{'operating', 'system'}`

Only relevant documents are selected from the weight matrix. As the matrix is a 2-d array, this access takes place in $\mathcal{O}(w)$ time, where w is the number of words in the processed query, which is effectively constant $\mathcal{O}(1)$ time (assuming the user does not query every word in the vocabulary).

This is mathematically equivalent to performing an inner product on the weight matrix with one-hot vectors for each word, but it is clearly more space-efficient to simply slice the rows in question.

In [11]:
```
v.d_w.loc[:, q].T
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	...	94	96	97	98	99	100	101	1
system	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
operating	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(

2 rows × 103 columns

The matrix is then summed along each column (corresponding to a document in the corpus), and sorted in descending order. This is then returned to our original data set and used to return the relevant documents to the user.

In [12]:
```
v.d_w.loc[:, q].T.apply(sum).sort_values(ascending=False)[:5]
```

Out[12]:
```
30    2.573724
12    2.573724
90    2.573724
83    1.992251
91    1.162947
dtype: float64
```

The "confidence" score is obtained by using the $\mathbf{tf}-\mathbf{idf}$ score through the sigmoid function (the s-curve), squishing all values between $[0, 1]$ with more extreme values on both ends. As of writing, this is not currently displayed to the user, but could be in the future.

In [13]:
```
v.query("operating systems", top_n=10)
```

Out[13]:

	title	body	confidence
id			
30	CSI 4139 Design of Secure Computer Systems (3 ...	Security policies. Security mechanisms. Physic...	0.929151
12	CSI 3131 Operating Systems (3 units)	Principles of operating systems. Operating sys...	0.929151
90	CSI 5312 Distributed Operating Systems Enginee...	Design issues of advanced multiprocessor distr...	0.929151
83	CSI 5175 Mobile Commerce Technologies (3 units)	Wireless networks support for m-commerce; m-co...	0.879981
91	CSI 5314 Object-Oriented Software Development ...	Issues in modeling and verifying quality and v...	0.761868
56	CSI 5134 Fault Tolerance (3 units)	Hardware and software techniques for fault tol...	0.761868
32	CSI 4141 Real Time Systems Design (3 units)	Definition of real-time systems; examples. C...	0.761868
89	CSI 5311 Distributed Databases and Transaction...	Principles involved in the design and implemen...	0.761868
24	CSI 4124 Foundation of Modelling and Simulatio...	The modelling and simulation process from a pr...	0.761868
35	CSI 5100 Data Integration (3 units)	Materialized and virtual approaches to integra...	0.641406

Modules - Optional

Phrase Query Indexing

File(s)

- `phrase_indexing.py`

Description

This module uses the Jaccard coefficient to identify phrases from a candidate set (identified as hyphenated words), as seen in class.

Demonstration

First, the candidate set is assembled as all hyphenated words in the corpus.

In [14]:
```
from phrase_indexing import *

candidates = identify_candidates(descriptions)
candidates[:5]
```

Out[14]:
```
[('computer', 'based'),
 ('object', 'oriented'),
 ('binary', 'trees'),
 ('object', 'oriented'),
 ('constraint', 'based')]
```

Then, bigrams of the corpus text are constructed, for calculation of Jaccard coefficients.

In [15]:
```
corpus = [remove_punc(t.lower(), rm_hyphens=True) for t in descriptions]
bigrams = make_bigrams(corpus)

print(find_phrases(candidates, bigrams, threshold=0.7))
```

```
{'entity-relationship': 1.0, 'one-way': 1.0, 'trap-door': 1.0, 'grey-box': 1.
0, 'well-separated': 1.0, 'fail-safe': 1.0, 'locality-sensitive': 1.0, 'large
-scale': 1.0}
```

Of course, the threshold can be tuned as well.

In [16]:
```
print(find_phrases(candidates, bigrams, threshold=0.5))
```

```
{'object-oriented': 0.5294117647058824, 'entity-relationship': 1.0, 'one-wa
y': 1.0, 'trap-door': 1.0, 'lambda-calculus': 0.6666666666666666, 'grey-box':
1.0, 'well-separated': 1.0, 'fail-safe': 1.0, 'locality-sensitive': 1.0, 'lar
ge-scale': 1.0, 'fault-tolerance': 0.5555555555555556}
```

These phrases are then stored for later use in preprocessing the corpus and user queries.

Spelling Correction

File(s)

- `spelling.py`

Description

This module performs spelling corrections on user queries. However, as this is an extra module (working alone, I only have to implement one), I took some liberties with my implementation and elected to use a character-gram model instead of minimum edit distance (mostly out of curiosity).

Demonstration

First, as usual, the query is pre-processed (note: this happens slightly differently for the `BRM`).

In [17]:
```
from spelling import *

q = "data maagement"
q = clean(q)
q
```

Out[17]:
```
{'data', 'maagement'}
```

Then, for each word in the query, if the word is not in the vocabulary, the spell check is applied. This is performed by computing the ratio of shared character-bigrams between the words. If the word is in the vocabulary, it is simply ignored (returned with 100% confidence). The word is also ignored if it is shorter than the `minimum_word_len` parameter, the default value of which is `4`, or if it is in `nltk.corpus.stopwords.words()`.

If the word ends in 's', and the word's singular form is in the vocabulary, then the word's singular form is returned. (*Note: This is a potential argument for having used stemming over lemmatization. Feedback on this is welcome.*)

Like minimum edit distance, then, it cannot correct words that are already spelled correctly; it is not context-aware.

In [18]:
```
for w in q:
    top_candidate, conf = spell_check(w, b.build_vocab())[0]
    print(f"{w} => {top_candidate} ({conf*100}%)")
```

```
data => data (100%)
maagement => management (50.0%)
```

As well, the threshold for which candidates are returned can be adjusted from its default value of 0.25. The low threshold reflects the choice of trigrams instead of bigrams as well as the fact that having at least one match is preferable to none at all (as the UI presently only takes the first match, though this will change in the future).

```
In [19]: print(spell_check("computional", b.build_vocab(), threshold=0.35))
```

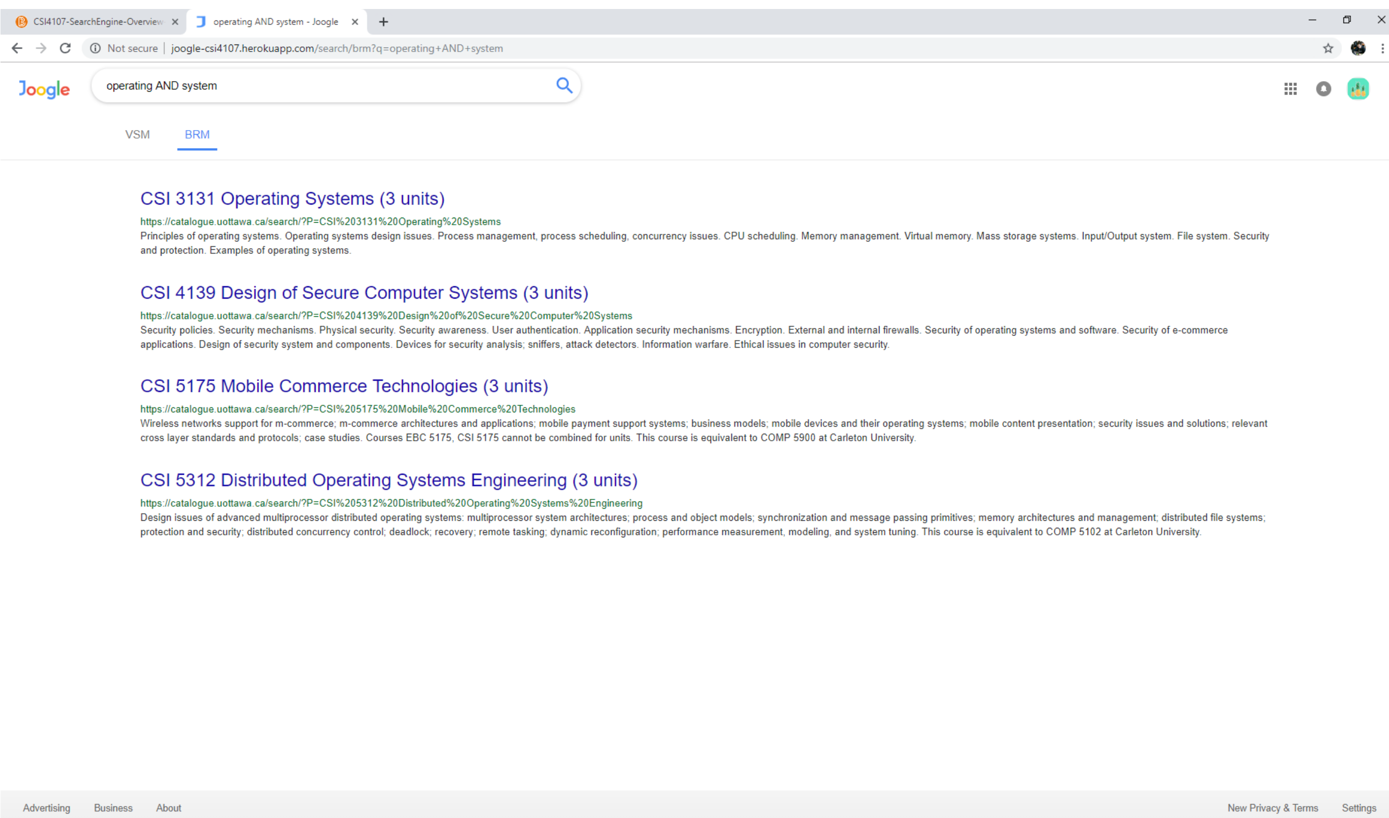
```
[('computational', 0.6666666666666666), ('computation', 0.5), ('computing', 0.45454545454545453), ('computer', 0.36363636363636365)]
```

Demo

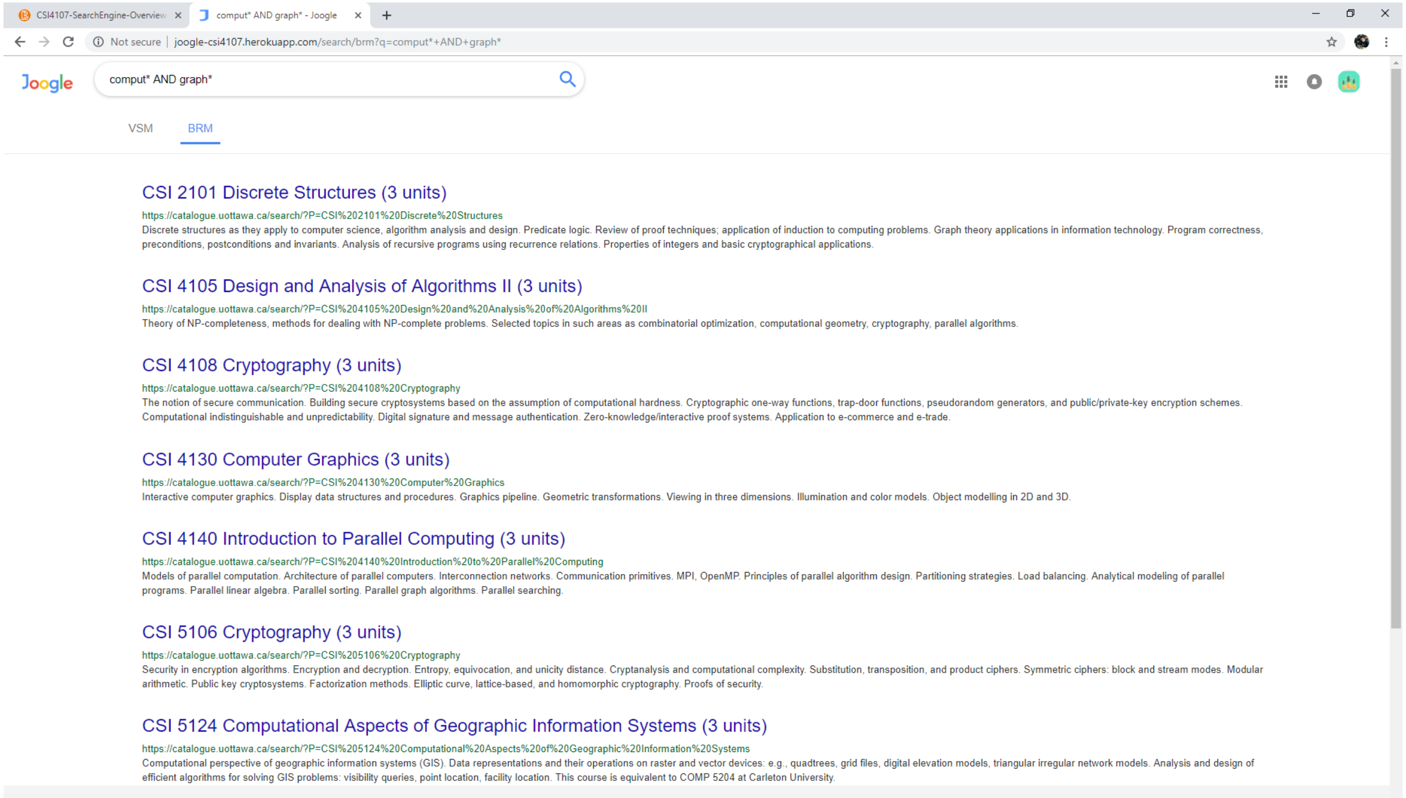
The following demonstrations are reflective of the [live website (https://joogle-csi4107.herokuapp.com)](https://joogle-csi4107.herokuapp.com) as of the date of submission.

BRM

(operating AND system)



(comput *AND graph*)


CSI4107-SearchEngine-Overview | comput* AND graph* - Joogle

Not secure | joogle-csi4107.herokuapp.com/search/brm?q=comput*+AND+graph*

Joogle comput* AND graph*

VSM BRM

CSI 2101 Discrete Structures (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%202101%20Discrete%20Structures
Discrete structures as they apply to computer science, algorithm analysis and design. Predicate logic. Review of proof techniques; application of induction to computing problems. Graph theory applications in information technology. Program correctness, preconditions, postconditions and invariants. Analysis of recursive programs using recurrence relations. Properties of integers and basic cryptographical applications.

CSI 4105 Design and Analysis of Algorithms II (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%204105%20Design%20and%20Analysis%20of%20Algorithms%20II
Theory of NP-completeness, methods for dealing with NP-complete problems. Selected topics in such areas as combinatorial optimization, computational geometry, cryptography, parallel algorithms.

CSI 4108 Cryptography (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%204108%20Cryptography
The notion of secure communication. Building secure cryptosystems based on the assumption of computational hardness. Cryptographic one-way functions, trap-door functions, pseudorandom generators, and public/private-key encryption schemes. Computational indistinguishable and unpredictability. Digital signature and message authentication. Zero-knowledge/interactive proof systems. Application to e-commerce and e-trade.

CSI 4130 Computer Graphics (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%204130%20Computer%20Graphics
Interactive computer graphics. Display data structures and procedures. Graphics pipeline. Geometric transformations. Viewing in three dimensions. Illumination and color models. Object modelling in 2D and 3D.

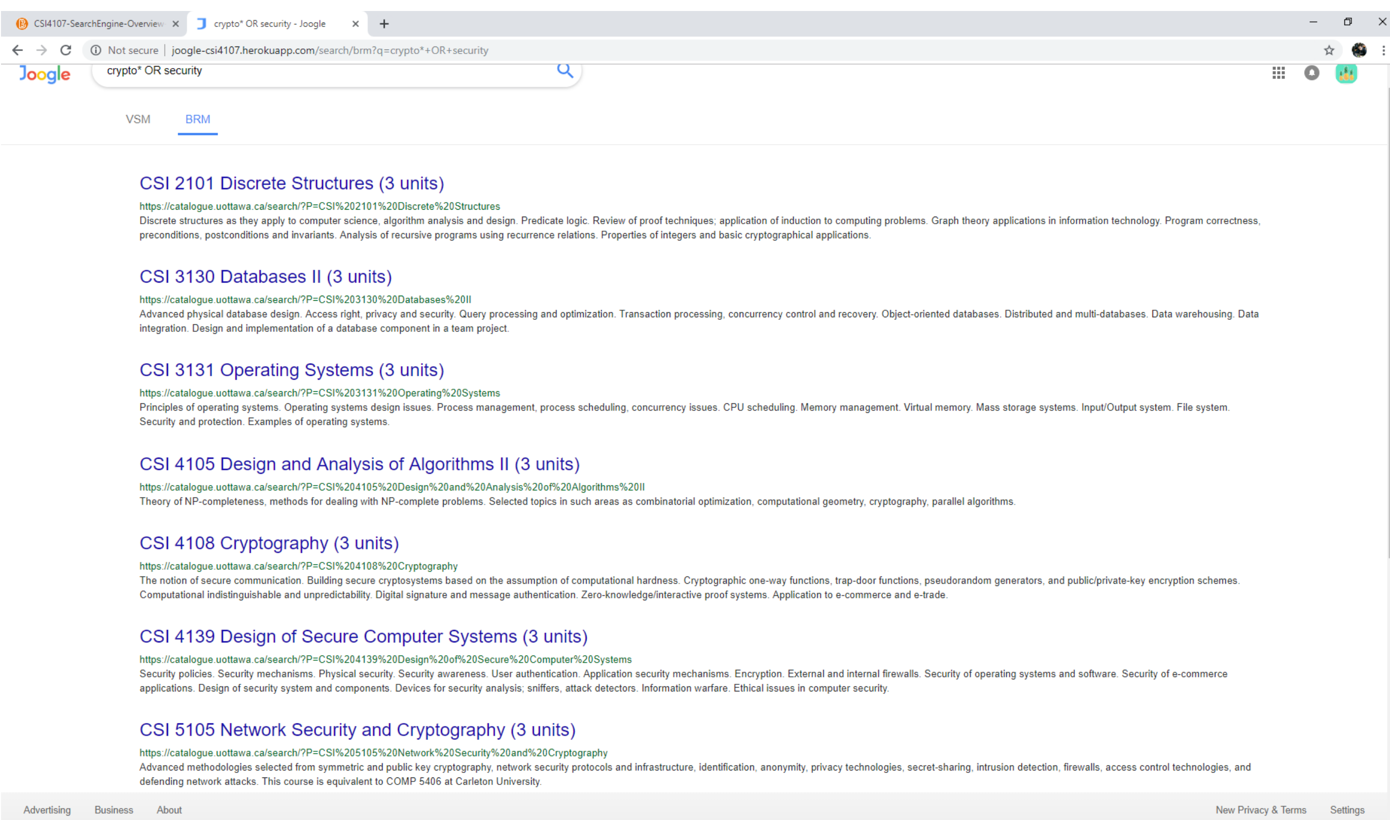
CSI 4140 Introduction to Parallel Computing (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%204140%20Introduction%20to%20Parallel%20Computing
Models of parallel computation. Architecture of parallel computers. Interconnection networks. Communication primitives. MPI, OpenMP. Principles of parallel algorithm design. Partitioning strategies. Load balancing. Analytical modeling of parallel programs. Parallel linear algebra. Parallel sorting. Parallel graph algorithms. Parallel searching.

CSI 5106 Cryptography (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%205106%20Cryptography
Security in encryption algorithms. Encryption and decryption. Entropy, equivocation, and unicity distance. Cryptanalysis and computational complexity. Substitution, transposition, and product ciphers. Symmetric ciphers: block and stream modes. Modular arithmetic. Public key cryptosystems. Factorization methods. Elliptic curve, lattice-based, and homomorphic cryptography. Proofs of security.

CSI 5124 Computational Aspects of Geographic Information Systems (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%205124%20Computational%20Aspects%20of%20Geographic%20Information%20Systems
Computational perspective of geographic information systems (GIS). Data representations and their operations on raster and vector devices: e.g., quadtrees, grid files, digital elevation models, triangular irregular network models. Analysis and design of efficient algorithms for solving GIS problems: visibility queries, point location, facility location. This course is equivalent to COMP 5204 at Carleton University.

Advertising Business About New Privacy & Terms Settings

(crypto* OR security)


CSI4107-SearchEngine-Overview | crypto* OR security - Joogle

Not secure | joogle-csi4107.herokuapp.com/search/brm?q=crypto*+OR+security

Joogle crypto* OR security

VSM BRM

CSI 2101 Discrete Structures (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%202101%20Discrete%20Structures
Discrete structures as they apply to computer science, algorithm analysis and design. Predicate logic. Review of proof techniques; application of induction to computing problems. Graph theory applications in information technology. Program correctness, preconditions, postconditions and invariants. Analysis of recursive programs using recurrence relations. Properties of integers and basic cryptographical applications.

CSI 3130 Databases II (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%203130%20Databases%20II
Advanced physical database design. Access right, privacy and security. Query processing and optimization. Transaction processing, concurrency control and recovery. Object-oriented databases. Distributed and multi-databases. Data warehousing. Data integration. Design and implementation of a database component in a team project.

CSI 3131 Operating Systems (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%203131%20Operating%20Systems
Principles of operating systems. Operating systems design issues. Process management, process scheduling, concurrency issues. CPU scheduling. Memory management. Virtual memory. Mass storage systems. Input/Output system. File system. Security and protection. Examples of operating systems.

CSI 4105 Design and Analysis of Algorithms II (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%204105%20Design%20and%20Analysis%20of%20Algorithms%20II
Theory of NP-completeness, methods for dealing with NP-complete problems. Selected topics in such areas as combinatorial optimization, computational geometry, cryptography, parallel algorithms.

CSI 4108 Cryptography (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%204108%20Cryptography
The notion of secure communication. Building secure cryptosystems based on the assumption of computational hardness. Cryptographic one-way functions, trap-door functions, pseudorandom generators, and public/private-key encryption schemes. Computational indistinguishable and unpredictability. Digital signature and message authentication. Zero-knowledge/interactive proof systems. Application to e-commerce and e-trade.

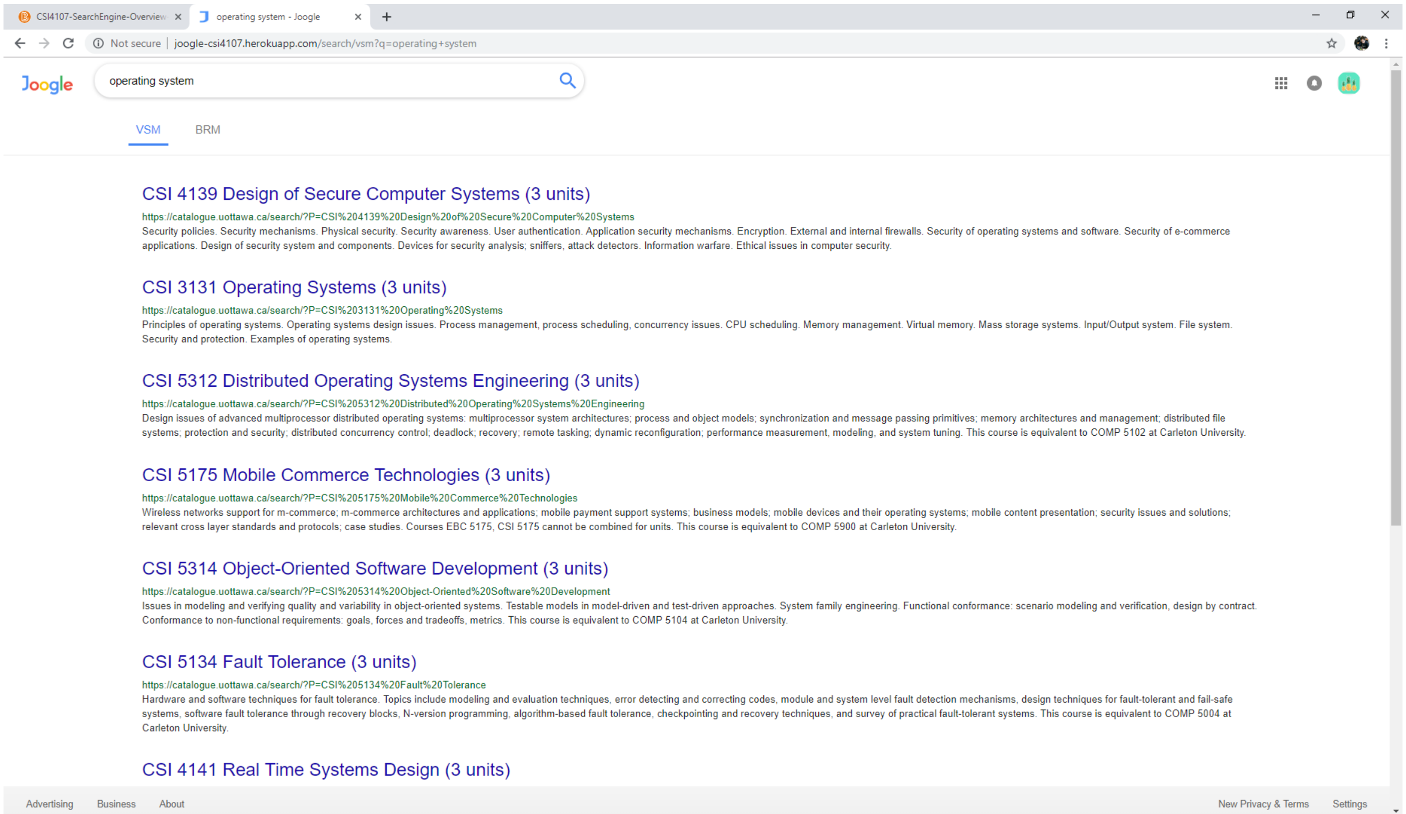
CSI 4139 Design of Secure Computer Systems (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%204139%20Design%20of%20Secure%20Computer%20Systems
Security policies. Security mechanisms. Physical security. Security awareness. User authentication. Application security mechanisms. Encryption. External and internal firewalls. Security of operating systems and software. Security of e-commerce applications. Design of security system and components. Devices for security analysis; sniffers, attack detectors. Information warfare. Ethical issues in computer security.

CSI 5105 Network Security and Cryptography (3 units)
https://catalogue.uottawa.ca/search/?P=CSI%205105%20Network%20Security%20and%20Cryptography
Advanced methodologies selected from symmetric and public key cryptography, network security protocols and infrastructure, identification, anonymity, privacy technologies, secret-sharing, intrusion detection, firewalls, access control technologies, and defending network attacks. This course is equivalent to COMP 5406 at Carleton University.

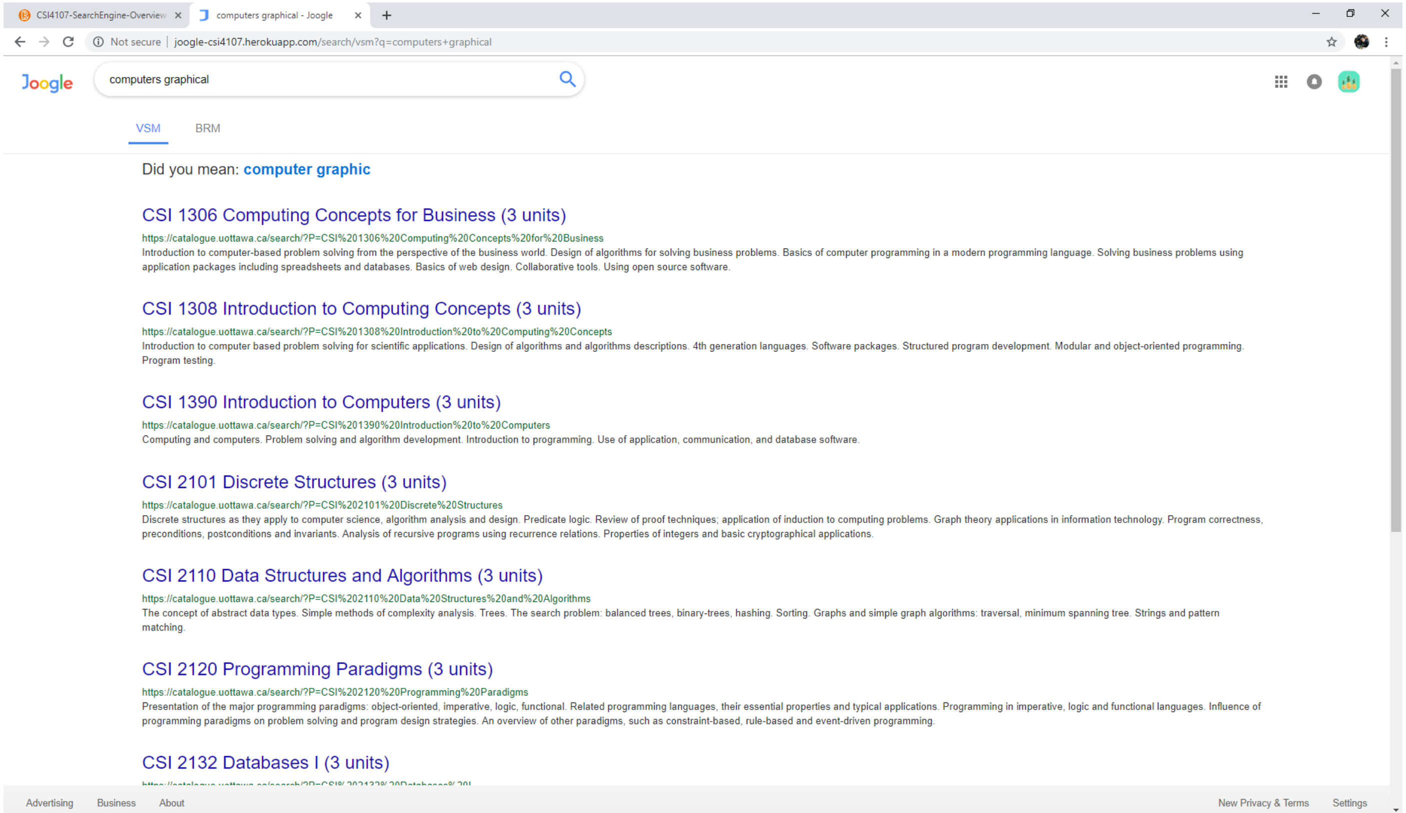
Advertising Business About New Privacy & Terms Settings

VSM

operating system



computers graphical


CSI4107-SearchEngine-Overview | computers graphical - Joogle

Not secure | joogle-csi4107.herokuapp.com/search/vsm?q=computers+graphical

Joogle computers graphical

VSM BRM

Did you mean: **computer graphic**

CSI 1306 Computing Concepts for Business (3 units)

https://catalogue.uottawa.ca/search/?P=CSI%201306%20Computing%20Concepts%20for%20Business

Introduction to computer-based problem solving from the perspective of the business world. Design of algorithms for solving business problems. Basics of computer programming in a modern programming language. Solving business problems using application packages including spreadsheets and databases. Basics of web design. Collaborative tools. Using open source software.

CSI 1308 Introduction to Computing Concepts (3 units)

https://catalogue.uottawa.ca/search/?P=CSI%201308%20Introduction%20to%20Computing%20Concepts

Introduction to computer based problem solving for scientific applications. Design of algorithms and algorithms descriptions. 4th generation languages. Software packages. Structured program development. Modular and object-oriented programming. Program testing.

CSI 1390 Introduction to Computers (3 units)

https://catalogue.uottawa.ca/search/?P=CSI%201390%20Introduction%20to%20Computers

Computing and computers. Problem solving and algorithm development. Introduction to programming. Use of application, communication, and database software.

CSI 2101 Discrete Structures (3 units)

https://catalogue.uottawa.ca/search/?P=CSI%202101%20Discrete%20Structures

Discrete structures as they apply to computer science, algorithm analysis and design. Predicate logic. Review of proof techniques; application of induction to computing problems. Graph theory applications in information technology. Program correctness, preconditions, postconditions and invariants. Analysis of recursive programs using recurrence relations. Properties of integers and basic cryptographical applications.

CSI 2110 Data Structures and Algorithms (3 units)

https://catalogue.uottawa.ca/search/?P=CSI%202110%20Data%20Structures%20and%20Algorithms

The concept of abstract data types. Simple methods of complexity analysis. Trees. The search problem: balanced trees, binary-trees, hashing. Sorting. Graphs and simple graph algorithms: traversal, minimum spanning tree. Strings and pattern matching.

CSI 2120 Programming Paradigms (3 units)

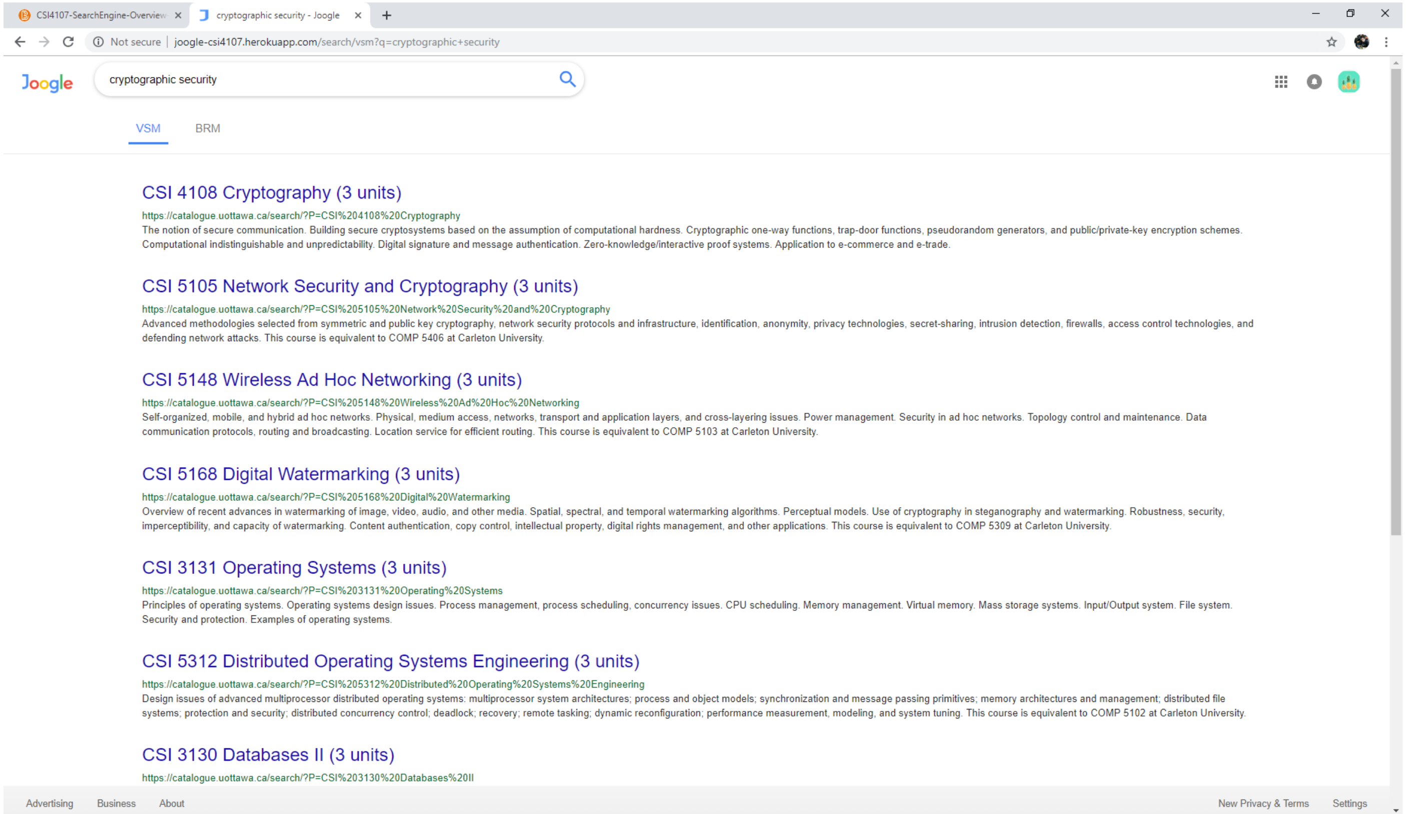
https://catalogue.uottawa.ca/search/?P=CSI%202120%20Programming%20Paradigms

Presentation of the major programming paradigms: object-oriented, imperative, logic, functional. Related programming languages, their essential properties and typical applications. Programming in imperative, logic and functional languages. Influence of programming paradigms on problem solving and program design strategies. An overview of other paradigms, such as constraint-based, rule-based and event-driven programming.

CSI 2132 Databases I (3 units)

Advertising Business About New Privacy & Terms Settings

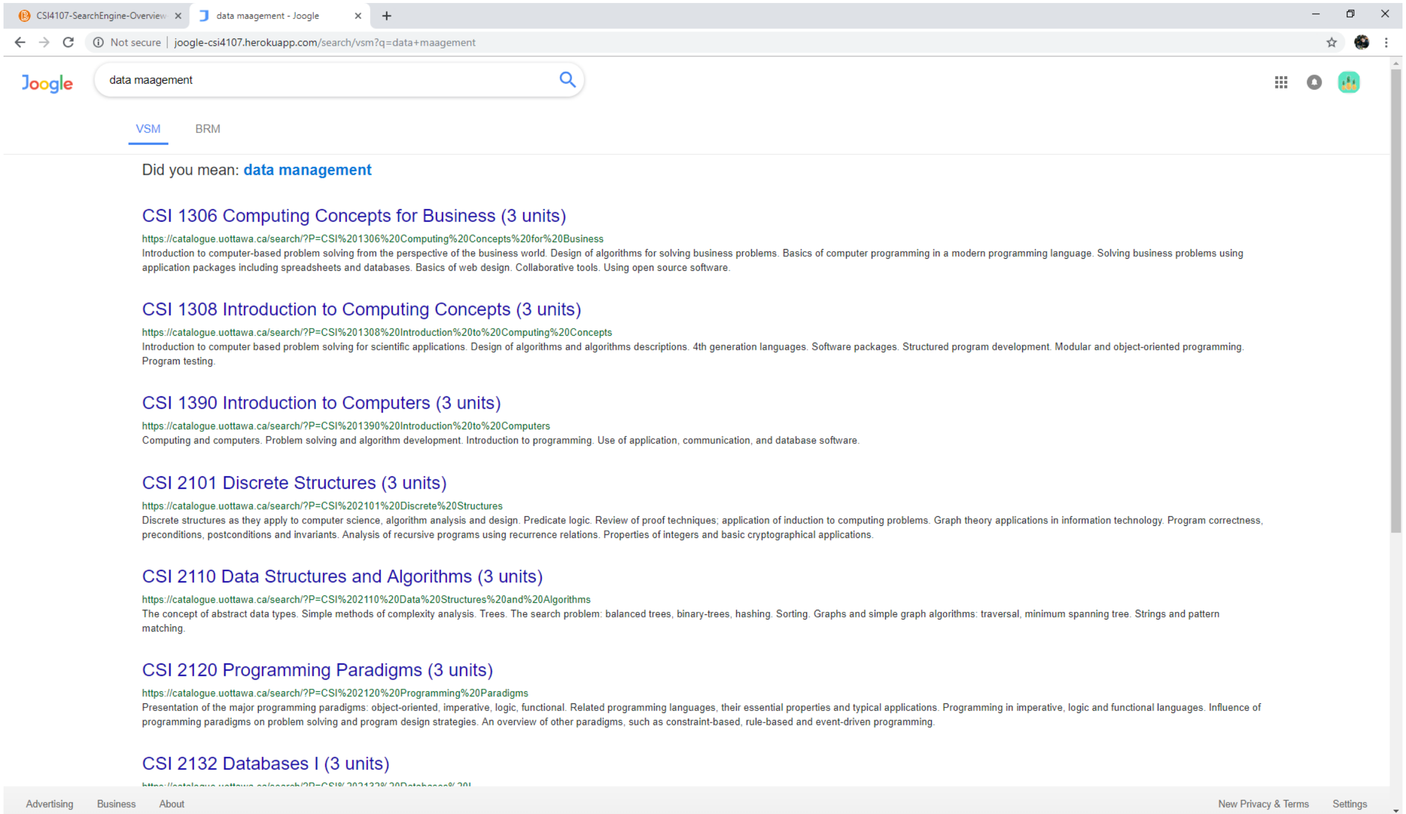
cryptographic security



Additional

data maagement

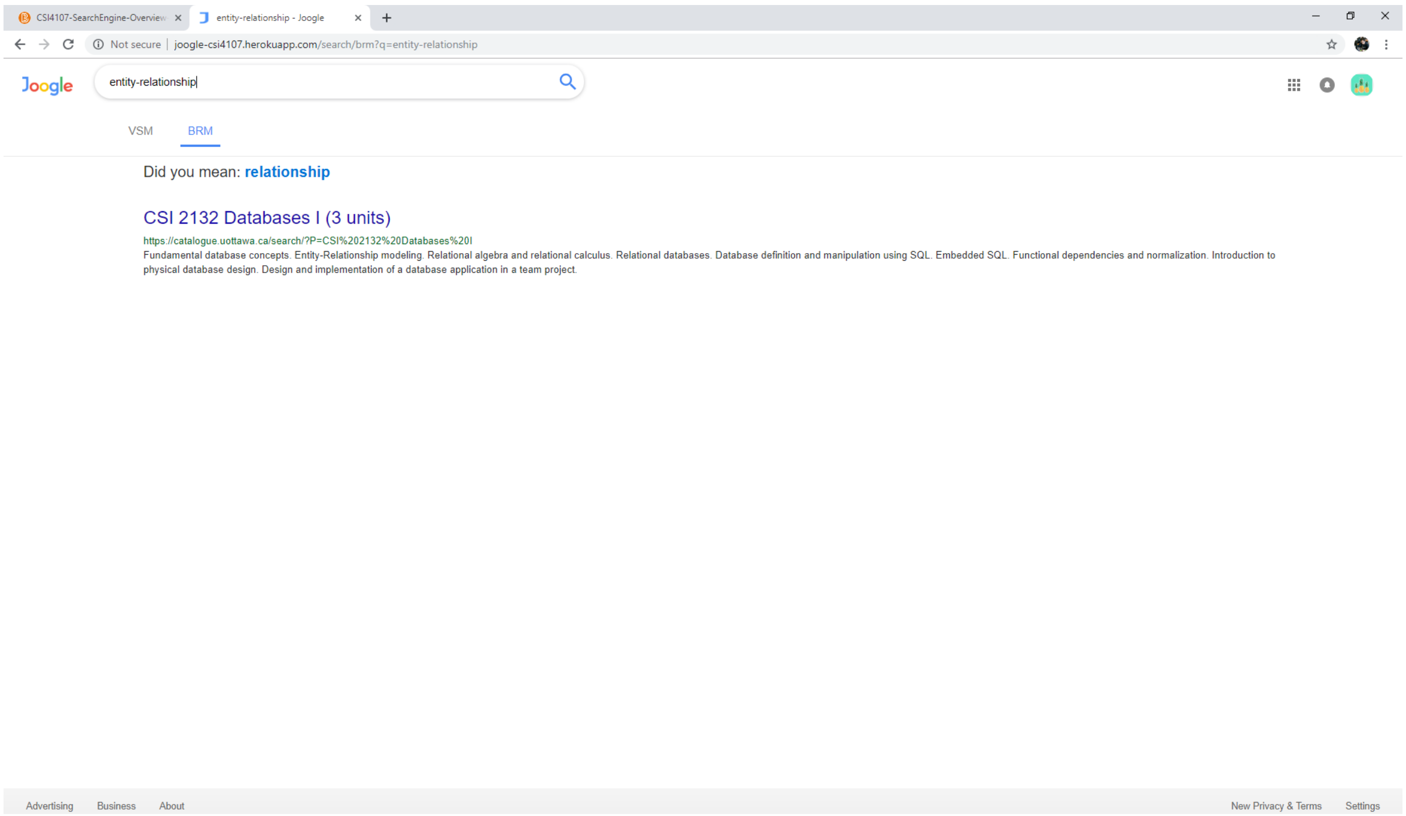
The first additional example showcases the implementation of the spell checking module (though it is also visible above). *Data maagement* is correctly revised and suggested to the user as *data management*.



entity-relationship

The second additional example showcases the implementation of phrase indexing. A search for `entity-relationship` would otherwise yield no results as *Entity-Relationship* would normally match neither (it would normally be stored as two separate words). However, the phrase is correctly indexed and matched in the boolean retrieval.

For some reason, the spell check fires here, despite the presence of the word in the vocabulary - this should not be the case and will be fixed in future versions.



Acknowledgements

First and foremost, most of the UI was inspired by/adapted from two sources. The index page was adapted from nicknish (https://codepen.io/nicknish/pen/cLbAg) on codepen.io, while the search result page was adapted from rmlynt (https://codepen.io/rmlynt/pen/AXkqJL). Rather than try to hide this by superficially changing things like variable names, I have left them primarily identical. However, all functionality (search, interactivity, infinite scrolling, etc.) is my own.

The logo was created festisite (https://www.festisite.com/logo/google/), and edited using lunapic (https://www110.lunapic.com/editor/).

As well, many code snippets from sources such as stack overflow have been referenced where appropriate throughout my code.