1. **Problem Statement**

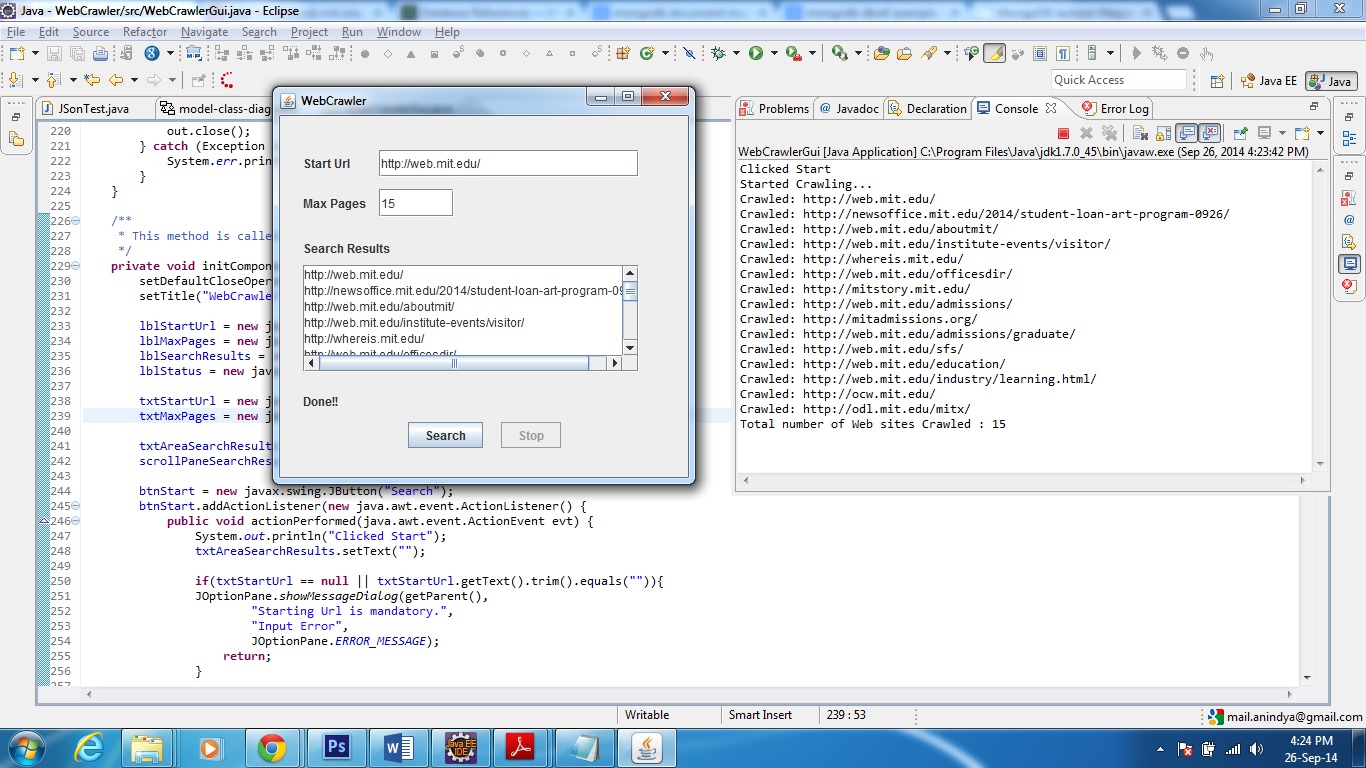
Write a web crawler which takes an initial URL of a university to start crawling and finds out all the hyperlinks from that URL and start crawling them too. The process continues. The goal of this crawling is to transform the unstructured/semi structured web page data of that given university into a structured form. Once the data is in a structured form we can then search on this data structure to find the url of a particular destination e.g. computer science department. This structured data can now be fed into analytics softwares to extract out meaningful information out of them.

Let us take example of MIT <http://web.mit.edu/> and we want to find out the computer science department. A human would do it intuitively by looking at the links and guessing and learning through experience and reach at the final destination through trial and error. We need to simulate this in a computer program and try to give a structure to this unstructured/semistructured web page data.

1. **Web Crawler**

The Webcrawler is rudimentary and currently 70% accurate mainly due to first two challenges as mentioned in the Challenges section. The Java program takes Start Url and Max Pages as inputs. Max Pages is the maximum number of unique web pages that it will crawl starting from the startUrl. It does not crawl already crawled pages. The program terminates when it has crawled Max no. of Pages or no more unique url to crawl. After crawling it dumps the content inside the <body> tag of each html file into a text file.

Below is a screenshot



**Case 1:** With startUrl as <http://web.mit.edu/> and Max Pages as 15 the results are

Clicked Start

Started Crawling...

Crawled: http://web.mit.edu/

Crawled: http://newsoffice.mit.edu/2014/student-loan-art-program-0926/

Crawled: http://web.mit.edu/aboutmit/

Crawled: http://web.mit.edu/institute-events/visitor/

Crawled: http://whereis.mit.edu/

Crawled: http://web.mit.edu/officesdir/

Crawled: http://mitstory.mit.edu/

Crawled: http://web.mit.edu/admissions/

Crawled: http://mitadmissions.org/

Crawled: http://web.mit.edu/admissions/graduate/

Crawled: http://web.mit.edu/sfs/

Crawled: http://web.mit.edu/education/

Crawled: http://web.mit.edu/industry/learning.html/

Crawled: http://ocw.mit.edu/

Crawled: http://odl.mit.edu/mitx/

Total number of Web sites Crawled : 15

**Case 2:** With startUrl as <http://engineering.tamu.edu/cse/people/ajiang> and Max Pages as 10 the results are

Clicked Start

Started Crawling...

Crawled: http://engineering.tamu.edu/cse/people/ajiang/

Crawled: https://services.tamu.edu/directory-search/

Crawled: http://aggiemap.tamu.edu/

Crawled: http://howdy.tamu.edu/

Crawled: http://engineering.tamu.edu/cse/

Crawled: http://faculty.cse.tamu.edu/ajiang/

Crawled: http://texas.gov/

Crawled: http://texashomelandsecurity.com/

Crawled: http://finance.tamu.edu/records/

Crawled: https://secure.ethicspoint.com/domain/en/report\_custom.asp?clientid=19681/

Total number of Web sites Crawled : 10

**Challenges**

* The hyperlink href urls are not always descriptive indicator of what they actually point to e.g.

href="<http://mitstory.mit.edu/>">history</a>

Though the link is for history but the actual href is mitstory

Another one

<a href="<http://odl.mit.edu/mitx/>"><i>MITx</i></a>

The link is for MIT Office of Digital Learning but it is difficult to make out from the href

So we cannot just only consider the href text but we also have to consider what is between the <a> and </a> tags

* Not all hyperlinks are absolute but sometimes they are relative and hence they are skipped and not crawled e.g.

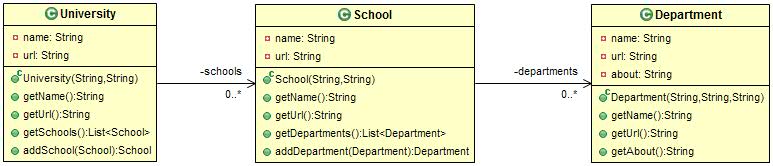
<a href="[/publications/](http://odl.mit.edu/mitx/)"><i>Publications</i></a>

* **Giving weights to search terms** – This is not incorporated yet in the crawler as I am not sure how to do it. For a given web page I have thought of 3 sections Url (weight 10%), title (weight 40%) and body (50%). If a text for example “Computer Science” is found in the body of a web page then I will give it a weight of 50% as compared to if found in title which I will give 40%. These are some guesstimate numbers and do not have any mathematical or scientific background. Based on these average weighted sum I will rank each of these webpages and add them to a Priority Queue and then pick up the top element of that queue and search again. I chose priority queue because I need a FIFO behavior and also Priority Queue takes at max 2n comparisons to construct and O(log N) comparison to get the highest ranked element. I think this simple algorithm won’t work in real life and we need to determine weight by applying machine learning algorithms. Also, the frequency of occurrence of a particular term and the relevance of that term in that web page need to be considered too. I am sure there must be much better algorithms already in place.

1. **Class Diagram**

Let us consider a trivial scenario with only 3 entities with which we try to model the search domain, University, School and Department. Almost all the universities in USA has at least these 3 entities in their website some way or the other. Schools may be called Colleges in some university.

Below is the class Diagram



A University has many schools and each school has many departments.

1. **Structured Data Representation**
   1. **Using JSON based representation – Using MongoDB**

A sample representation would be like below

{

"name" : "Texas A & M University",

"url" : "https://www.tamu.edu/",

"schools" : [ {

"name" : "Dwight Look College of Engineering",

"url" : "https://www.tamu.edu/about/departments.html#engineering",

"departments" : [ {

"name" : "Computer Science and Engineering",

"url" : "http://engineering.tamu.edu/cse/",

"about" : "About Department of Computer Science and Engineering"

}, {

"name" : "Electrical and Computer Engineering",

"url" : "http://engineering.tamu.edu/electrical",

"about" : "About Department of Electrical and Computer Engineering"

} ]

} ]

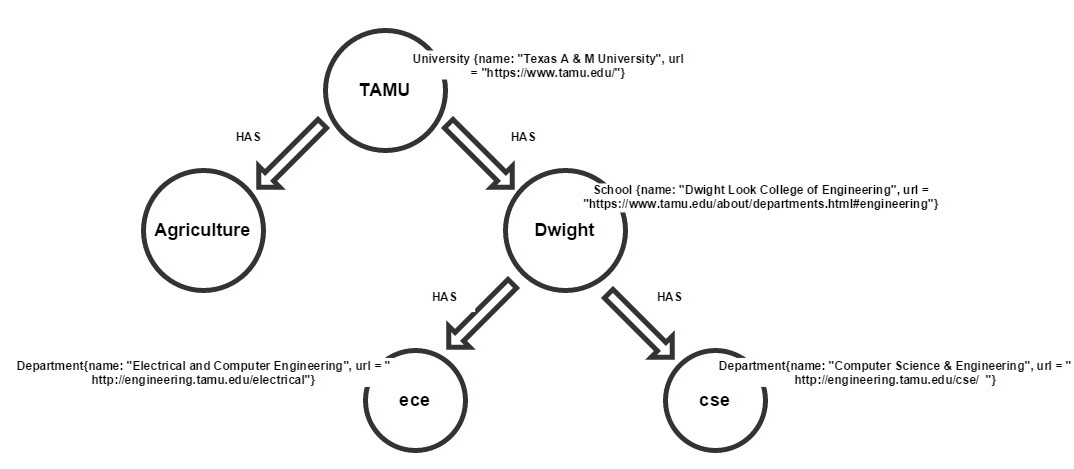
}

We can leverage MongoDB to store these objects and do appropriate query from db to get the desired result.

* 1. **Using Graph Database – Using Neo4j**

<http://www.neo4j.org/>

Neo4j is a highly scalable, robust (fully ACID) native graph database which supports Cypher query to extract out nodes or paths efficiently from complex graphs.



Here the entities University, School and Department are modelled as nodes of a graph and each of the node can have attributes (e.g. name, url). Also the edges between the nodes can be have roles e.g. HAS and the roles can also have attributes (not shown here). Now if one is interested in finding the url of the computer science department then he can issue a Cypher query as below

MATCH (univ:University)-[:HAS]->(School)->[:HAS]->(dept:Department)

WHERE univ.name = "TAMU" and dept.name=”Computer Science”

RETURN dept.url as Url

This will initiate a search within the graph database using graph algorithms and provide us the Department node or the attributes of that node we are interested in e.g. the url of the department.

Out of these 2 representations above I liked the Graph Database approach because it is more in synch with how webpages are structured and more intuitive.

1. **Books**

[*Introduction to Information Retrieval*](http://nlp.stanford.edu/IR-book/information-retrieval-book.html), Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze

1. **Courses**

Machine Learning course of Coursera by Dr. Andrew Ng of Stanford

1. **Tools**

I found the following can be useful in Machine Learning and Java

* Java Machine Learning Library (Java-ML) - <http://java-ml.sourceforge.net/> This is a collection of machine learning algorithms in Java
* A list of Machine Learning implementations and implementers in Java - <http://machinelearningmastery.com/java-machine-learning/>
* <http://www.inf.ed.ac.uk/teaching/courses/dme/html/Tutorial.pdf> - WEKA: Machine Learning Algorithms in Java. This tutorial is Chapter 8 of the book Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.