Big Data Processing: Coursework 2

BDP-03: Implementation of a large-scale Search Engine

Group I

Members:

Harrishan Sureshkumar,

Partiban Ramasamy Uthuchia – Pillay,

Qaiser Ali,

Shareef Sabra,

Akhil Nair

Contents

1: Motivation of the problem:

The objective of this project is to create an efficient search engine which processes the entire Wikipedia dataset and outputs results based on relevance to a chosen search term and how likely the produced result is to be the desired result.

Algorithms are utilised to calculate the priority and relevance of each article to what the user searches for. In addition, multiple ranking strategies were created in order to output results that match user preference to a high degree using; element tag extraction, priority assignment based on keyword densities and spam filtering.

2: Explanation of selected techniques:

For this project, it was collectively decided that using the Apache Hadoop framework along with java and map/reduce would be the more suitable method for carrying out this task. The reason for this decision was mainly because majority of the group members were already familiar with Hadoop over Spark and due to the lack of a distributed storage system in Spark.

The project will use the English Wikipedia dataset which is located in HDFS at **"/data/wiki" on** the Queen Mary servers. Wikipedia uses a style of mark-up which is unique to their website, therefore in order to access the dataset and analyse each article, the Cloud9 library was imported, more specifically the *'EnglishWikipediaPage'* library. This provides methods which were imperative to accessing the Wikipedia articles and obtaining features of the article [1]. The Cloud9 library was used only to parse the input of the Wikipedia dataset as well as to obtain the title of a page using the '*getTitle()'* method and the text of each page using the '*getWikiMarkup()'* method.

3: Design Rationale – Implementation, details and explanation:

Within the main class, there are three separate jobs which run alongside each other; each job contains a mapper and a reducer. The main class waits for each job to finish before moving on to the next job, the output data is then passed from the previous job as the input data for the next job.

Job 1: Building the Inverted Index

The first map reduce job builds an inverted index. This reads in the search term entered in terminal and stores it in a string to be used to check if each page contains the word. The pages that include the search term within its wiki mark-up are stored in the inverted index in sequence format. The key stored is the ID of the page and the value store is the Wikipedia page itself.

Job 2: Implementation of our own Page Rank Algorithm

Literature Review:

Upon weeks of research we found some key features used by other search engines, such as google [2], that we decided to implement in our Page Rank algorithm using our own priority assignment algorithms using techniques such as; keyword element extraction, keyword densities of pages and spam filtering. This map/reduce job extracts data from the page and uses certain aspects of the page to give it a priority. As the second job of the search engine main class, its purpose was to give each page passed to the 2nd mapper from the 1st mapper an integer value that represents it's "priority". A combination of multiple ranking strategies were used to compute the results. This is done using multiple methods to get a more accurate value for priority and so the results output is an imitation of searching the same word on the Wikipedia website. Our methods are implemented in the PageRank Mapper. The PageRank Mapper first gets the search term input in the terminal and sets it to the variable named "searchWord." This is so that we know what word we are searching for in each page. This variable heavily influences the accuracy of our multiple ranking algorithms.

Job 2.1: Obtaining element tags

The PageRank mapper takes in the page ID and corresponding EnglishWikipediaPage page per iteration, this is provided by the Cloud9 library which allows us to parse raw xml as a Wikipedia object. There are in-built methods within Cloud9, such as *page.getWikiMarkup(),* this returns the wiki mark-up of the page as a string, that allow us to view the page in a comprehendible format. (Other cloud9 methods aren't used).

The first half of the page ranking algorithm is to utilise the wiki mark-up format and search for repetitions of the searchWord. We created a table (see below) indicating how much priority should be added for each word found and the format it is found in. E.g. If the searchWord is found and it is surrounded by the tags "==", it is a header (type 2) and aggregates 340 to the priority.

The following table shows a summary and examples of the priority aggregation issued when the search word is found in the following element tags:

Description	Wiki mark-up	Priority aggregation
Title	getTitle()	+1000
Headings (H2)	== Level 1 ==	+340
Headings (H3)	== Level 2 ==	+220
Headings (H4)	== Level 4 ==	+160
Headings (H5)	== Level 5 ==	+100
Headings (H6)	== Level 6 ==	+60
Bold	'''bold'''	+58
Italic	''italic''	+55
Internal link	[[link to wiki page]]	+140
External link	[external website]	+20
Image Caption	[[File:Wiki.png|thumb|Caption]]	+100
Normal	searchWord	+1

Additionally, other methods are used to contribute to priority such as the number of contributions per Wikipedia page.

Job 2.2: Keyword Density

The keyword density of a page is a value containing the percentage of the times a given keyword appears on a page of a website. Knowing the keyword density of a page allows us to determine whether the given page is relevant to the specified keyword used as the search term. [3]

The following formula represents how the KD of a page is calculated [4]:

$$KD = \frac{(Nkr * Nwp)}{(Tkn)} * 100$$

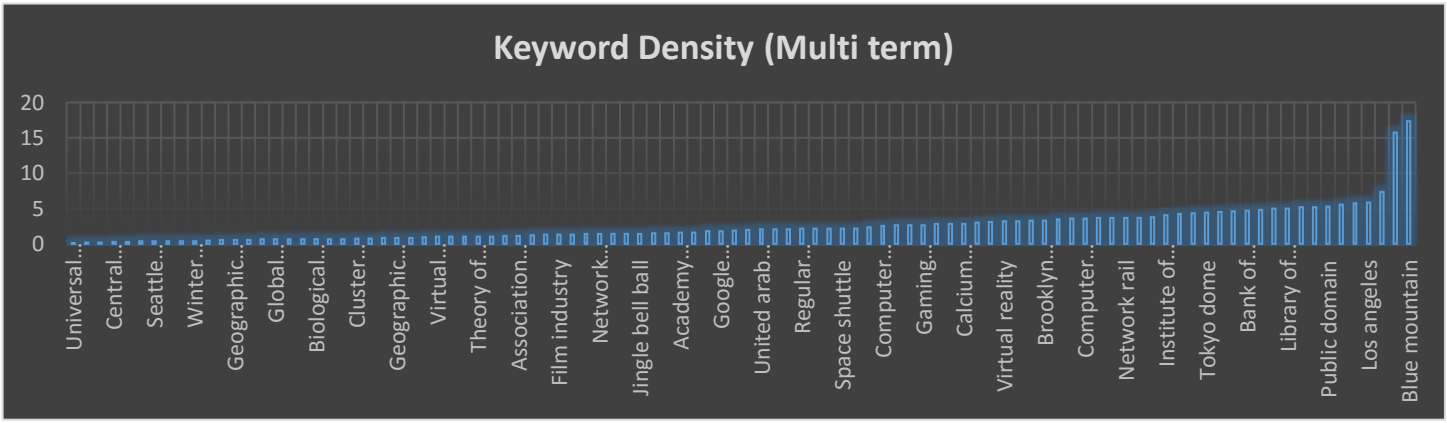
Where Nkr is the number of times the keyword/phrase occurs on the page, Nwp is the number of words in the phrase and Tkn is the total number of words on the whole page.

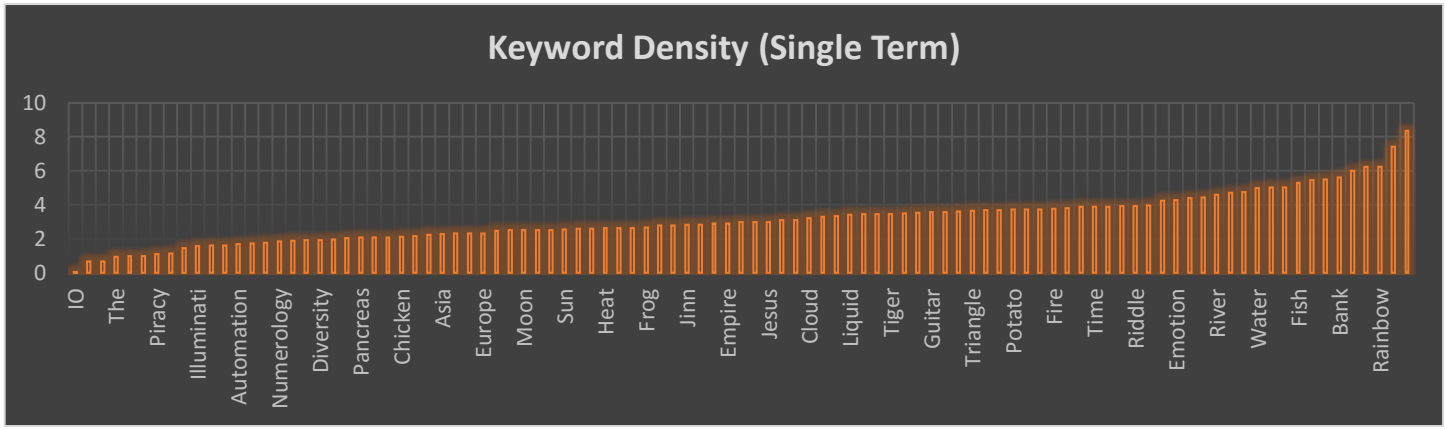
Job 2.2.1: Data collection

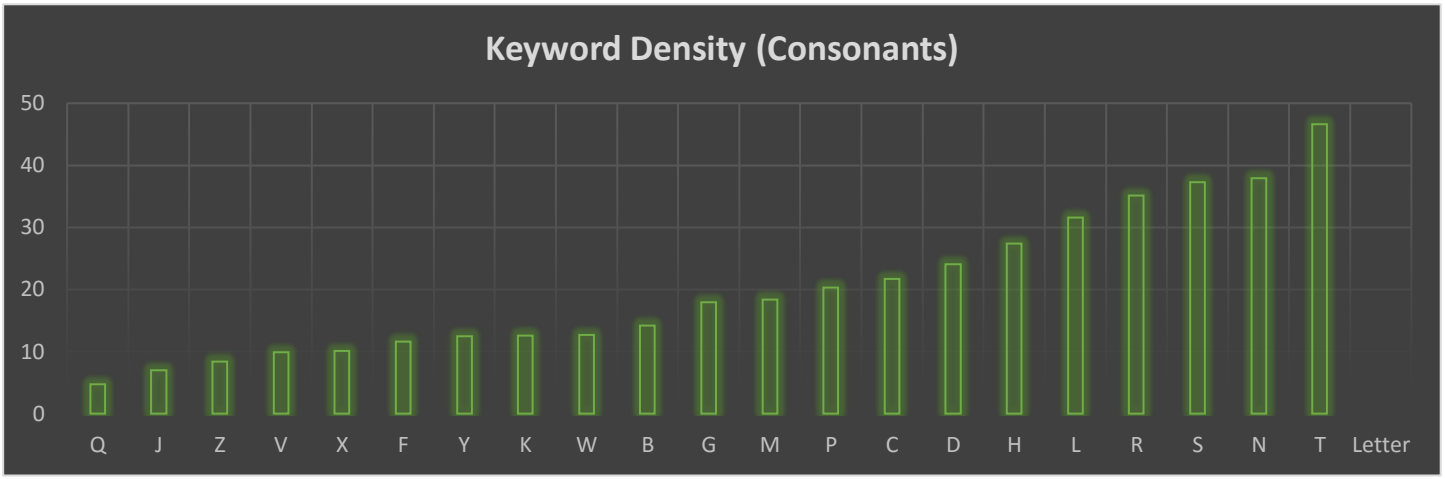
We decided to run some tests to find an optimal value for the keyword density (KD) which is optimized for the Wikipedia dataset. From our tests, we found that single words, multiple words, single characters and vowels had their own optimal ranges for the KD of a page. Based on this we collected data to find optimal ranges for these 4 types of search terms. (Check attached spreadsheet).

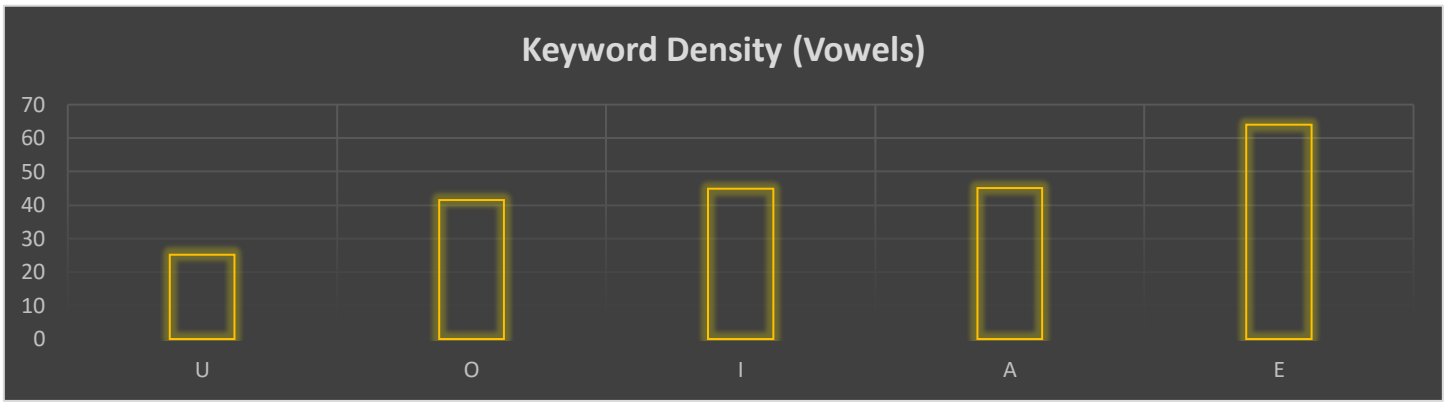
Job 2.2.2: Data Collection Results

We ran 100 tests for a collection of random single term search terms as well as another 100 tests for multi term searches, consonants (characters) and vowels. The following graphs summarise our results showing page title/ page's keyword density. (Actual collected data can be seen in the attached spreadsheet named "Data Collection").









Job 2.2:3 Data Analysis

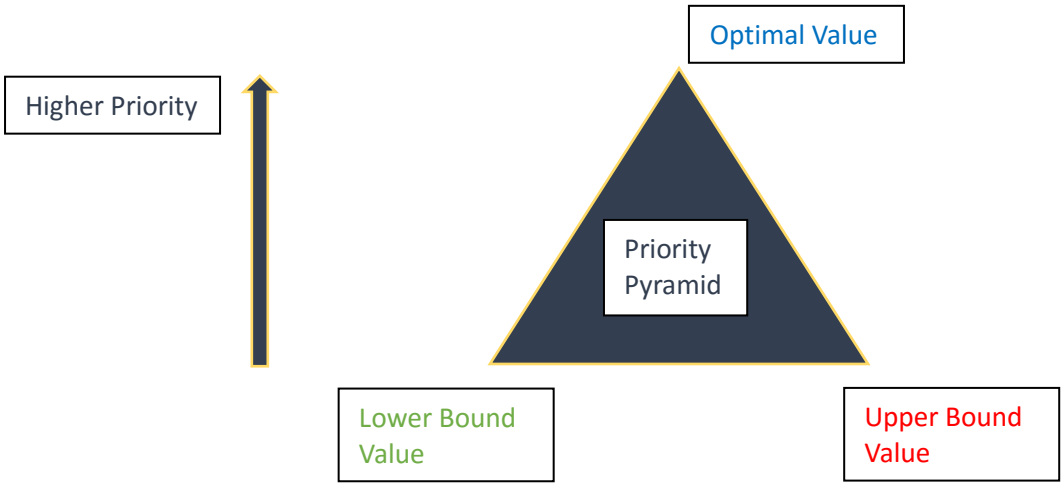
From the data we obtained, we worked out a range for the KD for each type of search term.

- The following table shows the average values obtained for the optimal value of the KD based on the search term type

	Lower bound	**Optimal Value**	**Upper bound**
Multi Term	0.0568	1.008	17.375
Single Term	0.0395	3.144	8.360
Character	4.759	20.091	46.662
Vowel	25.110	44.104	64.017

The lower bound is the smallest possible value obtained for the KD throughout our data and the upper bound represents the highest possible value for the KD per type throughout the data. The optimal value is an average of all the collected KD's from all pages from our collected data.

Job 2.2.4: Priority assignment using KD



This priority assignment algorithm has a pyramid-like structure as shown above. The keyword density (KD) for each page, in the majority amount of cases, will fall within the lower and upper bound values for the KD. The KD value will fall on either slope of the pyramid. The closer the KD is to the optimal value, the higher the priority boost. This is done by obtaining the absolute difference between the KD of the current page and the optimal value. This value is then put in an equation where priority= k / absolute difference, where k is a constant which keeps the priority in a suitable range.

Job 2.2.5: Spam Filtering

When using the keyword density as a factor for our page ranking system we found that certain spam pages with a large keyword frequency would end up in our final top ten results by acquiring high priority. We implemented a spam filter to filter out any pages such as these. This was done in the spamFilter() method which takes in the KD of the current page along with the upper and lower bounds of the search term type. It then checks if the current KD is greater than the upper bound and if it is, it marks the page as possible spam and decreases its priority. However, if the KD is less than the lower bound it is less likely to be a spam page but would have lower relevance so the priority is decreased slightly.

Page Rank Methods:

An explanation of our methods for the Page Rank algorithm:

The pageRank functionality utilises a range of methods we created to allow multiple priority ranking algorithms to work together to produce a more accurate range of results. The following is a brief explanation of what the methods do.

The 'map()' function will get the wiki markup format of the page, split the page into an array and remove white spaces within the page. This is also where the methods are called for formatting the searchWord to a writable format for the mapper ('formatSearchTerm()'), obtaining priority based on page elements 'ObtainPriorityForElements()' and obtaining priority based on the word being searched for 'ObtainKdPriority()'.

'formatSearchTerm()' will convert the string being searched to lower case and remove any white space before and after the string.

'ObtainPriorityForElements()' will get the priority of a page by getting the mark-up of the Wikipedia page, work out the number of contributions and add this to the total priority of an article. The method also finds whether the title of the Wikipedia page is the same as the search term, if this is the case then a significant amount of priority will be added to indicate its relevance. Finally, this method also works out the number of times this word occurs and adds the number of appearances to the priority along with the number of times the word appears within bold and italic fonts, in subheadings and in links/references. Separate methods have been created to find each of these.

- CountOccurances(substring, string): Simply returns the integer value indicating how many times a substring occurs in a string and is used in multiple circumstances including counting the number of occurrences of the search word term in the page.

- BoldAndItalicPriority(): Returns an integer which represents the priority of the appearances of the search words in a bold and/or italic font and add this number to the total priority
- headerPriority(): Returns an integer which represents the priority of the appearances of the search words within different types of headers e.g. h2,h3,etc. This then adds to priority depending on the size of the header where h2 is largest and h6 is smallest. This number is then added to the total priority of the Wikipedia page.
- LinkPriority(): Returns an integer which represents the priority of the appearances of the search words in any links, references whether external or internal as well as image captions. Depending on the type of link, a priority is calculated and returned e.g. Internal links are given higher priority than external links due to internal links being more reliable.
- arrayIndexOfString(): This method returns an array list that gives the string index position of every occurrence of the character specified. E.g. "[", "=", etc.
- checkRepeats(): This method returns the number of times a character is repeated in a row. It takes the starting position of a character and will continuously check if the character in the next position is the same, until it finally isn't. When first running the method, it checks if the position before also contains the same character and if so, returns 0.

'ObtainKdPriority()' will get the priority of a page by calculating the KD(Keyword density) of the search term(number of time the search term appears divided by the total number of words). The closer the KD is to the optimum value, the higher the priority will be. Several methods work alongside this method to obtain accurate priority values.

These methods will analyse the search term in order to assign the range which best applies to the search term:

- checkMultiTerm(): Checks for whether a search term has multiple words.
- checkSingleTerm(): Checks for whether a search term is a single word.
- checkCharacters(): Checks for whether the search term is a lone character which is a consonant.
- checkVowel(): Checks for whether the search term is a lone vowel.

Additional methods work calculating priorities using KD values:

- calculatePriority(): This will calculate a priority depending on how close the KD is to the optimal value. This value is added to the final priority of the current Wikipedia page.
- calculateKeywordDensity(): Calculates the keyword density for all pages regardless of type
- countWordsInStringAndKillWhitespace(): Takes the search term and counts number of words regardless of format and even if the search term has multiple whitespace in random places.

The following methods are used to filter our spam pages with little relevance for the entered search term but attempt to get into to the final top 10 results:

- spamFilter():Where KD is greater than the upper bound or lower than the lower bound, this would indicate that the page is likely to be of little relevance to what the user is searching for.
- spamCapture(page): Uses certain wiki text to detect spam/ irrelevant pages and flags them to have no priority boost.

Job 3: Top 10 results

This map/reduce job will take in key and value pairs as the input with the key being the title of the page and the value being the priority number assigned to it according to the PageRank algorithm that was implemented in the second job. It will then sort these in descending order with the page with the largest priority at the top and only display the top 10 results with the most relevant pages corresponding to the search term. The mapper is primarily used to split each line into 2 elements of the array fields and then checks to see if only 2 elements are present in the array. The mapper then writes these 2 fields as key value pairs to the reducer. The 2nd element of the fields array is trimmed to make sure no white space is passed when converting the string into an integer.

The reducer begins with the reduce method where all the values are taken from the mapper and every value is assigned to the integer, "sum". As there is only one instance of the reducer in each run it just needs to equate to it and doesn't need to be cumulatively added on, these values are then added to the hashMap. We then move onto the next method which is the "clean-up" method and this is where we call the 'sortByValues()' method which returns a sorted HashMap and allows for us to display the top ten results. The method to sort is called 'sortByValues()' and the unsorted hashMap [5] is used as the hashMap that it then sorts to output another hashMap which is sorted in descending order. The sort method is of type Map<k,v> so that hashMaps with key

value pairs can be input and returned from it. In the first line, the 'entrySet()' method is called and the returned set is then put into a linkedList and this makes sure that there are no duplicates and also makes sure that at most there is only one null element. The sort method is used from the collections library (this implements a modified merge sort algorithm [6]) and a custom comparator is used. This is because the default method compares by calling the 'compareTo()' method on the first object and the object being compared is taken as a parameter so for example, 'o1.compareTo(o2)' where o1 is Object1 and o2 is Object2. This would work for arrays and lists but as we are using a map<k,v>, a getValue() method needs to be used to get the integer so we would use the following ," o2.getValue().compareTo(o1.getValue())".

A new linkedHashMap is created and calls it's sortedTitlePriorityMap. As it is a linkedHashMap, unlike a normal HashMap, the key value pairs stay together after the sorting and shuffling. Then for every entry in the set 'entries' each key value pair is placed into the sortedTitlePriorityMap. This is then returned to the cleanup method where it had been called. In the cleanup() method the 10 entries showing the pages with the highest priorities are written to the output.

4: Results and Analysis

Obtained Results Table: Experiments have been carried out on the full dataset, including a range of search terms below:

"Aldous"		**"Pepsi"**	
Wiki Output	Our Output	Wiki Output	Our Output
1. Aldous Huxley 2. Aldous Harding 3. Aldous 4. William Aldous 5. Richard Aldous 6. Russel Brand 7. Get Him To the Greek 8. Peter Aldous 9. David Aldous 10. Alan Aldous	1. Aldous 2. Peter Aldous 3. Aldous Huxley 4. Stan Aldous 5. David Aldous 6. William Aldous 7. Mick Aldous 8. Lucette Aldous 9. W. Lens Aldous 10. David Aldous (disambiguation)	1.Pepsi 2.PepsiCo 3. List of Pepsi variations 4.Crystal Pepsi 5. Pepsi Max 6.Pepsi Globe 7. Pepsi Center 8. Pepsi & Shirlie 9.Pepsi Blue 10. Helen DeMacque	1. Pepsi 2. PEPSI 3. Pepsi Globe 4. Pepsi Blue 5. Pepsi Next 6. Pepsi Refresh Project 7. Crystal Pepsi 8. Pepsi Max 9. Pepsi Max (North America) 10. Diet Pepsi
"Salou"		**"Marvel"**	
Wiki Output	Our Output	Wiki Output	Our Output
1. Salou 2. Bachirou Salou 3. Salou Djibo 4. Salou Ibrahim 5. Salou railway station 6. Louis Salou 7. Tadjou Salou 8. Sant Andreu Salou 9. Salou Open Costa Daurada	1. Salou 2. Barceló (disambiguation) 3. Salou Open 4. Salou Djibo 5. Salou Ibrahim 6. Bachirou Salou 7. Salou (disambiguation) 8. Tadjou Salou 9. Sant Andreu Salou 10. Louis Salou	1.Marvel 2.Captain Marvel (Marvel Comics) 3.Marvel Comics 4.Marvel Cinematic Universe 5.Marvel Entertainment 6.Marvel Studios 7.List of Marvel Comics characters 8. Carol Danvers 9.List of television series based on Marvel Comics 10.List of films based on Marvel Comics	1. Marvel 2. MARVEL 3. Official Handbook of the Marvel Universe 4. Marvel Treasury Edition 5. List of Marvel Comics characters: S 6. Marvel Zombies (series) 7. List of Marvel Comics characters: A 8. List of first appearances in Marvel Comics publications 9. List of Marvel Comics publications (A–M) Official Marvel Index

“Harry”		**“Wind in the Willows”**	
Wiki Output	Our Output	Wiki Output	Our Output
1. Harry 2. Harry Potter (film series) 3. Harry Styles 4. Harry Potter 5. XXX (2002 film) 6. Bing Crosby 7. Roy Halladay 8. Rubeus Hagrid 9. Prince Harry 10. Hermione Granger	1. Harry 2. Meharry Medical College School of Dentistry 3. Harry Grayson 4. List of songs recorded by Harry Connick, Jr. 5. Harry Potter (film series) 6. Harry Potter in translation 7. List of supporting Harry Potter characters 8. Blind Harry Harry the Minstrel 9. Harry Fischel Institute (Machon Harry Fischel) for Research in Talmud 10. Harry Gamboa, Jr.	1. The Wind in the Willows 2. The Wind in the Willows (2006 film) 3. The Wind in the Willows (1996 film) 4. The Wind in the Willows (disambiguation) 5. The Wind in the Willows (1987 film) 6. The Wind in the Willows (1983 film) 7. The Wind in the Willows (TV series) 8. Wind in the Willows (1988 film) 9. The Wind in the Willows (1995 film) 10.The Wind in the Willows (musical)	1. Wind in the Willows 2. Wind in the willows 3. Wuthering Heights (fictional location) 4. The Wind in the Willows 5. The Adventures of Ichabod and Mr. Toad 6. The Wind in the Willows (1996 film) 7. Count Duckula 8. The Wind in the Willows (disambiguation) 9. The Wind in the Willows (TV series) 10. Wind in the Willows (1988 film)
“Harry Potter”		**“Crocodile Hunter”**	
Wiki Output	Our Output	Wiki Output	Our Output
1. Harry Potter 2. Harry Potter (film series) 3. Harry Potter and the Deathly Hallows 4. List of Harry Potter characters 5. Harry Potter and the Philosopher's Stone 6. Harry Potter (character) 7. Harry Potter and the Philosopher's Stone (film) 8. Harry Potter and the Cursed Child 9. Harry Potter and the Chamber of Secrets 10. Harry Potter and the Half-Blood Prince	1. Harry Potter 2. Harry potter 3. HARRY POTTER 4. J. K. Rowling 5. Harry Potter (film series) 6. List of supporting Harry Potter characters 7. Magical objects in Harry Potter 8. Lord Voldemort 9. Harry Potter in translation 10. Order of the Phoenix (fiction)	1. The Crocodile Hunter 2. Crocodile Hunter 3. Steve Irwin 4. The Crocodile Hunter: Collision Course 5. My Daddy, the Crocodile Hunter 6. The Crocodile Hunter Diaries 7. Michael "Crocodile" Dundee 8. List of The Crocodile Hunter episodes 9. Crocodile 10. Mugger crocodile	1. Crocodile Hunter 2. Crocodile hunter 3. Crocodile Hunters 4. Steve Irwin 5. Malcolm Douglas (documentary maker) 6. Wiggly Safari 7. Lolong 8. The Crocodile (film) 9. The Crocodile Hunter: Collision Course 10. Rob Bredl
“The Punisher”		**“Bob the Builder”**	
Wiki Output	Our Output	Wiki Output	Our Output
1. Punisher 2. The Punisher (TV series) 3. Punisher (disambiguation) 4. The Punisher (2004 series) 5. The Punisher (2004 film) 6. Punisher in film	1. List of The Punisher title 2. The Punisher in film 3. Mangindusa 4. Belting (beating) 5. List of The Punisher characters 6. The Punisher (2004 film) 7. The Punisher (1993 video	1. Bob the Builder 2. List of Bob the Builder characters 3. List of Bob the Builder episodes 4. Bob the Builder: The Album 5. List of Bob the Builder home video releases 6. 15 KB (351 words) - 22:38, 19	1. Bob the Builder 2. Bob the builder 3. Bob The Builder 4. Bangarang (song) 5. Rob Rackstraw 6. CBeebies 7. List of Bob the Builder

7. Punisher: War Zone 8. List of Punisher supporting characters 9. Punisher Max 10. The Punisher: No Mercy	game) 8. Punisher 9. List of Marvel Comics publications (N–Z) 10. Punisher: War Zone	October 2017 7. Bob the Builder: Festival of Fun 8. Can We Fix It? (category Bob the Builder) 9. Keith Chapman (category Alumni of Norwich University of the Arts) 10. Neil Morrissey (category Alumni of the Guildhall School of Music and Drama)	characters 8. List of Bob the Builder episodes 9. Bob the Builder: Can We Fix It? 10. List of European number-one hits of 20

Evaluation of Results and comparison with Wikipedia search engine:

From our results, we found our search engine provided appropriate pages for every search term, (single terms and multi terms) that were inputted. When compared to the results returned by the Wikipedia search engine [7] we found around half or more of the pages returned are identical to ours. Considering we used the dataset from 2013 and the current search engine for Wikipedia is using all data up to 2017 we concluded our search engine to be very accurate. In some cases, we found more useful pages returned by our search engine than when using the Wikipedia search engine. This resulted from the similarity analysis within our algorithm from page rank which produced related pages using keywords connected to the search word. For example, when searching "Bob the Builder", our 5th result is "Rob Rackstraw" and this man was the voice for the main character from the tv show "Bob the Builder", additionally the 6th result is "CBeebies" and this is the channel this tv show aired on. Similarly, the author "JK Rowling" and character "Lord Voldemort" were returned for the search, "Harry Potter". Further results are shown above. (All results were not included due to page limitations)

5: Additional project Goals:

Project Goal	Status
Implementation of PageRank ranking function [up to 20 marks]	**Complete**
Combination of multiple ranking strategies to compute results [up to 10 marks]	**Complete**
Implementation of state of the art ranking functions [up to 25 marks]	**Complete**
Thorough evaluation, including comparison with results obtained from search engines [up to 10 marks]	**Complete**
Multi-term searches [up to 10 marks]	**Complete**

6: References

[1] Cloud9 - http://grepcode.com/file/repo1.maven.org/maven2/edu.umd/cloud9/2.0.0/edu/umd/cloud9/collection/wikipedia/WikipediaPage.java

[2] Google's ranking features - https://searchengineland.com/8-major-google-ranking-signals-2017-278450

[3 Keyword Density - https://www.hobo-web.co.uk/keyword-density-seo-myth/

[4] Keyword Density Formula - https://en.wikipedia.org/wiki/Keyword_density

[5] HashMap- https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

[6] Merge Sort Algorithm Research - https://www.geeksforgeeks.org/merge-sort/

[7] Wikipedia Search Engine - https://en.wikipedia.org/w/index.php?search=&title=Special%3ASearch&profile=default&fulltext=1

All group members worked together in a fair and appropriate manner for this project (see Group Contributions.txt file)