HW3

**404 brain not found**

[**colab**](https://colab.research.google.com/drive/1TeyT44cEsJmIm4XTSvmia1N69QX6pKg8?usp=sharing) **(view only)**

***שאלה 1:***

מחלקה Relevent:

[1, 3]

מחלקה Non-Relevant:

[3, 5]

המרכז של Relevent = 2 והמרכז של Non-Relevent = 4 (סכום האיברים של המחלקה חלקי גודל המחלקה)

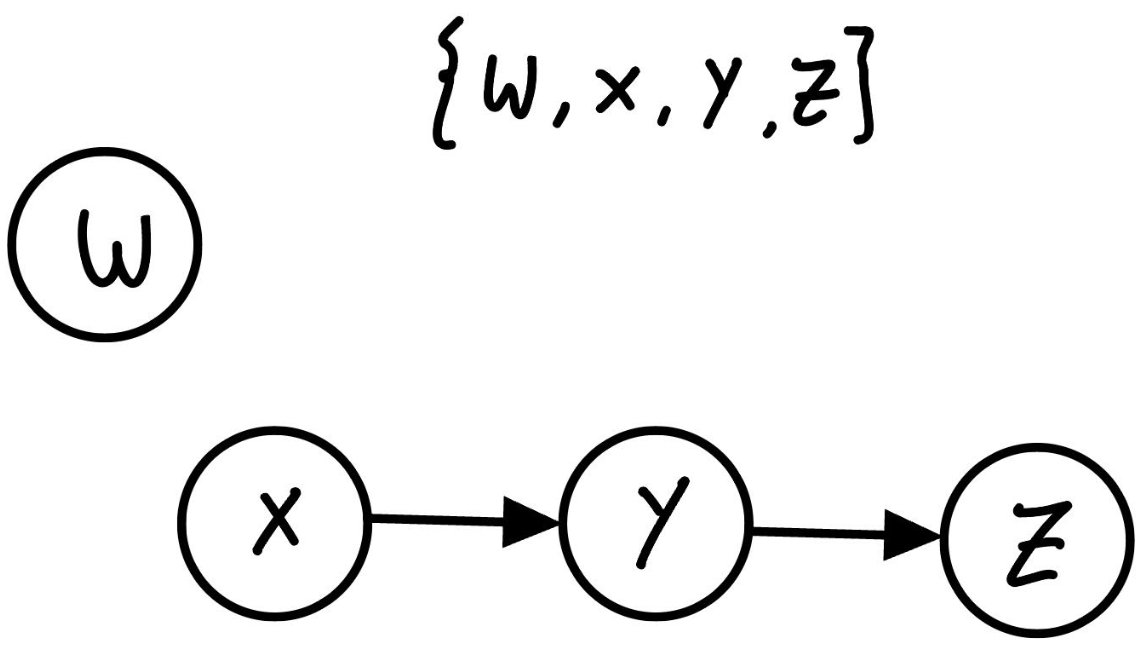
נניח שמגיע משהו חדש X = 2 למי לשייך?

מרחק מ-R הוא 2 ומרחק מ-NR הוא גם 2, אז מה שהאלגוריתם יקבע לא חובה שיהיה נכון, מה שהגיע חדש יכול להיות R ואהלגוריתם יסווג אותו ל-NR או הפוך.

***שאלה 2:***

סעיף א:

We want at least 4 nodes, only 2 edges total, and for each node:



Nodes = {W,X,Y,Z} , Edges = x->y, y->z

Lets calculate each node (in and out degree):

**W:**

in‐degree = 0 (no incoming edges)

out‐degree = 0 (no outgoing edges)

authority(W) = 0, hub(W) = 0

**X:**

in‐degree = 0 (no edges into X)

out‐degree = 1

authority(X) = 0, hub(X) = 1

**Y:**

in‐degree = 1 (from X)

out‐degree = 1 (edge Y->z)

authority(Y) = 1, hub(Y) = 1

**Z:**

in‐degree = 1 (from Y)

out‐degree = 0

authority(Z) = 1, hub(Z) = 0

We conclude that all nodes, **authority = in‐degree** and **hub = out‐degree.**

סעיף ב:

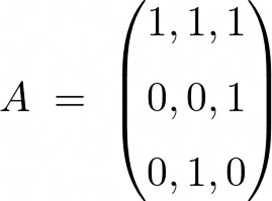
**1. Construct the Adjacency Matrix A**

Label the three pages as {A,B,C}. We form a 3x3 adjacency matrix A, whose rows and columns both list [A,B,C].  
 An entry if page i links to page j, else 0.

Hence the rows are:

* **Row for A** (row1) : → A, B, C → [1, 1, 1]
* **Row for B** (row 2) : → C only [0, 0, 1]
* **Row for C** (row 3): → B only [0, 1, 0]

So:

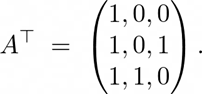


2. Hubs and Authorities via and

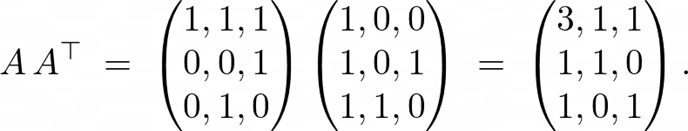
2.1 Hub Scores: and

So we can substitute and get:

Thus h (hub vector) is an **eigenvector** of corresponding to its largest eigenvalue.



then multiply :



We want the principal eigenvector h of this 3x3 matrix.

2.1.1 Solve for the Eigenvector:

Let then :

,

,

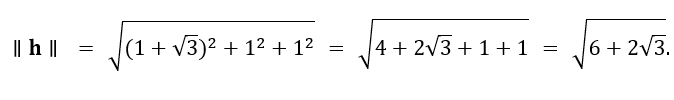
.

We notice that y = z. From the second equation above and from the third so :

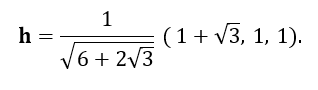
( is used to equal up to scaling).

2.1.2 Normalize

Let's do it with L2



So, we get:

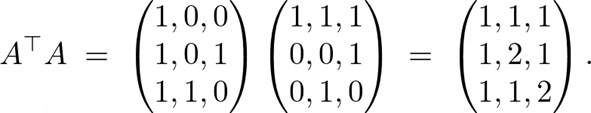


Finally:



2.2 Authority Scores )a.

We compute :



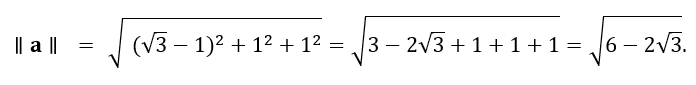
Let , then:

And we will have:

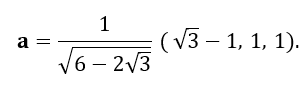
Again, we can see that in the principal solution.

Similar process will get us:

And normalizing using L2 will give us:



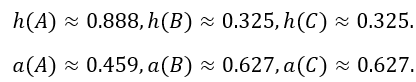
So:



Finally:



Final Scores:



Results:

has the largest **hub** score because it points to everyone (including itself).

B and C share the highest **authority** because each is pointed to by A and they point to each other.

I would suggest to pick A If we want a good hub, If we want a good authority lets pick B or C.

***שאלה 3:***

**First question about interesting technologies :**

1. Urban Dictionary API:

We fetch data from the Urban Dictionary API, and then parse the JSON results in order to add them into our local cache.

1. JSON Caching:

We store fetched definitions (in urban\_data.json) to avoid repeated calls to the API and speed up subsequent searches.

1. splitting queries by “or,”:

We added a feature where the user can get multiple definitions in one query by adding a “or” between each query, as an example : harry potter or john snow.

we will search for “harry potte” and then search for “john snow” if the user did not add a “or” we will take the hole query.

1. Custom Ranking Logic:

Instead of a traditional TF-IDF or other ranking systems, we implemented a custom ranking formula :

1. Search for New Slangs:

We can use an API to return 7 random definitions, then use the API 10 times to get a total of 70 definitions, and then we can return the newest terms with the highest score to the user.

1. Search using words.txt:

if we want to search in bulk we create a file called words.txt then in each line we write a query, after doing that the code will go over each line and get definitions for each query in that file.

**Second question : How Long Did Your Queries Take?**

Measured Time(25 seconds) to process all queries in words.txt.

what does it depend on : **Number of terms**: If words.txt has many lines or sub-queries separated by “or”, each one must be fetched from Urban Dictionary. the time also depends on the network connection because each request to the Urban Dictionary API depends on network speed and the API response time.finally it also depends on caching: Once a term is cached in urban\_data.json, the next time someone searches the same term its instant. If no data is cached, more calls happen to the API.

**Can It Be Improved?**

Yes, with multithreading in python we can parallelize the requests to the API.

Also a faster compiled language like C++ might improve text processing delays, though for network tasks, concurrency often helps more than switching languages.

and finally reducing or batching the number of API calls also helps (for instance, if the API supported multi-term queries in one request) but we need special request from the site because currently it is not public.

I want to take a moment to describe how search engine for urban dictionary works :

**Reading Input (word.txt)**:

* read each line from words.txt. Each line may contain multiple sub-queries delimited by " or ".
* If a line has no "or", it’s just one query. If it has "or", we split it into multiple sub-queries.

**Cleaning & Normalizing Queries**:

* We strip leading/trailing whitespace, remove double spaces.

**Caching / Lookups** (urban\_data.json):

* For each sub-query (example : "john snow", "harry potter"), we check if it exists **in the JSON cache**.
* If not cached, we fetch from the **Urban Dictionary API**, process the definitions (calculating the custom score, censoring any bad words), and store them in the JSON file for future use.

**Selecting Final Definitions**:

* For each sub-query, we identify the top 10 highest-ranked definitions out of all the definitions that we got from the API based on our own formula

**User Prompt / Interactive Query**:

* After processing words.txt, the program **prompts** the user to type a new query.
* The same logic applies: split by "or", use cache if available, otherwise fetch from the API, then display oldest/newest/highest-scoring definitions.

**Third question : Do the Released Pages Have Hubs? Authorities?**

### **Are There Hubs / Authorities Here?**

* Yes, in principle, each definition text can contain multiple [linked\_term] references to other pages.
* If a particular definition references *many* other terms, that definition can be considered a potential hub.
* A definition that *many other pages* link to (mention with [square bracket] references) could be considered an authority.

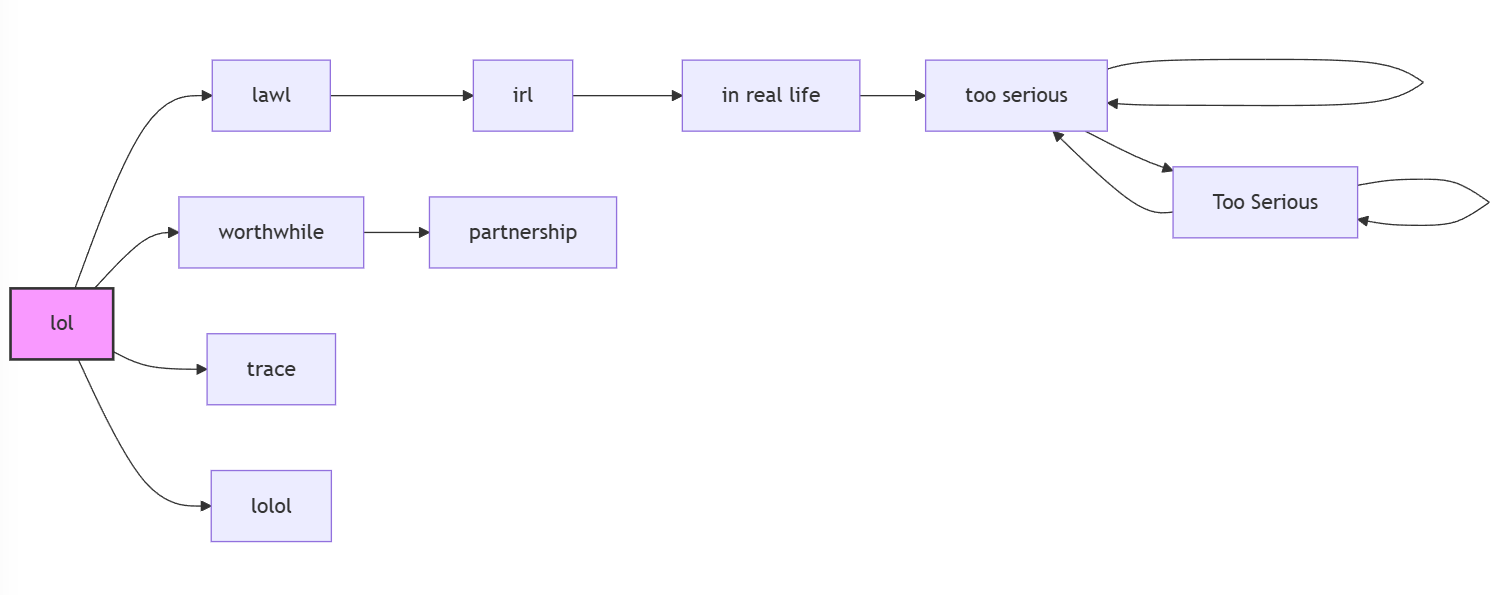
**Example**:

* If “harry potter (oldest definition)” references [wizard], [magic], [hogwarts], that's a potential hub.
* If *multiple definitions* from “wizard,” “magic,” “hogwarts,” “voldemort” , link back to the “harry potter” page, then “harry potter” might be a strong authority.

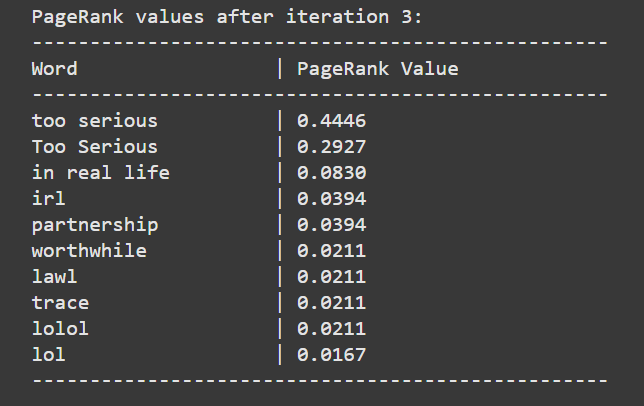
fourth Question: Select 10 Pages That the Crawler Returned & Calculate PageRank

an important note is that we did not use the definitions that were returned for the PageRank calculation because it is impossible to do and does not fit the requirement, instead, we used the word that the user has requested to search in order to expand into other words that are linked inside the description of the searched word

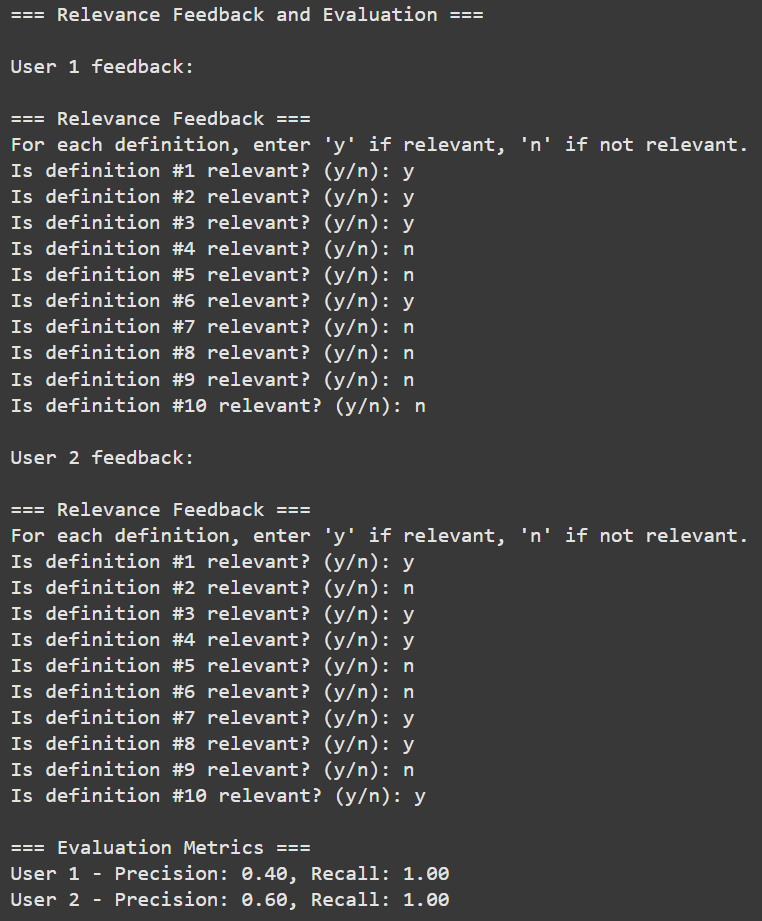
this is the structure of the words:



the reason why the structure looks like that is because we took a word from the user input which was “lol” and expanded it into different definitions that is why it seems like a straight line, and we limited the code to search for only 10 words that are connected to each other, and here is the PageRank after 3 iterations:



Fifth Question : Show Two Different Users the Ranking & Get Feedback on Relevance



we can't calculate Recall because we use an API that returns all relevant definitions, so we return all definitions but we only show the top 10 definitions using our scoring system, even though we show the top 10 we have essentially returned all the definitions which is why it is 100%.