Loay Sholy: 214682189 loaysh@post.bgu.ac.il

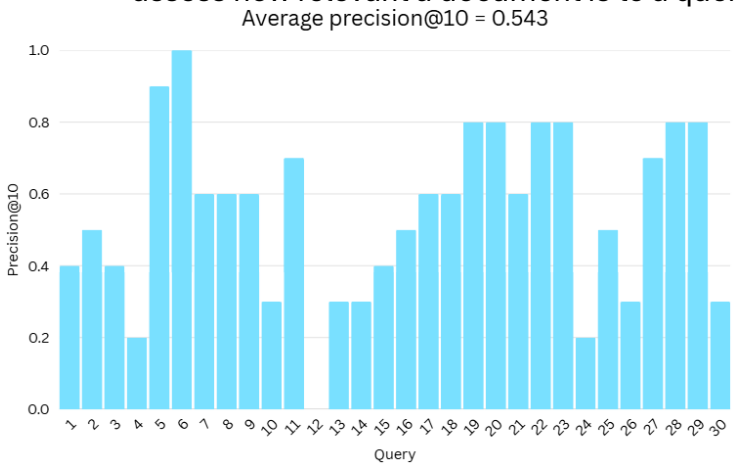
Roza Yusupov: 212997845 rozayus@post.bgu.ac.il

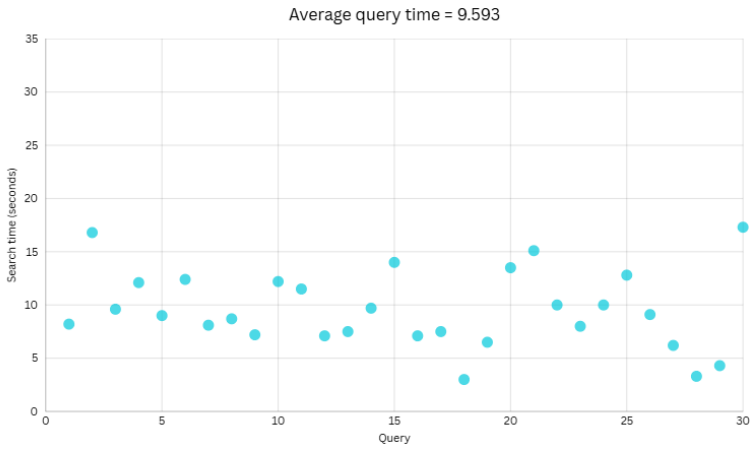
Mohammed Shihada: 325466951 Shihadam@bgu.ac.il

before we show our experiments we want to clarify that all the testing was done in our local python environment, the reason behind that is because before starting the project we knew we are going to rebuild the body inverted index and not use the index we built in practical assignment 3, we also knew we are going to build a title and anchor index as well. In other words we were afraid that we will run out of credits so we decided to take our work to local python environment where testing will be free.

Experiment no.1: Body index.

in this experiment we built our own version of the body inverted index, unlike the index built in assignment 3, this one contained 3 more attributes needed for the BM25 metric which we will address soon. The attributes added are: total corpus terms, N, doc len. To keep things brief we will not explain the meaning behind each attribute check the documentation in the "Inverted Index" class to understand the purpose of each attribute. The body index uses the BM25 metric used in practical assignment 4 (k=1.5,b=0.75) to assess how relevant a document is to a query. Here are the results we got:

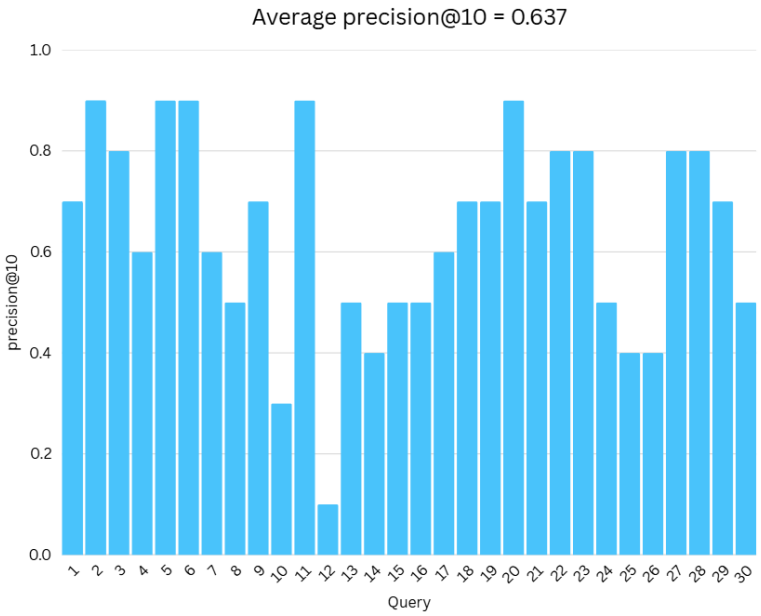


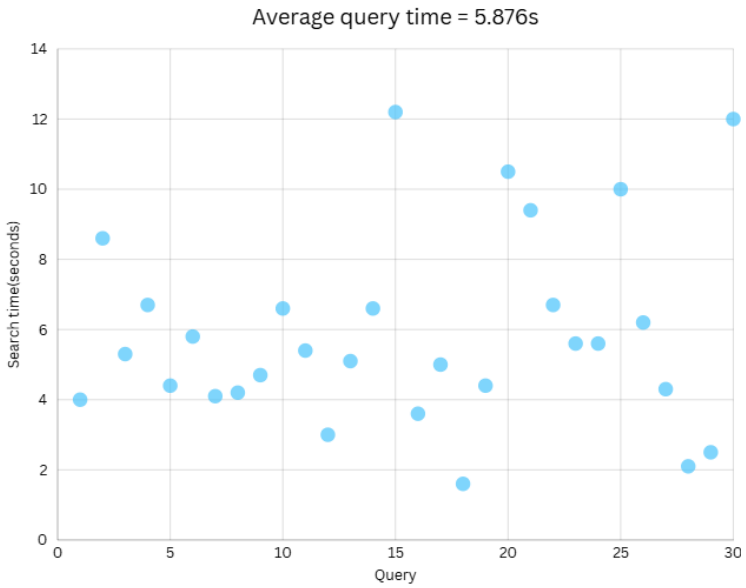


After we saw the results we were little surprised that our most default search engine did this well, altough we still don't meet the minimum requirements we are very close and it made us both very optimistic and motivated to continue. We noitced though that an average of 10 seconds for a query is bad and since we are only going to be adding more calculations we needed a solution.

Experiement no.2: body index, title index and threading.

To solve the average query time problem we decided to add threading, for every term in the query we create 2 threads, one that calculates the term score in the body index and one that calculates the the term score in the title index. In the end we combine the body and title indices results where the final result is simply 75% body index score and 25% title index score (weight was chosen after mulitple tries). Here are the results:





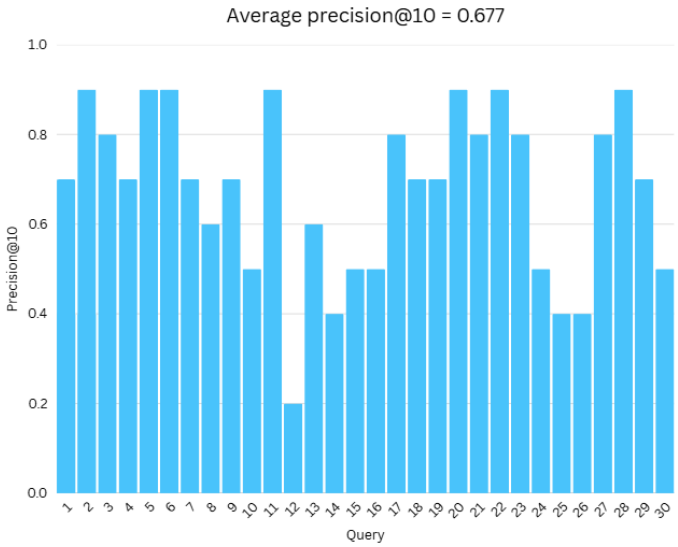
there was a significant improvement, alomost every single query had improved and now we barely meet the minimum requriments so we needed to add more stuff for better precision.

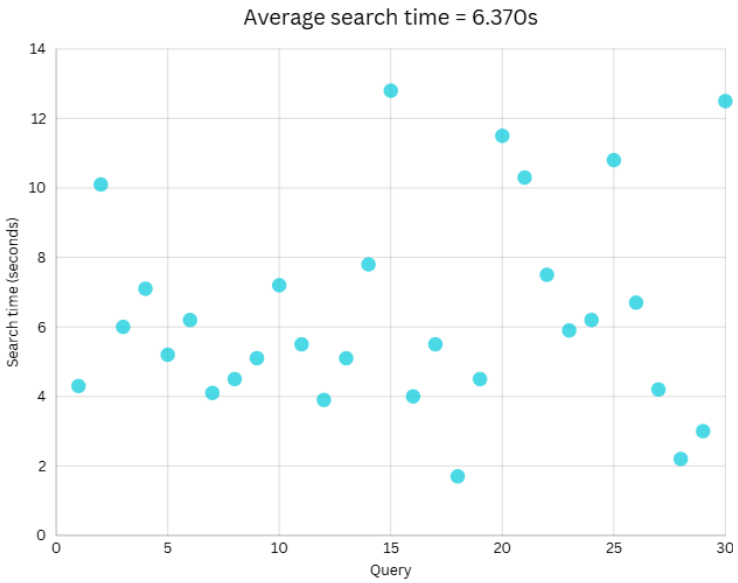
Experiment no.3: body index, title index, page rank and threading.

since in practial assignment 3 we calculated each document rank in the corpus we decided to use that data to increase our precision, we loaded the page rank file from the bucket and displayed some random documents score. The scores were much higher than the scores provided by the BM25s, for instance a document X could have a body BM score of 30 but a page rank score of 2000, this imbalance could lead to page rank dominance in the final scoring function which could ruin our accuracy. To solve this problem we decided to normalize the page rank values using this formula: $\text{Log}_{10}(\text{page\_rank} + 1) * 0.8$ (tuned).

to save time we precalculated the each document page rank score

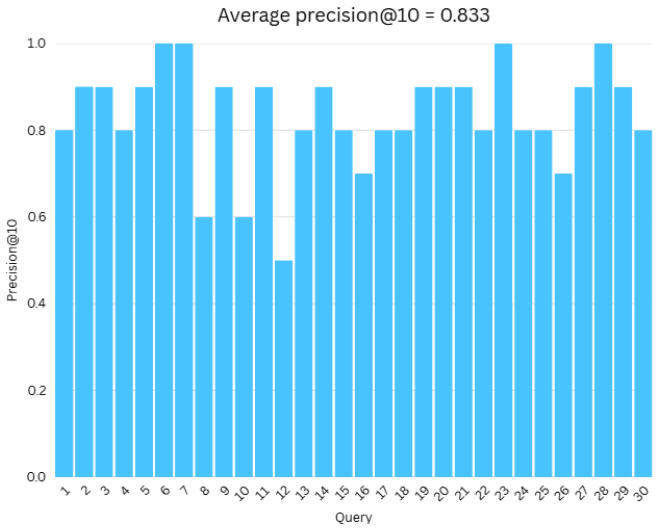
and saved the data in a pkl file then into our bucket for later use. Here are the results:

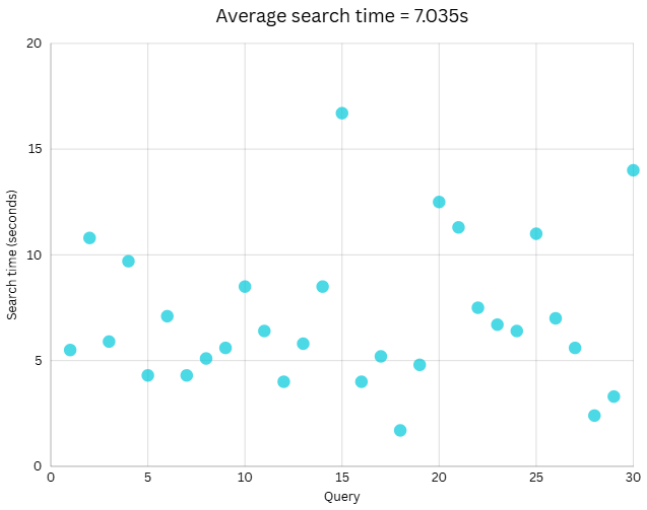




In the first glace improvement might look little but the page rank did help improve query 12 which is our worst performing query plus slighly increased some low perofiming queries.

Experiment no.4: body index, title index, page rank and page views and threading.
In pracitcal assimgent 1 we worked on page views and there there was an option to download pkl file that contains the page views data for every document in the corpus. We loaded the file to our bucket then similariy to page rank we downloaded the file and took a look on the data. Just as page rank we needed to normalize the data to prevent page views score dominance so we used this formula: $Log_{10}(page\_views + 1) * 1.4$ (tuned).
To save calculations time we precalculated the data and saved them in a pkl file then uploaded it to our bucket. We also tuned the body BM25 metric by setting the k = 0.7 instead of the default value 1.5 and tunned the title BM25 metric by setting b = 0.7 instead of the defult value 0.75. Here are the results:





Page views alone improved the average precision up to 0.760 and tunning both “k” and “b” helped us get up to 0.833 average precision and this was our best model.

A query that did really good for us is: "Ancient Egypt pyramids pharaohs", our engine scored a full mark on this query and the reason behind it is the topic of the query it self. Egypy is known for its pyramids which are also related to pharaohs, it is diffuclt to find a document title/body that contains such words in it while talking about irrelavent topic so almost every document we found was related.

A query that did bad is: “Steam locomotive transportation history", the reason why our engine found it hard to find relavent documents was mainly from the word “history” in the query. that word exists in our stop words meaning our indices do not contain that word. we extracted top 10 relavent documents for that query from the test set and we found almost every document title contained the word “history” so we concluded in order to fix this problem we need to try to rebuild the title and body index from scratch which we end up doing. In failed experiements section this will be addressed again.

Failed experiments: we had lot of failed experiments behind the scenese such as the anchor index, we had built an anchor index and also uploaded it in our bucket twice but the two indices failed to bring value to our results and we eventually gave up on them. The reason the anchor index did not work is because maybe the way we built it was not correct so the data in it was not reliable, unlike title and body index building the anchor index is slighly different because it requries unpacking a nested array of structs for every document which was messy to work with for us.
another failed experiment was the “revised title index”. After discovering why query number 12 was doing bad (the missing history term from the index) we deicded to rebuild the title index but it would include the term “history”. We tought this will fix this issue and even provide more accurate data for the rest of the queries but it ended up failing too. I could not find a reasonble reason behind why it was messier so we gave up on it as well.

Link to github: https://github.com/loay-arch/Search_Engine
Link to public bucket: https://storage.googleapis.com/214682189/