Data Structure

Team: 10

Team members: 111306001 資管二 林美婷 資管二 111306049徐華均

111306055 資管二 呂家華

Topic: Trip searching website and (Android) APP:

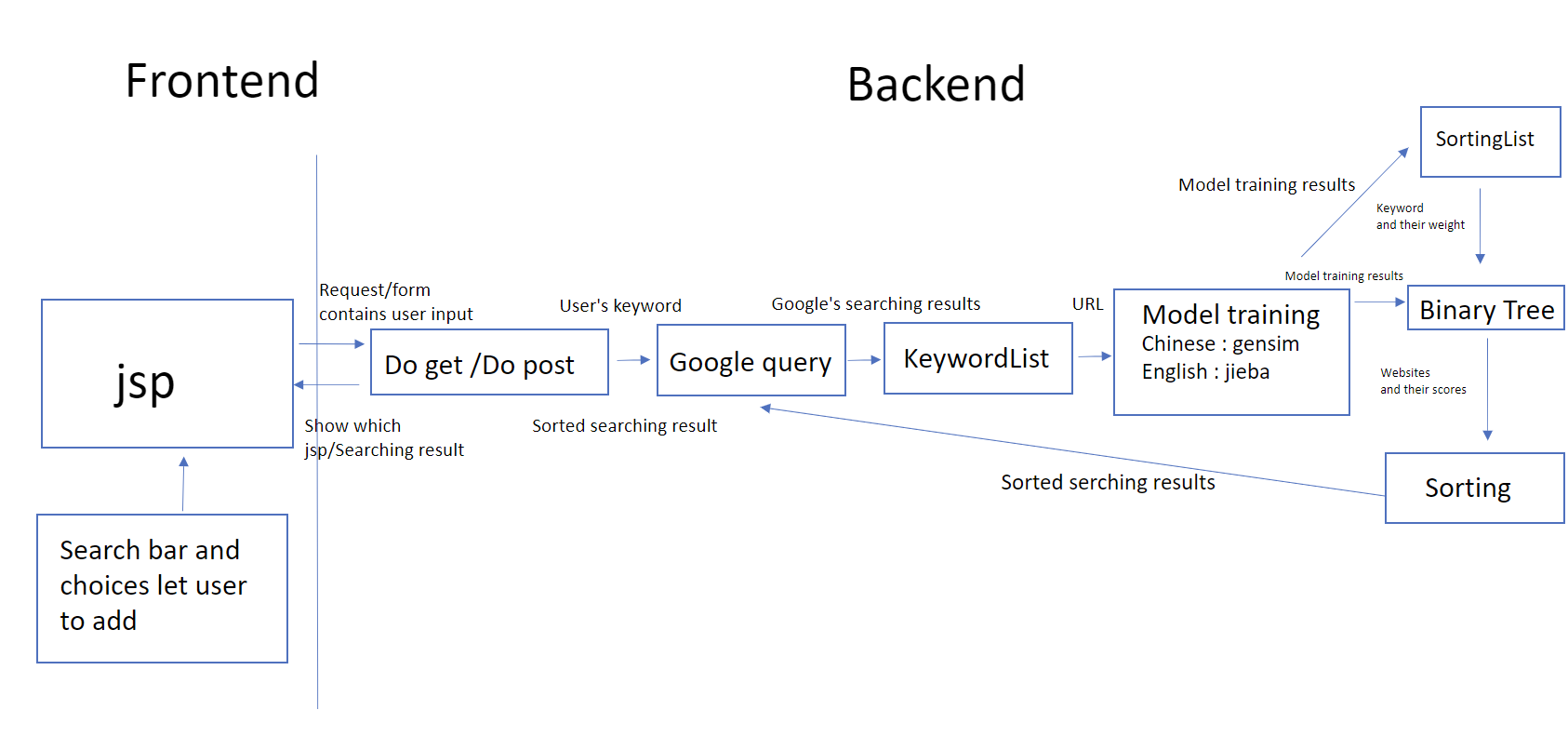
1. Introduction

If we simply enter a specific location into the Google search engine, we may encounter the following issues:

1. Too much ad: The results might be flooded with advertisements.
2. Inaccurate webpage results: Users might receive imprecise information due to various reasons such as incomplete planning or vague search queries.
3. If someone wishes to search for specific information about that location, such as recommended itineraries or visa application procedures, Google might not be able to accurately filter the desired results.

For these reasons, we aim to create a dedicated search engine tailored specifically for international travel purposes.

1. Search tricks

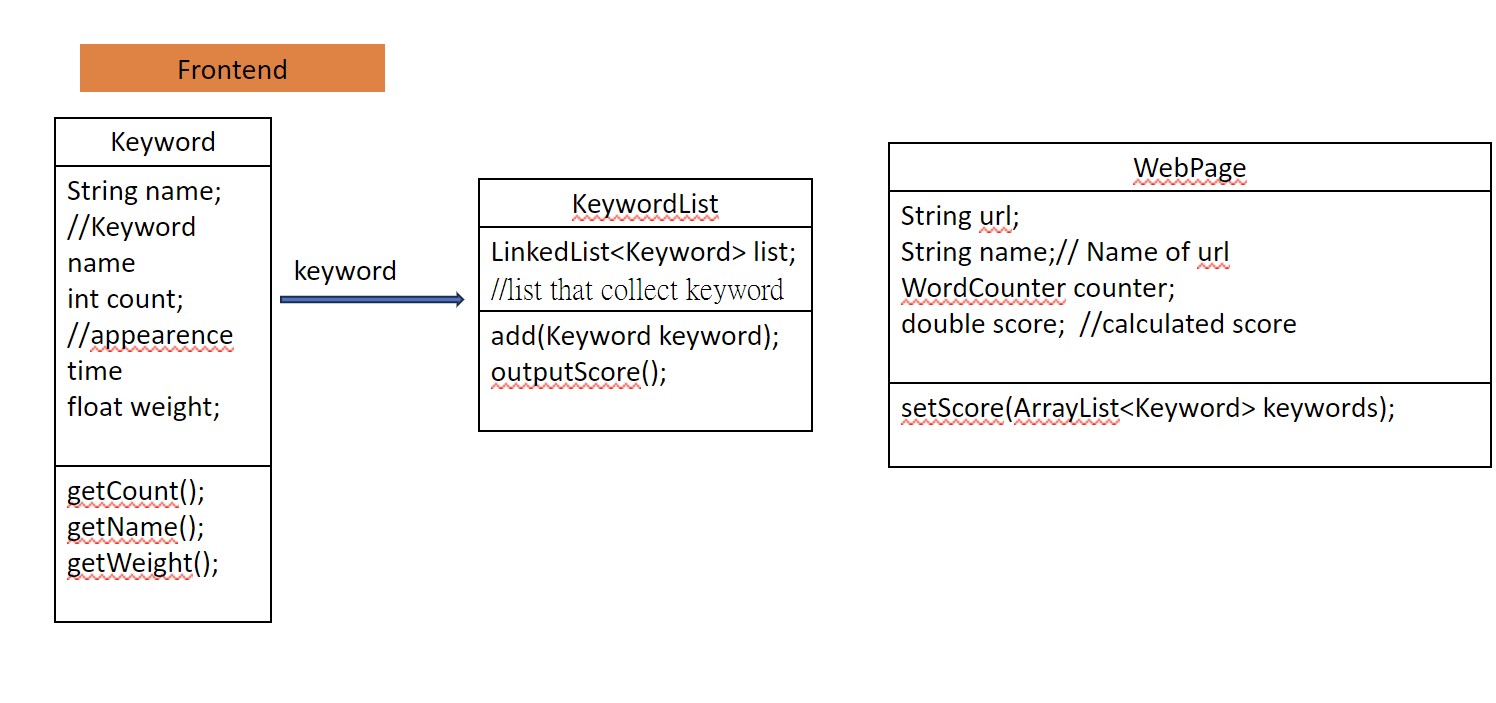


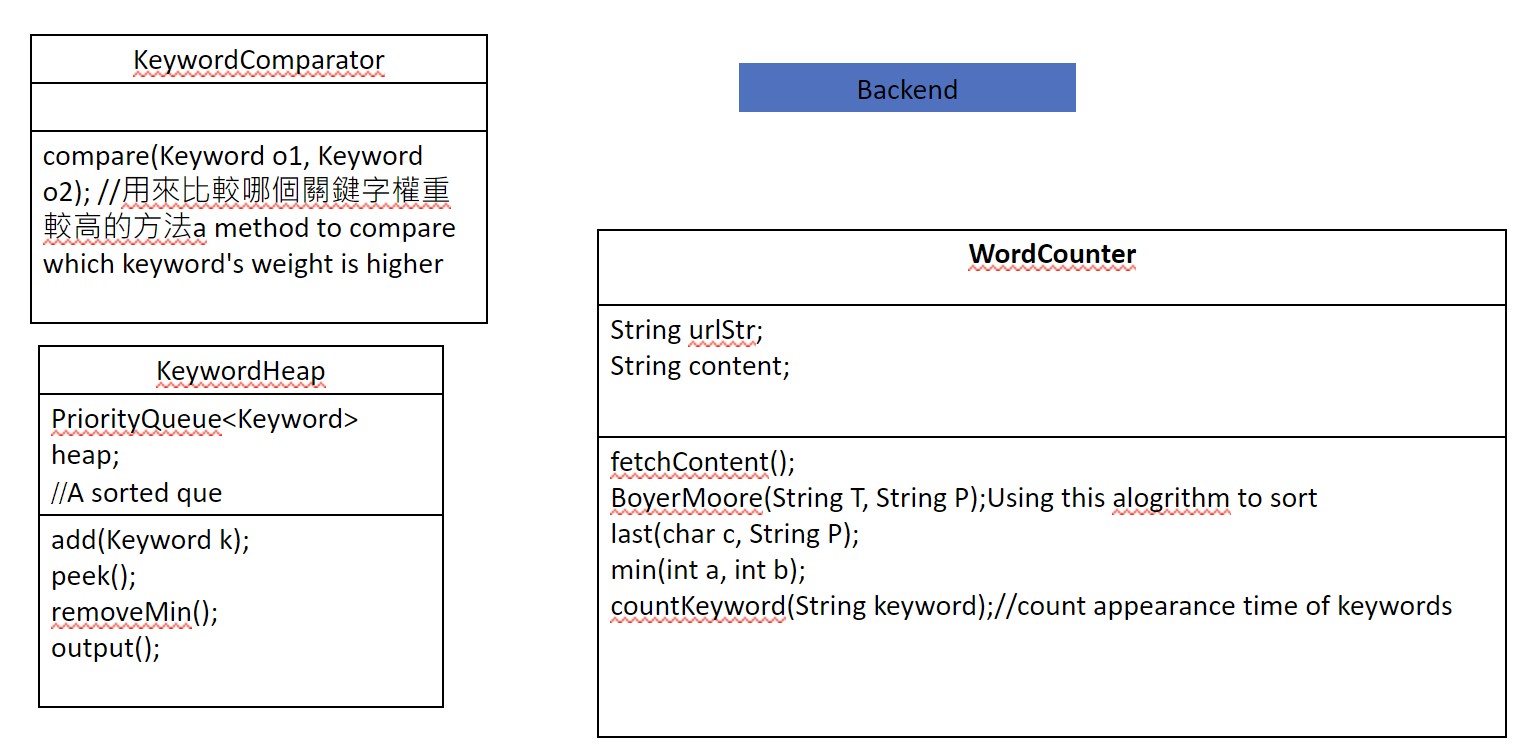
|  |  |
| --- | --- |
| Search tricks | Exmaple |
| User will input journey they want to search on Web page or phone app  \* There will be a dropdown menu for users to select either "All," "Travel Recommendations," or "Flights" to refine their search results. | The user inputs "Europe" on their mobile device and selects the category "Trip." |
| The user's entered keywords and category are input into the Google search engine, generating numerous websites with relevant information. | The user inputs "Europe" on their mobile device and selects the category "Trip." |
| Perform text analysis on the website content to identify additional related keywords (sub-keywords) associated with the user's input. These could aid in more accurately refining the search results. | Analyze the text of the resulting websites to identify similar keywords and their respective similarities. |
| We'll score and filter these sub-keywords to build a scoring tree, using it to assess whether the webpage meets the user's needs. | Further sort the search results based on the most important sub-keywords obtained from the previous step. |
| After constructing the tree, we'll have the final score for each webpage. We'll then sort these scores in descending order and transmit them to the frontend for display to the user. | Once sorted, present the results (websites) to the user in descending order. |

1. Score formulation

Total score = Number of occurrences of a keyword \* Weight of that keyword. (Weights is based on the geographical location).

1. System design & Class diagrams





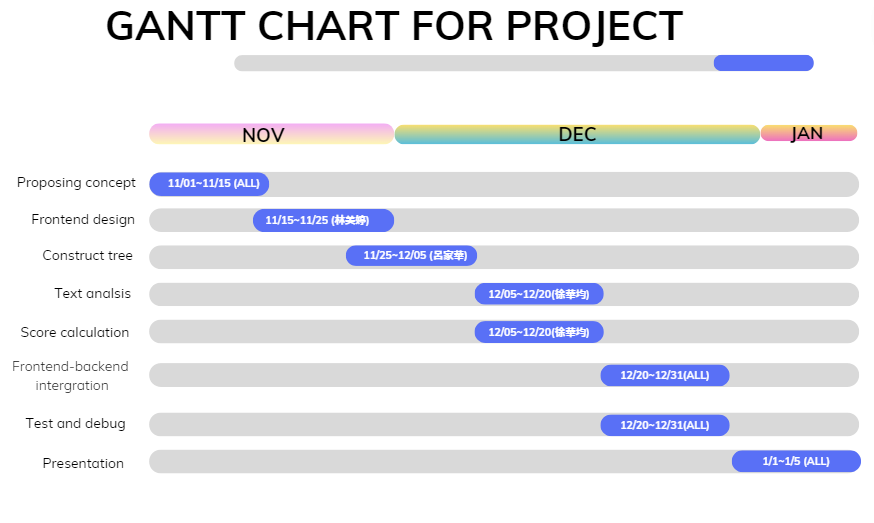
(The class and method would be adjusted before deadline)

1. Frontend: We plan to utilize a cross-platform framework to develop the UI, enabling deployment on both mobile and desktop browsers simultaneously.
2. Submit the keywords to Google search. (Reference: HW4) F
3. Create the scoring tree. (Reference: HW6)
4. For text analysis, we'll employ Word2Vec technology to convert text into vectors and calculate the similarity between two keywords using a pre-trained model in Python. (Reference: HW5)
5. Transmit the computational results back to a Java program for further scoring computations. (Reference: HW3)
6. Once we've calculated the scores for each webpage, we'll sort the webpages based on scores in descending order and then return them to the frontend.
7. Frontend:
   1. Website:

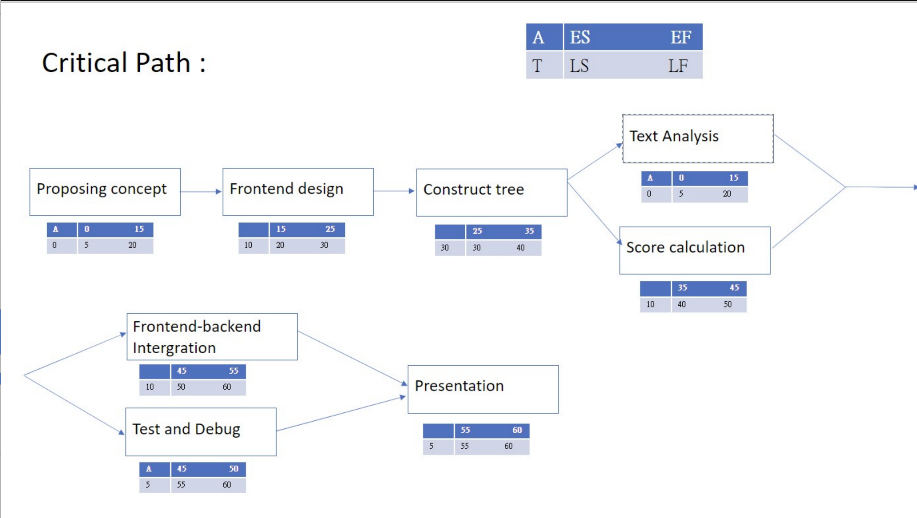
(Reference: HW11)

* 1. App:

1. Schedule

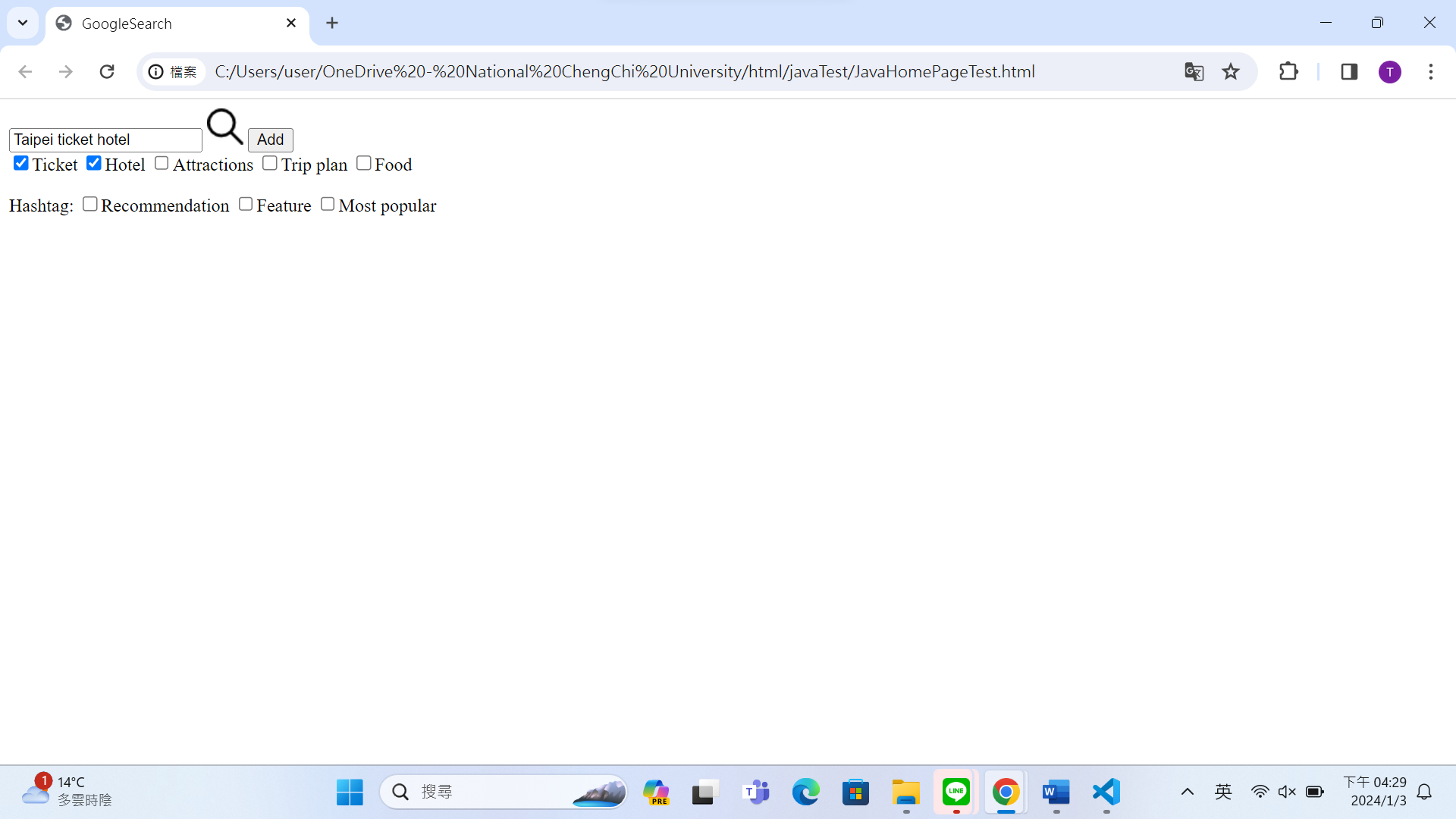


1. Critical path analysis



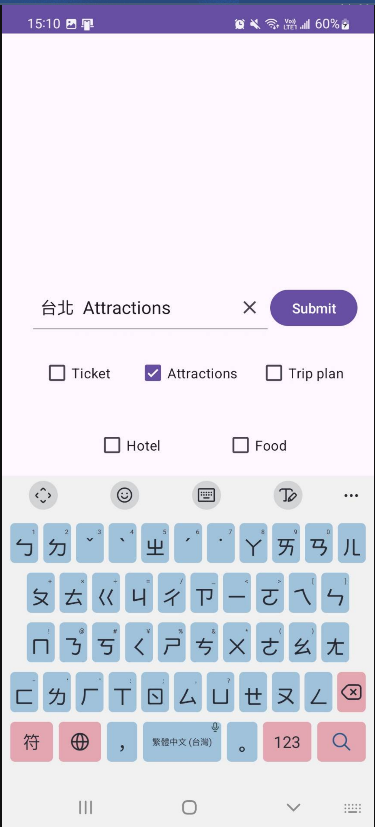
1. Challenges
   1. Machine learning for finding similar keywords.
   2. Matching the different language. (Including: Java, Python, Html and JavaScript)
   3. Deal with the problems between different
2. Work division
   1. Frontend (Web and App) Design, project writing and endtend - 林美婷
   2. Some endtend and project - 呂家華
   3. Word2Vec Keyword Related Queries and endtend - 徐華均
3. Frontend design:
   1. Website:

Based on HW11.



1. Input the place as keyword.
2. Check the checkbox needed.
3. Press “Add” button, it would add the chose keyword.
4. Press the loupe image. The page would show the top 10 results. And the search result shows without ad.
5. There are still some problems we will debug before deadline. (The input can only read the first word in eclipse.)
   1. (Android) APP:

We use Android Studio with java.



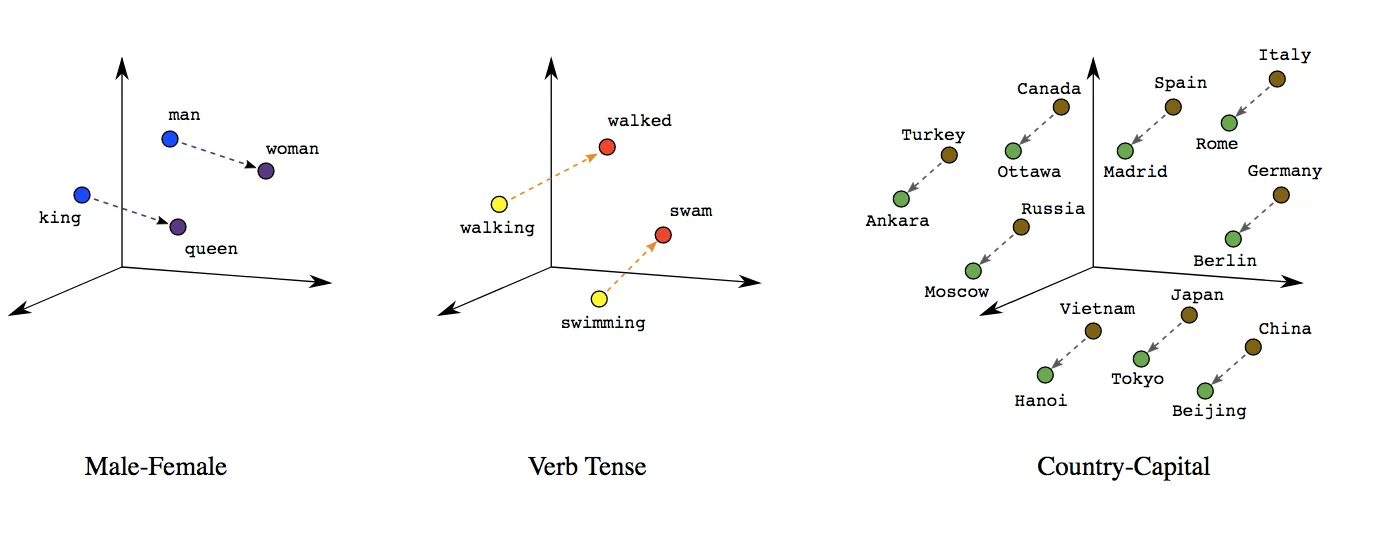
1. Input the place as keyword.
2. Check the checkbox needed, it would add the chose keyword.
3. Press the “Submit” button. The page would show the top 10 results. And the search result shows without ad. (There are still some problems we will debug before deadline.)
4. Search technologis:

We want to search for synonyms of a given keyword. So, we employed several machine learning techniques, and one of them is Word Vector.

1. Word Vector:

Word Vector is a technique used in various text-related NLP (Natural Language Processing) tasks, where discrete words are mapped into a continuous vector space. This enables models to learn using methods like Gradient Descent. We expect Word Vectors, after being trained on extensive text data, to capture relationships between words using vectors. For example: If Word Vectors have been thoroughly learned by the model, we'd expect the vectors for (woman - man) and (queen - king) to be close. Similarly, word vectors representing two semantically similar words should also be close to each other.

1. Word Segmentation:



In addition, when working on English NLP tasks, we typically handle a word as the smallest unit. However, this differs for Chinese. For instance, in the sentence "今天天氣真好" (Today's weather is really nice), the smallest units should be "『今天』『天氣』『真』『好』." Therefore, we need to segment Chinese sentences into individual units before processing them.

Implementation: Word vector: We use the “Gensim” package to handle this, utilizing pre-trained models available online for processing. Word segmentation system: We use the “Jieba” package for this purpose.

1. Relative position of cities:

Our goal is to find travel-related keywords, which can be quite challenging. For instance, when searching for the keyword "Brazil" directly through the Google search engine's "related searches," it often provides unrelated keywords like "Brazil time" or "Brazil English." These are not directly related to travel. Hence, we aim to change this situation by considering not only the queried city but also its neighboring cities, recommending nearby cities.

However, parsing city names is not straightforward due to potential variations in their names. For example, "California" and "Cal" may not seem significantly different to many people, but in programming terms, they are entirely different words, just like "California" and "Taiwan." Therefore, we standardize city names to English for consistent evaluation and comparison.

Implementation: Searching for all countries, cities, and their respective latitude and longitude: Utilizing data available online (Data source: <https://simplemaps.com/data/world-cities>).

We will consider a city's latitude, longitude, and population count for recommendations. Translating from Chinese or other languages to English: Utilizing the "google translation" package, which connects to the Google Translate API for translation purposes.

1. Python communicates with Java:

The approach we're taking is similar to the front-end HTML and back-end Java setup. We facilitate Java to transmit data to Python over the network using the “flask” package.

1. Weight design:

Assuming the keyword represents a region, city, or town, we'll identify the top 10 counties closest to it, all within the same country. Then, using "gensim", we'll measure the word vector distance between these counties and the original keyword, deriving weights. Considering "cosine similarity" as the method for distance measurement in "Gensim", which ranges from [-1, 1], assigning excessively high weights might affect the original keyword computation. Hence, we'll multiply the obtained weights by 0.5 to adjust and use them as weights for the new keyword.