19CSE212

DATA STRUCTURES AND ALGORITHMS

AUTOMATIC TEXT SUMMARIZATION
USING HYBRID DATA STRUCTURES

TEAM:

APOORVA S B : CB.EN.U4CSE21205
M SHIVA: CB.EN.U4CSE21235

Hybrid Data Structures:

Data structures are used to organize and access data efficiently to perform coding tasks with optimized space and time complexity. Each data structure allows the programmer to use a set of functions to alter and access data, and by combining different data structures, we can amalgamate their features to have a hybrid data structure capable of performing the tasks of its parent data structures with greater efficiency and flexibility.

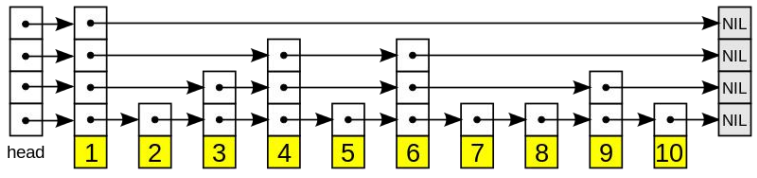
Hybrid data structures are used widely in day-to-day applications that solve real-life problems by efficiently organizing the available data in proper hybrid structures. An example is the use of B+ trees. B+ trees combine the characteristics of a binary search tree with that of a linked list. This allows efficient indexing and retrieval of a large amount of data.

A skip list is another hybrid data structure that combines linked lists with binary search trees. This allows for insertion, deletion, and searching with an average time complexity of O(log n).

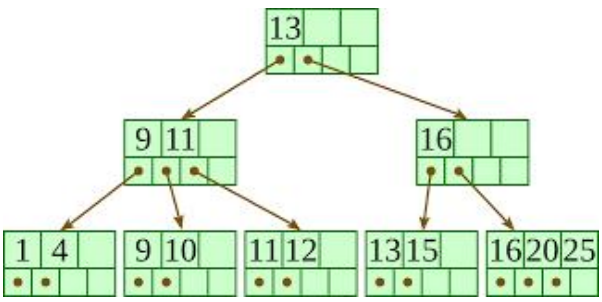
While both are derived from the same data structures, they have contrasting appearances and they are used for different purposes. The B+ tree is a balanced tree, where each node houses a linked list, whereas a skip list consists of a list, where each node has a forward pointer.

This is an example of hybrid data structures, which can be implemented and used for various tasks, based on the requirements.

Skip List



B+ tree



Text Summarization:

The project aims to develop a tool that can perform an extractive text summarization tool that allows the user to minimize the sentence count by choosing only the important sentences from the text corpus.

To achieve this goal we must quantify the importance of the sentences in the corpus. This can be achieved by using various algorithms, the ones we are implementing here are TF-IDF and PageRank.

We will be discussing TF-IDF in-depth.

TF - IDF:

TF-IDF is a metric that uses term frequency and inverse document frequency to highlight the importance of a sentence in a corpus of text. Term frequency ranks the words based on their number of occurrences in the given text file. The formal definition of the term frequency as given by Hans Peter Luhn is summarized as

"*The weight of a term that occurs in a document is simply proportional to the term frequency.*"

Inverse document frequency is used to eliminate the terms that occur very frequently. Hence, this makes sure that the weight of stop words, not necessary for the context is reduced and the important words are chosen instead. Hence, the specificity of the term is quantified as an inverse function to the number of its occurrences.

These two metrics are grouped and the importance of the sentences is ranked on their basis. This allows for an effective summary to be generated.

Chosen Hybrid Data Structure:

The hybrid data structure chosen for this task is a combination of a hash table and a heap. The use of this data structure allows for efficient storage of TF and IDF scores that are calculated and retrieved easily. The TF-IDF scores are stored in a hash table that is implemented using a dictionary of dictionaries and the sentences are stored in a heap to retrieve the sentences with the top TF-IDF values efficiently and quickly.

The use of this hybrid data structure of a hash map and a heap can allow the user application to quickly collect items of a given required frequency or parameter. Hence, the behavior expected by this data structure is a priority queue with updateable priorities.

The definition of the hybrid data structure is as below

```
class TFIDFSummarizer:
    def __init__(self, n):
        self.n = n
        self.tfidf_scores = defaultdict(dict)
        self.sentence_scores = []
```

Time and Space Complexities:

Time complexity:

1. preprocess() function: O(n), where n is the number of words in the sentence.
2. _calculate_tf() function: O(n), where n is the number of words in the sentence.
3. _calculate_idf() function: O(n^2), number of sentences times the average number of words in each sentence.
4. fit() function: O(n^2), number of sentences times the average number of words in each sentence.
5. summarize() function: O(n^2*log(n)), where m is the number of sentences and n is the average number of words in each sentence.
6. printSummary() function: O(n), where n is the number of sentences in the summary.

Space complexity:

1. preprocess() function: O(n), where n is the number of words in the sentence.
2. _calculate_tf() function: O(n), where n is the number of words in the sentence.
3. _calculate_idf() function: O(n^2), where m is the number of sentences and n is the average number of words in each sentence.
4. fit() function: O(n^2), number of sentences times the average number of words in each sentence.
5. summarize() function: O(n), where n is the number of sentences in the summary.
6. Overall, the space complexity is dominated by the dictionary of dictionaries used in the fit() function, which has a space complexity of O(n^2).

Comparison of the hybrid data structure with individual constituent data structures:

The hybrid data structure is a combination of a hash map and a heap. While a hash map is efficient at searching and returning stored values, a heap is used to arrange the values according to the requirements of the user. The hybrid data structure utilizes both key parts to arrange the sentences in ascending order and retrieve the scores of the words with a hash map. Allowing the code to run efficiently and flexibly.

Practical Applications:

The use of this data structure can be done wherever there is a requirement to obtain a lot of data, with similar scores. Indexing with a hash map allows fast retrieval while heap allows ranking efficiently. Use can be for e-commerce applications where the ranking of the reviews and products must be done based on the ratings of the product. This is required to highlight good products which enhance customer retention rates. The use of this data structure can also be done in supermarkets, where a product must be in big warehouses and their quantity must be updated along with other metrics, while maintaining them in a particular order, based on the requirements of the store.

Experimental setup and methodology used to measure the performance of the hybrid data structure:

The current experimental setup constitutes of multiple tests on data sets found on the internet. We have also used the NLTK corpus, which has a lot of text available for model testing. The efficiency of the data structure in summarizing the key points is remarkable and the use of TF-IDF as a parameter yields good results which encompass the key points in a document provided as an input.

Discussion of the datasets used and any specific considerations for the experiments:

The data sets constitute internet articles as well as the NLTK corpus, especially the state_union data set which comprises recorded presidential speeches. The TF-IDF summarizer was effective in summarizing the given texts and had drawbacks at very few points, wherein there was an anomaly in the text, such as a heading or a quote mentioned as it is. Hence, to evaluate this model specifically, we have considered data sets with only information and no quotes or dates, which cause wrong outputs due to incorrect processing.

Interpretation of results obtained from the experiments, including performance metrics and efficiency improvements:

The result of the project is a GUI-based application that runs on a hybrid data structure background, comprised of a hash map and a heap. They together can efficiently rank and summarize a given input text using the term frequencies and inverse document frequencies as a numerical standard for the text. The performance of the implementation varies based on the input size and data type, however, when provided with plain data conveyed by text, it displays great accuracy in selecting top sentences.

However, to overcome these few drawbacks and to address comparison metrics, we have implemented another algorithm called the Page Rank algorithm, popularly used by Google, using a hybrid data structure model of a graph and a heap. Here the sentences are ranked on their similarity, based on the number of incoming and outgoing edges from each node. The results obtained from both algorithms appear to be efficiently summarized, and the overall result with testing thus far is that both algorithms are almost equally effective in summarizing a given text. Further testing will be done and the results may change in due time.

PageRank Algorithm

The second implementation was using the PageRank algorithm, which uses graphs and heaps to summarize sentences based on the similarity parameter. Each sentence is compared and an edge is added based on the cosine similarity factor if the similarity>=0.5. The number of incoming and outgoing edges is used to determine the importance of a sentence. A node with multiple incoming and outgoing edges is generally not very important as several other sentences with similar meanings are present in the text. Hence, combining the graph data structure with a min heap, ordered based on the number of incoming and outgoing nodes, helps in picking the important sentences.

Chosen Hybrid Data Structure:

The hybrid data structure is a graph with a heap. The hybrid data structure maintains the connection between different sentences, based on their similarity, while the heap ranks them based on their incoming and outgoing edges. The use of this data structure allows for efficient maintenance of the similarities and the ordering and extraction of sentences.

Time and Space Complexities:

The time complexity of the summarization is O(n^2) since the iteration is performed for all the sentences concerning all the other sentences in the collection.

The space complexity is O(n) where n is the number of sentences.

Practical Applications:

The use of this hybrid data structure can be done for various purposes, where the similarity of the items needs to be compared. One of the examples is the use of this algorithm in search engines, for the efficient retrieval of similar web pages to the searched keyword.

Design choices and Trade-offs:

The code is built on the widely available open-source natural language toolkit NLTK. The entire code is then presented with a GUI made using the Tkinter package. The implementation allows the user to enter text in one field and details like the number of sentences are received output almost instantly. The trade-off is that TF-IDF or PageRank is not the most accurate algorithm for extractive text summarization and it might not give the most concise text summary always. The pagerank algorithm here uses cosine similarity while analyzing text samples, whereas there can be several other similarity algorithms with higher complexity that can be used for this purpose.

Conclusion:

The page rank algorithm allows us to do extractive text summarization by using sentence similarity as the main parameter, unlike TF-IDF, which used text frequency as the main parameter. Both the implemented algorithms maintain similar results, with similar drawbacks including the inability to capture the context of the passage. This disadvantage is common among extractive text summarization and can only be overcome using abstractive text summarization, which is outside the current scope of this project. As a comparison of these algorithms, it is found that while TF-IDF is faster, Page Rank can find and list important sentences better. They are efficiently implemented for this case study using two different types of hybrid data structures, which can be widely used for other purposes as well.

GitHub Repository: *https://github.com/Shiva1406/TextSummarizer*