A report on

Search Engine using Knowledge Graph

submitted in partial fulfillment of the requirements
for completion of Data Structures and Algorithms - 2 Lab of

SY COMP

in

Computer Engineering

by

Arnav Prasad, Prajwal Bhosale and Arjun Deodhar
612203011, 612203021 and 612203035

Under the guidance of

Prof Pratiksha Deshmukh
Professor
Department of Computer Engineering

Department of Computer Engineering,
COEP Technological University (COEP Tech)
(A Unitary Public University of Govt. of Maharashtra)
Shivajinagar, Pune-411005,Maharashtra, INDIA

Search Engine using Knowledge graphs.

April 16, 2024

Contents

1 Knowledge Graphs **2**
- 1.1 Introduction . . . 2
- 1.2 Definitions . . . 2
- 1.3 Applications . . . 3
  - 1.3.1 Social Graphs . . . 3
  - 1.3.2 Search Engines . . . 3
  - 1.3.3 Wikipedia . . . 3
  - 1.3.4 Banking and Finance . . . 4

2 Data Structures used **4**
- 2.1 AVL Tree . . . 4
  - 2.1.1 The AVL tree Data Structure . . . 4
  - 2.1.2 Why AVL Trees? . . . 4
  - 2.1.3 Worst case Time complexities of AVL Tree . . . 5
- 2.2 Max Heap . . . 5
  - 2.2.1 The maxheap Data Structure . . . 5
  - 2.2.2 Why maxheap data structure? . . . 6
  - 2.2.3 Worst case time complexities of max heap data structure . . . 6

3 Time Complexity Analysis of Knowledge Graph Algorithms **7**
- 3.1 Insertion . . . 7
- 3.2 Traversals and Querying . . . 8
  - 3.2.1 display_info_lines . . . 8
  - 3.2.2 print_info_lines . . . 9

4 Future Goals to be achieved **11**
- 4.1 Short-Term Goals . . . 11
  - 4.1.1 Implementation of the Serialize() function . . . 11
  - 4.1.2 Using the graph as a server - client system . . . 11
- 4.2 Long term goals . . . 11
  - 4.2.1 Implementing Entity Extraction using ML/NLP algorithms . . . 11

Search Engine using Knowledge Graphs

1 Knowledge Graphs

1.1 Introduction

In the vast landscape of data organization and analysis, knowledge graphs stand out as dynamic frameworks that encapsulate real-world entities and their intricate interconnections. Think of knowledge graphs like giant maps that show how different things in the world are connected. Instead of just listing facts, they link them together, like how your friends are connected on social media. These graphs are smart too! They can figure out new things by using logic, making them more than just collections of information.
In this report, we'll explore what makes knowledge graphs special and how they help us understand the world better.
By putting together different things, their connections, and the little details about them, knowledge graphs go beyond typical ways of organizing information. They give us a complete picture of how everything is linked in various fields of knowledge.

1.2 Definitions

A knowledge graph
(i) mainly describes real world entities and their interrelations, organized in a graph,
(ii) defines possible classes and relations of entities in a schema,
(iii) allows for potentially interrelating arbitrary entities with each other and
(iv) covers various topical domains.
A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge.

Knowledge graphs are large networks of entities, their semantic types, properties, and relationships between entities.

Based on the above definitions, we can refer to knowledge graphs as data graphs potentially enhanced with representations of schema, identity, and context.
A schema defines a high-level structure for the knowledge graph, identity denotes which nodes in the graph (or in external sources) refer to the same real-world entity, while context may indicate a specific setting in which some unit of knowledge is held true.

1.3 Applications

1.3.1 Social Graphs

1.Social media platforms use knowledge graphs to build social graphs of their users to help them maintain and view their users in the context of their connections with other users. Facebook's Entity Graph is a social graph created by Facebook.

2.Predictive knowledge graphs enable media platforms to recommend the most relevant items to their users based on a map of the user's specific interests. For example, movie recommendations by Netflix and friend suggestions from Facebook use predictive knowledge graphs.

3.Content Targeting: Using the structural relations between a user and its likes and dislikes through a behavioral knowledge graph, media platforms infer user preferences to deliver targeted content to user's feeds.

1.3.2 Search Engines

Two most popular search engine knowledge graphs today include the Google knowledge graphs and the Bing knowledge graph by Microsoft. These two knowledge graphs have boosted the way modern search engines operate in the following ways:

1. Search enrichment: By developing rich connections between the information stored on the web, knowledge graphs have refined search down to very specific contextual details. Search engines use knowledge graphs to create the best summary that relates to the search query.

2. Quick information retrieval: Thanks to knowledge graphs, traversing through large volumes of data is faster than ever before. The semantic web makes it easier to go deeper and broader to find the relevant path to navigate for every search query.

3. With knowledge graphs, search engines can perceive entities and properties from natural language voice searches.

1.3.3 Wikipedia

Wikidata is the knowledge graph that powers Wikipedia. It's like the backbone of Wikipedia's information system. In simple terms, it's a giant network of interconnected facts about everything from historical events to scientific concepts. Each piece

of information in Wikidata, like a person's birth date or a city's population, is linked to other related facts. This interconnectedness forms a web of knowledge, making it easier for people to explore and understand topics on Wikipedia.
Wikidata's structured data also helps improve the accuracy and reliability of Wikipedia articles, ensuring that information is up-to-date and consistent across different languages and topics.

1.3.4 Banking and Finance

Knowledge graphs have emerged as a blessing for the finance industry by offering them a secure way of managing their financial knowledge base. Goldman Sachs is a popular example of a banking institution that uses knowledge graphs for transaction and customer analysis.

2 Data Structures used

2.1 AVL Tree

2.1.1 The AVL tree Data Structure

An AVL tree defined as a self-balancing Binary Search Tree (BST) where the absolute difference between heights of left and right subtrees for any node cannot be more than one.
The difference between the heights of the left subtree and the right subtree for any node is known as the balance factor of the node.
In an AVL Tree, the balance factors of all the nodes has to be strictly -1 , 0 or 1.

2.1.2 Why AVL Trees?

Using AVL trees can significantly improve the efficiency of operations like searching, inserting, and deleting data.

1. Efficient Searching: AVL trees are balanced, meaning the depth of the tree is kept to a minimum. This ensures that searching for a specific item in the tree is fast, even as the tree grows larger.

2. Balanced Structure: Unlike regular binary search trees, AVL trees automatically balance themselves after each operation. This prevents the tree from becoming lopsided or skewed, which could slow down operations.

3. Predictable Performance: Because AVL trees maintain their balance, their performance is consistent. Regardless of the order in which data is inserted, the tree maintains its optimal structure, leading to predictable and reliable performance.

4. Optimized for Databases and Search Engines: AVL trees are commonly used in applications where fast searching and sorting are essential, such as databases and search engines. Their efficient structure makes them ideal for handling large volumes of data.

2.1.3 Worst case Time complexities of AVL Tree

1. Search/Lookup : In the worst case, the search operation in an AVL tree requires traversing from the root to a leaf node along a path of height h , where h is the height of the tree. Since AVL trees are balanced and their height is $O(\lg n)$, where n is the number of nodes, the worst-case time complexity for search is $O(\lg n)$.

2. Insertion : $O(\lg n)$.

3. Deletion : $O(\lg n)$.

4.Finding maximum/minimum element: $O(\lg n)$.

2.2 Max Heap

2.2.1 The maxheap Data Structure

A Max-Heap is defined as a type of Heap Data Structure in which each internal node is greater than or equal to its children.
In a Max-Heap the maximum key element is present at the root.
If a heap is stored in an Array arr, the arr[i] corresponds to the value of the node at position i in the heap,

Here we are assuming that the array starts from index zero.
arr[(i - 1) / 2] represents the parent of the node at index i,
arr[2 * i + 1] represents the left child of the node at index i, and
arr[2 * i + 2] represents the right child of the node at index i.

2.2.2 Why maxheap data structure?

1. Fast Insertion and Deletion: Both insertion and deletion operations in a heap have a time complexity of $O(\lg n)$, where n is the number of elements in the heap. This makes heaps suitable for applications where dynamic priority updates are required.

2. Heap Sort: Heapsort is a sorting algorithm that uses the heap data structure to achieve a time complexity of $O(n \lg n)$, making it faster than some other sorting algorithms in certain scenarios.

3. Memory Efficiency: Heaps are often implemented using arrays, which are more memory-efficient compared to other tree structures like binary search trees. This is because the heap structure doesn't require pointers to maintain parent-child relationships.

4. Priority Decidability : We wanted to prioritize sentences in the knowledge graph based on their 'weight', which could be easily done with the help of maxheap data structure.

2.2.3 Worst case time complexities of max heap data structure

1.Insertion and Deletion: $O(\lg n)$.

2. For sorting the array we follow the "while" procedure of the deleting the 1st element , swapping it with the last element and decreasing the length of the array visibility by 1 the swapped element at position 1st will be compared with its child and so necessary swapping will take place. We will loop this till the array length , So that we get the decreasing array of elements. The worst-case time complexity of heapify is $O(n \lg n)$ because it may need to traverse the height of the heap, which is logn.

3. Peek : Peek operation returns the maximum element from the Max Heap. Time Complexity = O(1).

3 Time Complexity Analysis of Knowledge Graph Algorithms

Knowledge graph is implemented using AVL trees and max heaps, and traversed by priority queues. This section deals with the worst-case time complexities of all the algorithms associated with the knowledge graph.

3.1 Insertion

Insertion involves parameters that affect the time complexity:

- `input_noun1` string size = `in_n1`
- `input_noun2` string size = `in_n2`
- `input_verb` size = `in_v`
- `input_verb_desc` size = `in_vd`
- `input_definition` size = `in_def`

All the operations that do not involve constant time are:

- Creation of `noun_3`

This is a concatenation operation, $T(concat) = O(\texttt{in\_n1}+\texttt{in\_n2}) = O(n)$, i.e., linear.

Now, the nouns, verb, and verb_desc are searched in the trees of the knowledge graph:

- If the number of nouns in the tree = p
- Maximum size of noun in the tree = k
- If the number of verbs in the tree = v
- Maximum size of verb in the tree = j

- If the number of verb_desc in the tree = v_d
- Maximum size of verb_desc in the tree = m

AVL search will need to traverse till the leaf nodes in the worst case. Each comparison is a string comparison and takes linear time:

$$T(total) = O(k \lg p) + O(v \lg j) + O(v\_d \lg m) = O(n \lg n)$$

If the nouns, verbs, etc., are not present, they are inserted again, taking $O(n \lg n)$ time.

Next, we check if the connection already exists. This involves searching in various max heaps:

$$T(n) = O(n)$$

Insertion into various heaps must be done if the edge does not exist. If the size of the heap = k, insertion is done in $O(\lg k)$:

$$T(n) = O(\lg n)$$

3.2 Traversals and Querying

The approach for traversing the Knowledge Graph is "weight proportional breadth-first traversal," i.e., when information about a node is to be searched, then the traversal algorithm allocates an amount of lines to each child of that node, which that node can print. This allocation is done based on the weights of children. Since the function is recursive, it will be called for the next level as well, with this similar allocation of lines.

The algorithms of every traversal have the same idea; hence, only two of them are discussed here in detail.

3.2.1 display_info_lines

This function takes a noun (string) and an integer, which is the maximum number of lines to be printed.

For worst-case analysis, we consider this maximum number to be arbitrarily large, and in the context of the C Programming Language, it can be `INT_MAX`.

`display_info_lines` isn't the recursive function though. That work is done by another function which `display_info_lines` calls internally

`display_info_lines does`

- search for noun the function will search for the noun in the noun tree of the knowledge graph As discussed above, this takes $O(kn \lg n)$ time where k = maximum length string in noun tree n = number of nouns in noun tree

- if noun does not exist in this case, a preorder traversal of the tree is done, and each string is matched with the input noun, and percentage of the matching is recorded

  On visiting each node, a comparison is done and the node is added into an array of `noun_node` pointers This entire process will again will take $O(kn \lg n)$ time

- array of choices This is displayed on the screen, and depending on the user's choice, the traversal is done Assuming that the user wants to traverse all the options, and there are k options Then, `print_info_lines` is called for each option

3.2.2 print_info_lines

We will now see the time complexity of `print_info_lines` There are two cases inside this function 1) Too few lines allocated for printing Here, the noun reached has less children than the lines allocated to it. Hence it will print its children's connections and the function will return. If k children are to be printed, then the `search_maxheap_delete` function will be called k times

If the noun has m connections in its `search_maxheap`, then each `search_maxheap_delete` operation will cost $O(\lg m)$. Note that $k \leq m$

$$T(k, m) = O(k \lg m)$$

There are enough children so that we can print them as well as go to the next level

Here, we first need to allocate some lines to each child of the noun In the above example this wasn't necessary since there wouldn't have been any children remaining to allocate lines

Here however, that is not the case

A traversal queue is used for this purpose, and all the k children of the noun are removed and lines are allocated to them

Allocation is an $O(1)$ operation since it just does some calculations
But for these calculations, total weight of all connections is necessary
This takes $O(k)$ time

Later on, constructing the queue while allocating overall takes
$O(k \lg k) + O(k)$

This is overall $O(k \lg k)$

Now, once the traversal queue has been constructed, `print_info_lines` will get called for each node of this queue, with some
number of $lines < totallines$

Where total lines are the lines given to the original function

After all the children are exhausted, and still some lines are remaining to print, we now enter the `subclass_maxheap` of the input noun
The same procedure is followed, allocation, queue construction and recursive function call

After this, the algorithm terminates either when

- there are no connections left to print
- we run out of lines to print

Now we will deal with the recursive call
Taking a look at the structure of the knowledge graph, it is designed such that the same connection will not get traversed / printed more than once.
In the worst case scenario, ALL nodes in the knowledge graph will have a connection with the node with which we started the breadth first traversal.
To worsen the condition (because that's what the worst case is), the node connections might be in a "linked list" fashion.
For example, if the input sentences are of the form:

- noun1 verb1 noun2 ... generates noun12
- noun12 verb2 noun3 ... generates noun123
- noun123 verb3 noun4 ... generates noun1234
- ... and so on

Here we can see that if the traversal begins at noun1, then it can be connected with $n/3$ nouns in the worst case, i.e., `print_info_lines` is called $n/3$ times which is $O(n)$.

The time complexity of `print_info_lines`, T_p, is $T_p = O(kn \lg n)$, where k is the maximum connections of a noun from all the nouns traversed and n is the total number of nodes (nouns) in the graph.

Since `display_info_lines` calls this function and does not increase the time complexity asymptotically, `display_info_lines` is also $O(kn \lg n)$ in the worst case.

4 Future Goals to be achieved

The goals to be achieved are classified into two categories, namely, Short term and Long term goals.

4.1 Short-Term Goals

4.1.1 Implementation of the Serialize() function

We are drawing inspiration from Java to create our own serialization function in C. To Develop a function within the program that allows the graph data structure to be serialized, enabling it to persist beyond the program's execution. Serialization ensures that the graph remains intact even after the program terminates, allowing for efficient storage and retrieval of the graph data.

4.1.2 Using the graph as a server - client system

We've noticed that creating the graph becomes more time-consuming as the input size grows. However, once the graph is built, we can use it efficiently for future queries on the same machine. To achieve this optimization, we'll need to understand socket programming for network communication and learn about how processes communicate with each other on the operating system level and across networks.

4.2 Long term goals

4.2.1 Implementing Entity Extraction using ML/NLP algorithms

Integrate machine learning and natural language processing algorithms to develop an entity extractor capable of processing non-structured data and inputting it into the graph. This long-term goal involves training models to extract relevant entities

and their relationships from unstructured text or data sources.