CHAPTER 5

A SEARCH ALGORITHM ON A RELATIONAL DATASET CONTAINING ALGORITHMS TO FACILITATE REAL TIME CODE GENERATION

5.1 INTRODUCTION

This chapter deals with the construction of a collection of algorithms in a structured relational database pattern that helps to store the algorithms in an intuitive way. The chapter portrays a search algorithm that works on the collection formed to give algorithms as output. The collection will be further called a dataset. These output algorithms can later be put to use in a number of domains as mentioned in the applications section.

The dataset is a connected graph in which the nodes consist of the algorithms and other parameters and the connected nodes have an edge with one or multiple weights. Once the dataset has been constructed new nodes can be added, modified and replaced easily.

The major feature of this dataset is that it can continuously grow and it can be continuously updated as at the very base of this dataset is nothing but a graph, which can be easily changed, modified and updated. By making the dataset open, the very same operations can be performed on the dataset by anyone. This makes it readily available to everyone for use as well as provides a ready base for making one’s algorithm available to everyone for use.
The search algorithm works on this dataset and based on the input keywords it searches for one or more algorithms that have the greatest relevance hit in solving the problem that is defined by the input keywords. These algorithms are given as the output of the search, which can be put to further use. The search algorithm also takes constraints of the algorithms into considerations and the total relevance hit is calculated and based on that the algorithm is searched and given as the output.

5.2 STRUCTURE OF THE DATASET

The dataset i.e. the collection of all the algorithms in a relational model must be constructed, as the dataset on its own is one of the inputs to the search algorithm that will give a relevant algorithm as the output. The construction of the dataset uses the concepts of graphs and thus invariably uses nodes and edges. The nodes are used to store all the information and the edges which shall be weighted and undirected will be used to store the relationship between two nodes or rather two algorithms.

The relational dataset is an undirected tree, which can be defined mathematically as follows

\[ G = \{V, E, W\} \]

where, \( n \) is the total number of vertices present inside the dataset.

\[ V = \{v_i; 1 \leq i \leq n\} \]  \hspace{1cm} (5.2)

\[ E = \{e_i; 1 \leq i \leq n-1: \forall e_i, e_1, v_1 \in V \& e_i, v_2 \in V\} \]  \hspace{1cm} (5.3)

Where \( v_1 \) is start node for that edge and \( v_2 \) is the node on which that edge is incident.
\[ W = \{w_i; 1 \leq i \leq n-1; \forall w_i, w_j \leftarrow e_i, w\} \] (5.4)

Where each \( w \) is a set of keywords.

### 5.2.1 Structure of the Nodes

Before going into the structure of the dataset let the structure of the nodes that the dataset will contain be defined. The nodes will consist of two entities, the algorithm file and the names of the authors. The algorithms must all be present in a readable text file and they must all be written in the structure provided below.

- Each algorithm must begin with ‘Start’ as the first step and ‘Stop’ as the last step.
- Each step must be given sequential numbers without any special characters. E.g. ‘1’, ‘2’, ‘3’ etc.
- Each sub-step must be given roman numbering in lower case without any special characters e.g. ‘i’, ‘ii’ etc.

General guidelines that must be followed in the algorithm file are

- The algorithm should be written in clear, crisp steps.
- There should be no ambiguity in the algorithm.
- All the definitions and the declarations must be properly given.

Thus, the combination of the author names and the file that contains the algorithm forms the contents of a particular node in the graph.
5.2.2 Structure of the Edges

The nodes are connected using edges. These edges are weighted but not with numerical weights. The weights on a particular edge will be the keywords that form the relation from one node to the other. With the formal definition given above for the edge set of $G$ in equation (5.3).

$$E = \{e_i; 1 \leq i \leq n - 1: \forall e_i, v_1 \in V \& e_i, v_2 \in V\}$$

where $v_1$ is the start node for that edge and $v_2$ is the node on which that edge is incident. We can clearly see that every edge $e_i$ will consist of three parameters, which are $v_1, v_2$ and the pointer to the set of weights $w_i$.

The keywords that are present in $w_i$ are the keywords that join $v_1$ to $v_2$. This relation states that the algorithm solved by $v_2$ solves some more additional problems described by the keywords in $w_i$ than $v_1$.

Each edge represents a unique relation and all edges are unique. The intersection of weights of different edges may or may not be disjoint and each and every node represents a single algorithm. There are three functions, which play a significant role in both the algorithm for creation of the dataset as well as the algorithm to search for an algorithm stored inside the dataset.

1. Function to extract the sub graphs from the dataset
2. Function to find the longest chain in the graph
3. Function to sort the paths that have been found

5.3 WORKING OF THE DATASET

For the dataset to be created as well as to be searched we need keywords. These keywords are nothing but a specific set of words that define
the problem that is to be added to the dataset or it defines the problem for which an algorithm is being searched from the dataset. This set of keywords can be denoted mathematically as $K$ where

$$K = \{k_i; 1 \leq i \leq m\} \quad (5.5)$$

Using these keywords the entire dataset will be searched to find the place to add a new algorithm or to find the node(s) that contain the algorithm(s) that will be used to solve the problem defined by the keywords. The three major functions that are called in sequence to find nodes are as follows. The commented pseudo-codes of the functions are given Appendix 3.

5.3.1 Function to extract the sub graphs from the dataset

The function does the following

- It finds all the edges from the tree $G$ such that at least one of the weights on each edge matched with at least one keyword.

- From the set of edges that have been computed from step one it stiches these edges into one or multiple subgraphs.

This function uses many sets that have been defined below,

$$K = \{k_i; 1 \leq i \leq m\},$$

this is the set of the input keys that have been given by the users where $m$ is some positive integer.

$$S = \{e_i; 1 \leq i < n - 1|\exists k_i \in K\} \in e_i.w,$$

this set is the set of all the edges in $G$ such that at least one of the weights on each of those edges has at least matching keyword in $K$.

$$G' = \{G'_i; \forall G'_i = \{V', E'\}|G'_i \subseteq G\},$$

this is a set of all the subgraphs that are formed form the edges that are present in $S$. 


\[ A = \{ v_i | \forall e_i, v_i = e_i \cdot v_2 \text{ where } e_i \in S \} \] this is the set of all the vertices that have the edges of S as their incident edges. These vertices contain the algorithm that is able to define the problem defined by K at least partially as each and every one of those vertices has at least one of the keywords matching with K in (5.5).

\[ B' = \{ B_i \}; B_i = \{ \{ \text{longest} \}, \{ \text{klist} \} \} \] (5.6)

where \text{longest} = the longest relevance path in the \( G_i \) subgraph and \( \text{klist} = \) the list of all the keywords on the longest relevance path. The algorithm to find the subgraphs can be defined as follows

\begin{verbatim}
extractSubgraphs()
{
    S' = {};
    \forall G'_i, G'_i V' = \{ \}, G'_i E' = \{ \};
    V'' = {};
    e_x = e_1  // e_1 is the first edge in the set S
    j = 1;
    do
    {
        v_x = e_x v_1
        G'_j V' = G'_j V' \cup \{ v_x \};
        for \forall v_i \in G'_j V' \& \& \forall v_t \in V''
        {
            S' = \{ e_t' : 1 \leq l < n - 1 | e_t, v_l = v_x \forall e_l \in S \}
            G'_j E' = G'_j E' \cup S';
            G'_j V' = G'_j V' \cup \{ v_{t'} v_l = e_{l'} v_2 \};
            S = S - S';
            S' = {};
        }
    } while (j < n);
\}
\end{verbatim}
\[ V'' = G_j \cap V' \cup V'' ; \]
e\_x = \{ e | e_2 = v_x \}; 

\[ if(e_x = \phi \& | S| \neq 0) \]

\{ 
  head \_j = v_x 
  j += ; 
  e_x = e_t ; 
\}

\} while(| S| \neq 0) ;
\}

5.3.1.1 Time complexity for algorithm to extract subgraph

The above algorithm forms a crucial part of all the algorithms for the dataset and thus its time complexity contributes significantly to the time complexity of those other algorithms and thus the time complexity of the above algorithm must be calculated.

To calculate the worst case time complexity let an assumption made that at every step, for all factors that can contribute to the runtime of the above algorithm the size of that factor is \( N \) which is very large.

- It is know that every \( v \) will take \( O(N) \) time complexity
- It is also know that every \( u \) will take \( O(N^2) \) time complexity as in every addition to the set there is a need to ensure that it does not already exist in the set to which it is being added.
Using the above two rules and the standard rules of finding the time complexity in the worst case, the total worst-case time complexity can be calculated.

The above algorithm has the time complexity as calculated below

- do - while loop – $O(N)$
- for - loop – $O(N)$
- A block of statements – $O(1)$

$$= O(1) + \left( O(N) \right) + O(N) + \left( O(N) \ast O(1) + O(O(N^2)) \right) + \sum \left( O(N^2) \right)$$

$$= O(1) + (O(N) \ast O(N) + (O(N) \ast O(N^2)) + (O(N^2)))$$

$$= O(1) + ((O(N) \ast O(N^3)) + O(N^2))$$

$$= O(1) + (O(N^4) \ast O(N^2))$$

$$= O(N^4)$$

Thus we can conclude that even in the worst case scenario the algorithm runs in polynomial time complexity.

5.3.1.2 Explanation of the algorithm to extract subgraph

In the above algorithm the edges having weights that match the keywords that have been given by the user are computed and then these edges are used to create the subgraph.
- The first computed edge is taken.
- The first node of this edge is taken as the initial node in the subgraph.
- All the computed nodes are searched for the edges that have the same first node as the initial node.
- All these edges along with their second nodes are added to the subgraph.
- This process is repeated for all the nodes that are in the subgraph.
- Once out of the for loop, it is calculated if the subgraph is yet to be complete by checking if the initial node is the second node for any of the edge in the set of computed edges. If yes then the first node of that node becomes the initial node and the process is repeated. Else if there are still some edges left in the computed set of edges then the next subgraph is made.
- All the above steps are repeated until the computed edge set is empty.

5.3.2 Function to Find the Longest Chain in the Graph

The second algorithm that is used to find the longest chain in all the algorithms for the data set is given below.

\[
\text{longestChain()} \\
\{ \\
\text{for } \forall G_i \in G' \\
\{ \\
\text{DFS}(G_i, \text{head}, i) \\
\} \\
\} \\
\text{DFS}(start, i)
\]
5.3.2.1 Time complexity of longest chain algorithm

To calculate the time complexity there is a need to calculate the time complexity of Depth First Search DFS( ) and to calculate the time
complexity of $\text{DFS}(\cdot)$ there is a need to calculate the time complexity of the \text{max}(\cdot) function.

The time complexity of the \text{max}(\cdot) function

\[ O(1) + (O(N) \times O(1)) \]

\[ = O(1) + O(N) \]

\[ = O(N) \]

The time complexity of the $\text{DFS}(\cdot)$ function

\[ O(N) \times (O(1) + (O(N) \times O(1) + \max(O(1), O(N)))) \]

\[ = O(N) \times (O(1) + O(N^2)) \]

\[ = O(N) \times O(N^2) = O(N^3) \]

As the recursive call happens $N$ times the entire time complexity is multiplied by $O(N)$ and thus we get $O(N^3)$

The time complexity of the $\text{longestChain}(\cdot)$ function is

\[ O(N) \times O(N^3) = O(N^4) \]

Thus it can be concluded that $\text{longestChain}(\cdot)$ has a polynomial time worst case time complexity.

5.3.2.2 Explanation of the longest chain algorithm

In the function for longestChain(), a depth first search is applied on each of the subgraphs that is computed to calculate the longest weighted path in them. Here the weights would be the total number weights in the path that
match with the keywords. This function results in two sets; one, which contains the vertices of the path and the other, contains all the keywords encountered in that path.

5.3.3 Function to sort the paths that have been found

```c
sortPaths()
{
    for ∀Bᵢ ∈ B'
    {
        for ∀Bⱼ ∈ B' \ {Bᵢ}
        {
            if(|Bᵢ.klist| < |Bⱼ.klist|)
                Bᵢ ↔ Bⱼ
        }
    }
}
```

5.3.3.1 Calculating the time complexity of sortPaths()

\[ O(N) \times (O(N) \times (O(1))) = O(N) \times O(N) = O(N^2) \]

Thus the function `sortPaths()` has a polynomial time run time complexity in its worst case scenario.

5.3.3.2 Explanation of sortPaths()

This function is used to sort the longest paths of all the subgraphs in descending order.
5.4 CREATION OF THE DATASET

The dataset creation is one of the most important processes related to the dataset as it includes adding new algorithms or replacing previous algorithms etc. Since both processes are those that will still take place even after the dataset is created hence this algorithm plays a vital role. Apart from that, this algorithm helps in building the entire tree that is the dataset.

In this algorithm an input node is provided along with a set of keywords that best define the problem that is solved by the algorithm

5.4.1 Algorithm to Create Dataset

create( )
{
    extractSubgraphs( );
    longestChain( );
    sortPaths( );
    \[ K = K - B_1.klist; \]
    \[ V_s = B_1.longest[|longest| - 1]; \]
    \[ V_i = getInput( ); \]
    if ( \( K = \phi \))
    {
        if (\( V_i.author = V_s.author \))
            \( V_i \leftarrow V_s \)
        else
        {
            \[ V = V \cup \{ V_i \}; \]
            \[ E = E \cup \{ e | e.V_1 = V_s.e.V_2 = V_i.e.w = "same" \}; \]
        }
    }
}
\begin{verbatim}
else
{
    V = V \cup \{V_i\};
    E = E \cup \{e | e.V_i = V_j \& e.V_2 = V_j \& e.w = K\};
    W = W \cup \{w_i = K\};
}
\end{verbatim}

\subsection{Time Complexity for Create Dataset Algorithm}

Using the results for of time complexity calculated for the previous functions, the time complexity of this function can be calculated. In this function the union operation is taking place with just one input hence it will take only $O(N)$ as it only needs to search the set to check if it has already been added or not before adding it.

\begin{align*}
O(N^4) + O(N^4) + O(N^2) + \max (O(1), O(N) + O(N), O(N) + O(N) + O(N) + O(N)) \\
= O(N^4) + O(N) \\
= O(N^4).
\end{align*}

Thus it can be concluded that create () function has a polynomial time run time complexity in its worst case scenario. Thus each and every operation of addition of a node or modification of a node in the dataset takes polynomial time to execute and since the creation of the dataset is a collection of these polynomial operations, the creation of the entire dataset also takes polynomial time complexity, hence the creation of the dataset belongs the P problem.

\subsection{Explanation of the Create Dataset Algorithm}

In this algorithm the previously defined functions are used to find the place where the input node will be attached. Once the longest chains of all
the sub graphs are obtained then sorting is done in a way such that the first longest chain is the chain that has matched the maximum number of keywords with those given as the input and thus the new node has to be attached to this very chain. If all the keywords have been matched then it means that the input node contains another version of the algorithm that already exists in the dataset. If the author of the last node in the chain and the author of the new node is the same then it means that the author has uploaded a new version of algorithm and thus it is replaced, else if it is from a different author then it is considered as a different version of the algorithm and added to the last node of the longest chain with the edge having the keyword as same which states that the algorithm is similar to its parent. If all the keyword has not been matched then the node is added to the end of the chain.

5.5 SEARCHING THE DATASET

The main purpose of creating such a dataset is to facilitate the problem solving and automatic code generation. However, for that to work a searching algorithm is needed. The search algorithm works on keywords. The search algorithm must be able to give a number of plausible solutions in the form of algorithms and all these solutions must be ranked to make it user friendly. These algorithms can be converted into code.

The proposition of one such algorithm is given below. It takes a set of input keywords and thereby with the help of that it searches for the algorithms.

5.5.1 The Search Algorithm

\[
\text{search}(K)
\]

\[
\{ \\
\text{extractSubgraphs( )}; 
\]
longestChain();
sortPaths();
for ∀Bᵢ ∈ B'
{
    output(Rankᵢ, Bᵢ, longest);
}
output(lowest_rank, A);
output(lowest_rank, A);

5.5.2 Time Complexity of Search Algorithm

Using the results for of time complexity calculated for the previous functions the time complexity of this function can be easily calculated.

\[ O(N^4) + O(N^4) + O(N^2) + O(N) + O(1) \]

\[ = O(N^4) \]

Thus it is concluded that the search function takes polynomial time complexity in the worst-case scenario. The problem of finding an algorithm from the dataset belongs to the P class.

5.5.3 Explanation of the Search Algorithm

To search the algorithm, there is a need to find the nodes that match the constraints that are set by the keywords. The functions defined previously helps in achieving the exact same thing. The longest chains are obtained consisting of nodes which satisfy the maximum matching with the given keywords, and the function sortPaths() sorts all the paths such that the first path is the one that has the maximum matches with the given inputs.
The final set A is also given as output as the set A consists of all the nodes that match at least one of the constraints set forth by the keywords. In this manner the dataset is created and the search can be also used to get the algorithms as the output.

5.6 USE OF THE DATASET

This kind of dataset facilitates a lot of applications however it is limited here for two applications

5.6.1 A Search Engine for Algorithms

This application is one of the most evident applications of this kind of dataset. This dataset provides one of the first and unique search engines specifically for algorithms as it can be added, as well as search for algorithms using these kinds of dataset can be achieved.

5.6.2 Real Time Generative Programming

The major feature of this kind of application is that it provides algorithms as output. In a real time generative programming model provided in the previous chapter, “Real Time Code Generation using Generative Programming Paradigm” a model for meta-programming that is used to code the problem after an algorithm for that problem has been obtained is achieved by this proposed method. This algorithm can be used by the meta-programming module. The algorithm helps the meta-programming module to code the program for the problem given by the user.

This dataset is at the core of any application of Generative Programming as this dataset provides the base inputs for any code generation, which are nothing but pseudo codes or algorithms. Thus this dataset has the most important application in Generative Programming Paradigm.
5.7 CONCLUSION

The creation of a relational dataset that can hold the algorithm is one of the first works that provides the basis to move in a direction in which auto code generation can be achieved and machines can solve problems by interacting with them by using various paradigms such as Natural Language Processing, Generative Programming and using the dataset even more can be achieved in automatic code generation. The dataset and the algorithms provide a path for search engines for algorithms and thus common platforms to spread algorithms as well as a common platform to search for algorithms that can help developers and coders to develop better code are achieved.