ACCESSIBILITY OF NODES IN A GRAPH USING NOVEL NOMENCLATURE AND COMPUTING TIME GAIN WHEN COMPARED TO USUAL METHODS

BY

S.Prathiba

CB.EN.U4CSE17141

BTech. Computer Science Engineering

Under the guidance of

Professor Rupesh Nasre

CSE department, IIT Madras

AMRITA VISHWA VIDYAPEETHAM, COIMBATORE

Nov, 2018

ABSTRACT

Graphs being a very powerful tool of representation of various statistics have always been a part and parcel of our lives. We come across various graphs applications in our day to day lives and the extent to which graph algorithms have evolved is immense. Some of the applications are to indicate the event flow of computation, google maps which help us in finding our destination, the social media which involves the concept of mutual friends and friend suggestions, in Operating systems for resource allocation where processes and resources are considered as vertices etc. All these involve graph algorithms where in each of them have their own goals.

In this project I have worked on finding the accessibility of nodes in a constant time with some pre-computations done on the graphs. This reduces the time taken for each query greatly when compared to the normal DFS method. When the data set is really huge this algorithm will prove to be highly efficient.

When graphs are taken they can be highly complicated with cycles, self loops etc. The tediousness of finding reachability within nodes also grows when the number of edges and vertices are very large. To resolve these types of difficulties in finding the accessibility between nodes this algorithm is highly useful.

The generation of graphs has been automated and for input and output the concept of file handling has been used. Results prove that the actual querying process takes a very meagre time when compared to the normal DFS algorithm. The sum of pre-computation time along with querying time is less than the time

taken to find reachability with normal DFS. This time gain may look trivial when the graph is small, but proves to be highly useful when we are dealing with a very huge data set like the whole social media interlinks.

The aim of this project was to compute the time gain in finding the accessibility of various nodes from one another .This has been successfully found and tabulated and a literature study has been carried out.

ALGORITHM

1. Considering the given graph and removing all the cycles present by the coalescing the individual strongly connected components into a single node. This will create a new directed acyclic graph (DAG).
2. Add a new vertex START, which has an edge from START to each of the zero in-degree vertices.
3. Add another new vertex END with an edge from each zero out-degree vertex to END.
4. Maintain the adjacency list for each vertex in the clockwise order of embedding of nodes.
5. Each node maintains a pair in which the first element of the pair is a result of the first DFS and the second element of the pair is a result of the second DFS.
6. Start the first DFS from the vertex START. Initialize a counter 'i =n+1'.
7. During this DFS traversal, traverse the adjacency list from **left to right** as required.
8. As the vertices are popped from the traversal stack, they are labeled with 'i 'at the end of the recursion call for that particular node. Then 'i ' is decremented.
9. END is named with n+1.
10. START is named 0.
11. All the nodes are numbered this way during the first DFS, and this constitutes for the first element of the pair of each node.
12. Start the second DFS from the node START .In this DFS traversal, we traverse the adjacency list from **right to left** and a similar naming is used.
13. When both the DFS's are complete, the pair for all the nodes is named.
14. The START and END vertices can now be removed which culminates the preprocessing of nodes.
15. To find if a node V is reachable from a node U, we can just compare the pairs of both the nodes.

U - (a1,a2)

V - (b1,b2)

If a1<=b1 ; a2<=b2

Then node V is reachable from node U.

Here the preprocessing happens in O(n) time. The comparison for reachabilty happens in O(1) time.

DETAILS OF THE CODE:

- This code (1.cpp) randomly generates an impartial graph which is taken as input for it to be processed in (2.cpp).
- When we execute 1.cpp, a text file called "Test_Cases_Directed_Unweighted_Graph" is generated in the same directory where we have put both the cpp files.
- The first row of the text file has 2 parameters which indicate the number of edges and vertices in the generated graph respectively. (The max value is 10^6, which is mentioned in the code as a comment).
- The corresponding rows of the graph give the details about the directed edges.
- In the text file the numbering starts from 1, but the code has been coded with respect to the 0 indexing.(i.e. if the first row has 253 7956,the first accessible vertex is 0 and the last accessible vertex is 252).
- After 1.cpp is run, and the text file is generated for our reference. We can now run 2.cpp, input the number of queries and get the output whether it is reachable or not and also obtain the time taken for that particular query. The result can correspondingly be checked upon in the generated text file also.

DESCRIPTION OF THE CODE (2.cpp):

- The initial Graph class in the code is for generating a Directed Acyclic Graph from the generated graph since this algorithm needs a DAG as an input.
- This is done by finding the strongly connected components and coalescing all the components of a single strongly connected component into a single node and repeating this process throughout the graph, and generating a completely new graph. This graph is used for further processing.
- The variable “cnt” gives the number of vertices in the new DAG.
- Then I declare a vector of sets called “dag” to main the adjacency list of each node in the graph.
- As we are combining the vertices of the old graph and creating new vertices, we need to update the adjacency lists of the group of vertices by combining them and removing the individual components of the strongly connected components. This is done by a set “com” so that duplicates are automatically removed.
- A set “checkout” is used to compute the out-degrees for the vertices. A vector “freq_in” is used to compute the in-degrees for the vertices.
- The double DFS is then carried out separately by the functions dfs_left and dfs_right respectively. The dfs_right does the DFS by traversing the adjacency lists from right to left.
- While performing the respective dfs’s the pair for each vertex is updated.
- The dfs_left updates the first component of the pair and dfs_right updates the second component of the pair.
- At the end of these DFS’s a vector of pairs (L) is ready.
- This completes the pre-computation process. Now the concept that if L[u] <L[v] then node v is reachable from node u .This is done in O(1) time.