Artificial Intelligence Project

Ego Network Analysis using Depth Limit Search

Project Members

Jaishree Dhage	Reg.No 140905350
Mahidhar	Reg.No 140905294
Tuhin Khare	Reg.No 140905056

ACKNOWLEDGEMENT

This project was developed as a part of Manipal Institute of Technology in co-operation with the Department of Computer Science and is a result of co-operation and support or a large number of faculties of the Department. We would like to take this opportunity and extend our gratitude to all of them who have contributed towards the success of this project directly or indirectly.

We extend our heartfelt thanks to Ms. Josephine Veena for giving us the opportunity to present this topic as a project and constantly guiding us throughout the process of development of the project in an academic as well as in a moral manner, without the academic guidance of whom this project wouldn't have been possible, her suggestions and instructions have served as a major contributor towards the completion of this project.

We would like to thank our respective parents and friends for their valuable suggestions and guidance in various phases of completion of the project.

ABSTRACT

Results the connection between two people in a particular ego network when the number of links which are to be traversed in the ego network are limited. This is similar to finding friends of friends of friends of friends and so on (depending on the limit chosen by the user) of a particular person in a social network, the graph constructed is of a random rewiring probability and has a random clustering coefficient.

In this project we are achieving the mentioned purpose above by generating a random social network graph which is an undirected graph on a user specified vertices and edges. We are using the depth limit search algorithm with a user-specified limit in order to traverse multiple links in the ego graph, for every node traversed the corresponding alters are listed out which then are recursively traversed in the same manner in as their ego. Apart from listing out the connections of connections for a particular ego the results also depict, the number of active connections of a person in the network.

This project can be implemented as a feature of professional social networking sites in order to create a better professional network for a particular person, also by combining greedy cost based parameters into the algorithm we can provide the best path in order to approach an indirect connection.

PROBLEM DEFINITION

The problem addressed by this project is to generate all the possible indirect connections of a particular person in the ego network with given limit on the number of edges that can be traversed. The names of all the people connected indirectly should be displayed along with the degree of indirectness i.e also the depth at which the connection is found.

Initial State : The initial state of the problem is the random undirected ego graph created with a user specified nodes and edges start of the execution.

Action : Depth limit search algorithm applied on the graph generated

Transition Model :
```
if (current node not in stack) {
        push node in stack
}
```

Environment : the generated ego network

Data Set Description : the data set is basically an undirected graph which each node consisting of a node id in range [0,n-1] where n is the total number of nodes specified by the user, alongside each node consists of a name attribute which signifies a person in the social network, this attribute is generated randomly at the time of the graph creation

OBJECTIVES

The major objectives of the projects are :-

- Create a Command Line Interface (CLI) which generates a random undirected ego-network graph for a user specified number of nodes , number of edges and a random rewiring probability and clustering coefficient
- Application of the Depth Limit Search Algorithm for a user specified limit on the generated ego-network and display the indirect connections of a specific person along with the extent of indirectness (depth at which the connection was found)
- Plot the Ego-Network graph using graph visualisation tools on a png file in 2D as well as in 3D
- Comparison of Depth Limit Search algorithm with several other algorithms implemented on the same ego network and list out of the advantages of each algorithm.

LITERATURE SURVEY

Social Networks are complex networks consisted of nodes and edges connected by a complex interconnections each node represents a person and edges represent the interconnections. Ego Network graphs are an abstract representation of a social network with circles representing nodes which in-turn represent a person in the actual social network. Ego-networks are used in order to understand as well as analyse real time offline as well as online social networks in a very efficient as well as precise manner.

The past researches on ego-network analysis are

- **Analysis of Online Social Network (OSN) Ego Networks by Institute of Informatcs and Telematics CNR, Italy**

  In this paper they aim to discover the presence of Dunbar' circles in OSN ego networks. With this purpose, they analyse a data set containing more than 23 million social interactions in Facebook to see whether they follow a particular structure mentioned below.

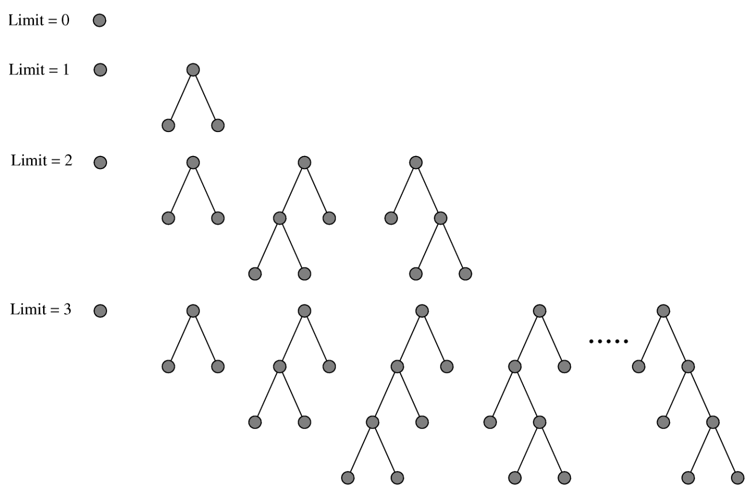
  They find that the properties of OSN ego networks have a strong similarity with t those found in offline ego networks.Namely, the typical number of circles in the structure of virtual ego networks is, on average, equal to 4 and the average scaling factor between the concentric circles of the social structure is near to 3, as found in real environments.Moreover, the sizes of the circles, i.e. the number of social relationships of each type, is remarkably similar to those existing in offline social networks. Notably, the average size of the OSN ego networks is very close to the well known Dunbar's number, which denotes the average size of ego networks in offline social networks.

- **Using Ego Networks in Surveys : methodological and empirical research issues by University of Trento, Italy**

  The paper is aimed at illustrating the use of social networks and specifically ego-centered networks (or personal networks) in surveys. The discussion focus on general framework of a an extensive research project on social capital (SC) among small entrepreneurs in Italian industrial districts. By means of CAPI techniques, personal networks of a sample of entrepreneurs (N=352) have been collected and different dimensions of SC have been measured looking at main aspects of the entrepreneurs activity (economic relations, family and friends, membership and participation to associations)red. The paper describes the methodological aspects of the project, discuss some results and concentrate on the design of empirical research with ego-centered networks.

IMPLEMENTATION

DESIGN



The basic design of the depth limited search algorithm is shown above.

PSEUDO CODE

```
DLS(node, goal, depth)
{
  if ( depth >= 0 )
       { if ( node == goal )
               return node for each child in expand(node)
              DLS(child, goal, depth-1)
       }
 }
```

NODE STRUCTURE

The node structure of the ego-network graph generated upon the execution of the code is of the format

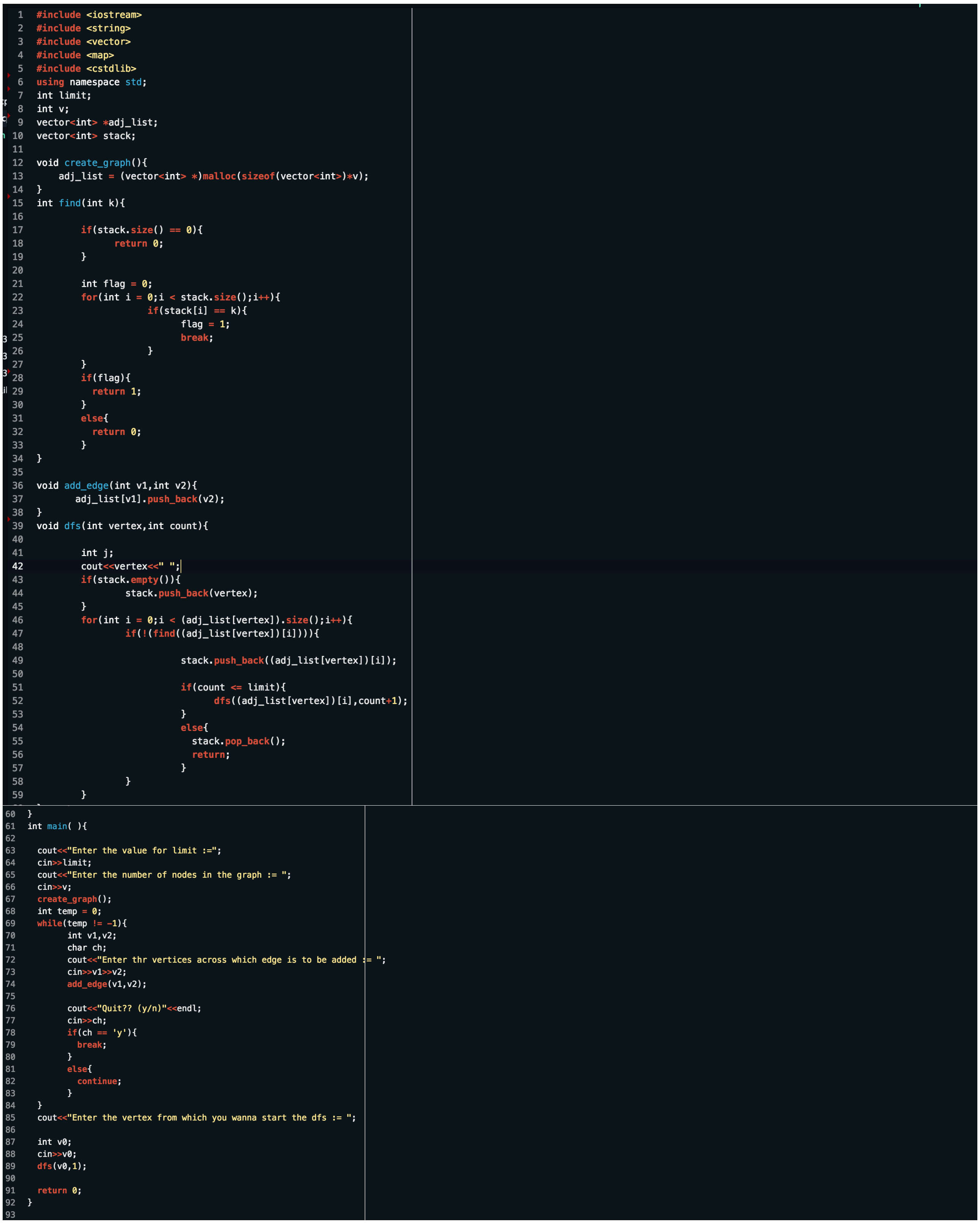
Node(i) = { node_id , name }

node_id - id for a particular node which ranges between [0,n-1] where n is the total number of nodes

name - name of the person who is representing that particular node in the ego-network.

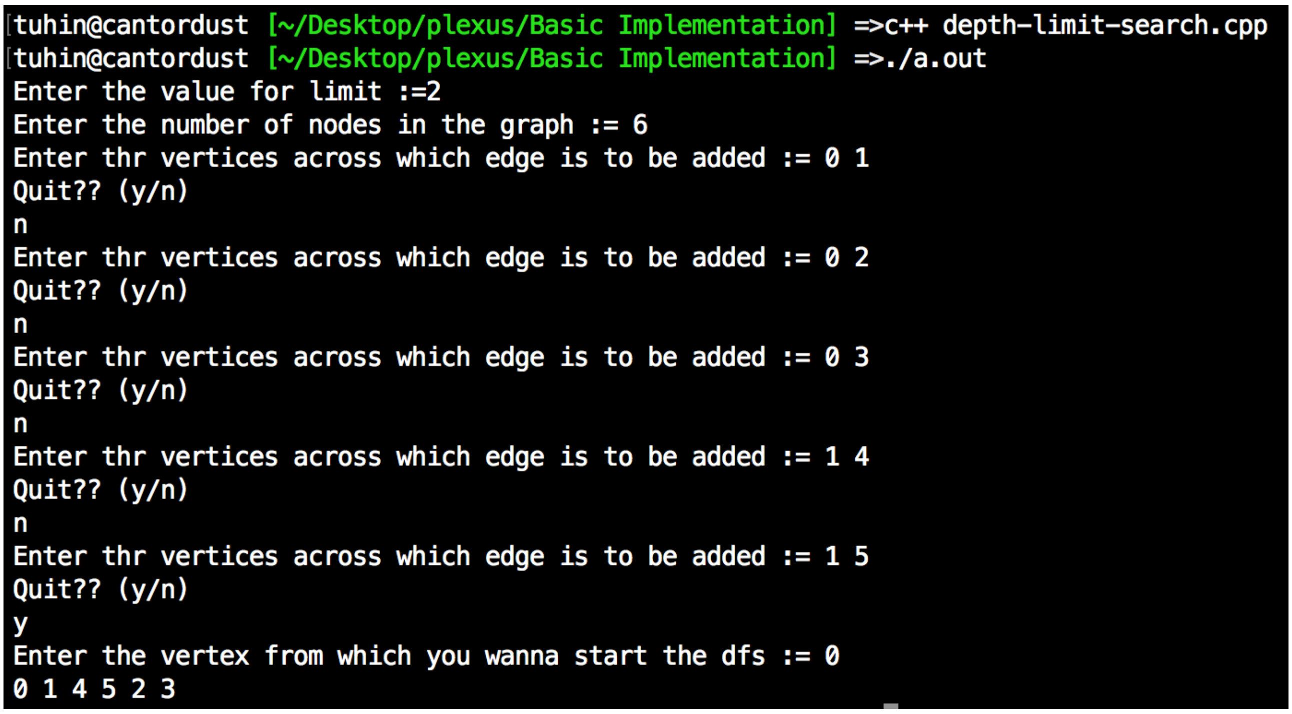
CODE

A basic Implementation of the depth first search algorithm in C++



```cpp
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <cstdlib>
using namespace std;
int limit;
int v;
vector<int> *adj_list;
vector<int> stack;

void create_graph(){
    adj_list = (vector<int> *)malloc(sizeof(vector<int>)*v);
}
int find(int k){

        if(stack.size() == 0){
              return 0;
        }

        int flag = 0;
        for(int i = 0;i < stack.size();i++){
                    if(stack[i] == k){
                          flag = 1;
                          break;
                    }
        }
        if(flag){
          return 1;
        }
        else{
          return 0;
        }
}

void add_edge(int v1,int v2){
       adj_list[v1].push_back(v2);
}
void dfs(int vertex,int count){

        int j;
        cout<<vertex<<" ";
        if(stack.empty()){
                stack.push_back(vertex);
        }
        for(int i = 0;i < (adj_list[vertex]).size();i++){
                if(!(find((adj_list[vertex])[i]))){

                          stack.push_back((adj_list[vertex])[i]);

                          if(count <= limit){
                                dfs((adj_list[vertex])[i],count+1);
                          }
                          else{
                            stack.pop_back();
                            return;
                          }
                }
        }
}
int main( ){

  cout<<"Enter the value for limit :=";
  cin>>limit;
  cout<<"Enter the number of nodes in the graph := ";
  cin>>v;
  create_graph();
  int temp = 0;
  while(temp != -1){
        int v1,v2;
        char ch;
        cout<<"Enter thr vertices across which edge is to be added := ";
        cin>>v1>>v2;
        add_edge(v1,v2);

        cout<<"Quit?? (y/n)"<<endl;
        cin>>ch;
        if(ch == 'y'){
          break;
        }
        else{
          continue;
        }
  }
  cout<<"Enter the vertex from which you wanna start the dfs := ";

  int v0;
  cin>>v0;
  dfs(v0,1);

  return 0;
}
```

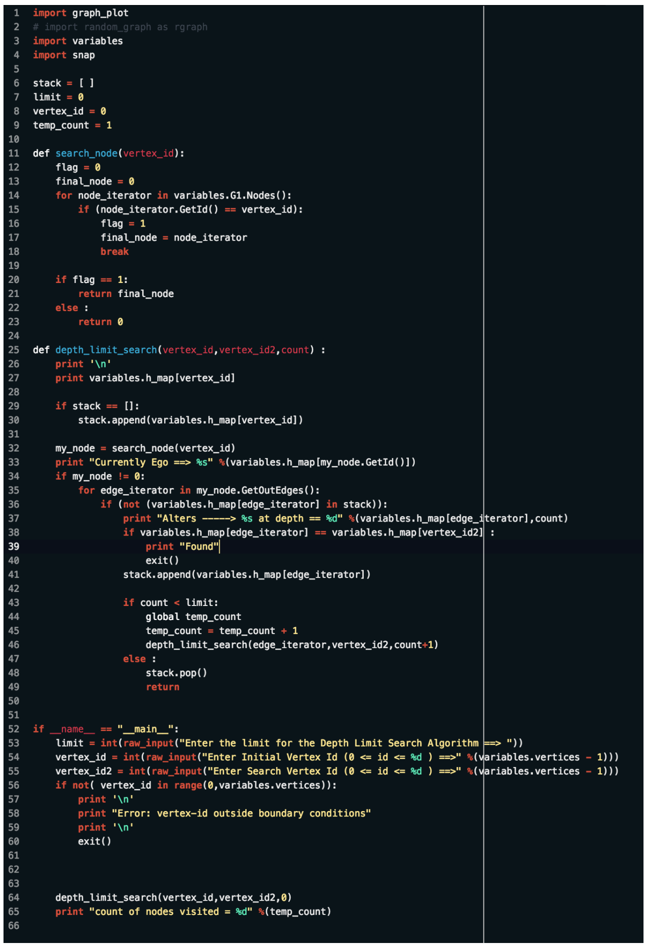
OUTPUT



```
tuhin@cantordust [~/Desktop/plexus/Basic Implementation] =>c++ depth-limit-search.cpp
tuhin@cantordust [~/Desktop/plexus/Basic Implementation] =>./a.out
Enter the value for limit :=2
Enter the number of nodes in the graph := 6
Enter thr vertices across which edge is to be added := 0 1
Quit?? (y/n)
n
Enter thr vertices across which edge is to be added := 0 2
Quit?? (y/n)
n
Enter thr vertices across which edge is to be added := 0 3
Quit?? (y/n)
n
Enter thr vertices across which edge is to be added := 1 4
Quit?? (y/n)
n
Enter thr vertices across which edge is to be added := 1 5
Quit?? (y/n)
y
Enter the vertex from which you wanna start the dfs := 0
0 1 4 5 2 3
```

CODE IMPLEMENTED ON THE CASE STUDY

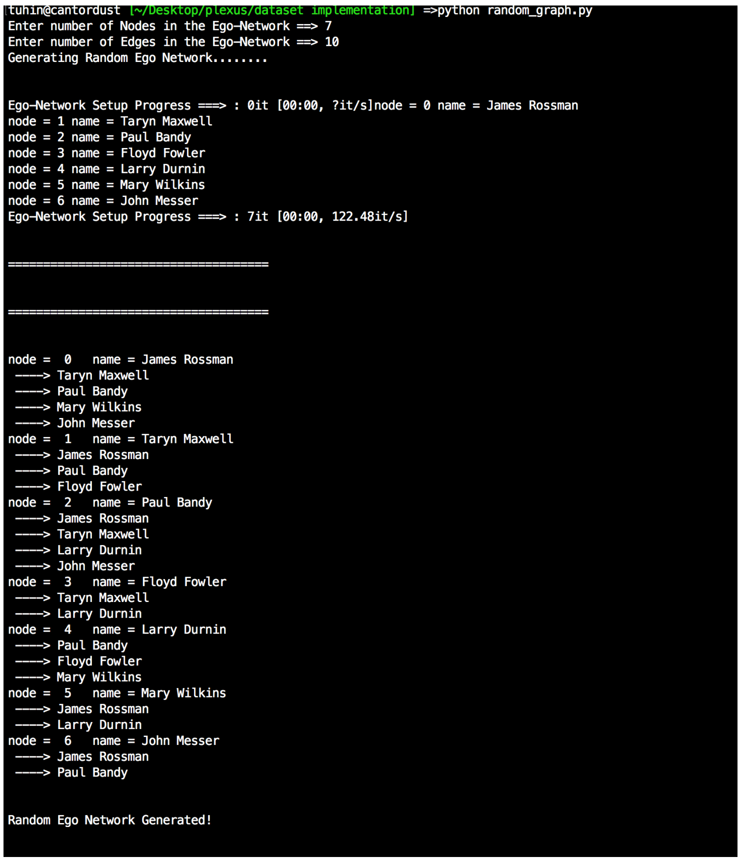
CODE



```python
import graph_plot
# import random_graph as rgraph
import variables
import snap

stack = [ ]
limit = 0
vertex_id = 0
temp_count = 1

def search_node(vertex_id):
    flag = 0
    final_node = 0
    for node_iterator in variables.G1.Nodes():
        if (node_iterator.GetId() == vertex_id):
            flag = 1
            final_node = node_iterator
            break

    if flag == 1:
        return final_node
    else :
        return 0

def depth_limit_search(vertex_id,vertex_id2,count) :
    print '\n'
    print variables.h_map[vertex_id]

    if stack == []:
        stack.append(variables.h_map[vertex_id])

    my_node = search_node(vertex_id)
    print "Currently Ego ==> %s" %(variables.h_map[my_node.GetId()])
    if my_node != 0:
        for edge_iterator in my_node.GetOutEdges():
            if (not (variables.h_map[edge_iterator] in stack)):
                print "Alters -----> %s at depth == %d" %(variables.h_map[edge_iterator],count)
                if variables.h_map[edge_iterator] == variables.h_map[vertex_id2] :
                    print "Found"
                    exit()
                stack.append(variables.h_map[edge_iterator])

                if count < limit:
                    global temp_count
                    temp_count = temp_count + 1
                    depth_limit_search(edge_iterator,vertex_id2,count+1)
                else :
                    stack.pop()
                    return


if __name__ == "__main__":
    limit = int(raw_input("Enter the limit for the Depth Limit Search Algorithm ==> "))
    vertex_id = int(raw_input("Enter Initial Vertex Id (0 <= id <= %d ) ==>" %(variables.vertices - 1)))
    vertex_id2 = int(raw_input("Enter Search Vertex Id (0 <= id <= %d ) ==>" %(variables.vertices - 1)))
    if not( vertex_id in range(0,variables.vertices)):
        print '\n'
        print "Error: vertex-id outside boundary conditions"
        print '\n'
        exit()



    depth_limit_search(vertex_id,vertex_id2,0)
    print "count of nodes visited = %d" %(temp_count)

```

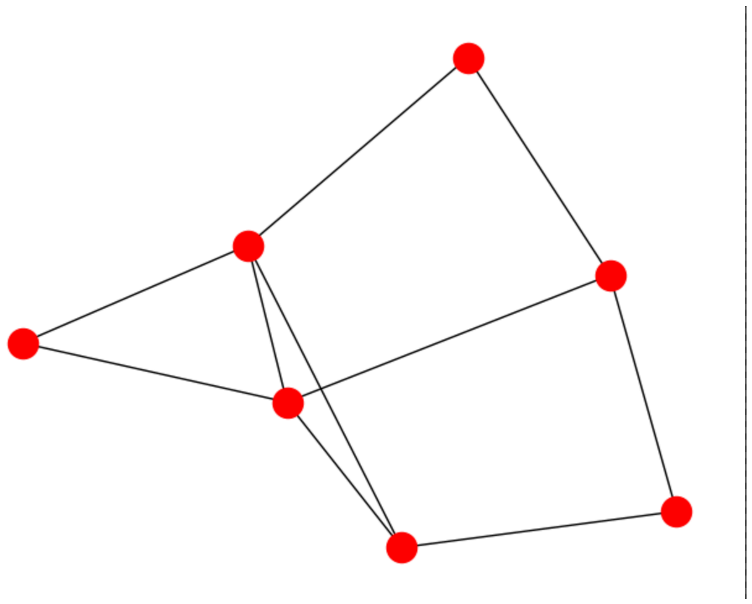
OUTPUT FOR THE TEST CASE

GENERATION OF GRAPH

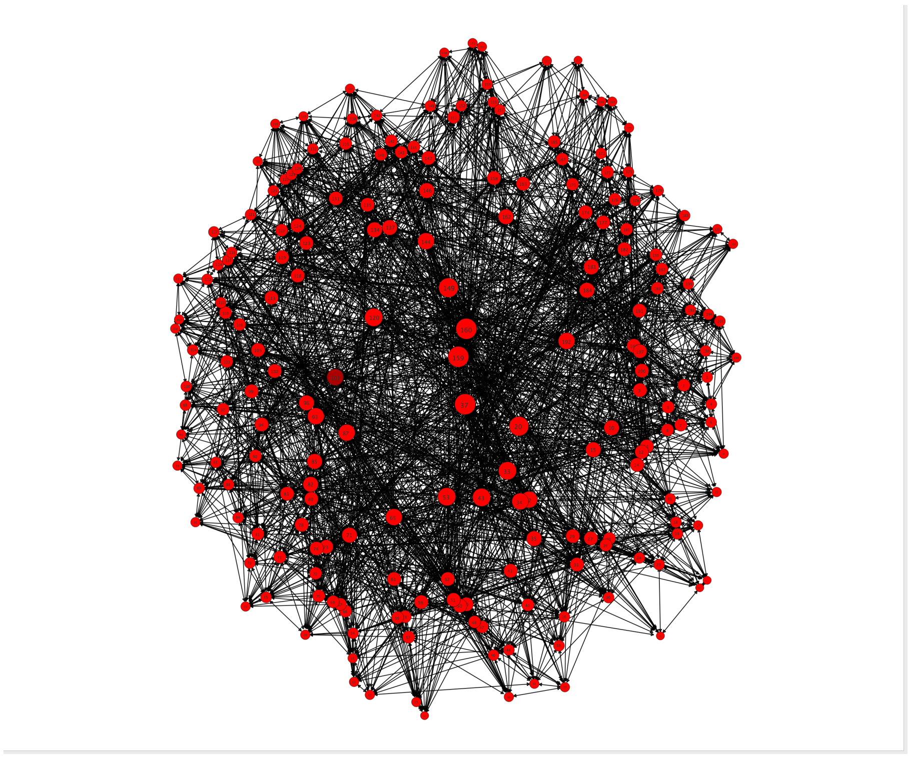


```
tuhin@cantordust [~/Desktop/plexus/dataset implementation] =>python random_graph.py
Enter number of Nodes in the Ego-Network ==> 7
Enter number of Edges in the Ego-Network ==> 10
Generating Random Ego Network........


Ego-Network Setup Progress ===> : 0it [00:00, ?it/s]node = 0 name = James Rossman
node = 1 name = Taryn Maxwell
node = 2 name = Paul Bandy
node = 3 name = Floyd Fowler
node = 4 name = Larry Durnin
node = 5 name = Mary Wilkins
node = 6 name = John Messer
Ego-Network Setup Progress ===> : 7it [00:00, 122.48it/s]


=====================================


=====================================


node =  0   name = James Rossman
 ----> Taryn Maxwell
 ----> Paul Bandy
 ----> Mary Wilkins
 ----> John Messer
node =  1   name = Taryn Maxwell
 ----> James Rossman
 ----> Paul Bandy
 ----> Floyd Fowler
node =  2   name = Paul Bandy
 ----> James Rossman
 ----> Taryn Maxwell
 ----> Larry Durnin
 ----> John Messer
node =  3   name = Floyd Fowler
 ----> Taryn Maxwell
 ----> Larry Durnin
node =  4   name = Larry Durnin
 ----> Paul Bandy
 ----> Floyd Fowler
 ----> Mary Wilkins
node =  5   name = Mary Wilkins
 ----> James Rossman
 ----> Larry Durnin
node =  6   name = John Messer
 ----> James Rossman
 ----> Paul Bandy


Random Ego Network Generated!
```

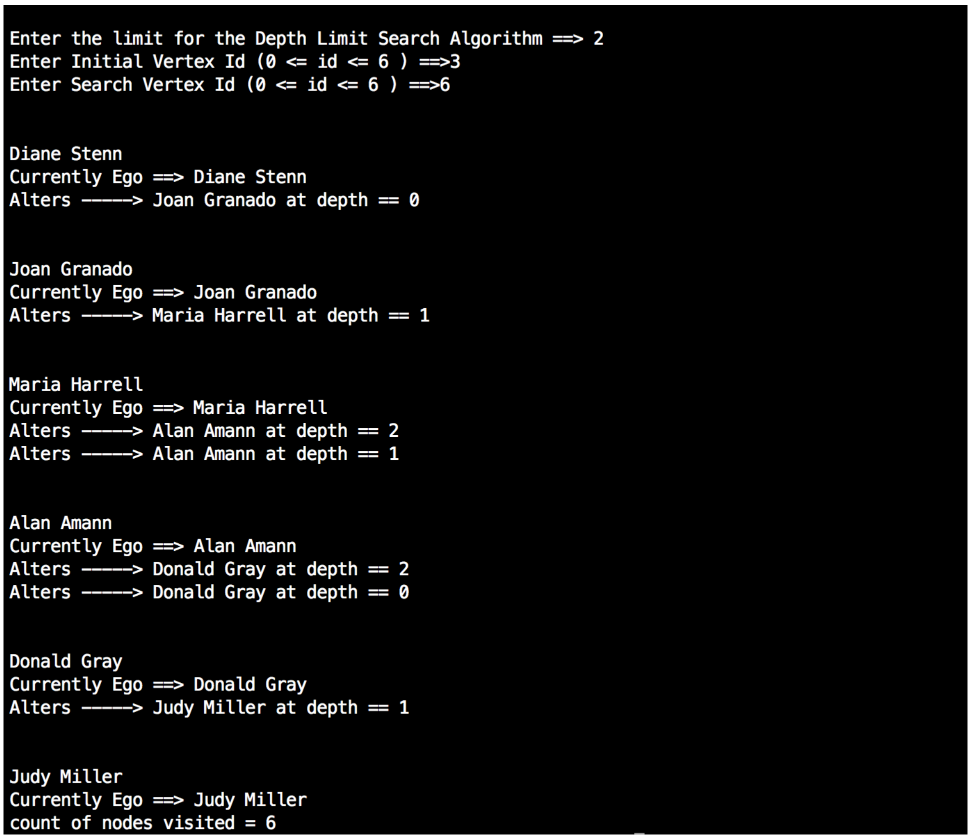
PLOTTING OF GRAPH



EGO NETWORK IN 3-D



DEPTH LIMIT SEARCH APPLIED ON EGO-NETWORK



```
Enter the limit for the Depth Limit Search Algorithm ==> 2
Enter Initial Vertex Id (0 <= id <= 6 ) ==>3
Enter Search Vertex Id (0 <= id <= 6 ) ==>6


Diane Stenn
Currently Ego ==> Diane Stenn
Alters ------> Joan Granado at depth == 0


Joan Granado
Currently Ego ==> Joan Granado
Alters ------> Maria Harrell at depth == 1


Maria Harrell
Currently Ego ==> Maria Harrell
Alters ------> Alan Amann at depth == 2
Alters ------> Alan Amann at depth == 1


Alan Amann
Currently Ego ==> Alan Amann
Alters ------> Donald Gray at depth == 2
Alters ------> Donald Gray at depth == 0


Donald Gray
Currently Ego ==> Donald Gray
Alters ------> Judy Miller at depth == 1


Judy Miller
Currently Ego ==> Judy Miller
count of nodes visited = 6
```

CONCLUSION

Depth limited search is better than depth first search in terms of the time complexity i.e time complexity for depth limited search is O(V + l) when an adjacency list is used and the complexity for depth first search is O(V+E) where, E is number of edges, V is number of vertices and l is the limit used for the depth limited search algorithm.