ISSN 2047-3338

Algorithms for a Path P_n Generated by DFS

Dr. H. B. Walikar[2], Shreedevi V. Shindhe[1], Ravikumar H. Roogi[3] and Ishwar Baidari[4]
[1,2,3,4]Department of Computer Science, Karnatak University, Dharwad, India

***Abstract*– In this paper we are dealing with some basic class of graphs that give a DFS tree which is a path consisting of all the vertices of the graph. Some of the graph classes result in $DFST(G) = P_n$ after applying some conditions and by giving proper labelling of vertices. The DFS algorithm has to be modified accordingly. We want to generate a P_n by applying DFS on the given graph G. We developed the algorithm with time complexity of $O(n^2)$.**

***Index Terms*– DFS, Algorithm, Graph and Complexity**

I. INTRODUCTION

1) Graph: A graph $G = (V, E)$ is an ordered pair of sets. Elements of V are called vertices or nodes, and elements of $E \subseteq V \times V$ are called edges or lines. We refer to V as the vertex set of G, with E being the edge set.

2) Walks, trails and paths in graphs: If u and v are two vertices in a graph G, a $u - v$ *walk* is an alternating sequence of vertices and edges starting with u and ending at v. Consecutive vertices and edges are incident. For the graph [6] in Fig. 1, an example of a walk is an a − e walk: a, b, c, b, e. In other words, we start at vertex a and travel to vertex b. From b, we go to c and then back to b again. Then we end our journey at e. Notice that consecutive vertices in a walk are adjacent to each other. One can think of vertices as destinations and edges as footpaths, say. We are allowed to have repeated vertices and edges in a walk. The number of edges in a walk is called its length. For instance, the walk a, b, c, b, e has length 4.

A trail is a walk with no repeating edges. For example, the

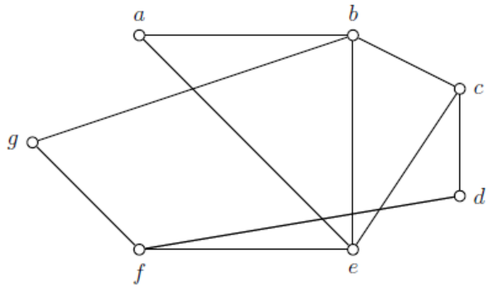


Fig. 1: Walking along a graph

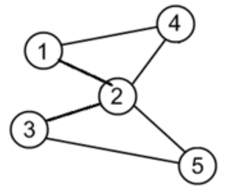
$a - b$ walk a, b, c, d, f, g, b in Fig. 1 is a trail. It does not contain any repeated edges, but it contains one repeated vertex, i.e. b. Nothing in the definition of a trail restricts a trail from having repeated vertices. Where the start and end vertices of a trail are the same, we say that the trail is a circuit, otherwise known as a closed trail. Thus the walk a, b, e, a is a circuit. A walk with no repeating vertices is called a path. Without any repeating vertices, a path cannot have repeating edges; hence a path is also a trail. A path whose start and end vertices are the same is called a cycle. For example, the walk a, b, c, e, a in Figure is a path and a *cycle*.

3) Depth First Search Algorithm (DFS): Depth-first search (DFS) is an algorithm for traversing the graph. We start the graph traversal at an arbitrary vertex and go down a particular branch until we reach a dead end. Then we back up and go as deep possible. In this way we visit all vertices, and all edges. The DFS can be used for [4].

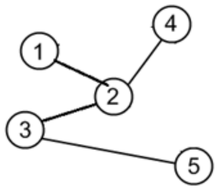
- Testing for connectivity
- Finding a Spanning Tree
- Finding Paths
- Finding a cycle

The DFS algorithm is given below [5]:

```
Void DFS(int v)
{
        u=adjacent(v);
        while(u)
      {
                If(u is not reached)
                        DFS(u);
                u=nextadjacent(v);
        }
}
```



(a) Graph G



(b)

Fig. 2: $DFST(G)$ with 1 as start vertex

On applying DFS (Depth First Search) on any graph G, it results is a tree T, called DFS Tree (DFST). This DFST can be any tree including a simple path. We get different trees with different labelling of vertices and selecting different start vertices. The resulting trees will be having different *diameters*.

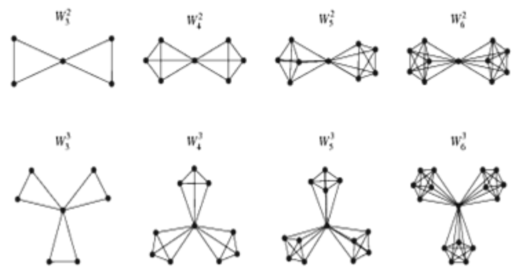


Fig. 3: Windmill graphs

II. PROPOSED WORK

Our aim is to find the class of graphs, which can result in *DFS* tree which is a path, called *dfs path*, after the application of DFS, i.e. $DFST(G) = P_n$. If such a dfs tree is a path then it will be having the highest diameter.

Some graphs will result in path by direct application of DFS where as some need modifications in the algorithm. The DFS algorithm starts with a start vertex. The start vertex which leads to a dfs path P_n, and is the end vertex of the path, then it is called *path generating vertex*. Not all graphs with result in $DFST(G) = P_n$. Some of the graphs will never give a dfs path P_n. Example: star graph, windmill graph W_x^y $x \geq 3, y \geq 3$. From Fig. 3 we can see that the first four graphs i.e., W_3^2, W_4^2, W_5^2 and W_6^2 we can get a path. For the next four wind mills i.e., W_3^3, W_4^3, W_5^3 and W_6^3 we can not get paths.

In this paper we are dealing with some basic class of graphs and their behaviour on applying DFS. Some of the graph classes result in $DFST(G) = P_n$ after applying some conditions and by giving proper labelling of vertices and modifying the DFS algorithm accordingly. We want to generate a path P_n by applying DFS on the given graph G. This problem appears as finding the Hamiltonian Path of a graph. But it is not the exact case. We are interested in dfs tree which is a P_n rather than Hamiltonian paths or longest path between two vertices. To elaborate the concept let us consider the following example.

Example: For the graph given in Fig. 4(a), when we apply

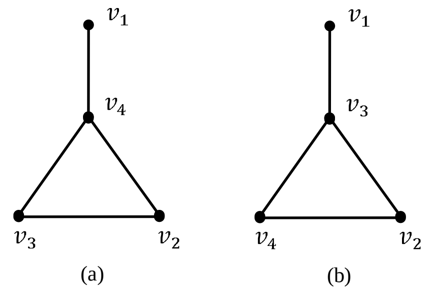


Fig. 4: An Example

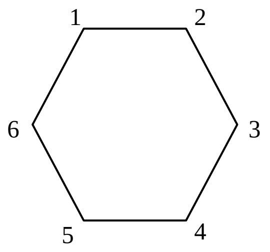


Fig. 5: The cycle C_6

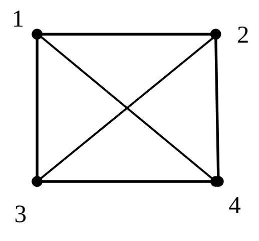


Fig. 6: Complete Graph K_4

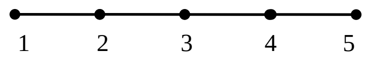


Fig. 7: path graph P_5

DFS with vertex v_1 as the start vertex then we get the path v_1, v_4, v_2, v_3. In this case the start vertex is one of the end

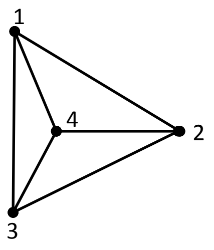


Fig. 8 : Tetrahedral Graph

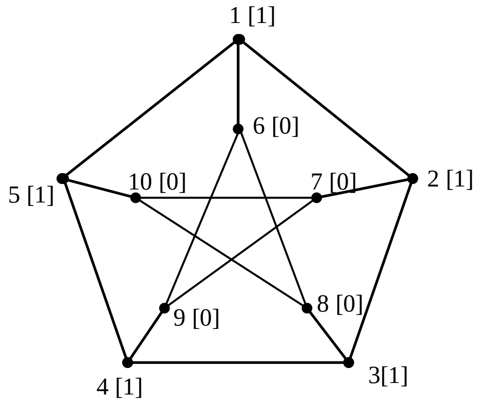


Fig. 9: Petersen Graph

vertex of the resultant dfs path. If we consider v_4 as the start vertex then the path obtained will be v_1, v_4, v_2, v_3. Here even though v_4 is the start vertex its not the end vertex of the resultant path, but still we got a path starting with vertex v_4. If we consider v_2 as tart vertex the dfs path obtained will be v_2, v_3, v_4, v_1. Now let us change the labelling of the same graph as in Fig. 4(b), and start with vertex v_2. Here the order of visiting vertices will be v_2, v_3, v_1, v_4 but it's not a path, hence v_2 is not a path generating vertex in this instance. This example clearly shows that different labelling of vertices has different effects on the resulting dfs tree.

III. GRAPH CLASSES THAT GIVE DFS PATH P_n BY APPLYING DFS WITHOUT ANY LABELLING OR CONDITIONS

1) Cycles: When G is a cycle (C_n), then applying the DFS simply gives a P_n. Of course the cycles result in P_n just by removing an edge also. But here we are studying with respect to DFS. In cycles every vertex is a path generating vertex. Example is shown in Fig. 5. When we apply DFS on this cycle with 2 as start vertex then we get the path 2,1,6,5,4,3.

2) Complete Graphs: By applying DFS on any complete graph we get a path P_n, denoted $DFST(K_n) = P_n$. For example refer Fig. 6. We can take any vertex as start vertex. Every vertex leads to a path P_n. As in cycles for compete graph also every vertex is a path generating vertex. From Fig. 6 we note that after applying DFS with 4 as start vertex the path generated is 4,1,2,3.

3) Paths: By applying DFS on any path graph P_n we get a dfs path P_n itself as result. It is denoted $DFST(P_n) = P_n$. For example refer Fig. 7. We can take any vertex as start vertex. Every vertex leads to a path P_n and hence every vertex is a path generating vertex.

4) Tetrahedral Graph: The Tetrahedral Graph is one of the platonic solids. On applying DFS algorithm the Tetrahedral Graph results into a dfs path P_n. Here also every vertex is path generating vertex. It can be seen in Fig. 8.

IV. GRAPH CLASSES THAT YIELD DFS PATH P_n BY PROPER LABELLING

1) Petersen Graph: In this section we will study about how the labels can be provide for the vertices of the graph so that the vertices are visited in a specific order. For some class of the graphs we can obtain $DFST(G) = P_n$by labelling vertices and modifying the DFS algorithm accordingly.

For example let us consider Petersen graph. Let us label the vertices in two ways as *outer* and *inner* vertices. We have to maintain an integer array to specify the outer vertices and inner vertices. From figure we observe that the vertices 1,2,3,4,5 are outer vertices and remaining are inner vertices. Let our array name be $int\ outer[] = \{1,1,1,1,1,0,0,0,0,0\}$. The outer vertices are marked as 1 and inner vertices as 0. The labelling of vertices is shown in square brackets.

With this labelling we can modify DFS algorithm to get dfs path. Method is very simple after labelling. We just have to start with a vertex, if the start vertex is labelled as outer i.e.,1, then first visit all outer vertices, then visit all the inner vertices by following the adjacency, and vice versa. For instance in this Petersen graph, we can select the vertex 5 as the start vertex. Then visit all vertices which are outer, then the sequence will be 5,1,2,3,4. After vertex 4 there are no more outer vertices to visit. Therefore visit the first inner vertex i.e.,9 and continuously visit all inner vertices to get the path 5,1,2,3,4,9,7,10,8,6.

This method is expressed in the form of algorithm as follows.

ALGORITHM 1:

1. For all the vertices of the graph label them as *inner* and *outer* vertices.
2. Let any vertex be a start vertex.
3. If the start vertex is inner then visit all the inner vertices first. Later visit all the outer vertices and vice versa, by modifying the DFS algorithm.
4. Store the path in an array.

2) Generalised Petersen Graphs: Now let us consider Generalised Petersen Graphs $G(n,k)$. In case of Generalised Petersen Graphs the previous method works well if n is odd. The labelling method remains the same. If n is even the inside vertices will form two cycles and this has to be dealt in different way.

The following example shows the difference of n being even and odd. Let us consider $G(6,2)$. Here if the start vertex

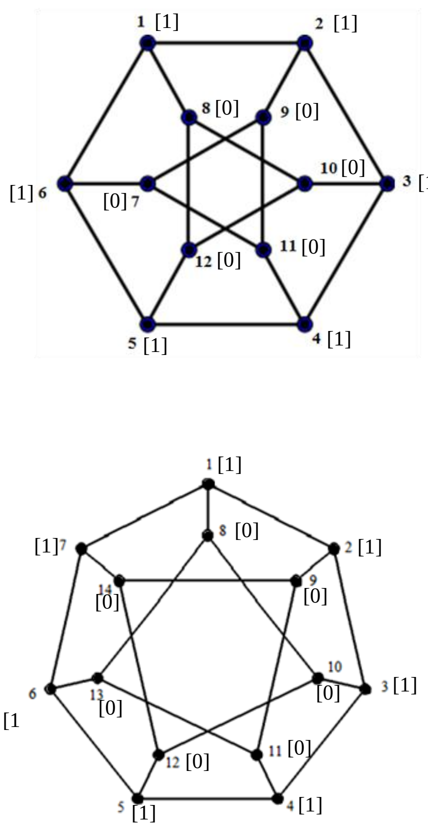


Fig. 10: $G(6,2)$ and $G(7,2)$

is an inner vertex then the inner vertices are visited first, but not the all inner vertices can be visited. Then all the outer vertices visited and then the remaining inner vertices. Since we are using recursive DFS it can be traced easily. On the other hand if the start vertex is the outer vertex then all the outer vertices are visited first then some of the inner vertices and lastly by visiting back the start vertex other inner vertices are visited. Form the graph $G(6,2)$ we see that if the vertex 1 is taken as start vertex then the order of visiting vertices is 1,2,3,4,5,6,7,9,11. No other inner vertex is adjacent to 11 so the control goes back to vertex 1 as 11,9,7,6,5,4,3,2,1. Now from 1, the next vertex visited are 1,8,10,12. Now all the vertices are visited and the path obtained is 12,10,8,1,2,3,4,5,6,7,9,11. It can be observed that the start vertex is not the end vertex of the path as in case if n was odd. We see that if the start vertex is an inner vertex then it becomes the end vertex and hence the path generating vertex too.

Consider the graph $G(7,2)$, here we can start with any start vertex say 1. Then visiting all outer vertices and later the inner vertices we get the path 1,2,3,4,5,6,7,8,9,10,11,12,13,14. We see that the start vertex is the end vertex of the path obtained; hence it's a path generating vertex.

The algorithm combining for both odd and even vertices is given as follows. The algorithm takes $O(n^2)$ time if adjacency matrix is used.

The steps are given as follows:

1. a. If the vertex v is an outer vertex, find the outer vertex u adjacent to v.
   b. Increment the path length.
   c. Call DFS recursively by passing u as the parameter, as long as u is not 0.
2. a. If the vertex v is an inner vertex, find the inner vertex u adjacent to v.
   b. Increment the path length.
   c. Call DFS recursively by passing u as the parameter, as long as u is not 0.
3. Repeat the steps 1 or 2 till path equals $n-1$.

The algorithm is given as follows:

ALGORITHM 2:

// o[]: array for labelling vertices as inner and //outer // vertices. If a vertex v is outer vertex the //o[v]=1 else // o[v]=0.
// outadjacent(v): finds the outer vertex adjacent //to v.
// inadjacent(v): finds the inner vertex adjacent //to v.

```
void DFS( int v )
{
    if(o[v]==1)
    {
        u=outadjacent(v);
        if(u!=0)
        {
            path++;
            DFS(u);

        }
        else
        {
            u=inadjacent(v);
            if(u!=0 && path<n-1)
            {
                path++;
                DFS(u);
            }
        }
 }
else
{
    if(o[v]==0)
        {
            u=inadjacent(v);
            if(u!=0)
            {
                path++;
                DFS(u);
            }
            else
            {
                u=outadjacent(v);
        if(u!=0 && path<n-1)
        {
            path++;
            DFS(u);
        }
    }
```

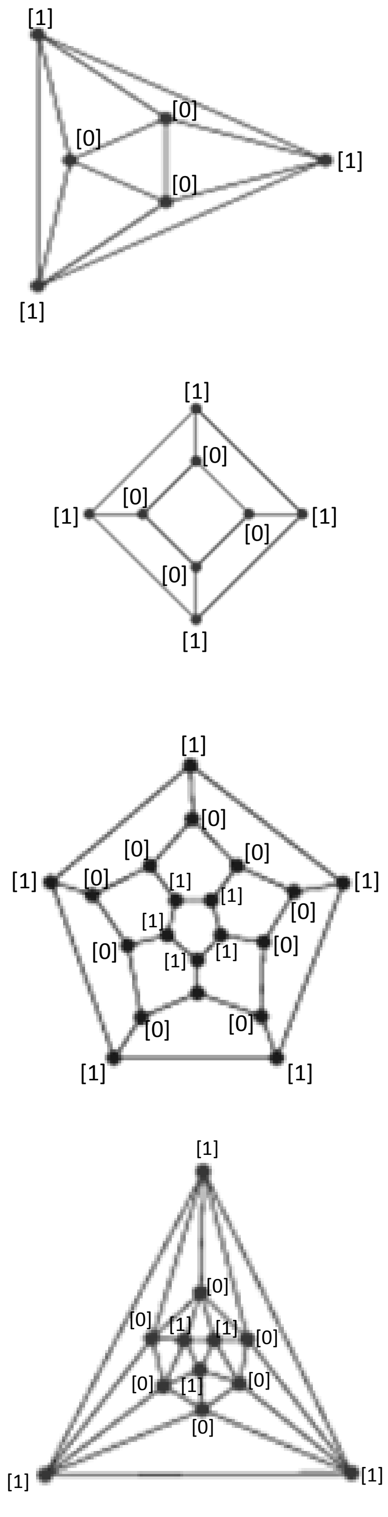


Fig. 11: The platonic graphs

```
        }
    }
}
```

3) Platonic Solids: The five platonic solids are tetrahedral graph, cubical graph, octahedral graph, dodecahedral graph and icosahedral graph. Among these, the tetrahedral graph gives a path by simple DFS as seen in Fig. 8. Cubical graph and octahedral graph give a path by using the algorithm given for Petersen graph. The next two platonic solids i.e., dodecahedral graph and icosahedral graph can not give a path by using algorithm 1 or algorithm 2. We have to modify algorithm 2 as given in algorithm 3. The complexity remains the same.

The modified code is marked grey.

ALGORITHM 3:

// o[]: array for labelling vertices as inner and //outer // vertices. If a vertex v is outer vertex the //o[v]=1 else // o[v]=0.
// outadjacent(v): finds the outer vertex adjacent //to v.
// inadjacent(v): finds the inner vertex adjacent //to v.

```
void DFS( int v )
{
    if(o[v]==1)
    {
        u=outadjacent(v);
        if(u!=0 && path<n-1)
        {
            path++;
            DFS(u);
            if(v==source && path>0 && path<n-1)
            {
                u=inadjacent(v);
                DFS(u);
            }
        }
        else
        {
            u=inadjacent(v);
            if(u!=0 && path<n-1)
            {
                path++;
                DFS(u);
             }
        }
    }
    else
    {
        if(o[v]==0)
        {
            u=inadjacent(v);
            if(u!=0 && path<n-1)
            {
                path++;
                DFS(u);
            if(v==source && path>0 && path<n-1)
            {
                u=outadjacent(v);
                DFS(u);
            }
        }
        else
        {
            u=outadjacent(v);
            if(u!=0 && path<n-1)
            {
                path++;
                DFS(u);
            }
        }
    }
  }
}
```

4) Wheel Graphs: In the mathematical discipline of graph theory, a wheel graph W_n is a graph with n vertices, formed by connecting a single vertex (which is known as the hub) to all vertices of an $(n-1)$-cycle. The edges of a wheel which include the hub are called spokes.

For these graphs the algorithm 1-3 can be used to form the path. The labeling for hub is [0] and labeling for all spokes is [1] or vice versa. Here the start vertex and hub both can be source generating vertices.

The wheel graphs give path without modifying DFS algorithm and without any labeling only if the start vertex is taken as the hub.

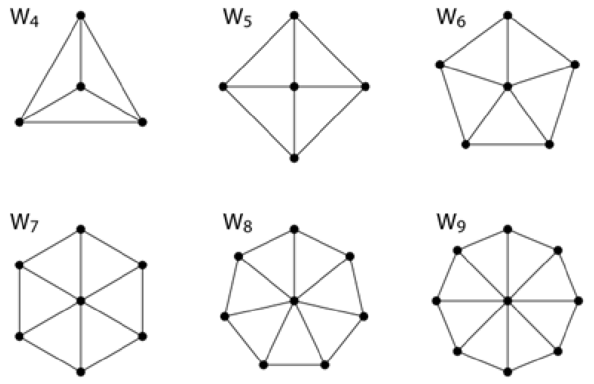


Fig. 12: Wheel Graphs

V. CONCLUSIONS

One of the basic applications of DFS is to find the paths. Here we are finding a path consisting of all the vertices of the given graph. The given algorithm takes $O(n^2)$ time if adjacency matrix is used. Other data structures may be used to reduce the complexity further.

Only few classes of graphs have been tested here. We can consider some more class of graphs for further analysis.

REFERENCES

[1] A. V. Goldberg and C. Silverstein. Implementations of Dijkstra's Algorithm Based on Multi-Level Buckets. In P. M. Pardalos, D.W. Hearn, and W.W. Hages, editors, *Lecture Notes in Economics and Mathematical System 450 (Refereed Proceedings)*, pages 292-327. Springer Verlag, 1997.

[2] Anany Levitin, Introduction to the design and analysis of Algorithms, 2nd Ed, 2008.

[3] Boris V. Cherkassky , Andrew V. Goldberg , Craig Silverstein, Buckets, Heaps, Lists, and Monotone Priority Queues, SIAM Journal on Computing, v.28 n.4, p.1326-1346, Aug. 1999.

[4] Coremen. T. H, Leiserson, C. E. Rivest R. L, and C Stein, Introduction to Algorithms, 2nd Ed. MIT press, Cambridge, MA, 2001.

[5] Data Structures, Algorithms, and Applications in C++, Sartaj Sahni. McGraw-Hill Education, 1998.

[6] Distance in graphs - Fred Buckley, Frank Harary, Addison-Wesley Pub. Co., ©1990 .

[7] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numer. Math.*, 1:269-271, 1959.

[8] Harary. F, Graph Theory, Addison Wesley.

[9] R.W. Floyd. Algorithm 97: Shortest path. *Comm. ACM*, 5:345, 1962.

[10] Tarjan, R. E. (1972), "Depth-first search and linear graph algorithms", SIAM Journal on Computing 1 (2): 146–160.