Given a graph G, for a given graph G1 check for associated digraph

Pradeep Gangwar
IHM2016501

Rahul Chanderiya
IIM2016002

Aditi Agrawal
IIM2016001

October 5, 2019

Abstract

The paper details the algorithm to check for isomorphism of two given graphs. For a given graph G, and a given graph G1 we have to check for associated digraph.

Keywords :- *Adjacency Matrix, Adjacency List, Undirected Graph, Directed Graph, Digraph, Degree of graph*

1 Introduction

Graphs are mathematical structures that represent pairwise relationship between objects. They perform important role in modeling data. They are useful in solving many real life applications such as social networks, transportation networks, document-link graphs and many more. Graph theoretical ideas are highly utilized by computer science applications. Especially in research areas of computer science such data mining, image segmentation, clustering, image capturing, networking etc. Let us introduce the important notations and definitions that we will use frequently in this paper.

1.1 Basic Definitions

1. *Graph:* Graph is a pair G=(V,E) where,
   - V is a set of vertices (or nodes), and
   - E $\subseteq$ (VxV) is a set of edges.

   A graph may be directed or undirected, if graph is undirected, then adjacency relation defined by edges is symmetric.

2. *Directed Graph:* A directed graph is graph, i.e., a set of objects (called vertices or nodes) that are connected together, where all the edges are directed from one vertex to another. A directed graph is sometimes called a digraph or a directed network. In contrast, a graph where the edges are bidirectional is called an undirected graph.

3. *Undirected Graph:* An undirected graph is graph, i.e., a set of objects (called vertices or nodes) that are connected together, where all the edges are bidirectional. An undirected graph is sometimes called an undirected network.

4. *Isomorphic Graph:* Two graphs G1 and G2 are said to be isomorphic if −
   - Their number of components (vertices and edges) are same.
   - Their edge connectivity is retained.

   In short, out of the two isomorphic graphs, one is a tweaked version of the other. An unlabelled graph also can be thought of as an isomorphic graph.

1.2 Objective

Here we will propose algorithms to find when a given graph G and G1, whether G1 is an associated digraph of G i.e. $G1 = D(G)$. In short we have to find whether two graphs are isomorphic or not given two combinations of graph.

2 Algorithm Design

Here we are designing an algorithm, that would check whether the two given directed/undirected/mixed graphs are equal(isomorphic) or not. Two Graphs in the form of adjacency matrix would be given as input, which would be analysed for the similarities between them. In the end the output would be Boolean value with true emphasising the graphs are equal and false for unequal graphs.

2.1 Algorithm Implementation

(i) Firstly, we will scan the adjacency matrix of both the graphs and calculate the degree of each vertex in the graphs

(ii) For calculating degree of the vertices, we would count the total number of outgoing edges from the given vertex.

(iii) Now we will have the base case check, if the total number of vertices with the given degree are not same in both the graphs then the graphs are not equal (isomorphic).

(iv) After having the base check, now we will compare the edges of the graphs.

(v) For comparing edges, firstly we will iterate all the edges of the first graph. We would find the degree of the connecting vertices of the edge and insert it into the map with pair of vertices degree as the key and count of the edges encountered as value.

(vi) Now we will iterate over the edges of the second graph.

(vii) Similar to step (iv) we would decrement the count value for the key encountered corresponding to given edge in the map.

(viii) Finally we will iterate over our map and check whether all the count value for given key-value pair is zero or not.

(ix) If for any key-value pair the count value is found not to be zero, then the given pair of graphs are not equal, else they are equal.

2.2 Pseudo Code

Algorithm 1 trivial_check()

```
if g1.size() != g2.size() then
    return false
end if
map<int, int> m1, m2
for itr=deg_v1.begin to deg_v1.end do
    increment m1[itr->second]
end for
for itr=deg_v2.begin to deg_v2.end do
    increment m2[itr->second]
end for
for itr=m1.begin to m1.end do
    if m2.find(itr->first) == m2.end() then
        return false
    end if
    if m2[itr->first] != itr->second then
        return false
    end if
end for
return true
```

Algorithm 2 assign_degree()

```
for i=0 to v do
    $c \leftarrow 0$
    for j=0 to v do
        if g1[i][j] equals 1 then
            increment c
        end if
    end for
    deg_v1[i] = c
end for
for i=0 to v do
    $c \leftarrow 0$
    for j=0 to v do
        if g2[i][j] equals 1 then
            increment c
        end if
    end for
    deg_v2[i] = c
end for
```

Algorithm 3 compare_edges()

```
map<tuple<int,int>,int> edge_deg

for i=0 to v do
    for j=0 to v do
        if g1[i][j] equals 0 then
            continue
        else
            $x \leftarrow deg\_v1[i]$
            $y \leftarrow deg\_v1[j]$
            $edge\_deg[make\_tuple(x, y)] + +$
        end if
    end for
end for
for i=0 to v do
    for j=0 to v do
        if g2[i][j] equals 0 then
            continue
        else
            $x \leftarrow deg\_v2[i]$
            $y \leftarrow deg\_v2[j]$
            $edge\_deg[make\_tuple(x, y)] - -$
        end if
    end for
end for
for itr=edge_deg.begin to edge_deg.end do
    if itr->second != 0 then
        return false
    end if
end for
```

Algorithm 4 public:cls(g1, g2)

```
1: $g1 \leftarrow g1\_temp$
2: $g2 \leftarrow g2\_temp$
3: $v \leftarrow g1.size()$
4: $assign\_degree()$
5: $ans \leftarrow trivial\_check()$
6: if ans equals false then
     return
7: end if
8: $ans \leftarrow compare\_edges()$
9: return
```

3 Analysis

3.1 Time Complexity Analysis

The time complexity of the given algorithm is constant for the best case and quadratic for the average and worst case with respect to total number of vertices present in the graph. This is evident from the fact that :-

(i) For the best case, if the total number of vertices in both the graph are unequal then we could clearly tell that the graphs are unequal. Thus this result in complexity of $O(1)$ in best case scenario.

(ii) For all the other cases we have to find the degree of each vertex of both the graph.

(iii) As to find the degree of each vertex in a graph we have to scan the adjacency matrix once, thus it result in $O(V^2)$ time complexity in average and worst case.

(iv) After finding the degree of each vertices of both the graph, we have to compare all the edges between both the graphs, thus this result in additional $O(E)$ complexity.

(v) Therefore the final complexity turns out to be $O(V^2) + O(E)$ which is equal to $O(V^2)$.

Best Case :

$time_{best} = O(1)$

Average case / Worst case :

$time_{average} = time_{worst} = O(V^2)$
where V is the total no. of vertices in the graph.

3.2 Space Complexity Analysis

Our algorithm requires an auxiliary space to store the degree of each vertex of both the graphs. Therefore the space complexity of our algorithm is linear in nature.

$space_{complexity} = O(V)$
where V is the total no. of vertices in the graph.

4 Experimental Study

4.1 Profiling

The best way to study an algorithm is by graphs and profiling.

4.1.1 Time Complexity

We have seen that
$time_{average} = time_{worst} = O(V^2)$
$time_{best} = O(1)$
So,the graphs for average and worst case time complexity remain the same.

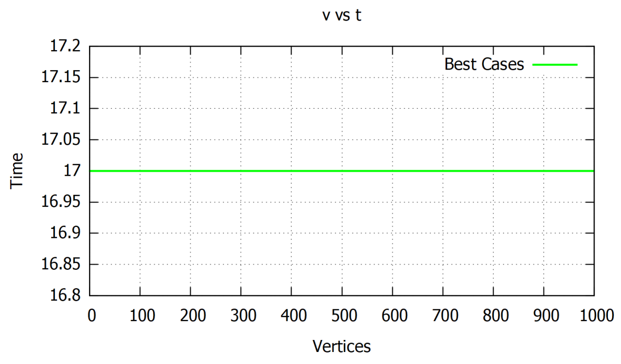

Figure 1: V vs T graph(Best case)

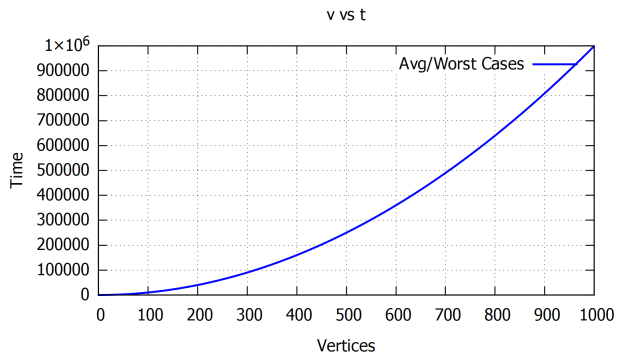

Figure 2: V vs T graph(Average/Worst case)

4.2 Sample Outputs

As the best way to understand something is through the working examples. Therefore this section consist of some examples to understand our algorithm better.

$$G = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$G1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Output: True

Output gives a boolean value of whether graphs are isomorphic or not.

5 Discussions

5.1 Applications

In computer science graphs are used to represent flow of data. Graphs have many real world applications. We use graphs to represent networks and use its algorithms to find optimal graphical solutions. Google maps uses graph theory to find various distances and suggest travelling routes. Graphs are also used by many image processing and machine learning libraries.

Isomorphism in graphs holds many applications in the following areas:

- Image processing
- Protein structure
- Computer design systems
- Chemical structures
- Social Networks

6 Conclusion

The main aim of this paper is to check if the given graphs are isomorphic. The main challenge is to compare one un-directed and given directed graph. For this we convert un-directed graph to directed by replacing its edges by two opposite bi-directional edges. The time-complexity as well as space complexity of the algorithm has also been shown in this paper. Furthermore, an abstract idea of how it can be used by the researches to solve different problems in real life is also presented by the paper.

References

[1] Keijo Ruohonen, Graph Theory, 2010,PP. 27-35

[2] Thomas H. Cormen is the co-author of Introduction to Algorithms

[3] Jonathan L. Gross and Jay Yellen, Graph Theory and its applications, Columbia University, New York, USA, 2005

[4] A. Gibbons, Algorithmic Graph Theory, Cambridge University Press, 2003.

[5] C. Godsil and G.Royle, Algorithmic Graph Theory, Cambridge University Press, 2001