Partiteness of a Graph

Lakshmi S
Namitha R H
Santrupti Nerli
Under the guidance of Prashantha Naduthota

06 November 2009

Contents

Abstract- *Partiteness gives whether a given graph is n-partite, bipartite or non-partite. The report includes the analysis of the time complexity of the graph problem. The graph coloring methodology is used to ascertain the kind of partite graph.*

0.1 Introduction

A problem can be well understood if it is pictorially represented. Graphs can be effectively used to depict real world issues. Solutions to graph problems are usually heuristic in nature. They can be optimized using few algorithms. However for few cases they can be disastrous. One of the techniques used to determine the partiteness of a graph is graph coloring. The graph coloring module decides the efficiency of the program.

0.2 Analysis of Code

The code generates random number of vertices if not fed by the user. The generation function is used to generate the random adjacency matrix. The matrix obtained is passed to the degree module which finds out the degree of each vertex. The array holding the degrees of the vertices along with the adjacency matrix is passed to the coloring module.

In the coloring module, the vertices are sorted according to the degree and is pushed onto the stack with the vertex having the highest degree at the top of the stack. The first vertex is assigned a color and is popped. The vertices that are not connected to the popped vertex are put into a separate array. The existance of the edge is checked between each set of vertices in the array. If there are any interconnected vertices then those vertices are left uncolored. The remaining vertices are colored with the same and are popped out from the stack. This process is executed till the stack becomes empty. The number of colors used determines whether the given graph is bipartite,n-partite or non partite.

0.3 Test Case

Here are few test cases for the code. Let us consider 4 vertices.

Case 1:Adjacency matrix
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

deg[0]:0
deg[1]:0
deg[2]:0
deg[3]:0

colors required:1

Therefore it is a nonpartite graph.
Lines executed:73.24% of 142
Case 2:Adjacency matrix
0 1 0 0
1 1 0 1
0 0 0 0
0 1 0 0

Coloring is not done because the there is a loop.

Therefore the input is invalid.
Lines executed:16.90% of 142
Case 3: Adjacency matrix
0 1 0 0
0 0 1 1
1 1 0 0
0 1 1 0

Coloring is not done because the adjacency matrix is not symmetric.

Therefore the input is invalid.
Lines executed:19.01% of 142
Case 4: Adjacency matrix
0 0 1 1
0 0 0 1
1 0 0 0
1 1 0 0

deg[0]:2
deg[1]:1
deg[2]:1
deg[3]:2

colors required:2

Therefore it is a bipartite graph.
Lines executed:72.54% of 142
Case 5: Adjacency matrix
0 0 1 1
0 0 0 1
1 0 0 1
1 1 1 0

deg[0]:2
deg[1]:1
deg[2]:1
deg[3]:3

colors required:3

Therefore it is 3-partite.
Lines executed:70.42% of 142
Case 6: Adjacency matrix
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0

deg[0]:3
deg[1]:3
deg[2]:3
deg[3]:3

colors required:4

Therefore it is 4-partite.
Lines executed:59.86% of 142

0.4 Random input

The input is generated in such a way that the diagonal elements of the adjacency matrix are always zero. It is because the graph cannot have loops. Secondly the adjacency matrix should always be symmetric because directed graphs are not considered here.
For a random generated graph with 21 vertices, the coverage of branches in the code is 66.90% out of 142 lines.

0.5 Verification

The colored array, adjacency matrix and the number of vertices are passed to the verification module which checks if connected vertices are of the same color. If the above case is true then it reports an error. Else it indicates that the verification is successful.

0.6 Time Complexity

Time complexity of the code is the time taken by the part of the code that takes maximum time to execute. Since the coloring function takes more time than other functions, the complexity depends on it i.e,

$$T(n) = T_{color}(n)$$

Let us analyse the time complexity of the coloring function. The coloring function includes sorting and assigning colors. T_{sort} is the time taken to sort the vertices according to the degree and T_{assign} is the time taken to assign colors to the vertices.

$$T_{color} = T_{sort} + T_{assign}$$

n represents the number of vertices and m represents the count of the vertices that are not connected to the popped vertex. And hence the complexity is calculated as shown below for T_{color}

$$T(n) = \sum_{i=0}^{n} \sum_{j=0}^{n} (1) + \sum_{i=0}^{n} \sum_{j=0}^{n-m} \sum_{k=j+1}^{n-m} (1)$$
$$T(n) = n(n) + \frac{1}{2}(n-m)(n-m+3)(n+1)$$
$$T(n) = \Theta(n^2) + \Theta(n^3)$$
$$T(n) \in \Theta(n^3)$$

0.7 Graph

The graph plotted gives the time taken by the code to compute the coloring module for different number of vertices. The adjacency matrix for different number of vertices is generated randomly and time taken to color is found out for each case.

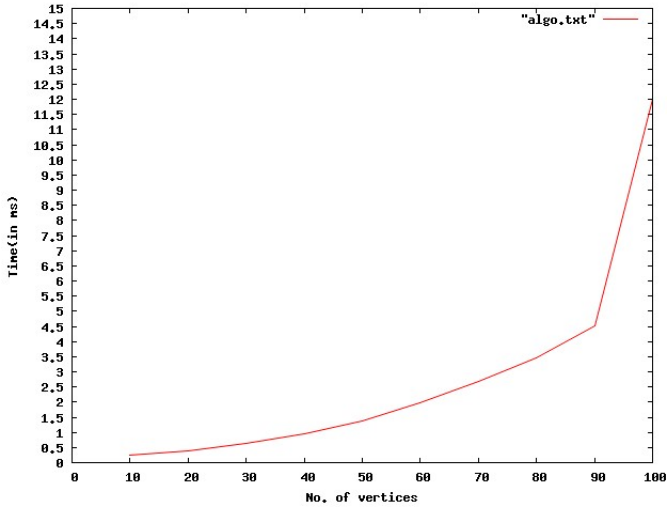

Figure 1: Graph Plot of coloring module with different set of inputs

0.8 Conclusion

It is already calculated that the average case running time for the algorithm is $\Theta(n^3)$. However the partiteness of a graph for large number of vertices can be deciphered by various other algorithms.

0.9 References

- System Requirement Specification.
- Design Document and Code.
- Anany Levitin:Introduction to the Design and Analysis of algorithms.2nd edition
- *http* : *//scienceblogs.com/goodmath/2007/06/graph_coloring_algorithms_1.php*