International Journal of Mathematical, Engineering, Biological and Applied Computing, 2022, 1, 448
www.scipublications.org/journal/index.php/ijmebac
DOI: 10.31586/ijmebac.2022.448

Short Communications & Source Code

Check if a Graph is Bipartite or not & Bipartite Graph Coloring using Java

Raja Marappan [1,*], Sarveshvara Raja [2], Shakthigaa Raja [3], Saraswatikaniga Raja [4]

[1] School of Computing, SASTRA Deemed University, Thanjavur, India
[2] Independent Researcher; raja.sarveshvara@gmail.com
[3] Independent Researcher; raja.shakthigaa@gmail.com
[4] Independent Researcher; r.saraswatikaniga@gmail.com

*Correspondence: Raja Marappan (raja_csmath@cse.sastra.edu, professor.m.raja@gmail.com)

Abstract: Nowadays, graphs including bigraphs are mostly used in various real-world applications such as search engines and social networks. The bigraph or bipartite graph is a graph whose vertex set is split into two disjoint vertex sets such that there is no edge between the same vertex set. The bipartite graphs are colored using only two colors. This article checks if a given graph is bipartite or not and finds the color assignments of the bipartite graph using Java implementation.

Keywords: Bigraphs, Bipartite Graphs, Graph Coloring, Social Networks, Color Assignments, Vertex Coloring, Search Engines

How to cite this paper:
Marappan, R., Raja, S., Raja, S., & Raja, S. (2022). Check if a Graph is Bipartite or not &amp; Bipartite Graph Coloring using Java. *International Journal of Mathematical, Engineering, Biological and Applied Computing*, 1(2), 61–67. Retrieved from https://www.scipublications.com/journal/index.php/ijmebac/article/view/448

Received: August 16, 2022
Accepted: September 16, 2022
Published: September 18, 2022

1. Introduction

A bigraph or bipartite graph G is a graph whose vertex set V(G) can be partitioned into two disjoint vertex sets such that there is no edge between the same vertex set [1, 2]. These graphs are said to be 2-colorable graphs that can be colored using only two colors [3]. This article checks if a given graph is bipartite or not using Java implementation. The complete bipartite graph or biclique is a graph in which every vertex in one set is incident with every vertex in the other set. The bipartite graph is shown in Figure 1. The complete bipartite graphs are shown in figures 2 and 3 [4, 5]. The bipartite graphs are applied in social networks and search engines. These graphs are also used to represent the binary relations between two object types [6, 7].

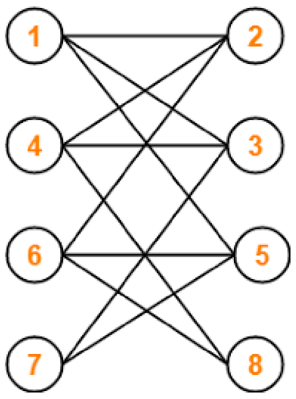


Figure 1. Bipartite graph

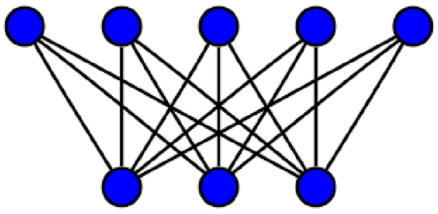


Figure 2. Complete bipartite graph

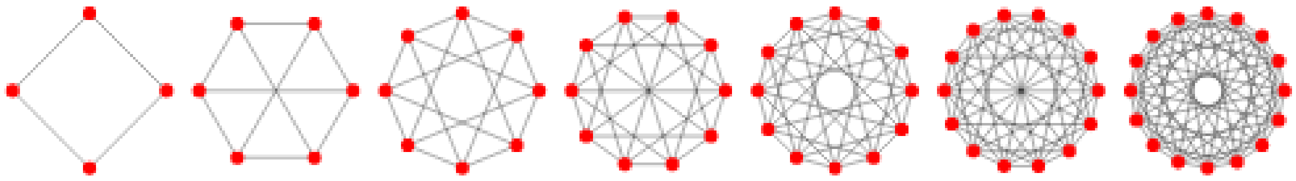


Figure 3. Complete bipartite graphs

2. Implementation in Java

The java implementation to check if the given graph is bipartite or not is given below. The color assignment of the bipartite graph is also printed.

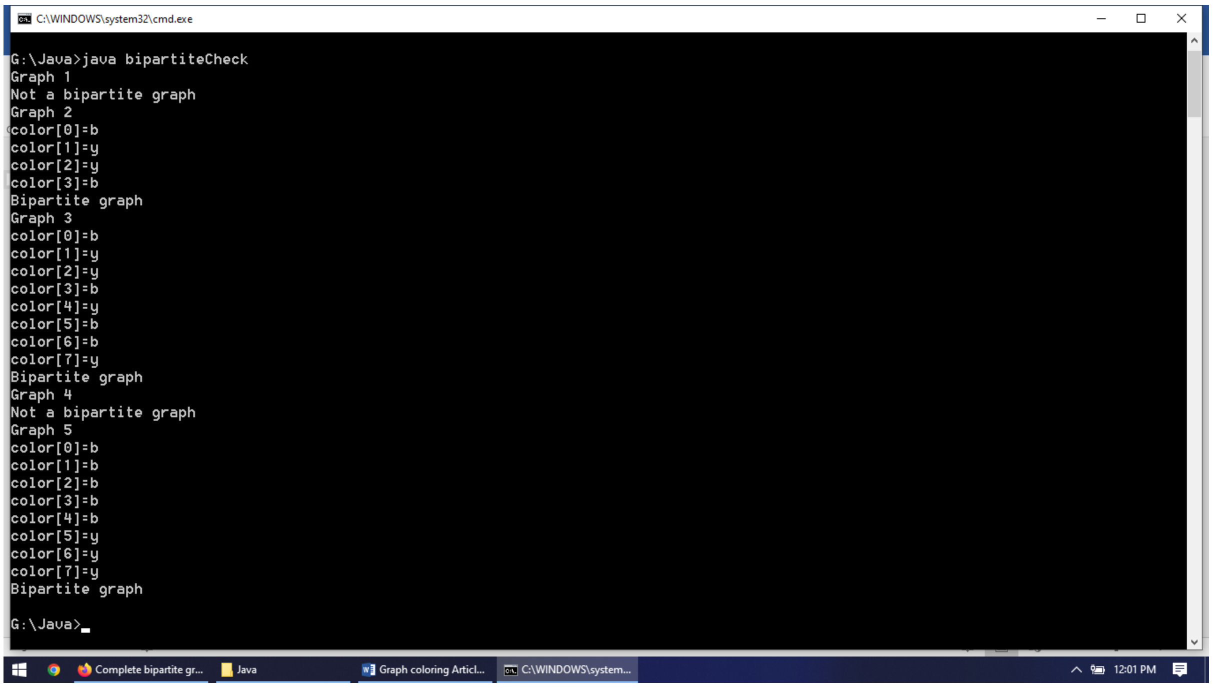
```
import java.util.*;
public class bipartiteCheck
{
    static List<List<Integer>> adjList;
    static void edgeAddition(int a, int b)
   {
    adjList.get(a).add(b);
    adjList.get(b).add(a);
   }
static boolean bipartiteCheckEx(int n)
{
// not colored – ‘n’
// colored blue – ‘b’
// colored yellow – ‘y’
char colorAssignment[] = new char[n];
for (int j = 0; j < n; j++)
            colorAssignment[j] = 'n';
Queue<Integer> queue = new LinkedList<>();
queue.add(0);
colorAssignment[0] = 'b';
while (!queue.isEmpty())
{
int h = queue.poll();
char ch = colorAssignment[h];
for(int g : adjList.get(h))
{
   if(colorAssignment[g] == ch) return false;
   if(colorAssignment[g] == 'n')
  {
        if(ch == 'b')
                         colorAssignment[g] = 'y';
                     else
                         colorAssignment[g]   = 'b';
        queue.add(g);
                   }
               }
          }
    for (int j = 0; j < n; j++)
              System.out.println("color["+j+"]="+colorAssignment[j]);
    return true;
    }
    public static void main(String ax[]){
           int n = 4;
           adjList = new ArrayList<>();
           for(int j = 0; j < n; j++)
               adjList.add(new ArrayList<>());

           edgeAddition(0, 1);
           edgeAddition(0, 2);
           edgeAddition(0, 3);
           edgeAddition(1, 3);
           edgeAddition(2, 3);
           System.out.println("Graph 1");
           if(bipartiteCheckEx(n))
               System.out.println("Bipartite graph");
           else
               System.out.println("Not a bipartite graph");

           adjList = new ArrayList<>();
           for(int i = 0; i < n; i++)
               adjList.add(new ArrayList<>());

           edgeAddition(0, 1);
           edgeAddition(0, 2);
           edgeAddition(1, 3);
           edgeAddition(2, 3);
           System.out.println("Graph 2");

           if(bipartiteCheckEx(n))
               System.out.println("Bipartite graph ");
           else
               System.out.println("Not a bipartite graph");

           n = 8;
           adjList = new ArrayList<>();
           for(int j = 0; j < 8; j++)
               adjList.add(new ArrayList<>());

           edgeAddition(0, 1);
           edgeAddition(0, 2);
           edgeAddition(0, 4);
           edgeAddition(3, 1);
```

```
edgeAddition(3, 2);
edgeAddition(3, 7);
edgeAddition(5, 1);
edgeAddition(5, 4);
edgeAddition(5, 7);
edgeAddition(6, 2);
edgeAddition(6, 4);
edgeAddition(6, 7);
System.out.println("Graph 3");

if(bipartiteCheckEx(n))
    System.out.println("Bipartite graph ");
else
    System.out.println("Not a bipartite graph");
n = 8;
adjList = new ArrayList<>();
for(int j = 0; j < 8; j++)
    adjList.add(new ArrayList<>());

edgeAddition(0, 1);
edgeAddition(0, 2);
edgeAddition(0, 4);
edgeAddition(3, 1);
edgeAddition(3, 2);
edgeAddition(3, 7);
edgeAddition(3, 5);
edgeAddition(5, 1);
edgeAddition(5, 4);
edgeAddition(5, 7);
edgeAddition(6, 2);
edgeAddition(6, 4);
edgeAddition(6, 7);
edgeAddition(1, 7);
System.out.println("Graph 4");

if(bipartiteCheckEx(n))
    System.out.println("Bipartite graph ");
else
    System.out.println("Not a bipartite graph");

n = 8;
adjList = new ArrayList<>();
for(int j = 0; j < 8; j++)
    adjList.add(new ArrayList<>());

edgeAddition(0, 5);
edgeAddition(0, 6);
edgeAddition(0, 7);
edgeAddition(1, 5);
edgeAddition(1, 6);
edgeAddition(1, 7);
edgeAddition(2, 5);
edgeAddition(2, 6);
```

```
            edgeAddition(2, 7);
            edgeAddition(3, 5);
            edgeAddition(3, 6);
            edgeAddition(3, 7);
            edgeAddition(4, 5);
            edgeAddition(4, 6);
            edgeAddition(4, 7);
            System.out.println("Graph 5");

            if(bipartiteCheckEx(n))
                System.out.println("Bipartite graph ");
            else
                System.out.println("Not a bipartite graph");
        }
    }
```

3. Results

The execution and the result of the Java program is shown in Figure 4. The graphs 1, 2, 3, 4 and 5 are given as the inputs and the results are shown in figures from 5 to 9 respectively. The program outputs the color assignments of the bipartite graphs.



```
G:\Java>java bipartiteCheck
Graph 1
Not a bipartite graph
Graph 2
color[0]=b
color[1]=y
color[2]=y
color[3]=b
Bipartite graph
Graph 3
color[0]=b
color[1]=y
color[2]=y
color[3]=b
color[4]=y
color[5]=b
color[6]=b
color[7]=y
Bipartite graph
Graph 4
Not a bipartite graph
Graph 5
color[0]=b
color[1]=b
color[2]=b
color[3]=b
color[4]=b
color[5]=y
color[6]=y
color[7]=y
Bipartite graph

G:\Java>_
```

Figure 4. Execution & result of the Java code

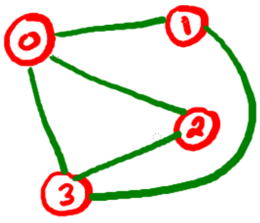


Figure 5. Graph 1 (Not a bipartite graph)

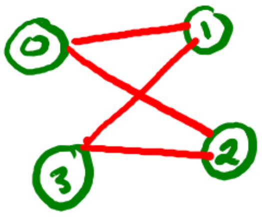


Figure 6. Graph 2 (Bipartite graph)

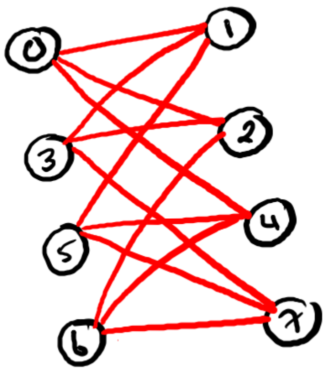


Figure 7. Graph 3 (Bipartite graph)

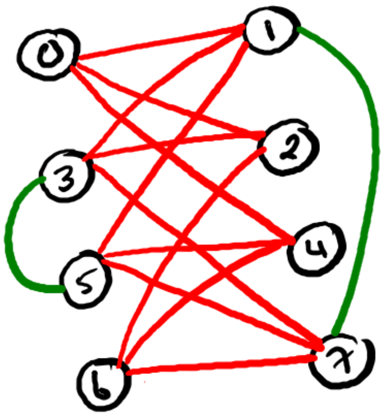


Figure 8. Graph 4 (Not a Bipartite graph)

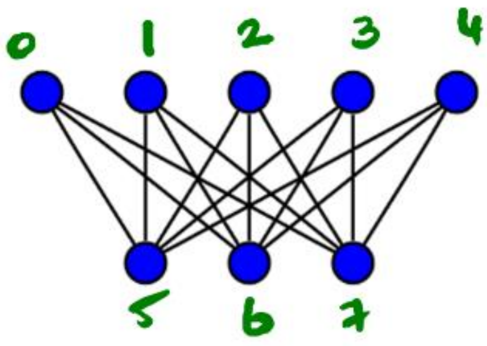


Figure 9. (Bipartite graph)

4. Conclusions & Future Work

This research presented the method to check if the given graph is bipartite or not. The color assignment of the bipartite graph is also obtained and the method is executed on some of the sample graphs and the results are presented. New strategies will be developed for bipartite graphs using the various soft computing methods [8-16].

References

[1] Lewis R.M.R.: A Guide to Graph Coloring, Algorithms and Applications. Springer International Publishing Switzerland (2016)

[2] Tommy R. Jensen, Bjarne Toft: Graph Coloring Problems. John Wiley & Sons.

[3] Marappan, R., Sethumadhavan, G. Solution to Graph Coloring Using Genetic and Tabu Search Procedures. Arab J Sci Eng 43, 525–542 (2018). https://doi.org/10.1007/s13369-017-2686-9

[4] Marc Demange; Tınaz Ekim; Bernard Ries; Cerasela Tanasescu: On some applications of the selective graph coloring problem. European Journal of Operational Research (2014)

[5] G. Sethumadhavan and R. Marappan, "A genetic algorithm for graph coloring using single parent conflict gene crossover and mutation with conflict gene removal procedure," 2013 IEEE International Conference on Computational Intelligence and Computing Research, 2013, pp. 1-6, doi: 10.1109/ICCIC.2013.6724190.

[6] Philippe Galinier; Alain Hertz: A survey of local search methods for graph coloring. Computers & Operations Research **33**(9), 2547-2562 (2006)

[7] Marappan, R.; Sethumadhavan, G. Complexity Analysis and Stochastic Convergence of Some Well-known Evolutionary Operators for Solving Graph Coloring Problem. Mathematics 2020, 8, 303. https://doi.org/10.3390/math8030303

[8] R. Marappan and G. Sethumadhavan, "A New Genetic Algorithm for Graph Coloring," 2013 Fifth International Conference on Computational Intelligence, Modelling and Simulation, 2013, pp. 49-54, doi: 10.1109/CIMSim.2013.17.

[9] Sanjukta Bhowmick; Paul D. Hovland, Improving the Performance of Graph Coloring Algorithms through Backtracking, ICCS 2008, Part I, LNCS 5101, pp. 873–882, 2008.

[10] Henning Hasemann, Juho Hirvonen, Joel Rybicki, and Jukka Suomela, Deterministic Local Algorithms, Unique Identifiers, and Fractional Graph Colouring, Springer-Verlag Berlin Heidelberg, pp. 48–60, 2012.

[11] S. Balakrishnan, T. Suresh and R. Marappan, "Solving Graph Coloring Problem Using New Greedy and Probabilistic Method," 2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS), 2022, pp. 1992-1995, doi: 10.1109/ICACCS54159.2022.9785139.

[12] Wen Sun, Jin-Kao Hao, Yuhao Zang, Xiangjing Lai, A solution-driven multilevel approach for graph coloring, Applied Soft Computing, Volume 104, 2021, 107174, https://doi.org/10.1016/j.asoc.2021.107174.

[13] Marappan, R., Bhaskaran, S. New evolutionary operators in coloring DIMACS challenge benchmark graphs. Int. j. inf. tecnol. (2022). DOI: https://doi.org/10.1007/s41870-022-01057-x

[14] Donatello Conte, Giuliano Grossi, Raffaella Lanzarotti, Jianyi Lin, Alessandro Petrini: Analysis of a parallel MCMC algorithm for graph coloring with nearly uniform balancing, Pattern Recognition Letters, Volume 149, 2021, Pages 30-36, https://doi.org/10.1016/j.patrec.2021.05.014.

[15] Angelini P.; Bekos M. A.; De Luca F.; Didimo W.; Kaufmann M.; Kobourov S.; Montecchiani F.; Raftopoulou C. N.; Roselli V.; Symvonis A.: Vertex-Coloring with Defects. Journal of Graph Algorithms and Applications (2017)

[16] Sennaiyan, B., & Suresh, T. (2022). Graph Coloring on Bipartite Graphs. International Journal of Mathematical, Engineering, Biological and Applied Computing, 1(2), 56–60.