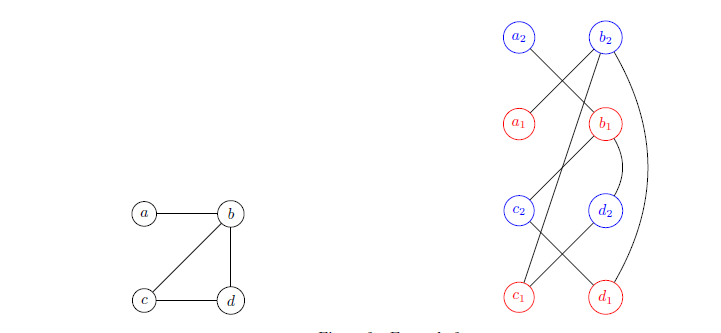
# Proof of Theorem 1:

**Theorem 1:** Let C be the number of connected components in G. Then the number of connected components in G’ is 2C if and only if the graph G is bipartite.

**Proof:**

In this new approach we are splitting every vertex to two vertices and drawing a opposite path this approach checks if a graph G has a cycle then the formed graph G’ will be connected as well and have the same connected components count as in G

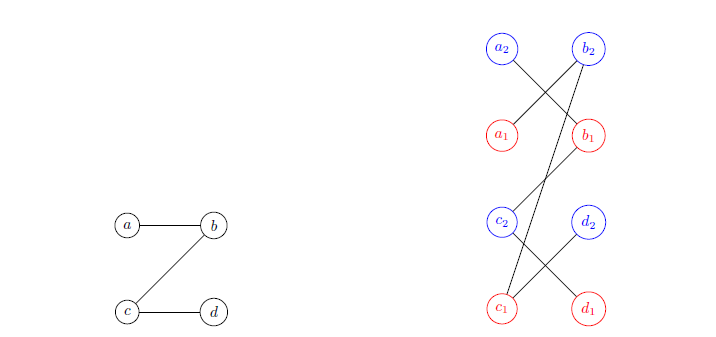
For example:



The above graph has a cycle then the formed graphs edges are connected because of that cycle and number of connected components will be same.

In case of a bipartite graph there is no cycle so the given approach of splitting vertices and edges into two and opposite direction will form a graph G’. G’ will have twice connected components as of G, a bipartite graph.

For Example:



The given approach is correct because we are making the same graph by splitting and going to opposite direction so that the resulting graph can show cycle if there was any.

The above second example shows the given graph has only one connected components but the resulting graph has two connected components so this theorem and approach is correct.

If C be the count of connected components in G then G’ will have 2C connected components only and only if G is bipartite means it can be split into two functions where one function will have all the disconnected vertices connected to 2nd function’s vertices.

# Describe an efficient algorithm for checking bipartiteness based on the above approach. What is the running time of your algorithm and why?

**Algorithm:**

* We can form another graph G’ from given graph G with the given rule.
* Let G = (V, E) be the input graph that you want to check for bipartiteness. We first transform G into a new graph G’ = (V’, E’) as follows. For each node v in V construct two nodes v1; v2 in V’ and for each edge (u, v) in E create two edges (u1, v2) and (u2, v1) in E0.
* Now we can count the connected components in G and G’ and tell if the graph is bipartite.
* For the counting of G and G’ I am going to use Depth First Search approach to count the disconnected graphs, which are count of connected components.

**Time Complexity:**

Since I am using DFS technique to count the edges so its time complexity is O(V+ E) for G graph and O(V’+E’) for G’ and for the formation of graph we have to go through edges which are E so

Time complexity = O(V + E) + O(V’ + E’) + E

# Code up your solution in C/C++/Python/Java. Your code should be well commented. Your code should compile, otherwise no points.

I am writing my code in python.

**Code:**

# class of Graph to implement all the graph related functions

class Graph():

# init function to declare class variables

def \_\_init\_\_(self,V):

self.V = V

self.edges = [[] for i in range(V)]

# addEdge function to add the edges of graph

def addEdge(self, v, w):

if w not in self.edges[v]:

self.edges[v].append(w)

if v not in self.edges[w]:

self.edges[w].append(v)

def DFSUtil(self, temp, v, visited):

# Mark the current vertex as visited

visited[v] = True

# Store the vertex to list

temp.append(v)

# Repeat for all vertices adjacent

# to this vertex v

for i in self.edges[v]:

if visited[i] == False:

# Update the list

temp = self.DFSUtil(temp, i, visited)

return temp

# is Bipartite checks the given graph and returns 1 if bipartite, else 0

def isBipartite(self):

# formation of G' graph

gprime= Graph(self.V\*2)

i=0;

for x in self.edges:

for y in x:

gprime.addEdge(i, y+self.V)

gprime.addEdge(y, i+self.V)

i=i+1;

# number of connected components in given graph

glength=len(self.NumberofconnectedComponents())

# number of connected components in formed graph by the algo

gprimelength =len(gprime.NumberofconnectedComponents())

print("Given Graph Connected Components length:")

print(glength)

print("Formed Graph Connected Components length:")

print(gprimelength)

# checking the condition of bipartiteness and returning the result

if glength\*2== gprimelength:

return 1;

else:

return 0;

# function to count connected components of the graph

def NumberofconnectedComponents(self):

visited = []

cc = []

for i in range(self.V):

visited.append(False)

for v in range(self.V):

if visited[v] == False:

temp = []

cc.append(self.DFSUtil(temp, v, visited))

return cc

# driver program

# testing for 1st graph in assignment

g = Graph(4)

# name of vertices is shown by digits Graph(4) produces a graph with vertices 0 to 3

# Adding edges for the first graph

g.addEdge(0,1)

g.addEdge(1,2)

g.addEdge(2, 3)

print("--------------------------------------------------------------")

print("Graph # 1 edges:")

print(g.edges)

if g.isBipartite()==1:

print("Given Graph is Bipartite")

else:

print("Given Graph is not Bipartite")

print("--------------------------------------------------------------")

print();

print("--------------------------------------------------------------")

# testing for the 2nd graph in assignment

print("Graph # 2 edges:")

g1 = Graph(4)

g1.addEdge(0,1)

g1.addEdge(1,2)

g1.addEdge(1,3)

g1.addEdge(2, 3)

print(g1.edges)

if g1.isBipartite()==1:

print("Given Graph is Bipartite")

else:

print("Given Graph is not Bipartite")

print("--------------------------------------------------------------")

**Output:**

