Experiment-2

Developing agent programs for real world problem - Graph coloring problem

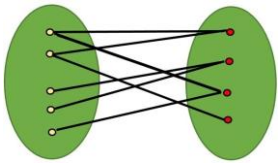
Team AI4Life
Sai Mohit Ambekar (137)
Sadekar Adesh(141)
Kapuluru Srinivasulu(142)
Praneet Botke(149)
Aayushi Goenka(151)
Sonia Raja(152)

Aim:

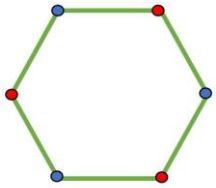
To check whether the given graph is Bipartite or not.

Procedure & Solution:

A Bipartite Graph is a graph whose vertices can be divided into two independent sets, U and V such that every edge (u, v) either connects a vertex from U to V or a vertex from V to U. In other words, for every edge (u, v), either u belongs to U and v to V, or U belongs to V and V to U. We can also say that there is no edge that connects vertices of same set.

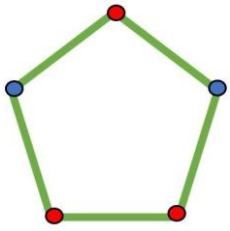


A bipartite graph is possible if the graph coloring is possible using two colors such that vertices in a set are colored with the same color. Note that it is possible to color a cycle graph with even cycle using two colors.



Cycle graph of length 6

It is not possible to color a cycle graph with odd cycle using two colors.



Cycle graph of length 5

Algorithm:

One approach is to check whether the graph is 2-colorable or not using backtracking algorithm m coloring problem.
Following is a simple algorithm to find out whether a given graph is Bipartite or not using Breadth First Search (BFS).
Assign RED color to the source vertex (putting into set U).

1. Color all the neighbors with BLUE color (putting into set V).
2. Color all neighbor's neighbor with RED color (putting into set U).
3. This way, assign color to all vertices such that it satisfies all the constraints of m way coloring problem where m = 2.
4. While assigning colors, if we find a neighbor which is colored with same color as current vertex, then the graph cannot be colored with 2 vertices (or graph is not Bipartite

Code:

main.py File (Ctrl+M)

```
class Graph():
    def __init__(self, V):
        self.V = V
        self.graph = [[0 for column in range(V)]\
        for row in range(V)]
    def isBipartite(self, src):
        colorArr = [-1] * self.V
        colorArr[src] = 1
        queue = []
        queue.append(src)
        while queue:
            u = queue.pop()
            if self.graph[u][u] == 1:
                return False;
            for v in range(self.V):
                if self.graph[u][v] == 1 and colorArr[v] == -1:
                    colorArr[v] = 1 - colorArr[u]
                    queue.append(v)
                elif self.graph[u][v] == 1 and colorArr[v] == colorArr[u]:
                    return False
            return True
# Driver program to test above function
g = Graph(4)
g.graph = [[0, 1, 0, 1],
[1, 0, 1, 0],
[0, 1, 0, 1],
[1, 0, 1, 0]
]
if g.isBipartite(0):
    print("Yes")
else:
    print("No")
```

Output:

Result:

The given graph is Bipartite in nature as only 2 colors were used.