Enlai Yii
2064210

Programming Assignment 4

1. Algorithm Explanation

Theorem: A graph is bipartite if and only if it has no odd length cycle, i.e., no cycle in the graph has an odd number of edges. Note that if a graph has no cycles it is bipartite (hence trees are bipartite graphs).

The algorithm will take a graph as a parameter and it will use the theorem to check if a graph is bipartite or not. The algorithm for *isBipartite* will call *Number_Of_Connected_Components* for the graph and graph prime which will also call *Transformed_Graph*. The *isBipartite* algorithm should return the given boolean statement *return 2C == C'* as that will be the condition to verify that the graph is bipartite.

The *Number_Of_Connected_Components* will use depth first search to count the number of components of a graph and it should return the total number of components.

The final function *Transformed_Graph* will create the transformed graph and the result should be twice the size of the original graph.

2. Runtime

Let n be the number of nodes and m be the number of edges in the graph. The main function, *isBipartite*, will call the *Number_Of_Connected_Components* function and that will take O(n + m). Since the function *Transformed_Graph* will be twice the original graph's size, when the function is called in *isBipartite*, the total time will be O(2(n + m)). And it could be simplified to O(n + m). The return boolean statement will only take O(1).

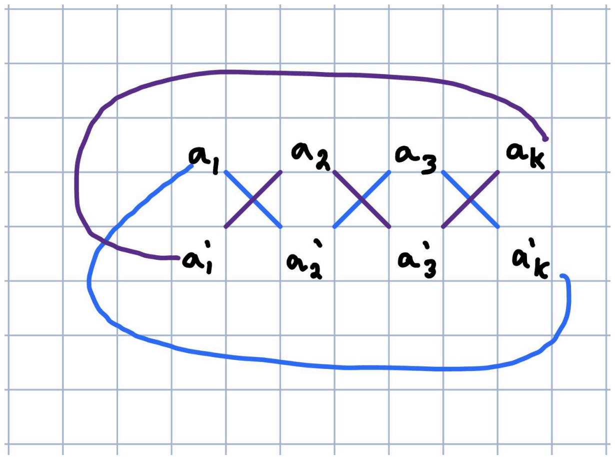
Since each edge will be visited exactly once in the algorithm, the depth first search is called for every node in the graph and each node will potentially view each neigher. The number of edges we visit will be at most 2m, so the total complexity for DFS will be O(n + 2m).

Thus the total runtime will be O(n + m).

3. Correctness

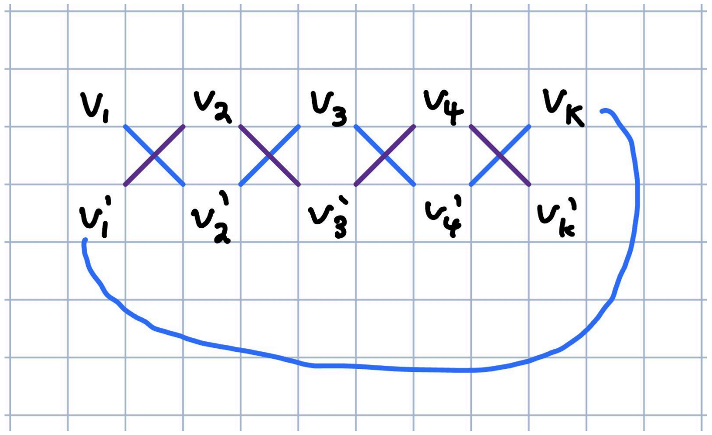
The algorithm will always determine if a graph is bipartite or not. We want to show that if a graph is bipartite, then the theorem is true, AND if the theorem is true, then the graph is bipartite.

First, we want to prove that if the graph is bipartite, then the theorem is correct. Let *a1*, *a2*, *a3*, *ak* be nodes in a graph called G, and G will have prime nodes. Here's an example of connect components for *a* nodes:



The example from above shows that there are 2 cycles for the graph when the algorithm is used. So, G will have one component and G prime will have 2 connected components for the graph. In general terms, there are an even number of cycles for the graph. Thus, the theorem is true for all k nodes with an even number of cycles.

Now, we want to show that if the theorem is true, then the graph is bipartite. Let *v1*, *v2*, *v3*, *v4*, and *vk* be nodes for graph G1. We will prove the 2nd statement by using a contrapositive statement, so the statement will be: if the graph is not bipartite, then the theorem is false. Here's an example of connected components for v nodes:

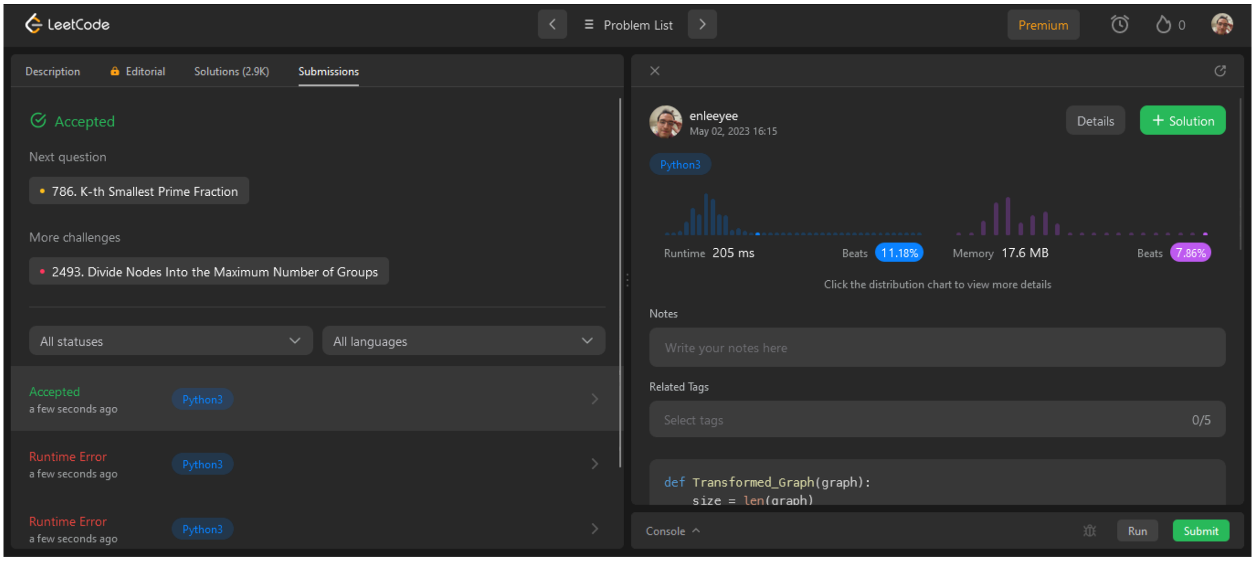


The example from above shows that an edge will connect nodes *vk* and *v1* prime when the algorithm is used. This will cause the graph to only have 1 connected component for G1 and G1 prime. Therefore, the graph is not bipartite, so the theorem is false. Hence, if the theorem is true, then the graph is bipartite and is proven by contrapositive.

With both parts, we have shown that if a graph is a bipartite graph, then the theorem is true; and, if the theorem is true, then the graph is a bipartite graph. Hence, the algorithm is correct for all cases.

4. LeetCode Submission

https://leetcode.com/problems/is-graph-bipartite/submissions/943469395/



Accepted

Next question

786. K-th Smallest Prime Fraction

More challenges

2493. Divide Nodes Into the Maximum Number of Groups

All statuses

All languages

Accepted
a few seconds ago
Python3

Runtime Error
a few seconds ago
Python3

Runtime Error
a few seconds ago
Python3

enleeyee
May 02, 2023 16:15

Python3

Runtime 205 ms Beats 11.18%

Memory 17.6 MB Beats 7.86%

Click the distribution chart to view more details

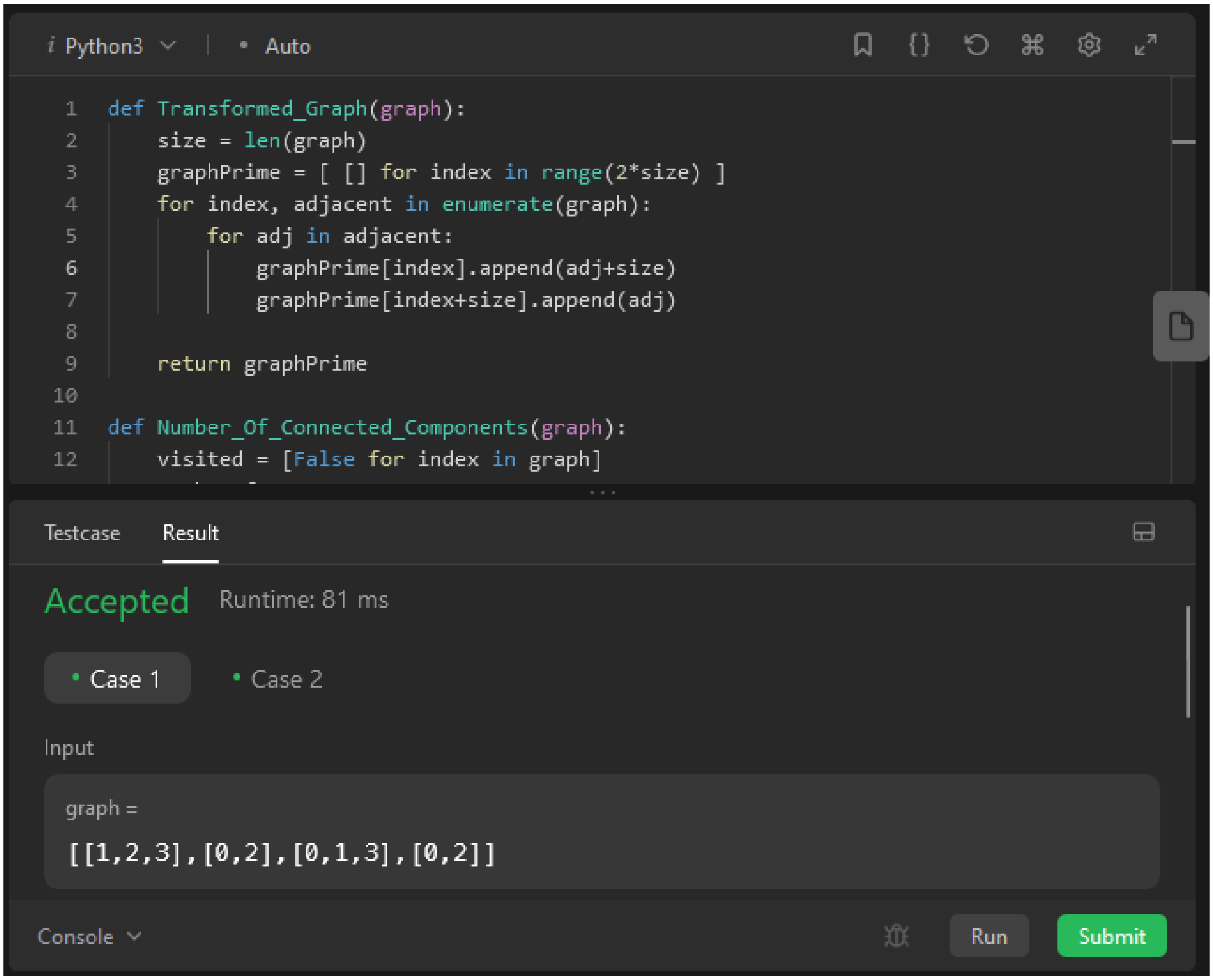
Notes

Write your notes here

Related Tags

Select tags 0/5

```
def Transformed_Graph(graph):
    size = len(graph)
```



```python
def Transformed_Graph(graph):
    size = len(graph)
    graphPrime = [ [] for index in range(2*size) ]
    for index, adjacent in enumerate(graph):
        for adj in adjacent:
            graphPrime[index].append(adj+size)
            graphPrime[index+size].append(adj)

    return graphPrime

def Number_Of_Connected_Components(graph):
    visited = [False for index in graph]
```

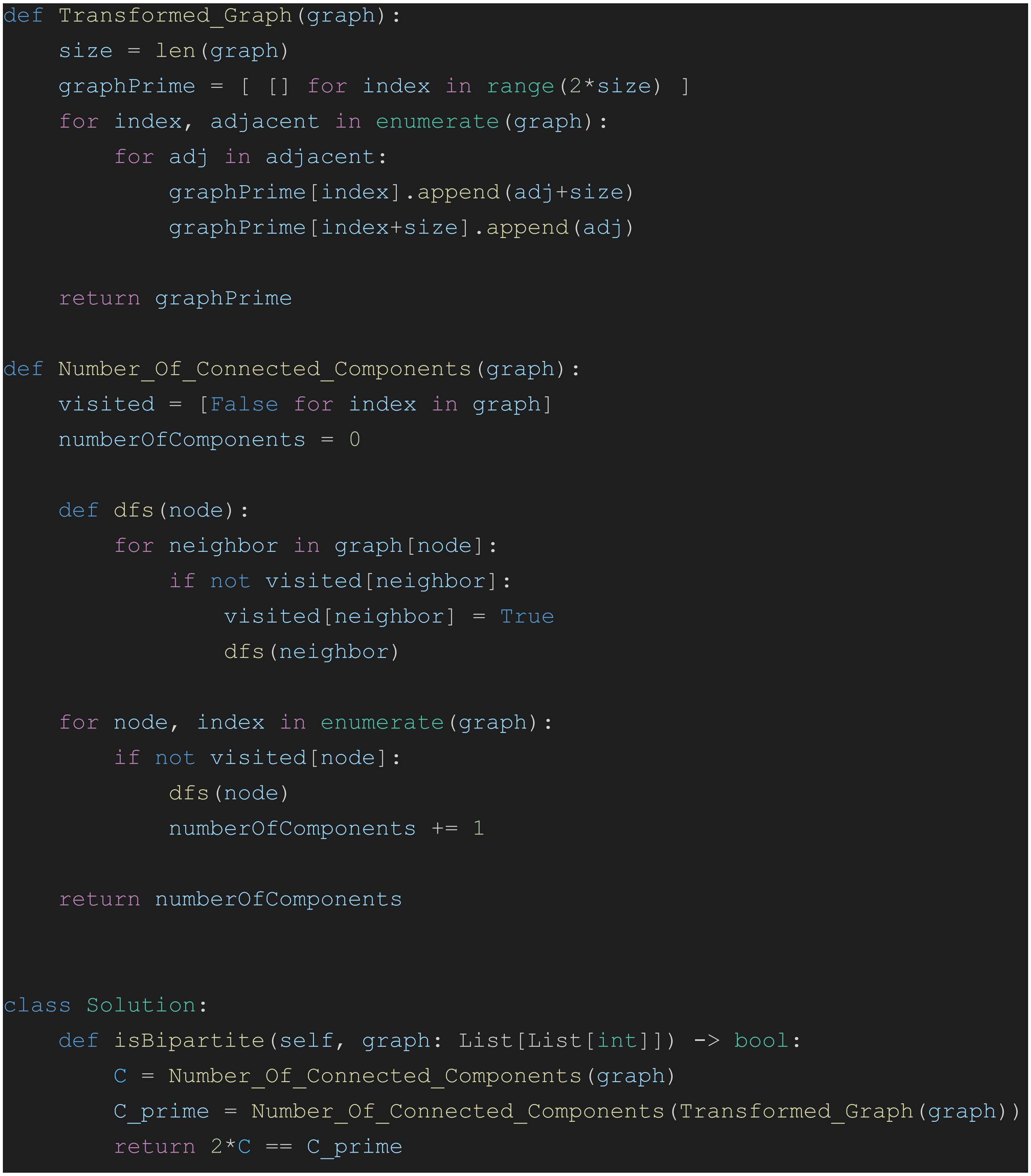
Accepted Runtime: 81 ms

Input

graph =

```
[[1,2,3],[0,2],[0,1,3],[0,2]]
```

5. Code



```python
def Transformed_Graph(graph):
    size = len(graph)
    graphPrime = [ [] for index in range(2*size) ]
    for index, adjacent in enumerate(graph):
        for adj in adjacent:
            graphPrime[index].append(adj+size)
            graphPrime[index+size].append(adj)

    return graphPrime

def Number_Of_Connected_Components(graph):
    visited = [False for index in graph]
    numberOfComponents = 0

    def dfs(node):
        for neighbor in graph[node]:
            if not visited[neighbor]:
                visited[neighbor] = True
                dfs(neighbor)

    for node, index in enumerate(graph):
        if not visited[node]:
            dfs(node)
            numberOfComponents += 1

    return numberOfComponents


class Solution:
    def isBipartite(self, graph: List[List[int]]) -> bool:
        C = Number_Of_Connected_Components(graph)
        C_prime = Number_Of_Connected_Components(Transformed_Graph(graph))
        return 2*C == C_prime
```