Same Color Paths

For this problem, you will be working with graphs. A graph is composed of a set of vertices with zero or more edges between them. The edges in a graphs can be *directed*, which means that the edge can only be traversed from its source and to its sink, or *undirected*, which means the edge can be traversed in either direction. We extend these terms to graphs: a *directed graph* has directed edges, while in an *undirected graph* has undirected edges. In this problem, you will be working with directed graphs.

For this problem, each vertex in the graph will have an associated color and name.

Here is a sample graph with five vertices and six edges.

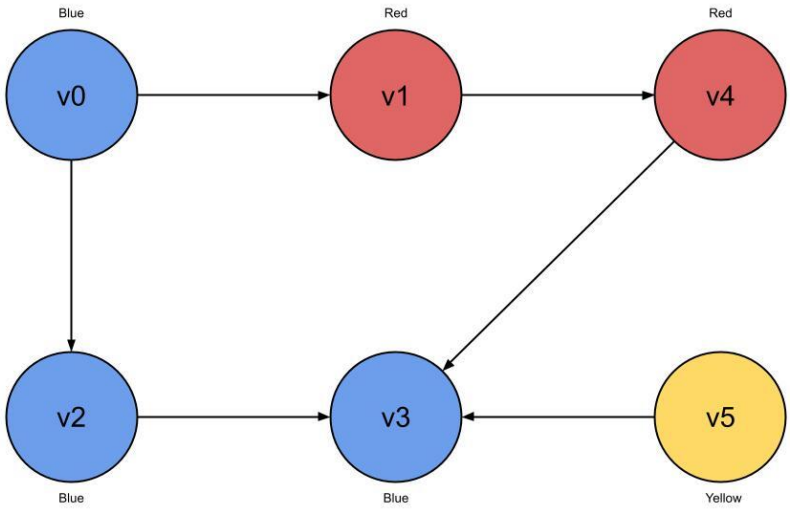

Figure 1: Example Graph

A *path* is a list vertices that are connected by edges in the graph. A path is a *same-color path*, if **all** the vertices on the path have the same color. Here are some examples:

Path	Same-color Path?
$v0 \to v2 \to v3$	Yes
$v0 \to v1 \to v4 \to v3$	No

Path	Same-color Path?
$v1 \rightarrow v4 \rightarrow v3$	No
$v5$	Yes

All the vertices in the first path are blue, so it is a same-color path. Some of the vertices on the second path are blue and some are red, so it is not a same-color path. The same is true of the third path. The fourth example path, which contains a single vertex, *is* a same-color path.

Your task is write a function `has_same_color_path`, that takes a graph, the name of a source vertex, and the name of a destination vertex. The function should return `True` if the graph contains a same-color path from the source to the destination. Otherwise, the function should return `False`.

Here are some examples using the graph shown above:

Source	Destination	Same-color Path?
v0	v3	Yes
v1	v3	No
v5	v5	Yes

You must solve the problem in **one** of Python 3, Java, or C++. See below for language-specific instructions and starter code.

For the practice problems, we have provided the necessary files for you. When you take the actual exam, you will be expected to copy code into files with specific names.

Python 3

Vertex and Graph

We have defined a `Vertex` class and a `Graph` class for representing vertices and graphs respectively:

```
from typing import Dict, List, Optional

class Vertex:
    '''
    A vertex in a graph.

    Attributes:
        name (str): The name of the vertex.
        color (str): The color of the vertex.
        neighbors (List(vertex)): A list of the vertices that are a
```

```
            destination of an edge starting at this vertex.

        Methods:
            add_edge: Adds an edge to this vertex.
        '''
        name : str
        color : str
        neighbors : List["Vertex"]

        def __init__(self, name: str, color: str):
            '''
            Initializes a vertex.

            Arguments:
                name (str): The name of the vertex.
                color (str): The color of the vertex.
            '''
            self.name = name
            self.color = color
            self.neighbors = []

        def add_edge(self, dest: "Vertex") -> None:
            '''
            Adds an edge to this vertex.

            Arguments:
                dest (Vertex): The vertex to add an edge to.

            Returns:
                None
            '''
            self.neighbors.append(dest)


    class Graph:
        '''
        A graph.

        Attributes:
            vertices (Dict[str, Vertex]): A dictionary of vertices in the graph.
                Note that these are vertex names mapped to vertex objects, and not
                vertex objects mapped to vertex objects representing edges.

        Methods:
            add_vertex: Adds a vertex to the graph.
            get_vertex: Gets a vertex from the graph.
```

```
    '''
    vertices : Dict[str, Vertex]

    def __init__(self):
        '''
        Initializes a graph.
        '''
        self.vertices = {}

    def add_vertex(self, vertex: Vertex) -> None:
        '''
        Adds a vertex to the graph.

        Arguments:
            vertex (Vertex): The vertex to add to the graph.

        Returns:
            None
        '''
        self.vertices[vertex.name] = vertex

    def get_vertex(self, name: str) -> Optional[Vertex]:
        '''
        Gets a vertex from the graph.

        Arguments:
            name (str): The name of the vertex to get.

        Returns:
            Optional[Vertex]: The vertex with the given name, or None if no
                vertex with that name exists.
        '''
        return self.vertices.get(name)
```

You can find this code in a file named `graph.py`

Same Color

Here is the skeleton code for this task:

```
'''
Distribution file for the same-color problem.
'''

from graph import Graph
```

```python
def has_same_color_path(graph: Graph, start: str, end: str) -> bool:
    '''
    Determines if there is a path between start and end where all nodes in the
    path are the same color.

    Arguments:
        graph (Graph): The graph to search.
        start (str): The starting node.
        end (str): The ending node.

    Returns:
        bool: True if there is a path between start and end where all nodes in
            the path are the same color, False otherwise.
    '''
    # TODO: Implement this function.
    pass
```

You can find this code in a file named `same_color.py`

Your task is to complete the function `has_same_color_path`. Given a graph, the name of a the starting vertex in the path, and the name of the ending vertex in the path, the function should return `True` if there exists a same-color path in the graph from the starting vertex to the ending vertex.

Java

Vertex

We have defined a `Vertex` class for representing vertices.

```java
import java.util.List;
import java.util.ArrayList;


/**
 * A class for representing vertices in a graph.
 *
 * Public attributes:
 *   name: The name of the vertex.
 *   color: The color of the vertex.
 *   neighbors: A list of the neighbors of this vertex
 *     (Vertex v is a neighbor of Vertex w, if the graph
 *      contains a directed edge from w to v.)
 *
 * Public methods:
 *   add_edge: Adds an from this vertex to another vertex
 **/
```

```
public class Vertex {
    public String name;
    public String color;
    public List<Vertex> neighbors;

    public Vertex(String name, String color) {
        this.name = name;
        this.color = color;
        this.neighbors = new ArrayList<Vertex>();

    }

    public void addEdge(Vertex v) {
        this.neighbors.add(v);
    }

    public String toString() {
        String rv = String.format("%s (%s):", this.name, this.color);
        for (Vertex v : this.neighbors) {
            rv += " " + v.name;
        }
        return rv;
    }
}
```

You can find this code in a file named `Vertex.java`.

Graph

We have also defined a `Graph` class for representing graphs:

```
import java.util.Map;
import java.util.HashMap;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

/**
 * A class for representing graphs
 *
 * Public attributes:
 *   vertices: the vertices in the graph represented as a map that
 *     maps vertex names to instances of the Vertex class
 *
 * Public methods:
 *   add_vertex: Adds a vertex to the graph.
 *   get_vertex: Gets a vertex from the graph.
 **/
```

```
public class Graph {
    public Map<String, Vertex> vertices;

    public Graph() {
        vertices = new HashMap<String, Vertex>();
    }

    /** Add vertex v to the graph
     *
     * Arguments:
     *   v: the vertex to add
     **/
    public void addVertex(Vertex v) {
        this.vertices.put(v.name, v);
    }

    /**
     * Get the vertex associated with a name
     *
     * Arguments:
     *   String name;
     *
     * Returns: the Vertex associated with the name or null
     **/
    public Vertex getVertex(String name) {
        if (this.vertices.containsKey(name)) {
            return this.vertices.get(name);
        } else {
            return null;
        }
    }

    public String toString() {
        String rv = "";
        List<String> vertexNames = new ArrayList<String>(this.vertices.keySet());
        Collections.sort(vertexNames);
        for (String vname : vertexNames) {
            Vertex v = this.getVertex(vname);
            rv += v + "\n";
        }
        return rv;
    }

}
```

You can find this code in a file named `Graph.java`.

Same Color

Here is the skeleton code for this task:

```
import java.util.Set;
import java.util.HashSet;


public class SameColor {
    /**
     * Determine whether there is a same color path in the graph between
     * two vertices.
     *
     * Arguments:
     *   g: The graph to search.
     *   start: the name of the source vertex
     *   end: the name of the destination vertex
     *
     * Returns: True if there is a path between start and end where
     *   all nodes in the path are the same color, False otherwise.
     **/
    public static Boolean hasSameColor(Graph g, String s, String d) {
        // TO DO: complete this function
        // The return is included to allow the skeleton code to compile.
        return null;
    }
}
```

You can find this code in a file named `SameColor.java`

Your task is to complete the function `hasSameColorPath`. Given a graph, the name of a the starting vertex in the path, and the name of the ending vertex in the path, the function should return `True` if there exists a same-color path in the graph from the starting vertex to the ending vertex.

C++

Vertex

We have defined a `Vertex` class for representing vertices. Here is the header file

for the `Vertex` class:

```
#ifndef _Vertex_
#define _Vertex_

#include <vector>
#include <string>

/**
 * A class for representing vertices in a graph.
 *
 * Public attributes:
 *   name: The name of the vertex.
 *   color: The color of the vertex.
 *   neighbors: A list of the neighbors of this vertex
 *    (Vertex v is a neighbor of Vertex w, if the graph
       contains a directed edge from w to v.)
 *
 * Public methods:
 *   add_edge: Adds an from this vertex to another vertex
 **/
class Vertex {
public:
std::string name;
std::string color;
std::vector<Vertex*> neighbors;

Vertex(std::string const &_name, std::string const &_color)
: name(_name), color(_color), neighbors(std::vector<Vertex*>()) {}

void addEdge(Vertex* v) {
    neighbors.push_back(v);
}

std::string to_string() const {
    std::string rv = name + " (" + color + "):";
    for (auto v : neighbors) {
        rv += v->name;
    }
    return rv;
}

};

#endif
```

You can find this code in a file named `Vertex.h`. This header file contains

all the code necessary for the `Vertex` class; there is no need for a separate implementation file.

Graph

We have also defined a `Graph` class for representing graphs. Here is the header file for the `Graph` class:

```
#ifndef _Graph_
#define _Graph_

#include "Vertex.h"
#include <vector>
#include <unordered_map>

/**
 * A class for representing graphs
 *
 * Public attributes:
 *   vertices: the vertices in the graph represented as a map that
 *     maps vertex names to instances of the Vertex class
 *
 * Public methods:
 *   add_vertex: Adds a vertex to the graph.
 *   get_vertex: Gets a vertex from the graph.
 **/
class Graph {
public:
std::unordered_map<std::string, Vertex*> vertices;

Graph() : vertices(std::unordered_map<std::string, Vertex*>()) {};

/** @brief vertex v to the graph
 *
 * @param v the vertex to add
 **/
void add_vertex(Vertex* v) { vertices[v->name] = v; }

/** @brief Get the vertex associated with a name
 *
 * @param name
 *
 * @return the Vertex associated with the name or null
 **/
Vertex* get_vertex(std::string name) {
    if (vertices.find(name) != vertices.end()) {
        return vertices[name];
```

```
    } else {
        return nullptr;
    }
}

std::string to_string() {
    std::string rv = "";
    std::unordered_map<std::string, Vertex*>::iterator pos;
    for (pos = vertices.begin(); pos != vertices.end(); ++pos) {
        rv += pos->first + "\n";
    }
    return rv;
}

};

#endif
```

You can find this code in a file named `Graph.h`. This header file contains all the code necessary for the `Graph` class; there is no need for a separate implementation file.

Same Color

Here is the header file for this task:

```
#ifndef _SameColor_
#define _SameColor_

#include "Vertex.h"
#include "Graph.h"
#include <vector>
#include <unordered_map>
#include <string>

/** @brief Determine whether there is a same color path in the graph between
 * two vertices.
 *
 * @param g The graph to search
 * @param start the name of the source vertex
 * @param end the name of the destination vertex
 *
 * @return True if there is a path between start and end where
 * all nodes in the path are the same color, False otherwise.
 **/
bool has_same_color_path(Graph* g, std::string const &s, std::string const &d);

#endif
```

You can find this code in a file named `SameColor.h`

And here is the skeleton code for this task:

```
#include "SameColor.h"
#include "Graph.h"
#include "Vertex.h"
#include <vector>
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <utility>


bool has_same_color_path(Graph* g, std::string const &s, std::string const &d) {
    // TO DO: complete this function
    return false;
}
```

You can find this code in a file named `SameColor.cpp`

Your task is to complete the function `has_same_color_path`. Given a graph, the name of a the starting vertex in the path, and the name of the ending vertex in the path, the function should return `True` if there exists a same-color path in the graph from the starting vertex to the ending vertex.

You must only modify and submit file `SameColor.cpp` Do not include a `main` function in the `SameColor.cpp` file, as that will interfere with our automated tests. If you want to test your implementation informally, please do so by creating a separate file containing a `main` function.