# COMPLETE SEARCH – BRUTE FORCE

GROUP 7

Nguyễn Quốc Trường 21521604

Trần Xuân Minh 21520352

A graph is said to be bipartite if all its vertices can be partitioned into two disjoint subsets X and Y so that every edge connects a vertex in X with a vertex in Y. (One can also say that a graph is bipartite if its vertices can be colored in two colors so that every edge has its vertices colored in different colors; such graphs are also called 2-colorable.)

1. Design a DFS-based algorithm for checking whether a graph is bipartite.
2. Design a BFS-based algorithm for checking whether a graph is bipartite.

Answer:

1. DFS-based algorithm for checking whether a graph is bipartite:

To check whether a graph is bipartite or not, we can use a DFS (Depth-First Search) algorithm to traverse all the vertices of the graph and mark each vertex as one of two different colors, for example, black and white. During the traversal, if we encounter an edge between two vertices of the same color, then the graph is not bipartite. The DFS algorithm will traverse all the vertices of the graph, so the time complexity is O(V + E), where V is the number of vertices of the graph and E is the number of edges.

Here is the pseudocode of the DFS algorithm to check whether a graph is bipartite or not:

```
function isBipartiteDFS(graph):
    colors = [None] * len(graph)
    for vertex in graph:
        if colors[vertex] is None:
            colors[vertex] = "black"
            if not dfs_visit(vertex, colors, graph):
                return False
    return True
```

```
function dfs_visit(vertex, colors, graph):
    for neighbor in graph[vertex]:
        if colors[neighbor] is None:
            colors[neighbor] = "white" if colors[vertex] == "black" else "black"
            if not dfs_visit(neighbor, colors, graph):
                return False
        elif colors[neighbor] == colors[vertex]:
            return False
    return True
```

Here, graph is an adjacency list of the graph, colors is an array to store the color of each vertex, and dfs_visit is a recursive function to traverse the neighbors of a vertex and mark the vertices visited.

2. BFS-based algorithm for checking whether a graph is bipartite:

The BFS (Breadth-First Search) algorithm can also be used to check whether a graph is bipartite or not. The vertices are traversed using a queue, and each vertex visited is marked with one of two different colors as in the DFS algorithm. If we encounter an edge between two vertices of the same color, then the graph is not bipartite. The BFS algorithm will traverse all the vertices of the graph, so the time complexity is also O(V + E).

Here is the pseudocode of the BFS algorithm to check whether a graph is bipartite or not:

```
function isBipartiteBFS(graph):
    colors = [None] * len(graph)
    for vertex in graph:
        if colors[vertex] is None:
            colors[vertex] = "black"
            queue = [vertex]
            while queue:
```

```
            current = queue.pop(0)
            for neighbor in graph[current]:
                if colors[neighbor] is None:
                    colors[neighbor] = "white" if colors[current] == "black" else "black"
                    queue.append(neighbor)
                elif colors[neighbor] == colors[current]:
                    return False
    return True
```

Here, graph is an adjacency list of the graph, colors is an array to store the color of each vertex, and the algorithm uses a queue to traverse the vertices of the graph.