```
from collections import defaultdict

def adjacency_list(edges):

    adjacency_list = defaultdict(set)

    for u, v in edges:
        adjacency_list[u].add(v)
        adjacency_list[v].add(u)  # Так как граф неориентированный
    return {node: list(neighbors) for node, neighbors in adjacency_list.items()}

def file(filename):
    edges = []
    with open(filename, 'r') as file:
        for line in file:
            parts = line.strip().split()  # Разделяем строку по пробелу
            if len(parts) == 2:  # Проверяем, что ровно два элемента
                edges.append((int(parts[0]), int(parts[1])))
    return edges

test = adjacency_list(file('test.txt'))
data = adjacency_list(file('data.txt'))

print(test)

from collections import deque

def is_bipartite_bfs(graph):

    color = {}

    for node in graph:
```

```
        if node not in color:

            queue = deque([node])
            color[node] = 0

            while queue:
                current = queue.popleft()

                for neighbor in graph[current]:

                    if neighbor not in color:
                        color[neighbor] = 1 - color[current]
                        queue.append(neighbor)

                    elif color[neighbor] == color[current]:
                        return False
    return True

import time

start_time = time.time()
bipartite = is_bipartite_bfs(test)
end_time = time.time()

print(f'Время выполнения алгоритма проверки графа на двудольность (BFS): {end_time - start_time} секунд')
print(f'Двудольный ли граф? - {bipartite}')

import time

start_time = time.time()
bipartite = is_bipartite_bfs(data)
```

```
end_time = time.time()


print(f'Время выполнения алгоритма проверки графа на двудольность (BFS): {end_time - start_time} секунд')
print(f'Двудольный ли граф? - {bipartite}')


def is_bipartite_dfs(graph):

    color = {}

    def dfs(node, current_color):

        color[node] = current_color

        for neighbor in graph[node]:
            if neighbor not in color:
                # Рекурсивно вызываем DFS для соседей, с противоположным цветом
                if not dfs(neighbor, 1 - current_color):
                    return False
            elif color[neighbor] == color[node]:
                # Найдено ребро между вершинами одного цвета
                return False
        return True

    for node in graph:  # Проходим по всем вершинам (учитываем несвязные графы)
        if node not in color:
            if not dfs(node, 0):
                return False

    return True

import time
```

```
start_time = time.time()

bipartite = is_bipartite_dfs(test)

end_time = time.time()


print(f'Время выполнения алгоритма проверки графа на двудольность (DFS): {end_time - start_time} секунд')

print(f'Двудольный ли граф? - {bipartite}')


import time


start_time = time.time()

bipartite = is_bipartite_dfs(data)

end_time = time.time()


print(f'Время выполнения алгоритма проверки графа на двудольность (DFS): {end_time - start_time} секунд')

print(f'Двудольный ли граф? - {bipartite}')
```