The following exercise is based on the concept of so-called bipartite graphs:

A graph $(V, E)$ is called bipartite, if there exist $V_0$ and $V_1$ with $V_0 \subset V$ and $V_1 = V \setminus V_0$, and for all $(v, w) \in E$ there is either $v \in V_0$ and $w \in V_1$, or $w \in V_0$ and $v \in V_1$.

To put it simpler: for a bipartite graph, it is possible to attribute each node $v \in V$ with one of two "colors", say red and black, such that any edge $e \in E$ will connect a red and a black node (and no edge will connect edges of the same color).

**Exercise 1**

Give a prove to the following claim:

*If a graph $(V, E)$ is bipartite, then it cannot contain an odd cycle (i.e., a cycle of odd length).*

**Solution:**

We proof this by contradiction:

Assume that a bipartite graph contains a cycle $(v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n = v_0)$, where $n$ is odd. Assume that $v_0$ is coloured black, then $v_1$ has to be red (as it is connected to the black $v_0$), $v_2$ has to be black (as it is connected to the red $v_1$), etc. Thus all $v_{2i}$ are black and all $v_{2i+1}$ are red. However, if $n$ is odd, i.e. $n = 2i + 1$ for some $i$, then $v_n$ is red, and thus cannot be equal to $v_0$, which is black. This contradicts our initial assumption.
Exercise 2

Try to find an algorithm that tests whether a given graph is bipartite.

*Hint: you can build such an algorithm by extending one of the graph traversals we discussed in the lecture!*

**Solution:**

We can change breadth-first traversal into a colouring algorithm. Assume that we have \( n \) nodes with distinct key values \( \{1, \ldots, n\} \), then we can use an array \( M \) to store the colouring status: 0 means "uncoloured", 1 is for black, and 2 for red.

```java
BFbipartite(x:Node) {
    bipartite = true;
    ! uses queue of "active" nodes
    Queue active = { x };
    Mark[x.key] = 1;
    while active <> {} do
        ! remove first node from queue
        V = remove(active);
        ! determine the opposite colour of V:
        if Mark[V.key]=1 then newcolour=2 else newcolour=1 end if;
        ! visit all nodes W connect to V by an edge (V,W):
        forall (V,W) in V.edges do
            ! assign a colour to W, if it is still uncoloured
            if Mark[W.key] = 0 then
                Mark[W.key] = newcolour;
                ! check all nodes connected to W for a colour conflict:
                forall (W,Y) in V.edges do
                    if Mark[y.key] = newcolour then bipartite = false;
                end do;
                ! for BFT: append W to queue of active nodes:
                append(active, W);
            end if;
        end do;
    end while;
    return bipartite;
}
```

Note: instead of the assignment \( \text{bipartite} = \text{false} \), we could also immediately \( \text{return false} \), and thus not necessarily traverse the entire graph (and thus be much more efficient). However, we implement a full traversal here, as we will discuss a quite similar algorithm in the next exercise.
Exercise 3

Try to give a prove for the following claim (using the algorithm from Exercise 2):

If a graph \((V,E)\) is not bipartite, then it will contain an odd cycle.

Solution:

The algorithm of Exercise 3 contains a standard breadth-first search; we can simply change it by assigning a distance to the starting node \(s\) to each node, instead of marking it as black or red. We obtain the following algorithm:

\[
\text{BFdistance (x:Node) } \{
! \text{array Mark contains value } -1 \text{ in every element at start}
! \text{bipartite = true;}
! \text{uses queue of ”active” nodes}
\text{Queue active = \{ x \};}
\text{Mark[x.key] = 0;}
\text{while active <> {} do}
! \text{remove first node from queue}
\text{V = remove(active);}
! \text{determine the opposite colour of V:}
\text{dist := Mark[V.key];}
! \text{visit all nodes W connect to V by an edge (V,W):}
\text{forall (V,W) in V.edges do}
! \text{assign a distance to W, if it is still uncoloured}
\text{if Mark[W.key] = -1}
! \text{for BFT: append W to queue of active nodes:}
\text{append(active, W);}
\text{end if;}
\text{end do;}
\text{return bipartite;}
\}
\]

Note: as 0 is a valid distance (which is correct for the start node \(x\)), we have to use a different value (which is -1) to mark nodes that have not yet been visited.

Due to the identical breadth-first traversal structure of algorithms BFbipartite and BFdistance, it is obvious that nodes with an odd distance are marked red, while nodes with an even distance are marked black.

We now have two opposite situations:

1. There is no edge in the graph that connects two nodes with even distance (which would both be marked black by algorithm 2), and no edge that connects two nodes with odd distance (which would be red). In that case, we have a bipartite graph.
2. There is an edge in the graph that connects two nodes with odd or two nodes with even distances. As both nodes are connected to the starting node (via different paths) the edge between them generates a cycle. The length of this cycle is 1 plus either the sum of two odd numbers or the sum of two even numbers. In any case, the cycle length is an odd number.

Hence, a graph can either be bipartite or have an odd cycle.

Reference for all three exercises: