Algorithms and Datastructures

Assignment 2

Lucas van der Laan
s1047485

1

The number of the item in the list indicated the iteration of BFS.
In the discovered list, we denote (p, i) where p is the predecessor and i is the iteration.
Initialization:
$Q = [2], \text{Discovered} = [2]$

1. $Q = [3, 4], \text{Discovered} = [2, 3(2, 1), 4(2, 1)], \text{Done} = [2]$
2. $Q = [4], \text{Discovered} = [2, 3(2, 1), 4(2, 1)], \text{Done} = [2, 3]$
3. $Q = [5], \text{Discovered} = [2, 3(2, 1), 4(2, 1), 5(4, 3)], \text{Done} = [2, 3, 4]$
4. $Q = [1], \text{Discovered} = [2, 3(2, 1), 4(2, 1), 5(4, 3), 1(5, 4)], \text{Done} = [2, 3, 4, 5]$
5. $Q = [], \text{Discovered} = [2, 3(2, 1), 4(2, 1), 5(4, 3), 1(5, 4)], \text{Done} = [2, 3, 4, 5, 1]$

2

Algorithm 1 Push

```
Data: Two queues q1, q2 with functions: size, enqueue, dequeue
procedure Push(n)
    size ← q1.size()
    if size == 0 then
        q1.enqueue(n)
    else
        i ← 0
        while i < size do
            q2.enqueue(q1.dequeue())
            i ← i + 1
        end while
        q1.enqueue(n)
        i ← 0
        while i < size do
            q1.enqueue(q2.dequeue())
            i ← i + 1
        end while
    end if
    return n
end procedure
```

Algorithm 2 Pop

```
Data: Two queues q1, q2 with functions: size, enqueue, dequeue
procedure Pop
    return q1.dequeue()
end procedure
```

Push: $\mathcal{O}(n)$, we have $2n$ because we go through a loop n times twice.
Pop: $\mathcal{O}(1)$, we only dequeue a single item which is by itself only 1 action.

3

Algorithm 3 Find universal sink - Adjacency Matrix

a)

```
Require: adj - The adjacency matrix
i ← j ← 0
while i < |V| and j < |V| do
    if adj[i][j] = 1 then
        i ← i + 1
    else
        j ← j + 1
    end if
end while
if i > |V| then
    return False
end if
for k..|V| do
    if adj[i][k] = 1 then
        return False
    end if
    if adj[k][i] = 0 and k ≠ i then
        return False
    end if
end for
return True
```

Algorithm 4 Find universal sink - Adjacency List

b)

```
in ← out ← [0 · |V|]                    ▷ lists of length |V| filled with zeros
for i..|V| do
    out[i] = |adj[i]|
    for each j ∈ adj[i] do
        in ← in + 1
    end for
end for
for i..|in| do
    if in[i] = |V| − 1 and out[i] = 0 then
        return True
    end if
end for
return False
```

$\mathcal{O}(|V|)$ does not exist because we can only access the adjacencies by going through all the adjacency lists. This makes it $\mathcal{O}(|V| + |E|)$.

4

No time left, I did find an algorithm real quickly online that does it using DFS. But that is not my solution, so I will not just quickly paste it here: https://www.baeldung.com/cs/detecting-cycles-in-directed-graph

5

DISCLAIMER: I realized after I made it that my assumptions were wrong, but I didn't have the time to fix it.
Just to give some assumptions that I made, based on reading the text:

- **Inter-species friendships:** This means that only A-H or H-A friendships exist, because inter-species means "between" species, ergo between a human and an android. If this assumption is wrong, I see no way to partition based on human/android.

- **Complete list of the r pairs:** This means that all the humanlike entities are connected through pairs to each other.

- r**:** R is is a list of friendships that contain indexes the n

---

Algorithm 5 Create Adjacency Matrix

Data:
    r - The friendships consisting of {first, second} which are both indexes to n,
    n - group of human-like species
procedure CreateAdjMatrix(n, r)
    $a_{ij} \leftarrow [|n|][|n|]$
    for $i \leftarrow 0$ to $|r|$ **do**
        $a_{ij}[r.first][r.second] = 1$
        $a_{ij}[r.second][r.first] = 1$
    end for
    return $\leftarrow a_{ij}$
end procedure

Algorithm 6 Partition Human-like species

```
Require:
    r - The friendships consisting of {first, second} which are both indexes to n
    n - group of human-like species
Q1, Q2 ← Queue
list1, list2 ← []
adj ← CreateAdjMatrix(n, r)
Enqueue(Q1, 0)                              ▷ We select that we start at 0
while Q1 ≠ ∅  or Q2 ≠ ∅ do
    while Q1 ≠ ∅ do
        u ← Dequeue(Q1)
        for each v ∈ adj[u] do
            if v == 1  and v ∉ list2 then
                Enqueue(Q2, v)
                ListAdd(list2, v)
            end if
        end for
    end while
    while Q2 ≠ ∅ do
        u ← Dequeue(Q2)
        for each v ∈ adj[u] do
            if v == 1  and v ∉ list1 then
                Enqueue(Q1, v)
                ListAdd(list1, v)
            end if
        end for
    end while
end while
```

At the end of this algorithm, we have 2 separate lists, 1 for the humans and 1 for the androids. We do however not know which list contains the humans and which list contains the androids. If we cannot assume that all humanlike entities are connected, as I assumed at the start, then we can simply check if they are all connected. We can simply check at the end, by looping through $|n|$ and check if the entry is in either list1 or list 2. If the entry is in neither, we know that we did not partition it correctly.

This algorithm is functionally correct, because we do a modified BFS, in which we have an adjacency matrix that contains all the edges. We can partition everything correctly, because we are adding every edge to the opposite list and then rinse and repeat, like in BFS.

This algorithm is $\mathcal{O}(n+r)$ because, just like in BFS, we go through every vertex once (n) and the creation of the adjacency matrix is a loop through $|r|$, thus it is $\mathcal{O}(n+r)$.