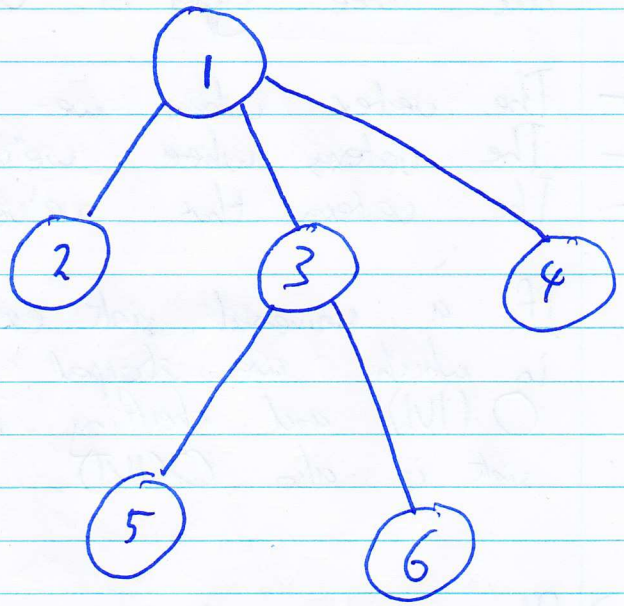
III

1.

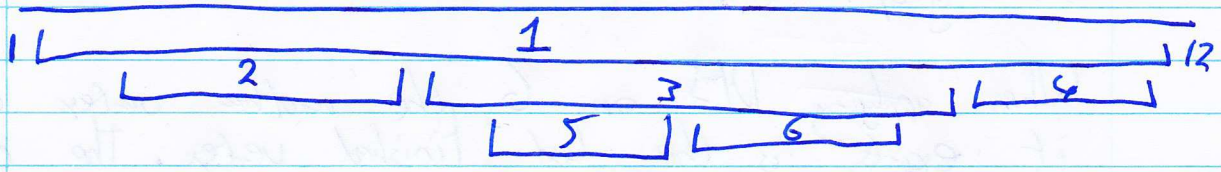
Start = 10

1. d[1] = 1
2. d[2] = 2
3. f[2] = 3
4. d[3] = 4
5. d[5] = 5
6. f[5] = 6
7. d[6] = 7
8. f[6] = 8
9. f[3] = 9
10. d[4] = 10
11. f[4] = 11
12. f[1] = 12

DFS-tree:



Timeline:



2. We can start from a vertex 'a' and try to walk towards the sink. We can do that by picking an edge out of a to another vertex 'b'. We can keep doing this until we've come to a vertex that has no edges coming out of it. (we also never walk back to a vertex we've already been to) Now there are three types of vertices:

- The vertex where we finished
- The vertexes where we've been
- The vertexes that we've not come across

If a universal sink exists, it must be the vertex in which we stopped the walk. The walk was $O(|V|)$ and checking if it is indeed a universal sink is also $O(|V|)$.

3. ~~Als er een vertex is die alle andere vertexes kan bereiken dan betekent dat dat we deze vertex kunnen vinden door DFS op~~

If there exists a vertex u for which all other vertices are accessible, we can find this vertex u by applying DFS on graph G.

When applying DFS on G, the mothe vertex u, if it exists, is the last finished vertex. The only thing we need to do is to keep track of the last finished vertex V.

After this we only need to check if all other vertices are accessible from V.

Algorithm:

DFS (G)
for each vertex $u \in V[G]$
color[u] ← white

Algorithm:

1. Apply DFS and keep track of last finished vertex
2. Run DFS from last finished vertex v and check if all vertices are accessible.

4. Procedure findCycle (G)
for each $x \in V[G]$ do
mark[x] ← white
parent[x] ← nil

for each $x \in V[G]$ do
if x is not white
if (mark[x] = white)
if cc(G, x) = true then
return true

return false

```
Procedure cc(G, x)
    mark[x] ← grey
    for each v ∈ Adj[x]
        if mark[y] == white
            parent[y] ← x
            if cc(G, y) = true
                return true

        else if mark[y] == grey
            display (y)
            z ← x
            while z ≠ y
                display z
                z ← parent[z]
            return true

    mark[x] ← black
    return false
```

5.

DFS-Visit(u):

```
color[u] ← Gray
time ← time + 1
d[u] ← time
for each v ∈ Adj[u]
    if color[v] = White
        then π[v] ← u
        DFS-Visit(v)
        print((u,v) - Tree edge)
    else if color[v] = gray
        print((u,v) - Back edge)
    else if color[v] = black
        if d[v] > d[u]
            print((u,v) - Forward edge)
        else
            print((u,v) - Cross edge
```

```
color[u] ← black
time ← time + 1
f[u] ← time
```