3)

LAB 2

- Strong Connectivity (slide 3)
  - Write a Pseudo-Code

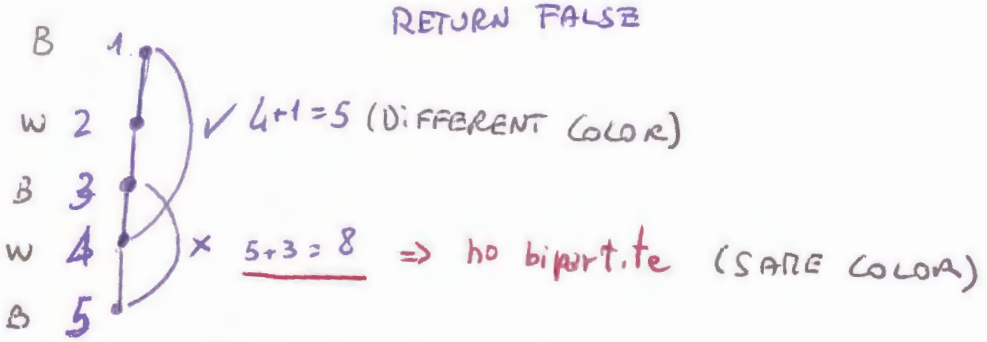
- Bipartite Graph / 2-Colorability
  + write a Pseudo-Code using BFS (ASSUMING THAT THE GRAPH IS CONNECTED)

We run BFS with an extra array, COLOR[n]
s.t. if $(v,v') \in$ edge
if v' is added to $L[i+1]$ then

$$Color[v'] = \begin{cases} blue & \text{if } i+1 \text{ is ODD} \\ white & \text{otherwise} \end{cases}$$

else if Layer[v] = Layer[v'] then
return FALSE

+ WRITE a PseudoCode using DFS-RECURSIVE
  - We use the index associated by DFS to each node
  - if $(v,v') \in G$ and EXPLORED(v) AND $v' \in$ Stack
    v.index + v'.index is EVEN =>
    RETURN FALSE



BIPARTITE-DFS(G)
For each $v \in G$ do v.index = ∅ /* node is not explored
index = 1
2COLOR(v), for some $v \in G$

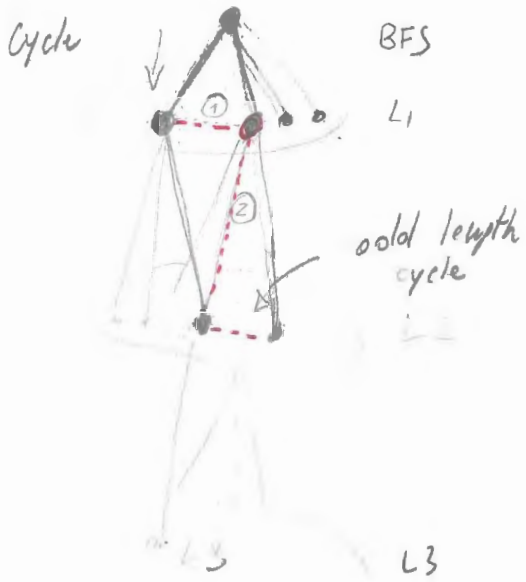
2COLOR(v)
IF ODD index then COLOR[v] = BLUE
ELSE COLOR[v] = WHITE
v.index = index
index = index + 1 ; S.push(v)
For each $(v,v') \in G$
IF v'.index = 0
2COLOR(v')
S.Pop(v) ; index = index + 1

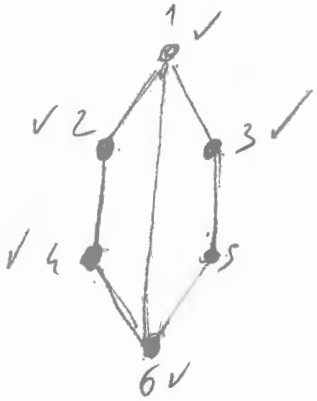
ELSE IF ($v' \in$ Stack ∧ v'.index + v.index is EVEN)
Then Return false

LAB-2

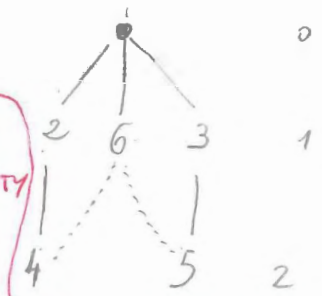
3.1

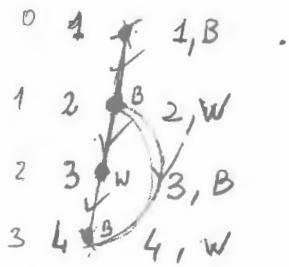
Cycles





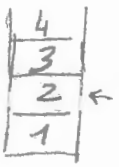
check odd-lenght cycles using DFS => 2 COLORABILITY

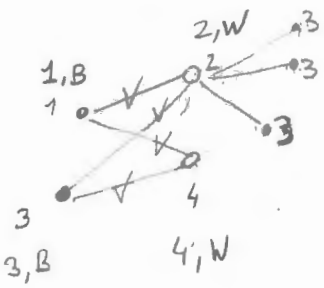




2COLOR(1)
2COLOR(2)
2COLOR(3)
2COLOR(4)
FAIL

INDEX = 1, 2, 3, 4, 5

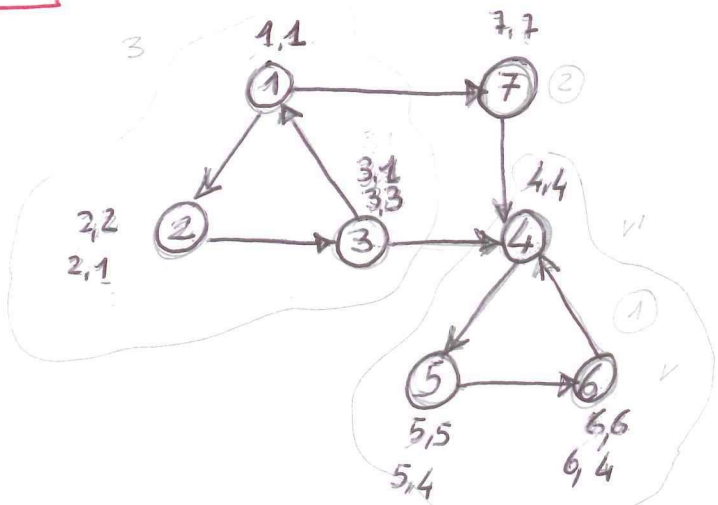




2COLOR(1)
2COLOR(2)
2COLOR(3)
2COLOR(4)

INDEX = 1, 2, 3, 4

LAB 2



i = 0
i = 1
i = 2
i = 3

DIRECTED Graphs
STRONG - COMPONENTS
TARJAN - Algorithm

CASE WHERE THE ADJ for node 3 is : ADJ[3] = {4,1}

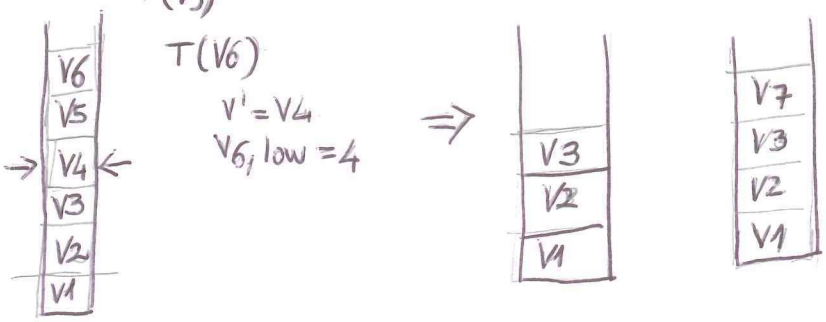
T(V4) ← OUT: {4,5,6}

T(V5)

T(V6)

V' = V4

V6, low = 4



T(1)

T(2)

T(3)

V' = 1

V3.min = 1

T(1)

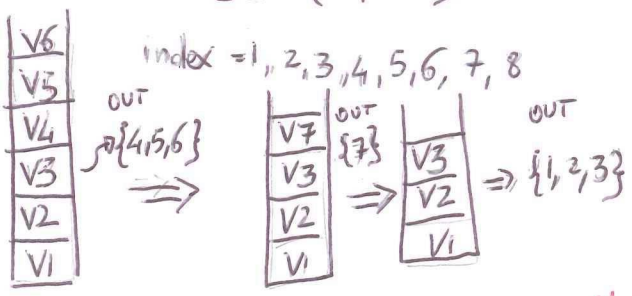
OUT = {3,2,1}

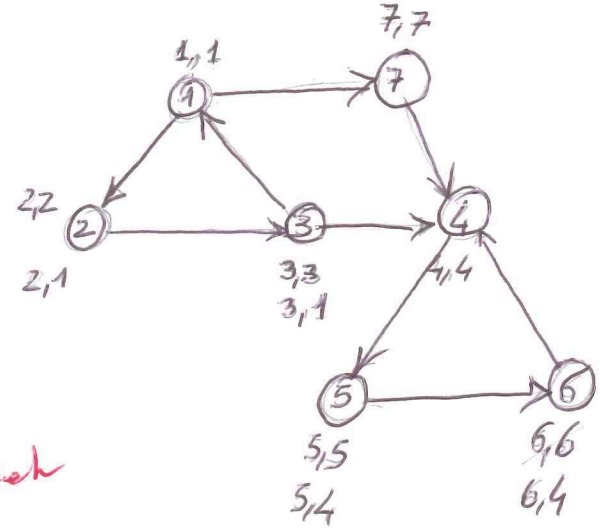
T(1)

T(7) ← OUT

{7}

CASE WHERE ADJ[3] = {1, 4}





Show:

1. Nesting of recursive calls
2. Value of the variables at each recursive call
3. Intermediate OUTPUT
4. Stack.

Vedere 2-SAT using TARJAN

see: https://renzoggd.wordpress.com/2014/01/17/2-satisfiability/

LAB 3

1. Topological Order

1.1. Modify the Algorithm in the slides (3.1) ~~with~~ adding the array count [w] and a Stack S containing the nodes without incoming edges.
Assume we can check, given ~~a~~ vertices w, ✔, whether ~~there~~ $(v, w) \in G$.

1.2 Run the algorithm on the following DAG