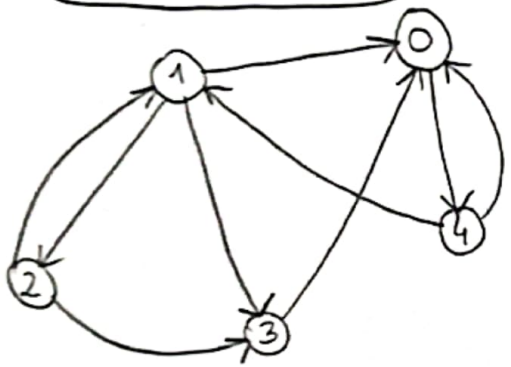
= LAB 2 =

Problem 1: Lowest length path between <start> and <end> using forward BFS from the starting vertex.

EXAMPLE 1



$V = \{0, 1, 2, 3, 4\}$; $E = \{(0,4), (1,0), (1,2), (1,3), (2,1), (2,3), (3,0), (3,2), (4,0), (4,1)\}$

We'll run lowest_length_path(1,4).

Firstly, the function <lowest_length_path> runs a modified version of BFS starting from <start> and ending at <end>.

--dict_out : {0:[4], 1:[0,2,3], 2:[1,3], 3:[0,2], 4:[0,1]}

start=1, end=4	queue	top-of-queue	neighbour	visited-dictionary	dist-dictionary	pred-dictionary
	← 1 ←	—	—	k: 0 1 2 3 4 v: F T F F F	k: 0 1 2 3 4 v: ∞ 0 ∞ ∞ ∞	k: 0 1 2 3 4 v: None None None None None
		1	0	0 1 2 3 4 T T F F F	0 1 2 3 4 1 0 ∞ ∞ ∞	0 1 2 3 4 1 None None None None
	← 0 ←		2	0 1 2 3 4 T T T F F	0 1 2 3 4 1 0 1 ∞ ∞	0 1 2 3 4 1 None 1 None None
	← 0 2 ←		3	0 1 2 3 4 T T T T F	0 1 2 3 4 1 0 1 1 ∞	0 1 2 3 4 1 None 1 1 None
	← 0 2 3 ←					

start = 1, end = 4

queue	top-of-queue	neighbour	visited-dictionary	dist-dictionary	prev-dictionary
← 2 3 ←	0	4	0 1 2 3 4 / T T T T T	0 1 2 3 4 / 1 0 1 1 2	0 1 2 3 4 / 1 None 1 1 0

neighbour == end_vertex (4) → STOP. The BFS returns the 3 dictionaries: <visited>, <prev>, and <dist>. The <lowest_length_path> function will use <visited> and <prev> to determine the existance of a path between the 2 vertices and to find the lowest length path between the 2 vertices.

The reverse path is built from prev :

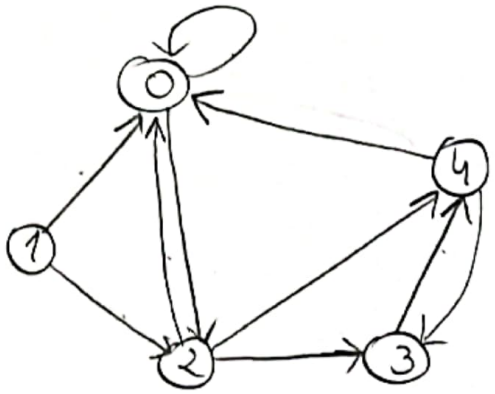
0	1	2	3	4
1	None	1	1	0

starting from end = 4

prev[4] = 0, prev[0] = 1, prev[1] = None STOP.

→ reverse_path = [1, 0, 4] ← this list will be returned by the function

EXAMPLE 2



$V = \{0, 1, 2, 3, 4\}$; $E = \{(0,0), (0,2), (1,0), (1,2), (2,0), (2,3), (2,4), (3,4), (4,0), (4,3)\}$

We'll run lowest_length_path(0, 3).

adj_out: {0:[0,2], 1:[0,2], 2:[0,3,4], 3:[4], 4:[0,3]}

start = 0, end = 3

queue	top_of_queue	neighbour	visited	dist	prev
[0]	—	—	0 1 2 3 4 / T F F F F	0 1 2 3 4 / 0 ∞ ∞ ∞ ∞	0 1 2 3 4 / None None None None None
[2]	0	2	0 1 2 3 4 / T F T F F	0 1 2 3 4 / 0 ∞ 1 ∞ ∞	0 1 2 3 4 / None None 0 None None
[3, 4]	2	3, 4	0 1 2 3 4 / T F T T F	0 1 2 3 4 / 0 ∞ 1 2 ∞	0 1 2 3 4 / None None 0 2 None

neighbour == end_vertex(3) → STOP

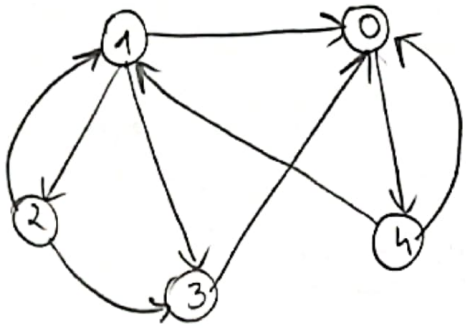
prev:

0	1	2	3	4
None	None	0	2	None

prev[3] = 2; prev[2] = 0; prev[0] = None → reverse_path = [0, 2, 3]

Problem 1 Bonus : Find the strongly connected components of a graph in $O(n+m)$.

EXAMPLE 1



$V = \{0, 1, 2, 3, 4\}$; $E = \{(0,4), (1,0), (1,2), (1,3), (2,1), (2,3), (3,0), (3,2), (4,0), (4,1)\}$

-- dict_out : $\{0:[4],\ 1:[0,2,3],\ 2:[1,3],\ 3:[0,2],\ 4:[0,1]\}$

dfs 1:

stack	visited	vertex (1)
[ ]	[False] * 5	0
	[T, F, F, F, F]	4
	[T, F, F, F, T]	1
	[T, T, F, F, T]	2
	[T, T, T, F, T]	3
	[T, T, T, T, T]	

stack	visited	vertex (2)
[3]	[T] * 5	3
[3, 2]		2
[3, 2, 1]		1
[3, 2, 1, 4]		4
[3, 2, 1, 4, 0]		0

stack = [3, 2, 1, 4, 0]

transpose_graph() ⇒ --dict_out : {0:[1,3,4], 1:[2,4], 2:[1,3], 3:[1,2], 4:[0]}

dfs2:

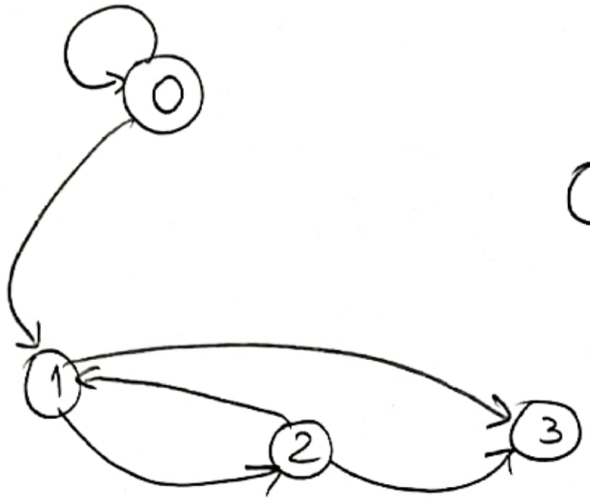
stack	vertex	visited	scc
[3, 2, 1, 4, 0]		[False] * 5	[ ]
[3, 2, 1, 4]	0	[T, F, F, F, F]	[[ ]]
	1	[T, T, F, F, F]	[[0]]
	2	[T, T, T, F, F]	[[0, 1]]
	3	[T, T, T, T, F]	[[0, 1, 2]]
	2		[[0, 1, 2, 3]]
	1		
	4	[T, T, T, T, T]	[[0, 1, 2, 3, 4]]

⇒ The strongly connected components of the example graph are:

[[0, 1, 2, 3, 4]]

(just one scc)

EXAMPLE 2



④

- dict_out : { 0 : [0, 1], 1 : [2, 3], 2 : [1, 3], 3 : [ ], 4 [ ] }

dfs 1 :

stack	visited	vertex ①
[ ]	[False] * 5	0
	[T, F, F, F, F]	1
	[T, T, F, F, F]	2
	[T, T, T, F, F]	3
	[T, T, T, T, F]	
[3]		

stack	visited	vertex ②
[3]		2
[3, 2]		1
[3, 2, 1]		0
[3, 2, 1, 0]		4
[3, 2, 1, 0, 4]		

stack = [3, 2, 1, 0, 4] transpose-graph ⇒ dict_out: {0: [0], 1: [0, 2], 2: [1], 3: [1, 2], 4: []}

dfs 2

stack	vertex	visited	scc
[3, 2, 1, 0, 4]	4	[F, F, F, F, F]	[[ ]]
[3, 2, 1, 0]		[F, F, F, F, T]	[[4]]
[3, 2, 1]	0	[T, F, F, F, T]	[[4], [ ]]
[3, 2]	1	[T, T, F, F, T]	[[4], [0]]
	2	[T, T, T, F, T]	[[4], [0], [ ]]
[3]	2		[[4], [0], [1]]
[ ]	3	[T, T, T, T, T]	[[4], [0], [1, 2]]
			[[4], [0], [1, 2], [ ]]
			[[4], [0], [1, 2], [3]]

⇒ The strongly connected components are:
[[4], [0], [1, 2], [3]]