Loops solving in OpenRC

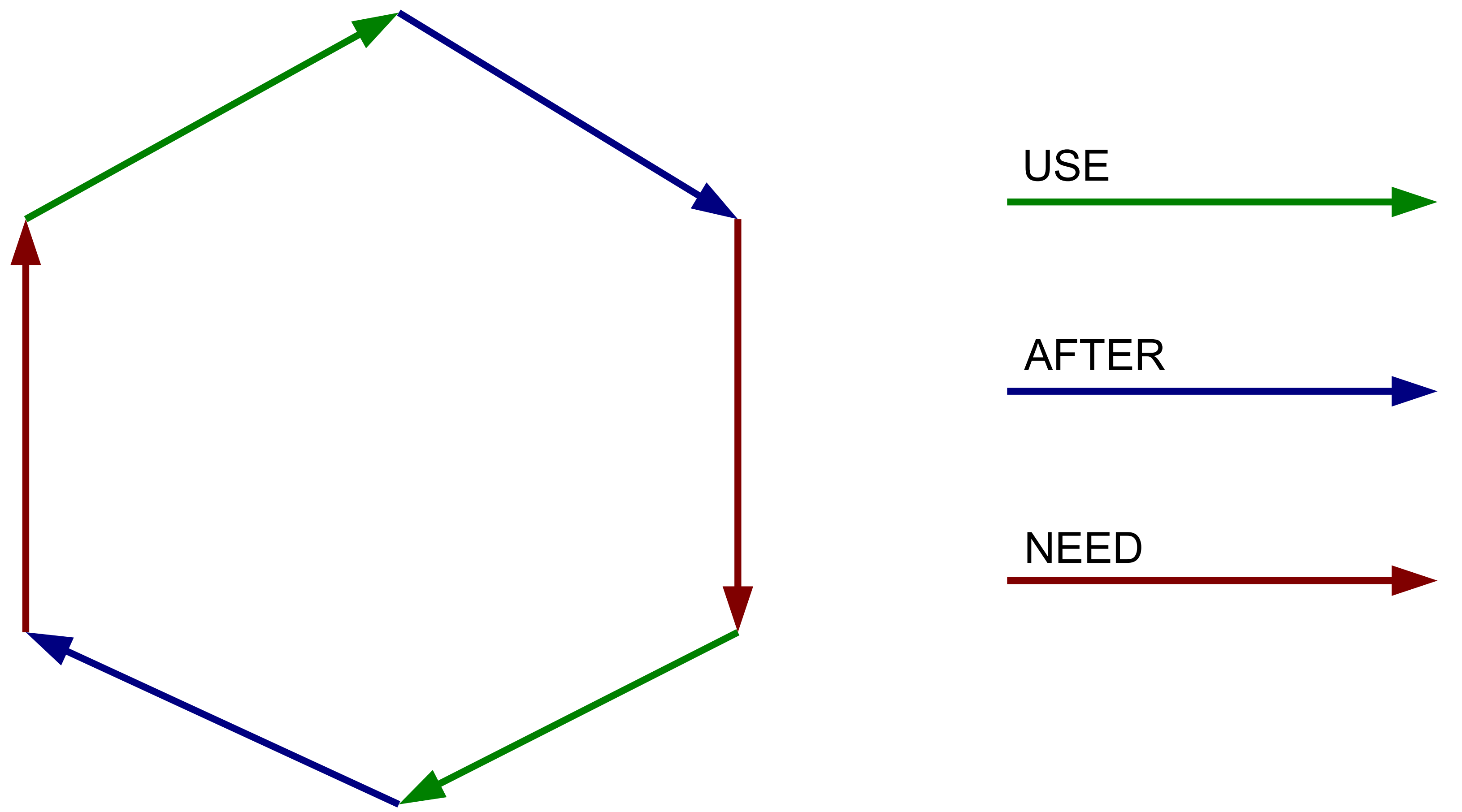
Dmitry Yu Okunev <dyokunev@ut.mephi.ru> 0x8E30679C

Explanation of method that I used to detect loops in my patch for OpenRC “early loop detector”.

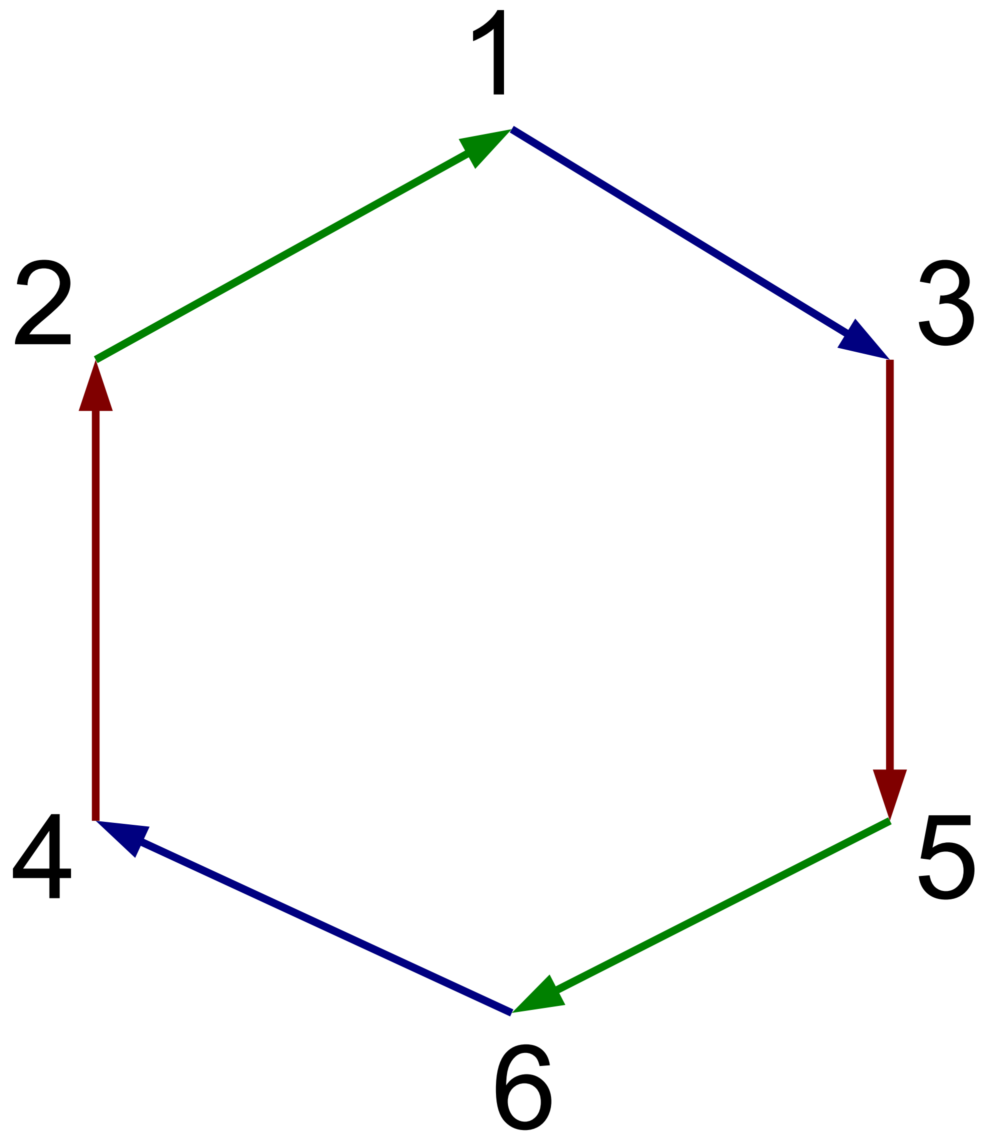
I don't know how is this algorithm called, because I solved the problem on piece of paper by myself. Sorry…

Also please sorry for my English.

Let's imagine a simple loop

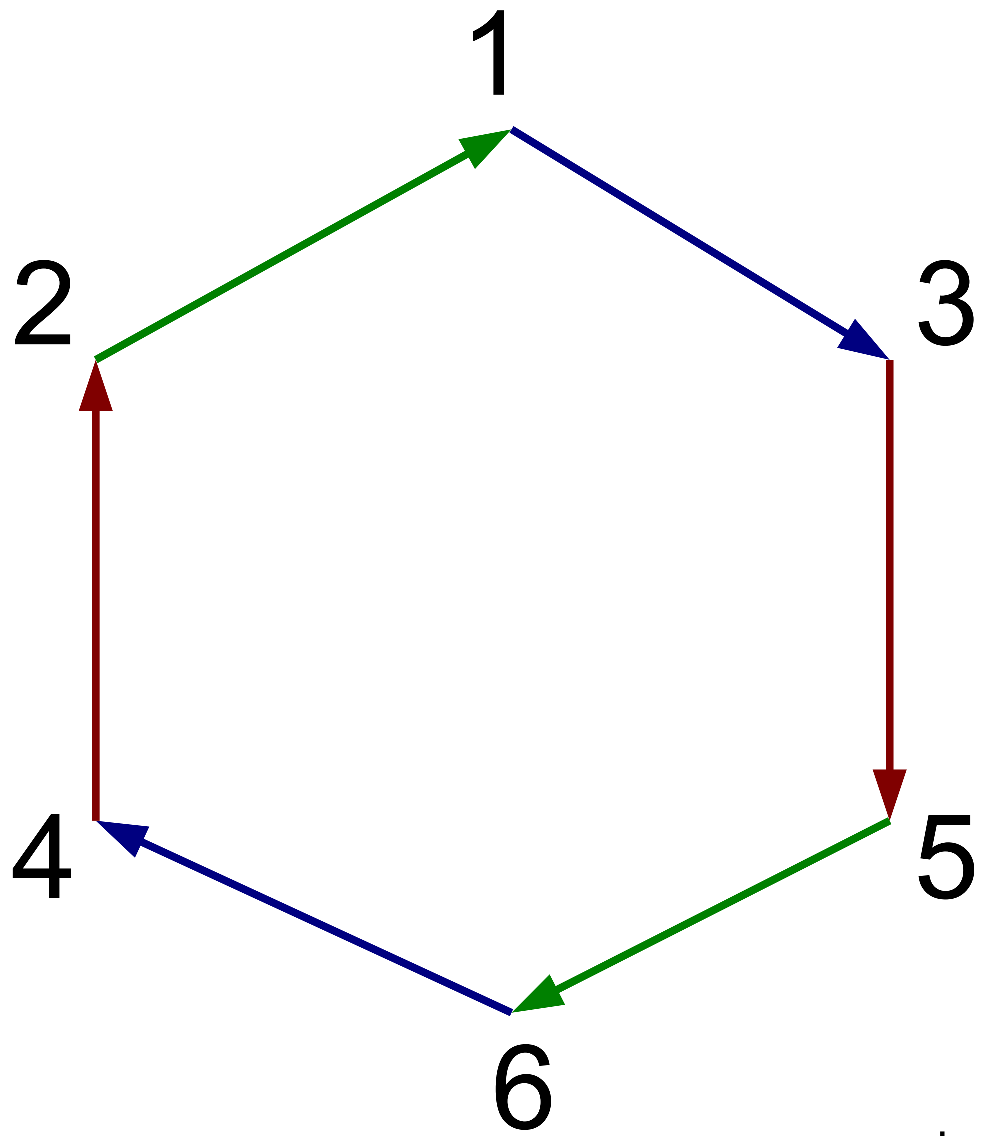


Enumerating vertices (services)



Numbering "doesn't know" anything about the loop, so it can not be done sequentially along the loop chain.

Building dependency pre-matrixes: “use”, “after” and “need”



1 →	1 → 3	1 →
2 → 1	2 →	2 →
3 →	3 →	3 → 5
4 →	4 →	4 → 2
5 → 6	5 →	5 →
6 →	6 → 4	6 →

Result “use” matrix:

0						
1	1					
0						
0						
1	6					
0						

number of dependencies

dependencies

Building mixed pre-matrix for all dependency types.

$$\begin{tikzcd}
 & 1 \arrow[dr] & \\
2 \arrow[ur] & & 3 \arrow[d] \\
4 \arrow[u] & & 5 \arrow[dl] \\
 & 6 \arrow[ul] &
\end{tikzcd}$$

1 →	1 → 3	1 →		1 → 3
2 → 1	2 →	2 →		2 → 1
3 →	3 →	3 → 5	➡	3 → 5
4 →	4 →	4 → 2		4 → 2
5 → 6	5 →	5 →		5 → 6
6 →	6 → 4	6 →		6 → 4

Result mixed (pre-)matrix:

1	3					
1	1					
1	5					
1	2					
1	6					
1	4					

number of dependencies

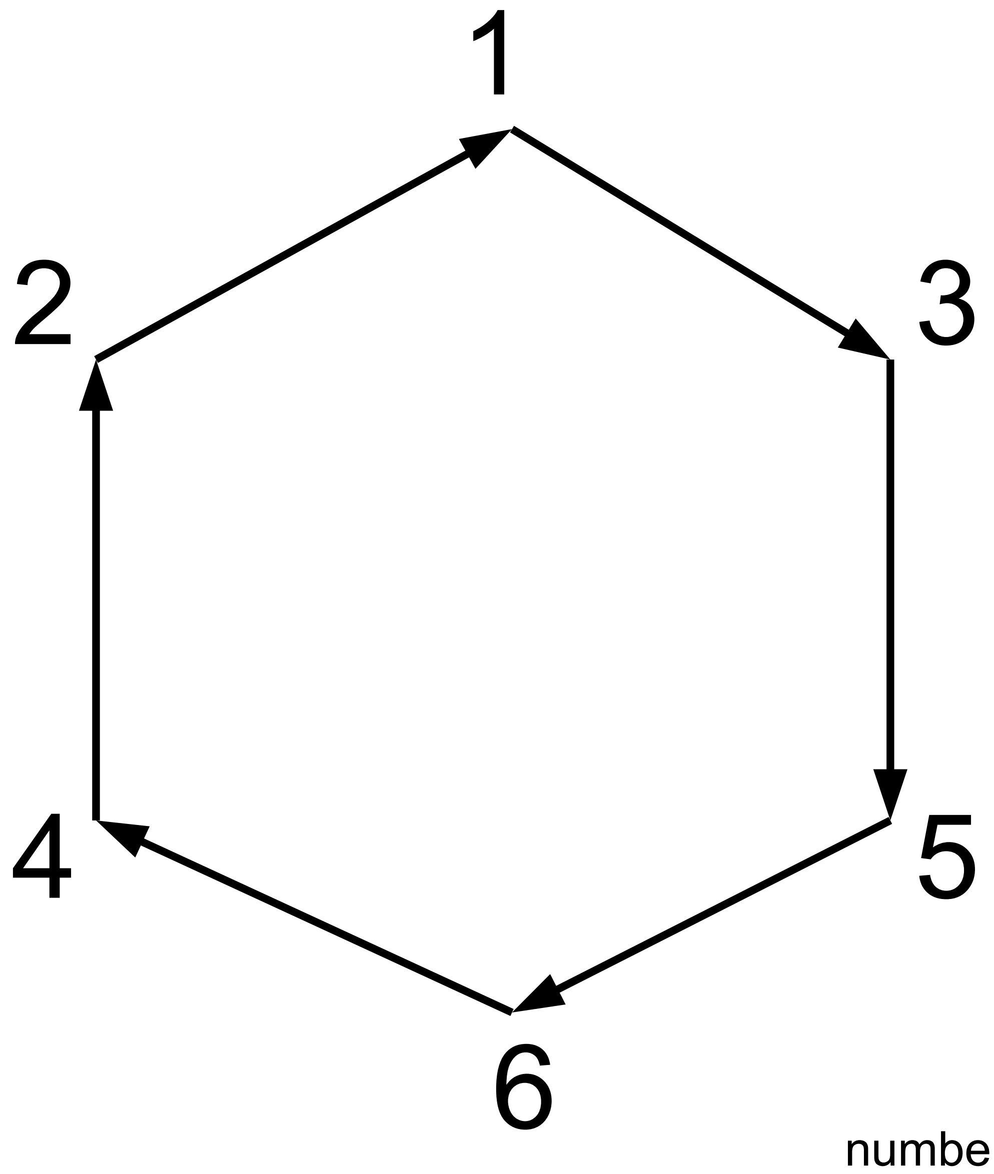
dependencies

Building matrix for all dependencies for every service (vertex)

$$\begin{tikzcd}
& 1 \arrow[dr] & \\
2 \arrow[ur] & & 3 \arrow[d] \\
4 \arrow[u] & & 5 \arrow[dl] \\
& 6 \arrow[ul] &
\end{tikzcd}$$

Loop through each vertex, and look into depending dependencies, this complementing their own dependencies.

Building matrix for all dependencies for every service (vertex)



The 1-st step

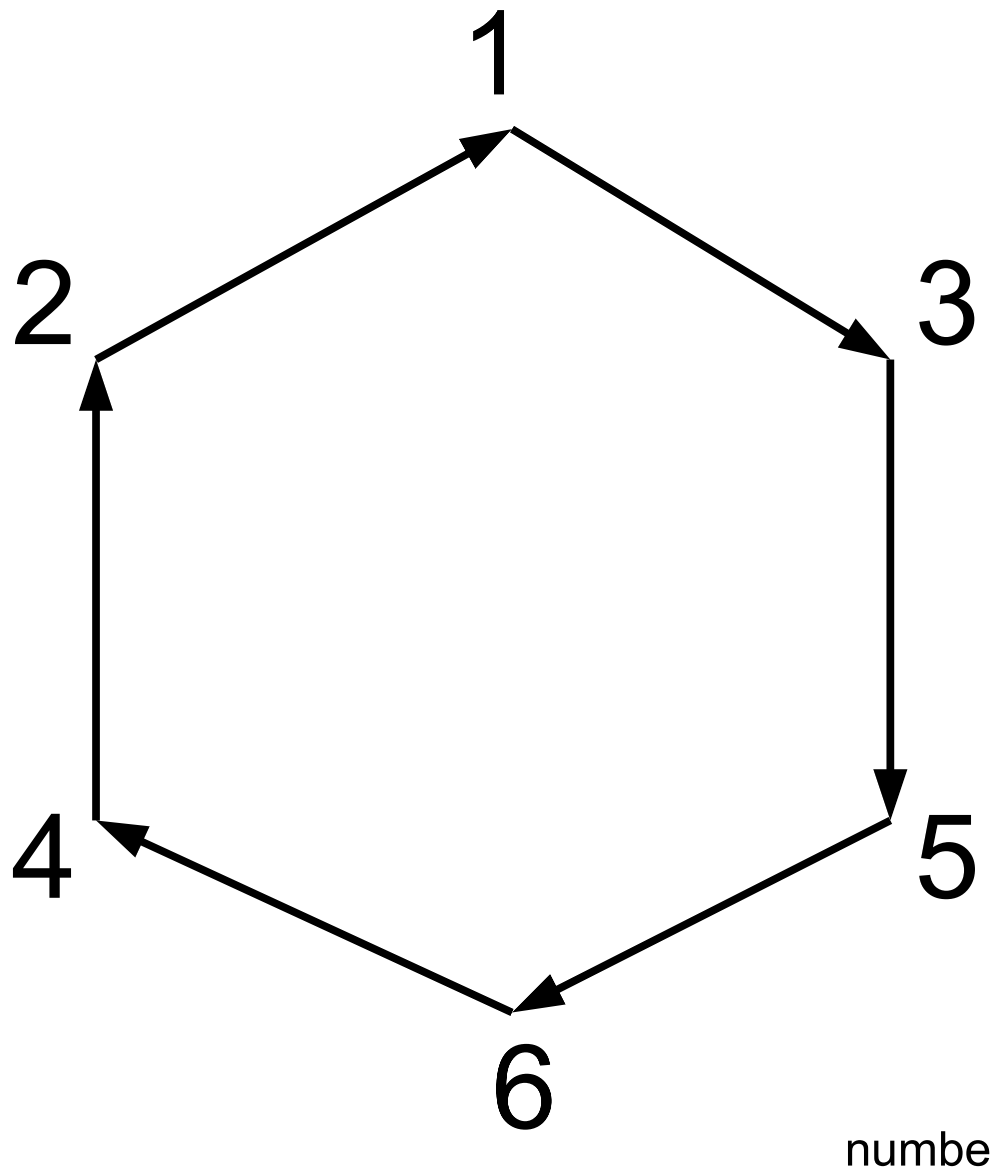
Result mixed matrix:

1	2	3	5				
2	1	1					
3	1	5					
4	1	2					
5	1	6					
6	1	4					

number of dependencies

dependencies

Building matrix for all dependencies for every service (vertex)



The 2-nd step

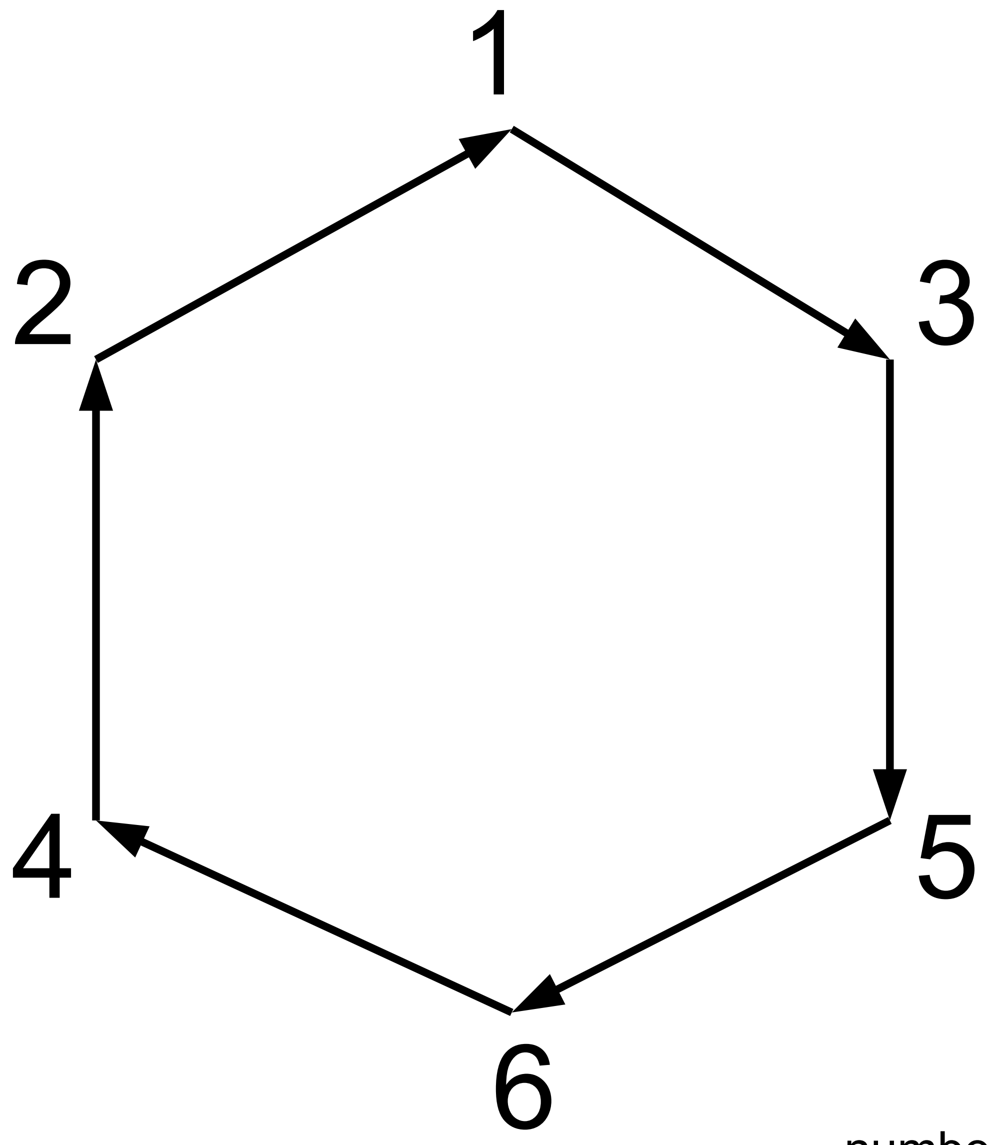
Result mixed matrix:

1	2	3	5		
2	3	1	3	5	
3	1	5			
4	1	2			
5	1	6			
6	1	4			

number of dependencies

dependencies

Building matrix for all dependencies for every service (vertex)



The 3-rd step

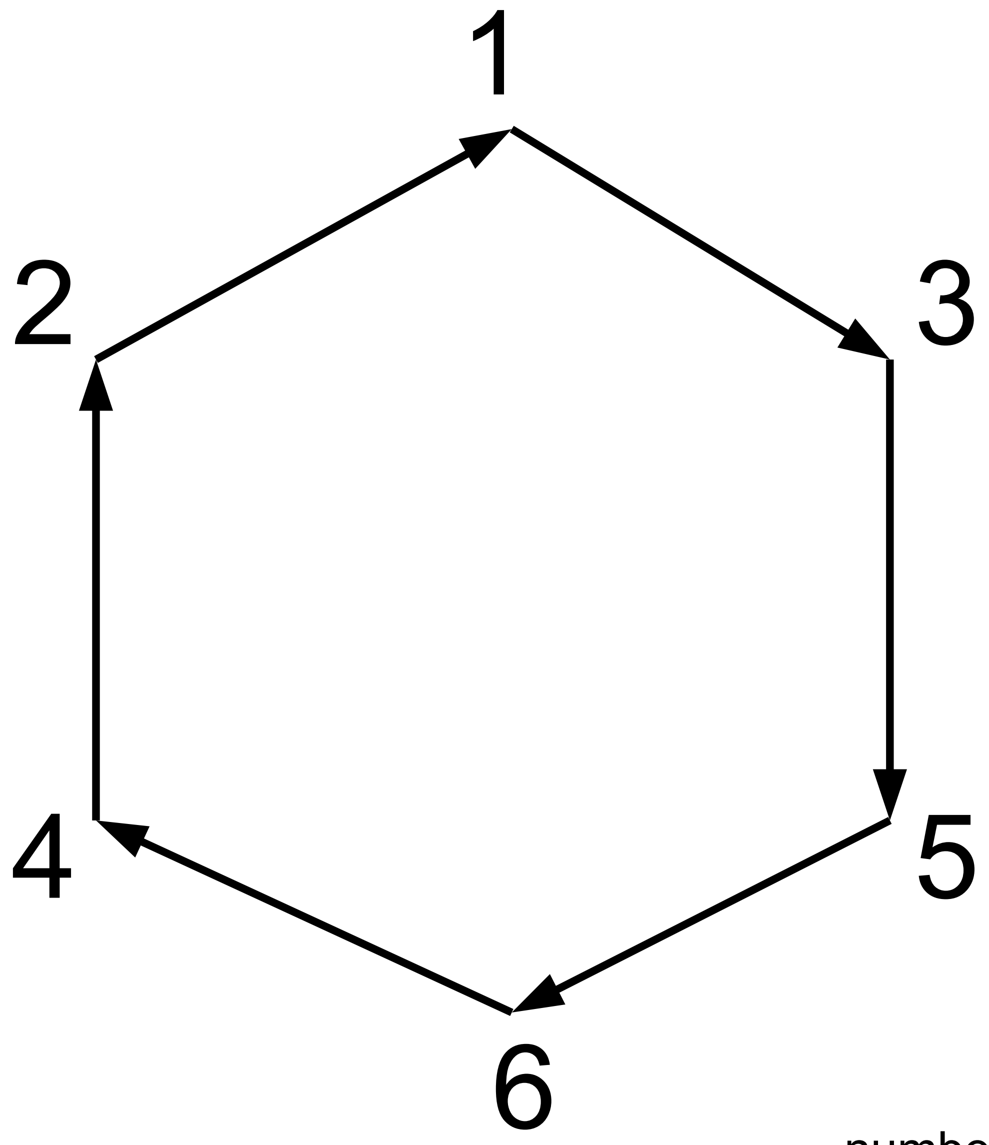
Result mixed matrix:

1	2	3	5	
2	3	1	3	5
3	2	5	6	
4	1	2		
5	1	6		
6	1	4		

number of dependencies

dependencies

Building matrix for all dependencies for every service (vertex)



The 6-th step

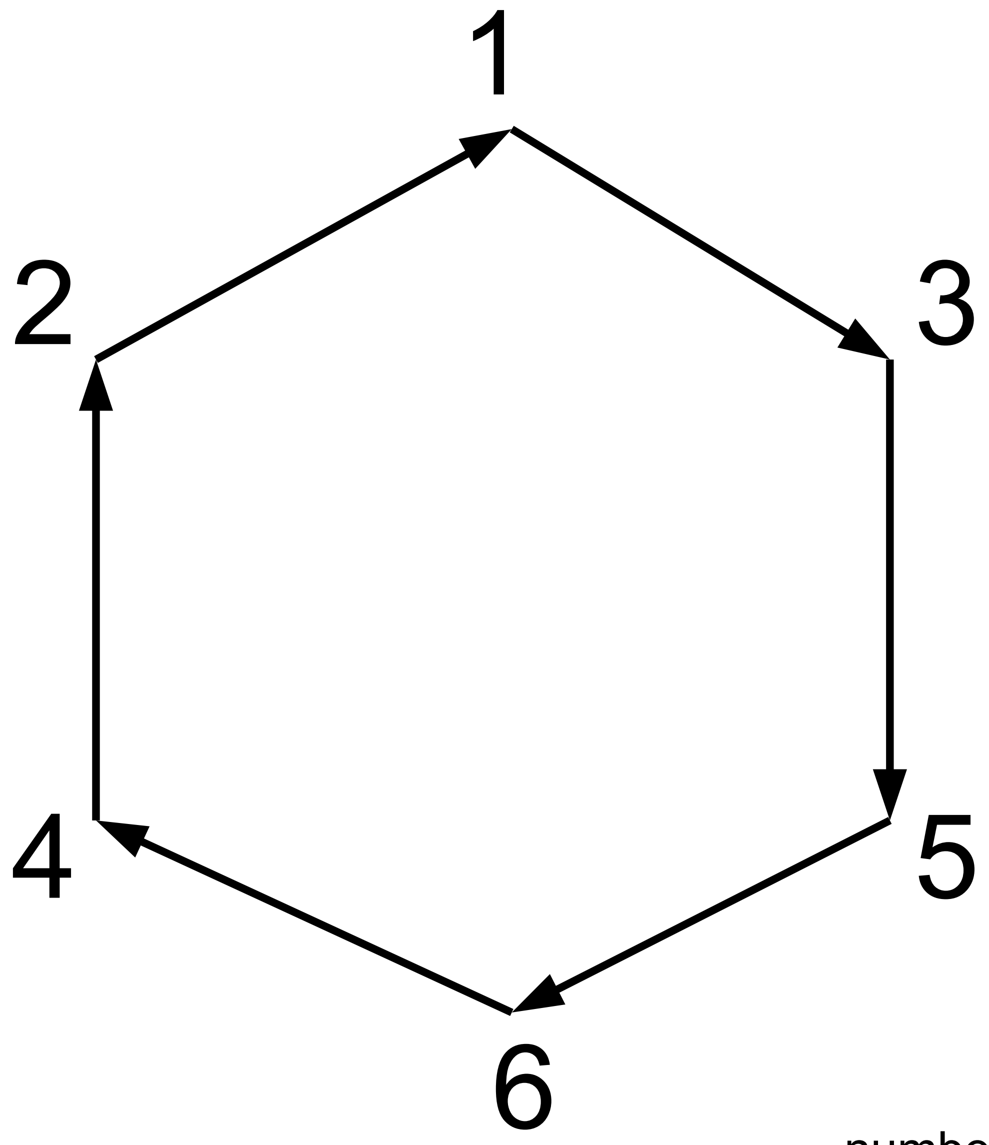
Result mixed matrix:

1	2	3	5			
2	3	1	3	5		
3	2	5	6			
4	4	2	1	3	5	
5	2	6	4			
6	5	4	2	1	3	5

number of dependencies

dependencies

Building matrix for all dependencies for every service (vertex)



As you can see the last verticles have more information about their dependencies that the first one. That's why it's more optimal to do reverse-way iteration instead of one more direct.

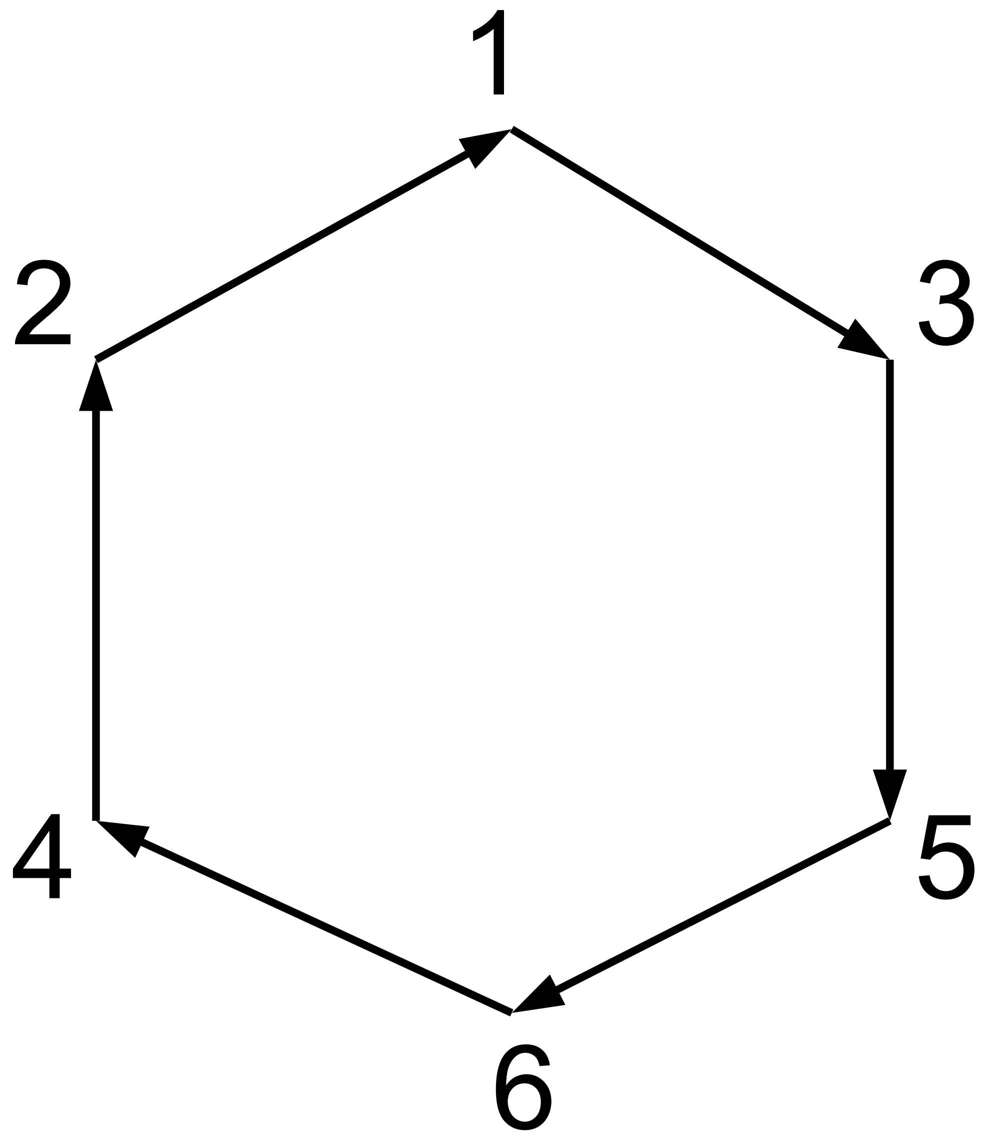
Result mixed matrix:

1	2	3	5			
2	3	1	3	5		
3	2	5	6			
4	4	2	1	3	5	
5	2	6	4			
6	5	4	2	1	3	5

number of dependencies

dependencies

Building matrix for all dependencies for every service (vertex)



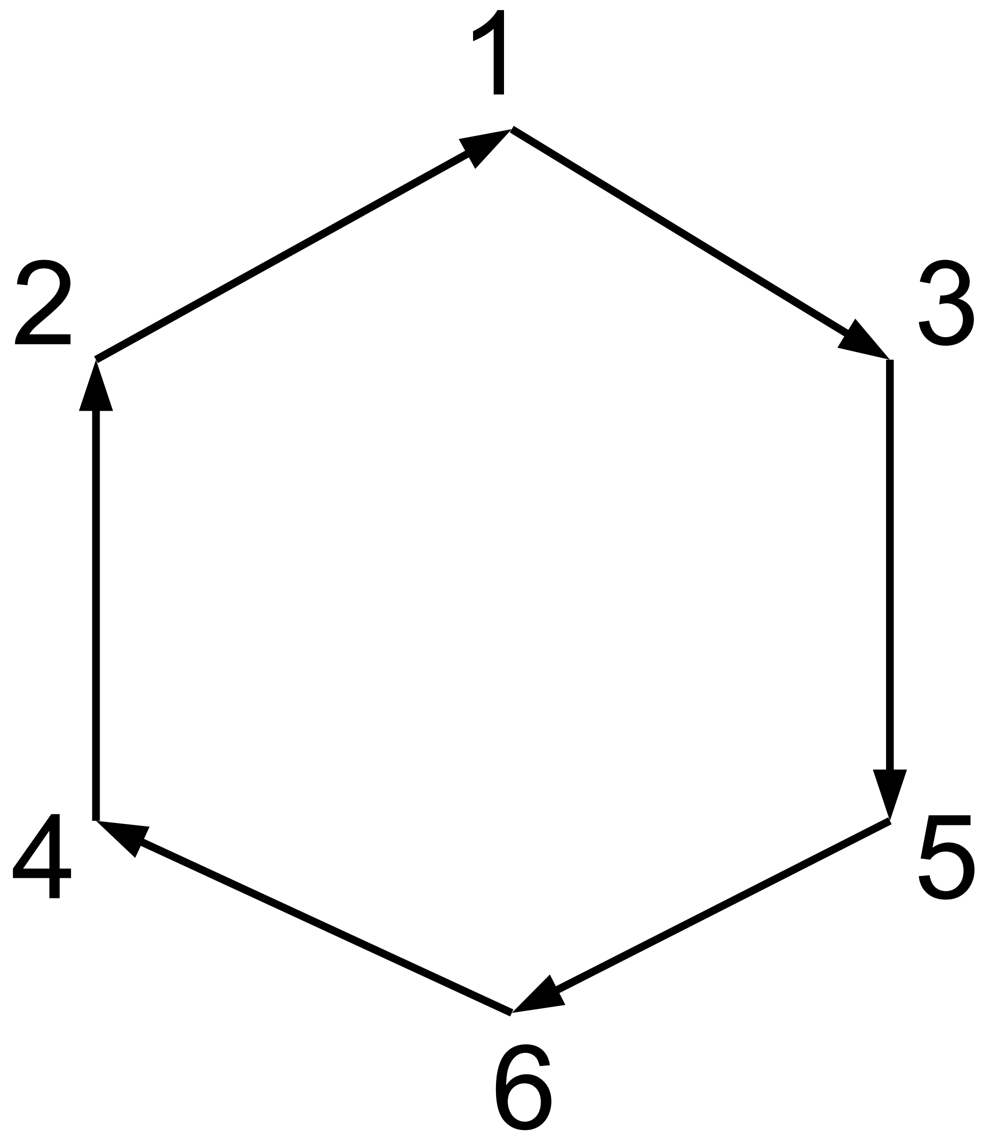
The 1-st step of the reverse-way iteration

Result mixed matrix:

1	2	3	5				
2	3	1	3	5			
3	2	5	6				
4	4	2	1	3	5		
5	2	6	4				
6	6	4	2	1	3	5	**6**

You can see a loop right here

Building matrix for all dependencies for every service (vertex)



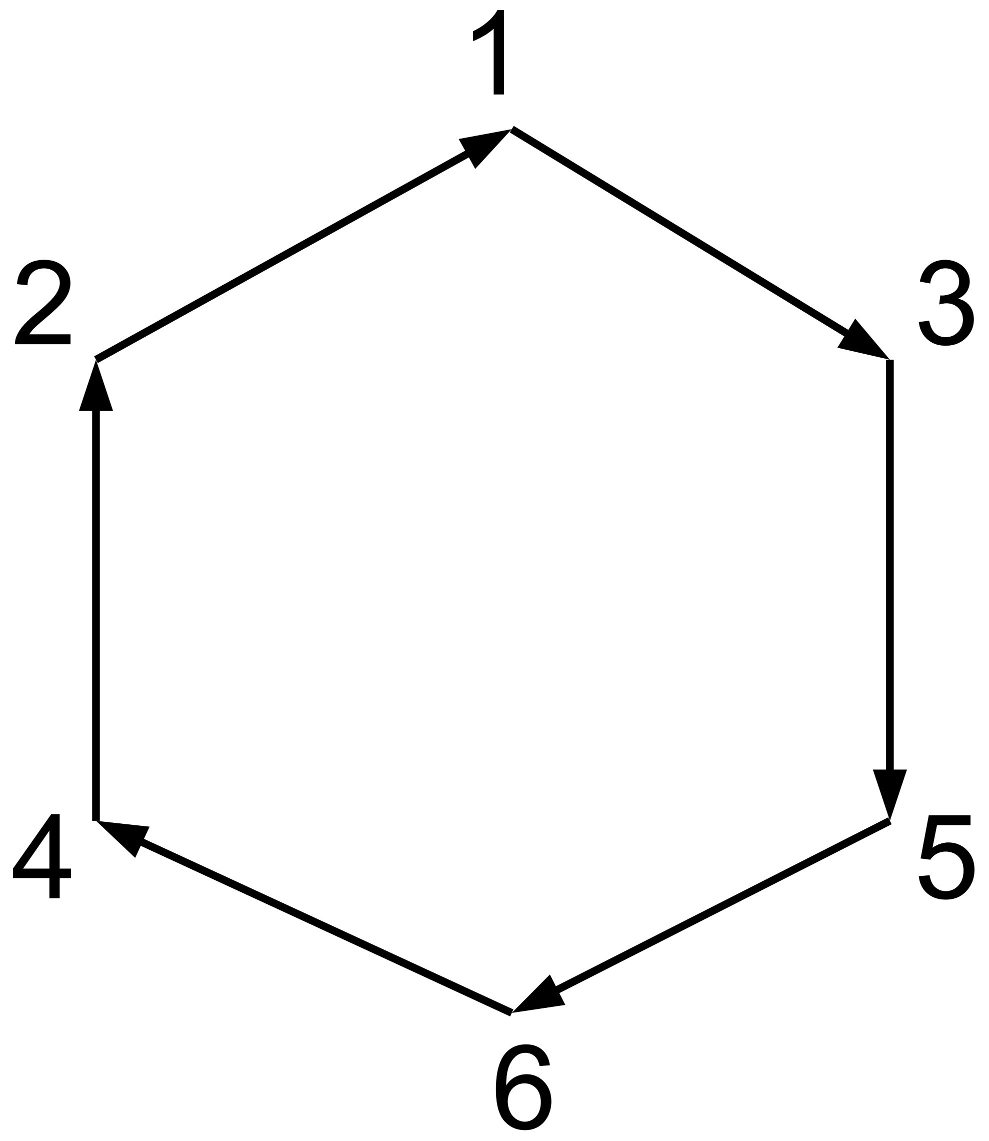
The 2-nd step of the reverse-way iteration

Result mixed matrix:

1	2	3	5				
2	3	1	3	5			
3	2	5	6				
4	4	2	1	3	5		
5	6	6	4	2	1	3	**5**
6	6	4	2	1	3	5	**6**

and here

Building matrix for all dependencies for every service (vertex)



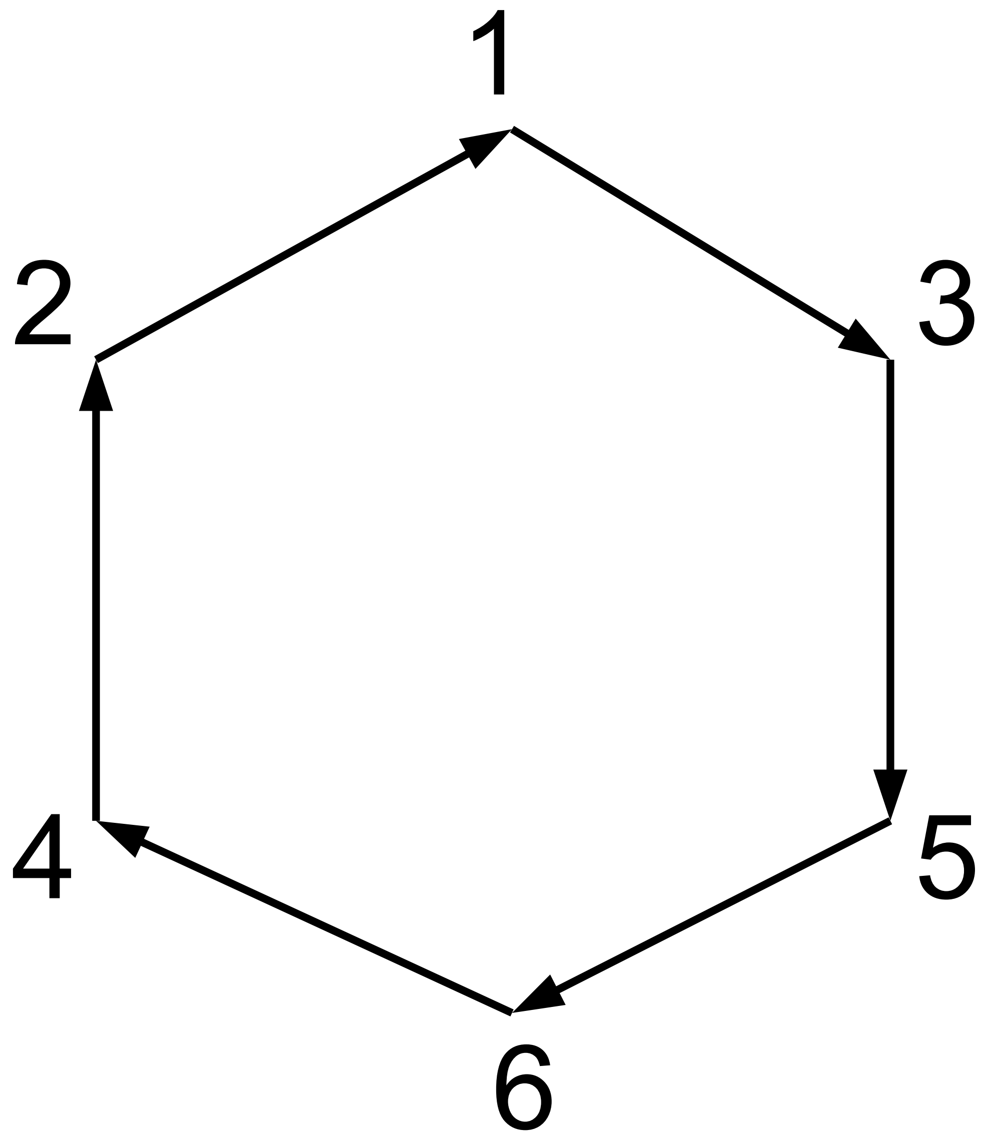
The 3-rd step of the reverse-way iteration

Result mixed matrix:

1	2	3	5				
2	3	1	3	5			
3	2	5	6				
4	6	2	1	3	5	6	4
5	6	6	4	2	1	3	**5**
6	6	4	2	1	3	5	**6**

and here...

Building matrix for all dependencies for every service (vertex)



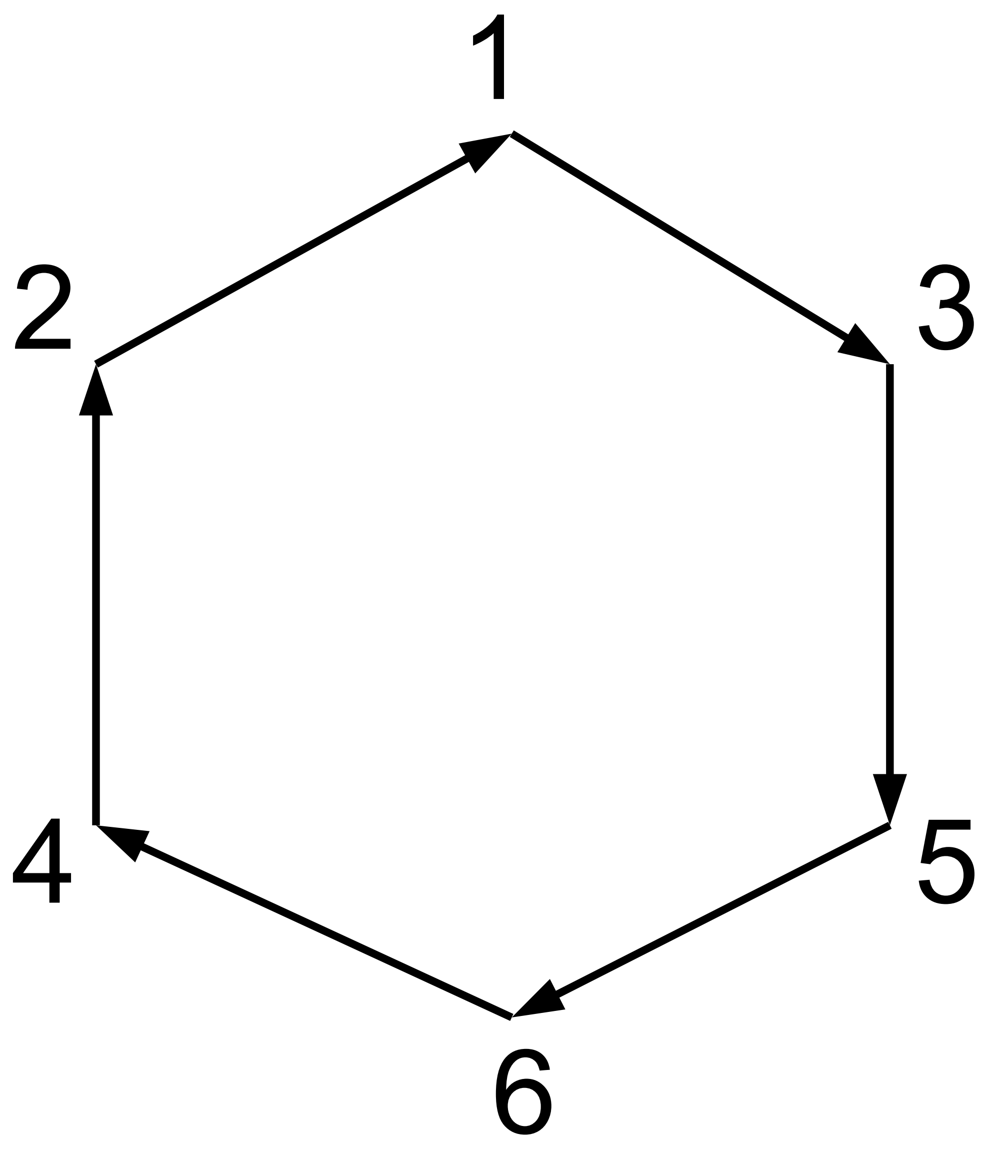
The 6-th step of the reverse-way iteration

Result mixed matrix:

1	6	3	5	6	4	2	1
2	6	1	3	5	6	4	2
3	6	5	6	4	2	1	3
4	6	2	1	3	5	6	4
5	6	6	4	2	1	3	5
6	6	4	2	1	3	5	6

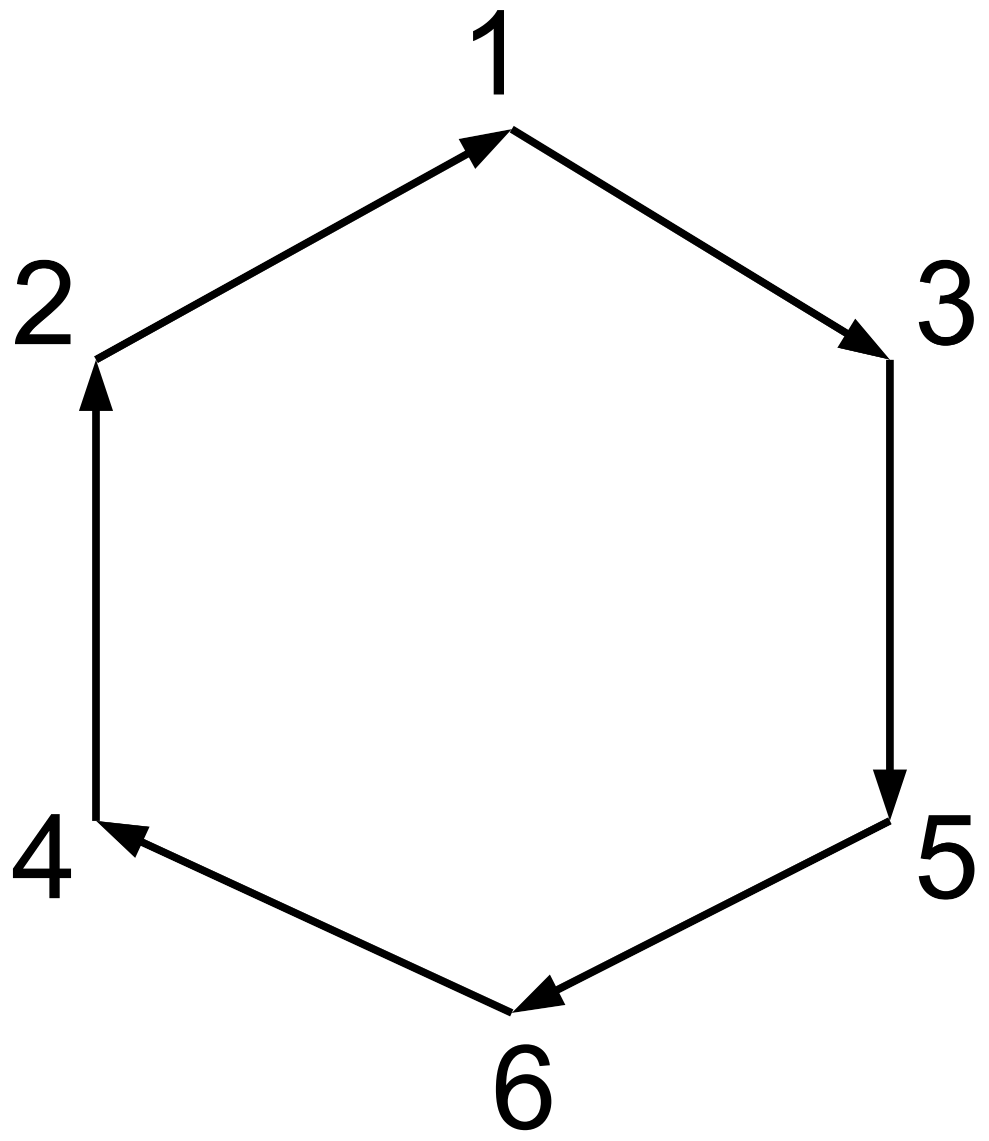
and even here

Building matrix for all dependencies for every service (vertex)



I have a hypothesis, that the only one full iteration is enough to detect any loop. But I have no prove of that. So to be sure, “direct+reverse” iterations are repeating until no modifications will be made into the matrix

Detecting the loops

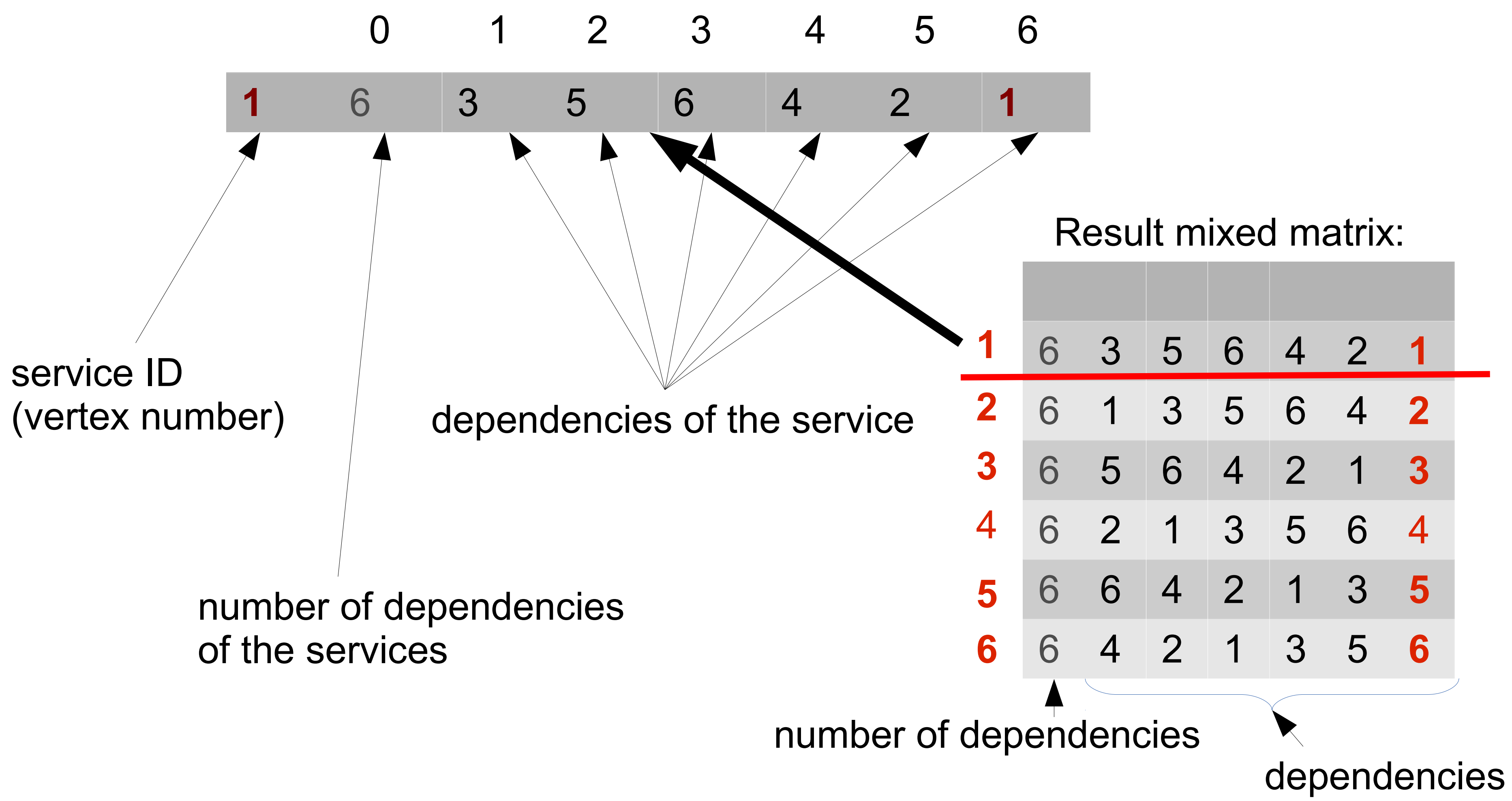


Detecting the loops after that is very simple task. It's just need to check if service (vertex) is depended on itself.

Result mixed matrix:

1	6	3	5	6	4	2	**1**
2	6	1	3	5	6	4	**2**
3	6	5	6	4	2	1	**3**
4	6	2	1	3	5	6	4
5	6	6	4	2	1	3	**5**
6	6	4	2	1	3	5	**6**

Solving the loops



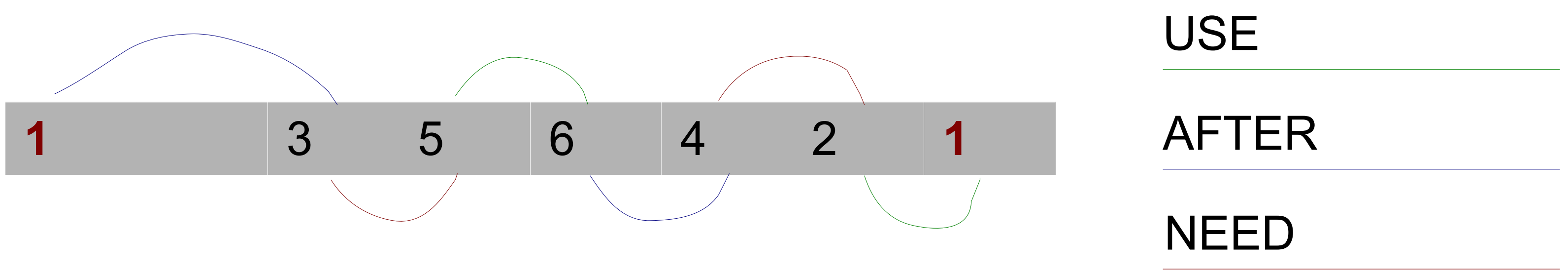
	0	1	2	3	4	5	6
1	6	3	5	6	4	2	1

Result mixed matrix:

1	6	3	5	6	4	2	1
2	6	1	3	5	6	4	2
3	6	5	6	4	2	1	3
4	6	2	1	3	5	6	4
5	6	6	4	2	1	3	5
6	6	4	2	1	3	5	6

Solving the loops

Any later dependency in a line of the matrix may be caused only by the earlier one (or by the service itself). So using pre-matrixes of dependencies of any type we can restore the picture of dependencies.



So the solver is searching a dependency to break with next rules:

- Try to break “use” dependency if it's possible, otherwise “after” dependency.
- Between dependencies of the same type, remove dependency with the least number of the parent dependencies in this chain.
- Ceteris paribus, break dependency the nearest to the service.

In this example the broken dependency will be 5 $\rightarrow$ 6.