POSCAT Seminar 9 : Graph 1

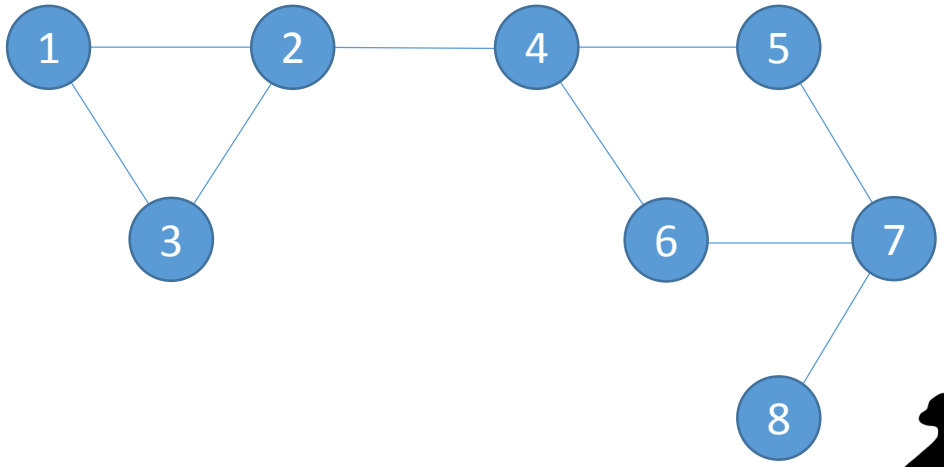
yougatup @ POSCAT

Topic

- Topic today
  - Basic concept
  - Graph representation
    - Adjacency matrix
    - Adjacency list
    - Tradeoffs
  - Graph Traversal
    - Depth First Search
    - Breadth First Search
    - Flood Fill
    - Connected Component
    - Strongly Connected Component

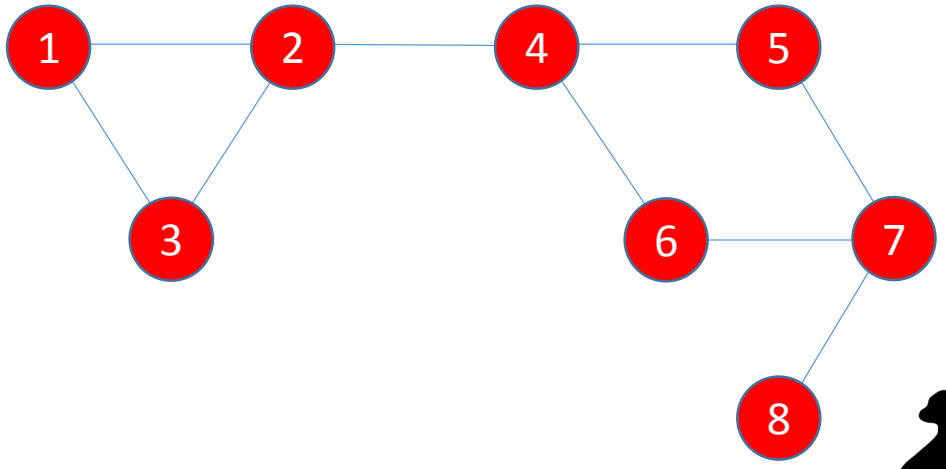
Graph

- Basic Concept
  - $G = (V, E)$
  - A set of vertices and edges



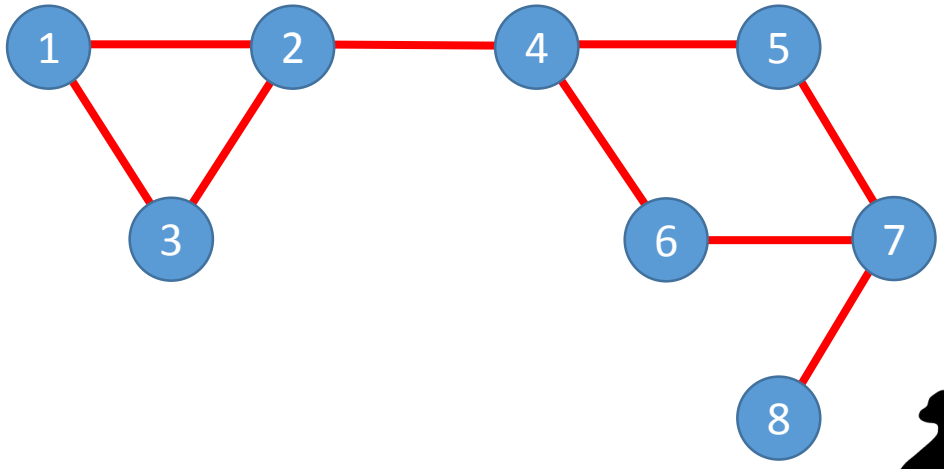
Graph

- Terminology
  - Nodes



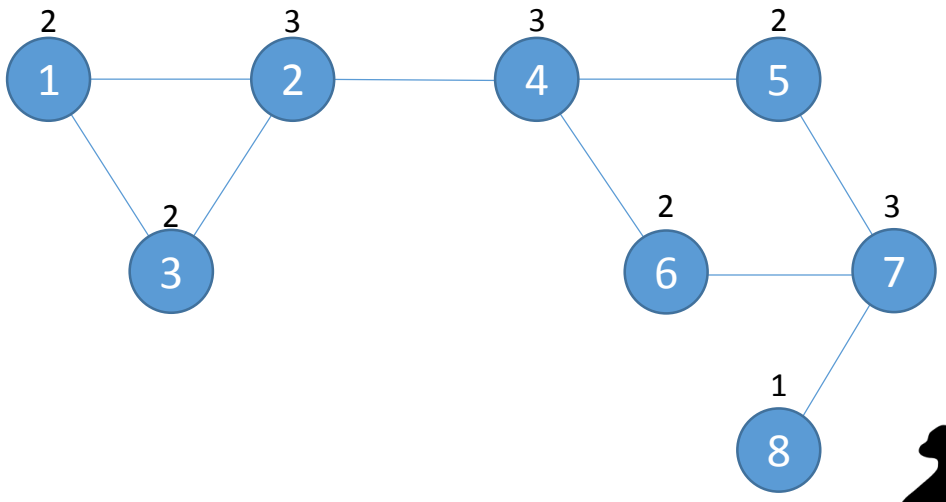
Graph

- Terminology
  - Edges



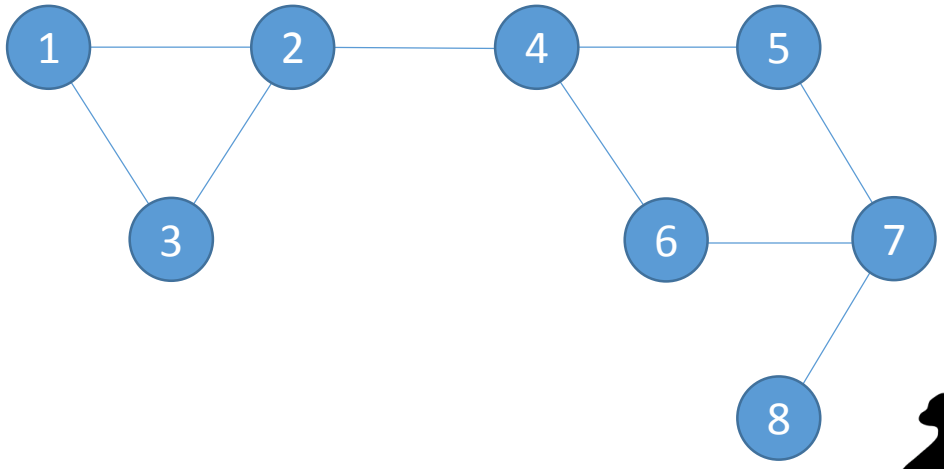
Graph

- Basic Concept
  - $Degree(v)$: the number of adjacent edges for a vertex v



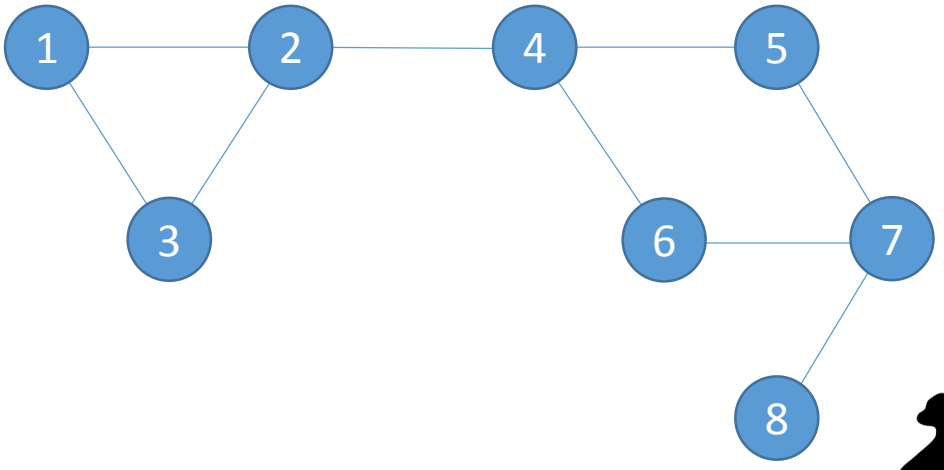
Graph

- Basic Concept
  - $Connectedness$: a graph is connected if there exist always a **path** which connects v and w for all v, w



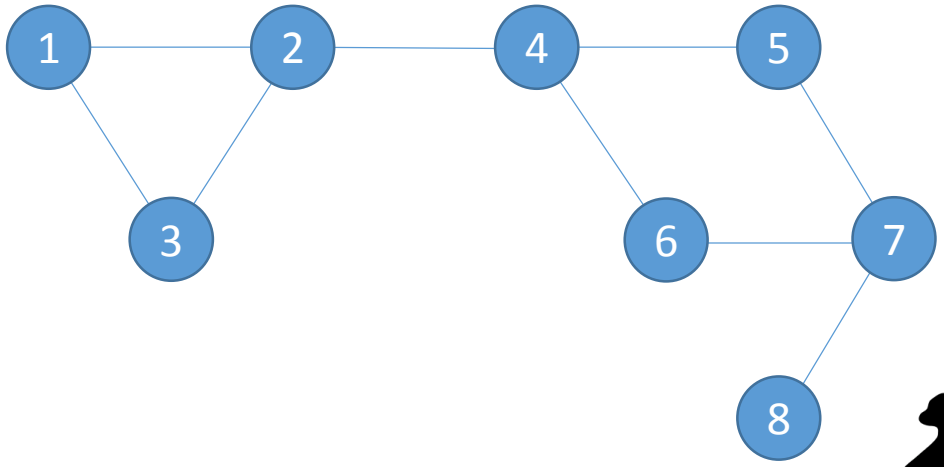
Graph Representation

- Representation
  - How can we represent a graph in the computer ?
  - Adjacency matrix & list



Graph Representation

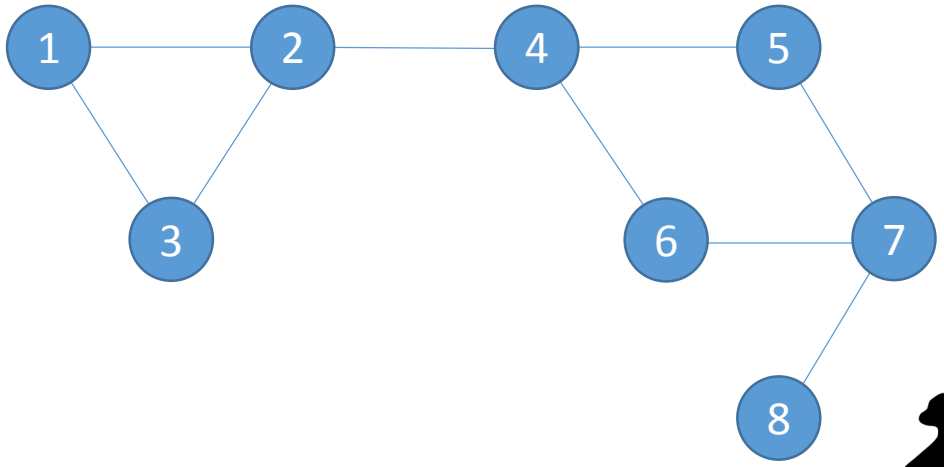
- Adjacency Matrix
  - Use matrix ! (you may already know this)
  - $G_{ij} = 1 \quad if\ there\ is\ a\ edge\ between\ vertex\ i\ and\ j$
    $0 \quad otherwise$



Graph Representation

- Adjacency Matrix
  - Use matrix ! (you may already know this)
  - $G_{ij} = 1 \quad if\ there\ is\ a\ edge\ between\ vertex\ i\ and\ j$
    $0 \quad otherwise$

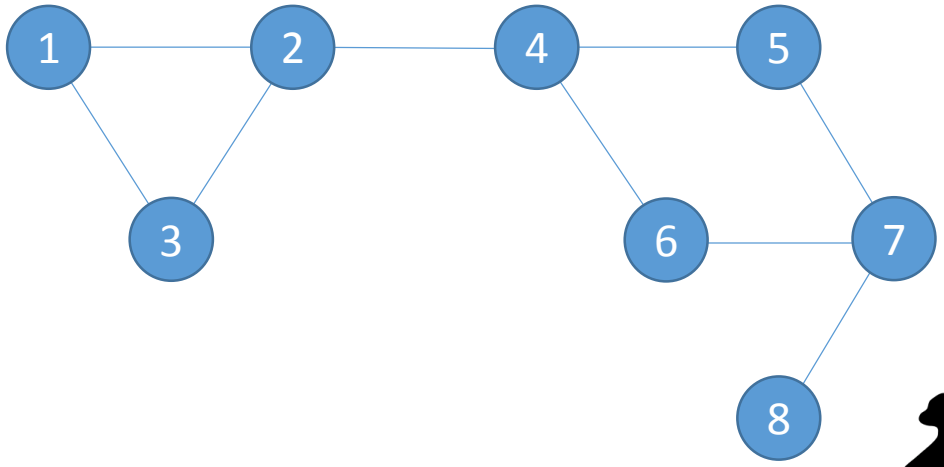
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	1	0	0	1	1	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	1	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0



Graph Representation

- Adjacency Matrix
  - Use matrix ! (you may already know this)
  - $G_{ij} = 1 \quad if\ there\ is\ a\ edge\ between\ vertex\ i\ and\ j$
    $0 \quad otherwise$

	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	1	0	0	1	1	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	1	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0

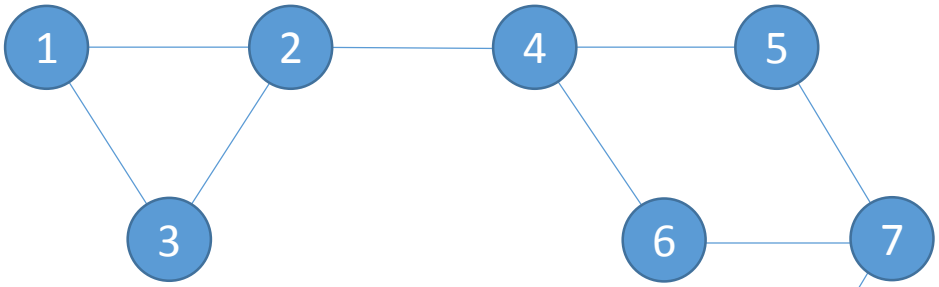


Is it good ?

Graph Representation

- Adjacency Matrix
  - We can determine whether there is a edge between i and j directly

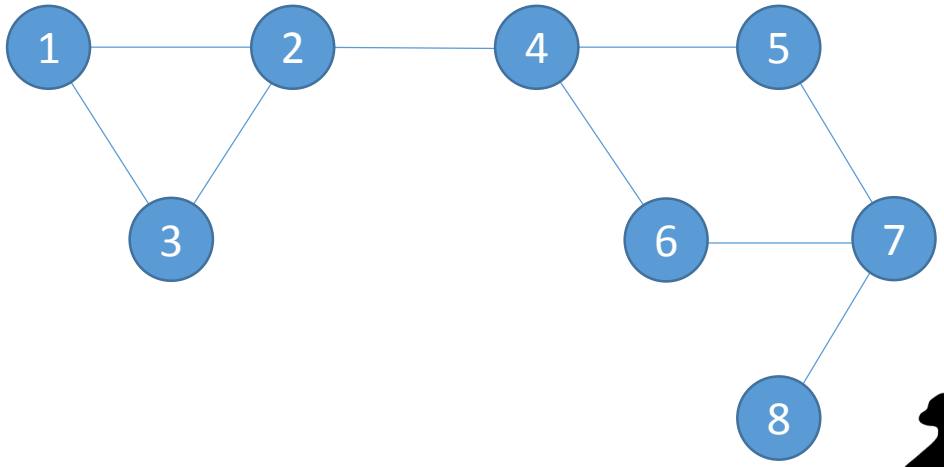
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	1	0	0	1	1	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	1	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0



Graph Representation

- Adjacency Matrix
  - We can determine whether there is a edge between i and j directly

	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	1	0	0	1	1	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	1	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0

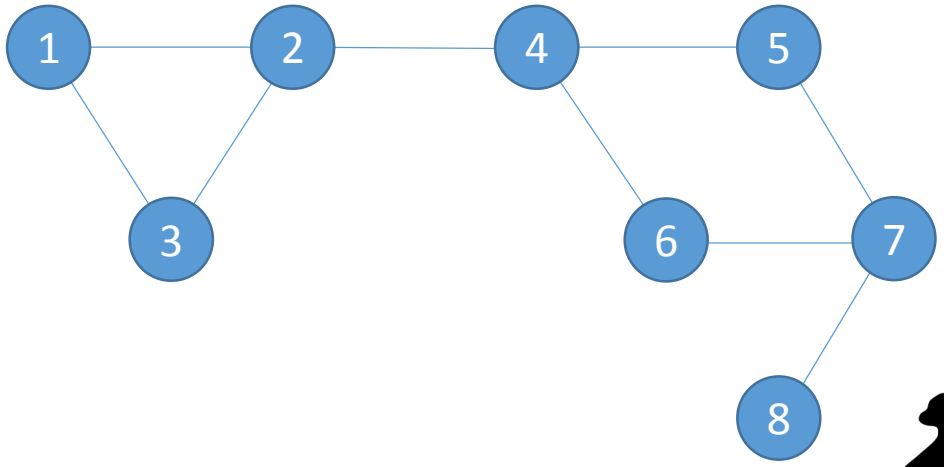


Is there a edge between 3 and 4 ?

Graph Representation

- Adjacency Matrix
  - We can determine whether there is a edge between i and j directly

	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	1	1	0	**0**	0	0	0	0
4	0	1	0	0	1	1	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	1	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0

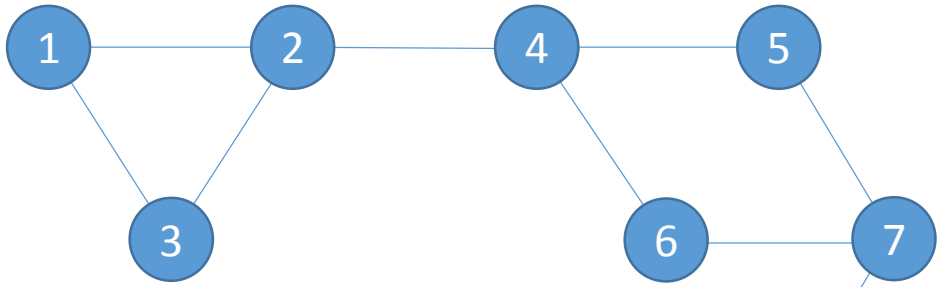


Is there a edge between 3 and 4 ? **No!**

Graph Representation

- Adjacency Matrix
  - We can determine whether there is a edge between i and j directly
  - We have to maintain whole matrix to save a graph

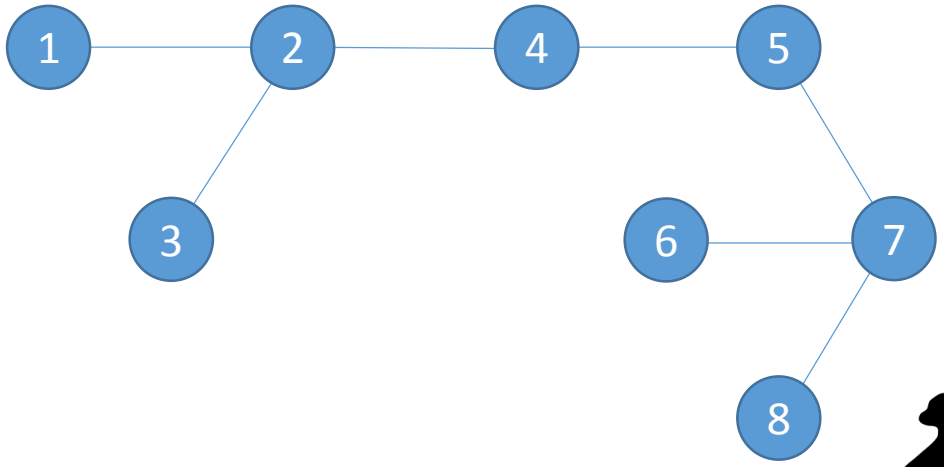
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	1	0	0	1	1	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	1	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0



Graph Representation

- Adjacency Matrix
  - We can determine whether there is a edge between i and j directly
  - We have to maintain whole matrix to save a graph

	1	2	3	4	5	6	7	8
1	0	1	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	0	1	0	0	0	0	0	0
4	0	1	0	0	1	0	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	0	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0

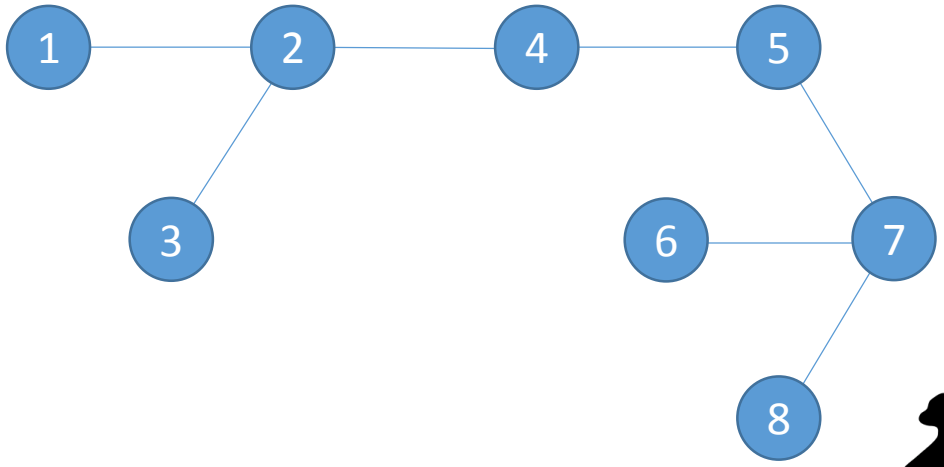


How about this ?

Graph Representation

- Adjacency Matrix
  - We can determine whether there is a edge between i and j directly
  - We have to maintain whole matrix to save a graph

	1	2	3	4	5	6	7	8
1	0	1	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	0	1	0	0	0	0	0	0
4	0	1	0	0	1	0	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	0	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0

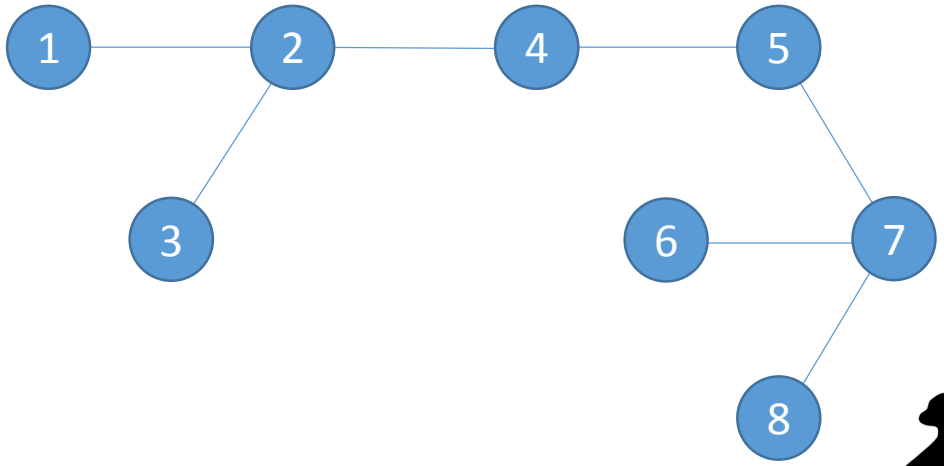


Compare the number of 1 and 0

Graph Representation

- Adjacency Matrix
  - We can determine whether there is a edge between i and j directly
  - We have to maintain whole matrix to save a graph

	1	2	3	4	5	6	7	8
1	0	1	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	0	1	0	0	0	0	0	0
4	0	1	0	0	1	0	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	0	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0

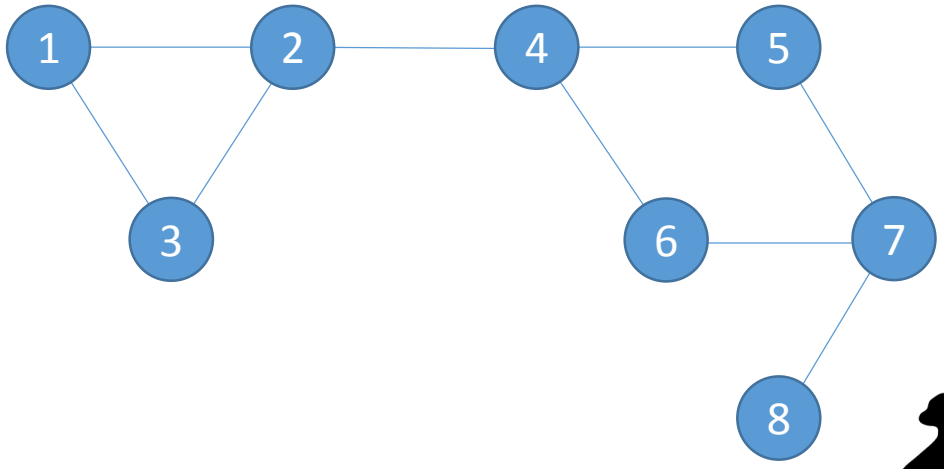


Compare the number of 1 and 0
Can we handle a graph with 100,000 vertex ?

Graph Representation

- Adjacency Matrix
  - We can determine whether there is a edge between i and j directly
  - We have to maintain whole matrix to save a graph
  - It takes $O(V)$ to find all the adjacent vertices for a vertex v

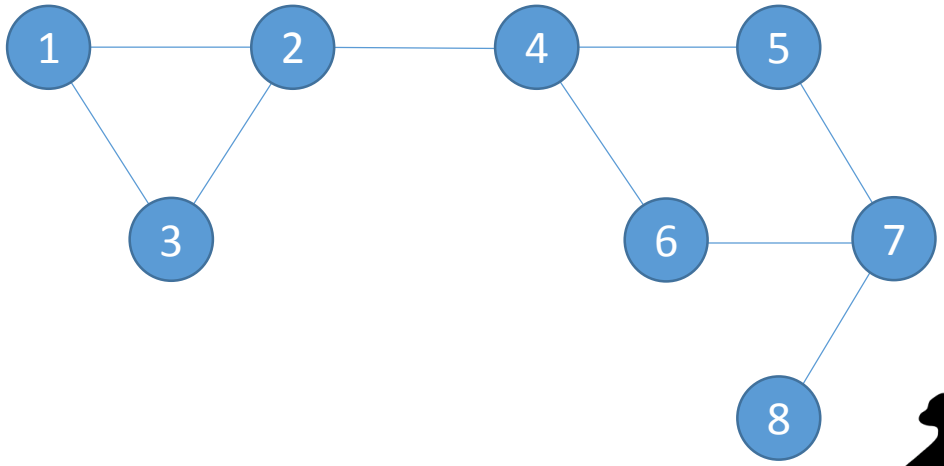
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	1	0	0	1	1	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	1	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0



Graph Representation

- Adjacency Matrix
  - We can determine whether there is a edge between i and j directly
  - We have to maintain whole matrix to save a graph
  - It takes $O(V)$ to find all the adjacent vertices for a vertex v

	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	1	0	0	1	1	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	1	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0

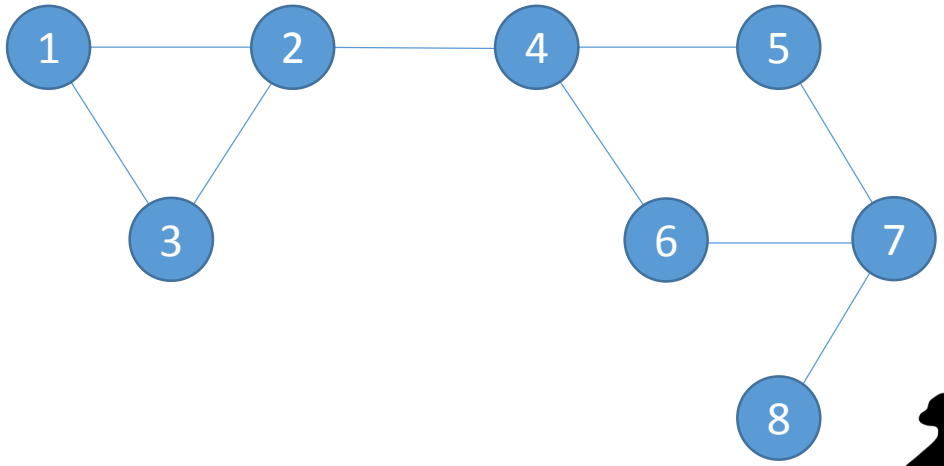


Give me all the adjacent vertices for vertex 3

Graph Representation

- Adjacency Matrix
  - We can determine whether there is a edge between i and j directly
  - We have to maintain whole matrix to save a graph
  - It takes $O(V)$ to find all the adjacent vertices for a vertex v

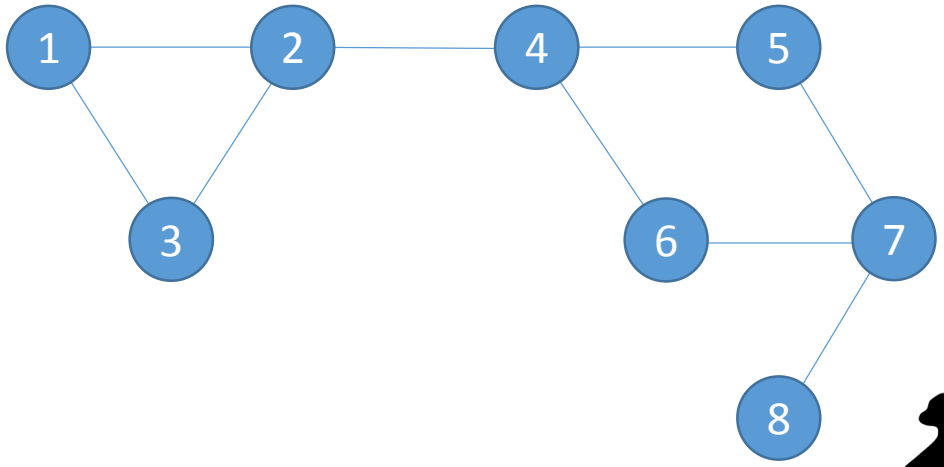
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	1	0	0	1	1	0	0
5	0	0	0	1	0	0	1	0
6	0	0	0	1	0	0	1	0
7	0	0	0	0	1	1	0	1
8	0	0	0	0	0	0	1	0



Give me all the adjacent vertices for vertex 3
→ It takes $O(V)$

Graph Representation

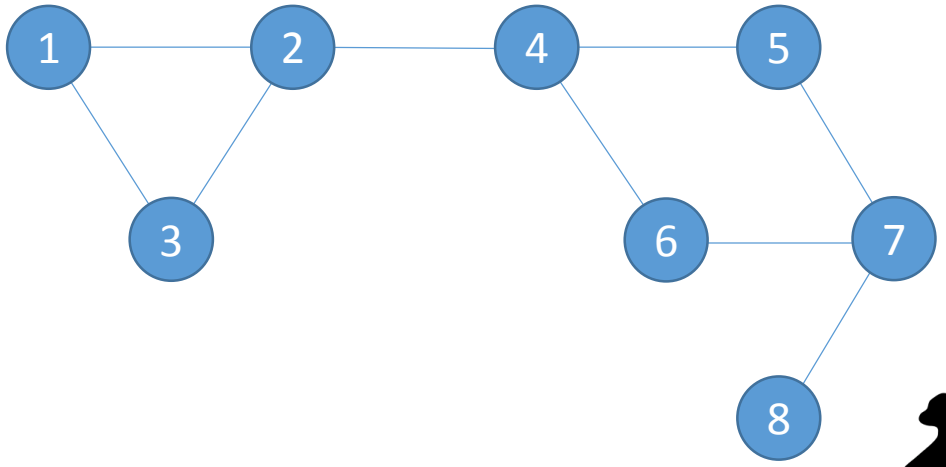
- Adjacency List
  - Handle just adjacent vertices for all vertices



Graph Representation

- Adjacency List
  - Handle just adjacent vertices for all vertices

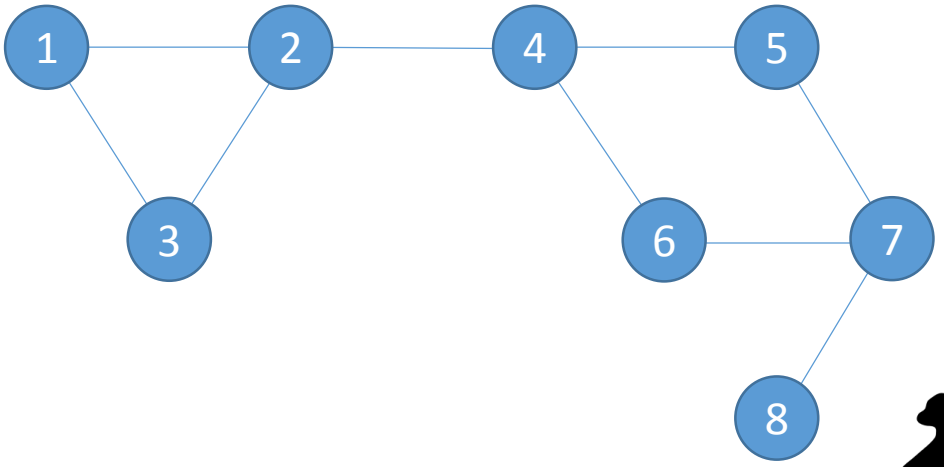
1	2	3	
2	1	3	4
3	1	2	
4	2	5	6
5	4	7	
6	4	7	
7	5	6	8



Graph Representation

- Adjacency List
  - Handle just adjacent vertices for all vertices
  - Optimal space complexity (i.e. no redundant space)

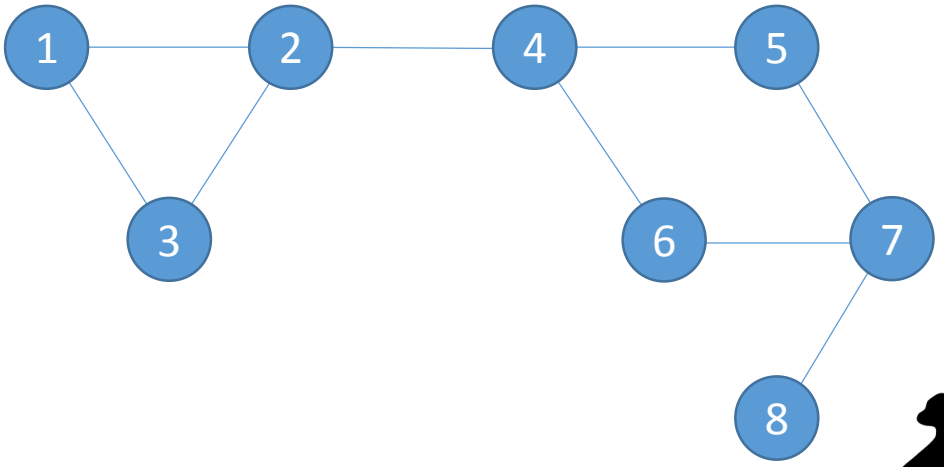
1	2	3	
2	1	3	4
3	1	2	
4	2	5	6
5	4	7	
6	4	7	
7	5	6	8



Graph Representation

- Adjacency List
  - Handle just adjacent vertices for all vertices
  - Optimal space complexity (i.e. no redundant space)
  - It takes O($degree(v)$) to find all the adjacent vertices for vertex v

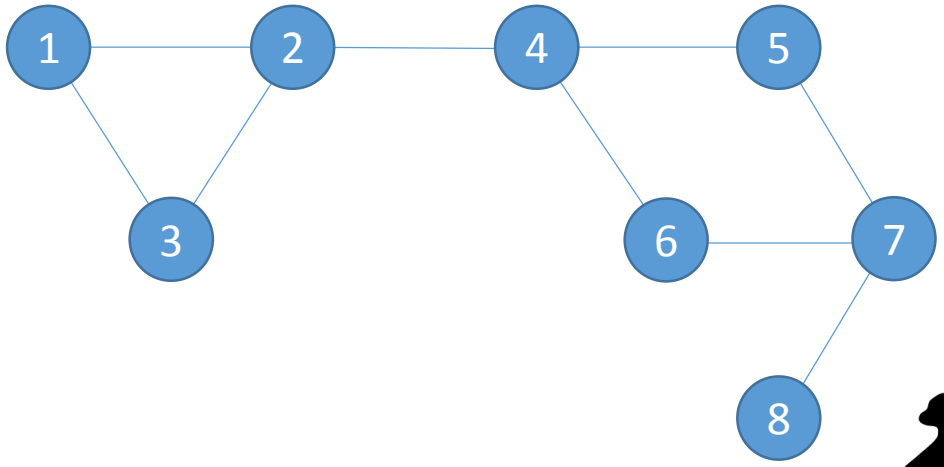
1	2	3	
2	1	3	4
3	1	2	
4	2	5	6
5	4	7	
6	4	7	
7	5	6	8



Graph Representation

- Adjacency List
  - Handle just adjacent vertices for all vertices
  - Optimal space complexity (i.e. no redundant space)
  - It takes O($degree(v)$) to find all the adjacent vertices for vertex v
  - It take also O($degree(v)$) to determine the existence of edge

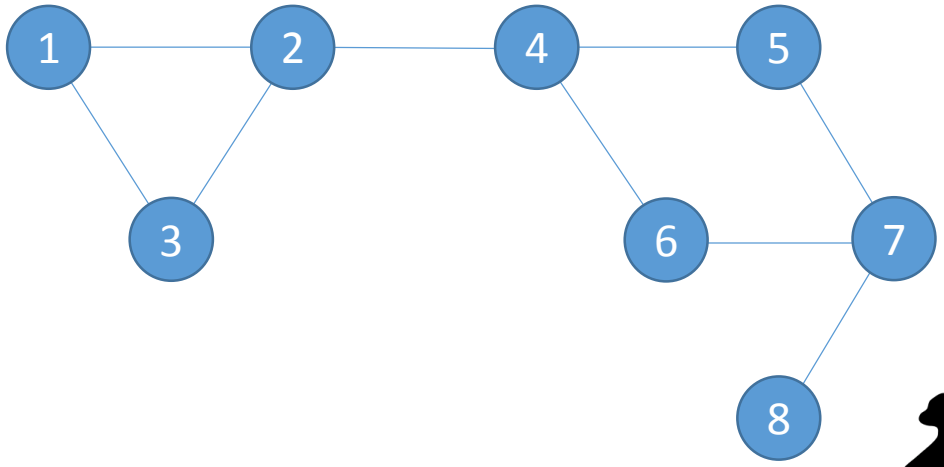
1	2	3	
2	1	3	4
3	1	2	
4	2	5	6
5	4	7	
6	4	7	
7	5	6	8



Graph Representation

- Adjacency List
  - Handle just adjacent vertices for all vertices
  - Optimal space complexity (i.e. no redundant space)
  - It takes O($degree(v)$) to find all the adjacent vertices for vertex v
  - It take also O($degree(v)$) to determine the existence of edge

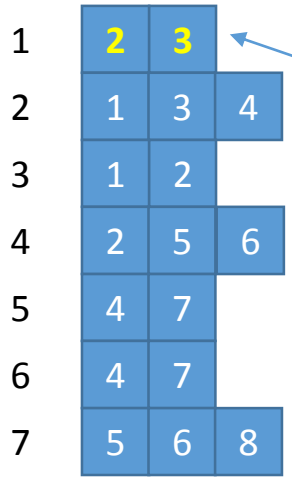
1	2	3	
2	1	3	4
3	1	2	
4	2	5	6
5	4	7	
6	4	7	
7	5	6	8



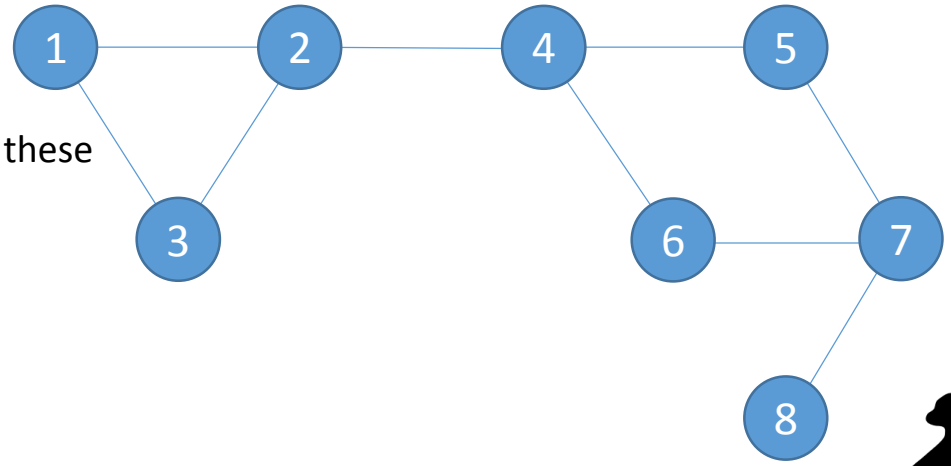
Is there edge between vertex 1 and 3 ?

Graph Representation

- Adjacency List
  - Handle just adjacent vertices for all vertices
  - Optimal space complexity (i.e. no redundant space)
  - It takes O($degree(v)$) to find all the adjacent vertices for vertex v
  - It take also O($degree(v)$) to determine the existence of edge



We have to consider all of these



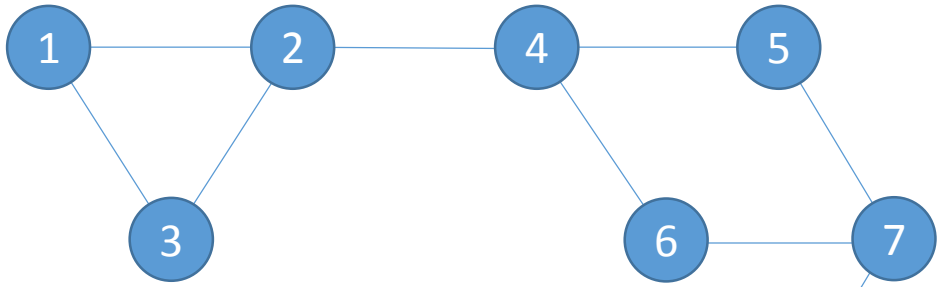
Is there edge between vertex 1 and 3 ?

Graph Representation

- Adjacency List

  By the way, is it possible to manage this kind of structure ?

1	2	3	
2	1	3	4
3	1	2	
4	2	5	6
5	4	7	
6	4	7	
7	5	6	8



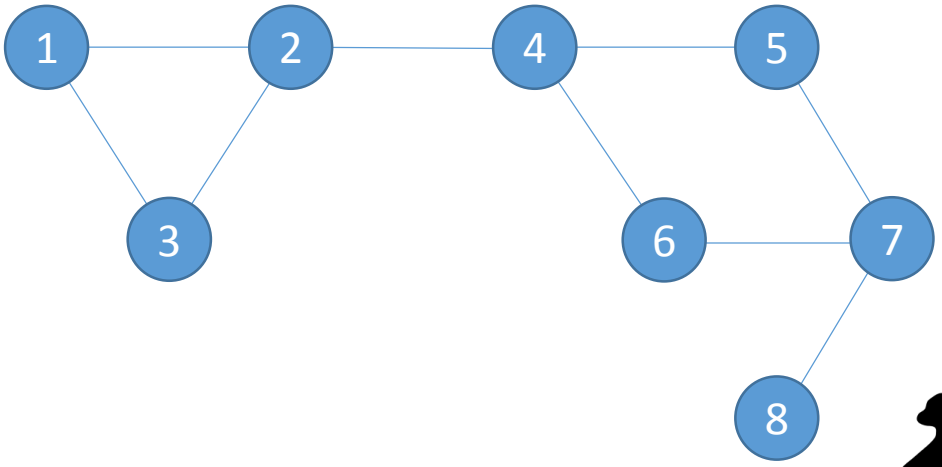
Graph Representation

- Adjacency List

  By the way, is it possible to manage this kind of structure ?

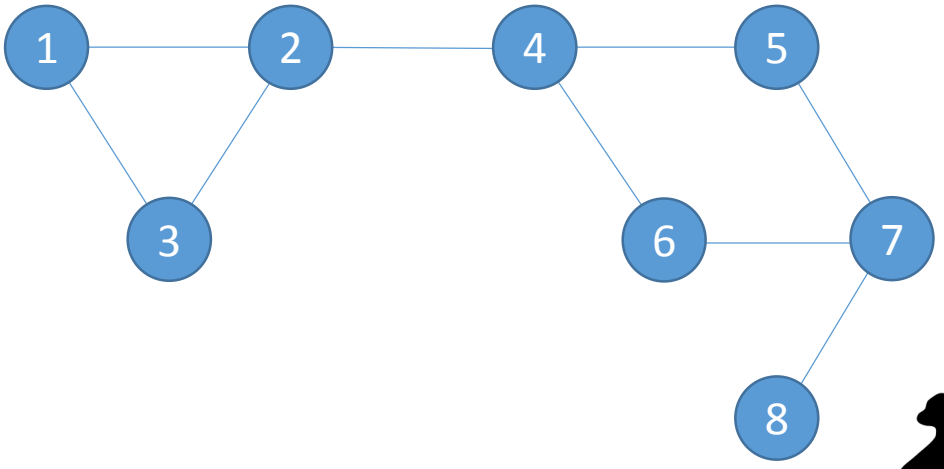
  Yes we can! Use STL vector (I'll give some simple lecture for you today)

1	2	3	
2	1	3	4
3	1	2	
4	2	5	6
5	4	7	
6	4	7	
7	5	6	8



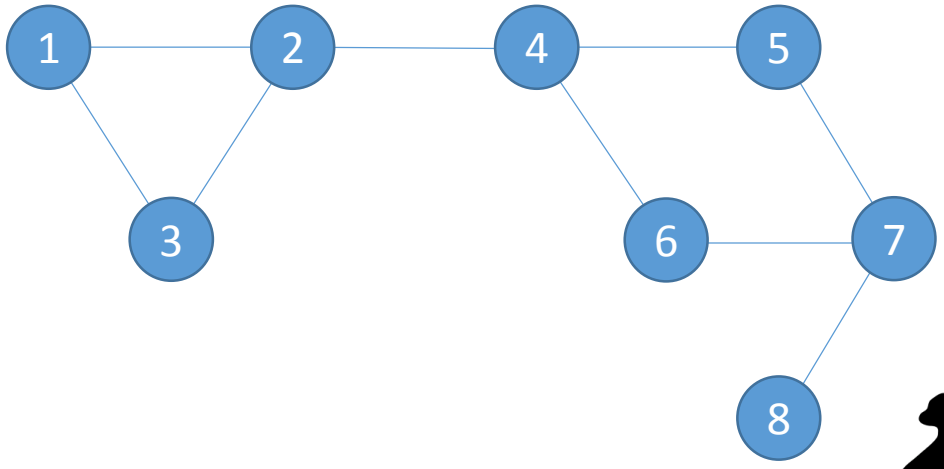
Graph Representation

- Summary
  - Adjacency matrix VS Adjacency list
  - Existence of edge ?
  - Find all the neighboring vertices ?
  - Space ?



Graph Representation

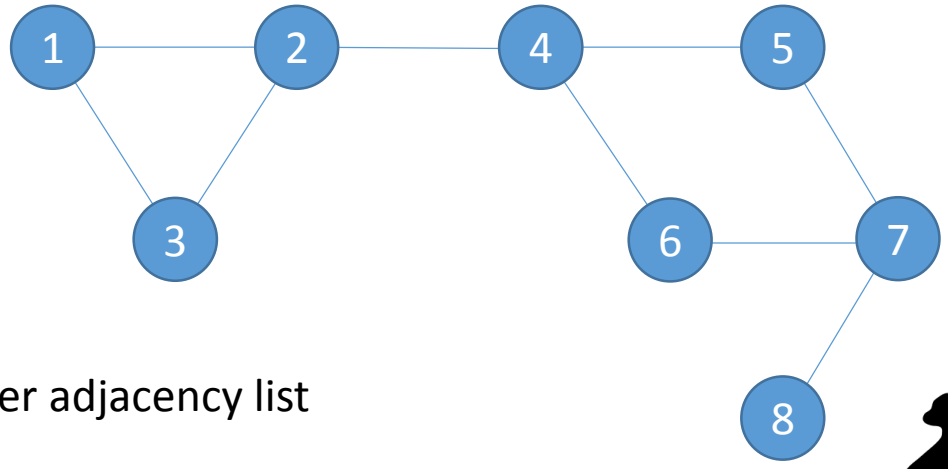
- Summary
  - Adjacency matrix VS Adjacency list
  - Existence of edge ?
  - Find all the neighboring vertices ?
  - Space ?



Which one is better ?

Graph Representation

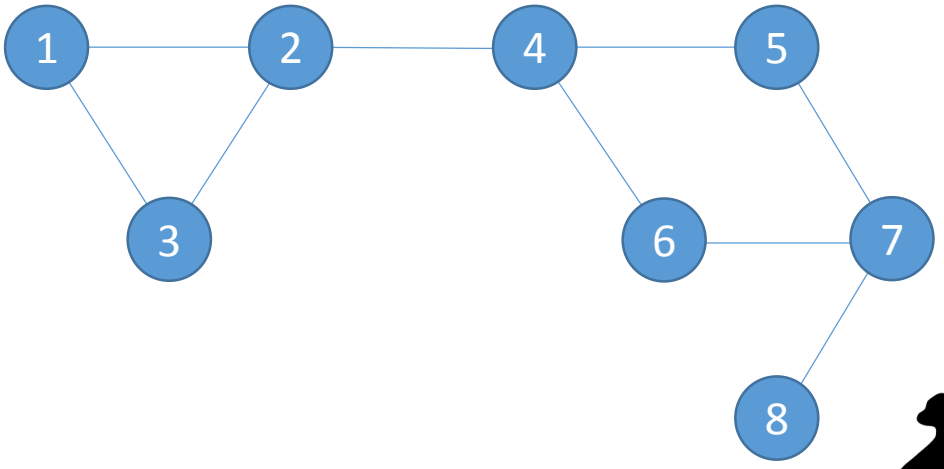
- Summary
  - Adjacency matrix VS Adjacency list
  - Existence of edge ?
  - Find all the neighboring vertices ?
  - Space ?



Which one is better ?
It depends, but usually we prefer adjacency list

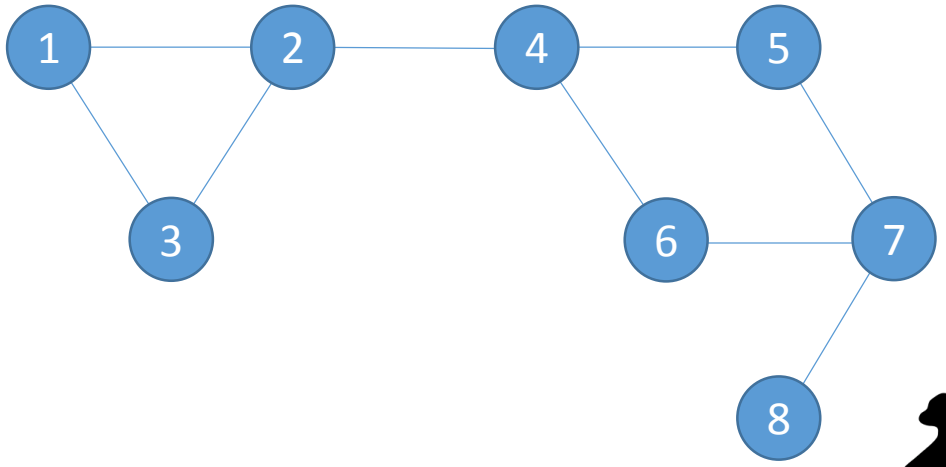
Graph Representation

- Summary
  - Adjacency matrix VS Adjacency list
  - Existence of edge ?
  - Find all the neighboring vertices ?
  - Space ?



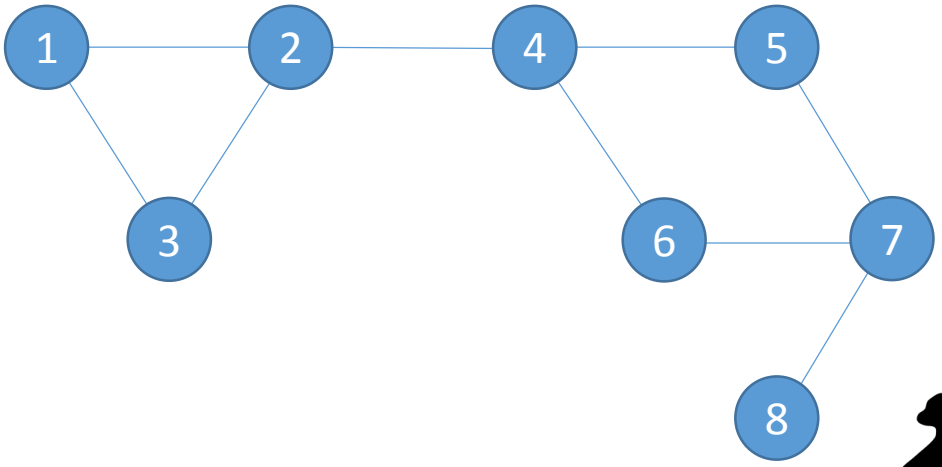
Graph Traversal

- Traversal paradigm
  - Basically, graph is also data structure
  - In other words, we want to contain some information into a graph
  - Therefore, we have to be able to traverse whole graph to find datas
  - Depth First Search VS Breadth First Search



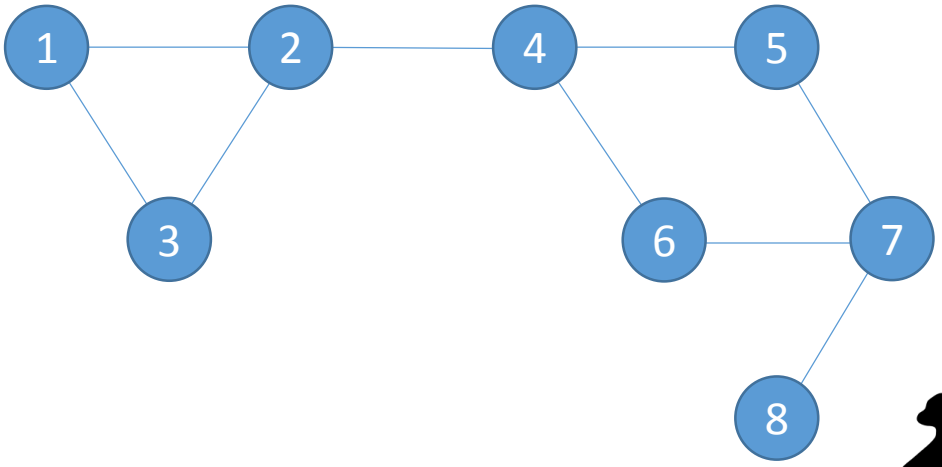
Graph Traversal

- Depth First Search
  - Use stack to traverse
  - Invarient : The current vertex is pointed by the top of the stack
    (Discuss it later. Forget it)



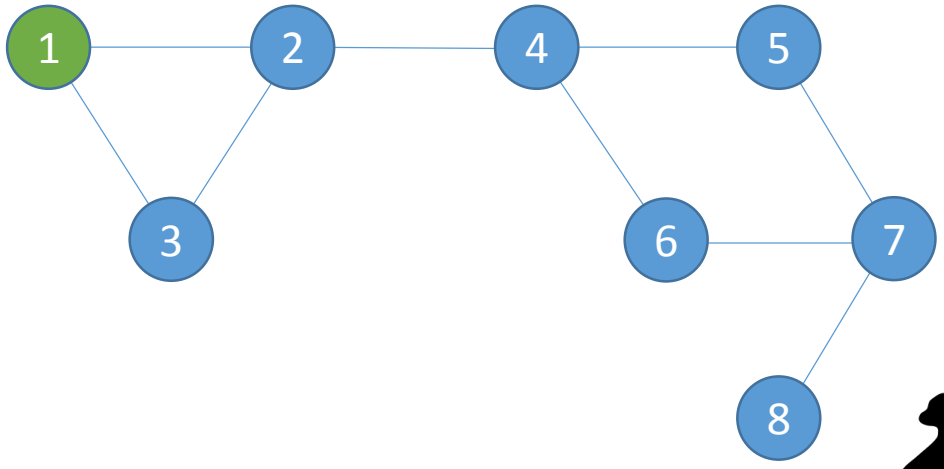
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



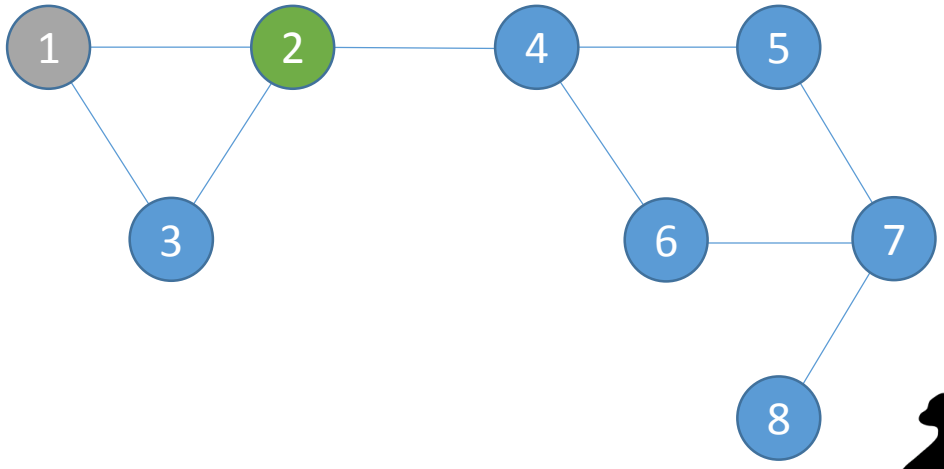
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



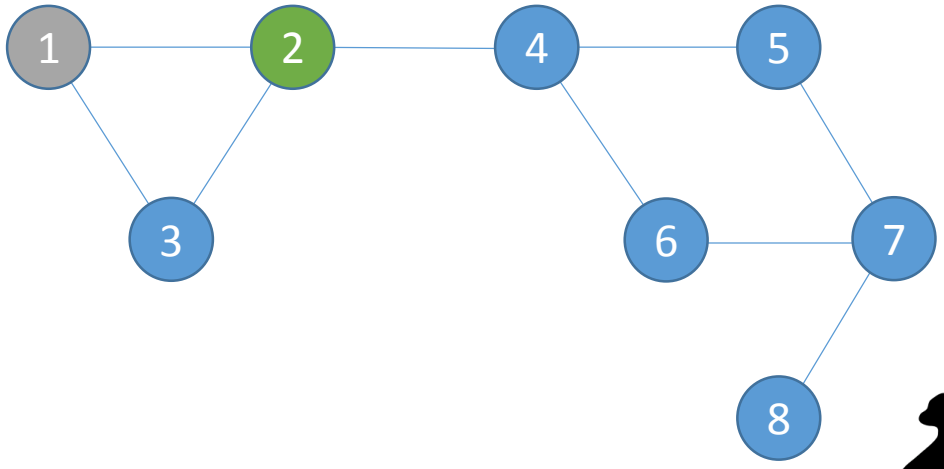
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



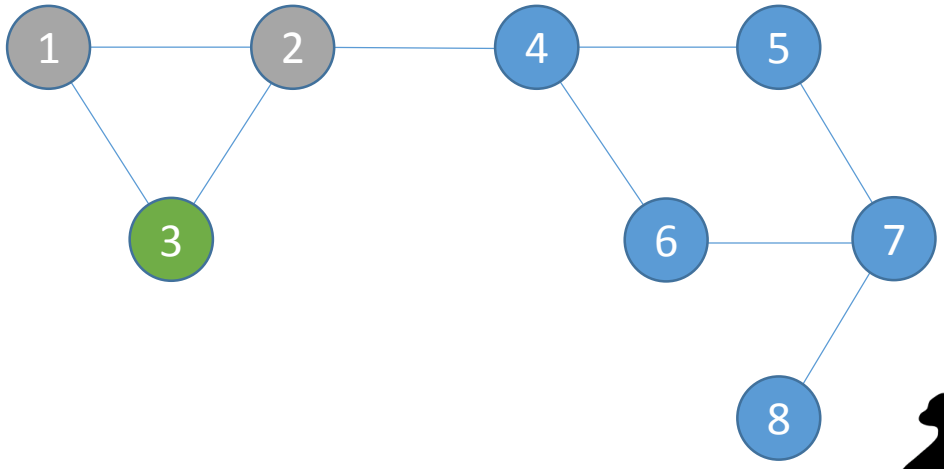
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



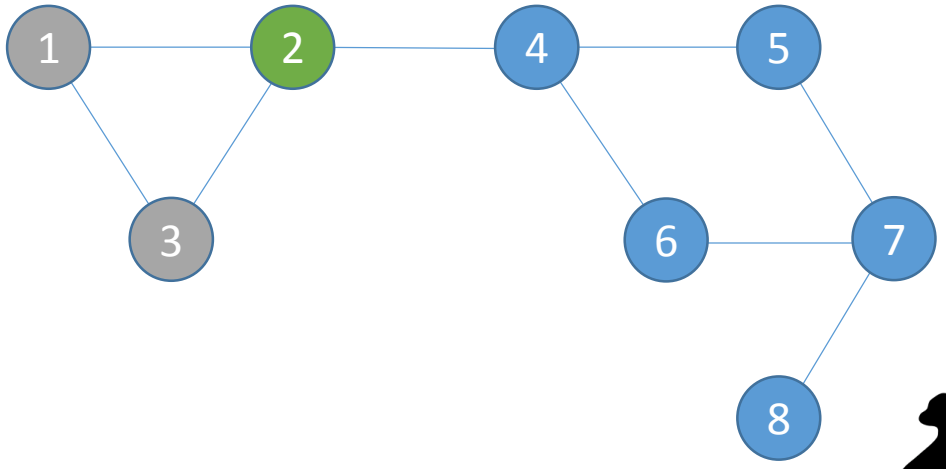
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



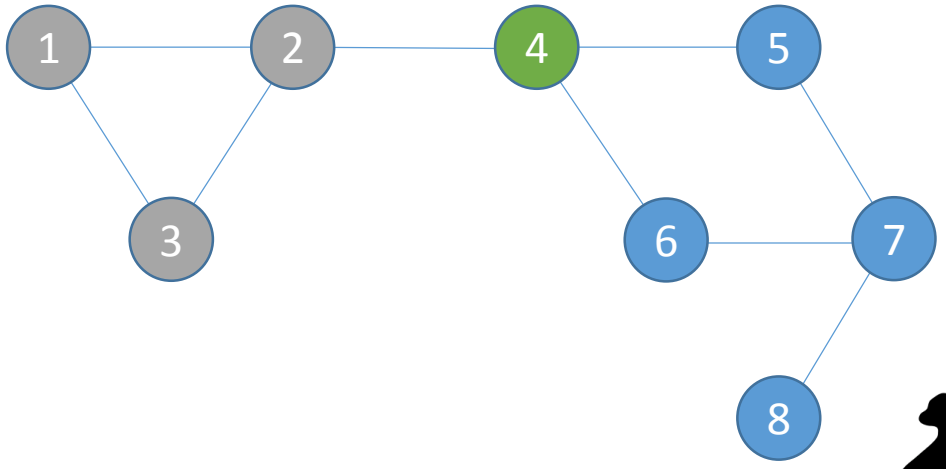
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



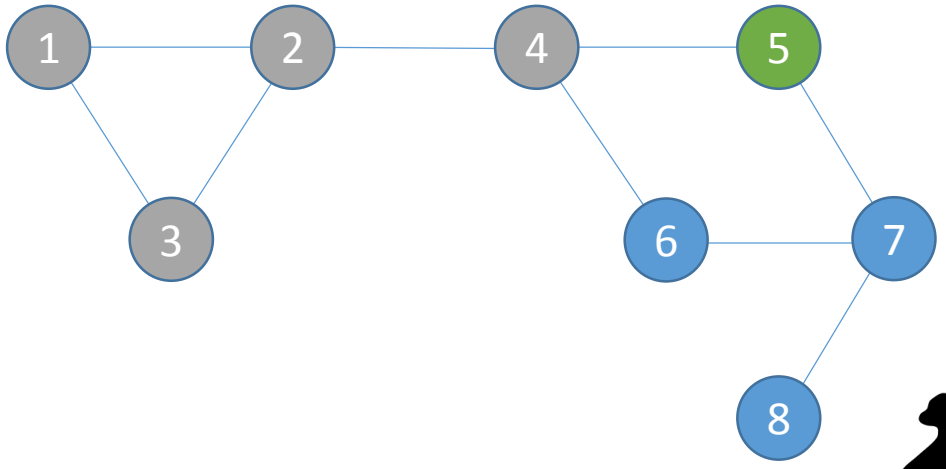
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



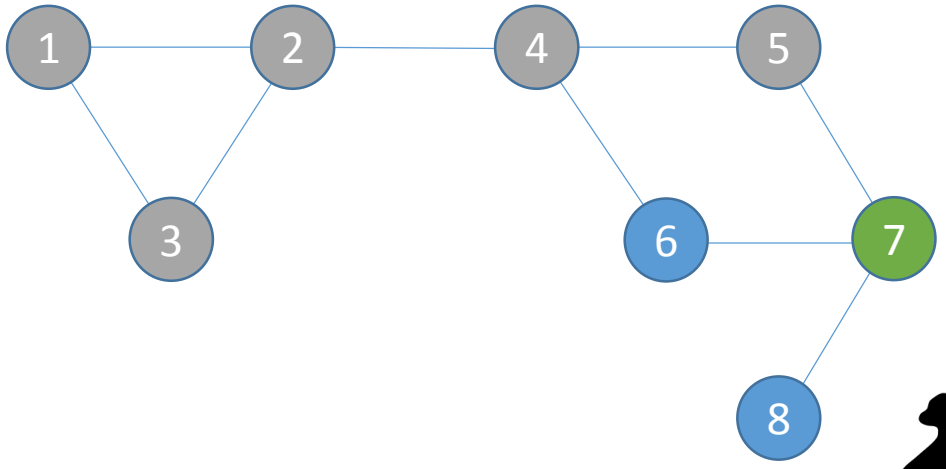
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



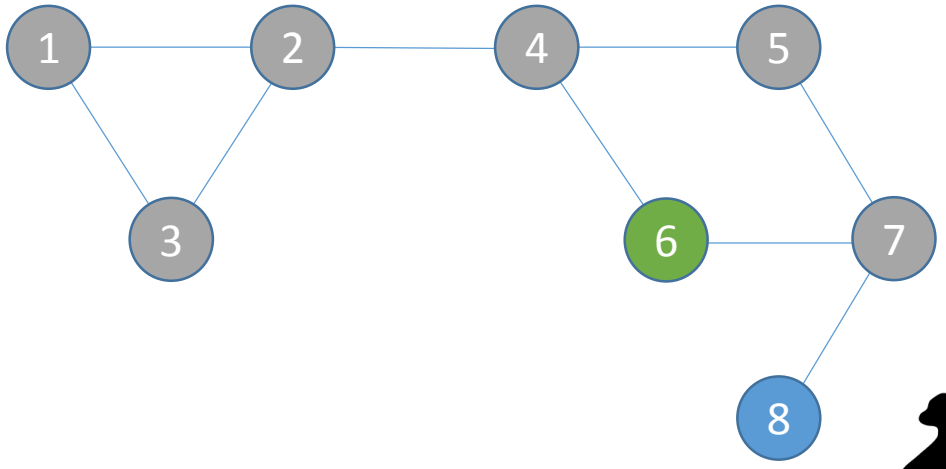
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



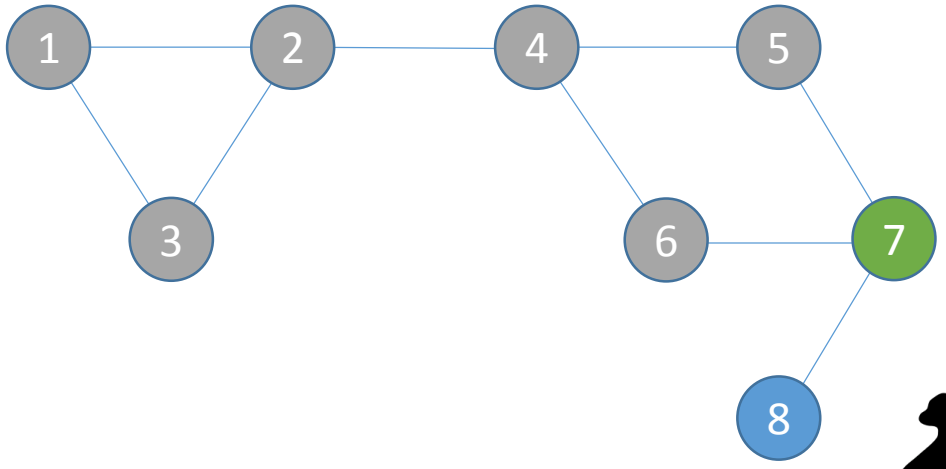
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



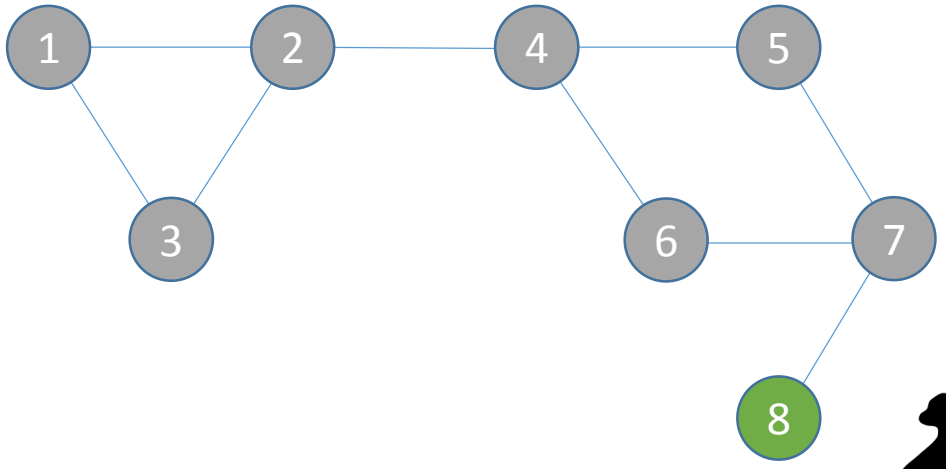
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



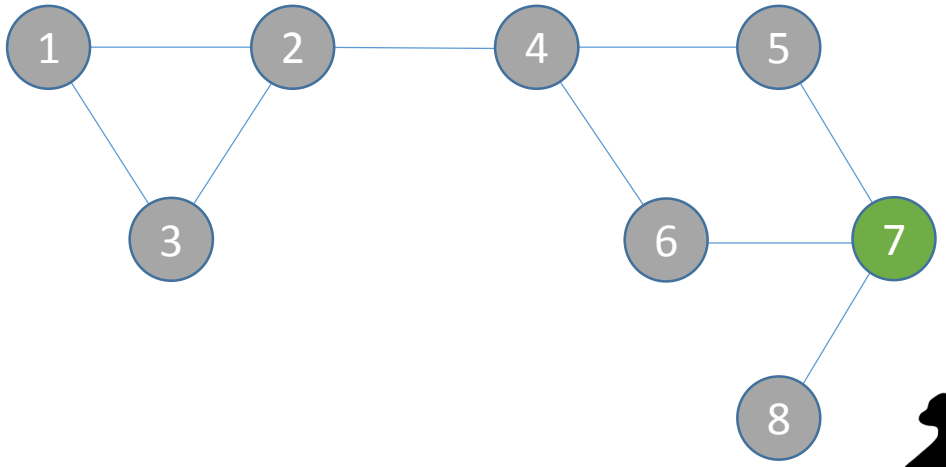
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



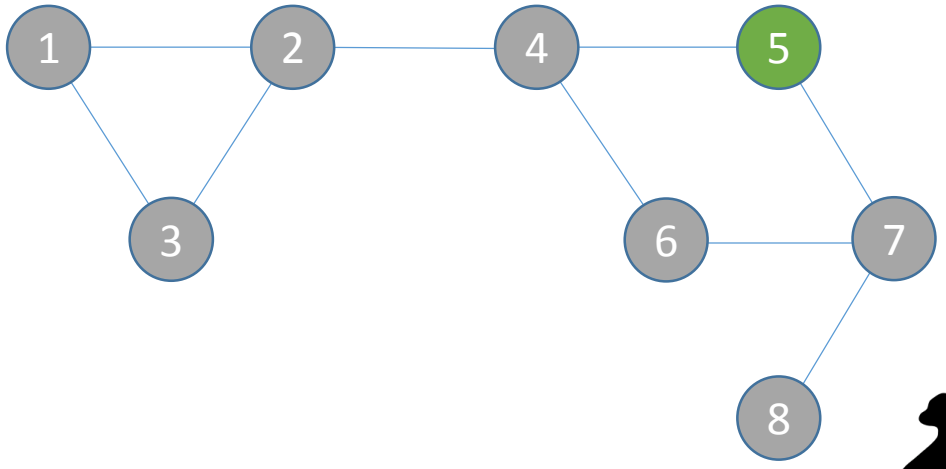
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



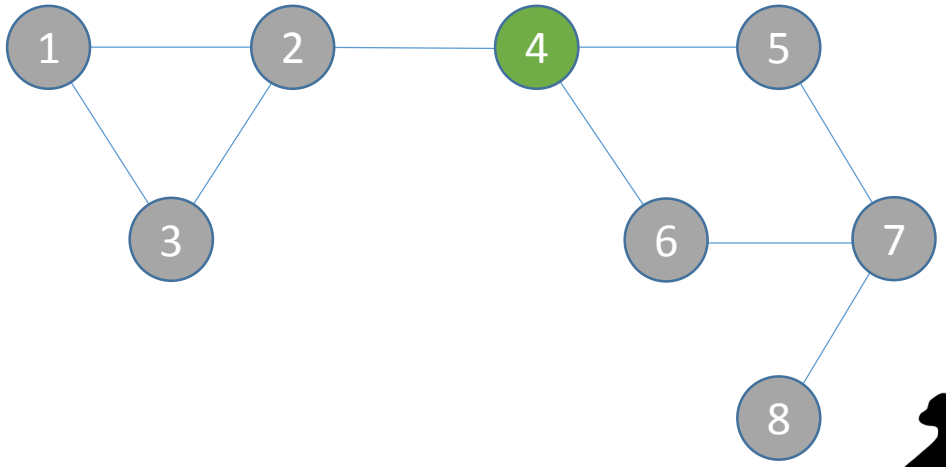
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



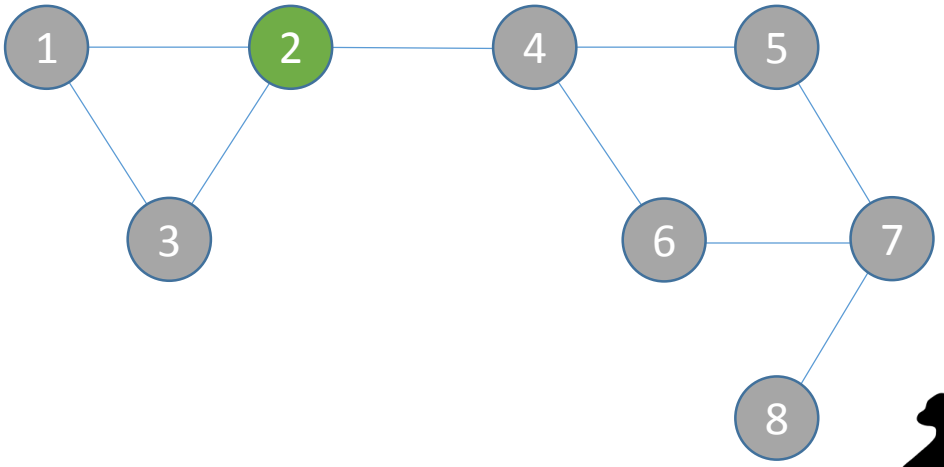
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



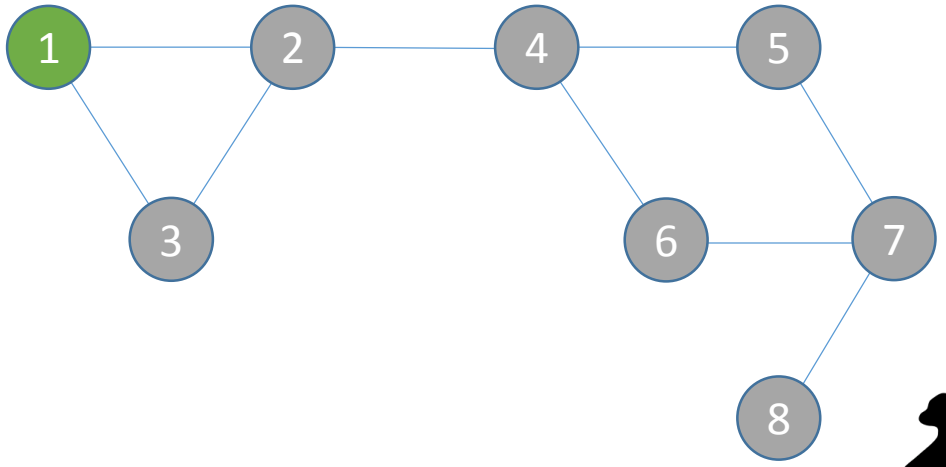
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



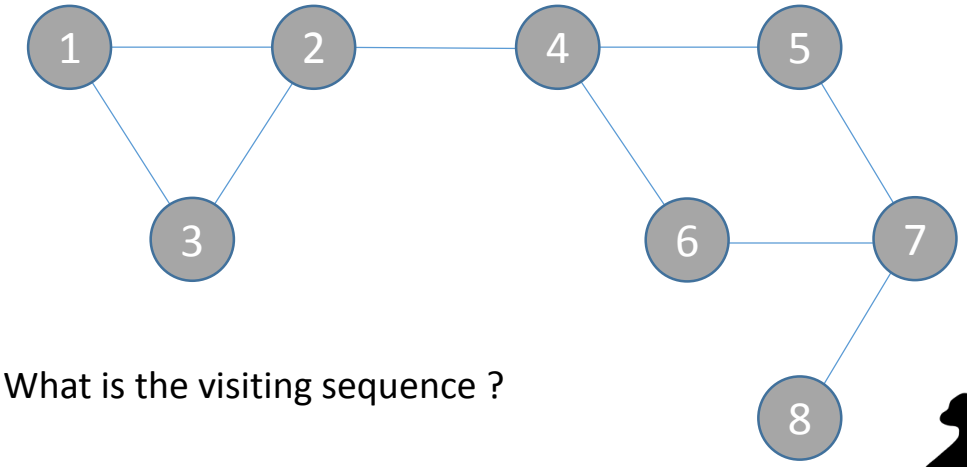
Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



Graph Traversal

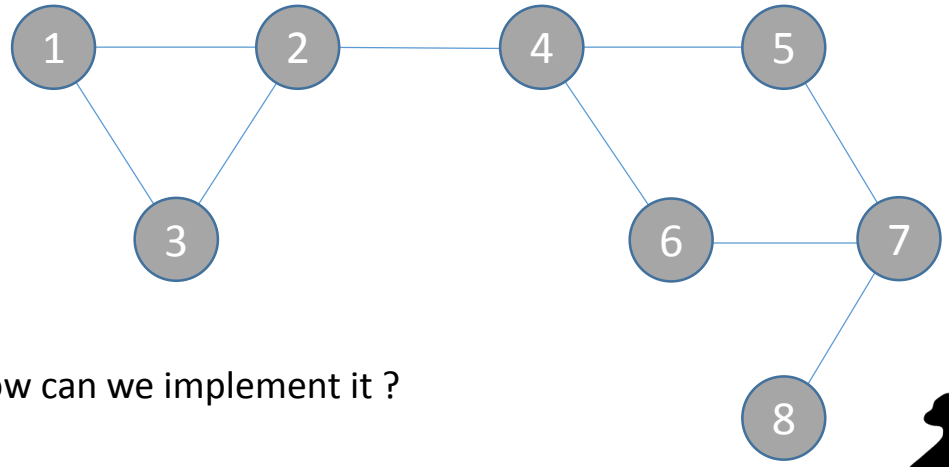
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



Done ! What is the visiting sequence ?

Graph Traversal

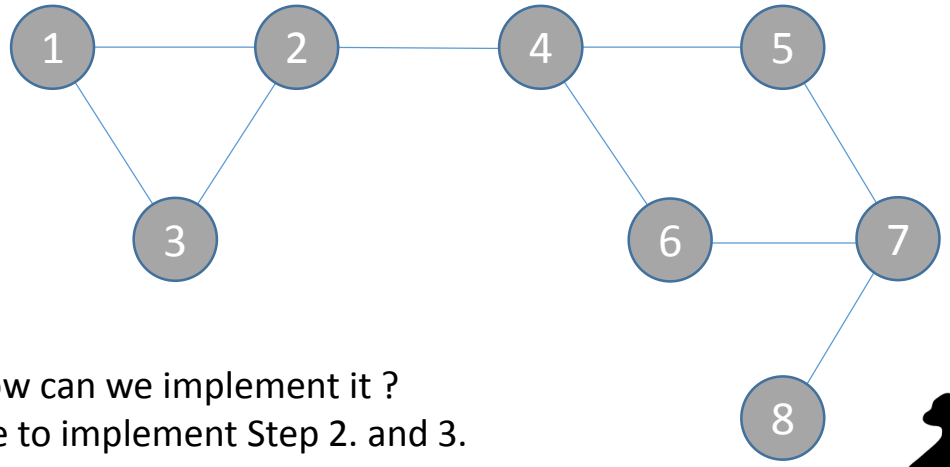
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



Well, how can we implement it ?

Graph Traversal

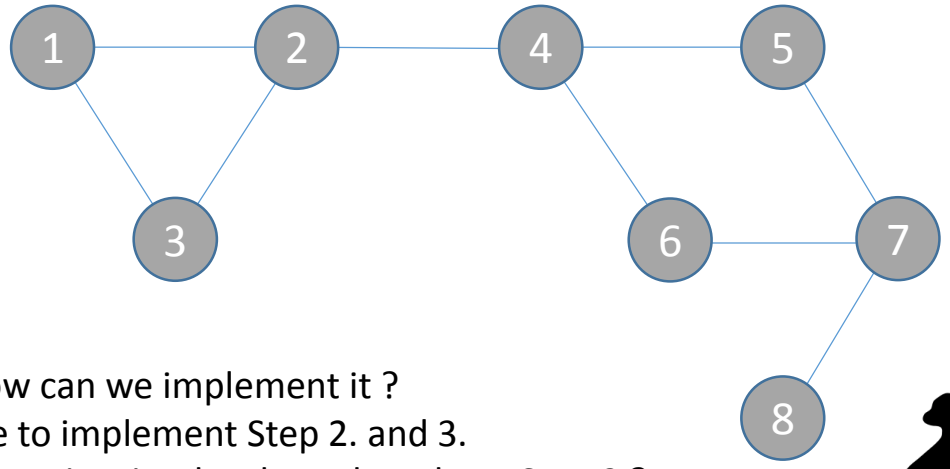
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



Well, how can we implement it ?
We have to implement Step 2. and 3.

Graph Traversal

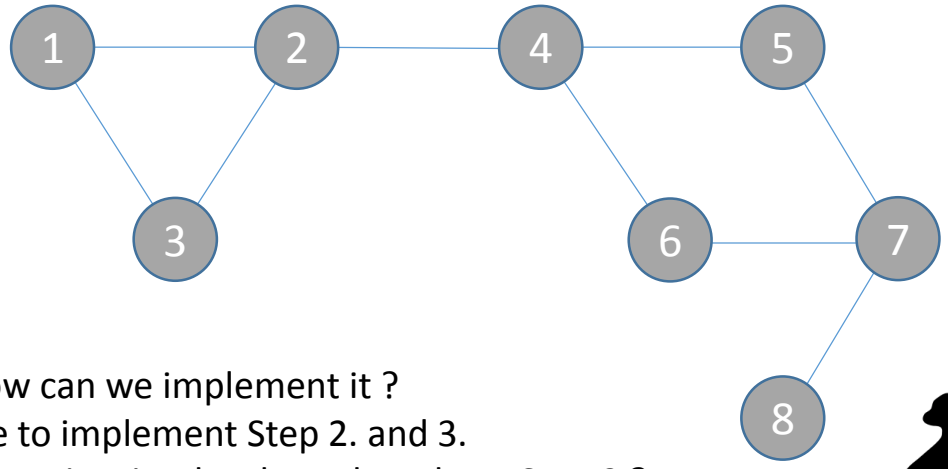
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



Well, how can we implement it ?
We have to implement Step 2. and 3.
Step 2. is quite simple… but what about Step 3 ?
We should move backward.

Graph Traversal

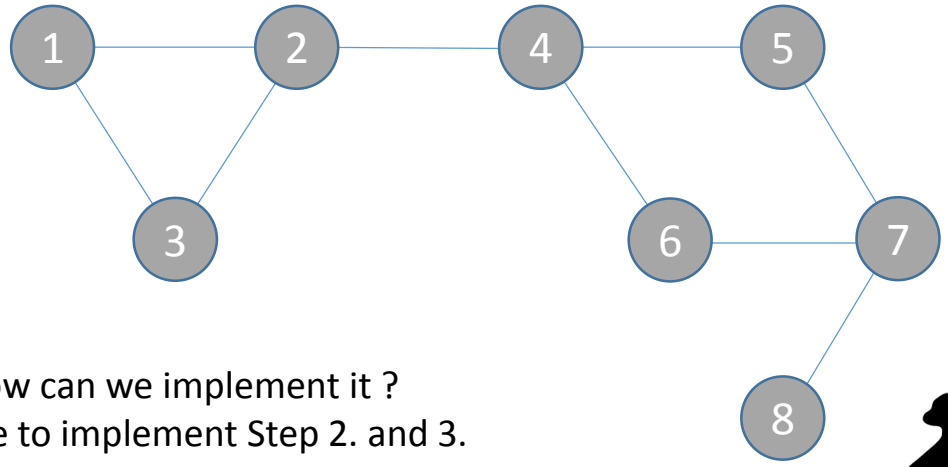
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.



Well, how can we implement it ?
We have to implement Step 2. and 3.
Step 2. is quite simple… but what about Step 3 ?
We should move backward.
How can we remember the past vertices ?

Graph Traversal

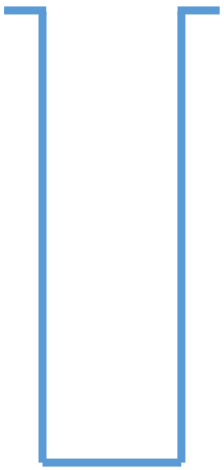
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.

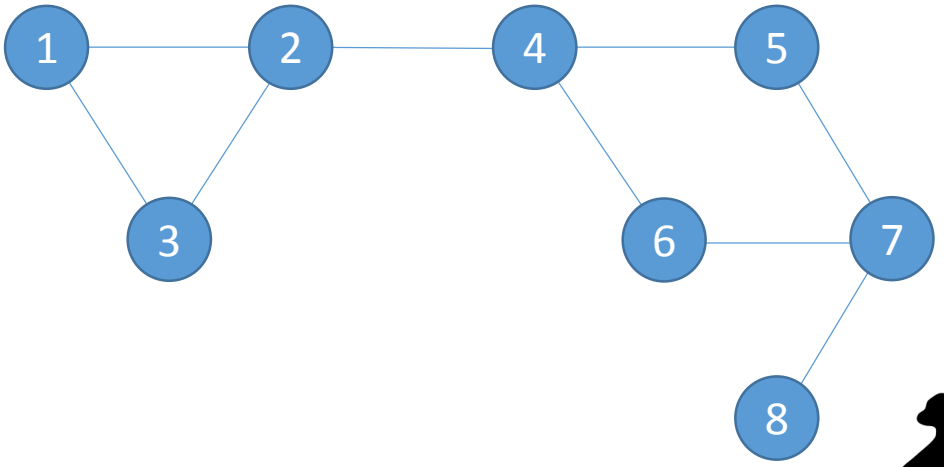


Well, how can we implement it ?
We have to implement Step 2. and 3.
Step 2. is quite simple… but what about Step 3 ?
We should move backward.
How can we remember the past vertices ? **STACK !**

Graph Traversal

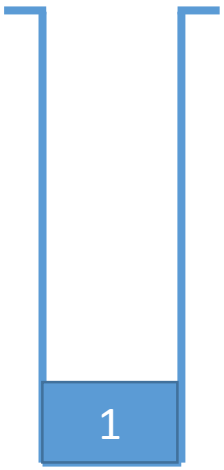
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.

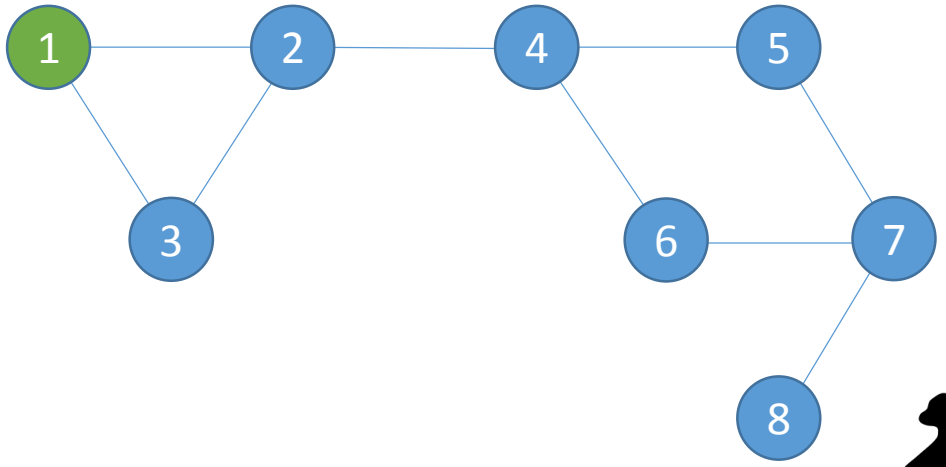




Graph Traversal

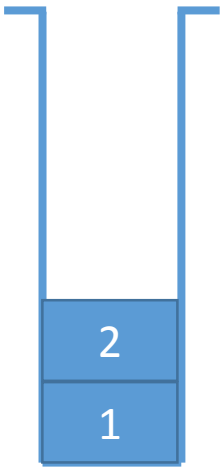
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.

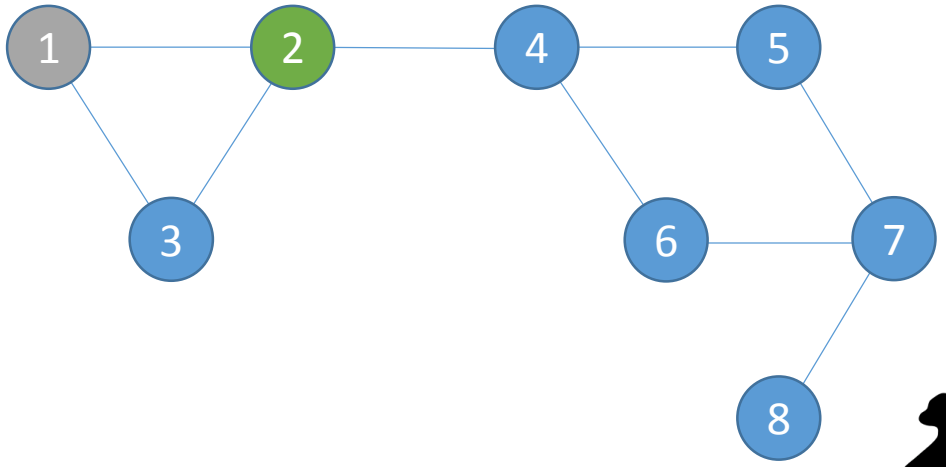




Graph Traversal

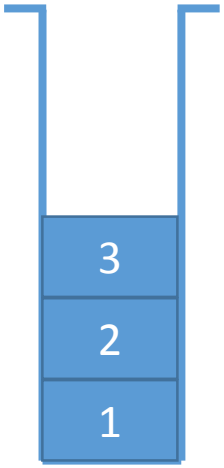
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.

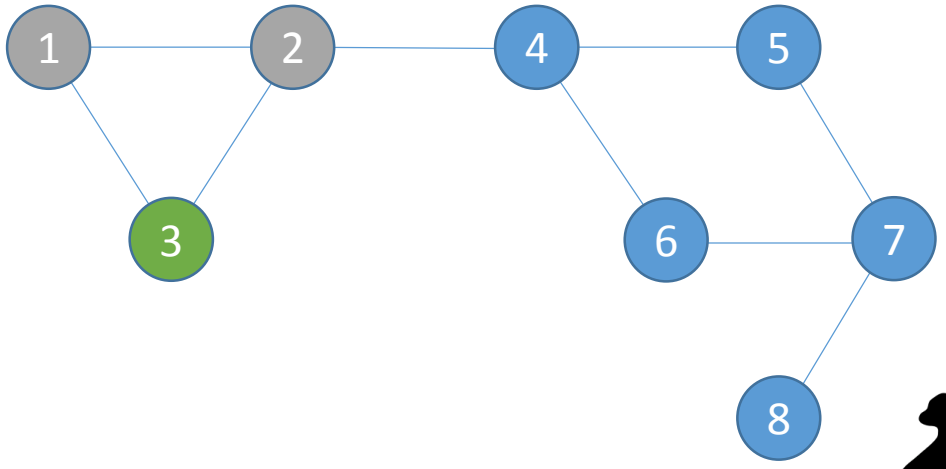




Graph Traversal

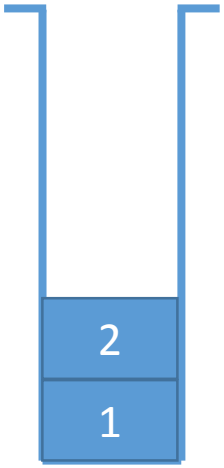
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.

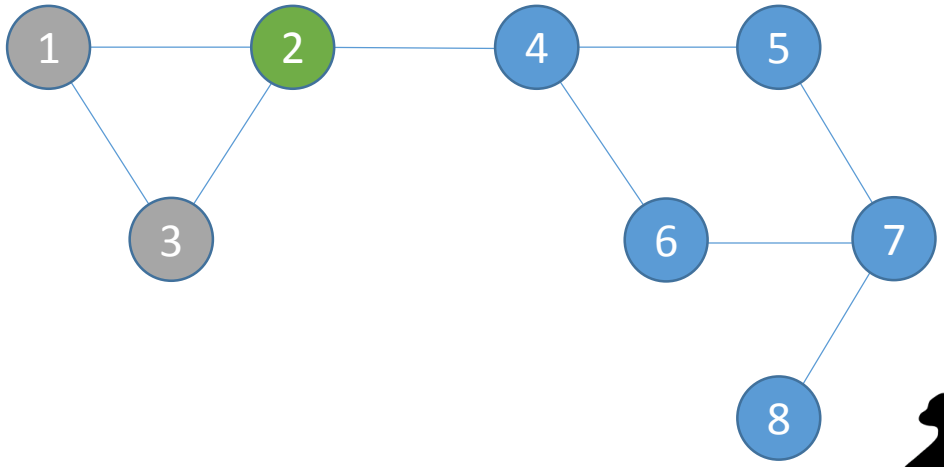




Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.

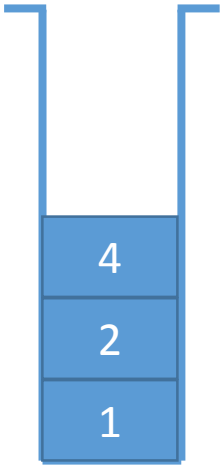


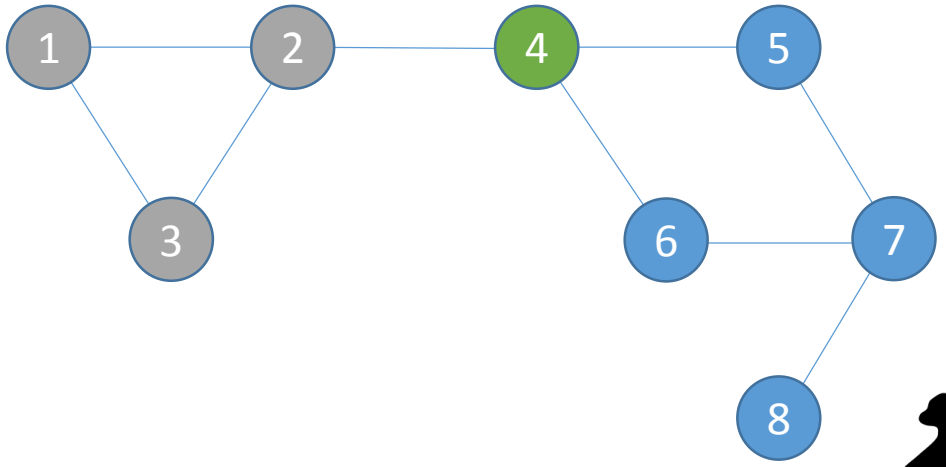


Simply pop ! Then we get a vertex just before !

Graph Traversal

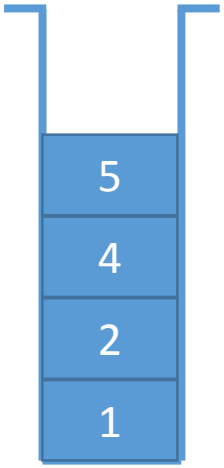
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.

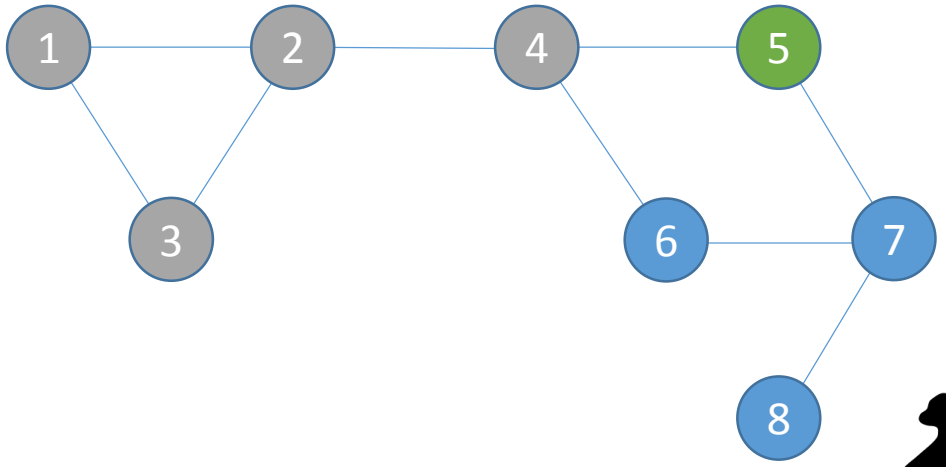




Graph Traversal

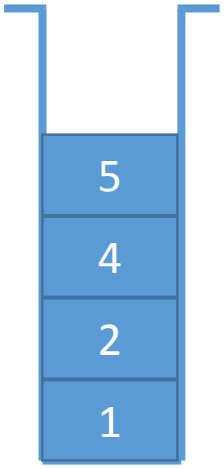
- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.

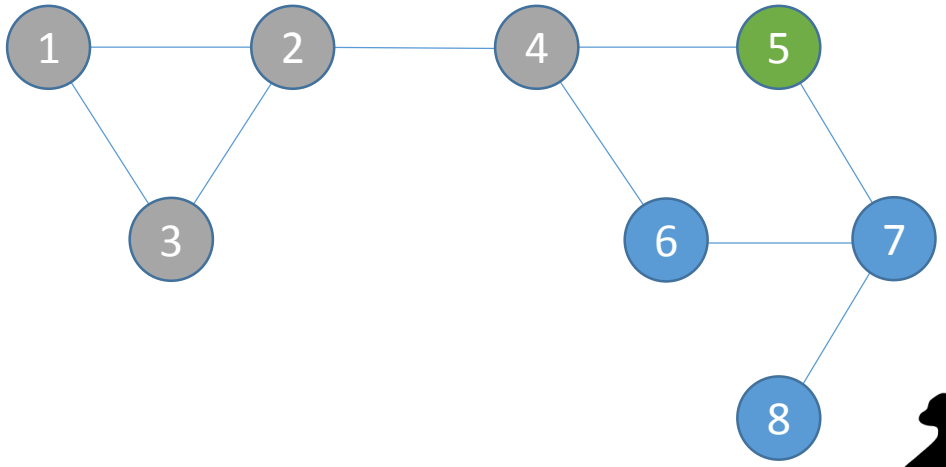




Graph Traversal

- Depth First Search
  - Procedure
    1. Select a start vertex
    2. Move to adjacent vertex we don't visit yet
    3. If there is no such a vertex, move backward
    4. Go to step 2.

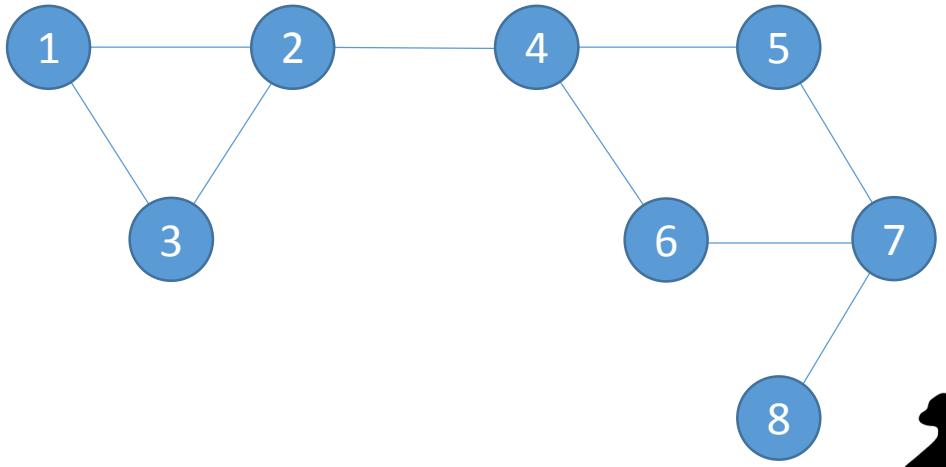




Please do it yourself ☺

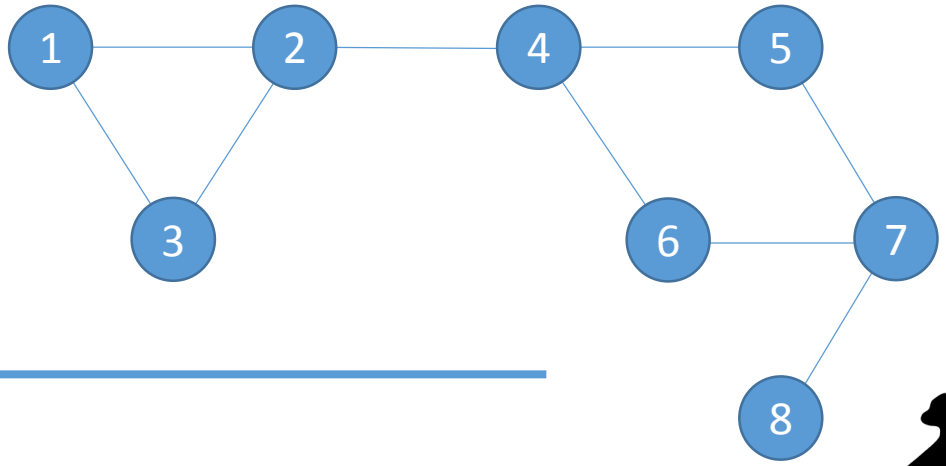
Graph Traversal

- Breadth First Search
  - Use queue to traverse
  - Invarient : The current vertex is pointed by front pointer



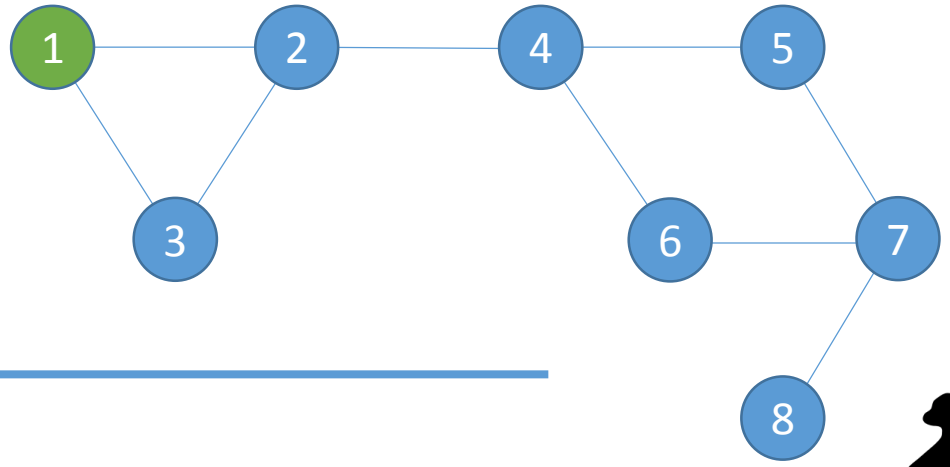
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



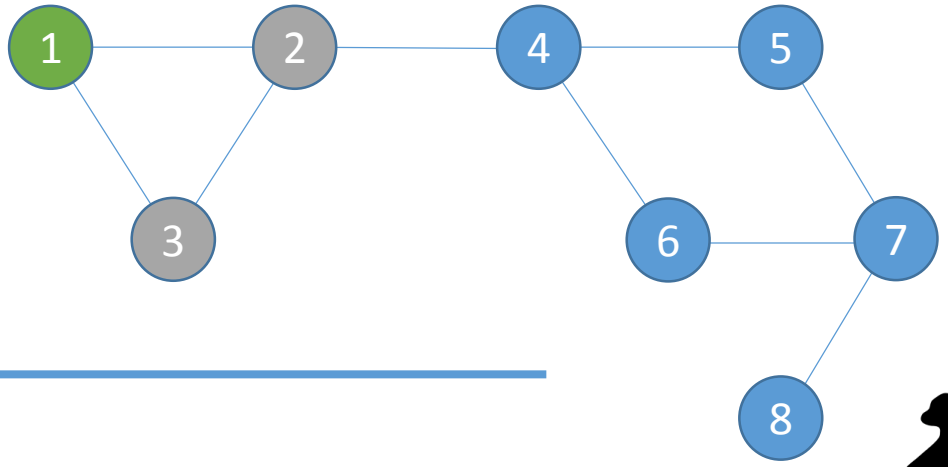
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



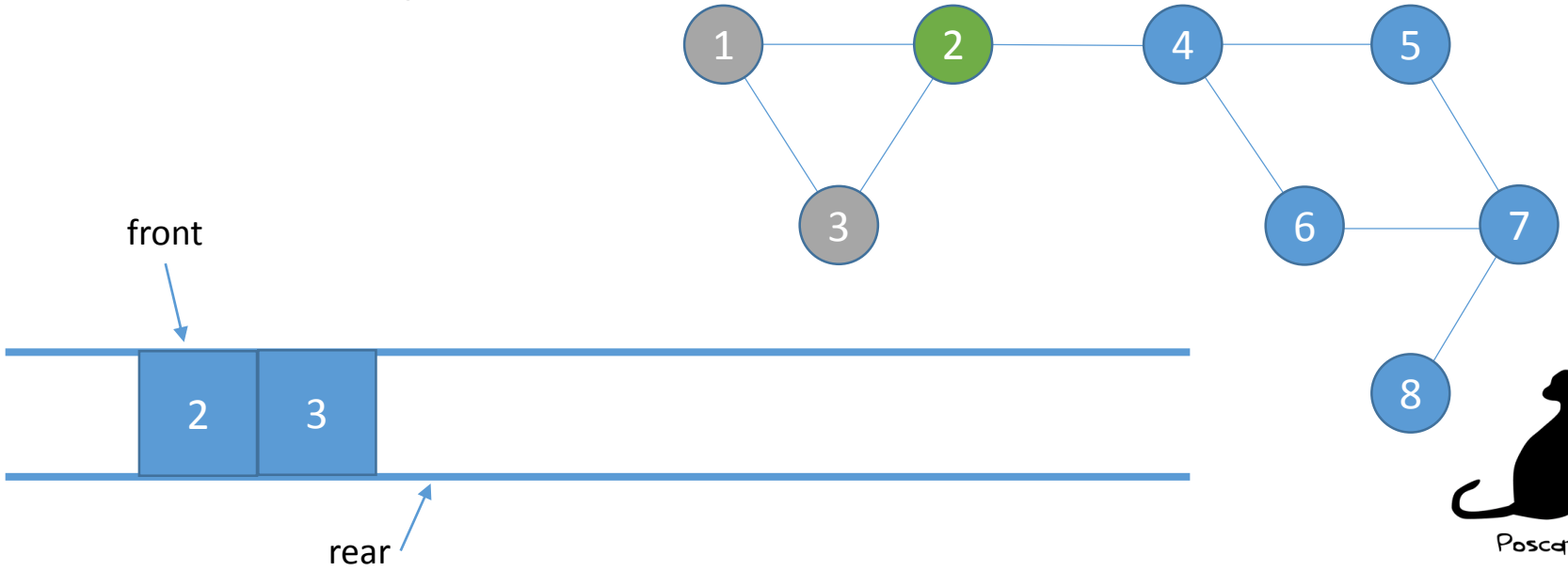
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



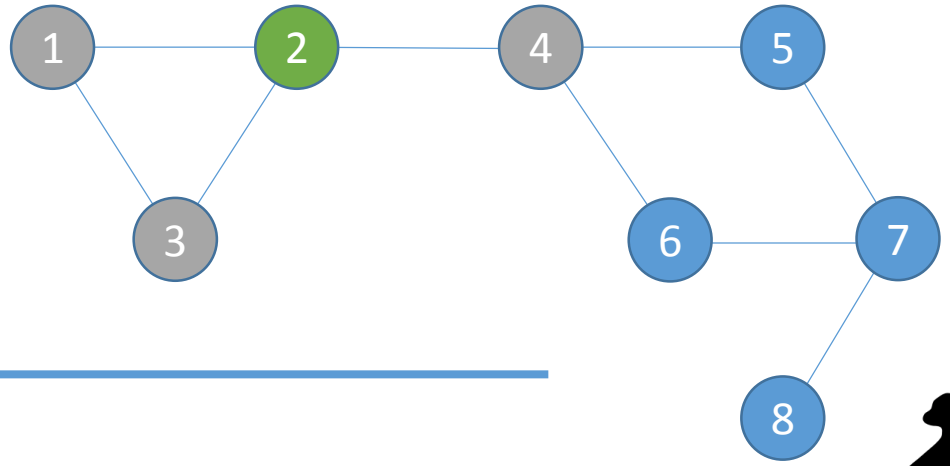
Graph Traversal

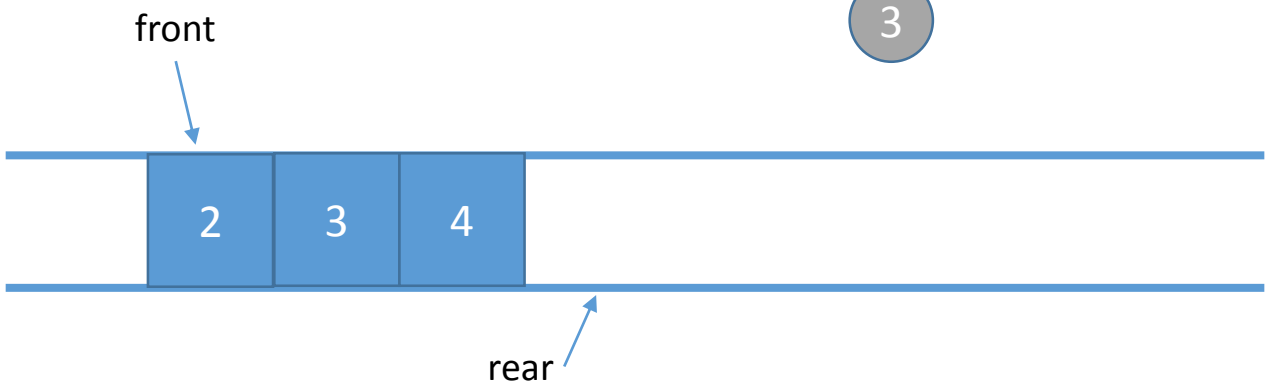
- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



Graph Traversal

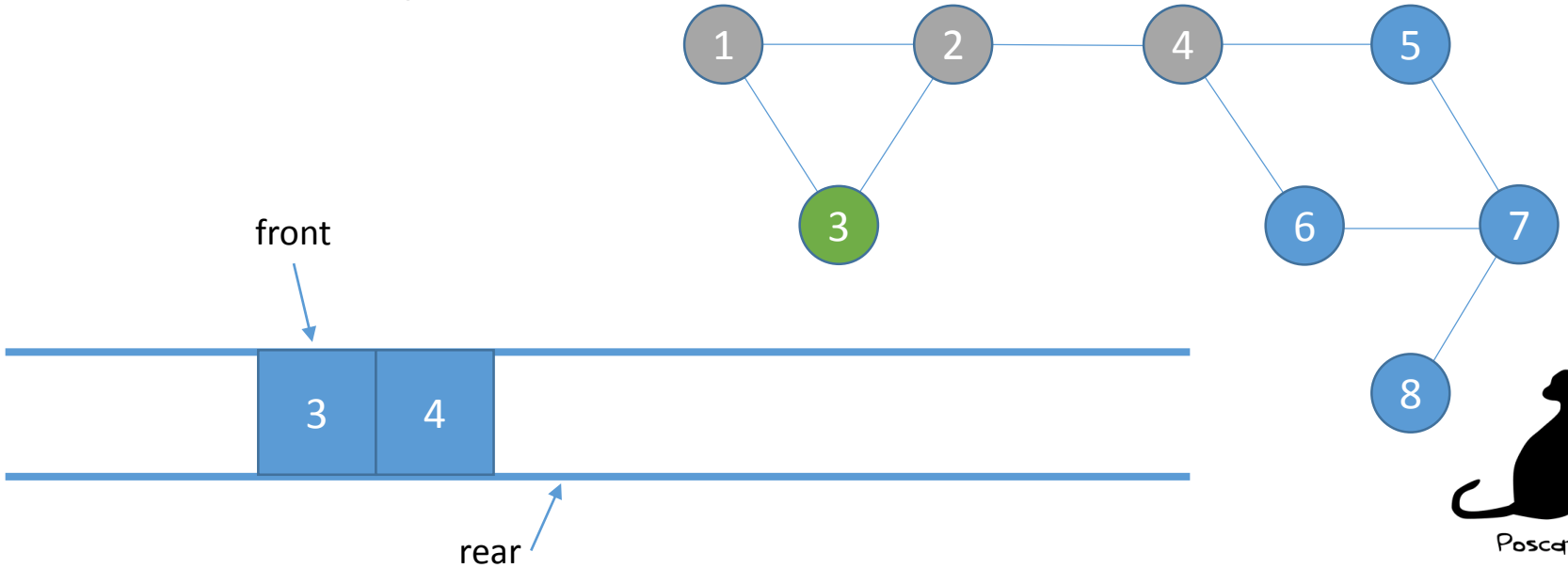
- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.





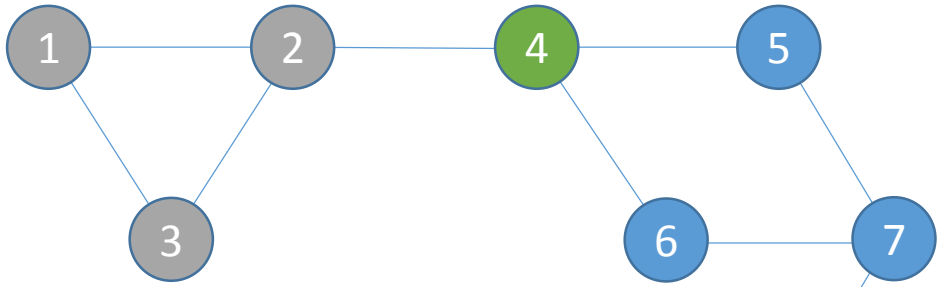
Graph Traversal

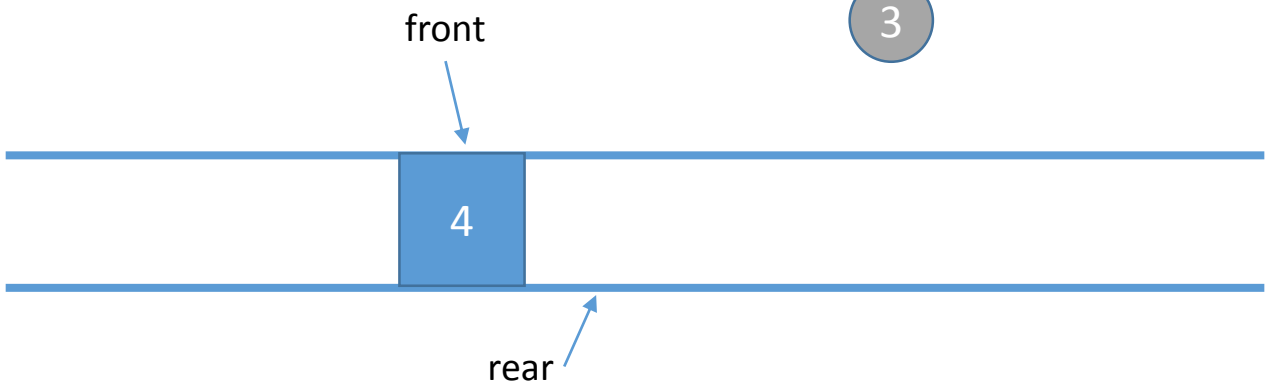
- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



Graph Traversal

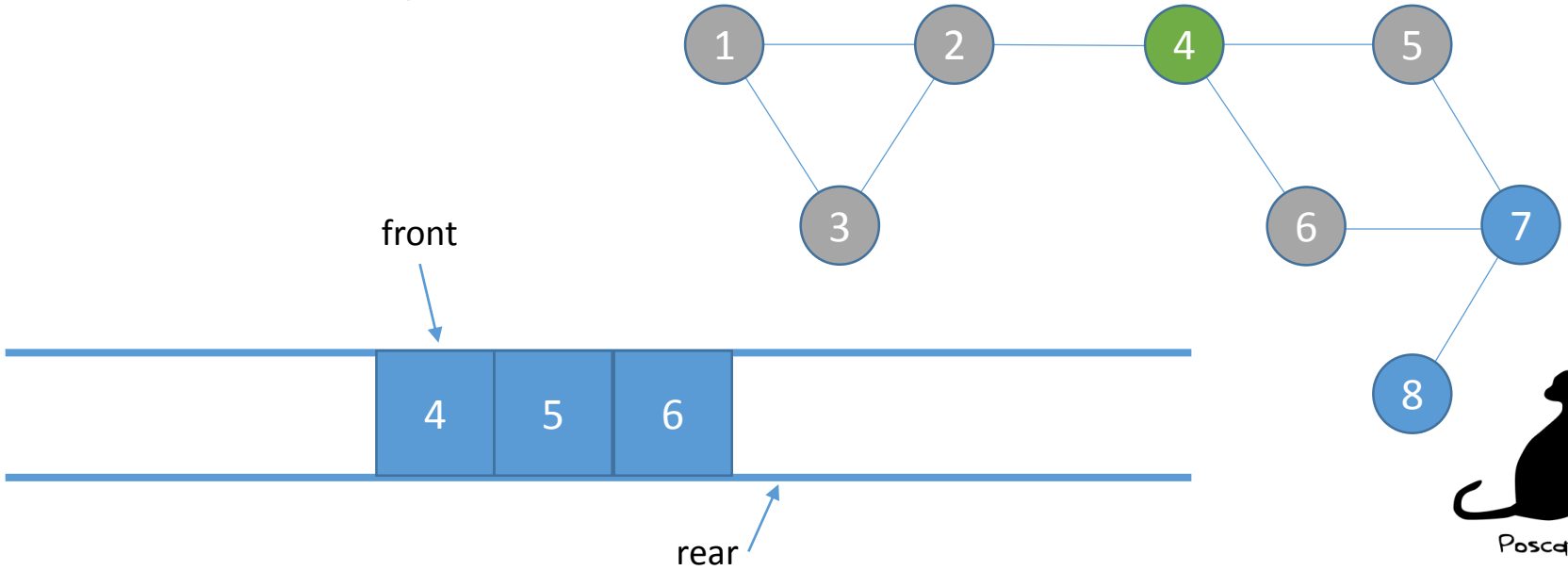
- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.





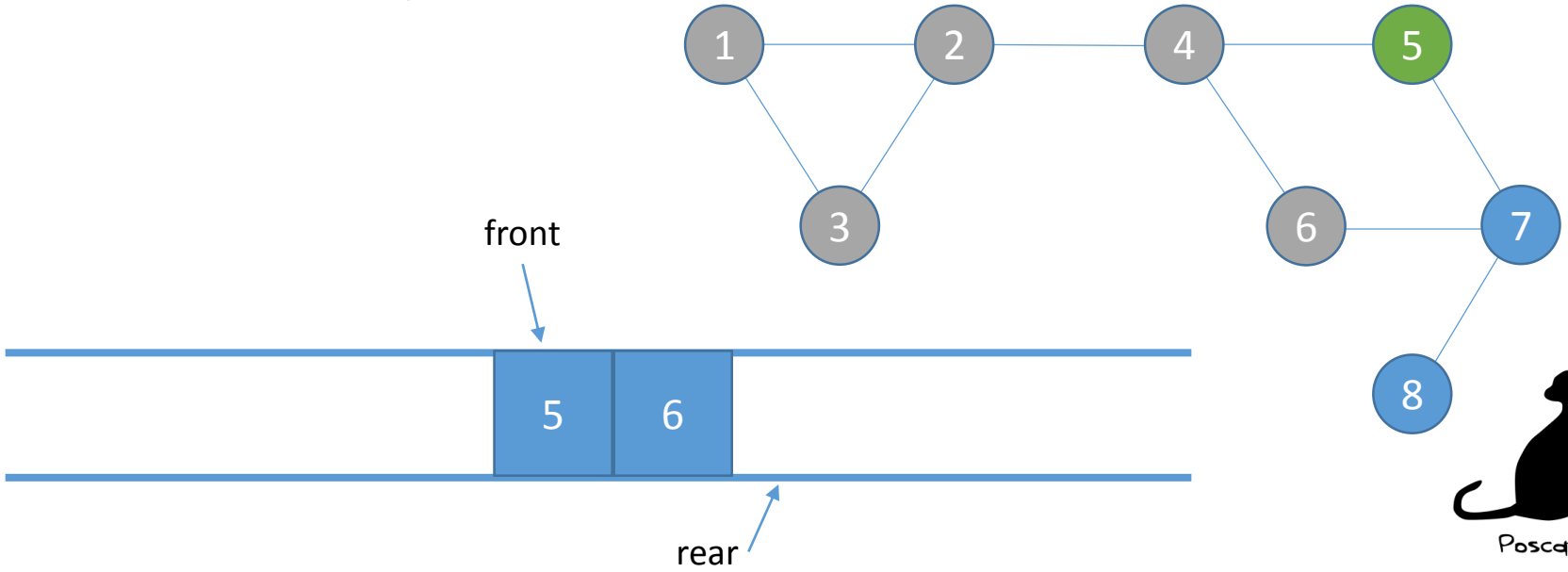
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



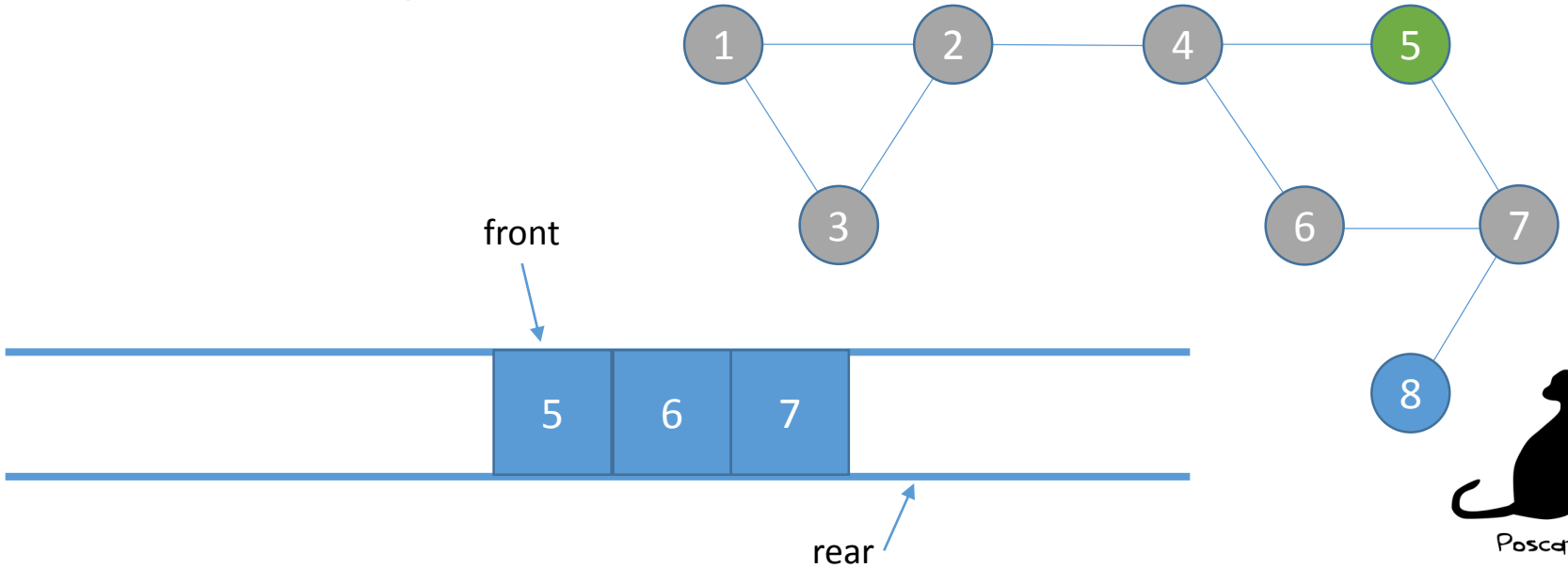
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



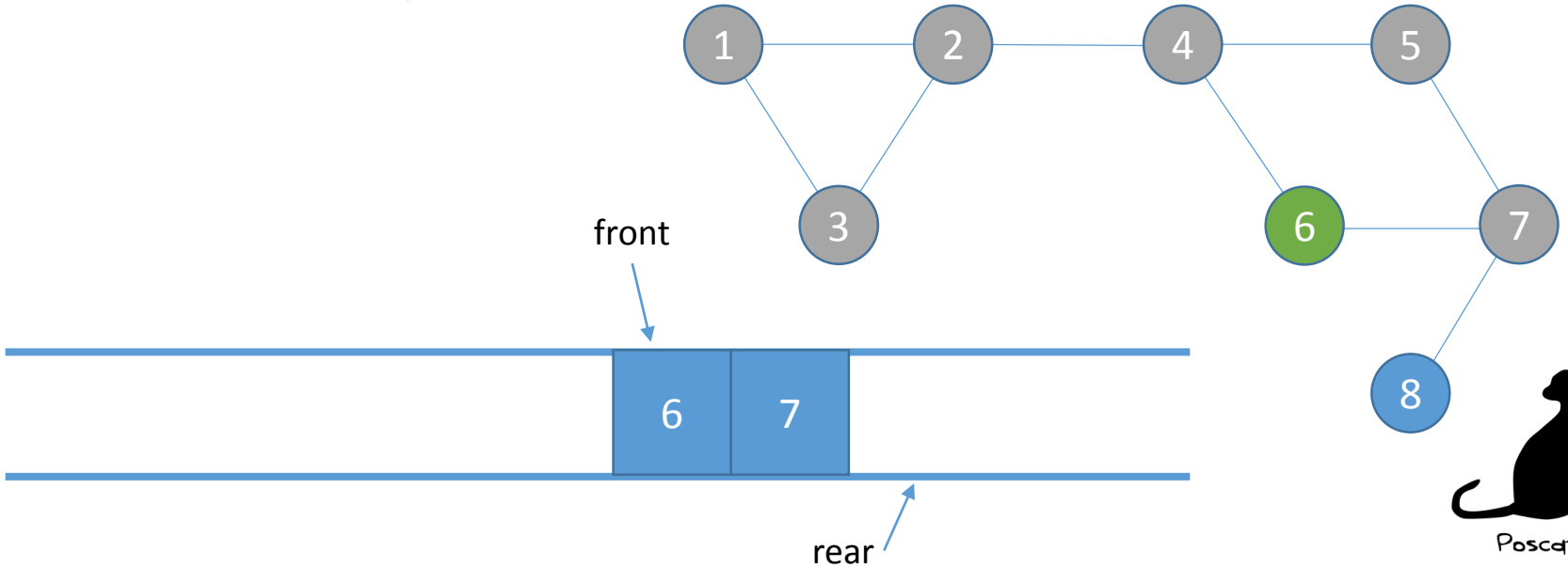
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



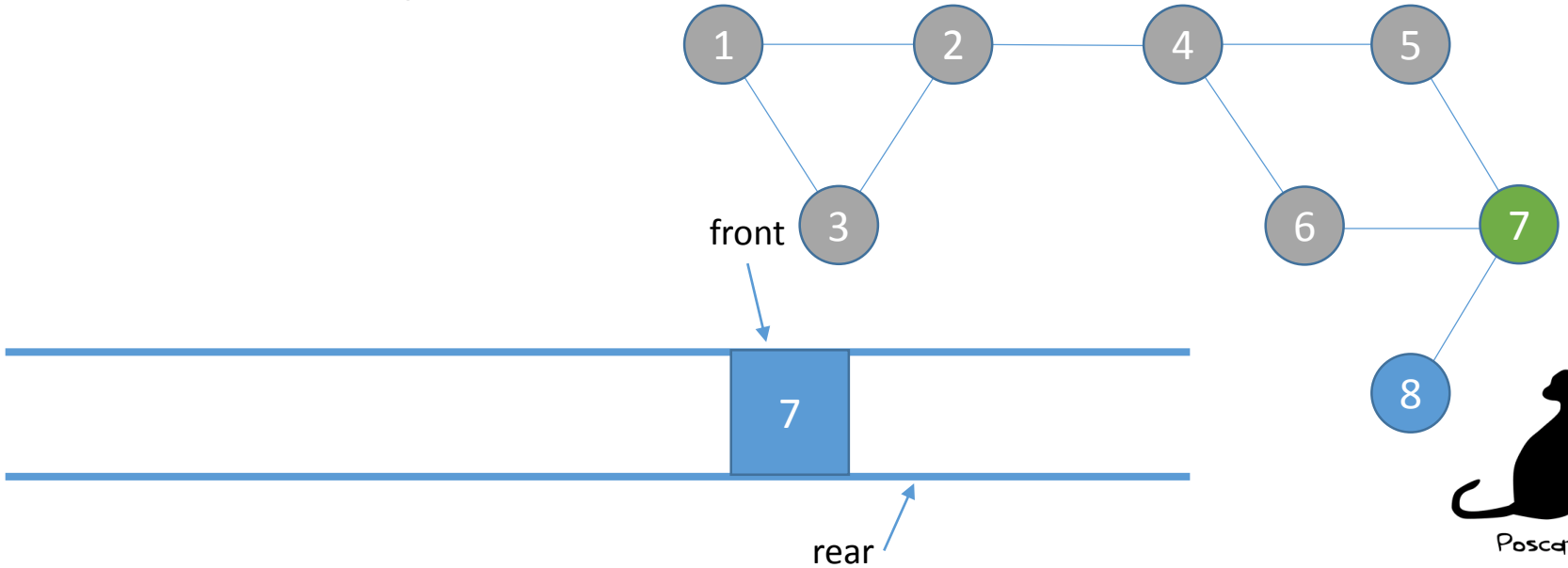
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



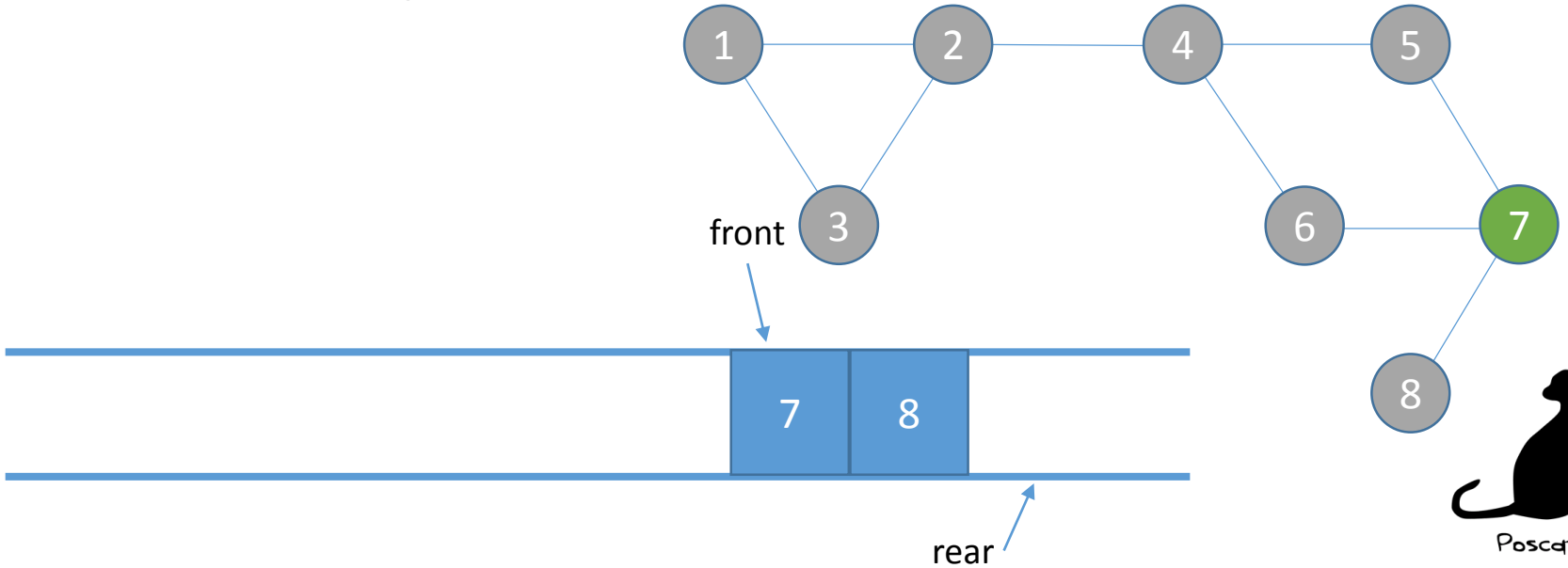
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



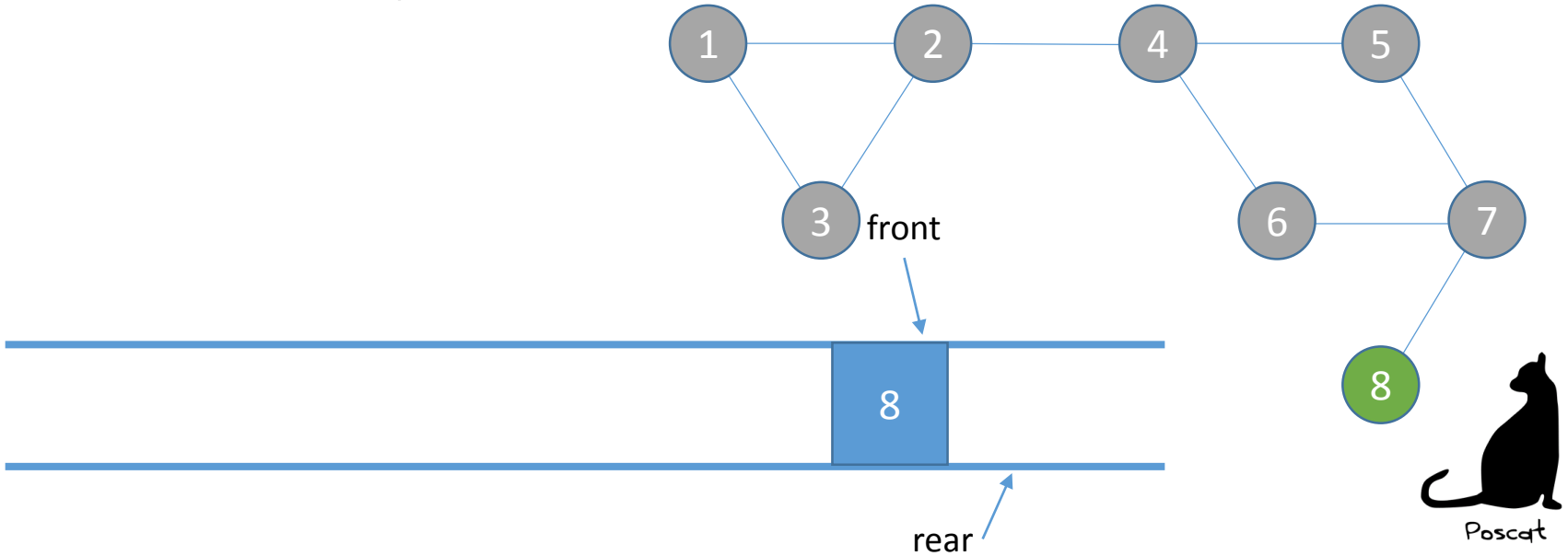
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



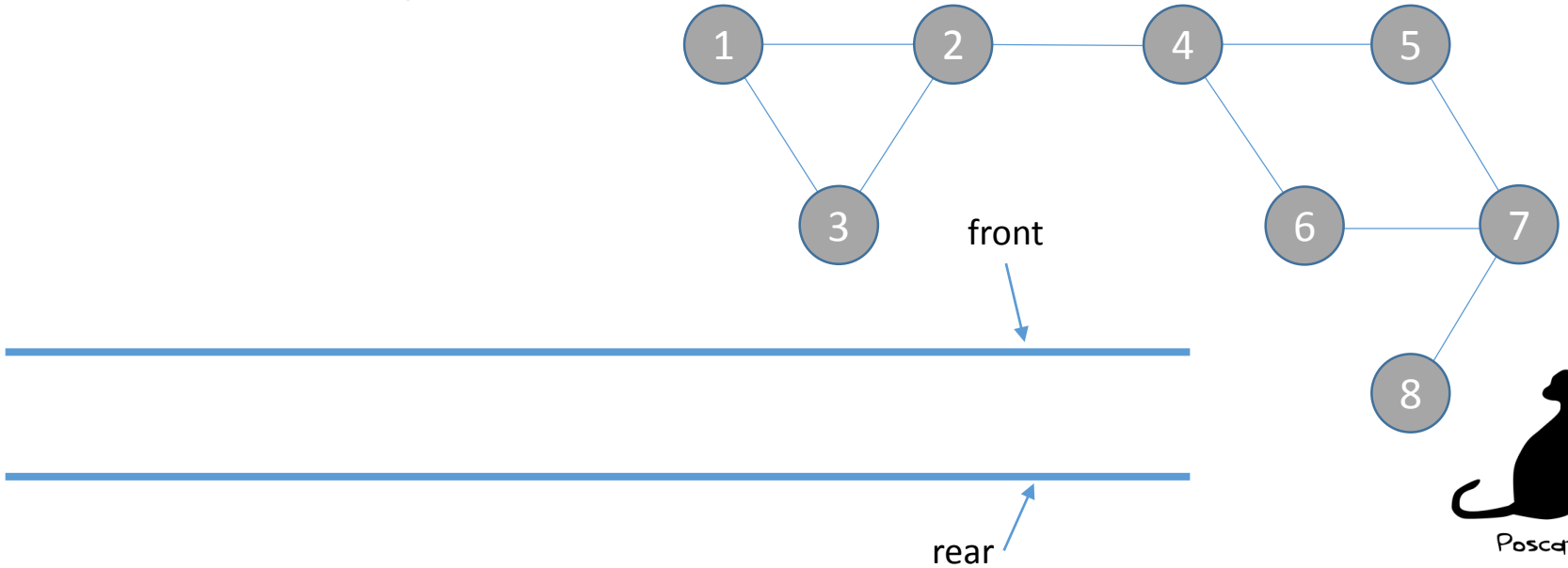
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



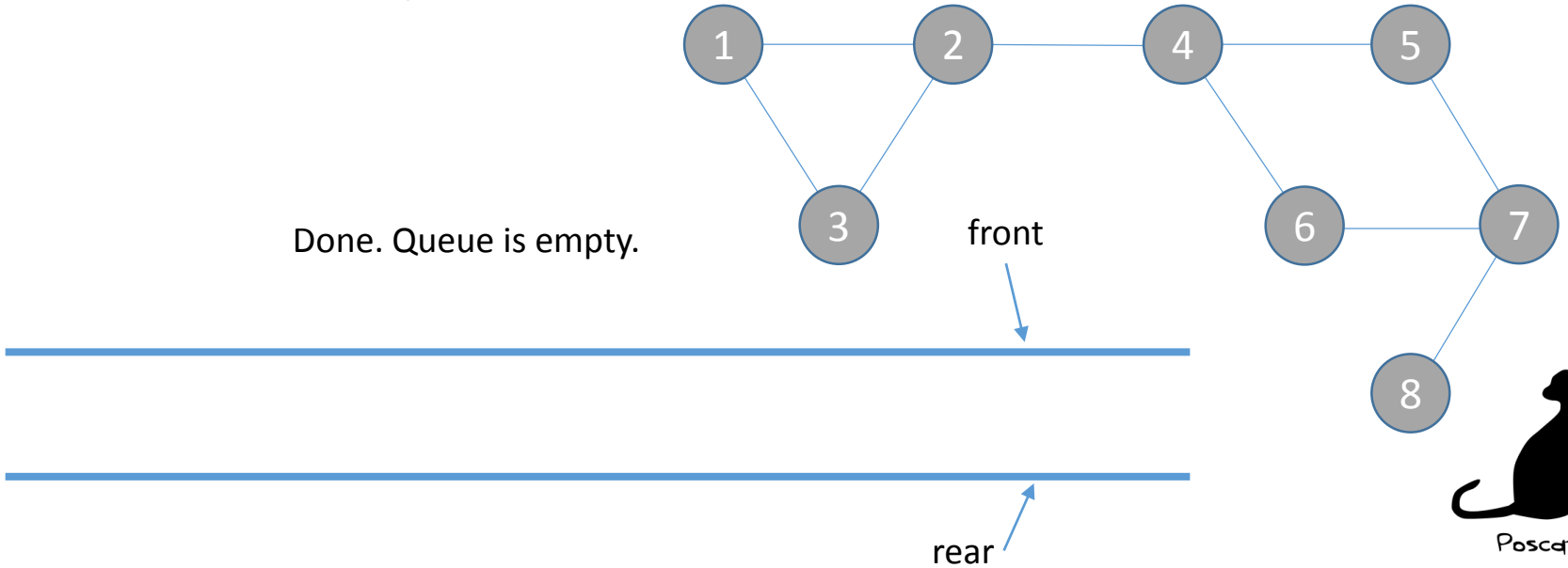
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



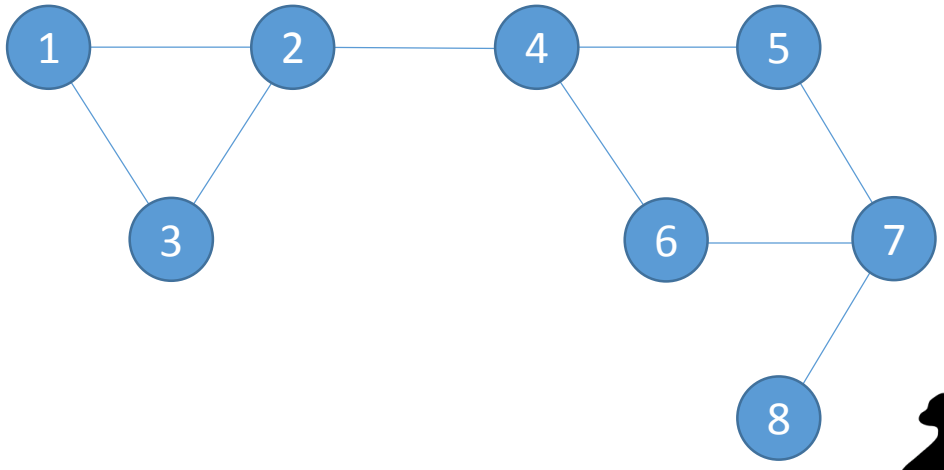
Graph Traversal

- Breadth First Search
  - Procedure
    1. Select a start vertex
    2. Push all the adjacent vertices into queue
    3. Pop to get a new vertex
    4. Go to step 2.



Graph Traversal

- DFS vs BFS
  - We have to choose what search technique to use
  - Example ?



Flood Fill

- Problem

  Given a map, categorize each cell as inner or outer cell

  Suppose that the outer-most cell is always 0

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  Given a map, categorize each cell as inner or outer cell

  Suppose that the outer-most cell is always 0

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  Any idea ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  Any idea ? Modeling it as a graph !

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  Any idea ? Modeling it as a graph !

  Each cell is considered as a vertex. How about edge ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  Any idea ? Modeling it as a graph !

  there is a edge if two vertices are adjacent in the map

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  Perform DFS or BFS starting from the outermost cell !

  However, we never move to the '1' cell

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  BFS ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  Can we apply this flood fill algorithm to solve a problem ?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	1	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Flood Fill

- Problem

  Find the minimum length from the start cell and the end cell

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0	0	0
0	1	1	0	1	0	1	1	1	0
0	0	0	0	0	0	0	1	0	0

Flood Fill

- Problem

  Find the minimum length from the start cell and the end cell Here !

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0	0	0
0	1	1	0	1	0	1	1	1	0
0	0	0	0	0	0	0	1	0	0

Flood Fill

- Problem

  Find the minimum length from the start cell and the end cell

  However, there is another path which is longer !

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0	0	0
0	1	1	0	1	0	1	1	1	0
0	0	0	0	0	0	0	1	0	0

Flood Fill

- Problem

  How can we find the optimal path ?

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0	0	0
0	1	1	0	1	0	1	1	1	0
0	0	0	0	0	0	0	1	0	0

Flood Fill

- Problem

  How can we find the optimal path ?

  Let $P(i,j) = the\ shortest\ length\ to\ (i,j)$, and fill it by using flood fill

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0	0	0
0	1	1	0	1	0	1	1	1	0
0	0	0	0	0	0	0	1	0	0

Flood Fill

- Problem

  Which traversal is better? DFS or BFS ?

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0	0	0
0	1	1	0	1	0	1	1	1	0
0	0	0	0	0	0	0	1	0	0

Flood Fill

- Problem

  Which traversal is better? DFS or BFS ?

  Definitely, BFS is better in this time. Think about the reason

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0	0	0
0	1	1	0	1	0	1	1	1	0
0	0	0	0	0	0	0	1	0	0

Flood Fill

- Problem

  How can we find the optimal path ?

  Let $P(i,j) = the\ shortest\ length\ to\ (i,j)$, and fill it by using flood fill

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0	0	0
0	1	1	0	1	0	1	1	1	0
0	0	0	0	0	0	0	1	0	0

Flood Fill

- Problem

  How can we find the optimal path ?

  Let $P(i,j) = the\ shortest\ length\ to\ (i,j)$, and fill it by using flood fill

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0	0	0
1	1	1	0	1	0	1	1	1	0
0	1	0	0	0	0	0	1	0	0

Flood Fill

- Problem

  How can we find the optimal path ?

  Let $P(i,j) = the\ shortest\ length\ to\ (i,j)$, and fill it by using flood fill

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0
2	1	0	0	1	0	1	0	0	0
1	1	1	0	1	0	1	1	1	0
0	1	2	0	0	0	0	1	0	0

Flood Fill

- Problem

  How can we find the optimal path ?

  Let $P(i,j) = the\ shortest\ length\ to\ (i,j)$, and fill it by using flood fill

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	0	0	1	0
3	1	1	0	1	0	1	0	1	0
2	1	0	0	1	0	1	0	0	0
1	1	1	0	1	0	1	1	1	0
0	1	2	3	0	0	0	1	0	0

Flood Fill

- Problem

  How can we find the optimal path ?

  Let $P(i,j) = the\ shortest\ length\ to\ (i,j)$, and fill it by using flood fill

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
0	1	1	1	1	0	0	0	1	0
0	1	1	0	0	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
4	1	1	0	1	0	0	0	1	0
3	1	1	0	1	0	1	0	1	0
2	1	0	0	1	0	1	0	0	0
1	1	1	4	1	0	1	1	1	0
0	1	2	3	4	0	0	1	0	0

Flood Fill

- Problem

  How can we find the optimal path ?

  Let $P(i,j) = the\ shortest\ length\ to\ (i,j)$, and fill it by using flood fill

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

Flood Fill

- Problem

  How can we find the optimal path ?

  Let $P(i,j) = the\ shortest\ length\ to\ (i,j)$, and fill it by using flood fill

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

Flood Fill

- Problem

  How can we find the optimal path ?

  Let $P(i,j) = the\ shortest\ length\ to\ (i,j)$, and fill it by using flood fill

0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
7	1	1	1	1	0	0	0	1	0
6	1	1	0	0	0	1	0	1	0
5	1	1	8	0	0	1	1	0	0
4	1	1	7	1	0	0	0	1	0
3	1	1	6	1	8	1	0	1	0
2	1	6	5	1	7	1	0	0	0
1	1	1	4	1	6	1	1	1	0
0	1	2	3	4	5	6	1	0	0

Flood Fill

- Problem

  How can we find the optimal path ?

  Let $P(i,j) = the\ shortest\ length\ to\ (i,j)$, and fill it by using flood fill

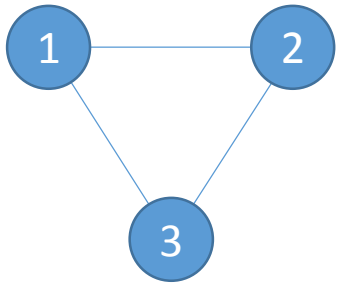
0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
7	1	1	1	1	0	0	0	1	0
6	1	1	9	0	0	1	0	1	0
5	1	1	8	9	0	1	1	0	0
4	1	1	7	1	9	0	0	1	0
3	1	1	6	1	8	1	0	1	0
2	1	6	5	1	7	1	0	0	0
1	1	1	4	1	6	1	1	1	0
0	1	2	3	4	5	6	1	0	0

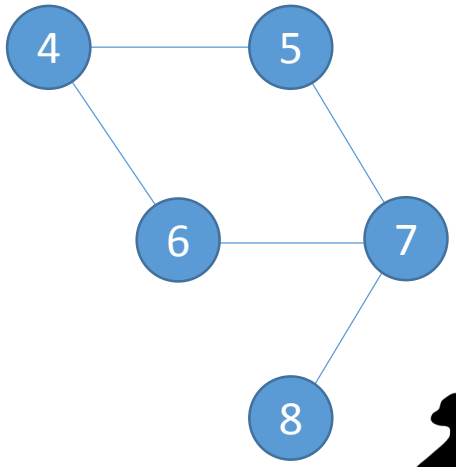
Fill it yourself ☺

Connected Component

- Problem

  Given a graph, determine whether this graph is connected or not

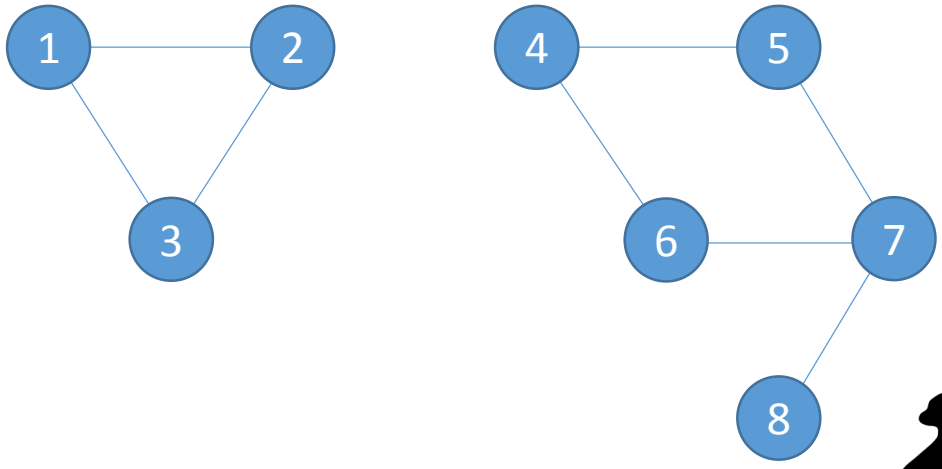




Connected Component

- Problem

  Given a graph, determine whether this graph is connected or not

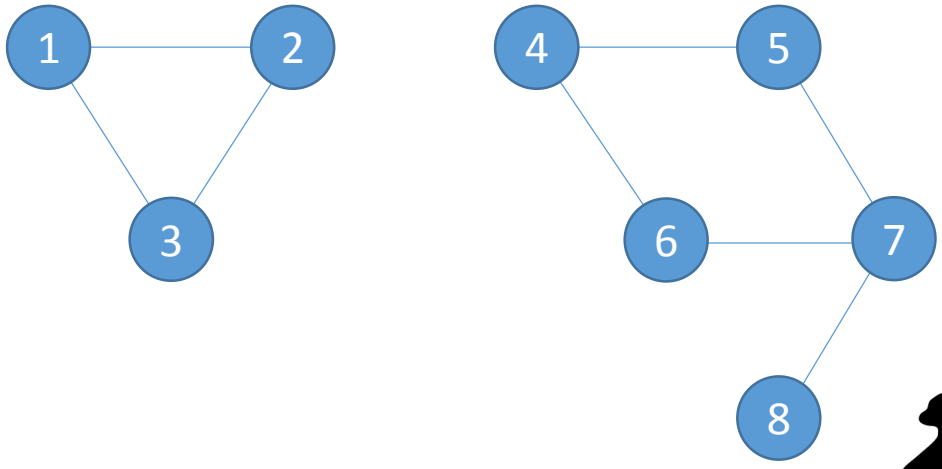


Is it connected ?

Connected Component

- Problem

  Given a graph, determine whether this graph is connected or not



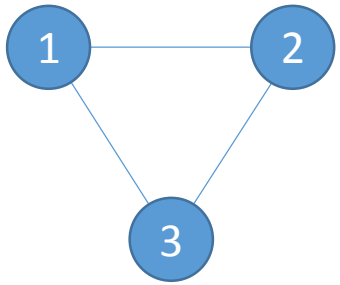
Is it connected ? **NO!**

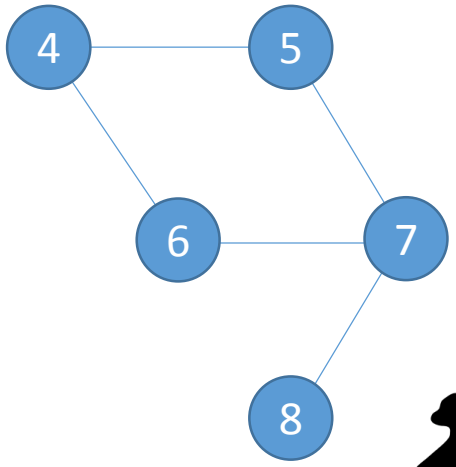
Connected Component

- Problem

  Given a graph, determine whether this graph is connected or not

  One traversal from any start vertex is enough. Easy.





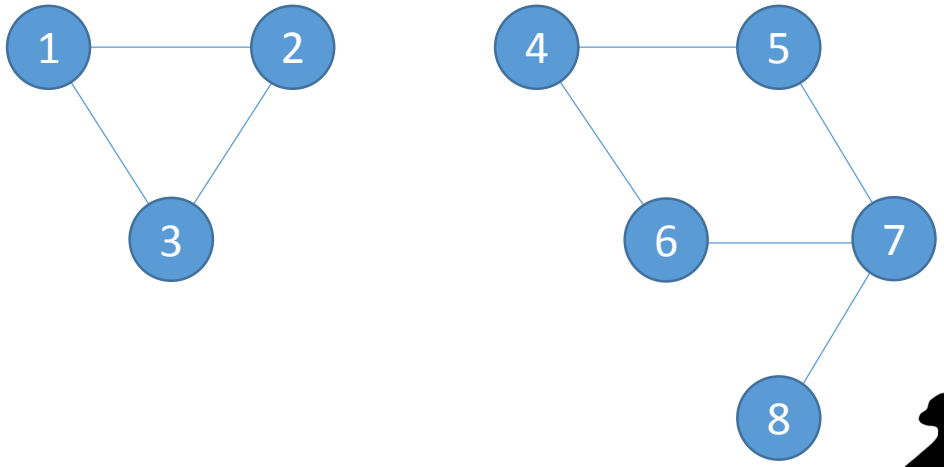
Connected Component

- Problem

  Given a graph, determine whether this graph is connected or not

  One traversal from any start vertex is enough. Easy.

  What if this graph is directed graph ?



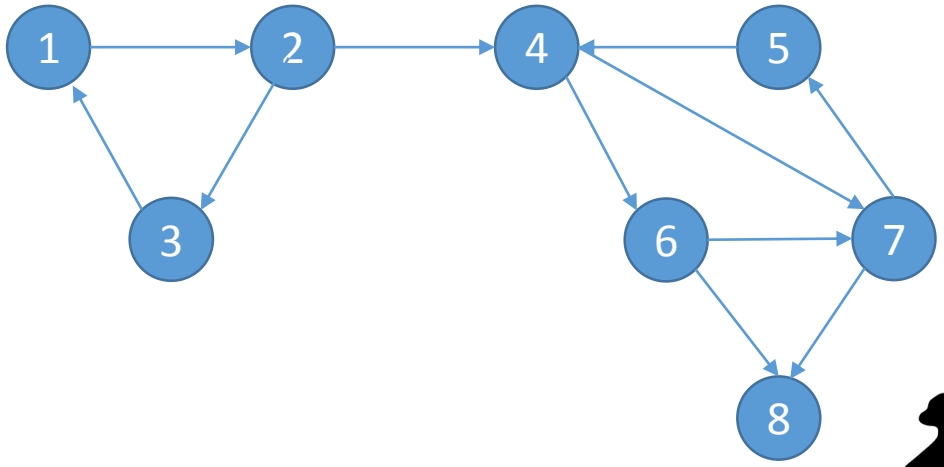
Connected Component

- Problem

  Given a graph, determine whether this graph is connected or not

  One traversal from any start vertex is enough. Easy.

  What if this graph is directed graph ?

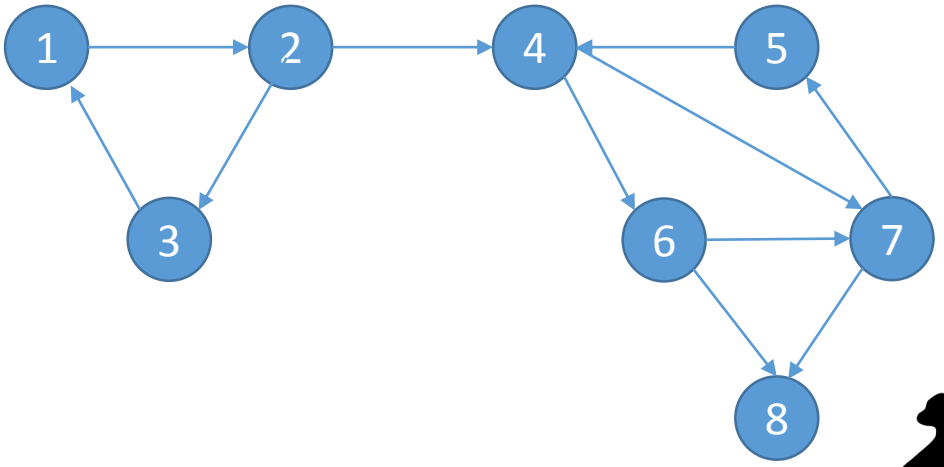


Strongly Connected Component

- Problem

  Given a direct graph, Find Strongly Connected Component

  SCC : a graph is strongly connected if every vertex is reachable from every other vertex

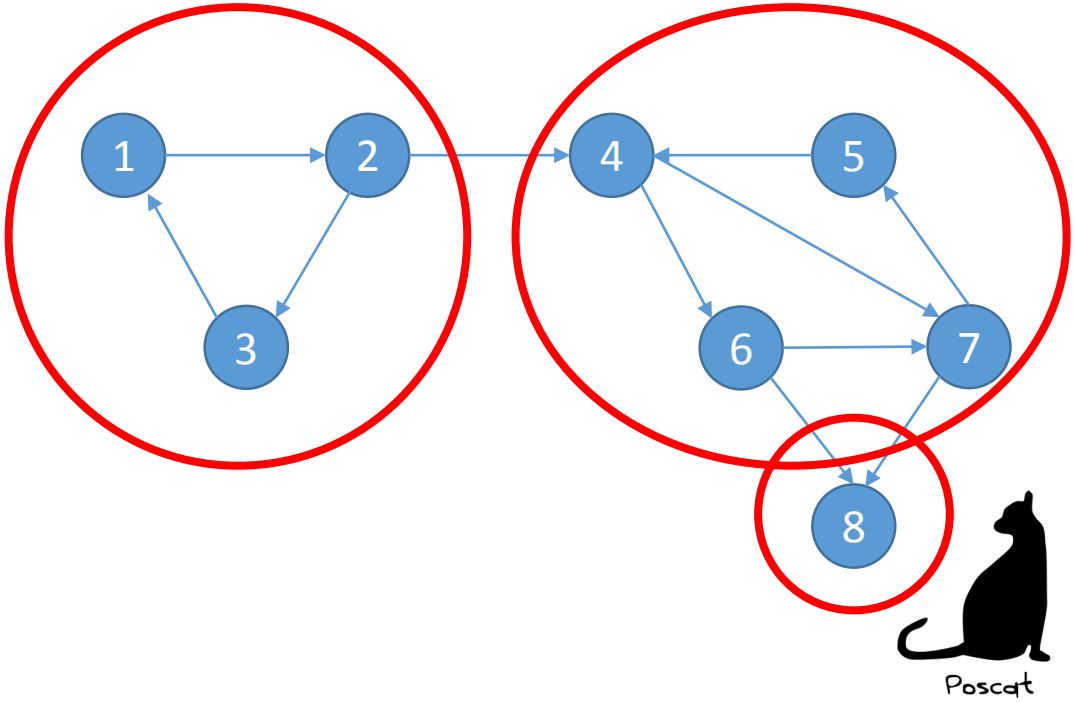


Strongly Connected Component

- Problem

  Given a direct graph, Find Strongly Connected Component

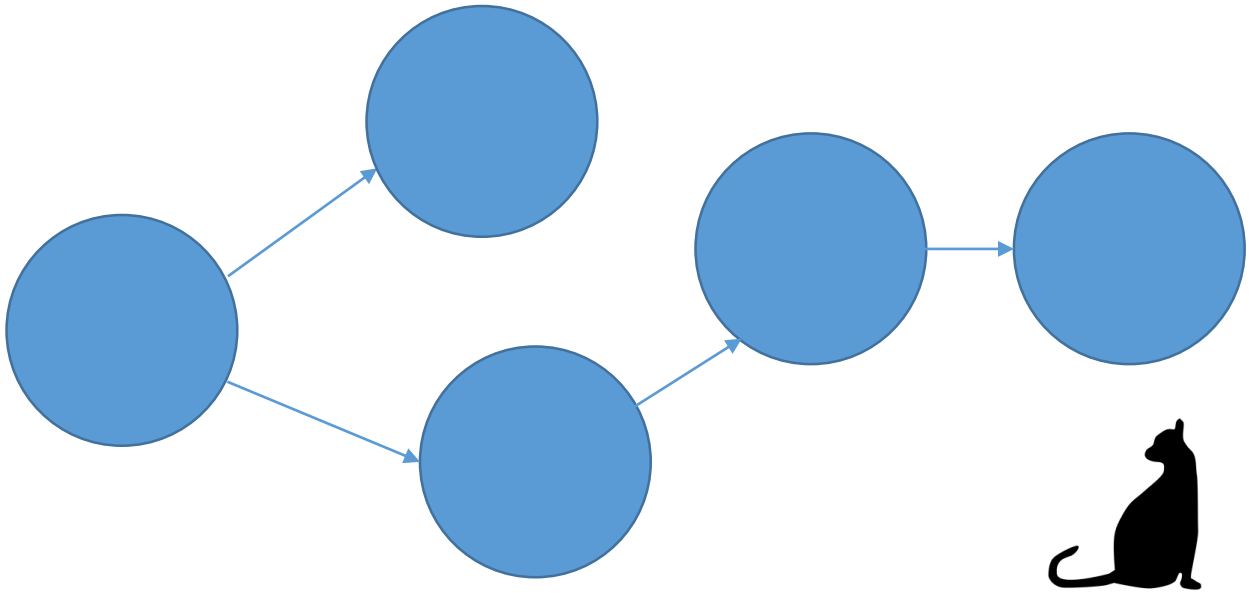
  SCC : a graph is strongly connected if every vertex is reachable from every other vertex



Strongly Connected Component

- Problem

  Suppose that there is strongly connected component. (just imagine)



Strongly Connected Component

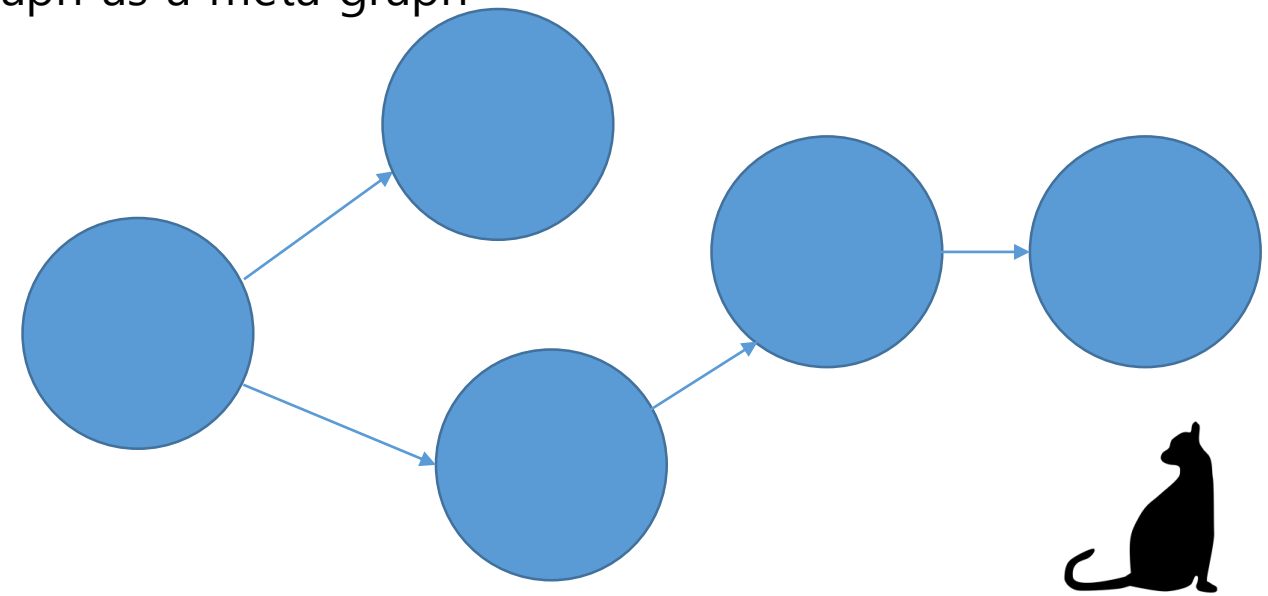
- Problem

  Suppose that there is strongly connected component. (just imagine)

  Then we can make a graph consisting of "big" vertex

  (big vertex makes a strongly connected component)

  Let me call this graph as a meta graph

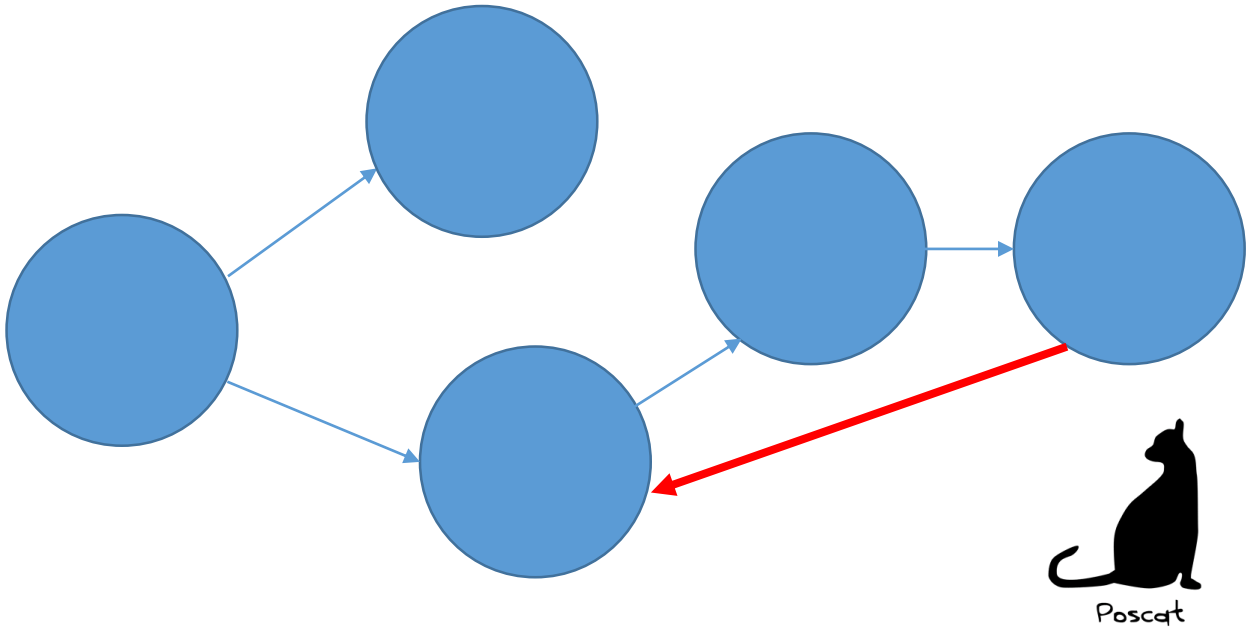


Strongly Connected Component

- Problem

  Trivially, this meta graph has no cycle.

  If not, they had to shrink into a vertex

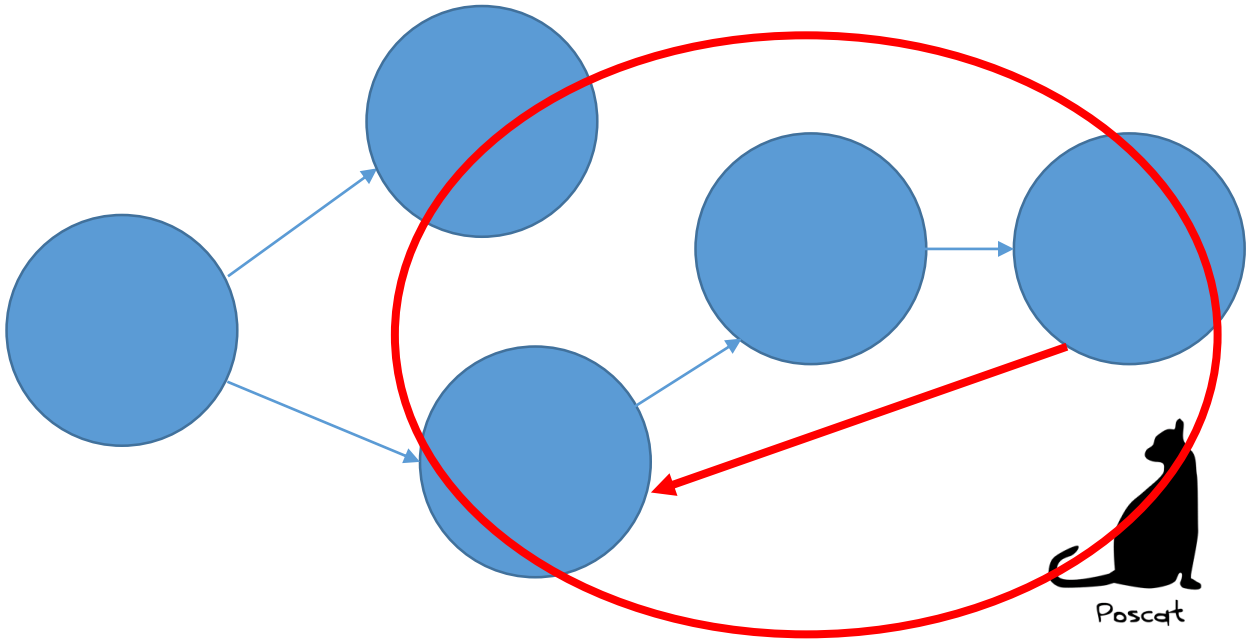


Strongly Connected Component

- Problem

  Trivially, this meta graph has no cycle.

  If not, they had to shrink into a vertex

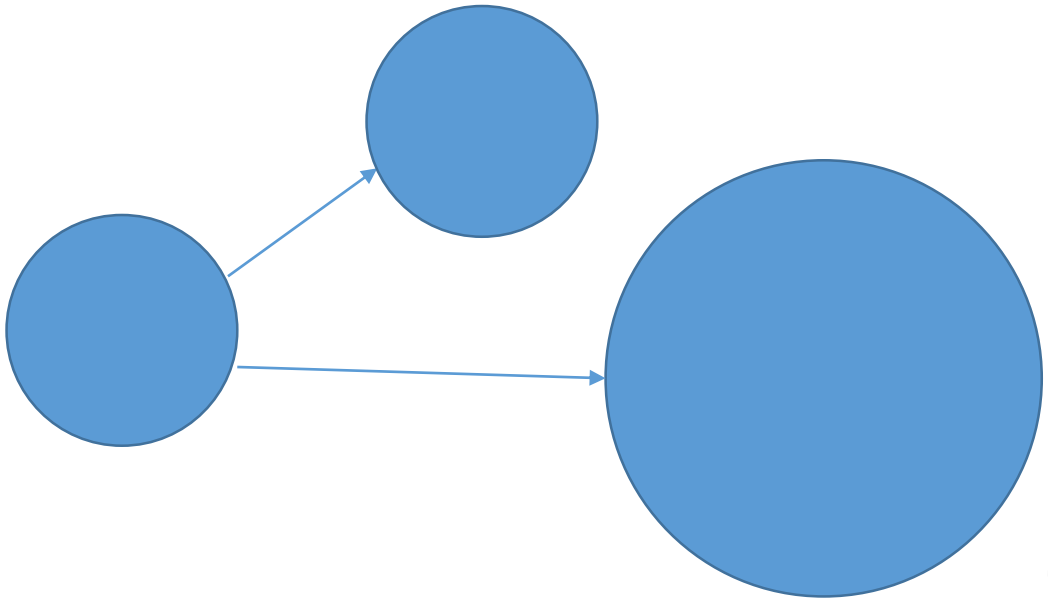


Strongly Connected Component

- Problem

  Trivially, this meta graph has no cycle.

  If not, they had to shrink into a vertex

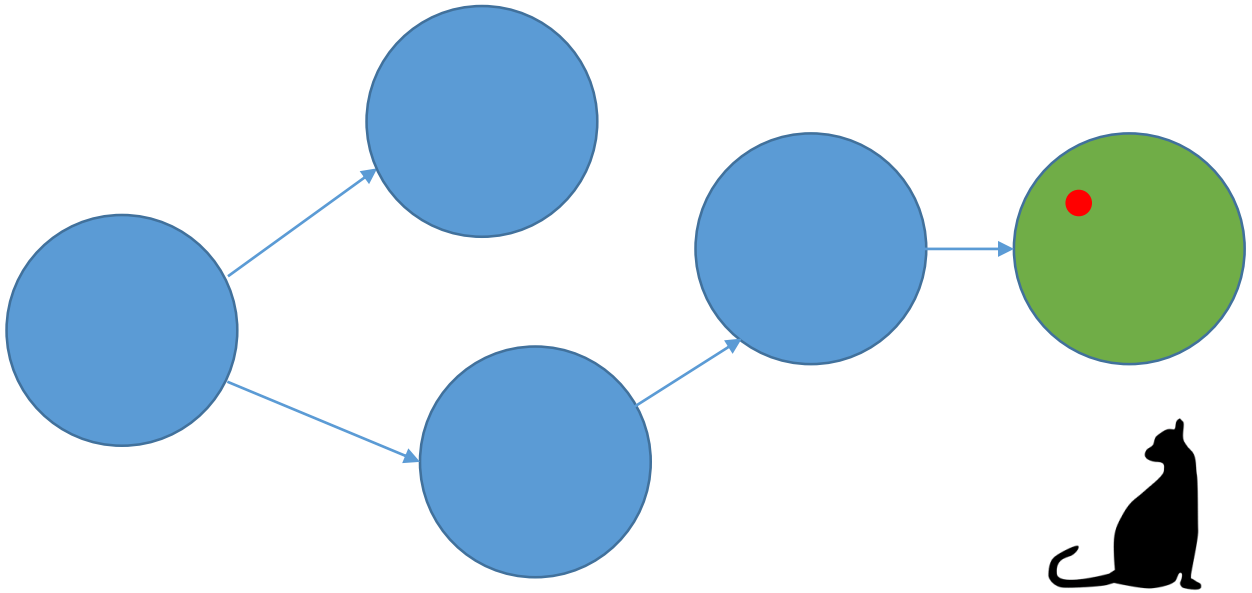


Strongly Connected Component

- Problem

  Think about the leaf big node. (a big node which has no outer edge)

  Choose one vertex from the leaf big node.

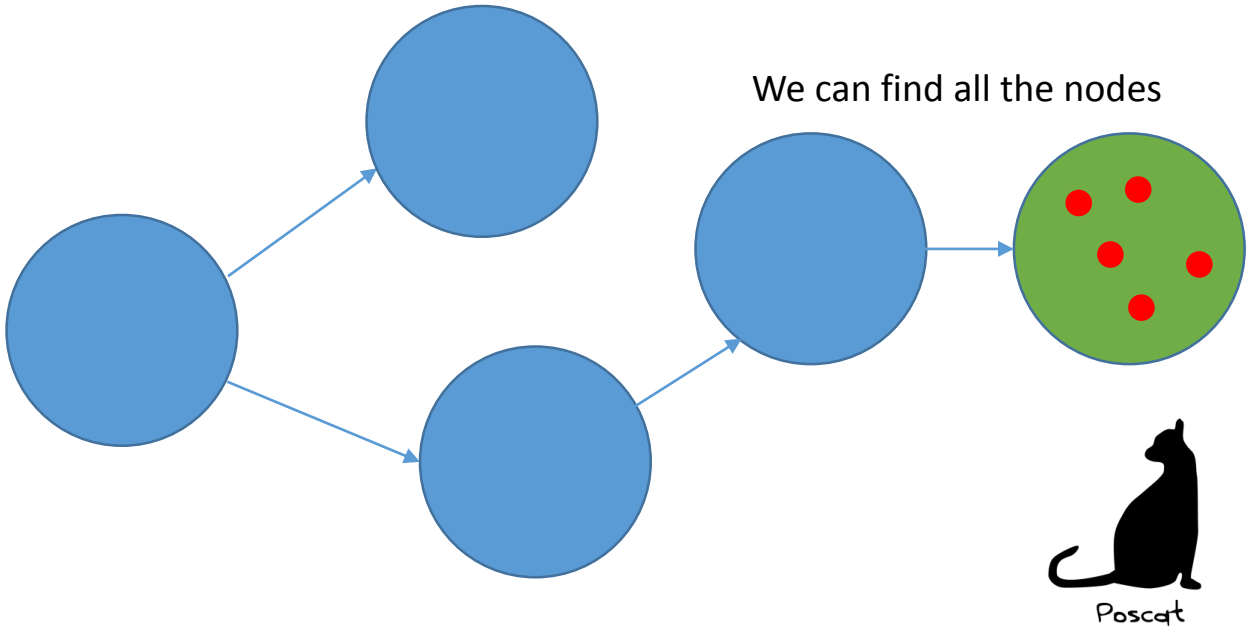


Strongly Connected Component

- Problem

  If we perform traverse started from the vertex, then it will traverse whole big node because the big node is SCC

  If not, contradiction.

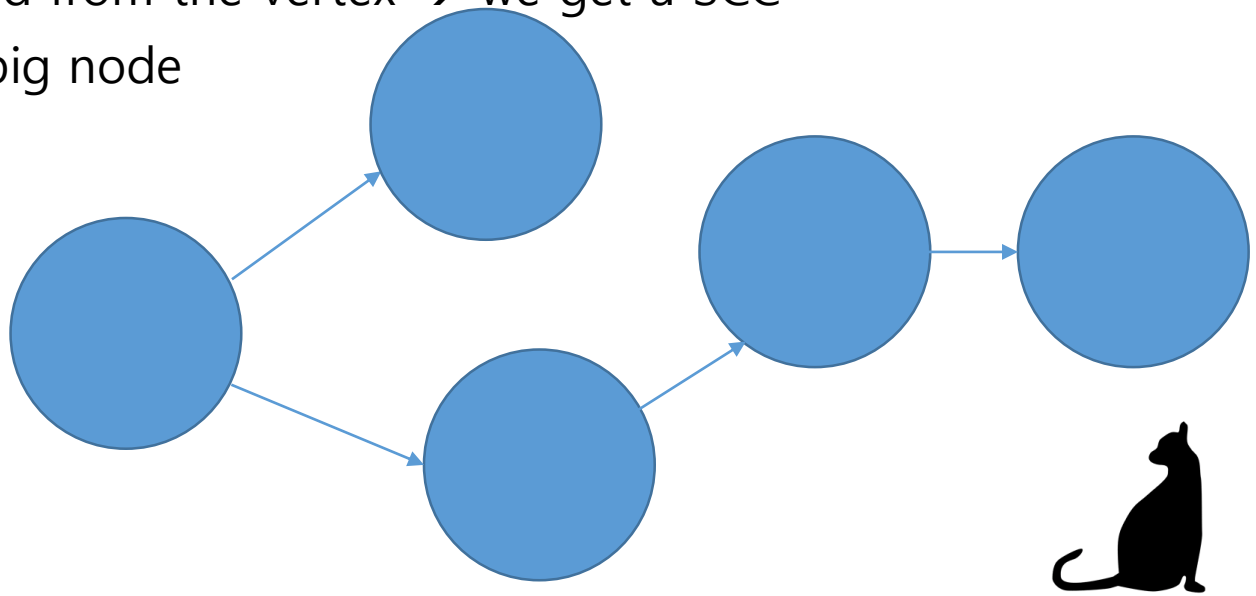


Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex → we get a SCC
  3. Remove that big node
  4. Repeat it

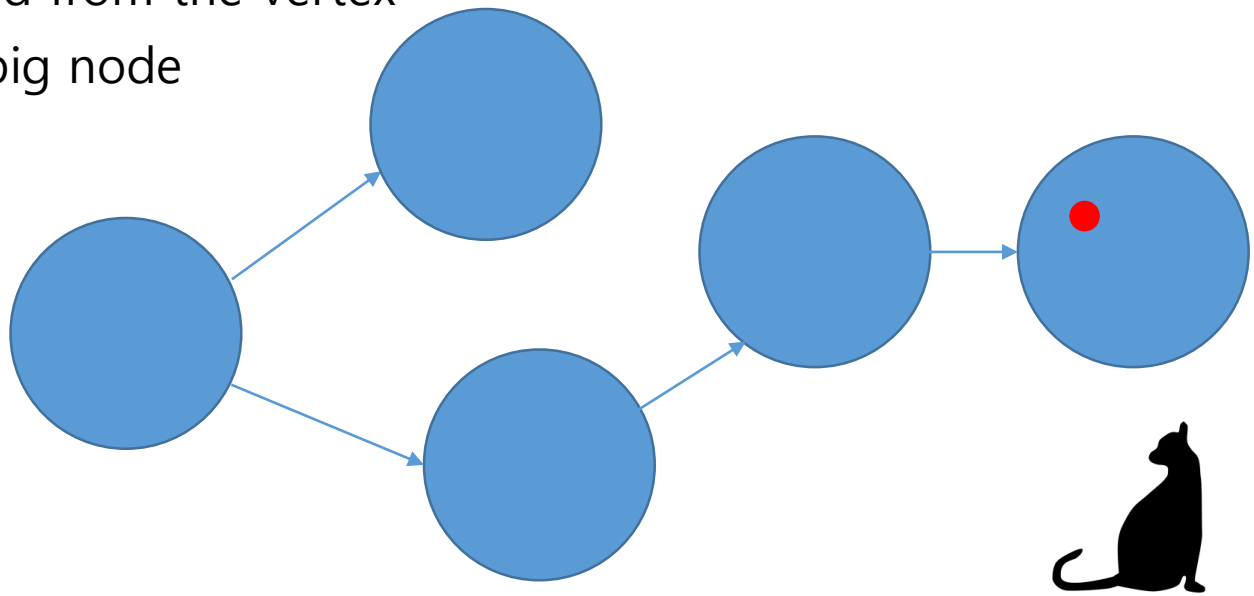


Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

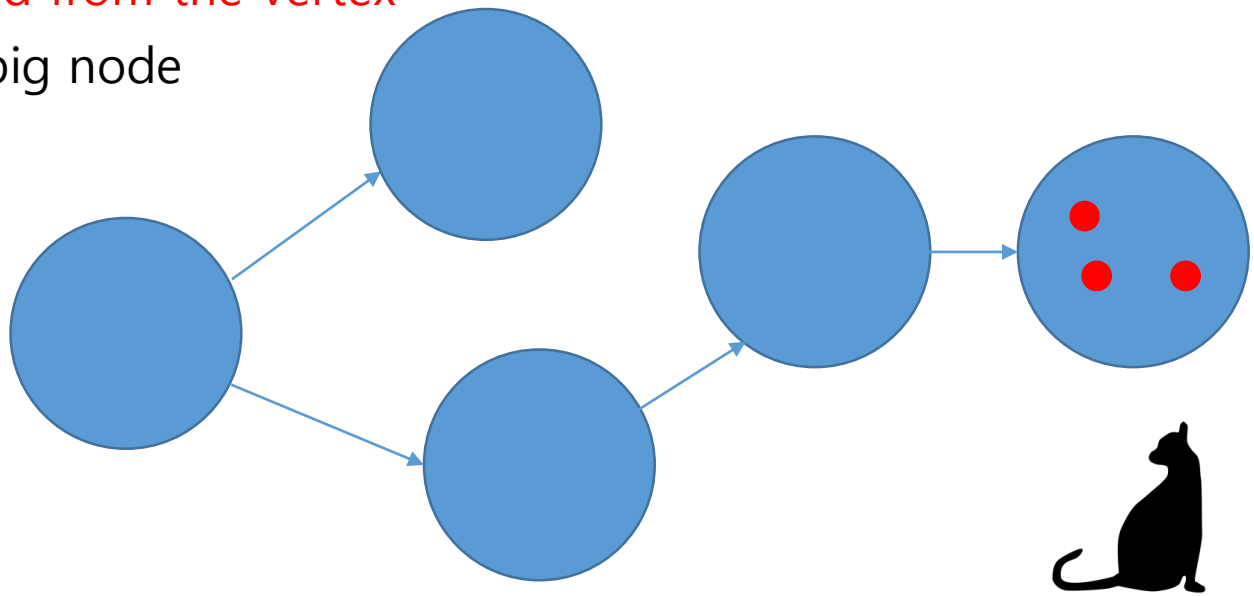


Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

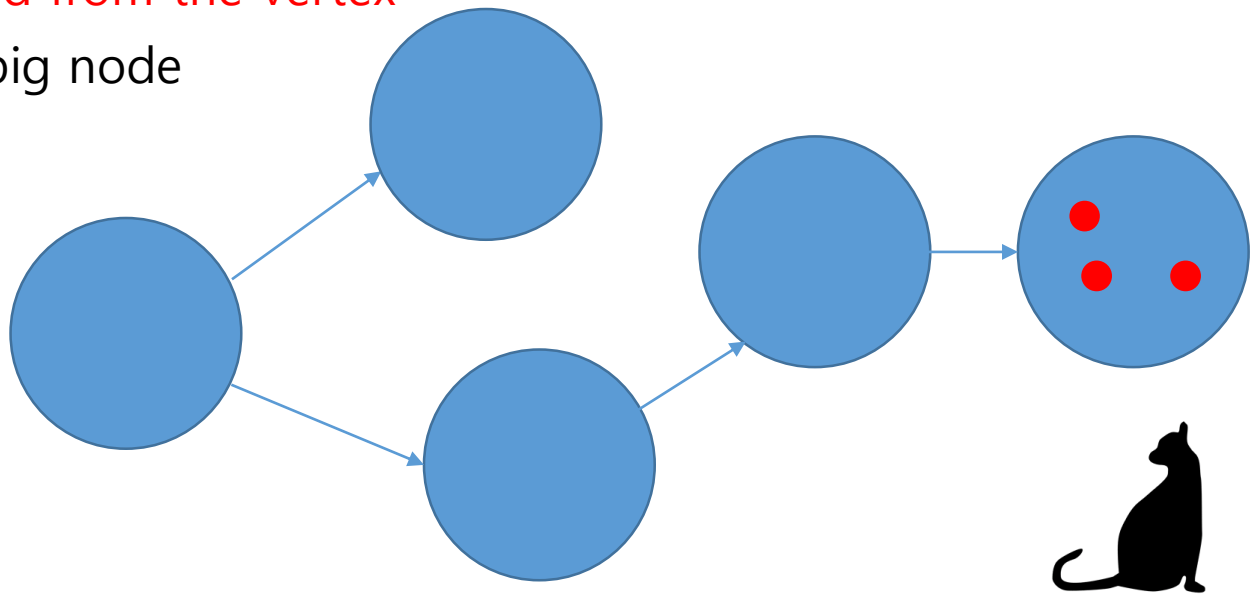


Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

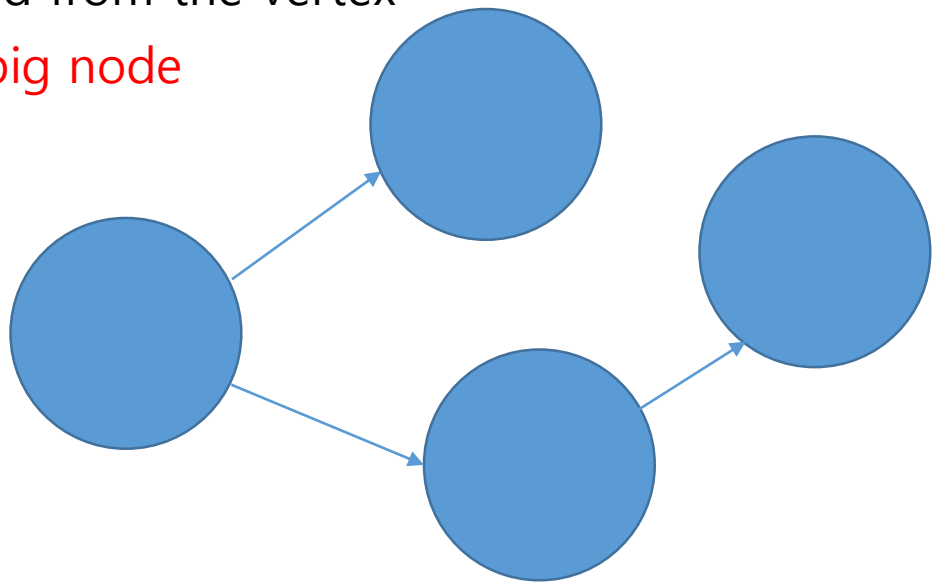


Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

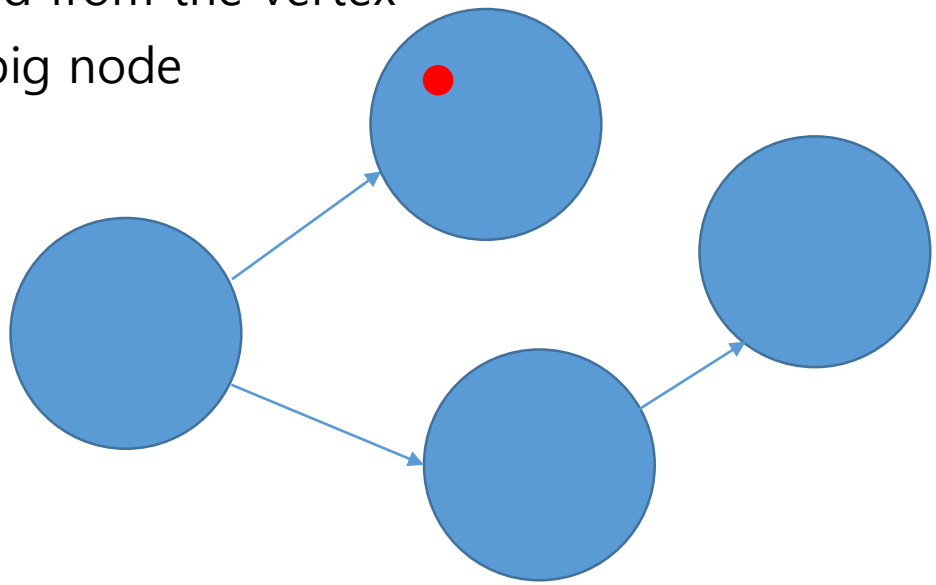


Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

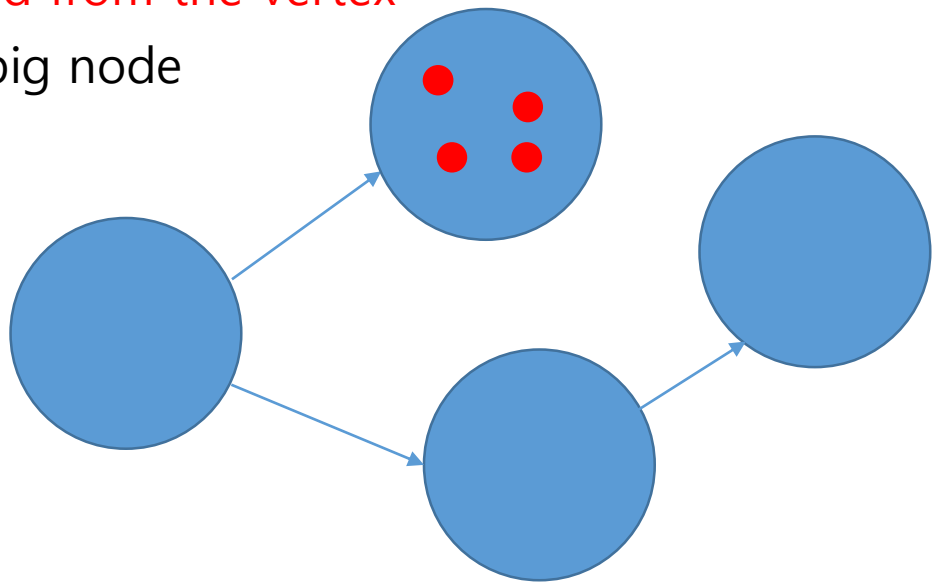


Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

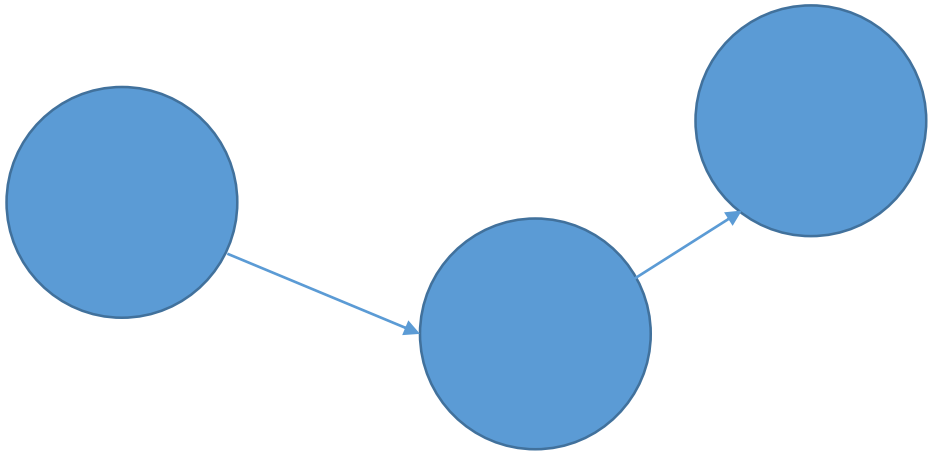


Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

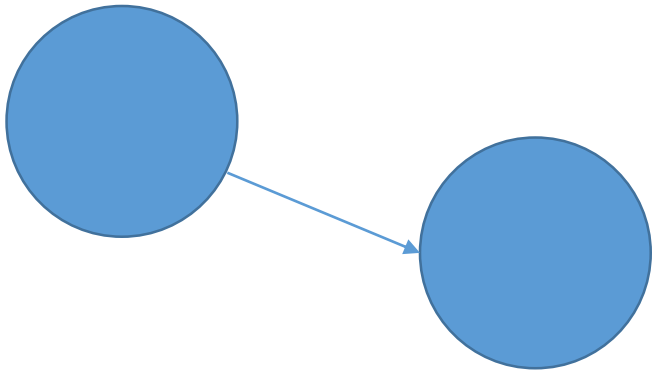


Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

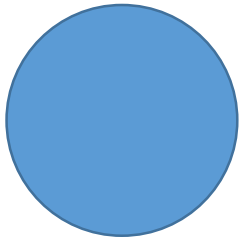


Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it



Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

  Is it correct ?

Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. Find a vertex within the leaf big node
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

  Is it correct ? Sure ☺

Strongly Connected Component

- Problem

  Therefore, we can derive a beautiful algorithm !

  1. **Find a vertex** within **the leaf big node**
  2. Traverse started from the vertex
  3. Remove that big node
  4. Repeat it

  However, how can we find a vertex in step 1 ?

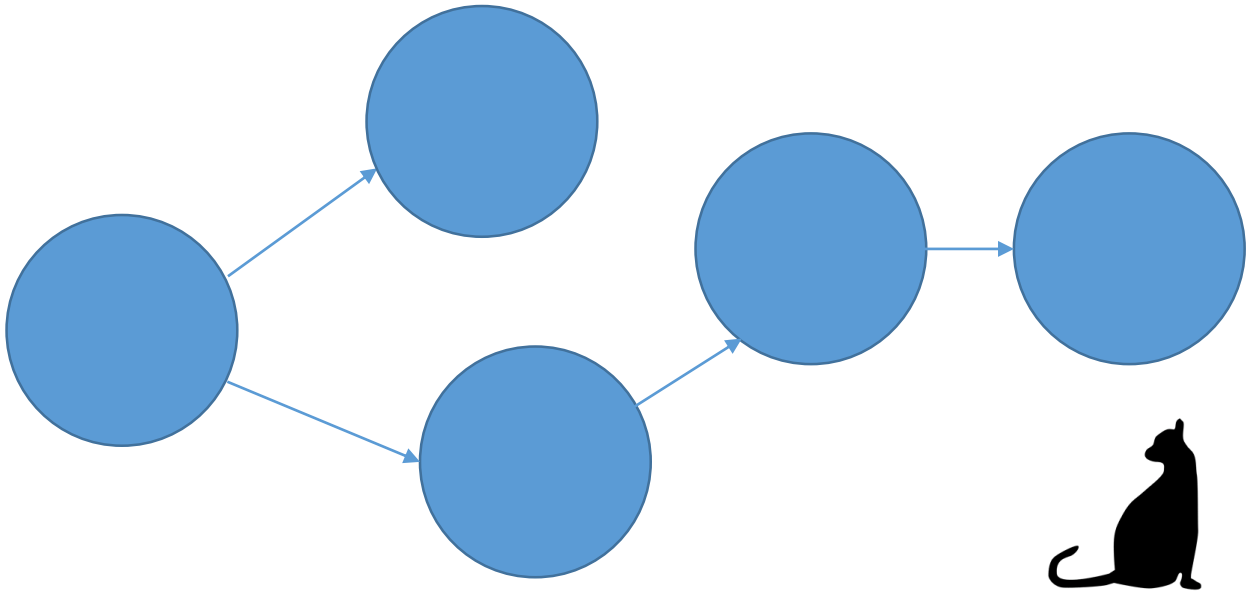
  We don't know whether a vertex is contained in the leaf big node or not

  Actually, it is impossible

Strongly Connected Component

- Problem

  OTL ...

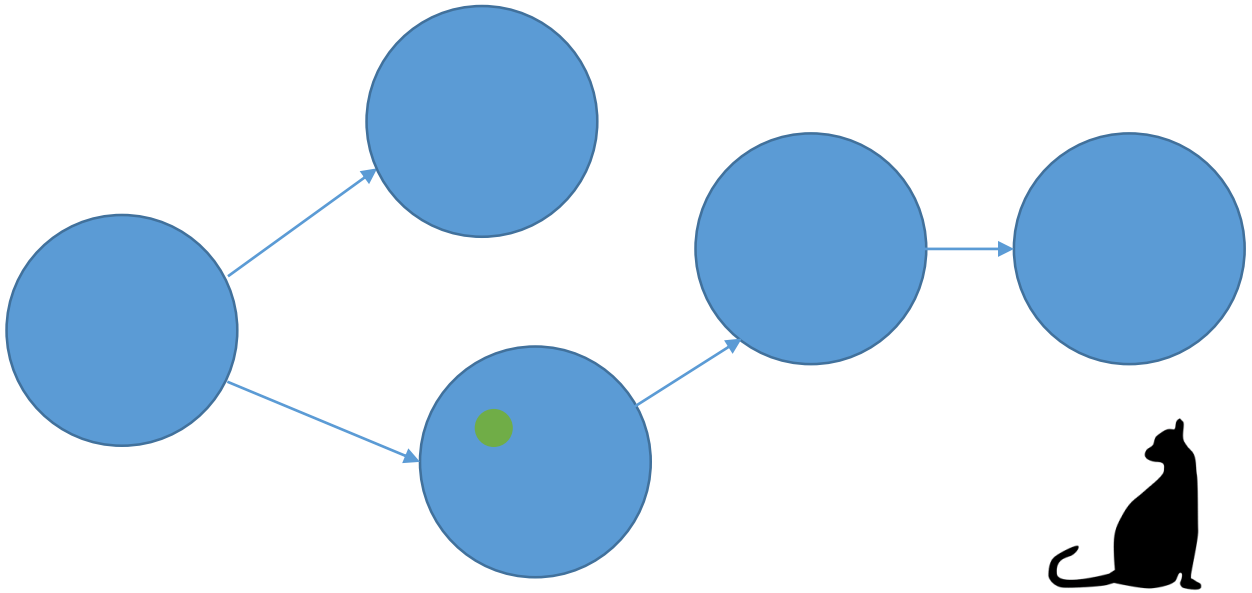


Strongly Connected Component

- Problem

  OTL …

  We can choose a vertex, but we don't know whether it is contained in the leaf big node

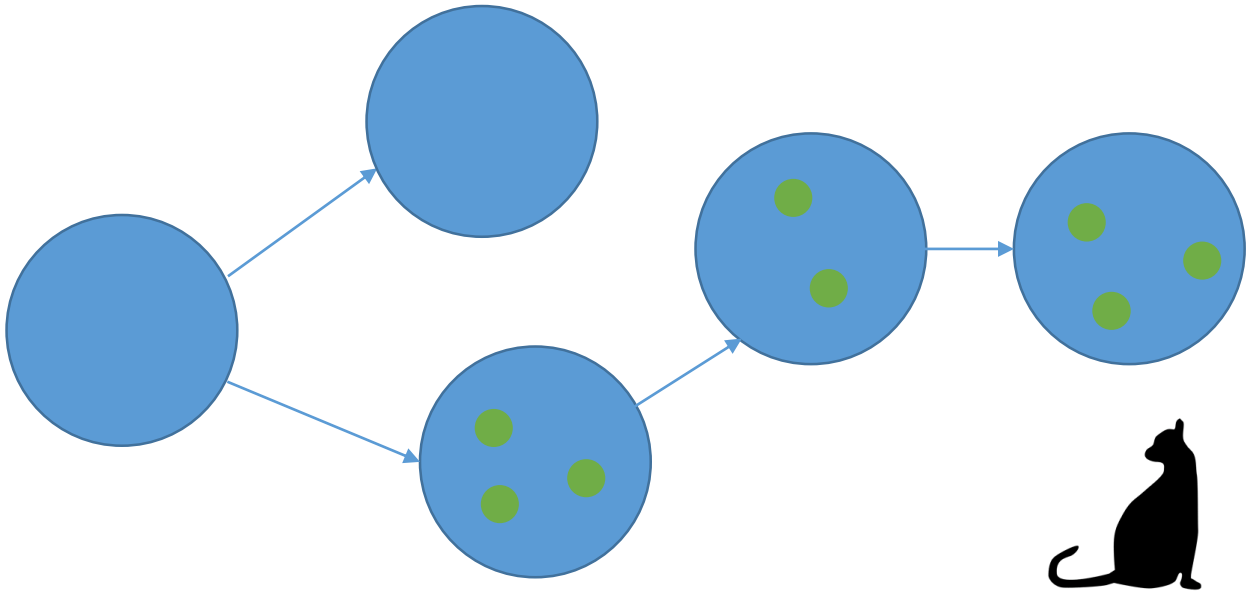


Strongly Connected Component

- Problem

  OTL ...

  Then, just traverse it ! Then it will traverse all the big node reached from the vertex

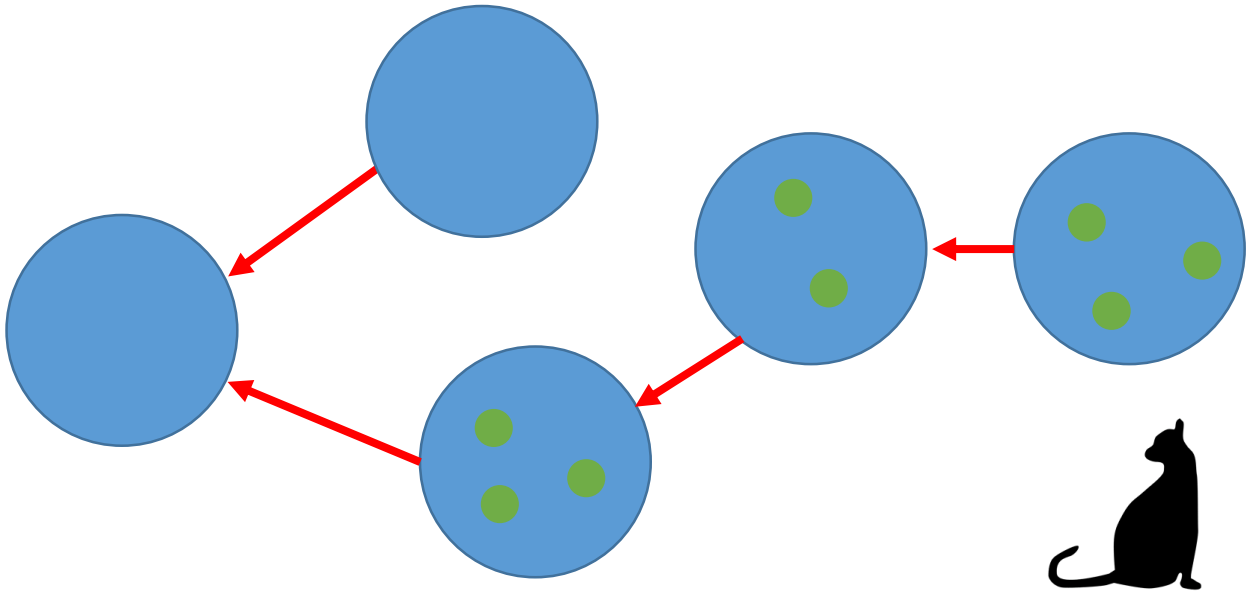


Strongly Connected Component

- Problem

  OTL ...

  And reverse the whole graph !



Strongly Connected Component

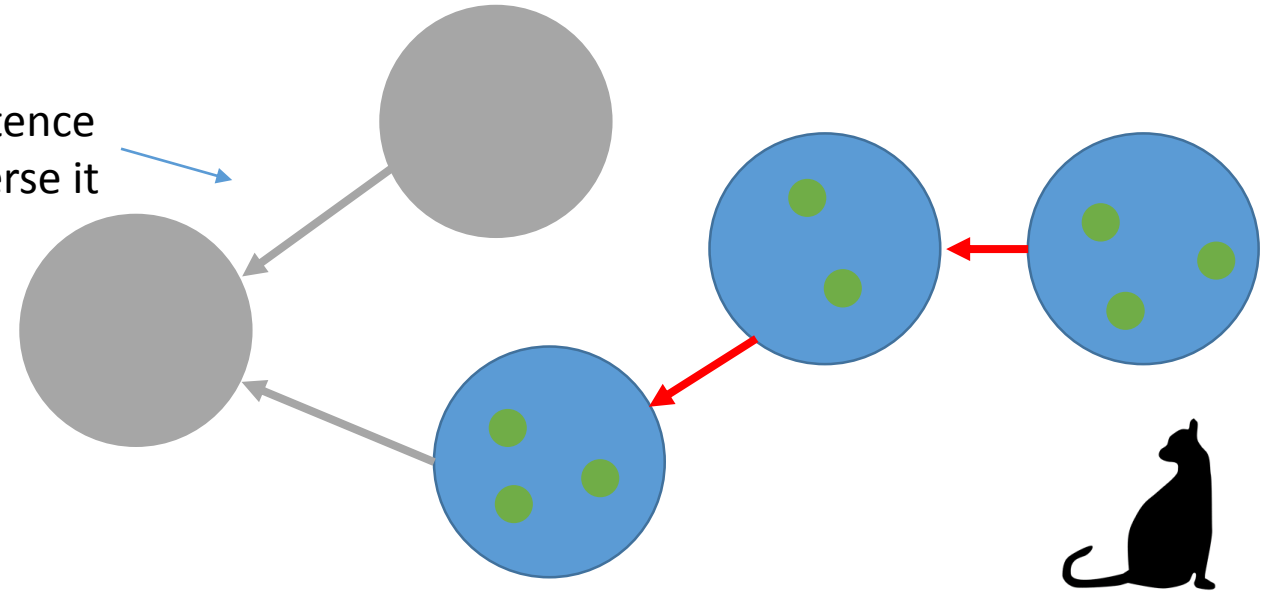
- Problem

  OTL …

  And reverse the whole graph !

  Then our start vertex becomes the vertex contained in the leaf node !

We don’t know its existence
because we don’t traverse it



Strongly Connected Component

- Problem

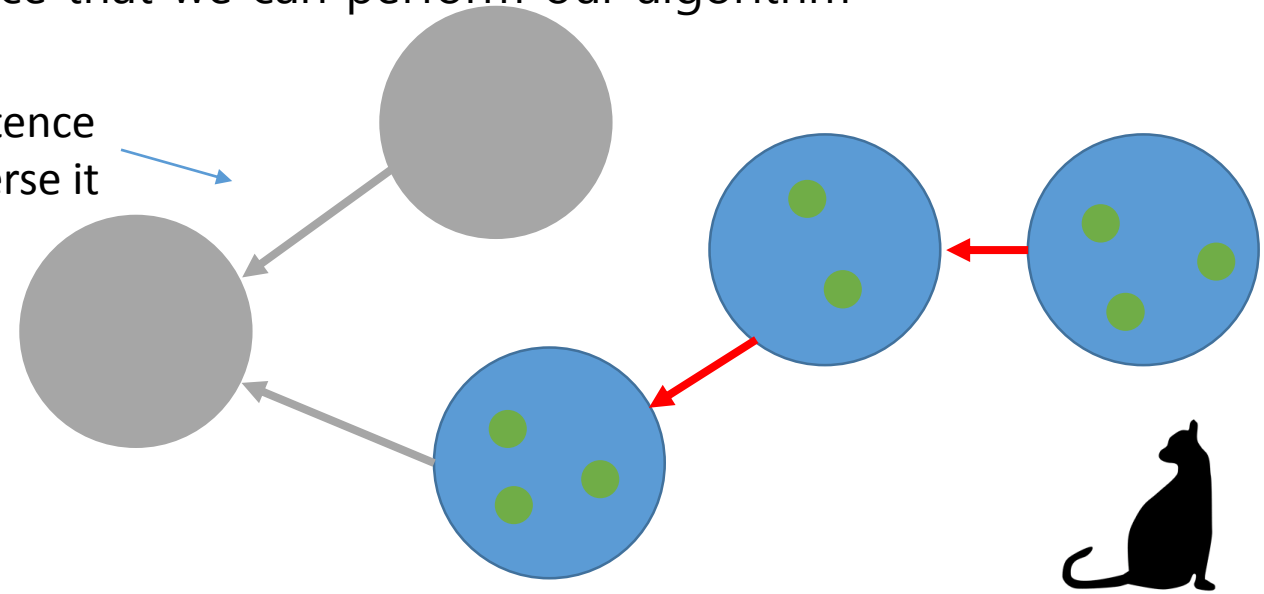
  OTL …

  And reverse the whole graph !

  Then our start vertex becomes the vertex contained in the leaf node !

  It is strong evidence that we can perform our algorithm

We don’t know its existence
because we don’t traverse it

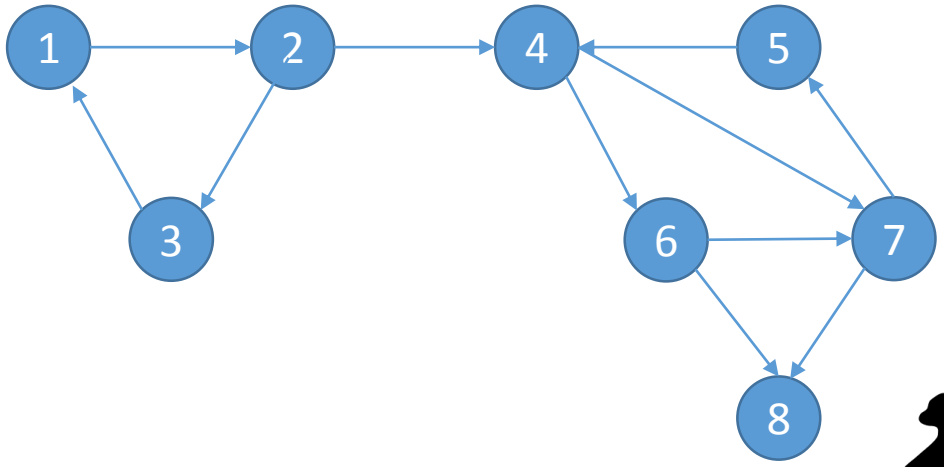


Strongly Connected Component

- Overall procedure
  1. Choose a start vertex
  2. Perform DFS, and write the **exit time** for all the vertices
  3. After doing DFS, reverse whole graph
  4. Perform DFS or BFS starting from a vertex whose exit time is the largest
  5. All the vertices gathering in step 4 performs a SCC

Strongly Connected Component

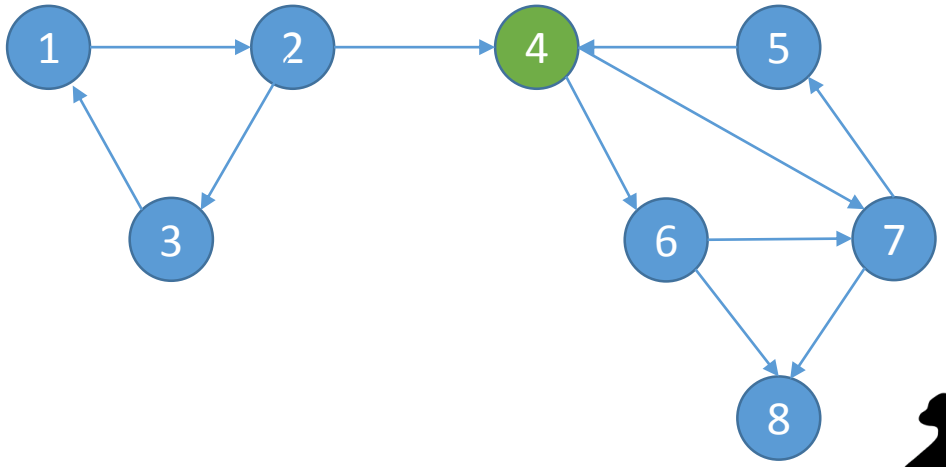
- Overall procedure



Strongly Connected Component

- Overall procedure

  Pick a start vertex

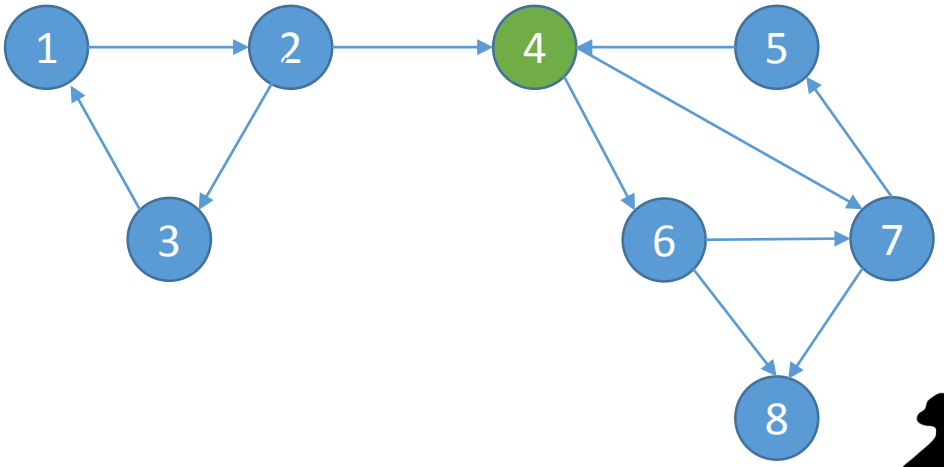


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

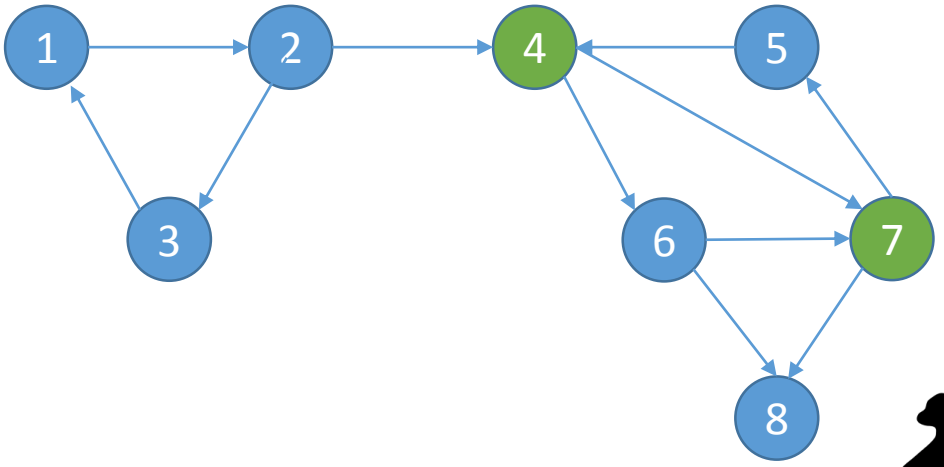


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

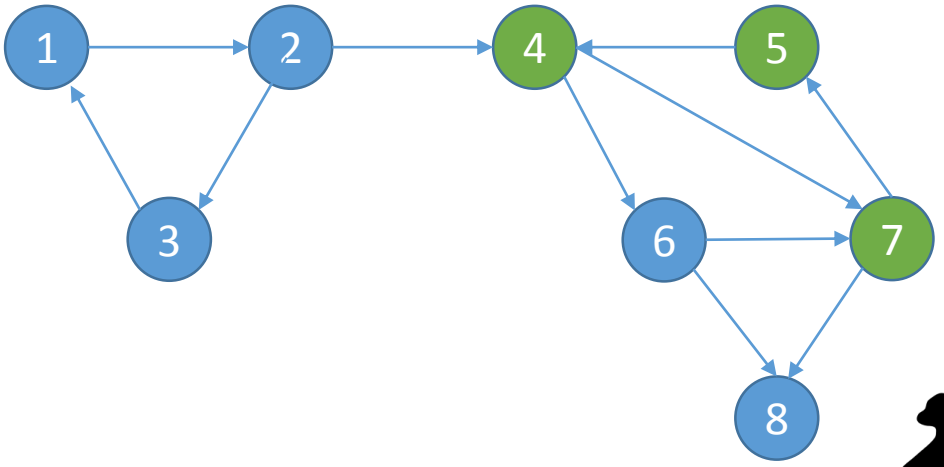


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

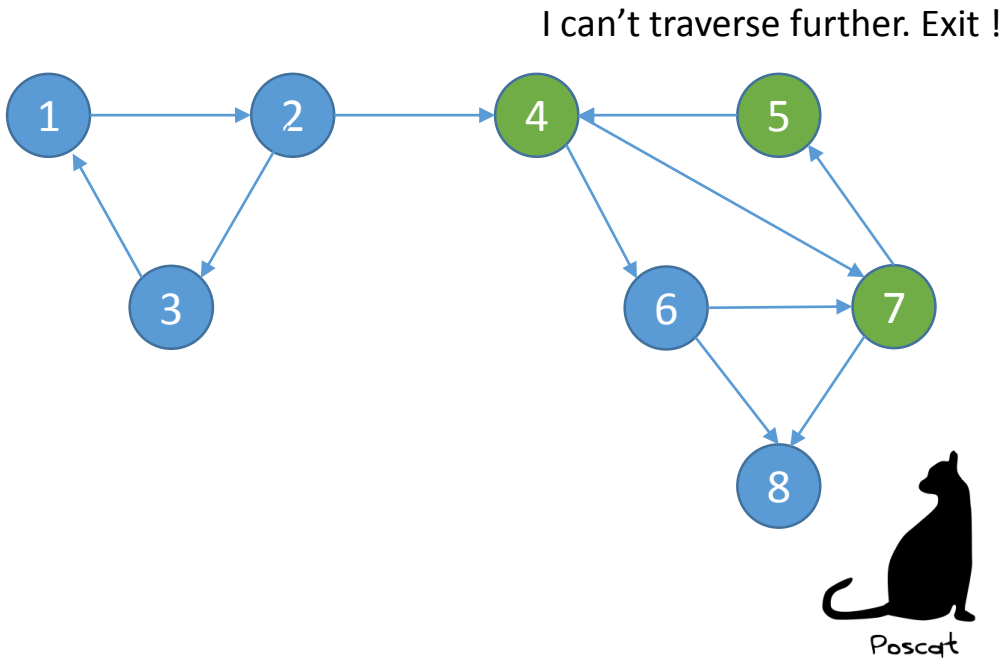


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

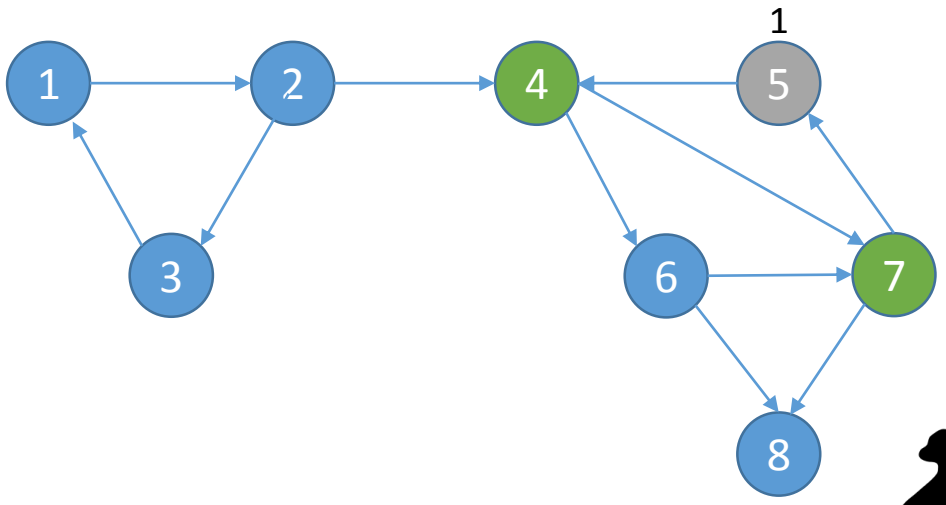


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

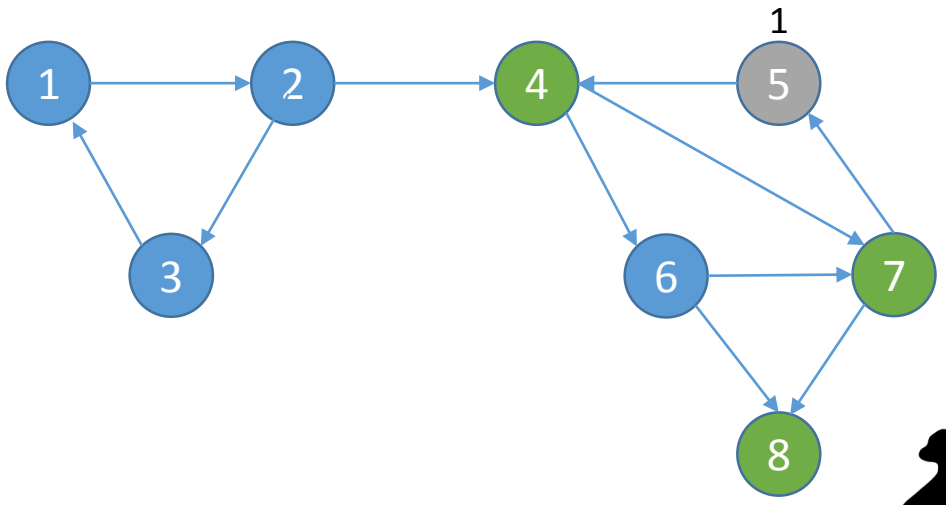


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

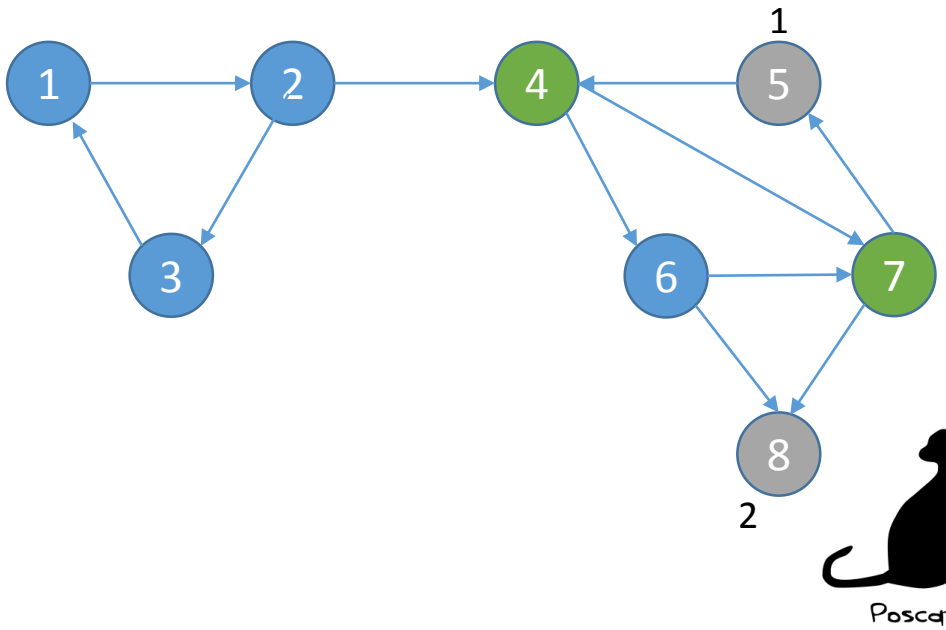


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

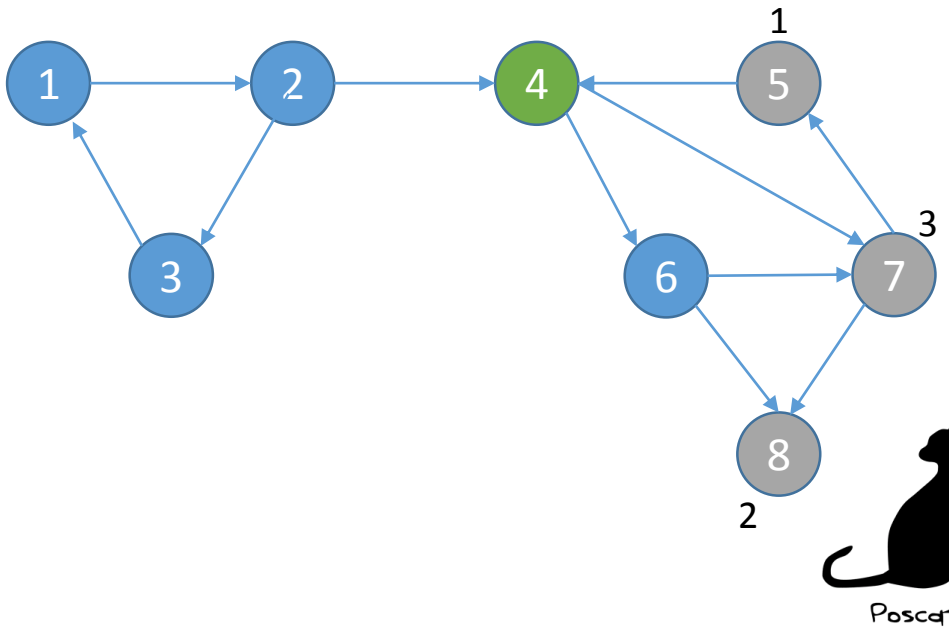


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

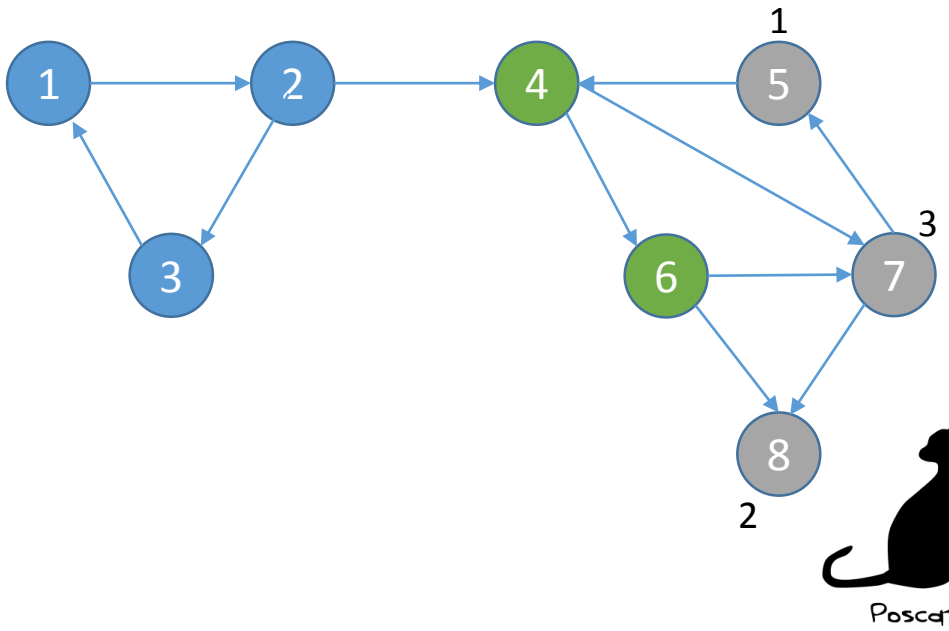


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

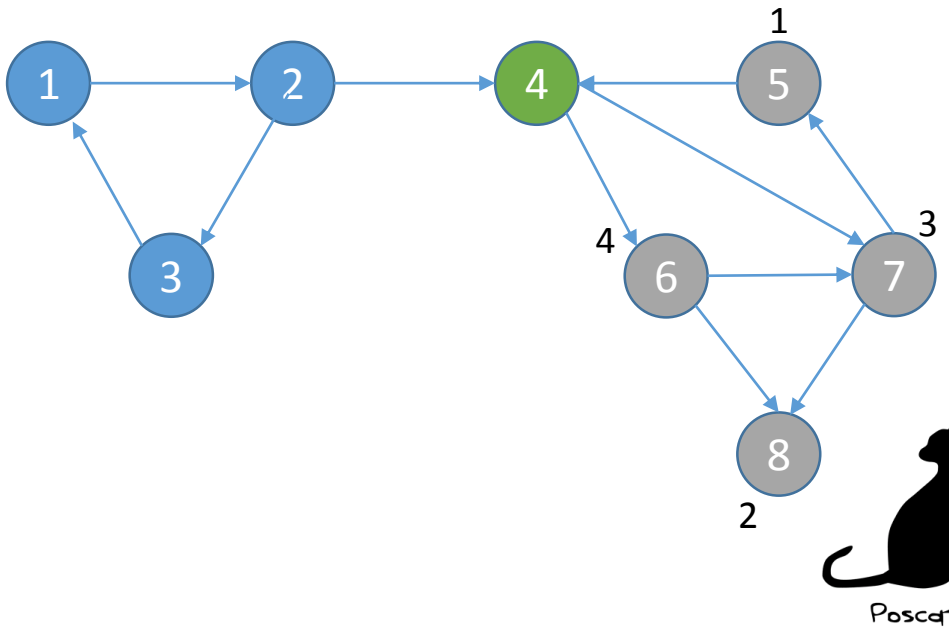


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

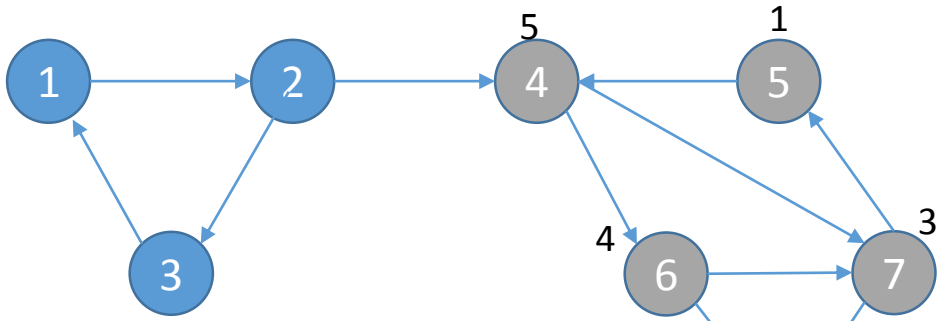


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

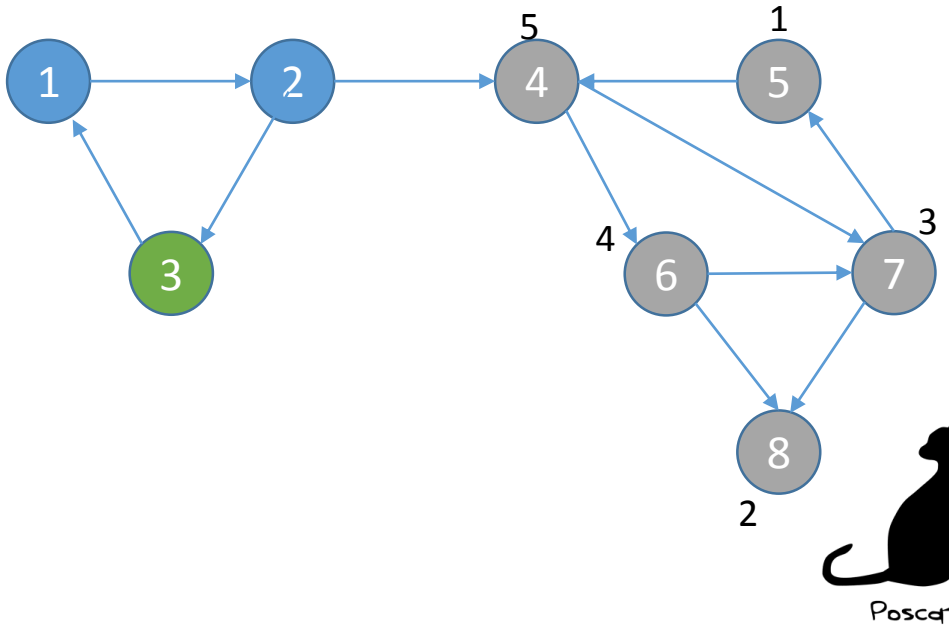


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

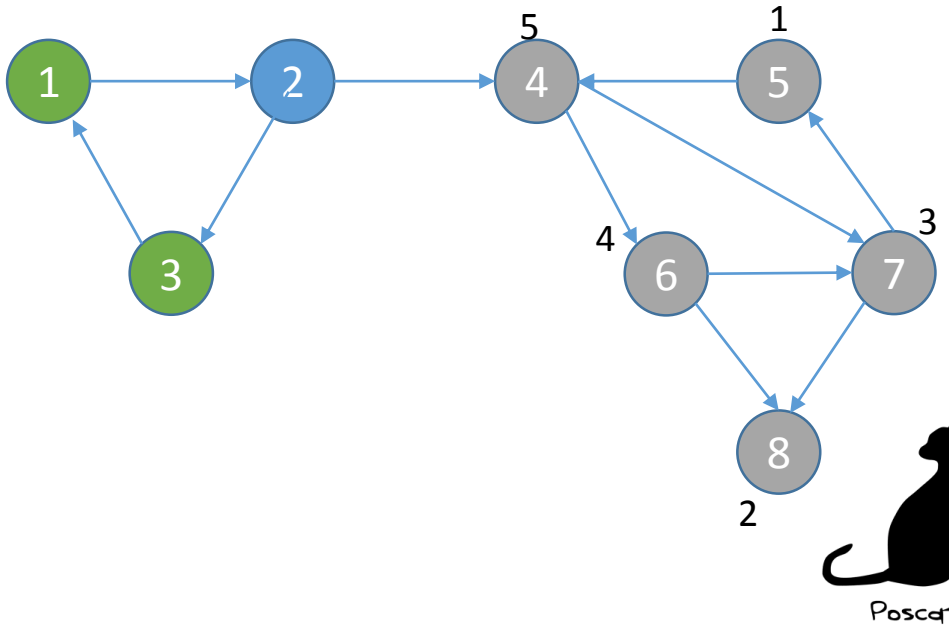


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

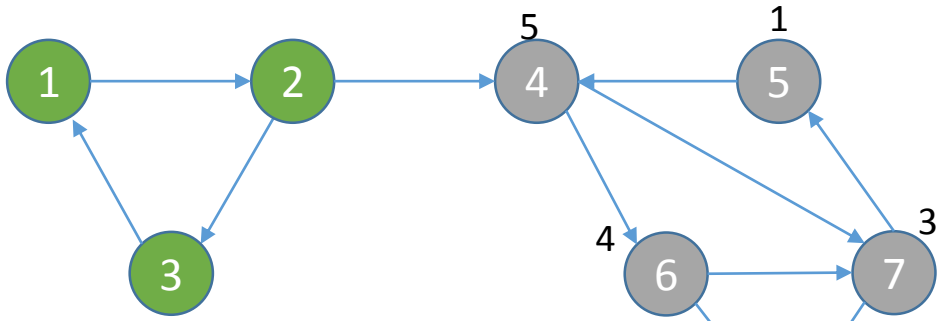


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

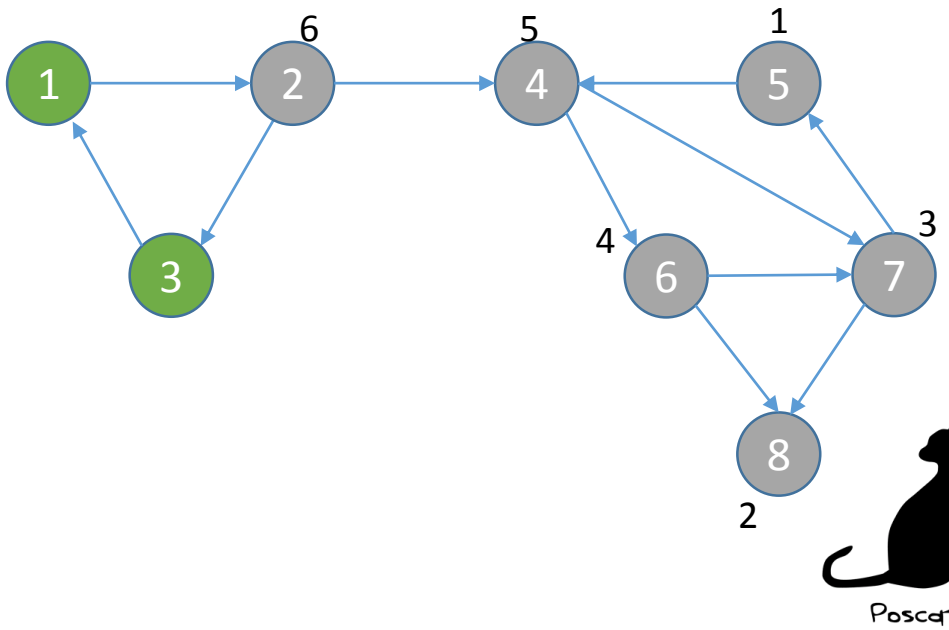


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

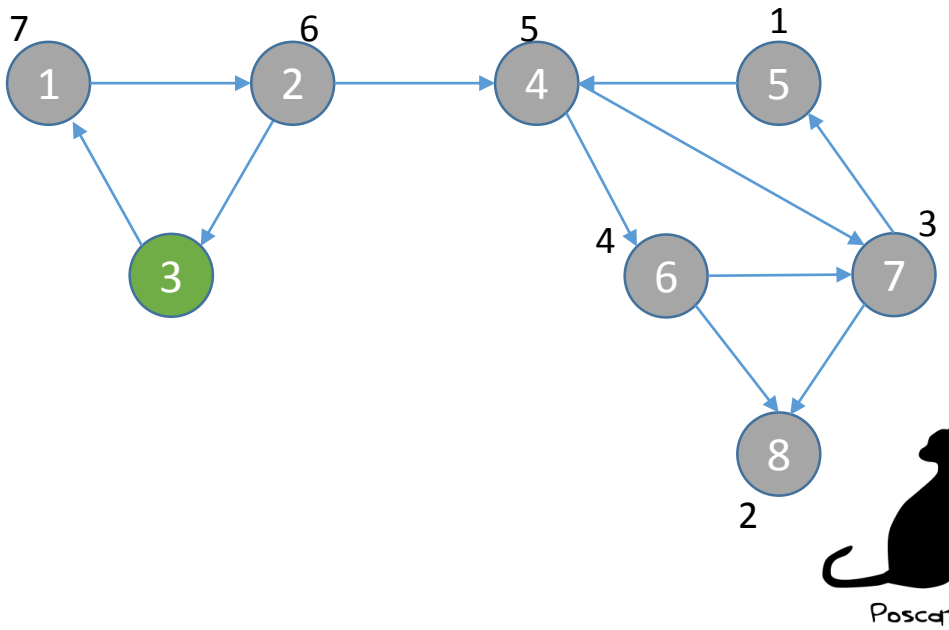


Strongly Connected Component

- Overall procedure

  Pick a start vertex

  Perform DFS and write the exit time

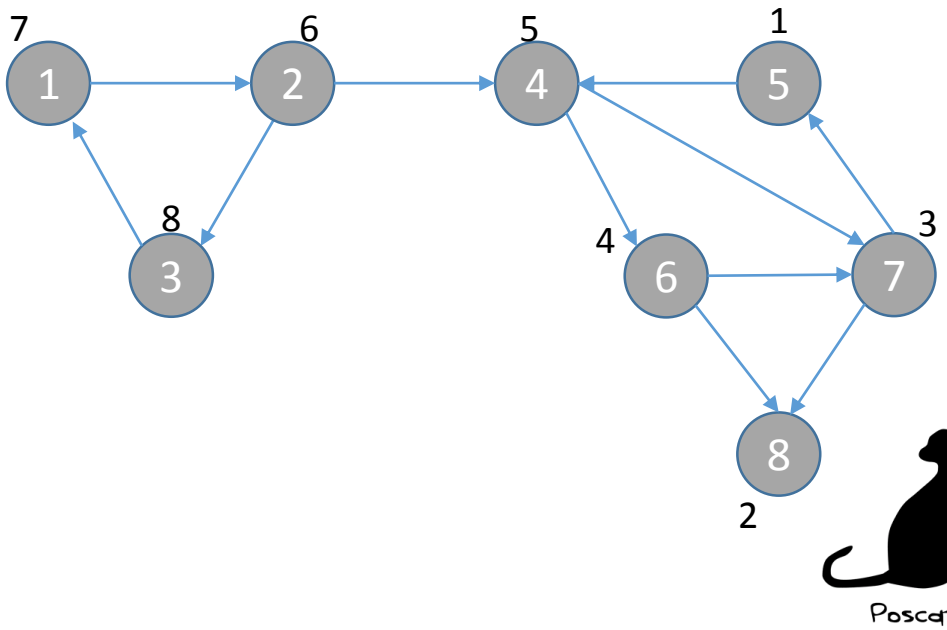


Strongly Connected Component

- Overall procedure

  Pick a start vertex

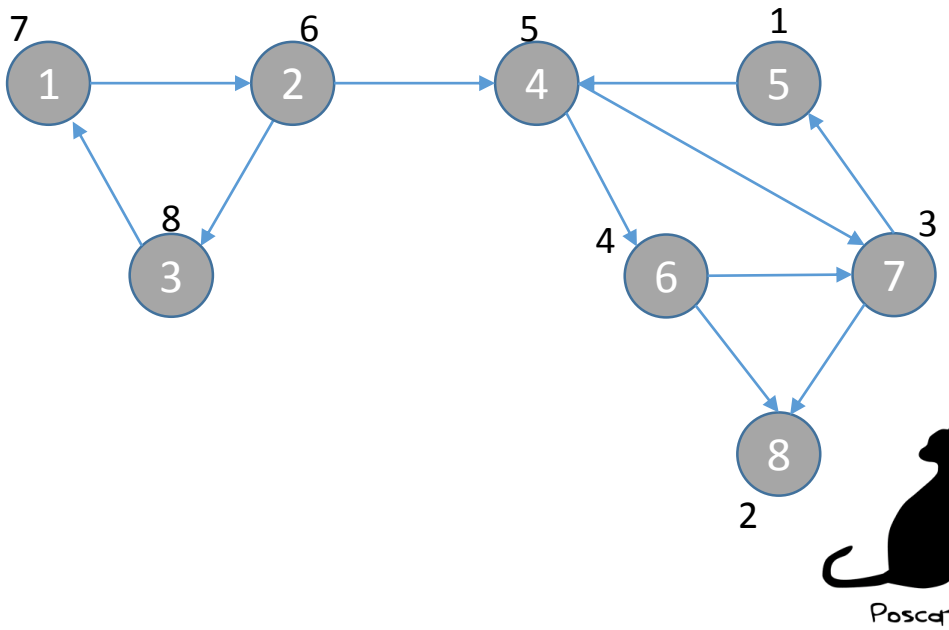
  Perform DFS and write the exit time



Strongly Connected Component

- Overall procedure

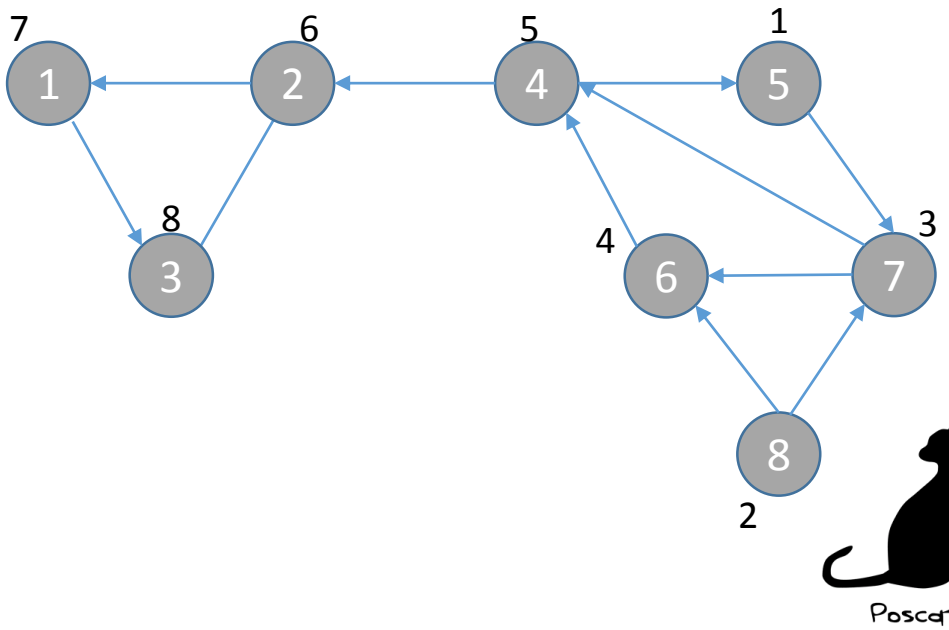
  After writing all the exit time, reverse whole graph



Strongly Connected Component

- Overall procedure

  After writing all the exit time, reverse whole graph

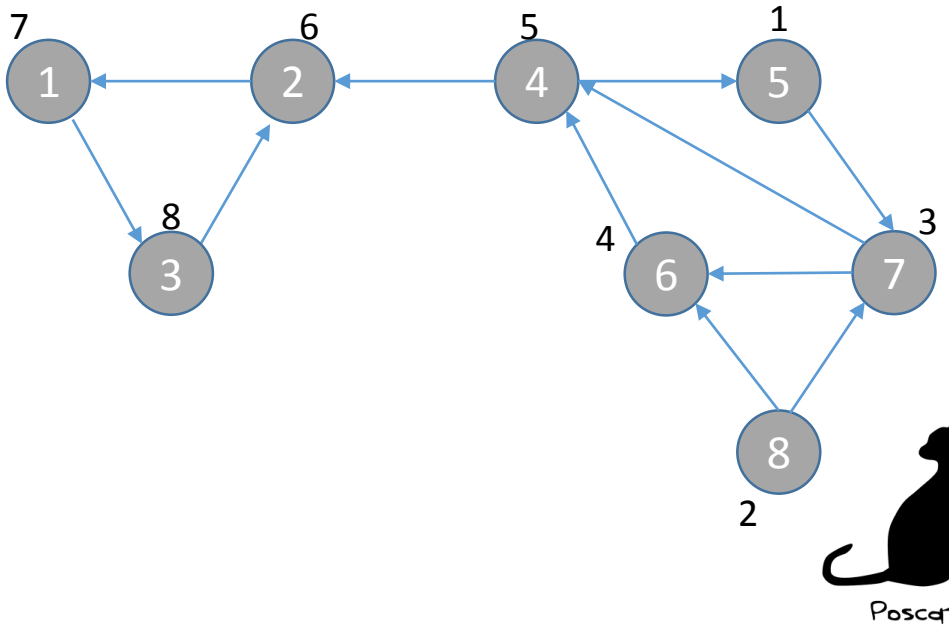


Strongly Connected Component

- Overall procedure

  Perform DFS or BFS started from a vertex with highest exit time

  All the vertices reached from the start vertex makes a SCC

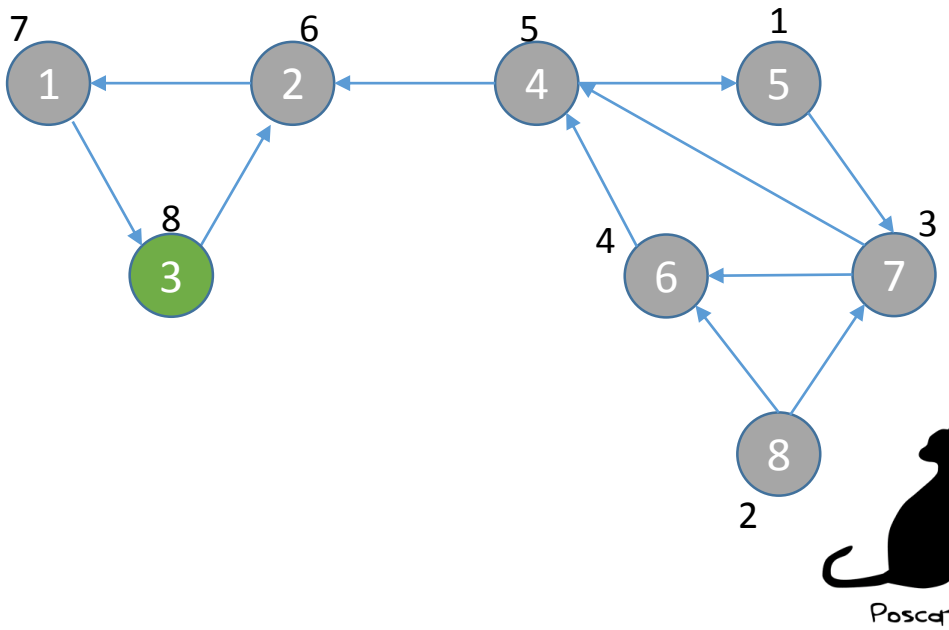


Strongly Connected Component

- Overall procedure

  Perform DFS or BFS started from a vertex with highest exit time

  All the vertices reached from the start vertex makes a SCC

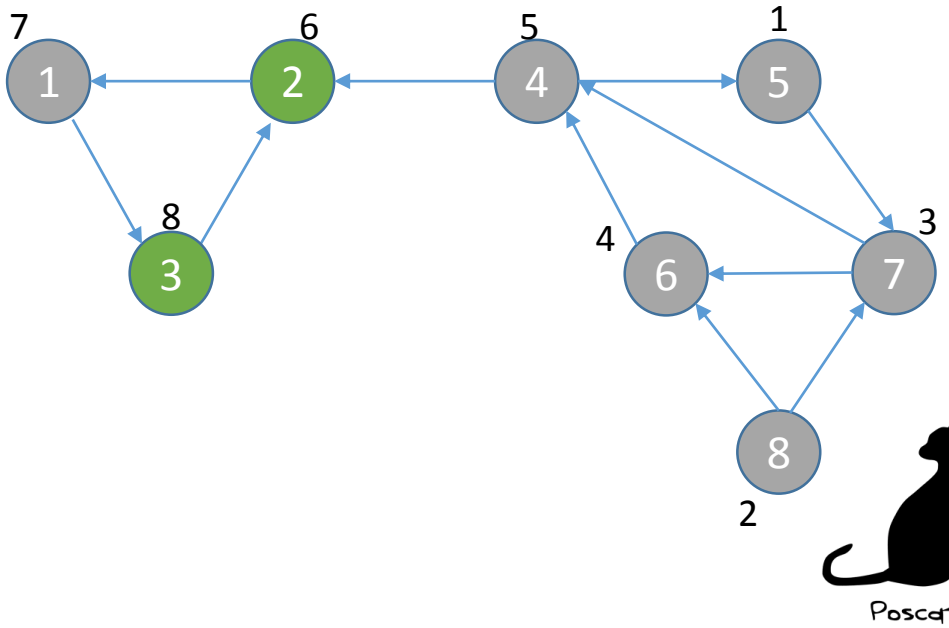


Strongly Connected Component

- Overall procedure

  Perform DFS or BFS started from a vertex with highest exit time

  All the vertices reached from the start vertex makes a SCC

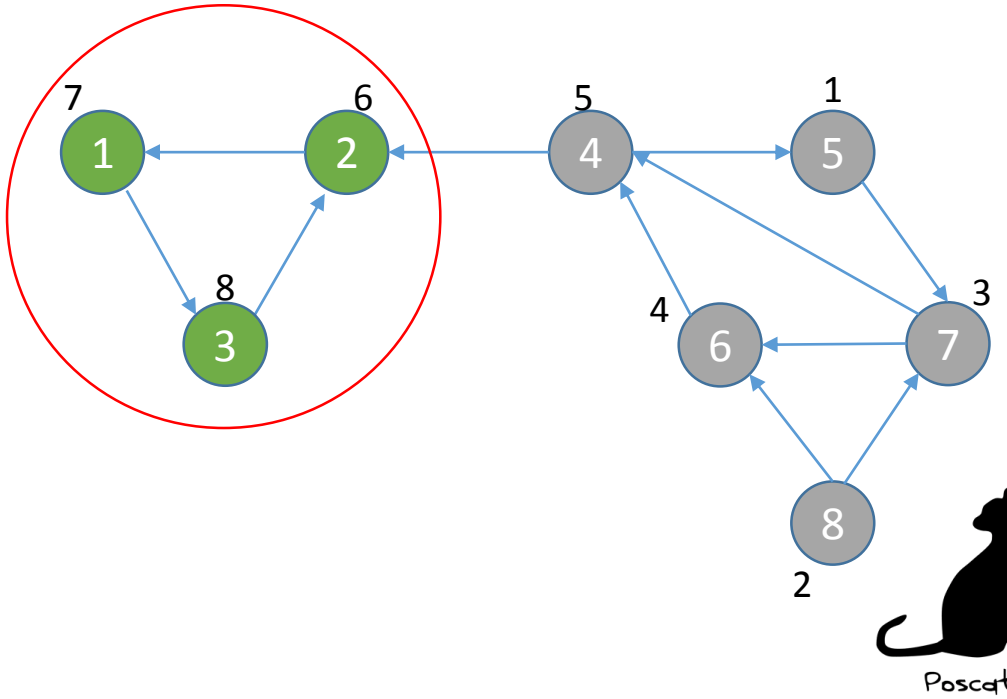


Strongly Connected Component

- Overall procedure

  Perform DFS or BFS started from a vertex with highest exit time

  All the vertices reached from the start vertex makes a SCC

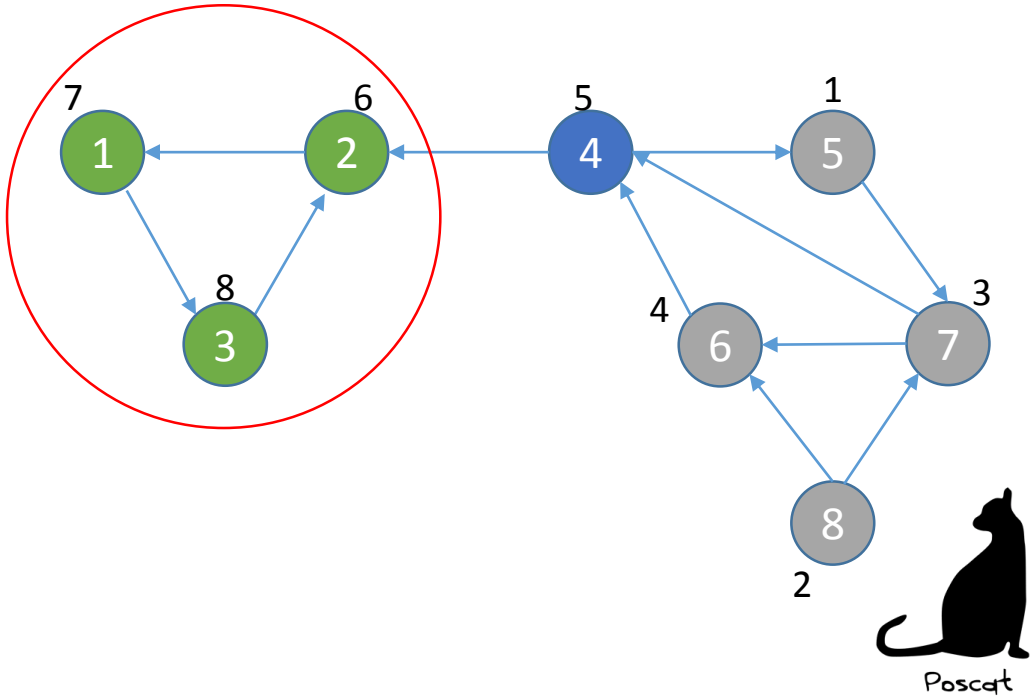


Strongly Connected Component

- Overall procedure

  Perform DFS or BFS started from a vertex with highest exit time

  All the vertices reached from the start vertex makes a SCC

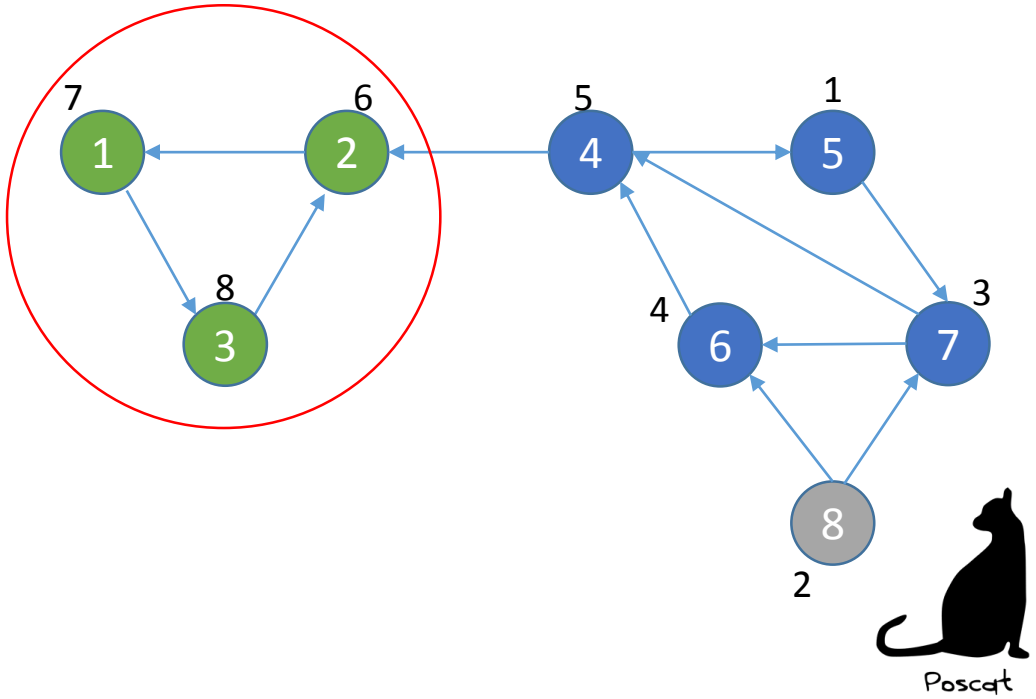


Strongly Connected Component

- Overall procedure

  Perform DFS or BFS started from a vertex with highest exit time

  All the vertices reached from the start vertex makes a SCC

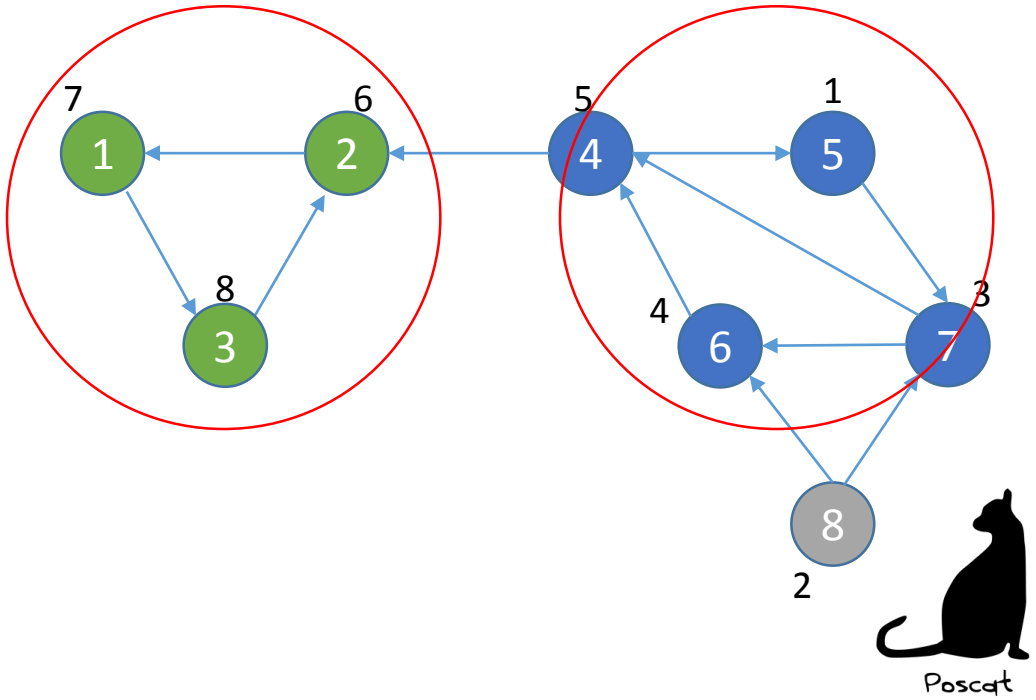


Strongly Connected Component

- Overall procedure

  Perform DFS or BFS started from a vertex with highest exit time

  All the vertices reached from the start vertex makes a SCC

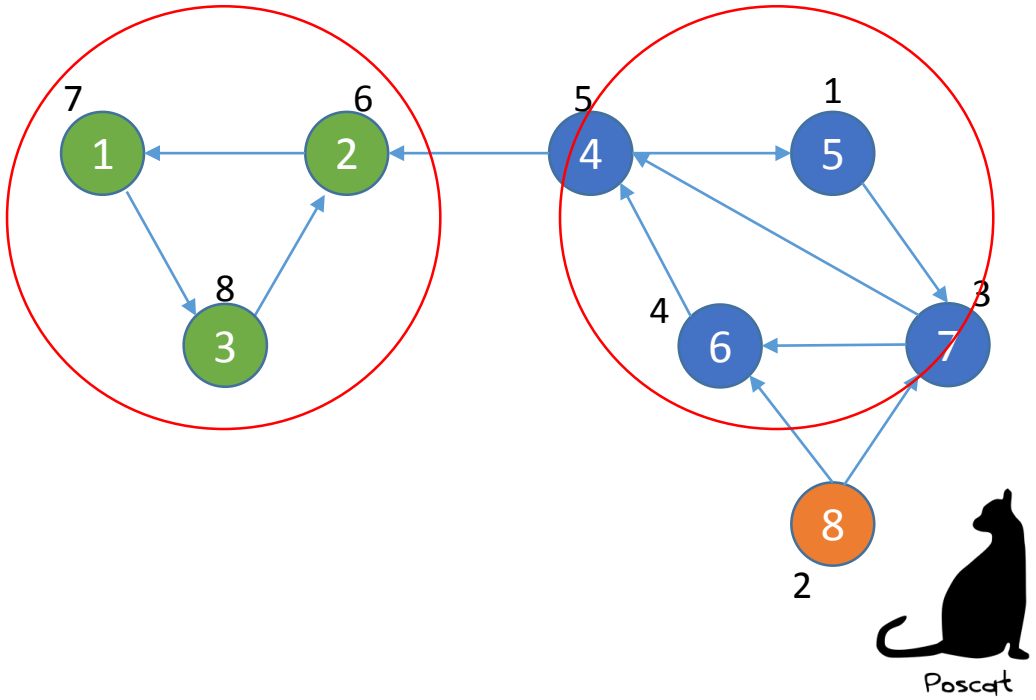


Strongly Connected Component

- Overall procedure

  Perform DFS or BFS started from a vertex with highest exit time

  All the vertices reached from the start vertex makes a SCC

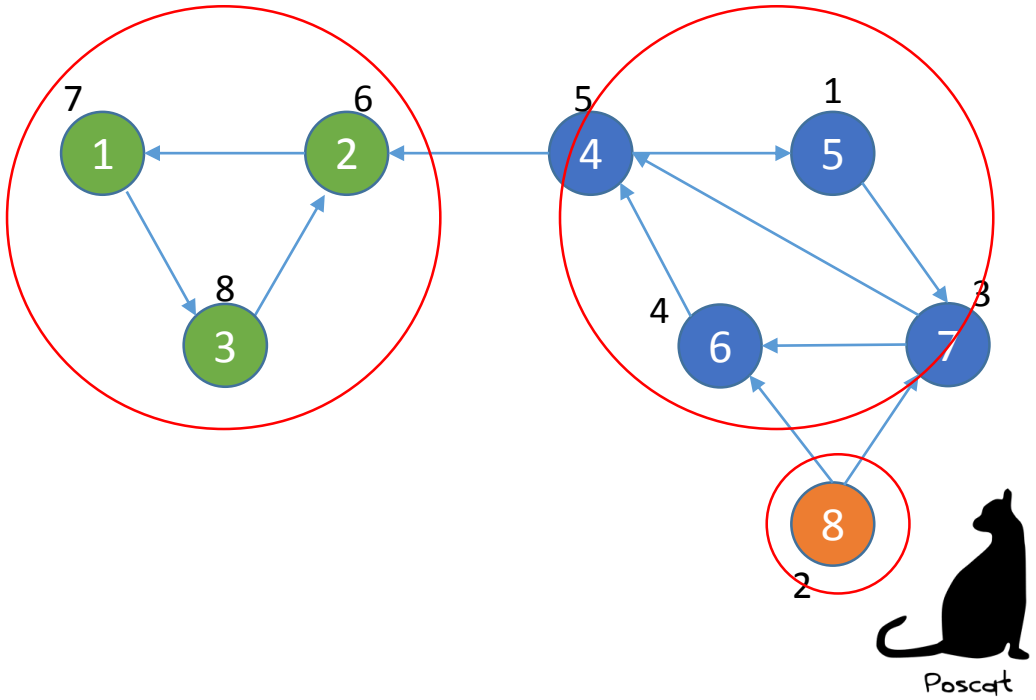


Strongly Connected Component

- Overall procedure

  Perform DFS or BFS started from a vertex with highest exit time

  All the vertices reached from the start vertex makes a SCC

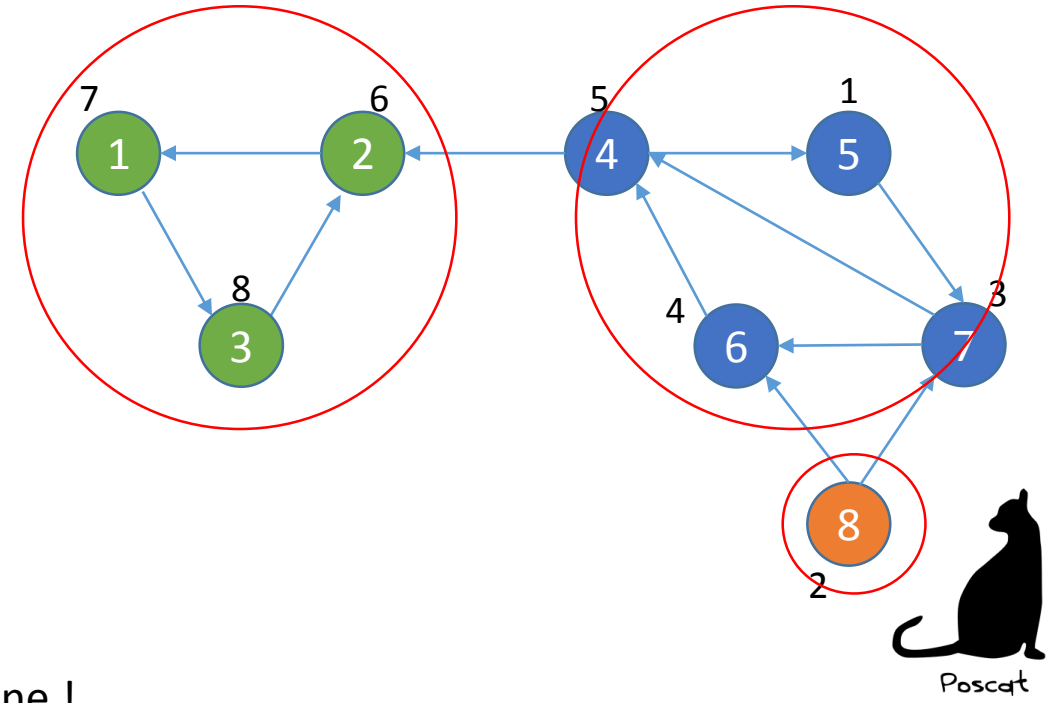


Strongly Connected Component

- Overall procedure

  Perform DFS or BFS started from a vertex with highest exit time

  All the vertices reached from the start vertex makes a SCC



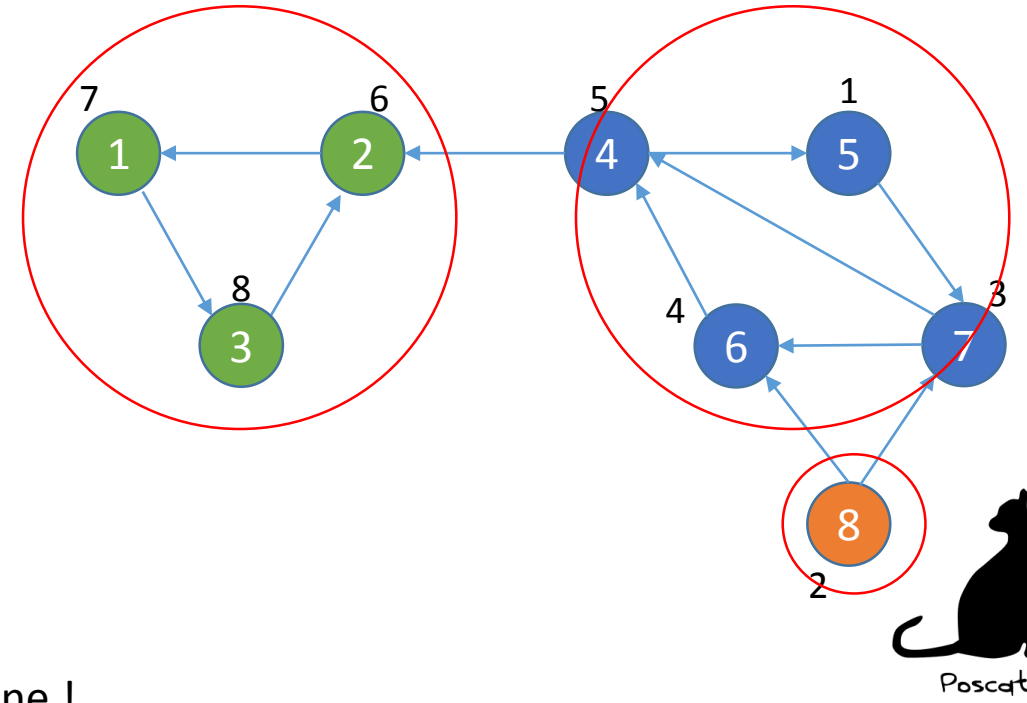
Done !

Strongly Connected Component

- Analysis

  It is correct ?

  How long does it takes ?



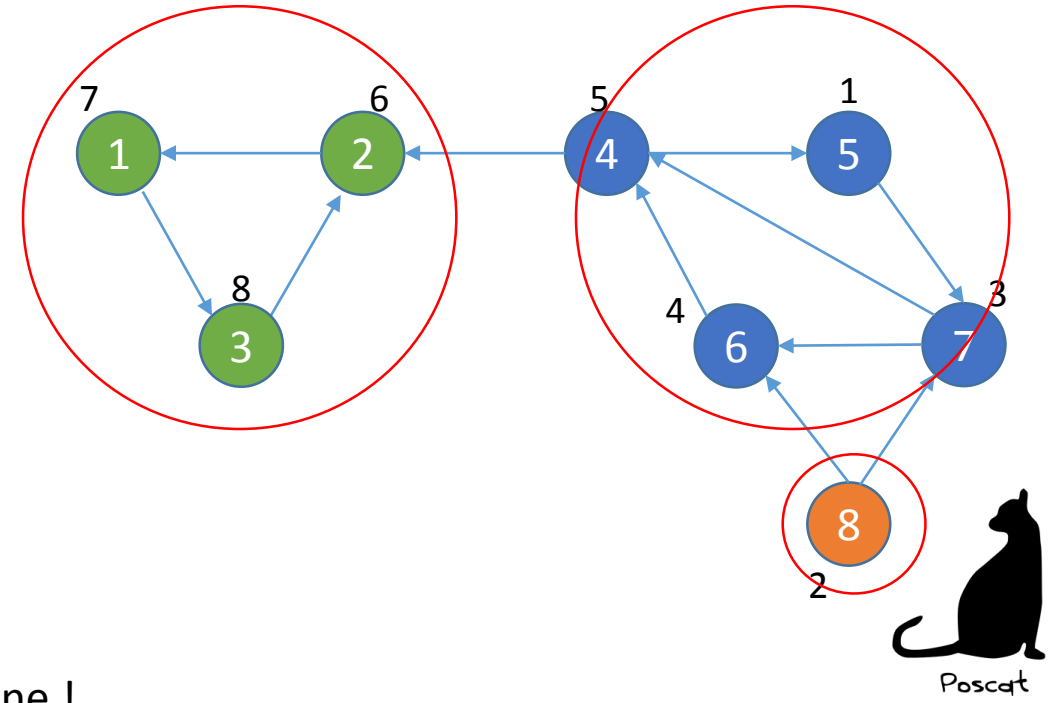
Done !

Strongly Connected Component

- Analysis

  It is correct ? We need some detail proof

  How long does it takes ? We need 2 DFS (or 1 DFS + 1 BFS)



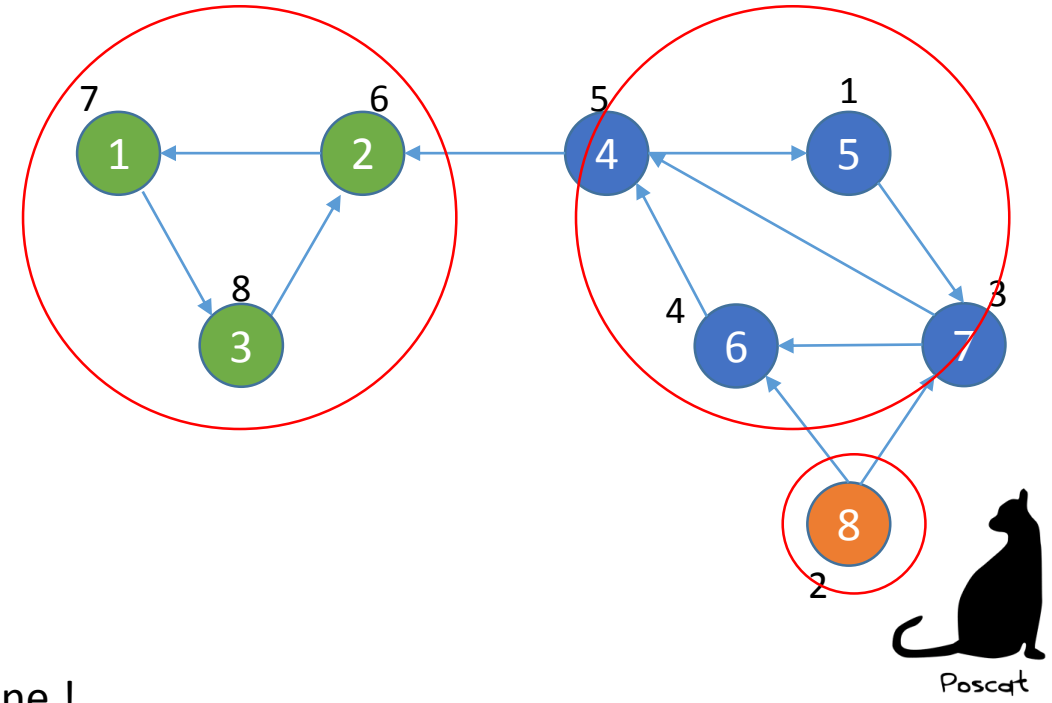
Done !

Strongly Connected Component

- Analysis

  It is correct ? We need some detail proof

  How long does it takes ? **$O(V + E)$**



Done !