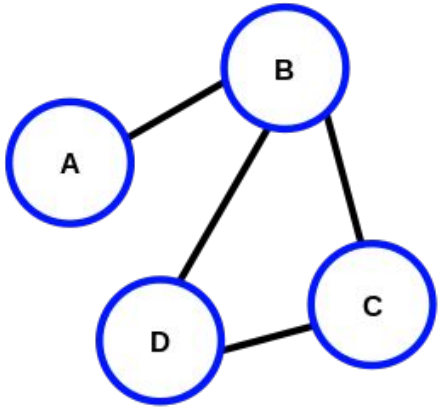
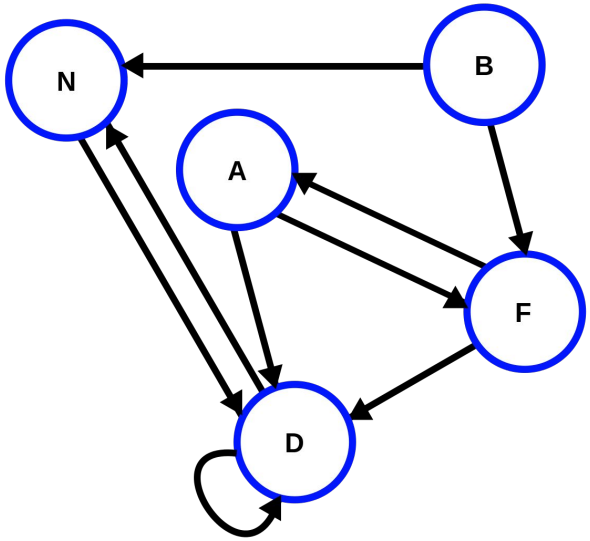
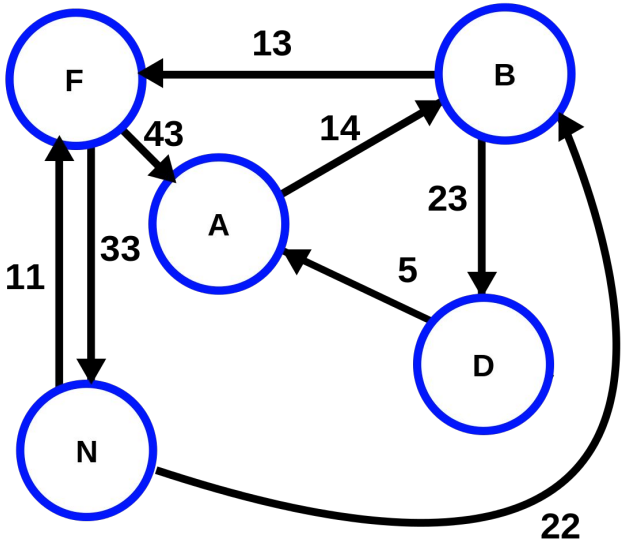
Graphs



Unweighted, undirected

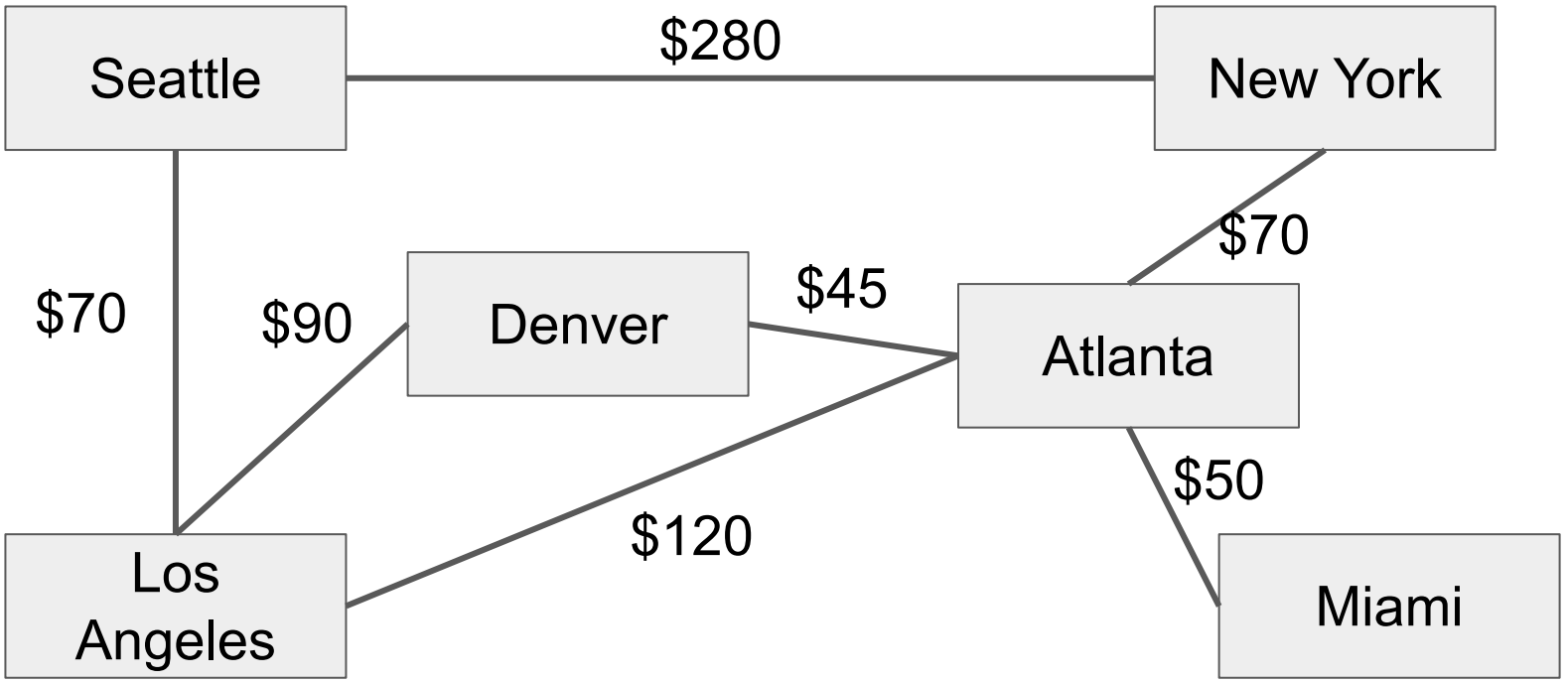


Unweighted, directed



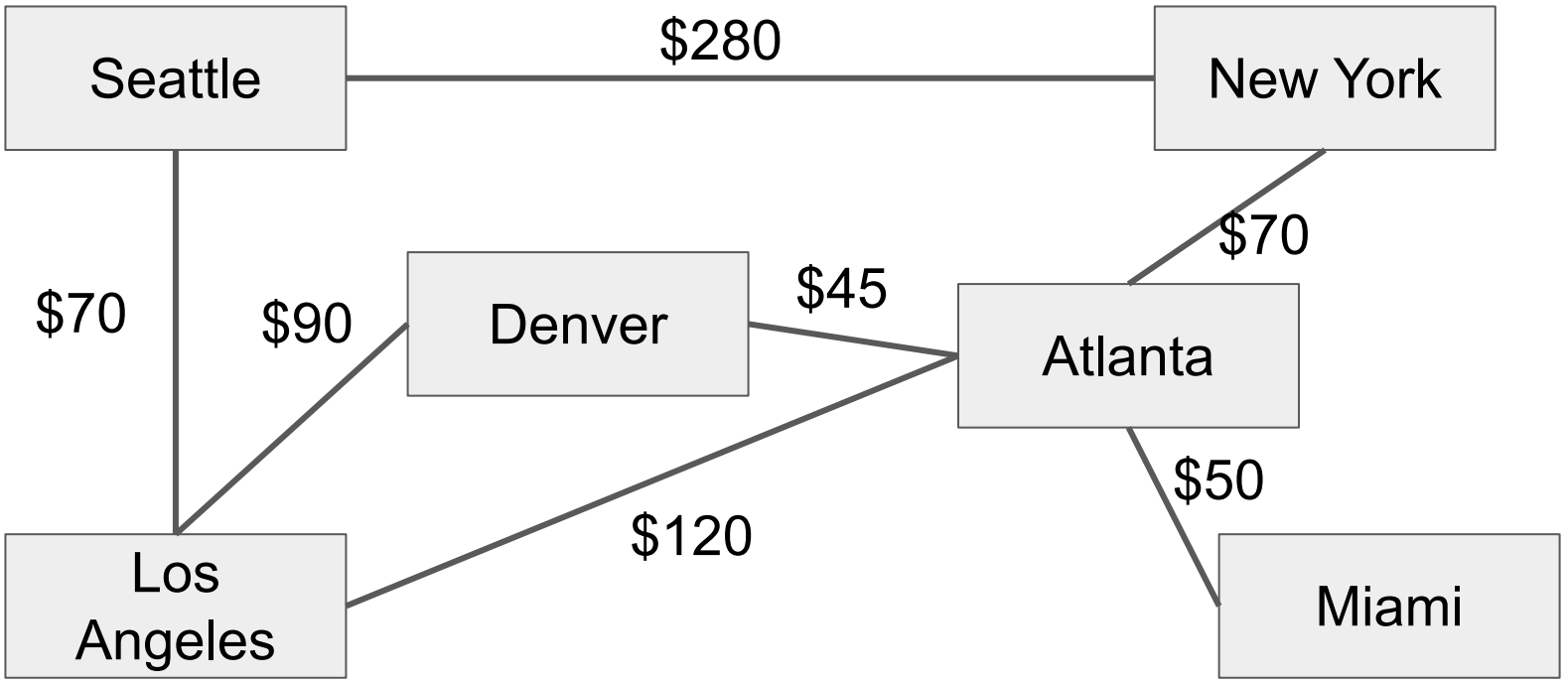
Weighted, directed

Graphs



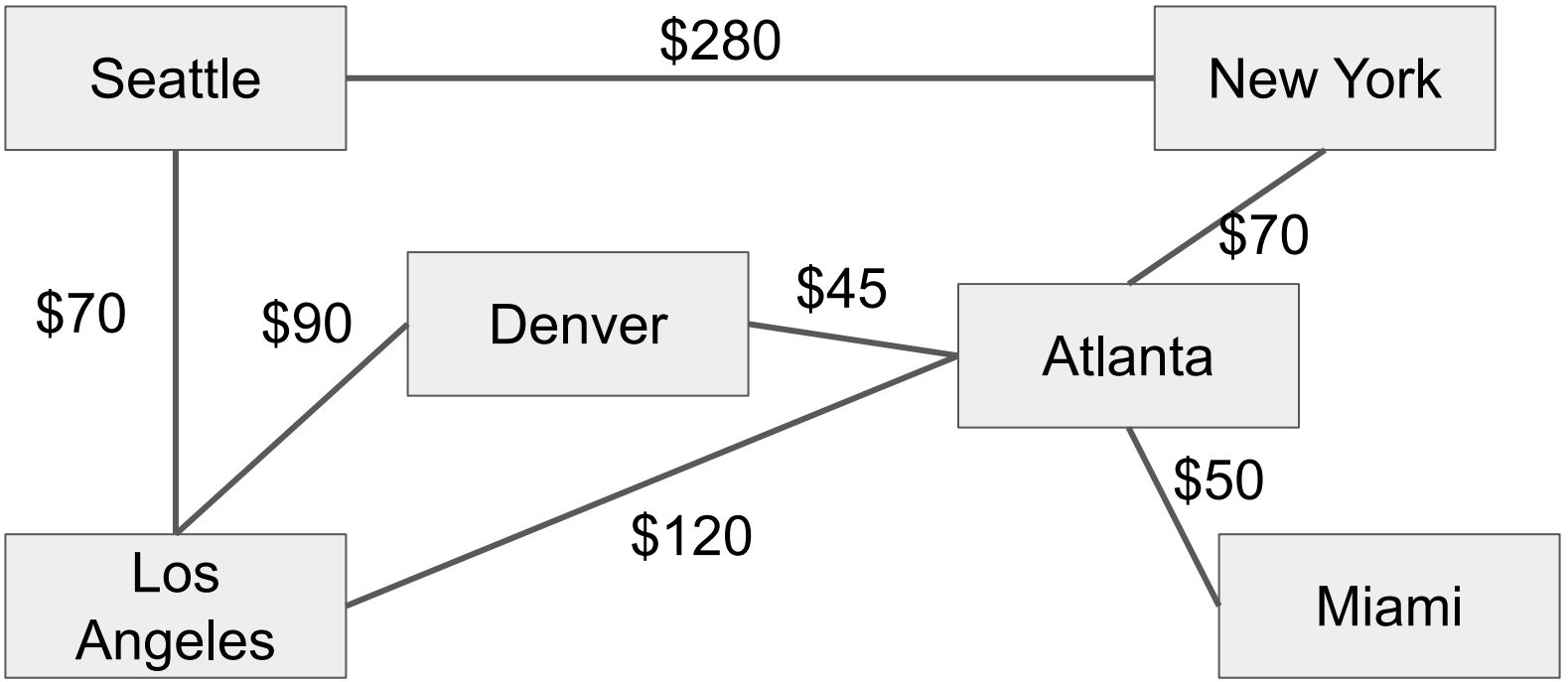
What type of a graph is this?

Graphs



Undirected, weighted

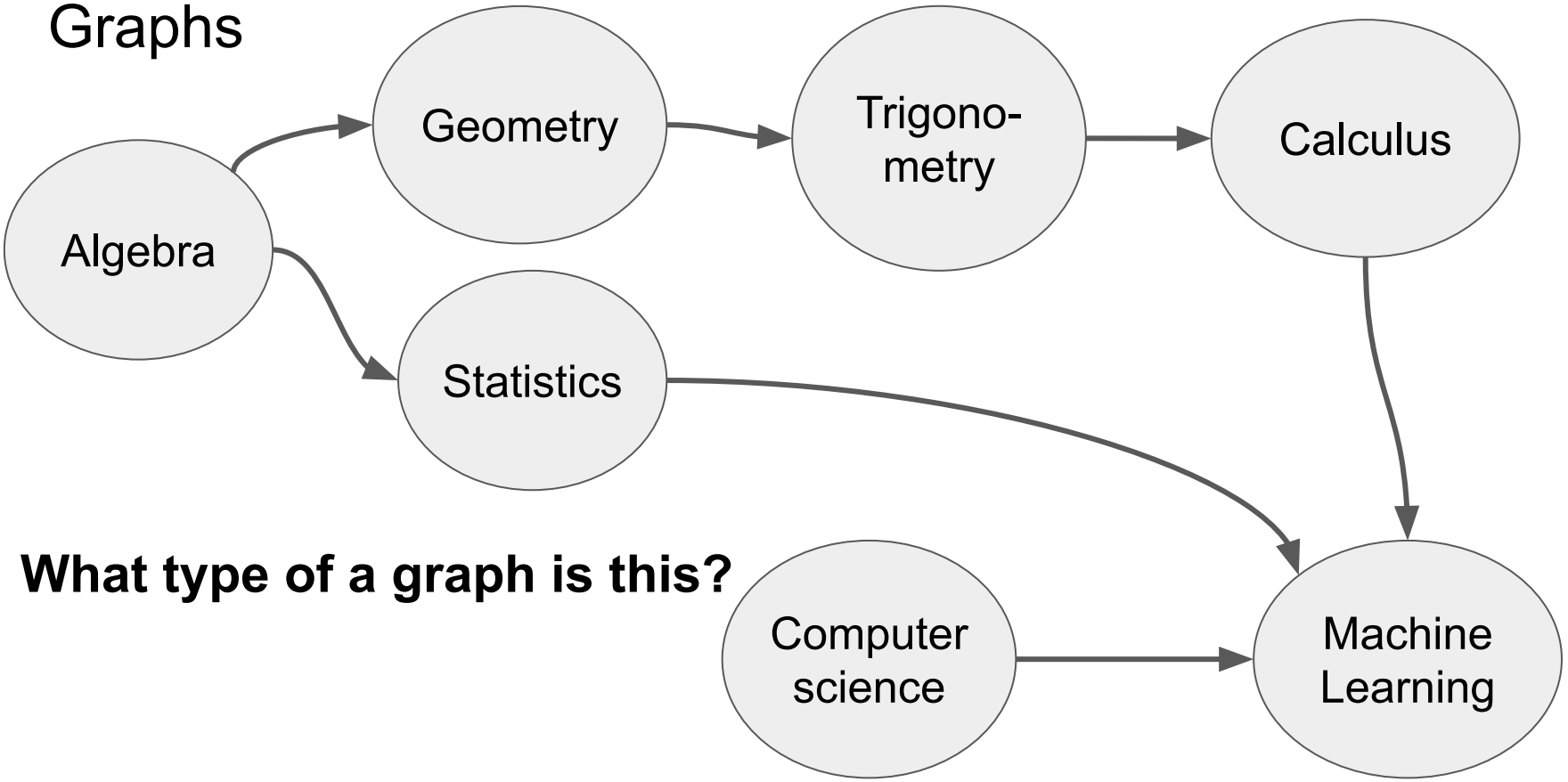
Graphs



Undirected, weighted

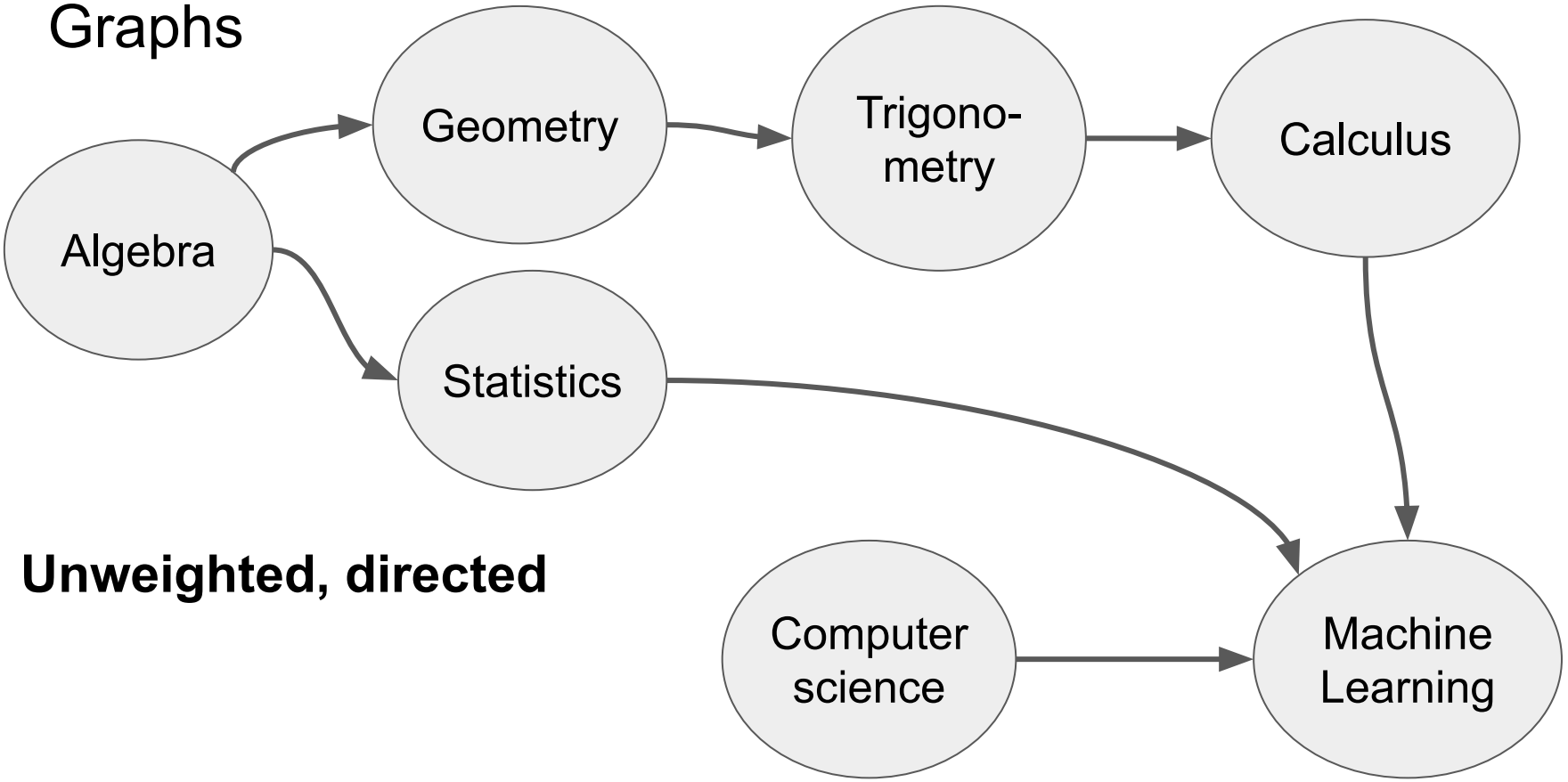
What is the cheapest way to get to Miami from LA?

Graphs



What type of a graph is this?

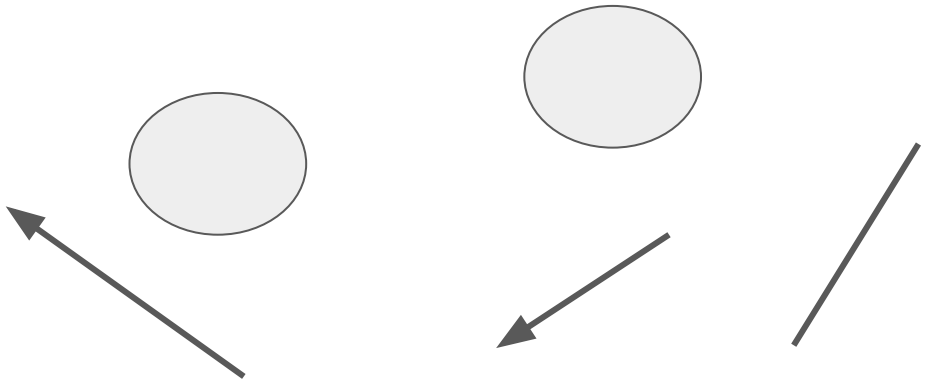
Graphs



Unweighted, directed

Graph representation

- Vertices
- Edges



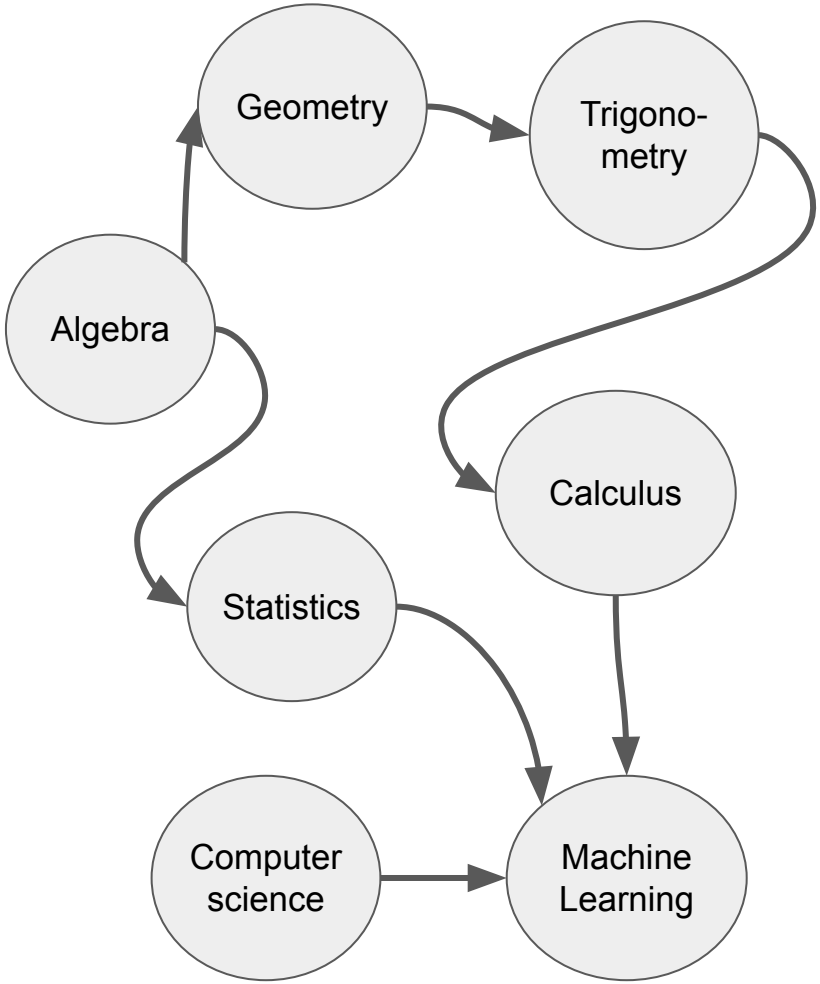
How to connect vertices through edges?

Graph applications examples

- **Networking:** finding the shortest paths, managing the flow
- **Google maps:** building transportation systems
- **Google page ranking:** evaluating pages by the number of references they have
- **Facebook:** friends suggestion algorithm
- **Neural networks**
- **Biological networks**

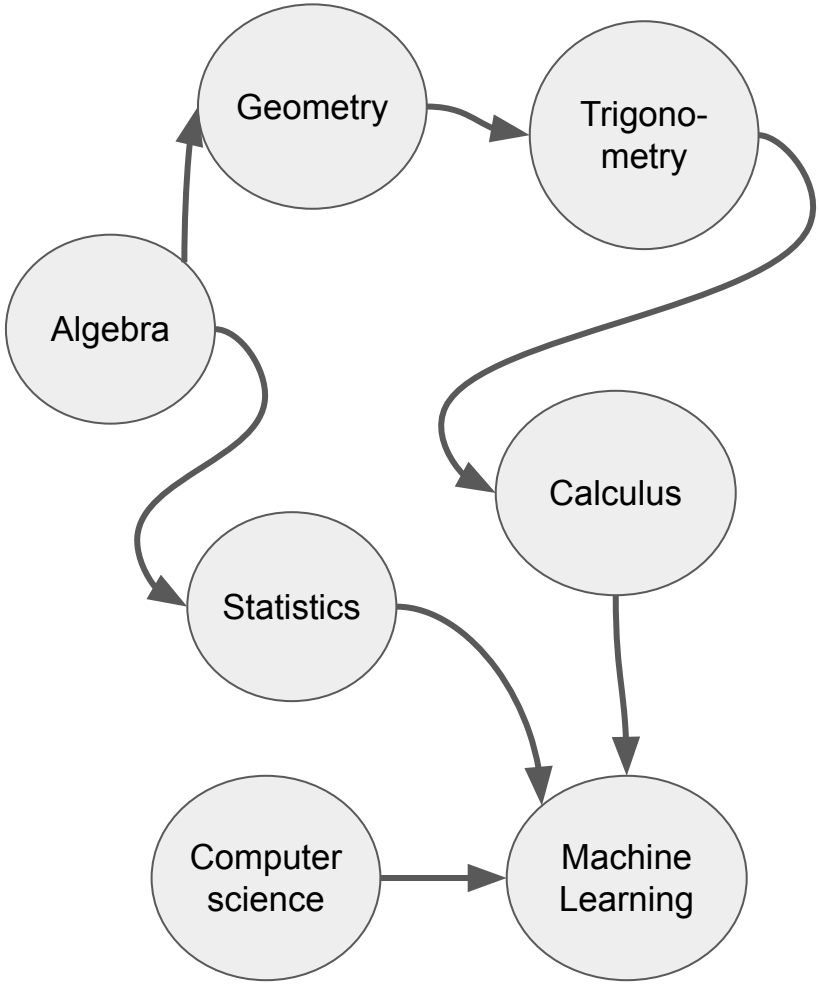
Graph representation: adjacency matrix

	Alg	Geo	Tri	Cal	Sta	CS	ML
Alg							
Geo							
Tri							
Cal							
Sta							
CS							
ML							



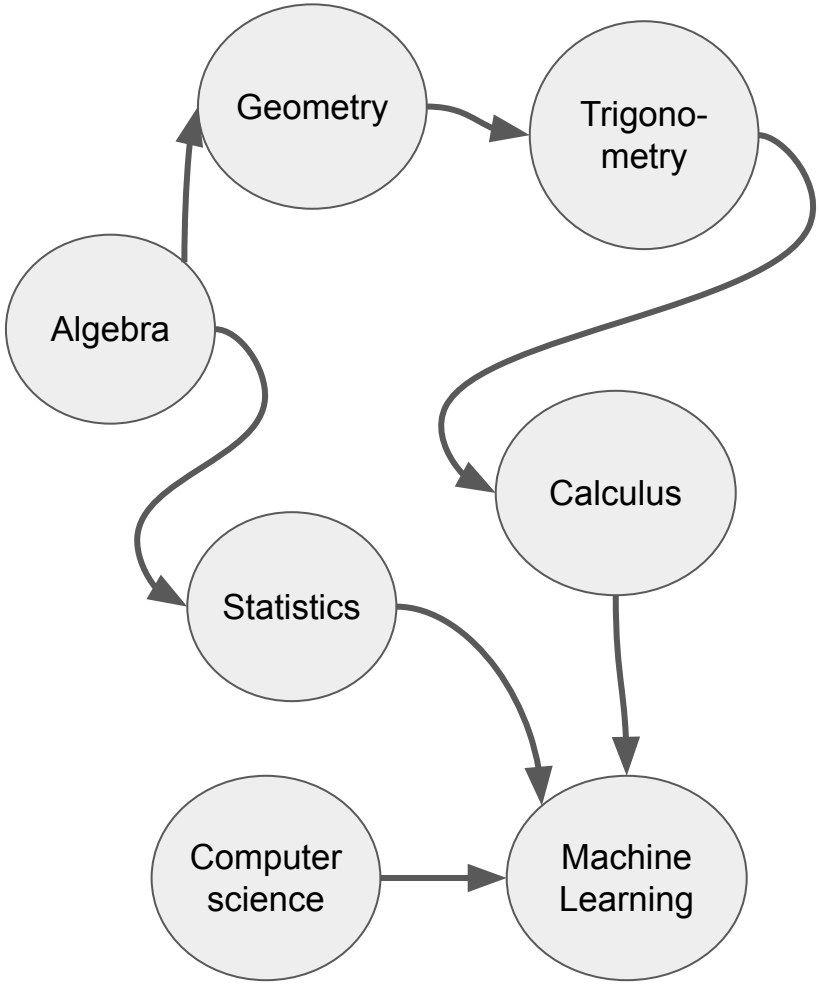
Graph representation: adjacency matrix

	Alg	Geo	Tri	Cal	Sta	CS	ML
Alg		1					
Geo							
Tri							
Cal							
Sta							
CS							
ML							



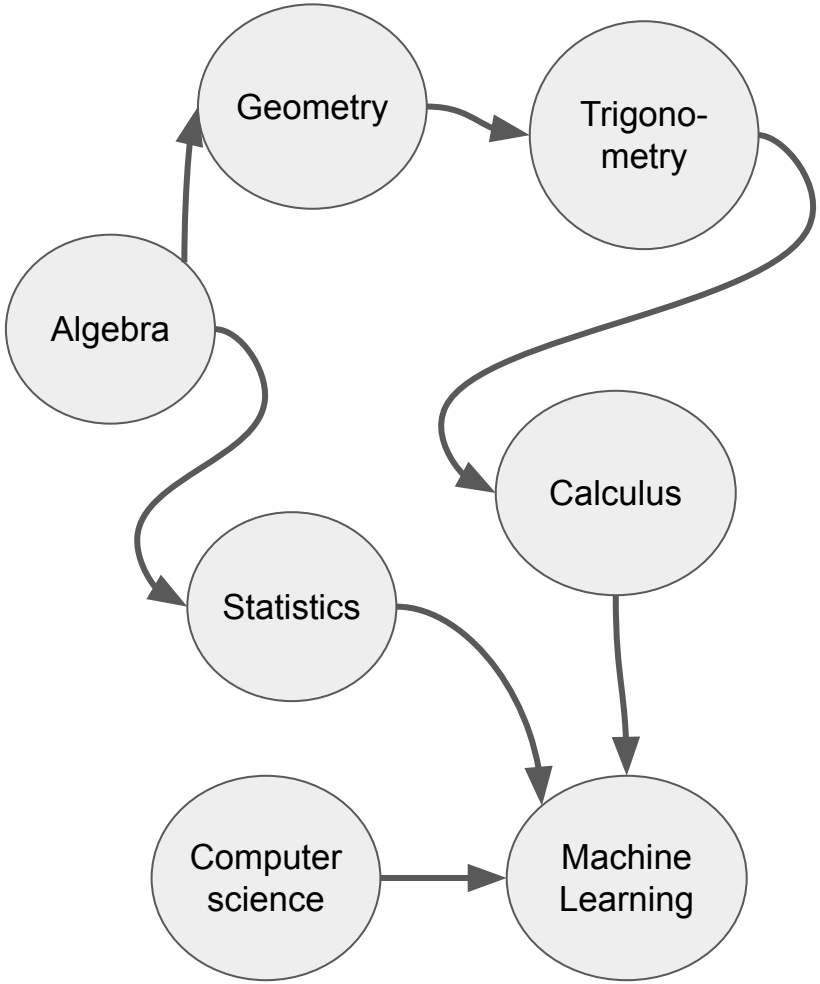
Graph representation: adjacency matrix

	Alg	Geo	Tri	Cal	Sta	CS	ML
Alg		1	0	0	1	0	0
Geo							
Tri							
Cal							
Sta							
CS							
ML							



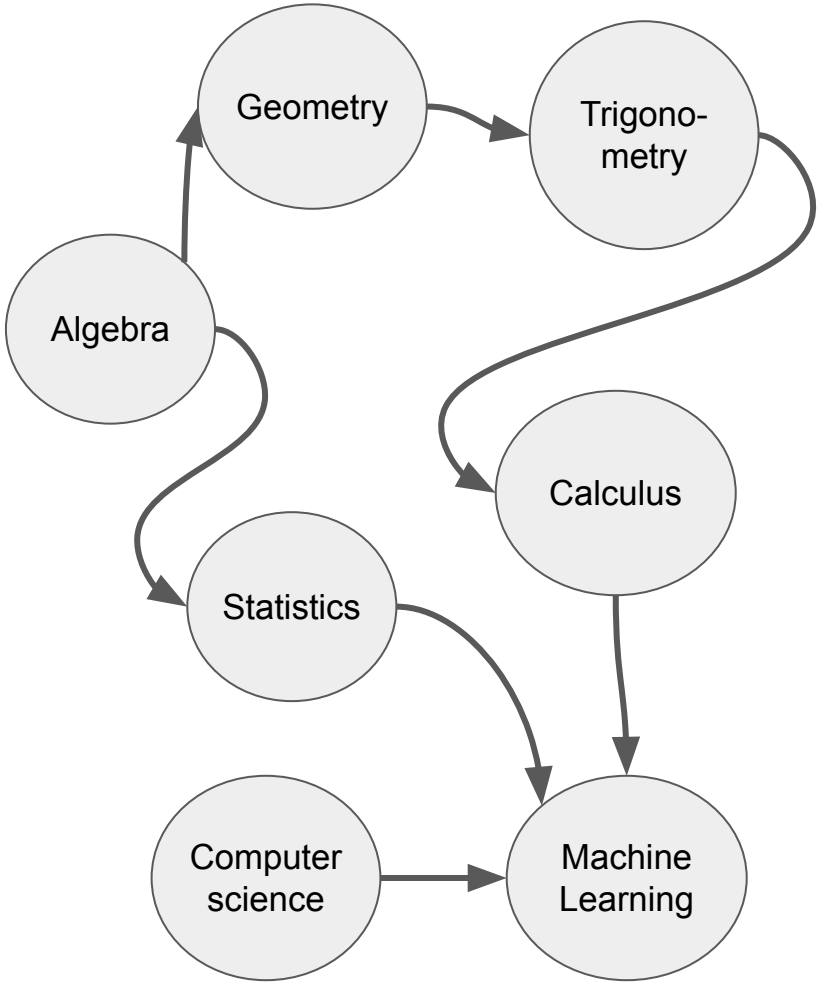
Graph representation: adjacency matrix

	Alg	Geo	Tri	Cal	Sta	CS	ML
Alg		1	0	0	1	0	0
Geo	0		1	0	0	0	0
Tri							
Cal							
Sta							
CS							
ML							



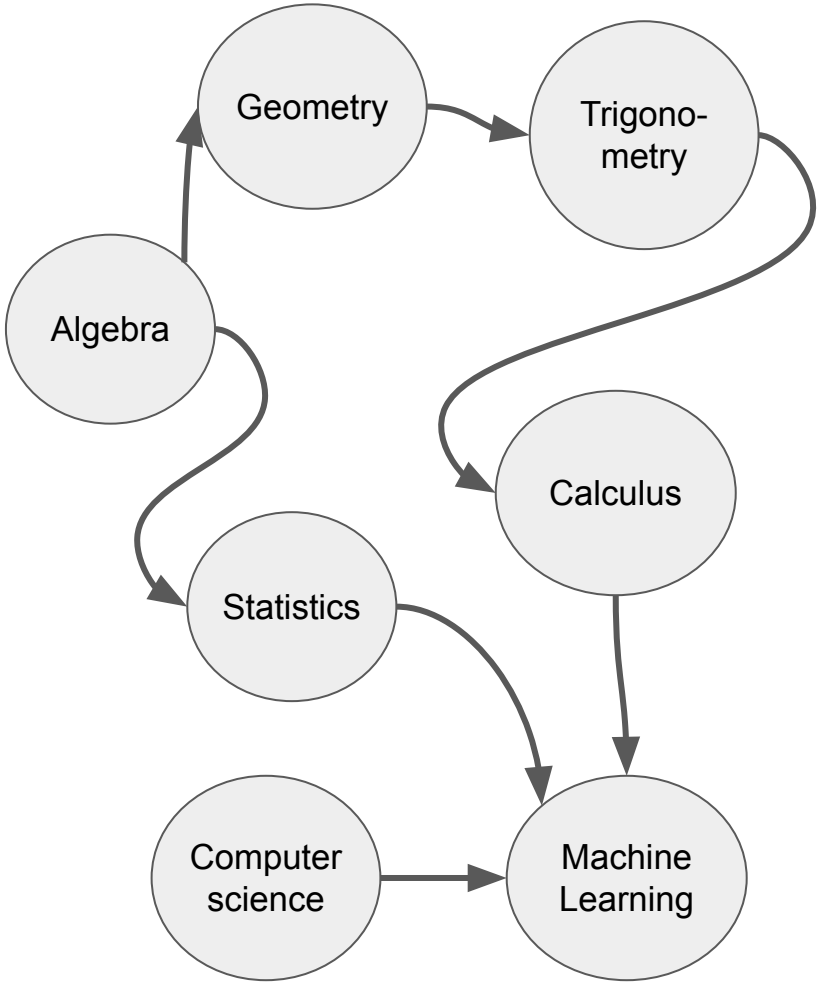
Graph representation: adjacency matrix

	Alg	Geo	Tri	Cal	Sta	CS	ML
Alg		1	0	0	1	0	0
Geo	0		1	0	0	0	0
Tri	0	0		1	0	0	0
Cal							
Sta							
CS							
ML							



Graph representation: adjacency matrix

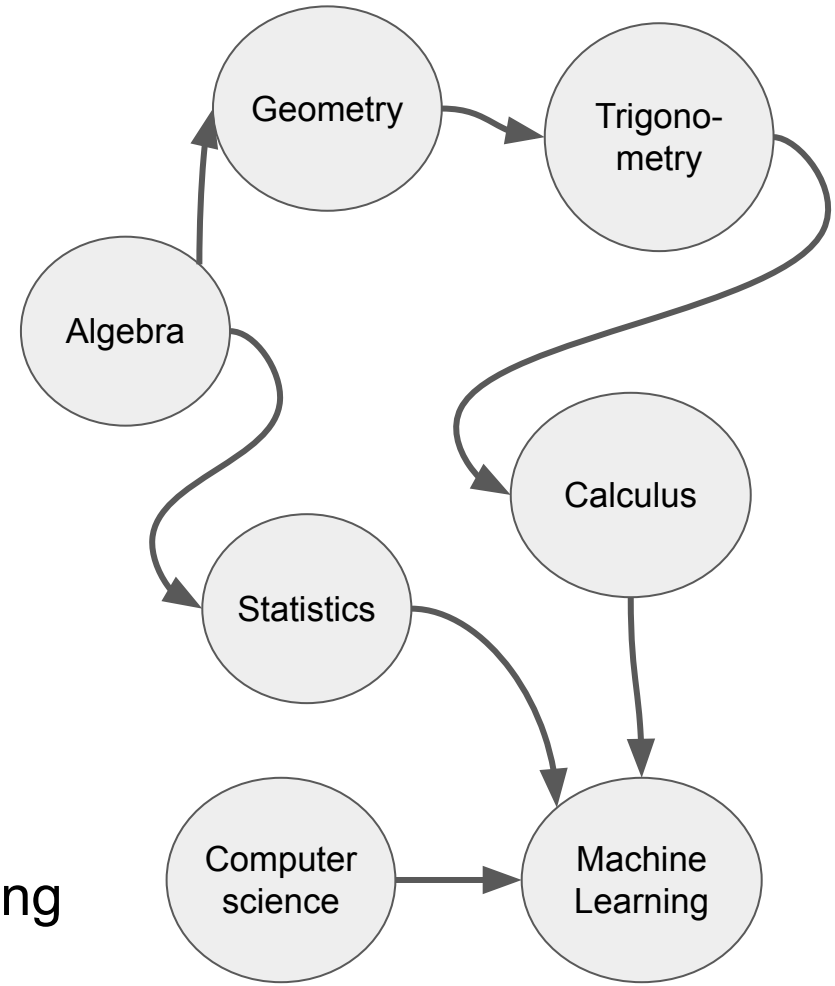
	Alg	Geo	Tri	Cal	Sta	CS	ML
Alg		1	0	0	1	0	0
Geo	0		1	0	0	0	0
Tri	0	0		1	0	0	0
Cal	0	0	0		0	0	1
Sta	0	0	0	0		0	1
CS	0	0	0	0	0		1
ML	0	0	0	0	0	0	



Graph representation: adjacency list

keys **values**

- **Algebra:** Geometry, Statistics
- **Geometry:** Trigonometry
- **Trigonometry**: Calculus
- **Calculus**: Machine Learning
- **Statistics**: Machine Learning
- **Computer Science:** Machine learning



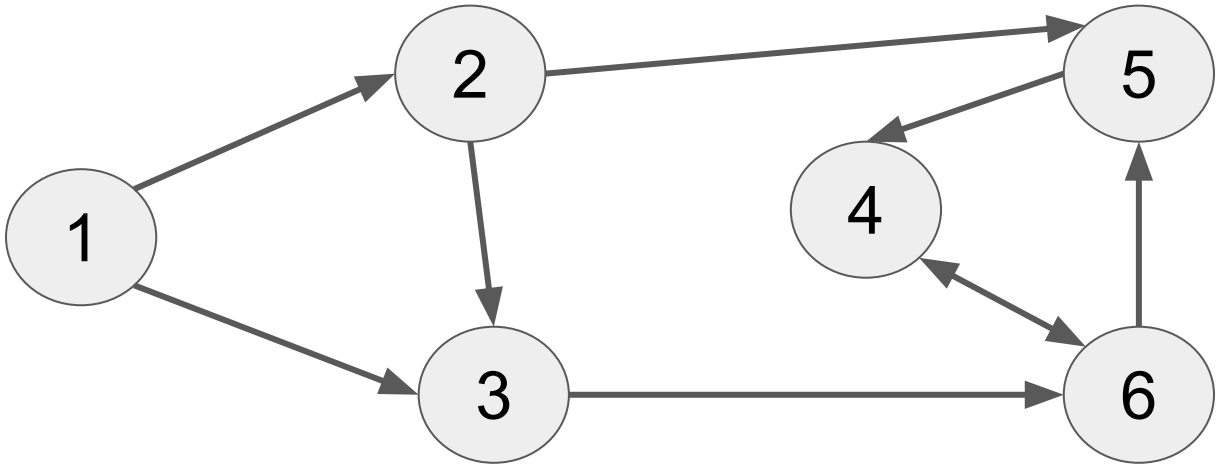
Graph traversals

- The process of visiting each vertex in a graph.
  - Checking, updating vertexes during the visit.
- Is there a challenge compared to tree traversals?
  - A different structure: node can be visited multiple times
- Traversal algorithms:
  - Breadth first search
  - Depth first search

Traversal algorithms

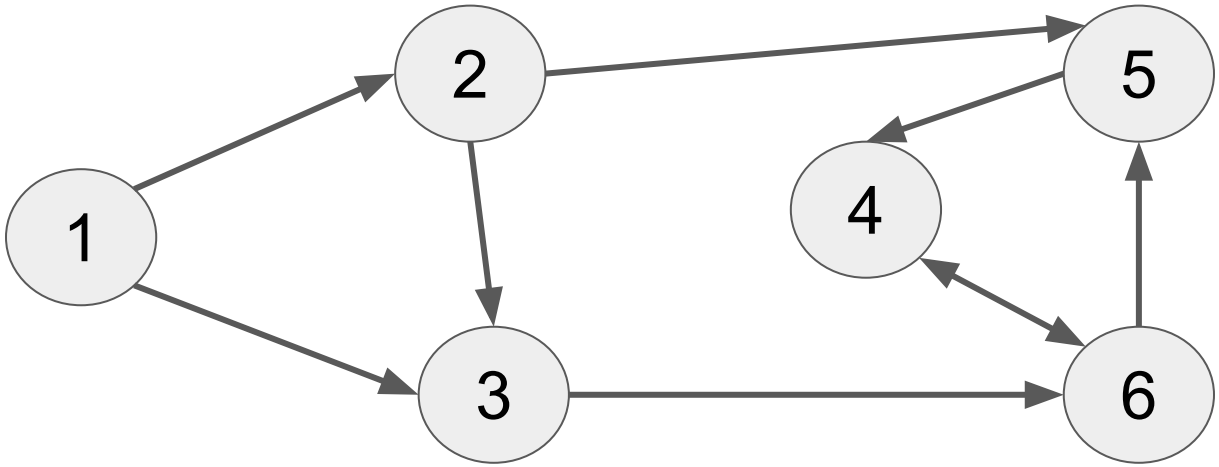
- DFS: visiting a child before siblings
  - Implementation?
  - Adding children into a stack
- BFS: visiting a sibling before visiting children
  - Implementation?
  - Adding children into a queue
- **Practice: implement BFS and DFS for a graph**

Depth-first search

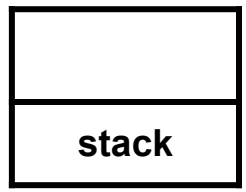


dfs(node=1, graph)

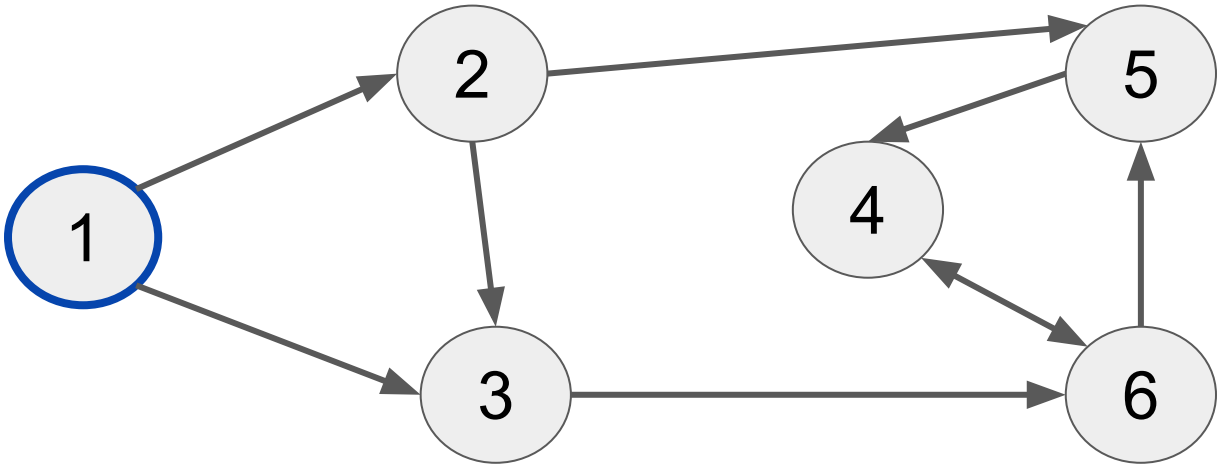
Depth-first search



dfs(node=1, graph)



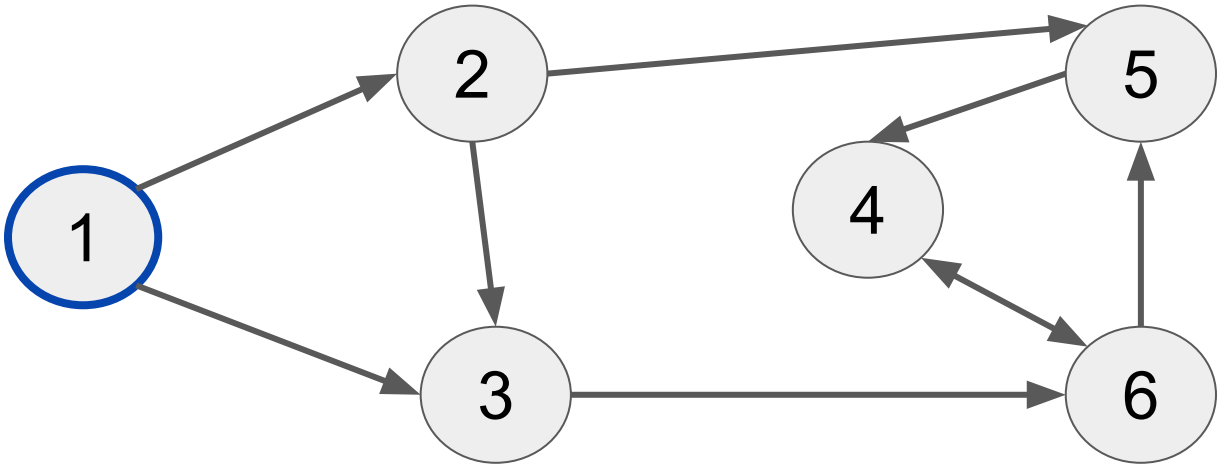
Depth-first search



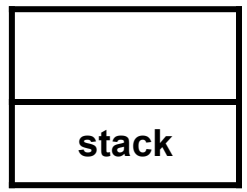
dfs(node=1, graph)

1
stack

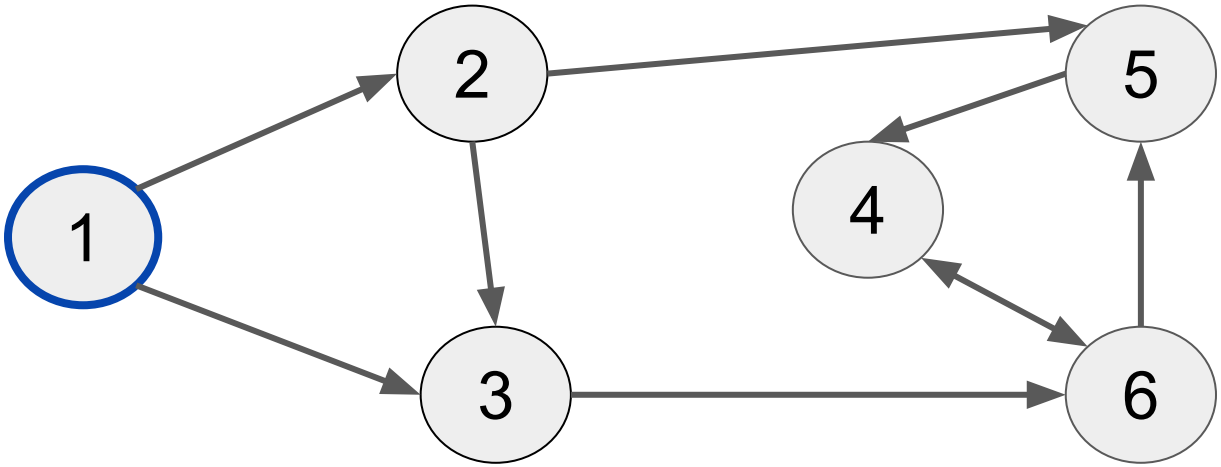
Depth-first search



dfs(node=1, graph)



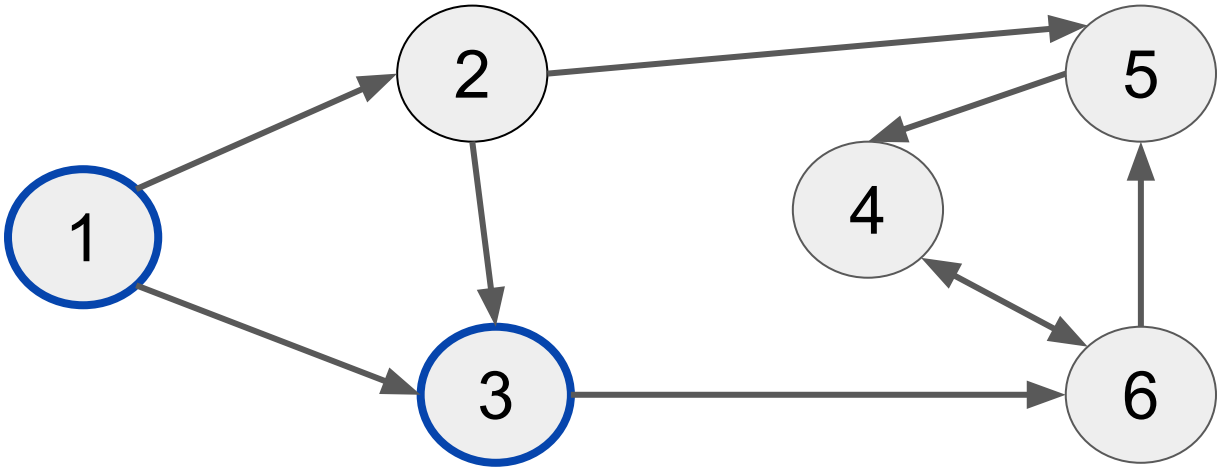
Depth-first search



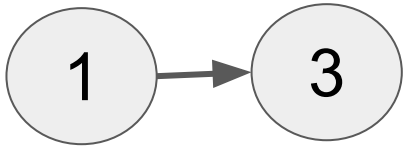
dfs(node=1, graph)

3
2
stack

Depth-first search

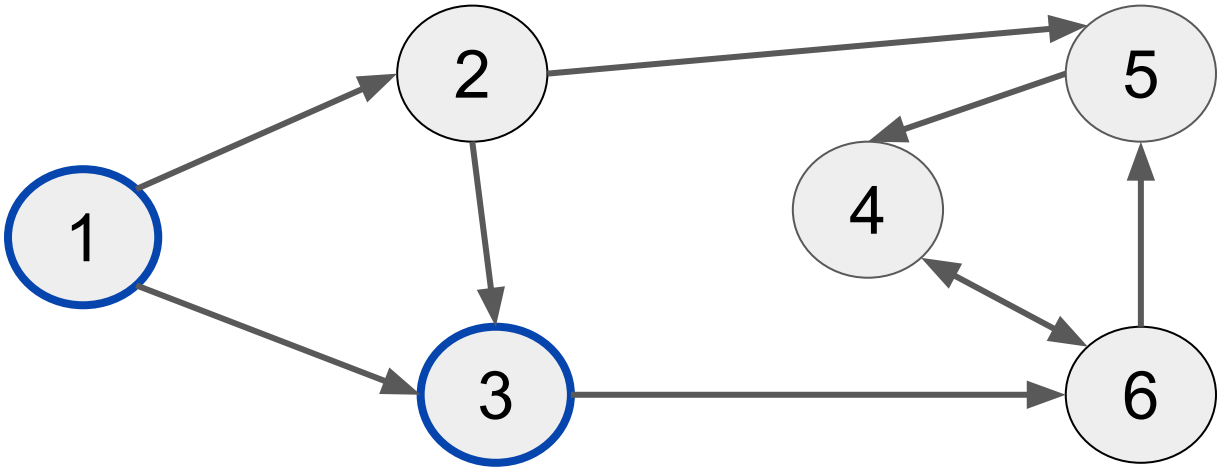


dfs(node=1, graph)

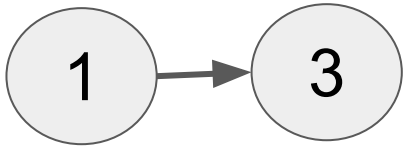


2
stack

Depth-first search

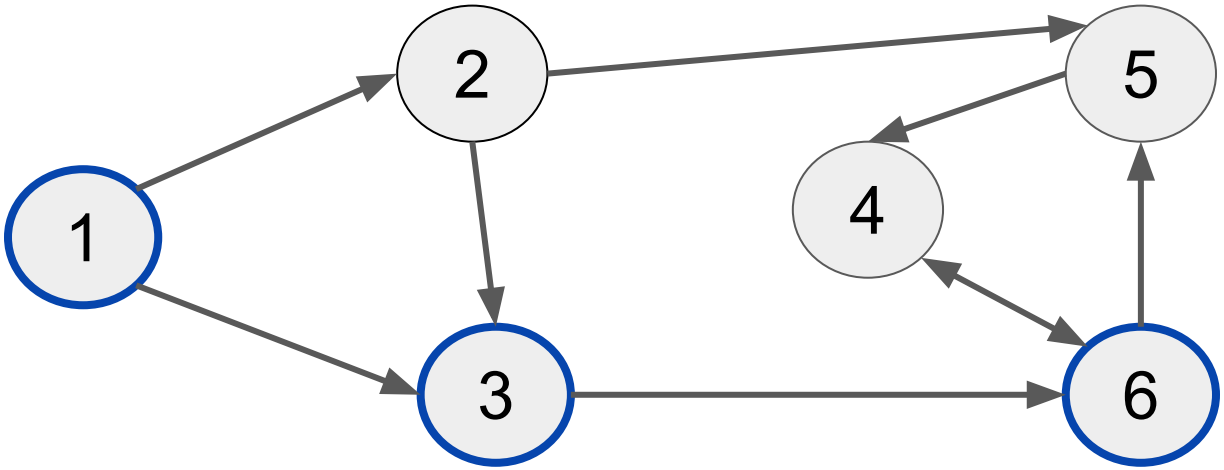


dfs(node=1, graph)



6
2
stack

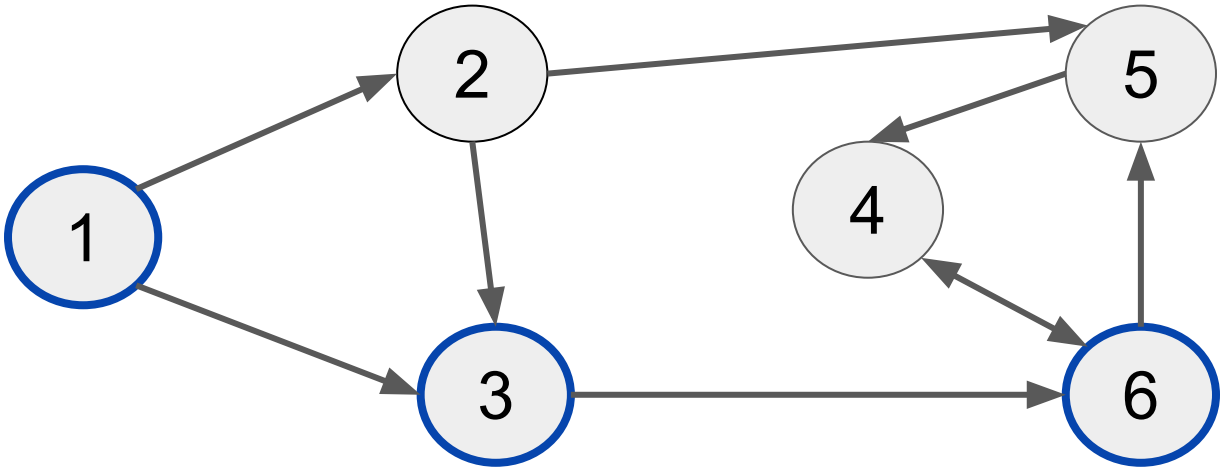
Depth-first search



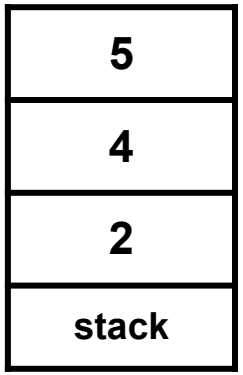
dfs(node=1, graph)

2
stack

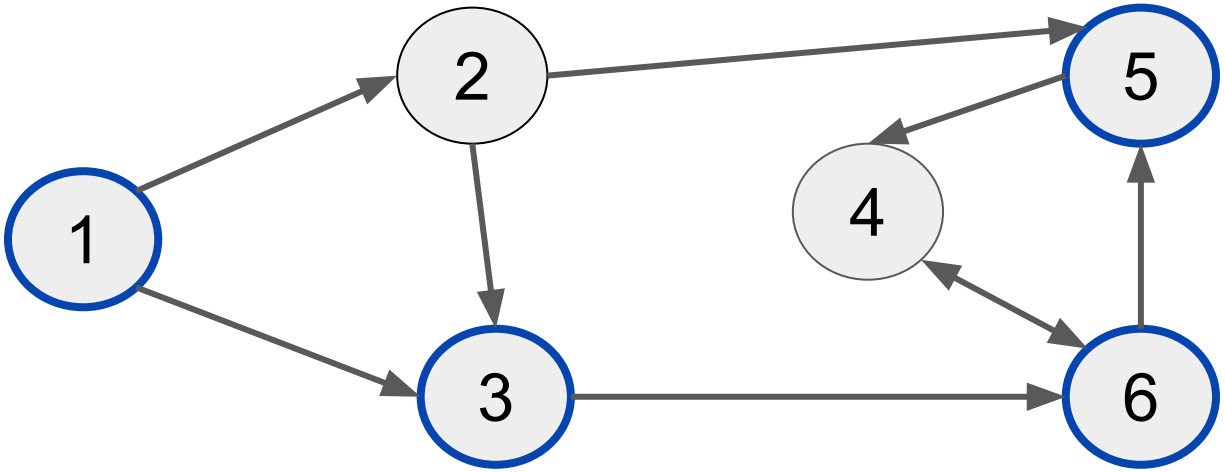
Depth-first search



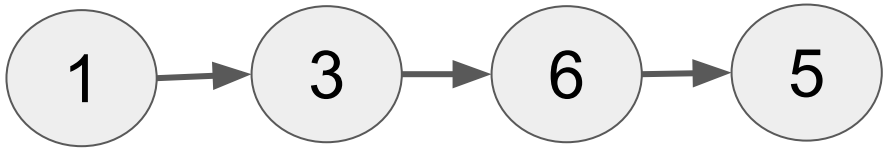
dfs(node=1, graph)



Depth-first search

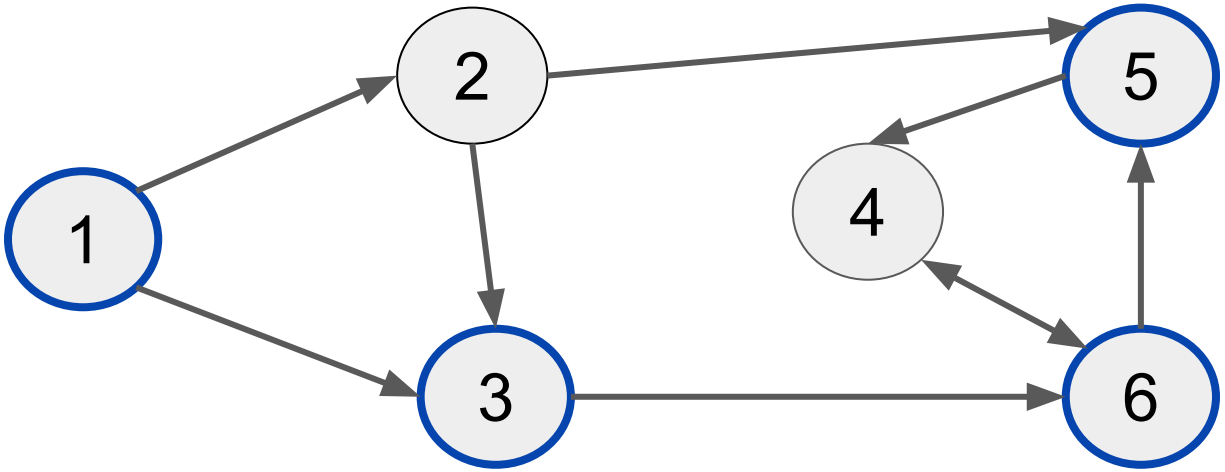


dfs(node=1, graph)

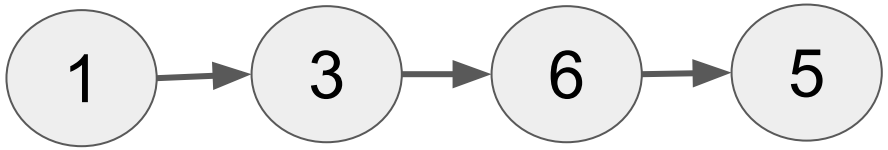


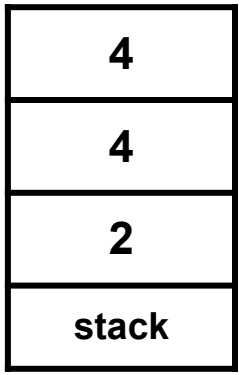
stack
2
4

Depth-first search

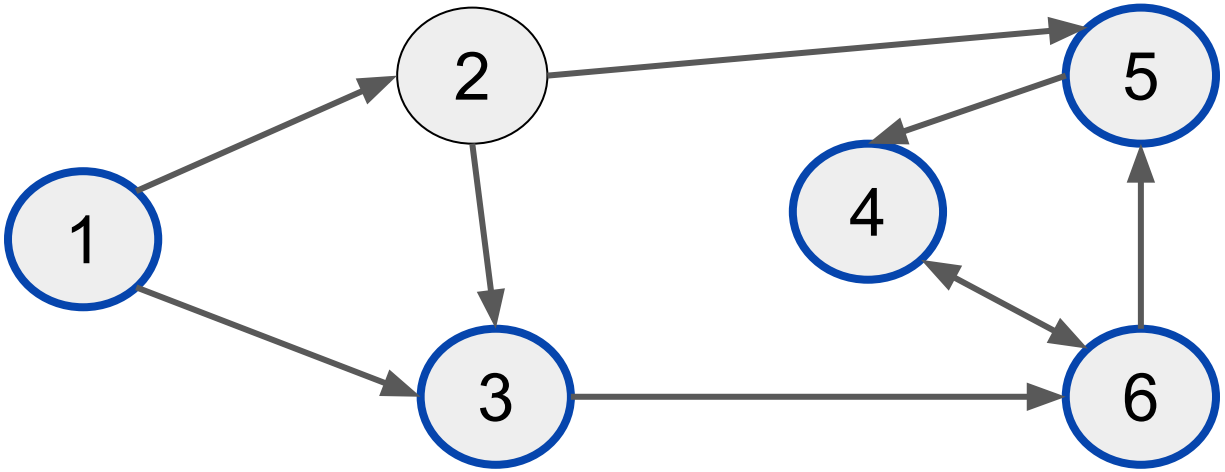


dfs(node=1, graph)

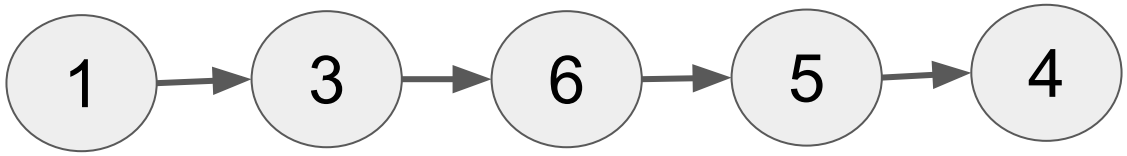




Depth-first search

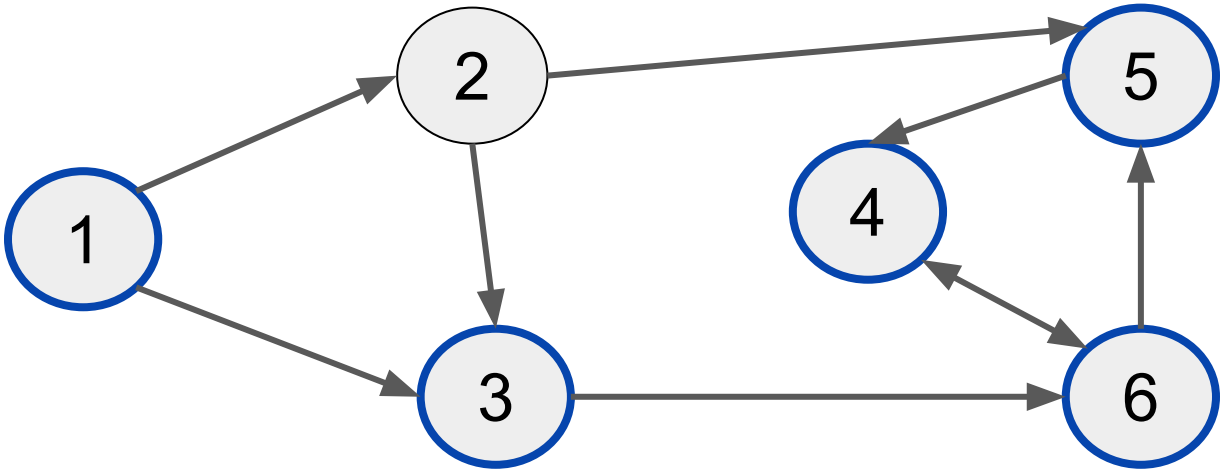


dfs(node=1, graph)

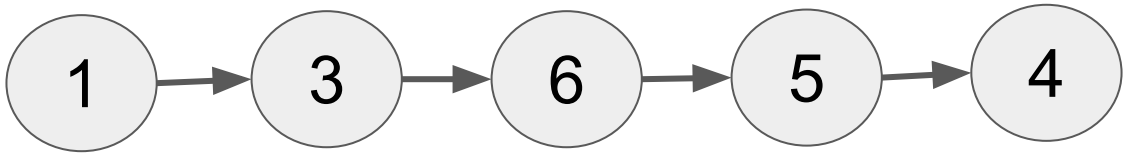


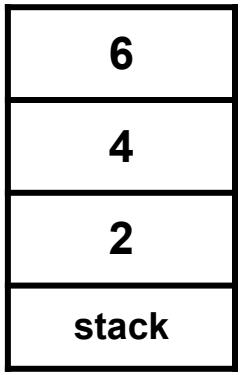
stack
2
4

Depth-first search

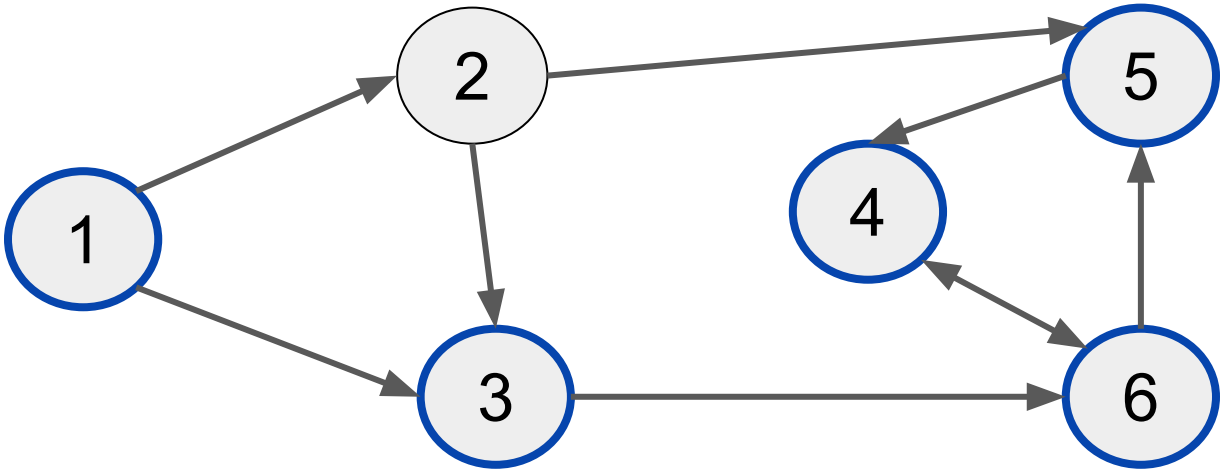


dfs(node=1, graph)

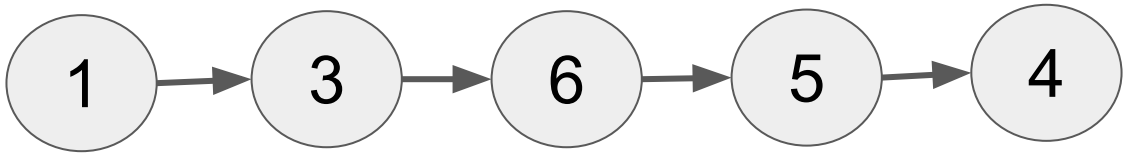


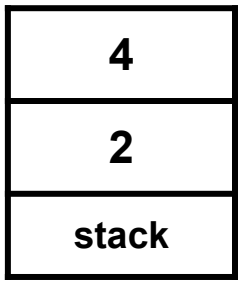


Depth-first search

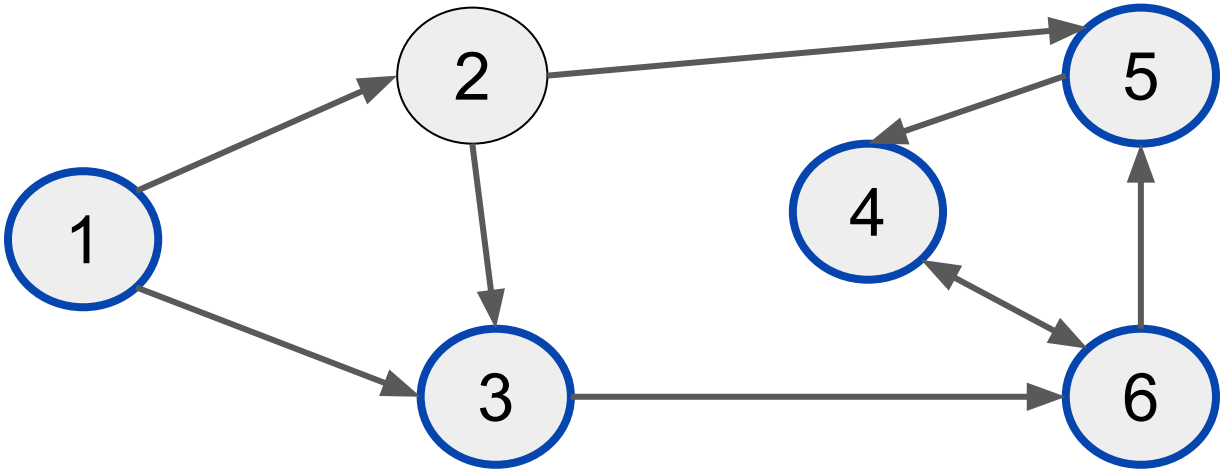


dfs(node=1, graph)

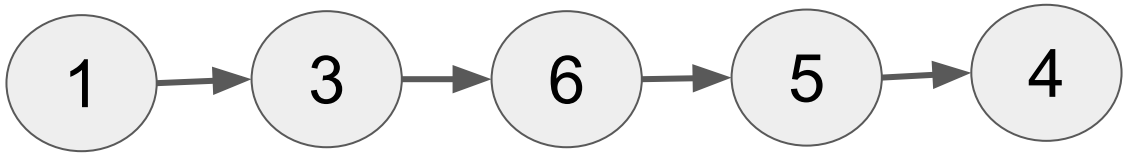


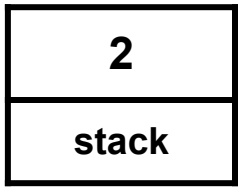


Depth-first search

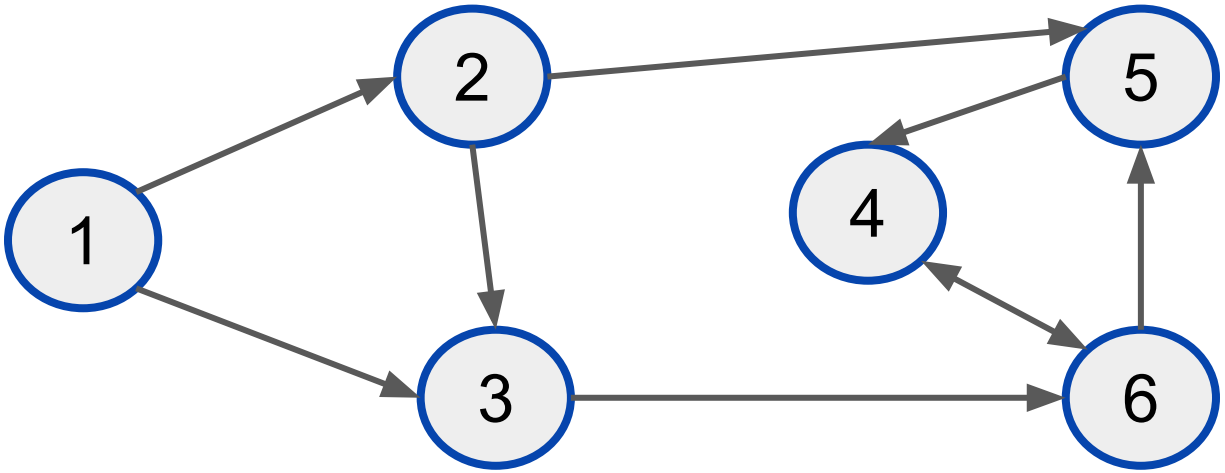


dfs(node=1, graph)

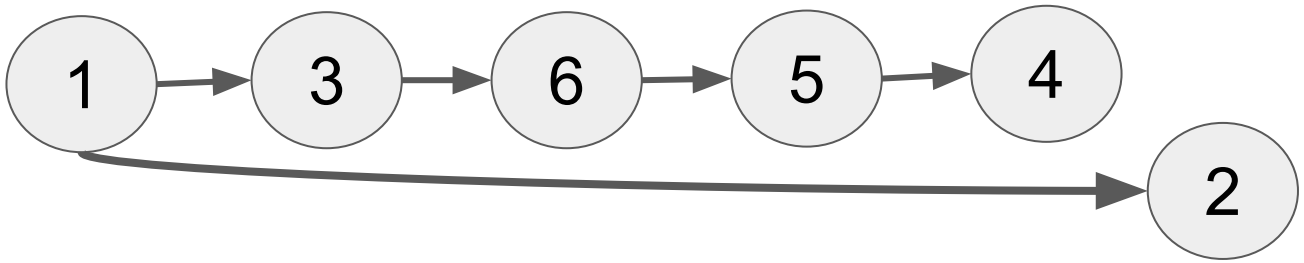




Depth-first search

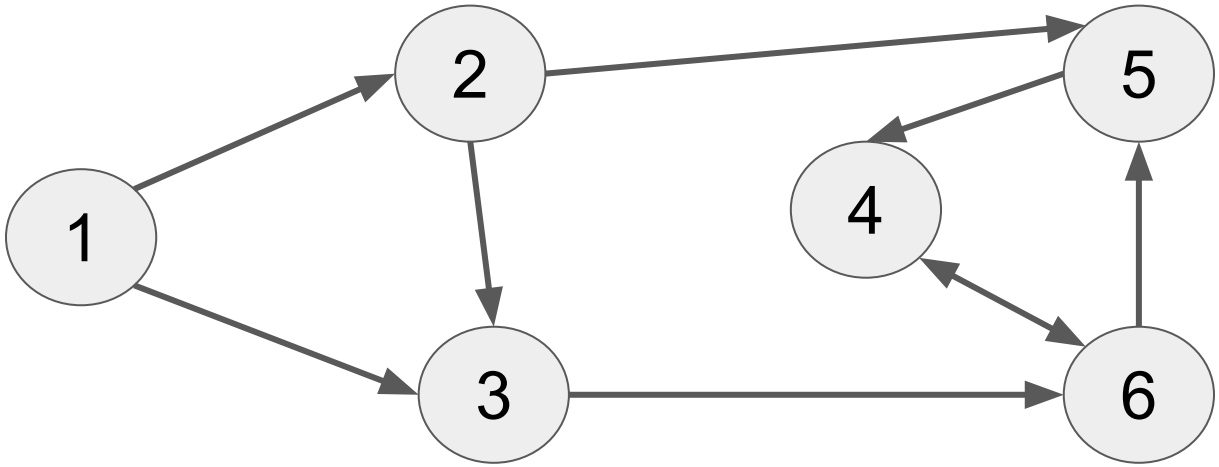


dfs(node=1, graph)



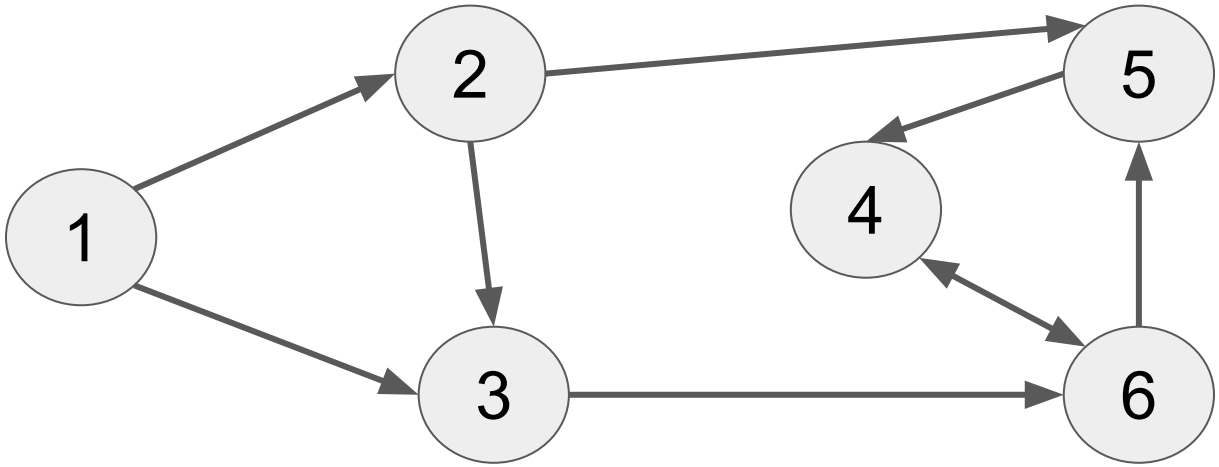
stack

Breadth-first search



bfs(node=1, graph)

Breadth-first search

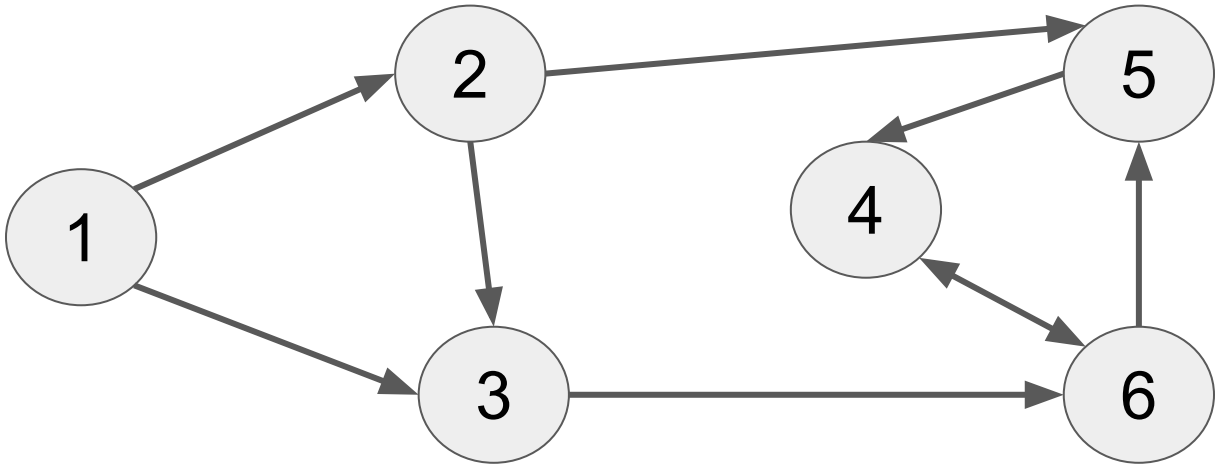


bfs(node=1, graph)

front	rear

Queue

Breadth-first search

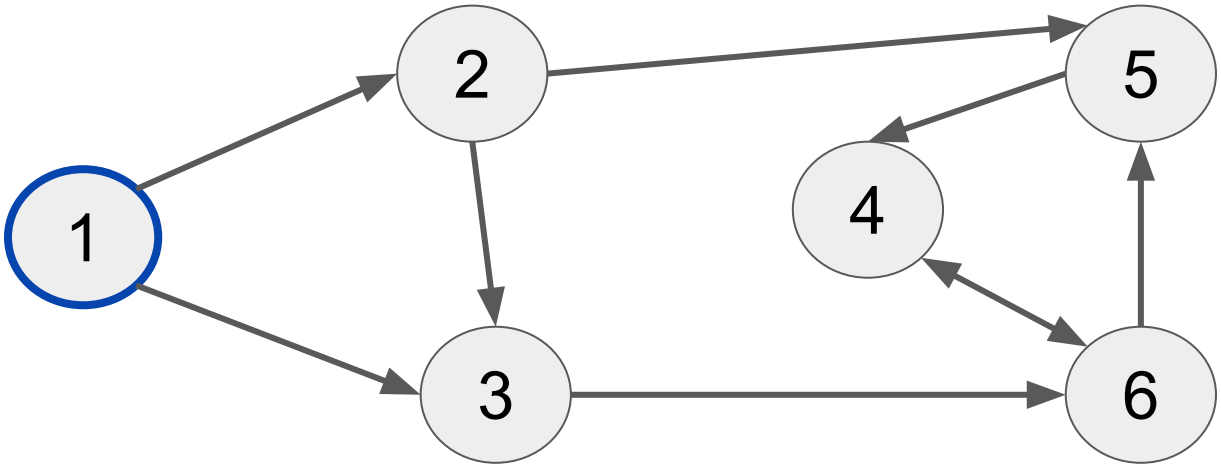


bfs(node=1, graph)

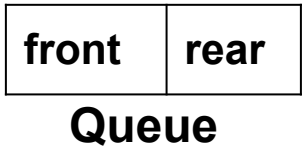
front	1	rear

Queue

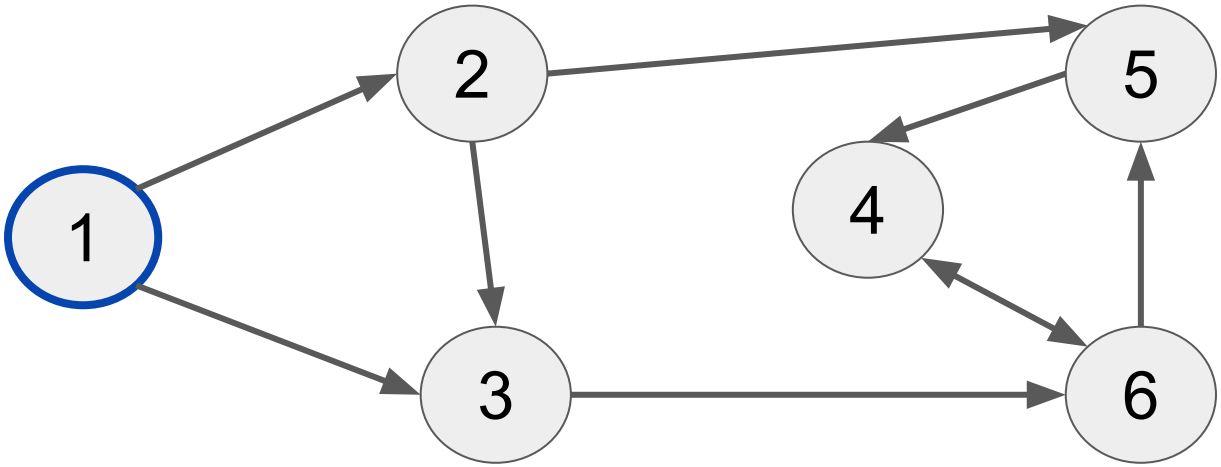
Breadth-first search



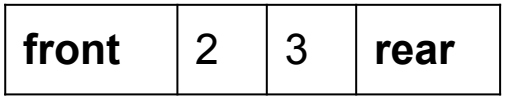
bfs(node=1, graph)



Breadth-first search

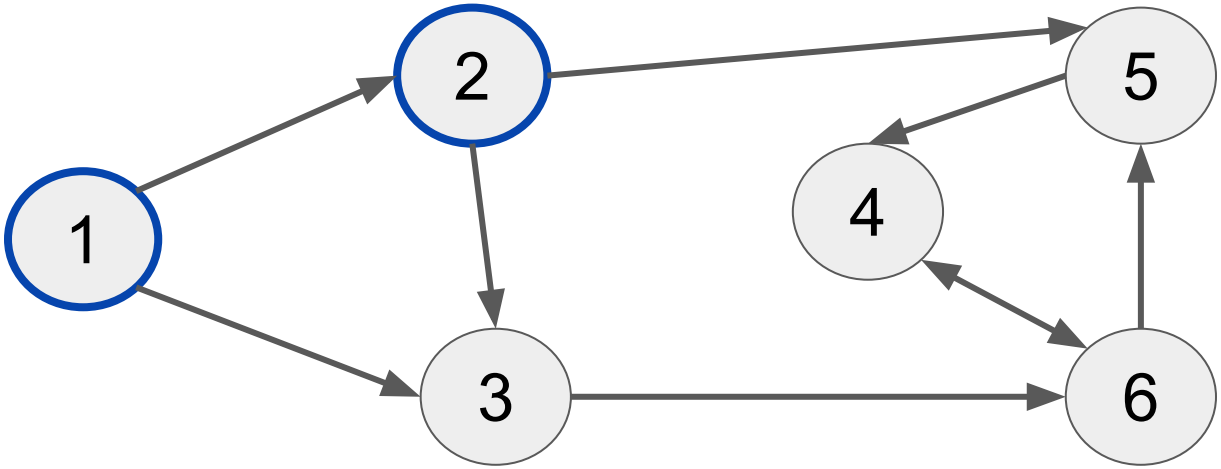


bfs(node=1, graph)

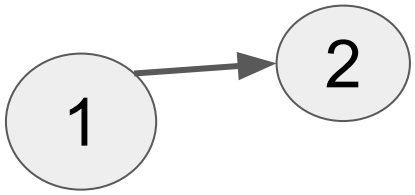


Queue

Breadth-first search



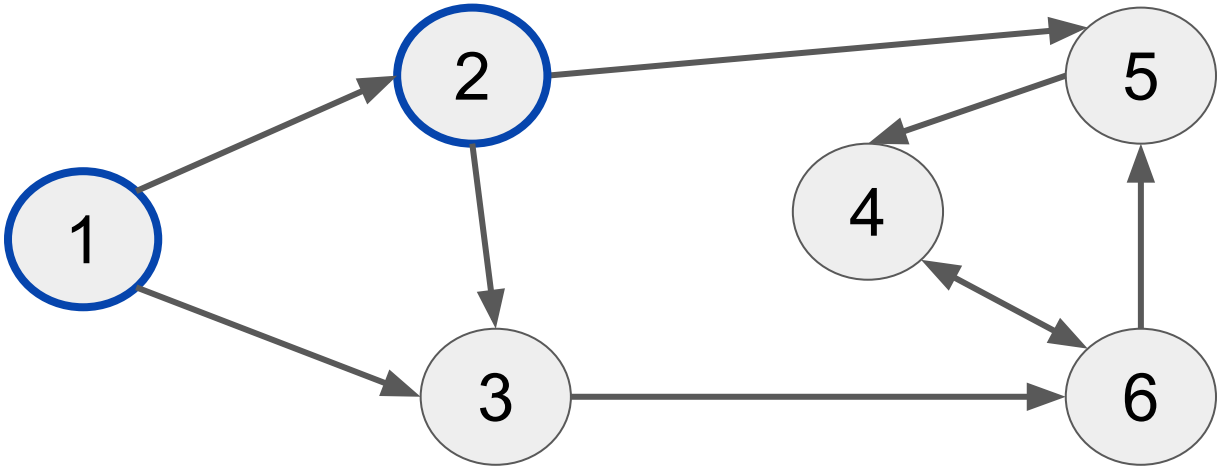
bfs(node=1, graph)



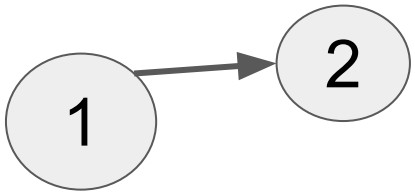
front	3	rear

Queue

Breadth-first search



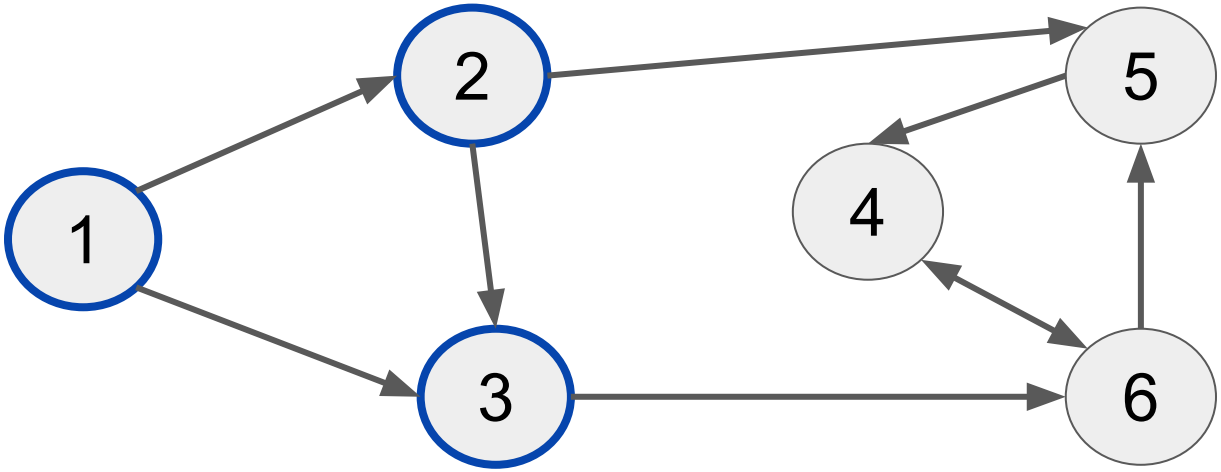
bfs(node=1, graph)



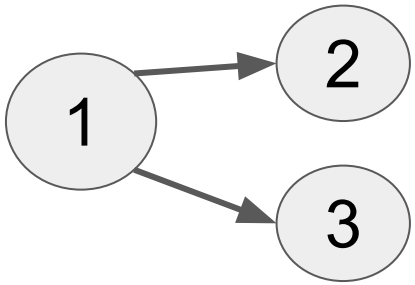
front	3	5	3	rear

Queue

Breadth-first search



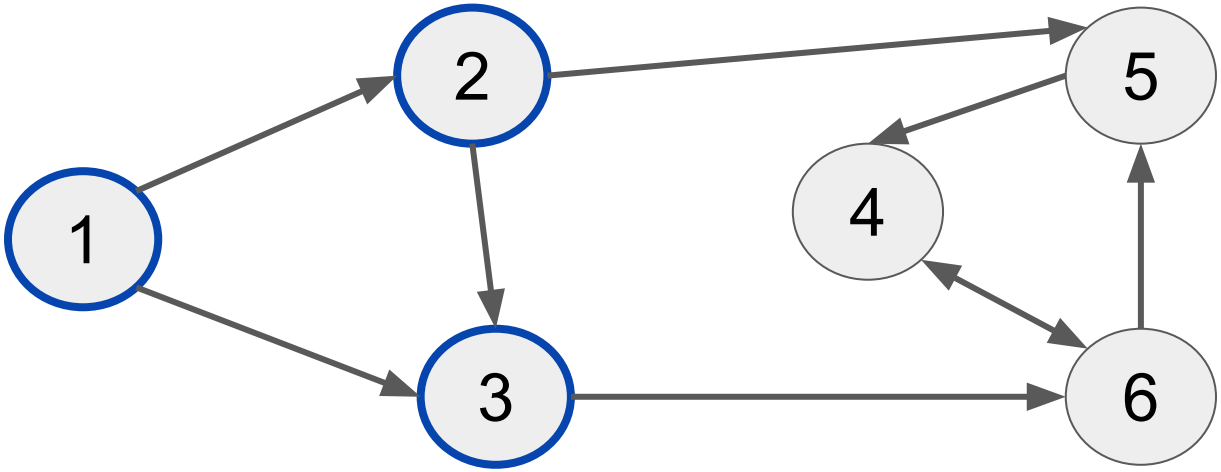
bfs(node=1, graph)



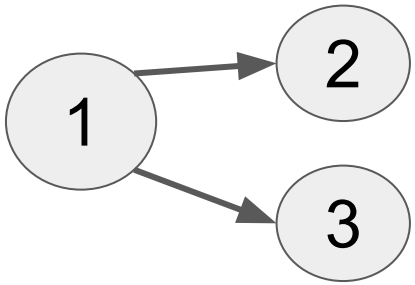
front	5	3	rear

Queue

Breadth-first search



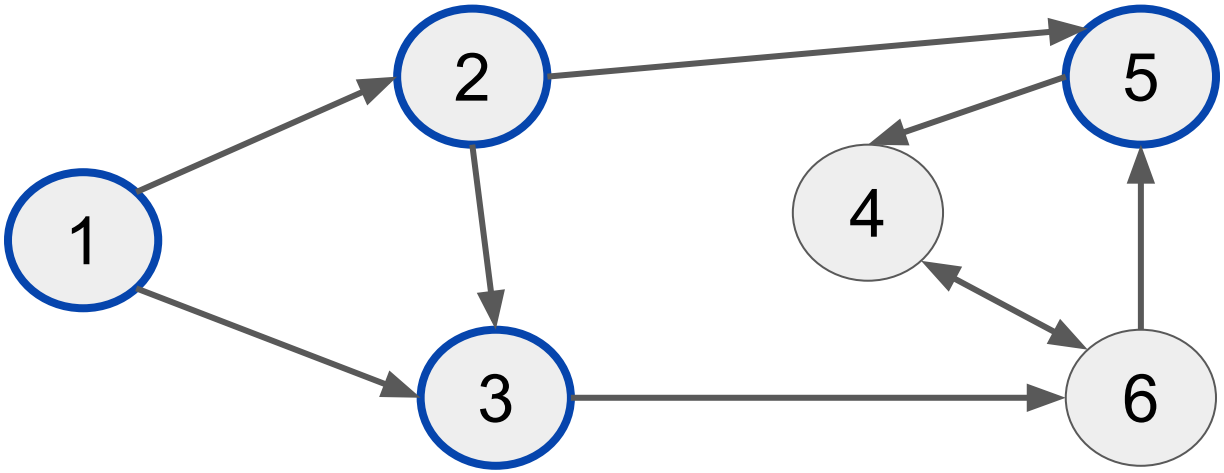
bfs(node=1, graph)



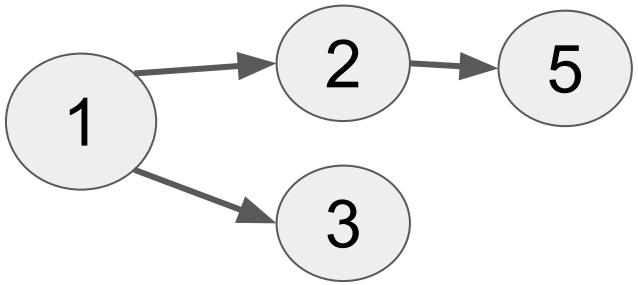
front	5	3	6	rear

Queue

Breadth-first search



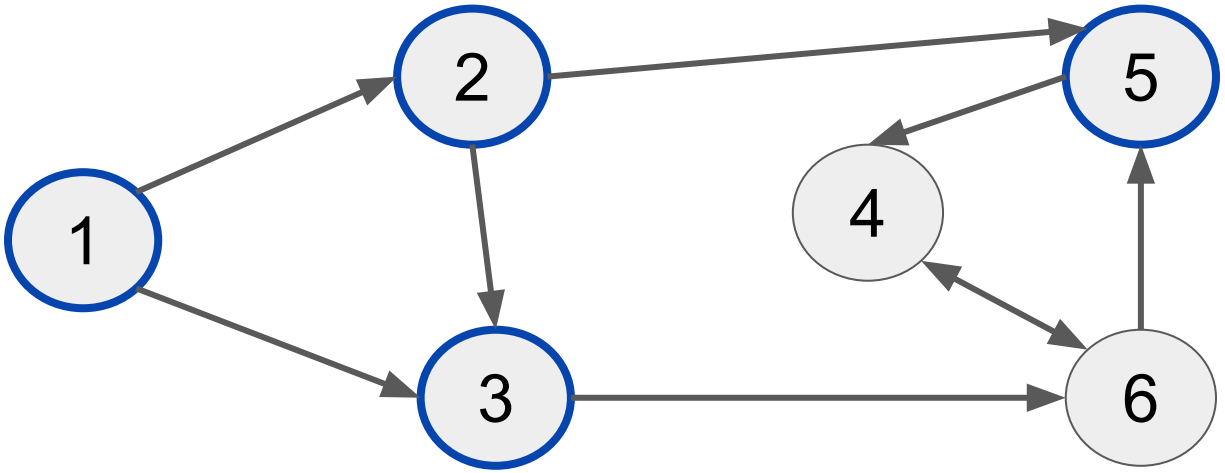
bfs(node=1, graph)



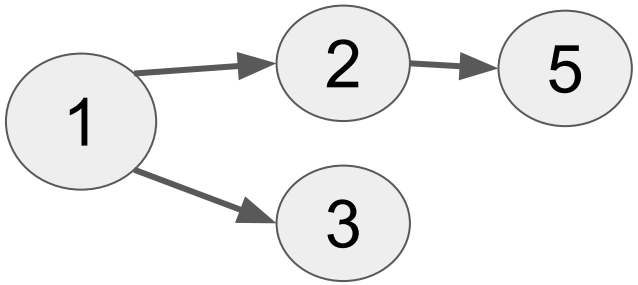
front	3	6	rear

Queue

Breadth-first search



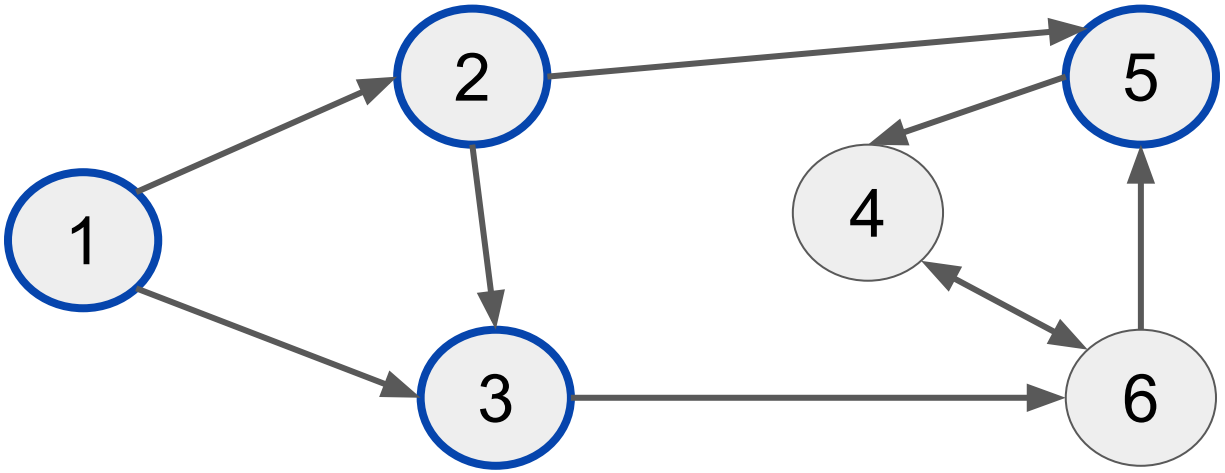
bfs(node=1, graph)



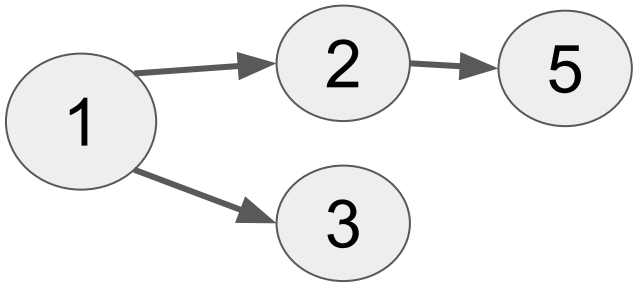
front	3	6	4	rear

Queue

Breadth-first search



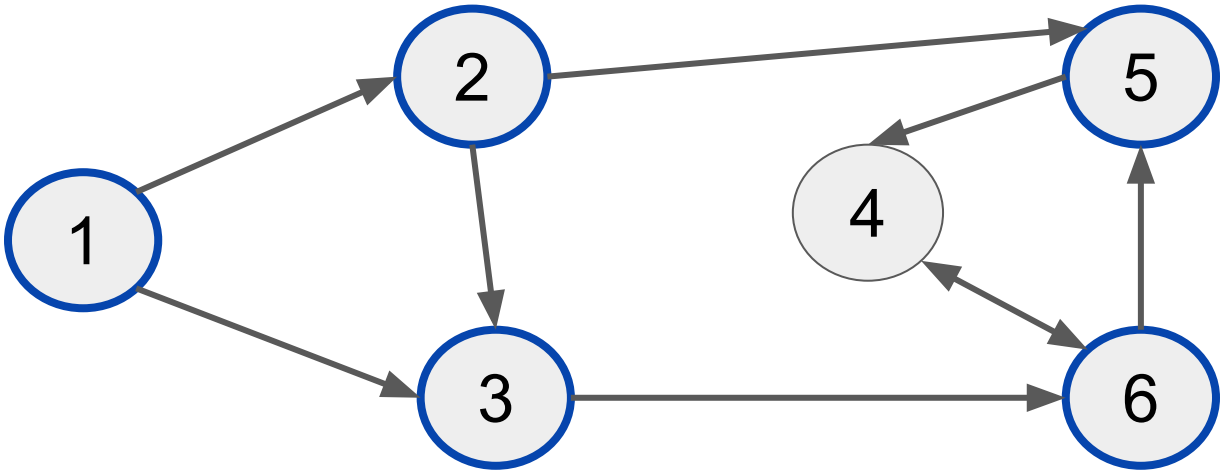
bfs(node=1, graph)



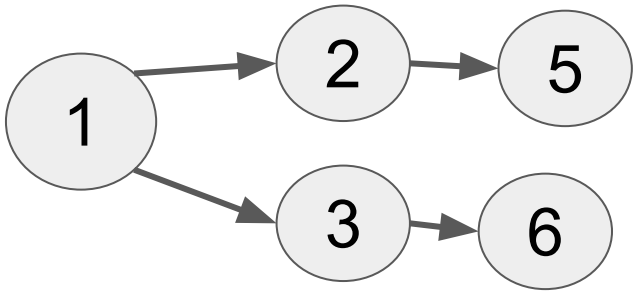
front	6	4	rear

Queue

Breadth-first search



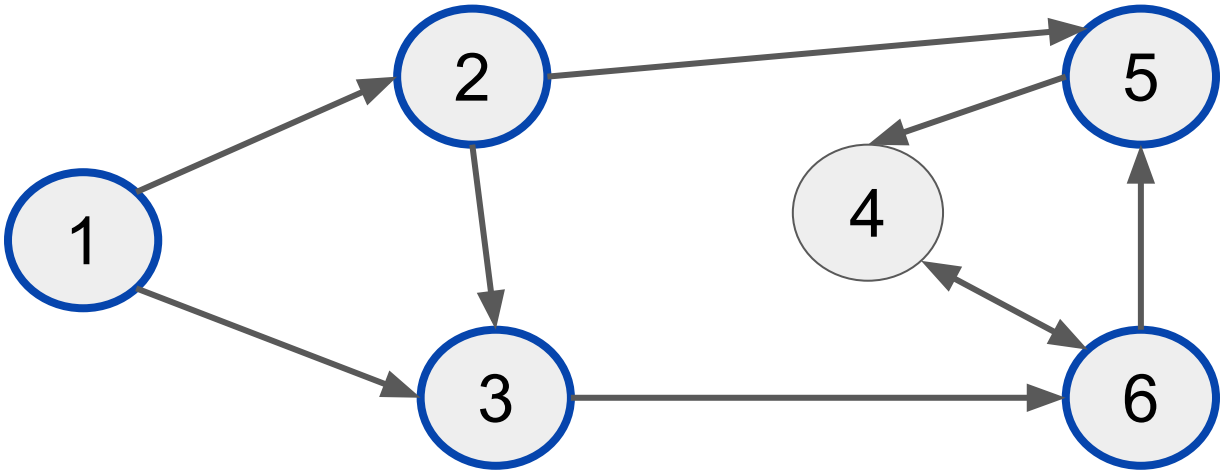
bfs(node=1, graph)



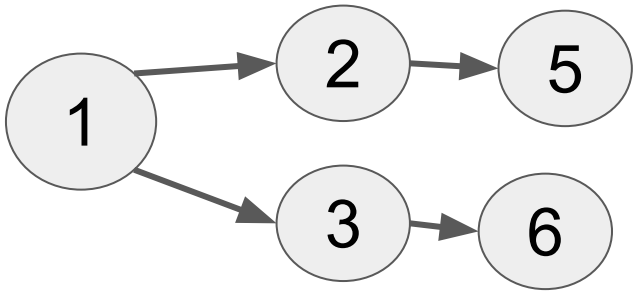
front	4	rear

Queue

Breadth-first search



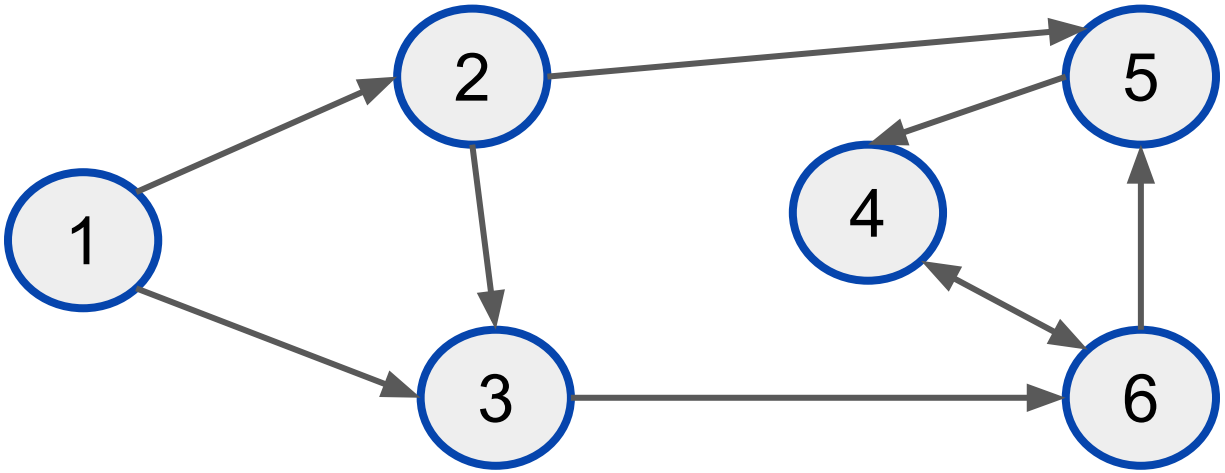
bfs(node=1, graph)



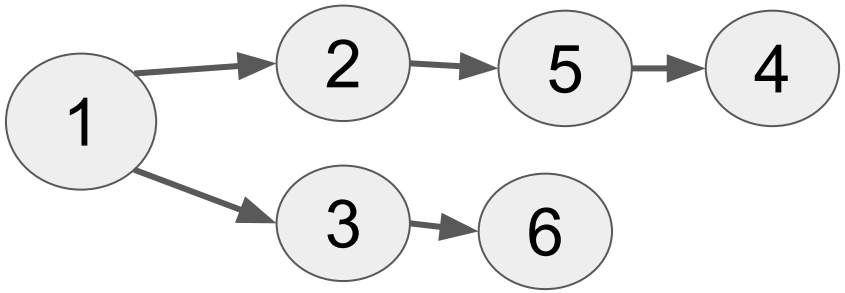
front	4	4	5	rear

Queue

Breadth-first search



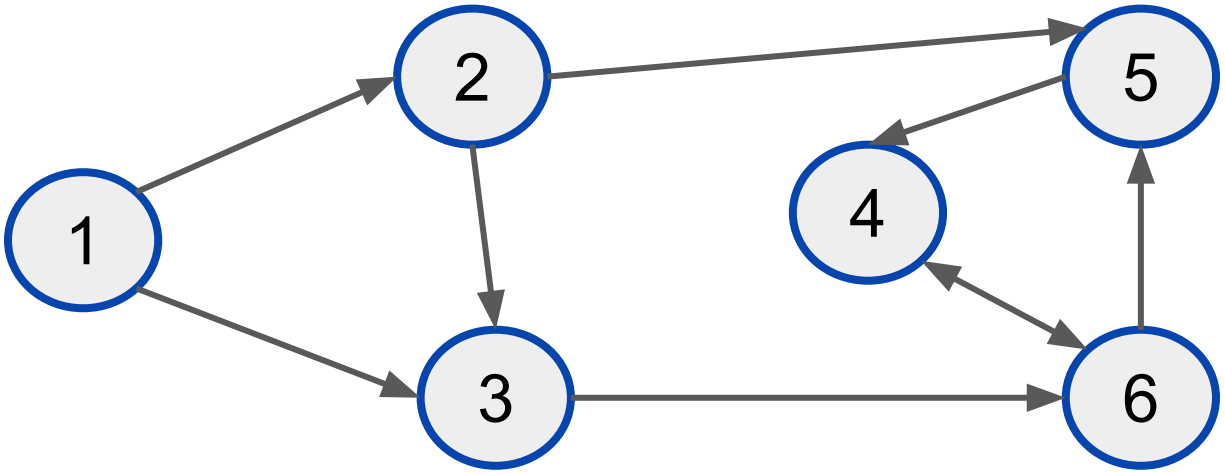
bfs(node=1, graph)



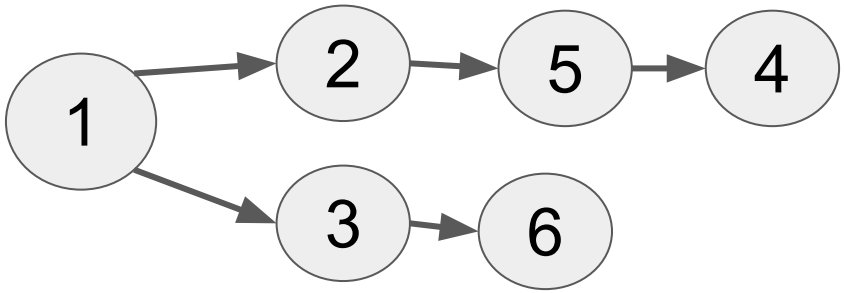
front	4	5	rear

Queue

Breadth-first search



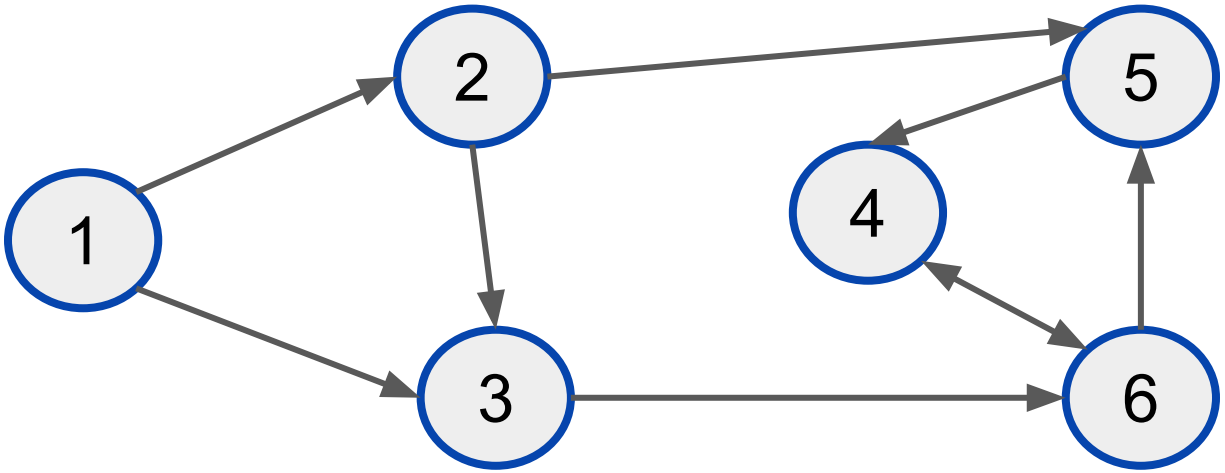
bfs(node=1, graph)



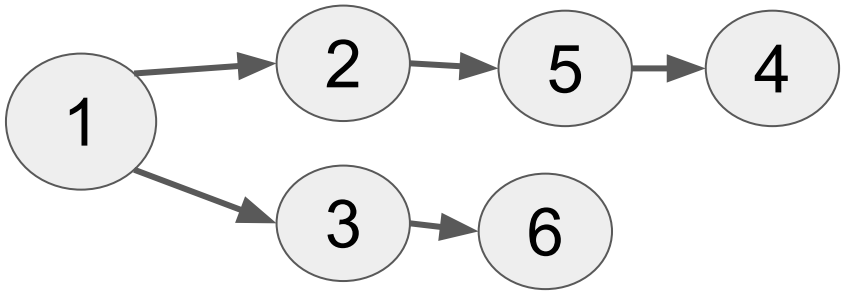
front	4	5	6	rear

Queue

Breadth-first search



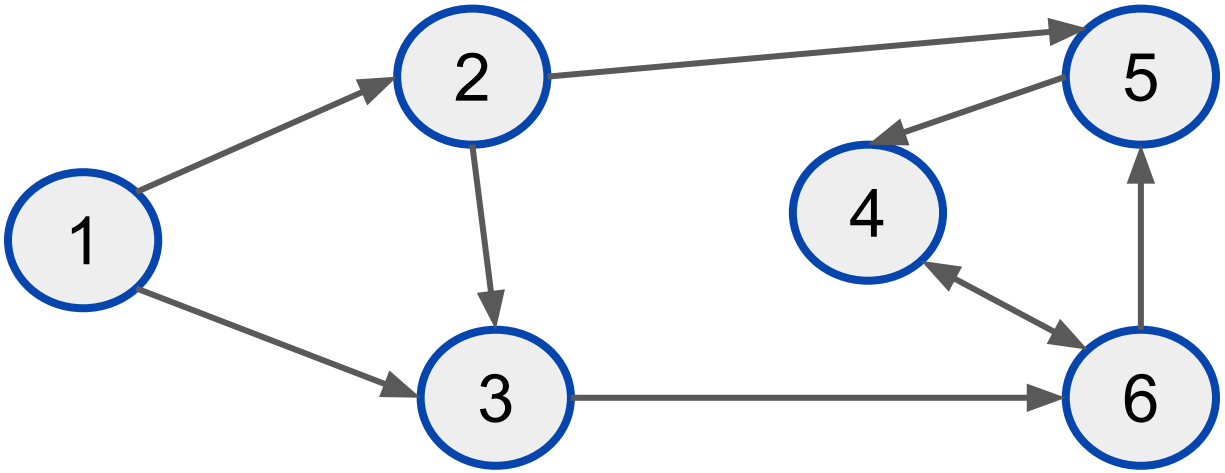
bfs(node=1, graph)



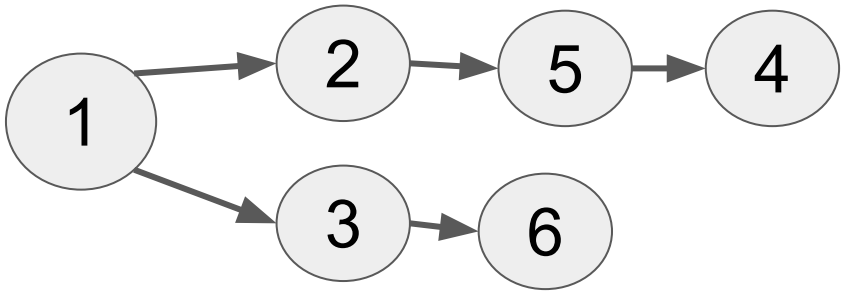
front	5	6	rear

Queue

Breadth-first search



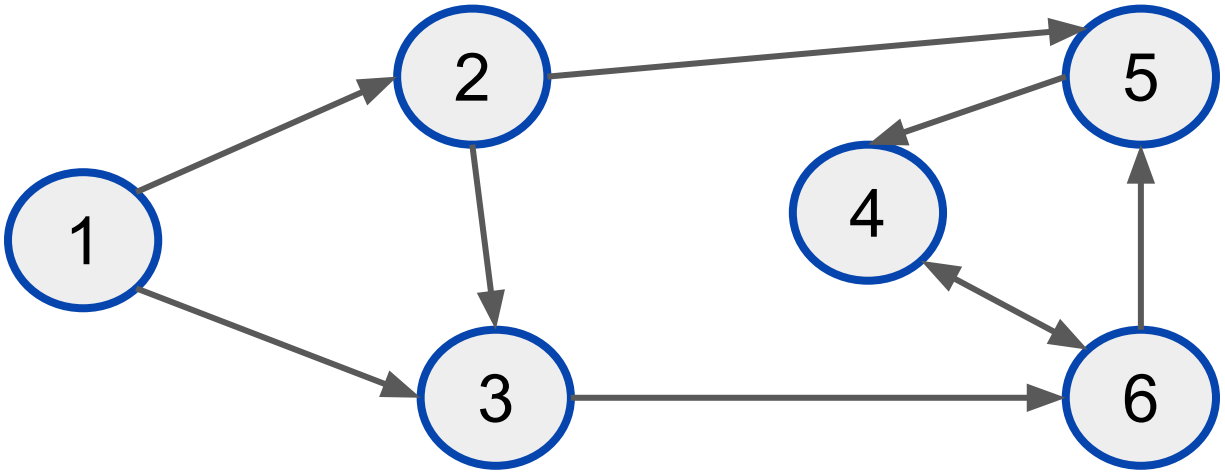
bfs(node=1, graph)



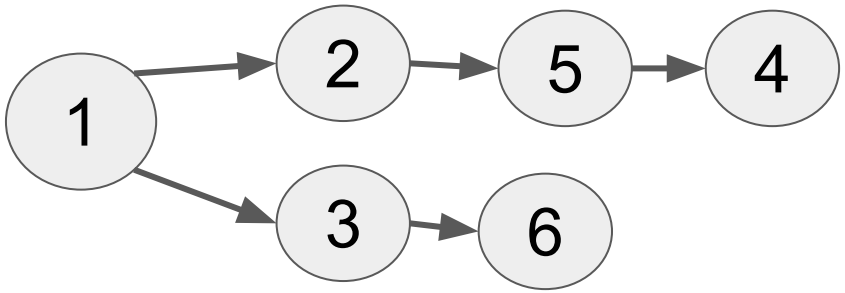
front	6	rear

Queue

Breadth-first search



bfs(node=1, graph)

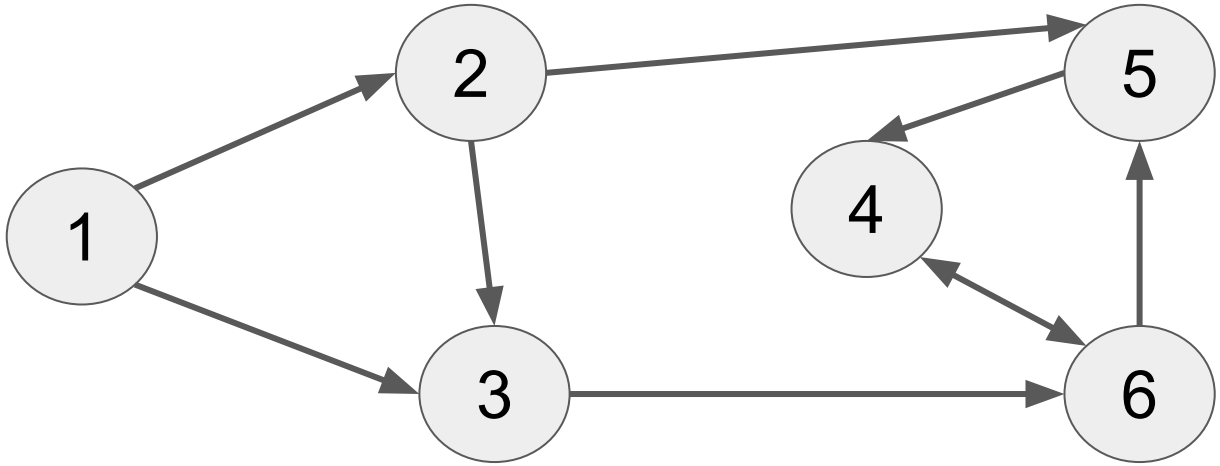


front	rear

Queue

Practice problem

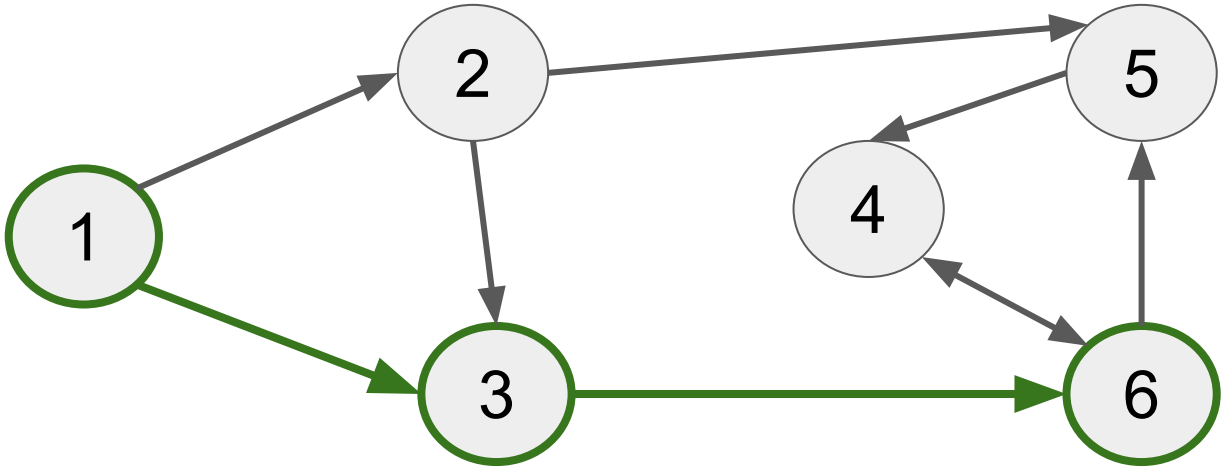
- Given a non-weighted directed graph with adjacency list:
  - Find the shortest path between from SRC to DST



bfs(node=1, graph)

Practice problem

- Given a non-weighted directed graph with adjacency list:
  - Find the shortest path between from SRC to DST



bfs(node=1, graph)