

Complex-Network Modelling and Inference

Lecture 4: Graph connectivity and traversal

Matthew Roughan

`<matthew.roughan@adelaide.edu.au>`

https://roughan.info/notes/Network_Modelling/

School of Mathematical Sciences,
University of Adelaide

January 14, 2025

Section 1

Connectivity

Connectivity

Definition

Two nodes are *connected* if a path exists between them.

Definition

A graph is *connected* if all pairs of nodes are connected.

Definition

A *strongly connected* digraph is connected in the sense above, whereas a *weakly connected* digraph is connected if we include all reverse links.

Definition

A graph is *k-edge connected* if the graph remains connected after the removal of any set of $k - 1$ edges, and *k-node connected* if the graph remains connected after the removal of any set of $k - 1$ nodes.

Definition (Cut)

A *cut* is a partition of the nodes of a graph into two subsets $C = (S, T)$.

Definition (Cut-set)

The *cut-set* of a cut $C = (S, T)$ is the set of edges

$$\{(u, v) \in E \mid u \in S, v \in T\}$$

i.e., the edges that cross the cut.

Definition (Edge Cut)

An (minimum) *edge cut* is the minimum number of edges whose removal disconnects two nodes i and j , *i.e.*, a minimal cut-set with $i \in S$ and $j \in T$.

Menger's theorem

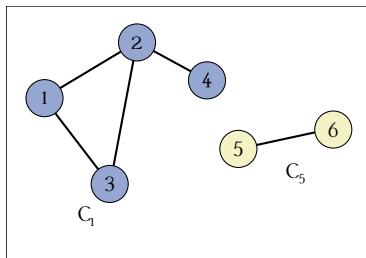
Theorem (Edge-connectivity version)

For an undirected graph G , the size of the minimum edge cut for an arbitrary pair of nodes $i \neq j$ is equal to the maximum number of edge-disjoint paths from i to j .

- Edge-disjoint means they share no common edges
- There is also a node connectivity version
- It also works for digraphs and infinite graphs
- The theorem is generalised in many optimisation algorithms: e.g., maximum flow algorithms.

Connected Components

- A *connected component* is a maximal connected subgraph



- The set of connected components $\{C_i\}$ form a *partition* of the nodes

Definition (Partition)

A *partition* is a set of covering and disjoint subsets, i.e., $\{C_i\}_{i=1}^n$ is a partition of C iff

$$\bigcup_{i=1}^n C_i = C \quad \text{and} \quad C_i \cap C_j = \phi, \quad \forall i \neq j.$$

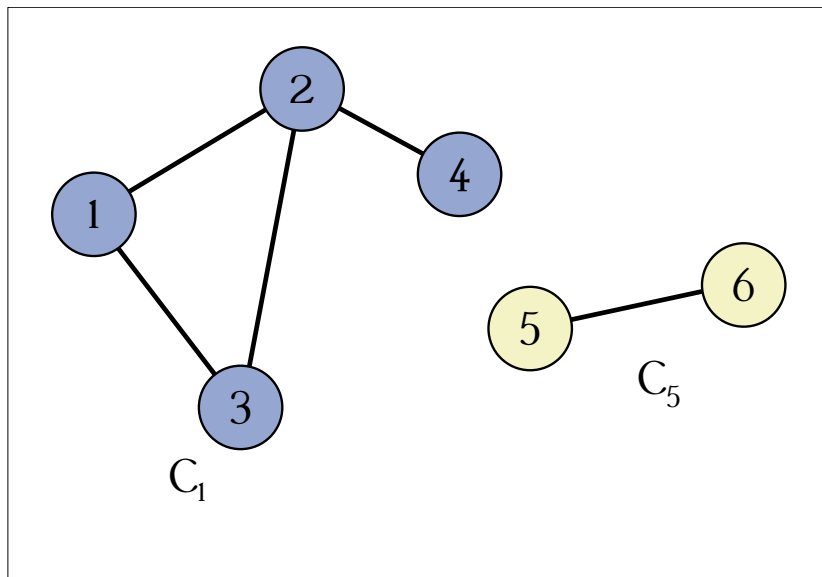
Connected Components Algorithm

Data: A Graph $G = (N, E)$

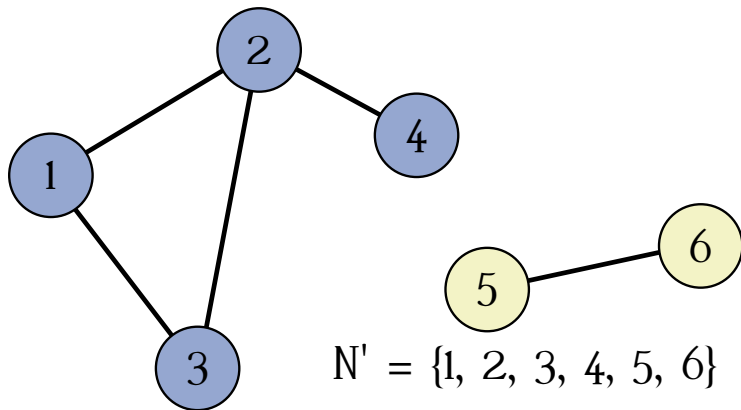
Result: A set of connected components $\{C_i\}$

```
1 Initialise  $N' = N$ ;  
2 while ( $N' \neq \phi$ ) do  
3     Choose a node  $i \in N'$  and delete it from  $N'$ ;  
4     Set  $C_i = \{i\}$  and  $L = \{(i, j) | (i, j) \in E\}$ ;  
5     while ( $L \neq \phi$ ) do  
6         Choose a link  $(k, m) \in L$ ;  
7         if  $m \notin C_i$  then  
8             add  $m$  to  $C_i$ ;  
9             delete  $m$  from  $N'$ ;  
10            add all links  $(m, l) \in E$  to  $L$ ;  
11        end  
12        delete  $(k, m)$  from  $L$ ;  
13    end  
14 end
```

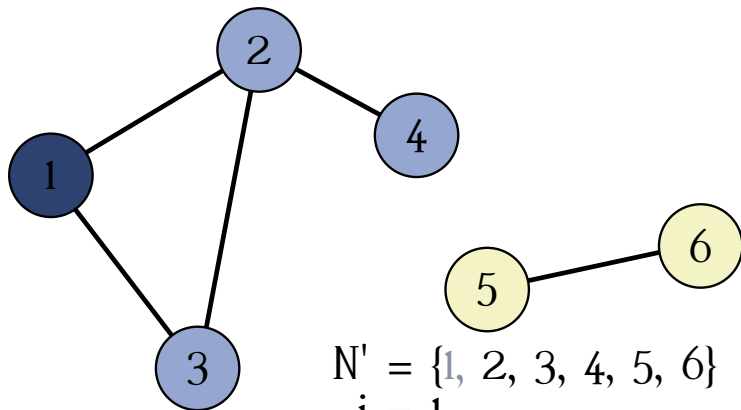
Connected Components Example



Connected Components Example



Connected Components Example



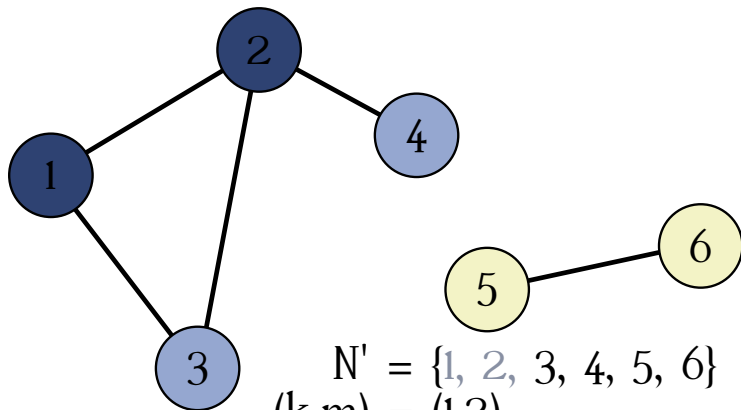
$$N' = \{1, 2, 3, 4, 5, 6\}$$

$$i = 1$$

$$C_1 = \{1\}$$

$$L = \{(1,2), (1,3)\}$$

Connected Components Example



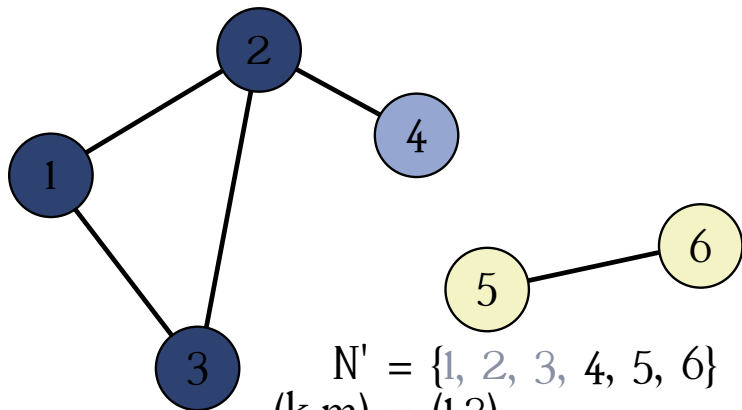
$$N' = \{1, 2, 3, 4, 5, 6\}$$

$$(k,m) = (1,2)$$

$$C_1 = \{1, 2\}$$

$$L = \{(1,2), (1,3), (2,4)\}$$

Connected Components Example



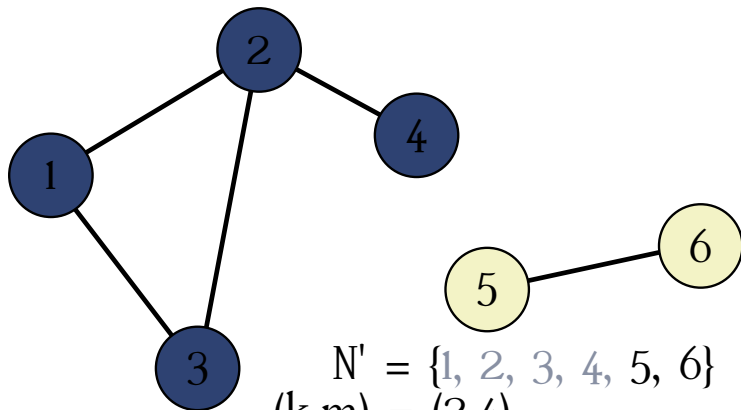
$$N' = \{1, 2, 3, 4, 5, 6\}$$

$$(k,m) = (1,3)$$

$$C_1 = \{1, 2, 3\}$$

$$L = \{(1,2), (1,3), (2,4)\}$$

Connected Components Example



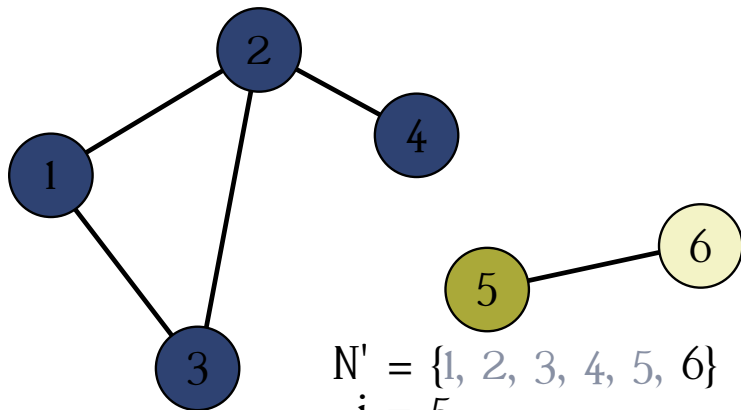
$$N' = \{1, 2, 3, 4, 5, 6\}$$

$$(k,m) = (2,4)$$

$$C_1 = \{1, 2, 3, 4\}$$

$$L = \{(1,2), (1,3), (2,4)\}$$

Connected Components Example



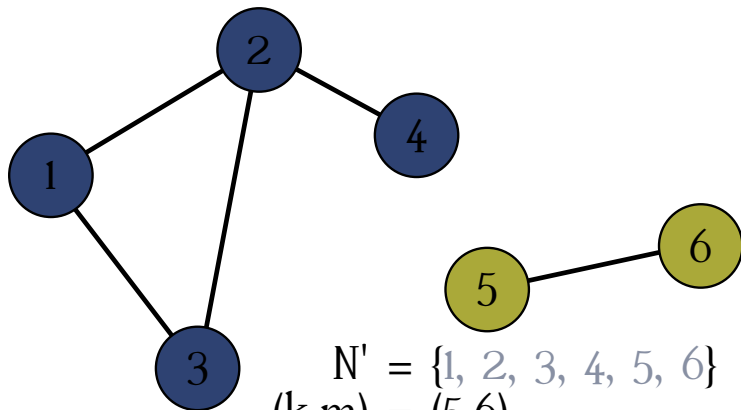
$$N' = \{1, 2, 3, 4, 5, 6\}$$

$$i = 5$$

$$C_5 = \{5\}$$

$$L = \{ (5,6) \}$$

Connected Components Example



$$\begin{aligned}N' &= \{1, 2, 3, 4, 5, 6\} \\(k,m) &= (5,6) \\C_5 &= \{5, 6\} \\L &= \{ (5,6) \}\end{aligned}$$

Application 1

Key requirements for critical infrastructure networks (e.g., Internet, Water, Power, ...)

- Reliability

Application 1

Key requirements for critical infrastructure networks (e.g., Internet, Water, Power, ...)

- Reliability
- Reliability

Application 1

Key requirements for critical infrastructure networks (e.g., Internet, Water, Power, ...)

- Reliability
- Reliability
- Reliability

Application 1

Key requirements for critical infrastructure networks (e.g., Internet, Water, Power, ...)

- Reliability
- Reliability
- Reliability
- Cost

Application 1

Key requirements for critical infrastructure networks (e.g., Internet, Water, Power, ...)

- Reliability
- Reliability
- Reliability
- Cost
- Performance

Application 1

Key requirements for critical infrastructure networks (e.g., Internet, Water, Power, ...)

- Reliability
- Reliability
- Reliability
- Cost
- Performance
- Reliability

Application 1

The simplest definition of “reliability” used in networks is some variant of k -connectedness

- The particular variant depends on the failure modes of the network
 - ▶ do the nodes fail, or the edges (or both)?
- Leads to network designs with redundancy
 - ▶ not necessarily k -fold redundancy
- Is this a good enough definition of reliability

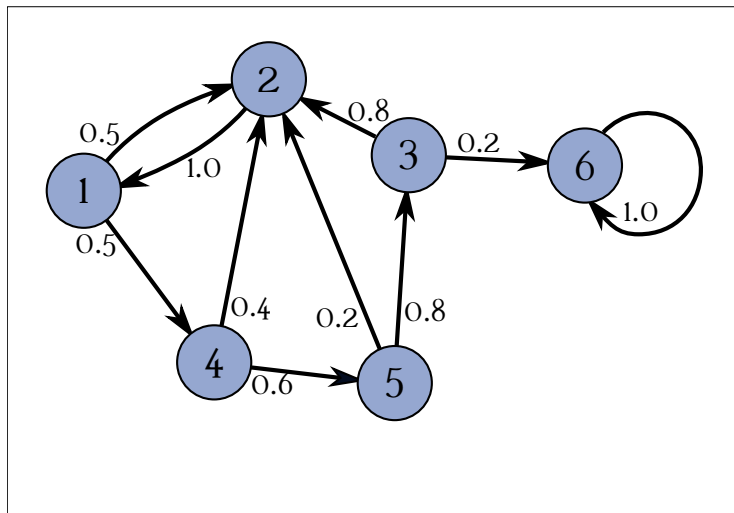
Application 2

Markov chains probability transition matrix

$$P = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.8 & 0.4 & 0.2 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.8 & 0.0 \\ 0.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.6 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.2 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

Application 2

Markov chains are described by a directed graph with self-loops, e.g.,



The transition matrix is just a weighted adjacency matrix.

Application 2

First step of studying a Markov chain is to check its properties

Definition

State i is *accessible* from state j if it is possible to get from i to j .

Accessible = a path from i to j exists

Definition

A Markov chain is *irreducible* if it is possible to get to any state from any state.

Irreducibility = strong connectivity of the graph

Definition

A *communicating class* is a maximal set of mutually accessible states.

Communicating class = connected component

Section 2

Graph Traversal

- Connected components as described above has a little vagueness
 - ▶ when I say “choose” how do you choose?
- It's an example of *graph traversal*
 - ▶ where we want to visit each node of a graph (at least once)
 - ▶ ordered nodes by connectivity
- Traversals used for lots of algorithms
 - ▶ could be to search for an element
 - ▶ or to calculate a value for each node

Maybe you don't have the whole graph stored in memory, but have to read bits, e.g., traversing Facebook graph

- There are two main strategies
 - ▶ Depth-First Search
 - ▶ Breadth-First Search
- For the sake of simplicity, we will assume graphs are connected
- Easiest to understand in neighbour-list representation

Depth-First Search

Visit a neighbour's children before you visit the next neighbour

```
1 Function DFS( $G, i$ );
   Input: A Graph  $G = (N, E)$ , and start node  $i \in N$ 
2 label  $i$  as explored;
3 forall  $j \in \text{neighbourhood}\{i\}$  do
4   |   if  $j$  is unexplored then
5   |   |   DFS( $G, j$ );
6   |   end
7 end
```

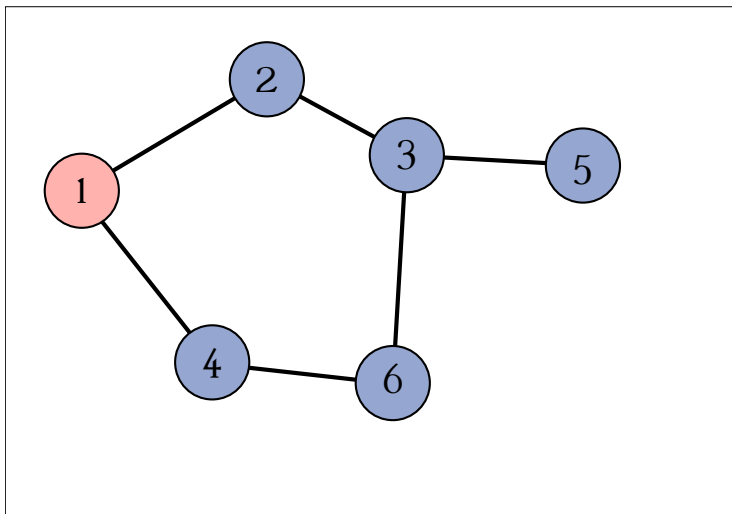
- We could make this faster by avoiding edges going backwards.
- At the moment the algorithm doesn't do anything
 - ▶ a *search* also checks something about the node, and returns the first one that checks out
 - ▶ but we might also do some sort of update
 - ▶ or use to find a connected component ...

Breadth-First Search

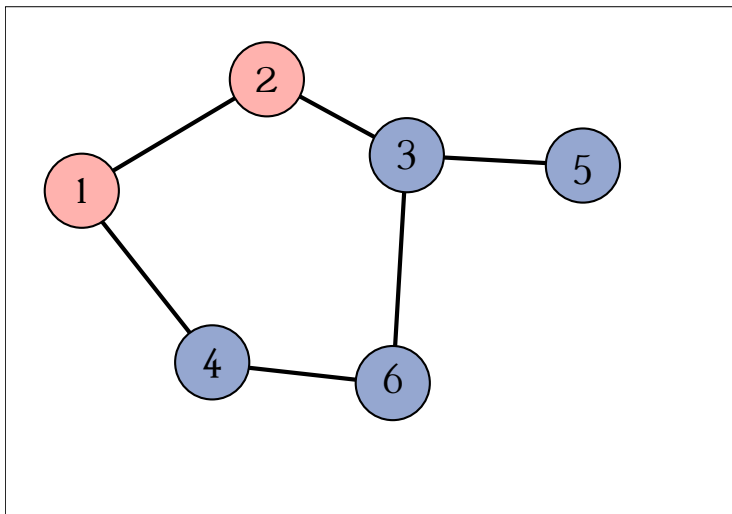
Visit all neighbours before you visit their children

```
1 Function BFS( $G, i$ );
   Input: A Graph  $G = (N, E)$ , and start node  $i \in N$ 
2 label  $i$  as explored;
3 create queue  $Q$ ;
4 put  $i$  on  $Q$ ;
5 while  $Q$  not empty do
6     take  $j$  off the front of  $Q$ ;
7     forall  $k \in \text{neighbourhood}\{j\}$  do
8         if  $k$  is unexplored then
9             label  $k$  as explored;
10            put  $k$  on  $Q$ ;
11        end
12    end
13 end
```

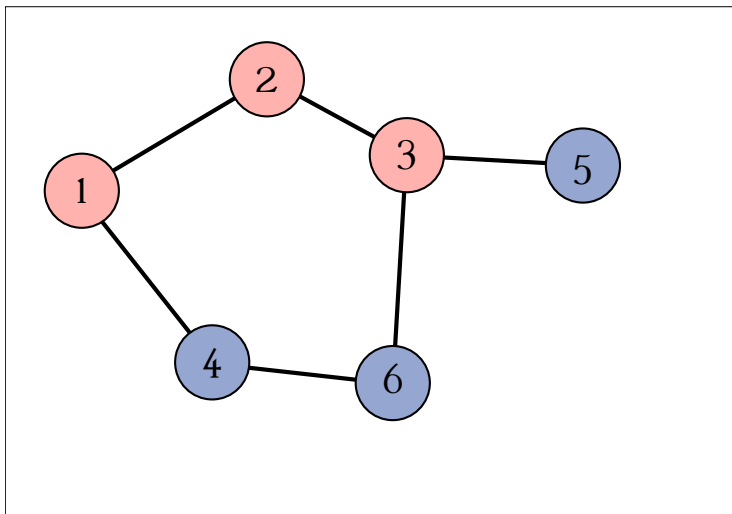
Depth-First Search Example



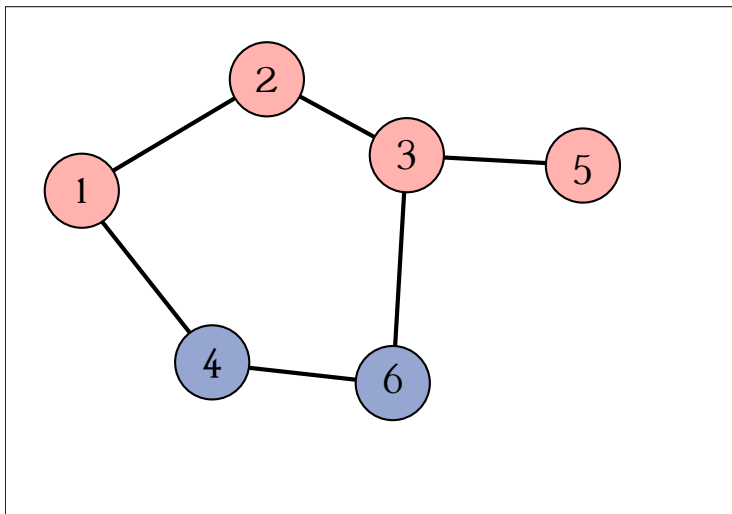
Depth-First Search Example



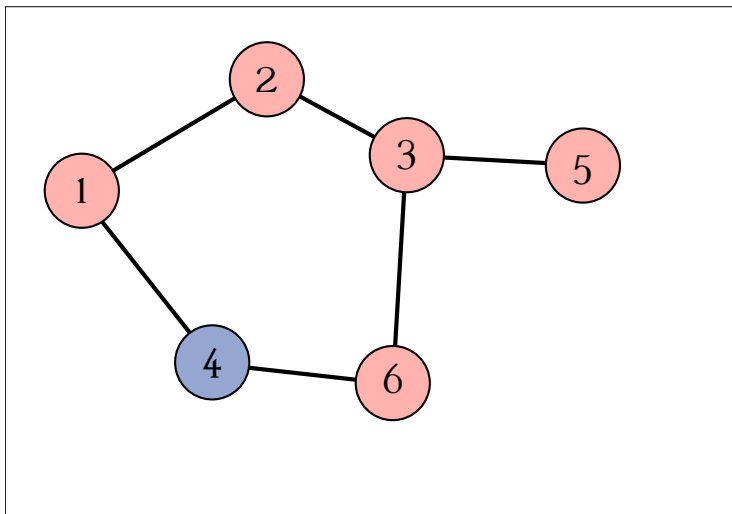
Depth-First Search Example



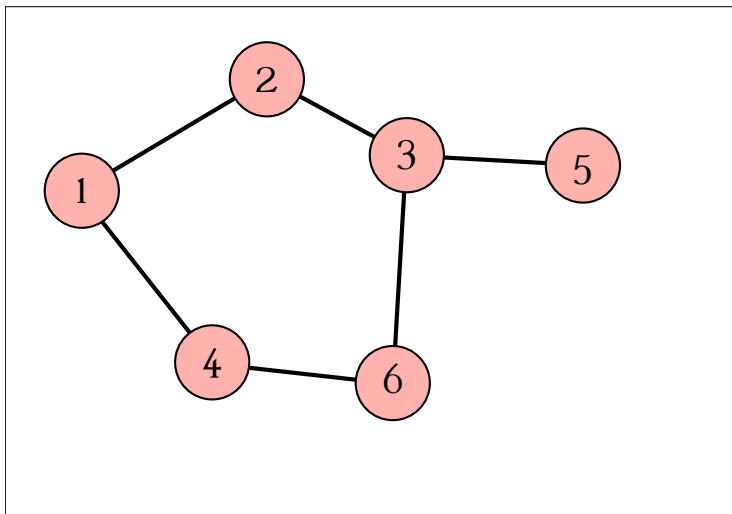
Depth-First Search Example



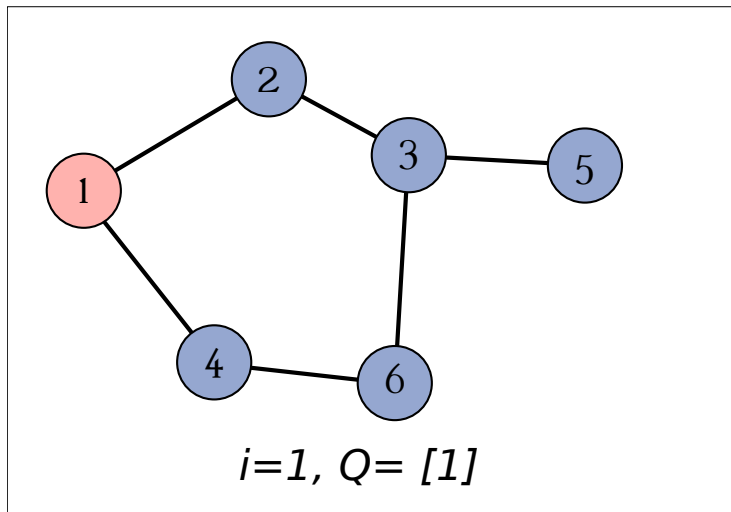
Depth-First Search Example



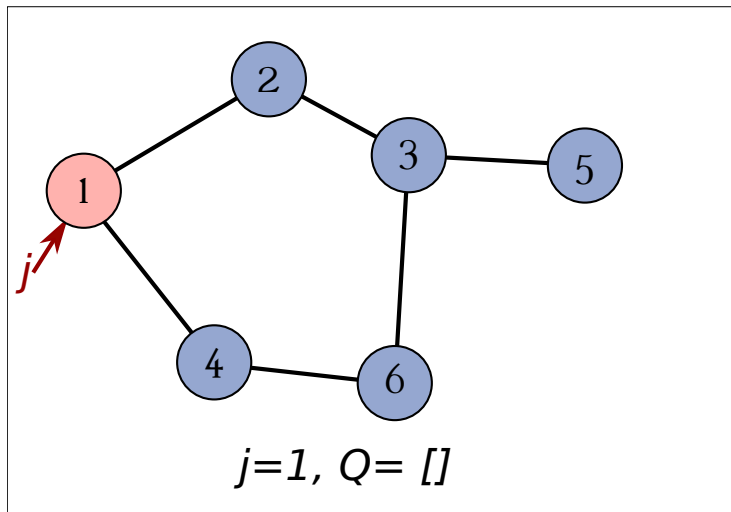
Depth-First Search Example



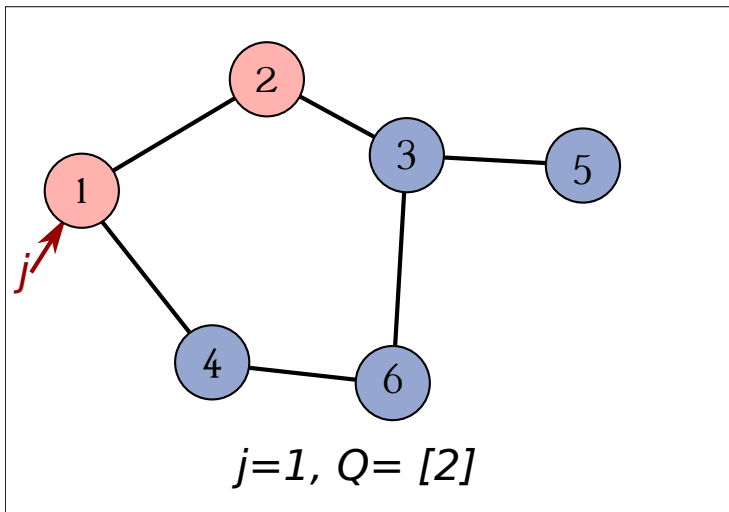
Breadth-First Search Example



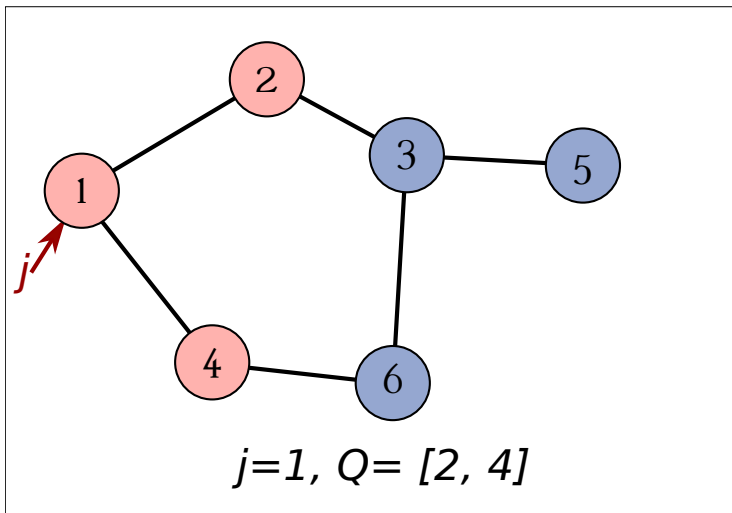
Breadth-First Search Example



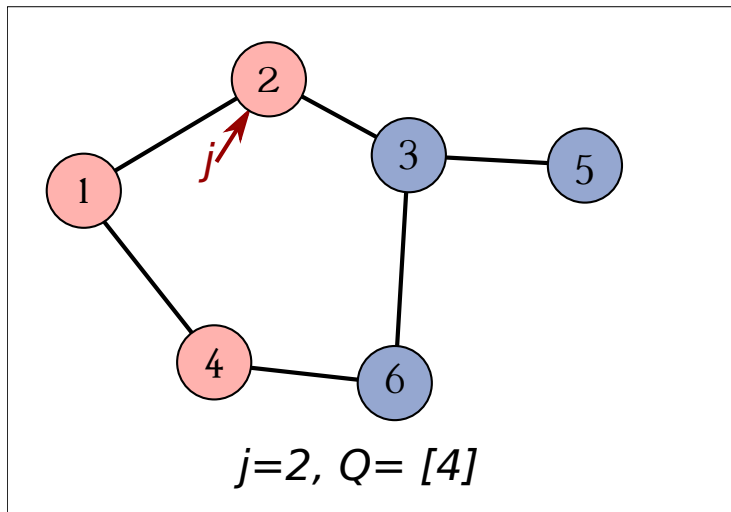
Breadth-First Search Example



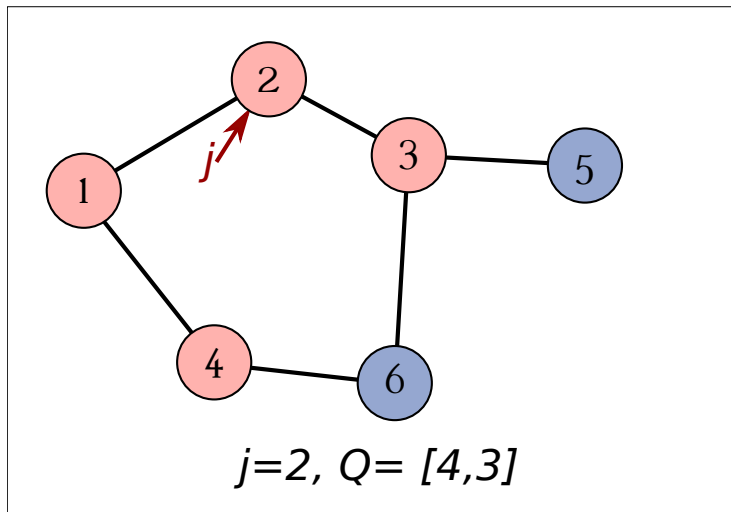
Breadth-First Search Example



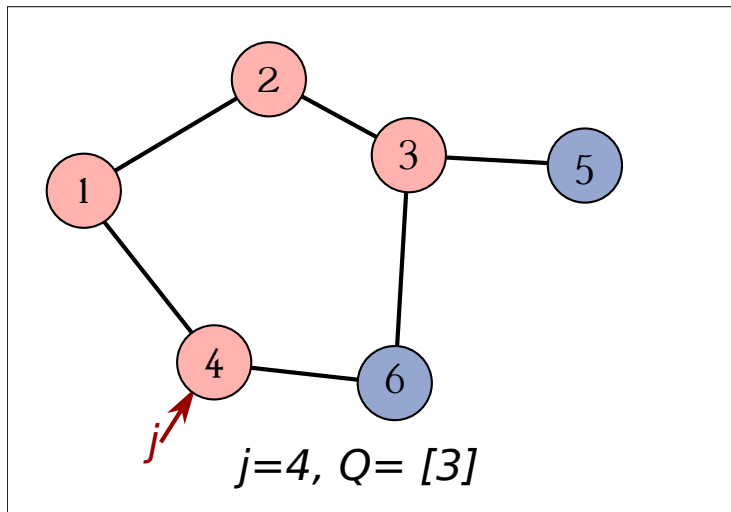
Breadth-First Search Example



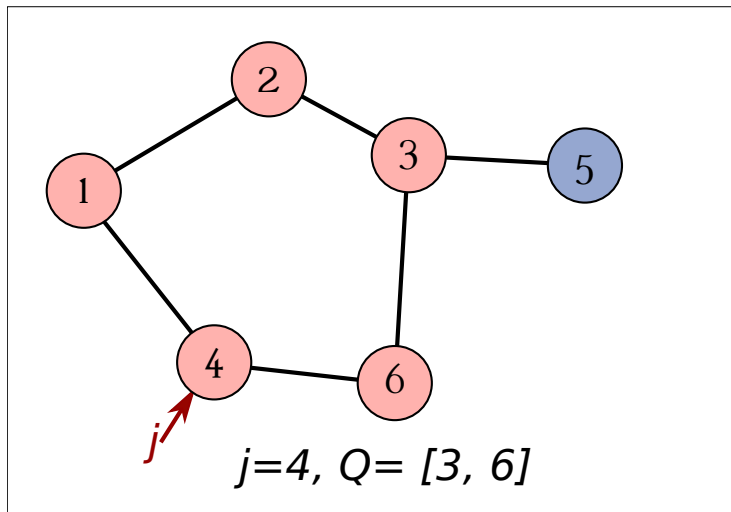
Breadth-First Search Example



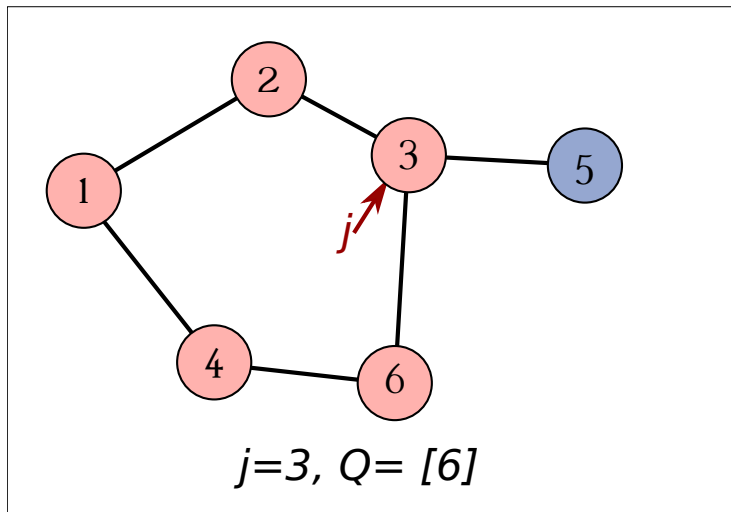
Breadth-First Search Example



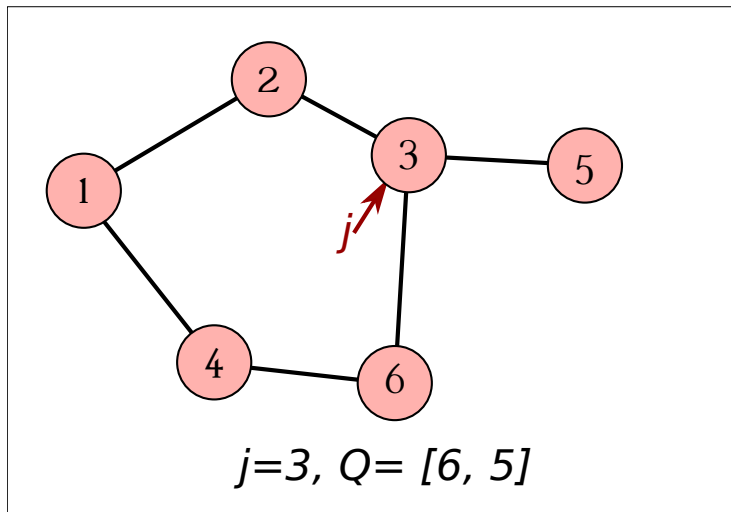
Breadth-First Search Example



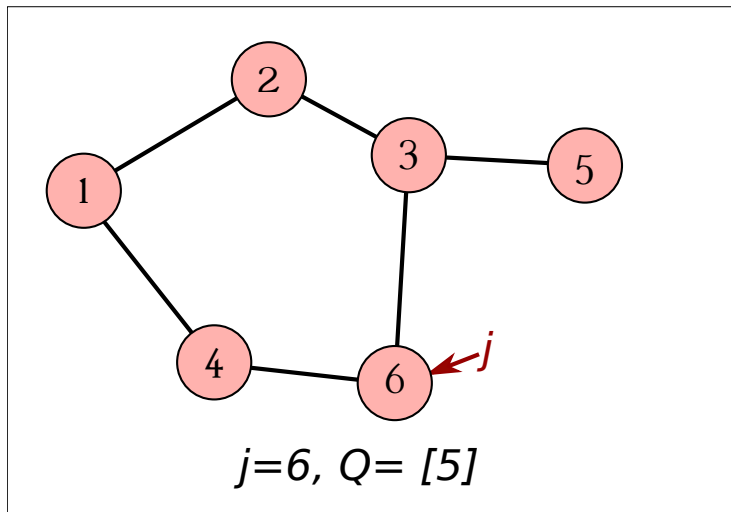
Breadth-First Search Example



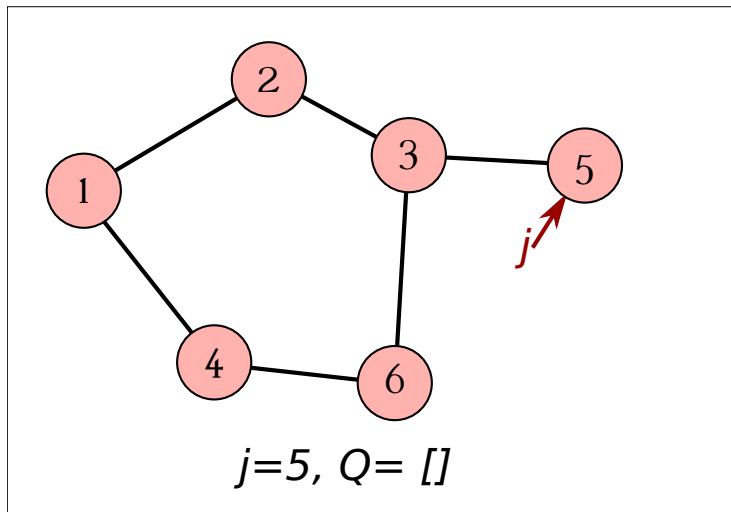
Breadth-First Search Example



Breadth-First Search Example



Breadth-First Search Example



Further reading I