Graphs

Department of Computer Science

HCMC University of Technology, Viet Nam

05, 2020

Outline

Outline

Introduction

Terminology

Representations of graphs

Graph Traversal

Graph Problems

Introduction

Graph

- Each node may have *multiple predecessors* as well as *multiple successors*.

Introduction

Graph

- Each node may have *multiple predecessors* as well as *multiple successors*.
- Graphs are used to represent *complex networks* and solve related problems.

Applications

- Modeling connectivity in computer and communication networks.

Applications

- Modeling connectivity in computer and communication networks.
- Representing a map as a set of locations with distances between locations

Applications

- Modeling connectivity in computer and communication networks.
- Representing a map as a set of locations with distances between locations
- Modeling flow capacities in transportation networks.

Applications

- Modeling connectivity in computer and communication networks.
- Representing a map as a set of locations with distances between locations
- Modeling flow capacities in transportation networks.
- Finding a path from a starting condition to a goal condition

Applications

- Modeling connectivity in computer and communication networks.
- Representing a map as a set of locations with distances between locations
- Modeling flow capacities in transportation networks.
- Finding a path from a starting condition to a goal condition
- Modeling computer algorithms, showing transitions from one program state to another

Applications

- Modeling connectivity in computer and communication networks.
- Representing a map as a set of locations with distances between locations
- Modeling flow capacities in transportation networks.
- Finding a path from a starting condition to a goal condition
- Modeling computer algorithms, showing transitions from one program state to another
- Finding an acceptable order for finishing subtasks in a complex activity, such as constructing large buildings.

Applications

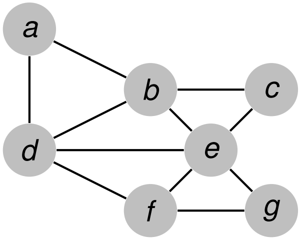
- Modeling connectivity in computer and communication networks.
- Representing a map as a set of locations with distances between locations
- Modeling flow capacities in transportation networks.
- Finding a path from a starting condition to a goal condition
- Modeling computer algorithms, showing transitions from one program state to another
- Finding an acceptable order for finishing subtasks in a complex activity, such as constructing large buildings.
- Modeling relationships such as family trees, business or military organizations, and scientific taxonomies.

Applications

- Modeling connectivity in computer and communication networks.
- Representing a map as a set of locations with distances between locations
- Modeling flow capacities in transportation networks.
- Finding a path from a starting condition to a goal condition
- Modeling computer algorithms, showing transitions from one program state to another
- Finding an acceptable order for finishing subtasks in a complex activity, such as constructing large buildings.
- Modeling relationships such as family trees, business or military organizations, and scientific taxonomies.
- ...

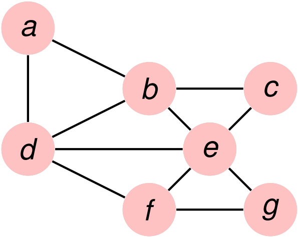
Outline

Terminology



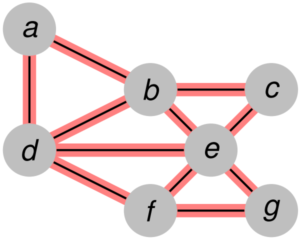
Terminology

- Vertex (vertices)



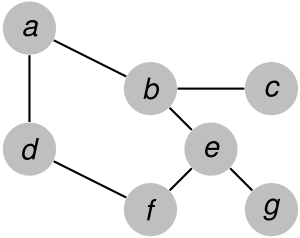
Terminology

- Vertex (vertices)
- Edge



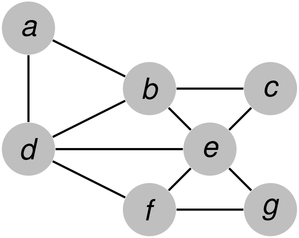
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete



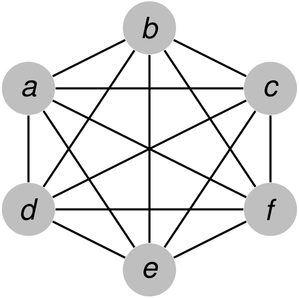
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete



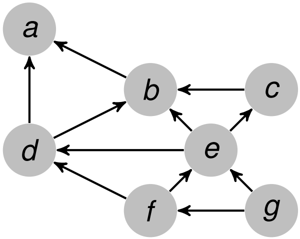
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete



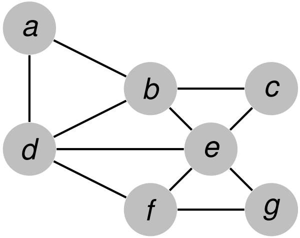
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected



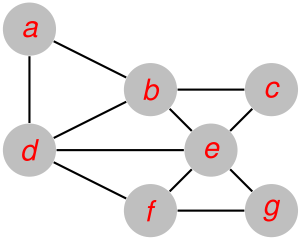
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected



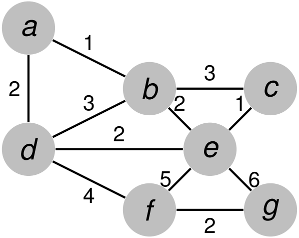
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted



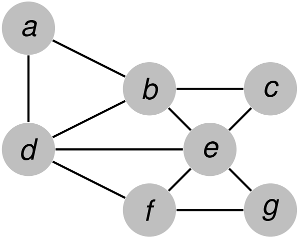
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted



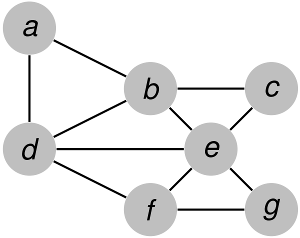
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors



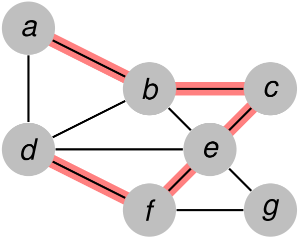
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors



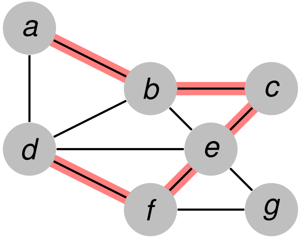
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors
- Path - Simple - Length - Cycle



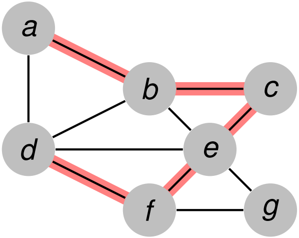
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors
- Path - Simple - Length - Cycle



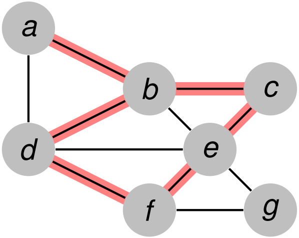
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors
- Path - Simple - Length - Cycle



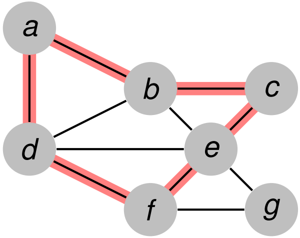
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors
- Path - Simple - Length - Cycle



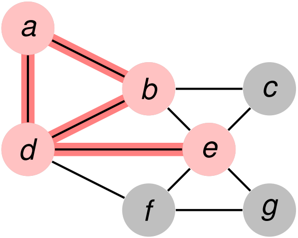
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors
- Path - Simple - Length - Cycle



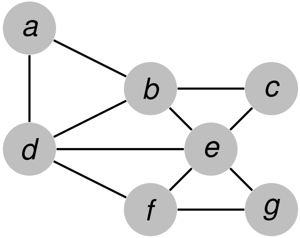
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors
- Path - Simple - Length - Cycle
- Subgraph



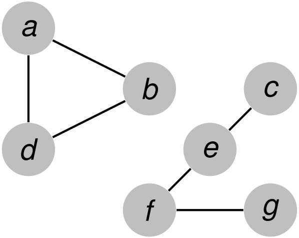
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors
- Path - Simple - Length - Cycle
- Subgraph
- Connected - Connected Components



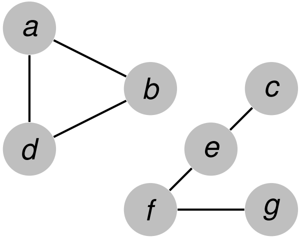
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors
- Path - Simple - Length - Cycle
- Subgraph
- Connected - Connected Components



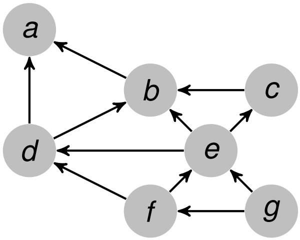
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors
- Path - Simple - Length - Cycle
- Subgraph
- Connected - Connected Components
- Acyclic - Acyclic Directed Graph



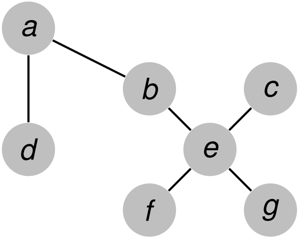
Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors
- Path - Simple - Length - Cycle
- Subgraph
- Connected - Connected Components
- Acyclic - Acyclic Directed Graph



Terminology

- Vertex (vertices)
- Edge
- Sparse - Dense - Complete
- Directed - Undirected
- Labeled - Weighted
- Adjacent vertices - Neighbors
- Path - Simple - Length - Cycle
- Subgraph
- Connected - Connected Components
- Acyclic - Acyclic Directed Graph
- Free tree



Outline

Introduction

Terminology

Representations of graphs

Graph Traversal

Graph Problems

Representations of graphs

- Adjacency matrix

Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges

Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros

Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros
    - Suitable for dense graph

Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros
    - Suitable for dense graph
    - No overhead for pointers

Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros
    - Suitable for dense graph
    - No overhead for pointers
  - Cons

Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros
    - Suitable for dense graph
    - No overhead for pointers
  - Cons
    - Waste memory for sparse graph

Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros
    - Suitable for dense graph
    - No overhead for pointers
  - Cons
    - Waste memory for sparse graph
    - Higher asymptotic cost

Representations of graphs

- Adjacency matrix
    - Use a matrix for vertices and edges
    - Pros
        - Suitable for dense graph
        - No overhead for pointers
    - Cons
        - Waste memory for sparse graph
        - Higher asymptotic cost
- Adjacency list

Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros
    - Suitable for dense graph
    - No overhead for pointers
  - Cons
    - Waste memory for sparse graph
    - Higher asymptotic cost
- Adjacency list
  - Use linked lists for vertices ans edges

Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros
    - Suitable for dense graph
    - No overhead for pointers
  - Cons
    - Waste memory for sparse graph
    - Higher asymptotic cost
- Adjacency list
  - Use linked lists for vertices ans edges
  - Pros

Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros
    - Suitable for dense graph
    - No overhead for pointers
  - Cons
    - Waste memory for sparse graph
    - Higher asymptotic cost
- Adjacency list
  - Use linked lists for vertices ans edges
  - Pros
    - $O(|V|+|E|)$ for space

Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros
    - Suitable for dense graph
    - No overhead for pointers
  - Cons
    - Waste memory for sparse graph
    - Higher asymptotic cost
- Adjacency list
  - Use linked lists for vertices ans edges
  - Pros
    - O(|V|+|E|) for space
    - Suitable for sparse graph

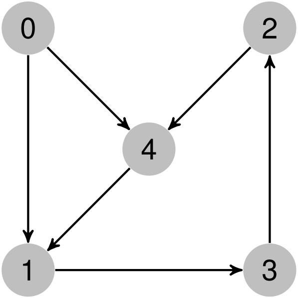
Representations of graphs

- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros
    - Suitable for dense graph
    - No overhead for pointers
  - Cons
    - Waste memory for sparse graph
    - Higher asymptotic cost
- Adjacency list
  - Use linked lists for vertices ans edges
  - Pros
    - O(|V|+|E|) for space
    - Suitable for sparse graph
  - Cons

Representations of graphs

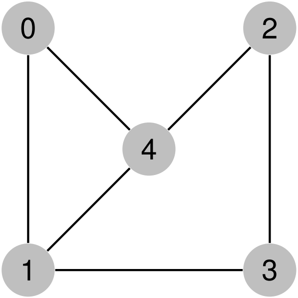
- Adjacency matrix
  - Use a matrix for vertices and edges
  - Pros
    - Suitable for dense graph
    - No overhead for pointers
  - Cons
    - Waste memory for sparse graph
    - Higher asymptotic cost
- Adjacency list
  - Use linked lists for vertices ans edges
  - Pros
    - $O(|V|+|E|)$ for space
    - Suitable for sparse graph
  - Cons
    - Overhead for pointers

Adjacency Matrix Representations of Graph



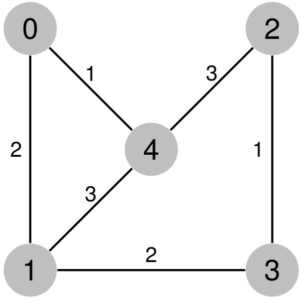
	0	1	2	3	4
0		1			1
1				1	
2					1
3			1		
4		1			

Adjacency Matrix Representations of Graph



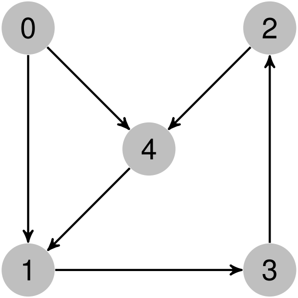
	0	1	2	3	4
0		1			1
1	1			1	1
2				1	1
3		1	1		
4	1	1	1		

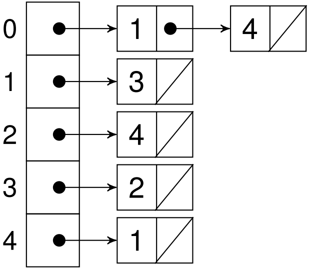
Adjacency Matrix Representations of Graph



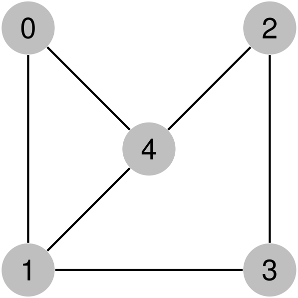
	0	1	2	3	4
0		2			1
1	2			2	3
2				1	3
3		2	1		
4	1	3	3		

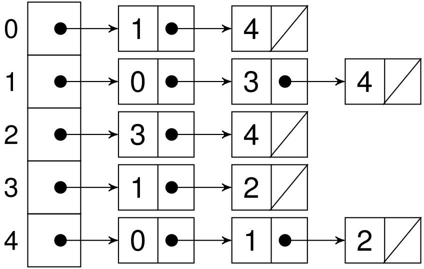
Adjacency List Representation of Graph





Adjacency List Representation of Graph





Outline

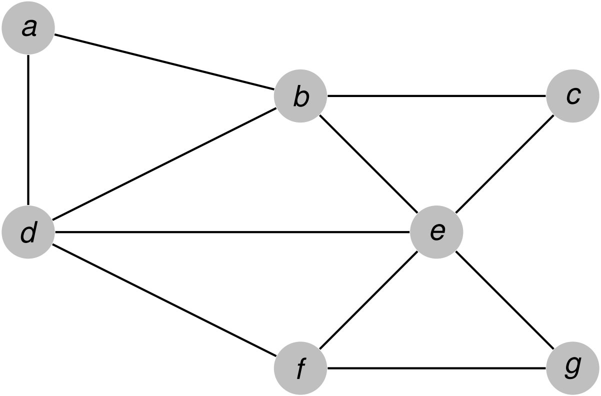
Graph Traversal

- Depth-first search (DFS): all of a vertex's **descendents** are processed **before** an **adjacent** vertex

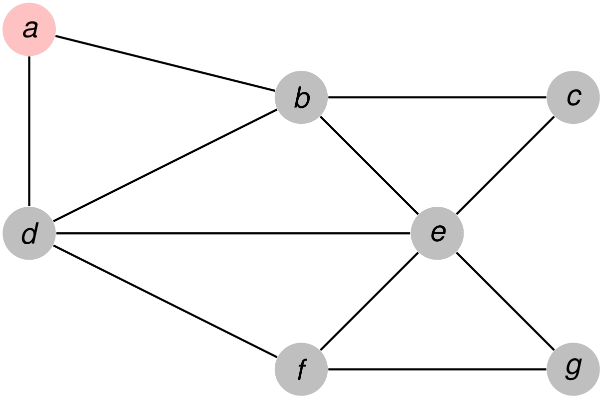
Graph Traversal

- Depth-first search (DFS): all of a vertex's **descendents** are processed **before** an **adjacent** vertex
- Breadth-first search (BFS): all **adjacent** vertices of a vertex are processed **before** the **descendents** of the vertex

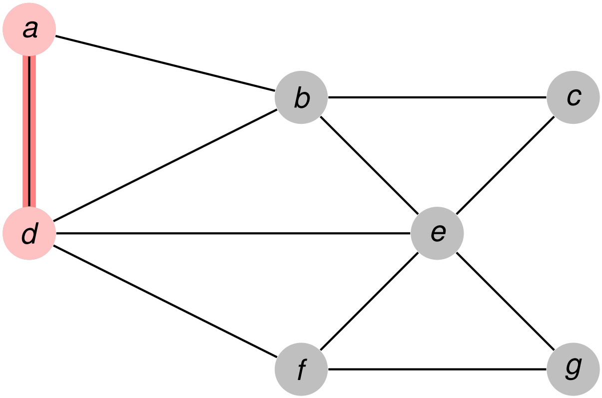
Depth-first search



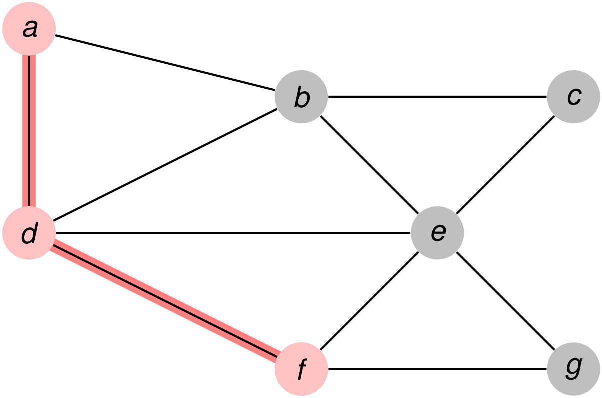
Depth-first search



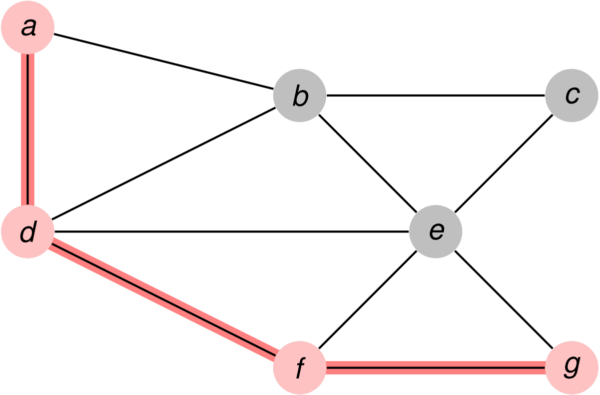
Depth-first search



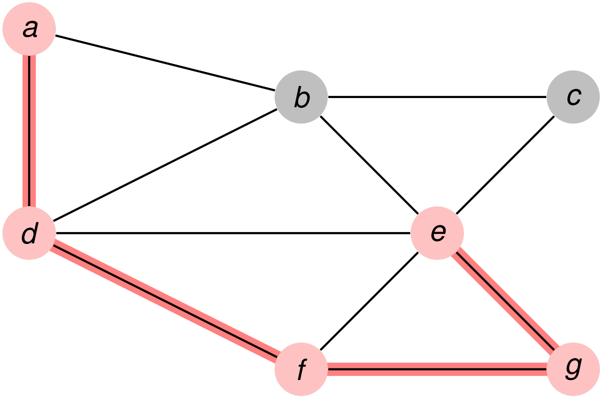
Depth-first search



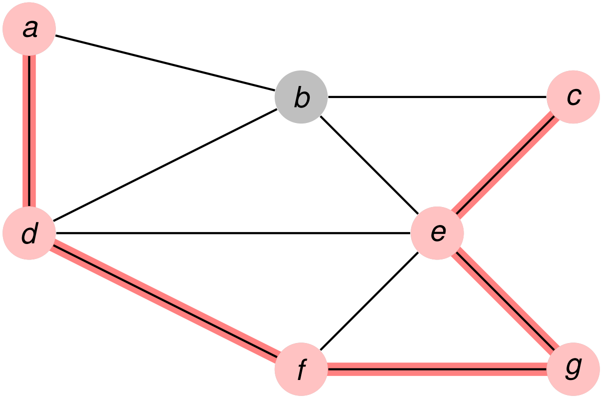
Depth-first search



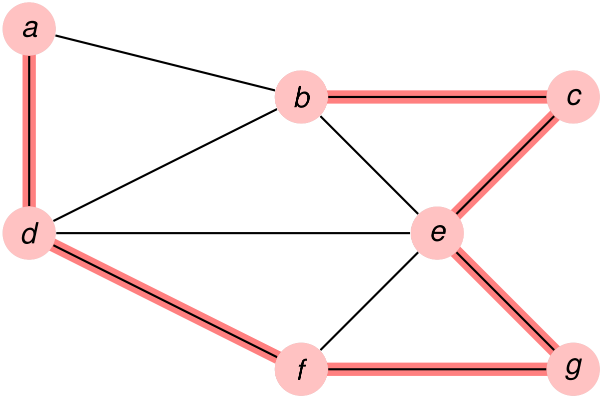
Depth-first search



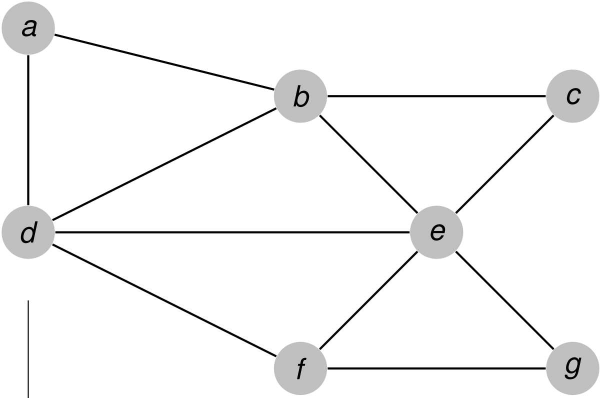
Depth-first search

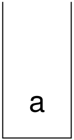


Depth-first search

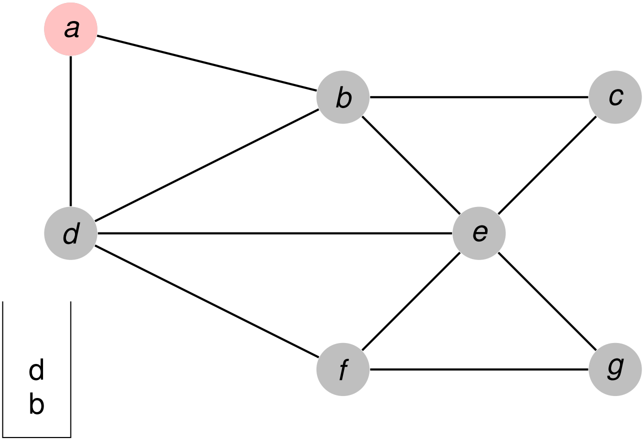


Depth-first search implementation

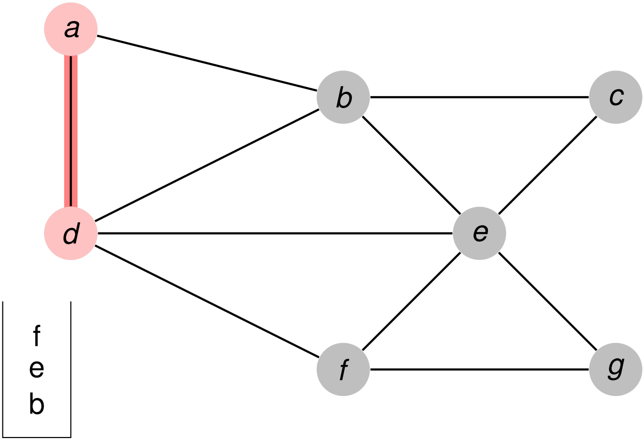




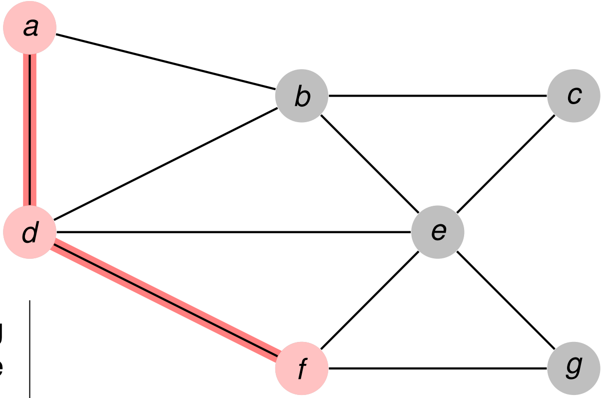
Depth-first search implementation

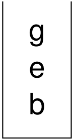


Depth-first search implementation

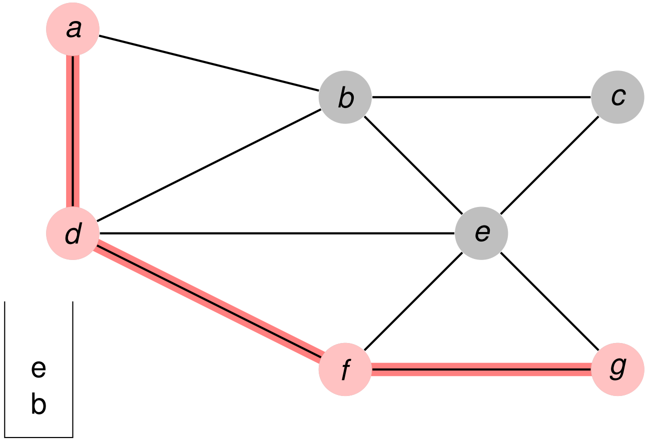


Depth-first search implementation

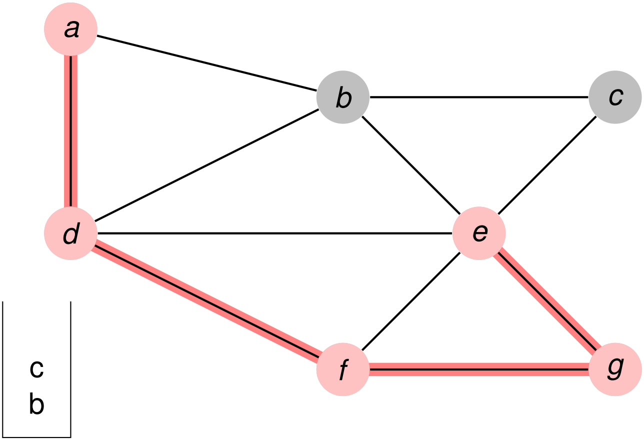




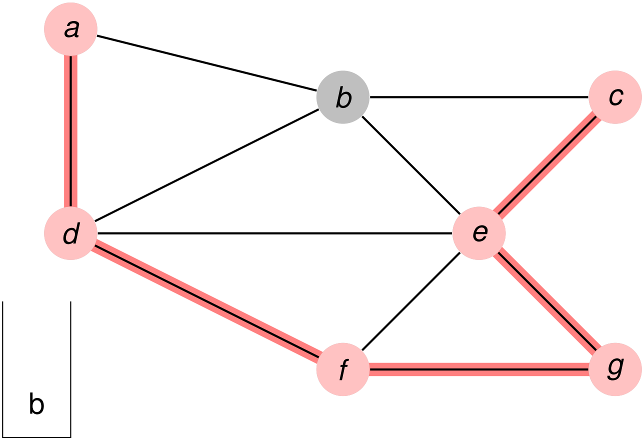
Depth-first search implementation



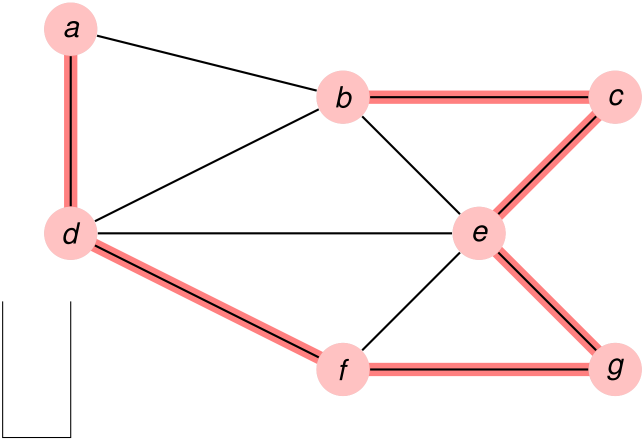
Depth-first search implementation



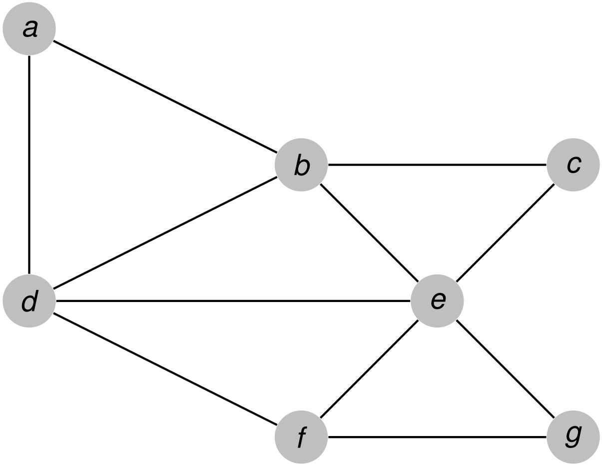
Depth-first search implementation



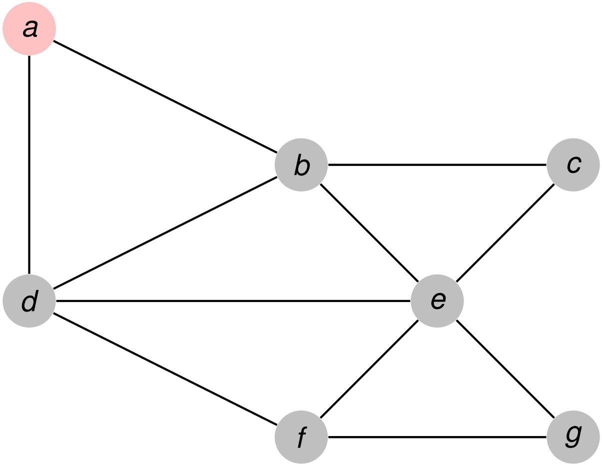
Depth-first search implementation



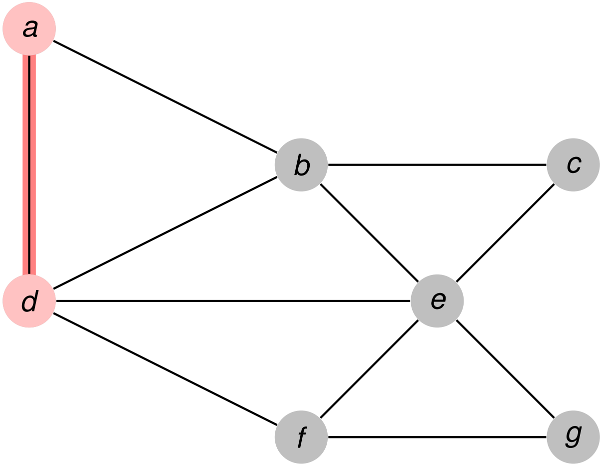
Breadth-first search



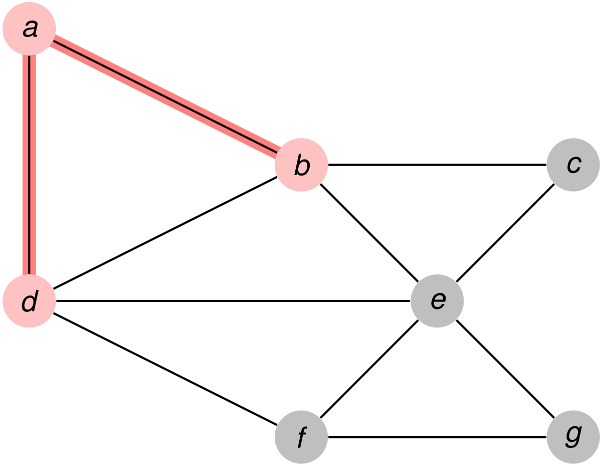
Breadth-first search



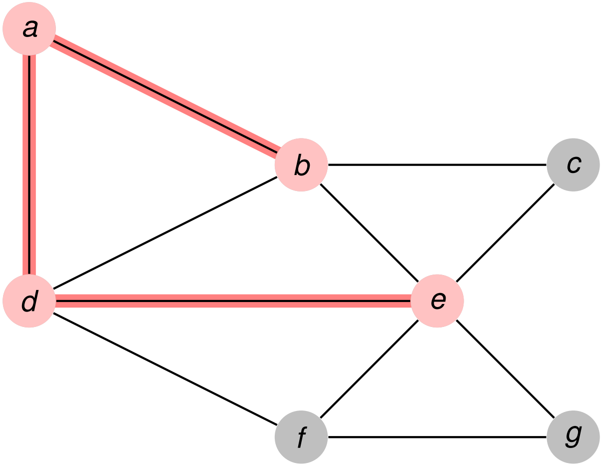
Breadth-first search



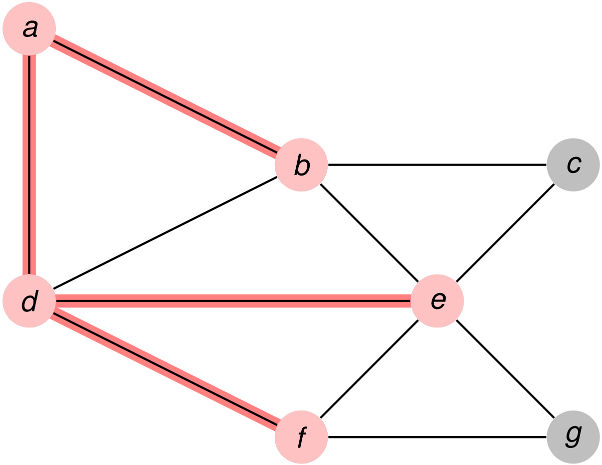
Breadth-first search



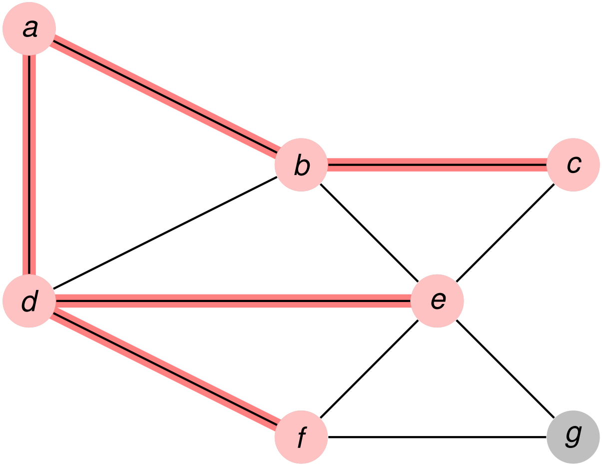
Breadth-first search



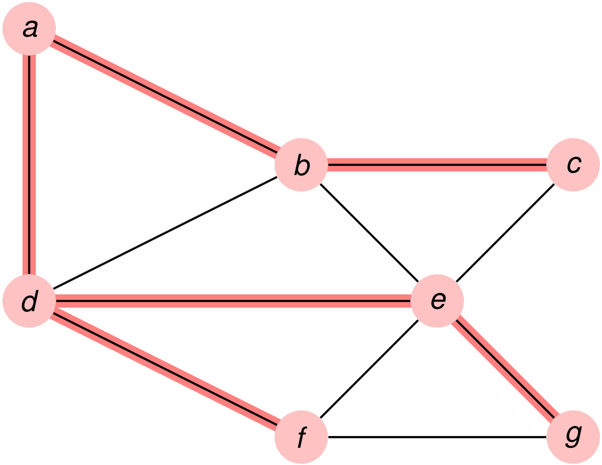
Breadth-first search



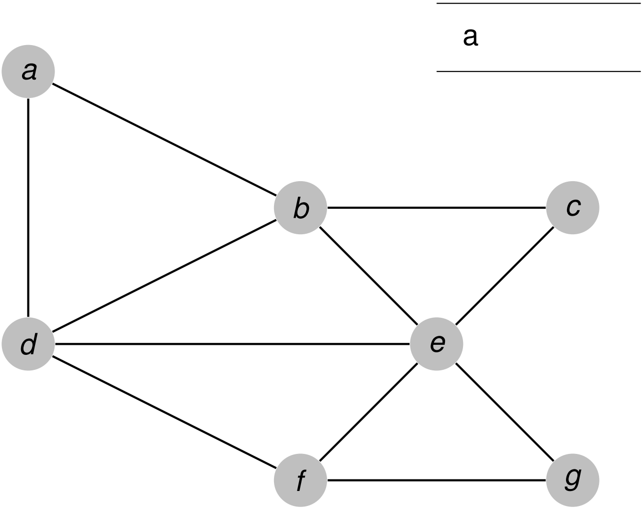
Breadth-first search



Breadth-first search

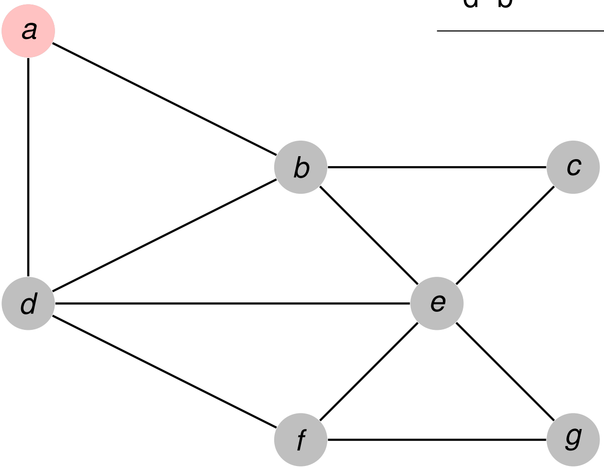


Breadth-first search implementation

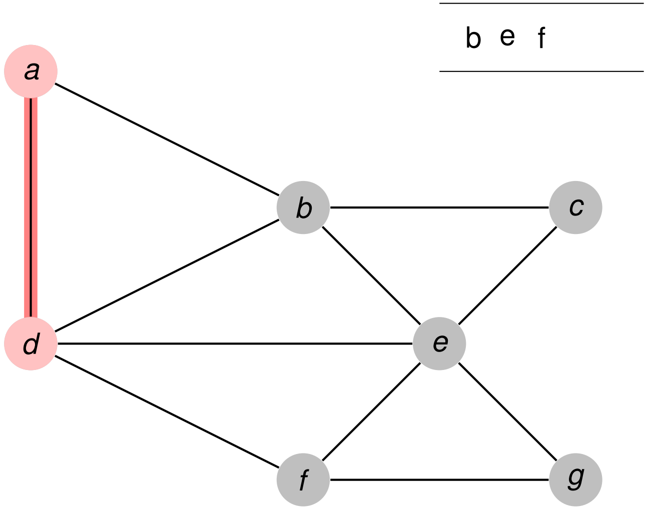


Breadth-first search implementation

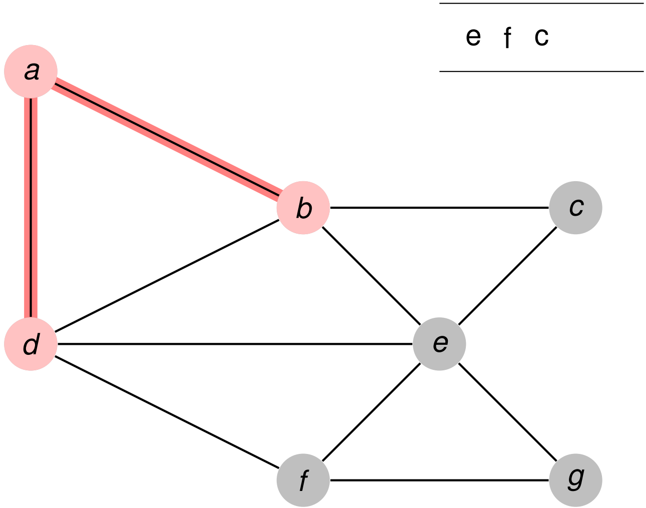
d b



Breadth-first search implementation

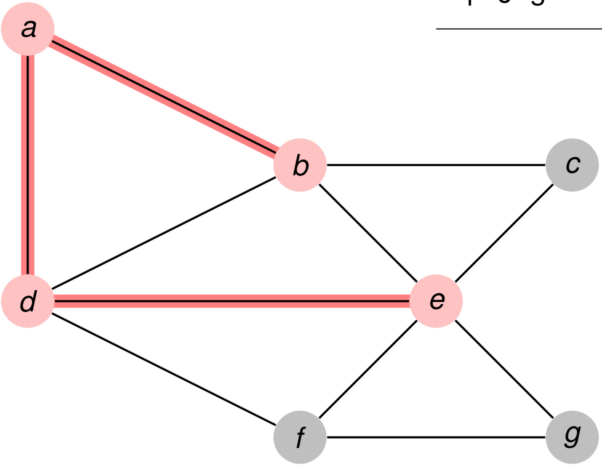


Breadth-first search implementation



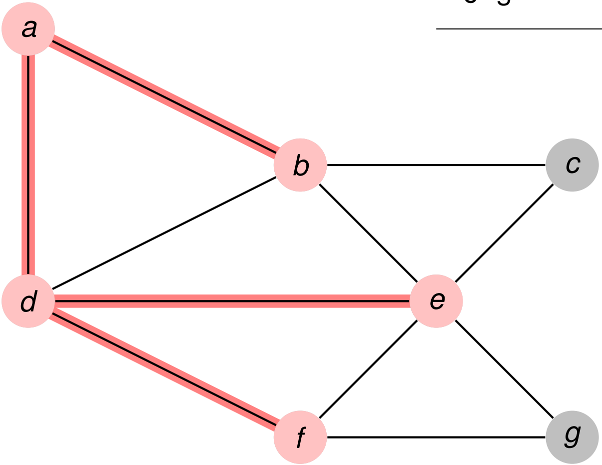
Breadth-first search implementation

f c g

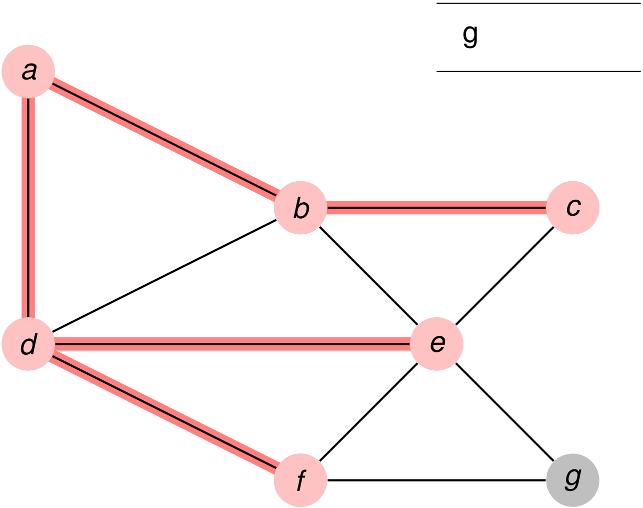


Breadth-first search implementation

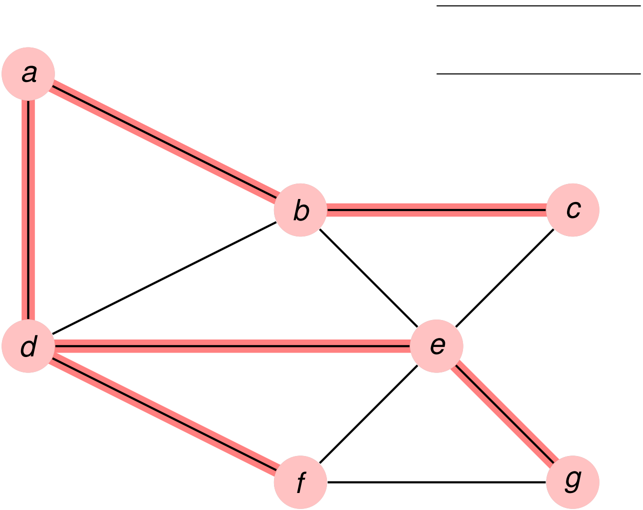
c g



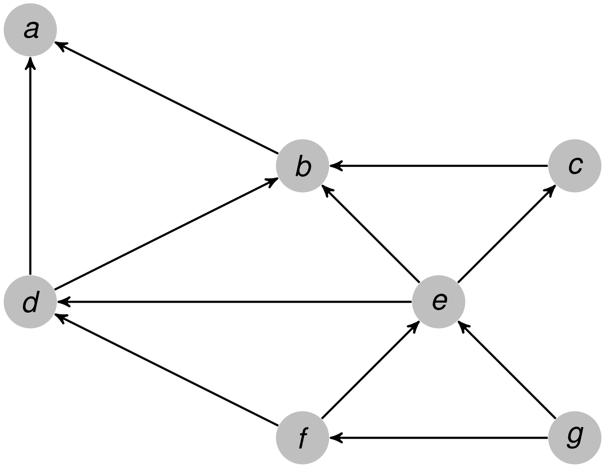
Breadth-first search implementation



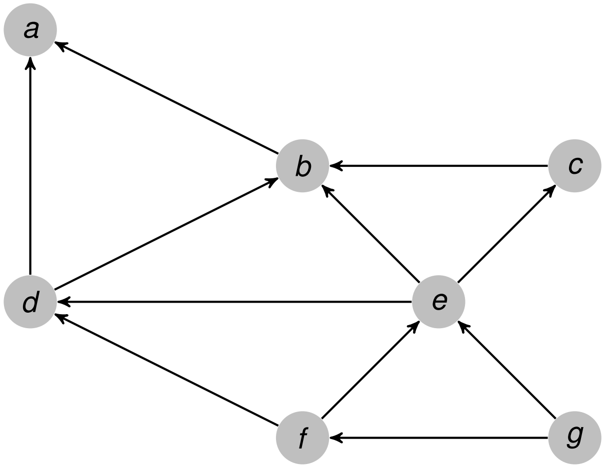
Breadth-first search implementation



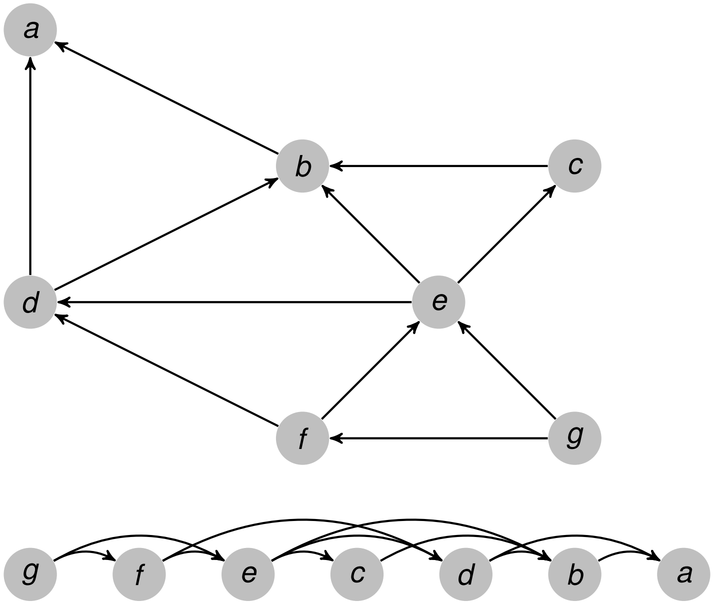
Topological Sort



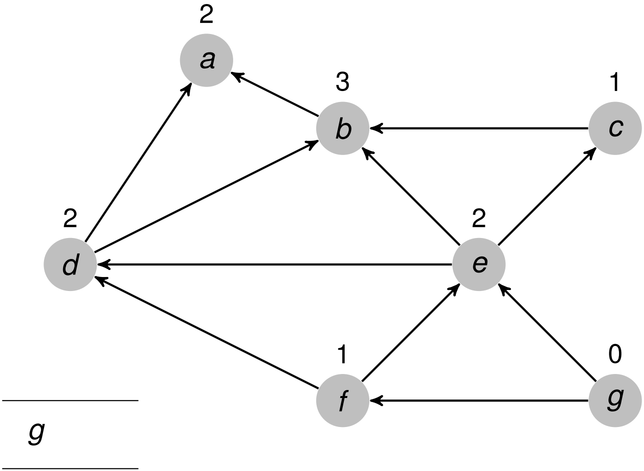
Topological Sort



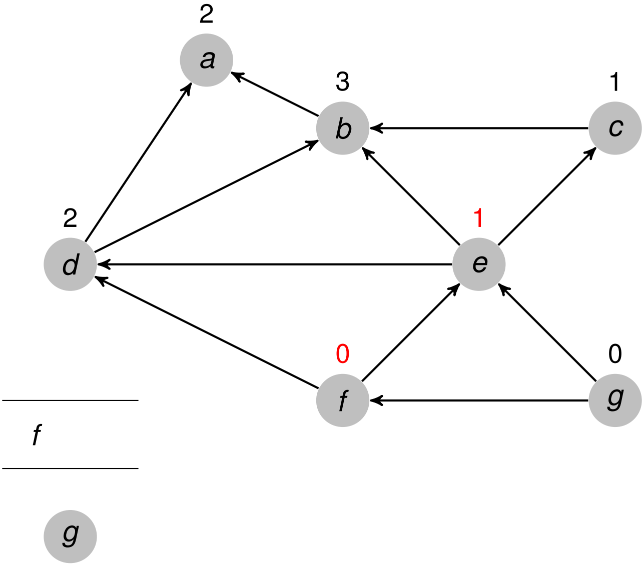
Topological Sort



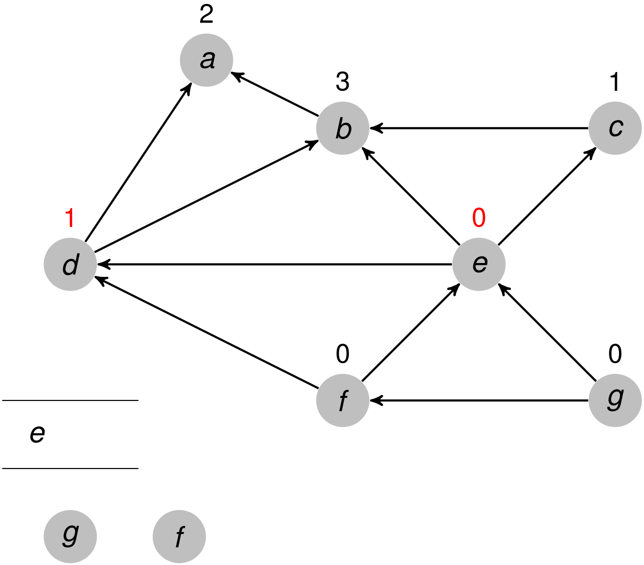
Topological Sort Implementation



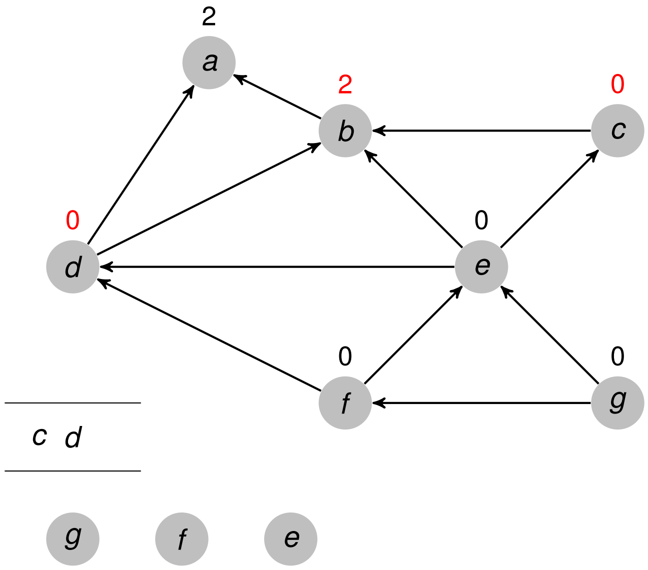
Topological Sort Implementation



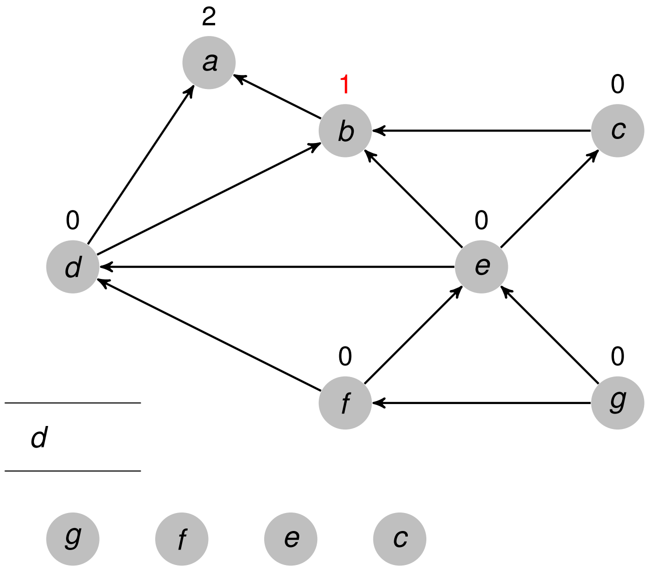
Topological Sort Implementation



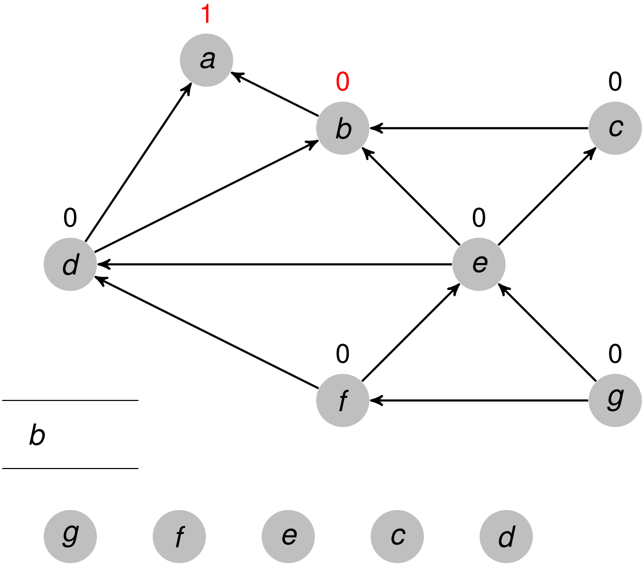
Topological Sort Implementation



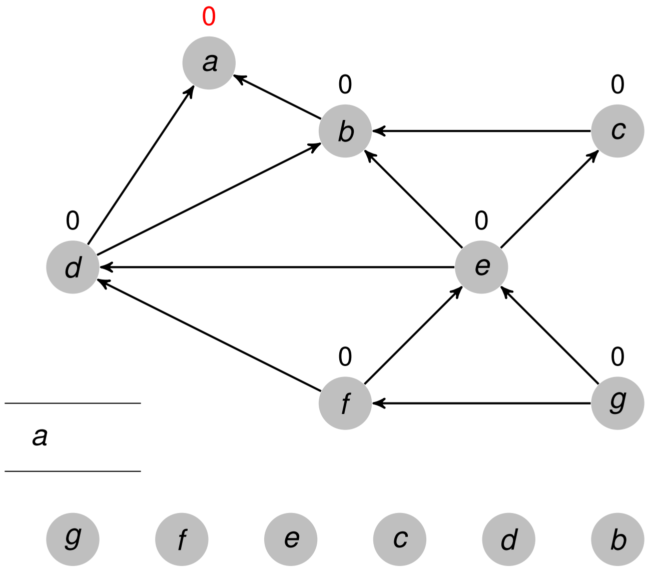
Topological Sort Implementation



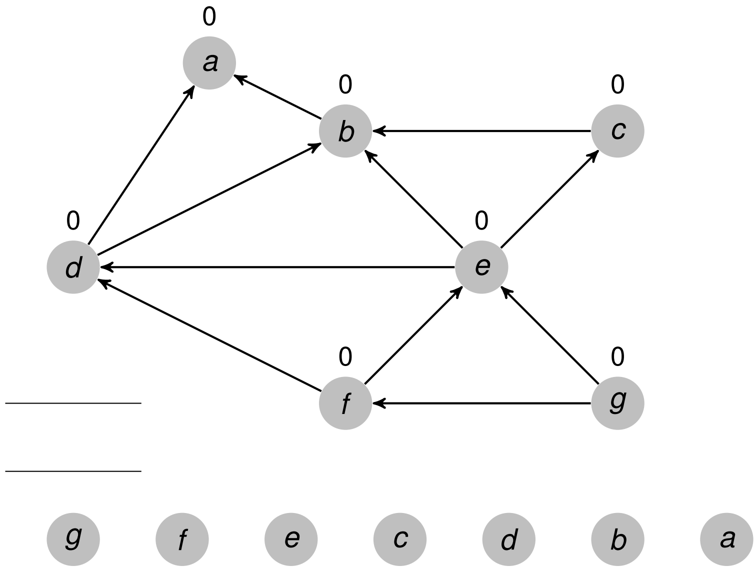
Topological Sort Implementation



Topological Sort Implementation



Topological Sort Implementation



Outline

Single-source shortest path problem

What ?

Given Vertex S in Graph G, find a shortest path from S to every other vertex in G.

Why ?

Find the cheapest way for one computer to broadcast a message to all other computers on the computer network.

How ?

- Dijkstra’s algorithm

Dijkstra's algorithm

1. Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree

Dijkstra's algorithm

1. Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree
2. Assign a distance value (0 for first, ∞ for others) to all vertices in the input graph.

Dijkstra's algorithm

1. Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree
2. Assign a distance value (0 for first, ∞ for others) to all vertices in the input graph.
3. While sptSet doesn't include all vertices

Dijkstra's algorithm

1. Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree
2. Assign a distance value (0 for first, ∞ for others) to all vertices in the input graph.
3. While sptSet doesn't include all vertices
   3.1 Pick a vertex u $\notin$ sptSet and has minimum distance value.

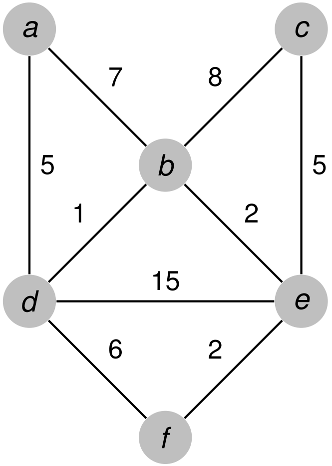
Dijkstra's algorithm

1. Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree
2. Assign a distance value (0 for first, ∞ for others) to all vertices in the input graph.
3. While sptSet doesn't include all vertices
    - 3.1 Pick a vertex u $\notin$ sptSet and has minimum distance value.
    - 3.2 Include u to sptSet.

Dijkstra's algorithm

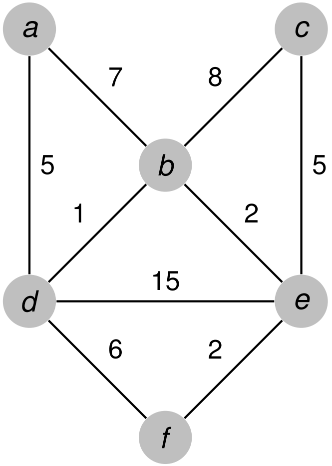
1. Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree
2. Assign a distance value (0 for first, ∞ for others) to all vertices in the input graph.
3. While sptSet doesn't include all vertices
   3.1 Pick a vertex u $\notin$ sptSet and has minimum distance value.
   3.2 Include u to sptSet.
   3.3 For every adjacent vertex v of u, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Example of Dijkstra's algorithm



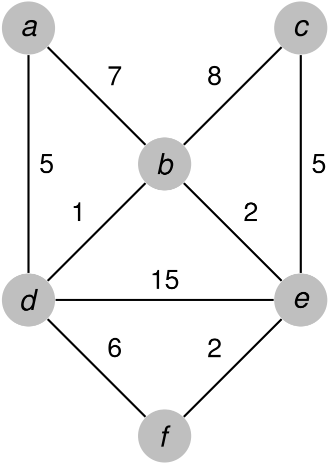
	a	b	c	d	e	f
Initial	0	∞	∞	∞	∞	∞

Example of Dijkstra's algorithm



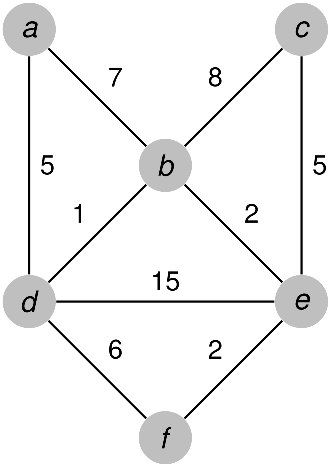
	a	b	c	d	e	f
Initial	0	∞	∞	∞	∞	∞
Proc.a	0	7	∞	5	∞	∞

Example of Dijkstra's algorithm



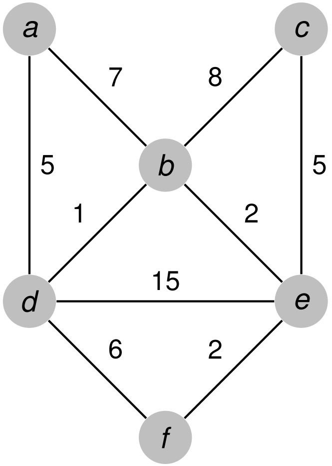
	a	b	c	d	e	f
Initial	0	∞	∞	∞	∞	∞
Proc.a	0	7	∞	5	∞	∞
Proc.d	0	6	∞	5	20	11

Example of Dijkstra's algorithm



	a	b	*c*	d	e	*f*
Initial	0	∞	∞	∞	∞	∞
Proc.a	0	7	∞	5	∞	∞
Proc.d	0	6	∞	5	20	11
Proc.b	0	6	14	5	8	11

Example of Dijkstra's algorithm



	a	b	c	d	e	f
Initial	0	∞	∞	∞	∞	∞
Proc.a	0	7	∞	5	∞	∞
Proc.d	0	6	∞	5	20	11
Proc.b	0	6	14	5	8	11
Proc.?	0	6		5		
Proc.?	0	6		5		
Proc.?	0	6		5		

Minimum Spanning Tree (MST)

What?

- Spanning tree: tree that contains all of the vertices in a connected graph.

Why?

How?

Minimum Spanning Tree (MST)

What?

- Spanning tree: tree that contains all of the vertices in a connected graph.
- Minimum spanning tree: spanning tree such that the sum of its weights are minimal.

Why?

How?

Minimum Spanning Tree (MST)

What?

- Spanning tree: tree that contains all of the vertices in a connected graph.
- Minimum spanning tree: spanning tree such that the sum of its weights are minimal.

Why?

- Communication

How?

Minimum Spanning Tree (MST)

What?

- Spanning tree: tree that contains all of the vertices in a connected graph.
- Minimum spanning tree: spanning tree such that the sum of its weights are minimal.

Why?

- Communication
- Water supply

How?

Minimum Spanning Tree (MST)

What?

- Spanning tree: tree that contains all of the vertices in a connected graph.
- Minimum spanning tree: spanning tree such that the sum of its weights are minimal.

Why?

- Communication
- Water supply
- Electricity Transmission

How?

Minimum Spanning Tree (MST)

What?

- Spanning tree: tree that contains all of the vertices in a connected graph.
- Minimum spanning tree: spanning tree such that the sum of its weights are minimal.

Why?

- Communication
- Water supply
- Electricity Transmission

How?

- Prim's algorithm

Minimum Spanning Tree (MST)

What?

- Spanning tree: tree that contains all of the vertices in a connected graph.
- Minimum spanning tree: spanning tree such that the sum of its weights are minimal.

Why?

- Communication
- Water supply
- Electricity Transmission

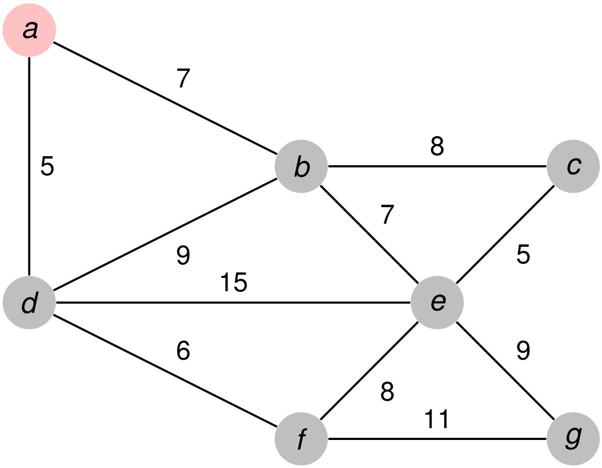
How?

- Prim's algorithm
- Kruskal's algorithm

Prim's Algorithm

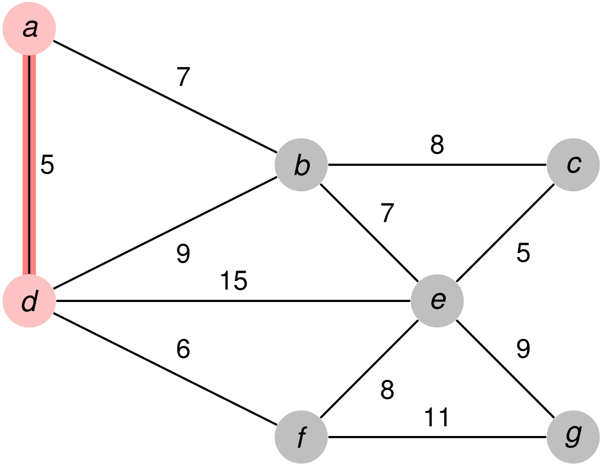
1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value (0 for first, ∞ for others) to all vertices in the input graph.
3. While mstSet doesn't include all vertices
   3.1 Pick a vertex u $\notin$ mstSet and has minimum key value.
   3.2 Include u to mstSet.
   3.3 For every adjacent vertex v of u, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v

Example of Prim's Algorithm



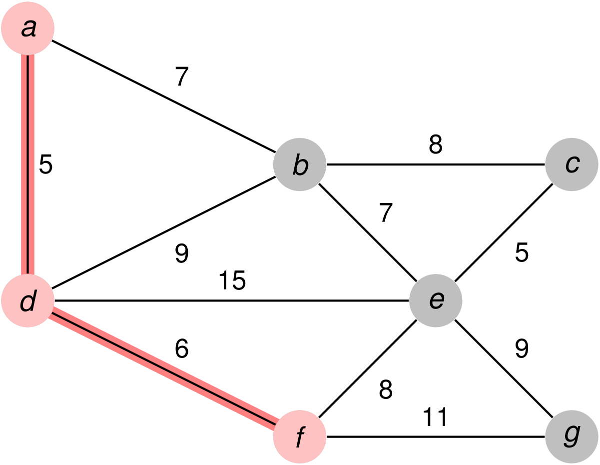
a	b	c	d	e	f	g
0	∞	∞	∞	∞	∞	∞

Example of Prim's Algorithm



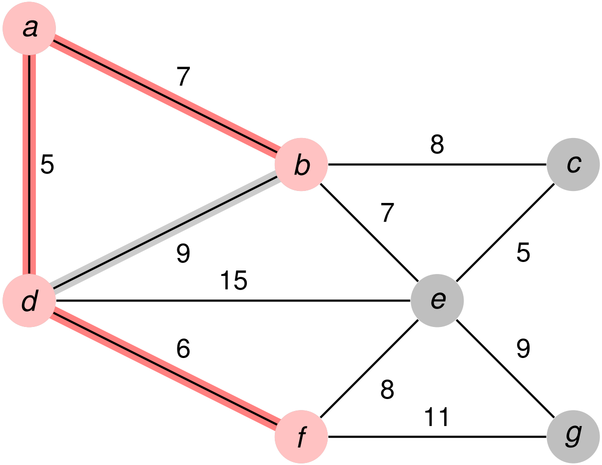
a	b	c	d	e	f	g
0	7	∞	5	∞	∞	∞

Example of Prim's Algorithm



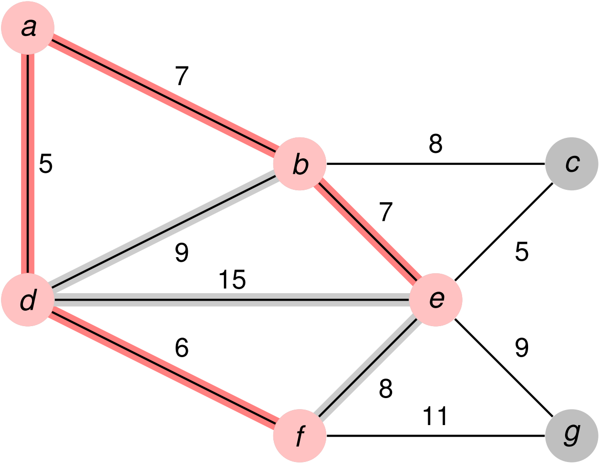
a	b	c	d	e	f	g
0	7	∞	5	15	6	∞

Example of Prim's Algorithm



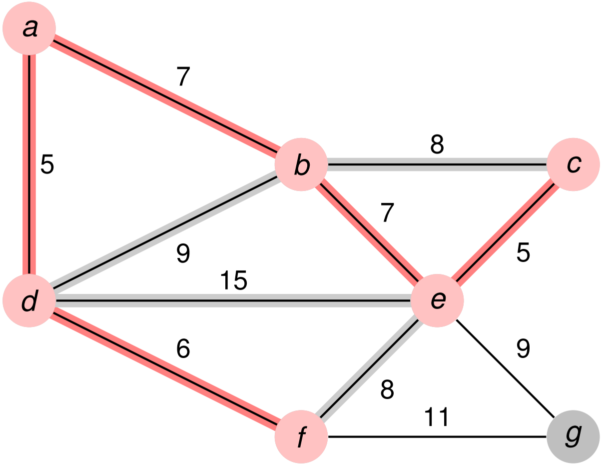
a	b	c	d	e	f	g
0	7	∞	5	8	6	11

Example of Prim's Algorithm



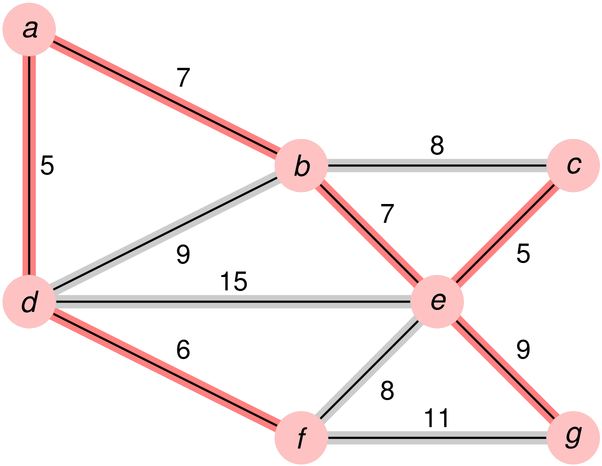
a	b	c	d	e	f	g
0	7	8	5	7	6	11

Example of Prim's Algorithm



a	b	c	d	e	f	g
0	7	5	5	7	6	9

Example of Prim's Algorithm



a	b	c	d	e	f	g
0	7	5	5	7	6	9

Kruskal's Algorithm

1. Sort all the edges in non-decreasing order of their weight.

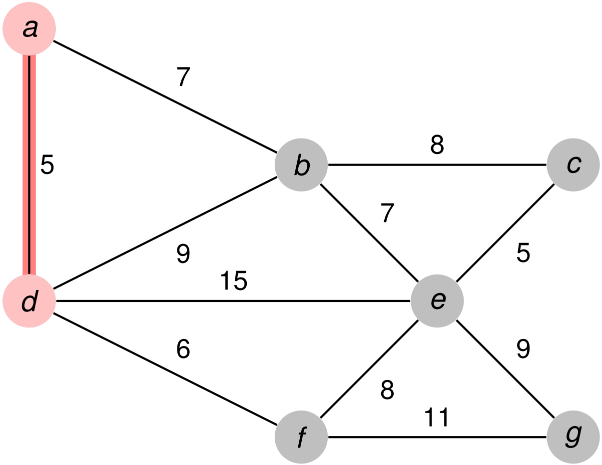
Kruskal's Algorithm

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

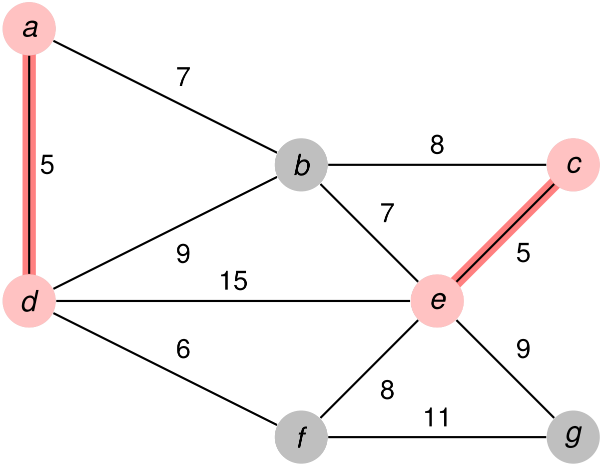
Kruskal's Algorithm

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step #2 until there are (V-1) edges in the spanning tree.

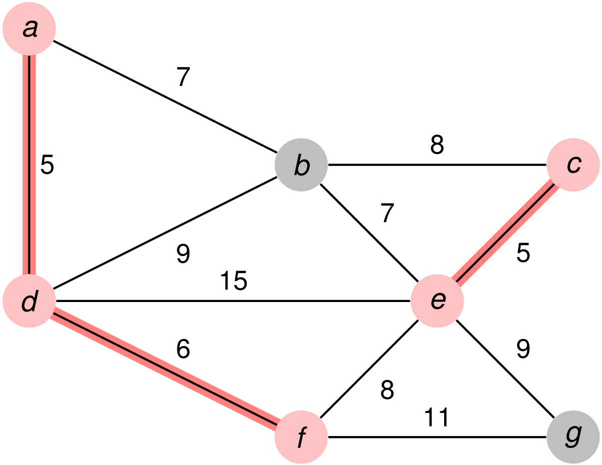
Example of Kruskal’s Algorithm



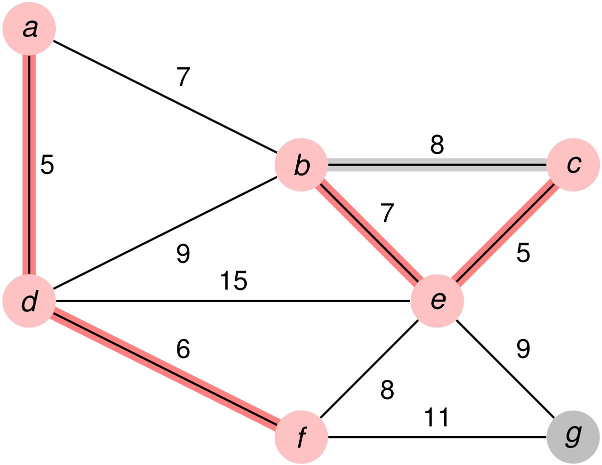
Example of Kruskal's Algorithm



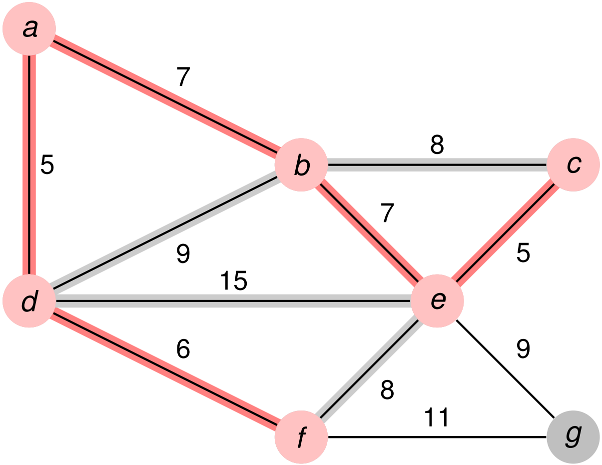
Example of Kruskal's Algorithm



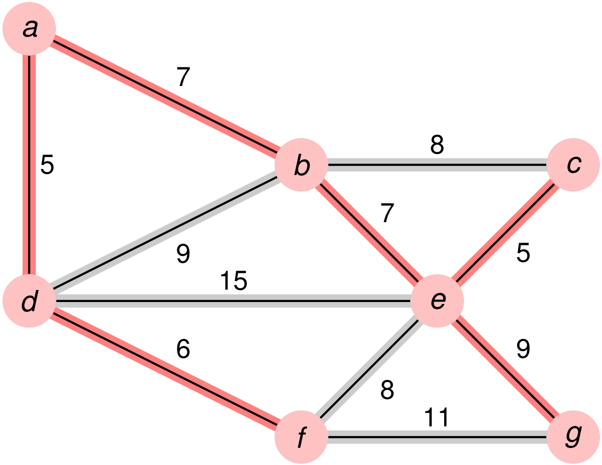
Example of Kruskal's Algorithm



Example of Kruskal's Algorithm



Example of Kruskal's Algorithm



Summary

- Terminology
- Graph Representations: Adjacent Matrix, Adjacent List
- Graph Traversal: DFS, BFS
- Topological Sort
- Graph Problems: Shortest-Path, MST