Intro to Graph Theory

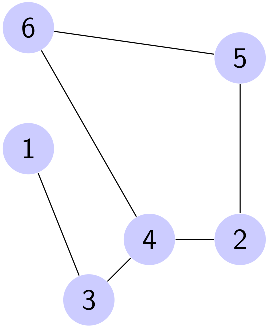
2014 IOI Camp 1

Robert Spencer

December 11, 2013

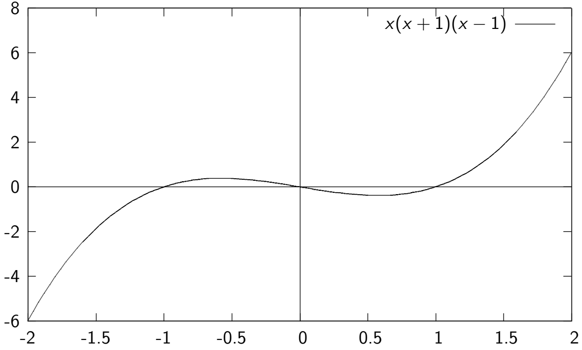
Introduction

This is a graph:



Introduction

This is not a graph:



Introduction

Definition

A graph is a collection of *nodes* connected by *edges* which may or may not be *directed* and/or *weighted*

Introduction

Definition

A graph is a collection of *nodes* connected by *edges* which may or may not be *directed* and/or *weighted*

Examples of graphs:

- A computer network (non-directed, non-weighted)
- A road map (non-directed, weighted)
- Winners in a chess tournament (directed, non-weighted)
- Payments in an economy (weighted, directed)

Paths and Cycles

Definition

- A path is a sequence of nodes such that each element is connected by an edge to the one before it.

Paths and Cycles

Definition

- A path is a sequence of nodes such that each element is connected by an edge to the one before it.
- A cycle is a path with its last element equal to its first.

Paths and Cycles

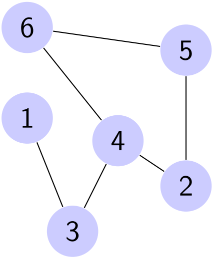
Definition

- A path is a sequence of nodes such that each element is connected by an edge to the one before it.
- A cycle is a path with its last element equal to its first.
- A connected graph is one which has a path joining every pair of vertices.

Definition

- A path is a sequence of nodes such that each element is connected by an edge to the one before it.
- A cycle is a path with its last element equal to its first.
- A connected graph is one which has a path joining every pair of vertices.

Can you find a path and a cycle?

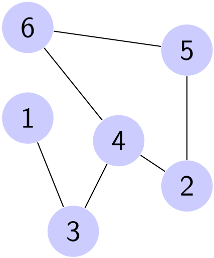


Paths and Cycles

Definition

- A path is a sequence of nodes such that each element is connected by an edge to the one before it.
- A cycle is a path with its last element equal to its first.
- A connected graph is one which has a path joining every pair of vertices.

Can you find a path and a cycle?



Example Answer:
Path: 1-3-4-2
Cycle: 4-2-5-6-4

Trees

Definition

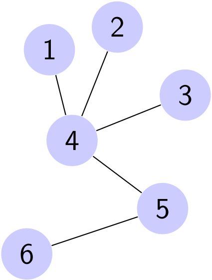
A tree is an un-directed complete graph with no cycles.

Trees

Definition

A tree is an un-directed complete graph with no cycles.

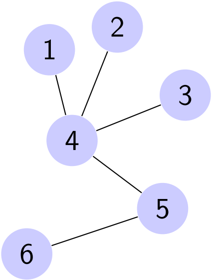
Example



Definition

A tree is an un-directed complete graph with no cycles.

Example



Theorem

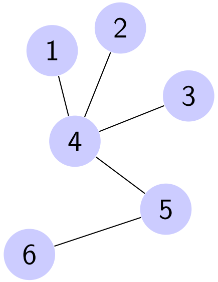
A tree of n vertices has $n-1$ edges.

Trees

Definition

A tree is an un-directed complete graph with no cycles.

Example



Theorem

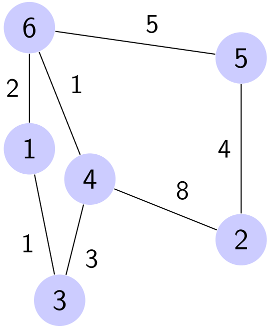
A tree of n vertices has $n - 1$ edges.

Proof.

Induction. Start with one vertex, and add subsequent ones. □

Trees

Weights are placed on edges, and can represent anything (lengths, costs, etc.)



How do we represent a graph?

Graph Representations

How do we represent a graph?

Lists of Neighbours

```
[(3,1),(6,2)]
[(4,8),(5,4)]
[(1,1),(4,3)]
[(2,8),(3,3),(6,1)]
[(2,4),(6,5)]
[(5,5)]
```

Memory $O(E)$

How do we represent a graph?

Lists of Neighbours

```
[(3,1),(6,2)]
[(4,8),(5,4)]
[(1,1),(4,3)]
[(2,8),(3,3),(6,1)]
[(2,4),(6,5)]
[(5,5)]
```

Memory $O(E)$

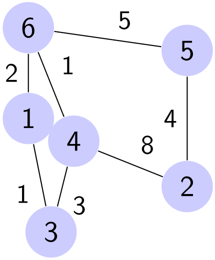
Adjacency Matrix

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 8 & 4 & 0 \\ 1 & 0 & 0 & 3 & 0 & 0 \\ 0 & 8 & 3 & 0 & 0 & 1 \\ 0 & 4 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 1 & 5 & 0 \end{pmatrix}$$

Memory $O(N^2)$

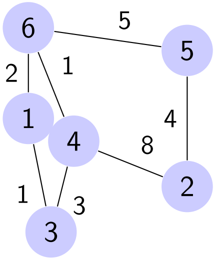
Traversal

Sometimes we want to visit all the nodes in a graph in a particular order. For example to search for a path/destination



Traversal

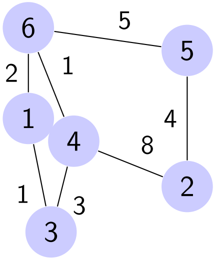
Sometimes we want to visit all the nodes in a graph in a particular order. For example to search for a path/destination



We may visit nodes more than once, as there may be more than one path. E.g. to get from 1 to 2, we may visit 4 twice: 1-6-4-2 or 1-3-4-2.

Traversal

Sometimes we want to visit all the nodes in a graph in a particular order. For example to search for a path/destination



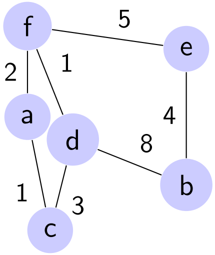
We may visit nodes more than once, as there may be more than one path. E.g. to get from 1 to 2, we may visit 4 twice: 1-6-4-2 or 1-3-4-2.

Often this is used to find the shortest route between two or more nodes.

Depth First Search

Depth First Search (DFS) visits the nodes as far as it can before backtracking (without visiting nodes more than once).

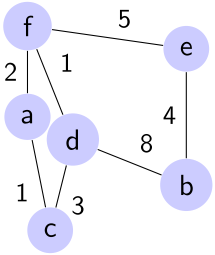
Sample Graph:



Depth First Search

Depth First Search (DFS) visits the nodes as far as it can before backtracking (without visiting nodes more than once).

Sample Graph:

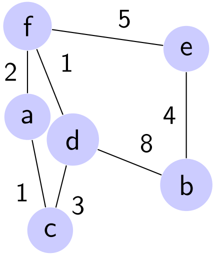


Psudocode:

```
def DFS(currNode, finalNode)
  if currNode==finalNode then
    return success
  set currNode visited
  foreach neighbour of currNode do
    if neighbour not visited then
    DFS(neighbour,finalNode)
  unset currNode visited
```

Depth First Search (DFS) visits the nodes as far as it can before backtracking (without visiting nodes more than once).

Sample Graph:



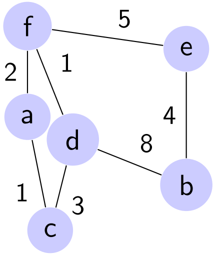
Psudocode:

```
def DFS(currNode, finalNode)
  if currNode==finalNode then
    return success
  set currNode visited
  foreach neighbour of currNode do
    if neighbour not visited then
    DFS(neighbour,finalNode)
  unset currNode visited
```

Nodes (starting from a) will be visited in this order:

Depth First Search (DFS) visits the nodes as far as it can before backtracking (without visiting nodes more than once).

Sample Graph:

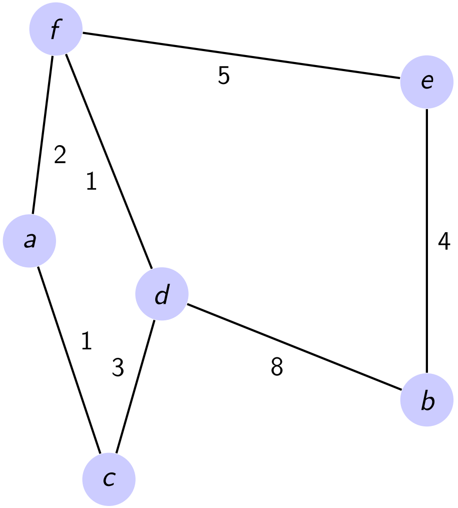


Psudocode:

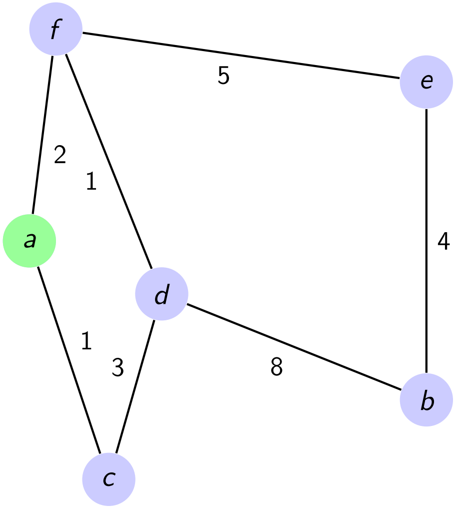
```
def DFS(currNode, finalNode)
  if currNode==finalNode then
    return success
  set currNode visited
  foreach neighbour of currNode do
    if neighbour not visited then
    DFS(neighbour,finalNode)
  unset currNode visited
```

Nodes (starting from a) will be visited in this order:
a-c-d-b-e-f-f-e-b-f-d-c-b-e

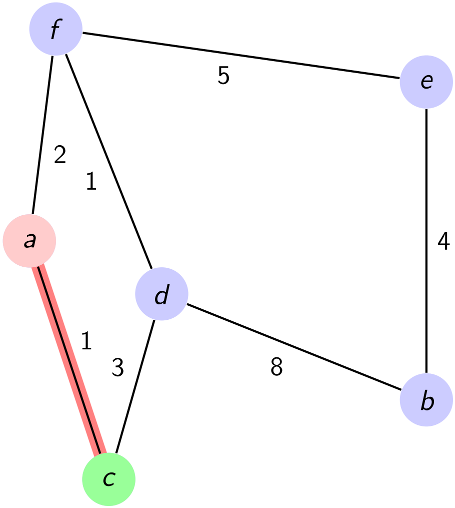
Depth First Search Example



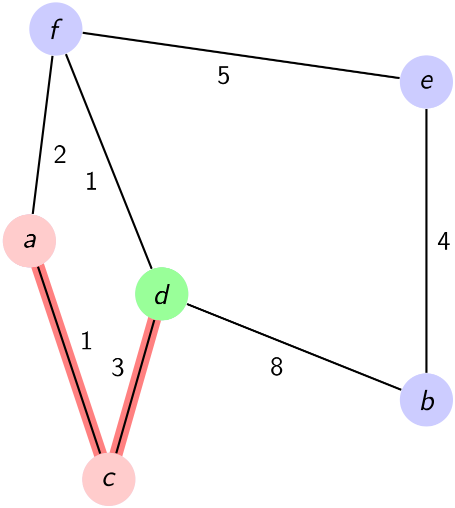
Depth First Search Example



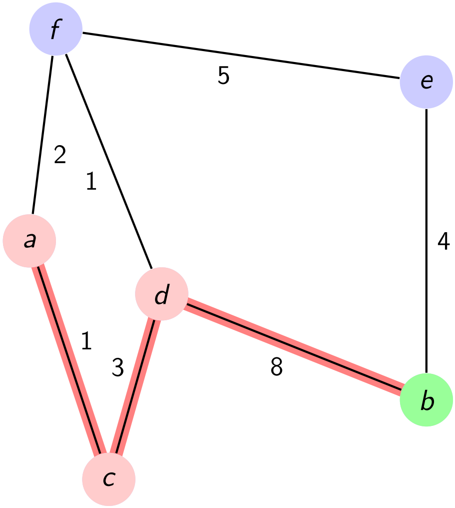
Depth First Search Example



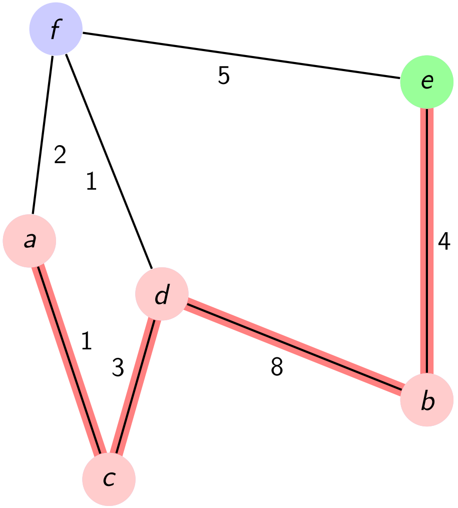
Depth First Search Example



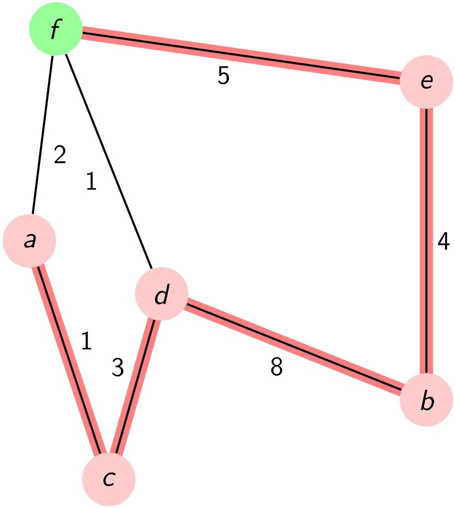
Depth First Search Example



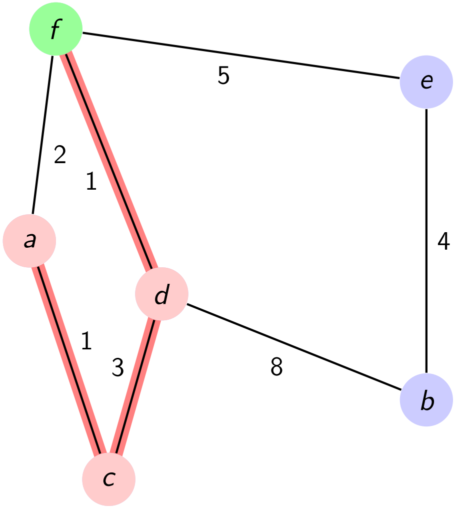
Depth First Search Example



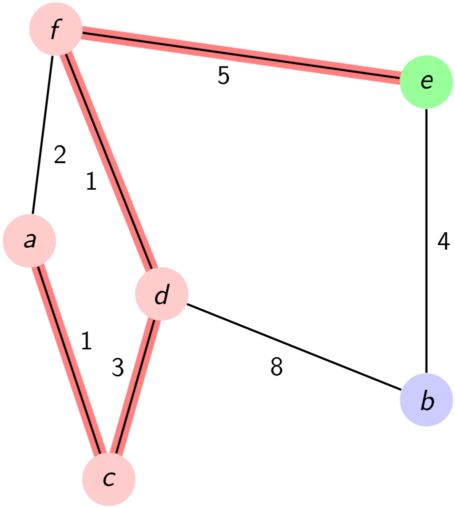
Depth First Search Example



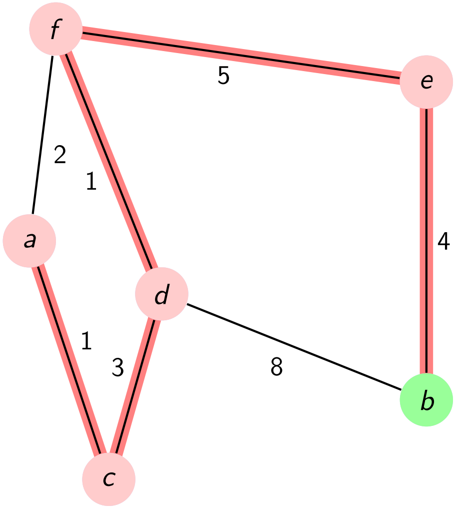
Depth First Search Example



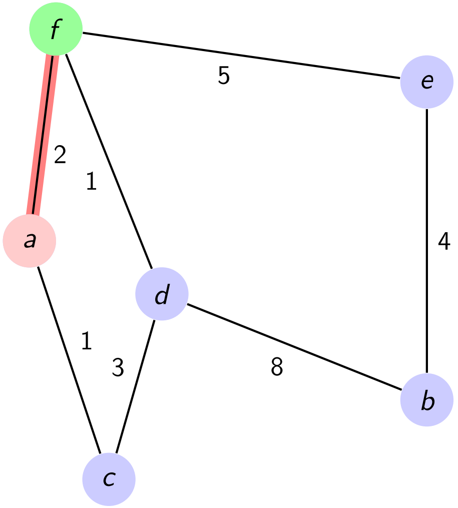
Depth First Search Example



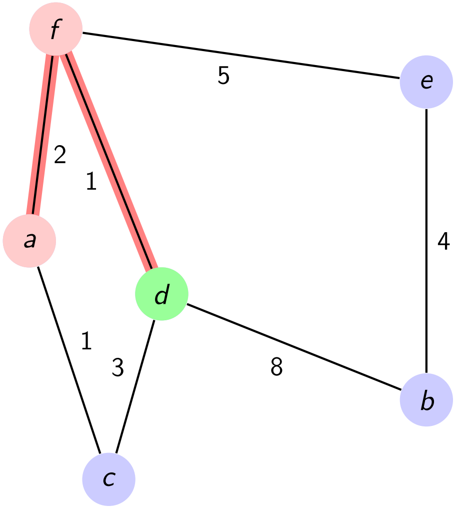
Depth First Search Example



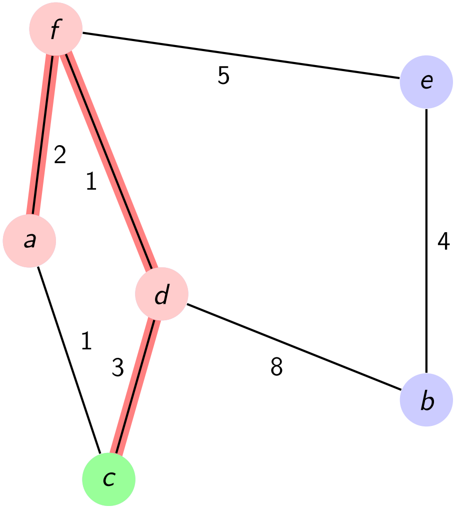
Depth First Search Example



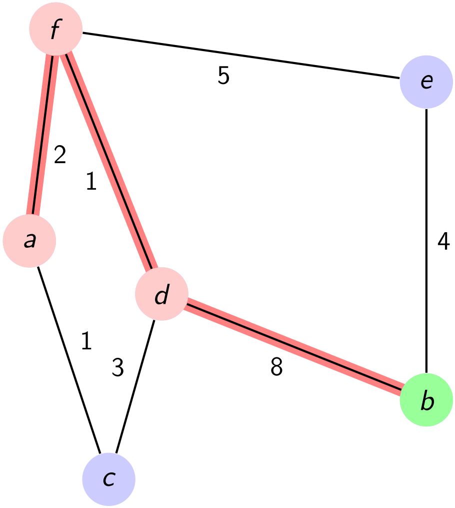
Depth First Search Example



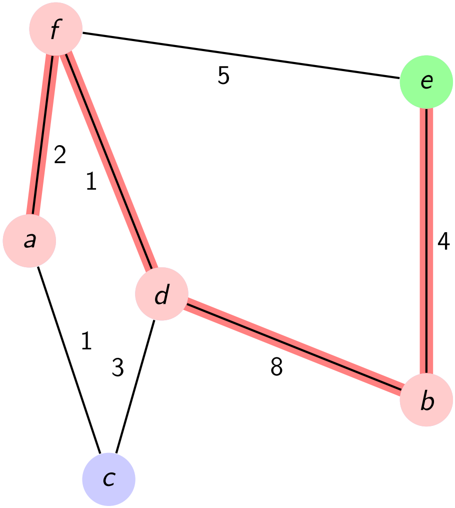
Depth First Search Example



Depth First Search Example



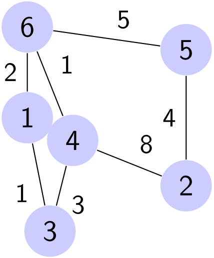
Depth First Search Example



Breadth First Search

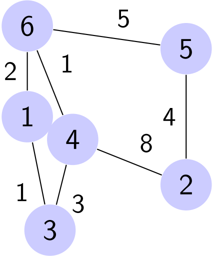
Breadth First Search (DFS) visits the nodes "in parallel" without backtracking.

Sample Graph:



Breadth First Search (DFS) visits the nodes "in parallel" without backtracking.

Sample Graph:

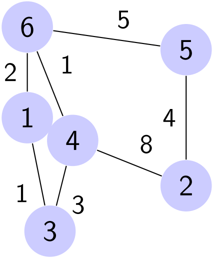


Psudocode:

```
def BFS(currNode, finalNode)
  add currNode to queue
  while queue not empty do
    pop first element as currNode
    set currNode visited
    foreach neighbour of currNode do
      if neighbour not visited
        add neighbour to queue
```

Breadth First Search (DFS) visits the nodes "in parallel" without backtracking.

Sample Graph:



Psudocode:

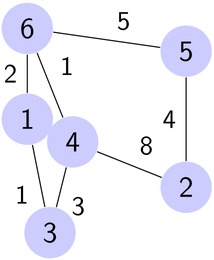
```
def BFS(currNode, finalNode)
  add currNode to queue
  while queue not empty do
    pop first element as currNode
    set currNode visited
    foreach neighbour of currNode do
      if neighbour not visited
        add neighbour to queue
```

Nodes (starting from 1) will be visited in this order:

Breadth First Search

Breadth First Search (DFS) visits the nodes "in parallel" without backtracking.

Sample Graph:

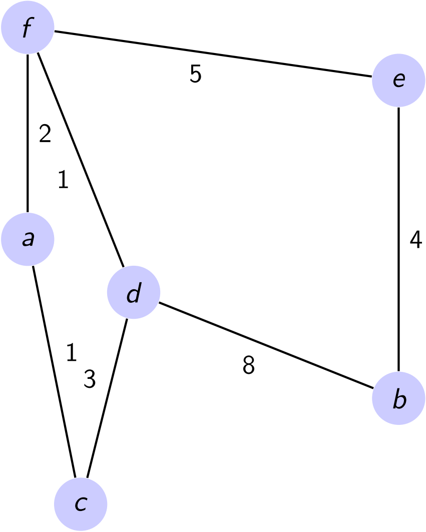


Psudocode:

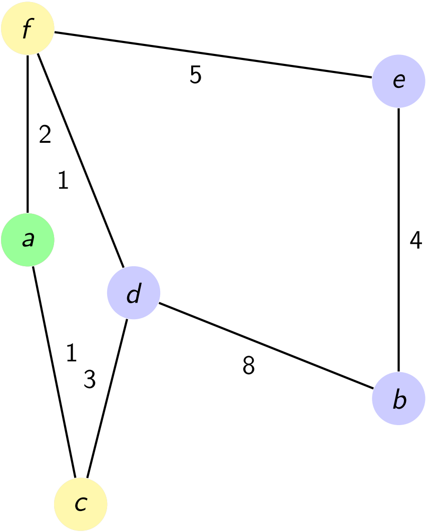
```
def BFS(currNode, finalNode)
  add currNode to queue
  while queue not empty do
    pop first element as currNode
    set currNode visited
    foreach neighbour of currNode do
      if neighbour not visited
        add neighbour to queue
```

Nodes (starting from 1) will be visited in this order: 1-3-6-4-5-2

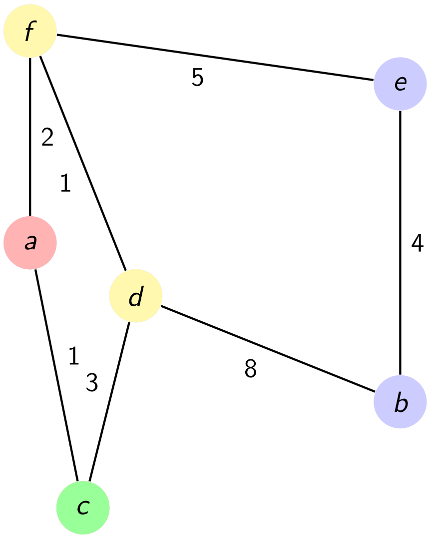
Breadth First Search Example



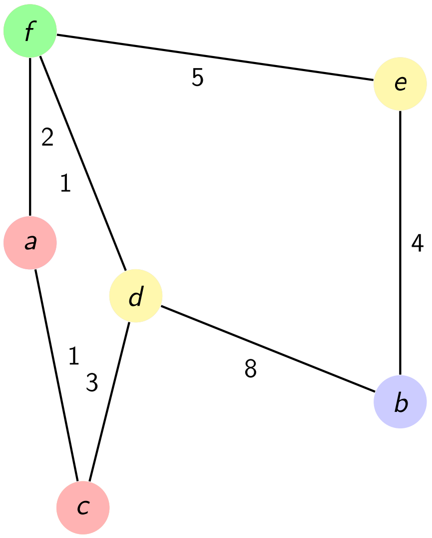
Breadth First Search Example



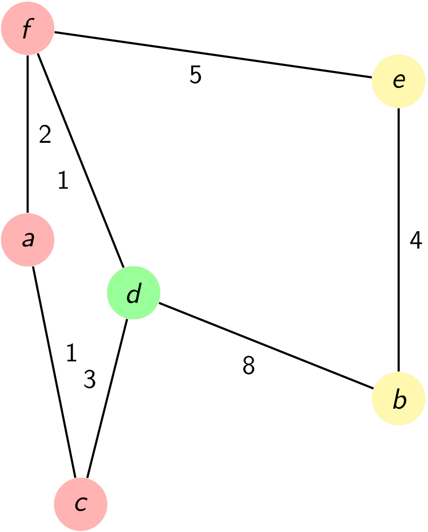
Breadth First Search Example



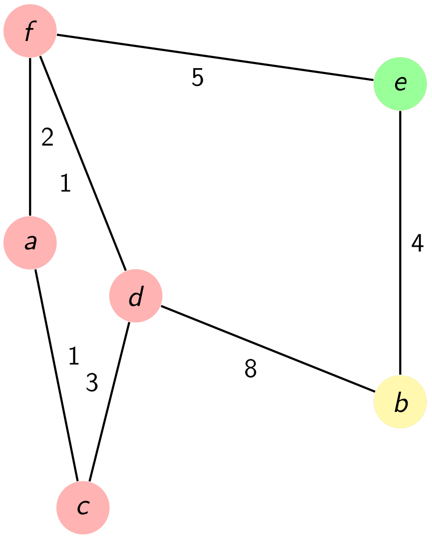
Breadth First Search Example



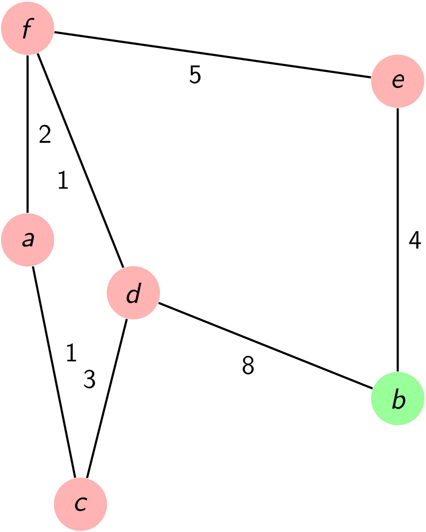
Breadth First Search Example



Breadth First Search Example



Breadth First Search Example



Dijkstra's Algorithm

Dijkstra's Algorithm finds the shortest distance from one node to all others. It is basically a BFS with a priority queue.

Psudocode:

```
set all distances INF
add (0, startNode) to queue
while queue not empty do
  currDists,currNode = queue.pop
  distances[currNode] = currDist
  for neighbour,distance in adjacent[currNode] do
    possNewDist = distances[currNode] + distance
    if distances[neighbour] > possNewDist then
     update neighbour to weight possNewDist in queue
```

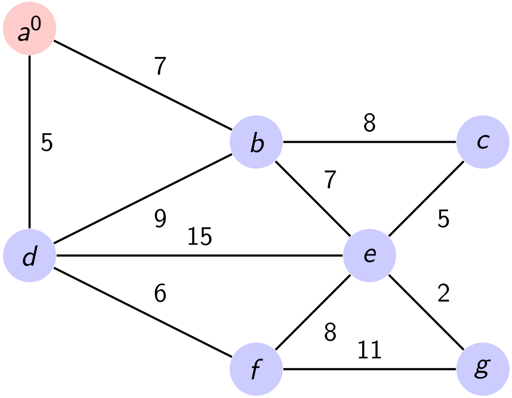
Dijkstra's Algorithm

Dijkstra's Algorithm finds the shortest distance from one node to all others. It is basically a BFS with a priority queue.

Psudocode:

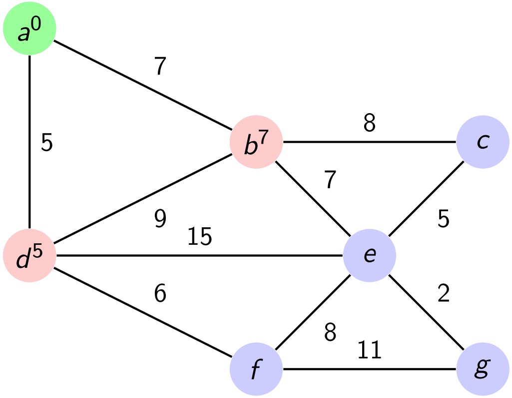
```
set all distances INF
add (0, startNode) to queue
while queue not empty do
  currDists,currNode = queue.pop
  distances[currNode] = currDist
  for neighbour,distance in adjacent[currNode] do
    possNewDist = distances[currNode] + distance
    if distances[neighbour] > possNewDist then
     update neighbour to weight possNewDist in queue
```

Dijkstra Example



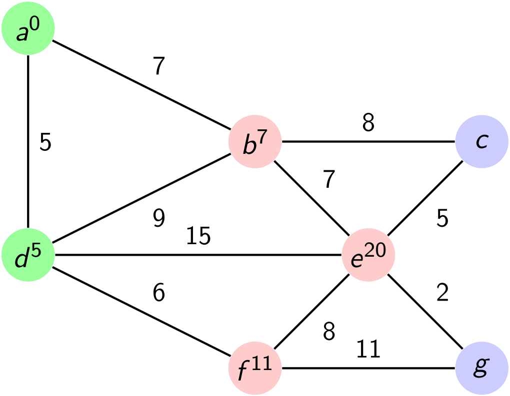
Queue: $\{(a, 0)\}$

Dijkstra Example



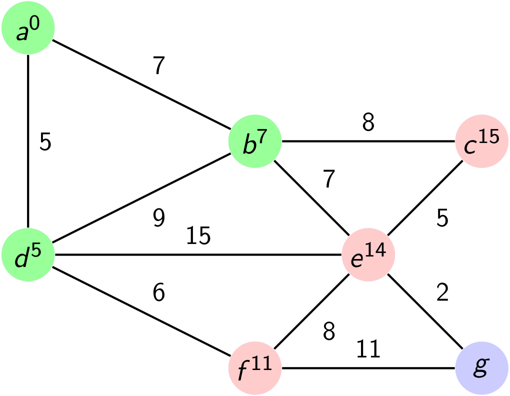
Queue: $\{(d,5),(b,7)\}$

Dijkstra Example



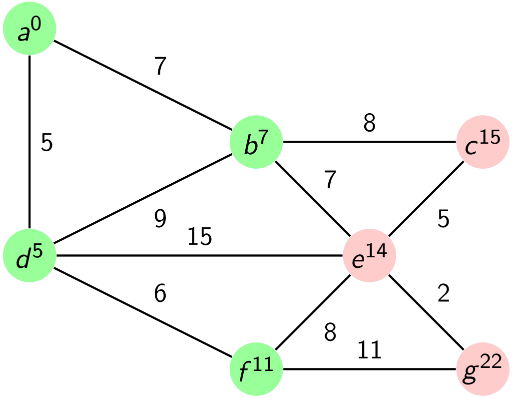
Queue: $\{(b,7),(f,11),(e,20)\}$

Dijkstra Example



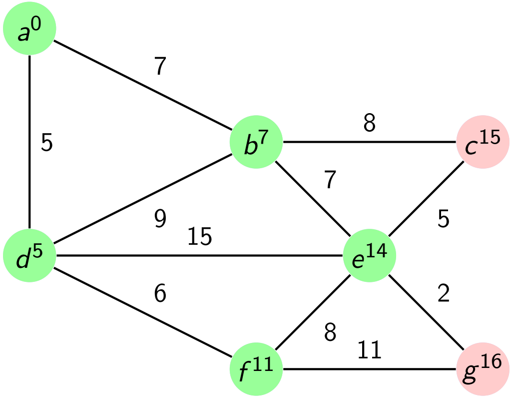
Queue: $\{(f, 11), (e, 14), (c, 15)\}$

Dijkstra Example



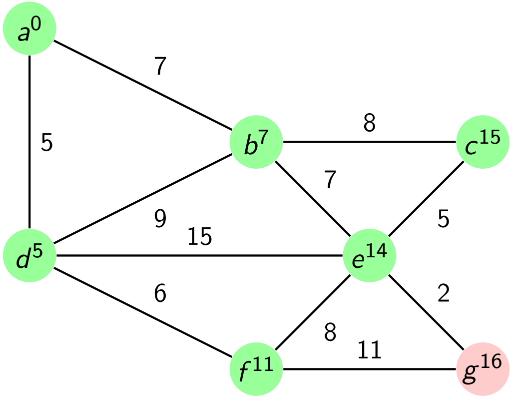
Queue: $\{(e, 14), (c, 15), (g, 22)\}$

Dijkstra Example



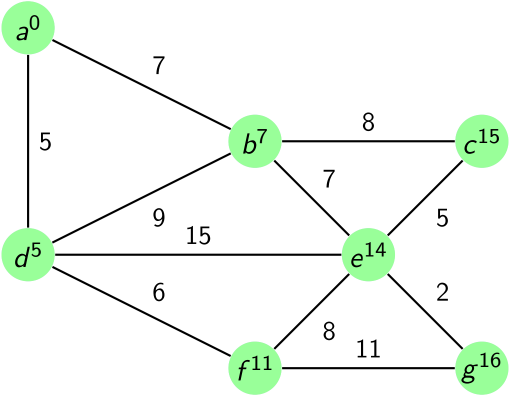
Queue: $\{(c, 15), (g, 16)\}$

Dijkstra Example



Queue: $\{(g, 16)\}$

Dijkstra Example

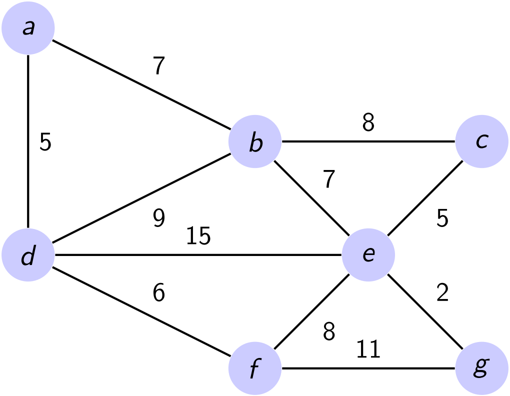


Queue: {}

Minimum Spanning Tree

Definition

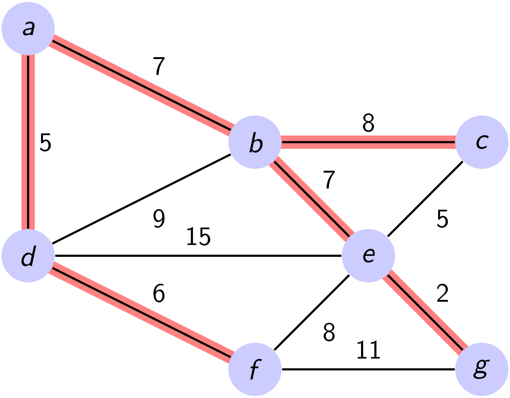
A *minimum spanning tree* is a subset of edges in a weighted undirected graph such that the edges form a tree containing all the nodes, and the sum of the weights of the tree is minimal.



Minimum Spanning Tree

Definition

A *minimum spanning tree* is a subset of edges in a weighted undirected graph such that the edges form a tree containing all the nodes, and the sum of the weights of the tree is minimal.



Prim's Algorithm finds the minimum spanning tree from a given graph.

Prim's Algorithm

Prim's Algorithm finds the minimum spanning tree from a given graph.

Algorithm

- Set all vertices to "not in the tree" except a starting vertex.
- While there are vertices not in the tree, add the vertex which is connected to the tree by the shortest edge to the tree.

Prim's Algorithm

Prim's Algorithm finds the minimum spanning tree from a given graph.

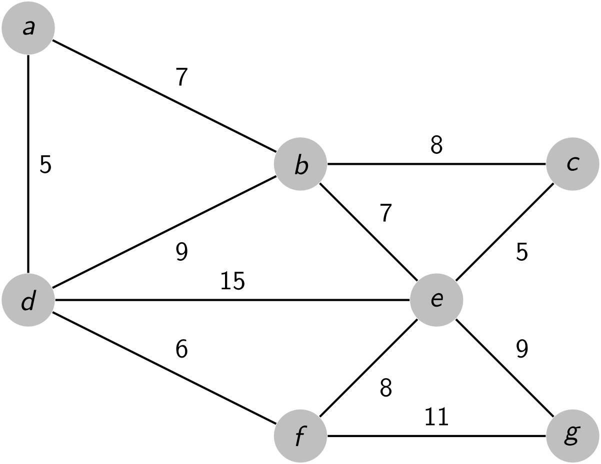
Algorithm

- Set all vertices to "not in the tree" except a starting vertex.
- While there are vertices not in the tree, add the vertex which is connected to the tree by the shortest edge to the tree.

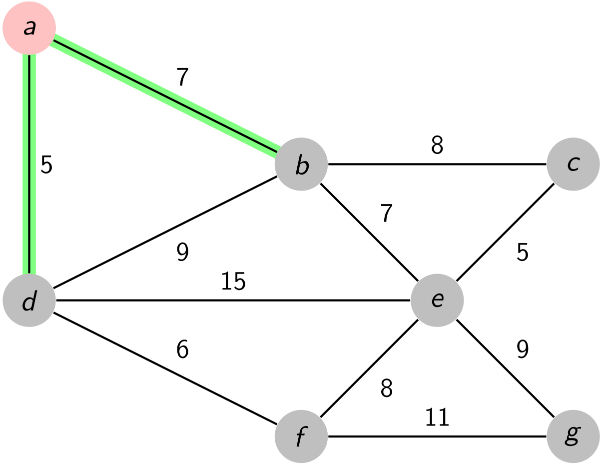
Technical notes

- Keep a priority queue of edges. Each step pull off an edge, check if it joins a new vertex. If it does, add all the edges from that vertex to the queue.
- Runs in $O(E \log V)$ with a binary heap as priority queue.

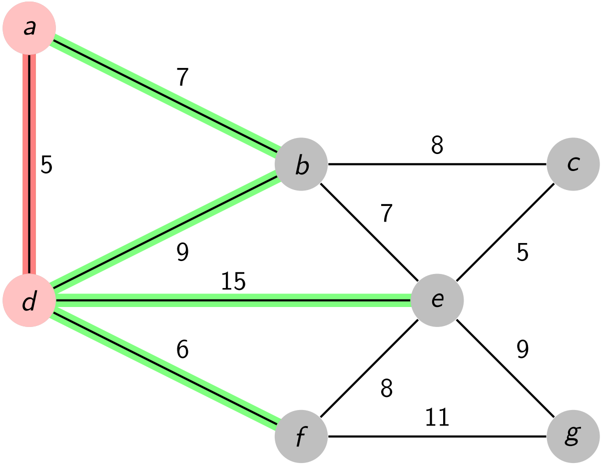
Prim's Algorithm Example



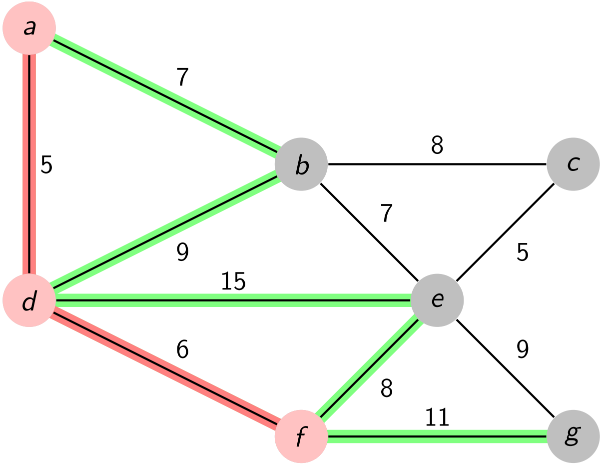
Prim's Algorithm Example



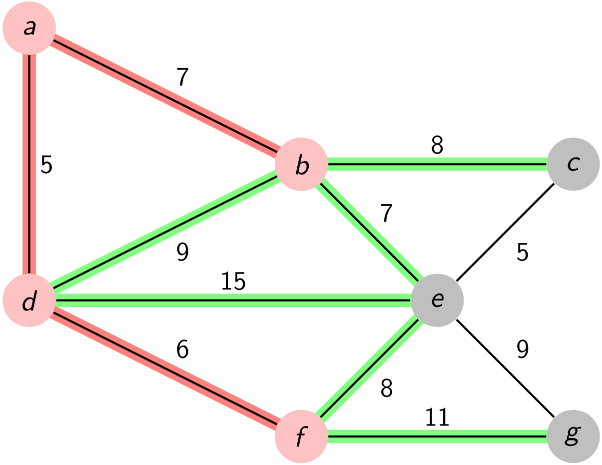
Prim's Algorithm Example



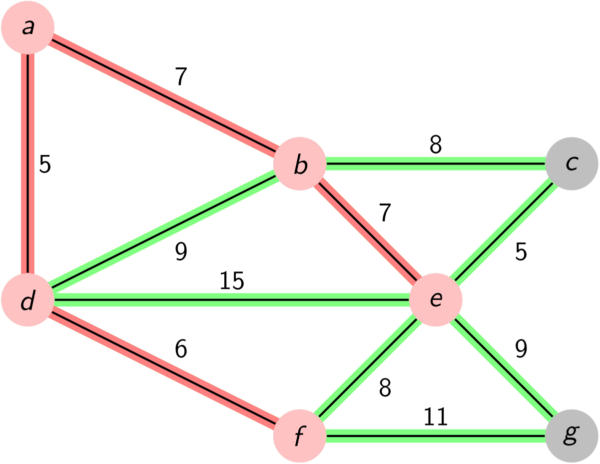
Prim's Algorithm Example



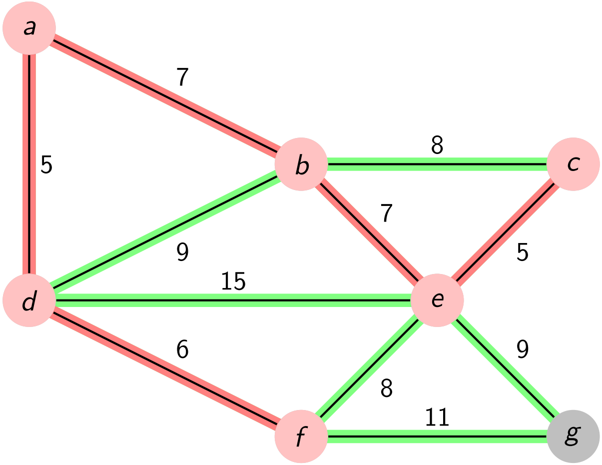
Prim's Algorithm Example



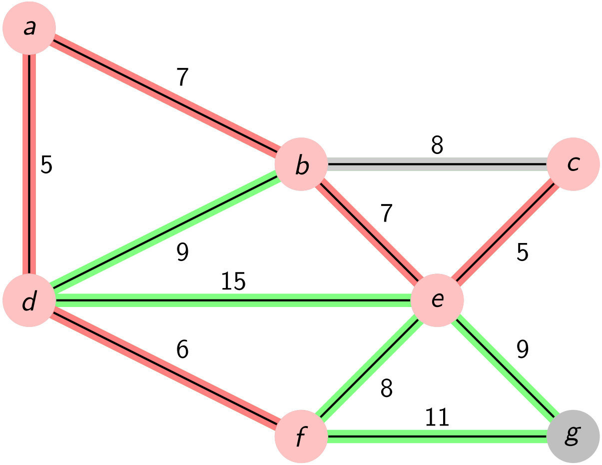
Prim's Algorithm Example



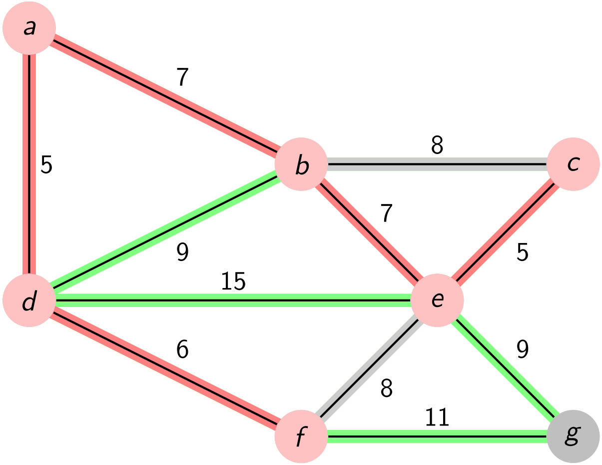
Prim's Algorithm Example



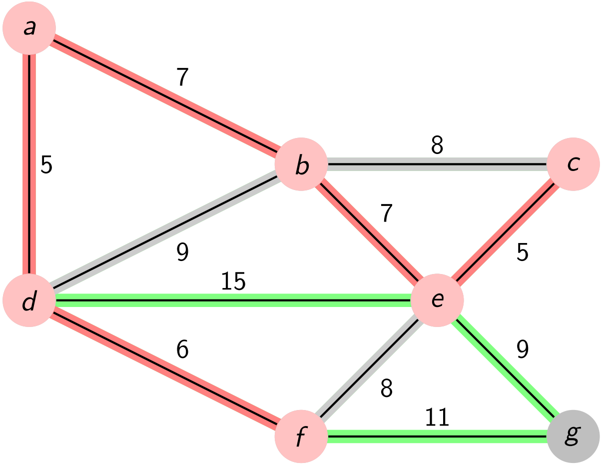
Prim's Algorithm Example



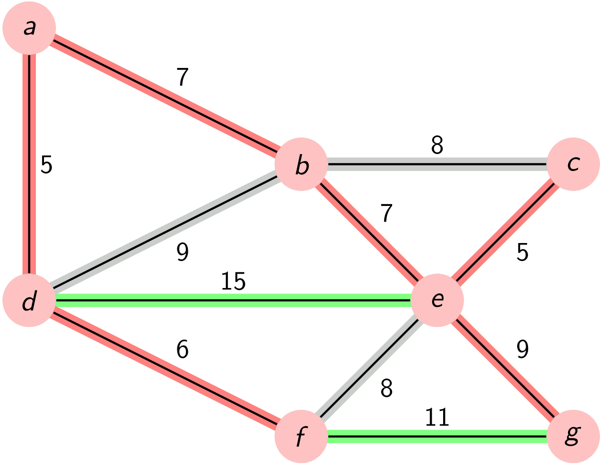
Prim's Algorithm Example



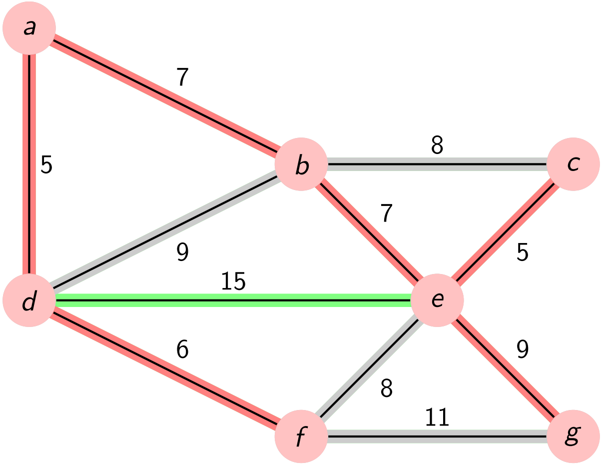
Prim's Algorithm Example



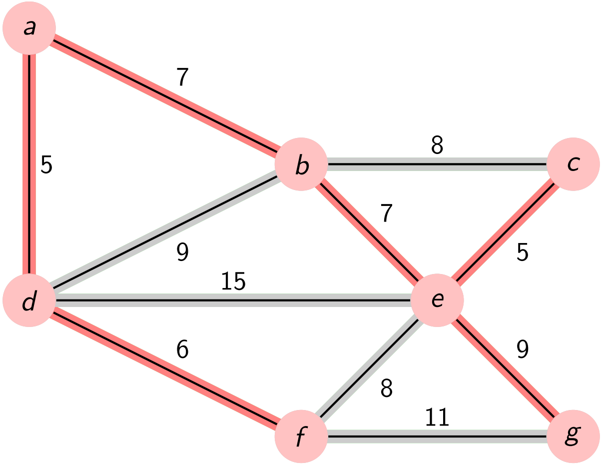
Prim's Algorithm Example



Prim's Algorithm Example



Prim's Algorithm Example



Kruskal's Algorithm

Kruskal's Algorithm is the "dual" of Prim's. It also finds the minimum spanning tree.

Kruskal's Algorithm

Kruskal's Algorithm is the "dual" of Prim's. It also finds the minimum spanning tree.

Algorithm:

- Consider each vertex as a tree, containing no edges and just itself.
- While we don't have a MST, consider the smallest edge not yet considered. If it joins two different trees, include it in the MST.

Kruskal's Algorithm

Kruskal's Algorithm is the "dual" of Prim's. It also finds the minimum spanning tree.

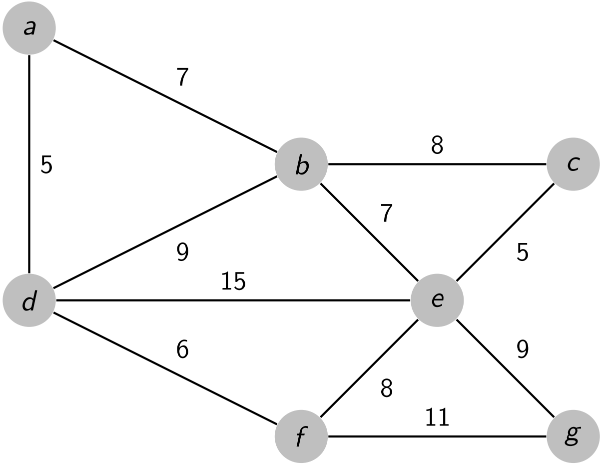
Algorithm:

- Consider each vertex as a tree, containing no edges and just itself.
- While we don't have a MST, consider the smallest edge not yet considered. If it joins two different trees, include it in the MST.

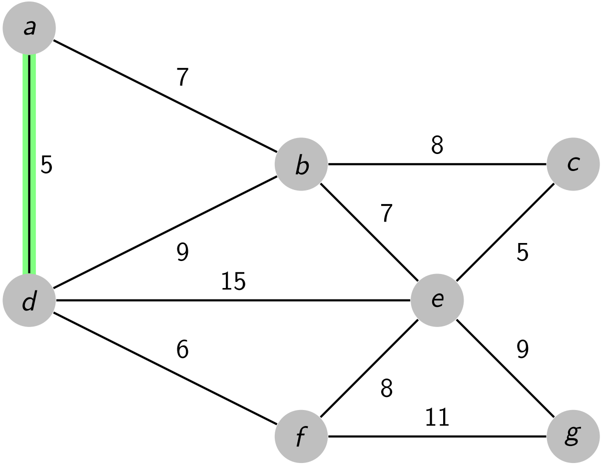
Technical Notes:

- Use "union-find" to hold the different trees.
- Complexity $O(E \log E)$

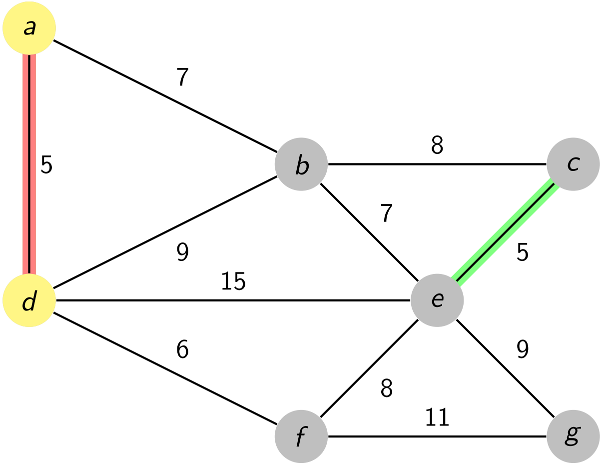
Kruskal's Algorithm Example



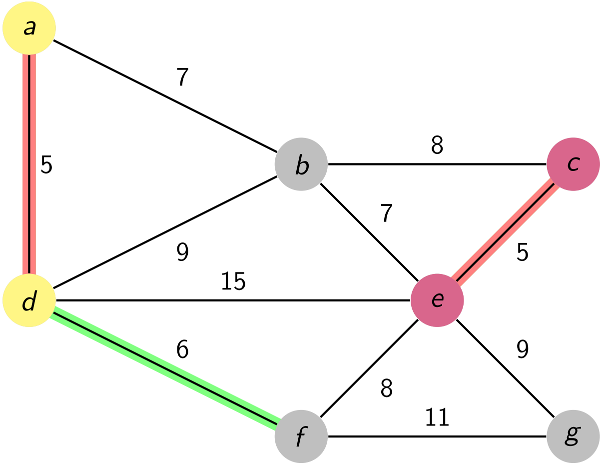
Kruskal's Algorithm Example



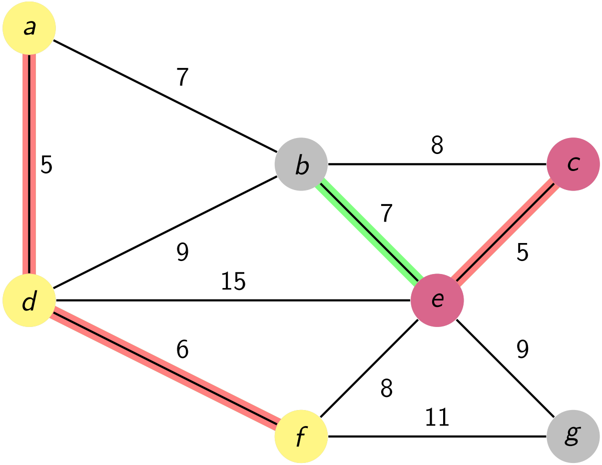
Kruskal's Algorithm Example



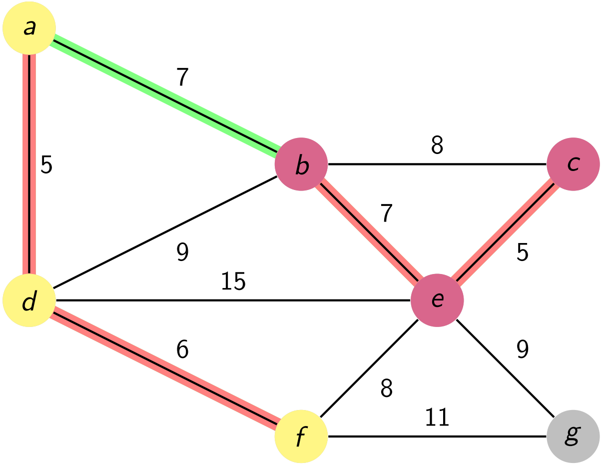
Kruskal's Algorithm Example



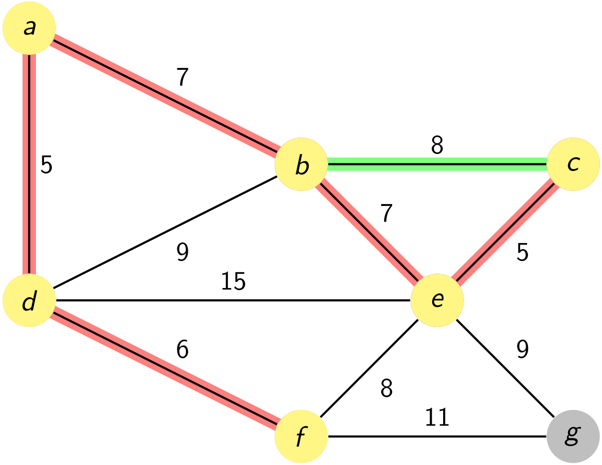
Kruskal's Algorithm Example



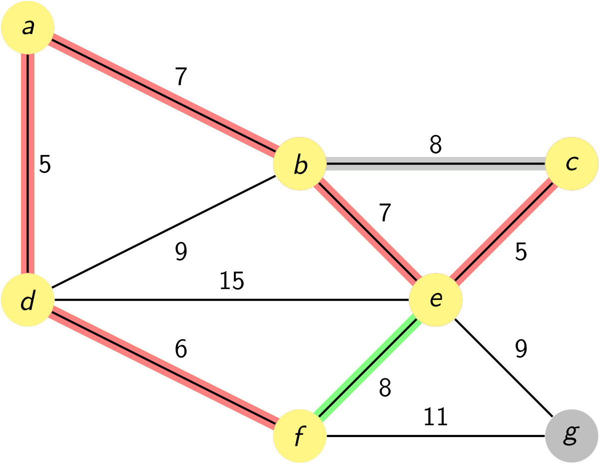
Kruskal's Algorithm Example



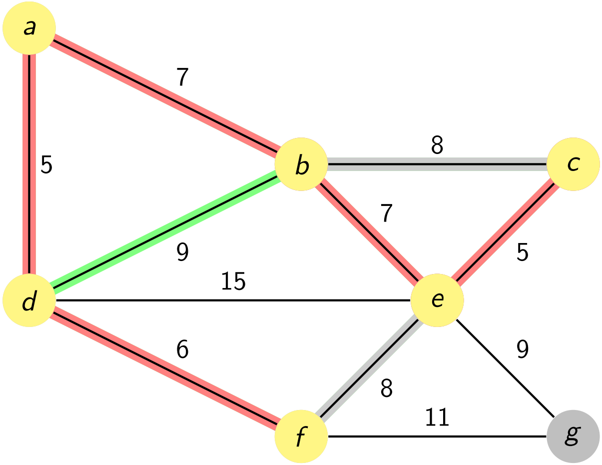
Kruskal's Algorithm Example



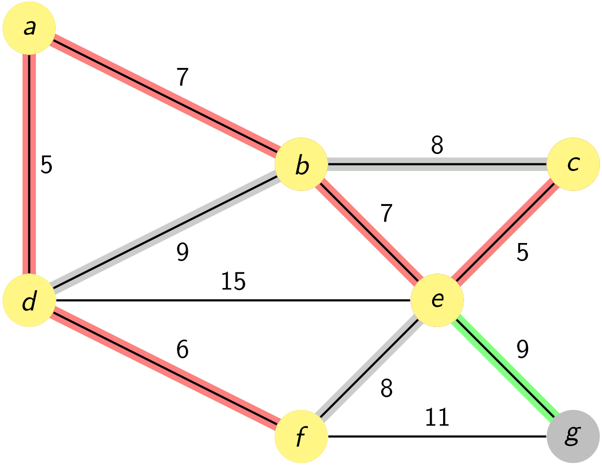
Kruskal's Algorithm Example



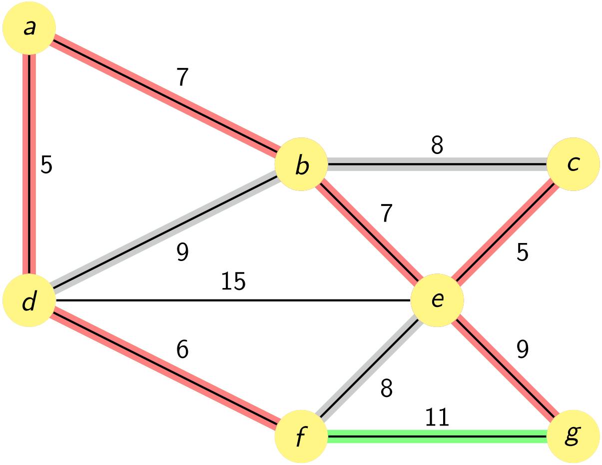
Kruskal's Algorithm Example



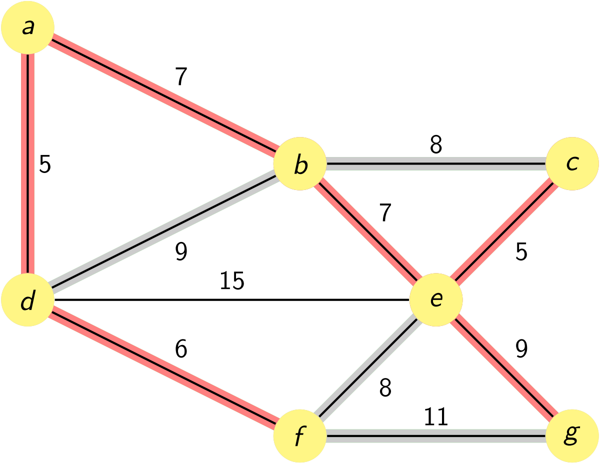
Kruskal's Algorithm Example



Kruskal's Algorithm Example



Kruskal's Algorithm Example



Kruskal's Algorithm Example