Lecture 4b:Depth-First Search

Amitabh Trehan

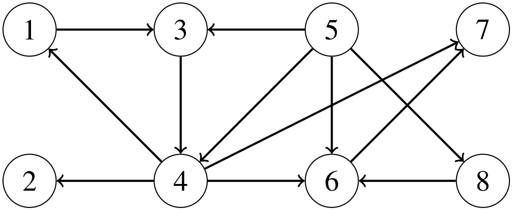
amitabh.trehan@durham.ac.uk

**Based on the slides of ADS-21/22 by Dr. George Mertzios*

Depth-first search

- Like BFS, Depth-first search explores the graph (but does not find distances to the source).
- In contrast to BFS, when a vertex is discovered it is immediately explored.
- Two timestamps are recorded for each vertex, d and f; the discovery and finish times. We can also record predecessors again.
- Again colours are used:
  - white for undiscovered,
  - grey for discovered but not finished, and
  - black for finished.

Example

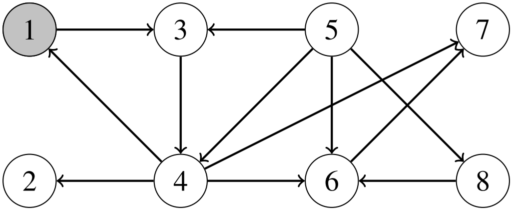


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

	1	2	3	4	5	6	7	8
d	1							
f								

time$=1$

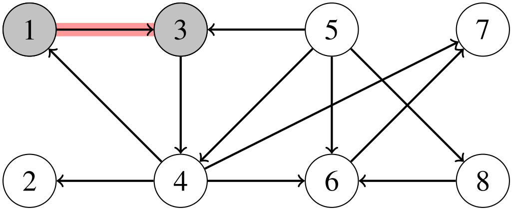


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

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

time= 2

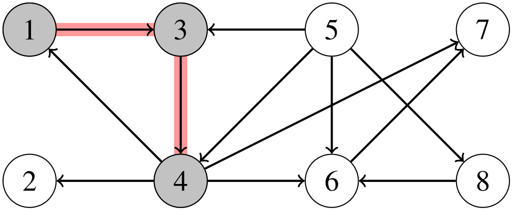


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

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

time= 3

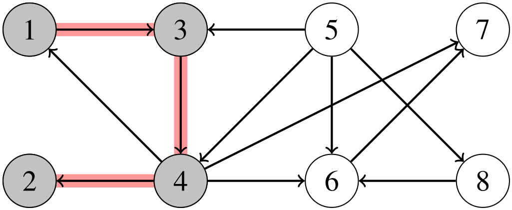


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

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

time= 4

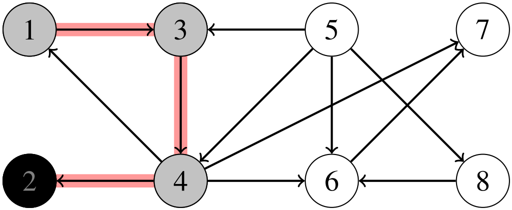


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

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

time= 5

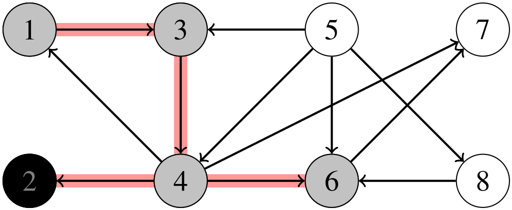


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

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

time= 6

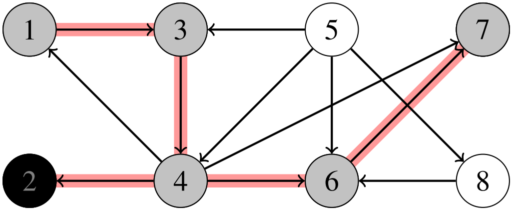


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

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

time= 7

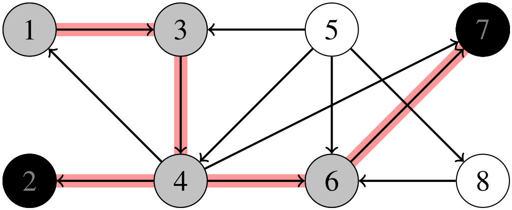


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

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

time= 8

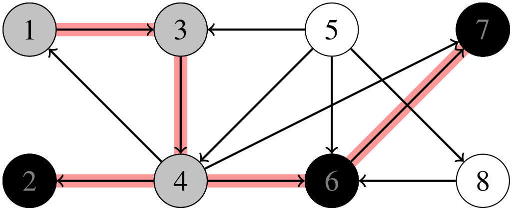


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

	1	2	3	4	5	6	7	8
d	1	4	2	3		6	7	
f		5				9	8	

time= 9

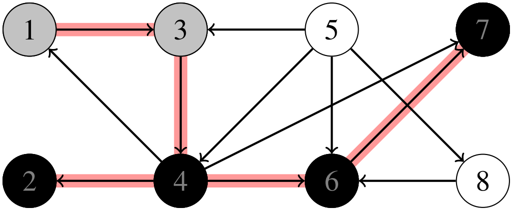


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

	1	2	3	4	5	6	7	8
d	1	4	2	3		6	7	
f		5		10		9	8	

time= 10

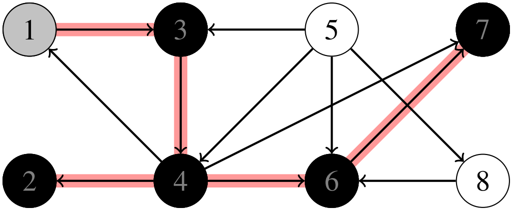


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

	1	2	3	4	5	6	7	8
d	1	4	2	3		6	7	
f		5	11	10		9	8	

time= 11

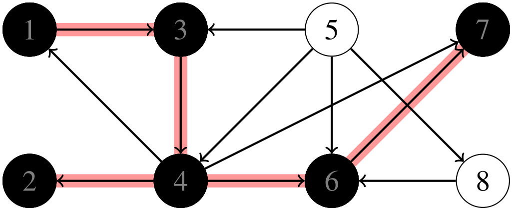


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

	1	2	3	4	5	6	7	8
d	1	4	2	3		6	7	
f	12	5	11	10		9	8	

time= 12

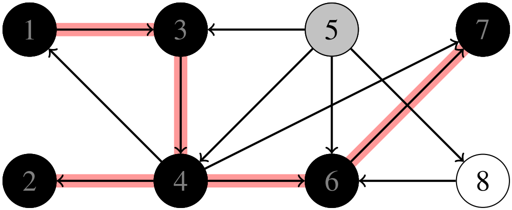


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

	1	2	3	4	5	6	7	8
d	1	4	2	3	13	6	7	
f	12	5	11	10		9	8	

time= 13

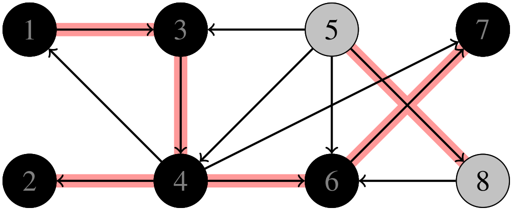


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

	1	2	3	4	5	6	7	8
d	1	4	2	3	13	6	7	14
f	12	5	11	10		9	8	

time= 14

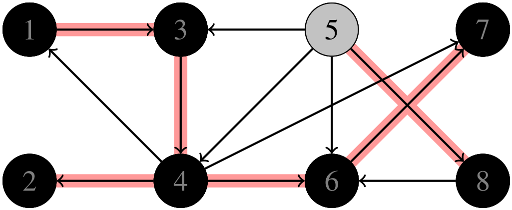


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

	1	2	3	4	5	6	7	8
d	1	4	2	3	13	6	7	14
f	12	5	11	10		9	8	15

time= 15

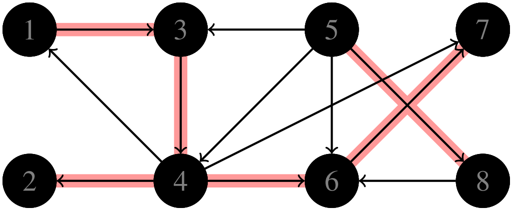


- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Example

	1	2	3	4	5	6	7	8
d	1	4	2	3	13	6	7	14
f	12	5	11	10	16	9	8	15

time= 16



- Initialize: source vertex grey, others white; source discovered at time 1.
- Repeat:
  - Increment the time.
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current.
  - Else colour the current vertex black, note its finish time and return to its predecessor (or jump to an undiscovered vertex), or stop.

Depth-first search

```
DFS (G)
for each vertex u ∈ V[G]
    do colour[u] ← WHITE
       π[u] ← NIL
time ← 0
for each vertex u ∈ V[G]
    do if colour[u] = WHITE
       then DFS-VISIT(u)

DFS-VISIT(u)
colour[u] ← GREY                        [vertex u has just been discovered]
time ← time+1
d[u] ← time
for each vertex v ∈ Adj[u]                              [explore edge (u, v)]
    do if colour[v] = WHITE
       then π[v] ← u
            DFS-VISIT(v)
colour[u] ← BLACK                                     [u has been processed]
f[u] ← time ← time+1
```

Analysis

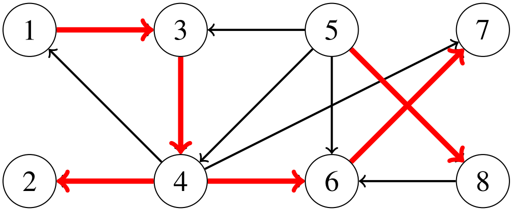
- Initialization takes time $O(V)$.
- Time $O(V)$ is spent on incrementing time, colouring vertices and updating d and f.
- Each vertex in each adjacency list is considered at most once. This takes time $O(E)$.
- Total time is $O(V+E)$.

Analysis

- Initialization takes time $O(V)$.
- Time $O(V)$ is spent on incrementing time, colouring vertices and updating d and f.
- Each vertex in each adjacency list is considered at most once. This takes time $O(E)$.
- Total time is $O(V+E)$.

The edges used for discovering new vertices form the depth-first tree (or forest). Again, we can find this with a predecessor array.

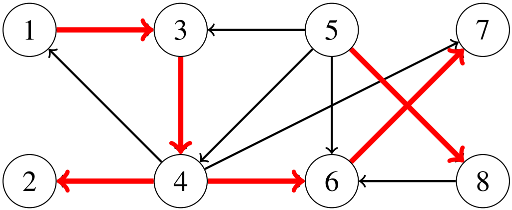
Example



Once we have run DFS on a graph we can construct the predecessor subgraph. This has the same vertex set as the graph, and for each vertex v there is an edge from the predecessor of v to v.

The predecessor subgraph is a depth-first forest.

Example



Once we have run DFS on a graph we can construct the predecessor subgraph. This has the same vertex set as the graph, and for each vertex v there is an edge from the predecessor of v to v.

The predecessor subgraph is a depth-first forest.

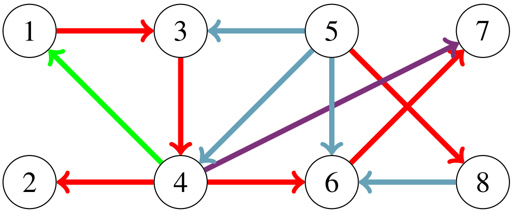
How many trees do we have in the above forest?

Classification of the edges

Once we have obtained a DFS-forest for a graph G, we can classify the edges of G.

- Tree edges are those edges in the DFS-forest.
- Back edges are edges that join a vertex to an ancestor.
- Forward edges are edges not in the tree that join a vertex to its descendant.
- Cross edges: all other edges.

Example



Tree edges

Forward edges

Back edges

Cross edges

Classification of the edges

The classification is ambiguous for undirected graphs (back edges and forward edges are the same thing),

Classification of the edges

The classification is ambiguous for undirected graphs (back edges and forward edges are the same thing),

Let us refine the definition: suppose that e is an edge that joins a vertex u to its descendant v.

- e is a forward edge if DFS first considers e from u.
- e is a back edge if DFS first considers e from v.

Classification of the edges

The classification is ambiguous for undirected graphs (back edges and forward edges are the same thing),

Let us refine the definition: suppose that e is an edge that joins a vertex u to its descendant v.

- e is a forward edge if DFS first considers e from u.
- e is a back edge if DFS first considers e from v.

Theorem

In an undirected graph, every edge is a tree edge or a back edge.

Classification of the edges

The classification is ambiguous for undirected graphs (back edges and forward edges are the same thing),

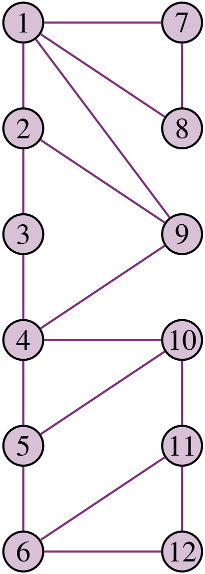
Let us refine the definition: suppose that e is an edge that joins a vertex u to its descendant v.

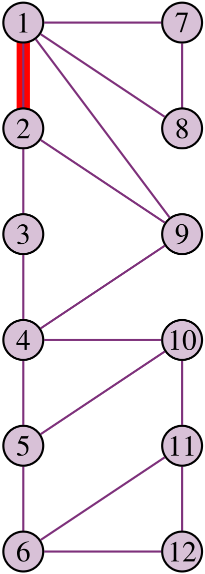
- e is a forward edge if DFS first considers e from u.
- e is a back edge if DFS first considers e from v.

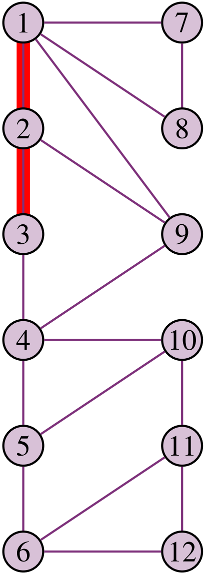
Theorem

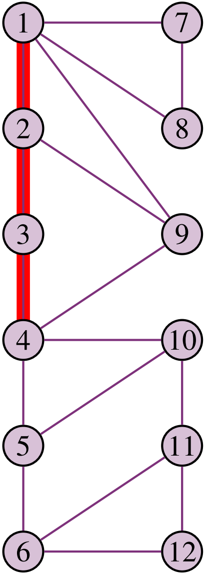
In an undirected graph, every edge is a tree edge or a back edge.

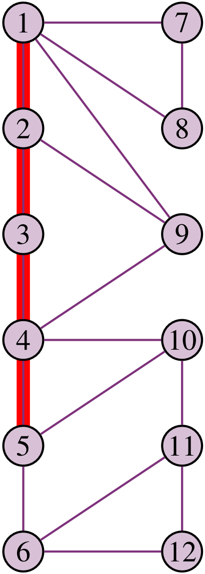
Questions: Which types of edges can be found in a directed graph? What if BFS is used?

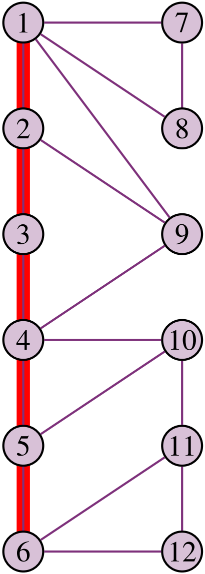

1
7
2
8
3
9
4
10
5
11
6
12

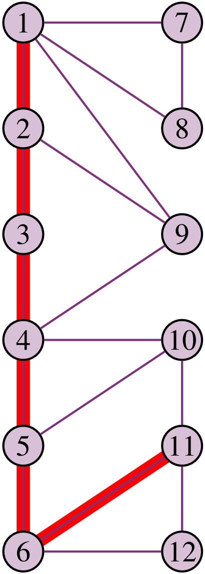

1
7
2
8
3
9
4
10
5
11
6
12

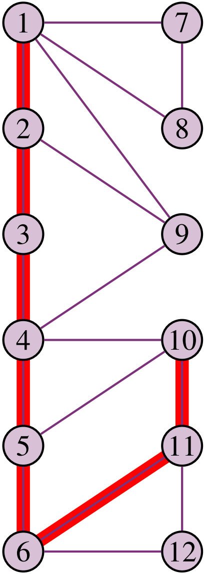

1
7
2
8
3
9
4
10
5
11
6
12

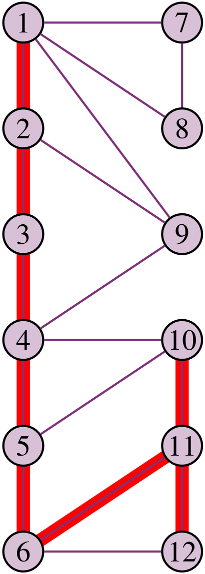

1
7
2
8
3
9
4
10
5
11
6
12

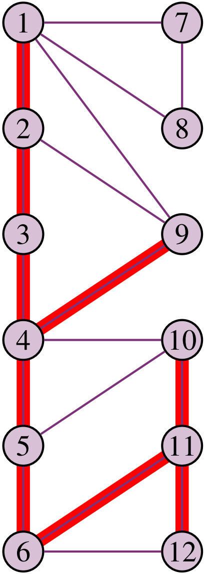

1
7
2
8
3
9
4
10
5
11
6
12

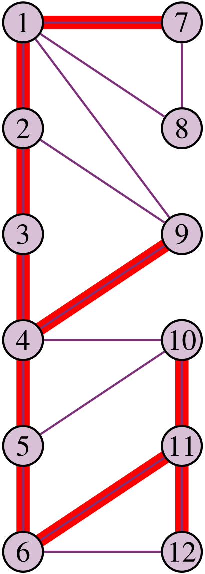

1
7
2
8
3
9
4
10
5
11
6
12

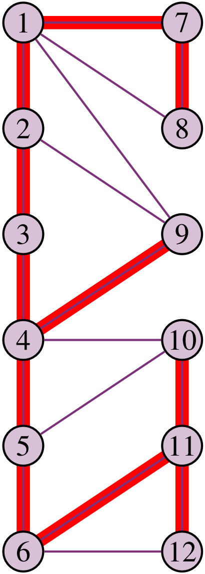

1
7
2
8
3
9
4
10
5
11
6
12

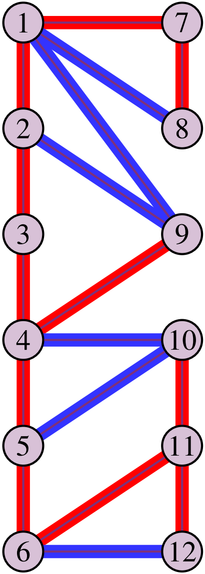

1
7
2
8
3
9
4
10
5
11
6
12

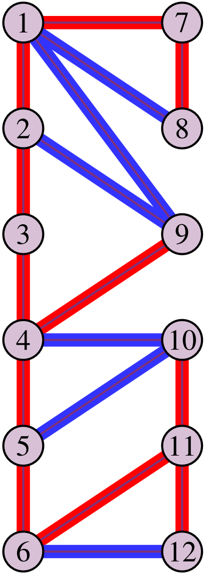

1
7
2
8
3
9
4
10
5
11
6
12


1
7
2
8
3
9
4
10
5
11
6
12

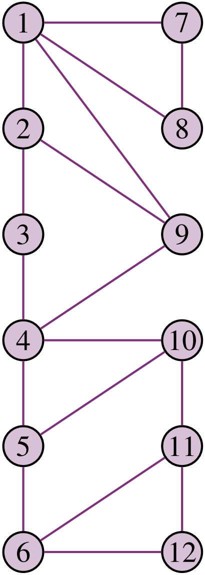

1
7
2
8
3
9
4
10
5
11
6
12


1
7
2
8
3
9
4
10
5
11
6
12

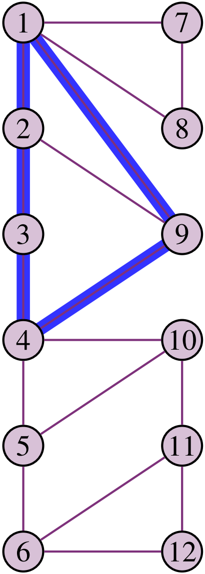

1
7
2
8
3
9
4
10
5
11
6
12



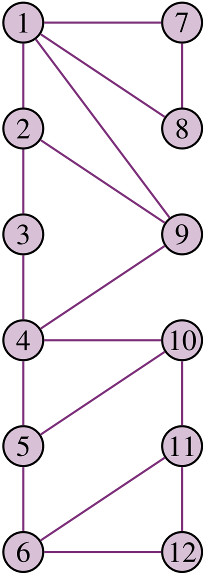
- Every edge in an undirected graph is either a tree edge or a back edge.



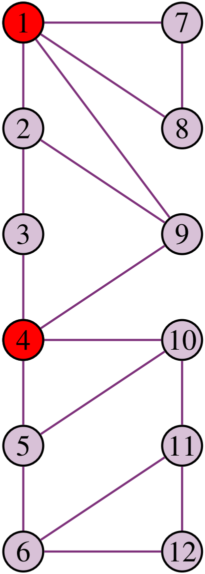
- Every edge in an undirected graph is either a tree edge or a back edge.
- A graph is connected if each pair of vertices is joined by a path.



- Every edge in an undirected graph is either a tree edge or a back edge.
- A graph is connected if each pair of vertices is joined by a path.
- A cycle is a sequence of edges that start and end at the same vertex.



- Every edge in an undirected graph is either a tree edge or a back edge.
- A graph is connected if each pair of vertices is joined by a path.
- A cycle is a sequence of edges that start and end at the same vertex.
- An articulation point is a vertex whose removal disconnects the graph.



- Every edge in an undirected graph is either a tree edge or a back edge.
- A graph is connected if each pair of vertices is joined by a path.
- A cycle is a sequence of edges that start and end at the same vertex.
- An articulation point is a vertex whose removal disconnects the graph.

Using Depth-First Search

Can we adapt Depth-First Search to obtain algorithms that

- check whether a graph is connected?
- discover a cycle in a graph (or conclude that none exists)?
- find all the articulation points in a graph?