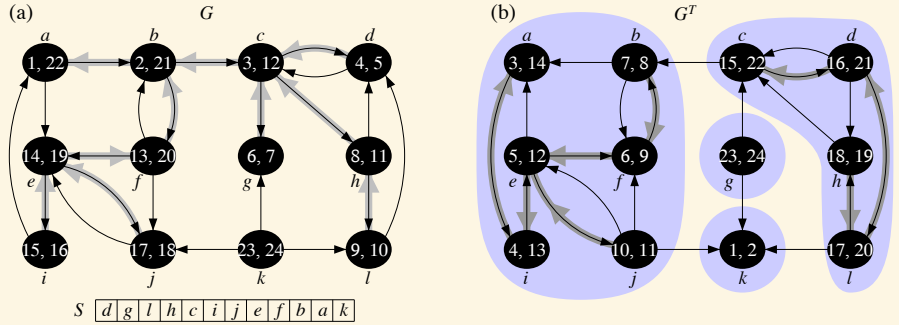
Lesson 02: Depth-First Search and Identifying Strongly Connected Components

Michael T. Gastner (28 February 2023)



Disclaimer: These slides are based on and occasionally quote from 'Introduction to Algorithms' (3rd ed.) by Cormen et al. (2009), MIT Press.

Review of Previous Lesson

Introduction to Graphs and Breadth-First Search

Here are the main takeaways from the previous lesson:

- A graph $G = (V, E)$ is a mathematical representation of a network, where V is a set of vertices and E is a set of edges (i.e. pairs of vertices).
- Graphs can be directed or undirected.
- A graph is called simple if E does not contain self-loops or parallel edges. In this course, we assume that graphs are simple unless stated otherwise.
- Graphs can be used to model data sets across various domains (e.g. technology, transport and social networks).
- Graphs can be represented by adjacency lists or adjacency matrices.
- Breadth-first search is an algorithm to determine whether a vertex v is reachable from a vertex u. If yes, then the output of breadth-first search can be used to determine a shortest path from u to v.

2 / 21

Learning Objectives

By the end of this lesson, you should be able to . . .

- Recall graph-related terminology (e.g. depth-first tree, strongly connected component).
- Apply depth-first search to a given graph.
- State the growth of the running time of depth-first search as a function of the number of vertices and the number of edges.
- Determine strongly connected components algorithmically.

3 / 21

Depth-First Search

Graph Traversal Algorithm

Like breadth-first search, depth-first search starts exploring a graph by discovering a source vertex. Afterwards, both algorithms repeatedly step along an edge from an already discovered vertex to a still undiscovered one.

Unlike breadth-first search, depth-first search proceeds by exploring from the most recently (rather than the earliest) discovered vertex that has an edge to an undiscovered vertex.

4 / 21

Vertex Attributes in Depth-First Search

Parent and Timestamps

When a vertex v is discovered from a vertex u, depth-first search sets the following two attributes:

- $v.\pi = u$ to indicate that u is the **parent** of v.
- A timestamp $v.d$ equal to v's **discovery time**.

Once v is discovered, $v.\pi$ and $v.d$ will never change again.

At a later stage, v receives a timestamp $v.f$ that records v's **finishing time**.

5 / 21

Vertex Attributes in Depth-First Search

Colours

Each vertex v has one of the following **colours**:

- $v.colour = \text{WHITE}$ while v is still undiscovered.
- $v.colour = \text{GREY}$ when v is discovered but not yet finished.
- $v.colour = \text{BLACK}$ after v is finished.

At the start of a depth-first search, all vertices are WHITE.

At the end, all vertices are BLACK.

6 / 21

Depth-First Search in Pseudocode

Recursive Implementation

In the following pseudocode, $time$ is a global variable used for timestamping.

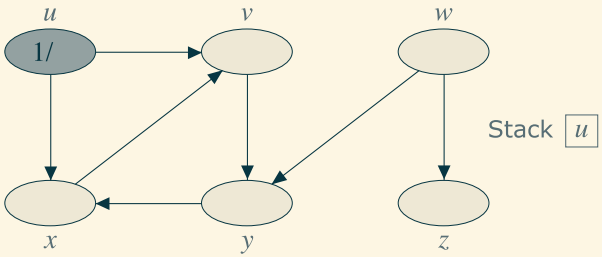
```
DFS(G)
1  for each vertex u ∈ G.V
2      u.colour = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.colour == WHITE
7          DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.colour = GREY
 4  for each v ∈ G.Adj[u]
 5      if v.colour == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.colour = BLACK
 9  time = time + 1
10  u.f = time
```


7 / 21

Example
Directed Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

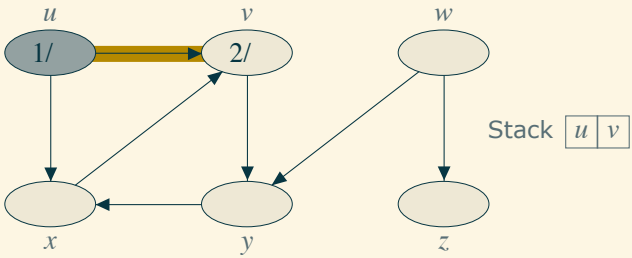


Stack $\boxed{u}$

8 / 21

Example
Directed Graph

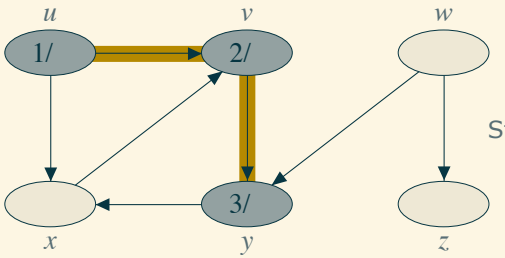
1st number inside vertex = Discovery time
2nd number = Finishing time



8 / 21

Example
Directed Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

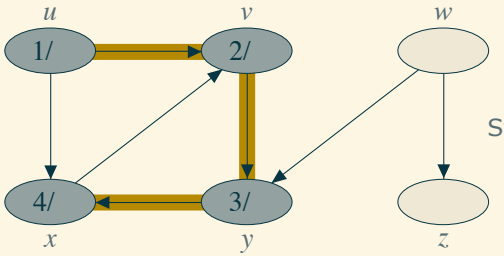


Stack | u | v | y |

8 / 21

Example
Directed Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

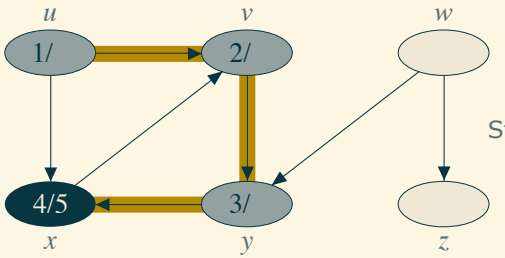


Stack | u | v | y | x |

8 / 21

Example
Directed Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

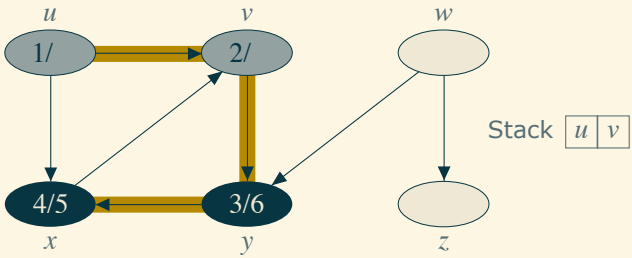


Stack | u | v | y |

8 / 21

Example
Directed Graph

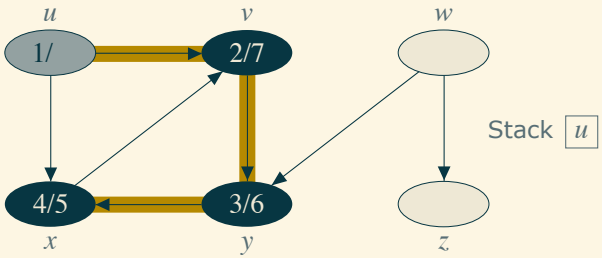
1st number inside vertex = Discovery time
2nd number = Finishing time



8 / 21

Example
Directed Graph

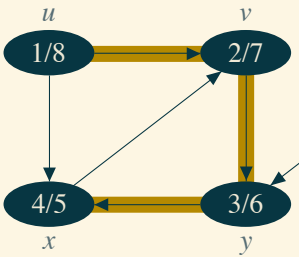
1st number inside vertex = Discovery time
2nd number = Finishing time



8 / 21

Example
Directed Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

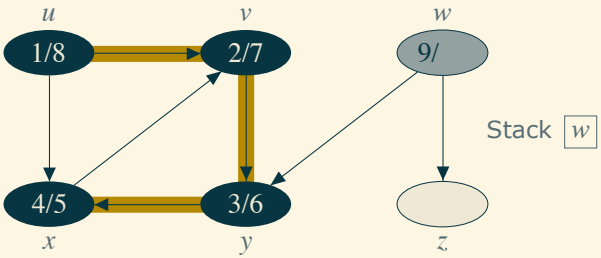


Stack empty

8 / 21

Example
Directed Graph

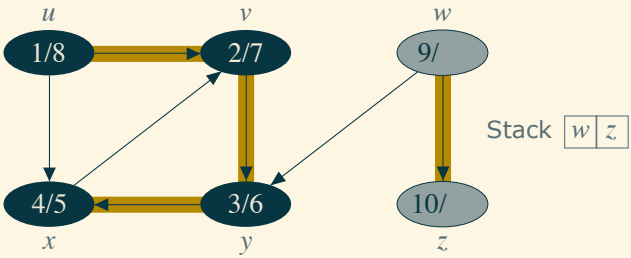
1st number inside vertex = Discovery time
2nd number = Finishing time



8 / 21

Example
Directed Graph

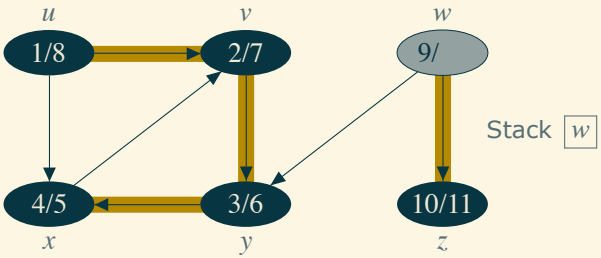
1st number inside vertex = Discovery time
2nd number = Finishing time



8 / 21

Example
Directed Graph

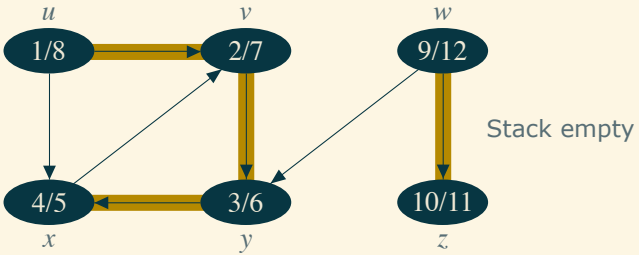
1st number inside vertex = Discovery time
2nd number = Finishing time



8 / 21

Example
Directed Graph

1st number inside vertex = Discovery time
2nd number = Finishing time



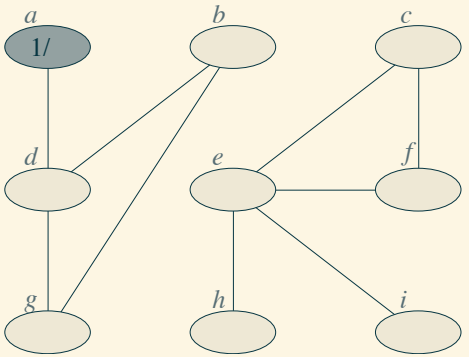
The **depth-first forest** consists of the edges from parents to their child vertices (light brown in this illustration). Here, the depth-first consists of two **depth-first trees**: $u \to x \to y \to x$ and $w \to z$.

8 / 21

Example

Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time



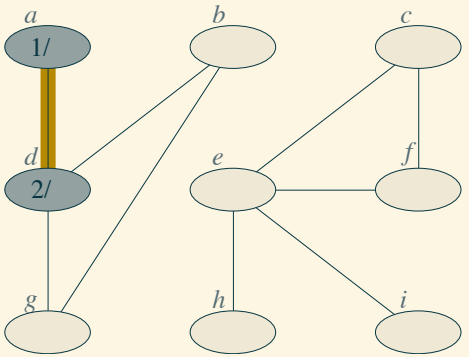
Stack $\boxed{a}$

9 / 21

Example

Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time



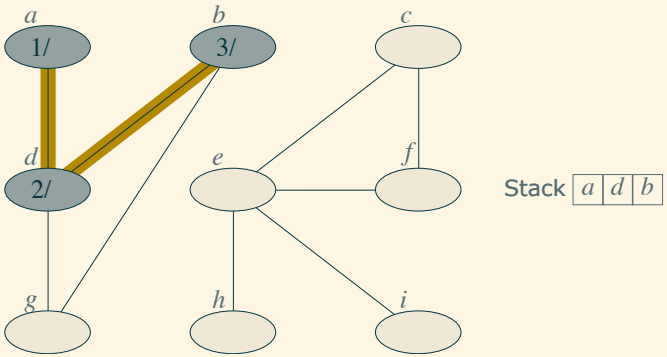
Stack | a | d |

9 / 21

Example

Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

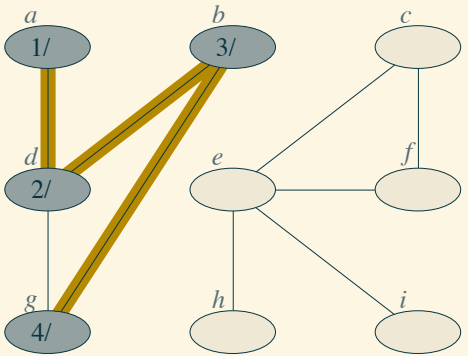


9 / 21

Example

Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time



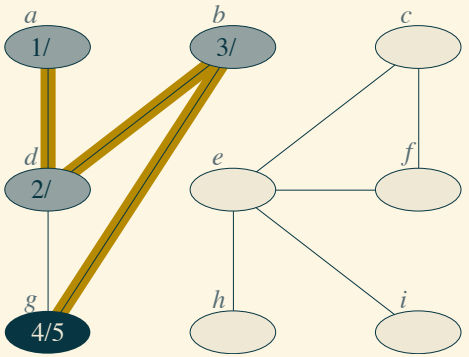
Stack | a | b | d | g |

9 / 21

Example

Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time



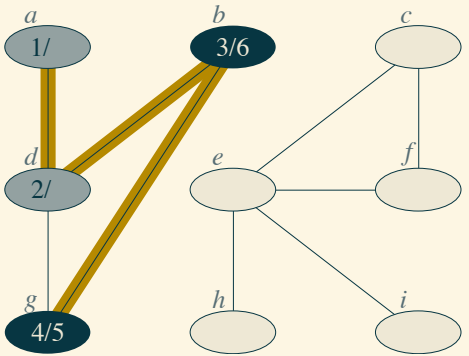
Stack $\boxed{a}\boxed{\not{d}}\boxed{\not{b}}$

9 / 21

Example

Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time



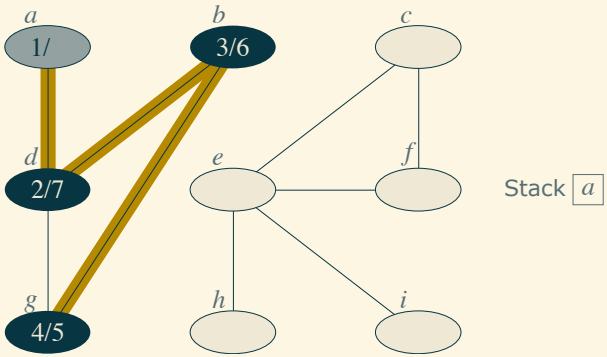
Stack $\boxed{a}\boxed{\not{d}}$

9 / 21

Example

Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

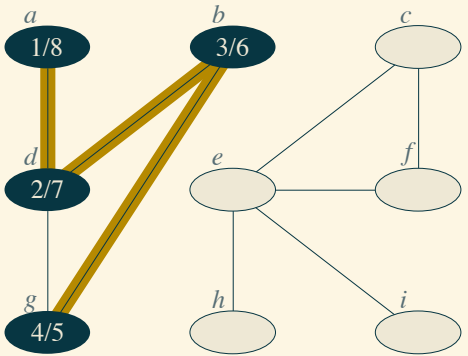


Stack $\boxed{a}$

9 / 21

Example
Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

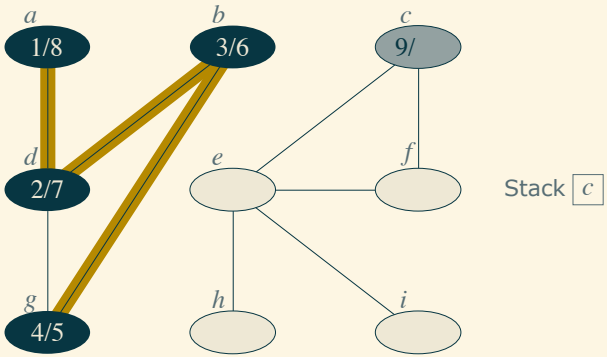


Stack empty

9 / 21

Example
Undirected Graph

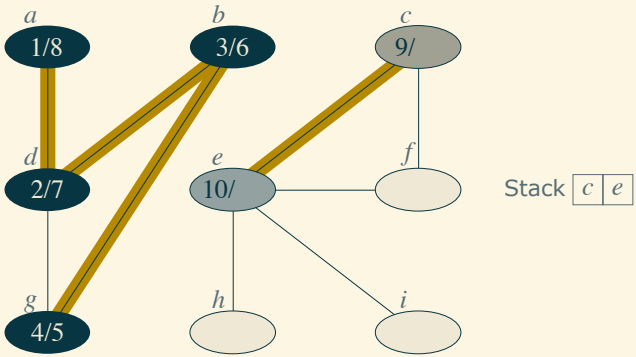
1st number inside vertex = Discovery time
2nd number = Finishing time



9 / 21

Example
Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

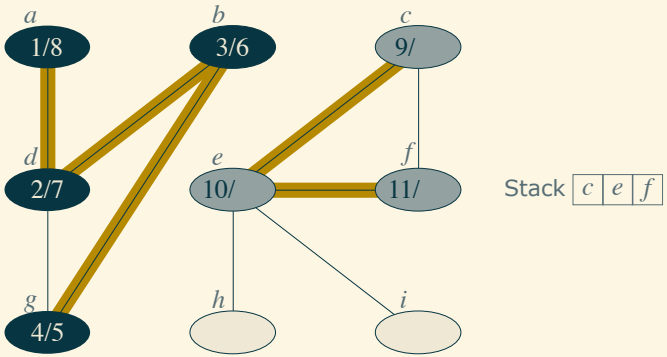


9 / 21

Example

Undirected Graph

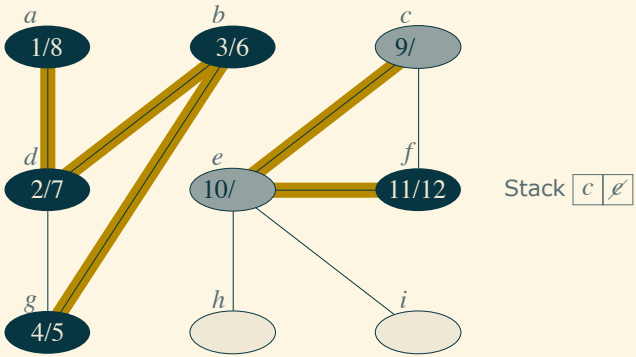
1st number inside vertex = Discovery time
2nd number = Finishing time



9 / 21

Example
Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

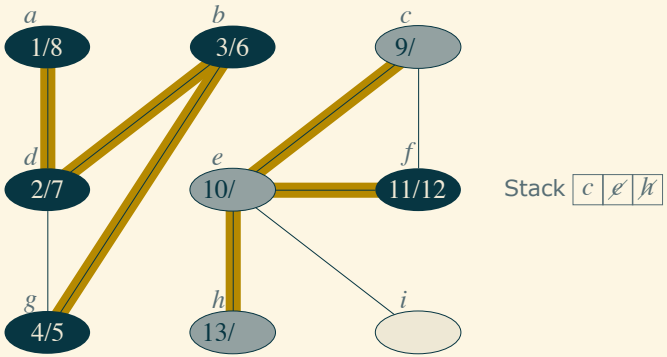


Stack c $\not{e}$

9 / 21

Example
Undirected Graph

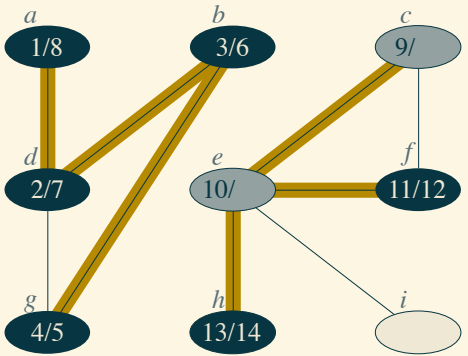
1st number inside vertex = Discovery time
2nd number = Finishing time



9 / 21

Example
Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

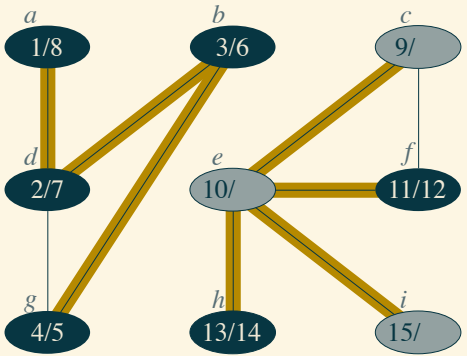


Stack c $\not{e}$

9 / 21

Example
Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time

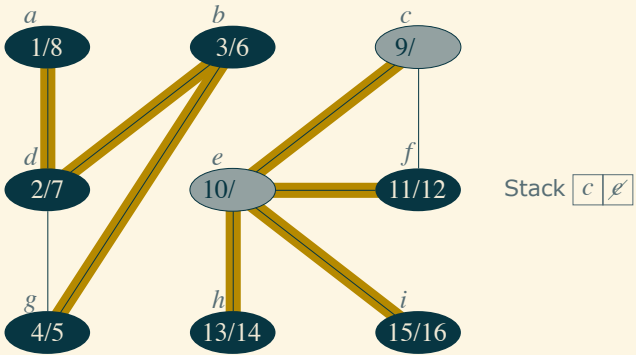


Stack c $\not{e}$ $\not{i}$

9 / 21

Example
Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time



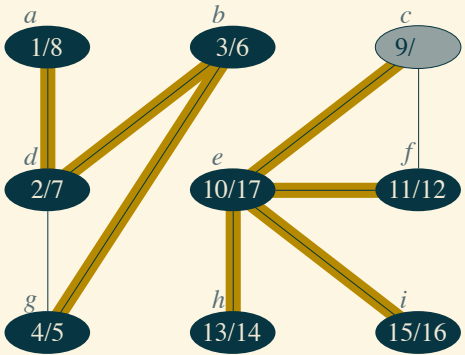
Stack c $\not{e}$

9 / 21

Example

Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time



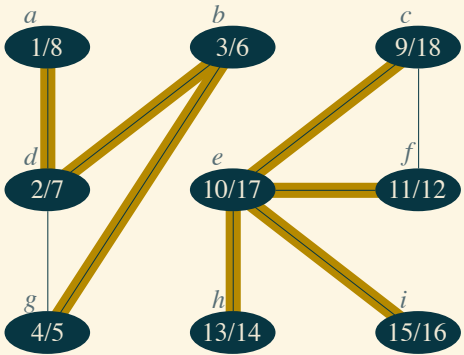
Stack $\boxed{c}$

9 / 21

Example

Undirected Graph

1st number inside vertex = Discovery time
2nd number = Finishing time



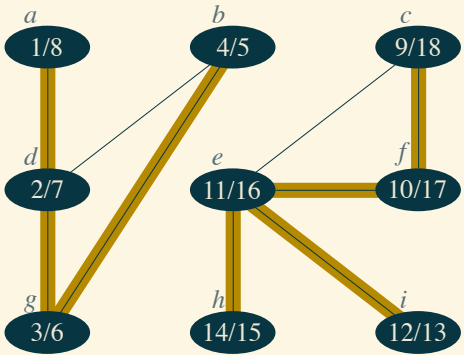
Stack empty

9 / 21

Depth-First Trees are Not Unique

They Depend on the Order of Vertices in the Adjacency List

Alternative traversal order results in different depth-first trees.



10 / 21

Running Time of Depth-First Search

Aggregate Analysis

We analyse the ⏩ pseudocode to determine the running time:

- The loops on lines 1–3 and 5–7 of DFS take time $\Theta(V)$, exclusive of the time to execute the calls to DFS-VISIT.
- The procedure DFS-VISIT is called exactly once for each vertex because the vertex u on which DFS-VISIT is invoked must be white and the first thing DFS-VISIT does is paint vertex u grey.
- During an execution of DFS-VISIT, the loop on lines 4–7 executes $|Adj[v]|$ times. In the previous lesson, we learnt that $\sum_{v \in V} |Adj[v]| = \Theta(E)$. Therefore, the aggregate cost of executing lines 4–7 of DFS-VISIT during a run of DFS is $\Theta(E)$.

In summary, the running time of DFS is $\Theta(V + E)$.

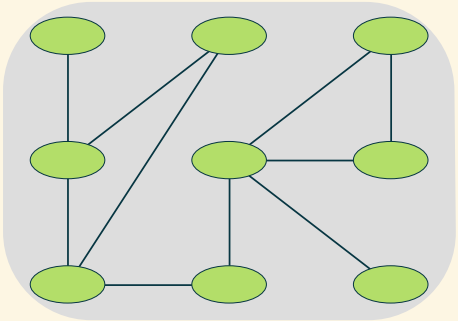
11 / 21

Connectedness of an Undirected Graph

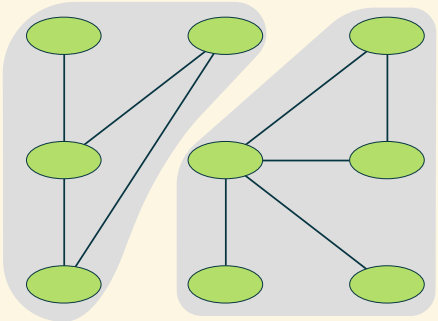
Every Vertex Is Reachable From All Other Vertices

An undirected graph is called **connected** if there exists a path between any two vertices in the graph.

Connected



Disconnected



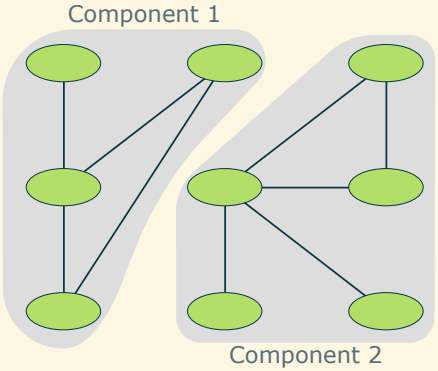
12 / 21

Connected Components in an Undirected Graph

Maximal Connected Subgraphs

A **connected component** (or simply a **component**) of an undirected graph is a subgraph that is connected and not a part of any larger connected subgraph.

Therefore, for any two vertices u and v in the graph, u belongs to the same connected component as v if and only if there exists a path from v to u.



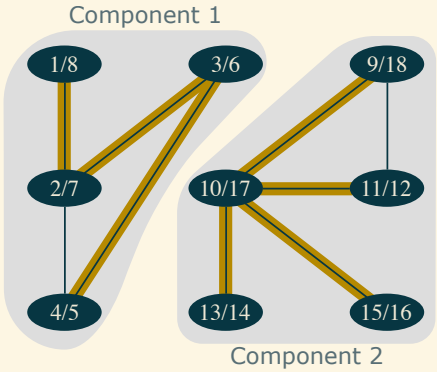
13 / 21

How to Find Connected Components in an Undirected Graph

Analyse Depth-First Forest

In an undirected graph, vertices that are in the same connected component are also in the same depth-first search tree of any depth-search forest.

Conversely, if two vertices are in the same depth-first tree, then they are also in the same connected component.



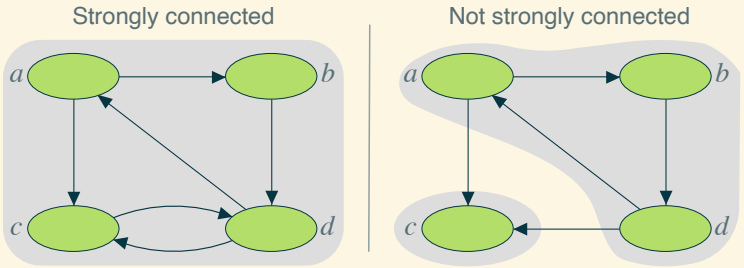
14 / 21

Strong Connectedness of a Directed Graph

Every Vertex is Reachable From All Other Vertices via Directed Paths

A directed graph is called **strongly connected** if there exists a directed path between every ordered pair of vertices in the graph.

The graph shown below on the right is not strongly connected because, for example, vertex a is not reachable from vertex c.



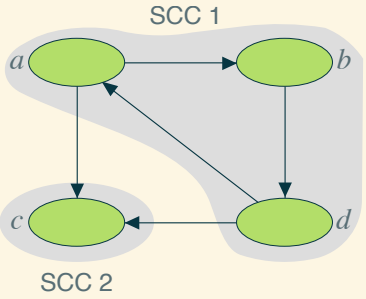
15 / 21

Strongly Connected Components in a Directed Graph

Maximal Strongly Connected Subgraphs

A **strongly connected component** (SCC) of a directed graph is a subgraph that is strongly connected and not part of any larger strongly connected subgraph.

Therefore, for any two vertices u and v in the graph, u belongs to the same SCC as v if and only if there exist paths from u to v and from v to u.



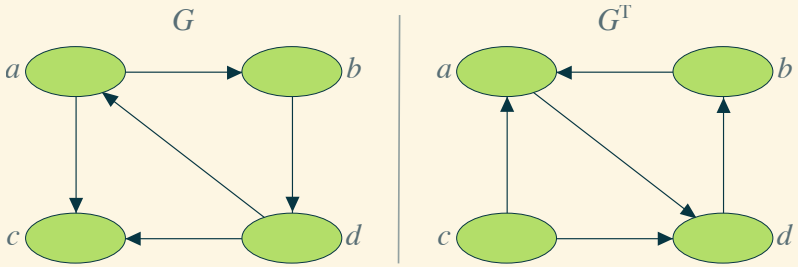
16 / 21

Transpose of a Directed Graph

Reversing all edges

Next, we want to learn how to determine the strongly connected components in a directed graph. Here is a useful auxiliary object.

The **transpose** of a directed graph $G = (V, E)$ is the graph $G^{\mathrm{T}} = (V, E^{\mathrm{T}})$ with the edge set $E^{\mathrm{T}} = \{(u, v) : (v, u) \in E\}$.



17 / 21

Determining Strongly Connected Components

Pseudocode

STRONGLY-CONNECTED-COMPONENTS(G)

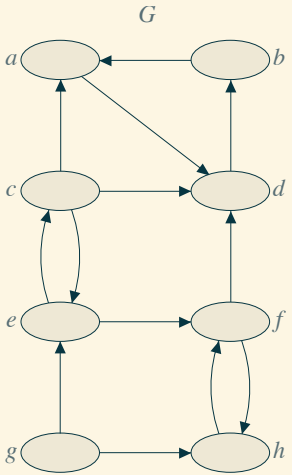
1. Call DFS(G) to compute finishing time $u.f$ for each vertex u.
2. Compute G^{T}.
3. Call DFS(G^{T}), but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).
4. Output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component.

It is possible to implement STRONGLY-CONNECTED-COMPONENTS such that its running time is $\Theta(V + E)$.

18 / 21

Determining Strongly Connected Components

Example 1

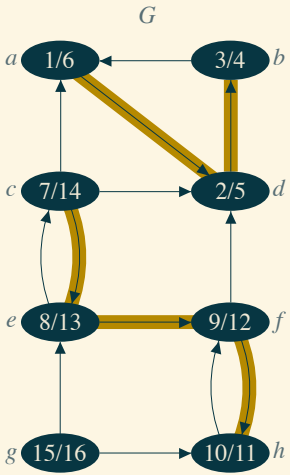


19 / 21

Determining Strongly Connected Components

Example 1

1. Call $\mathrm{DFS}(G)$ **to compute finishing time** $u.f$ **for each vertex** u.



Vertices in ascending order of finishing time

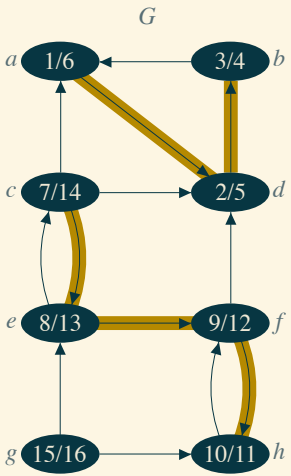
b	d	a	h	f	e	c	g

19 / 21

Determining Strongly Connected Components

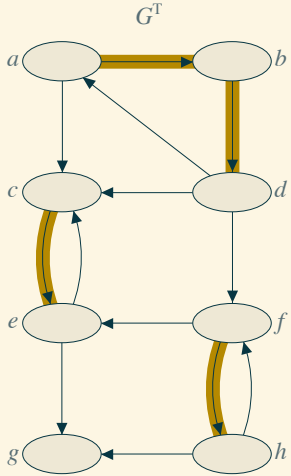
Example 1

2. Compute G^{T}.



Vertices in ascending order of finishing time

b	d	a	h	f	e	c	g

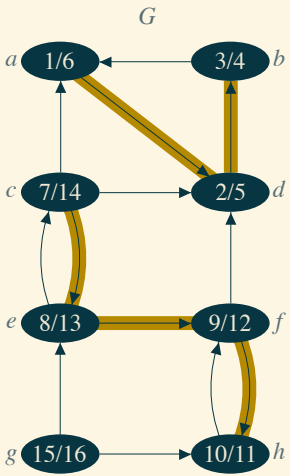


19 / 21

Determining Strongly Connected Components

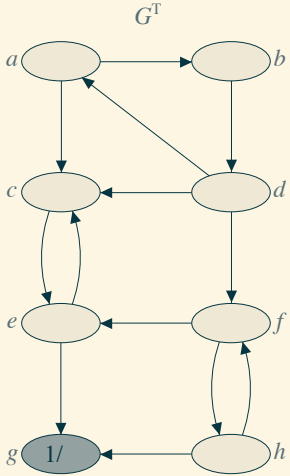
Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).



Vertices in ascending order of finishing time

b	d	a	h	f	e	c	g

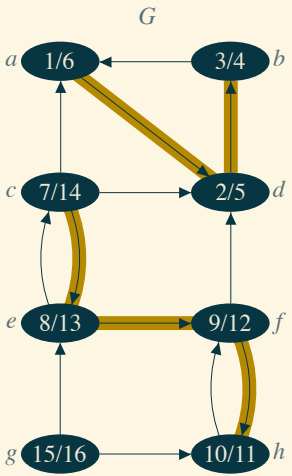


19 / 21

Determining Strongly Connected Components

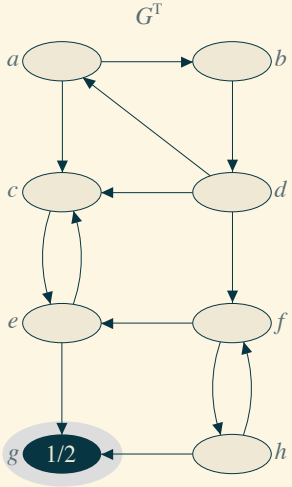
Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).



Vertices in ascending order of finishing time

b	d	a	h	f	e	c	g

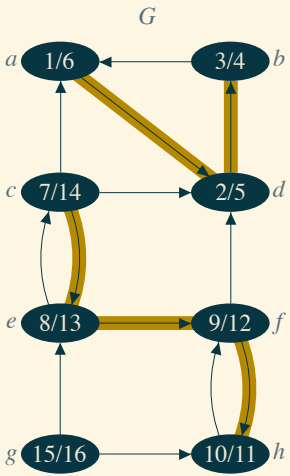


19 / 21

Determining Strongly Connected Components

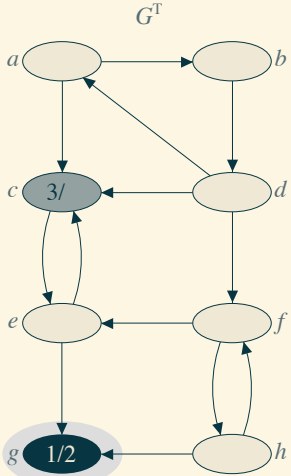
Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).



Vertices in ascending order of finishing time

b	d	a	h	f	e	c	g

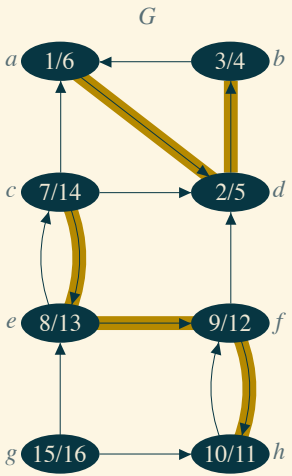


19 / 21

Determining Strongly Connected Components

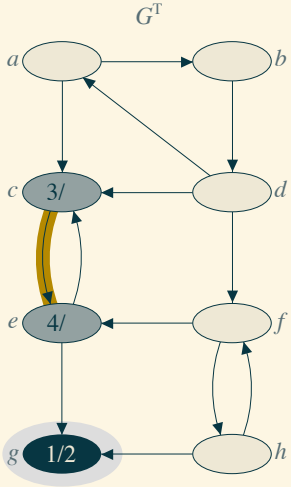
Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).



Vertices in ascending order of finishing time

b	d	a	h	f	e	c	g

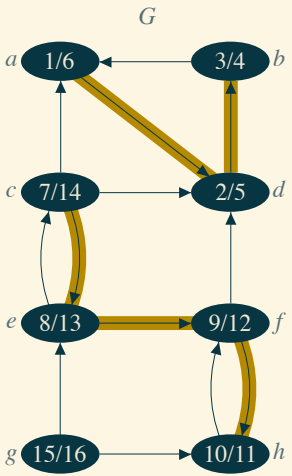


19 / 21

Determining Strongly Connected Components

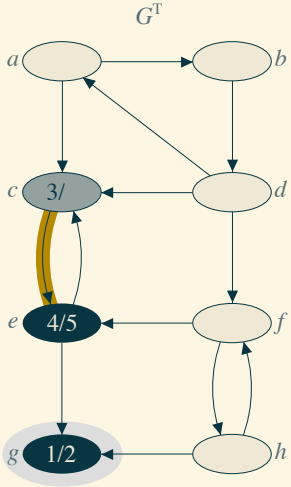
Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).



Vertices in ascending order of finishing time

b	d	a	h	f	e	c	g

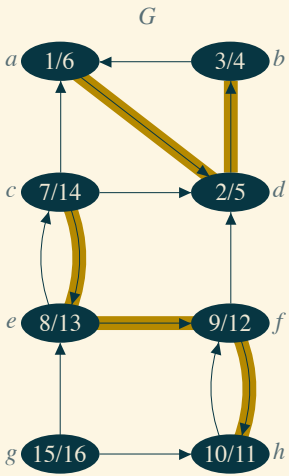


19 / 21

Determining Strongly Connected Components

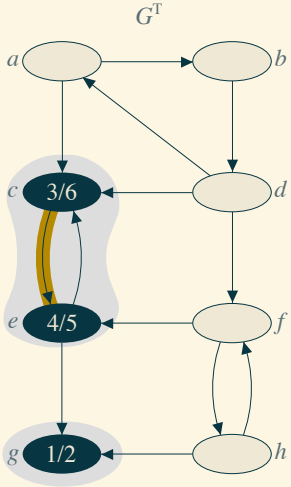
Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).



Vertices in ascending order of finishing time

b	d	a	h	f	e	c	g

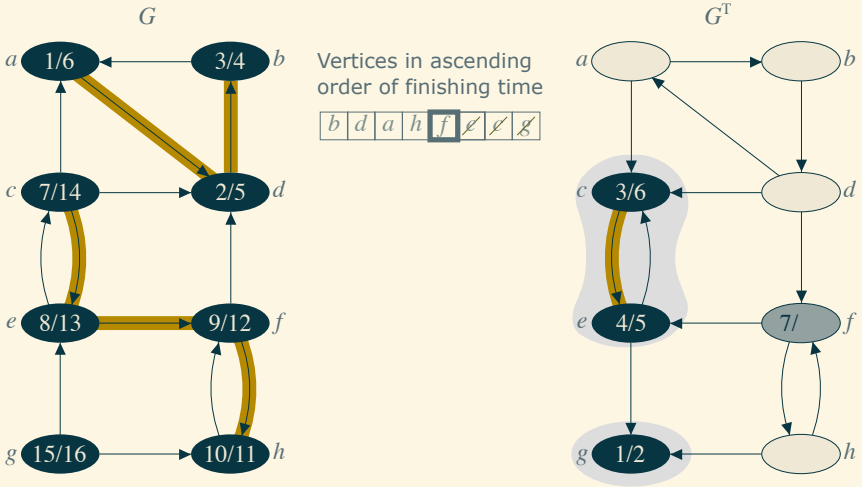


19 / 21

Determining Strongly Connected Components

Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).

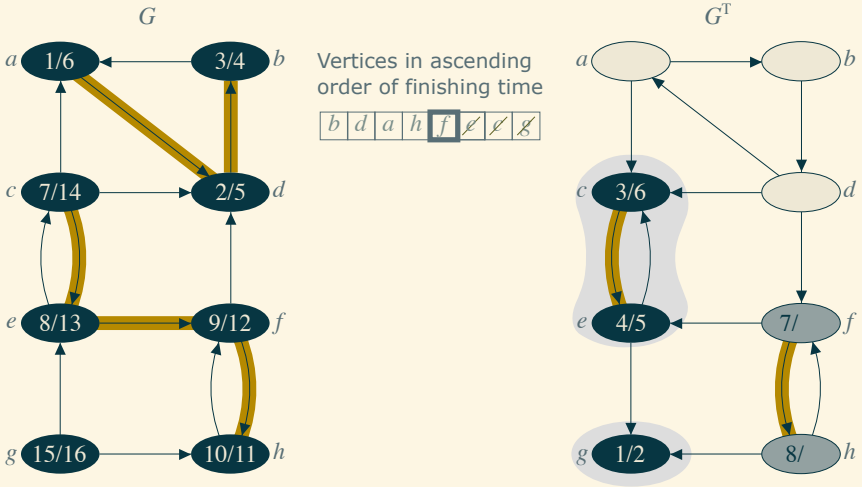


19 / 21

Determining Strongly Connected Components

Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).

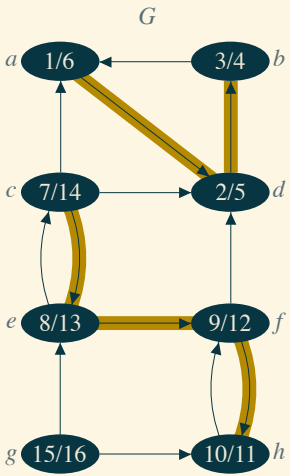


19 / 21

Determining Strongly Connected Components

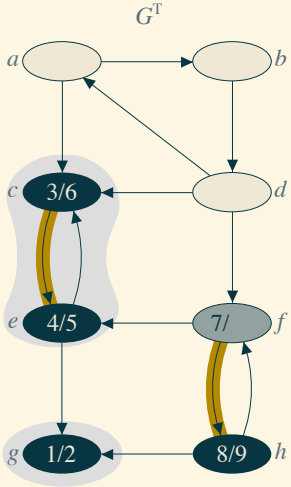
Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).



Vertices in ascending order of finishing time

b	d	a	h	f	✓	✓	✓

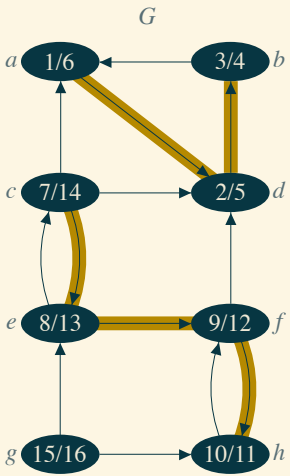


19 / 21

Determining Strongly Connected Components

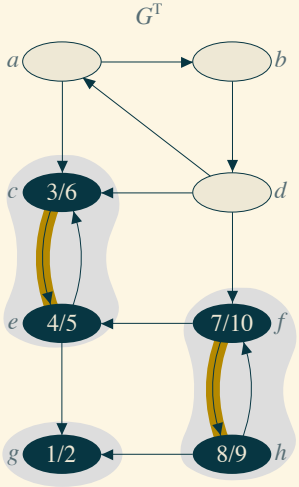
Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).



Vertices in ascending order of finishing time

b	d	a	h	f	$\not{e}$	$\not{c}$	$\not{g}$

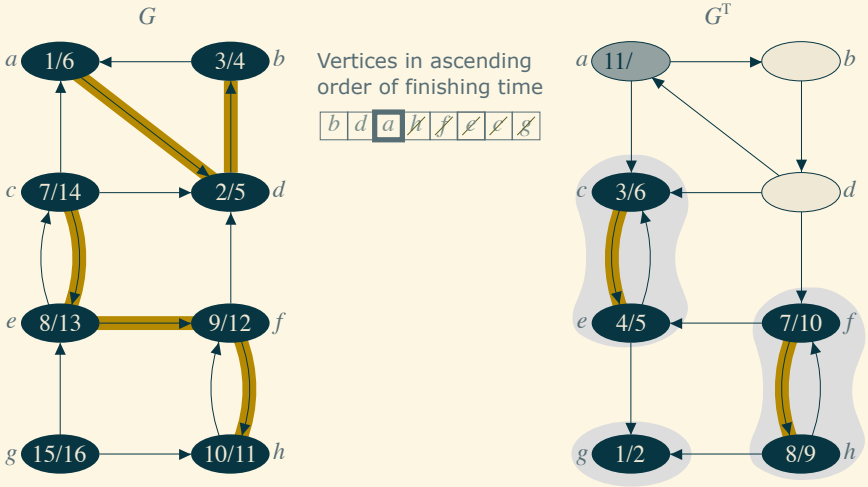


19 / 21

Determining Strongly Connected Components

Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).

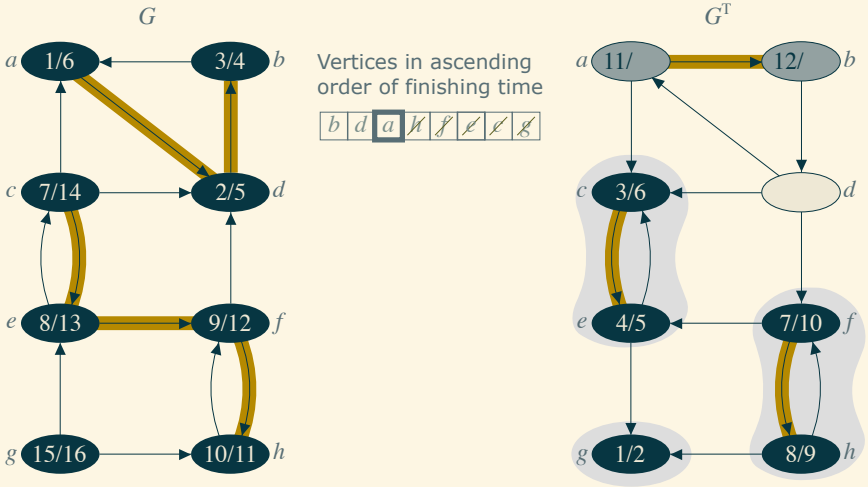


19 / 21

Determining Strongly Connected Components

Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).

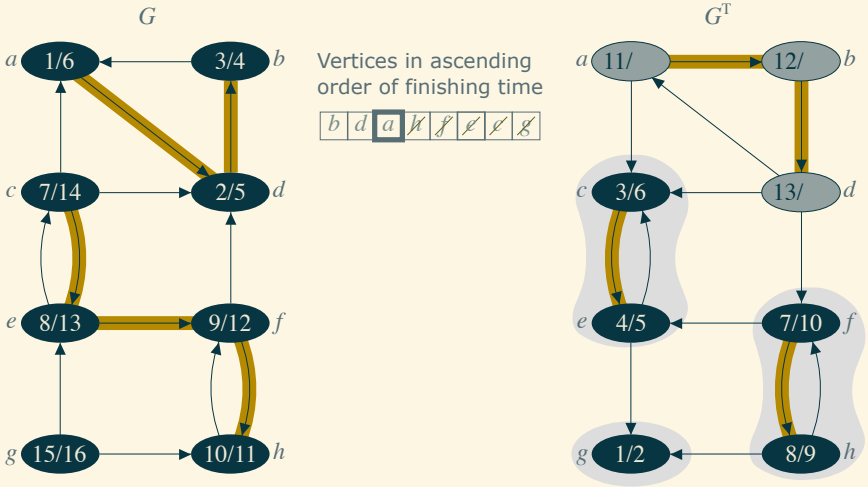


19 / 21

Determining Strongly Connected Components

Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$**, but, in the main loop of** DFS**, consider the vertices in order of decreasing** $u.f$ (as computed in line 1).

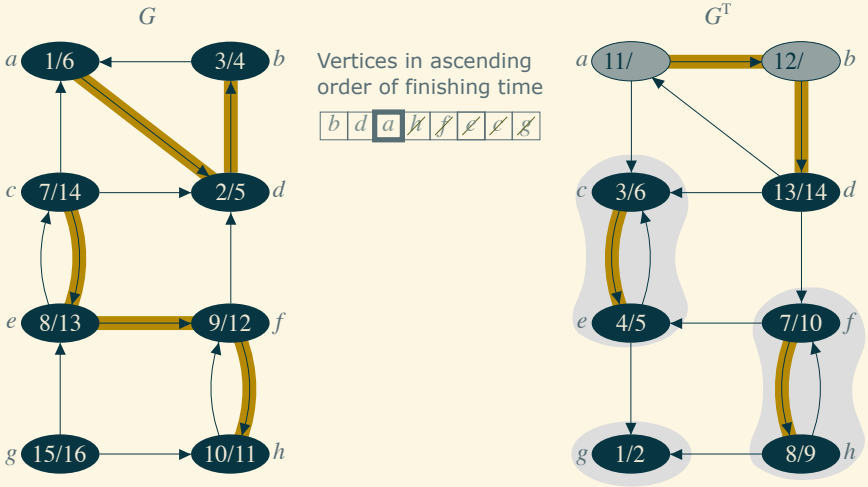


19 / 21

Determining Strongly Connected Components

Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).

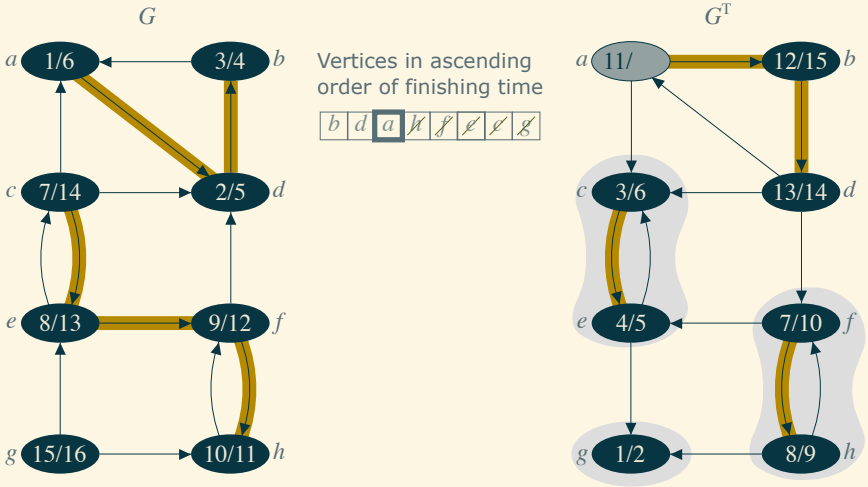


19 / 21

Determining Strongly Connected Components

Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$**, but, in the main loop of** DFS**, consider the vertices in order of decreasing** $u.f$ (as computed in line 1).

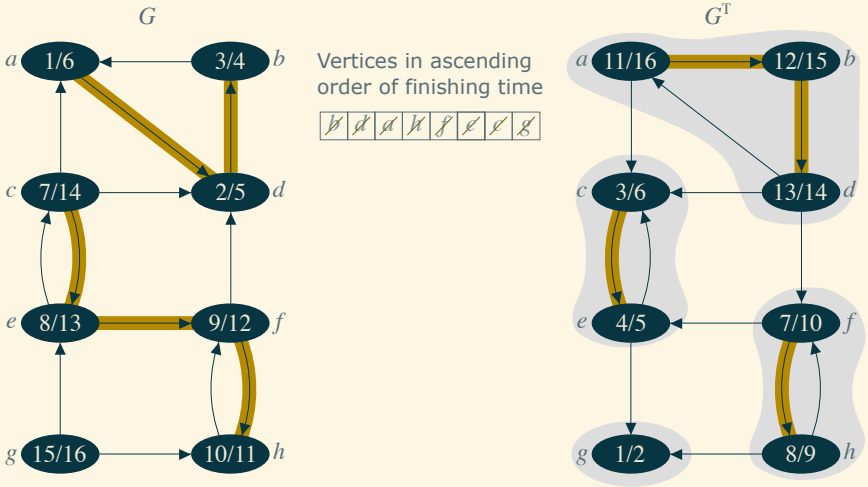


19 / 21

Determining Strongly Connected Components

Example 1

3. Call $\mathrm{DFS}(G^{\mathrm{T}})$, but, in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1).

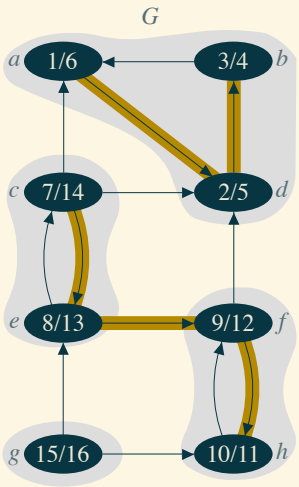


19 / 21

Determining Strongly Connected Components

Example 1

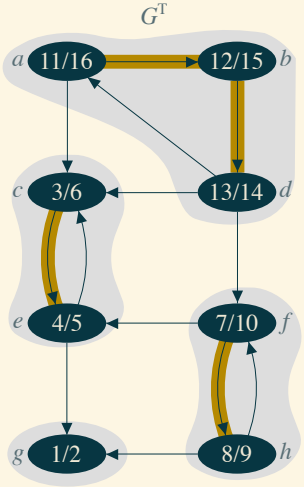
4. Output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component.



Vertices in ascending order of finishing time

~~f~~	~~d~~	~~a~~	~~b~~	~~c~~	~~e~~	~~h~~	~~g~~

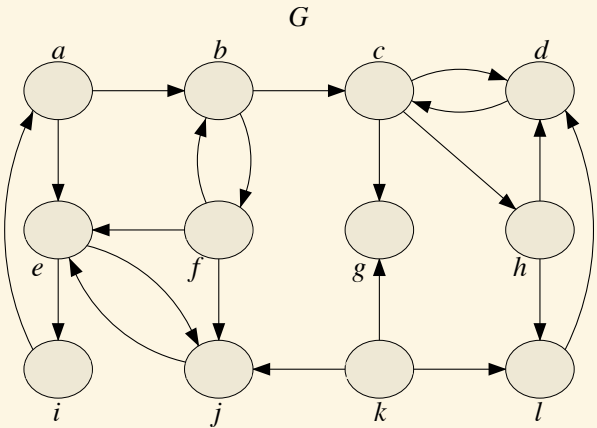
Strongly connected components:
$\{a, b, d\}, \{c, e\}, \{f, h\}, \{g\}$



19 / 21

Determining Strongly Connected Components

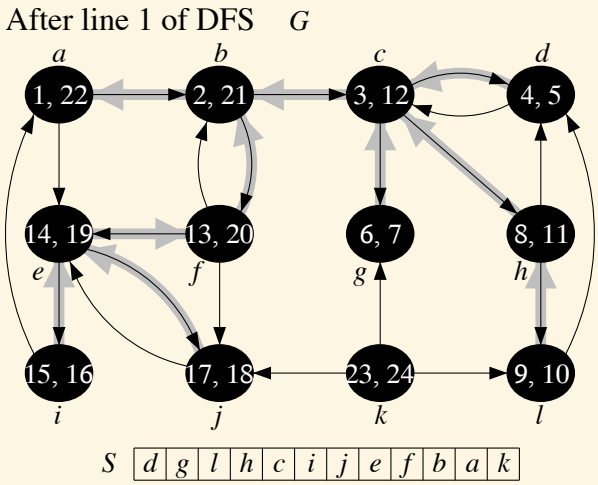
Example 2



20 / 21

Determining Strongly Connected Components

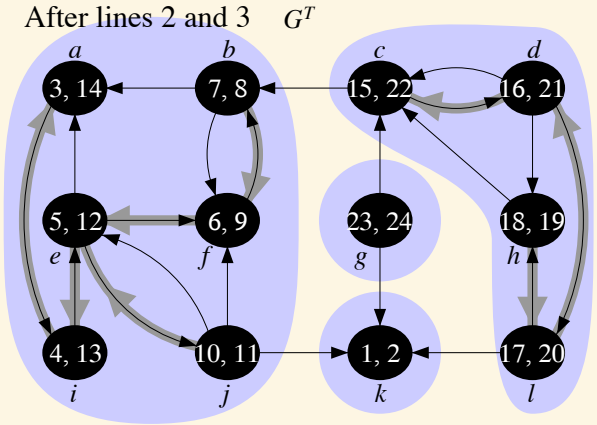
Example 2



20 / 21

Determining Strongly Connected Components

Example 2



After line 4: The strongly connected components are $\{a, b, e, f, i, j\}, \{c, d, h, l\}, \{g\}$ and $\{k\}$.

20 / 21

Outlook and Conclusion

This Lesson: Depth-First Search. Next Lesson: Heaps.

In the previous lesson, we learnt that breadth-first search is a graph traversal algorithm that calculates the distances from a source vertex to a target vertex.

In this lesson, we have seen that depth-first search is another graph traversal algorithm, which is often used to find components in undirected graphs and strongly connected components in directed graphs.

In all the graphs we have encountered so far, edges have only informed us that two vertices were linked. However, edges in many real-world graph have additional weights (e.g. distances or costs). Next time, we will introduce weighted graphs and minimum spanning trees, which are connected subgraphs containing all vertices that minimise the sum of the weights of the included edges.

21 / 21