Connected Components

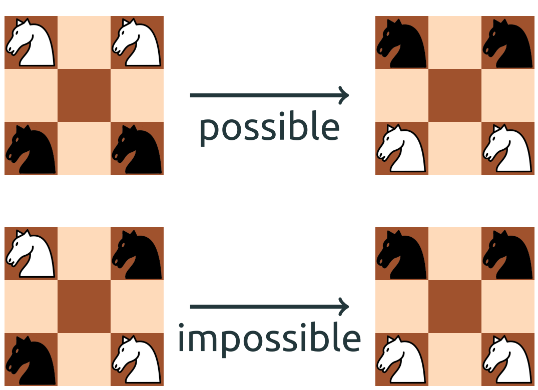
Alexander S. Kulikov

Steklov Mathematical Institute at St. Petersburg, Russian Academy of Sciences
and
University of California, San Diego

The Heaviest Stone

There are n stones of different weights. An expert knows the weights and wants to convince the court that a particular stone is the heaviest one. For this, he repeatedly uses a pan balance to compare the weights of some two stones. What is the minimum number of comparisons required?

Guarini Puzzle, Revisited



can we check this automatically instead of manually?

Hm...

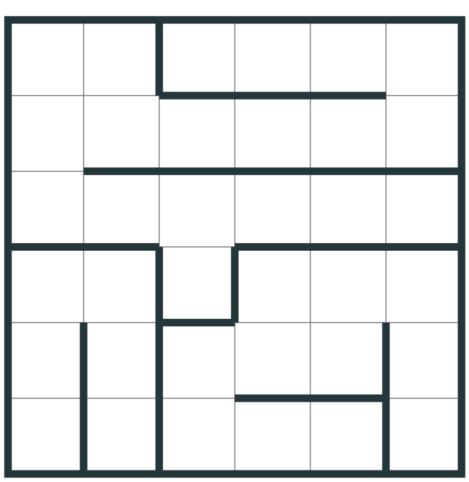
- What do these two unrelated puzzles have in common?

Hm...

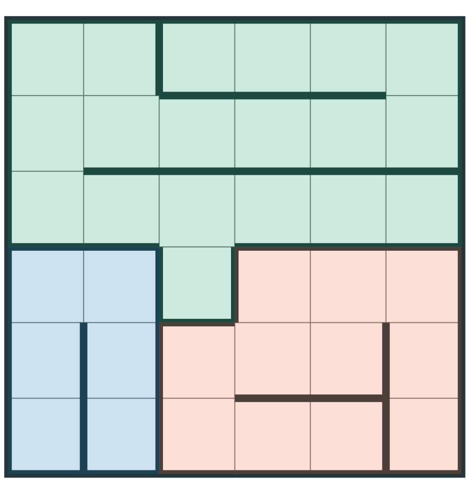
- What do these two unrelated puzzles have in common?
- They both can be solved by analyzing connected components of an underlying graph!

Outline

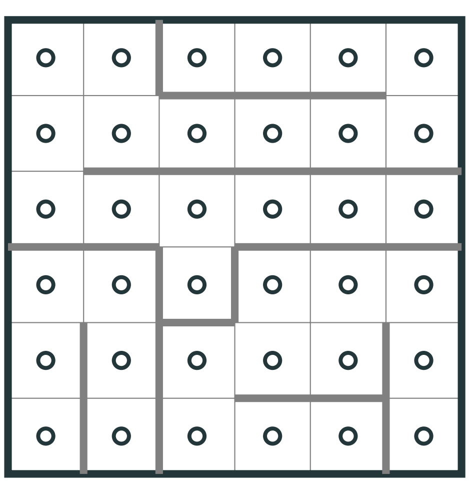
Connected Components in a Maze



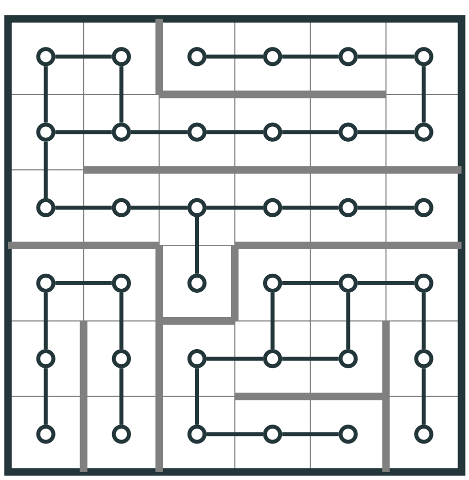
Connected Components in a Maze



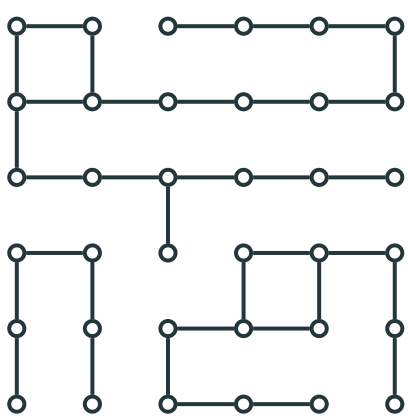
Connected Components in a Maze



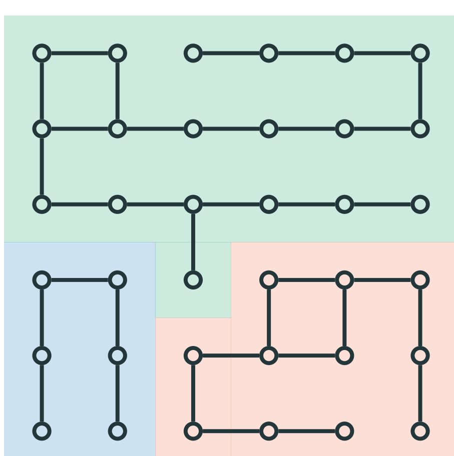
Connected Components in a Maze



Connected Components in a Maze



Connected Components in a Maze



Connected Graphs

- Consider an *undirected* graph

Connected Graphs

- Consider an *undirected* graph
- Two nodes are connected, if there is a path between them

Connected Graphs

- Consider an *undirected* graph
- Two nodes are connected, if there is a path between them
- It is transitive: if *u* and *v* are connected and *v* and *w* are connected, then *u* and *w* are connected, too

Connected Graphs

- Consider an *undirected* graph
- Two nodes are connected, if there is a path between them
- It is transitive: if *u* and *v* are connected and *v* and *w* are connected, then *u* and *w* are connected, too
- A graph is connected, if any two of its nodes are connected. In other words, there is a path between any two of its nodes

Connected Components

The nodes of any undirected graph can be partitioned into subsets called connected components:

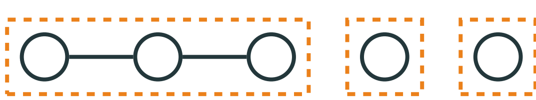
- Any node belongs to exactly one connected component
- Any two nodes from the same connected component are connected
- Any two nodes from different connected components are not connected

Examples

Examples

Examples

Examples

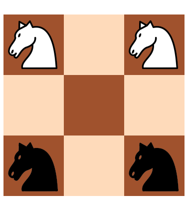


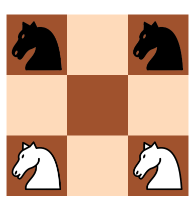
Examples

Examples

Outline

Revisiting the Guarini Puzzle





Given two configurations, check whether one is reachable from the other one

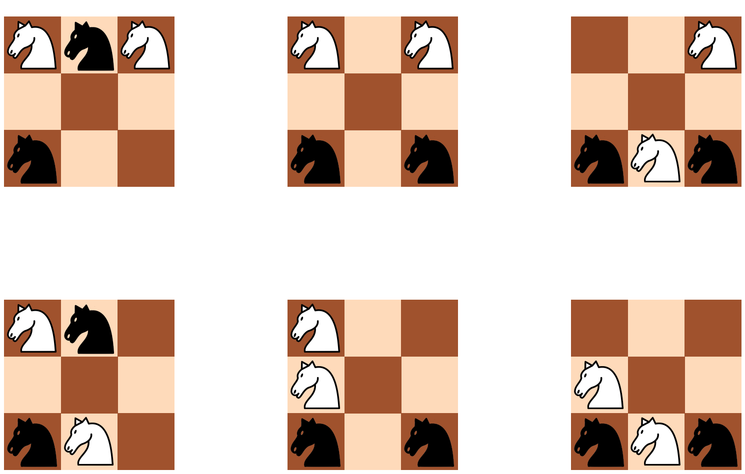
Graph of Configurations

- Consider a graph where the set of nodes is the set of all configurations, i.e., all possible 3×3 boards with two white knights and two black knights

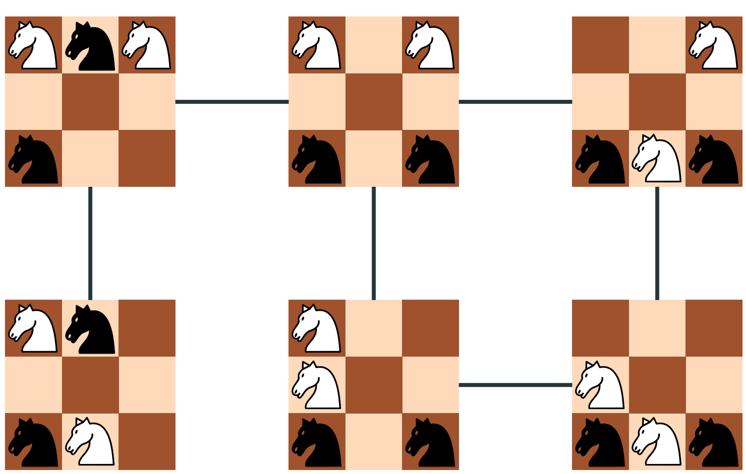
Graph of Configurations

- Consider a graph where the set of nodes is the set of all configurations, i.e., all possible 3×3 boards with two white knights and two black knights
- Join two nodes by an edge if their configurations are within a single move from each other

Graph of Configurations



Graph of Configurations



Solution

Then, one configuration is reachable from the other one, if and only if they belong to the same connected component!

Outline

Connected Components

Guarini Puzzle: Program

Lower Bound

The Heaviest Stone

Directed Acyclic Graphs

Strongly Connected Components

Lower Bound

Theorem

An undirected graph $G(V, E)$ has at least $|V| - |E|$ connected components.

Lower Bound

Theorem

An undirected graph $G(V, E)$ has at least $|V| - |E|$ connected components.

- If a graph is connected, then $|E| \geq |V| - 1$ (indeed, if $|E| \leq |V| - 2$, then, by the theorem, the graph has at least 2 connected components)

Lower Bound

Theorem

An undirected graph $G(V, E)$ has at least $|V| - |E|$ connected components.

- If a graph is connected, then $|E| \geq |V| - 1$ (indeed, if $|E| \leq |V| - 2$, then, by the theorem, the graph has at least 2 connected components)
- If $|E| = 0$, then every node forms a connected component

Lower Bound

Theorem

An undirected graph $G(V, E)$ has at least $|V| - |E|$ connected components.

- If a graph is connected, then $|E| \geq |V| - 1$ (indeed, if $|E| \leq |V| - 2$, then, by the theorem, the graph has at least 2 connected components)
- If $|E| = 0$, then every node forms a connected component
- The theorem is useless for graphs with $|E| \geq |V|$

Proof

- Start with an empty graph (containing no edges)

Proof

- Start with an empty graph (containing no edges)
- Initially, the number of connected components is $|V|$, it is indeed at least $|V| - |E| = |V|$

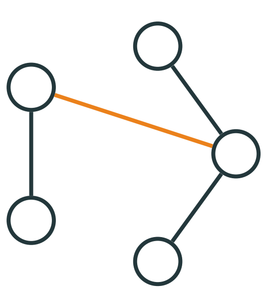
Proof

- Start with an empty graph (containing no edges)
- Initially, the number of connected components is $|V|$, it is indeed at least $|V| - |E| = |V|$
- Each time when we add a new edge, $|V| - |E|$ decreases by 1

Proof

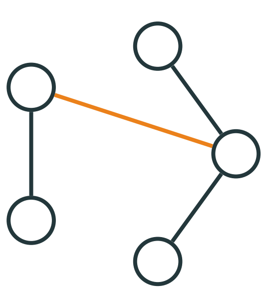
- Start with an empty graph (containing no edges)
- Initially, the number of connected components is $|V|$, it is indeed at least $|V| - |E| = |V|$
- Each time when we add a new edge, $|V| - |E|$ decreases by 1
- At the same time, the number of connected components either decreases by 1 or stays the same

Illustration

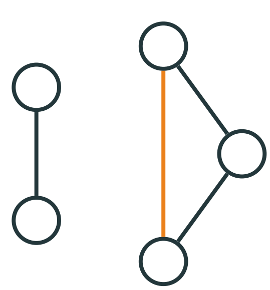


decreases

Illustration



decreases



stays the same

Outline

The Heaviest Stone

There are *n* stones of different weights. An expert knows the weights and wants to convince the court that a particular stone is the heaviest one. For this, he repeatedly uses a pan balance to compare the weights of some two stones. What is the minimum number of comparisons required?

Upper Bound

- $n - 1$ comparisons are definitely enough:

Upper Bound

- $n - 1$ comparisons are definitely enough:
  - the expert might compare the heaviest stone with all other $n - 1$ stones

Upper Bound

- $n - 1$ comparisons are definitely enough:
  - the expert might compare the heaviest stone with all other $n - 1$ stones
  - the expert can also order the stones by their weight ($w_1 < w_2 < \cdots < w_n$) and then perform comparisons $w_1 < w_2$, $w_2 < w_3$, $\ldots$, $w_{n-1} < w_n$; this will reveal the full order on stones

Upper Bound

- $n - 1$ comparisons are definitely enough:
  - the expert might compare the heaviest stone with all other $n - 1$ stones
  - the expert can also order the stones by their weight ($w_1 < w_2 < \cdots < w_n$) and then perform comparisons $w_1 < w_2$, $w_2 < w_3$, $\ldots$, $w_{n-1} < w_n$; this will reveal the full order on stones
- but is it optimal?

Upper Bound

- $n-1$ comparisons are definitely enough:
  - the expert might compare the heaviest stone with all other $n-1$ stones
  - the expert can also order the stones by their weight ($w_1 < w_2 < \cdots < w_n$) and then perform comparisons $w_1 < w_2$, $w_2 < w_3$, $\ldots$, $w_{n-1} < w_n$; this will reveal the full order on stones
- but is it optimal?
- yes!

Proof

- Consider the following graph: nodes are stones, two stones are joined by an edge if they were compared by the expert

Proof

- Consider the following graph: nodes are stones, two stones are joined by an edge if they were compared by the expert
- Note that we are not even interested in the results of comparisons performed by the expert

Proof

- Consider the following graph: nodes are stones, two stones are joined by an edge if they were compared by the expert
- Note that we are not even interested in the results of comparisons performed by the expert
- If there were less than $n - 1$ comparisons, then the graph contains at least two connected components

Proof

- Consider the following graph: nodes are stones, two stones are joined by an edge if they were compared by the expert
- Note that we are not even interested in the results of comparisons performed by the expert
- If there were less than $n - 1$ comparisons, then the graph contains at least two connected components
- But this means that the court is still not sure about the heaviest stone!

Outline

DAGs

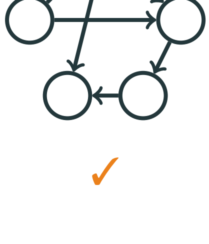
Definition

A directed acyclic graph, or simply a DAG, is a directed graph without cycles.

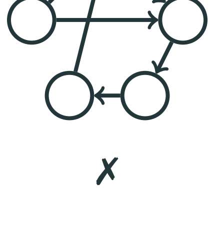
DAGs

Definition

A directed acyclic graph, or simply a DAG, is a directed graph without cycles.

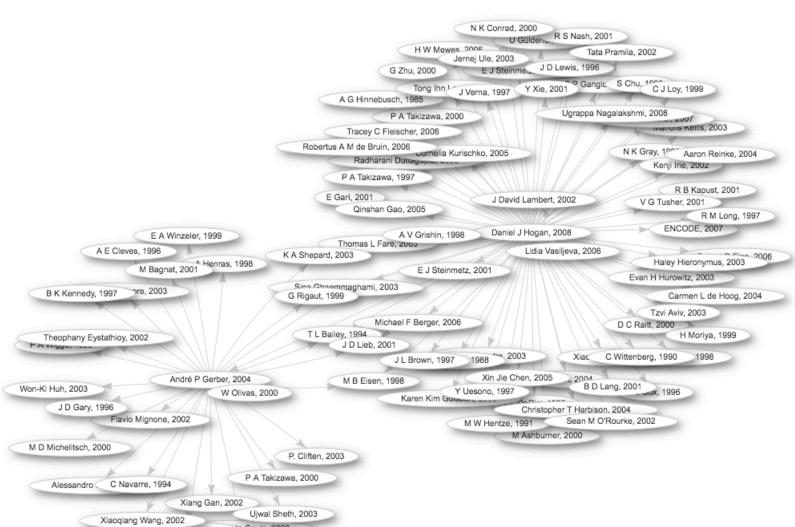


✓

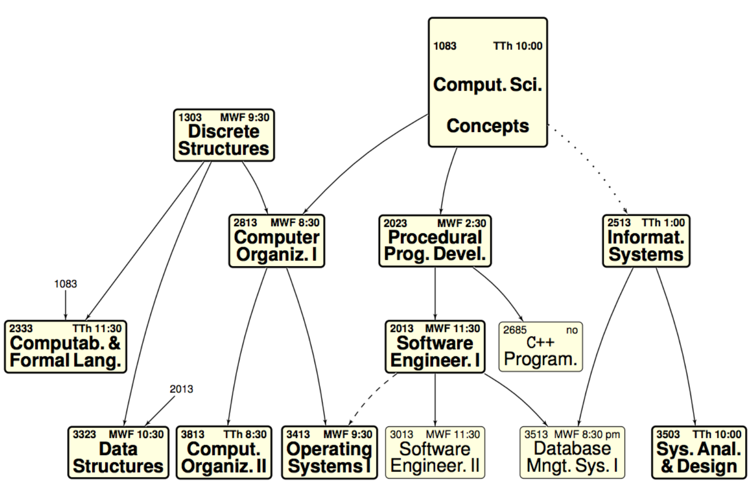


✗

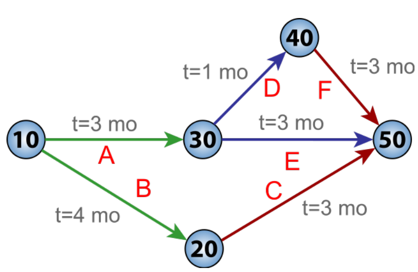
Citation Graph



Prerequisite Graph



Dependency Graph



Dependency Graph

- Consider the following (directed) dependency graph: nodes are jobs, there is a directed edge from *A* to *B* if the job *A* must be processed before *B*

Dependency Graph

- Consider the following (directed) dependency graph: nodes are jobs, there is a directed edge from A to B if the job A must be processed before B
- We want to process jobs one by one

Dependency Graph

- Consider the following (directed) dependency graph: nodes are jobs, there is a directed edge from A to B if the job A must be processed before B
- We want to process jobs one by one
- How to find an order of jobs satisfying all constraints?

Dependency Graph

- Consider the following (directed) dependency graph: nodes are jobs, there is a directed edge from A to B if the job A must be processed before B
- We want to process jobs one by one
- How to find an order of jobs satisfying all constraints?
- If there is a cycle in the graph, then there is no such order

Dependency Graph

- Consider the following (directed) dependency graph: nodes are jobs, there is a directed edge from A to B if the job A must be processed before B
- We want to process jobs one by one
- How to find an order of jobs satisfying all constraints?
- If there is a cycle in the graph, then there is no such order
- It turns out that this is the only obstacle: if the graph is acyclic, then there is an ordering of its vertices satisfying all the constraints!

Topological Ordering

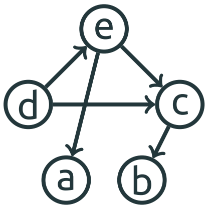
Definition

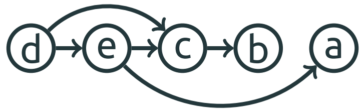
A topological ordering of a directed graph is an ordering of its vertices such that, for each edge (u, v), u comes before v.

Topological Ordering

Definition

A topological ordering of a directed graph is an ordering of its vertices such that, for each edge (u, v), u comes before v.





Every DAG Can Be Ordered

Theorem

Every DAG has a topological ordering.

Every DAG Can Be Ordered

Theorem

Every DAG has a topological ordering.

Proof

- We'll show that every DAG has a sink — a node with no outgoing edges

Every DAG Can Be Ordered

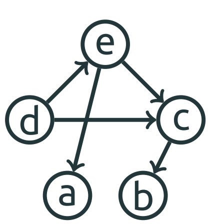
Theorem

Every DAG has a topological ordering.

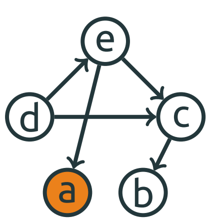
Proof

- We'll show that every DAG has a sink — a node with no outgoing edges
- Take a sink, put it to the end of the ordering, remove it from the graph (this keeps the graph acyclic), and repeat

Example



Example



Example

Example

Example

Example

d

d e c b a

Every DAG Has a Sink

- Assume that a DAG does not have a sink: for every node, there is at least one outgoing edge

Every DAG Has a Sink

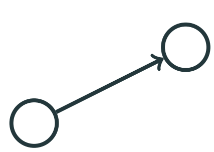
- Assume that a DAG does not have a sink: for every node, there is at least one outgoing edge
- Start a walk from any vertex:

Every DAG Has a Sink

- Assume that a DAG does not have a sink: for every node, there is at least one outgoing edge
- Start a walk from any vertex:

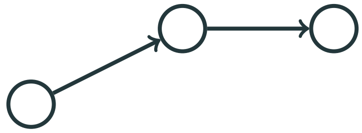
Every DAG Has a Sink

- Assume that a DAG does not have a sink: for every node, there is at least one outgoing edge
- Start a walk from any vertex:



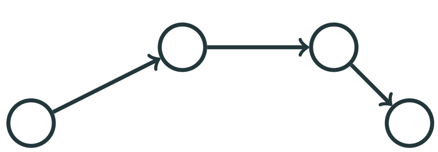
Every DAG Has a Sink

- Assume that a DAG does not have a sink: for every node, there is at least one outgoing edge
- Start a walk from any vertex:



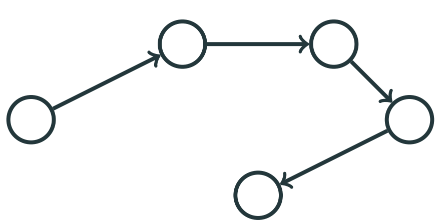
Every DAG Has a Sink

- Assume that a DAG does not have a sink: for every node, there is at least one outgoing edge
- Start a walk from any vertex:



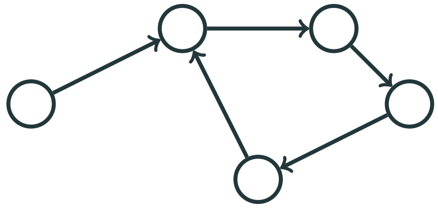
Every DAG Has a Sink

- Assume that a DAG does not have a sink: for every node, there is at least one outgoing edge
- Start a walk from any vertex:



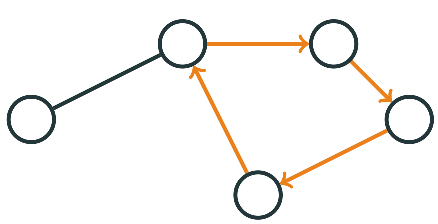
Every DAG Has a Sink

- Assume that a DAG does not have a sink: for every node, there is at least one outgoing edge
- Start a walk from any vertex:



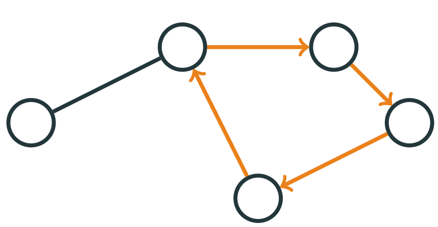
Every DAG Has a Sink

- Assume that a DAG does not have a sink: for every node, there is at least one outgoing edge
- Start a walk from any vertex:



Every DAG Has a Sink

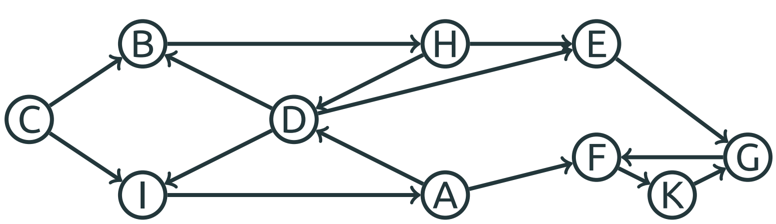
- Assume that a DAG does not have a sink: for every node, there is at least one outgoing edge
- Start a walk from any vertex:



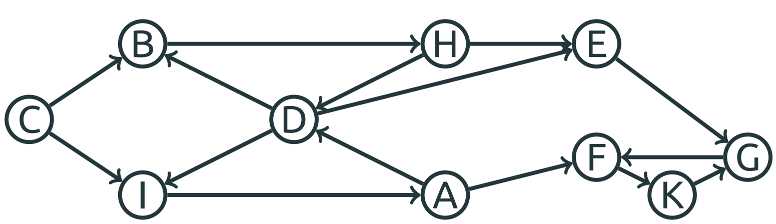
- A contradiction!

Outline

Is This Graph Connected?

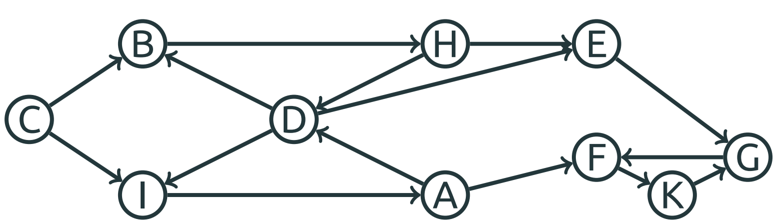


Is This Graph Connected?



- On one hand, this graph is connected: it cannot be "pulled apart"

Is This Graph Connected?



- On one hand, this graph is connected: it cannot be "pulled apart"
- On the other hand, it is not connected: e.g., there is no path from A to C

Strongly Connected Components

- In a directed graph, nodes u, v are connected, if there is a path from u to v *and* a path from v to u

Strongly Connected Components

- In a directed graph, nodes *u*, *v* are connected, if there is a path from *u* to *v* *and* a path from *v* to *u*
- Nodes of any directed graph can be partitioned into subsets called strongly connected components (SCCs):

Strongly Connected Components

- In a directed graph, nodes u, v are connected, if there is a path from u to v *and* a path from v to u
- Nodes of any directed graph can be partitioned into subsets called strongly connected components (SCCs):
  - every node belongs to exactly one SCC

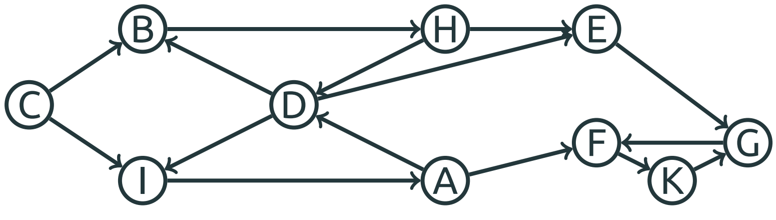
Strongly Connected Components

- In a directed graph, nodes u, v are connected, if there is a path from u to v *and* a path from v to u
- Nodes of any directed graph can be partitioned into subsets called strongly connected components (SCCs):
    - every node belongs to exactly one SCC
    - nodes from the same SCC are connected

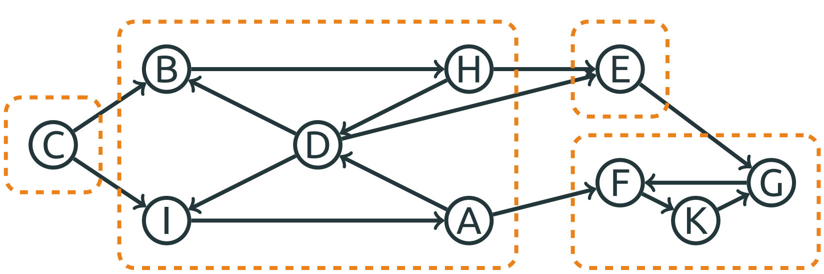
Strongly Connected Components

- In a directed graph, nodes *u*, *v* are connected, if there is a path from *u* to *v* *and* a path from *v* to *u*
- Nodes of any directed graph can be partitioned into subsets called strongly connected components (SCCs):
    - every node belongs to exactly one SCC
    - nodes from the same SCC are connected
    - nodes from different SCCs are not connected

Example



Example



Example