Euler Tour Trees

Outline for Today

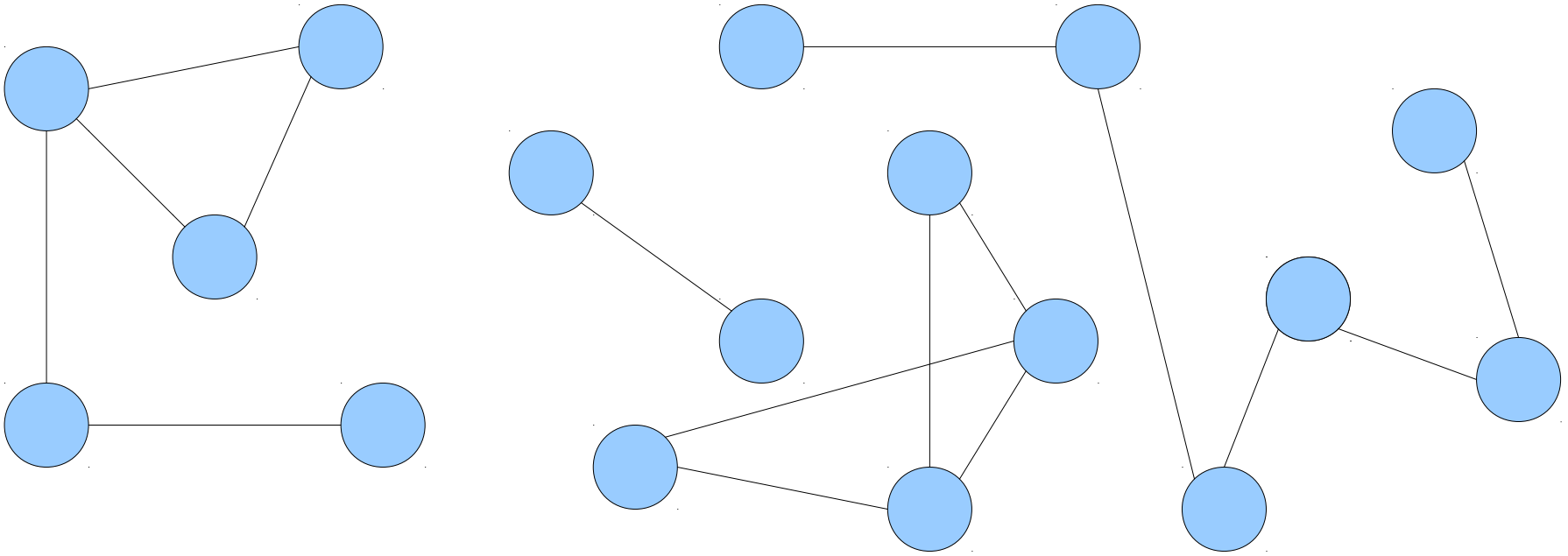
- **Dynamic Connectivity on Trees**
  - Maintaining connectivity in a changing environment.
- **Euler Tour Trees**
  - A data structure for dynamic connectivity.
- **The Bottleneck Path Problem**
  - Putting everything together.

The Dynamic Connectivity Problem

The Connectivity Problem

- The **graph connectivity problem** is the following:

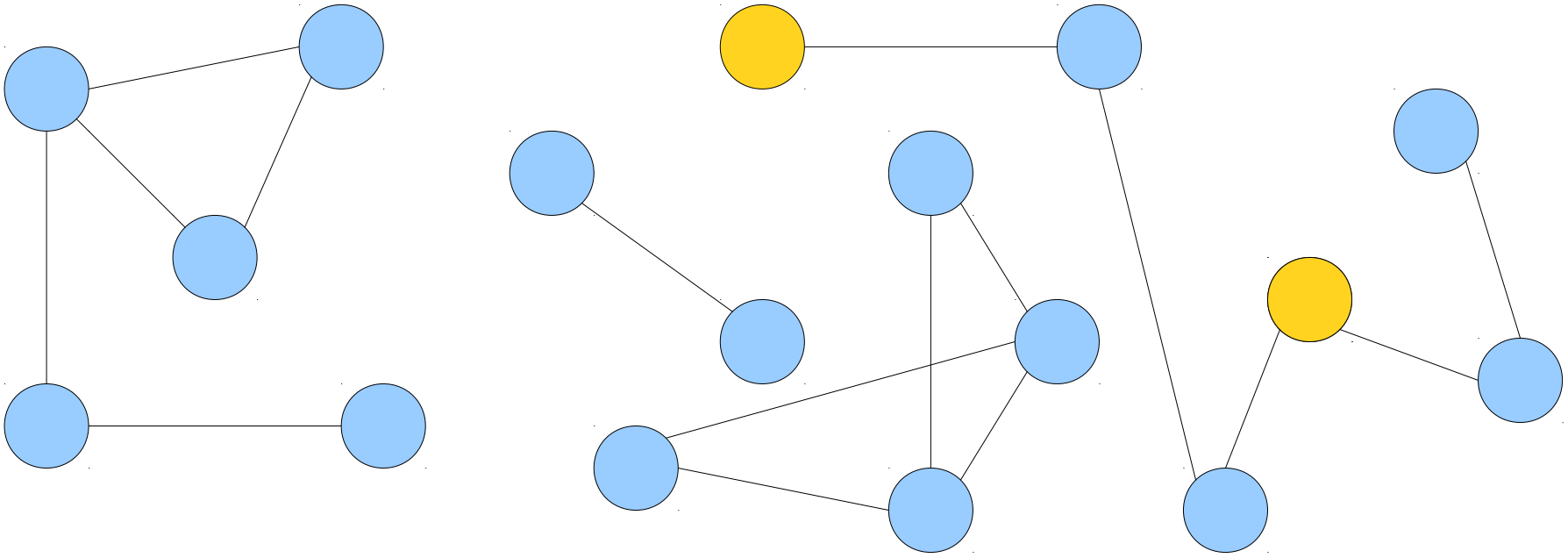
  Given an undirected graph *G*, preprocess the graph so that queries of the form "are nodes *u* and *v* connected?"



The Connectivity Problem

- The **graph connectivity problem** is the following:

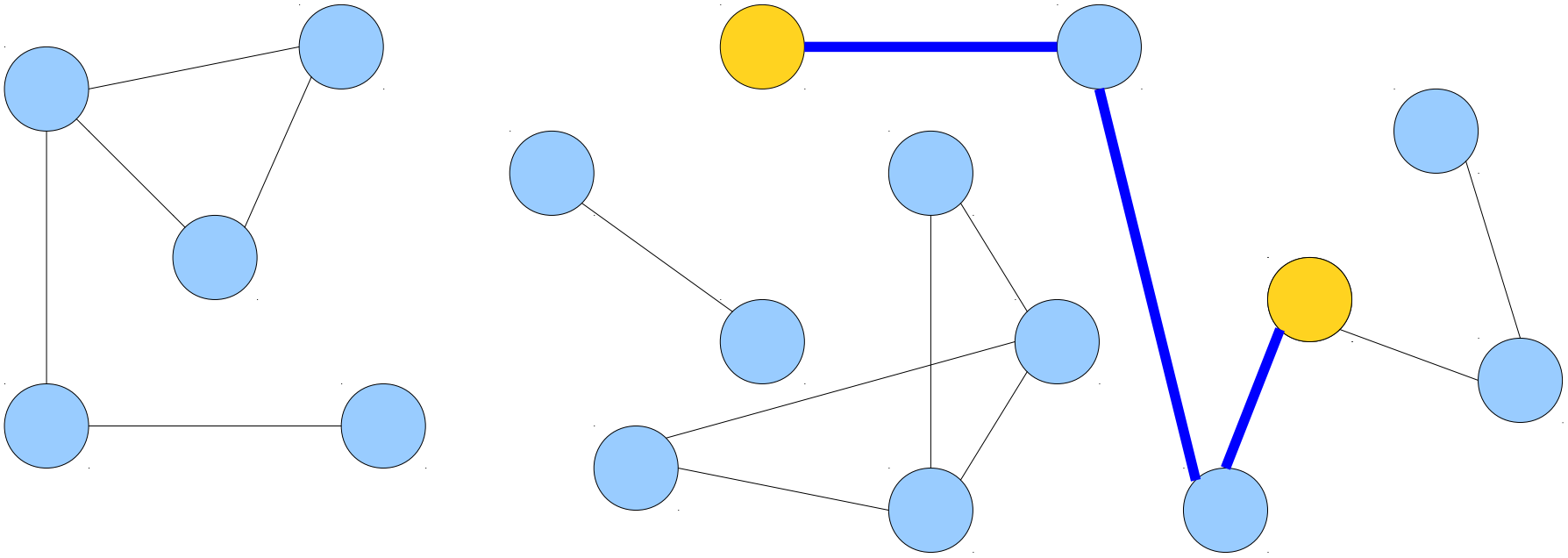
  Given an undirected graph *G*, preprocess the graph so that queries of the form “are nodes *u* and *v* connected?”



The Connectivity Problem

- The **graph connectivity problem** is the following:

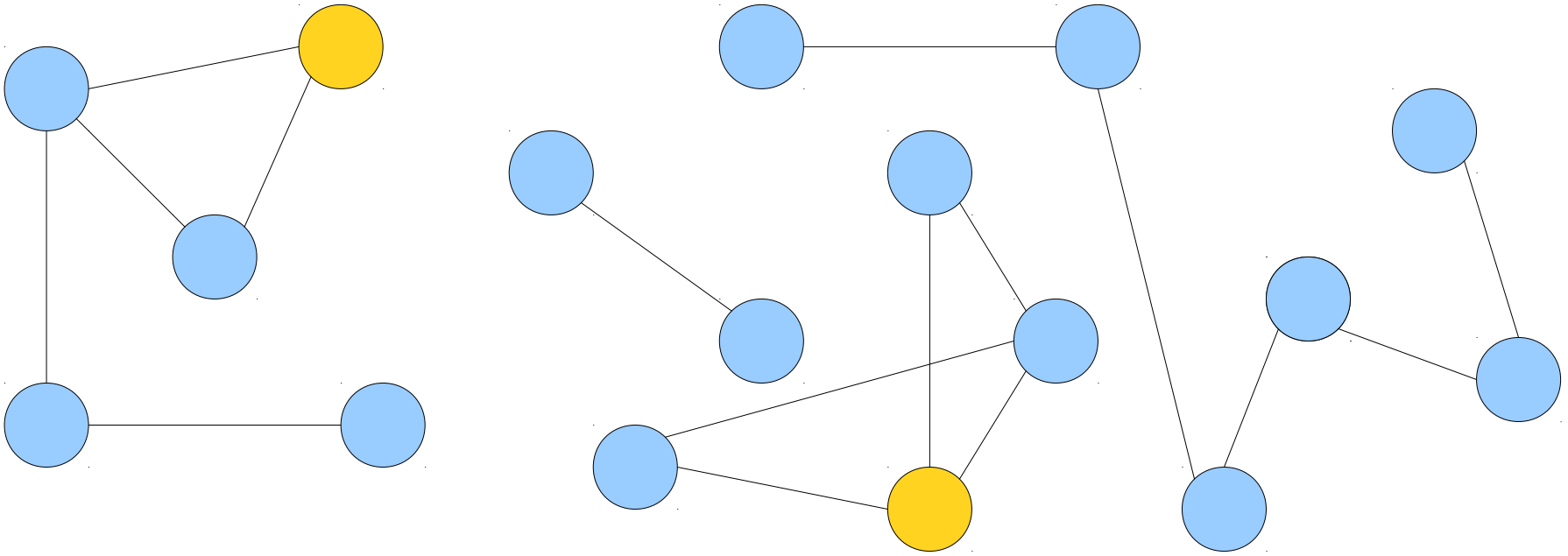
  Given an undirected graph *G*, preprocess the graph so that queries of the form "are nodes *u* and *v* connected?"



The Connectivity Problem

- The **graph connectivity problem** is the following:

  Given an undirected graph *G*, preprocess the graph so that queries of the form “are nodes *u* and *v* connected?”

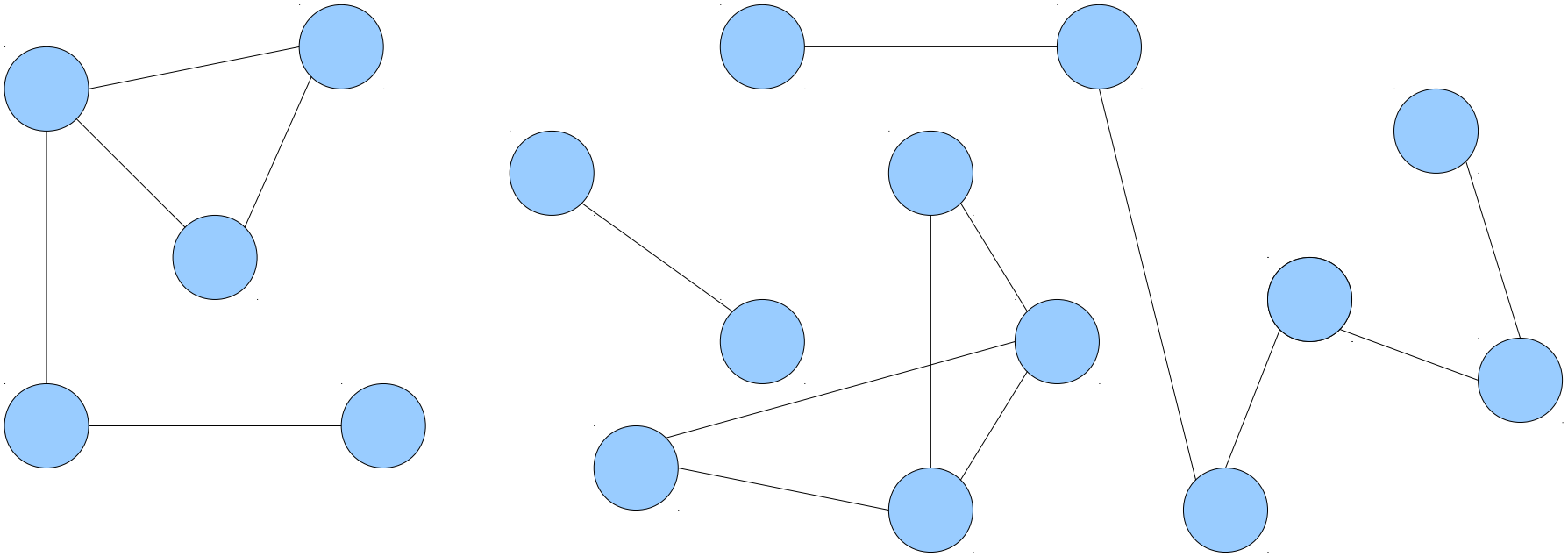


The Connectivity Problem

- The **graph connectivity problem** is the following:

  Given an undirected graph G, preprocess the graph so that queries of the form "are nodes u and v connected?"

- Using $\Theta(m + n)$ preprocessing, can preprocess the graph to answer queries in time O(1).

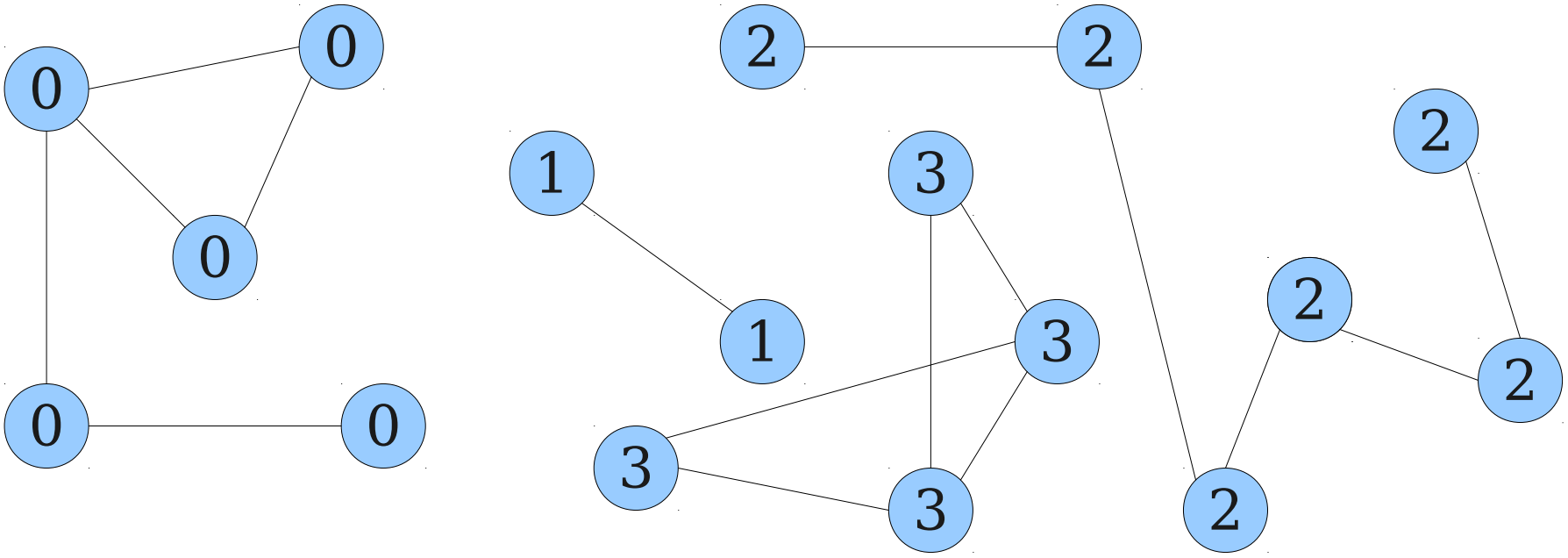


The Connectivity Problem

- The **graph connectivity problem** is the following:

  Given an undirected graph G, preprocess the graph so that queries of the form "are nodes u and v connected?"

- Using $\Theta(m + n)$ preprocessing, can preprocess the graph to answer queries in time O(1).

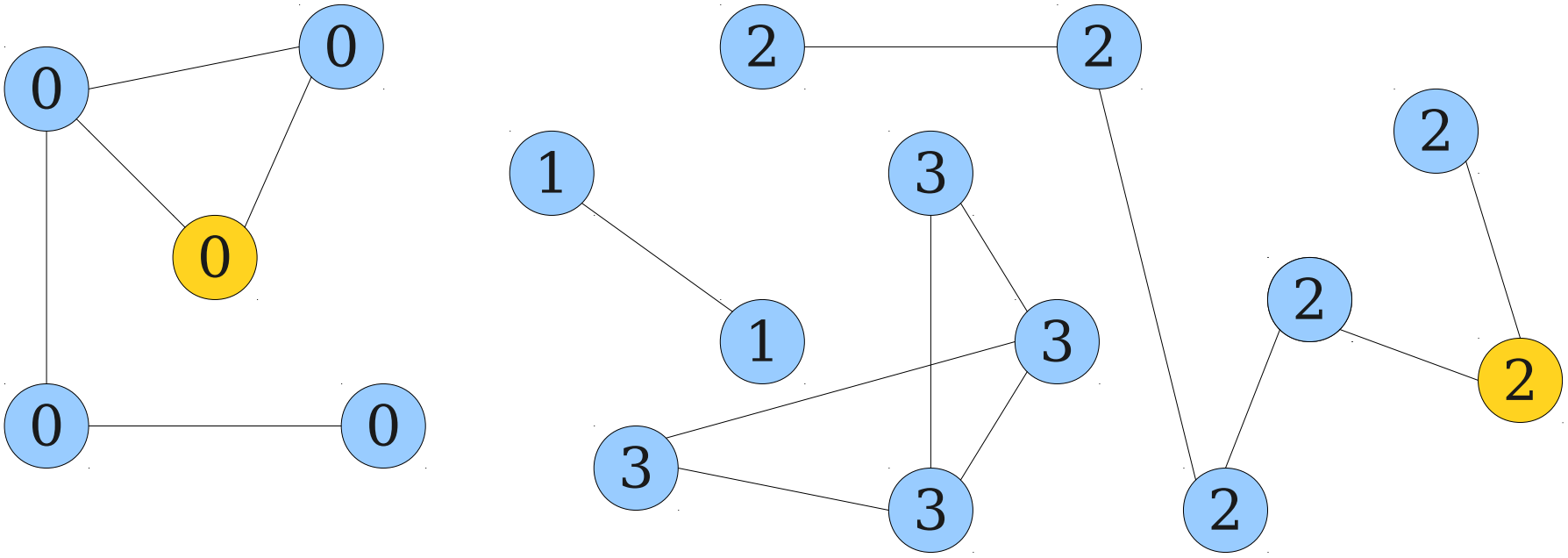


The Connectivity Problem

- The **graph connectivity problem** is the following:

  Given an undirected graph G, preprocess the graph so that queries of the form "are nodes u and v connected?"

- Using $\Theta(m + n)$ preprocessing, can preprocess the graph to answer queries in time O(1).

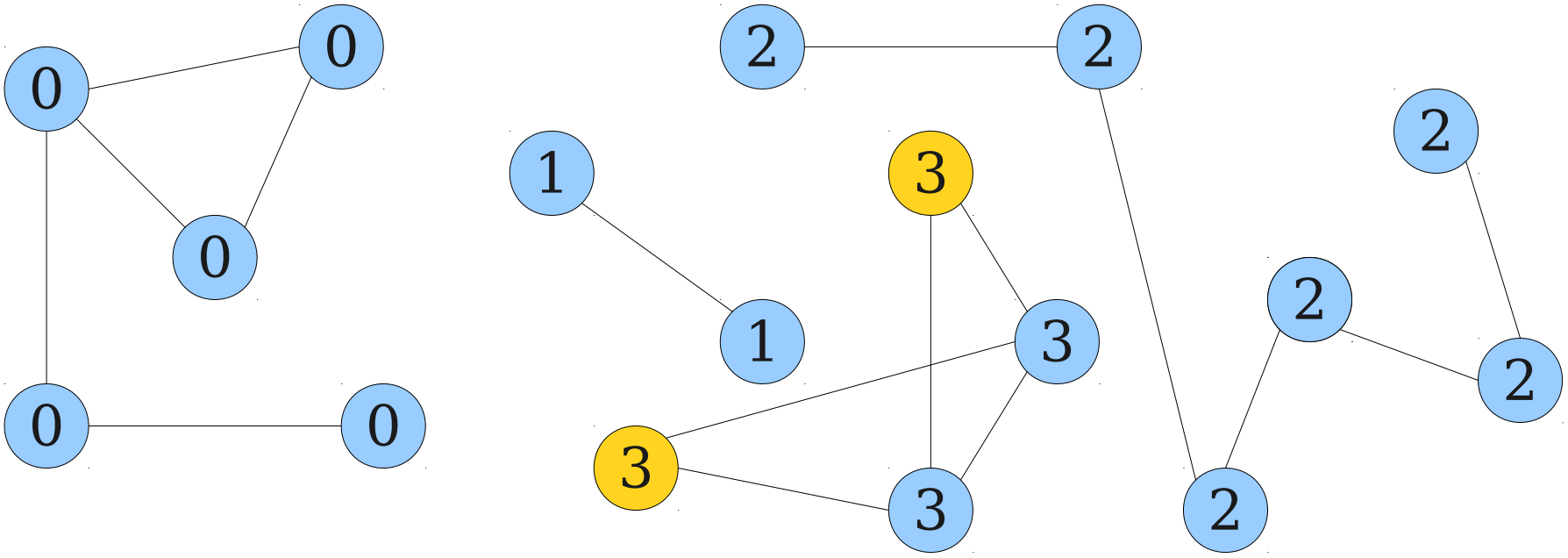


The Connectivity Problem

- The **graph connectivity problem** is the following:

  Given an undirected graph G, preprocess the graph so that queries of the form "are nodes u and v connected?"

- Using $\Theta(m + n)$ preprocessing, can preprocess the graph to answer queries in time O(1).

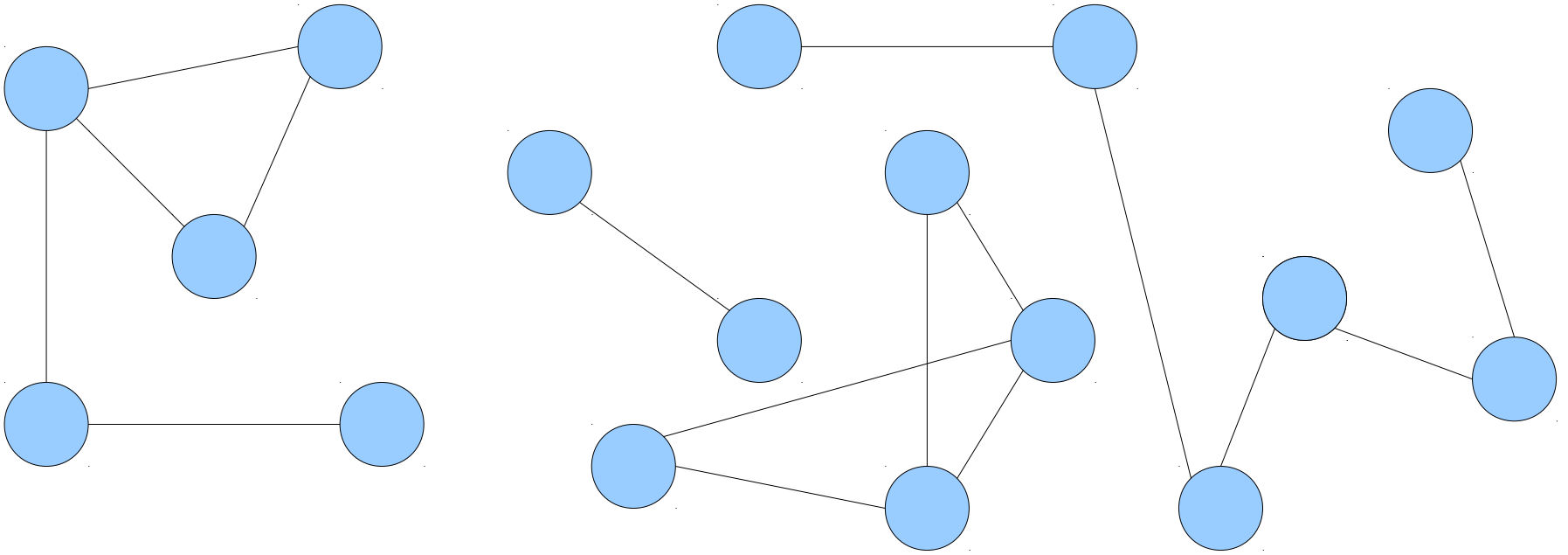


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

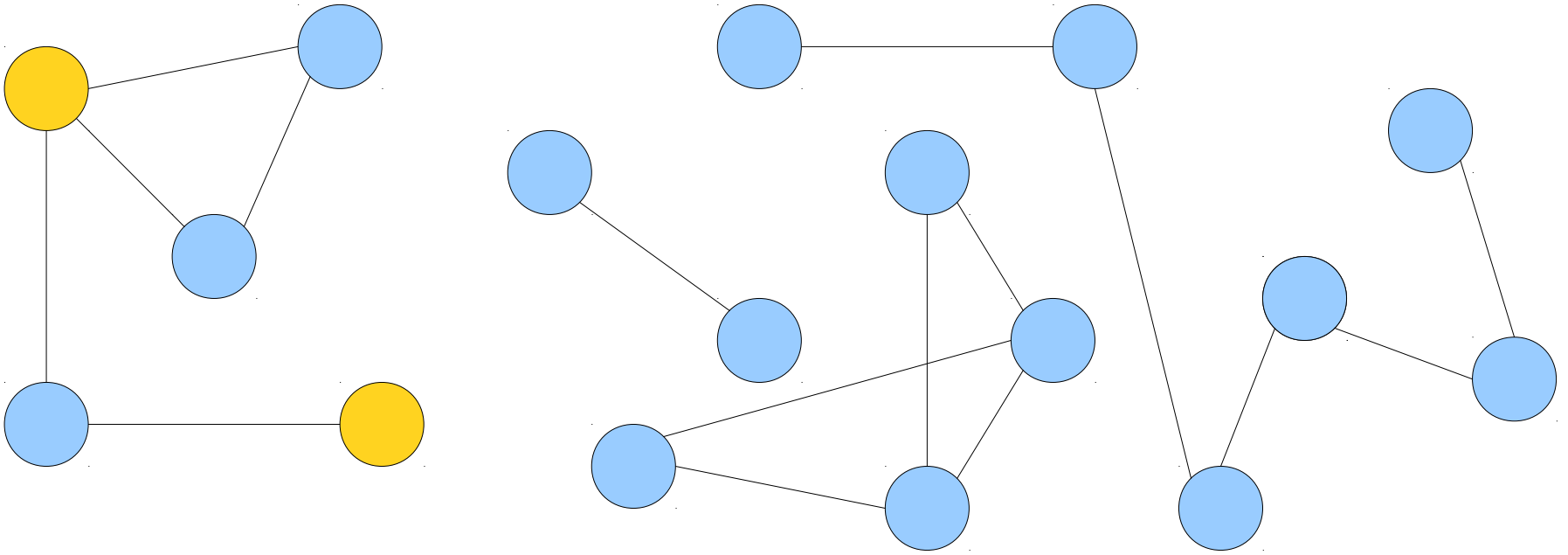


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

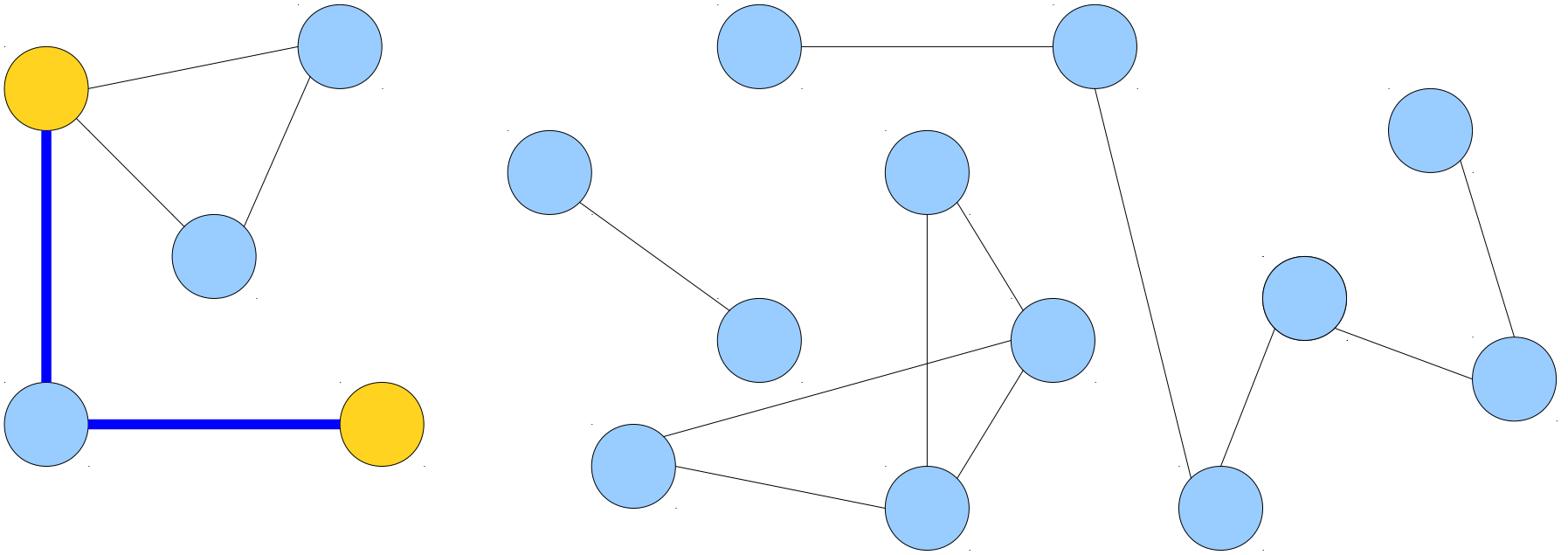


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

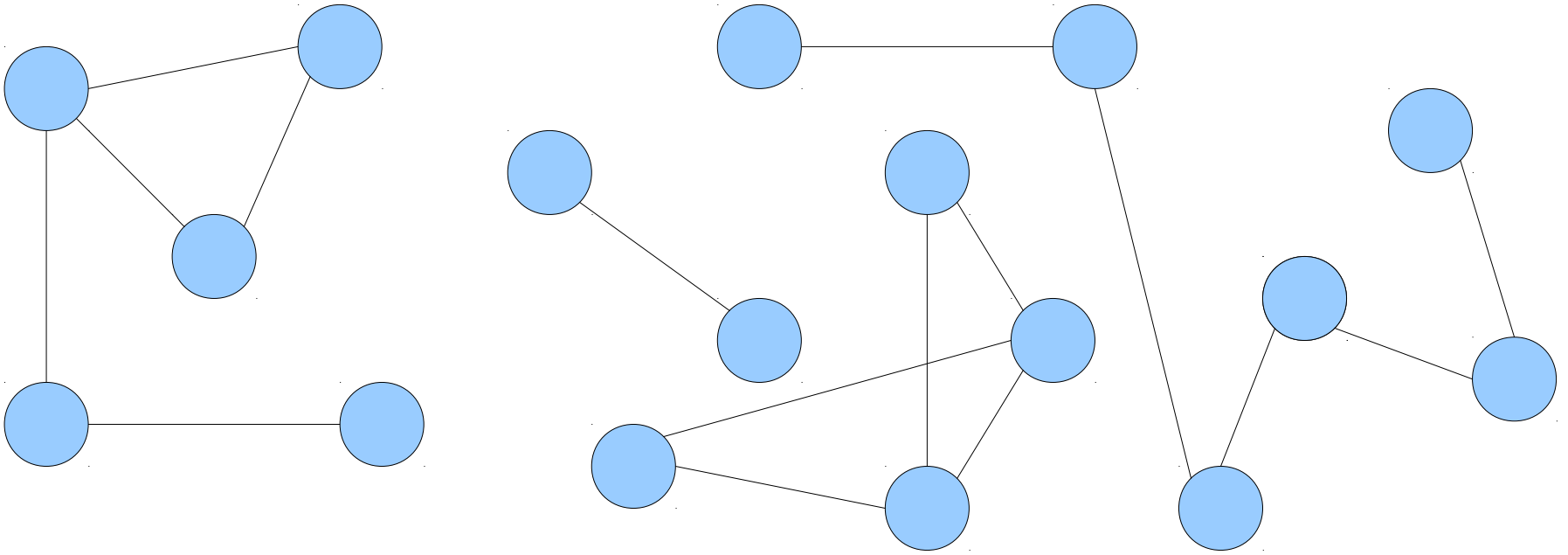


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

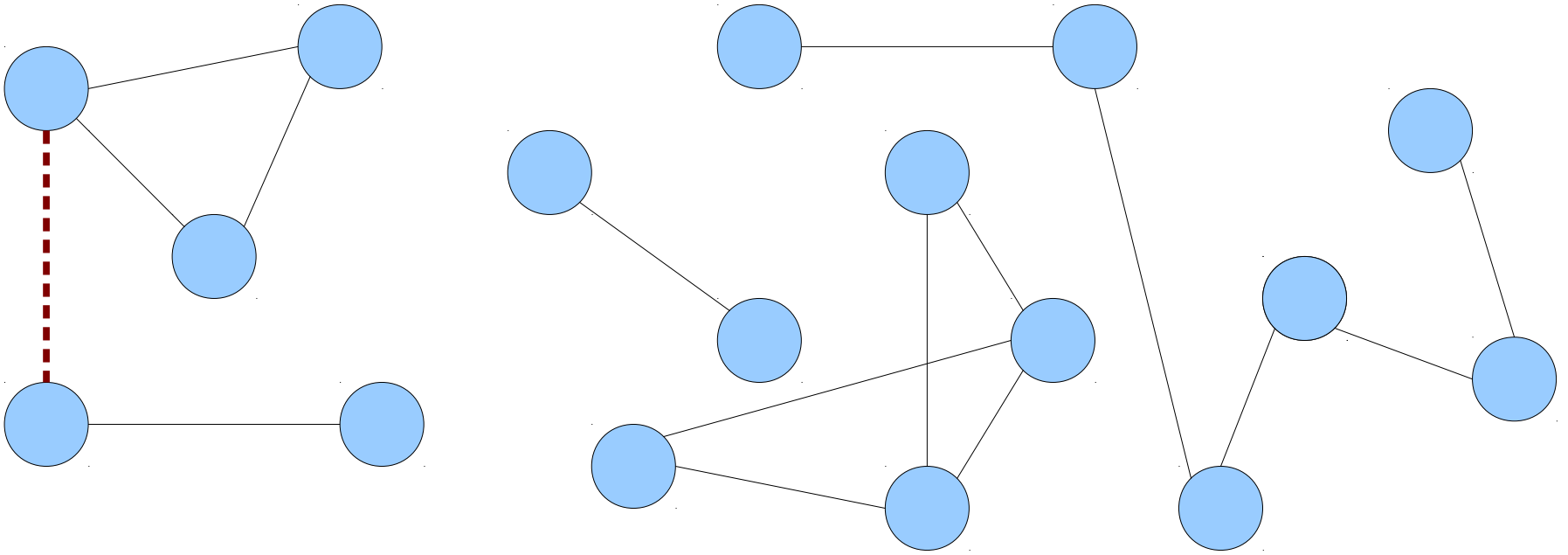


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

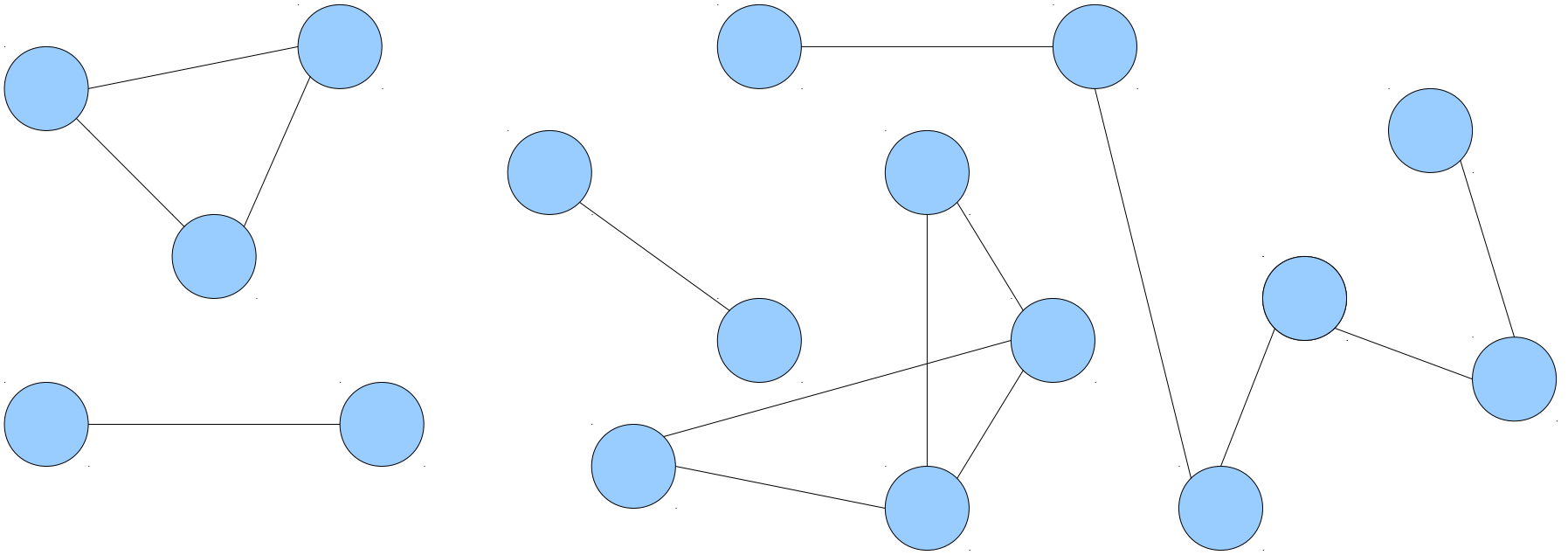


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

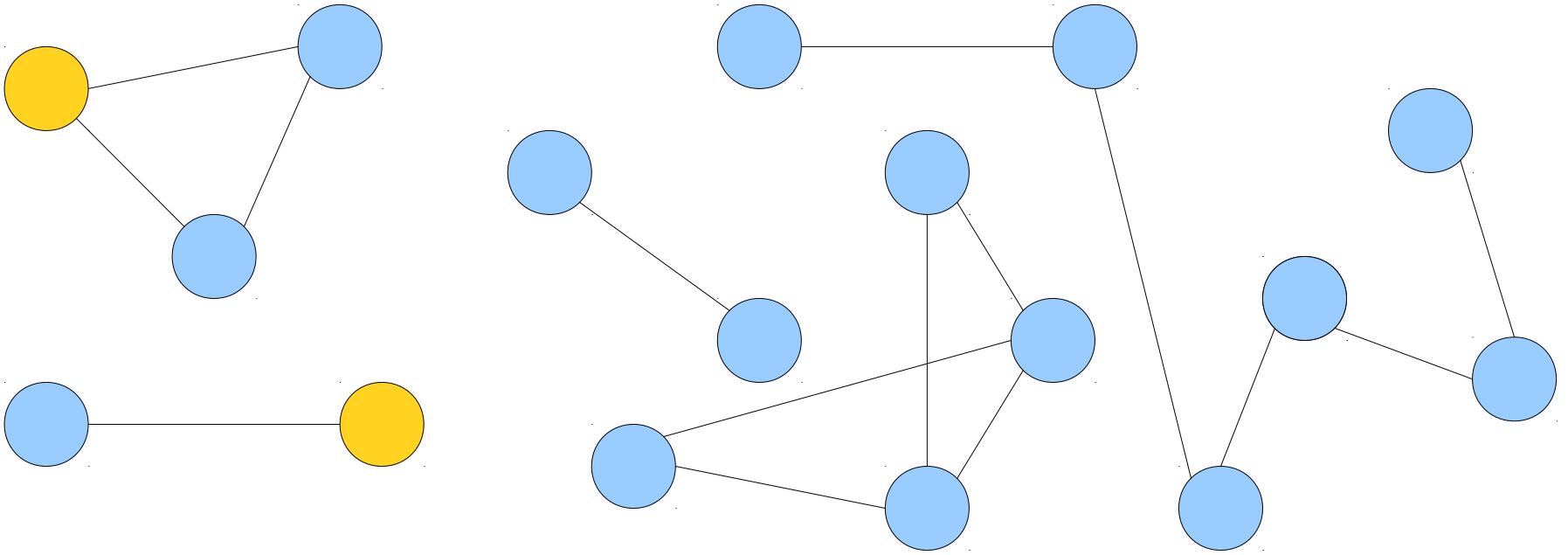


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

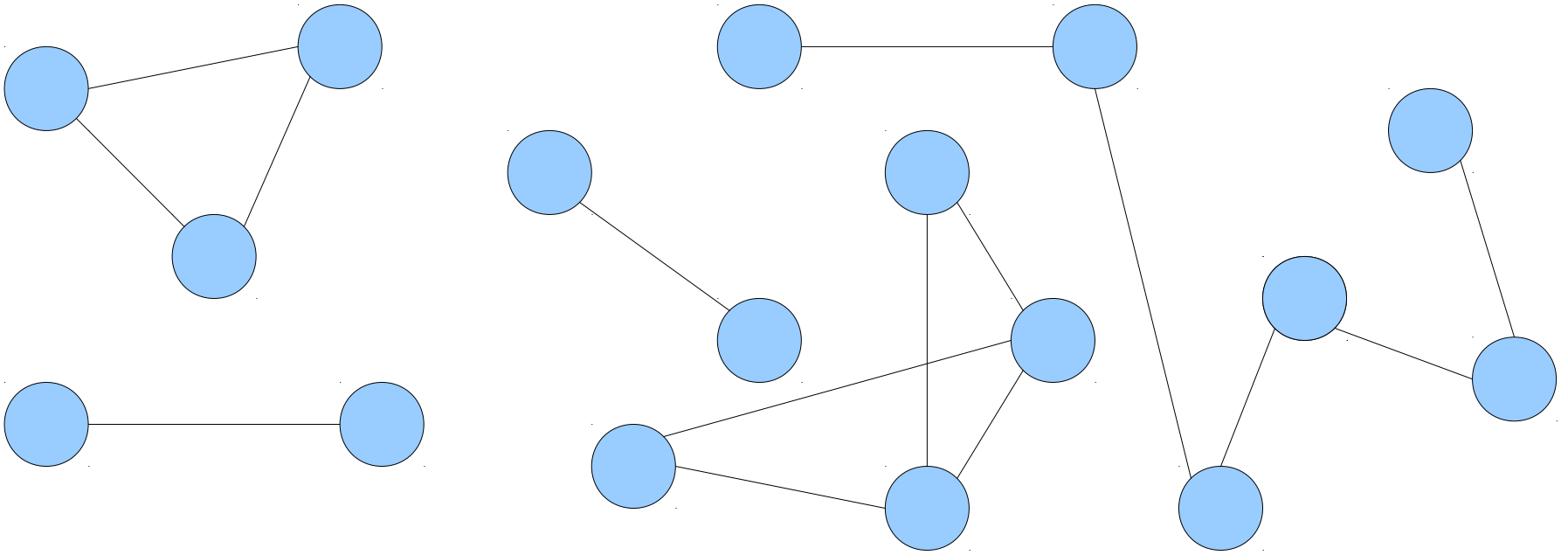


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

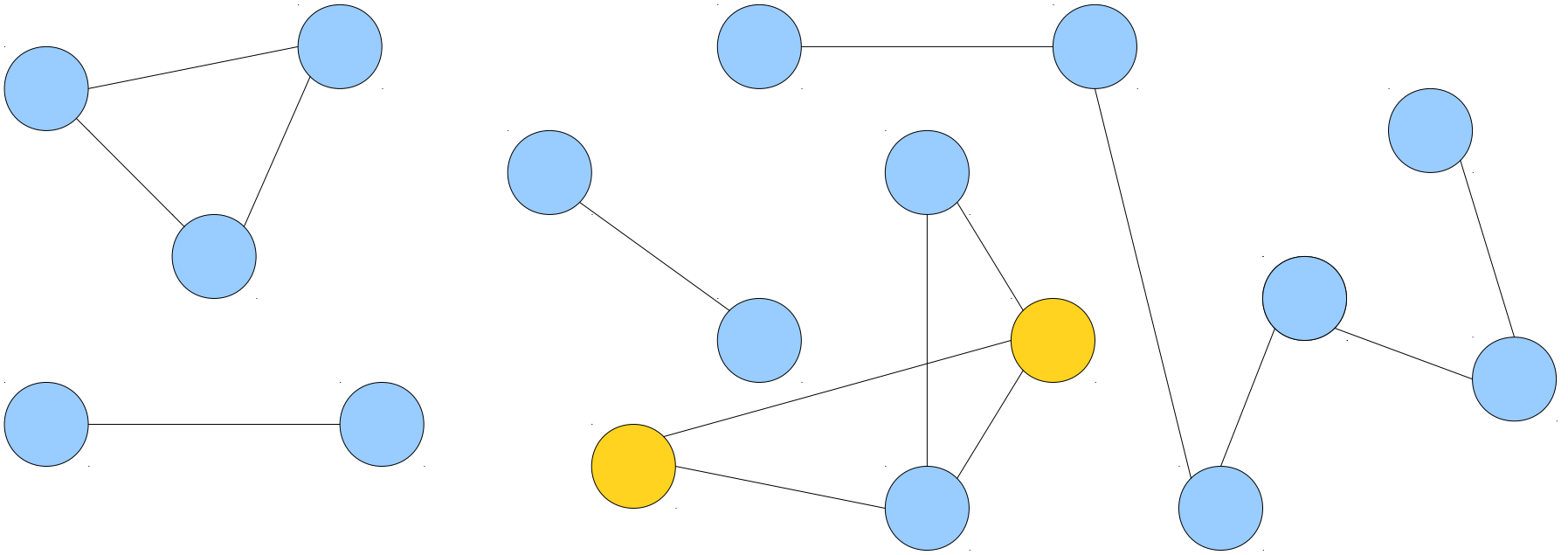


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

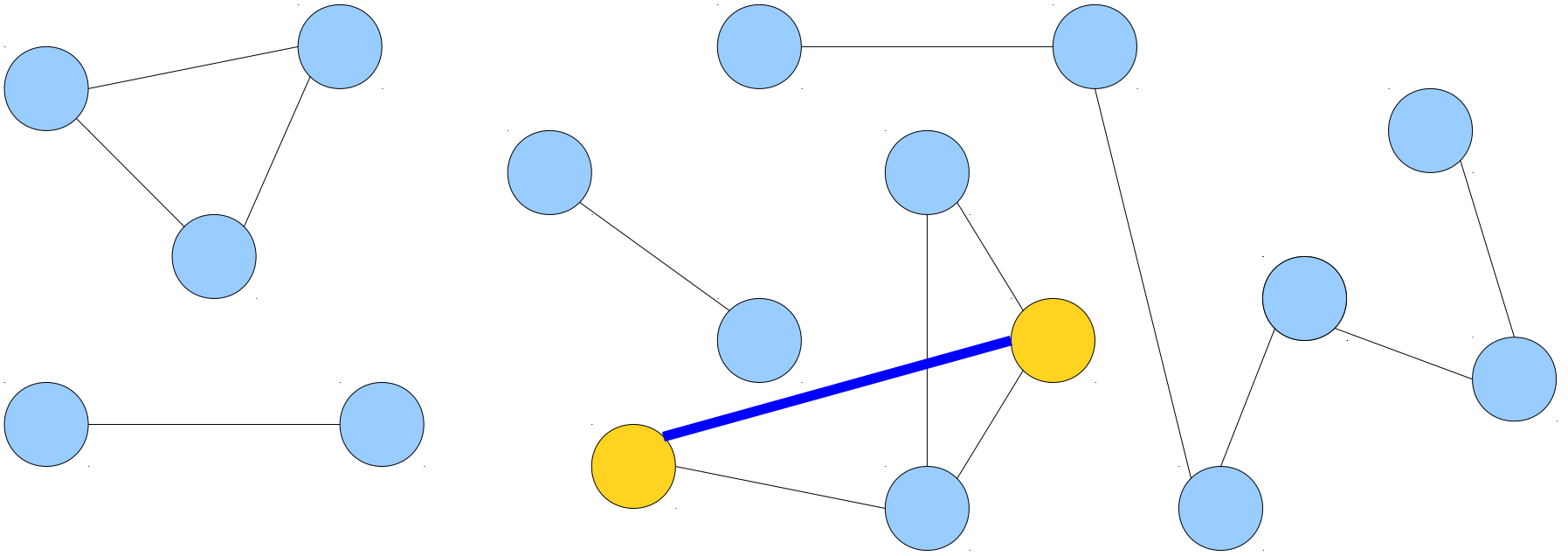


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

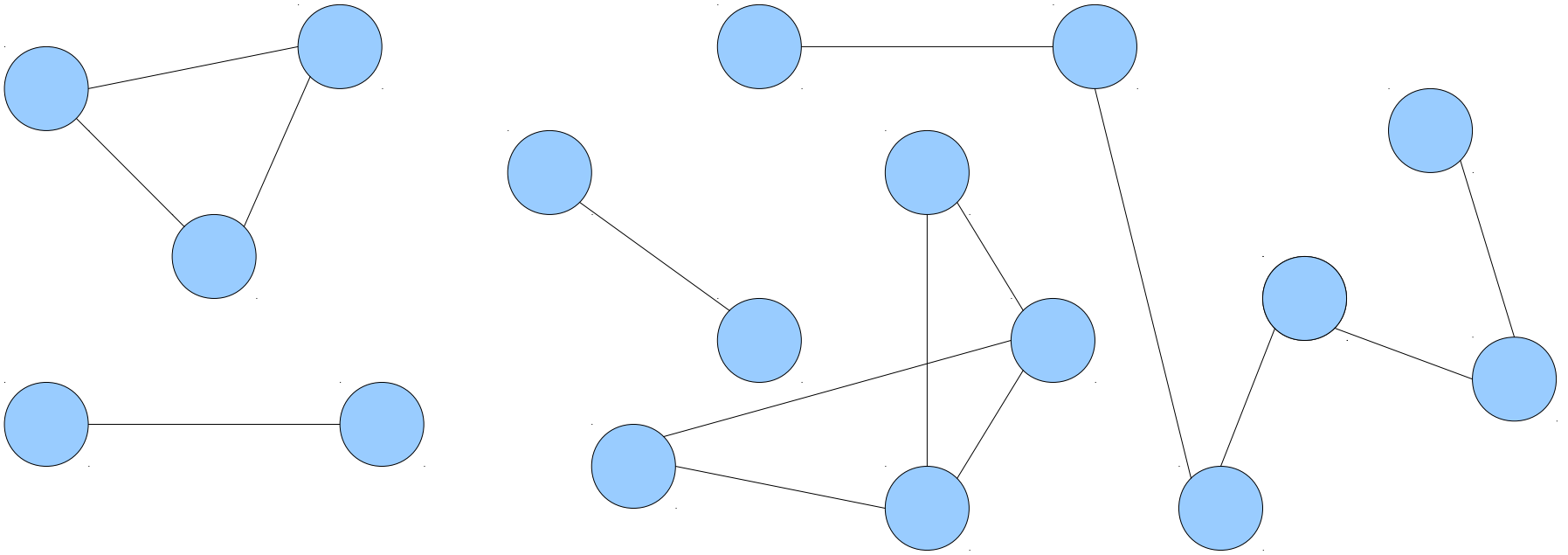


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

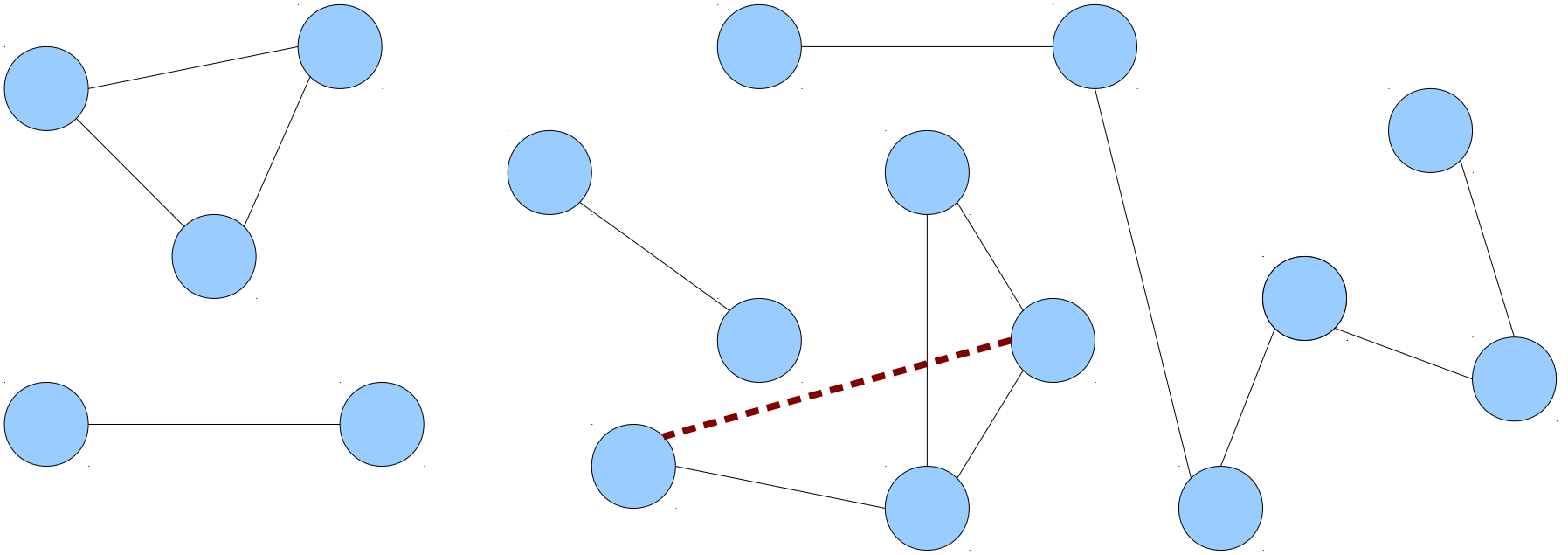


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

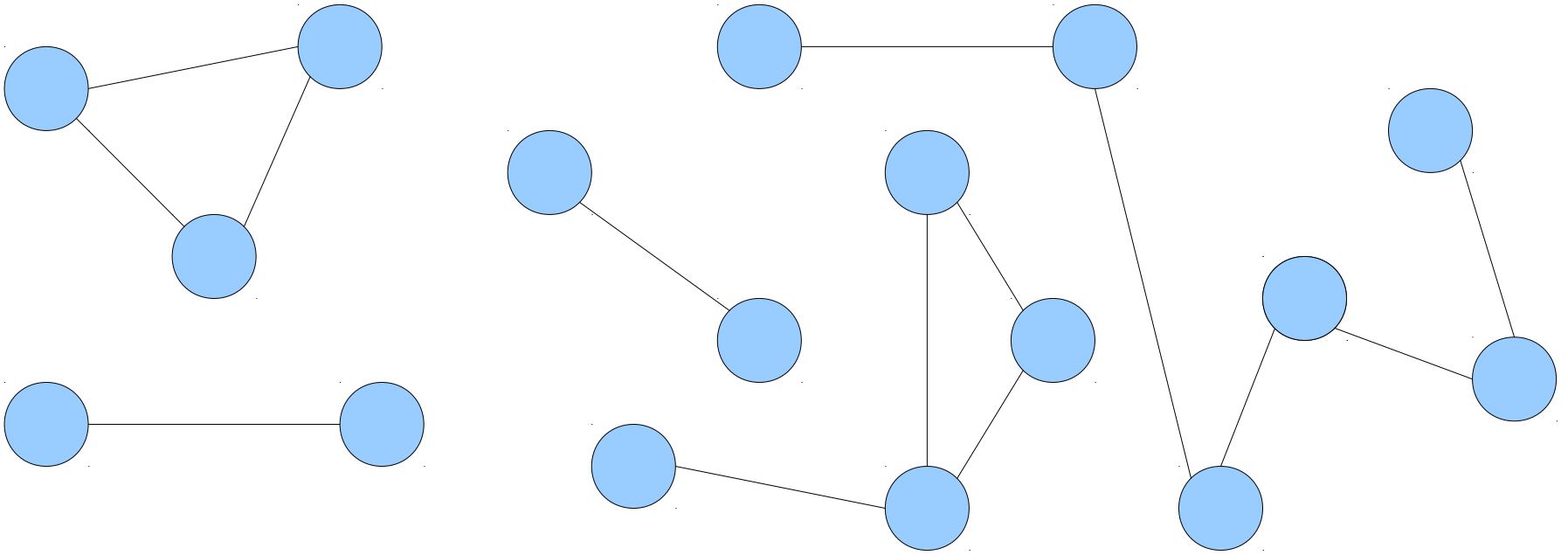


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

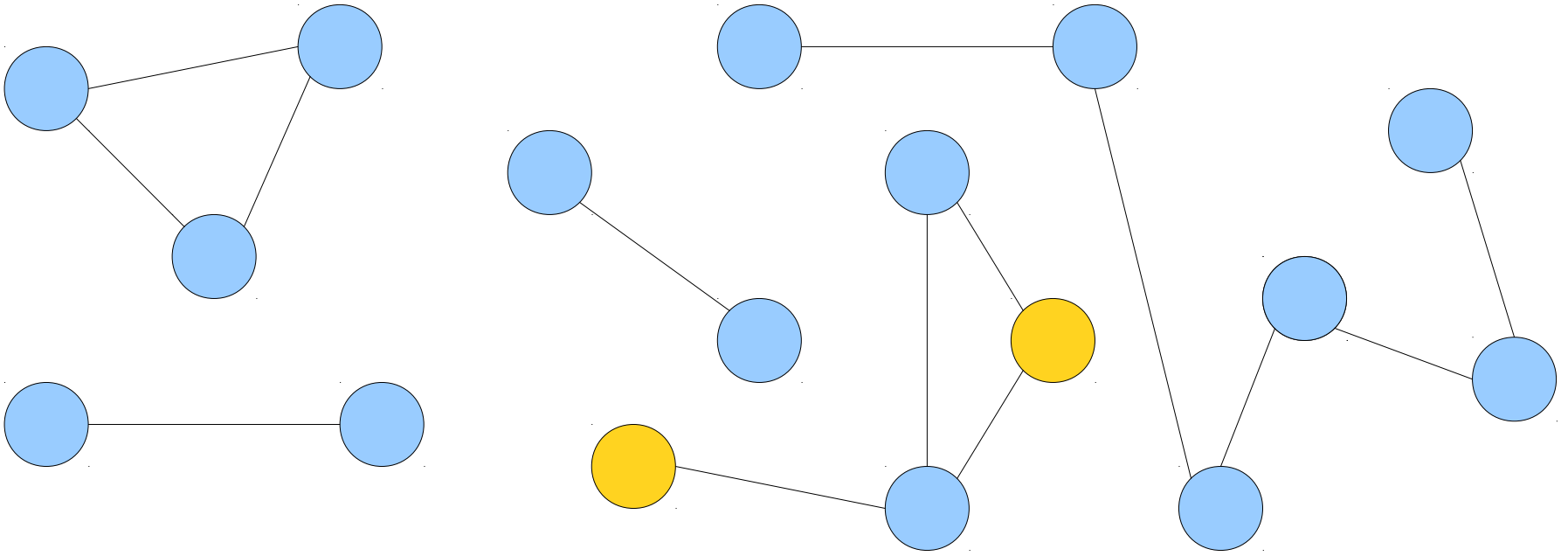


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

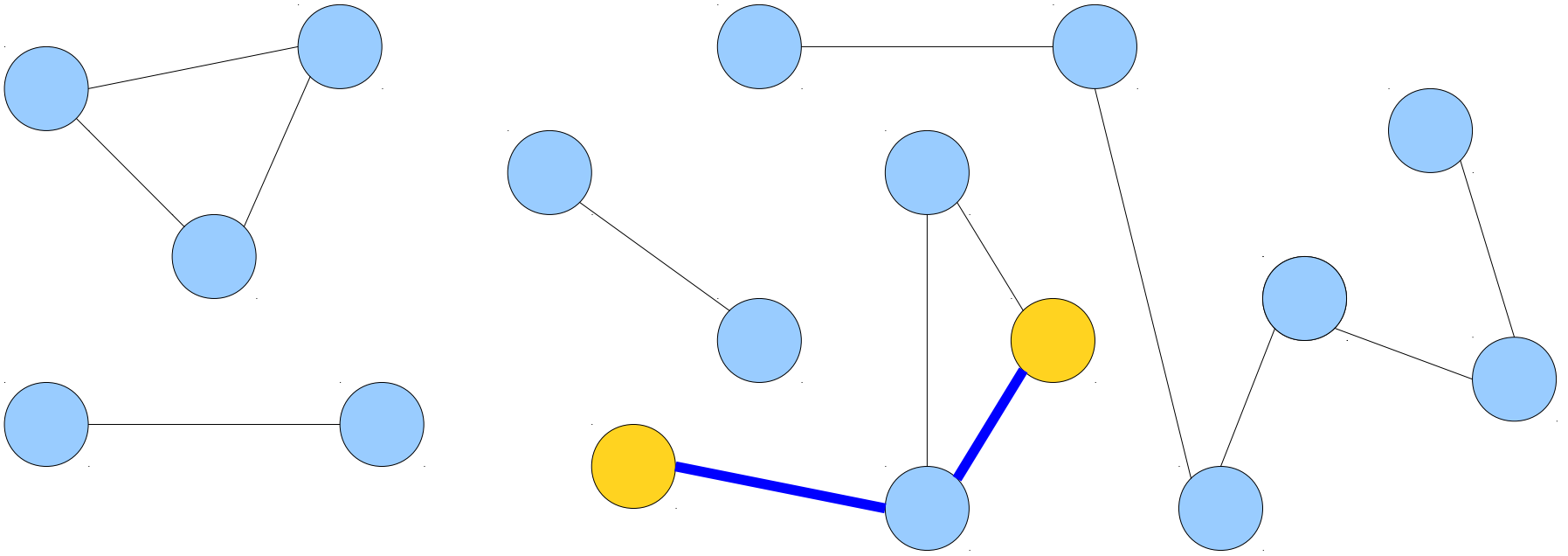


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

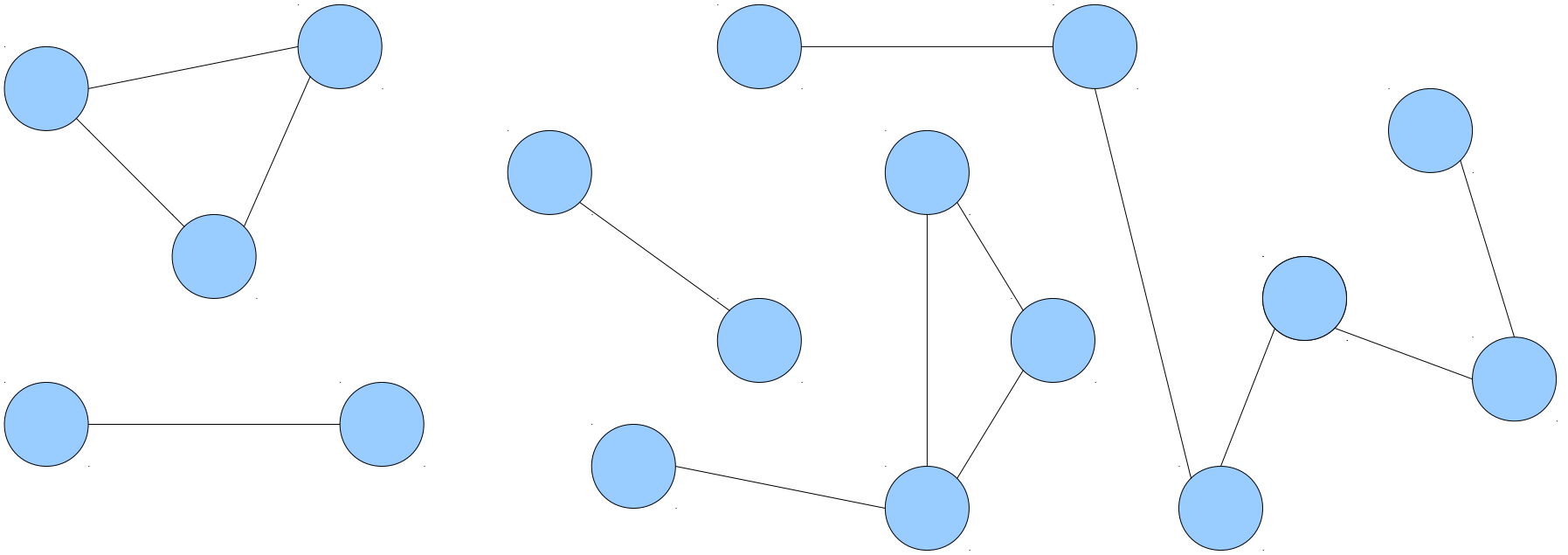


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

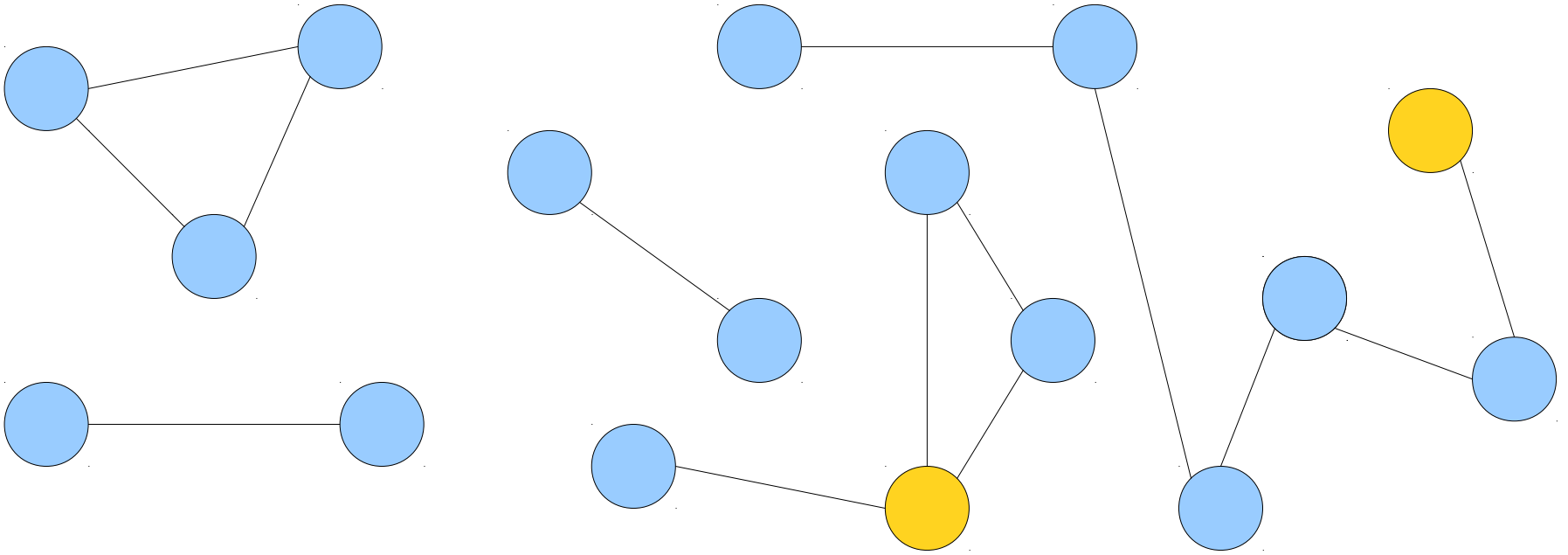


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

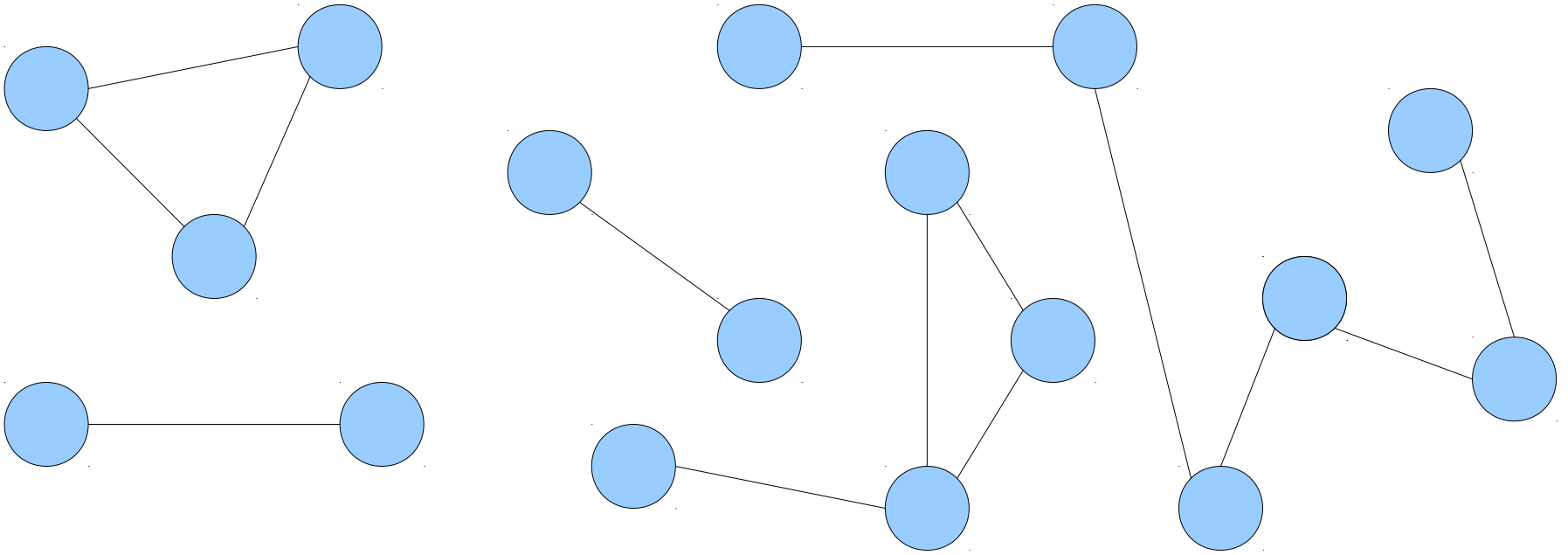


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

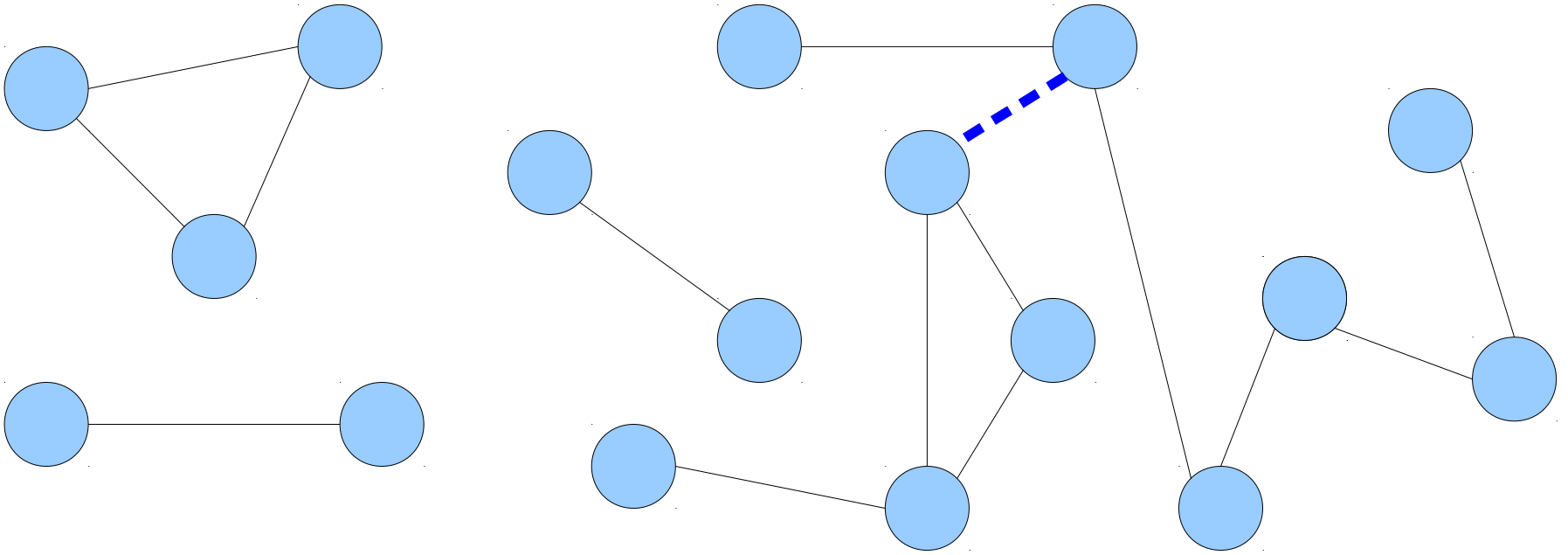


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

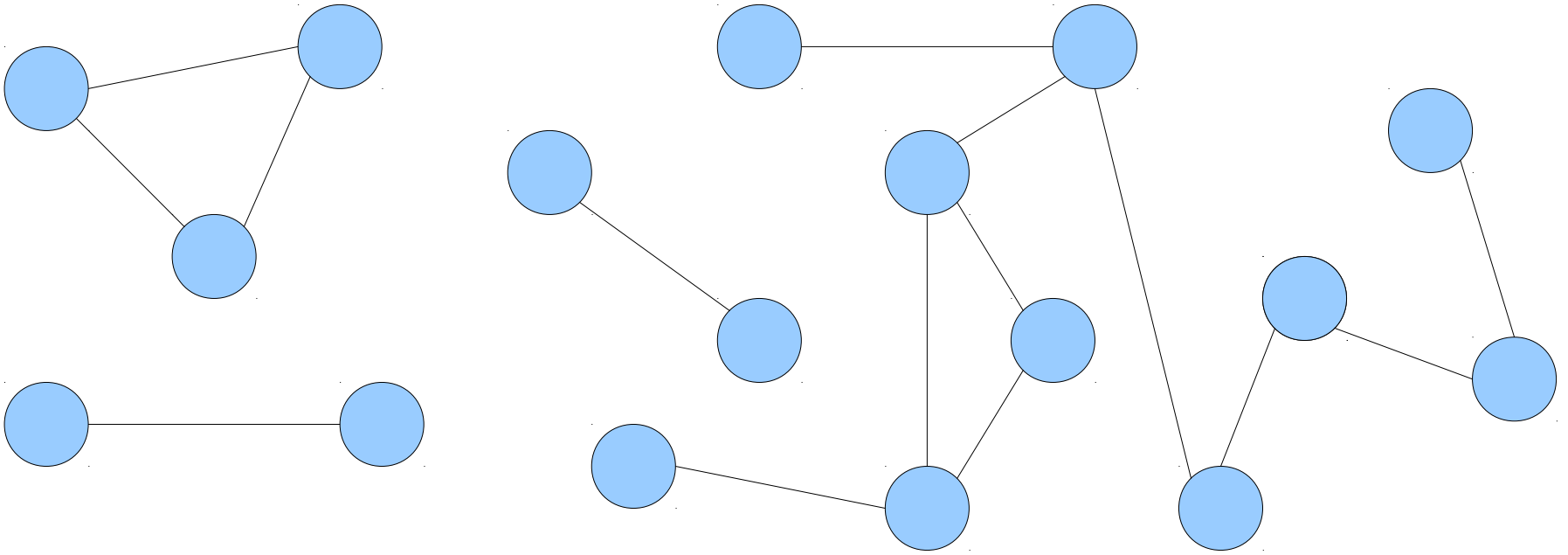


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

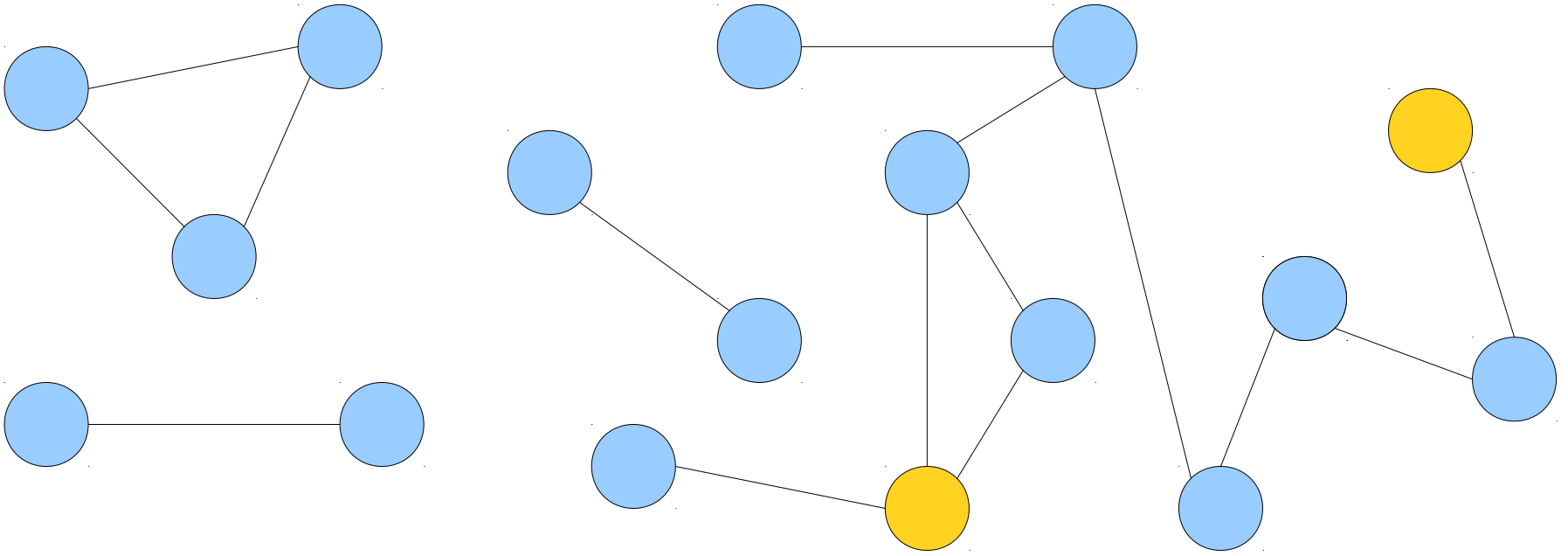


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!

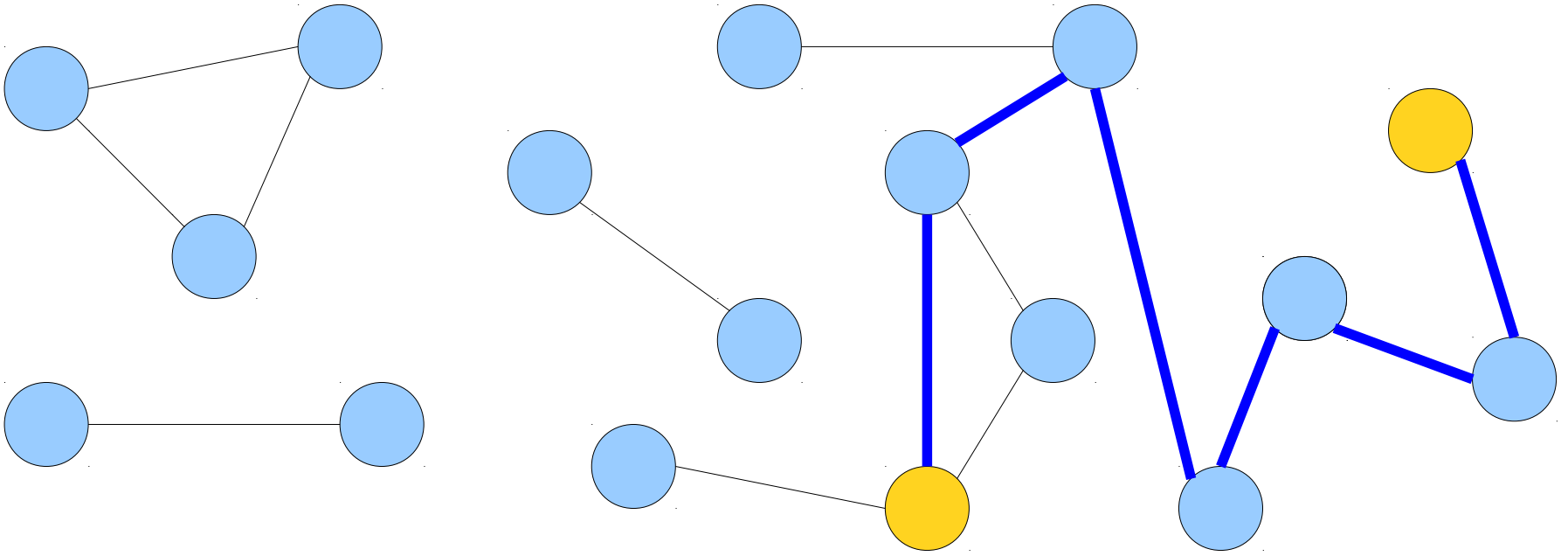


Dynamic Connectivity

- The **dynamic connectivity problem** is the following:

  Maintain an undirected graph G so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- This is a *much* harder problem!



Dynamic Connectivity

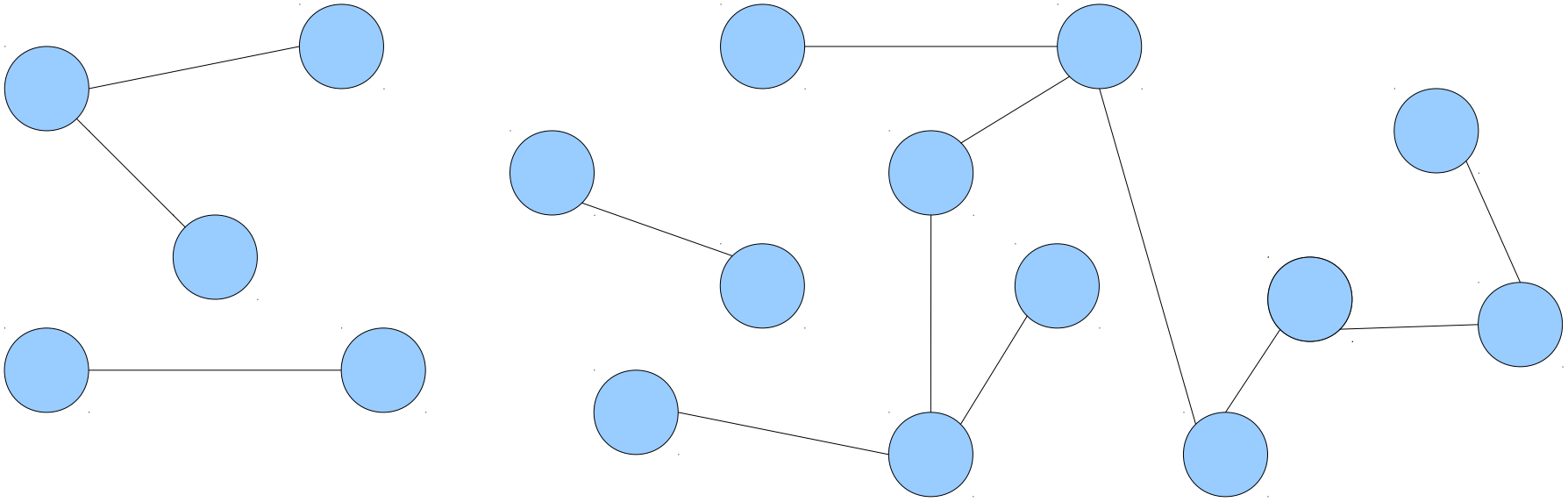
- Best known data structure:
  - Edge insertions and deletions take average time $O(\log^2 n)$ time.
    - (Notation: $\log^{(k)} n$ is log log … log n, k times. $\log^k n$ is $(\log n)^k$.
  - Connectivity queries take time $O(\log^2 n / \log \log n)$.
- **This is a topic for later in the quarter**. The solution is not trivial.
- Today, we'll look at a restricted version of the problem that will serve a building block for the general version.

Dynamic Connectivity in Forests

- Consider the following special-case of the dynamic connectivity problem:

  Maintain an undirected **forest** *G* so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- Each deleted edge splits a tree in two; each added edge joins two trees and never closes a cycle.

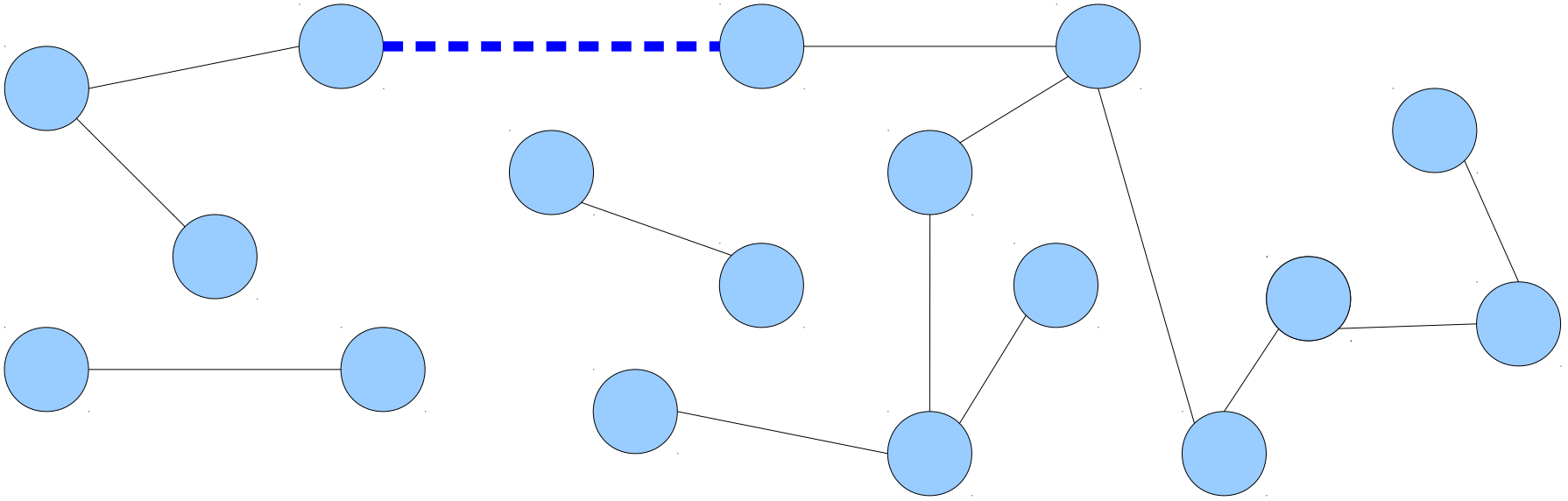


Dynamic Connectivity in Forests

- Consider the following special-case of the dynamic connectivity problem:

  Maintain an undirected **forest** *G* so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- Each deleted edge splits a tree in two; each added edge joins two trees and never closes a cycle.

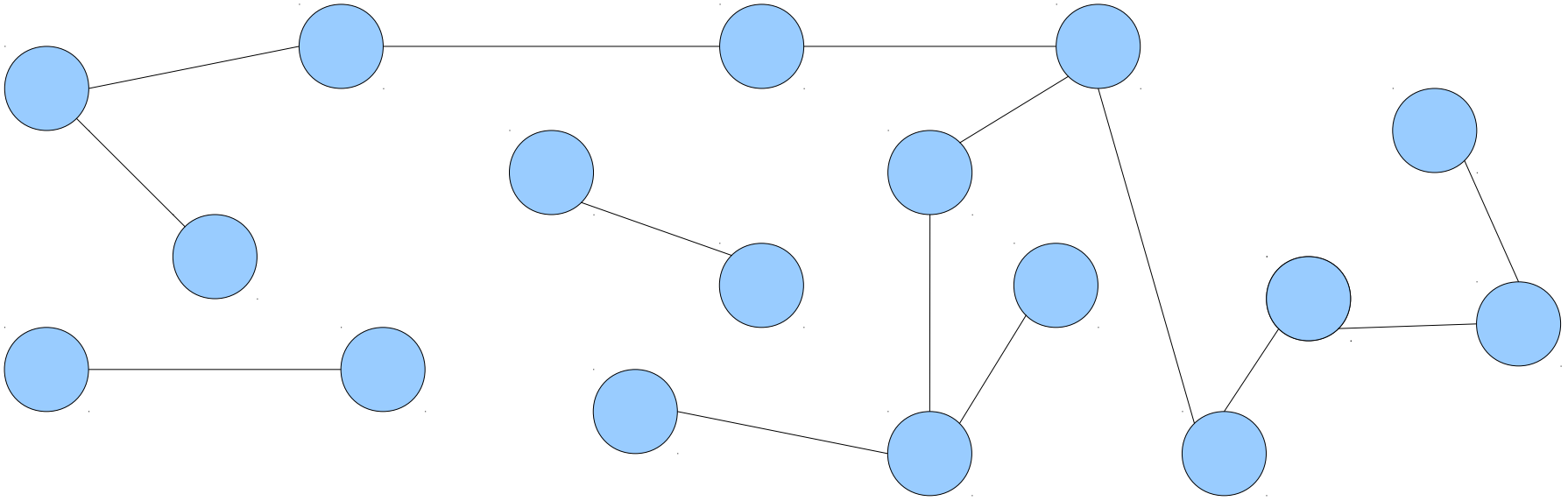


Dynamic Connectivity in Forests

- Consider the following special-case of the dynamic connectivity problem:

  Maintain an undirected **forest** *G* so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- Each deleted edge splits a tree in two; each added edge joins two trees and never closes a cycle.

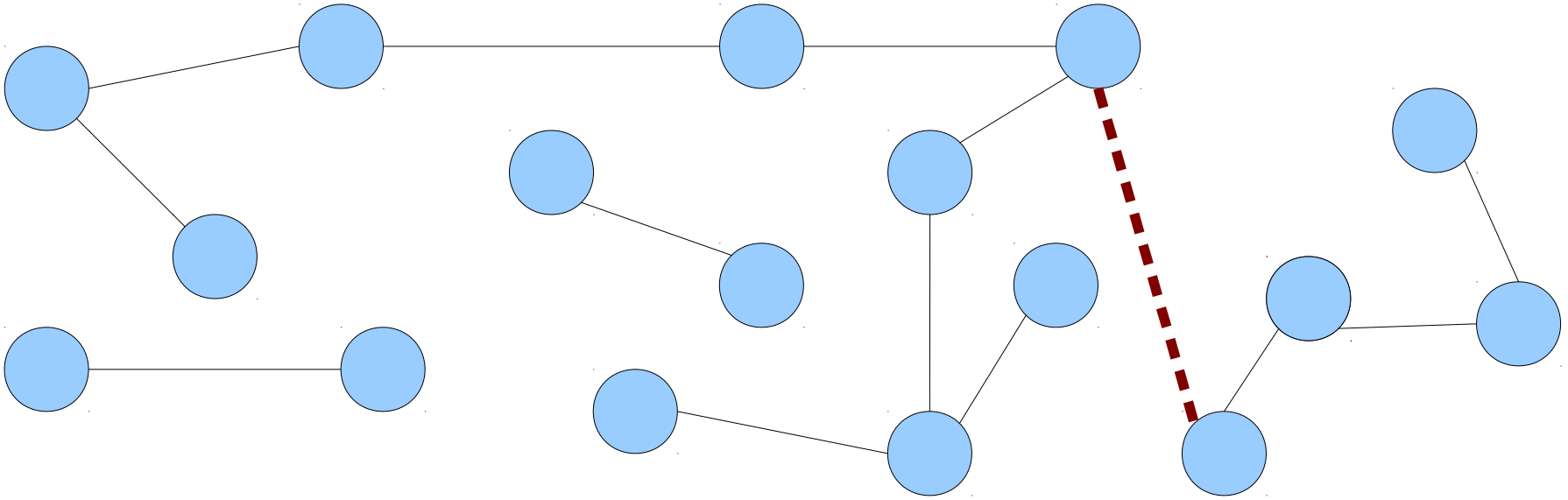


Dynamic Connectivity in Forests

- Consider the following special-case of the dynamic connectivity problem:

  Maintain an undirected **forest** *G* so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- Each deleted edge splits a tree in two; each added edge joins two trees and never closes a cycle.

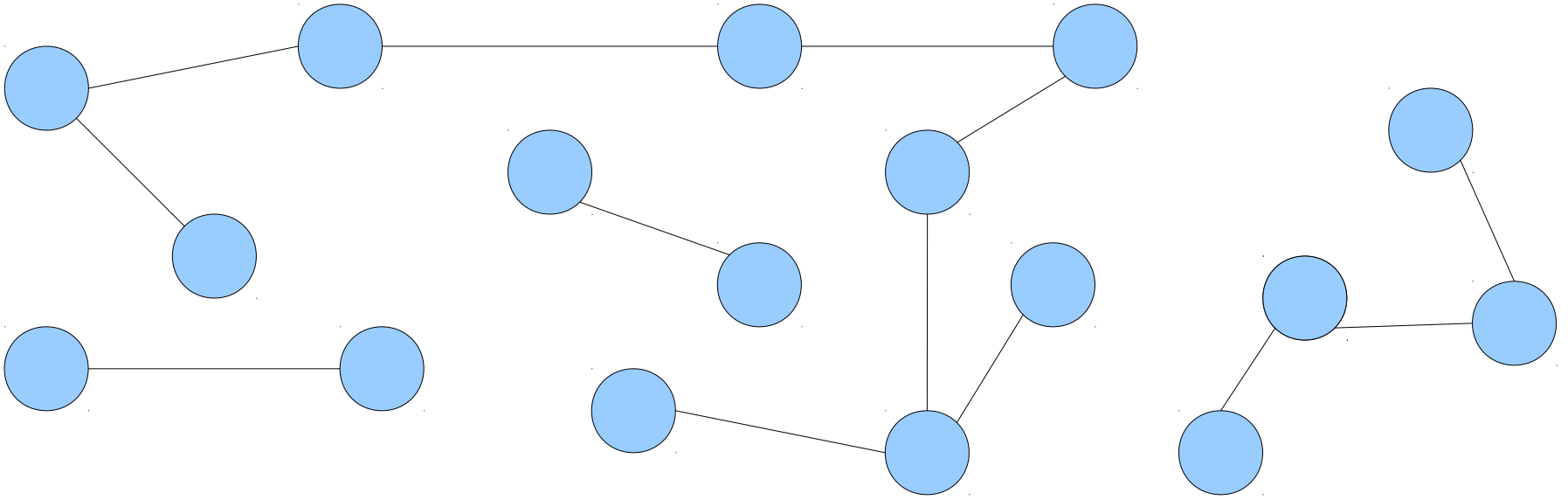


Dynamic Connectivity in Forests

- Consider the following special-case of the dynamic connectivity problem:

  Maintain an undirected **forest** *G* so that edges may be inserted an deleted and connectivity queries may be answered efficiently.

- Each deleted edge splits a tree in two; each added edge joins two trees and never closes a cycle.



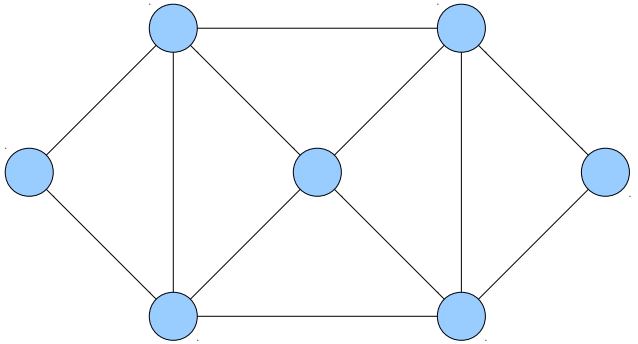
Dynamic Connectivity in Forests

- **Goal**: Support these three operations:
  - *link*(u, v): Add in edge $\{u, v\}$. The assumption is that u and v are in separate trees.
  - *cut*(u, v): Cut the edge $\{u, v\}$. The assumption is that the edge exists in the tree.
  - *is-connected*(u, v): Return whether u and v are connected.
- The data structure we'll develop can perform these operations time **O(log *n*)** each.

Euler Tours

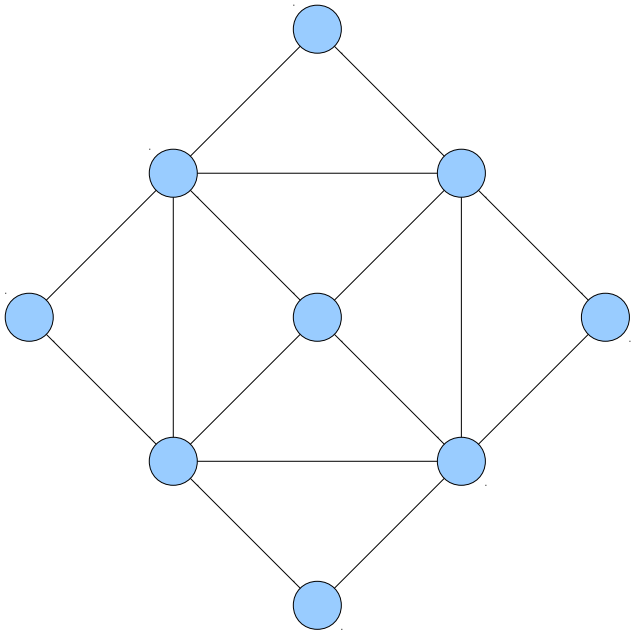
Euler Tours

- In a graph *G*, an **Euler tour** is a path through the graph that visits every edge exactly once.
- Mathematically formulates the “trace this figure without picking up your pencil or redrawing any lines” puzzles.



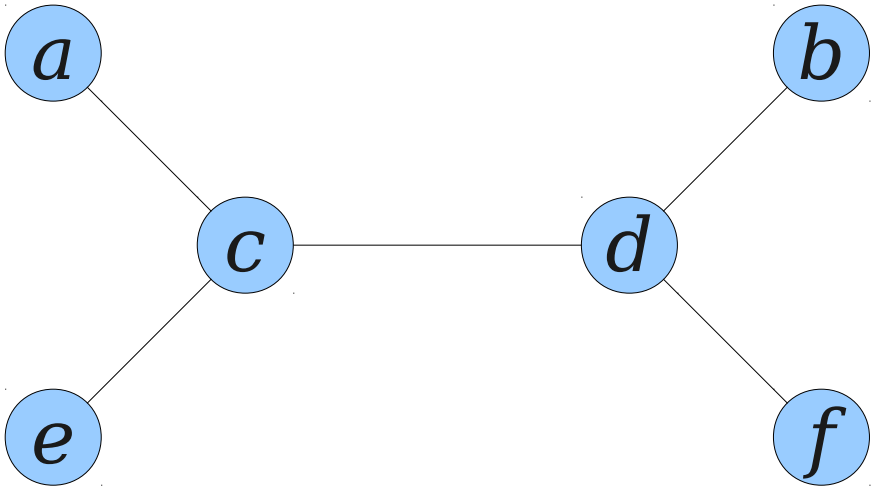
Euler Tours

- In a graph *G*, an **Euler tour** is a path through the graph that visits every edge exactly once.
- Mathematically formulates the “trace this figure without picking up your pencil or redrawing any lines” puzzles.



Euler Tours on Trees

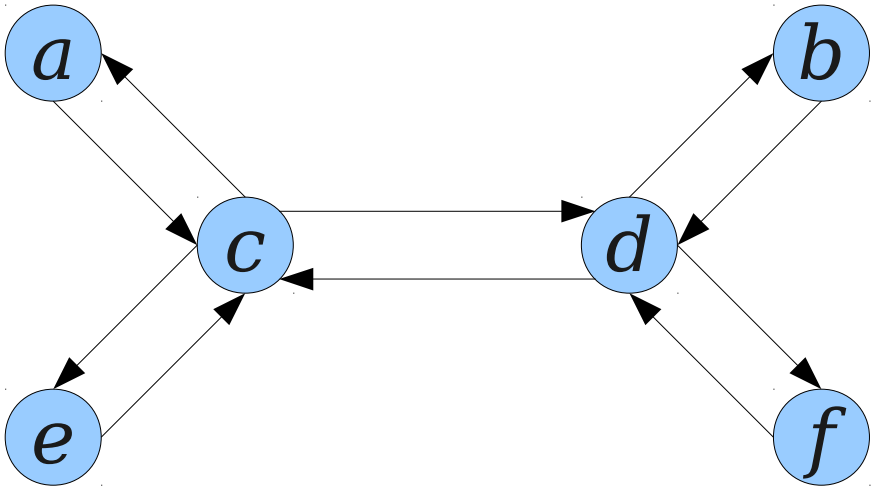
- In general, trees do not have Euler tours.



- **Technique:** replace each edge $\{u, v\}$ with two edges (u, v) and (v, u).
- Resulting graph has an Euler tour.

Euler Tours on Trees

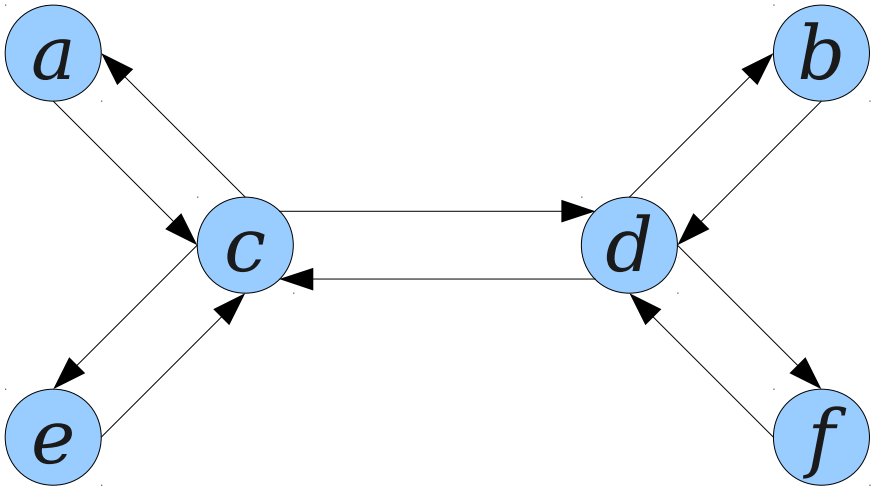
- In general, trees do not have Euler tours.



- **Technique:** replace each edge $\{u, v\}$ with two edges (u, v) and (v, u).
- Resulting graph has an Euler tour.

Euler Tours on Trees

- In general, trees do not have Euler tours.

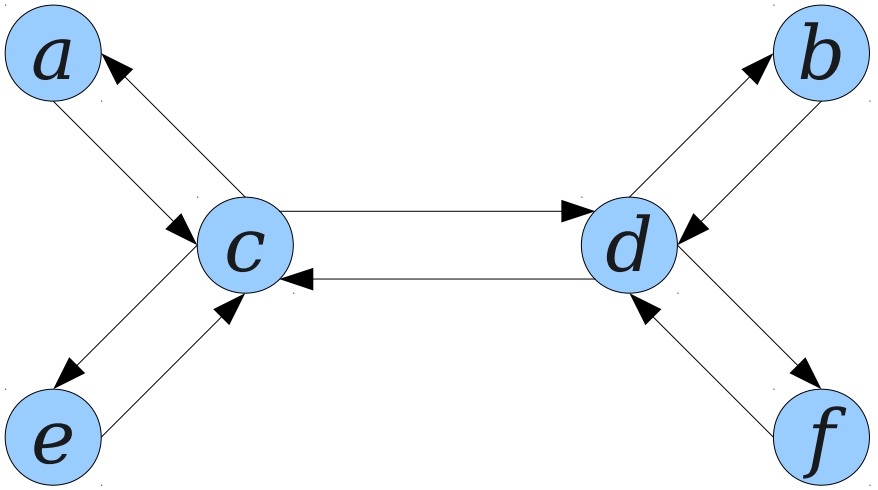


a c d b d f d c e c a

- **Technique:** replace each edge $\{u, v\}$ with two edges (u, v) and (v, u).
- Resulting graph has an Euler tour.

Euler Tours on Trees

- In general, trees do not have Euler tours.

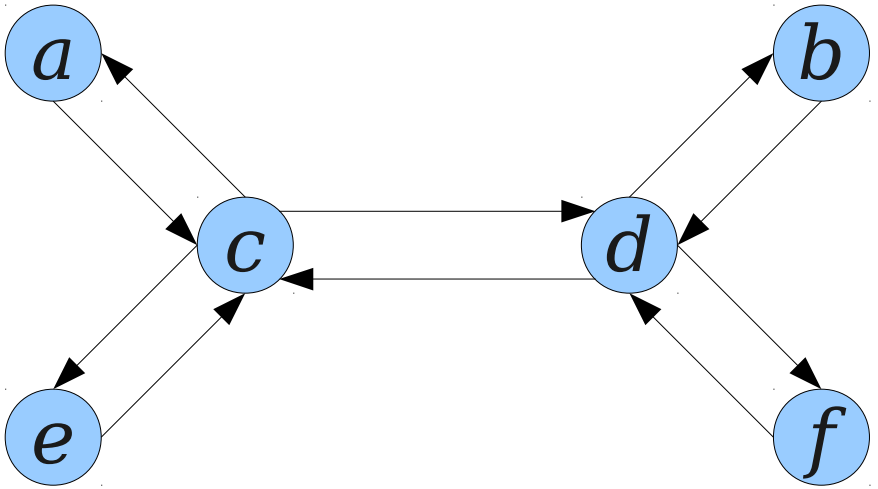


c a c e c d b d f d c

- **Technique:** replace each edge $\{u, v\}$ with two edges (u, v) and (v, u).
- Resulting graph has an Euler tour.

Euler Tours on Trees

- In general, trees do not have Euler tours.



a c e c d f d b d c a

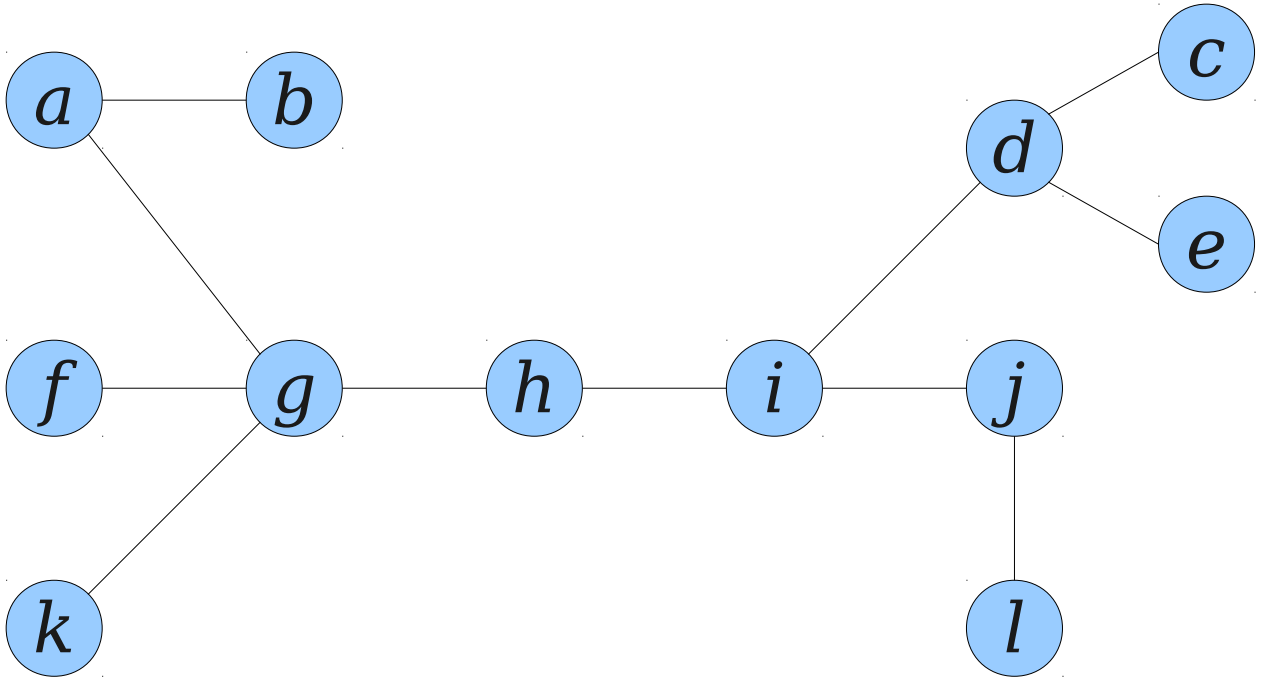
- **Technique:** replace each edge $\{u, v\}$ with two edges (u, v) and (v, u).
- Resulting graph has an Euler tour.

Euler Tours on Trees

- The data structure we'll design today is called an **Euler tour tree**.
- **High-level idea:** Instead of storing the trees in the forest, store their Euler tours.
- Each edge insertion or deletion translates into a set of manipulations on the Euler tours of the trees in the forest.
- Checking whether two nodes are connected can be done by checking if they're in the same Euler tour.

Properties of Euler Tours

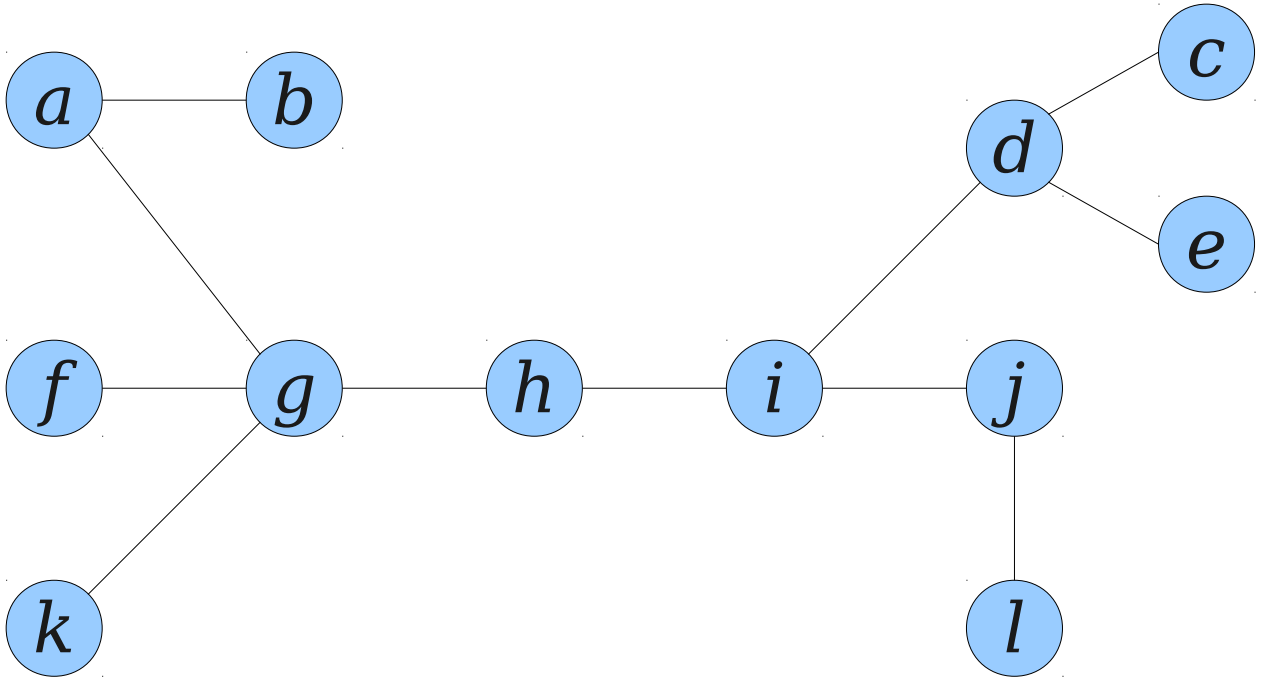
- The sequence of nodes visited in an Euler tour of a tree is closely connected to the structure of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Properties of Euler Tours

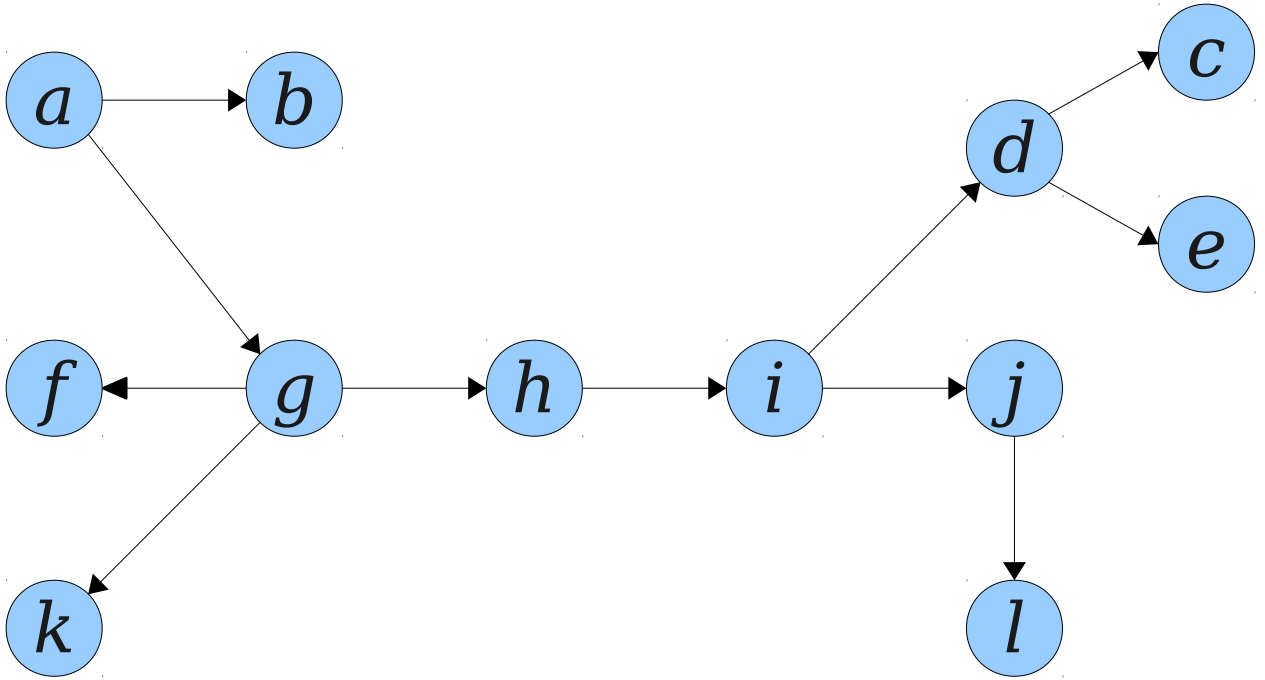
- The sequence of nodes visited in an Euler tour of a tree is closely connected to the structure of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Properties of Euler Tours

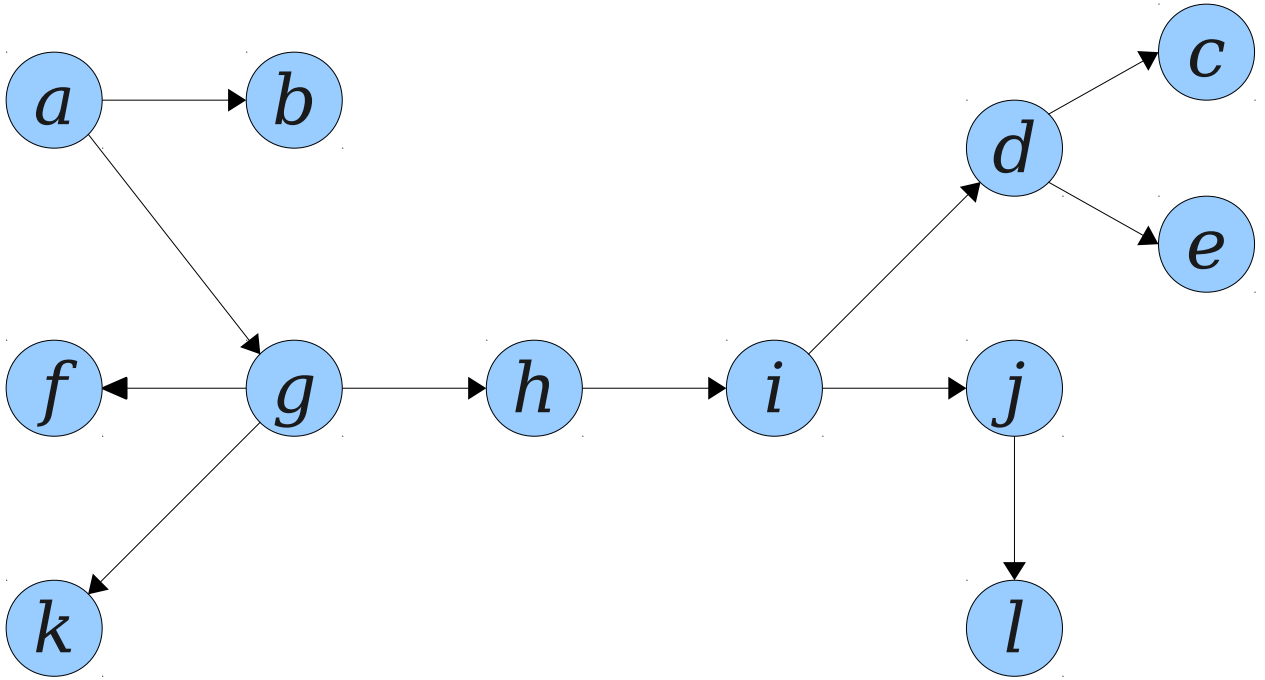
- The sequence of nodes visited in an Euler tour of a tree is closely connected to the structure of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Properties of Euler Tours

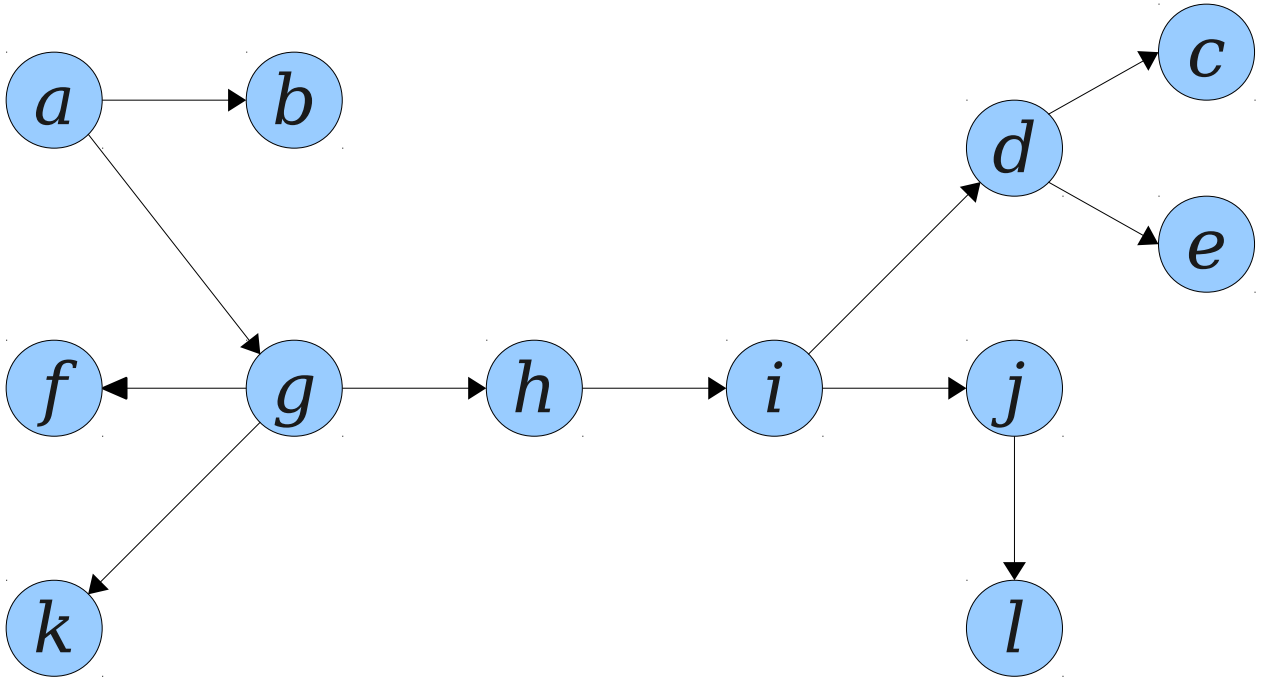
- The sequence of nodes visited in an Euler tour of a tree is closely connected to the structure of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Properties of Euler Tours

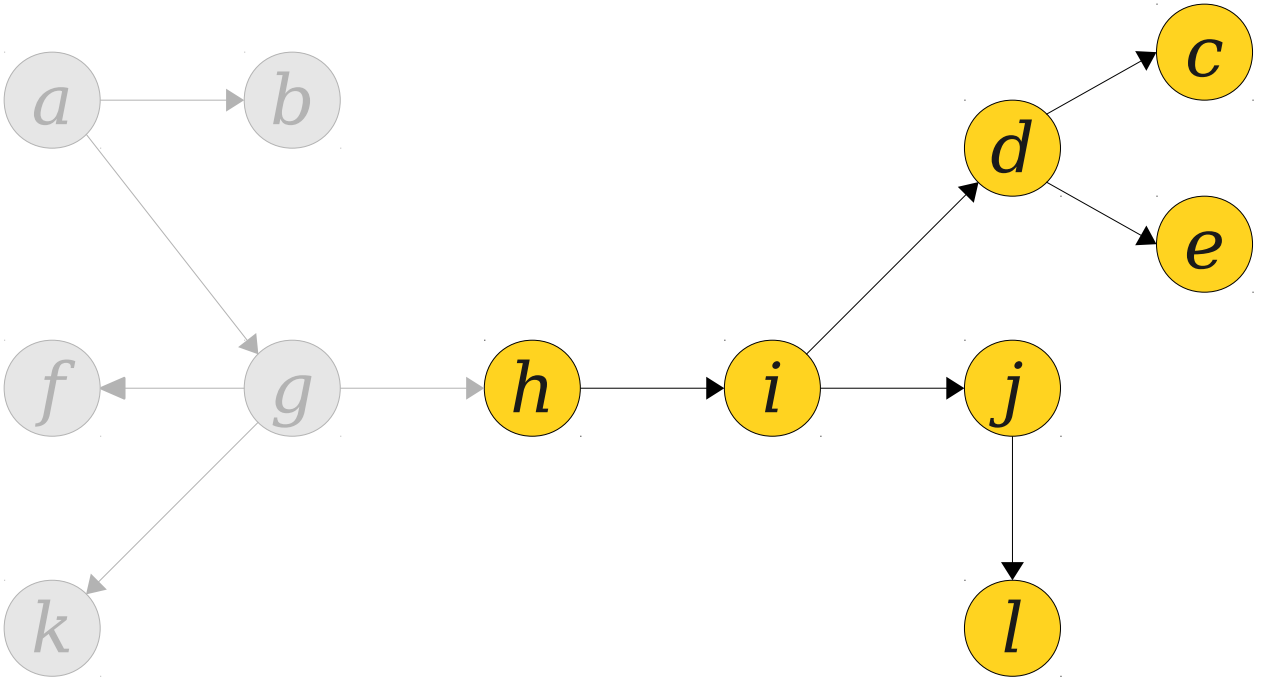
- The sequence of nodes visited in an Euler tour of a tree is closely connected to the structure of the tree.



a b a g ***h i d c d e d i j l j i h*** *g f g k g a*

Properties of Euler Tours

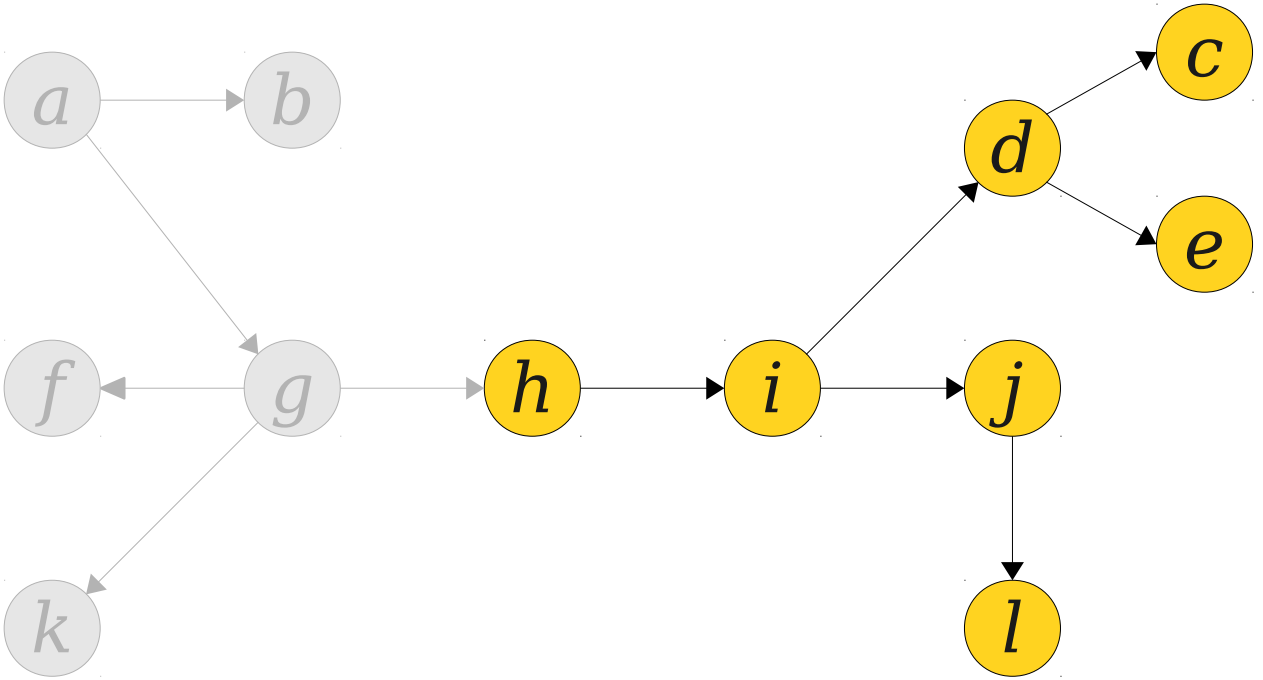
- The sequence of nodes visited in an Euler tour of a tree is closely connected to the structure of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Properties of Euler Tours

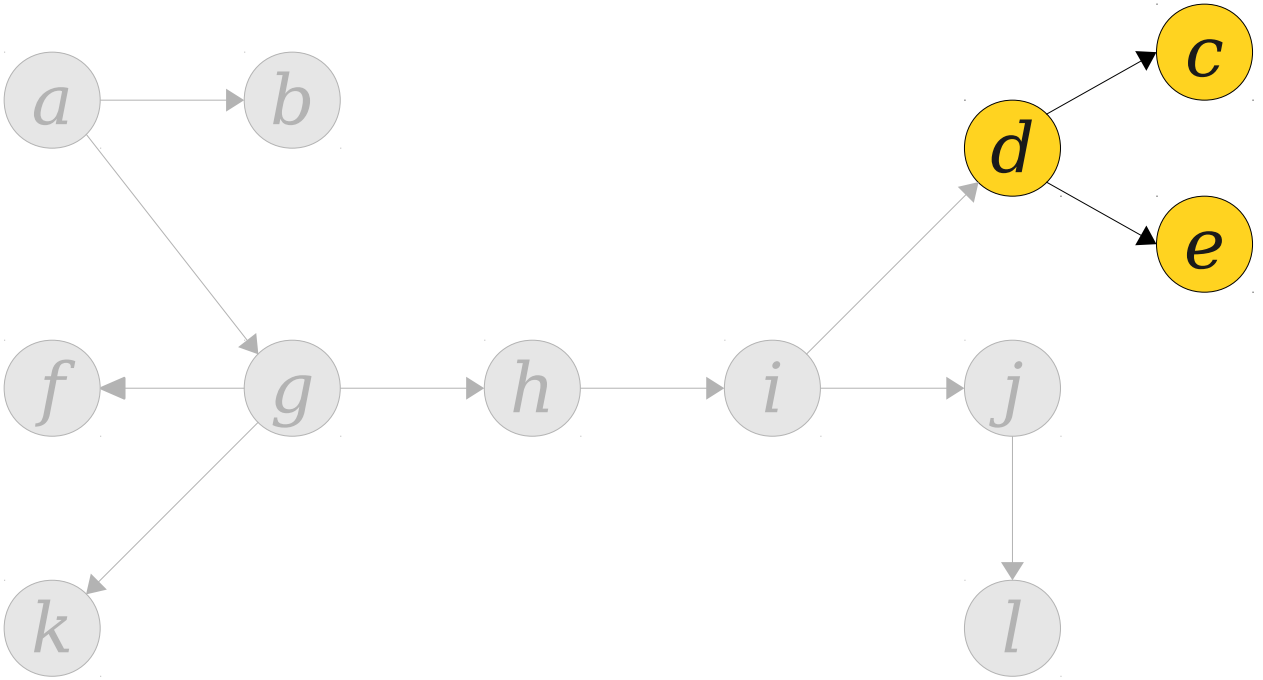
- The sequence of nodes visited in an Euler tour of a tree is closely connected to the structure of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Properties of Euler Tours

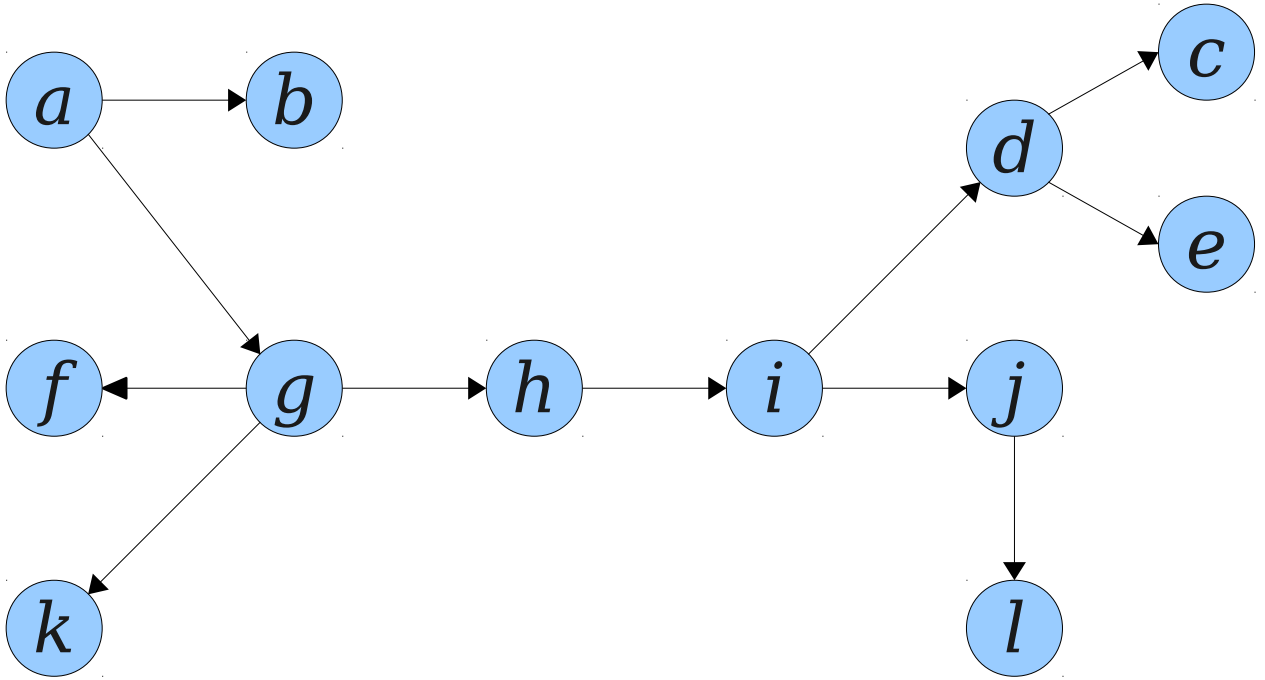
- The sequence of nodes visited in an Euler tour of a tree is closely connected to the structure of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Properties of Euler Tours

- The sequence of nodes visited in an Euler tour of a tree is closely connected to the structure of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Properties of Euler Tours

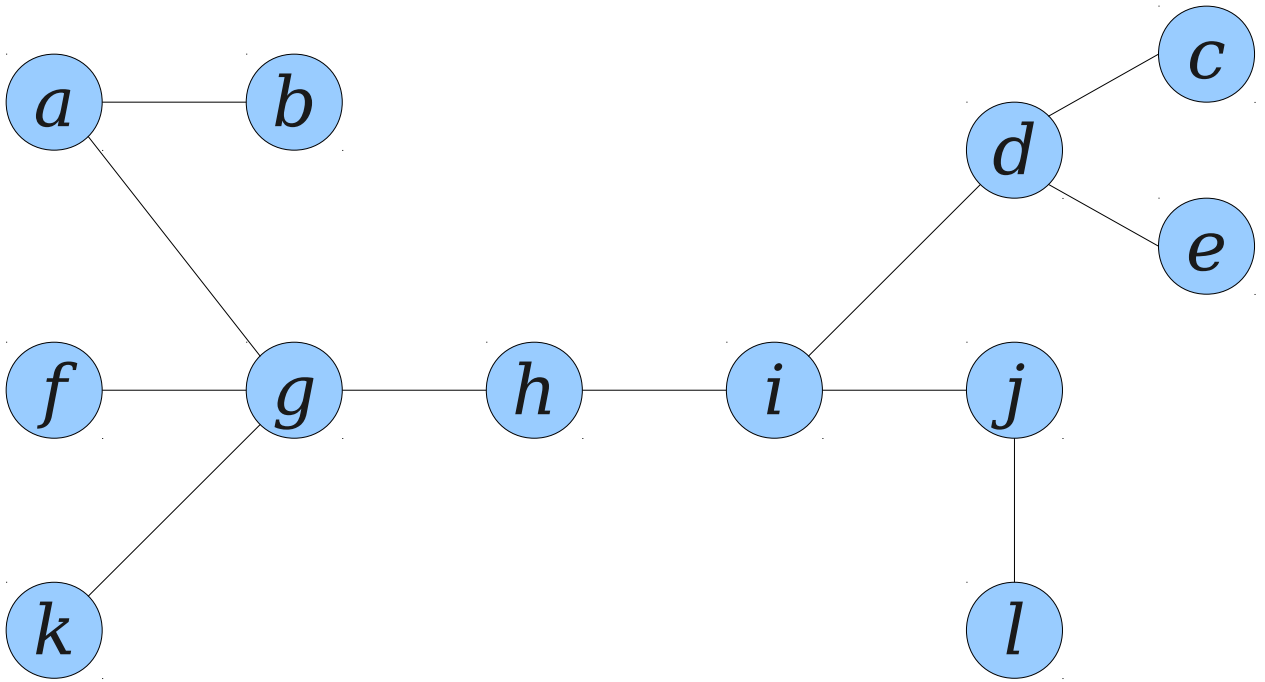
- The sequence of nodes visited in an Euler tour of a tree is closely connected to the structure of the tree.
- Begin by directing all edges toward the the first node in the tour.
- **Claim:** The sequences of nodes visited between the first and last instance of a node v gives an Euler tour of the subtree rooted at v.

Rerooting a Tour

- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.

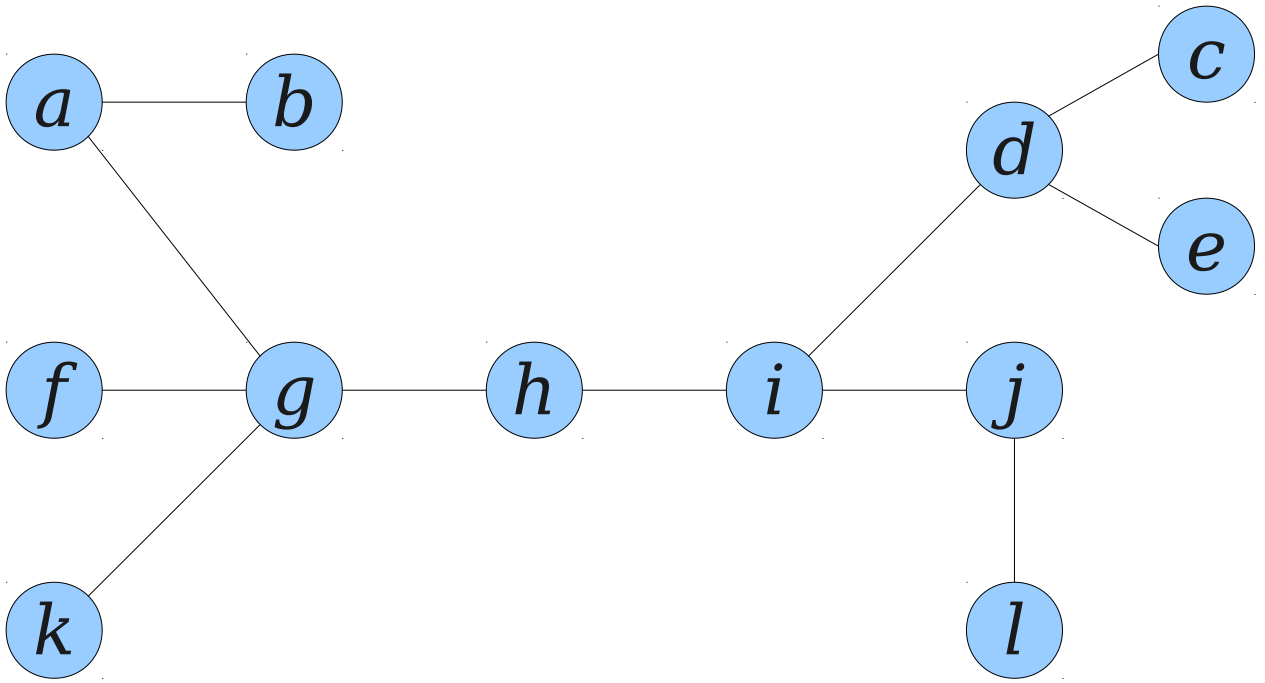
Rerooting a Tour

- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



Rerooting a Tour

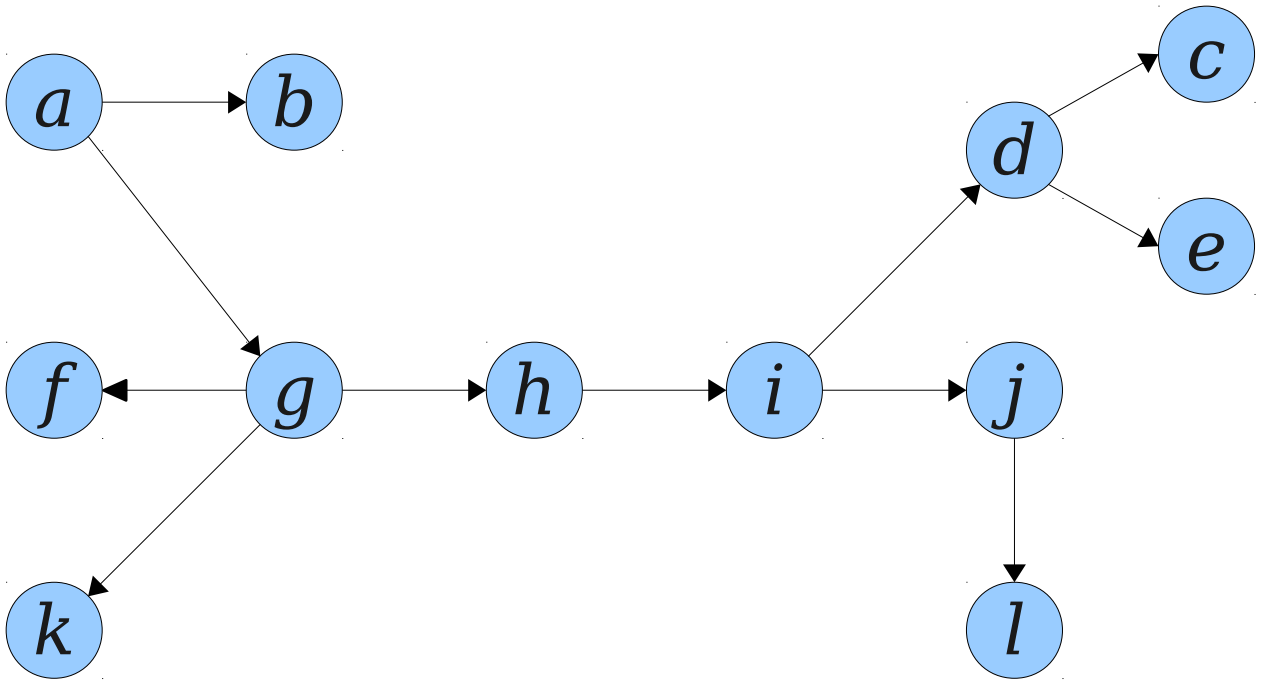
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Rerooting a Tour

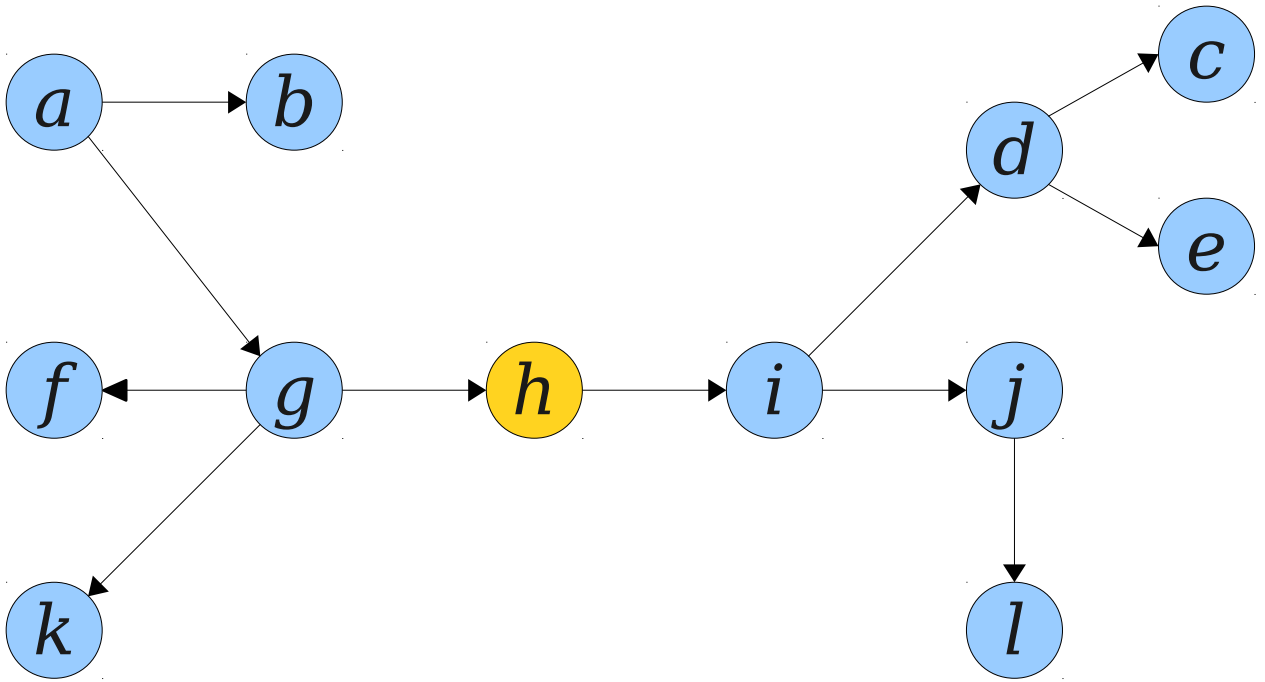
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Rerooting a Tour

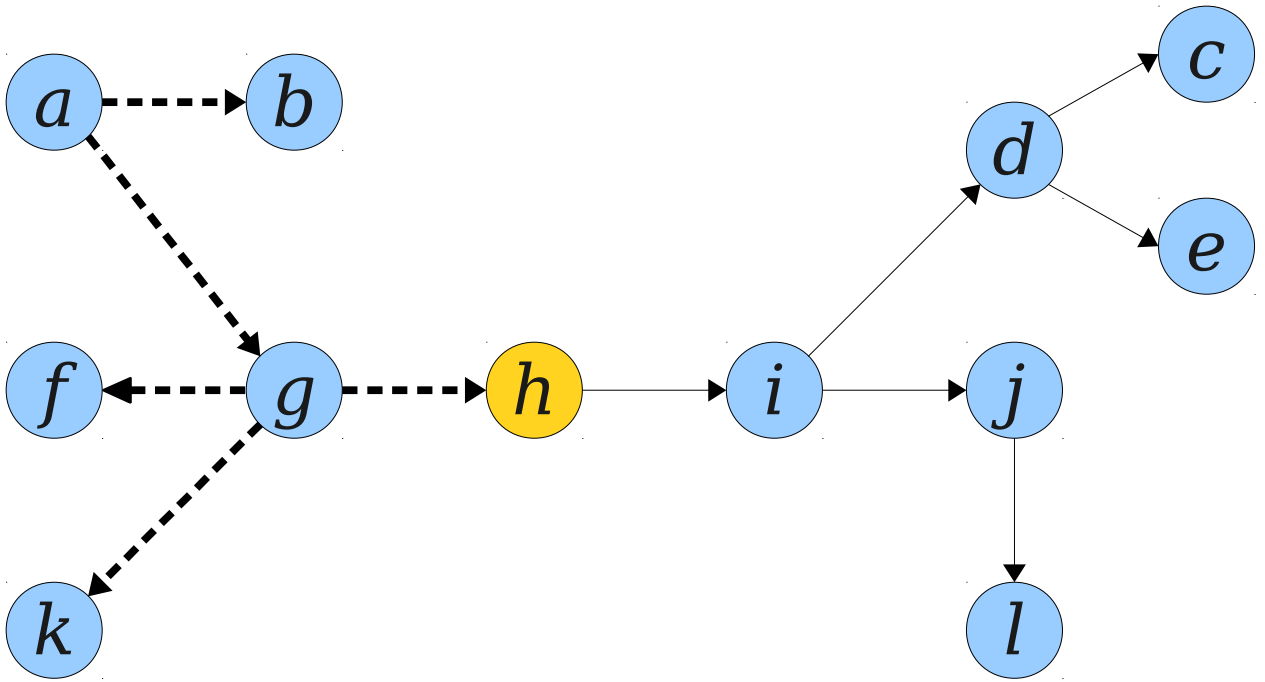
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Rerooting a Tour

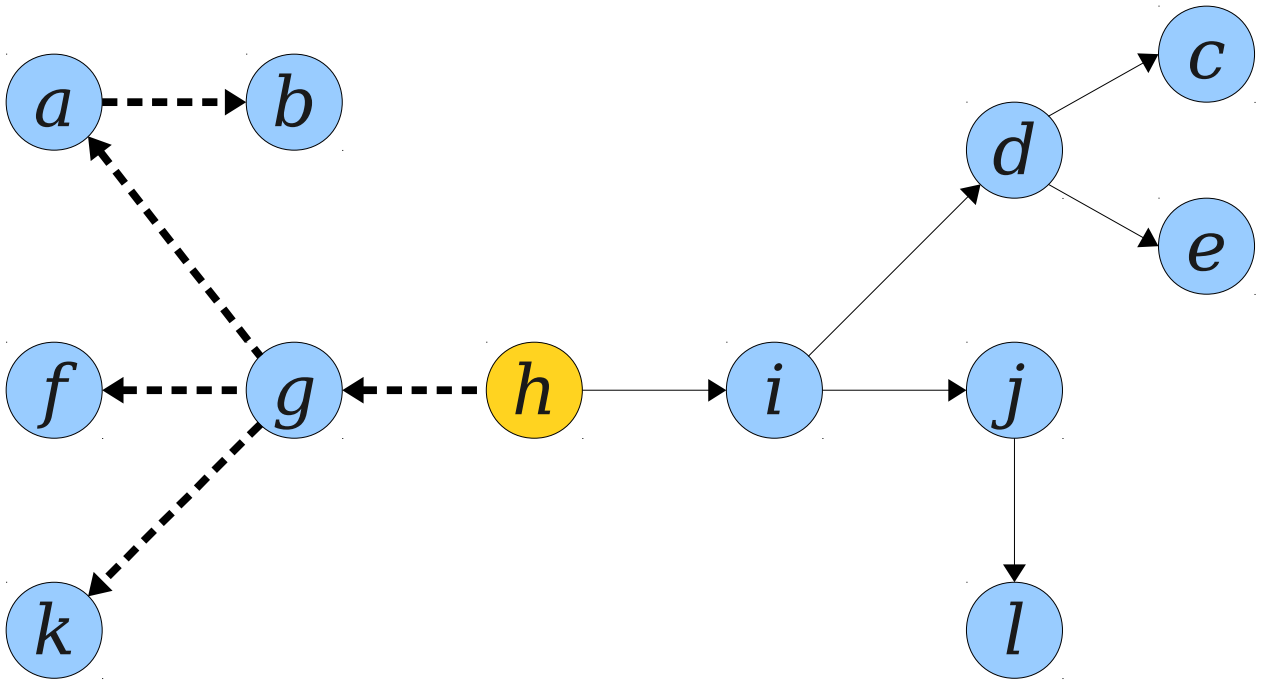
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Rerooting a Tour

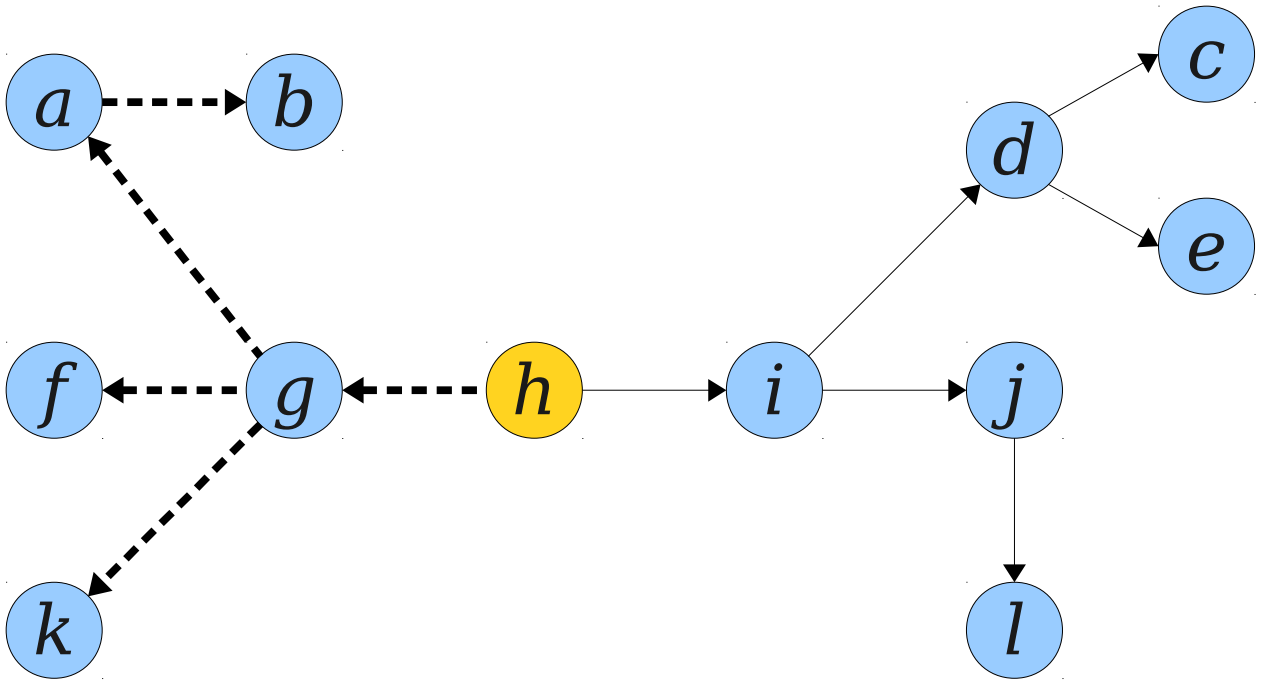
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Rerooting a Tour

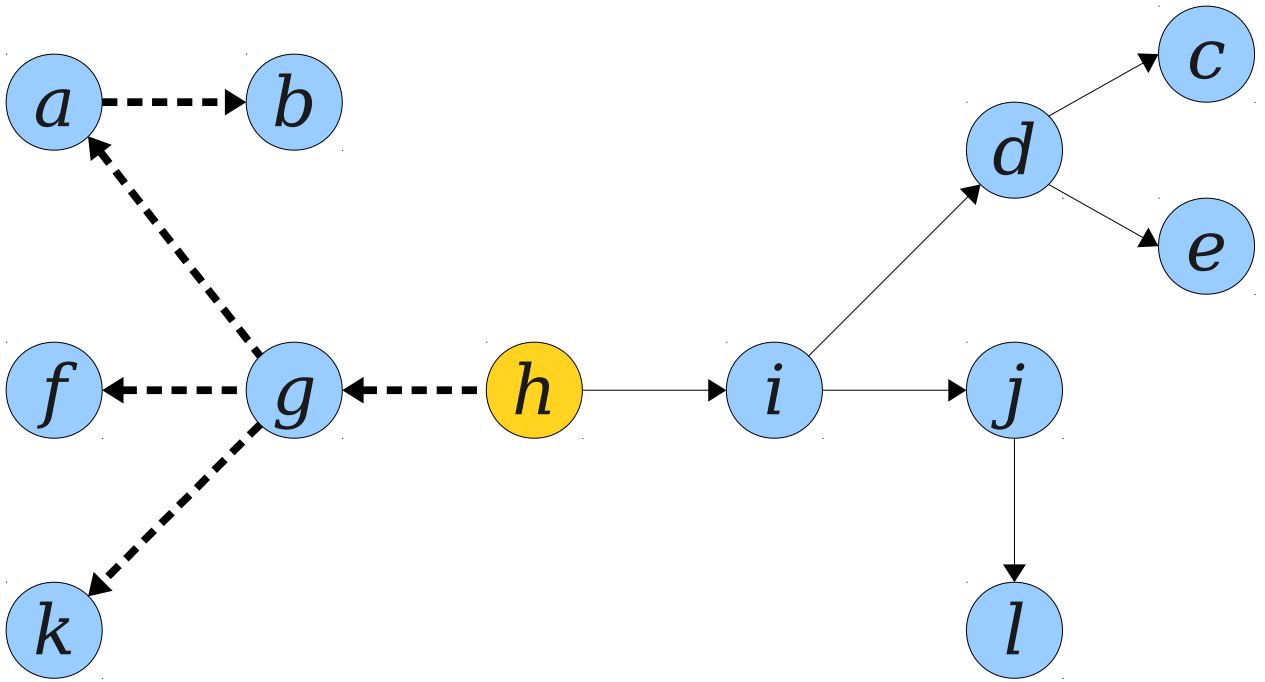
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



a b a g ***h i d c d e d i j l j i h*** *g f g k g a*

Rerooting a Tour

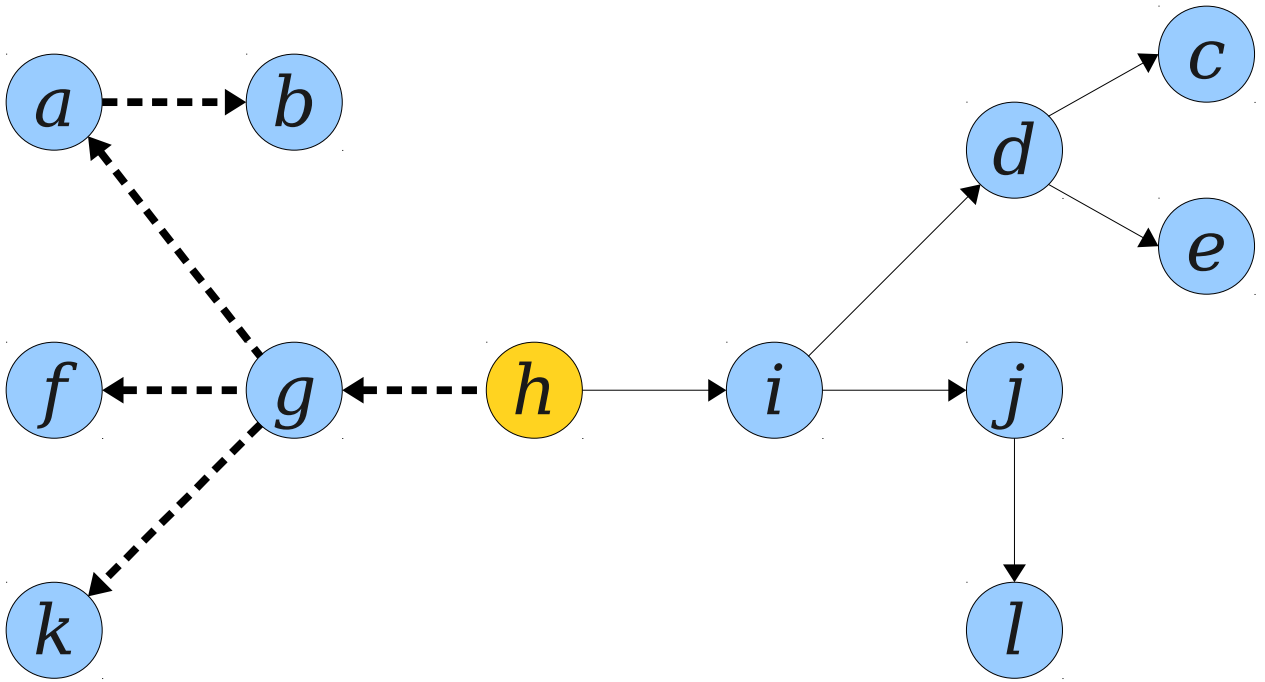
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Rerooting a Tour

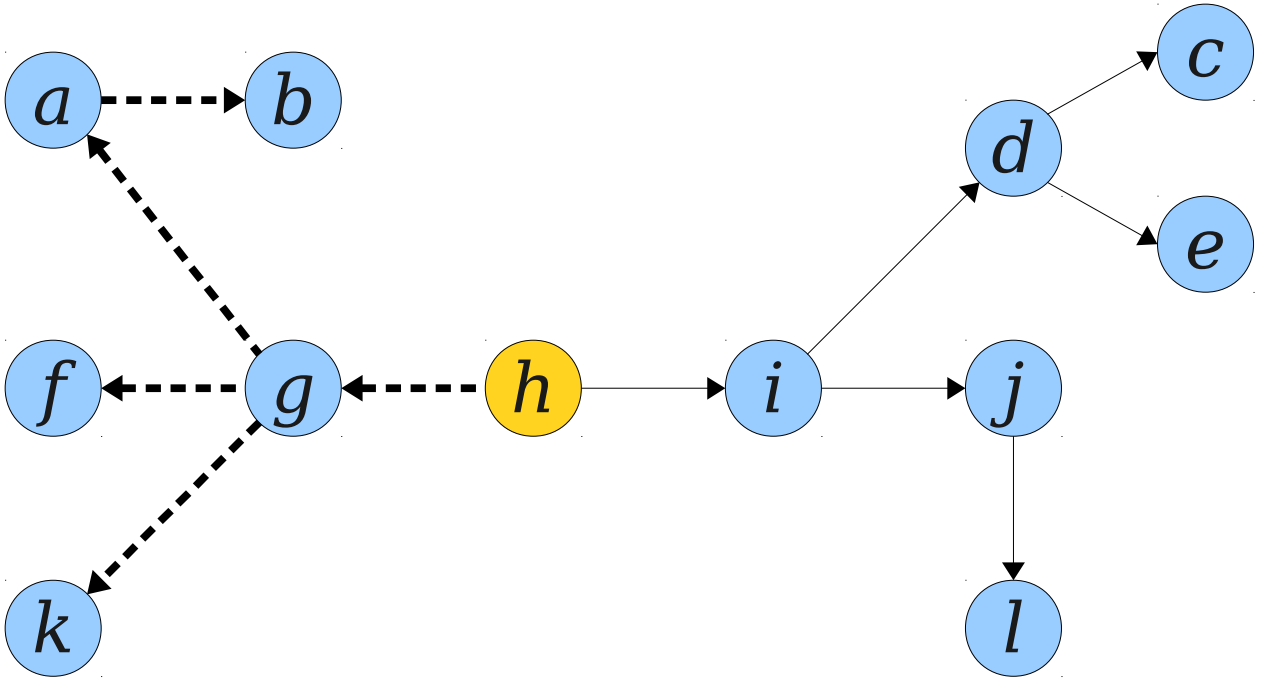
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Rerooting a Tour

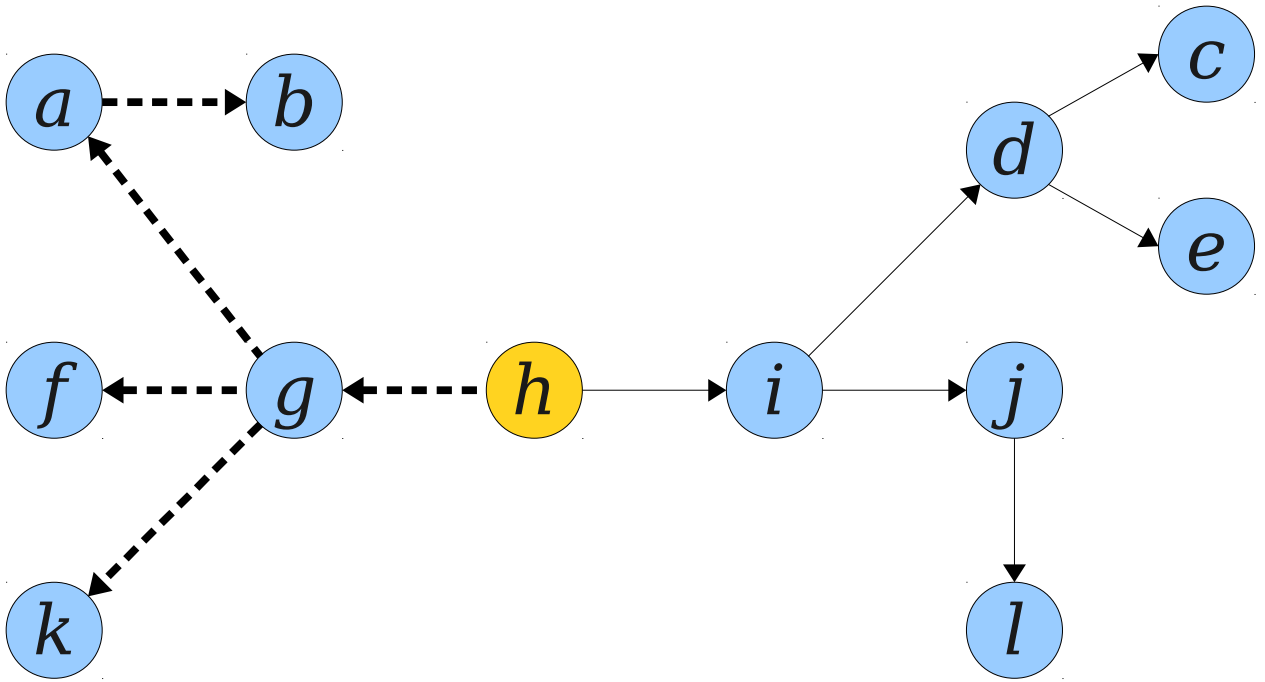
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



a b a g h i d c d e d i j l j i h g f g k g a

Rerooting a Tour

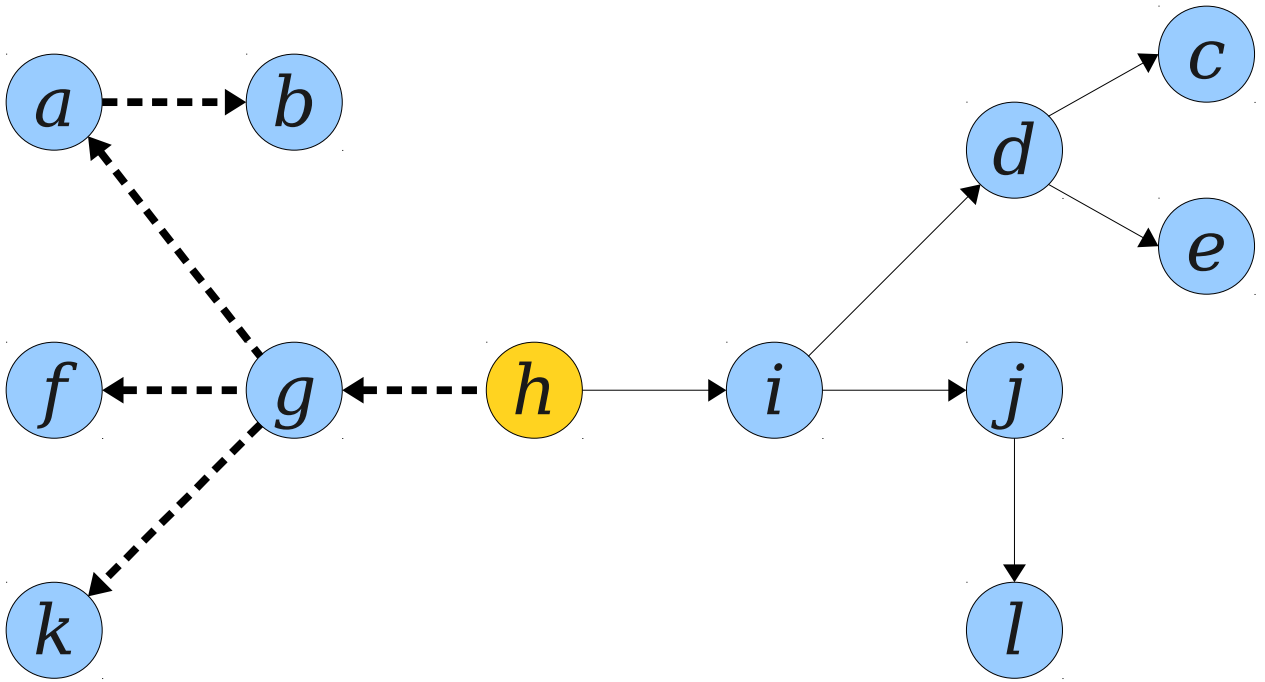
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



h i d c d e d i j l j i h ***g f g k g a*** ***a b a g***

Rerooting a Tour

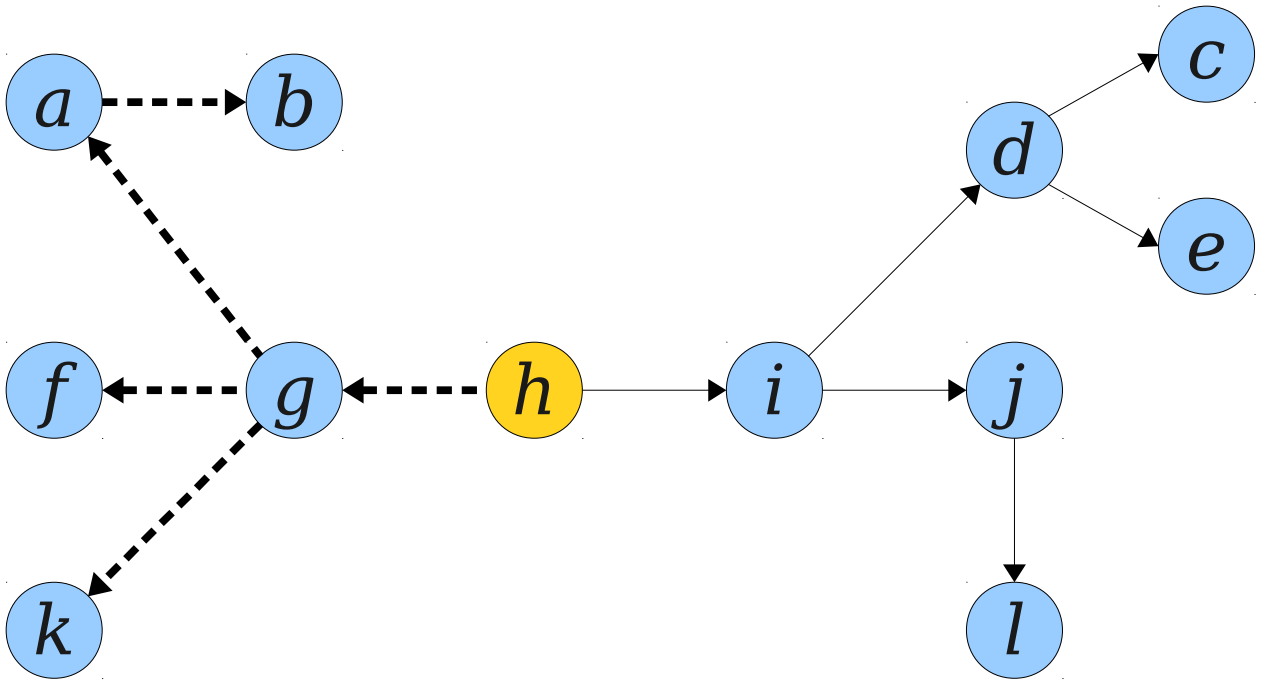
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



h i d c d e d i j l j i h ***g f g k g a a b a g***

Rerooting a Tour

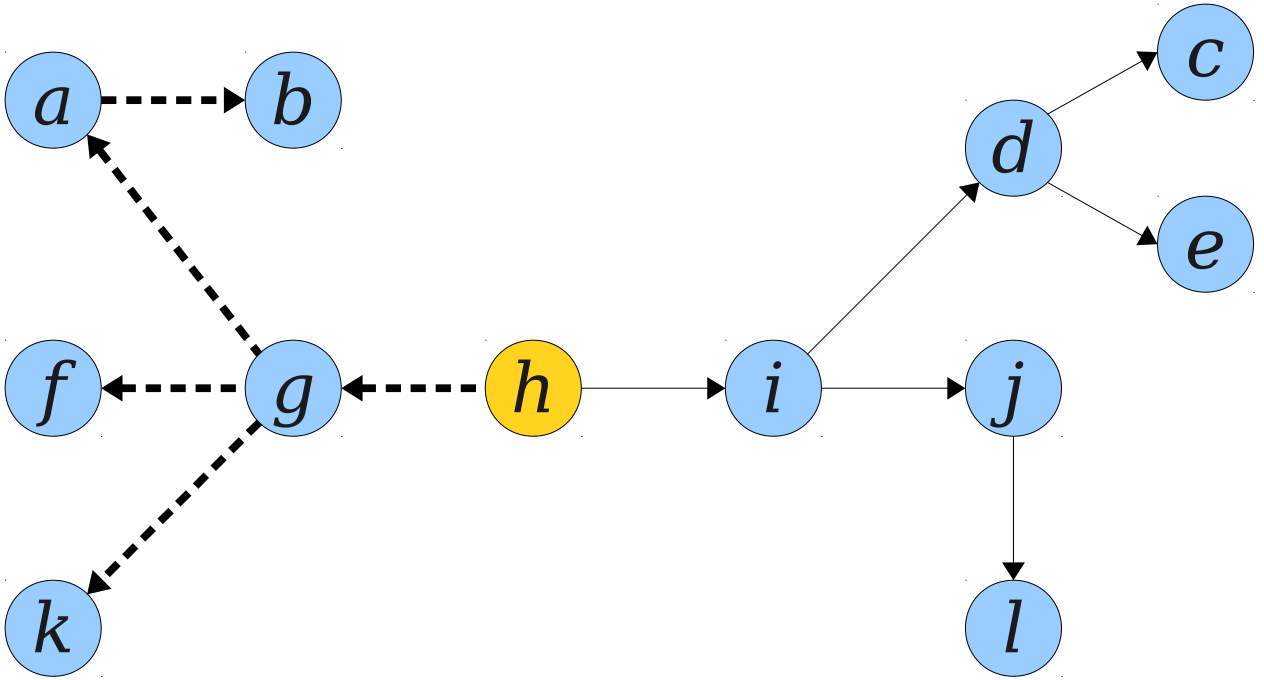
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



h i d c d e d i j l j i h ***g f g k g a b a g***

Rerooting a Tour

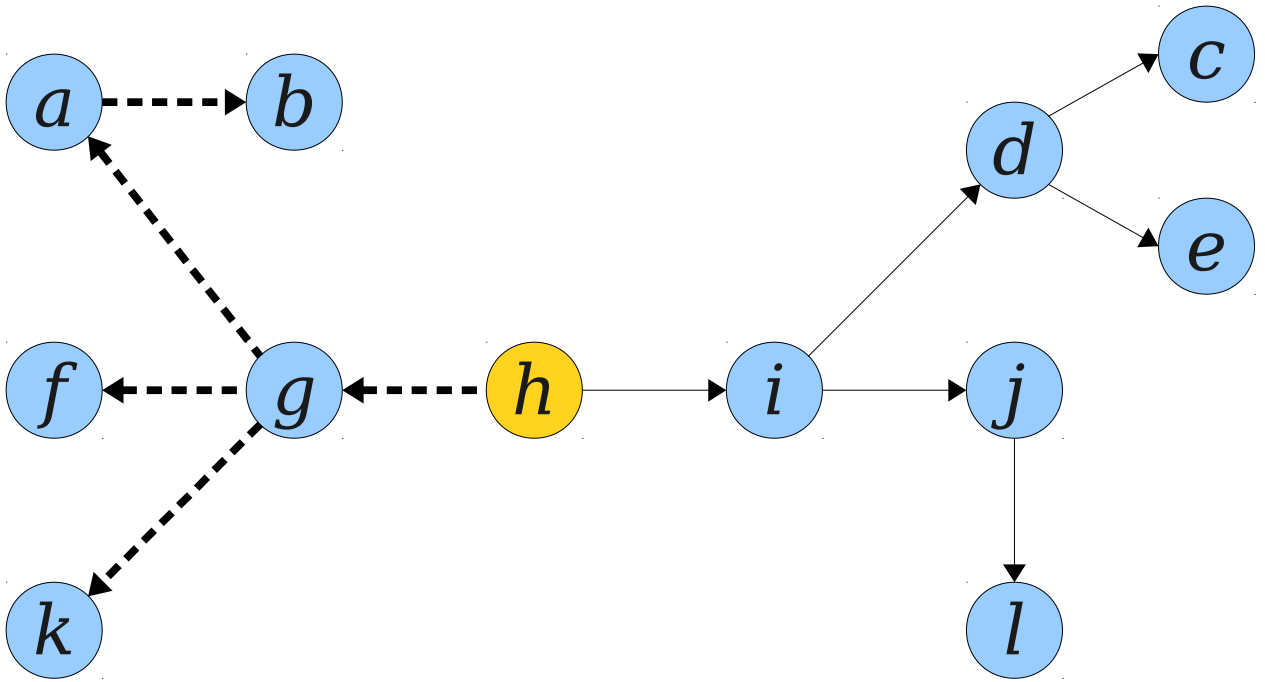
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



h i d c d e d i j l j i h ***g f g k g a b a g*** ***h***

Rerooting a Tour

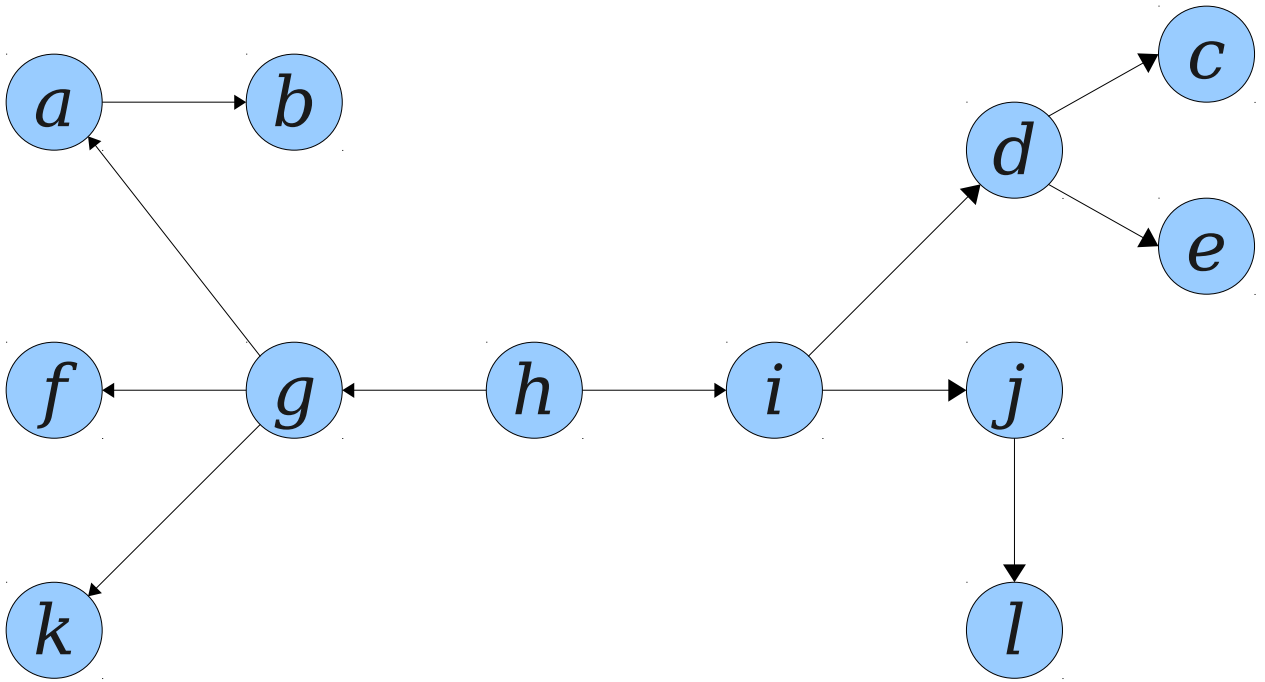
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



h i d c d e d i j l j i h g f g k g a b a g h

Rerooting a Tour

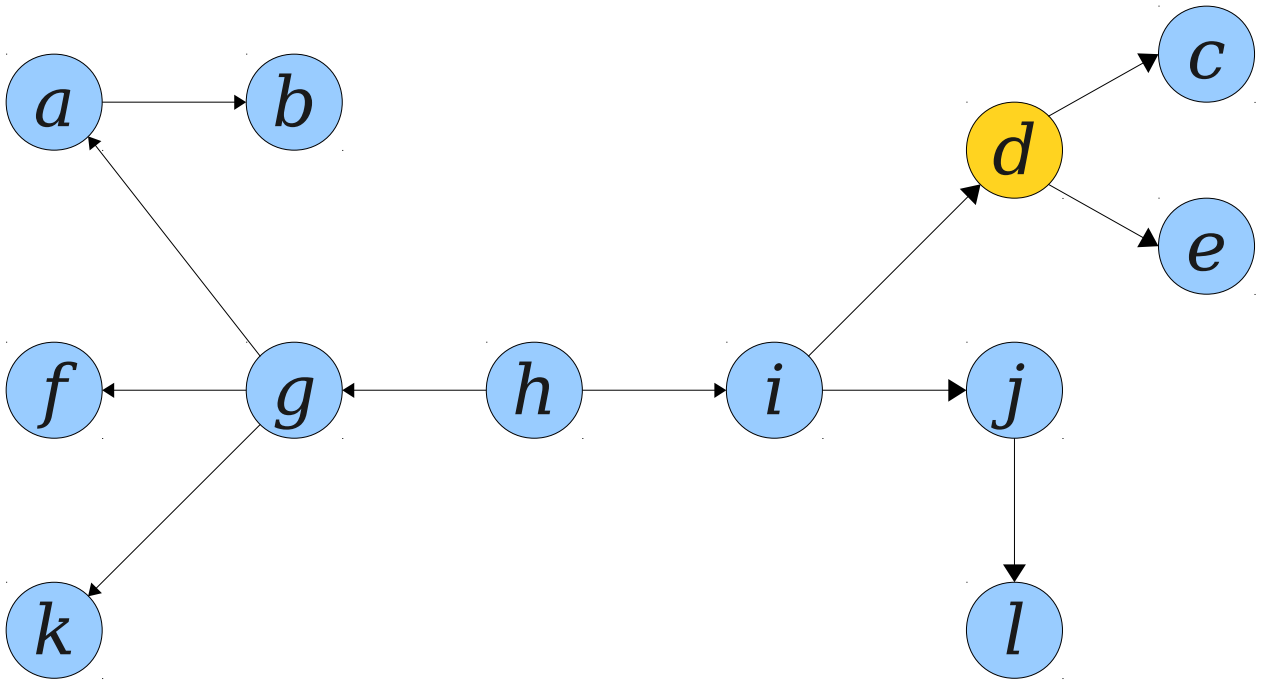
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



h i d c d e d i j l j i h g f g k g a b a g h

Rerooting a Tour

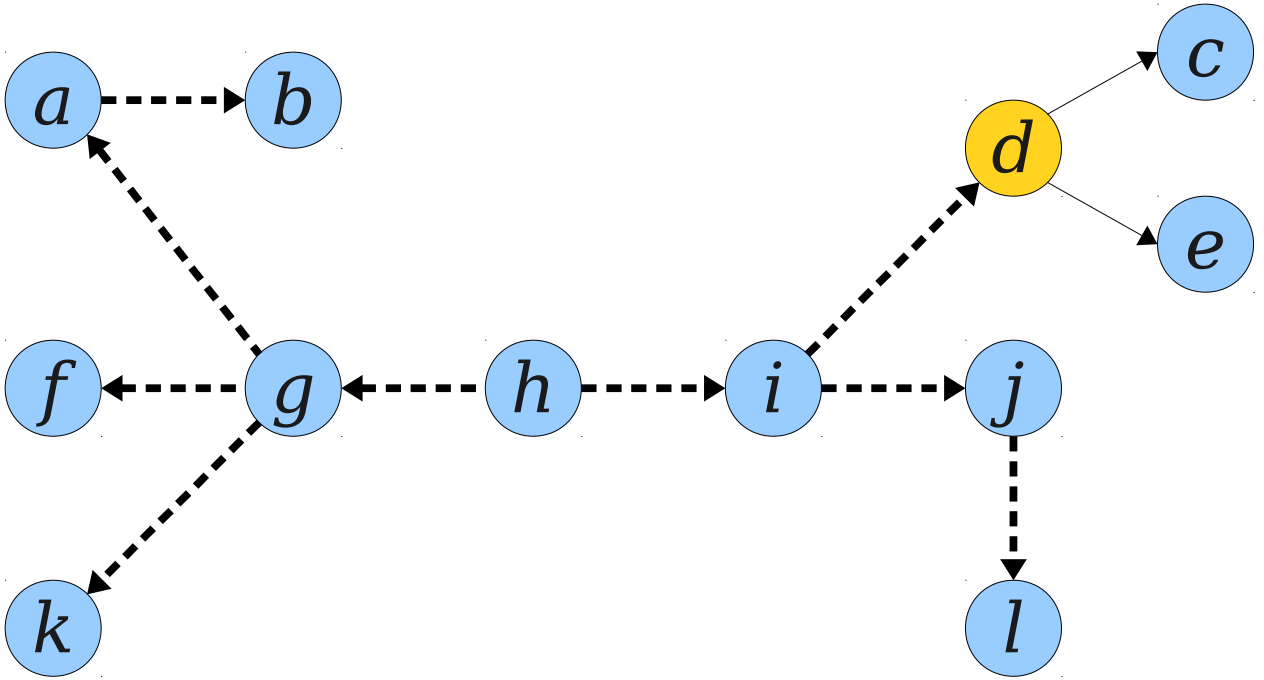
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



h i d c d e d i j l j i h g f g k g a b a g h

Rerooting a Tour

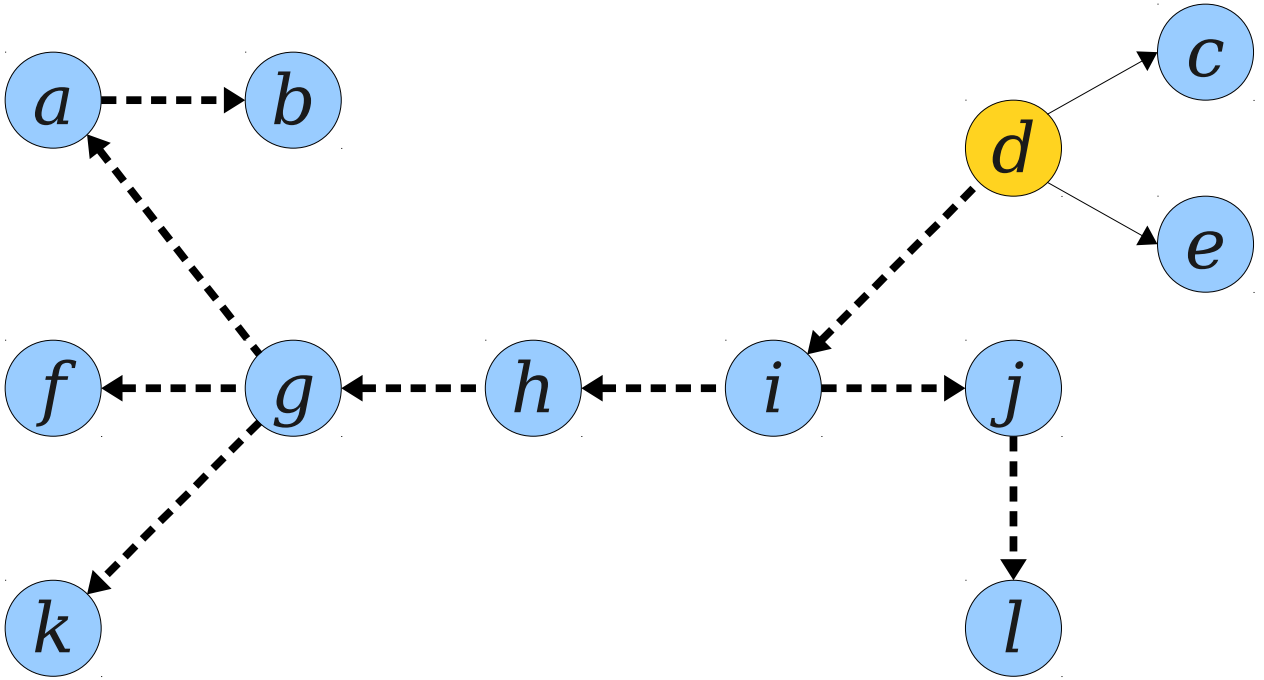
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



h i d c d e d i j l j i h g f g k g a b a g h

Rerooting a Tour

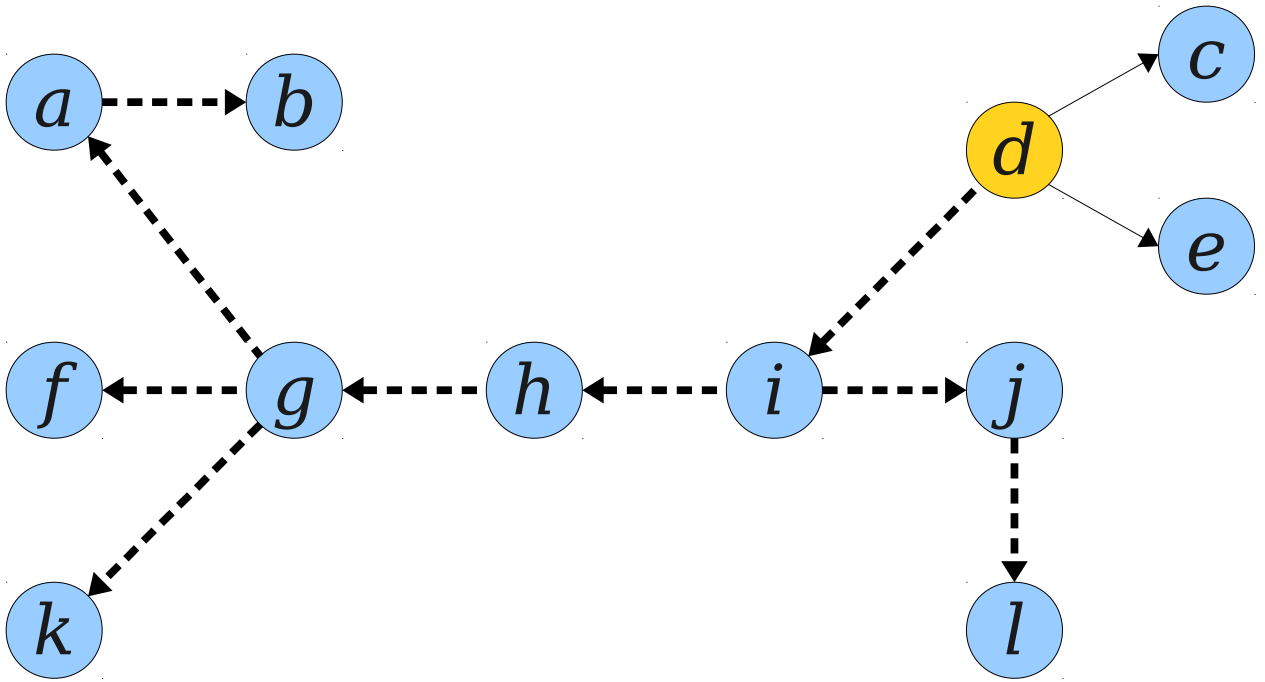
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



h i d c d e d i j l j i h g f g k g a b a g h

Rerooting a Tour

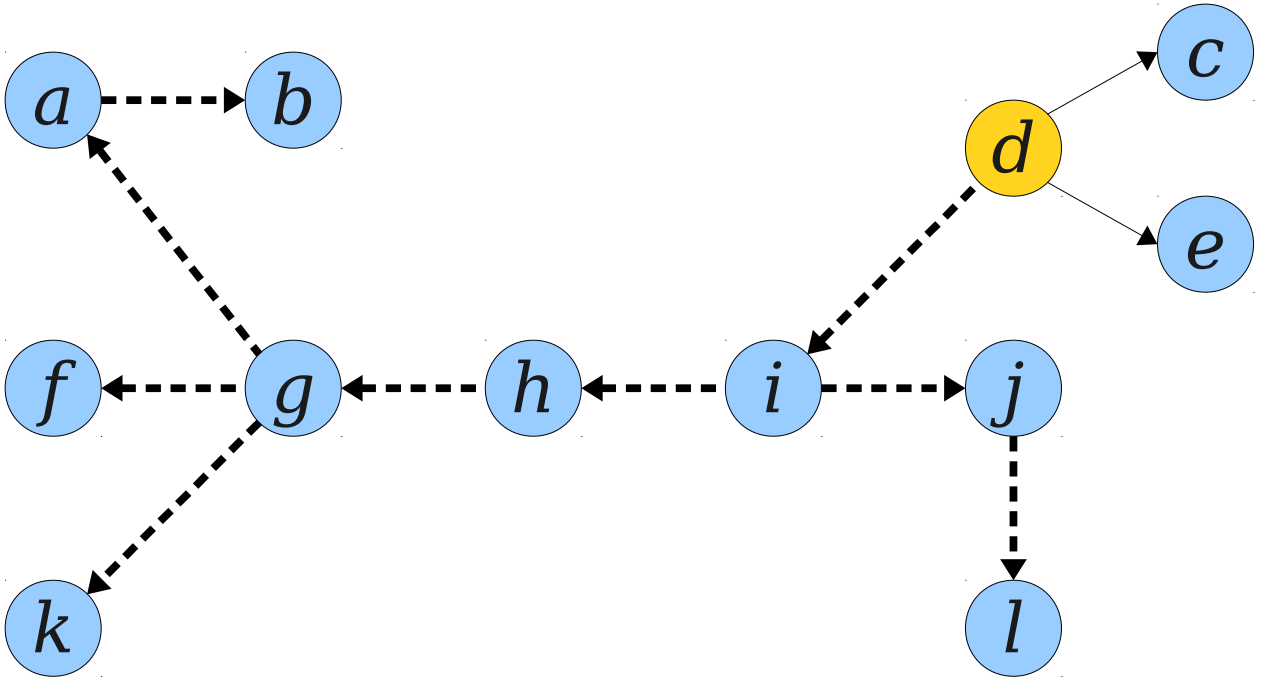
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



h i d c d e d i j l j i h g f g k g a b a g h

Rerooting a Tour

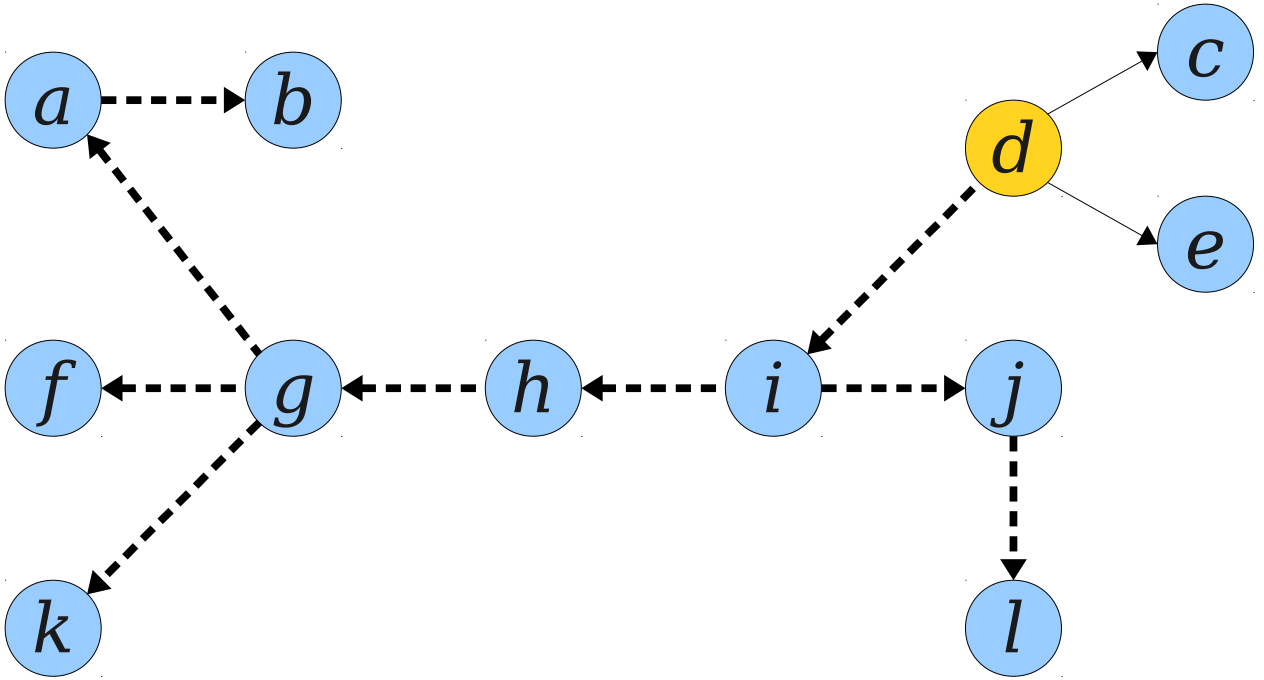
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



h i d c d e d i j l j i h g f g k g a b a g h

Rerooting a Tour

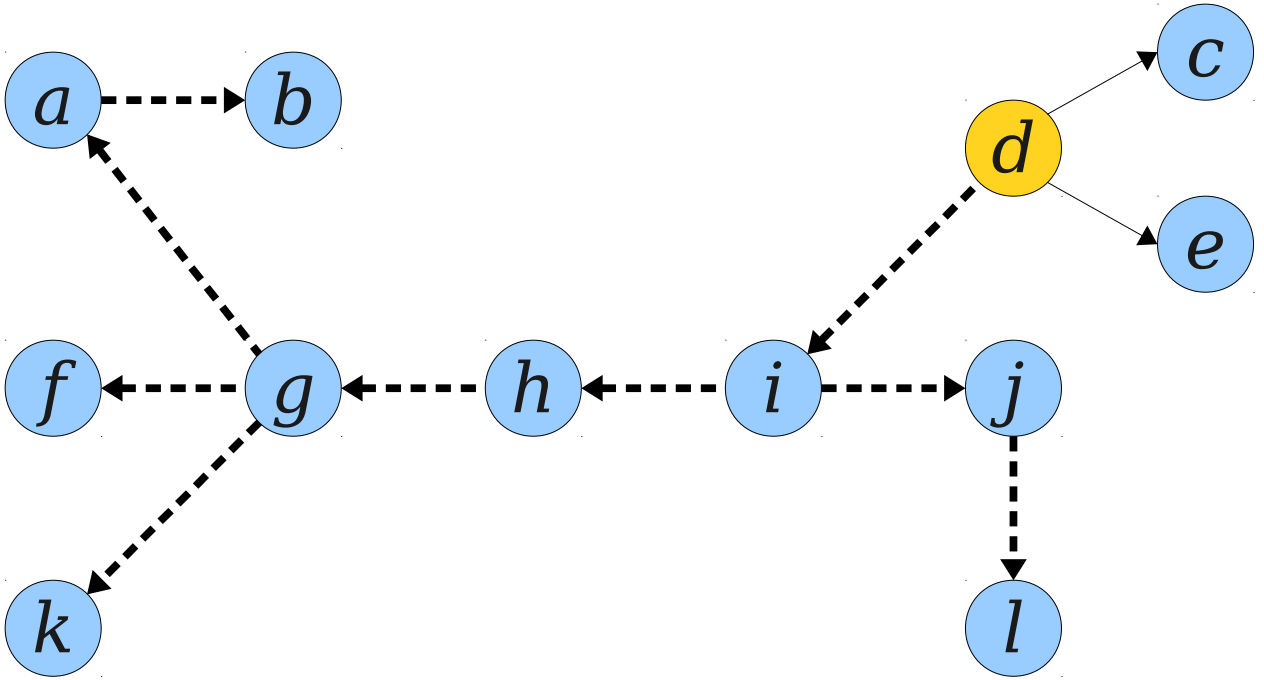
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



d c d e d ***i j l j i h g f g k g a b a g h*** ***h i***

Rerooting a Tour

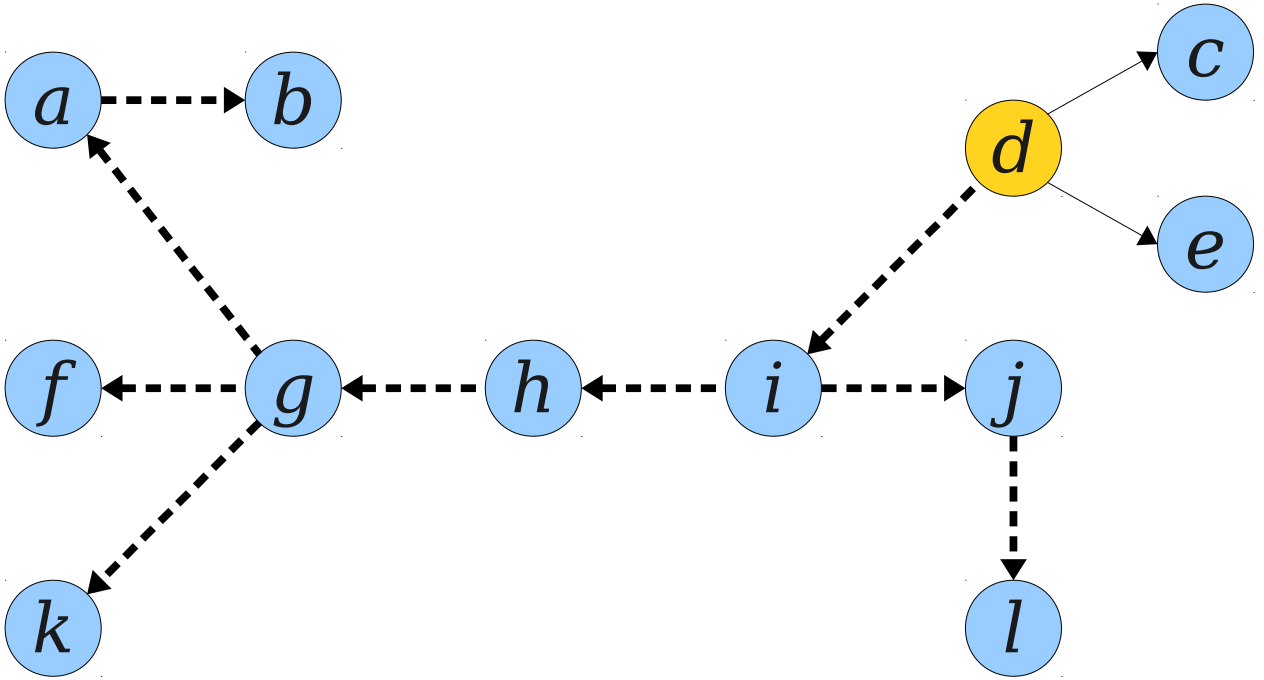
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



d c d e d i j l j i h g f g k g a b a g h h i

Rerooting a Tour

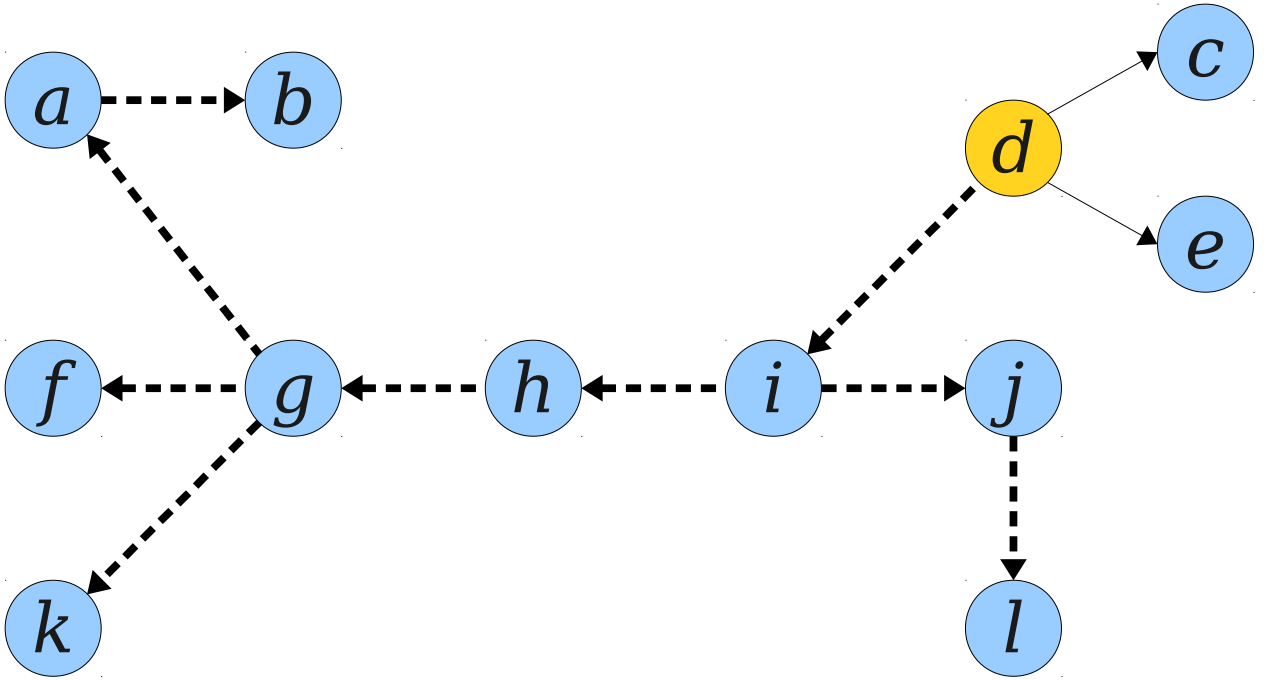
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



d c d e d i j l j i h g f g k g a b a g h i

Rerooting a Tour

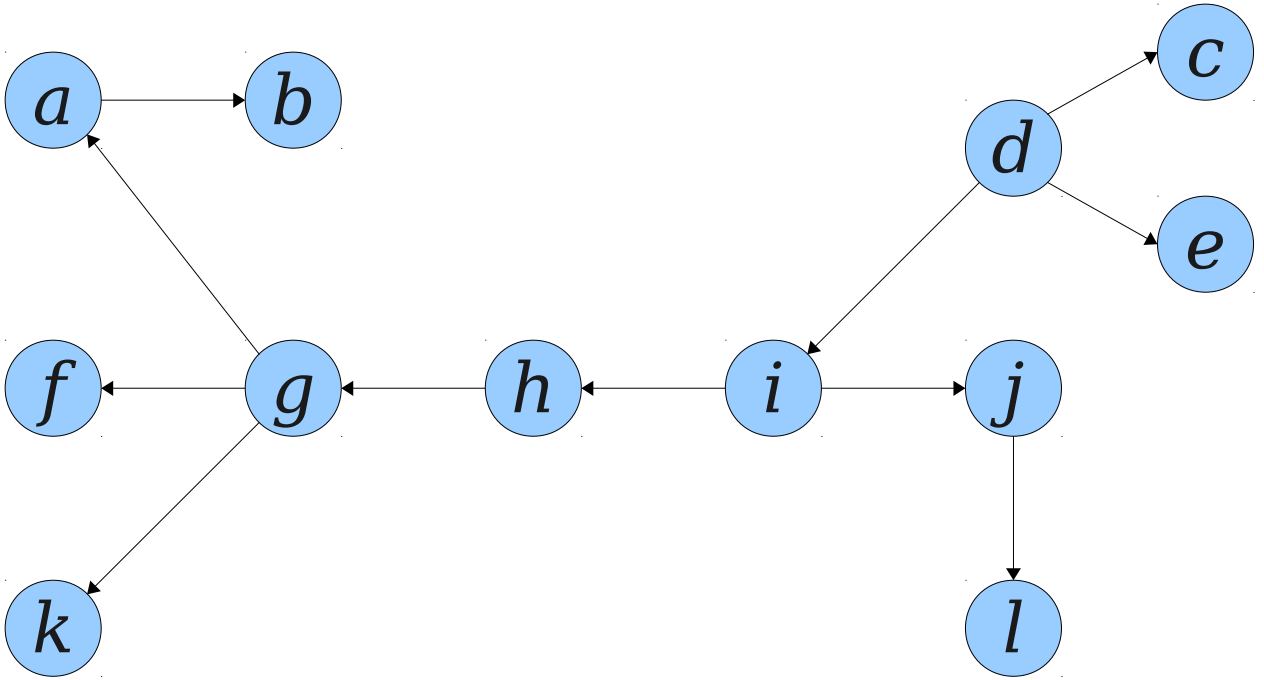
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



d c d e d i j l j i h g f g k g a b a g h i d

Rerooting a Tour

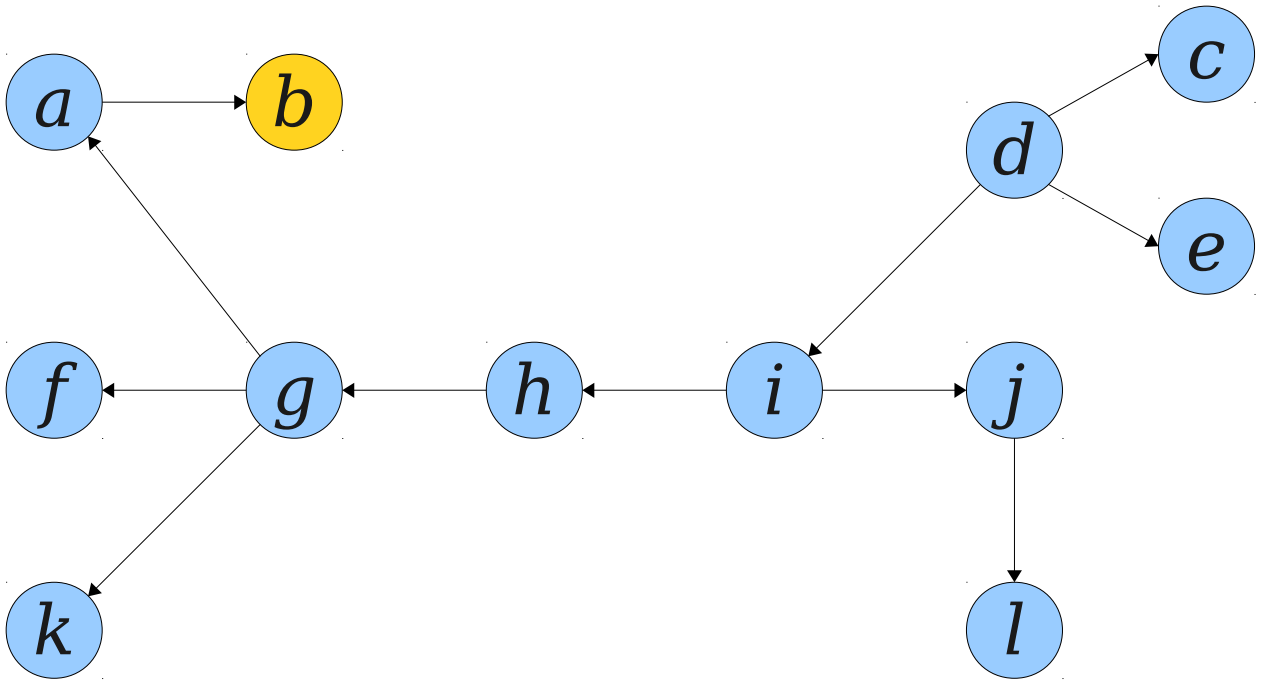
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



d c d e d i j l j i h g f g k g a b a g h i d

Rerooting a Tour

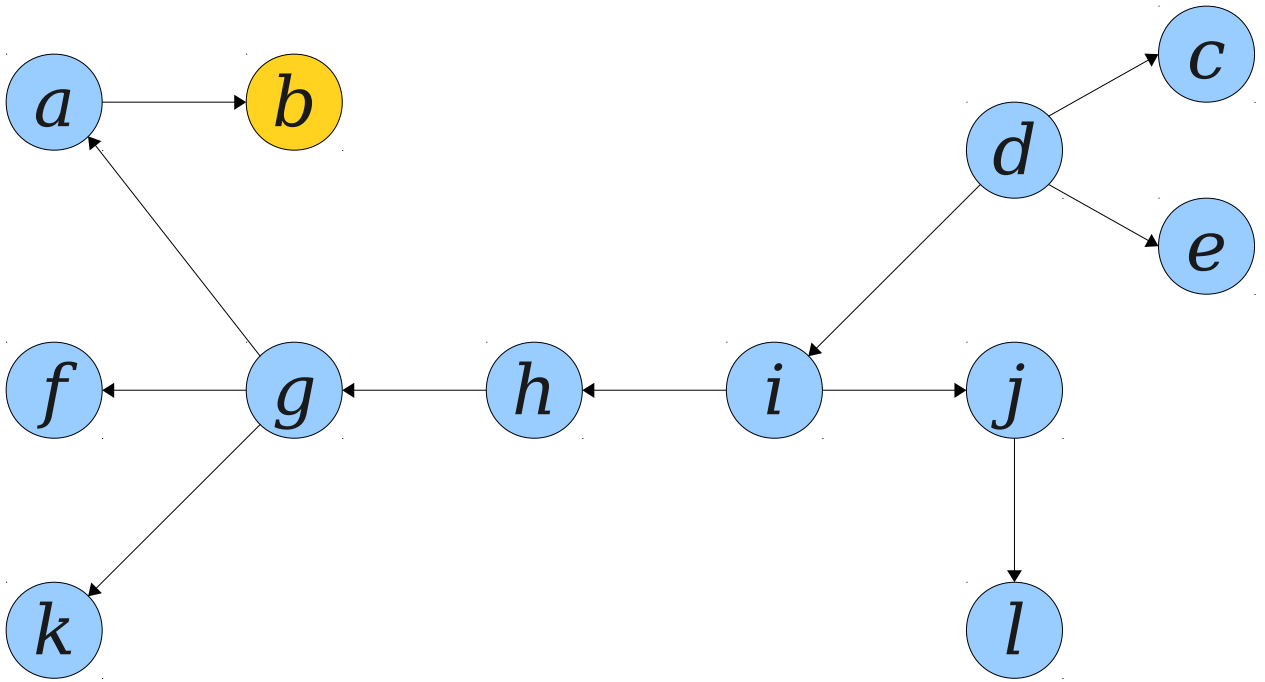
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



d c d e d i j l j i h g f g k g a b a g h i d

Rerooting a Tour

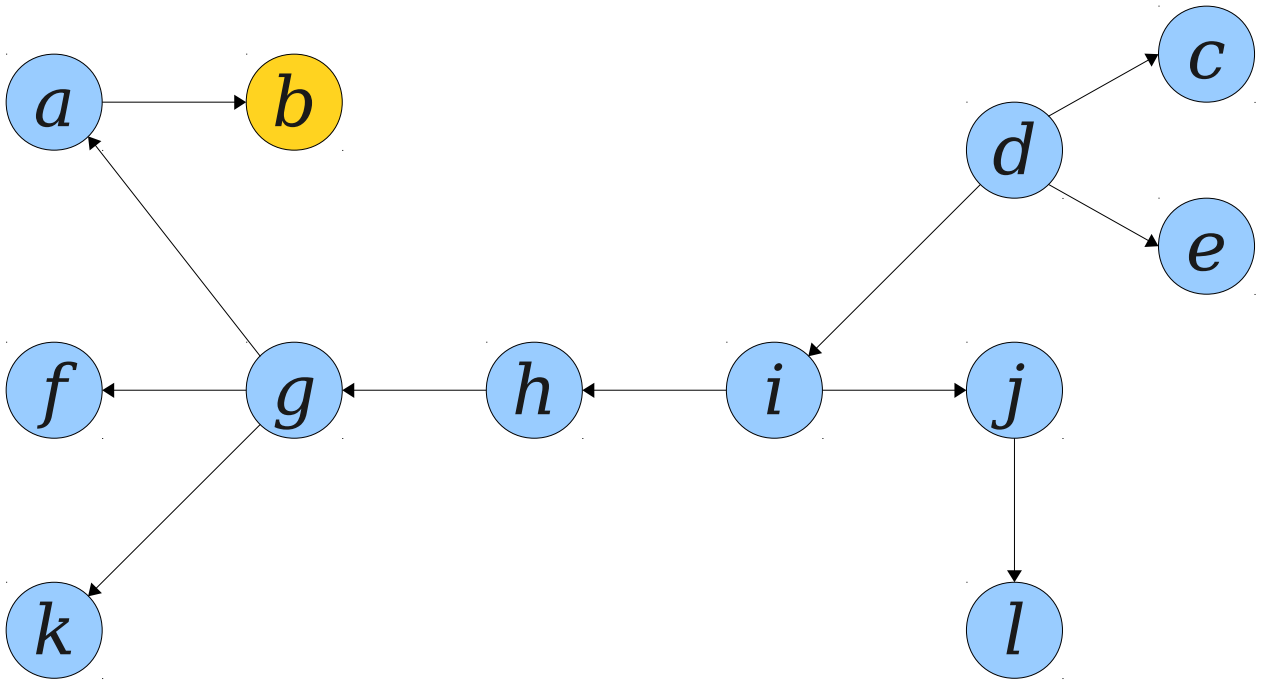
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



d c d e d i j l j i h g f g k g a b a g h i d

Rerooting a Tour

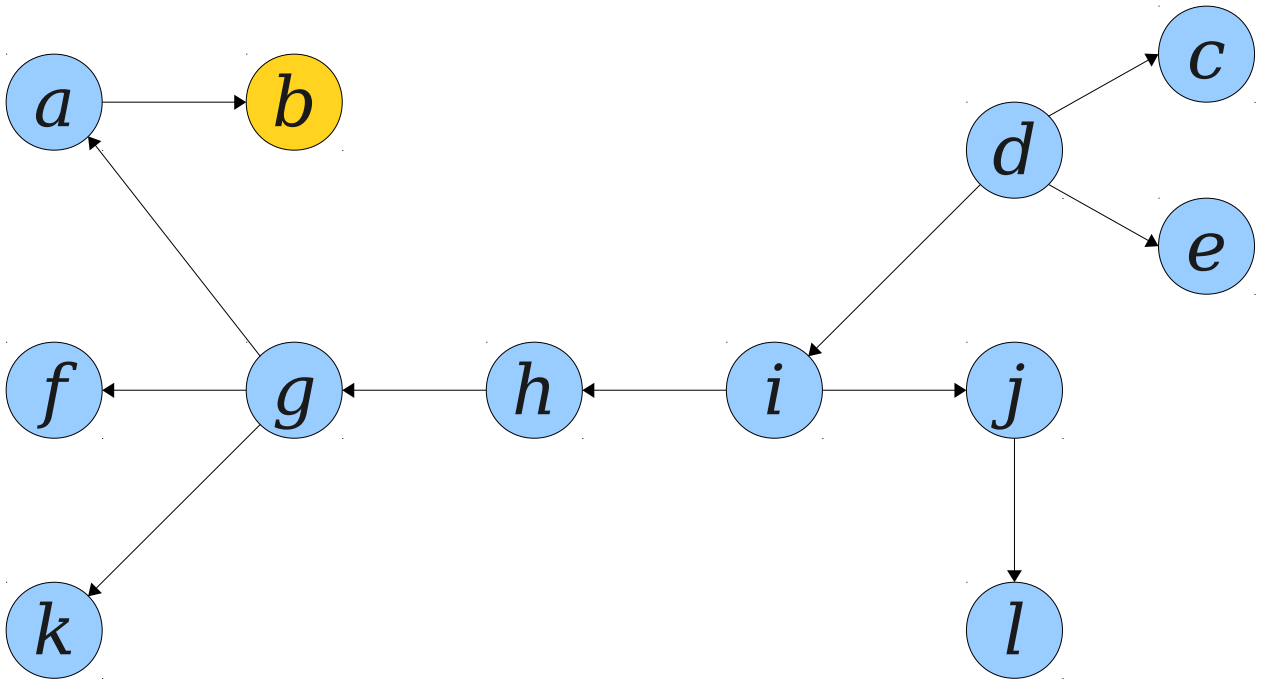
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



d c d e d i j l j i h g f g k g a b a g h i d

Rerooting a Tour

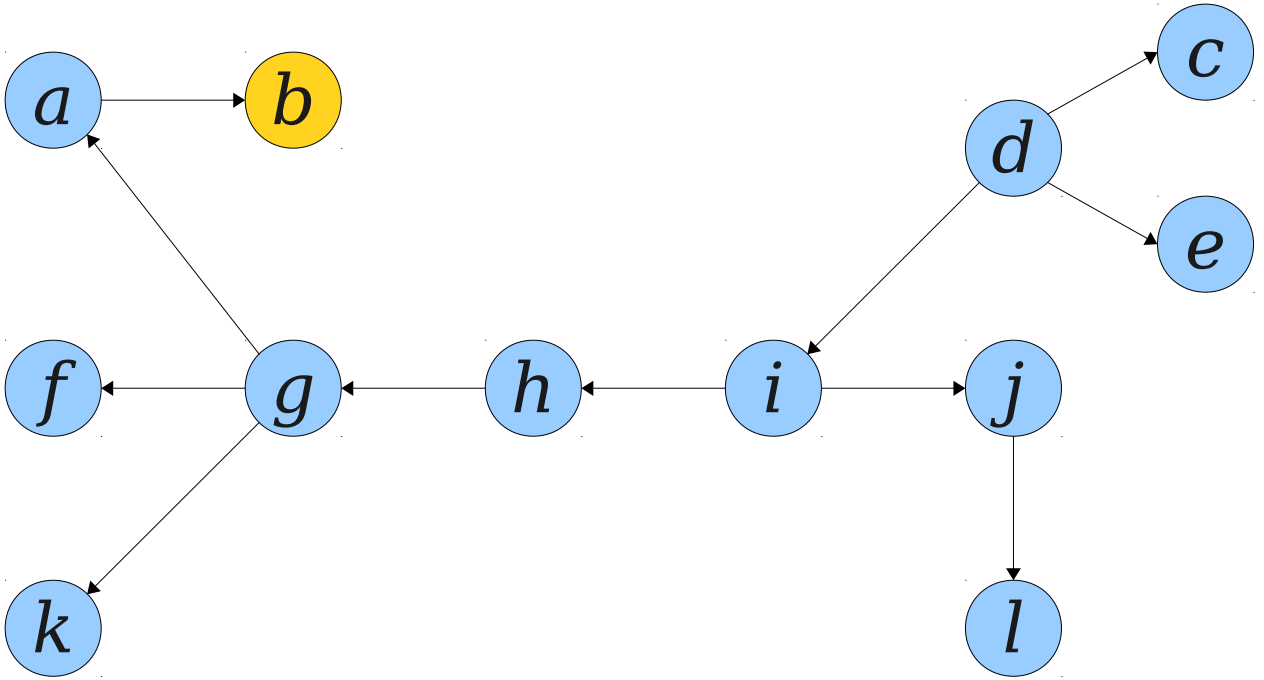
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



b a g h i d d c d e d i j l j i h g f g k g a

Rerooting a Tour

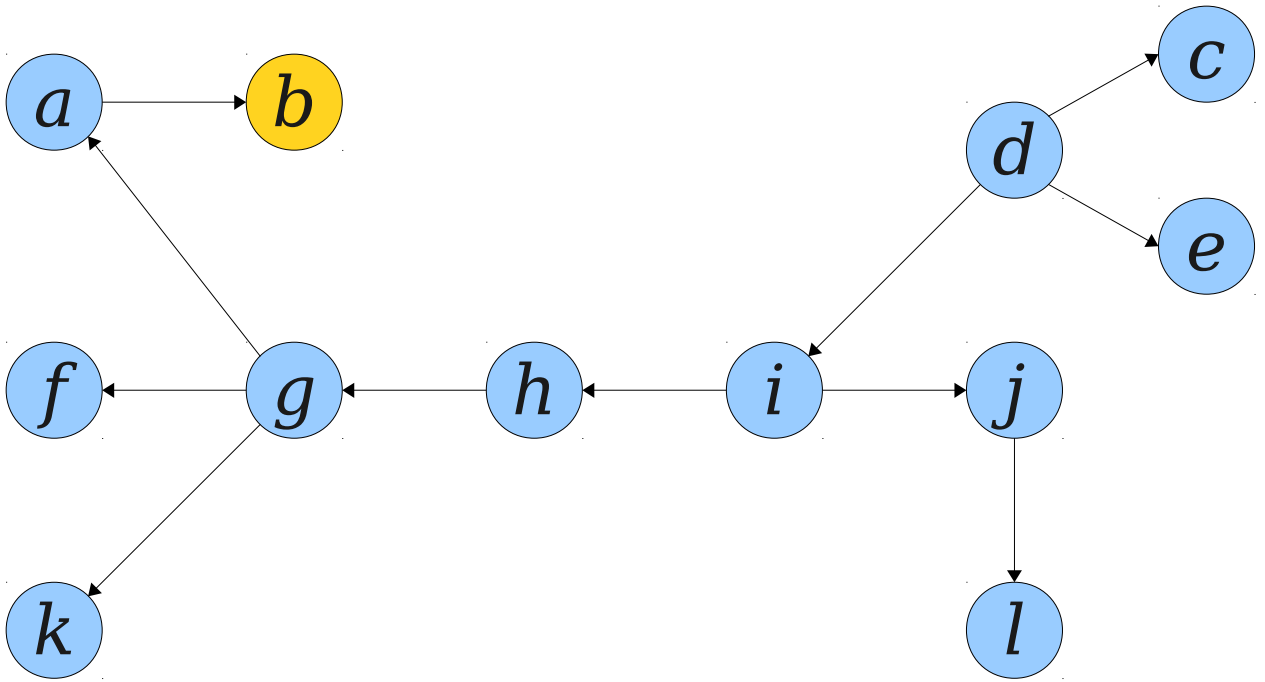
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



b a g h i d d c d e d i j l j i h g f g k g a

Rerooting a Tour

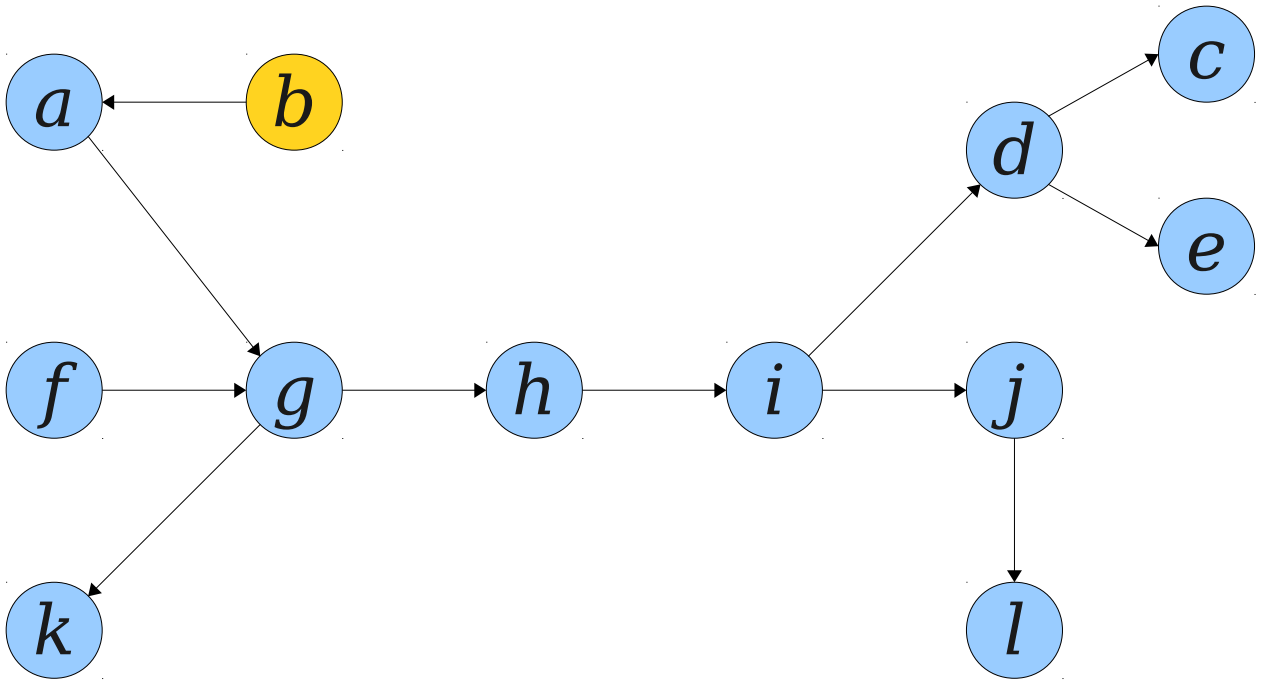
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



b a g h i d c d e d i j l j i h g f g k g a

Rerooting a Tour

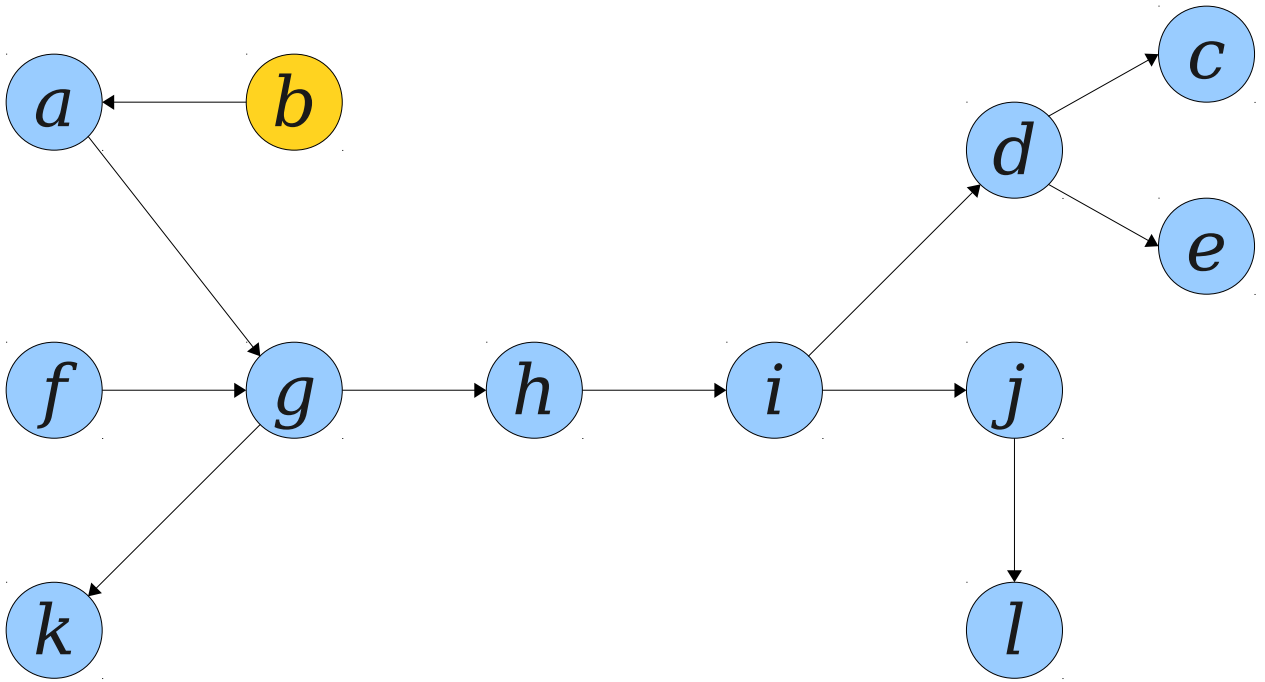
- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



b a g h i d c d e d i j l j i h g f g k g a b

Rerooting a Tour

- The subtrees defined by ranges in Euler tours on trees depend on the root.
- In some cases, we will need to change the root of the tree.



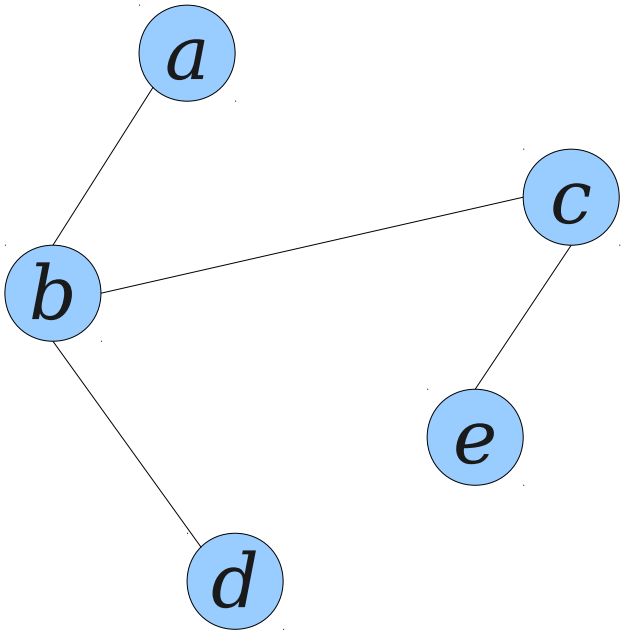
b a g h i d c d e d i j l j i h g f g k g a b

Rerooting a Tour

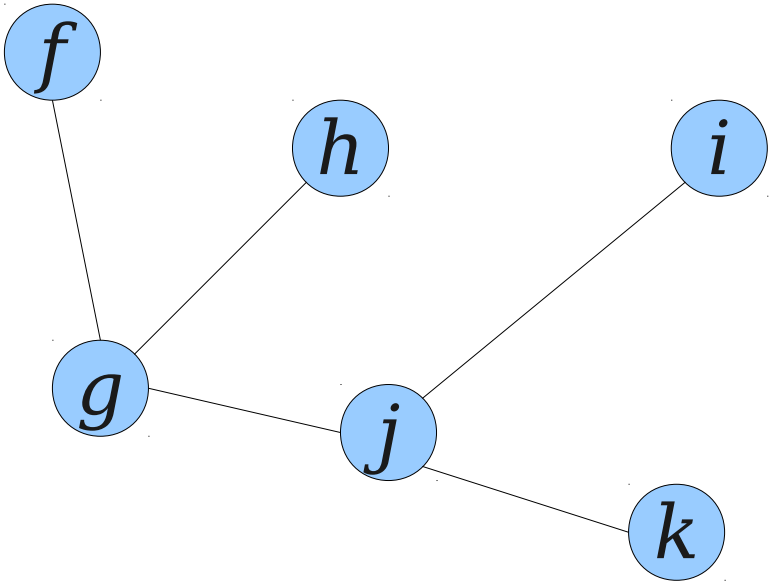
- **Algorithm:**
  - Split the tour into three parts: S_1, R, and S_2, where R consists of the nodes between the first and last occurrence of the new root r.
  - Delete the first node in S_1.
  - Concatenate R, S_2, S_1, $\{r\}$.

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:



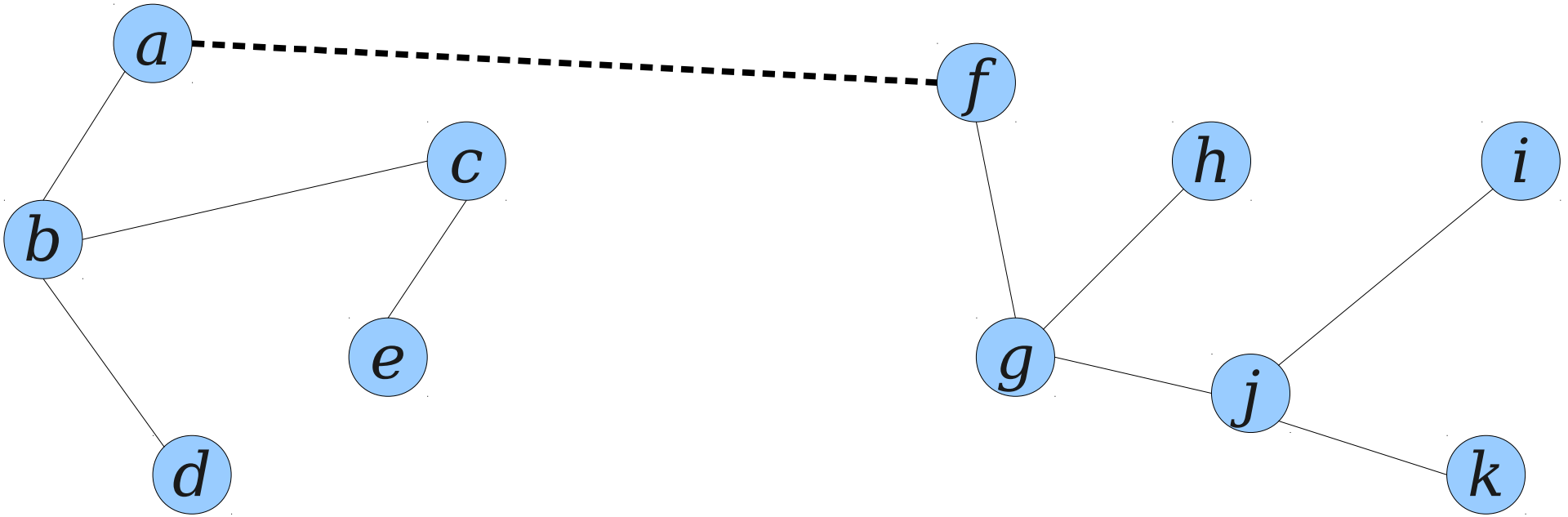
a b d b c e c b a



f g j k j i j g h g f

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

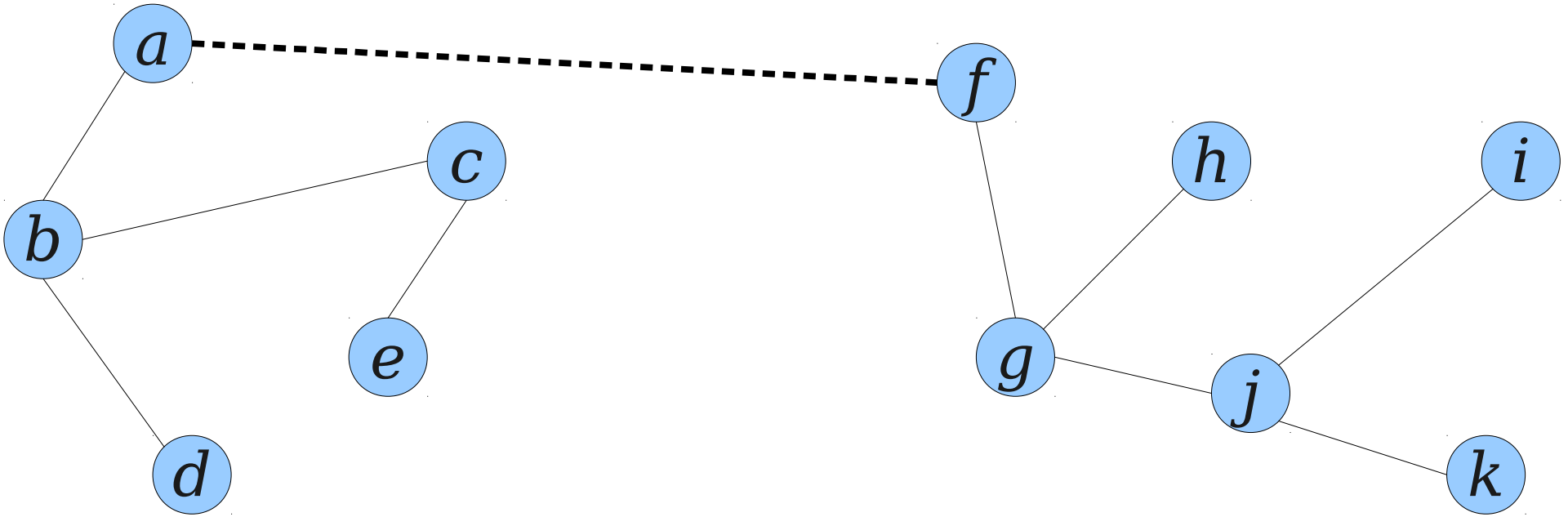


a b d b c e c b a

f g j k j i j g h g f

Euler Tours and Dynamic Trees

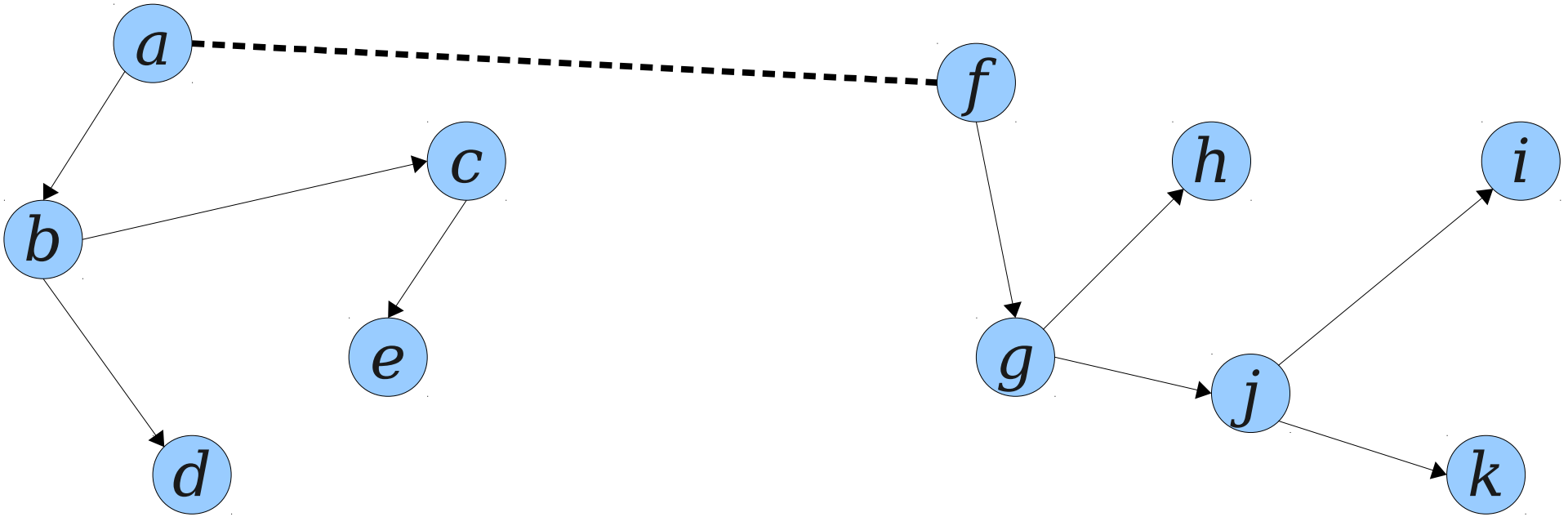
- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:



a *b d b c e c b a* ***f*** *g j k j i j g h g f*

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

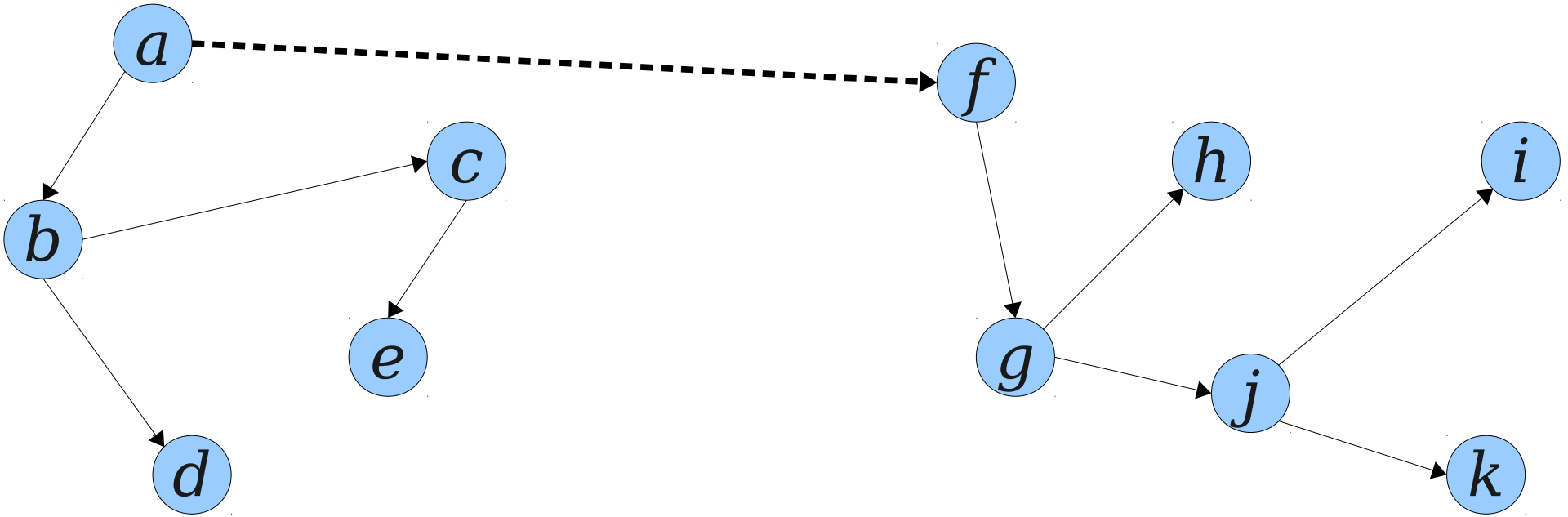


a *b d b c e c b a*

f *g j k j i j g h g f*

Euler Tours and Dynamic Trees

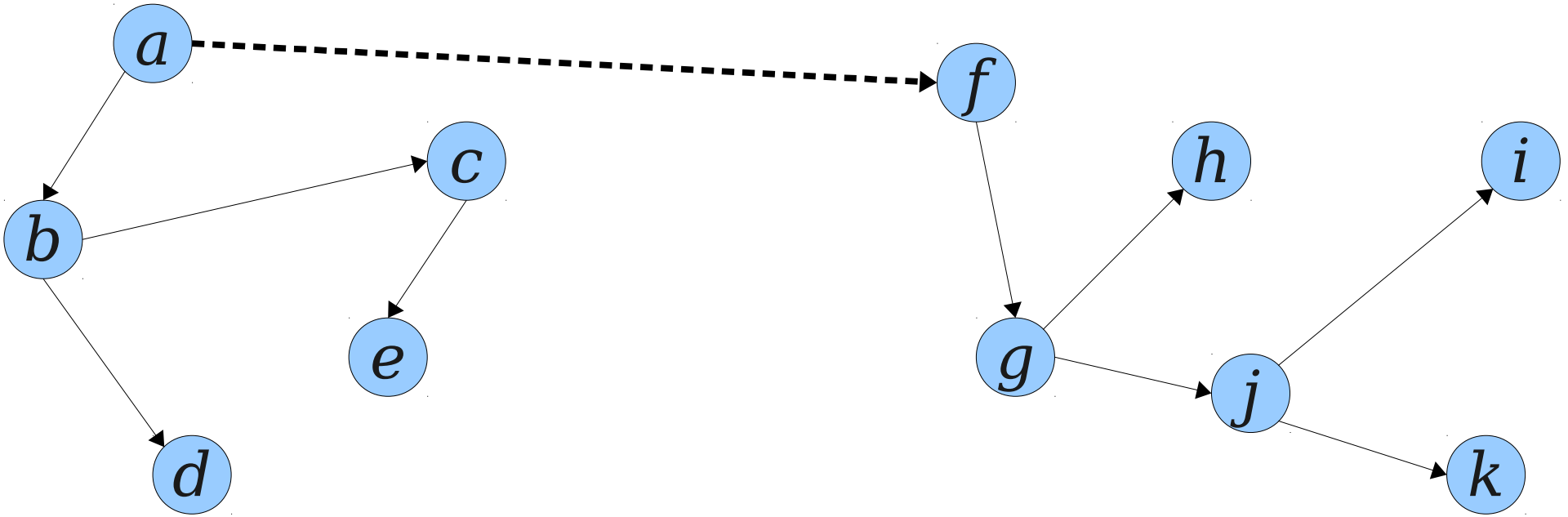
- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:



a *b d b c e c b a* ***f*** *g j k j i j g h g f*

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

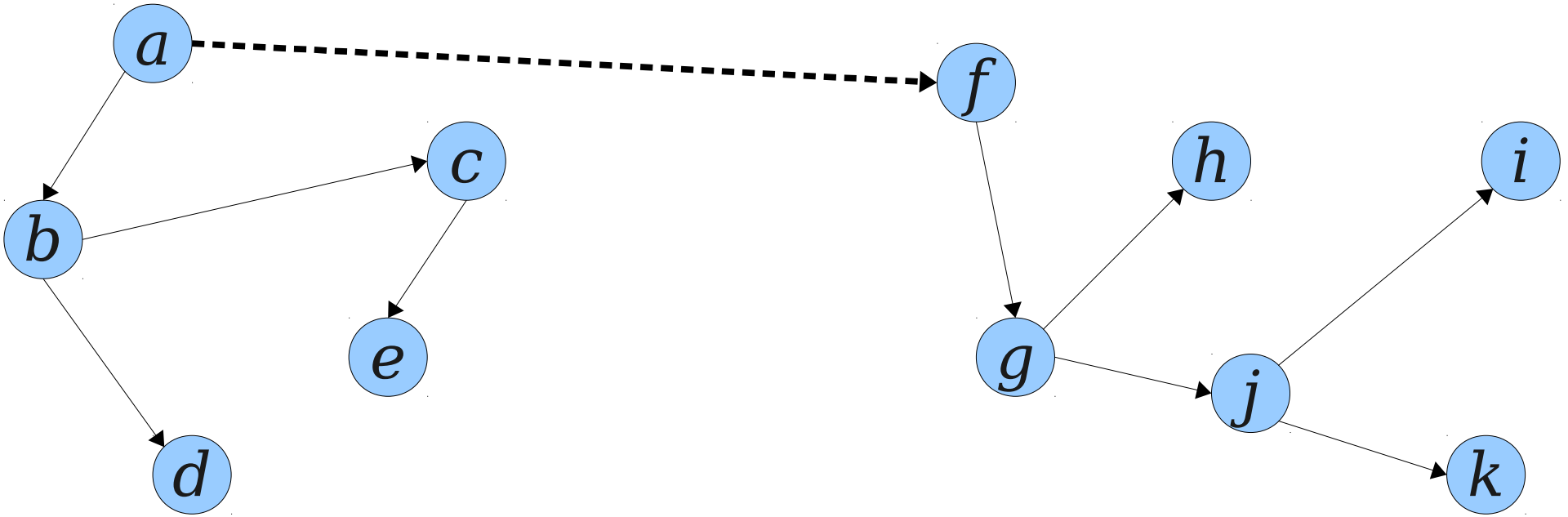


a b d b c e c b a

f g j k j i j g h g f

Euler Tours and Dynamic Trees

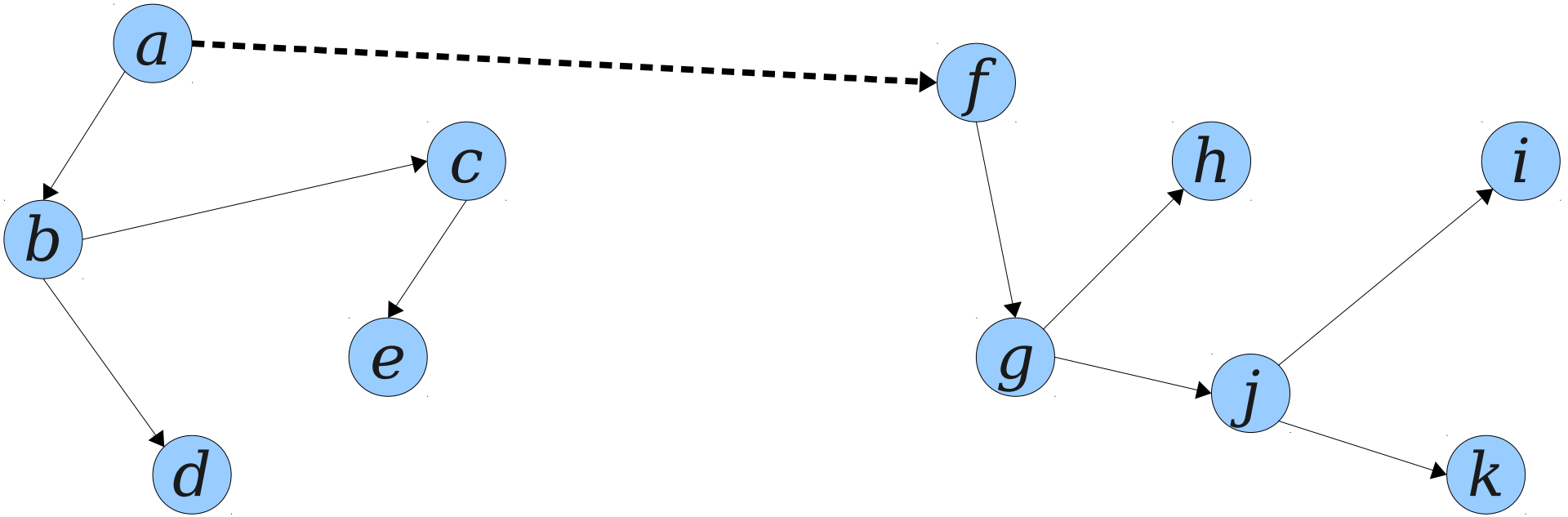
- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:



a b d b c e c b a f g j k j i j g h g f

Euler Tours and Dynamic Trees

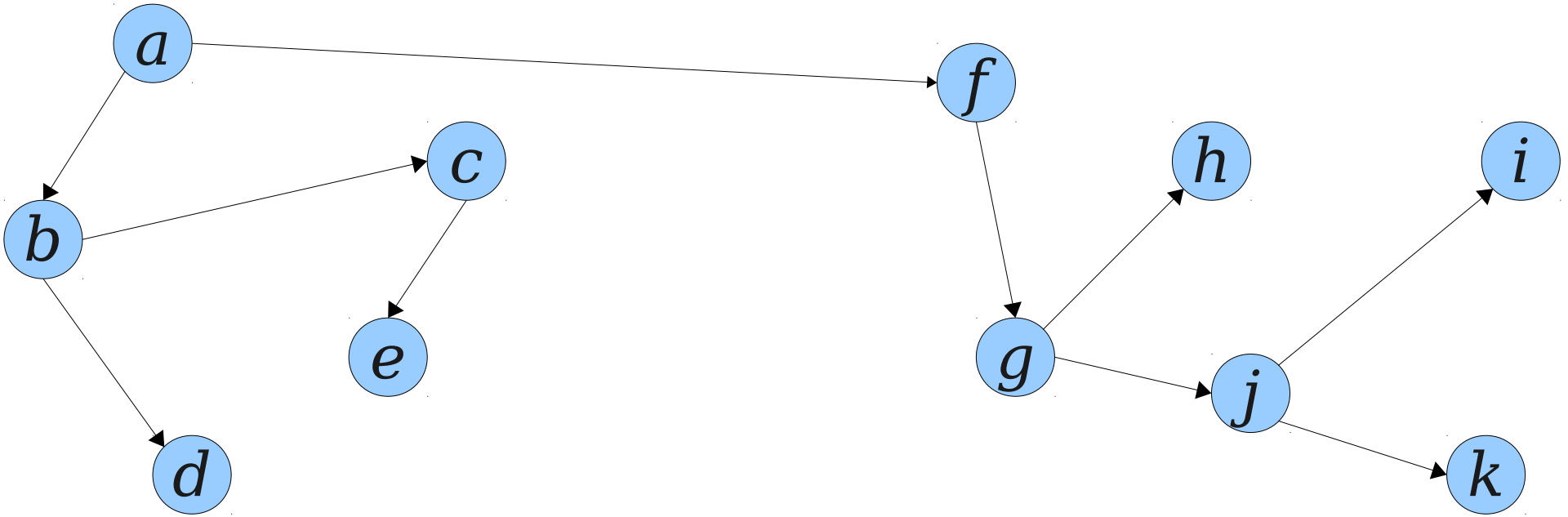
- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:



a b d b c e c b a f g j k j i j g h g f a

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:



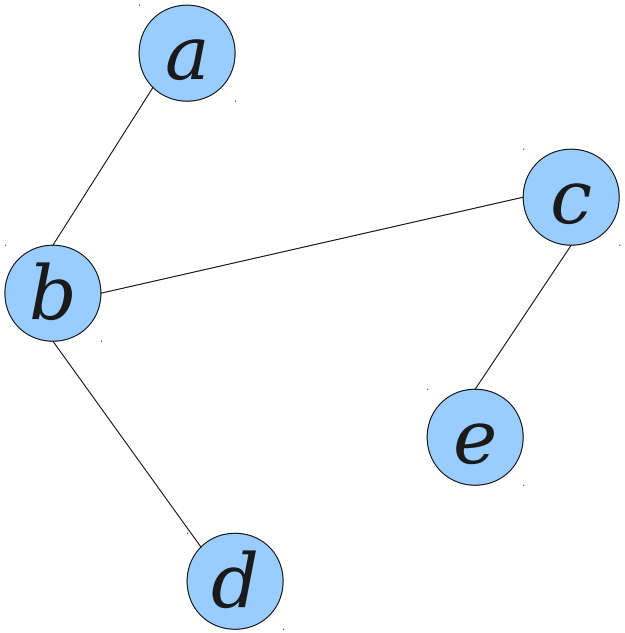
a b d b c e c b a f g j k j i j g h g f a

Euler Tours and Dynamic Trees

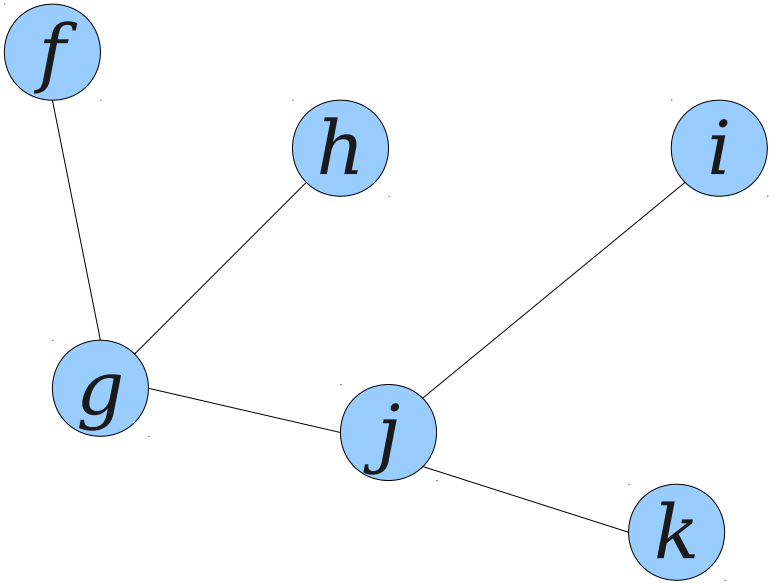
- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:



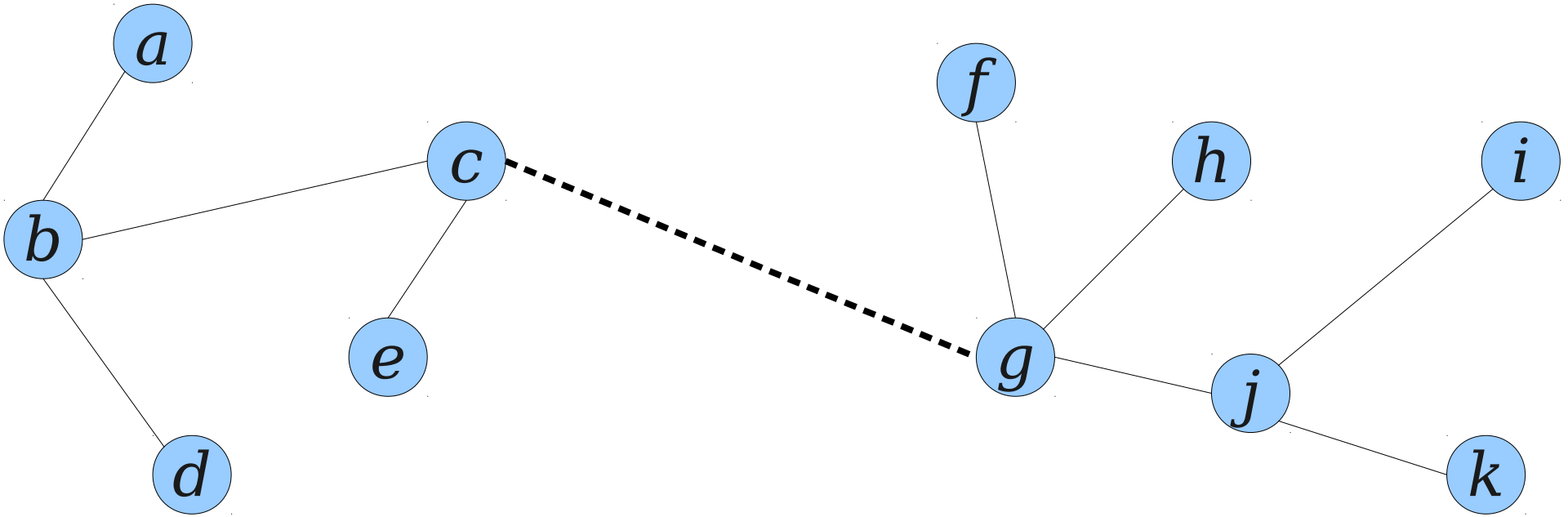
a b d b c e c b a



f g j k j i j g h g f

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

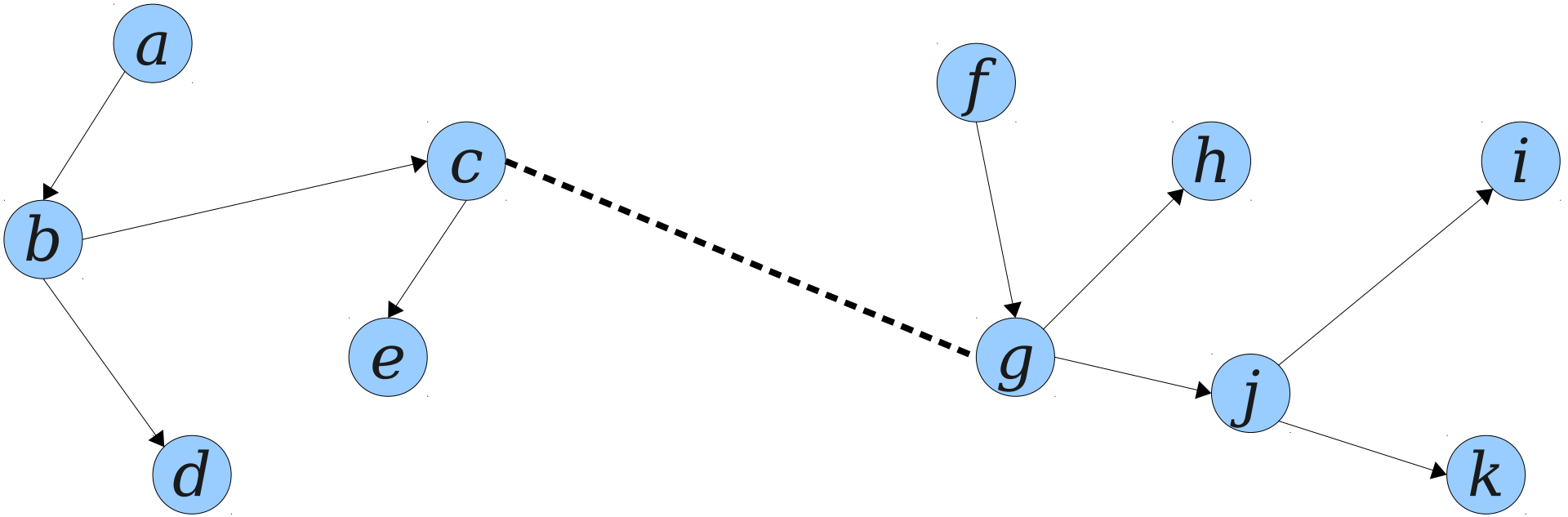


a b d b c e c b a

f g j k j i j g h g f

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

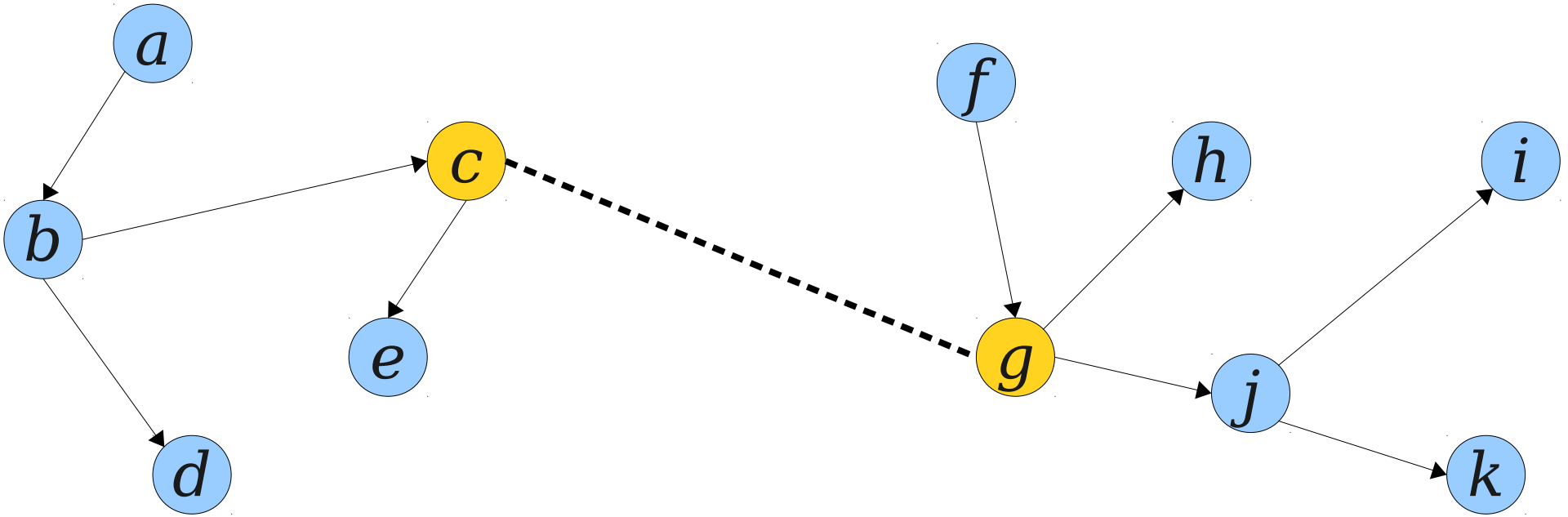


a b d b c e c b a

f g j k j i j g h g f

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

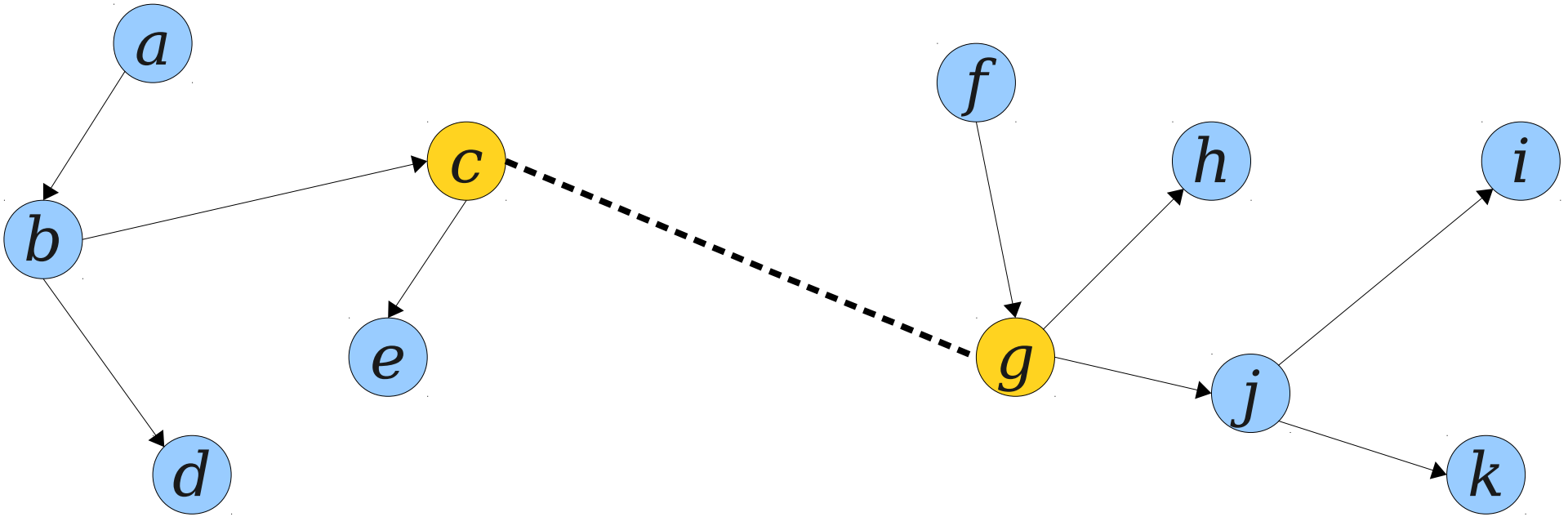


a b d b c e c b a

f g j k j i j g h g f

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

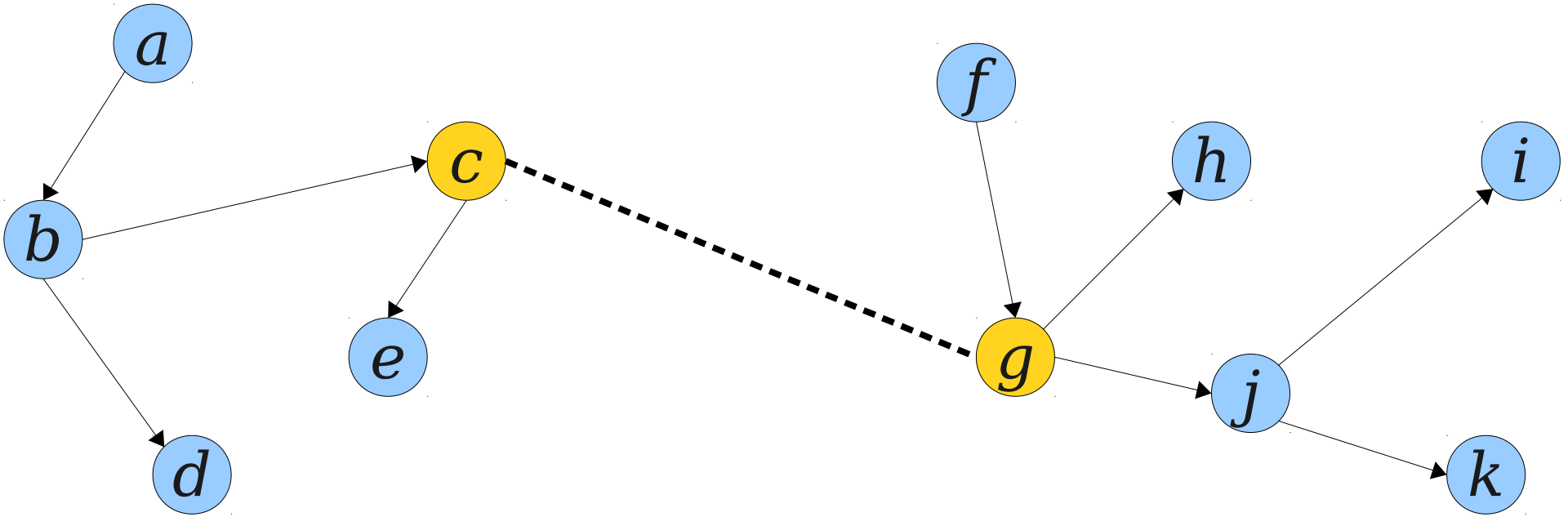


a b d b c e c b a

f g j k j i j g h g f

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

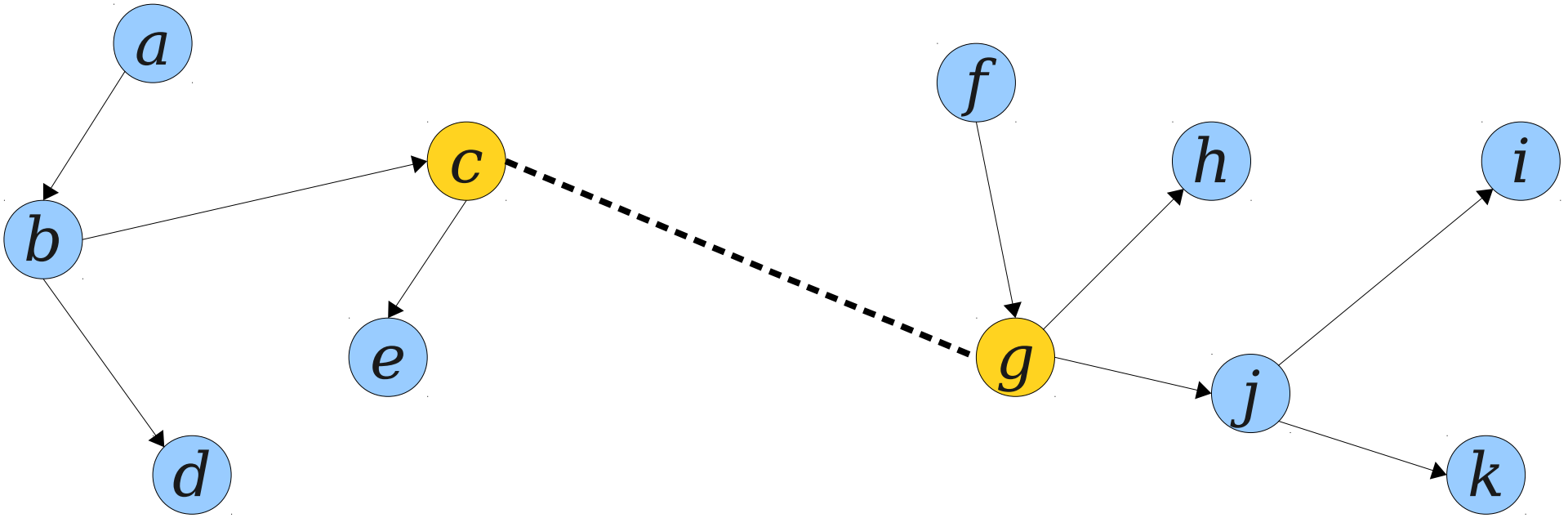


c e c b a a b d b

g j k j i j g h g f f

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

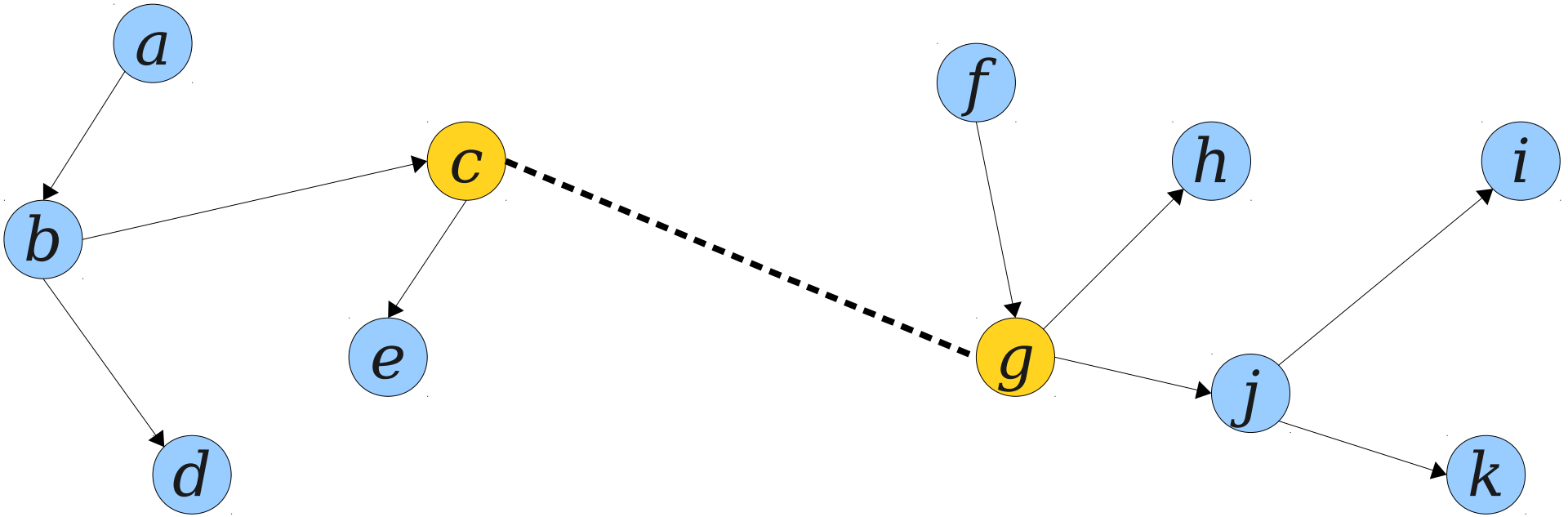


c e c b a b d b

g j k j i j g h g f

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

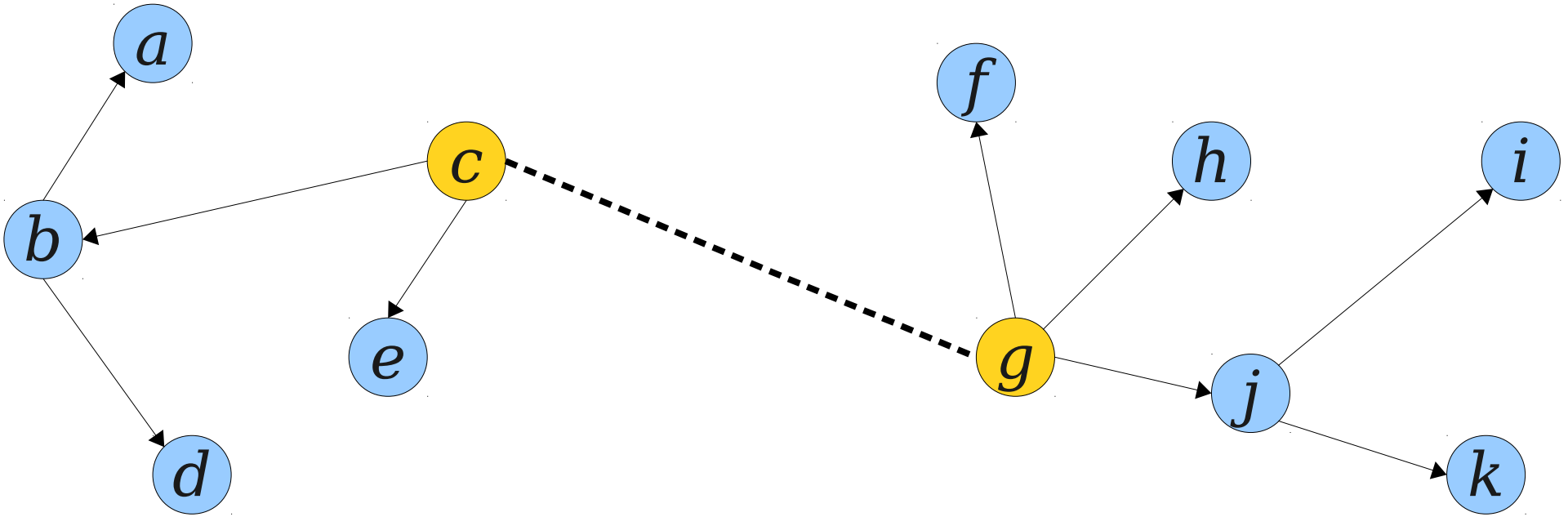


c e c b a b d b c

g j k j i j g h g f g

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

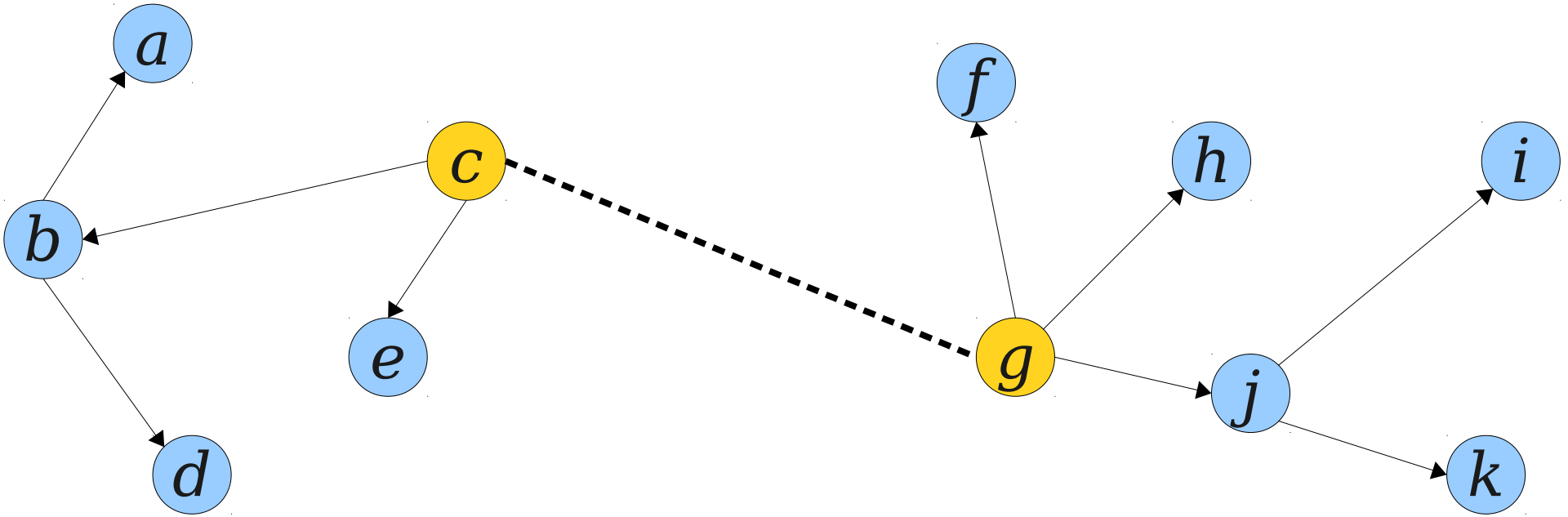


c e c b a b d b c

g j k j i j g h g f g

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

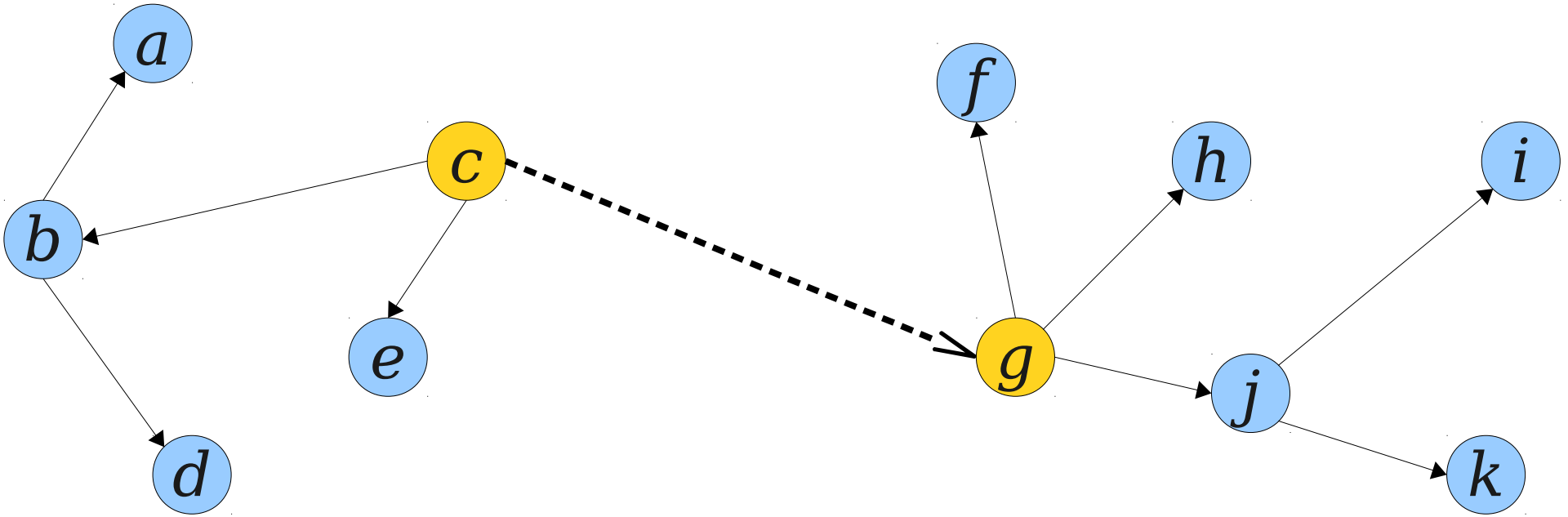


c e c b a b d b c

g j k j i j g h g f g

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

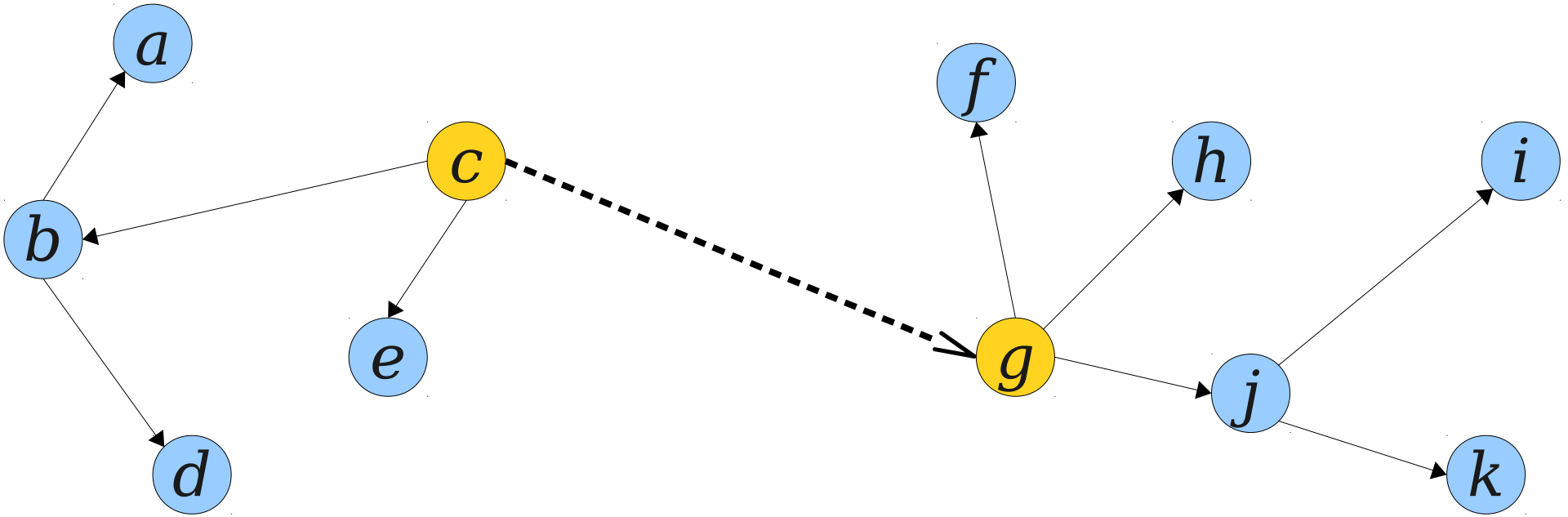


c e c b a b d b c

g j k j i j g h g f g

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:

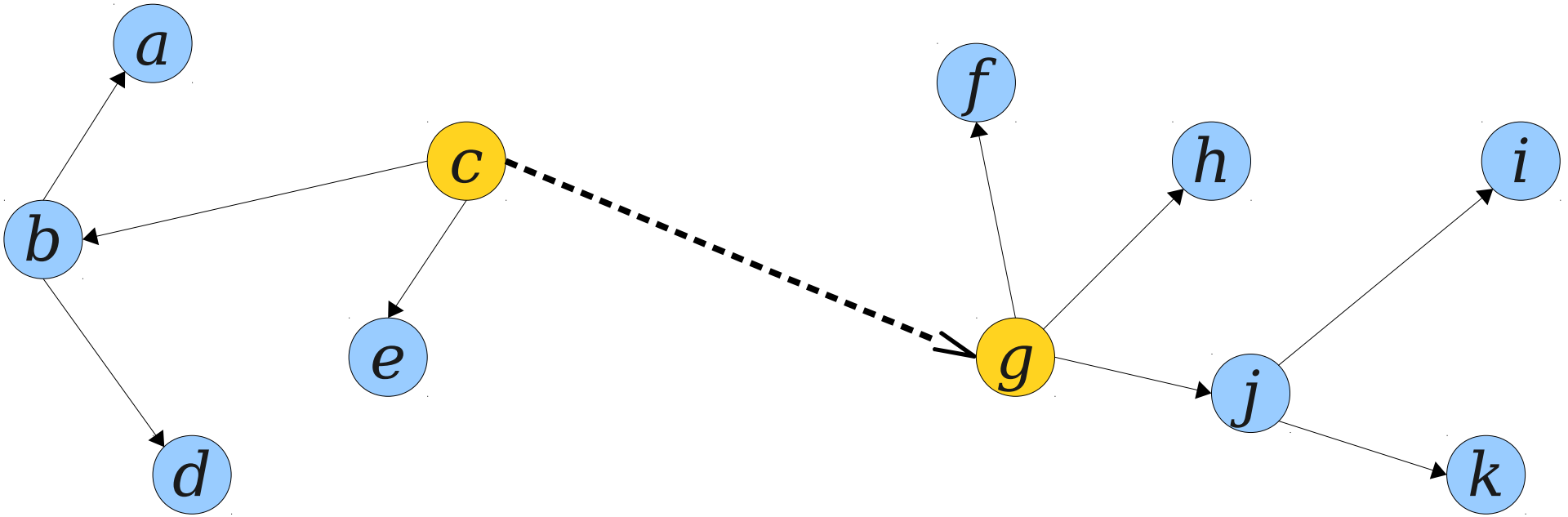


c e c b a b d b c

g j k j i j g h g f g

Euler Tours and Dynamic Trees

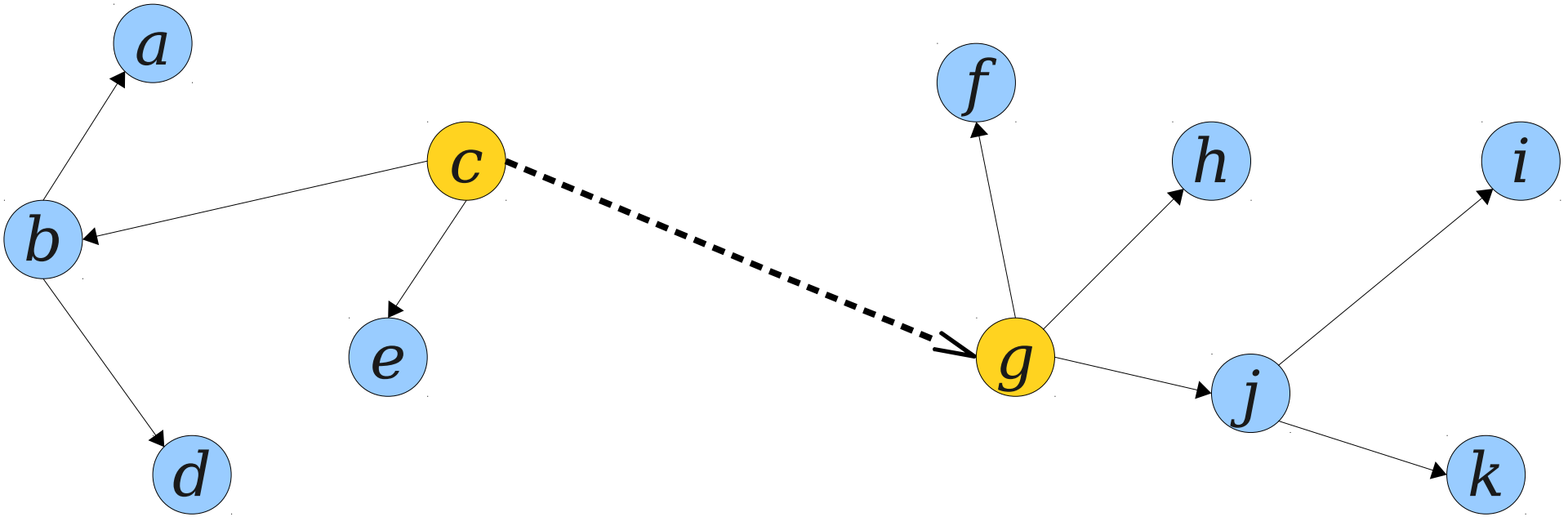
- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:



c e c b a b d b c g j k j i j g h g f g

Euler Tours and Dynamic Trees

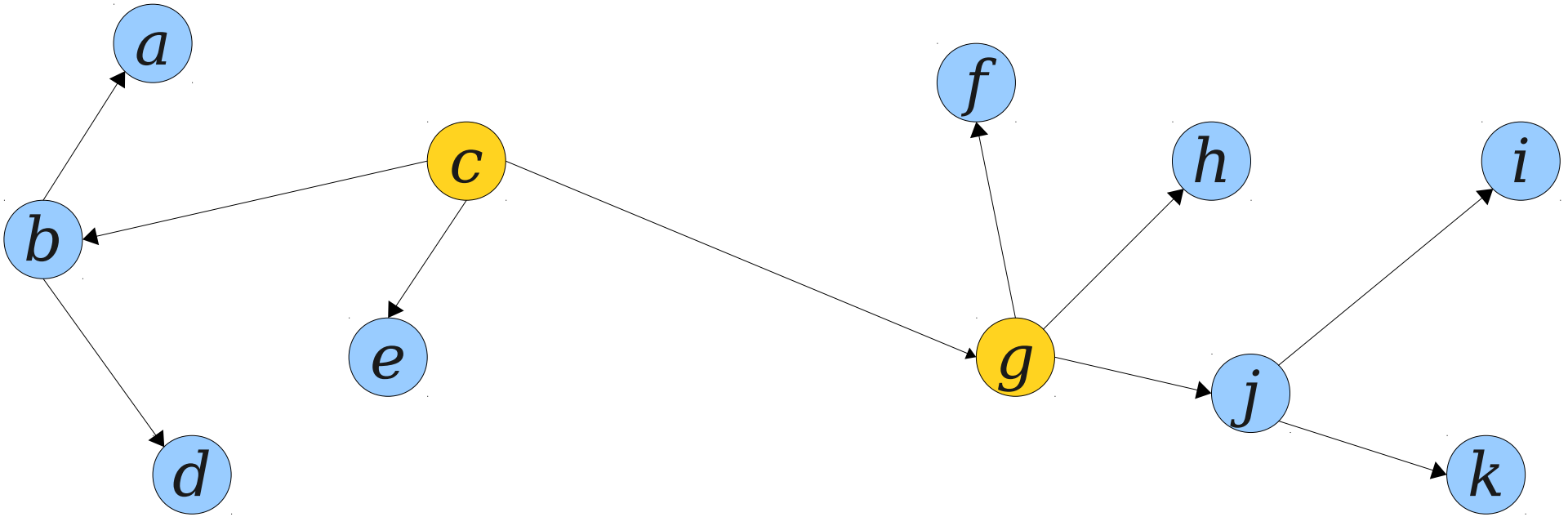
- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:



c e c b a b d b c g j k j i j g h g f g c

Euler Tours and Dynamic Trees

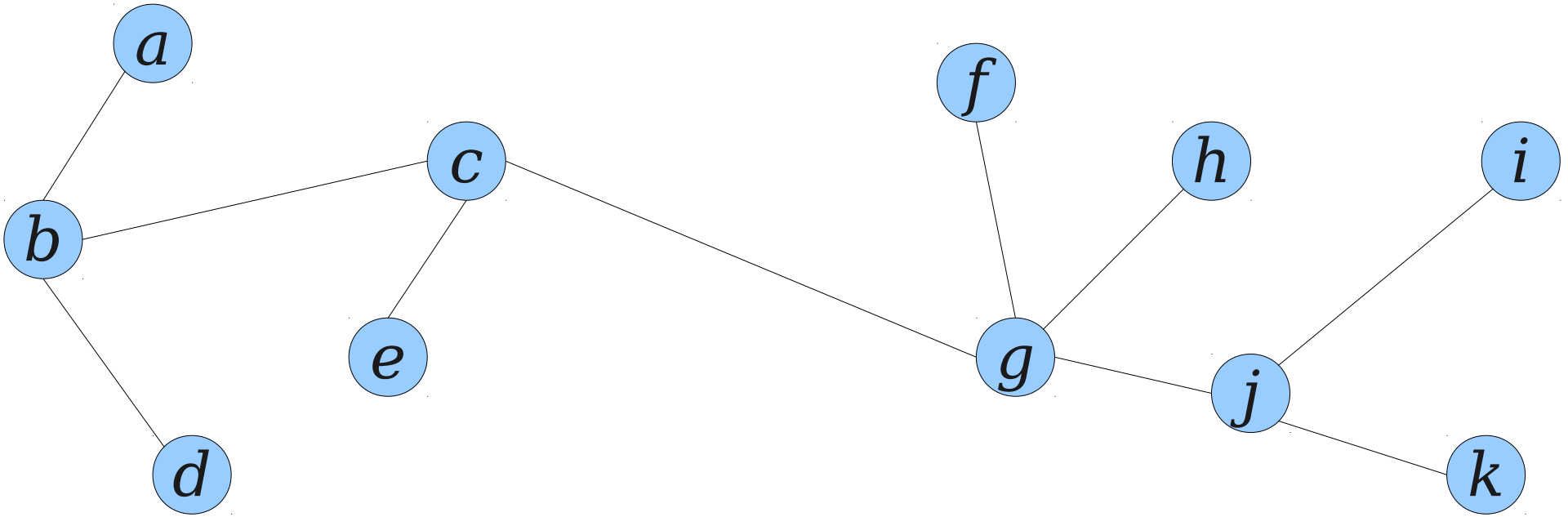
- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:



c e c b a b d b c g j k j i j g h g f g c

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- Watch what happens to the Euler tours:



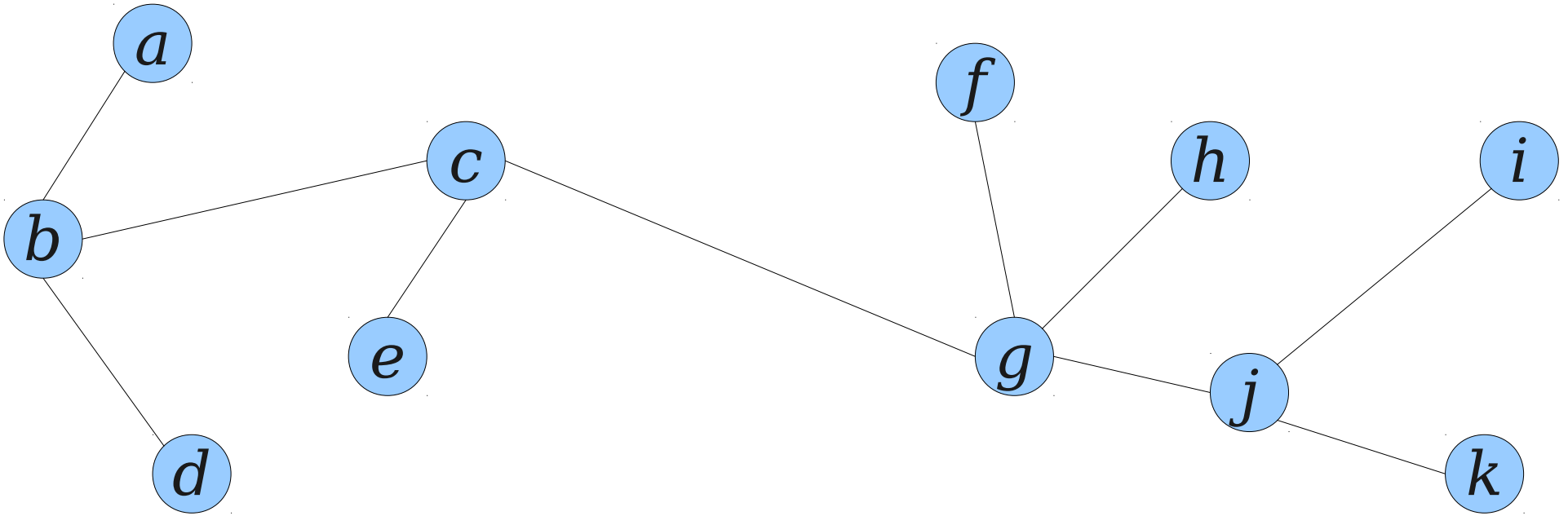
c e c b a b d b c g j k j i j g h g f g c

Euler Tours and Dynamic Trees

- Given two trees T_1 and T_2, where $u \in T_1$ and $v \in T_2$, executing ***link(u, v)*** links the trees together by adding edge $\{u, v\}$.
- To link T_1 and T_2 by adding $\{u, v\}$:
  - Let E_1 and E_2 be Euler tours of T_1 and T_2, respectively.
  - Rotate E_1 to root the tour at u.
  - Rotate E_2 to root the tour at v.
  - Concatenate E_1, E_2, $\{u\}$.

Euler Tours and Dynamic Trees

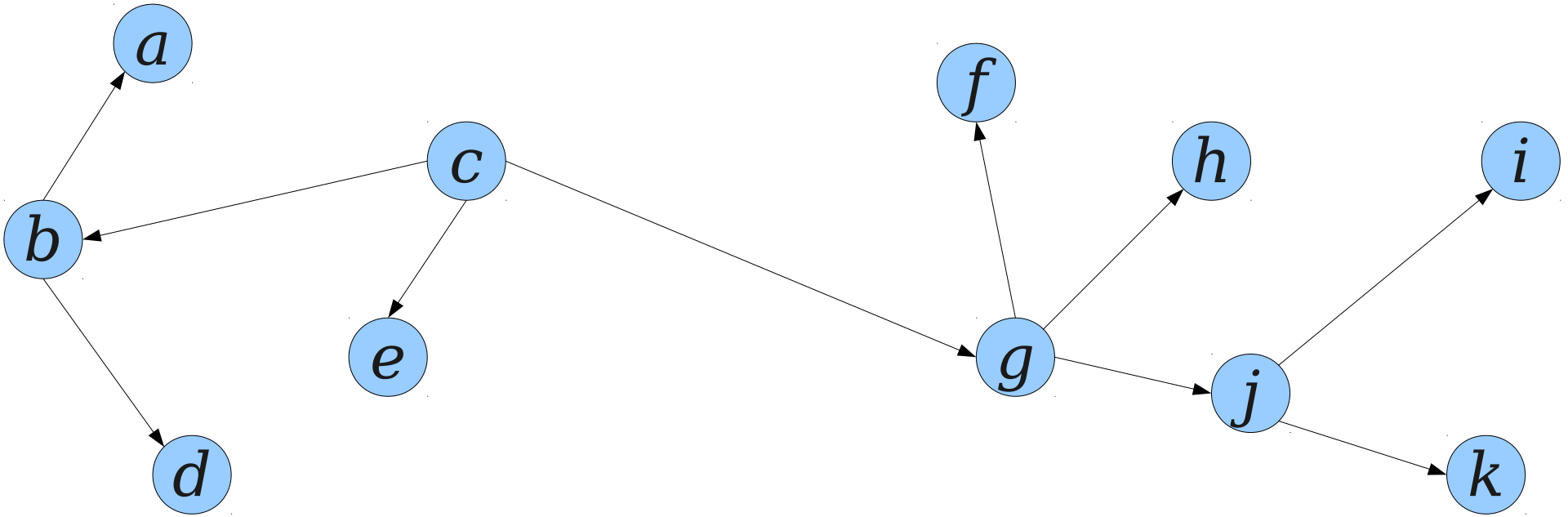
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



c e c b a b d b c g j k j i j g h g f g c

Euler Tours and Dynamic Trees

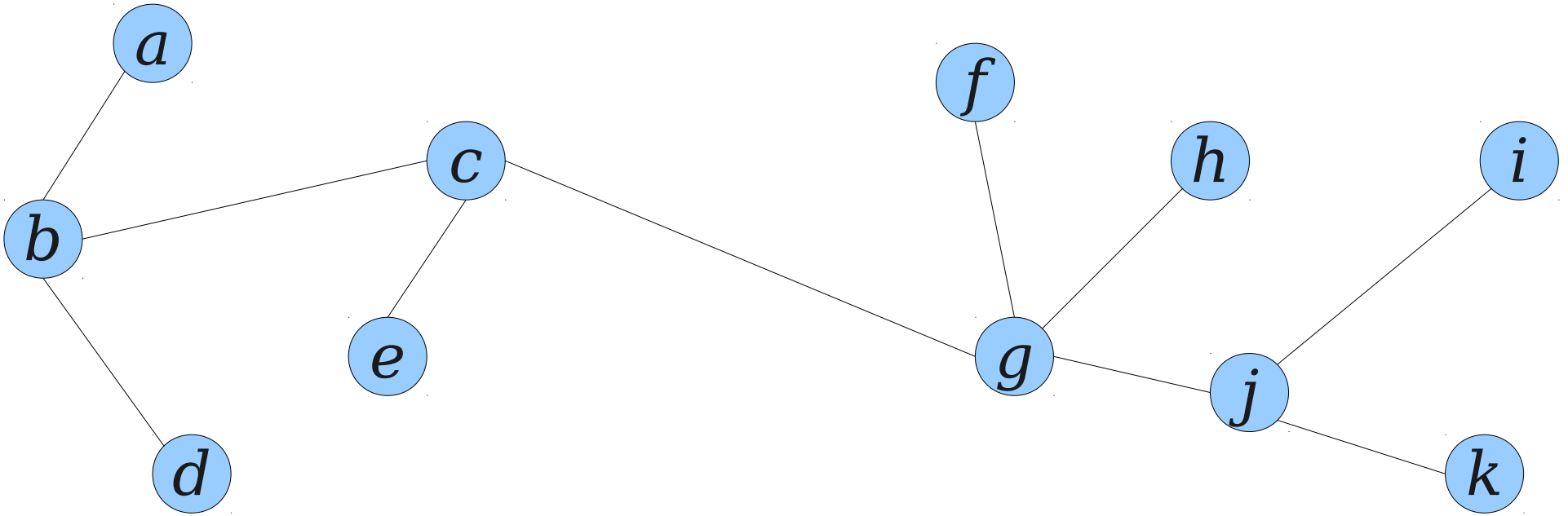
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



c e c b a b d b c g j k j i j g h g f g c

Euler Tours and Dynamic Trees

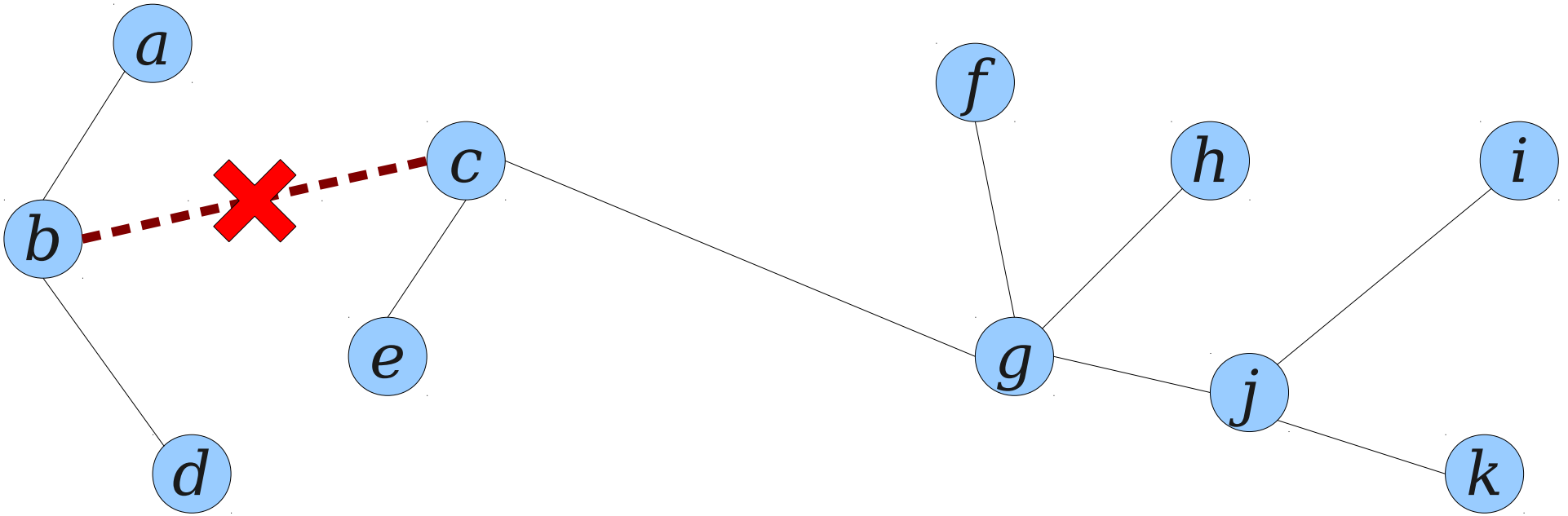
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



c e c b a b d b c g j k j i j g h g f g c

Euler Tours and Dynamic Trees

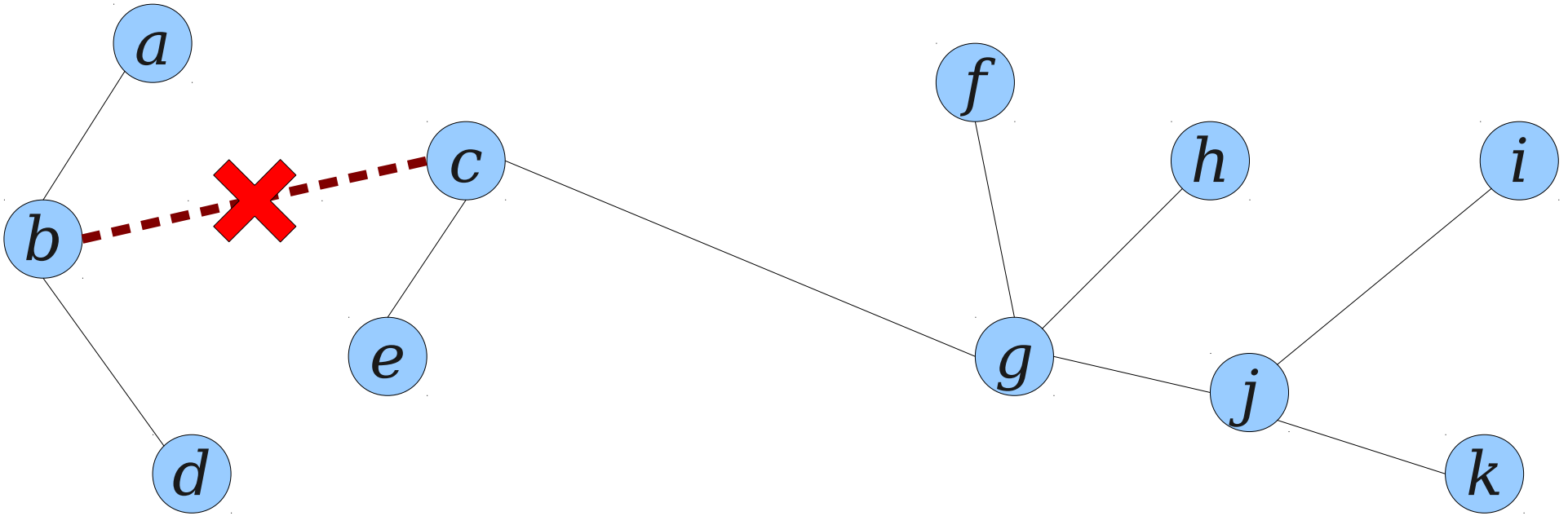
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



c e c b a b d b c g j k j i j g h g f g c

Euler Tours and Dynamic Trees

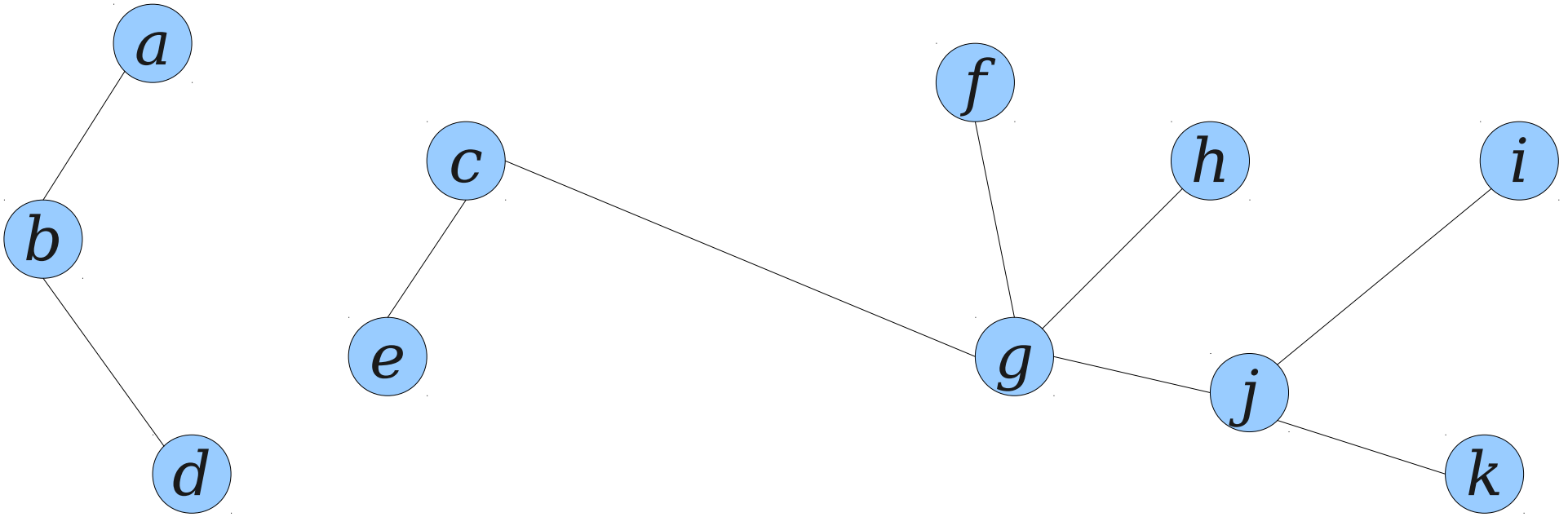
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



c e c b a b d b c g j k j i j g h g f g c

Euler Tours and Dynamic Trees

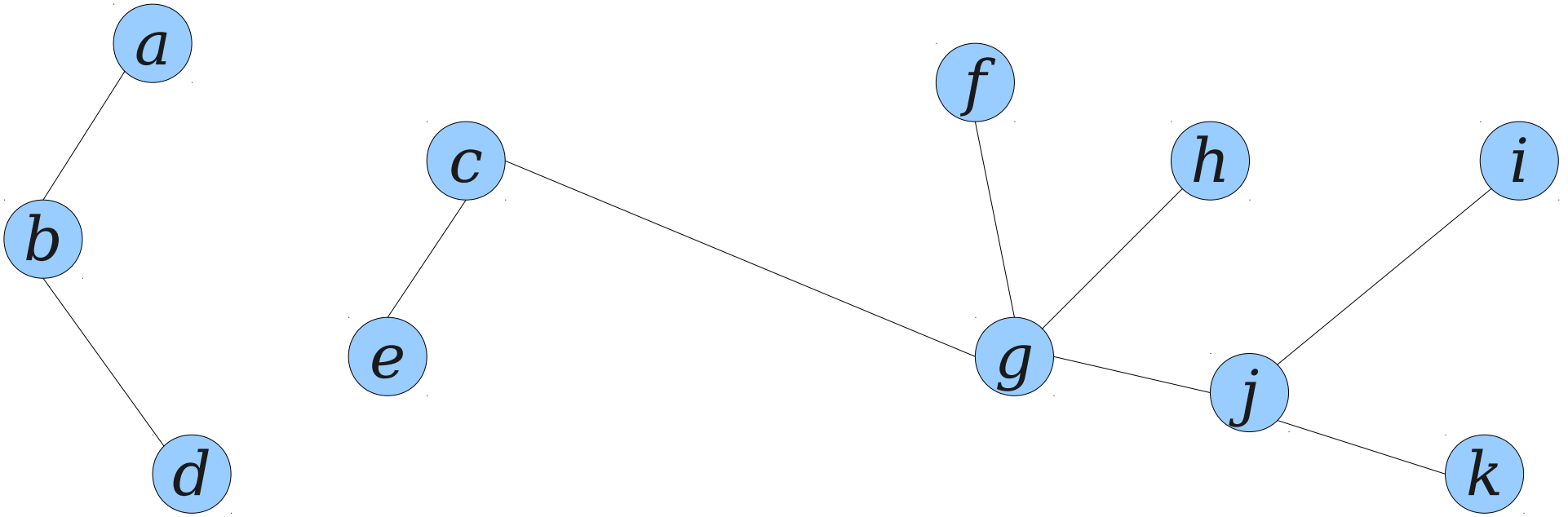
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



c e c b a b d b c g j k j i j g h g f g c

Euler Tours and Dynamic Trees

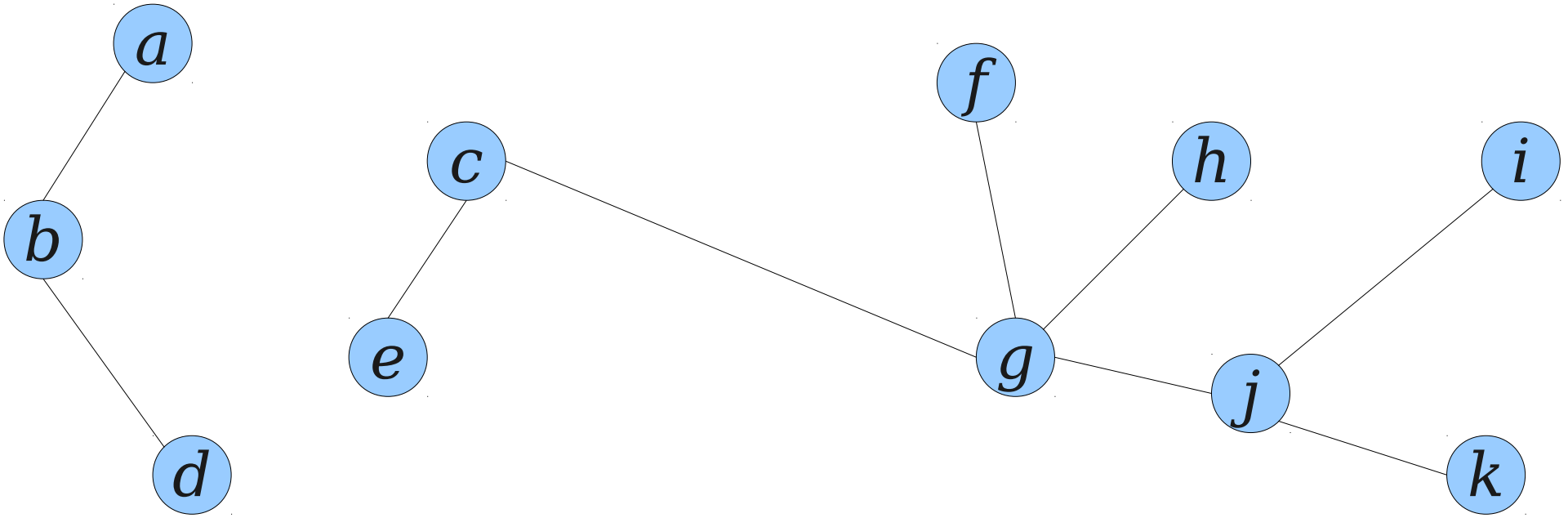
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



c e c b a b d b c g j k j i j g h g f g c

Euler Tours and Dynamic Trees

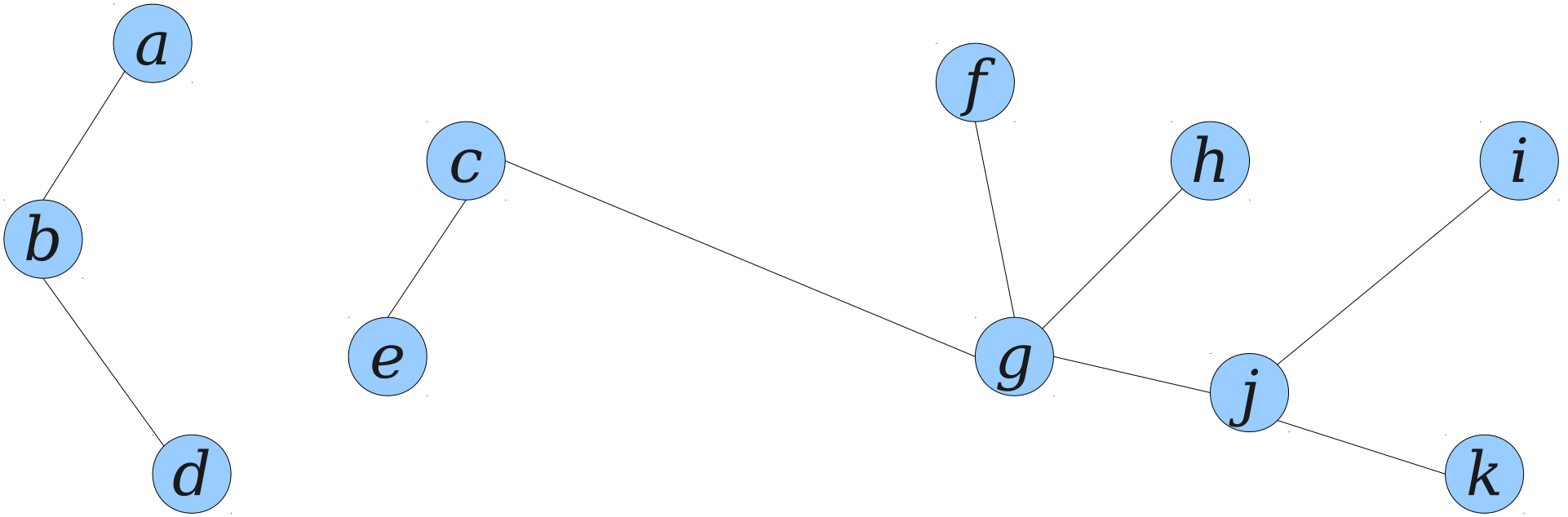
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***c e c*** ***c g j k j i j g h g f g c***

Euler Tours and Dynamic Trees

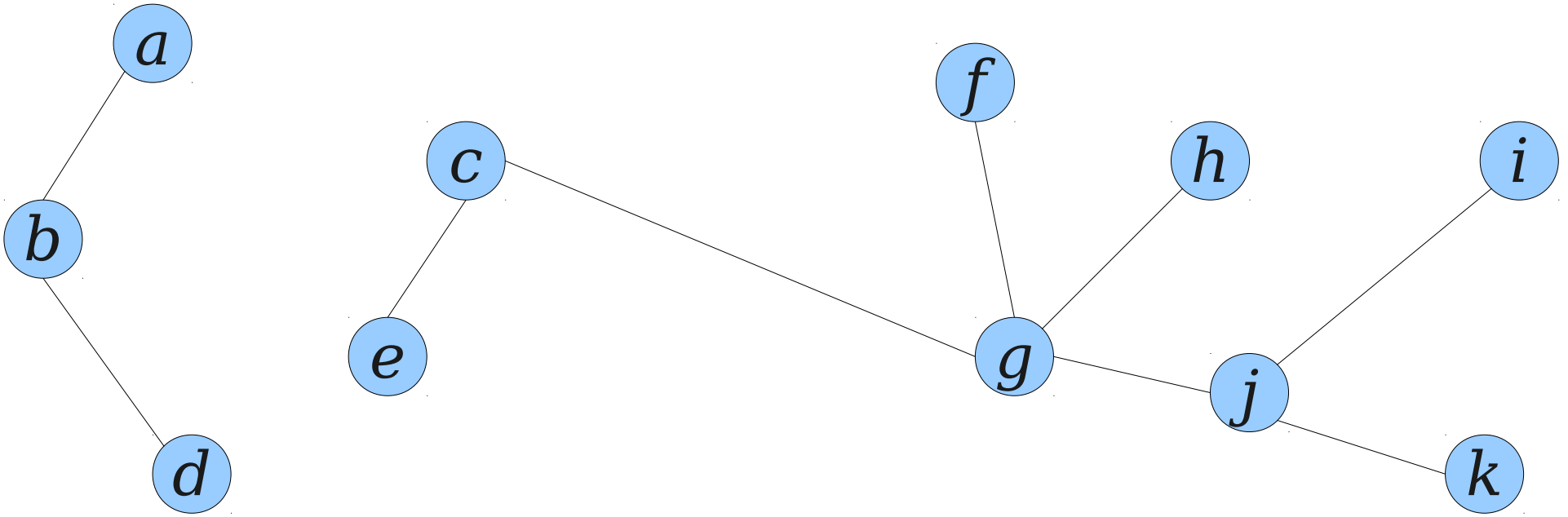
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b **c e c c g j k j i j g h g f g c**

Euler Tours and Dynamic Trees

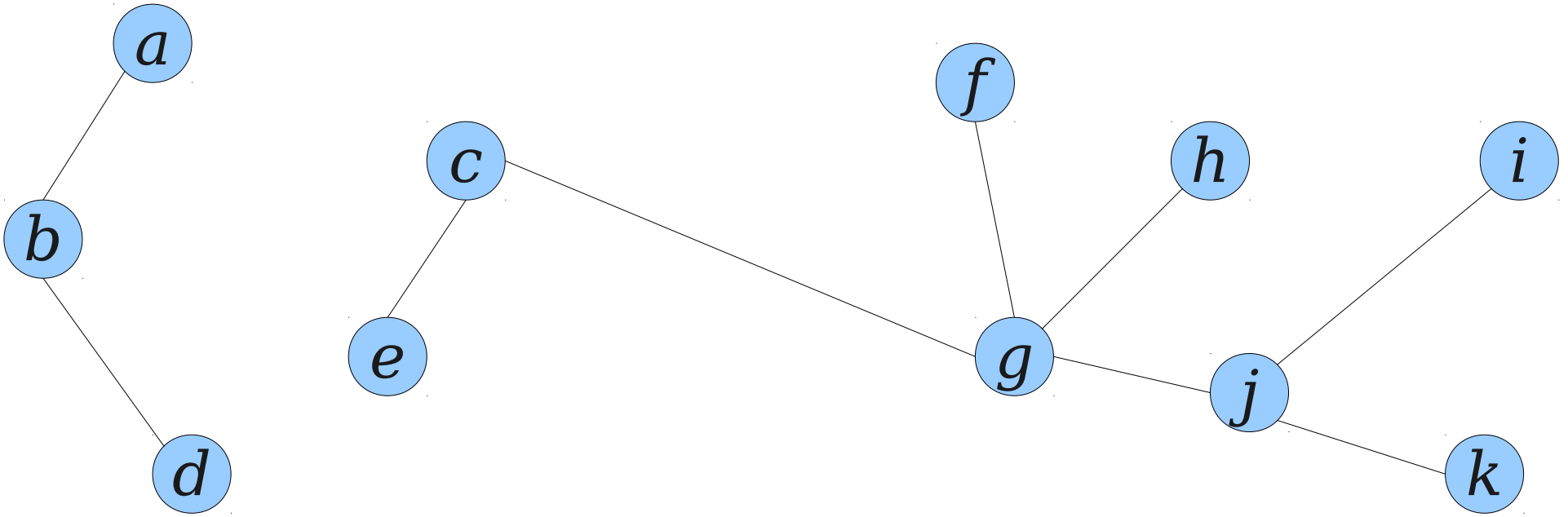
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b **c e c g j k j i j g h g f g c**

Euler Tours and Dynamic Trees

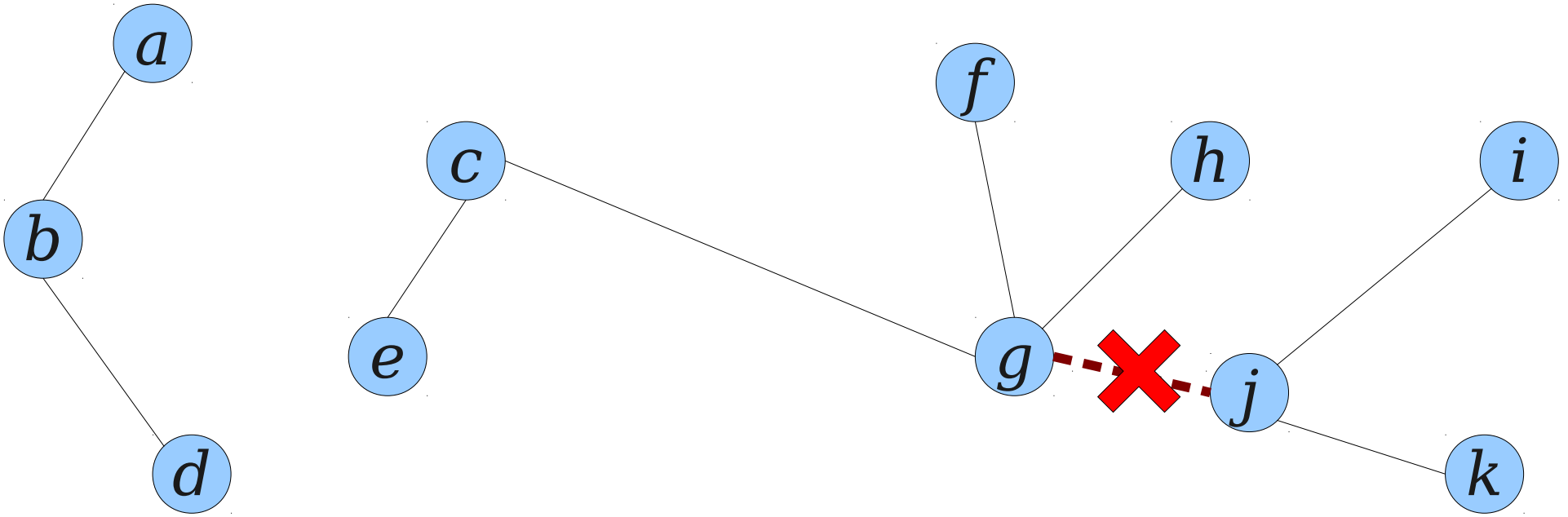
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***c e c g j k j i j g h g f g c***

Euler Tours and Dynamic Trees

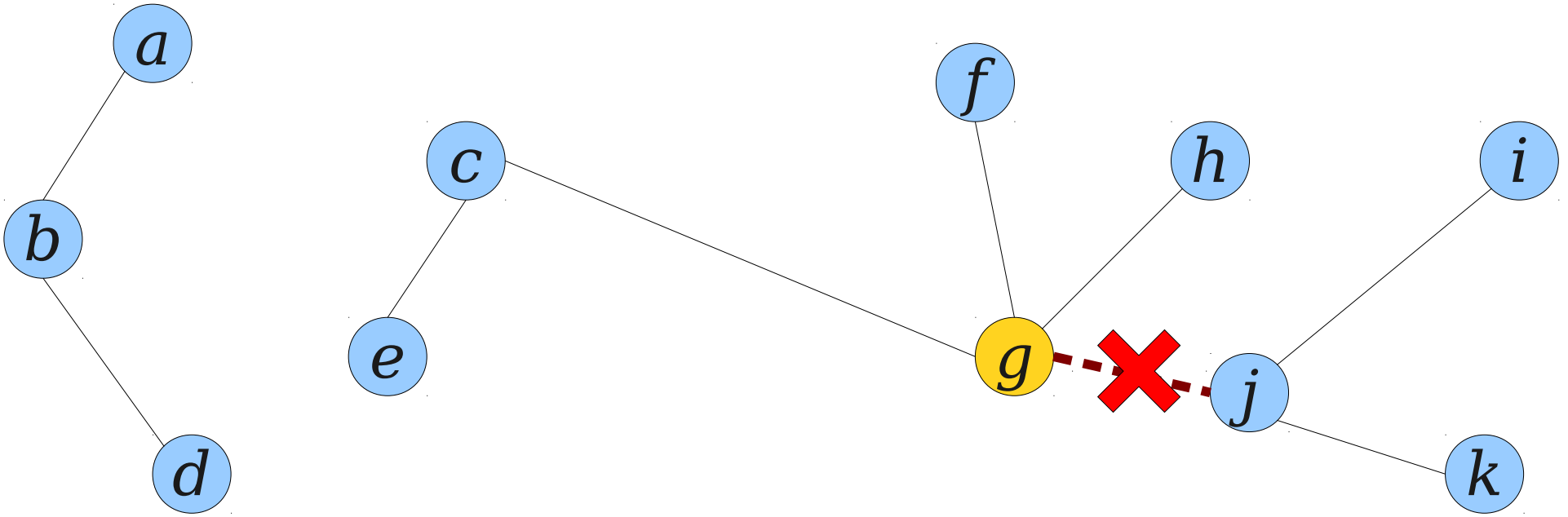
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***c e c g j k j i j g h g f g c***

Euler Tours and Dynamic Trees

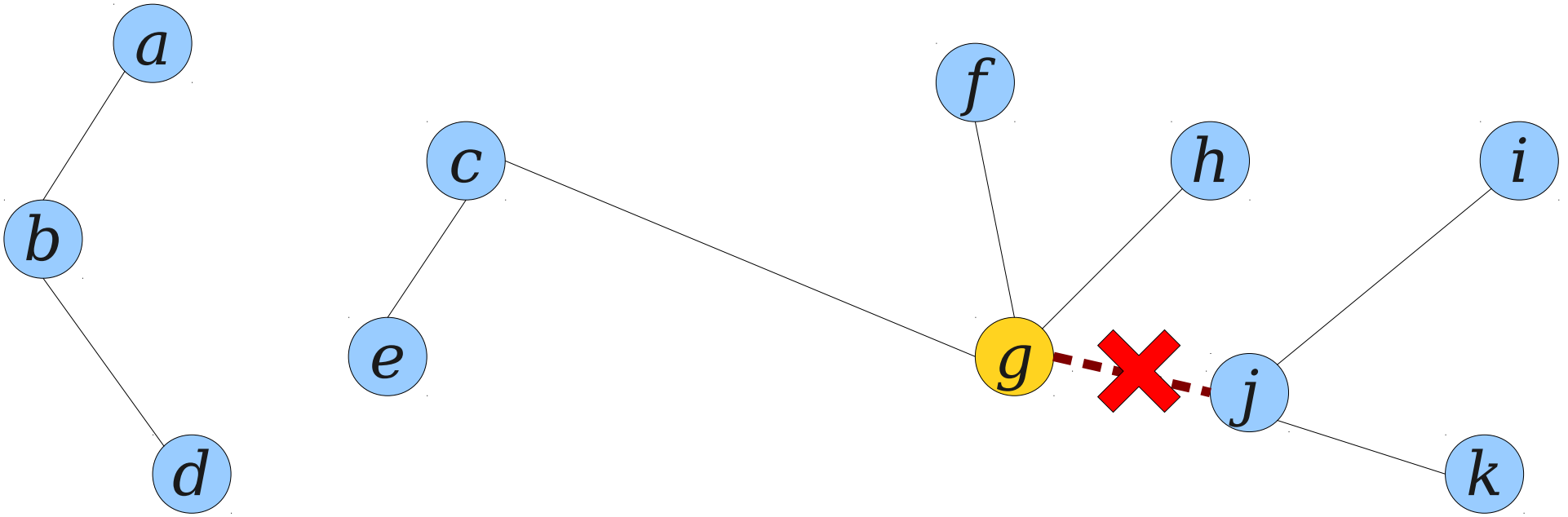
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***c e c g j k j i j g h g f g c***

Euler Tours and Dynamic Trees

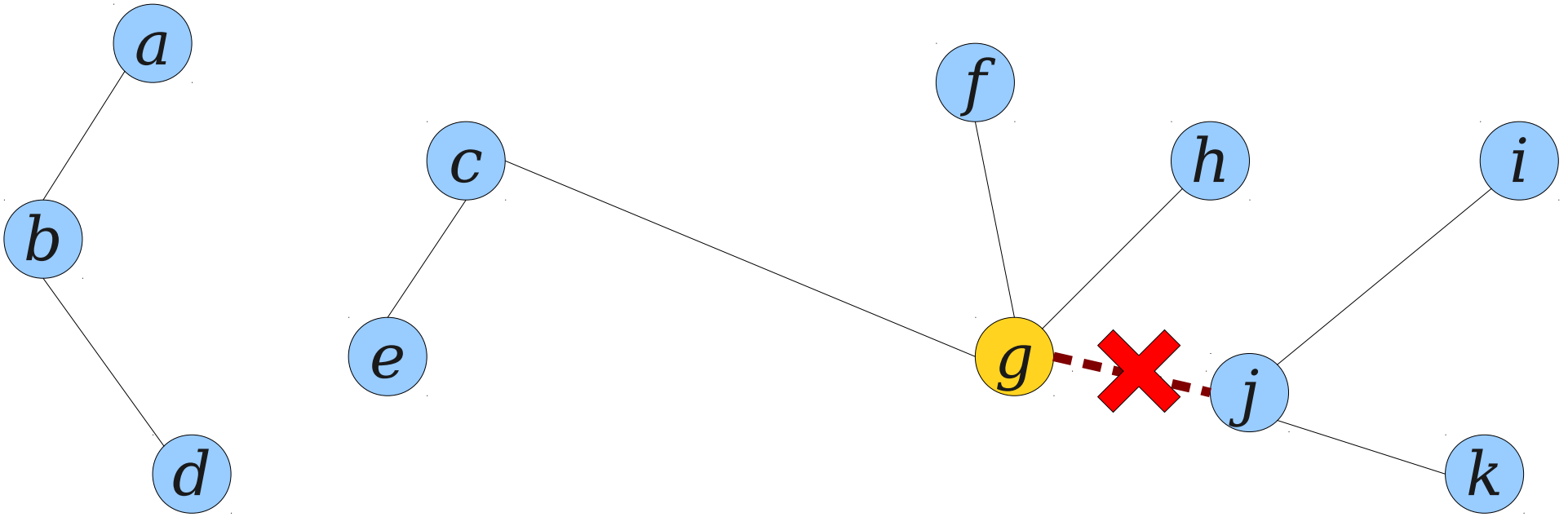
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***c e c g j k j i j g h g f g c***

Euler Tours and Dynamic Trees

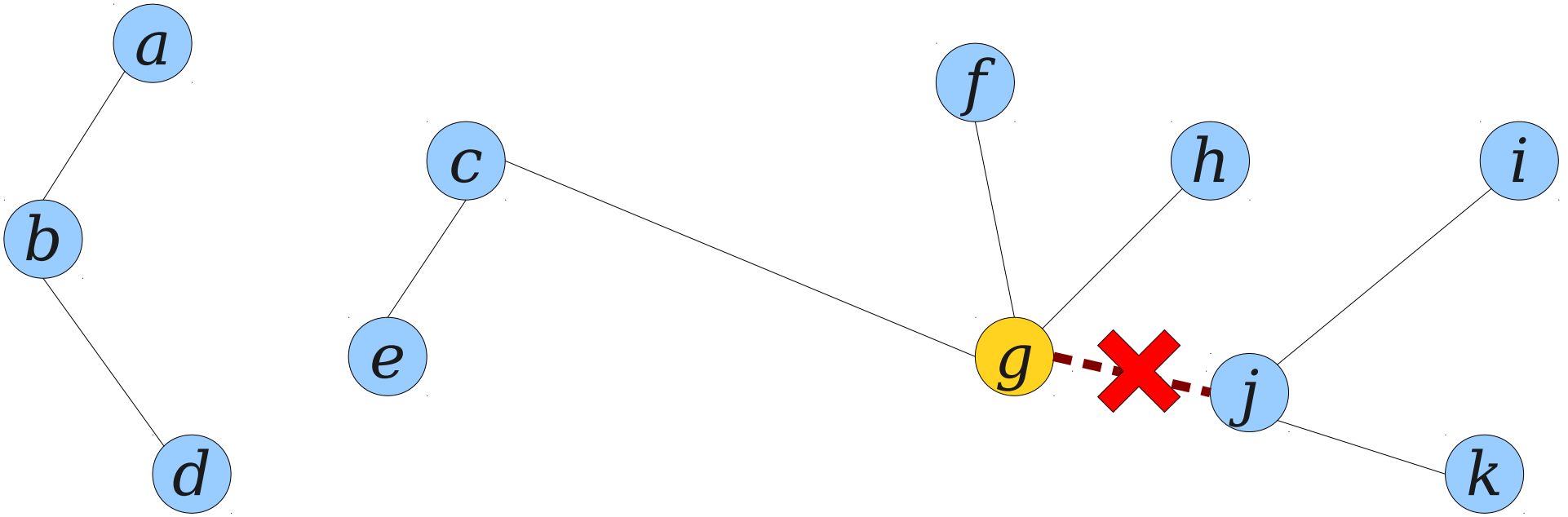
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b **c e c** **g j k j i j g h g f g** **c**

Euler Tours and Dynamic Trees

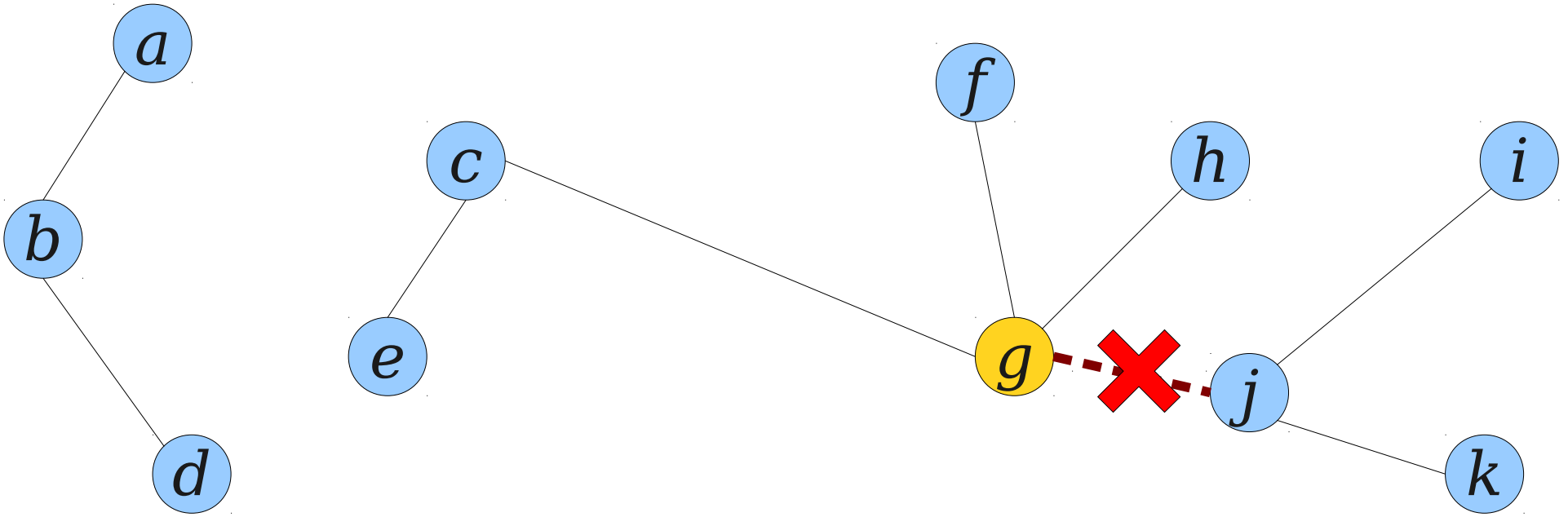
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b **g j k j i j g h g f g** **c e c** **c**

Euler Tours and Dynamic Trees

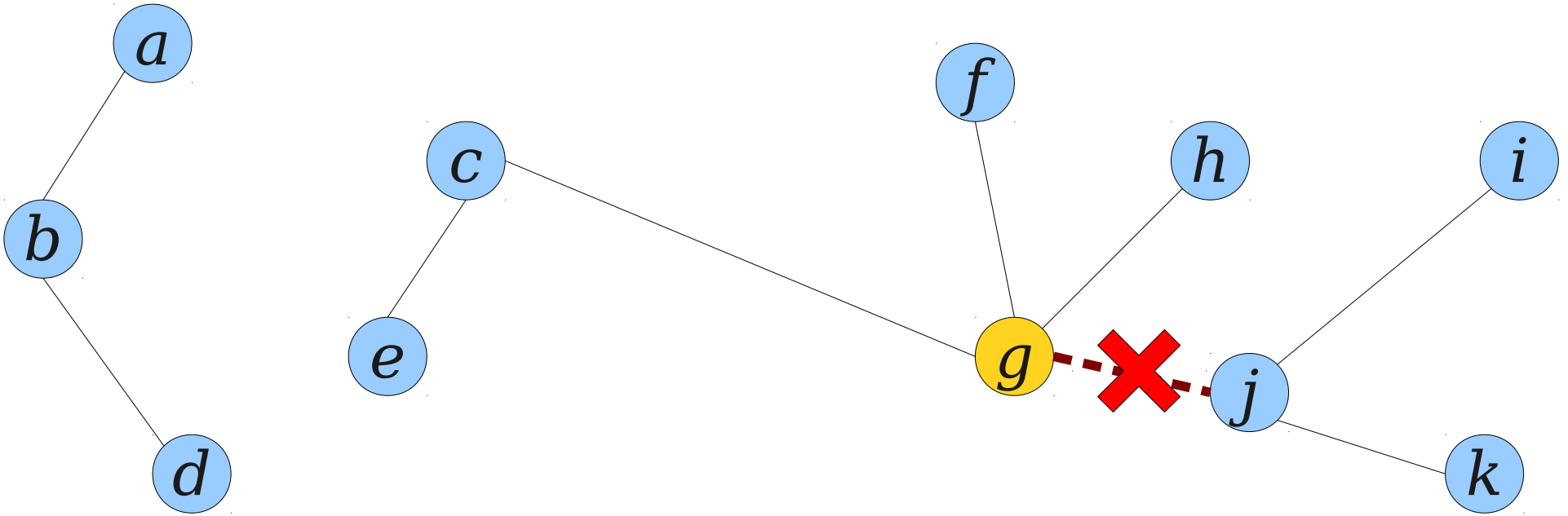
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***g j k j i j g h g f g c c e c***

Euler Tours and Dynamic Trees

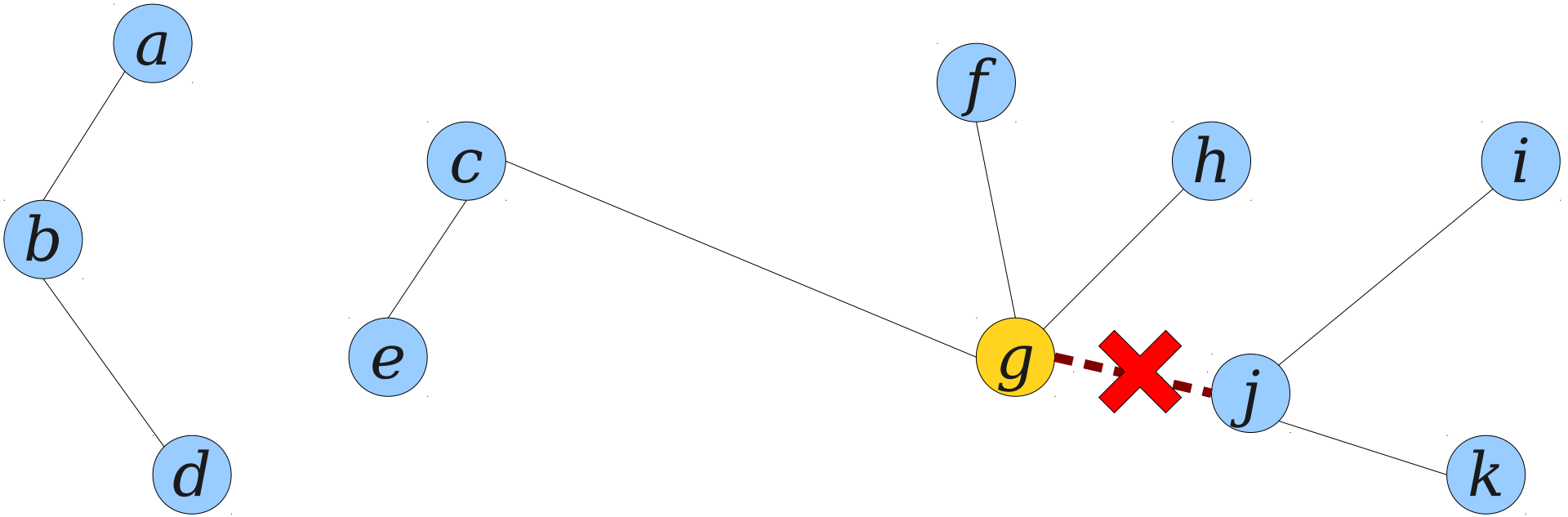
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***g j k j i j g h g f g c e c***

Euler Tours and Dynamic Trees

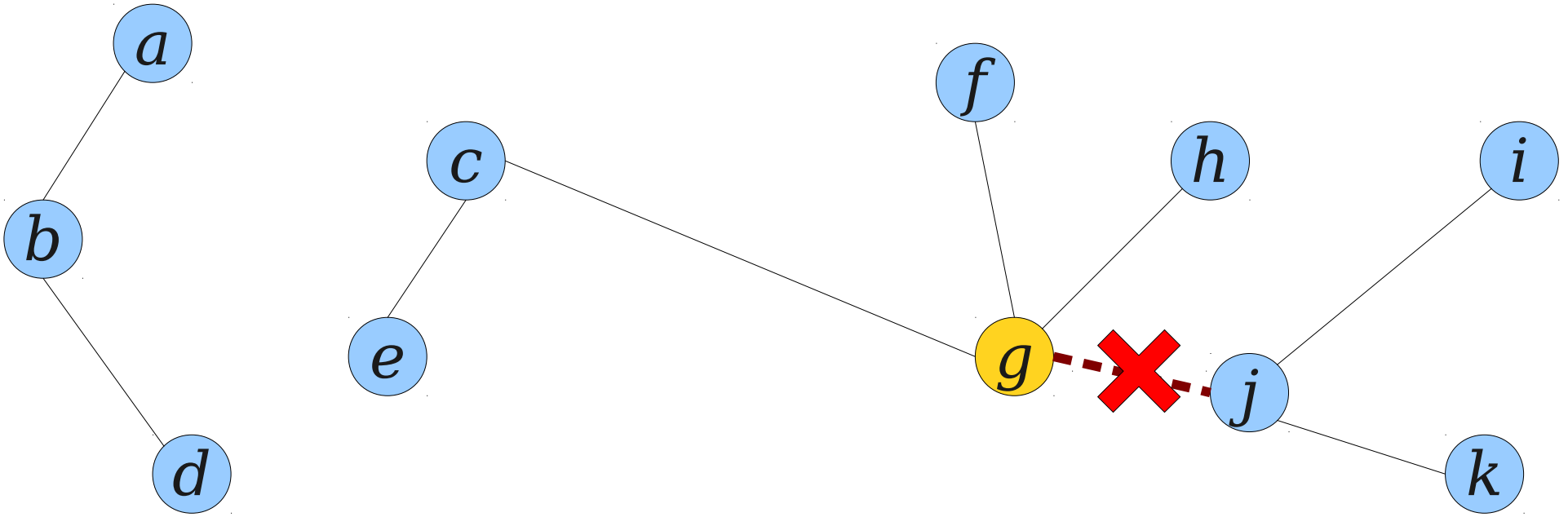
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***g j k j i j g h g f g c e c g***

Euler Tours and Dynamic Trees

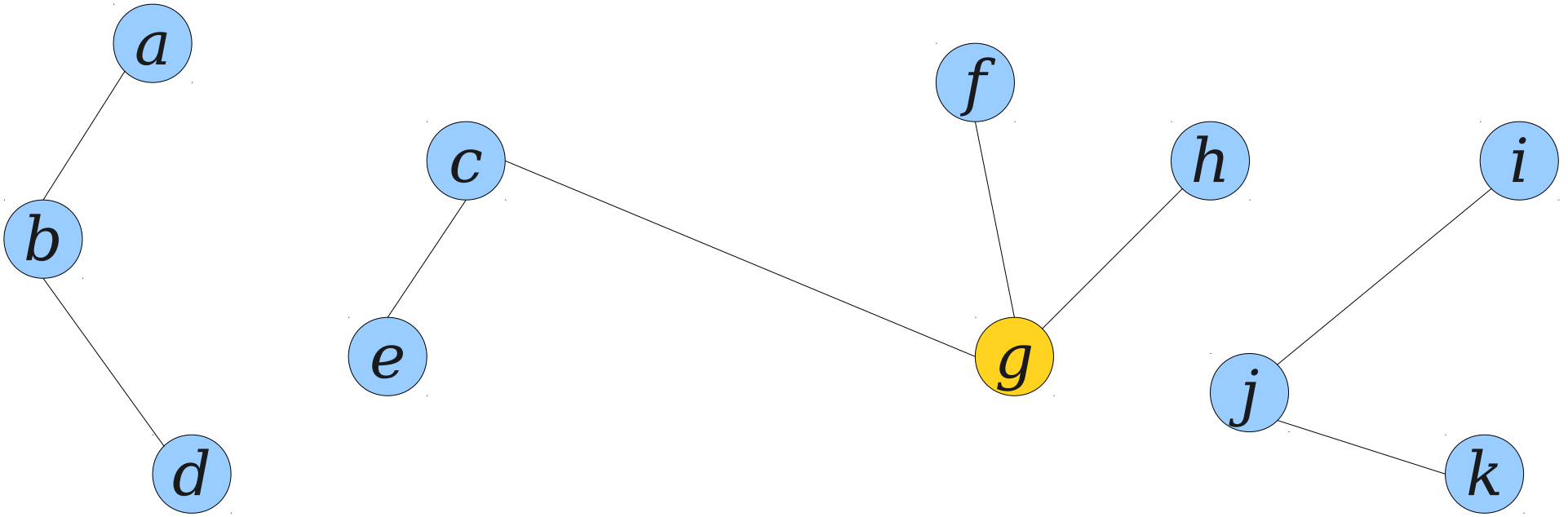
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***g j k j i j g h g f g c e c g***

Euler Tours and Dynamic Trees

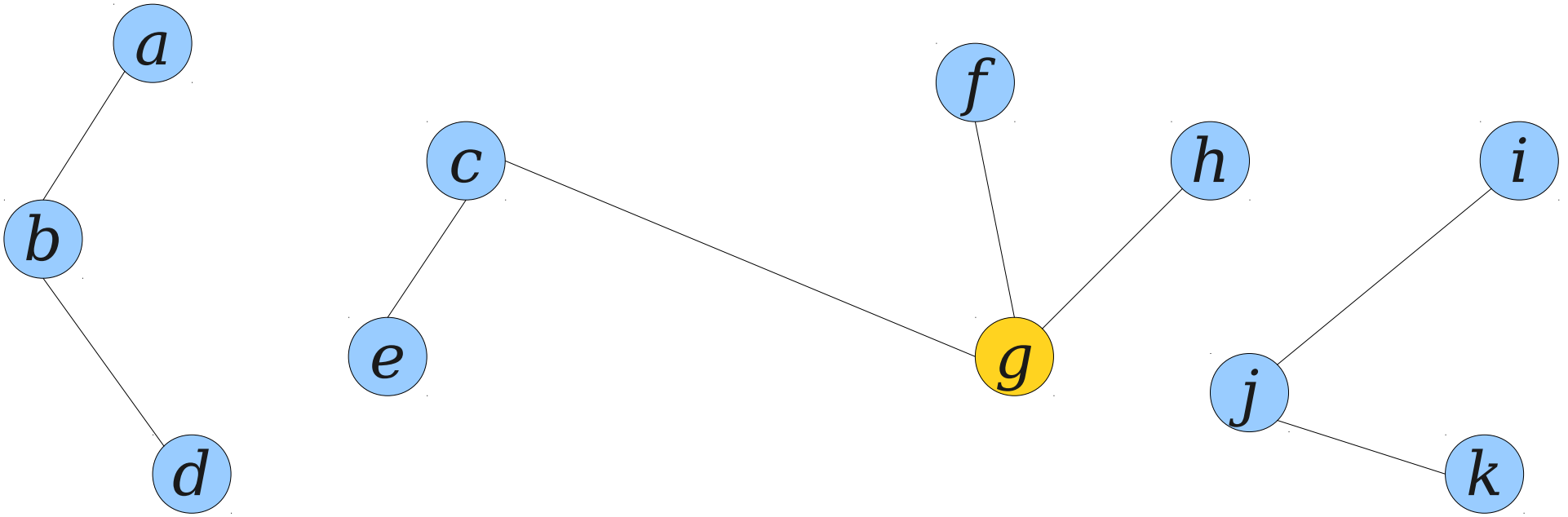
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***g j k j i j g h g f g c e c g***

Euler Tours and Dynamic Trees

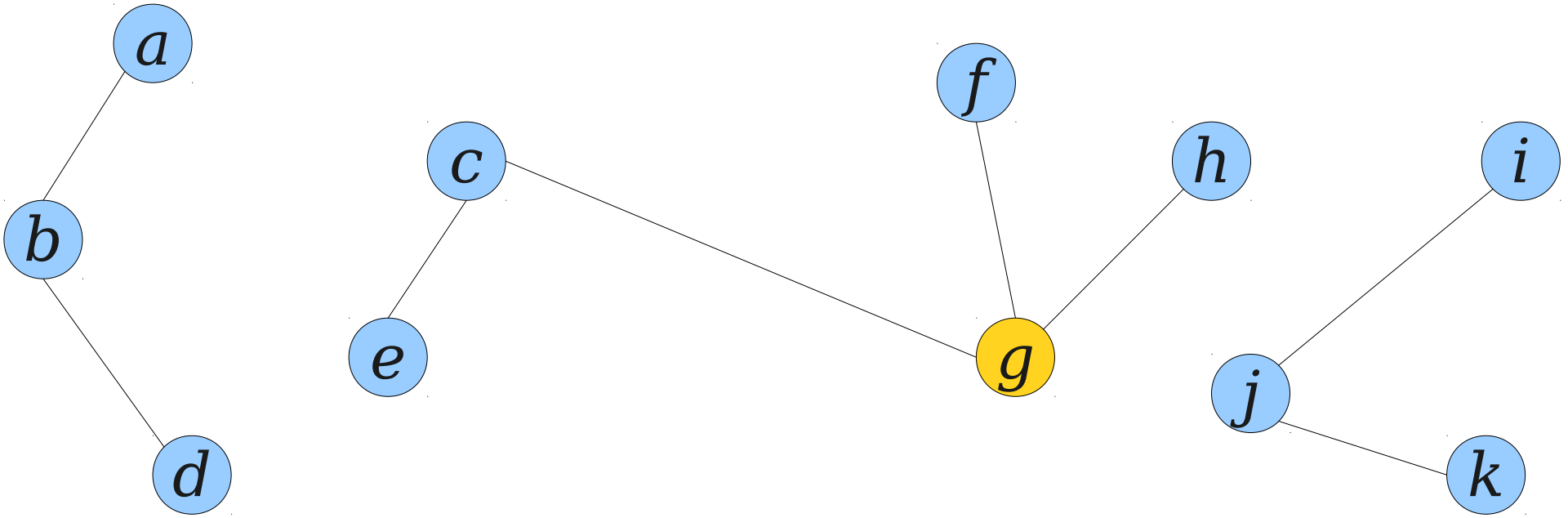
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***g j k j i j g h g f g c e c g***

Euler Tours and Dynamic Trees

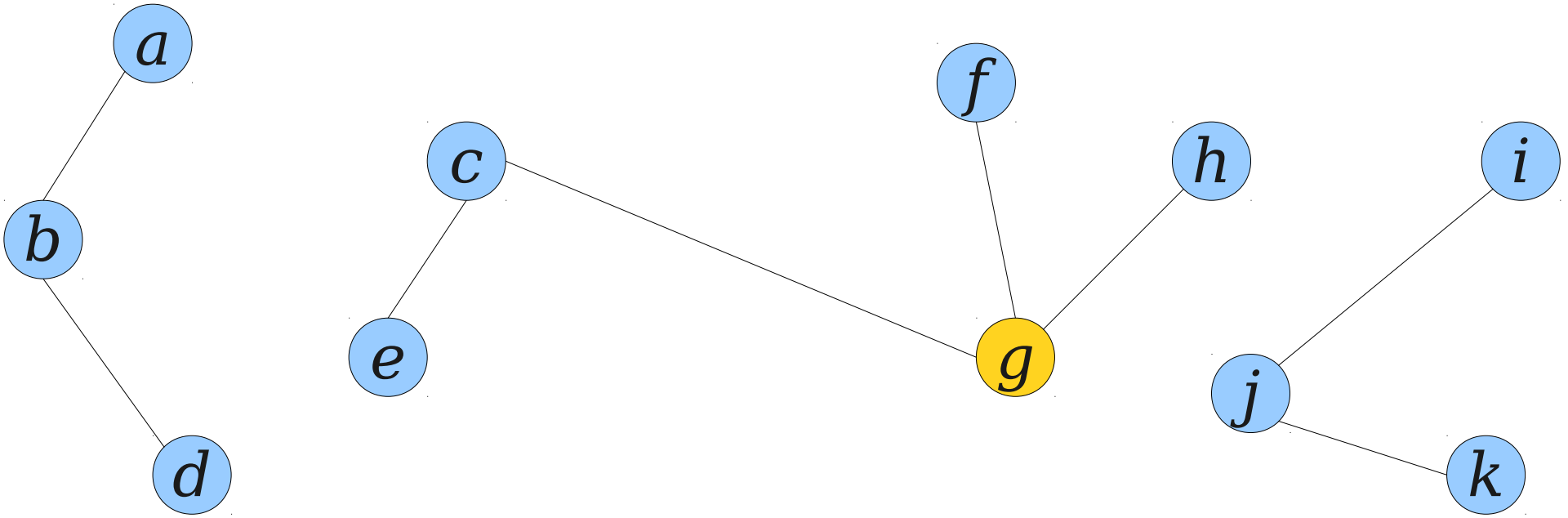
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b **g** **j k j i j** **g h g f g c e c g**

Euler Tours and Dynamic Trees

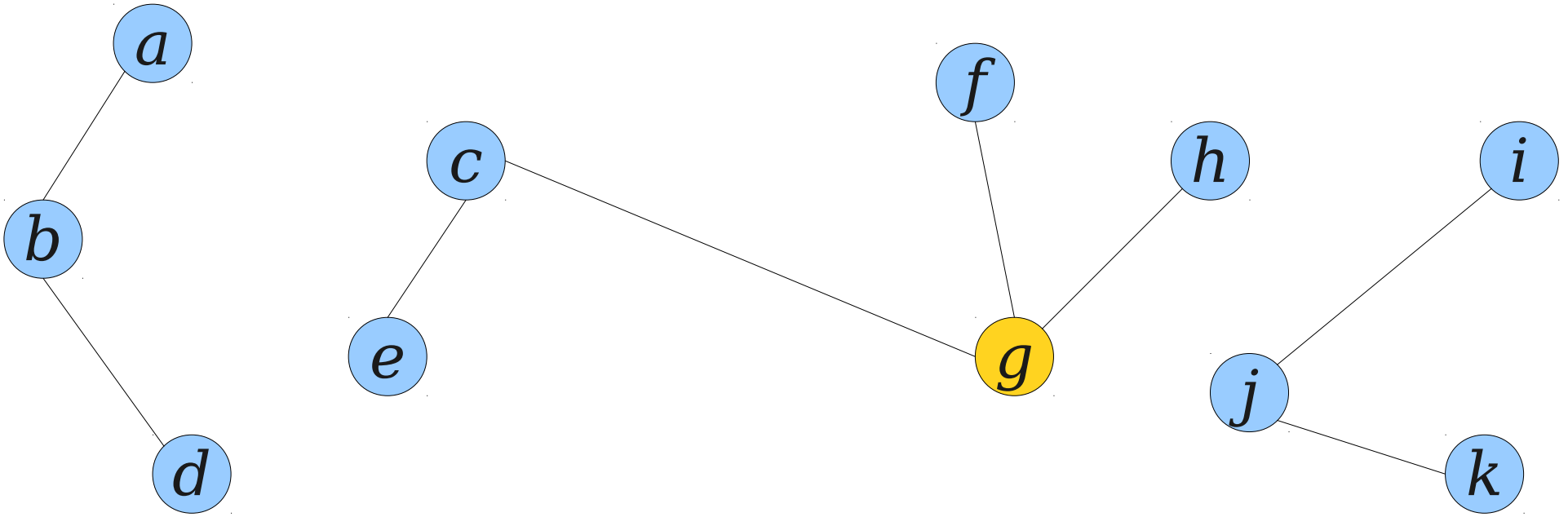
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***g*** ***g h g f g c e c g*** ***j k j i j***

Euler Tours and Dynamic Trees

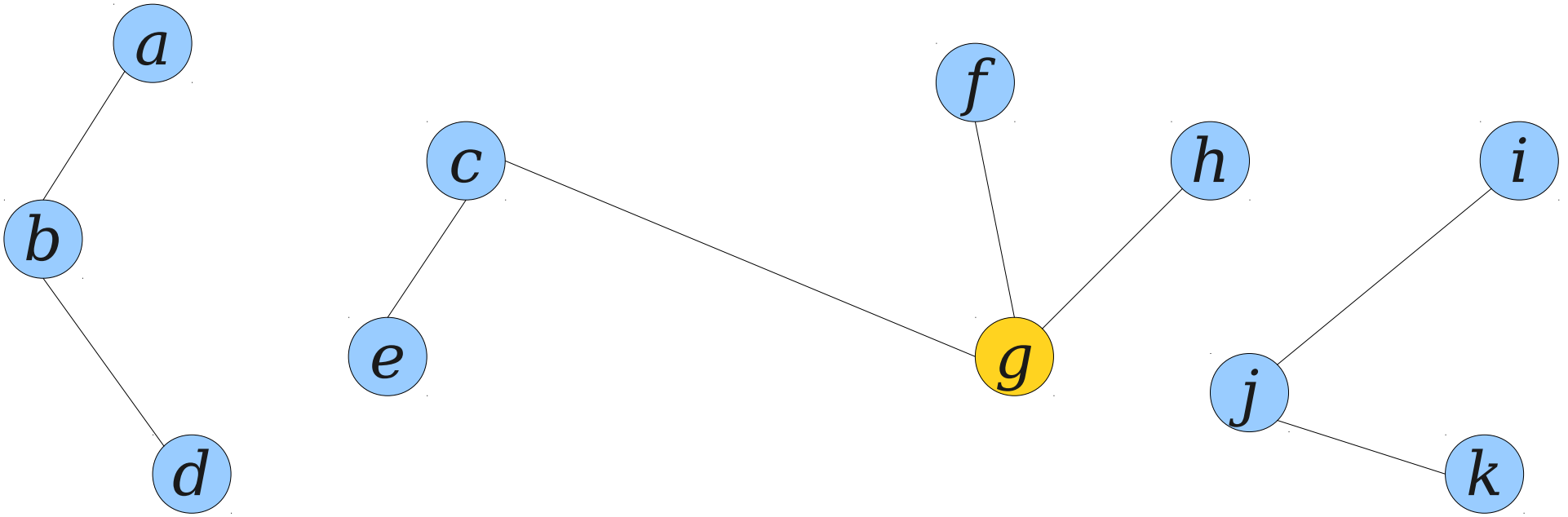
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***g g h g f g c e c g*** ***j k j i j***

Euler Tours and Dynamic Trees

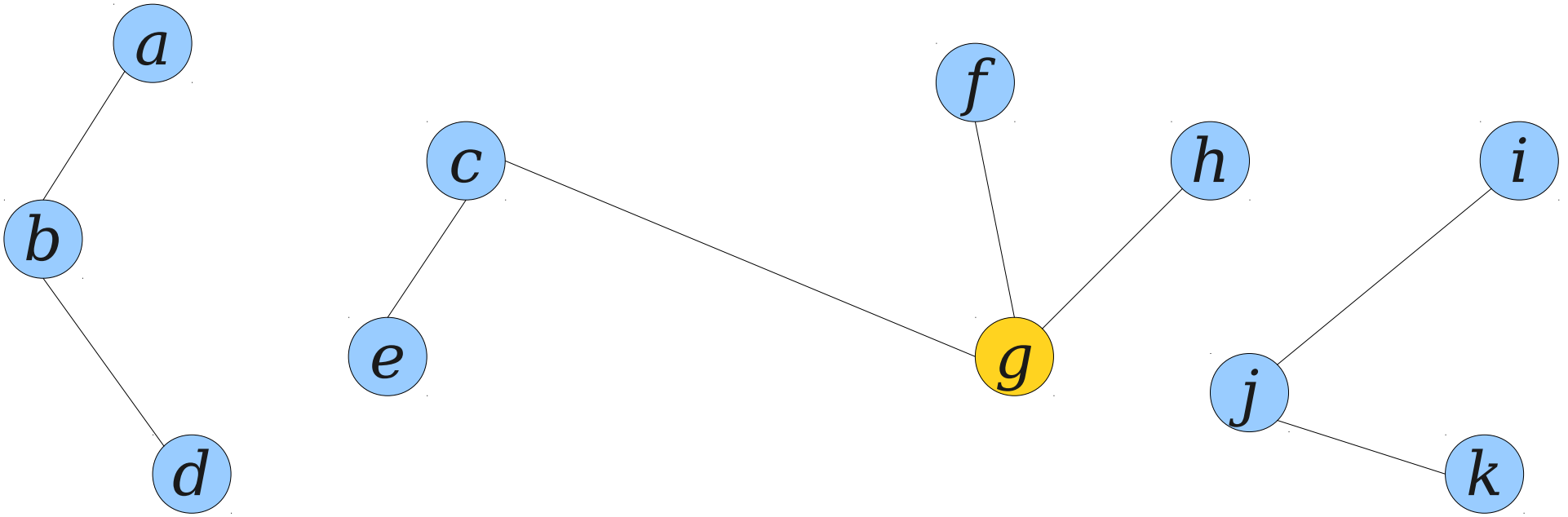
- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***g h g f g c e c g*** ***j k j i j***

Euler Tours and Dynamic Trees

- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- Watch what happens to the Euler tour of T:



b a b d b ***g h g f g c e c g*** ***j k j i j***

Euler Tours and Dynamic Trees

- Given a tree T, executing **cut(u, v)** cuts the edge $\{u, v\}$ from the tree (assuming it exists).
- To cut T into T_1 and T_2 by cutting $\{u, v\}$:
  - Let E be an Euler tour for T.
  - Rotate u to the front of E.
  - Split E into E_1, V, E_2, where V is the span between the first and last occurrence of v.
  - T_1 has the Euler tour formed by concatenating E_1 and E_2, deleting the extra u at the join point.
  - T_2 has Euler tour V.

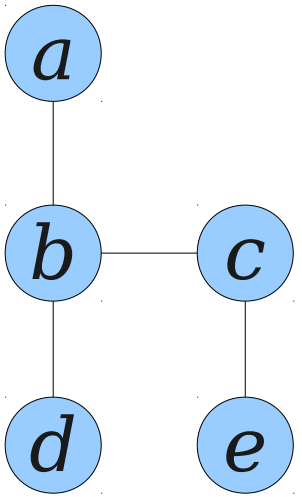
The Story So Far

- **Goal:** Implement *link*, *cut*, and *is-connected* as efficiently as possible.
- By representing trees via their Euler tours, can implement *link* and *cut* so that only O(1) joins and splits are necessary per operation.
- Questions to answer:
  - How do we efficiently implement these joins and splits?
  - Once we have the tours, how do we answer connectivity queries?

Implementing the Structure

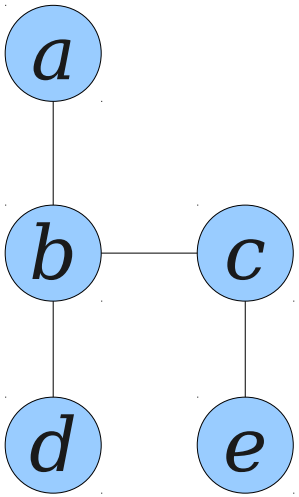
- The operations we have seen require us to be able to efficiently do the following:
  - Identify the first and last copy of a node in a sequence.
  - Split a sequence at those positions.
  - Concatenate sequences.
  - Add a new copy of a node to a sequence.
  - Delete a duplicate copy of a node from a sequence.
- How do we do this efficiently?

An Initial Idea: Linked Lists

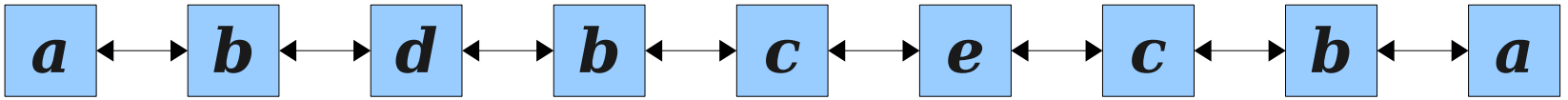


a b d b c e c b a

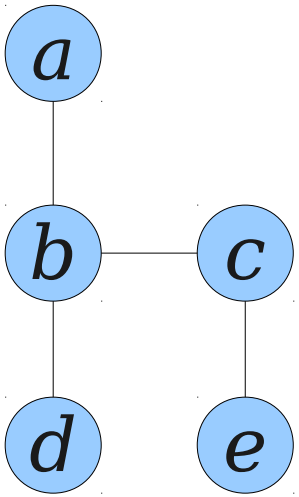
An Initial Idea: Linked Lists



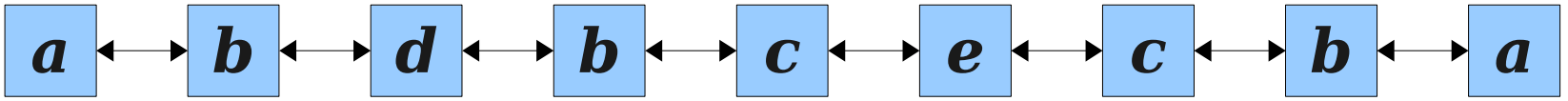
a b d b c e c b a



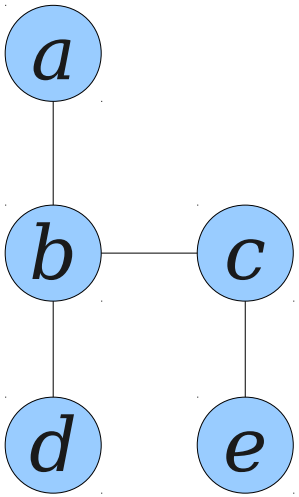
An Initial Idea: Linked Lists



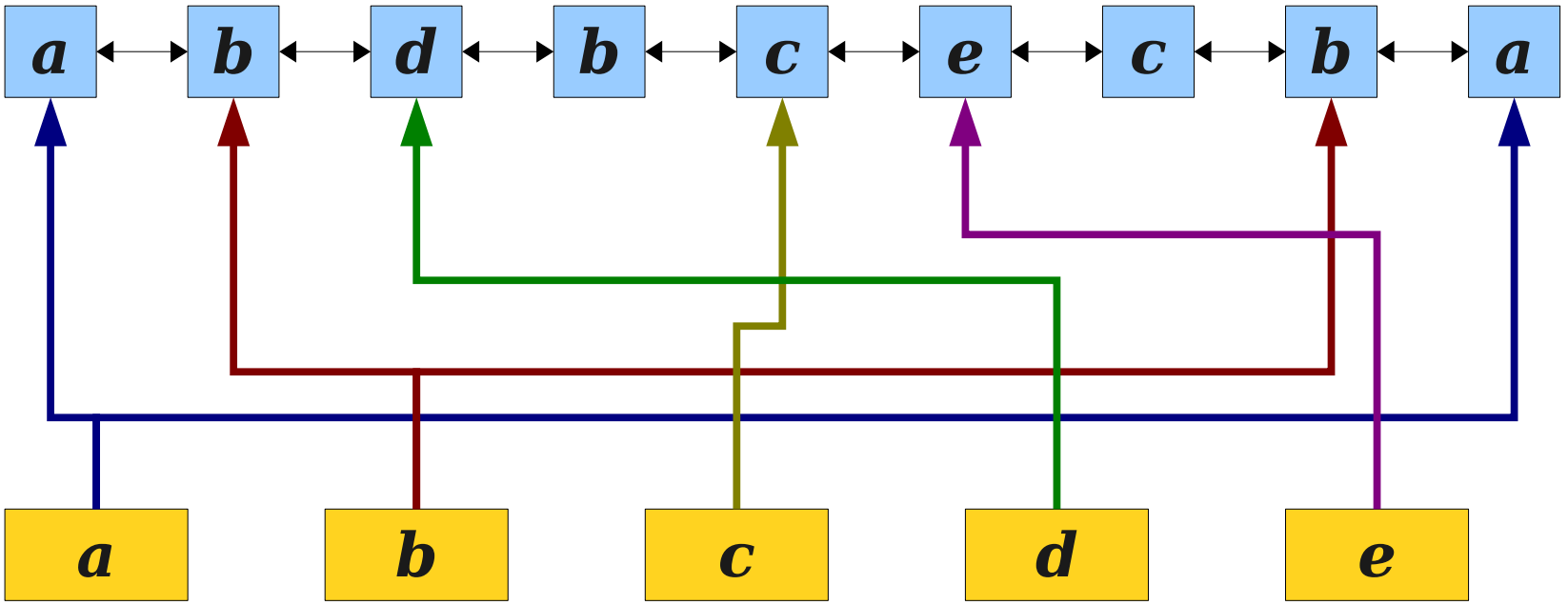
a b d b c e c b a



An Initial Idea: Linked Lists

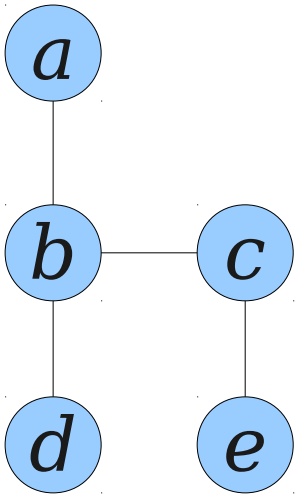


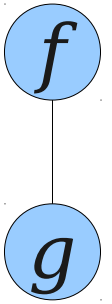
a b d b c e c b a



An Initial Idea: Linked Lists

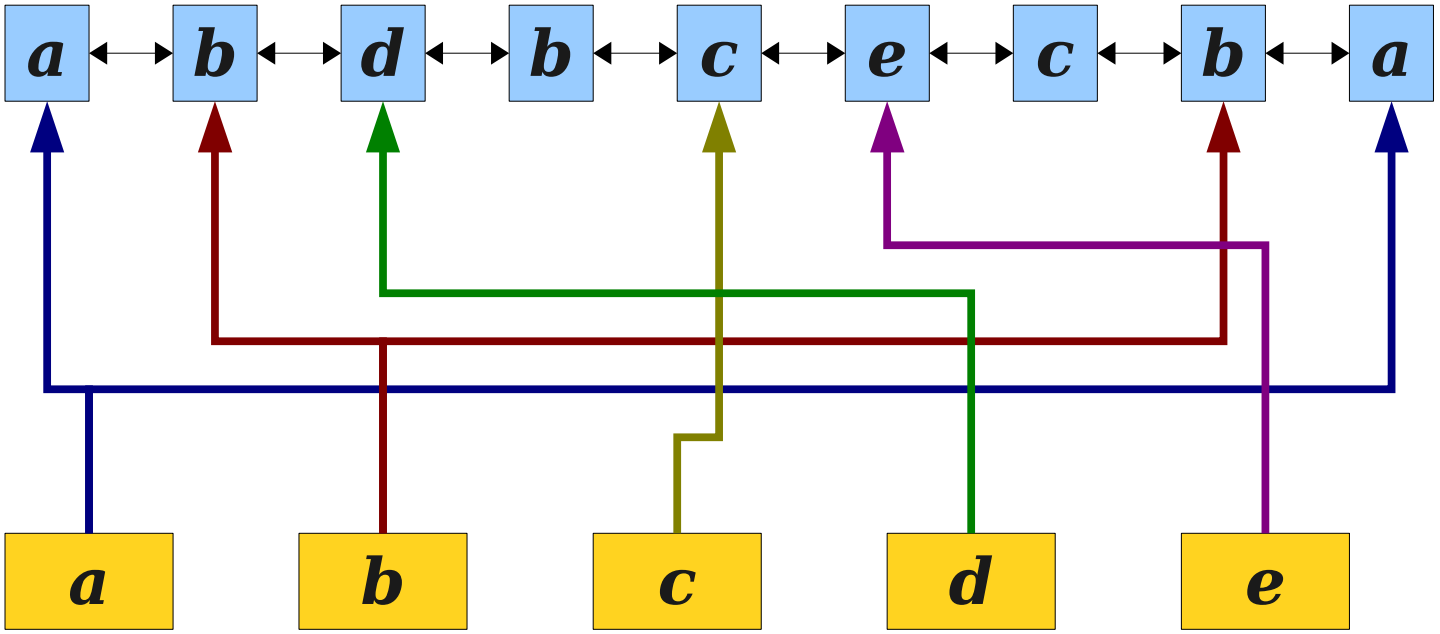
- Each split or concatenate takes time O(1).
- The first and last copy of a node can be identified in time O(1).
- A new copy of a node can be appended to the end of the sequence in time O(1).
- A redundant copy of a node can be deleted in time O(1).
- Everything sounds great!
- **Question:** How do you test for connectivity?

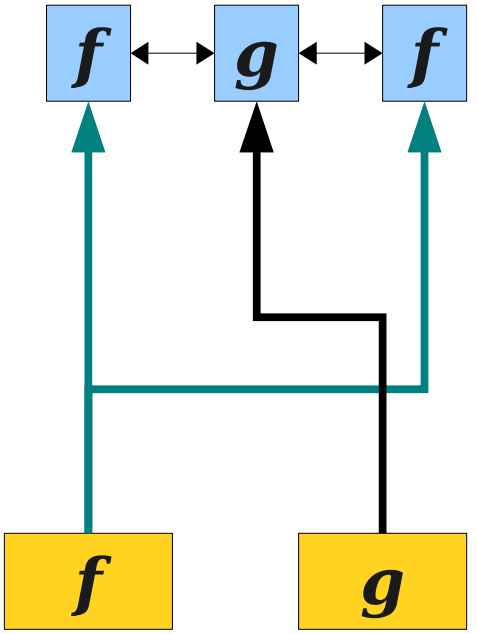




a b d b c e c b a

f g f



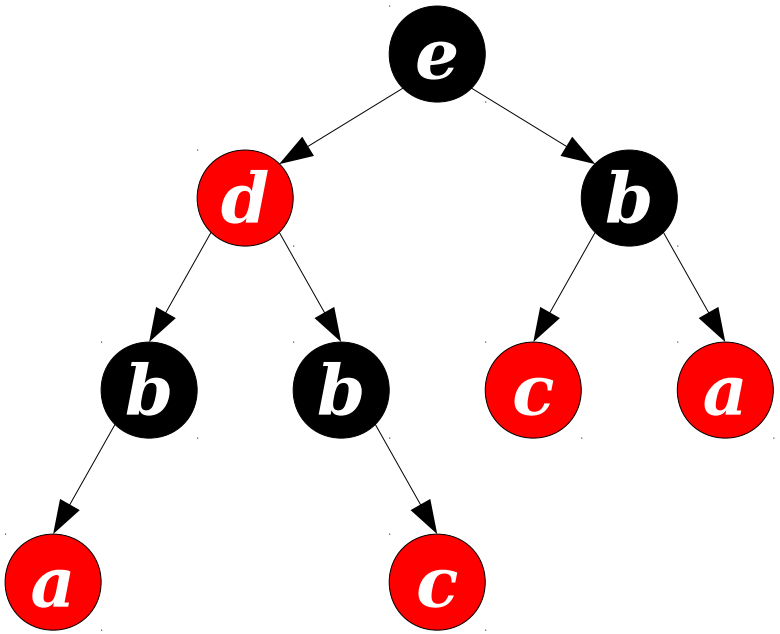


The Story So Far

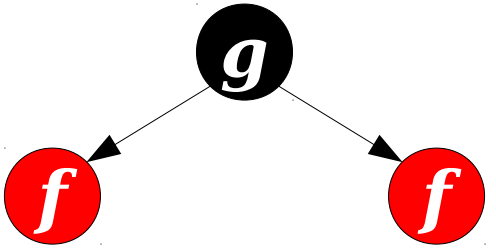
- Euler tours give a simple, flexible encoding of tree structures.
- Using doubly-linked lists, concatenation and splits take time O(1) each, but testing connectivity takes time $\Theta(n)$ in the worst-case.
- Can we do better?

Using Balanced Trees

- **Claim:** It is possible to represent sequences of elements balanced binary trees.
- These are not binary *search* trees. We're using the *shape* of a red/black tree to ensure balance.



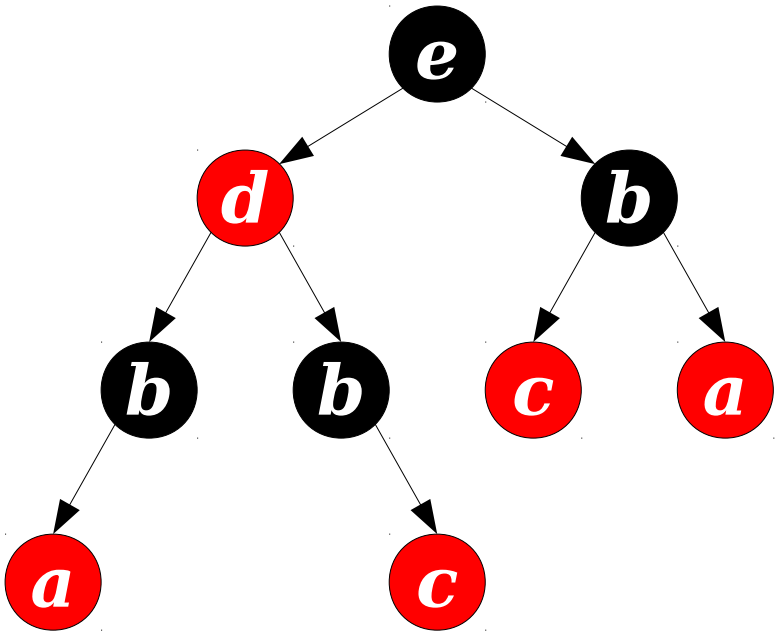
a b d b c e c b a



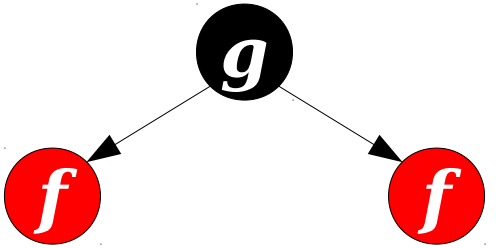
f g f

Using Balanced Trees

- **Observation 1:** Can still store pointers to the first and last occurrence of each tree node.



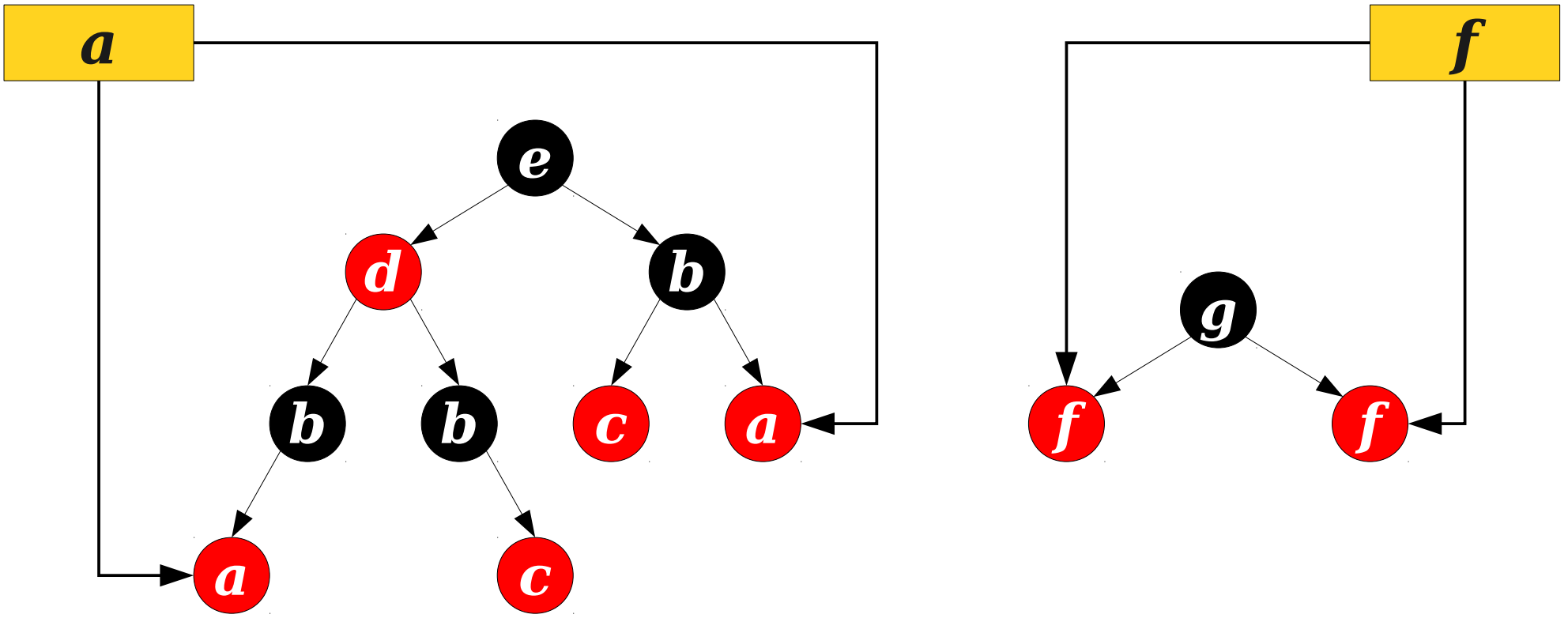
a b d b c e c b a



f g f

Using Balanced Trees

- **Observation 1:** Can still store pointers to the first and last occurrence of each tree node.

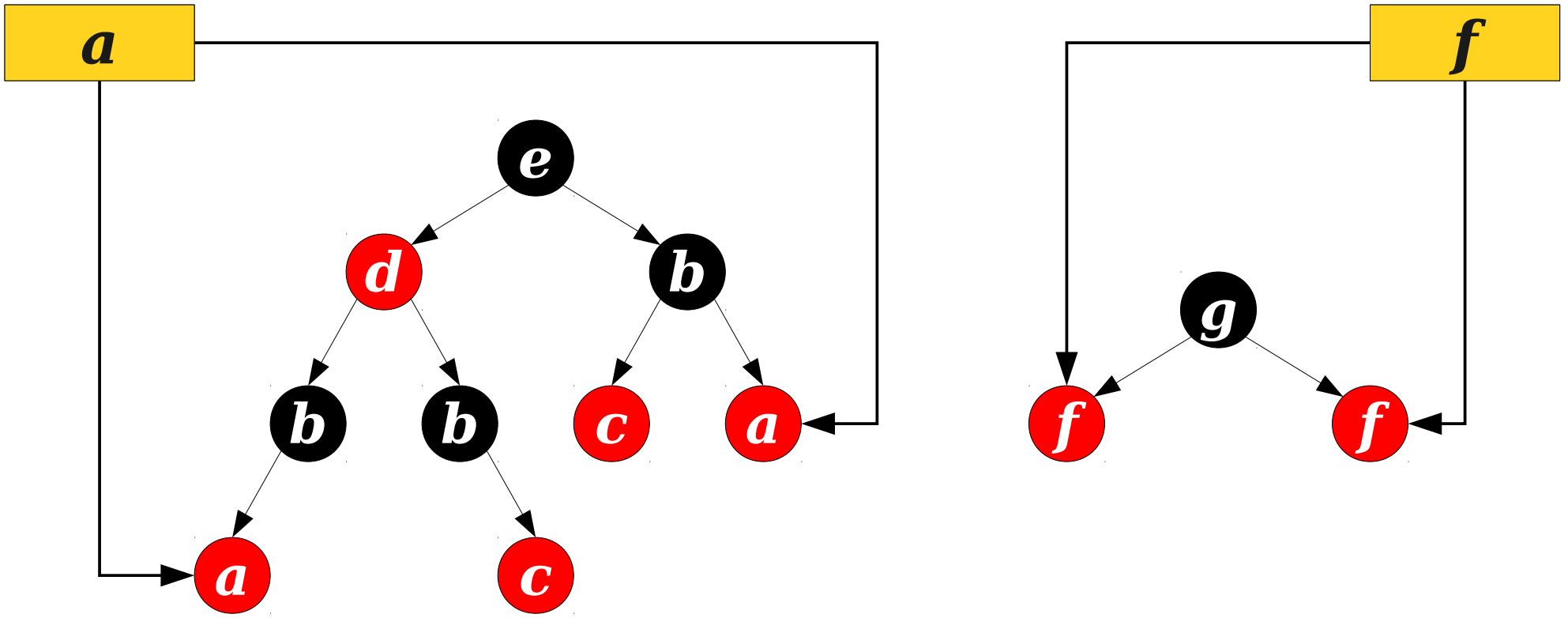


a b d b c e c b a

f g f

Using Balanced Trees

- **Observation 2:** If nodes store pointers to their parents, can answer *is-connected*(*u*, *v*) in time O(log *n*) by seeing if *u* and *v* are in the same tree.

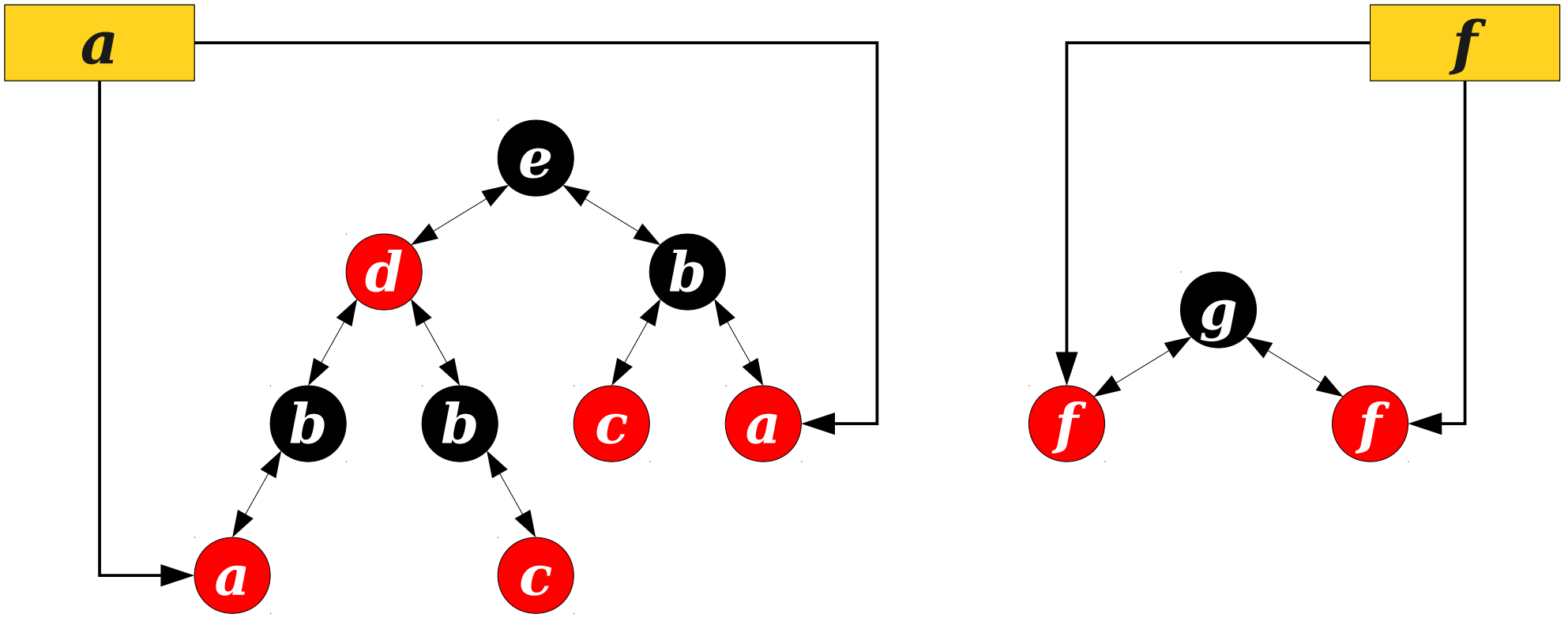


a b d b c e c b a

f g f

Using Balanced Trees

- **Observation 2:** If nodes store pointers to their parents, can answer *is-connected*(u, v) in time O(log n) by seeing if u and v are in the same tree.

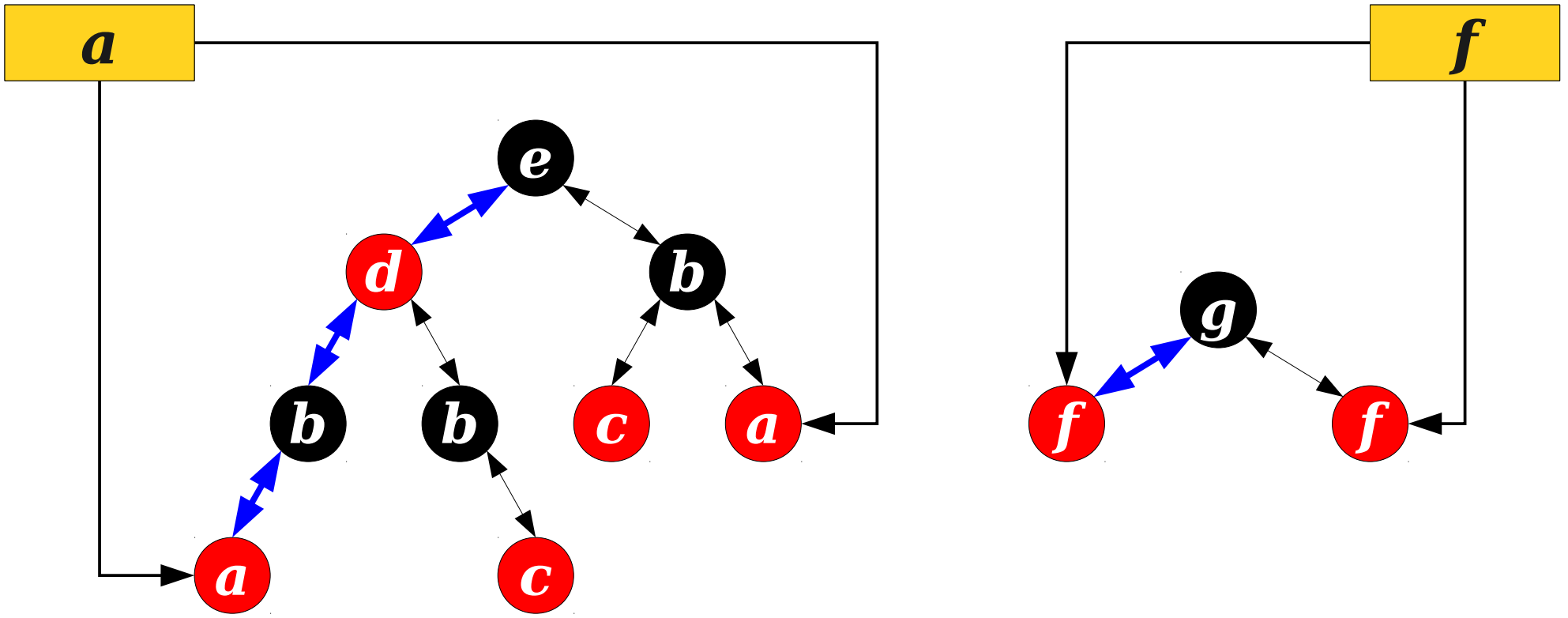


a b d b c e c b a

f g f

Using Balanced Trees

- **Observation 2:** If nodes store pointers to their parents, can answer *is-connected*(u, v) in time O(log n) by seeing if u and v are in the same tree.

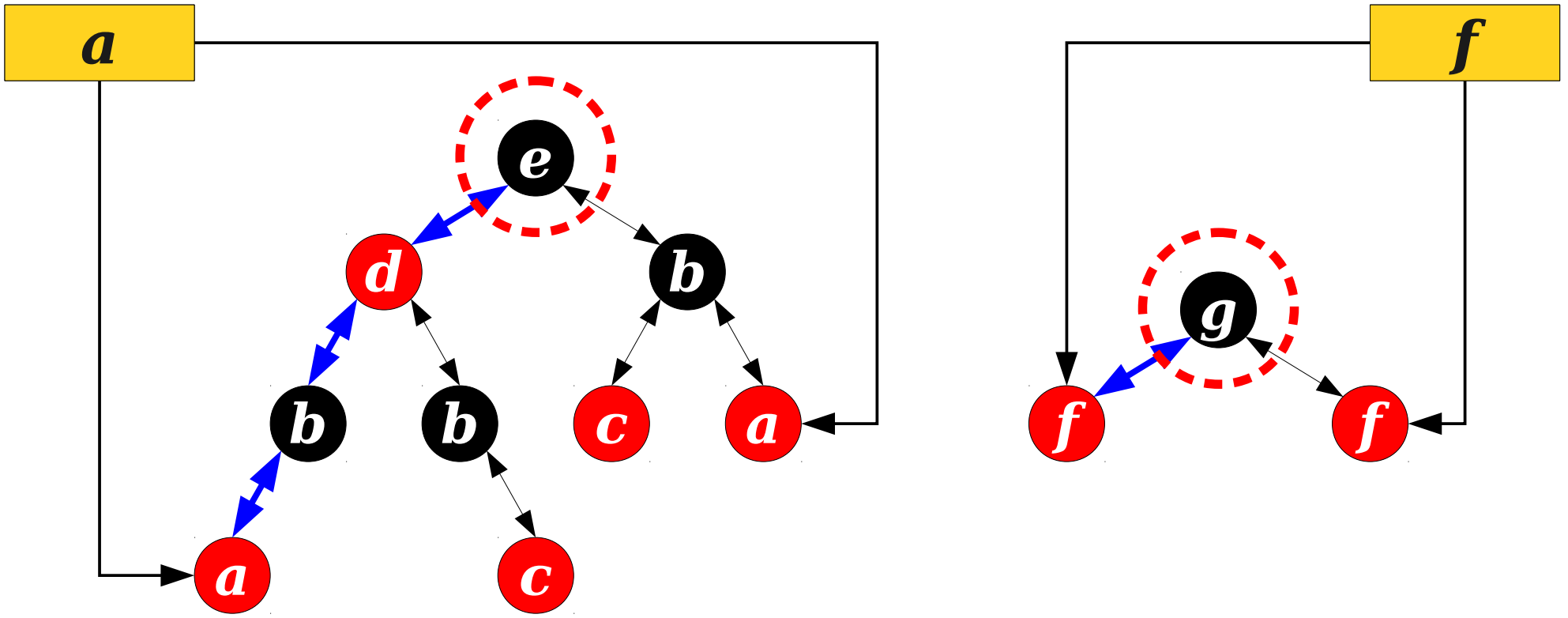


a b d b c e c b a

f g f

Using Balanced Trees

- **Observation 2:** If nodes store pointers to their parents, can answer *is-connected*(u, v) in time O(log n) by seeing if u and v are in the same tree.

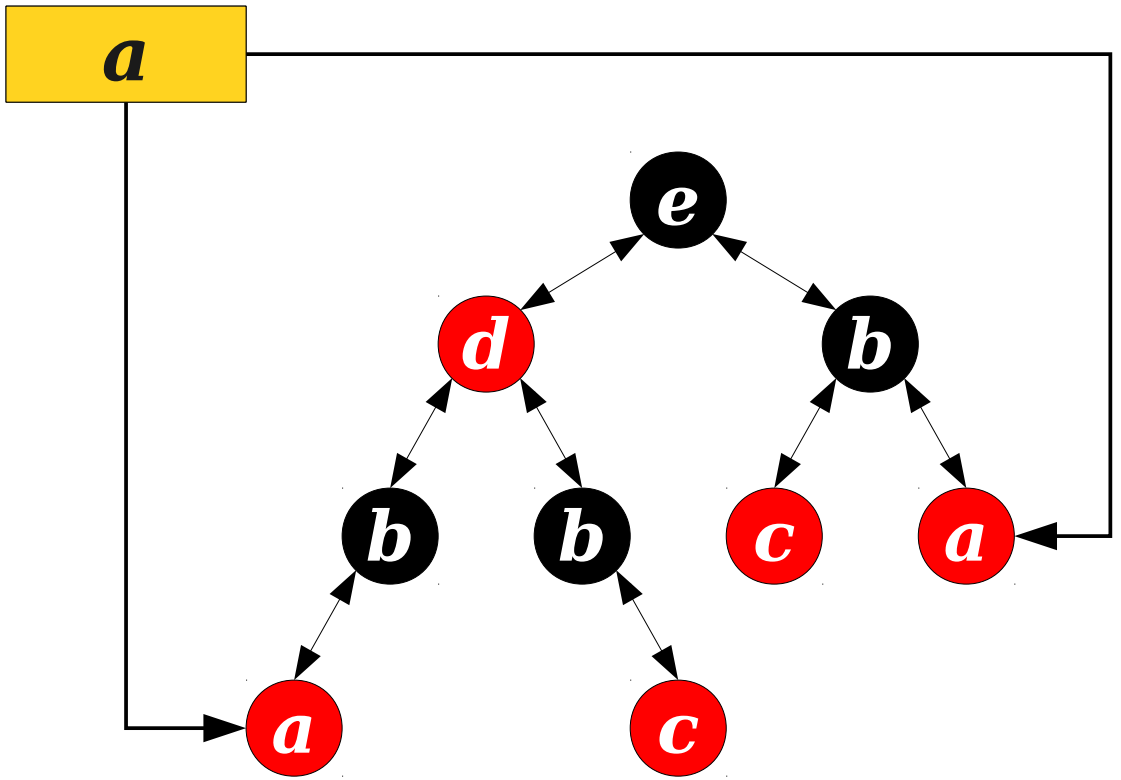


a b d b c e c b a

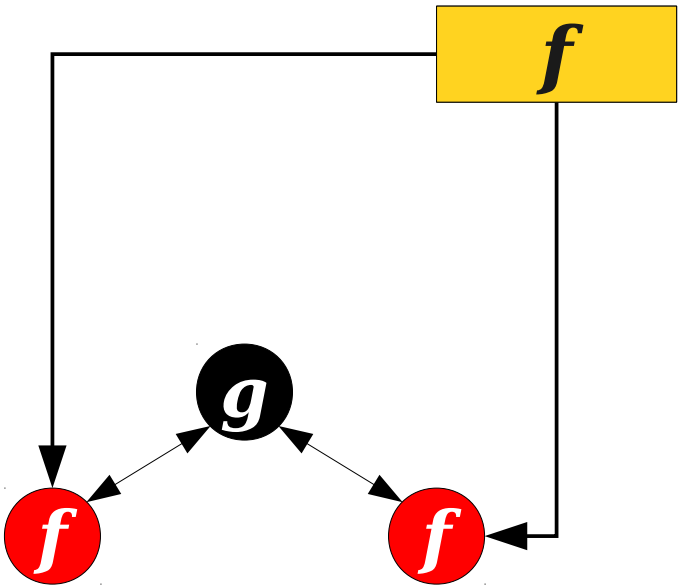
f g f

Using Balanced Trees

- **Observation 3:** Red/black trees can be split and joined in time O(log *n*) each.



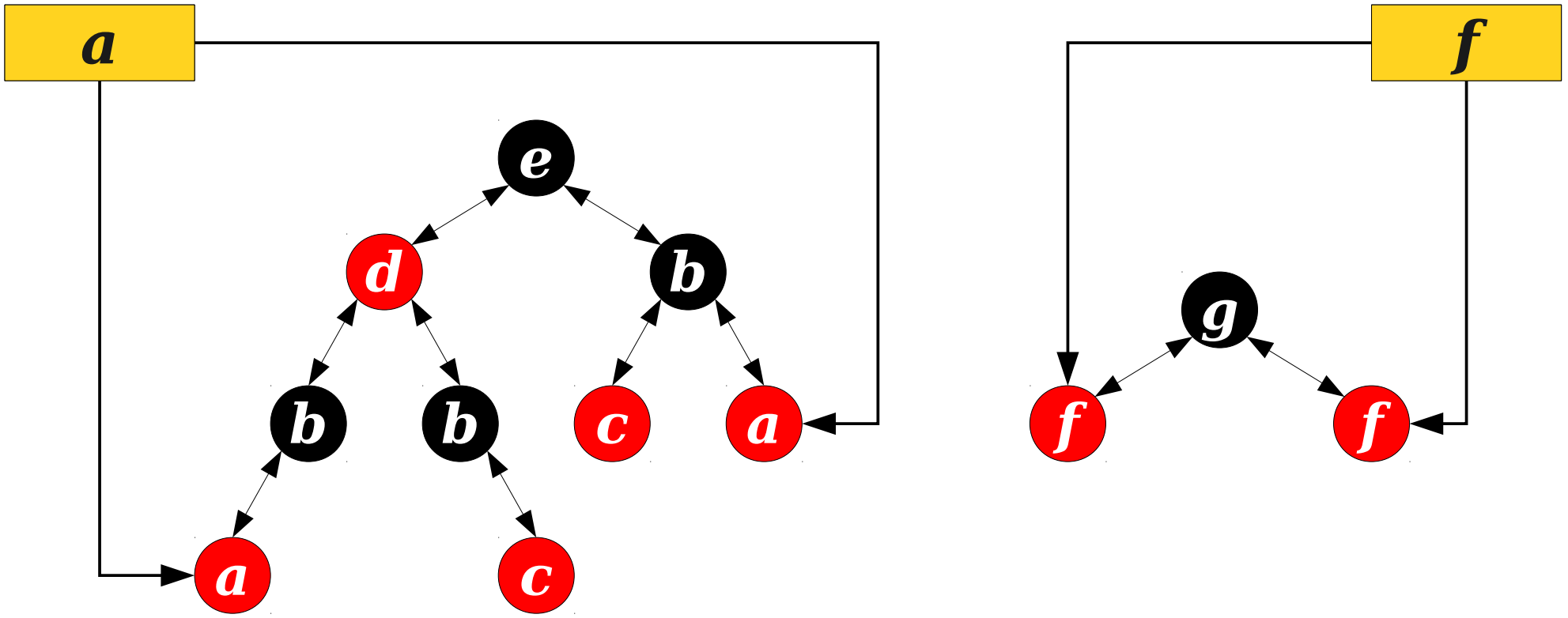
a b d b c e c b a



f g f

Using Balanced Trees

- **Observation 3:** Red/black trees can be split and joined in time O(log n) each.

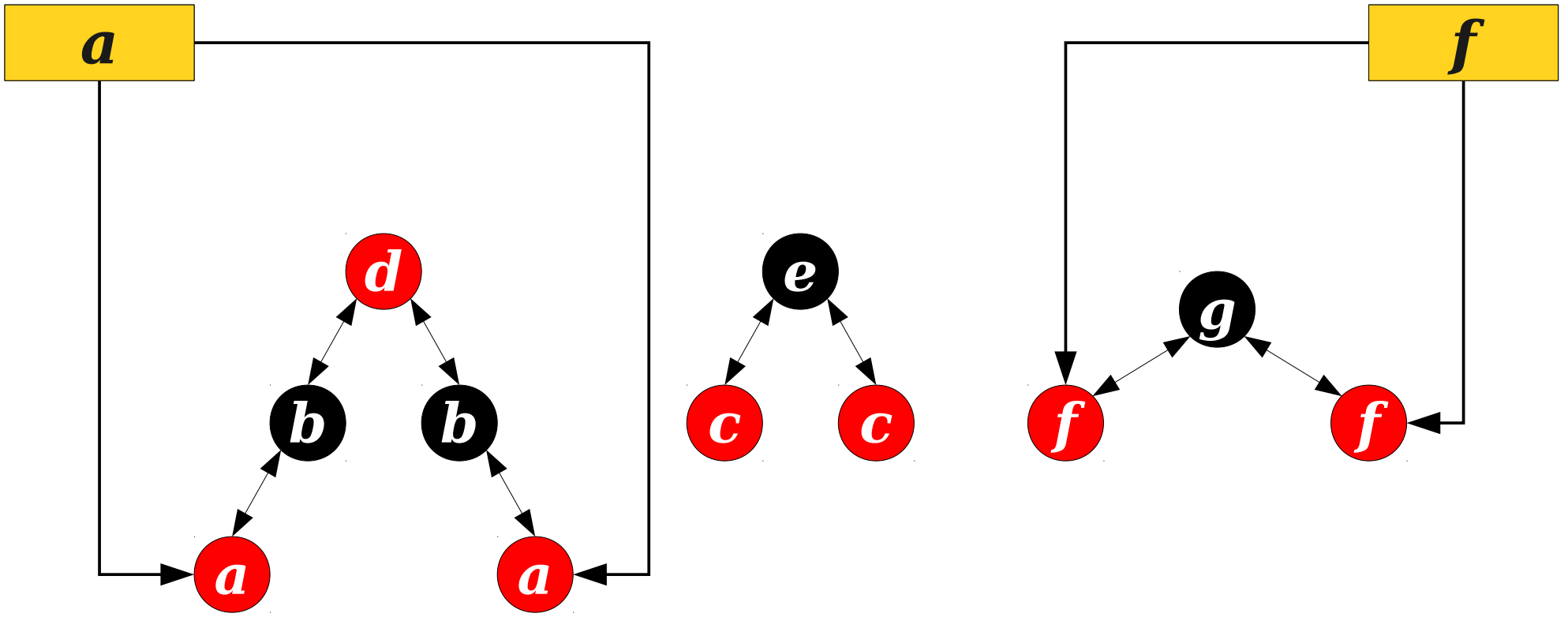


a b d b c e c b a

f g f

Using Balanced Trees

- **Observation 3:** Red/black trees can be split and joined in time O(log n) each.



Euler Tour Trees

- The data structure:
  - Represent each tree as an Euler tour.
  - Store those sequences as balanced binary trees.
  - Each node in the original forest stores a pointer to its first and last occurrence.
  - Each node in the balanced trees stores a pointer to its parent.
- *link*, *cut*, and *is-connected* queries take time only **O(log *n*)** each.

Augmented Euler Tour Trees

- Euler tour trees are layered atop red/black trees.
- We can therefore augment Euler tour trees to store additional information about each tree.
- Examples:
  - Keep track of the minimum-weight or maximum-weight each in each tree.
  - Keep track of the number of nodes in each tree.
- We'll use this both later today and in a few weeks when we talk about dynamic graph connectivity.

Time-Out for Announcements!

CS Casual Dinner

- Casual dinner for women studying computer science **tomorrow night** on the Gates Fifth Floor, 6PM – 8PM.
- Everyone is welcome!

OH Update

- This week only: I'll be covering the Thursday office hours and the TAs will cover today's OH right after class.
- Thursday OH rooms TBA.
- We will probably be changing Thursday OH times in future weeks; details later.

Your Questions

“Could you recommend some (other) awesome CS classes (preferably in Theory, but doesn't have to be)?”

My Recommendations

CS143 (Compilers)
CS343 (Dynamic Analysis)
CS154 (Automata and Complexity Theory)
CS181 (Computers, Ethics, and Public Policy)
CS261 / CS361B (Advanced Algorithms)
CS224W (Social Network Analysis)

“What percentage of student submitted the first problem set individually versus in pairs?”

I'm going to hold off on announcing this until Wednesday when we get an estimate for Problem Set Two. I expect there will be an increase.

“I've never written a Makefile before. Where can I find the resources to learn how to make one for Assignment 2?”

Check out the CS107 resources page:

http://cs107.stanford.edu/resources.html

“What's the best way to impress everyone at a cocktail party?”

Balancing a fork and a spoon on a toothpick, then setting the toothpick on fire. Either that or walking through a sheet of paper.

Everyone will find you impressive if you talk about the history of food and nutrition. Bonus points for throwing out the words “nixtamalization” and “orangery.”

Or just be nice and a good listener. ☺

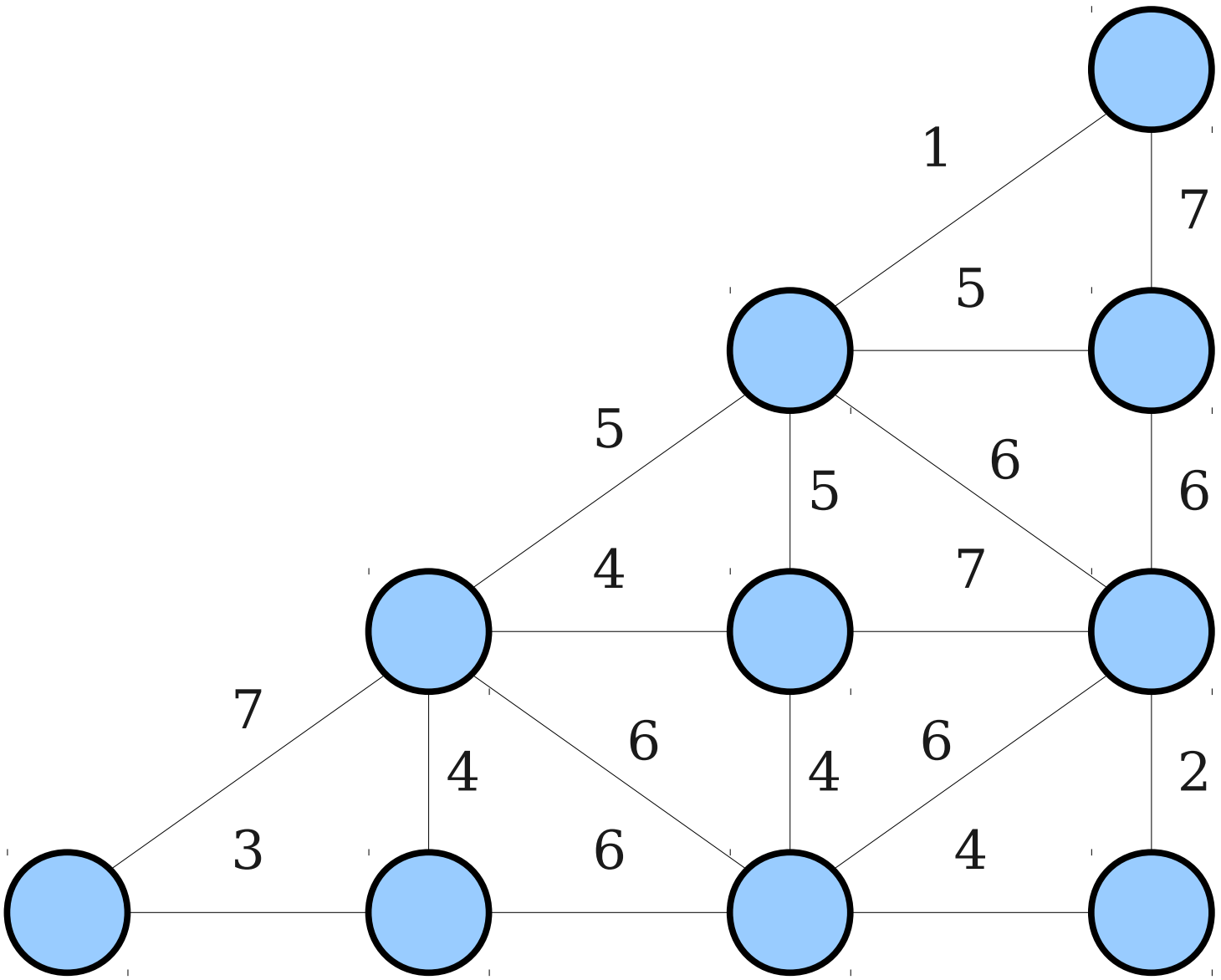
Back to CS166!

Application: **Bottleneck Edge Queries**

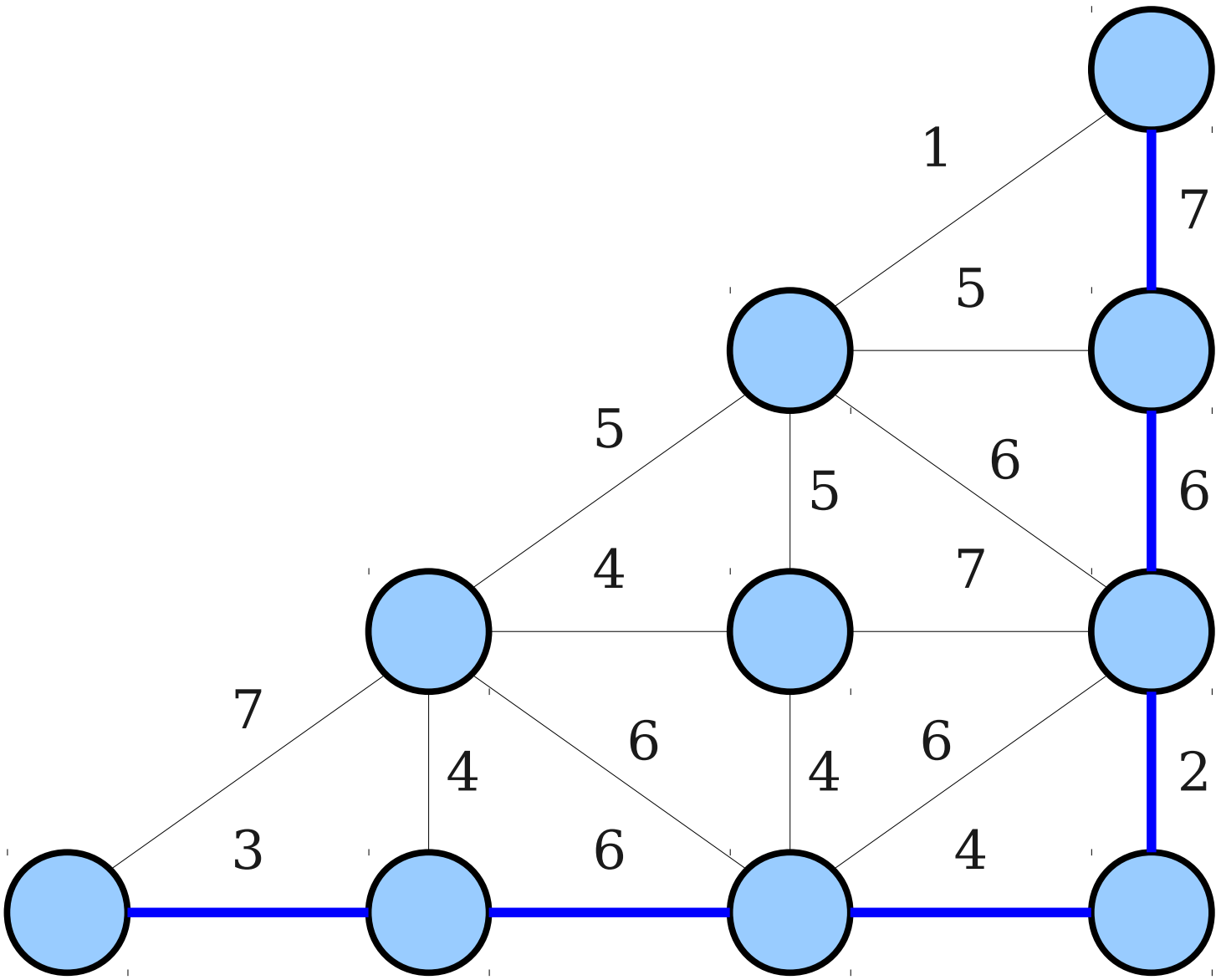
Bottleneck Paths

- Let G be an undirected graph where each edge has an associated, positive, real-valued capacity.
- The **bottleneck edge** on a path is the edge on the path with minimum capacity.
- **Challenge:** Find the maximum-bottleneck path between nodes u and v.

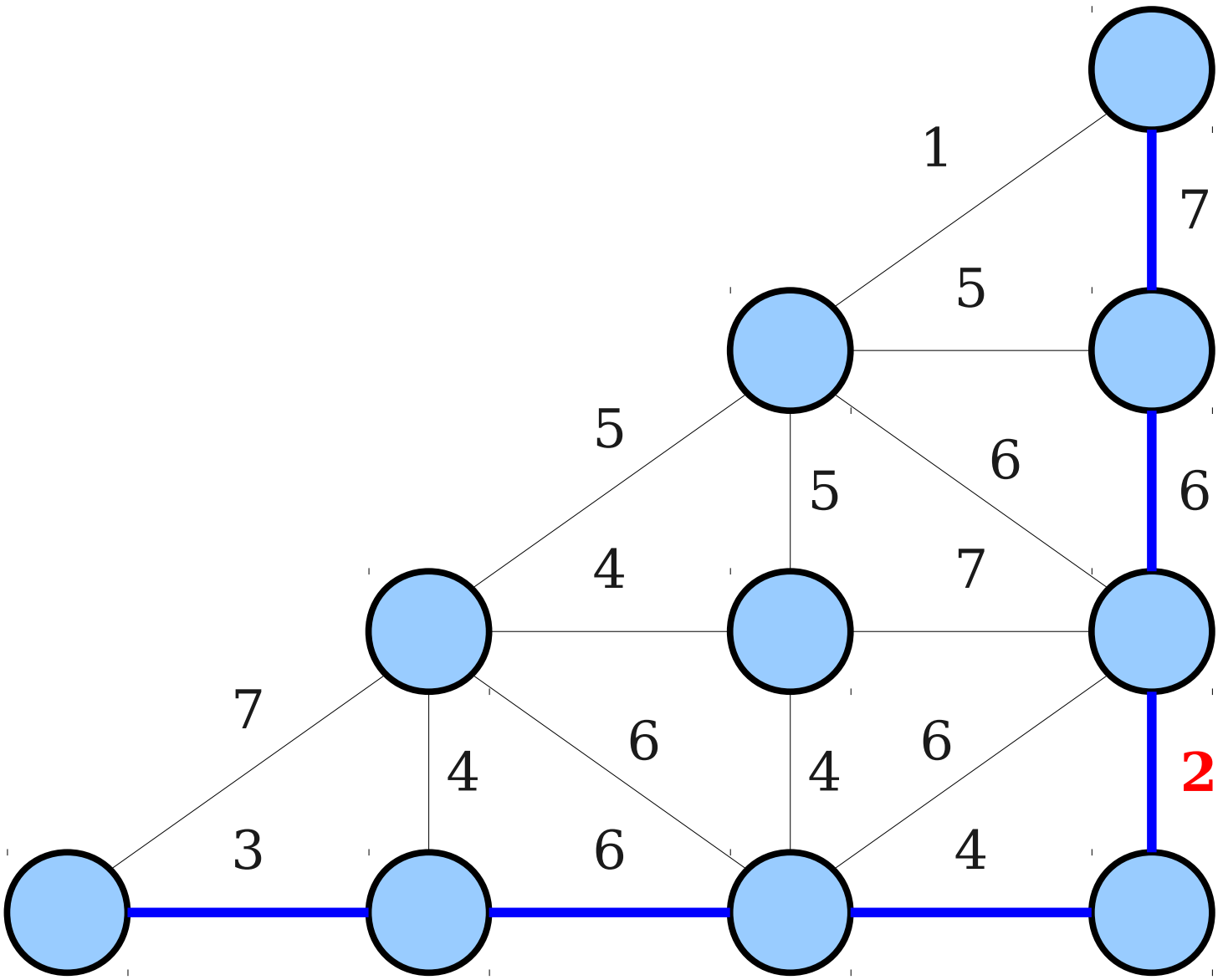
Bottleneck Paths



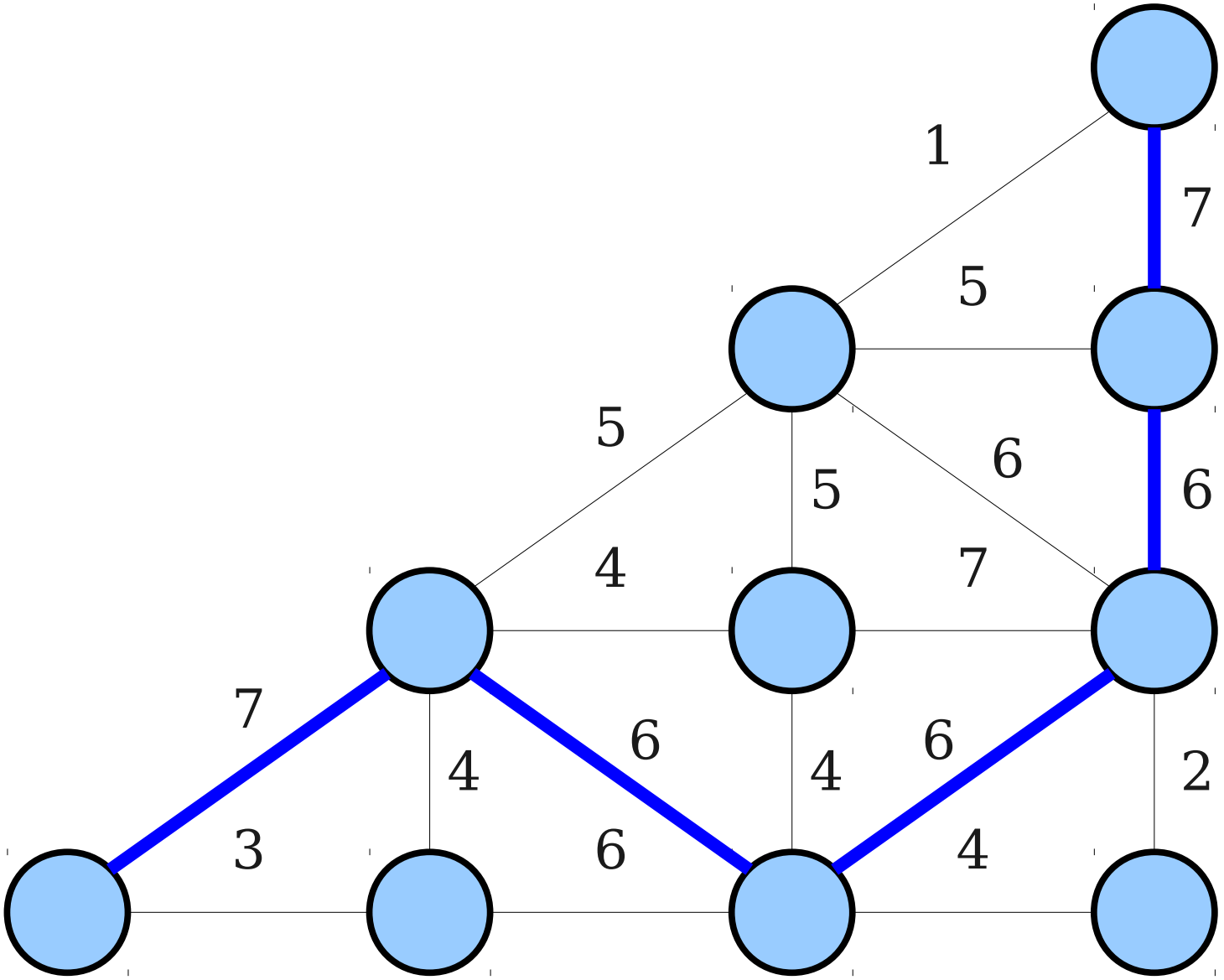
Bottleneck Paths



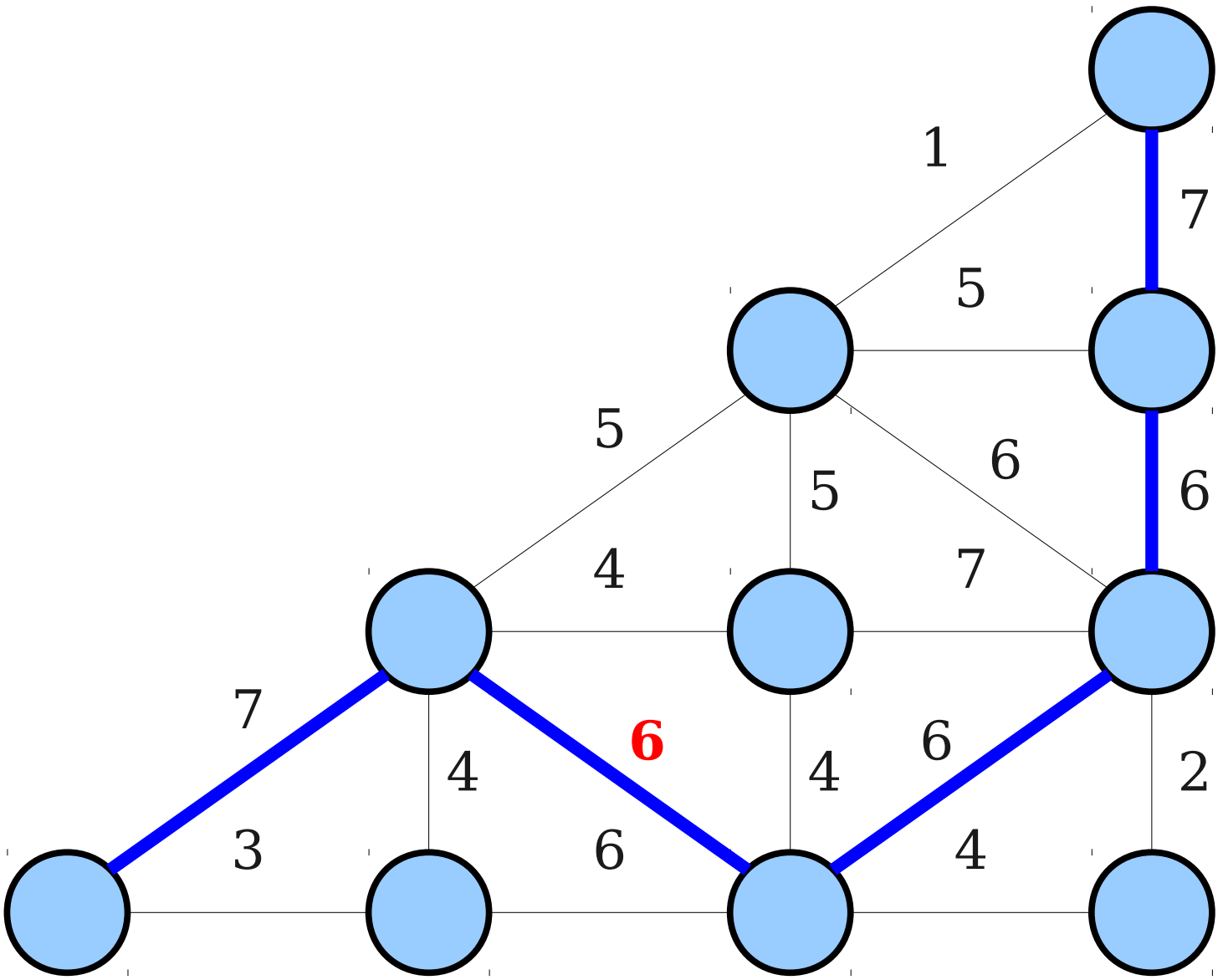
Bottleneck Paths



Bottleneck Paths



Bottleneck Paths



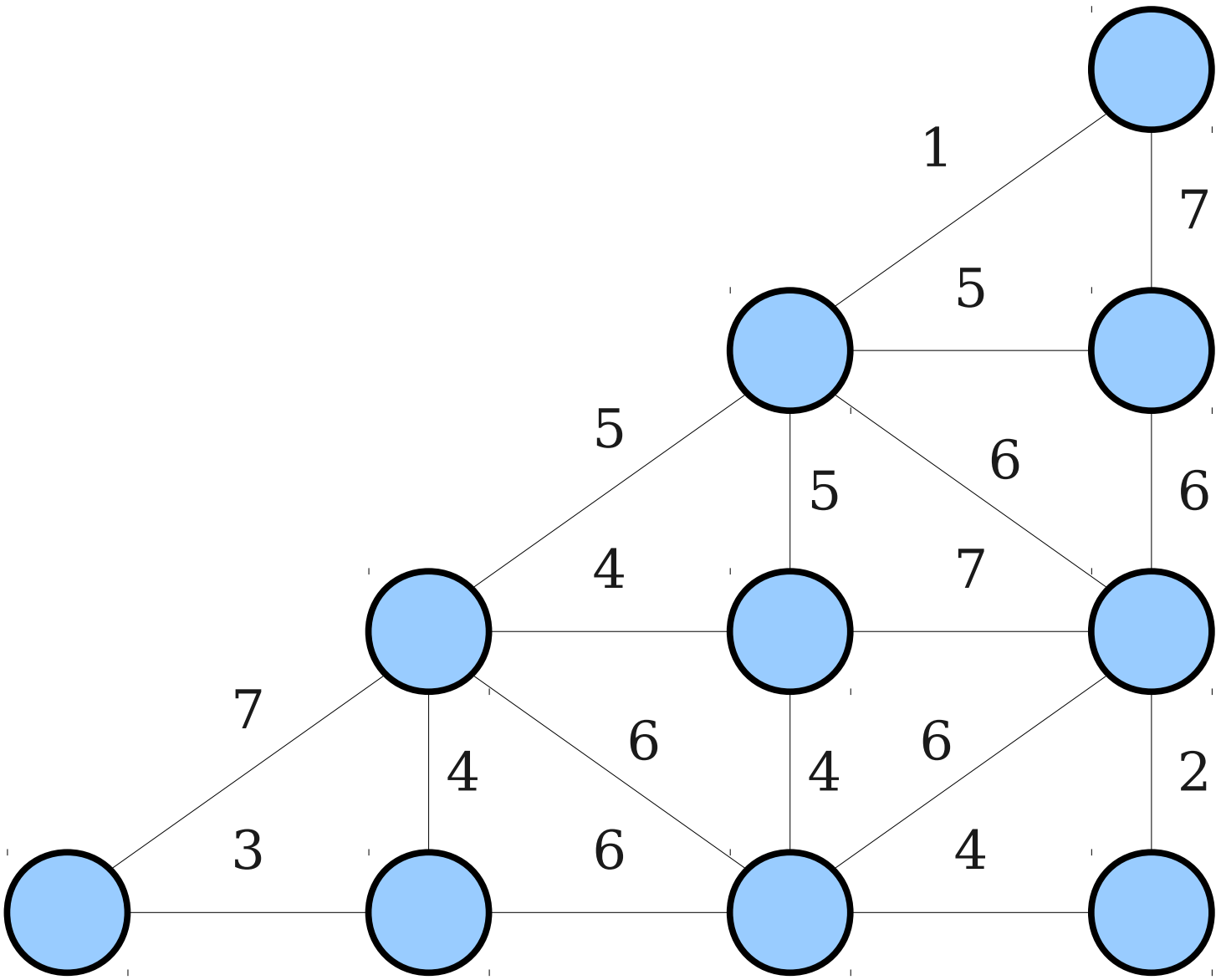
Bottleneck Edge Queries

- The **bottleneck edge query problem** is the following:

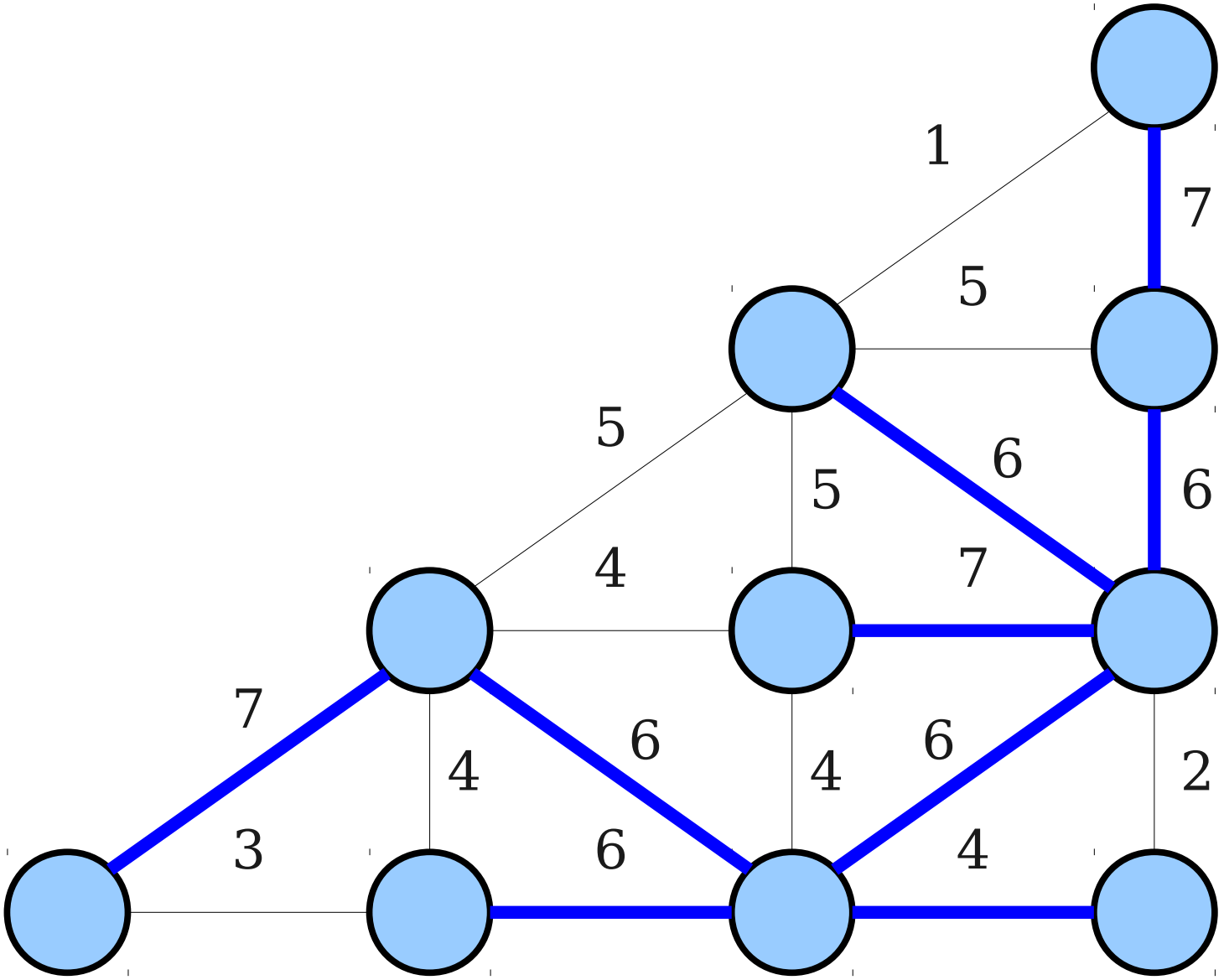
  Given an undirected graph G with edge capacities, preprocess the graph so that the bottleneck edge between any pair of nodes u and v can be found efficiently.

- Applications in network routing, shipping, and maximum flow algorithms.
- Like RMQ, could solve by precomputing the bottleneck paths between all pairs of nodes.
- ***Can we do better?***

Bottleneck Paths



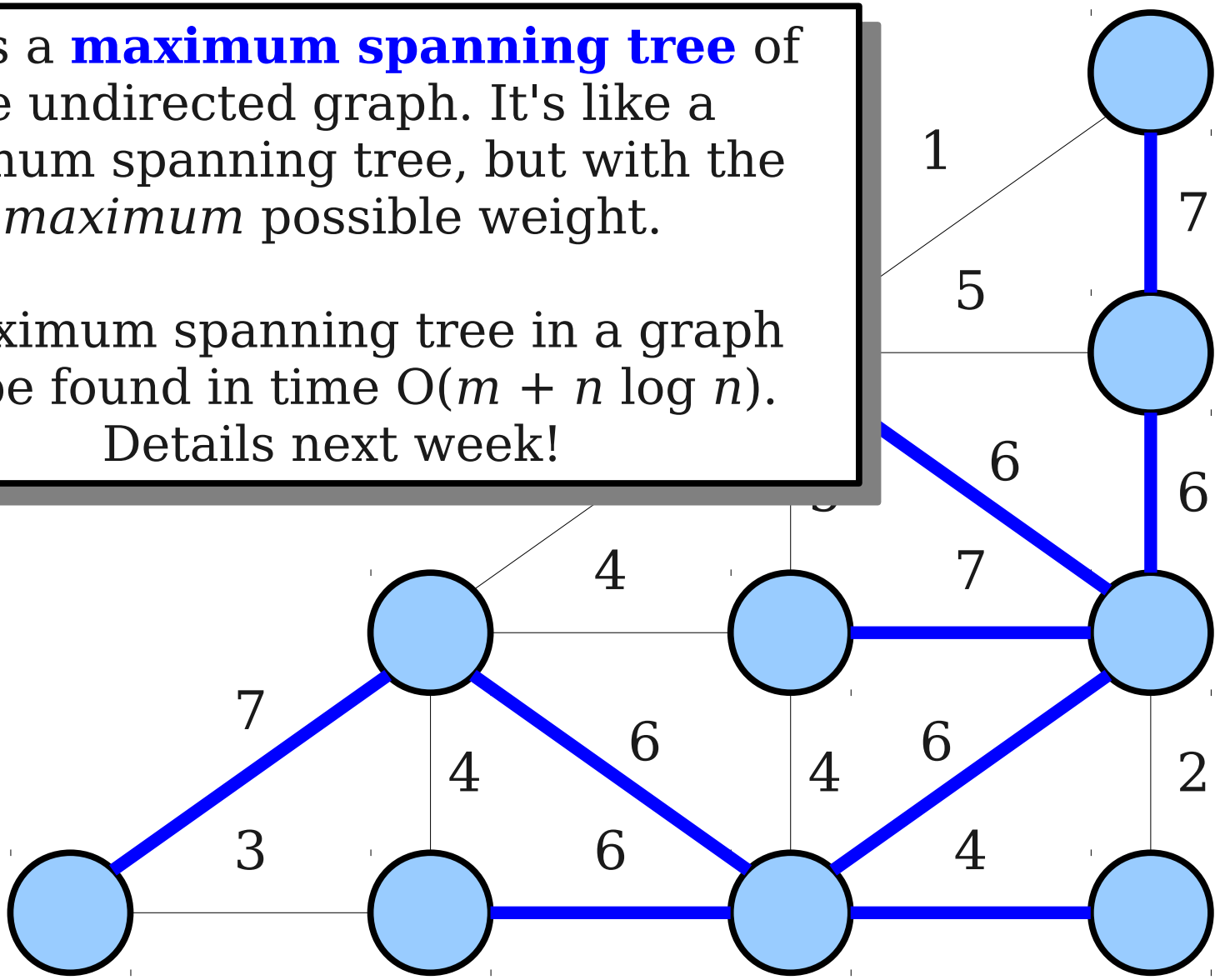
Bottleneck Paths



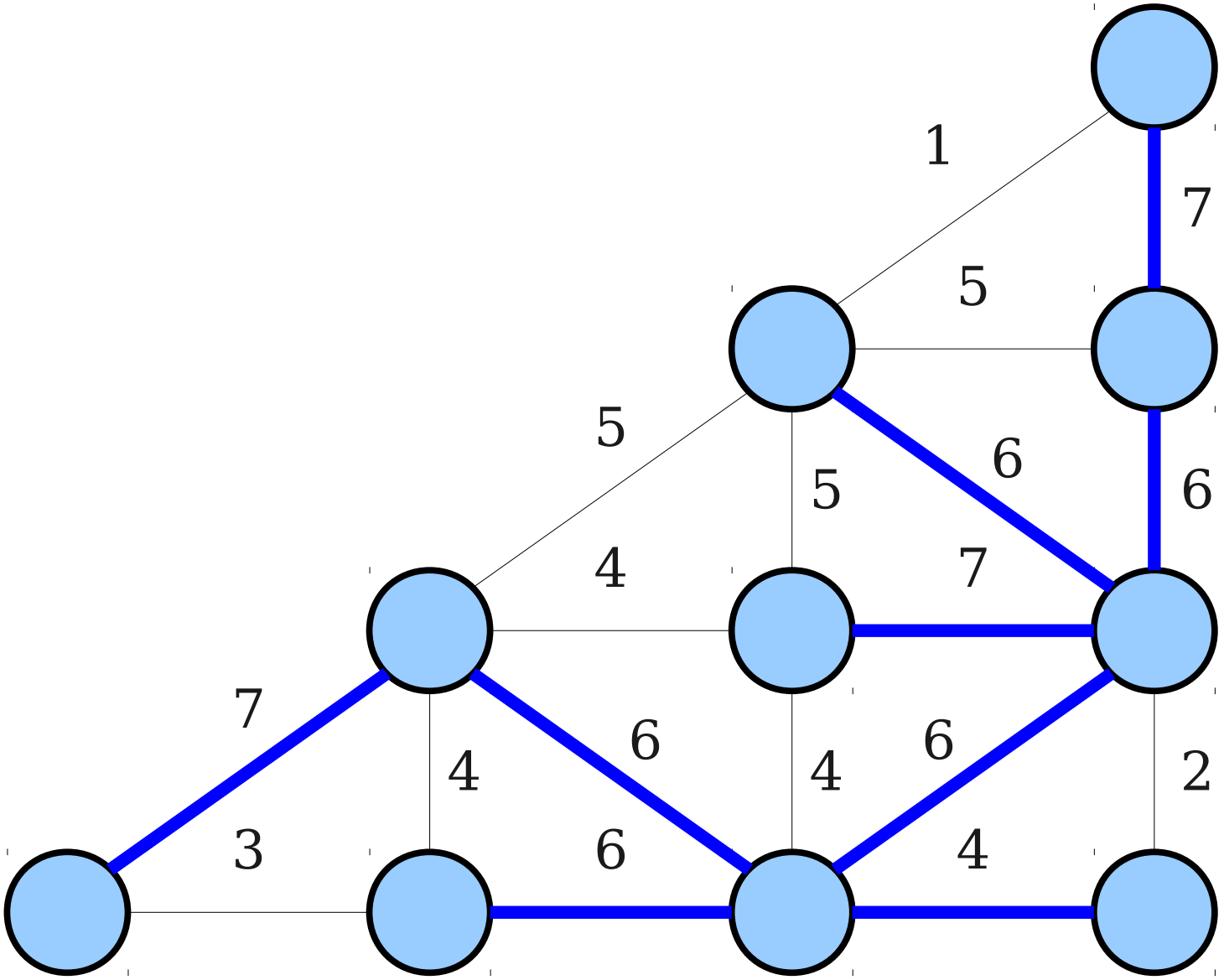
Bottleneck Paths

This is a **maximum spanning tree** of the undirected graph. It's like a minimum spanning tree, but with the *maximum* possible weight.

A maximum spanning tree in a graph can be found in time $O(m + n \log n)$. Details next week!

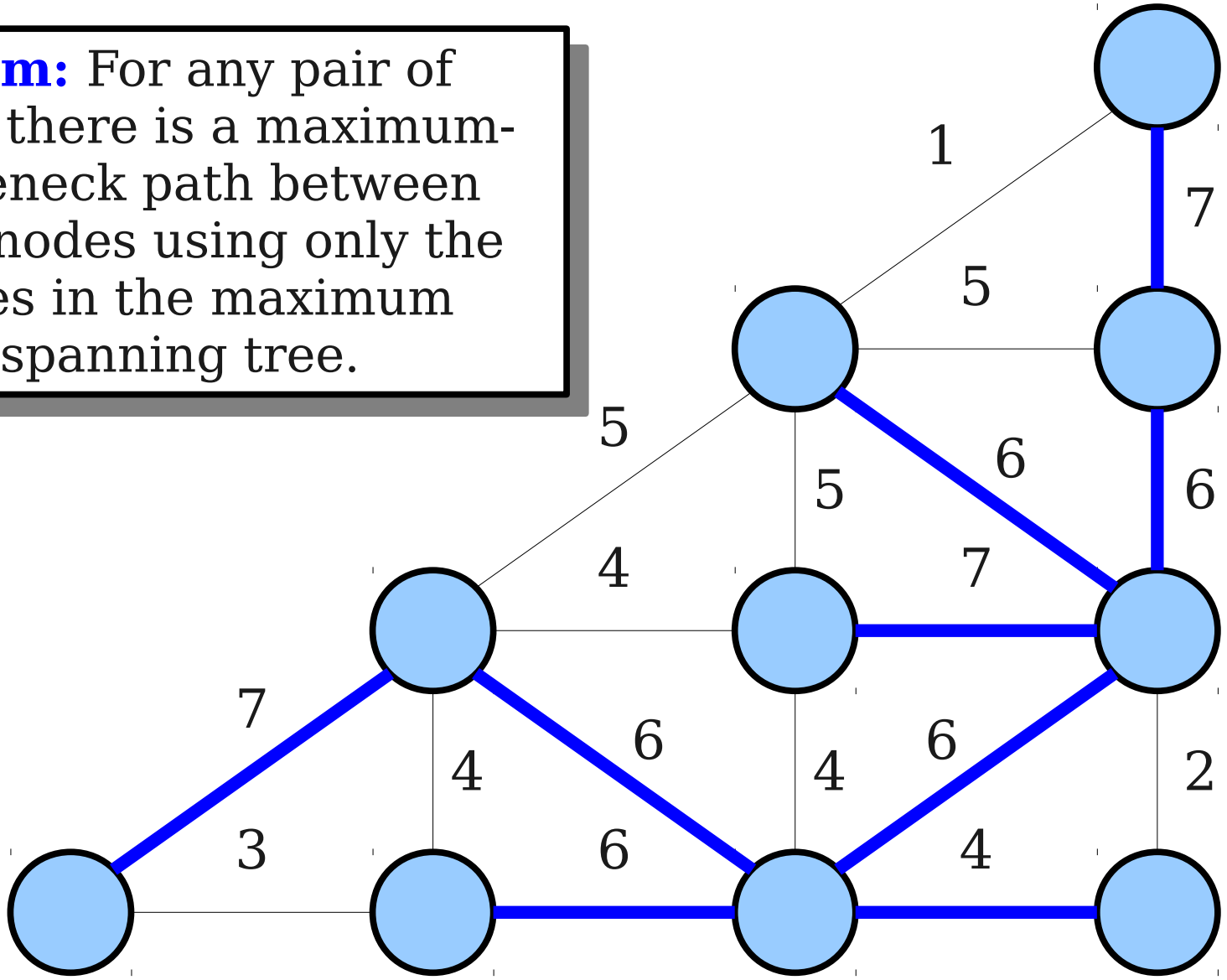


Bottleneck Paths

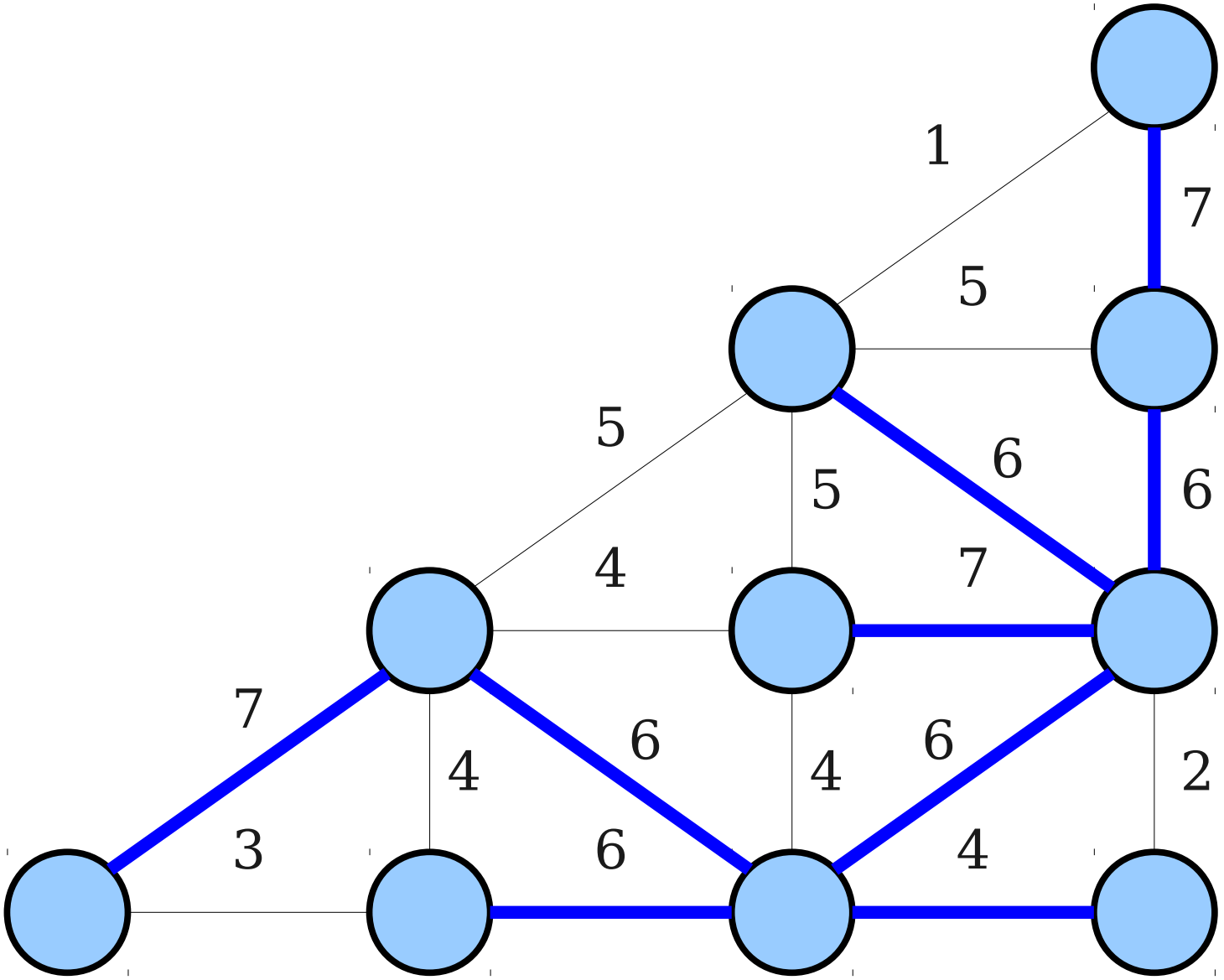


Bottleneck Paths

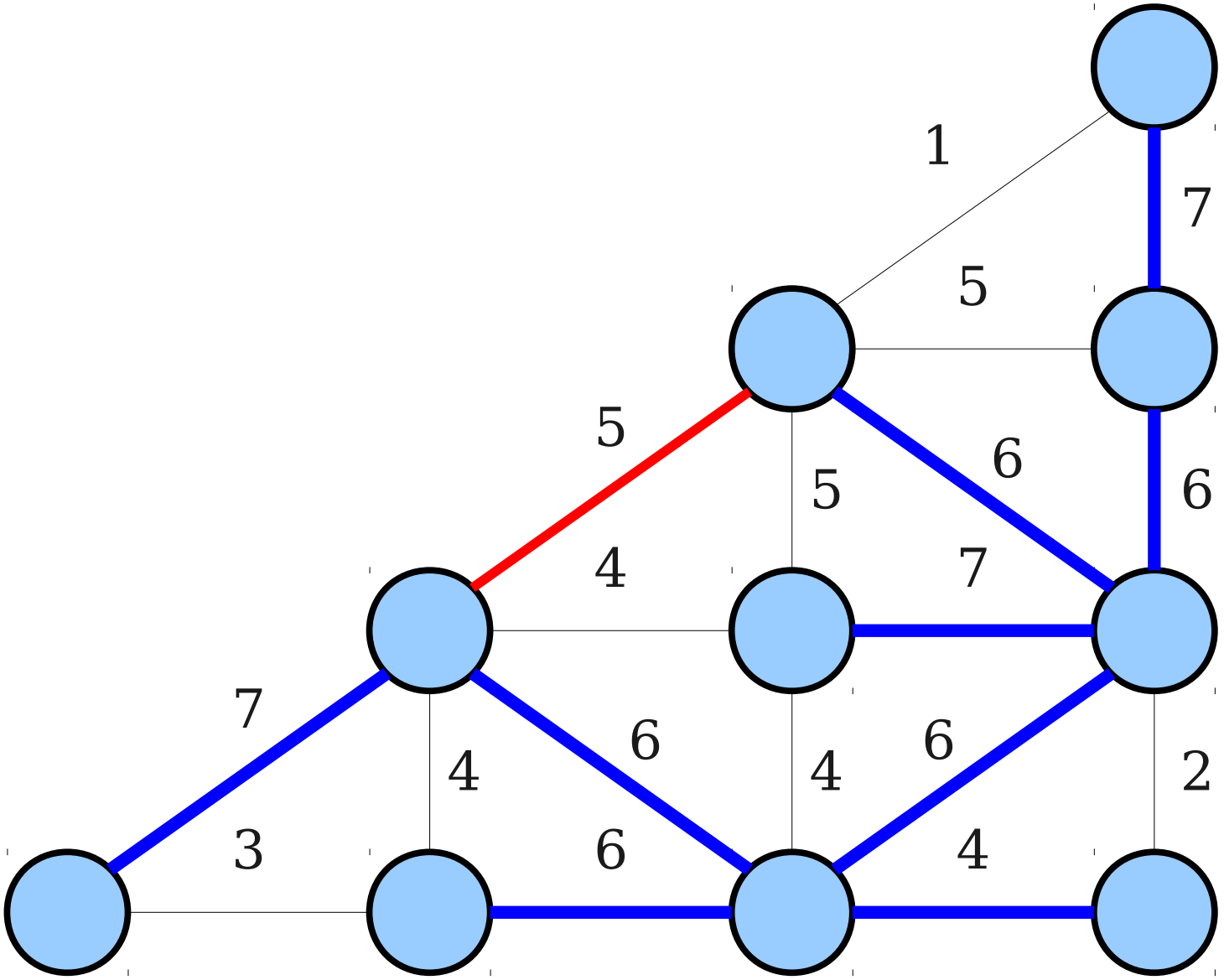
Claim: For any pair of nodes, there is a maximum-bottleneck path between those nodes using only the edges in the maximum spanning tree.



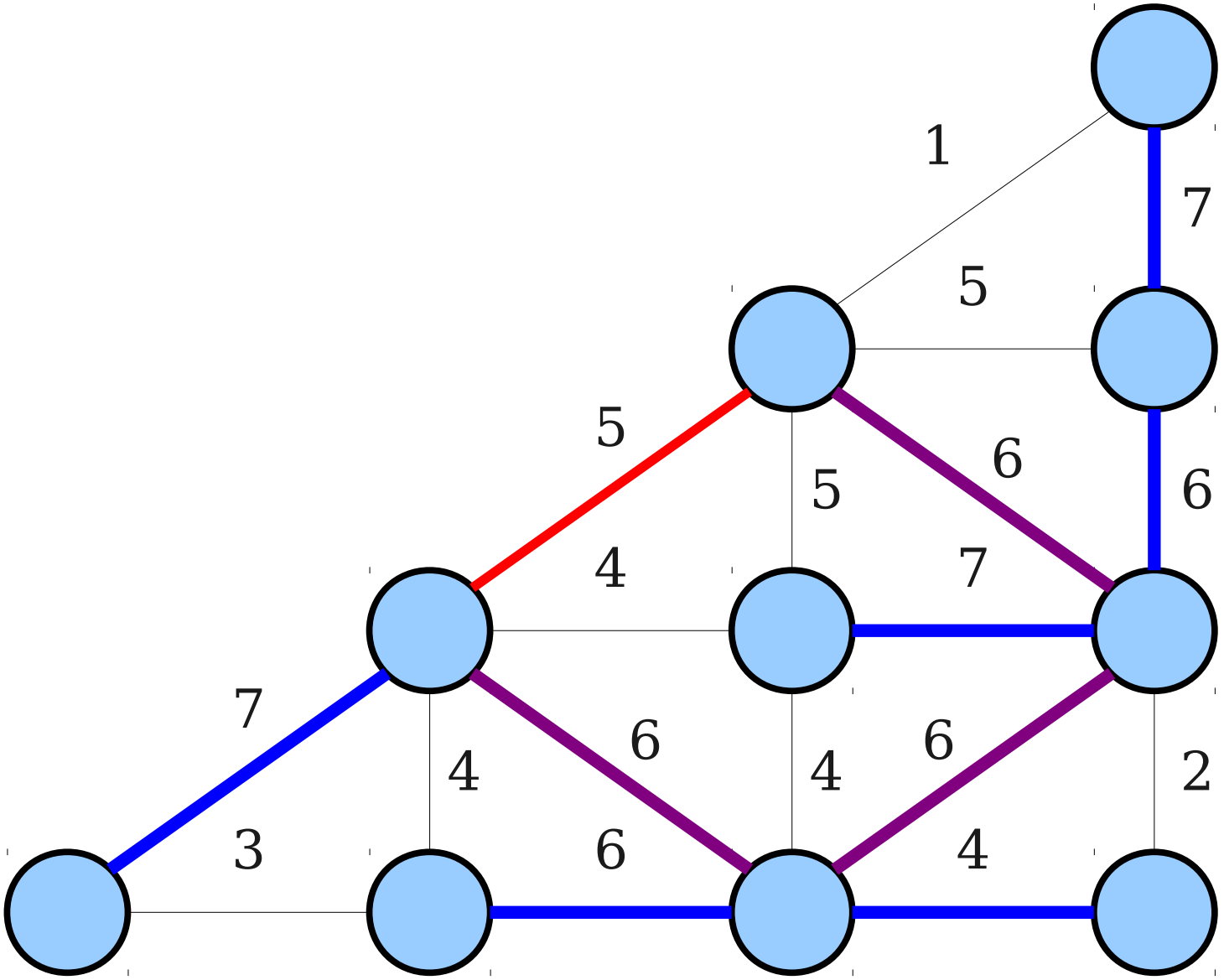
Bottleneck Paths



Bottleneck Paths



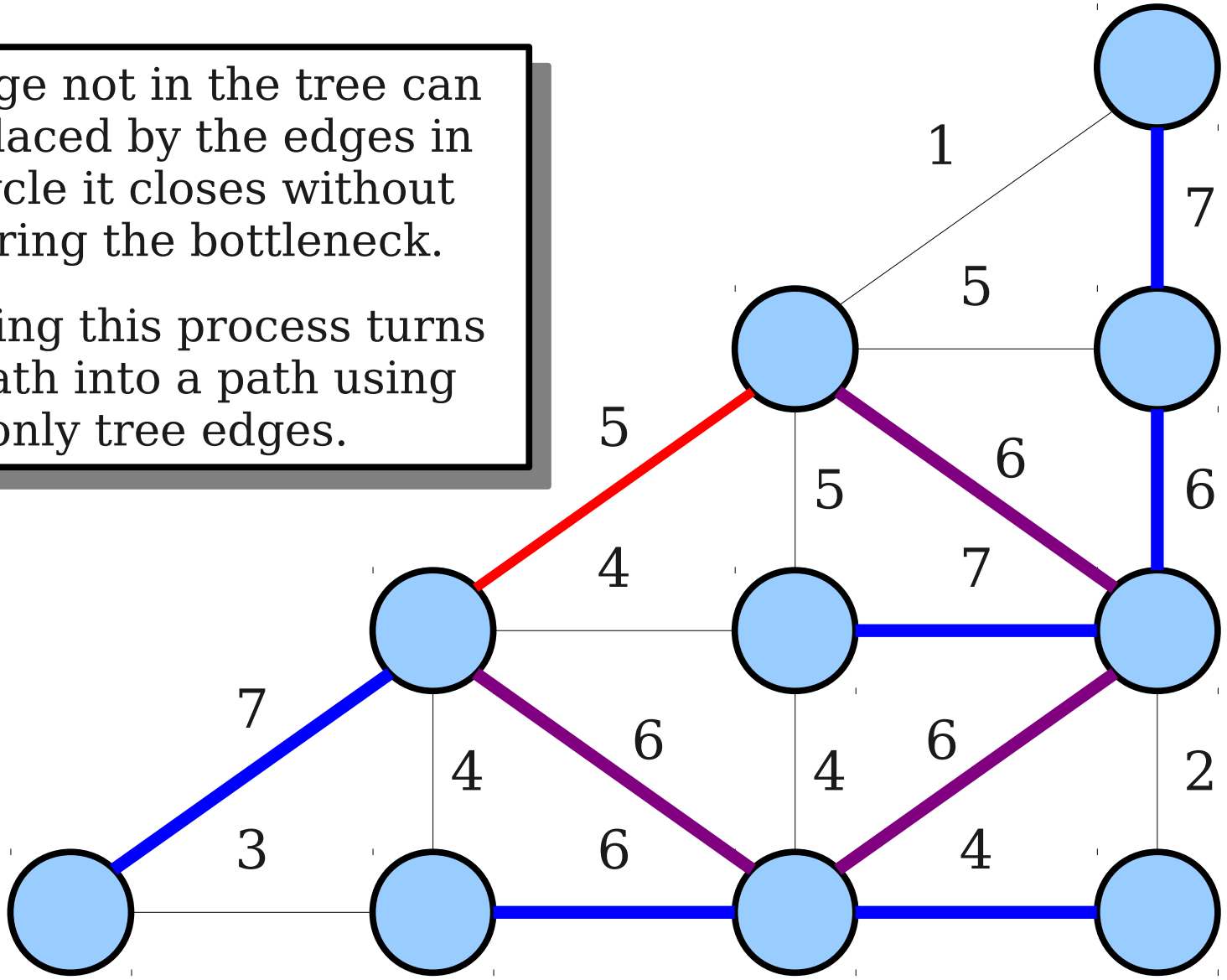
Bottleneck Paths



Bottleneck Paths

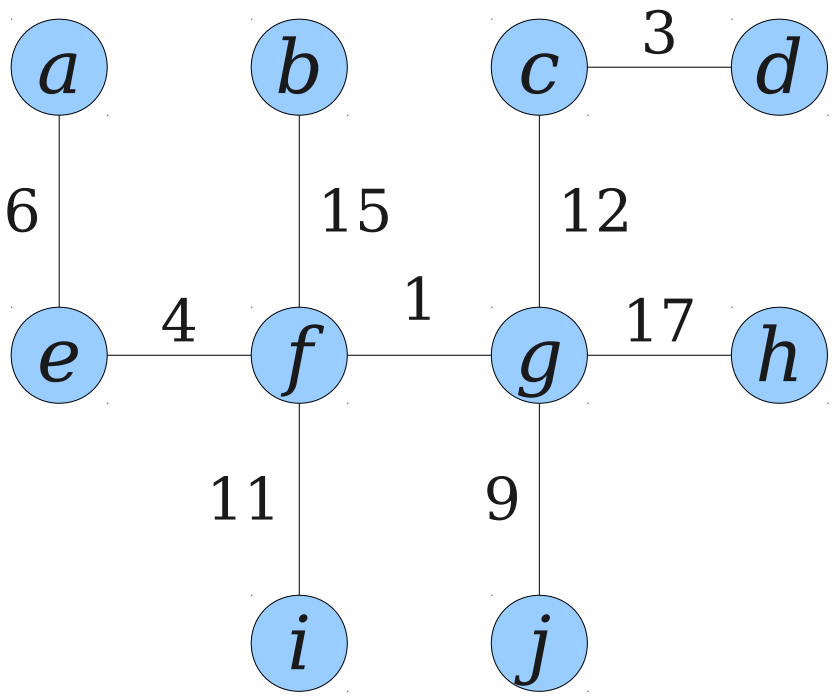
Any edge not in the tree can be replaced by the edges in the cycle it closes without lowering the bottleneck.

Repeating this process turns any path into a path using only tree edges.



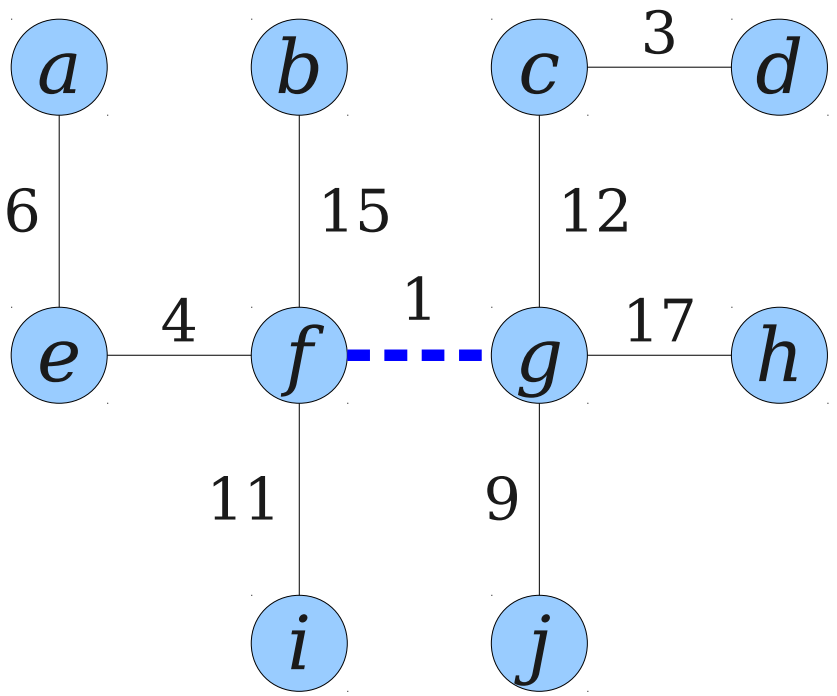
Bottlenecks and Trees

- Because of the previous observation, we only really need to focus on this problem as applied to trees.
- **Good News:** There is an elegant recursive observation about this problem on trees.



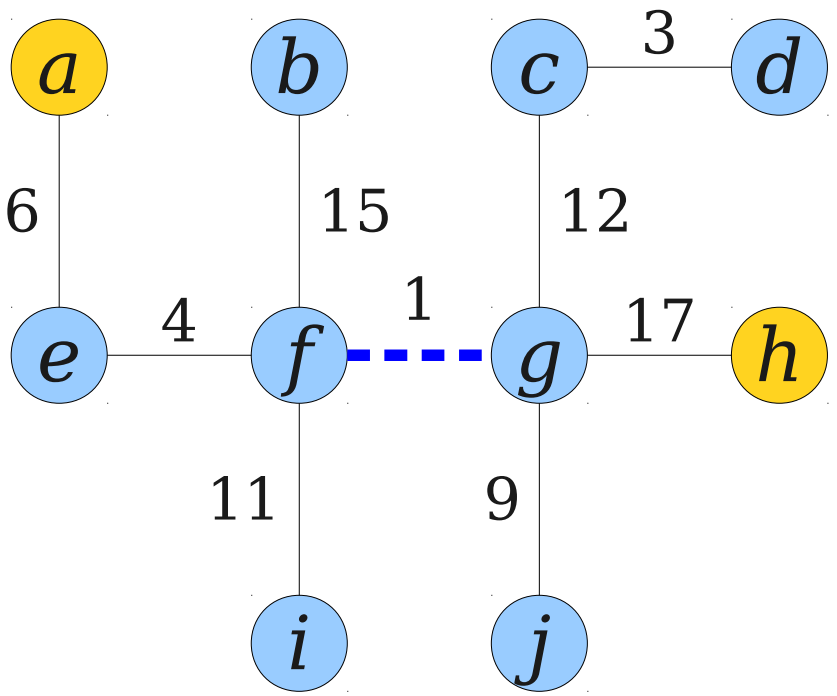
Bottlenecks and Trees

- Because of the previous observation, we only really need to focus on this problem as applied to trees.
- **Good News:** There is an elegant recursive observation about this problem on trees.



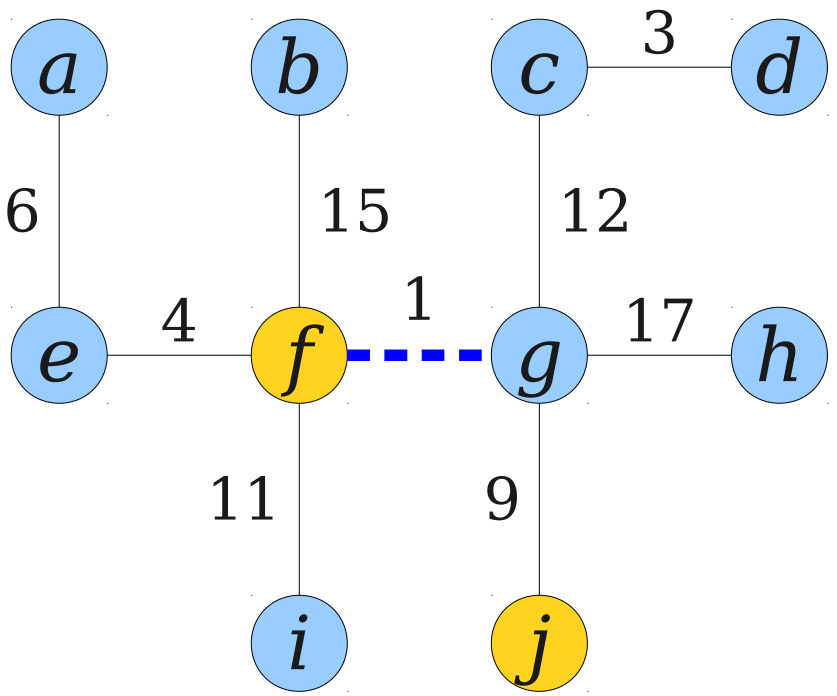
Bottlenecks and Trees

- Because of the previous observation, we only really need to focus on this problem as applied to trees.
- **Good News:** There is an elegant recursive observation about this problem on trees.



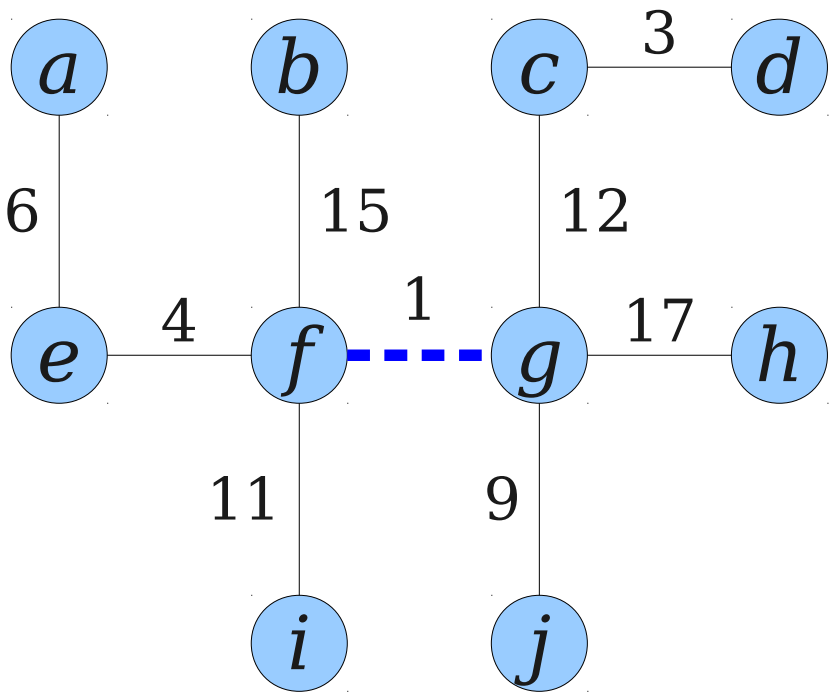
Bottlenecks and Trees

- Because of the previous observation, we only really need to focus on this problem as applied to trees.
- **Good News:** There is an elegant recursive observation about this problem on trees.



Bottlenecks and Trees

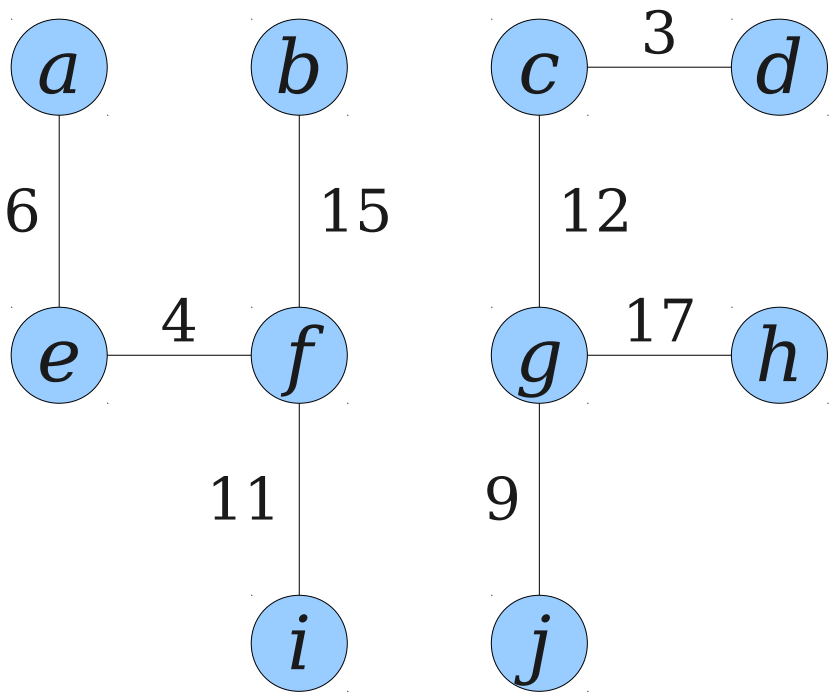
- Because of the previous observation, we only really need to focus on this problem as applied to trees.
- **Good News:** There is an elegant recursive observation about this problem on trees.



Any nodes on opposite sides of this edge have bottleneck cost 1.

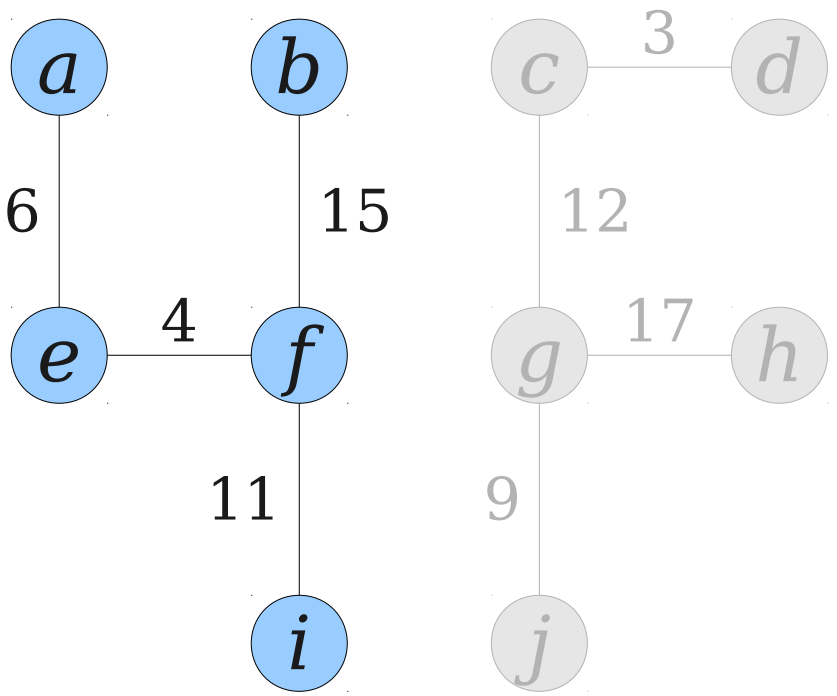
Bottlenecks and Trees

- Because of the previous observation, we only really need to focus on this problem as applied to trees.
- **Good News:** There is an elegant recursive observation about this problem on trees.



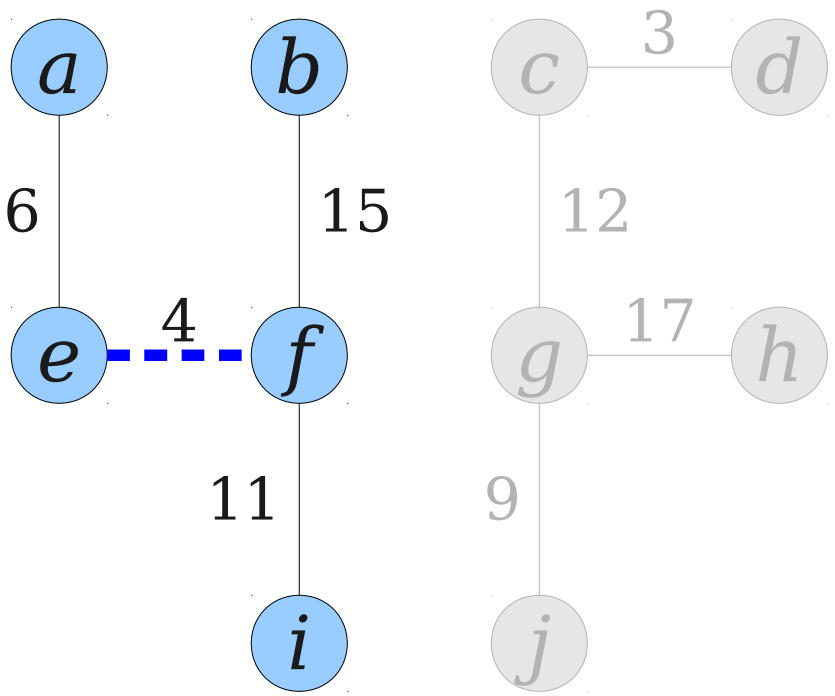
Bottlenecks and Trees

- Because of the previous observation, we only really need to focus on this problem as applied to trees.
- **Good News:** There is an elegant recursive observation about this problem on trees.



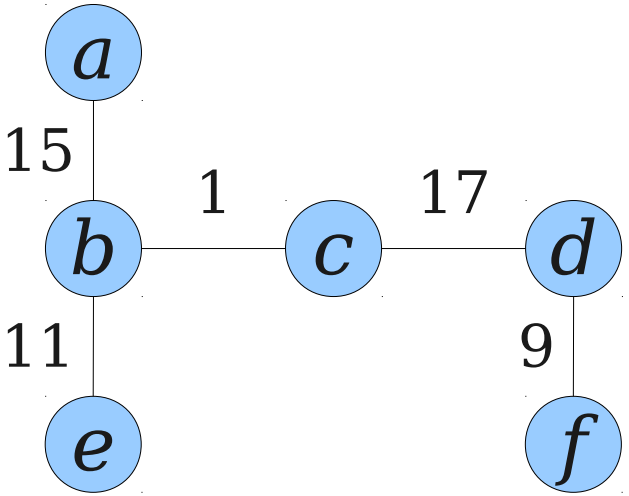
Bottlenecks and Trees

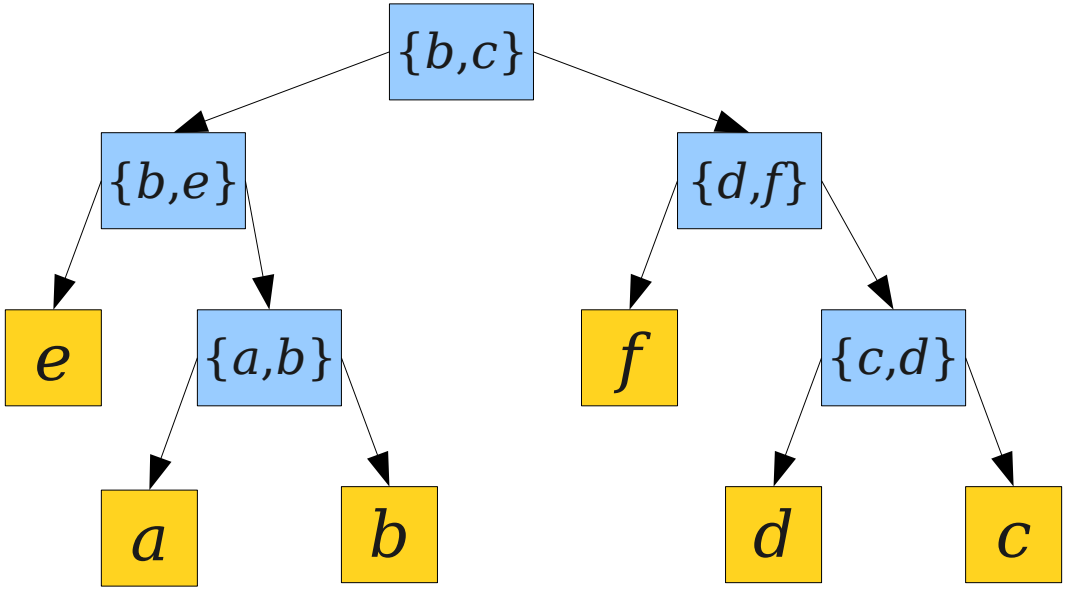
- Because of the previous observation, we only really need to focus on this problem as applied to trees.
- **Good News:** There is an elegant recursive observation about this problem on trees.



Cartesian Trees of Trees

- Given a tree T, the **Cartesian tree of tree T** is defined recursively:
  - The root of the tree is the minimum-capacity edge in T; call that edge $\{u, v\}$.
  - The left subtree is the Cartesian tree of the u subtree and the right subtree is the Cartesian tree of the v subtree.
  - The Cartesian tree of a one-node tree is that node itself.





Cartesian Trees of Trees

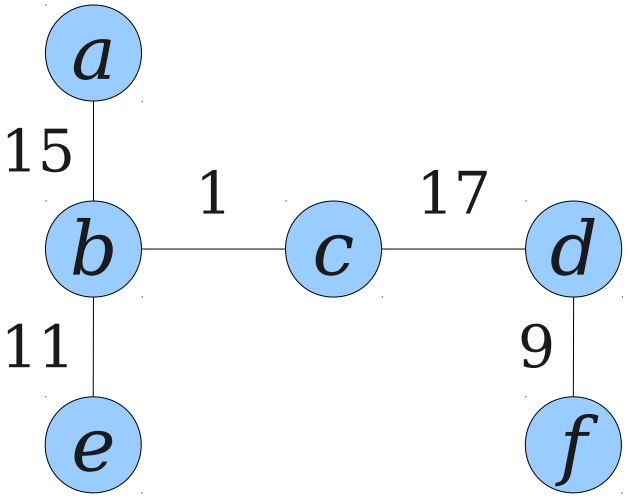
- Cartesian trees of trees are useful for the following reason:

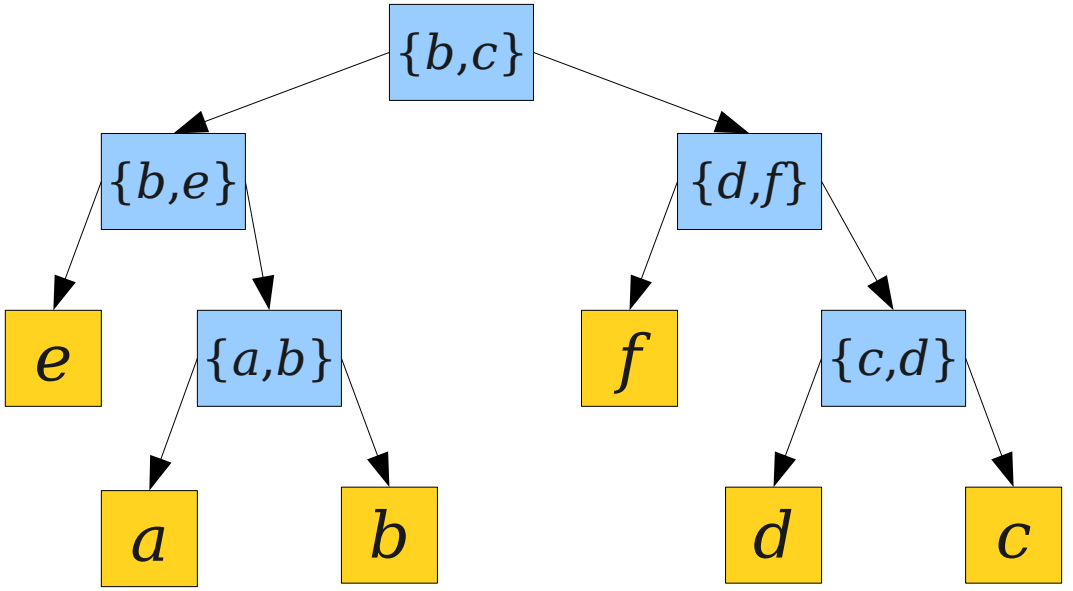
  In tree T, the bottleneck between u and v is the lowest common ancestor of u in v in the Cartesian tree of T.

Cartesian Trees of Trees

- Cartesian trees of trees are useful for the following reason:

  In tree T, the bottleneck between u and v is the lowest common ancestor of u in v in the Cartesian tree of T.

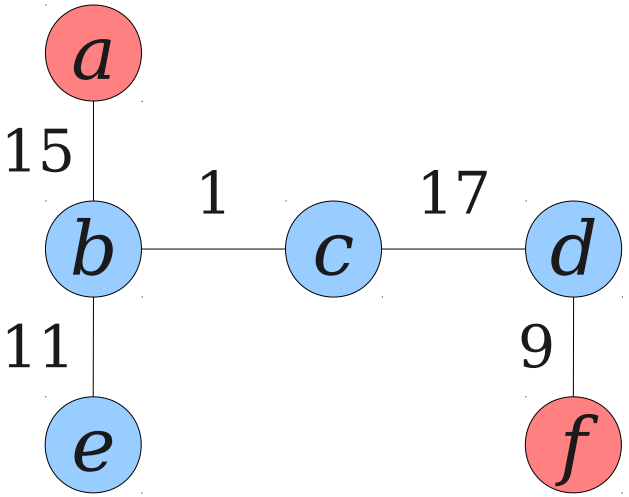


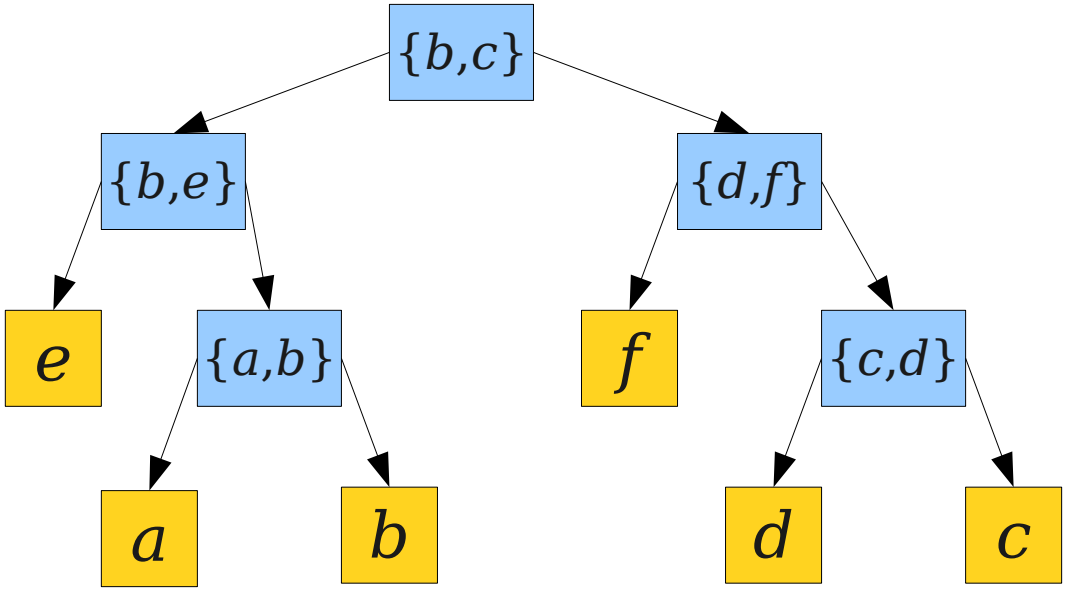


Cartesian Trees of Trees

- Cartesian trees of trees are useful for the following reason:

  In tree T, the bottleneck between u and v is the lowest common ancestor of u in v in the Cartesian tree of T.

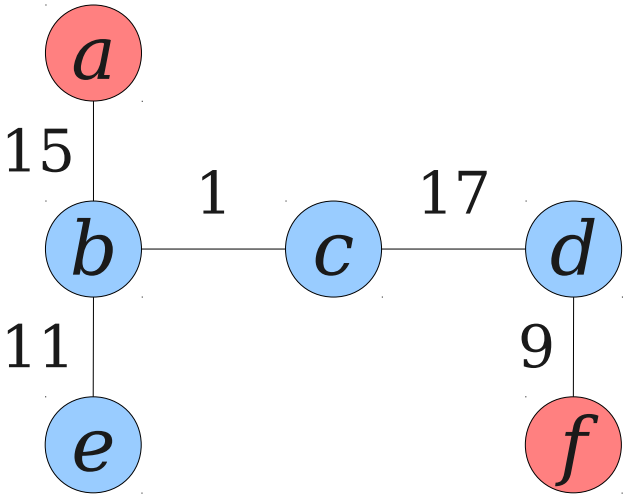


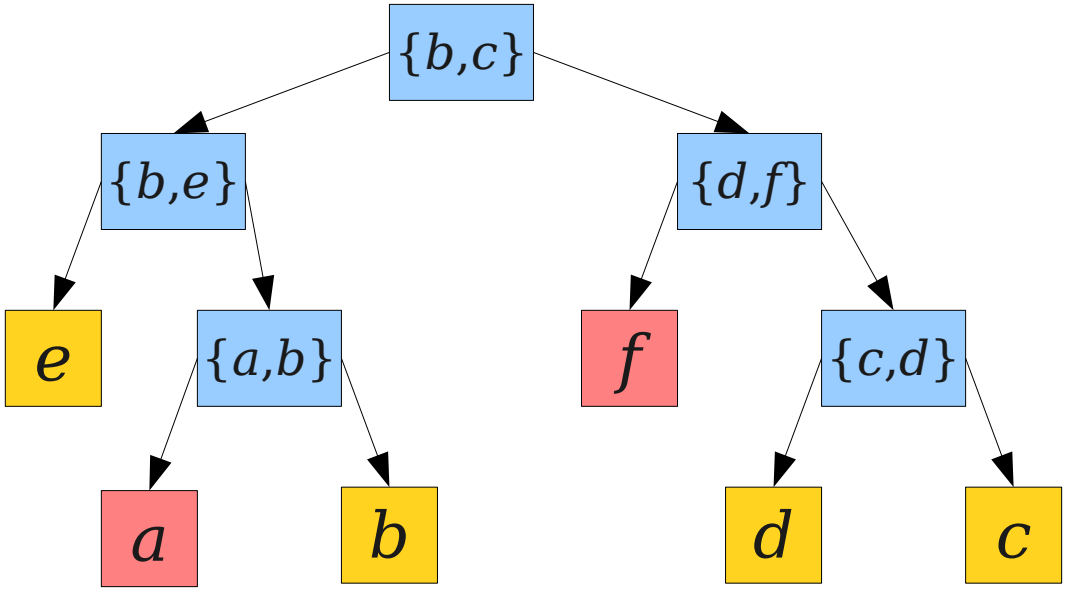


Cartesian Trees of Trees

- Cartesian trees of trees are useful for the following reason:

  In tree T, the bottleneck between u and v is the lowest common ancestor of u in v in the Cartesian tree of T.

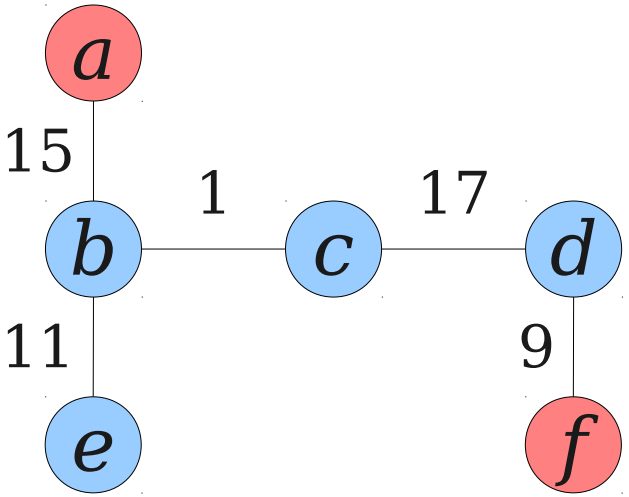


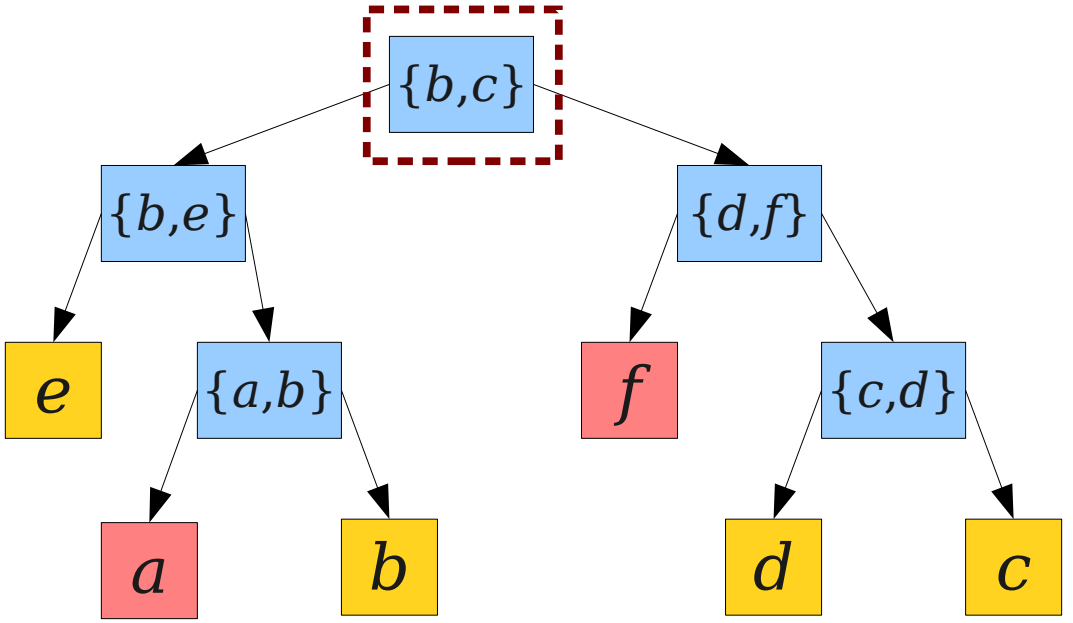


Cartesian Trees of Trees

- Cartesian trees of trees are useful for the following reason:

  In tree T, the bottleneck between u and v is the lowest common ancestor of u in v in the Cartesian tree of T.

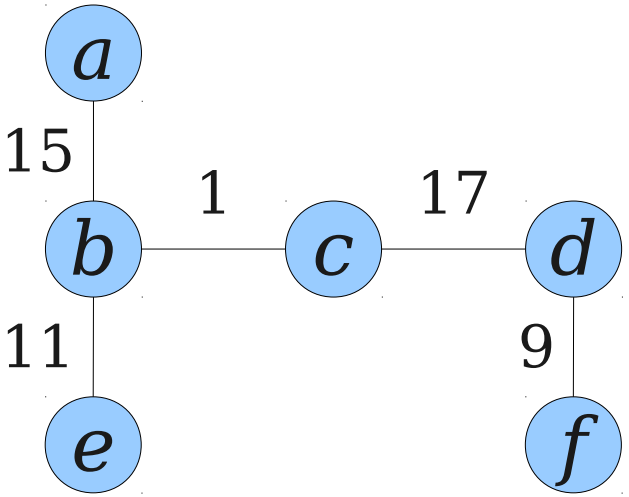


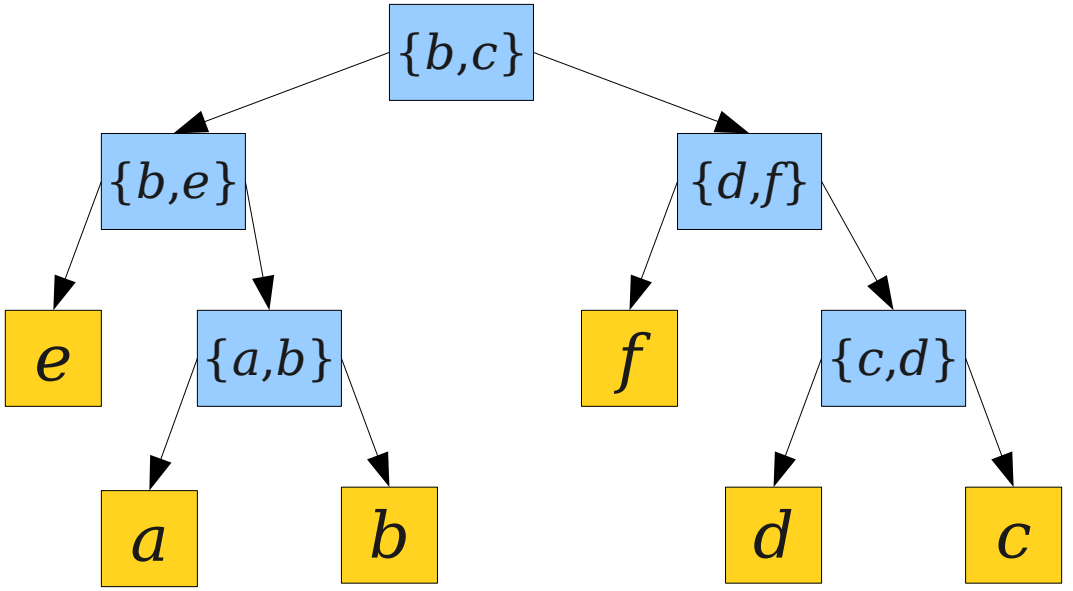


Cartesian Trees of Trees

- Cartesian trees of trees are useful for the following reason:

  In tree T, the bottleneck between u and v is the lowest common ancestor of u in v in the Cartesian tree of T.

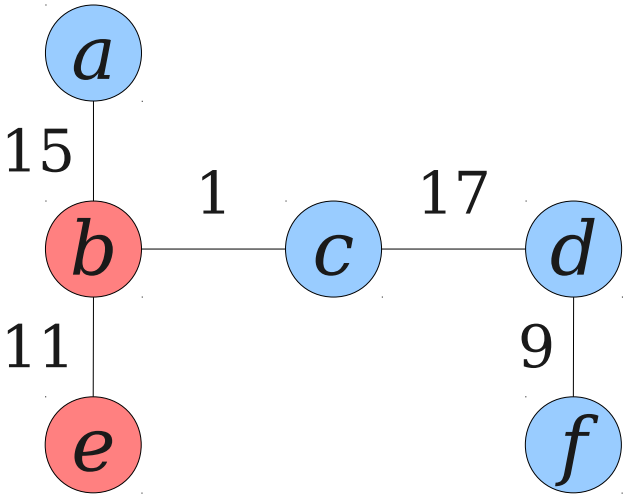


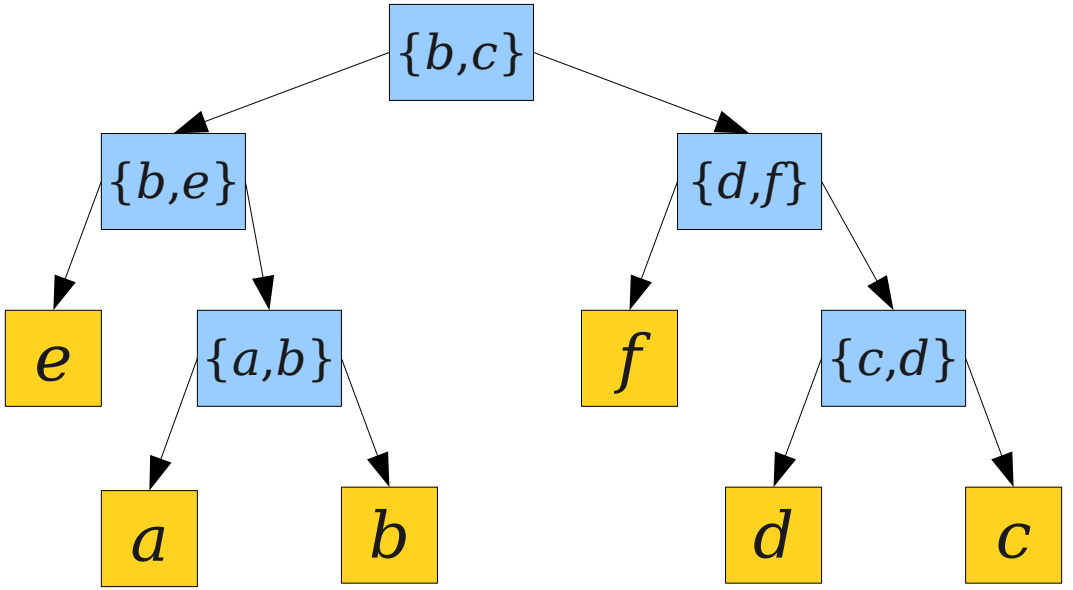


Cartesian Trees of Trees

- Cartesian trees of trees are useful for the following reason:

  In tree T, the bottleneck between u and v is the lowest common ancestor of u in v in the Cartesian tree of T.

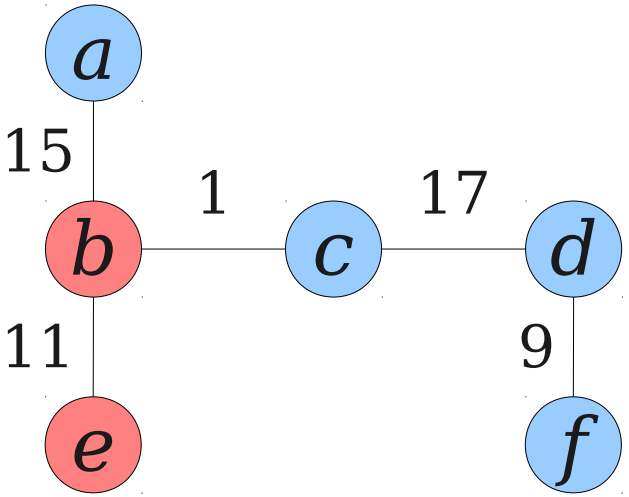


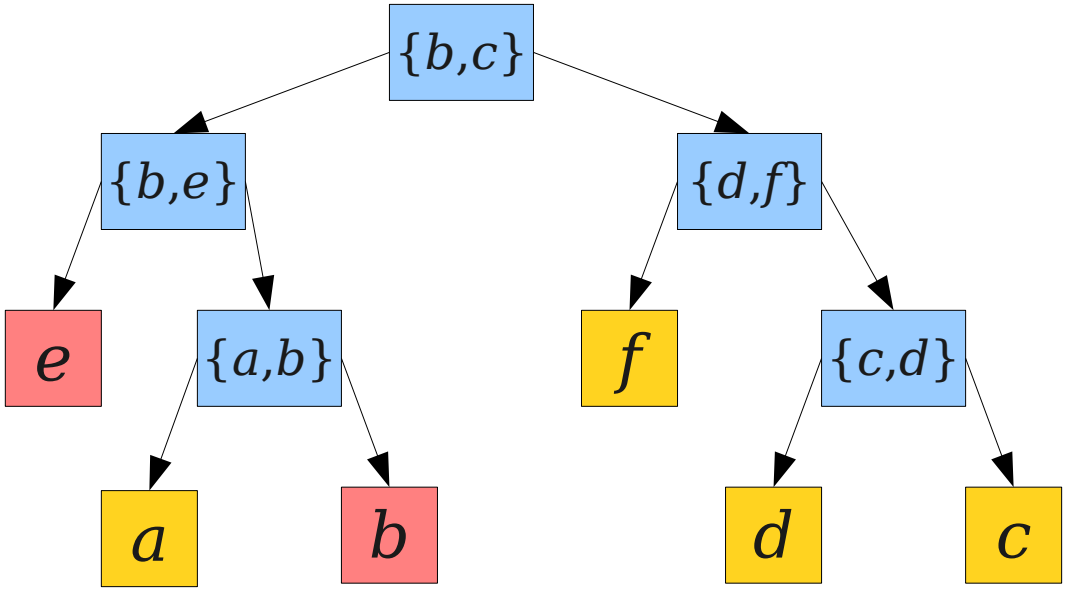


Cartesian Trees of Trees

- Cartesian trees of trees are useful for the following reason:

  In tree T, the bottleneck between u and v is the lowest common ancestor of u in v in the Cartesian tree of T.

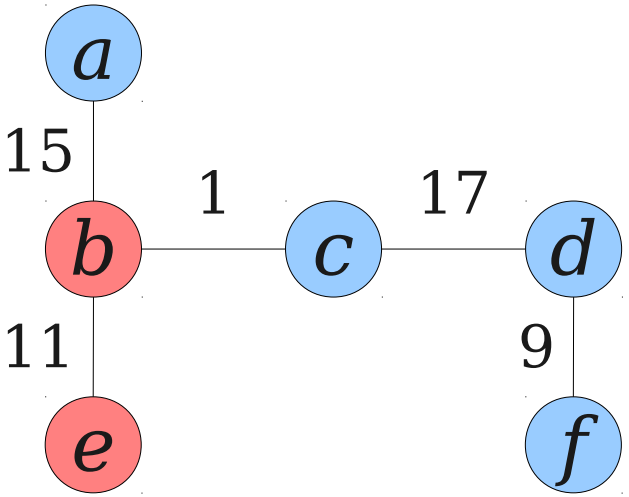


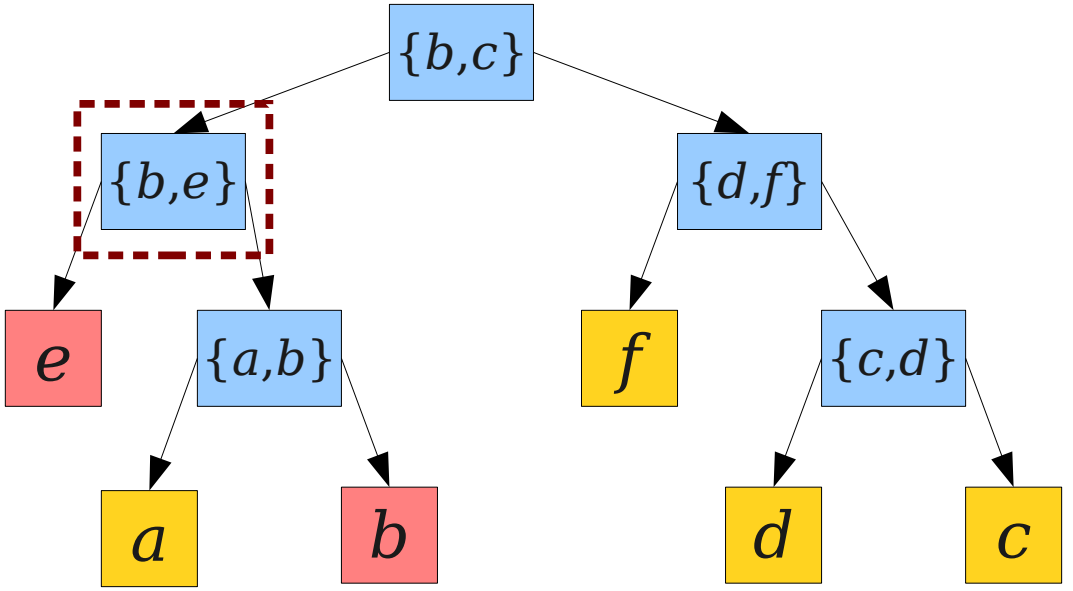


Cartesian Trees of Trees

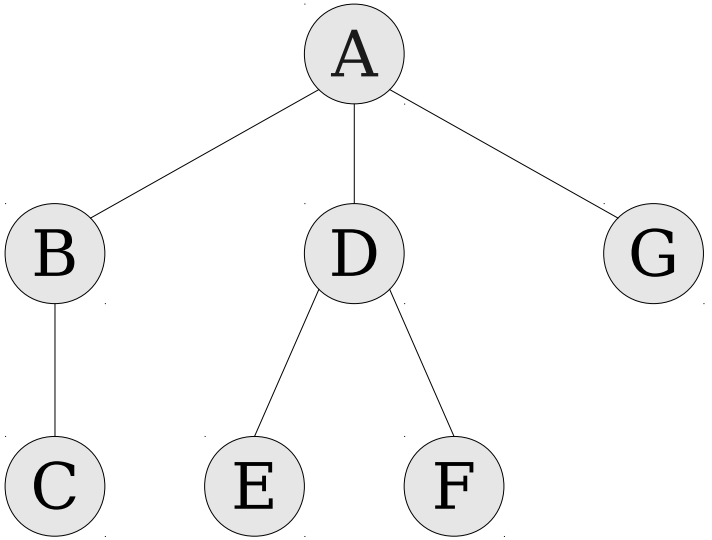
- Cartesian trees of trees are useful for the following reason:

  In tree T, the bottleneck between u and v is the lowest common ancestor of u in v in the Cartesian tree of T.

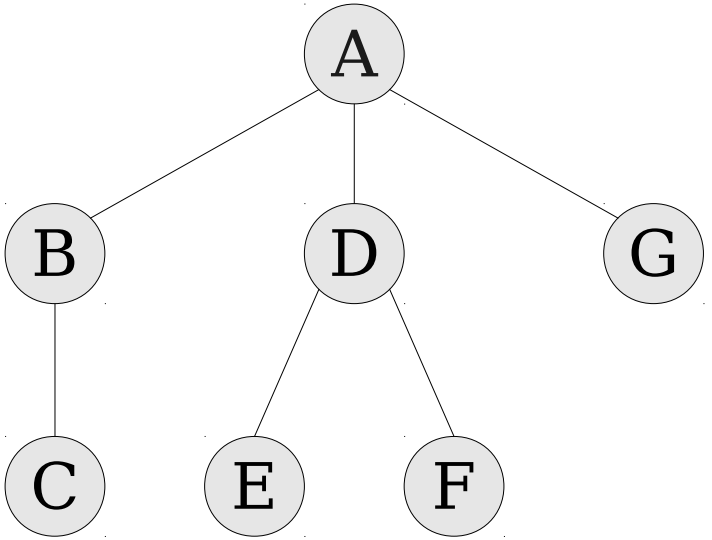




RMQ and LCA

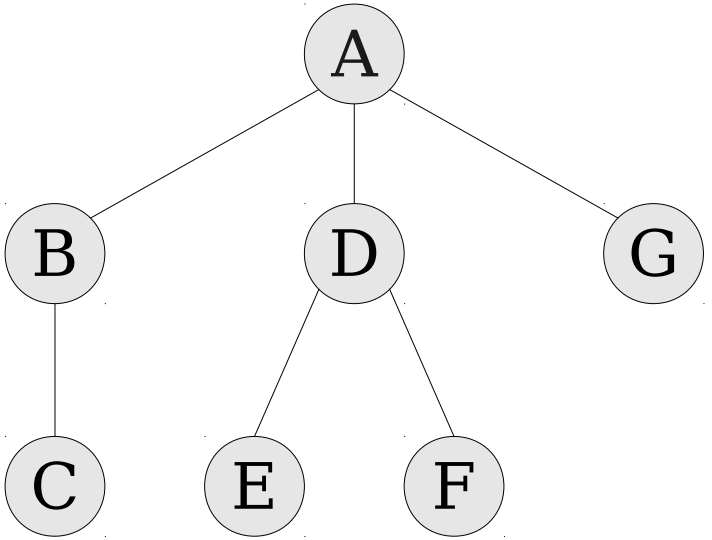


RMQ and LCA



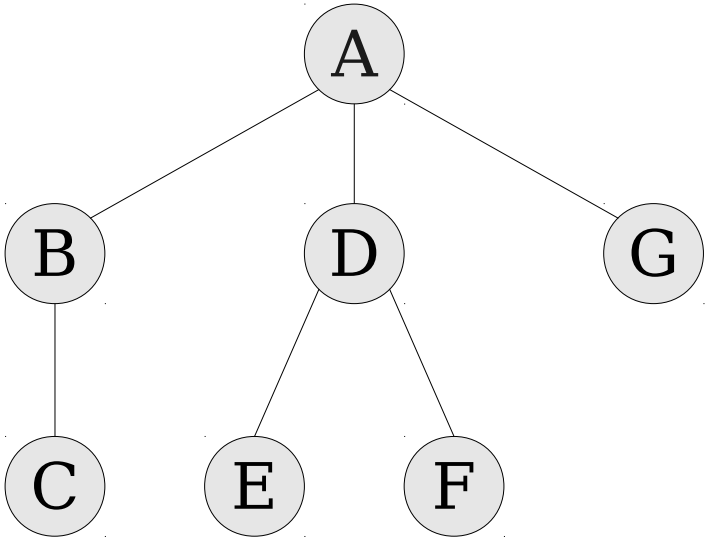
A	B	C	B	A	D	E	D	F	D	A	G	A

RMQ and LCA



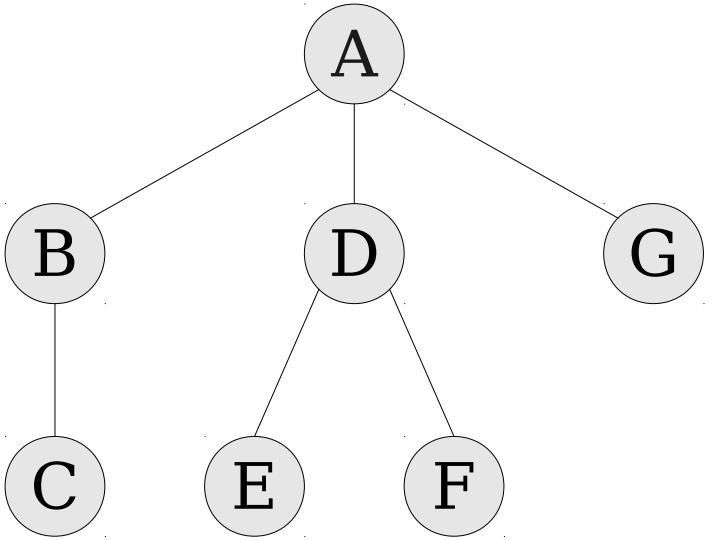
A	B	C	B	A	D	E	D	F	D	A	G	A
0	1	2	1	0	1	2	1	2	1	0	1	0

RMQ and LCA



A	B	C	B	A	D	E	D	F	D	A	G	A
0	1	2	1	0	1	2	1	2	1	0	1	0

RMQ and LCA



A	B	C	B	A	D	E	D	F	D	A	G	A
0	1	2	1	0	1	2	1	2	1	0	1	0

Solving RMQ on the Euler tour of a tree gives a solution to LCA on that tree. LCA has an $\langle O(n), O(1) \rangle$ solution!

The Solution

- As preprocessing:
  - Compute a maximum spanning tree T^* in time $O(m + n \log n)$.
  - Construct the Cartesian tree of T^* in time $O(n \log n)$ (details in a second).
  - Construct an LCA data structure for that Cartesian tree in time $O(n)$.
  - Total preprocessing: **$O(m + n \log n)$**.
- To make a query for the bottleneck edge between u and v:
  - Compute the LCA of u and v.
  - Total time: **$O(1)$**.

Summary

- Trees can be represented via Euler tours.
- Given the Euler tours of two trees, those trees can be linked by doing O(1) concatenations and splits.
- Given the Euler tour of a tree, can cut that tree by doing O(1) concatenations and splits.
- Representing the Euler tours of various trees as balanced trees gives O(log n) link, cut and is-connected operations.
- The bottleneck edge query problem can be solved in time $\langle O(m + n \log n), O(1) \rangle$ using appropriate data structures.

Next Time

- **Amortized Analysis**
  - Can we trade worst-case efficiency for global efficiency?
- **The Banker's Method**
  - Putting credits on data structures.
- **The Potential Method**
  - Defining the potential energy of a data structure.
- **Amortized Efficient Data Structures**
  - A sampling of amortized data structures.