Dynamic Graph Algorithms

Christian Wulff-Nilsen
University of Copenhagen

November 14, 2019

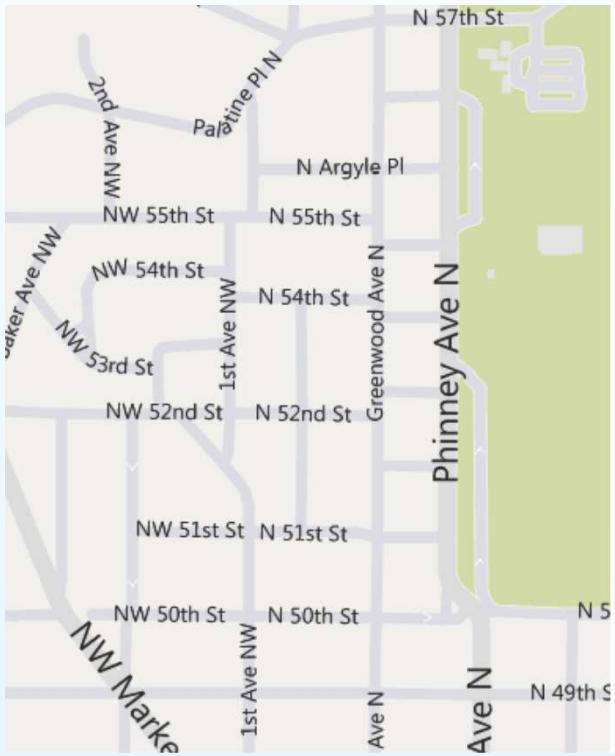
Graph algorithms and data structures

Graph algorithms and data structures

- Graphs and graph problems are everywhere:

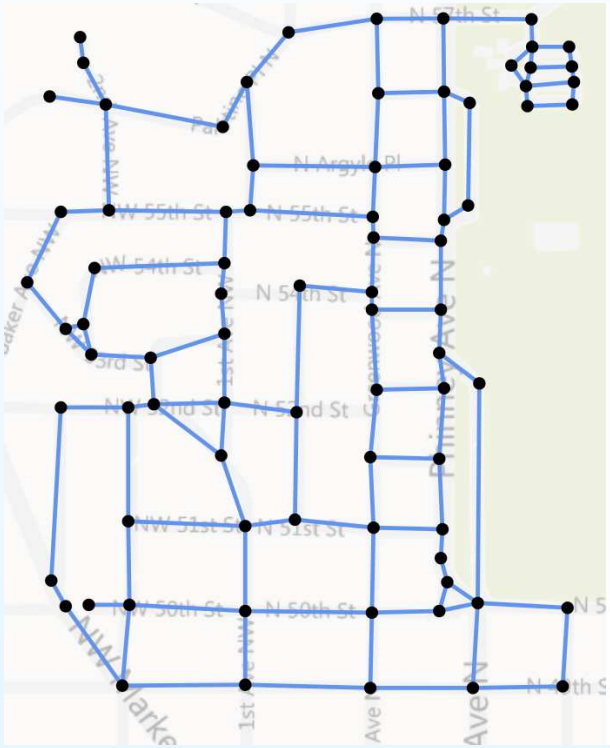
Graph algorithms and data structures

- Graphs and graph problems are everywhere:



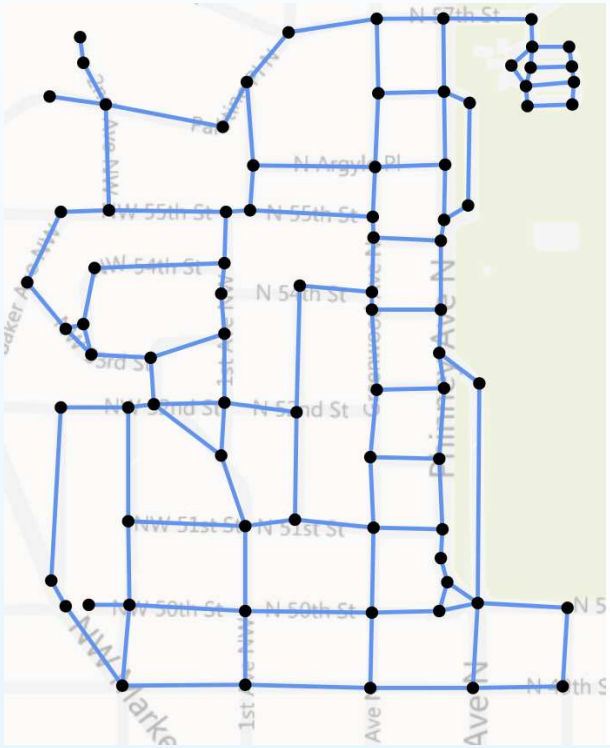
Graph algorithms and data structures

- Graphs and graph problems are everywhere:



Graph algorithms and data structures

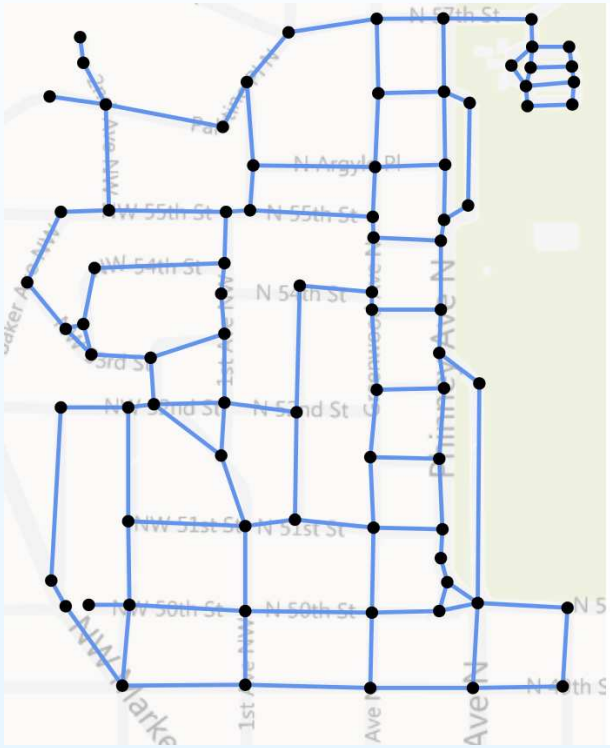
- Graphs and graph problems are everywhere:



- Often, we want to model graphs that change over time

Graph algorithms and data structures

- Graphs and graph problems are everywhere:



- Often, we want to model graphs that change over time
- Dynamic algorithms and data structures are tailor-made to handle changes in very large graphs

Graphs

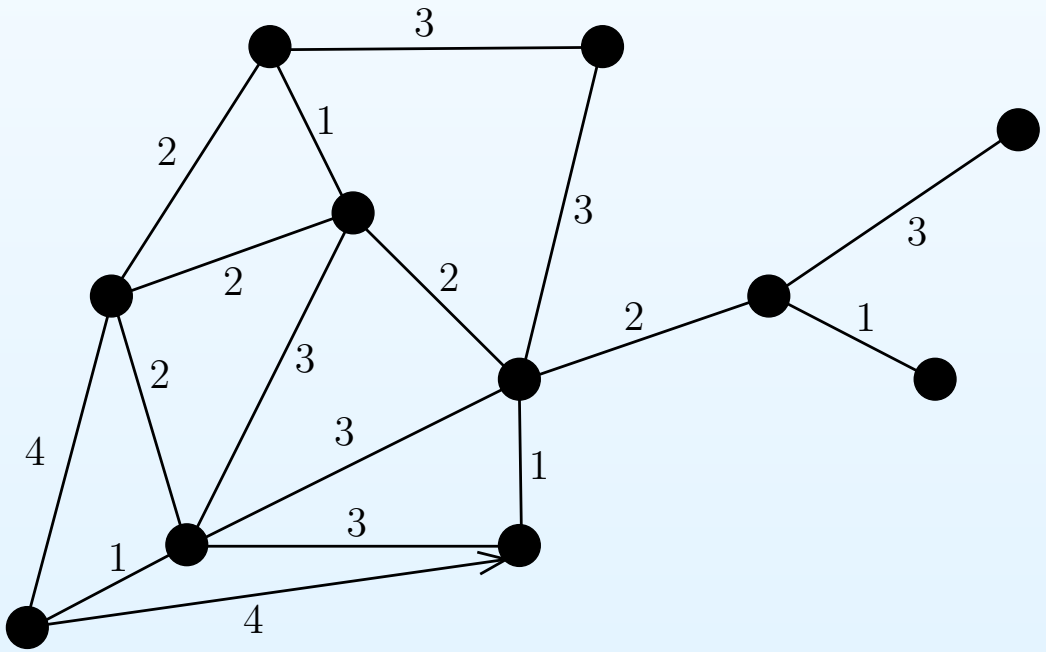
- A graph G is a pair (V, E) where V is the set of *vertices* and E is the set of connections between these vertices called *edges*

Graphs

- A graph G is a pair (V, E) where V is the set of *vertices* and E is the set of connections between these vertices called *edges*
- Let $m = |E|$ and $n = |V|$ in the following

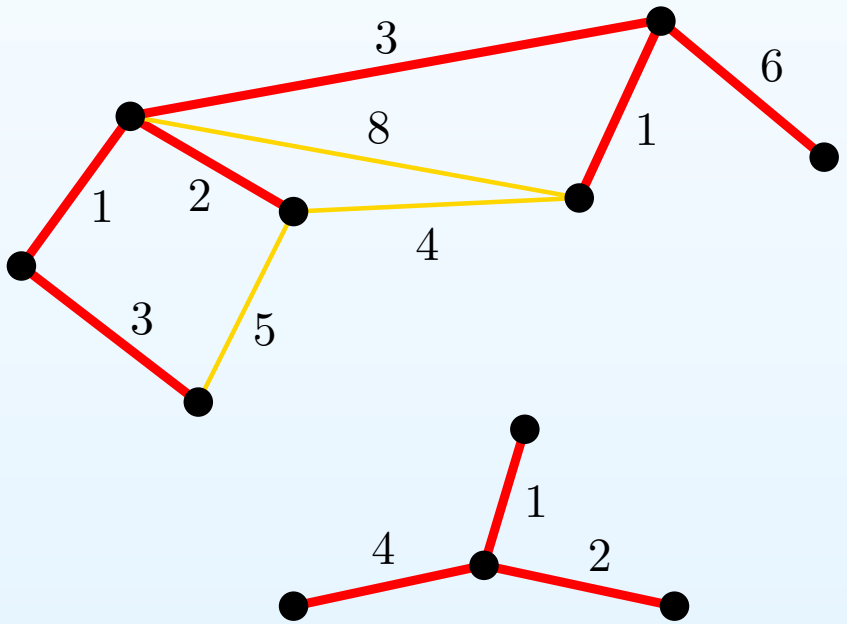
Graphs

- A graph G is a pair (V, E) where V is the set of *vertices* and E is the set of connections between these vertices called *edges*
- Let $m = |E|$ and $n = |V|$ in the following
- Edges may have weights and may be directed or undirected:



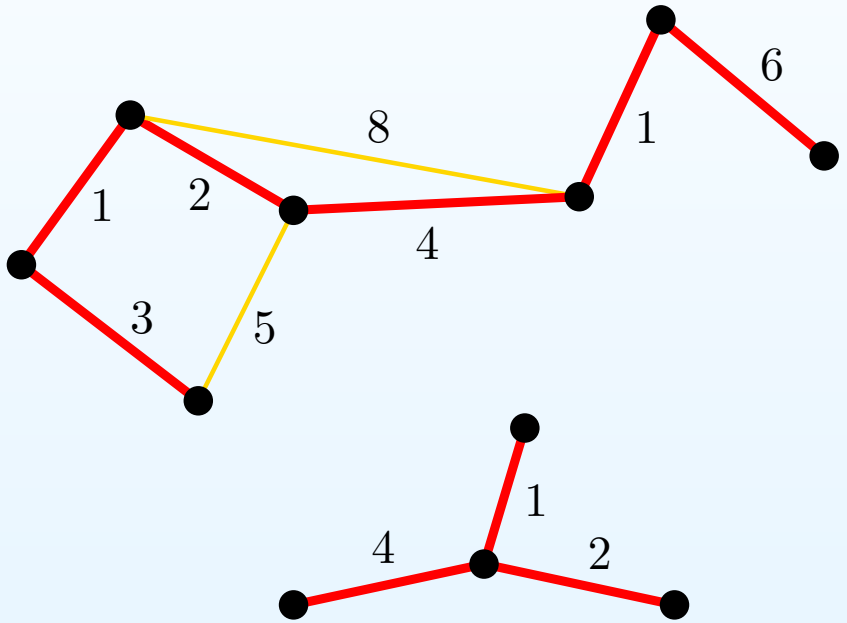
A classical graph problem: minimum spanning forest

- We have near-linear time algorithms (Prim, Kruskal,...) to compute a minimum spanning forest of a given graph:



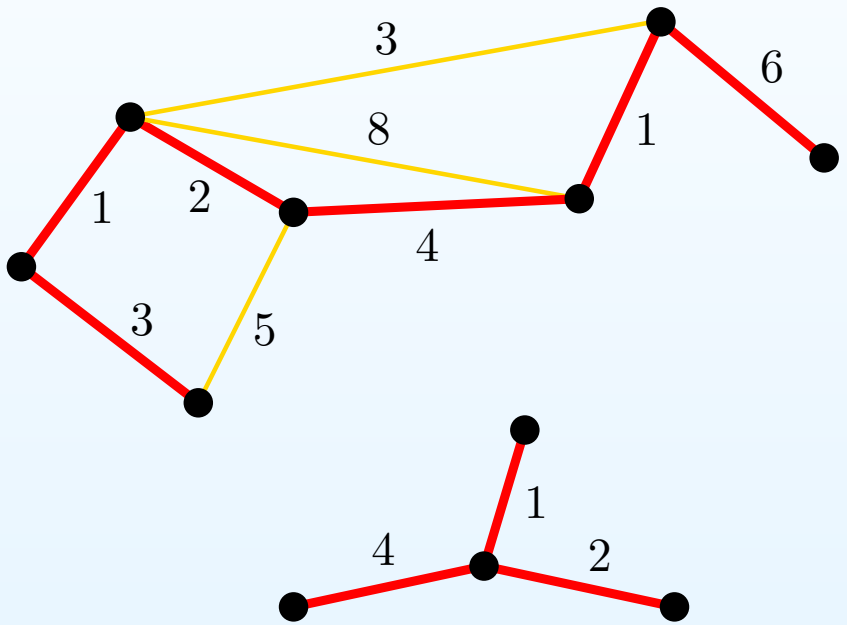
A classical graph problem: minimum spanning forest

- Maintaining a minimum spanning forest in a dynamic graph:



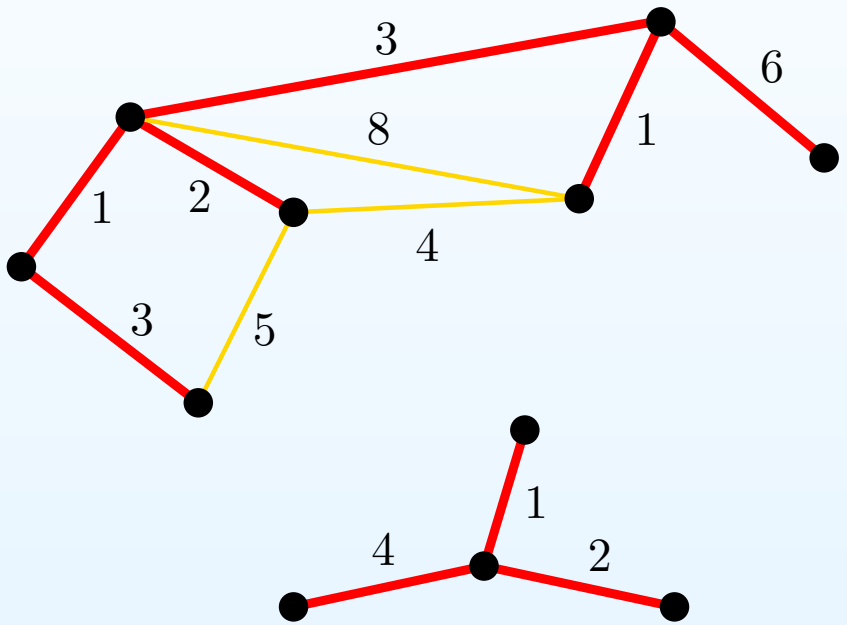
A classical graph problem: minimum spanning forest

- Maintaining a minimum spanning forest in a dynamic graph:



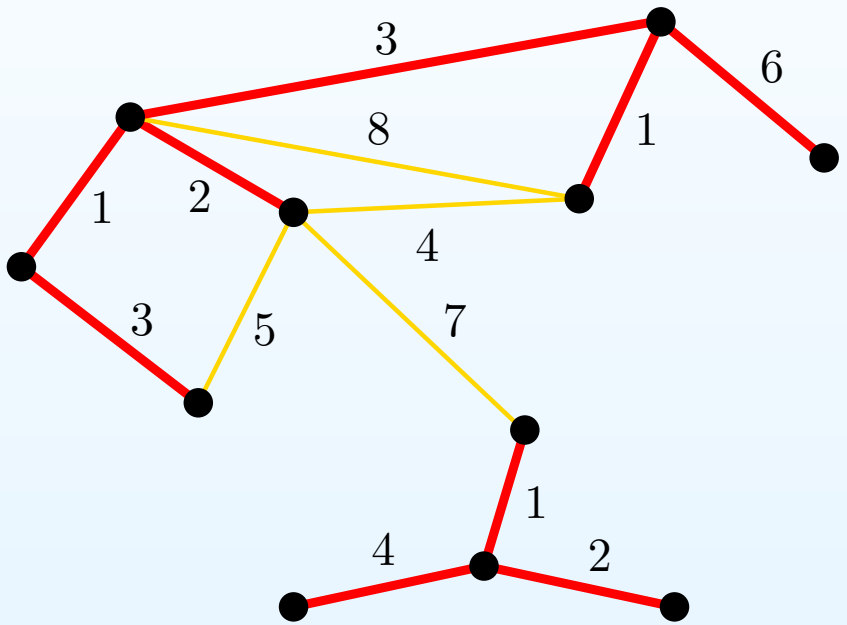
A classical graph problem: minimum spanning forest

- Maintaining a minimum spanning forest in a dynamic graph:



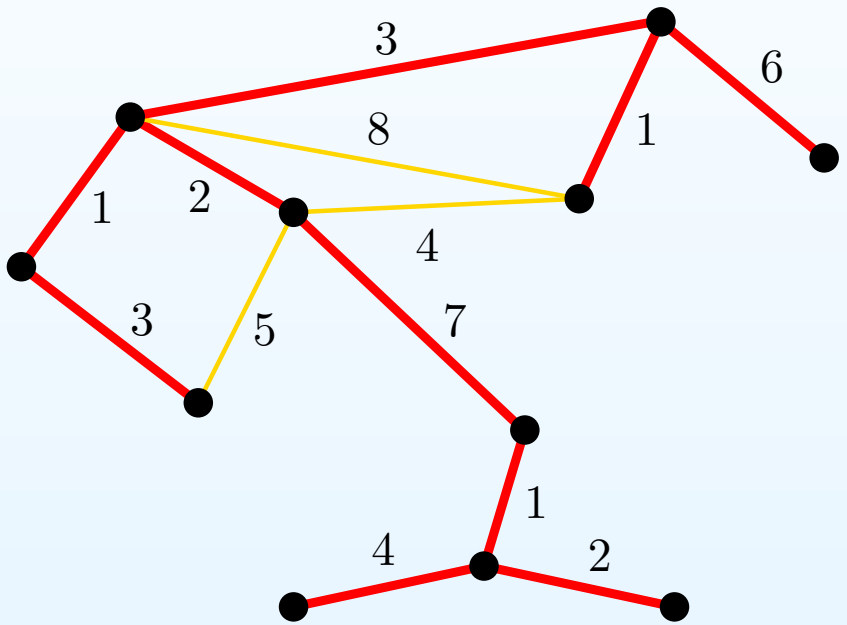
A classical graph problem: minimum spanning forest

- Maintaining a minimum spanning forest in a dynamic graph:



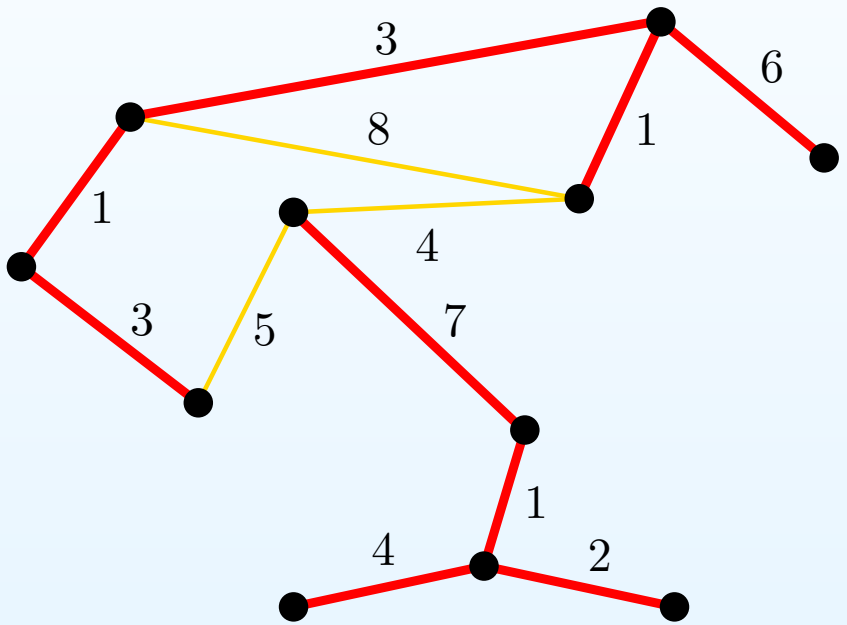
A classical graph problem: minimum spanning forest

- Maintaining a minimum spanning forest in a dynamic graph:



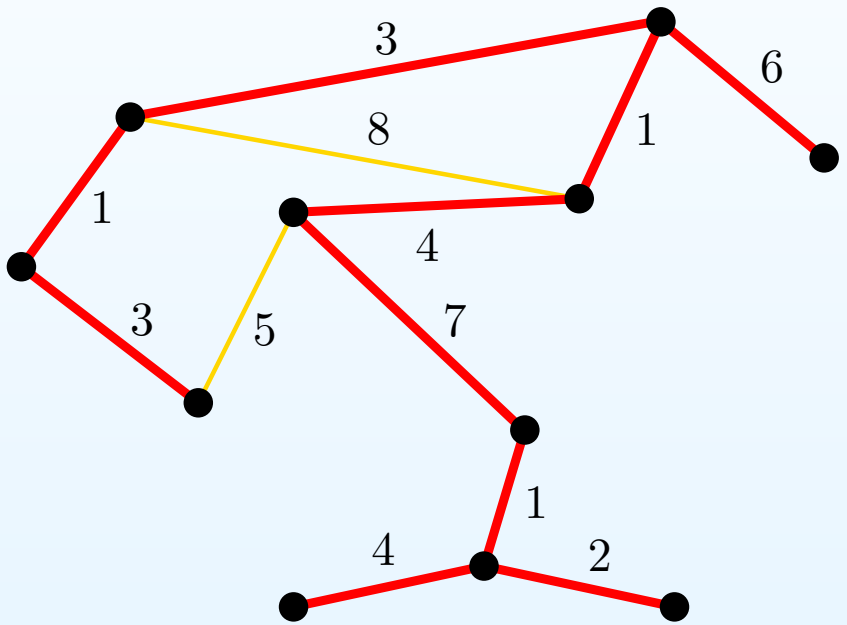
A classical graph problem: minimum spanning forest

- Maintaining a minimum spanning forest in a dynamic graph:



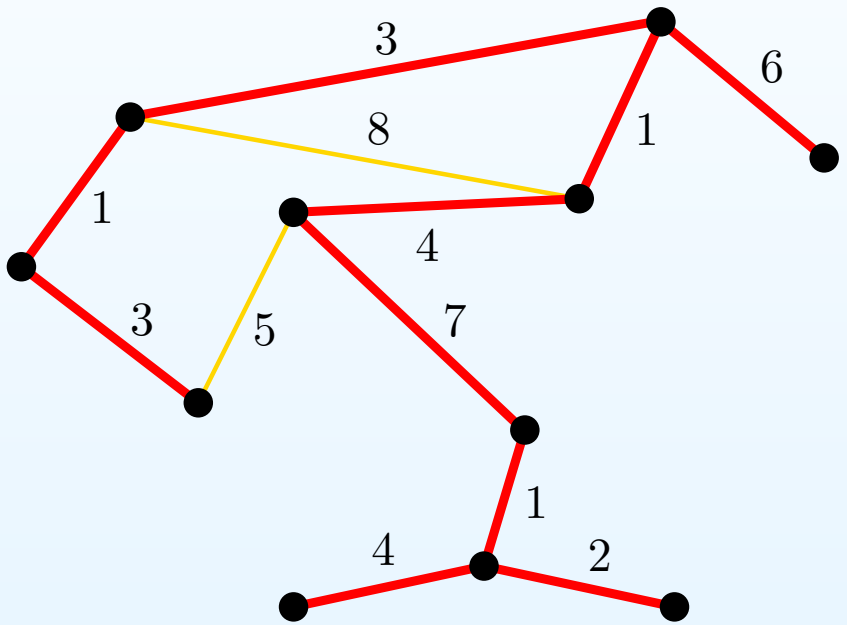
A classical graph problem: minimum spanning forest

- Maintaining a minimum spanning forest in a dynamic graph:



A classical graph problem: minimum spanning forest

- Maintaining a minimum spanning forest in a dynamic graph:



- Can we handle each update faster than rebuilding from scratch?

Dynamic connectivity

- A data structure for dynamic connectivity should support the following in a dynamic graph G:

Dynamic connectivity

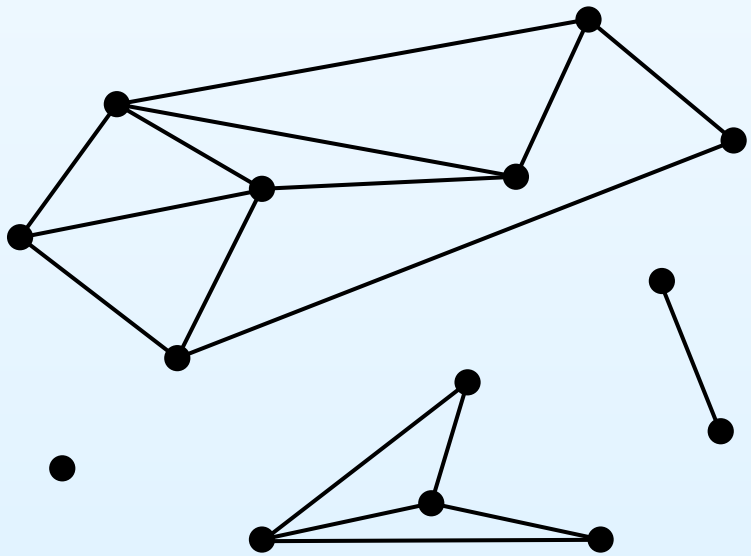
- A data structure for dynamic connectivity should support the following in a dynamic graph G:
  - The insertion/deletion of an edge in G

Dynamic connectivity

- A data structure for dynamic connectivity should support the following in a dynamic graph G:
  - The insertion/deletion of an edge in G
  - A query for whether two vertices u and v are connected in the current graph G

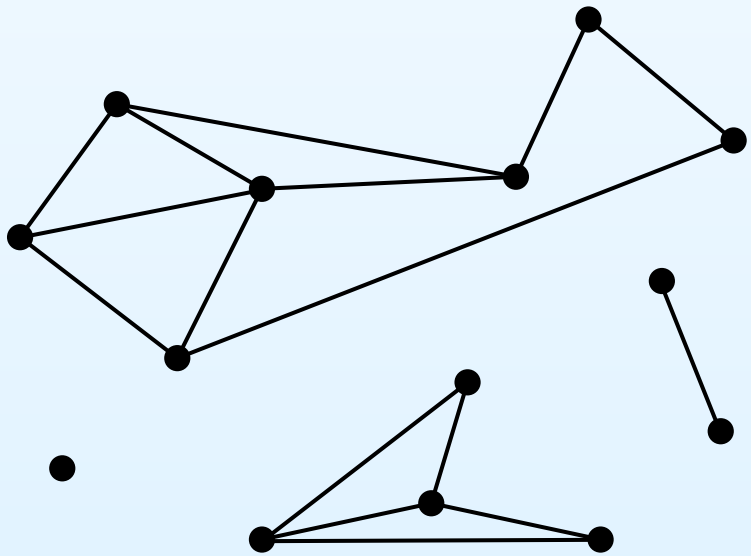
Dynamic connectivity

- A data structure for dynamic connectivity should support the following in a dynamic graph G:
  - The insertion/deletion of an edge in G
  - A query for whether two vertices u and v are connected in the current graph G
- Example:



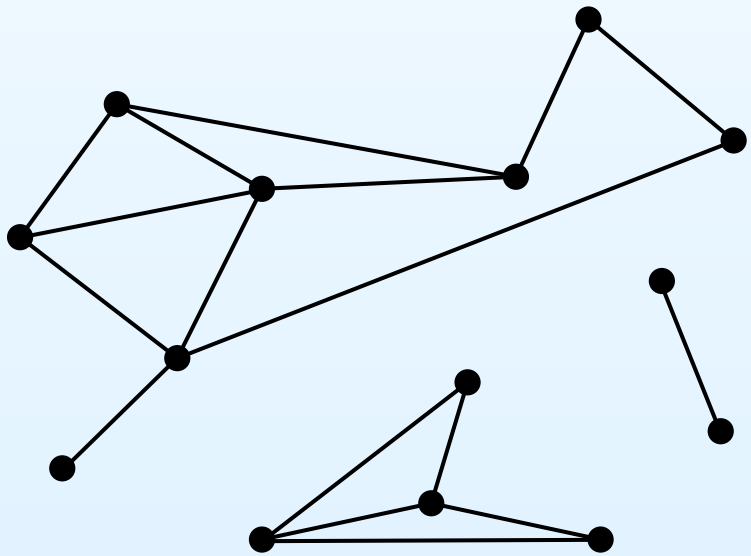
Dynamic connectivity

- A data structure for dynamic connectivity should support the following in a dynamic graph G:
  - The insertion/deletion of an edge in G
  - A query for whether two vertices u and v are connected in the current graph G
- Example:



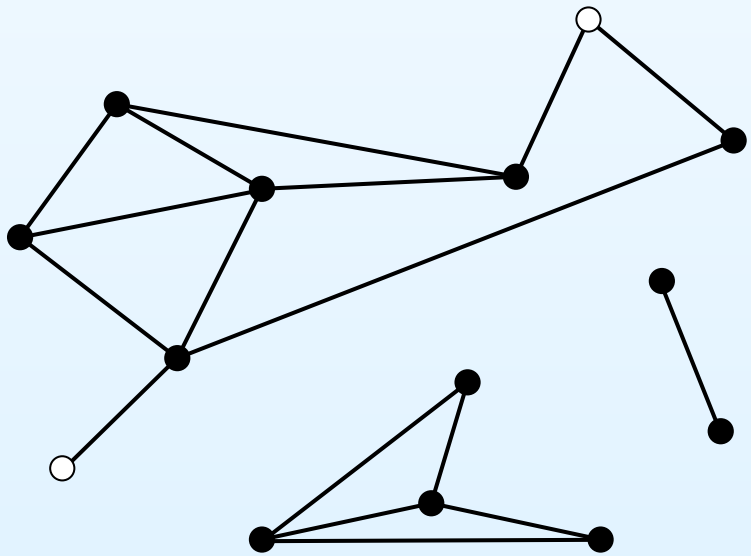
Dynamic connectivity

- A data structure for dynamic connectivity should support the following in a dynamic graph G:
  - The insertion/deletion of an edge in G
  - A query for whether two vertices u and v are connected in the current graph G
- Example:



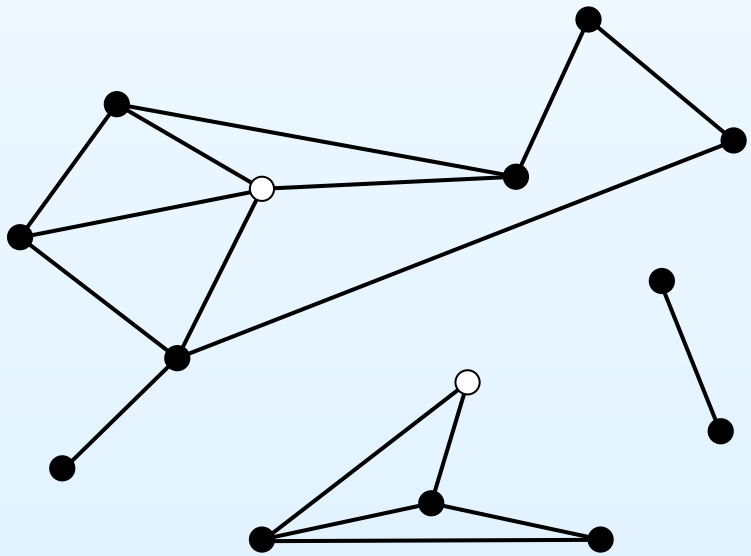
Dynamic connectivity

- A data structure for dynamic connectivity should support the following in a dynamic graph G:
  - The insertion/deletion of an edge in G
  - A query for whether two vertices u and v are connected in the current graph G
- Example:

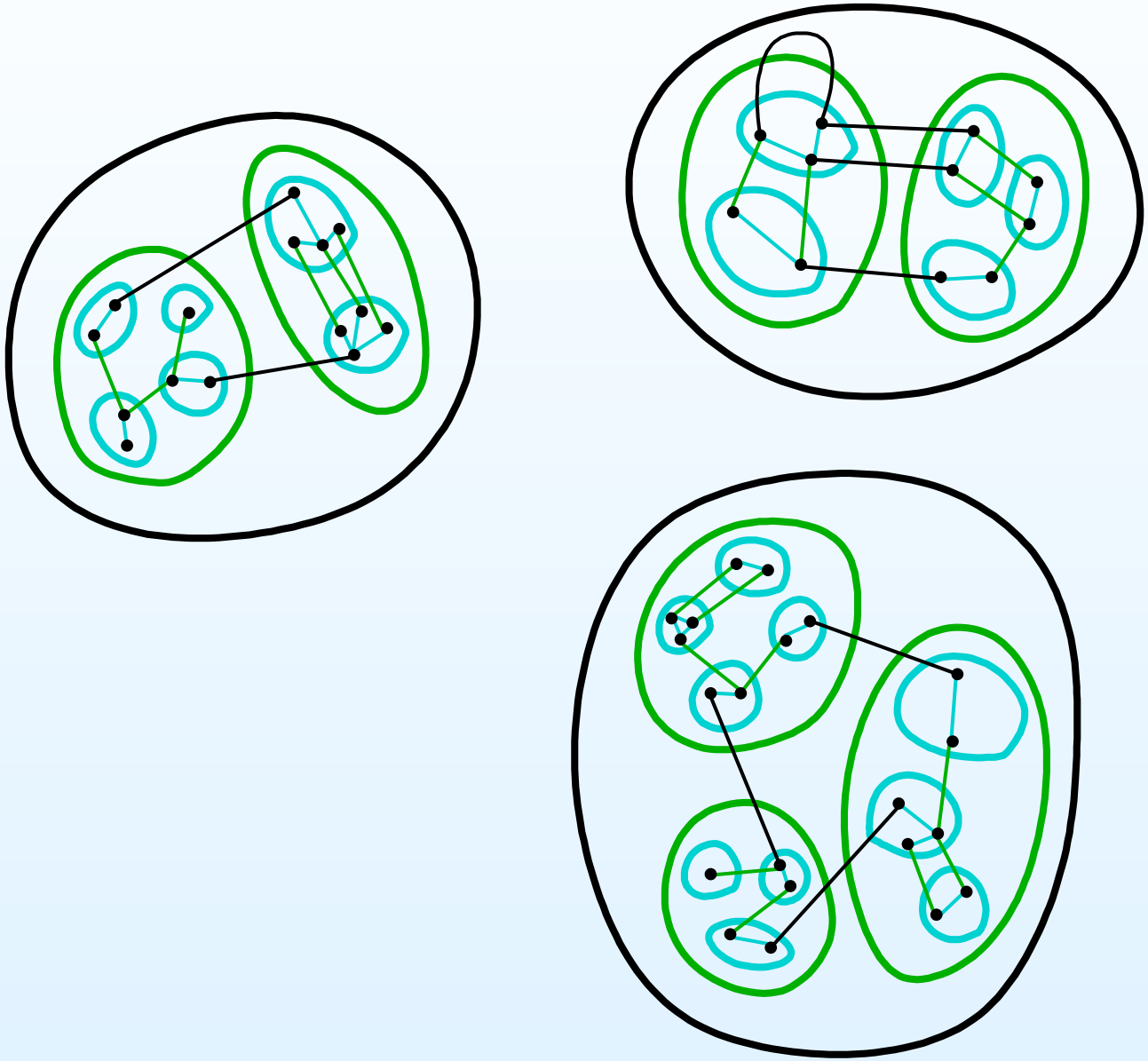


Dynamic connectivity

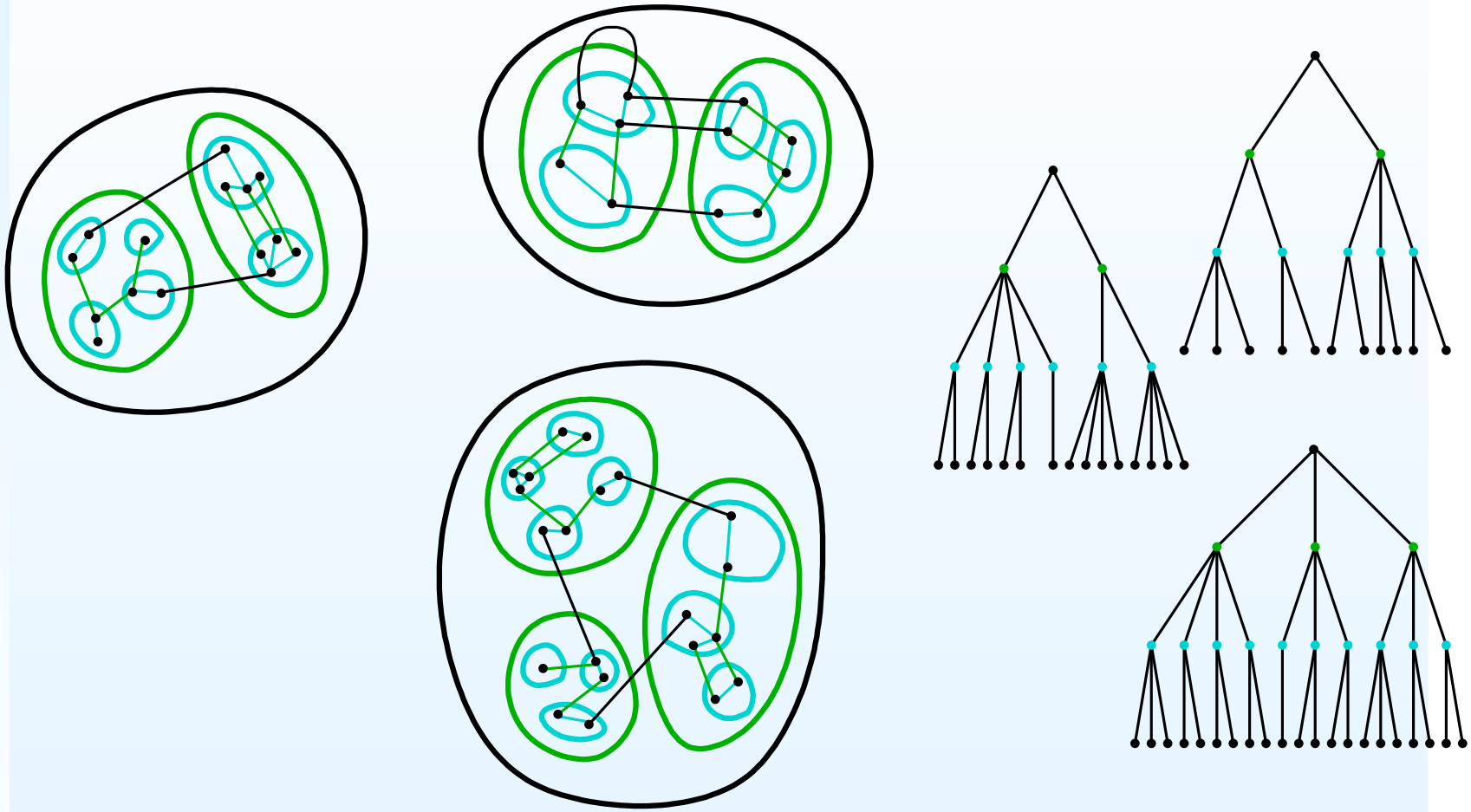
- A data structure for dynamic connectivity should support the following in a dynamic graph G:
  - The insertion/deletion of an edge in G
  - A query for whether two vertices u and v are connected in the current graph G
- Example:



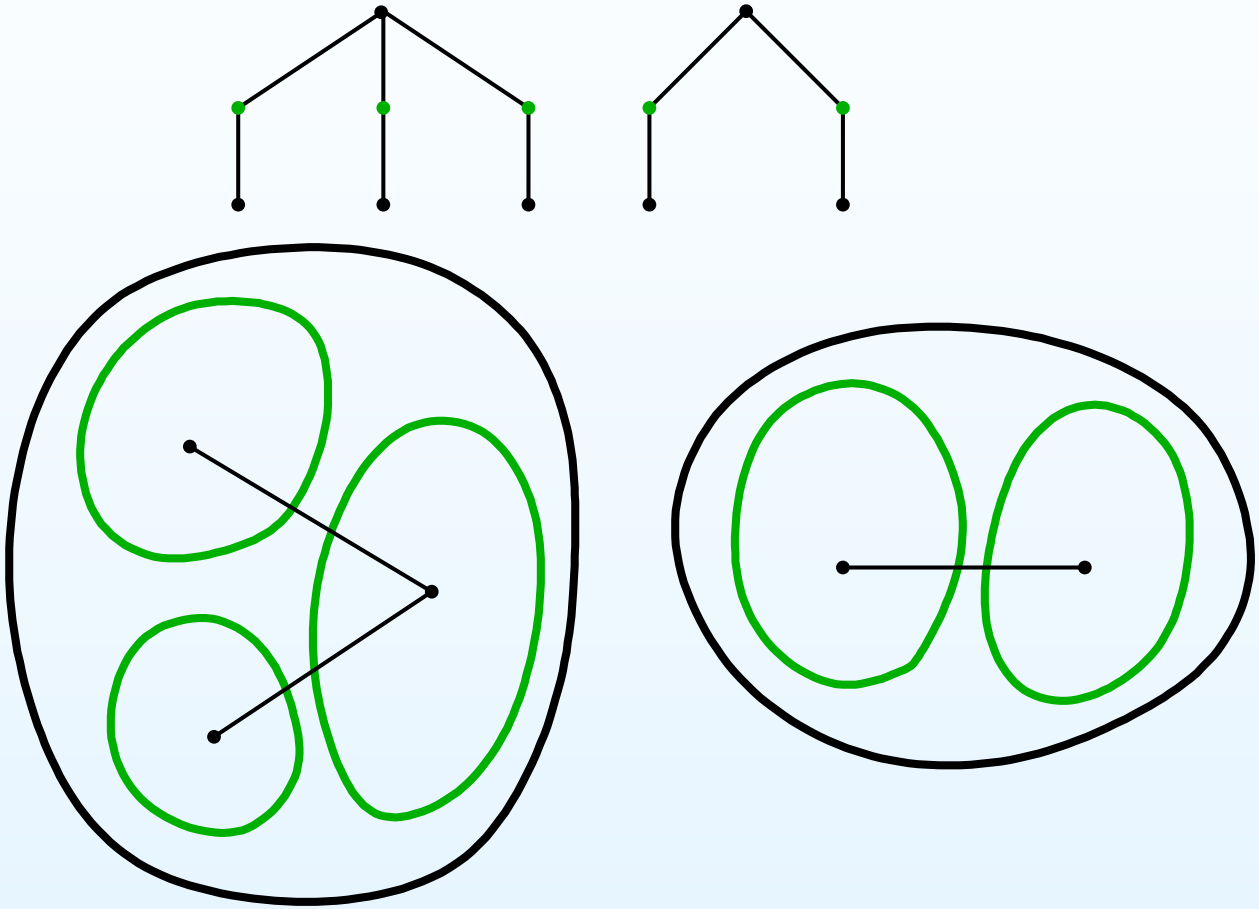
Clusters



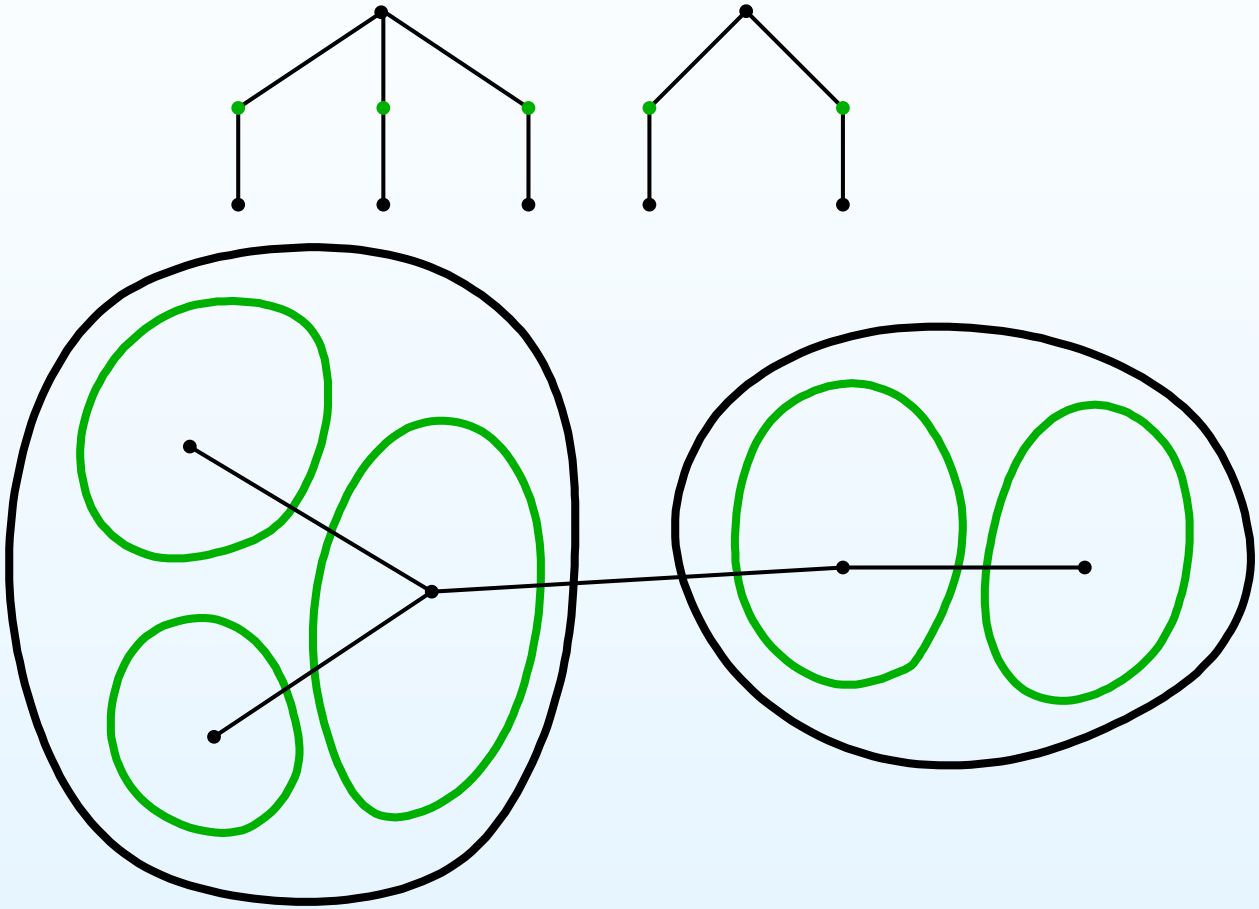
Cluster forest



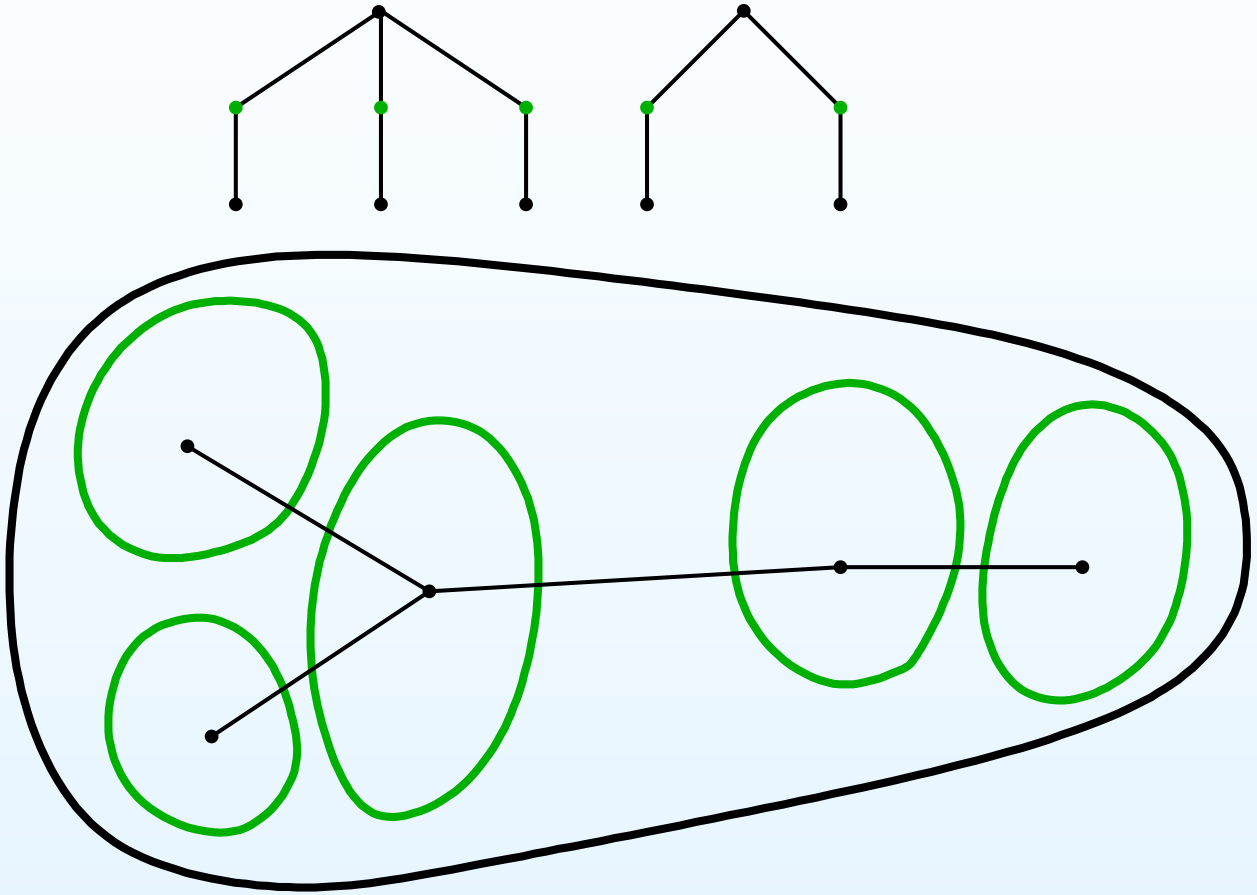
Handling an update $\texttt{insert}(u, v)$



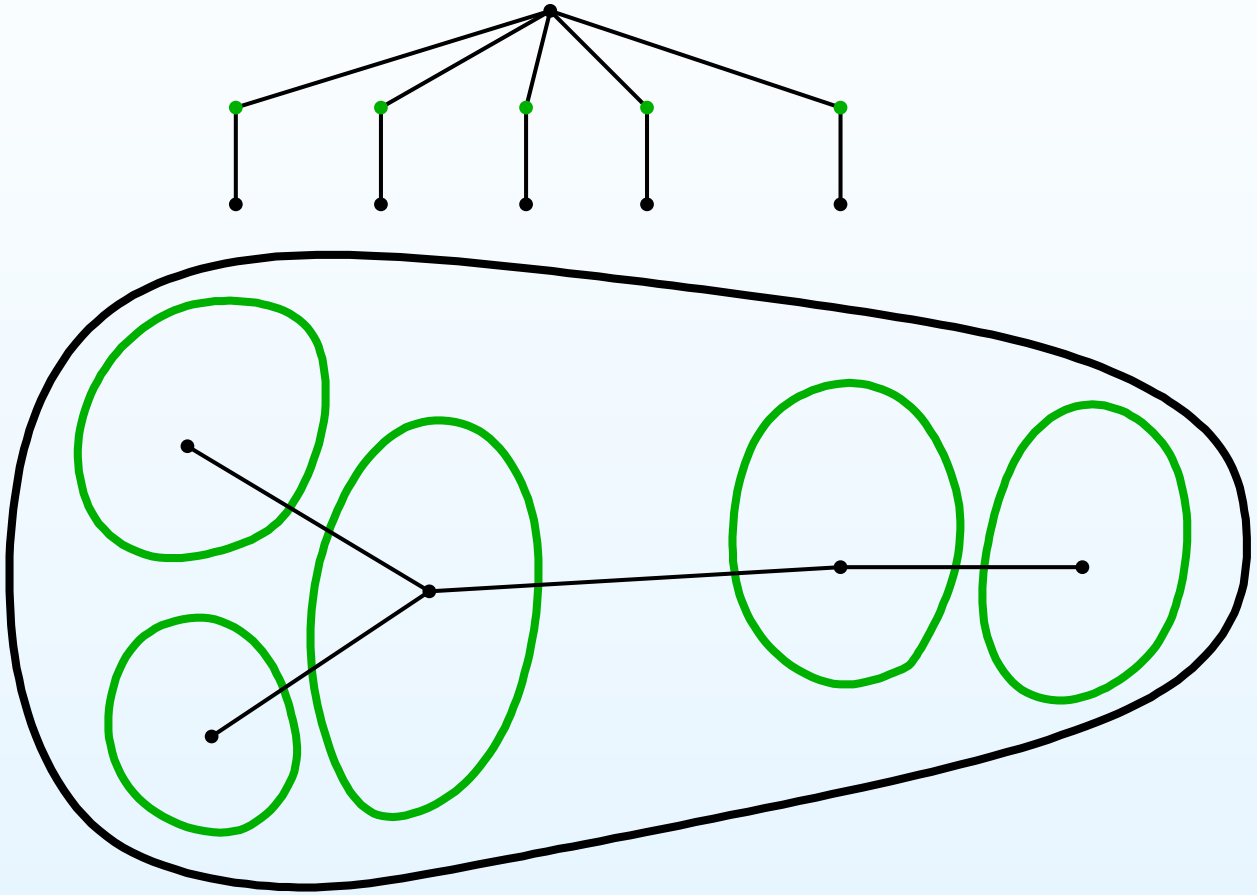
Handling an update $\texttt{insert}(u, v)$



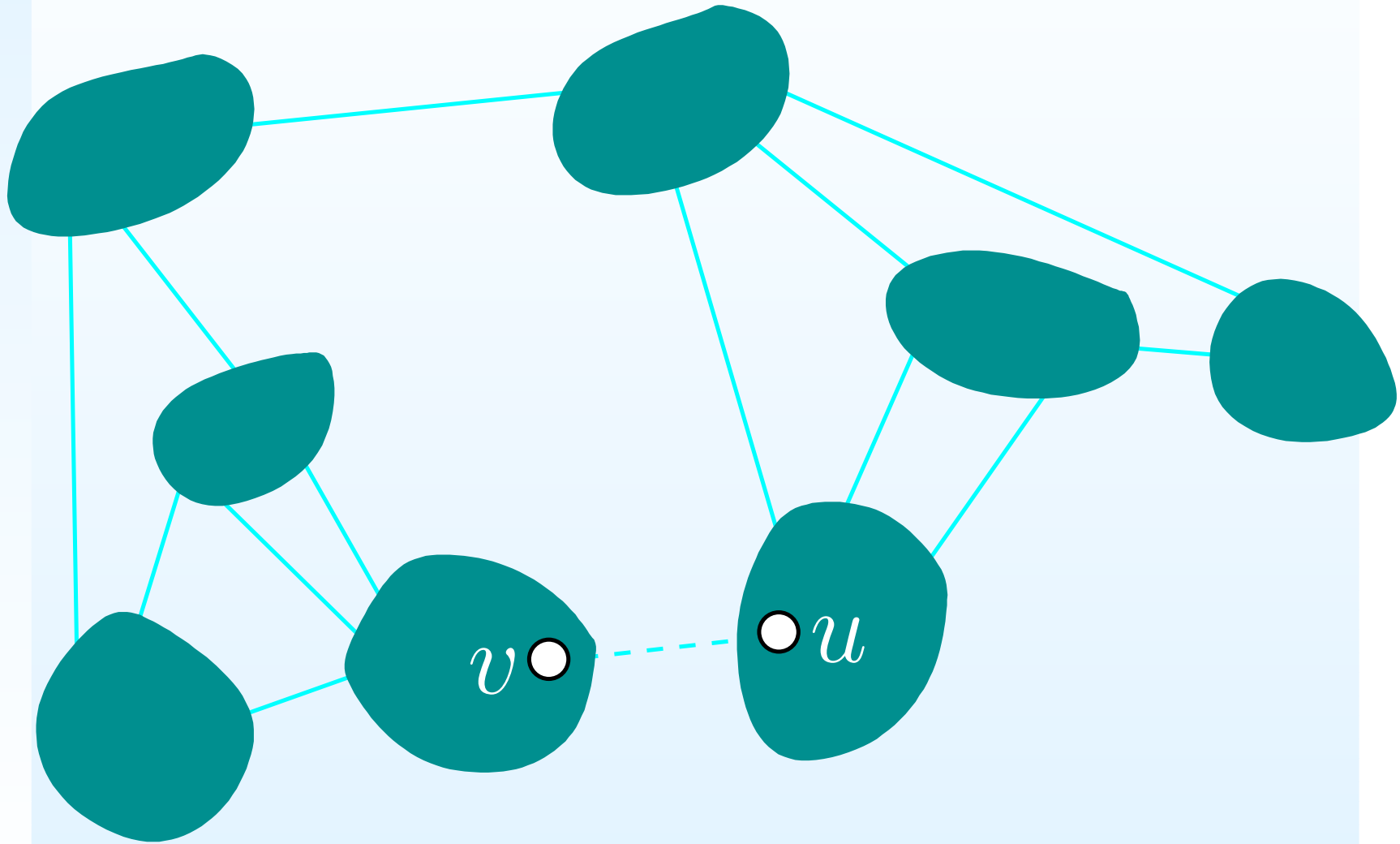
Handling an update $\texttt{insert}(u, v)$



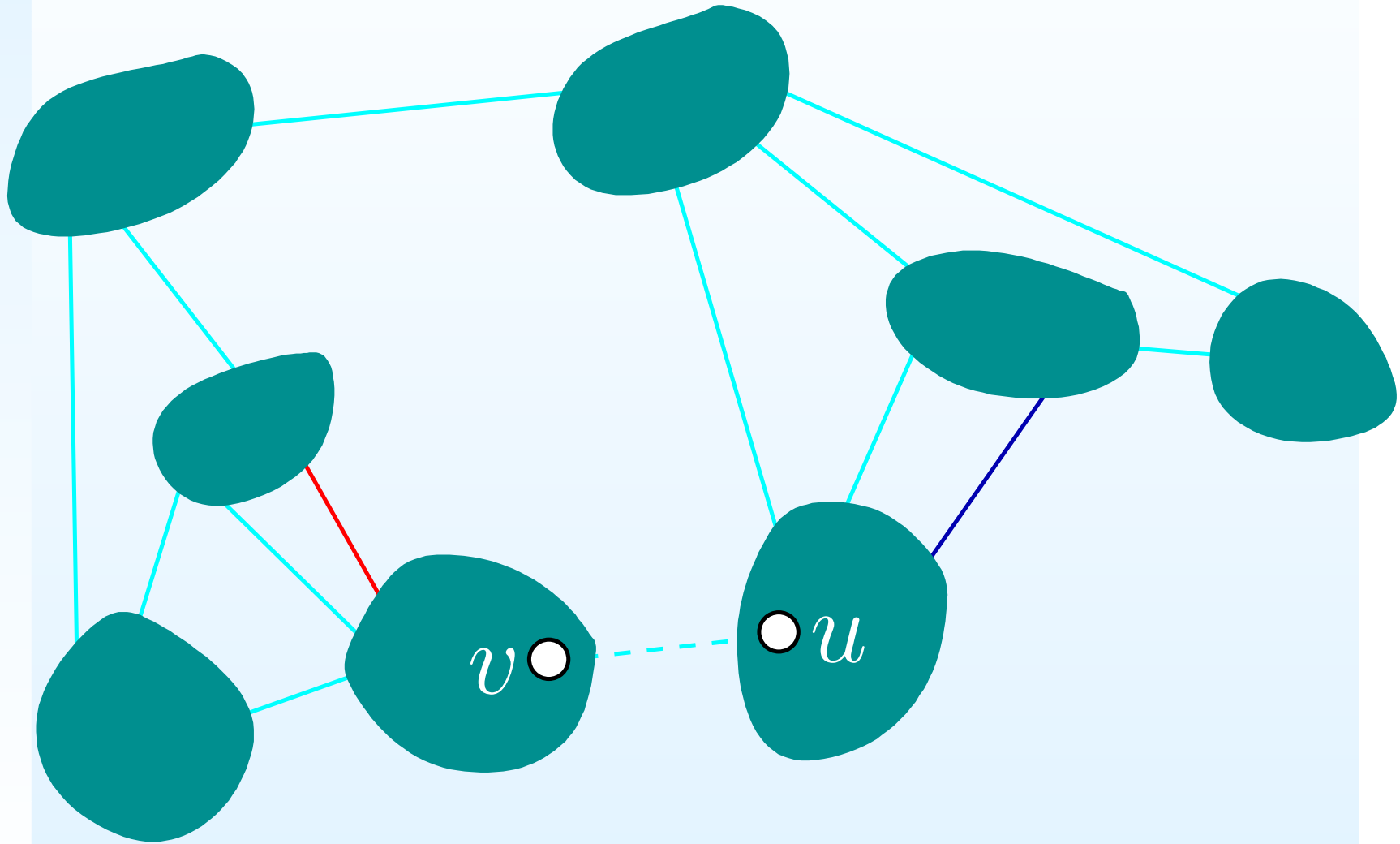
Handling an update $\texttt{insert}(u, v)$



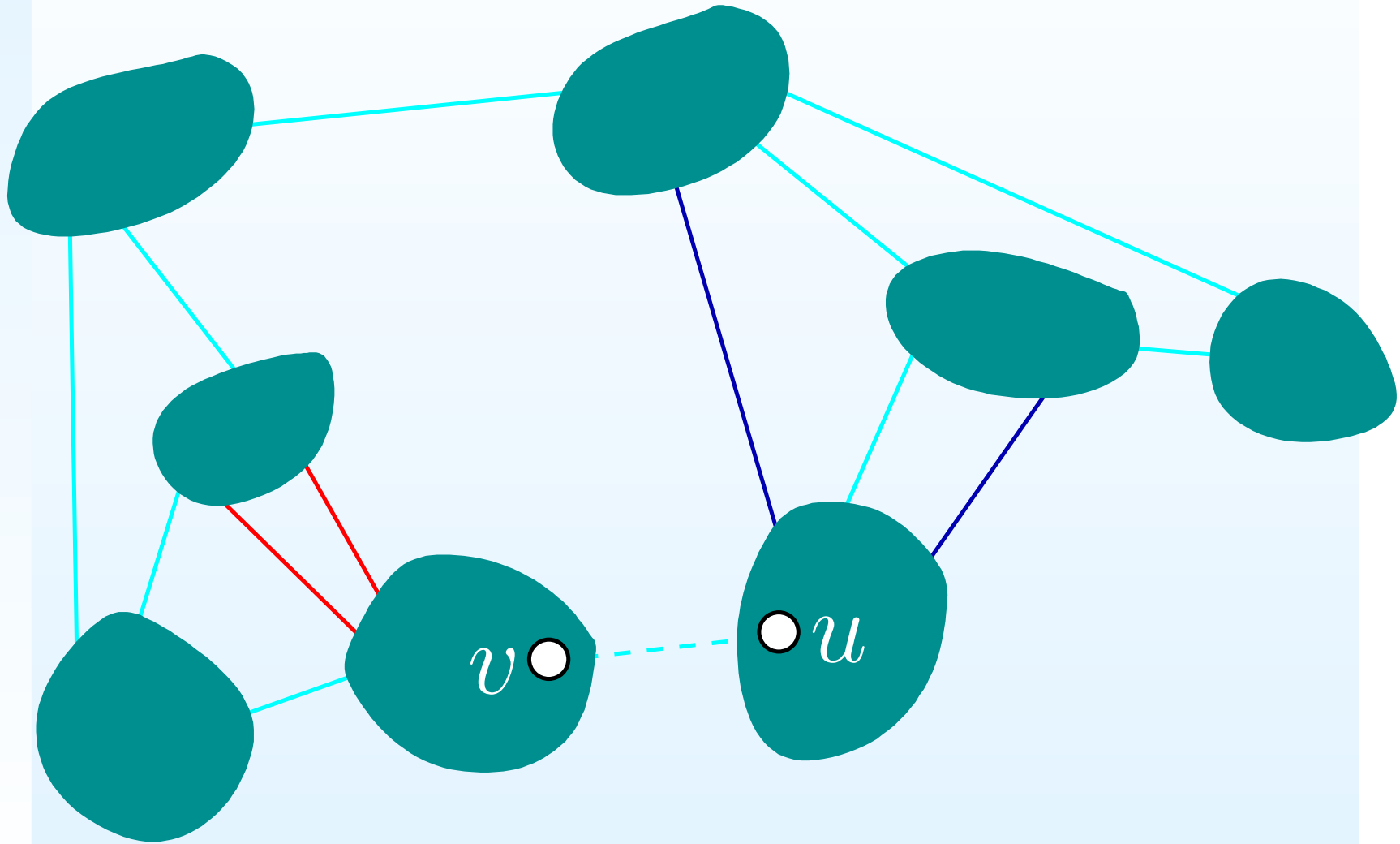
A vertex of M_i is explored by both search procedures



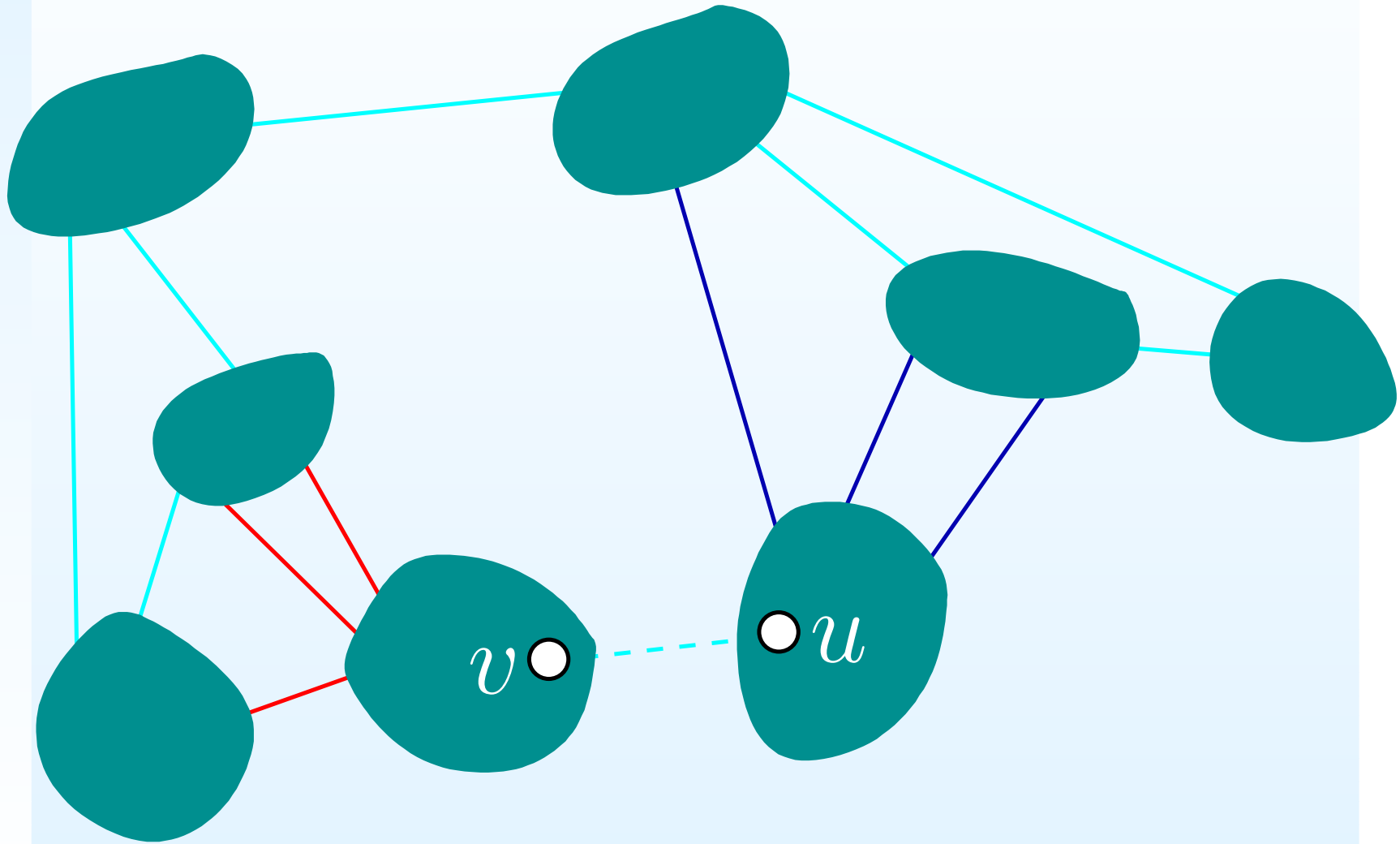
A vertex of M_i is explored by both search procedures



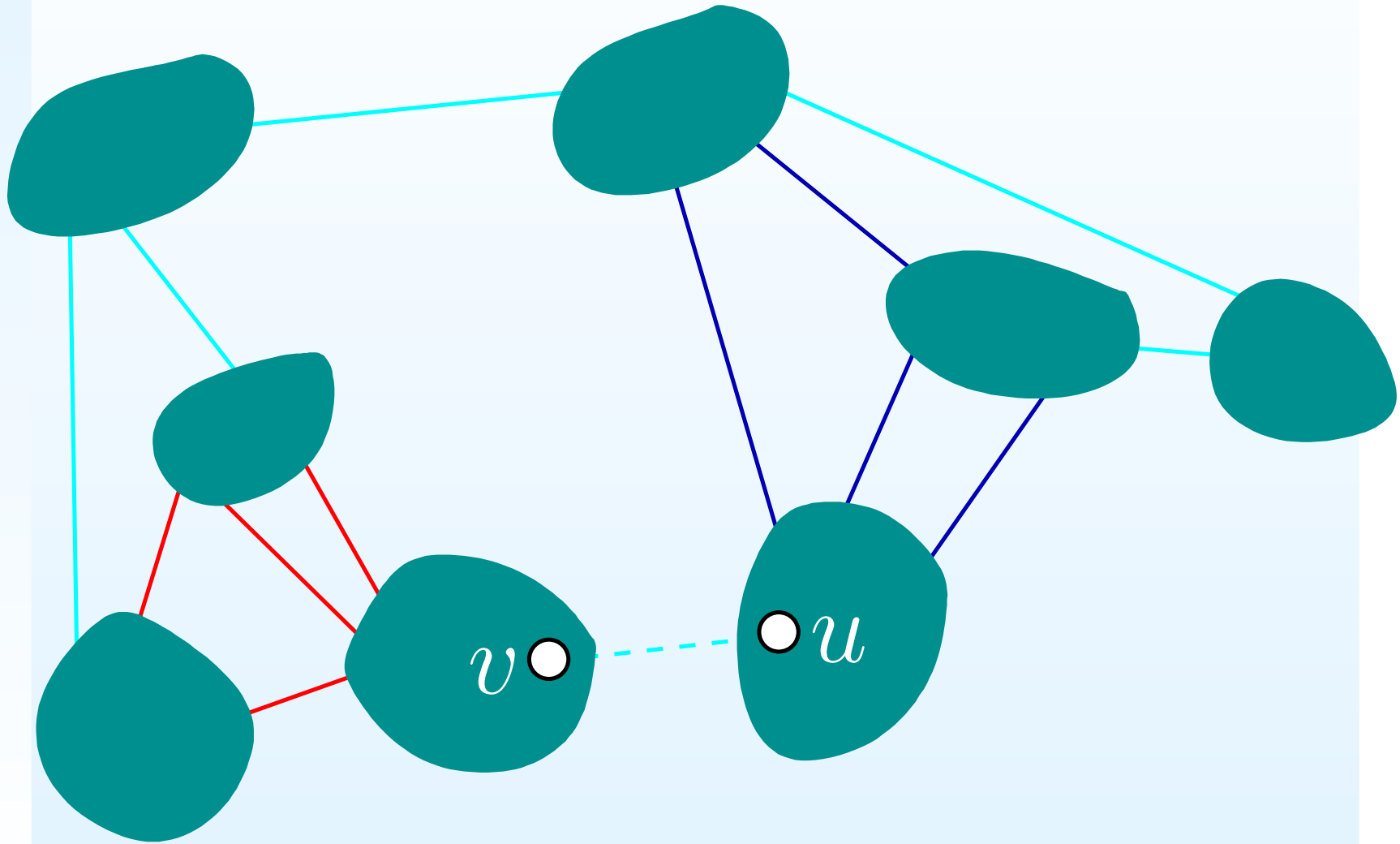
A vertex of M_i is explored by both search procedures



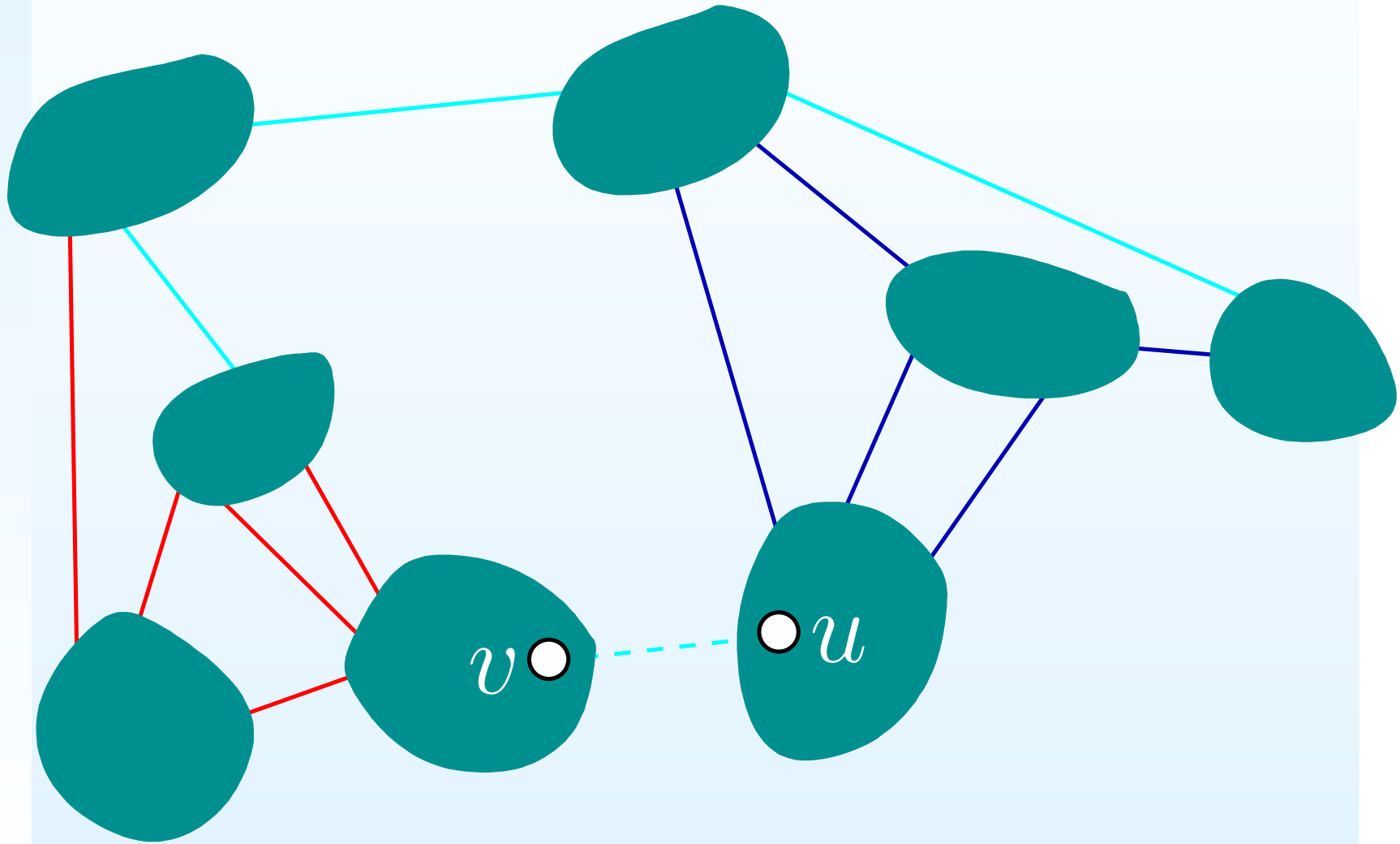
A vertex of M_i is explored by both search procedures



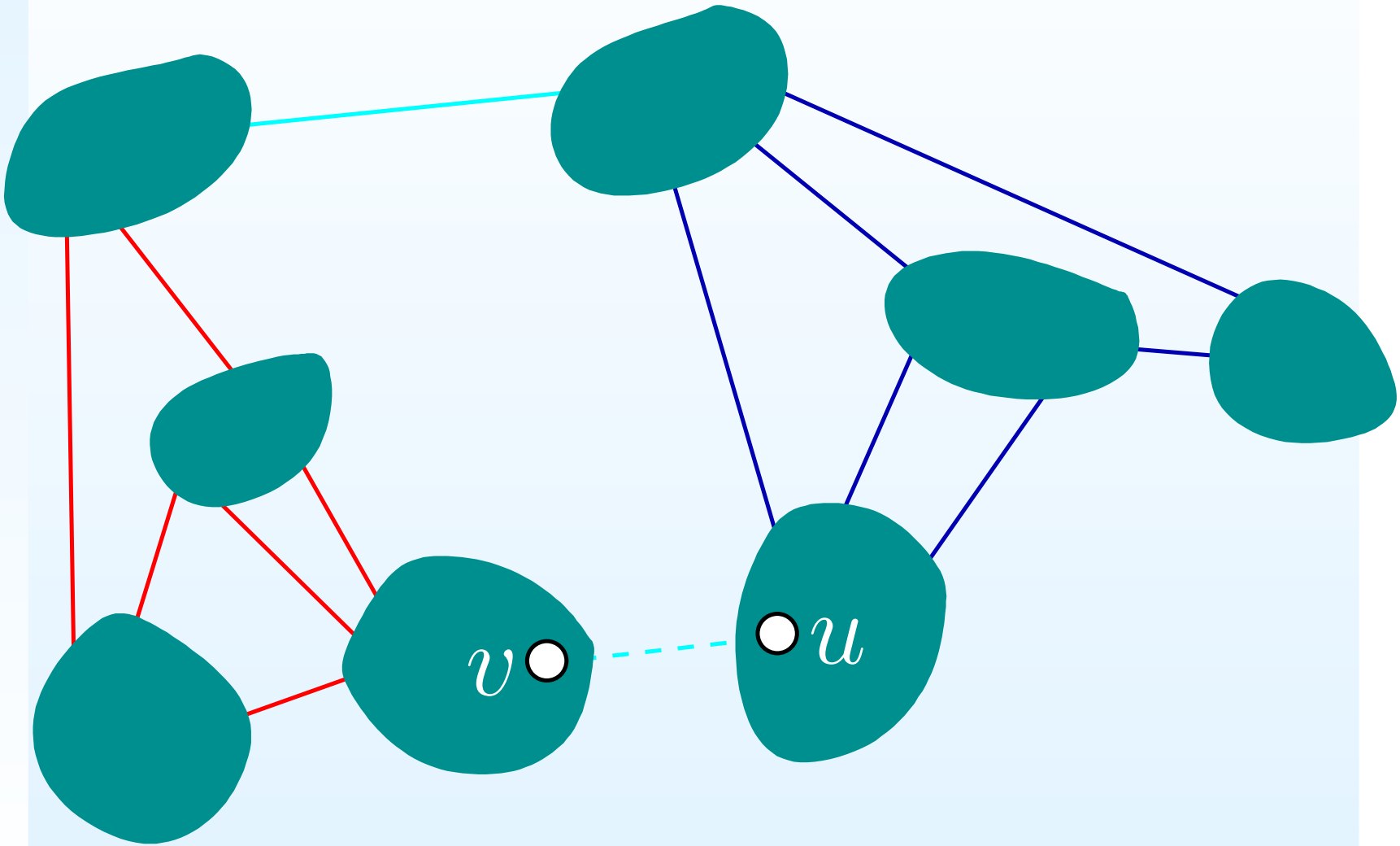
A vertex of M_i is explored by both search procedures



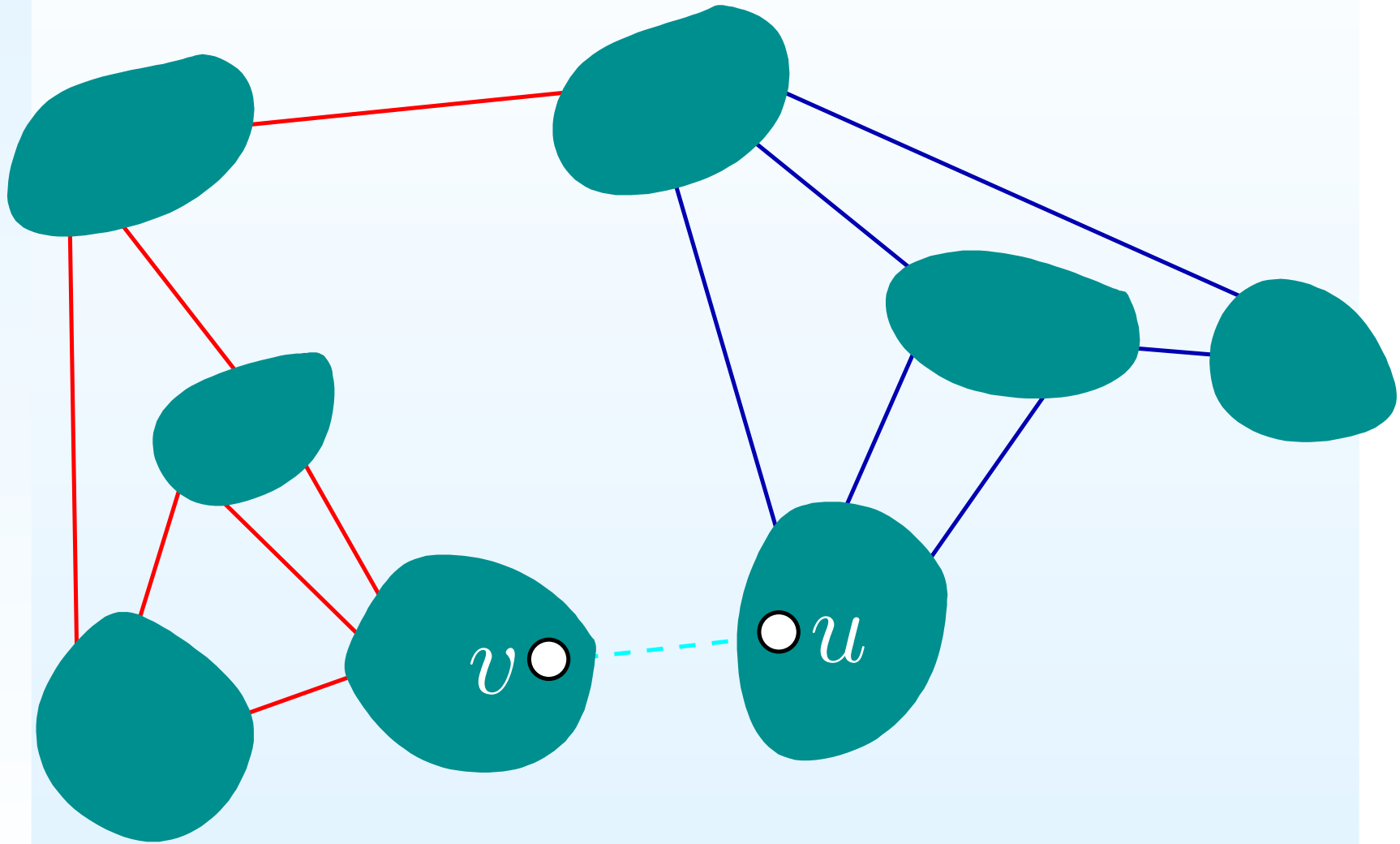
A vertex of M_i is explored by both search procedures



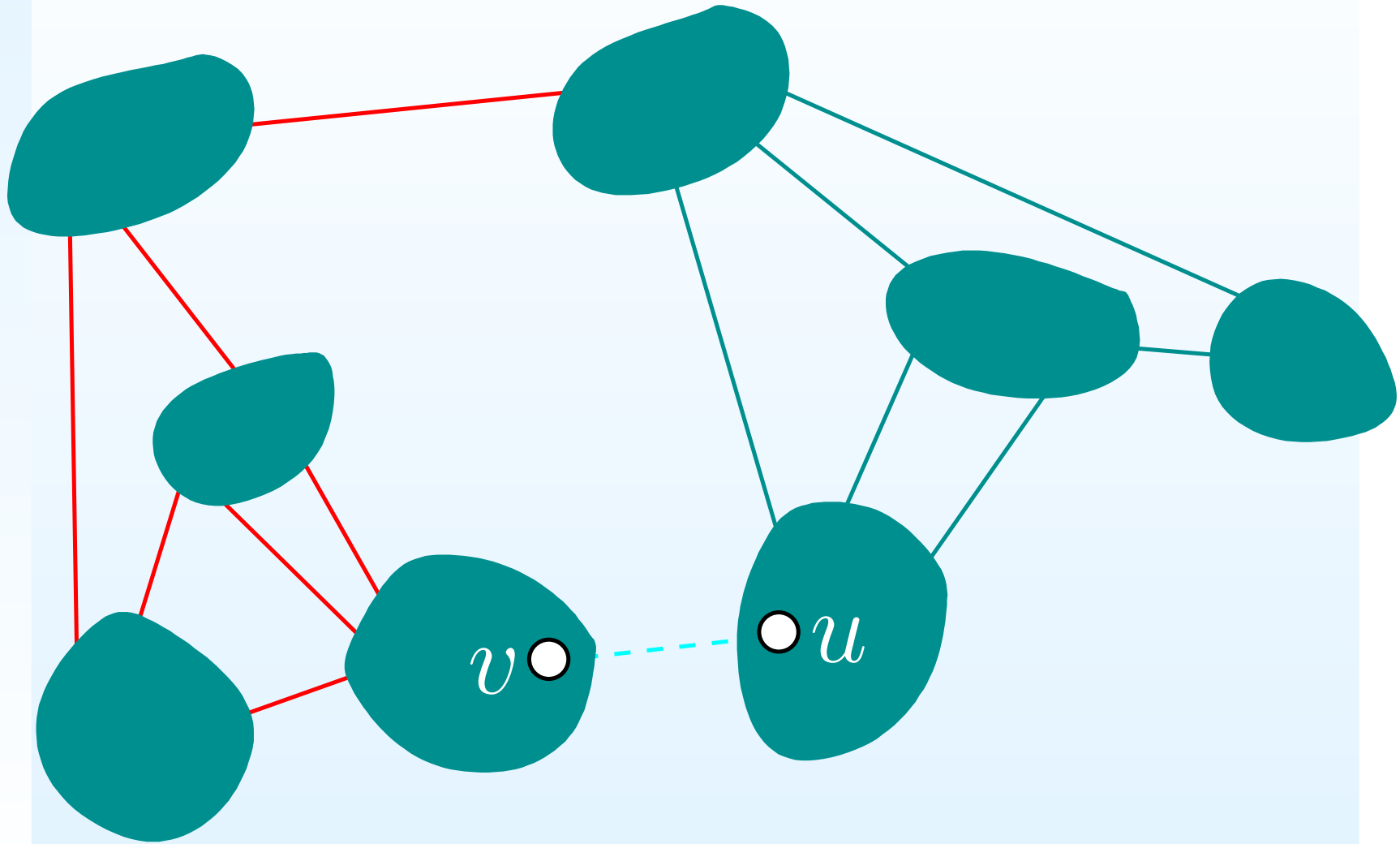
A vertex of M_i is explored by both search procedures



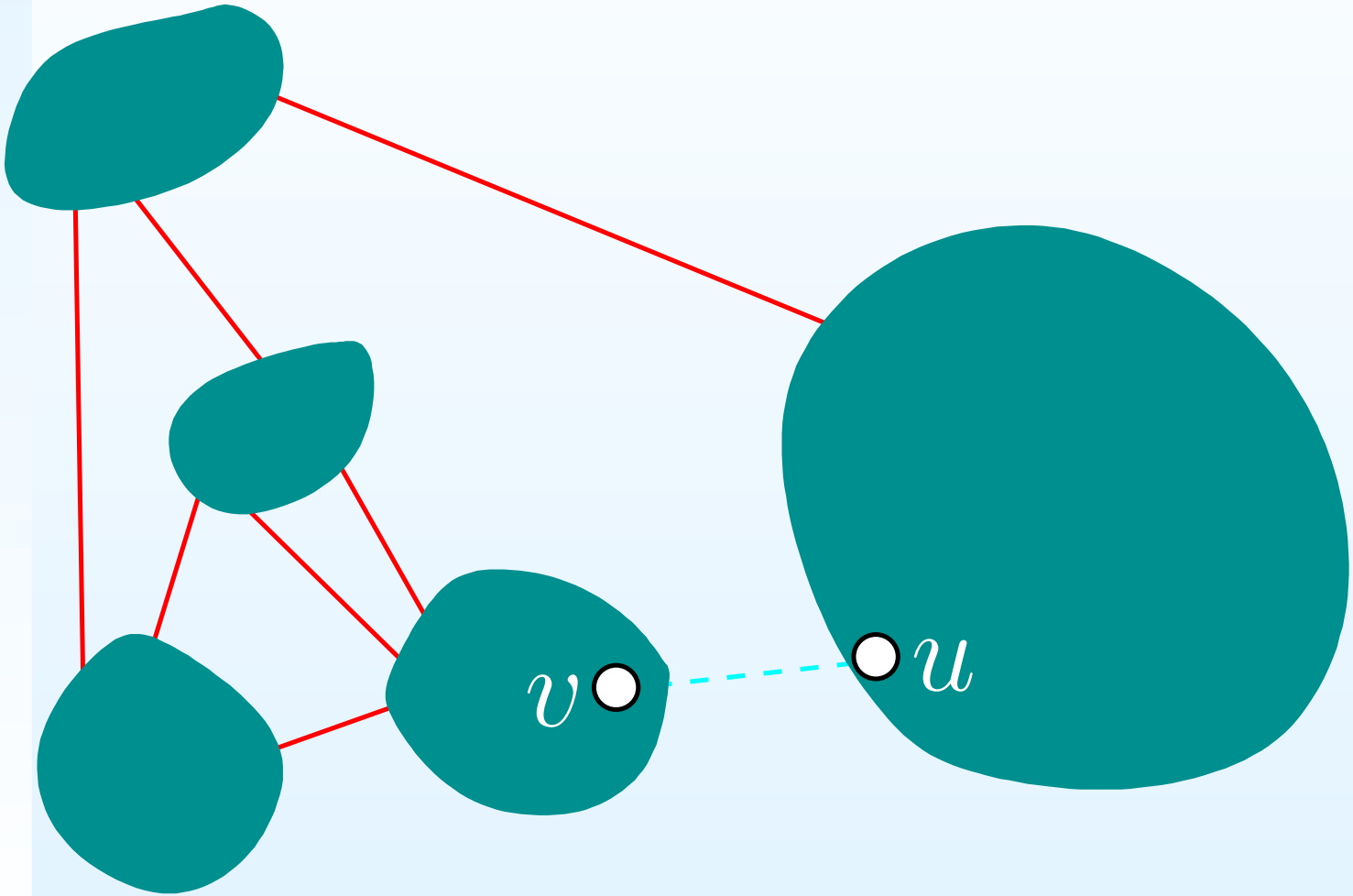
A vertex of M_i is explored by both search procedures



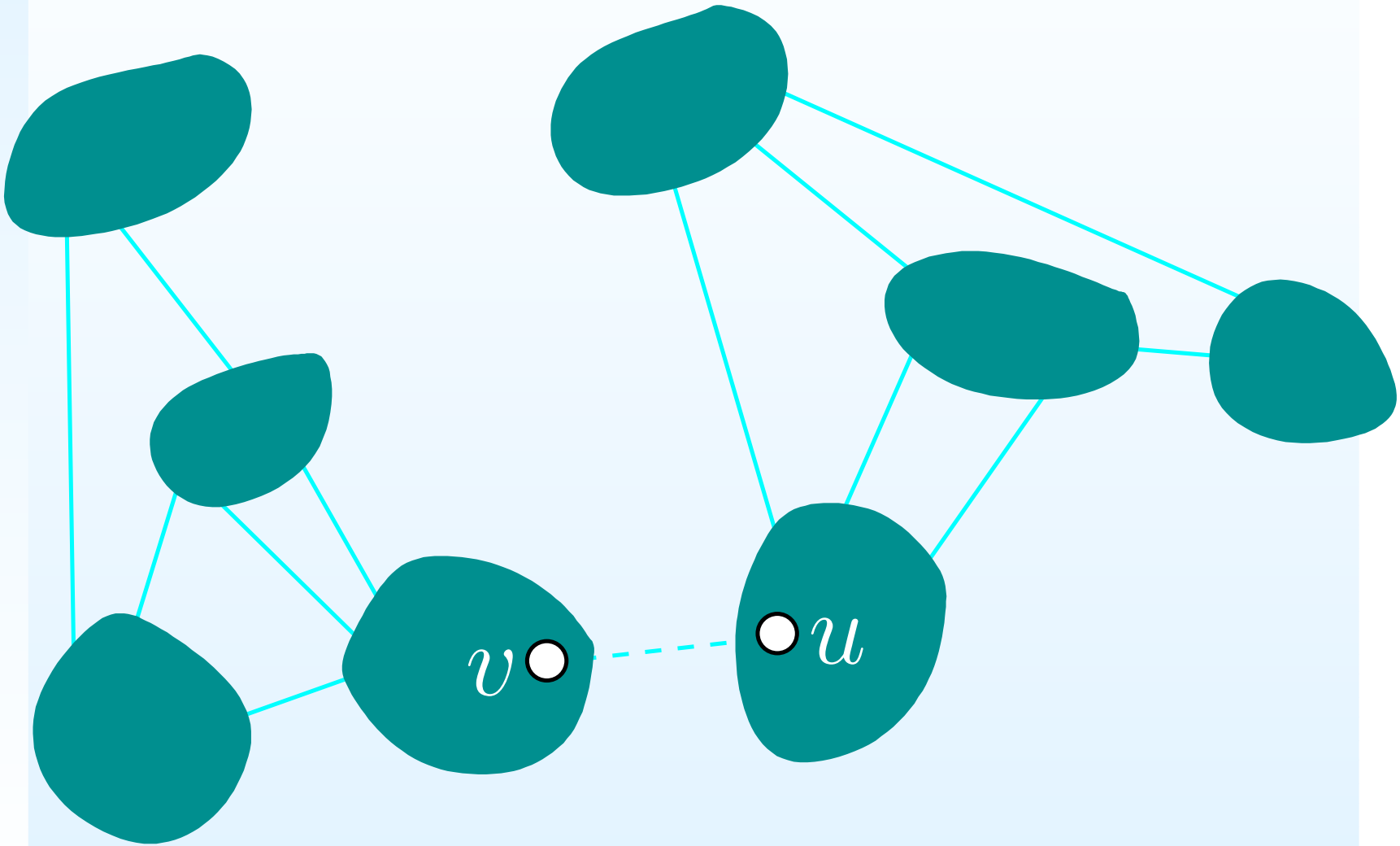
A vertex of M_i is explored by both search procedures



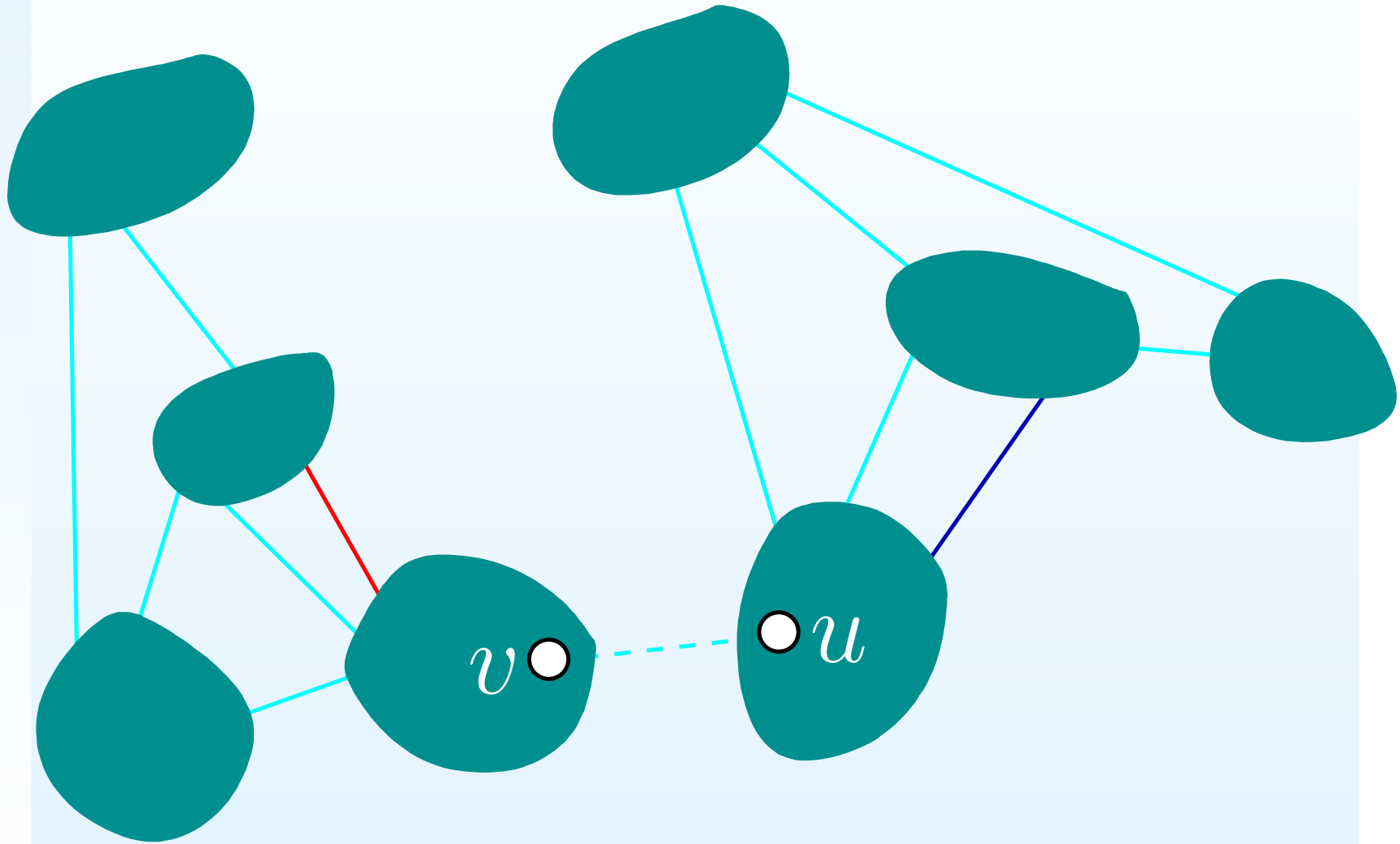
A vertex of M_i is explored by both search procedures



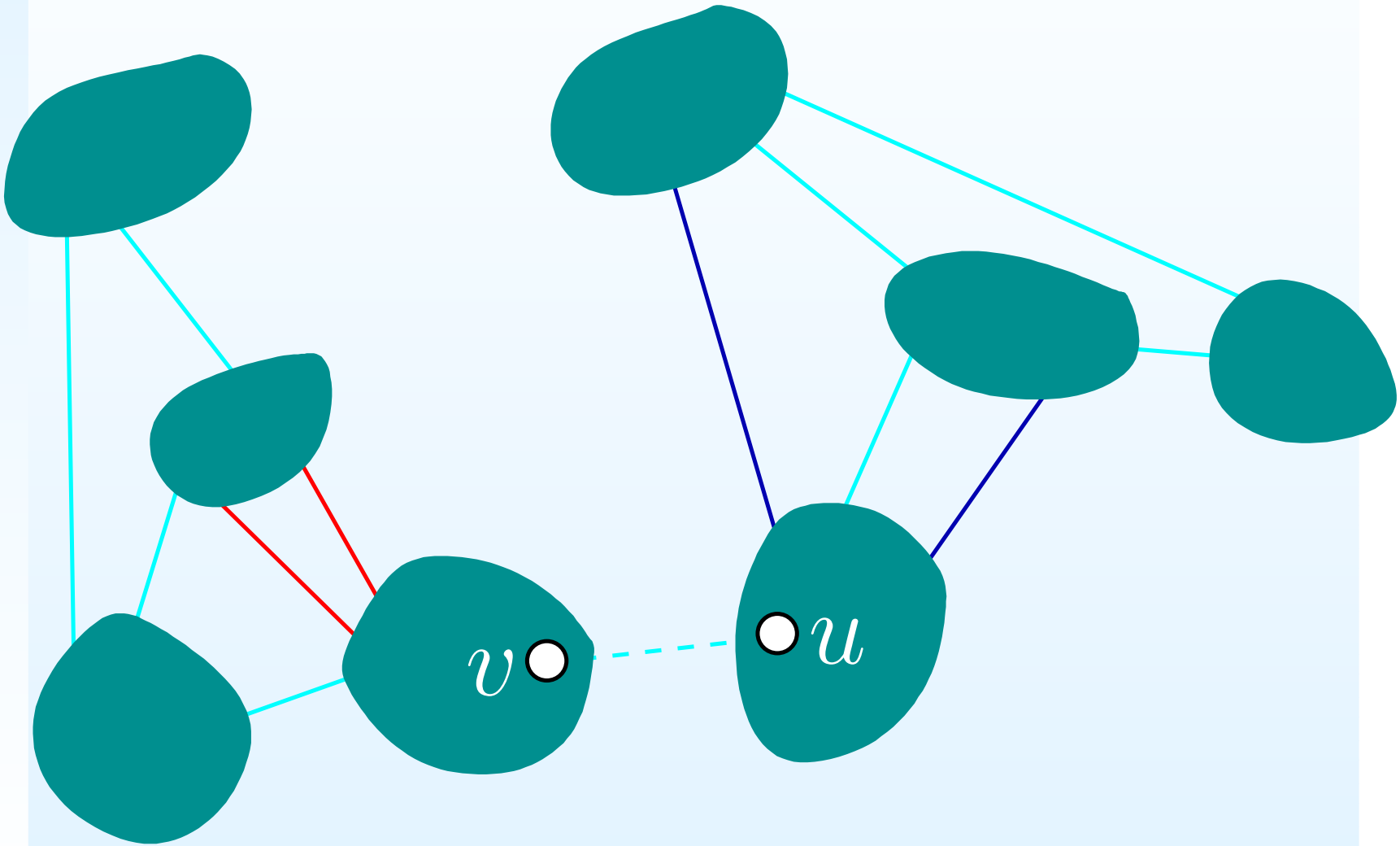
A search procedure has no more edges to explore



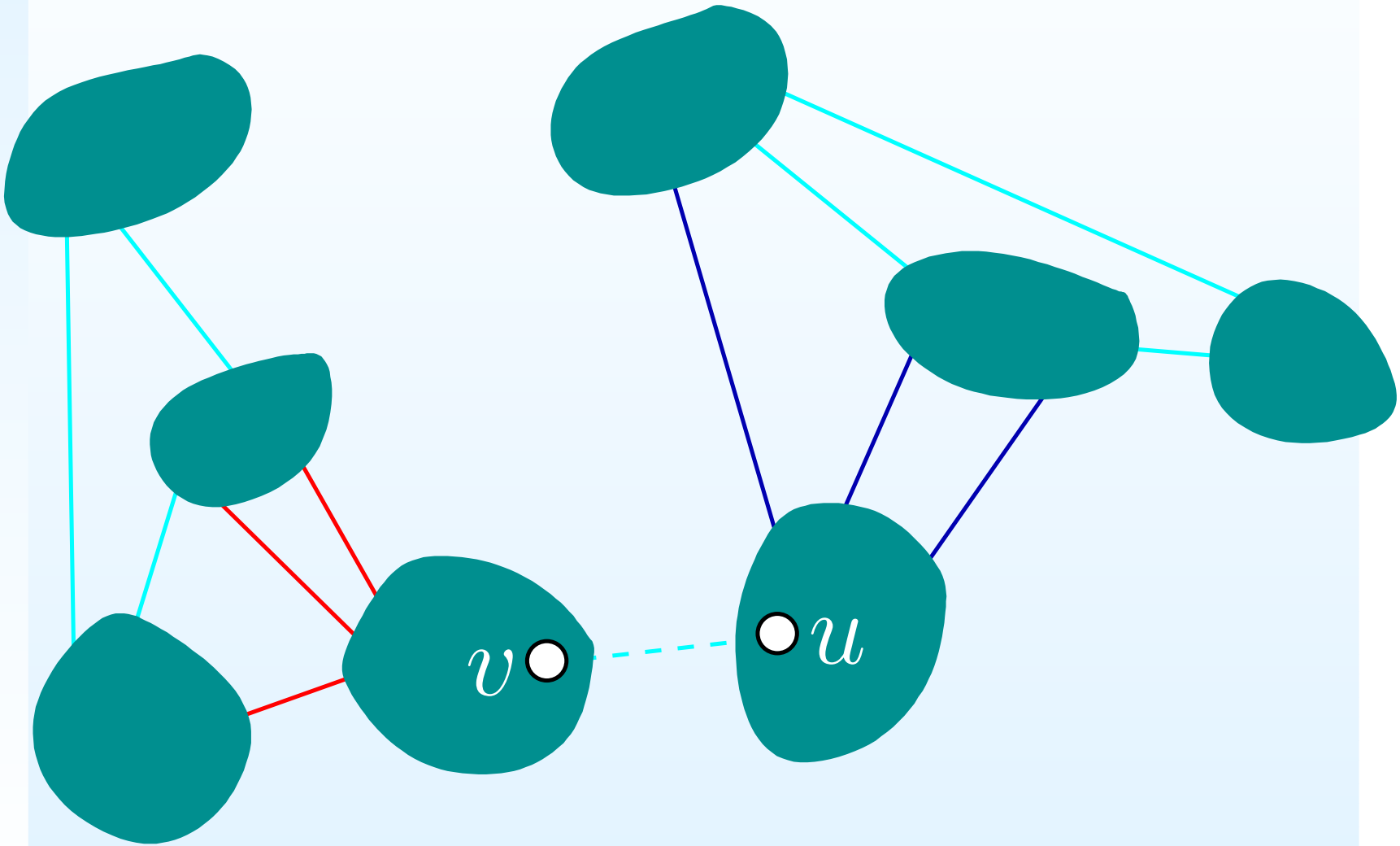
A search procedure has no more edges to explore



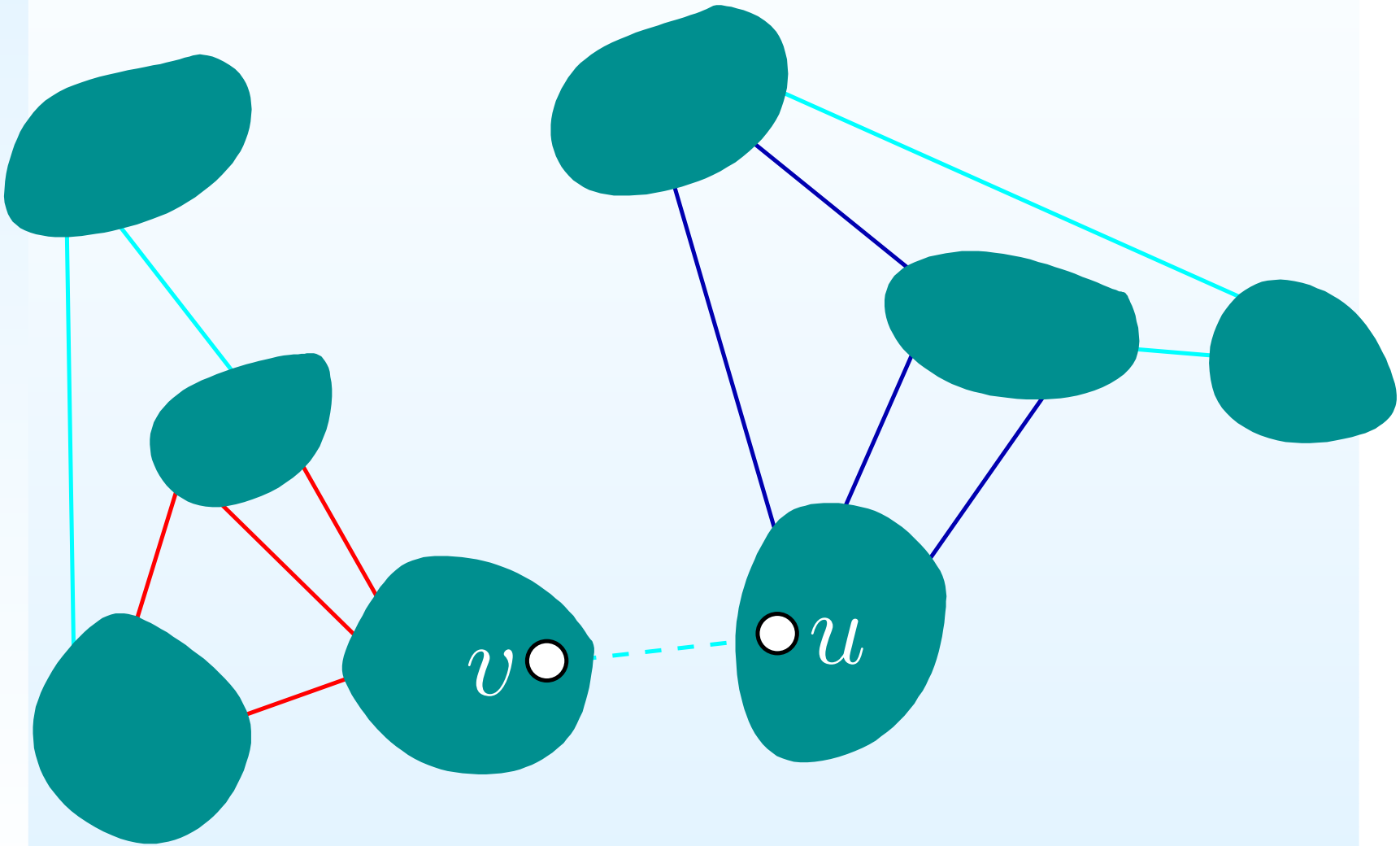
A search procedure has no more edges to explore



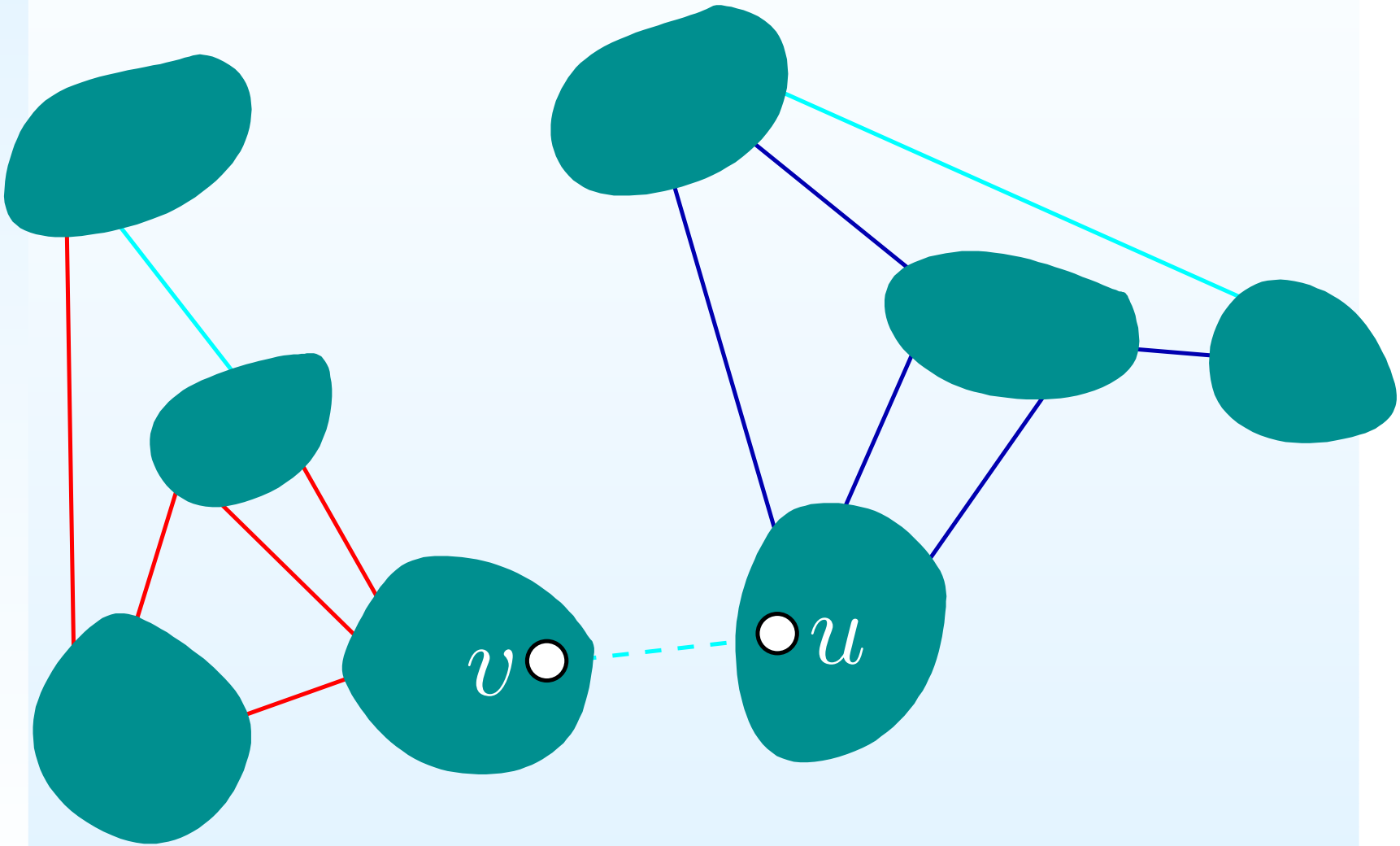
A search procedure has no more edges to explore



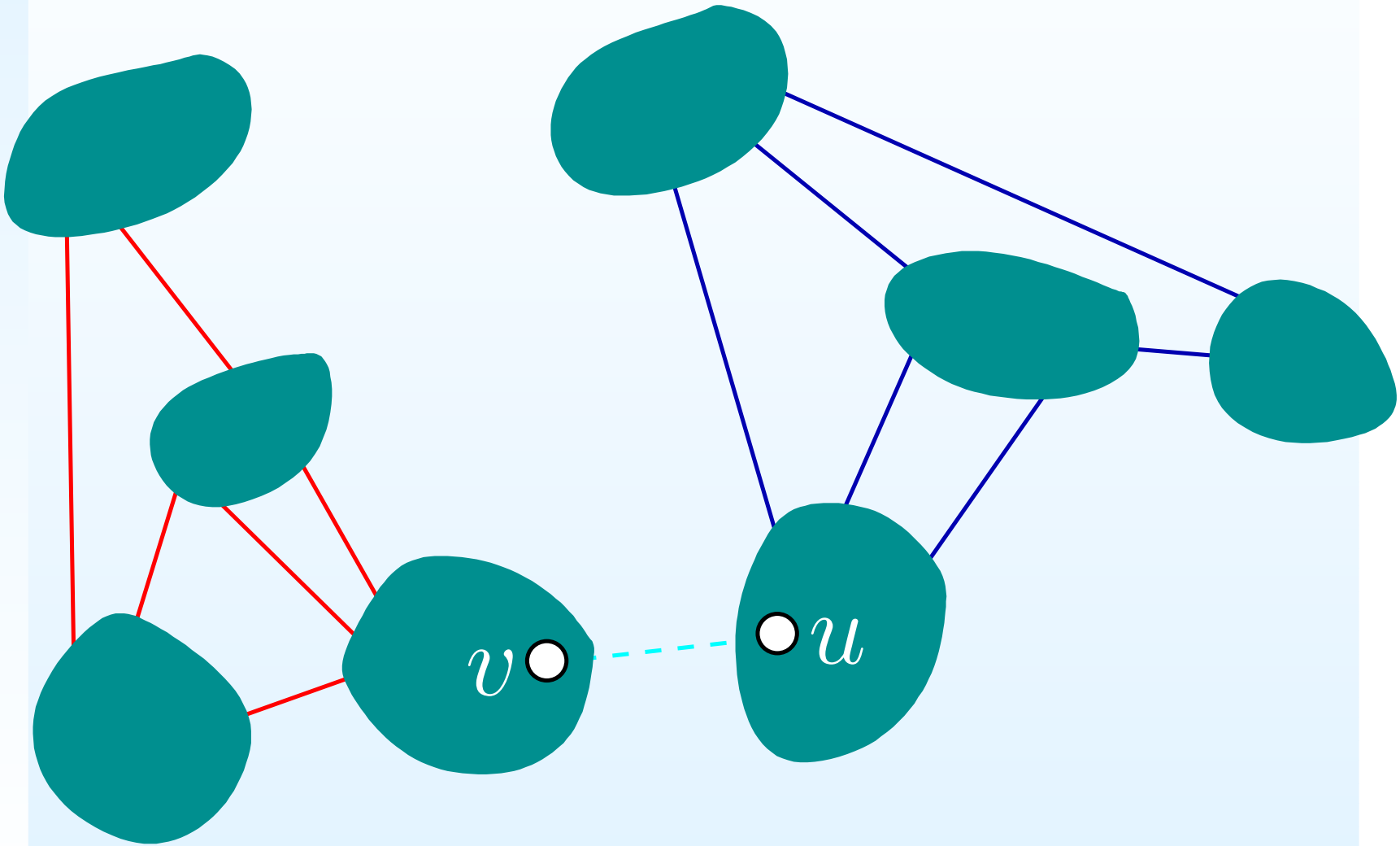
A search procedure has no more edges to explore



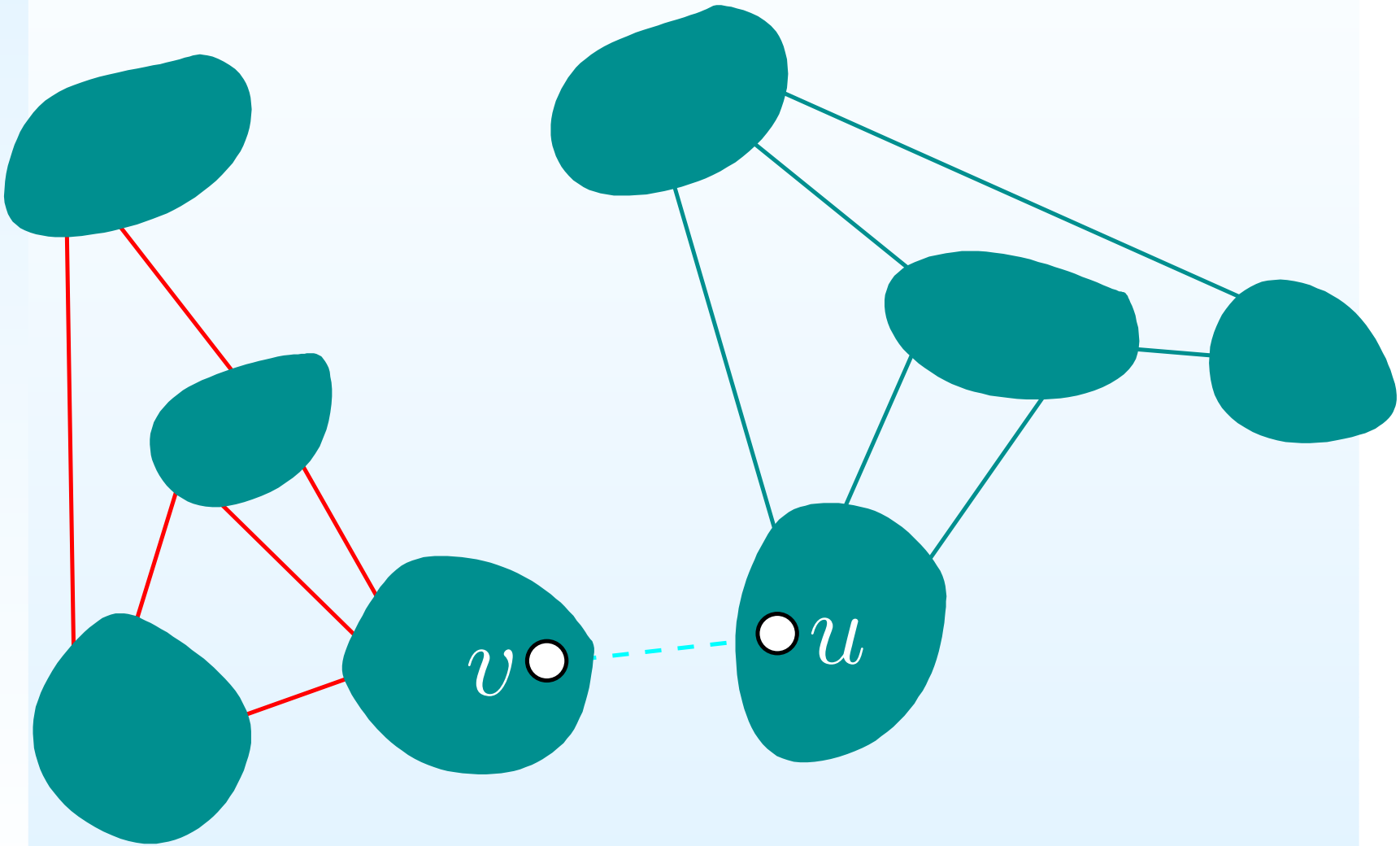
A search procedure has no more edges to explore



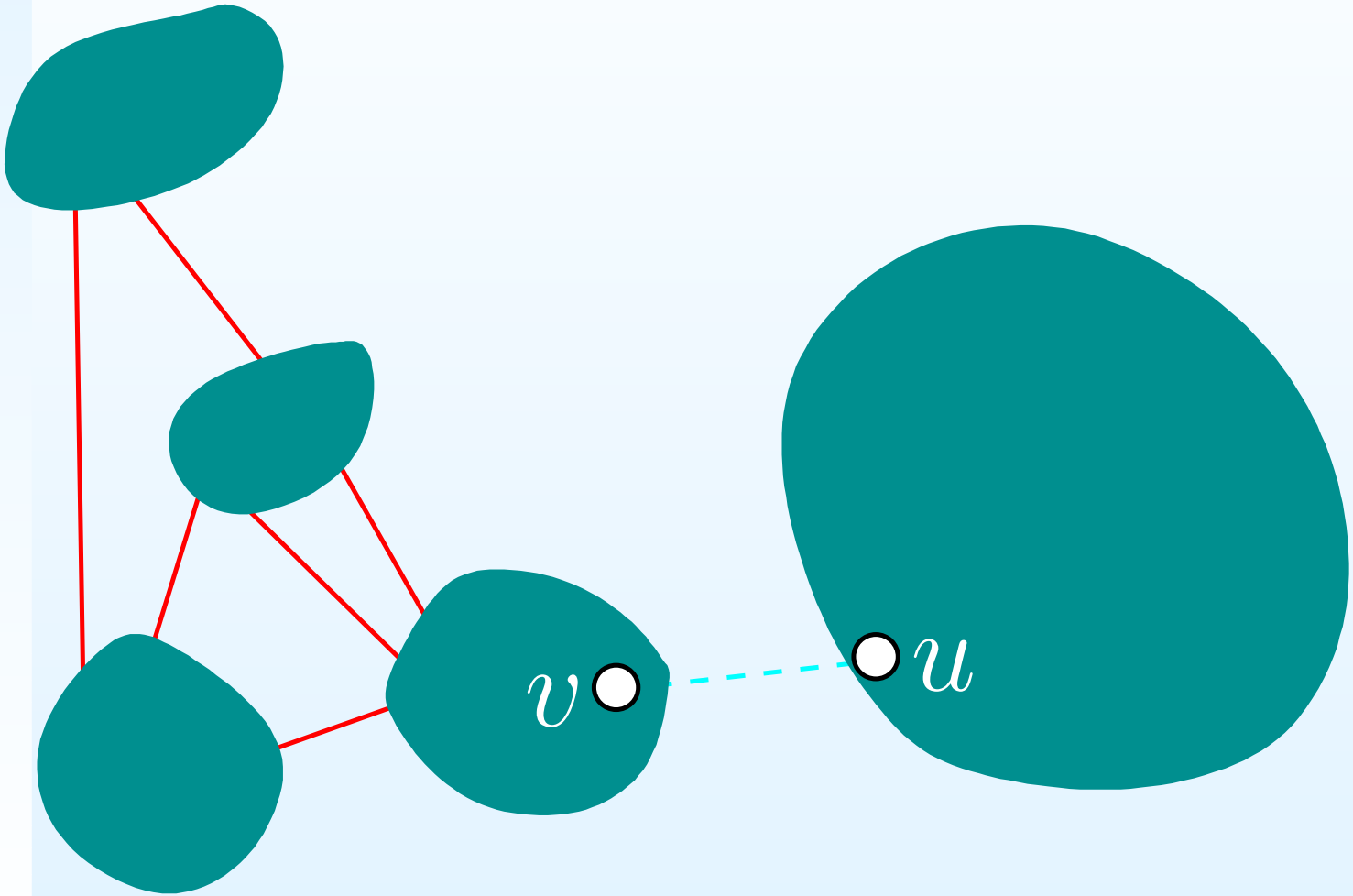
A search procedure has no more edges to explore



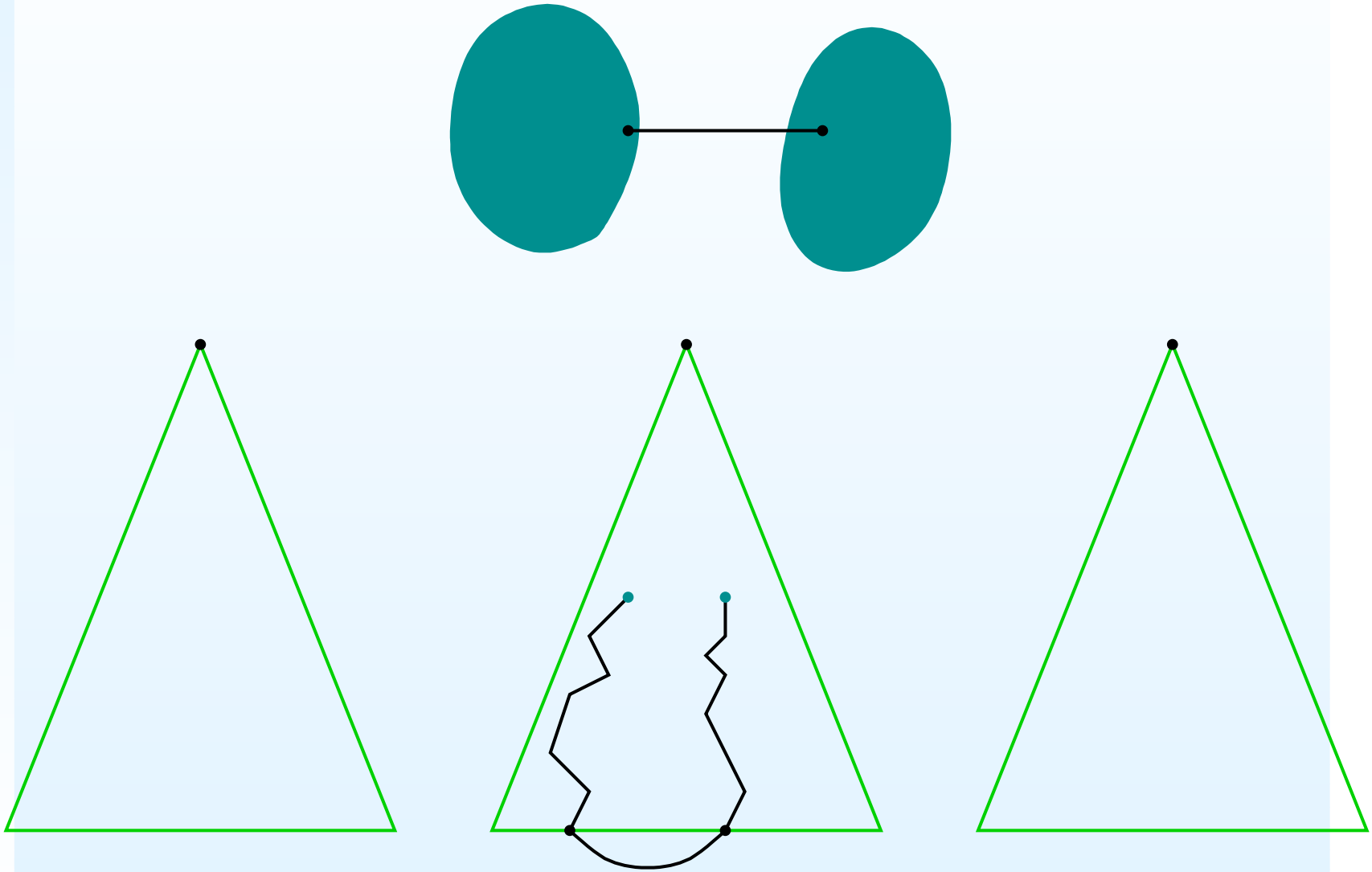
A search procedure has no more edges to explore



A search procedure has no more edges to explore



Traversing a single graph edge

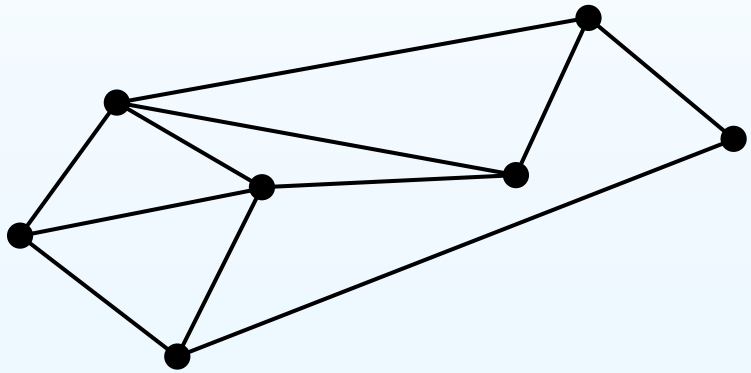


Spanning tree of a graph

- A *spanning tree* of a connected undirected graph G is a tree contained in G which contains all vertices of G:

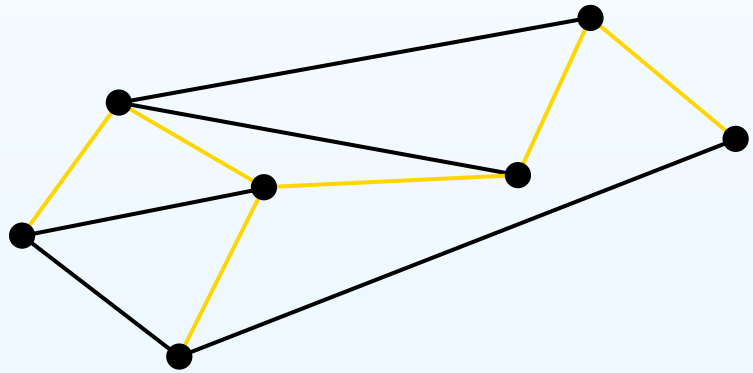
Spanning tree of a graph

- A *spanning tree* of a connected undirected graph G is a tree contained in G which contains all vertices of G:



Spanning tree of a graph

- A *spanning tree* of a connected undirected graph G is a tree contained in G which contains all vertices of G:

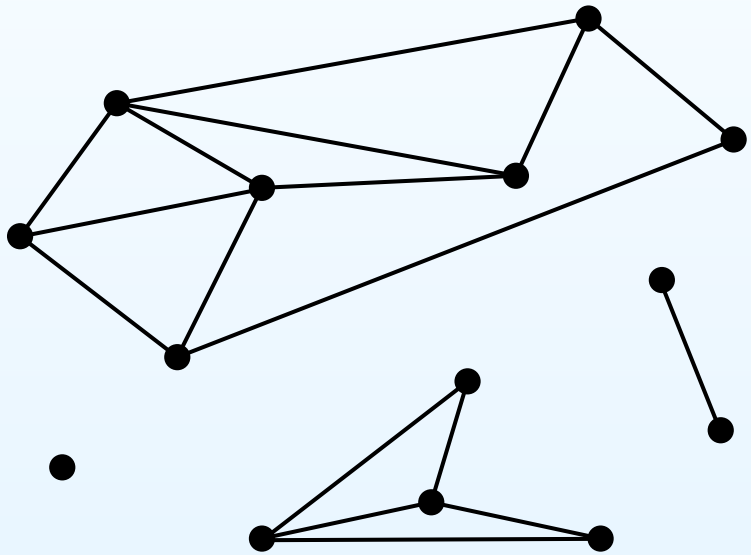


Spanning forest of a graph

- A *spanning forest* of a graph G consists of a spanning tree of each connected component of G:

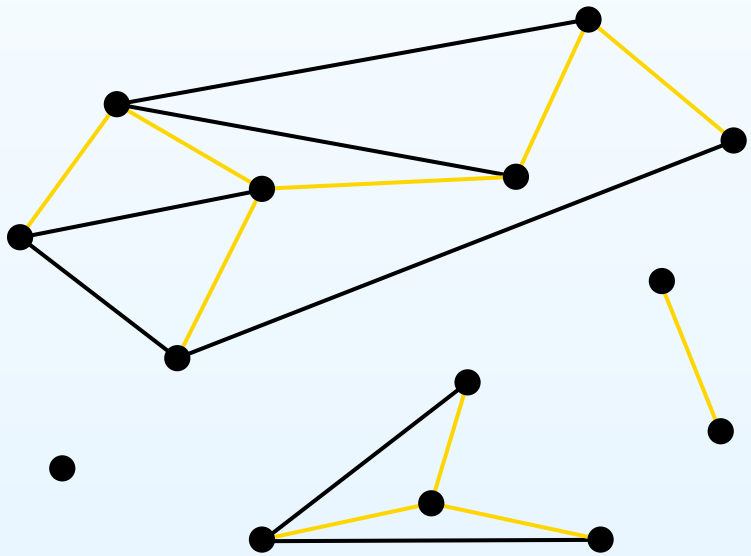
Spanning forest of a graph

- A *spanning forest* of a graph G consists of a spanning tree of each connected component of G:



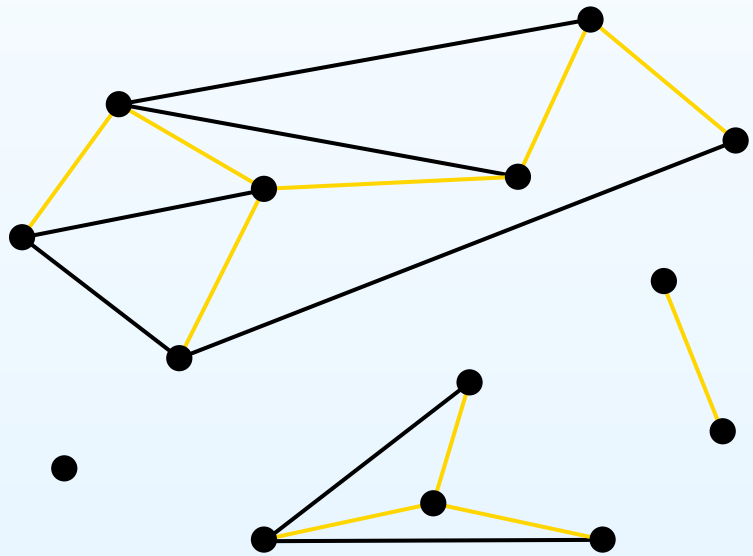
Spanning forest of a graph

- A *spanning forest* of a graph G consists of a spanning tree of each connected component of G:



Spanning forest of a graph

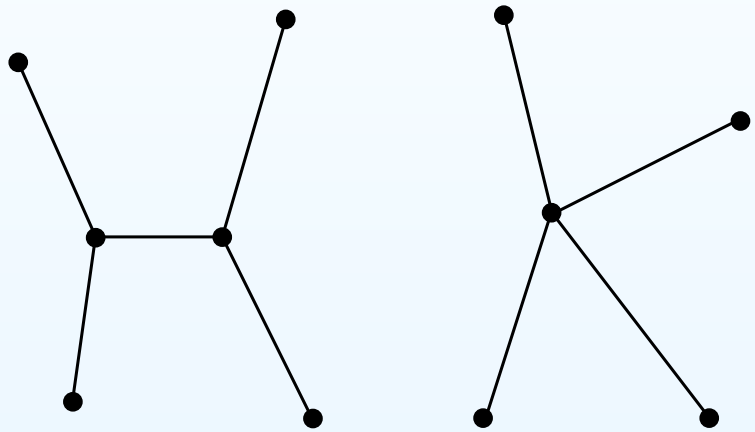
- A *spanning forest* of a graph G consists of a spanning tree of each connected component of G:



- If we can efficiently maintain a spanning forest of a dynamic graph G then we also have an efficient data structure for dynamic connectivity

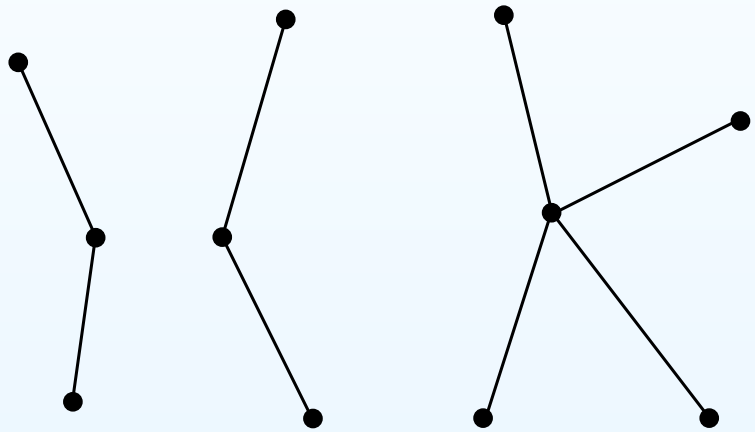
Dynamic Trees

- Suppose we want a data structure maintaining some information about a dynamic forest where there is *no underlying graph*:



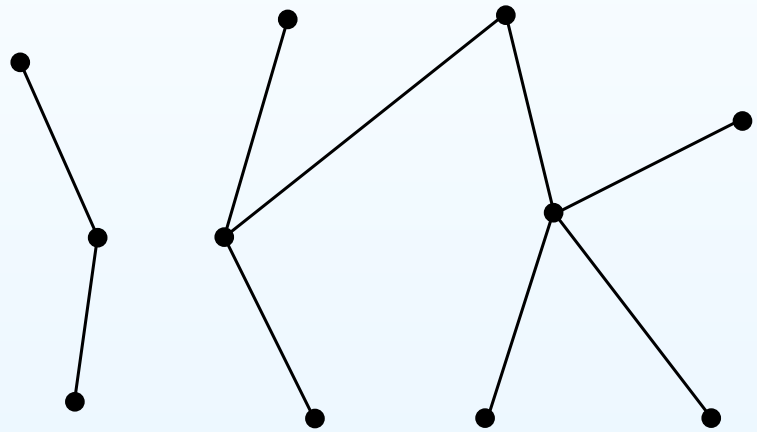
Dynamic Trees

- Suppose we want a data structure maintaining some information about a dynamic forest where there is *no underlying graph*:



Dynamic Trees

- Suppose we want a data structure maintaining some information about a dynamic forest where there is *no underlying graph*:



Dynamic Trees

- The data structure should be able to:

Dynamic Trees

- The data structure should be able to:
  - Insert/delete an edge

Dynamic Trees

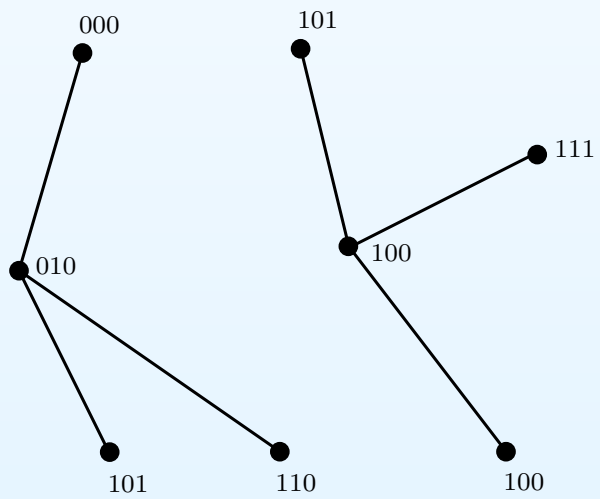
- The data structure should be able to:
  - Insert/delete an edge
  - Update the key value of a vertex

Dynamic Trees

- The data structure should be able to:
  - Insert/delete an edge
  - Update the key value of a vertex
  - Report the sum or the bitwise AND/OR/XOR of all vertices in the tree containing a given vertex

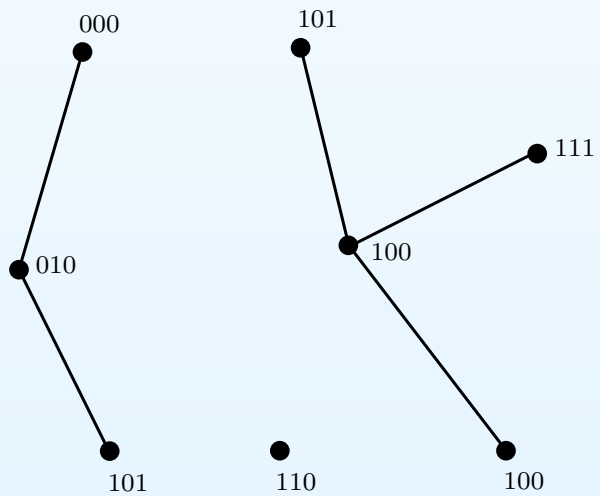
Dynamic Trees

- The data structure should be able to:
  - Insert/delete an edge
  - Update the key value of a vertex
  - Report the sum or the bitwise AND/OR/XOR of all vertices in the tree containing a given vertex



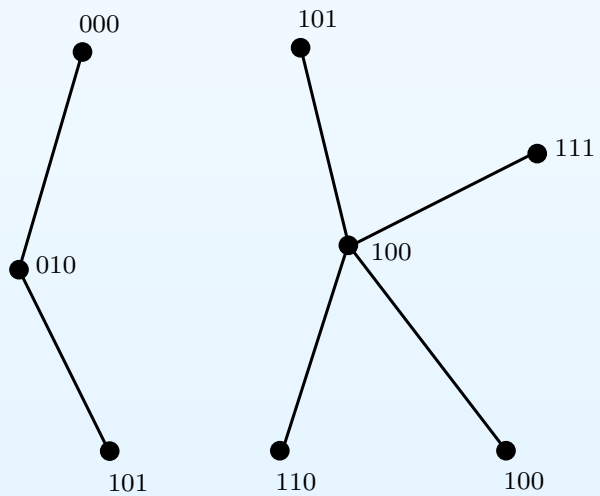
Dynamic Trees

- The data structure should be able to:
  - Insert/delete an edge
  - Update the key value of a vertex
  - Report the sum or the bitwise AND/OR/XOR of all vertices in the tree containing a given vertex



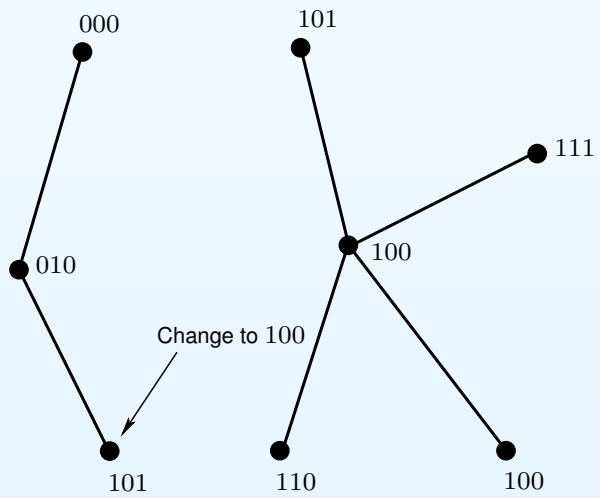
Dynamic Trees

- The data structure should be able to:
  - Insert/delete an edge
  - Update the key value of a vertex
  - Report the sum or the bitwise AND/OR/XOR of all vertices in the tree containing a given vertex



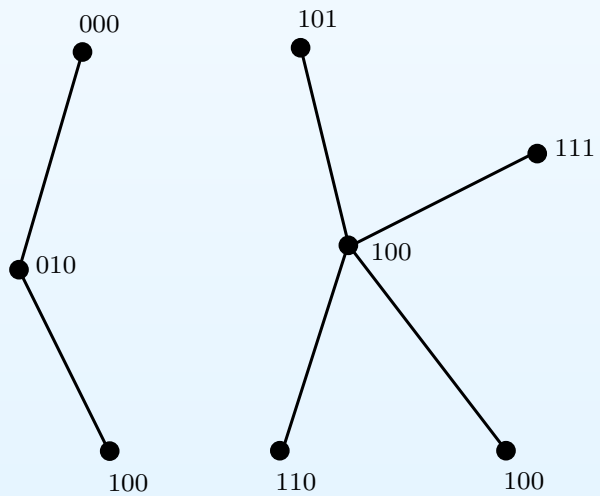
Dynamic Trees

- The data structure should be able to:
  - Insert/delete an edge
  - Update the key value of a vertex
  - Report the sum or the bitwise AND/OR/XOR of all vertices in the tree containing a given vertex



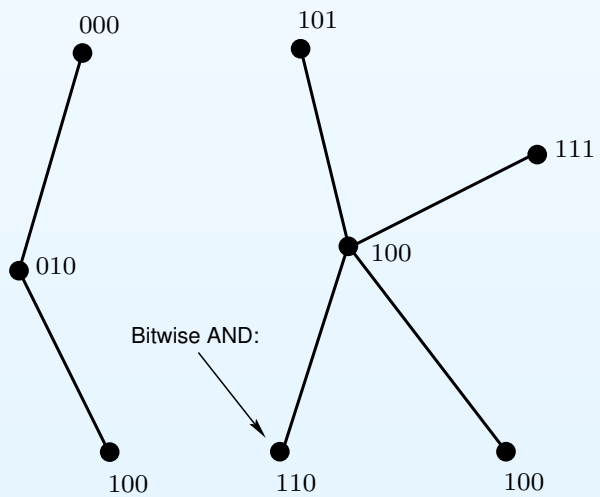
Dynamic Trees

- The data structure should be able to:
  - Insert/delete an edge
  - Update the key value of a vertex
  - Report the sum or the bitwise AND/OR/XOR of all vertices in the tree containing a given vertex



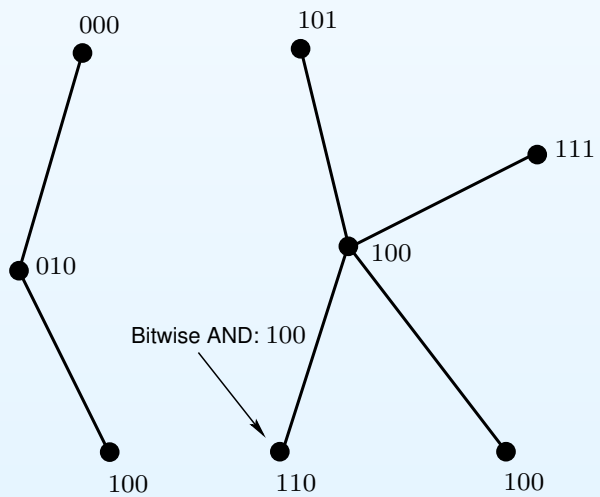
Dynamic Trees

- The data structure should be able to:
  - Insert/delete an edge
  - Update the key value of a vertex
  - Report the sum or the bitwise AND/OR/XOR of all vertices in the tree containing a given vertex



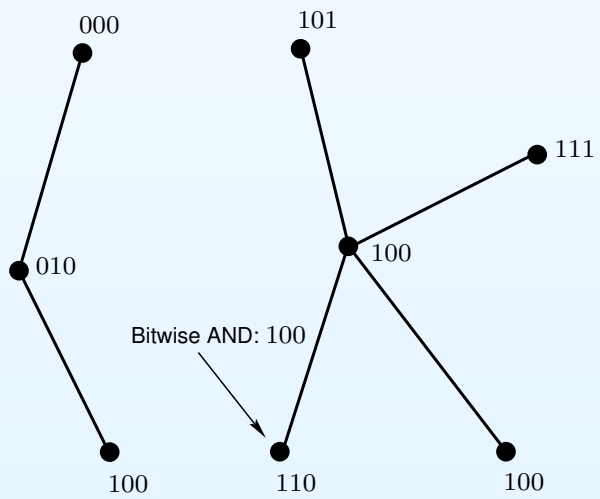
Dynamic Trees

- The data structure should be able to:
  - Insert/delete an edge
  - Update the key value of a vertex
  - Report the sum or the bitwise AND/OR/XOR of all vertices in the tree containing a given vertex



Dynamic Trees

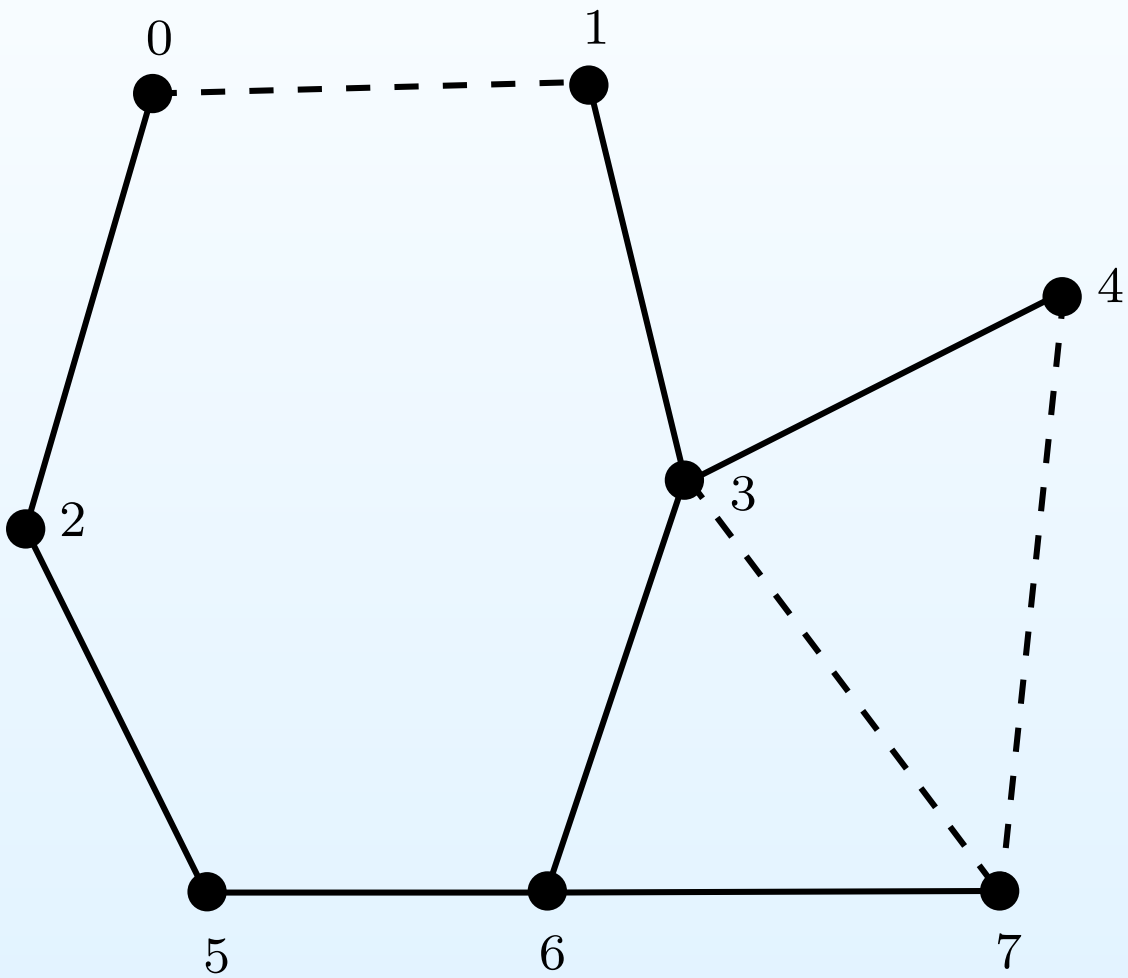
- The data structure should be able to:
  - Insert/delete an edge
  - Update the key value of a vertex
  - Report the sum or the bitwise AND/OR/XOR of all vertices in the tree containing a given vertex



- There is a dynamic tree data structure supporting each such operation in $O(\log n)$ time

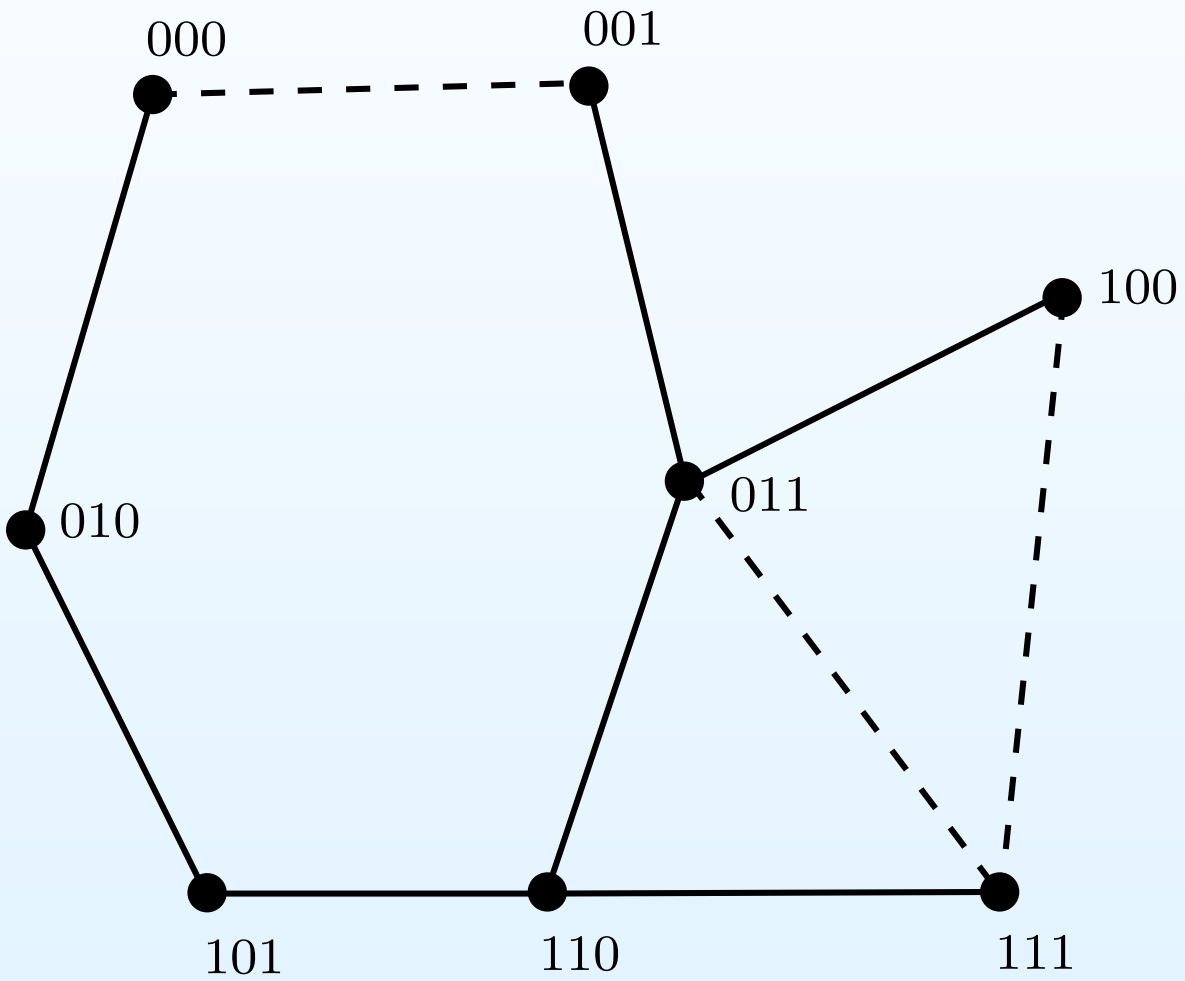
Fully-dynamic connectivity

- Efficiently maintaining a spanning forest of a graph:



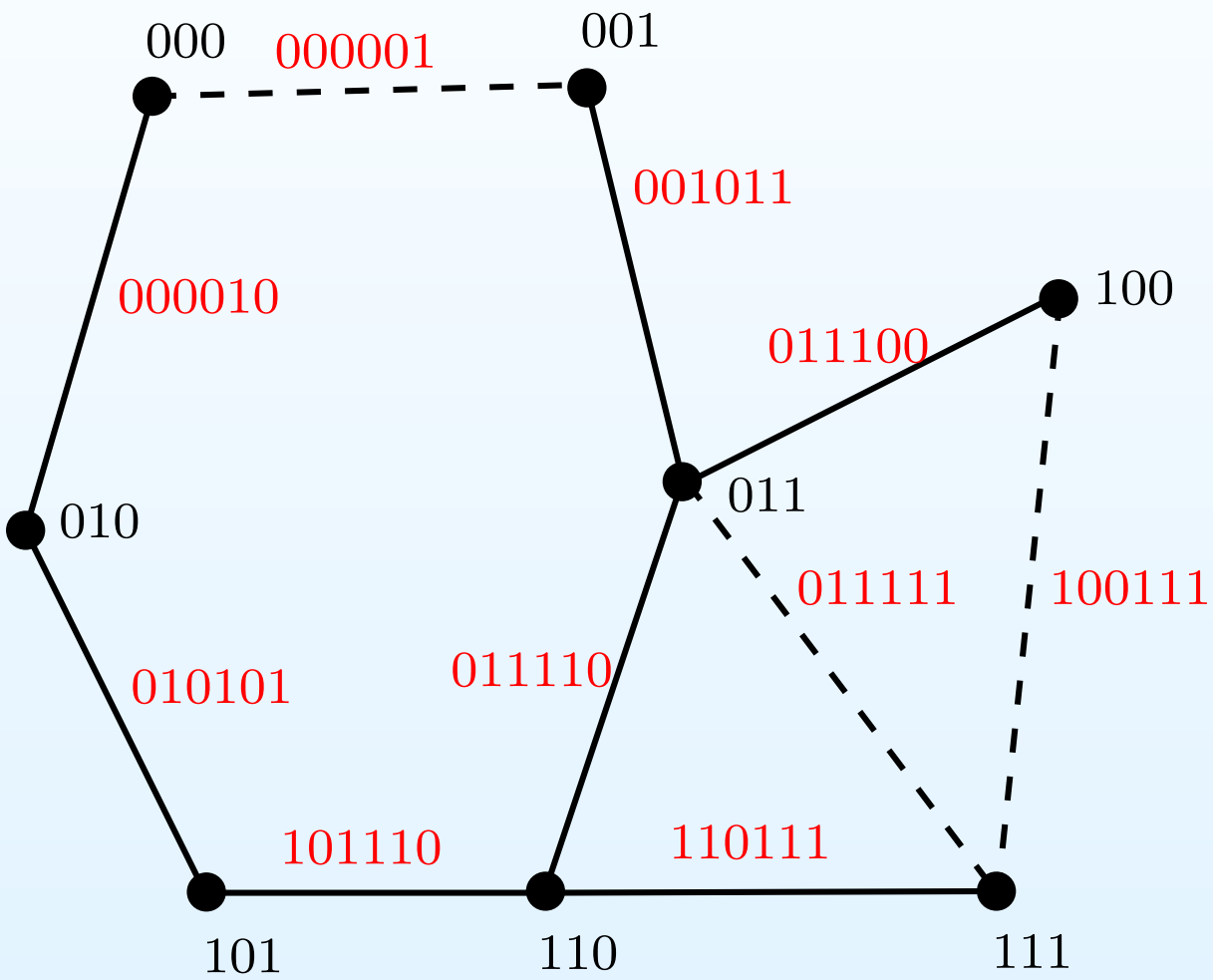
Fully-dynamic connectivity

- Efficiently maintaining a spanning forest of a graph:



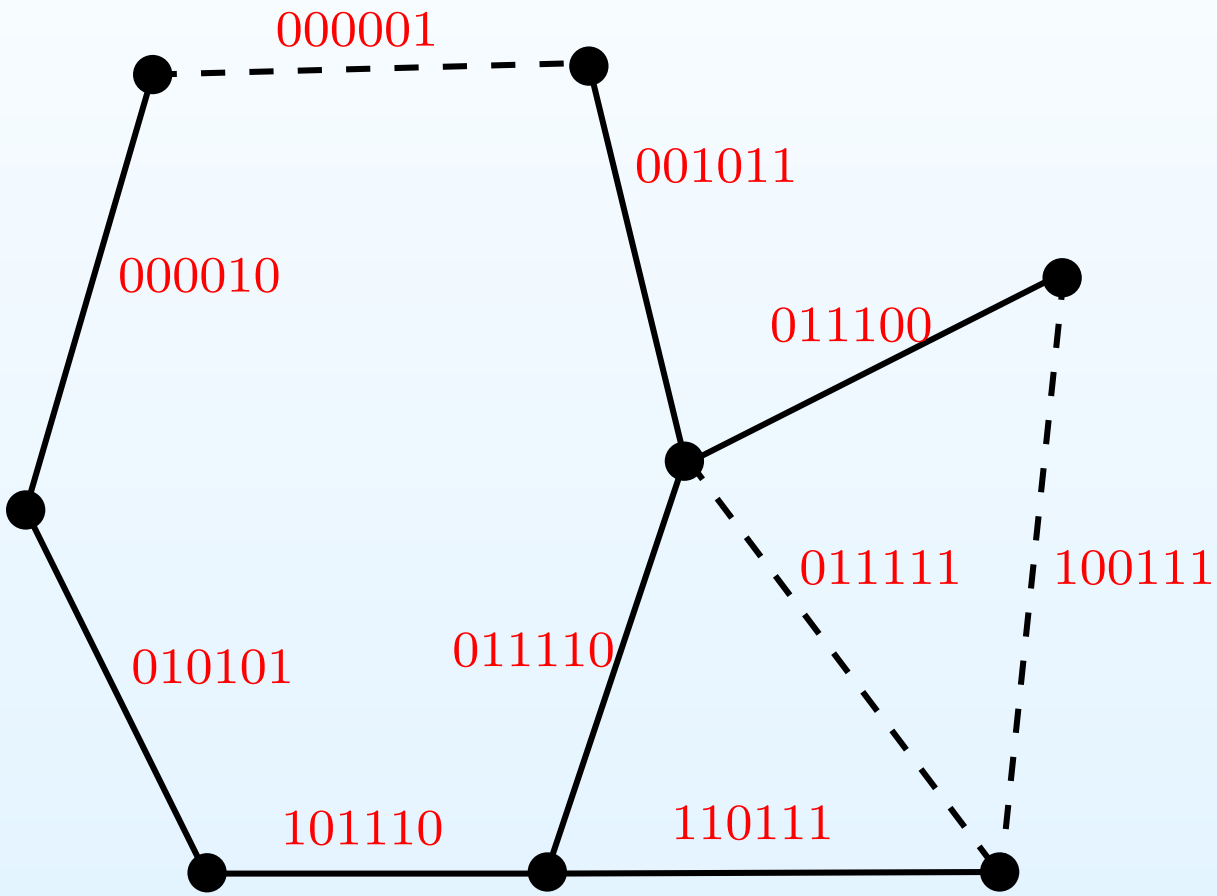
Fully-dynamic connectivity

- Efficiently maintaining a spanning forest of a graph:



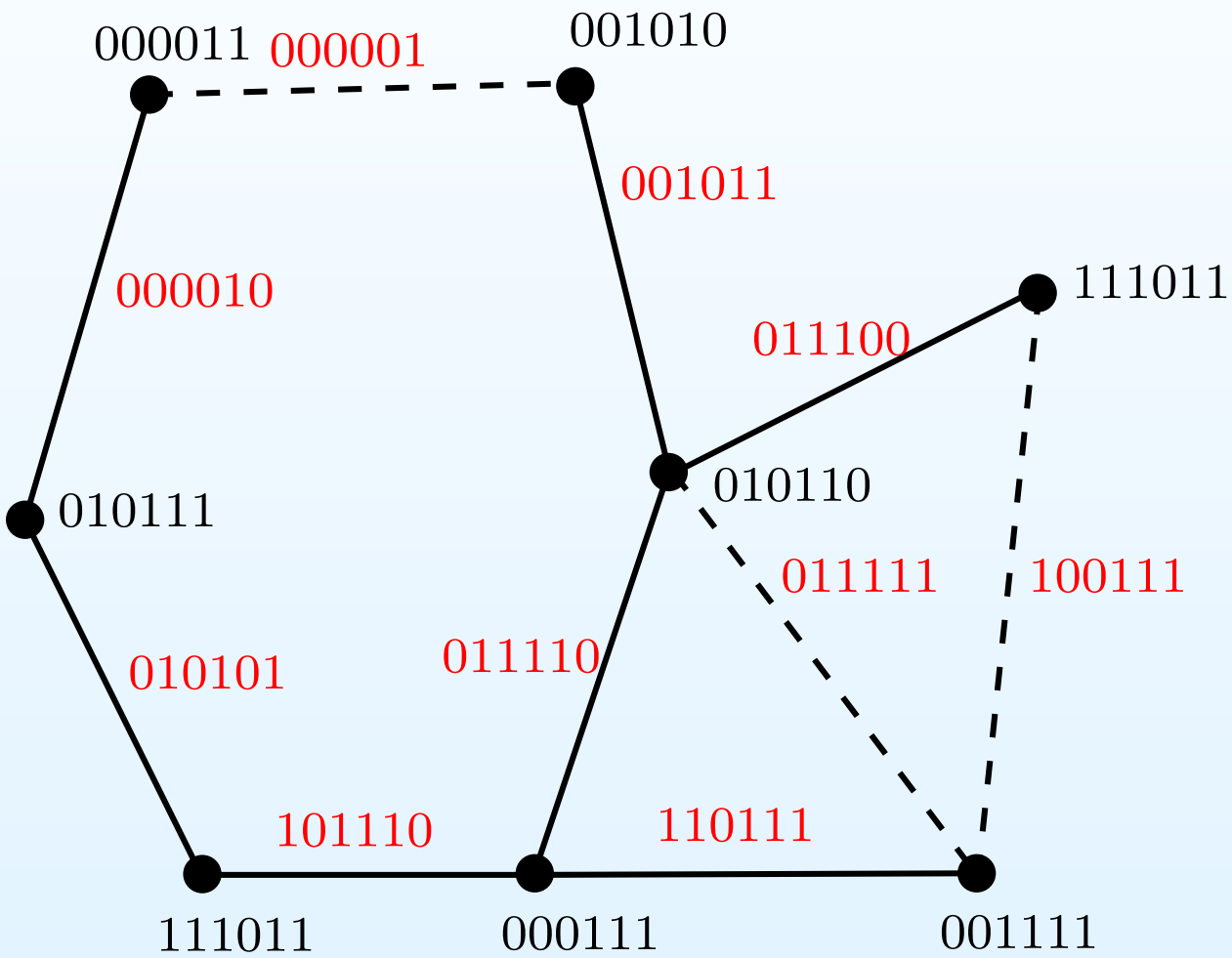
Fully-dynamic connectivity

- Efficiently maintaining a spanning forest of a graph:



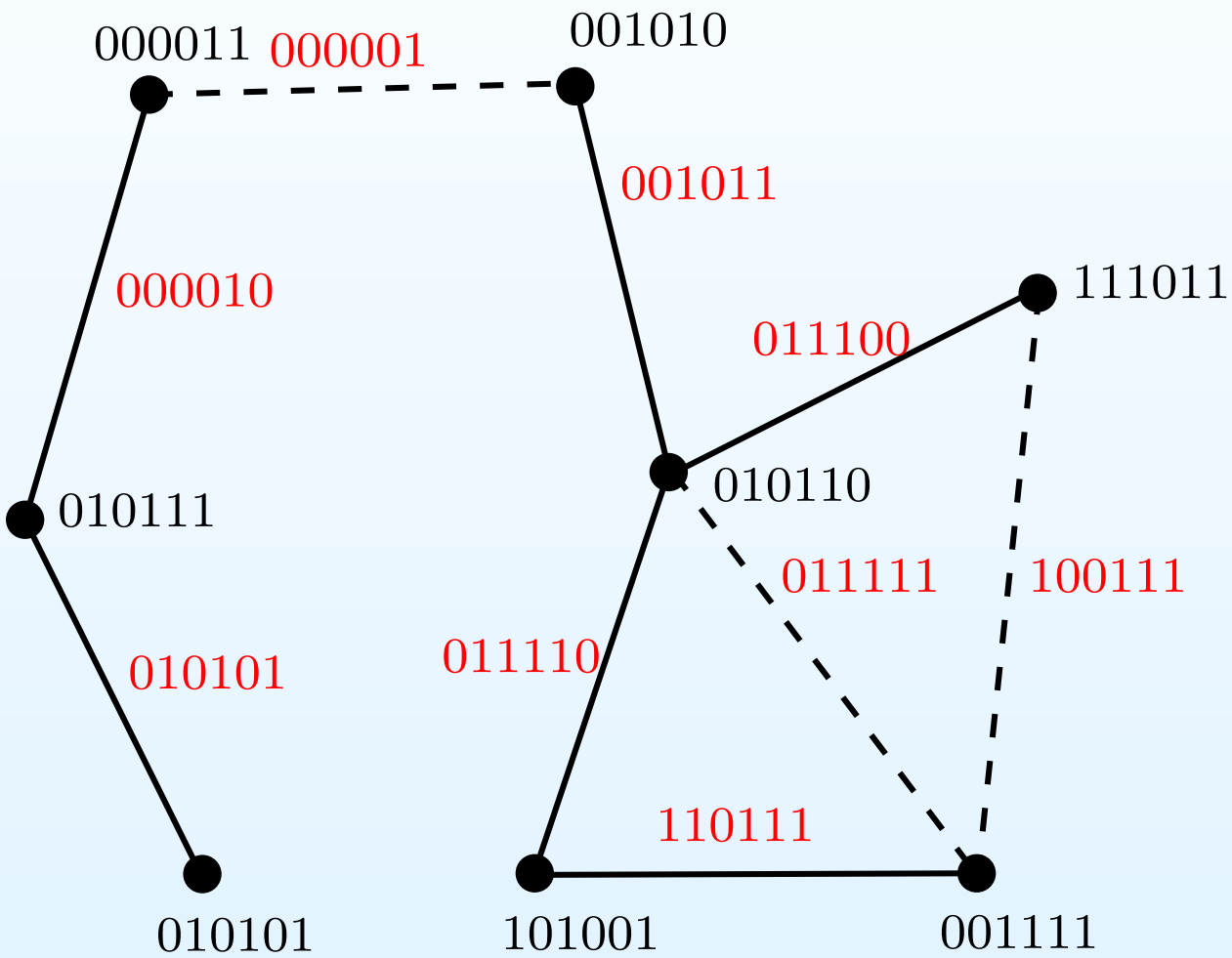
Fully-dynamic connectivity

- Efficiently maintaining a spanning forest of a graph:



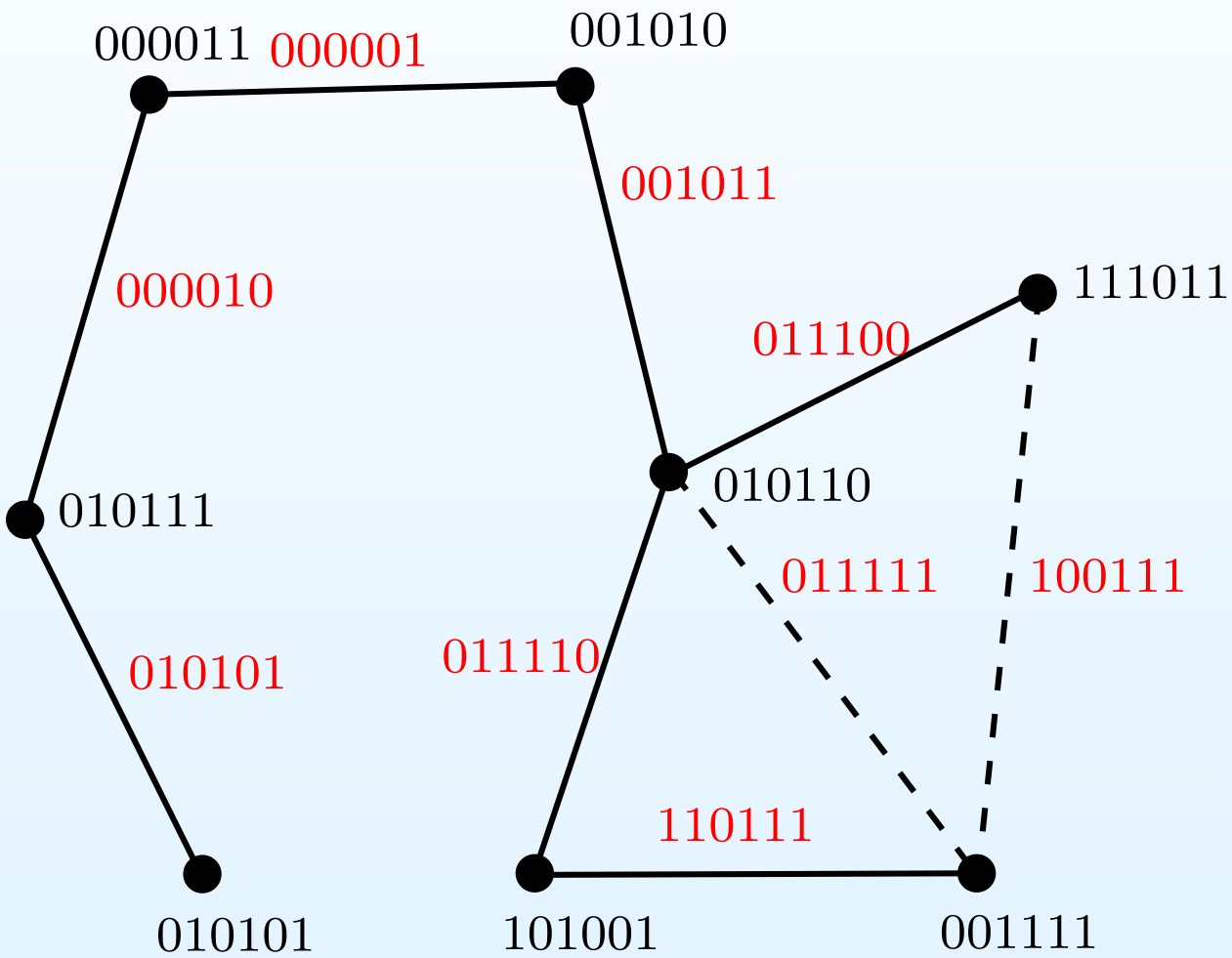
Fully-dynamic connectivity

- Efficiently maintaining a spanning forest of a graph:



Fully-dynamic connectivity

- Efficiently maintaining a spanning forest of a graph:

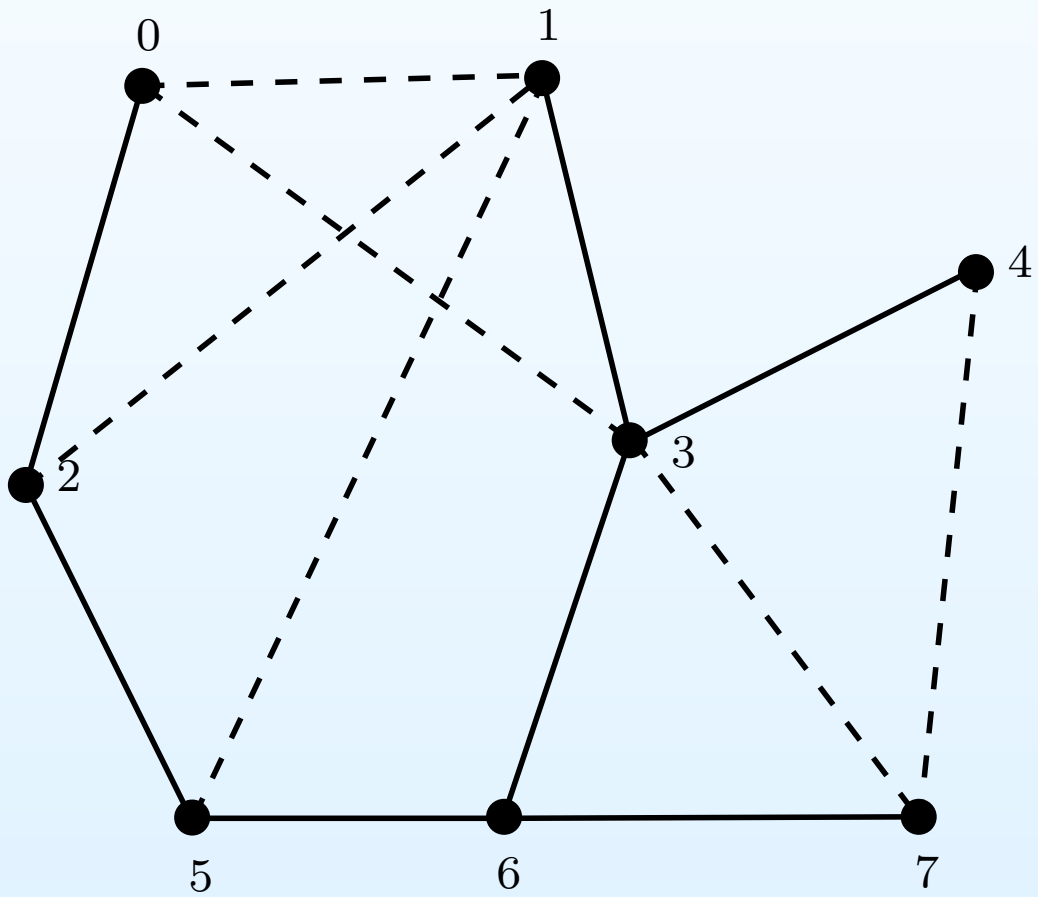


Fully-dynamic connectivity

- The previous only works if there is at most one edge that can reconnect the tree.

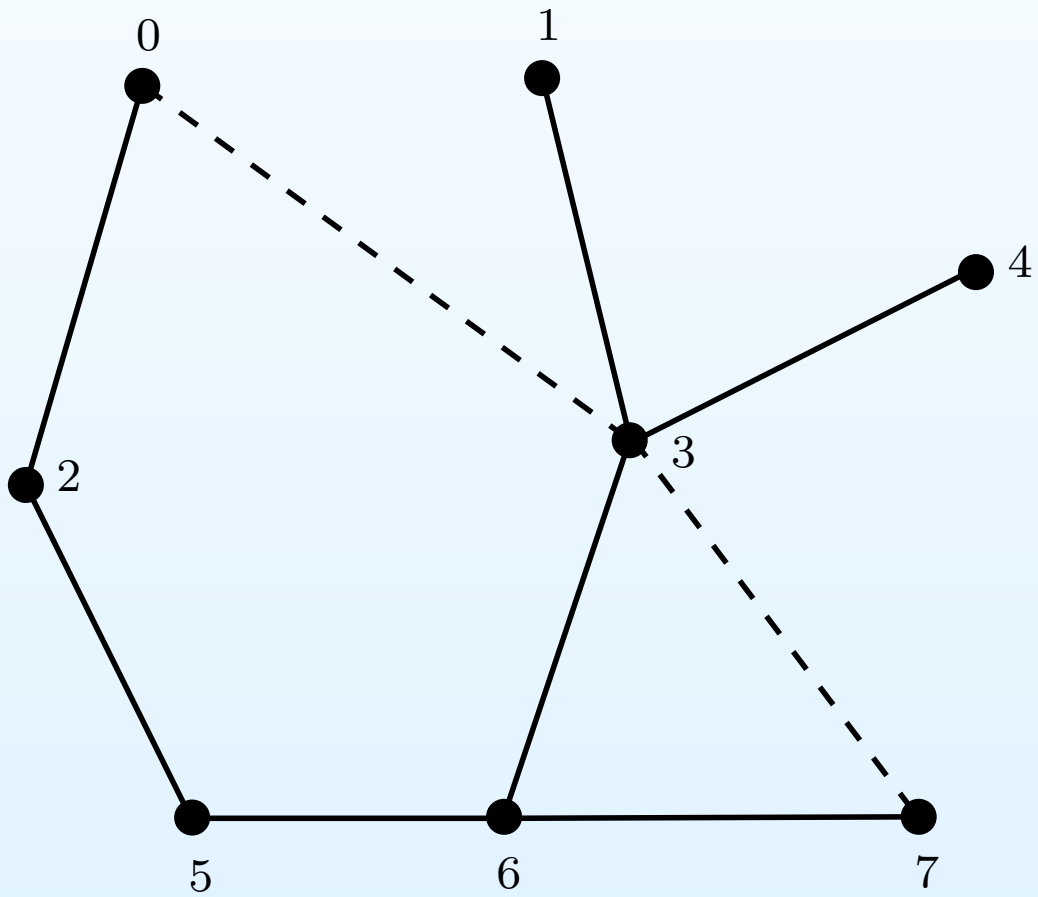
Fully-dynamic connectivity

- The previous only works if there is at most one edge that can reconnect the tree.
- Handling multiple reconnecting edges with random sampling:



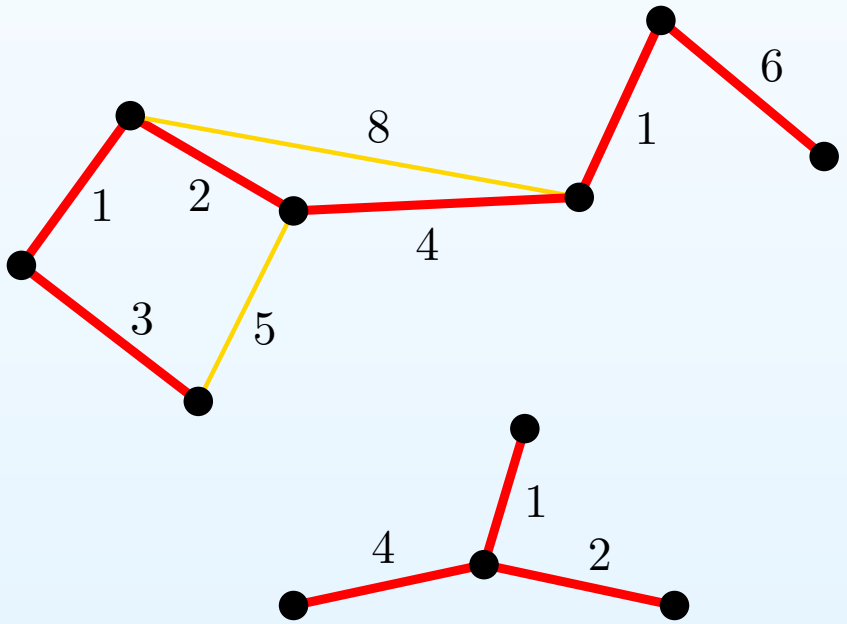
Fully-dynamic connectivity

- The previous only works if there is at most one edge that can reconnect the tree.
- Handling multiple reconnecting edges with random sampling:



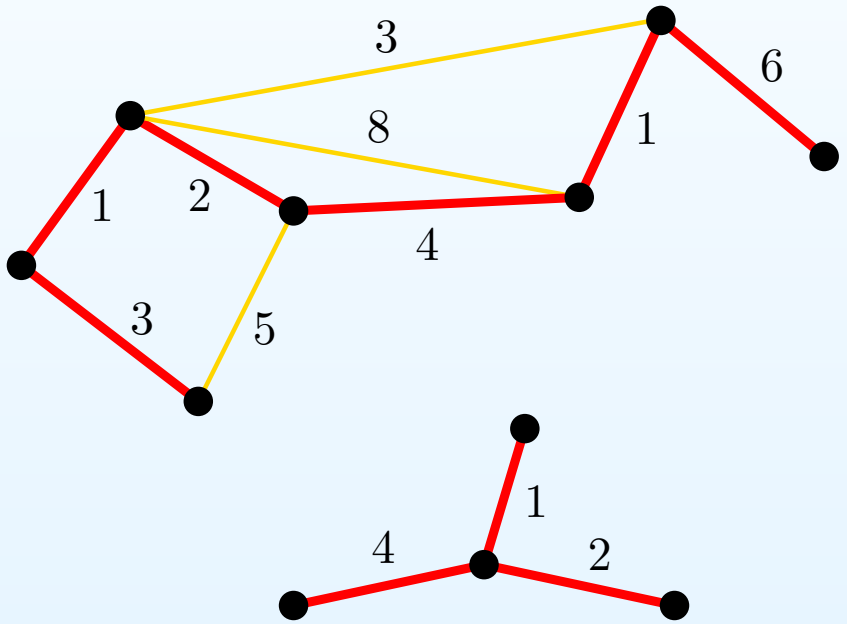
Dynamic minimum spanning forest

- Suppose that instead of maintaining a spanning forest, we want to maintain a *minimum* spanning forest:



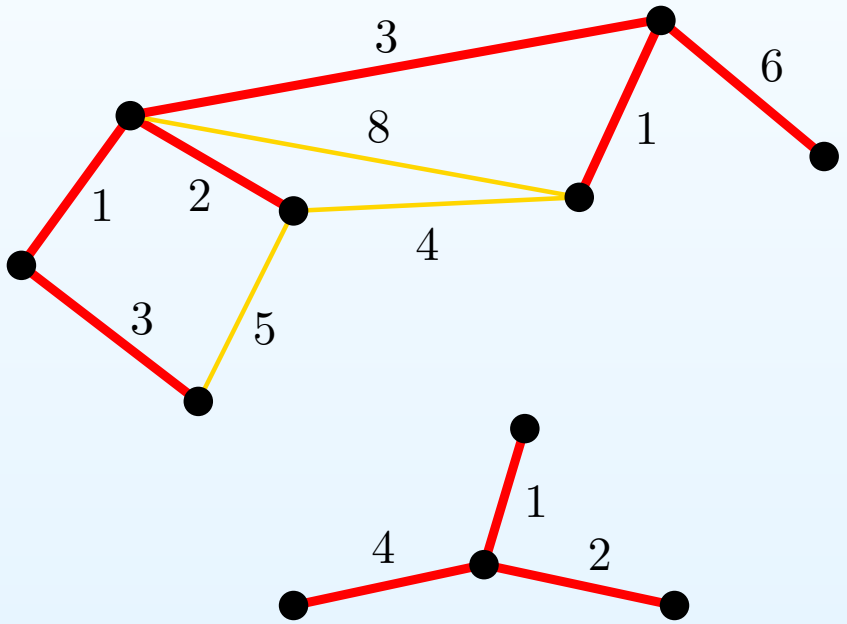
Dynamic minimum spanning forest

- Suppose that instead of maintaining a spanning forest, we want to maintain a *minimum* spanning forest:



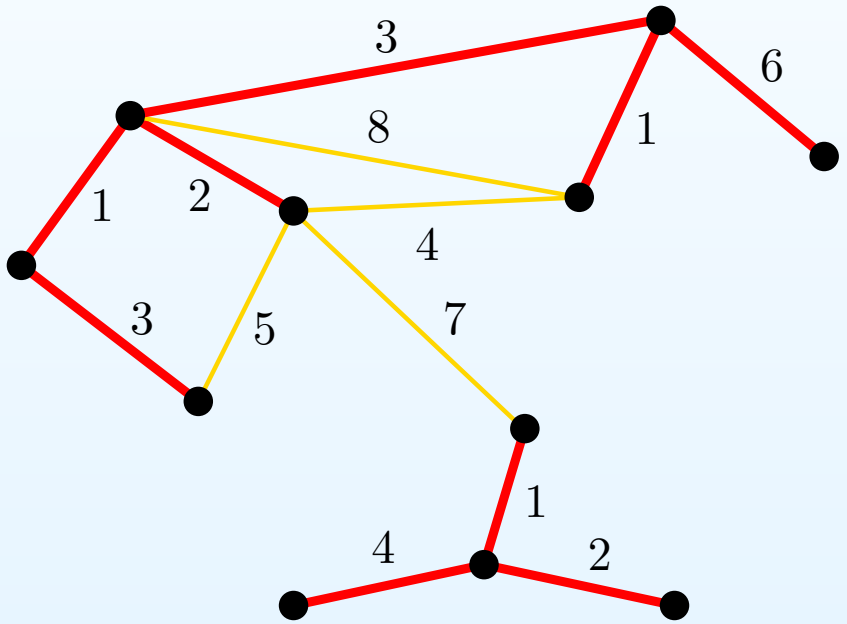
Dynamic minimum spanning forest

- Suppose that instead of maintaining a spanning forest, we want to maintain a *minimum* spanning forest:



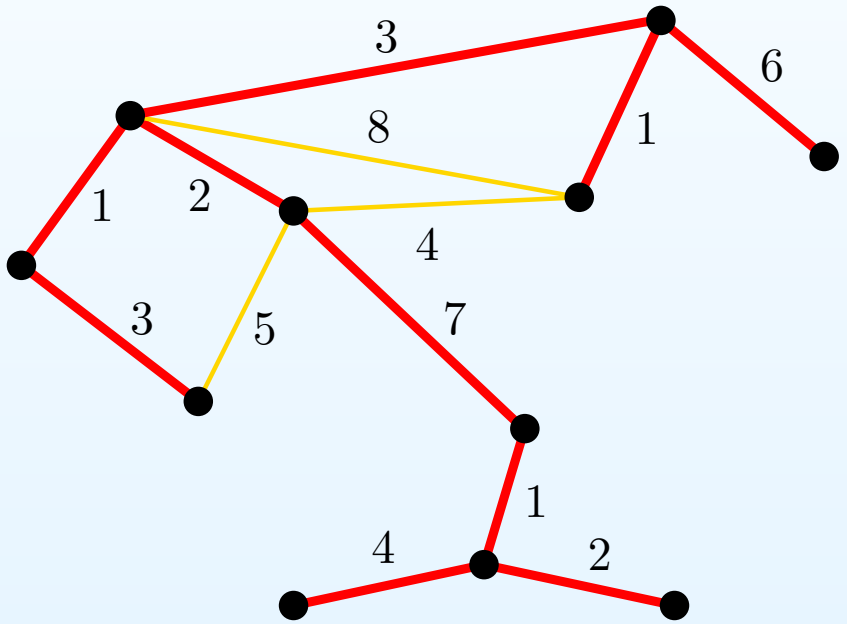
Dynamic minimum spanning forest

- Suppose that instead of maintaining a spanning forest, we want to maintain a *minimum* spanning forest:



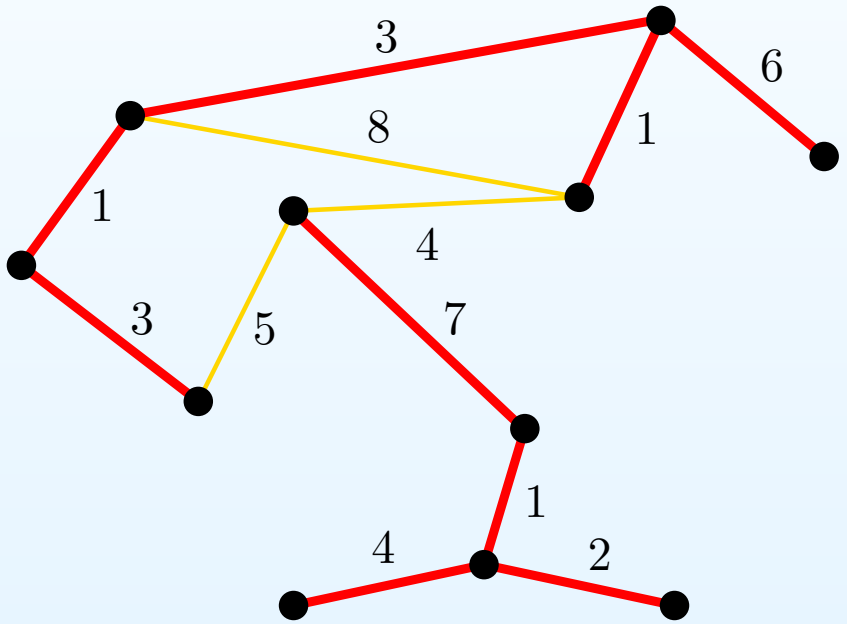
Dynamic minimum spanning forest

- Suppose that instead of maintaining a spanning forest, we want to maintain a *minimum* spanning forest:



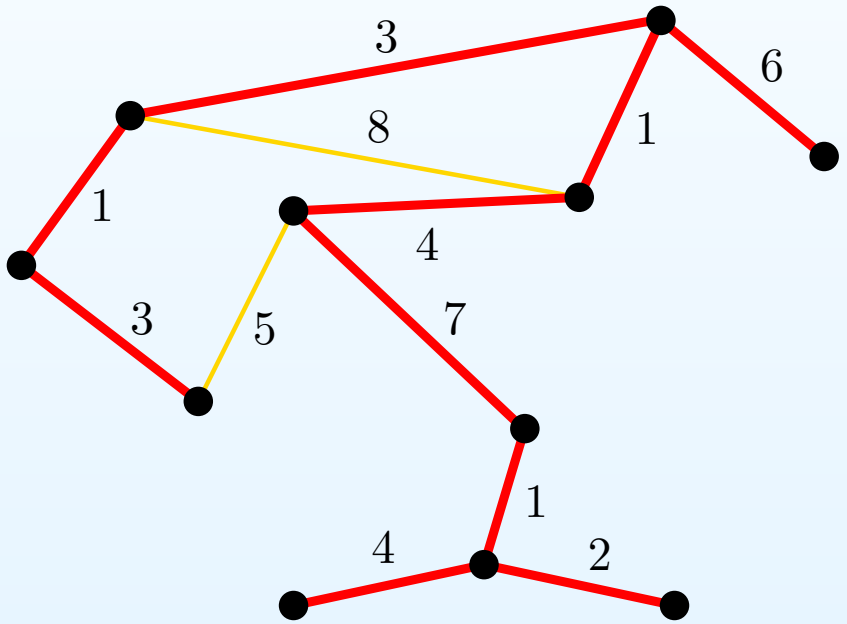
Dynamic minimum spanning forest

- Suppose that instead of maintaining a spanning forest, we want to maintain a *minimum* spanning forest:



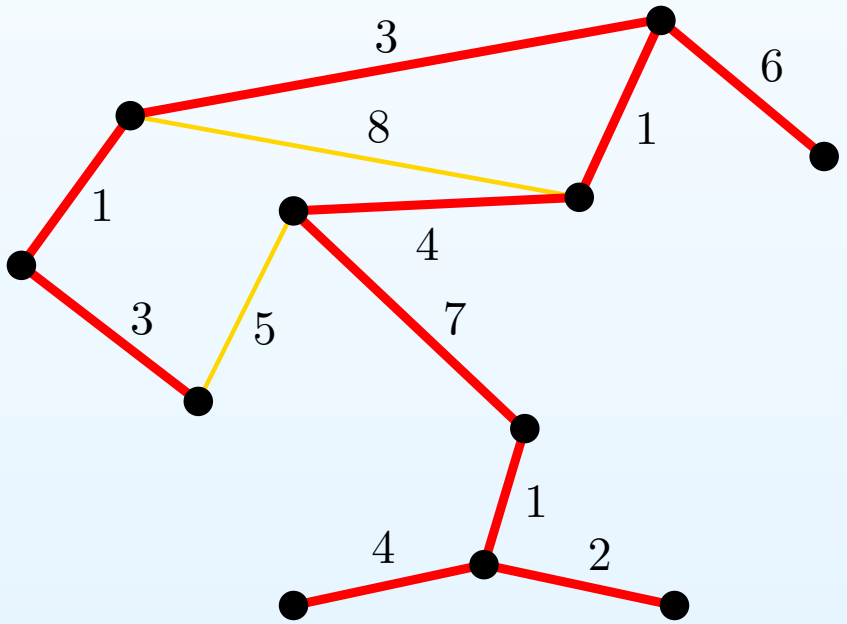
Dynamic minimum spanning forest

- Suppose that instead of maintaining a spanning forest, we want to maintain a *minimum* spanning forest:



Dynamic minimum spanning forest

- Suppose that instead of maintaining a spanning forest, we want to maintain a *minimum* spanning forest:



- Does the XOR trick work here?

Types of randomized algorithms

- For several graph problems, state-of-the-art algorithms are randomized

Types of randomized algorithms

- For several graph problems, state-of-the-art algorithms are randomized
- Some of these algorithms are *Las Vegas*, some are *Monte Carlo*.

Types of randomized algorithms

- For several graph problems, state-of-the-art algorithms are randomized
- Some of these algorithms are *Las Vegas*, some are *Monte Carlo*.
- Las Vegas algorithm: produces the correct solution but time/space random

Types of randomized algorithms

- For several graph problems, state-of-the-art algorithms are randomized
- Some of these algorithms are *Las Vegas*, some are *Monte Carlo*.
- Las Vegas algorithm: produces the correct solution but time/space random
- Monte Carlo algorithm: may give incorrect solutions but time/space typically not random

Issues with randomized dynamic graph algorithms

- A randomized dynamic graph algorithm either needs to:
  - hide its random bits from the user or
  - assume that the user is not adaptive

Issues with randomized algorithms

- Concrete example: Randomized approximate shortest path algorithm supporting:

Issues with randomized algorithms

- Concrete example: Randomized approximate shortest path algorithm supporting:
  - Updates: insertion/deletion of a single edge

Issues with randomized algorithms

- Concrete example: Randomized approximate shortest path algorithm supporting:
  - Updates: insertion/deletion of a single edge
  - Queries: given query vertices u and v, output a value $\tilde{d}_G(u, v)$ such that

$$d_G(u, v) \leq \tilde{d}_G(u, v) \leq \delta \cdot d_G(u, v)$$

Issues with randomized algorithms

- Concrete example: Randomized approximate shortest path algorithm supporting:
  - Updates: insertion/deletion of a single edge
  - Queries: given query vertices u and v, output a value $\tilde{d}_G(u, v)$ such that

$$d_G(u, v) \leq \tilde{d}_G(u, v) \leq \delta \cdot d_G(u, v)$$

- Suppose a user only has access to the values $\tilde{d}_G(u, v)$ obtained from queries

Issues with randomized algorithms

- Concrete example: Randomized approximate shortest path algorithm supporting:
  - Updates: insertion/deletion of a single edge
  - Queries: given query vertices u and v, output a value $\tilde{d}_G(u, v)$ such that

$$d_G(u, v) \leq \tilde{d}_G(u, v) \leq \delta \cdot d_G(u, v)$$

- Suppose a user only has access to the values $\tilde{d}_G(u, v)$ obtained from queries
- Even this may potentially reveal too much information about the random bits of the algorithm

Oblivious versus adaptive adversary

- Oblivious adversary: entire sequence of updates/queries fixed in advance

Oblivious versus adaptive adversary

- Oblivious adversary: entire sequence of updates/queries fixed in advance
- Adaptive adversary: the opposite of oblivious adversary
  - is allowed to choose subsequent updates/queries based on the information revealed by the algorithm so far

Oblivious versus adaptive adversary

- Oblivious adversary: entire sequence of updates/queries fixed in advance
- Adaptive adversary: the opposite of oblivious adversary
  - is allowed to choose subsequent updates/queries based on the information revealed by the algorithm so far
- Assuming an oblivious adversary:
  - deals with the issue on the previous slides

Oblivious versus adaptive adversary

- Oblivious adversary: entire sequence of updates/queries fixed in advance
- Adaptive adversary: the opposite of oblivious adversary
  - is allowed to choose subsequent updates/queries based on the information revealed by the algorithm so far
- Assuming an oblivious adversary:
  - deals with the issue on the previous slides
  - but significantly limits the range of applications for the algorithm

Oblivious versus adaptive adversary

- Oblivious adversary: entire sequence of updates/queries fixed in advance
- Adaptive adversary: the opposite of oblivious adversary
  - is allowed to choose subsequent updates/queries based on the information revealed by the algorithm so far
- Assuming an oblivious adversary:
  - deals with the issue on the previous slides
  - but significantly limits the range of applications for the algorithm
- Two ways of allowing an adversary to be adaptive:

Oblivious versus adaptive adversary

- Oblivious adversary: entire sequence of updates/queries fixed in advance
- Adaptive adversary: the opposite of oblivious adversary
  - is allowed to choose subsequent updates/queries based on the information revealed by the algorithm so far
- Assuming an oblivious adversary:
  - deals with the issue on the previous slides
  - but significantly limits the range of applications for the algorithm
- Two ways of allowing an adversary to be adaptive:
  - develop a deterministic algorithm

Oblivious versus adaptive adversary

- Oblivious adversary: entire sequence of updates/queries fixed in advance
- Adaptive adversary: the opposite of oblivious adversary
  - is allowed to choose subsequent updates/queries based on the information revealed by the algorithm so far
- Assuming an oblivious adversary:
  - deals with the issue on the previous slides
  - but significantly limits the range of applications for the algorithm
- Two ways of allowing an adversary to be adaptive:
  - develop a deterministic algorithm
  - develop a randomized algorithm and prove that it does not reveal (enough) information about its random bits