Distributed minimum spanning tree problem

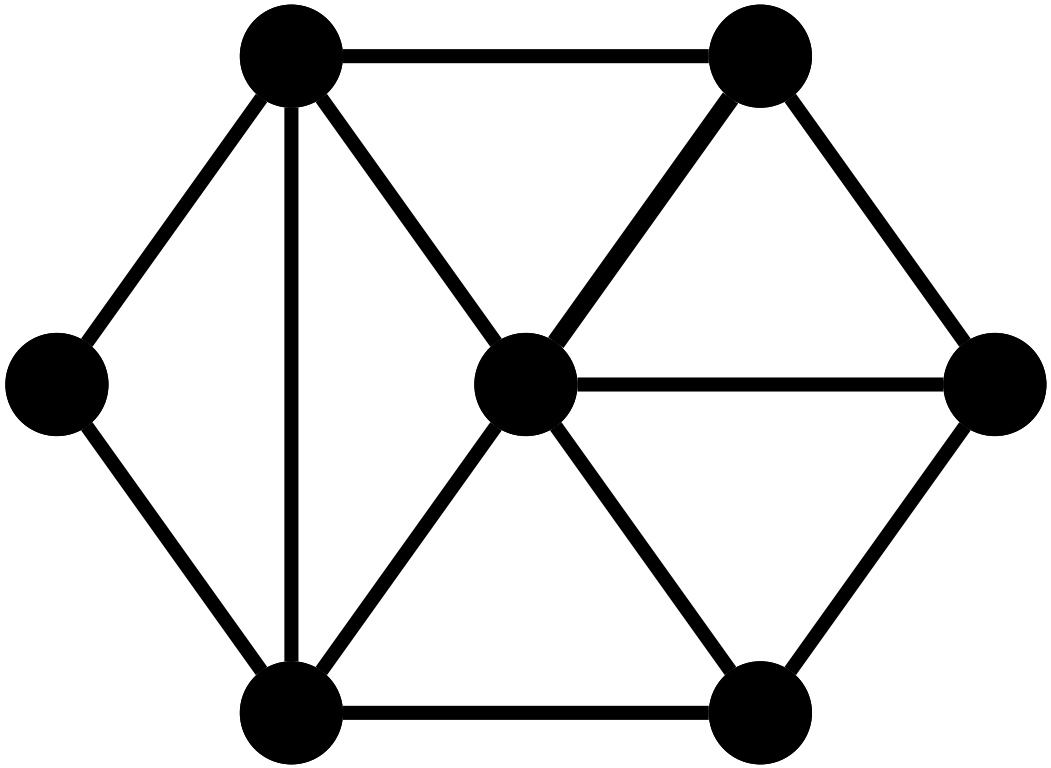
Kustaa Kangas

University of Helsinki, Finland

November 8, 2012

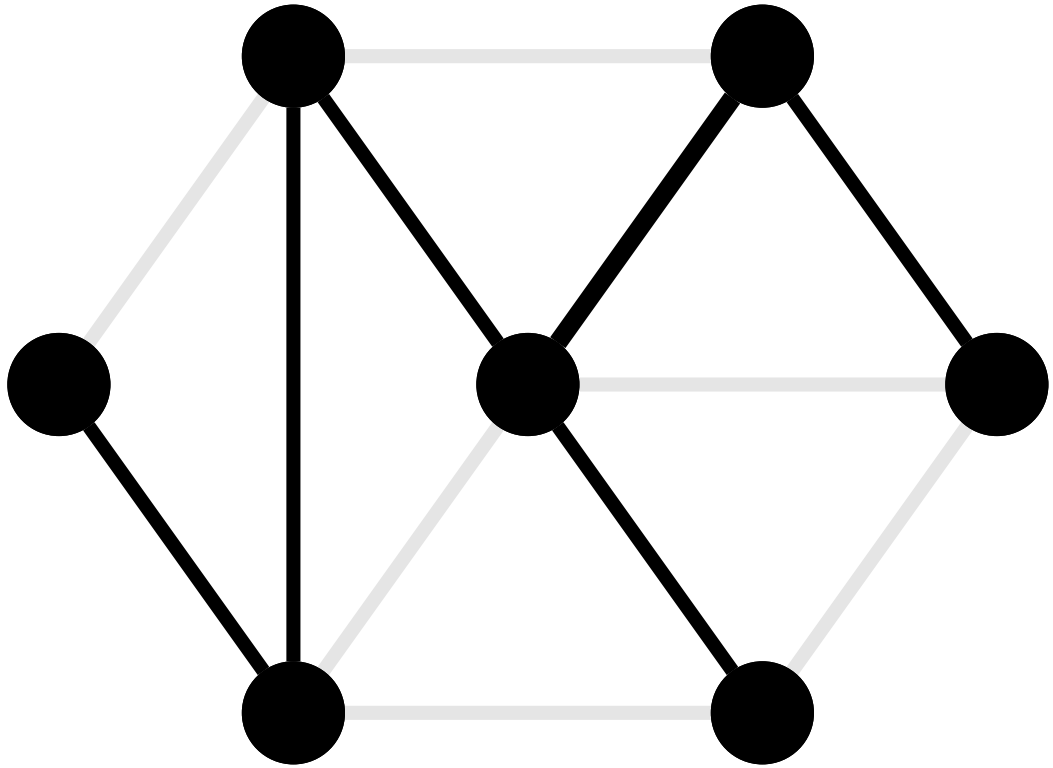
Minimum spanning tree

A spanning tree of an undirected graph is a subtree containing all vertices.



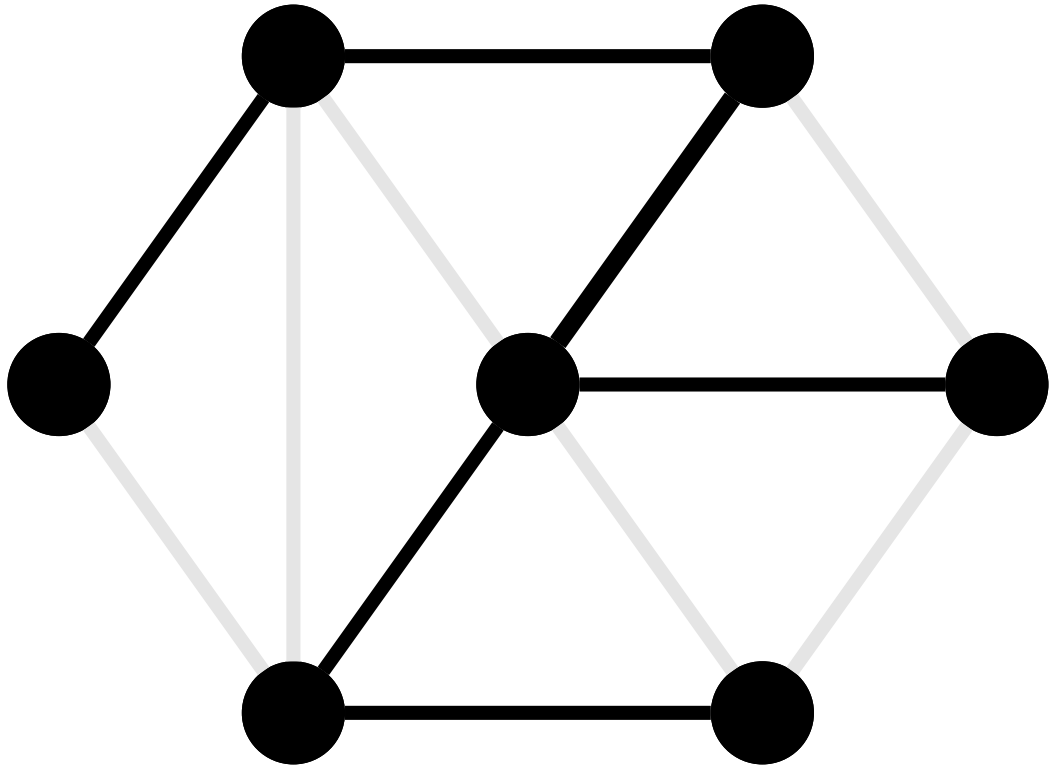
Minimum spanning tree

A spanning tree of an undirected graph is a subtree containing all vertices.



Minimum spanning tree

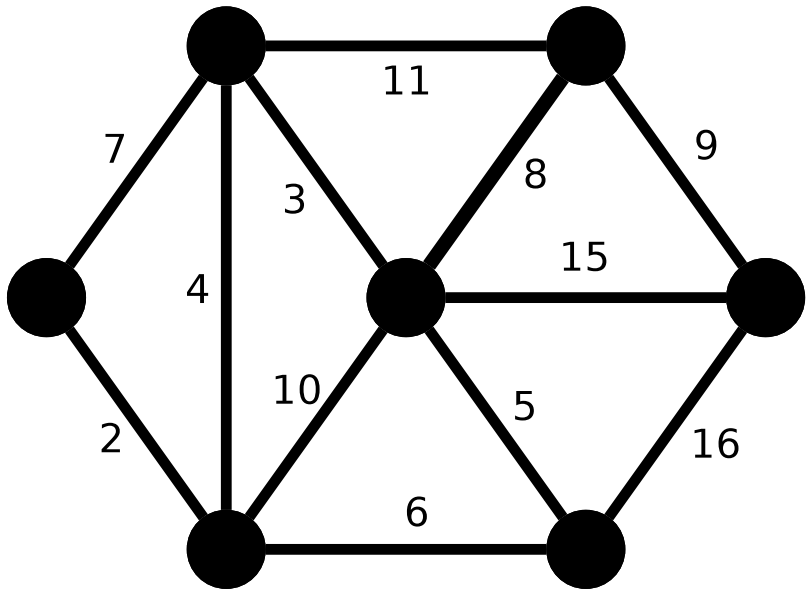
A spanning tree of an undirected graph is a subtree containing all vertices.



Minimum spanning tree

For a weighted graph, the weight of a spanning tree is the sum of the weights of its edges.

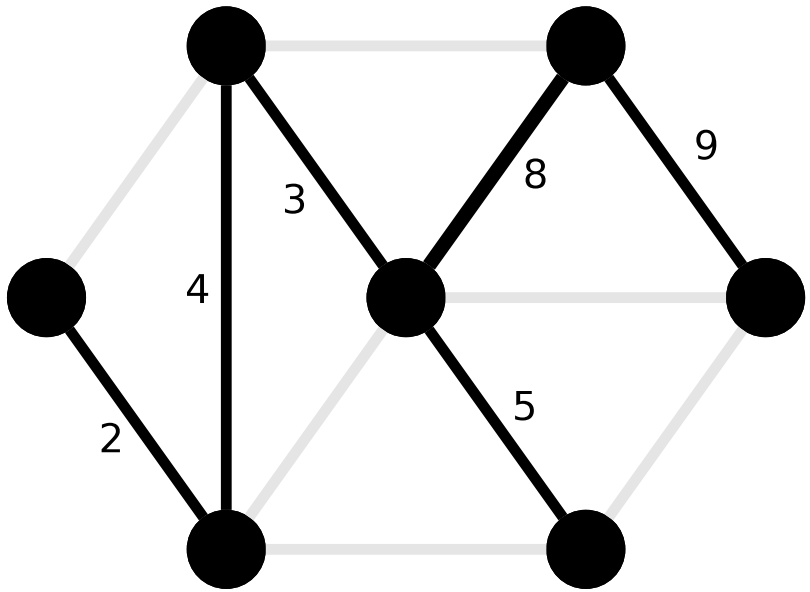
- A tree with a minimum weight is called a *minimum spanning tree*.
- For simplicity assume that all weights are disctinct so the MST is unique (proof in the report).



Minimum spanning tree

For a weighted graph, the weight of a spanning tree is the sum of the weights of its edges.

- A tree with a minimum weight is called a *minimum spanning tree*.
- For simplicity assume that all weights are disctinct so the MST is unique (proof in the report).



Finding minimum spanning tree

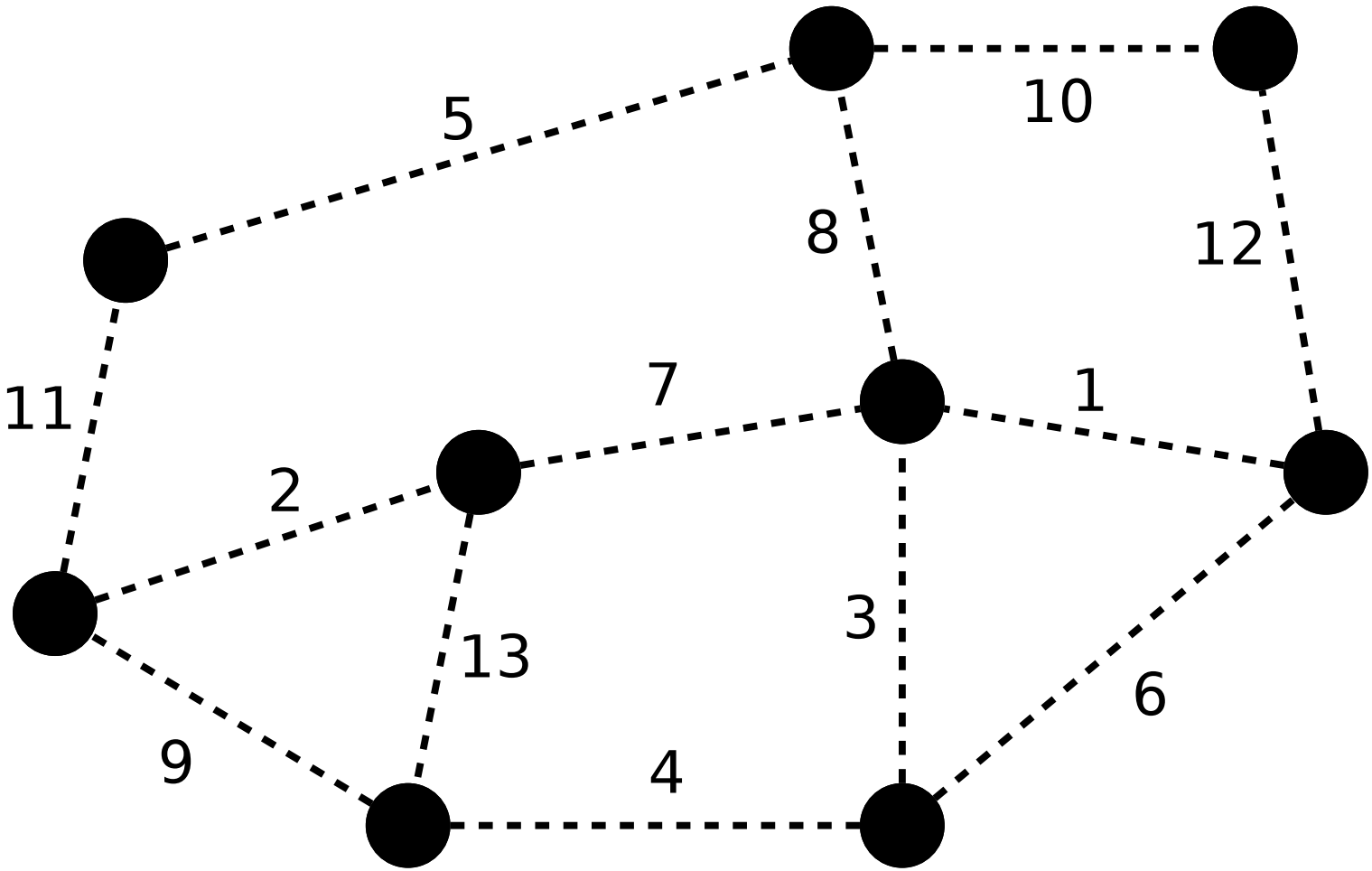
How to find the MST of a given graph?

Fast and simple centralized algorithms:

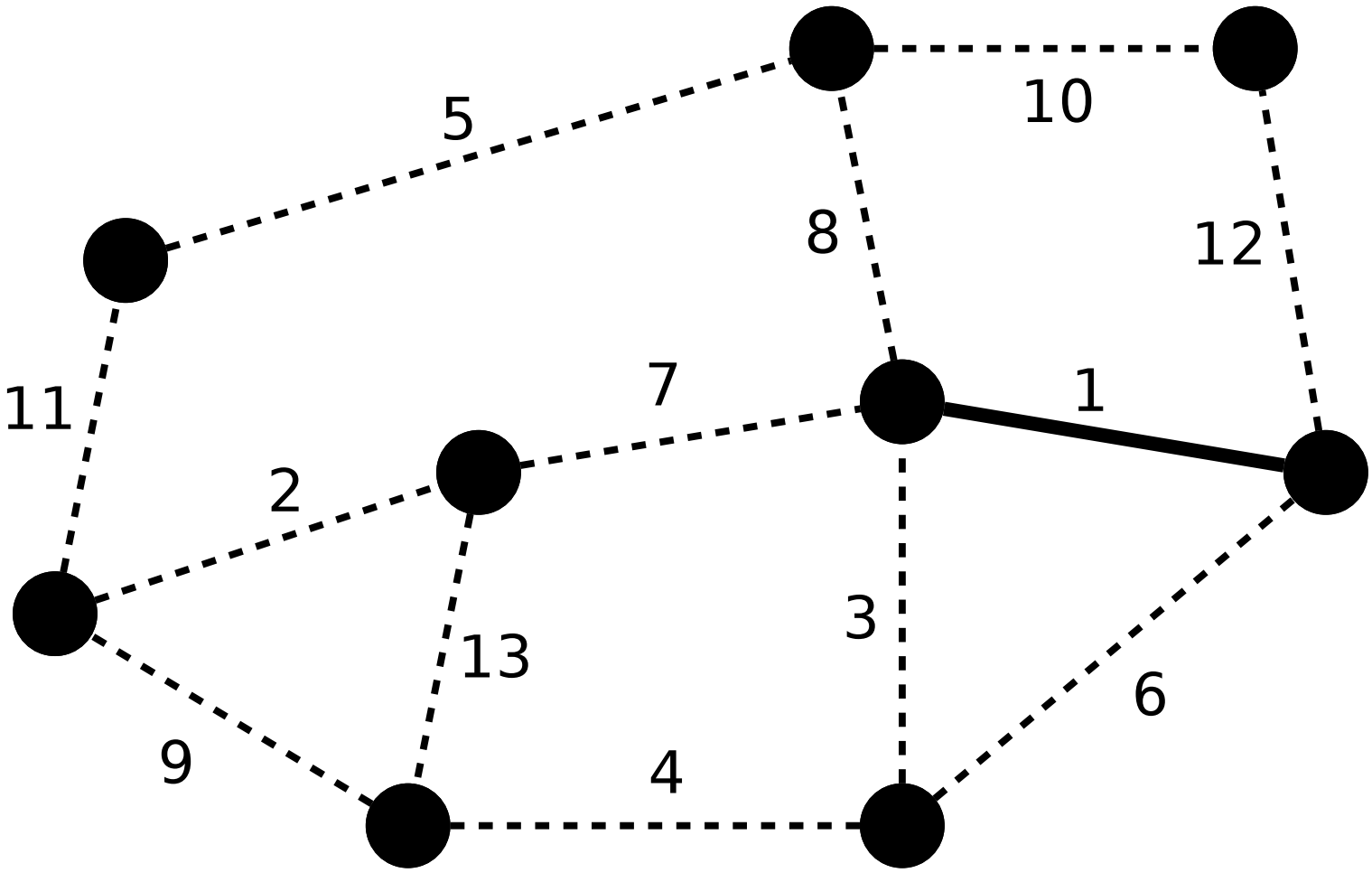
- Kruskal: add edges greedily, avoid cycles.
- Prim: expand a single fragment greedily.

Run in $\mathcal{O}(|E| \log |N|)$.

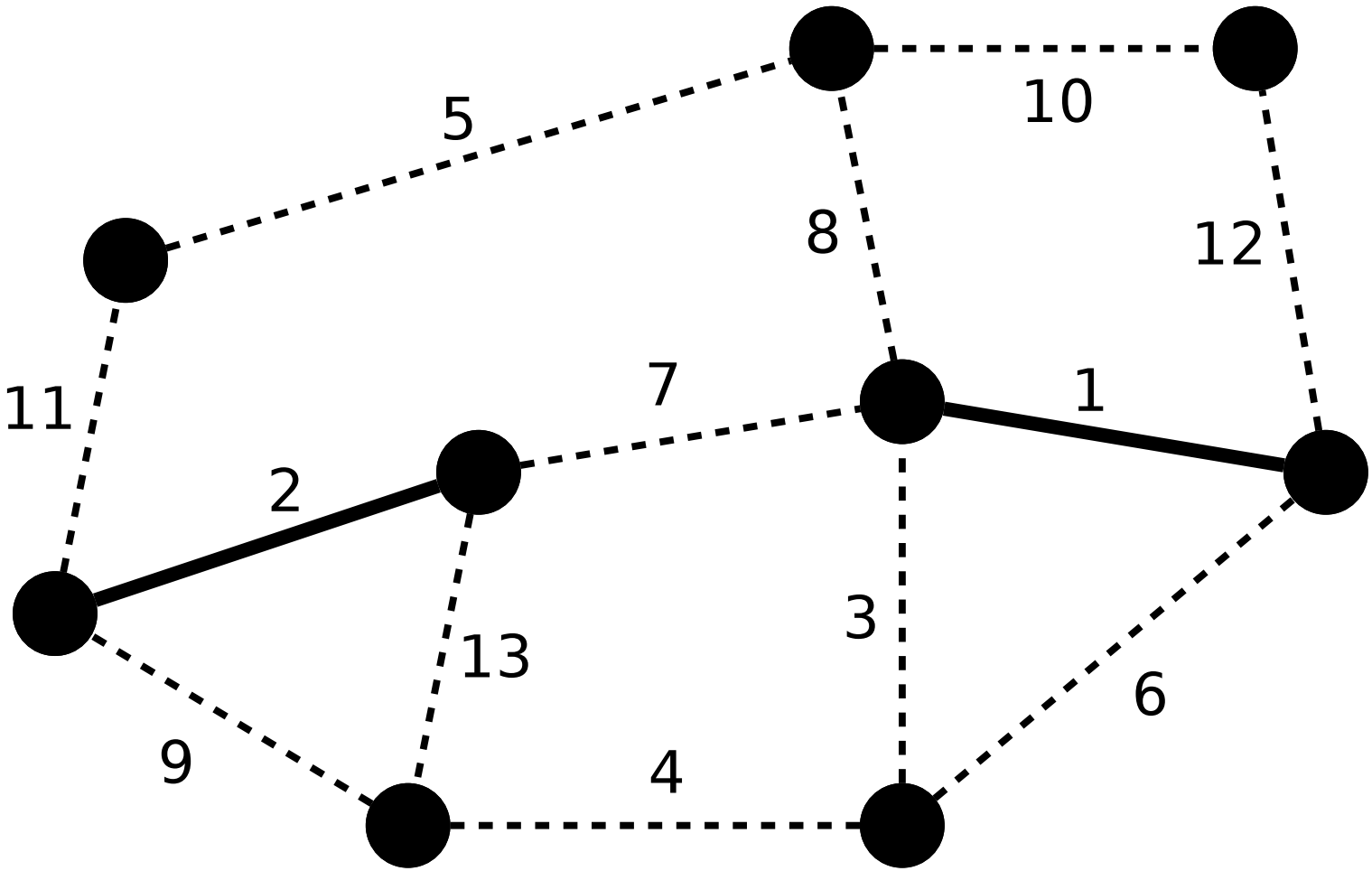
Kruskal's algorithm



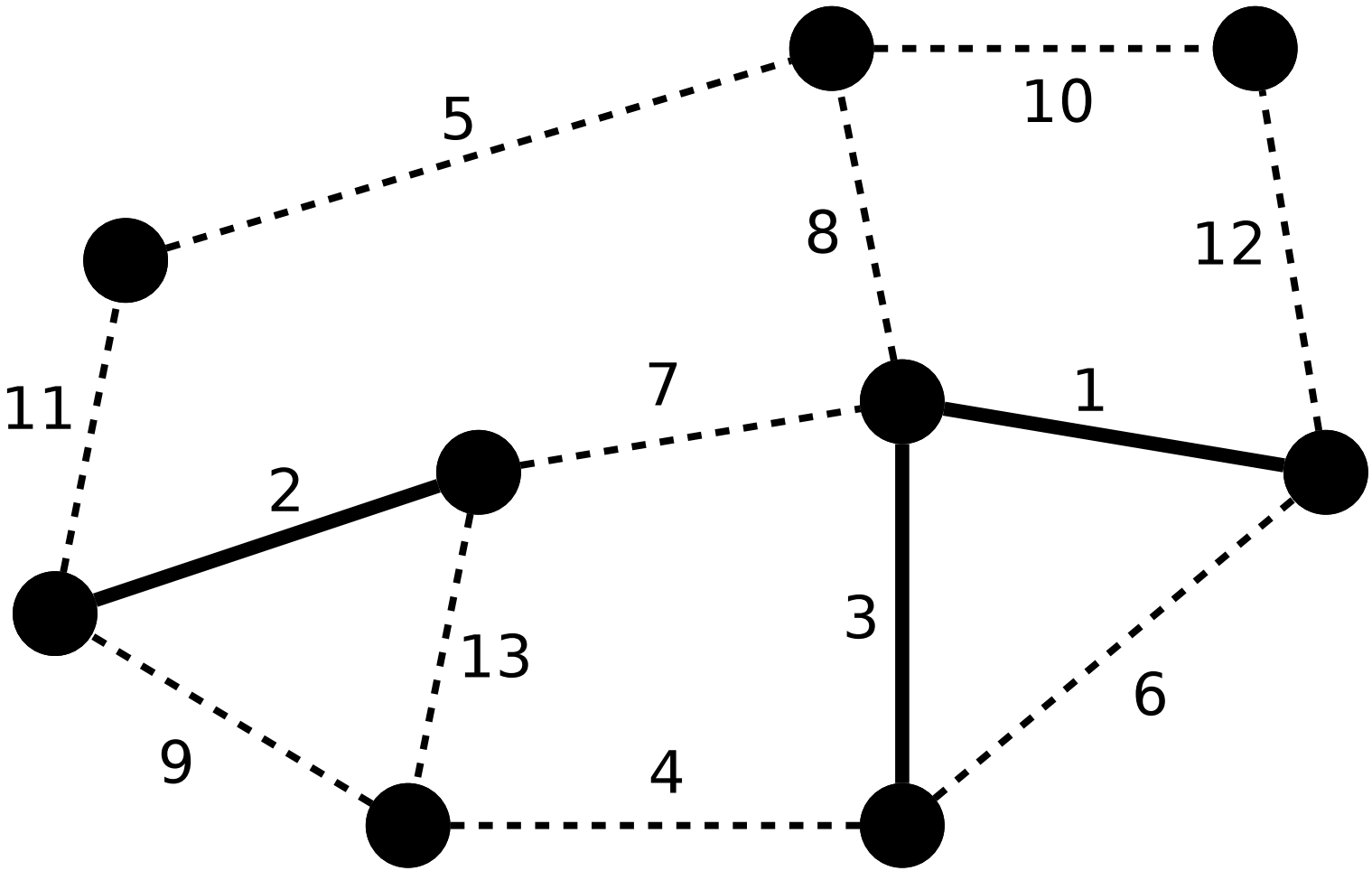
Kruskal's algorithm



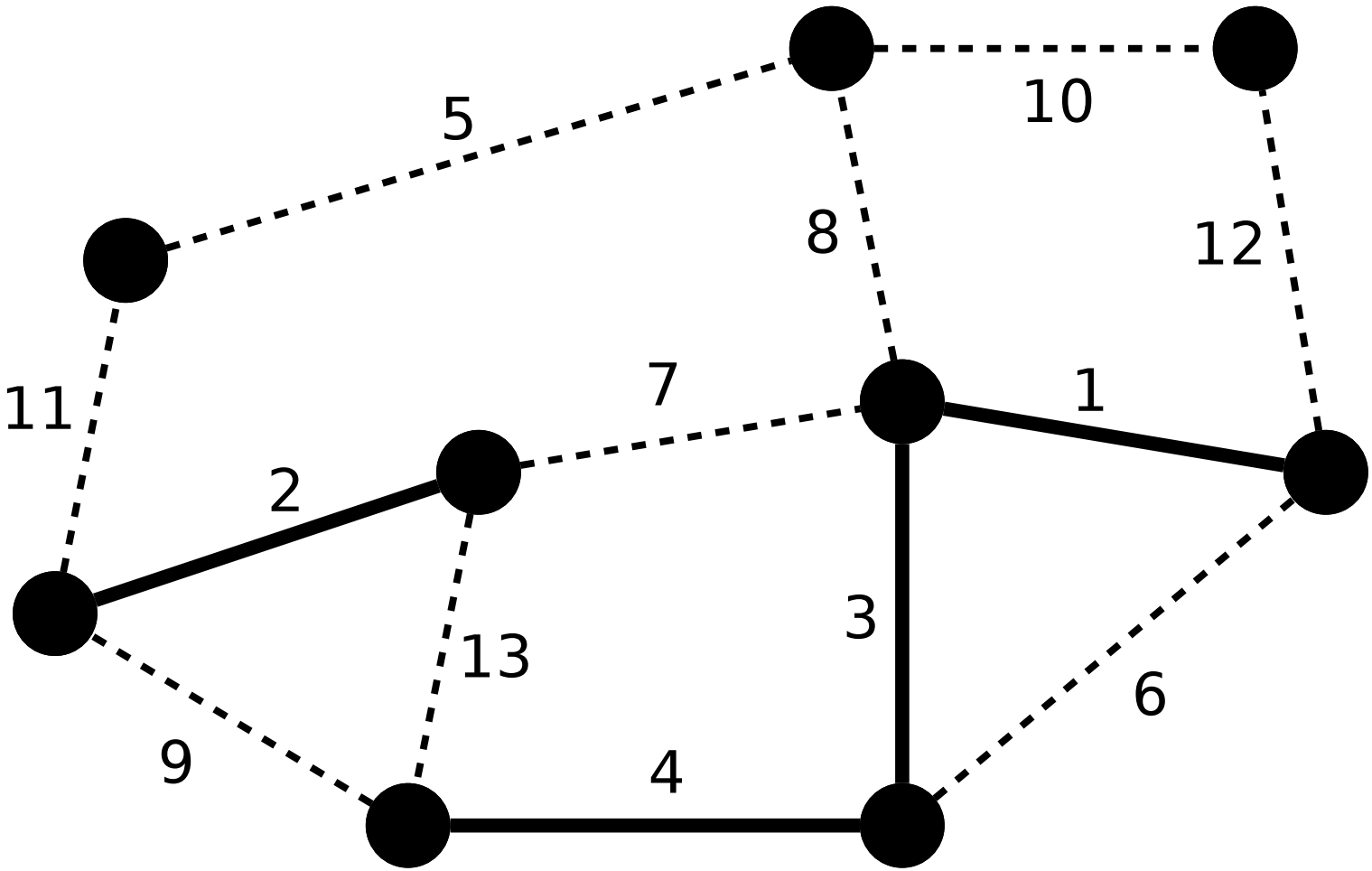
Kruskal's algorithm



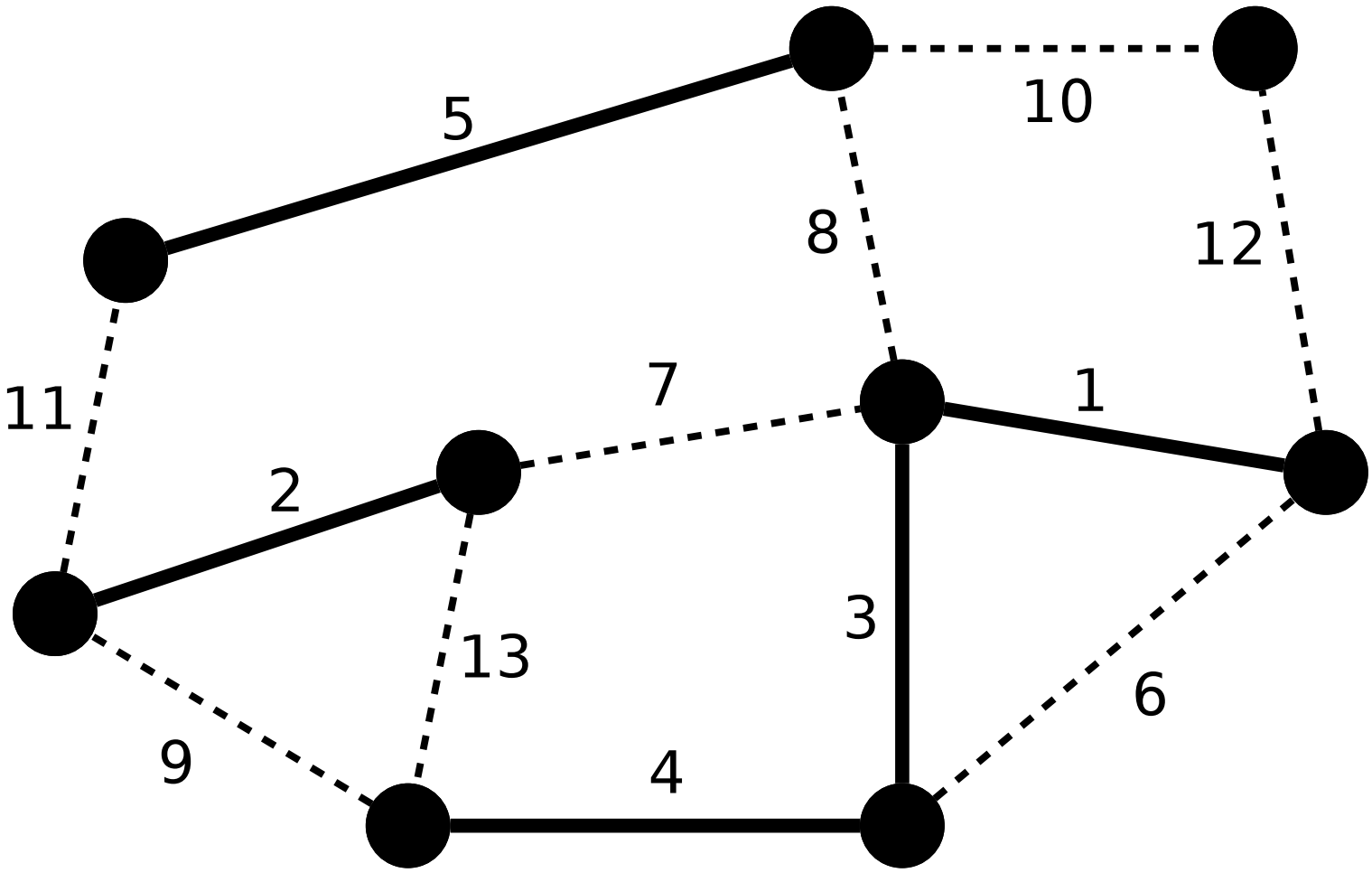
Kruskal's algorithm



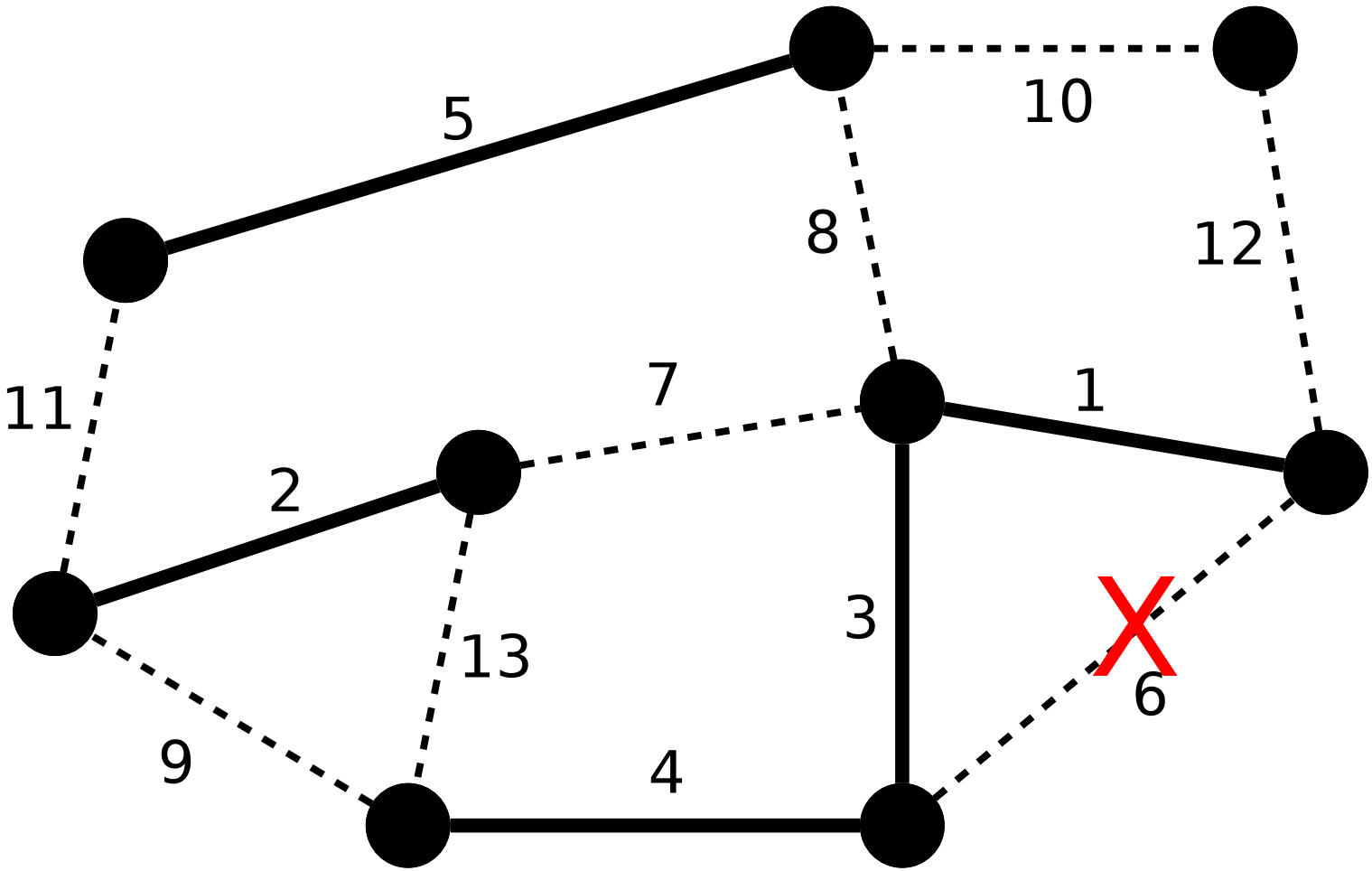
Kruskal's algorithm



Kruskal's algorithm



Kruskal's algorithm



Finding minimum spanning tree

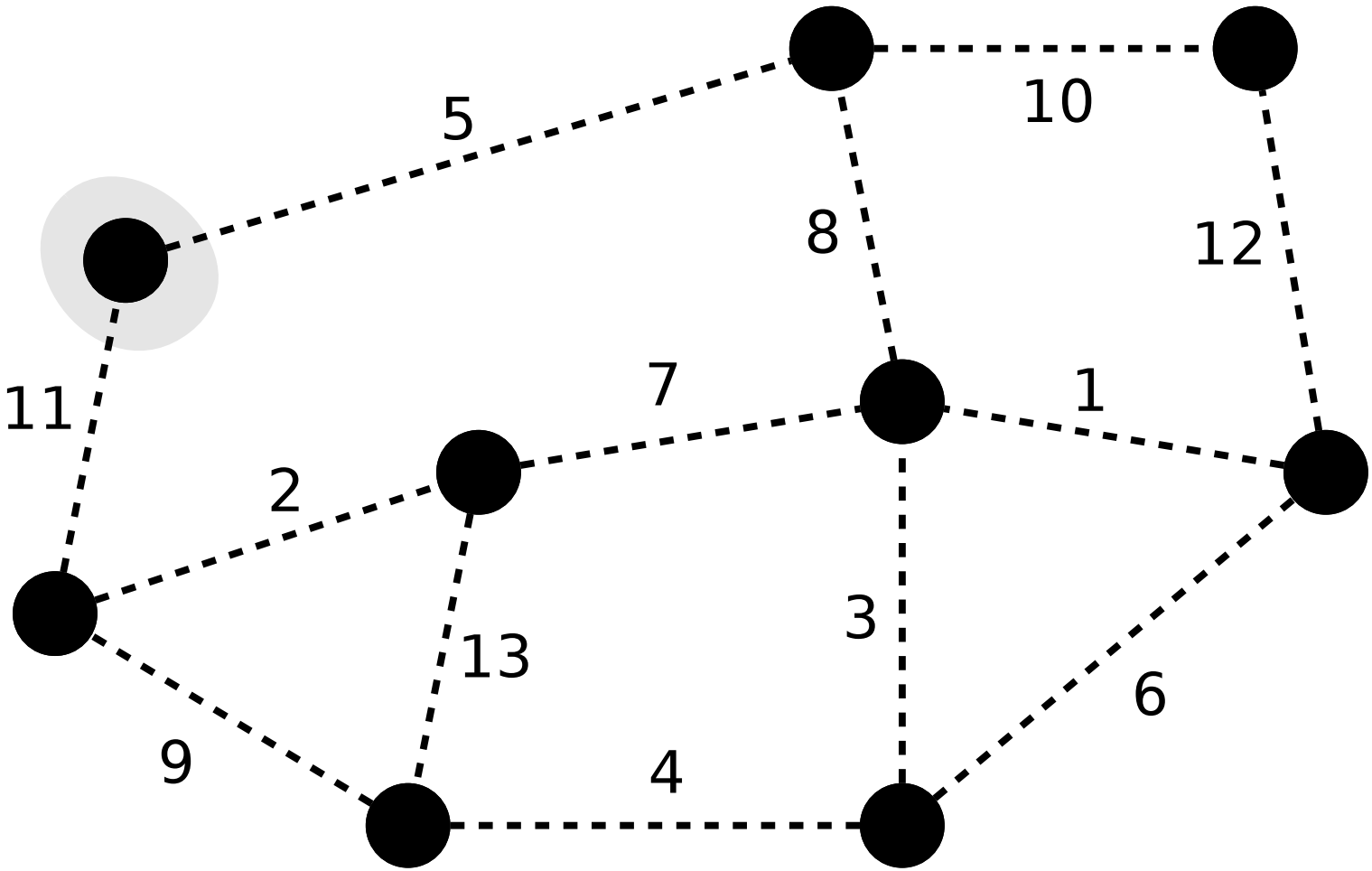
How to find the MST of a given graph?

Fast and simple centralized algorithms:

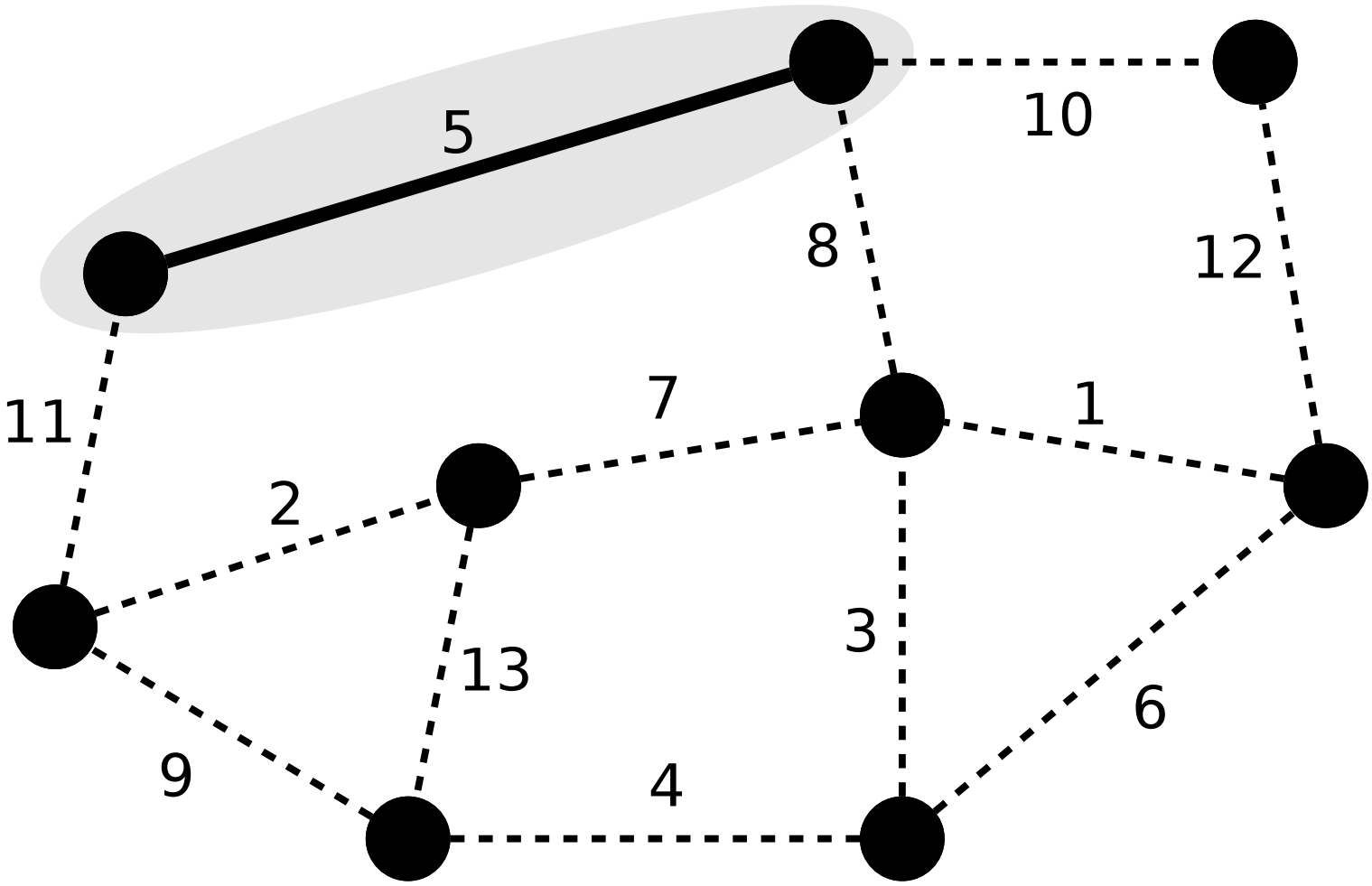
- Kruskal: add edges greedily, avoid cycles.
- Prim: expand a single fragment greedily.

Run in $\mathcal{O}(|E| \log |N|)$.

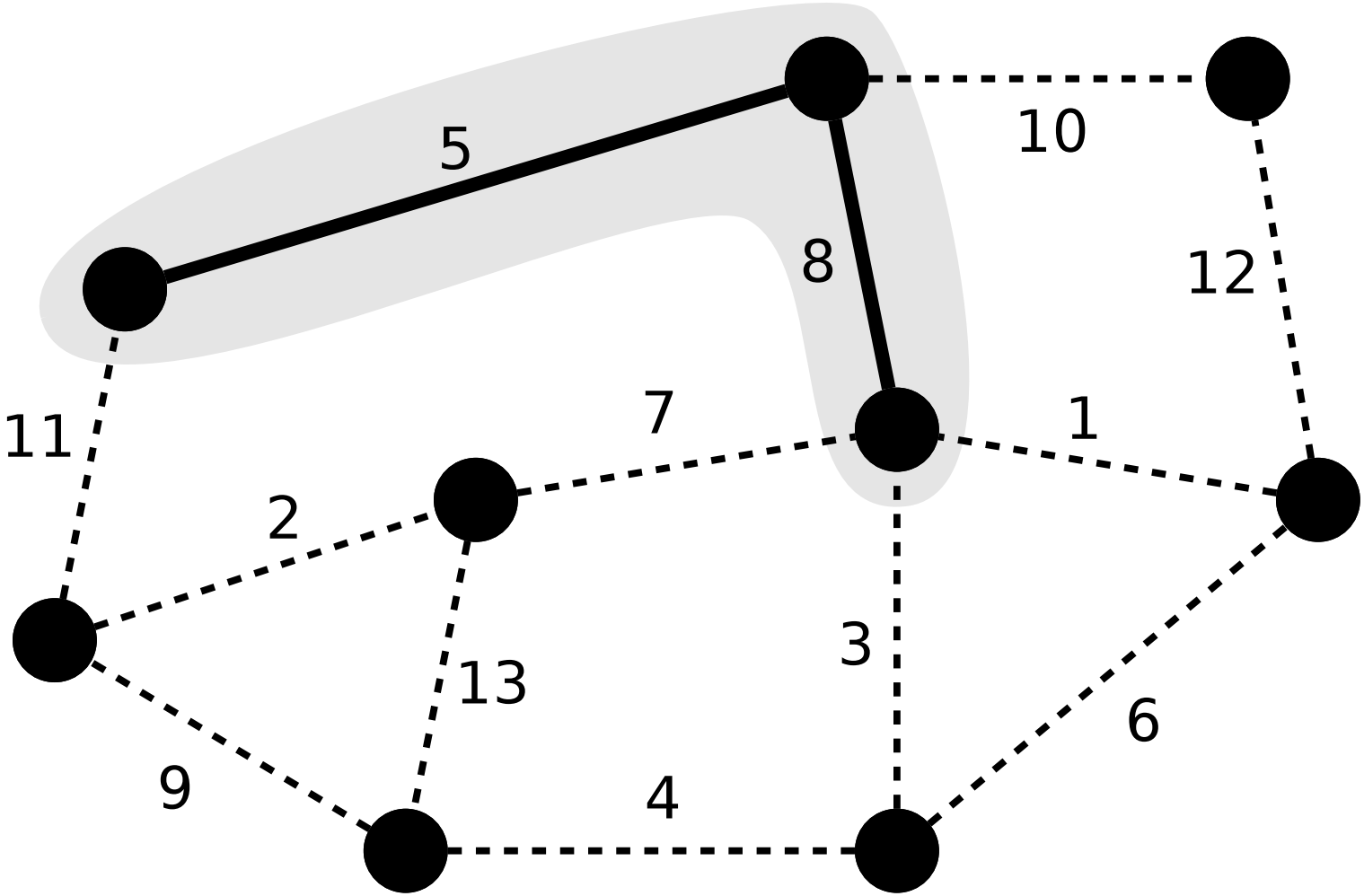
Prim's algorithm



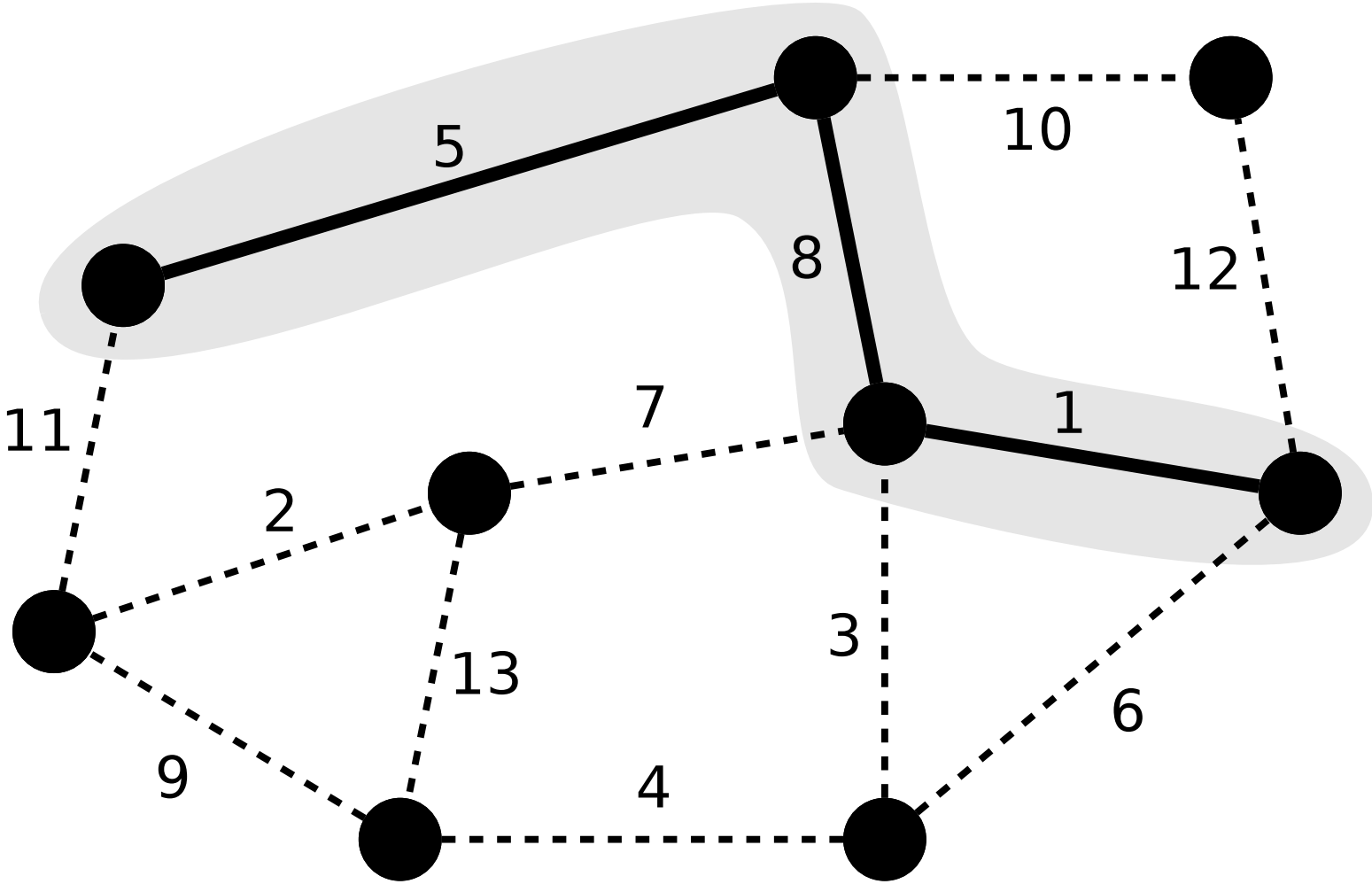
Prim's algorithm



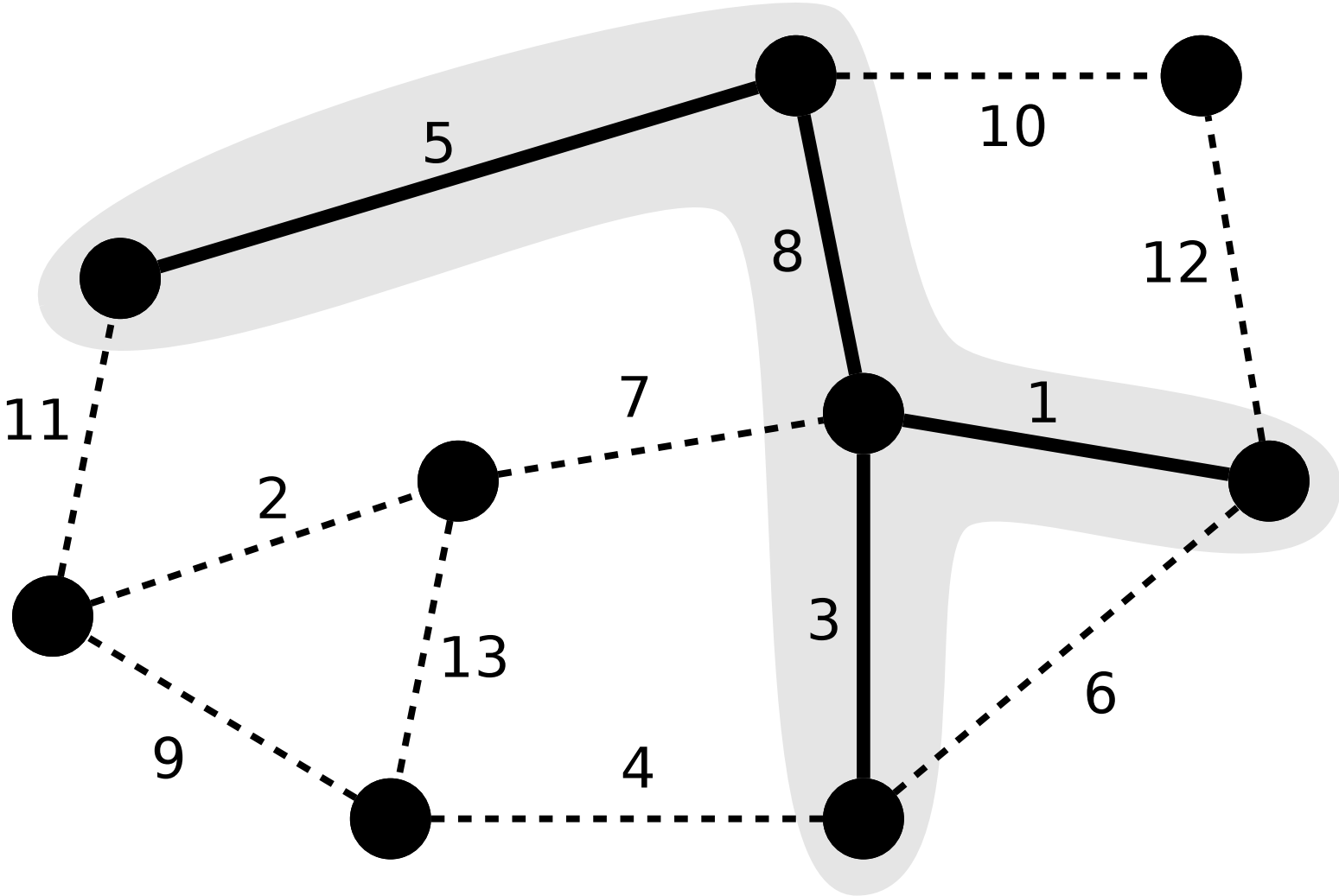
Prim's algorithm



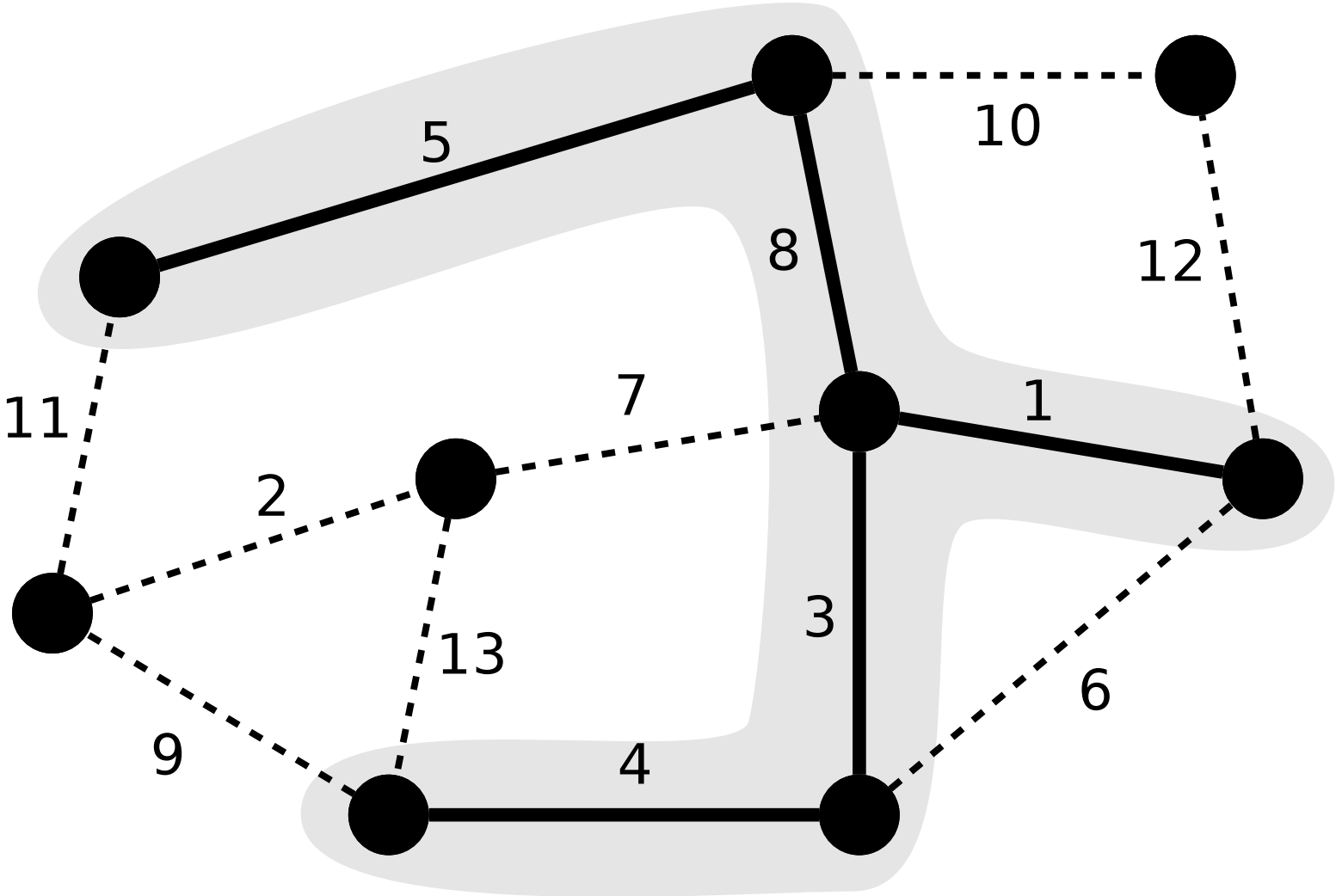
Prim's algorithm



Prim's algorithm



Prim's algorithm



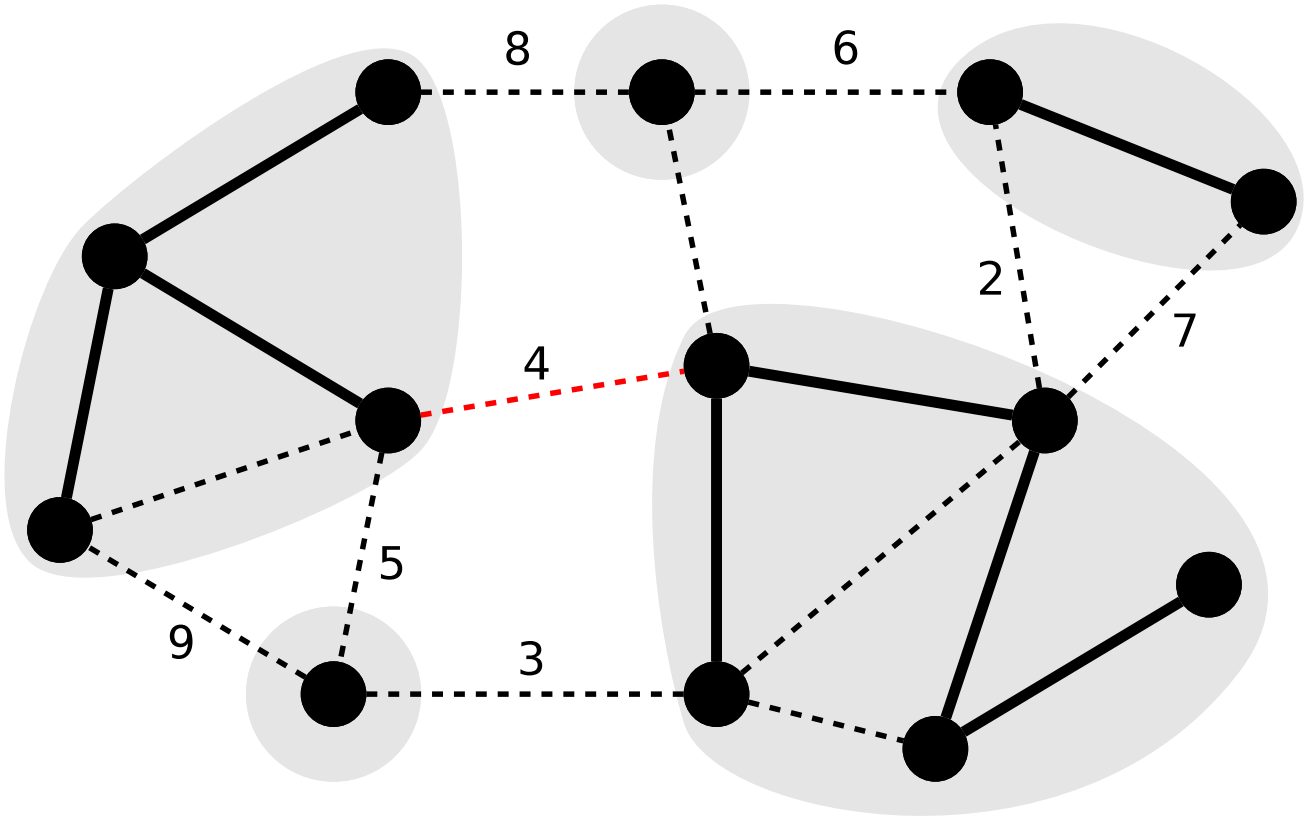
Finding minimum spanning tree

Fast and common in the centralized setting.
Slow in the distributed setting:

- Kruskal: requires global knowledge of the graph.
- Prim: requires communication within a large fragment.

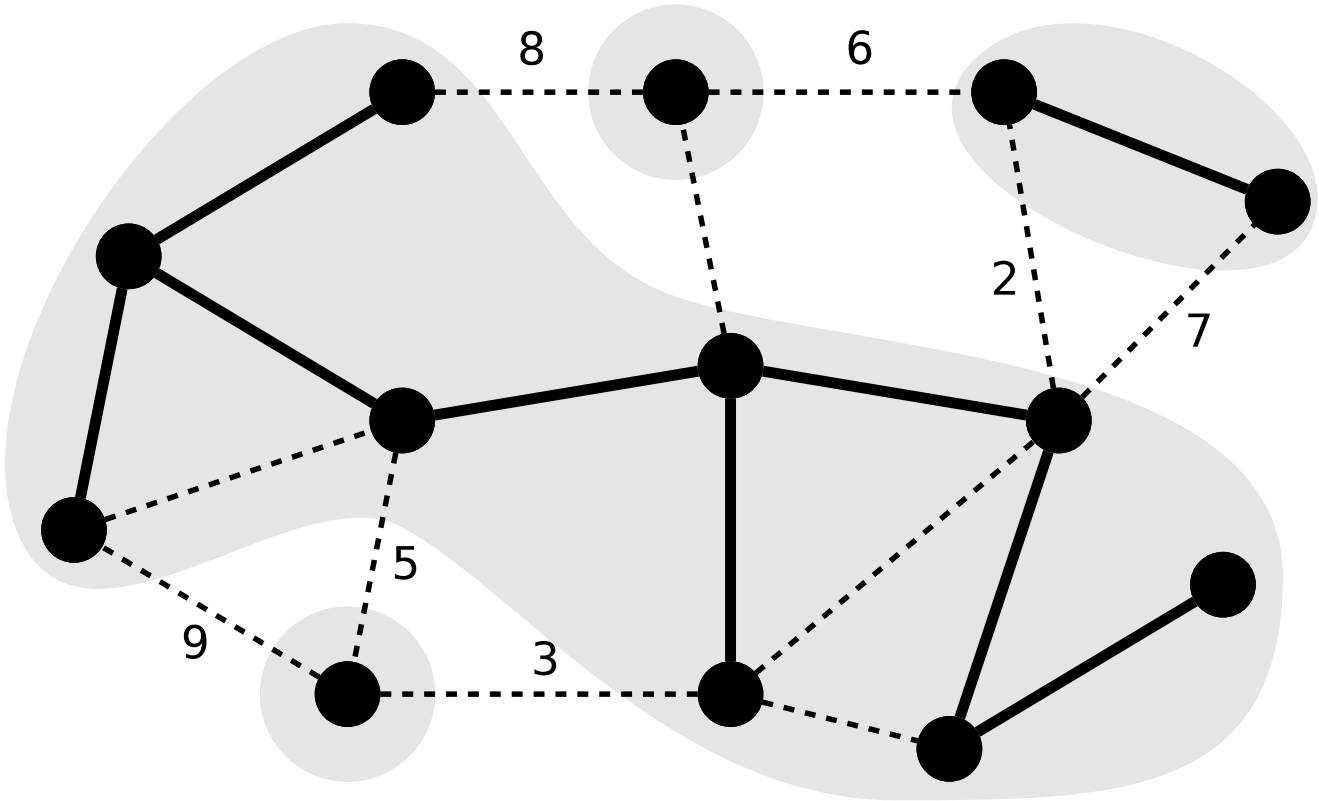
Finding minimum spanning tree

Adding the minimum weight outgoing edge of *any* known fragment of the MST produces a new fragment of the MST.



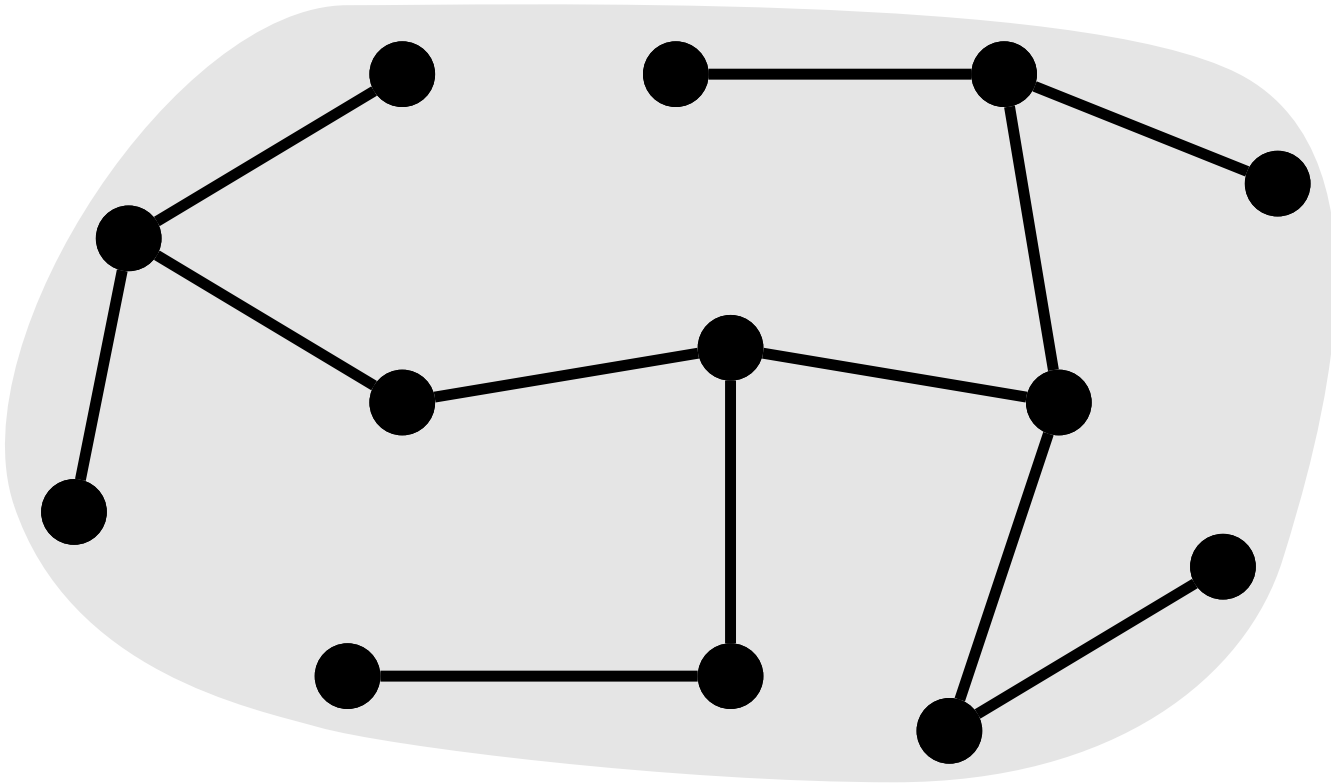
Finding minimum spanning tree

Idea: Start with one fragment per node. Each fragment finds the minimum weight outgoing edge independently.



Finding minimum spanning tree

Fragments with a common minimum weight outgoing edge combine until there is one fragment containing the whole MST.



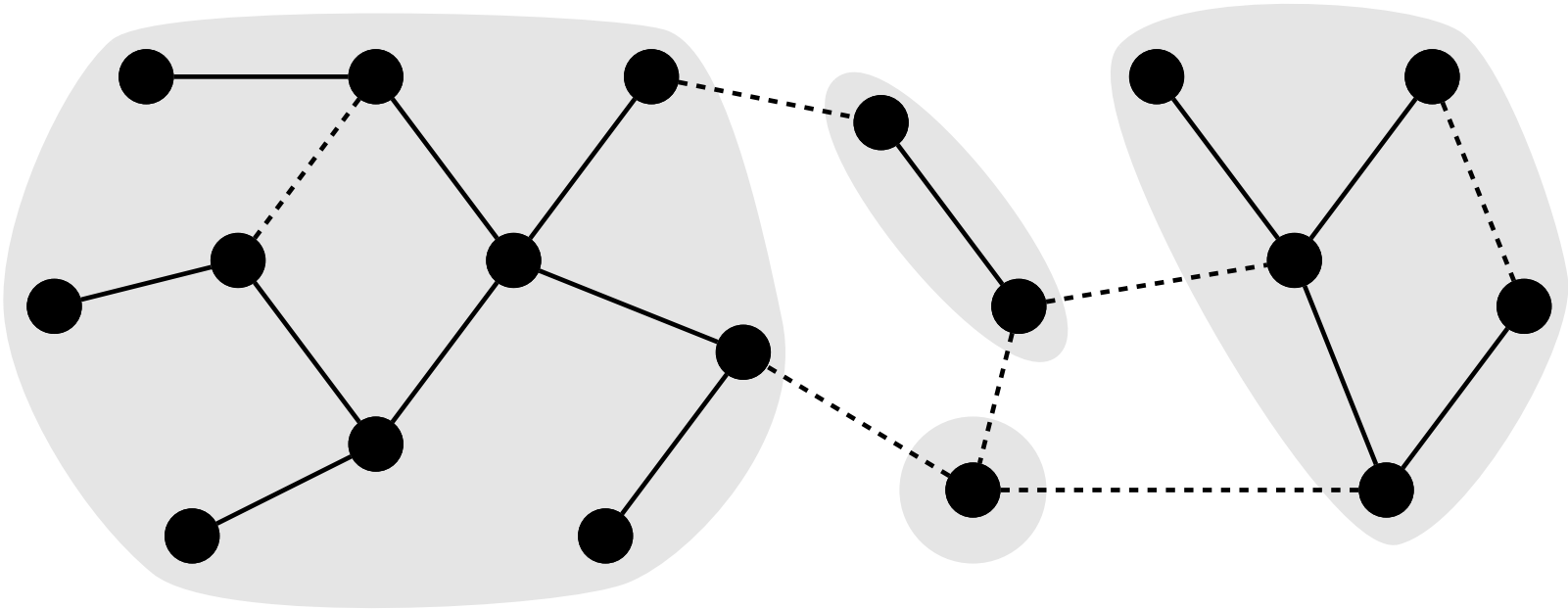
Model of computation

A classical algorithm by Gallager et al. [1983] does this in the asynchronous message passing model:

- Each node runs the same simple local algorithm.
- Each node initially knows only the weights of its incident edges.
- Adjacent nodes can exchange (short) messages.
- Asynchronous: messages arrive after an unknown but bounded delay.
- Performance measured in time and message complexity.

Maintaining fragments on node level

Each node keeps track of which incident edges are part of its fragment.
How to find the minimum weight outgoing edge?
Trivial for single nodes: just pick the minimum weight incident edge.

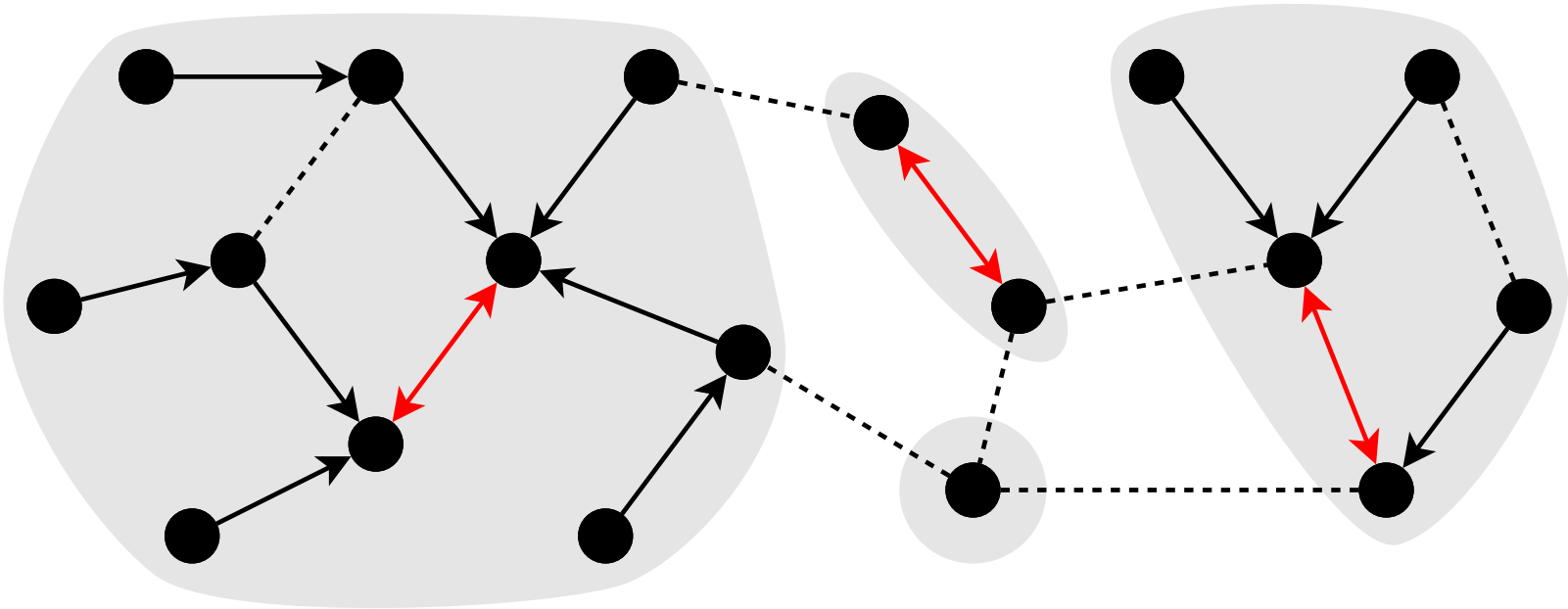


All nodes know which incident edges are in the fragment.

Maintaining fragments on node level

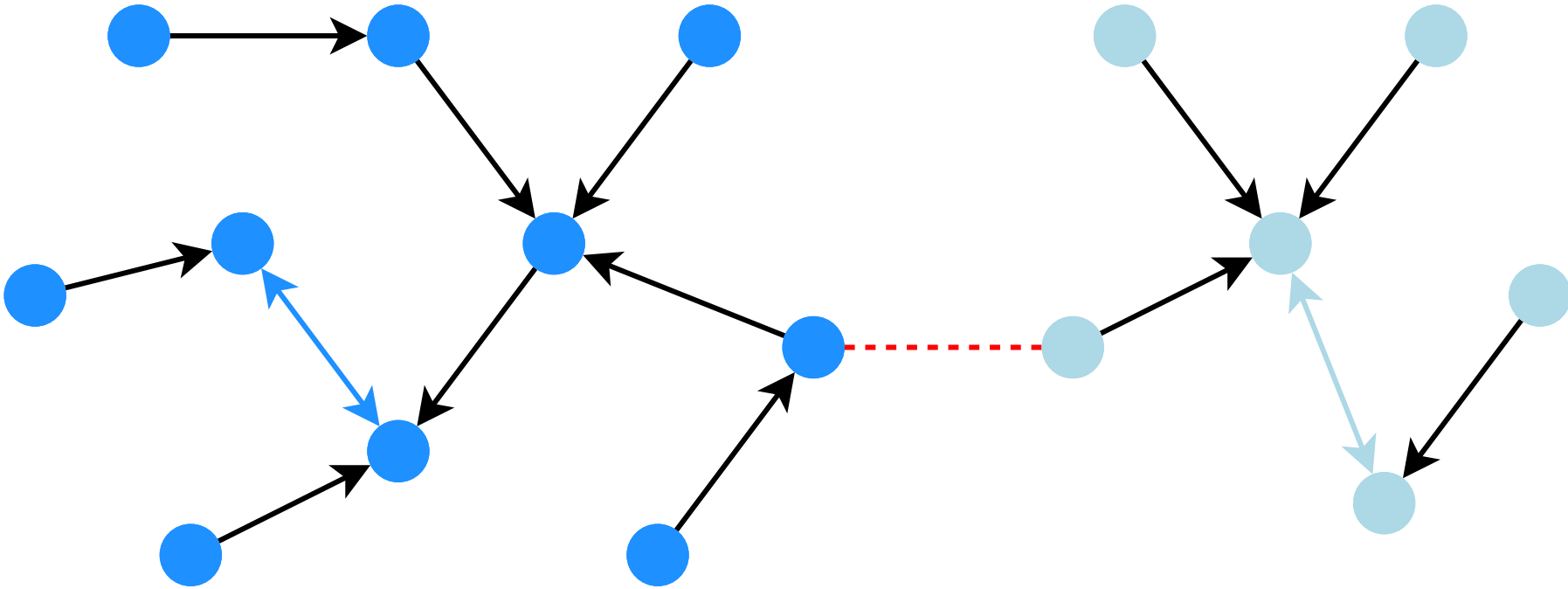
How about larger fragments? When two fragments combine, the edge joining them becomes the core of the new fragment. It serves as

1. a unique identity of the fragment
2. a hub of computation within the fragment

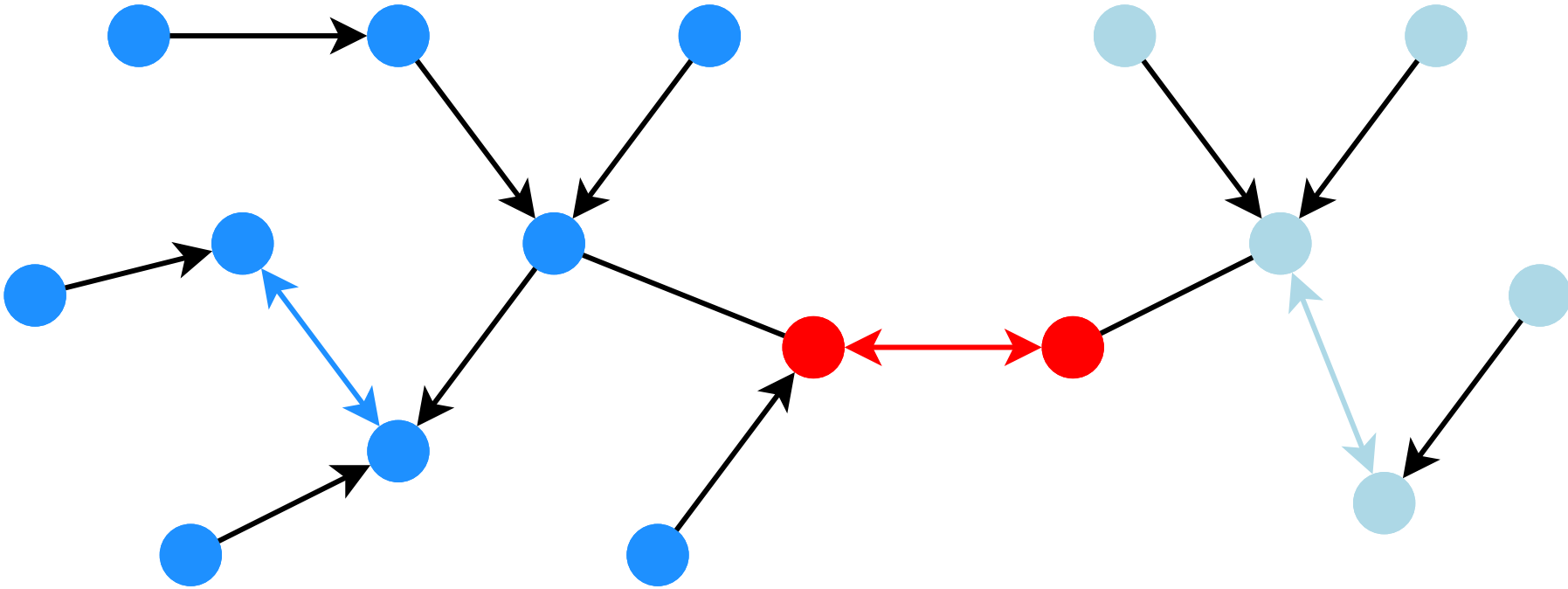


All nodes know the identity and the edge leading towards the core.

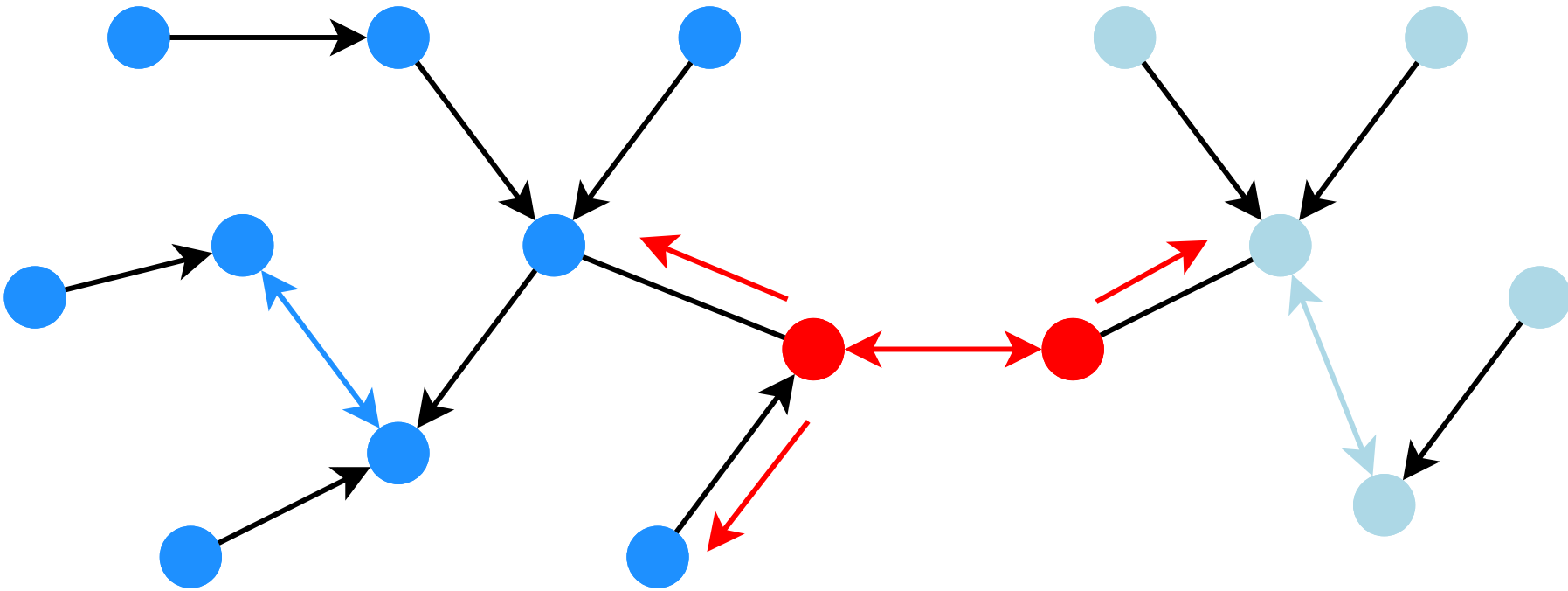
Finding the minimum weight outgoing edge



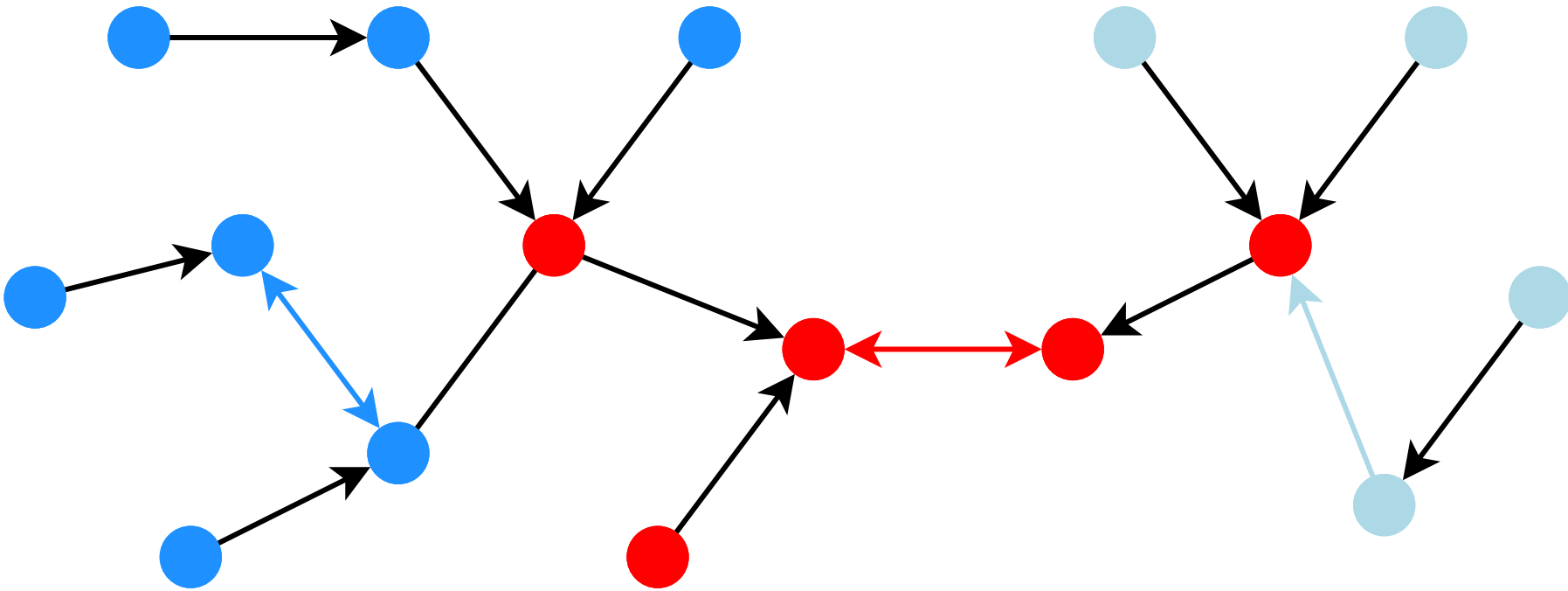
Finding the minimum weight outgoing edge



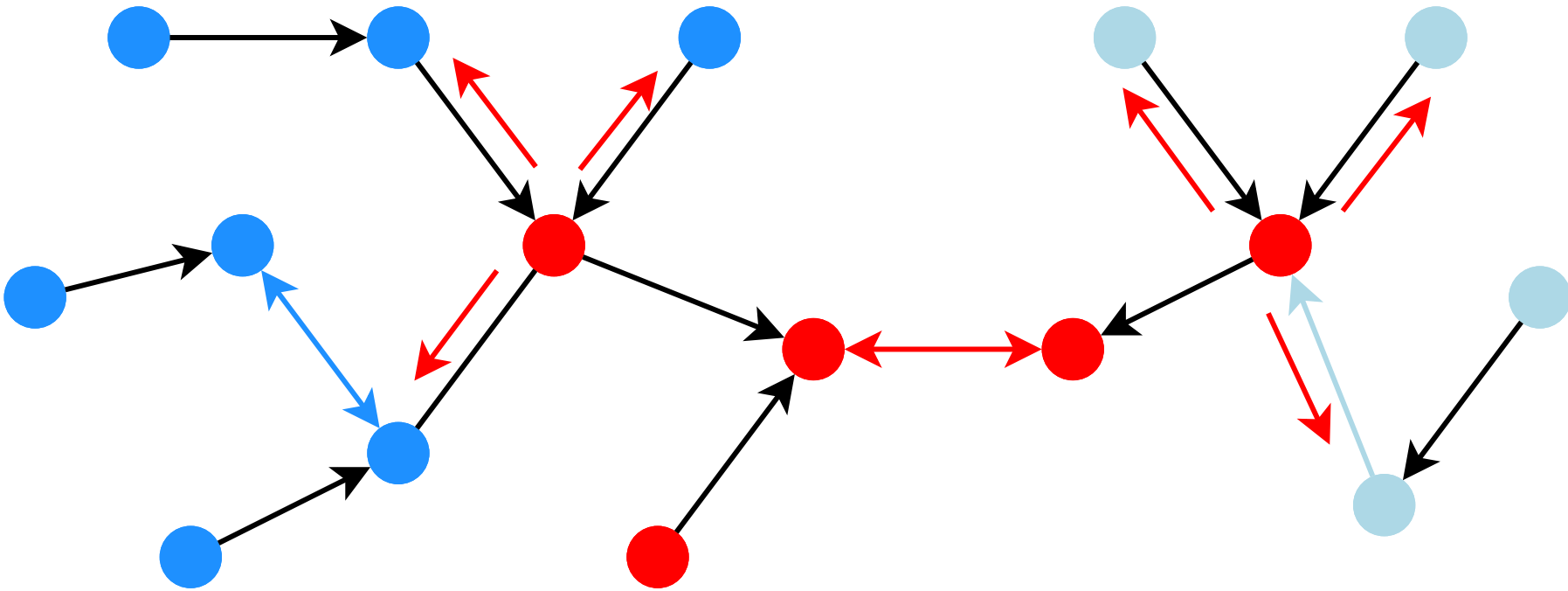
Finding the minimum weight outgoing edge



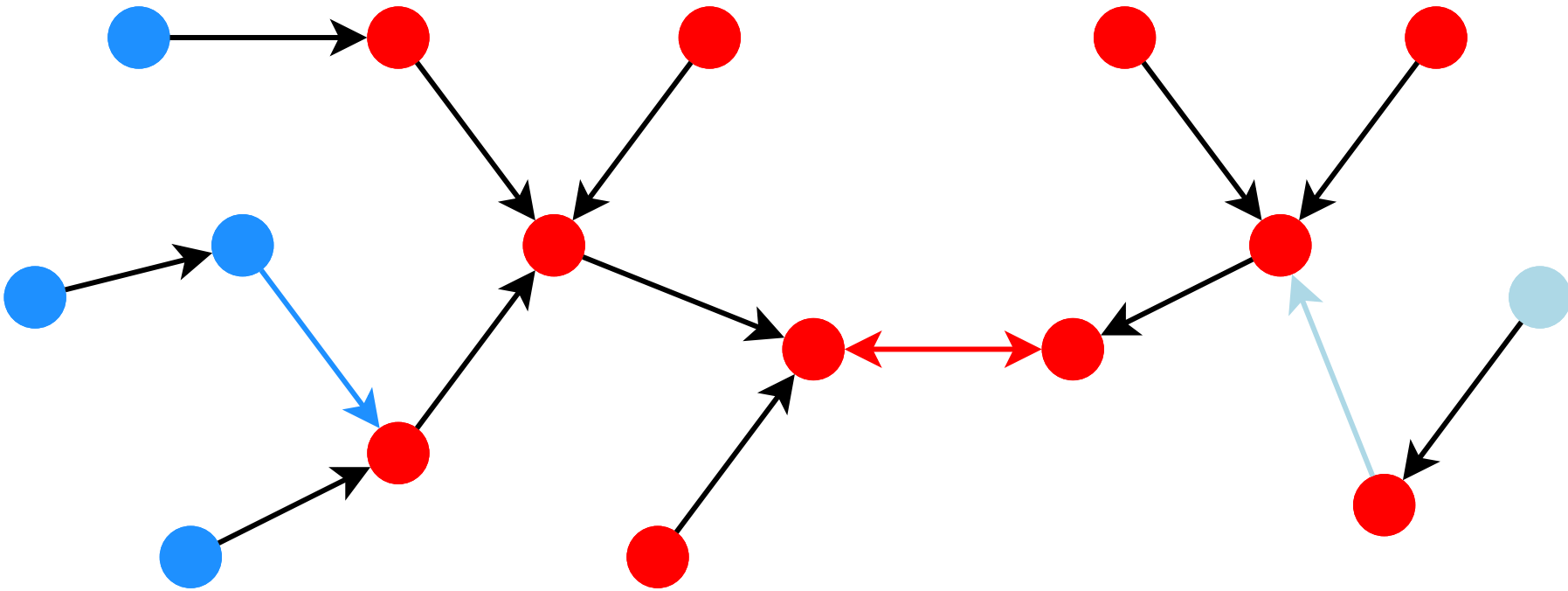
Finding the minimum weight outgoing edge



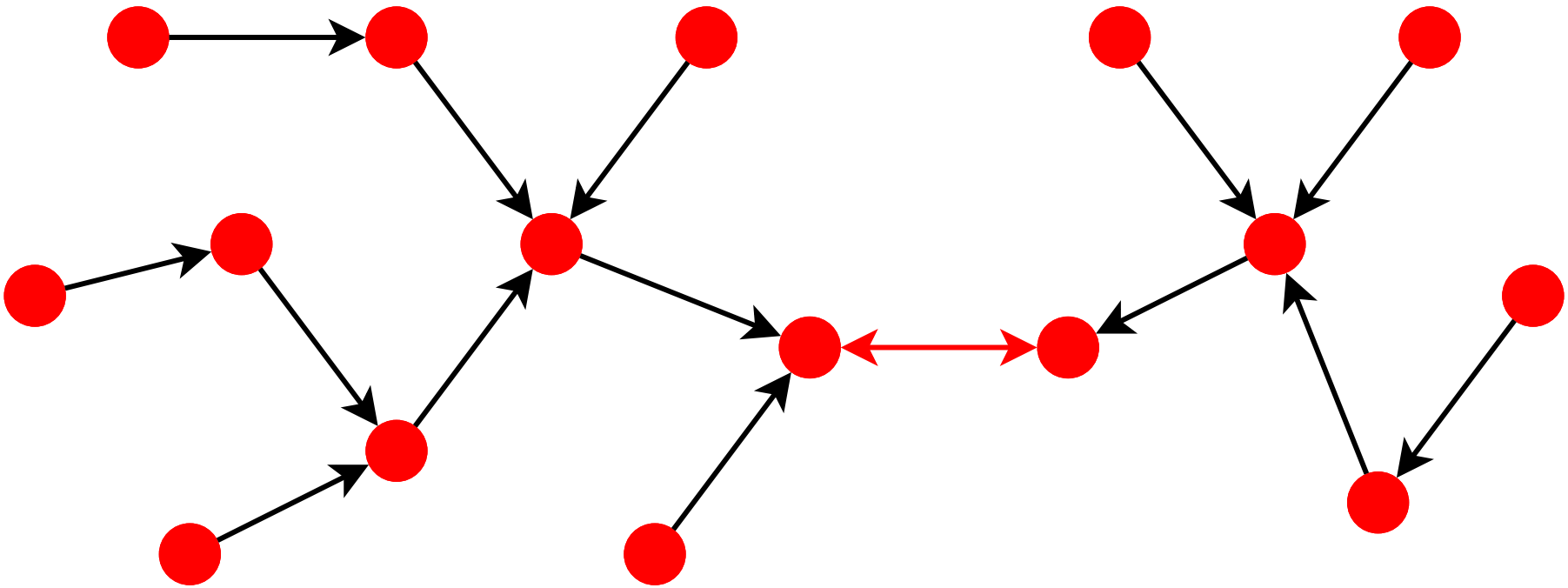
Finding the minimum weight outgoing edge



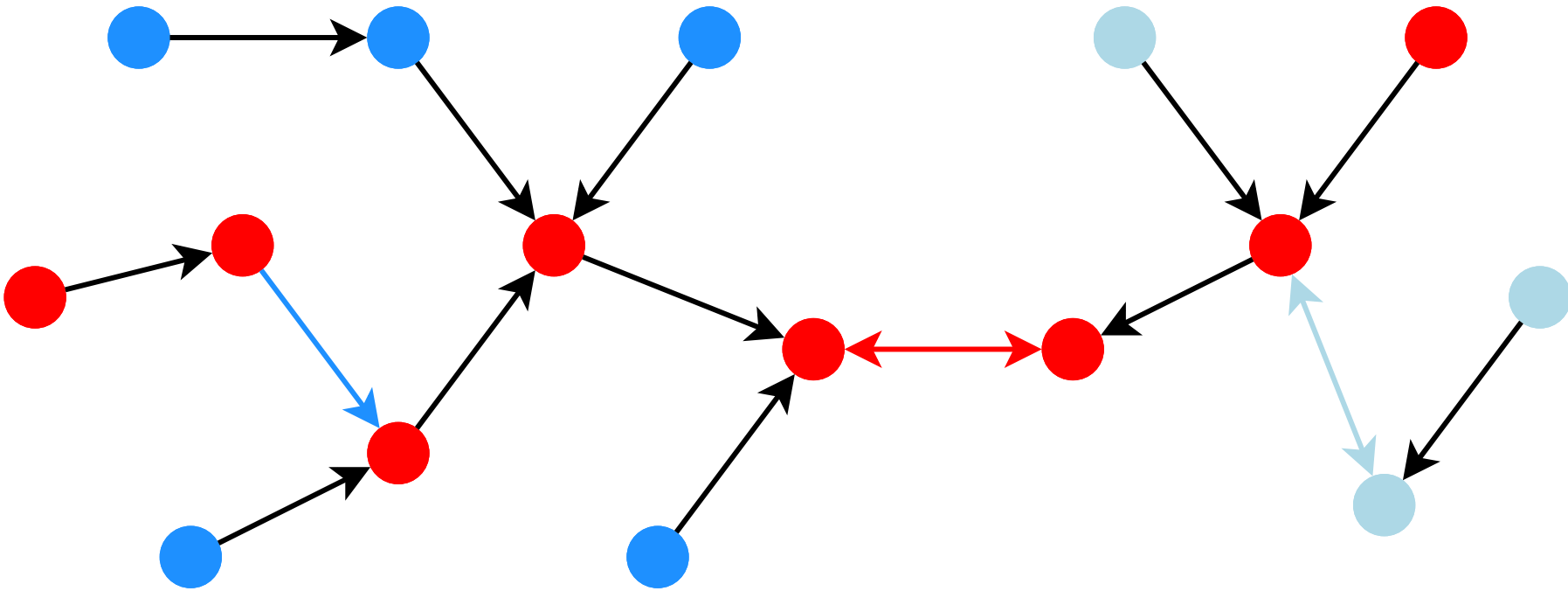
Finding the minimum weight outgoing edge



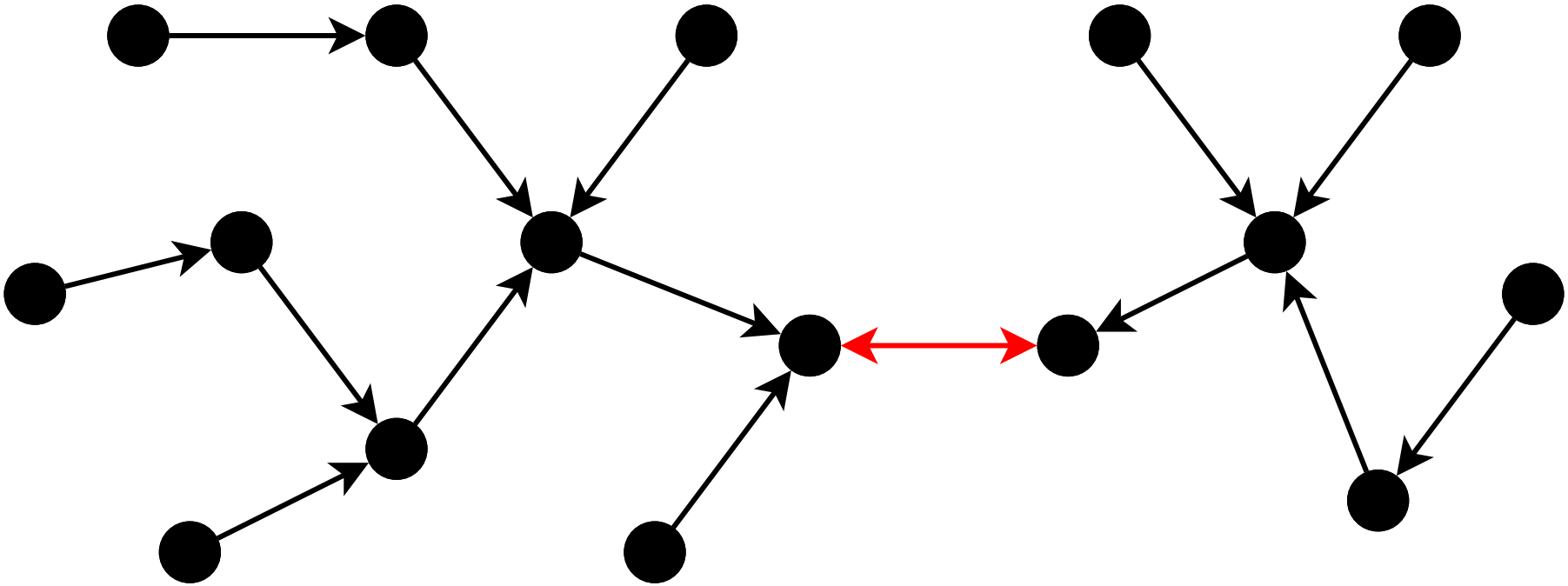
Finding the minimum weight outgoing edge



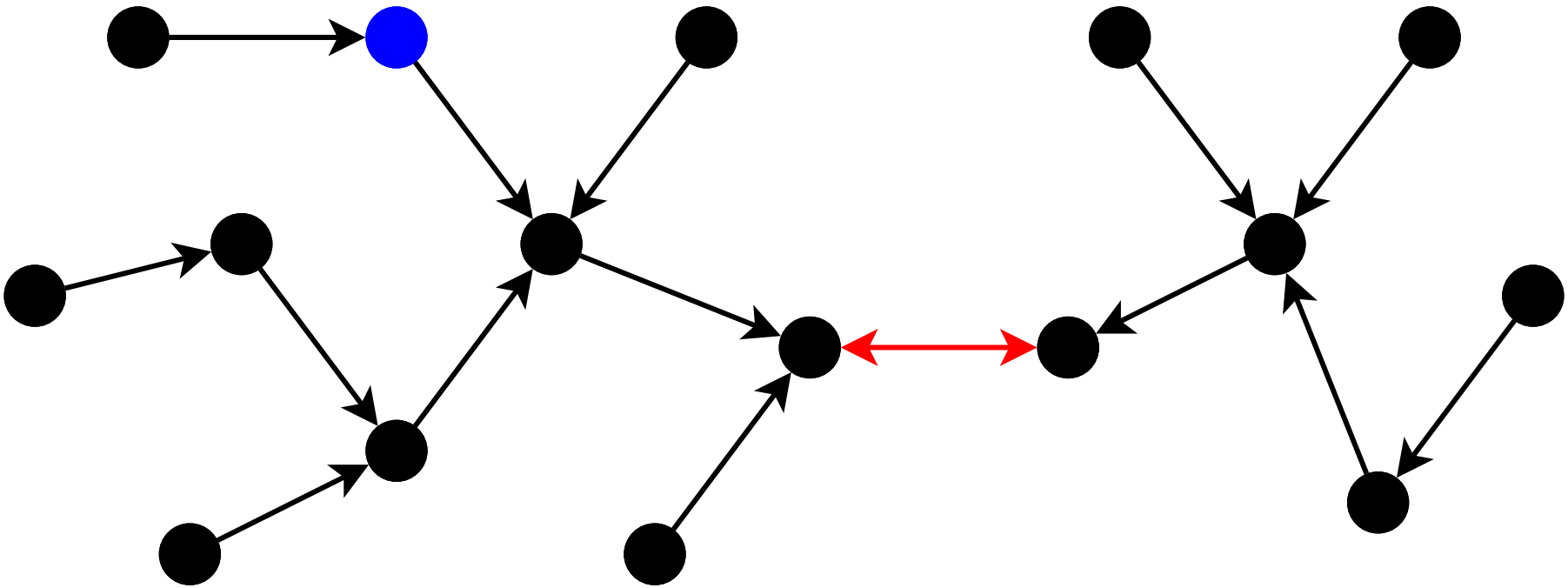
Finding the minimum weight outgoing edge



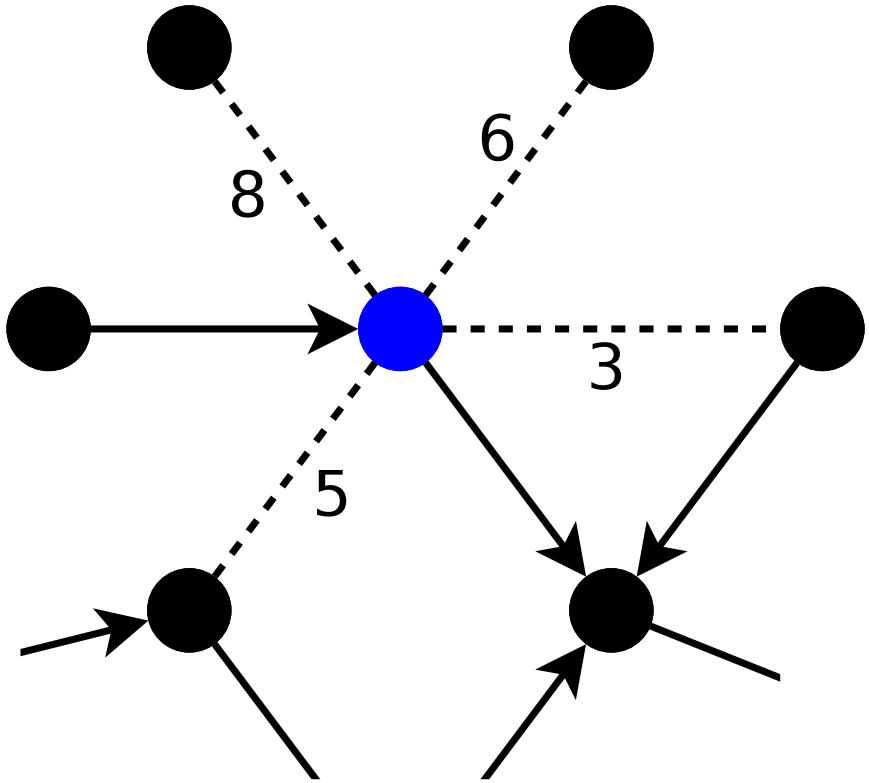
Finding the minimum weight outgoing edge



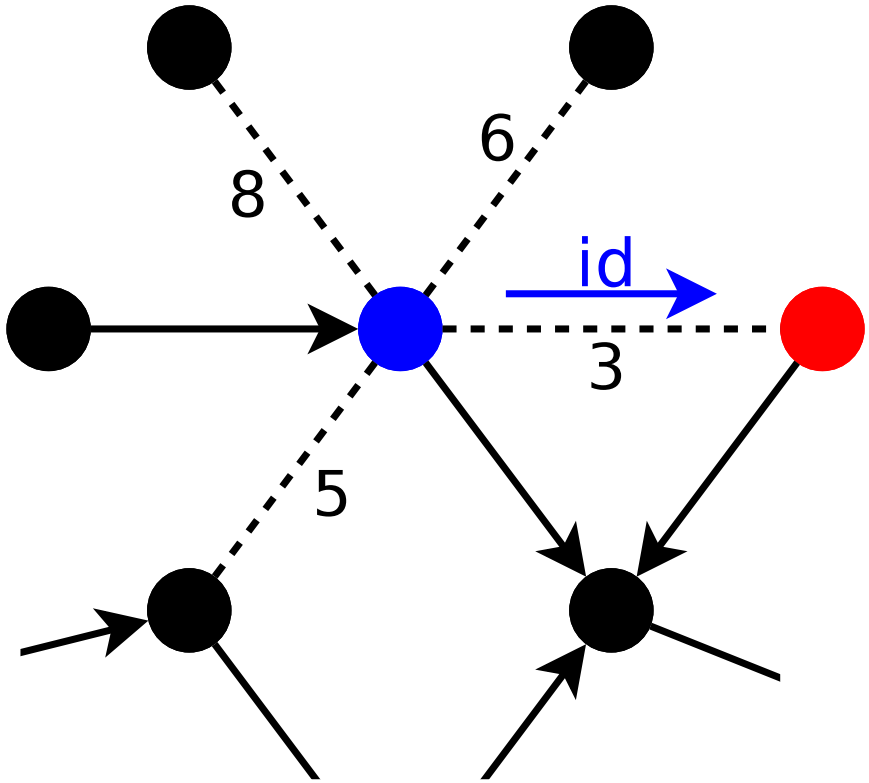
Finding the minimum weight outgoing edge



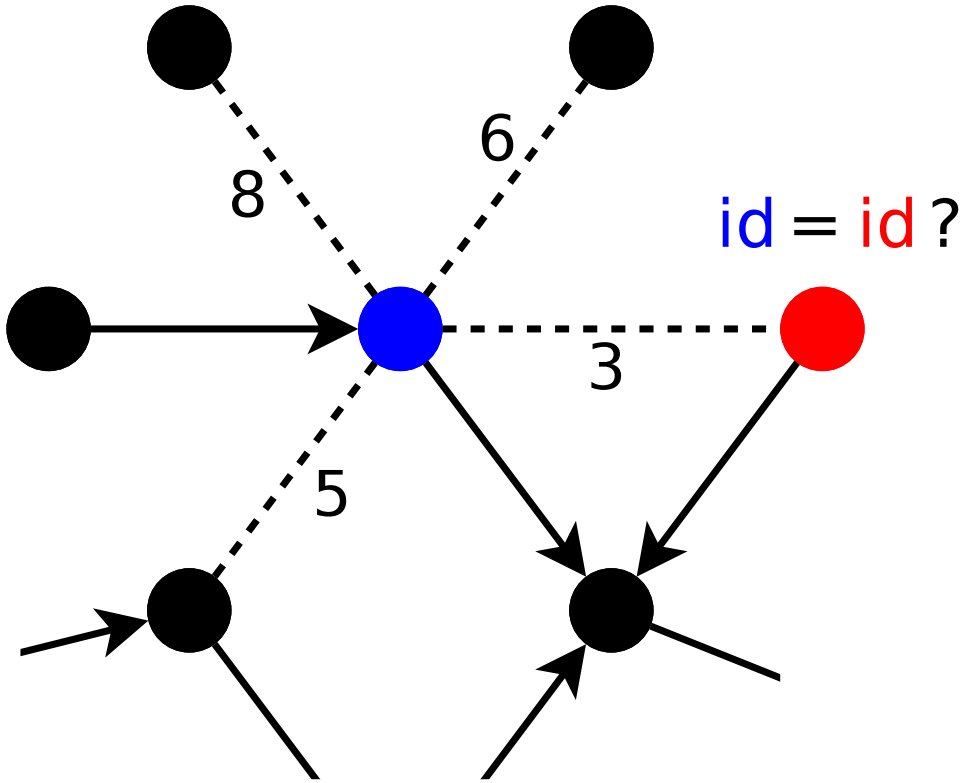
Finding the minimum weight outgoing edge



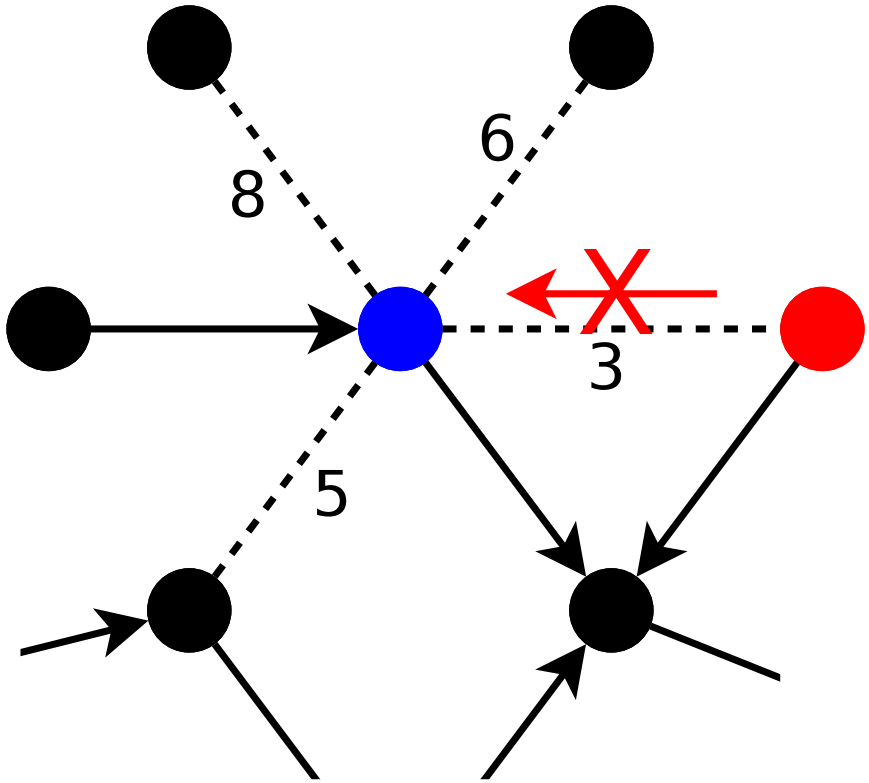
Finding the minimum weight outgoing edge



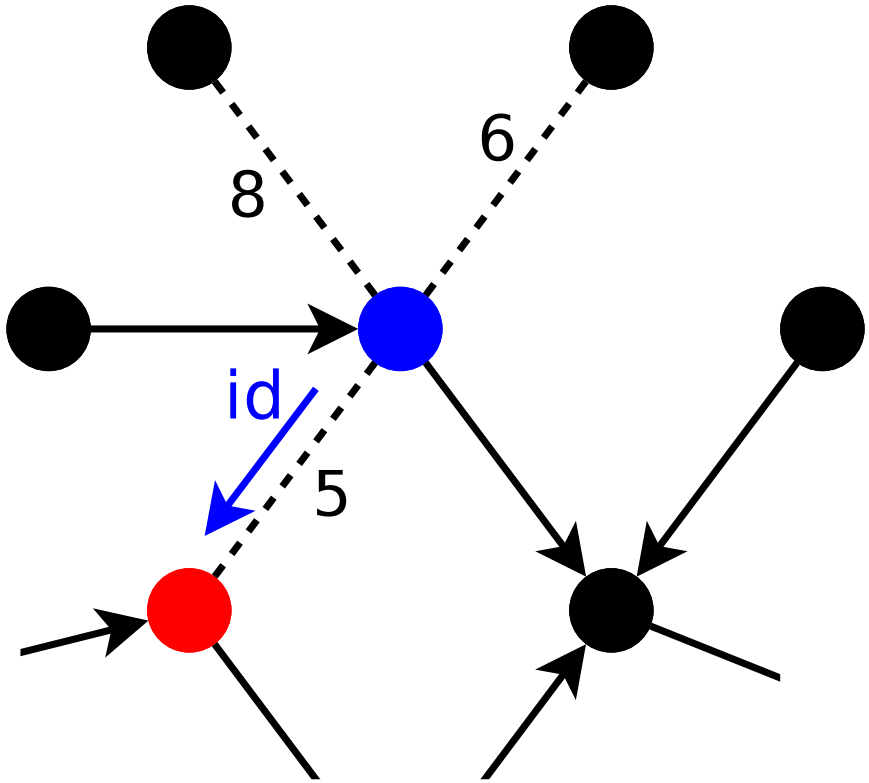
Finding the minimum weight outgoing edge



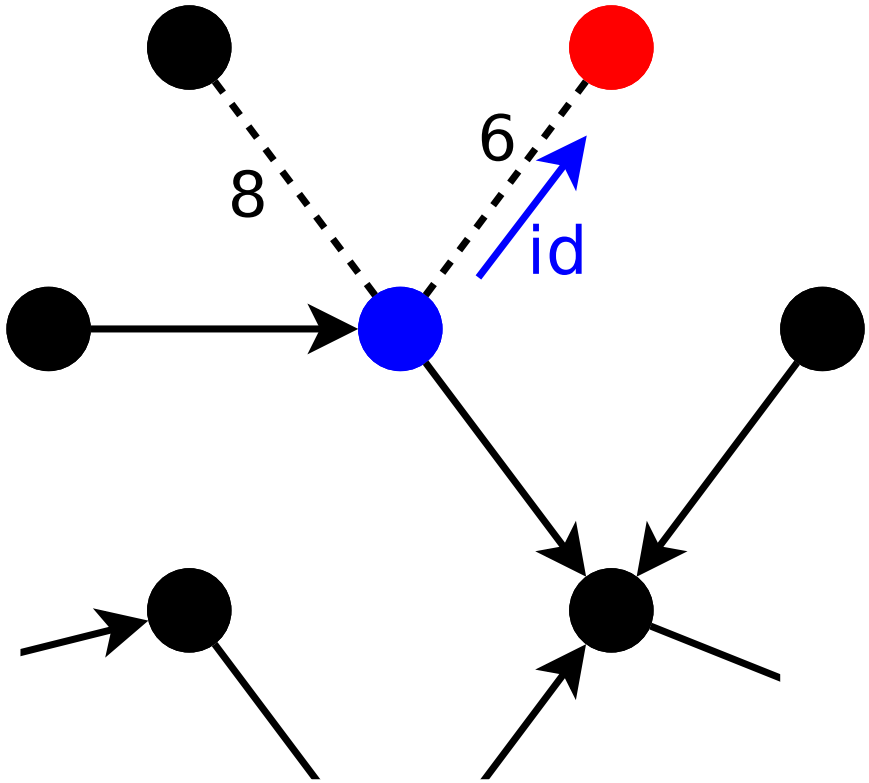
Finding the minimum weight outgoing edge



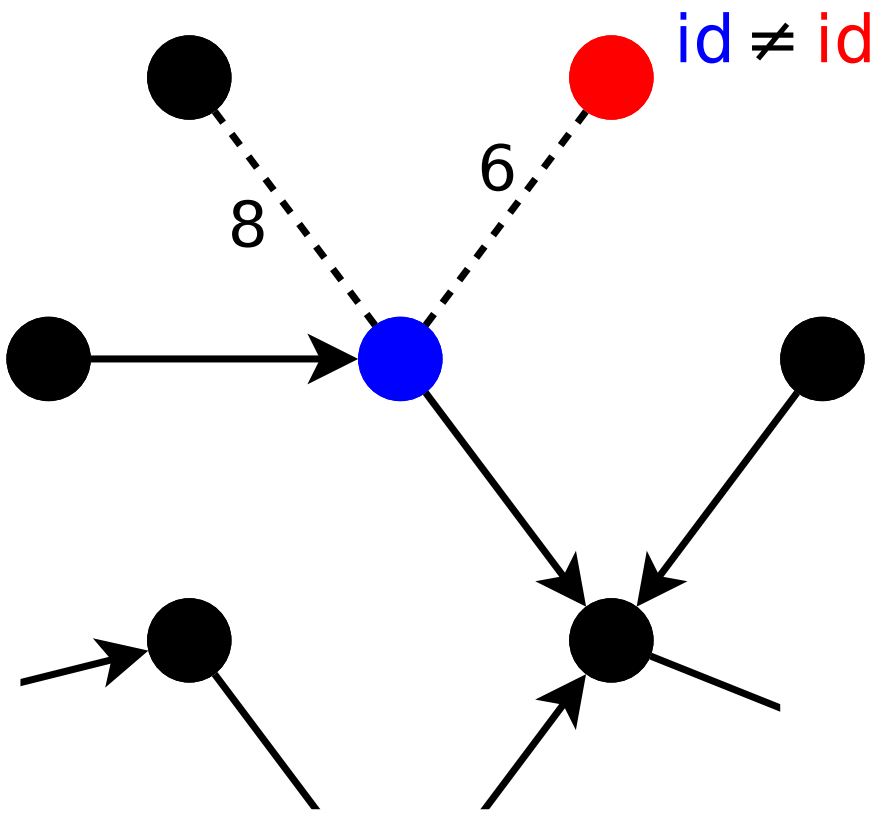
Finding the minimum weight outgoing edge



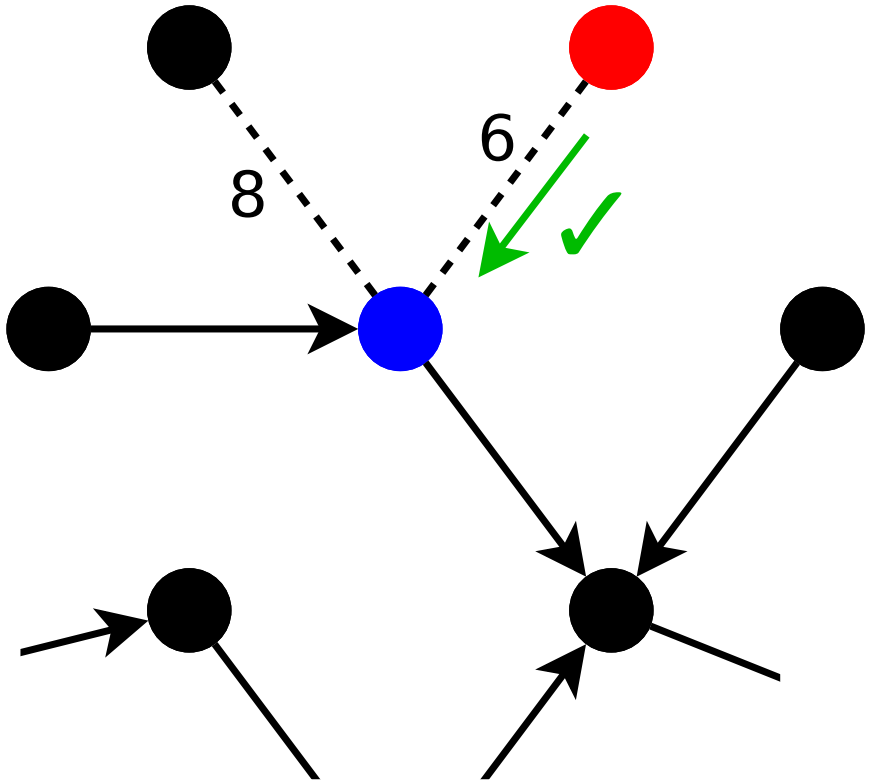
Finding the minimum weight outgoing edge



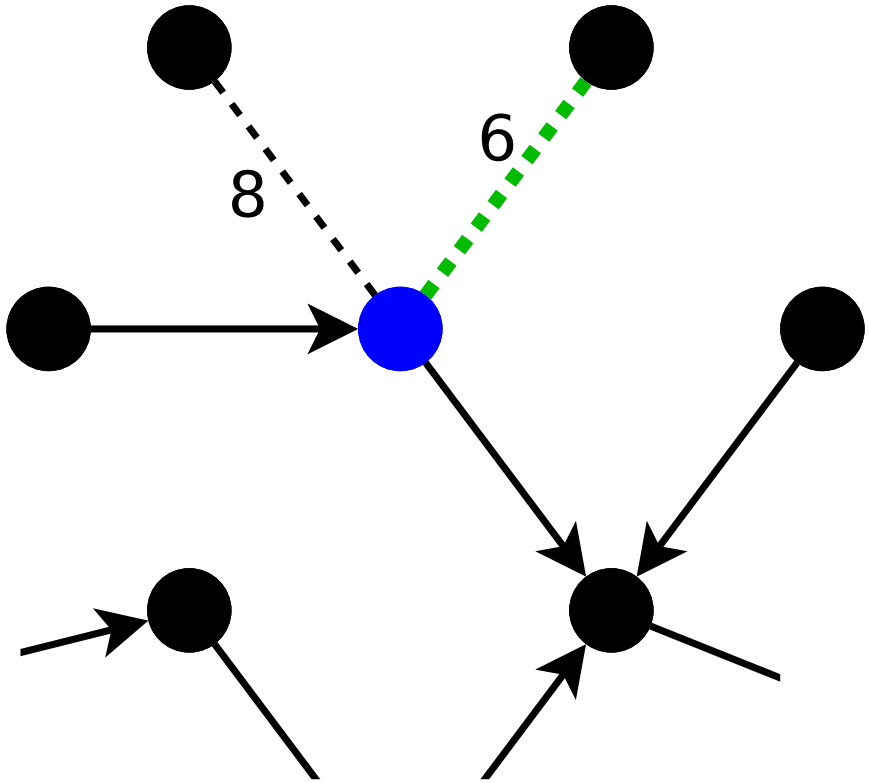
Finding the minimum weight outgoing edge



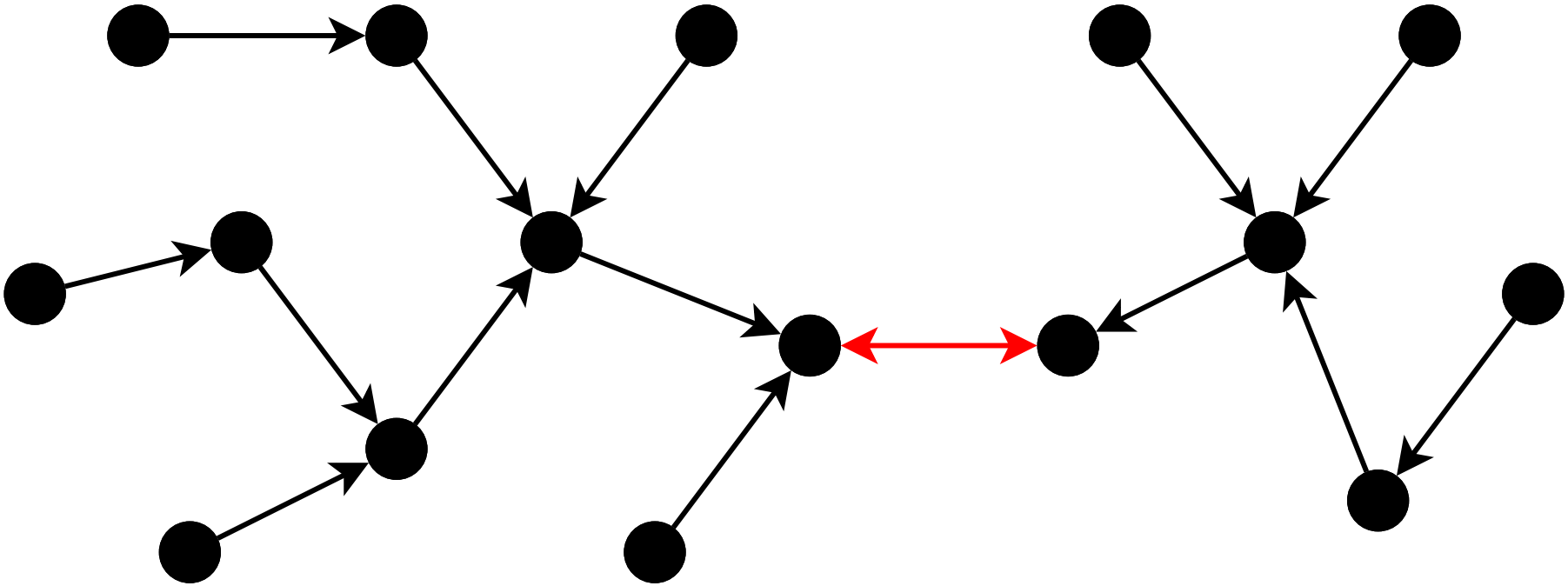
Finding the minimum weight outgoing edge



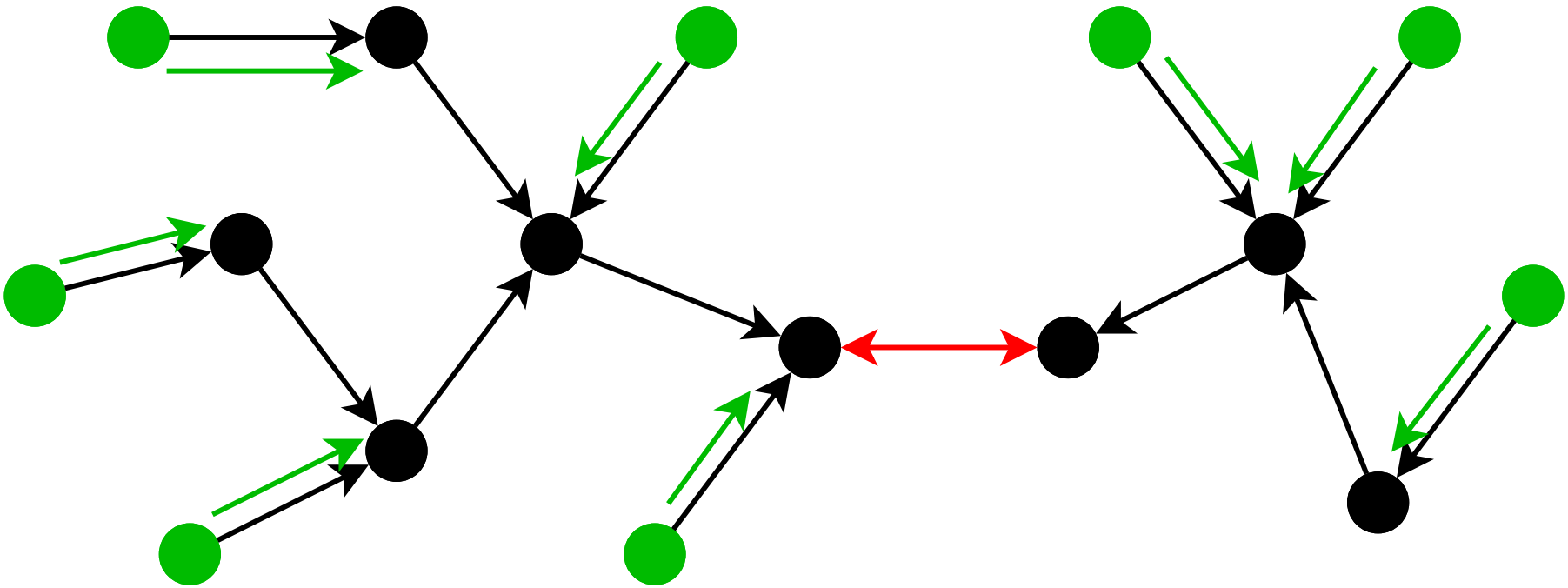
Finding the minimum weight outgoing edge



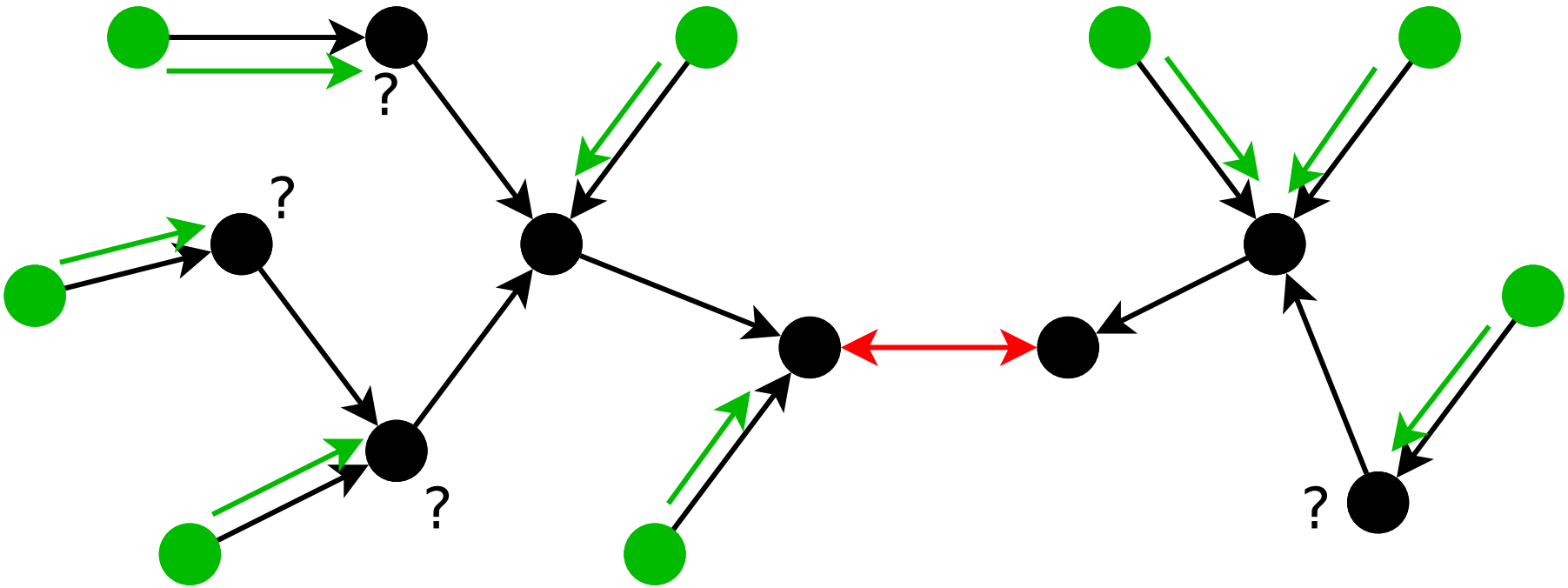
Finding the minimum weight outgoing edge



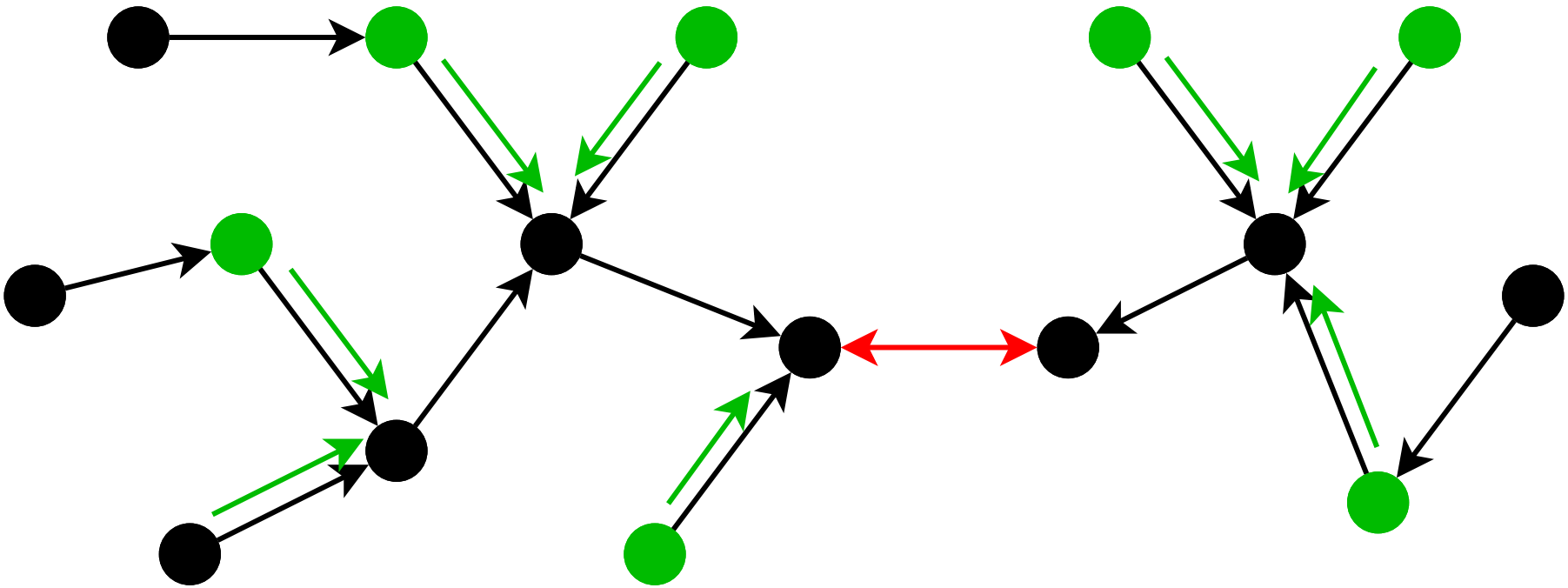
Finding the minimum weight outgoing edge



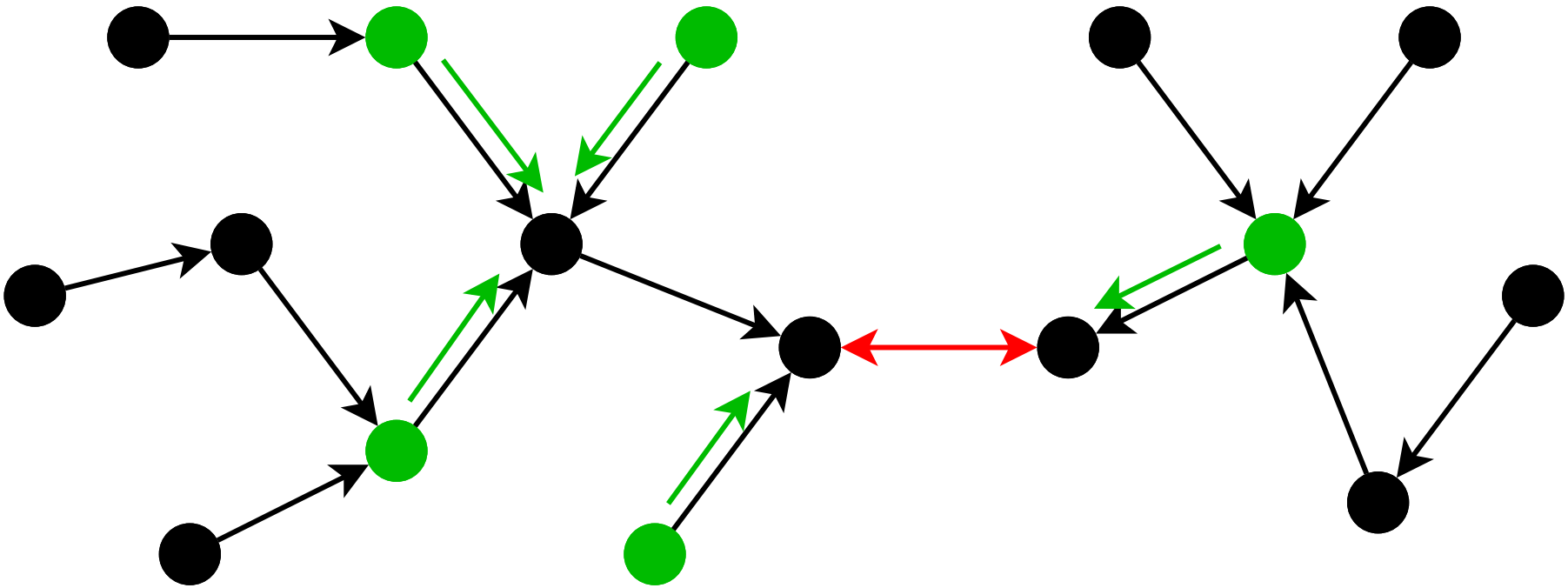
Finding the minimum weight outgoing edge



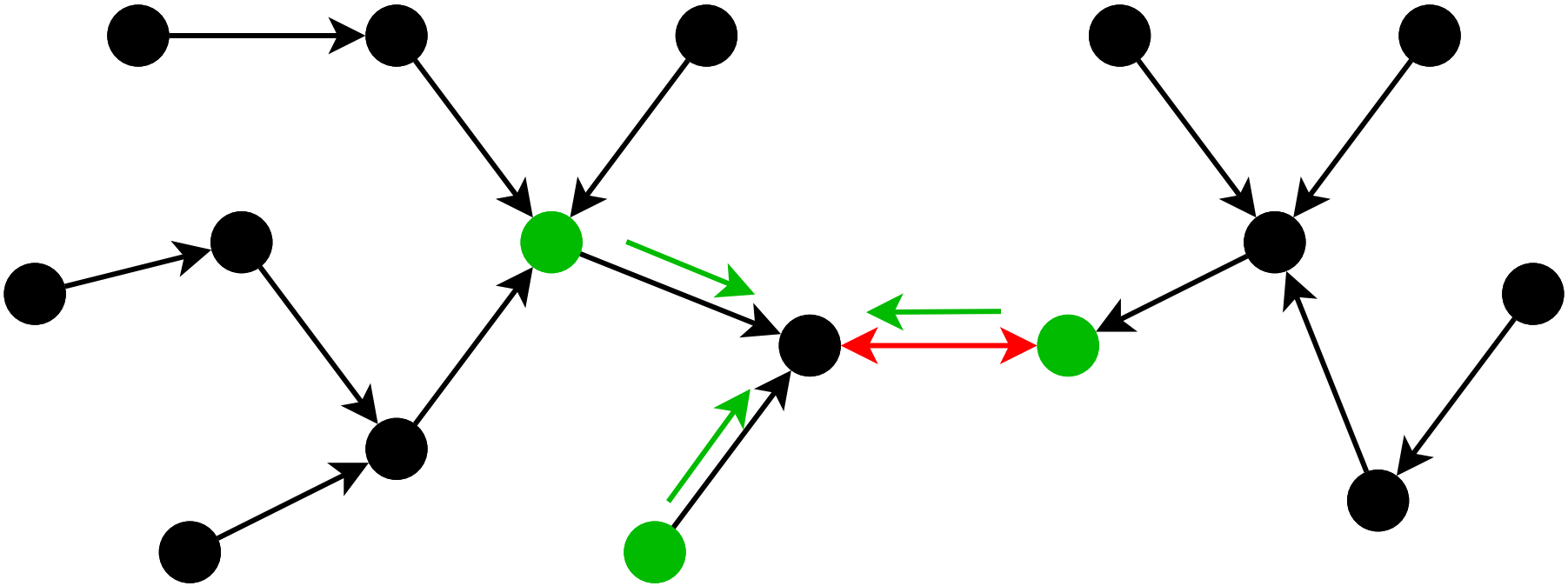
Finding the minimum weight outgoing edge



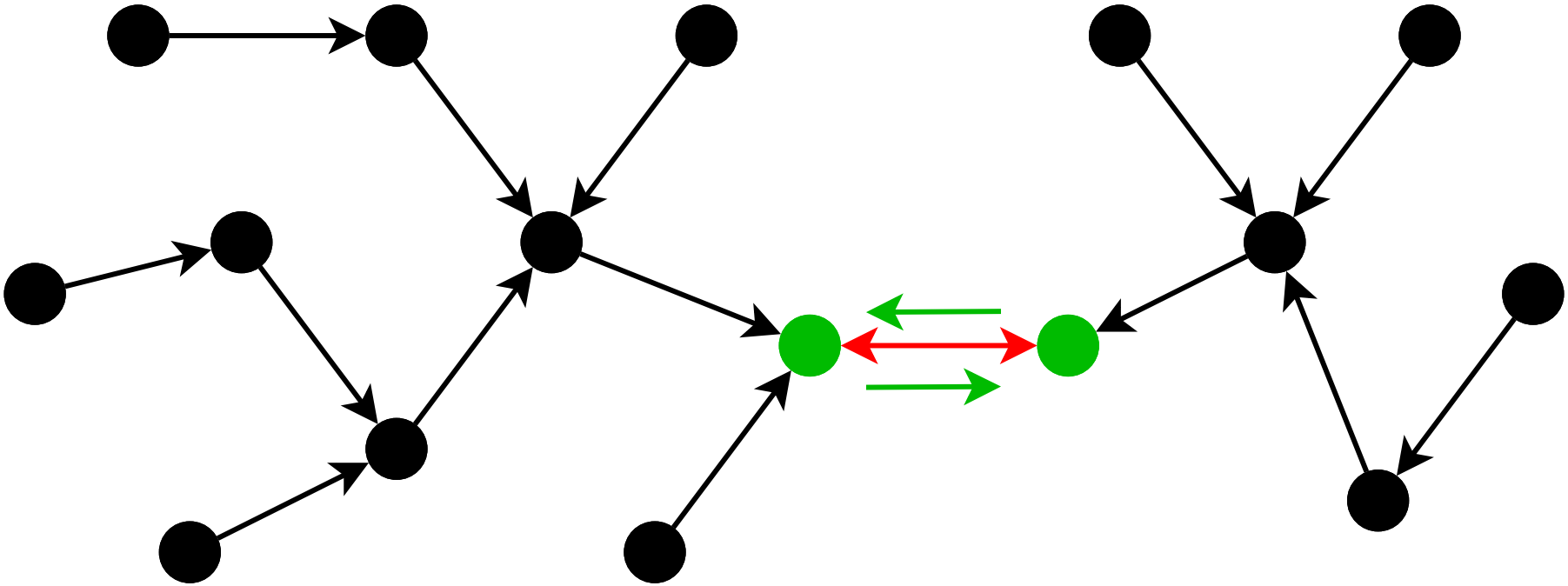
Finding the minimum weight outgoing edge



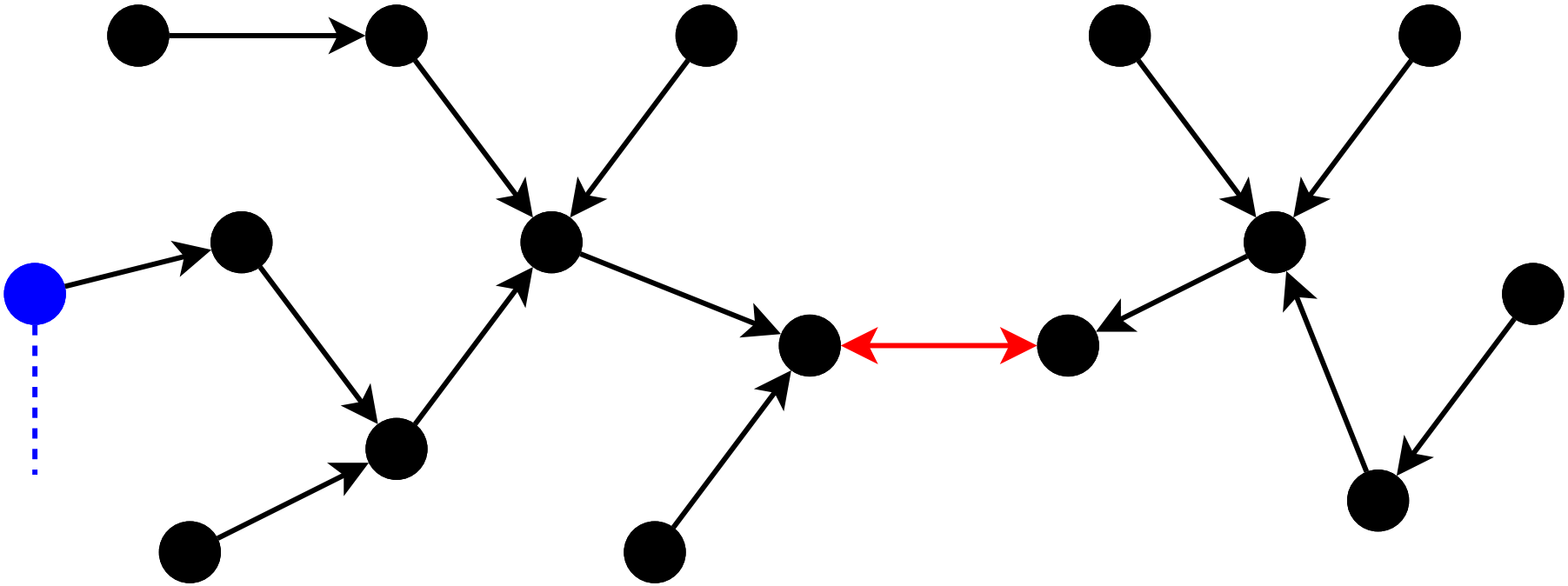
Finding the minimum weight outgoing edge



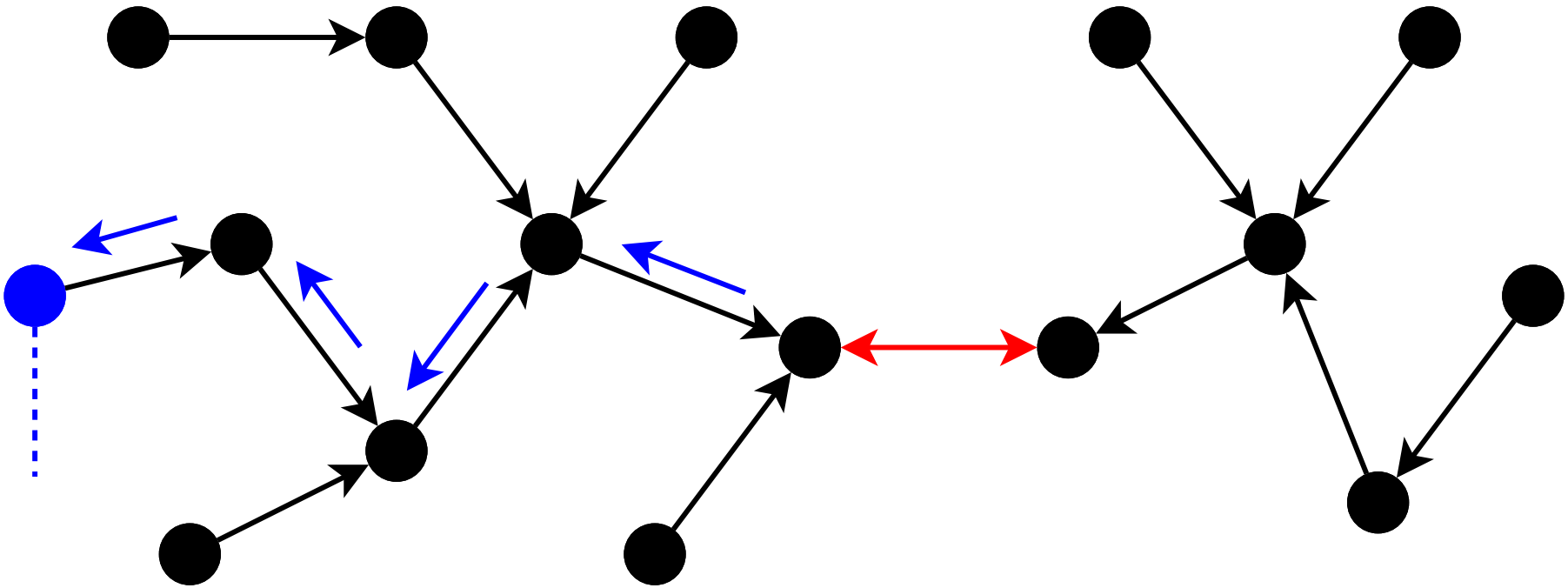
Finding the minimum weight outgoing edge



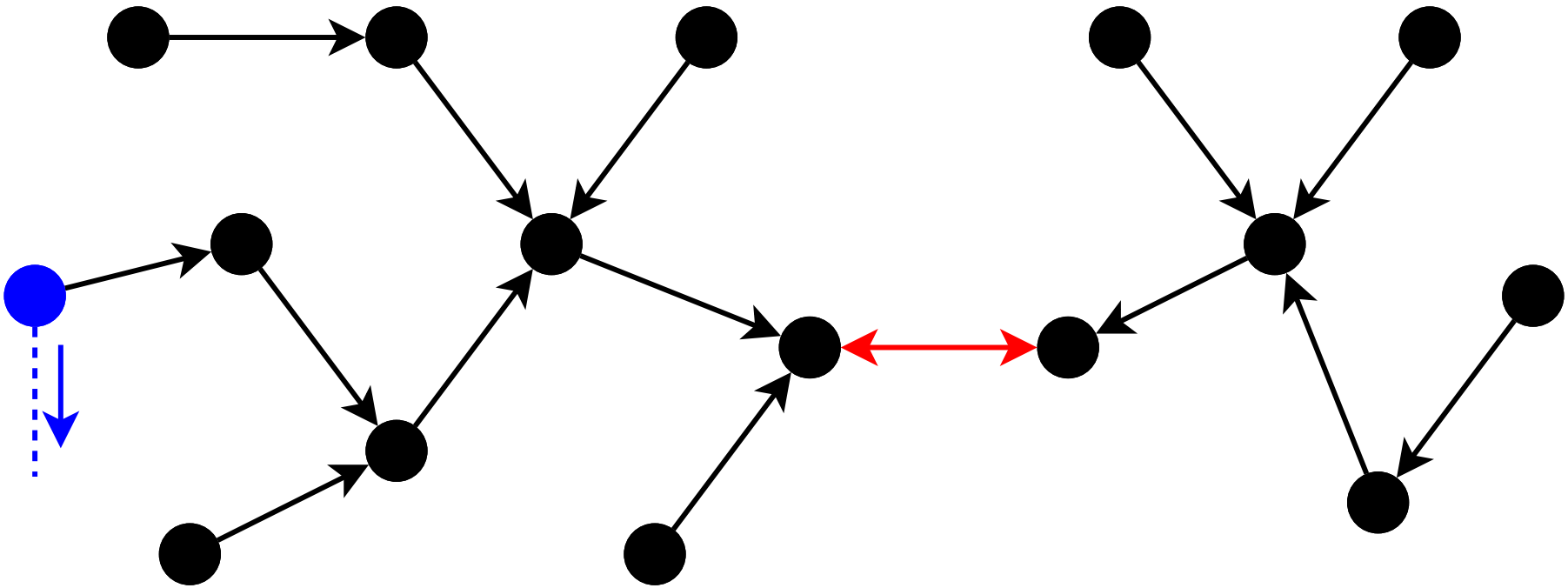
Finding the minimum weight outgoing edge



Finding the minimum weight outgoing edge



Finding the minimum weight outgoing edge



Finding minimum spanning tree

Order of combinations matters!

- Expanding one node at a time is slow.
- We want to at least double fragment size every time.

Rules to achieve this:

- Fragments with a single node are at *level* zero.
- Two fragments of level L may combine into a fragment of level $L + 1$.
- A lower level fragment may simply *join* another fragment.

$\Rightarrow$ A fragment of level L has at least 2^L nodes.

Complexity and conclusion

At most $2E + 5N \log_2 N$ messages needed.

- 2 messages for each edge rejection
- 5 other messages / node & level
- N nodes, at most $\log_2 N$ levels

The time complexity is $\mathcal{O}(N \log N)$.

- See proof in the final report.

An implementation (using HTML5 canvas) is available at
`http://www.cs.helsinki.fi/u/jwkangas/dmst/dmst.html`