Connected Components in MapReduce and Beyond

R. Kiveris, S. Lattanzi, V. Mirrokni, V. Rastogi, S. Vassilvitskii

Google

Modern Massive Algorithmics

Communication:

- Can be the overwhelming cost
- In practice constant factors matter a whole lot

Data Skew:

- Most datasets are heavy tailed
- Naive data distribution can be disastrous
- In synchronous environments must wait for slowest shard
  - “Curse of the last reducer”

Modern Massive Algorithmics

Communication:

- Can be the overwhelming cost
- In practice constant factors matter a whole lot

Data Skew:

- Most datasets are heavy tailed
- Naive data distribution can be disastrous
- In synchronous environments must wait for slowest shard
  - “Curse of the last reducer”

Algorithms:

- Embarrassingly parallel may also be embarrassingly slow
- New techniques to minimize communication & skew

Today: Graph Connectivity

Classical problem

- Many parallel algorithms
  - PRAM
  - MapReduce
  - Pregel
  - ...
- Subroutine in many other problems
  - MST
  - Clustering
  - Multiway cuts
  - ...

Today: Graph Connectivity

Classical problem

- Many parallel algorithms
  - PRAM
  - MapReduce
  - Pregel
  - ...
- Subroutine in many other problems
  - MST
  - Clustering
  - Multiway cuts
  - ...

Want to optimize for very large graphs

- Billions of nodes, 100s of billions of edges
- Typically sparse
- Do not fit in memory (10s+ TBs)

Approach

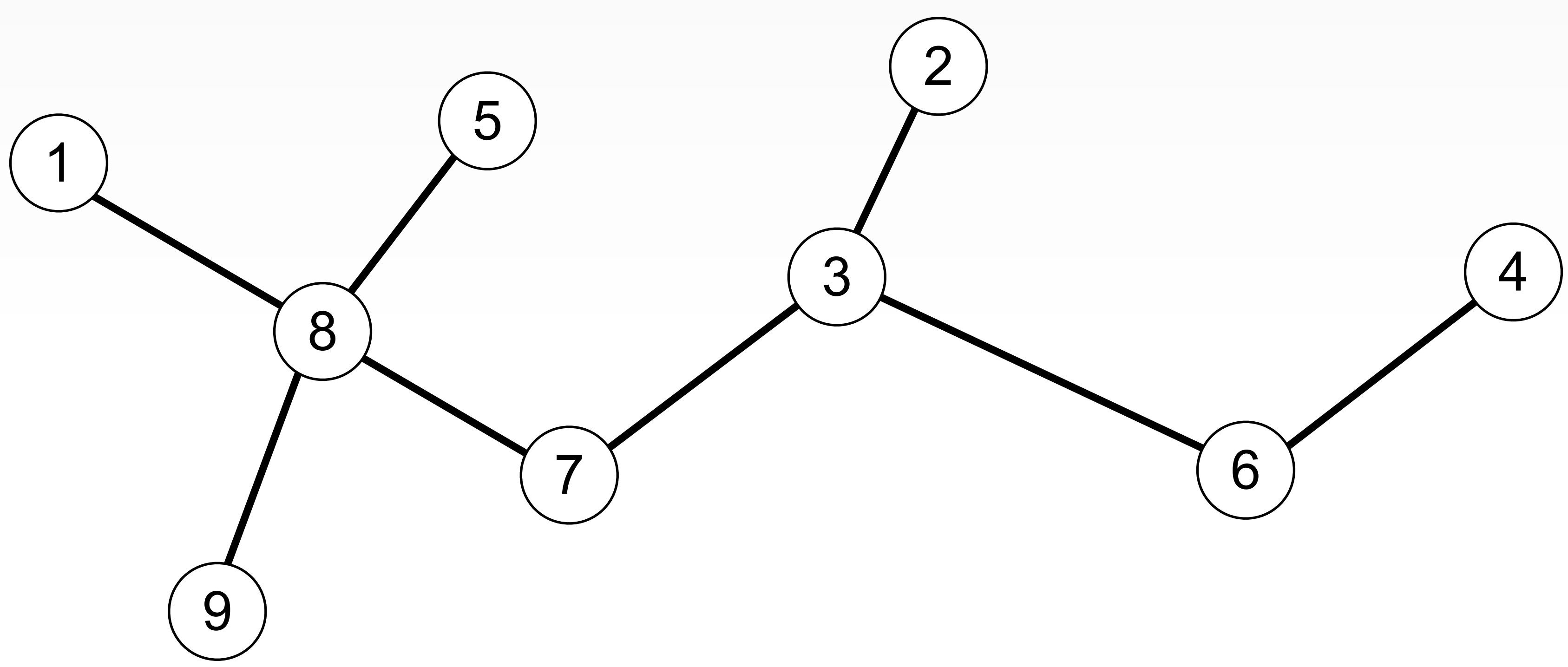
Transform the graph into a union of stars, one for each connected component.

Approach

Transform the graph into a union of stars, one for each connected component.

Begin:

- Every node has a unique id
- Assigned arbitrarily



Approach

Transform the graph into a union of stars, one for each connected component.

Begin:

- Every node has a unique id
- Assigned arbitrarily

Two Local Operations:

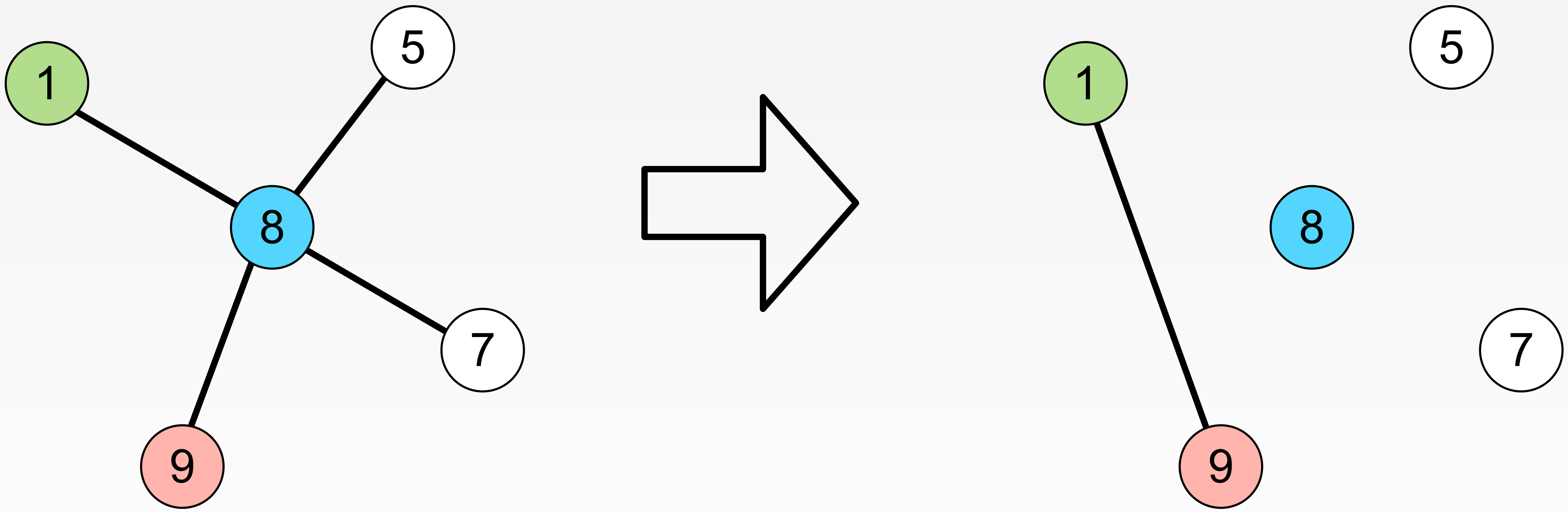
- Only look at a node and its neighbors
- Prescribe which edges should exist in the next round

Operations

- `LargeStar(v)`: Connect all strictly larger neighbors to the min neighbor including self.

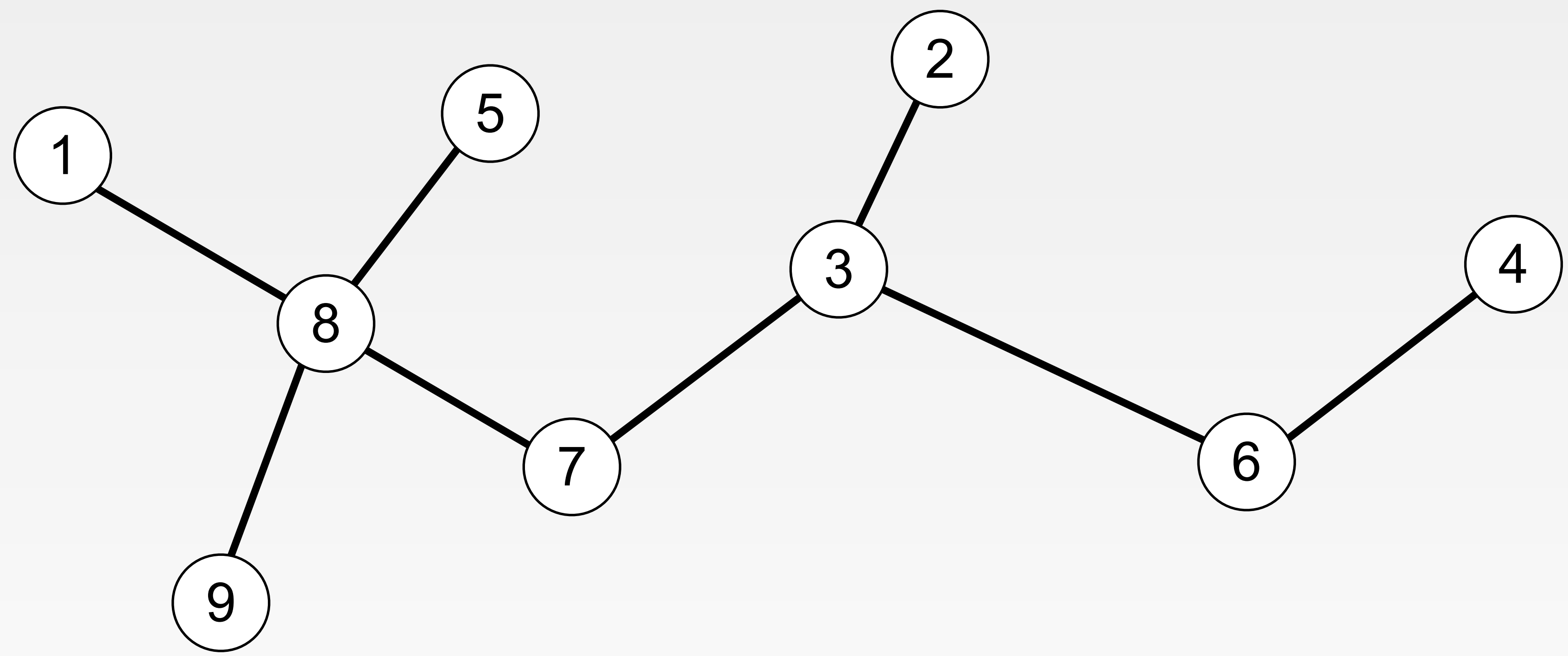
Operations

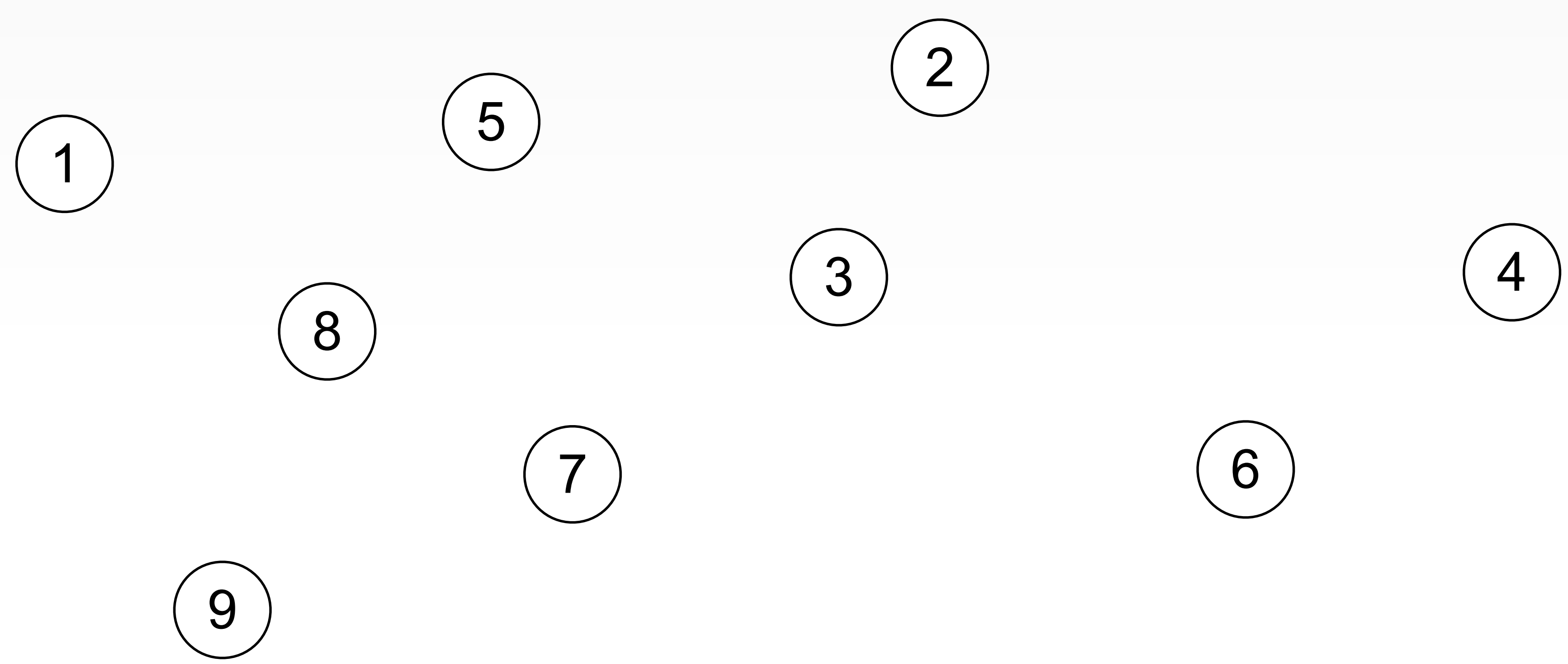
- `LargeStar(v)`: Connect all strictly larger neighbors to the min neighbor including self.



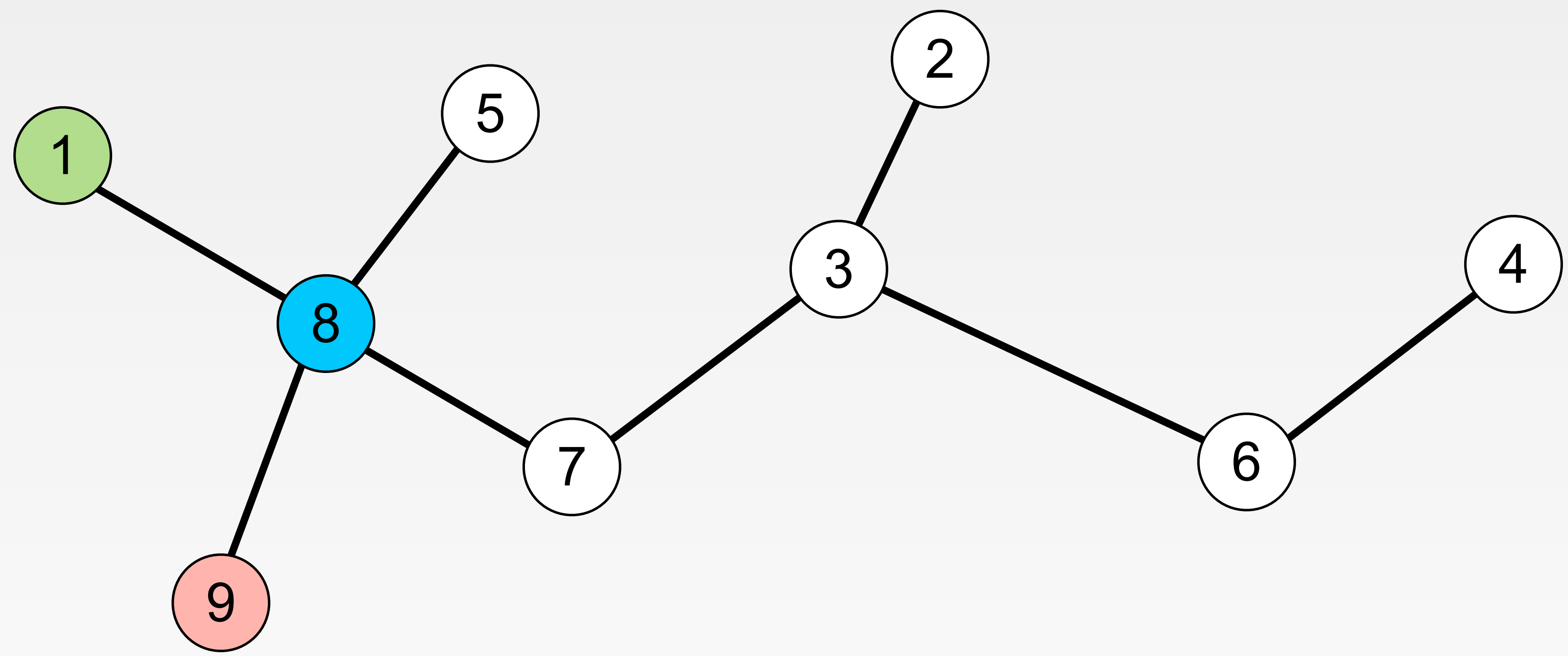
- Do this in parallel on every node to build a new graph

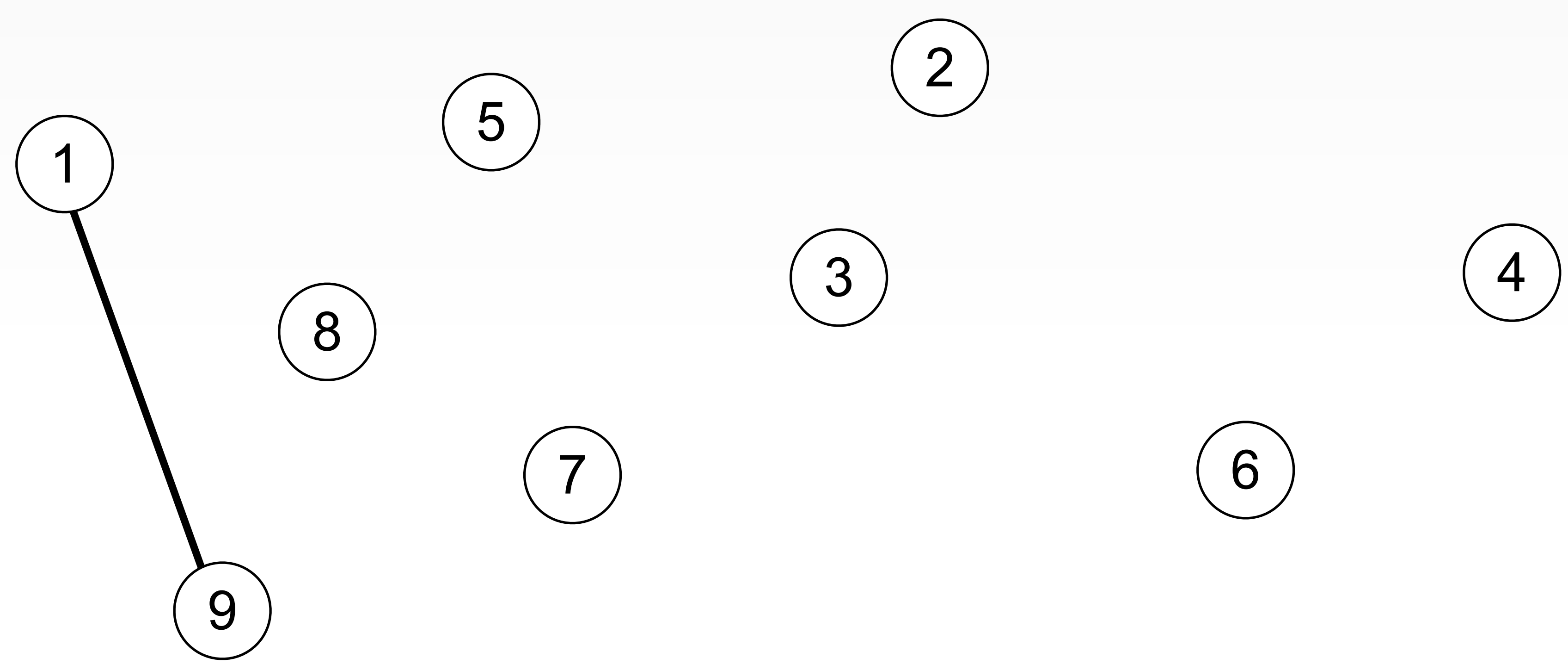
Example



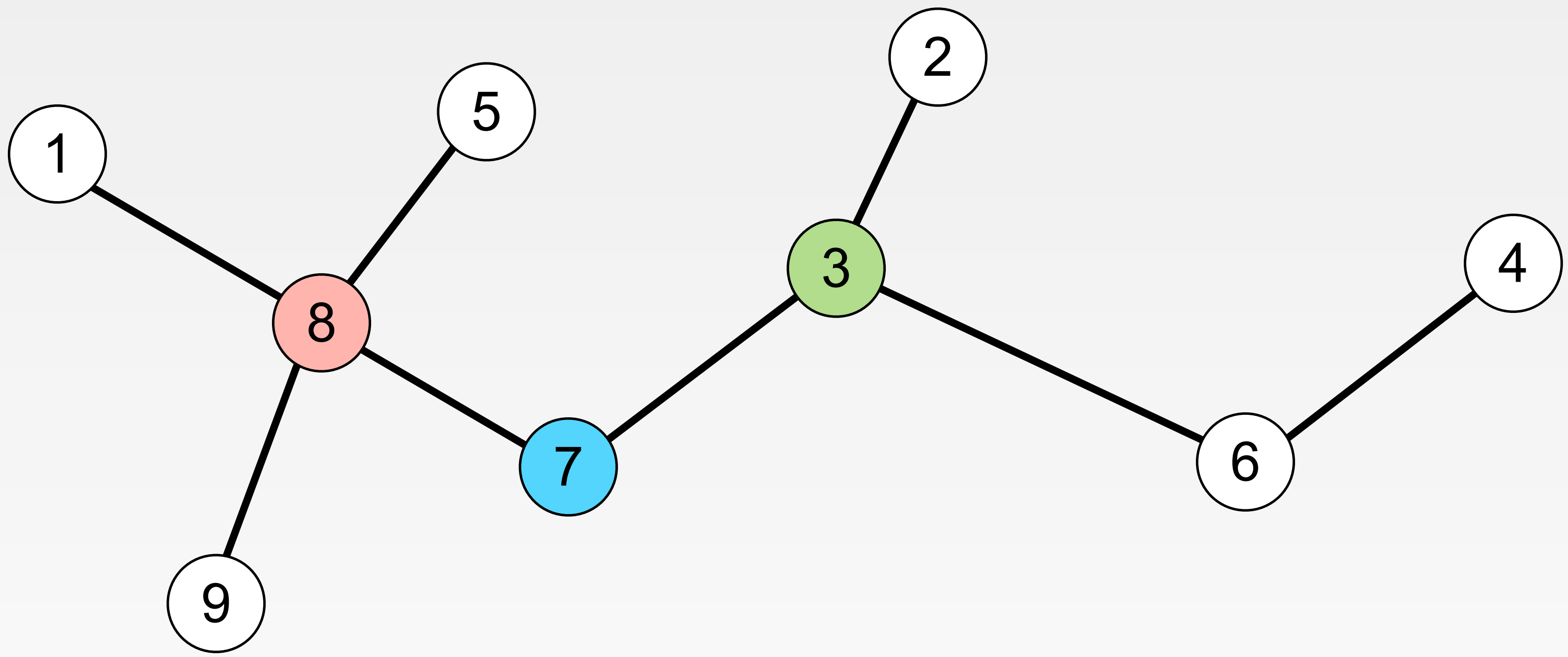


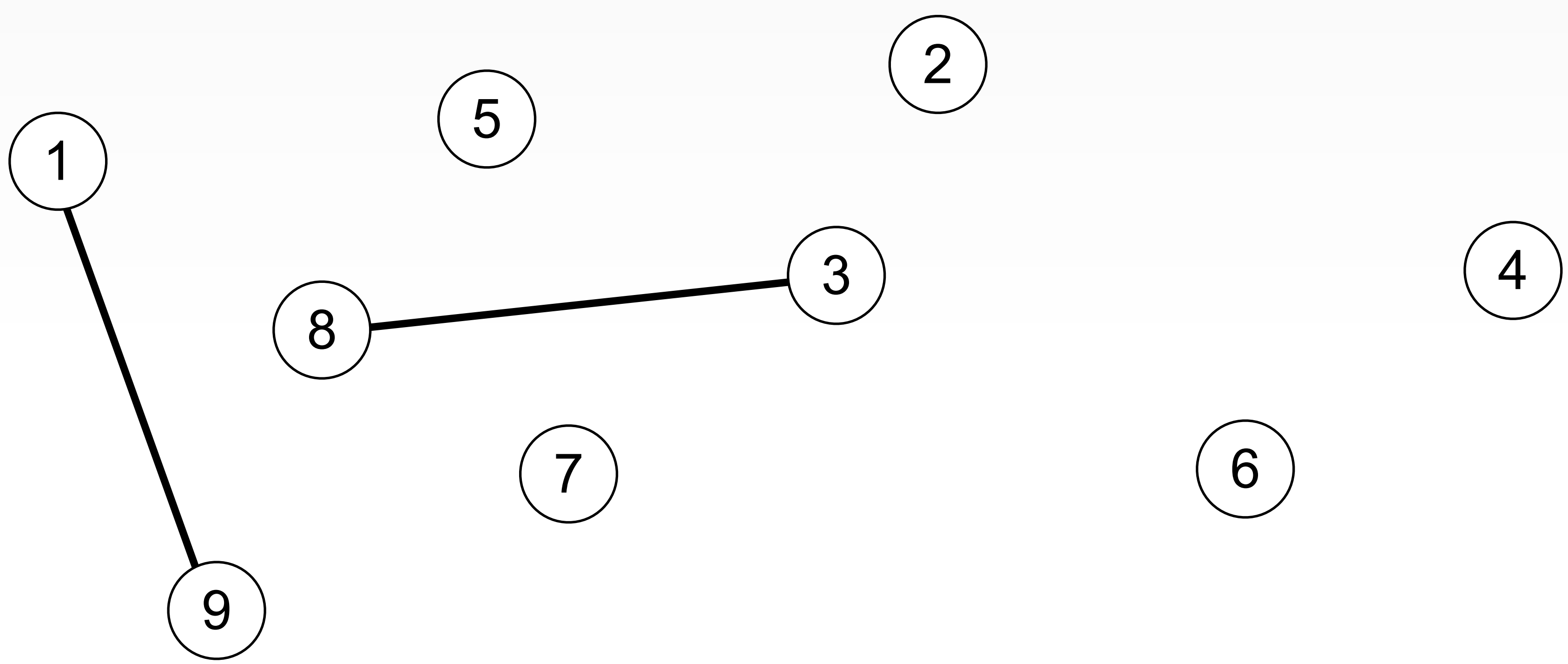
Example



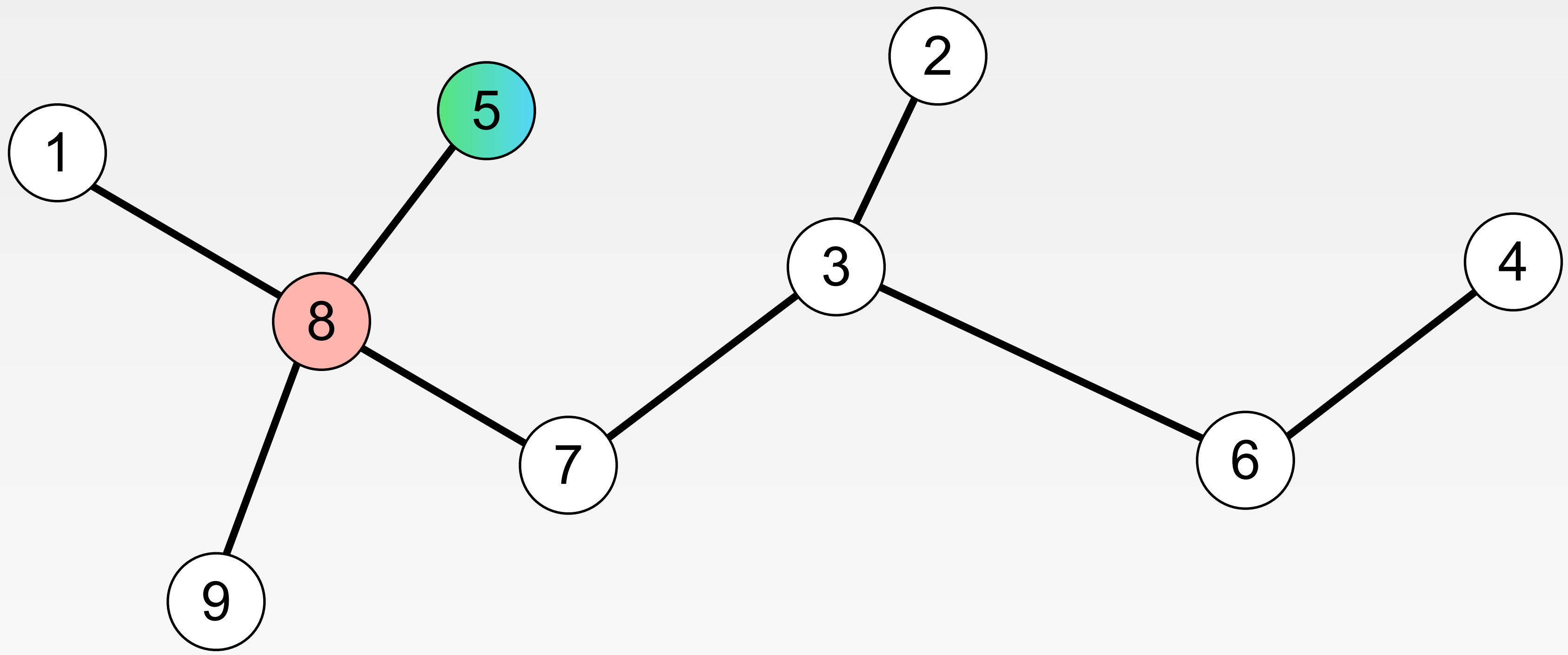


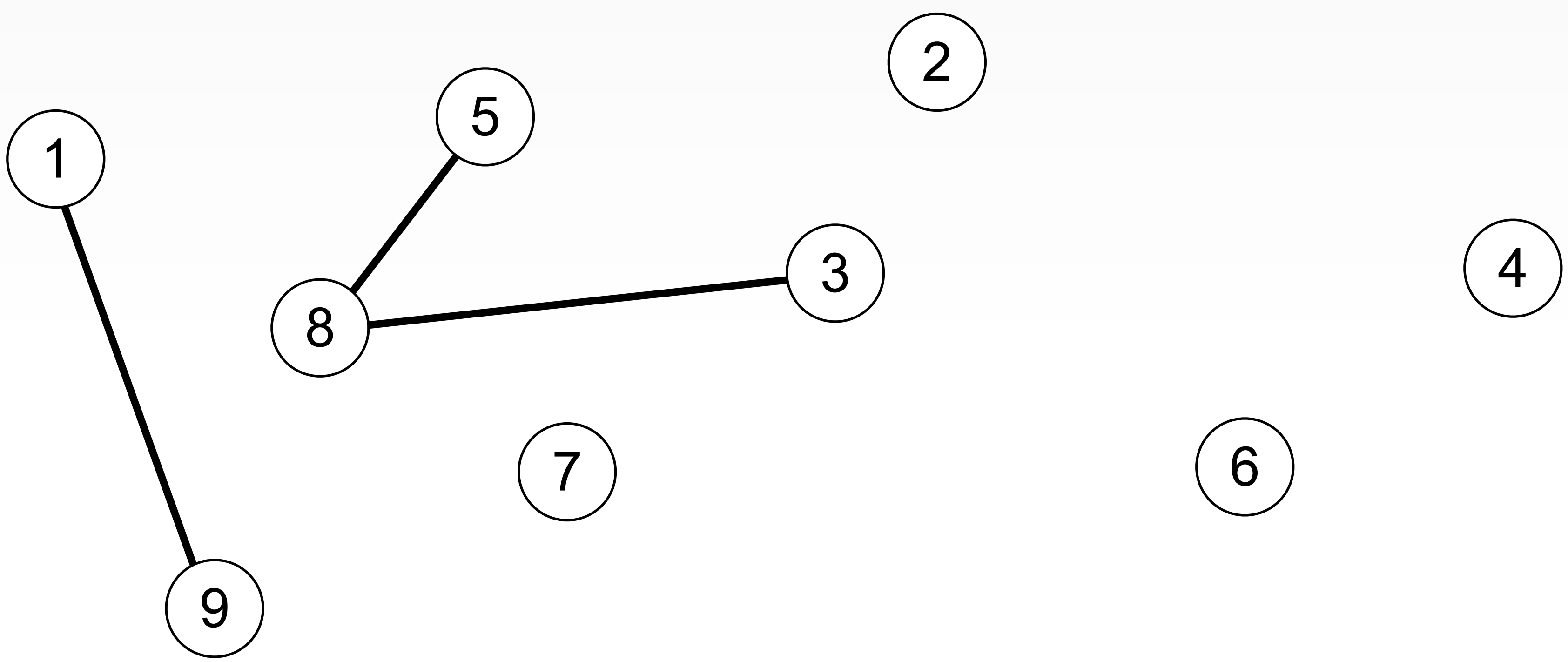
Example



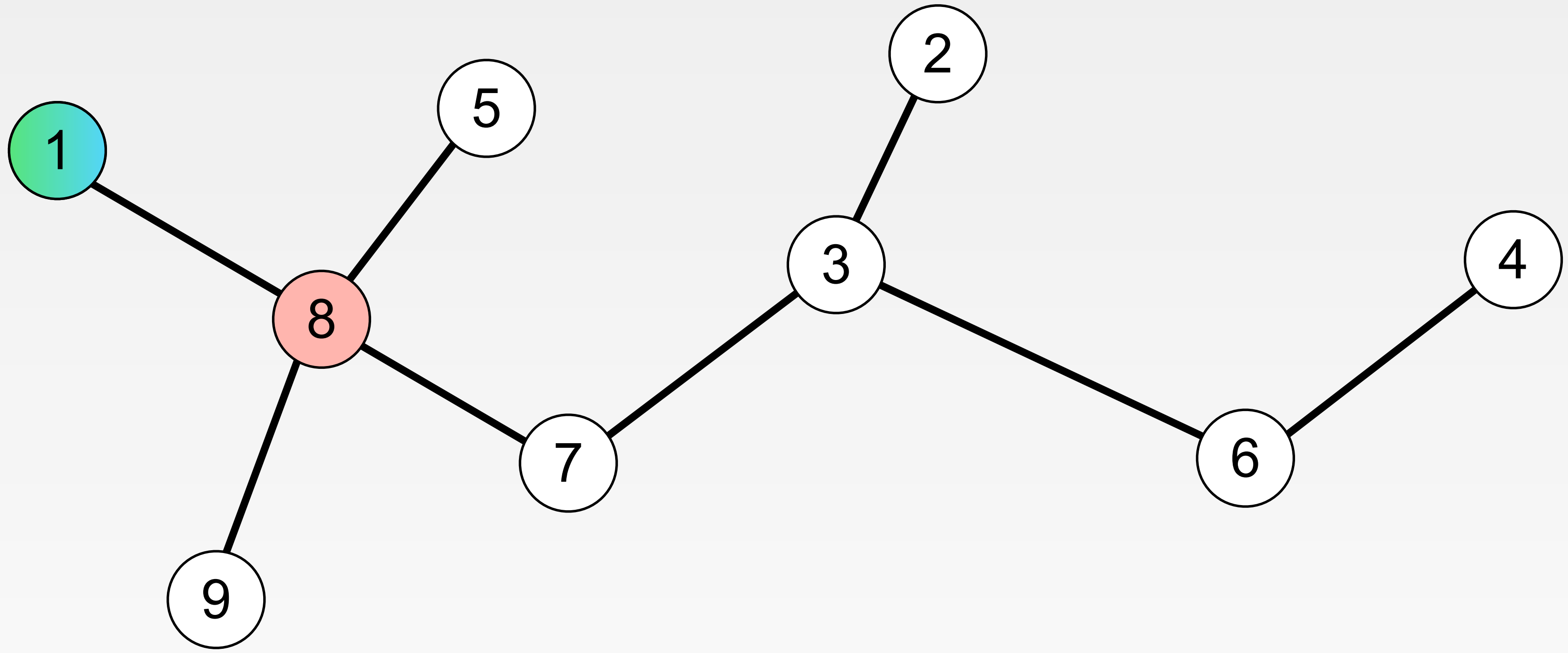


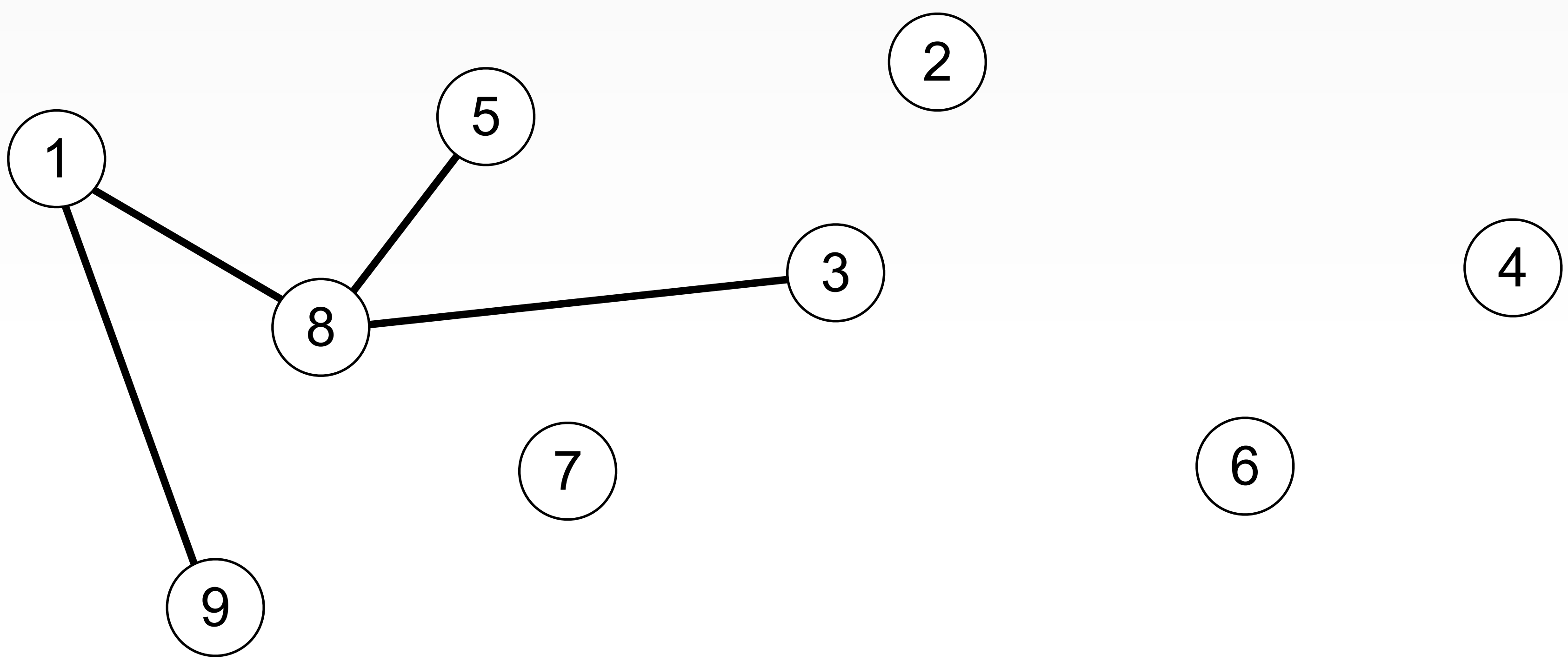
Example



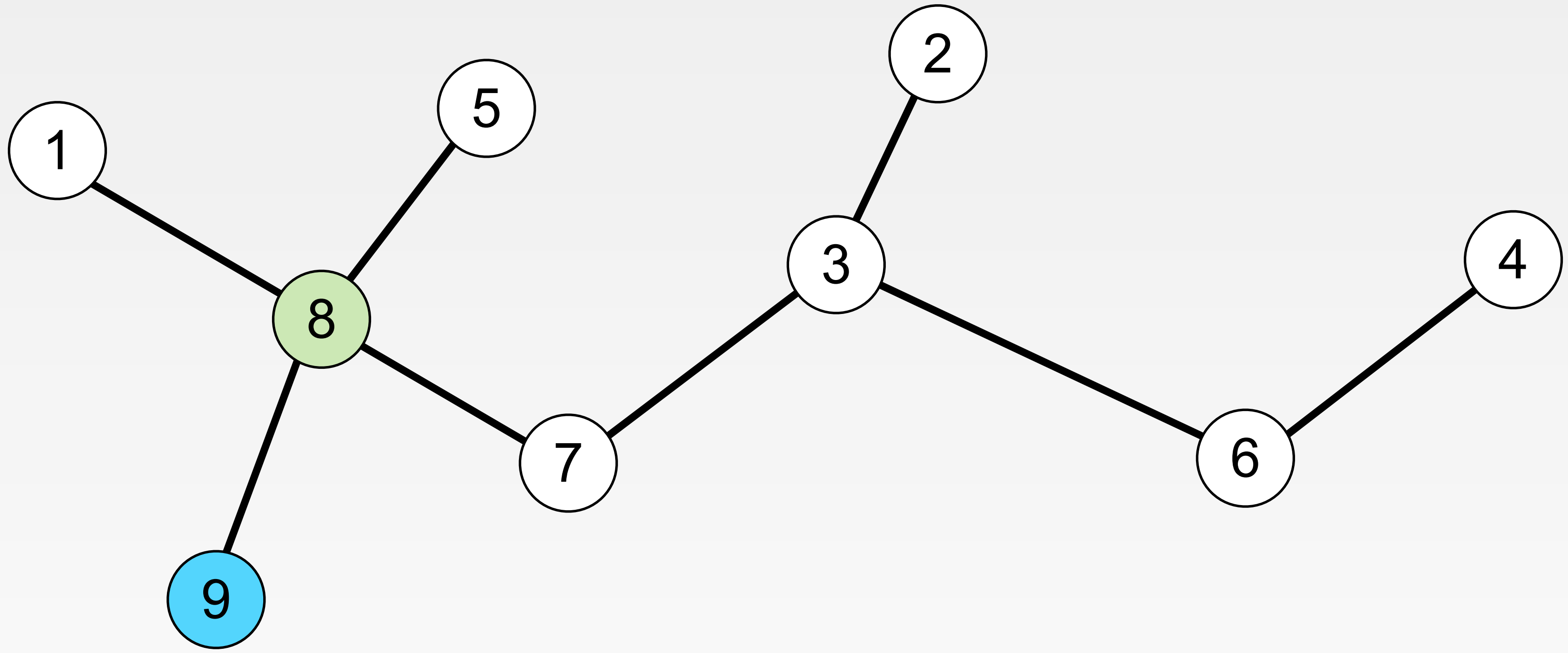


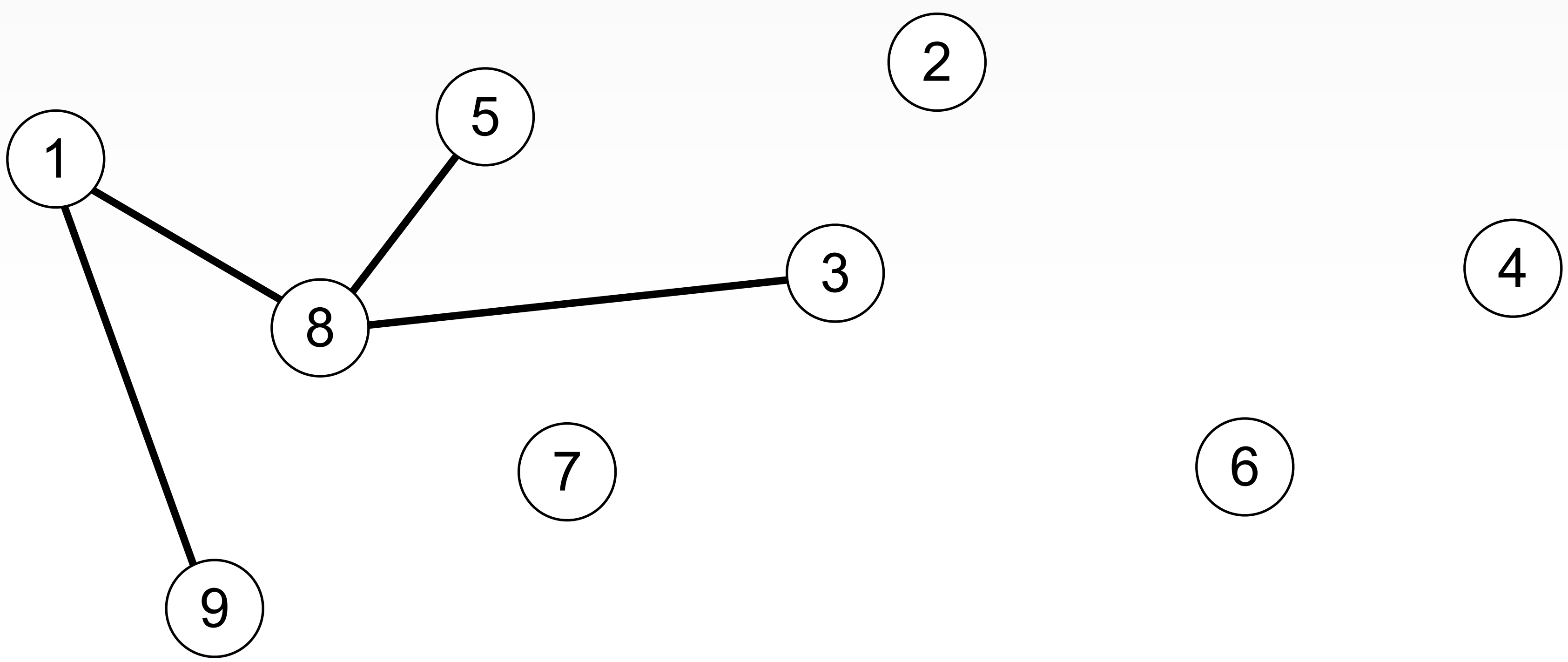
Example



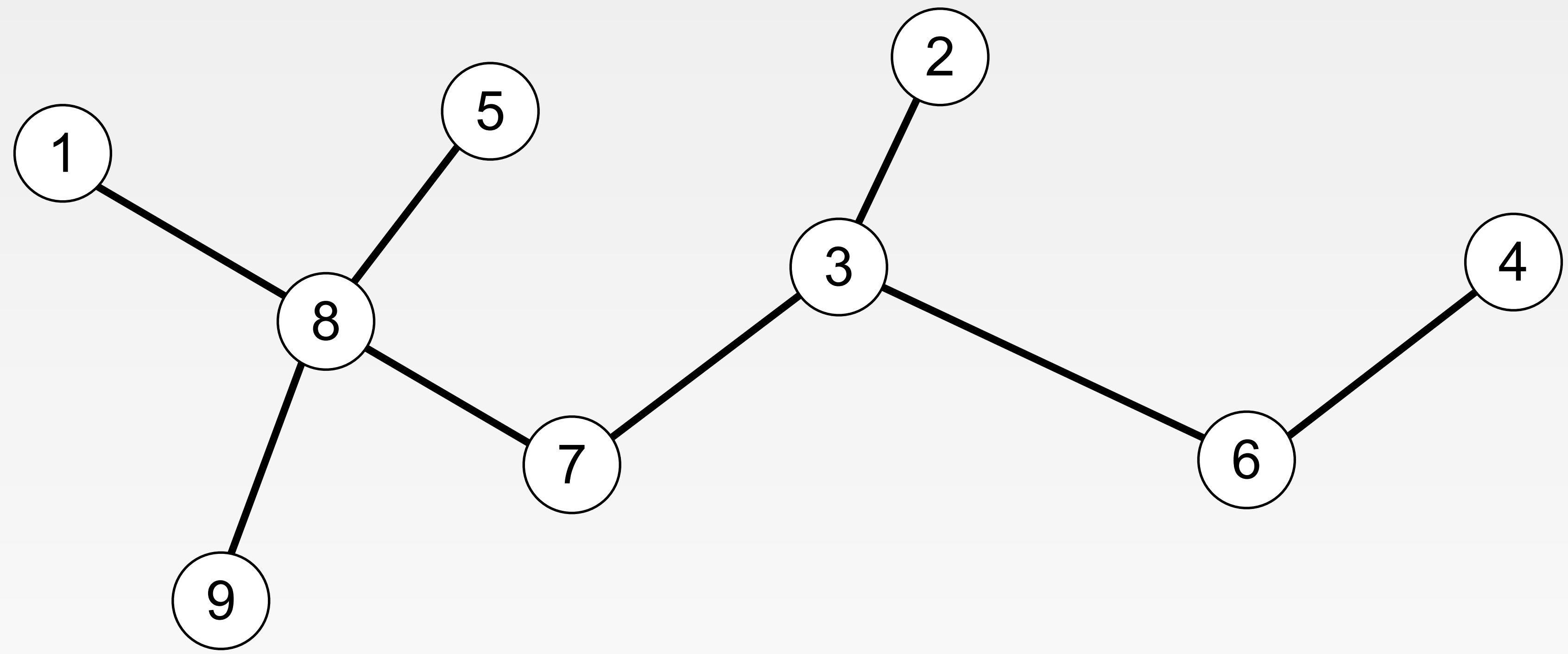


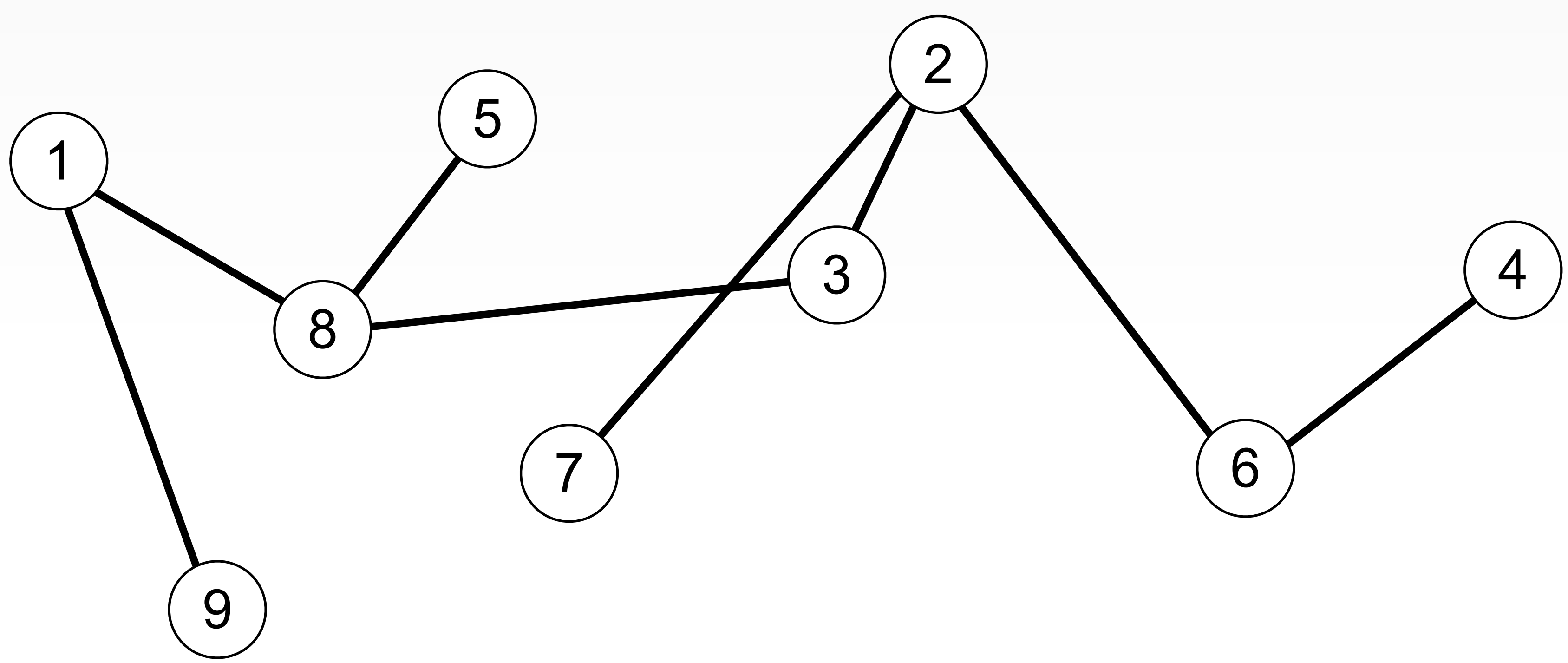
Example





Example

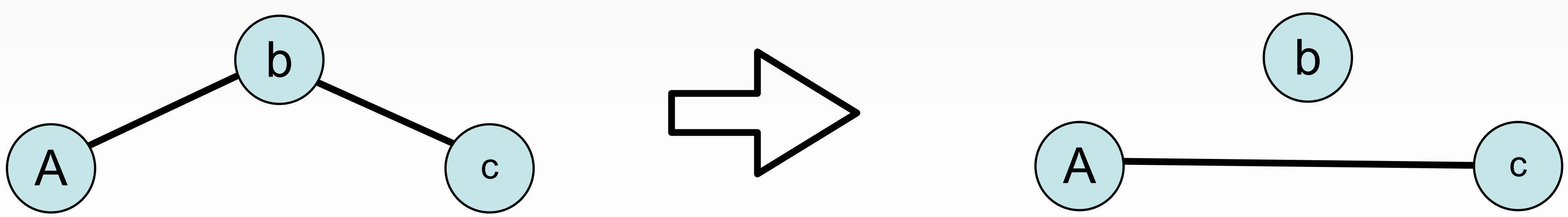




LargeStar Connectivity

Lemma: Executing **`LargeStar`** in parallel preserves connectivity

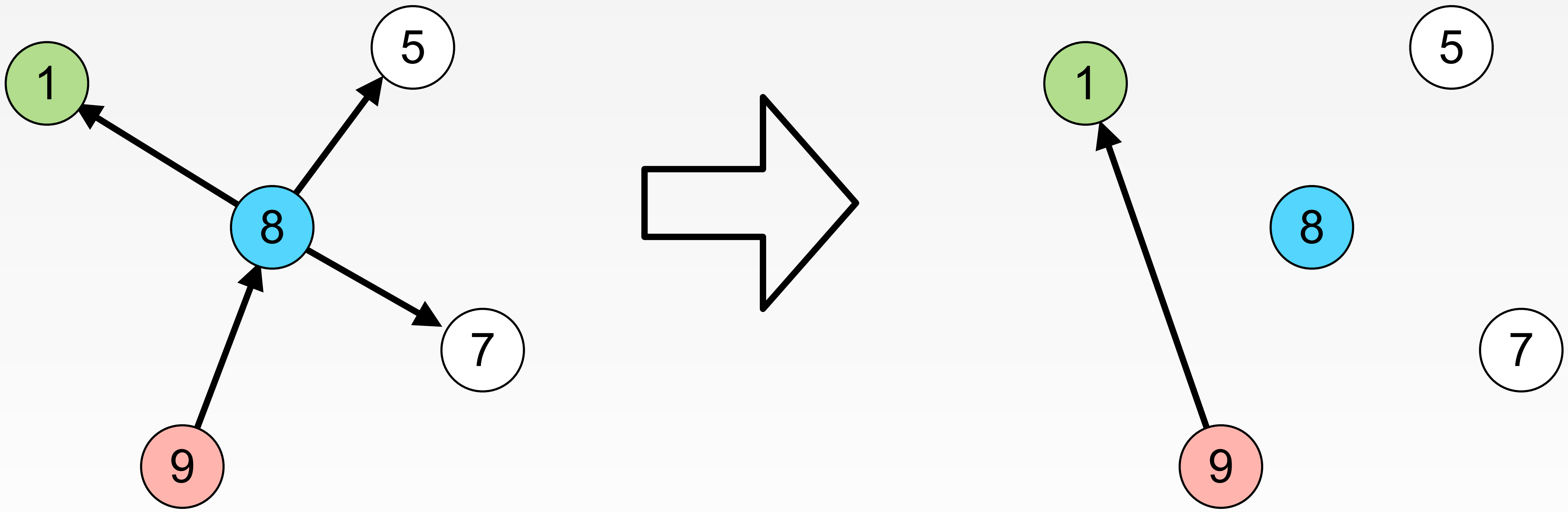
- WLOG assume A > b. If b is min neighbor of A we are done
- If b has no smaller neighbors (local min) we are done
- Else: A > b > c and:



  - Now need to reason about connectivity of (b,c)
  - Show (b,c) connected by induction on node rank

LargeStar: Reinterpretation

- **`LargeStar(v)`**: Connect all strictly larger neighbors to the min neighbor including self.

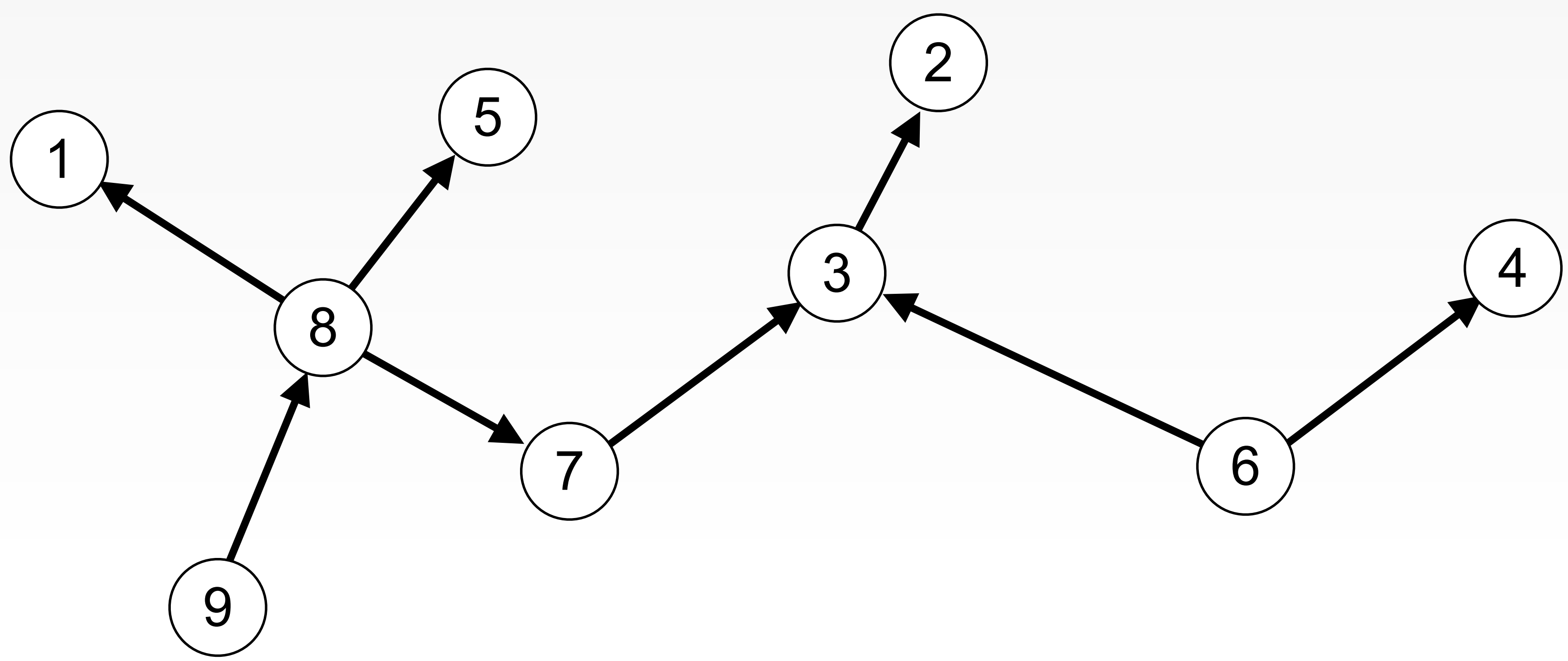


- Orient all edges from larger to smaller
- **`LargeStar`** = tell children to connect to smallest parent

LargeStar Fixed Point

Fixed point if:

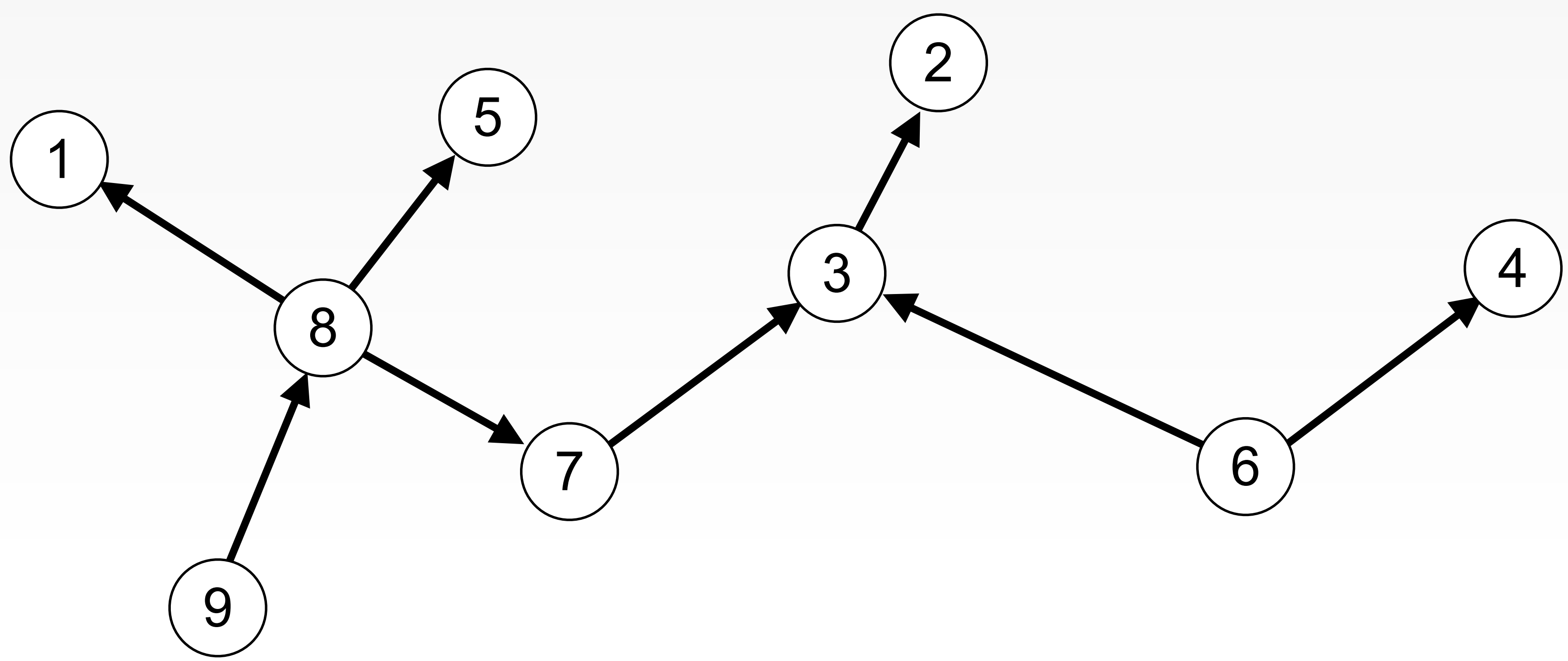
- Every node is a local min or connected to local minima
- Orient edges from larger nodes to smaller nodes
  - Fixed point if graph is DAG of height 2



LargeStar Fixed Point

Fixed point if:

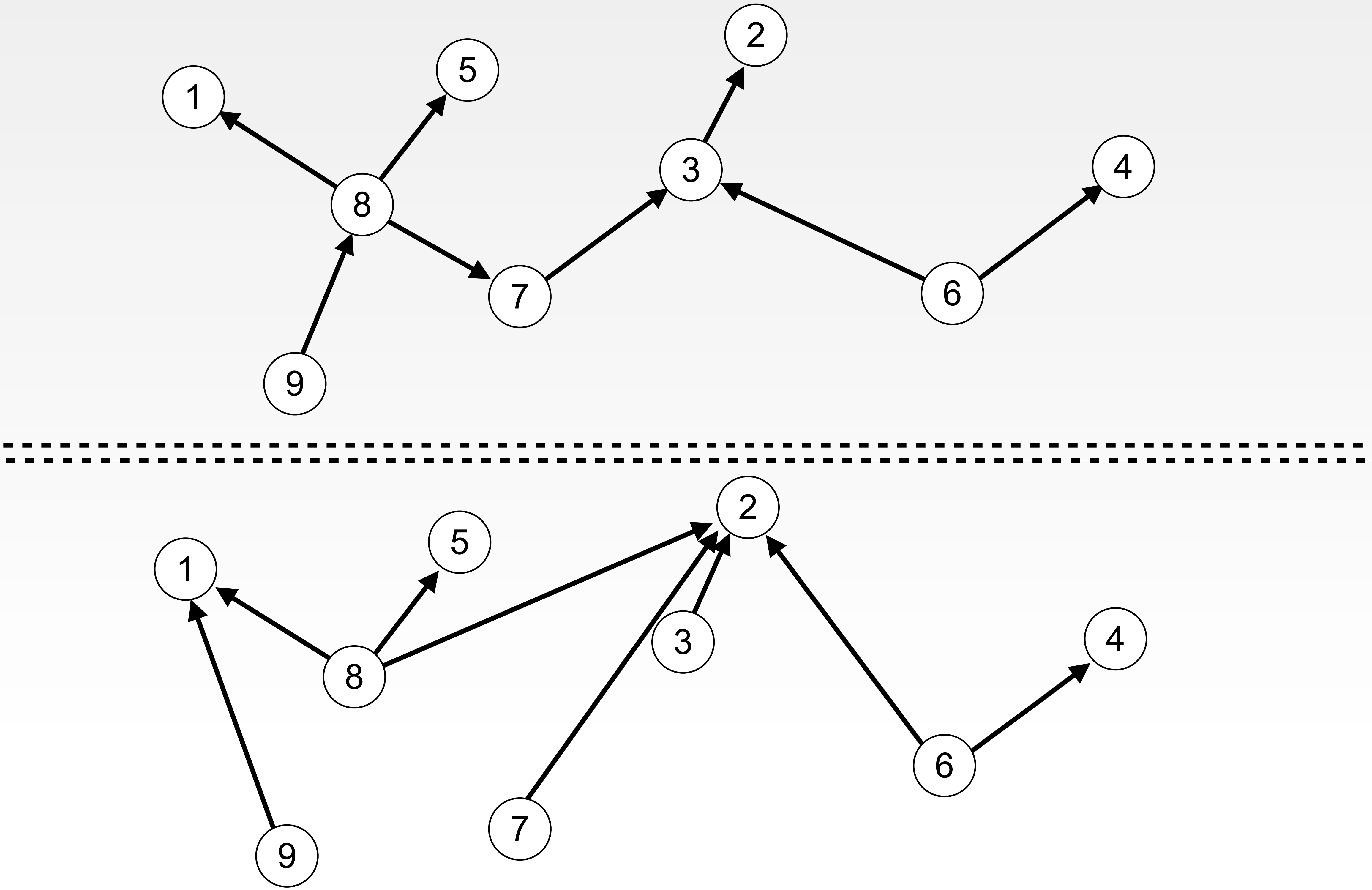
- Every node is a local min or connected to local minima
- Orient edges from larger nodes to smaller nodes
  - Fixed point if graph is DAG of height 2



Progress:

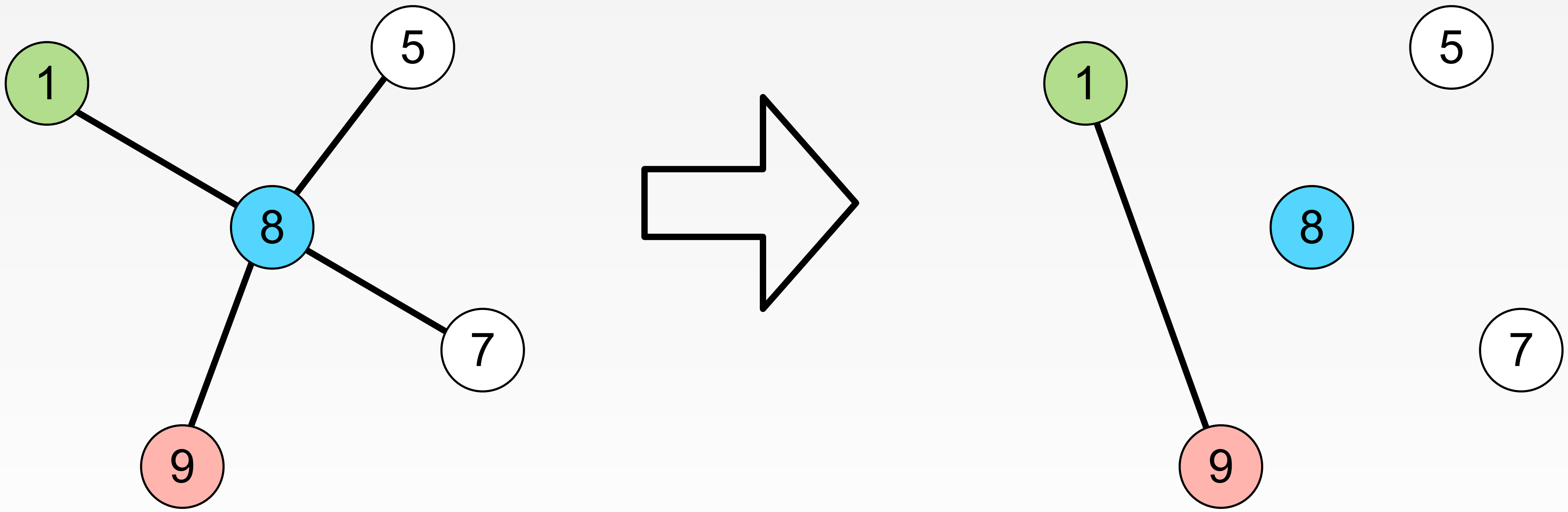
- Every **`LargeStar`** operation reduces height by a constant factor

LargeStar Fixed Point



Operations

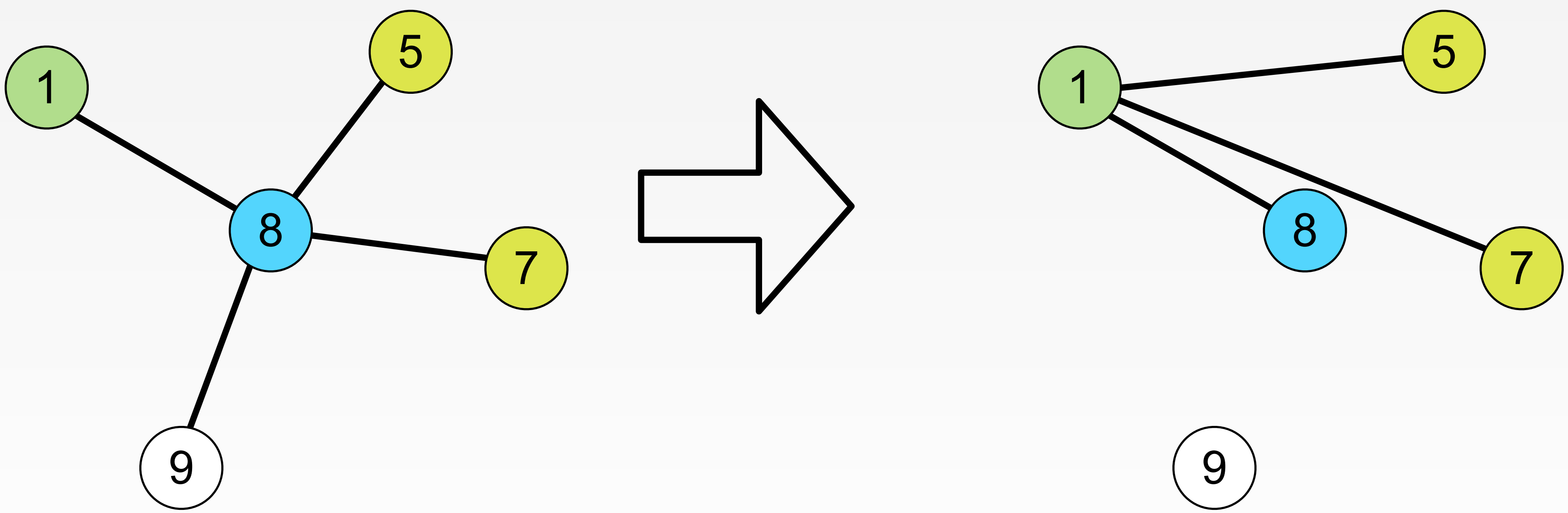
- **`LargeStar(v)`**: Connect all strictly larger neighbors to the min neighbor including self.



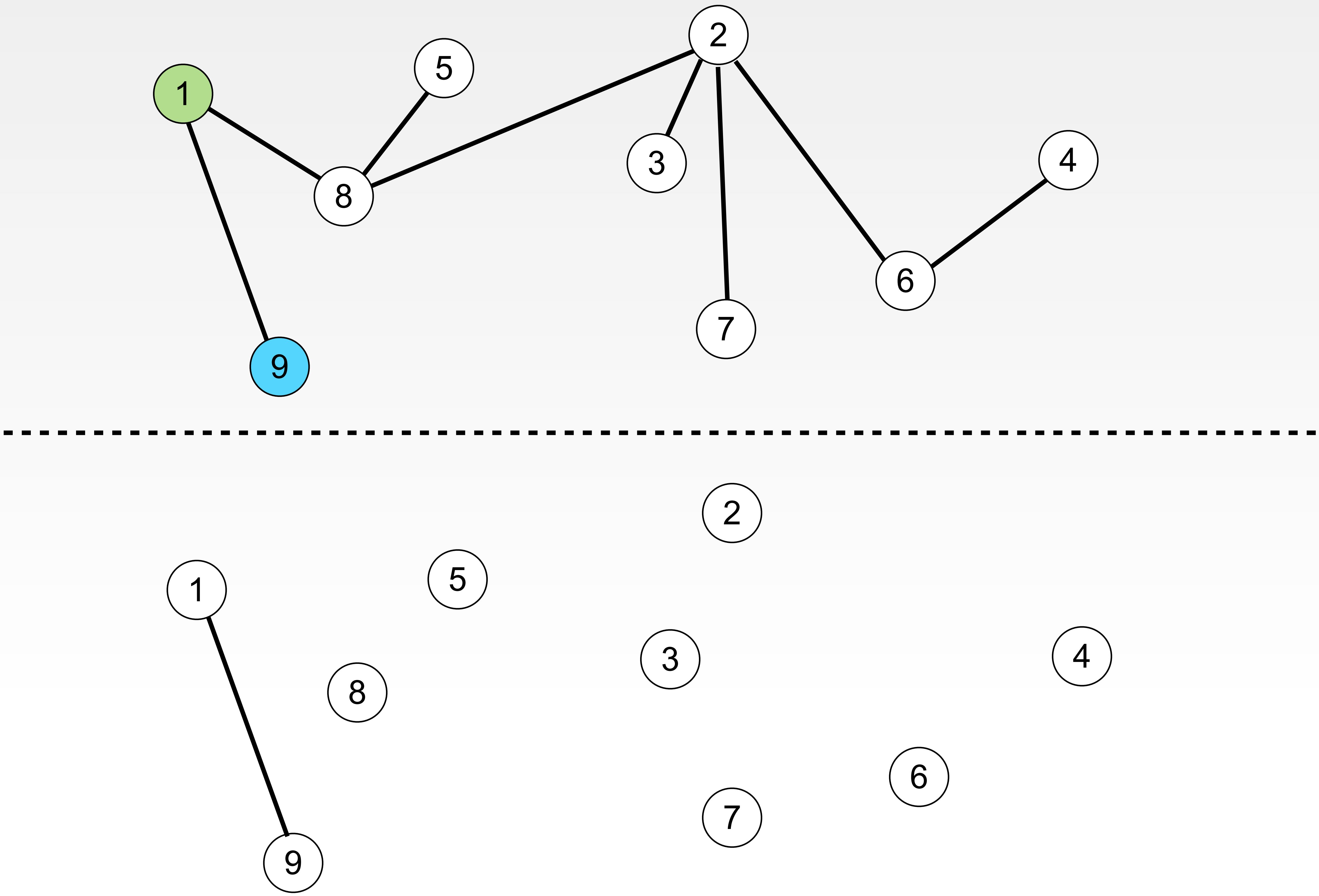
- Do this in parallel on every node to build a new graph

Operations

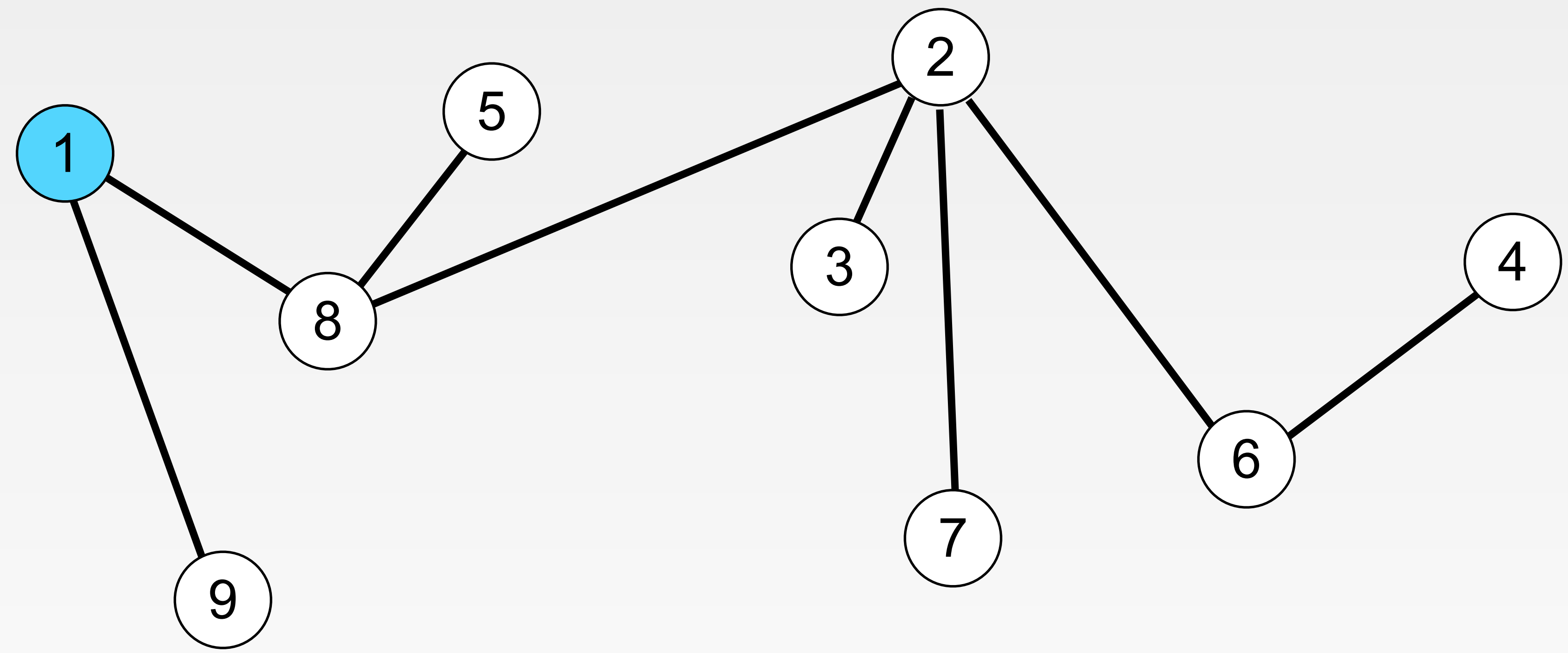
- `SmallStar(v)`: Connect all smaller neighbors and self to the min neighbor.

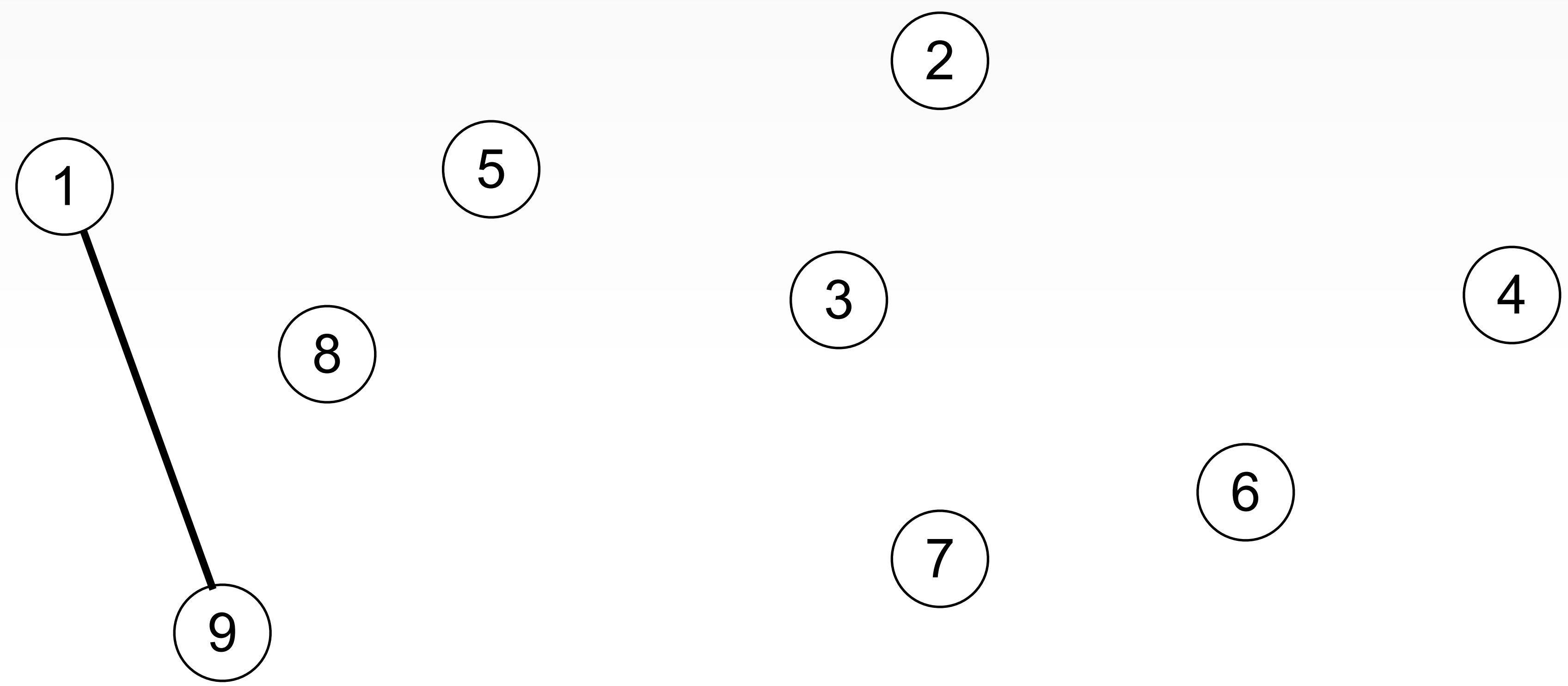


Example

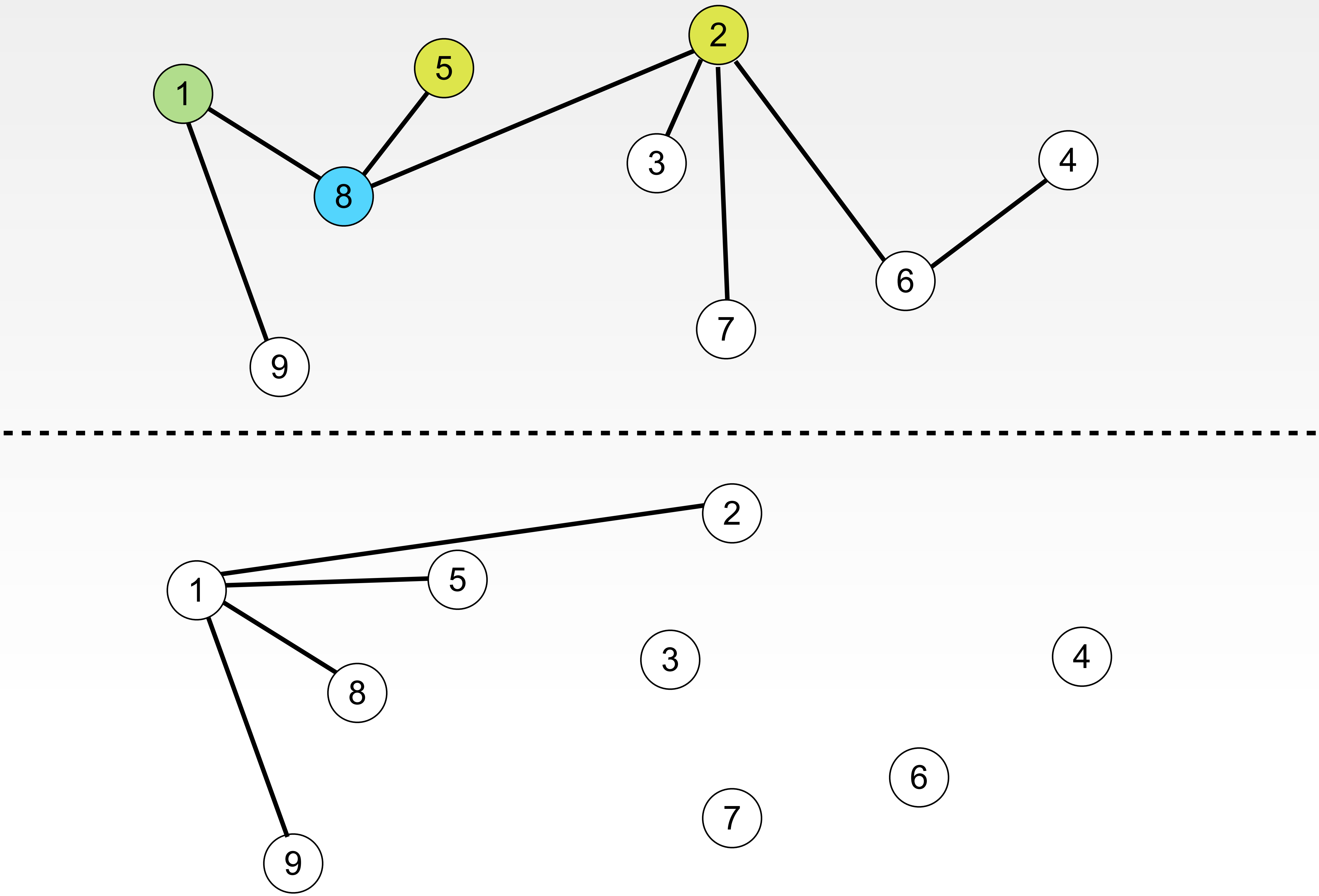


Example

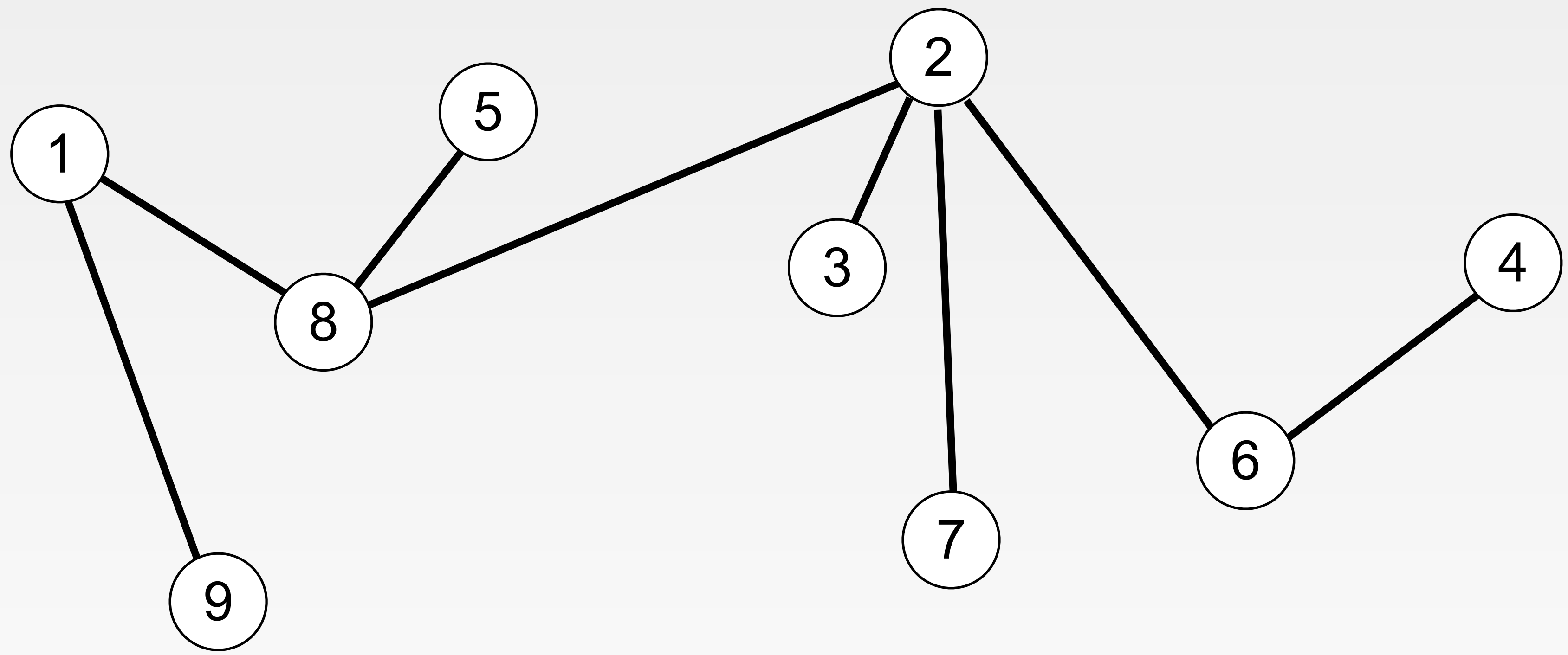


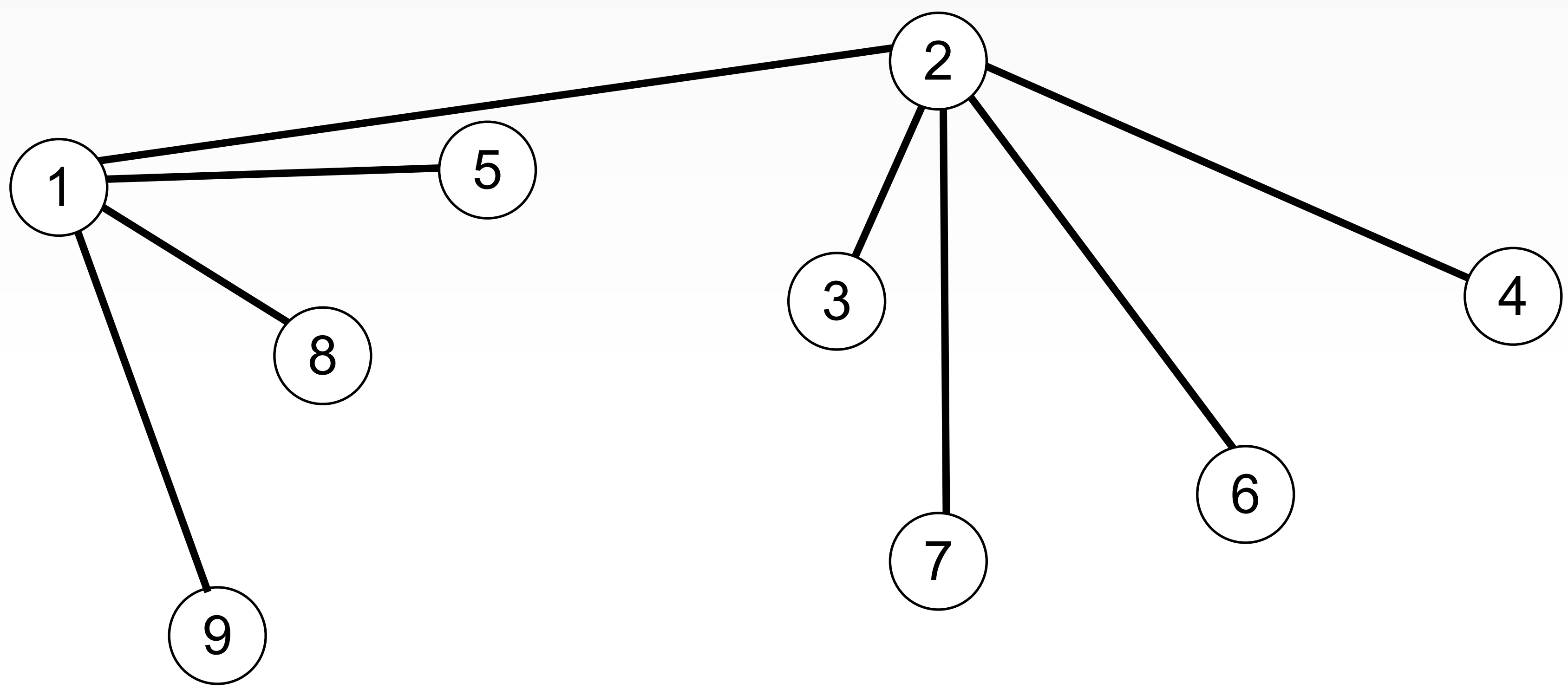


Example



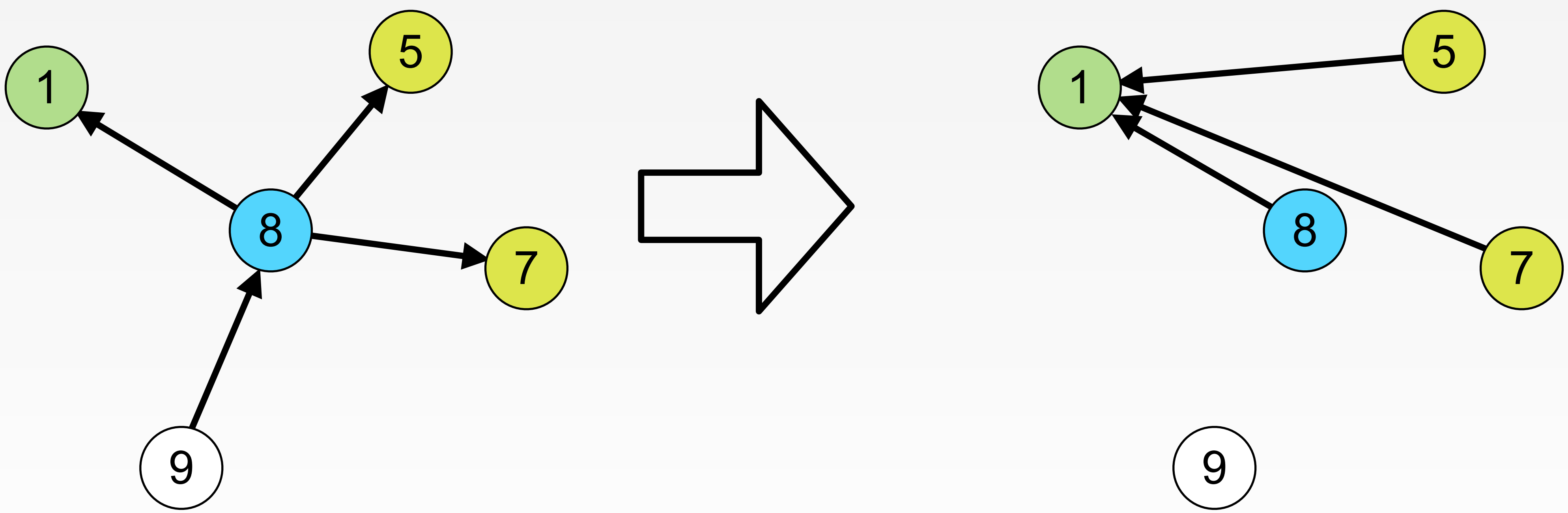
Example





Operations

- **`SmallStar(v)`**: Connect all smaller neighbors and self to the min neighbor.



- Connect all parents (and self) to the minimum parent.

SmallStar Analysis

Lemma: **`SmallStar`** preserves connectivity

- Similar argument as before

SmallStar Analysis

Lemma: **`SmallStar`** preserves connectivity

- Similar argument as before

Progress:

- Run **`LargeStar`** to completion
- Run one iteration of **`SmallStar`**
- Run **`LargeStar`** to completion again
- The number of local minima (maximal nodes in the DAG) reduces by a constant factor

Overall Algorithm

Input:

- Set of edges, with a unique label per node

```
Repeat until convergence
    Repeat until convergence
        LargeStar
    SmallStar
```

Theorem:

- The above algorithm converges in $O(\log^2 n)$ rounds.

Implementation

Implementation

Both can be easily implemented in MapReduce

- Or Pregel, or Giraph, or ...

`LargeStar`:

`Map` `(u;v):`

- `Emit (u;v), Emit (v;u)`

`Reduce` `(u;` $v_1, v_2, \ldots, v_k$`):`

- `m = argmin label(`v_i`)`
- `Emit (v,m) for all v with label(v) > label(m)`

Discussion

Convergence:

- $\log^2 n$: is tight
- The graph is used to define communication structure from time to time
- Number of edges **does not increase** at every time step

Making it Practical

Making it Practical

Approach 1 (Systems):

- **`LargeStar`** is equivalent to finding one of the maxima in the DAG reachable from each vertex
- Can do this with a fast distributed hash table (DHT) to "walk up to the root"
  - Keep the min id of the parent in a DHT
  - Similar to path compression

Making it Practical

Approach 1 (Systems):

- **`LargeStar`** is equivalent to finding one of the maxima in the DAG reachable from each vertex
- Can do this with a fast distributed hash table (DHT) to "walk up to the root"
  - Keep the min id of the parent in a DHT
  - Similar to path compression

Approach 2 (Algorithms):

- Instead of waiting for **`LargeStar`** to converge, just interleave **`LargeStar`** and **`SmallStar`**.
  - **`Repeat`** `Until Convergence:`

```
SmallStar
LargeStar
```

- Can prove convergence
- Appears to converge even faster (conjecture $O(\log(n))$ rounds)

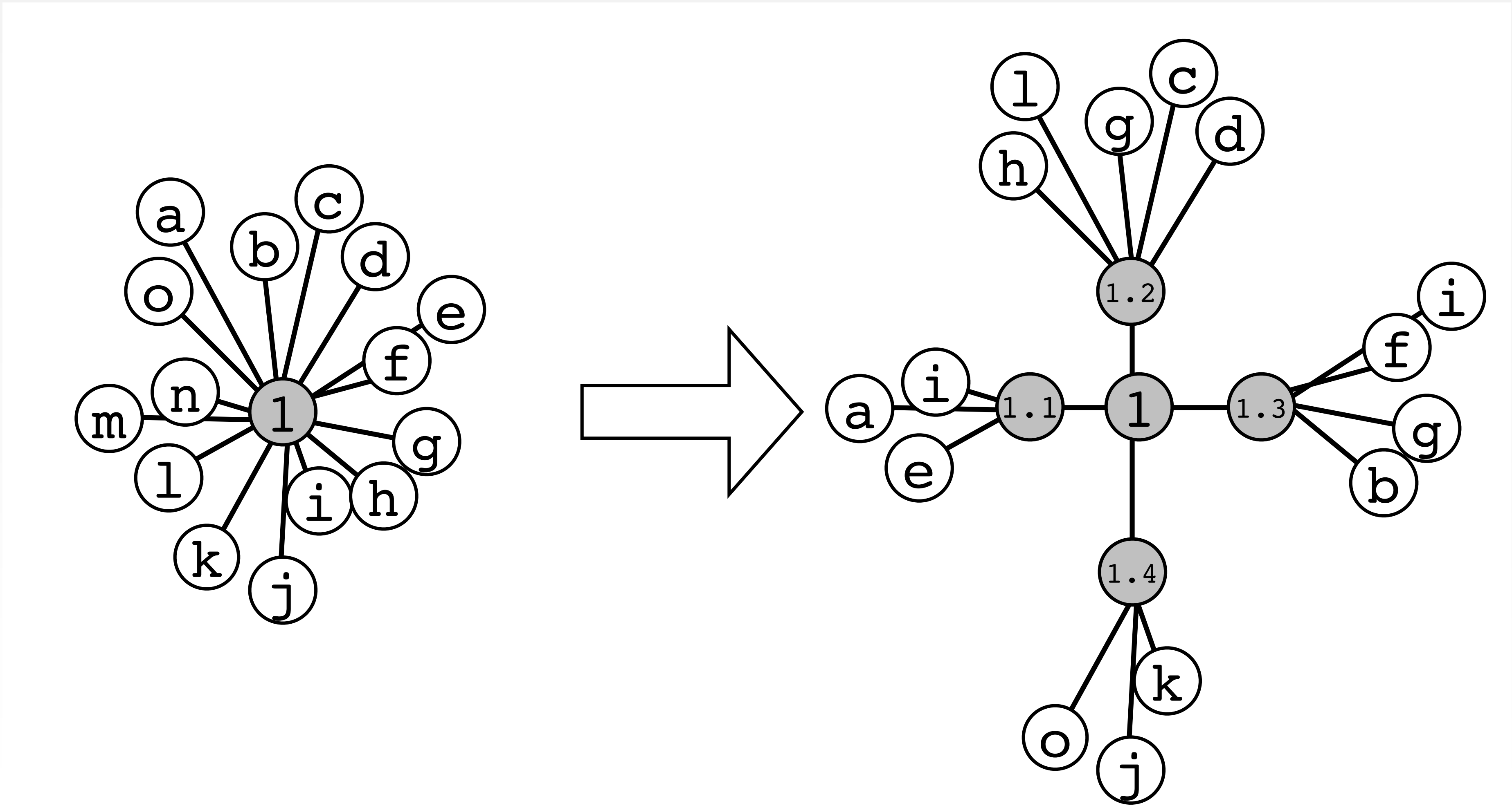
Watching Out for Skew

Final output:

- Union of stars, one for each component
- This should worry you!
  - In case of a single component, get one star with linear degree
  - In case of skewed component sizes, also get one star with linear degree

Dealing with Skew

Divide the computation of the minimum



- Can do this recursively c times
- Increase number of rounds by 1/c, each node's input at most n^c

But does it work?

Data (subset):

- UK Web graph: 106M nodes, 6.6B edges
- Google+ subgraph: 178M nodes, 2.9B edges
- Keyword similarity : 371M nodes, 3.5B edges
- Document similarity: 4,700M nodes, 452B edges

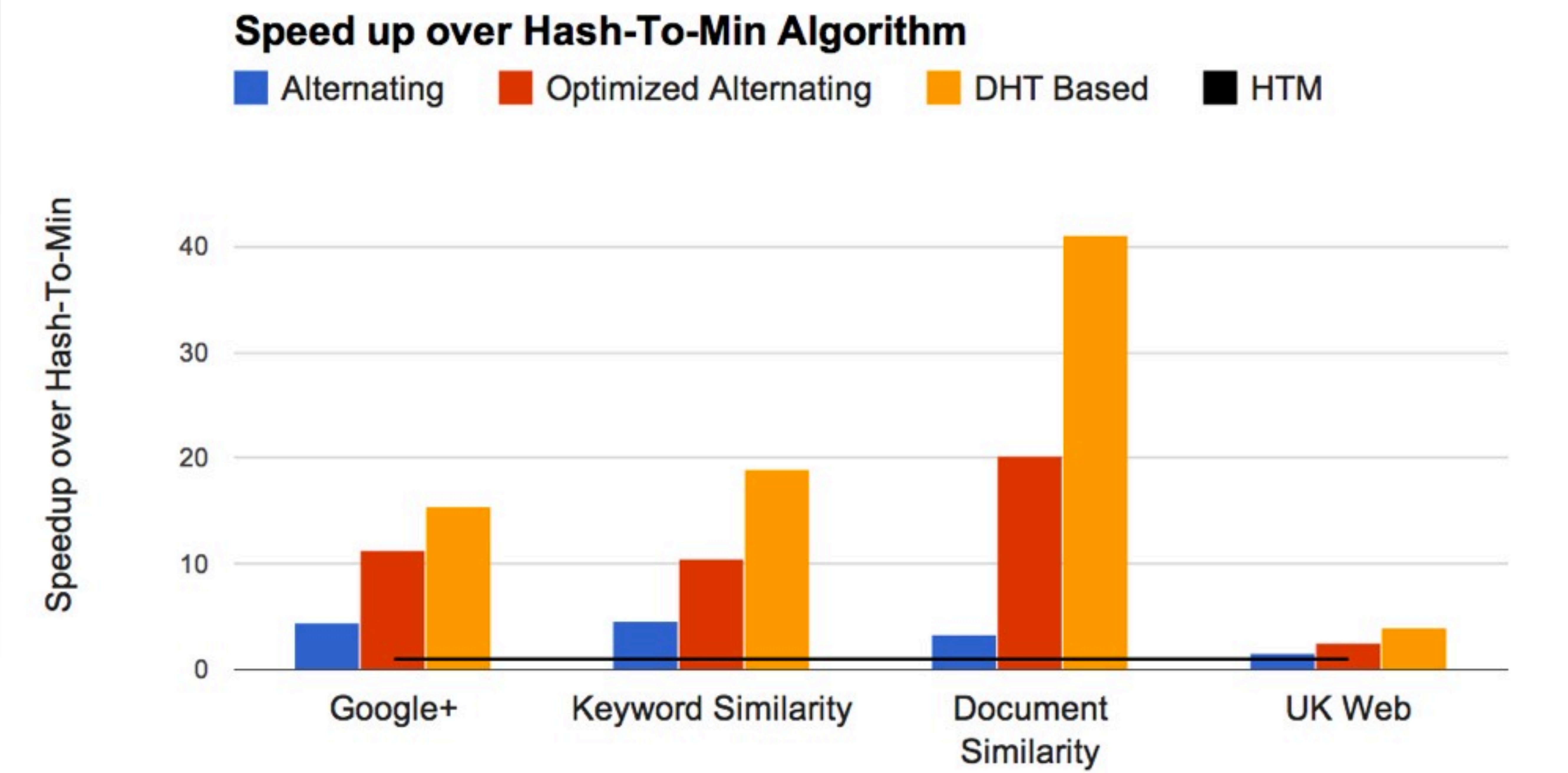
Algorithms:

- Hash2Min (previous MapReduce state of the art)
- DHT Implementation
- Alternating algorithm(skew optimized & non-optimized)

Setup:

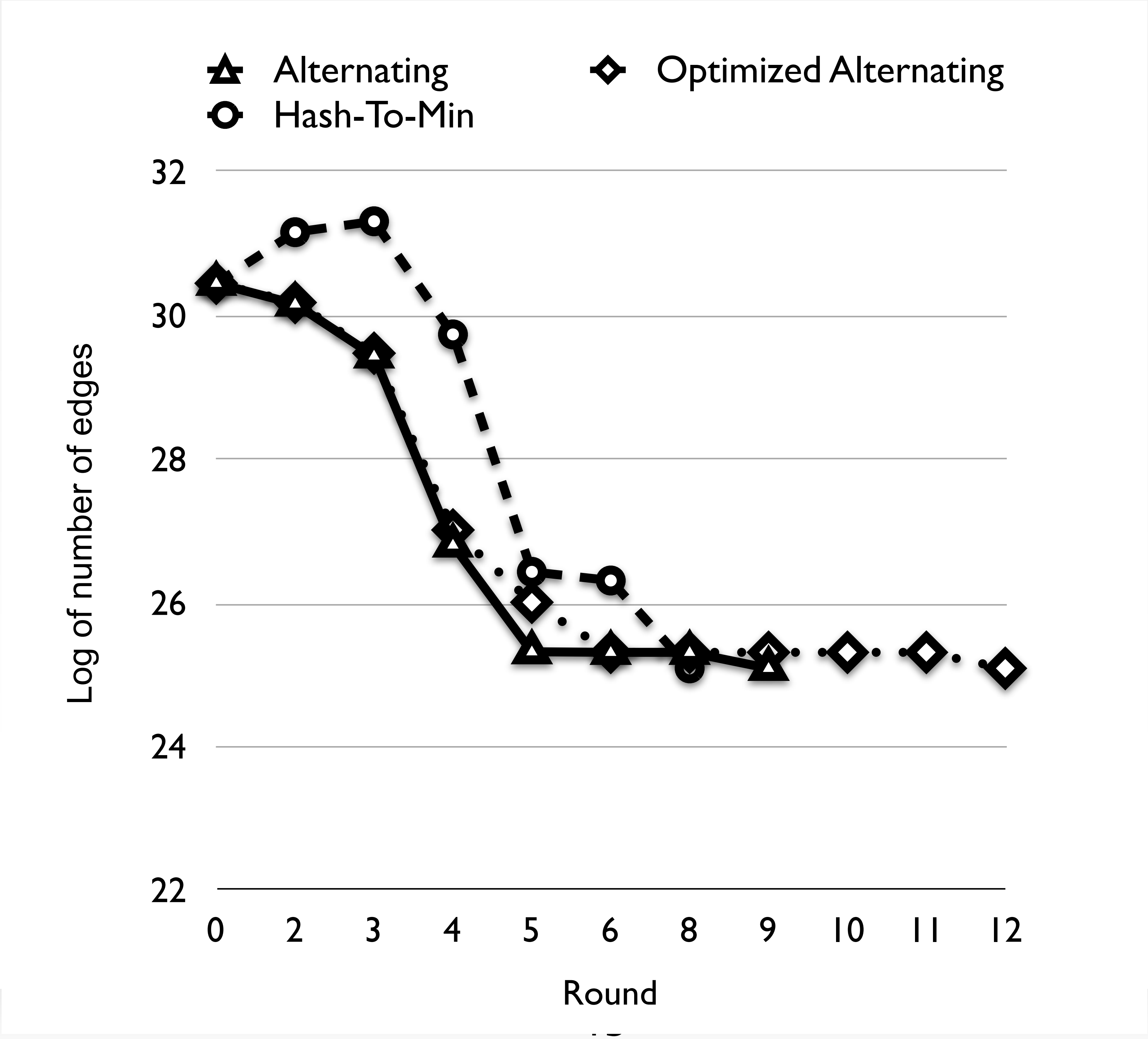
- Loaded cluster, look at median running times

Speedups



- 20x-40x faster on the document similarity graph
- Smaller improvements on smaller graphs

Graph Size



Alternating
Optimized Alternating
Hash-To-Min
32
30
28
26
24
22
0
2
3
4
5
6
8
9
10
11
12
Log of number of edges
Round

Conclusion

Connected Components

- Simple, local algorithms with $O(\log^2 n)$ round complexity
- Communication efficient (number of edges non-increasing)
- Open: Prove $O(\log n)$
- Open: Prove $\sim\log n$ lower bounds!

Conclusion

Connected Components

- Simple, local algorithms with $O(\log^2 n)$ round complexity
- Communication efficient (number of edges non-increasing)
- Open: Prove $O(\log n)$
- Open: Prove $\sim\log n$ lower bounds!

Algorithms:

- Evolve with the underlying system architecture
- Avoid embarrassingly slow embarrassingly parallel implementations

Thank You