CONGEST model
bandwidth limitations

Alkida Balliu
University of Freiburg

Part of the slides are from Jukka Suomela

CONGEST model

- **LOCAL** model: arbitrarily large messages
- **CONGEST** model: **O(log n)-bit** messages

CONGEST model

- Any of these can be encoded in $O(\log n)$-bit messages:
    - node identifier
    - number of nodes
    - number of edges
    - distance between two nodes …

CONGEST model

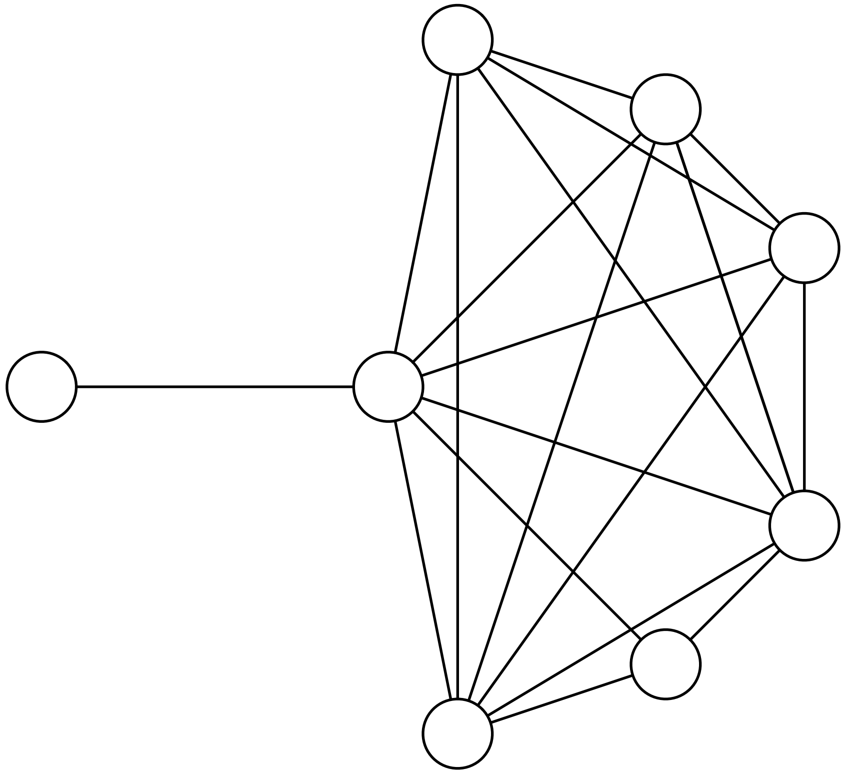
- Many algorithms that we have seen use small messages
  - can be used directly in CONGEST:
    - Example: coloring algorithms seen in the lectures
- There are some **exceptions**

Solving everything in LOCAL

- **Gather** the whole graph + solve the problem locally (e.g., by brute force)
  - $O(\mathrm{diam}(G))$ rounds
  - See animation here: https://jukkasuomela.fi/animations/local-horizon.gif

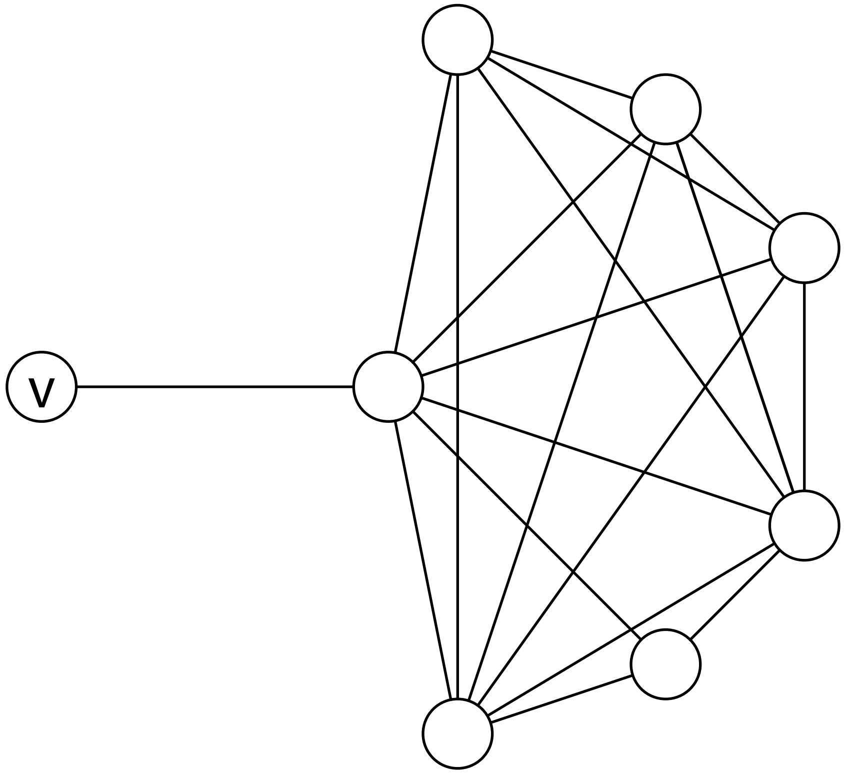
Algorithm Gather

- May need $\Omega(n^2)$-bit messages
  - Nodes have IDs from 1 to n



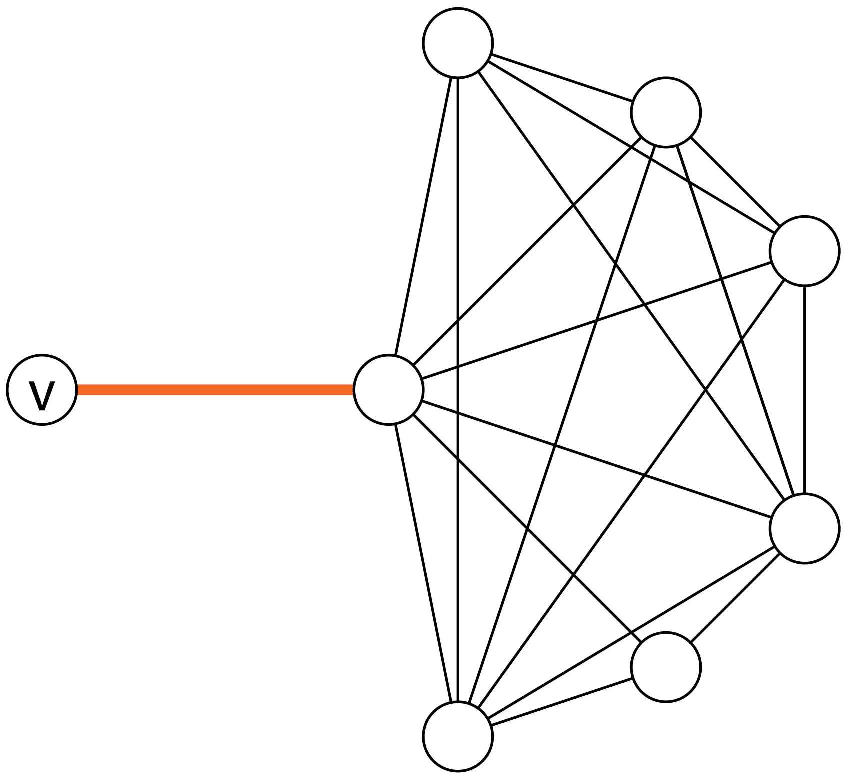
Algorithm Gather

- May need $\Omega(n^2)$-bit messages
  - Nodes have IDs from 1 to n



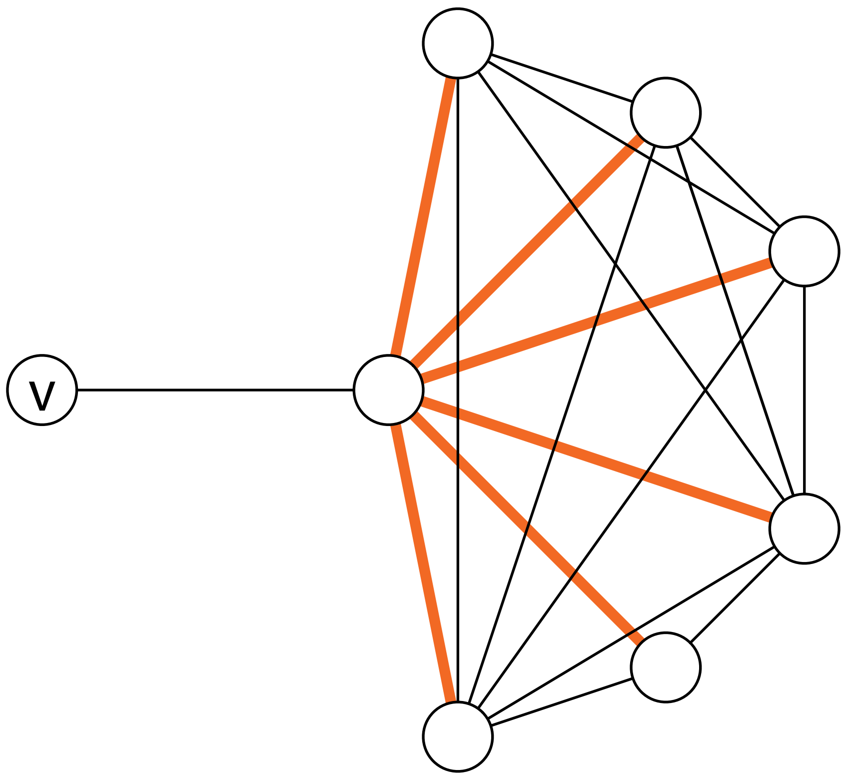
Algorithm Gather

- May need $\Omega(n^2)$-bit messages
    - Round 1



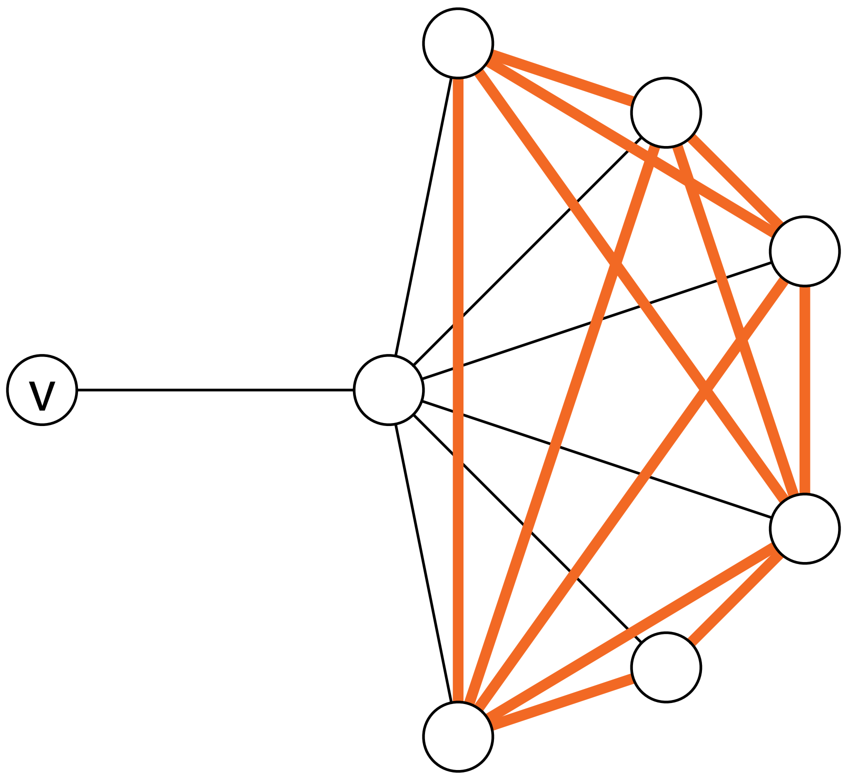
Algorithm Gather

- May need $\Omega(n^2)$-bit messages
  - Round 2



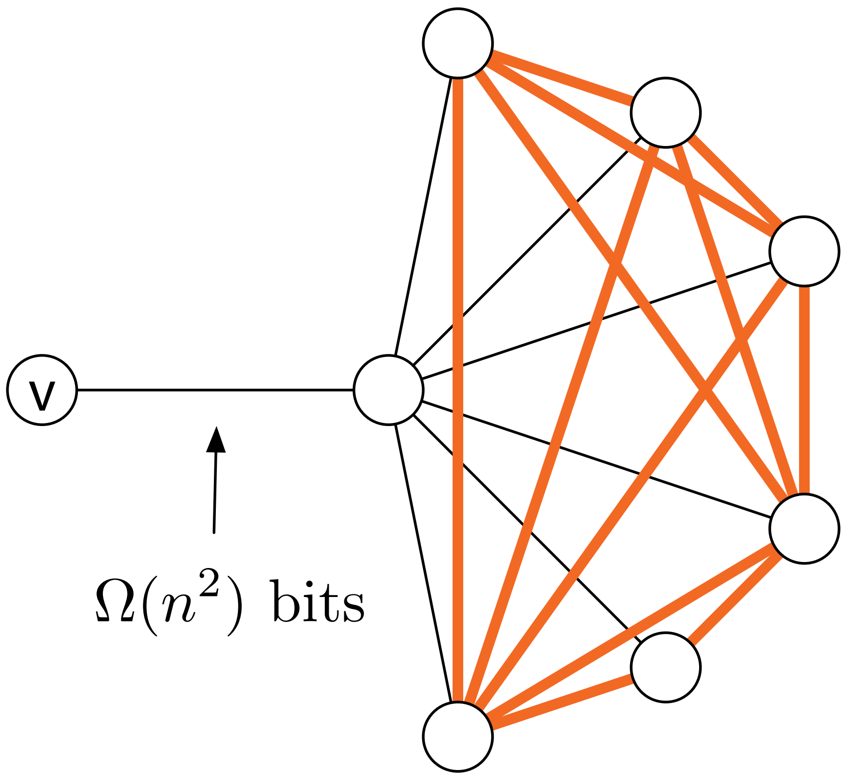
Algorithm Gather

- May need $\Omega(n^2)$-bit messages
    - Round 3



Algorithm Gather

- May need $\Omega(n^2)$-bit messages
  - Round 3, send the adjacency matrix



Algorithm Gather

- May need $\Omega(n^2)$-bit messages
- Cannot directly be used in CONGEST
- **Exercise**: gather all the graph in CONGEST in $O(|E|)$ rounds

CONGEST model

- $O(n)$ time trivial in the LOCAL model
  - brute force approach: Gather + solve locally
- **$O(n)$** time **non-trivial** in the **CONGEST** model

Today

- How to find **all-pairs shortest paths (APSP)** in $O(n)$ **time** in the CONGEST model
  [Holzer, Wattenhofer]

- Lower bound of $\Omega(n/\log n)$ rounds for **APSP**
  [Frischknecht, Holzer, Wattenhofer]

Today

- How to find **all-pairs shortest paths (APSP)** in $O(n)$ **time** in the CONGEST model
  [Holzer, Wattenhofer]

- Lower bound of $\Omega(n/\log n)$ rounds for **APSP**
  [Frischknecht, Holzer, Wattenhofer]

(Complexity of APSP in CONGEST: $\Theta(n/\log n)$)

Today

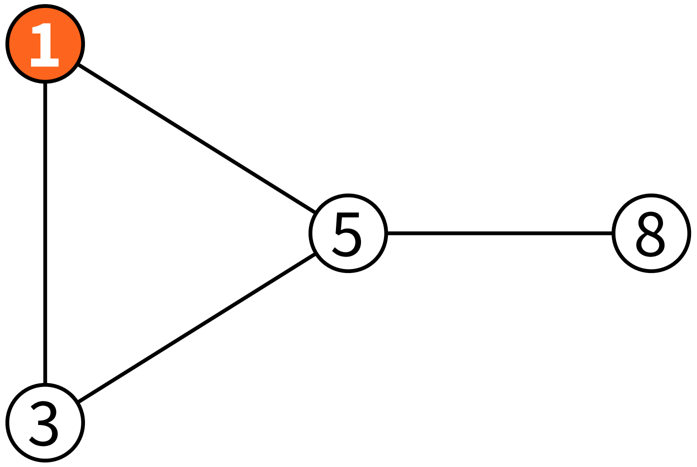
- How to find **all-pairs shortest paths (APSP)** in **$O(n)$ time** in the CONGEST model ☜
  [Holzer, Wattenhofer]

- Lower bound of **$\Omega(n/\log n)$** rounds for **APSP**
  [Frischknecht, Holzer, Wattenhofer]

(Complexity of APSP in CONGEST: **$\Theta(n/\log n)$**)

Single-source shortest paths

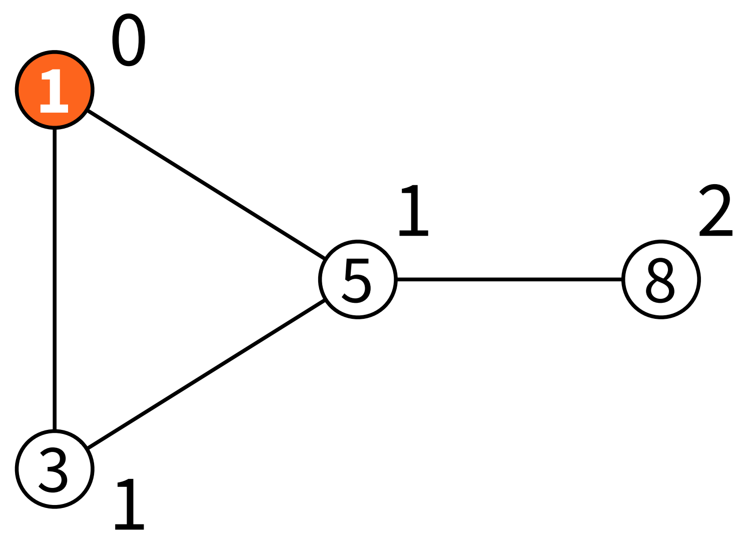
Input:



Single-source shortest paths

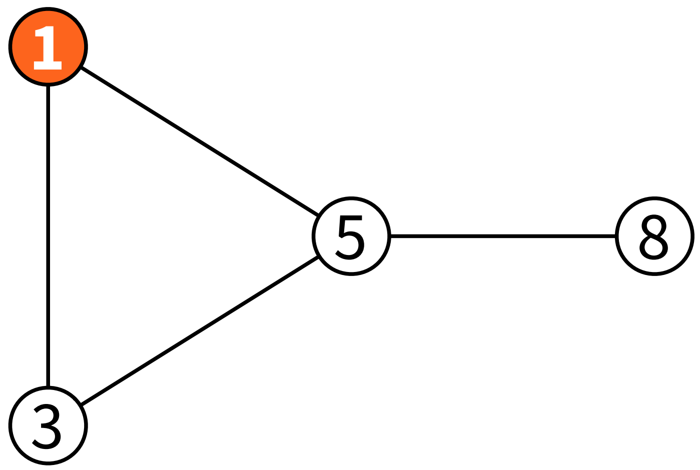
Distances from *s*

Output:



BFS tree

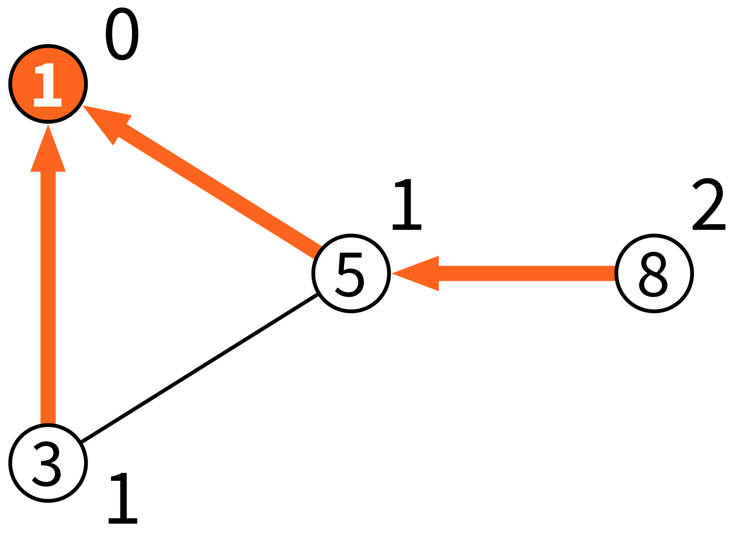
Input:



BFS tree

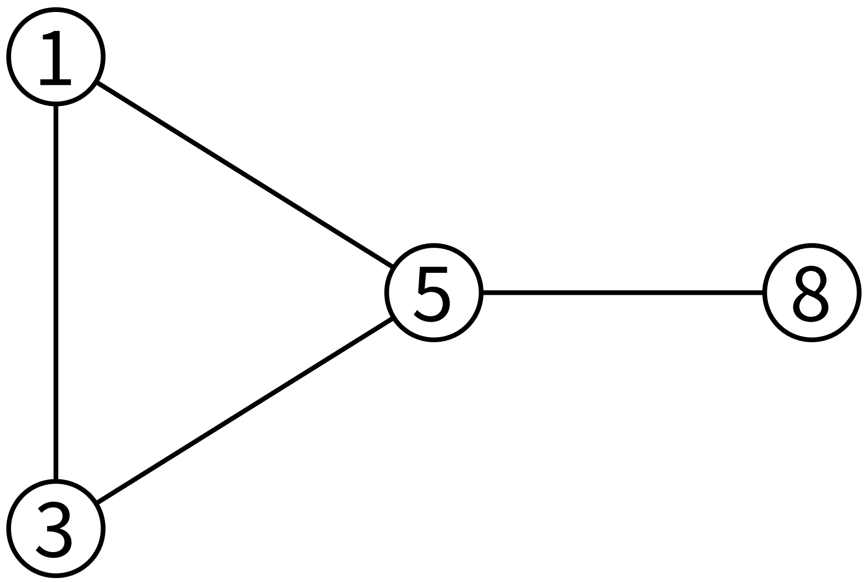
- Distances from *s* + shortest paths

Output:



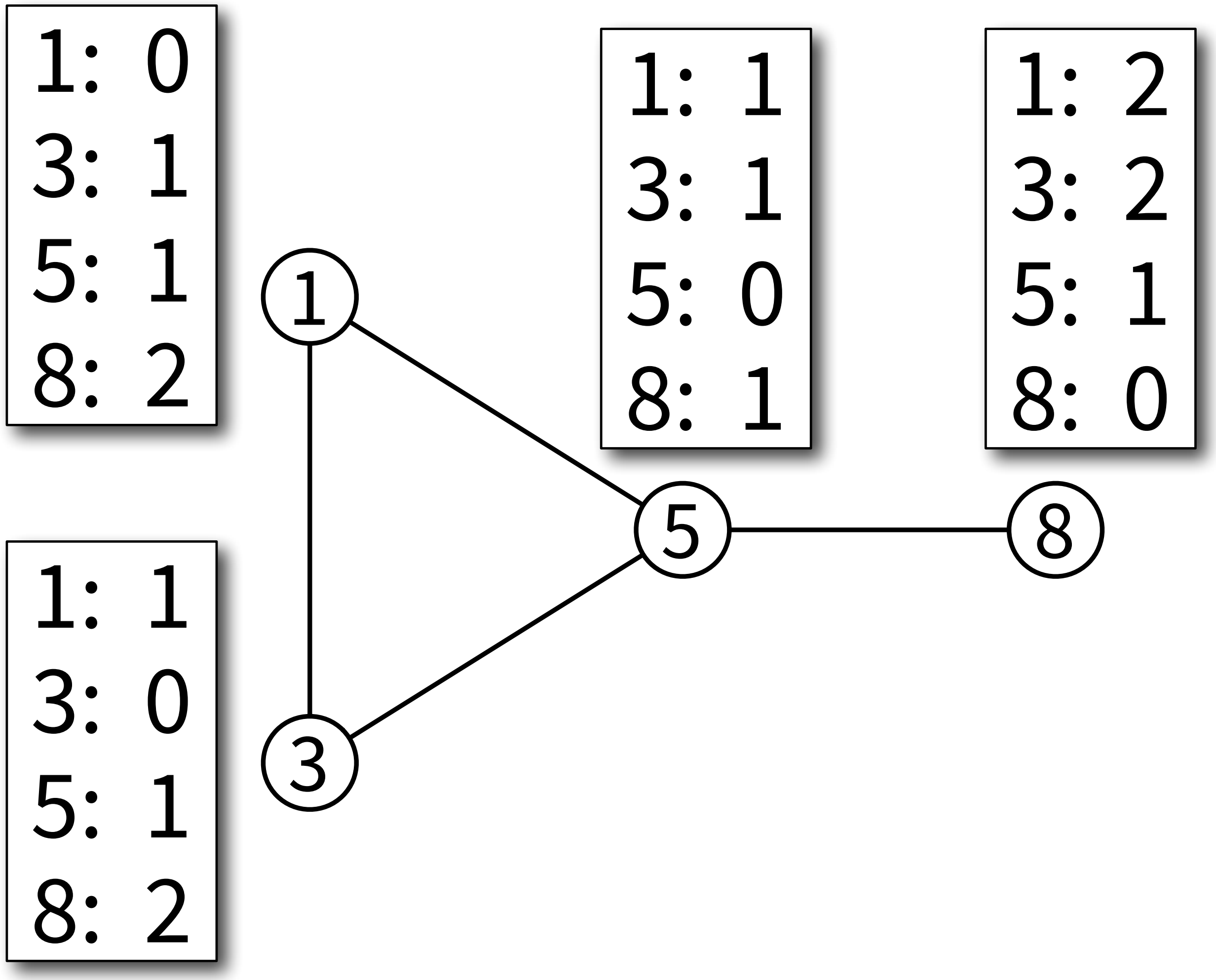
All-pairs shortest paths

Input:



All-pairs shortest paths

Output:

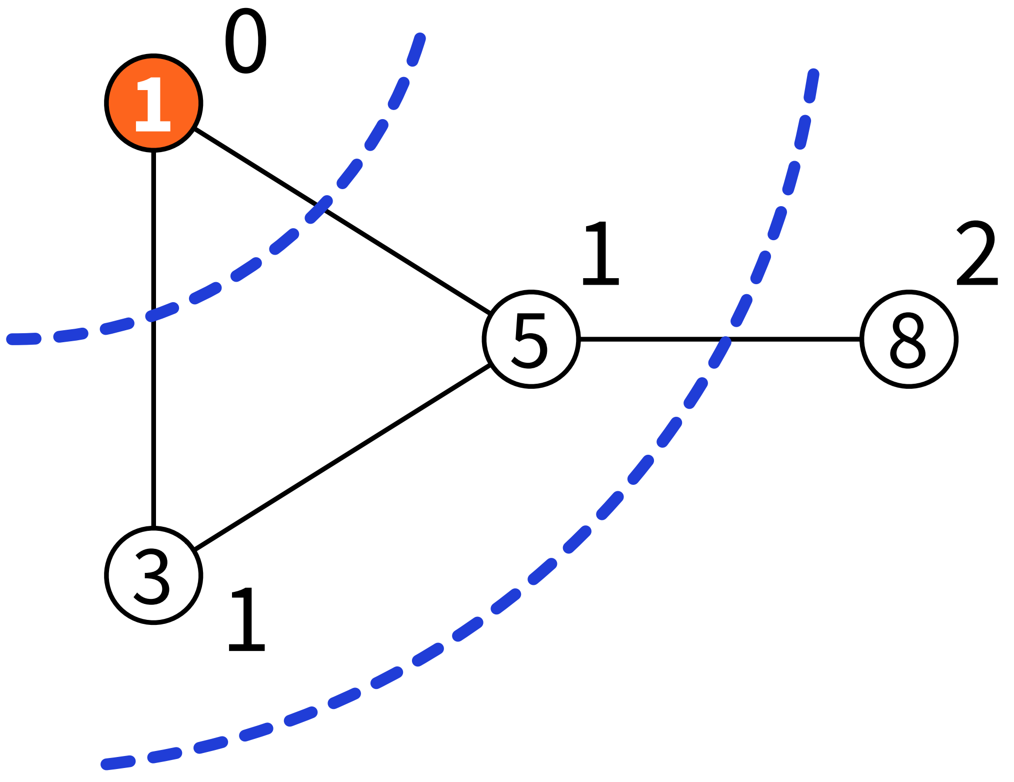


Algorithm Wave

- Solves **single-source shortest paths (SSSP)** in time $O(\mathrm{diam}(G))$
- Leader/source sends a message “wave”, switches to state 0, stops
- Wave received in round t for the first time: send “wave”, switch to state t, stop
- In time $O(\mathrm{diam}(G))$ all nodes receive the wave

Algorithm Wave

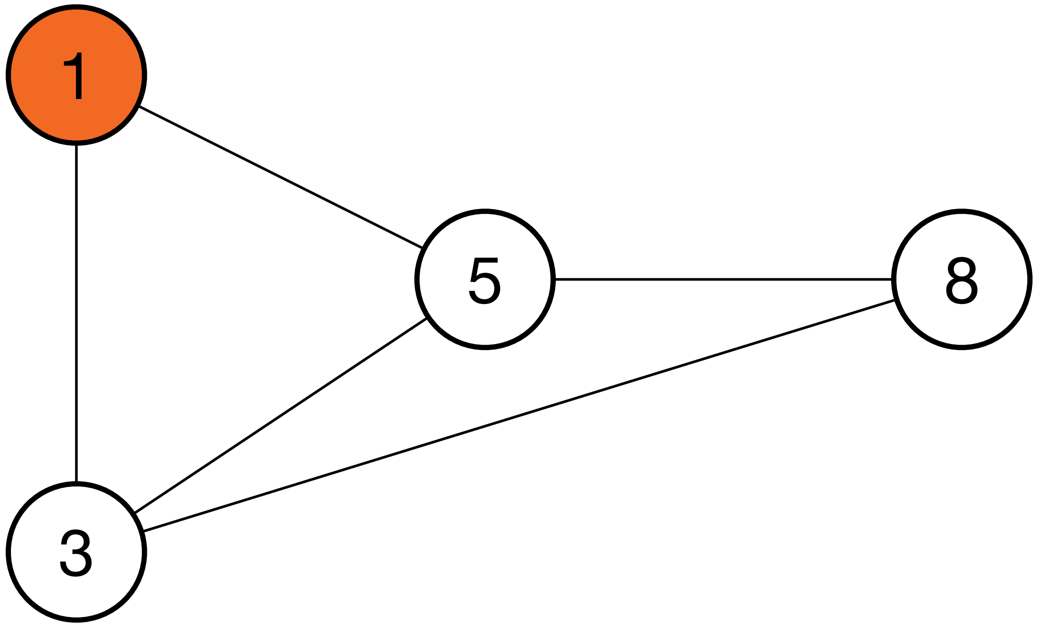
Output:



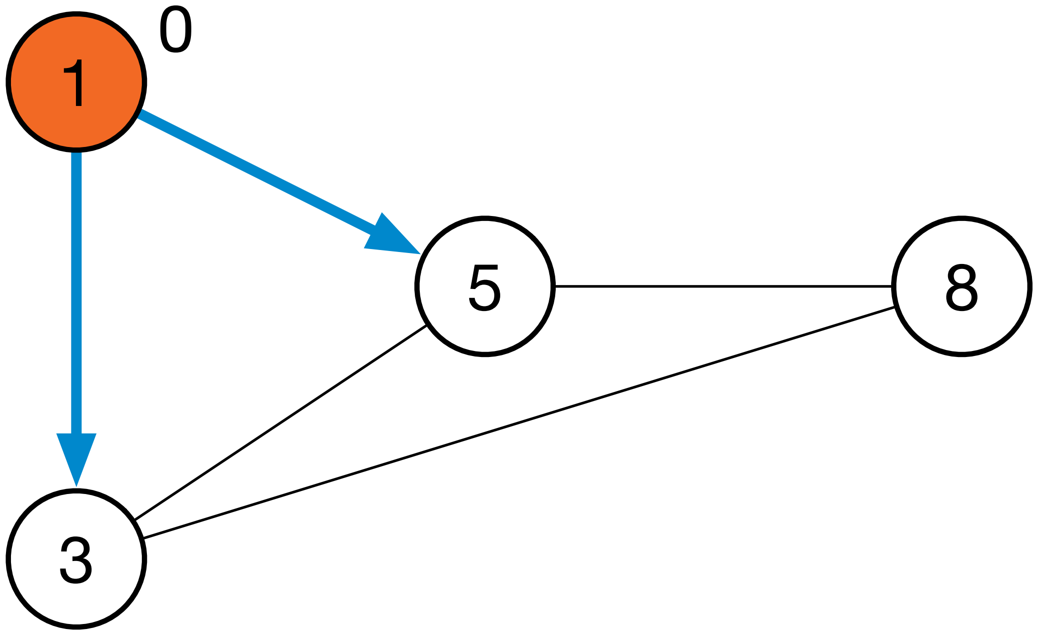
Algorithm BFS

- **Wave + handshakes**
- Tree construction:
  - “proposal” + “accept”
  - everyone knows their parent & children
- Acknowledgements back from leaf nodes

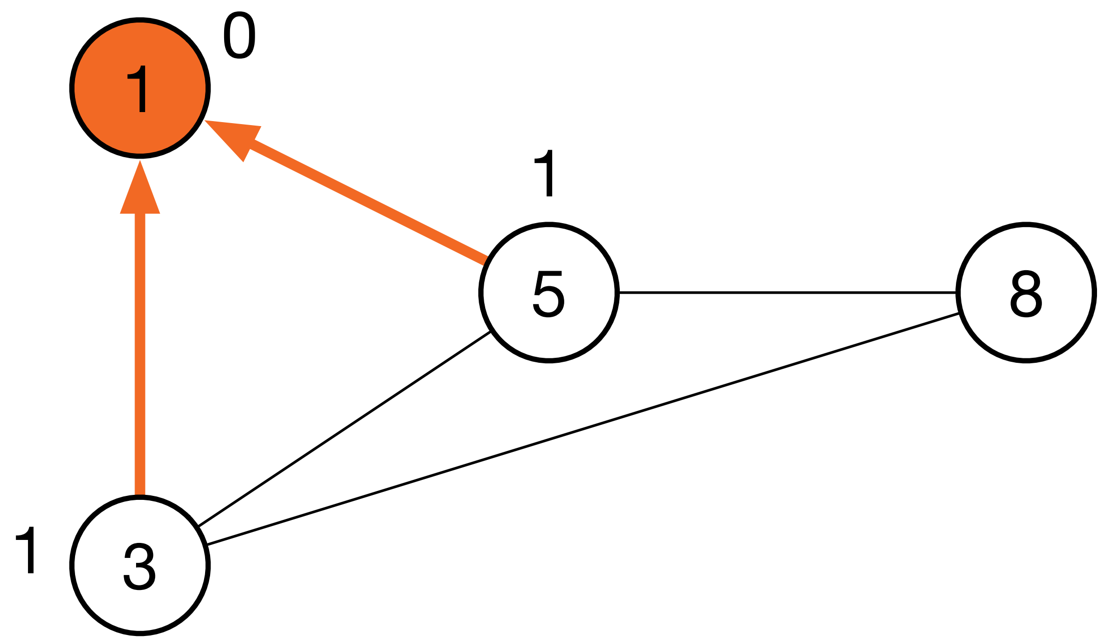
Algorithm BFS



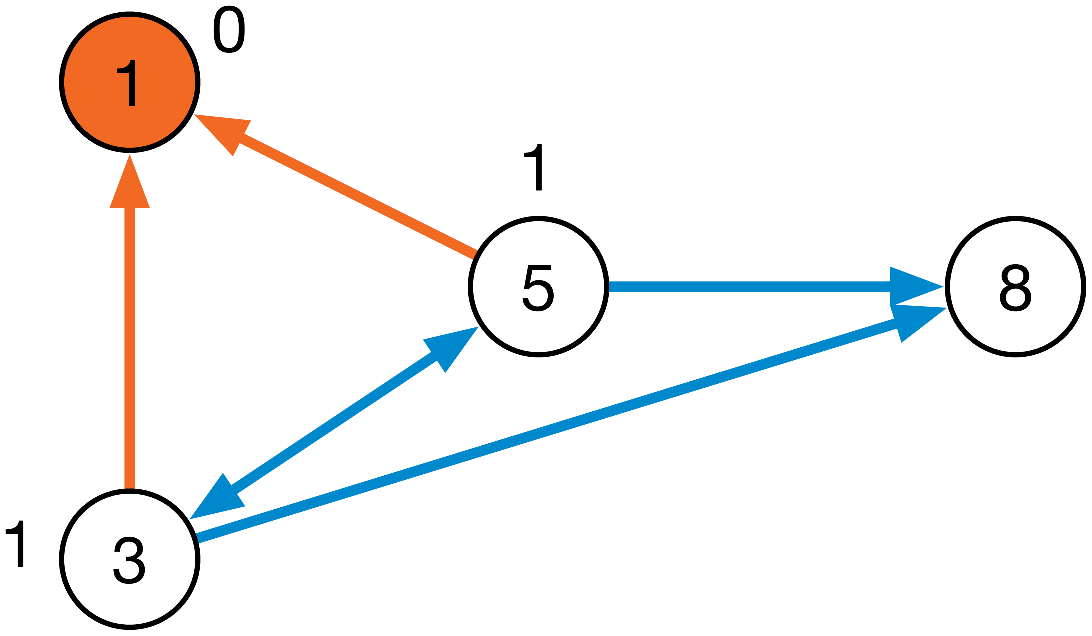
Algorithm BFS



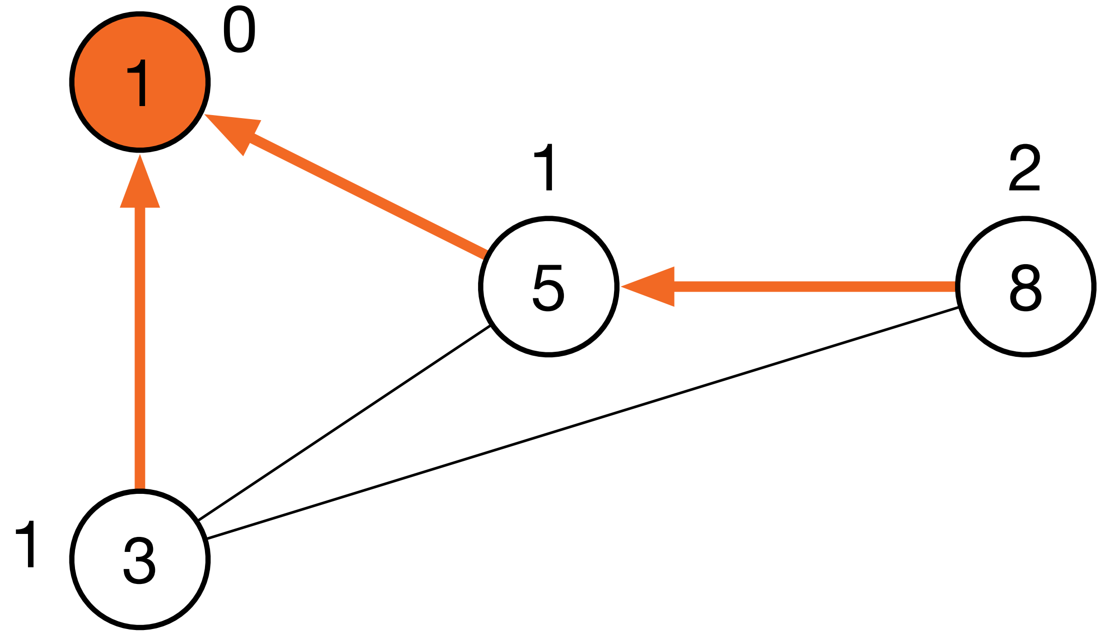
Algorithm BFS



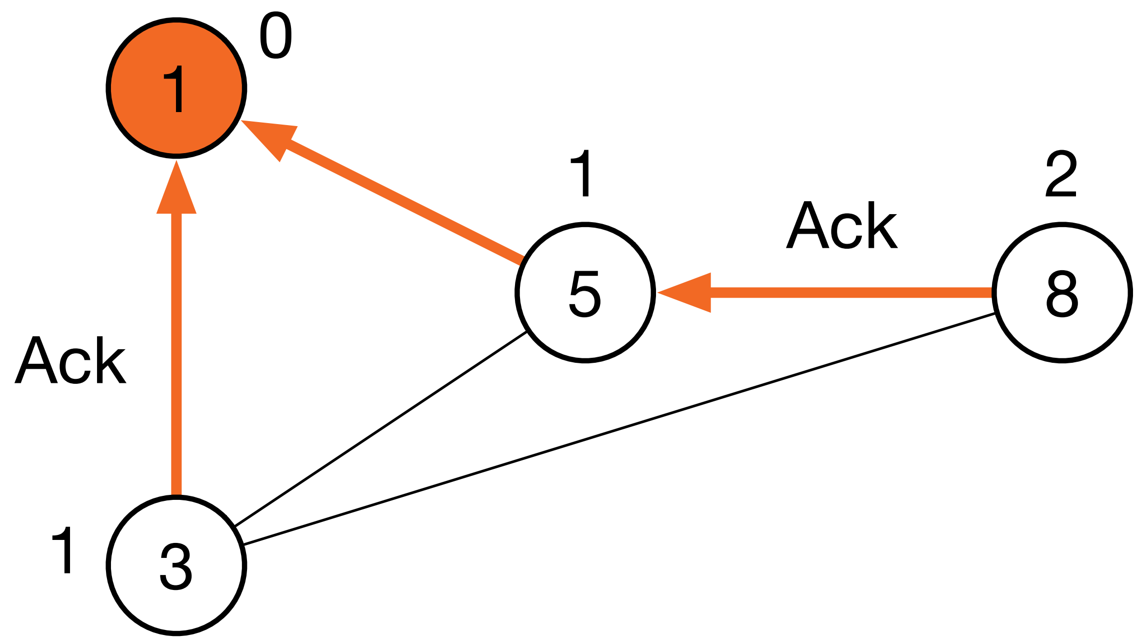
Algorithm BFS



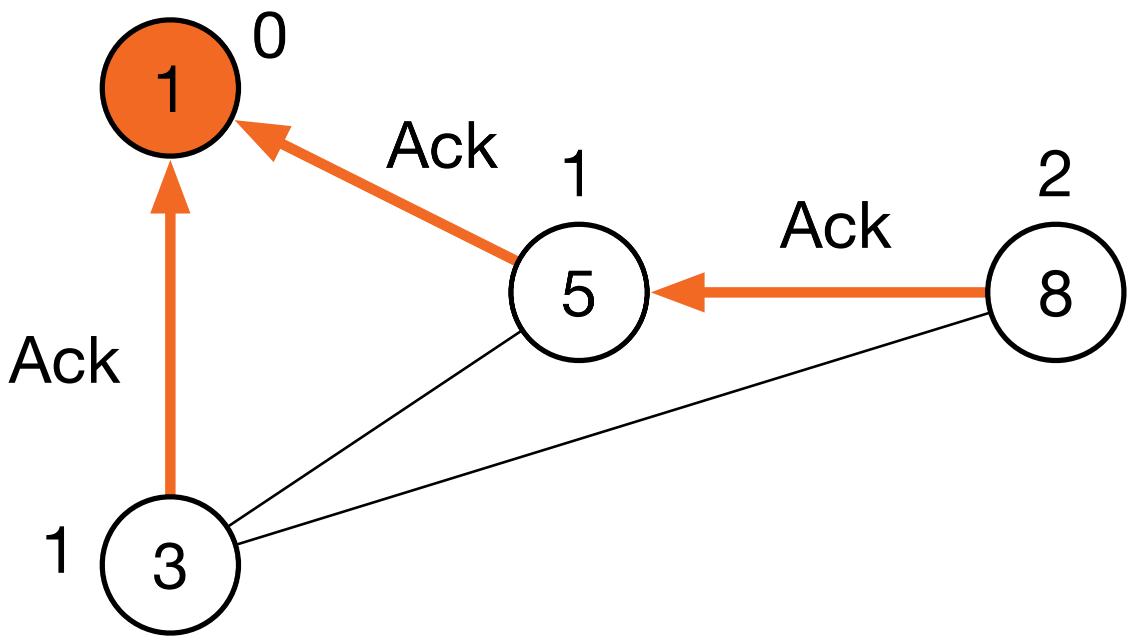
Algorithm BFS



Algorithm BFS



Algorithm BFS



Algorithm Leader

- Each node creates a separate BFS process
  - each node v pretends to be the root
  - messages of the BFS started by v contain ID(v)
- When two BFS processes “collide”, the one with the smaller root “wins”
  - each node only needs to send messages related to ***one BFS process***
- One tree wins everyone else → leader

Recap until now

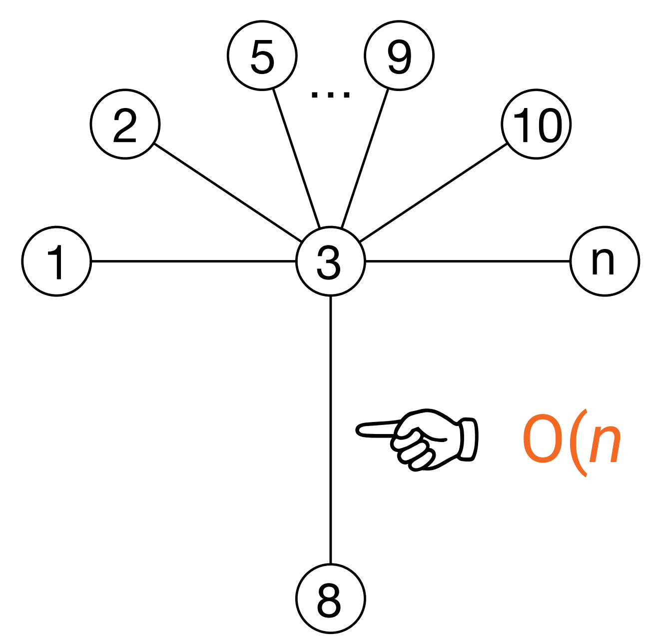
- **SSSP**: Wave algorithm
- **BFS tree**: Wave algorithm + acceptance/rejections
- **Leader election**: Many BFS in parallel
- All these problems can be solved in $\boldsymbol{O(\mathrm{diam}(G))}$ rounds in the **CONGEST** model

Algorithm APSP

- **Basic idea**: run Wave from each node
- **Challenge**: congestion

Algorithm APSP

- **Basic idea**: run Wave from each node
- **Challenge**: congestion
  - all waves parallel → too many bits per edge



O(n log n) bits

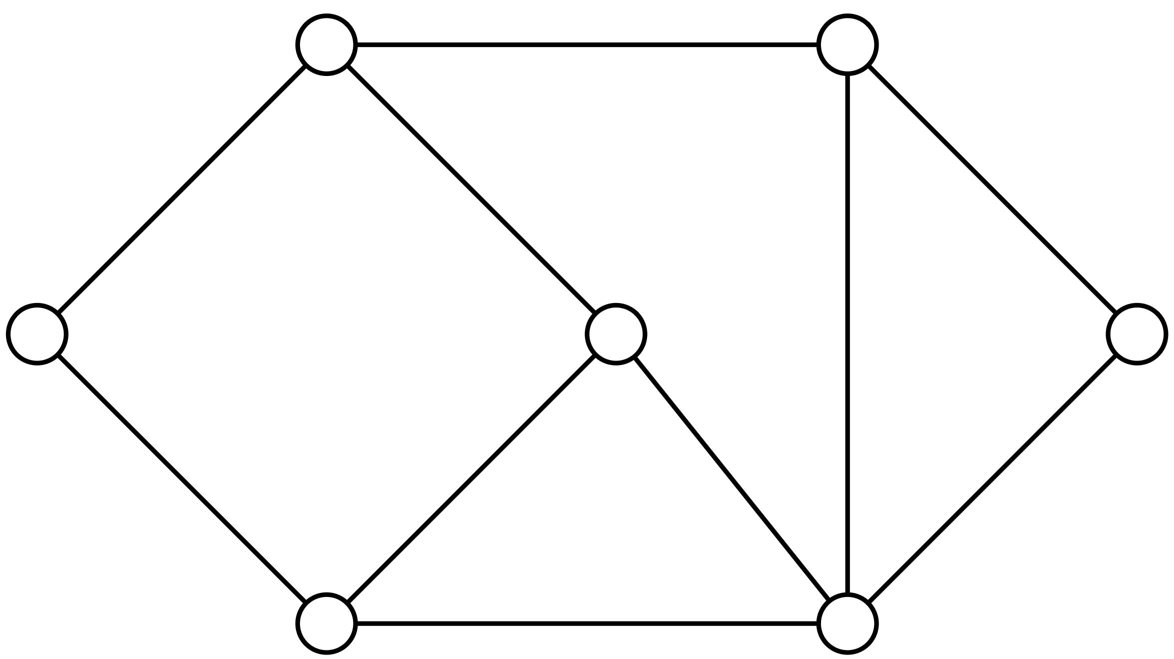
Algorithm APSP

- **Basic idea**: run Wave from each node
- **Challenge**: congestion
  - all waves parallel → too many bits per edge
  - all waves sequentially → takes too long

Algorithm APSP

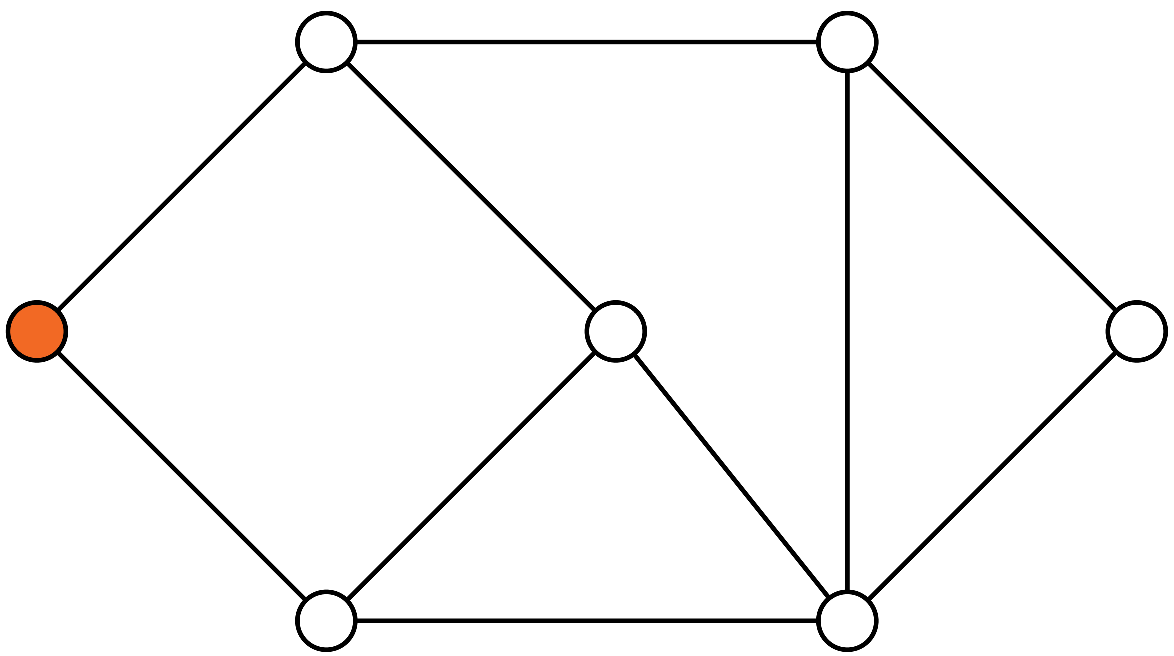
- **Basic idea**: run Wave from each node
- **Challenge**: congestion
  - all waves parallel → too many bits per edge
  - all waves sequentially → takes too long
- Solution: **pipelining**
  - all waves in parallel in such a way that each node propagates at most one wave per round

Algorithm APSP



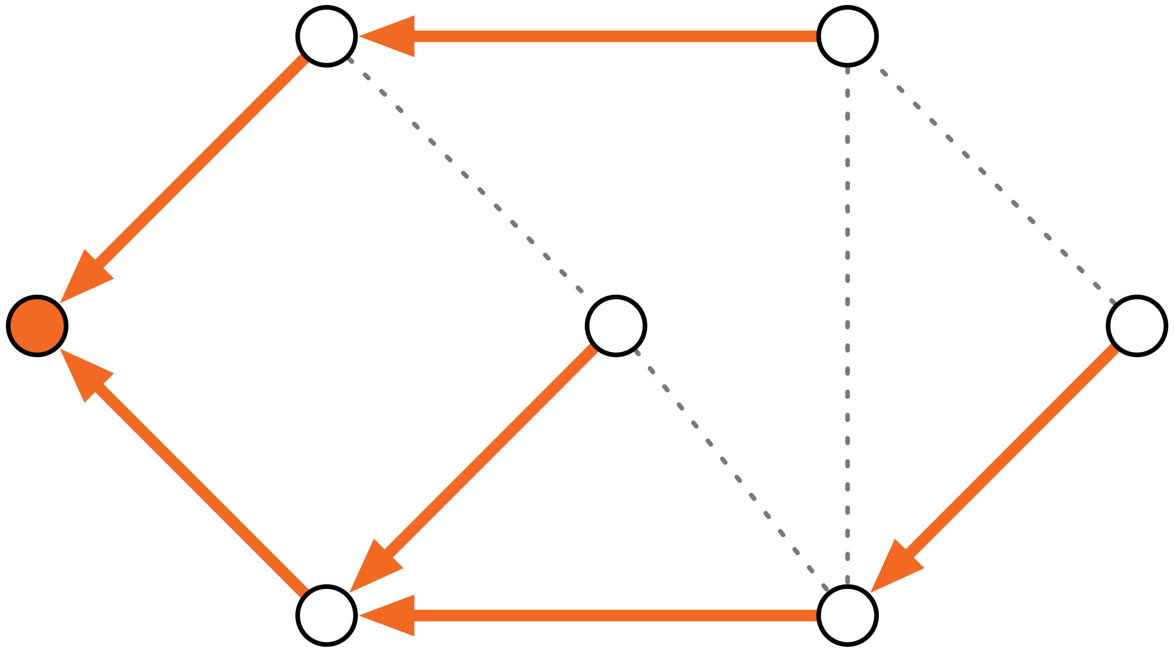
Algorithm APSP

- Elect leader



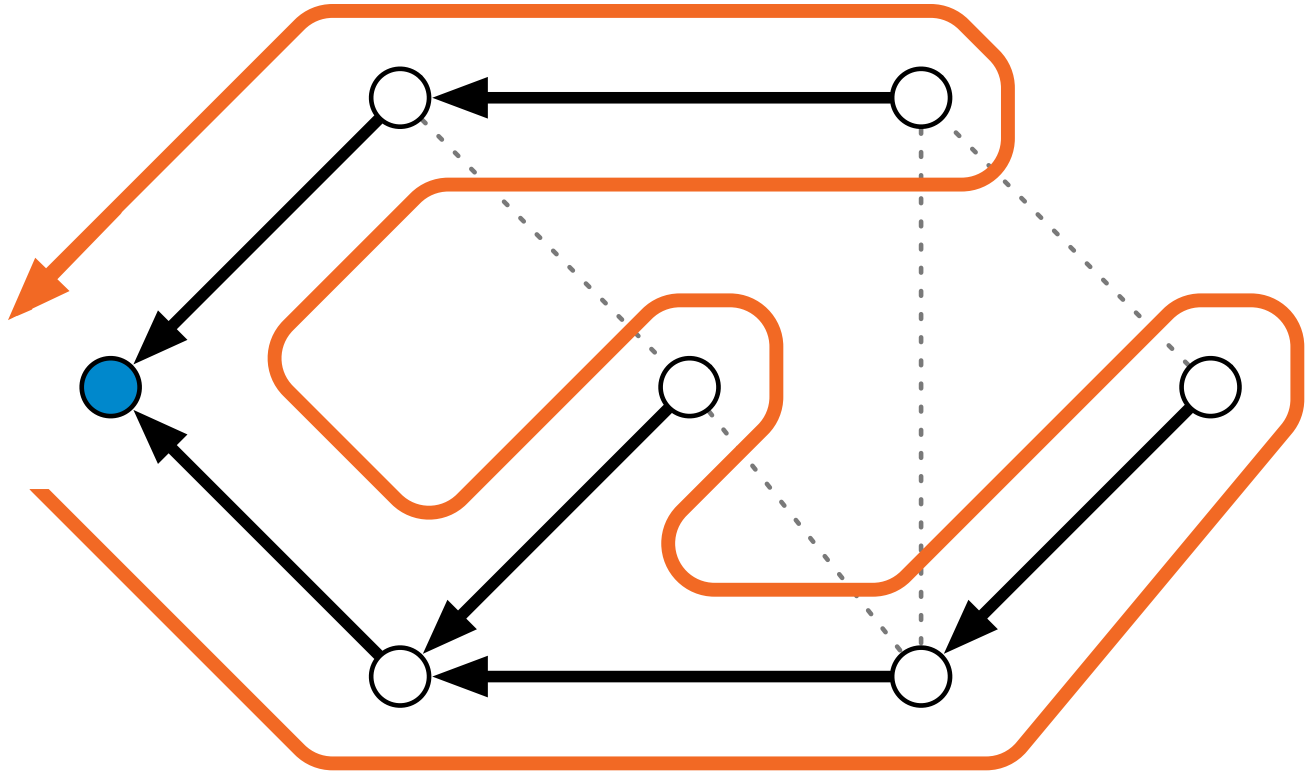
Algorithm APSP

- Elect leader, construct BFS tree



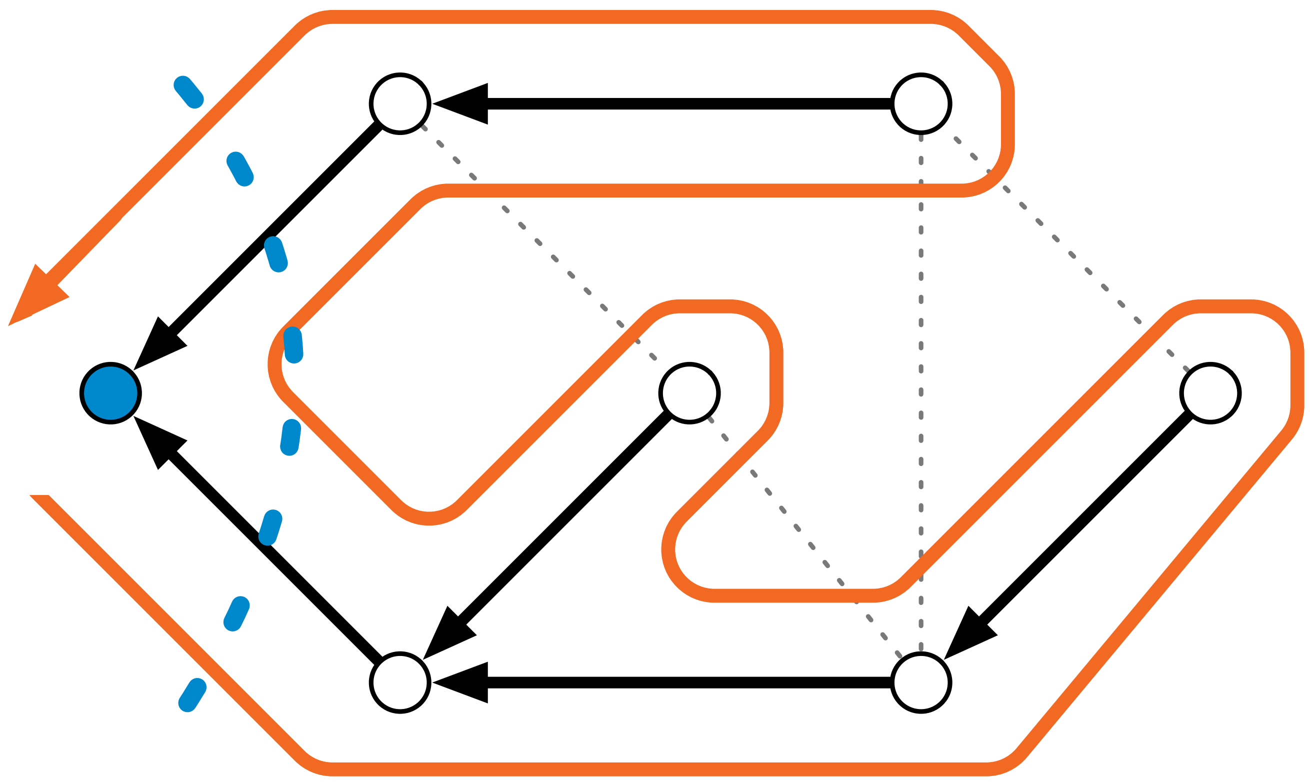
Algorithm APSP

- Move token along BFS tree slowly (every 2 rounds)



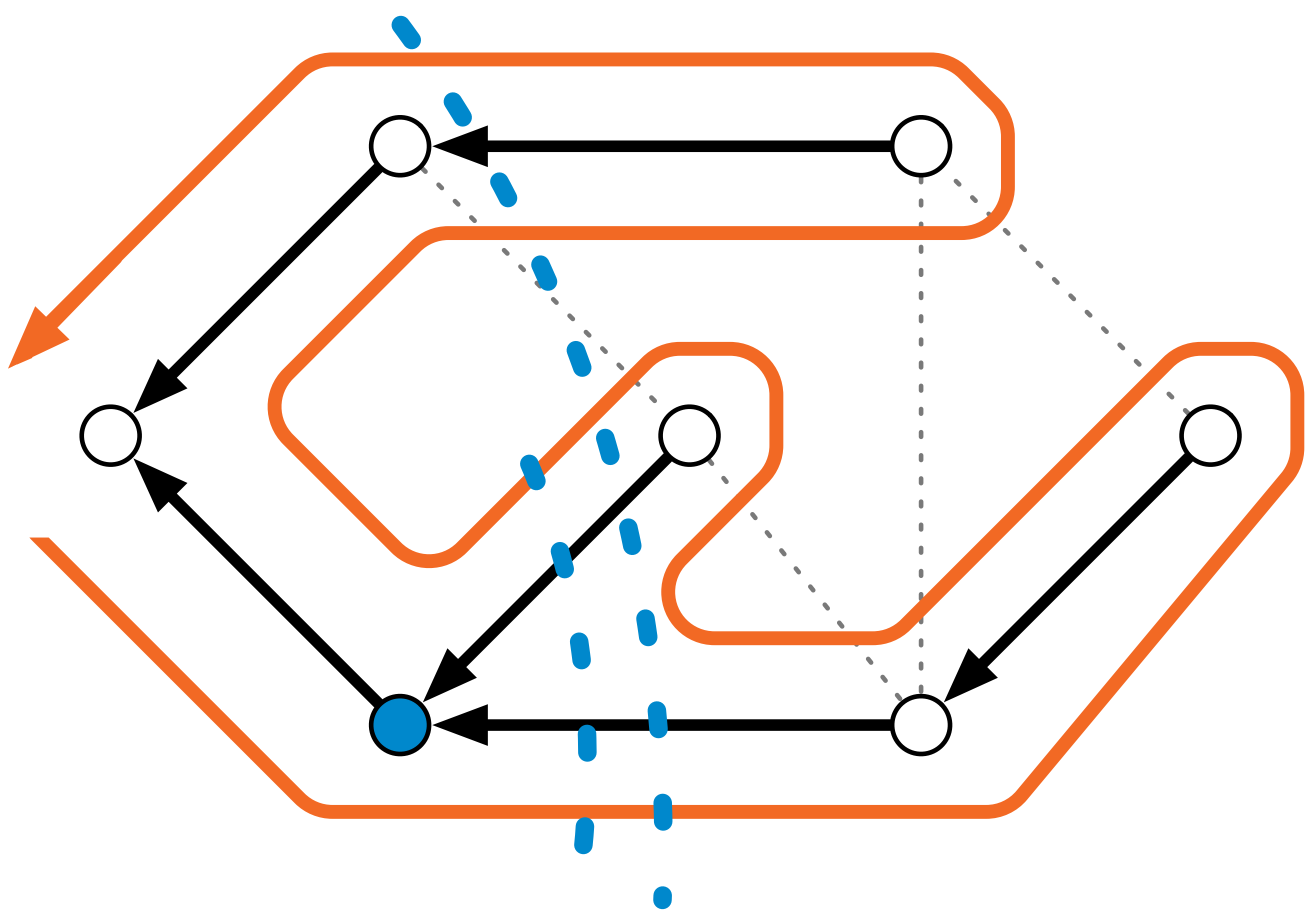
Algorithm APSP

- Move token along BFS tree slowly (every 2 rounds)



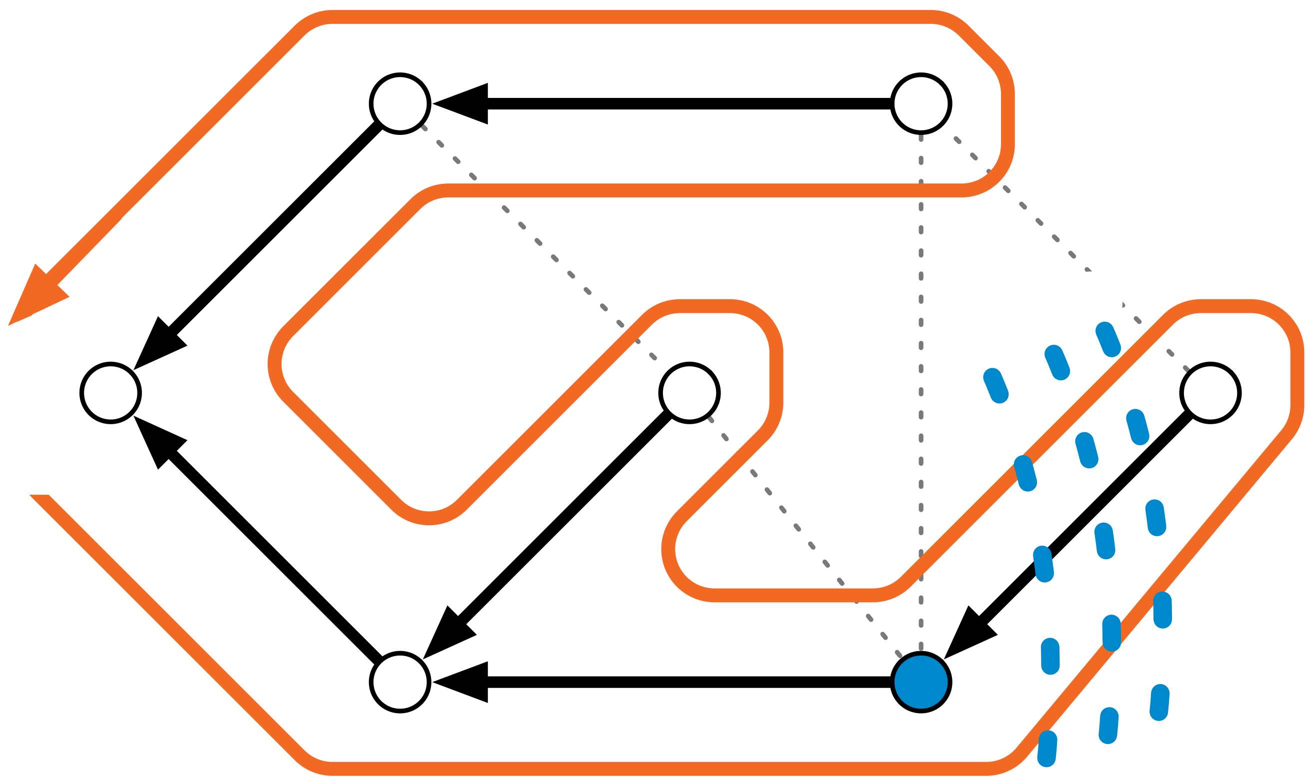
Algorithm APSP

- Move token along BFS tree slowly (every 2 rounds)



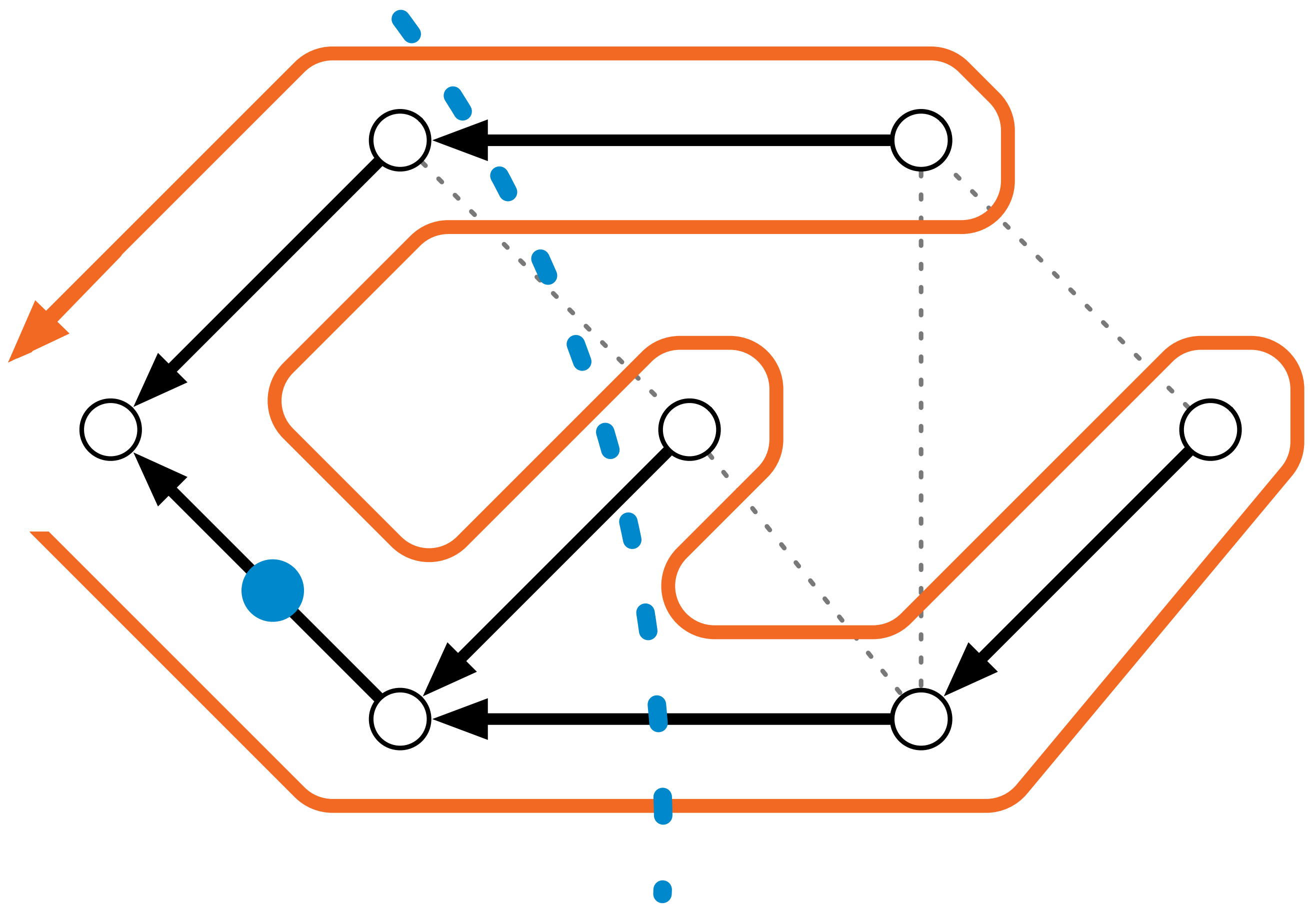
Algorithm APSP

- Move token along BFS tree slowly (every 2 rounds)



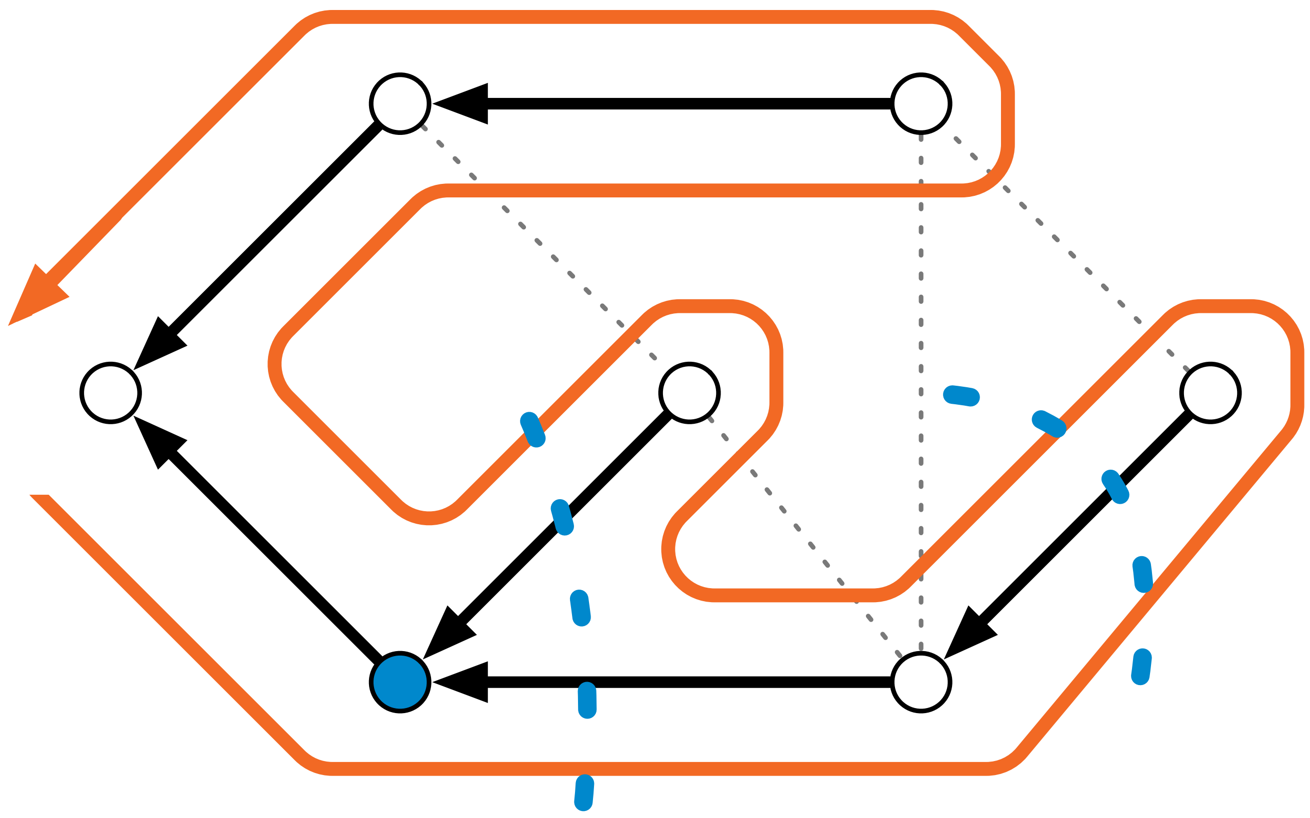
Algorithm APSP

- Move token along BFS tree slowly (every 2 rounds)



Algorithm APSP

- Move token along BFS tree slowly (every 2 rounds)



Algorithm APSP

- See animation here: https://jukkasuomela.fi/apsp/

Algorithm APSP: runtime

- Leader + BFS: $O(\text{diam}(G))$ rounds
- $|E|$ in a BFS tree: $\boldsymbol{n-1}$
- Token traverses **2** times each edge of the BFS tree
- Total number of rounds:
  - $\mathbf{2(2(}\boldsymbol{n}\mathbf{-1)) + }\boldsymbol{O}\mathbf{(diam(}\boldsymbol{G}\mathbf{))} \in$ **$O(n)$ rounds**

Pipelining

- n operations, each operation takes time t
- **Parallel**: t rounds, bad congestion
- **Sequential**: nt rounds, no congestion
- **Pipelining**: $n + t$ rounds, no congestion

Lower bound for APSP

APSP requires **$\Omega(n/\log n)$** rounds

Lower bound for APSP

Lower bound of **$\Omega(n/\log n)$** for **computing the diameter**

⇩

Lower bound of **$\Omega(n/\log n)$** for **APSP**

From APSP to Diameter

- Given a **solution for APSP**, we can compute the **diameter in $O(\mathrm{diam}(G))$** rounds

From APSP to Diameter

- Given a **solution for APSP**, we can compute the **diameter in $O(\mathrm{diam}(G))$** rounds
  - Each node stores its maximum distance in G

From APSP to Diameter

- Given a **solution for APSP**, we can compute the **diameter in $O(\text{diam}(G))$** rounds
  - Each node stores its maximum distance in G
  - Construct a BFS tree in $O(\text{diam}(G))$

From APSP to Diameter

- Given a **solution for APSP**, we can compute the **diameter in $O(\mathrm{diam}(G))$** rounds
  - Each node stores its maximum distance in G
  - Construct a BFS tree in $O(\mathrm{diam}(G))$
  - Leaves send their maximum distance to parents

From APSP to Diameter

- Given a **solution for APSP**, we can compute the **diameter in $O(\text{diam}(G))$** rounds
  - Each node stores its maximum distance in G
  - Construct a BFS tree in $O(\text{diam}(G))$
  - Leaves send their maximum distance to parents
  - Non-leaves compute the maximum distance among their own and the ones of its children, send to parent

From APSP to Diameter

- Given a **solution for APSP**, we can compute the **diameter in $O(\mathrm{diam}(G))$** rounds
  - Each node stores its maximum distance in G
  - Construct a BFS tree in $O(\mathrm{diam}(G))$
  - Leaves send their maximum distance to parents
  - Non-leaves compute the maximum distance among their own and the ones of its children, send to parent
  - Root broadcasts

Lower bound for diameter $\Longrightarrow$ Lower bound for APSP

- Compute diameter in: T(APSP) + $O(\mathrm{diam}(G))$ rounds
- If computing the **diameter requires $\boldsymbol{\Omega(n/\log n)}$** rounds

$$\Downarrow$$

- **APSP must require $\boldsymbol{\Omega(n/\log n)}$** rounds
  in all graphs with diameter $o(n/\log n)$
  - T(APSP) + $o(n/\log n) \in \Omega(n/\log n) \Rightarrow$
    T(APSP) $\in$ $\boldsymbol{\Omega(n/\log n)}$

Lower bound for diameter $\Longrightarrow$ Lower bound for APSP

- Compute diameter in: T(APSP) + $O(\text{diam}(G))$ rounds
- If computing the **diameter requires $\boldsymbol{\Omega(n/\log n)}$** rounds

⇩

- **APSP must require $\boldsymbol{\Omega(n/\log n)}$** rounds
  in all graphs with diameter $\boldsymbol{o(n/\log n)}$ ☜
  - T(APSP) + $o(n/\log n) \in \Omega(n/\log n) \Rightarrow$
    T(APSP) $\in$ **$\boldsymbol{\Omega(n/\log n)}$**

Lower bound for diameter $\Longrightarrow$ Lower bound for APSP

- Compute diameter in: T(APSP) + $O(\mathrm{diam}(G))$ rounds
- If computing the **diameter requires $\Omega(n/\log n)$** rounds

$$\Downarrow$$

- **APSP must require $\Omega(n/\log n)$** rounds in all graphs with diameter $o(n/\log n)$
  - T(APSP) + $o(n/\log n) \in \Omega(n/\log n) \Rightarrow$ **T(APSP) $\in \Omega(n/\log n)$**

Computing the diameter

- Computing the diameter requires $\Omega(n/\log n)$ [Frischknecht, Holzer, Wattenhofer]
- The proof uses known results from 2-party **communication complexity**
  - Studies the minimum amount of communication (nr. of bits) needed in order to compute functions whose arguments are distributed among several parties
  - **Set disjointness** between **2 communication parties**

Set disjointness

Set disjointness

Set disjointness

Set disjointness

$\mathcal{A}$

Set disjointness

$\mathcal{A}$

$\mathcal{B}$

Set disjointness

- $\mathcal{A}, \mathcal{B} \subseteq \{1, 2, \dots, k\}$

$\mathcal{A}$

$\mathcal{B}$

Set disjointness

- $\mathcal{A}, \mathcal{B} \subseteq \{1, 2, \ldots, k\}$
- Output: 1 if $\mathcal{A} \cap \mathcal{B} = \emptyset$; 0 otherwise

$\mathcal{A}$

$\mathcal{B}$

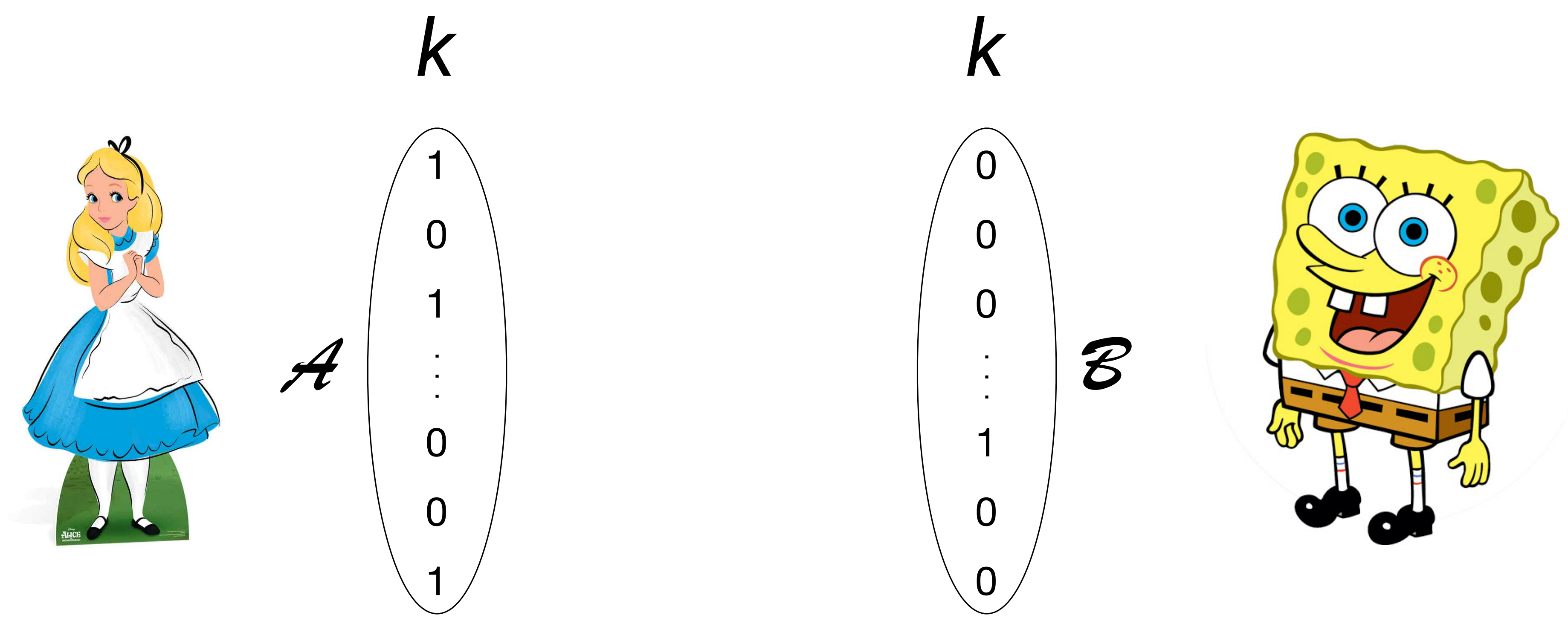
Set disjointness

- $\mathcal{A}, \mathcal{B} \subseteq \{1, 2, \ldots, k\}$
- Output: 1 if $\mathcal{A} \cap \mathcal{B} = \emptyset$; 0 otherwise
- String of k bits: 1 in position i if the i-th element is present, 0 otherwise

$$\mathcal{A} \begin{pmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \end{pmatrix} \mathcal{B}$$

Set disjointness

Alice and Bob need to **exchange Ω(k) bits** in order to solve set disjointness



Computing the diameter: Lower bound idea

$$\begin{pmatrix}1\\0\\1\\\vdots\\0\\0\\1\end{pmatrix}_{k} \qquad\qquad \begin{pmatrix}0\\0\\0\\\vdots\\1\\0\\0\end{pmatrix}_{k}$$

Computing the diameter: Lower bound idea

Algorithm that computes the diameter $\Longrightarrow$ Solution to the set disjointness problem

k $\Theta(\sqrt{k})$ $\Theta(\sqrt{k})$ k

Computing the diameter: Lower bound idea

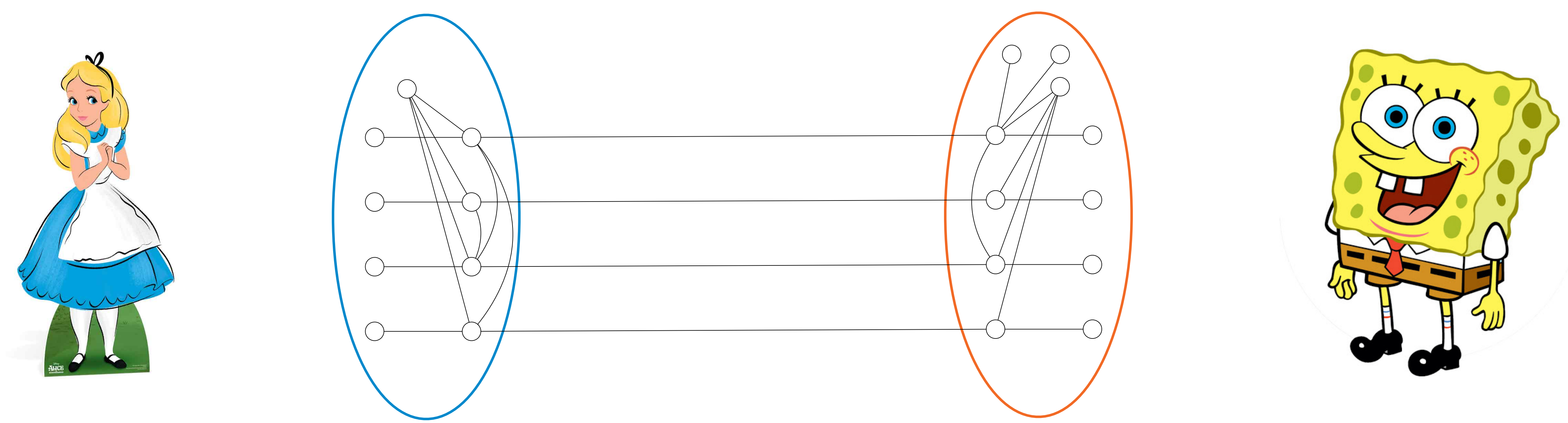
Diameter = **4** ⇒ the sets are **disjoint**

Diameter ≥ **5** ⇒ the sets are **not** disjoint

k $\Theta(\sqrt{k})$ $\Theta(\sqrt{k})$ k

Computing the diameter: Lower bound idea

Diameter in o(n/log n) rounds ⇒ Diameter exchanging o(k) bits ⇒ Set disjointness exchanging o(k) bits



k $\Theta(\sqrt{k})$ $\Theta(\sqrt{k})$ k

Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

$k = 9$

Lower bound for computing the diameter

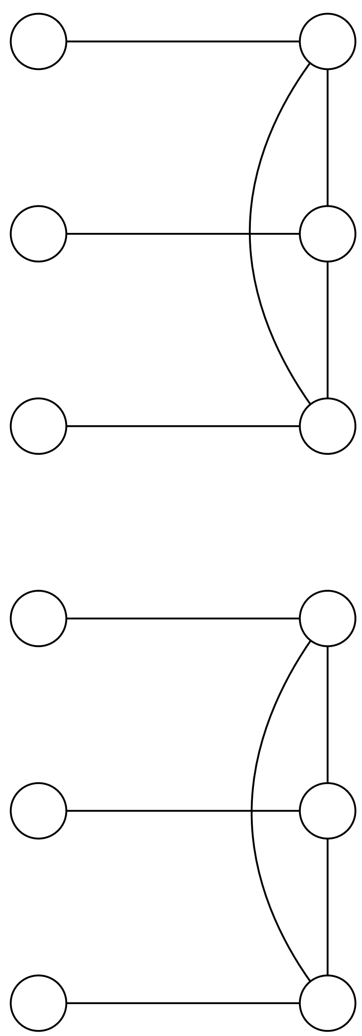
1
0
0
0
1
1
0
1
0

$k = 9$

Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

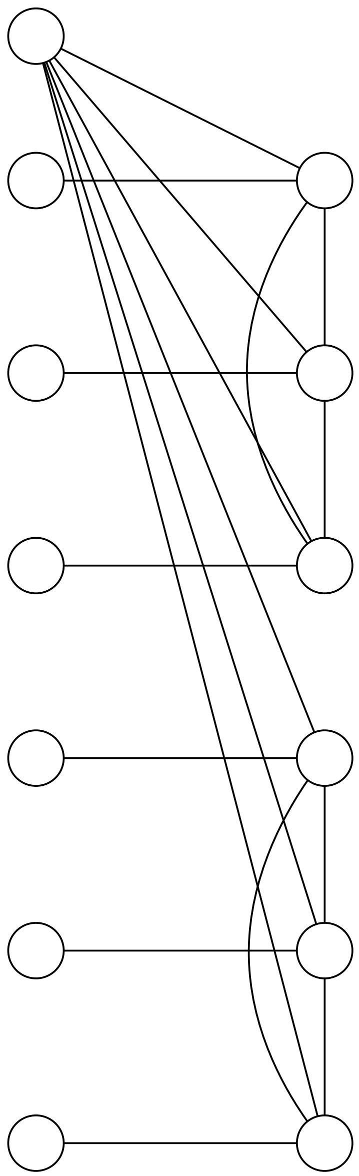
$k = 9$



Lower bound for computing the diameter

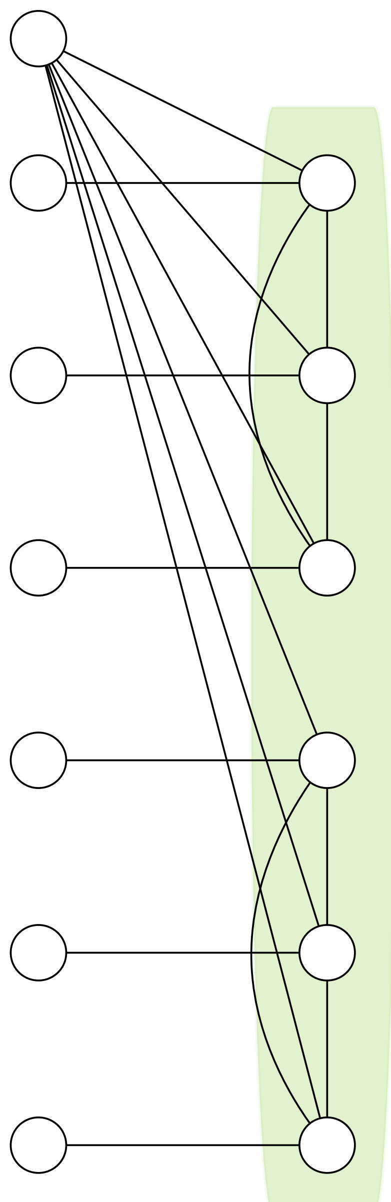
1
0
0
0
1
1
0
1
0

$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

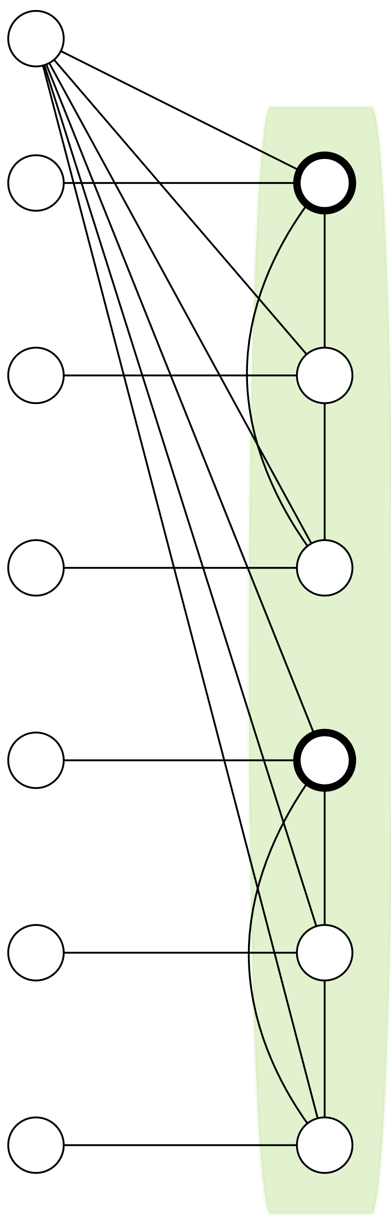


$k = 9$

Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

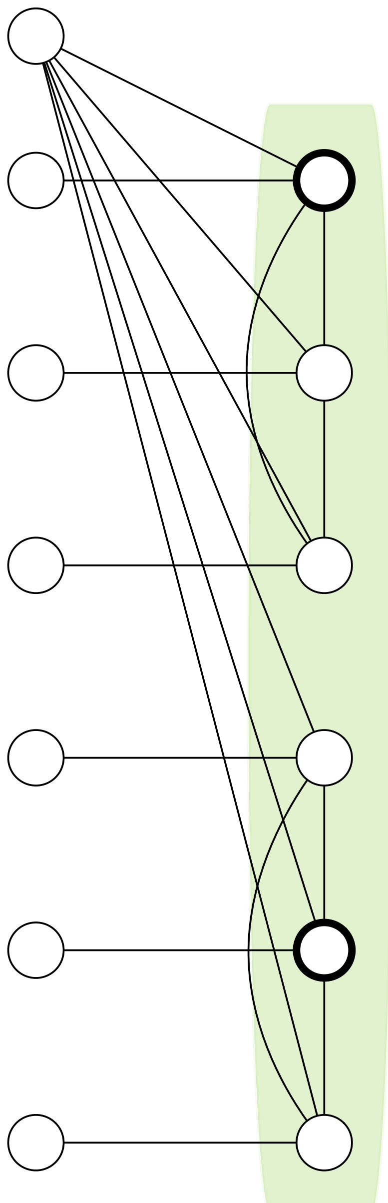
$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

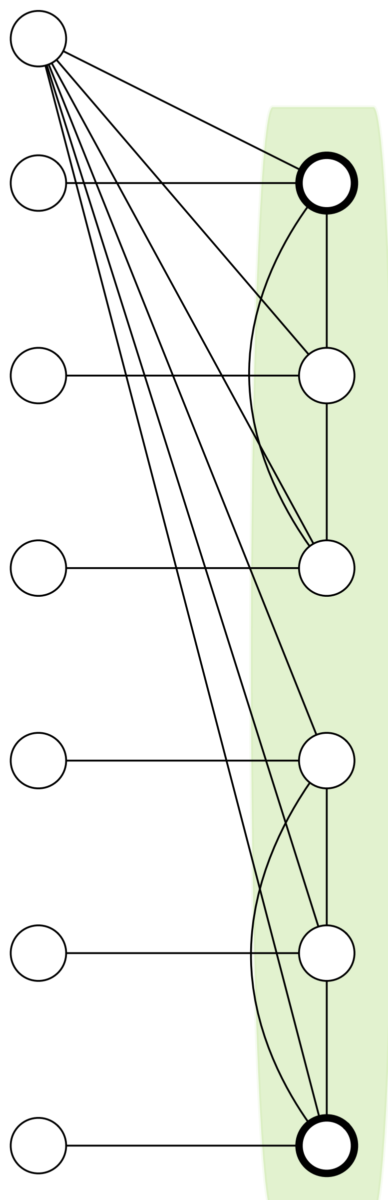
$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

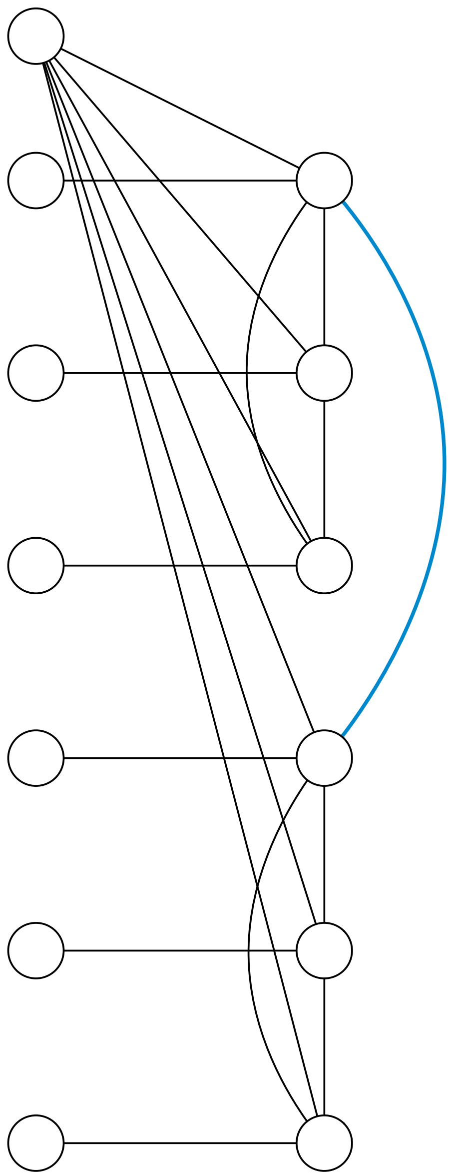
$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

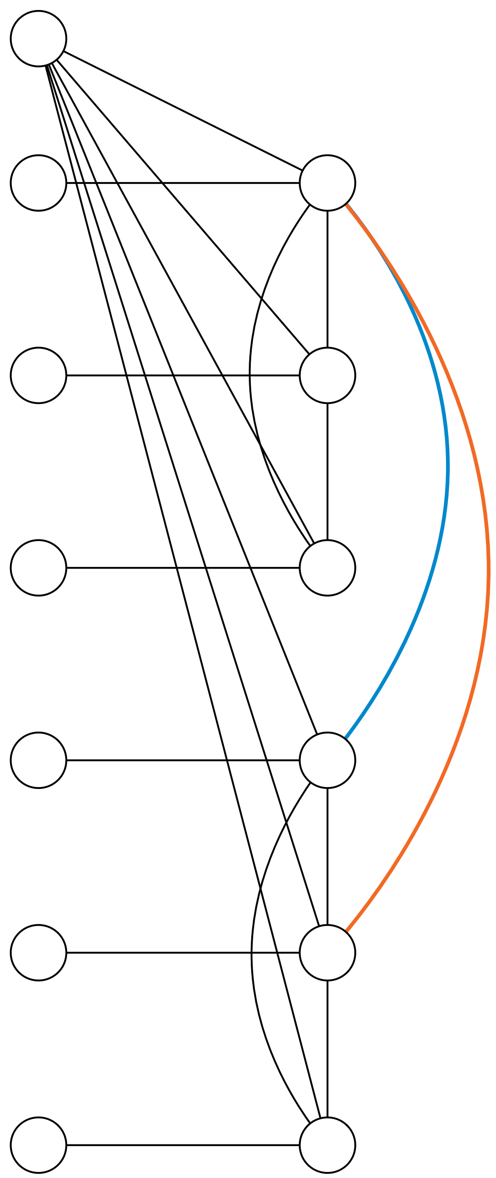
$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

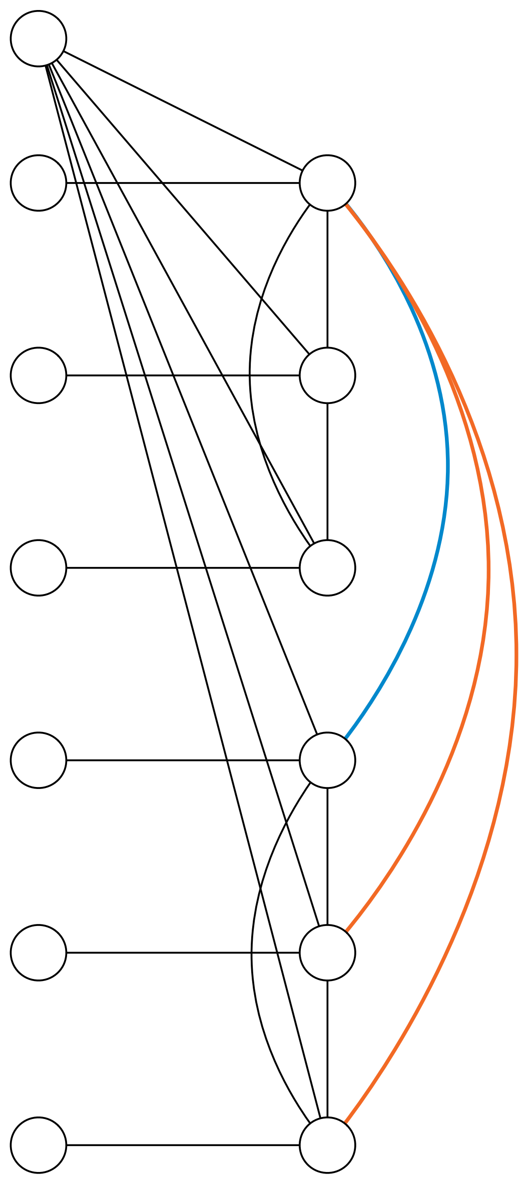
$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

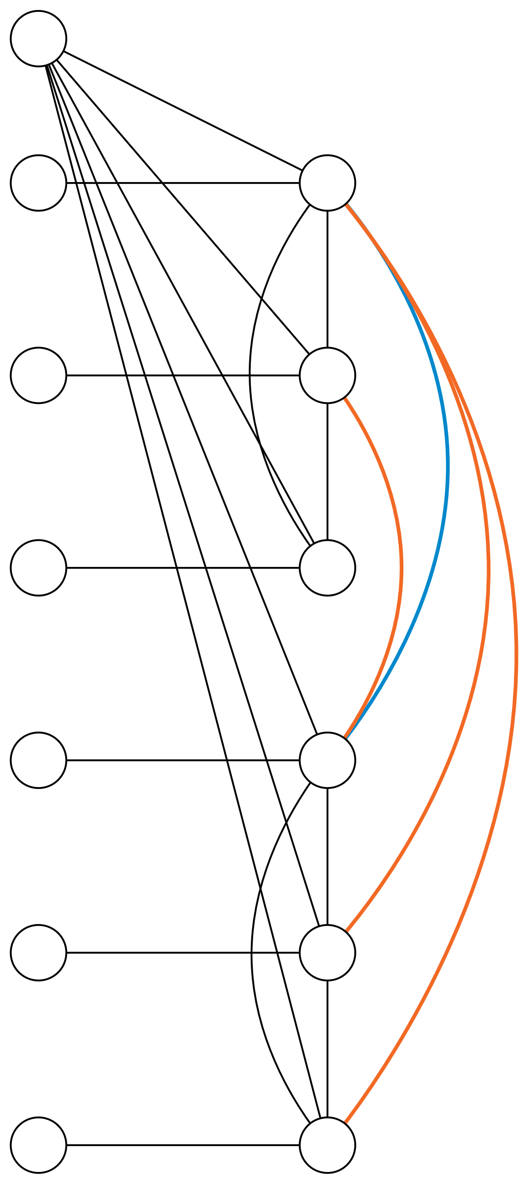
$k = 9$



Lower bound for computing the diameter

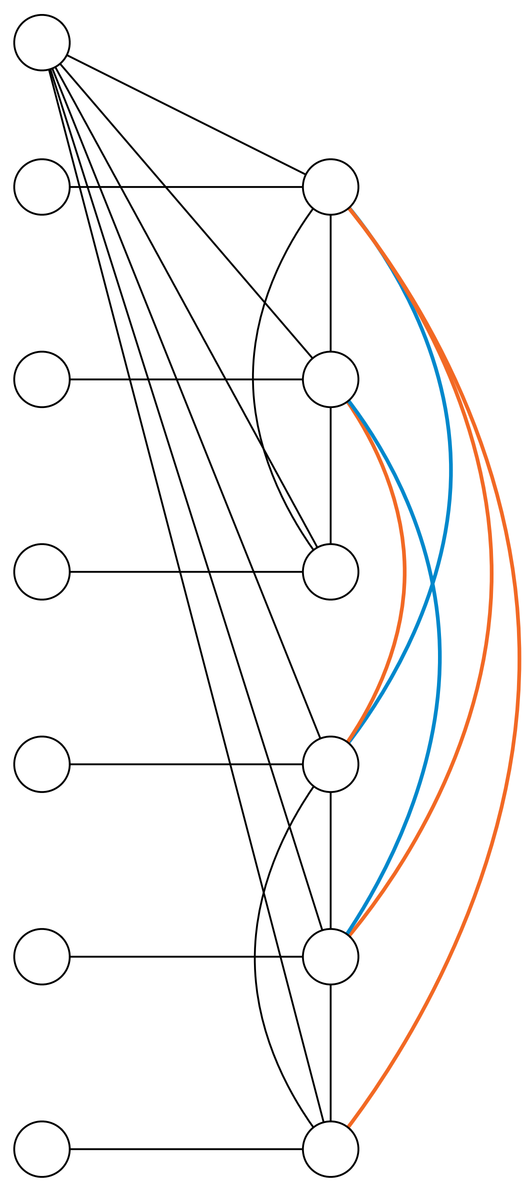
1
0
0
0
1
1
0
1
0

$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

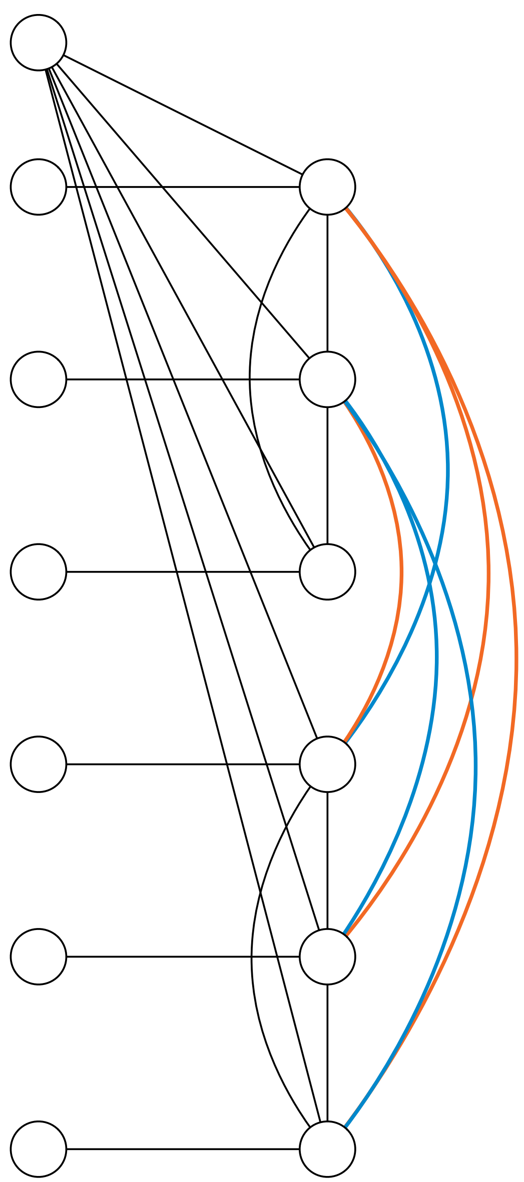


$k = 9$

Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

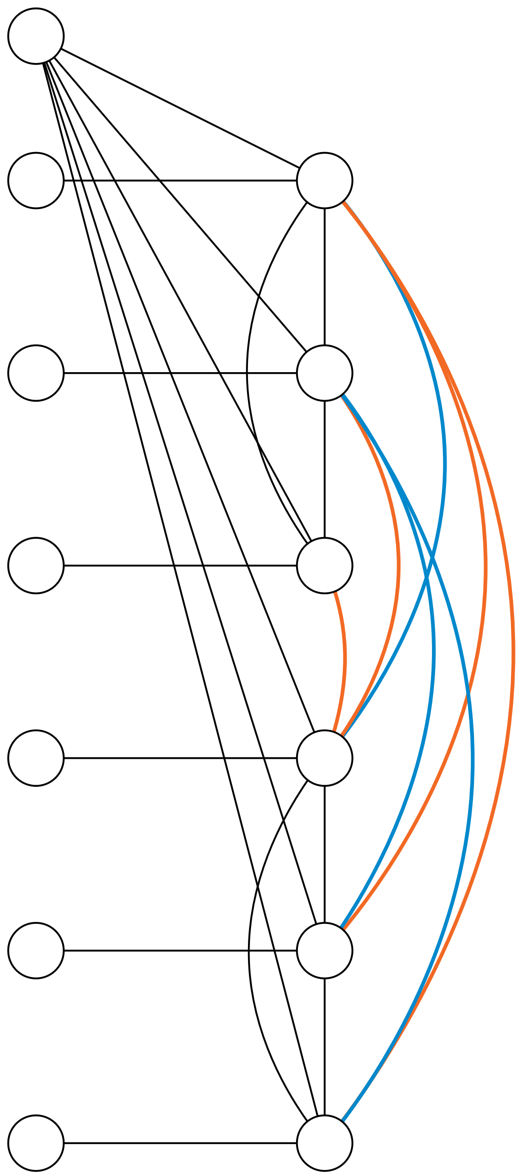
$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

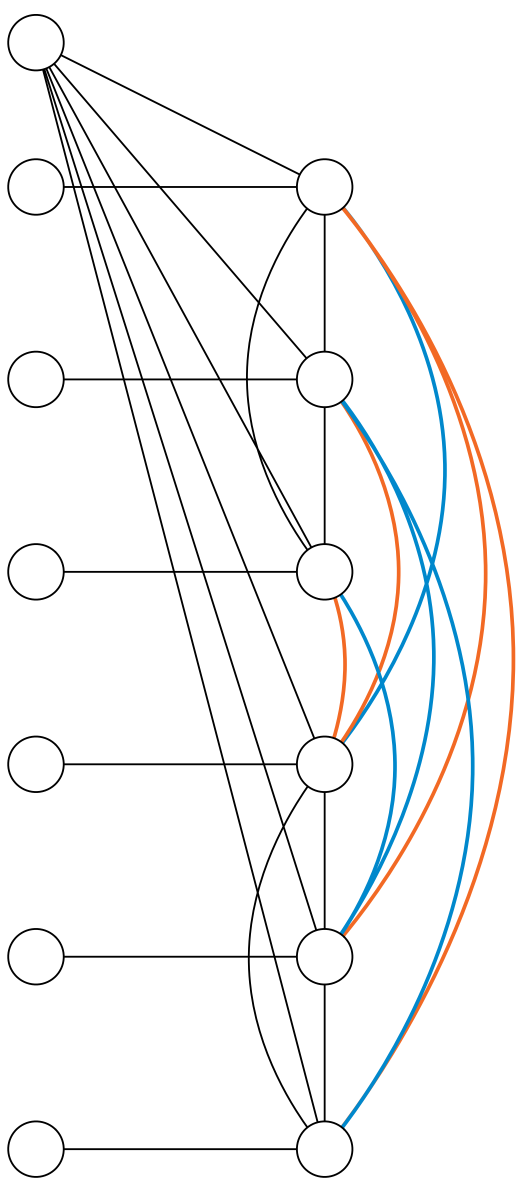
$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

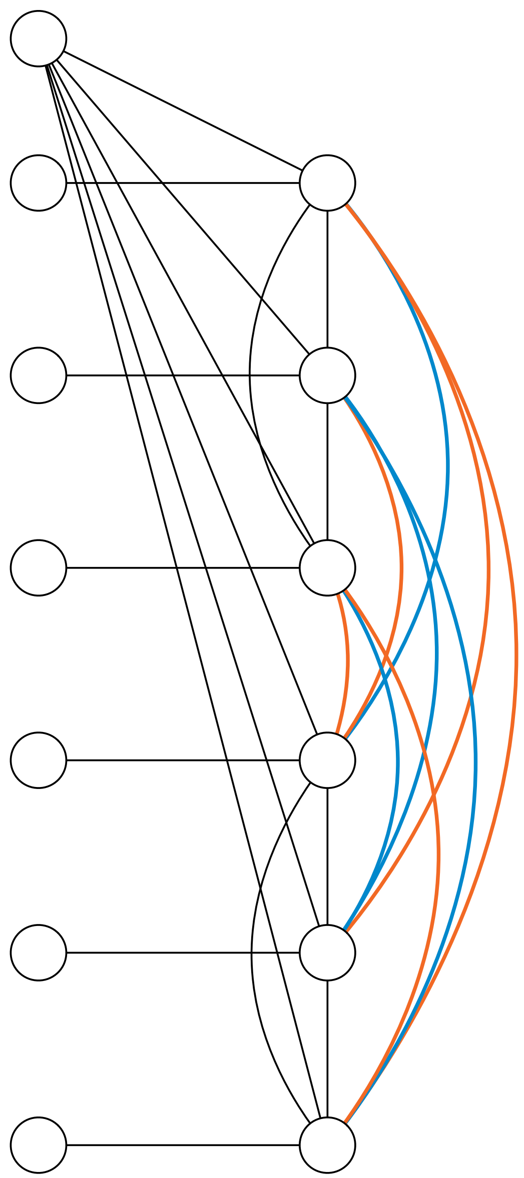
$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

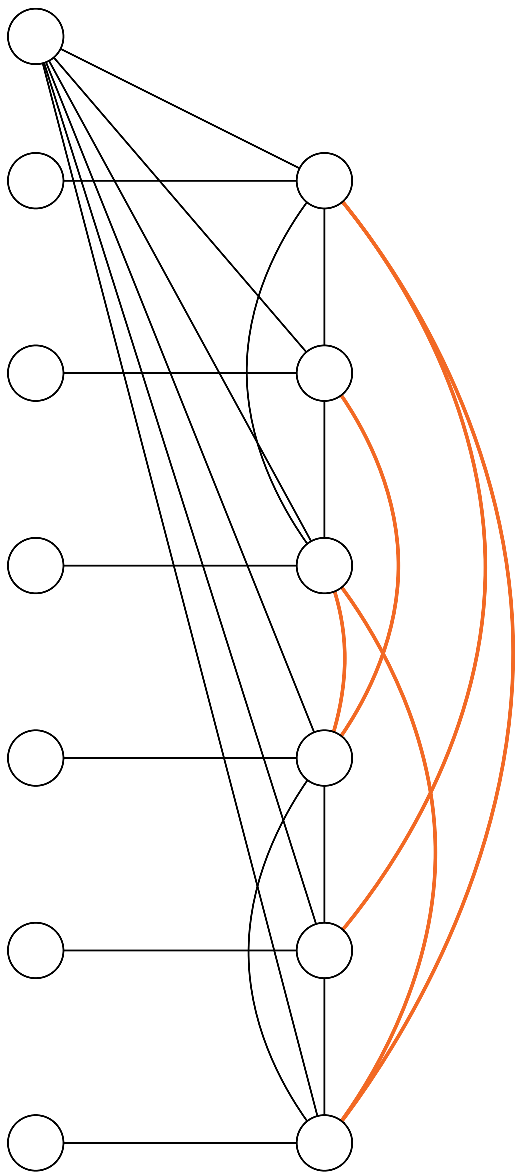
$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

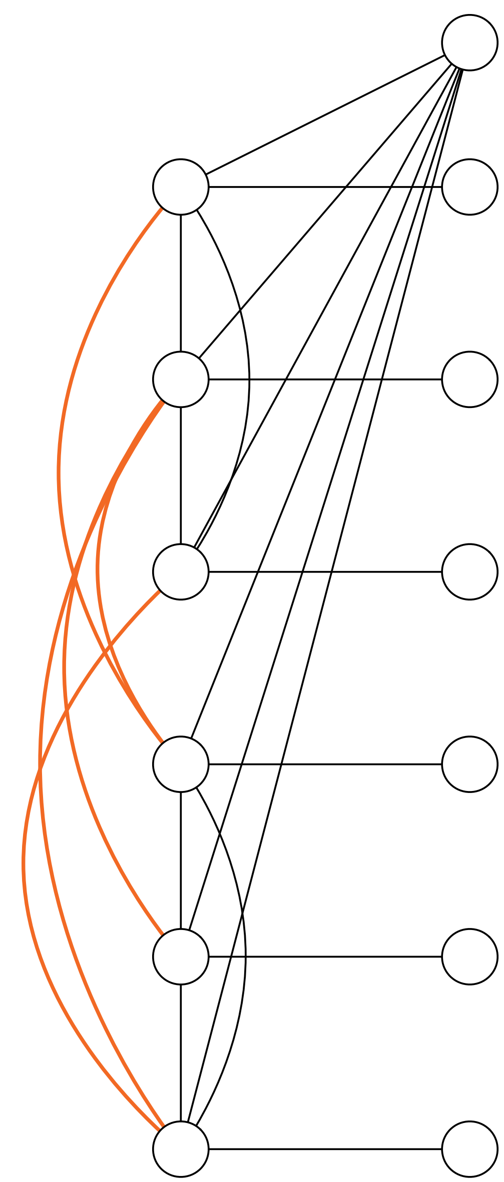
$k = 9$



Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

$k = 9$



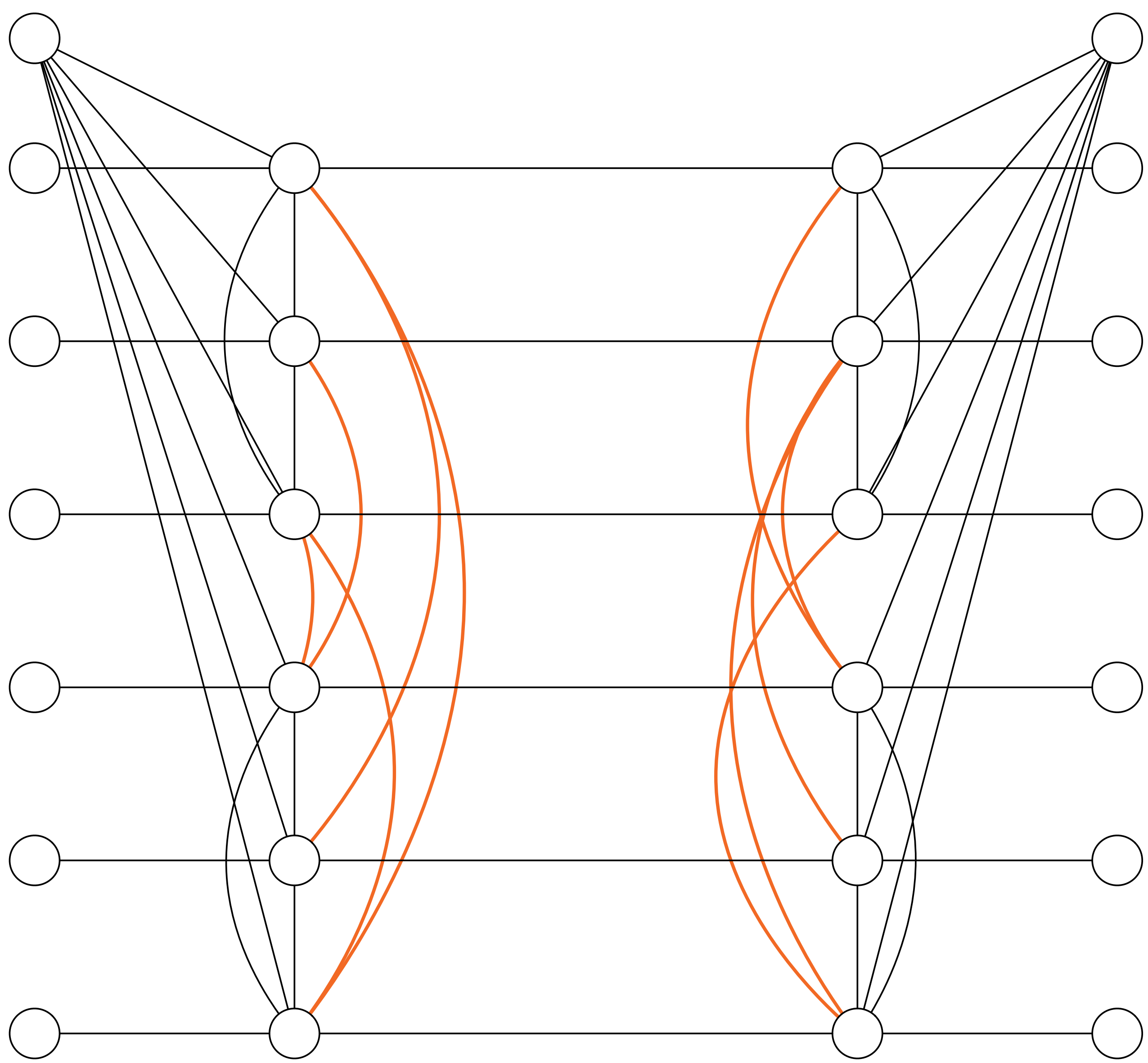
0
1
1
0
0
0
1
1
0

$k = 9$

Lower bound for computing the diameter

1
0
0
0
1
1
0
1
0

$k = 9$



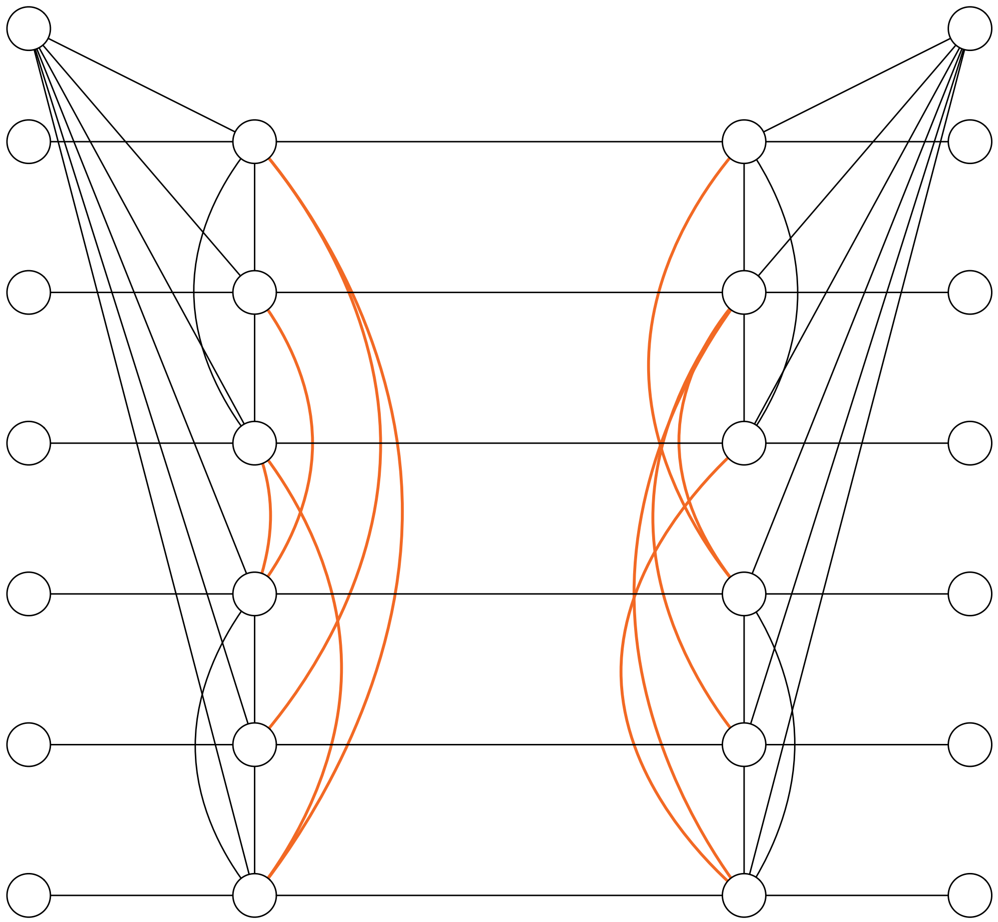
0
1
1
0
0
0
1
1
0

$k = 9$

Lower bound for computing the diameter

Diameter = 4 if the **sets are disjoint**
otherwise **diameter ≥ 5**

1
0
0
0
1
1
0
1
0

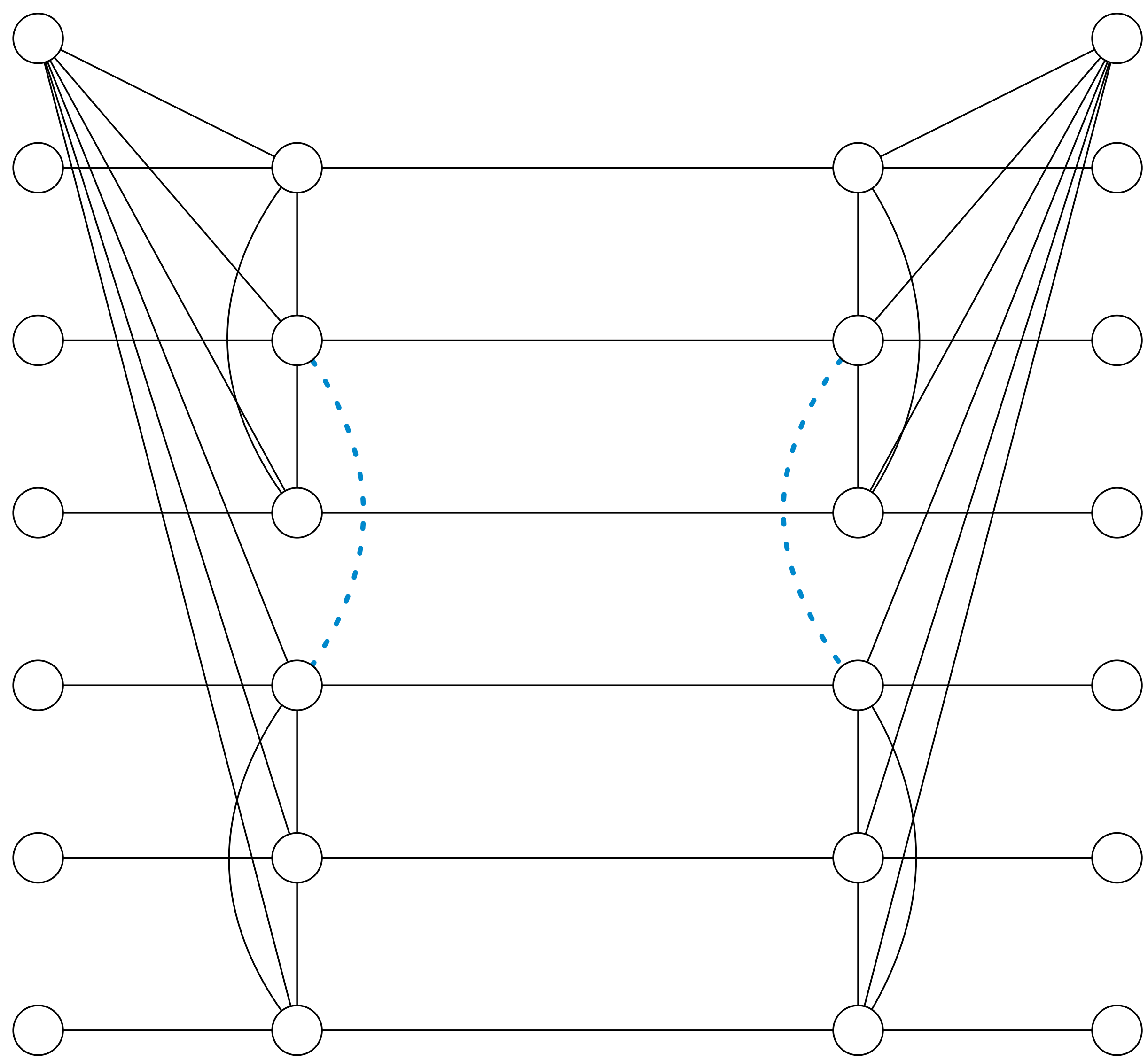


0
1
1
0
0
0
1
1
0

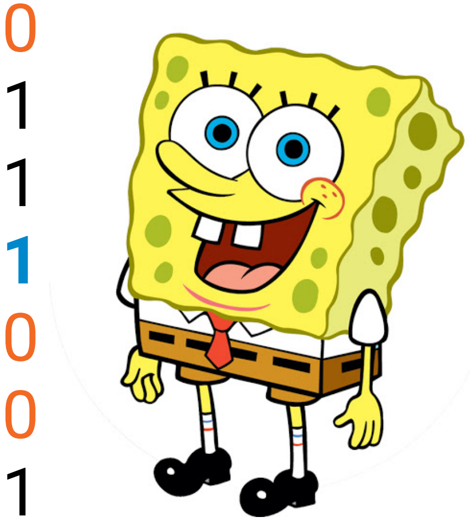
Lower bound for computing the diameter

Sets are **not** disjoint, **diameter ≥ 5**

1
0
0
1
1
1
0
1
0



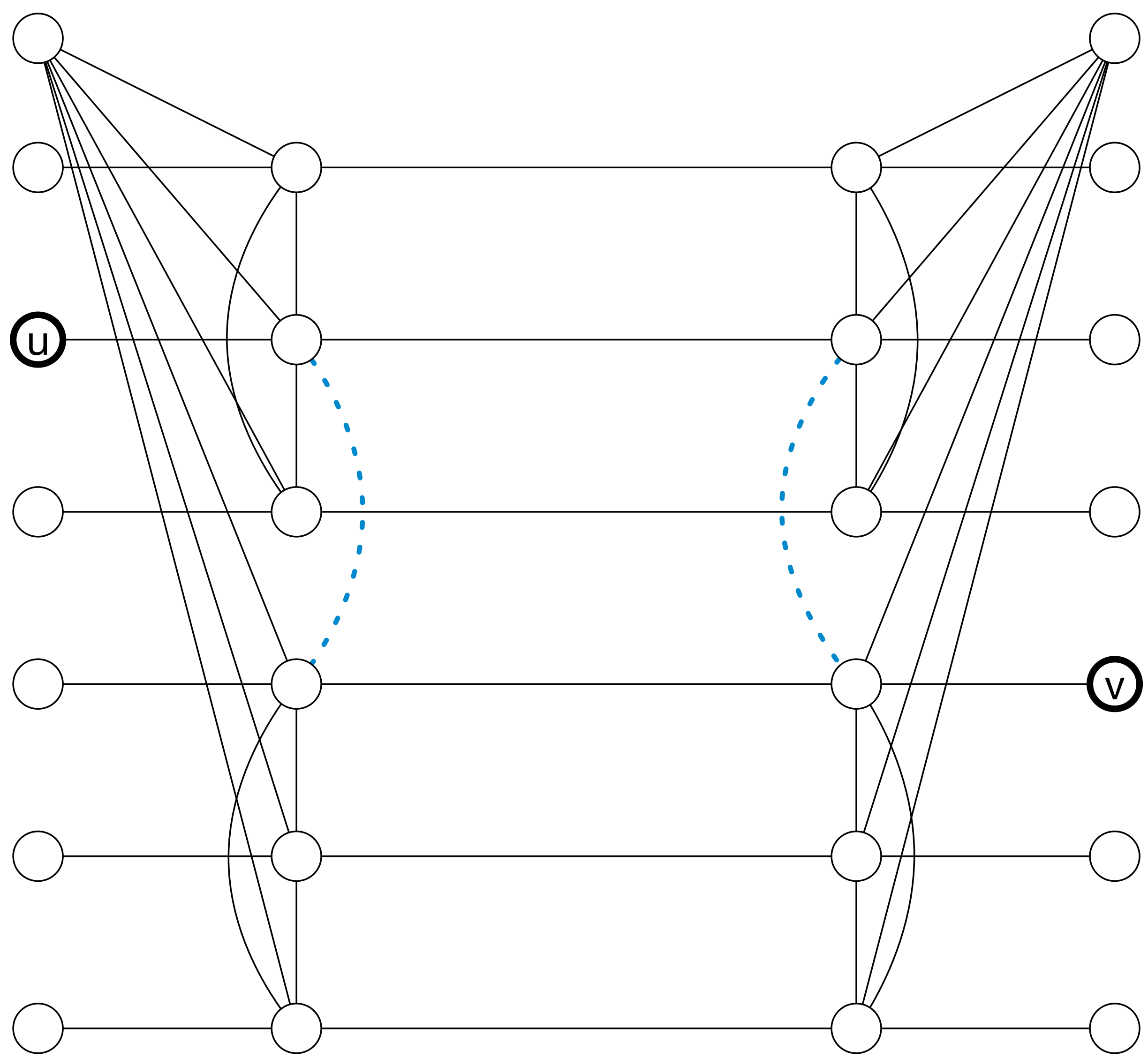
0
1
1
1
0
0
1
0
0



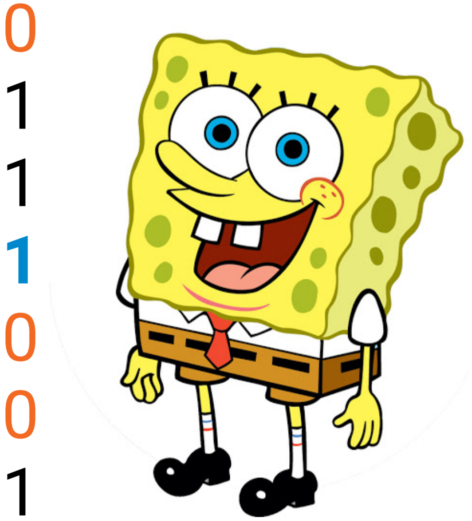
Lower bound for computing the diameter

Sets are **not** disjoint, **diameter ≥ 5**

1
0
0
1
1
1
0
1
0



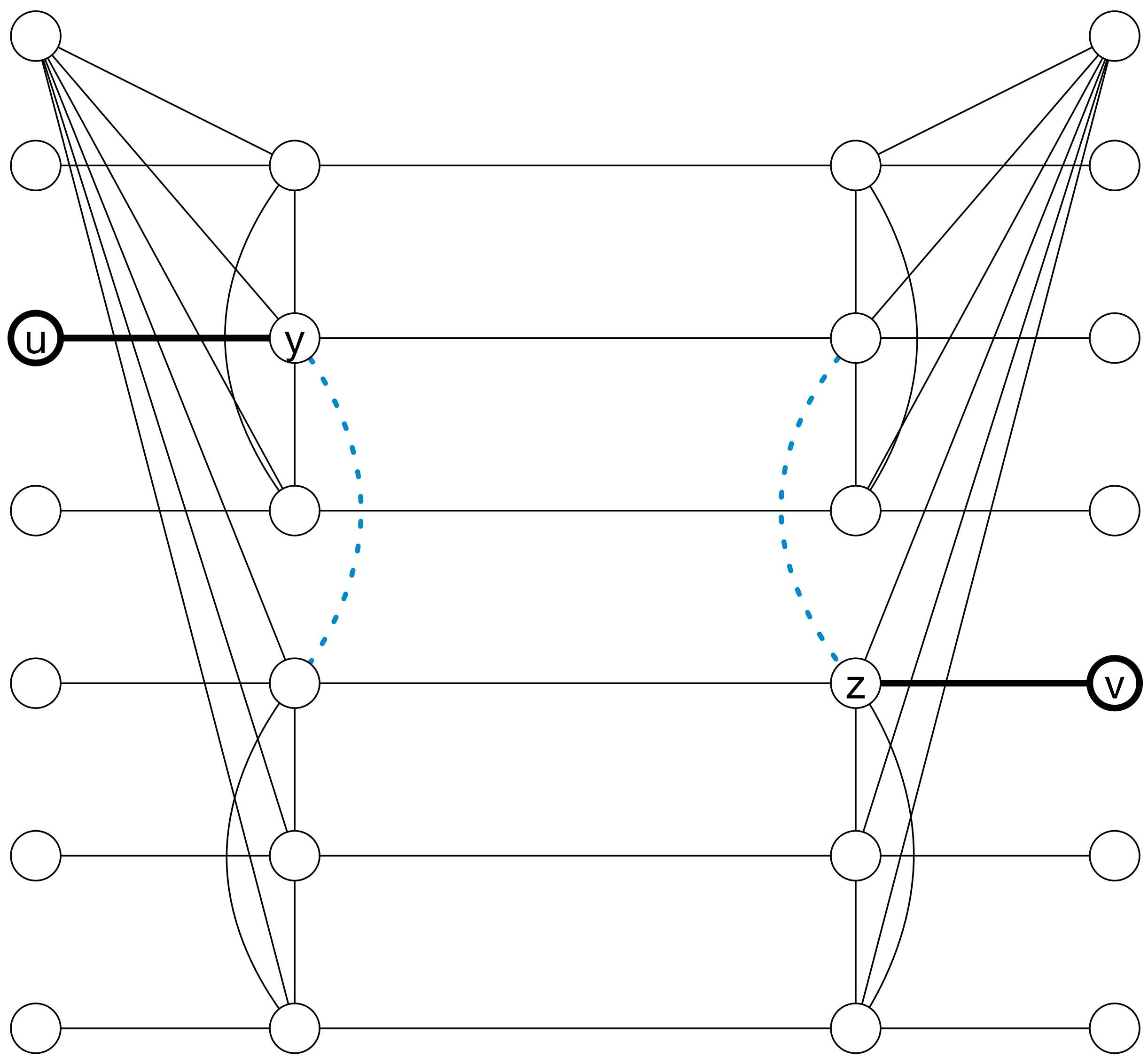
0
1
1
1
0
0
1
0
0



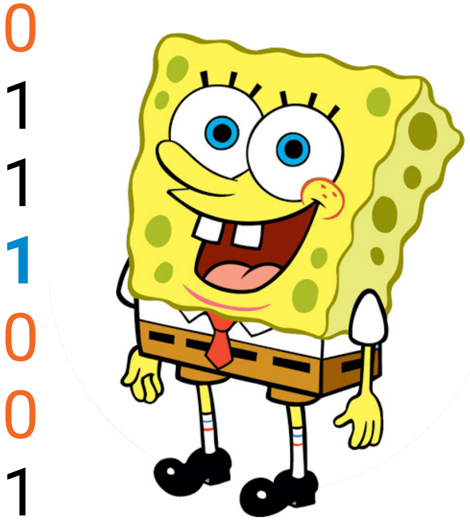
Lower bound for computing the diameter

Sets are **not** disjoint, **diameter ≥ 5**

1
0
0
1
1
1
0
1
0



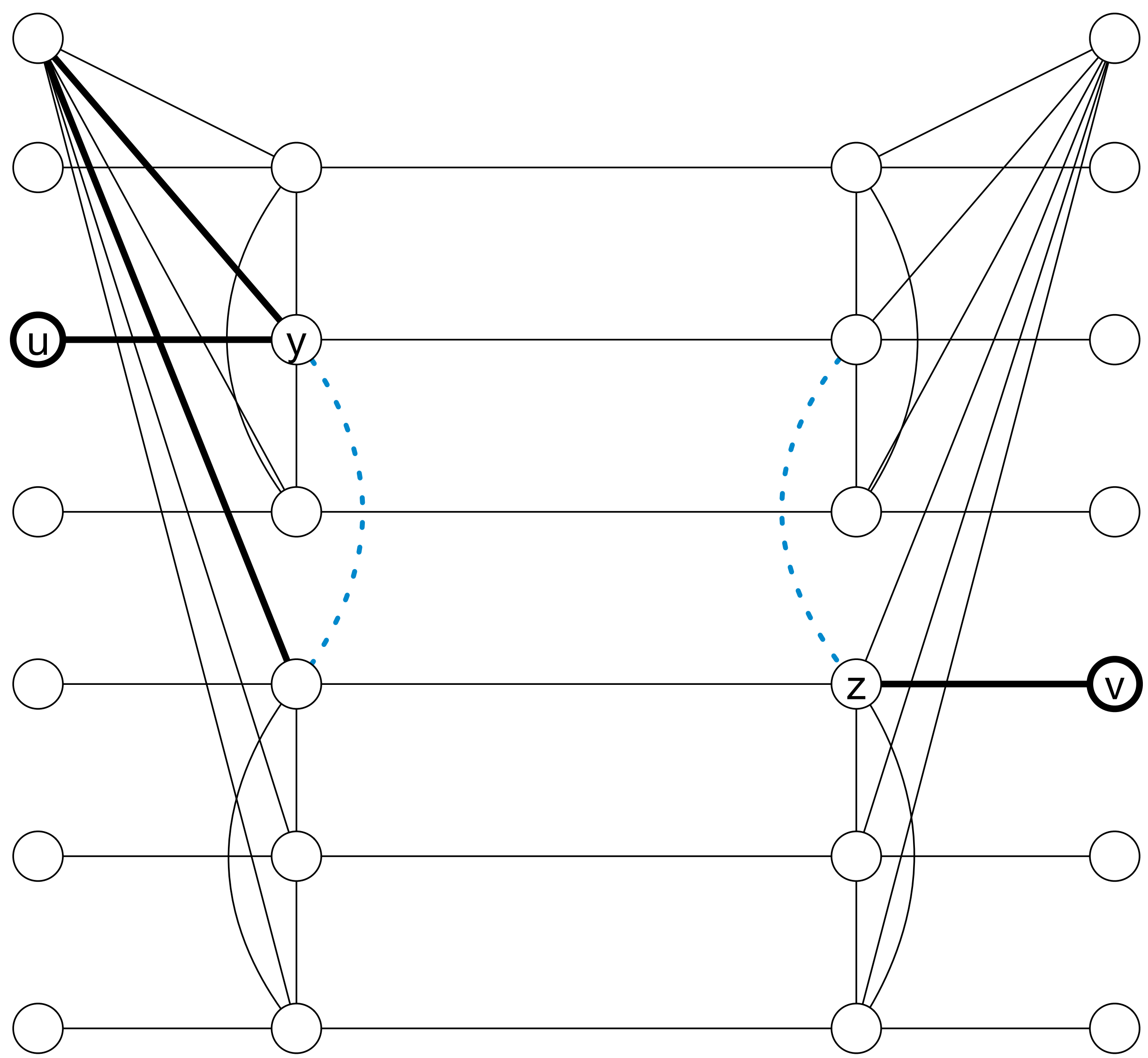
0
1
1
1
0
0
1
0
0



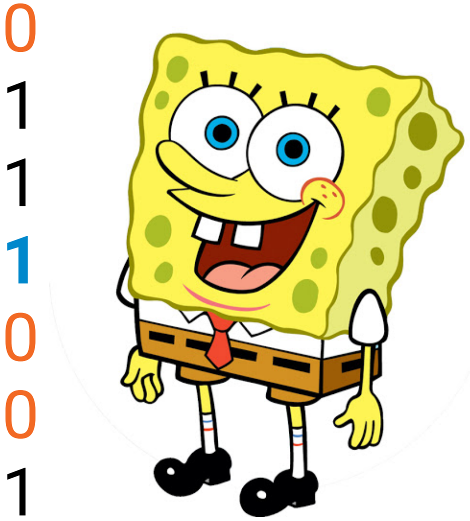
Lower bound for computing the diameter

Sets are **not** disjoint, **diameter ≥ 5**

1
0
0
1
1
1
0
1
0



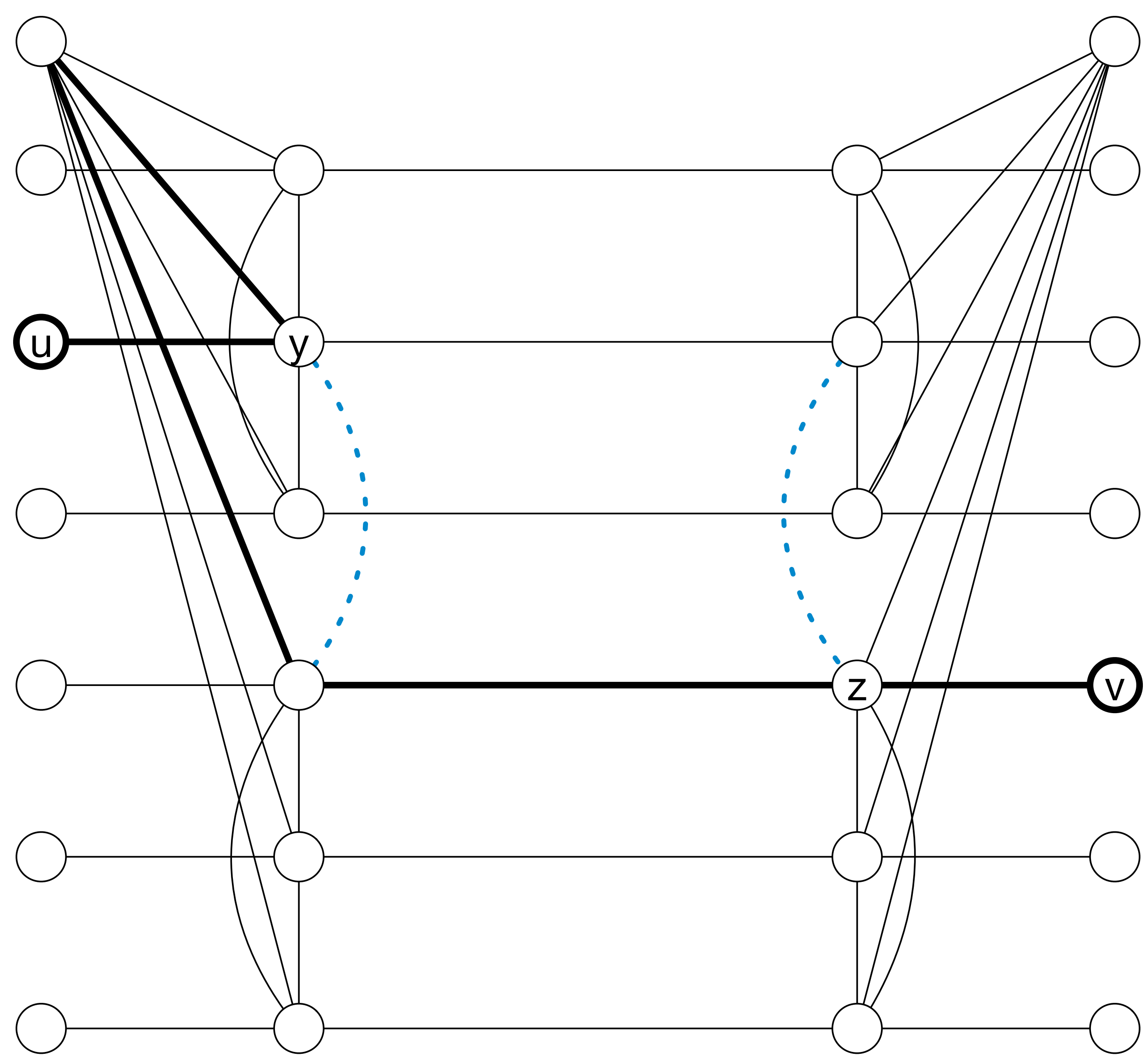
0
1
1
1
0
0
1
0
0



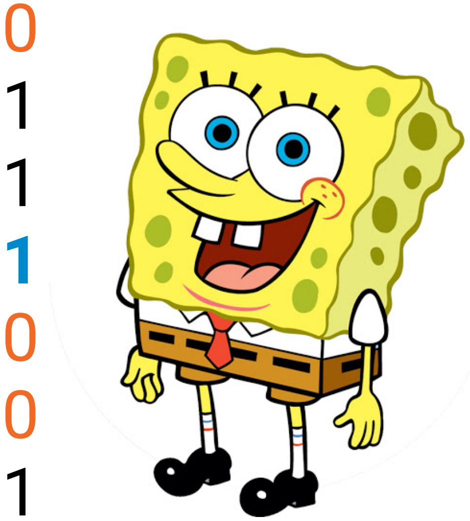
Lower bound for computing the diameter

Sets are **not** disjoint, **diameter ≥ 5**

1
0
0
1
1
1
0
1
0



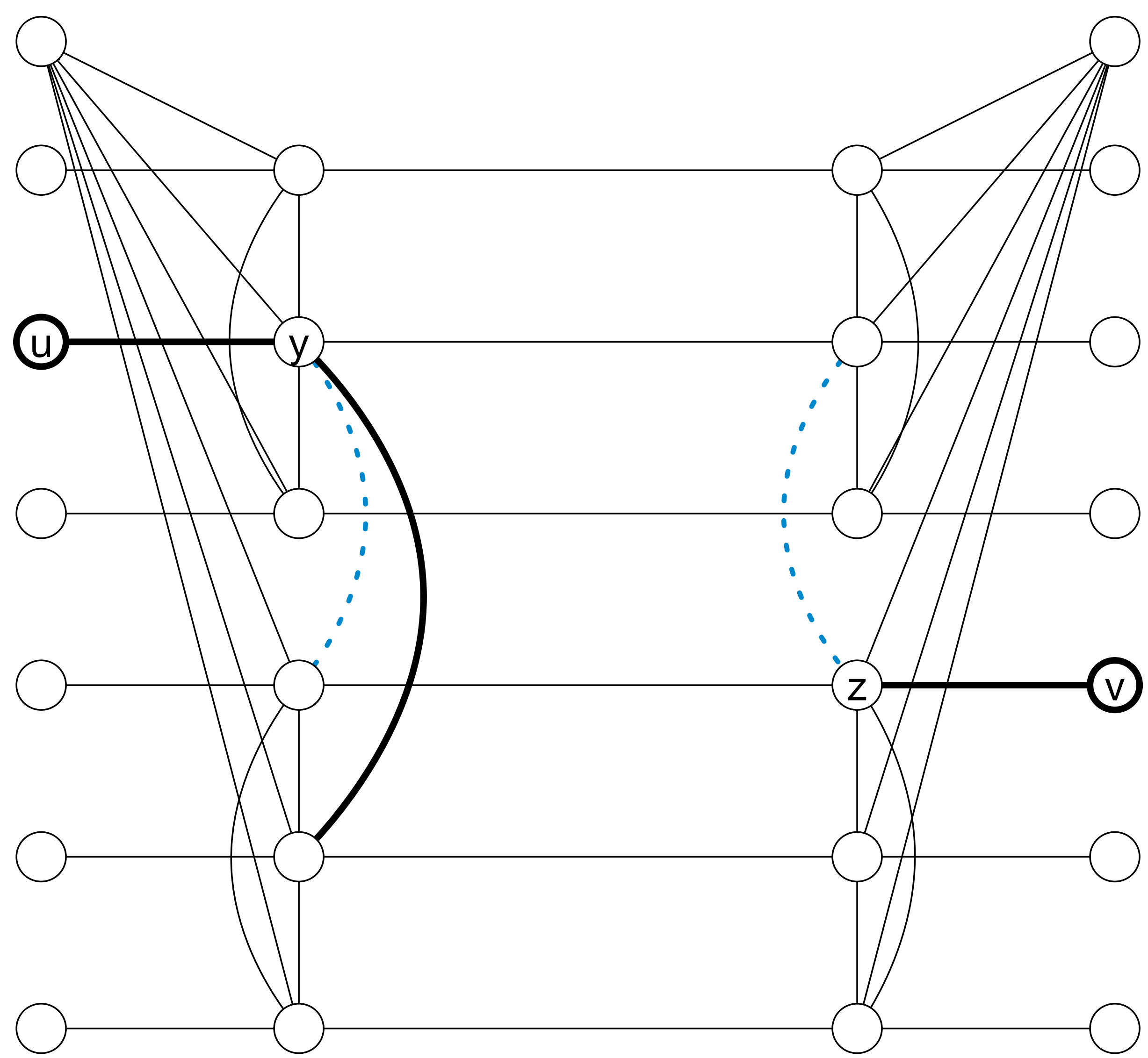
0
1
1
1
0
0
1
0
0



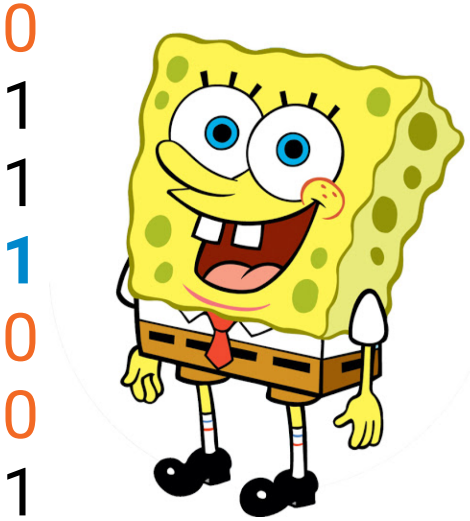
Lower bound for computing the diameter

Sets are **not** disjoint, **diameter ≥ 5**

1
0
0
1
1
1
0
1
0

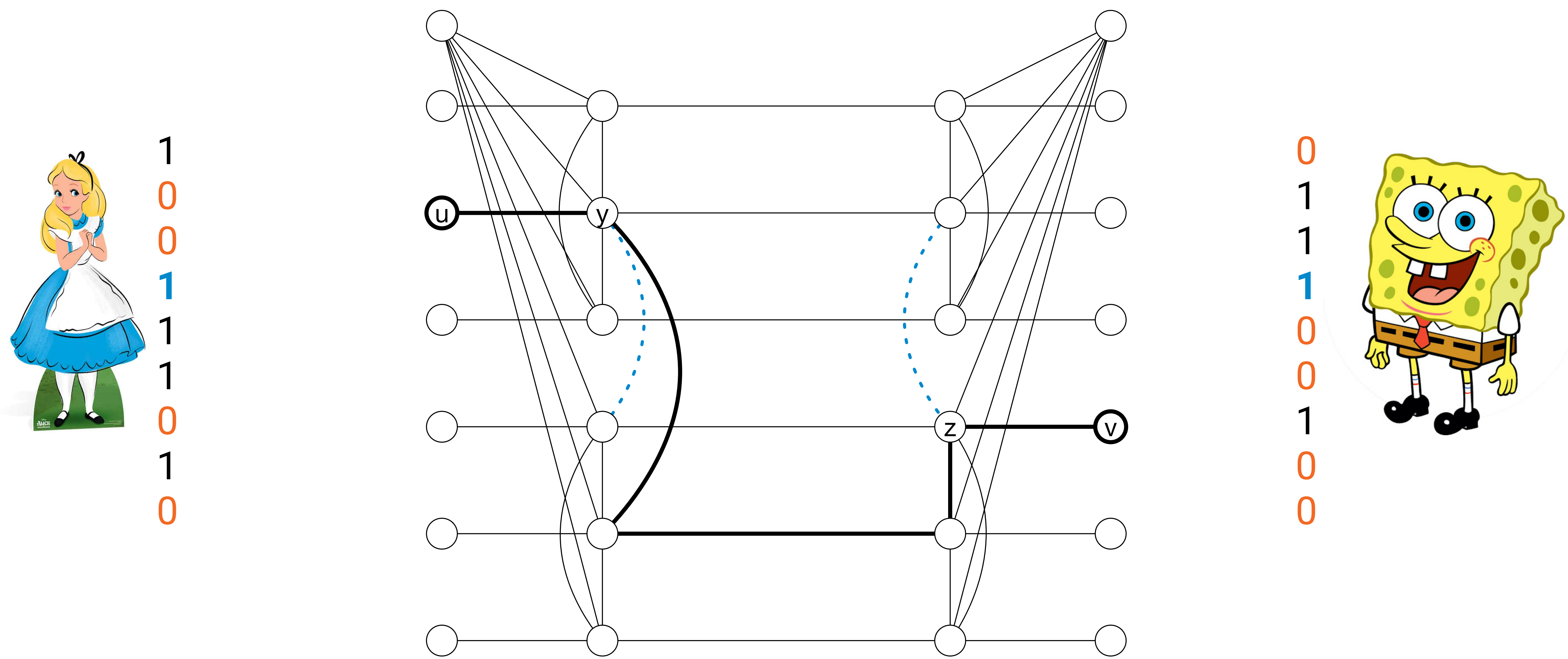


0
1
1
1
0
0
1
0
0



Lower bound for computing the diameter

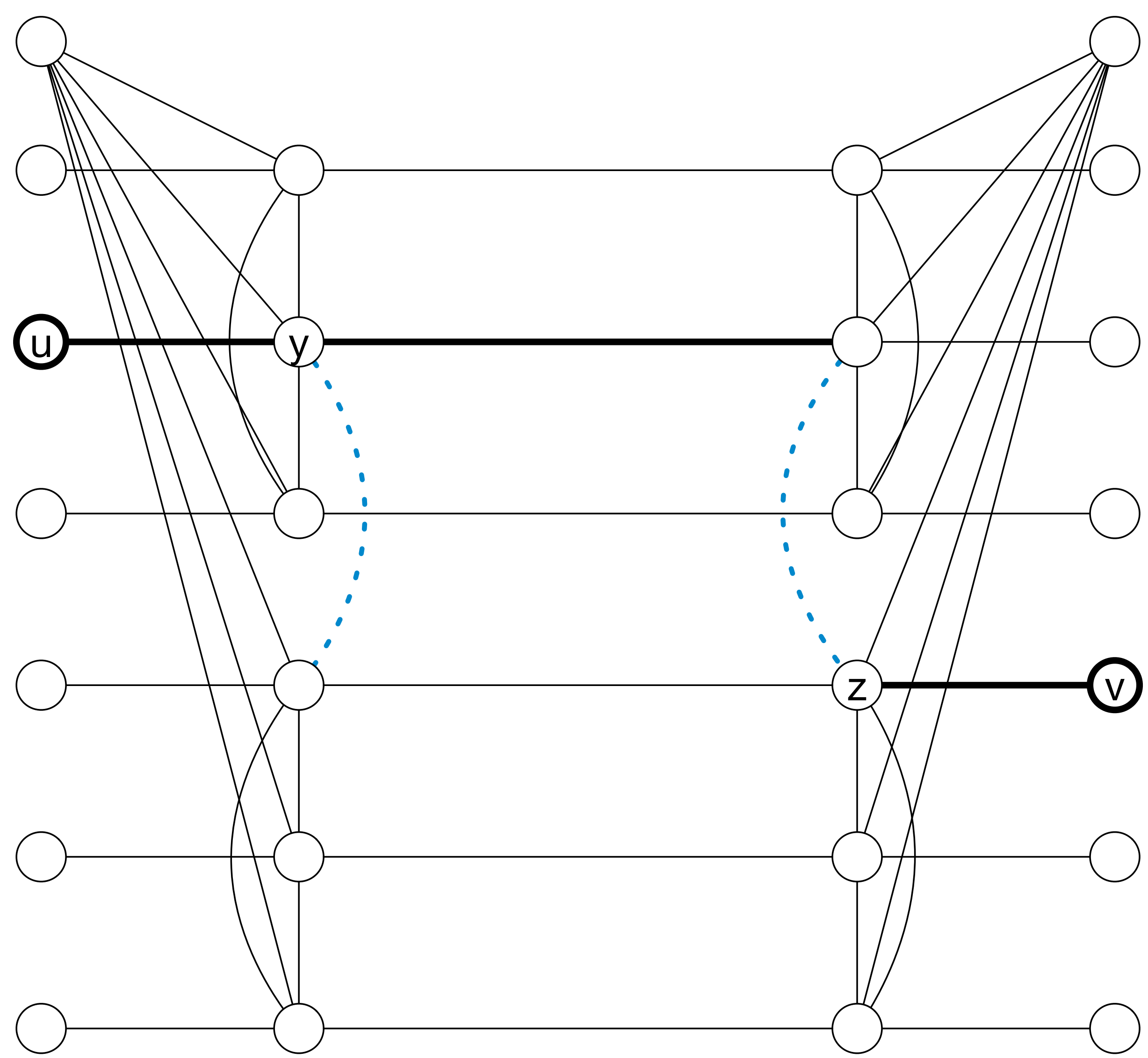
Sets are **not** disjoint, **diameter ≥ 5**



Lower bound for computing the diameter

Sets are **not** disjoint, **diameter ≥ 5**

1
0
0
1
1
1
0
1
0

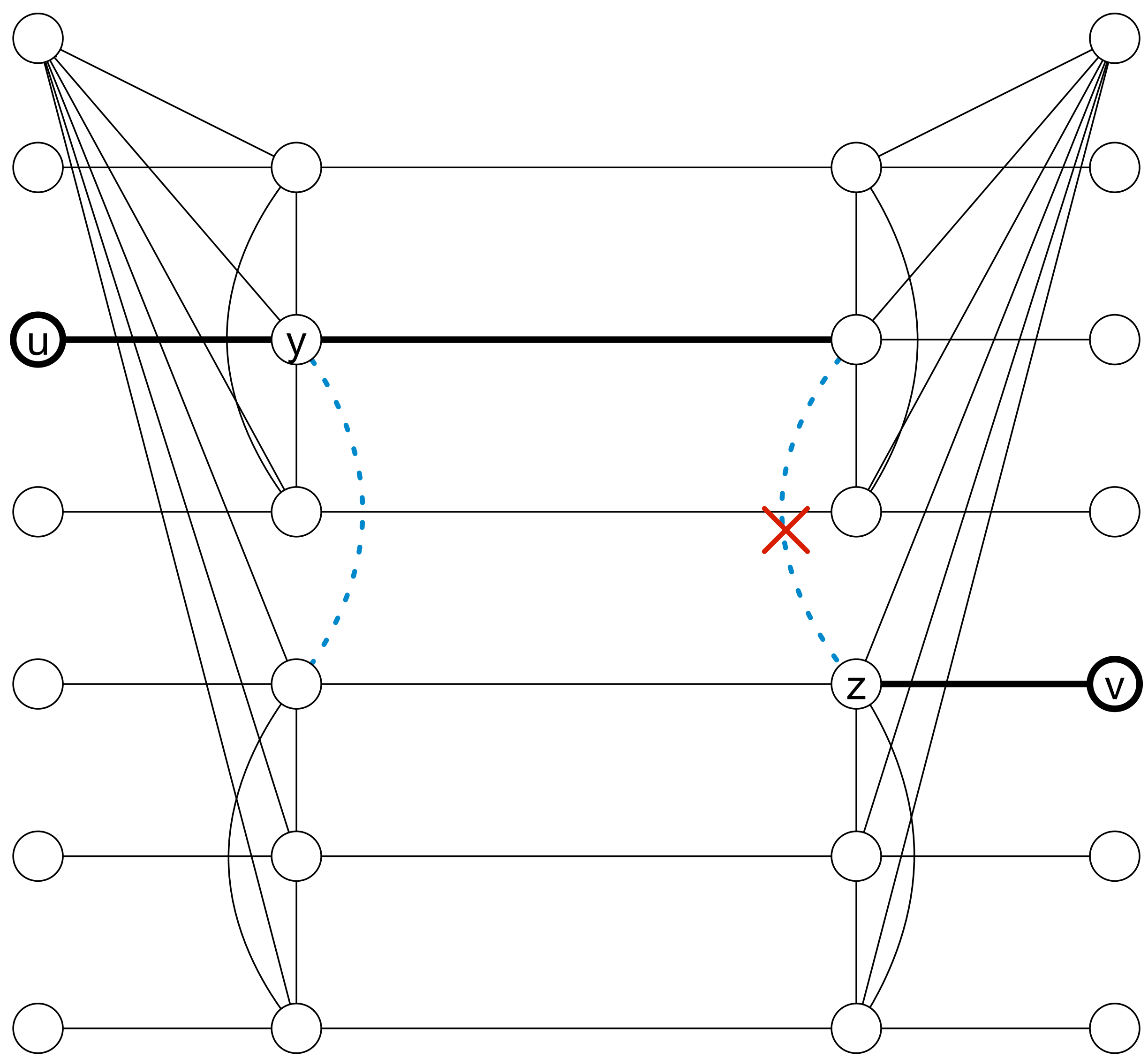


0
1
1
1
0
0
1
0
0

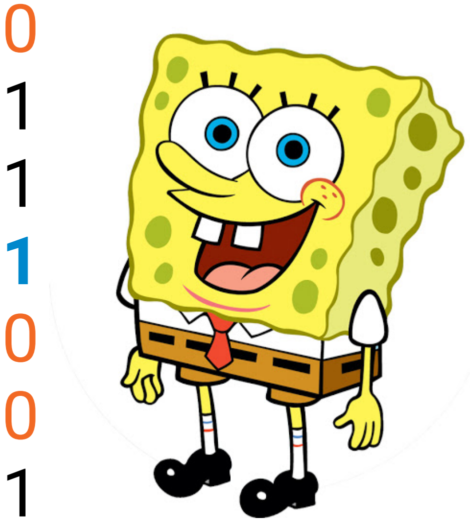
Lower bound for computing the diameter

Sets are **not** disjoint, **diameter ≥ 5**

1
0
0
1
1
1
0
1
0



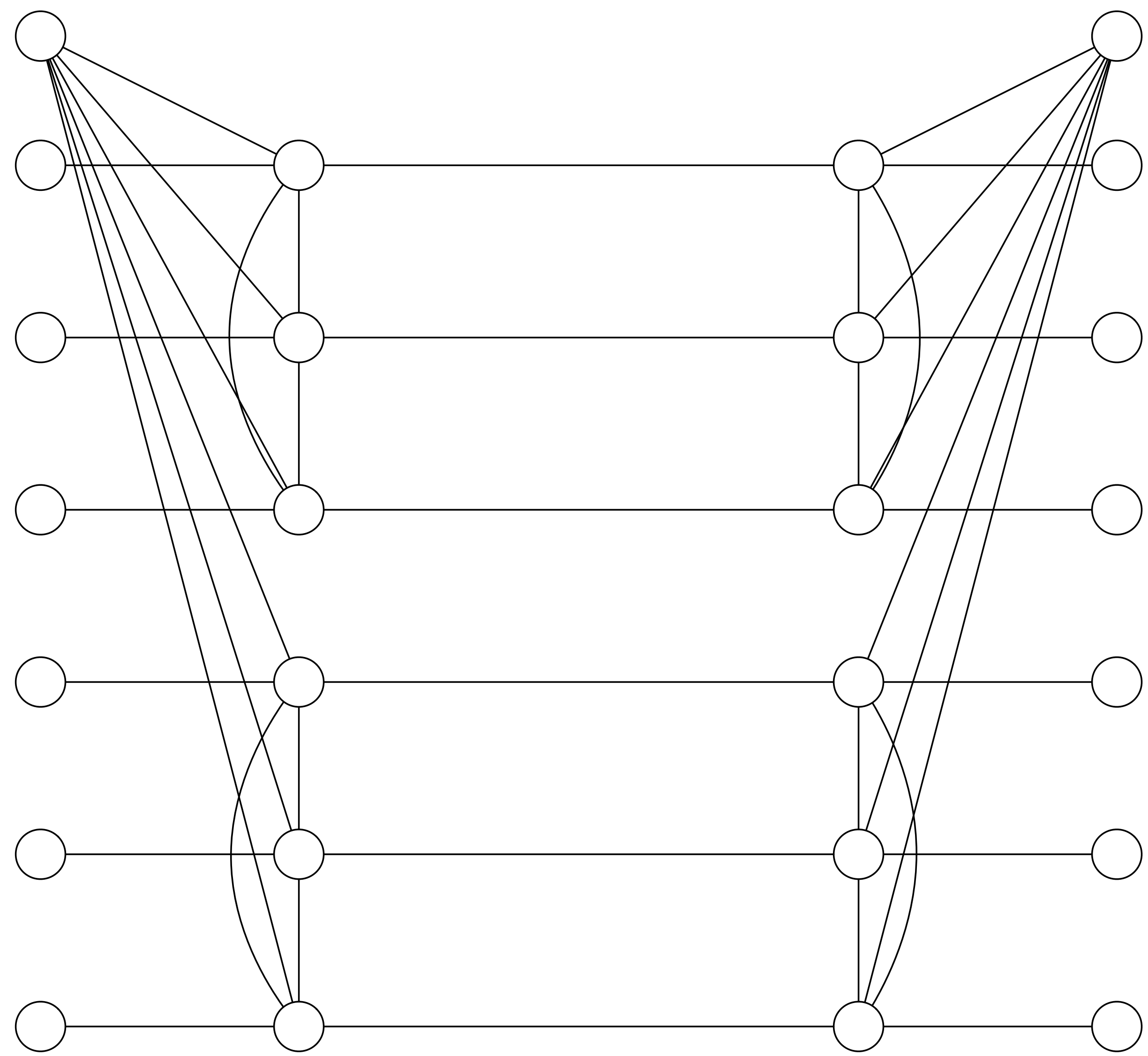
0
1
1
1
0
0
1
0
0



Lower bound for computing the diameter

Sets are disjoint, **diameter = 4**

1
0
0
0
1
1
0
1
0

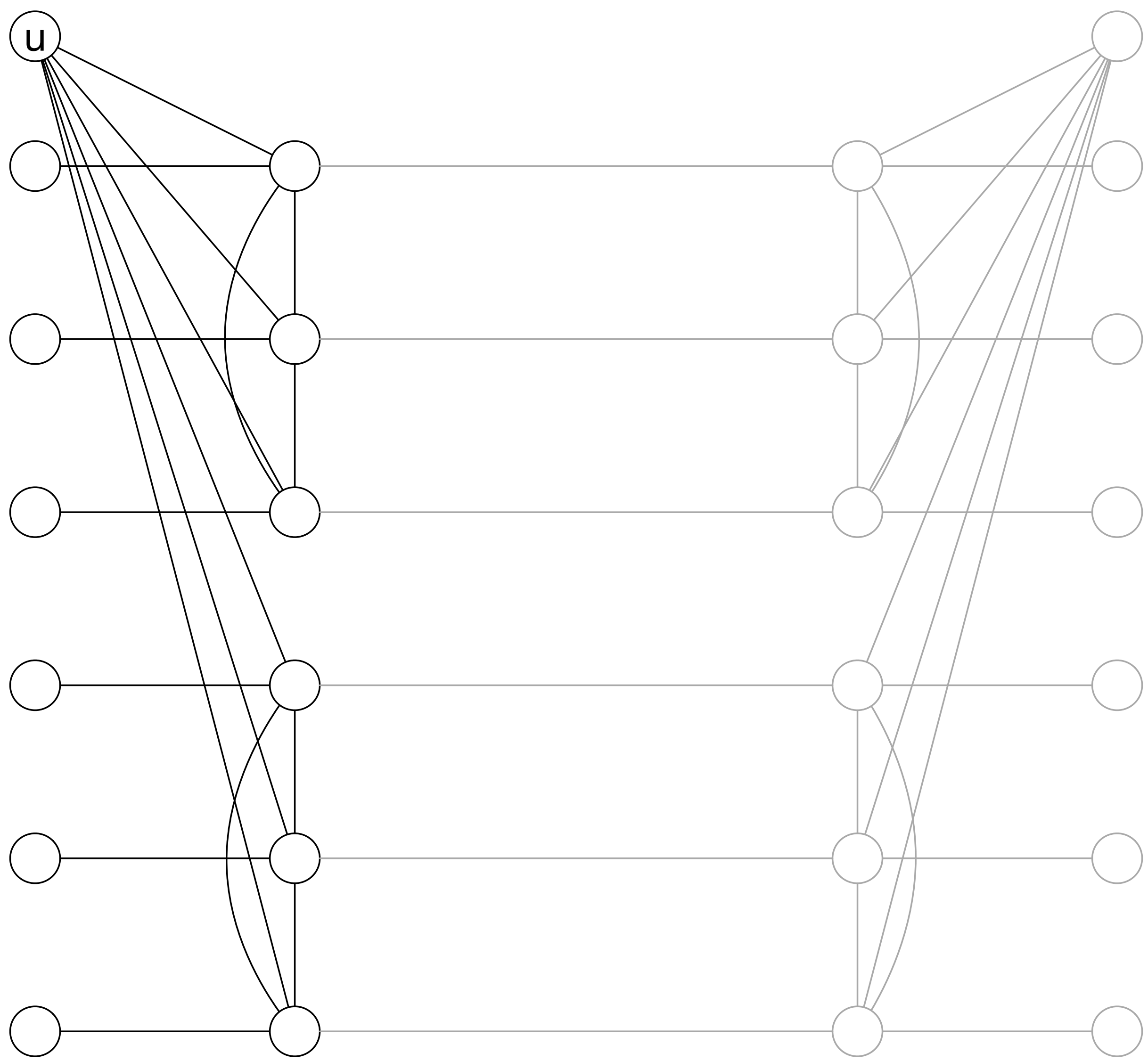


0
1
1
0
0
0
1
0
0

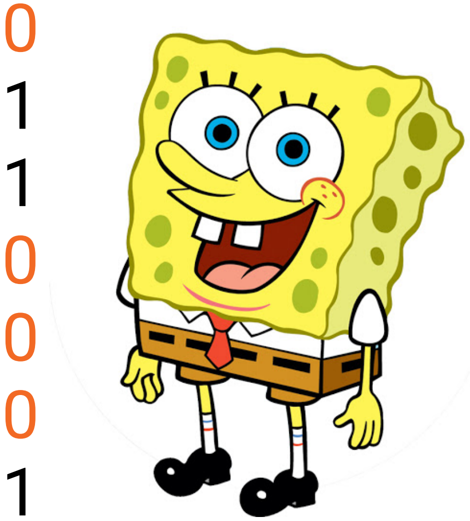
Lower bound for computing the diameter

Sets are disjoint, **diameter = 4**

1
0
0
0
1
1
0
1
0



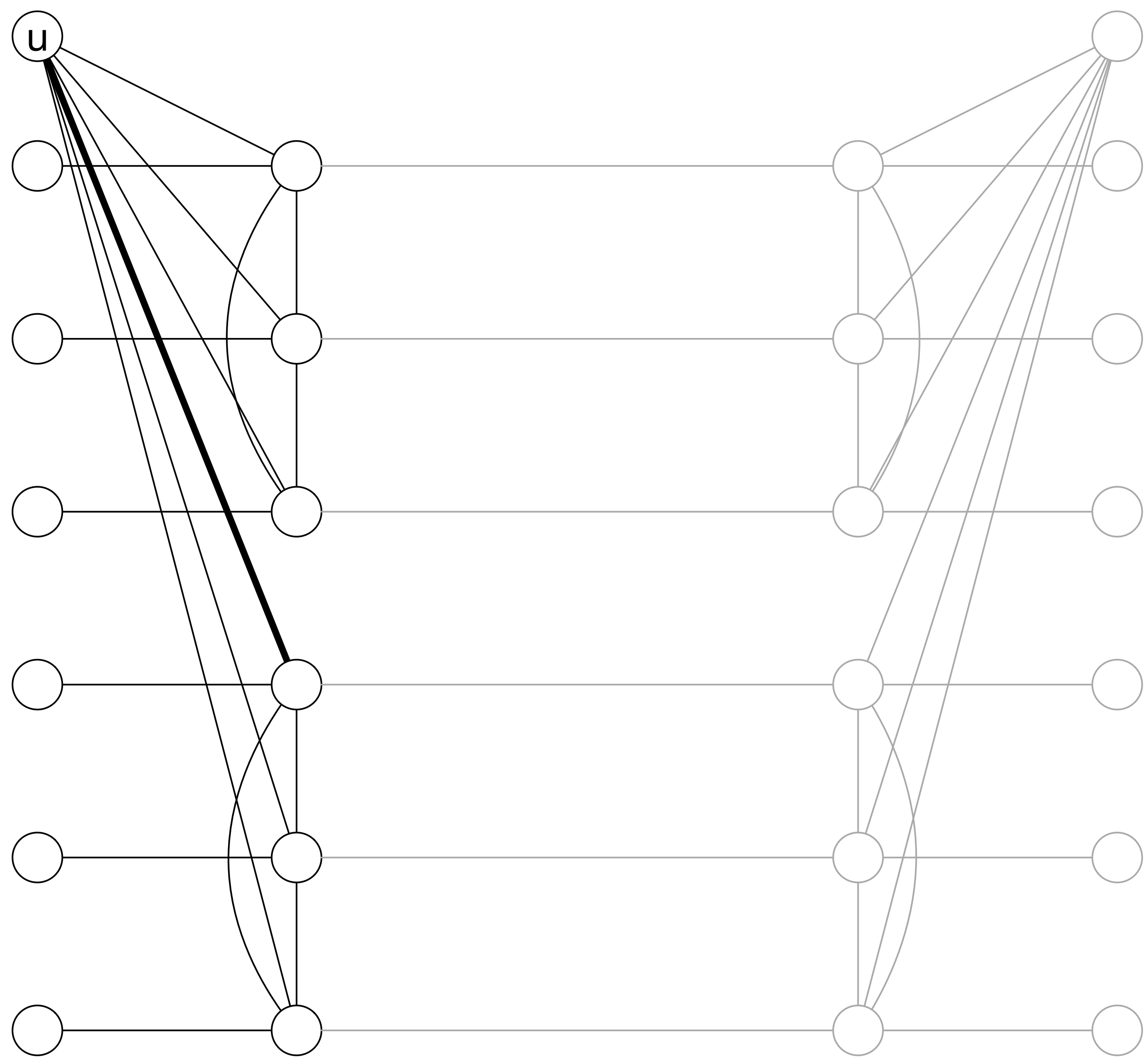
0
1
1
0
0
0
1
0
0



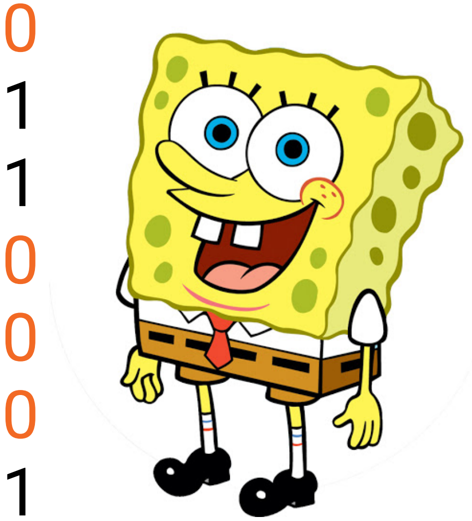
Lower bound for computing the diameter

Sets are disjoint, **diameter = 4**

1
0
0
0
1
1
0
1
0



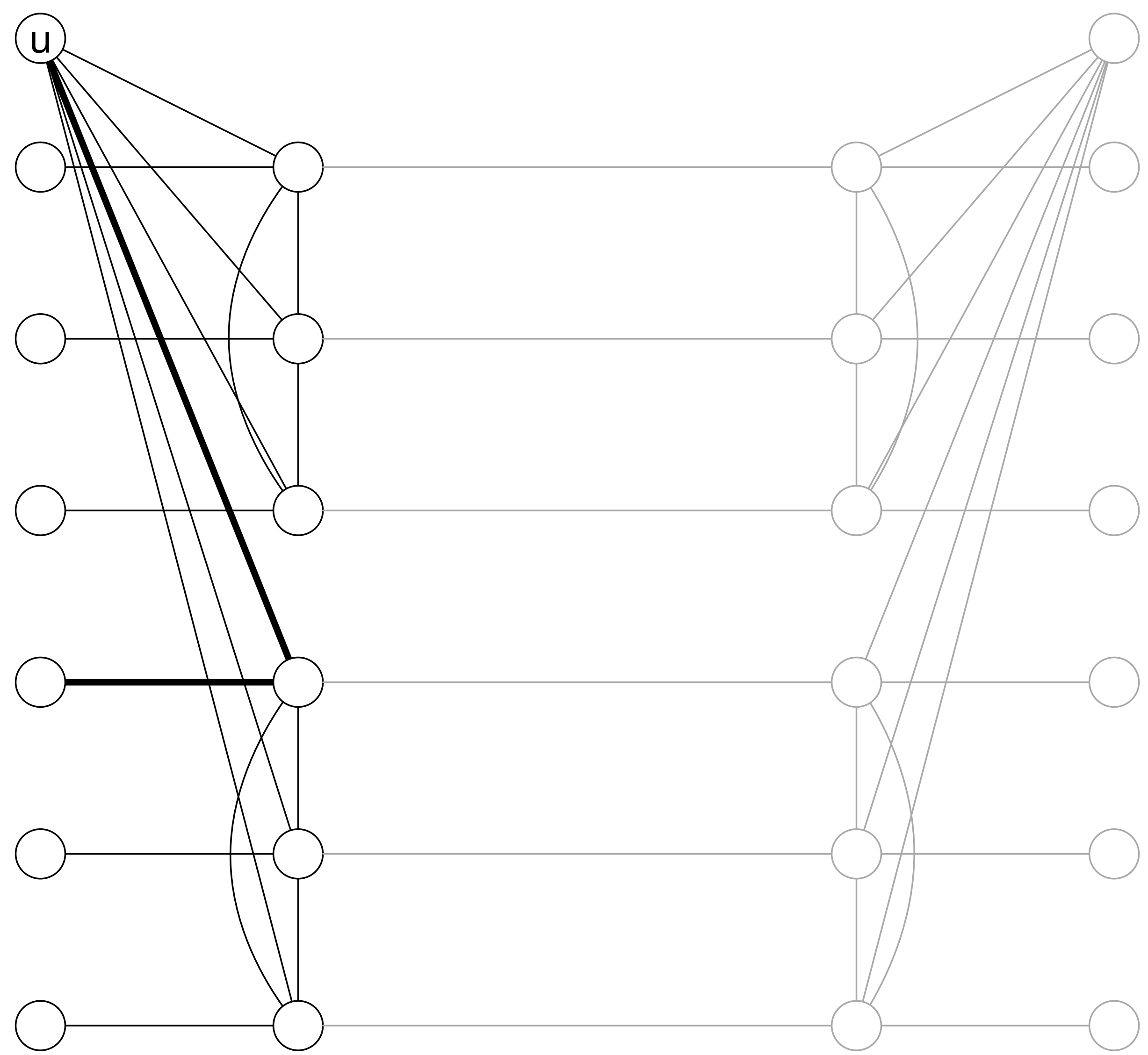
0
1
1
0
0
0
1
0
0



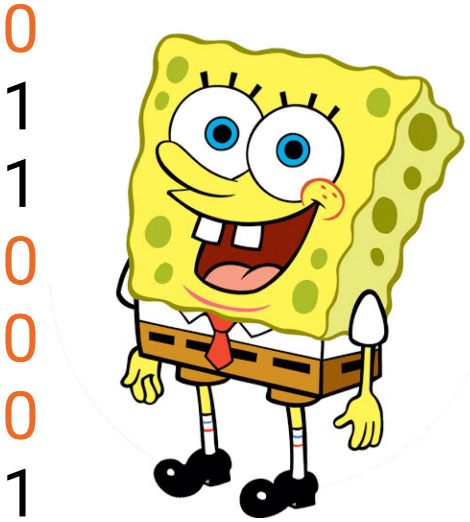
Lower bound for computing the diameter

Sets are disjoint, **diameter = 4**

1
0
0
0
1
1
0
1
0



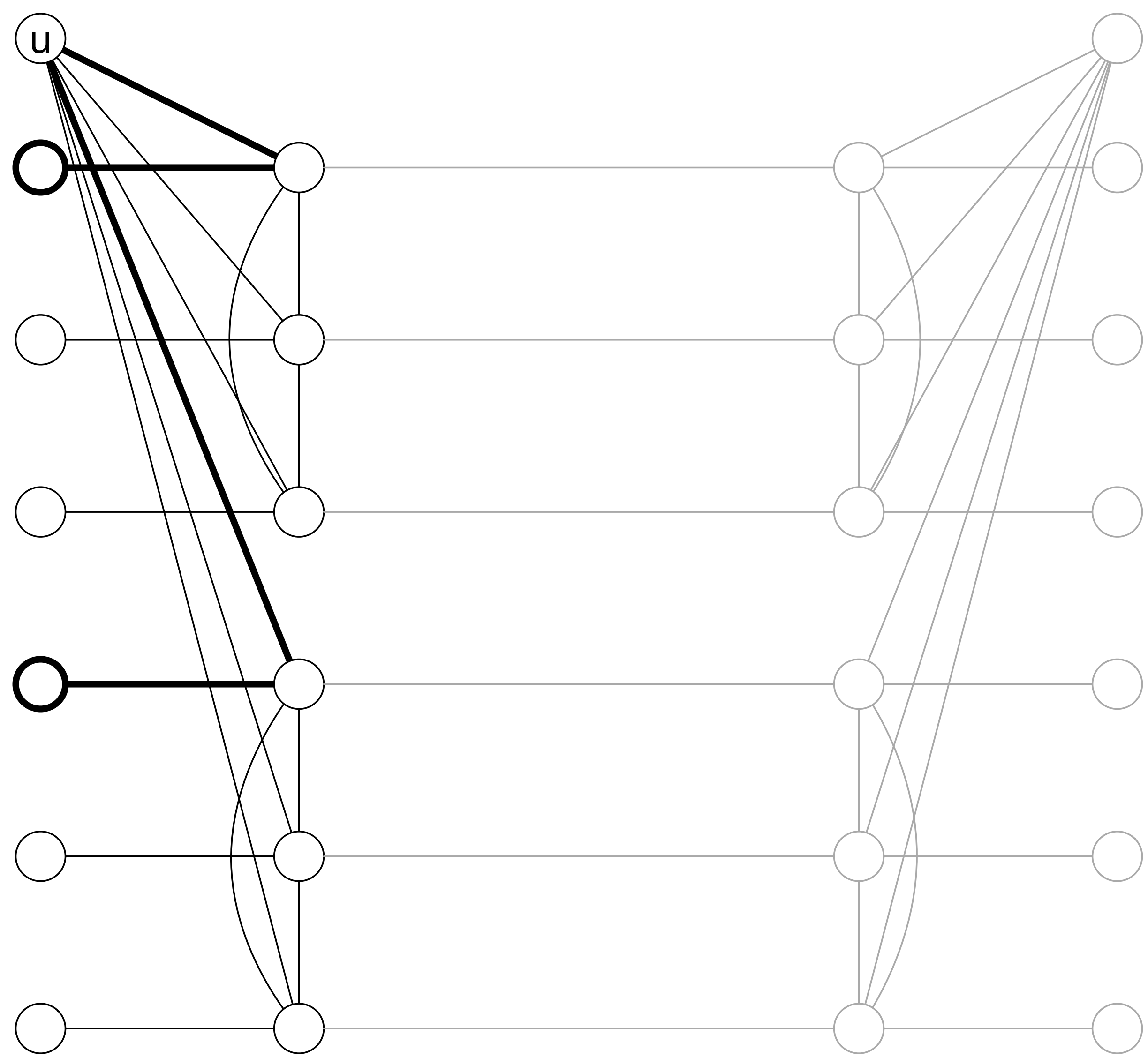
0
1
1
0
0
0
1
0
0



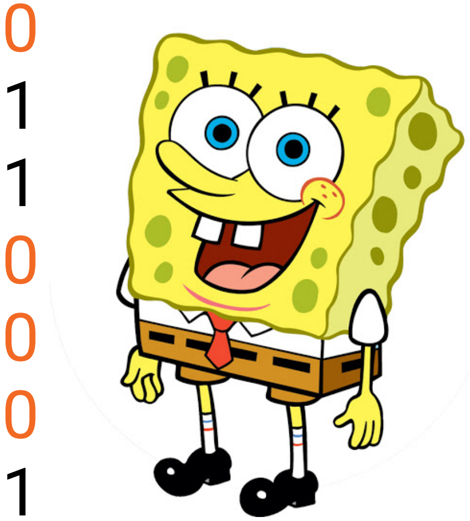
Lower bound for computing the diameter

Sets are disjoint, **diameter = 4**

1
0
0
0
1
1
0
1
0



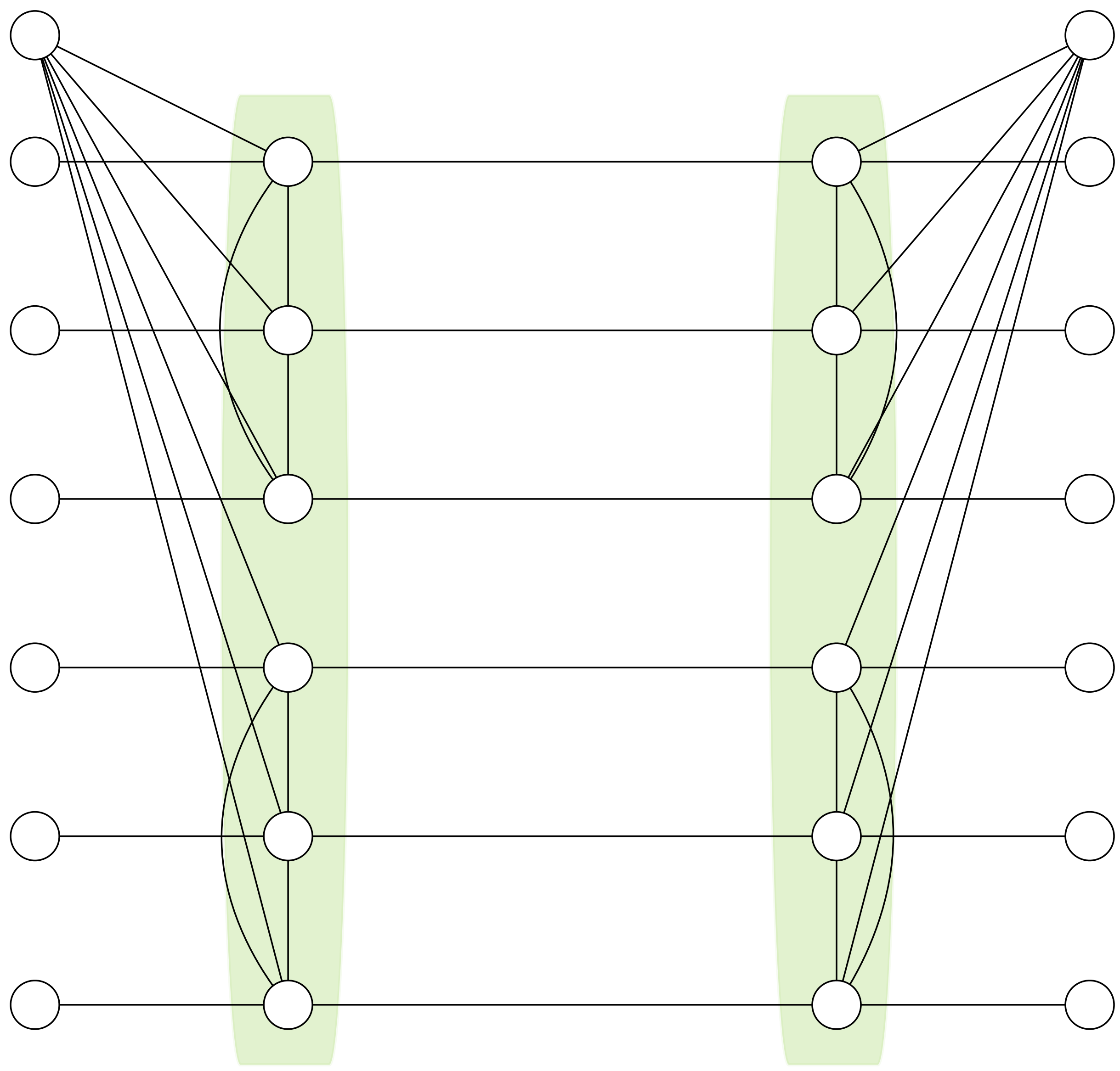
0
1
1
0
0
0
1
0
0



Lower bound for computing the diameter

Sets are disjoint, **diameter = 4**

1
0
0
0
1
1
0
1
0

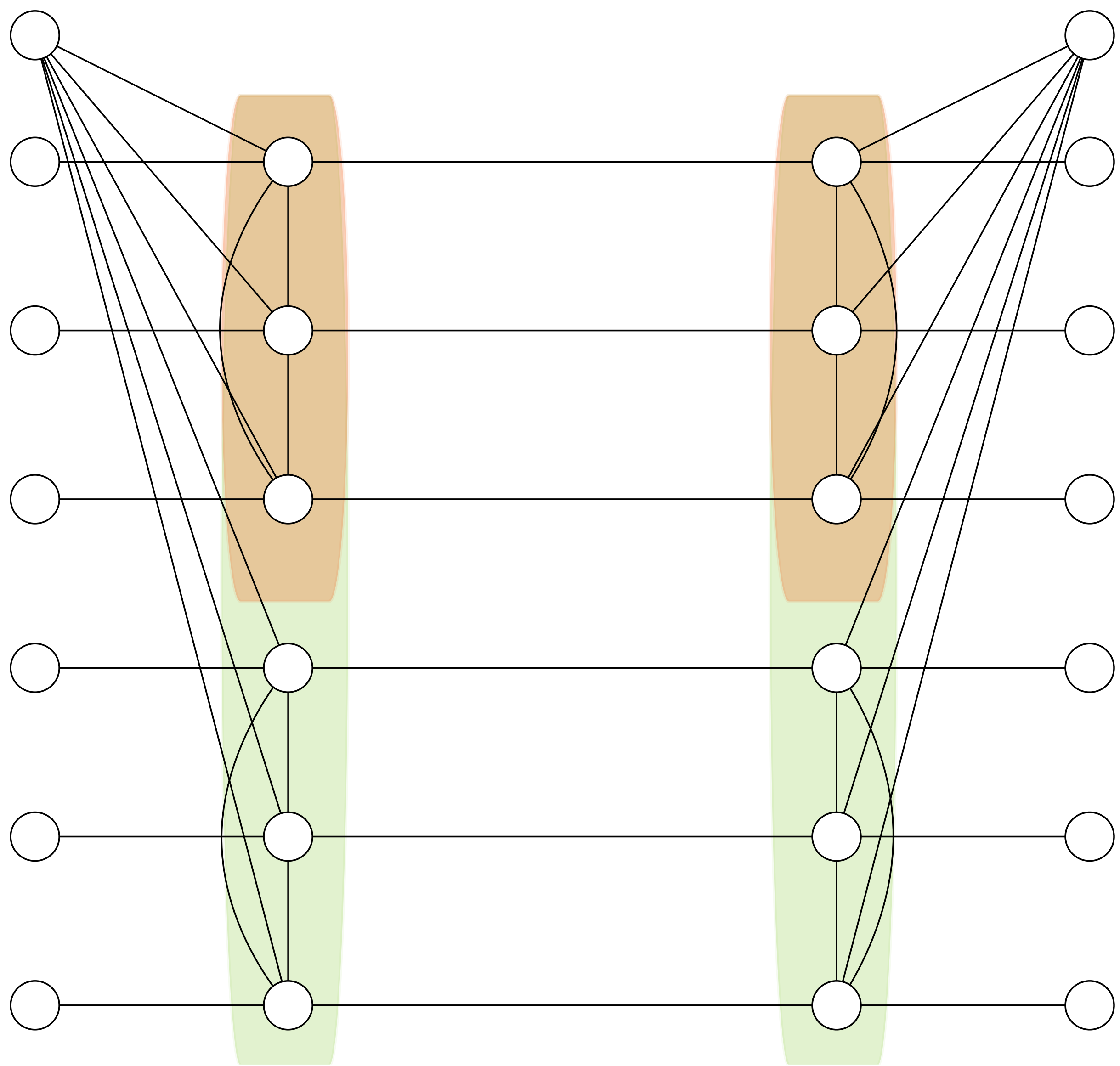


0
1
1
0
0
0
1
0
0

Lower bound for computing the diameter

Sets are disjoint, **diameter = 4**

1
0
0
0
1
1
0
1
0

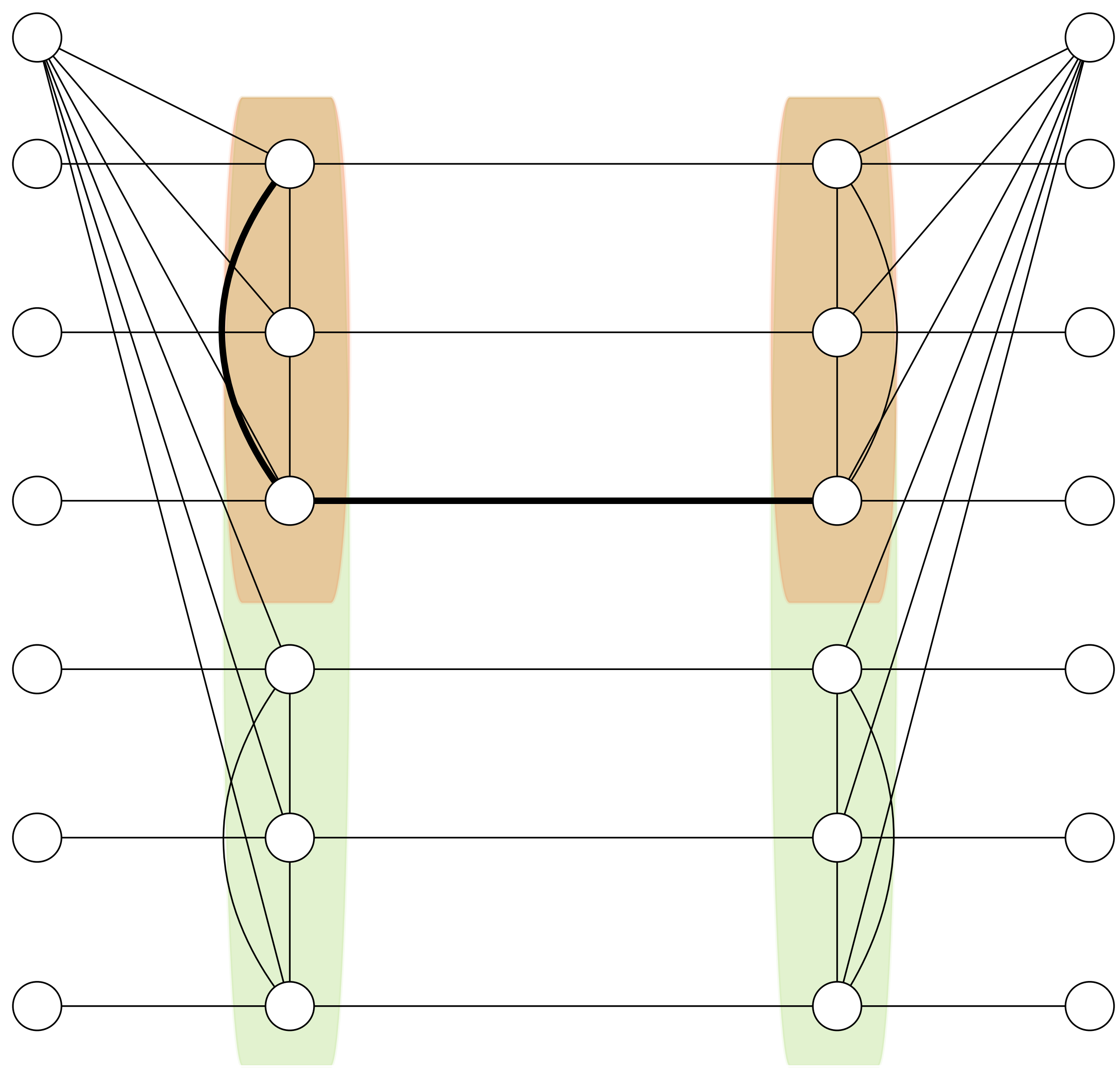


0
1
1
0
0
0
1
0
0

Lower bound for computing the diameter

Sets are disjoint, **diameter = 4**

1
0
0
0
1
1
0
1
0

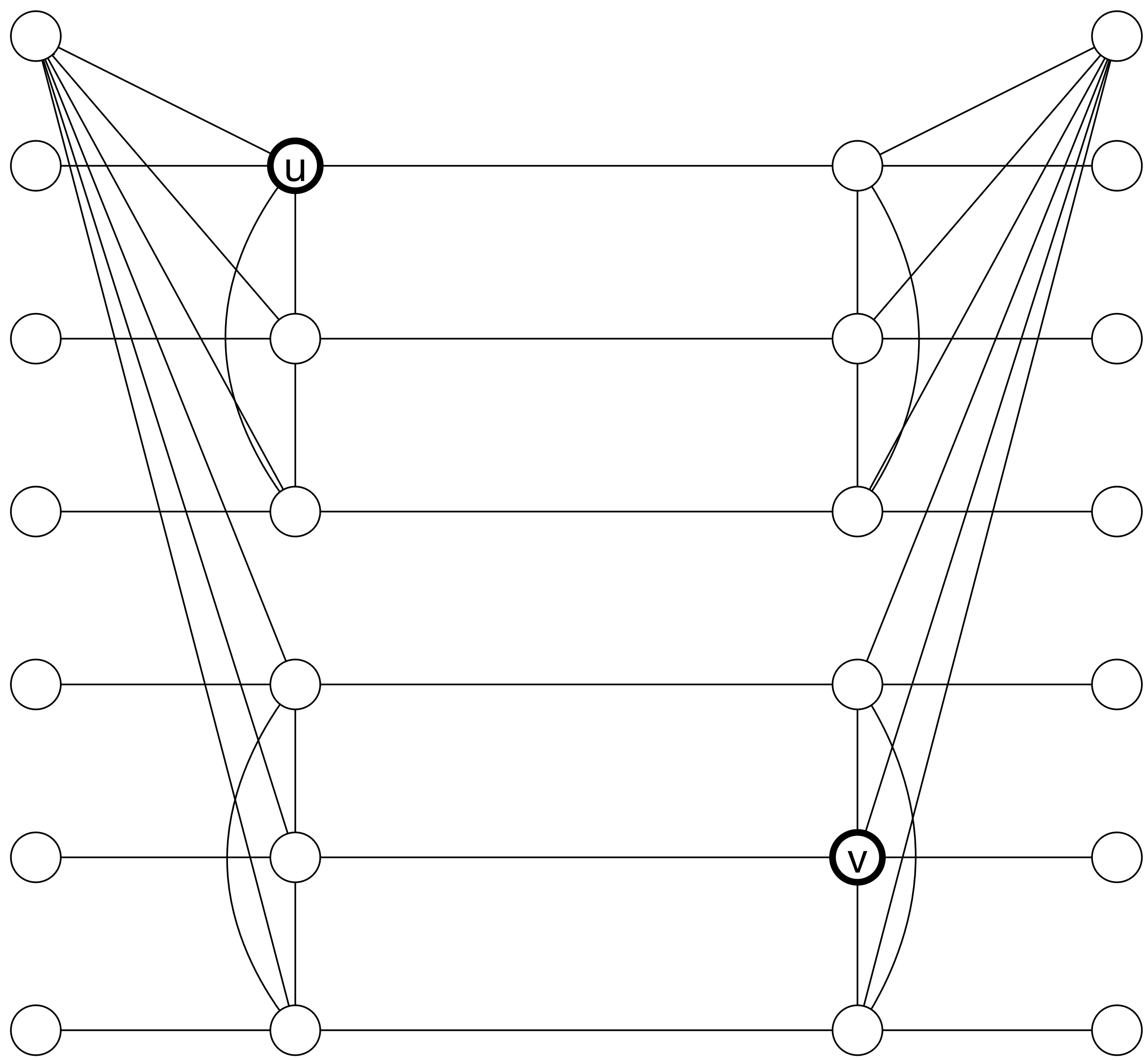


0
1
1
0
0
0
1
0
0

Lower bound for computing the diameter

Sets are disjoint, **diameter = 4**

1
0
0
0
1
1
0
1
0

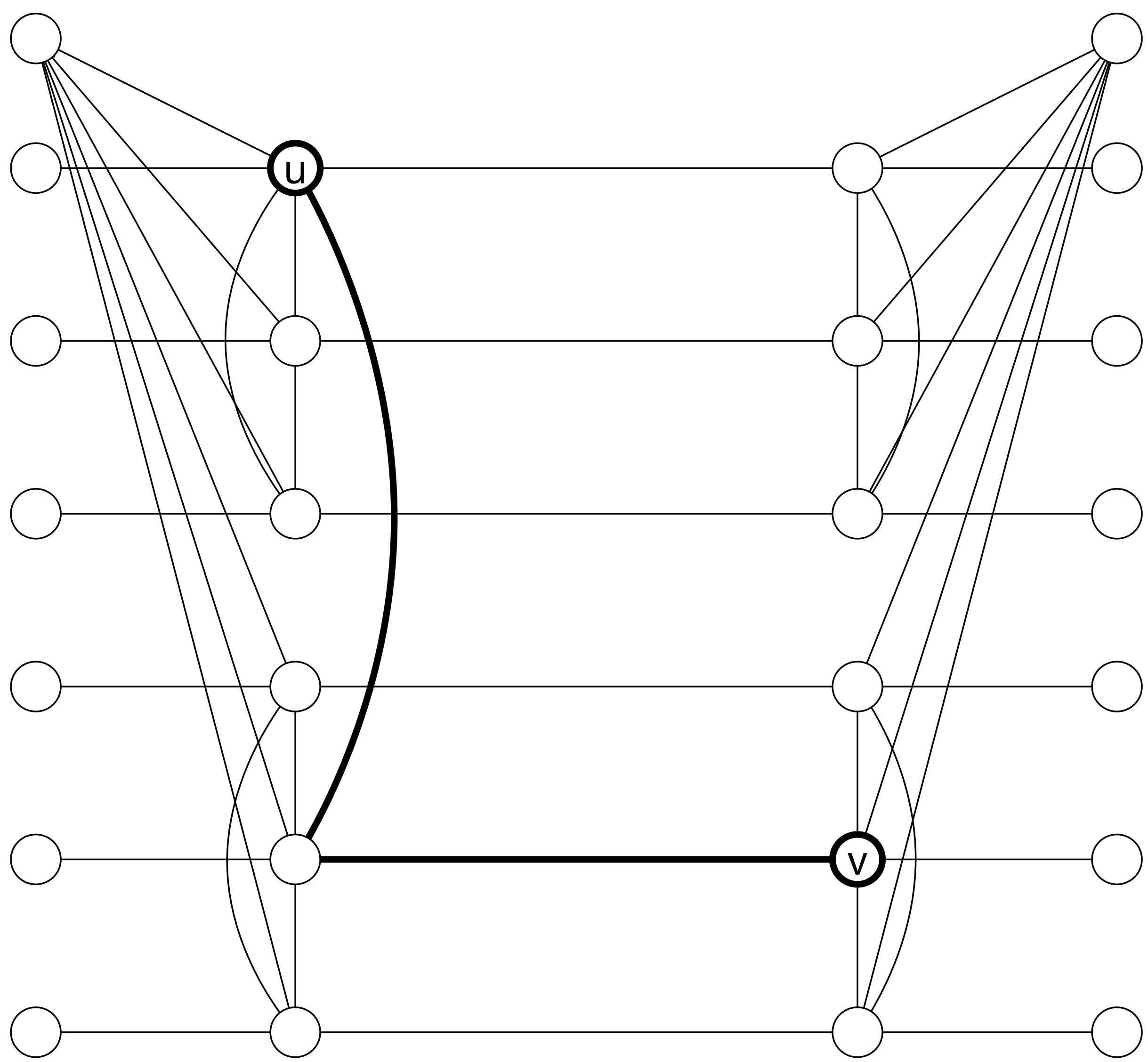


0
1
1
0
0
0
1
0
0

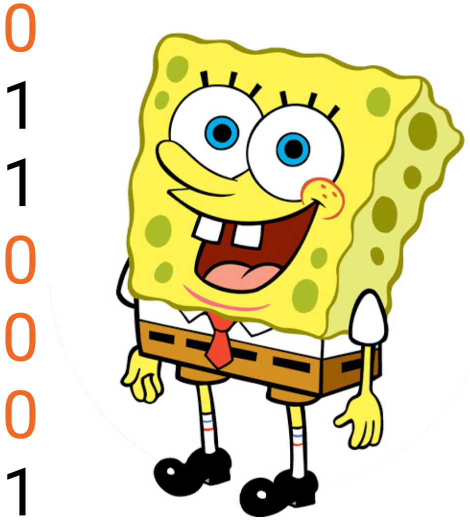
Lower bound for computing the diameter

Sets are disjoint, **diameter = 4**

1
0
0
0
1
1
0
1
0



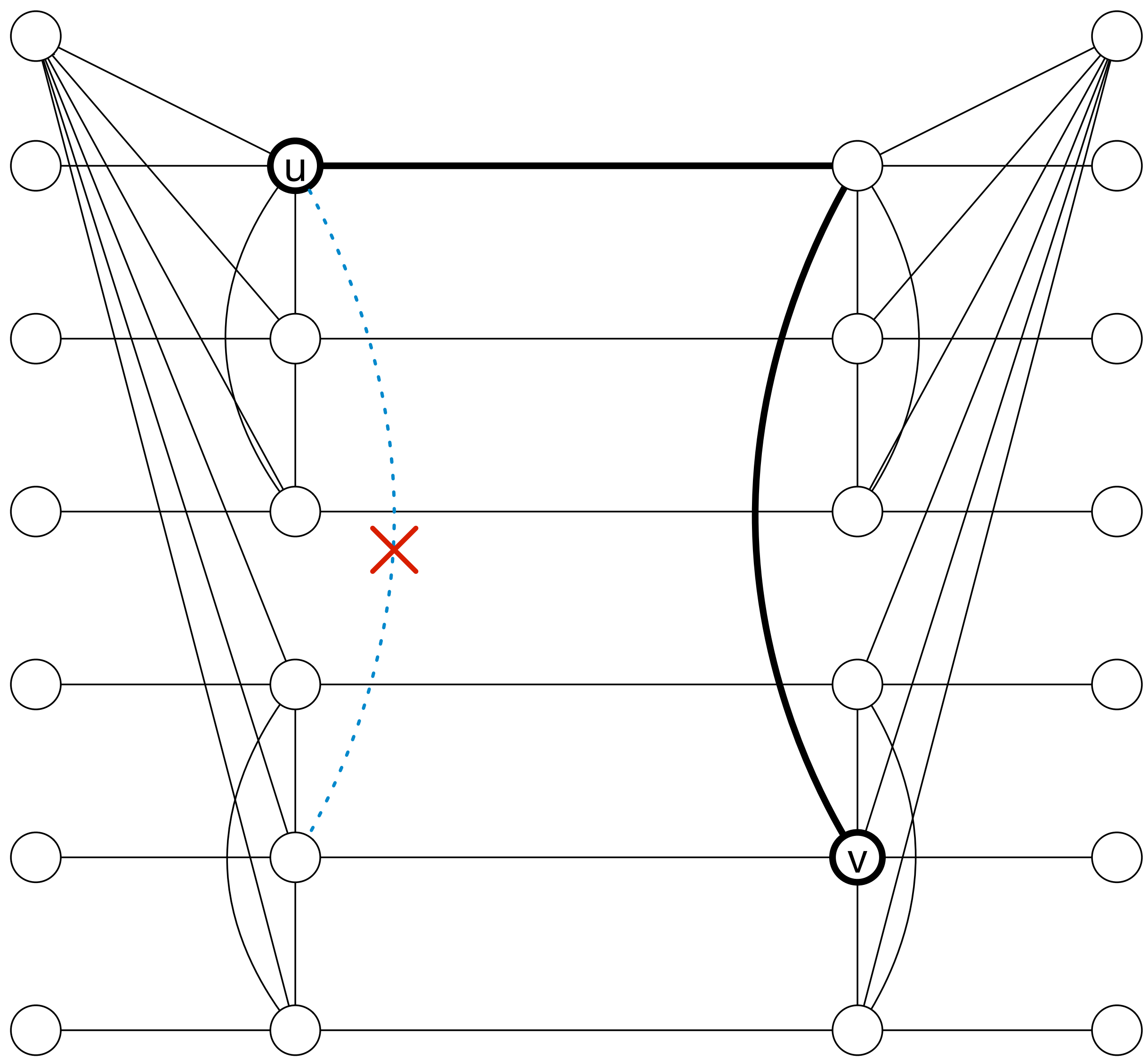
0
1
1
0
0
0
1
0
0



Lower bound for computing the diameter

Sets are disjoint, **diameter = 4**

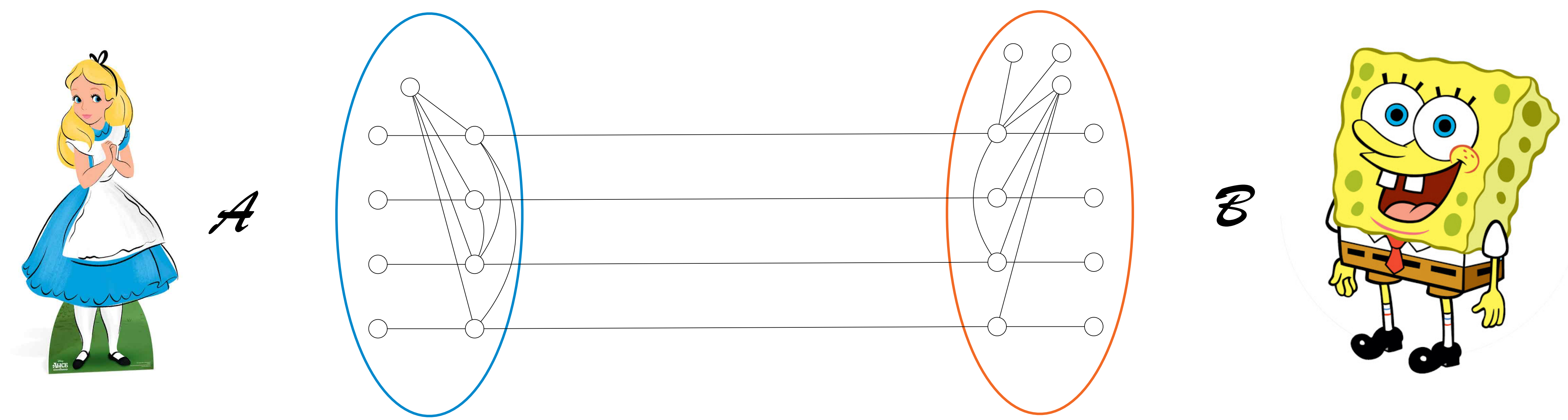
1
0
0
0
1
1
0
1
0



0
1
1
0
0
0
1
0
0

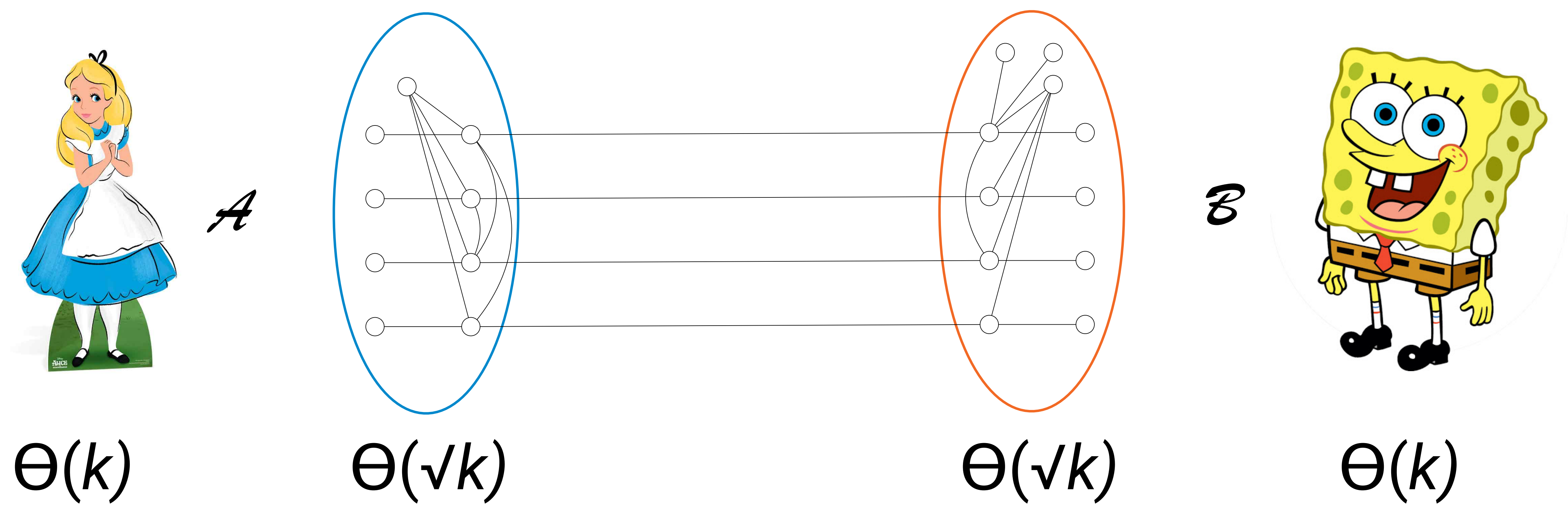
Lower bound for computing the diameter

- Diameter = 4 ⇒ sets are disjoint
- Diameter ≥ 5 ⇒ are not disjoint



Lower bound for computing the diameter

- Diameter = **4** ⇒ sets are **disjoint**
- Diameter ≥ **5** ⇒ are **not** disjoint



Lower bound for computing the diameter

- Diameter = **4** $\Rightarrow$ sets are **disjoint**
- Diameter $\geq$ **5** $\Rightarrow$ are **not** disjoint

$$n = \Theta(\sqrt{k})$$

$\mathcal{A}$

$\mathcal{B}$

$\Theta(k)$ $\Theta(\sqrt{k})$ $\Theta(\sqrt{k})$ $\Theta(k)$

Lower bound for computing the diameter

- Diameter = **4** ⇒ sets are **disjoint**
- Diameter ≥ **5** ⇒ are **not** disjoint

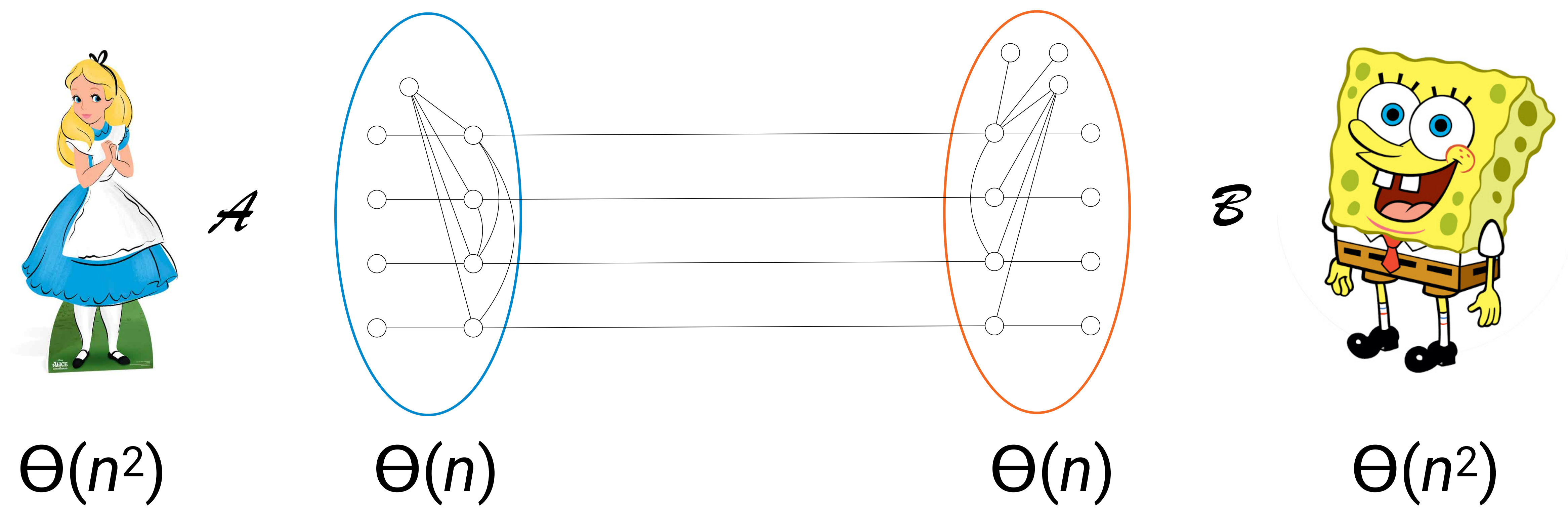
$\mathcal{A}$

$\mathcal{B}$

$\Theta(n^2)$ $\Theta(n)$ $\Theta(n)$ $\Theta(n^2)$

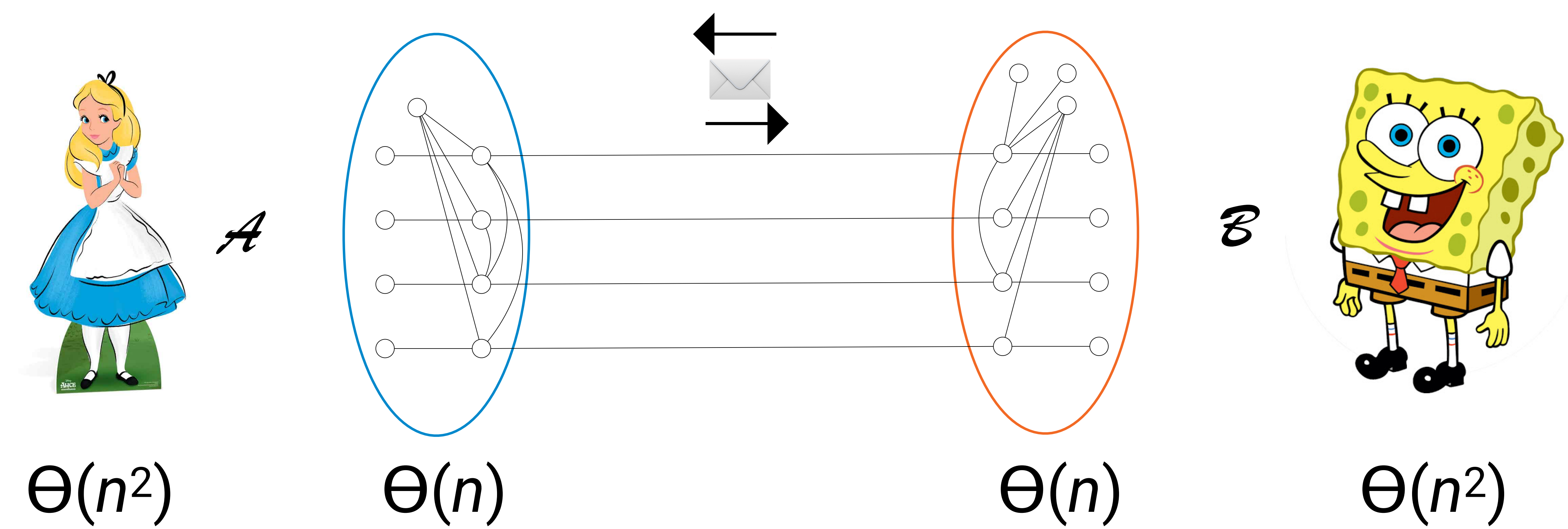
Lower bound for computing the diameter

Suppose we have an **algorithm *A*** for computing the diameter in the CONGEST model in time ***T*(*A*, *n*)**



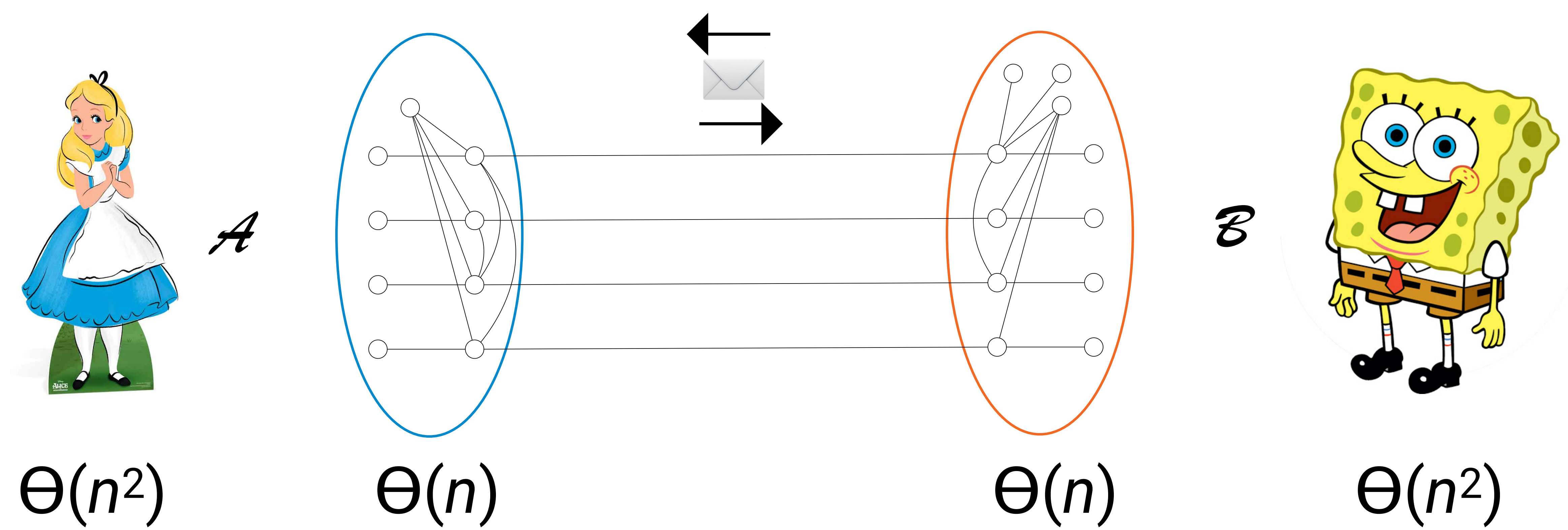
Lower bound for computing the diameter

- Simulate ***A*** $\Rightarrow$ solve **set disjointness**



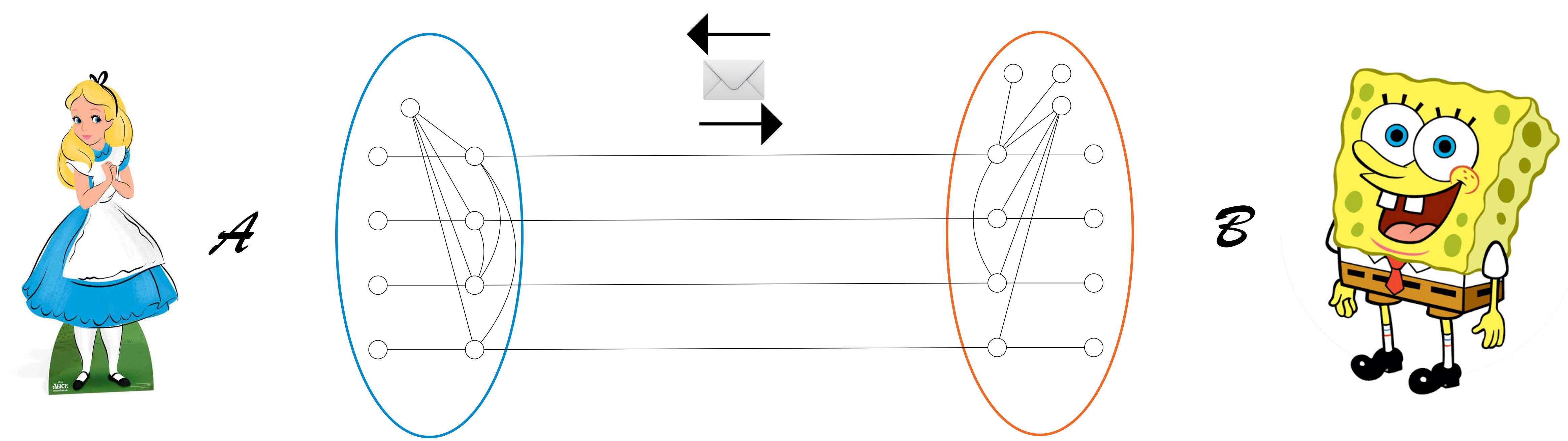
Lower bound for computing the diameter

- Simulate ***A*** ⇒ solve **set disjointness**
- **1 round** of simulation of *A*: exchange **Θ(*n* log *n*) bits**



Lower bound for computing the diameter

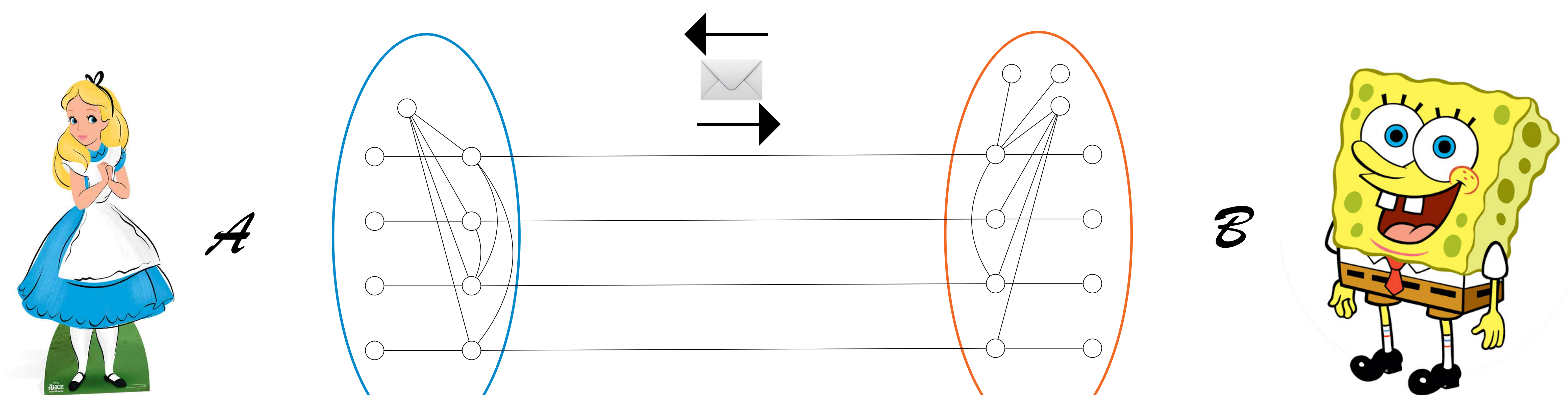
- Simulate ***A*** $\Rightarrow$ solve **set disjointness**
- **1 round** of simulation of A: exchange $\mathbf{\Theta(n \log n)}$ **bits**
- Total: $T(A, n) \times \Theta(n \log n)$



$\Theta(n^2)$ $\Theta(n)$ $\Theta(n)$ $\Theta(n^2)$

Lower bound for computing the diameter

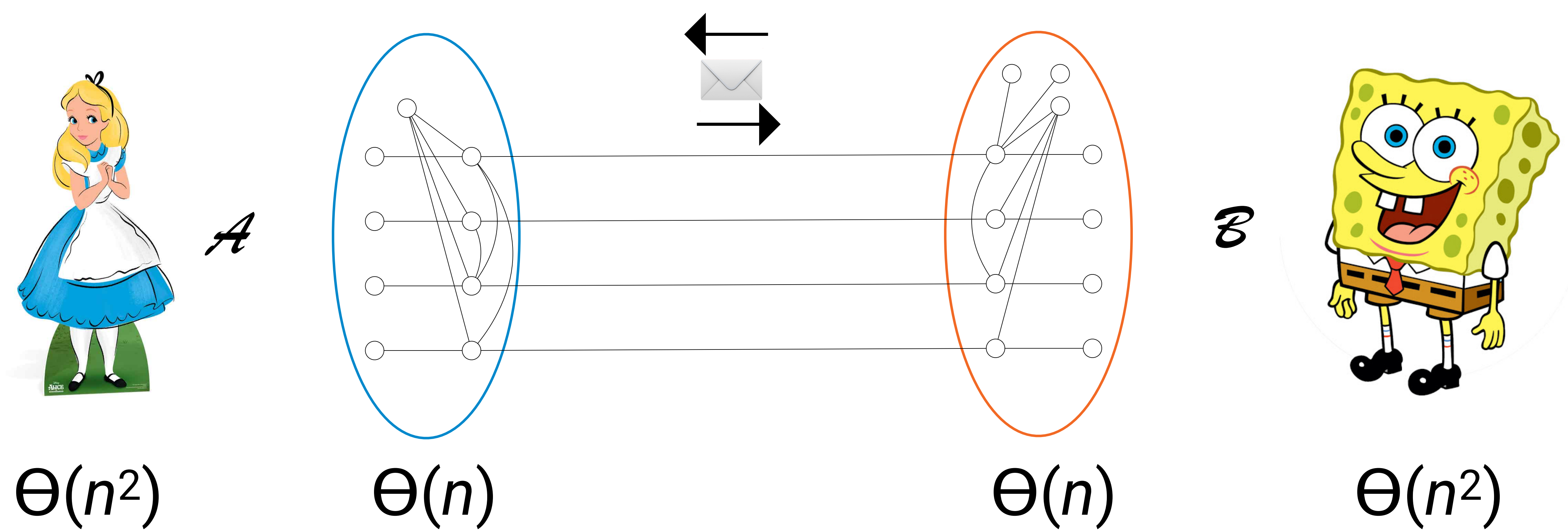
- Simulate *A* ⇒ solve **set disjointness**
- **1 round** of simulation of *A*: exchange $\mathbf{\Theta(n \log n)}$ **bits**
- Total: $T(A, n) \times \Theta(n \log n) \in \Omega(n^2)$



$\Theta(n^2)$ $\Theta(n)$ $\Theta(n)$ $\Theta(n^2)$

Lower bound for computing the diameter

- Simulate ***A*** $\Rightarrow$ solve **set disjointness**
- **1 round** of simulation of A: exchange $\boldsymbol{\Theta(n \log n)}$ **bits**
- Total: $T(A, n) \times \Theta(n \log n) \in \Omega\,(n^2) \Rightarrow$ $\boldsymbol{T(A, n) \in \Omega\,(n/\log n)}$



Summary

- **LOCAL** model: unlimited bandwidth
- **CONGEST** model: $O(\log n)$ bandwidth
- $O(n)$ or $O(\mathrm{diam}(G))$ time is no longer trivial
- Example:
  - **APSP** in time $O(n)$, pipelining helps
  - **APSP** requires $\Omega(n/\log n)$ rounds