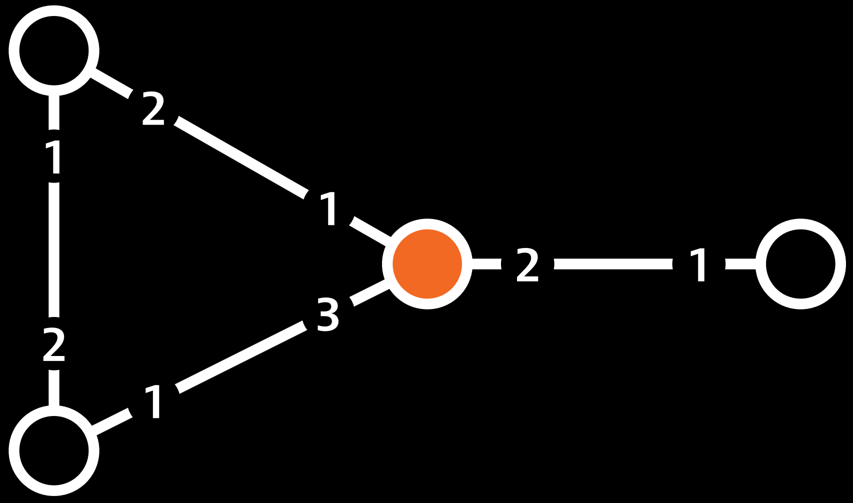
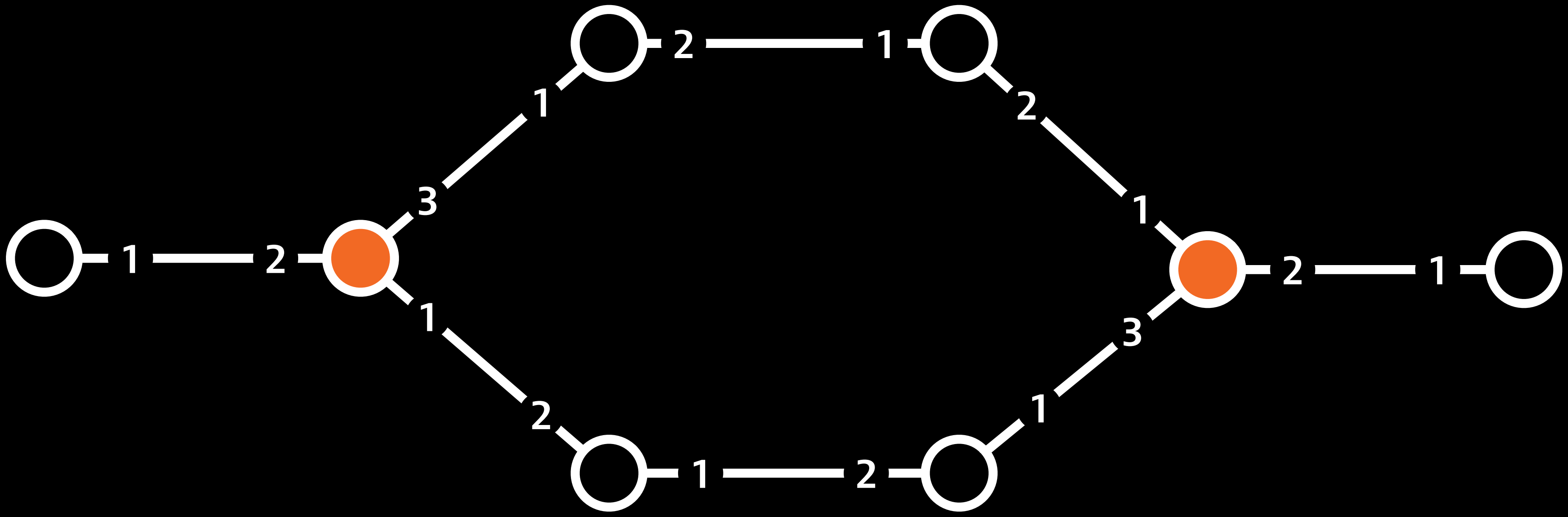
Aalto University

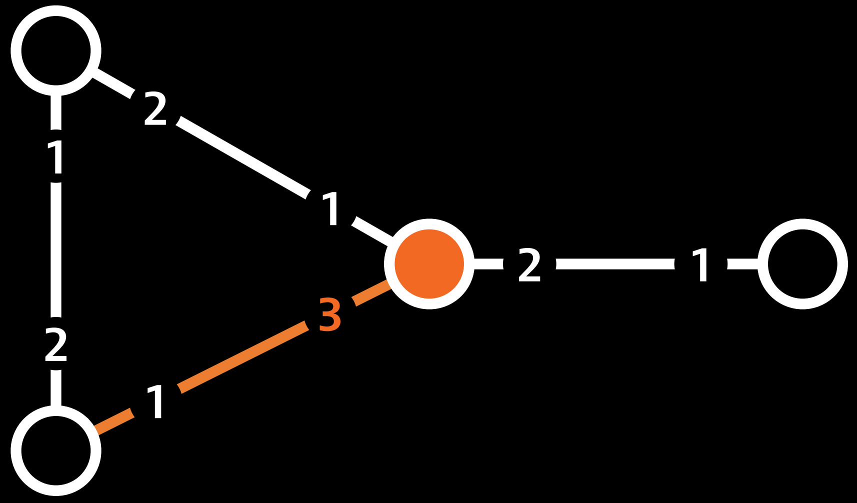
Distributed Algorithms 2021

7 Covering maps

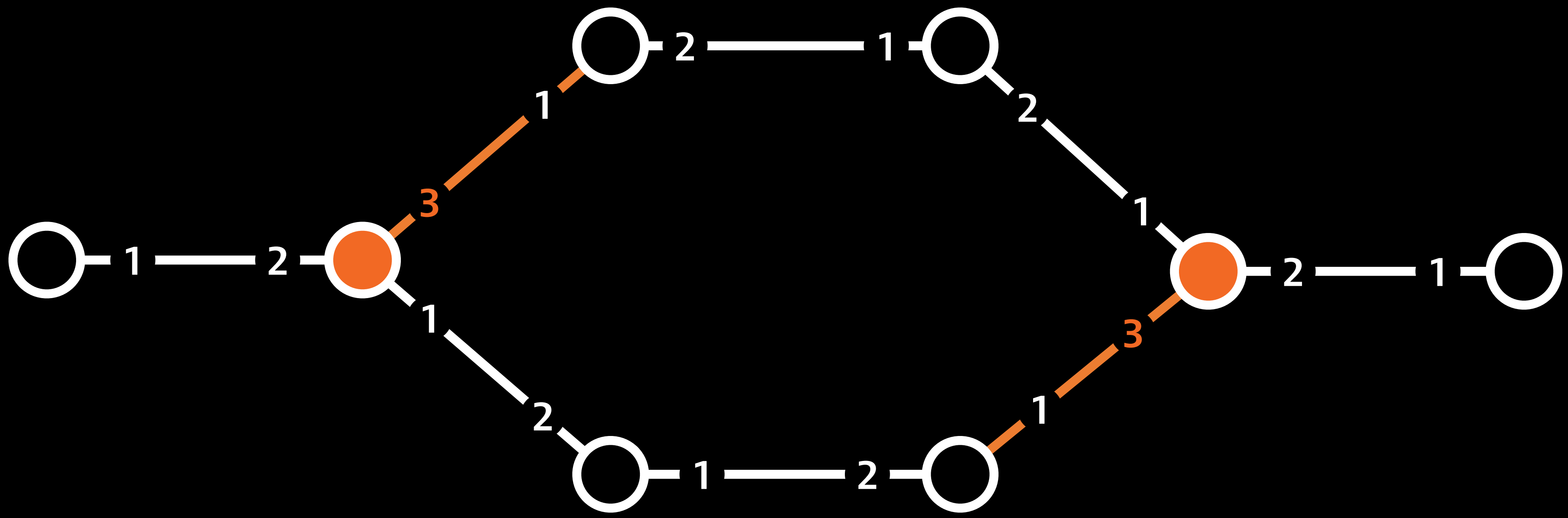


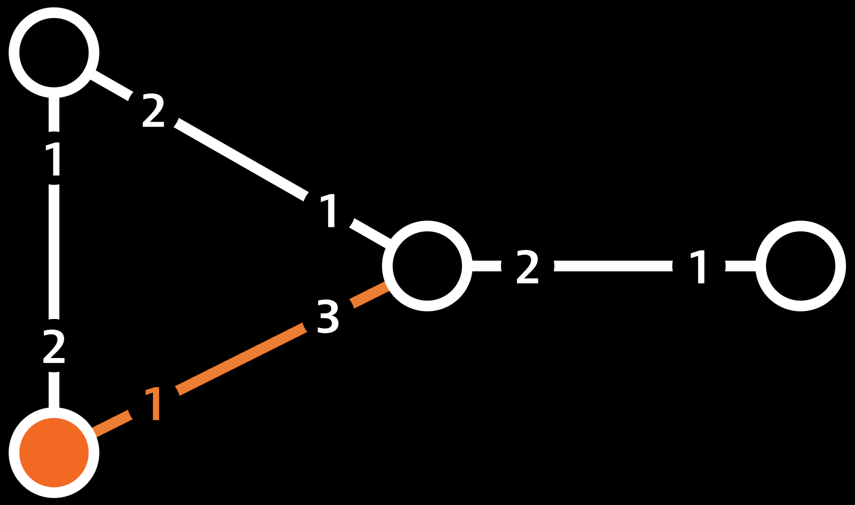
You are in a room
with three doors,
labeled 1, 2, and 3.
> _





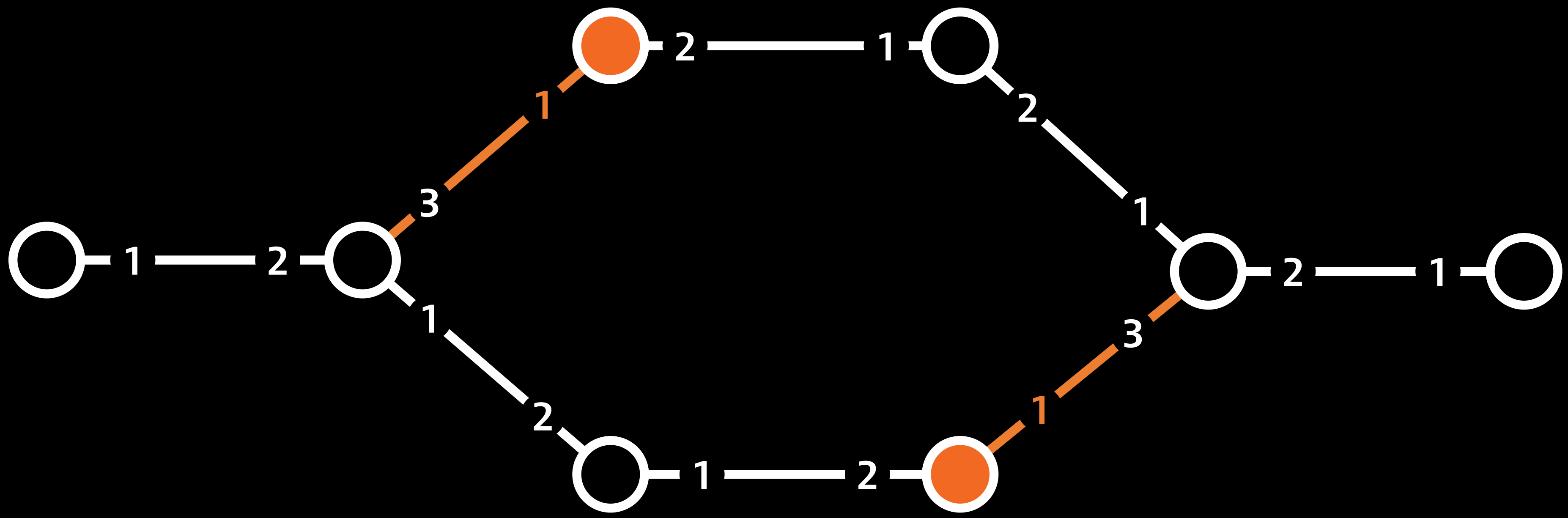
You are in a room
with three doors,
labeled 1, 2, and 3.
> open door 3

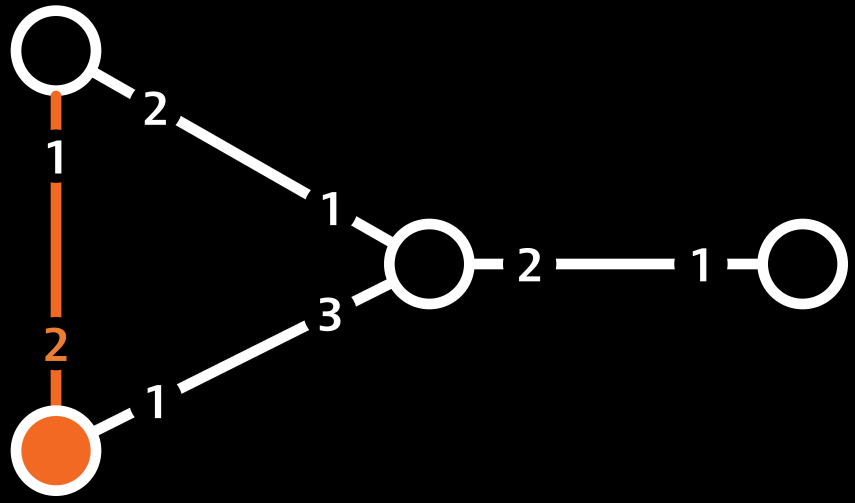




You enter a room with two doors, labeled 1 and 2. You just came in through doorway 1.

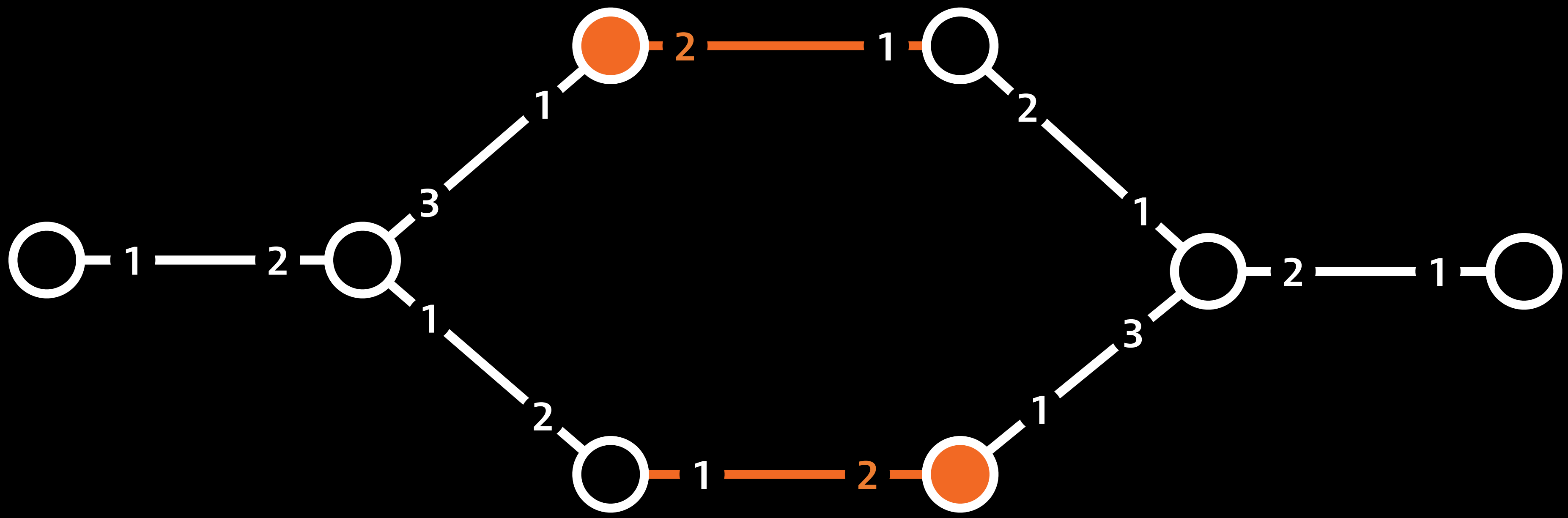
> _

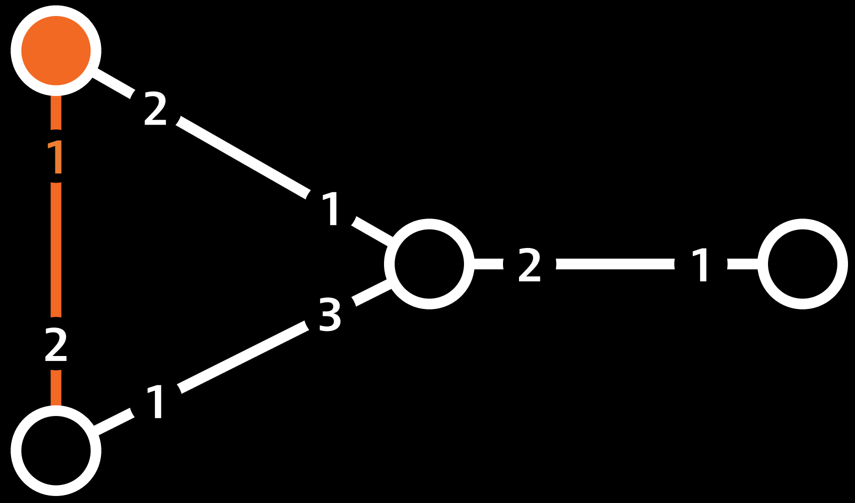




You enter a room with two doors, labeled 1 and 2. You just came in through doorway 1.

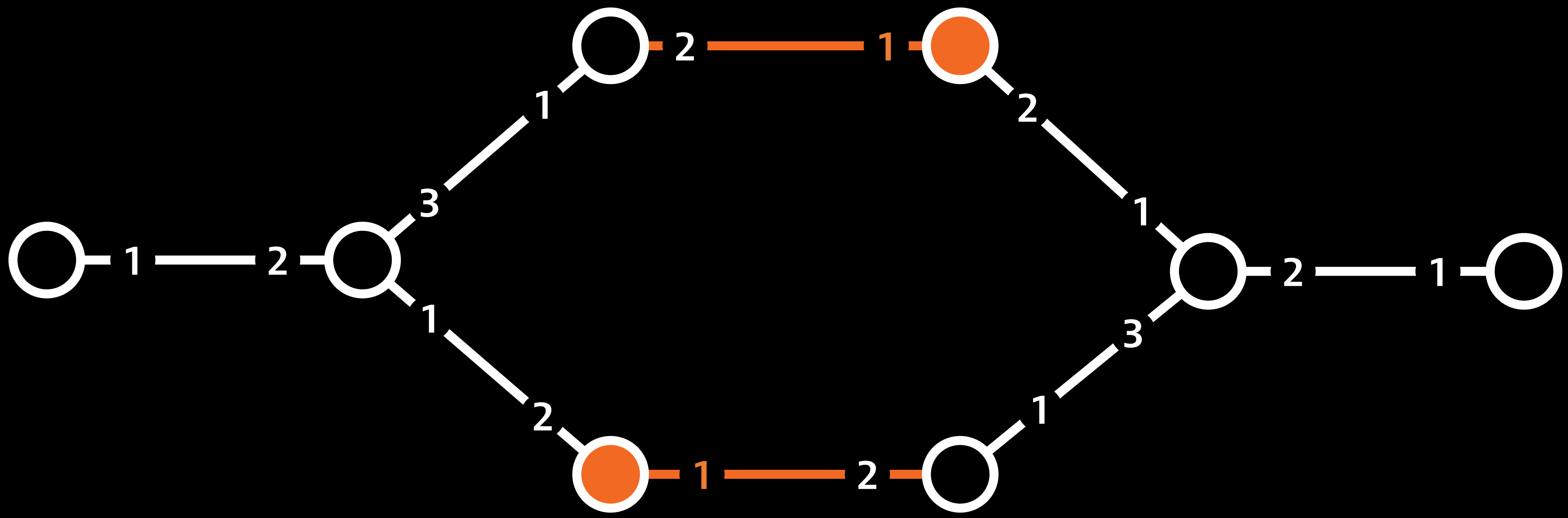
> open door 2

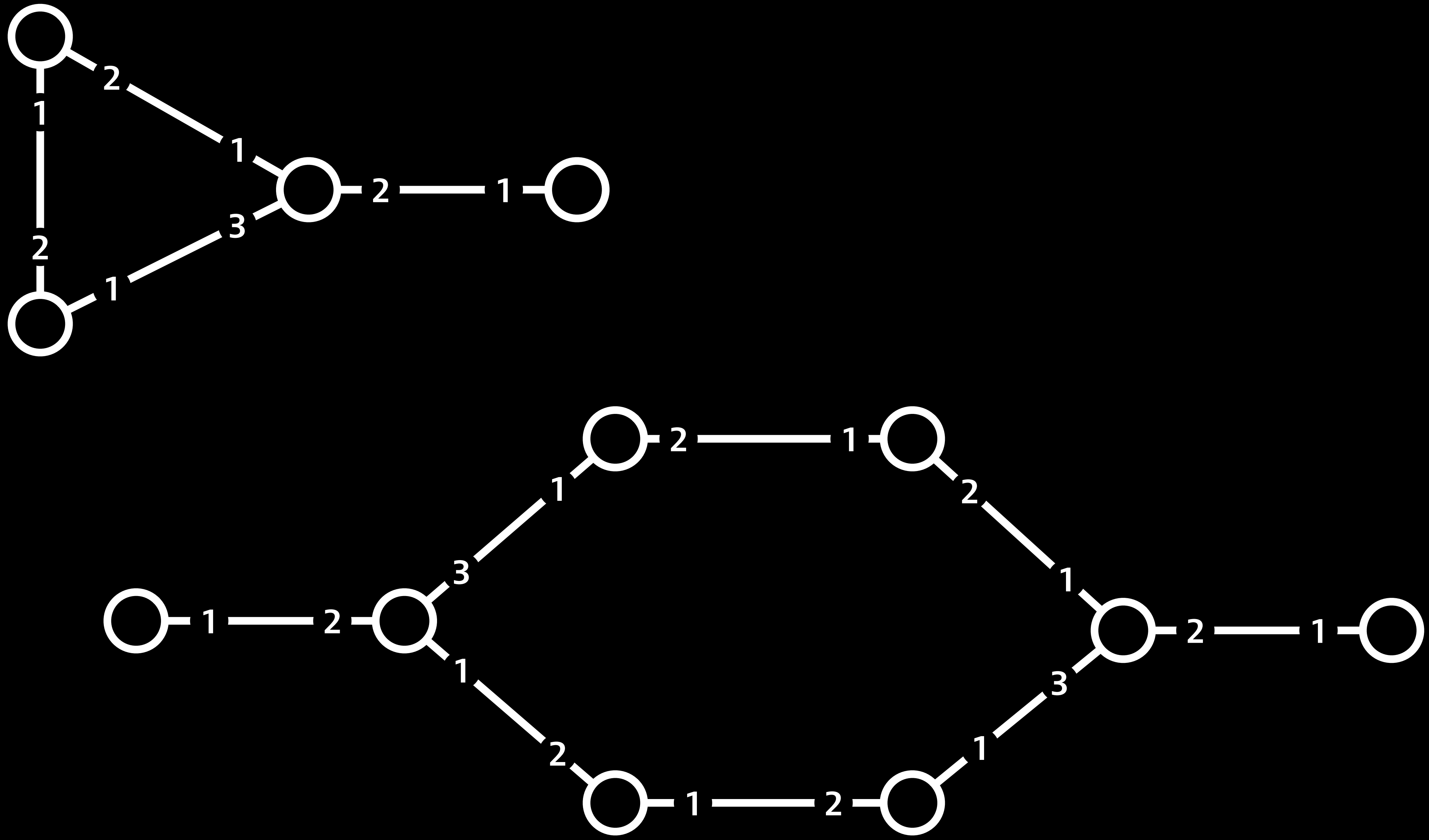




You enter a room with two doors, labeled 1 and 2. You just came in through doorway 1.

> _




2
1
1
2
1
3
2
1
2
1
2
1
3
1
2
1
1
2
1
2
3
1
2
1
1
2

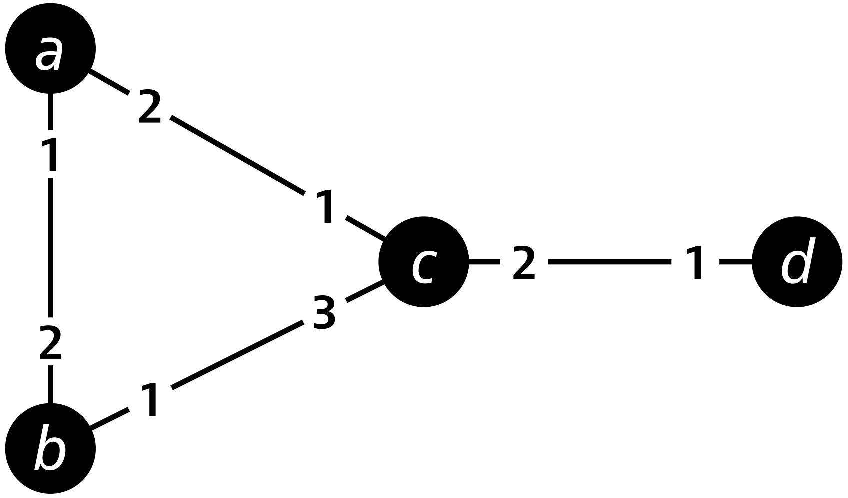
High-level plan

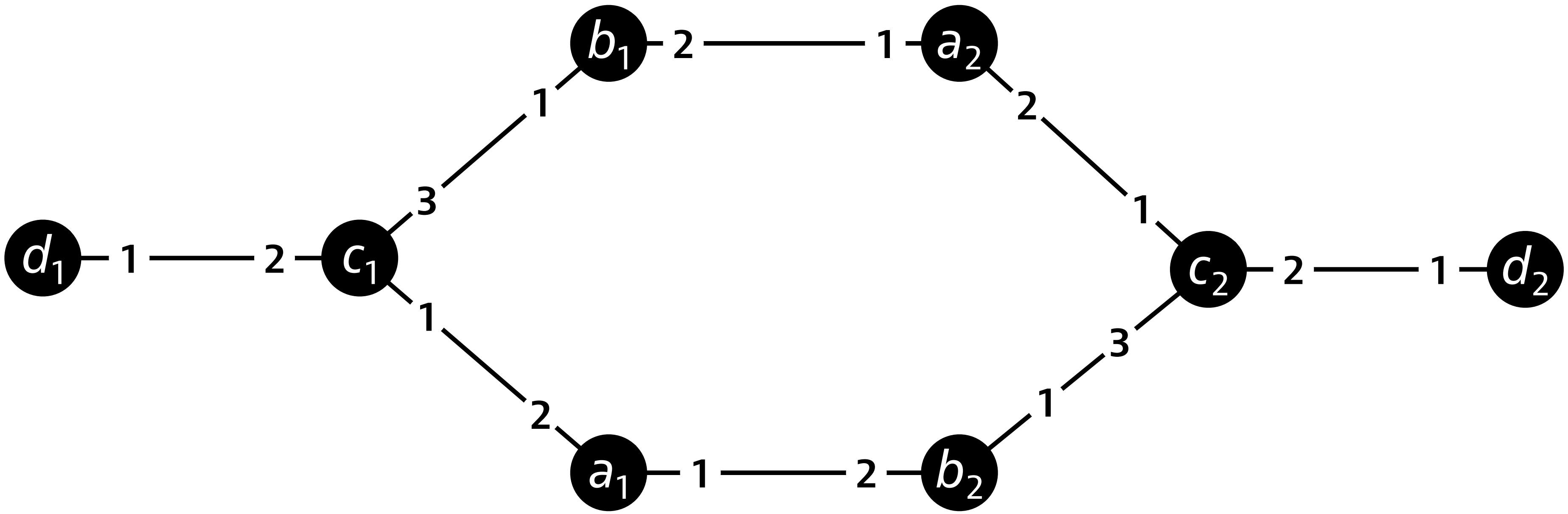
- **Goal:**
  - show that problem X cannot be solved in the port-numbering model
- **General approach:**
  - construct port-numbered networks so that some nodes u, v, … will always produce the ***same output***
  - show that if u, v, … have the same output, then it is ***not a feasible solution*** for X

Covering maps used here

Covering map

- Two port-numbered networks:
  - $N = (V, P, p)$
  - $N' = (V', P', p')$
- Surjection ***f*: *V* → *V'*** that preserves:
  - inputs
  - degrees
  - connections
  - port numbers


a
b
c
d
2
1
1
2
3
1
2
1


d_1
c_1
b_1
a_1
a_2
b_2
c_2
d_2
1
2
3
1
1
2
2
1
1
2
2
1
1
3
2
1

Covering map

- "Fools" any deterministic algorithm
- If f is a covering map from N to N', then:
  - v and $f(v)$ have the same state ***before*** round 1
  - v and $f(v)$ send the same messages in round 1
  - v and $f(v)$ receive the same messages in round 1
  - v and $f(v)$ have the same state ***after*** round 1

Covering map

- "Fools" any deterministic algorithm
- If f is a covering map from N to N', then:
  - v and $f(v)$ have the same state ***before*** round T
  - v and $f(v)$ send the same messages in round T
  - v and $f(v)$ receive the same messages in round T
  - v and $f(v)$ have the same state ***after*** round T

Common steps

- Starting point: graph problem X
- Which graph G would be a “hard instance”?
- How to choose a port numbering N of G ?
- How to choose the other network N' ?
- How to construct mapping from N to N' ?

Example: 2-node path

Example: 4-node path

Example: two cycles

Common setup

- ***N* is the network we care about**
  - simple port-numbered network
  - well-defined and interesting underlying graph
- ***N′* is something strange**
  - not necessarily a simple port-numbered network
  - running A in N' makes no sense
  - **introduced only to analyze what happens when we run *A* in *N***

Observations

- We can use covering maps to construct ***universal*** counterexamples
  - **adaptive:** *"for any given algorithm A we can find a hard input N"*
  - **universal:** *"there is an input N that is hard for any algorithm A"*