

Harvard · CS50-AI | Introduction to Artificial Intelligence with Python (2020)

CS50-AI (2020) · 课程资料包 @ShowMeAI



视频

中英双语字幕



课件

一键打包下载



笔记

官方笔记翻译



代码

作业项目解析



视频 · B 站 [扫码或点击链接]

<https://www.bilibili.com/video/BV1AQ4y1y7wy>



课件 & 代码 · 博客 [扫码或点击链接]

<http://blog.showmeai.tech/harvard-cs50-ai>

深度优先
广度优先

贝叶斯

聚类
马尔可夫

A* 算法
自然语言处理

RNN

tensorflow
反向传播

SVM
强化学习

监督学习

无监督
语法解析

搜索

神经网络
CNN

Awesome AI Courses Notes Cheatsheets 是 [ShowMeAI](#) 资料库的分支系列，覆盖最具知名度的 **TOP50+** 门 AI 课程，旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

点击 课程名称，跳转至课程 **资料包** 页面，**一键下载** 课程全部资料！

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n
# Awesome AI Courses Notes Cheatsheets · 持续更新中			
知识图谱	图机器学习	深度强化学习	自动驾驶
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094



微信公众号

资料下载方式 2：扫码点击 **底部菜单栏**
称为 **AI 内容创作者**？回复 [添砖加瓦]

Introduction to
Artificial Intelligence
with Python

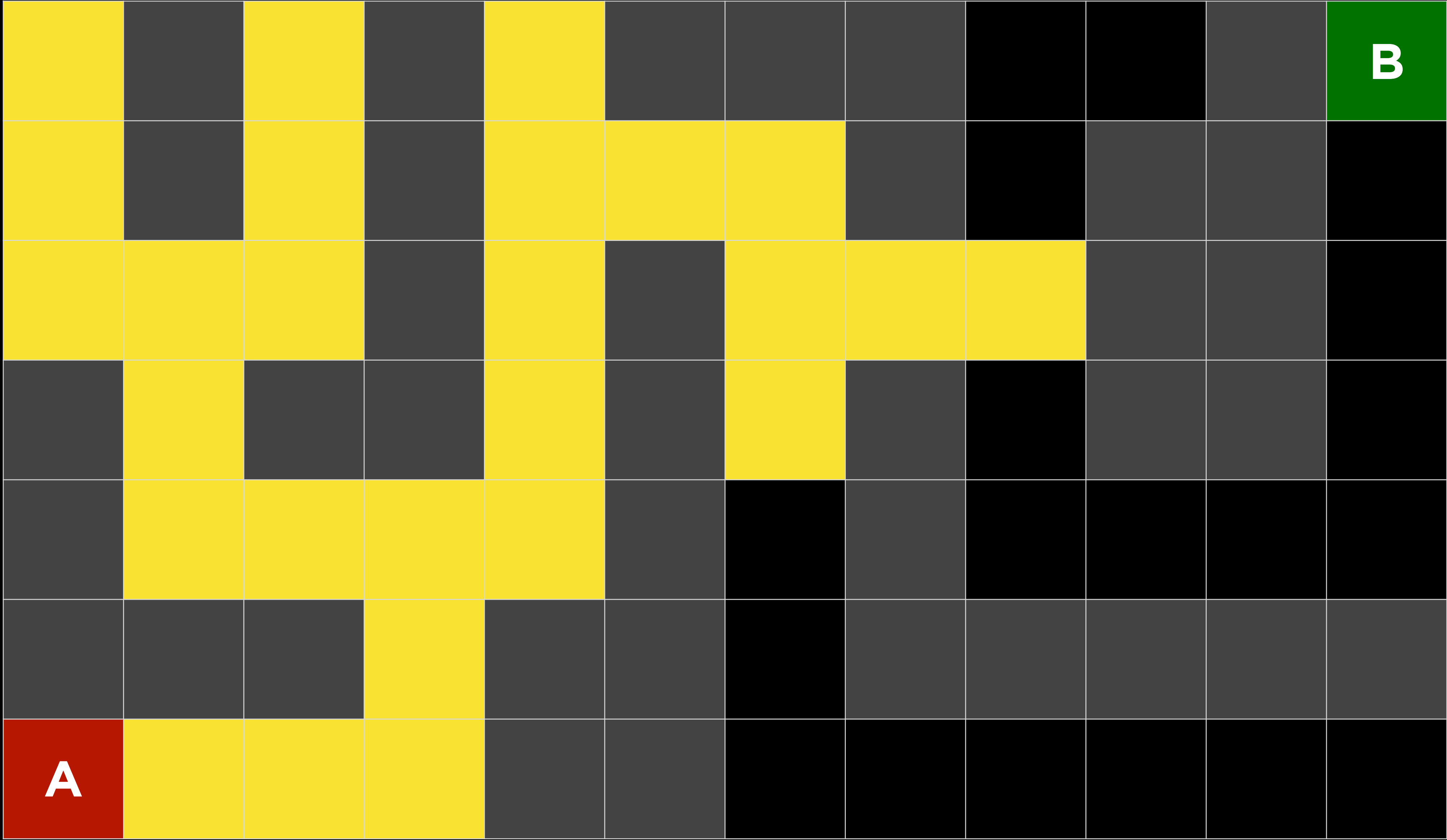
Optimization

optimization

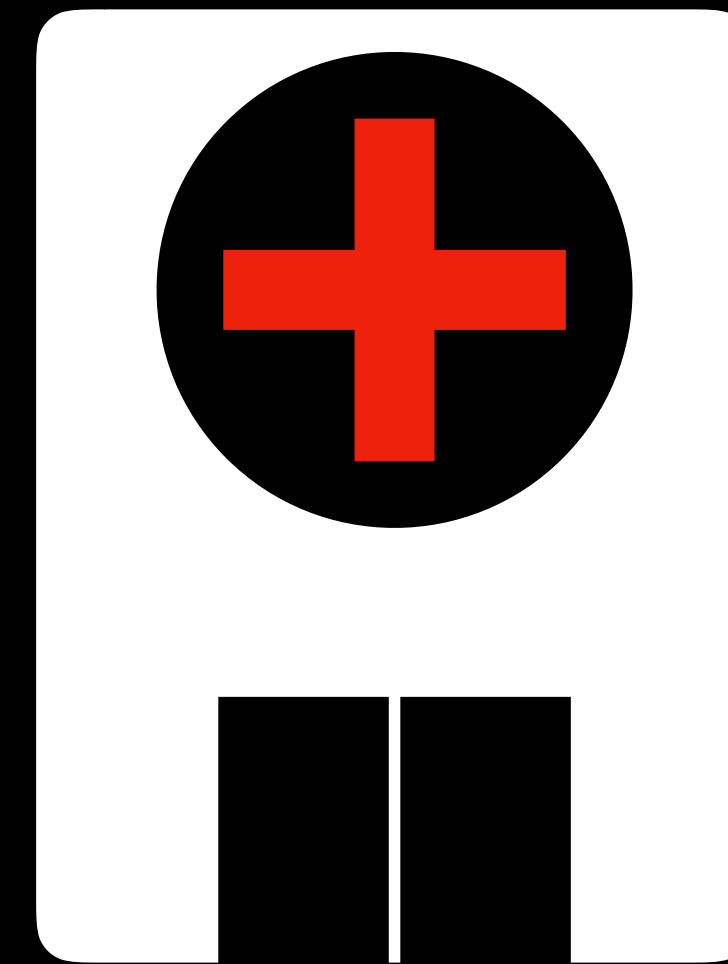
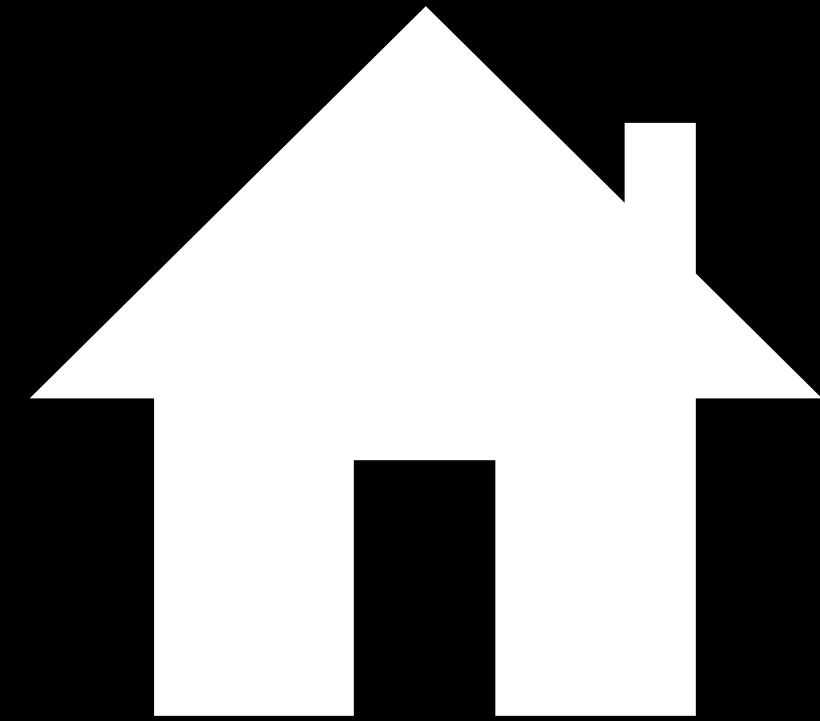
choosing the best option from a set of options

local search

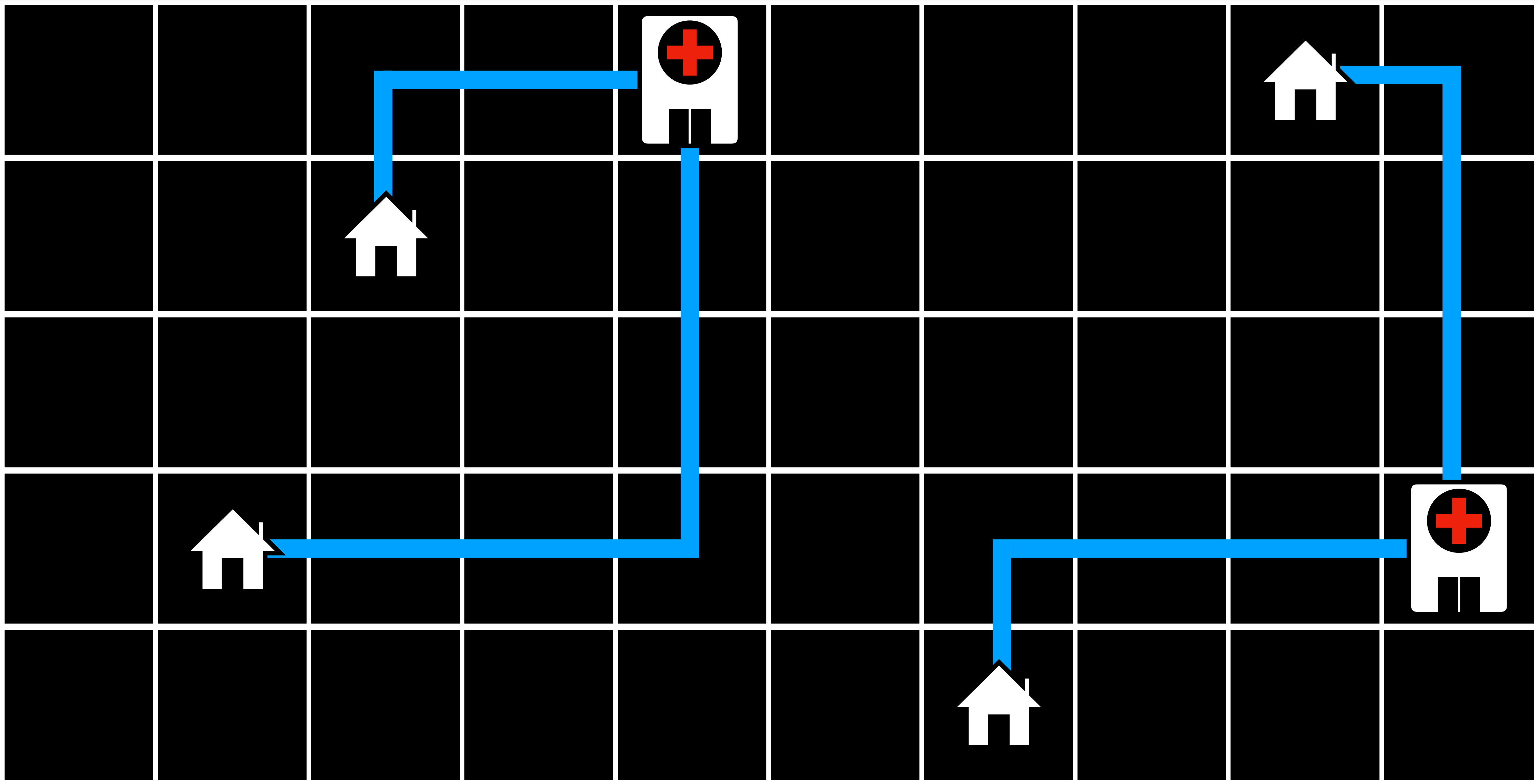
search algorithms that maintain a single node and searches by moving to a neighboring node



[illegible]

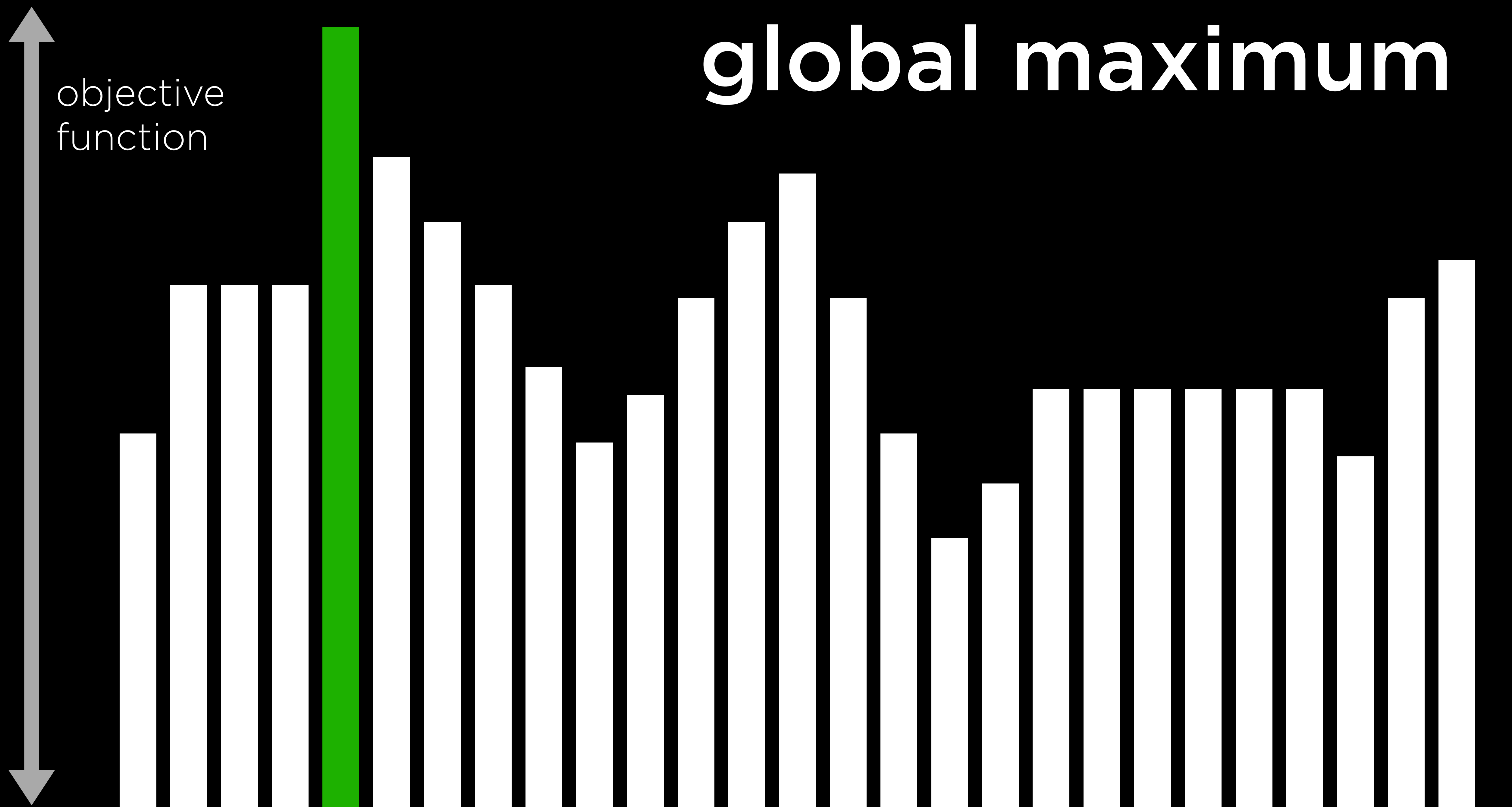


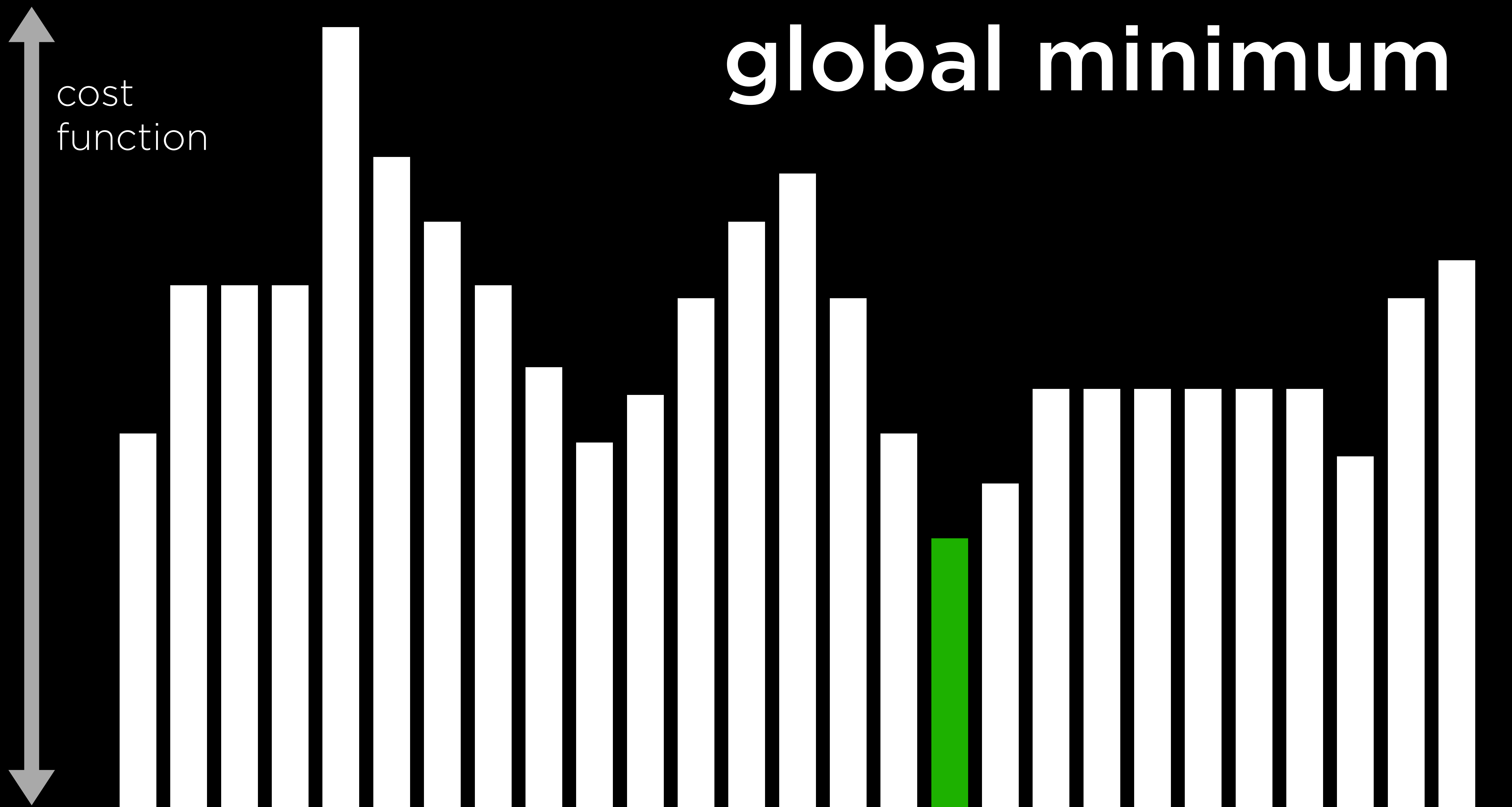
Cost: 17



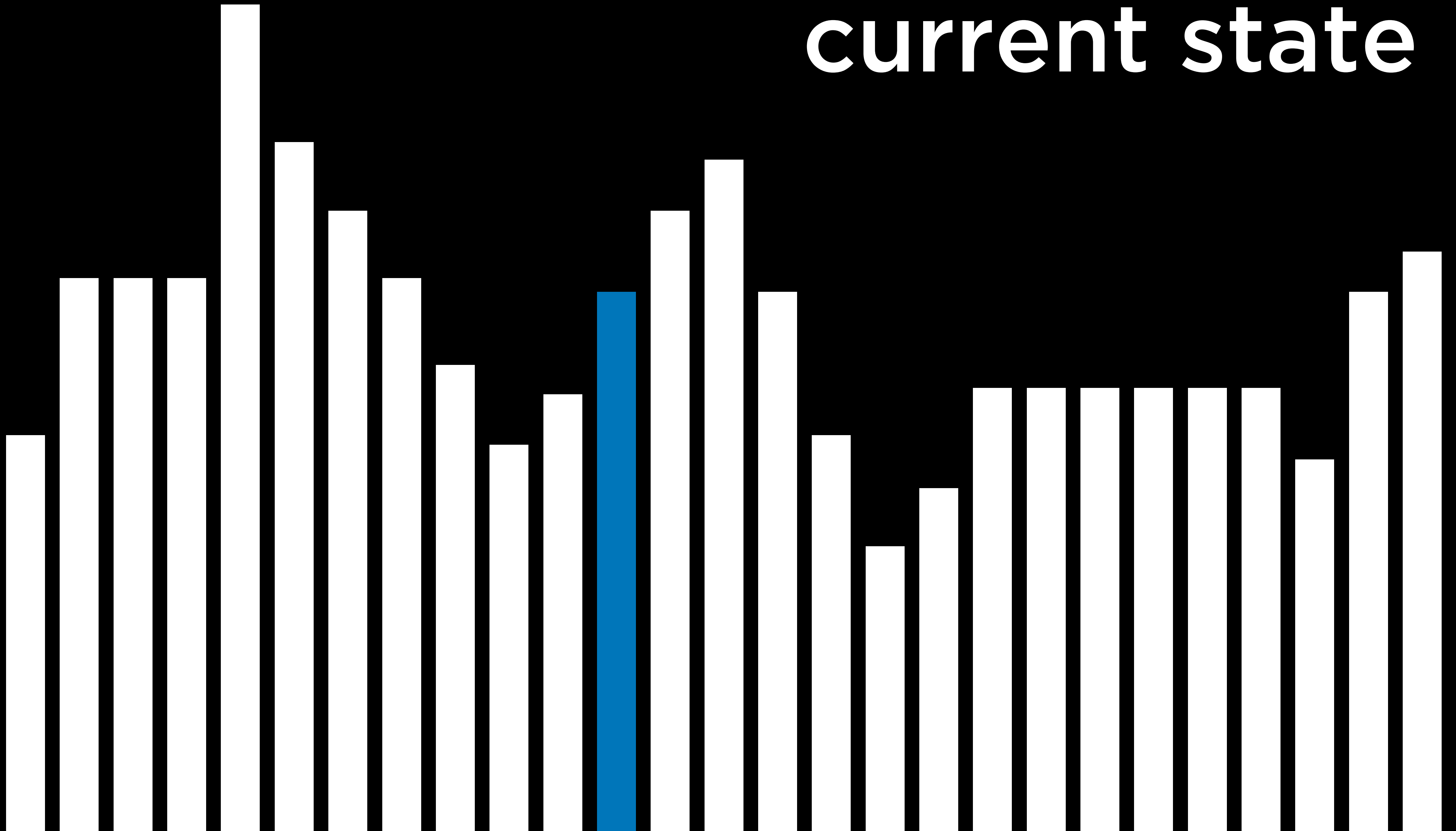
state-space landscape



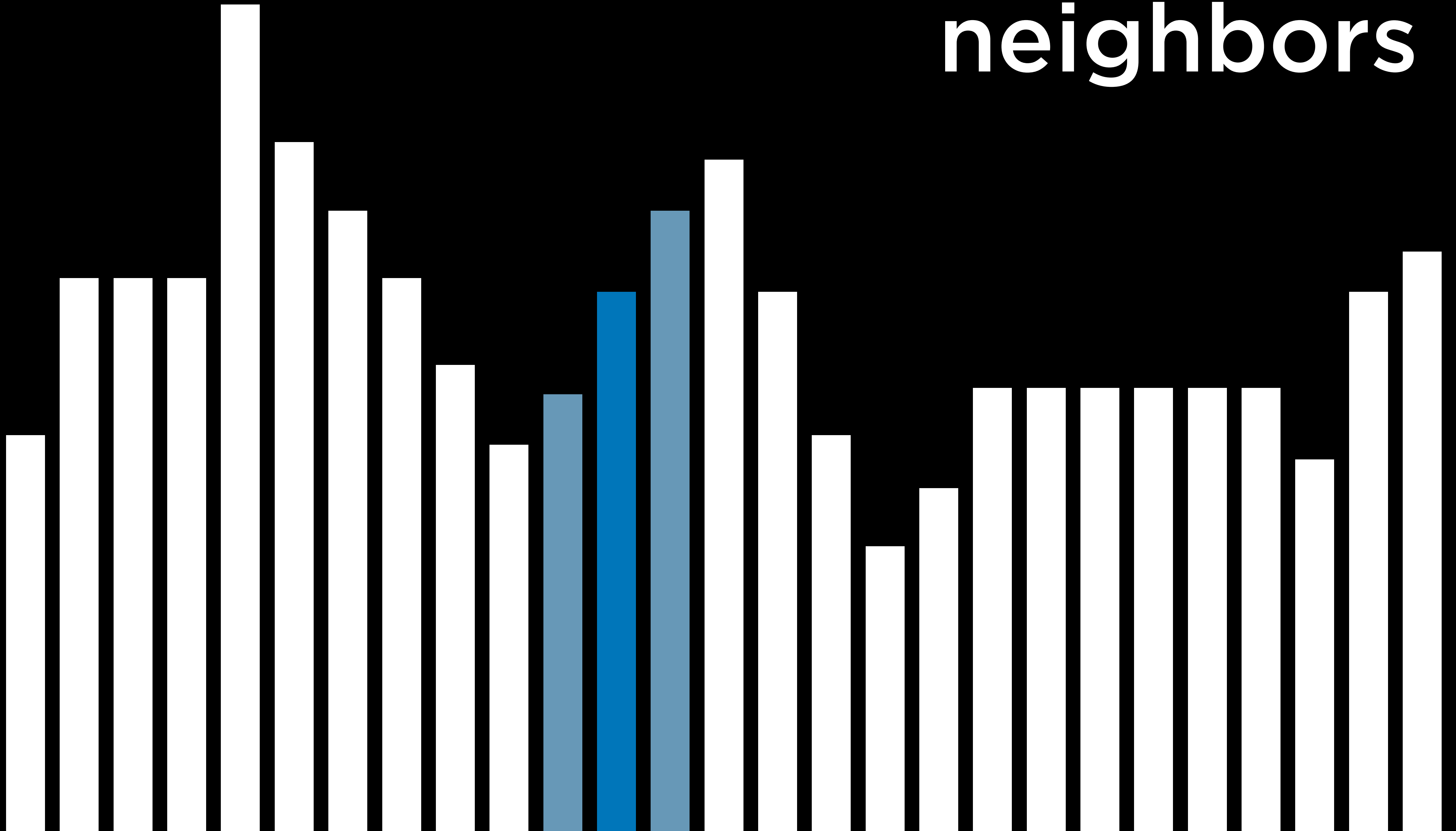




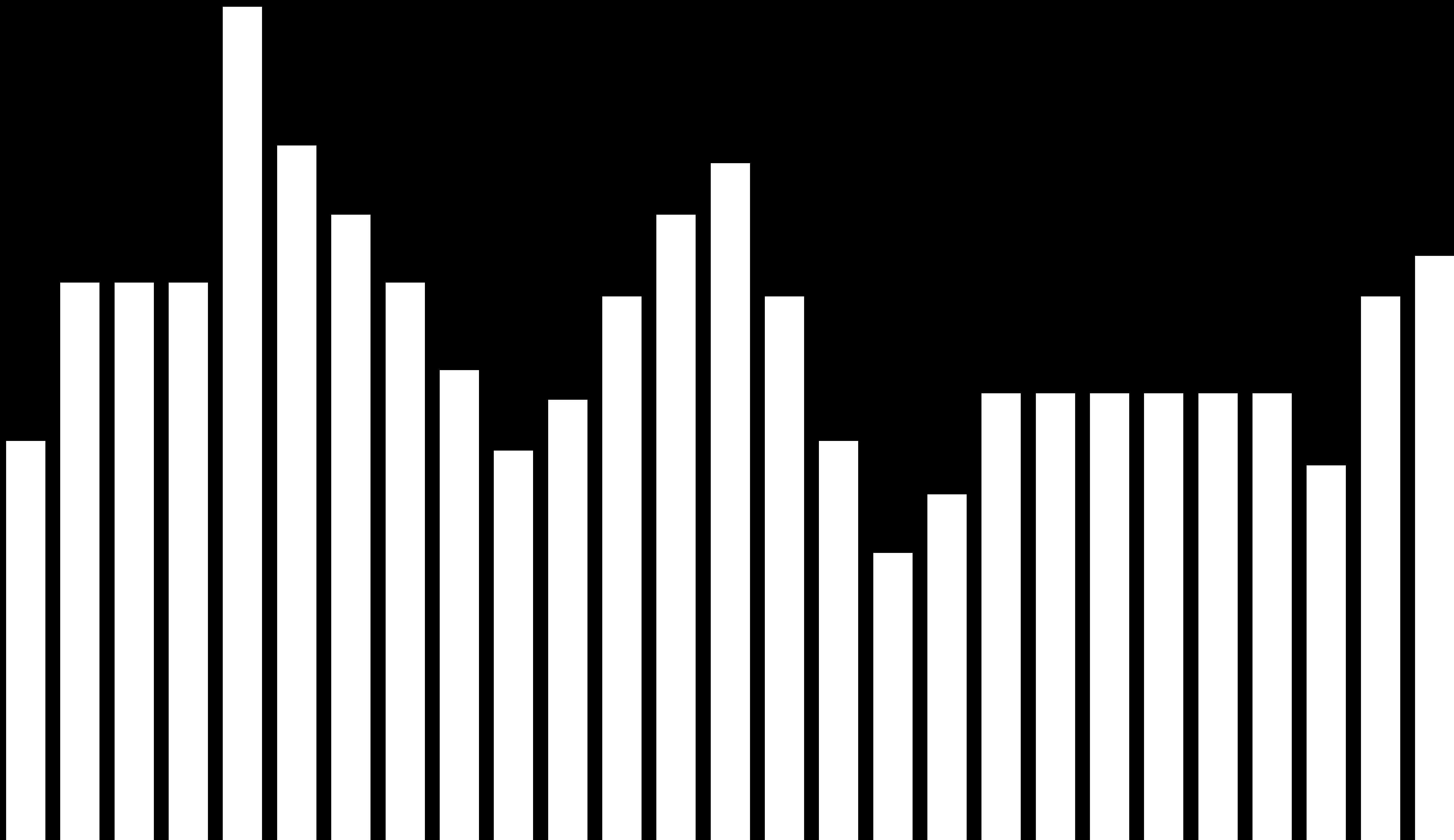
current state

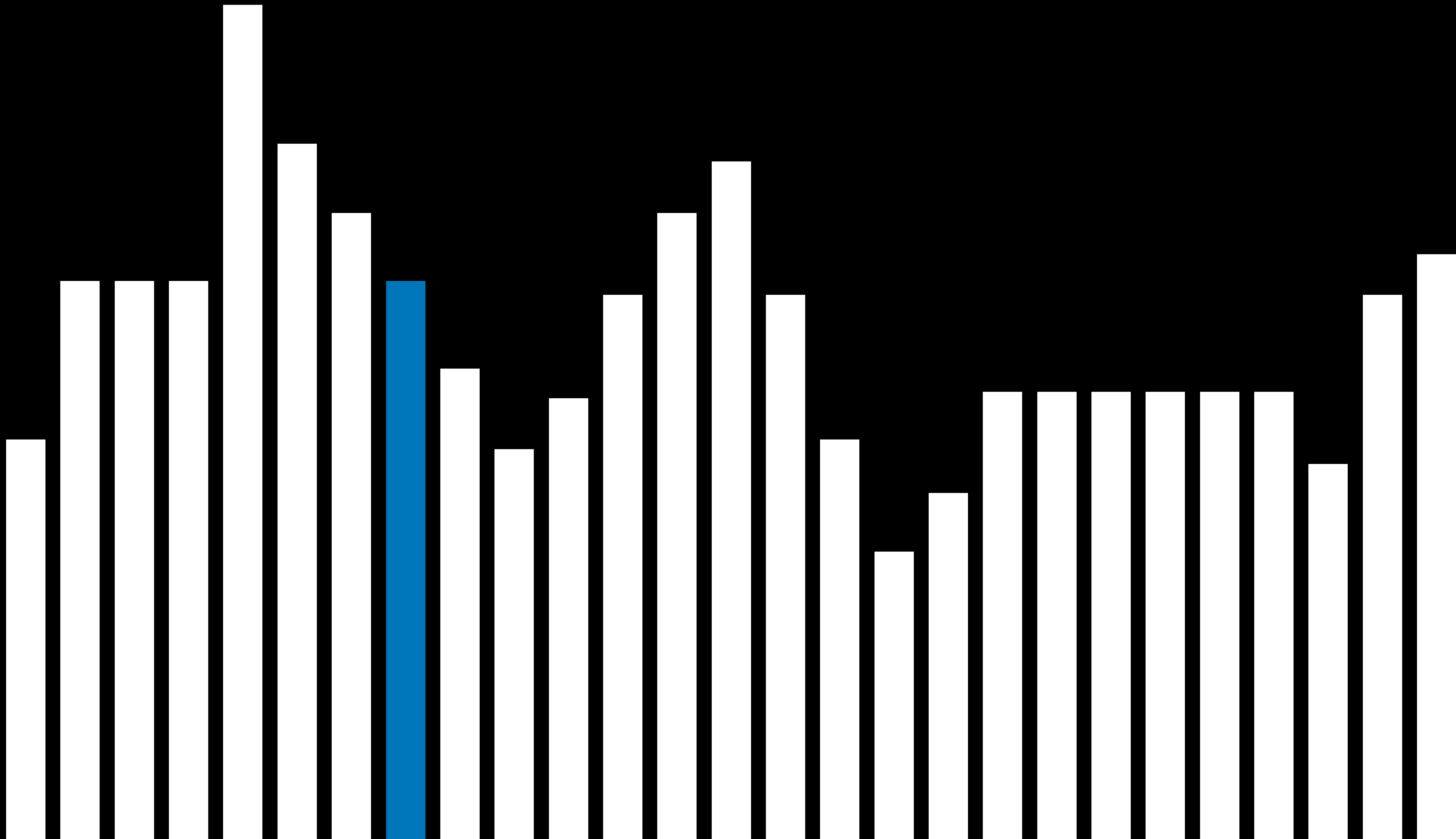


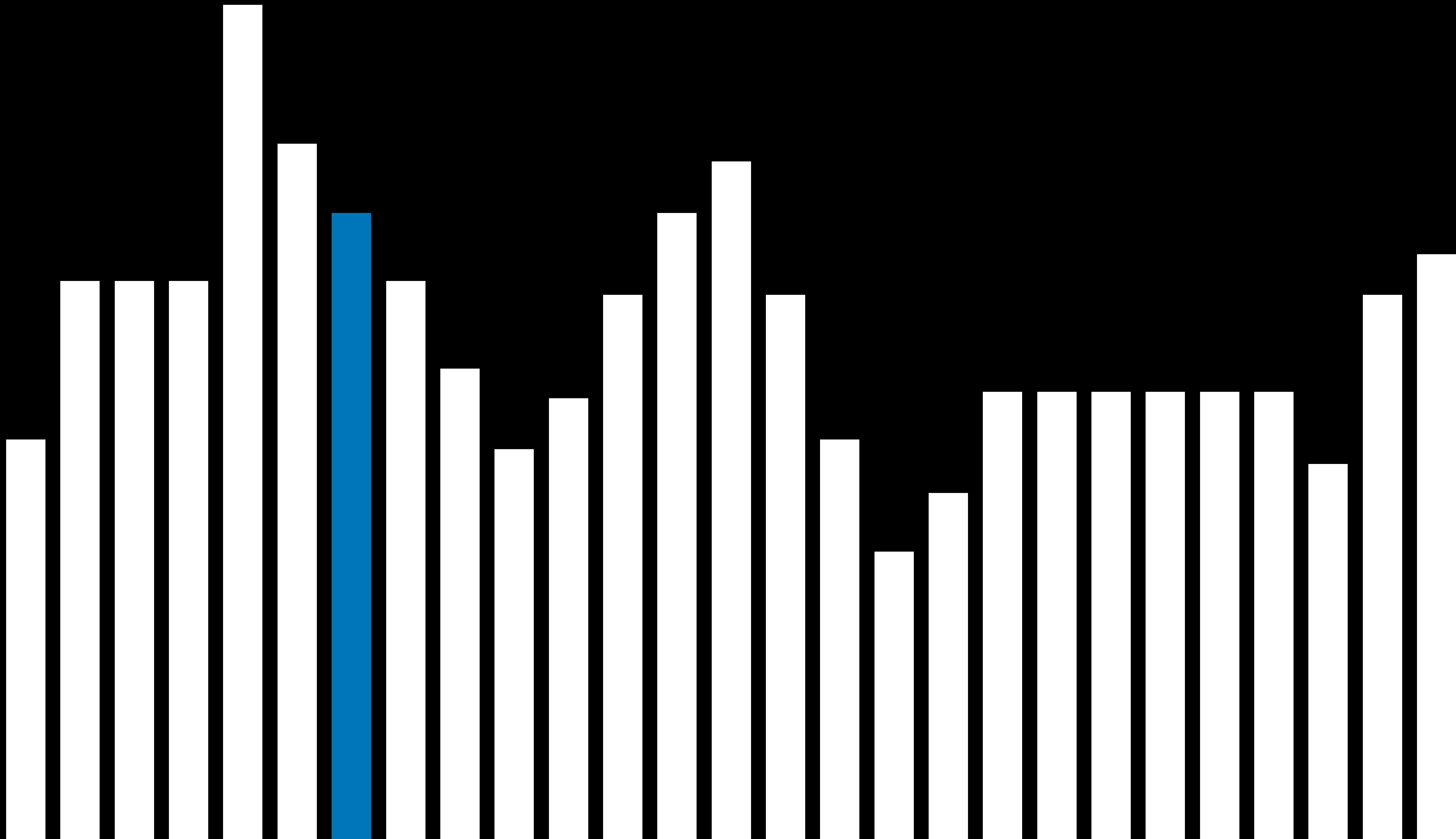
neighbors

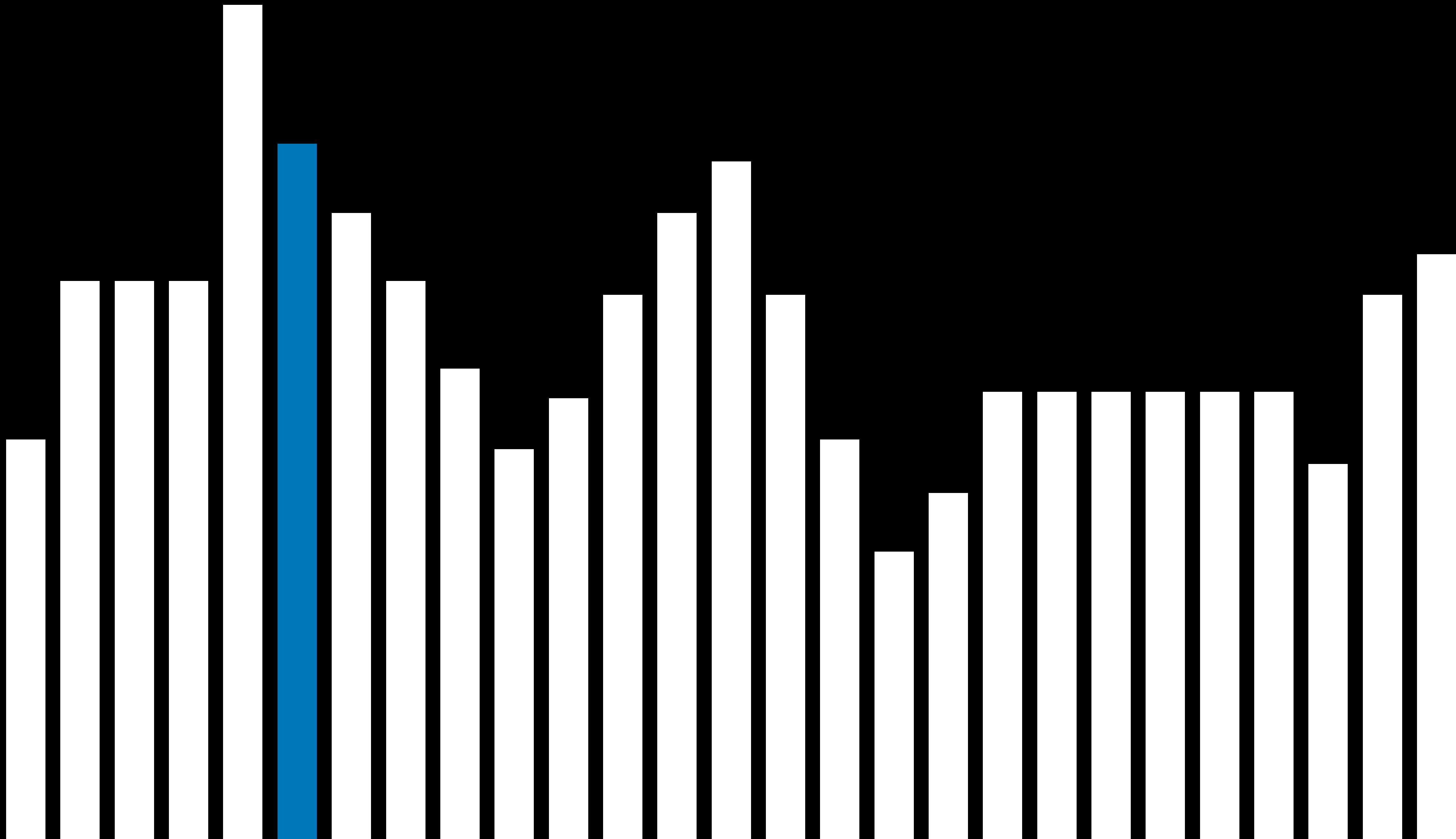


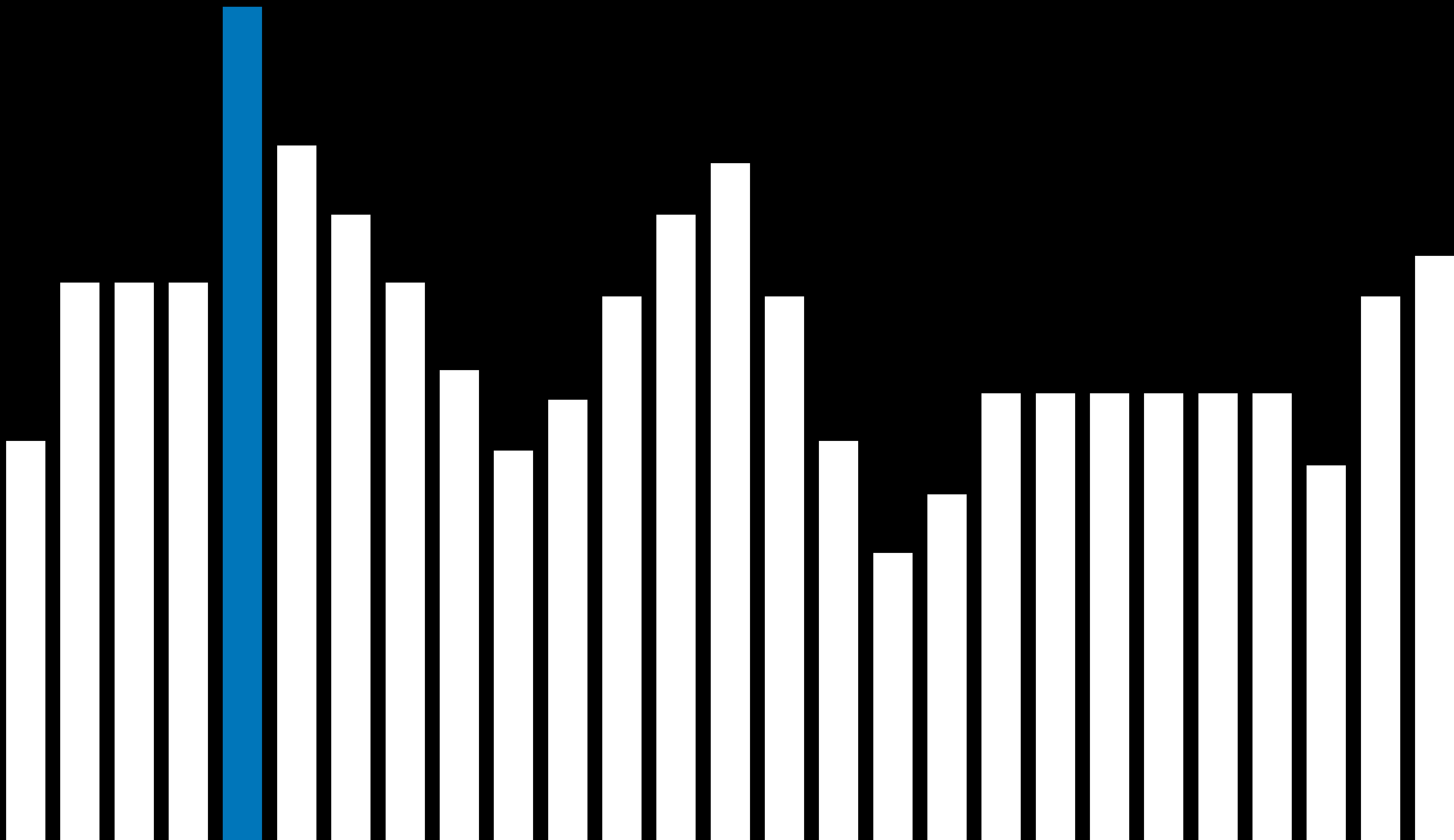
Hill Climbing

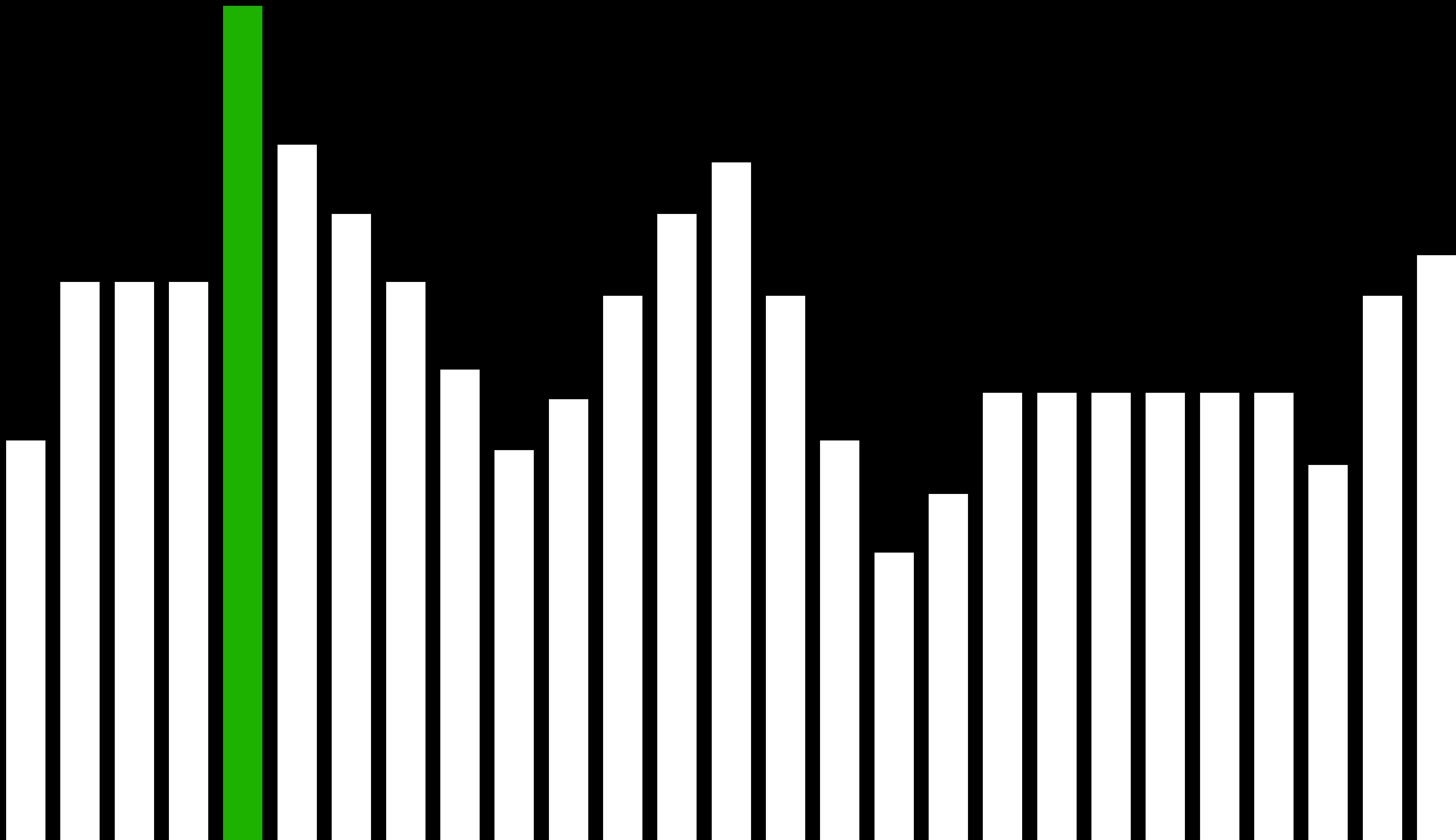


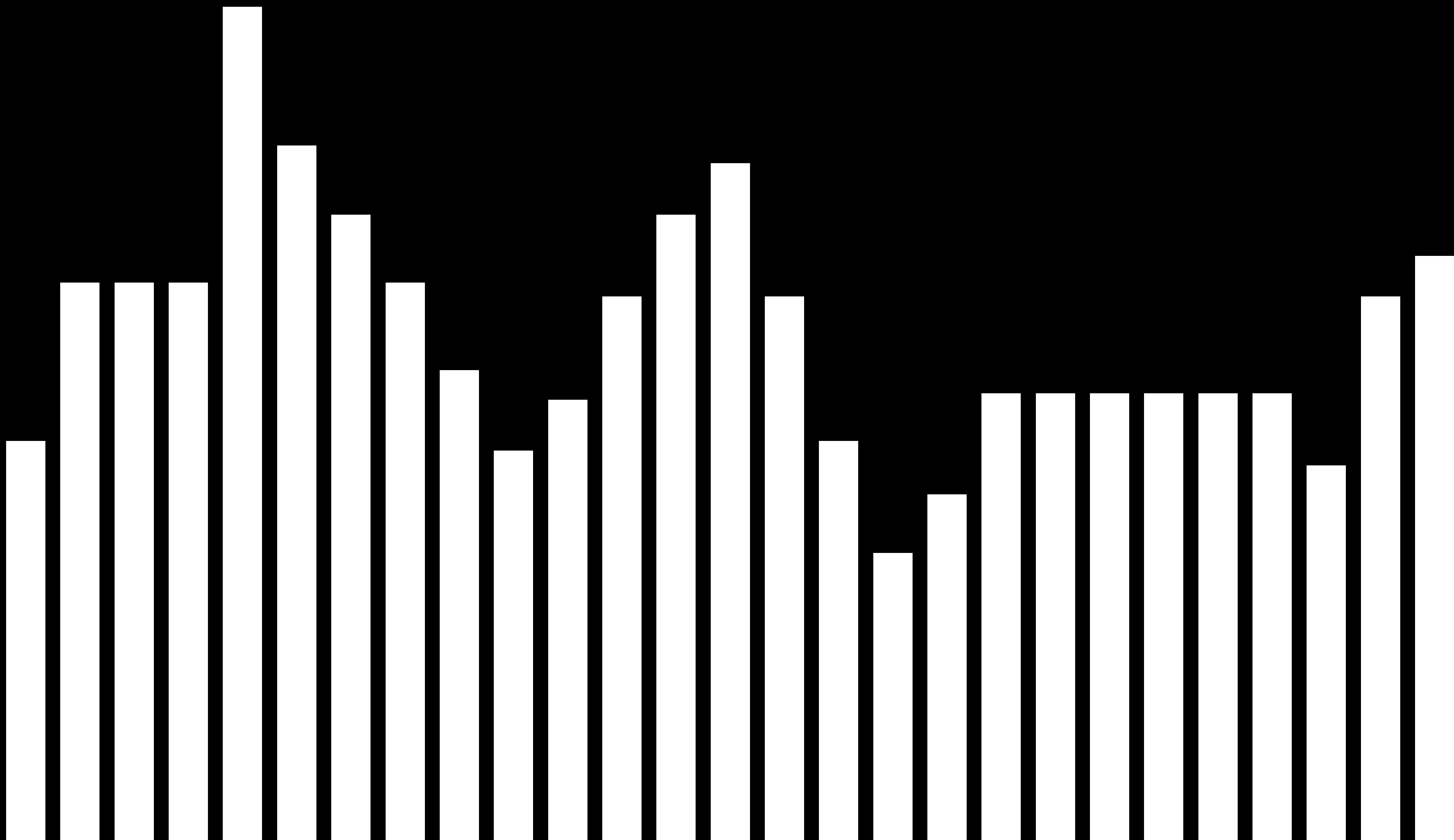


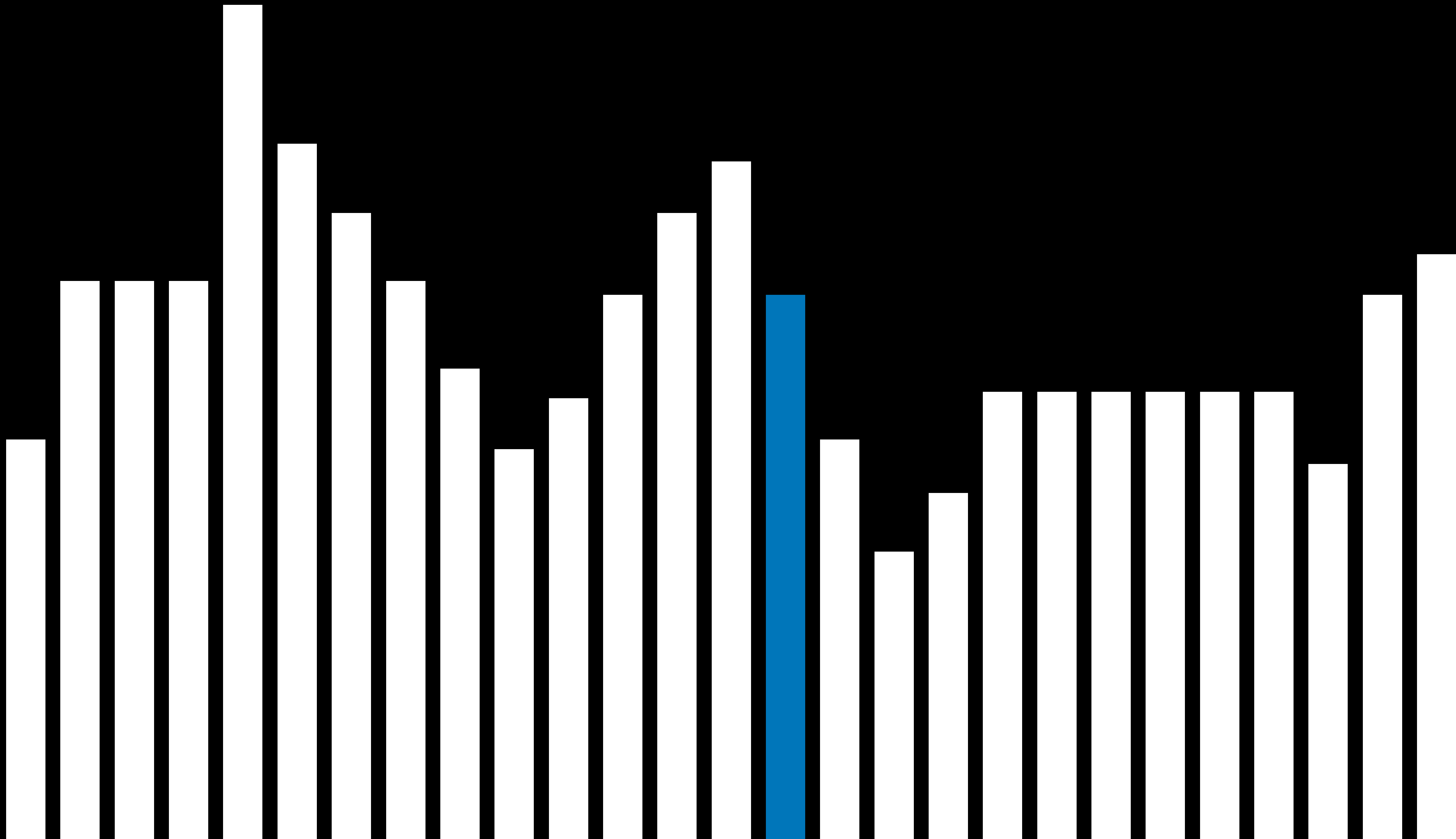


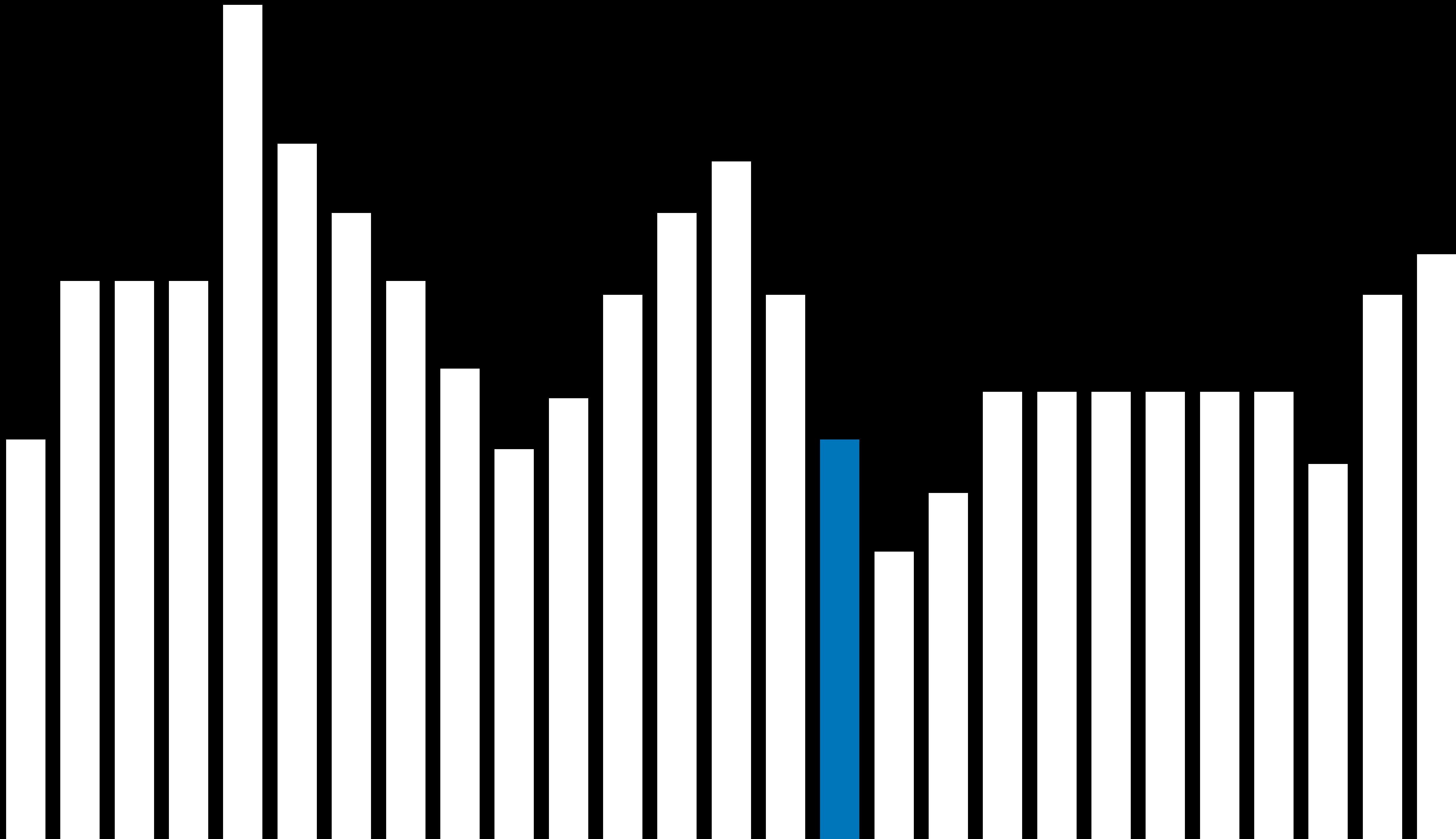


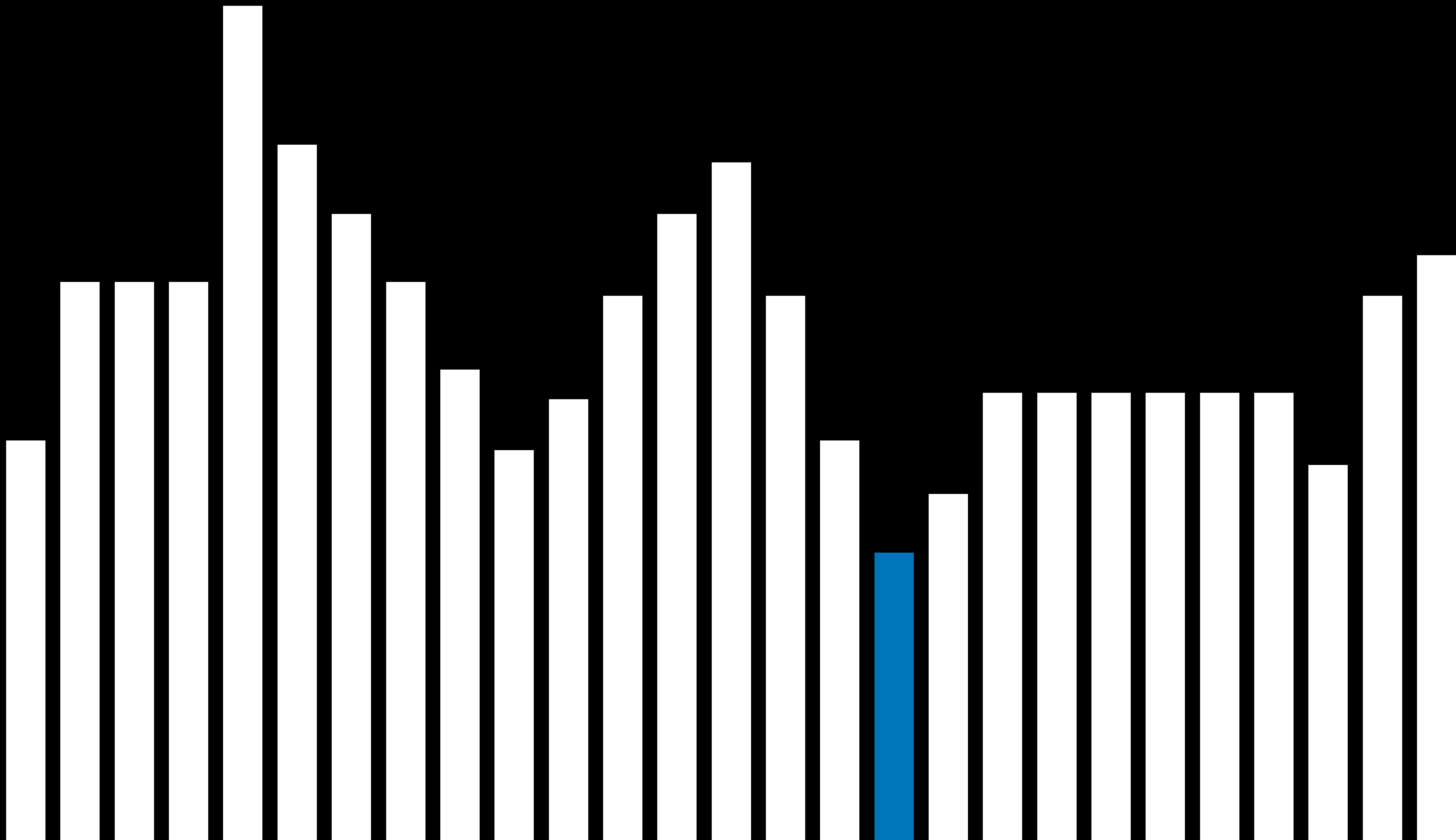


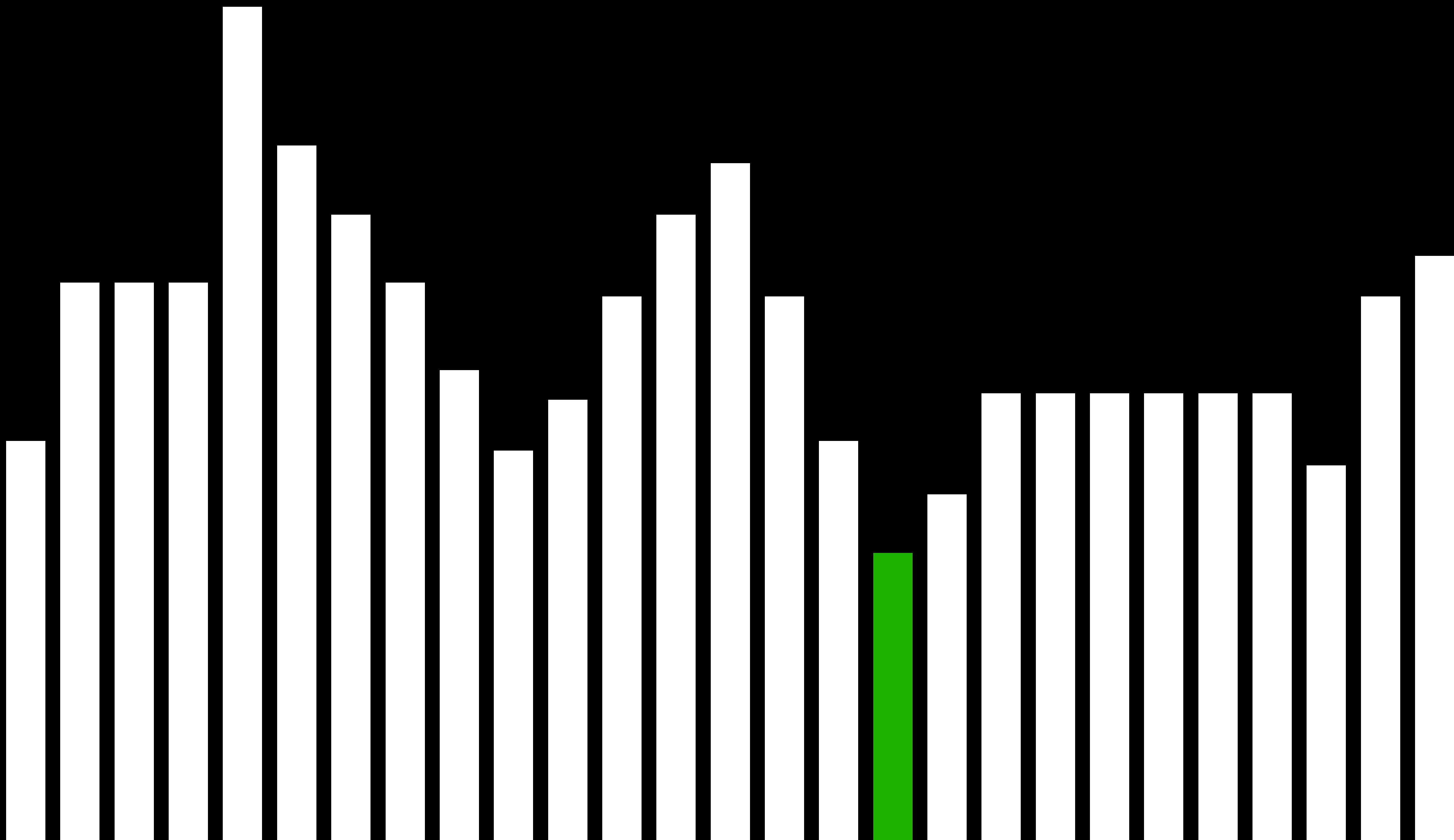








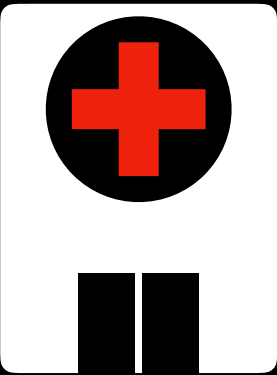
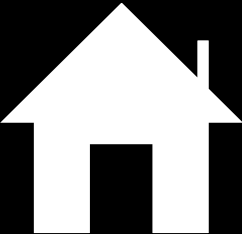
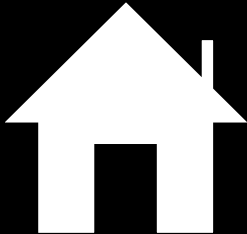

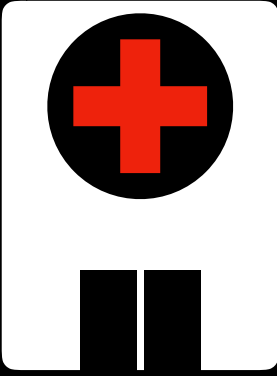
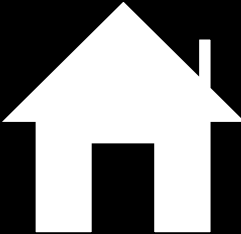




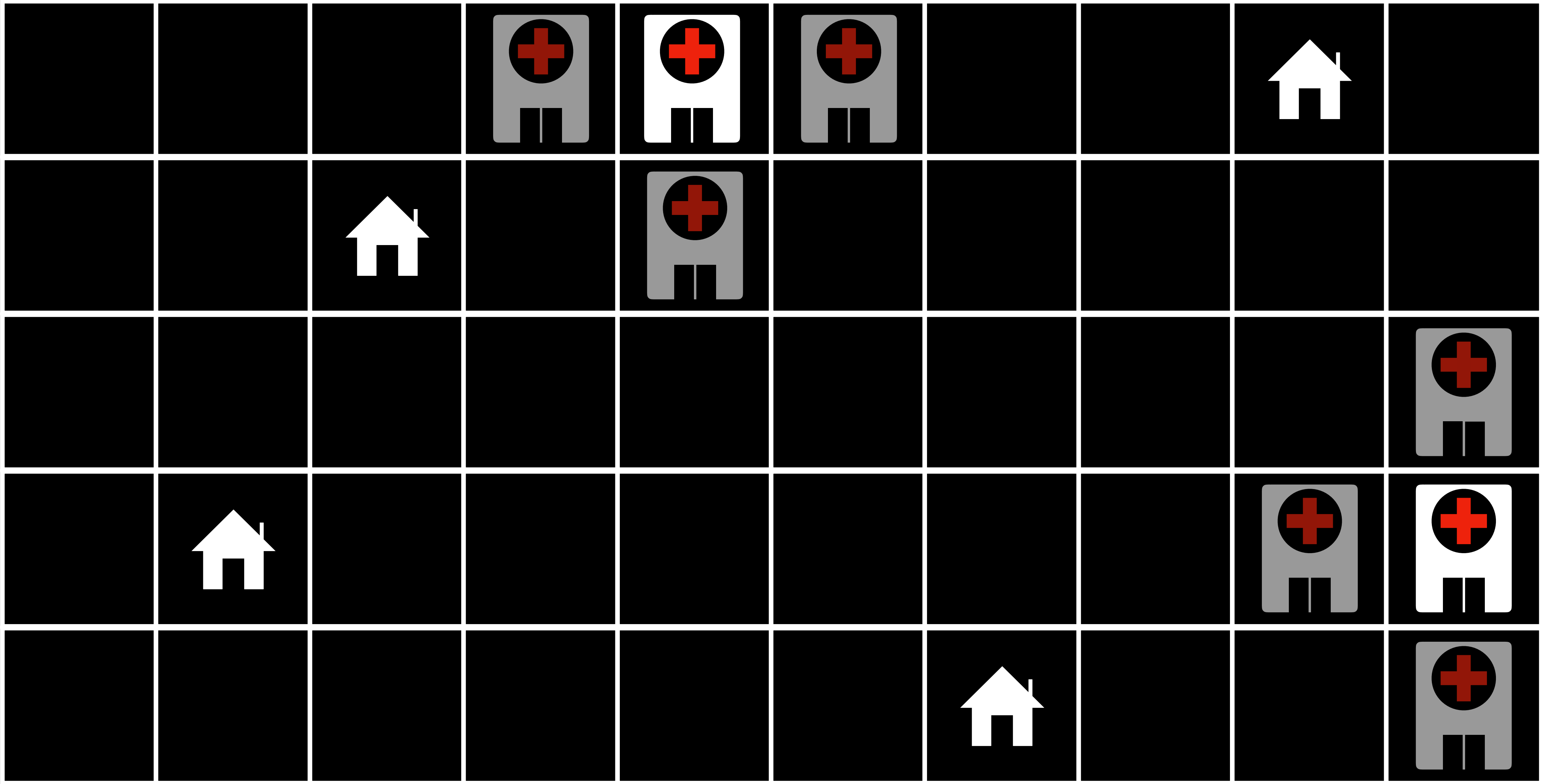
Hill Climbing

```
function HILL-CLIMB(problem):  
    current = initial state of problem  
    repeat:  
        neighbor = highest valued neighbor of current  
        if neighbor not better than current:  
            return current  
        current = neighbor
```

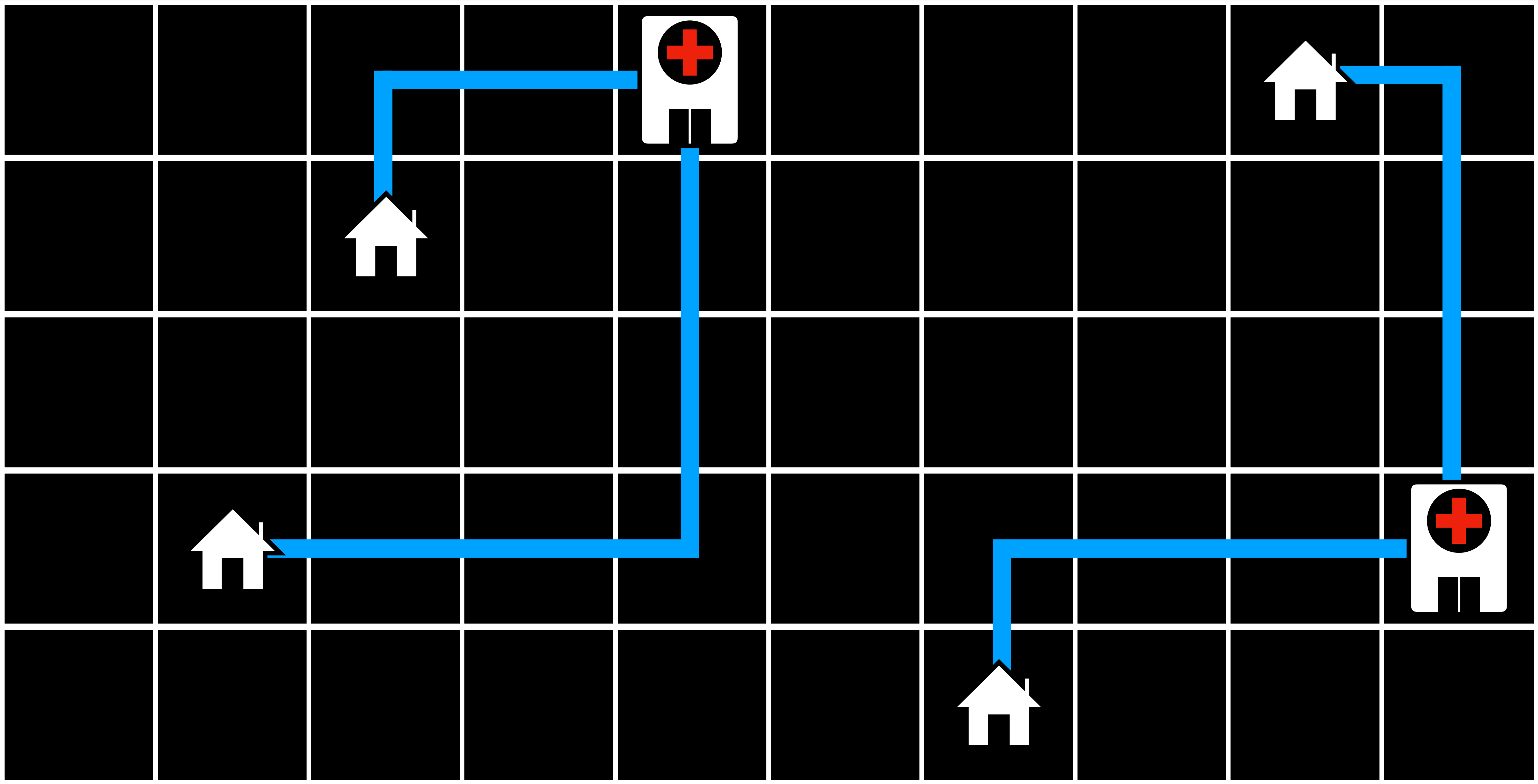
Cost: 17

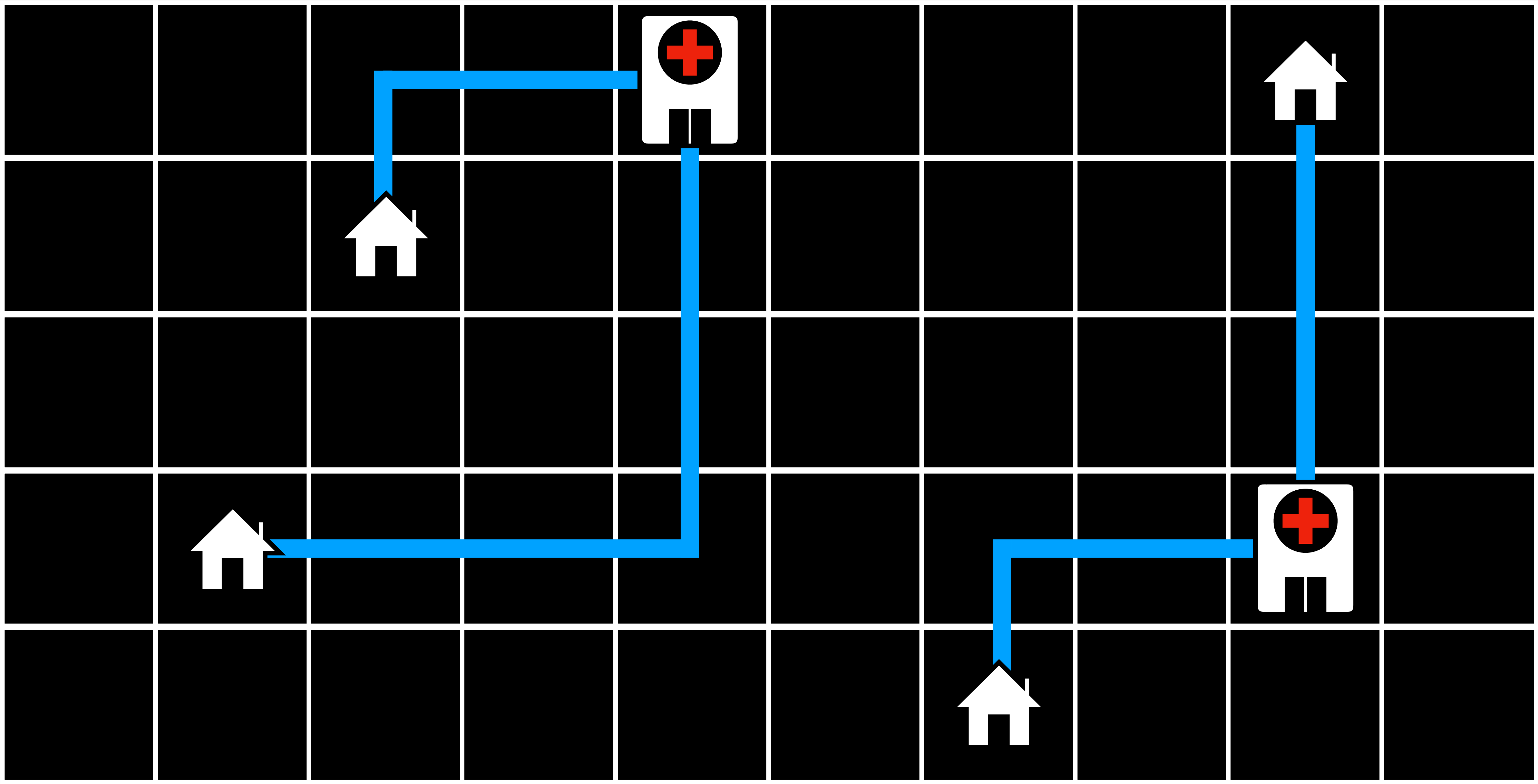
Cost: 17



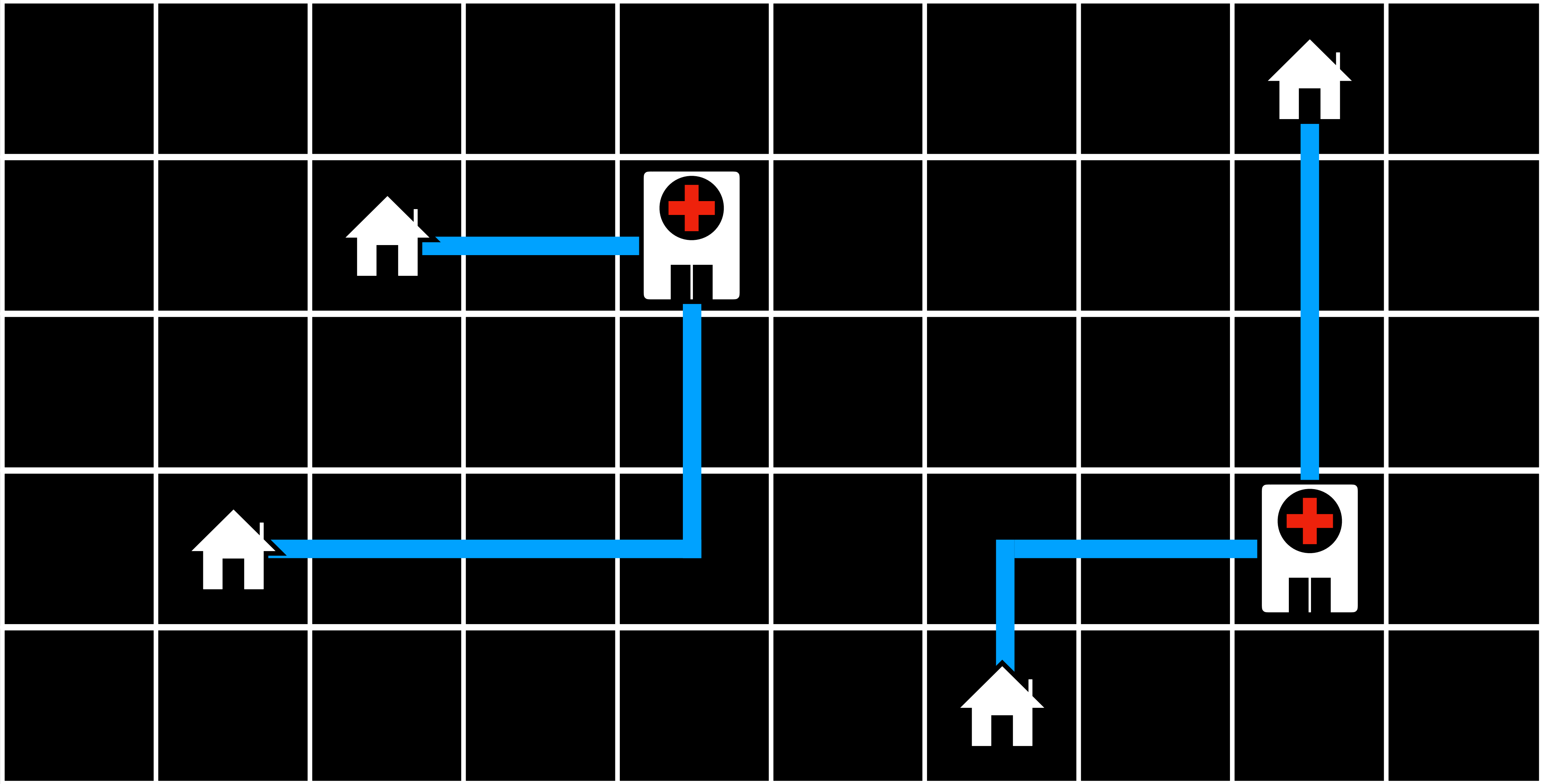
Cost: 17



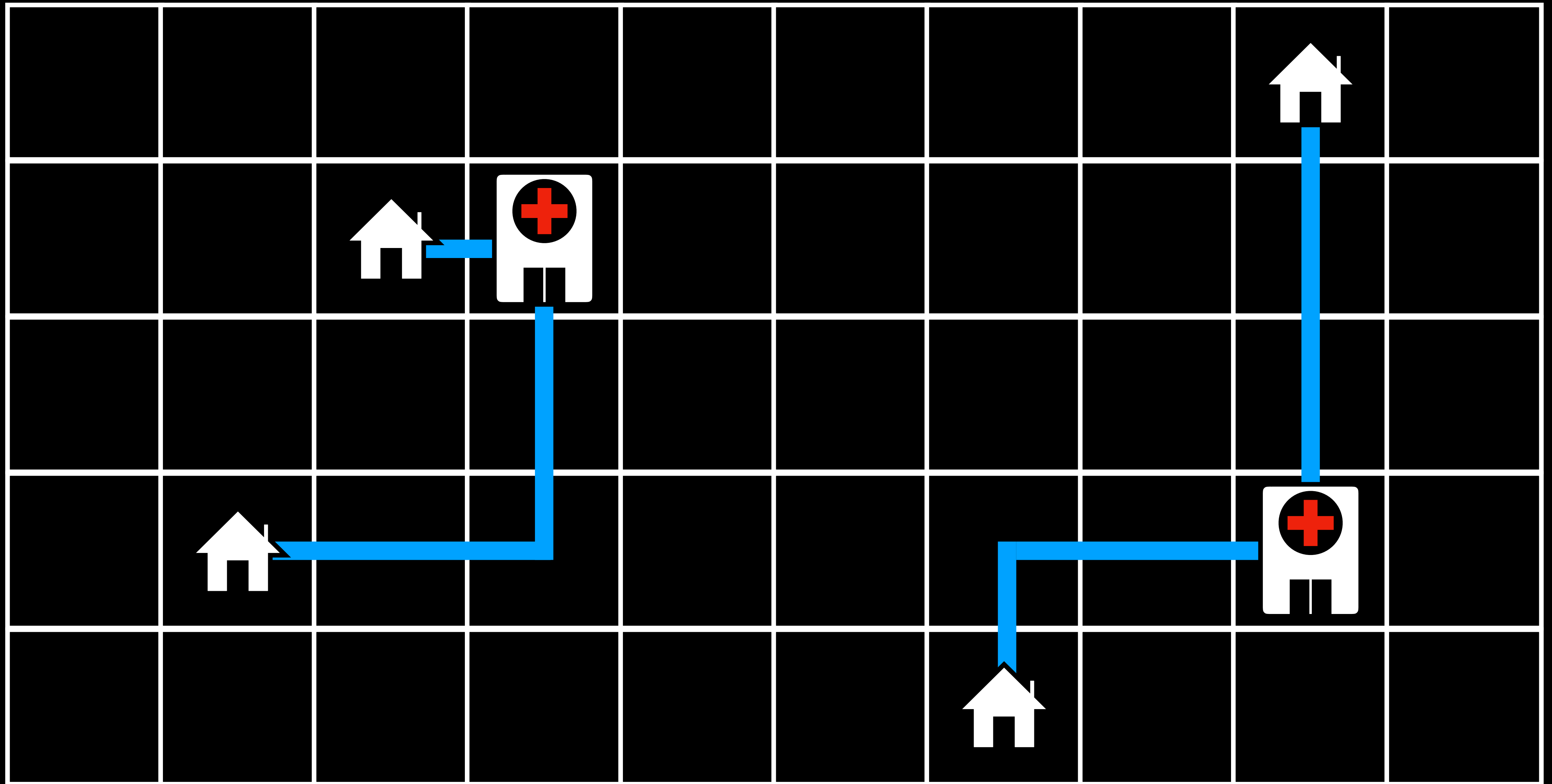
Cost: 15



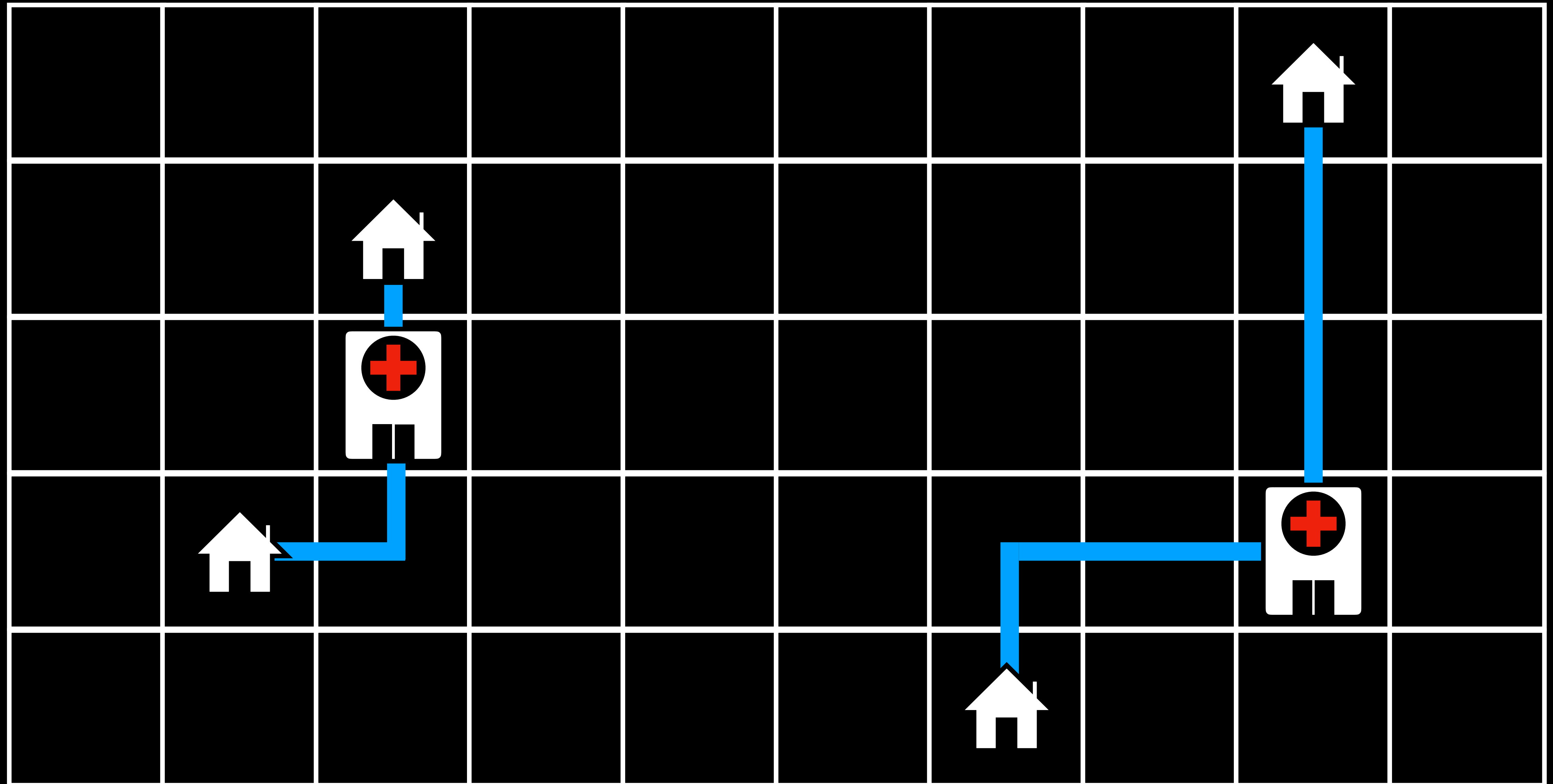
Cost: 13

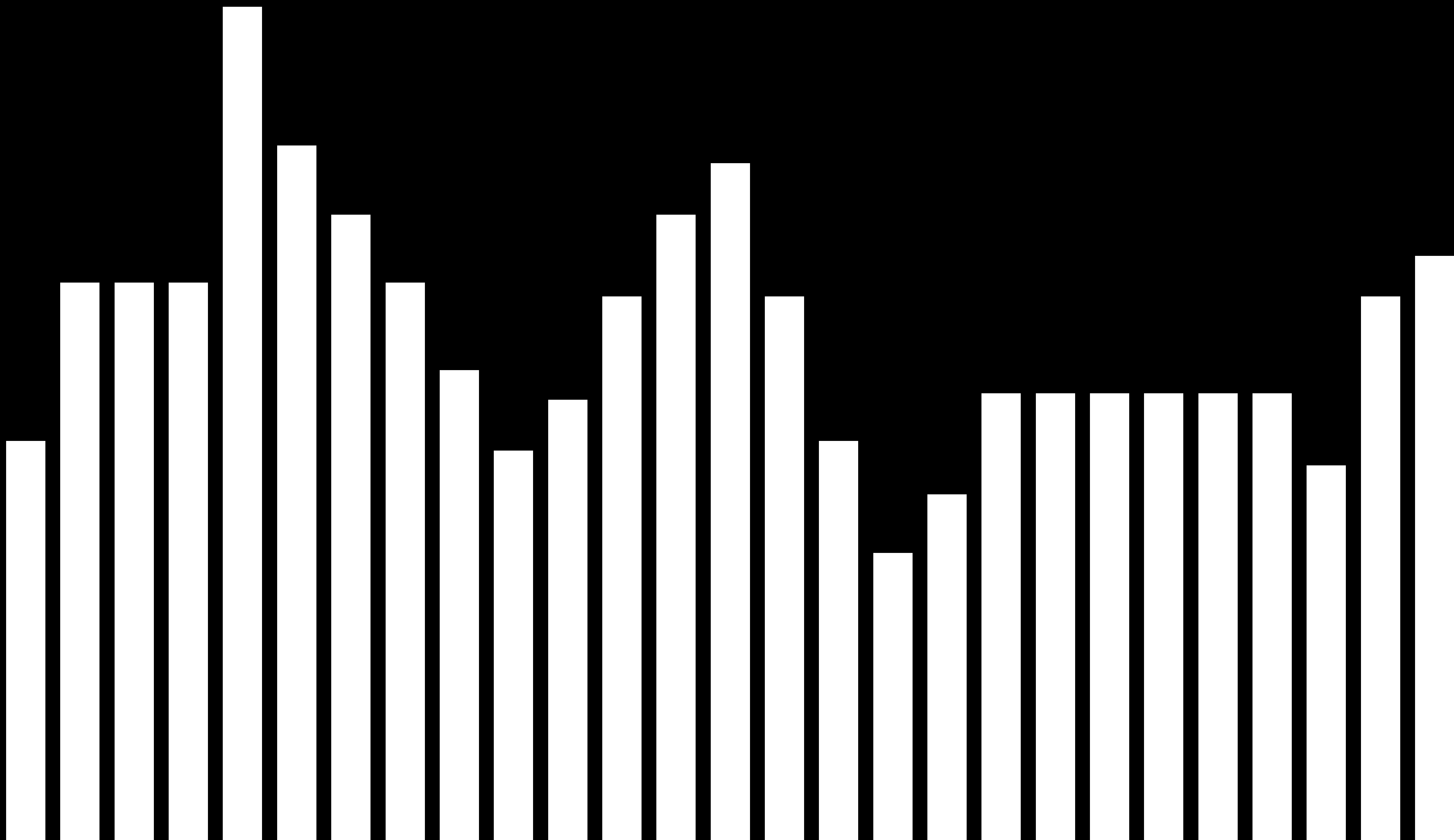


Cost: 11



Cost: 9





global maximum



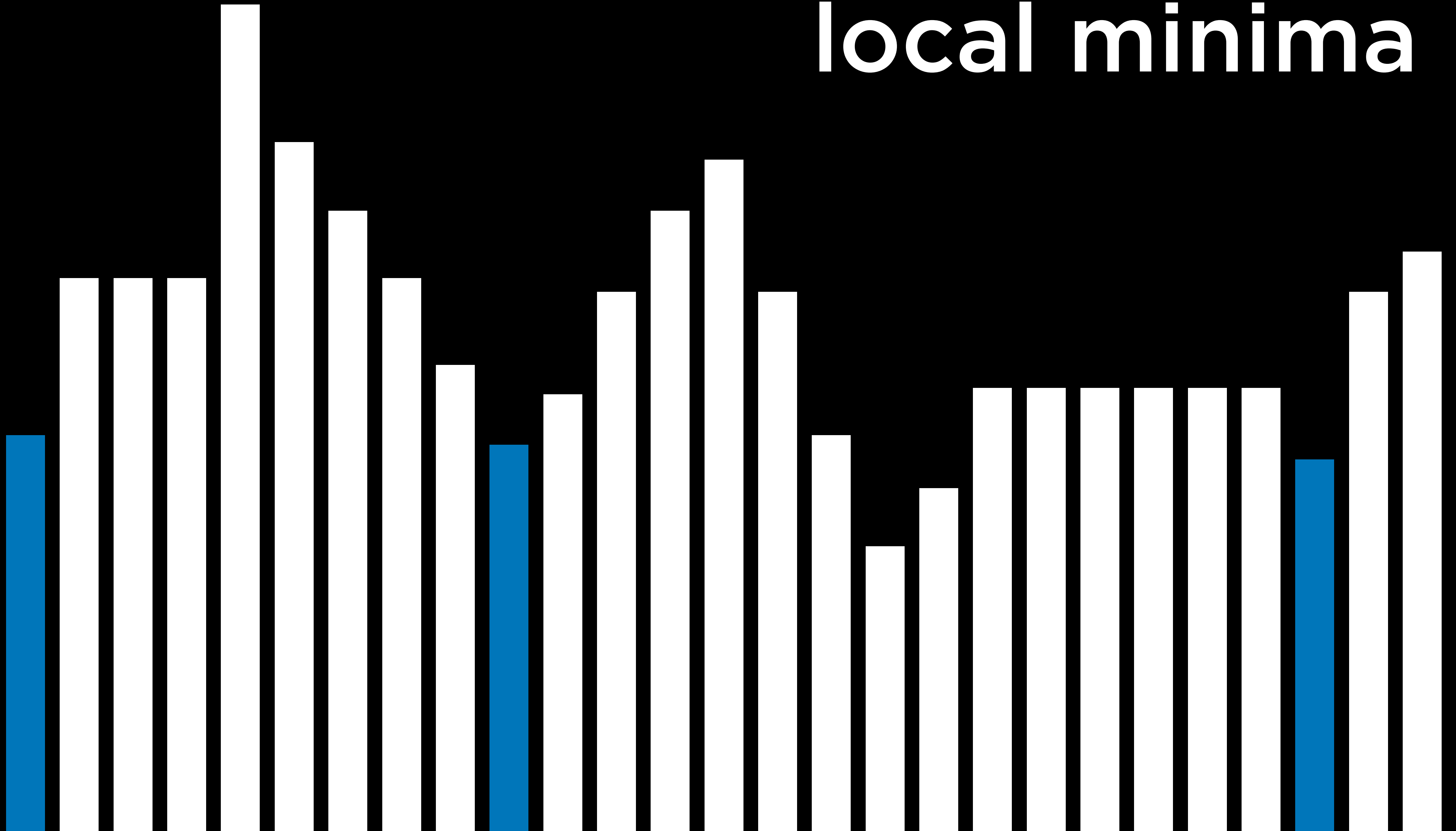
local maxima

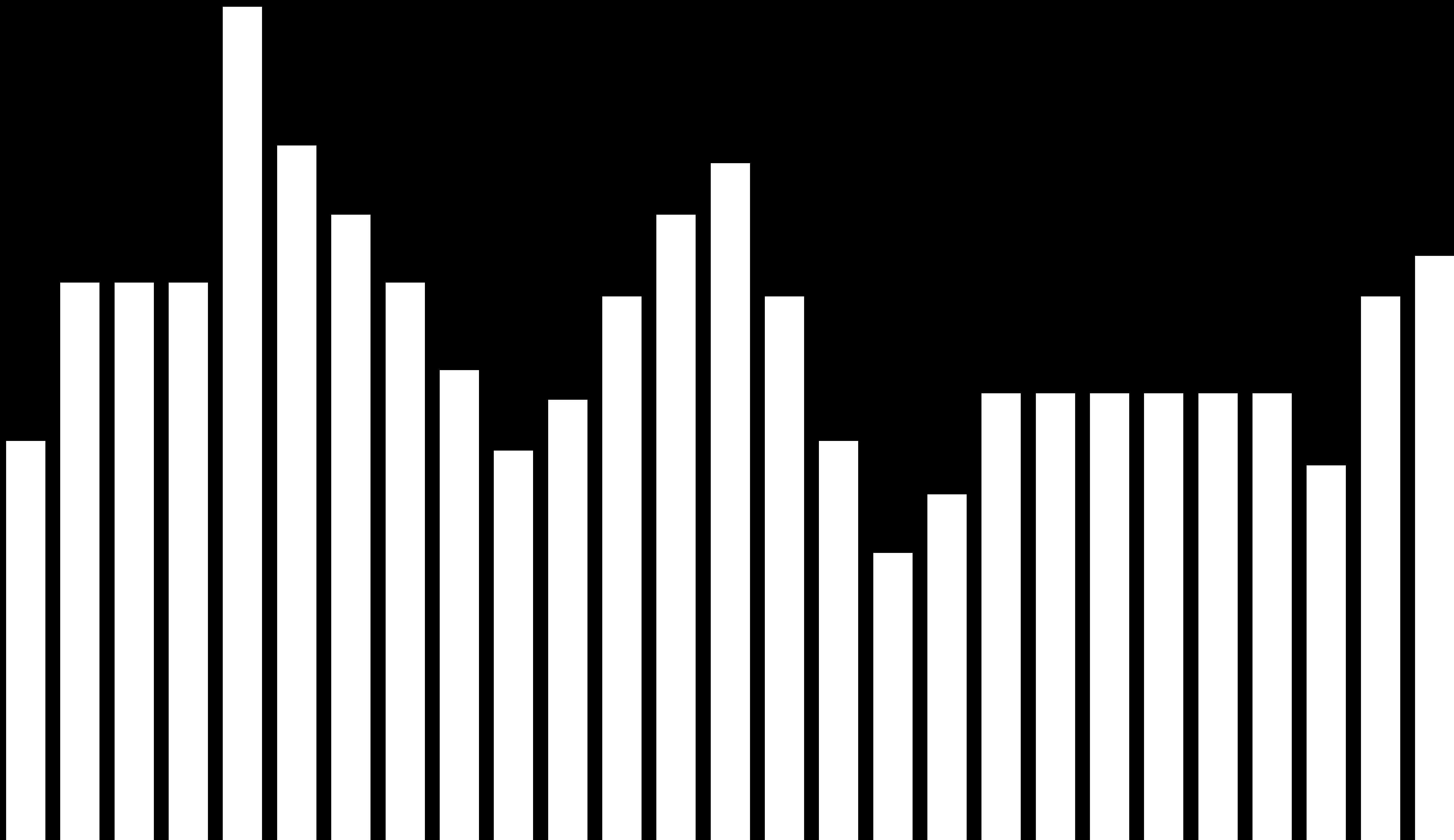


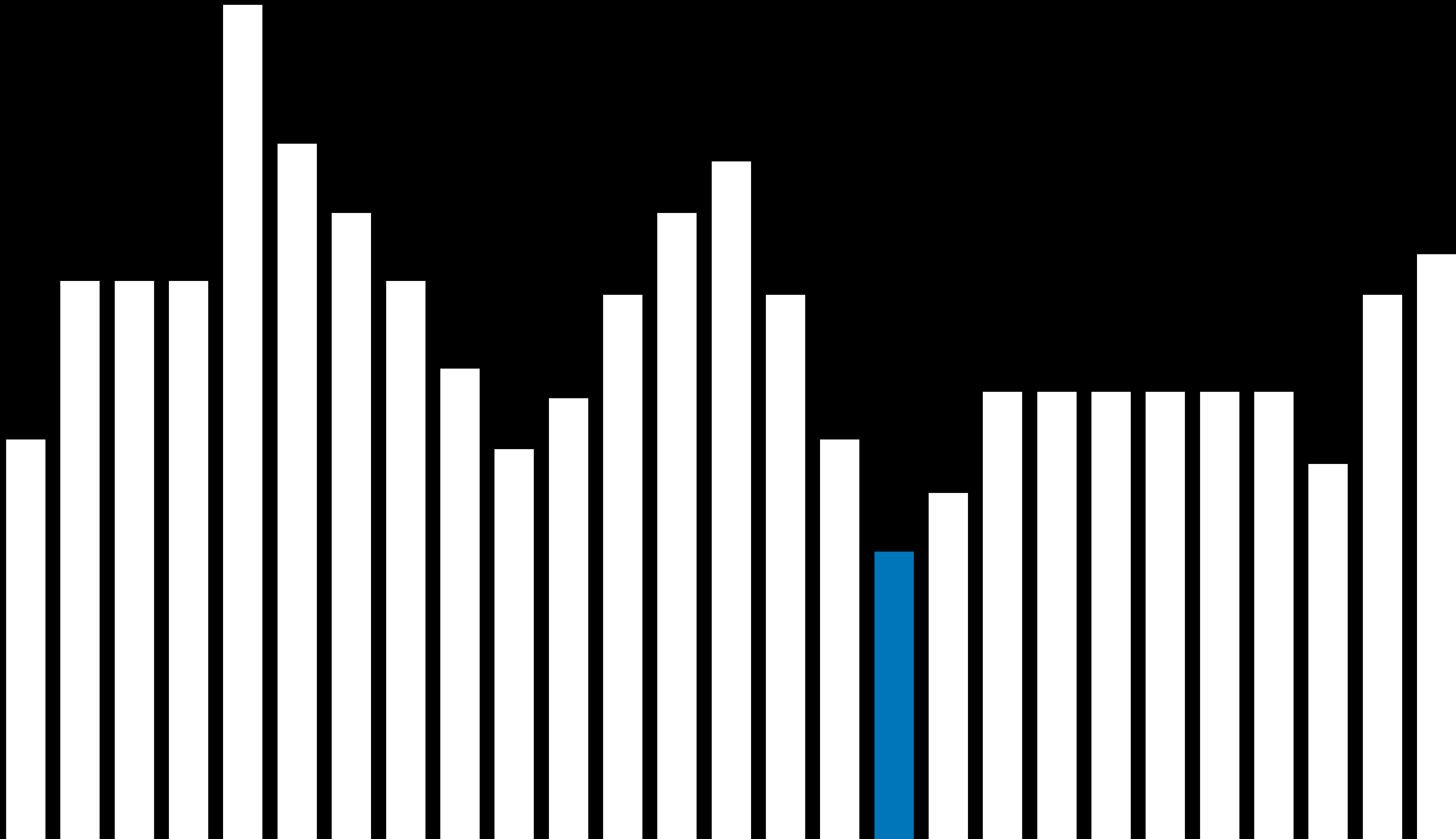
global minimum

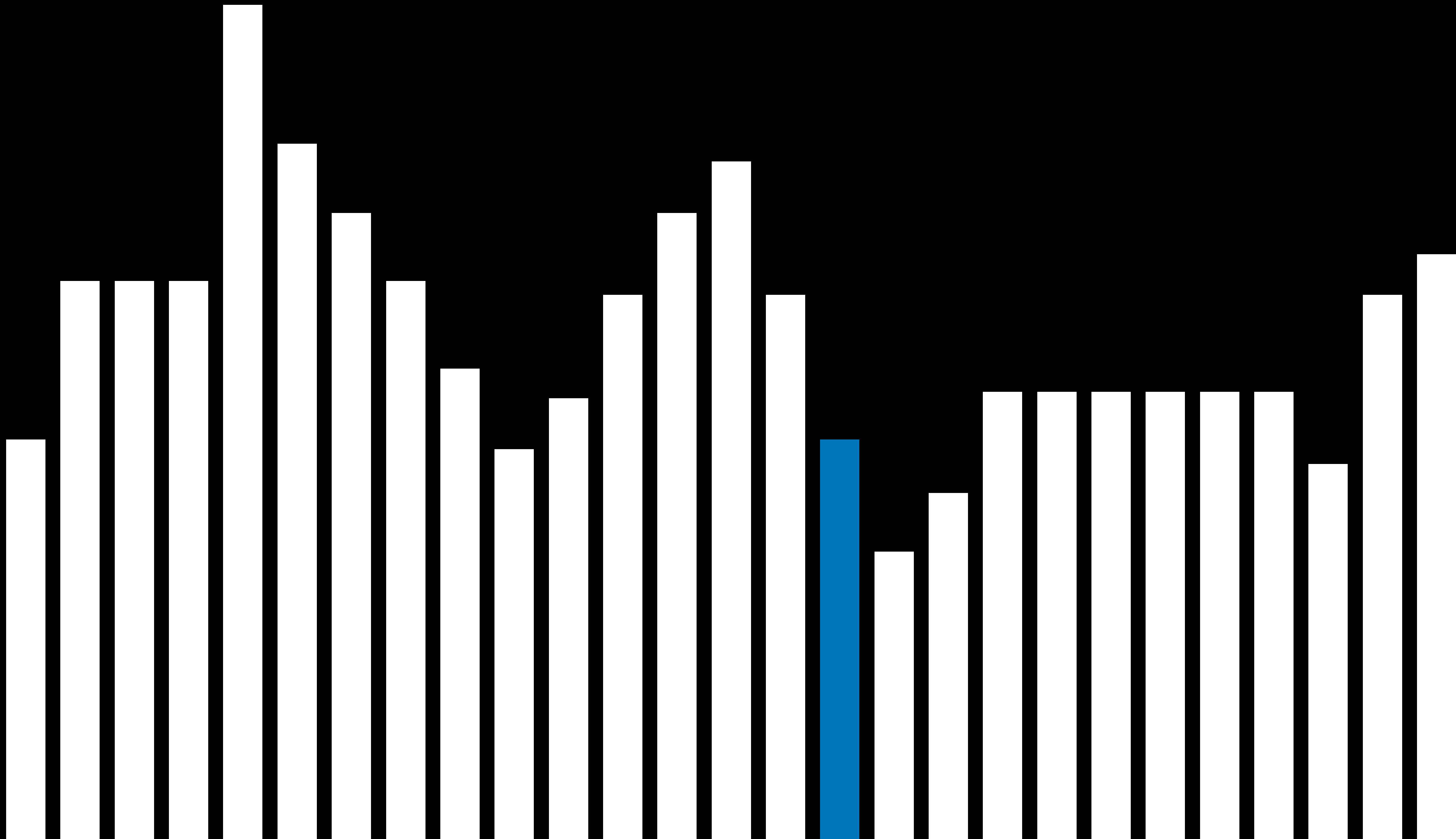


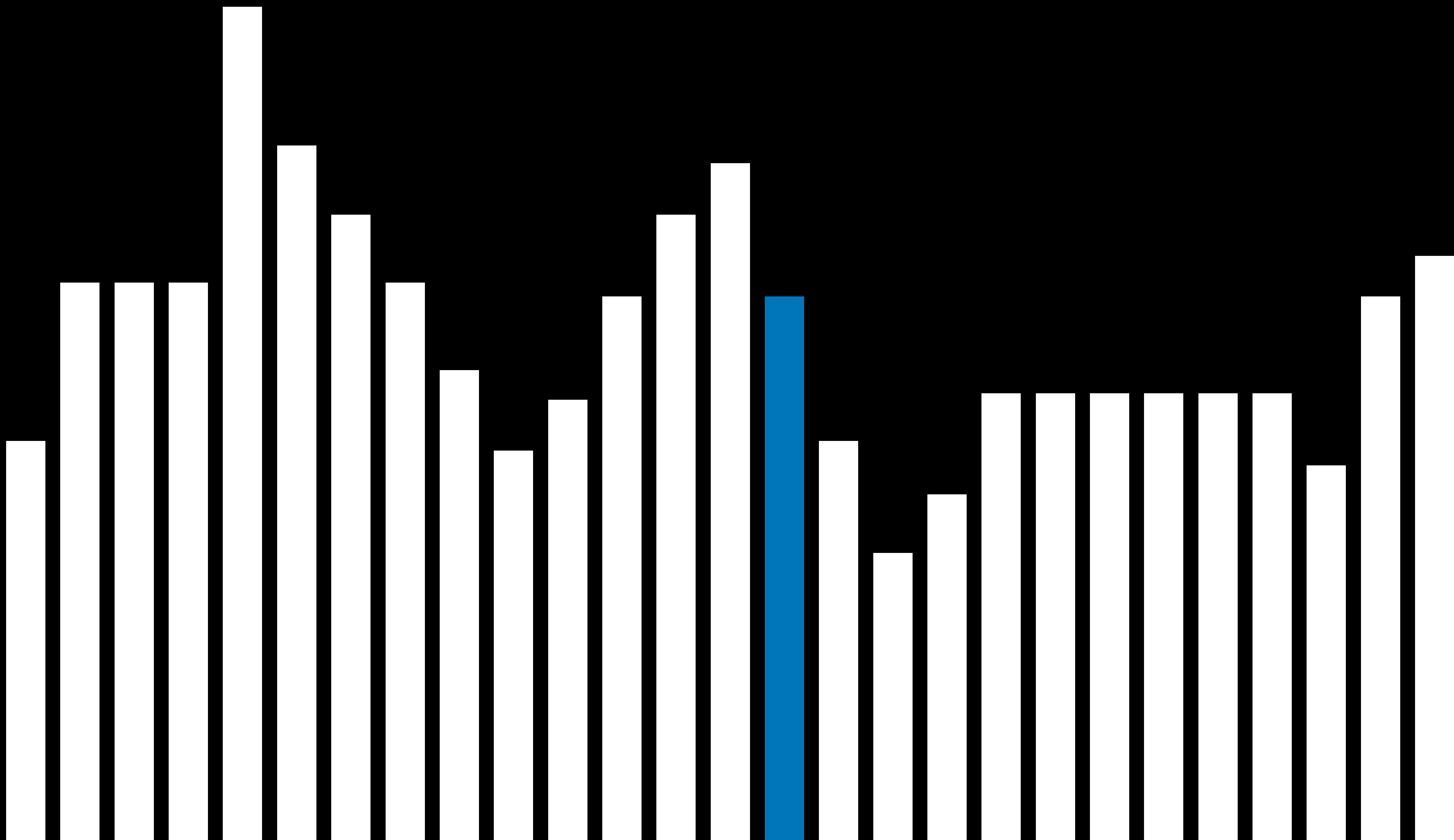
local minima

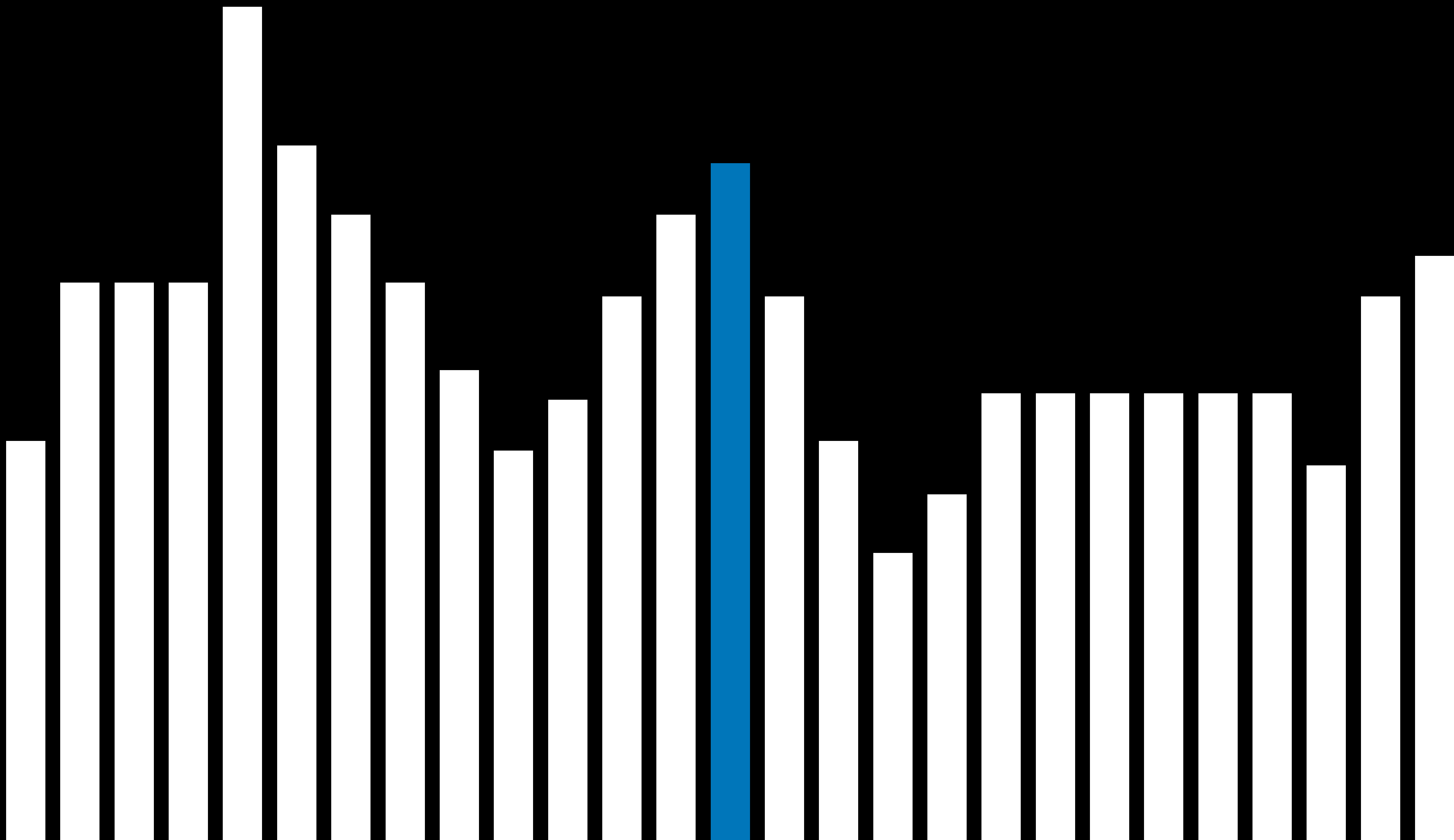








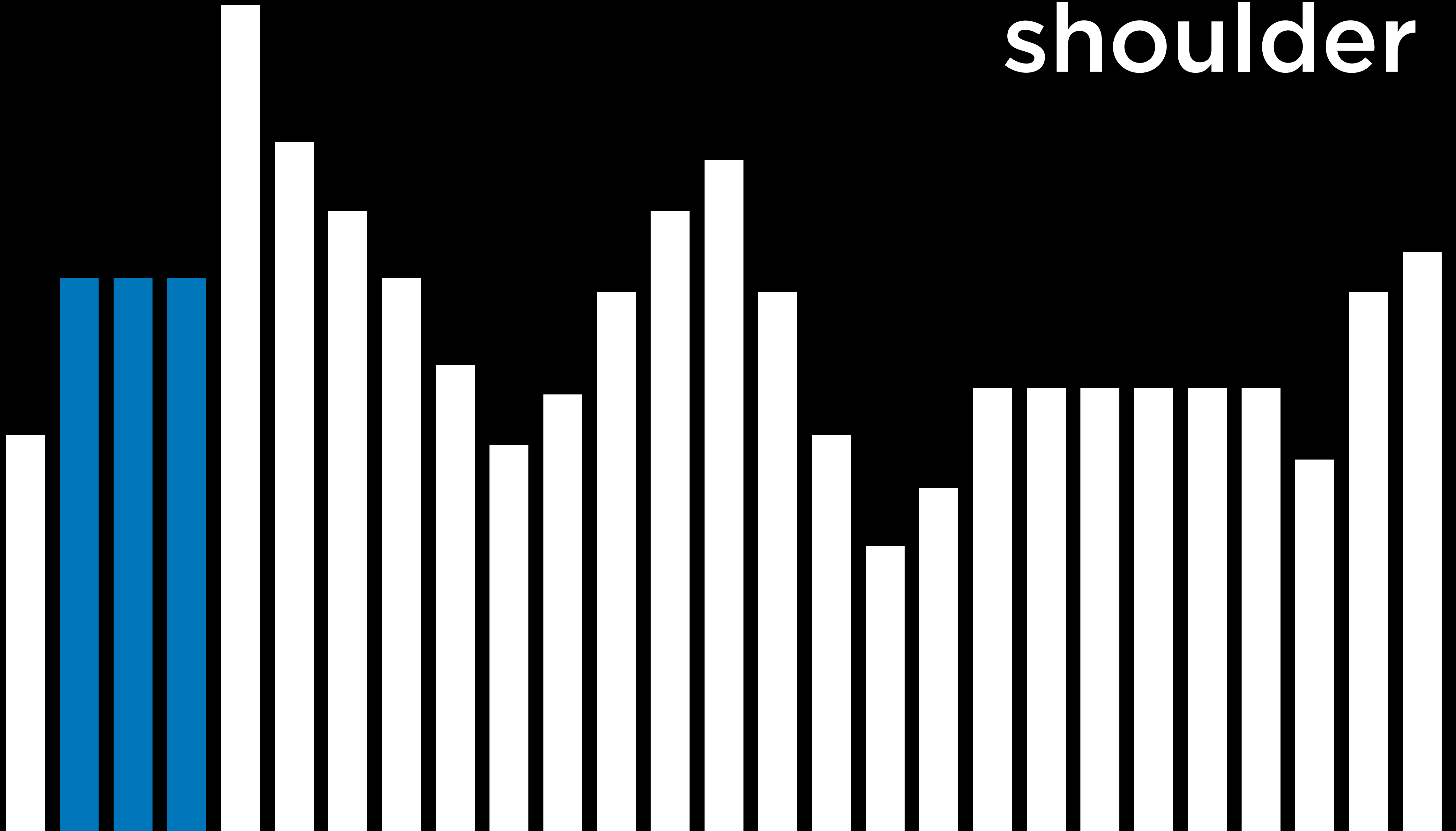




flat local maximum



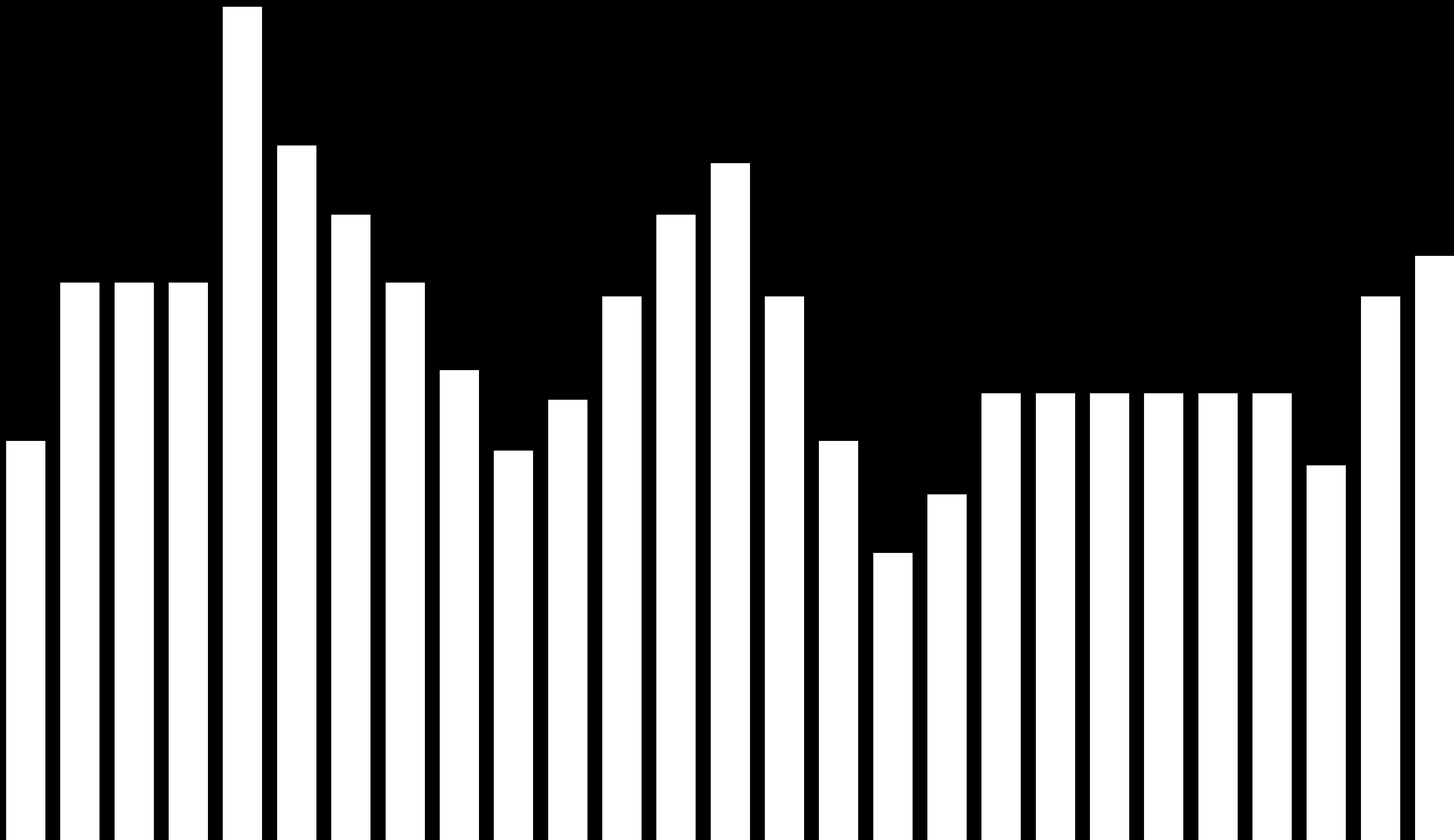
shoulder

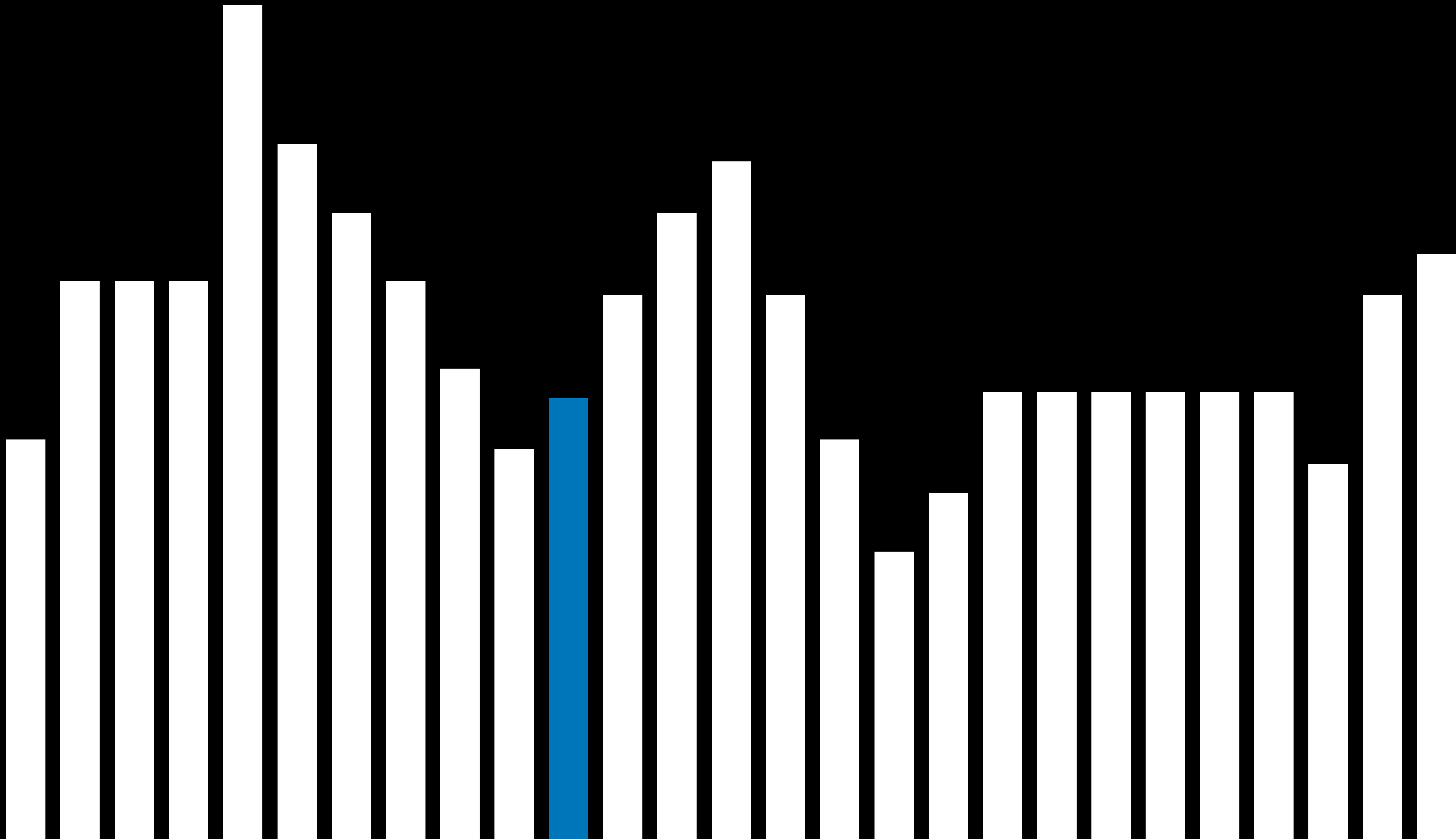


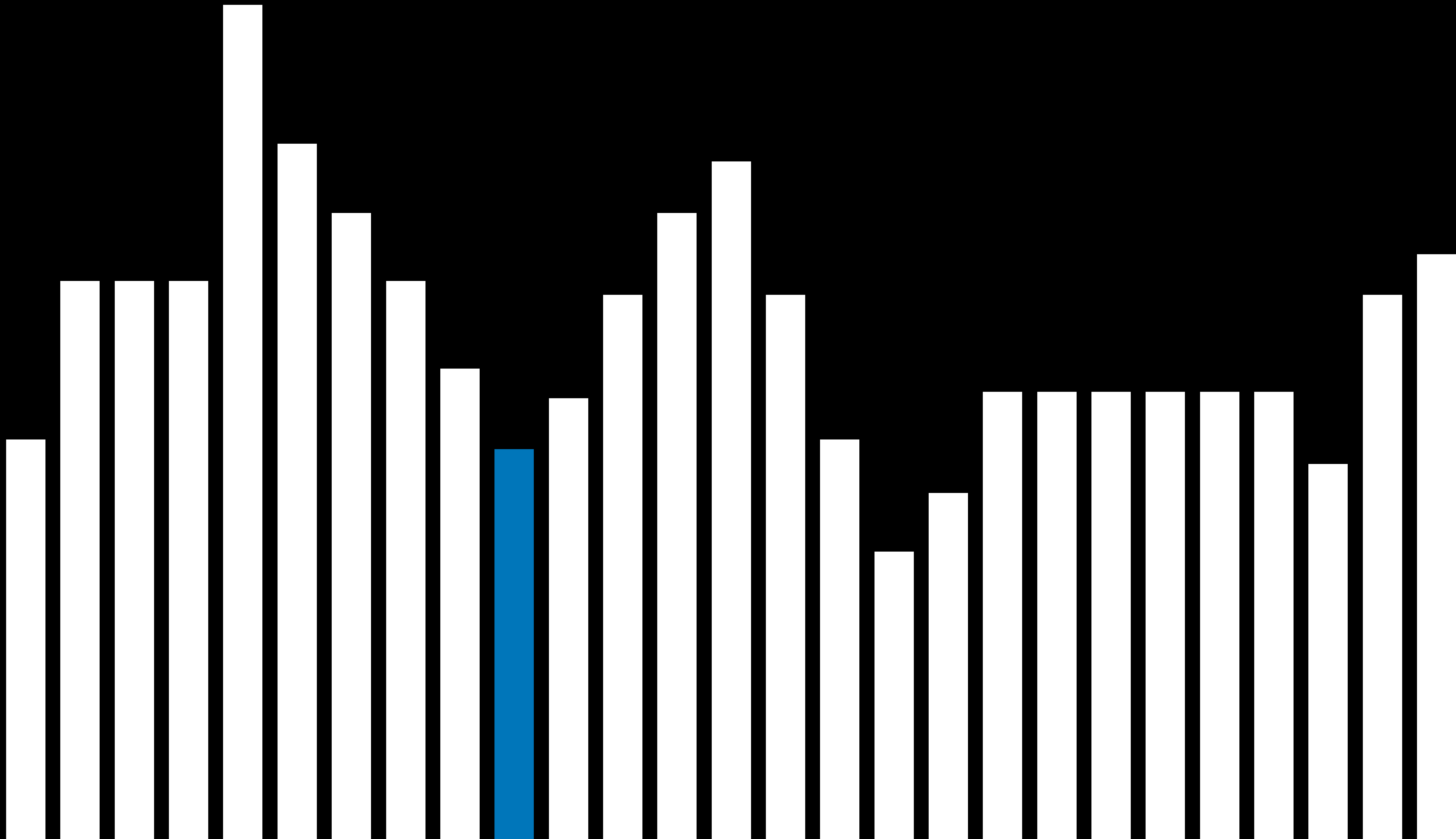
Hill Climbing Variants

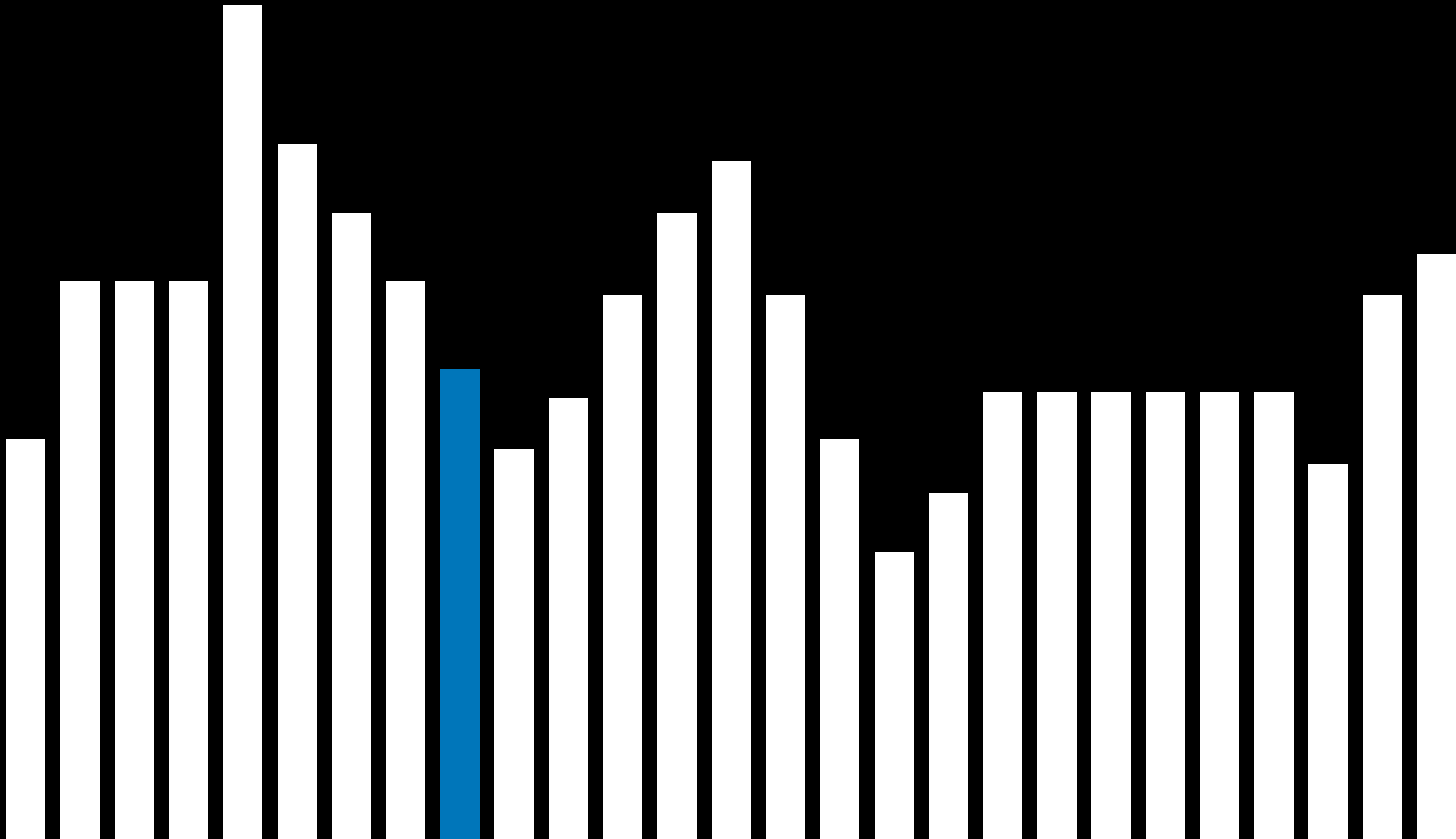
Variant	Definition
steepest-ascent	choose the highest-valued neighbor
stochastic	choose randomly from higher-valued neighbors
first-choice	choose the first higher-valued neighbor
random-restart	conduct hill climbing multiple times
local beam search	chooses the k highest-valued neighbors

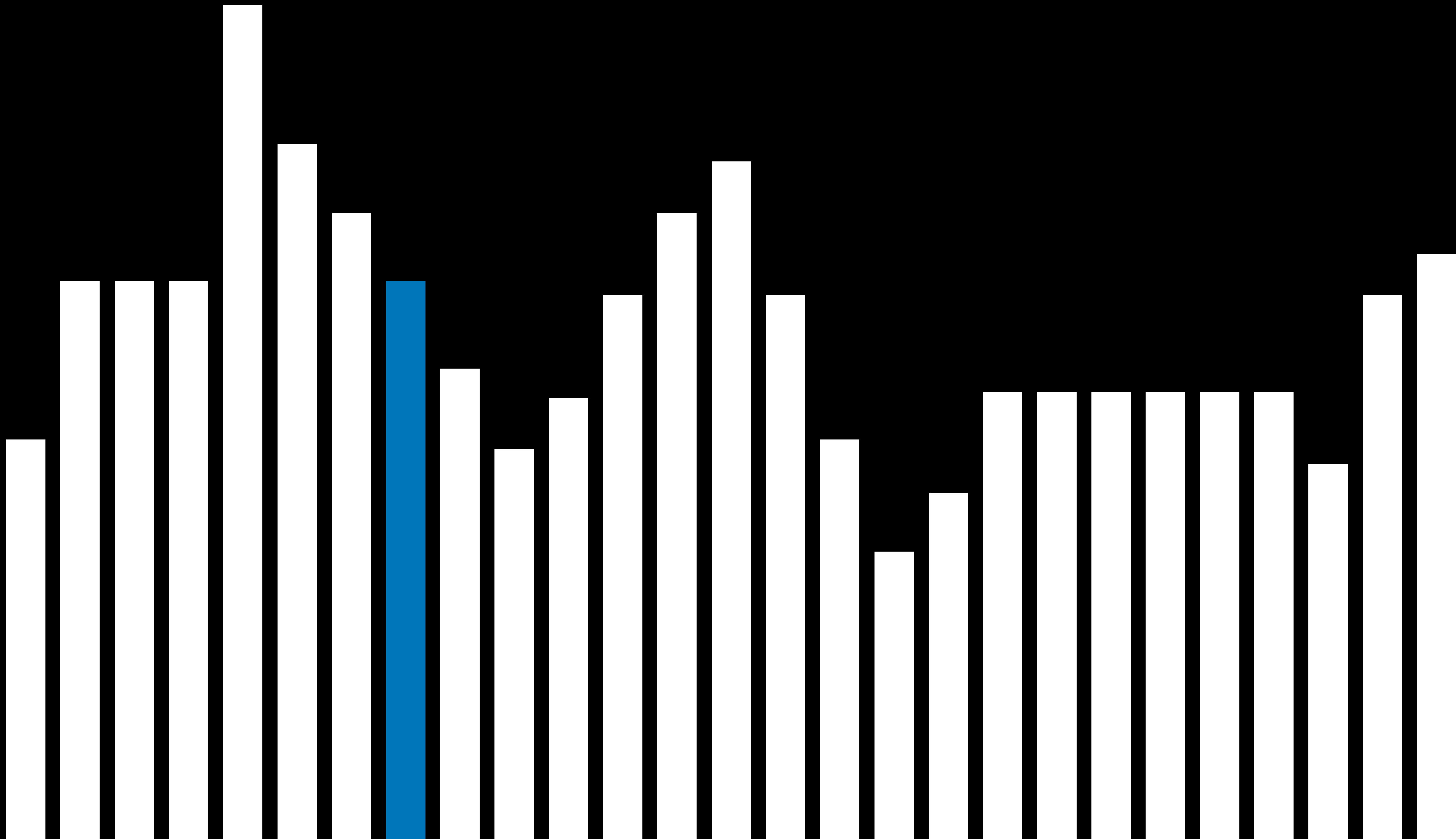
Simulated Annealing

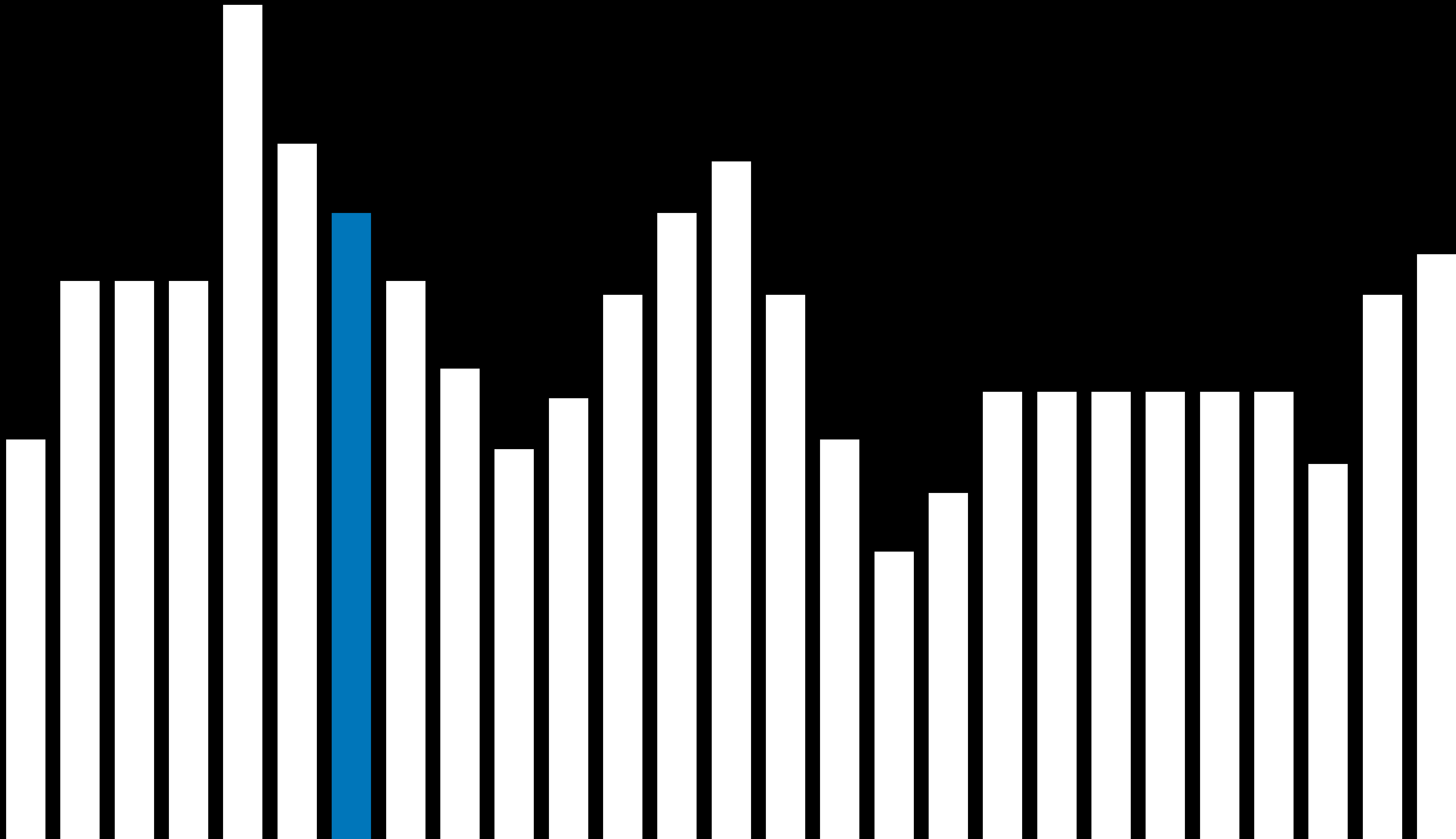


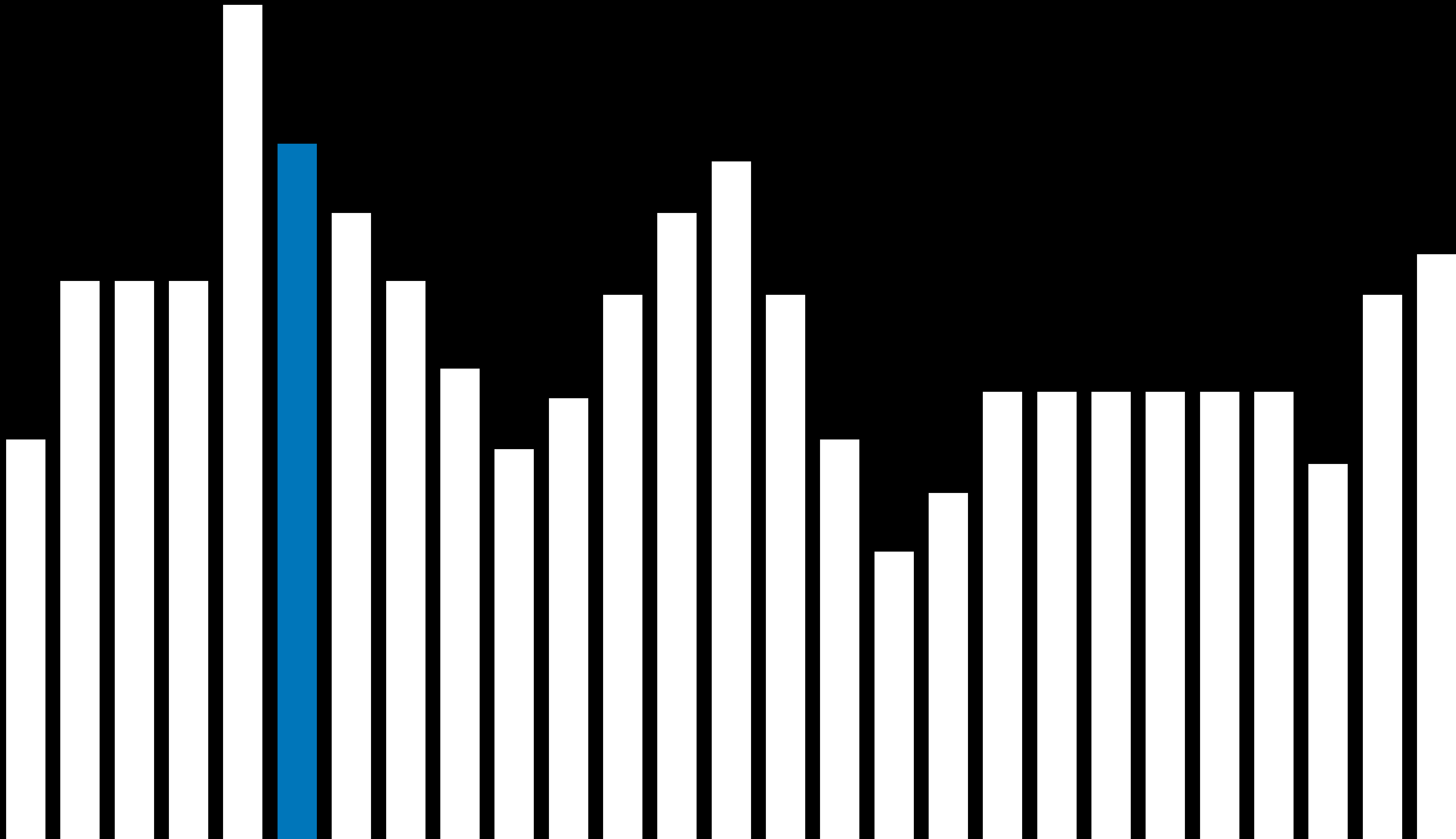


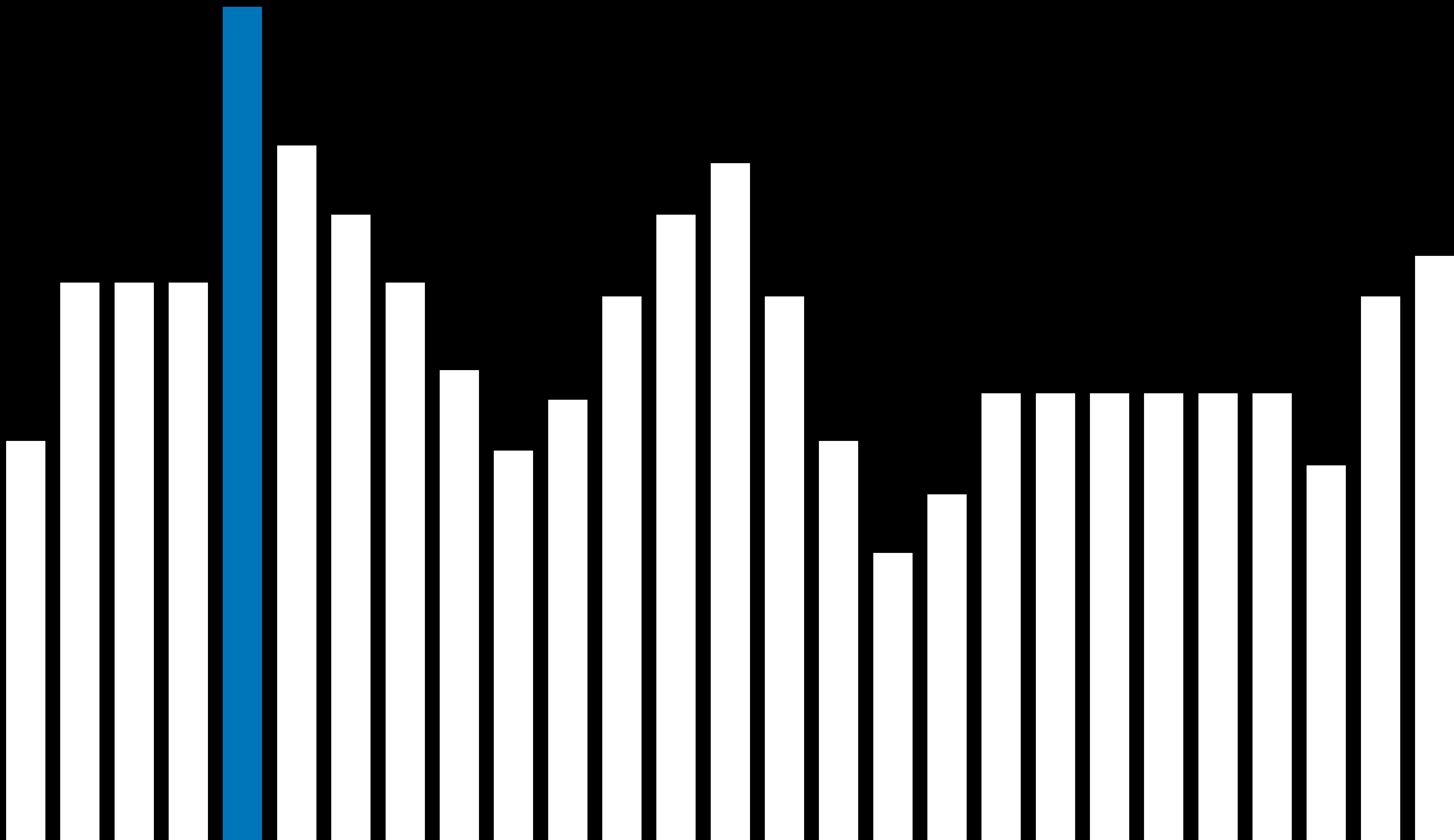












Simulated Annealing

- Early on, higher "temperature": more likely to accept neighbors that are worse than current state
- Later on, lower "temperature": less likely to accept neighbors that are worse than current state

Simulated Annealing

function SIMULATED-ANNEALING(*problem*, *max*):

current = initial state of *problem*

 for $t = 1$ to *max*:

$T = \text{TEMPERATURE}(t)$

neighbor = random neighbor of *current*

ΔE = how much better *neighbor* is than *current*

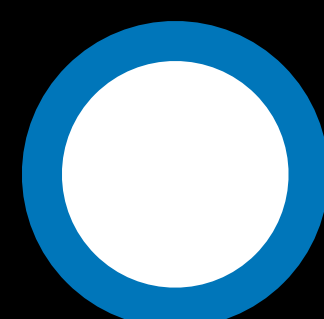
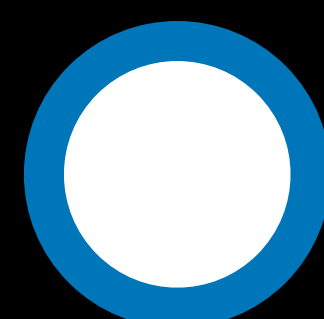
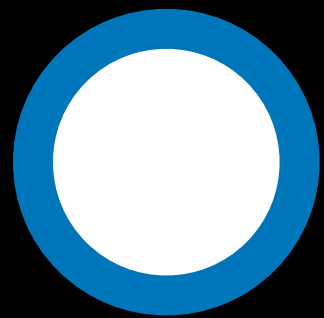
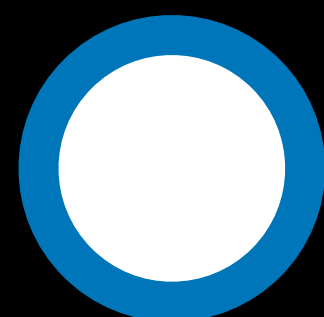
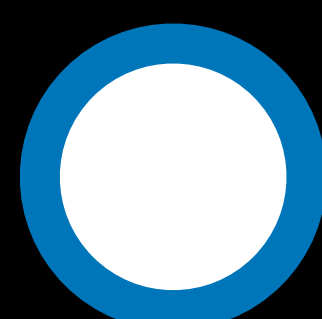
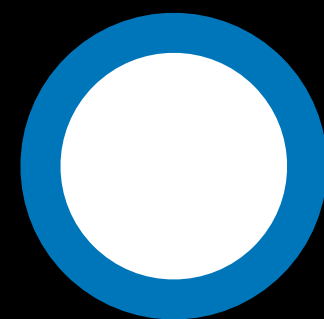
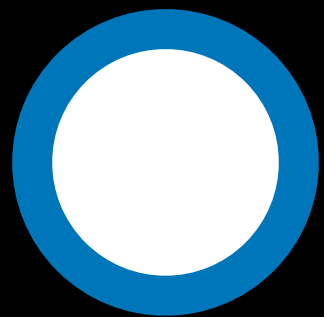
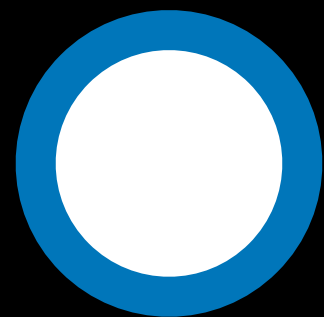
 if $\Delta E > 0$:

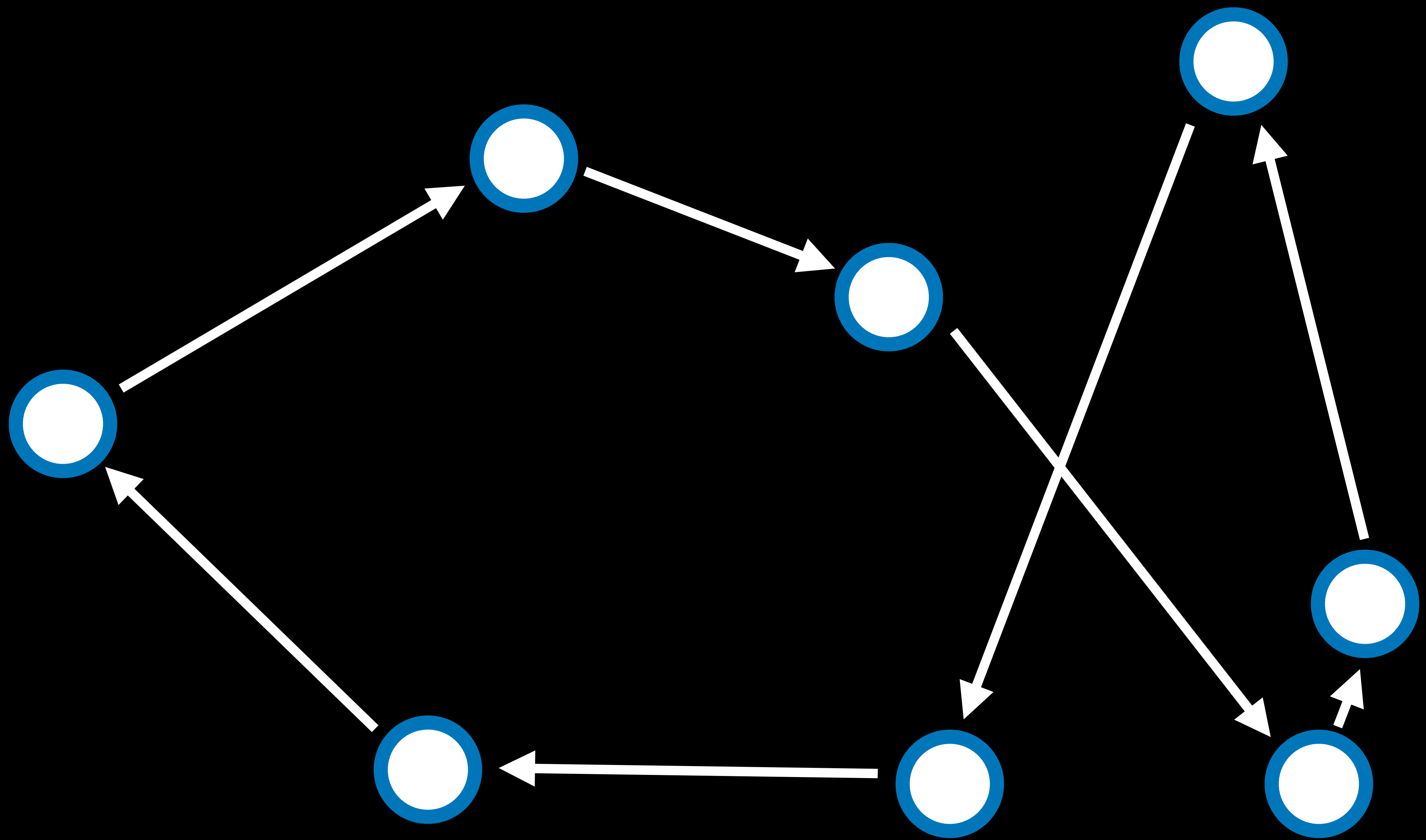
current = *neighbor*

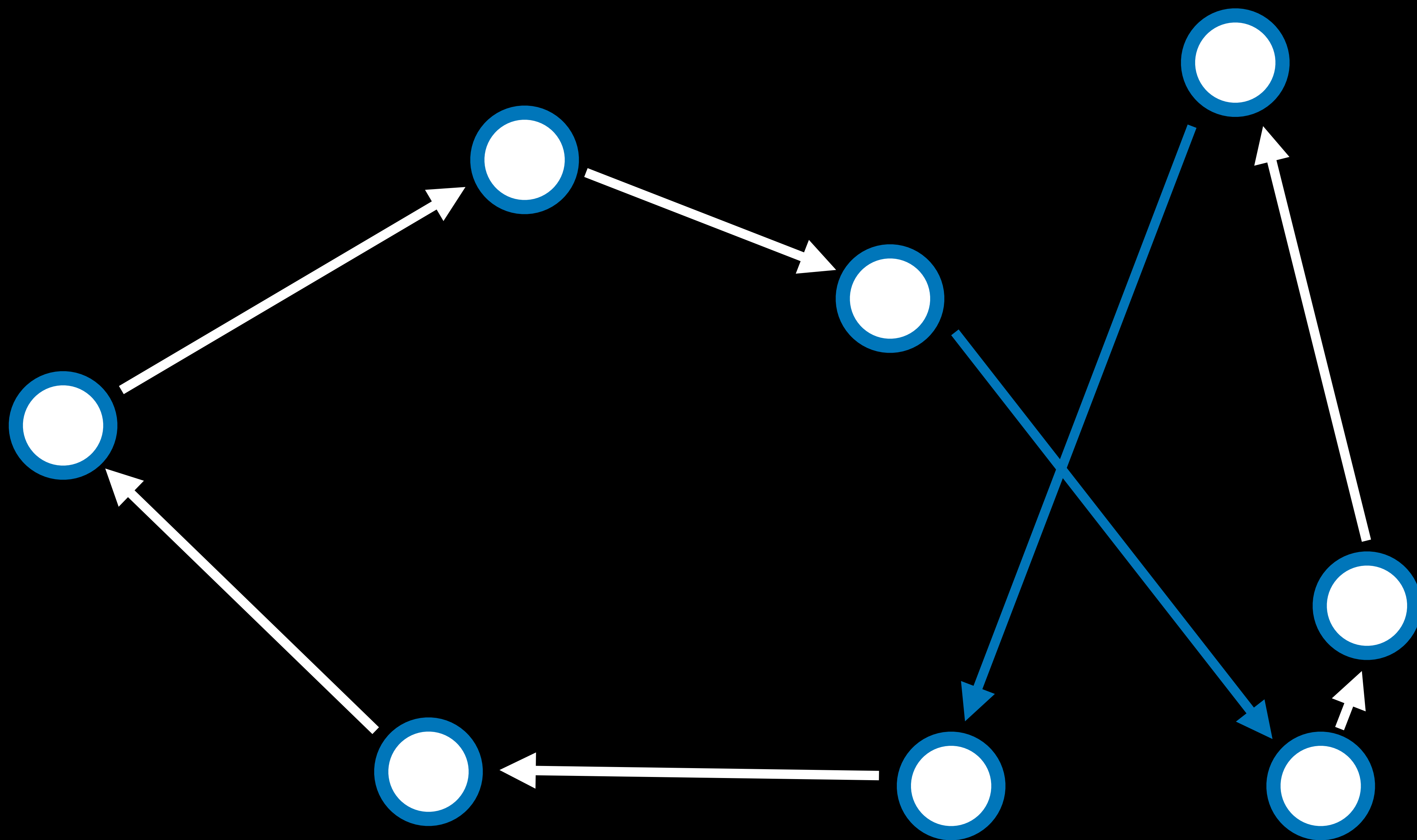
 with probability $e^{\Delta E/T}$ set *current* = *neighbor*

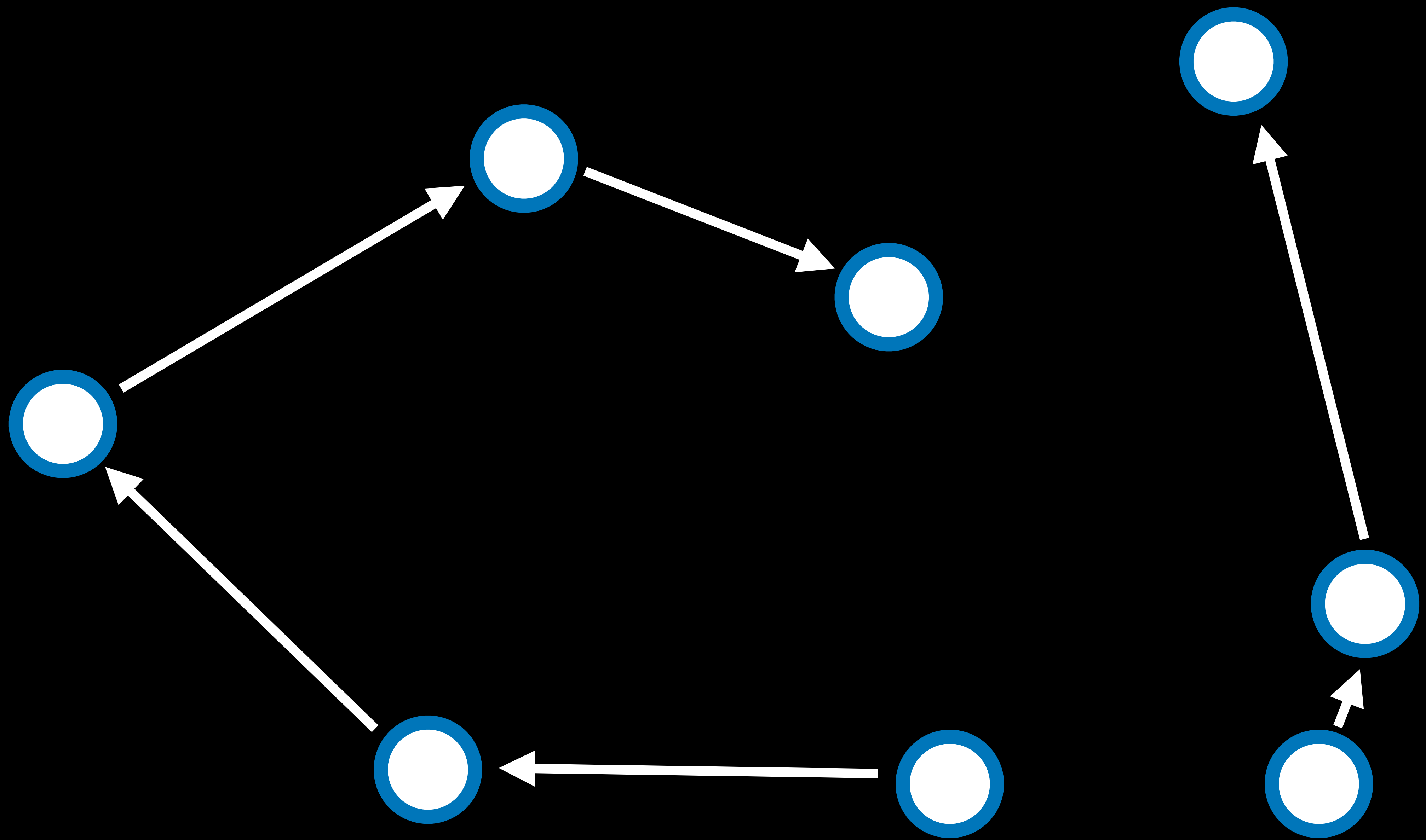
 return *current*

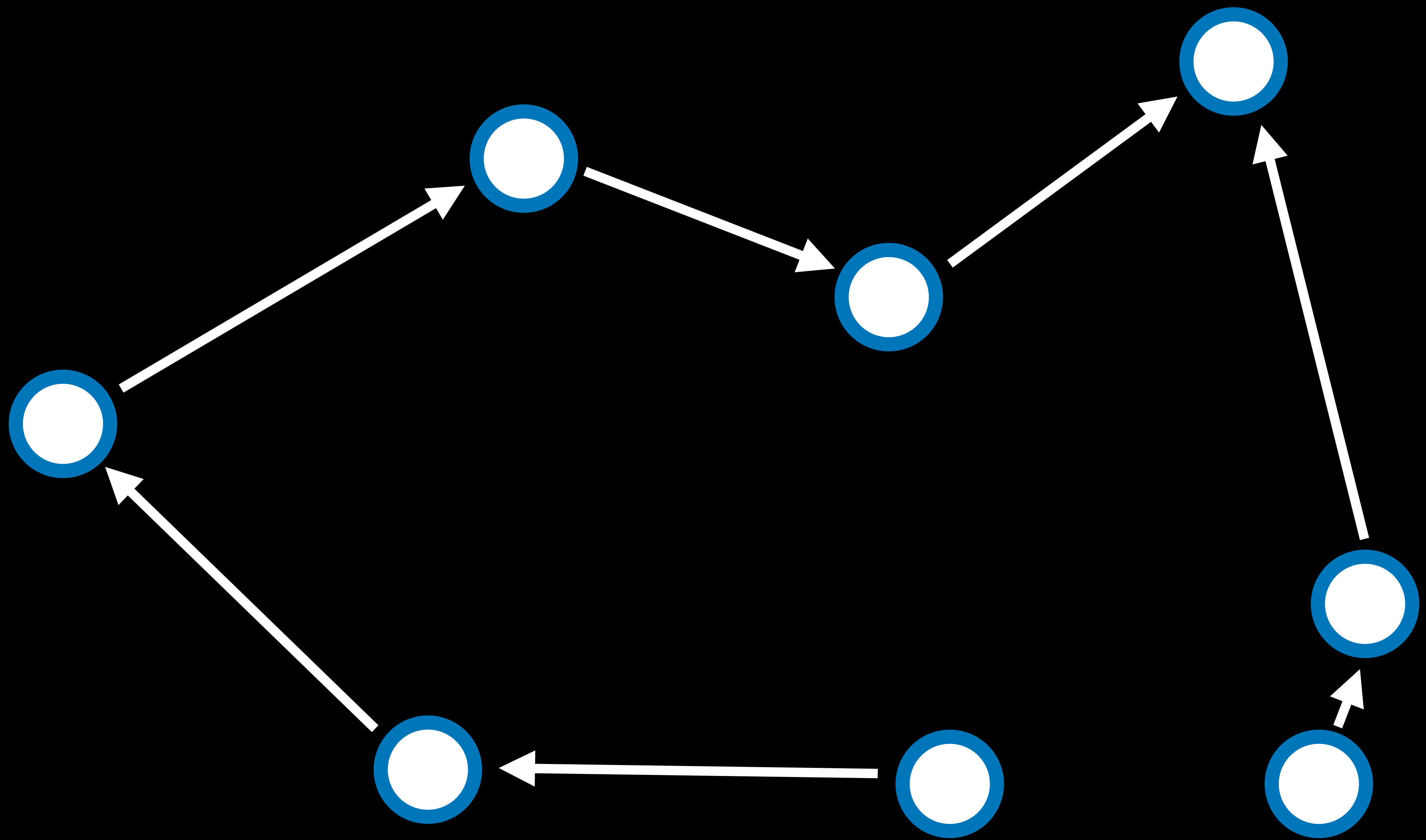
Traveling Salesman Problem

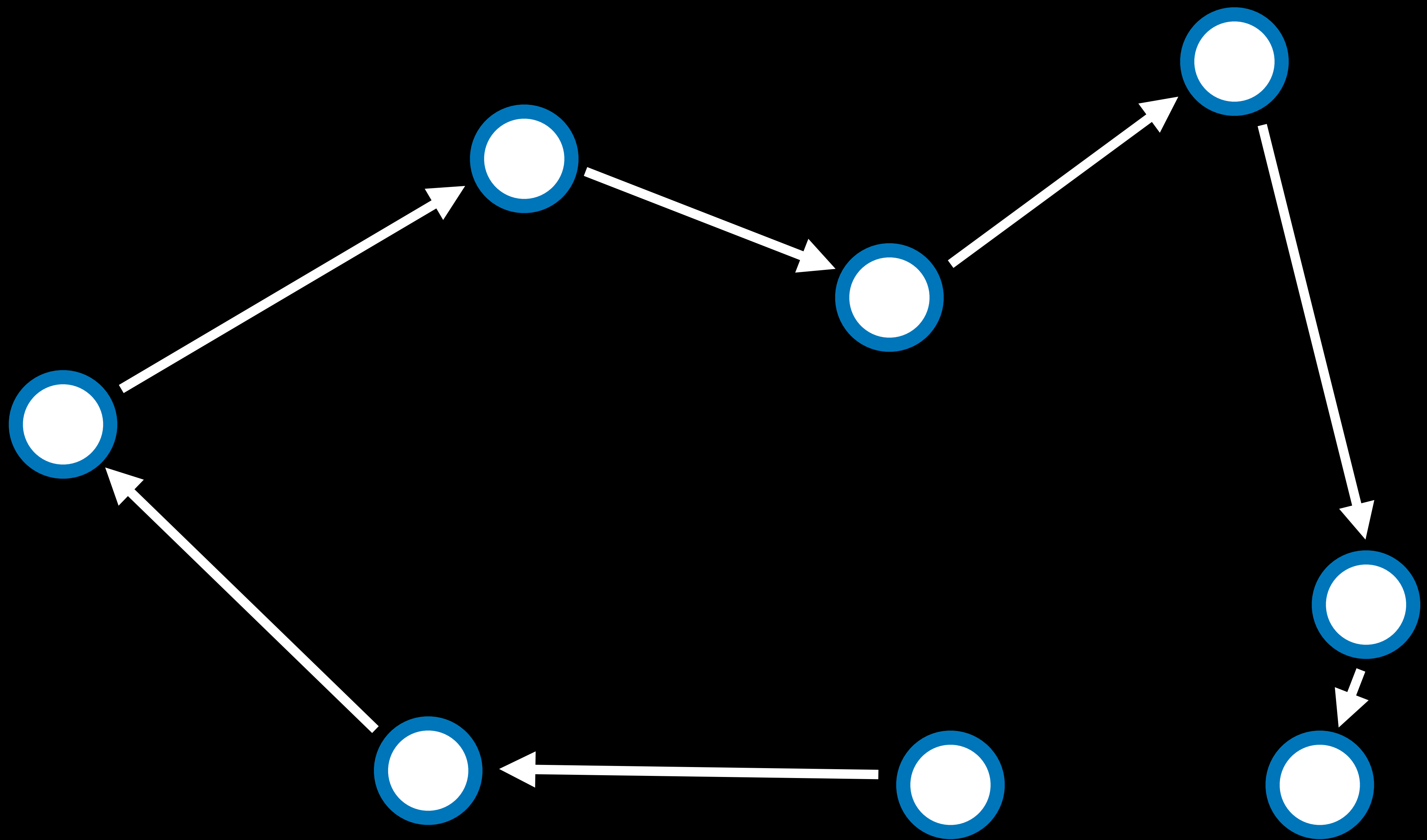


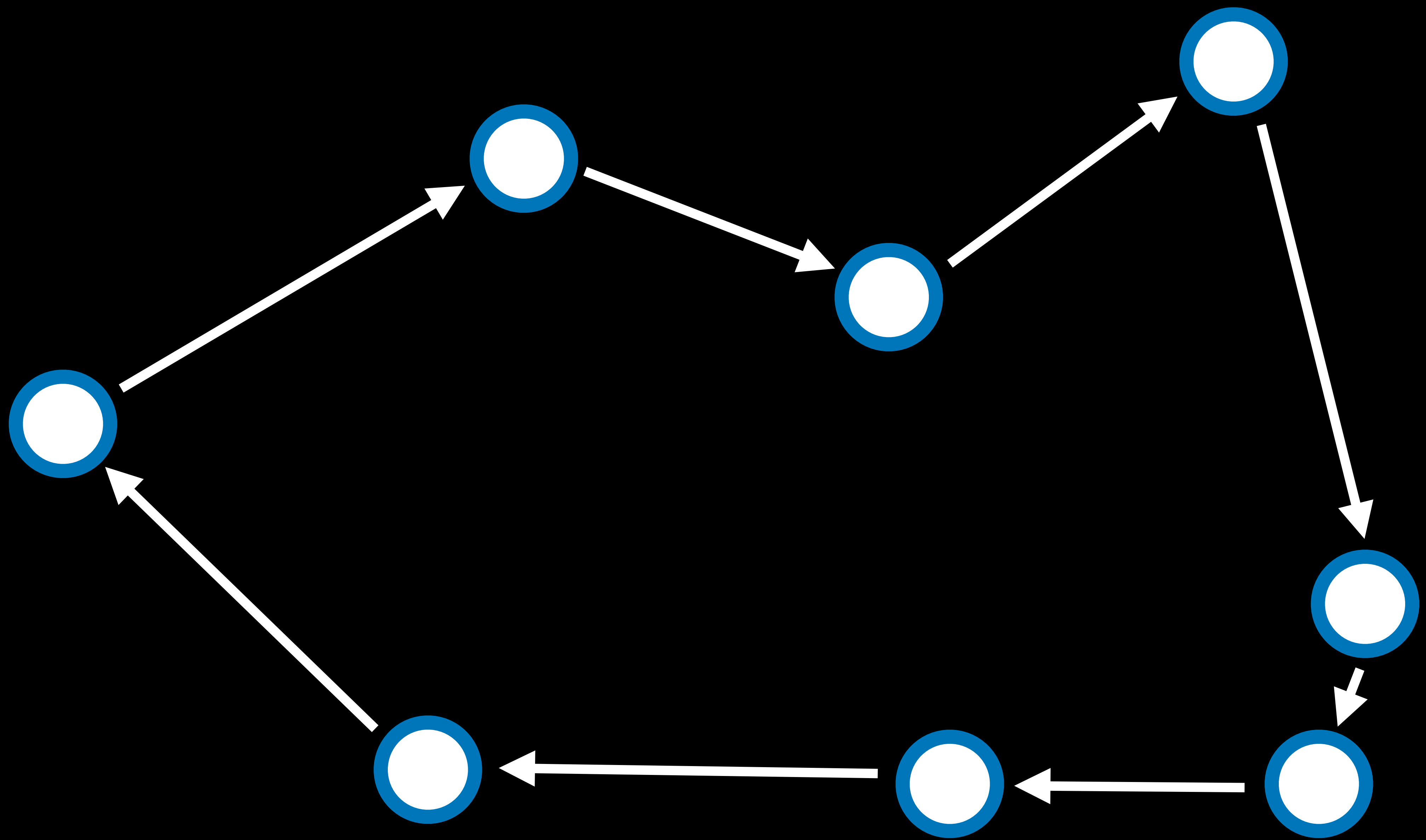












Linear Programming

Linear Programming

- Minimize a cost function $c_1x_1 + c_2x_2 + \dots + c_nx_n$
- With constraints of form $a_1x_1 + a_2x_2 + \dots + a_nx_n \leq b$
or of form $a_1x_1 + a_2x_2 + \dots + a_nx_n = b$
- With bounds for each variable $l_i \leq x_i \leq u_i$

Linear Programming Example

- Two machines X_1 and X_2 . X_1 costs \$50/hour to run, X_2 costs \$80/hour to run. Goal is to minimize cost.
- X_1 requires 5 units of labor per hour. X_2 requires 2 units of labor per hour. Total of 20 units of labor to spend.
- X_1 produces 10 units of output per hour. X_2 produces 12 units of output per hour. Company needs 90 units of output.

Linear Programming Example

Cost Function: $50x_1 + 80x_2$

- X_1 requires 5 units of labor per hour. X_2 requires 2 units of labor per hour. Total of 20 units of labor to spend.
- X_1 produces 10 units of output per hour. X_2 produces 12 units of output per hour. Company needs 90 units of output.

Linear Programming Example

Cost Function: $50x_1 + 80x_2$

Constraint: $5x_1 + 2x_2 \leq 20$

- X_1 produces 10 units of output per hour. X_2 produces 12 units of output per hour. Company needs 90 units of output.

Linear Programming Example

Cost Function: $50x_1 + 80x_2$

Constraint: $5x_1 + 2x_2 \leq 20$

Constraint: $10x_1 + 12x_2 \geq 90$

Linear Programming Example

Cost Function: $50x_1 + 80x_2$

Constraint: $5x_1 + 2x_2 \leq 20$

Constraint: $(-10x_1) + (-12x_2) \leq -90$

Linear Programming Algorithms

- Simplex
- Interior-Point

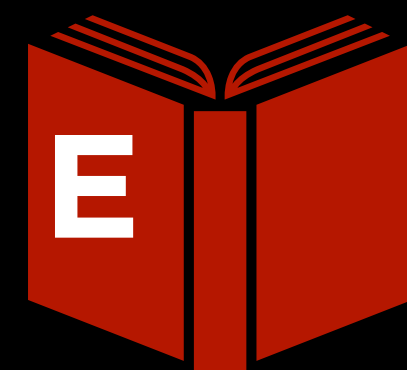
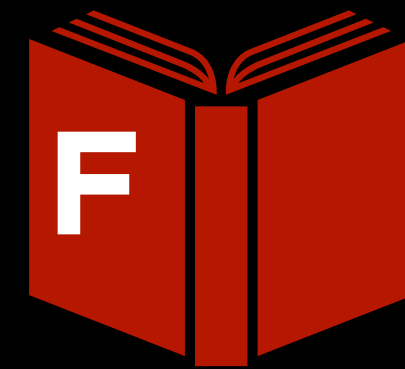
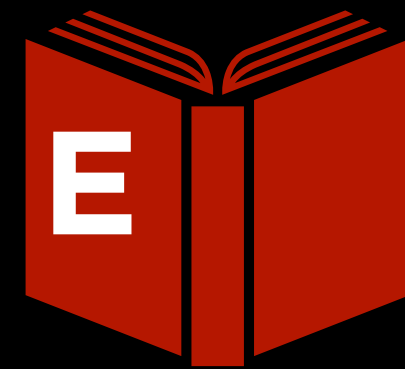
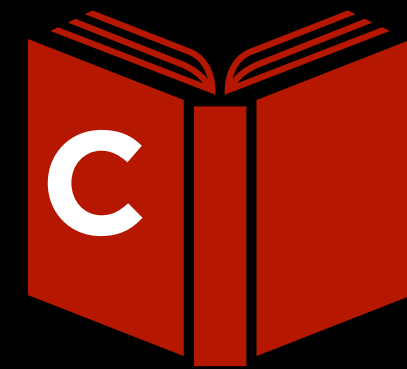
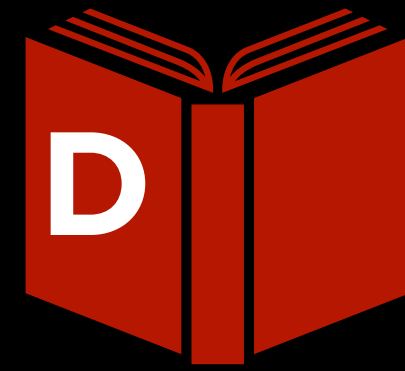
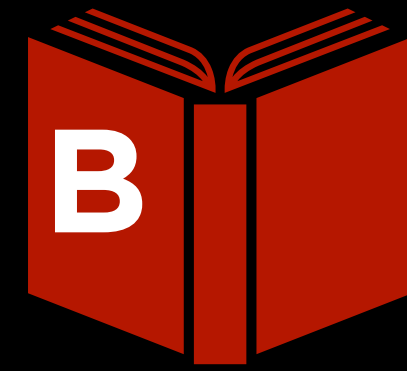
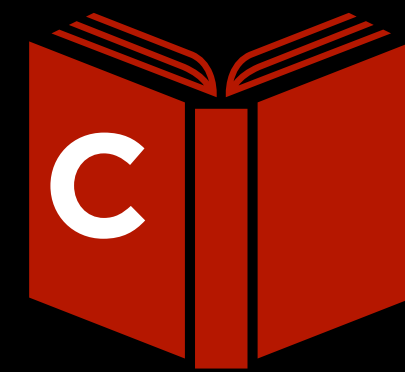
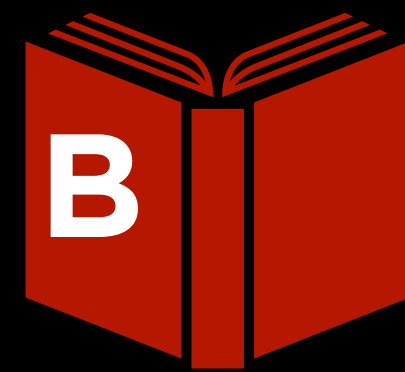
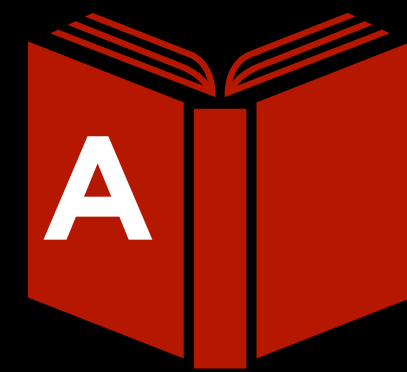
Constraint Satisfaction

Student:



Student:

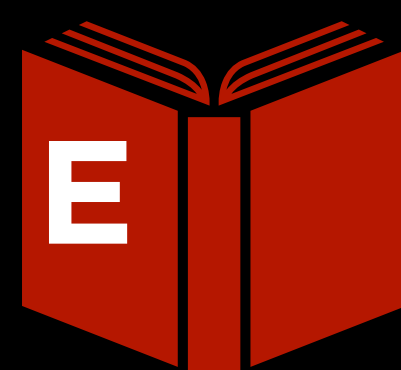
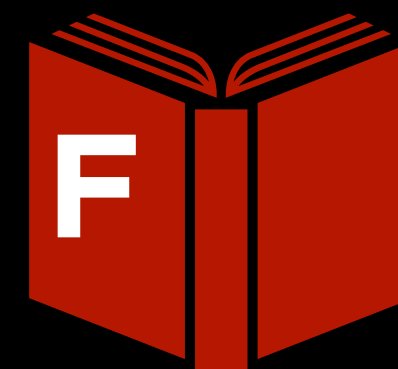
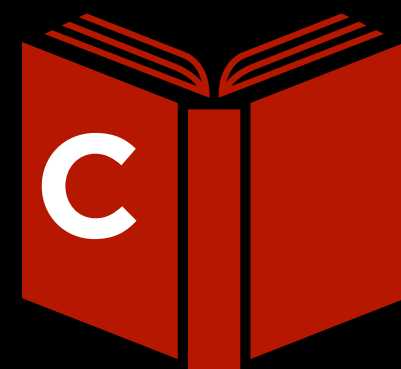
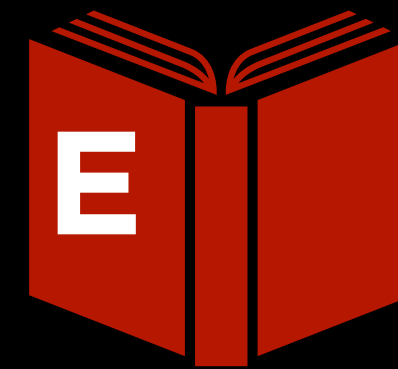
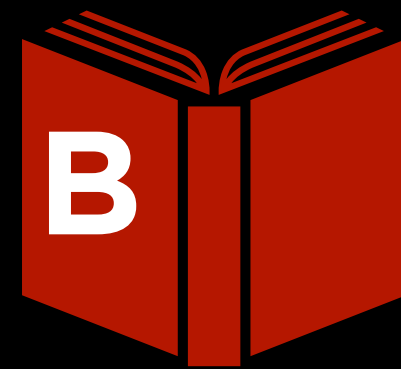
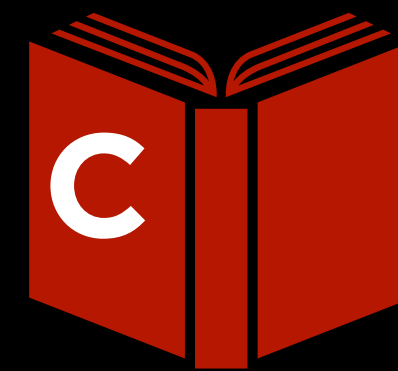
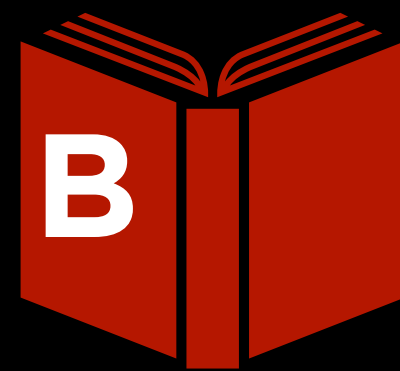
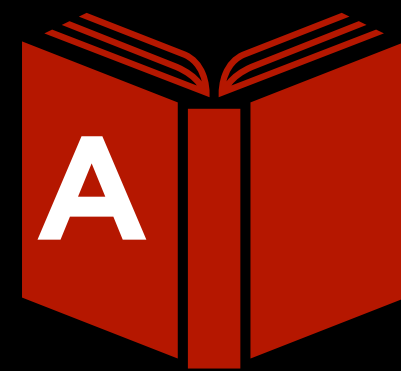
Taking classes:



Student:



Taking classes:

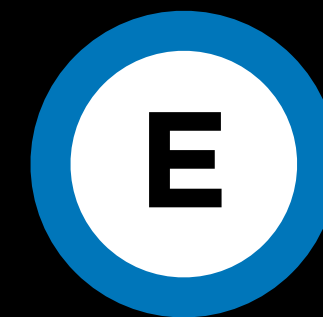
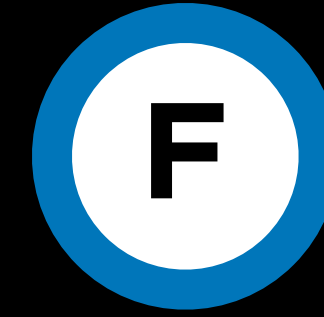
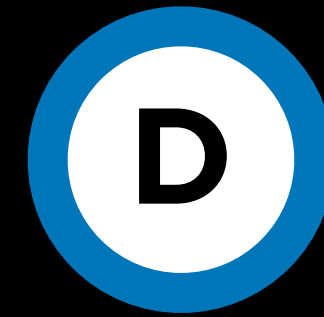
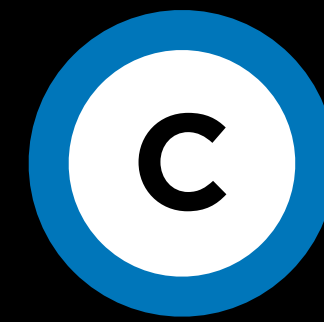
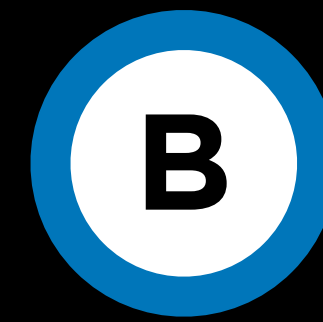
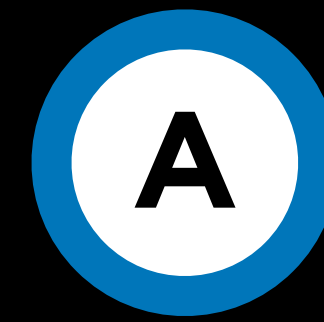
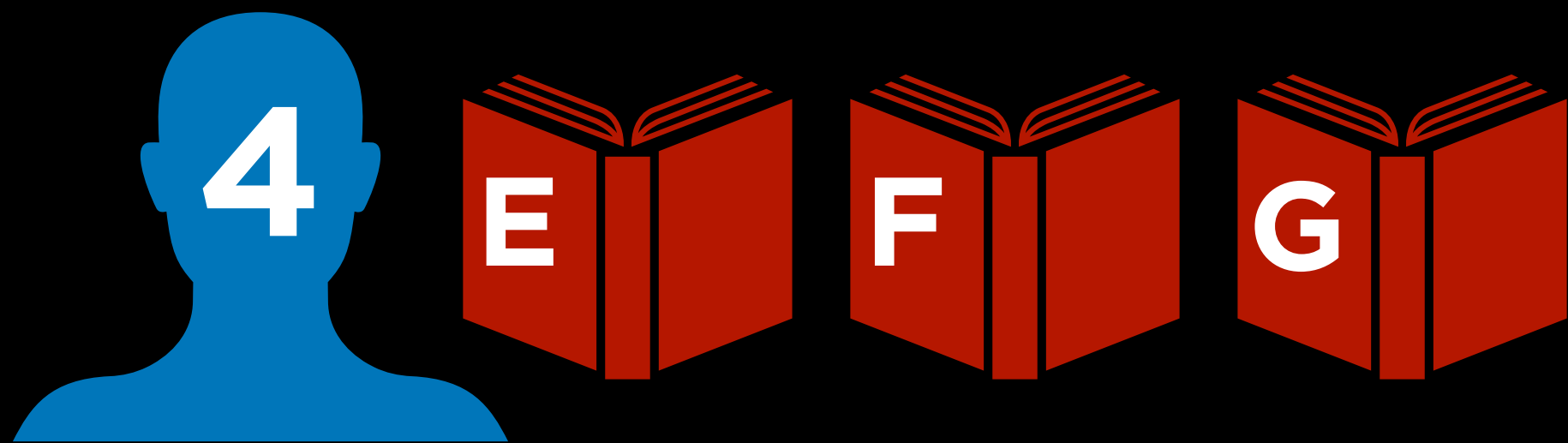
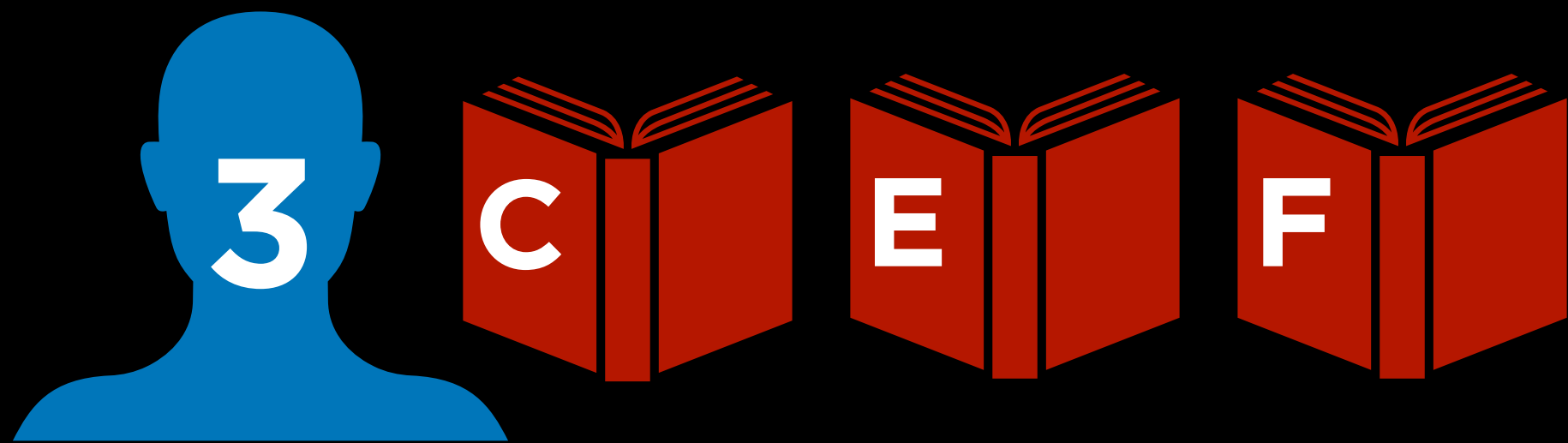
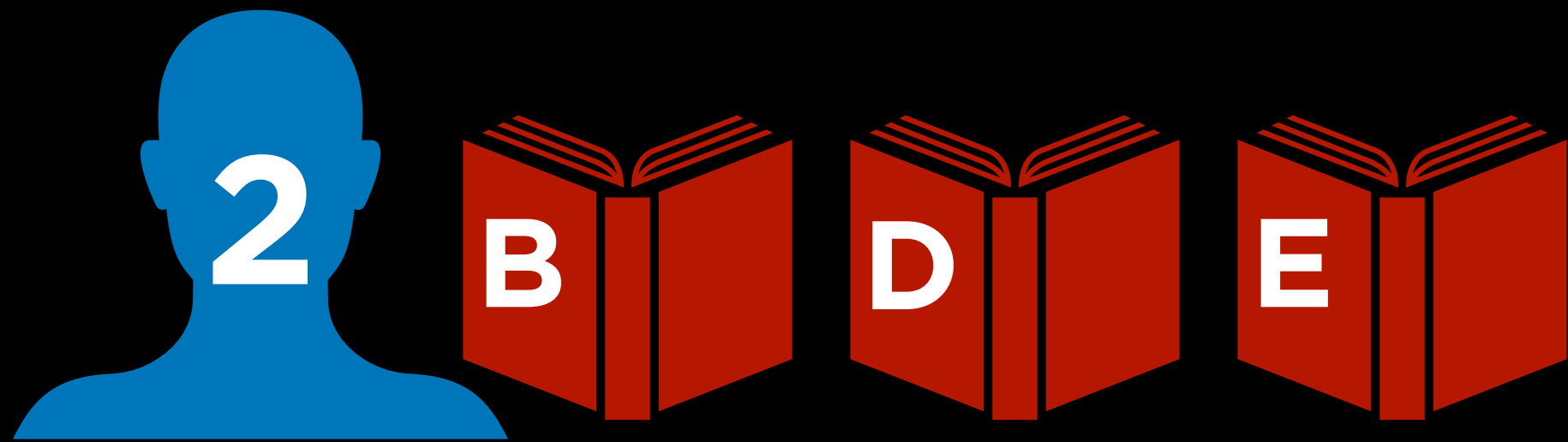
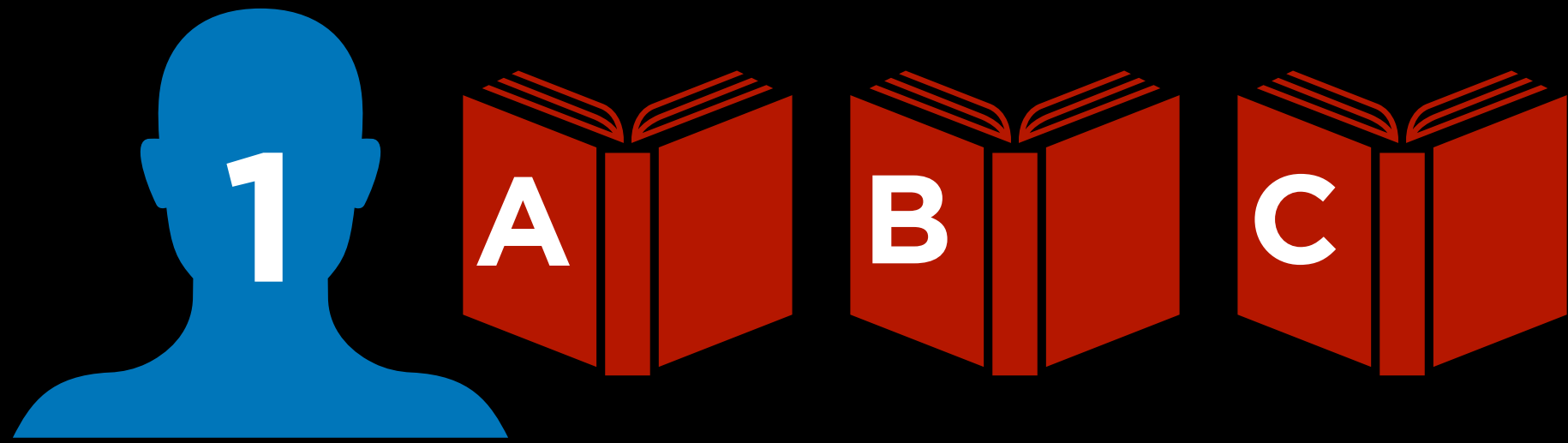


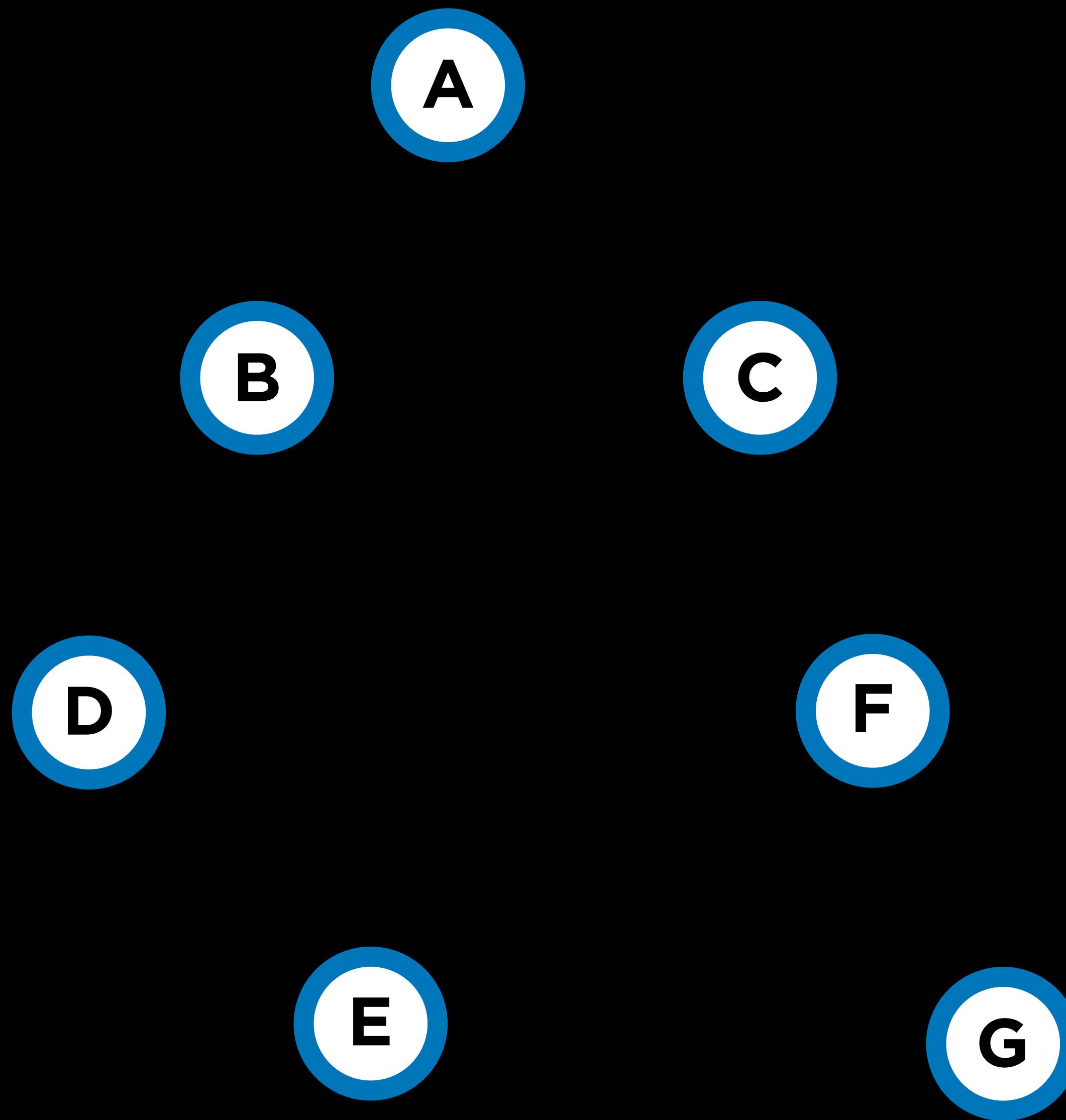
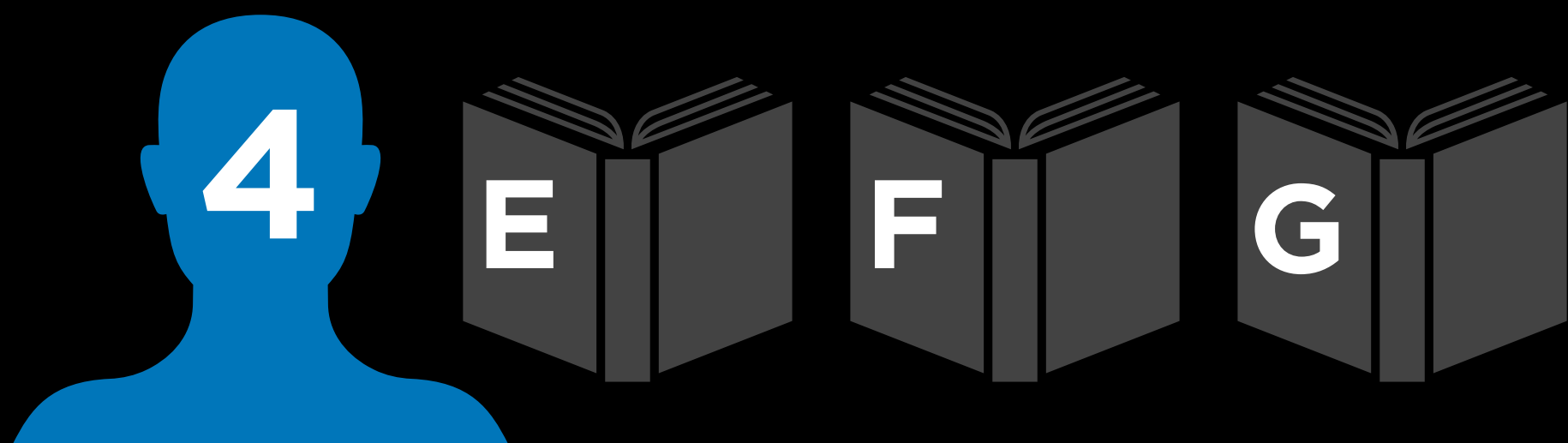
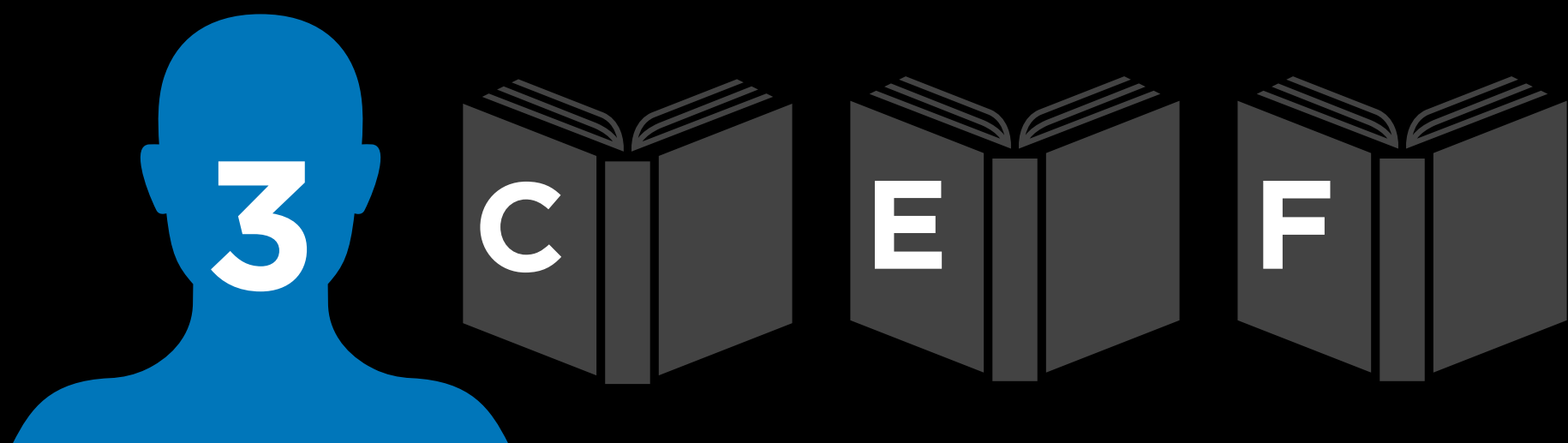
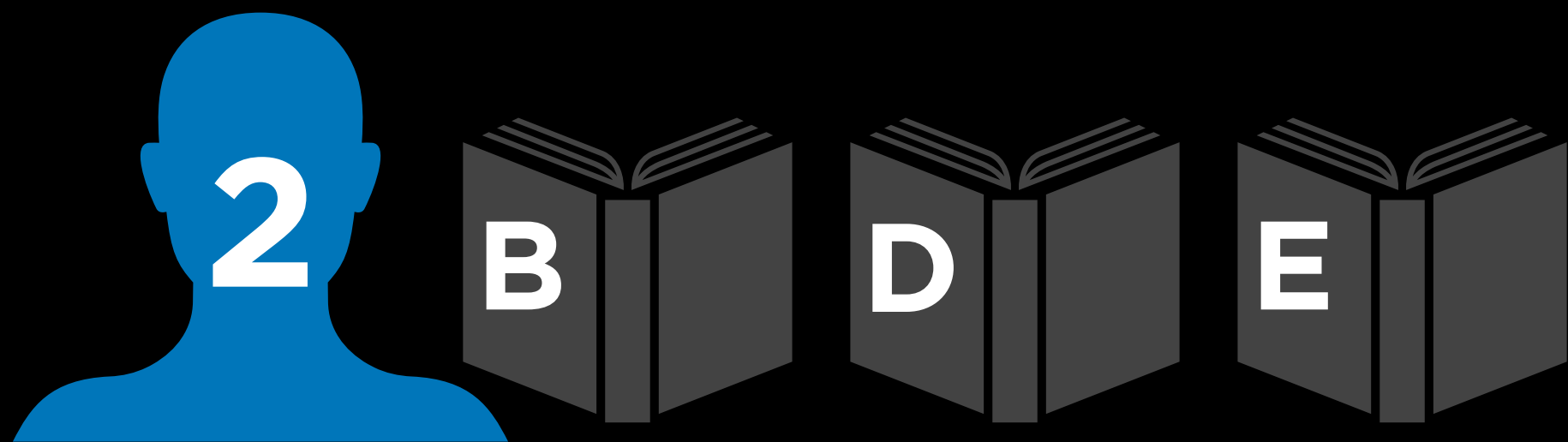
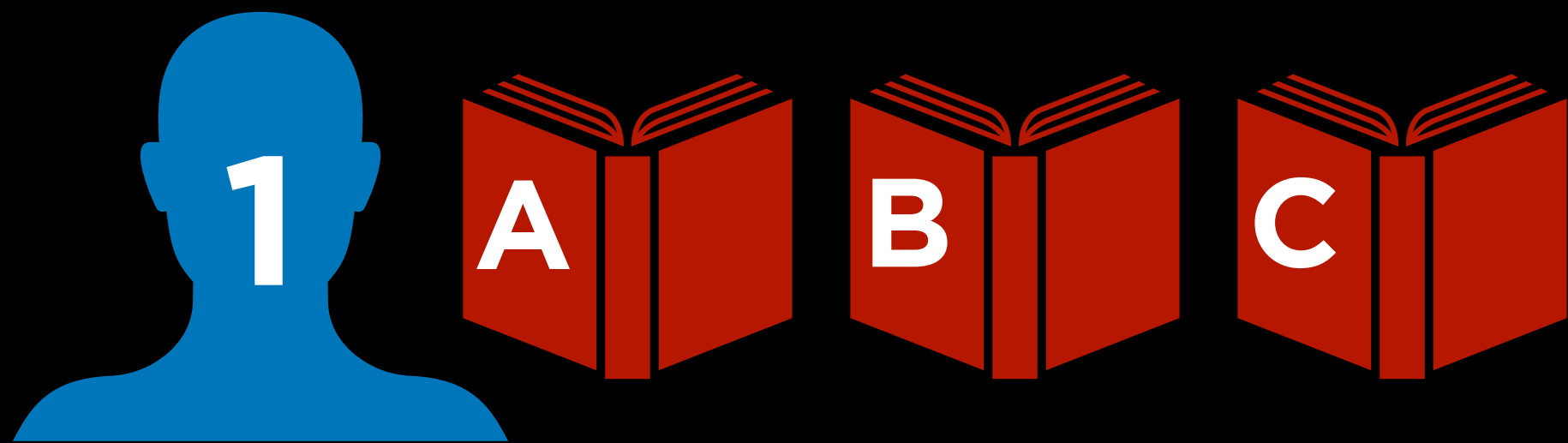
Exam slots:

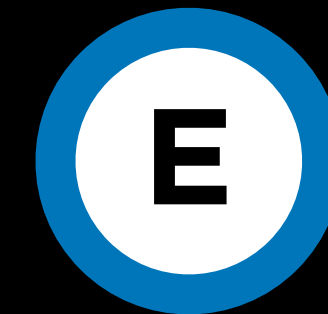
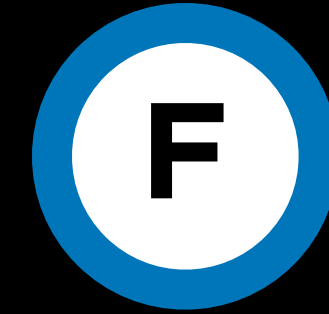
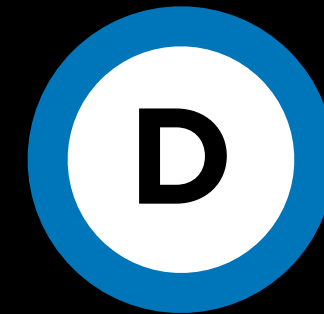
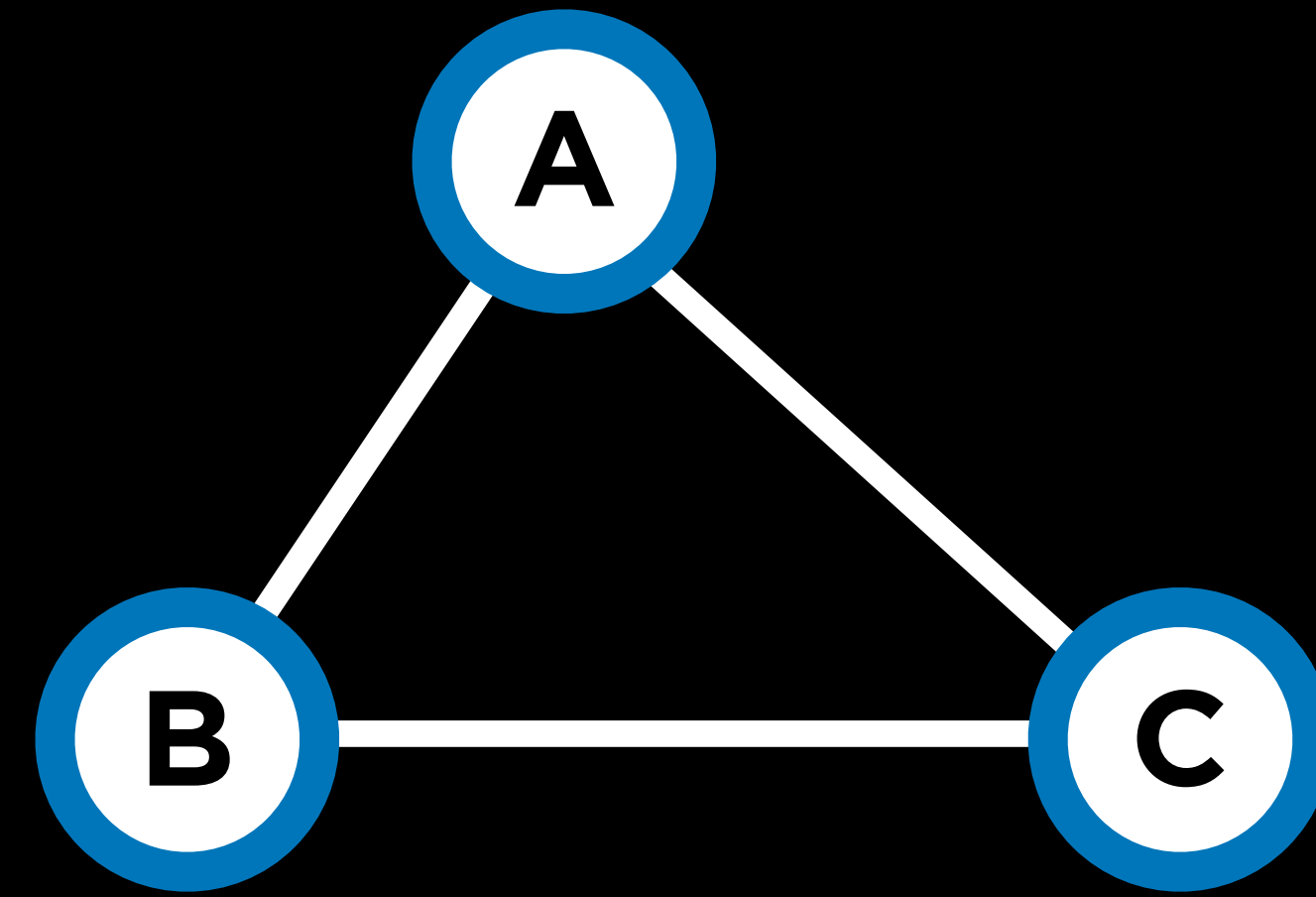
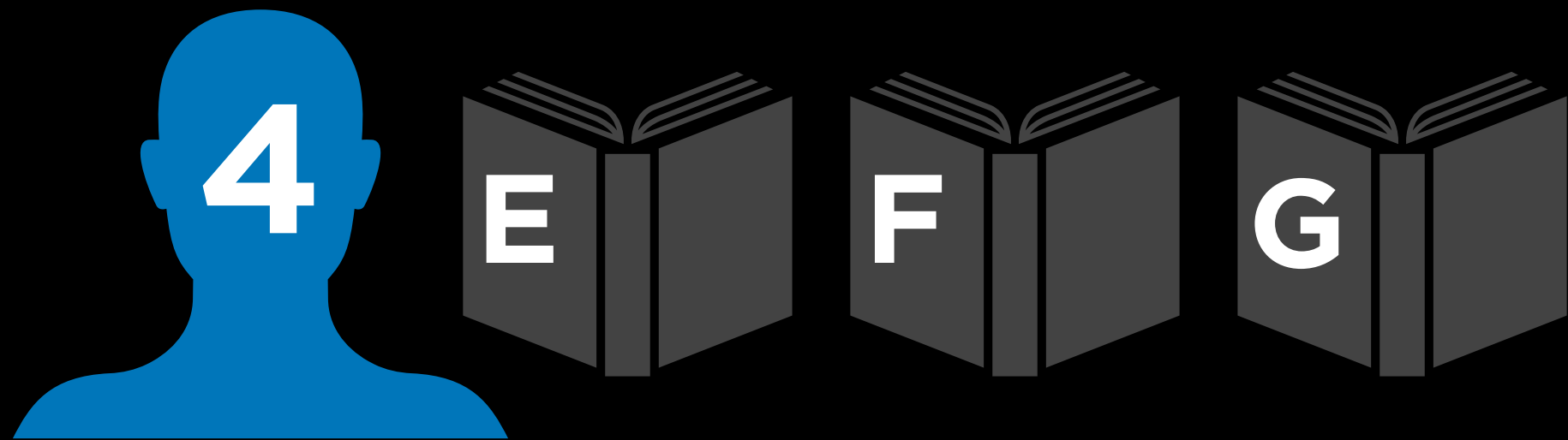
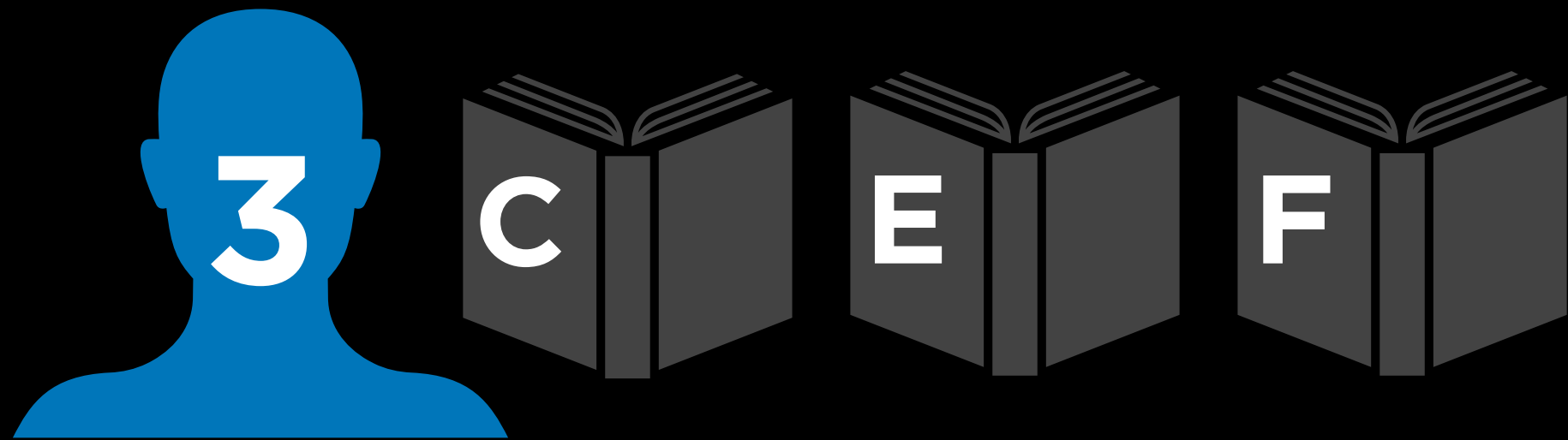
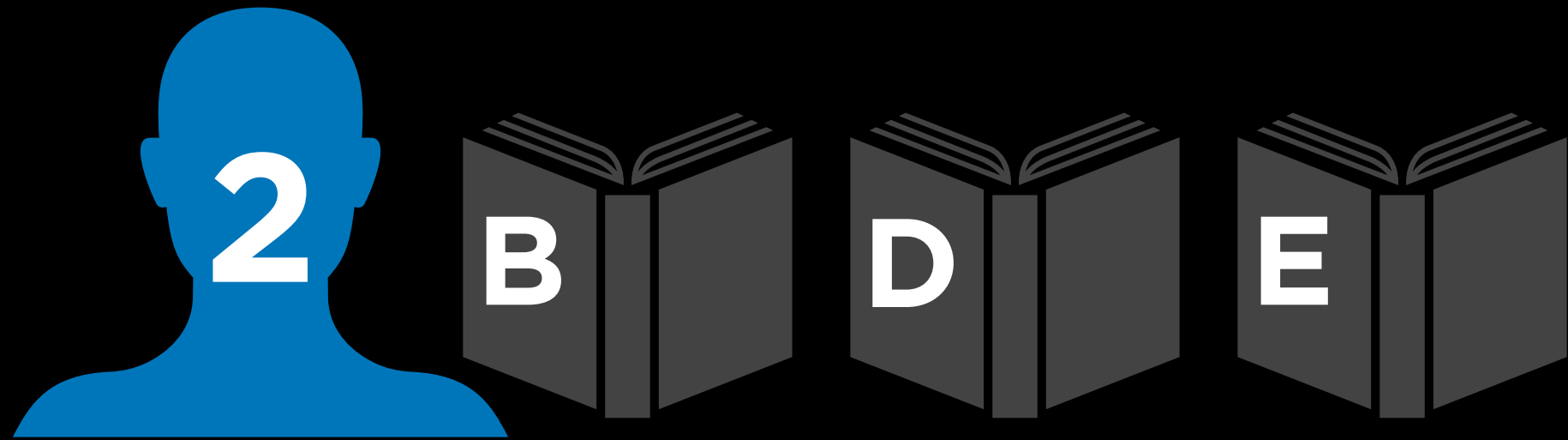
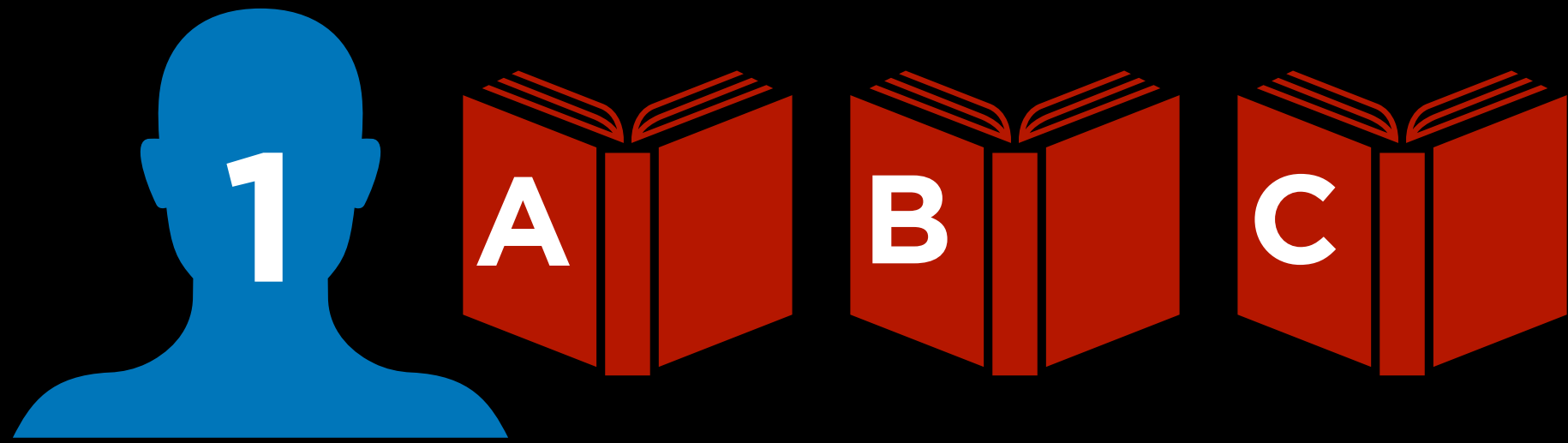
Monday

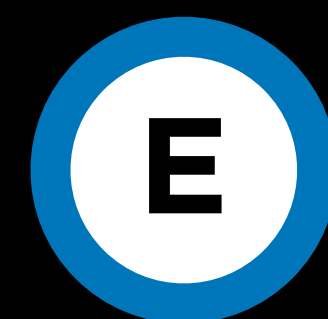
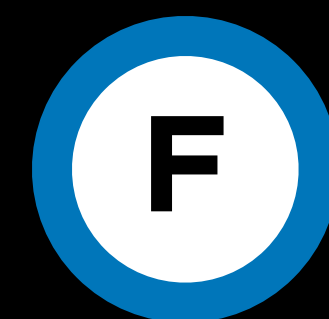
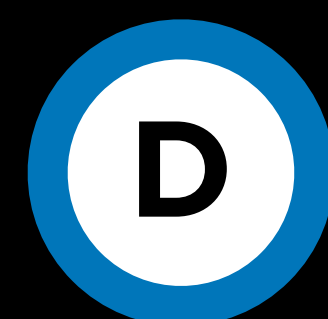
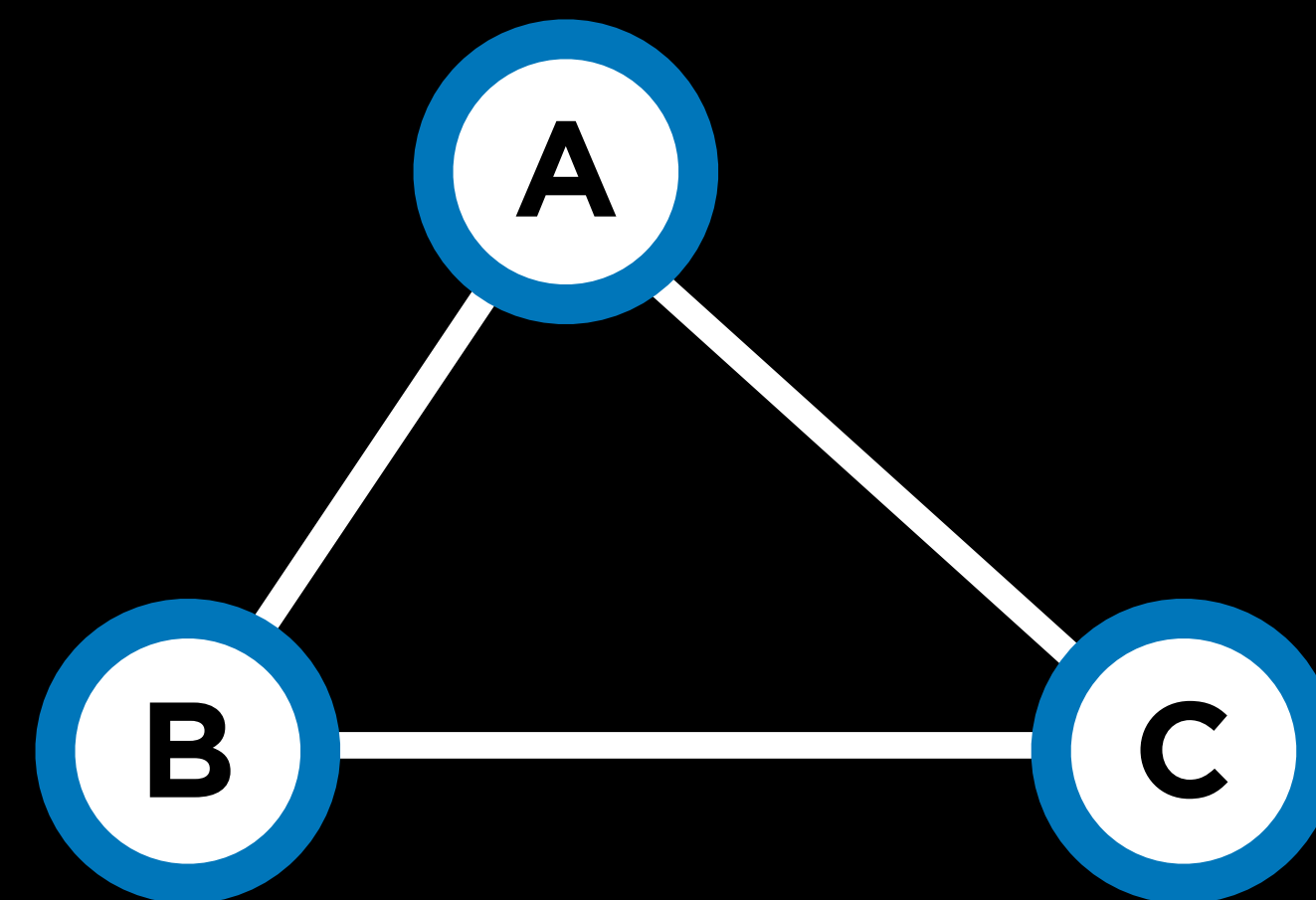
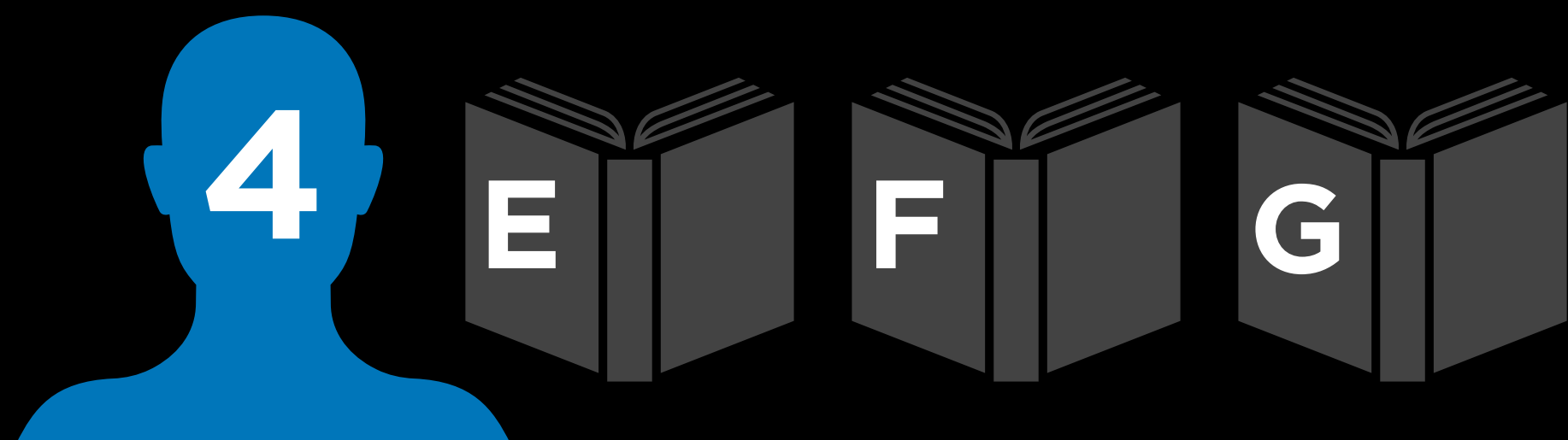
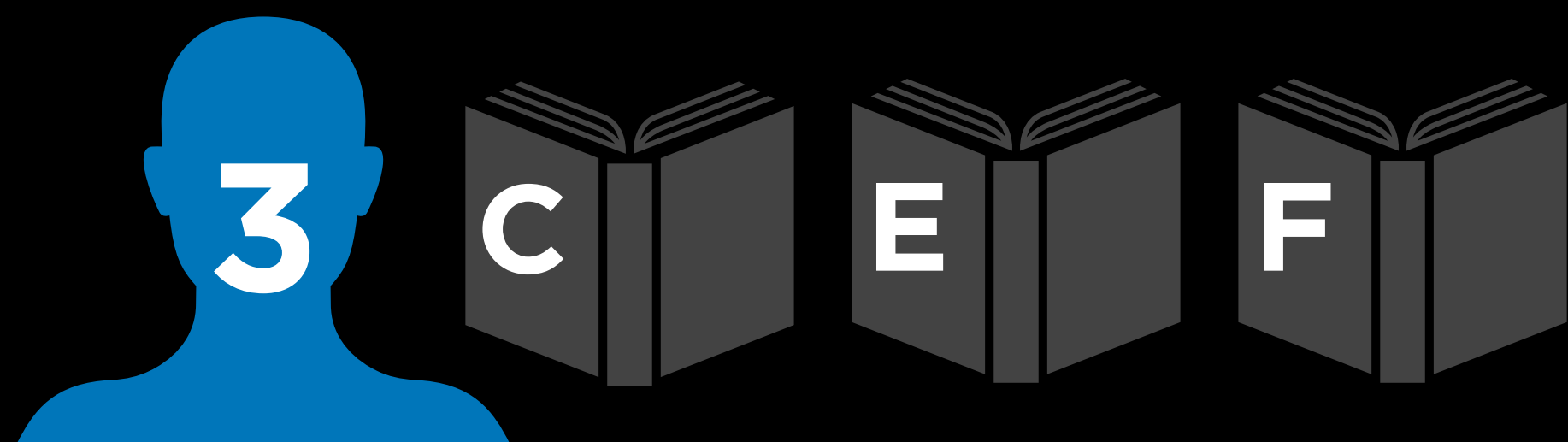
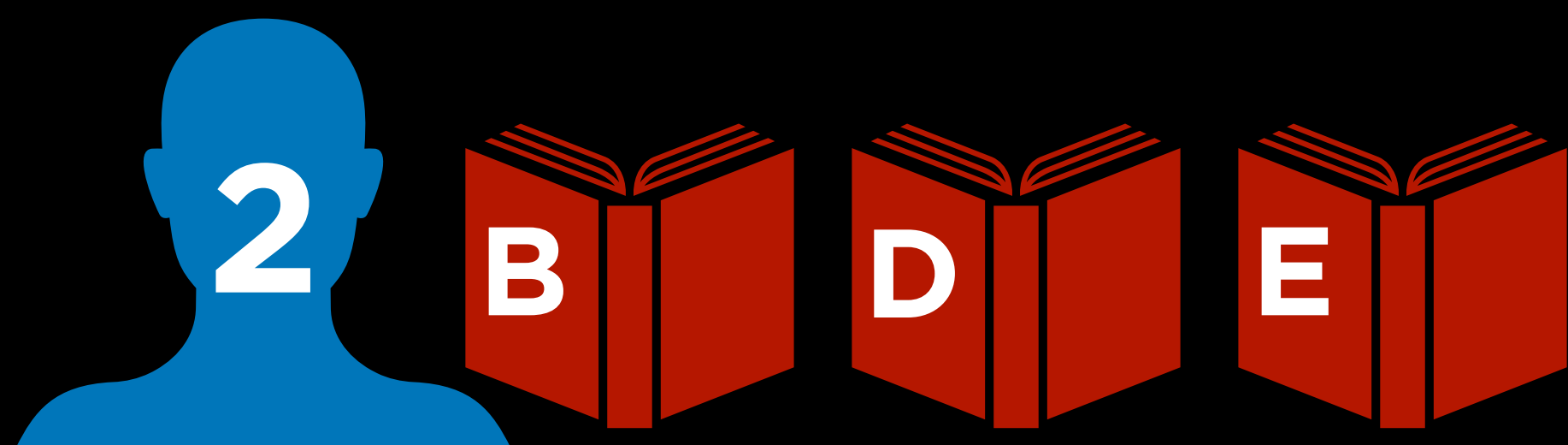
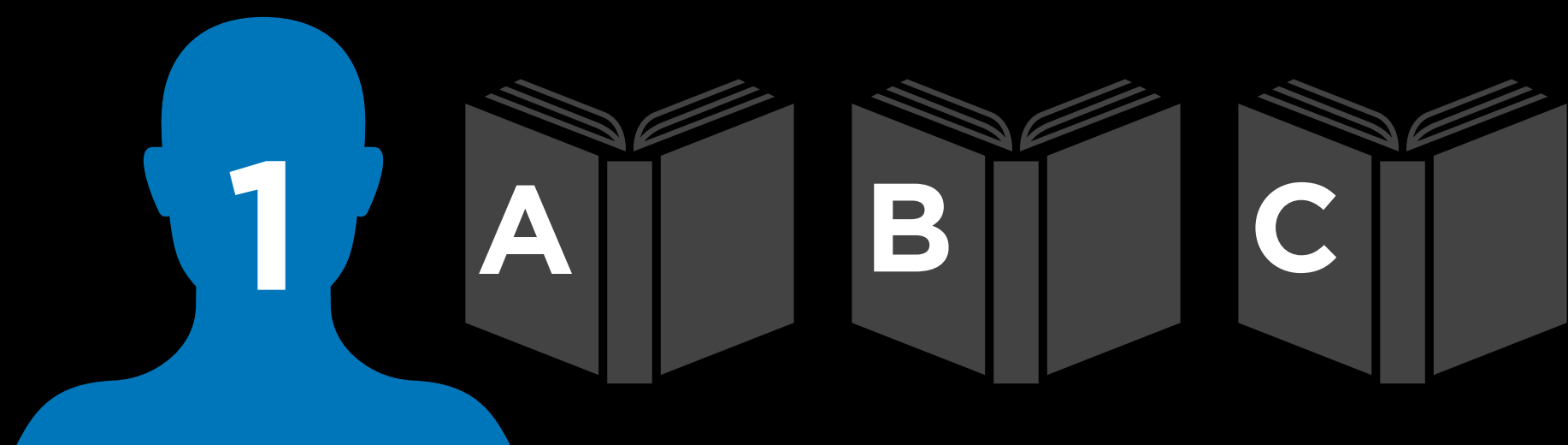
Tuesday

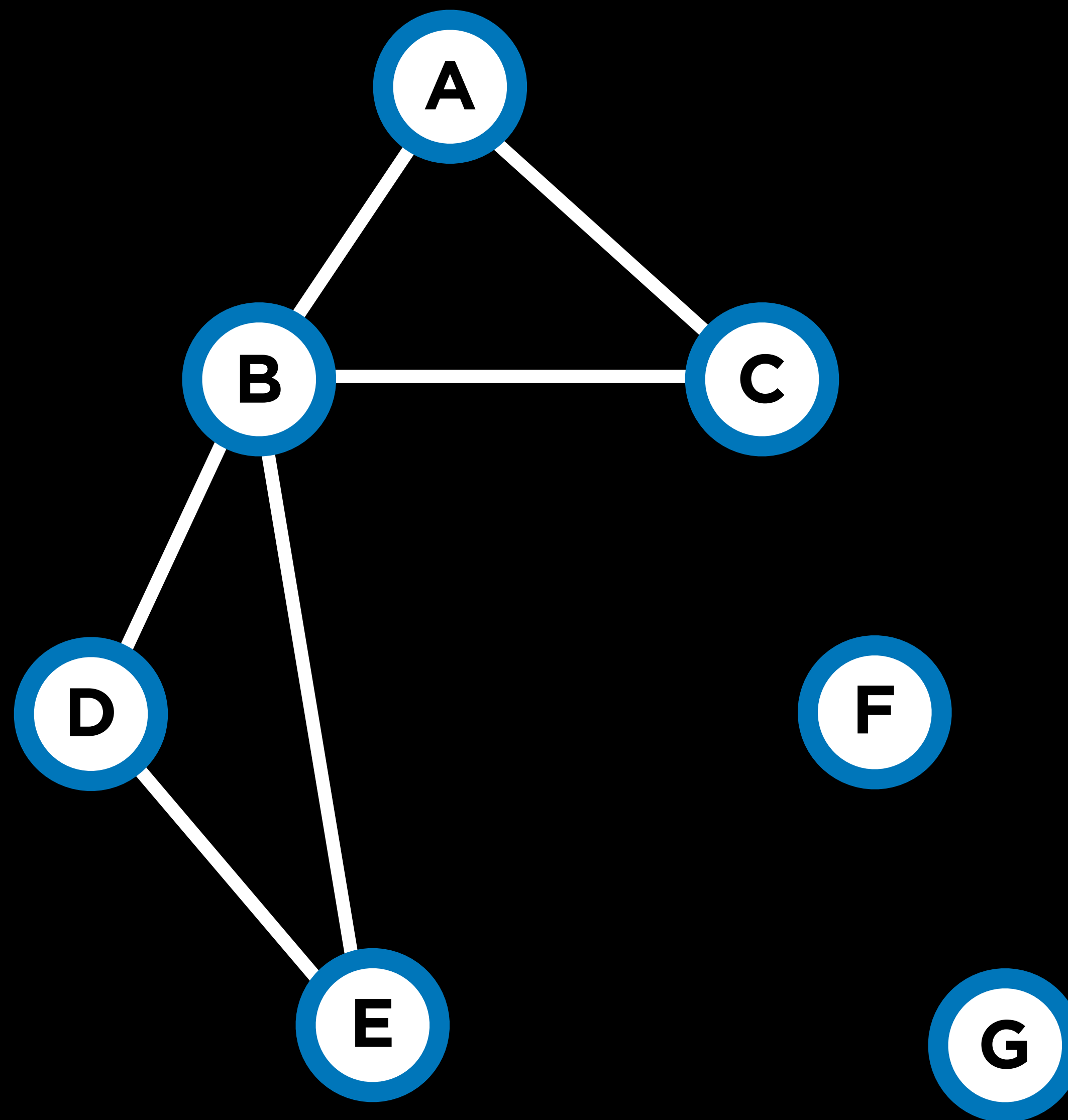
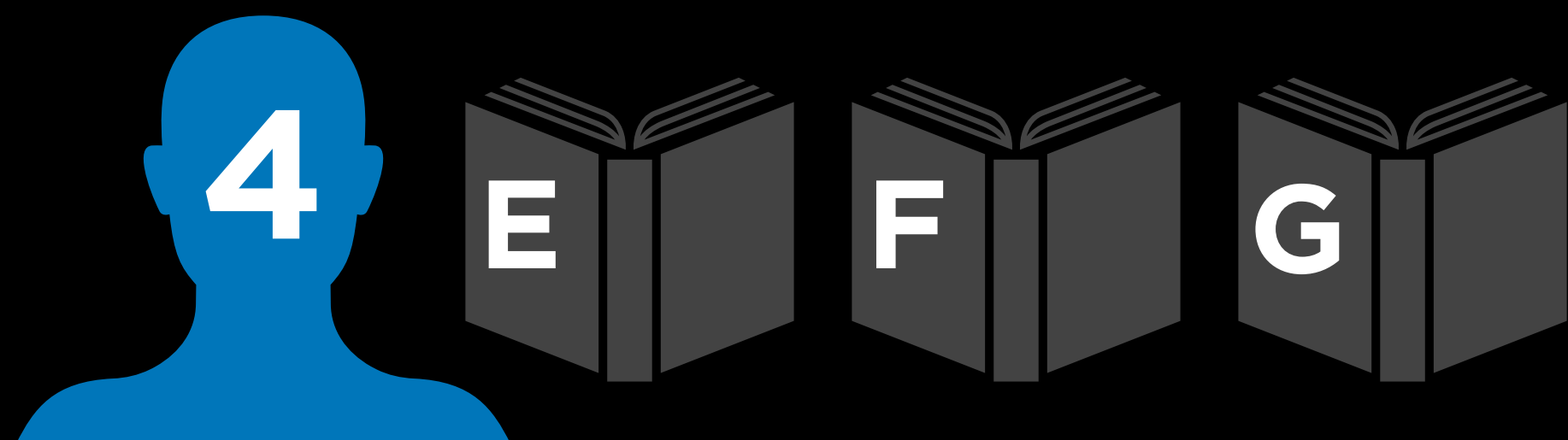
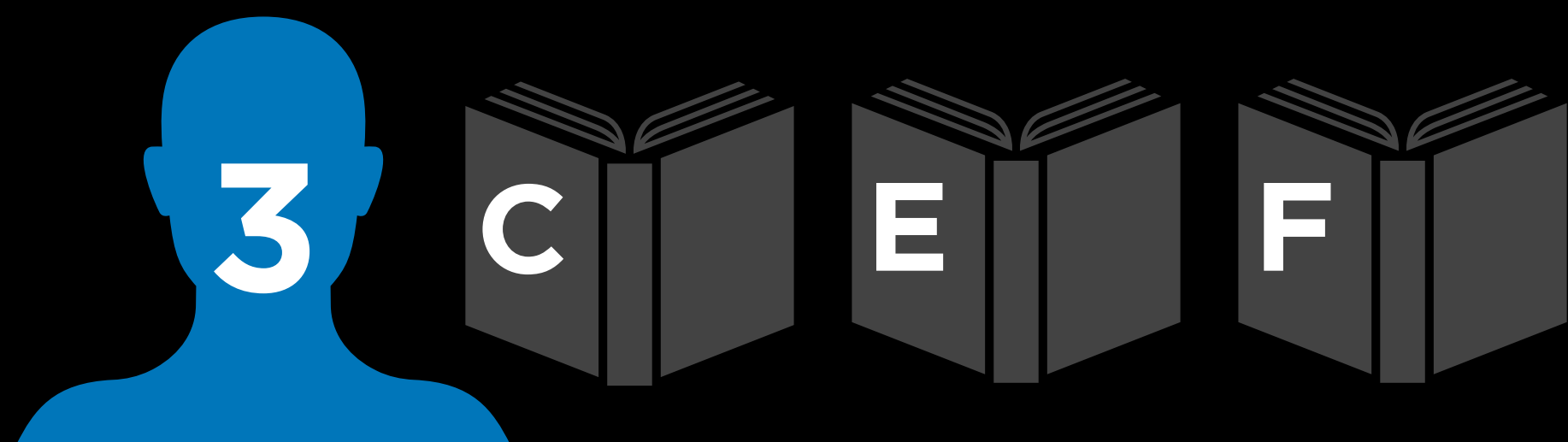
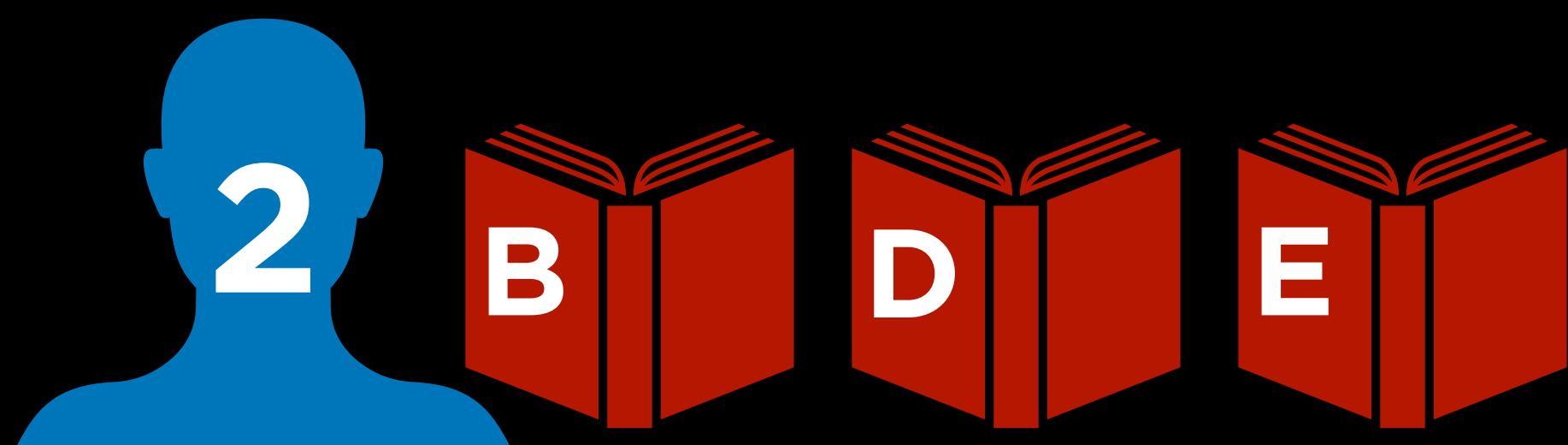
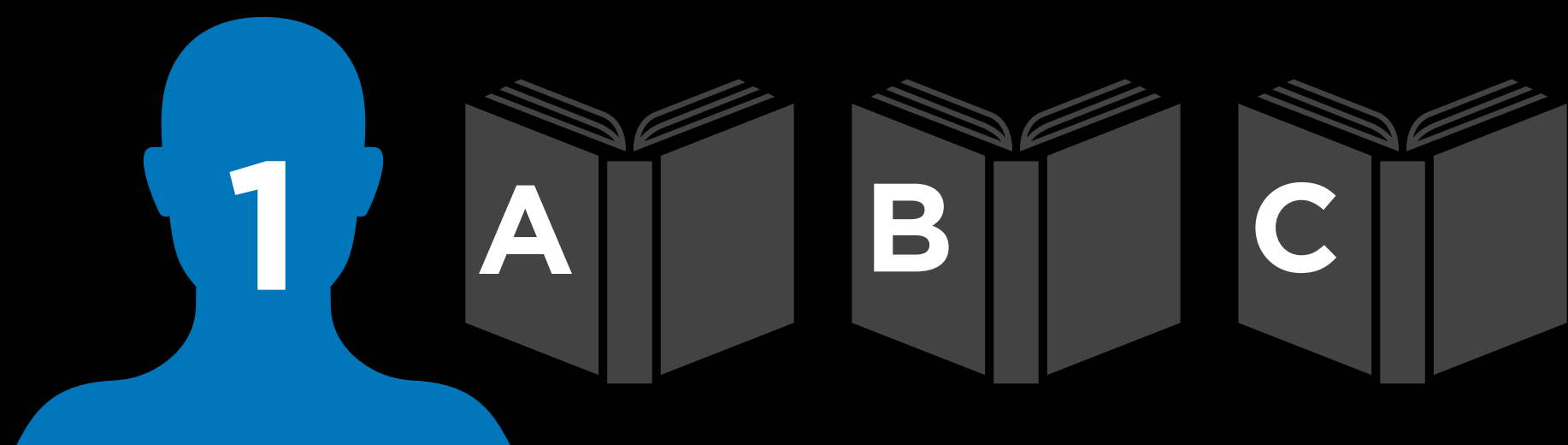
Wednesday

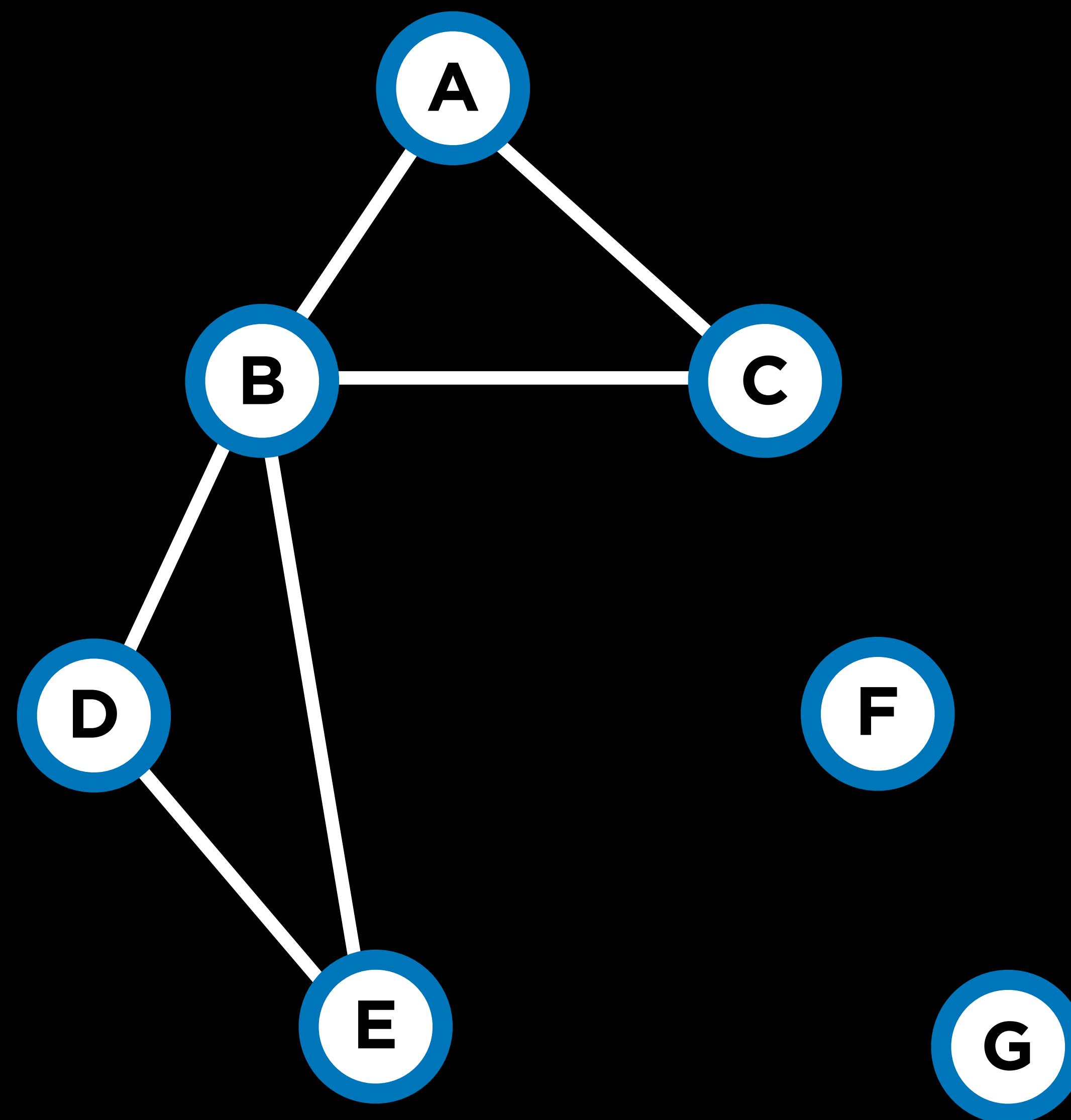
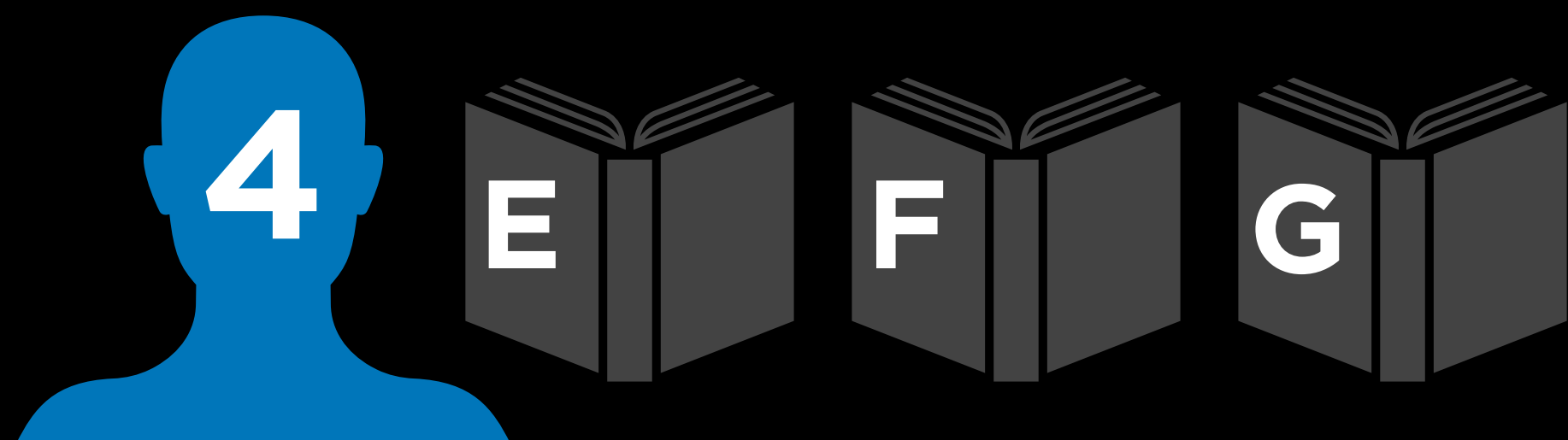
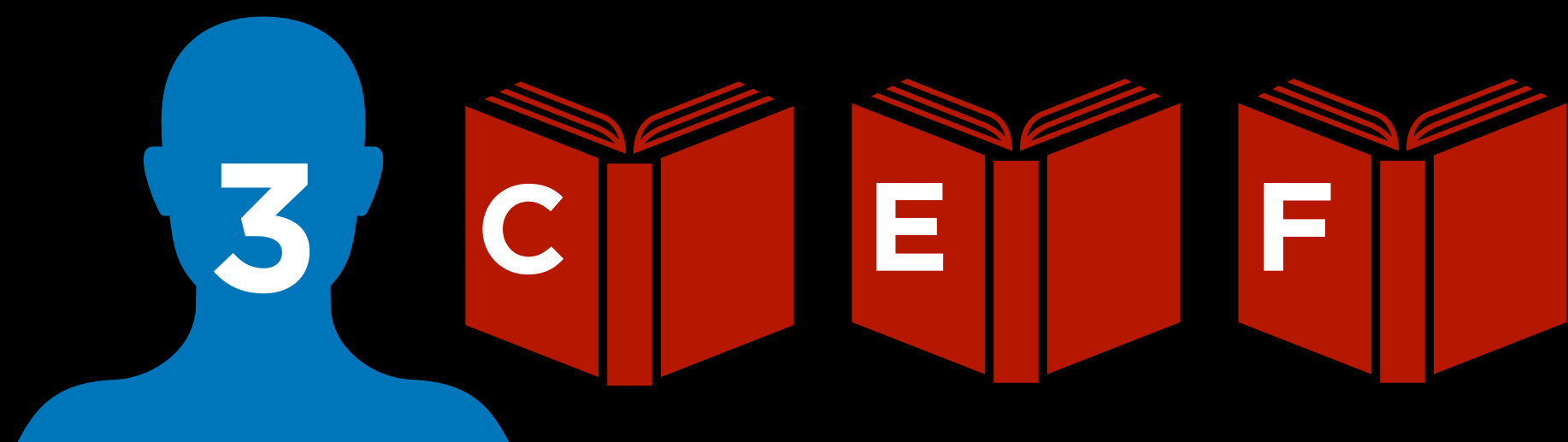
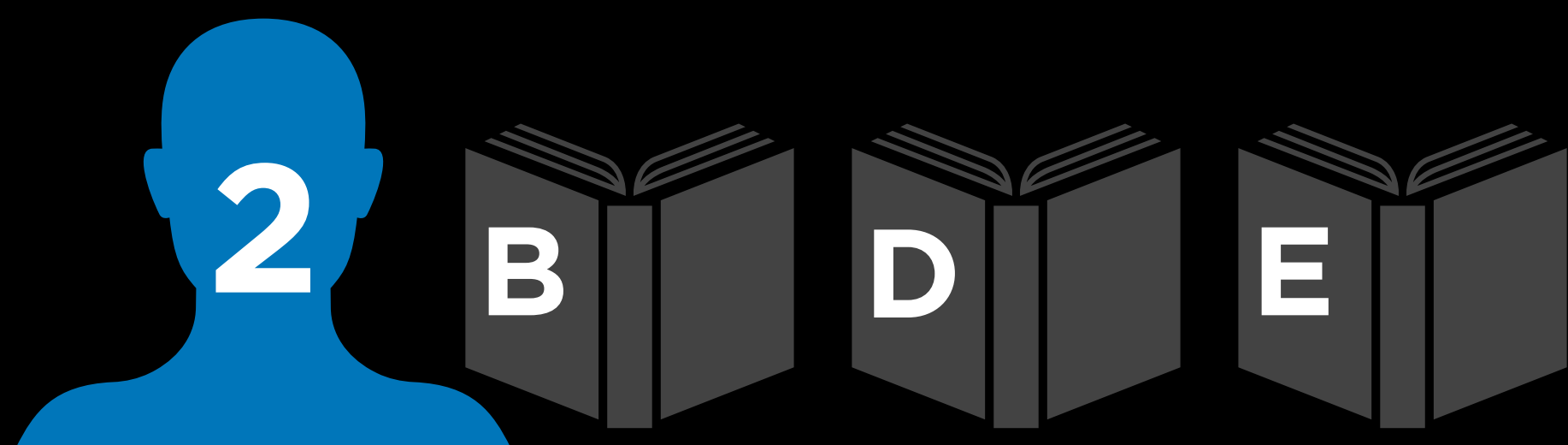
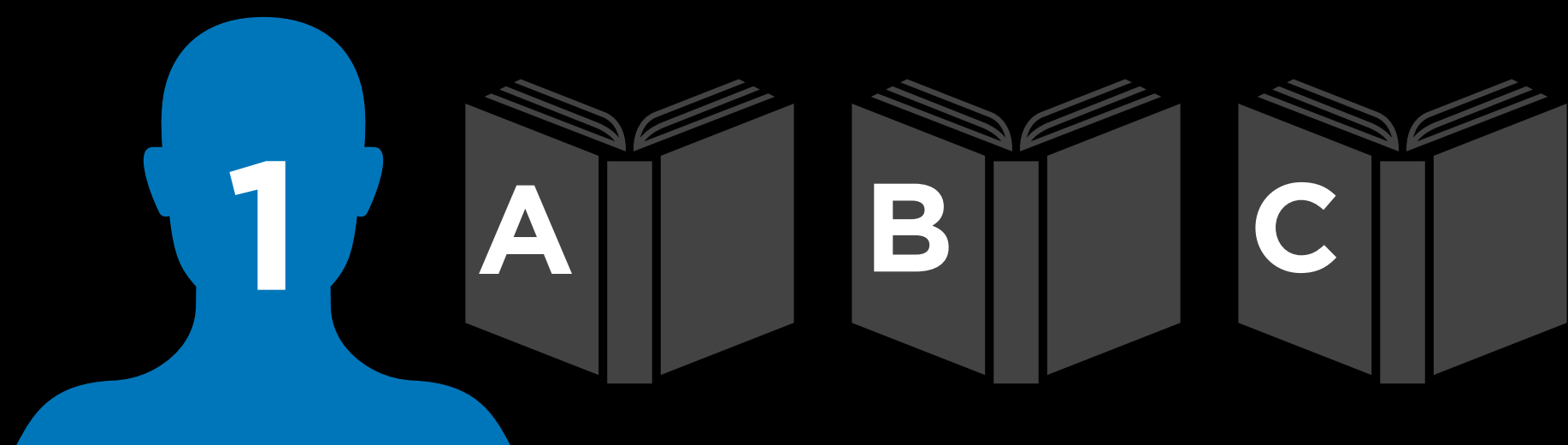


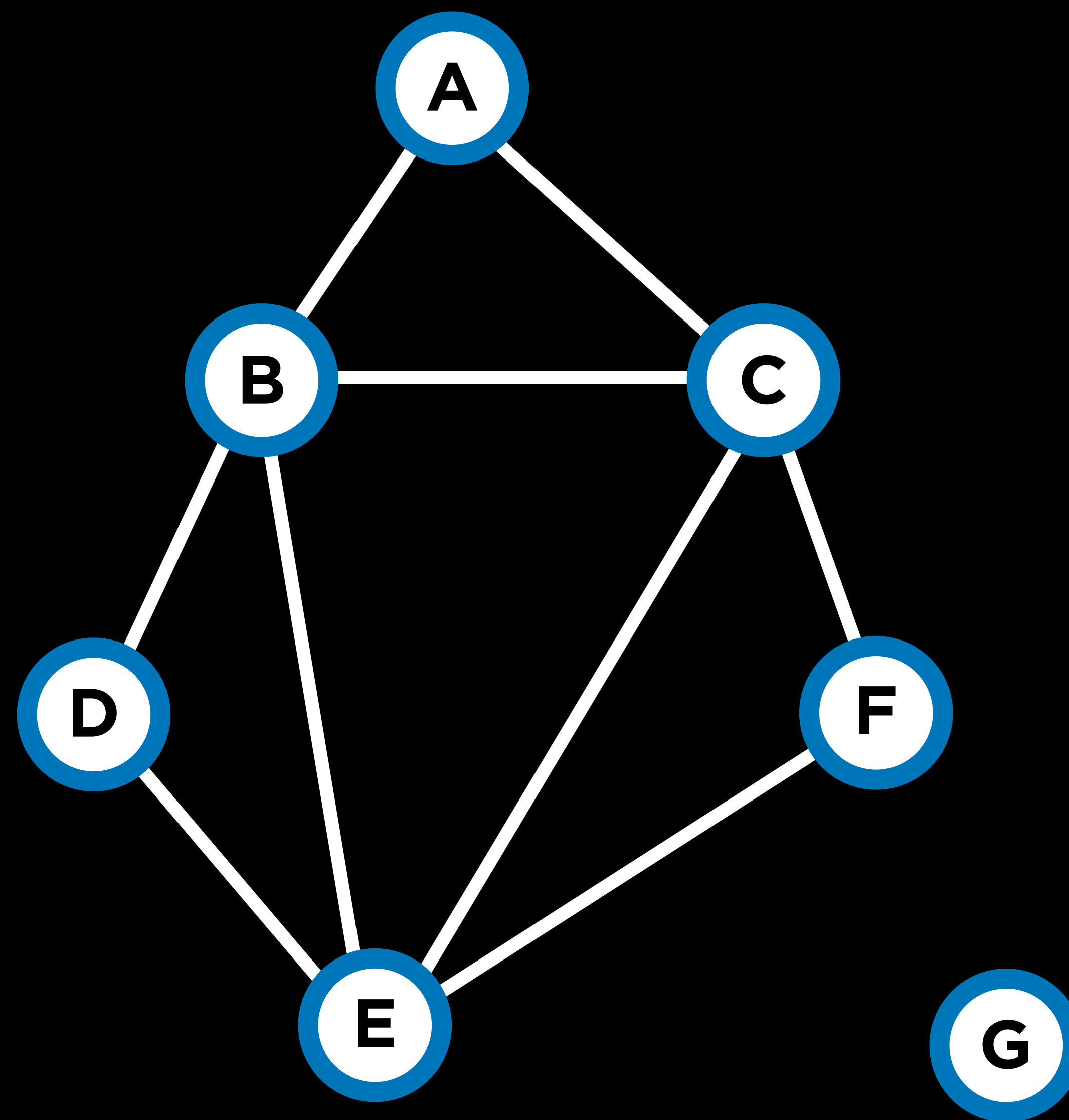
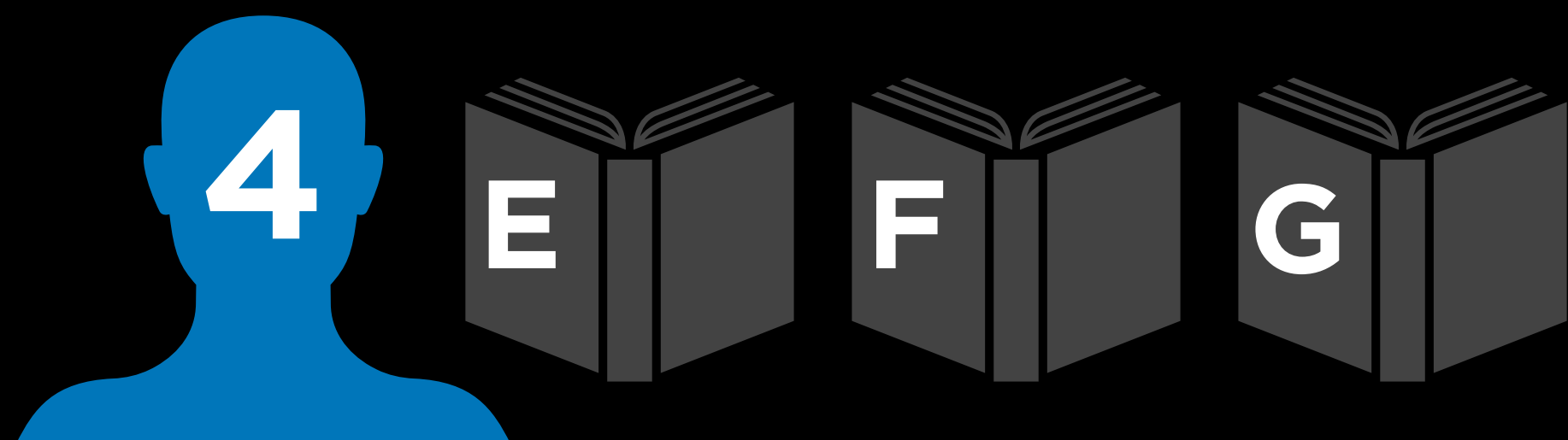
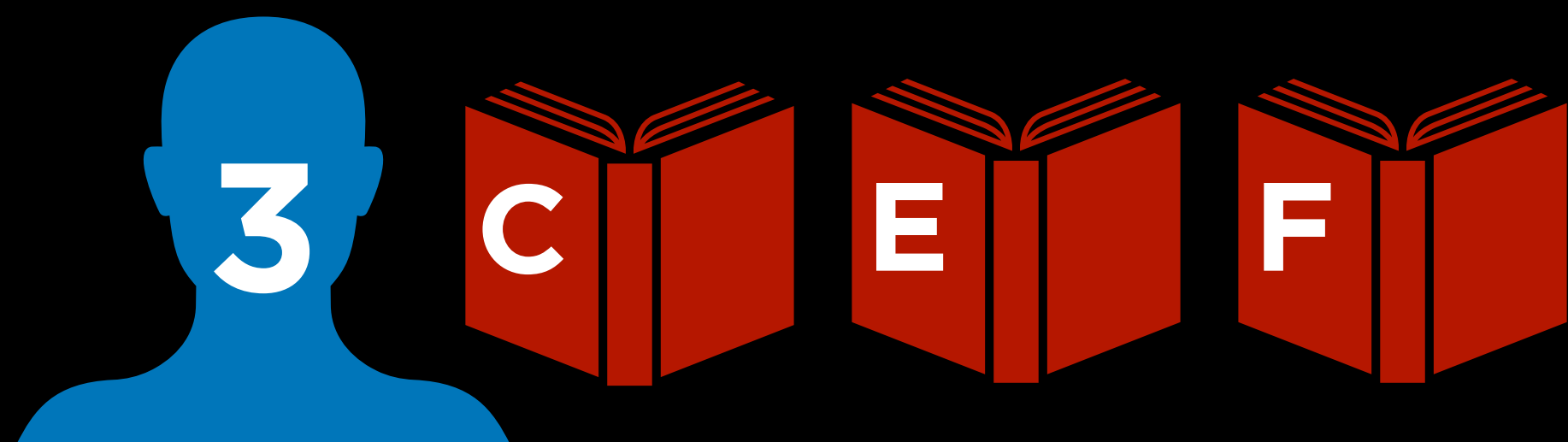
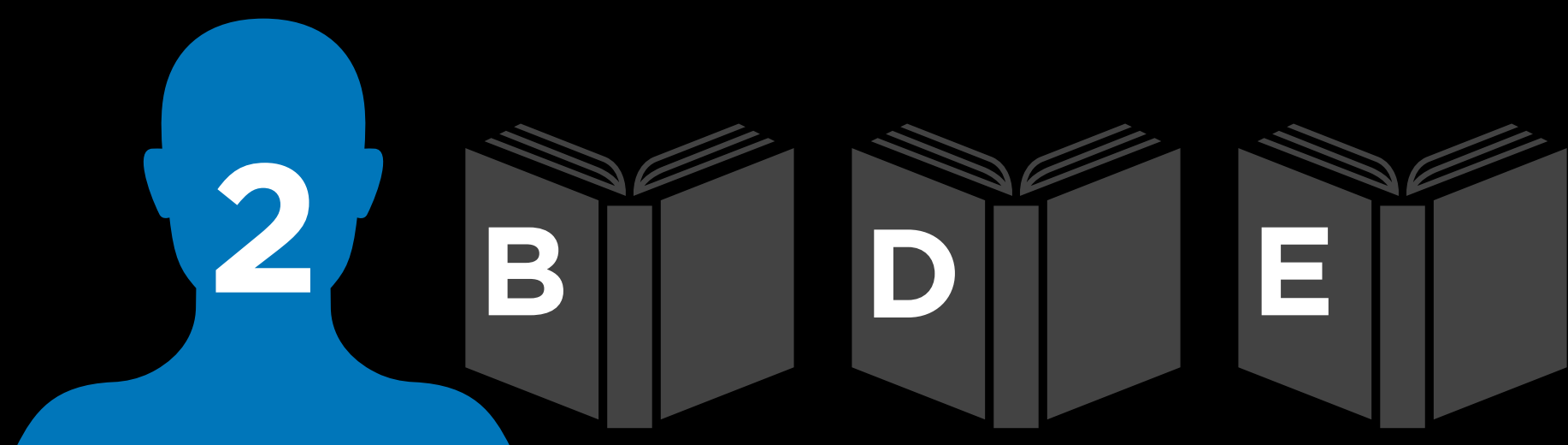
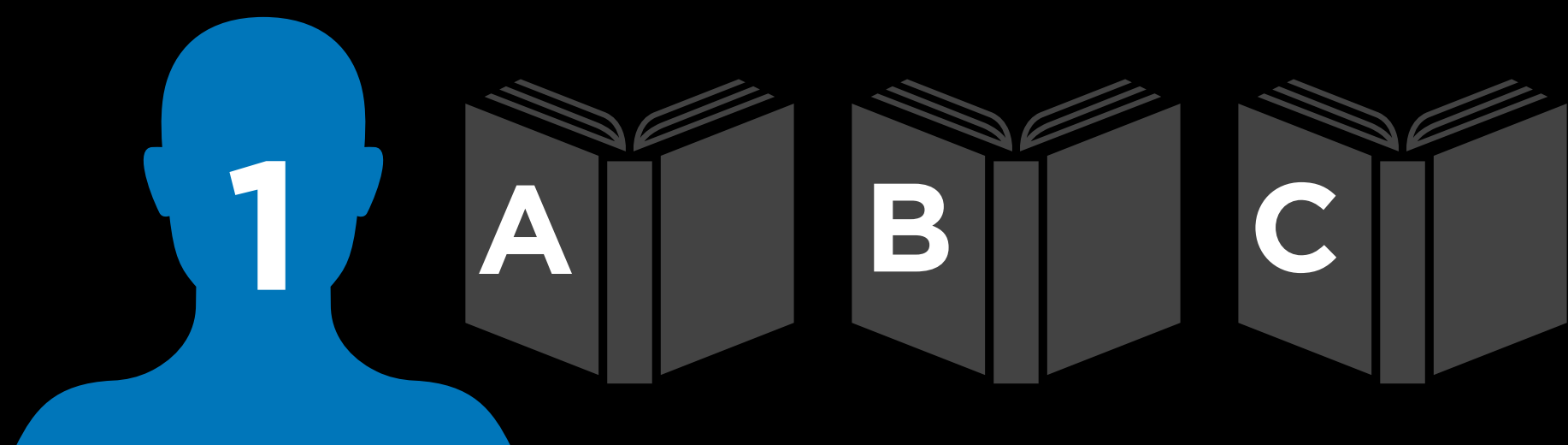


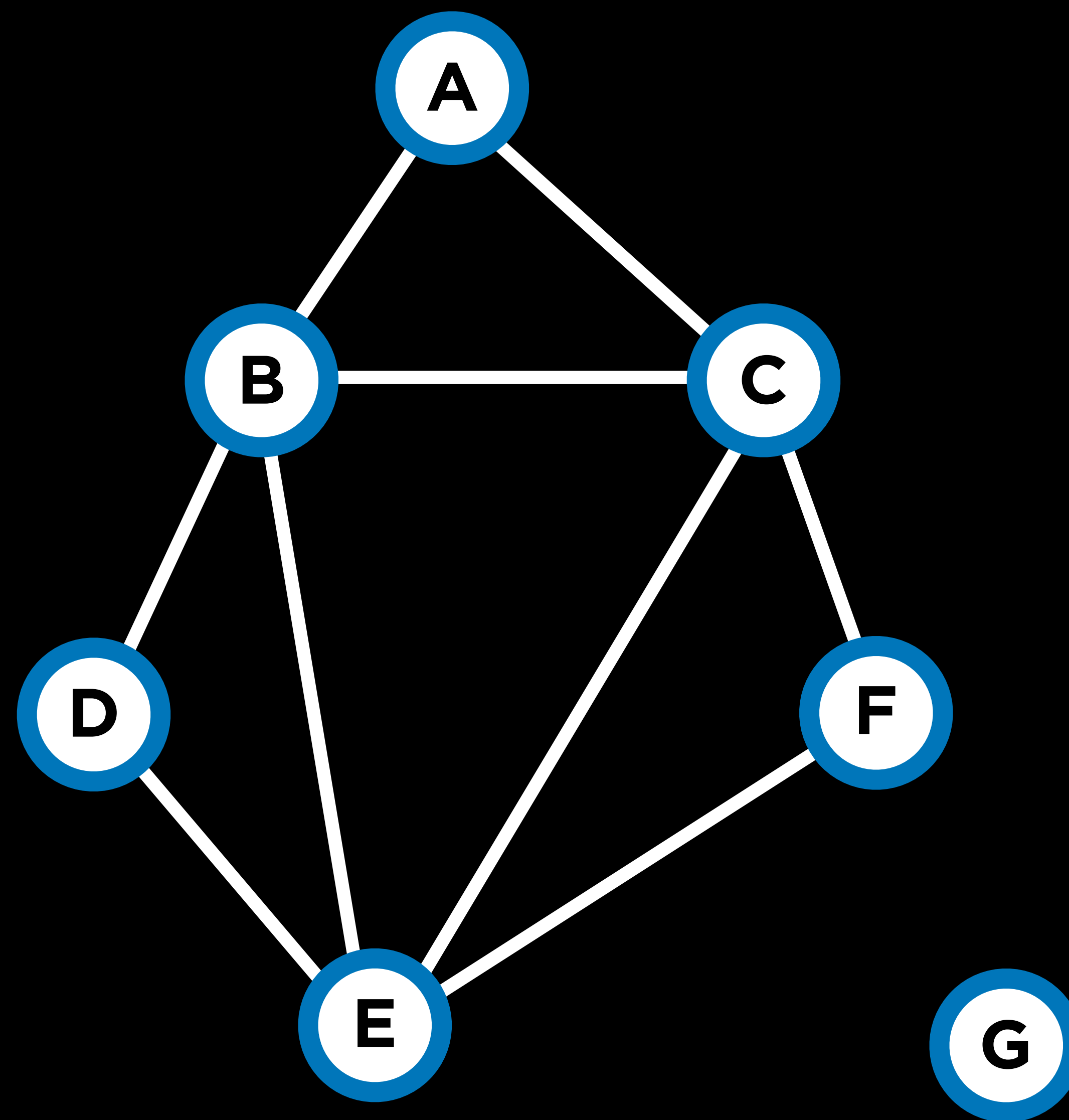
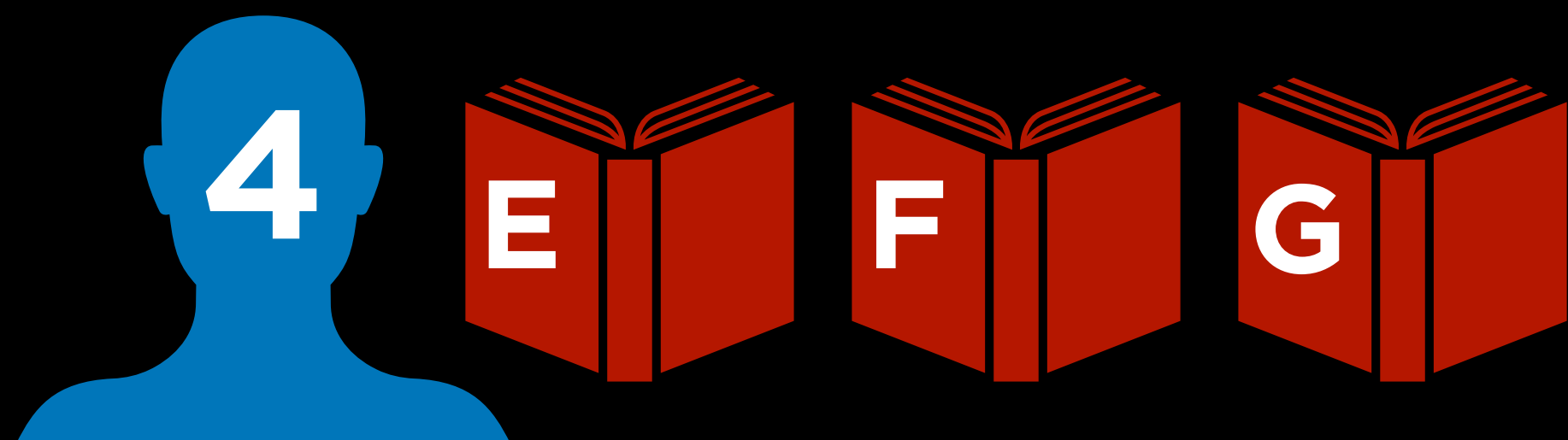
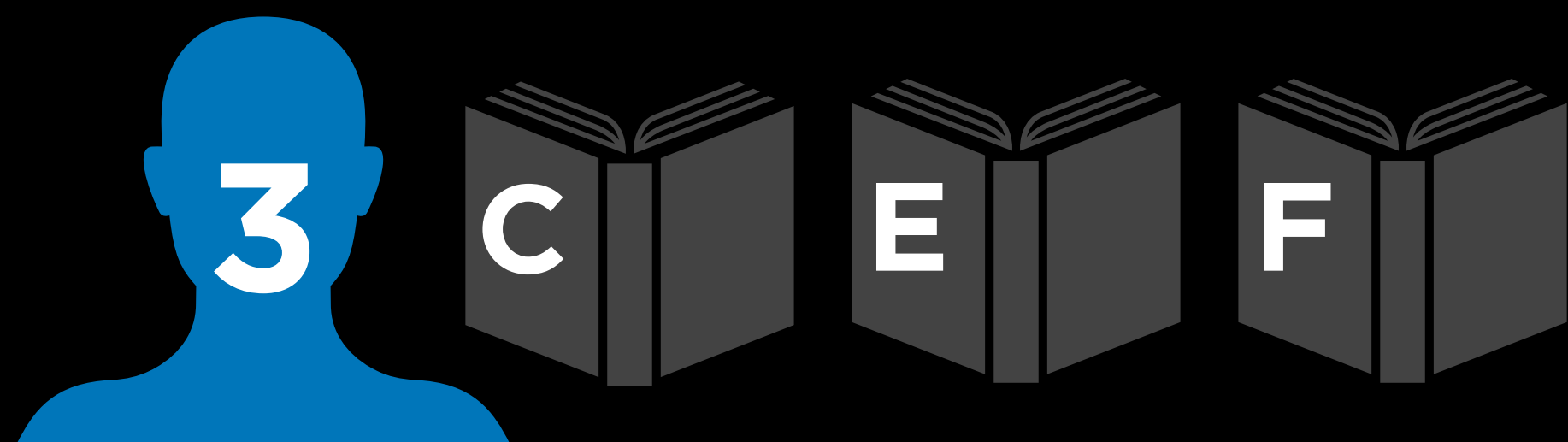
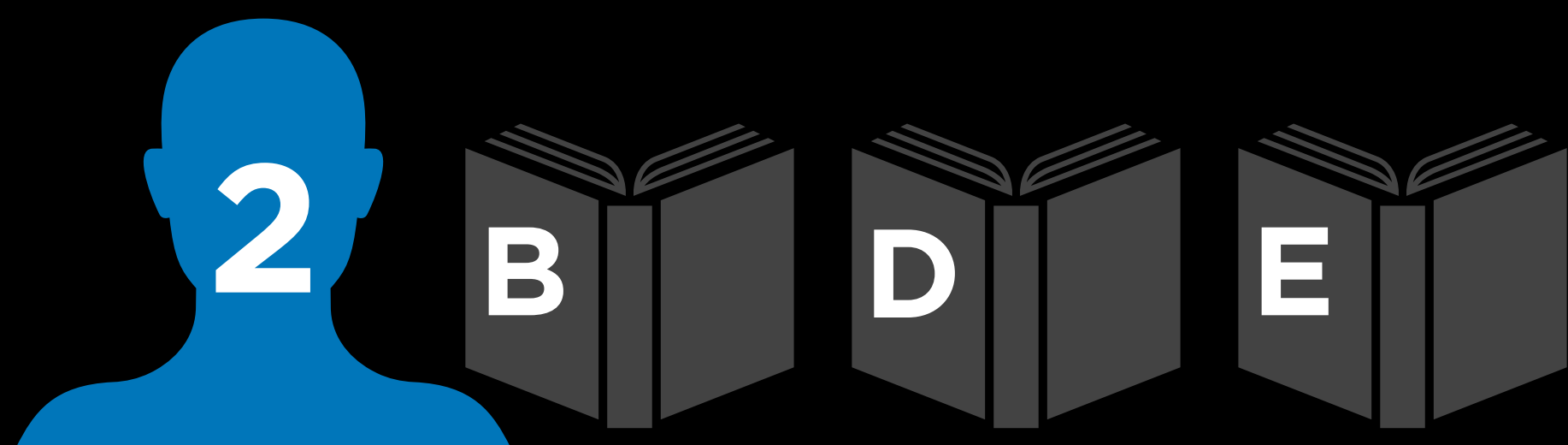
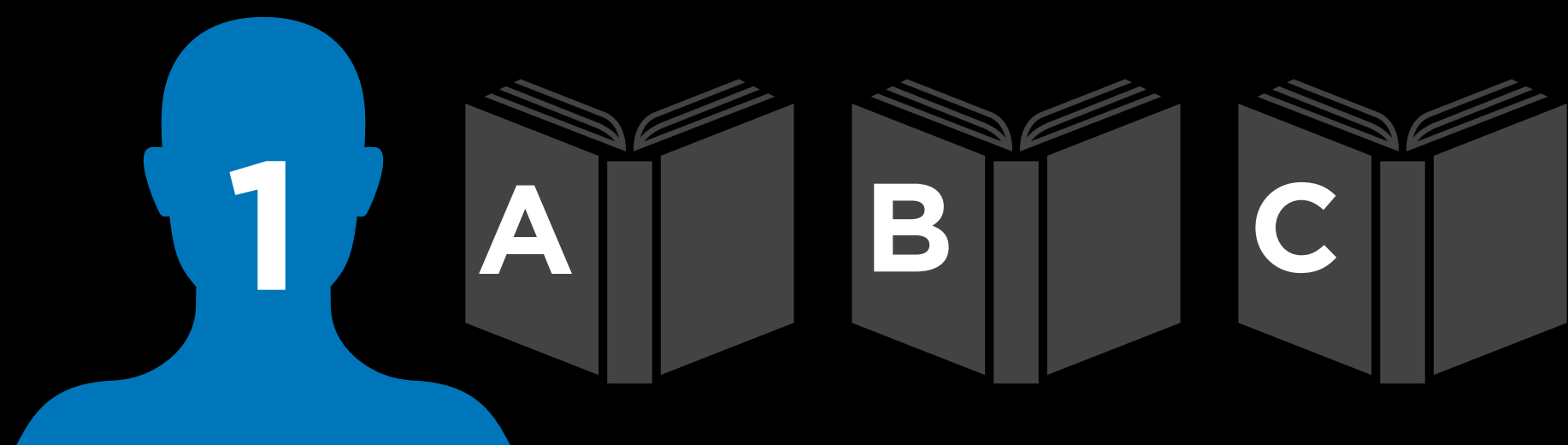


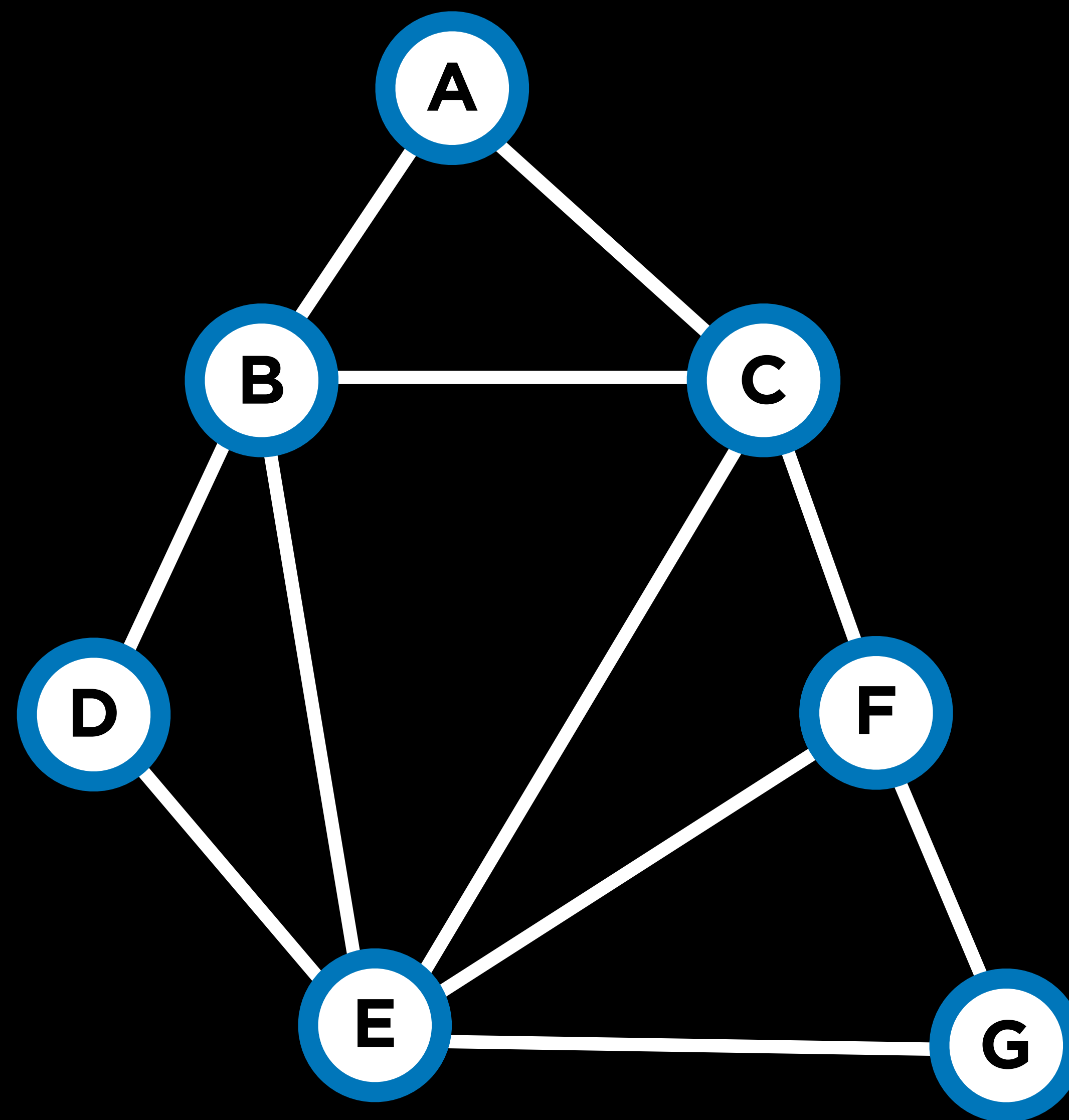
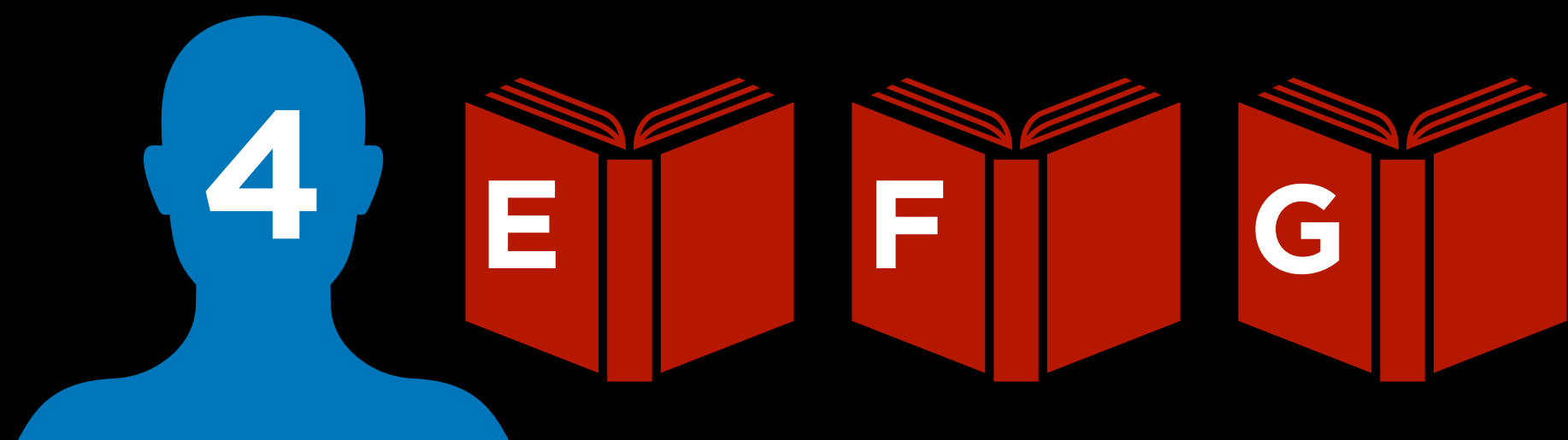
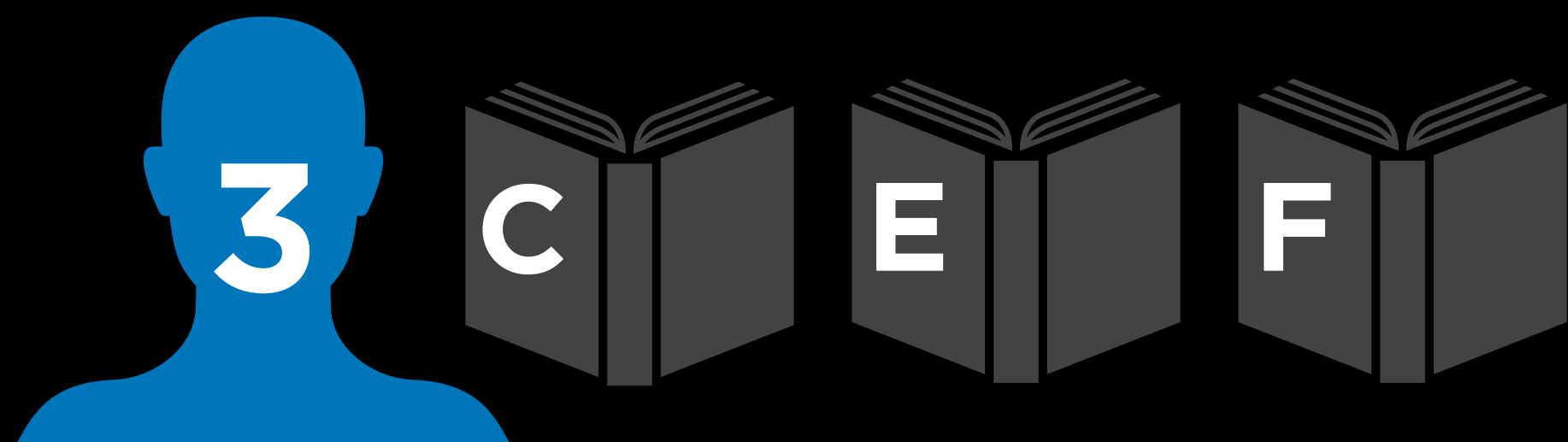
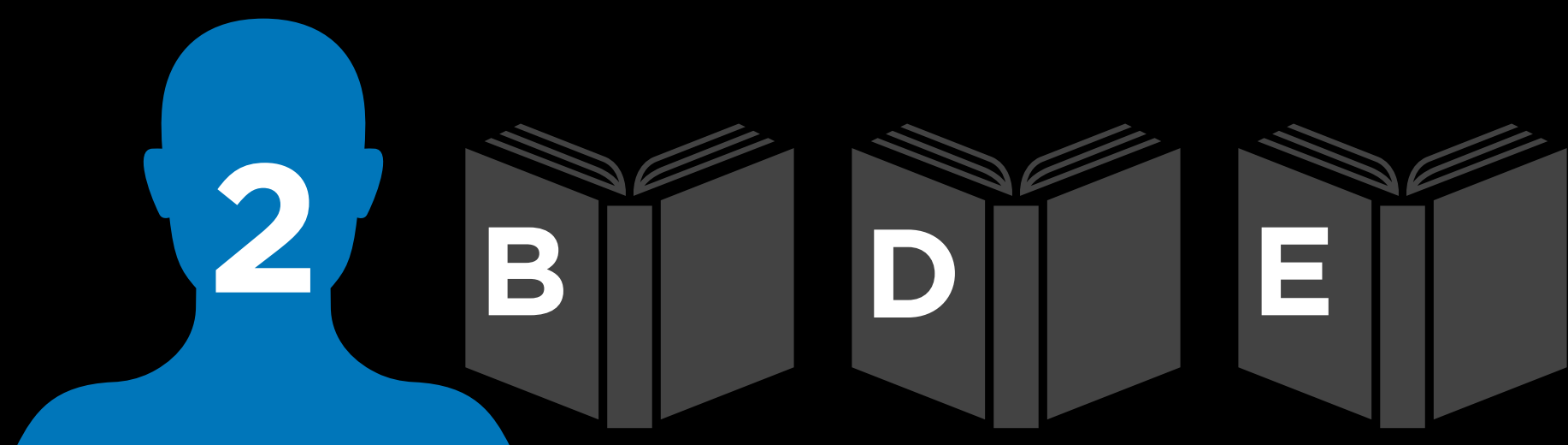
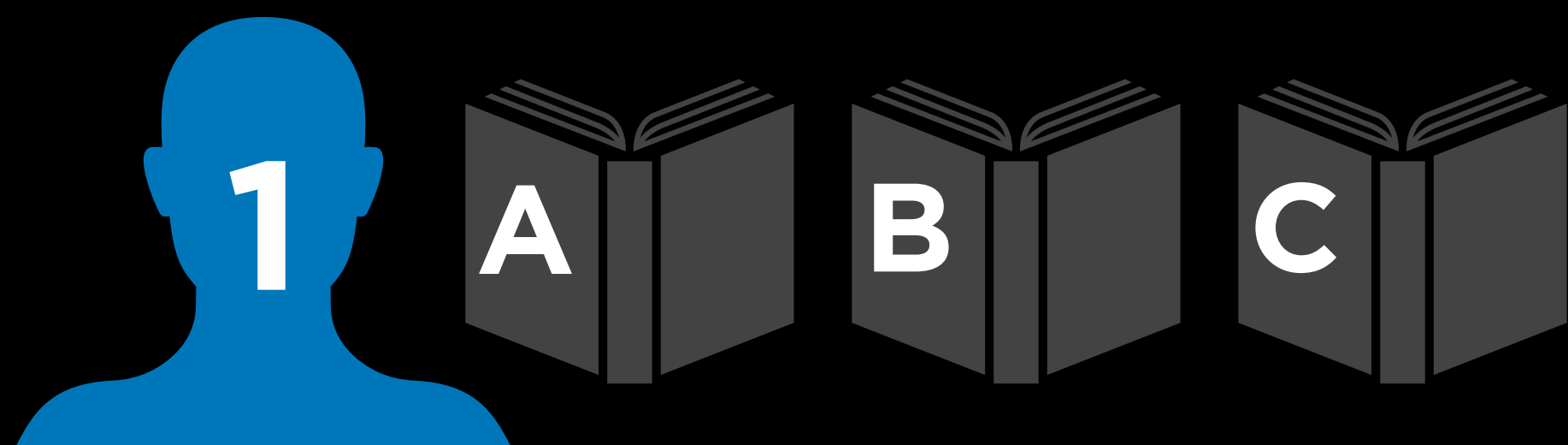


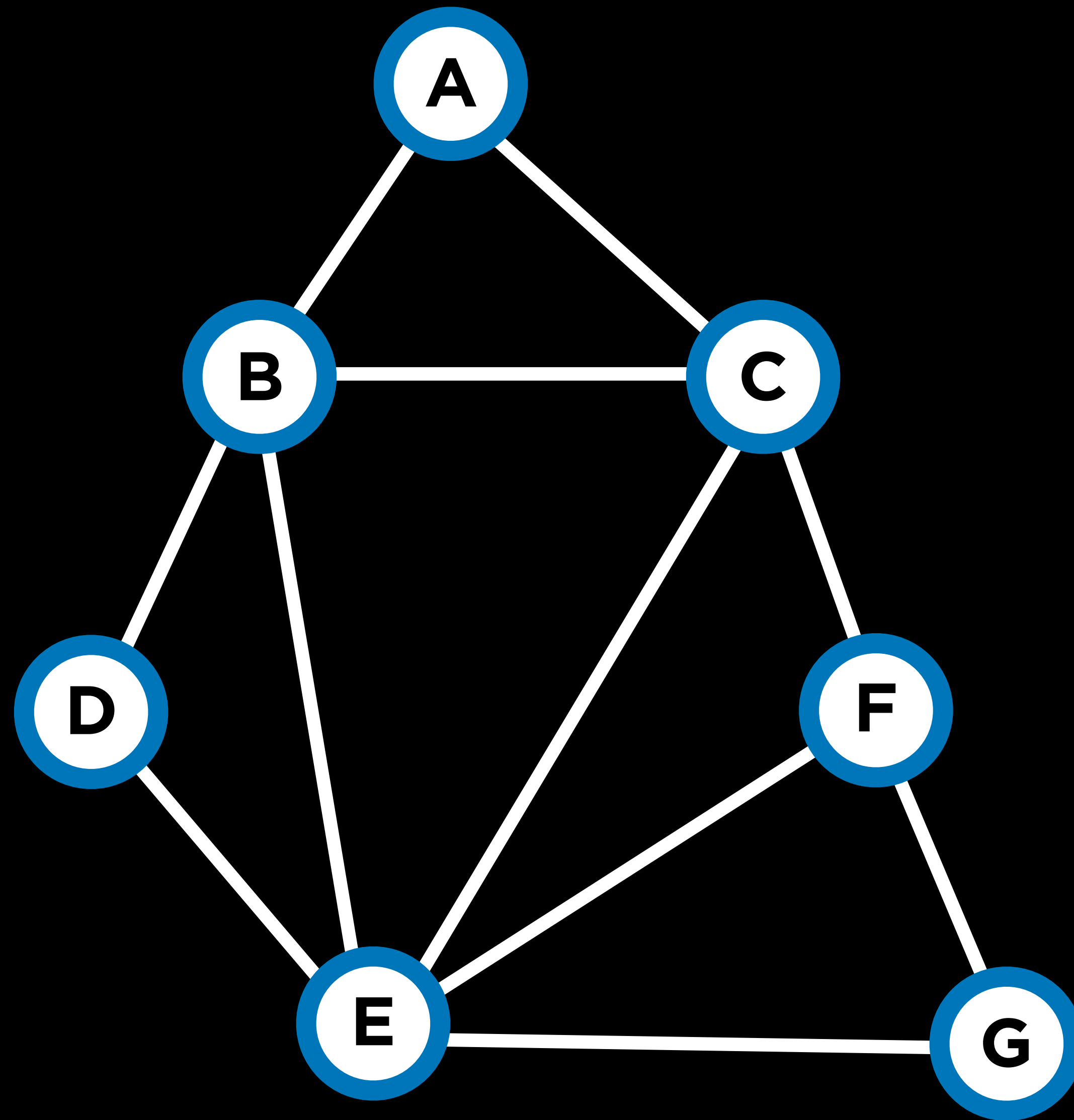












Constraint Satisfaction Problem

- Set of variables $\{X_1, X_2, \dots, X_n\}$
- Set of domains for each variable $\{D_1, D_2, \dots, D_n\}$
- Set of constraints C

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Variables

$\{(0, 2), (1, 1), (1, 2), (2, 0), \dots\}$

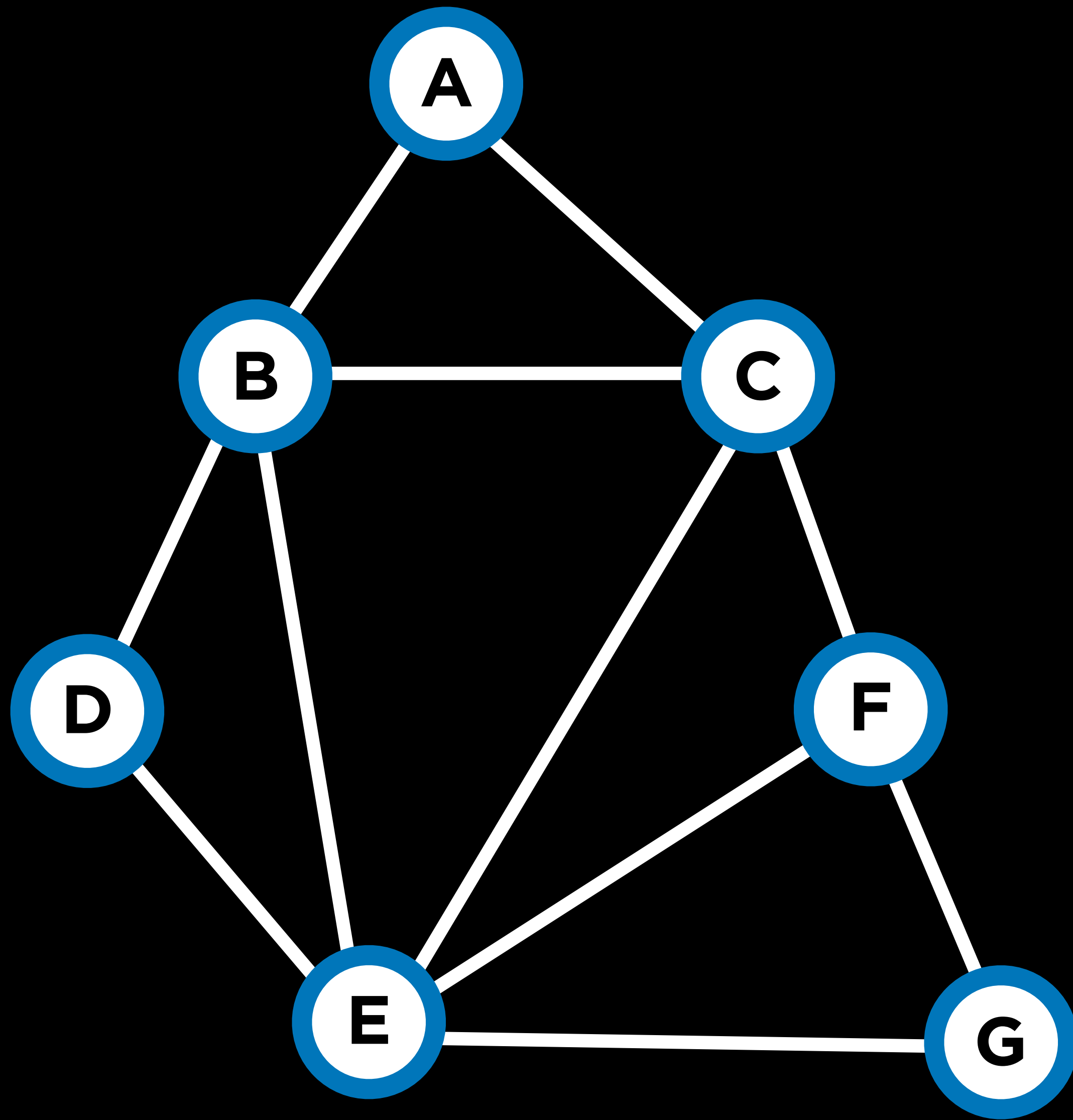
Domains

$\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

for each variable

Constraints

$\{(0, 2) \neq (1, 1) \neq (1, 2) \neq (2, 0), \dots\}$



Variables

$\{A, B, C, D, E, F, G\}$

Domains

$\{Monday, Tuesday, Wednesday\}$

for each variable

Constraints

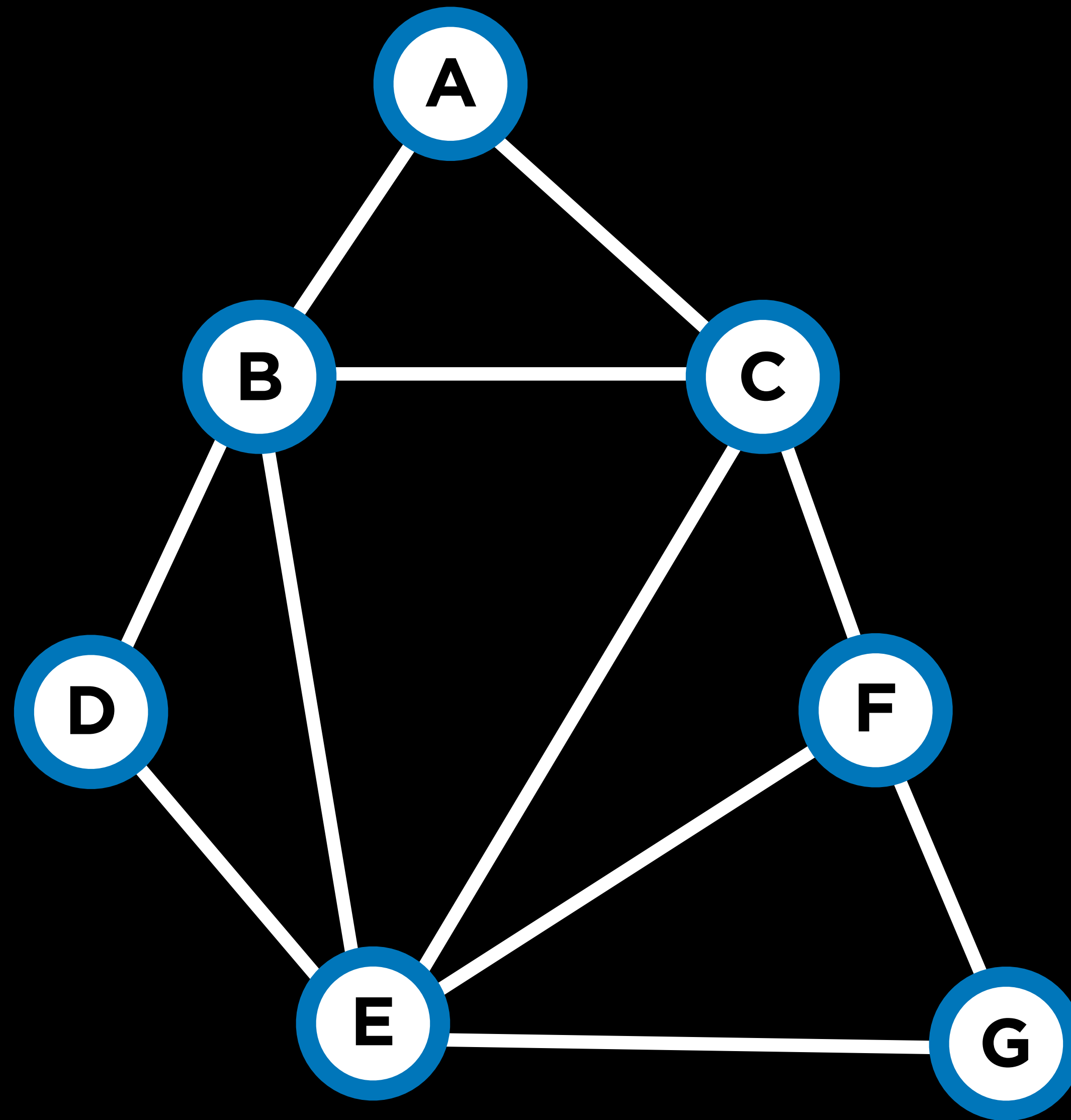
$\{A \neq B, A \neq C, B \neq C, B \neq D, B \neq E, C \neq E, C \neq F, D \neq E, E \neq F, E \neq G, F \neq G\}$

hard constraints

constraints that must be satisfied in a correct solution

soft constraints

constraints that express some notion of which solutions are preferred over others



unary constraint

constraint involving only one variable

unary constraint

$$\{A \neq \textit{Monday}\}$$

binary constraint

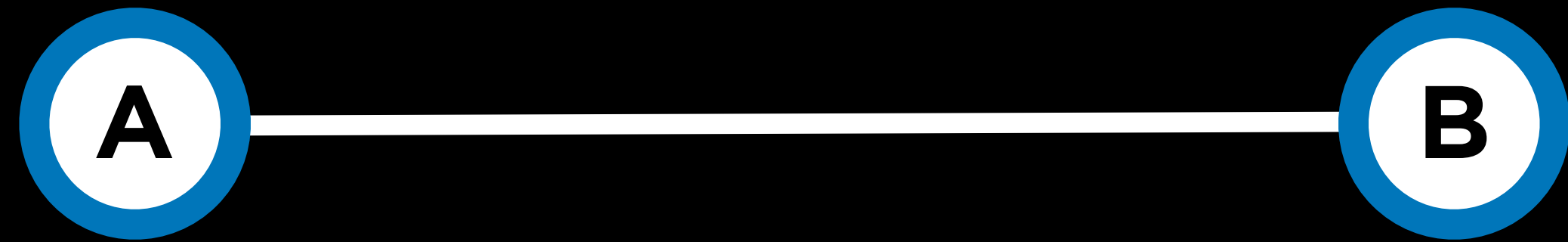
constraint involving two variables

binary constraint

$$\{A \neq B\}$$

node consistency

when all the values in a variable's domain satisfy the variable's unary constraints



$\{Mon, Tue, Wed\}$

$\{Mon, Tue, Wed\}$

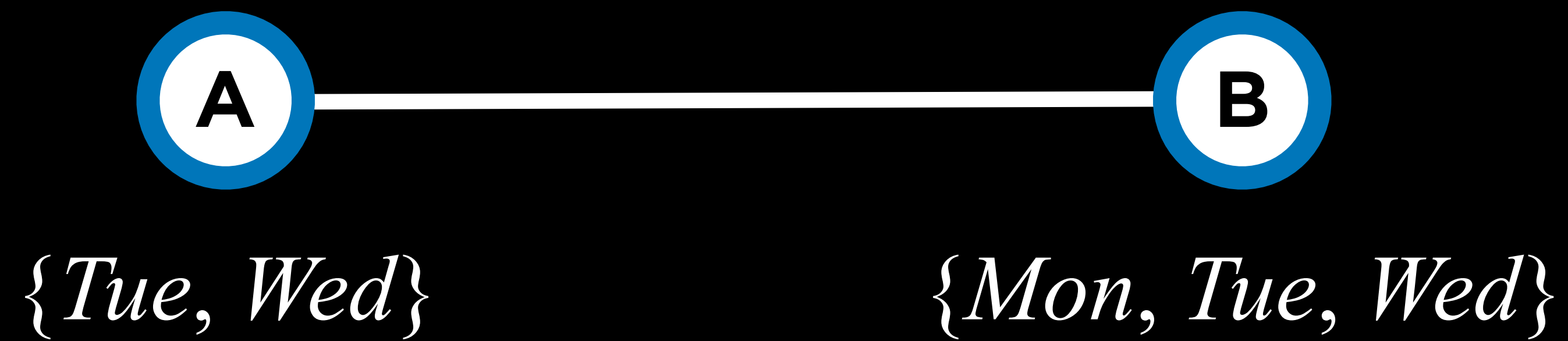
$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$



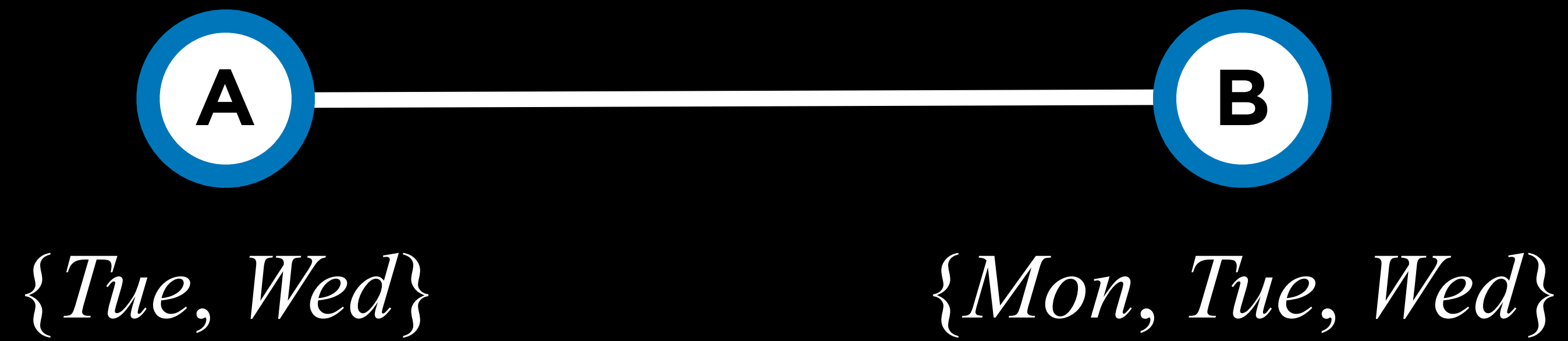
$\{Mon, Tue, Wed\}$

$\{Mon, Tue, Wed\}$

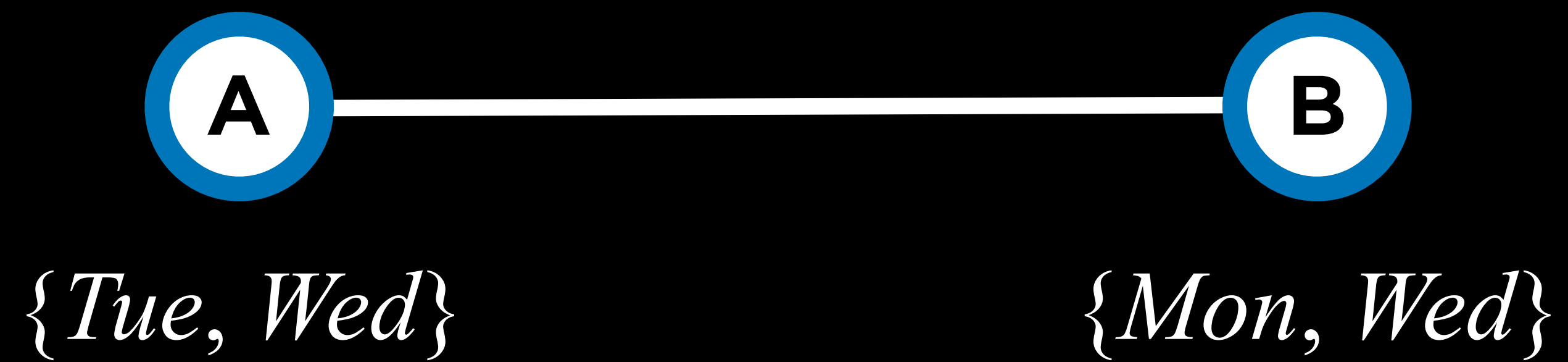
$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$



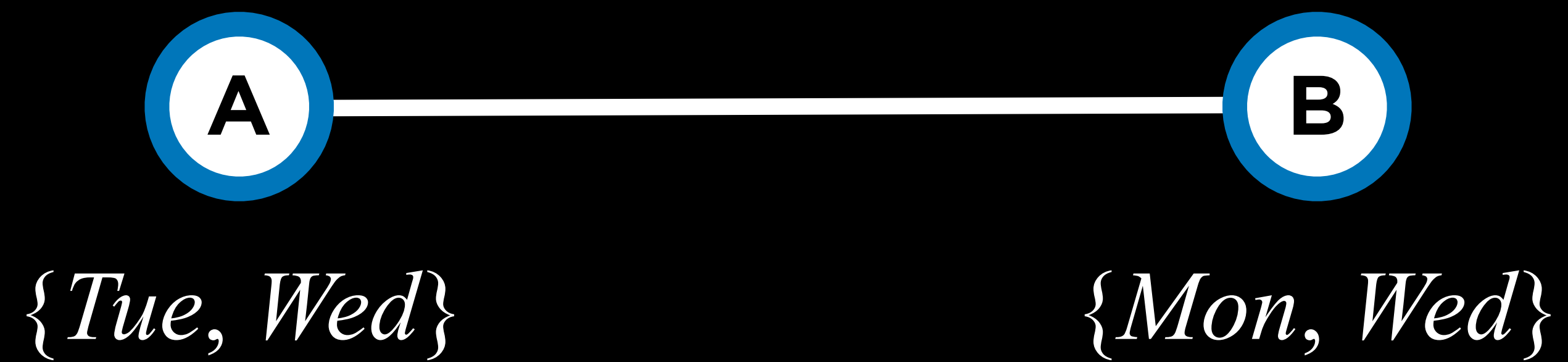
$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$



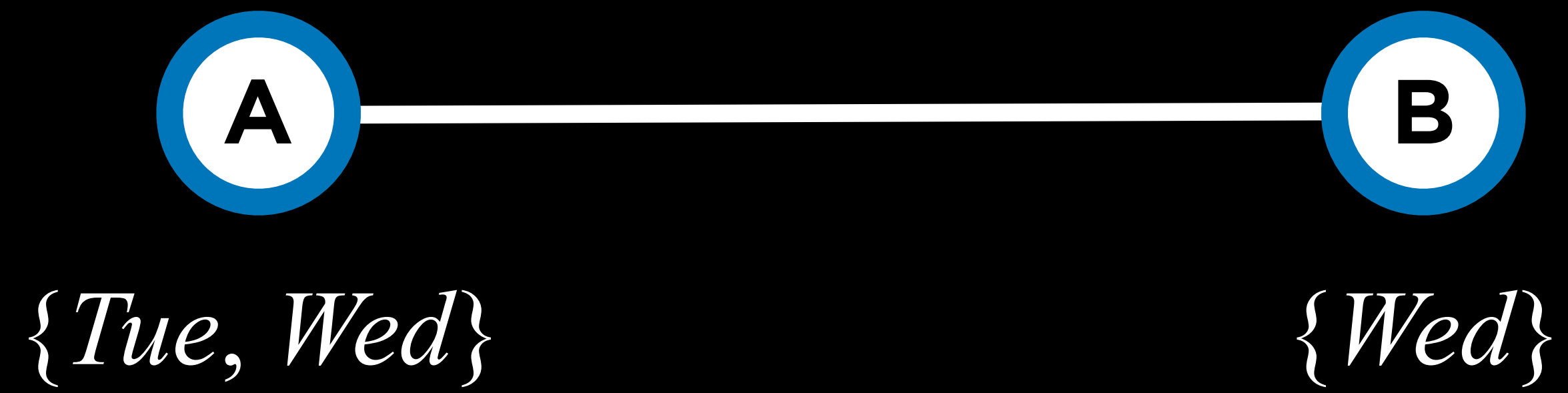
$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$



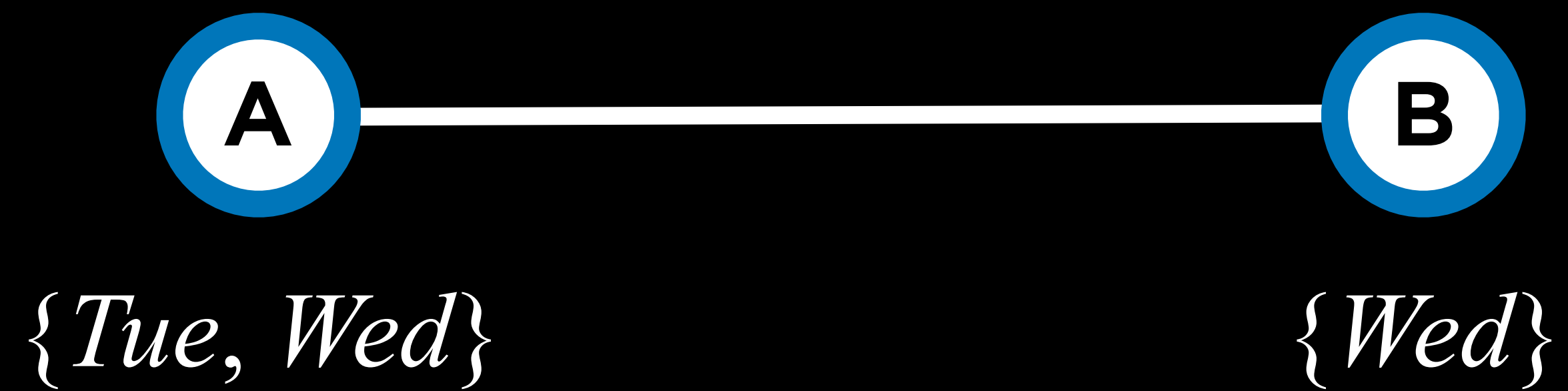
$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$



$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$



$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$



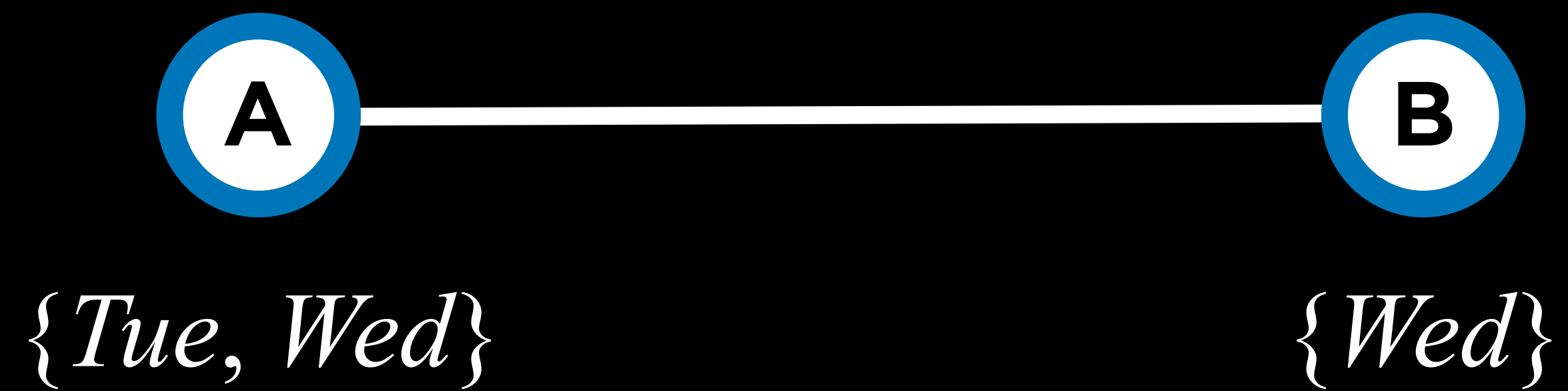
$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$

arc consistency

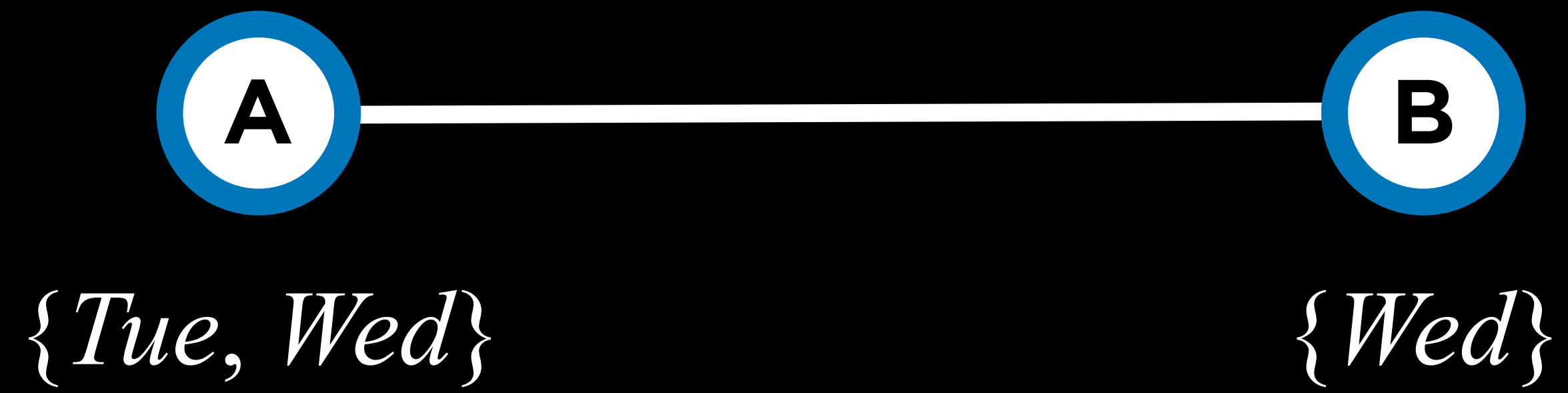
when all the values in a variable's domain satisfy the variable's binary constraints

arc consistency

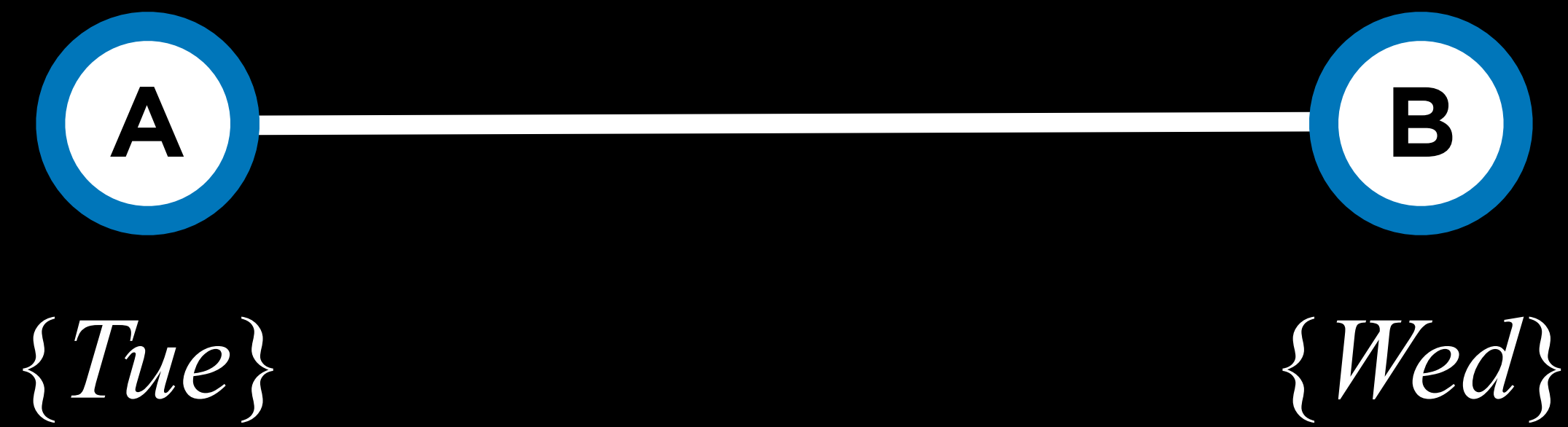
To make X arc-consistent with respect to Y ,
remove elements from X 's domain until every
choice for X has a possible choice for Y



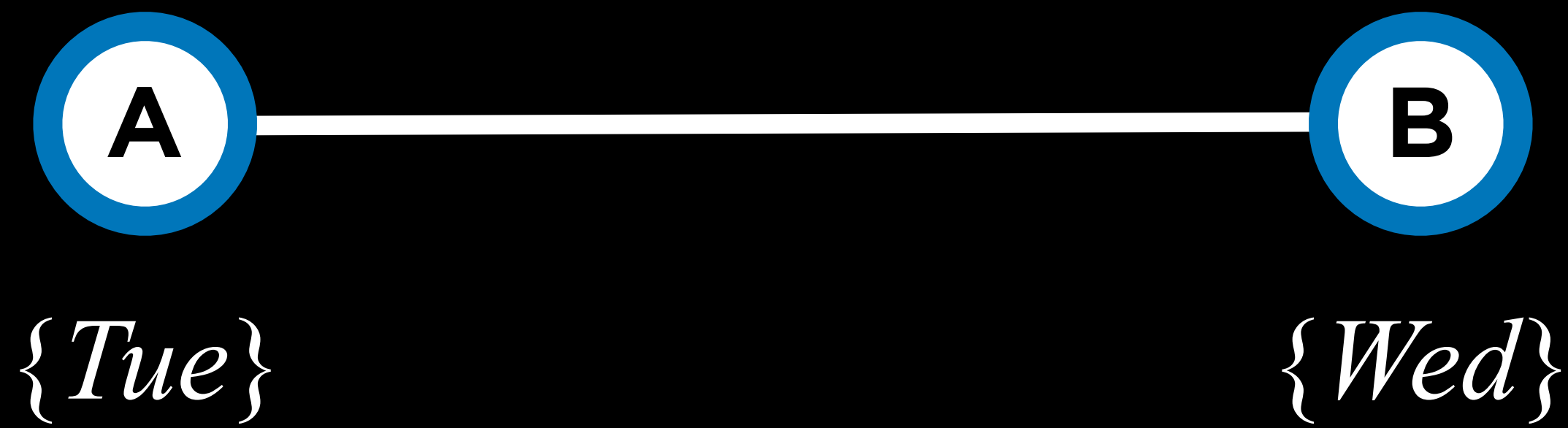
$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$



$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$



$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$



$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$

Arc Consistency

```
function REVISE(csp, X, Y):  
    revised = false  
    for x in X.domain:  
        if no y in Y.domain satisfies constraint for (X, Y):  
            delete x from X.domain  
            revised = true  
    return revised
```

Arc Consistency

function AC-3(csp):

$queue =$ all arcs in csp

 while $queue$ non-empty:

$(X, Y) =$ DEQUEUE($queue$)

 if REVISE(csp, X, Y):

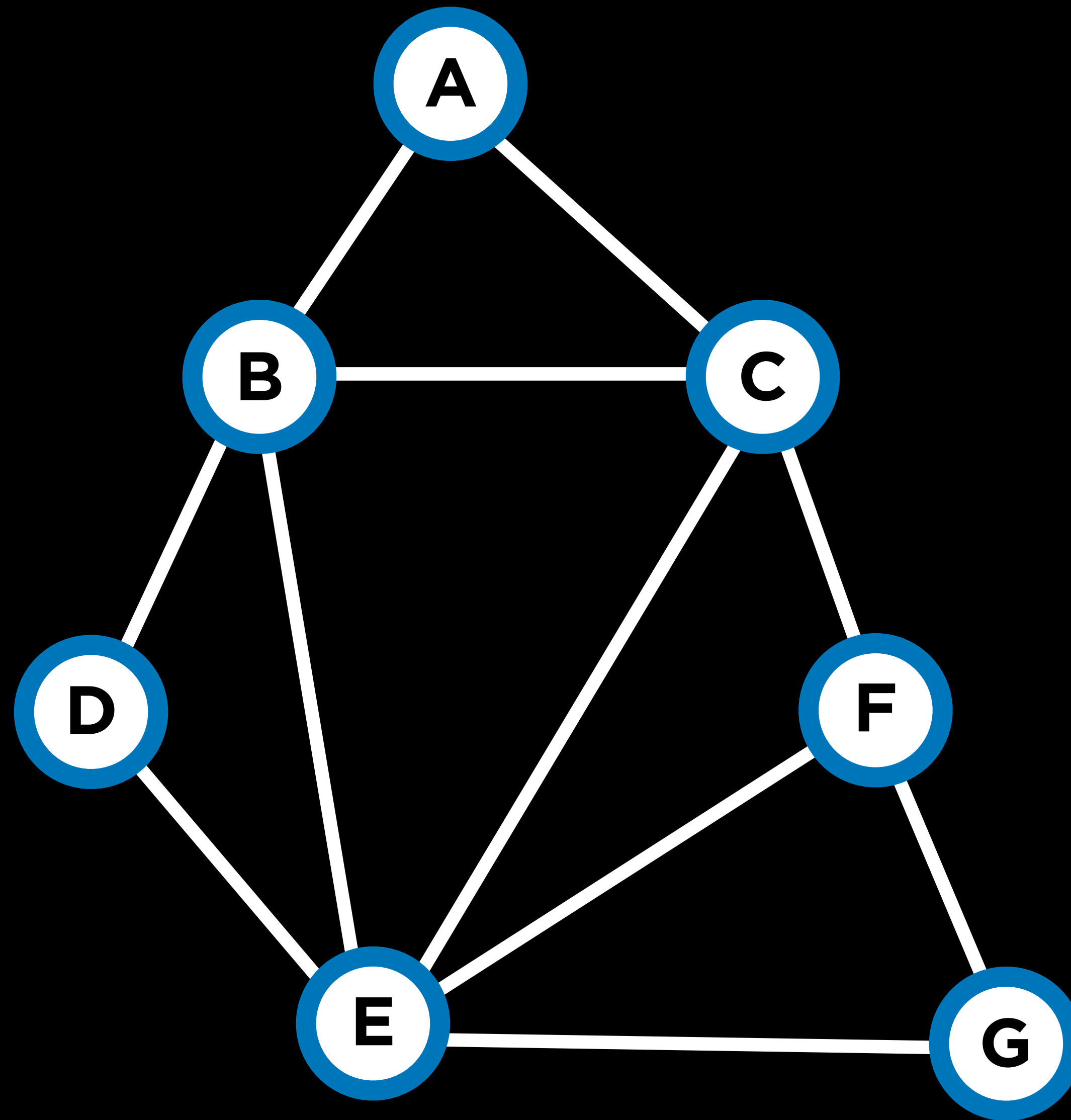
 if size of $X.domain == 0$:

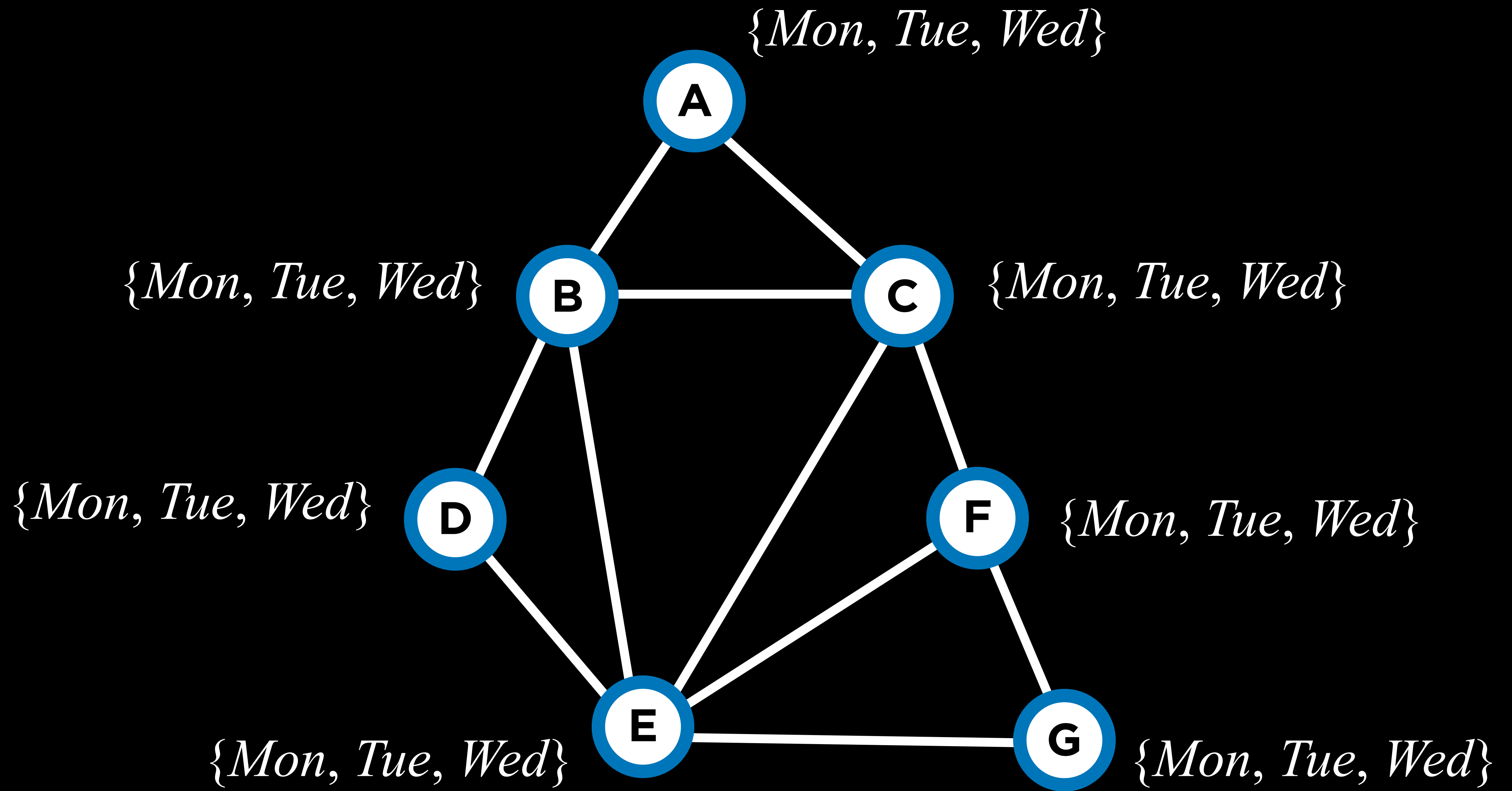
 return *false*

 for each Z in $X.neighbors - \{Y\}$:

 ENQUEUE($queue, (Z, X)$)

 return *true*





Search Problems

- initial state
- actions
- transition model
- goal test
- path cost function

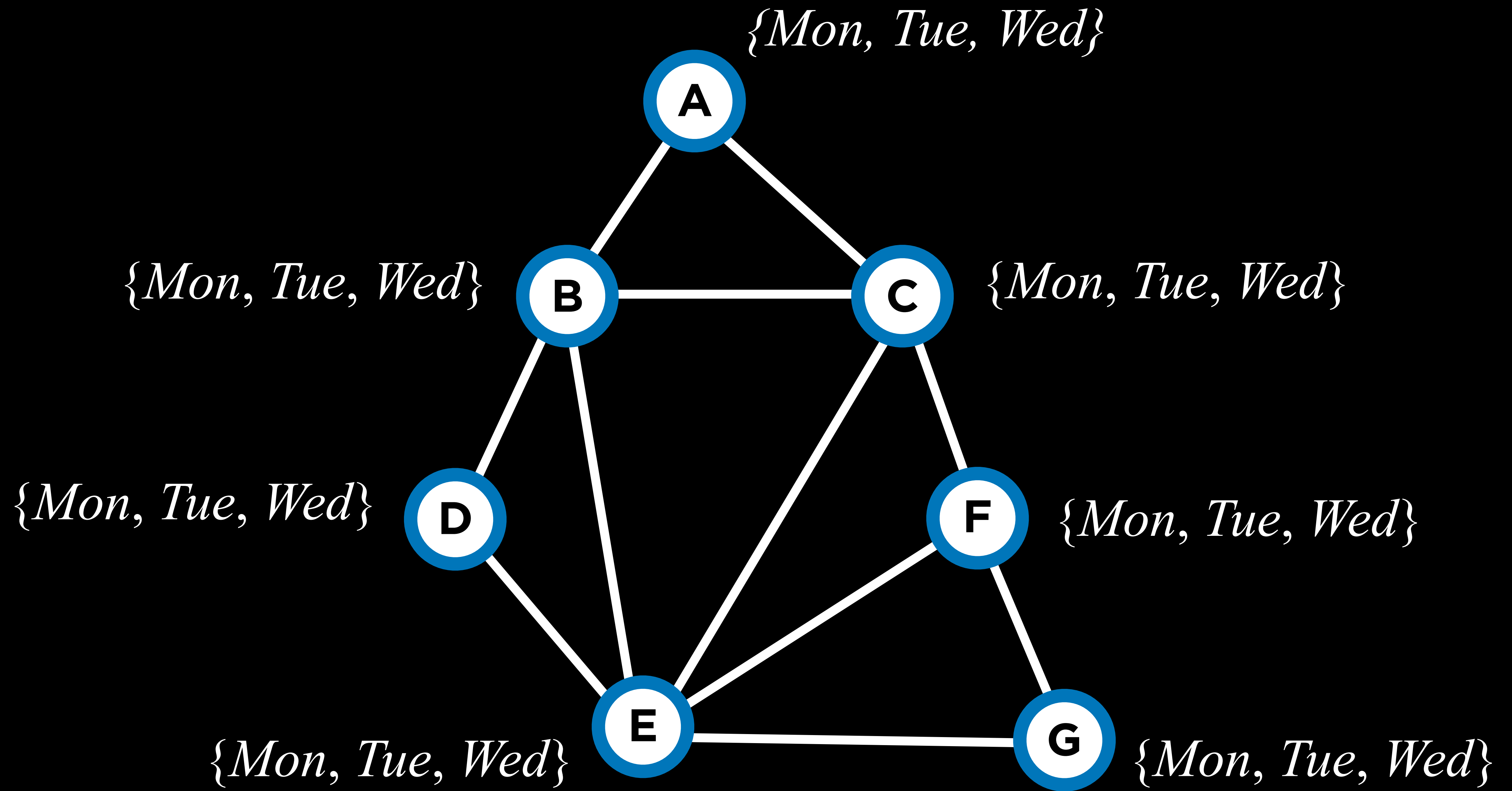
CSPs as Search Problems

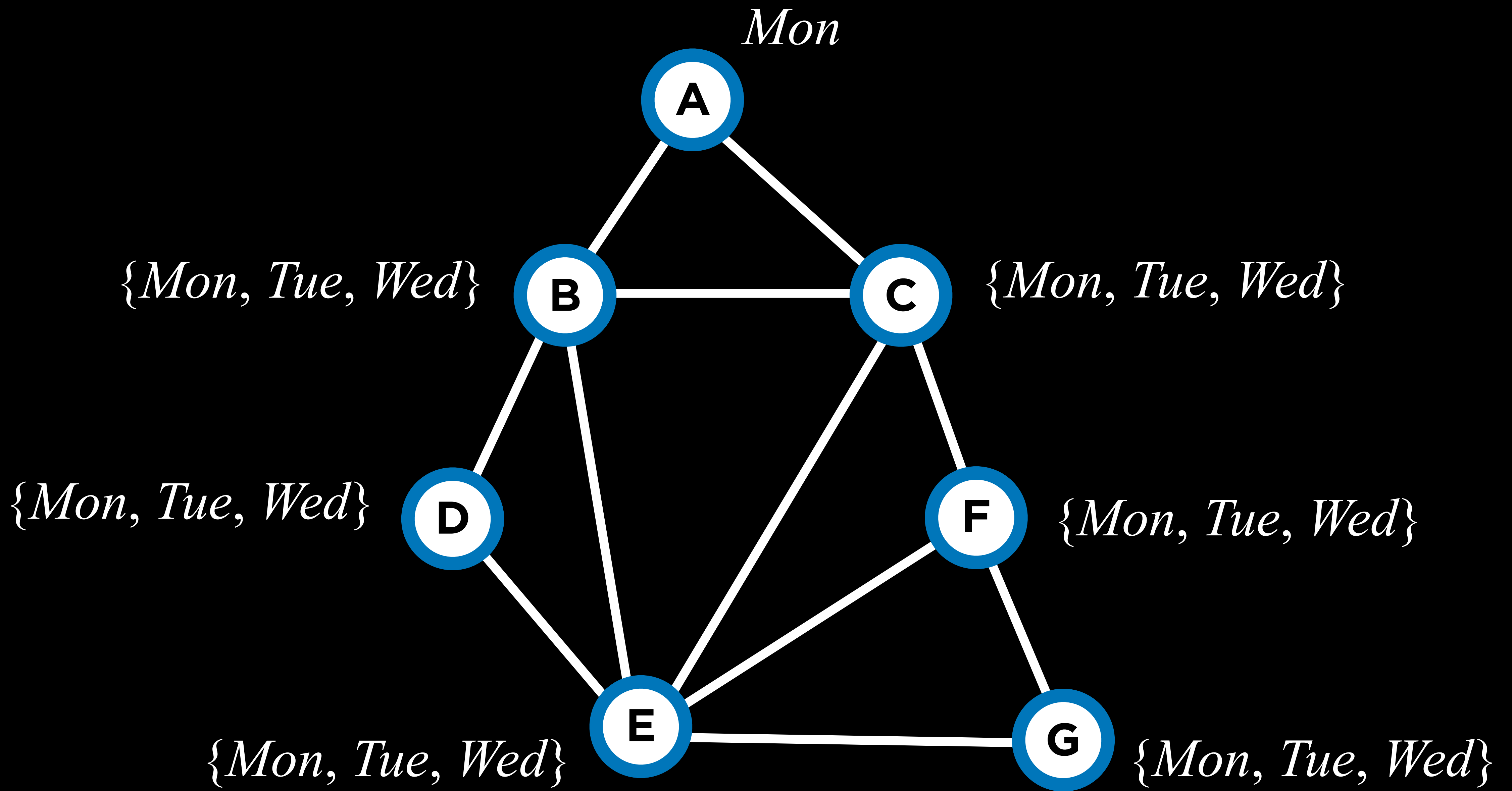
- initial state: empty assignment (no variables)
- actions: add a $\{variable = value\}$ to assignment
- transition model: shows how adding an assignment changes the assignment
- goal test: check if all variables assigned and constraints all satisfied
- path cost function: all paths have same cost

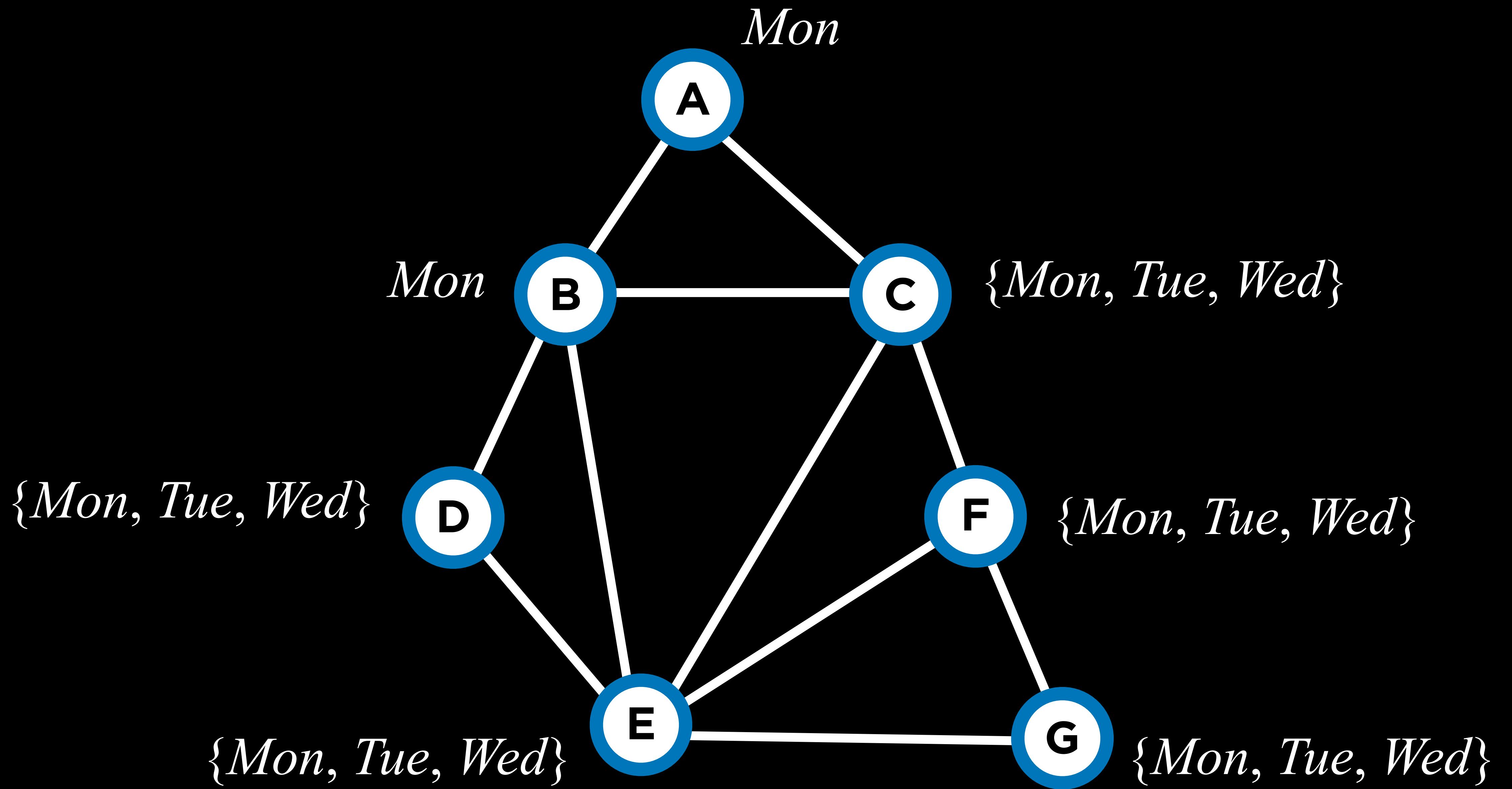
Backtracking Search

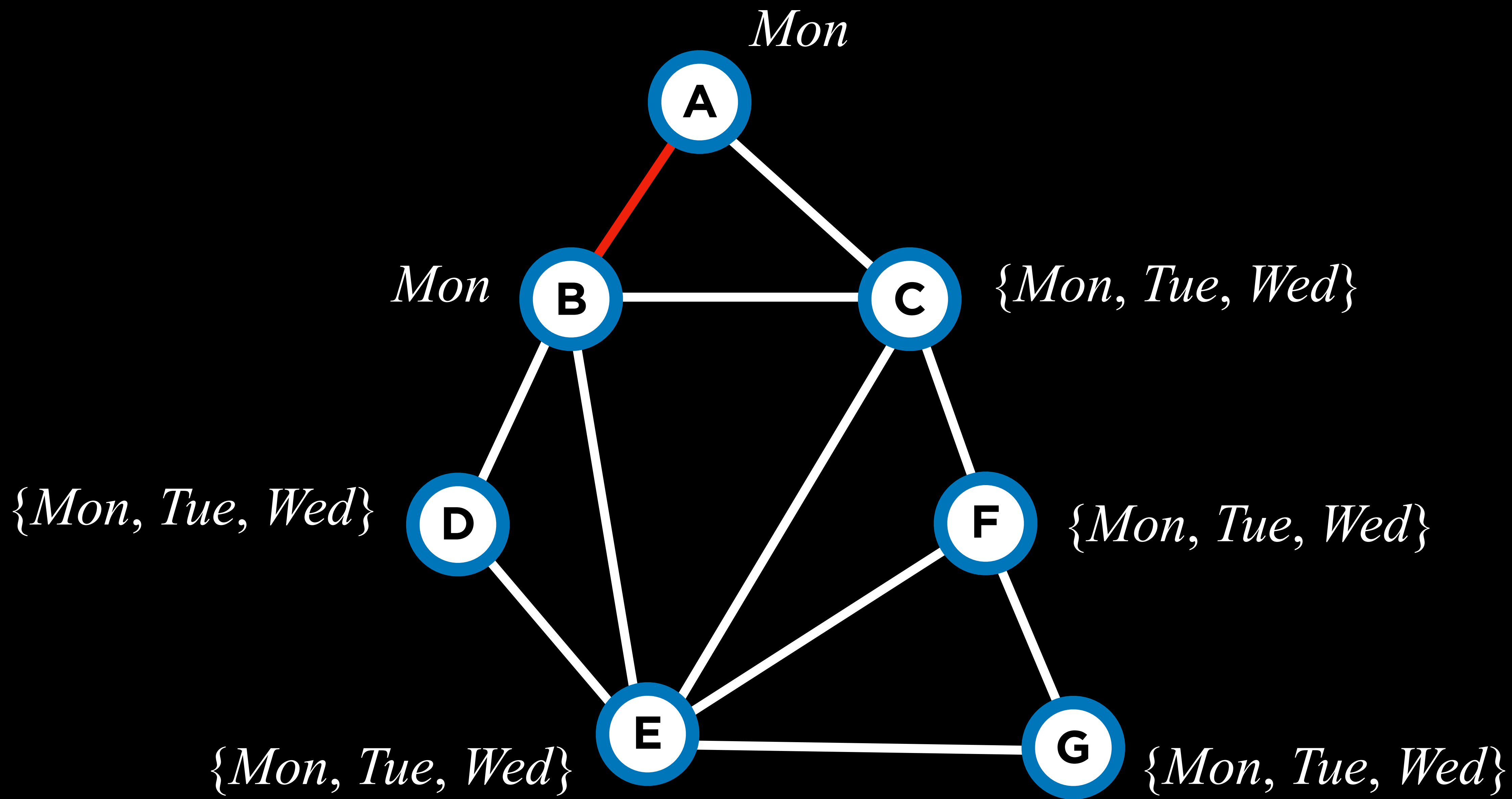
Backtracking Search

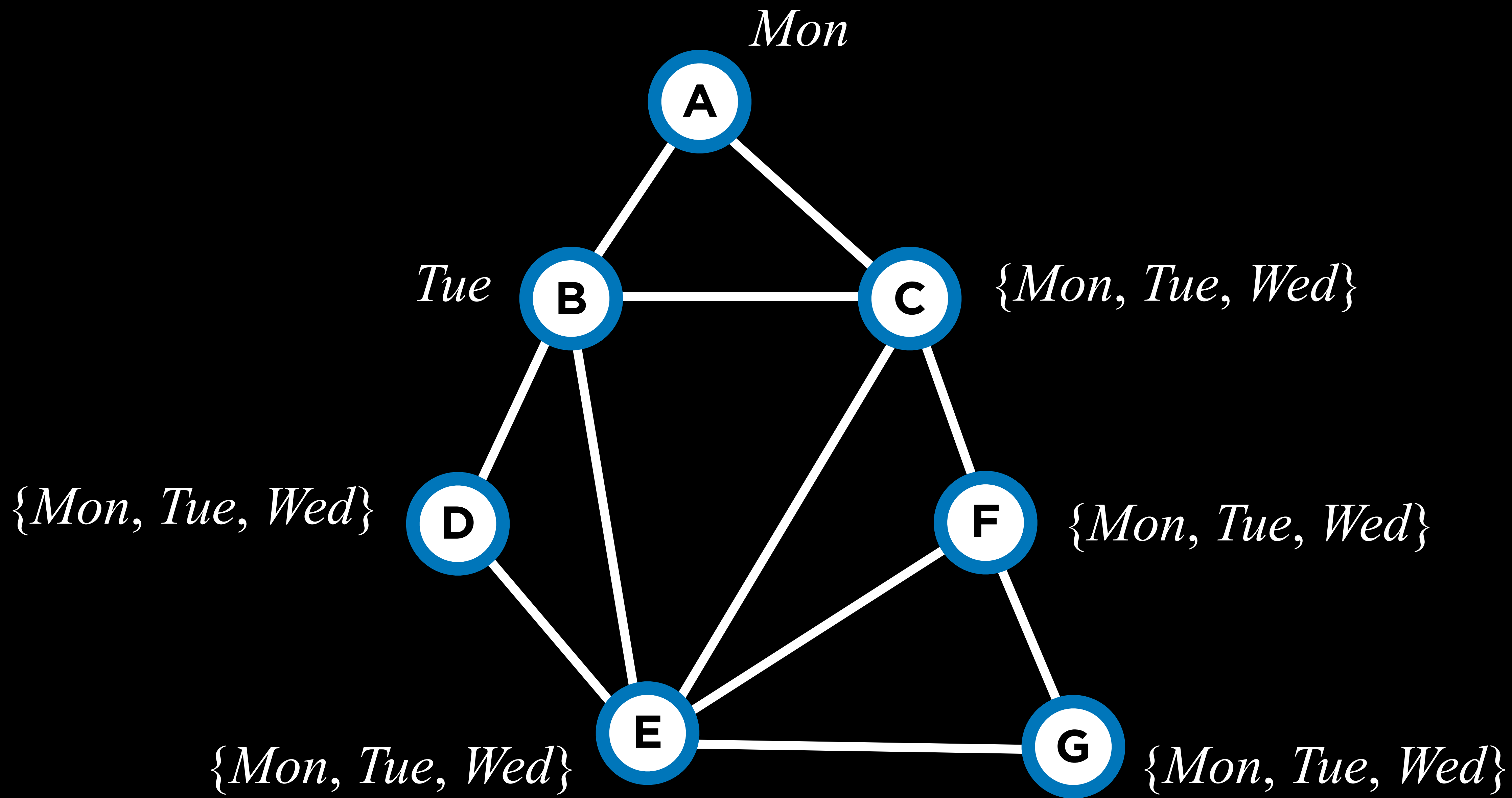
```
function BACKTRACK(assignment, csp):  
    if assignment complete: return assignment  
    var = SELECT-UNASSIGNED-VAR(assignment, csp)  
    for value in DOMAIN-VALUES(var, assignment, csp):  
        if value consistent with assignment:  
            add {var = value} to assignment  
            result = BACKTRACK(assignment, csp)  
            if result ≠ failure: return result  
        remove {var = value} from assignment  
    return failure
```

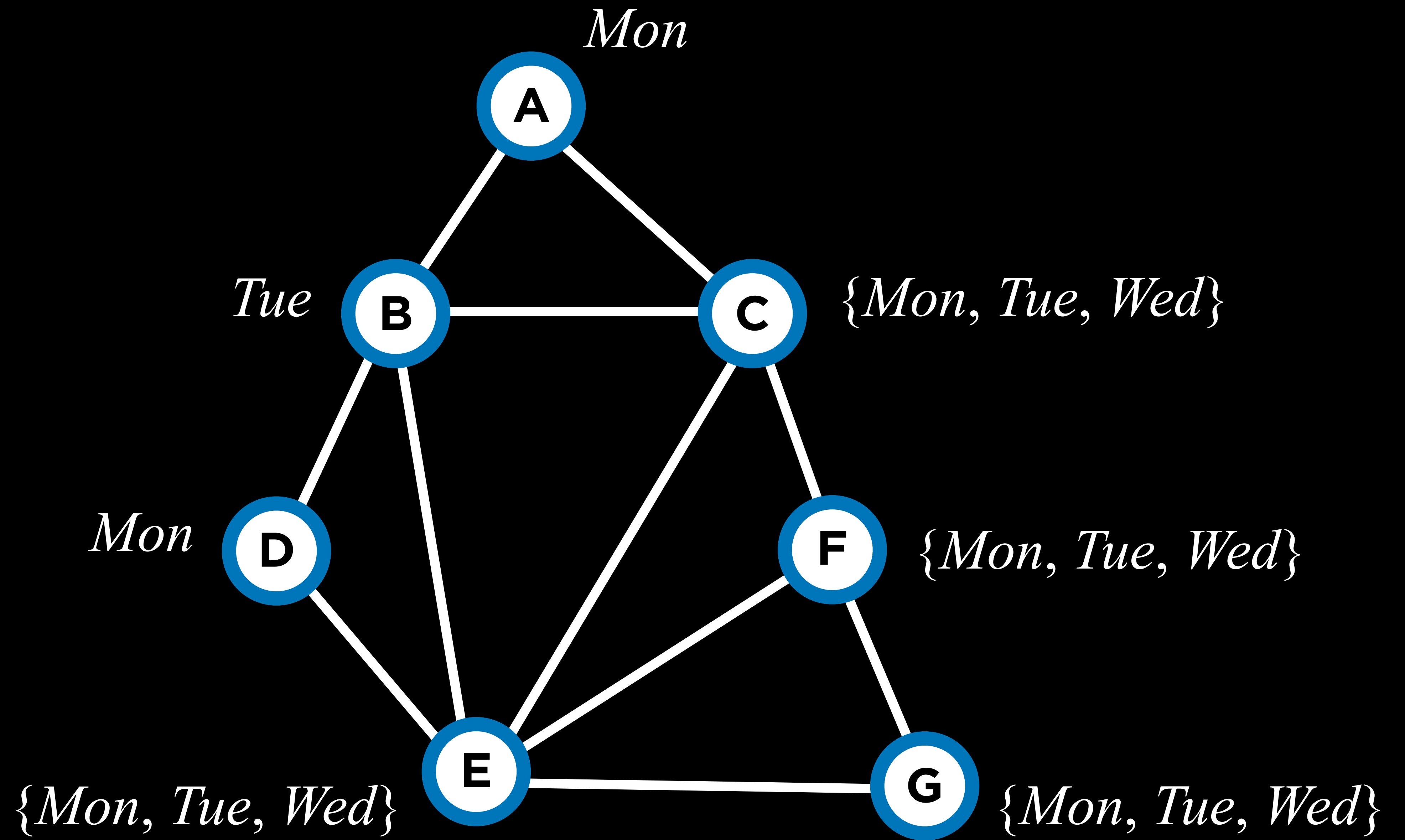


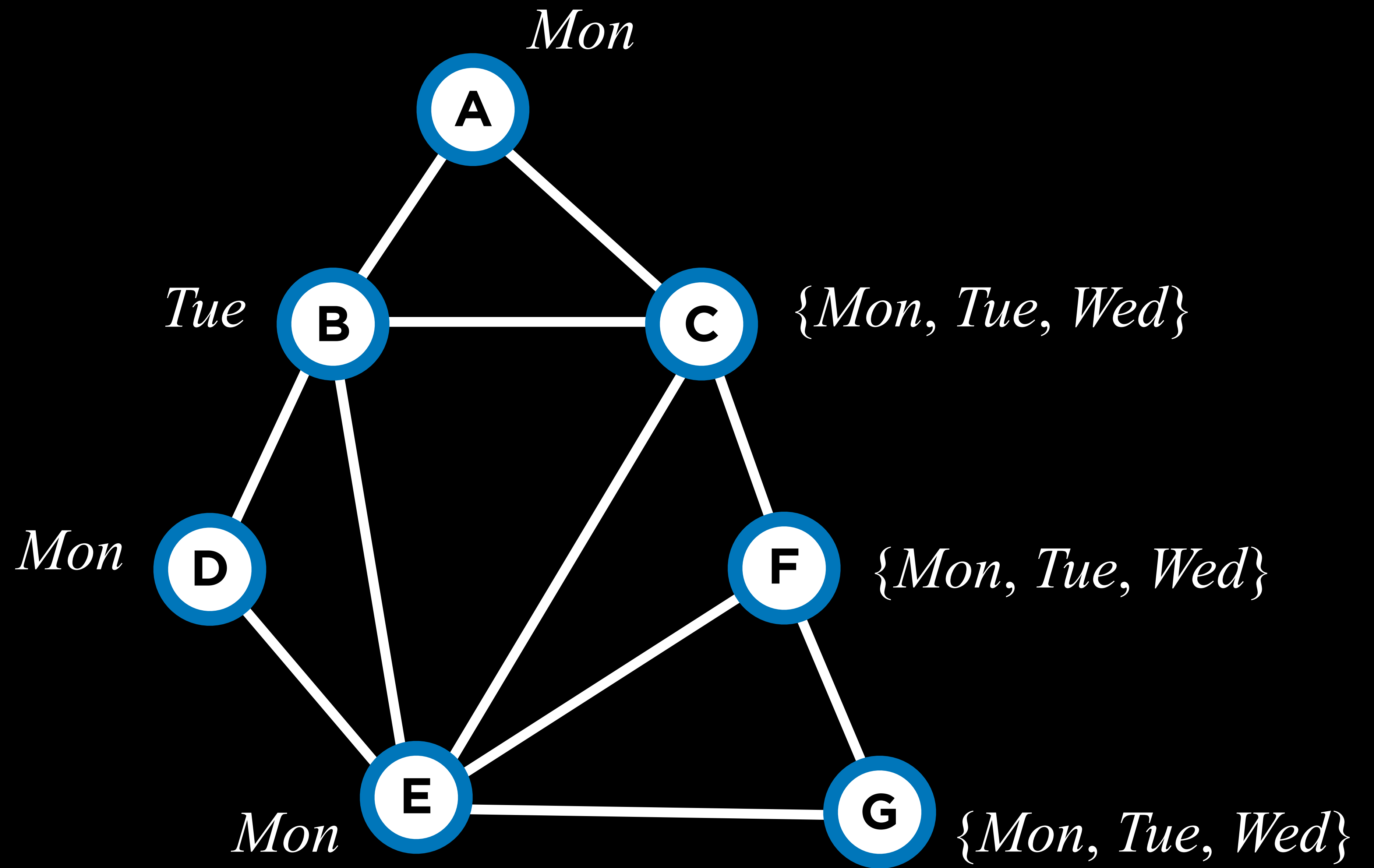


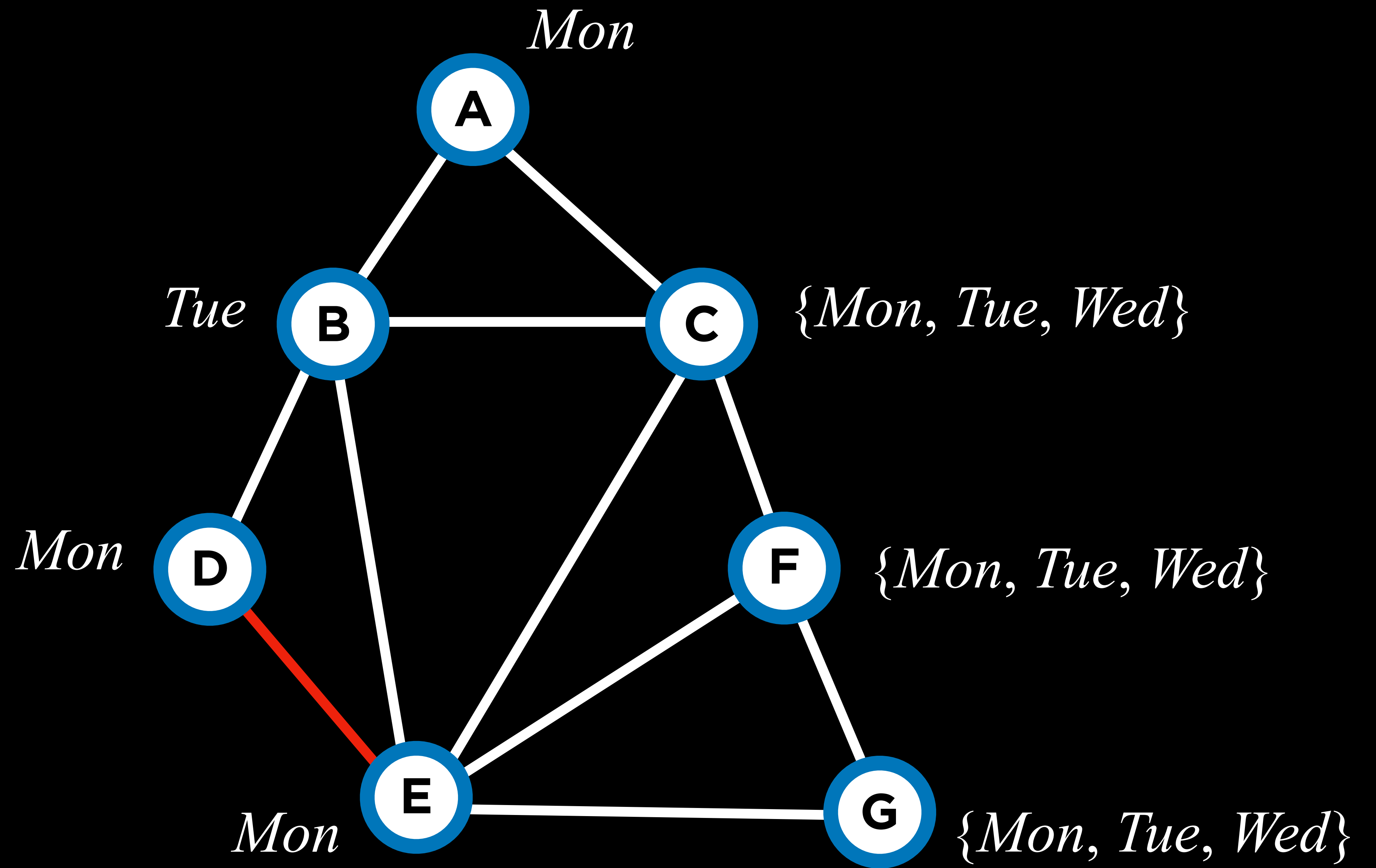


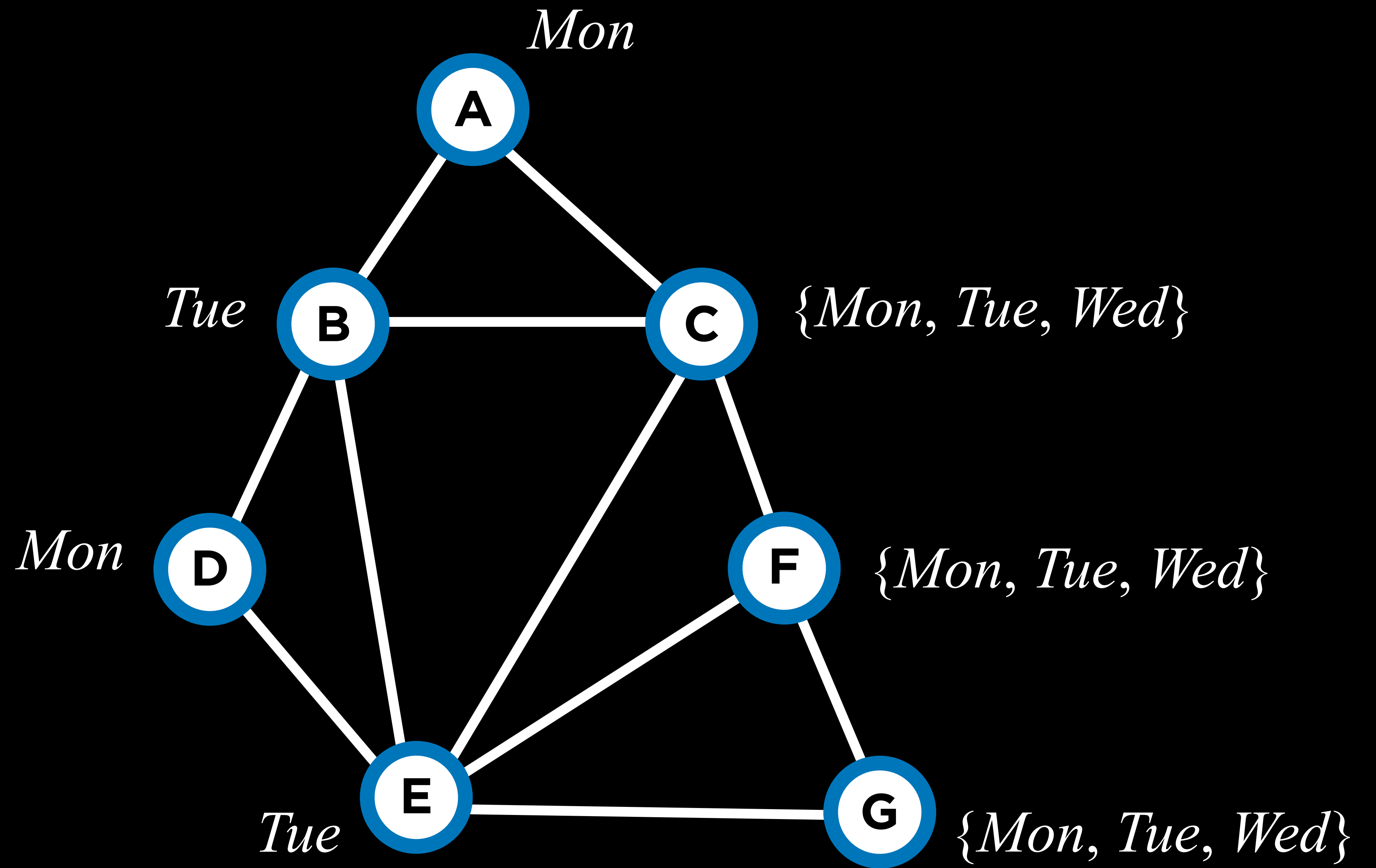


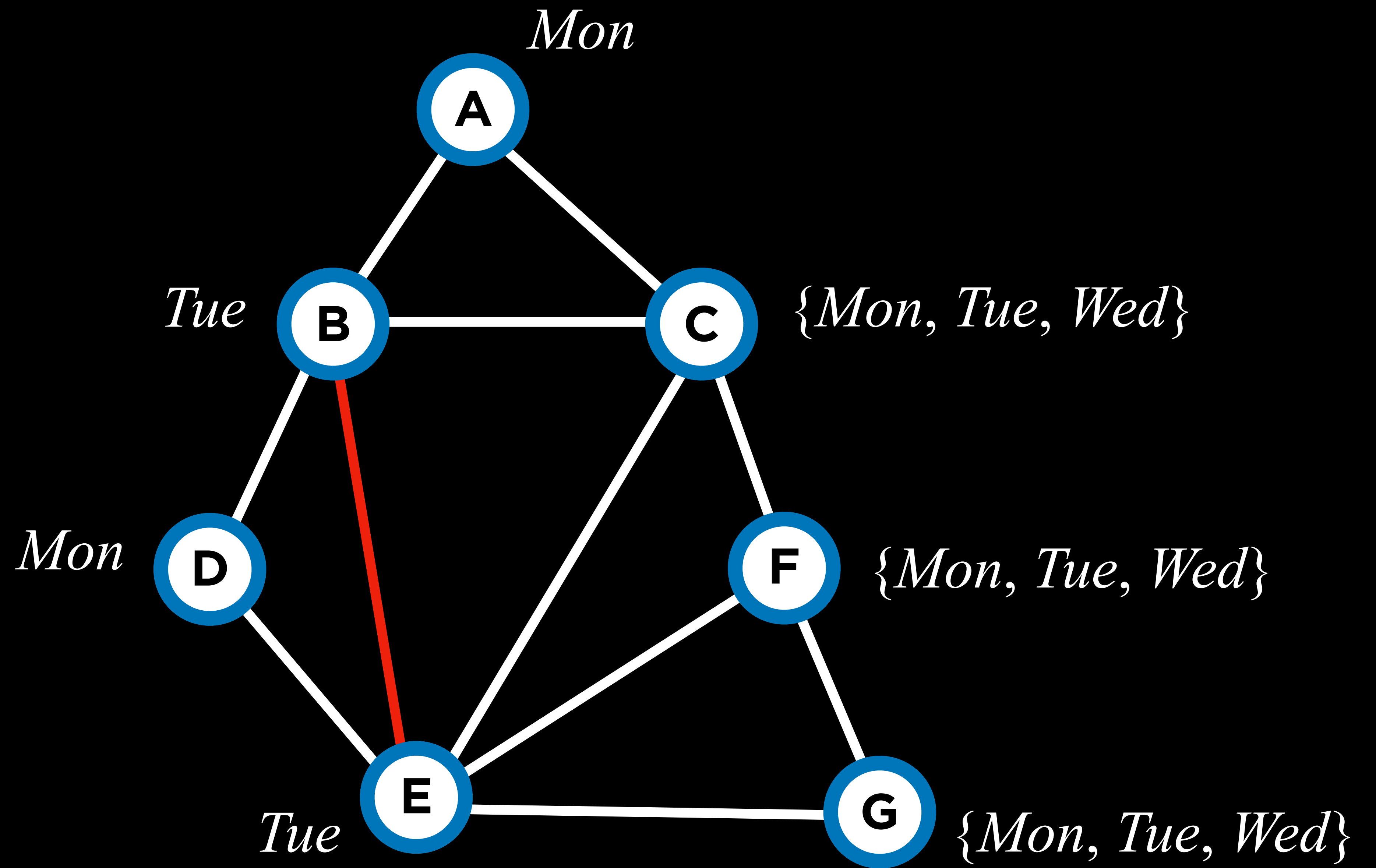


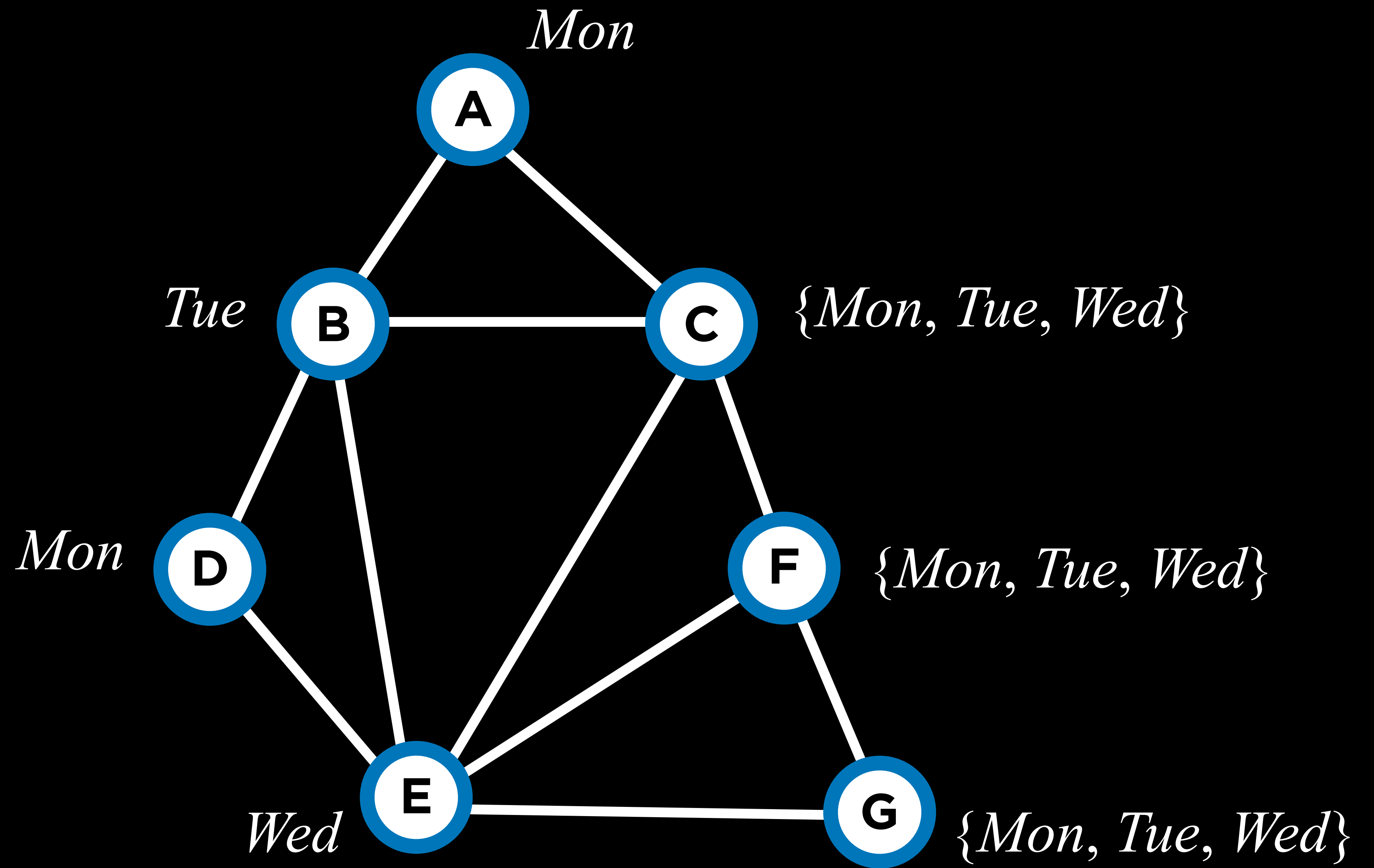


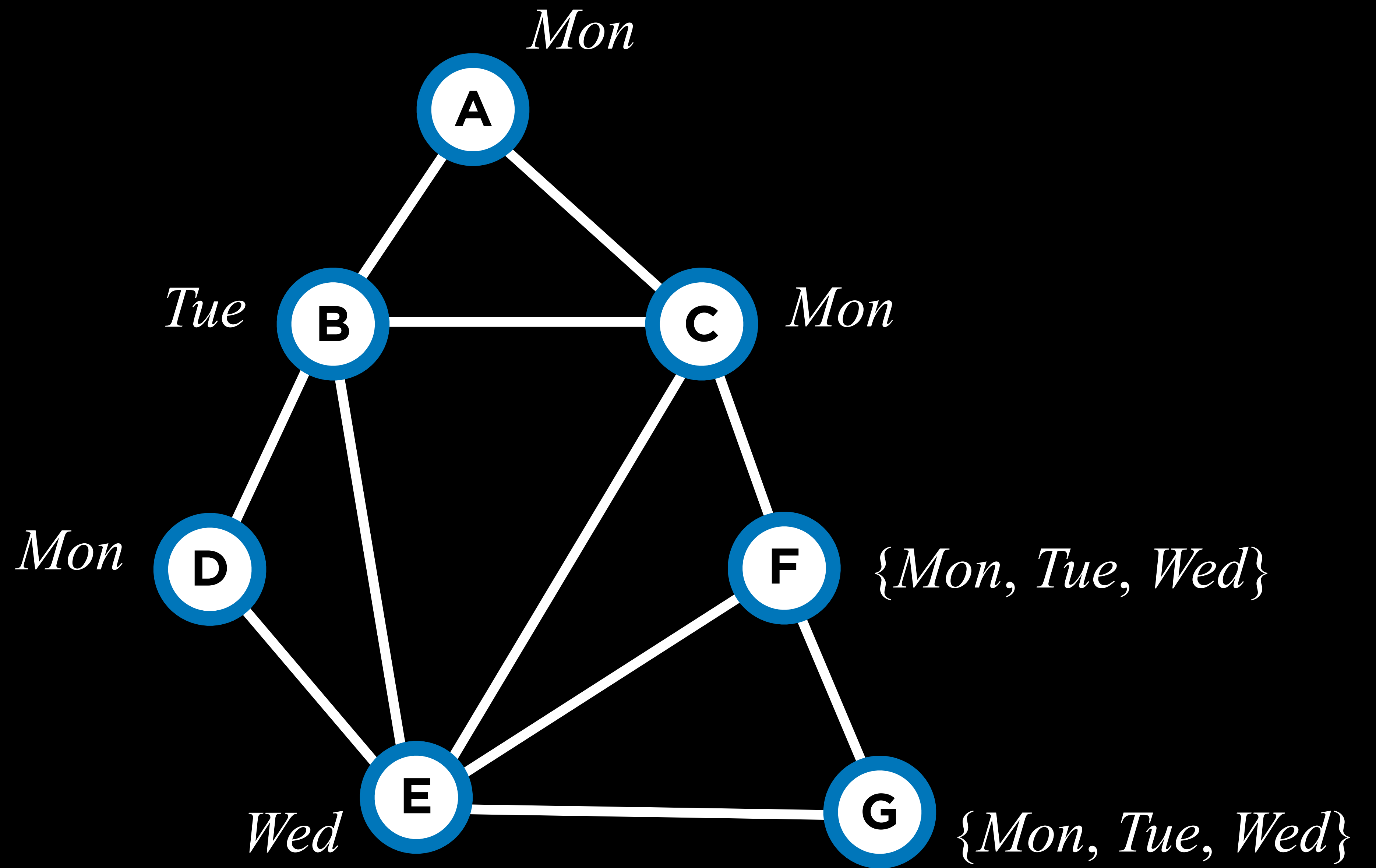


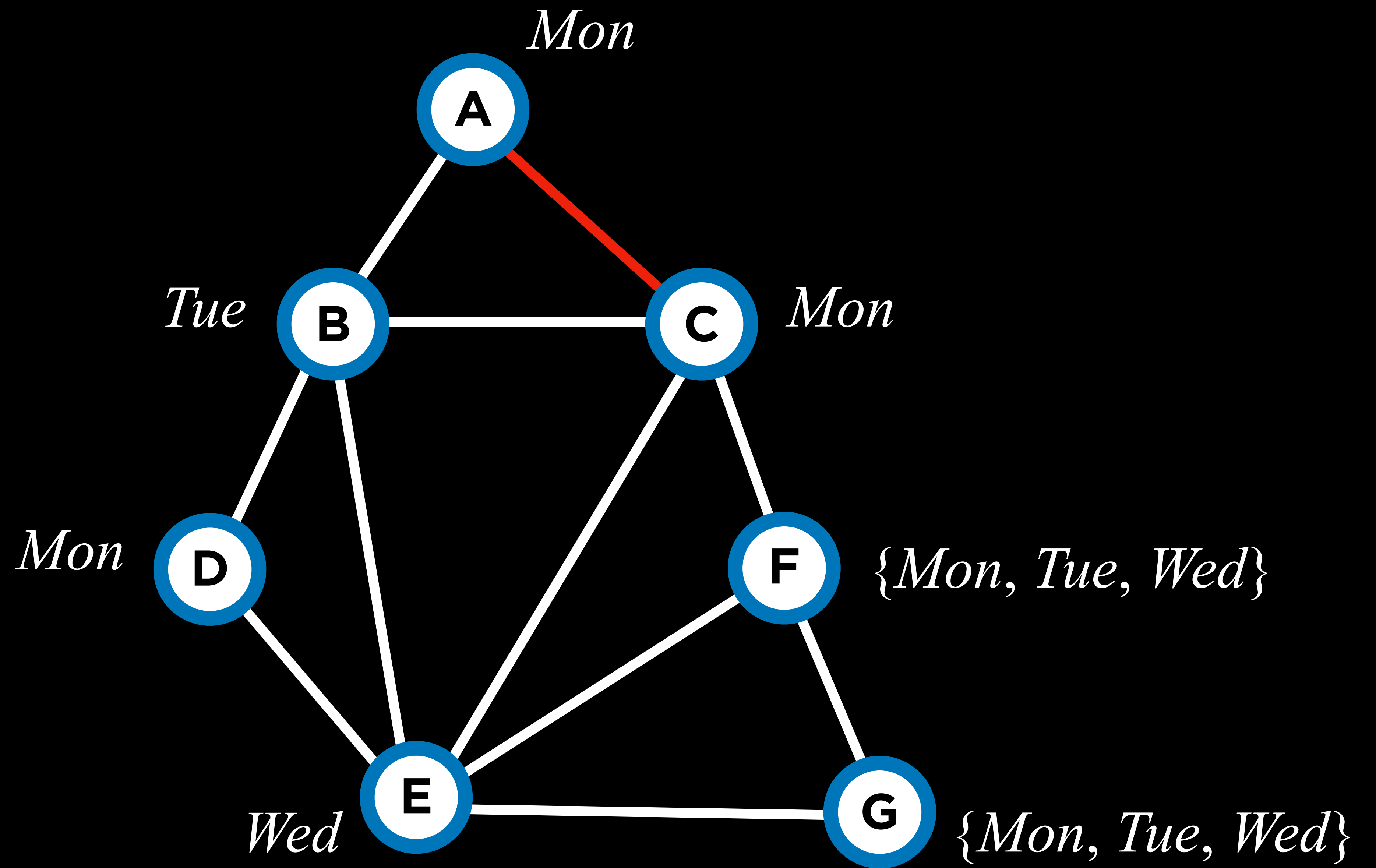


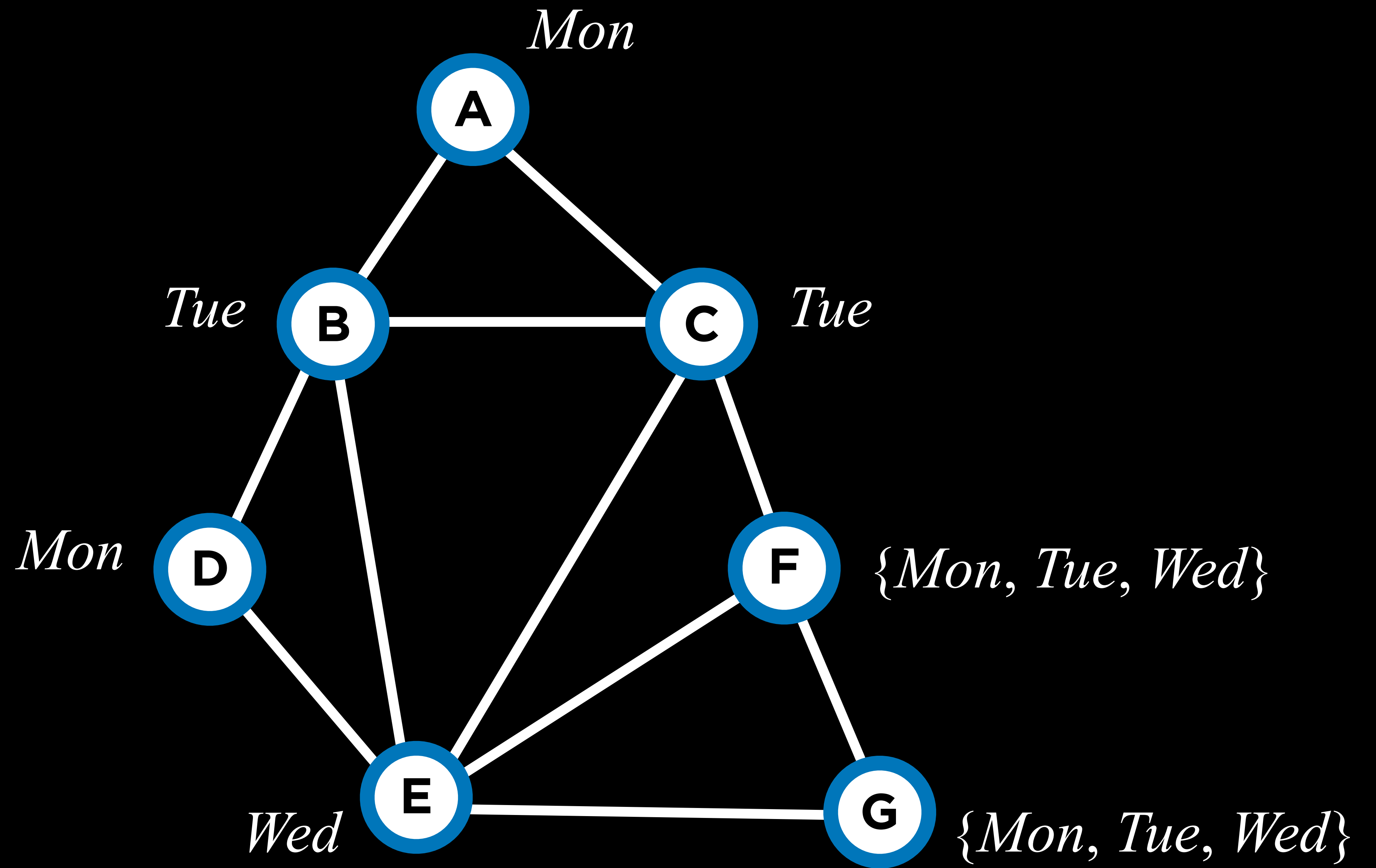


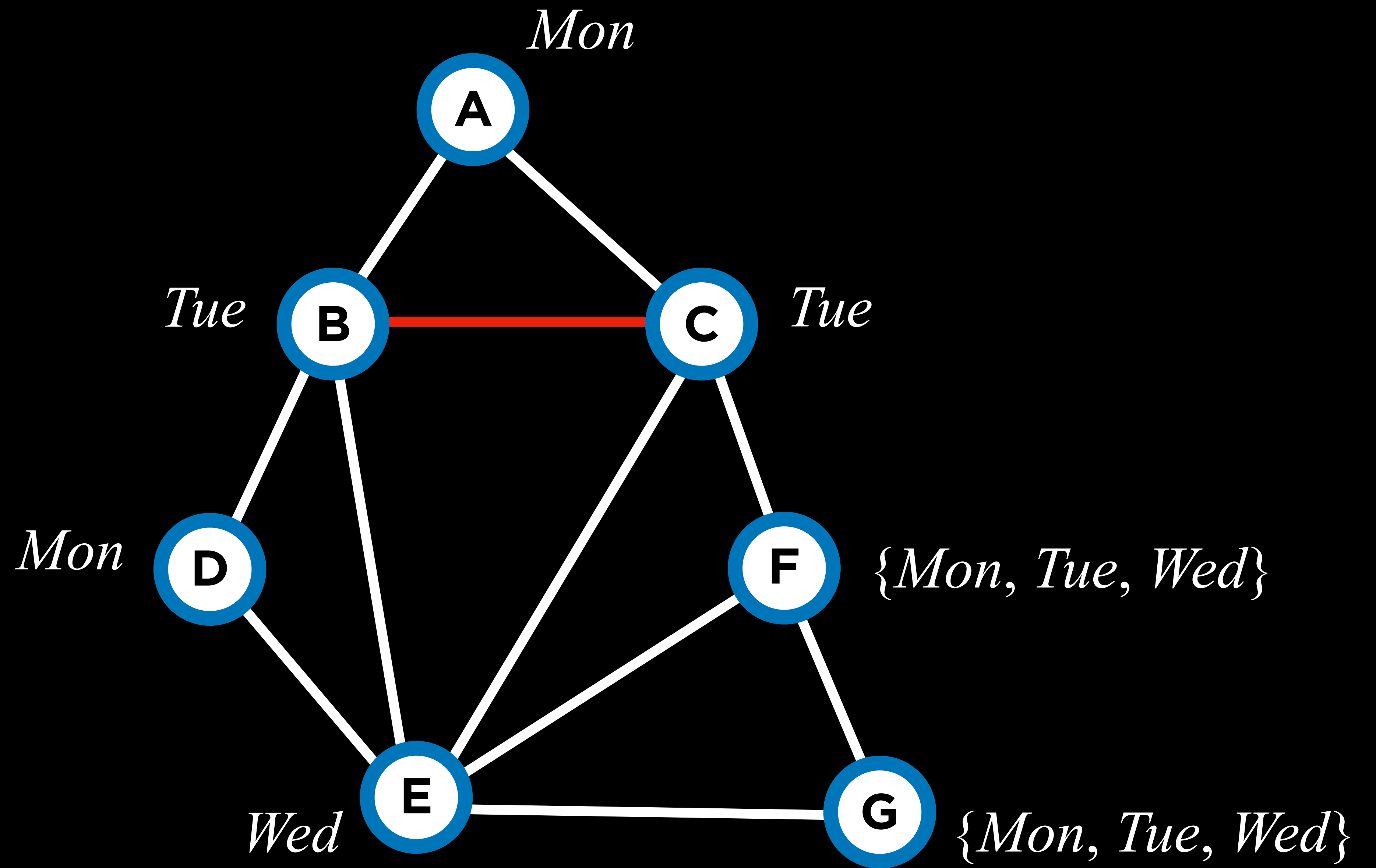


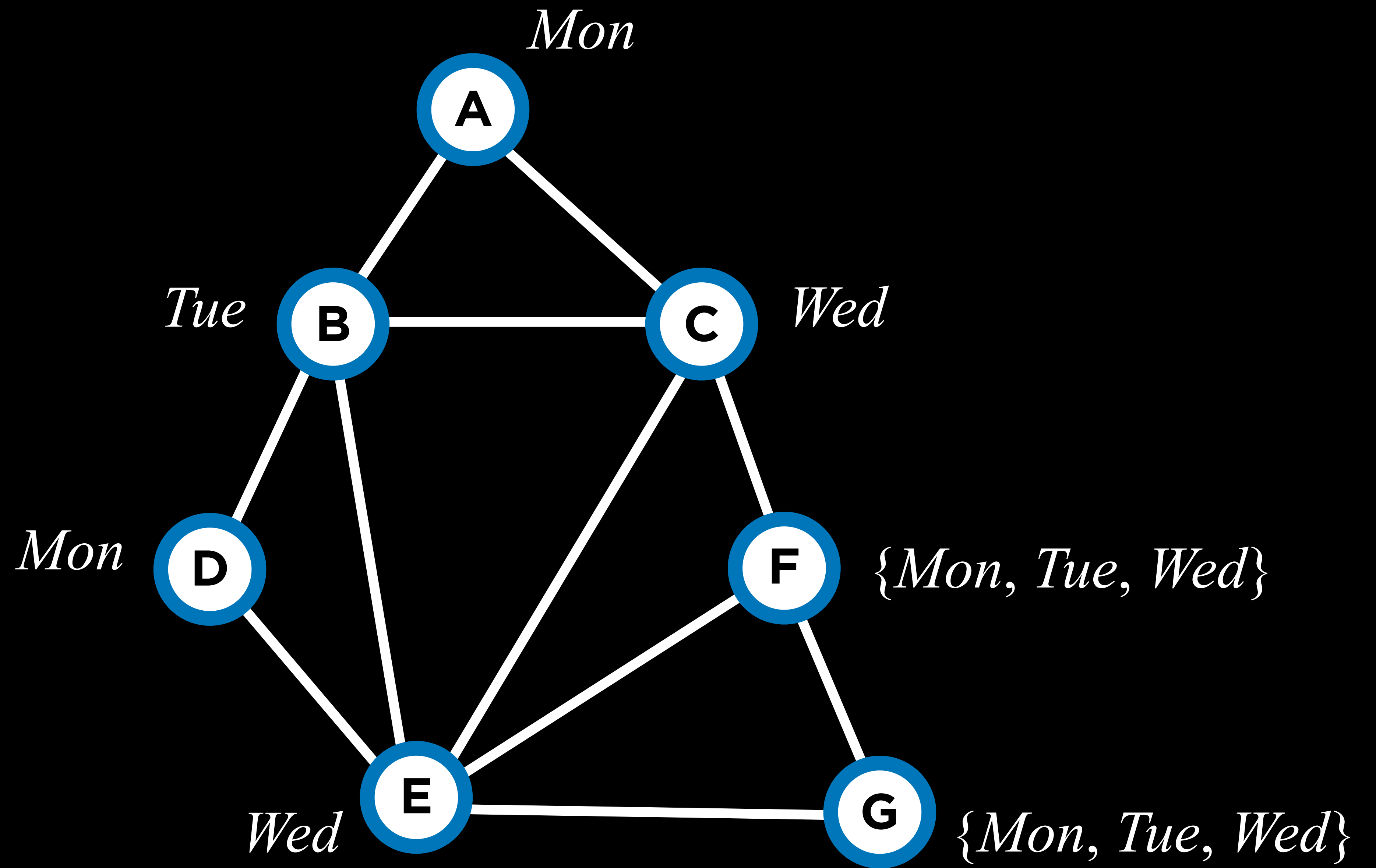


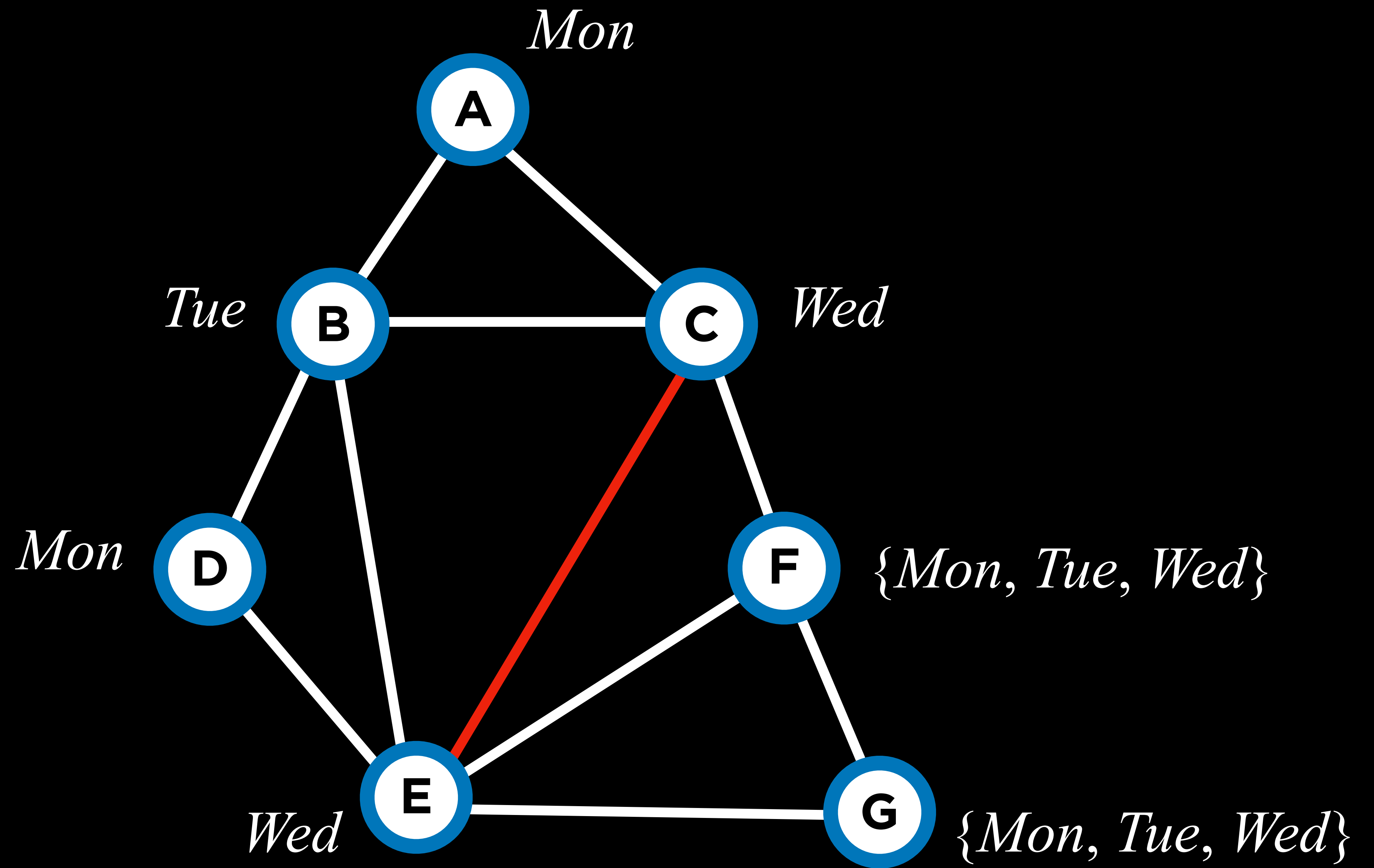


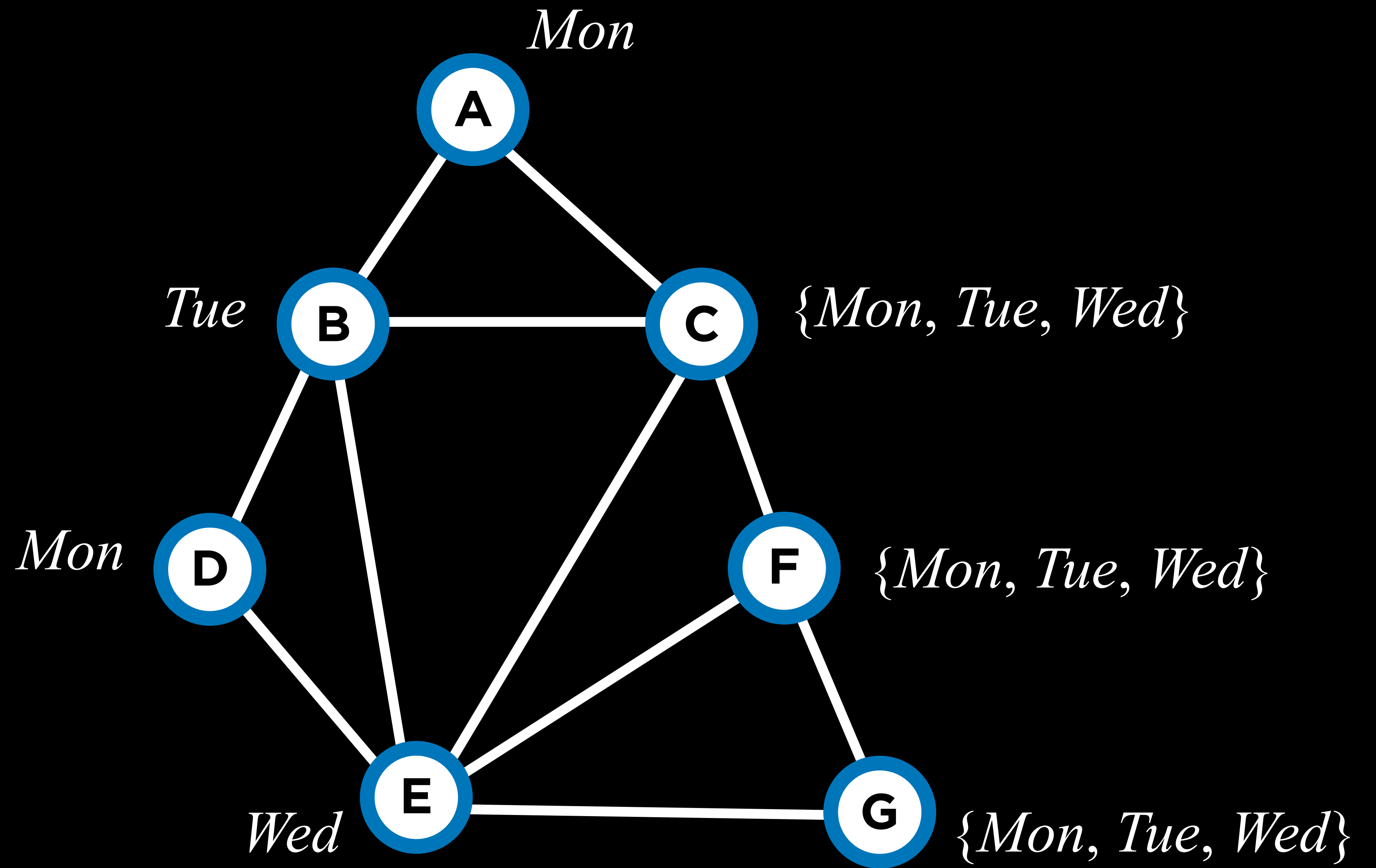


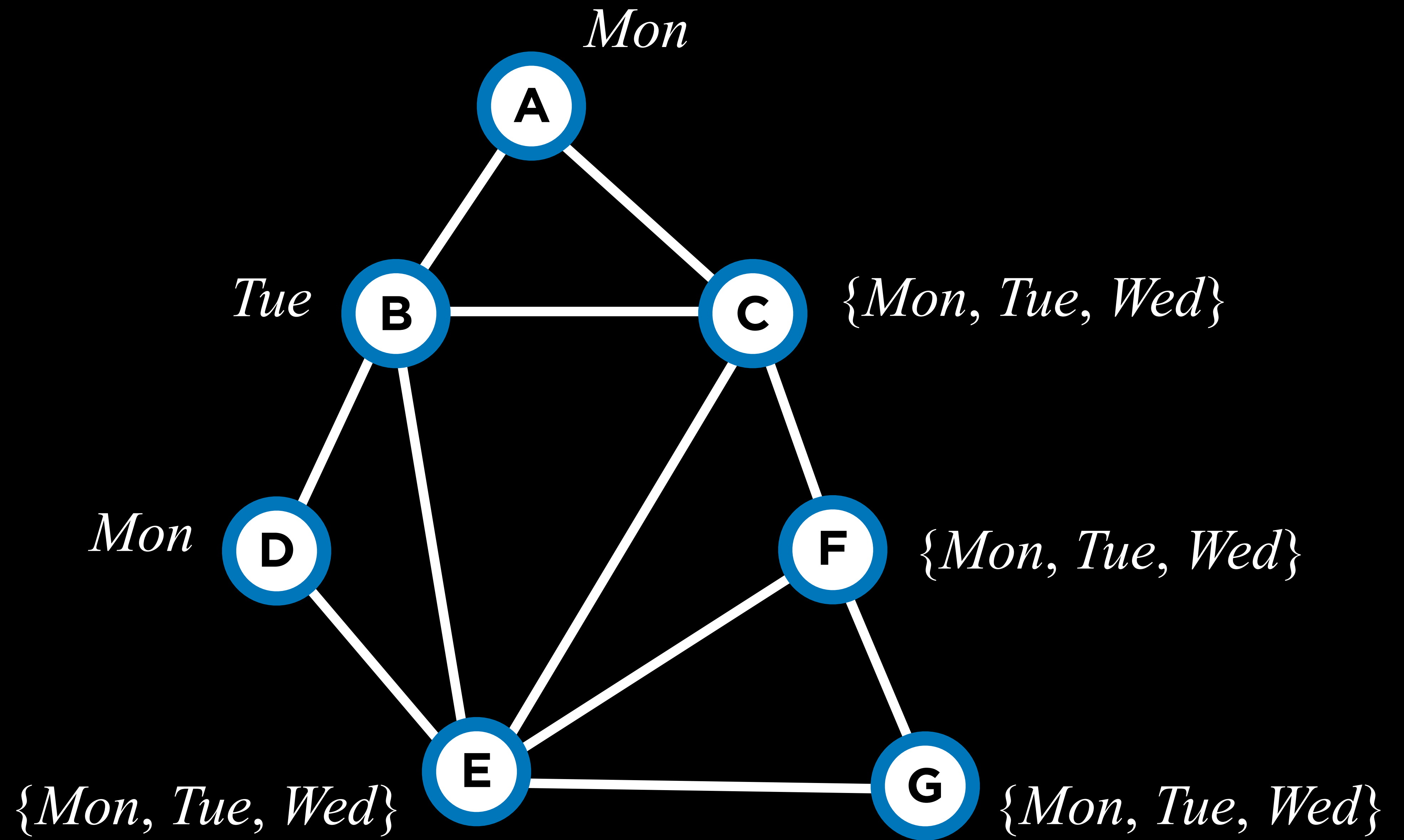


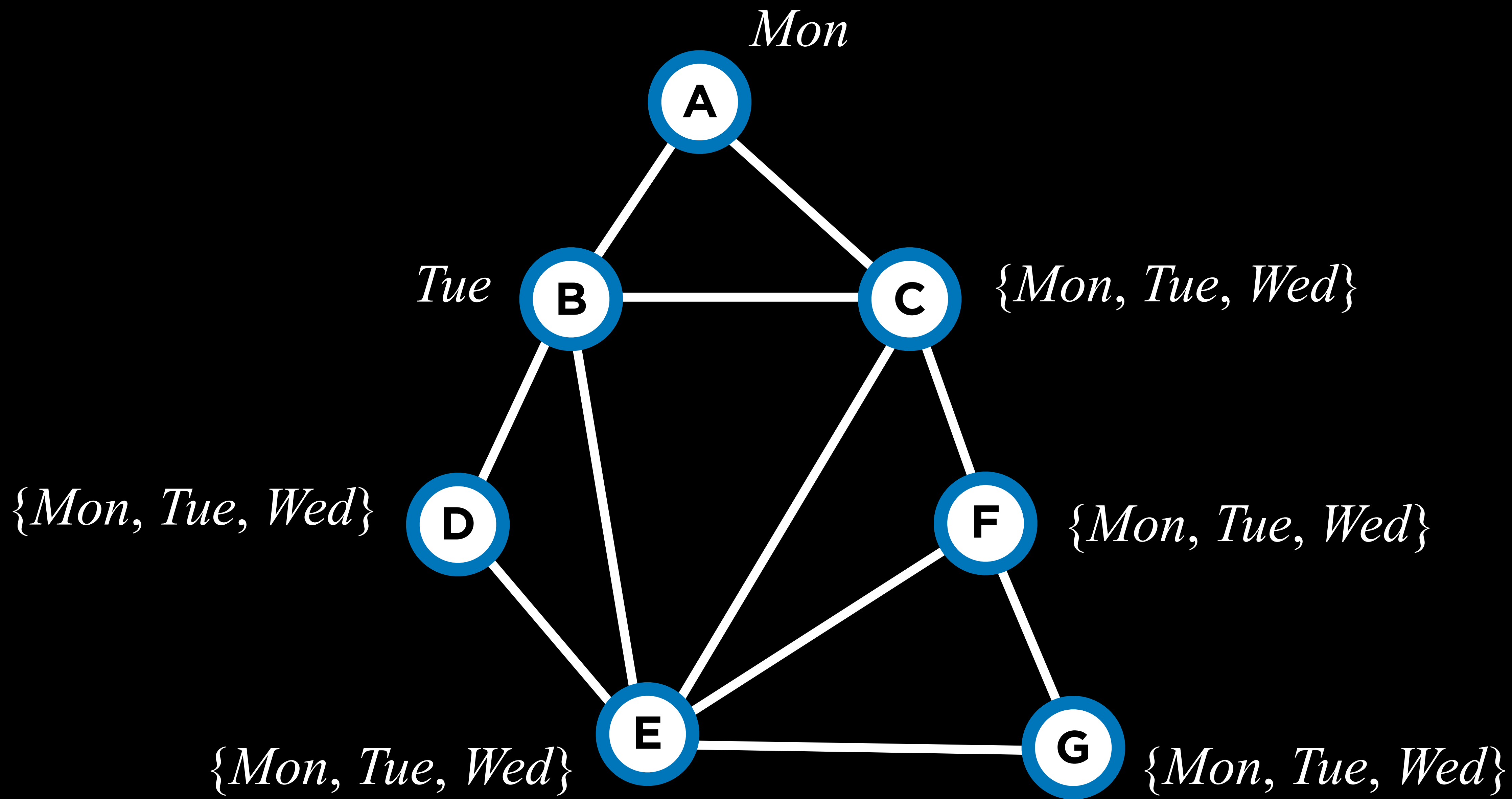


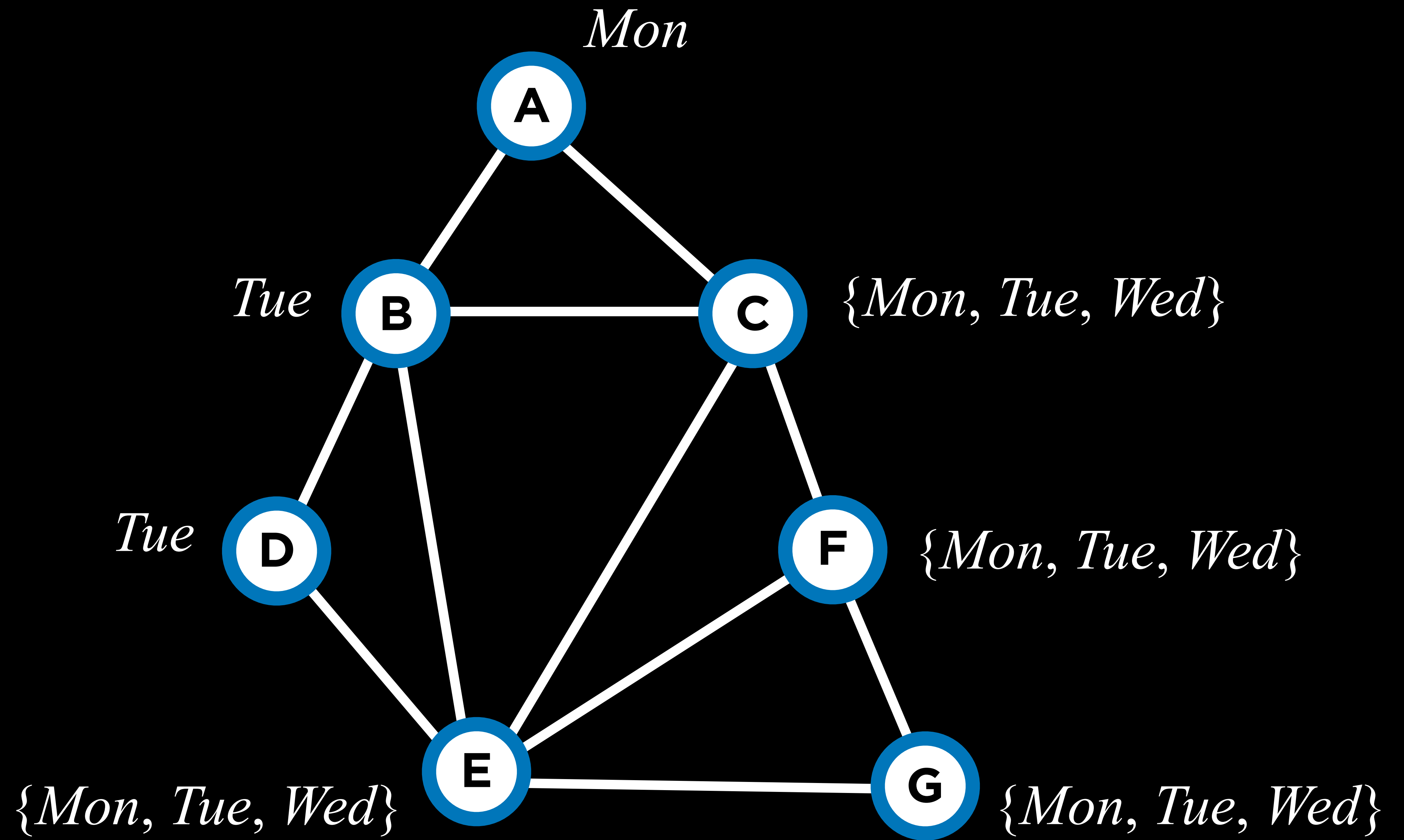


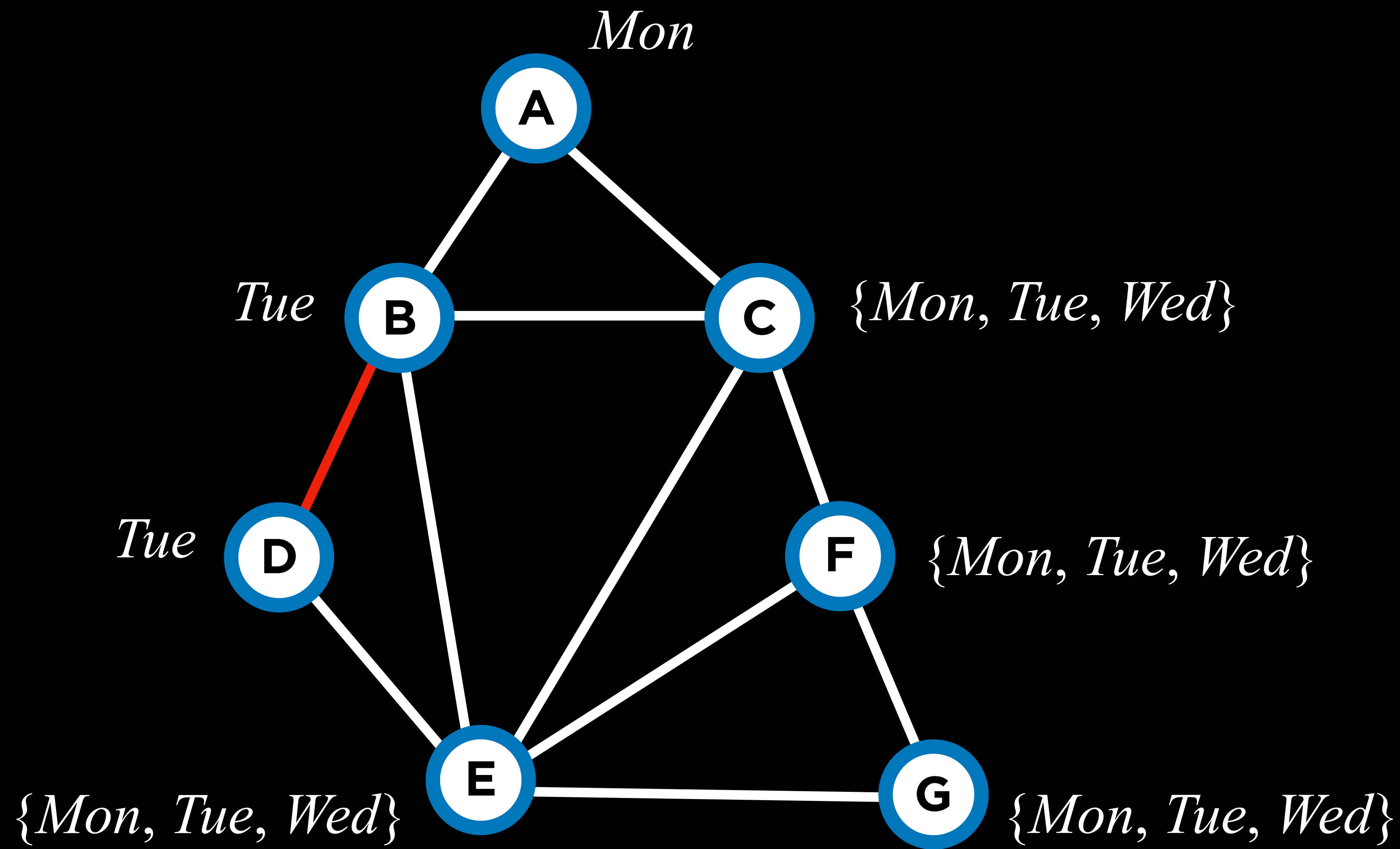


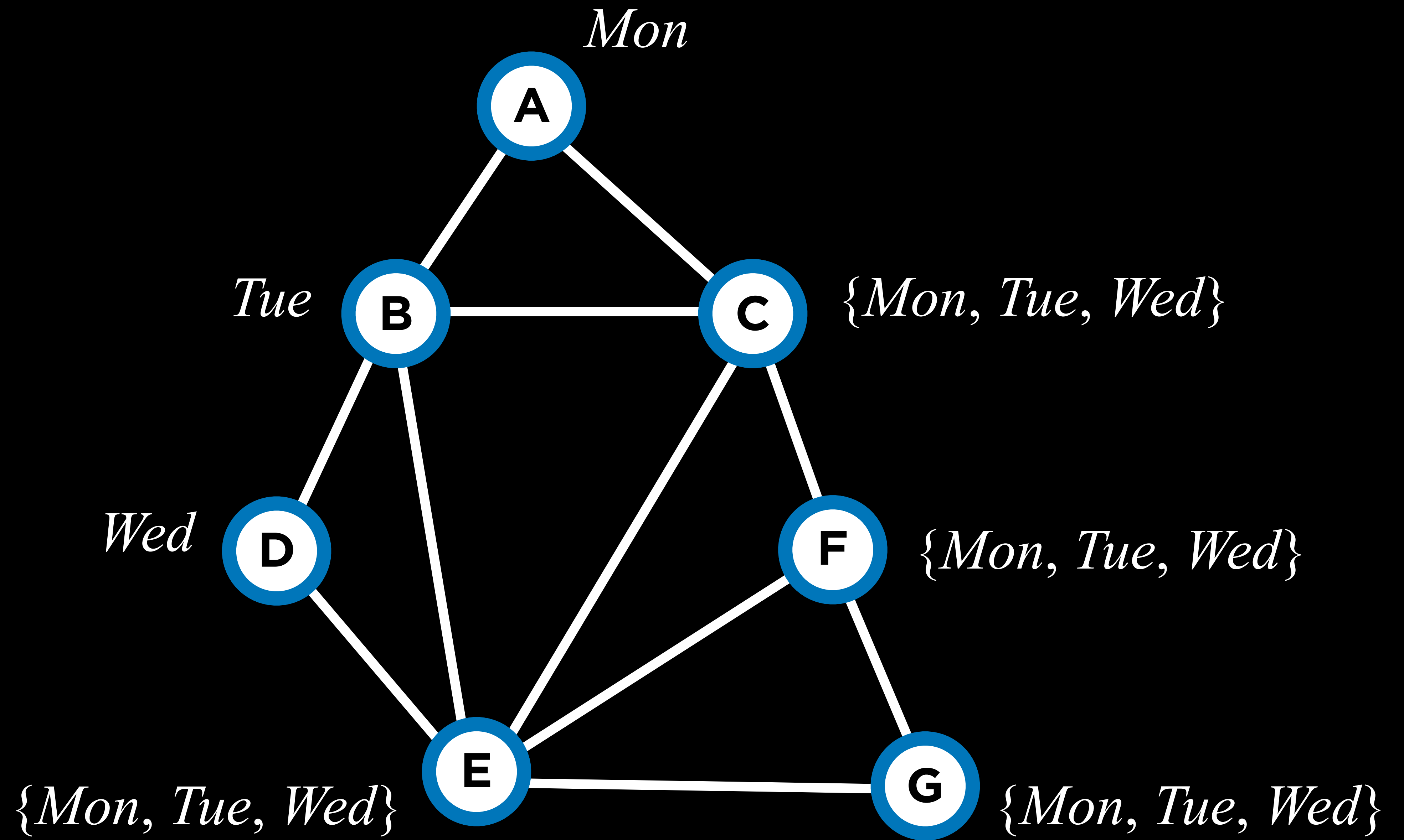


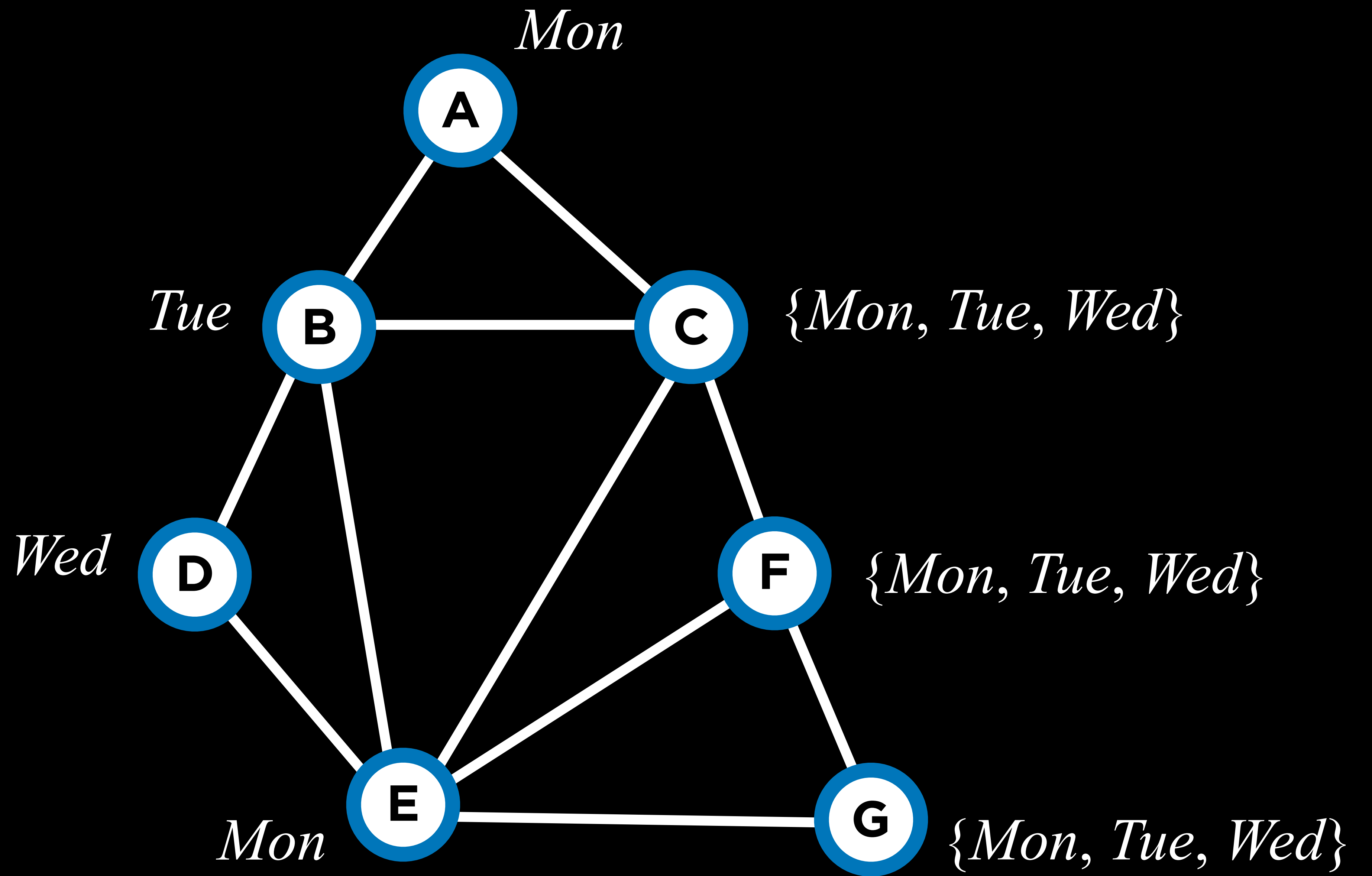


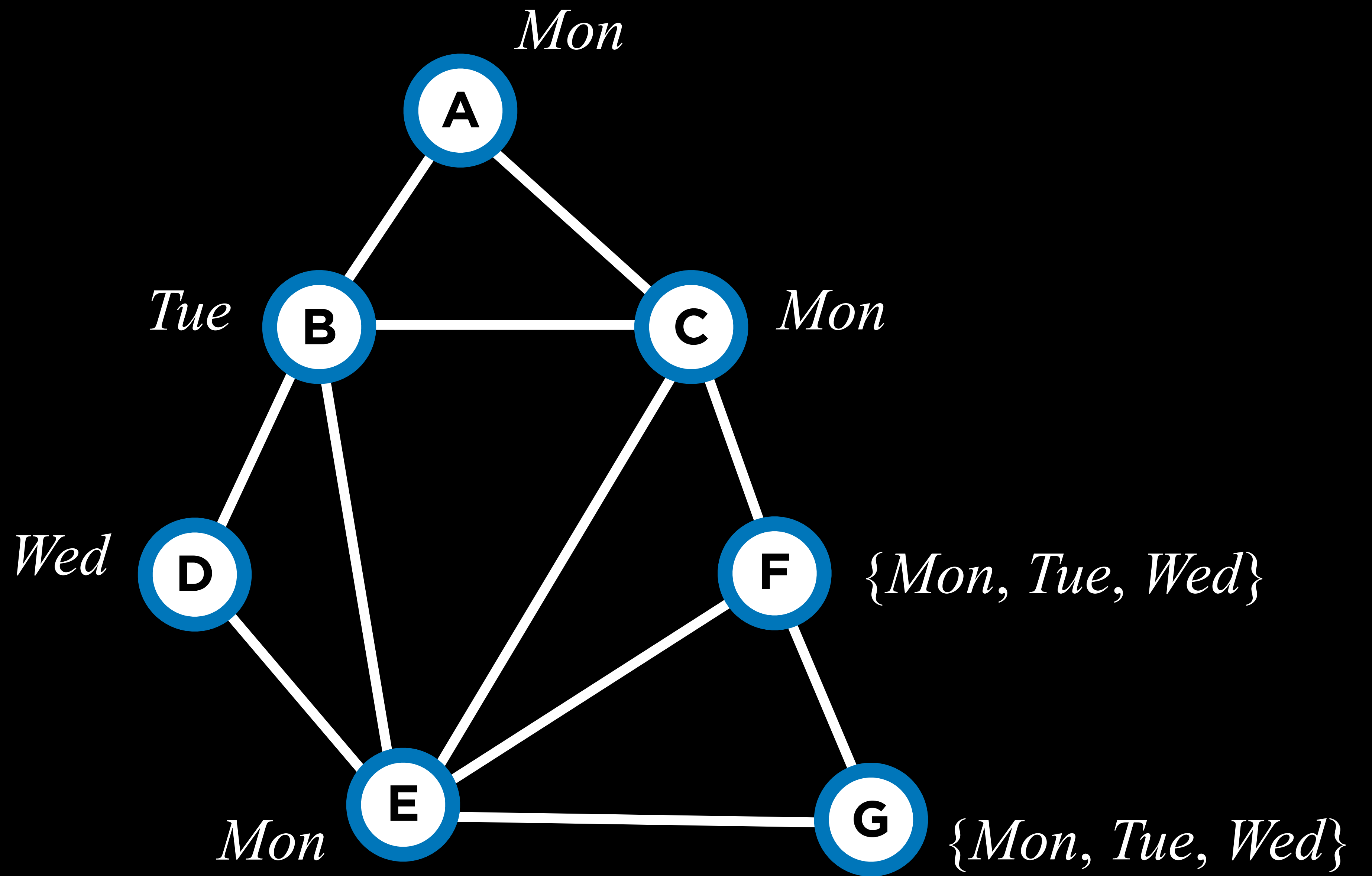


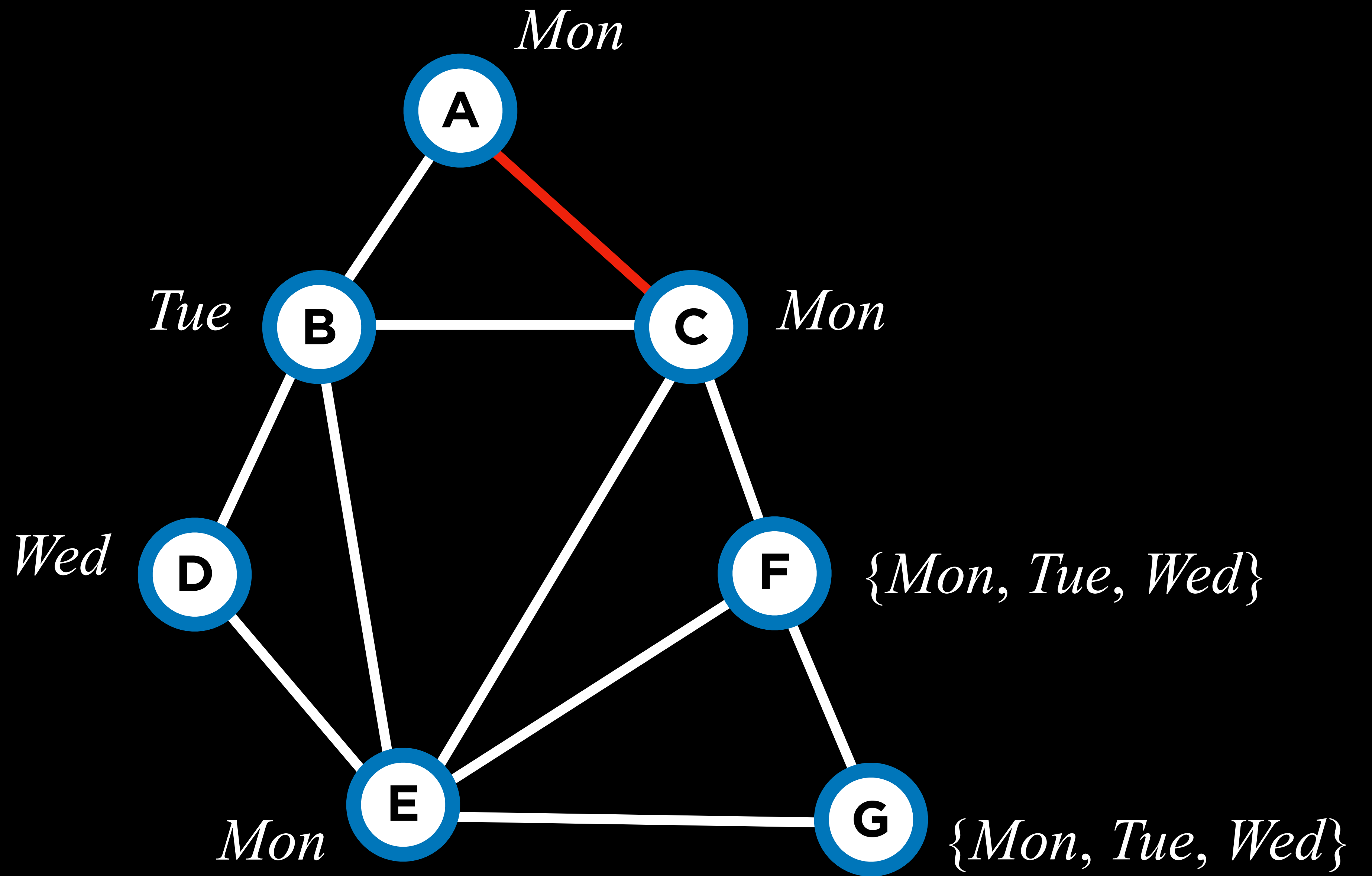


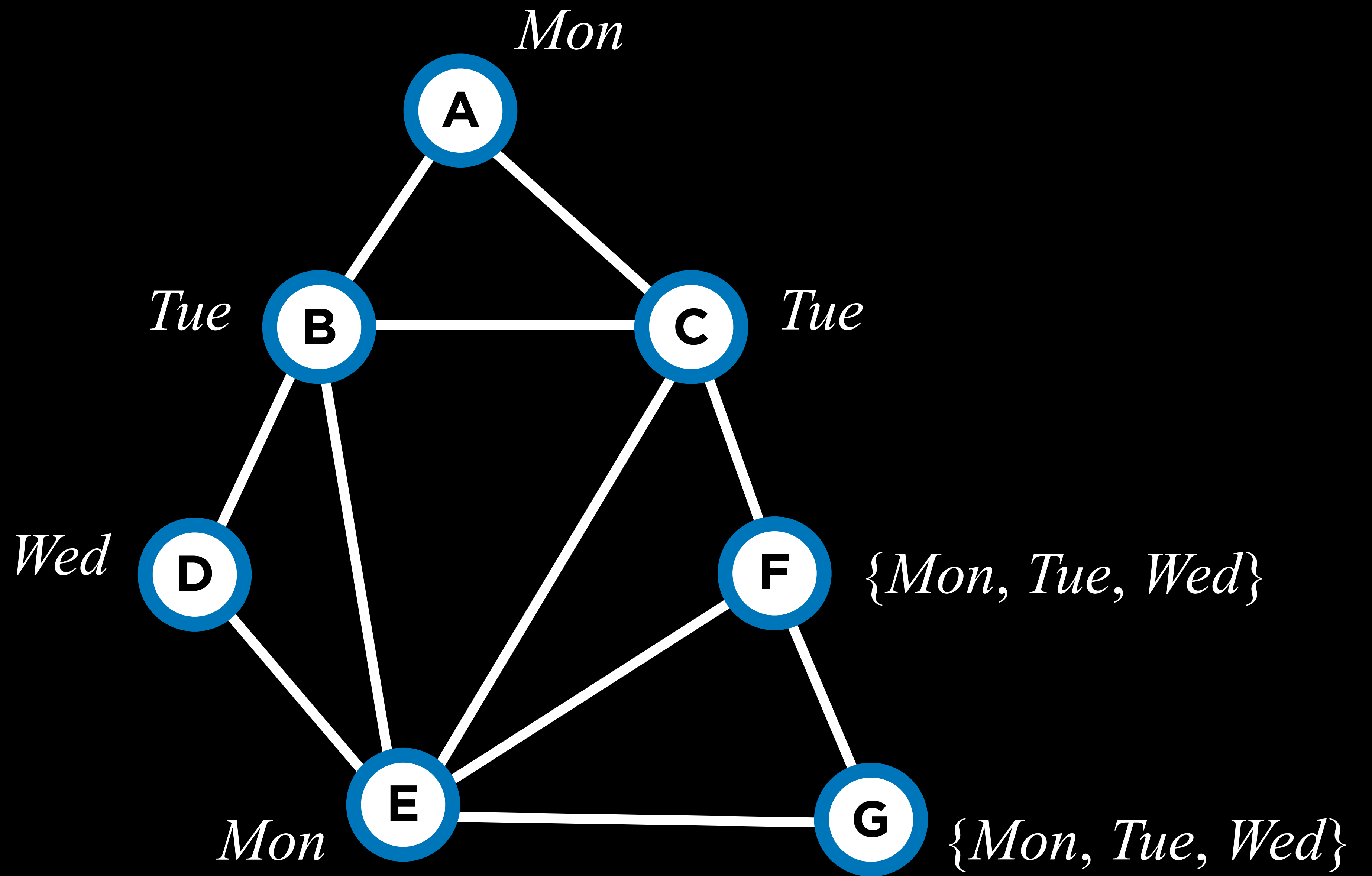


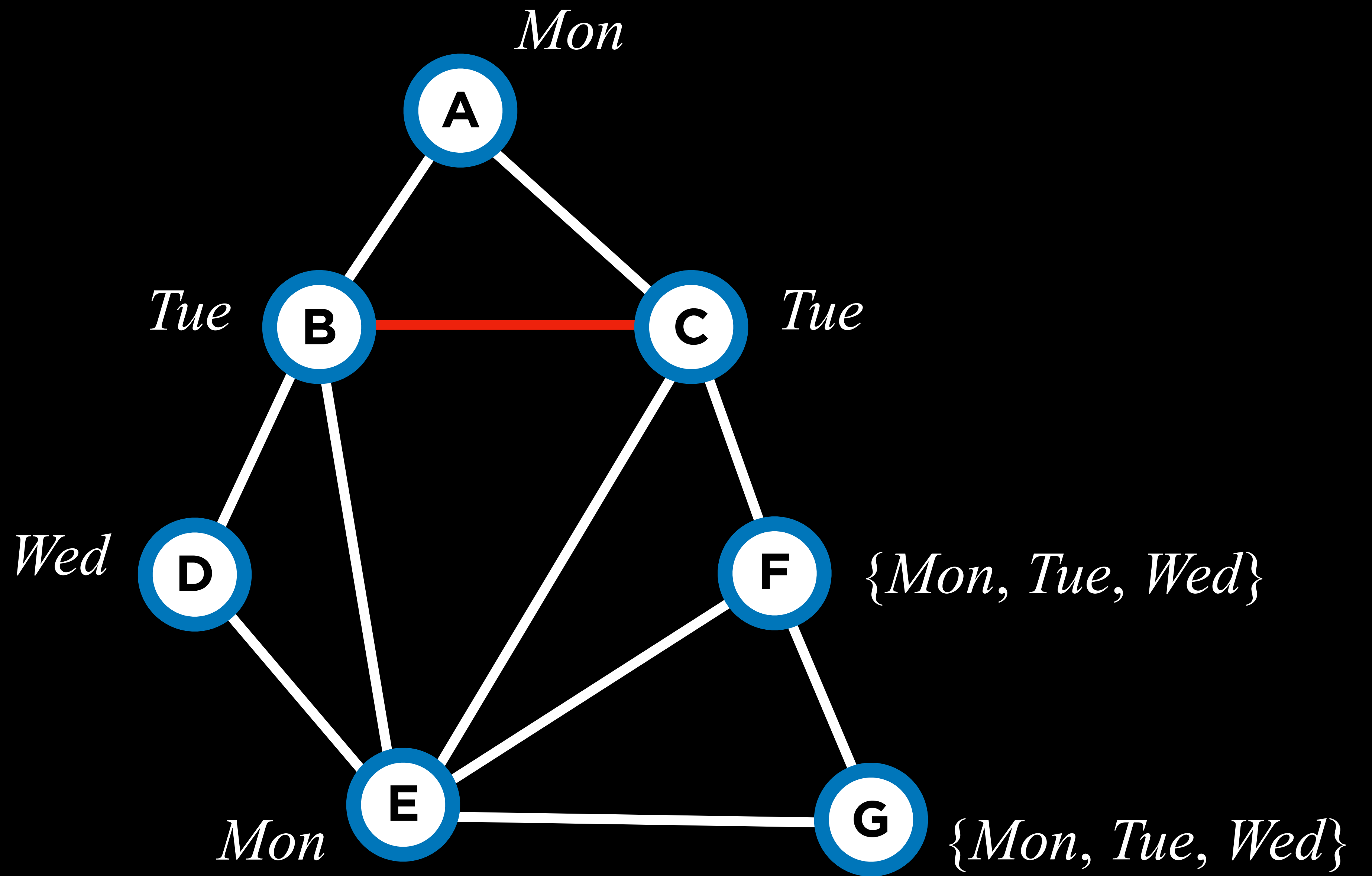


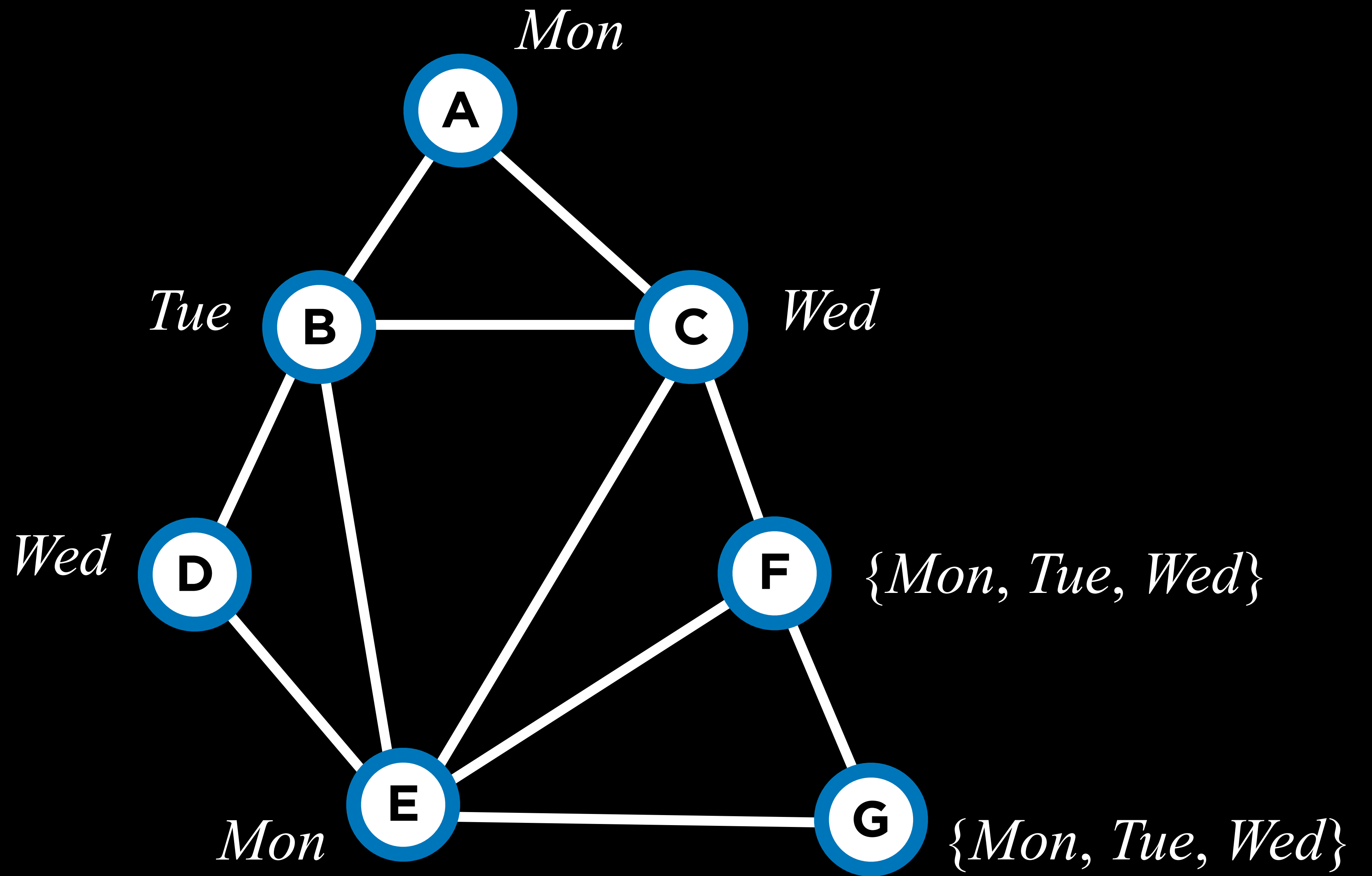


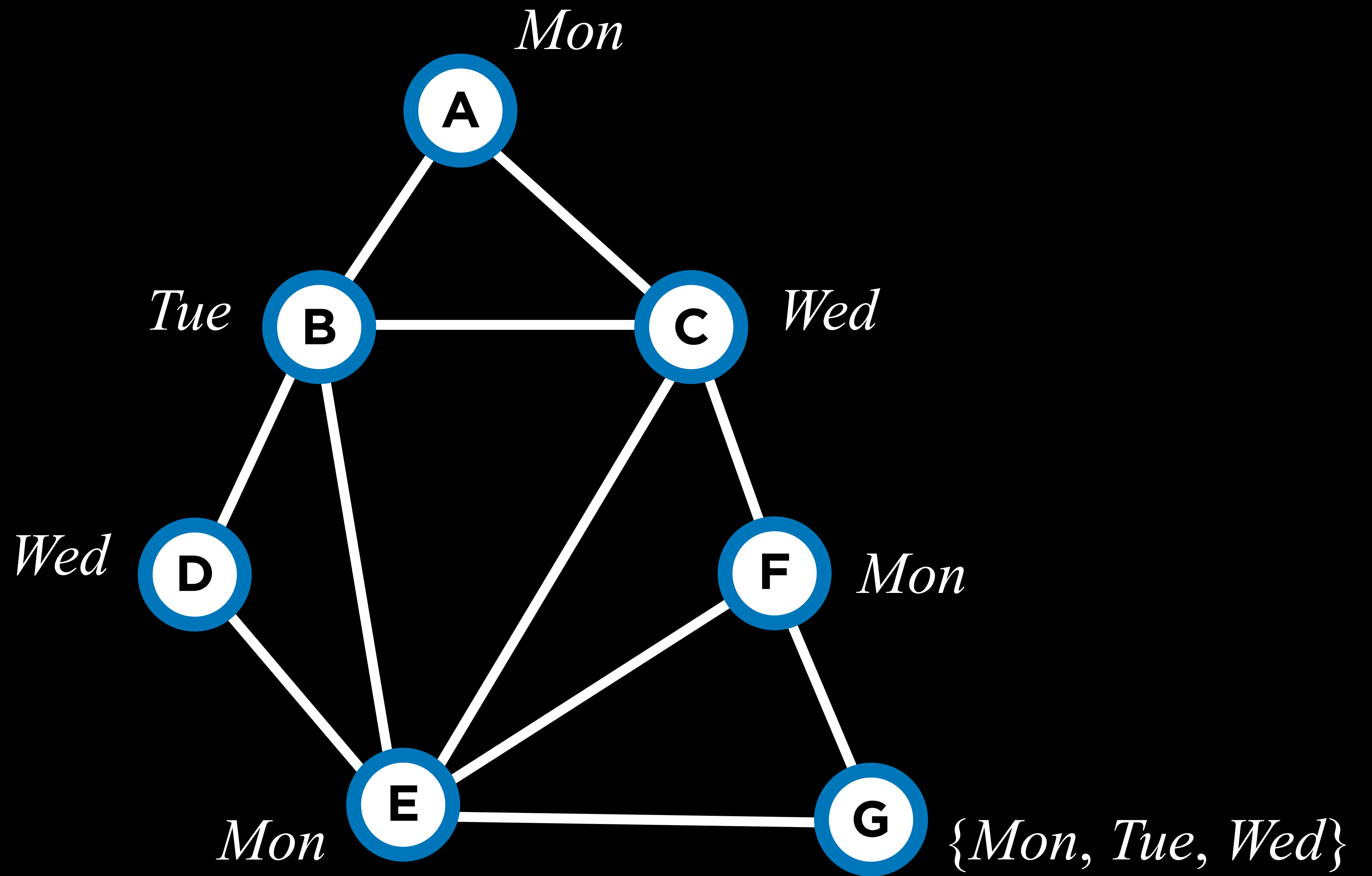


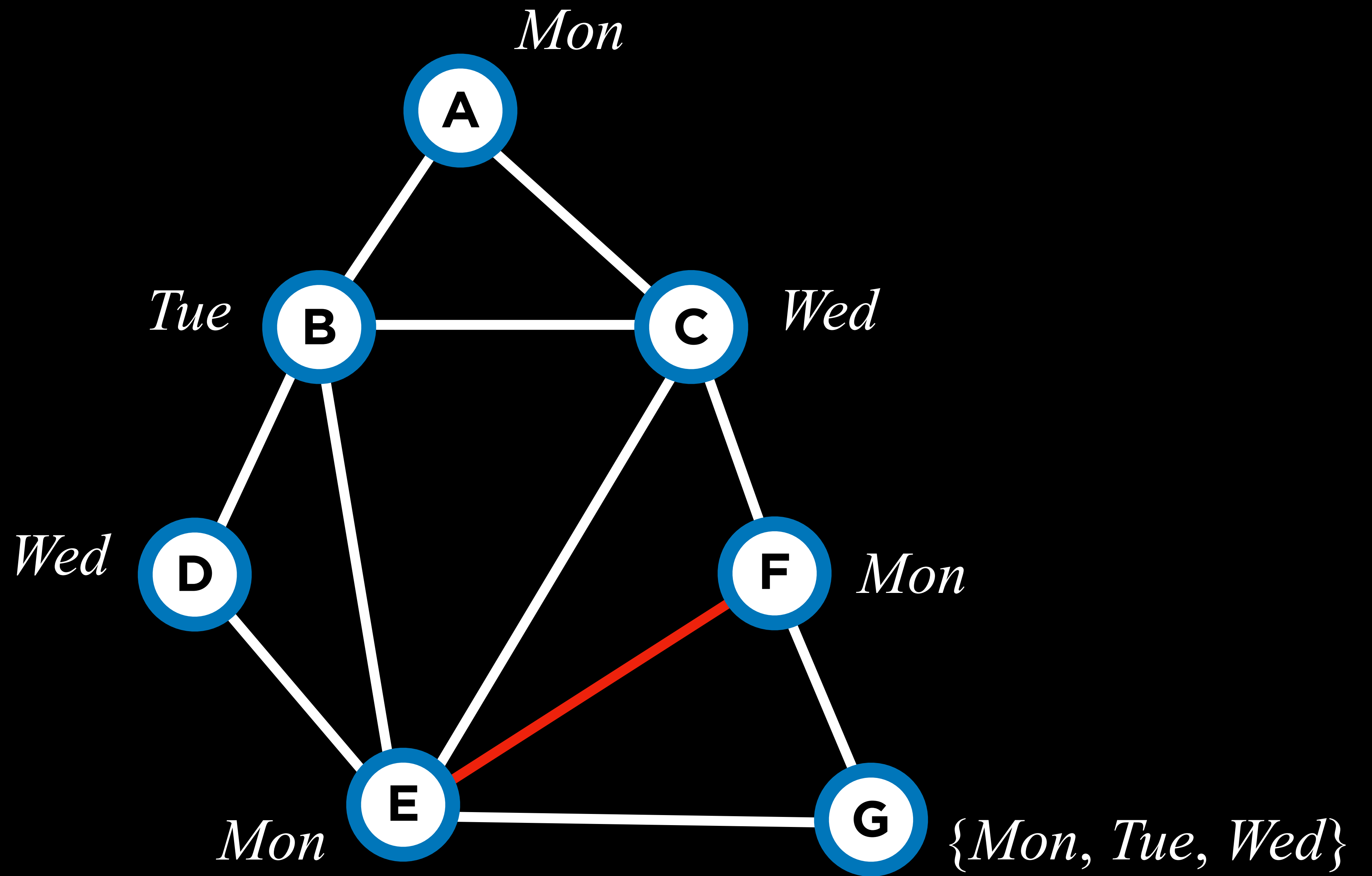


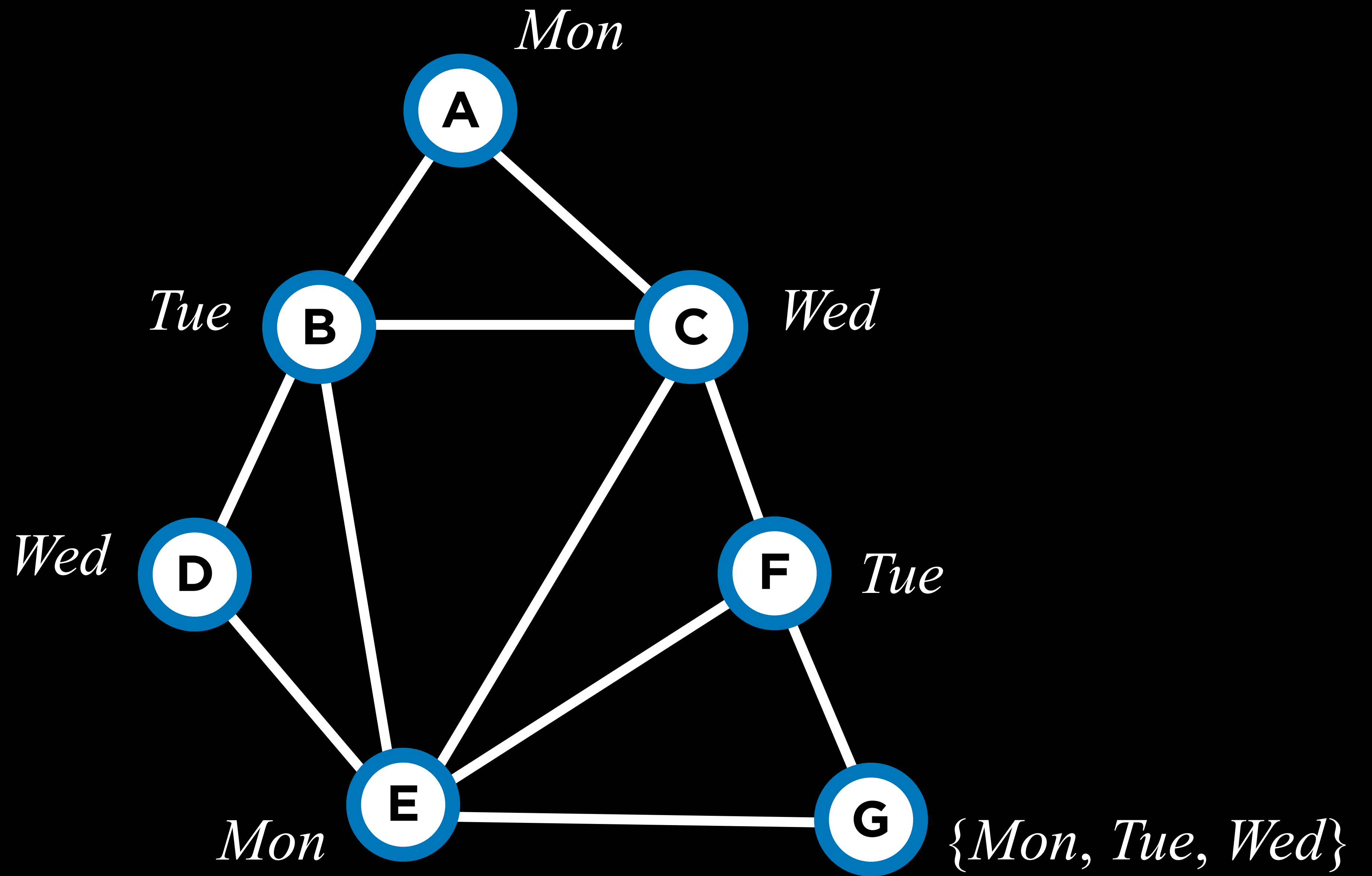


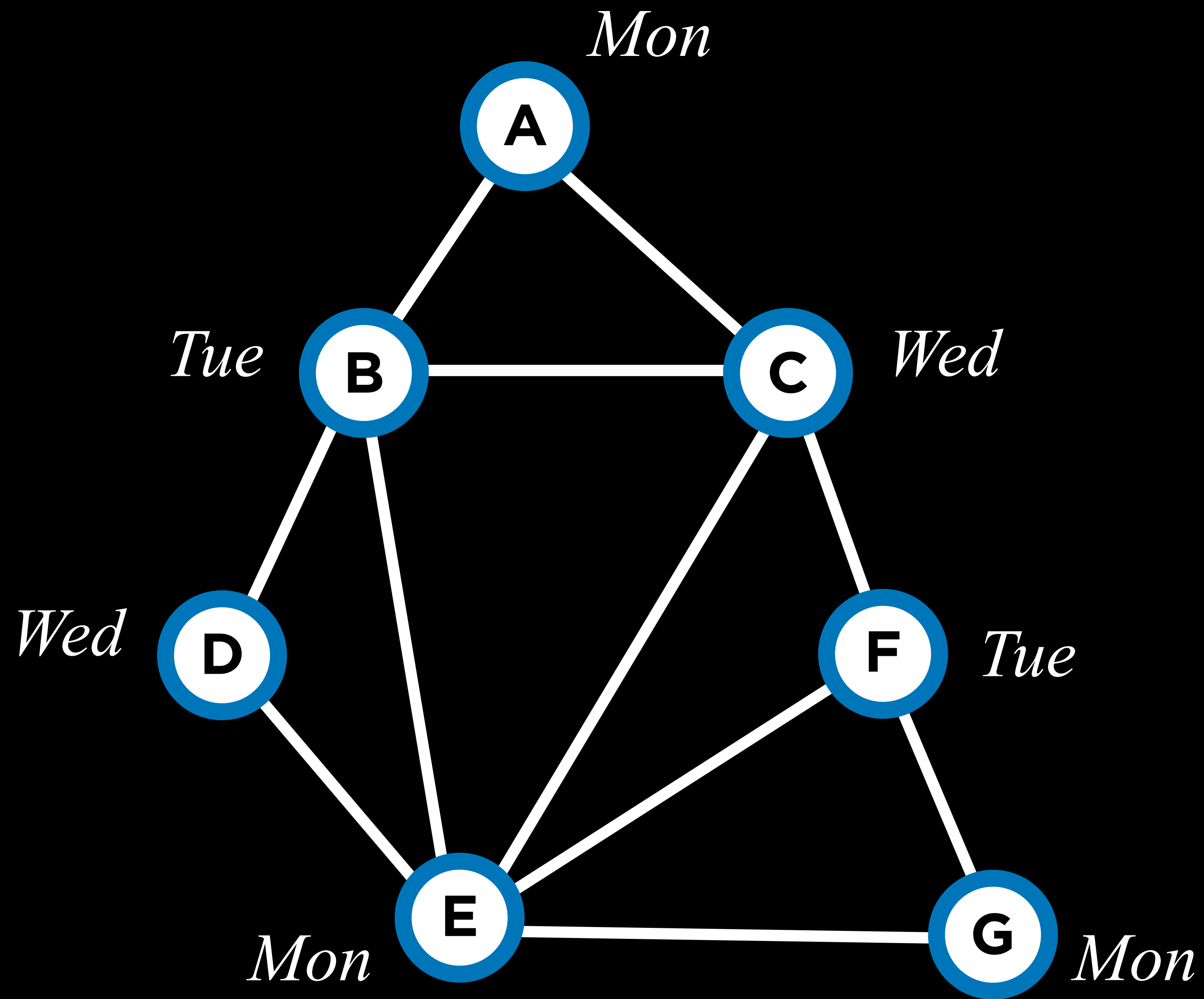


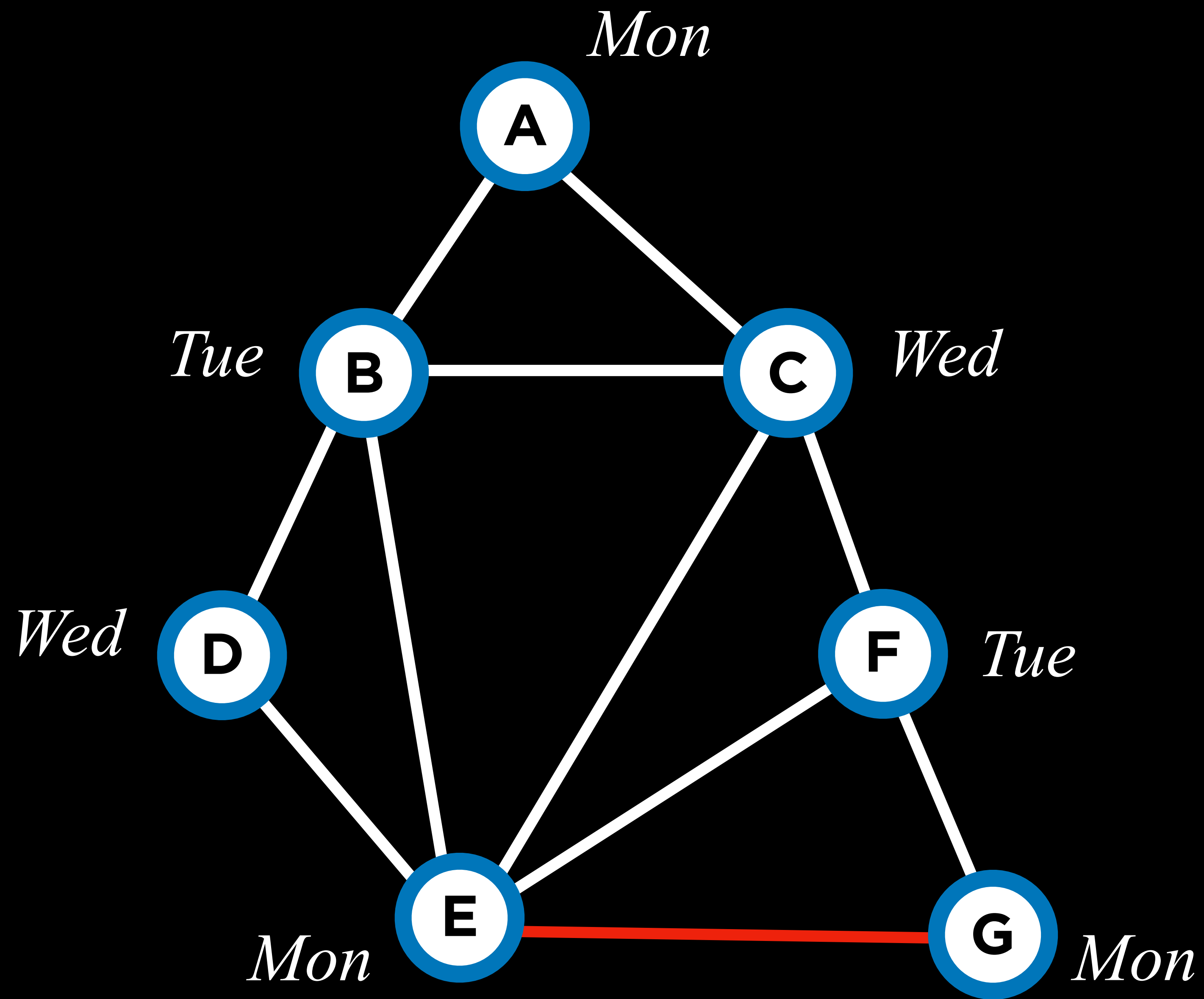


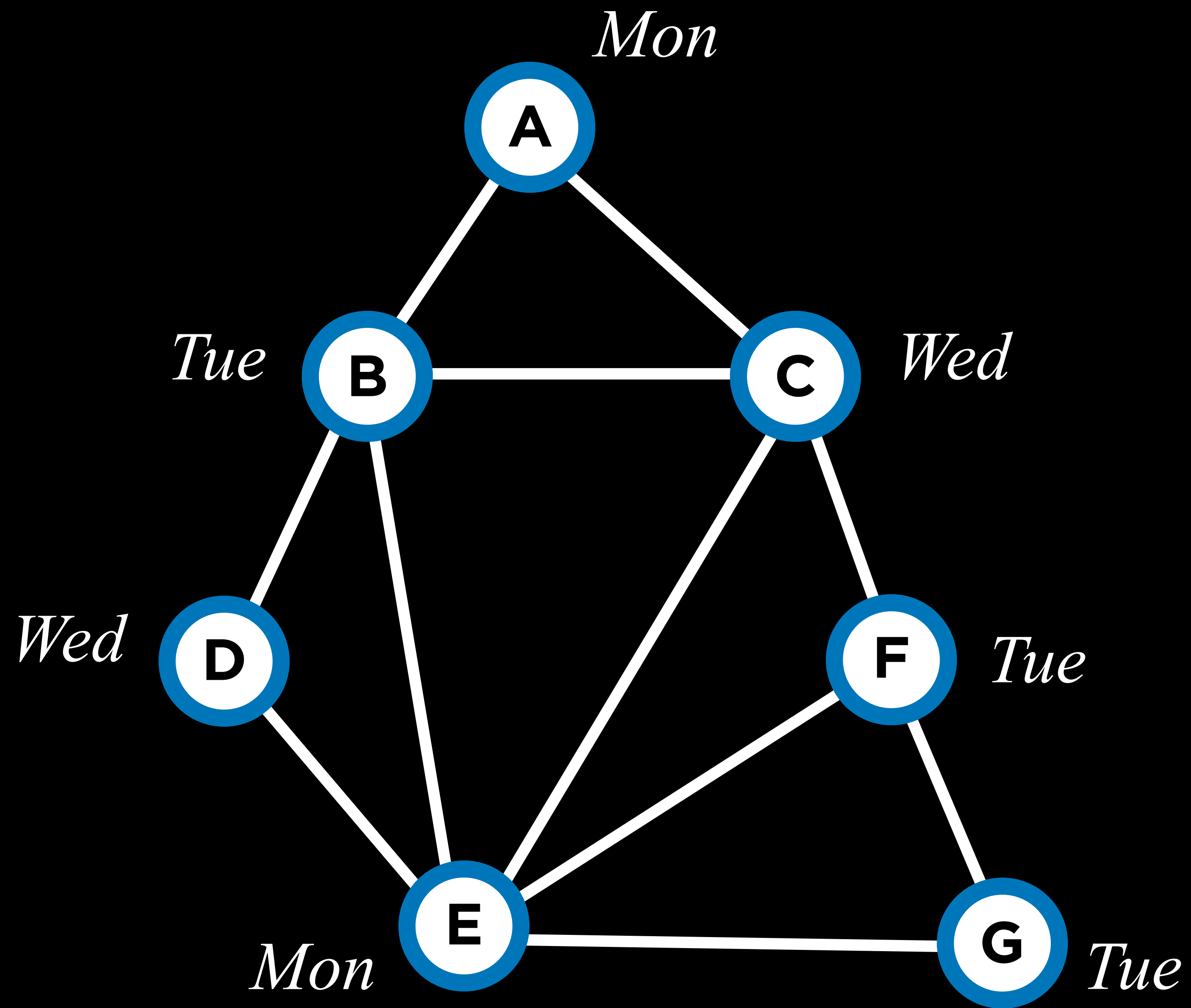


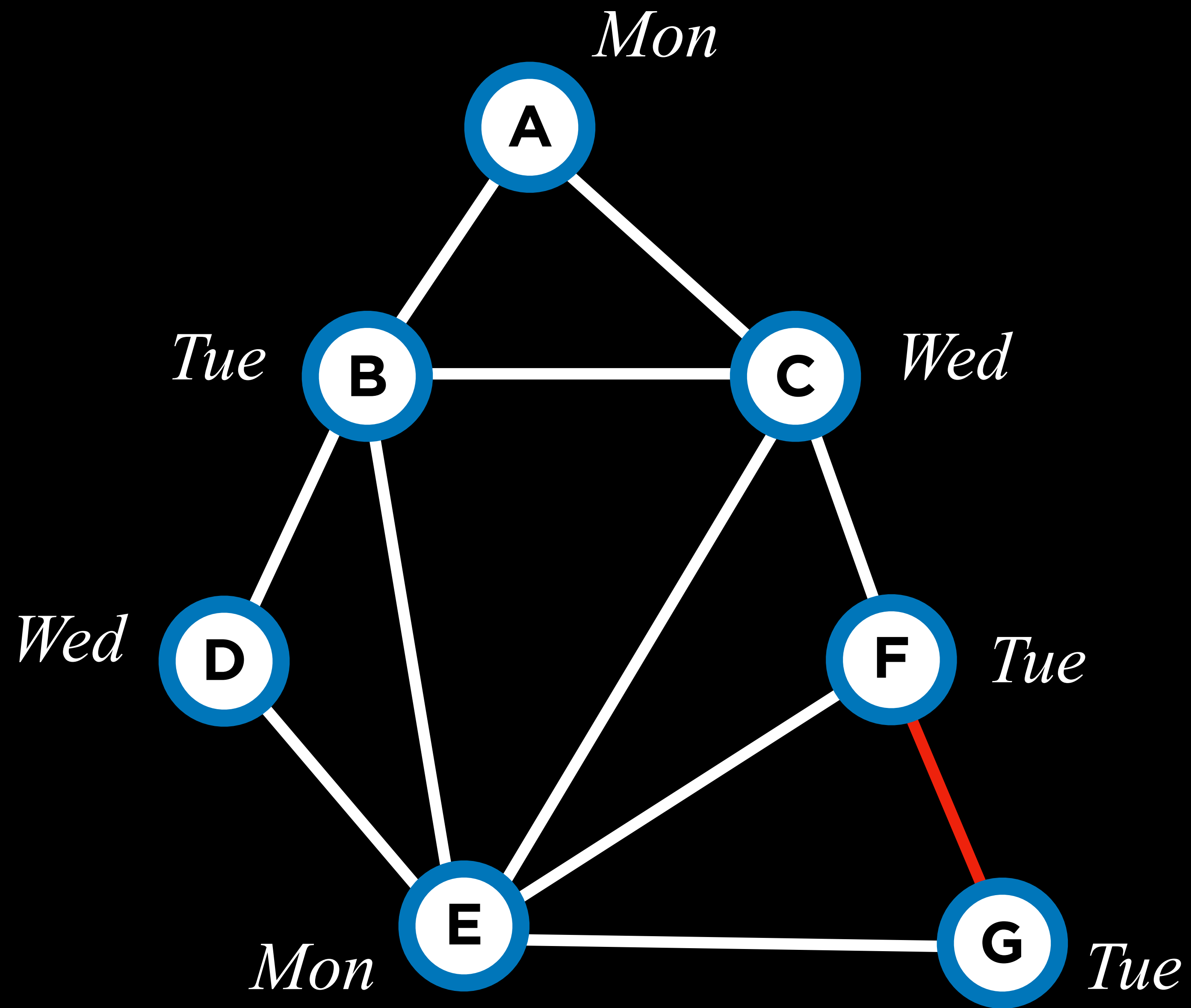


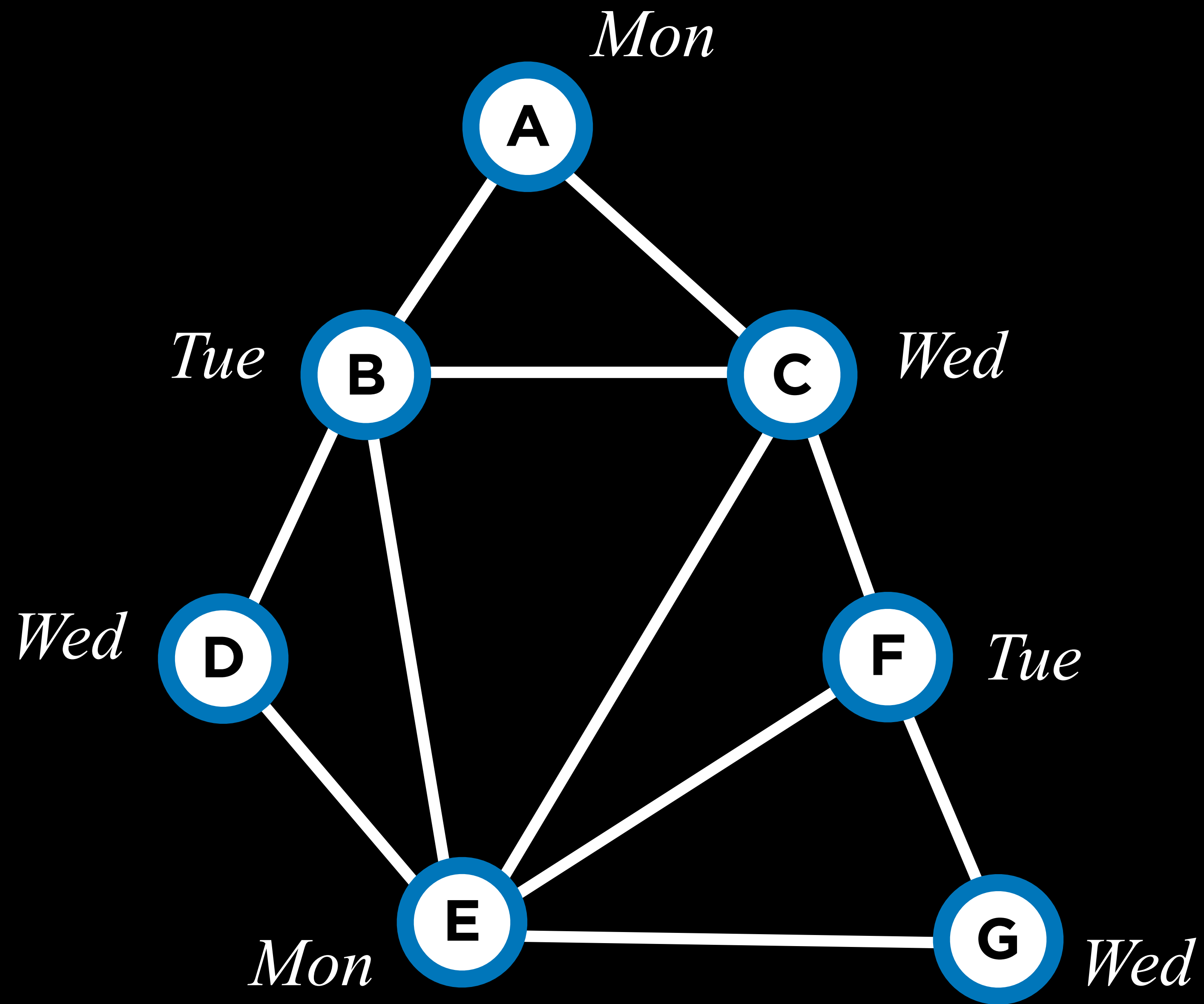




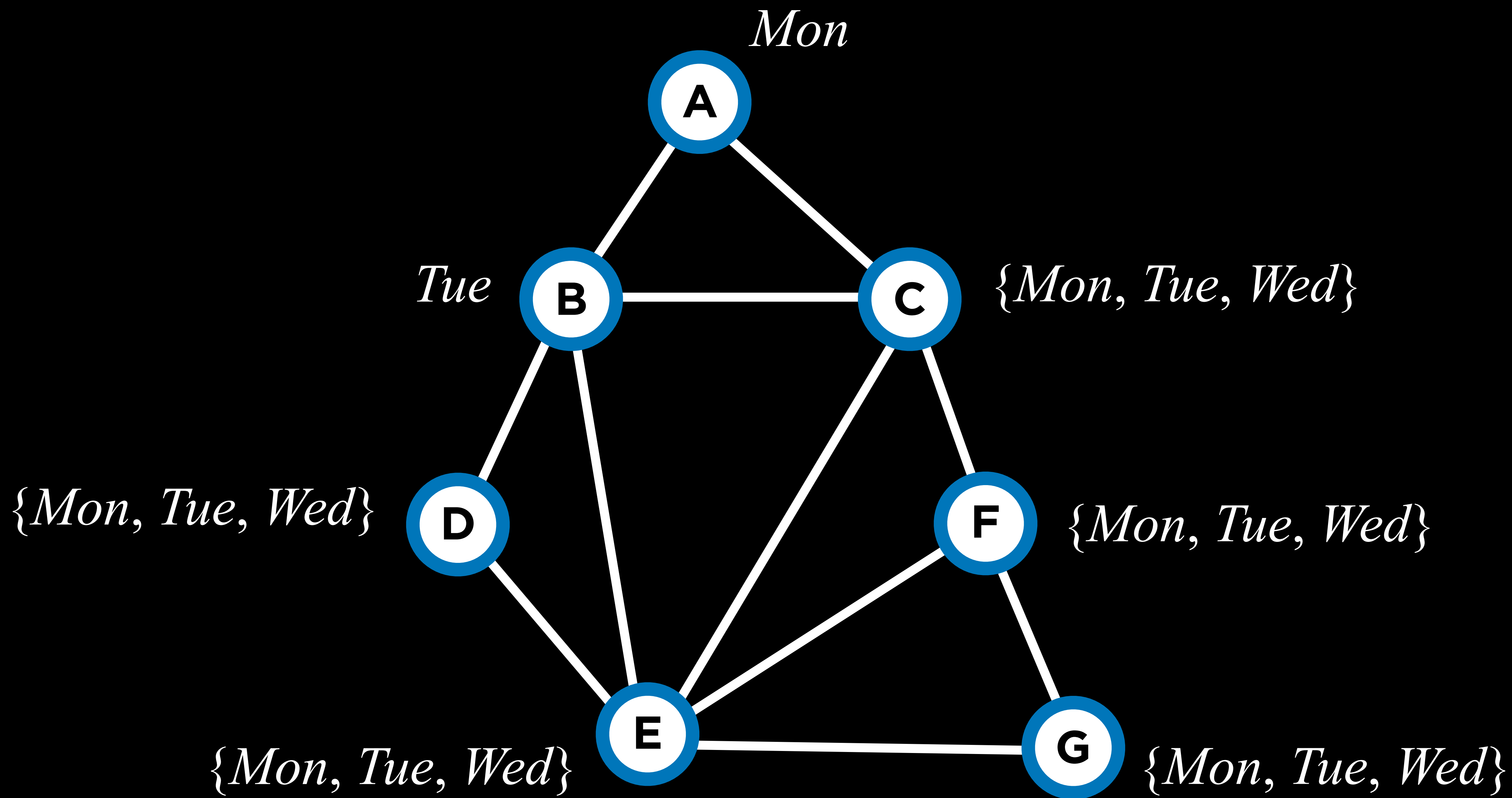


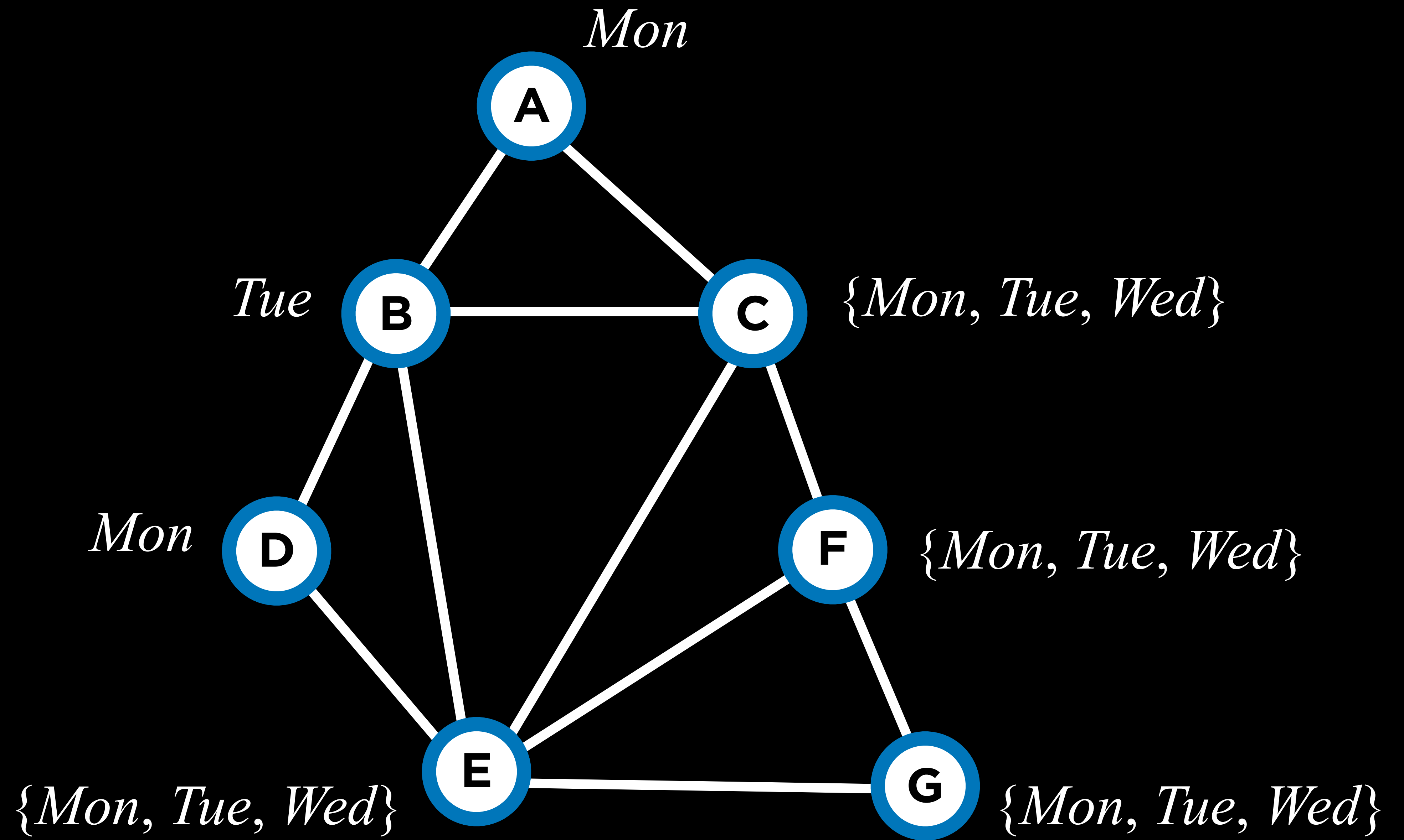


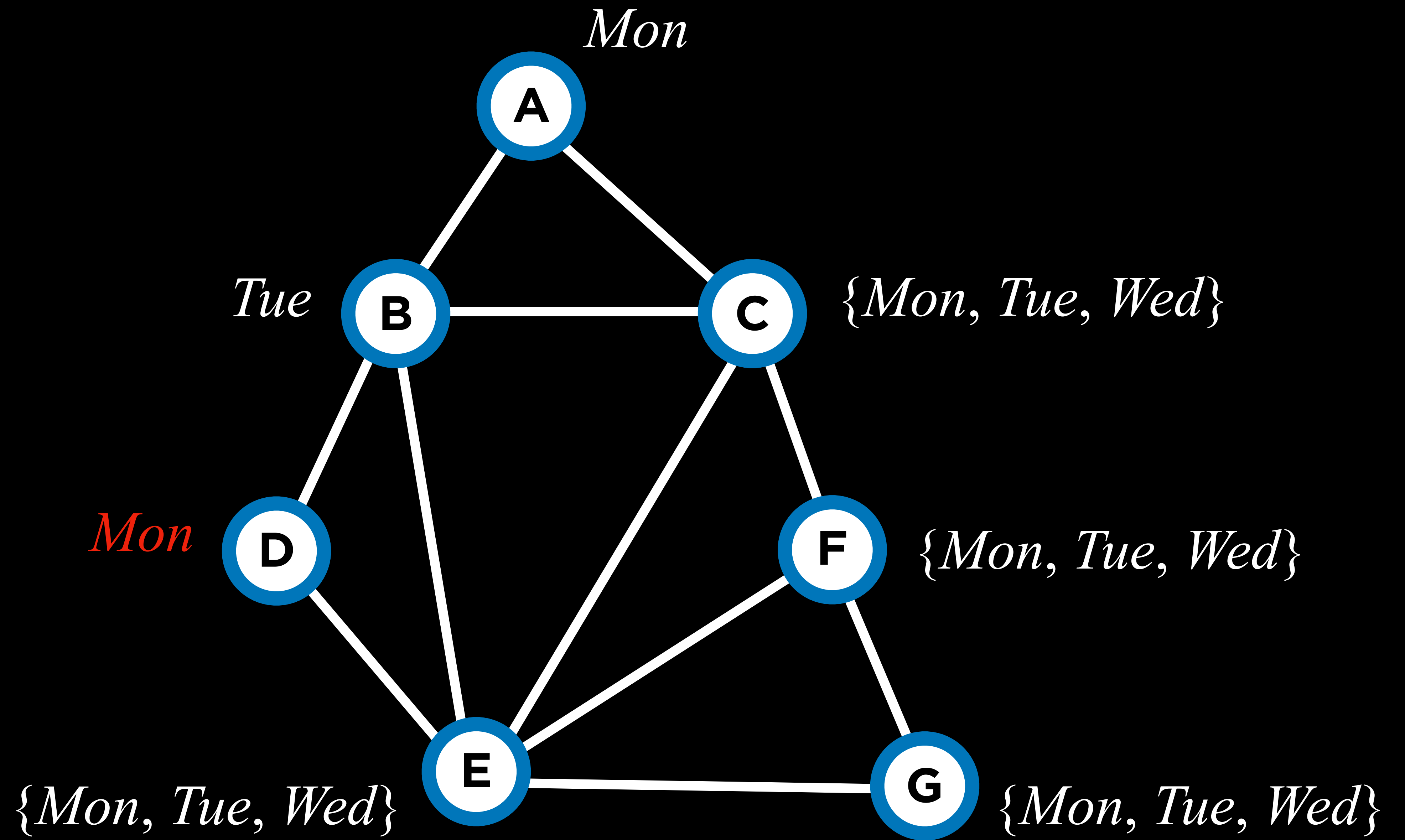


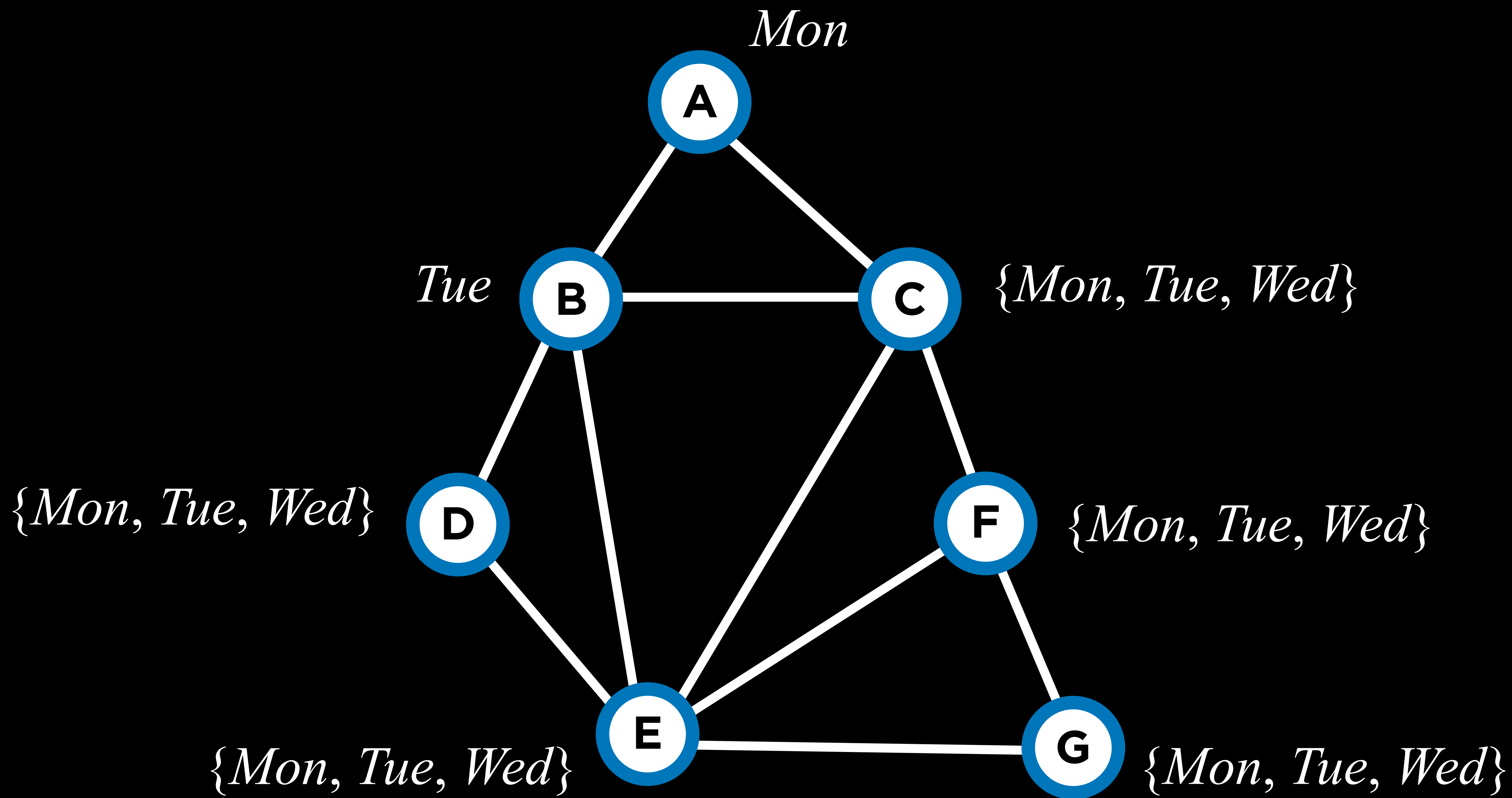


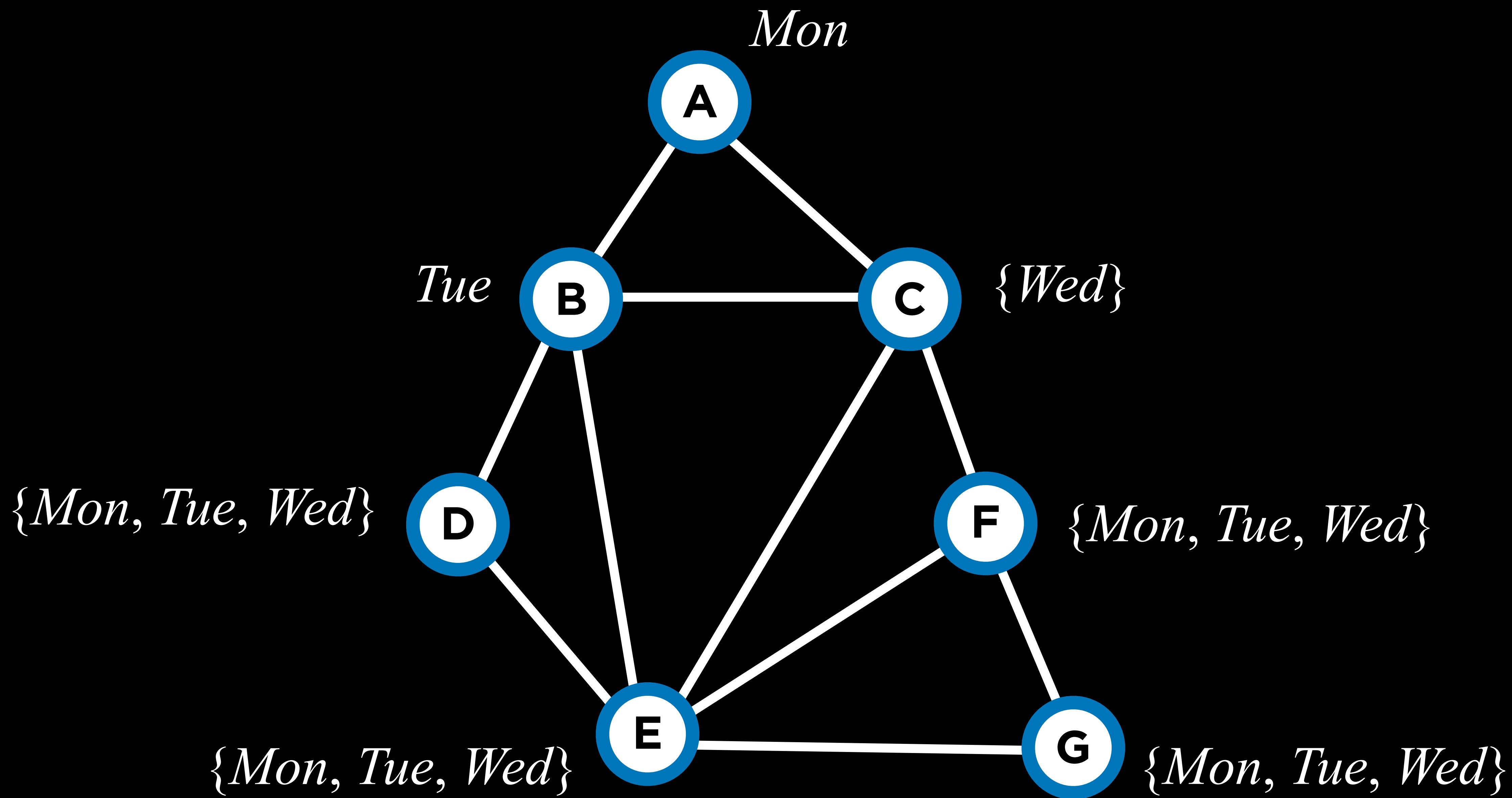
Inference

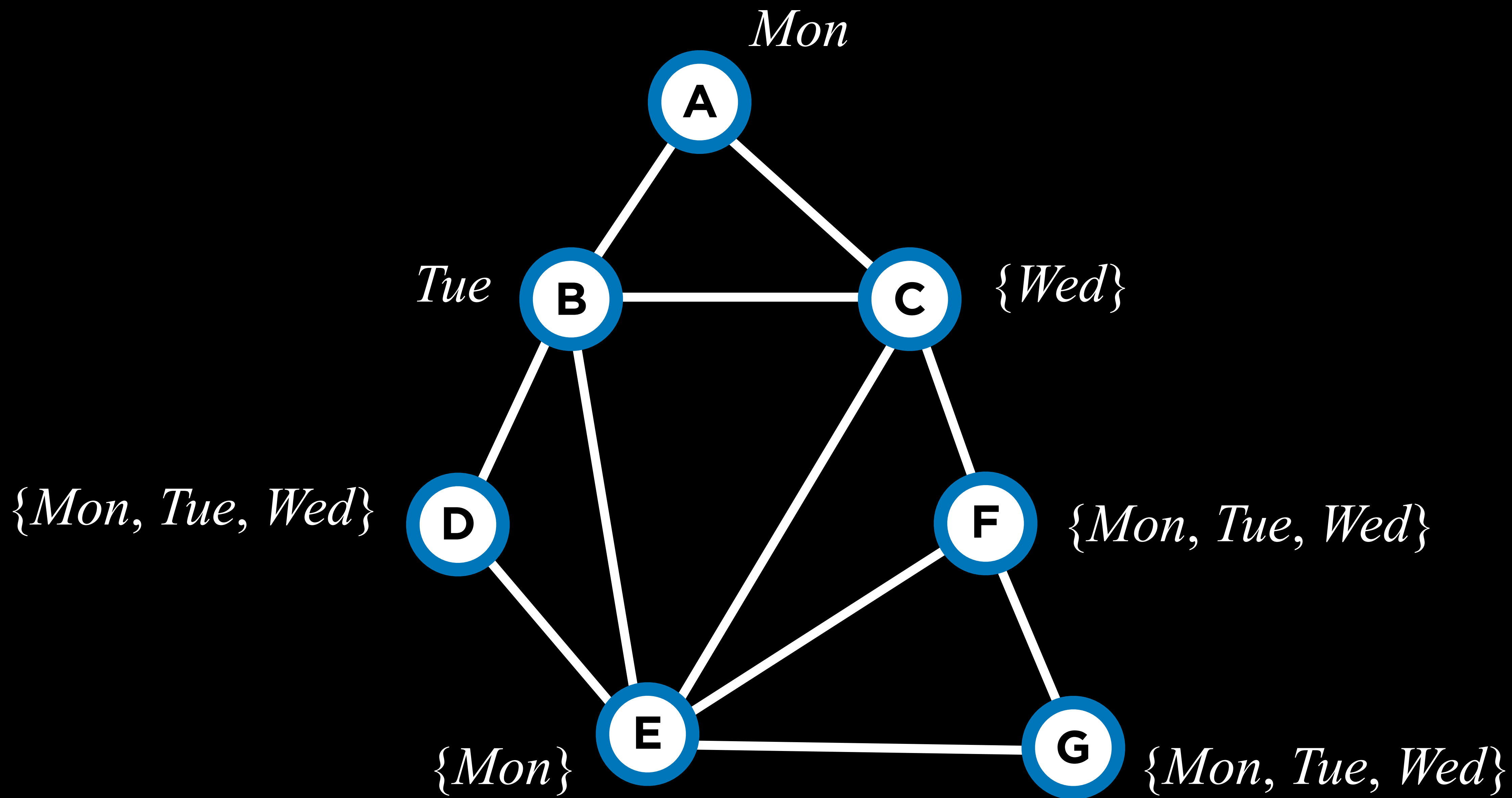


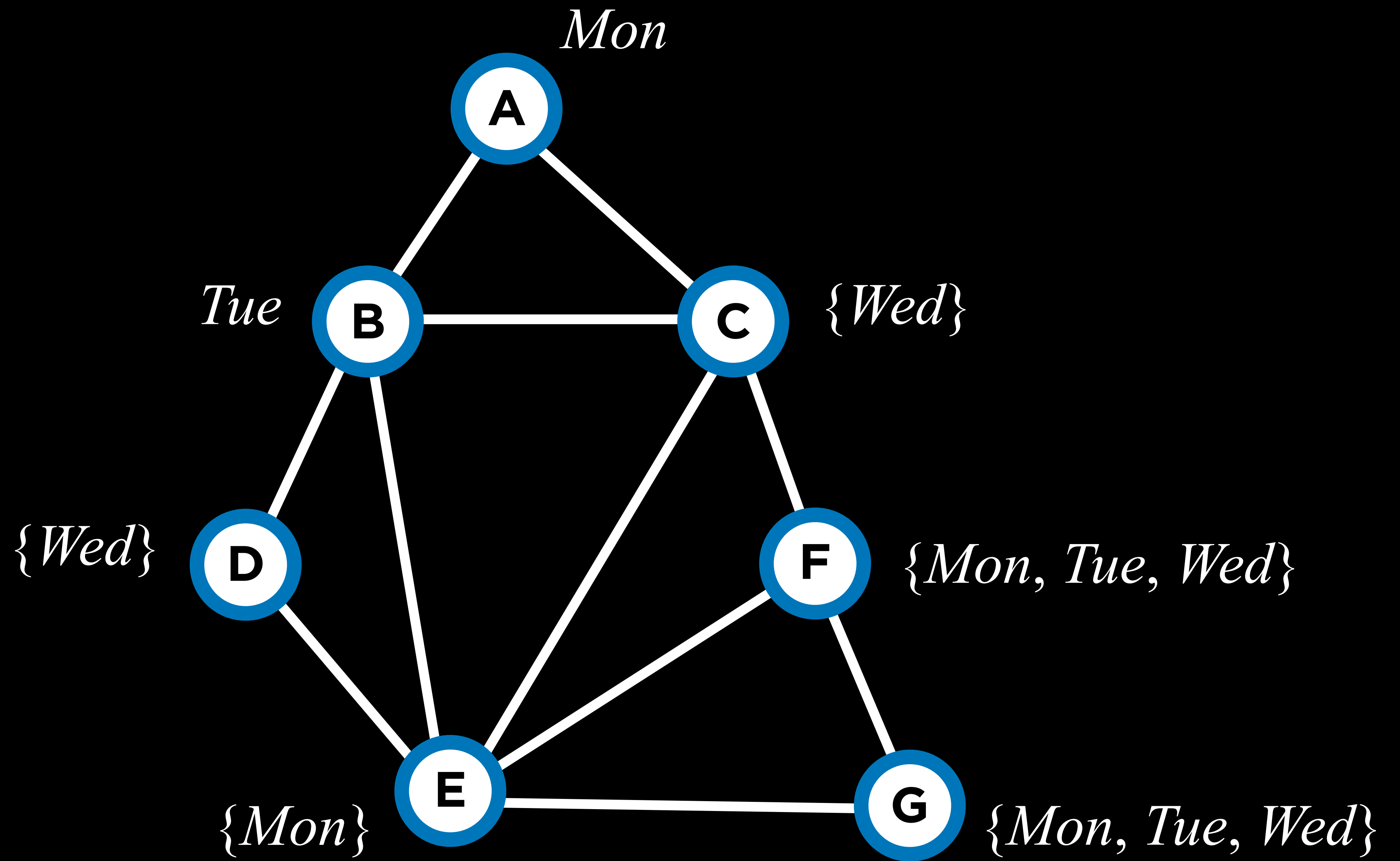


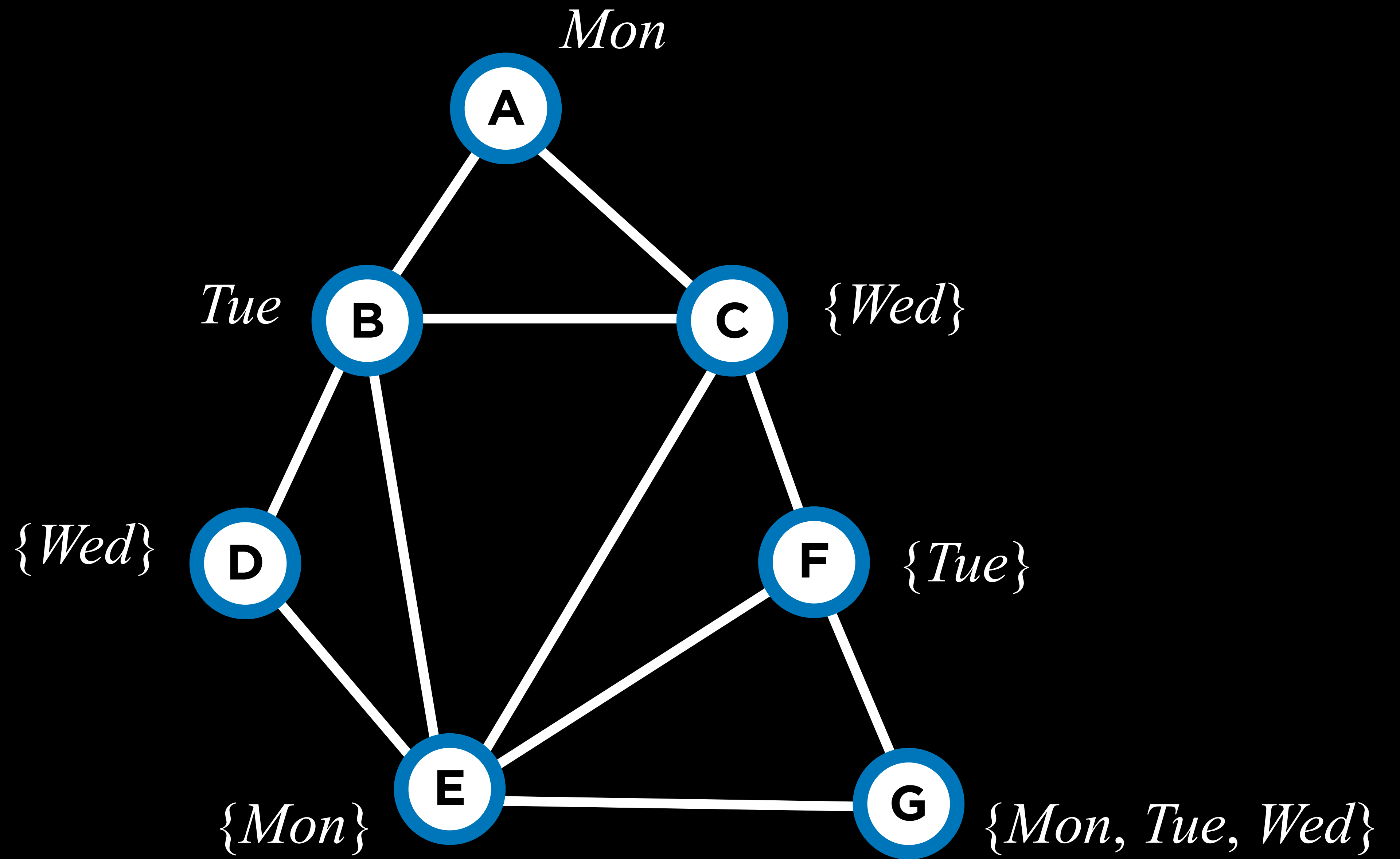


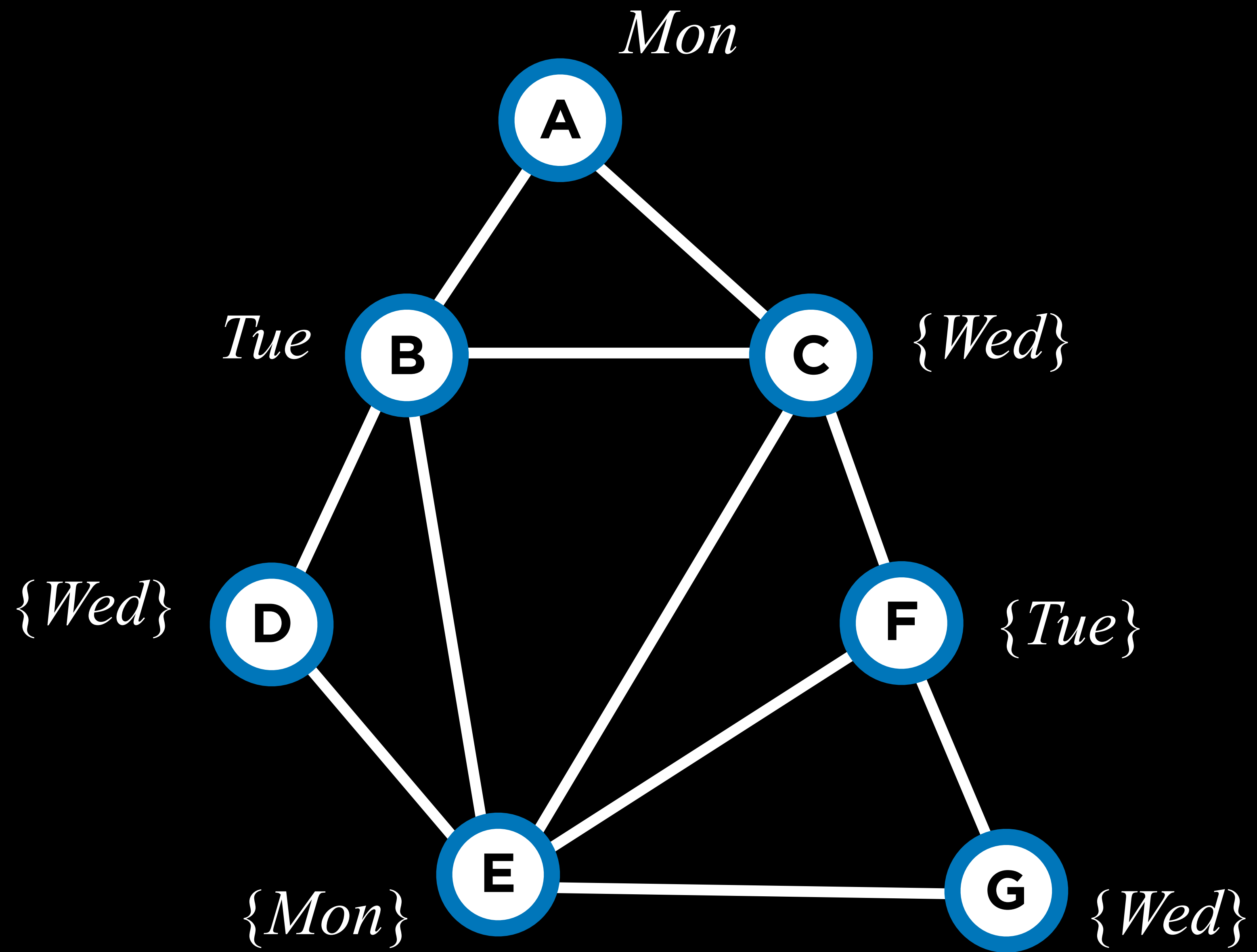


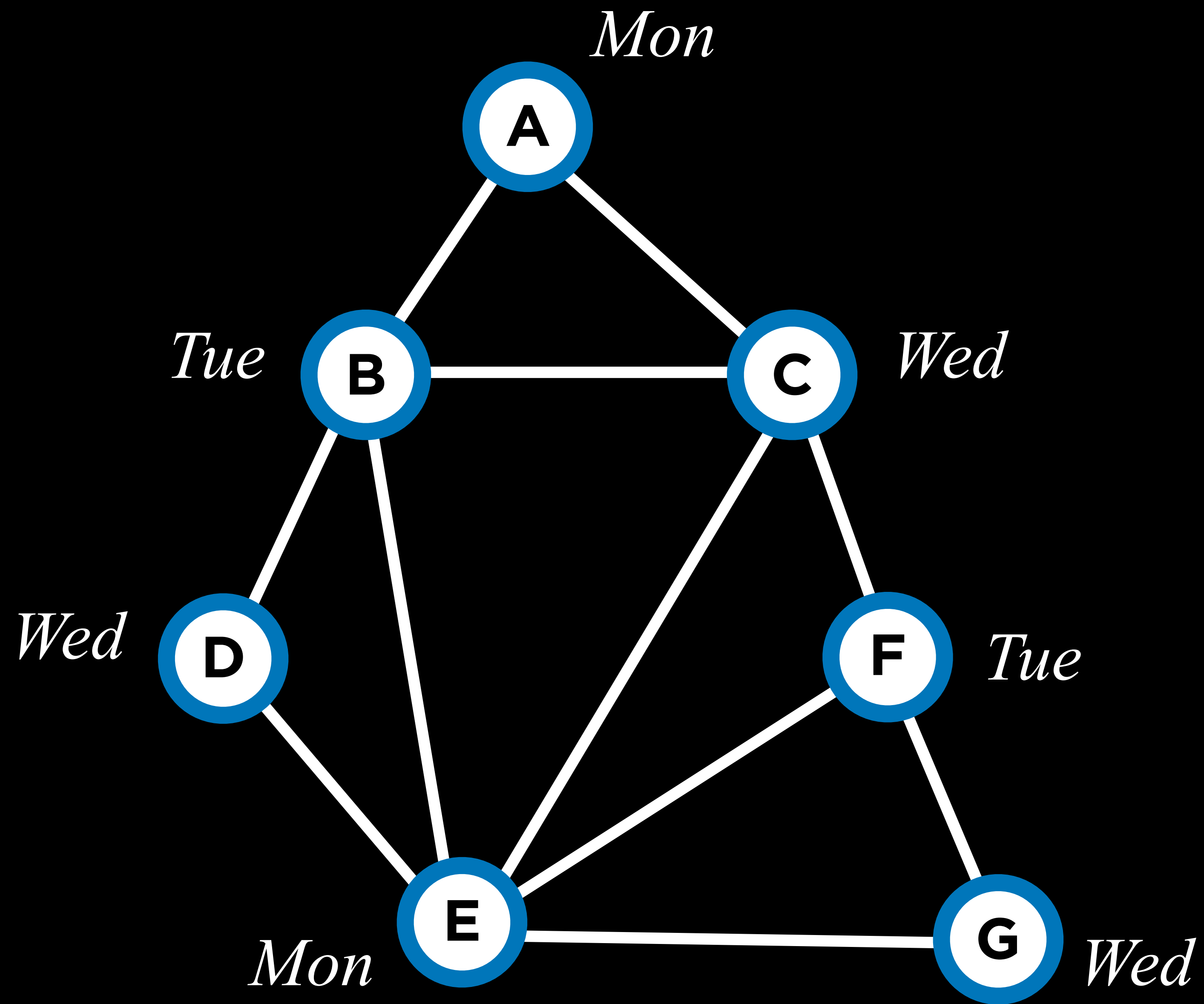












maintaining arc-consistency

algorithm for enforcing arc-consistency
every time we make a new assignment

maintaining arc-consistency

When we make a new assignment to X , call AC-3, starting with a queue of all arcs (Y, X) where Y is a neighbor of X

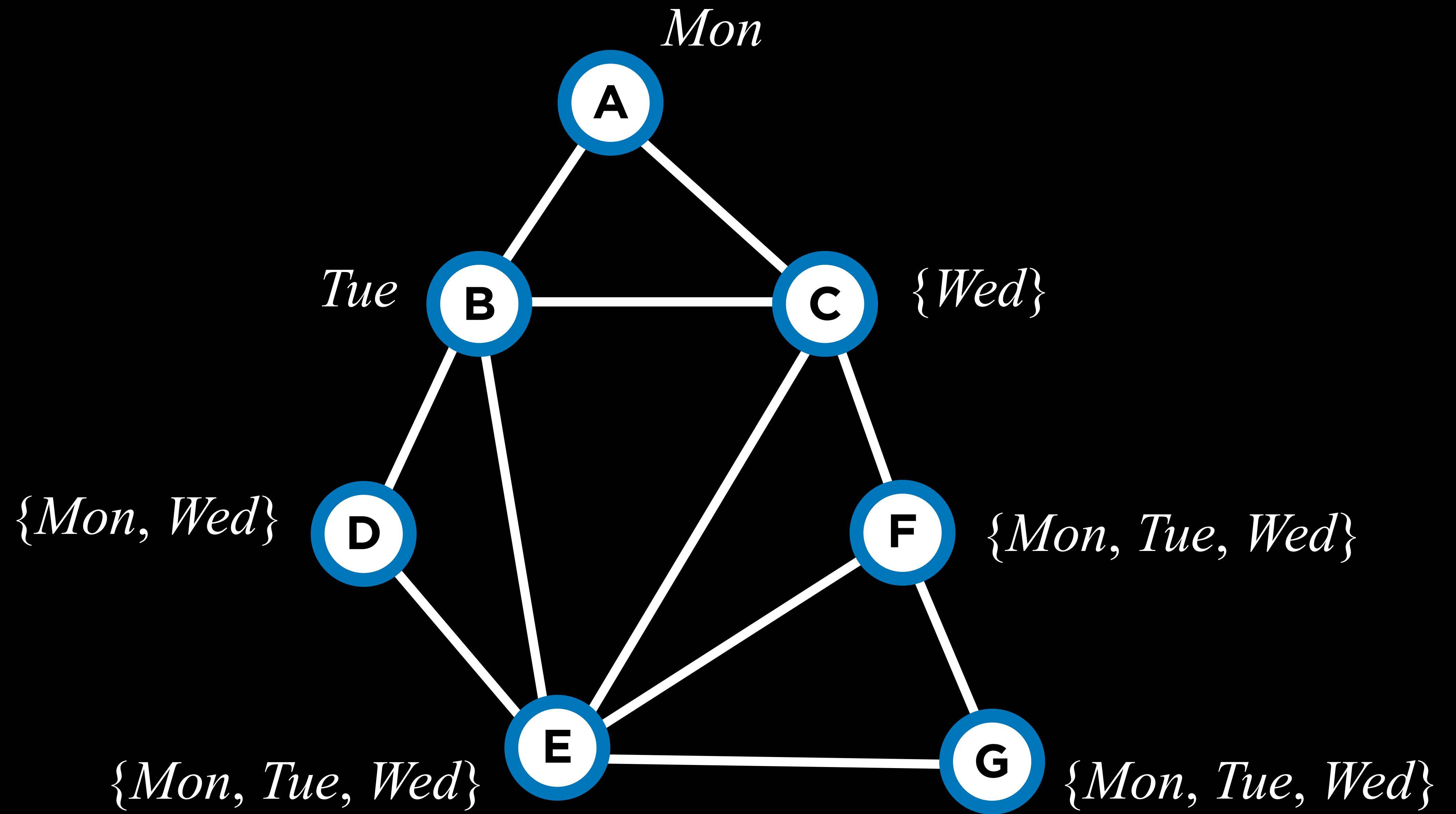
```
function BACKTRACK(assignment, csp):  
  if assignment complete: return assignment  
  var = SELECT-UNASSIGNED-VAR(assignment, csp)  
  for value in DOMAIN-VALUES(var, assignment, csp):  
    if value consistent with assignment:  
      add {var = value} to assignment  
      inferences = INFERENCE(assignment, csp)  
      if inferences ≠ failure: add inferences to assignment  
      result = BACKTRACK(assignment, csp)  
      if result ≠ failure: return result  
      remove {var = value} and inferences from assignment  
  return failure
```

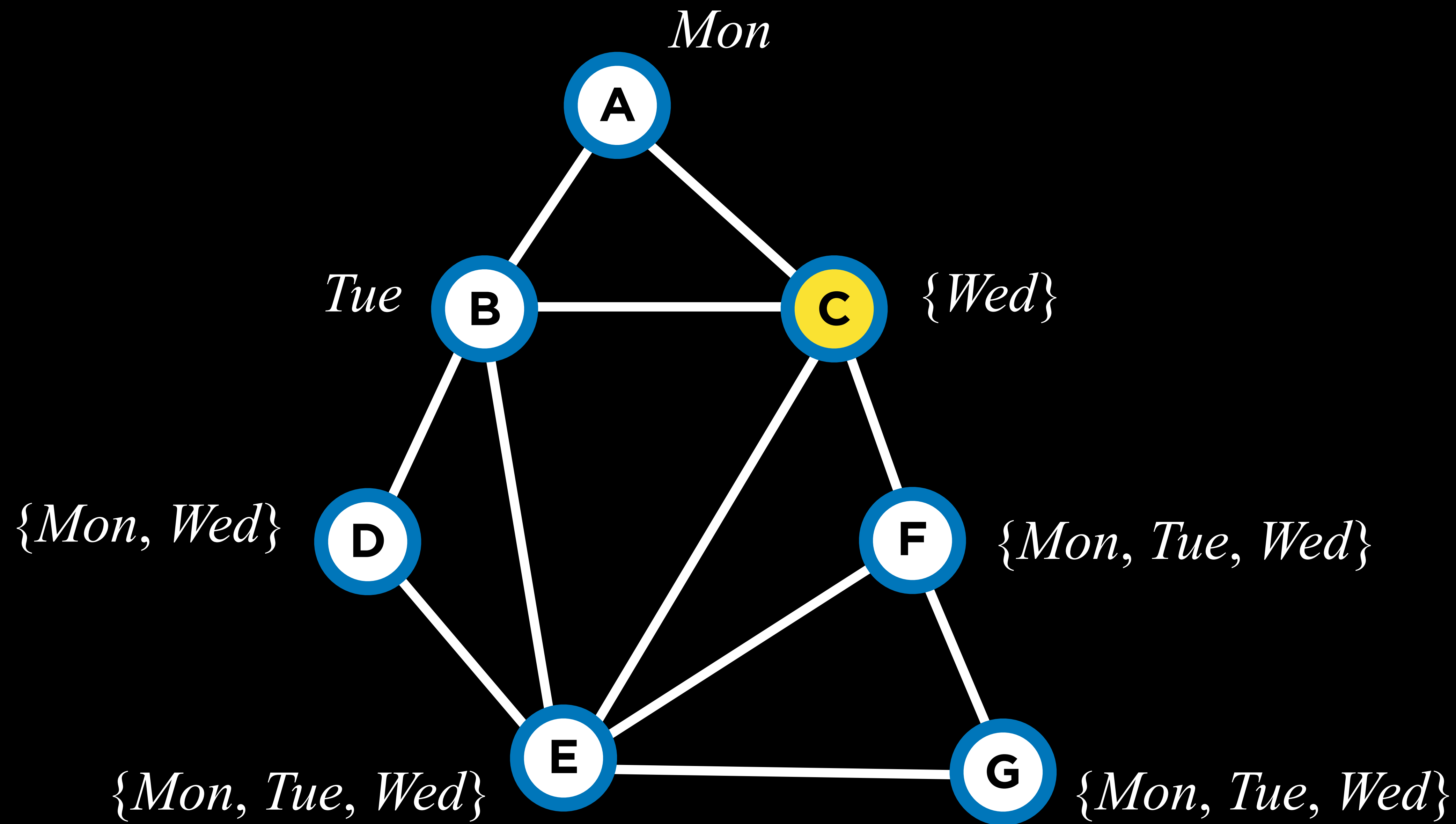
```
function BACKTRACK(assignment, csp):  
  if assignment complete: return assignment  
  var = SELECT-UNASSIGNED-VAR(assignment, csp)  
  for value in DOMAIN-VALUES(var, assignment, csp):  
    if value consistent with assignment:  
      add {var = value} to assignment  
      inferences = INFERENCE(assignment, csp)  
      if inferences ≠ failure: add inferences to assignment  
      result = BACKTRACK(assignment, csp)  
      if result ≠ failure: return result  
      remove {var = value} and inferences from assignment  
  return failure
```

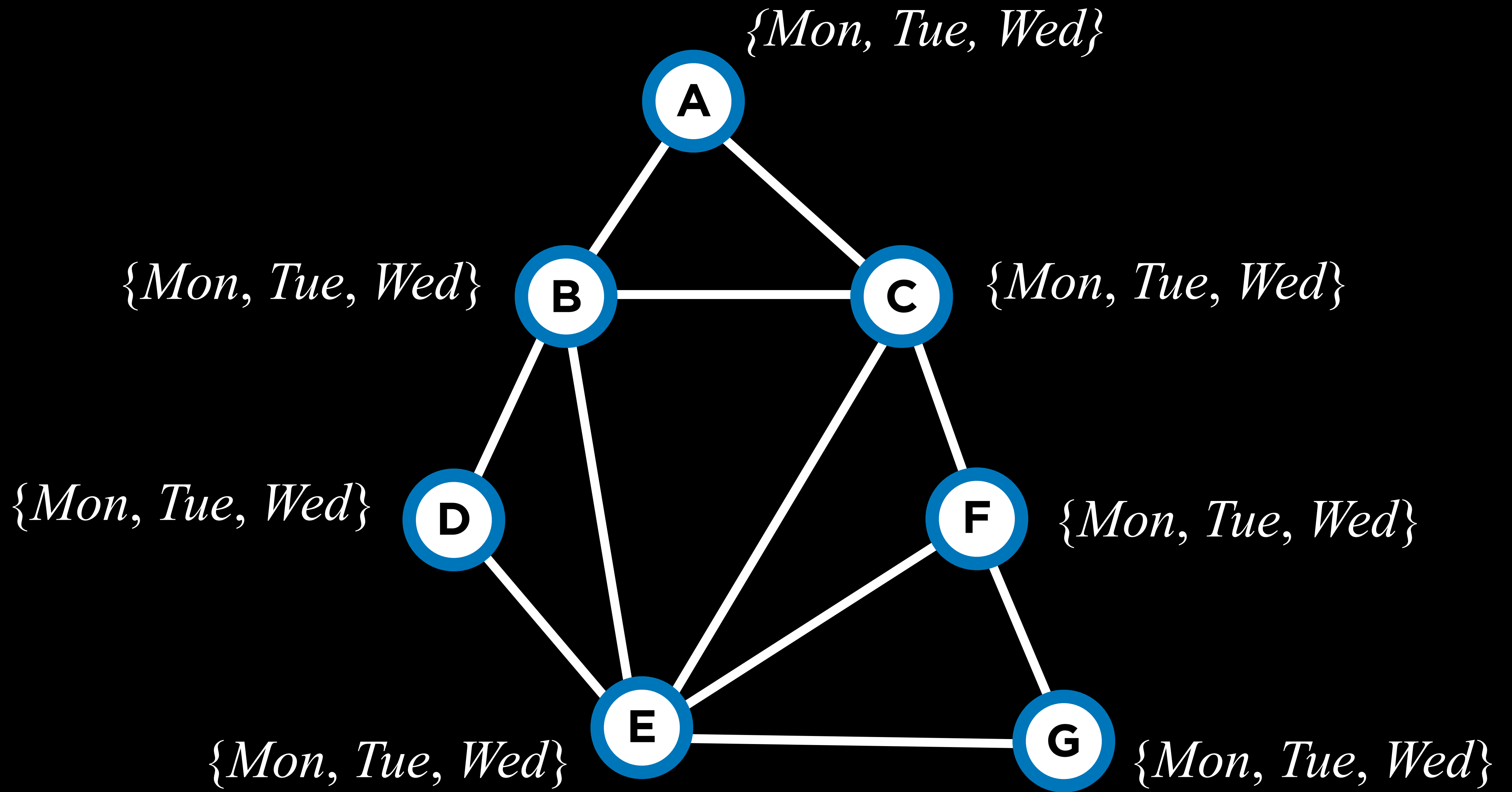
```
function BACKTRACK(assignment, csp):  
  if assignment complete: return assignment  
  var = SELECT-UNASSIGNED-VAR(assignment, csp)  
  for value in DOMAIN-VALUES(var, assignment, csp):  
    if value consistent with assignment:  
      add {var = value} to assignment  
      inferences = INFERENCE(assignment, csp)  
      if inferences ≠ failure: add inferences to assignment  
      result = BACKTRACK(assignment, csp)  
      if result ≠ failure: return result  
      remove {var = value} and inferences from assignment  
  return failure
```

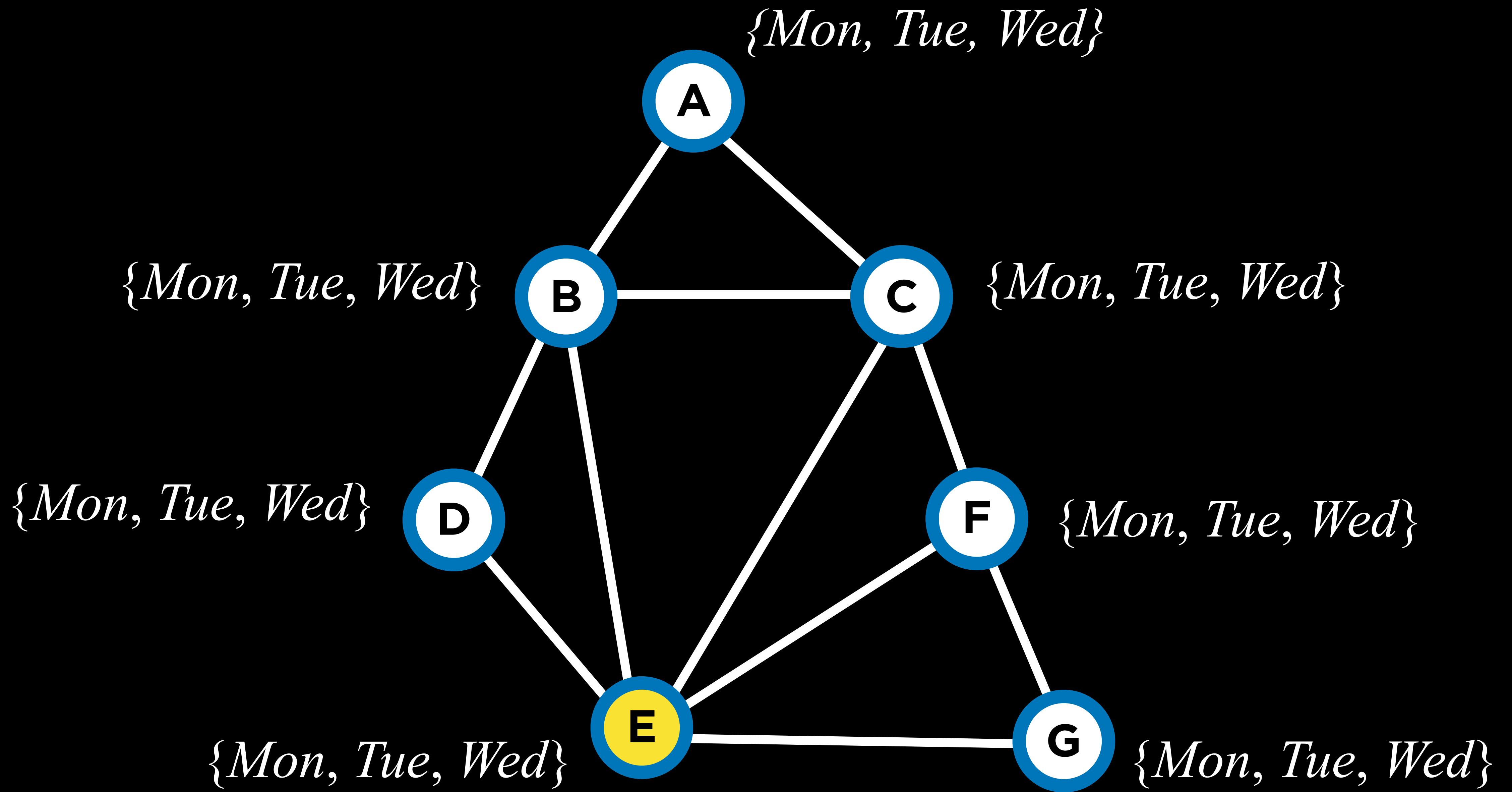
SELECT-UNASSIGNED-VAR

- **minimum remaining values (MRV)** heuristic: select the variable that has the smallest domain
- **degree** heuristic: select the variable that has the highest degree







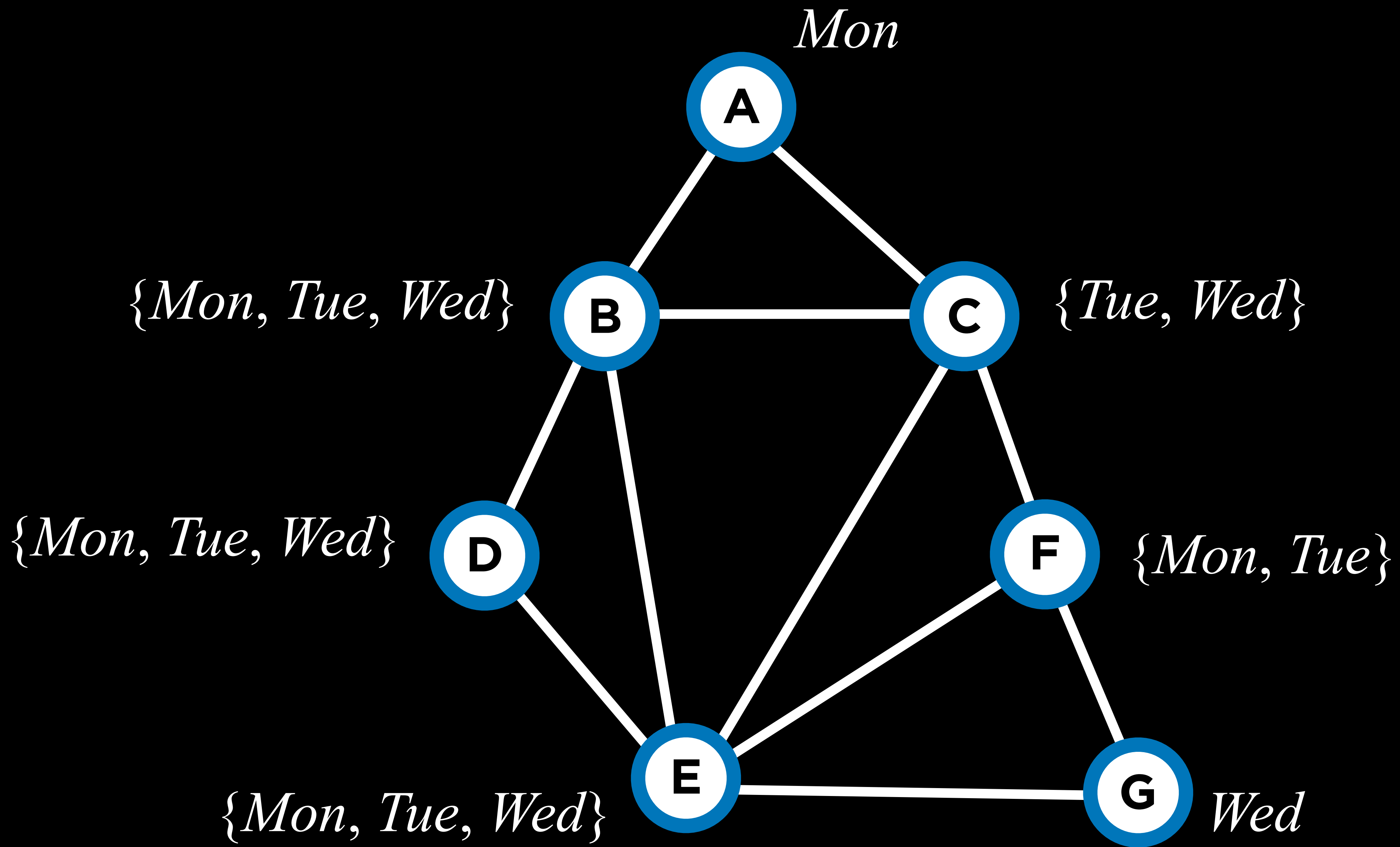


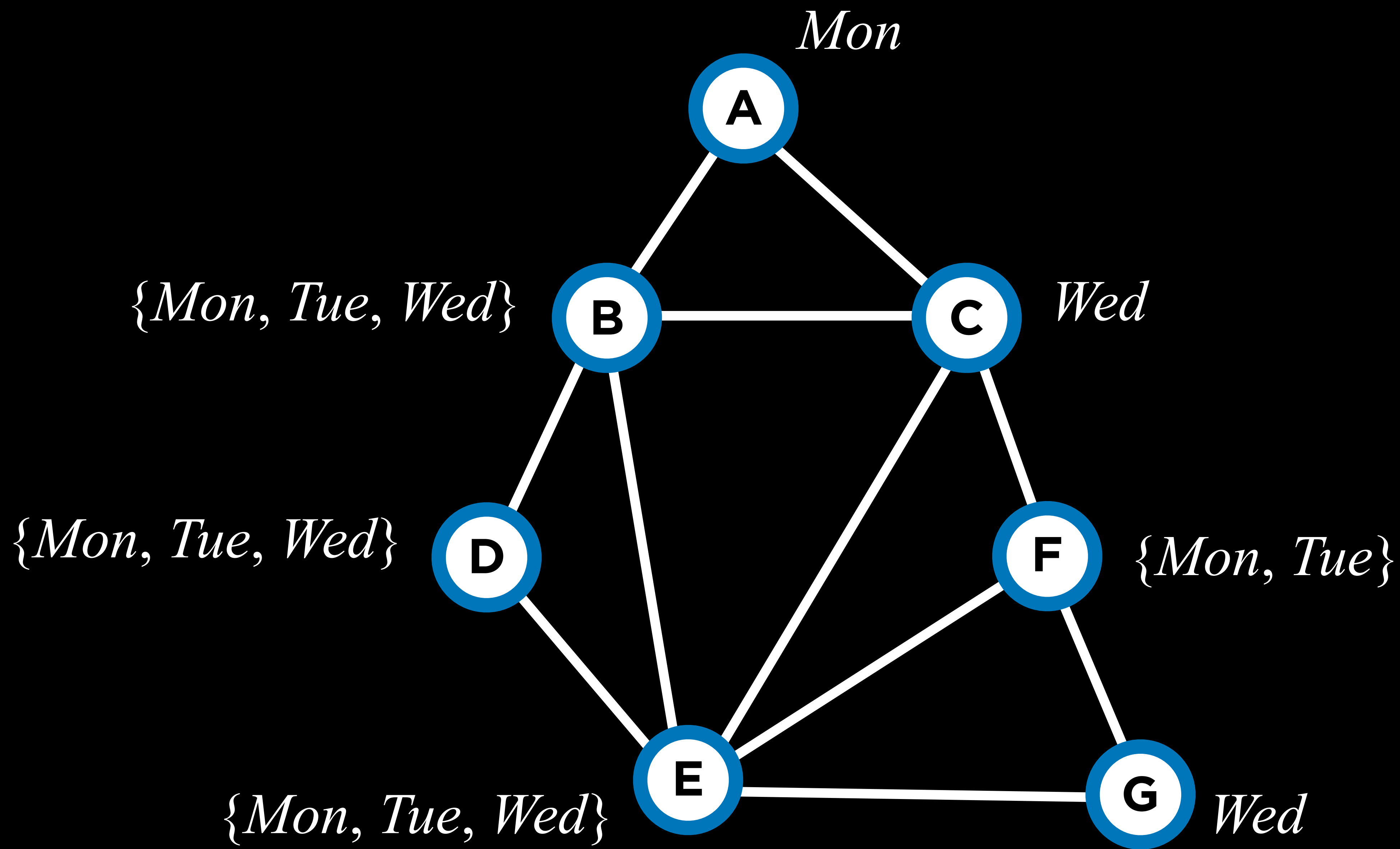
```
function BACKTRACK(assignment, csp):  
  if assignment complete: return assignment  
  var = SELECT-UNASSIGNED-VAR(assignment, csp)  
  for value in DOMAIN-VALUES(var, assignment, csp):  
    if value consistent with assignment:  
      add {var = value} to assignment  
      inferences = INFERENCE(assignment, csp)  
      if inferences ≠ failure: add inferences to assignment  
      result = BACKTRACK(assignment, csp)  
      if result ≠ failure: return result  
      remove {var = value} and inferences from assignment  
  return failure
```

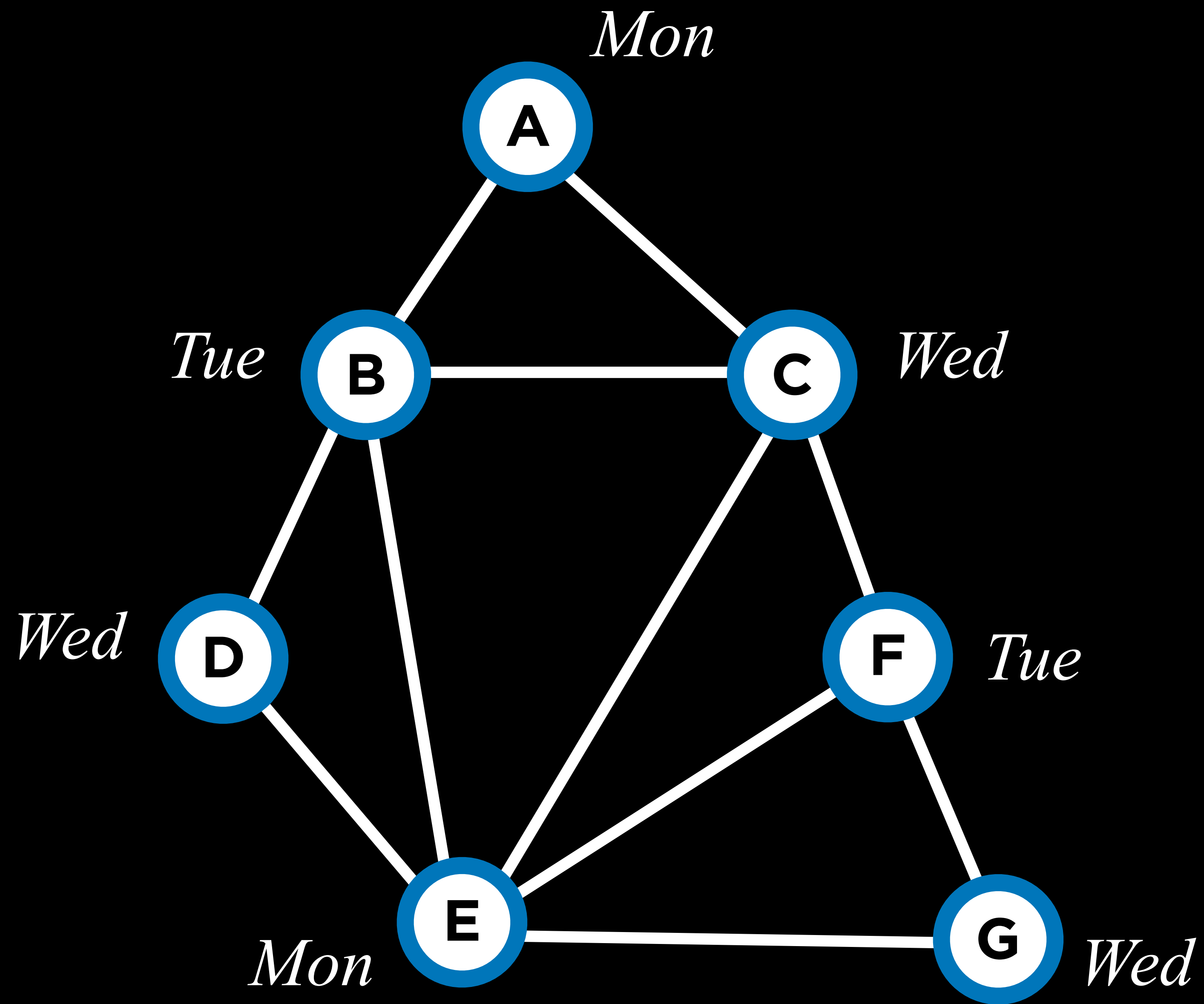
```
function BACKTRACK(assignment, csp):  
  if assignment complete: return assignment  
  var = SELECT-UNASSIGNED-VAR(assignment, csp)  
  for value in DOMAIN-VALUES(var, assignment, csp):  
    if value consistent with assignment:  
      add {var = value} to assignment  
      inferences = INFERENCE(assignment, csp)  
      if inferences ≠ failure: add inferences to assignment  
      result = BACKTRACK(assignment, csp)  
      if result ≠ failure: return result  
      remove {var = value} and inferences from assignment  
  return failure
```

DOMAIN-VALUES

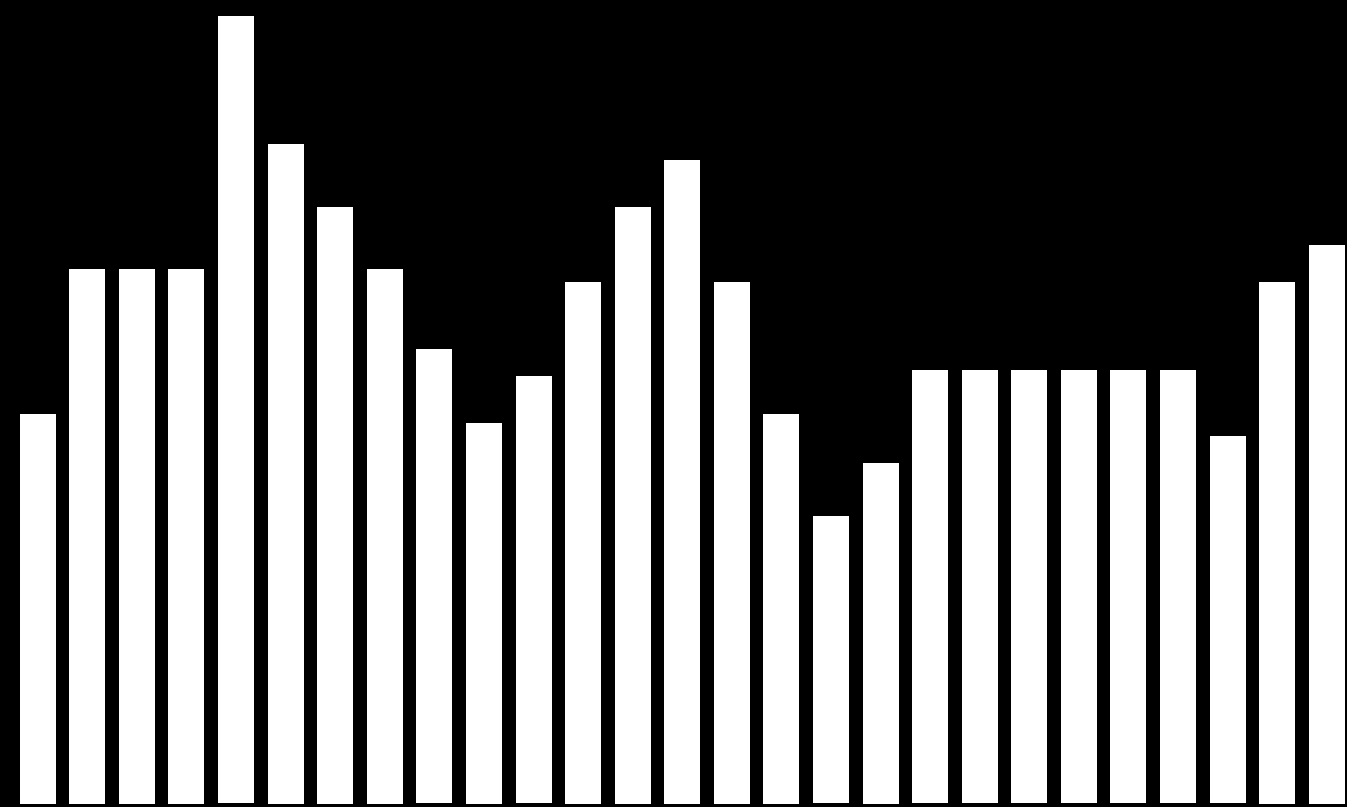
- **least-constraining values** heuristic: return variables in order by number of choices that are ruled out for neighboring variables
 - try least-constraining values first







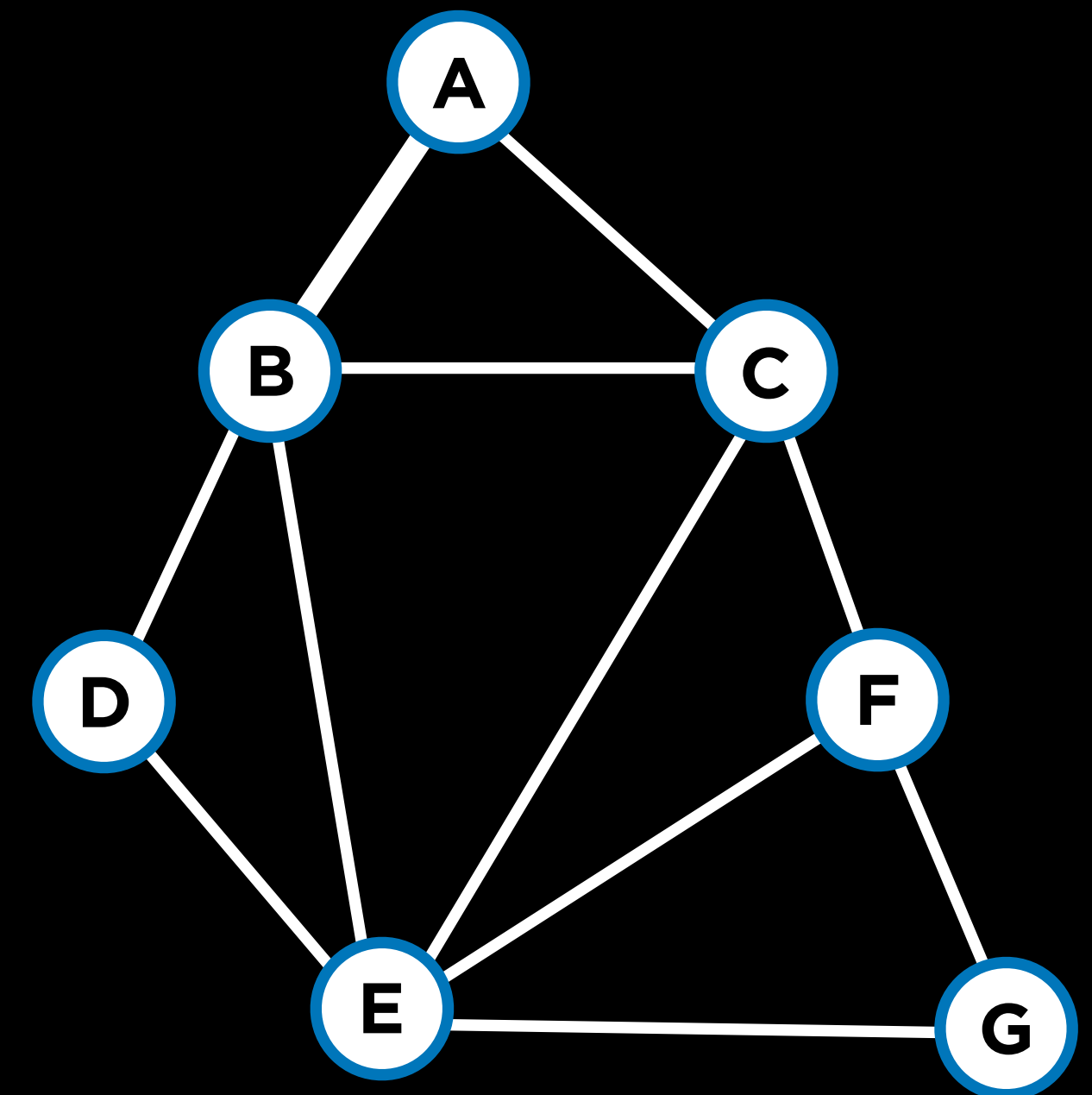
Problem Formulation



Local
Search

$$\begin{aligned} &50x_1 + 80x_2 \\ &5x_1 + 2x_2 \leq 20 \\ &(-10x_1) + (-12x_2) \leq -90 \end{aligned}$$

Linear
Programming



Constraint
Satisfaction

Optimization

Introduction to
Artificial Intelligence
with Python

Harvard · CS50-AI | Introduction to Artificial Intelligence with Python (2020)

CS50-AI (2020) · 课程资料包 @ShowMeAI



视频

中英双语字幕



课件

一键打包下载



笔记

官方笔记翻译



代码

作业项目解析



视频 · B 站 [扫码或点击链接]

<https://www.bilibili.com/video/BV1AQ4y1y7wy>



课件 & 代码 · 博客 [扫码或点击链接]

<http://blog.showmeai.tech/harvard-cs50-ai>

深度优先
广度优先

贝叶斯

聚类

马尔可夫

A* 算法

自然语言处理

RNN

tensorflow

反向传播

SVM

强化学习

监督学习

无监督

语法解析

搜索

神经网络

CNN

Awesome AI Courses Notes Cheatsheets 是 [ShowMeAI](#) 资料库的分支系列，覆盖最具知名度的 **TOP50+** 门 AI 课程，旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

点击 课程名称，跳转至课程 **资料包** 页面，**一键下载** 课程全部资料！

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n
# Awesome AI Courses Notes Cheatsheets · 持续更新中			
知识图谱	图机器学习	深度强化学习	自动驾驶
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094



微信公众号

资料下载方式 2：扫码点击 **底部菜单栏**

称为 **AI 内容创作者**？回复 [添砖加瓦]