Introduction to
Artificial Intelligence
with Python

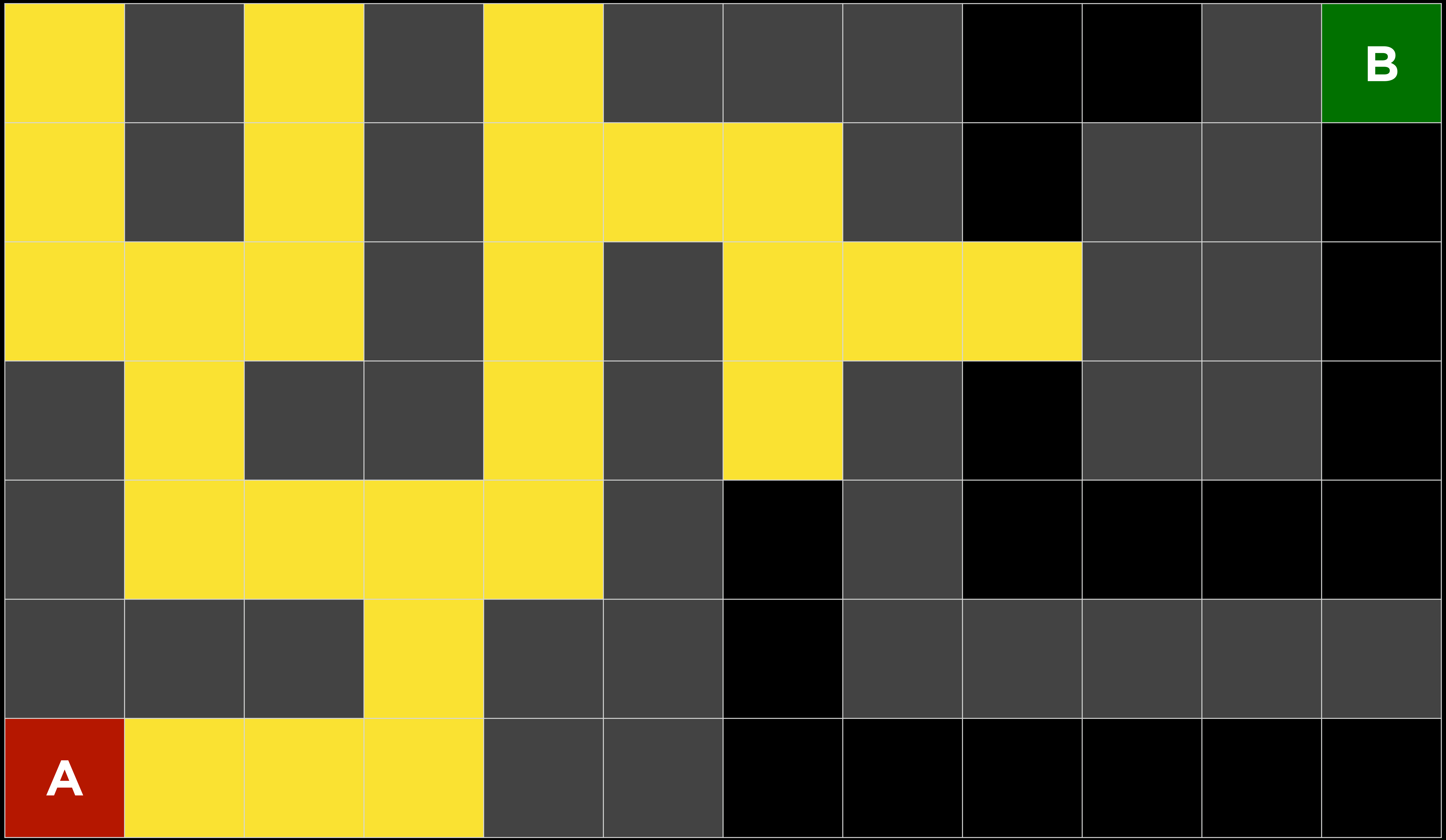
Optimization

optimization

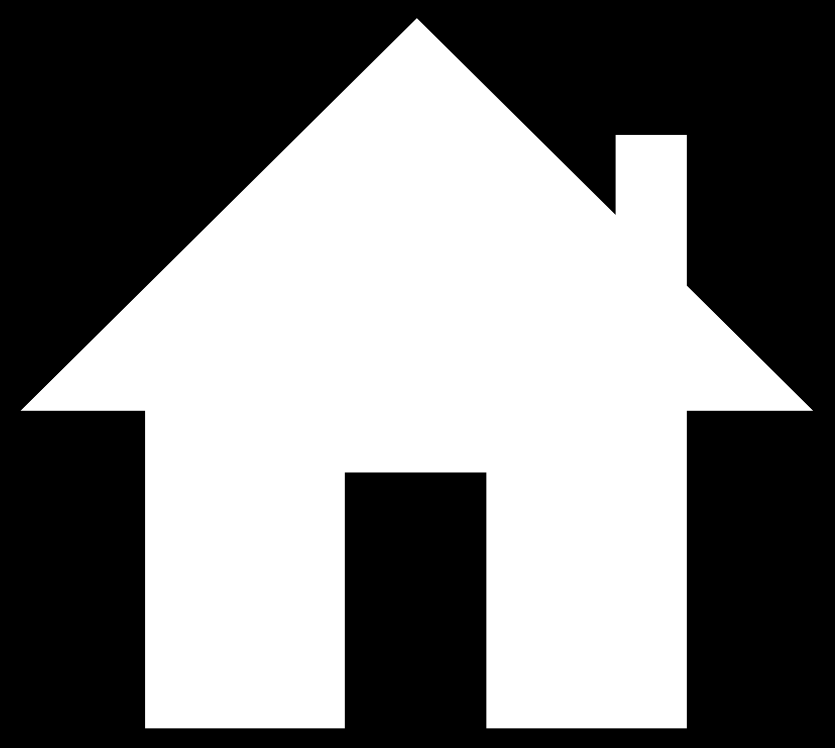
choosing the best option from a set of options

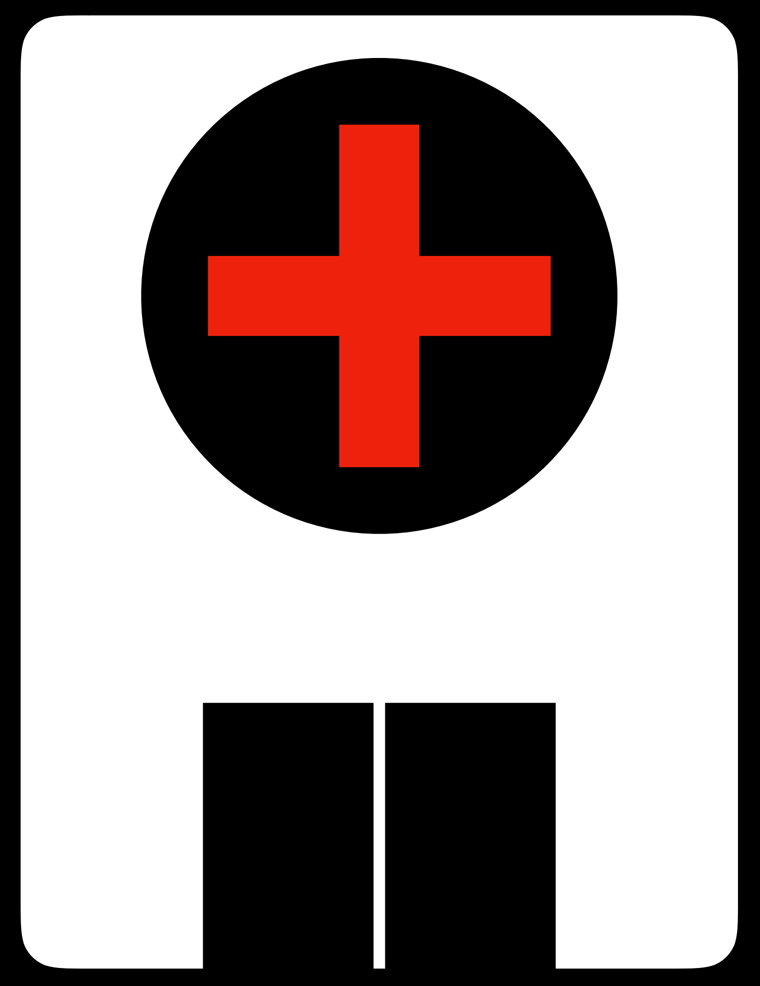
local search

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

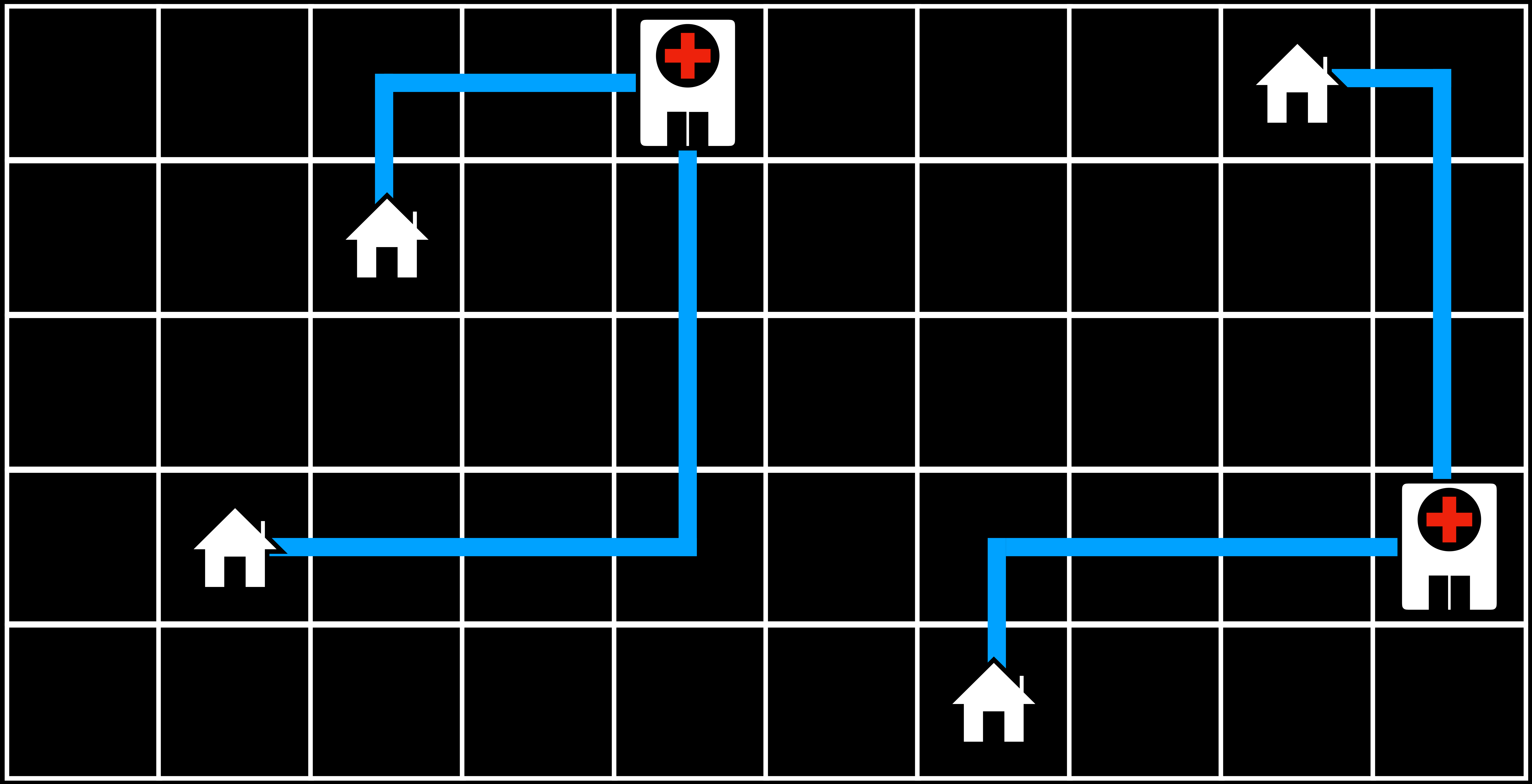

B
A


B





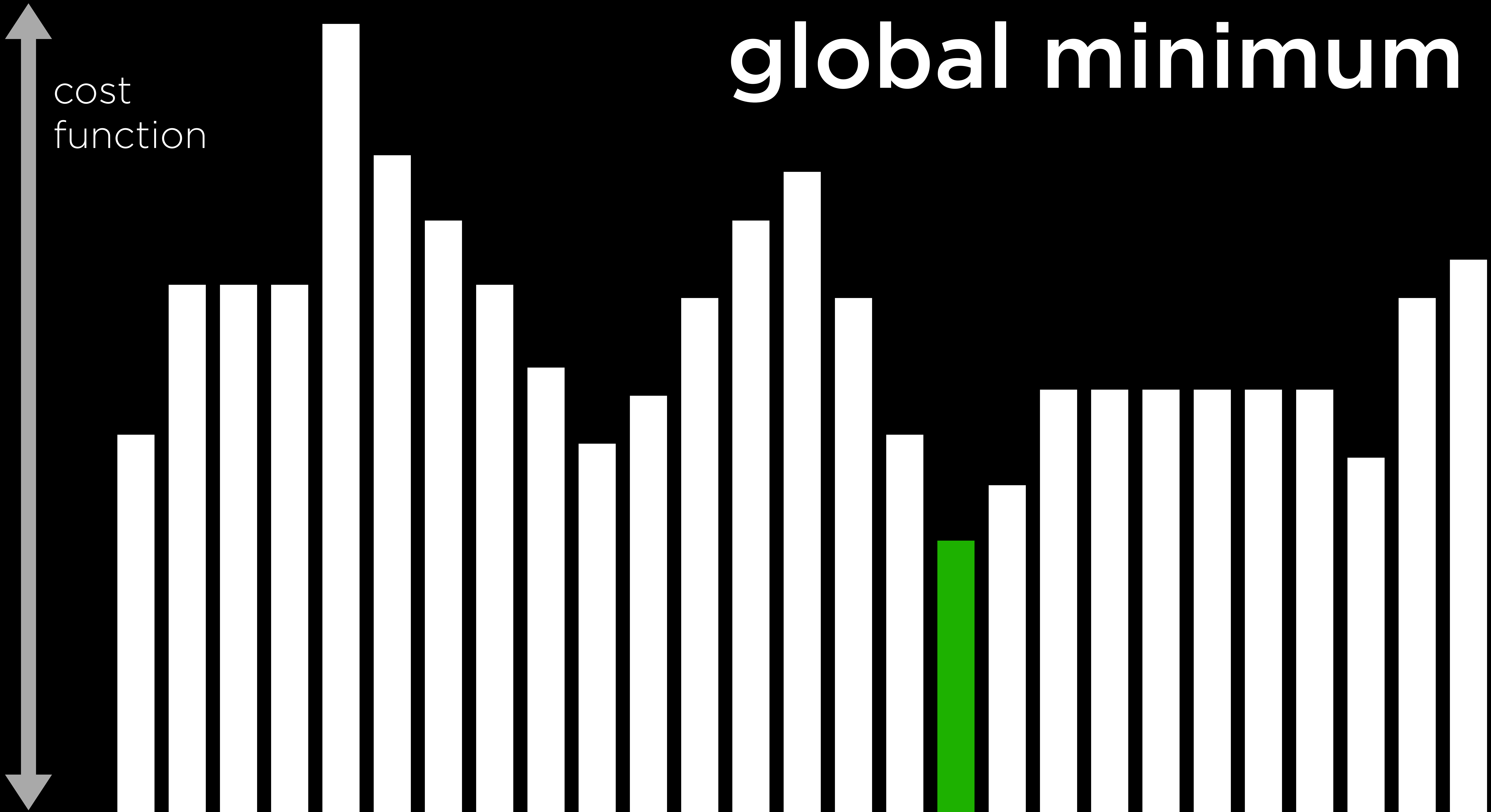
Cost: 17



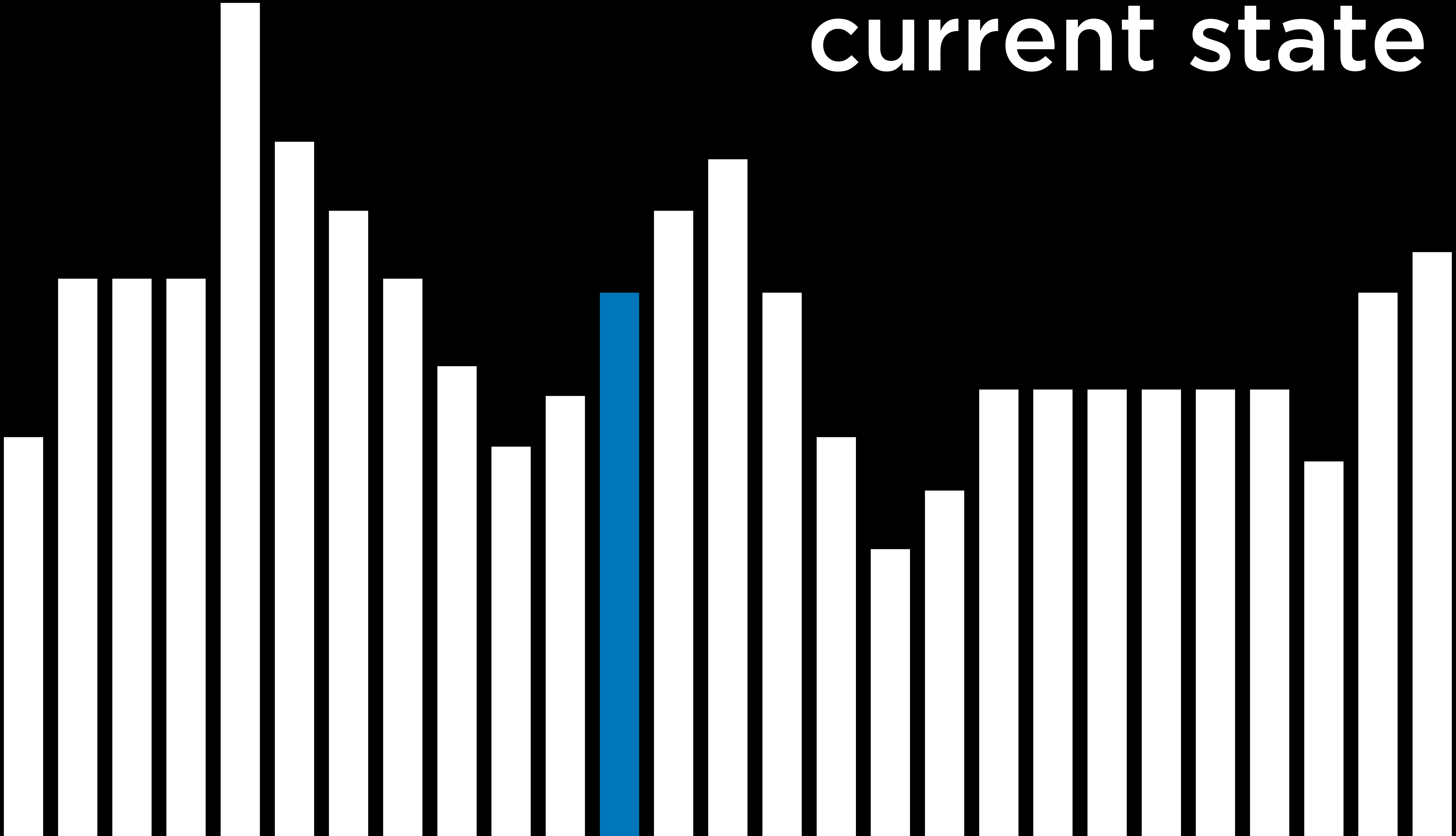
state-space landscape


global maximum
objective
function

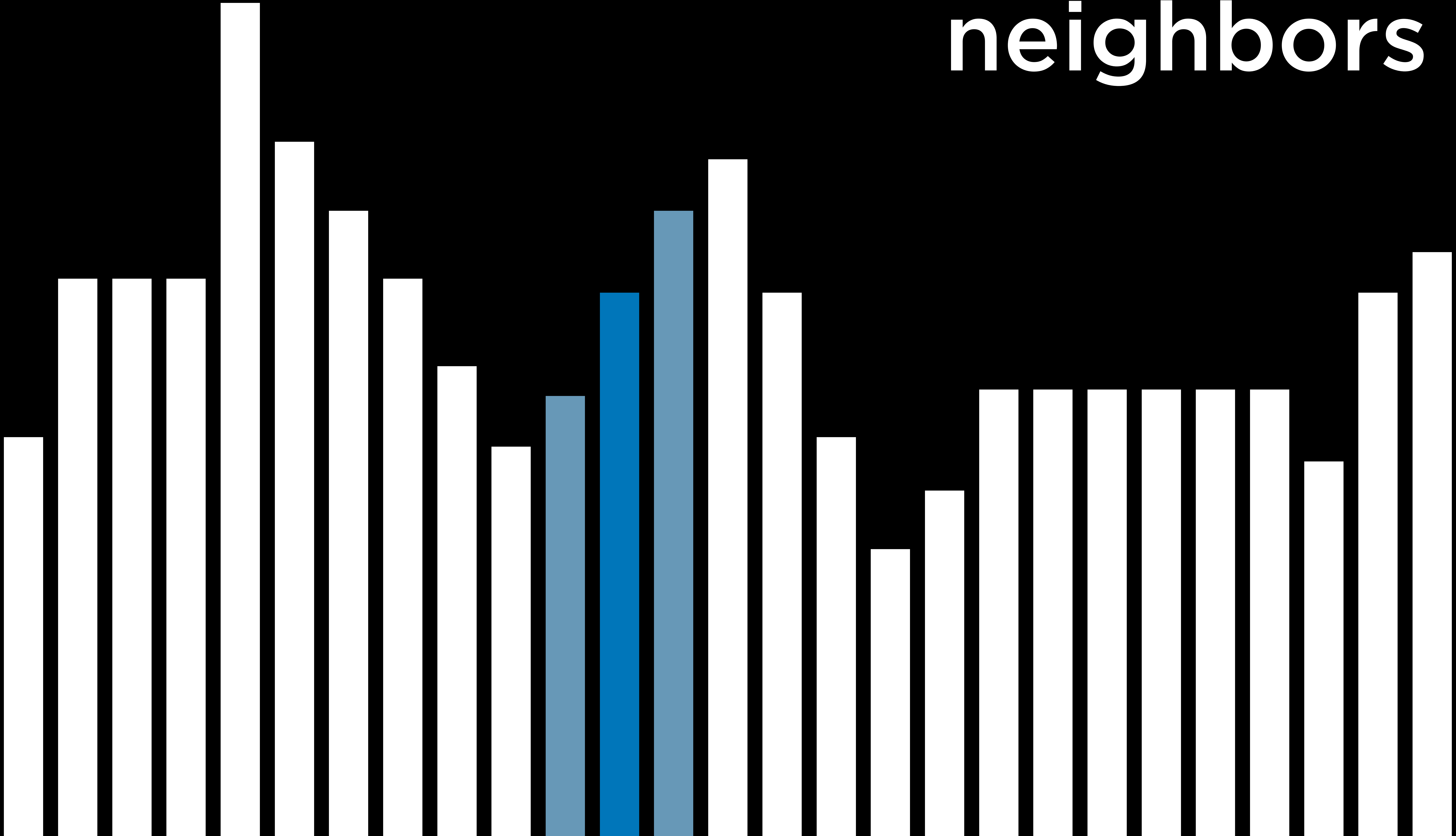
global minimum



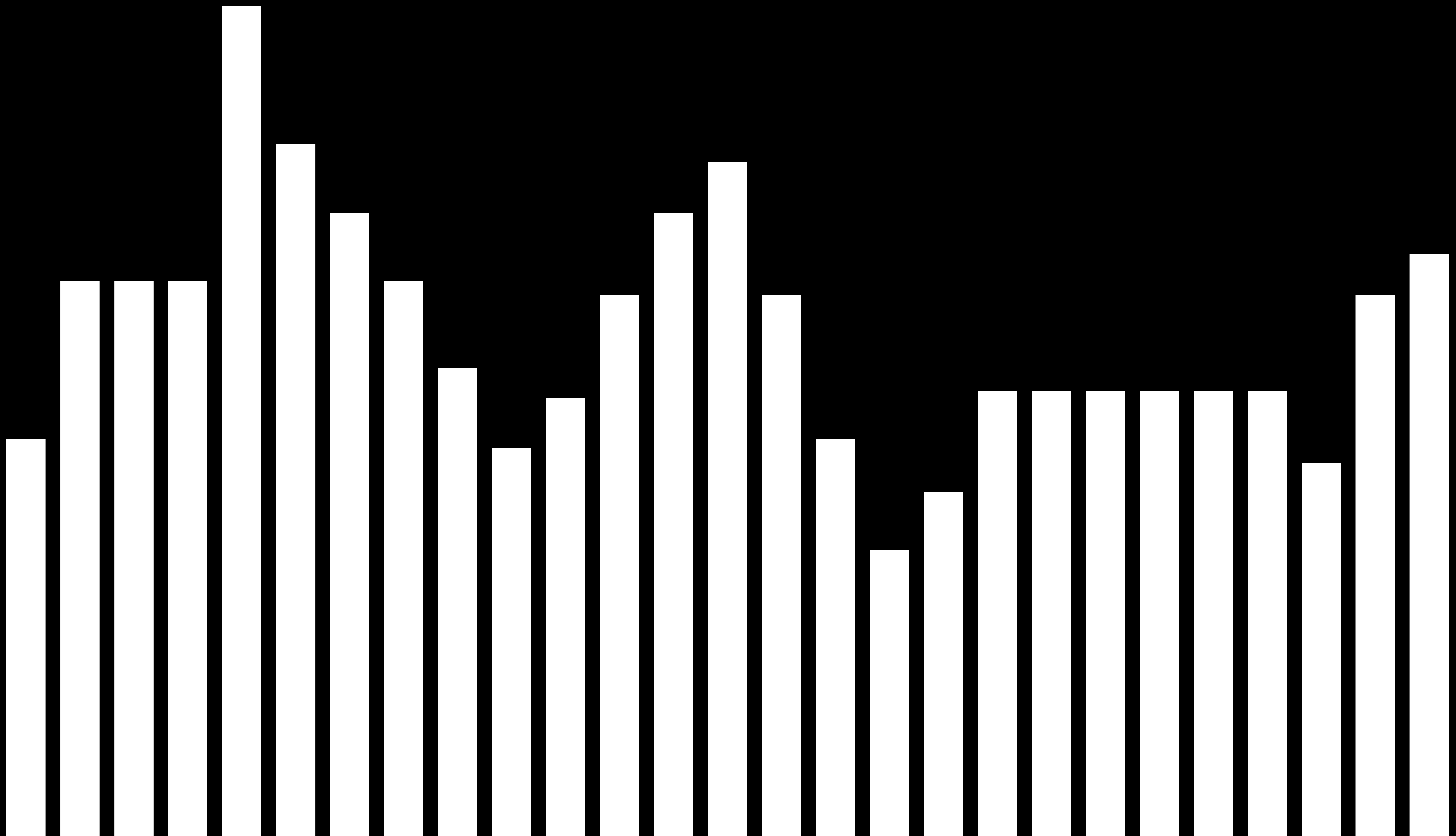
current state

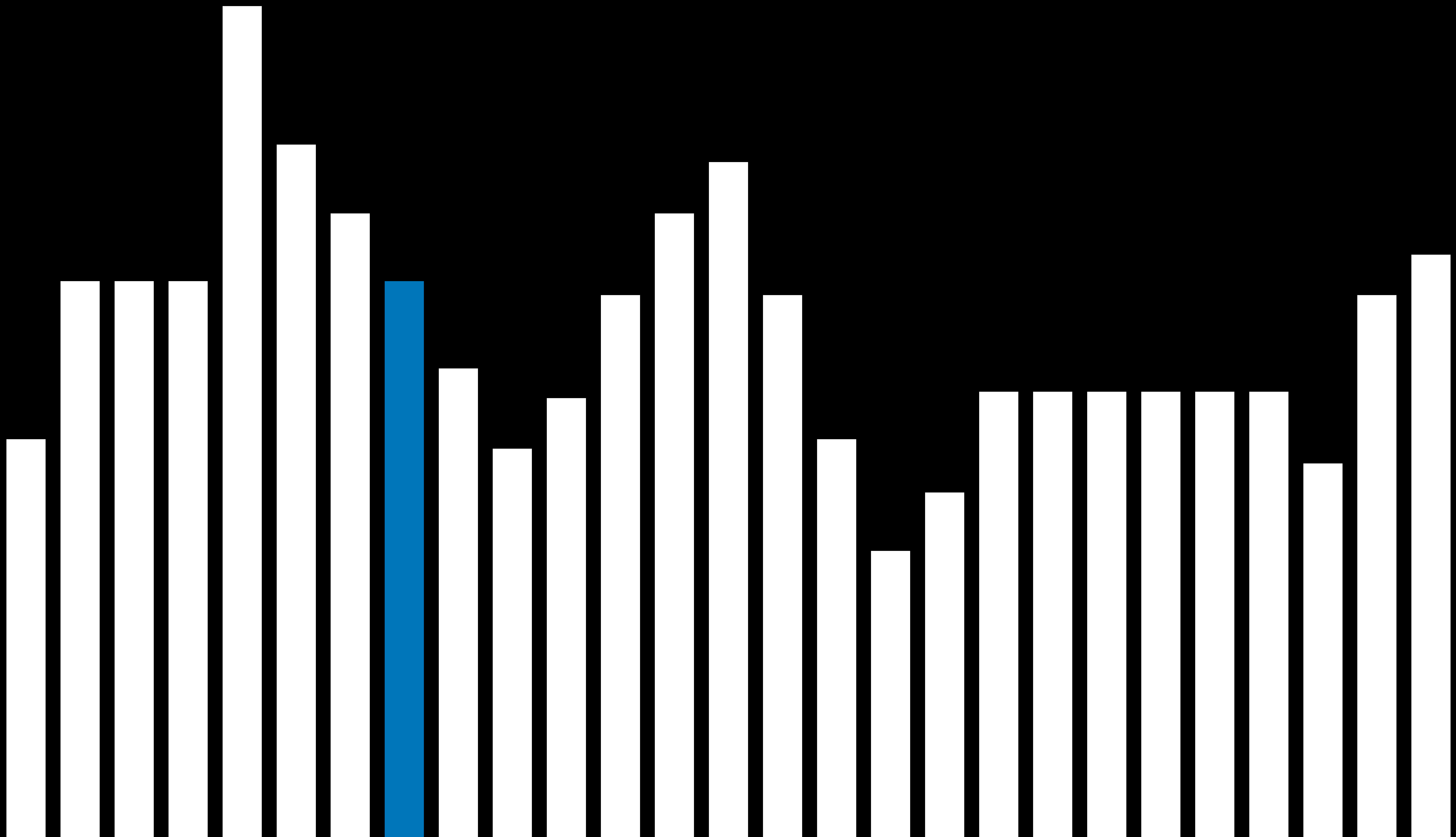


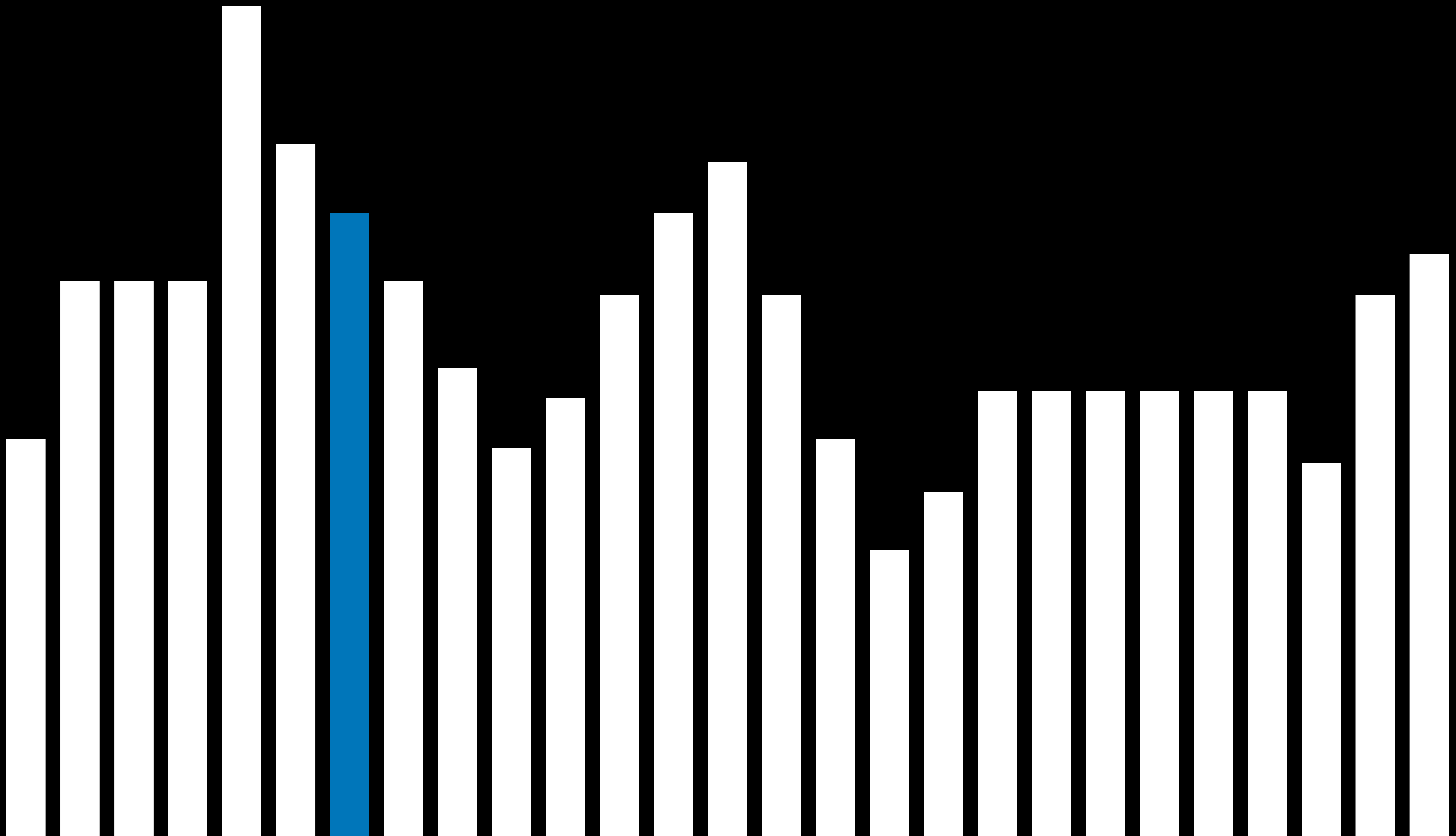
neighbors

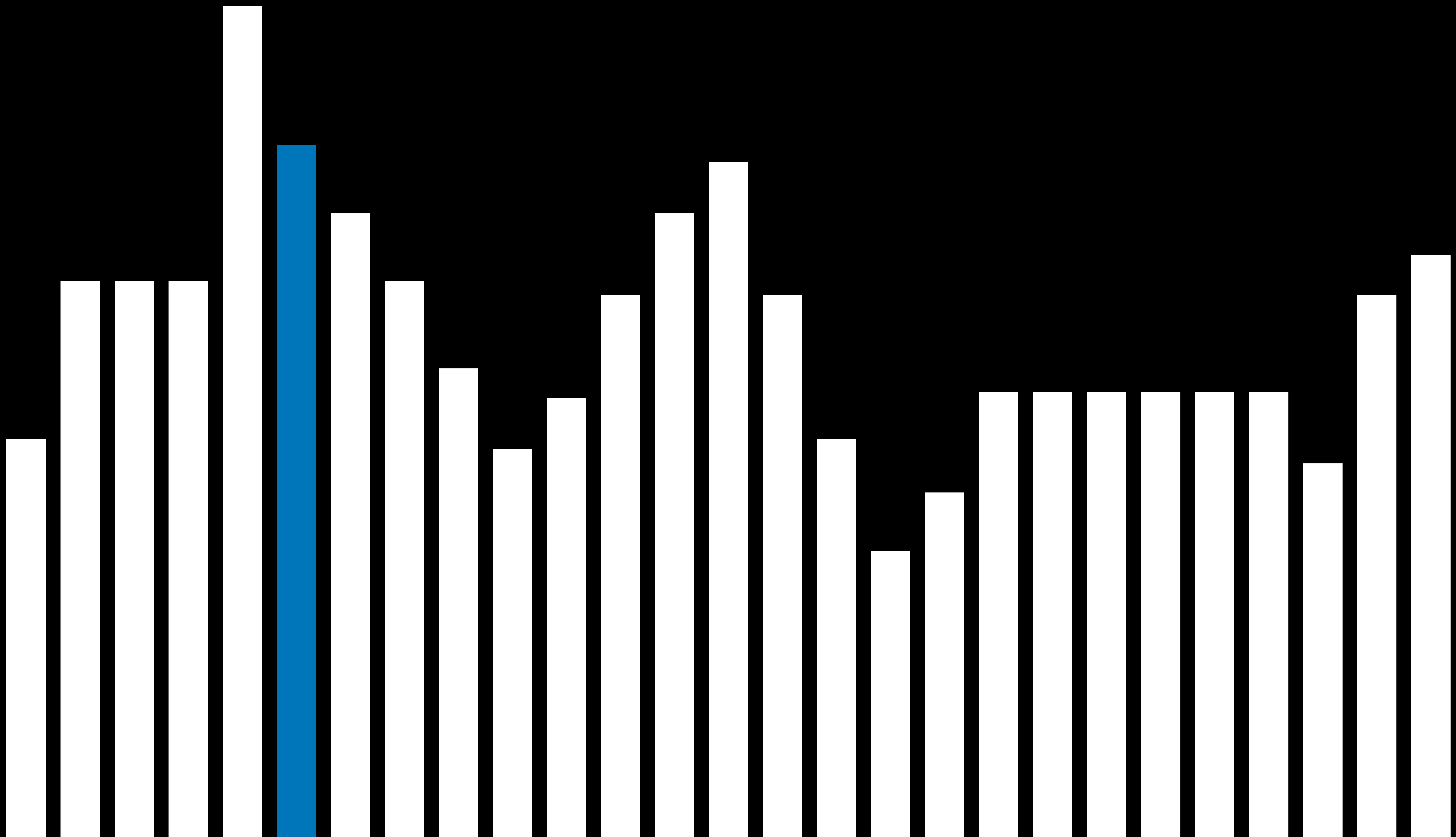


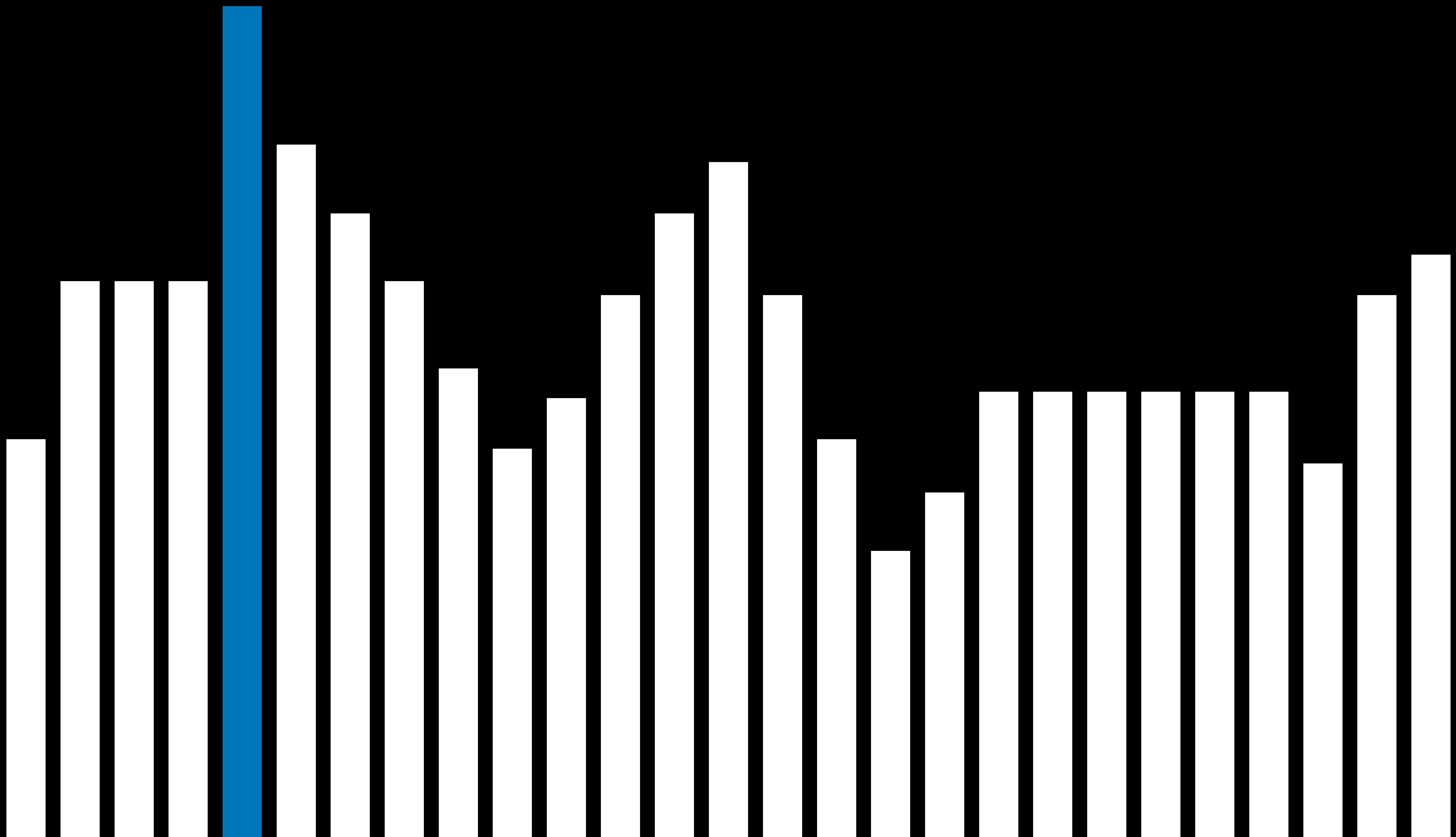
Hill Climbing

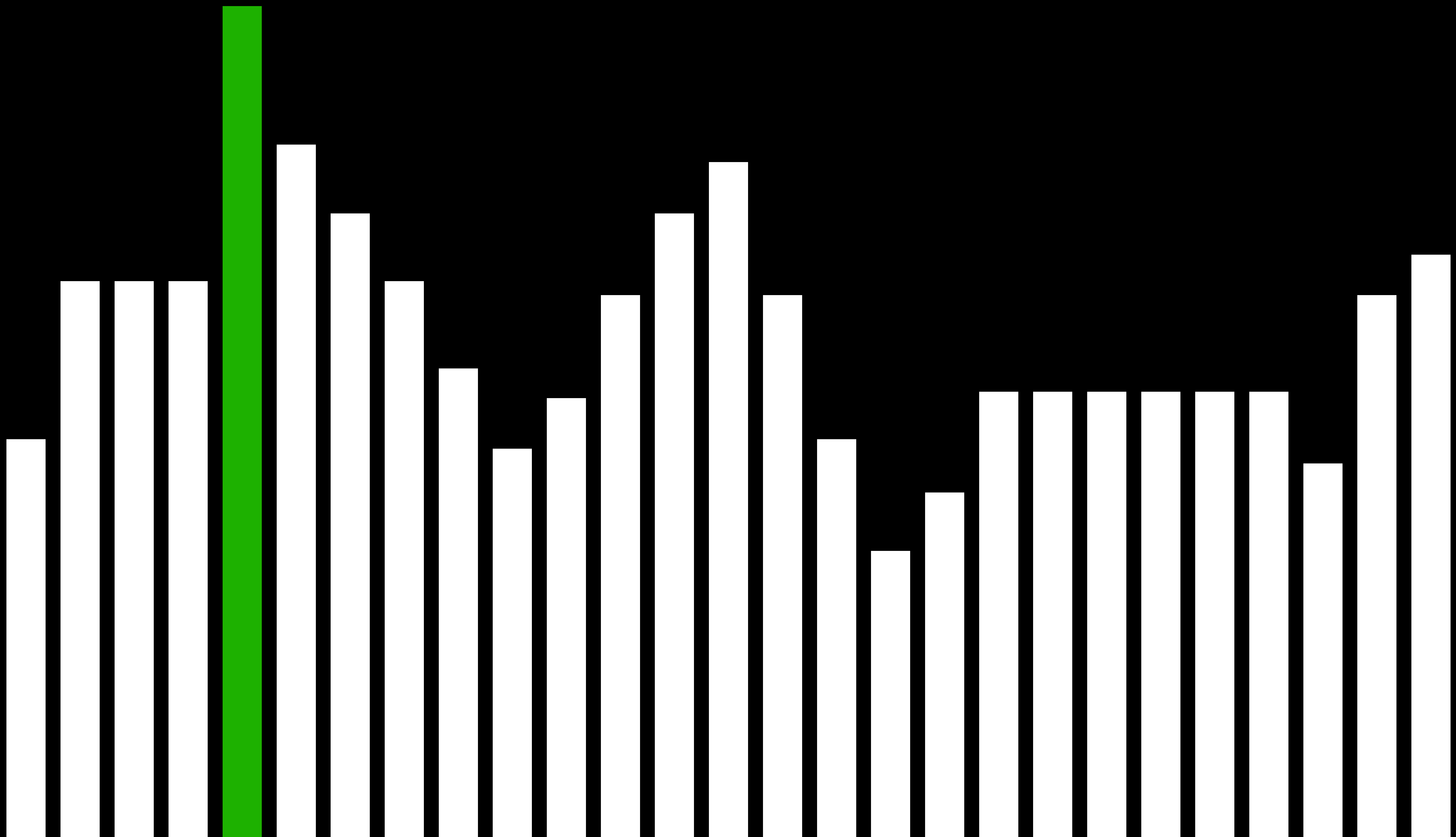


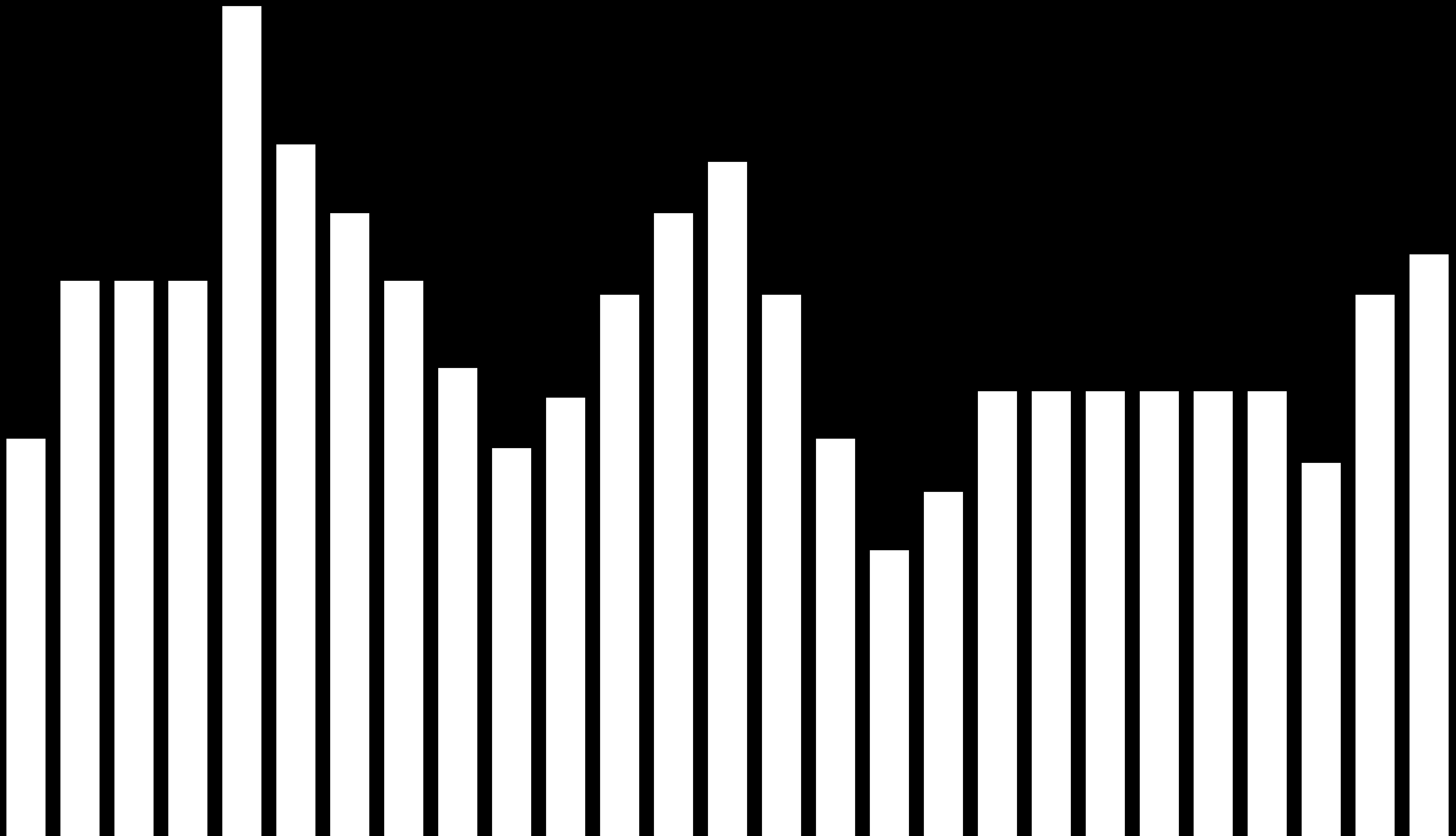


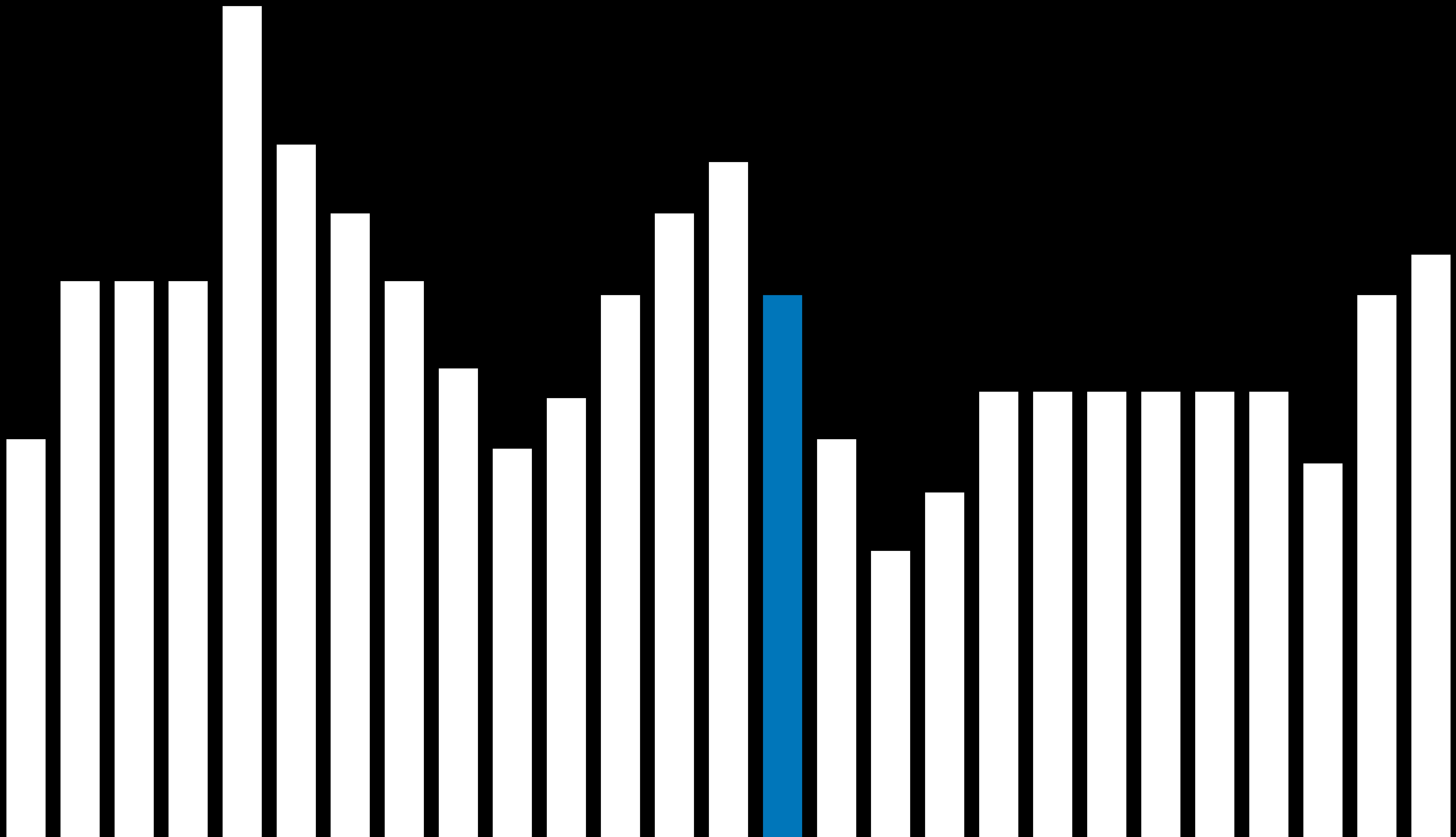


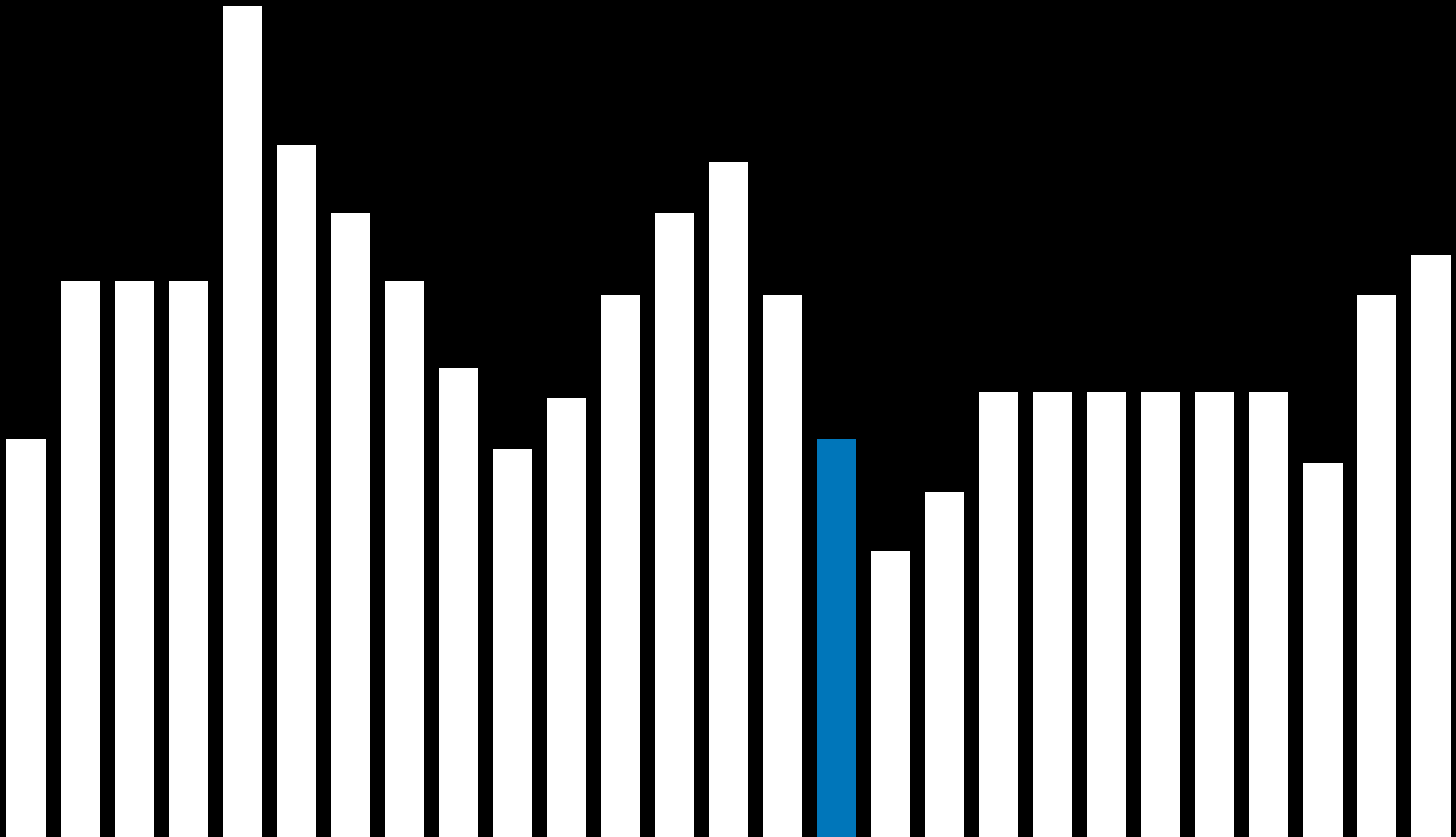


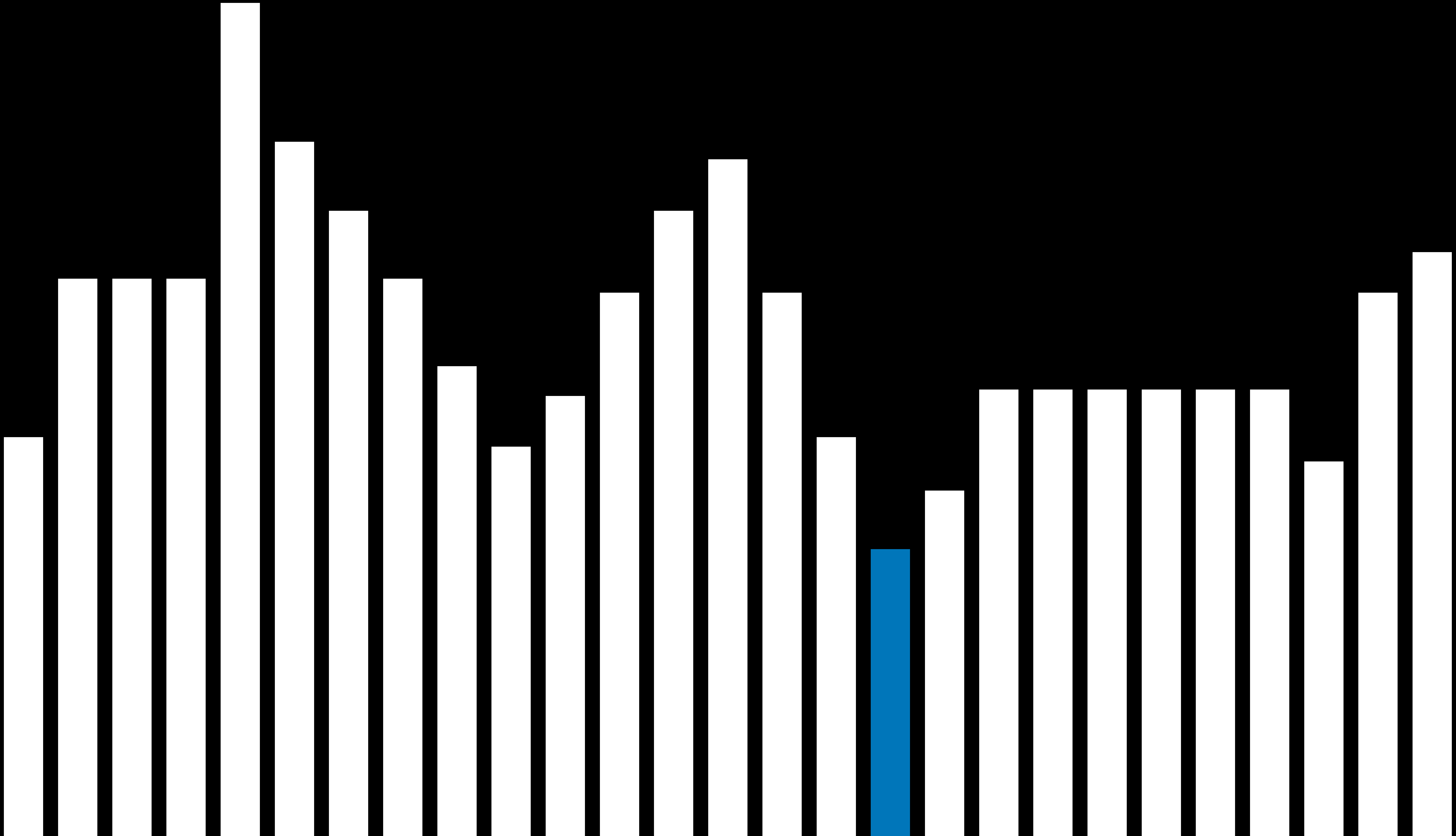


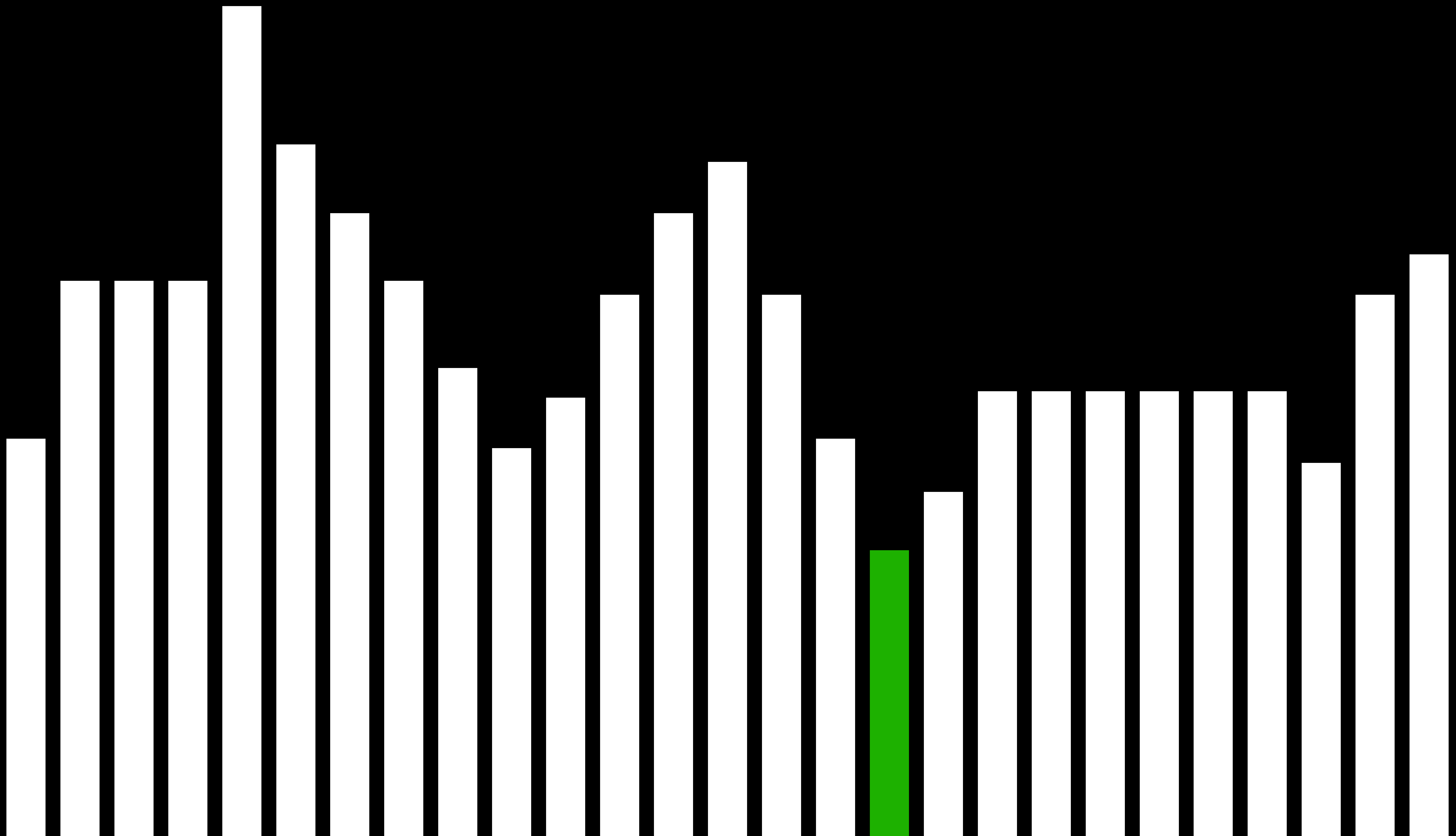








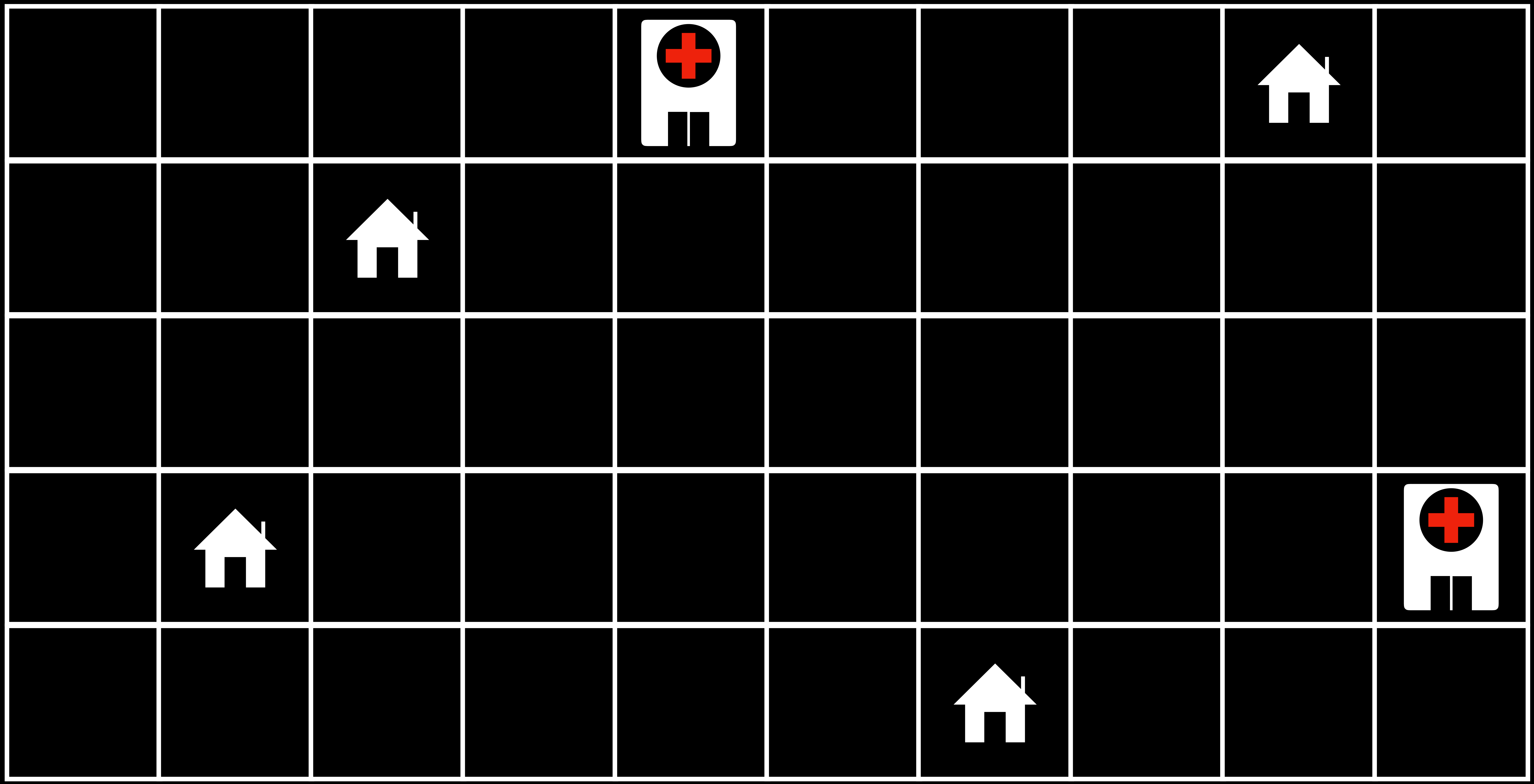




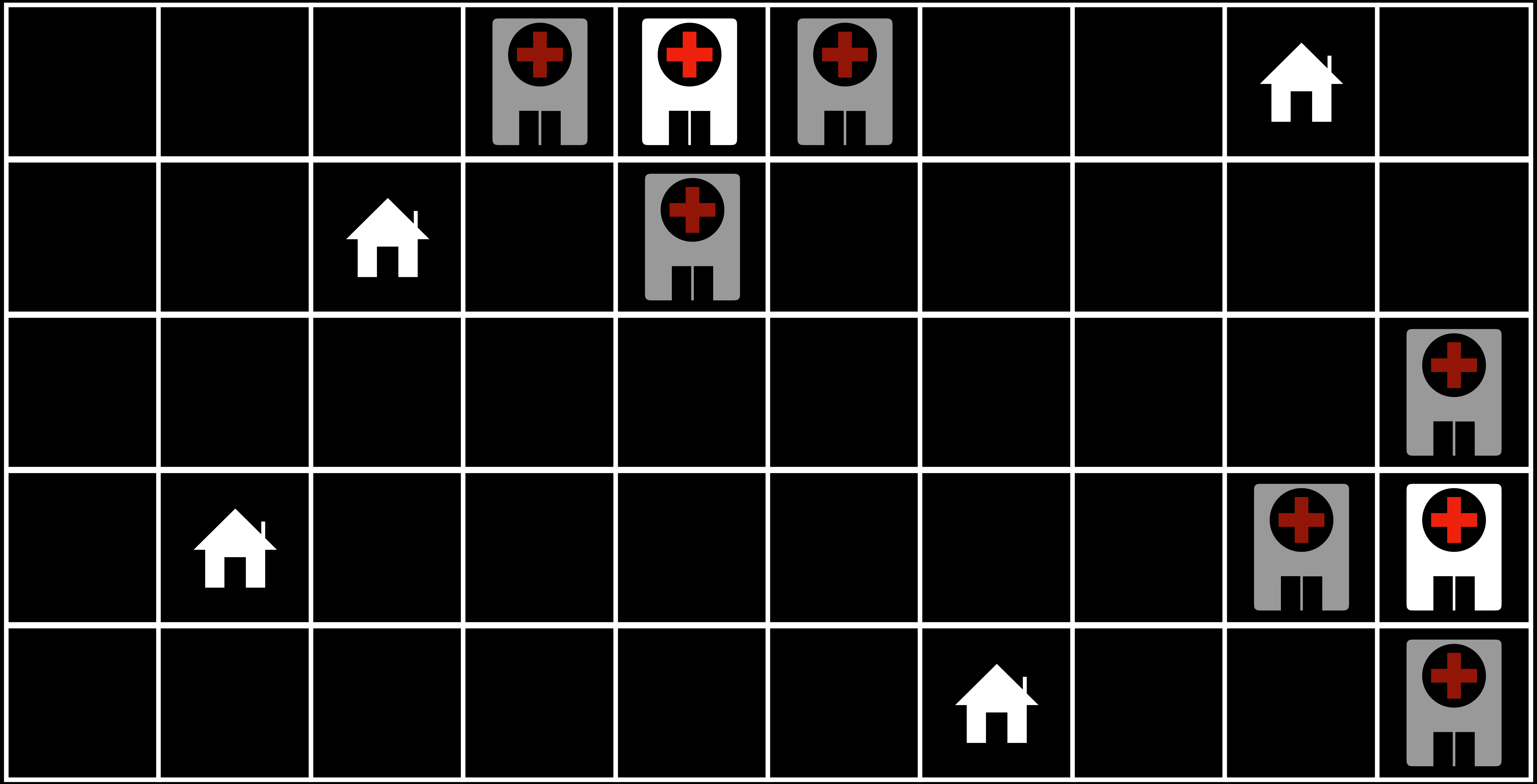
Hill Climbing

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*

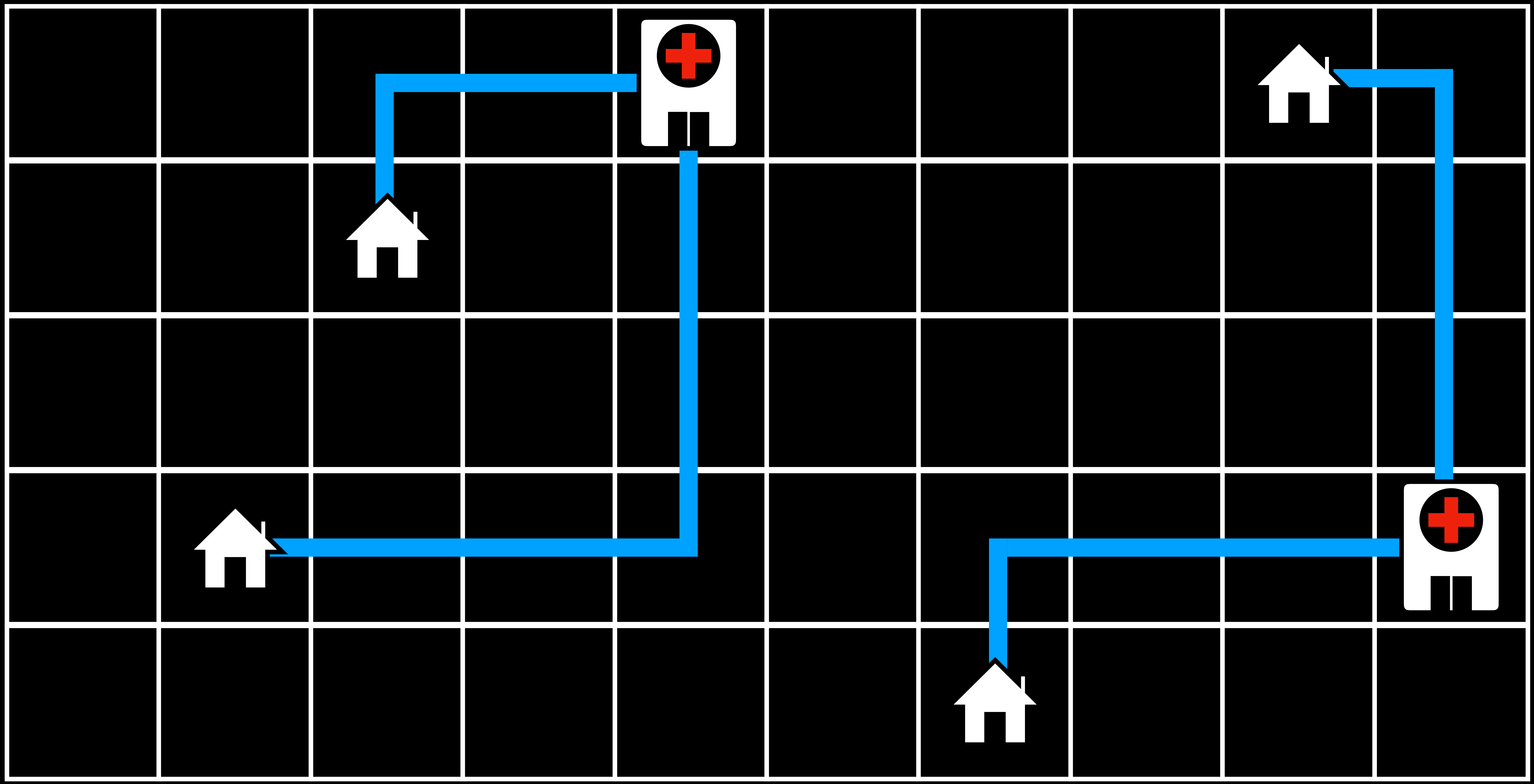
Cost: 17

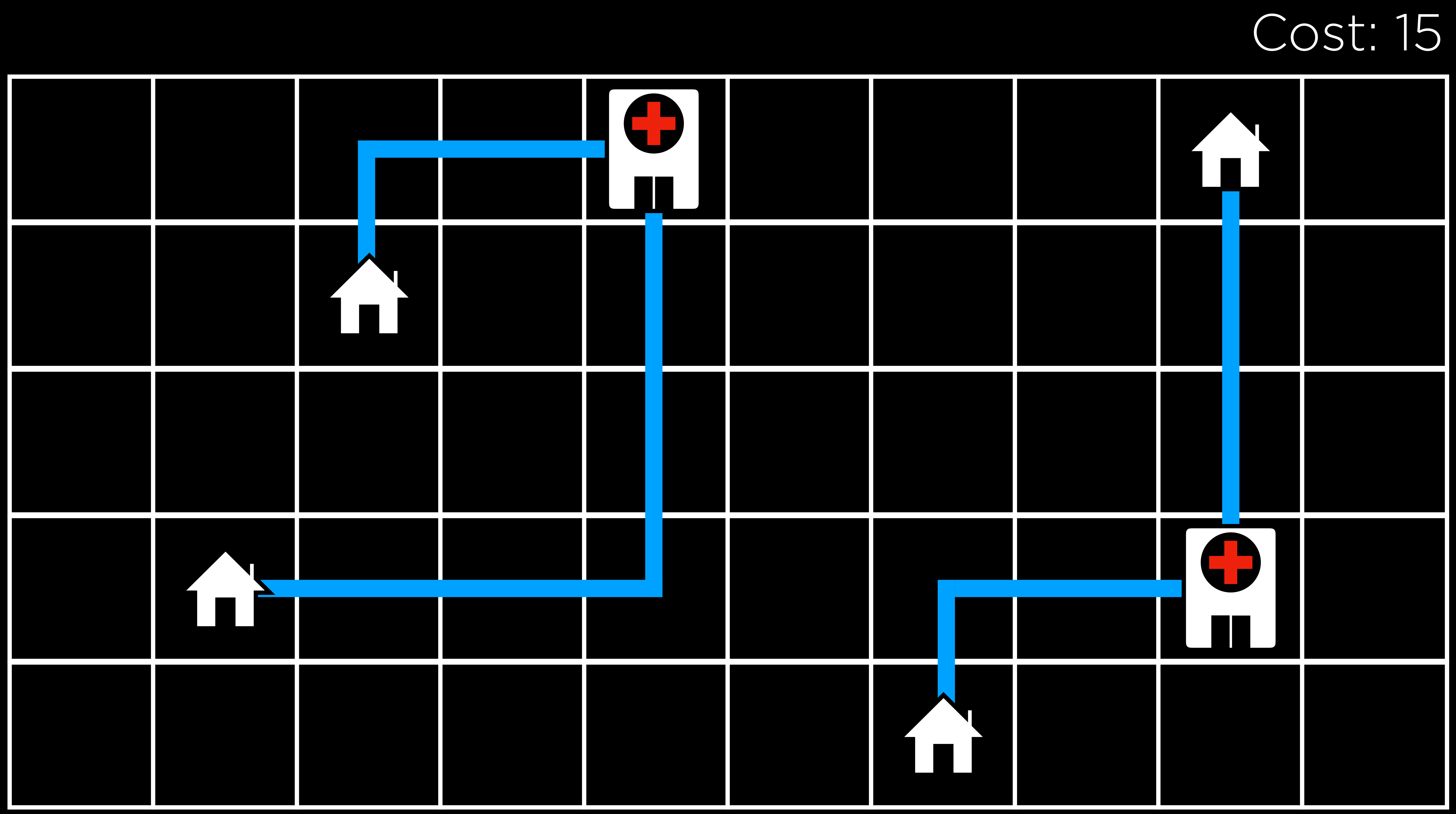


Cost: 17

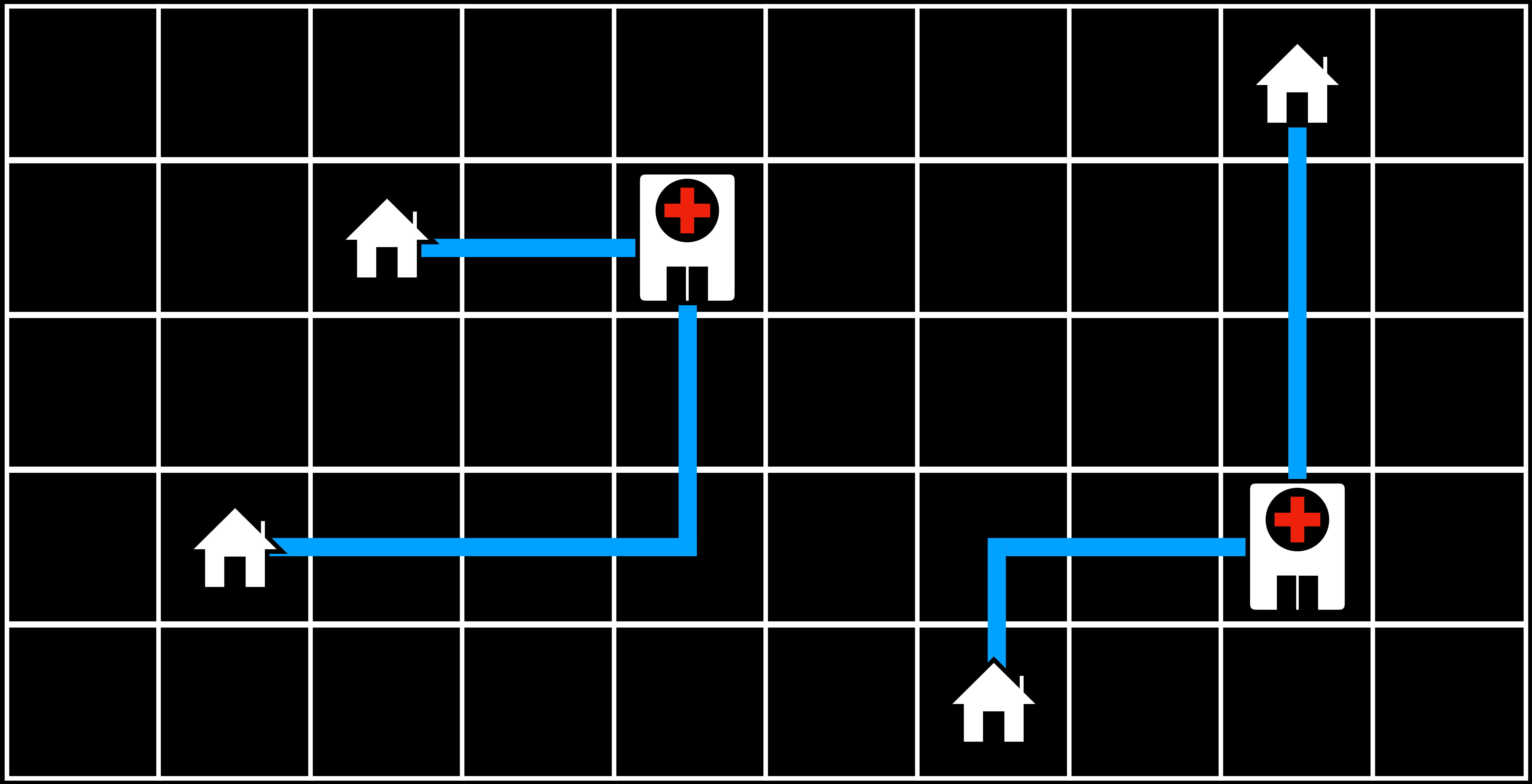


Cost: 17

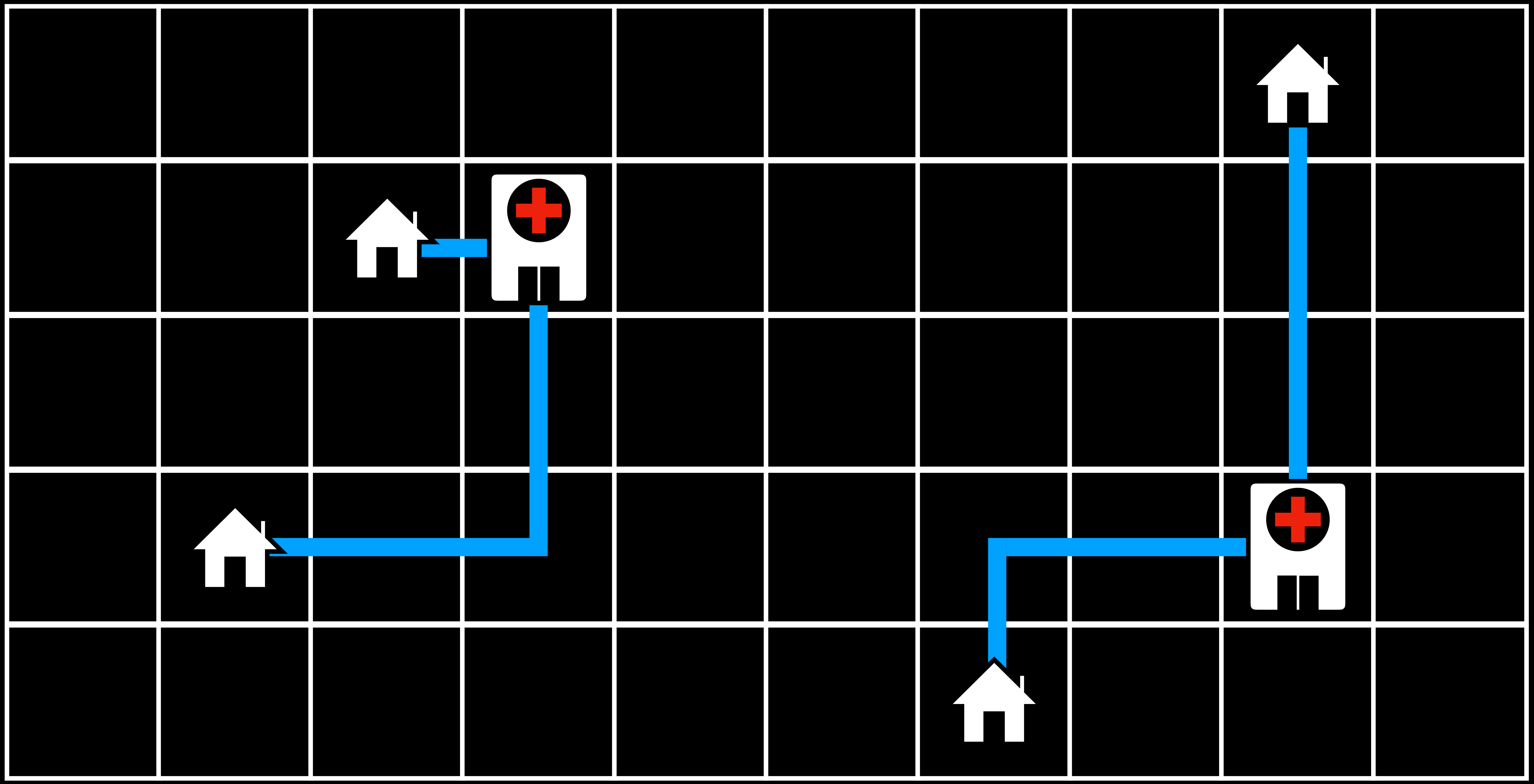


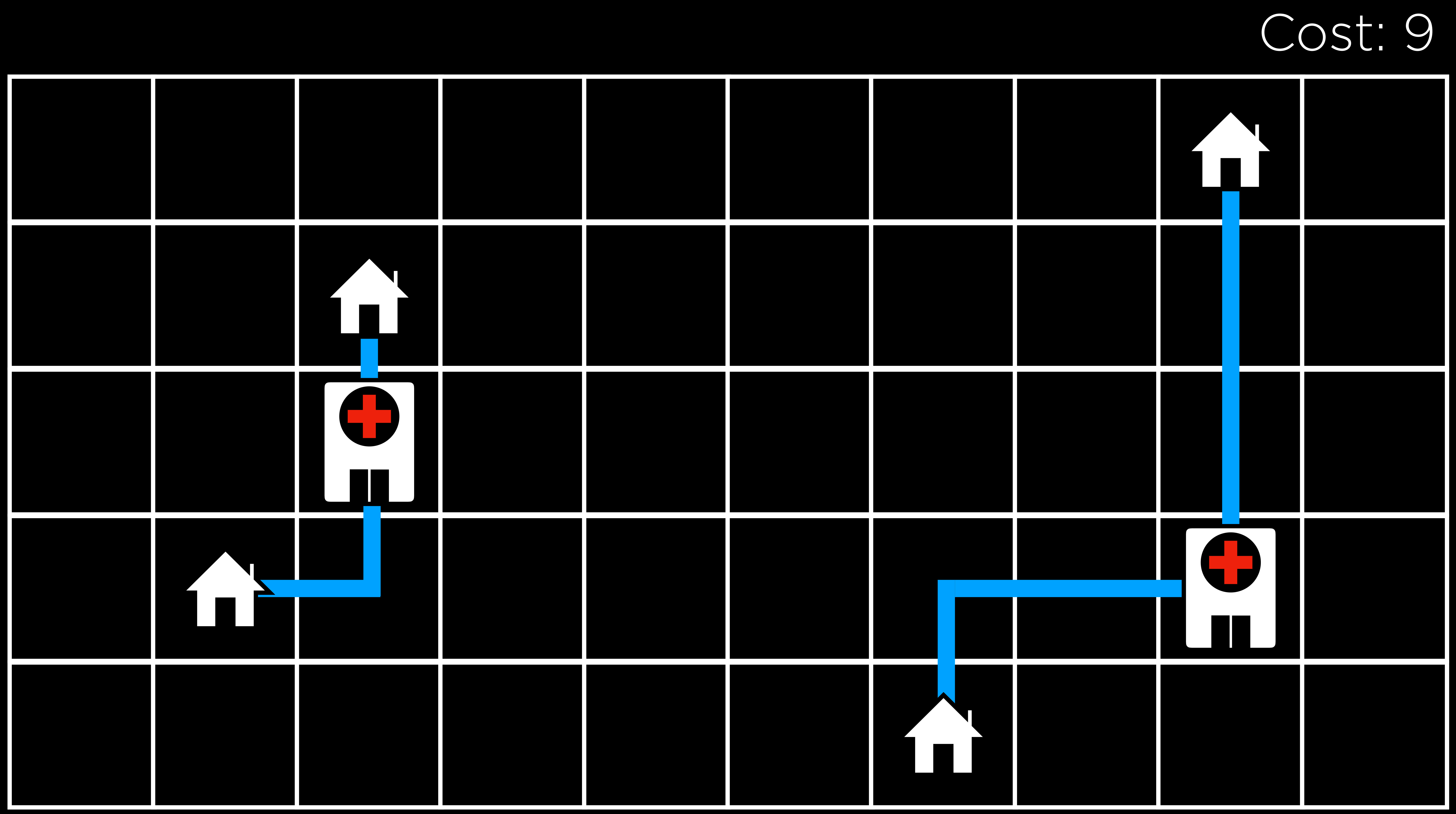

Cost: 15

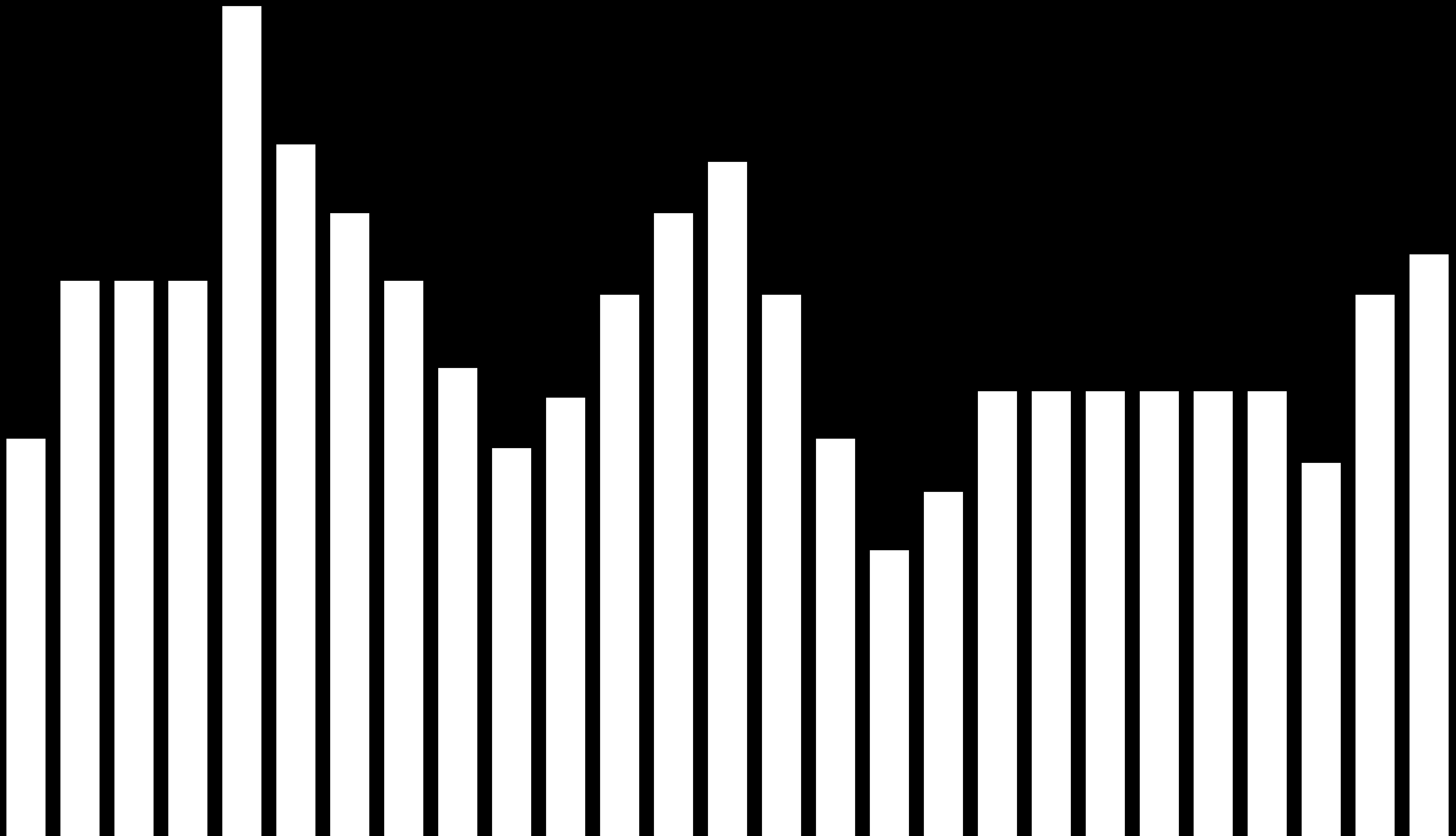
Cost: 13



Cost: 11

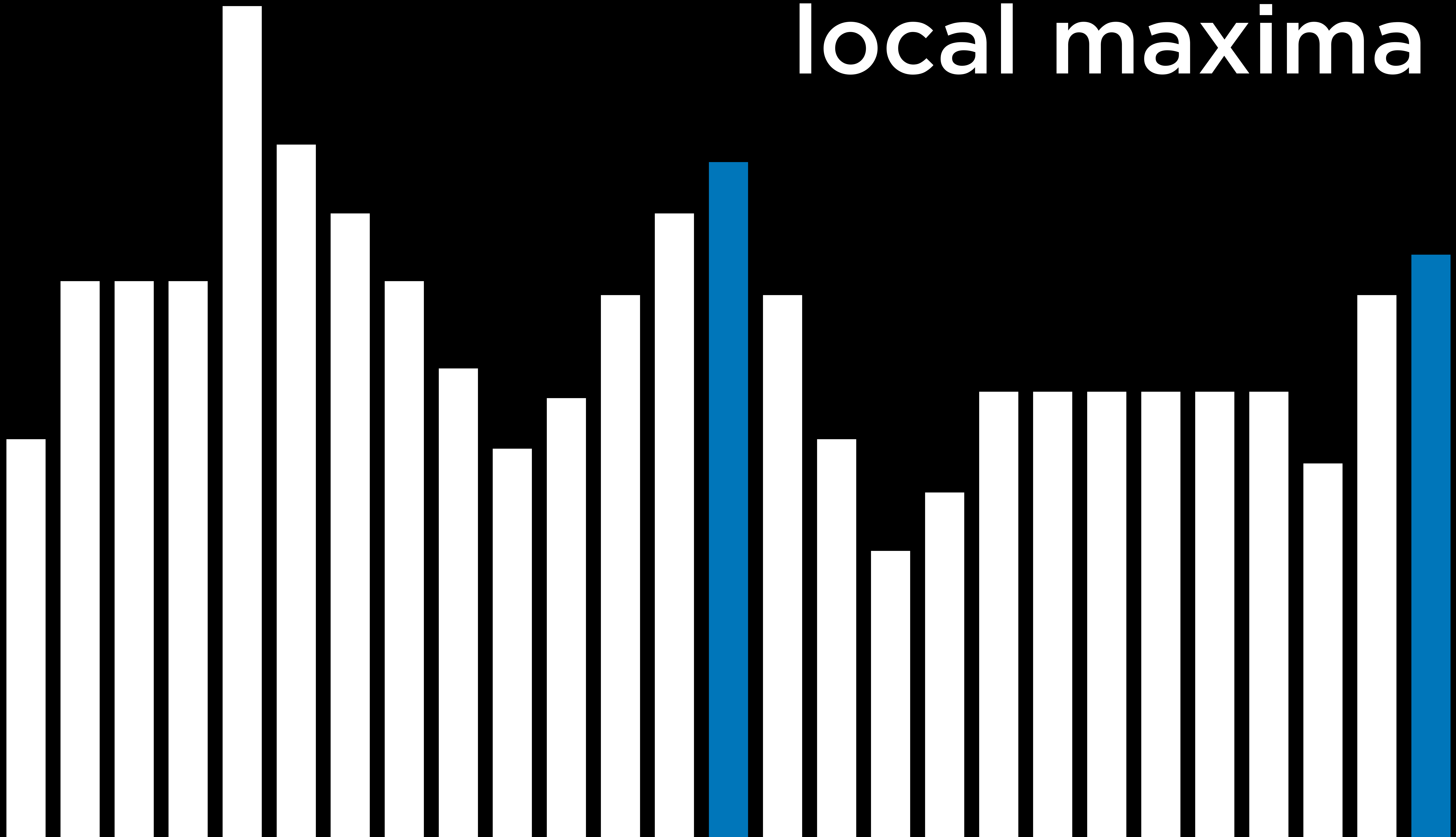


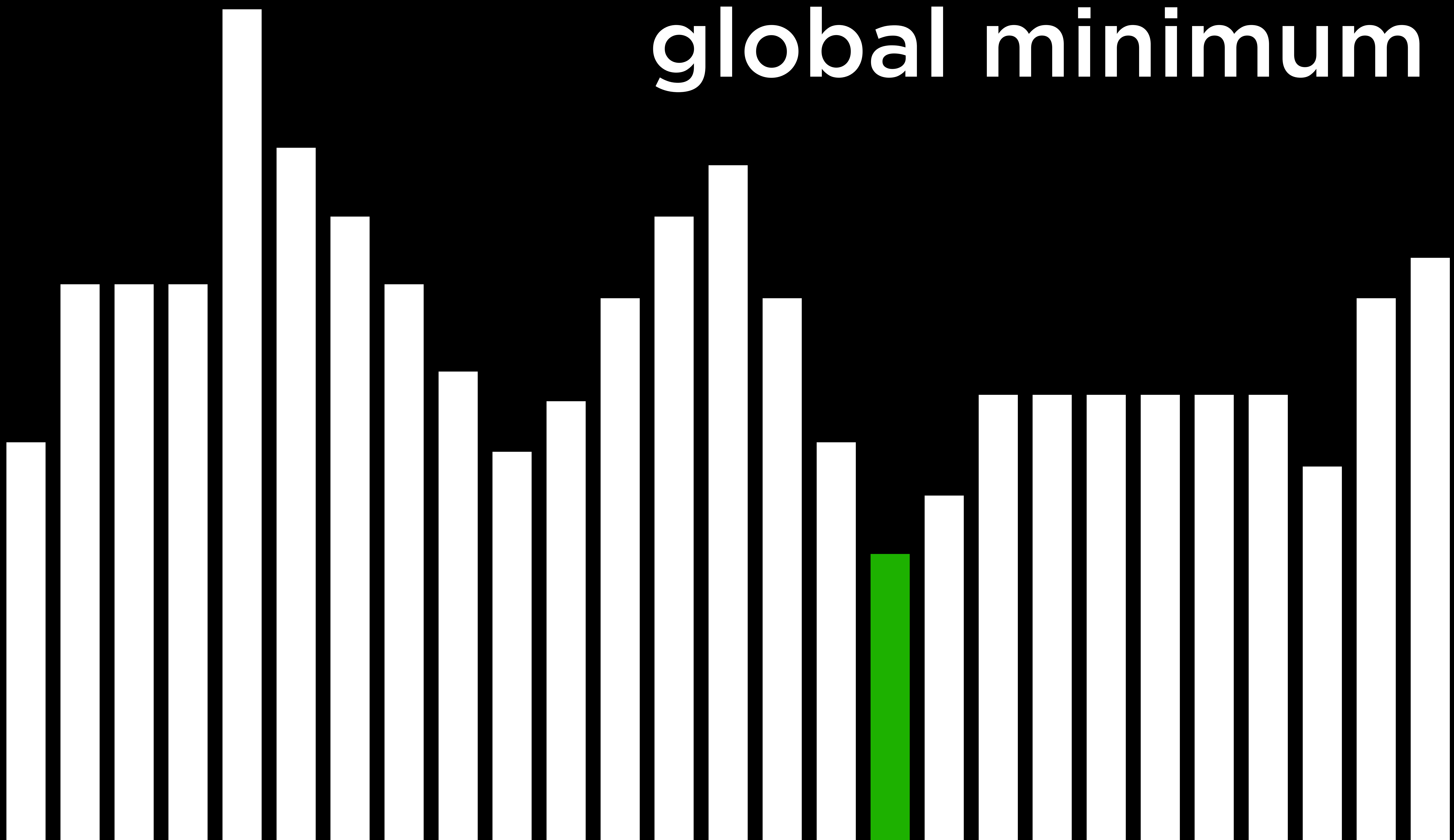

Cost: 9



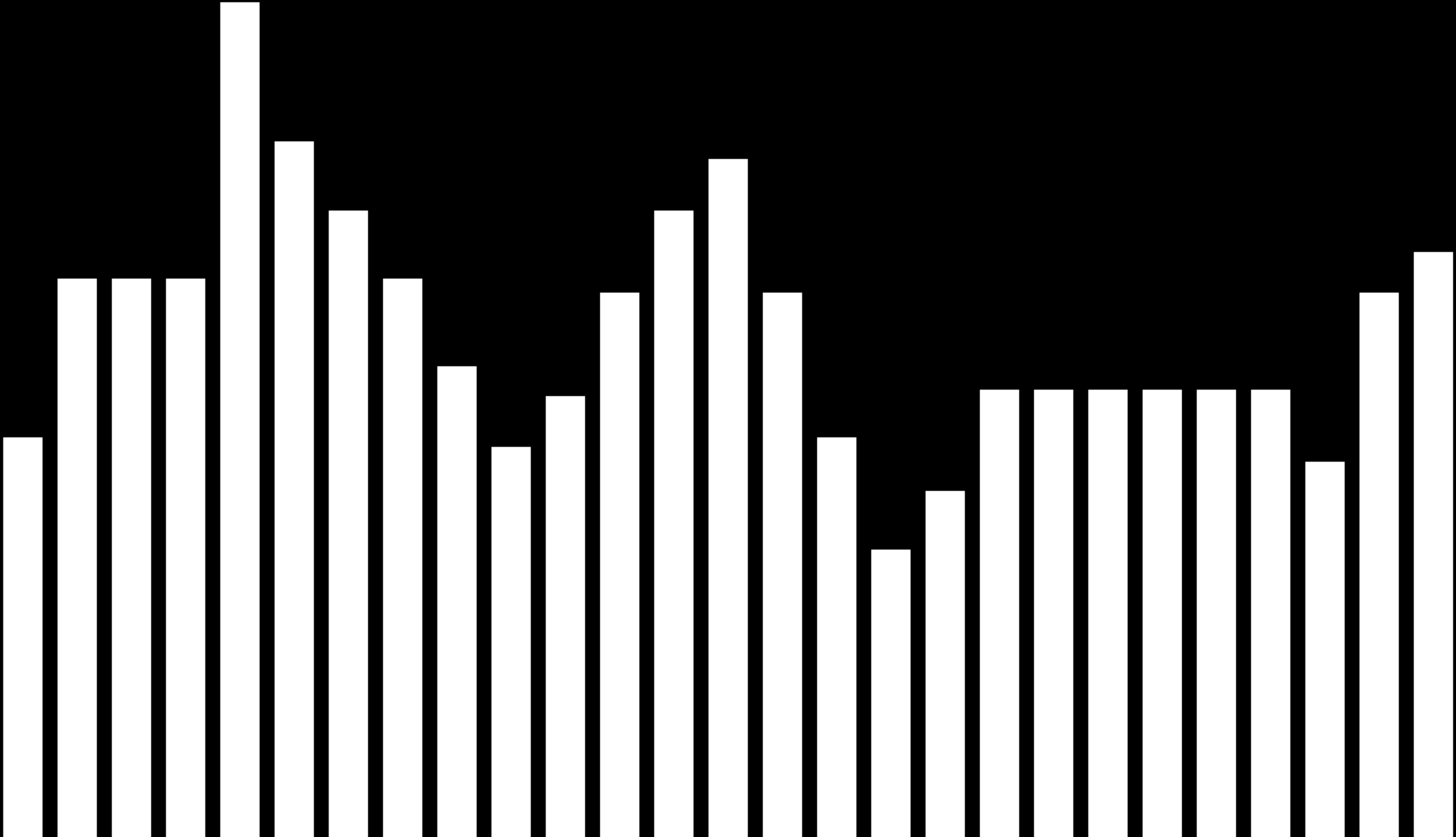

global maximum

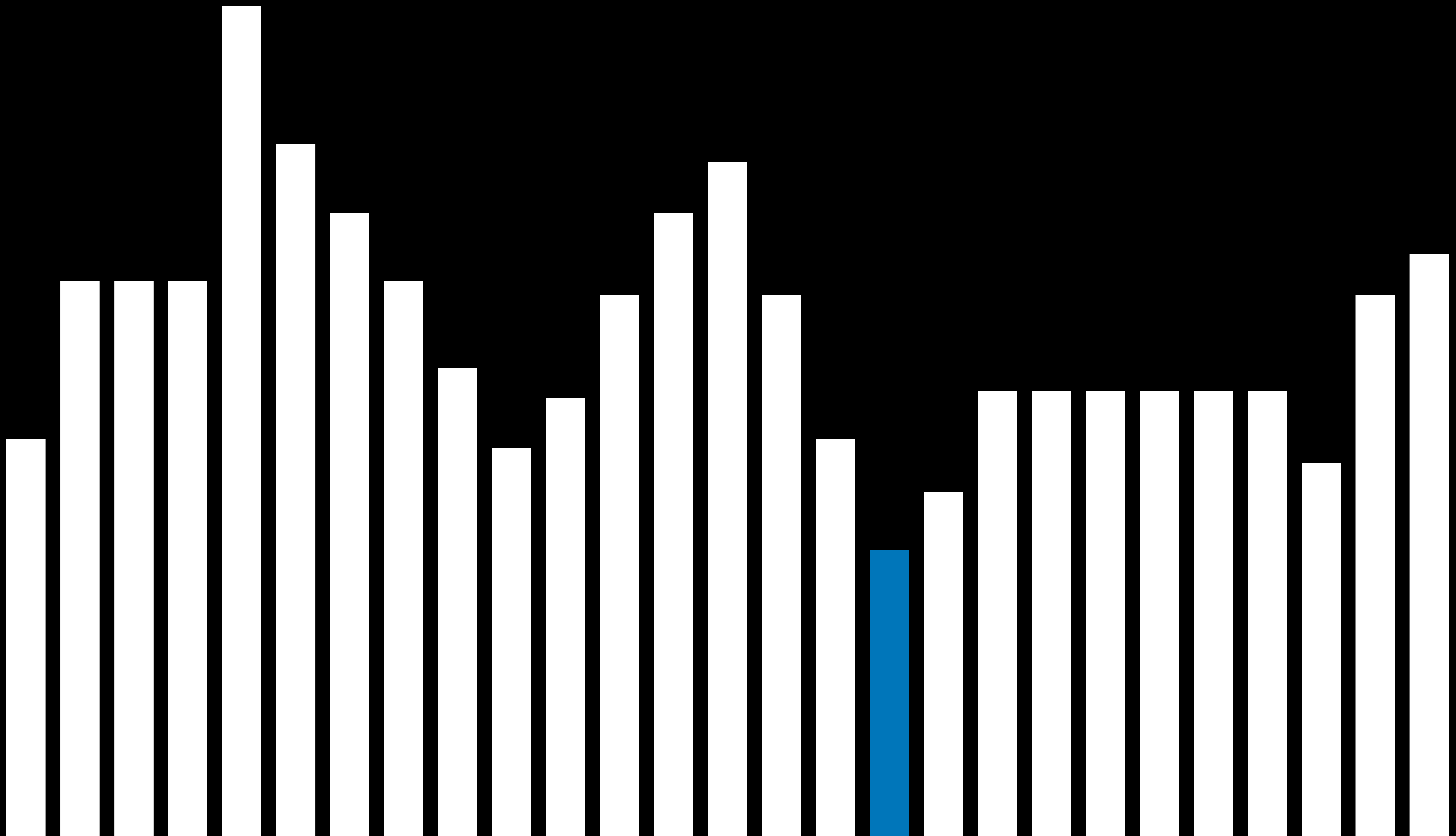
local maxima

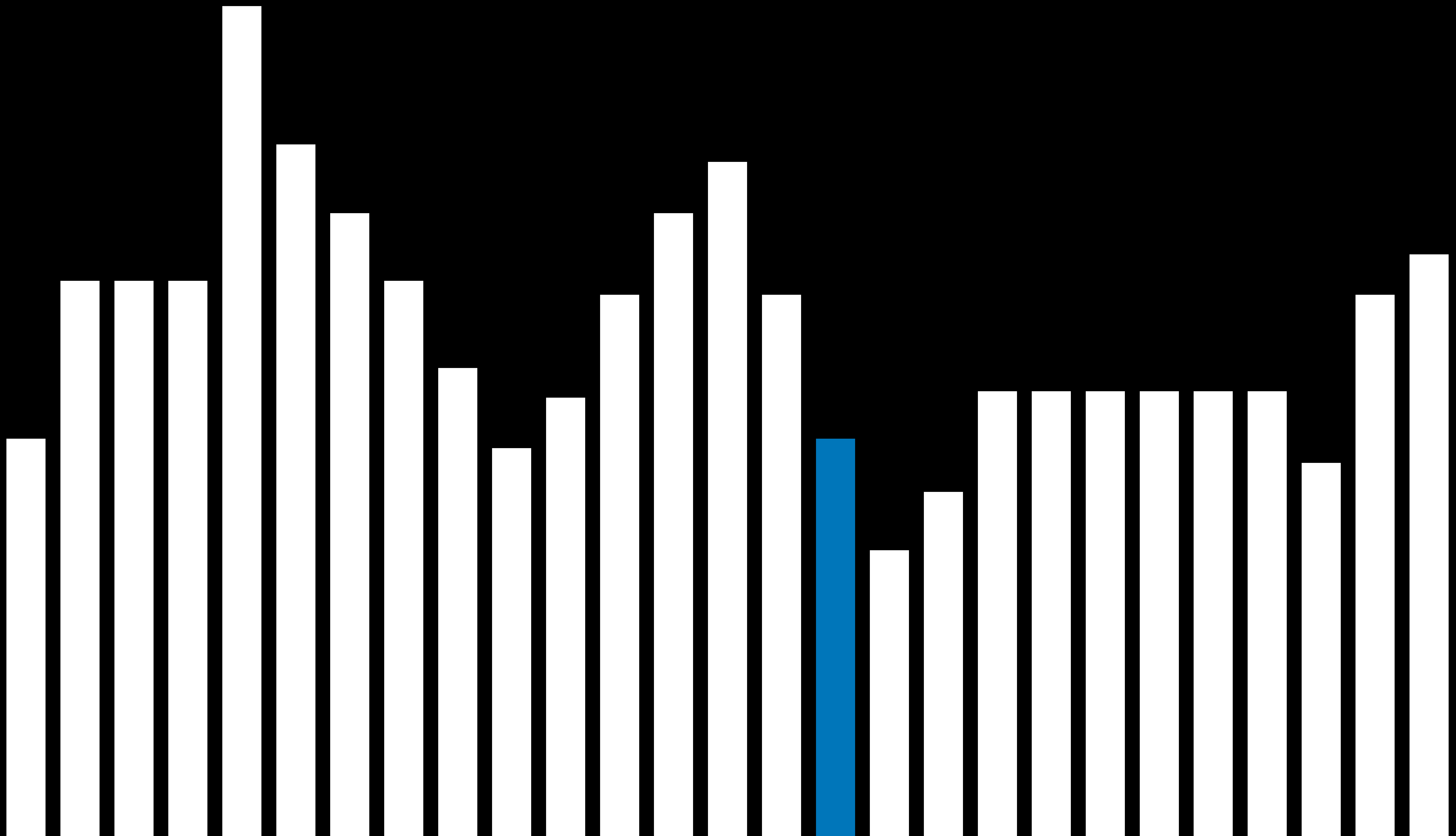


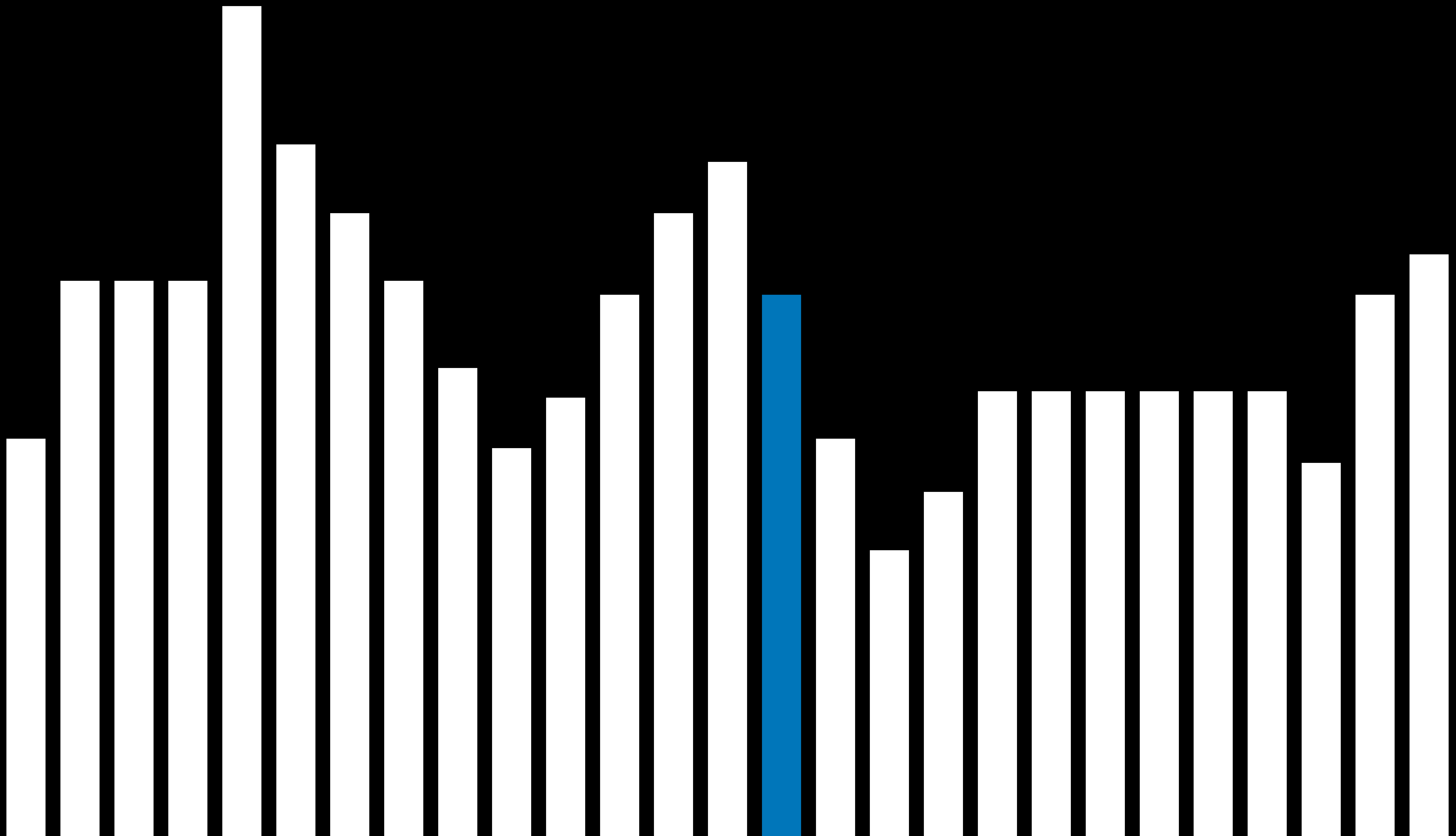

global minimum

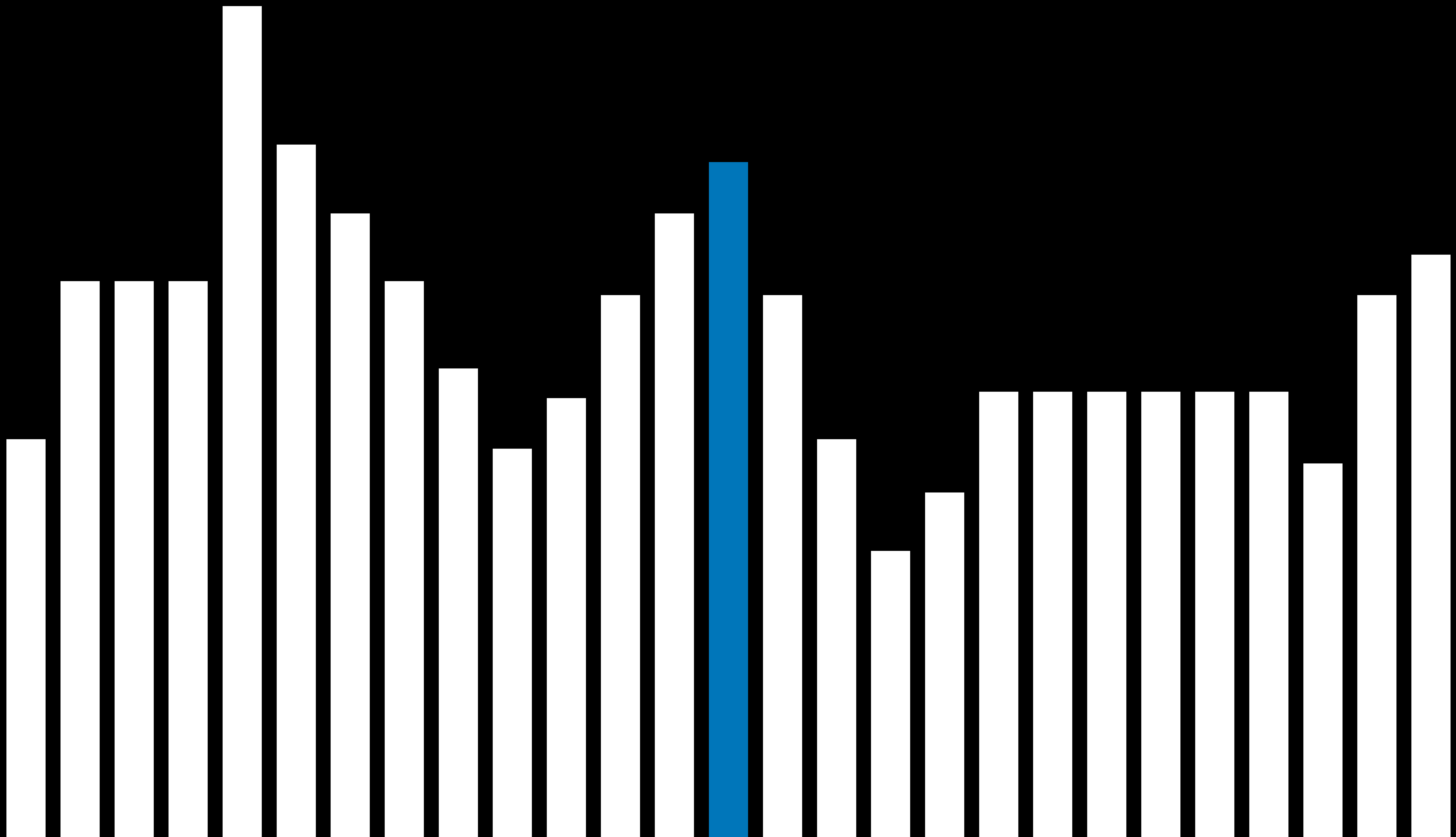

local minima



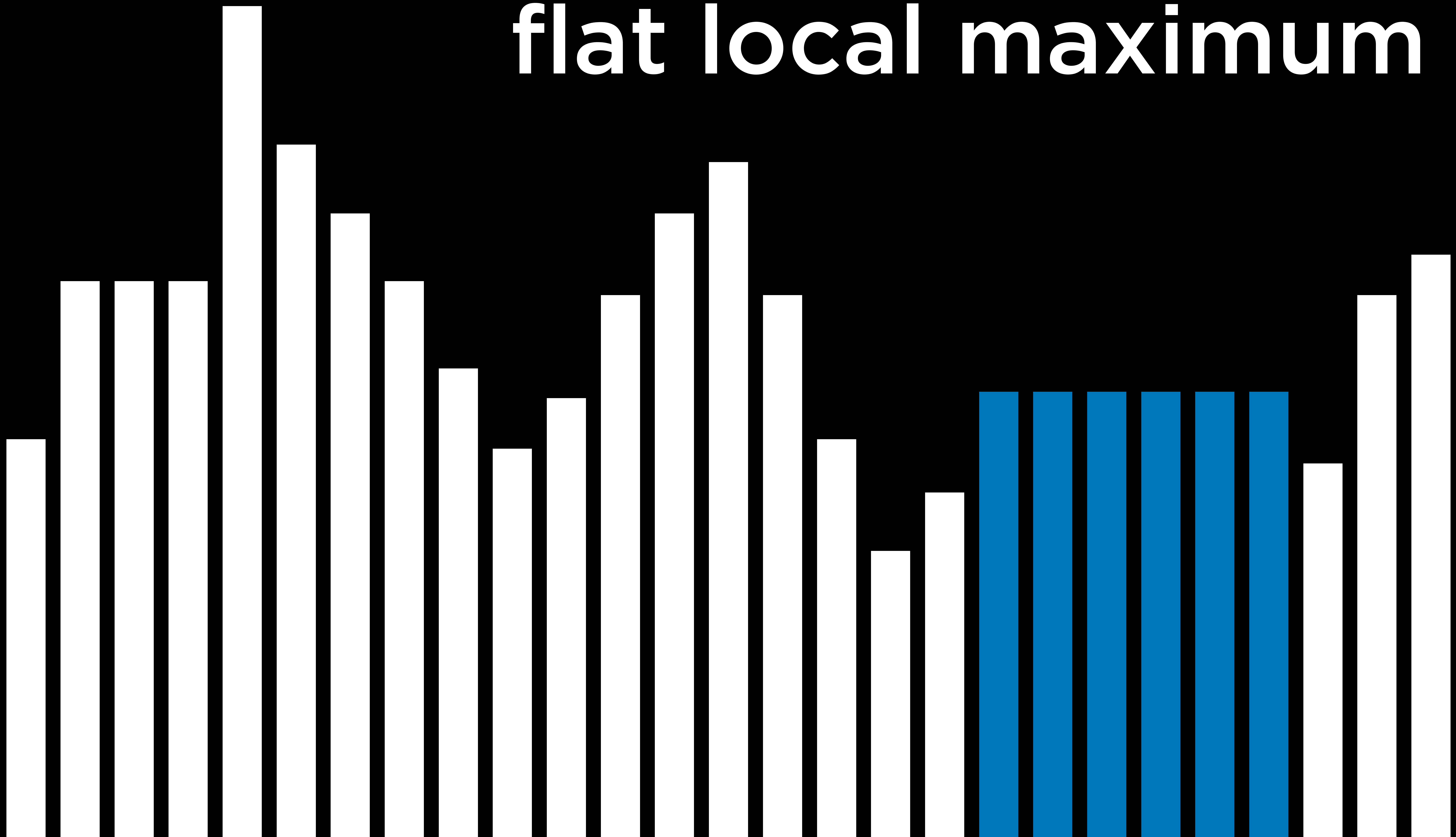




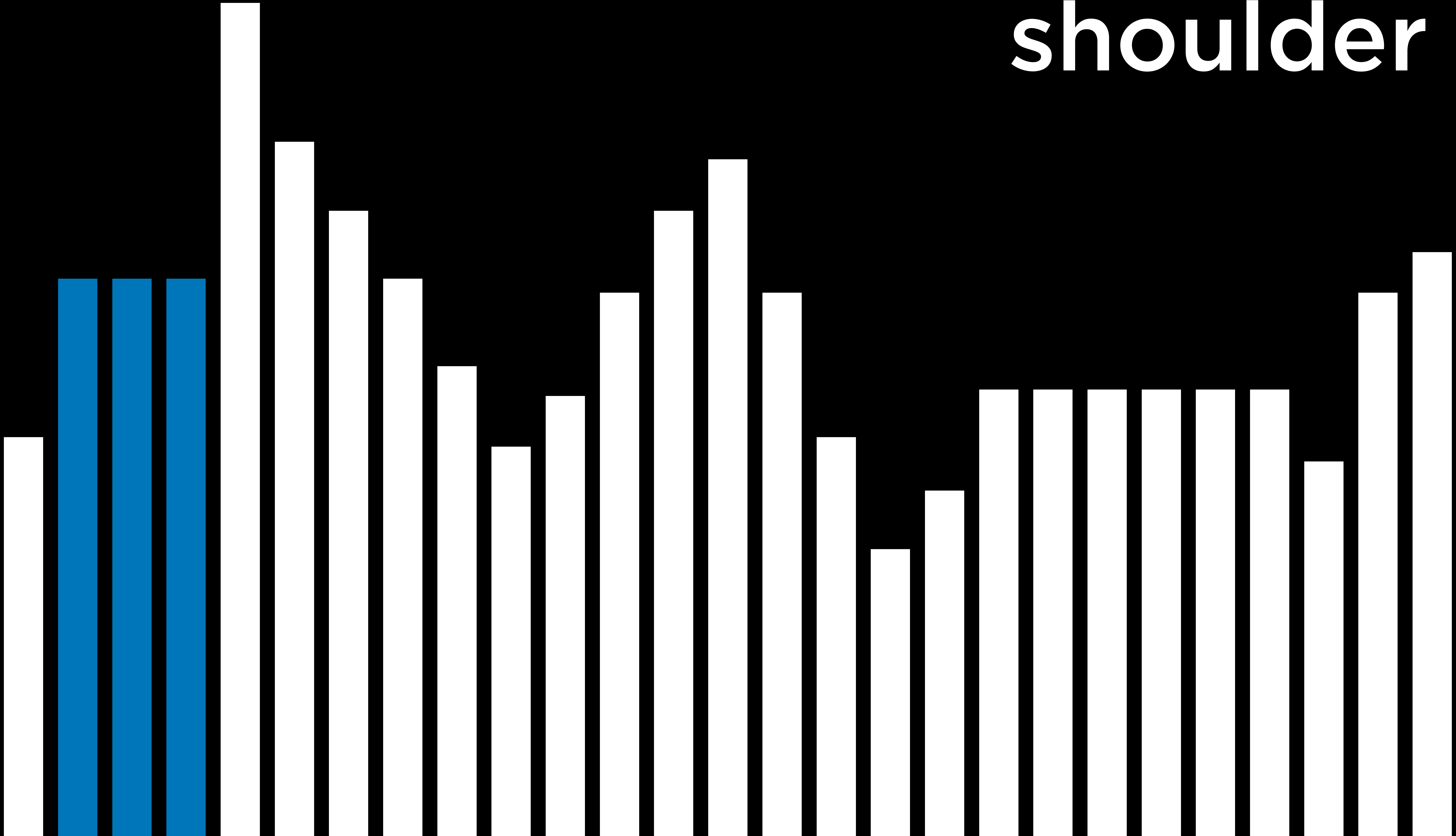




flat local maximum



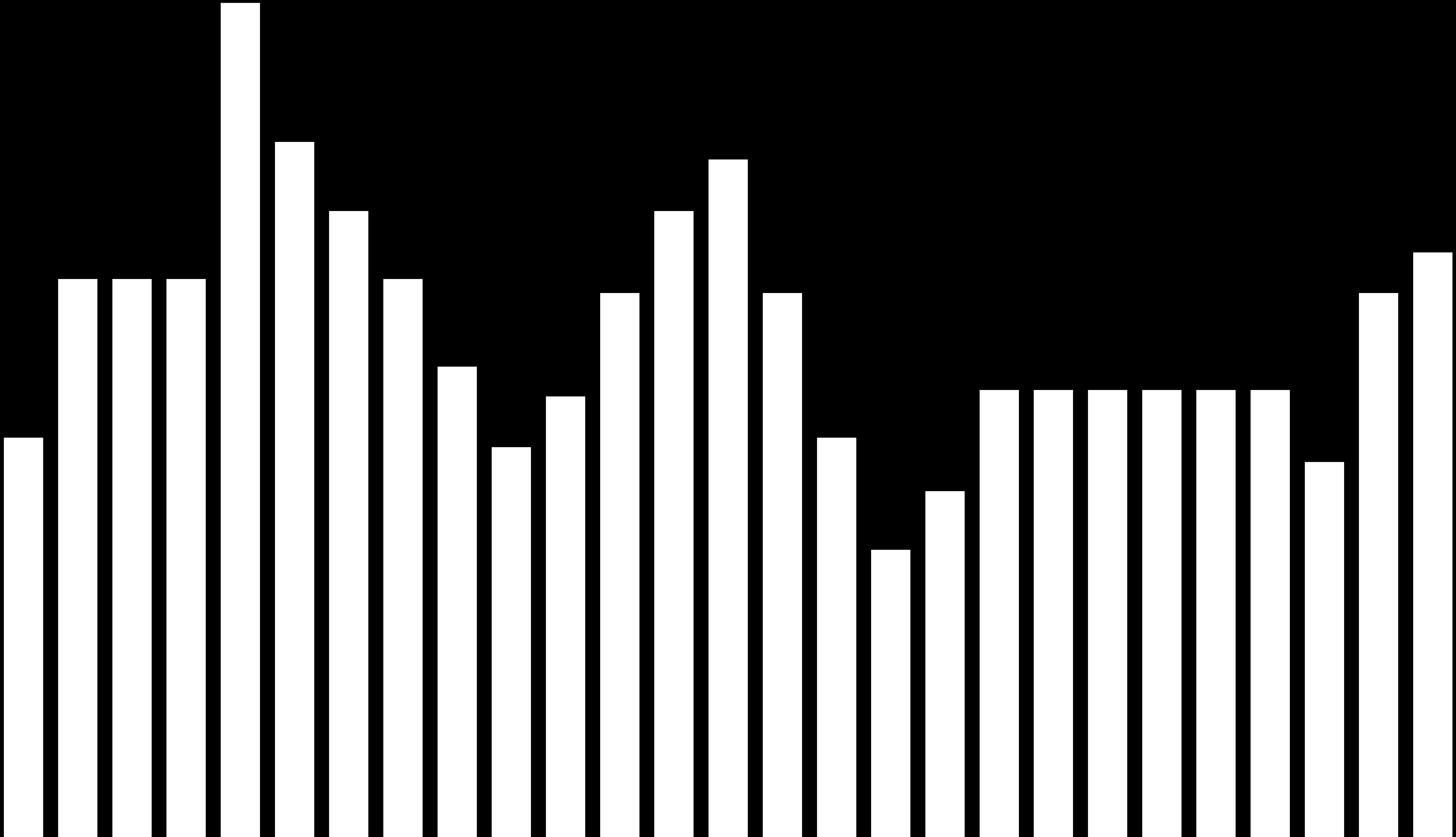
shoulder

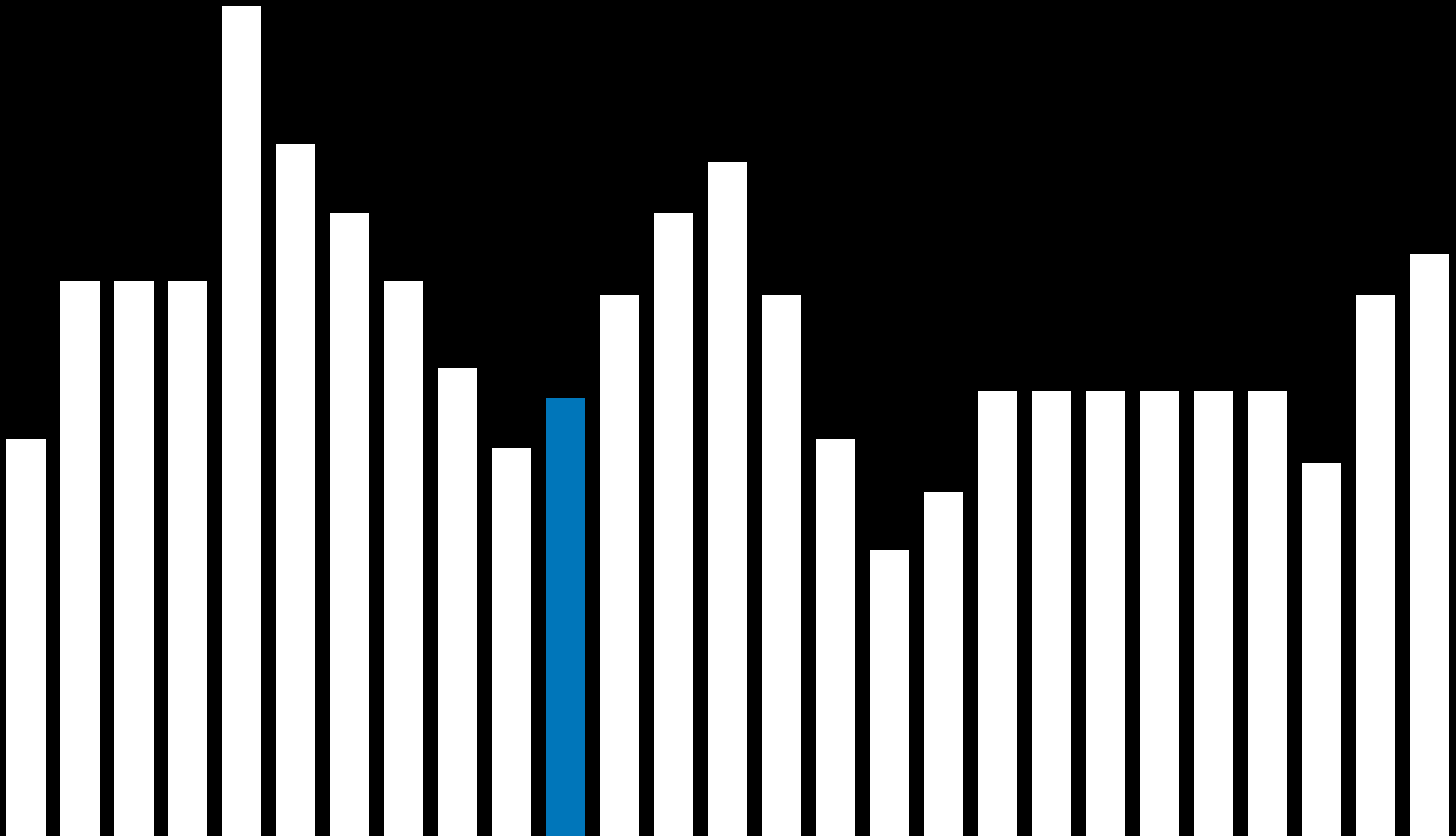


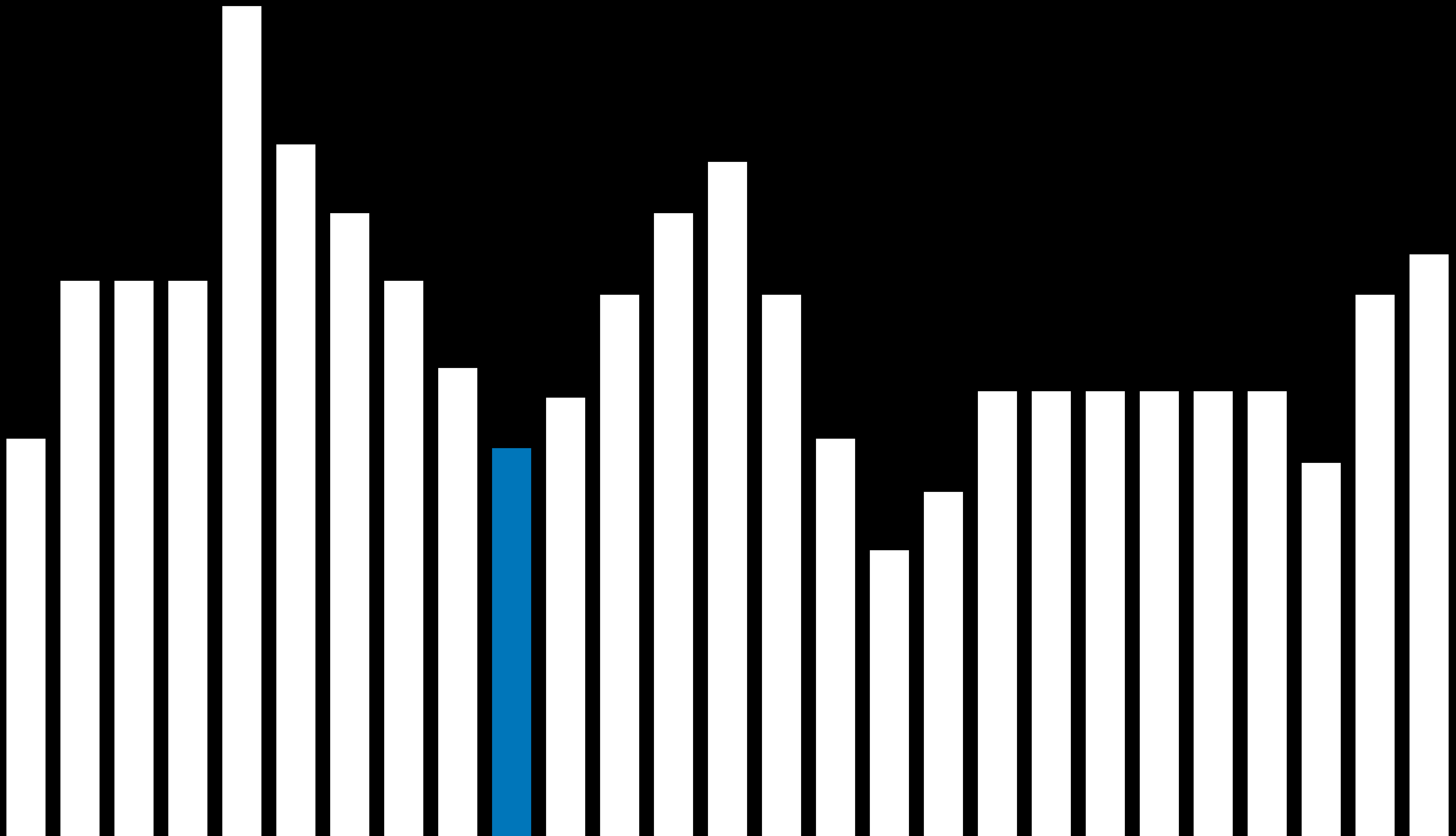
Hill Climbing Variants

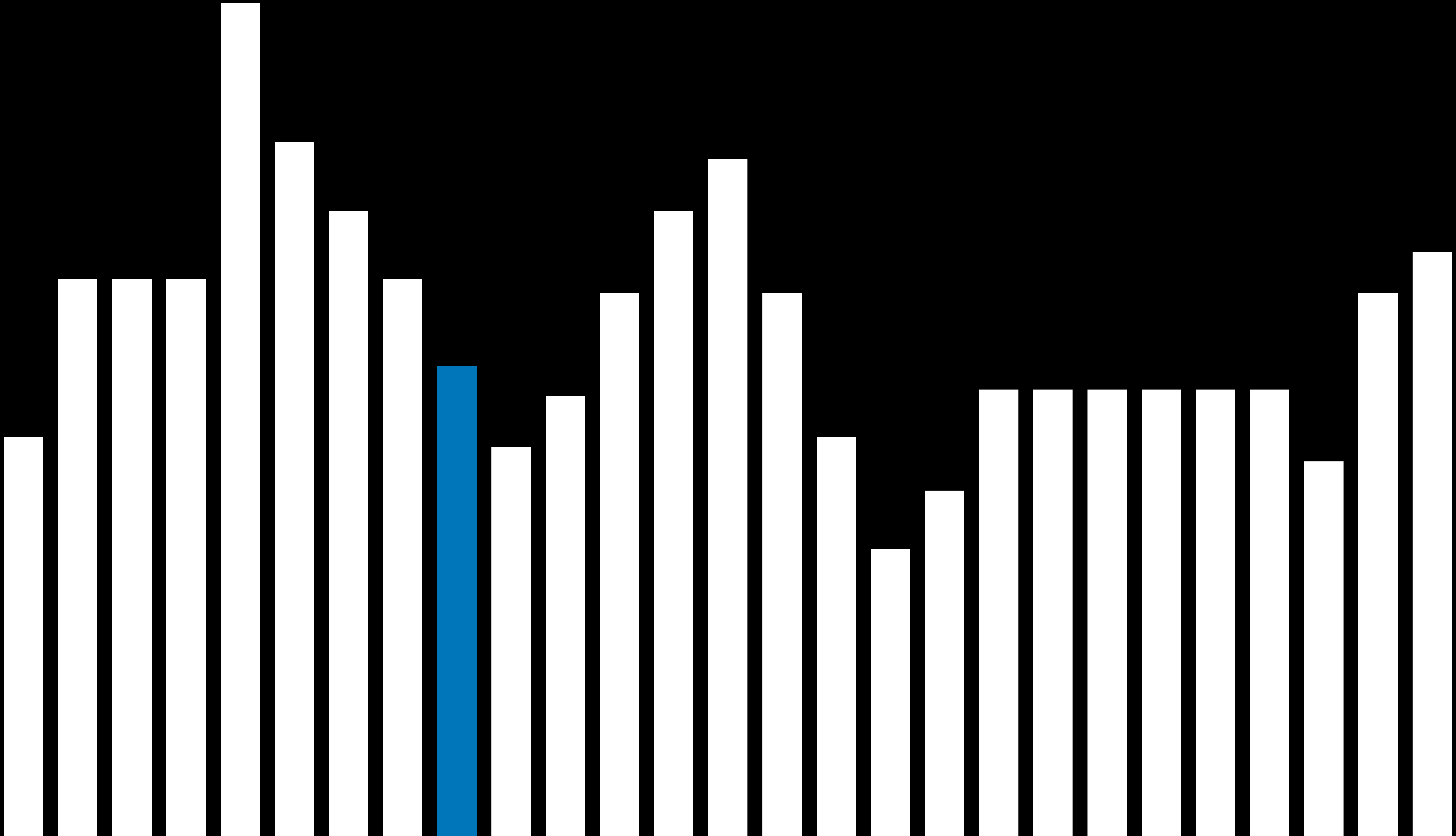
Variant	Definition
steepest-ascent	choose the highest-valued neighbor
stochastic	choose randomly from higher-valued neighbors
first-choice	choose the first higher-valued neighbor
random-restart	conduct hill climbing multiple times
local beam search	chooses the *k* highest-valued neighbors

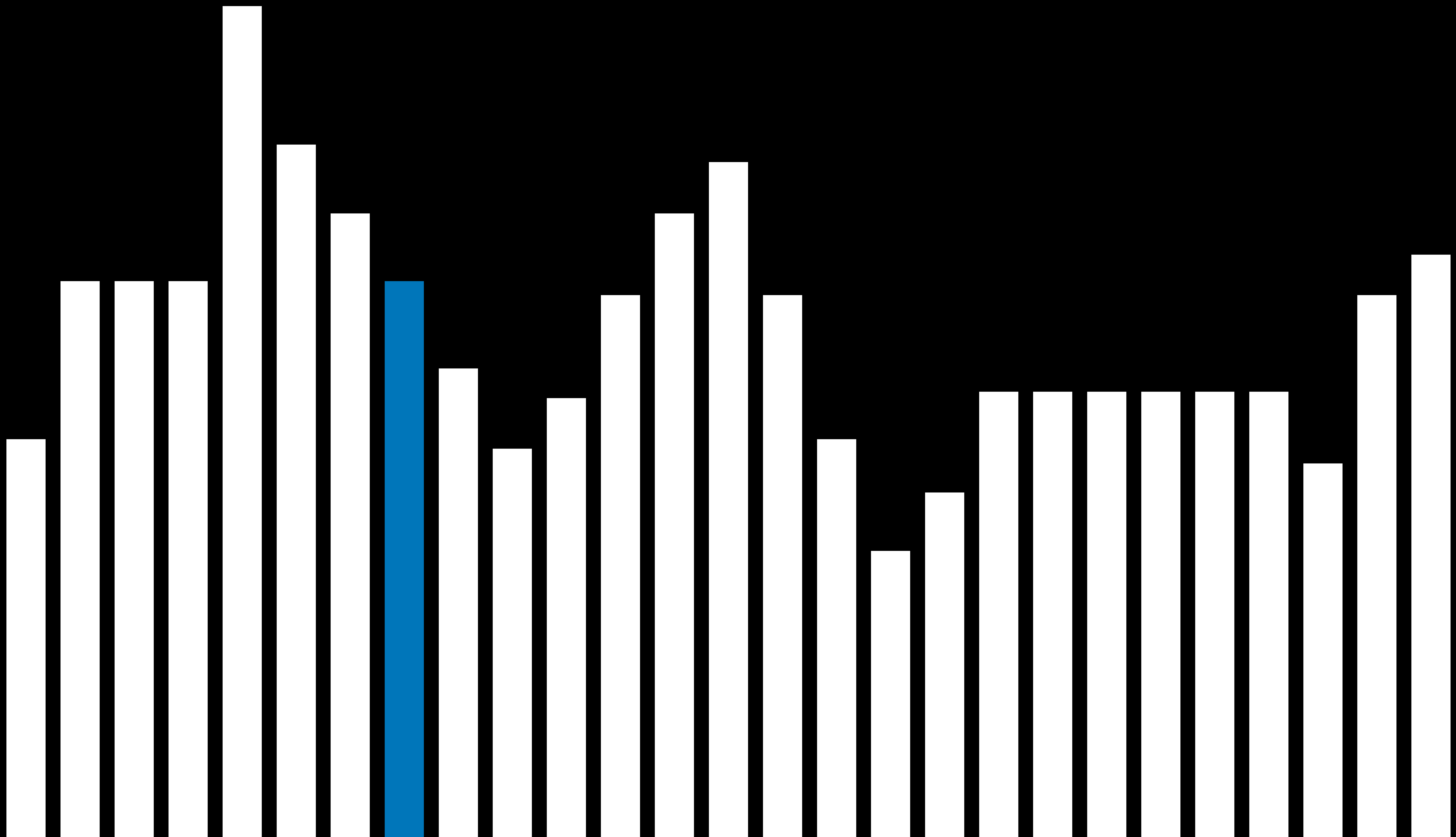
Simulated Annealing

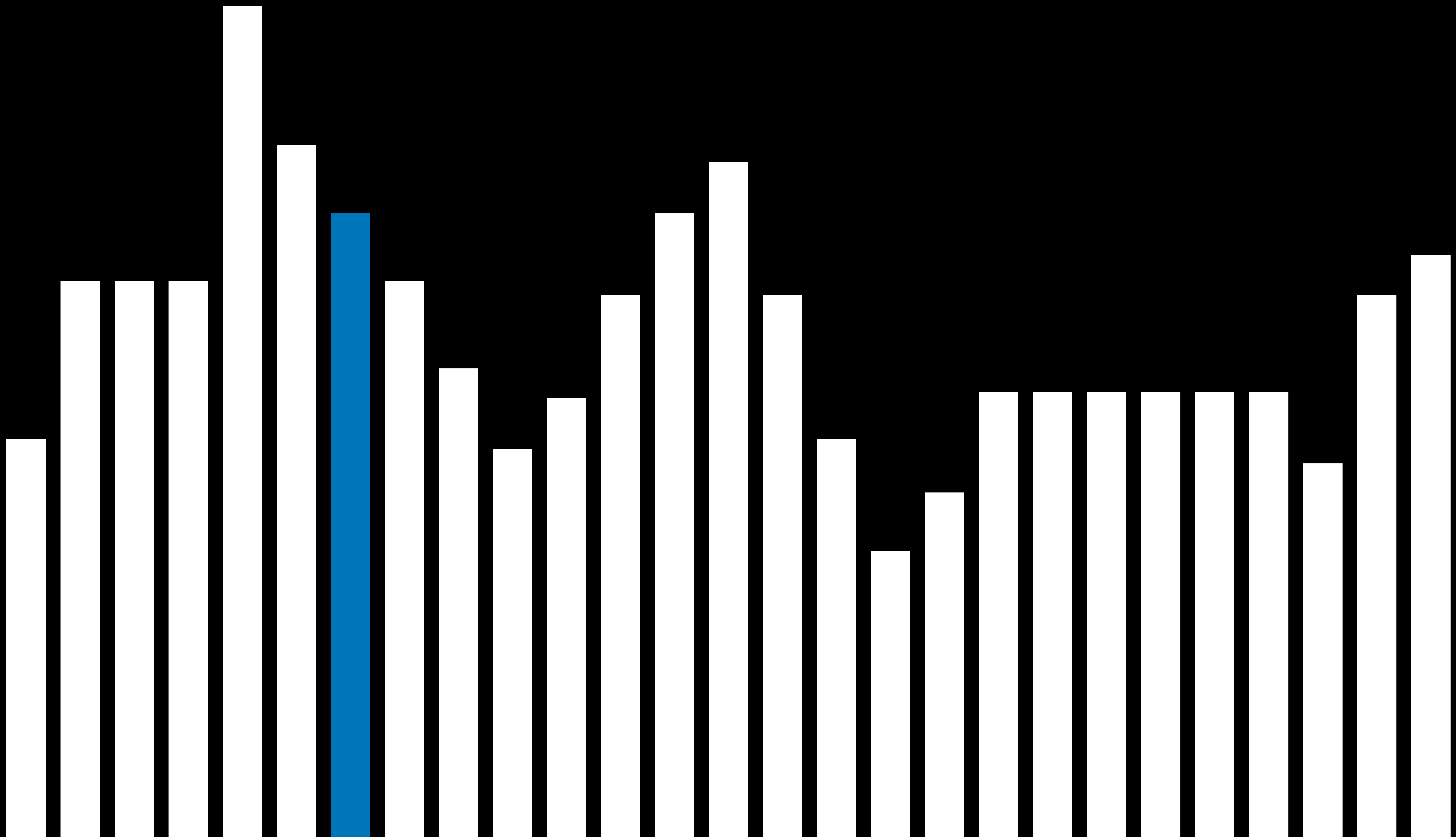


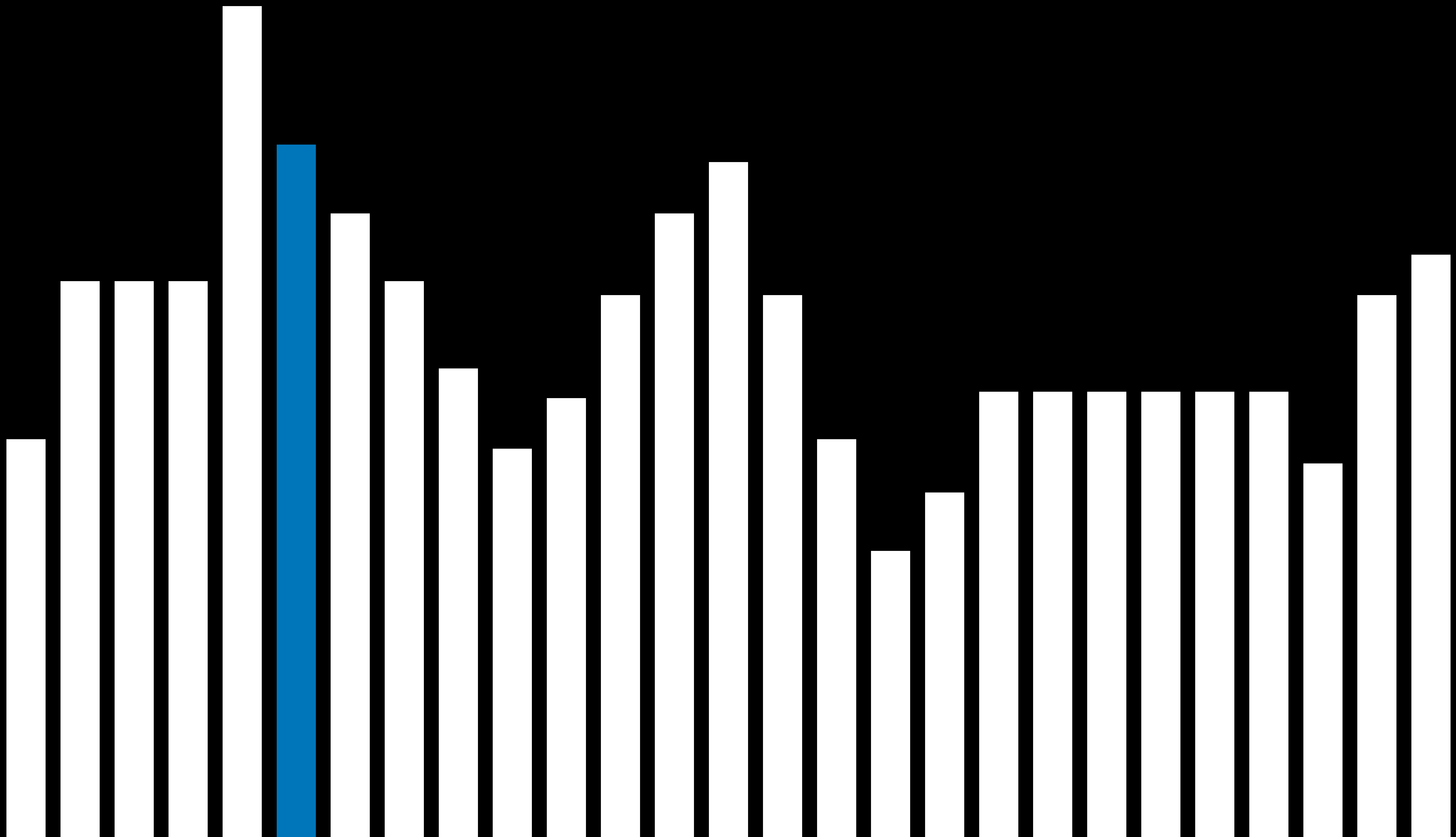


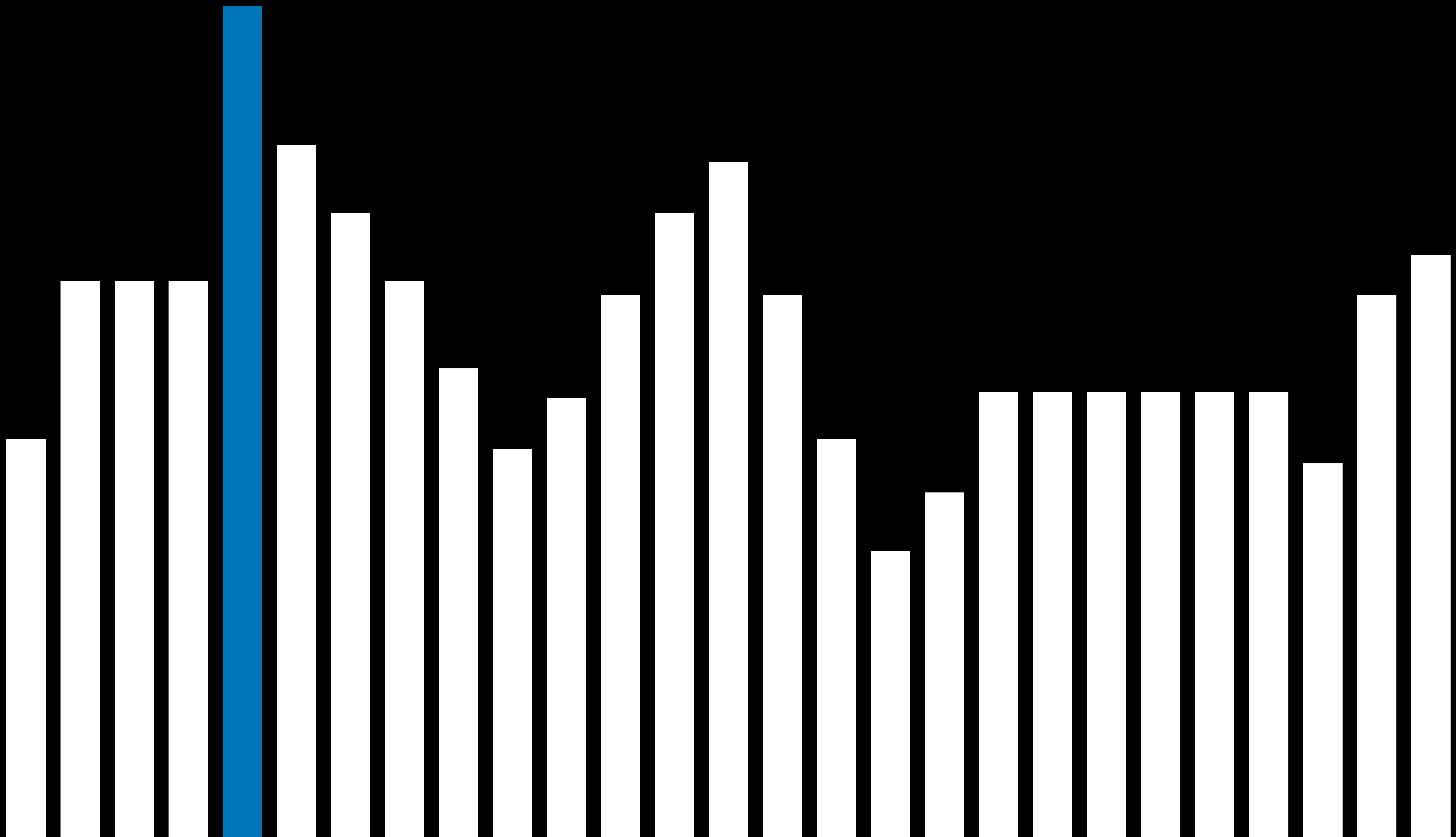












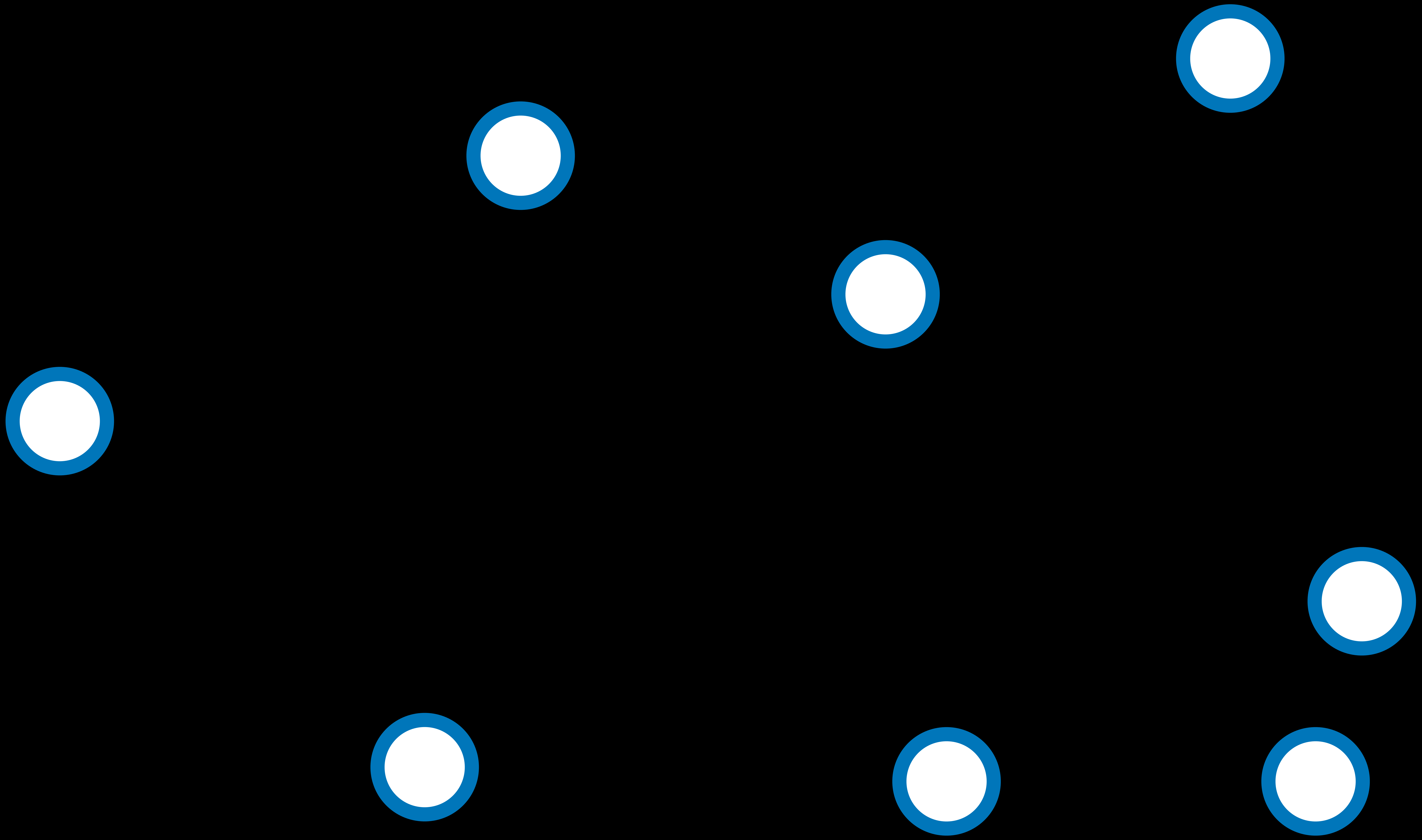
Simulated Annealing

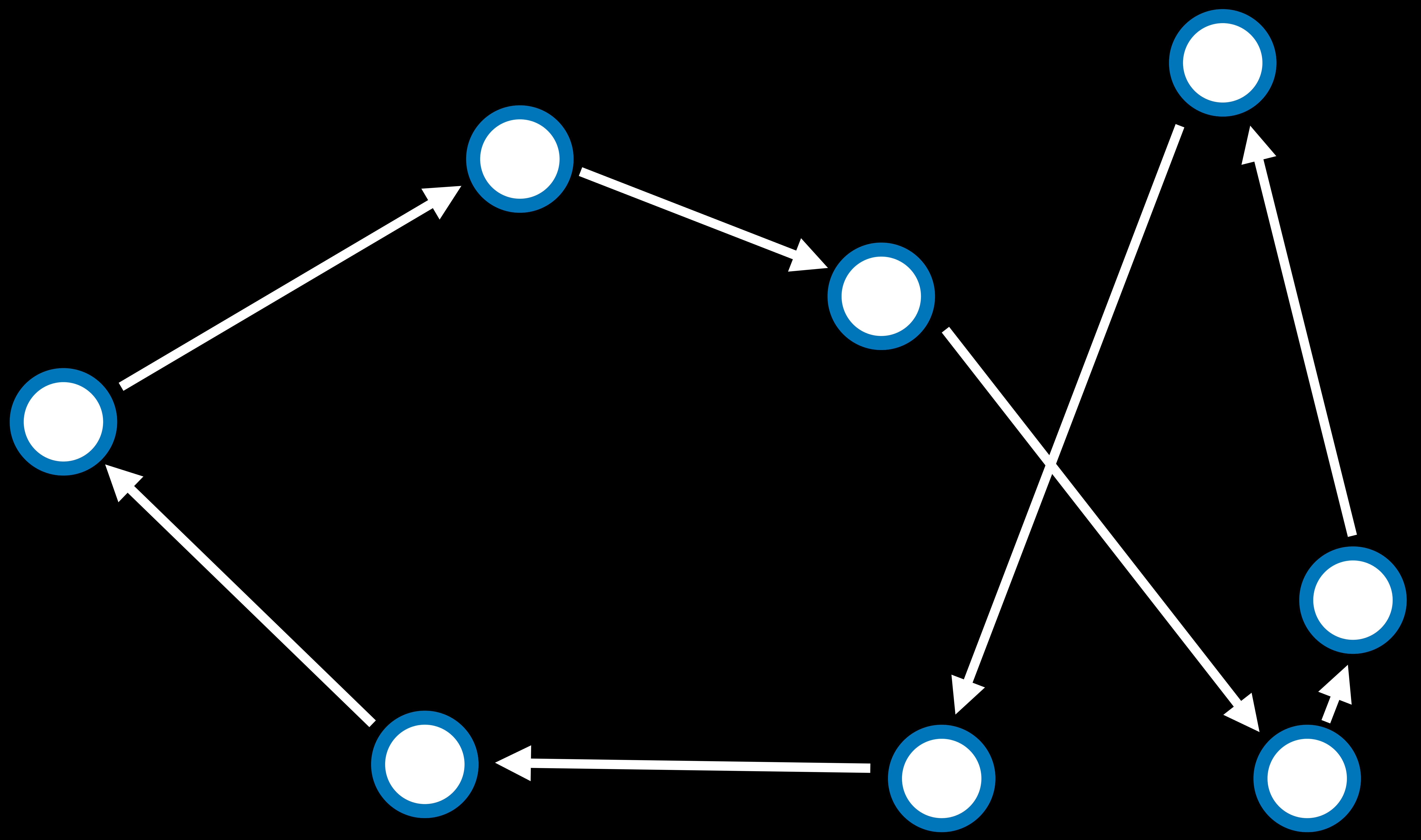
- Early on, higher "temperature": more likely to accept neighbors that are worse than current state
- Later on, lower "temperature": less likely to accept neighbors that are worse than current state

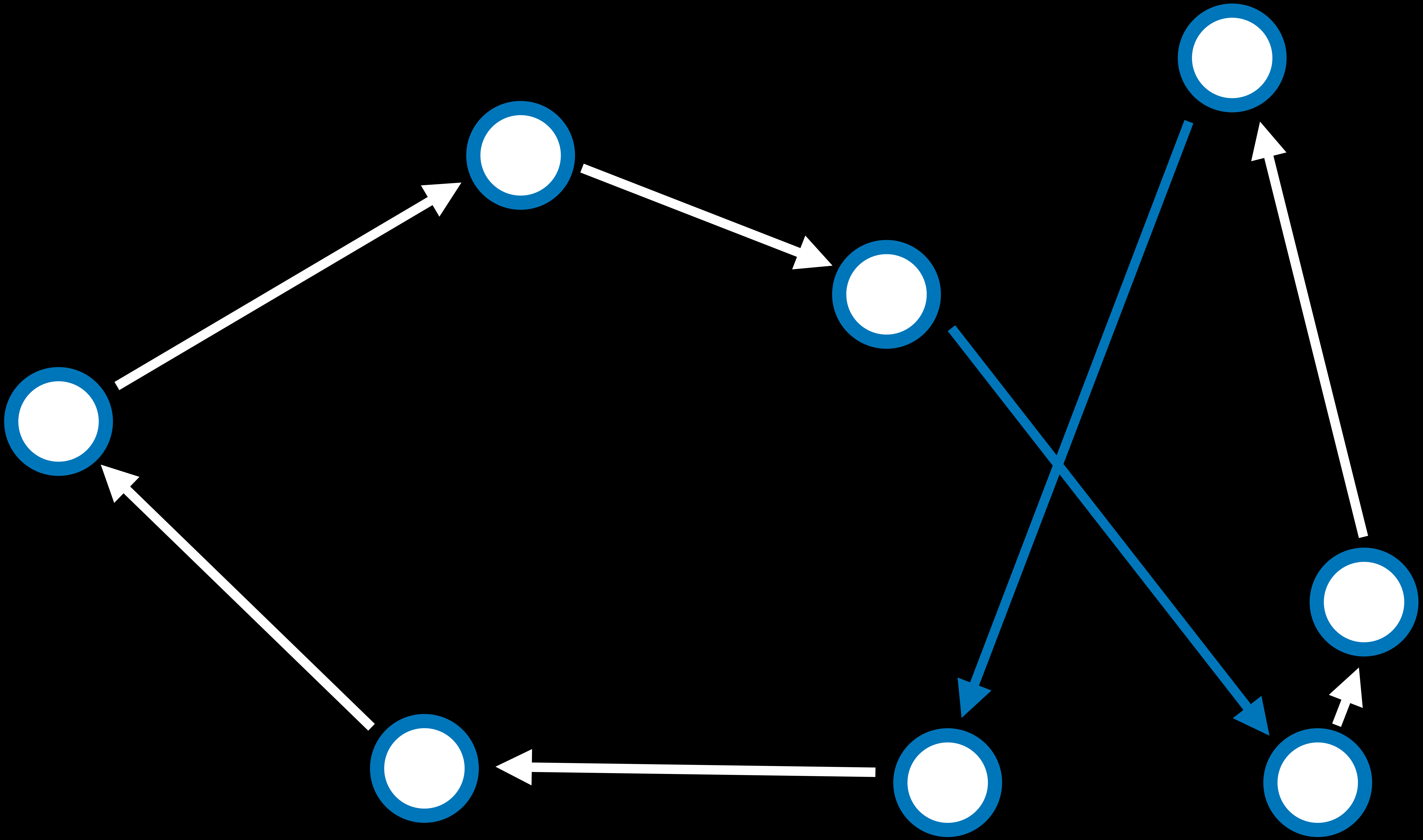
Simulated Annealing

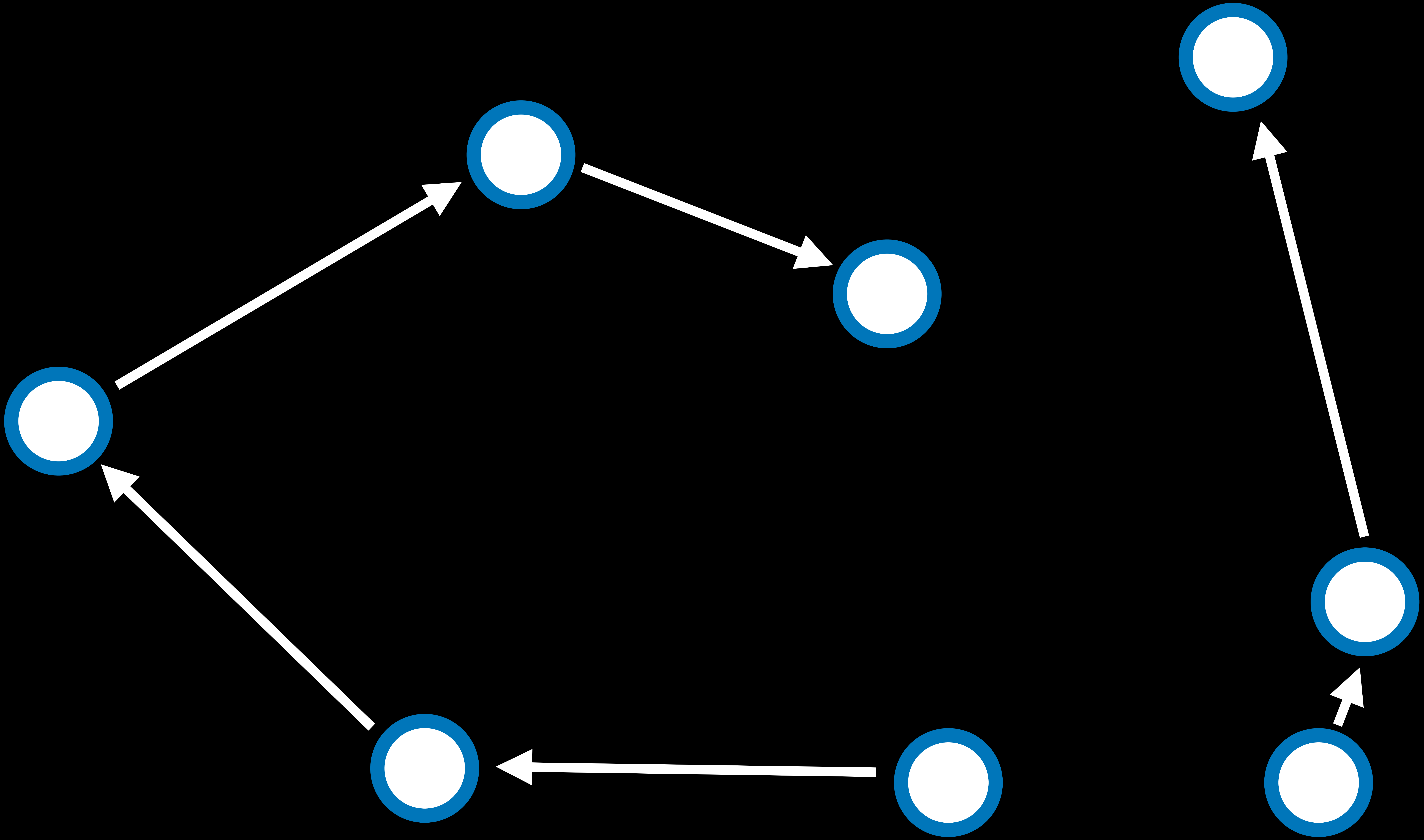
function SIMULATED-ANNEALING(*problem*, *max*):
    current = initial state of *problem*
    for *t* = 1 to *max*:
        T = TEMPERATURE(*t*)
        neighbor = random neighbor of *current*
        ΔE = how much better *neighbor* is than *current*
        if $\Delta E > 0$:
            current = *neighbor*
        with probability $e^{\Delta E/T}$ set *current* = *neighbor*
    return *current*

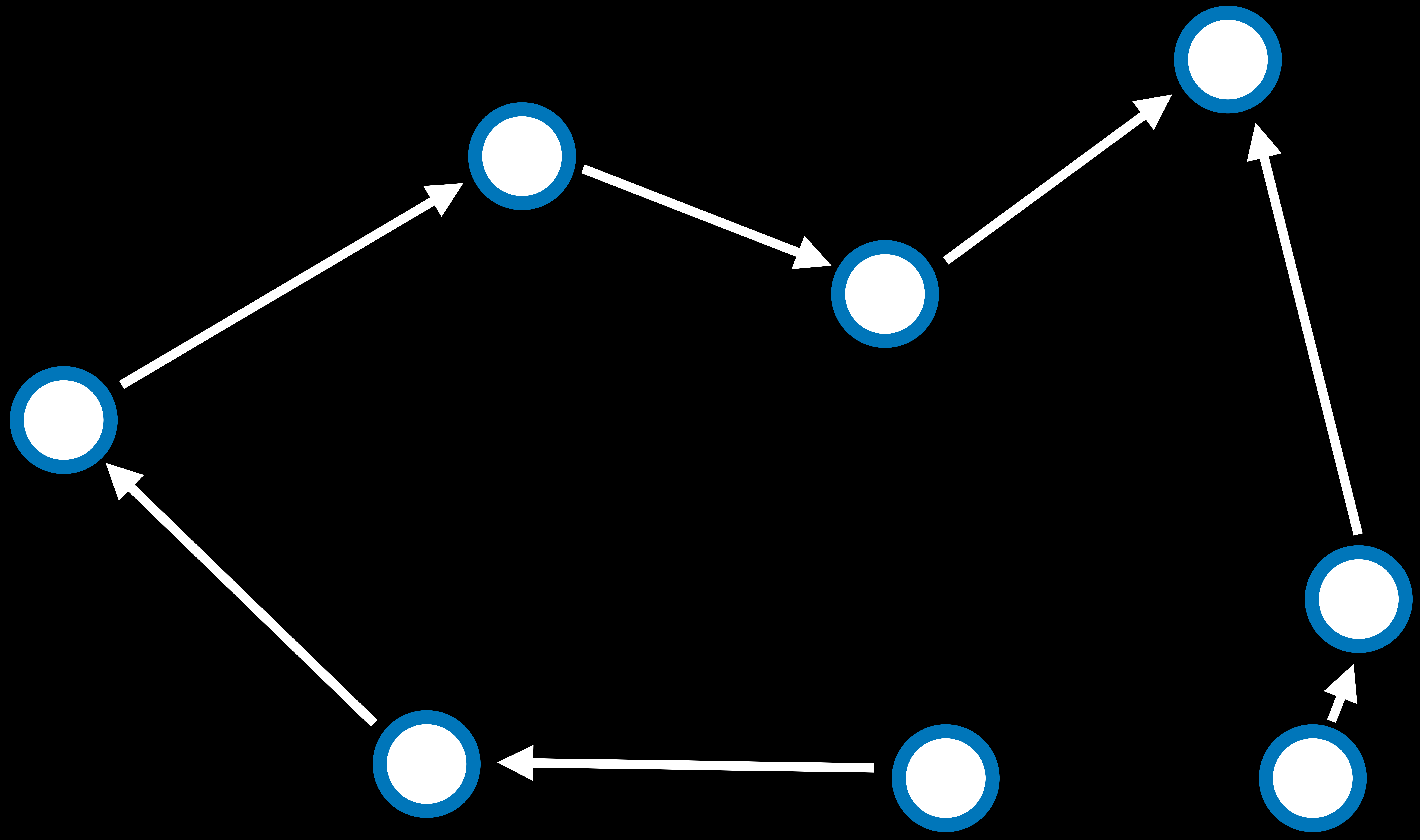
Traveling Salesman Problem

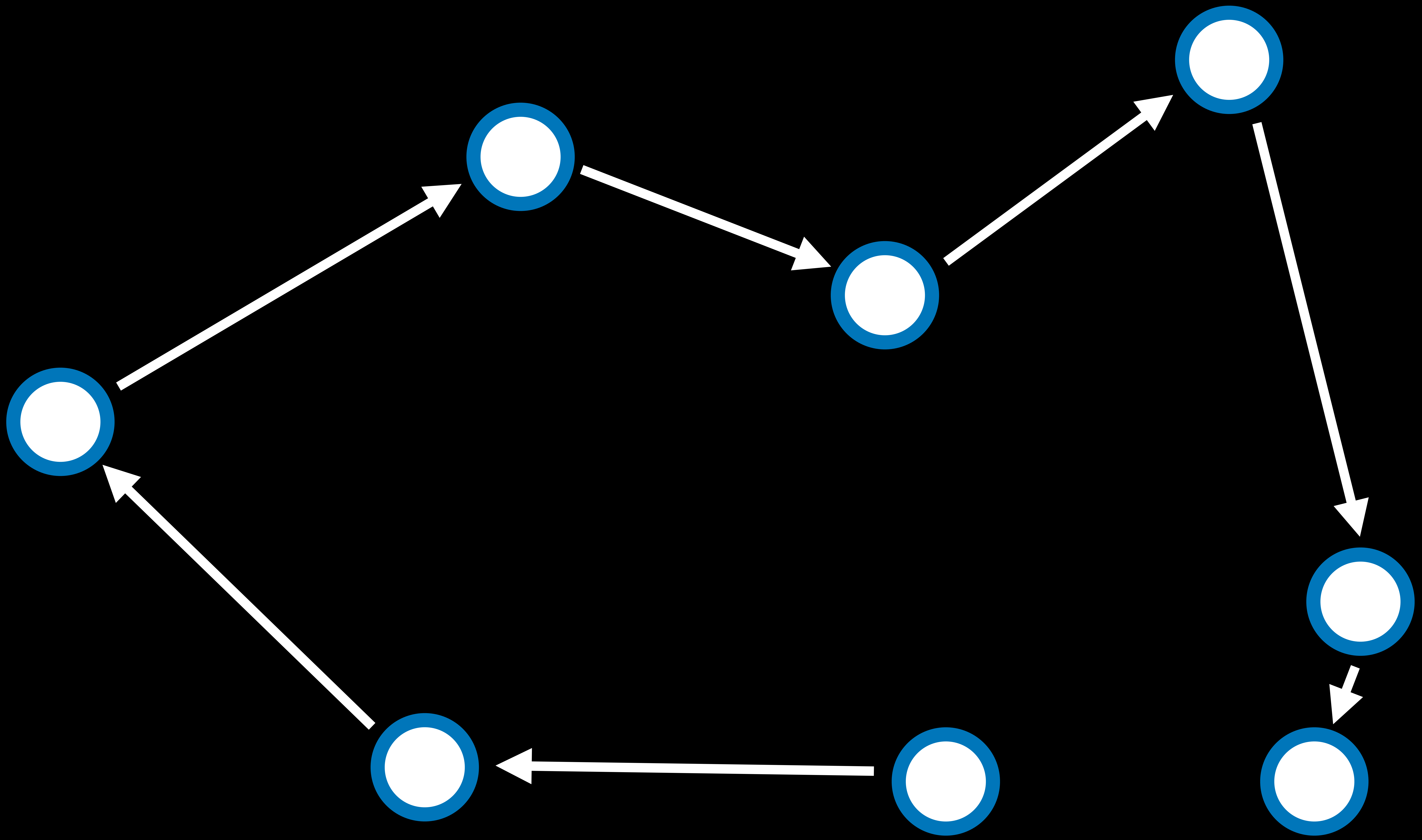


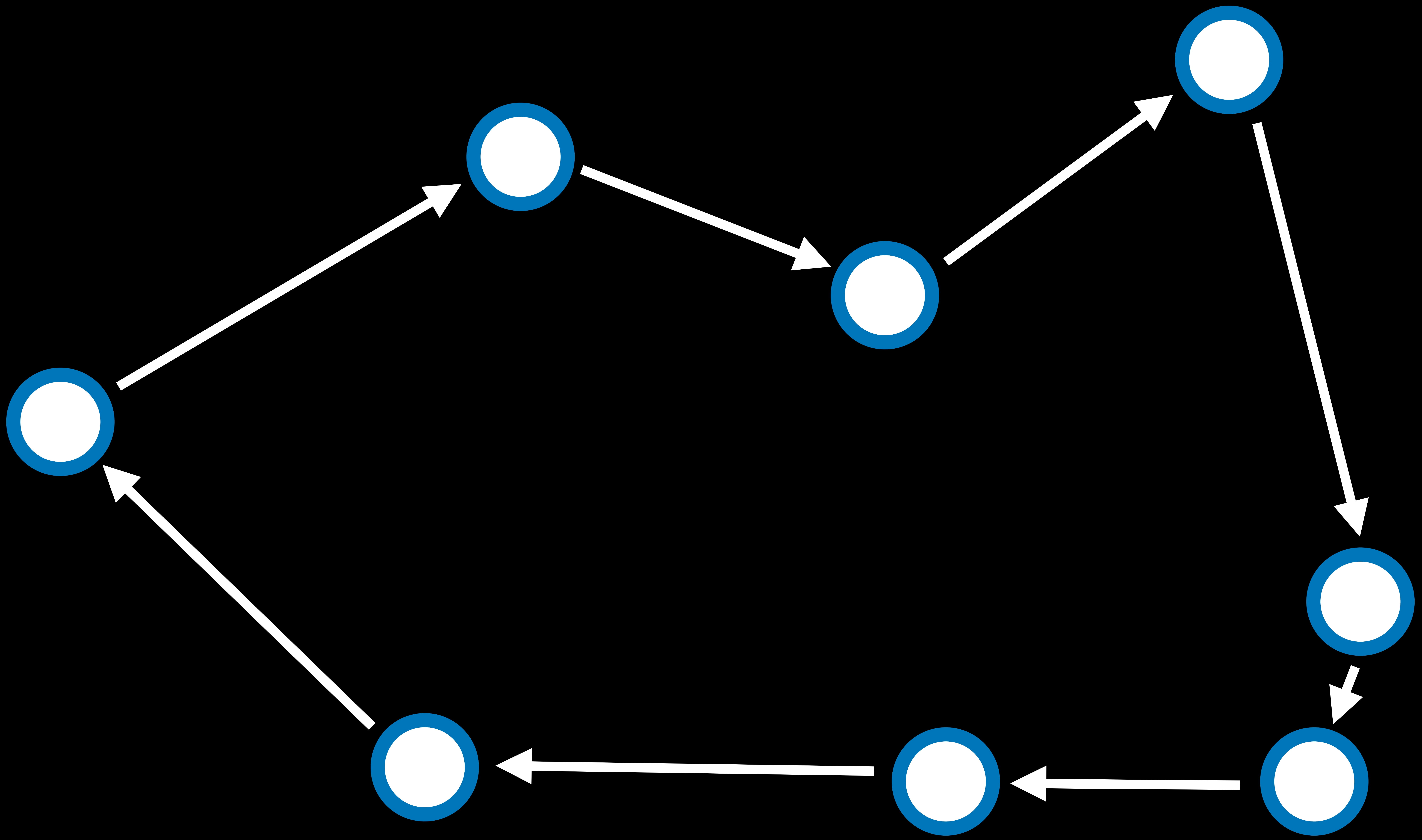












Linear Programming

Linear Programming

- Minimize a cost function $c_1x_1 + c_2x_2 + ... + c_nx_n$
- With constraints of form $a_1x_1 + a_2x_2 + ... + a_nx_n \leq b$ or of form $a_1x_1 + a_2x_2 + ... + a_nx_n = b$
- 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: $5x_1 + 2x_2 \leq 20$

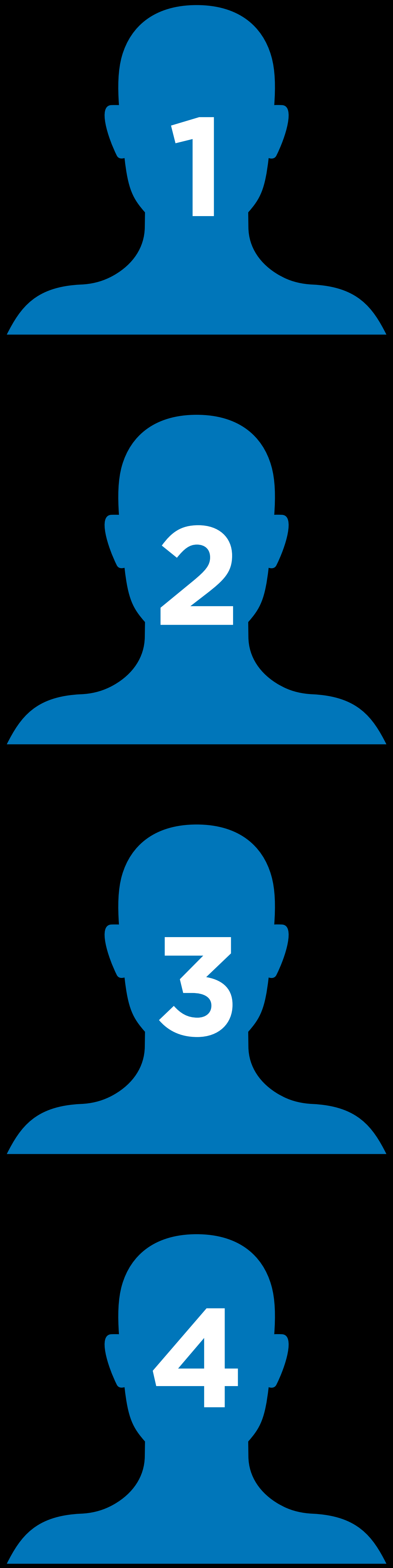
Constraint: $(-10x_1) + (-12x_2) \leq -90$

Linear Programming Algorithms

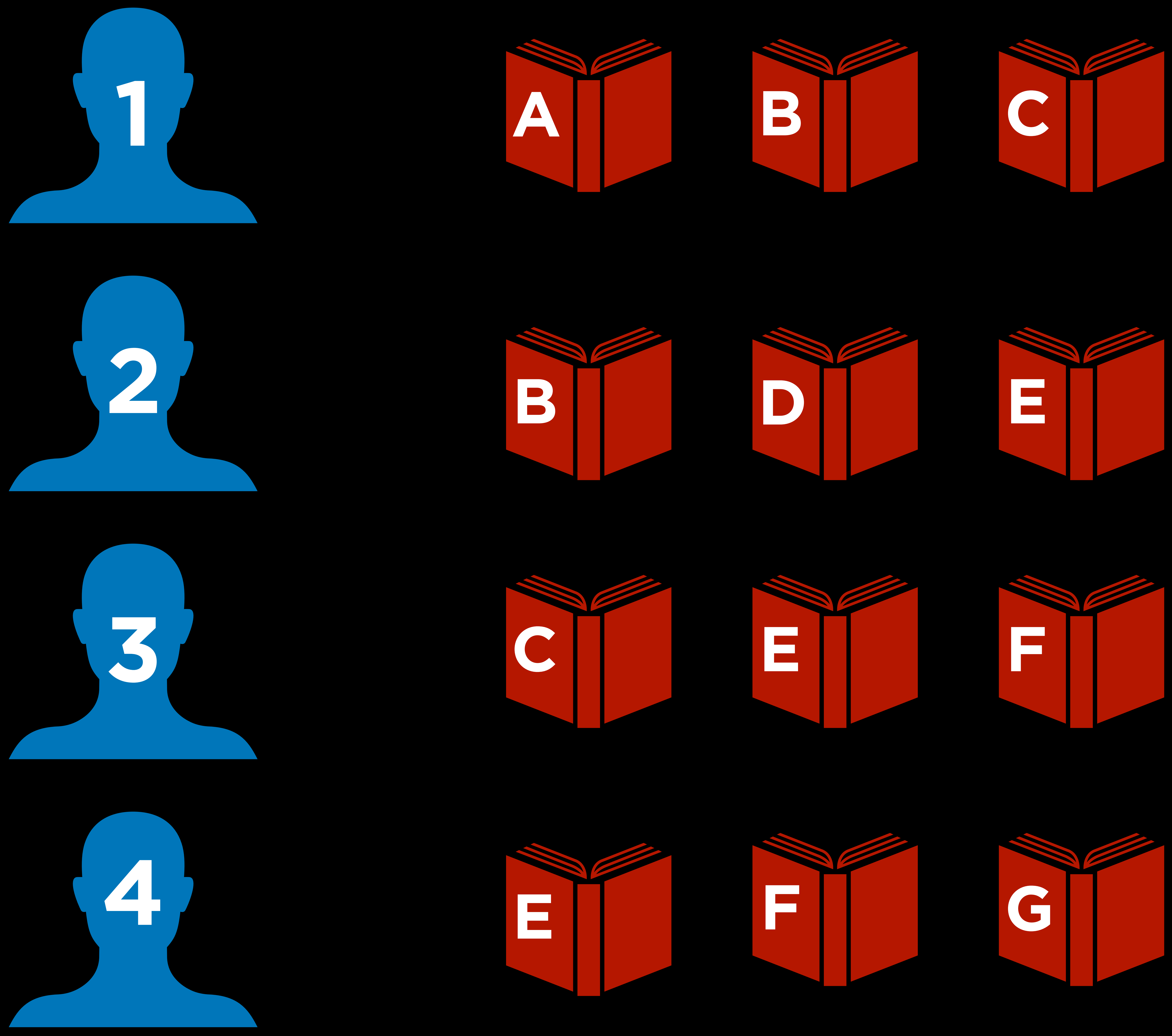
- Simplex
- Interior-Point

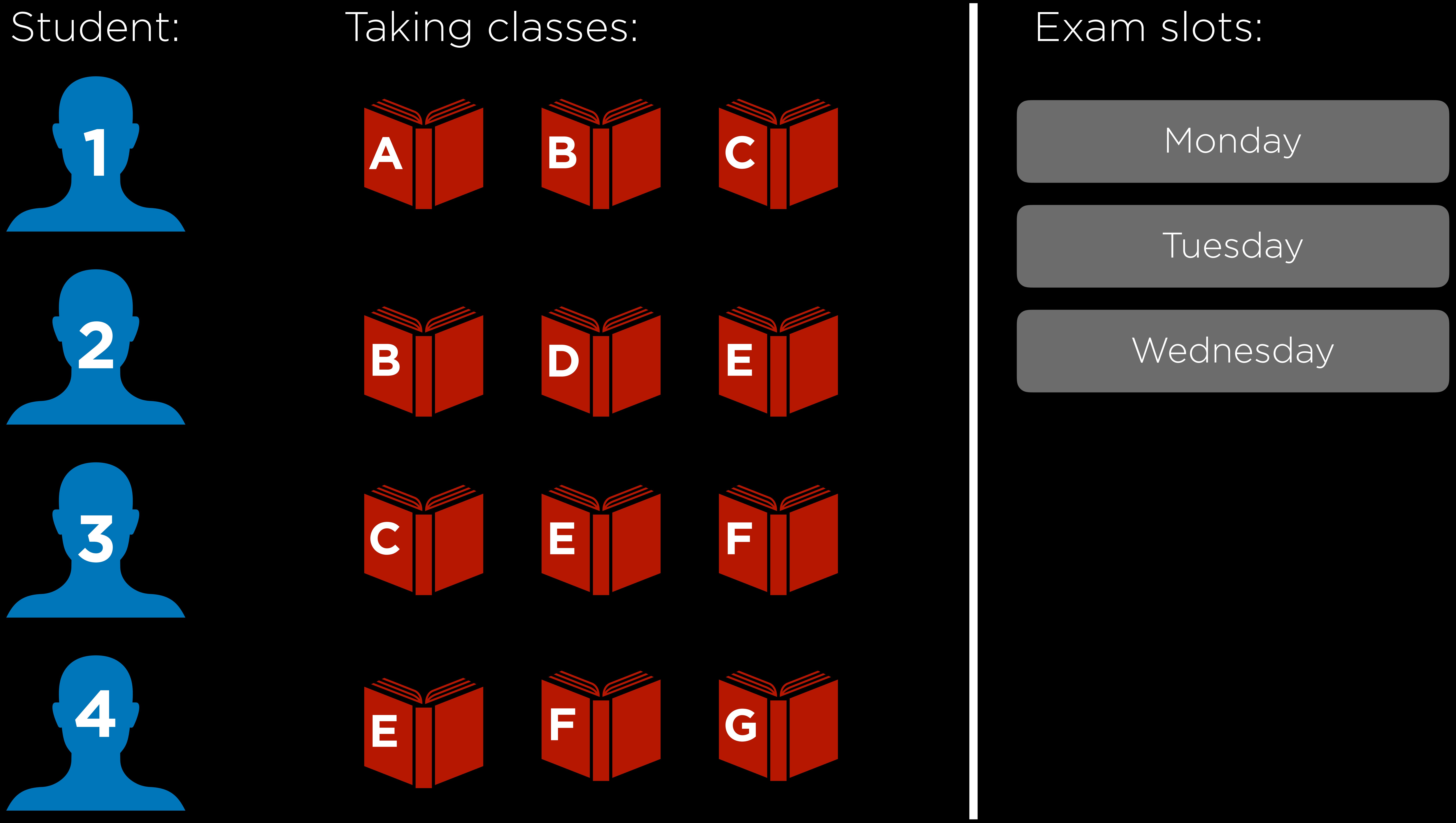
Constraint Satisfaction

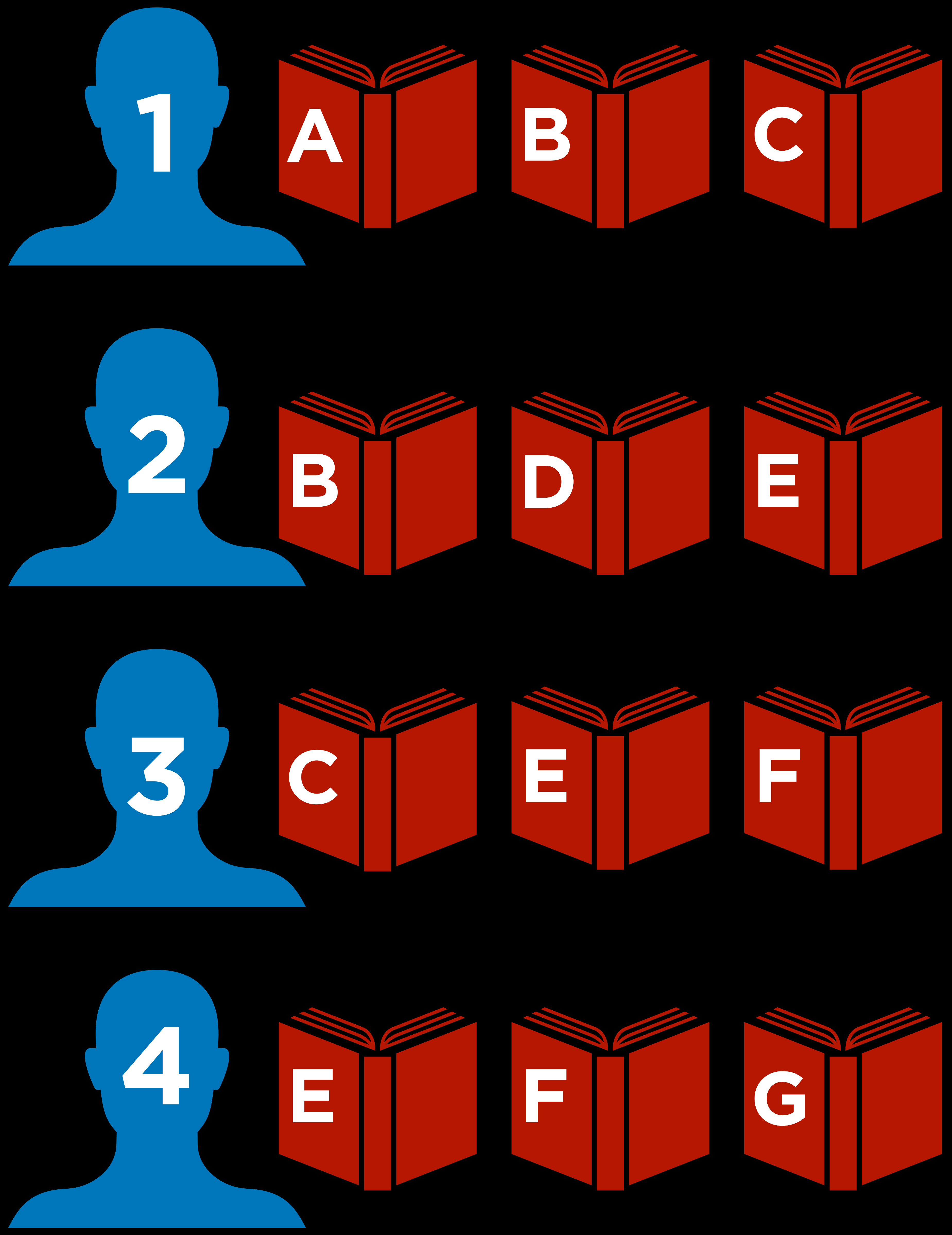
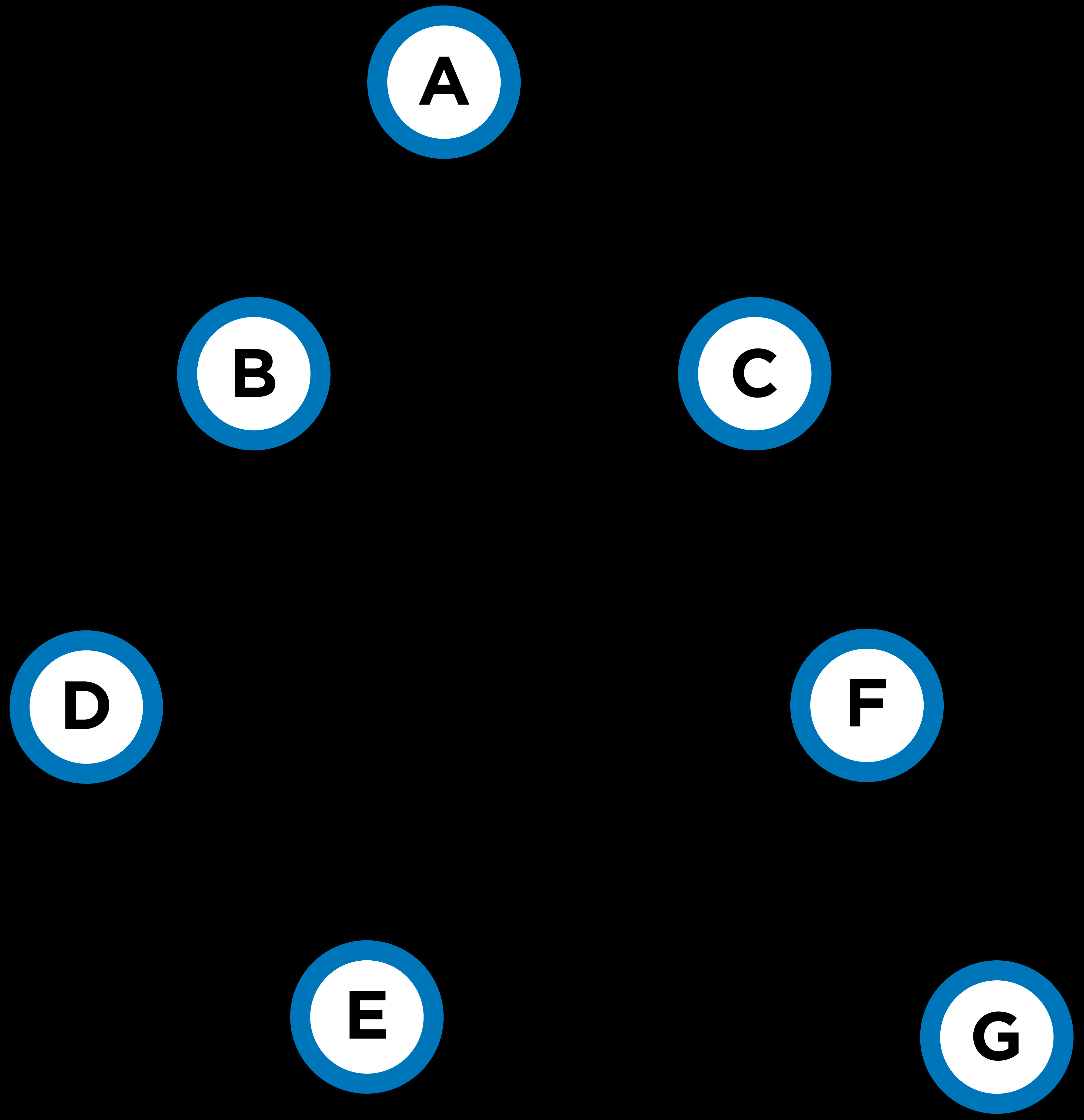
Student:

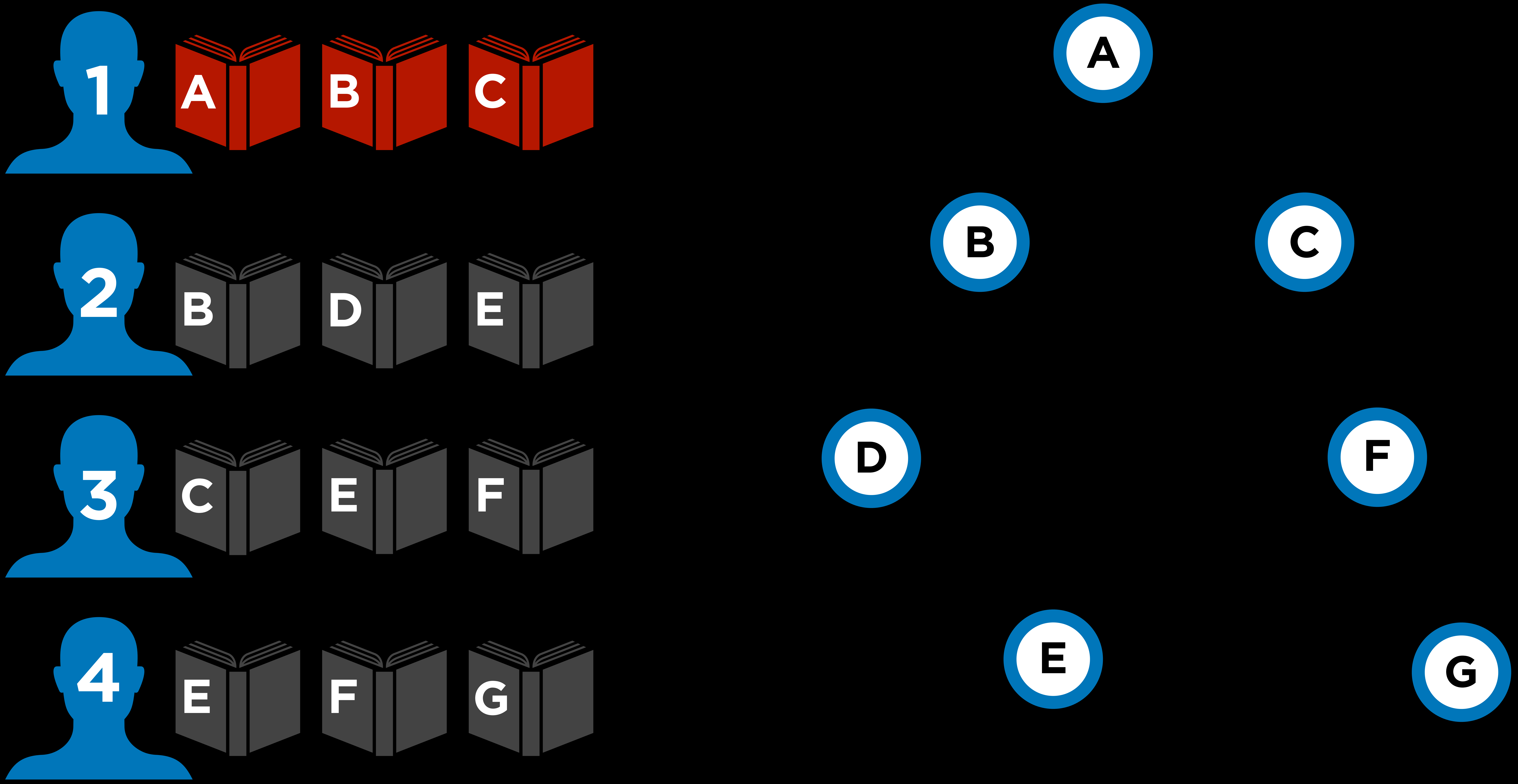


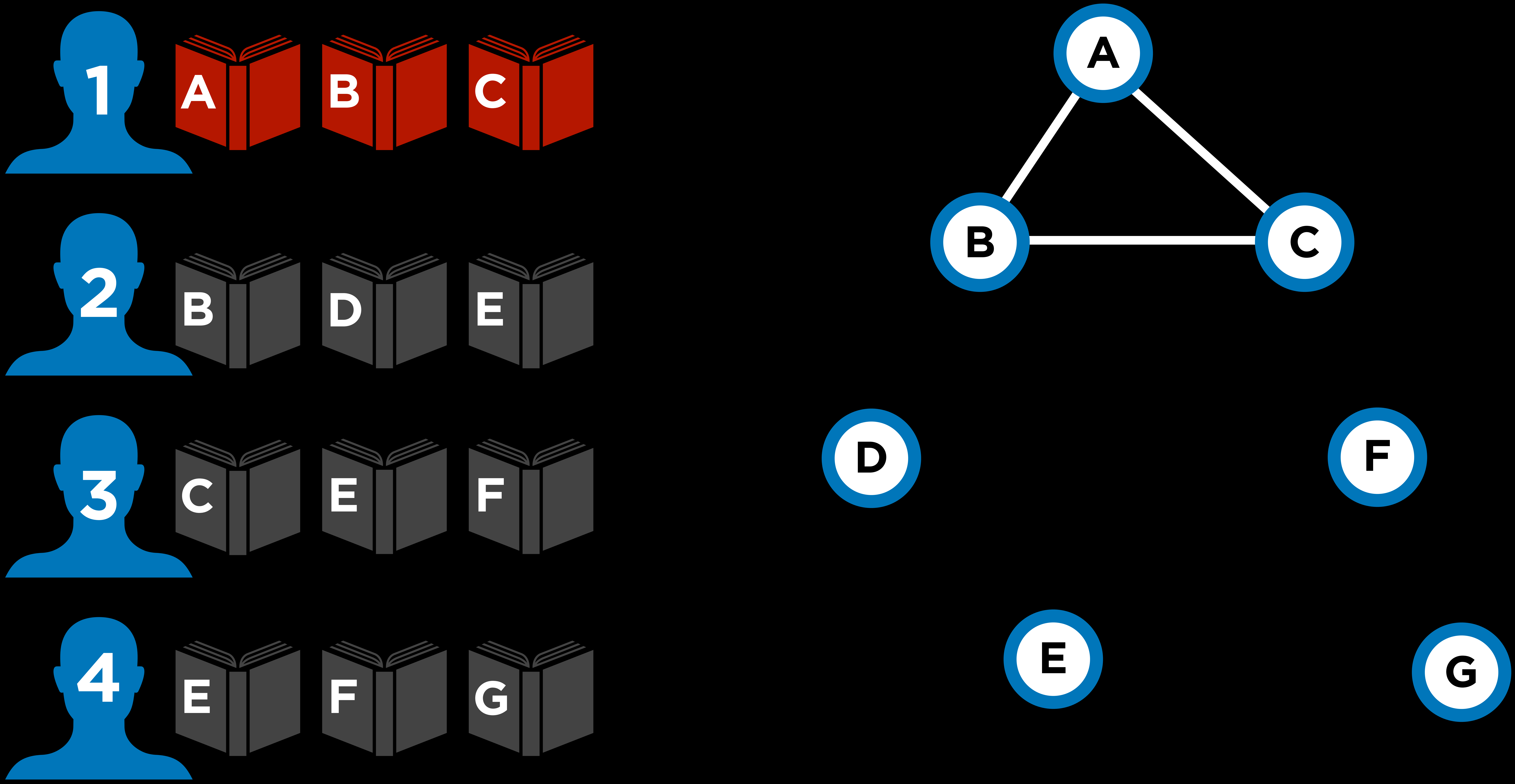
Student: Taking classes:

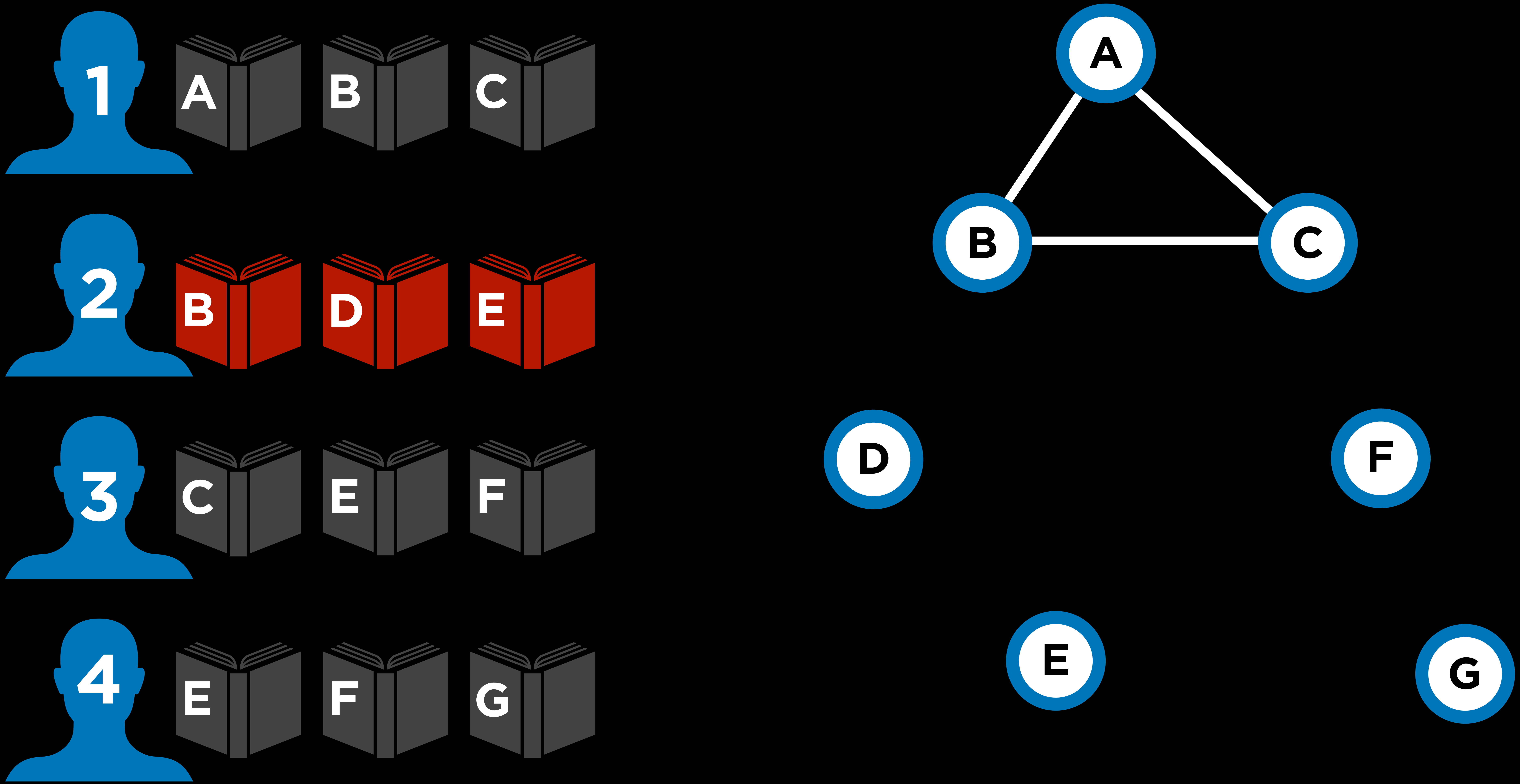


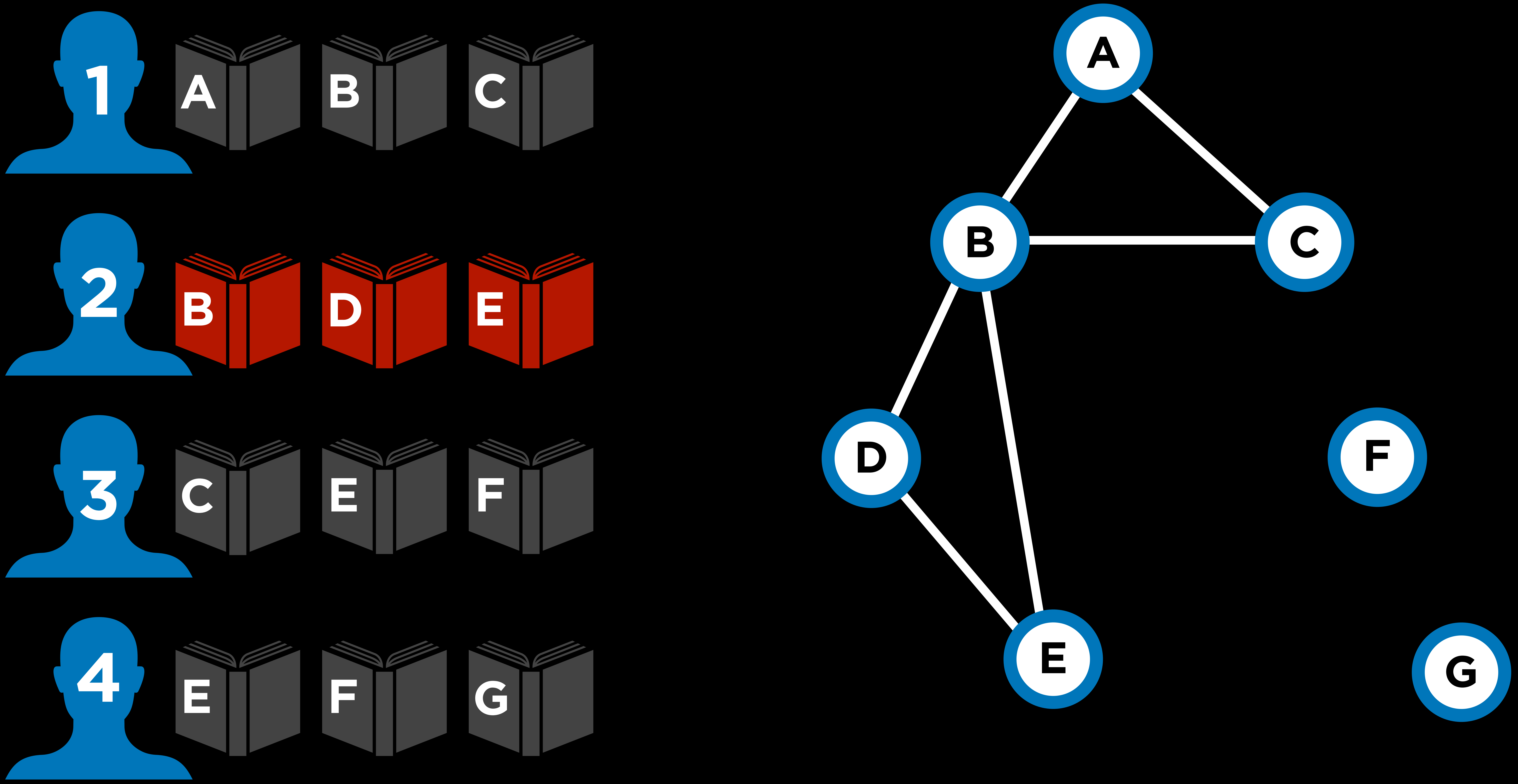

Student:
Taking classes:
Exam slots:
1
A
B
C
2
B
D
E
3
C
E
F
4
E
F
G
Monday
Tuesday
Wednesday


1 A B C
2 B D E
3 C E F
4 E F G

A
B
C
D
F
E
G

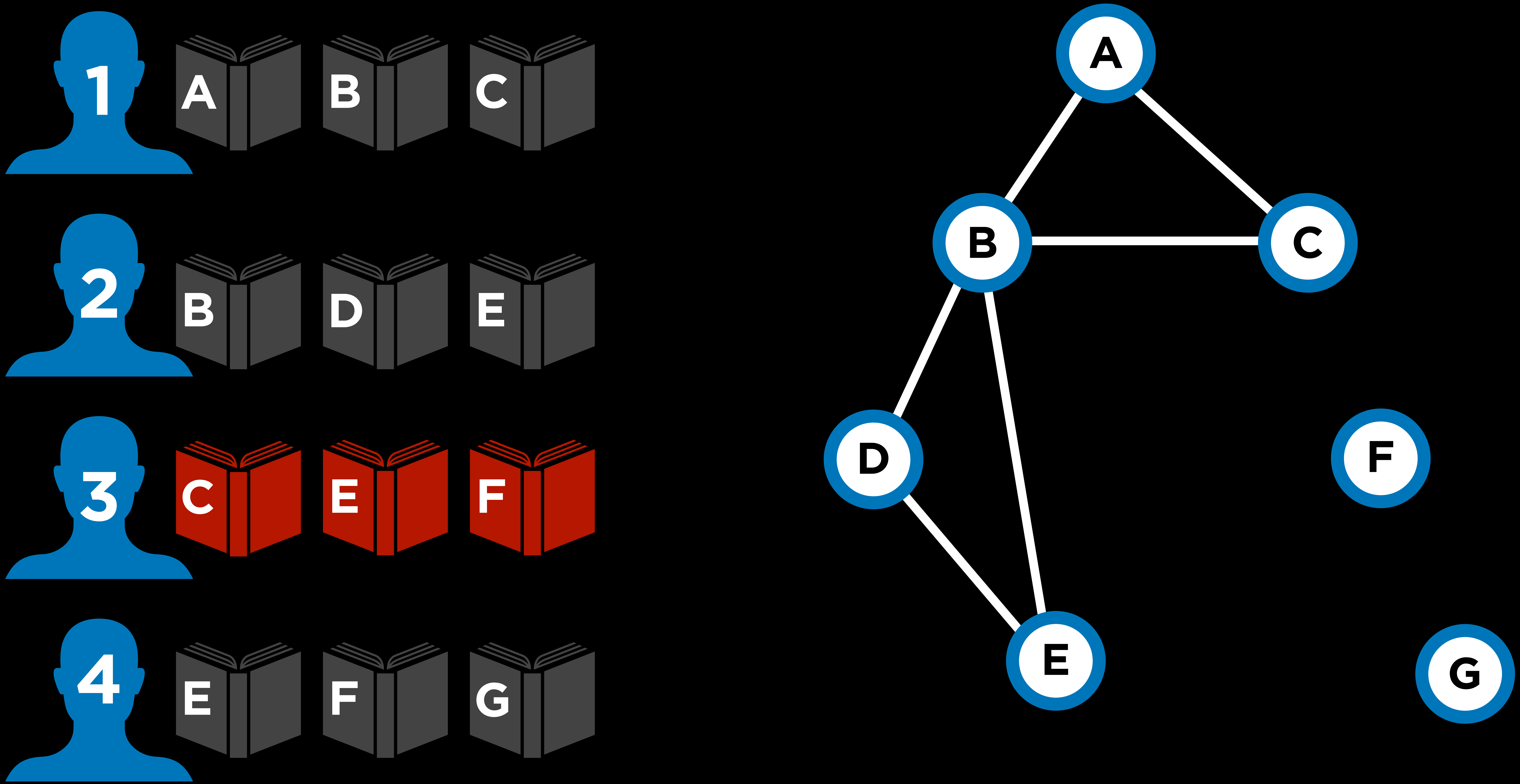

1
A
B
C
2
B
D
E
3
C
E
F
4
E
F
G
A
B
C
D
F
E
G

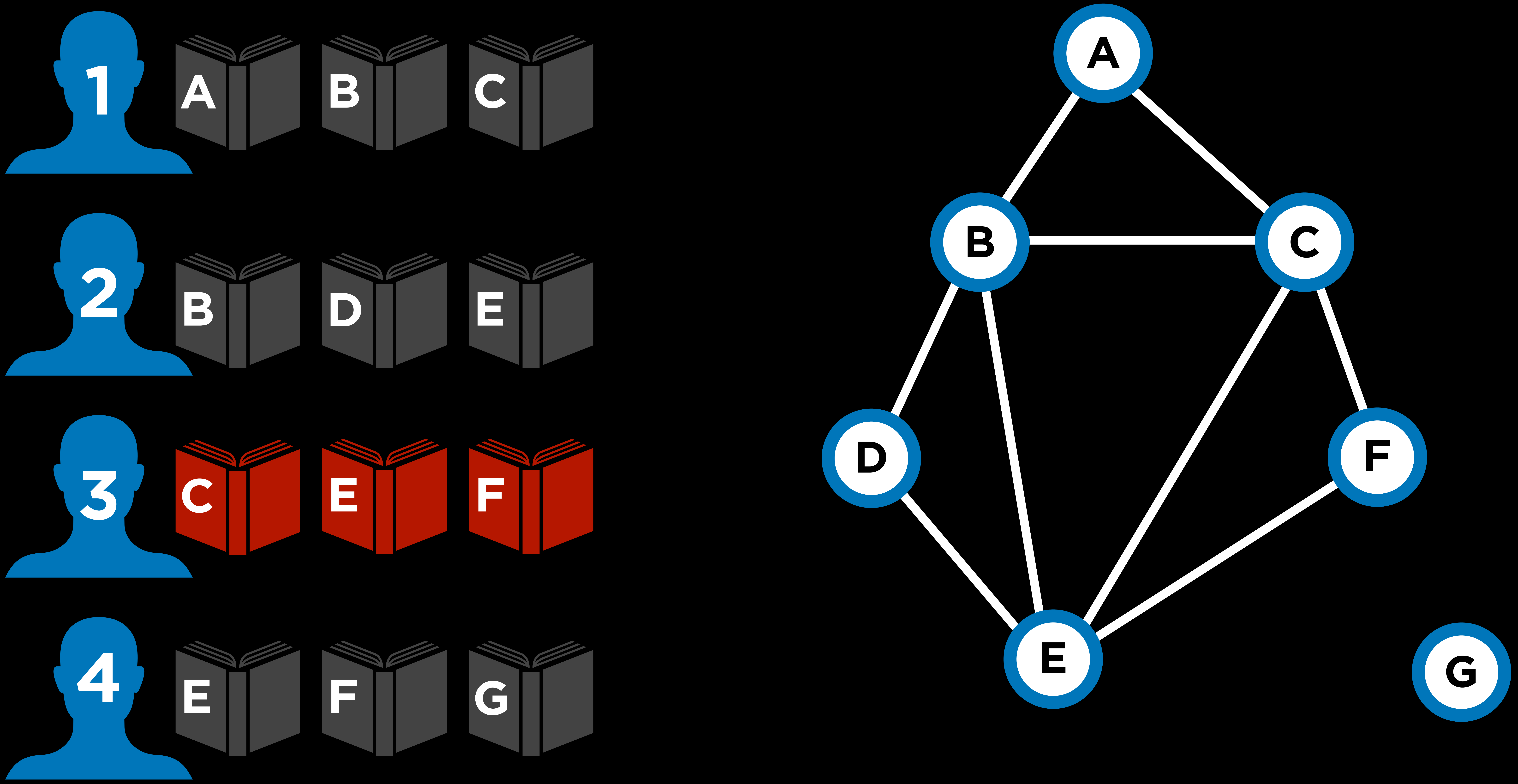

1
A
B
C
2
B
D
E
3
C
E
F
4
E
F
G
A
B
C
D
F
E
G

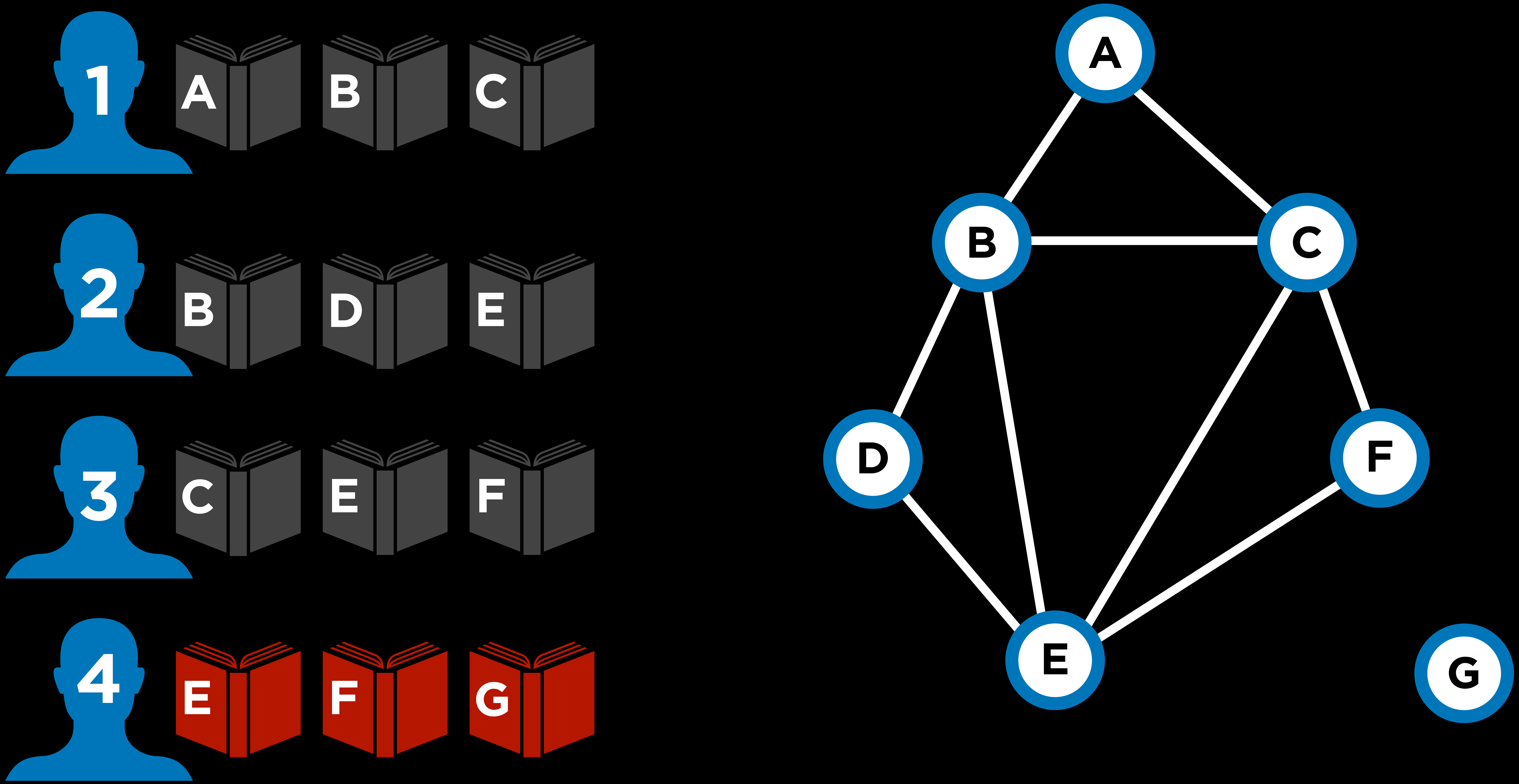

1
A
B
C
2
B
D
E
3
C
E
F
4
E
F
G
A
B
C
D
F
E
G

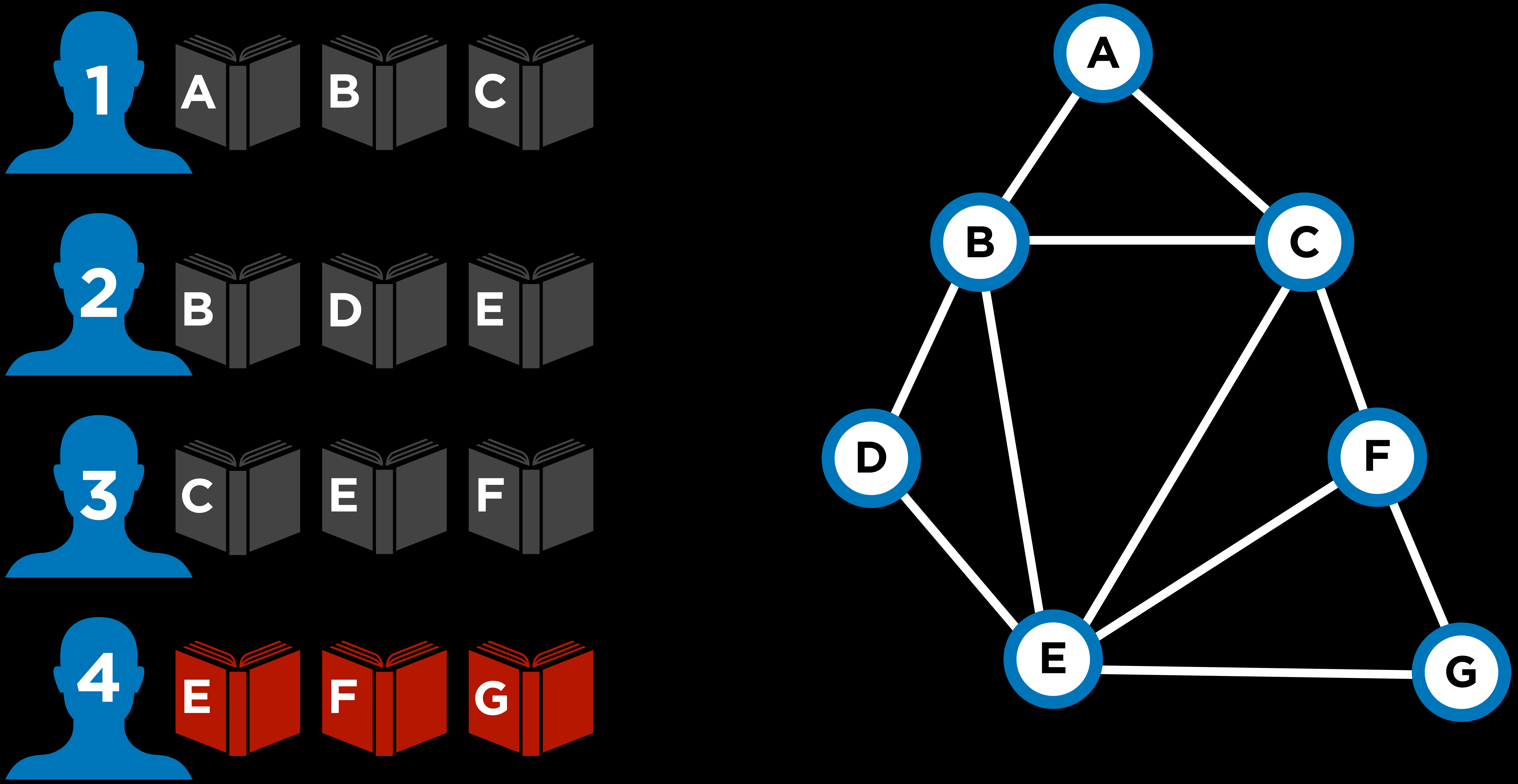


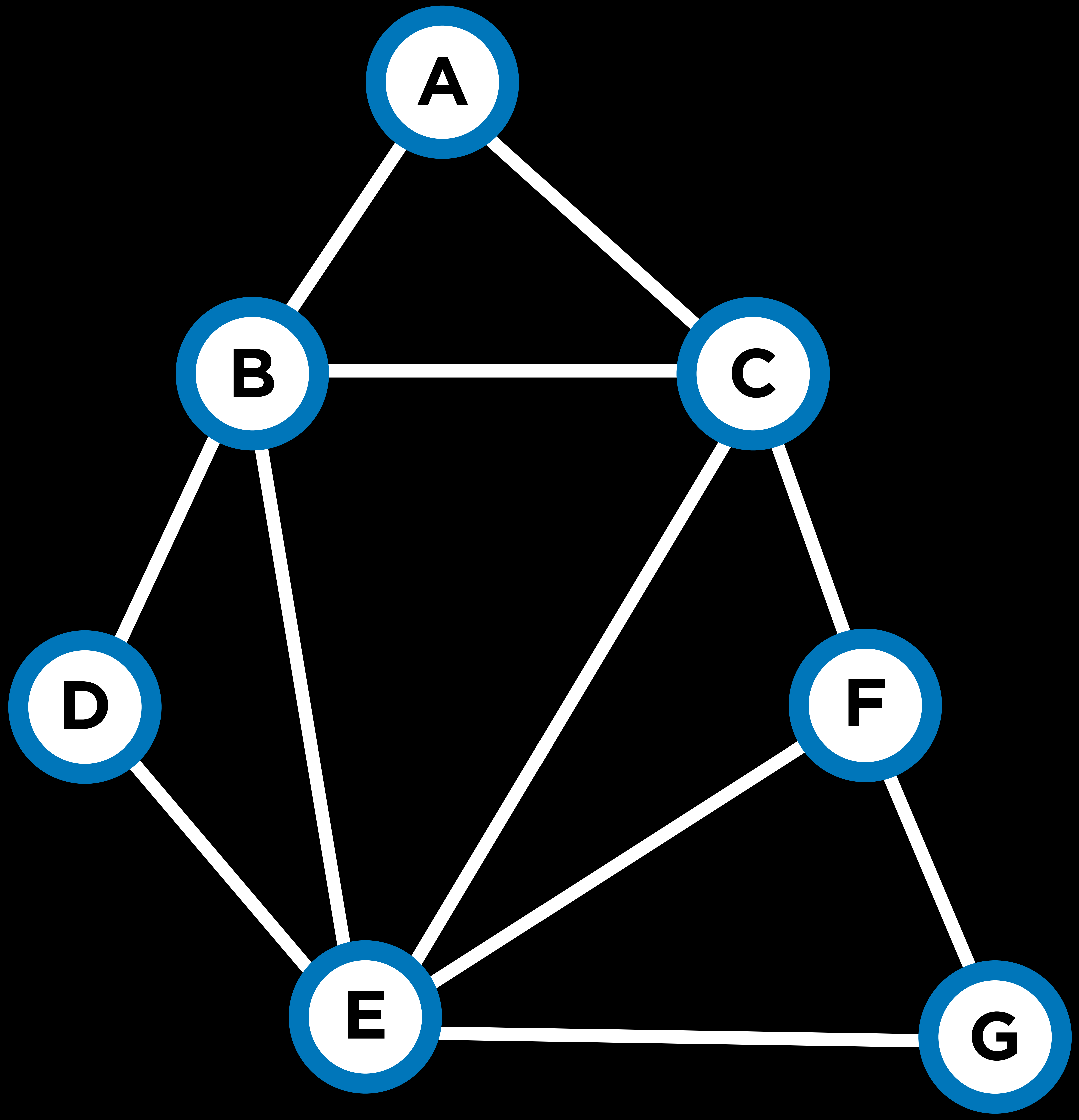
1
A
B
C
2
B
D
E
3
C
E
F
4
E
F
G
A
B
C
D
E
F
G


1
A
B
C
2
B
D
E
3
C
E
F
4
E
F
G
A
B
C
D
E
F
G


1
A
B
C
2
B
D
E
3
C
E
F
4
E
F
G
A
B
C
D
F
E
G


1
A
B
C
2
B
D
E
3
C
E
F
4
E
F
G
A
B
C
D
F
E
G


1
A
B
C
2
B
D
E
3
C
E
F
4
E
F
G
A
B
C
D
F
E
G


A
B
C
D
E
F
G

Constraint Satisfaction Problem

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

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

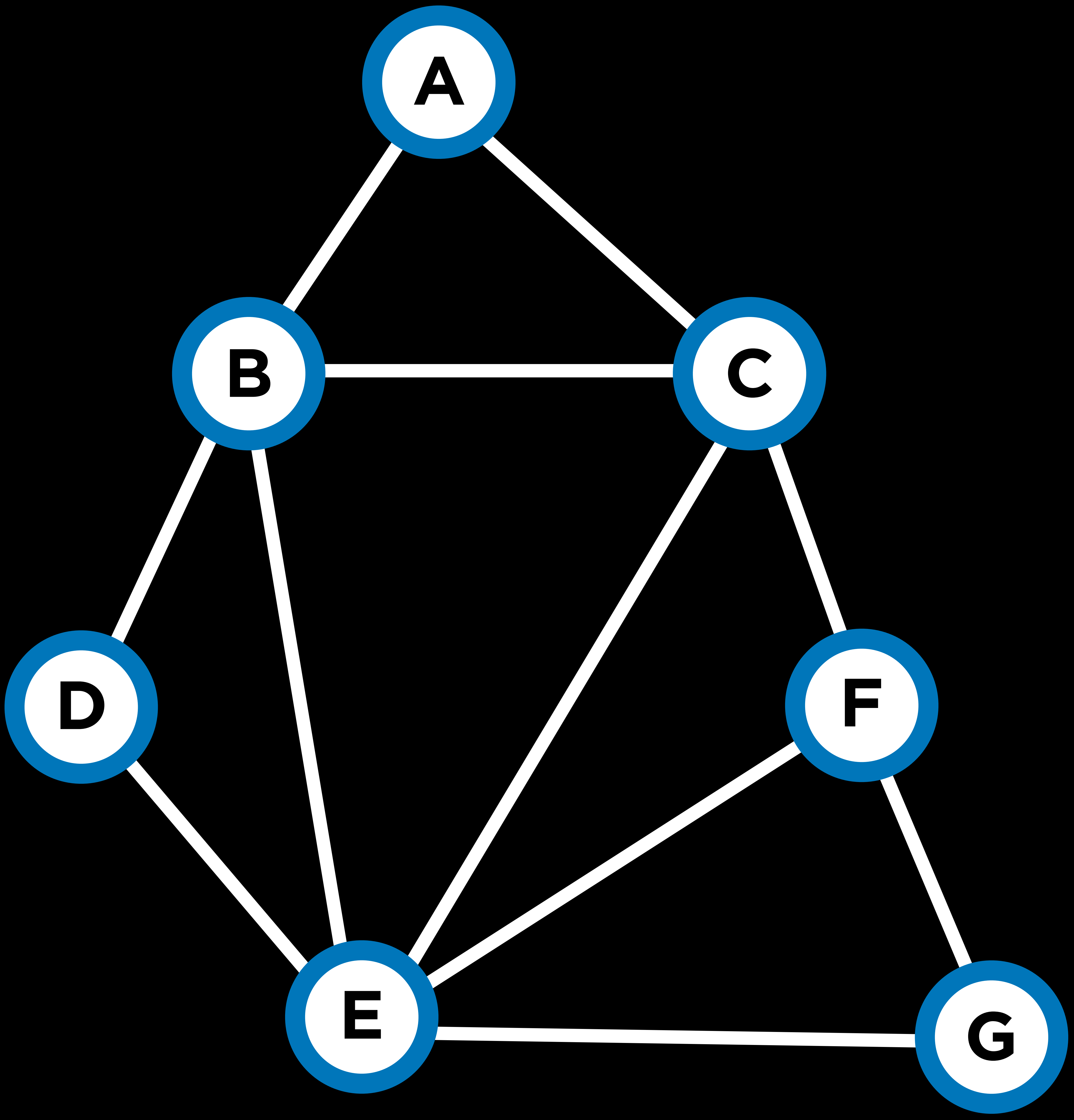
Domains

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

for each variable

Constraints

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



Variables

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

Domains

$\{Monday, Tuesday, Wednesday\}$

for each variable

Constraints

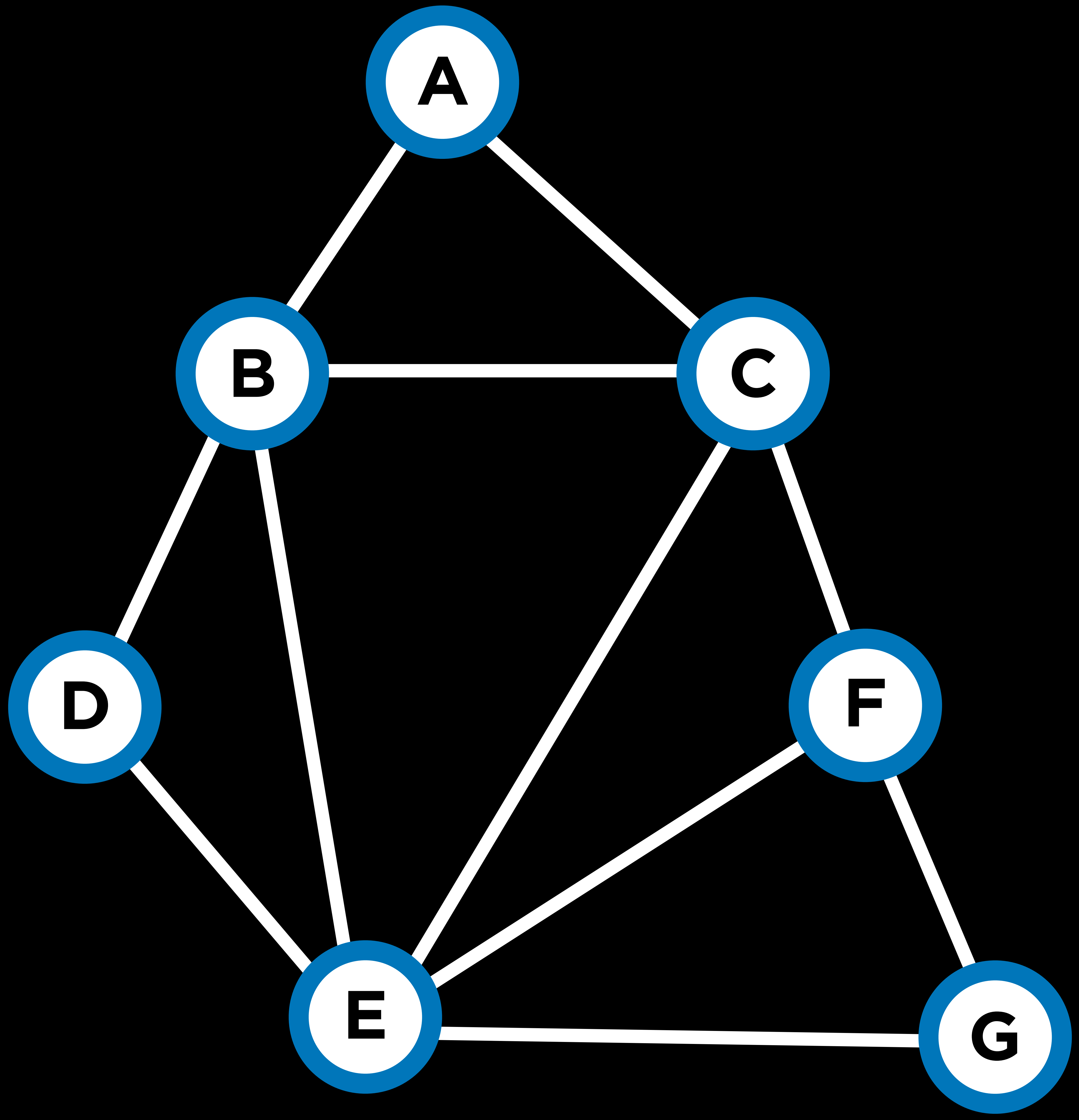
$\{A \neq B, A \neq C, B \neq C, B \neq D, B \neq E, C \neq E,$
$C \neq F, D \neq E, E \neq F, E \neq G, F \neq G\}$

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


A
B
C
D
E
F
G

unary constraint

constraint involving only one variable

unary constraint

$\{A \neq Monday\}$

binary constraint

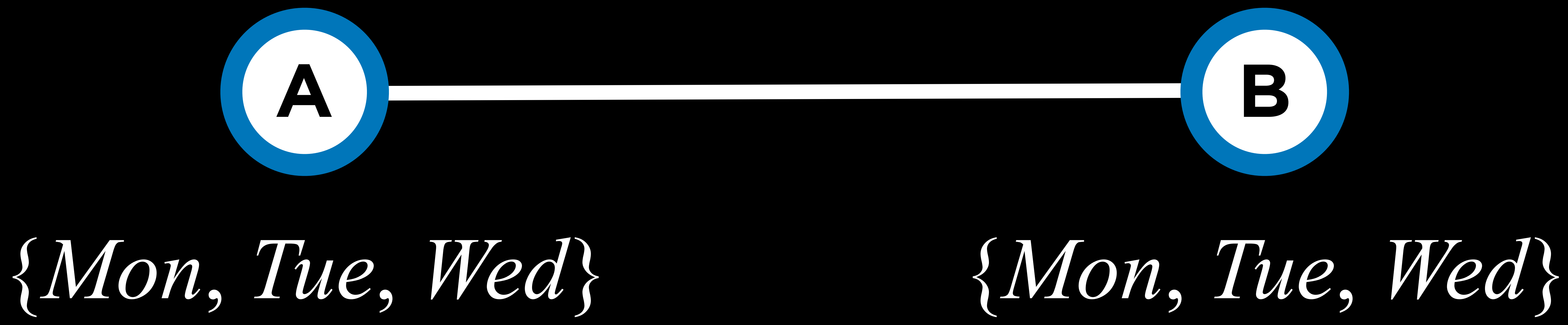
constraint involving two variables

binary constraint

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

node consistency

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



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

$\{Mon, Tue, Wed\}$ $\{Mon, Tue, Wed\}$

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

$\{Tue, Wed\}$ $\{Mon, Tue, Wed\}$

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

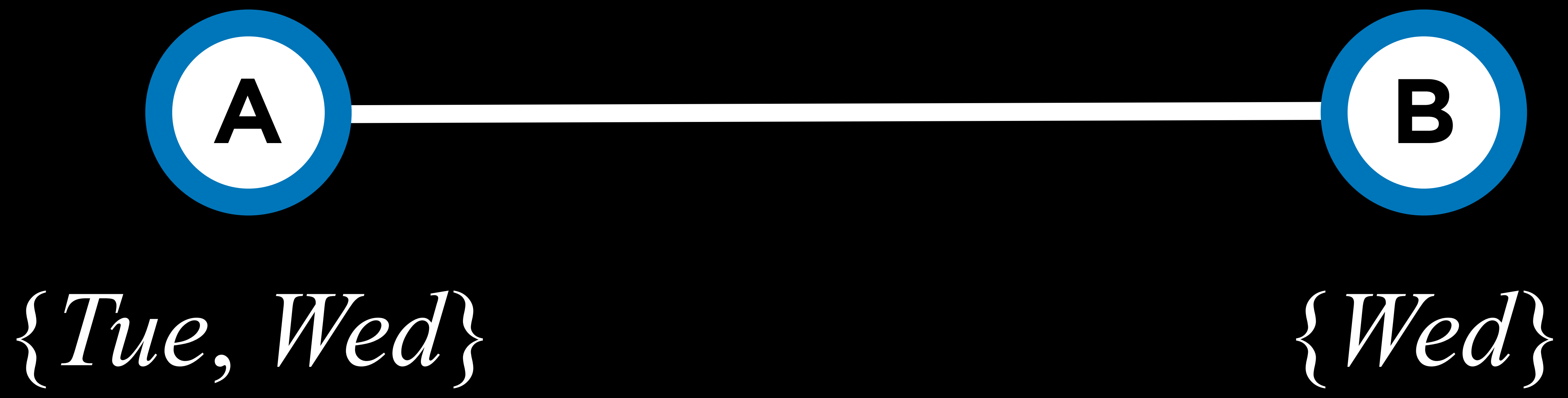
$$\{A \neq Mon,\ B \neq Tue,\ B \neq Mon,\ A \neq B\}$$

$\{Tue, Wed\}$ $\{Mon, Wed\}$

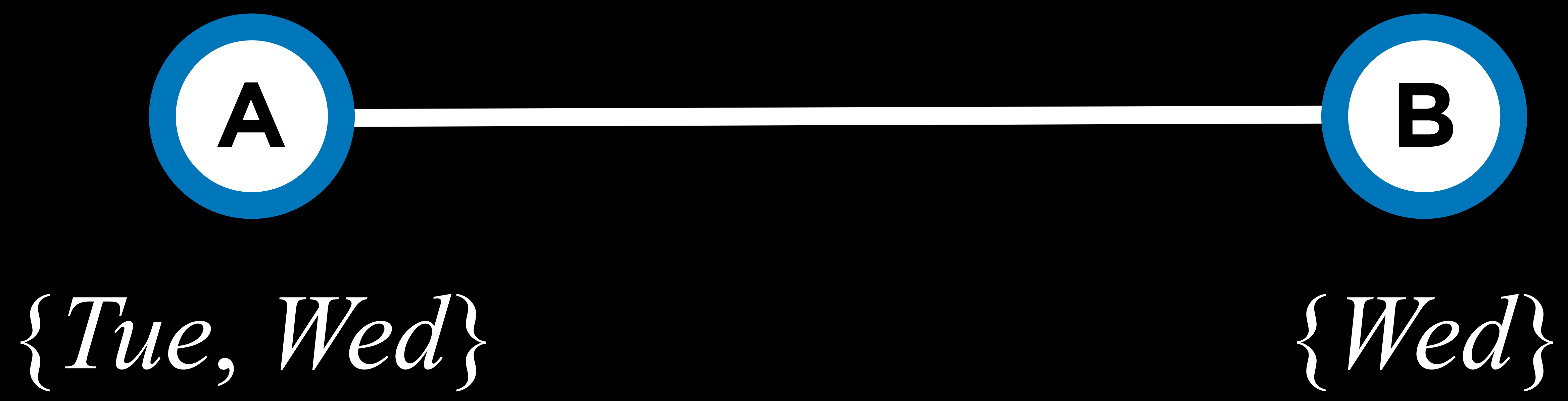
$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$

$\{Tue, Wed\}$ $\{Mon, Wed\}$

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



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



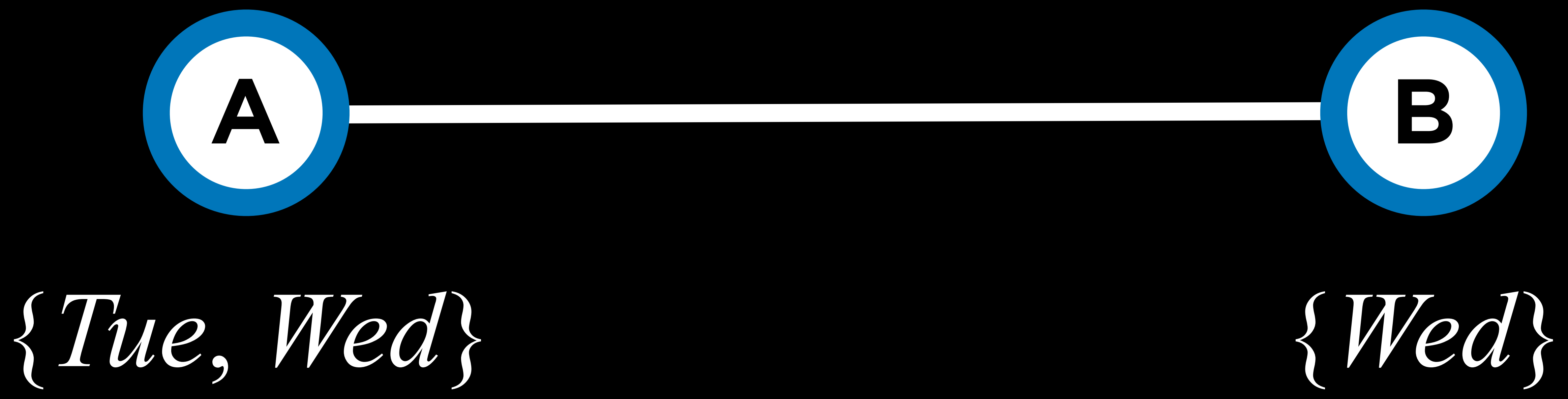
$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$

arc consistency

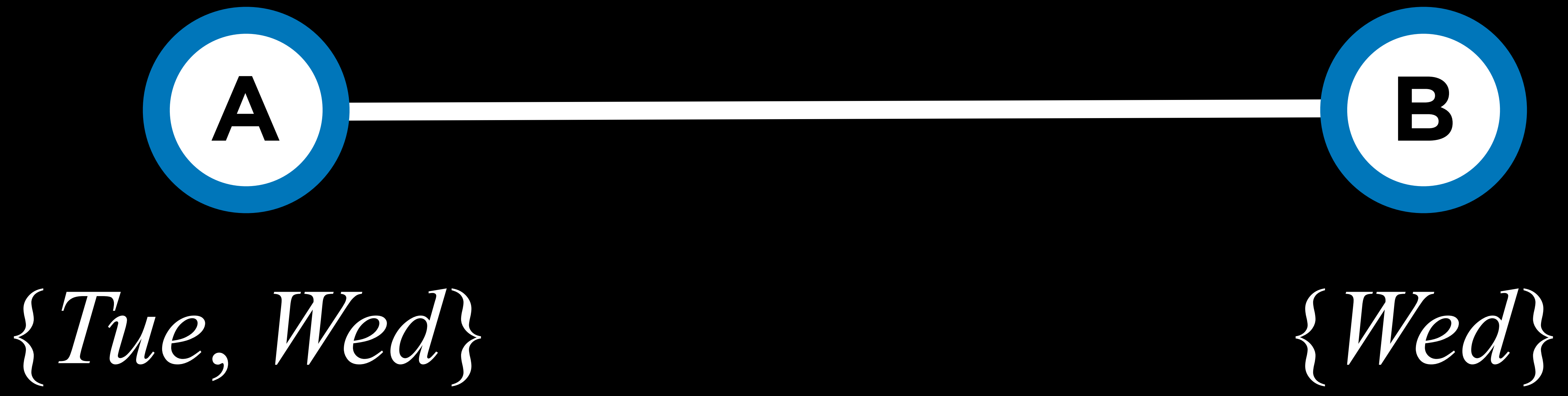
when all the values in a variable's domain satisfy the variable's binary constraints

arc consistency

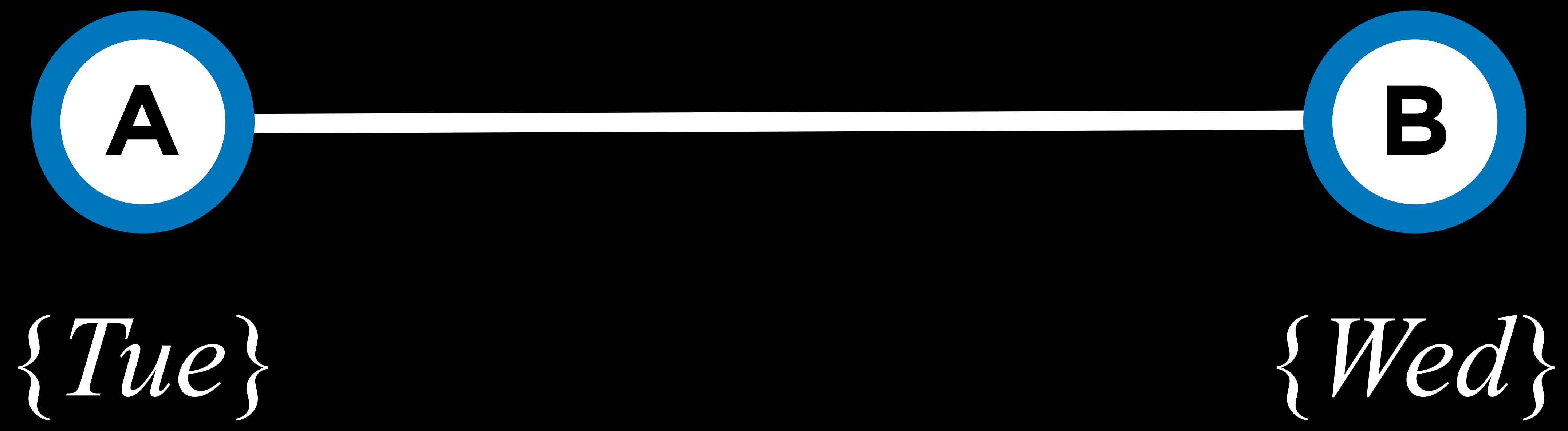
To make X arc-consistent with respect to Y, remove elements from X's domain until every choice for X has a possible choice for Y



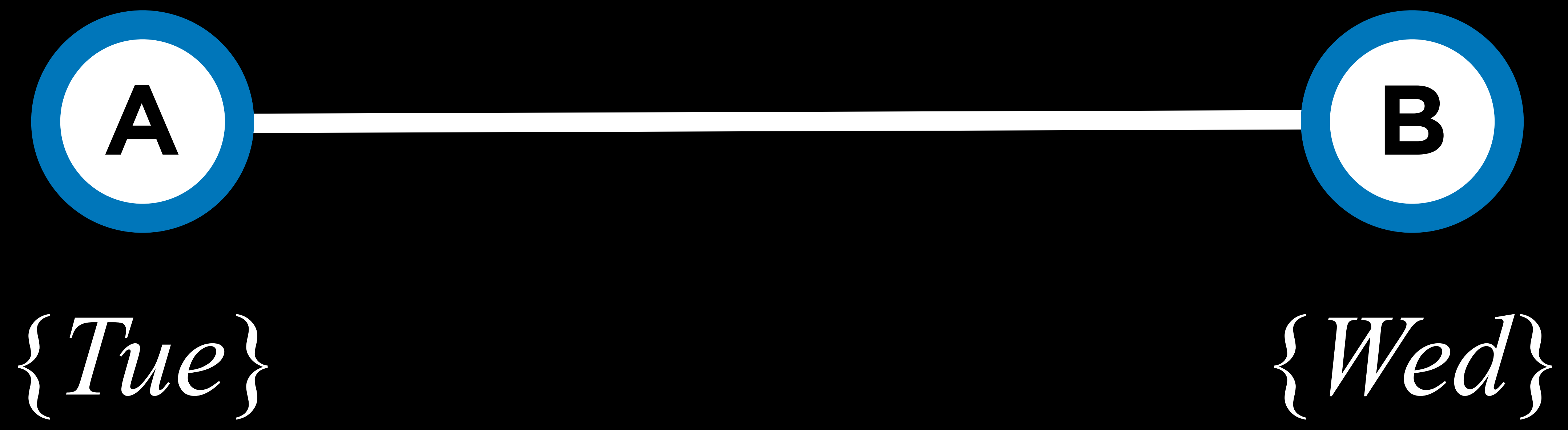
$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$



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



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



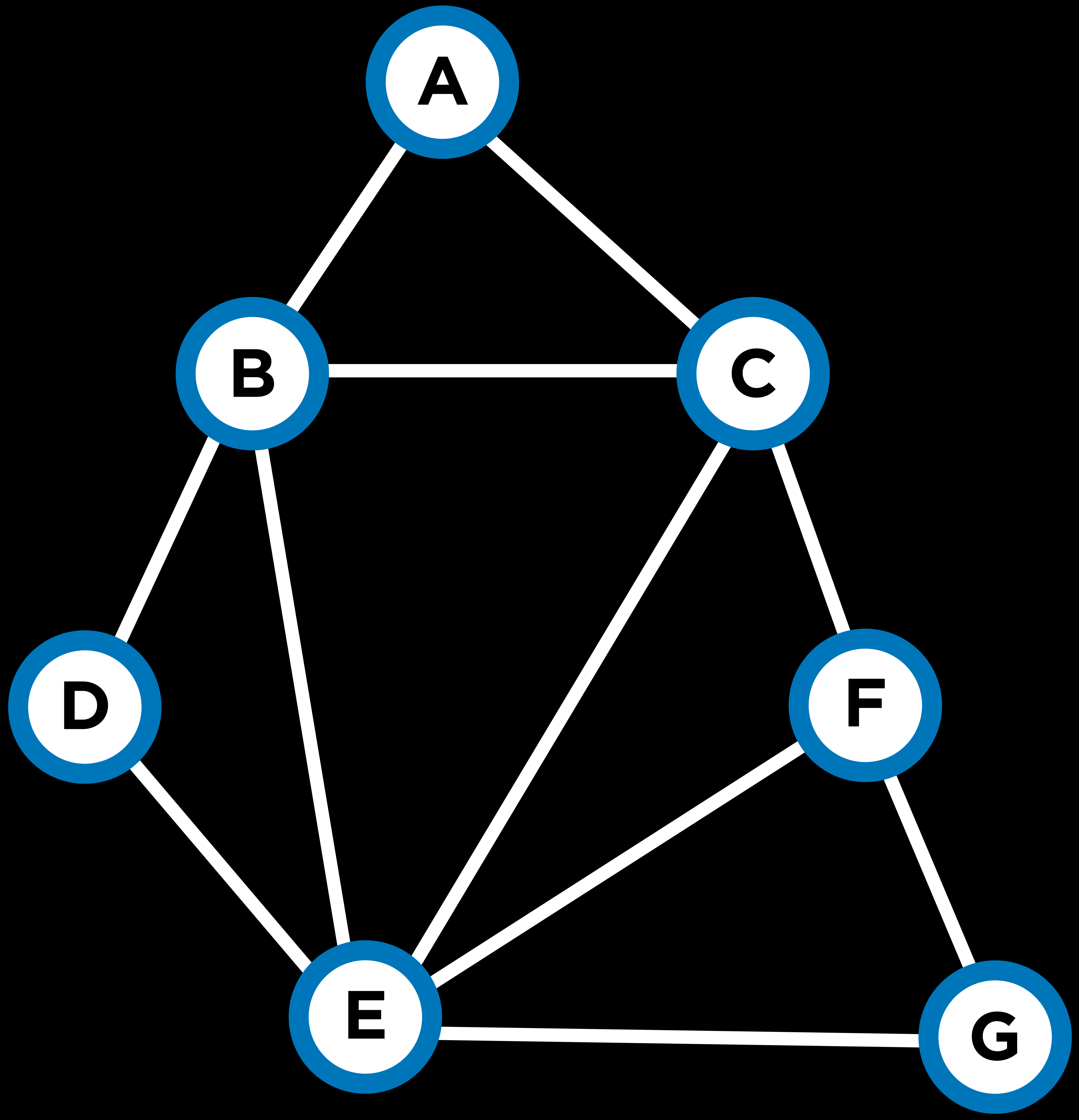
$$\{A \neq Mon, B \neq Tue, B \neq Mon, A \neq B\}$$

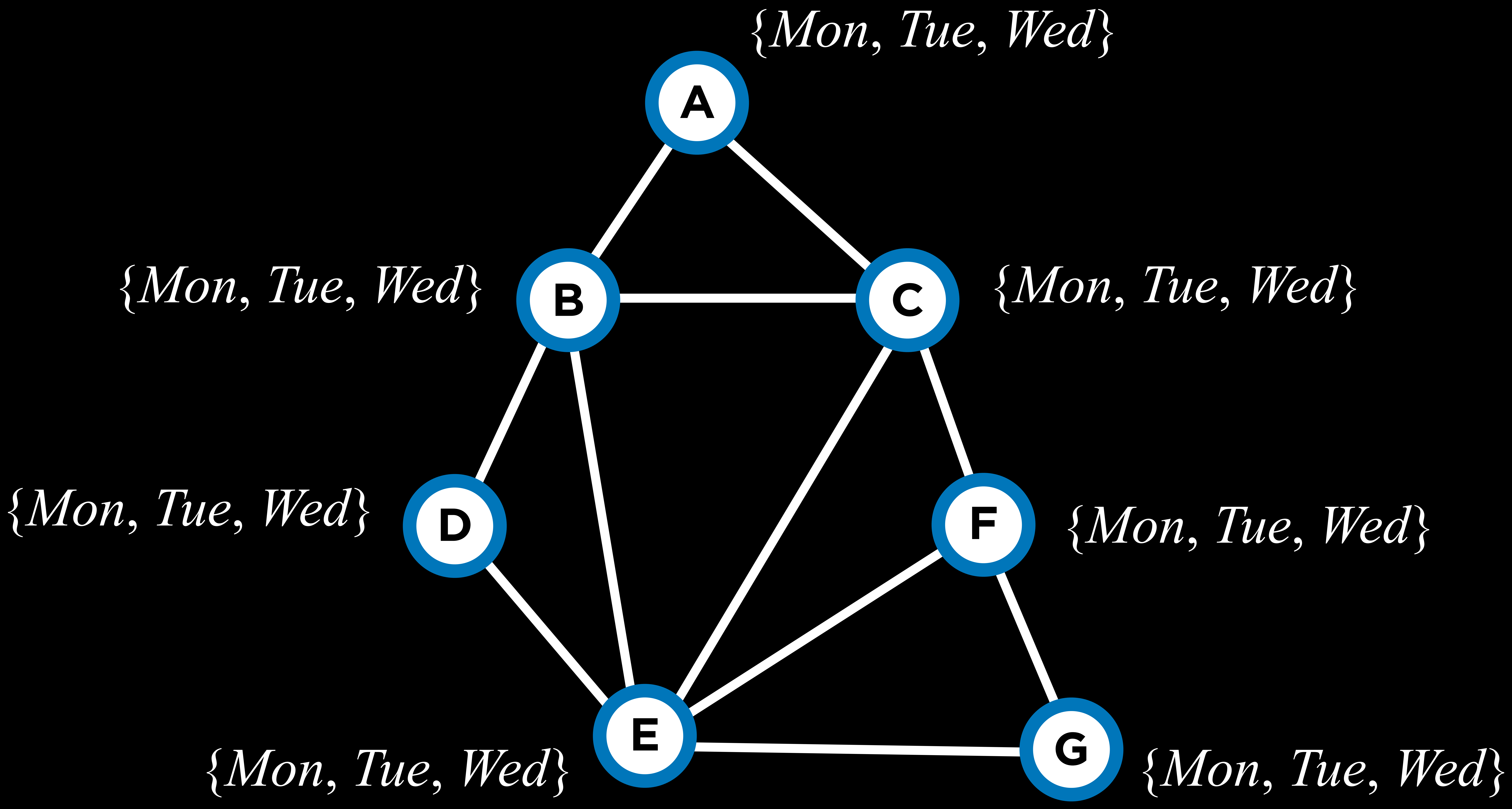
Arc Consistency

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
```


A
B
C
D
F
E
G



{Mon, Tue, Wed}
A
{Mon, Tue, Wed}
B
C
{Mon, Tue, Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}
{Mon, Tue, Wed}

Search Problems

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

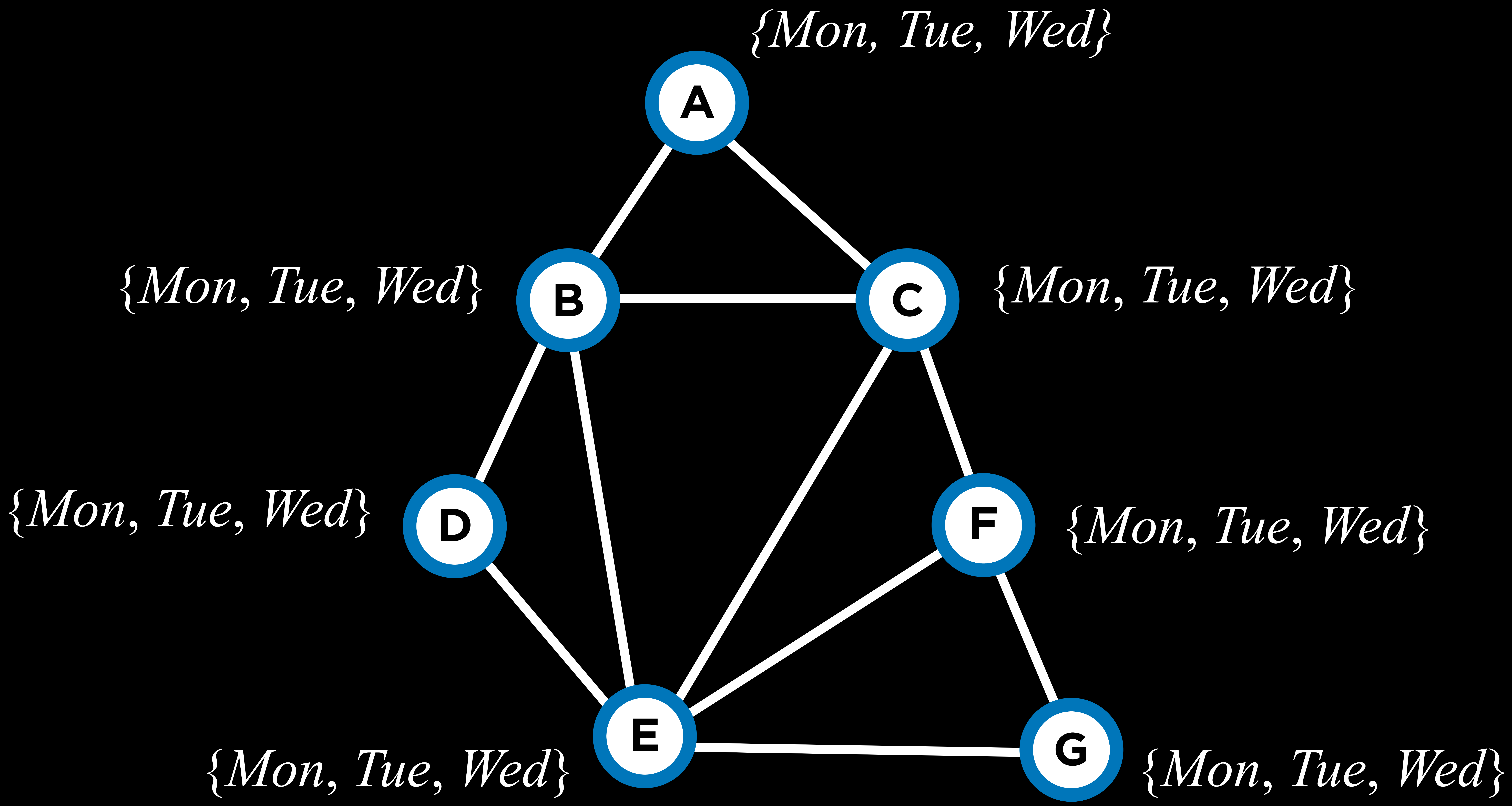
CSPs as Search Problems

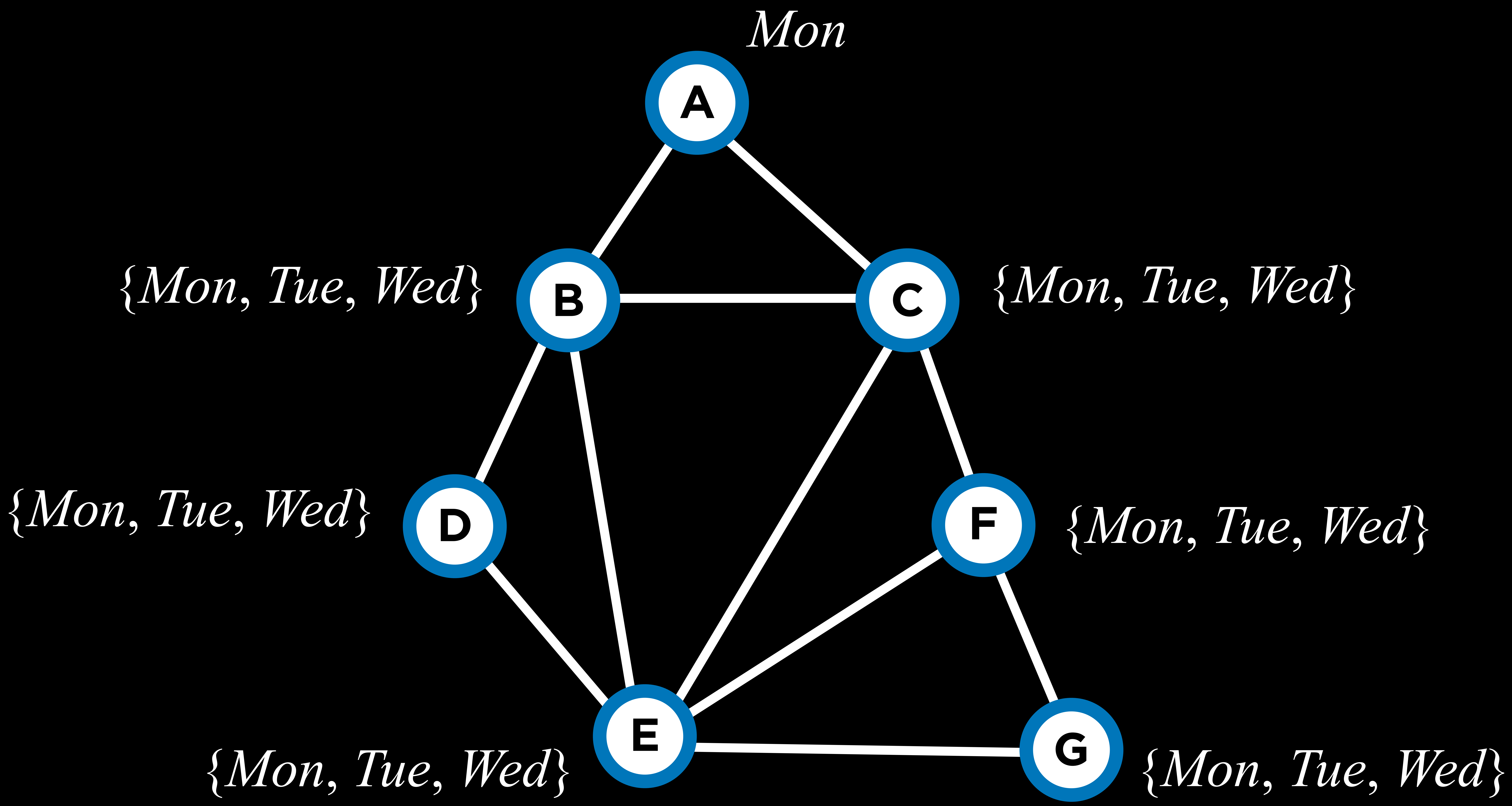
- initial state: empty assignment (no variables)
- actions: add a $\{variable = value\}$ to assignment
- transition model: shows how adding an assignment changes the assignment
- goal test: check if all variables assigned and constraints all satisfied
- path cost function: all paths have same cost

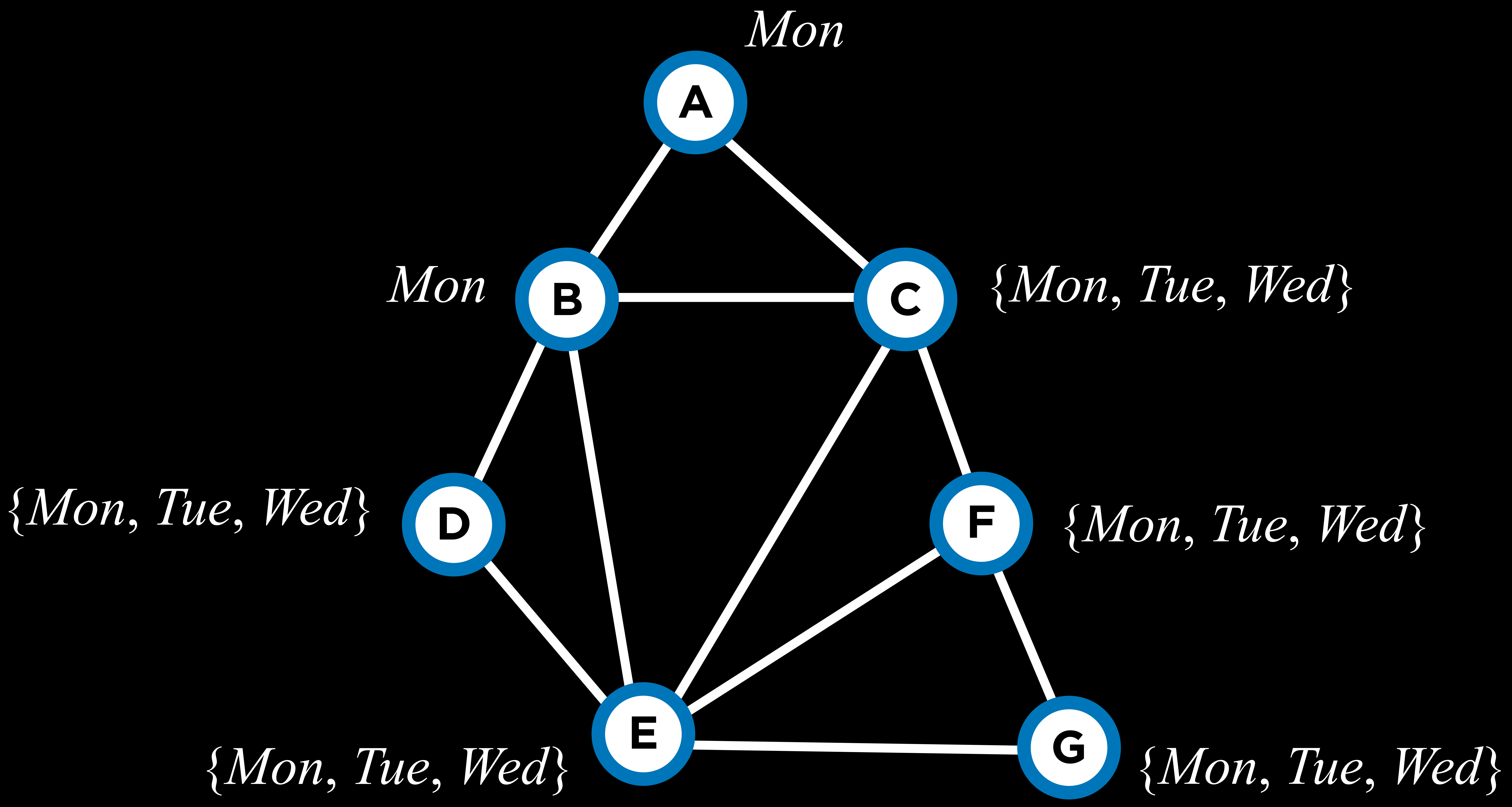
Backtracking Search

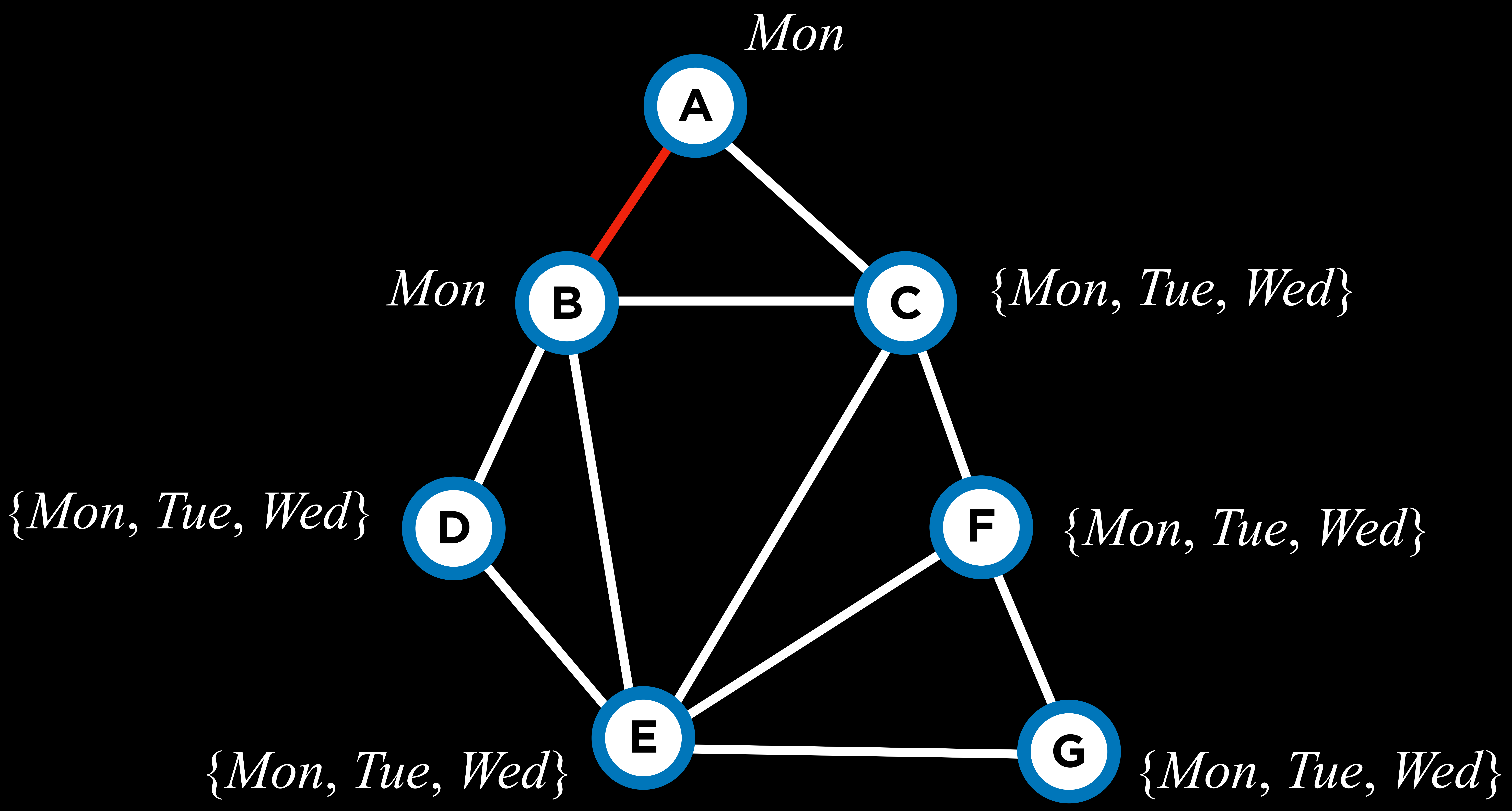
Backtracking Search

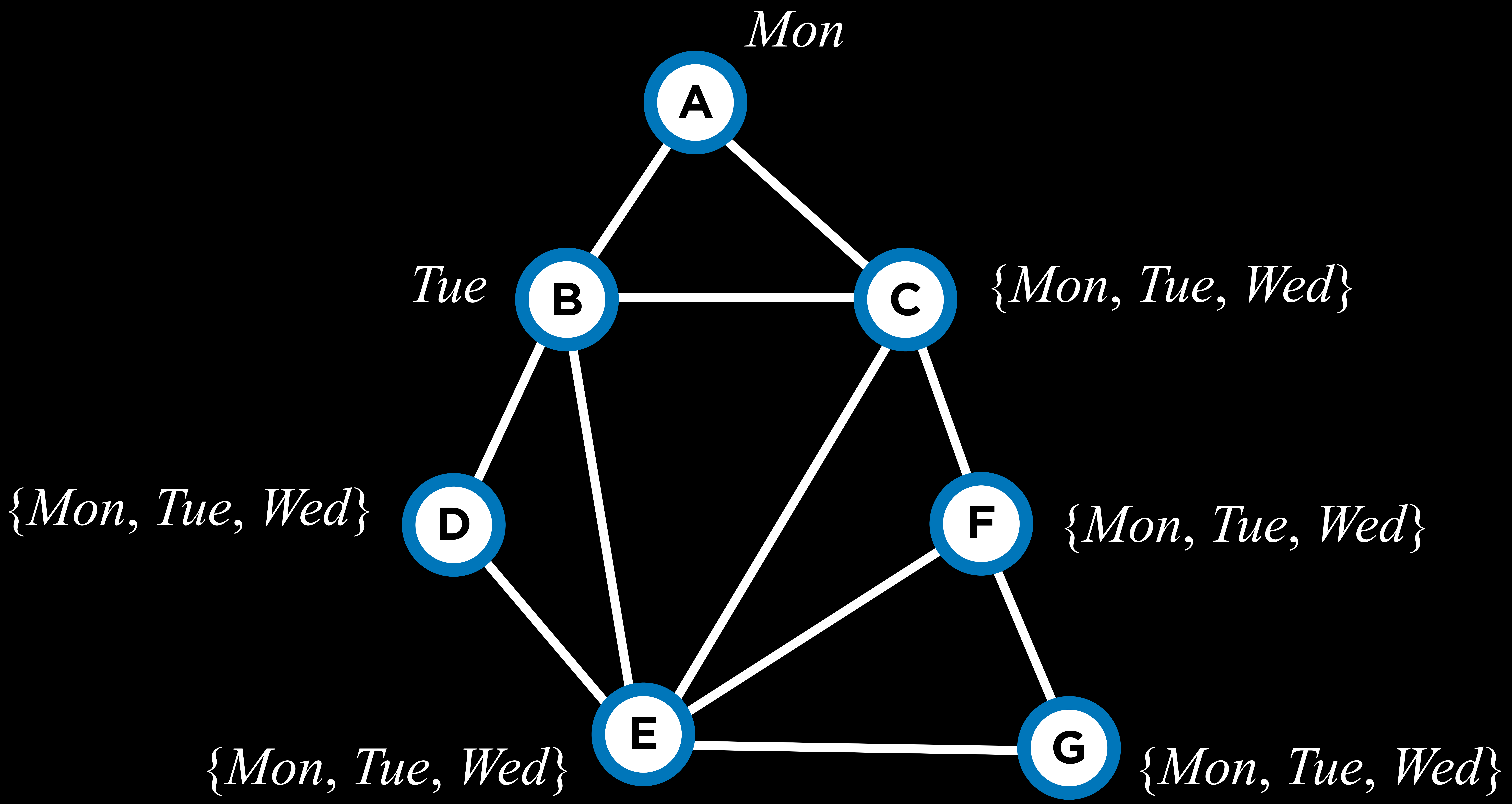
```
function BACKTRACK(assignment, csp):
    if assignment complete: return assignment
    var = SELECT-UNASSIGNED-VAR(assignment, csp)
    for value in DOMAIN-VALUES(var, assignment, csp):
        if value consistent with assignment:
            add {var = value} to assignment
            result = BACKTRACK(assignment,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

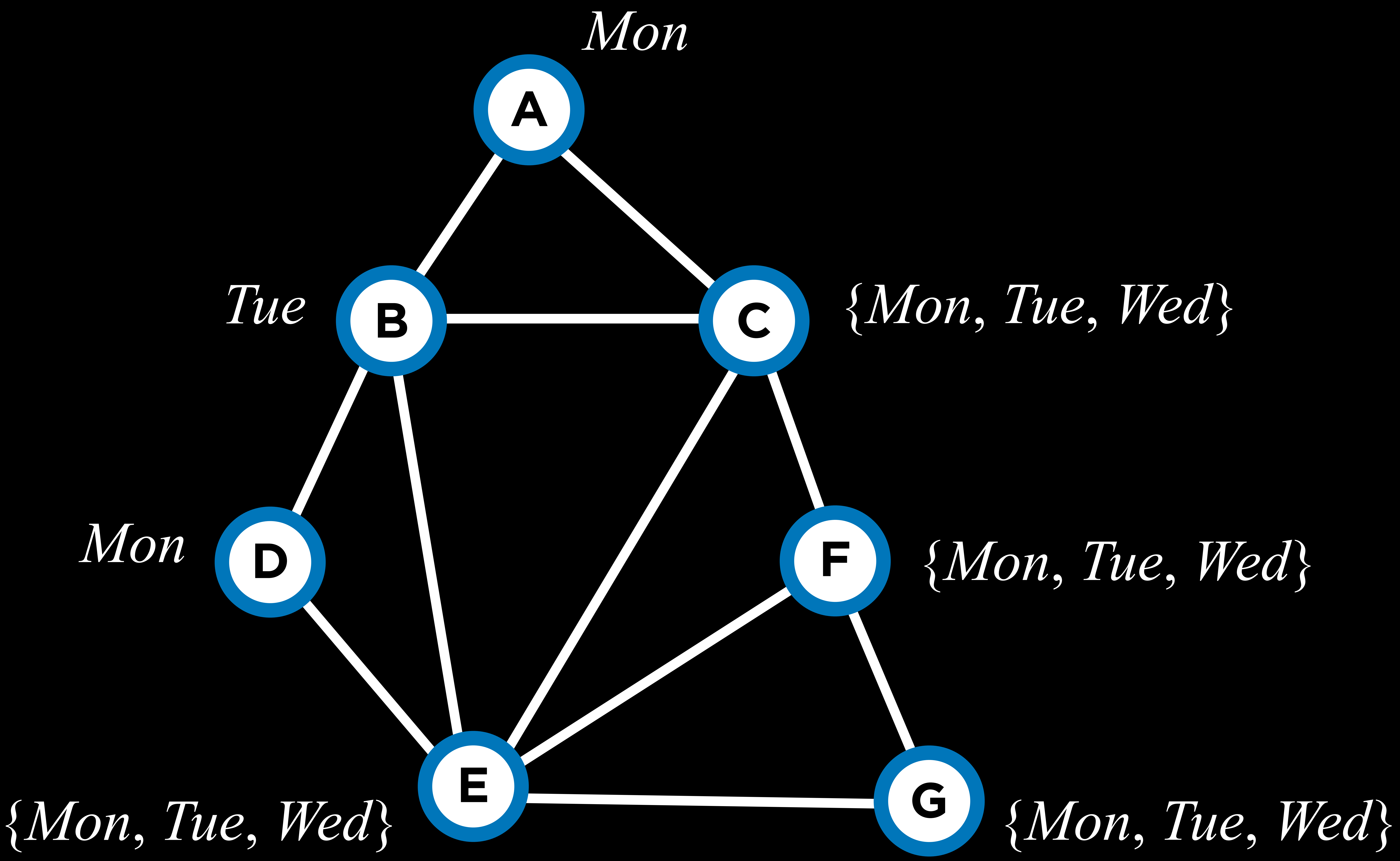

{Mon, Tue, Wed}
A
{Mon, Tue, Wed}
B
C
{Mon, Tue, Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}

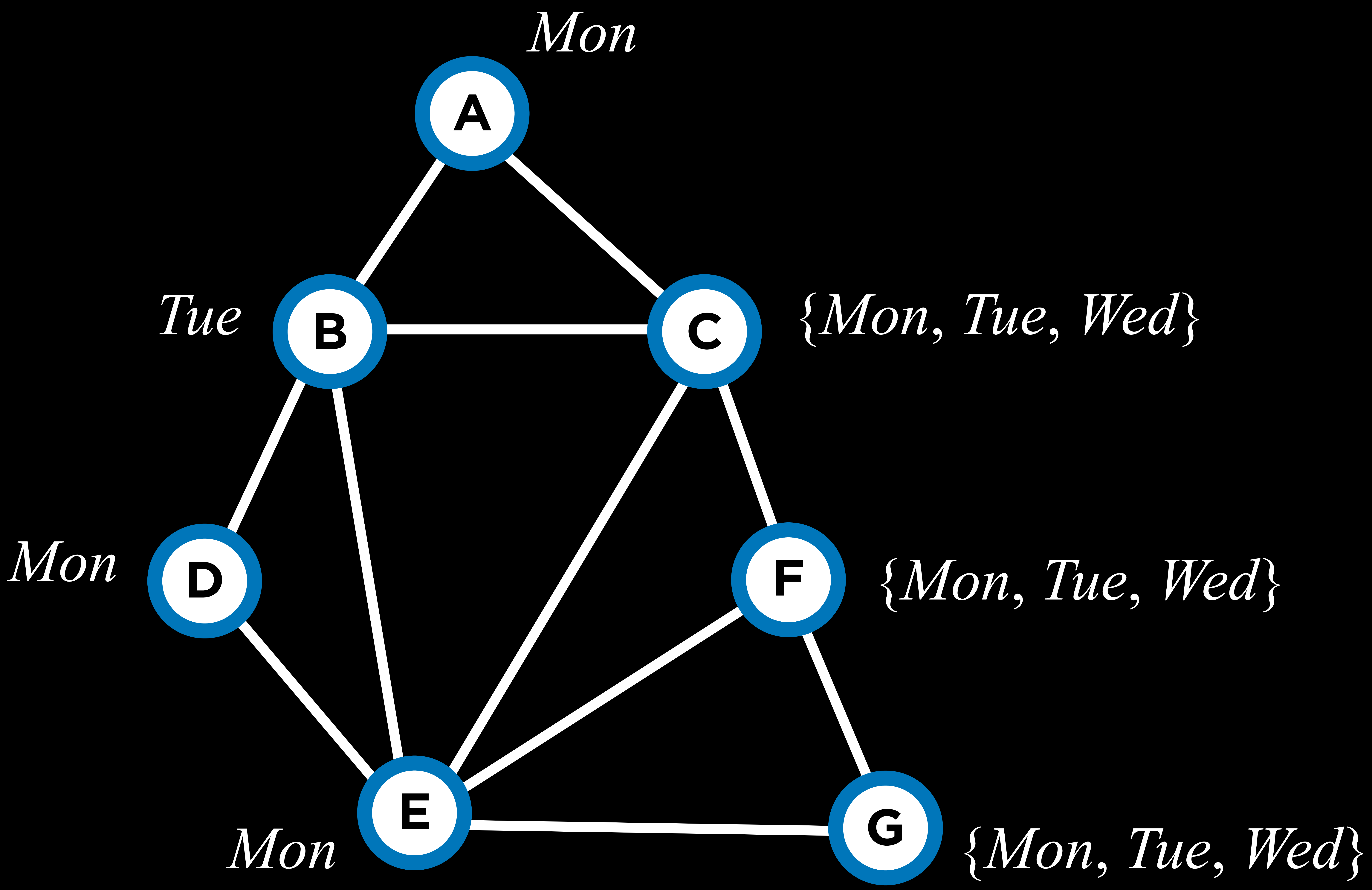

Mon
A
{Mon, Tue, Wed}
B
C
{Mon, Tue, Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}

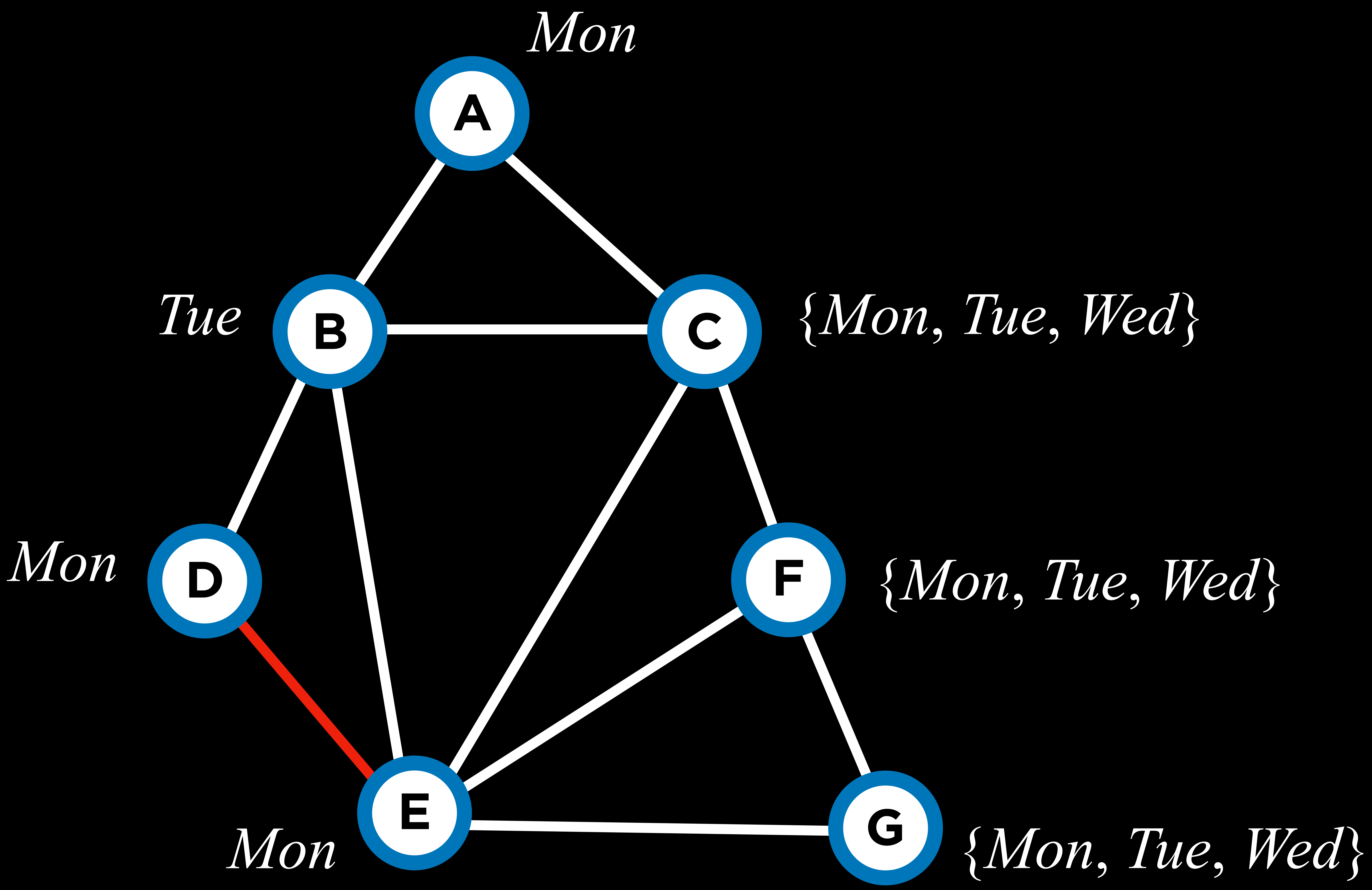

Mon
A
Mon
B
C
{Mon, Tue, Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
E
{Mon, Tue, Wed}
G
{Mon, Tue, Wed}

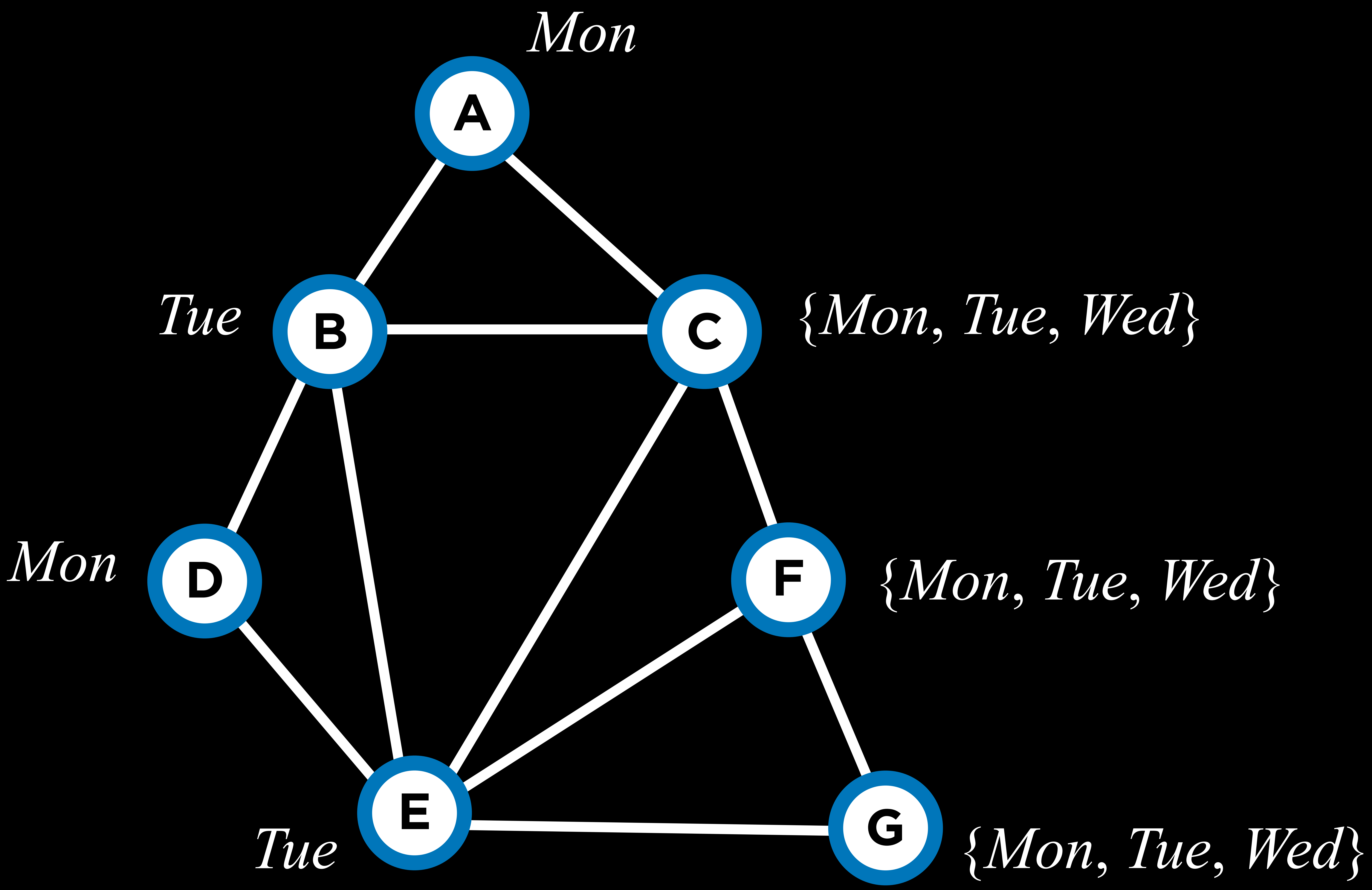

Mon
A
Mon
B
C
{Mon, Tue, Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}

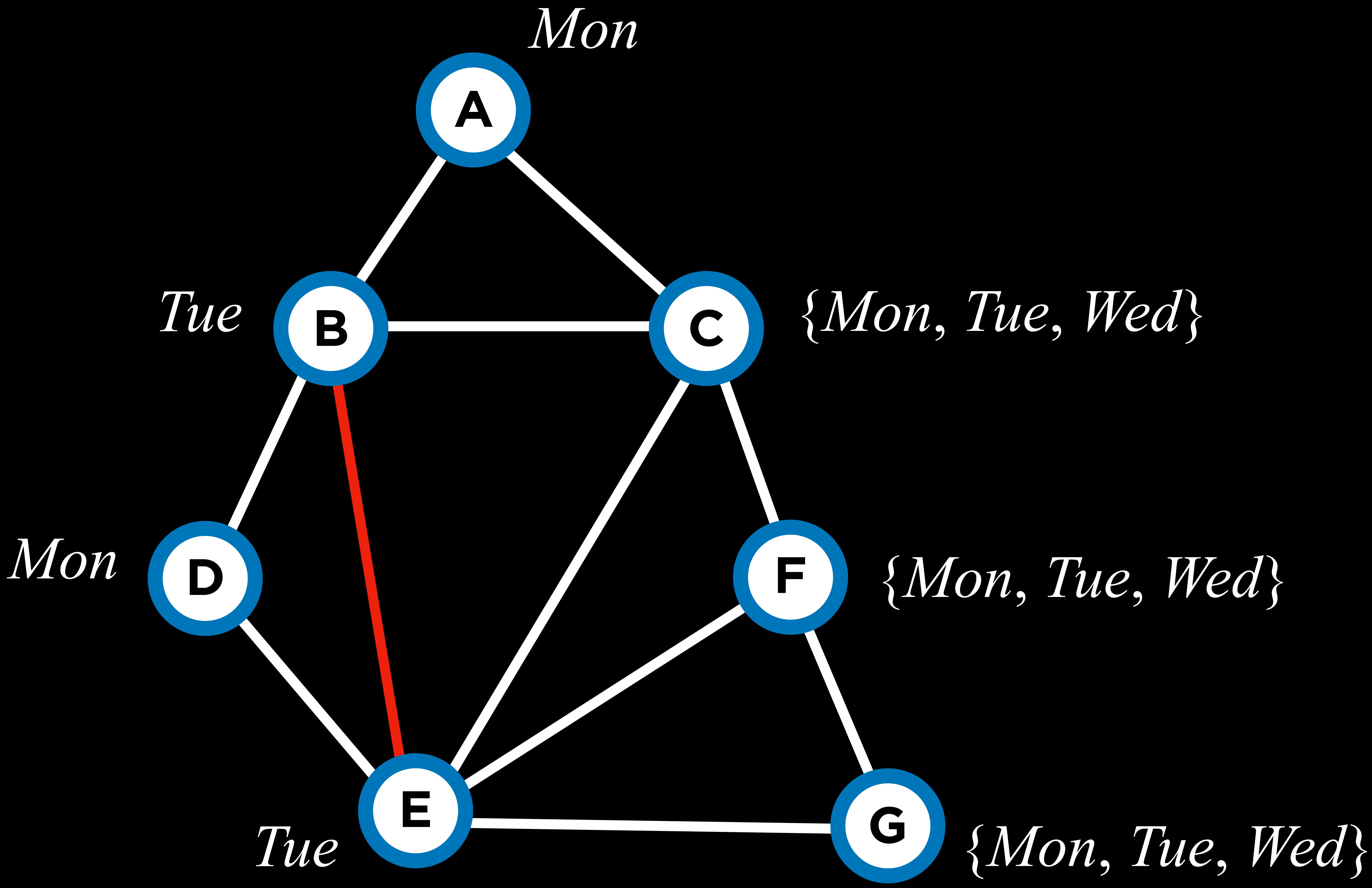

Mon
A
Tue
B
C
{Mon, Tue, Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}

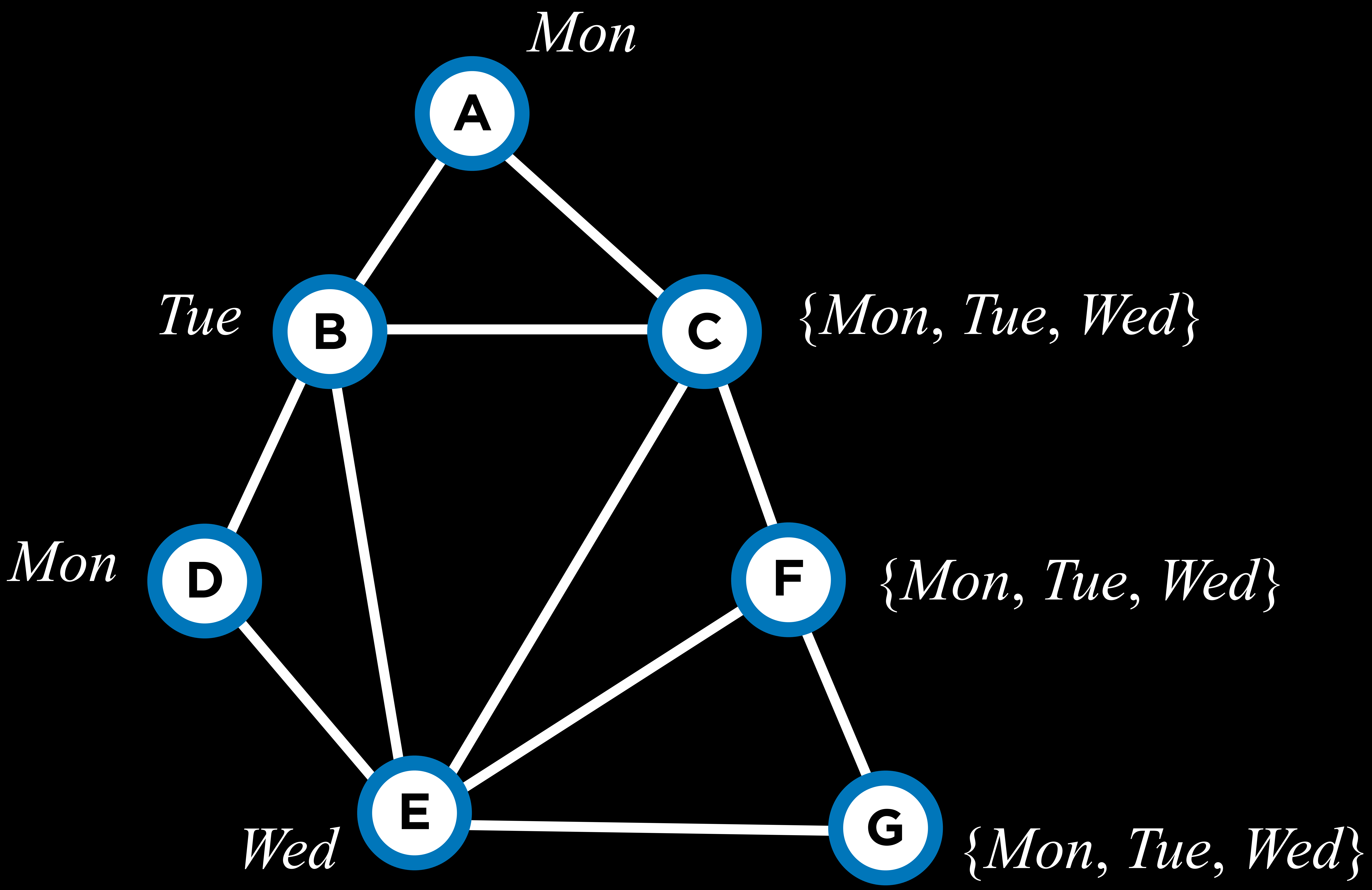

Mon
A
Tue
B
C
{Mon, Tue, Wed}
Mon
D
F
{Mon, Tue, Wed}
E
{Mon, Tue, Wed}
G
{Mon, Tue, Wed}

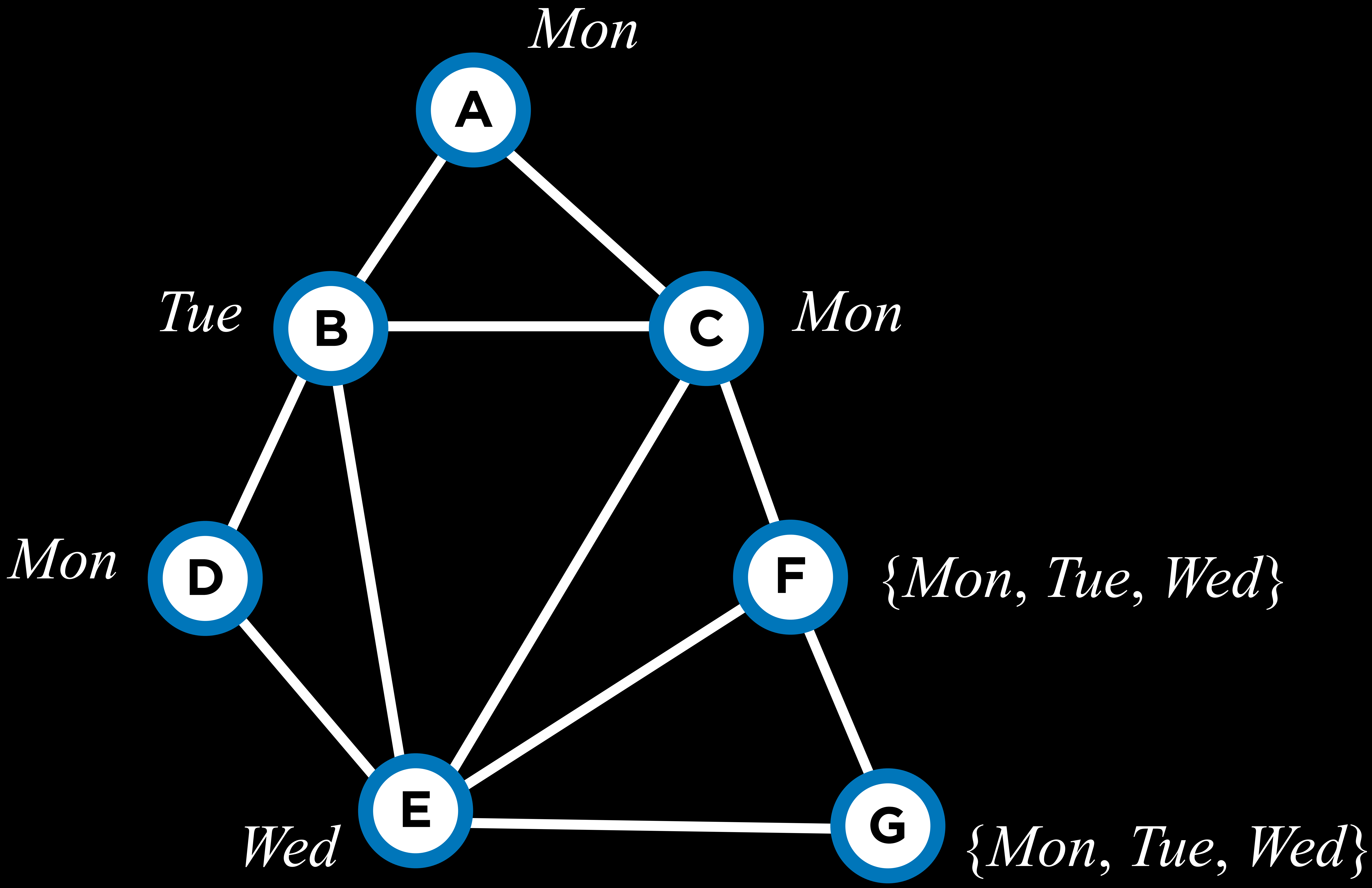

Mon
A
Tue
B
C
{Mon, Tue, Wed}
Mon
D
F
{Mon, Tue, Wed}
E
Mon
G
{Mon, Tue, Wed}


Mon
A
Tue
B
C
{Mon, Tue, Wed}
Mon
D
F
{Mon, Tue, Wed}
Mon
E
G
{Mon, Tue, Wed}

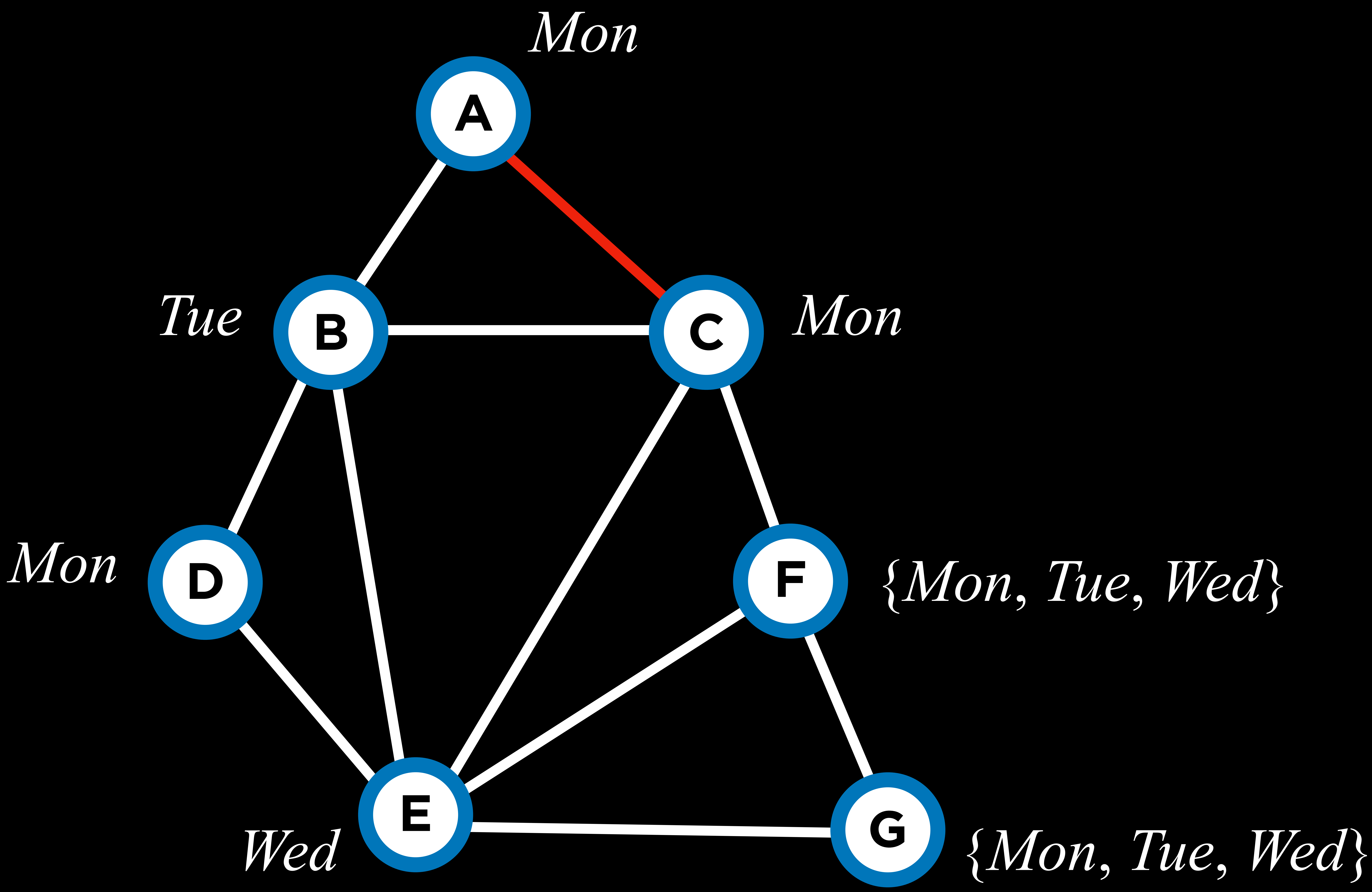

Mon
A
Tue
B
C
{Mon, Tue, Wed}
Mon
D
F
{Mon, Tue, Wed}
E
Tue
G
{Mon, Tue, Wed}


Mon
A
Tue
B
C
{Mon, Tue, Wed}
Mon
D
F
{Mon, Tue, Wed}
E
Tue
G
{Mon, Tue, Wed}

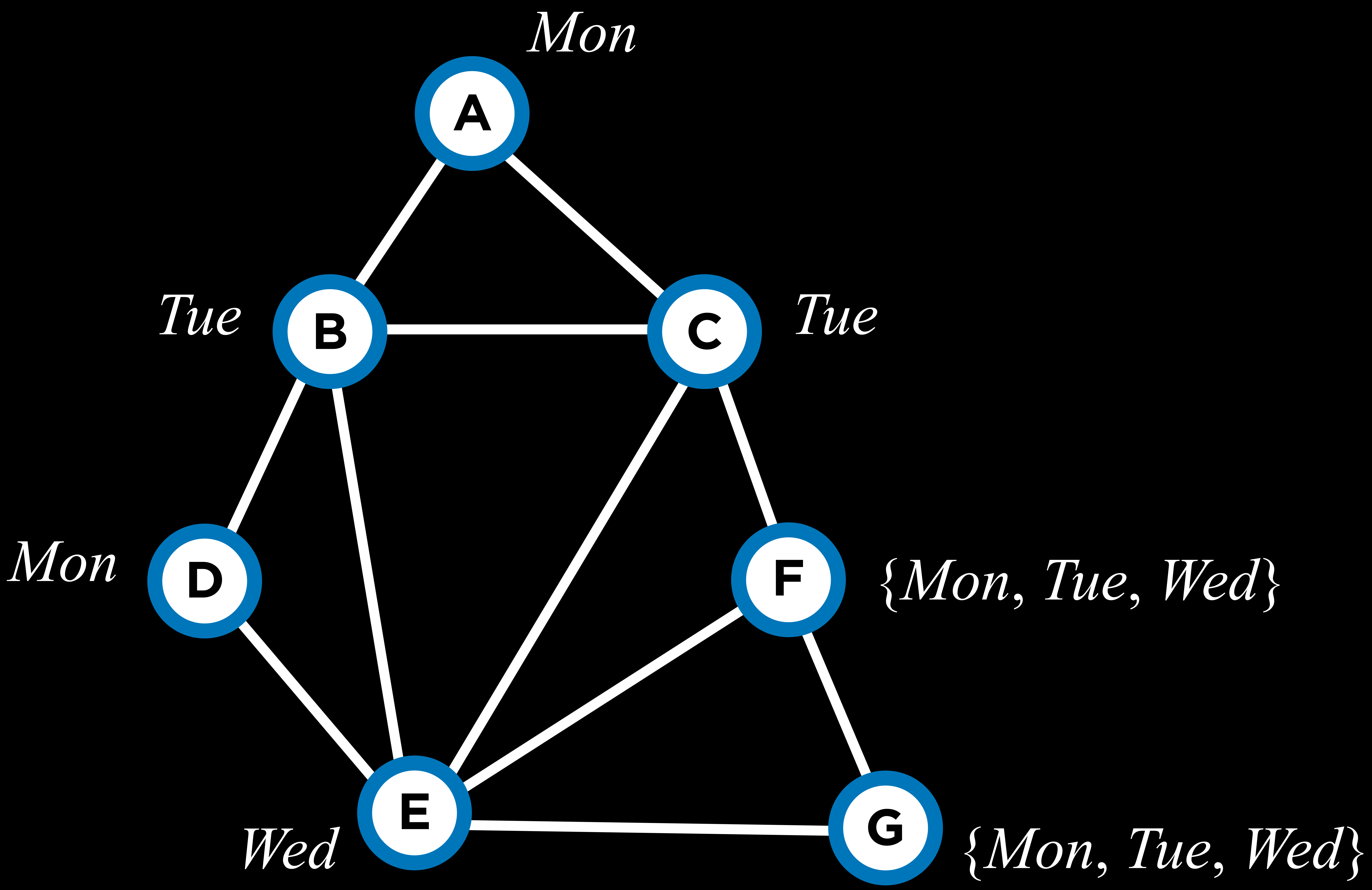

Mon
A
Tue
B
C
{Mon, Tue, Wed}
Mon
D
F
{Mon, Tue, Wed}
E
Wed
G
{Mon, Tue, Wed}

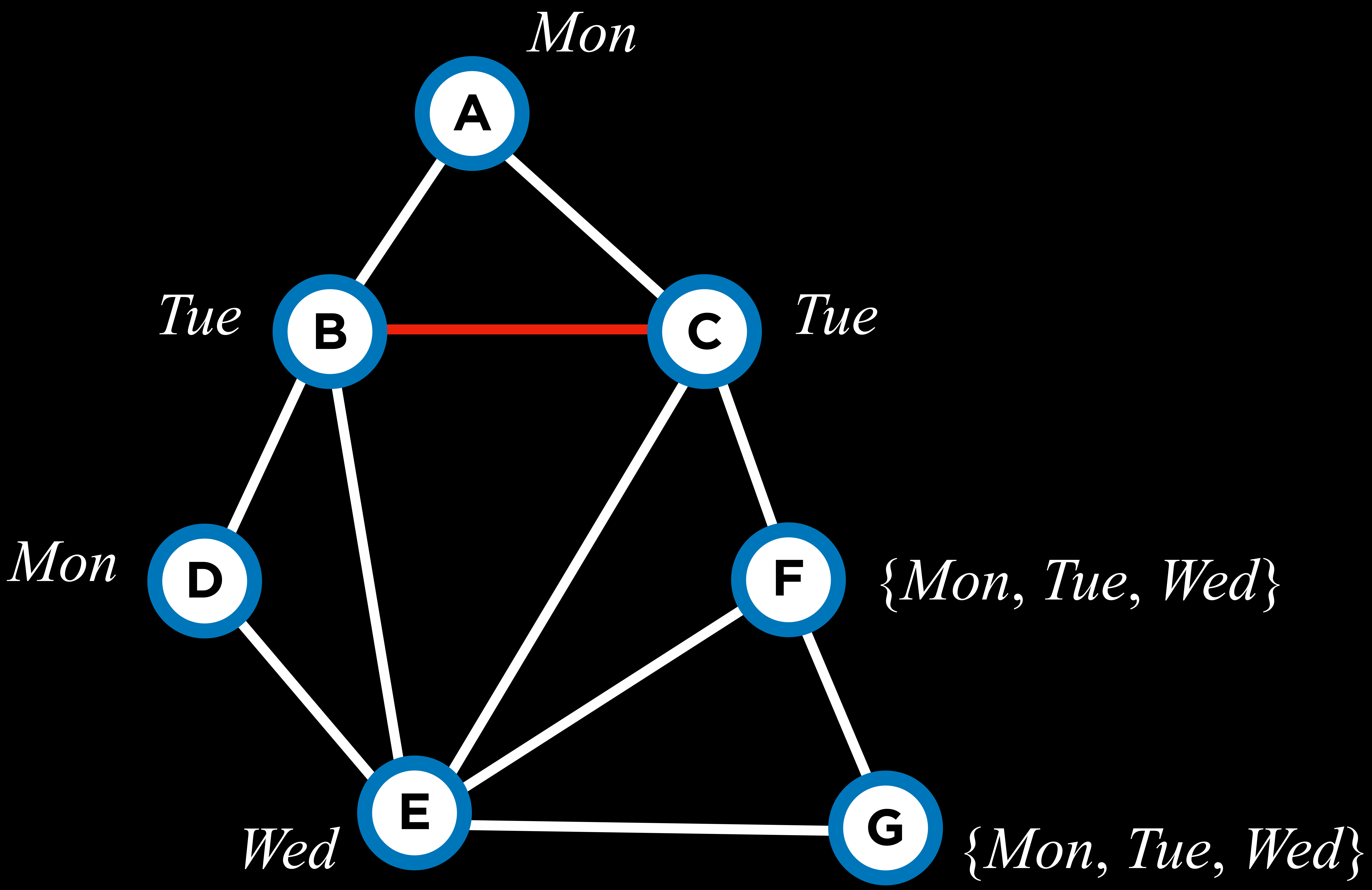


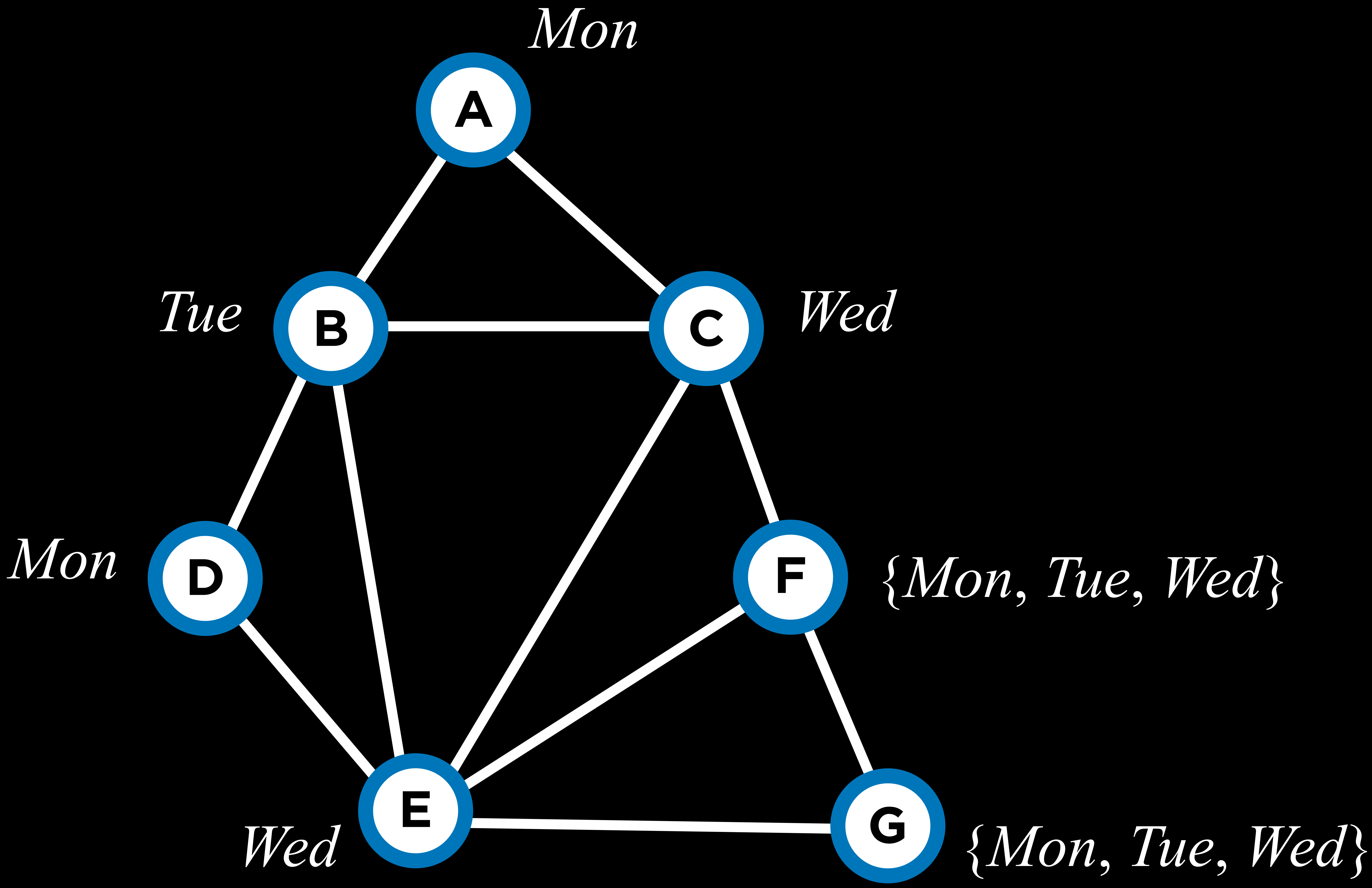
Mon
A
Tue
B
C
Mon
Mon
D
F
{Mon, Tue, Wed}
Wed
E
G
{Mon, Tue, Wed}



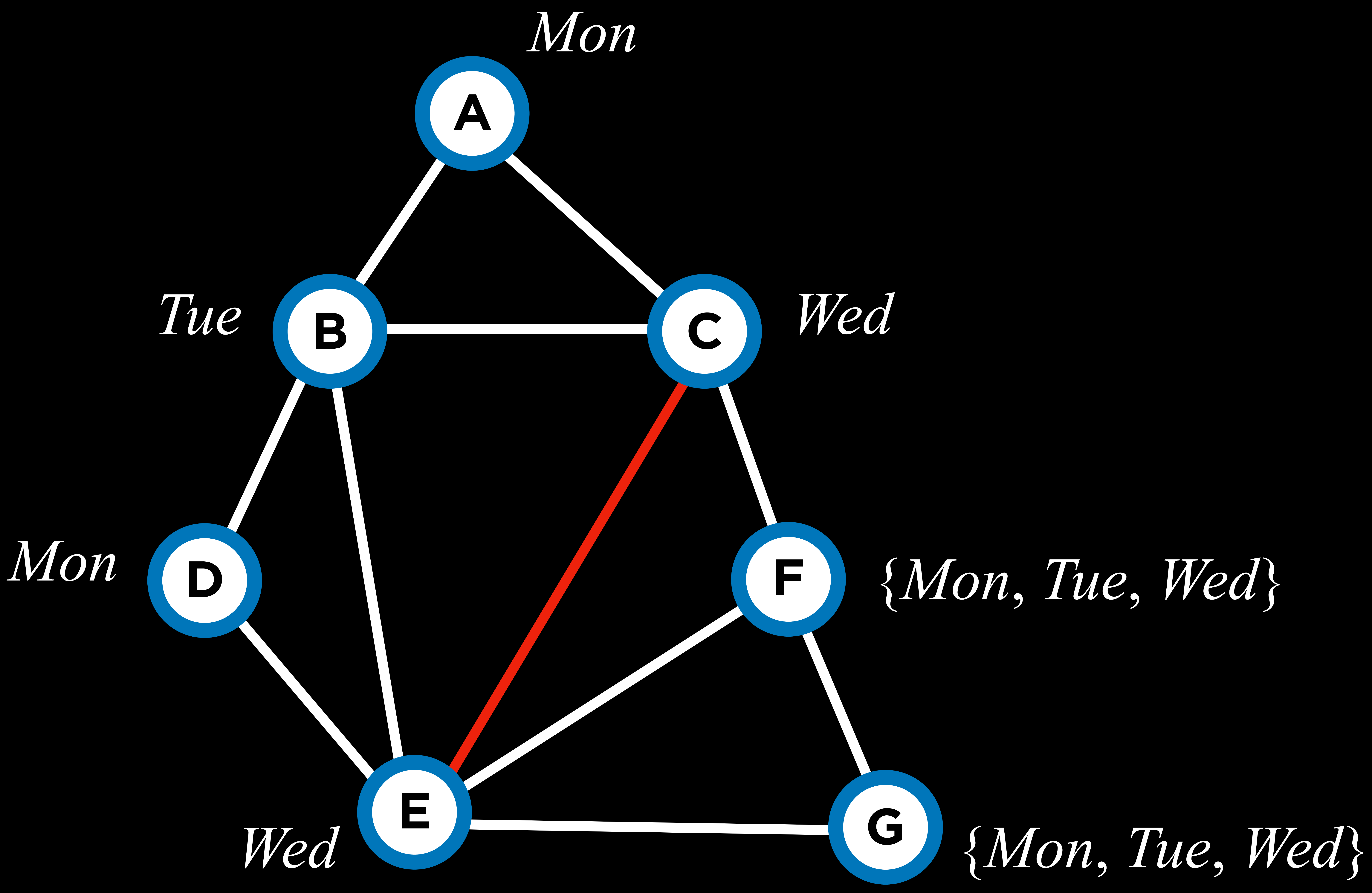
Mon
A
Tue
B
C
Mon
Mon
D
F
{Mon, Tue, Wed}
Wed
E
G
{Mon, Tue, Wed}


Mon
A
Tue
B
C
Tue
Mon
D
F
{Mon, Tue, Wed}
Wed
E
G
{Mon, Tue, Wed}

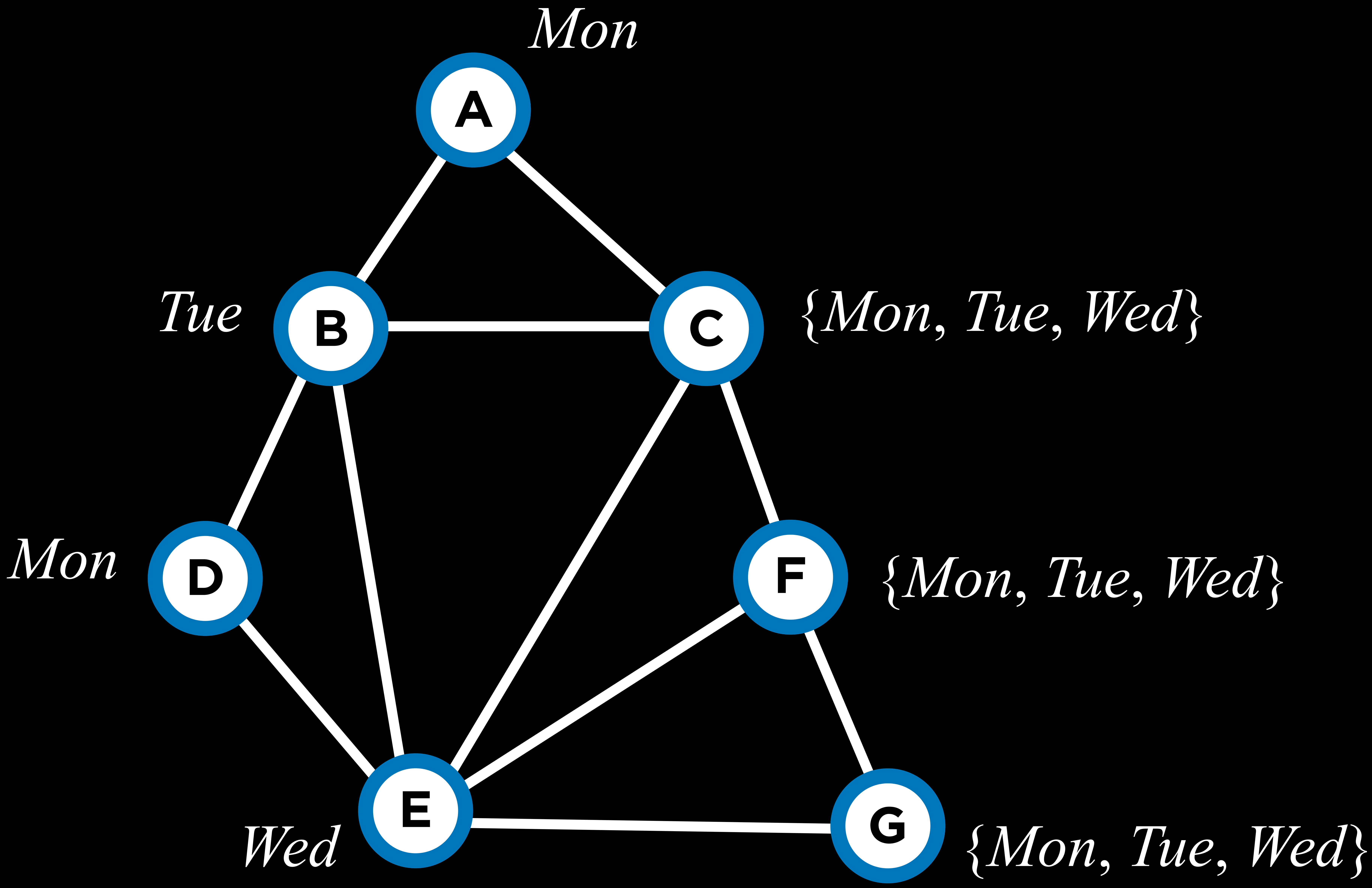

Mon
A
Tue
B
C
Tue
Mon
D
F
{Mon, Tue, Wed}
Wed
E
G
{Mon, Tue, Wed}

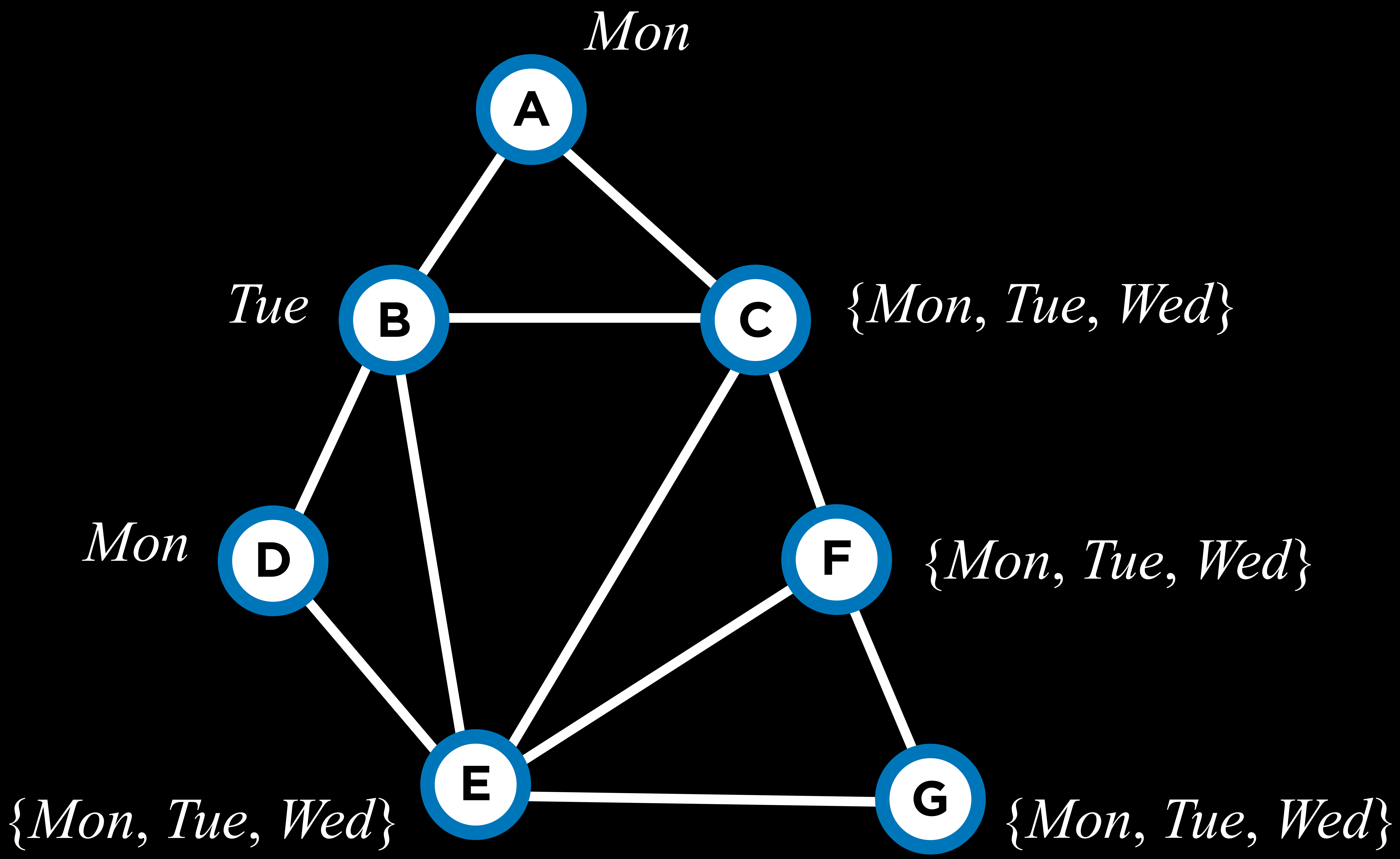


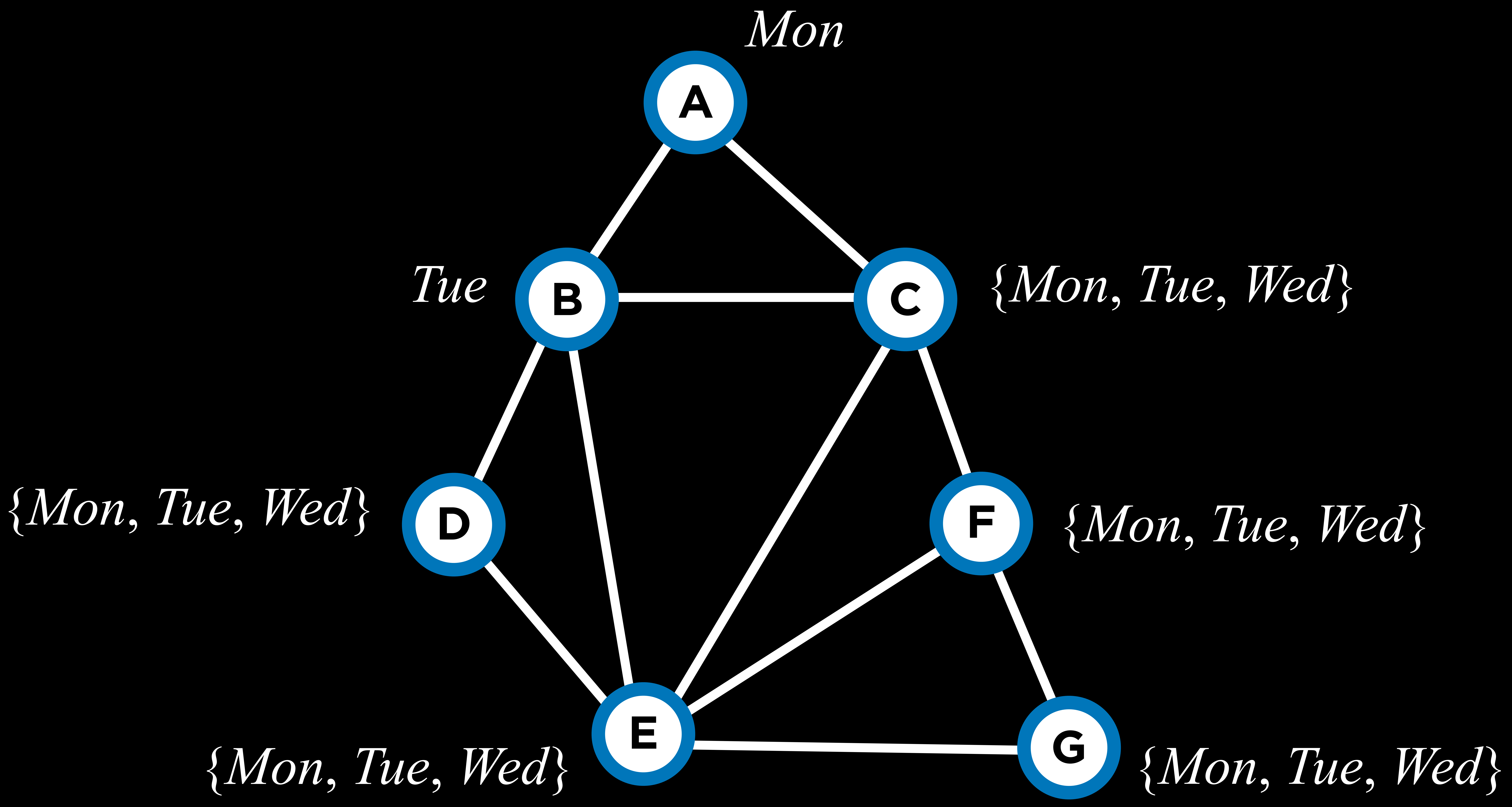
Mon
A
Tue
B
C
Wed
Mon
D
F
{Mon, Tue, Wed}
E
Wed
G
{Mon, Tue, Wed}



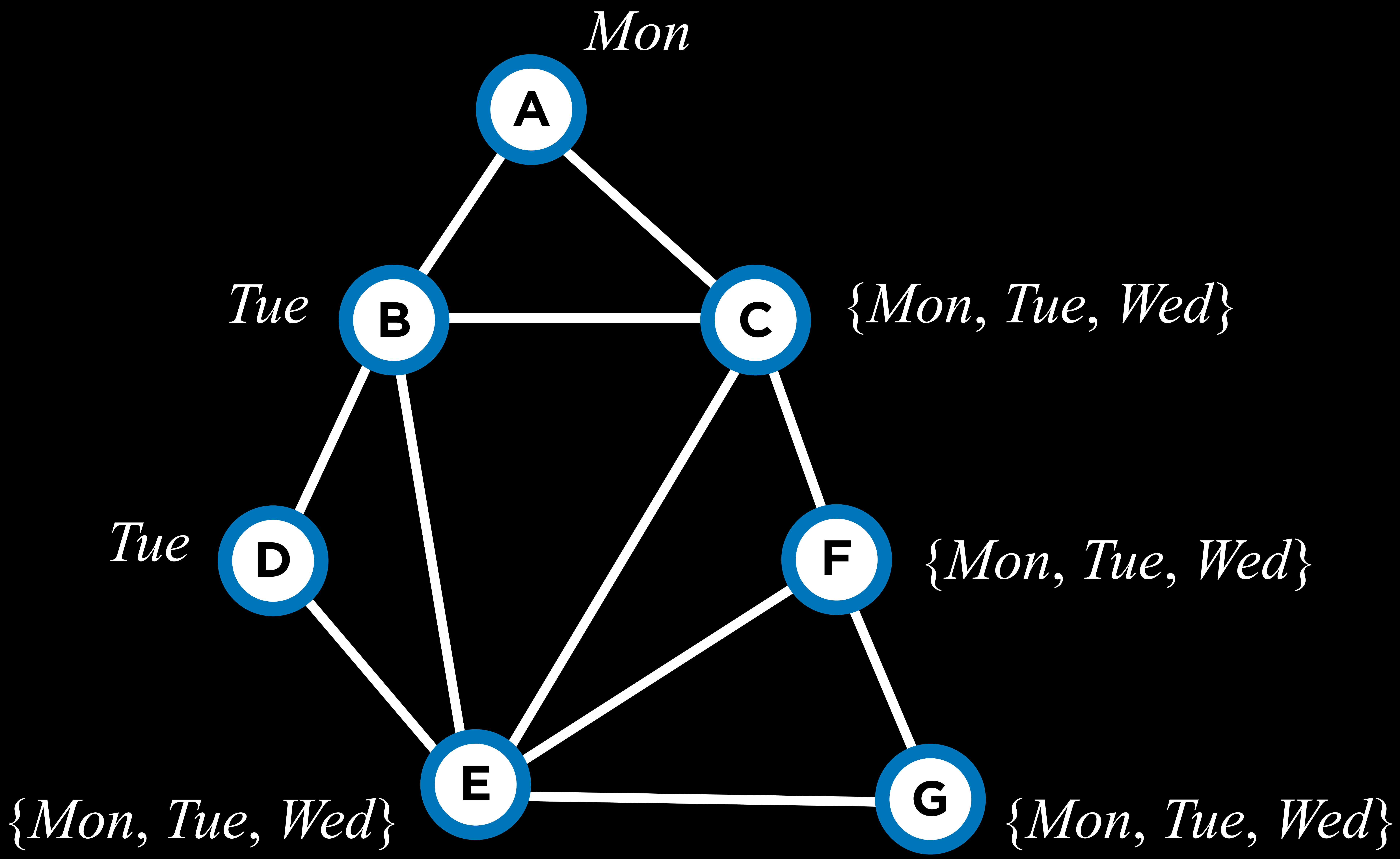
Mon
A
Tue
B
C
Wed
Mon
D
F
{Mon, Tue, Wed}
Wed
E
G
{Mon, Tue, Wed}

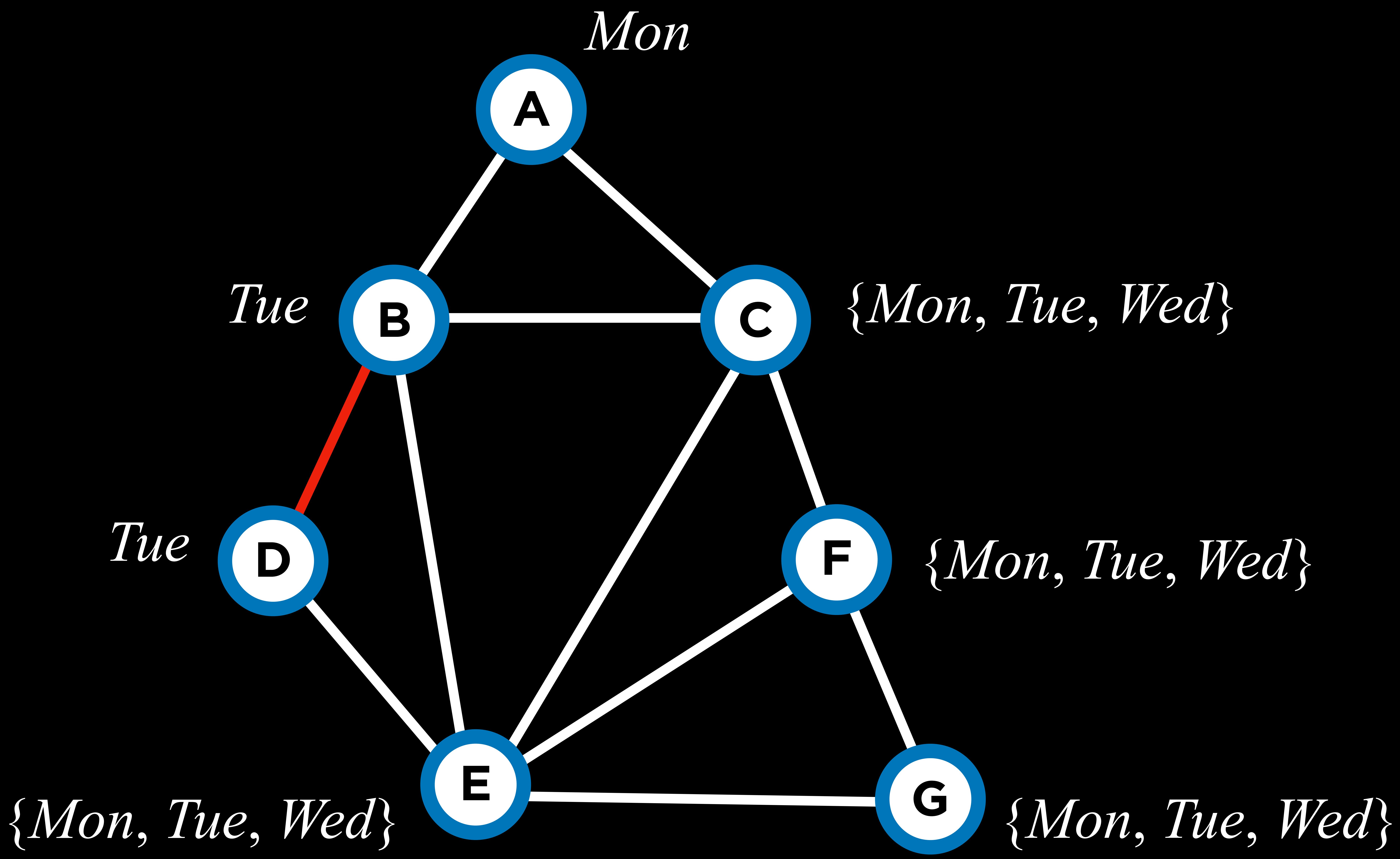

Mon
A
Tue
B
C
{Mon, Tue, Wed}
Mon
D
F
{Mon, Tue, Wed}
Wed
E
G
{Mon, Tue, Wed}

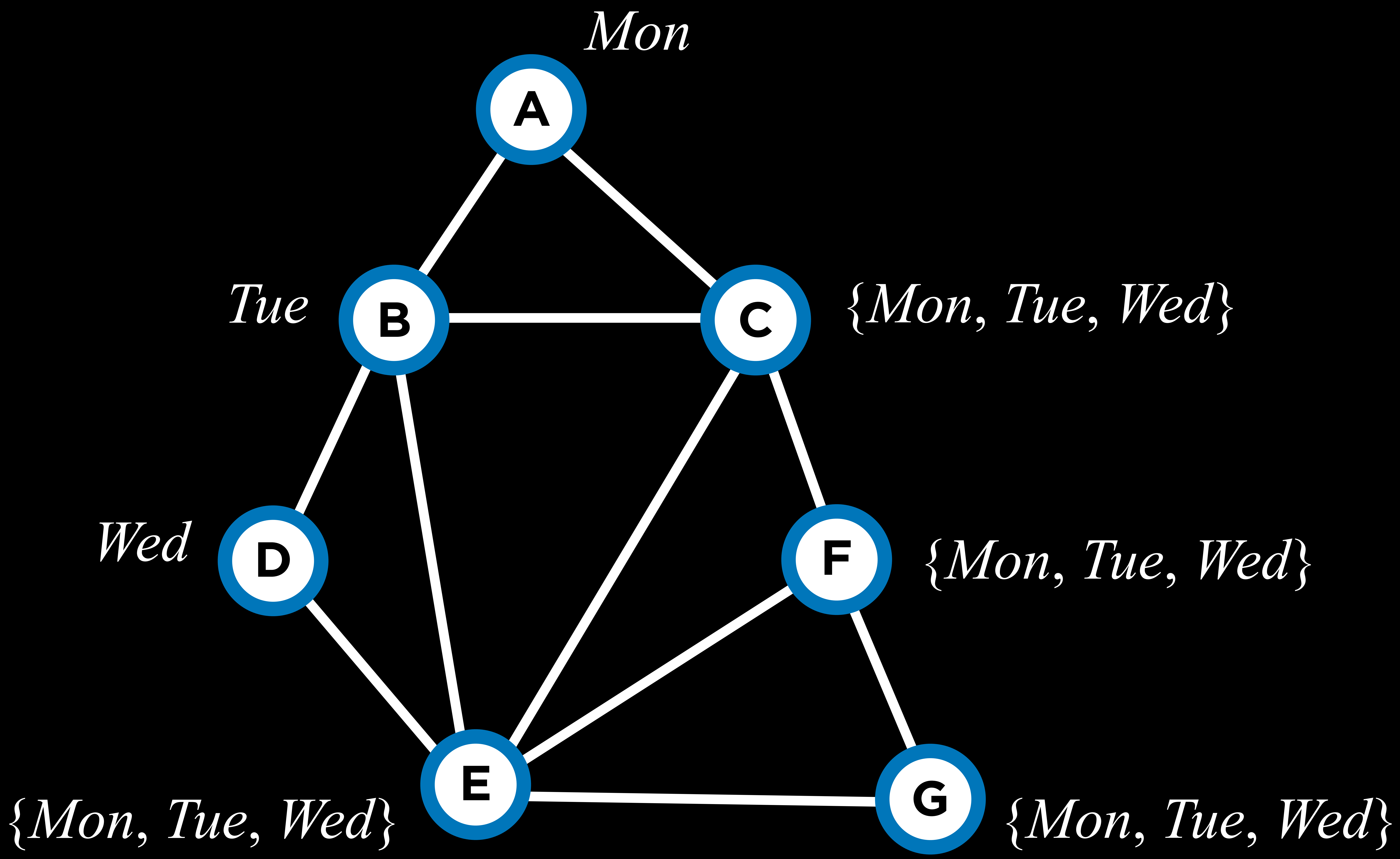

Mon
A
Tue
B
C
{Mon, Tue, Wed}
Mon
D
F
{Mon, Tue, Wed}
E
{Mon, Tue, Wed}
G
{Mon, Tue, Wed}

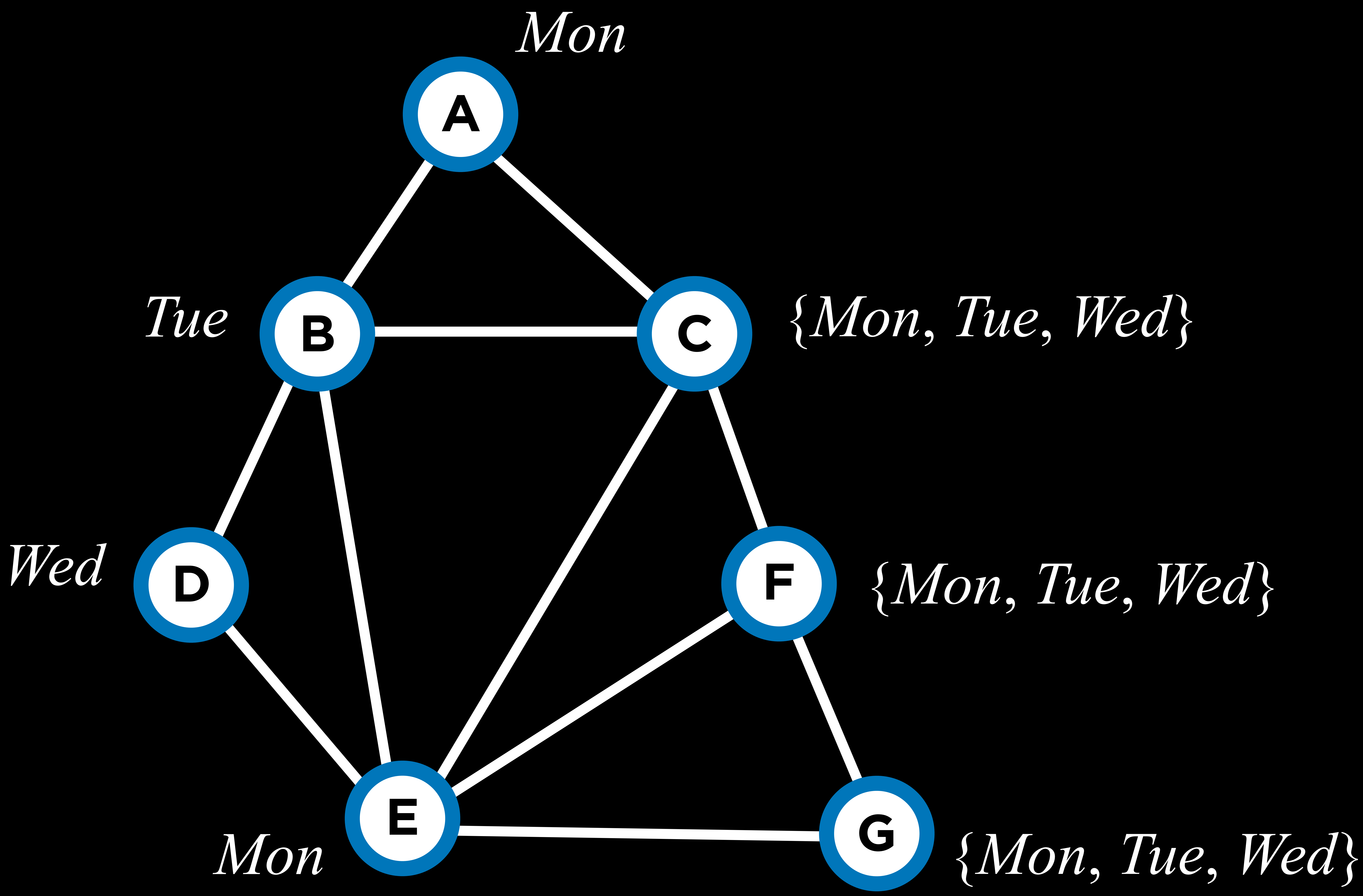


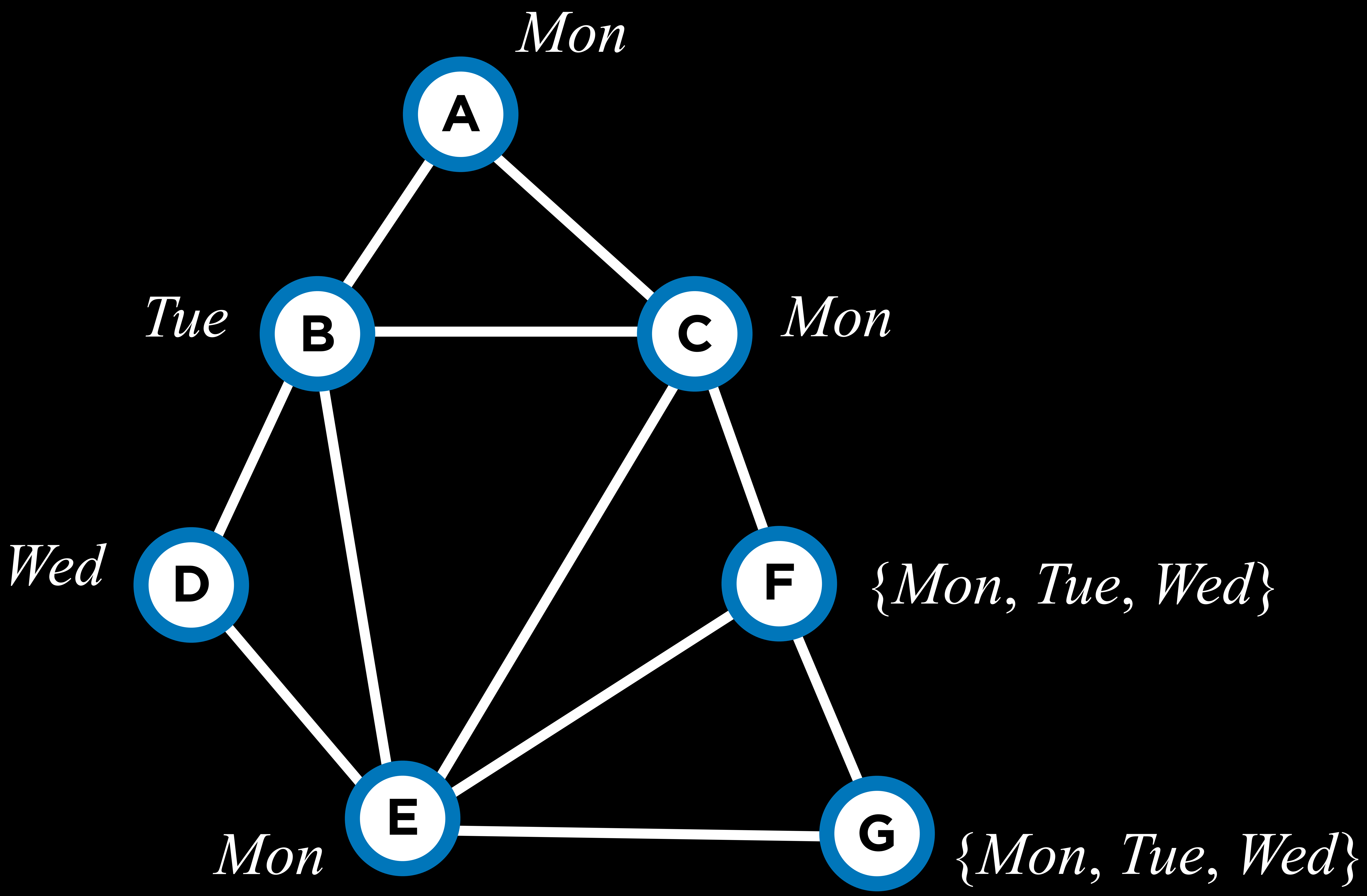
Mon
A
Tue
B
C
{Mon, Tue, Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}

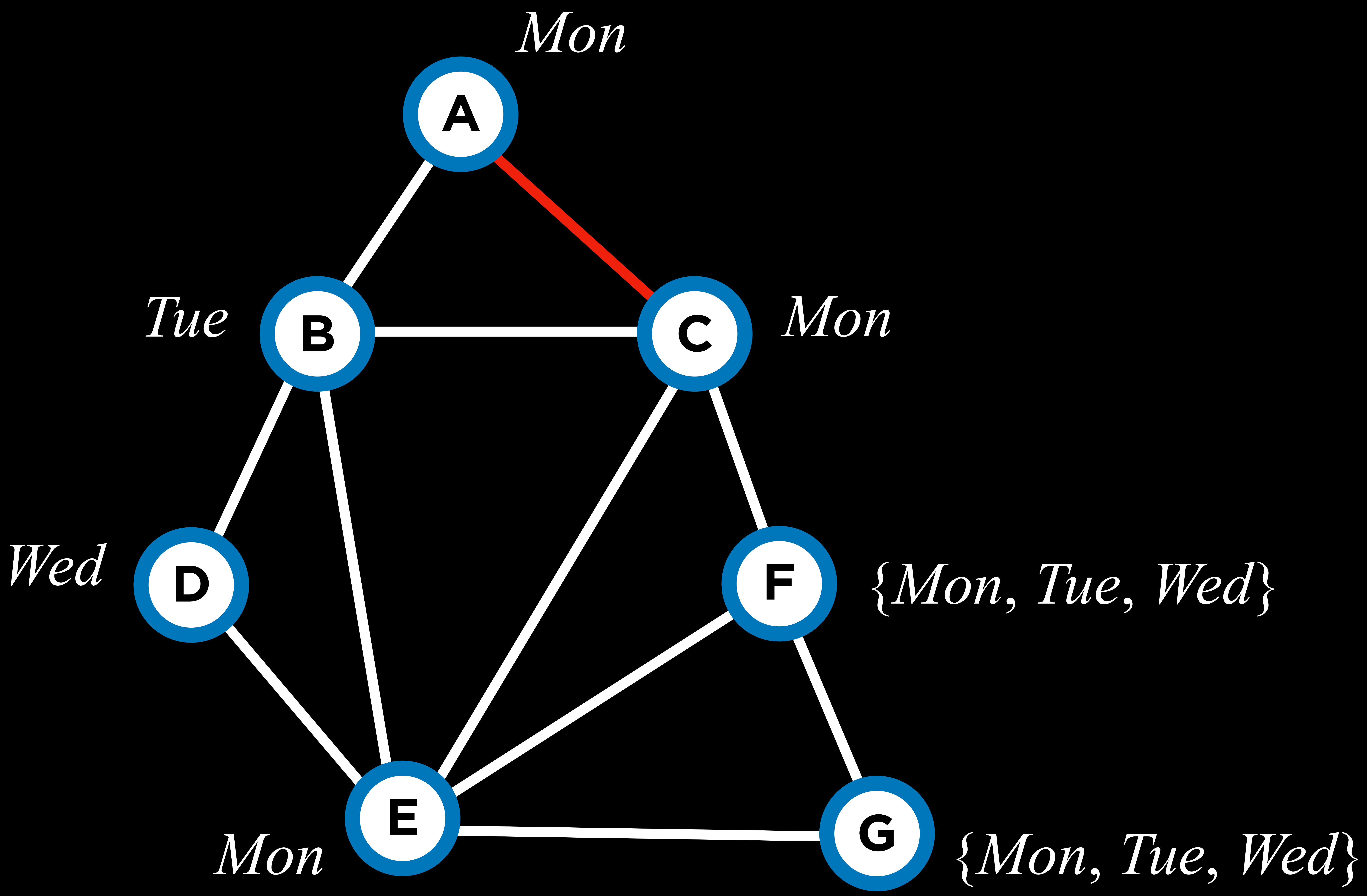

Mon
A
Tue
B
C
{Mon, Tue, Wed}
Tue
D
F
{Mon, Tue, Wed}
E
{Mon, Tue, Wed}
G
{Mon, Tue, Wed}

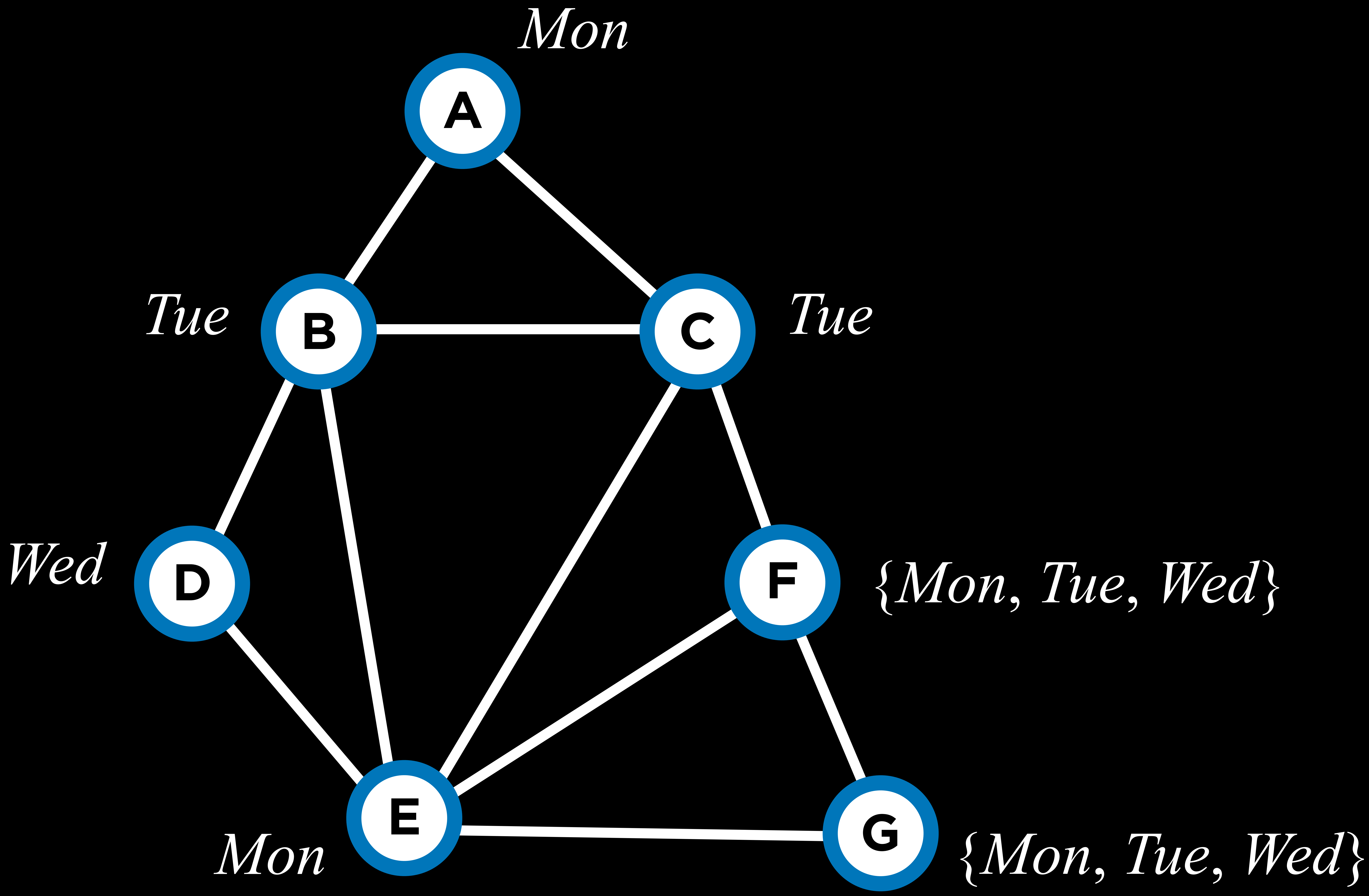

Mon
A
Tue
B
C
{Mon, Tue, Wed}
Tue
D
F
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}
{Mon, Tue, Wed}

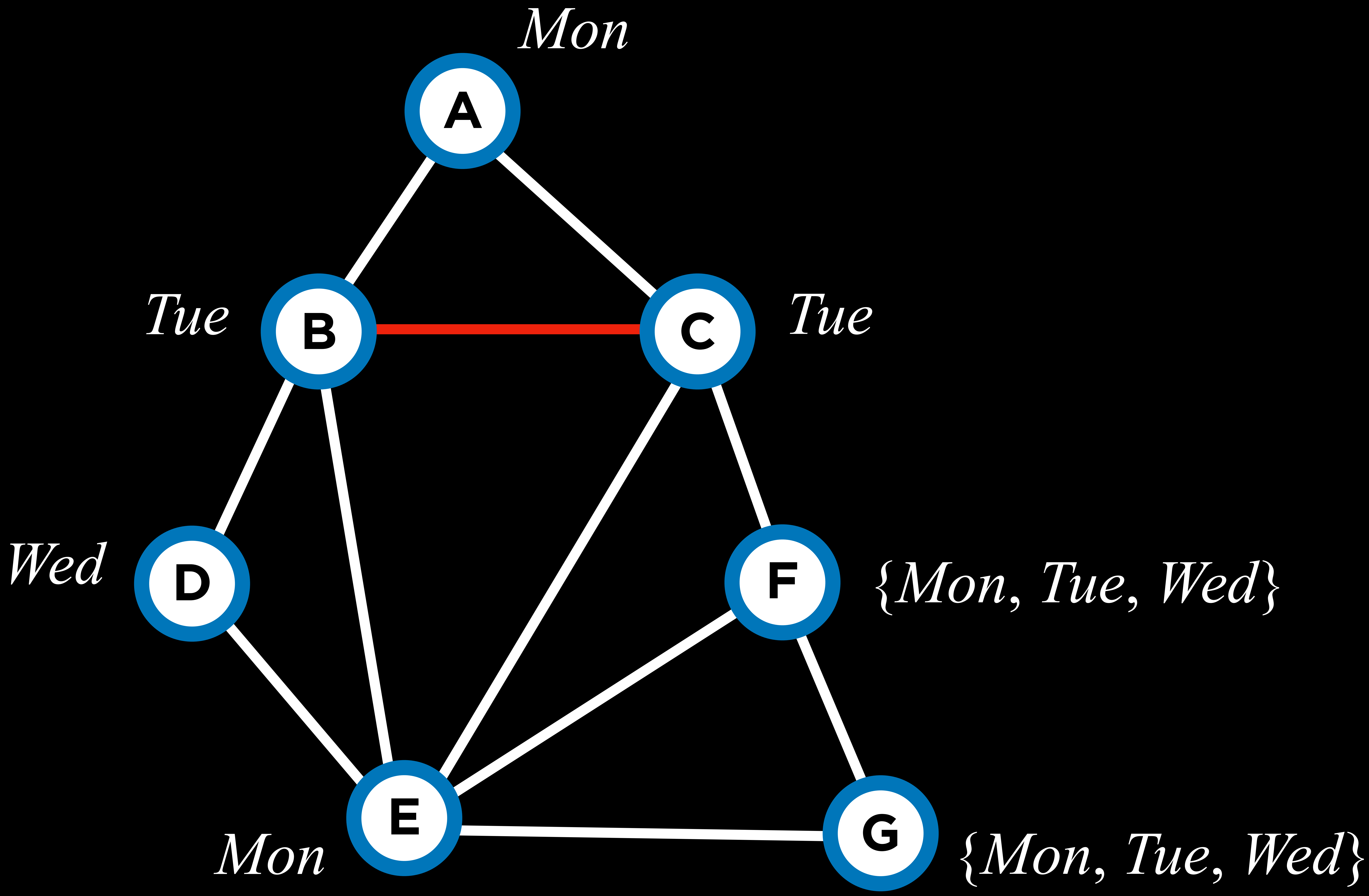

Mon
A
Tue
B
C
{Mon, Tue, Wed}
Wed
D
F
{Mon, Tue, Wed}
E
{Mon, Tue, Wed}
G
{Mon, Tue, Wed}

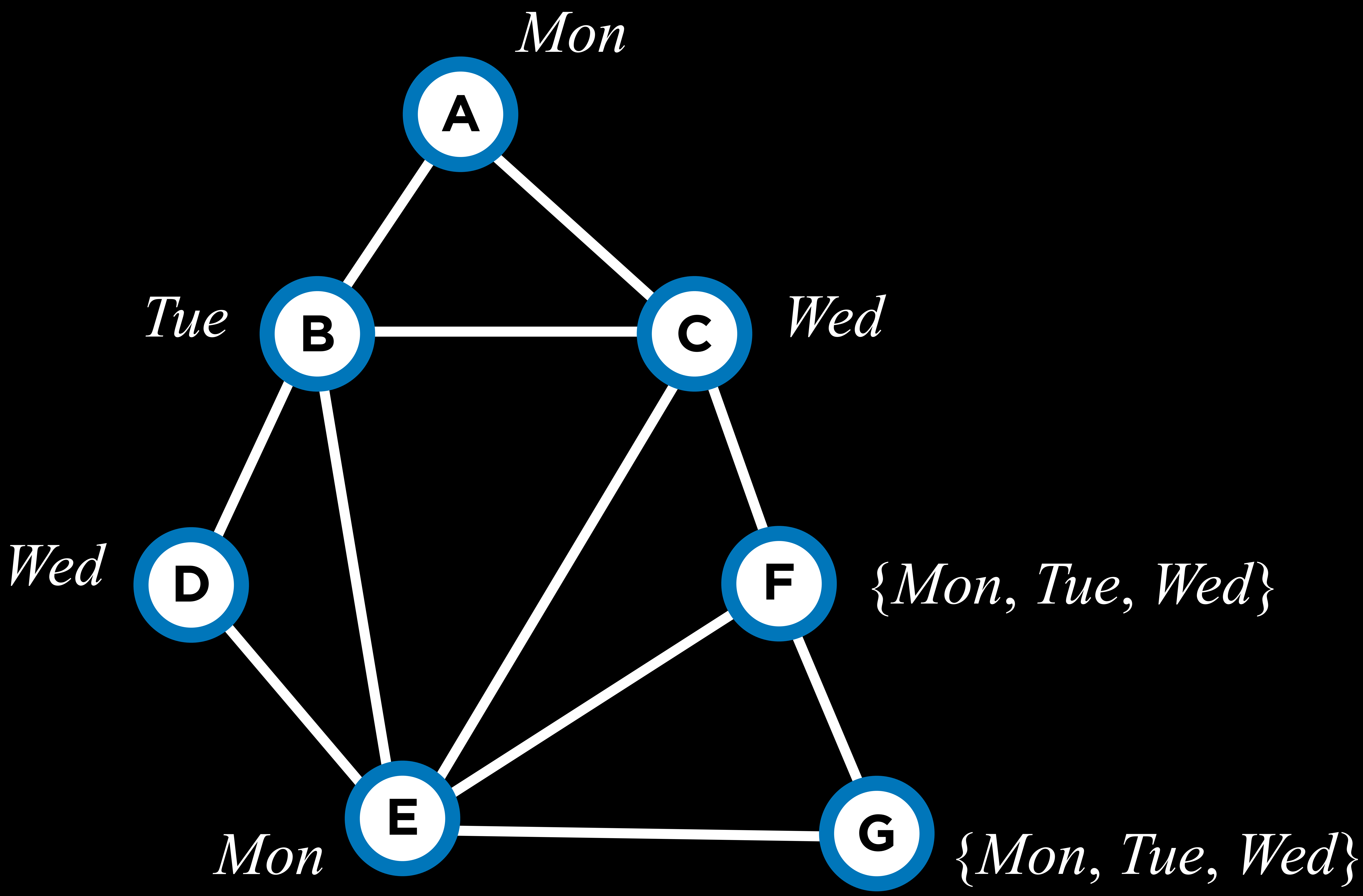

Mon
A
Tue
B
C
{Mon, Tue, Wed}
Wed
D
F
{Mon, Tue, Wed}
E
Mon
G
{Mon, Tue, Wed}

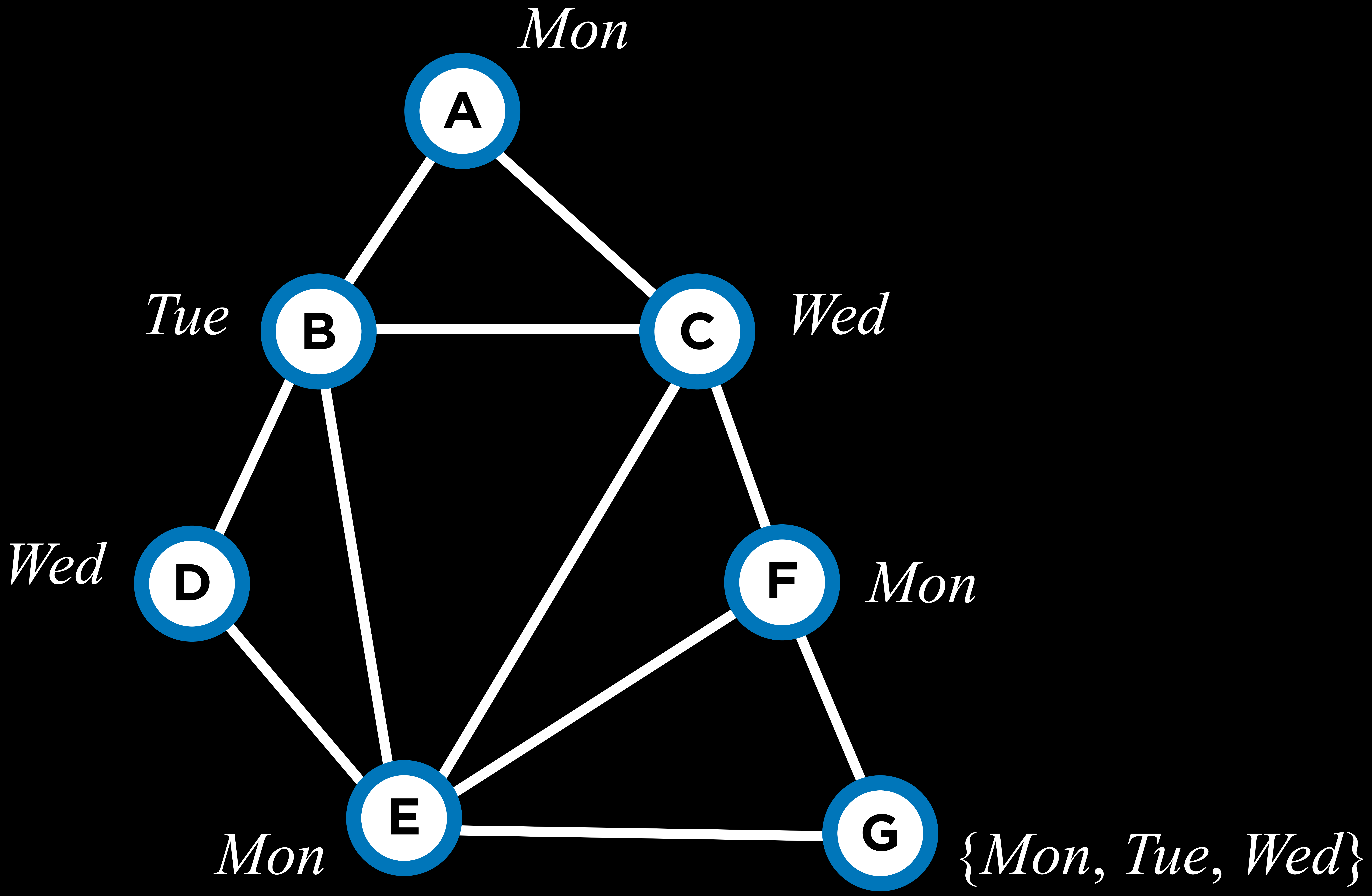

Mon
A
Tue
B
C
Mon
Wed
D
F
{Mon, Tue, Wed}
E
Mon
G
{Mon, Tue, Wed}

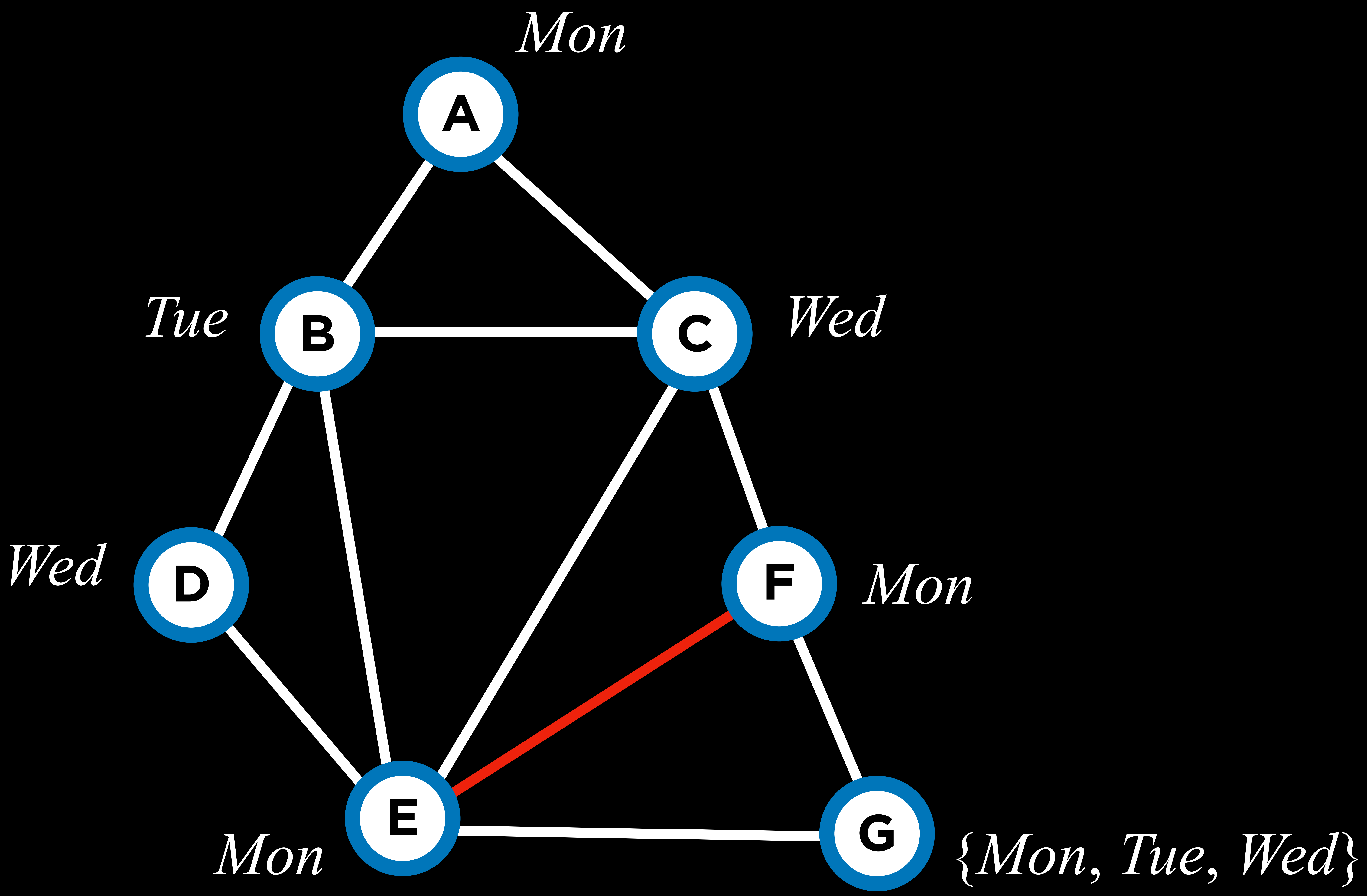

Mon
A
Tue
B
C
Mon
Wed
D
F
{Mon, Tue, Wed}
Mon
E
G
{Mon, Tue, Wed}

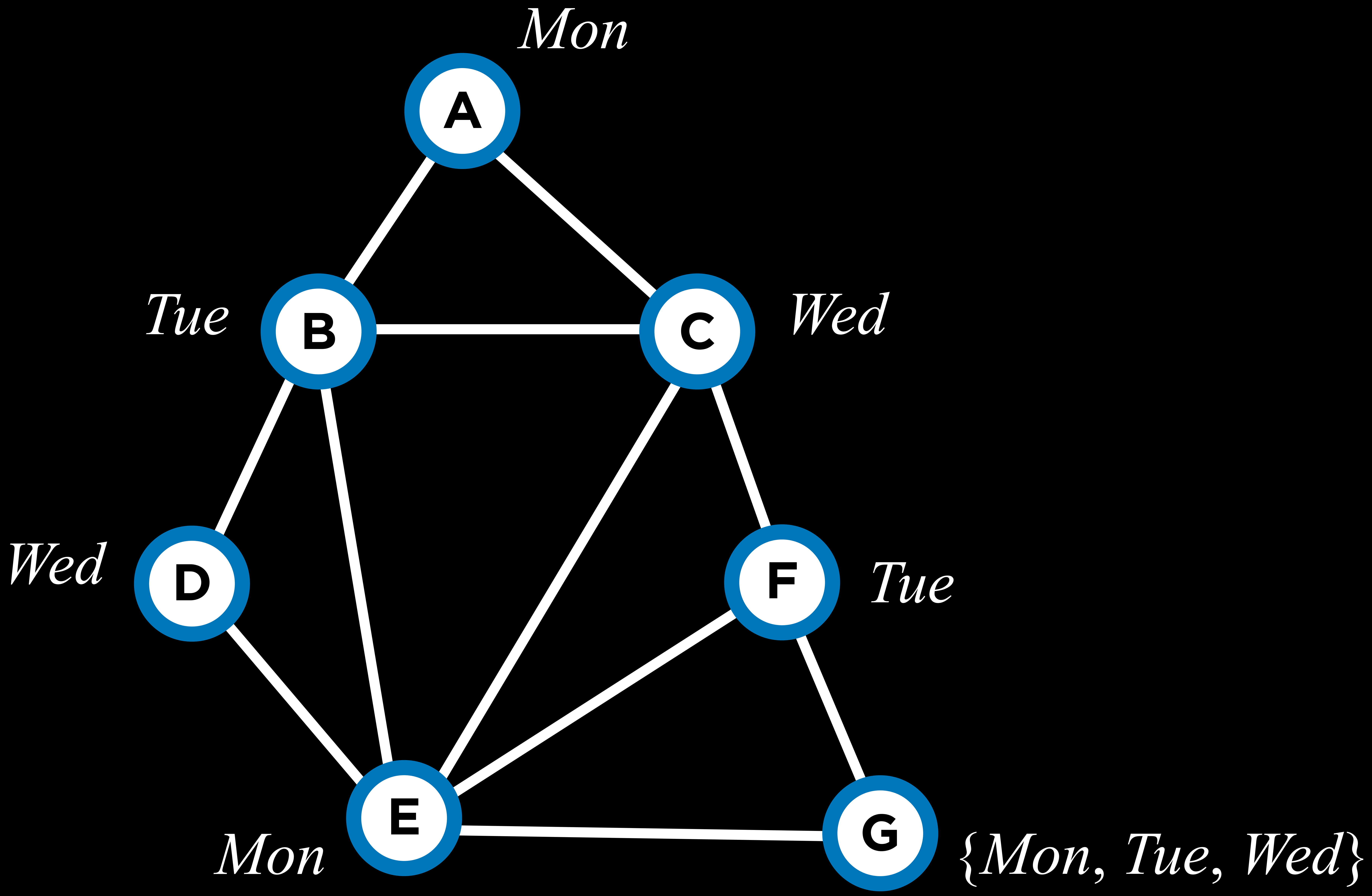

Mon
A
Tue
B
C
Tue
Wed
D
F
{Mon, Tue, Wed}
Mon
E
G
{Mon, Tue, Wed}

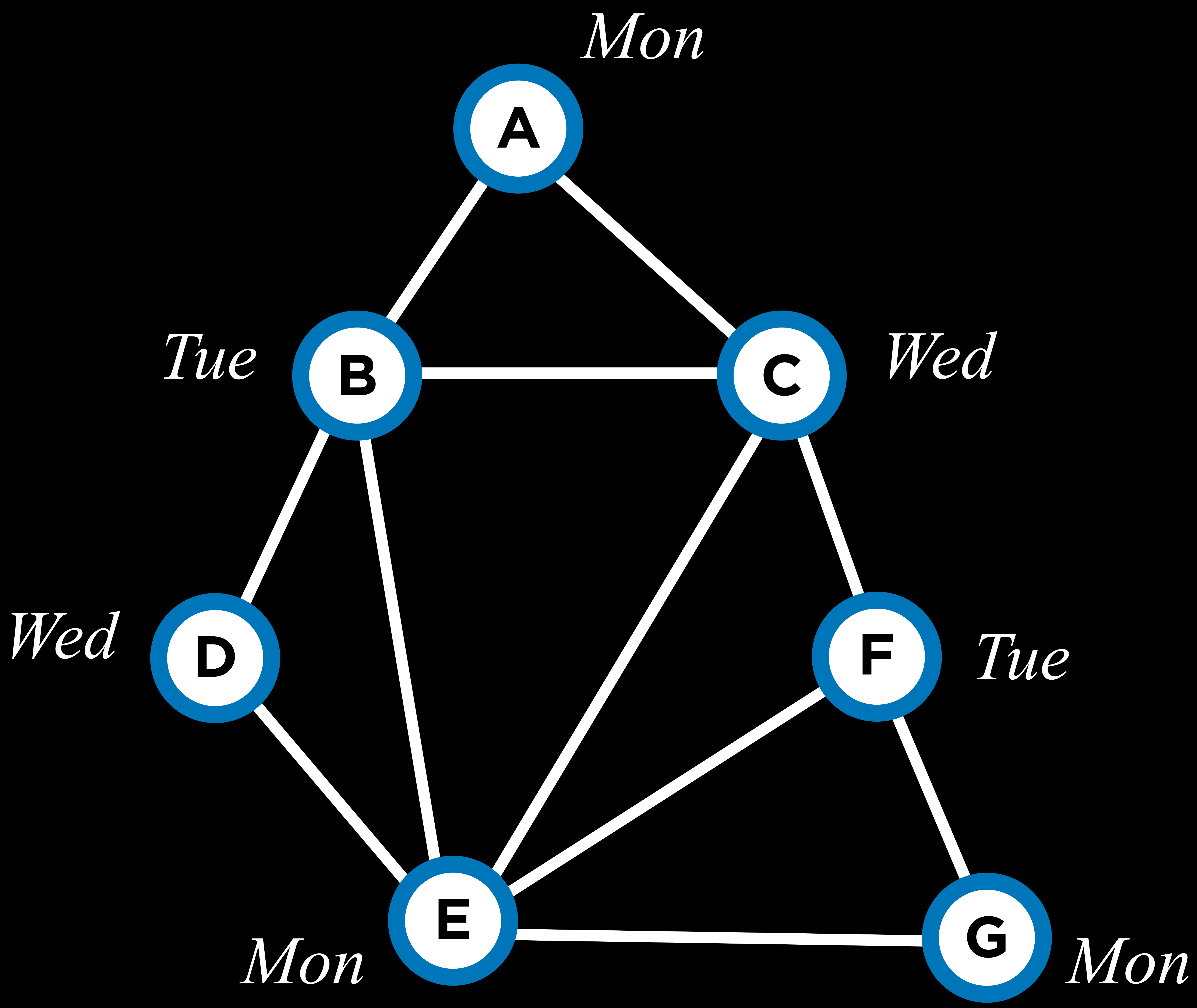

Mon
A
Tue
B
C
Tue
Wed
D
F
{Mon, Tue, Wed}
E
Mon
G
{Mon, Tue, Wed}

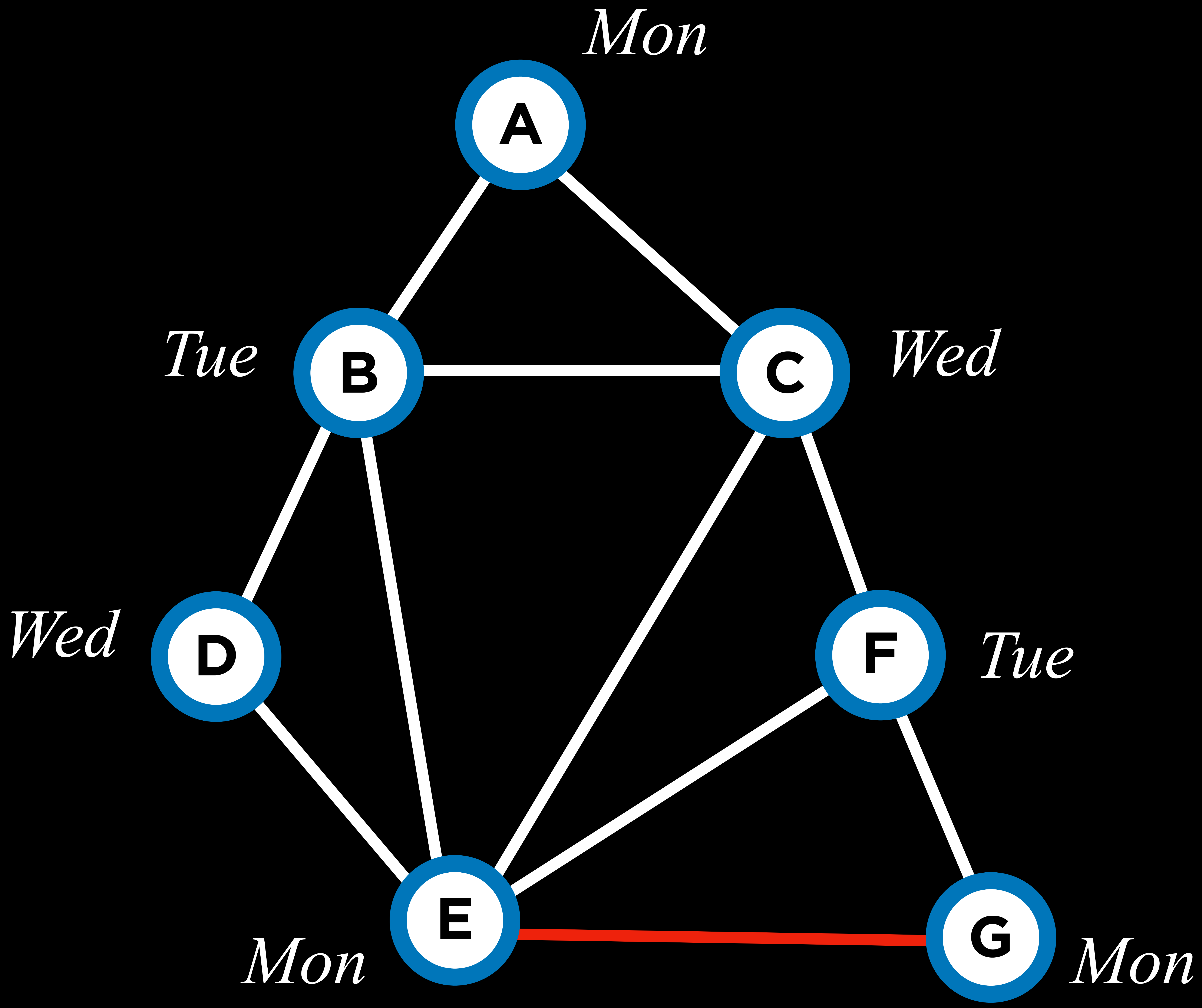

Mon
A
Tue
B
C
Wed
Wed
D
F
{Mon, Tue, Wed}
E
Mon
G
{Mon, Tue, Wed}

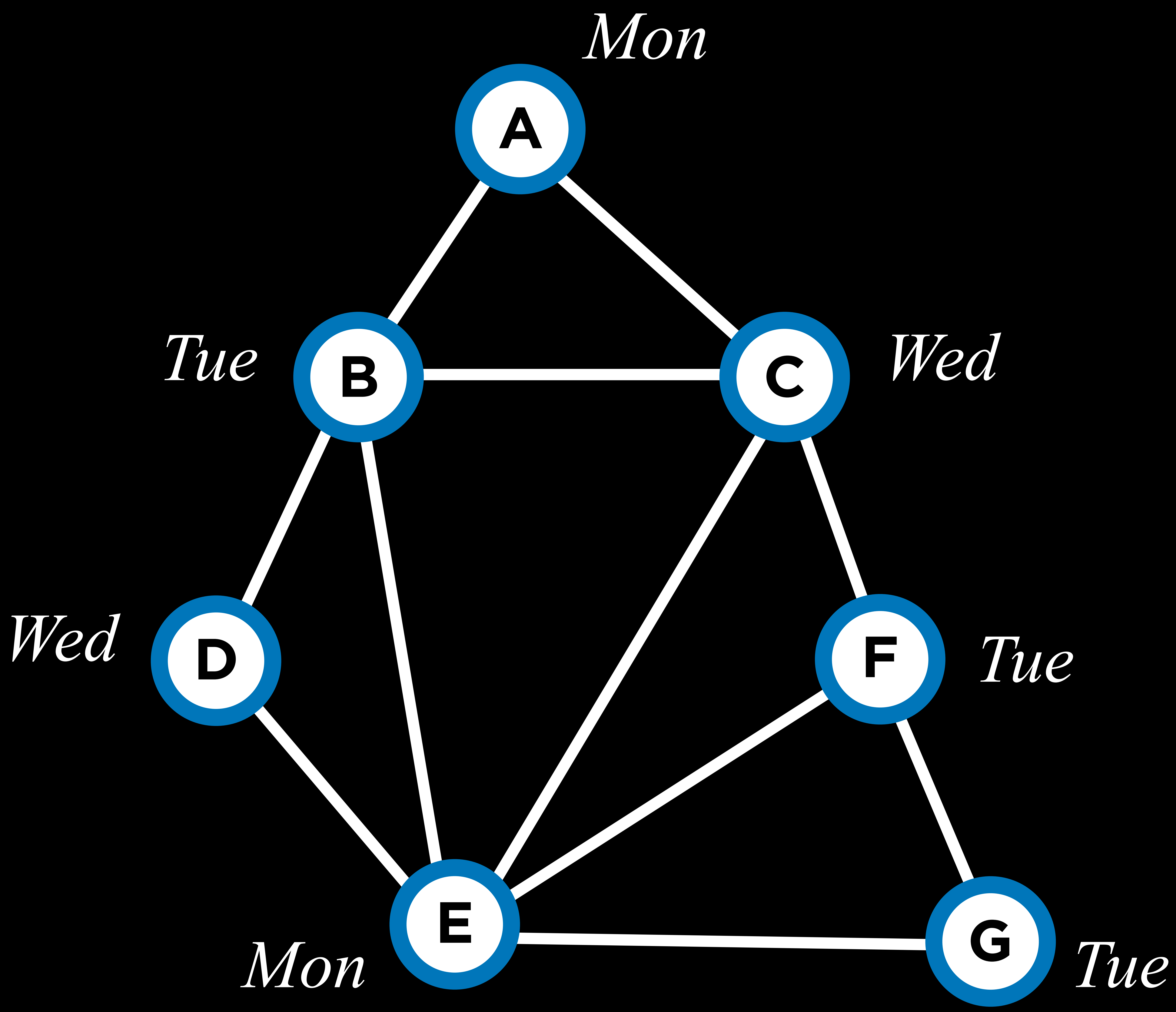

Mon
A
Tue
B
C
Wed
Wed
D
F
Mon
E
Mon
G
{Mon, Tue, Wed}

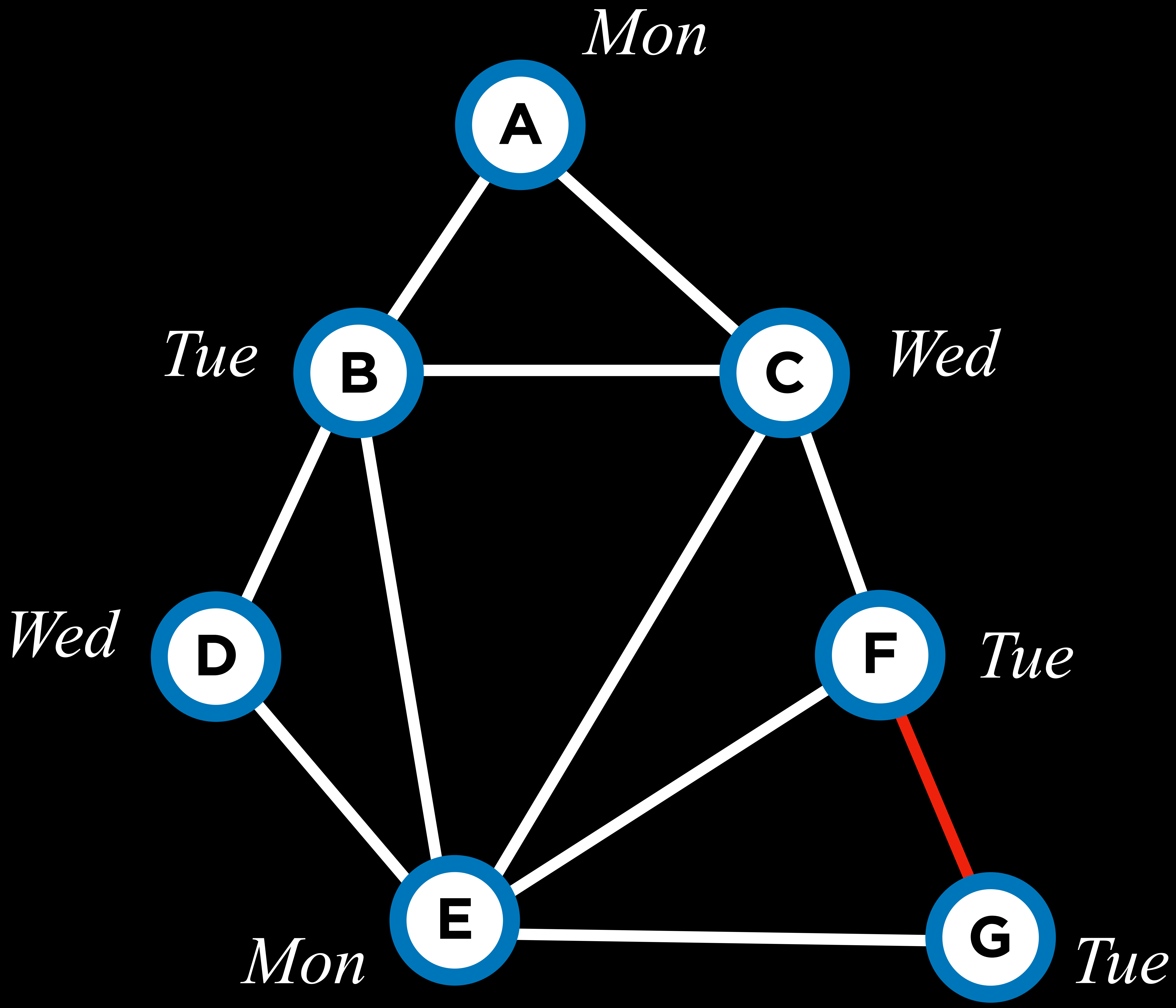

Mon
A
Tue
B
C
Wed
Wed
D
F
Mon
E
Mon
G
{Mon, Tue, Wed}

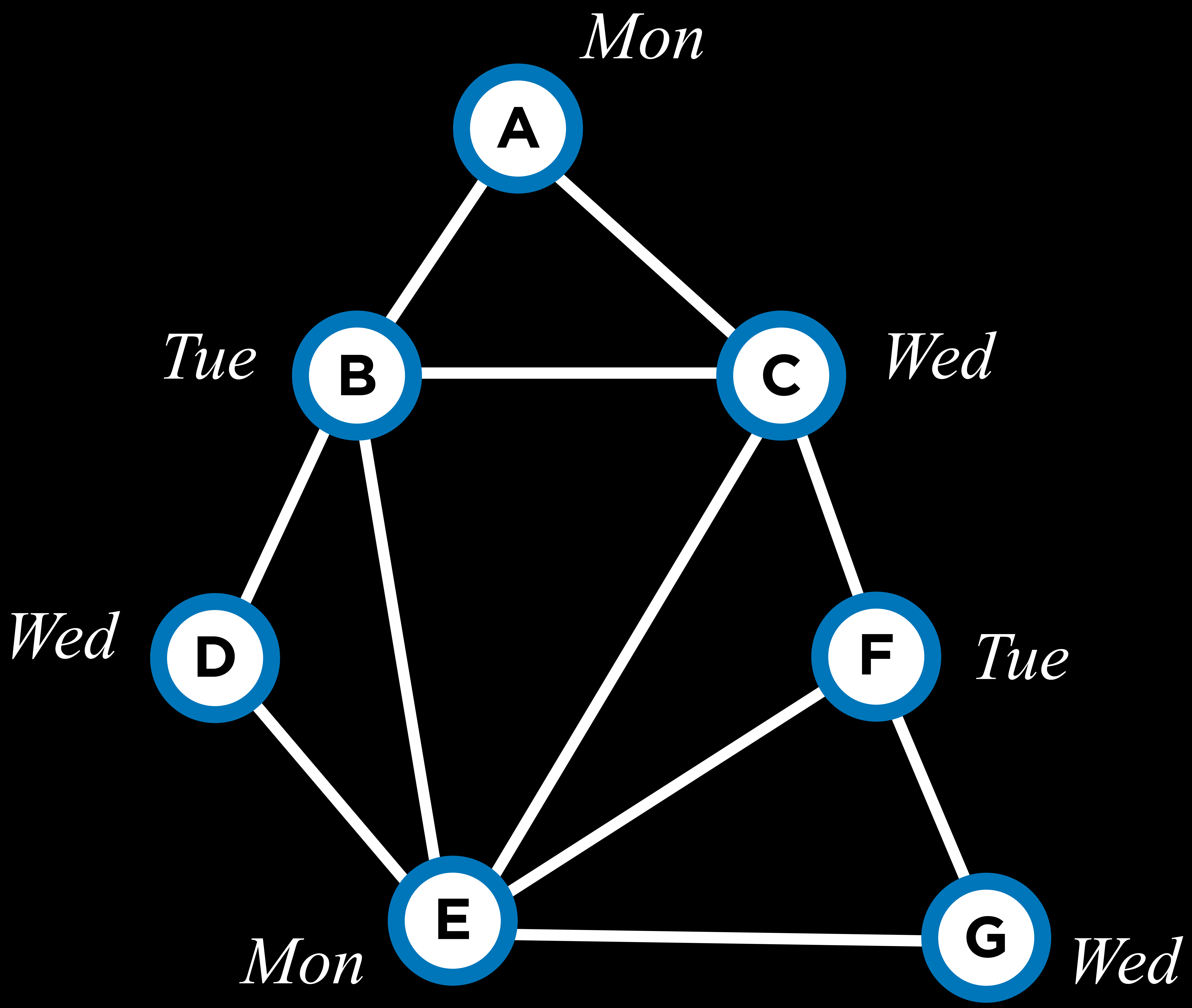

Mon
A
Tue
B
C
Wed
Wed
D
F
Tue
E
Mon
G
{Mon, Tue, Wed}


Mon
A
Tue
B
C
Wed
Wed
D
F
Tue
E
Mon
G
Mon

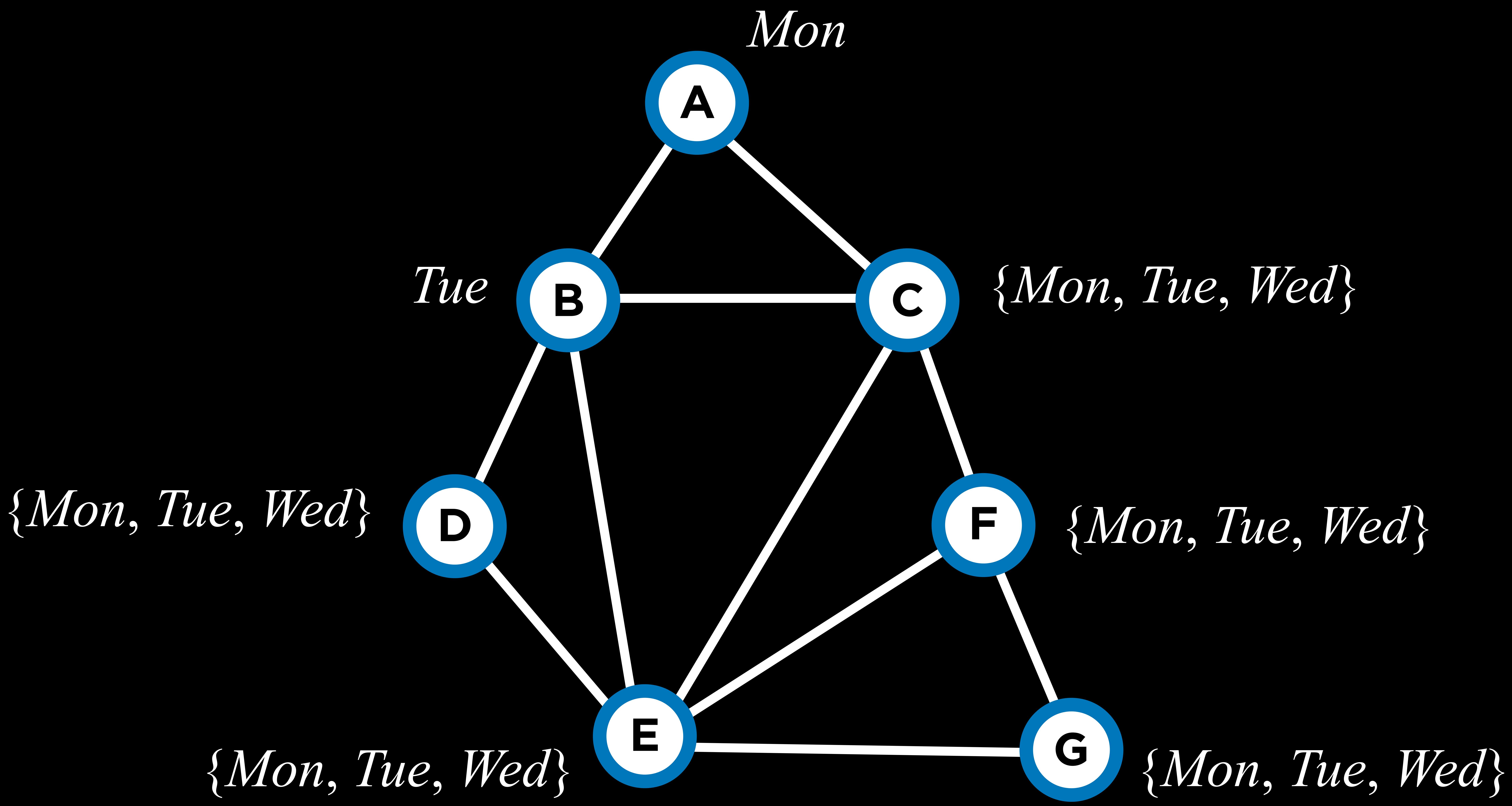

Mon
A
Tue
B
C
Wed
Wed
D
F
Tue
E
Mon
G
Mon

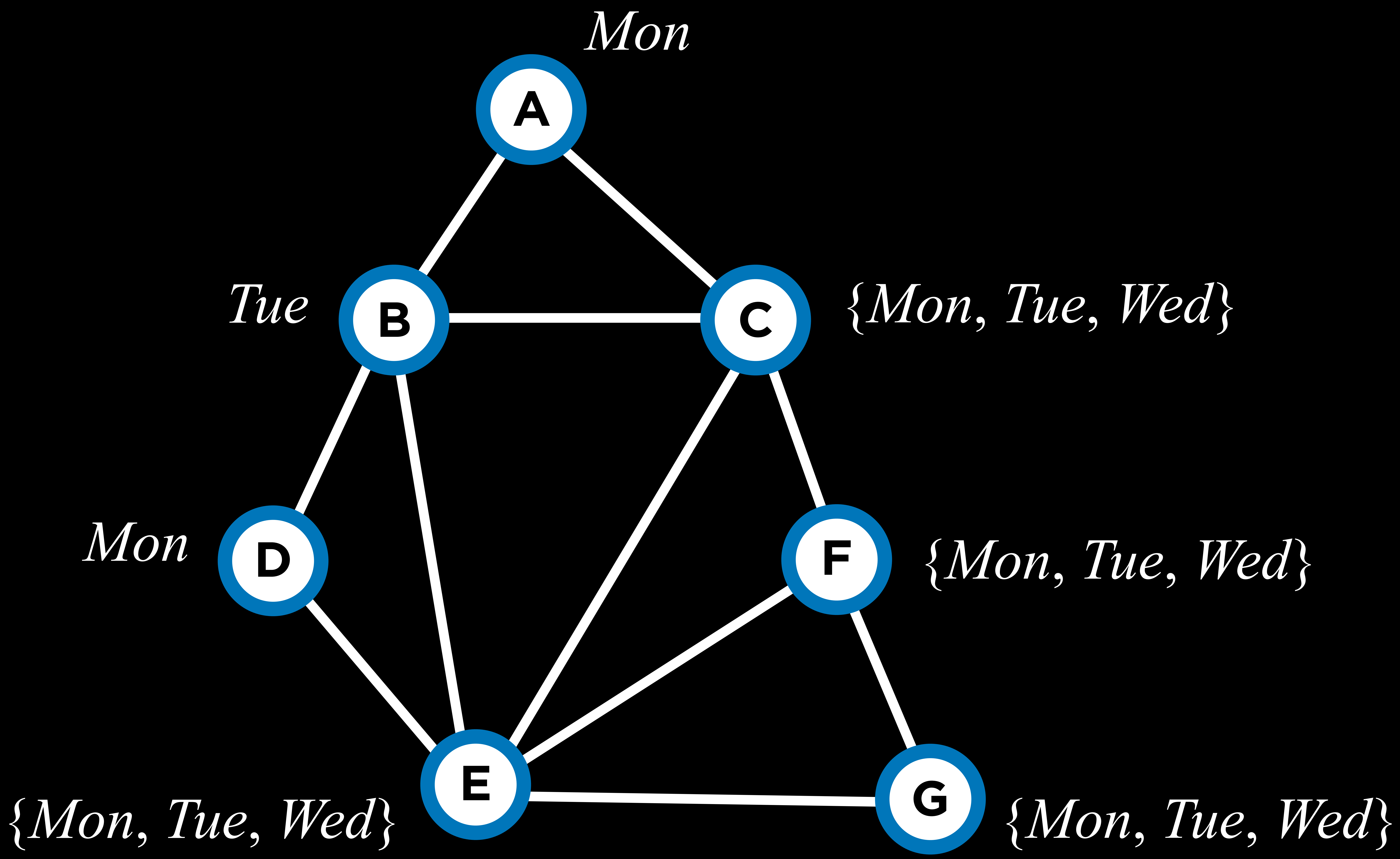

Mon
A
Tue
B
C
Wed
Wed
D
F
Tue
E
Mon
G
Tue

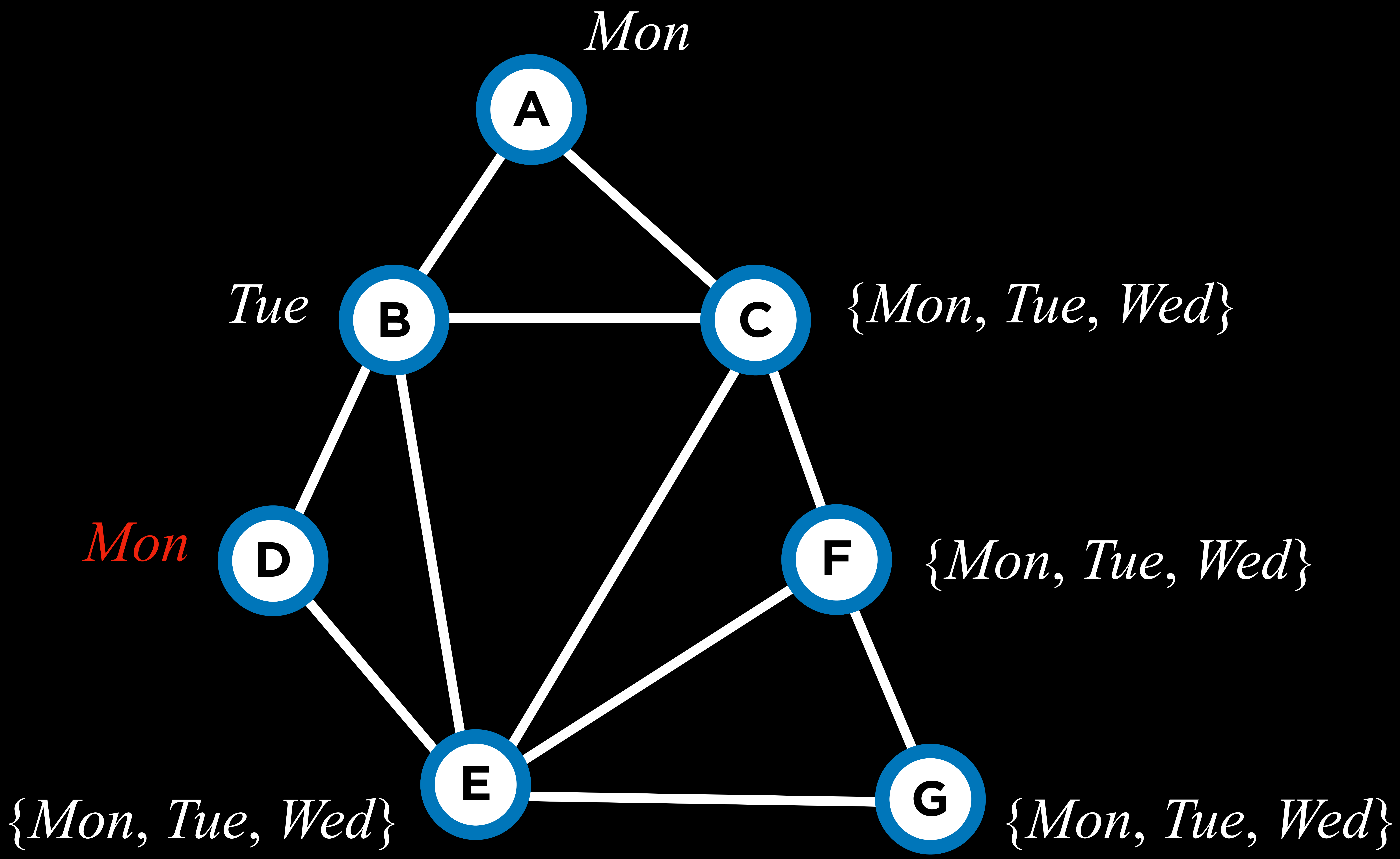

Mon
A
Tue
B
C
Wed
Wed
D
F
Tue
E
Mon
G
Tue


Mon
A
Tue
B
C
Wed
Wed
D
F
Tue
E
Mon
G
Wed

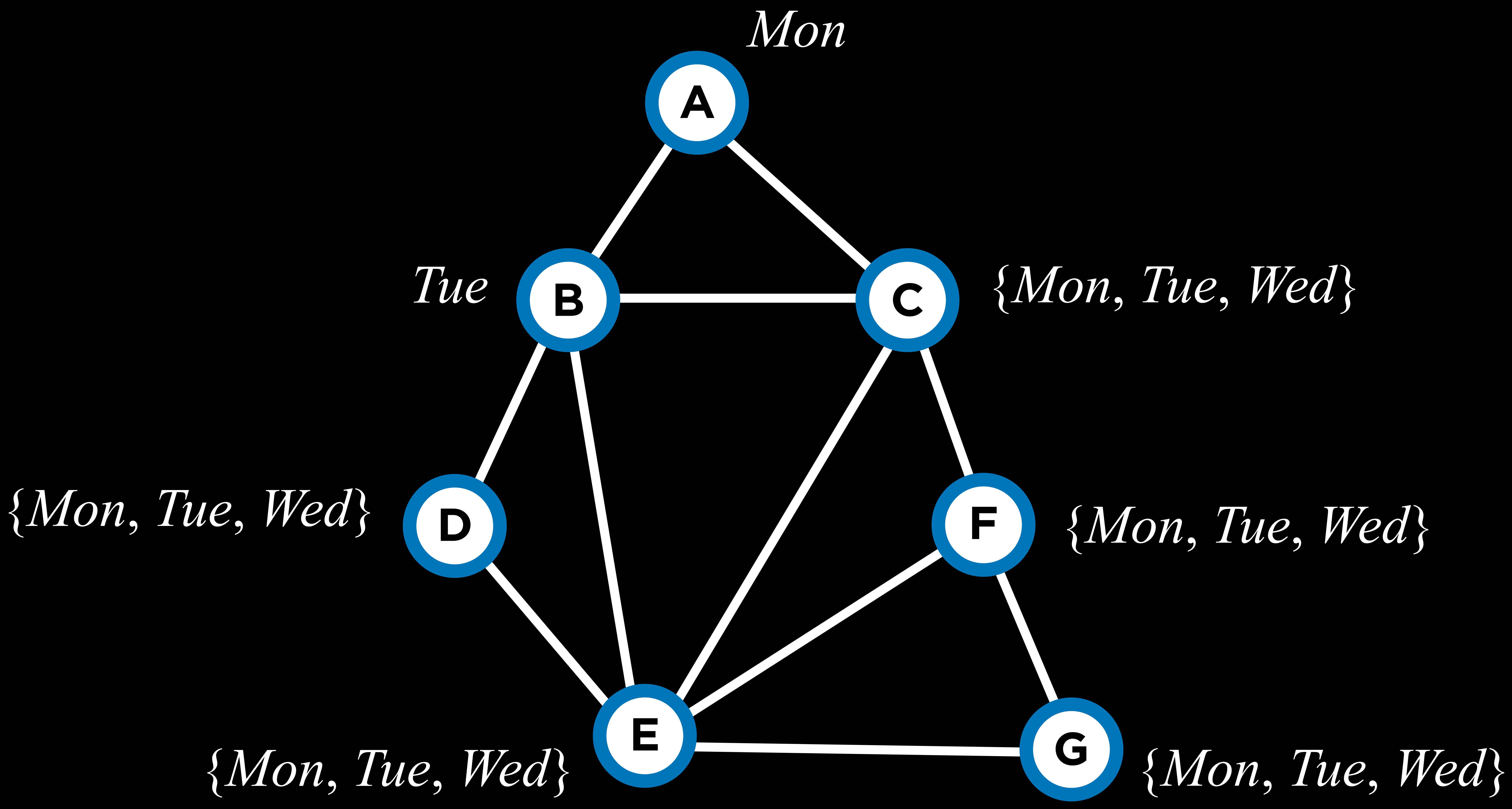
Inference

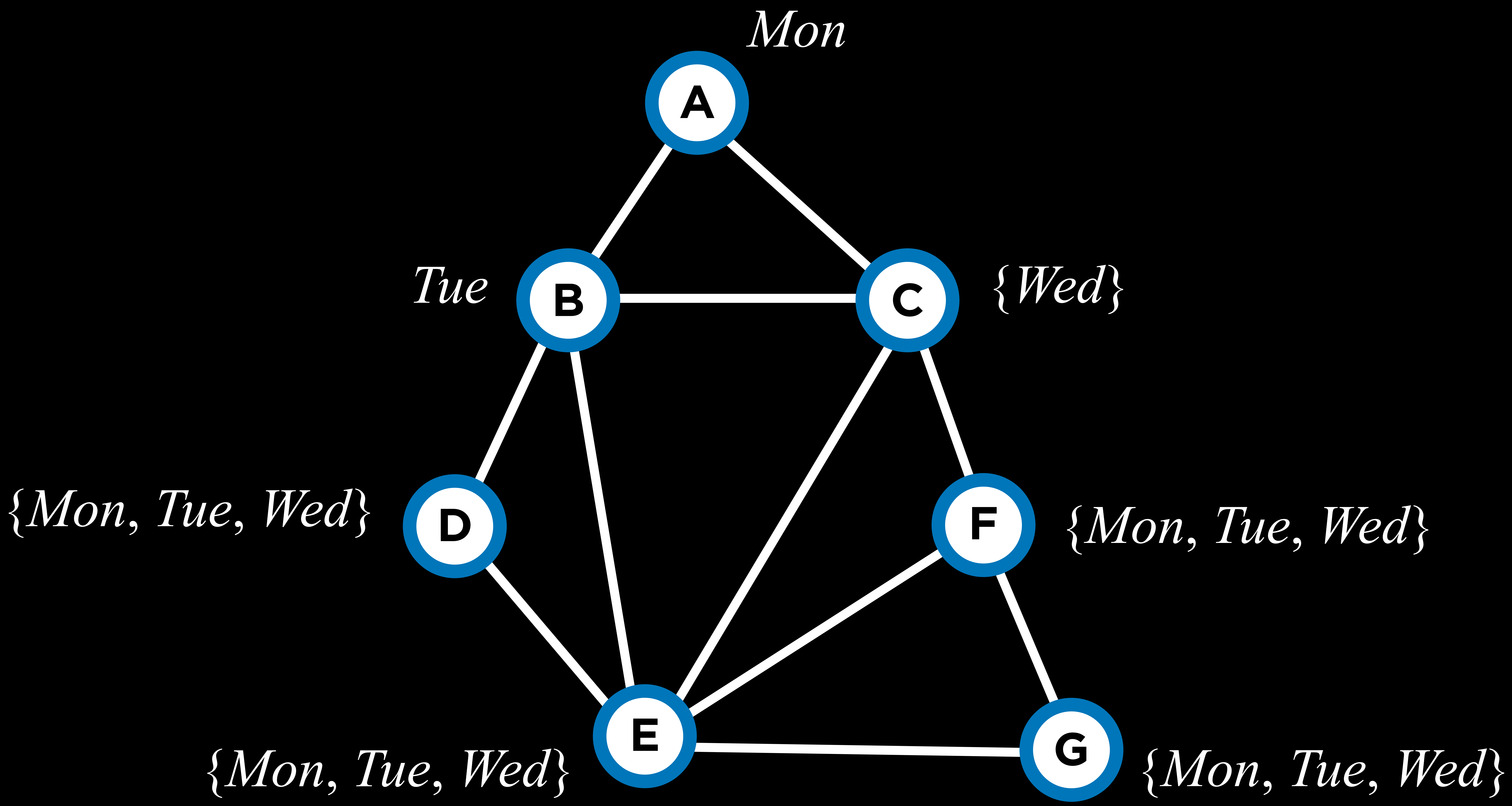

Mon
A
Tue
B
C
{Mon, Tue, Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}


Mon
A
Tue
B
C
{Mon, Tue, Wed}
Mon
D
F
{Mon, Tue, Wed}
E
{Mon, Tue, Wed}
G
{Mon, Tue, Wed}

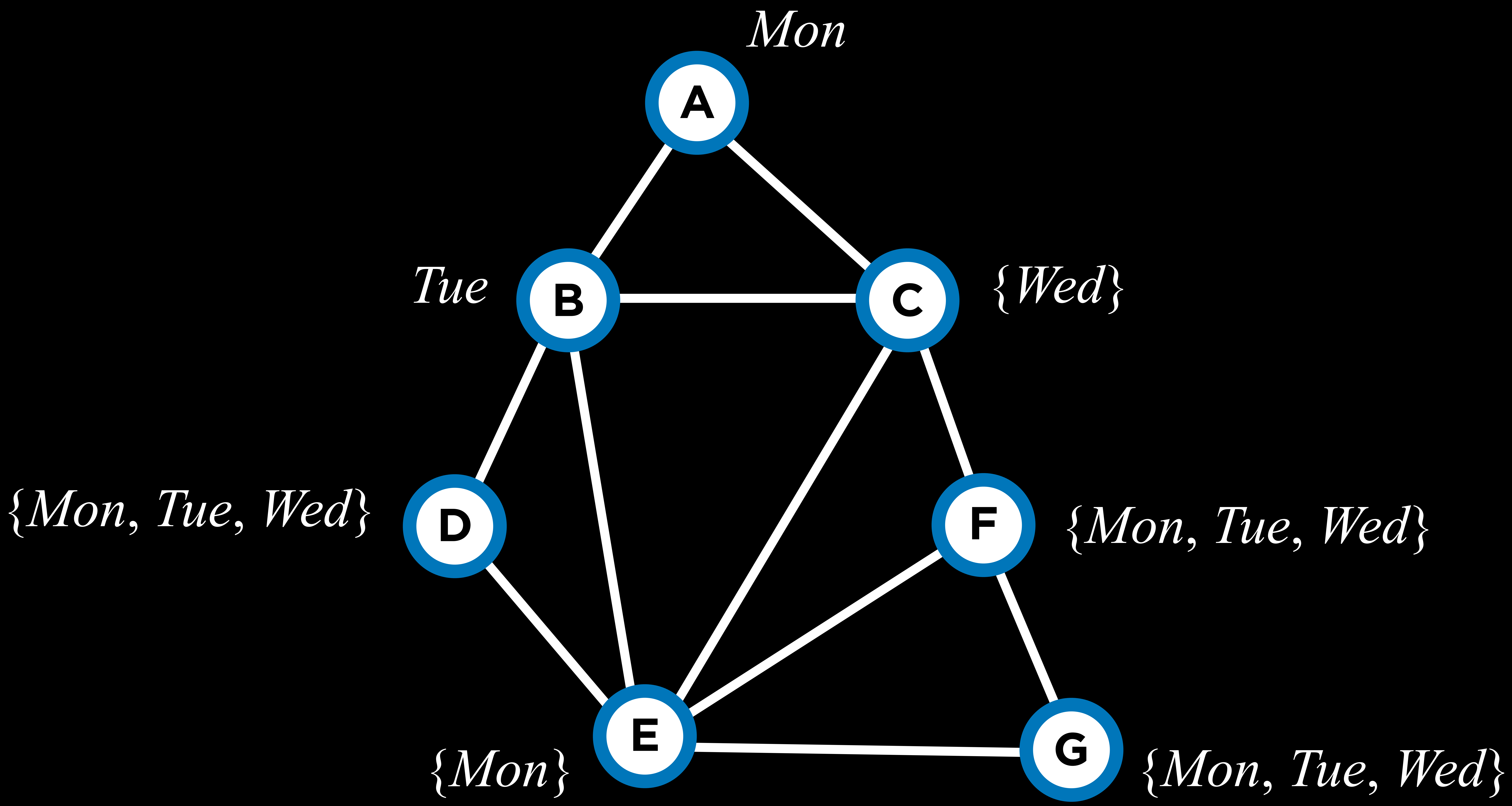


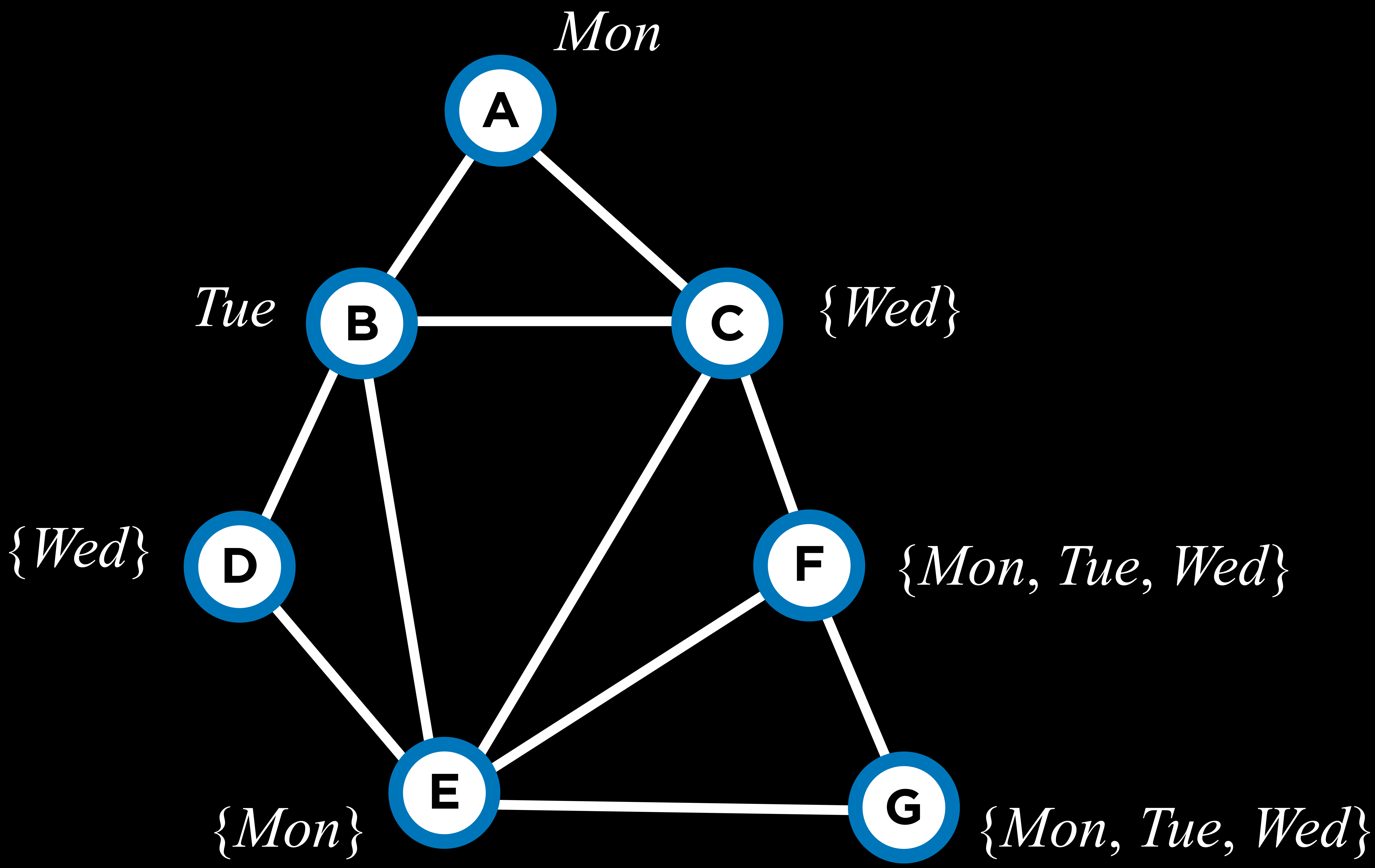
Mon
A
Tue
B
C
{Mon, Tue, Wed}
Mon
D
F
{Mon, Tue, Wed}
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}

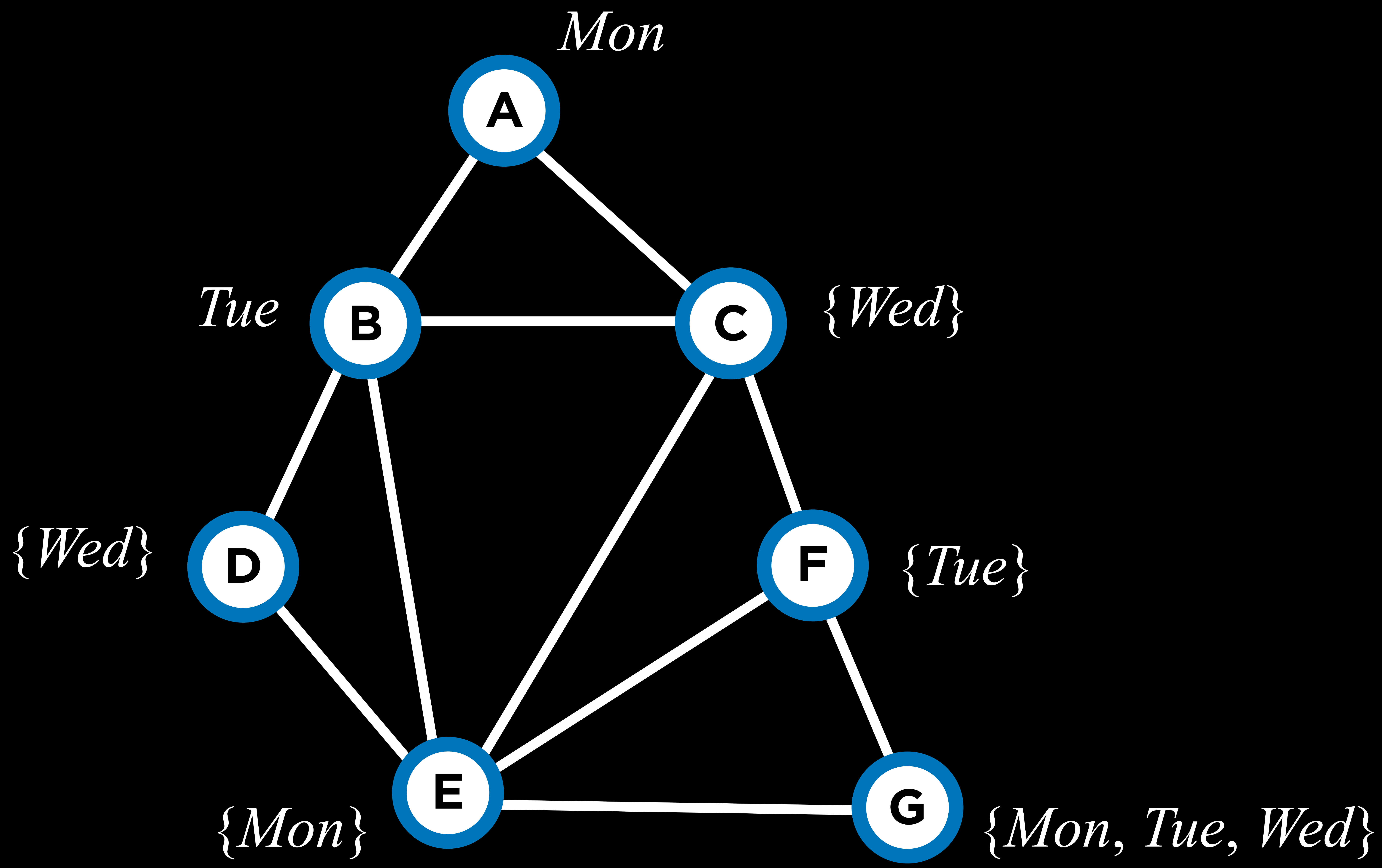

Mon
A
Tue
B
C
{Mon, Tue, Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}
{Mon, Tue, Wed}

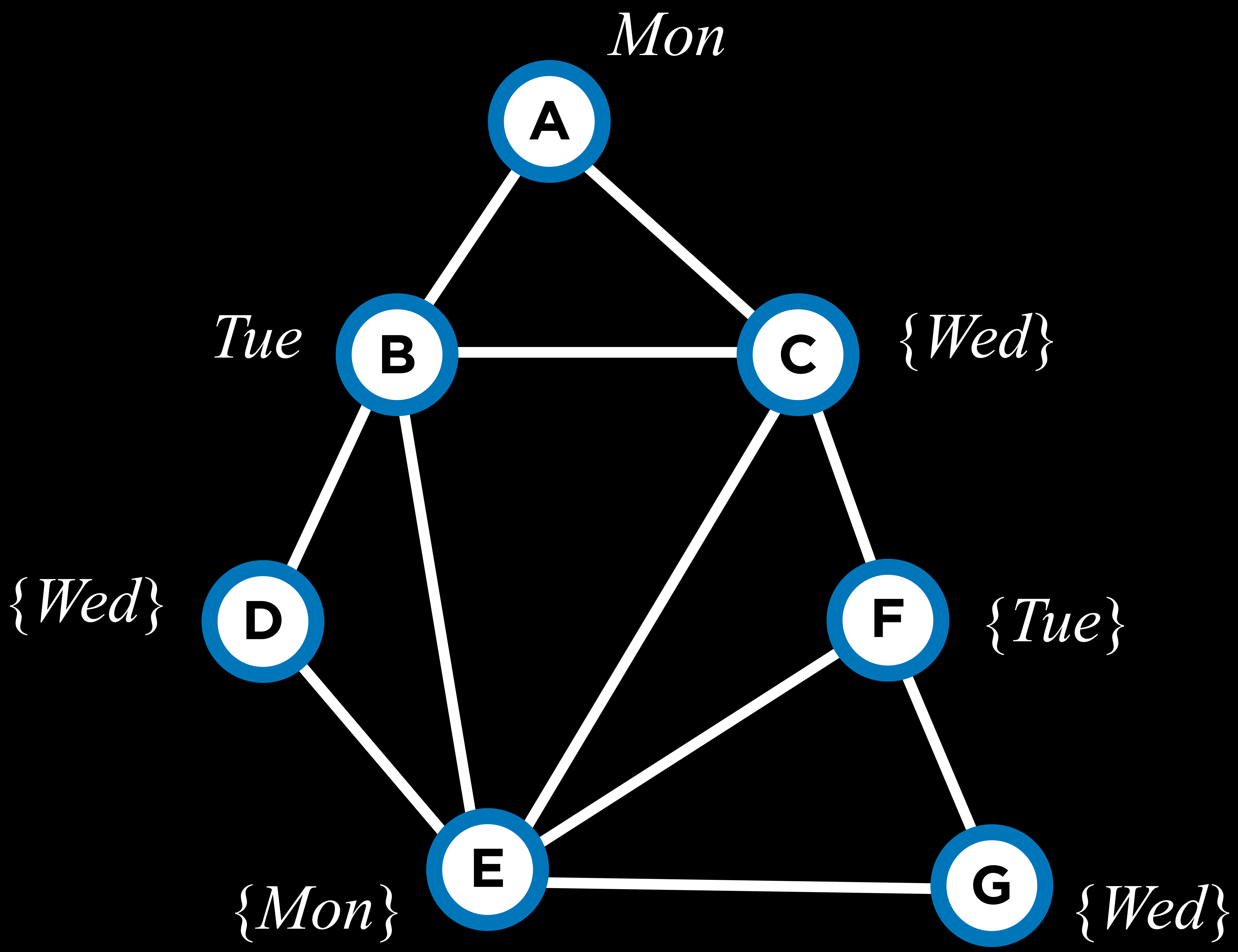


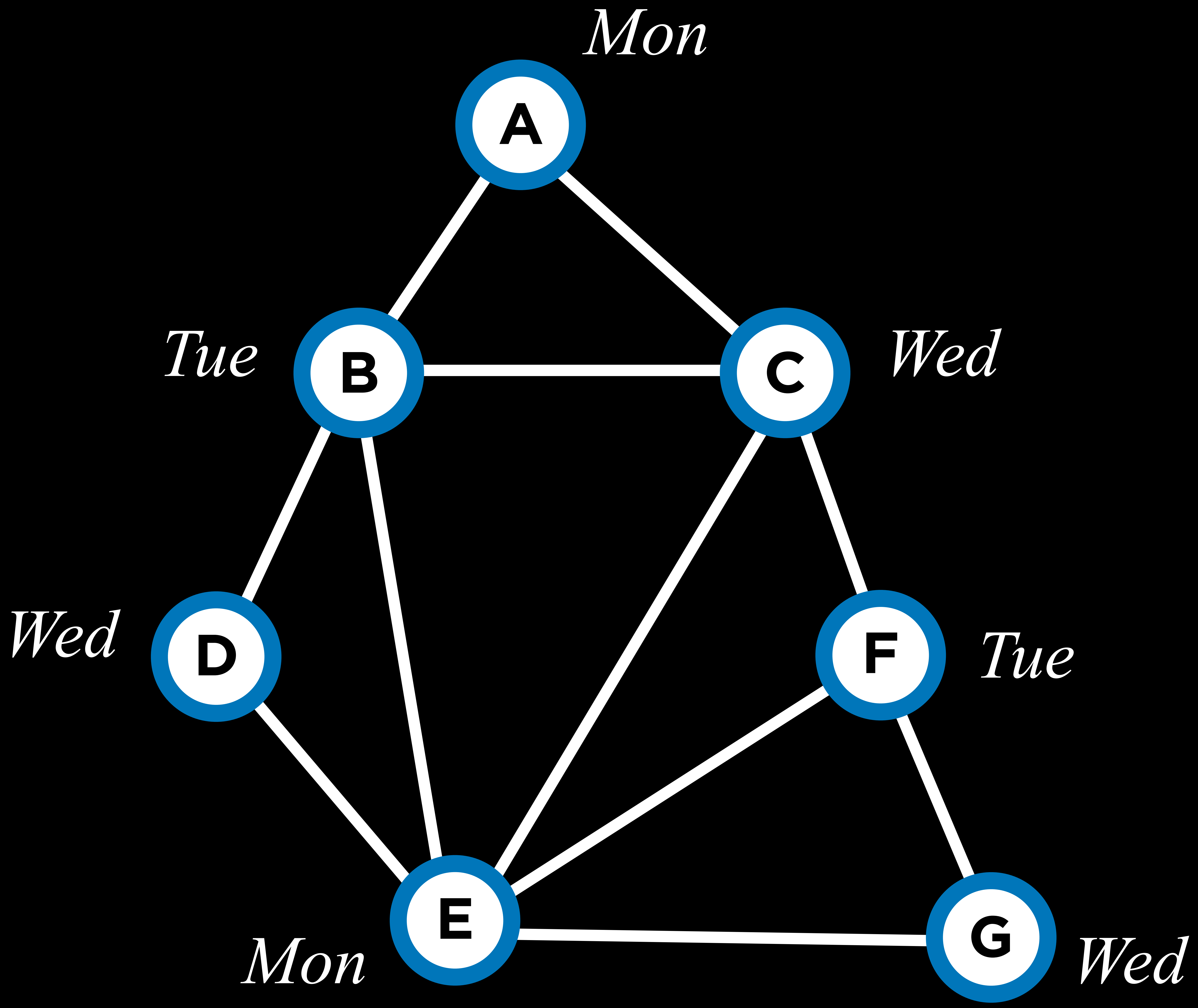
Mon
A
Tue
B
C
{Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}


Mon
A
Tue
B
C
{Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
E
{Mon}
G
{Mon, Tue, Wed}


Mon
A
Tue
B
C
{Wed}
{Wed}
D
F
{Mon, Tue, Wed}
E
{Mon}
G
{Mon, Tue, Wed}


Mon
A
Tue
B
C
{Wed}
{Wed}
D
F
{Tue}
{Mon}
E
G
{Mon, Tue, Wed}


Mon
A
Tue
B
C
{Wed}
{Wed}
D
F
{Tue}
{Mon}
E
G
{Wed}


Mon
A
Tue
B
C
Wed
Wed
D
F
Tue
E
Mon
G
Wed

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, calls $\mathrm{AC\text{-}3}$, starting with a queue of all arcs (Y, X) where Y is a neighbor of X

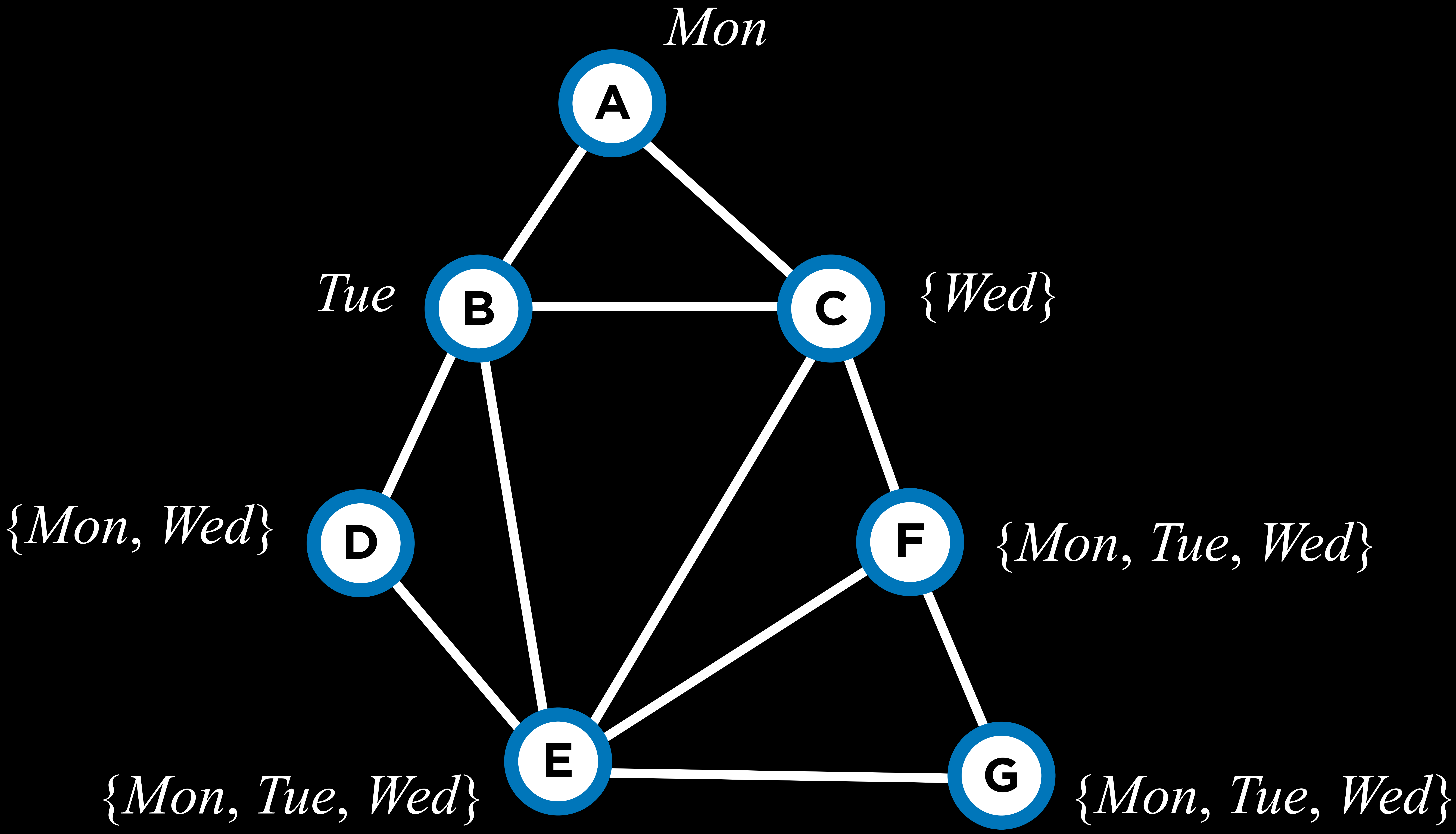
```
function BACKTRACK(assignment, csp):
    if assignment complete: return assignment
    var = SELECT-UNASSIGNED-VAR(assignment, csp)
    for value in DOMAIN-VALUES(var, assignment, csp):
        if value consistent with assignment:
            add {var = value} to assignment
            inferences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

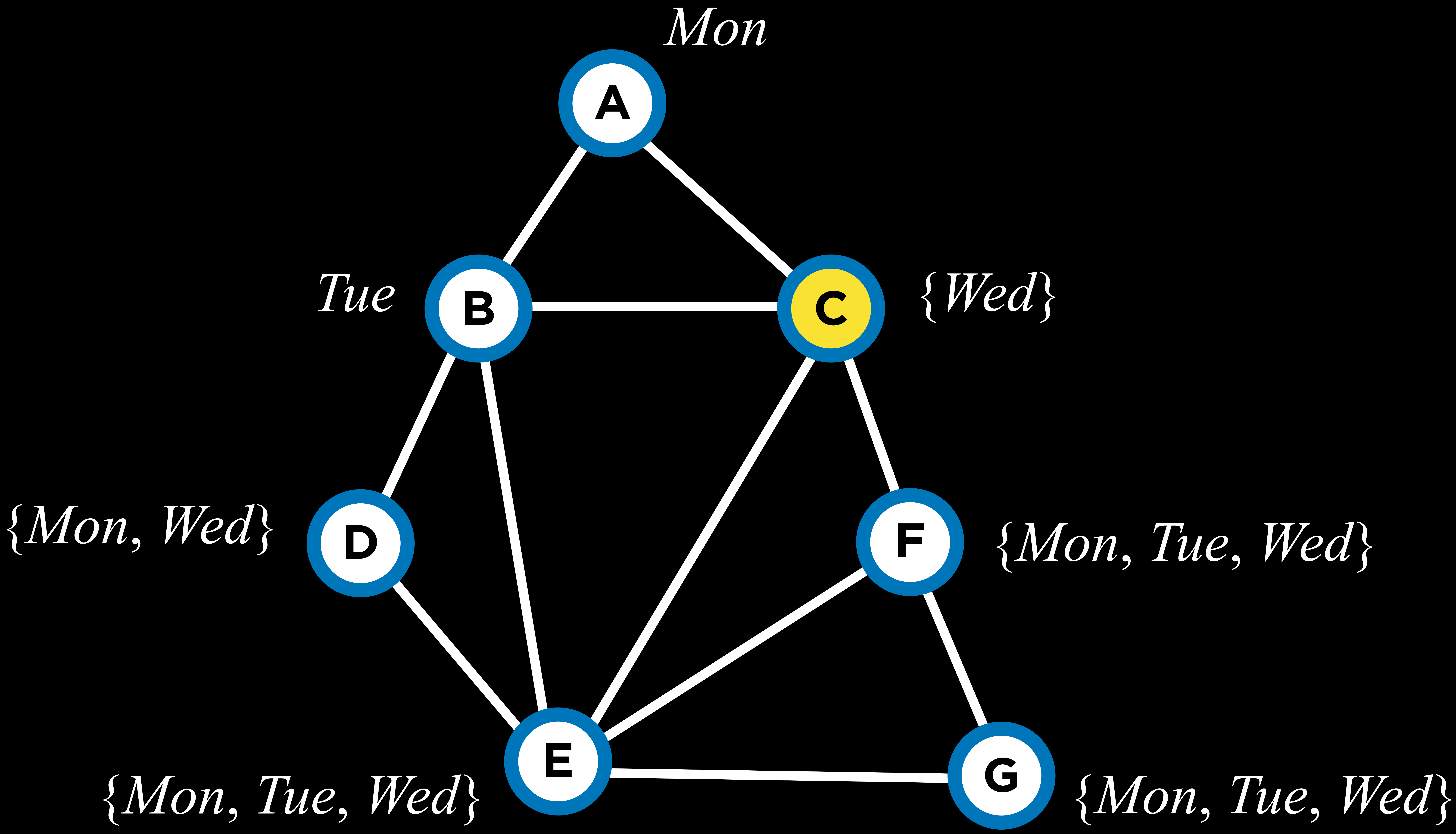
```
function BACKTRACK(assignment, csp):
    if assignment complete: return assignment
    var = SELECT-UNASSIGNED-VAR(assignment, csp)
    for value in DOMAIN-VALUES(var, assignment, csp):
        if value consistent with assignment:
            add {var = value} to assignment
            inferences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

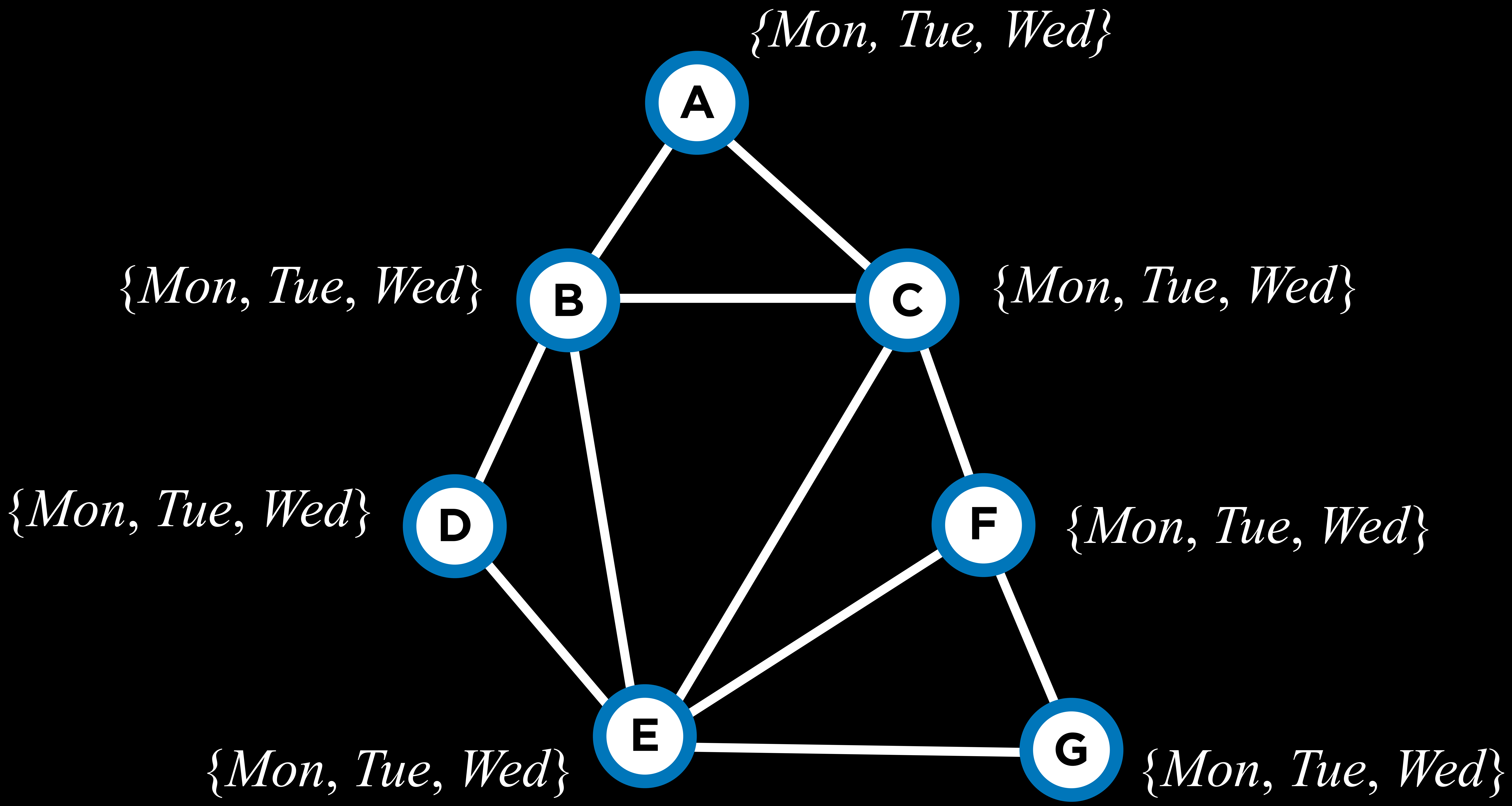
```
function BACKTRACK(assignment, csp):
    if assignment complete: return assignment
    var = SELECT-UNASSIGNED-VAR(assignment, csp)
    for value in DOMAIN-VALUES(var, assignment, csp):
        if value consistent with assignment:
            add {var = value} to assignment
            inferences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

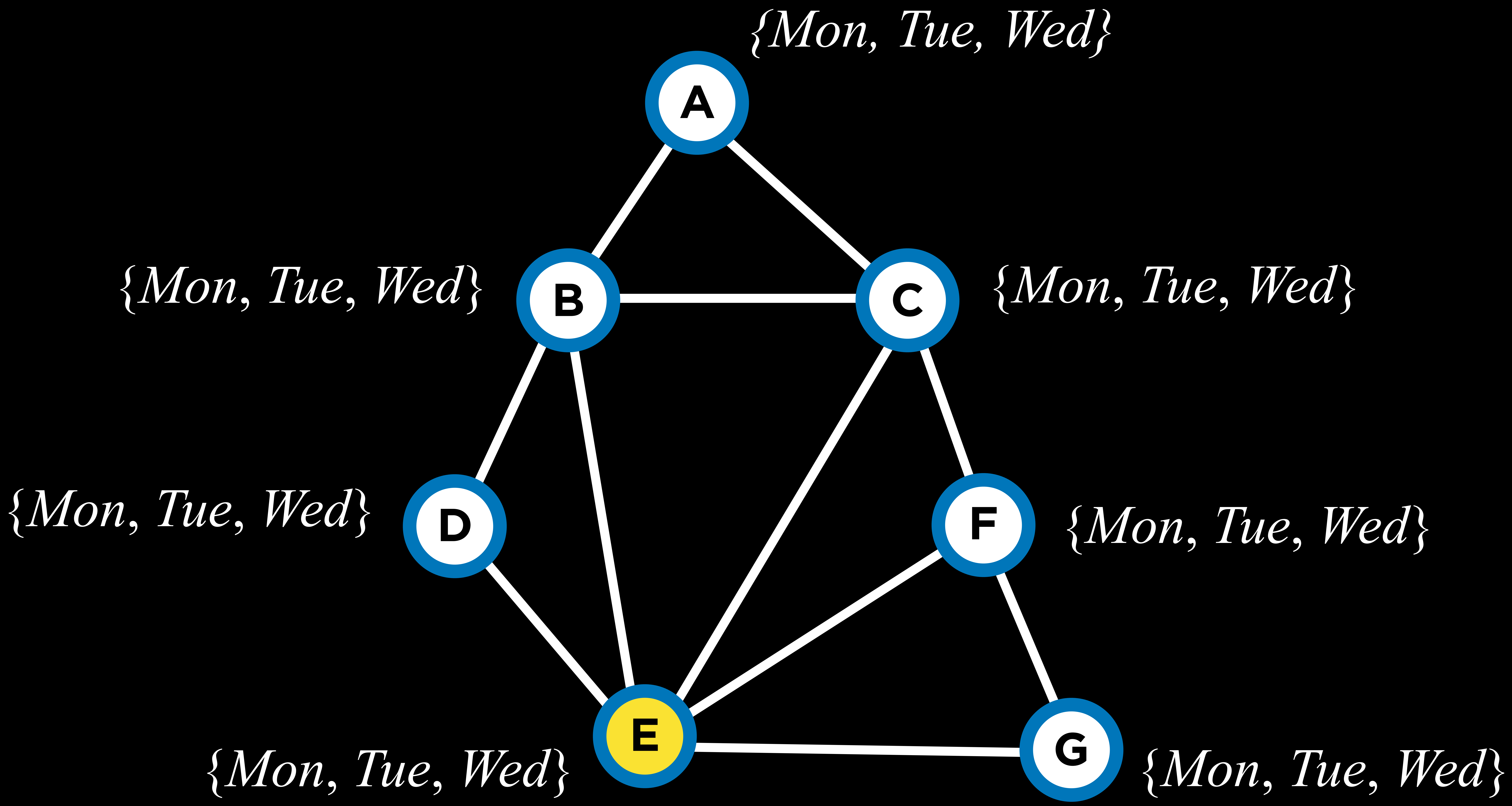
Select-Unassigned-Var

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


Mon
A
Tue
B
C
{Wed}
{Mon, Wed}
D
F
{Mon, Tue, Wed}
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}


Mon
A
Tue
B
C
{Wed}
{Mon, Wed}
D
F
{Mon, Tue, Wed}
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}


A
{Mon, Tue, Wed}
B
{Mon, Tue, Wed}
C
{Mon, Tue, Wed}
D
{Mon, Tue, Wed}
F
{Mon, Tue, Wed}
E
{Mon, Tue, Wed}
G
{Mon, Tue, Wed}

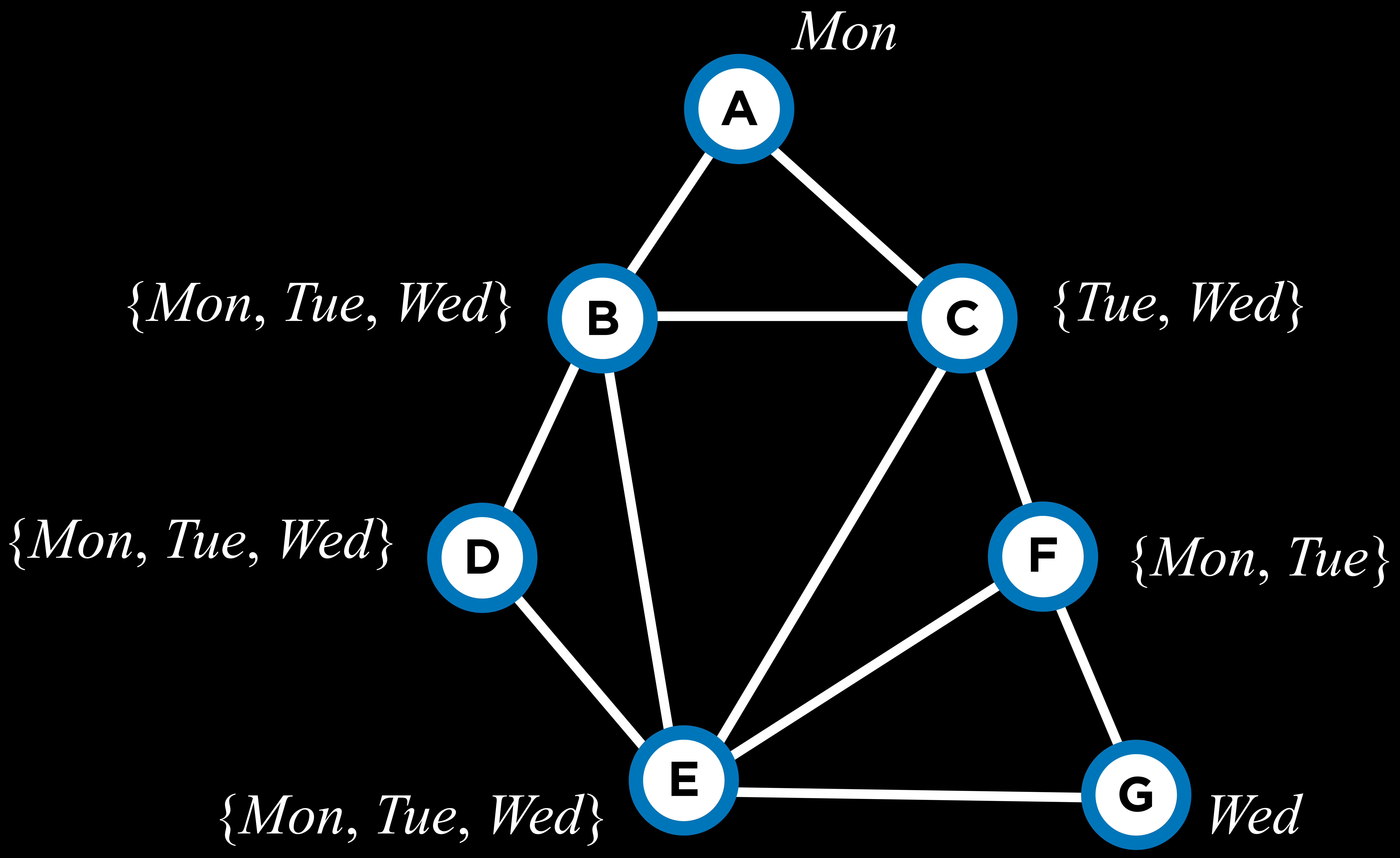

{Mon, Tue, Wed}
A
{Mon, Tue, Wed}
B
C
{Mon, Tue, Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue, Wed}
{Mon, Tue, Wed}
E
G
{Mon, Tue, Wed}

```
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
```

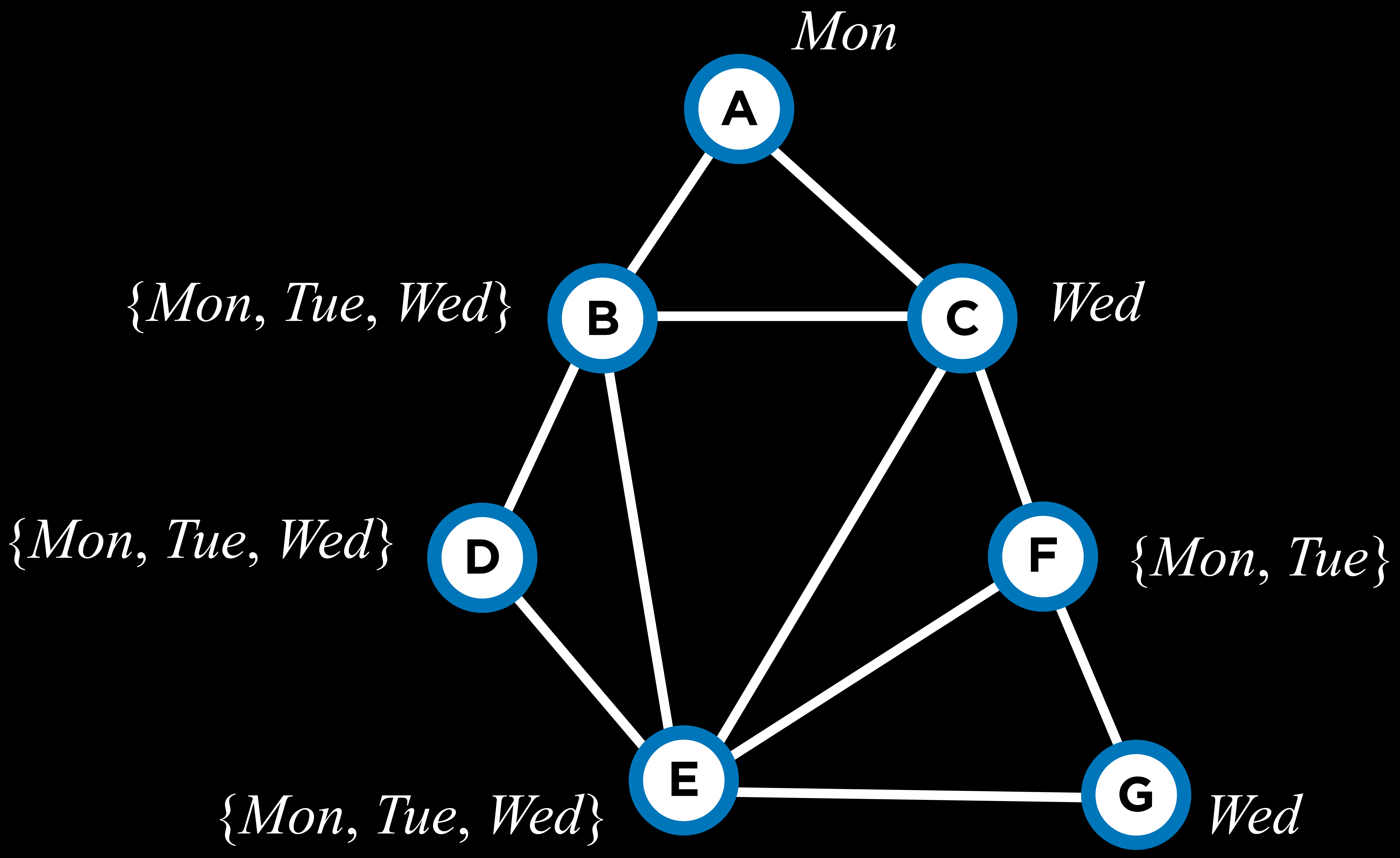
```
function BACKTRACK(assignment, csp):
    if assignment complete: return assignment
    var = SELECT-UNASSIGNED-VAR(assignment, csp)
    for value in DOMAIN-VALUES(var, assignment, csp):
        if value consistent with assignment:
            add {var = value} to assignment
            inferences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

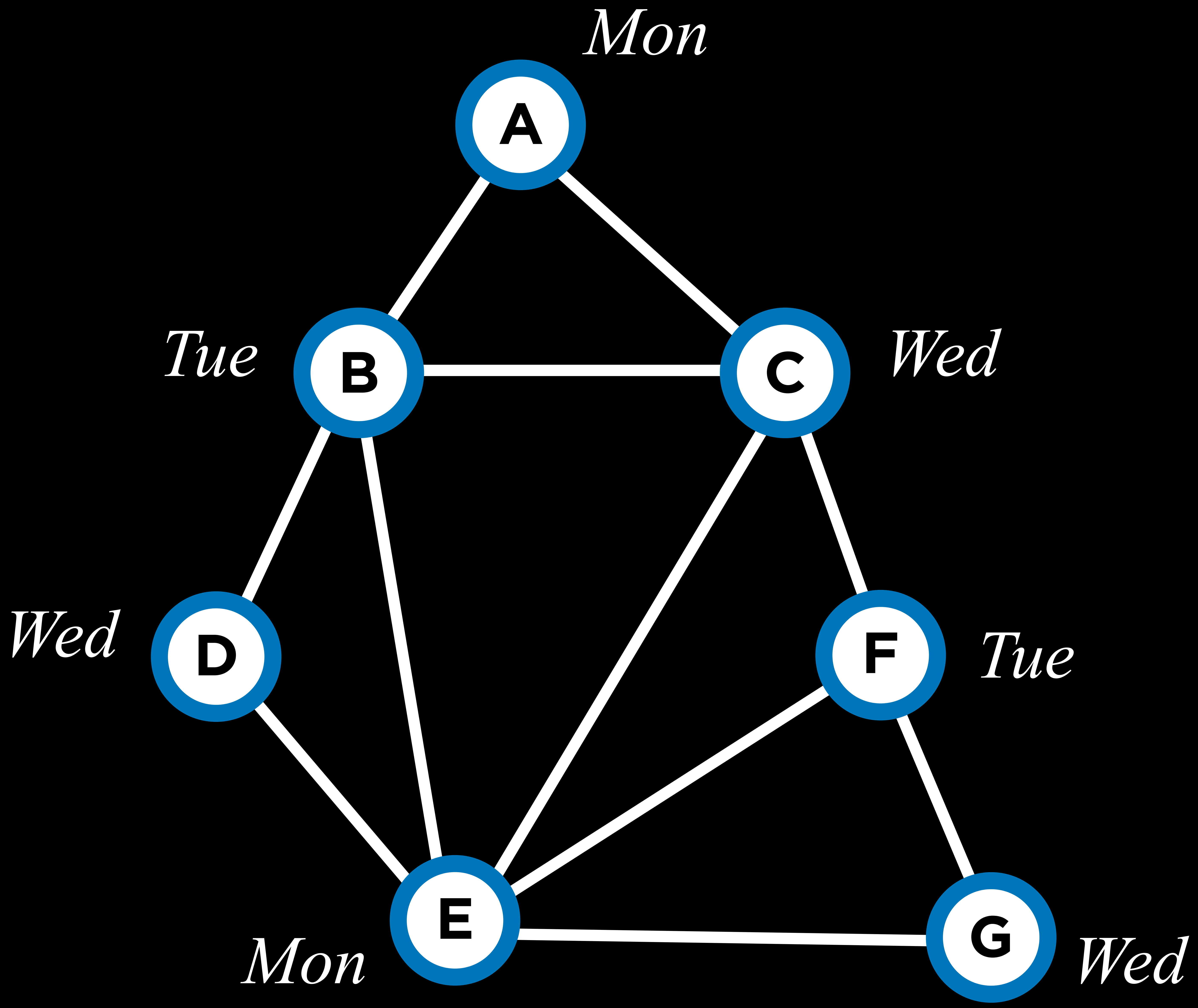
Domain-Values

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

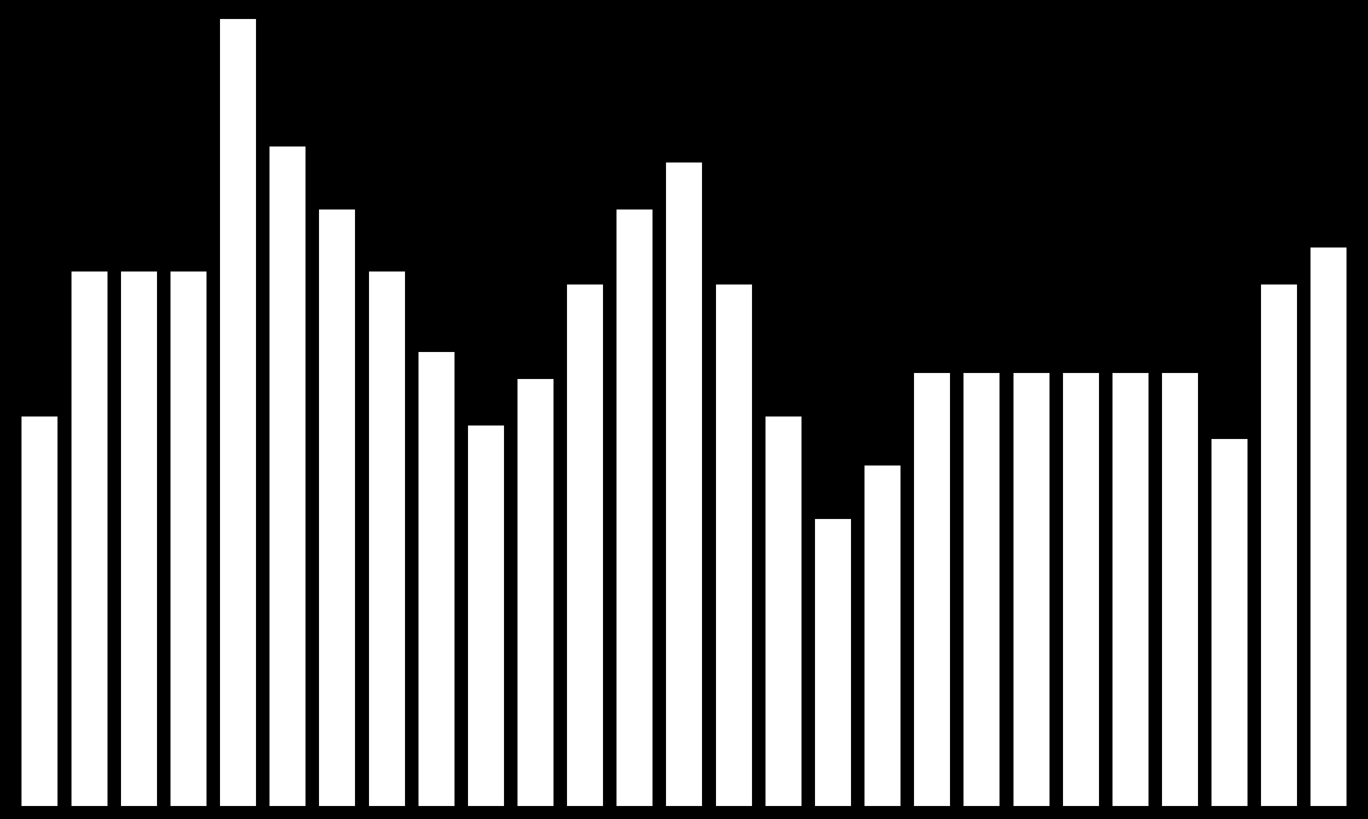


Mon
A
{Mon, Tue, Wed}
B
C
{Tue, Wed}
{Mon, Tue, Wed}
D
F
{Mon, Tue}
{Mon, Tue, Wed}
E
G
Wed


Mon
A
{Mon, Tue, Wed}
B
C
Wed
{Mon, Tue, Wed}
D
F
{Mon, Tue}
E
{Mon, Tue, Wed}
G
Wed


Mon
A
Tue
B
C
Wed
Wed
D
F
Tue
Mon
E
G
Wed

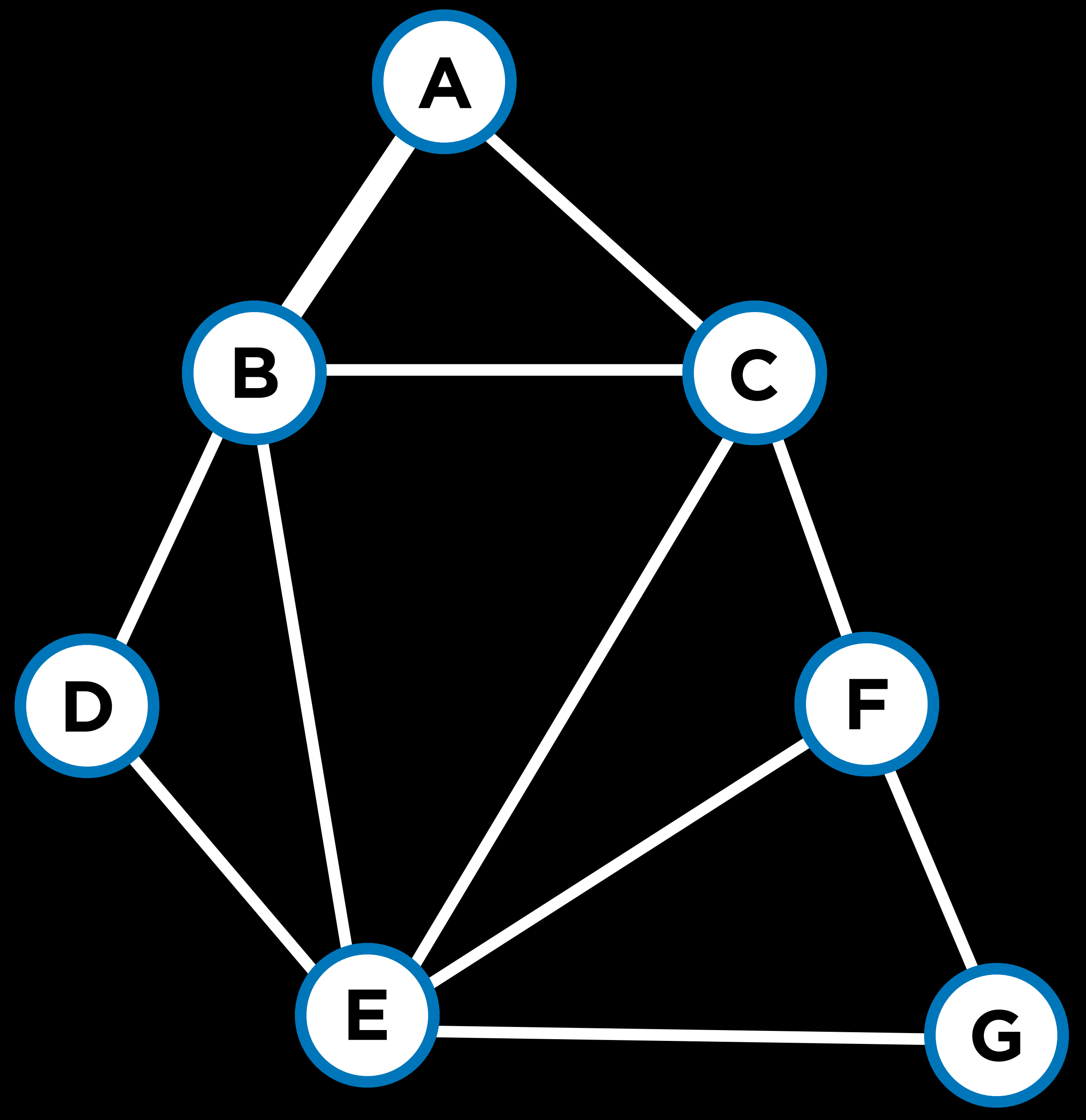
Problem Formulation



Local
Search

$$50x_1 + 80x_2$$
$$5x_1 + 2x_2 \leq 20$$
$$(-10x_1) + (-12x_2) \leq -90$$

Linear
Programming



Constraint
Satisfaction

Optimization

Introduction to

Artificial Intelligence

with Python