Artificial Intelligence

Constraint Satisfaction Problems

LESSON 9

prof. Antonino Staiano
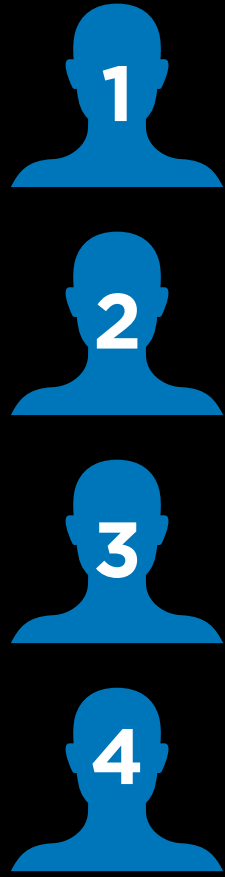
M.Sc. In ''Machine Learning e Big Data'' - University Parthenope of Naples
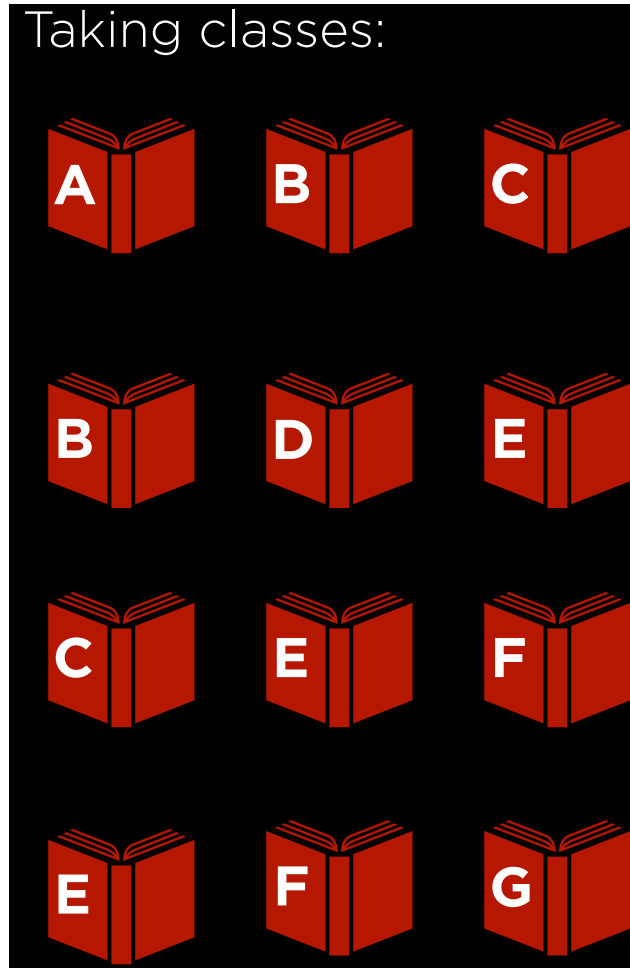
Constraint Satisfaction Problem

- The basic idea of a CSP is
  - having some number of variables that need to take on some values, figure out what values each of those variables should take on
  - However, the variables are subject to particular constraints that are going to limit what values those variables can actually take on
- Let's take a look at a real-world example …
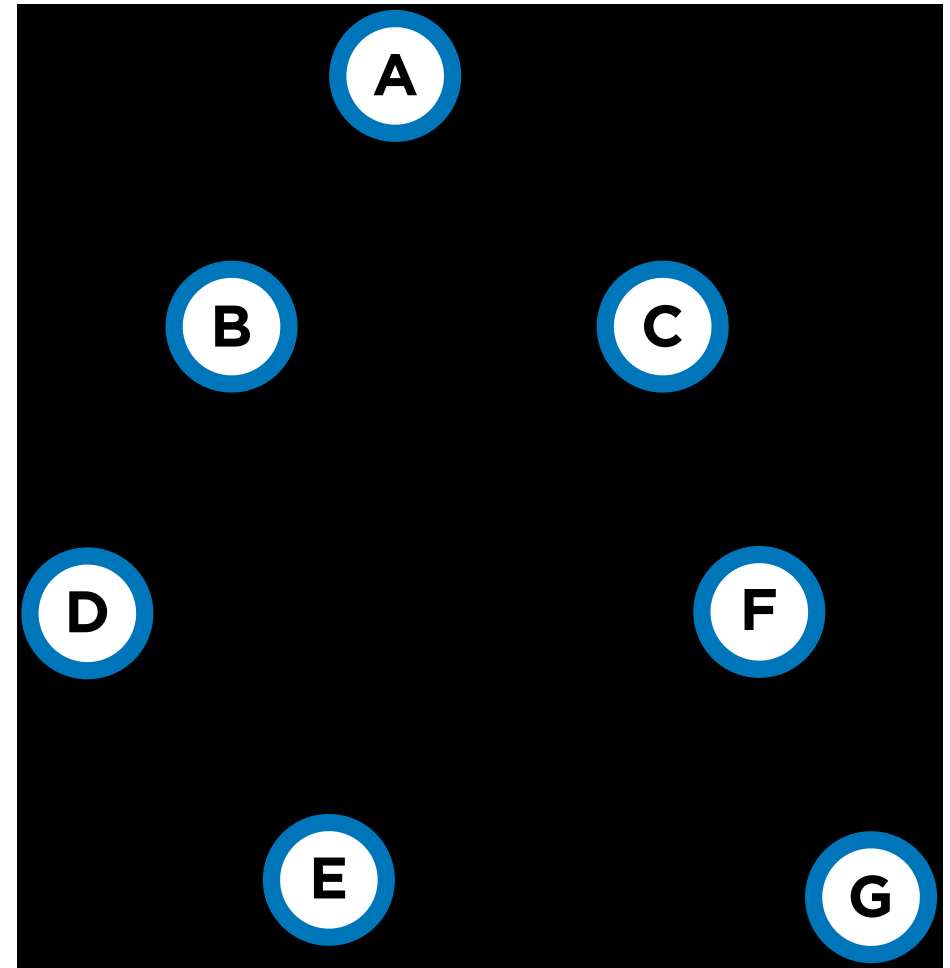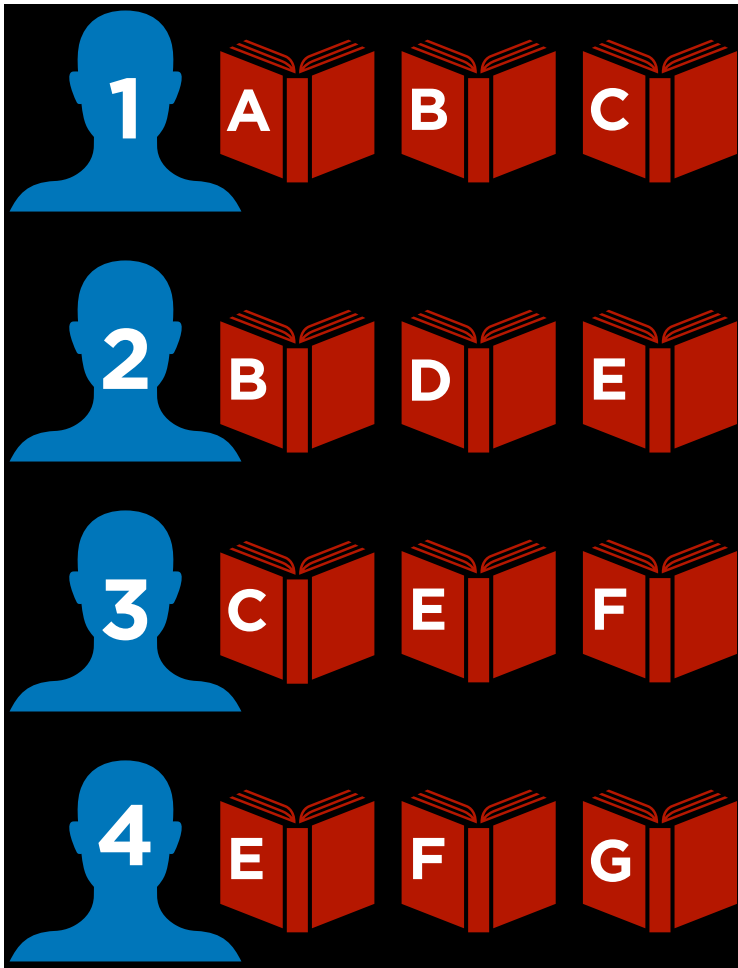
CSP Example: Exam Scheduling
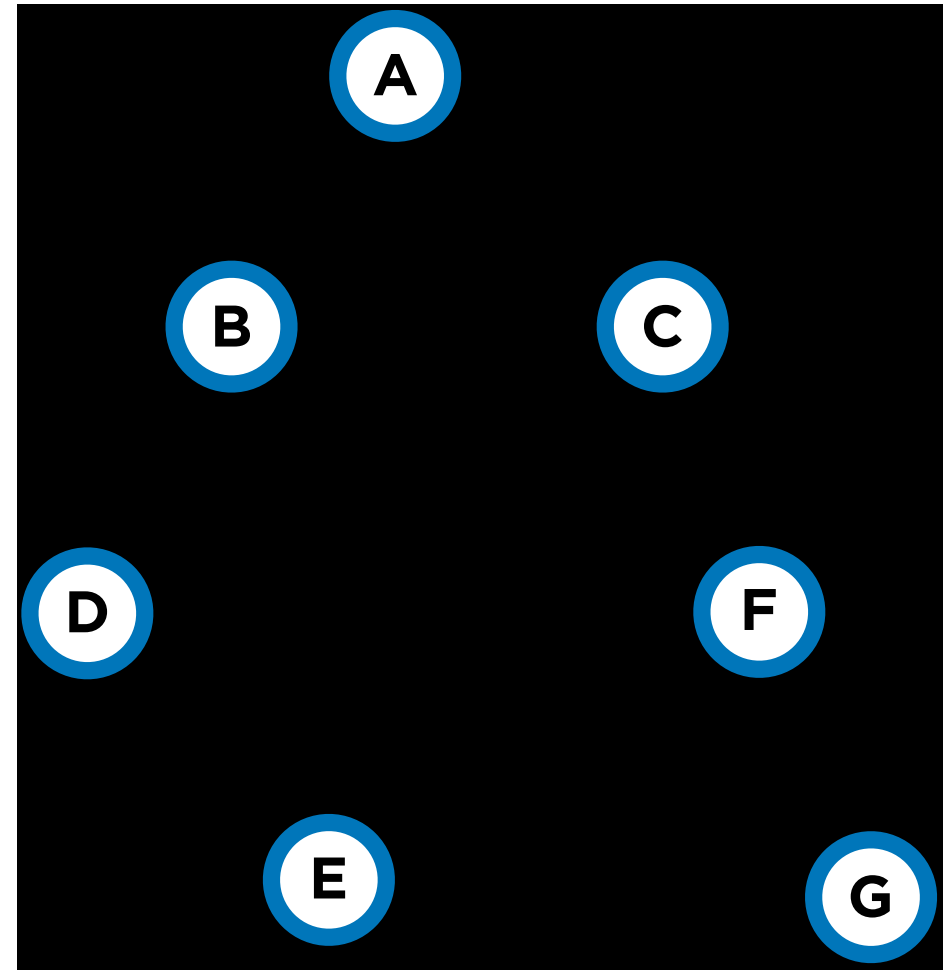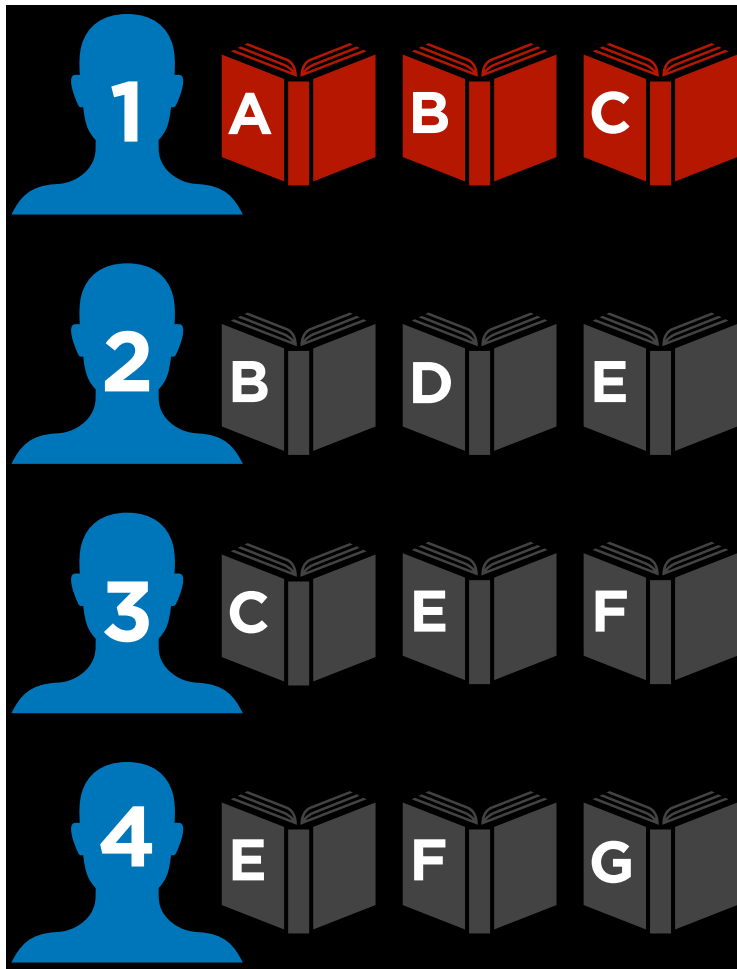
Student:

1

2

3

4

Taking classes:

A A B B C C

B B D D E E

C C E E F F

E E F F G G

Exam slots:

Monday

Tuesday

Wednesday

CSP Example: Exam Scheduling

CSP Example: Exam Scheduling

CSP Example: Exam Scheduling

CSP Example: Exam Scheduling

CSP Example: Exam Scheduling
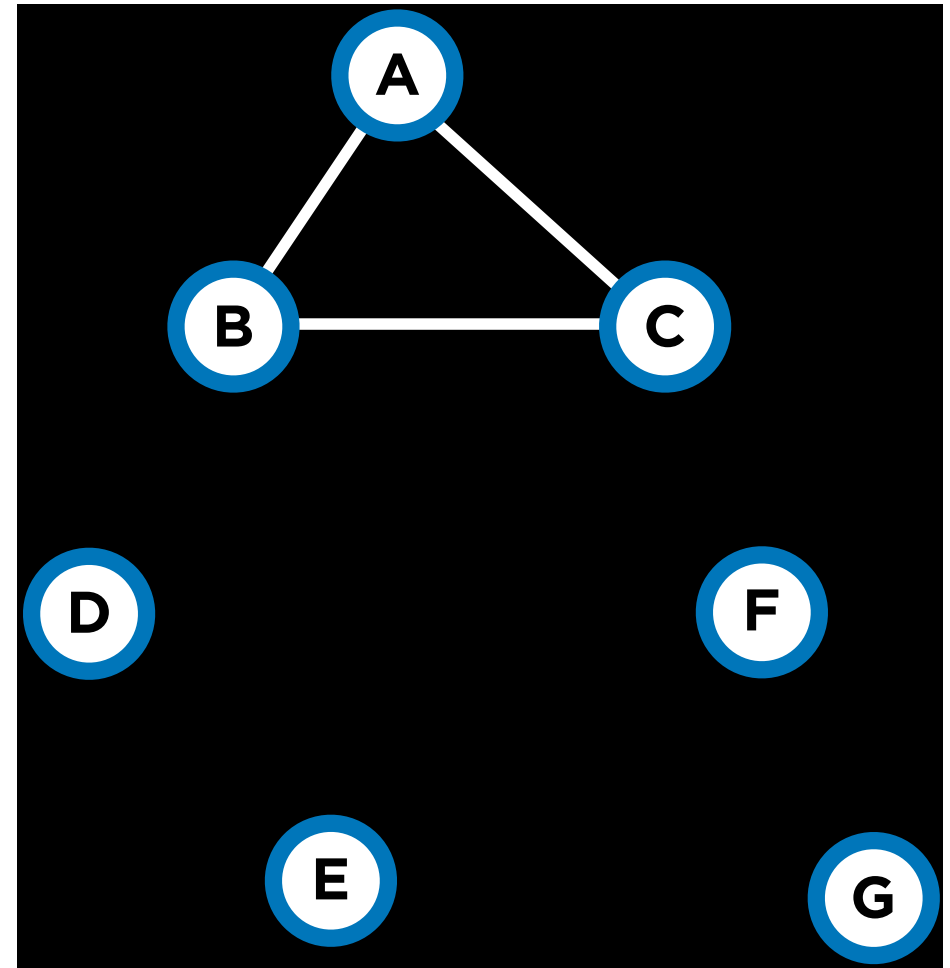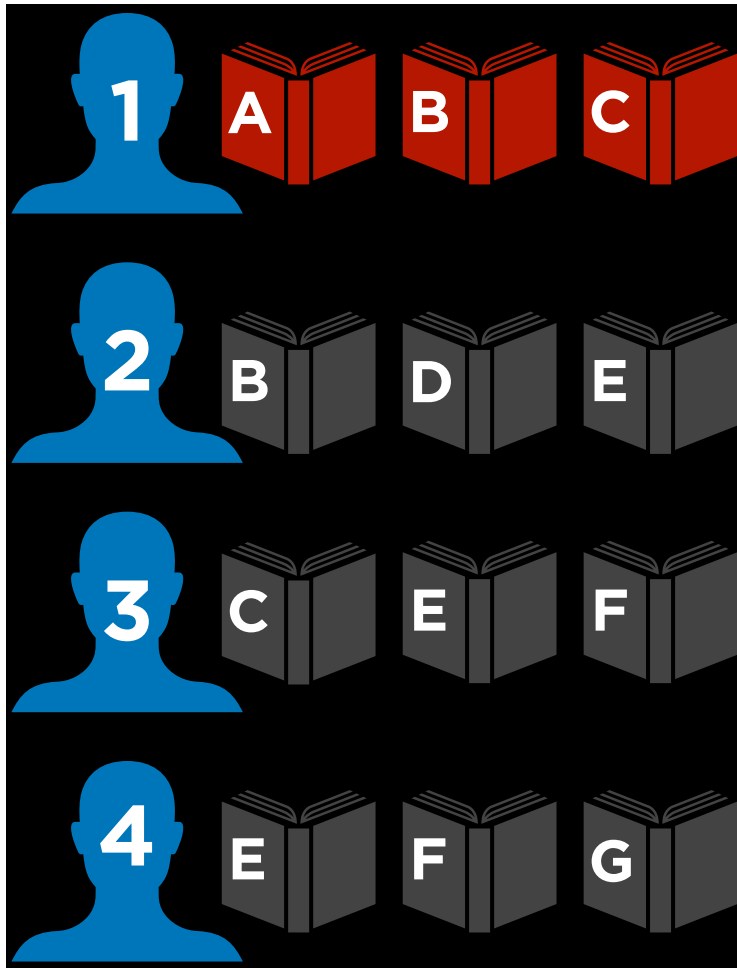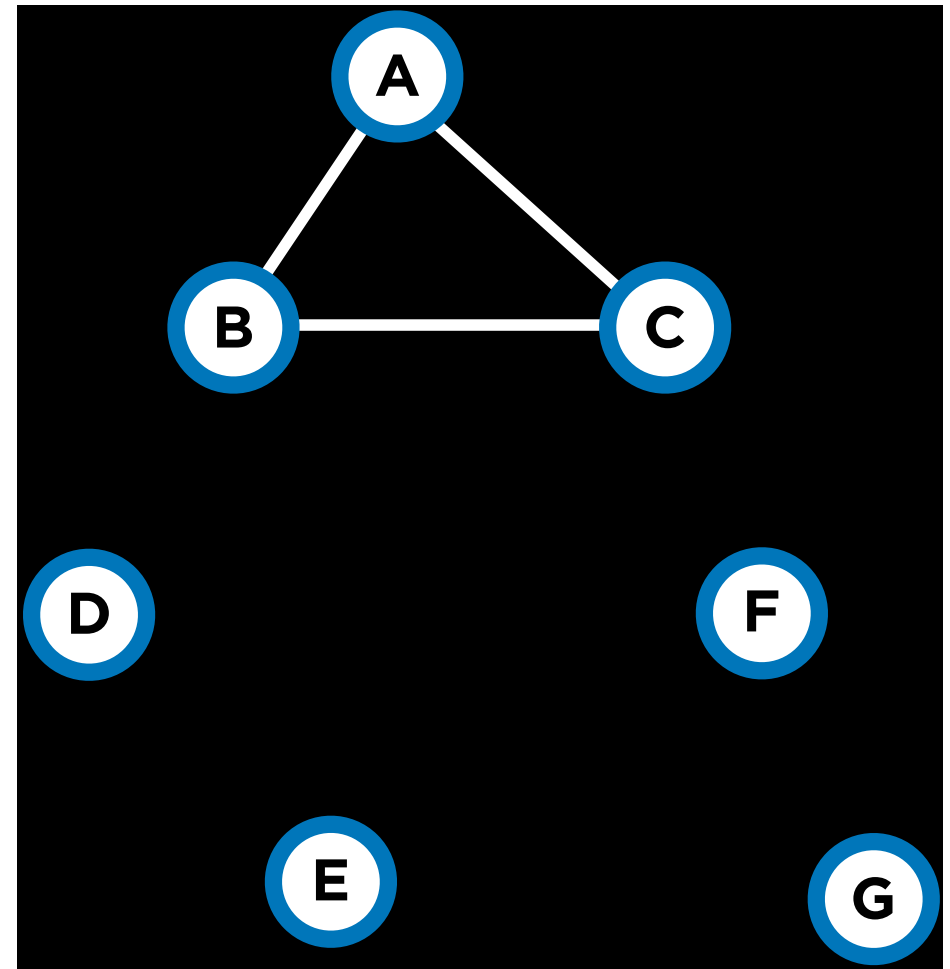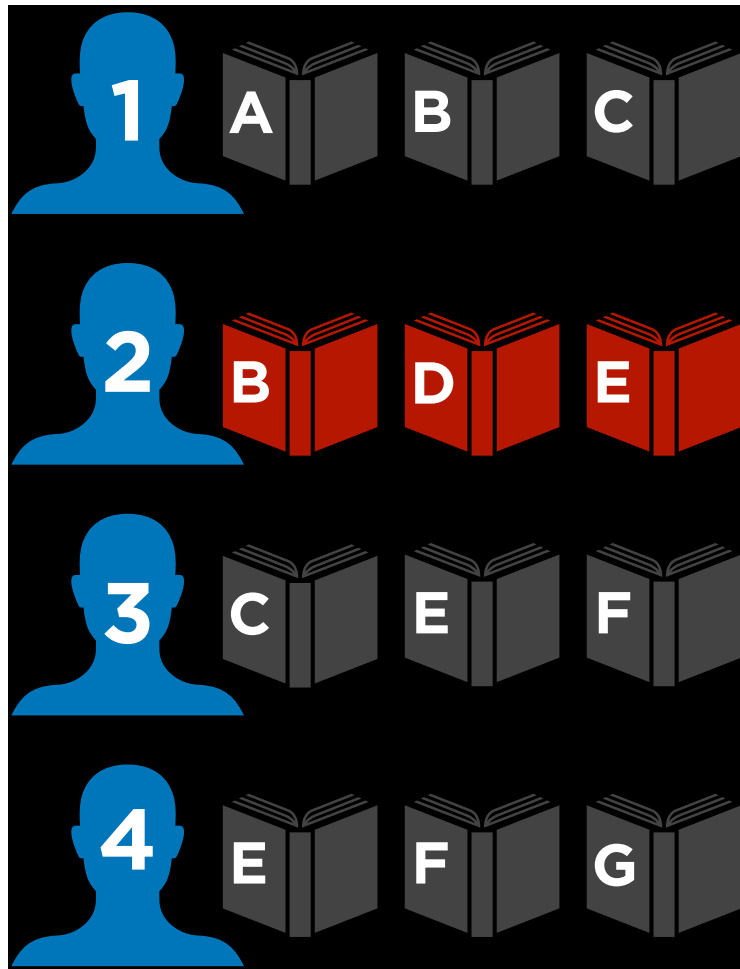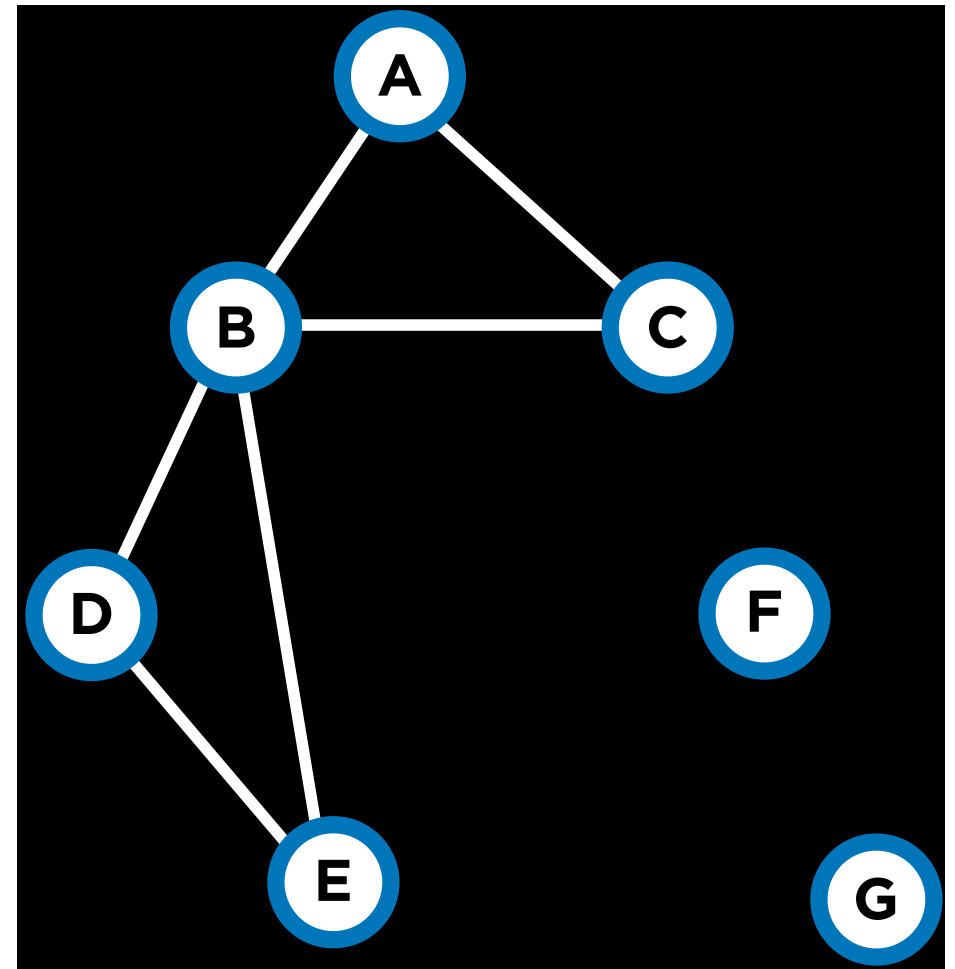
CSP Example: Exam Scheduling
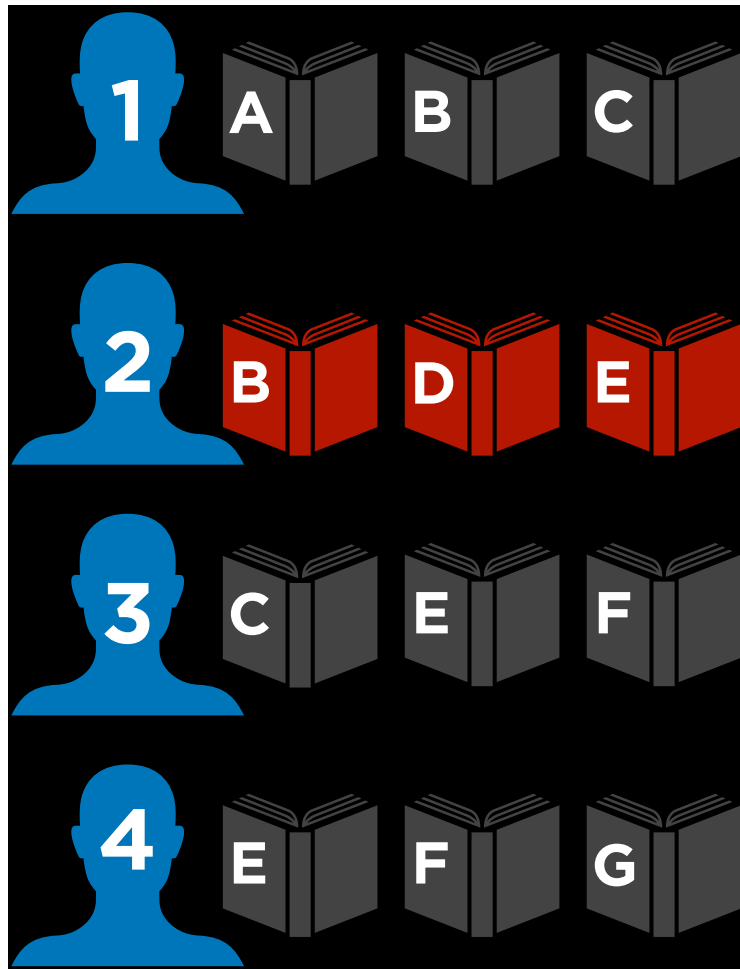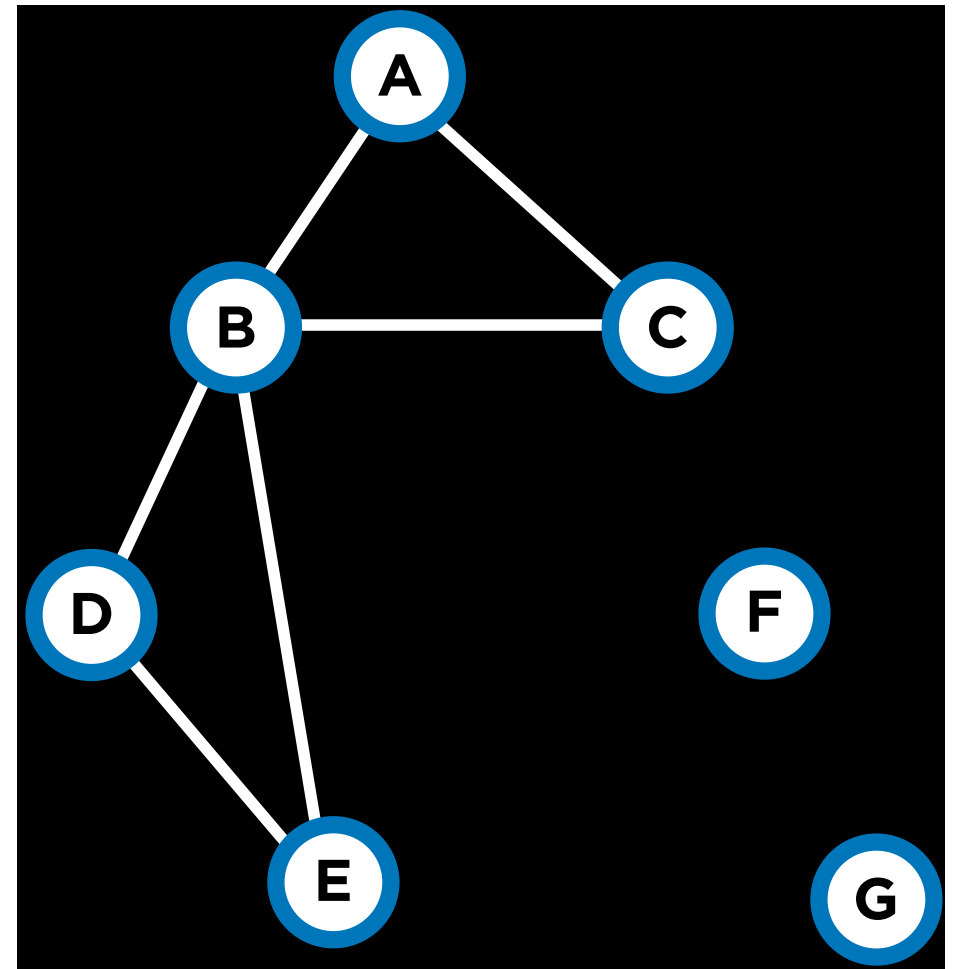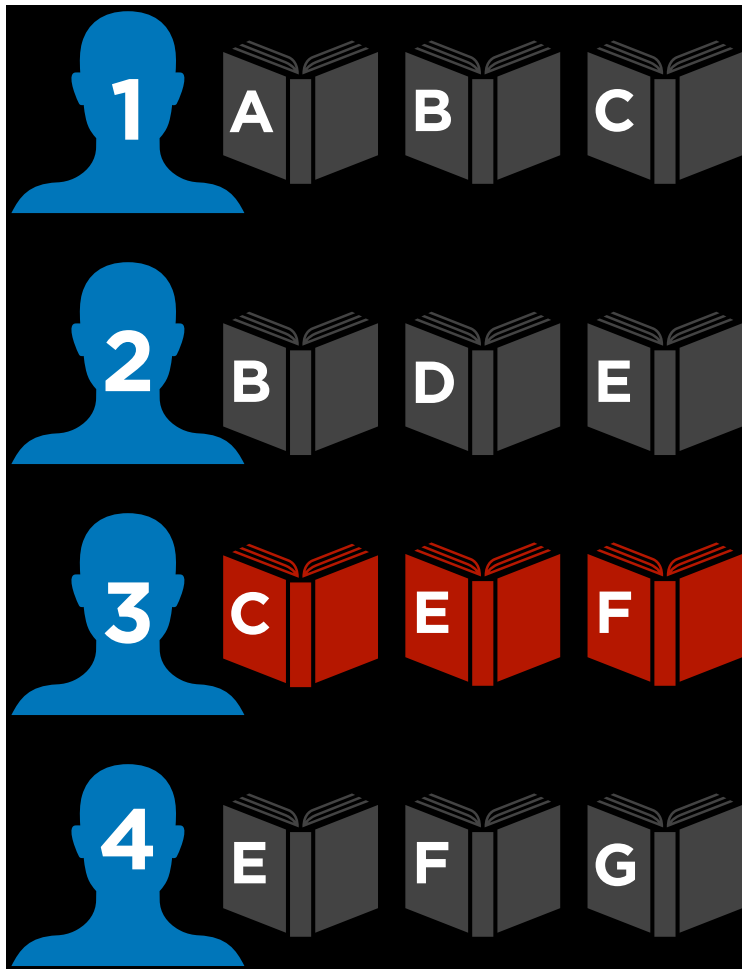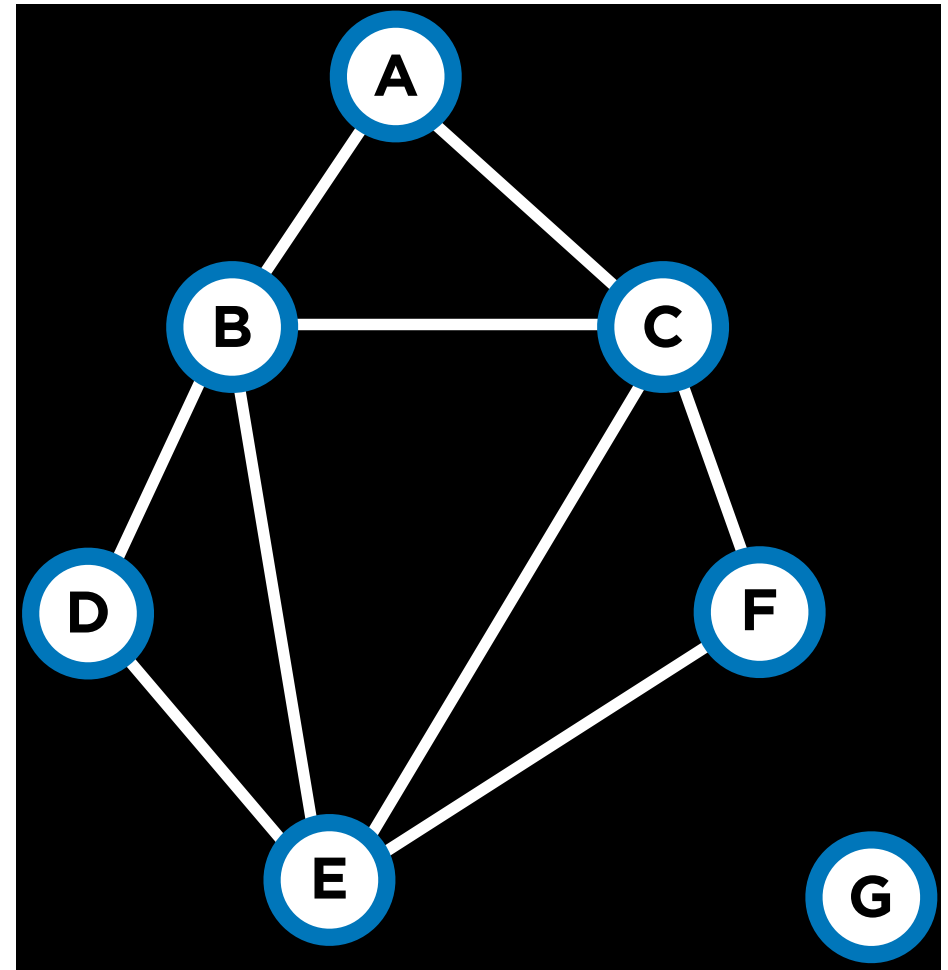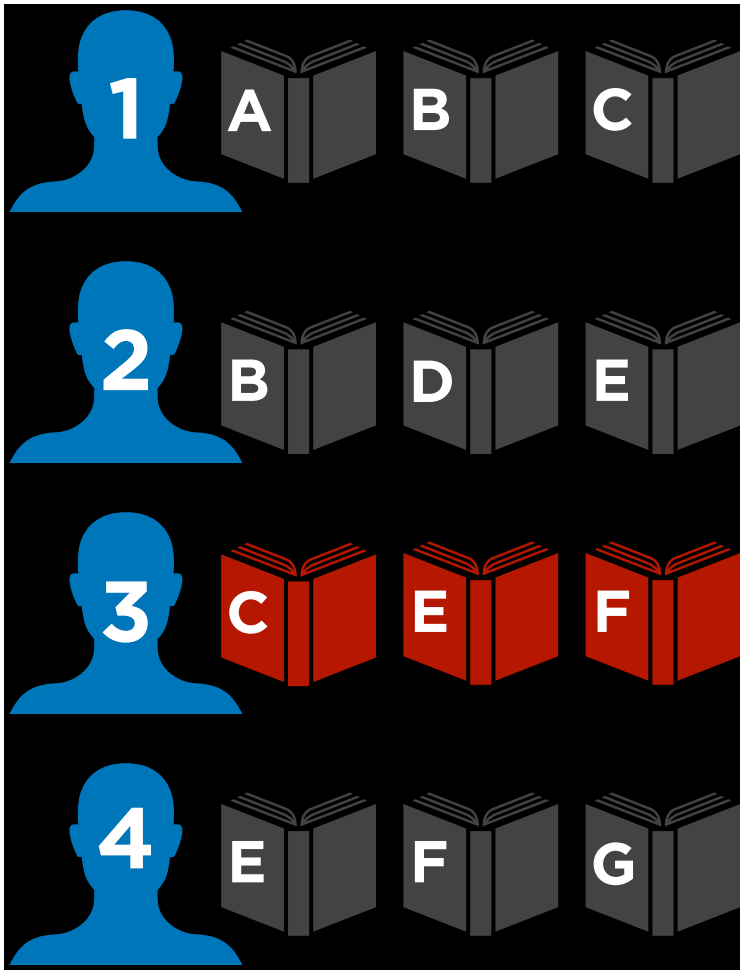
CSP Example: Exam Scheduling

CSP Example: Exam Scheduling
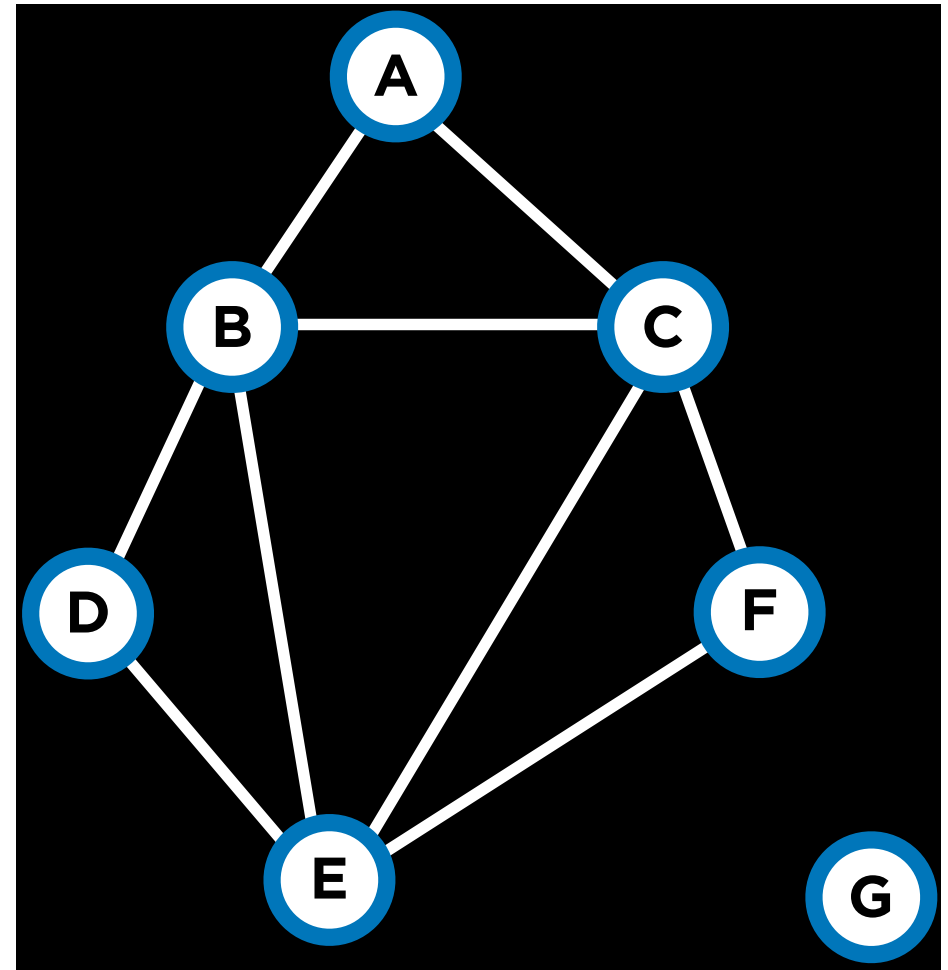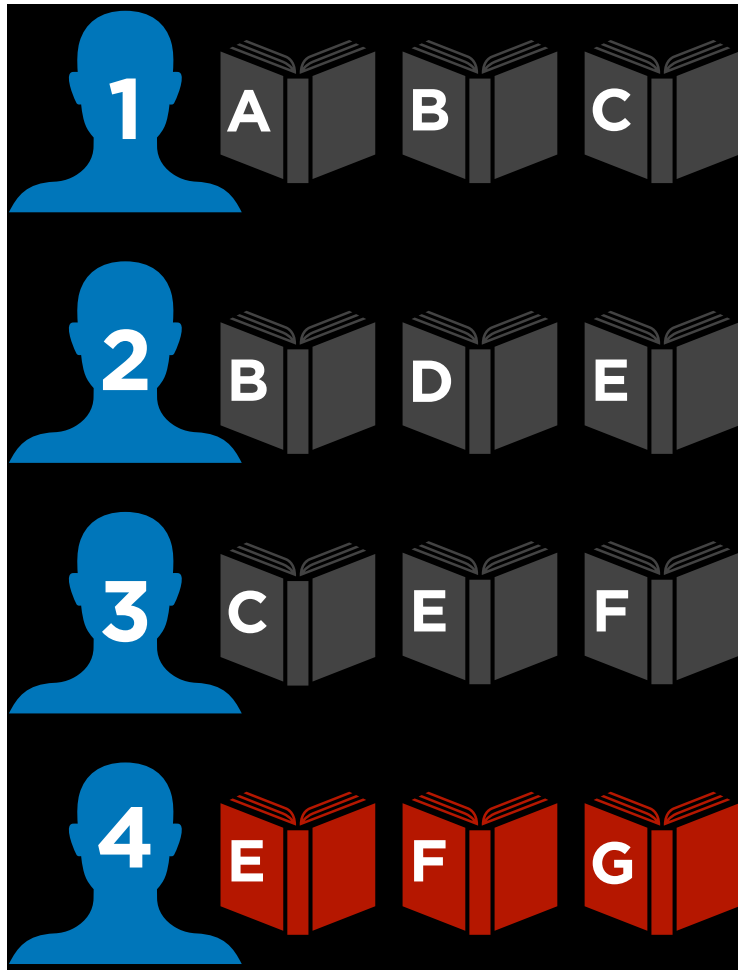
CSP Example: Exam Scheduling
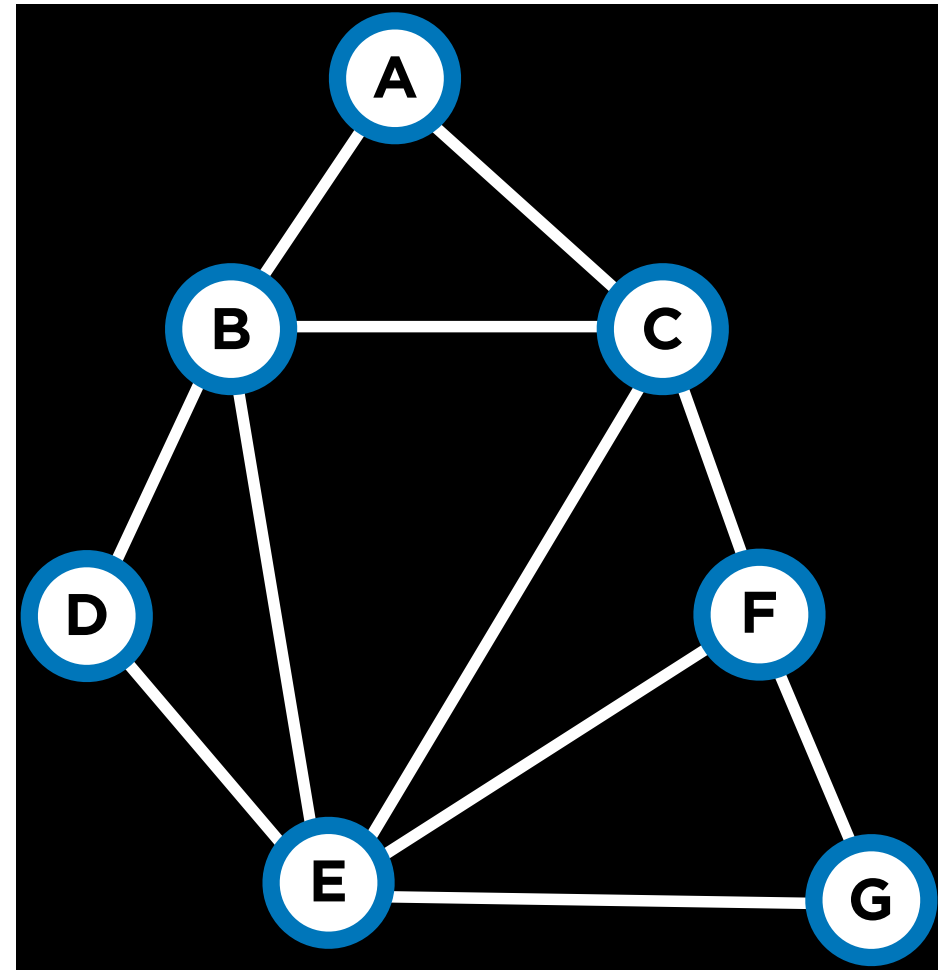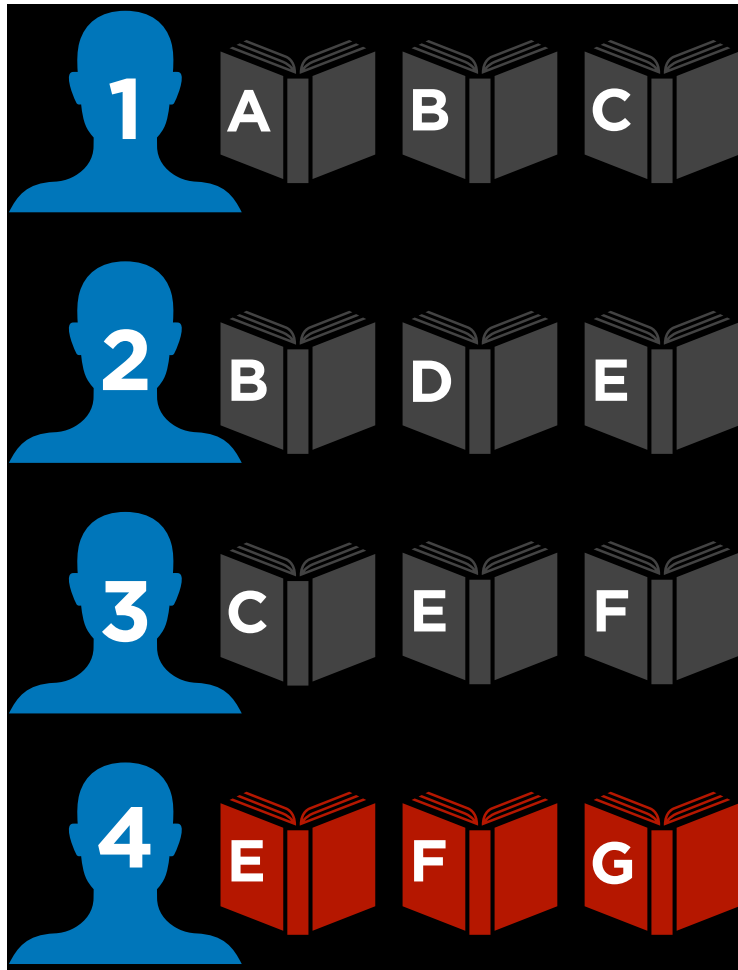
Constraint Graph for Exam Scheduling

- We end up with a graphical representation of all the variables and the constraints between those variables

- In this case, the constraints are inequality constraints

  - e.g., the A-B edge means that the values A takes on cannot be the same as B values
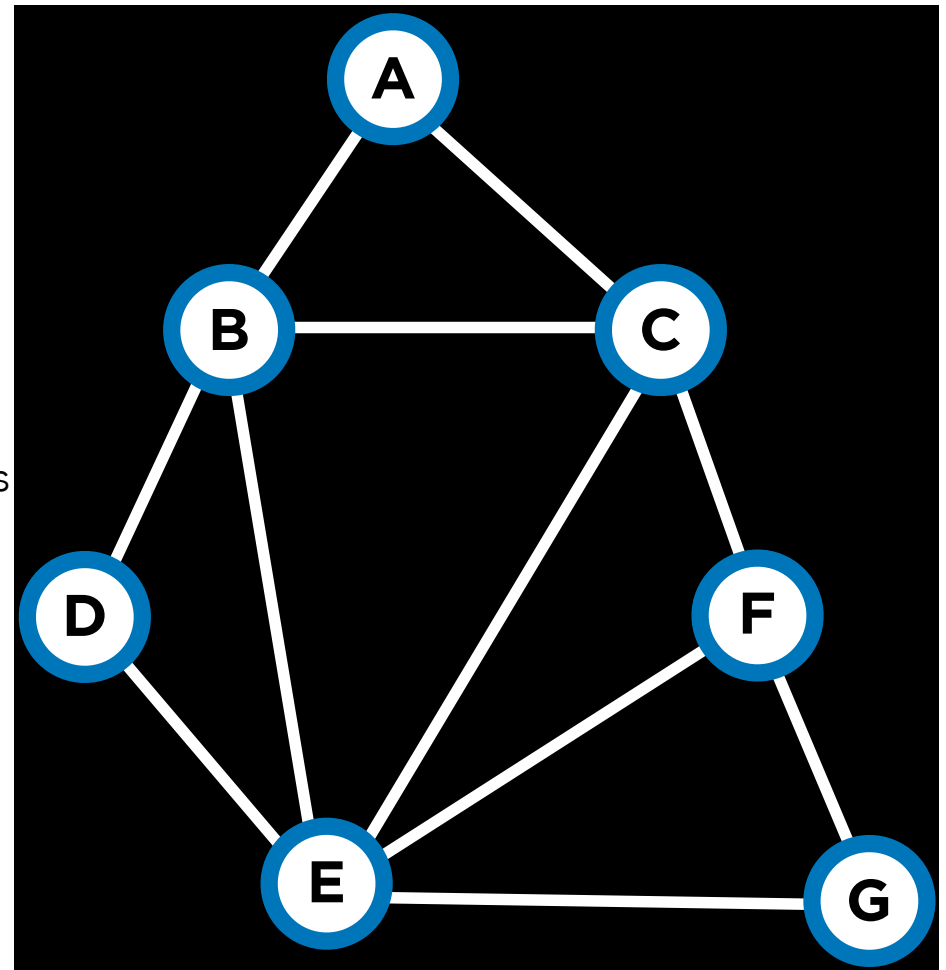
Constraint Satisfaction Problem

- Set of variables $\{X_1, X_2, \ldots, X_n\}$

- Set of domains for each variable $\{D_1, D_2, \ldots, D_n\}$

- Set of constraints C

- CSPs deal with assignments of values to variables
  - A complete assignment is one in which every variable is assigned a value, and a solution to a CSP is a consistent, complete assignment
  - A partial assignment is one that leaves some variables unassigned
  - A partial solution is a partial assignment that is consistent

Constraint Satisfaction Problem: Sudoku

Variables

{(0, 2), (1, 1), (1, 2), (2, 0), ...}

Domains

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

for each variable

Constraints

{(0, 2) ≠ (1, 1) ≠ (1, 2) ≠ (2, 0), ...}

Exam Scheduling Problem Formulation

Variables

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

Domains

$\{Monday, Tuesday, Wednesday\}$

for each variable

Constraints

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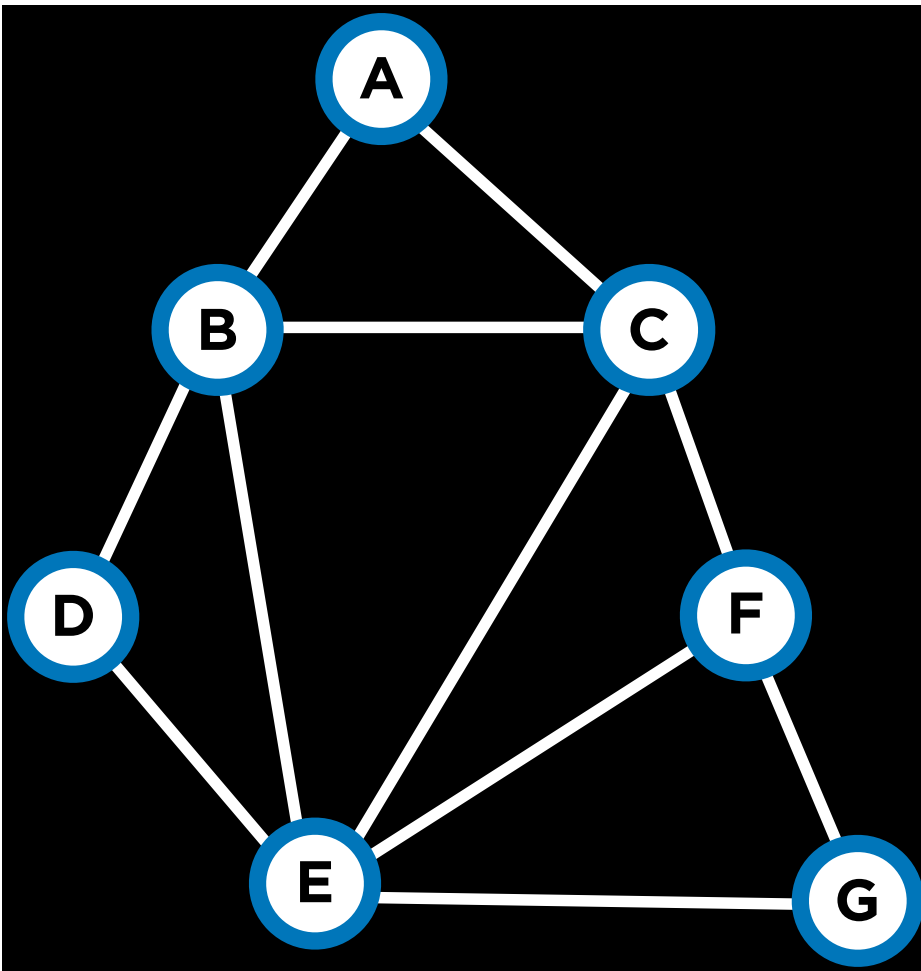$C{\neq}F, D{\neq}E, E{\neq}F, E{\neq}G, F{\neq}G\}$

Constraints

- Hard
  - Constraints that must be satisfied in a correct solution

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

- Soft
  - Constraints that express some notion of which solutions are preferred over others
    - The goal is to try to maximize the preference, that is, the preferences should be satisfied as much as possible

Exam Scheduling as a Hard Constraint Problem
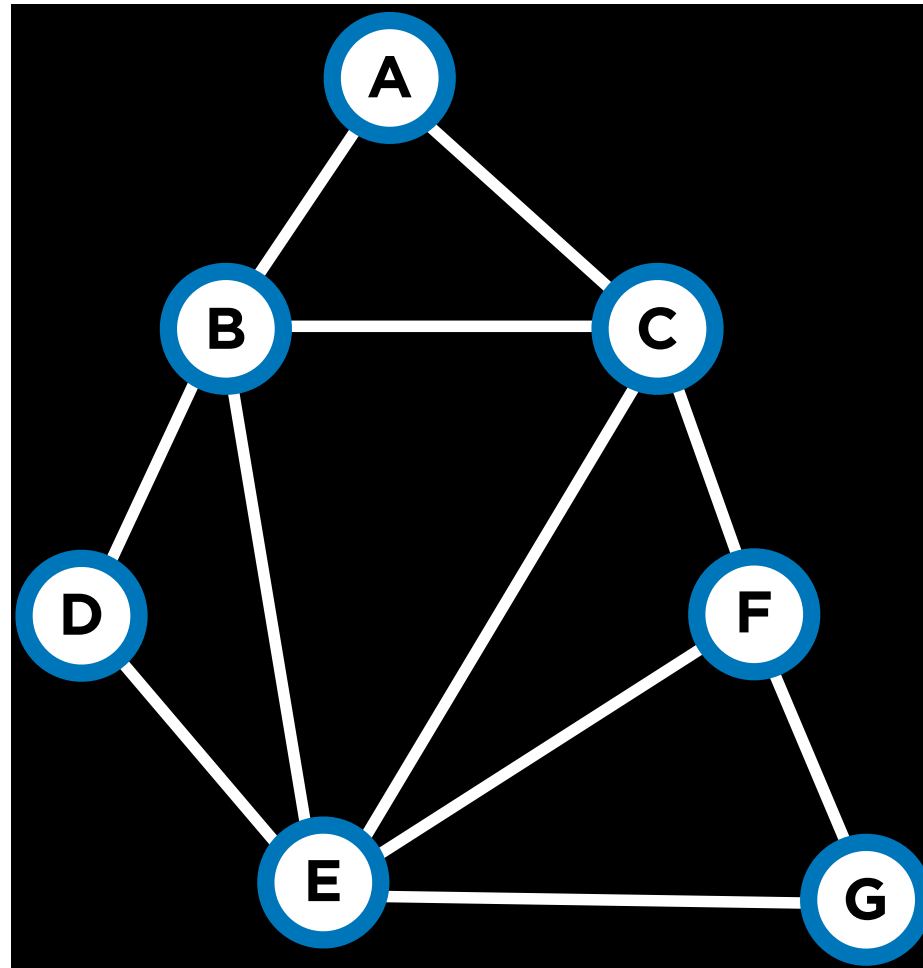
Unary and Binary Constraints

- Unary
  - Constraint involving only one variable
    - {A ≠ Monday}

- Binary
  - Constraint involving two variables
    - {A ≠ B}

Node Consistency

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

$$\{A, B\}$$

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

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

Node Consistency

A   B

A ———————————— B

{Mon, Tue, Wed}   {Mon, Tue, Wed}

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

Node Consistency

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

Node Consistency

Node Consistency

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

Node Consistency

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

Node Consistency

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

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

Node Consistency

- We have easily enforced node consistency

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

- However, different types of consistency can be considered …

Arc Consistency

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

- 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

Arc Consistency

A $\qquad$ B

A $\{Tue, Wed\}$ B $\{Wed\}$

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

Arc Consistency

A   B

$$\{Tue, Wed\} \qquad \{Wed\}$$

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

Arc Consistency


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

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
```
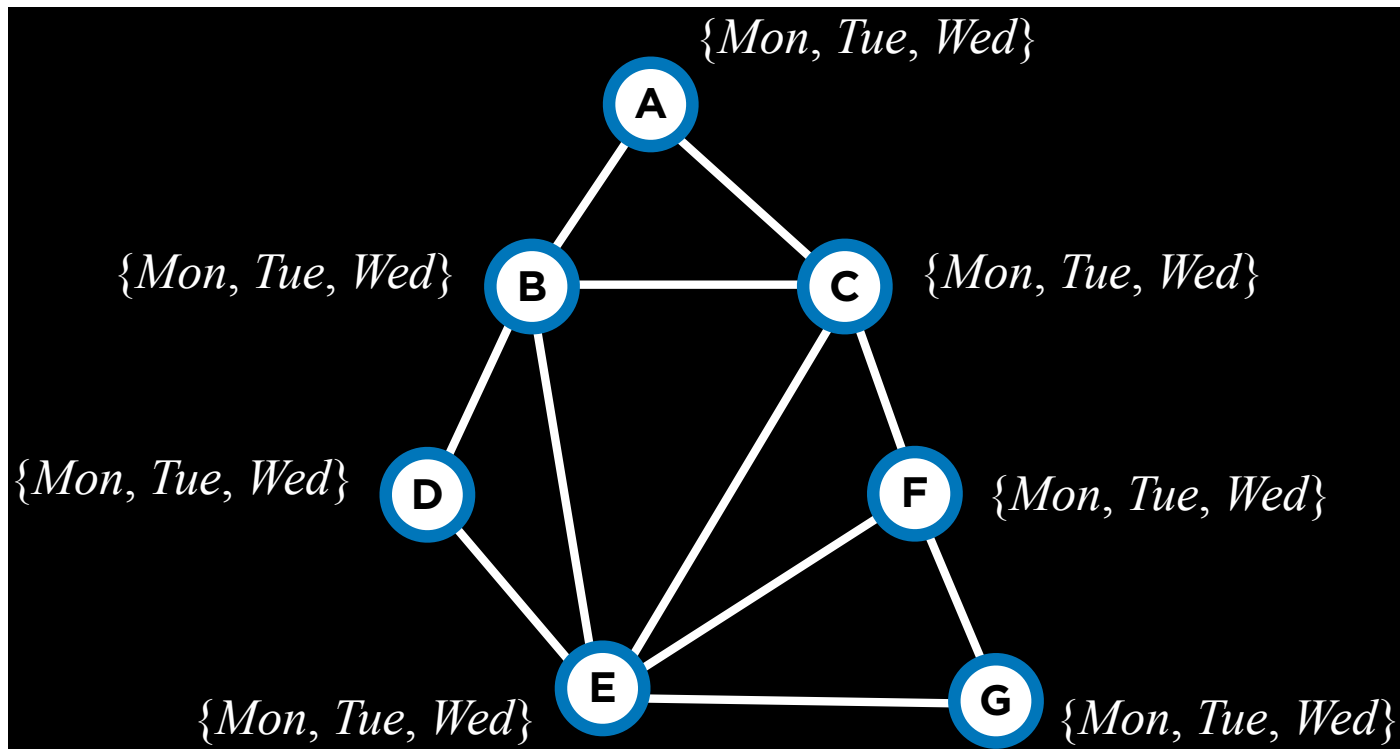
AC-3 Algorithm for Arc Consistency

```
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
```

Arc Consistency on a Graph
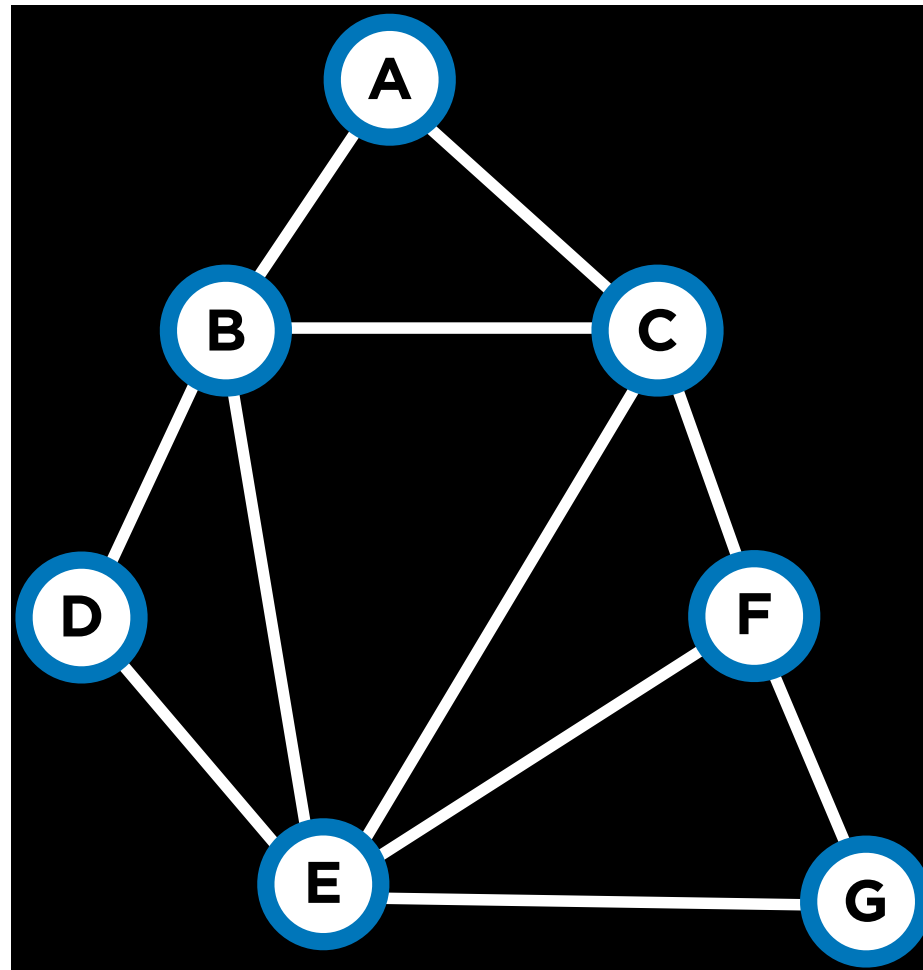
Arc Consistency on a Graph

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 the same cost

Backtracking Search

- The search algorithm generally used for CSPs is Backtracking Search

- Idea
  - Go ahead and make assignments from variables to values
    - If we arrive at a place where there is no way we can make any forward progress, while still preserving the constraints that we need to enforce, we'll go ahead and backtrack and try something else instead

Backtracking Search

```
function BACKTRACK(assignment, csp):
    if assignment complete: return assignment
    var = SELECT-UNASSIGNED-VAR(assignment, csp)
    for value in DOMAIN-VALUES(var, assignment):
        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
```

Backtracking in Practice
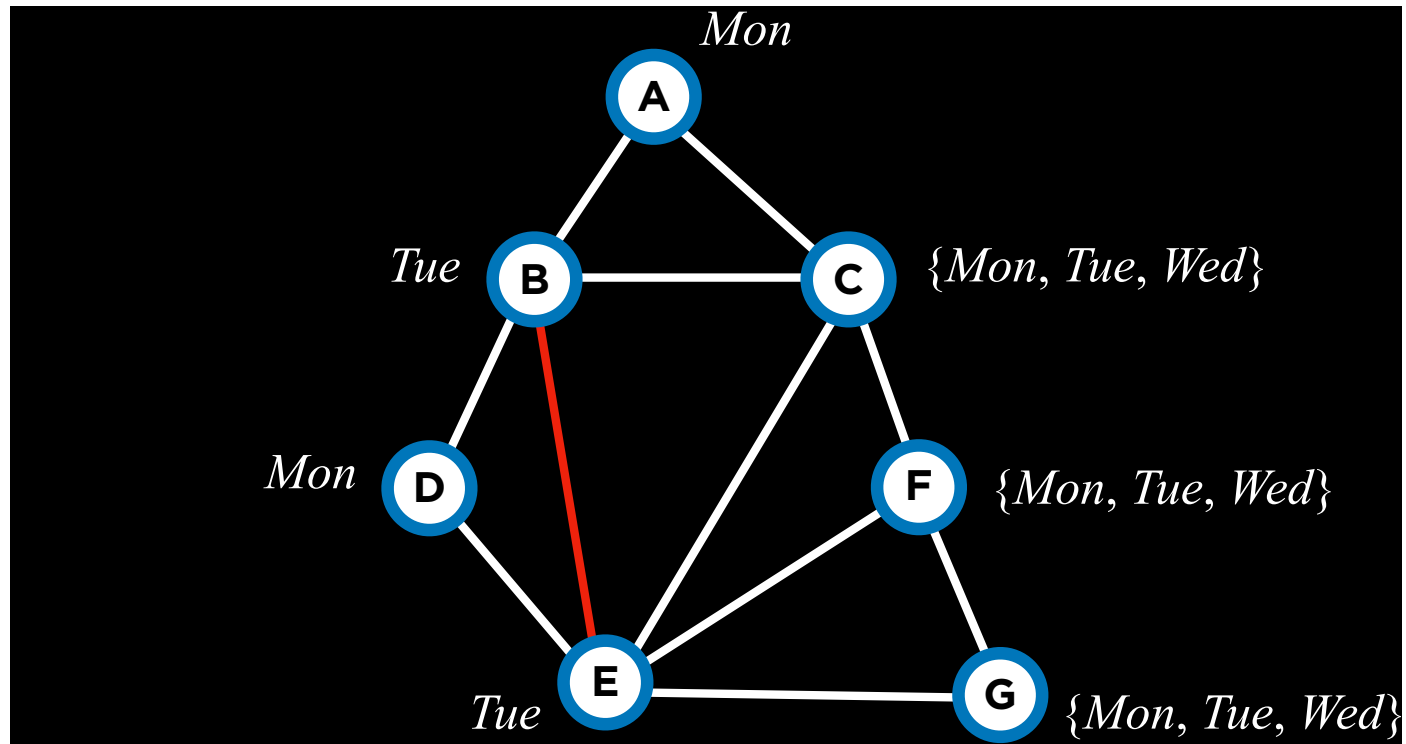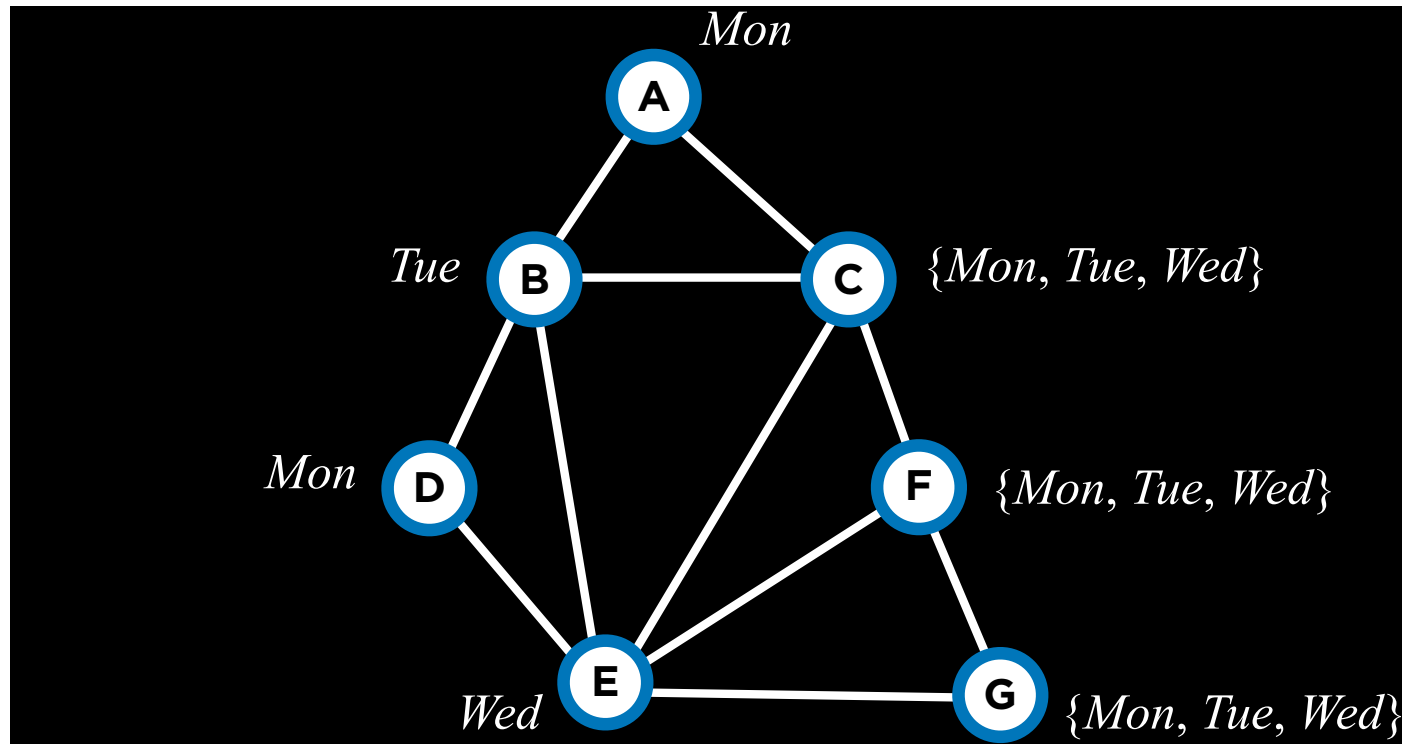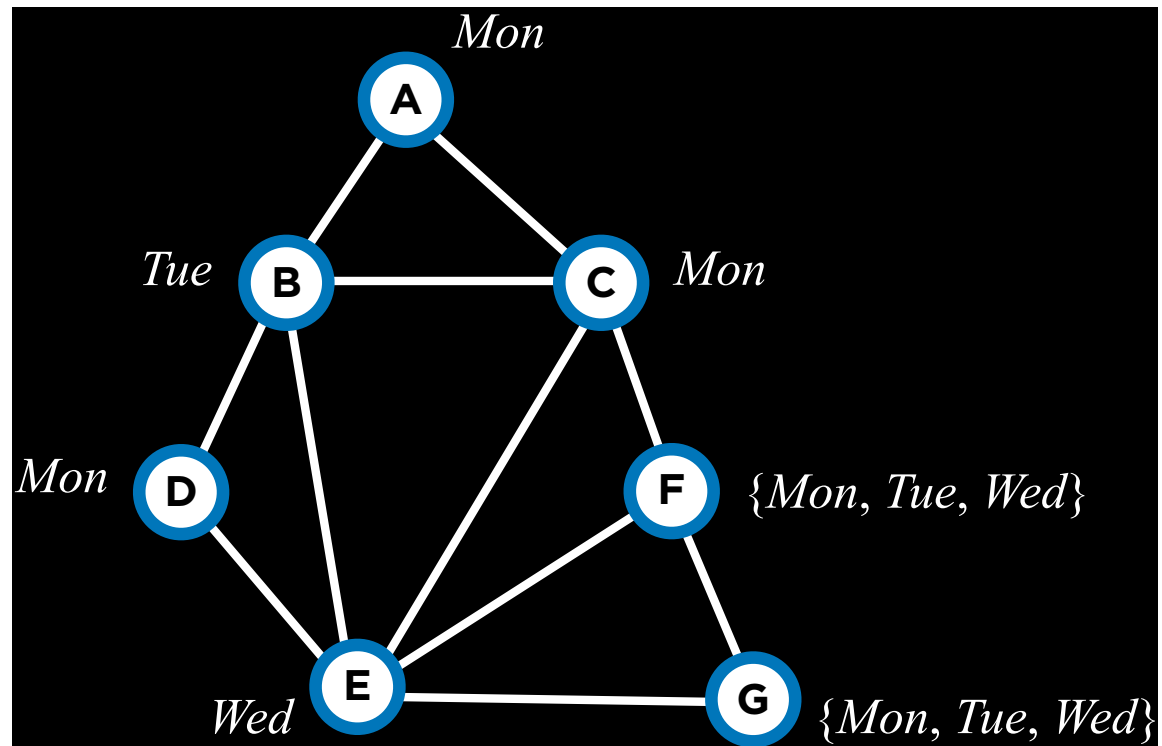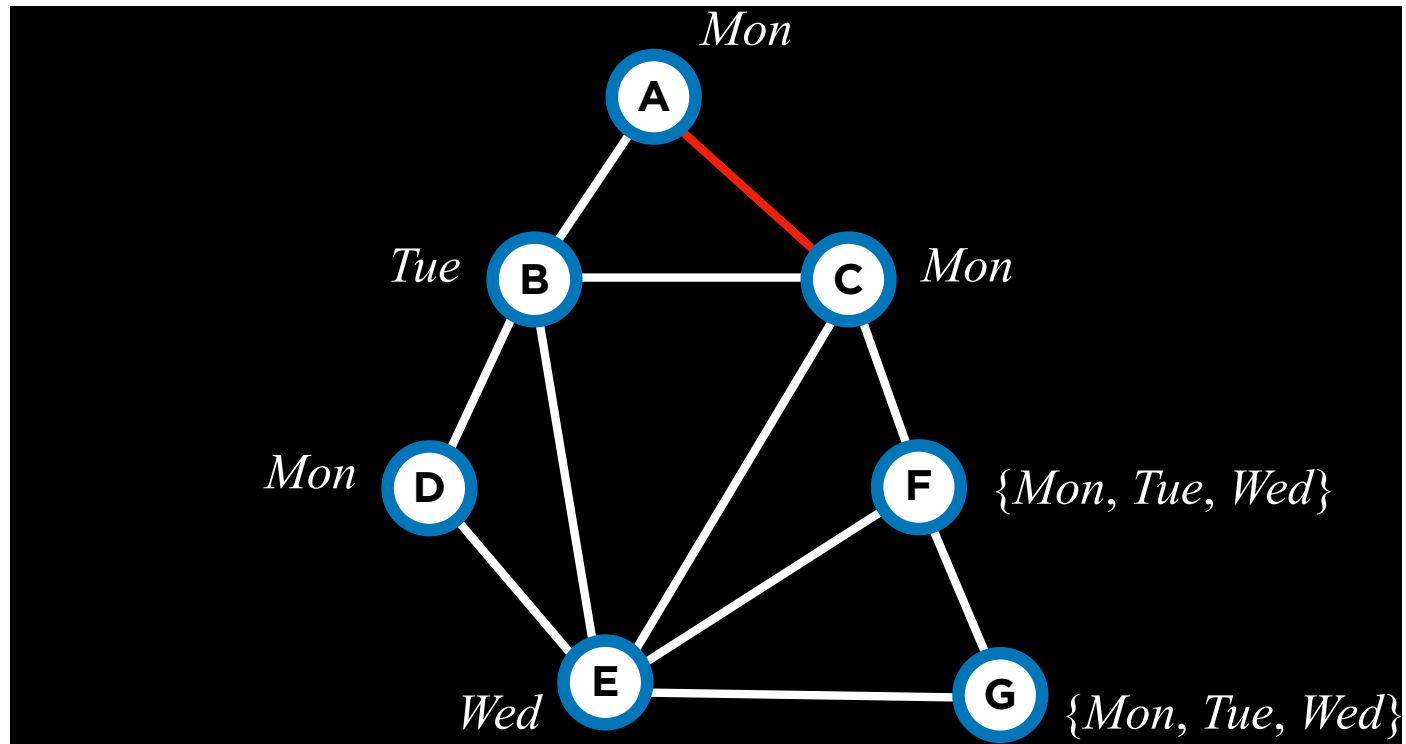
Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice
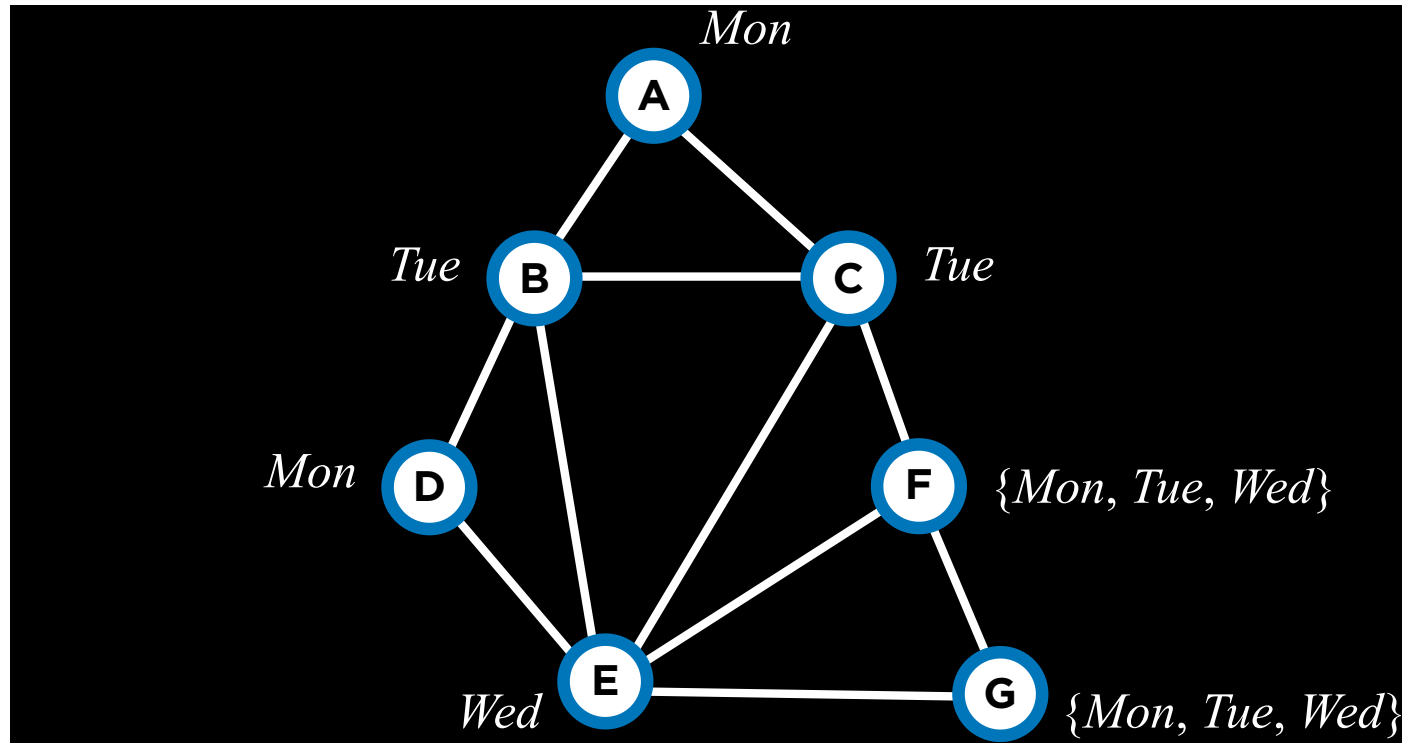
Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice
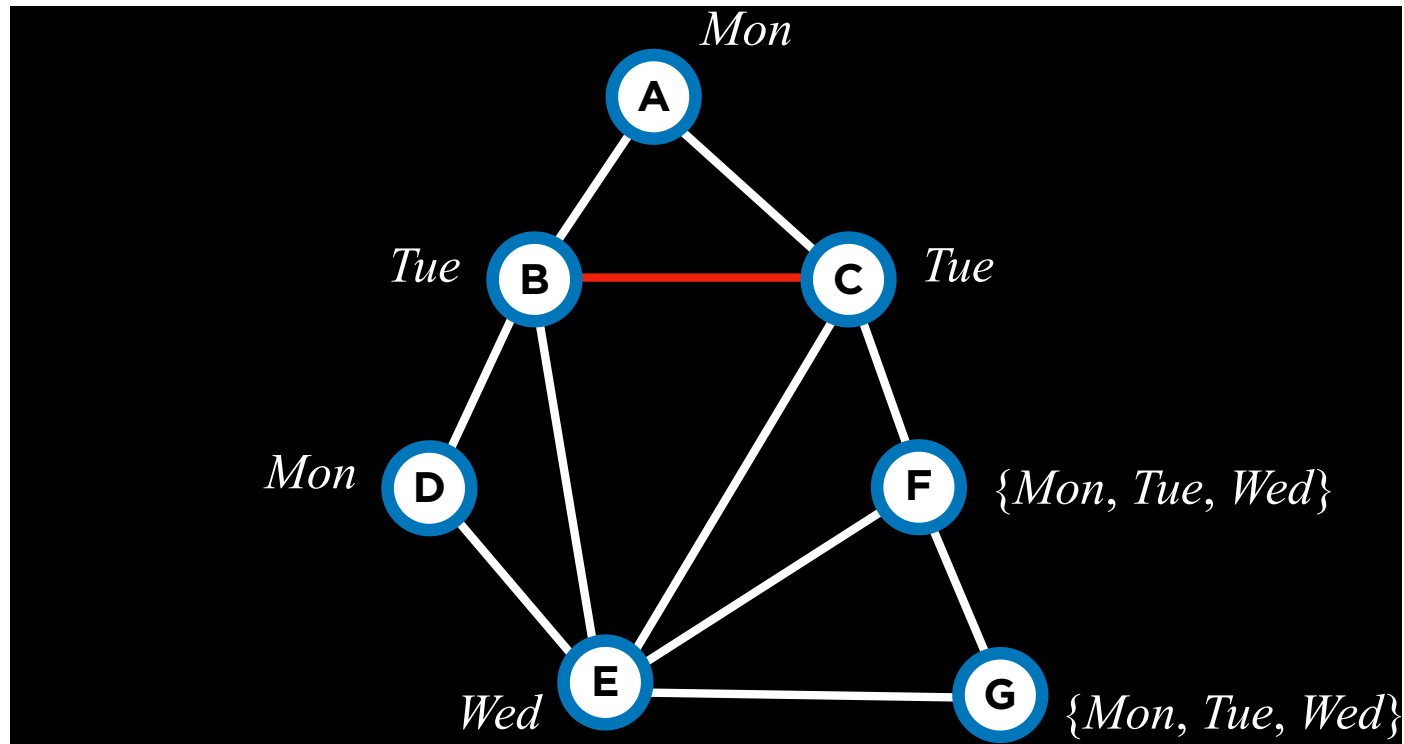
Backtracking in Practice

Backtracking in Practice

Backtracking in Practice
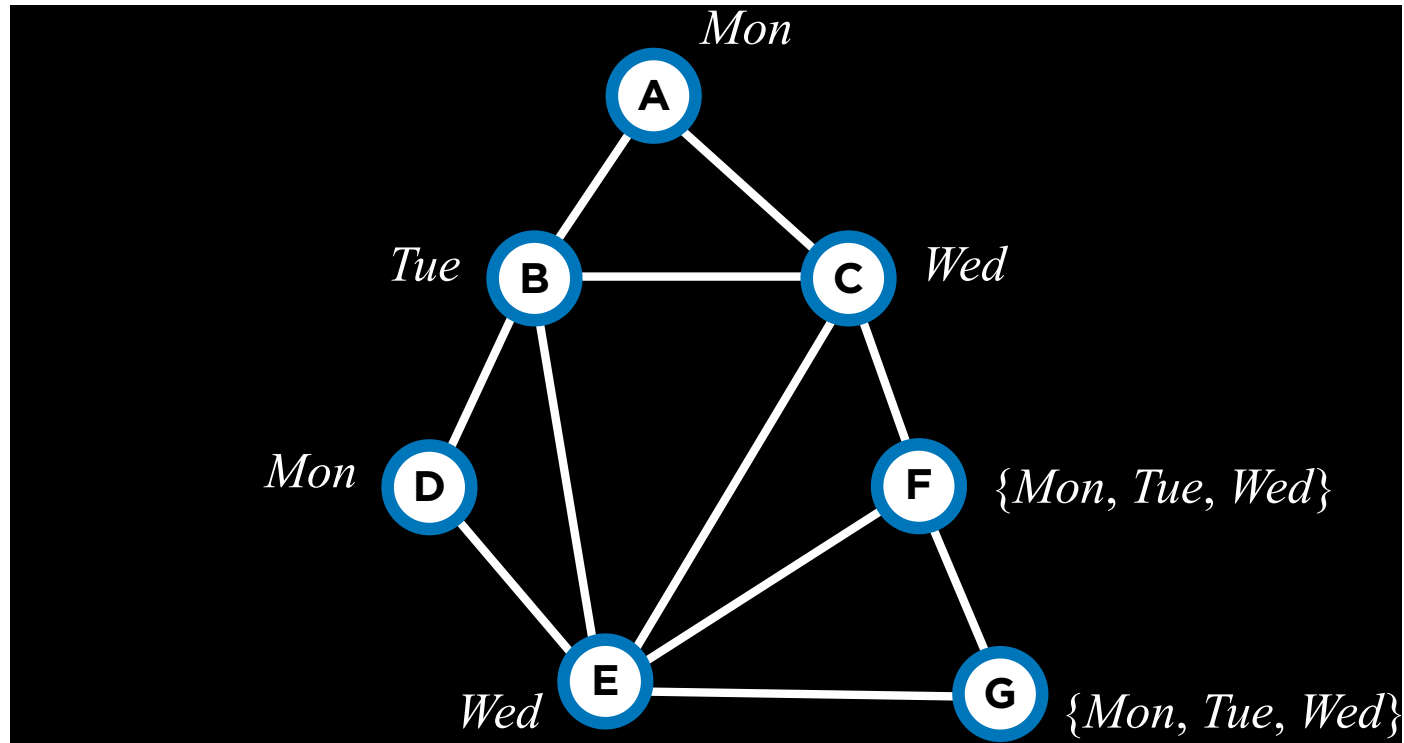
Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice
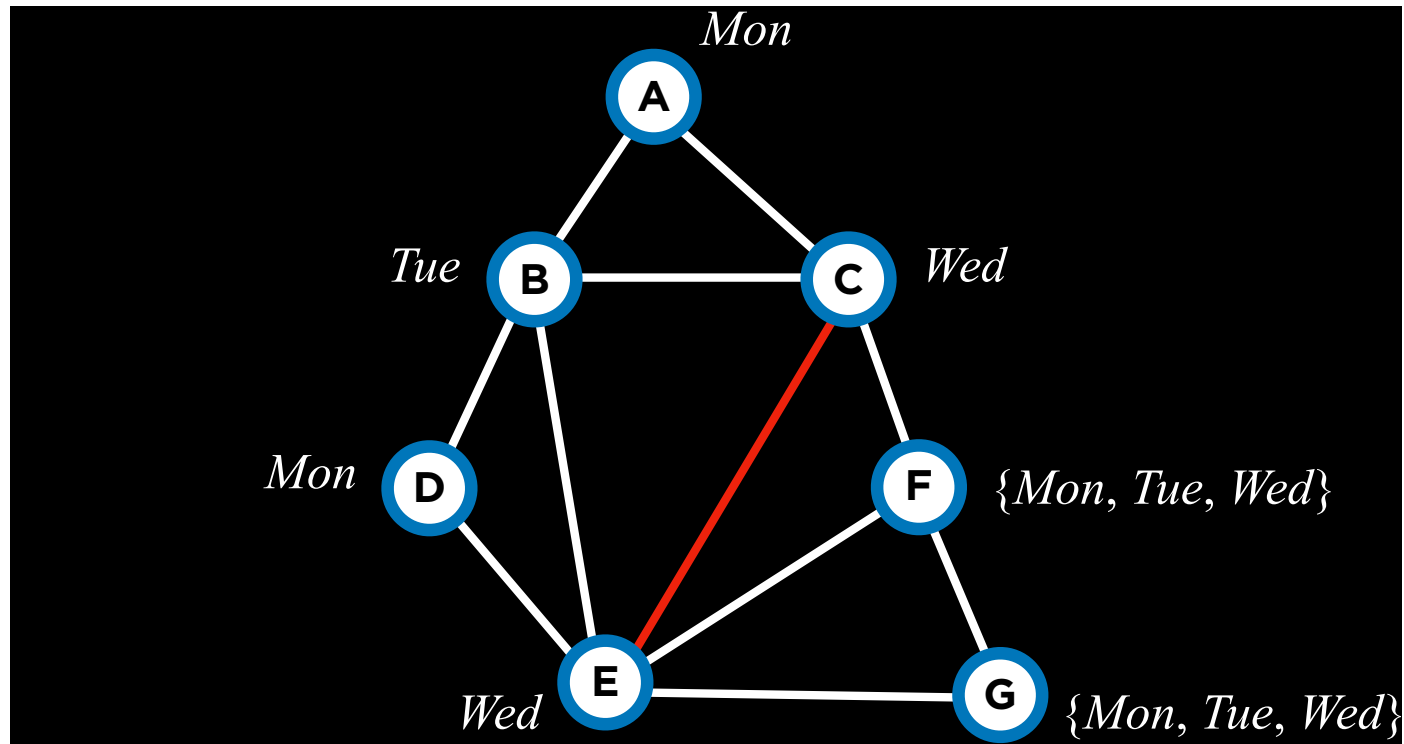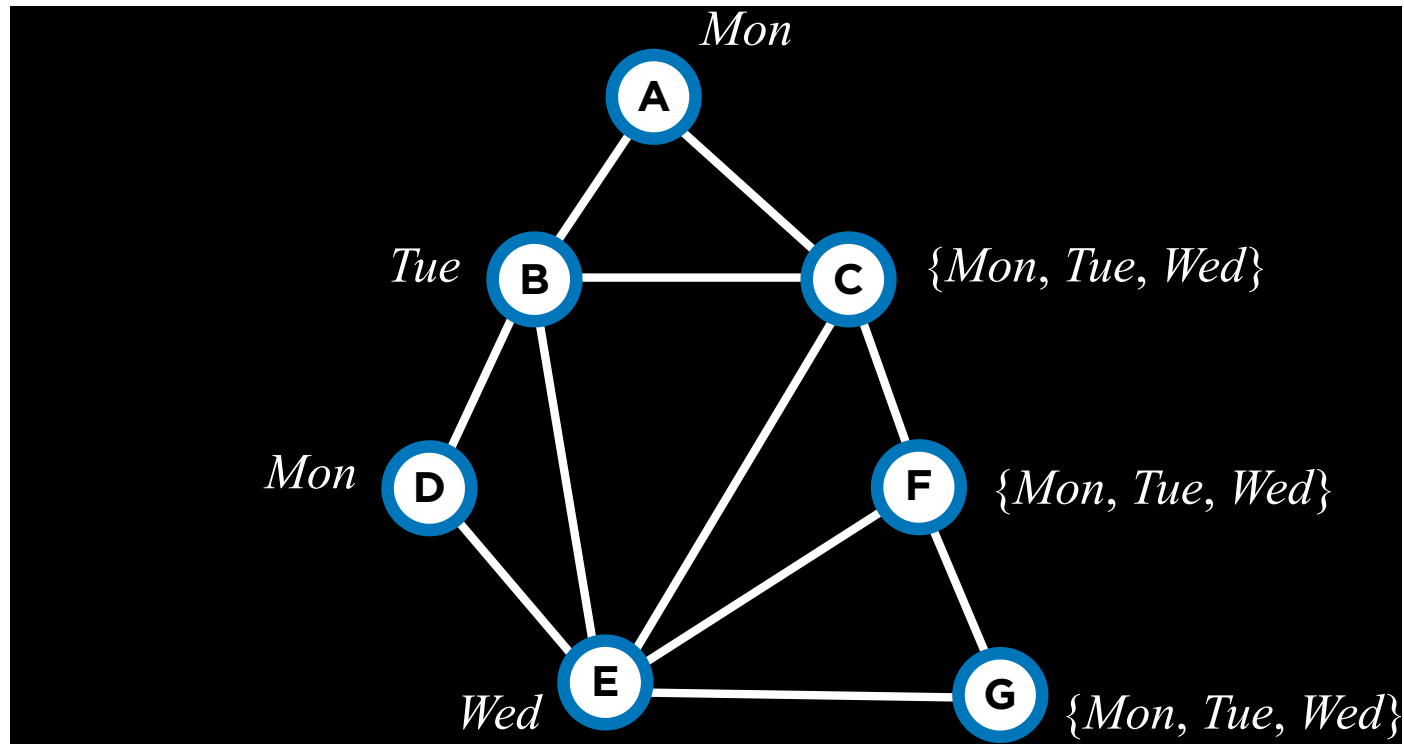
Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice
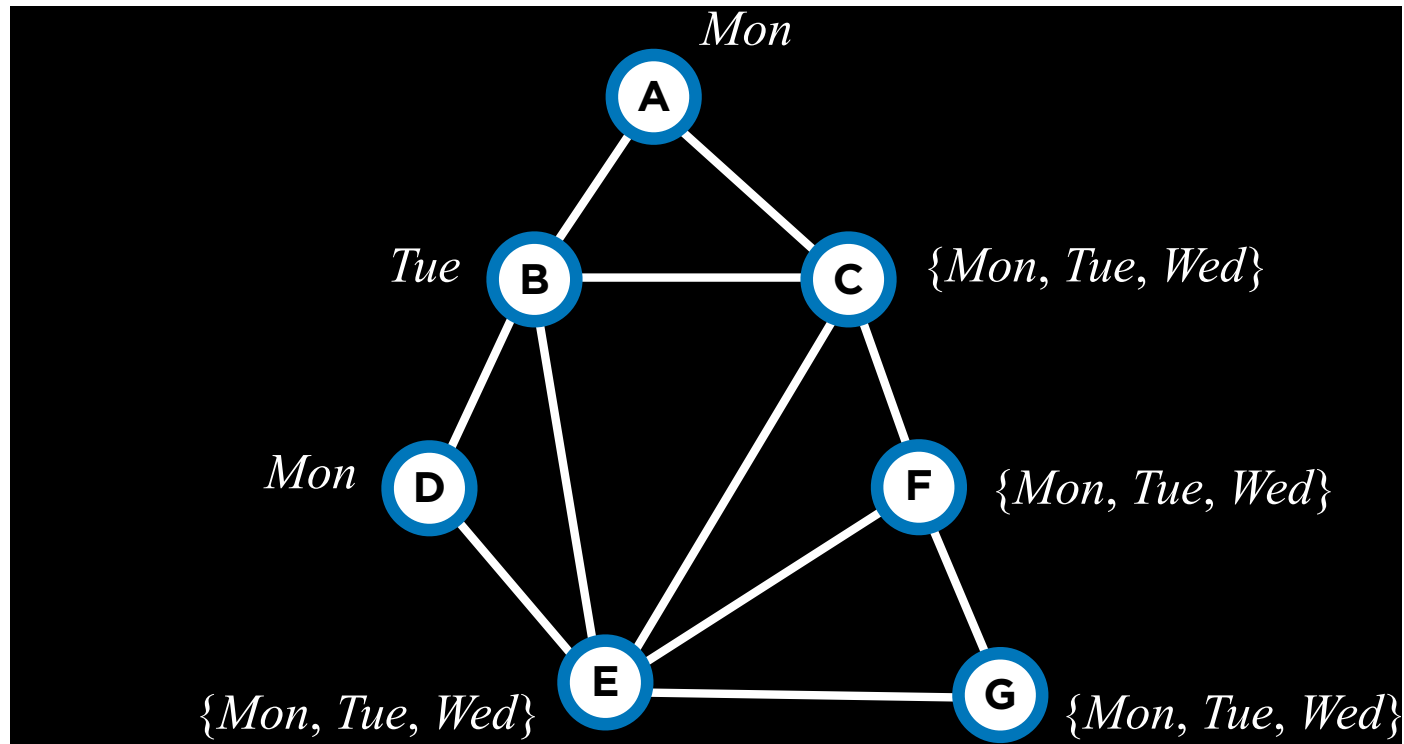
Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice
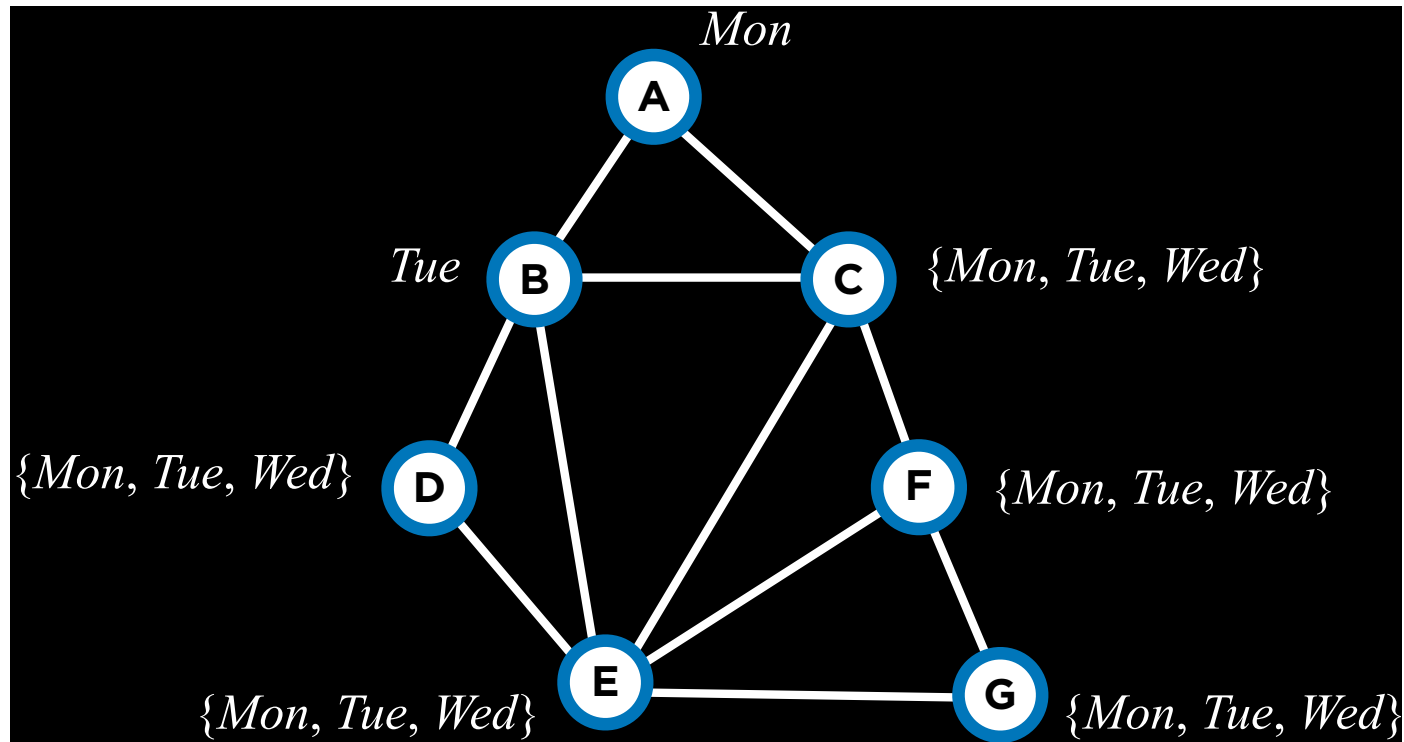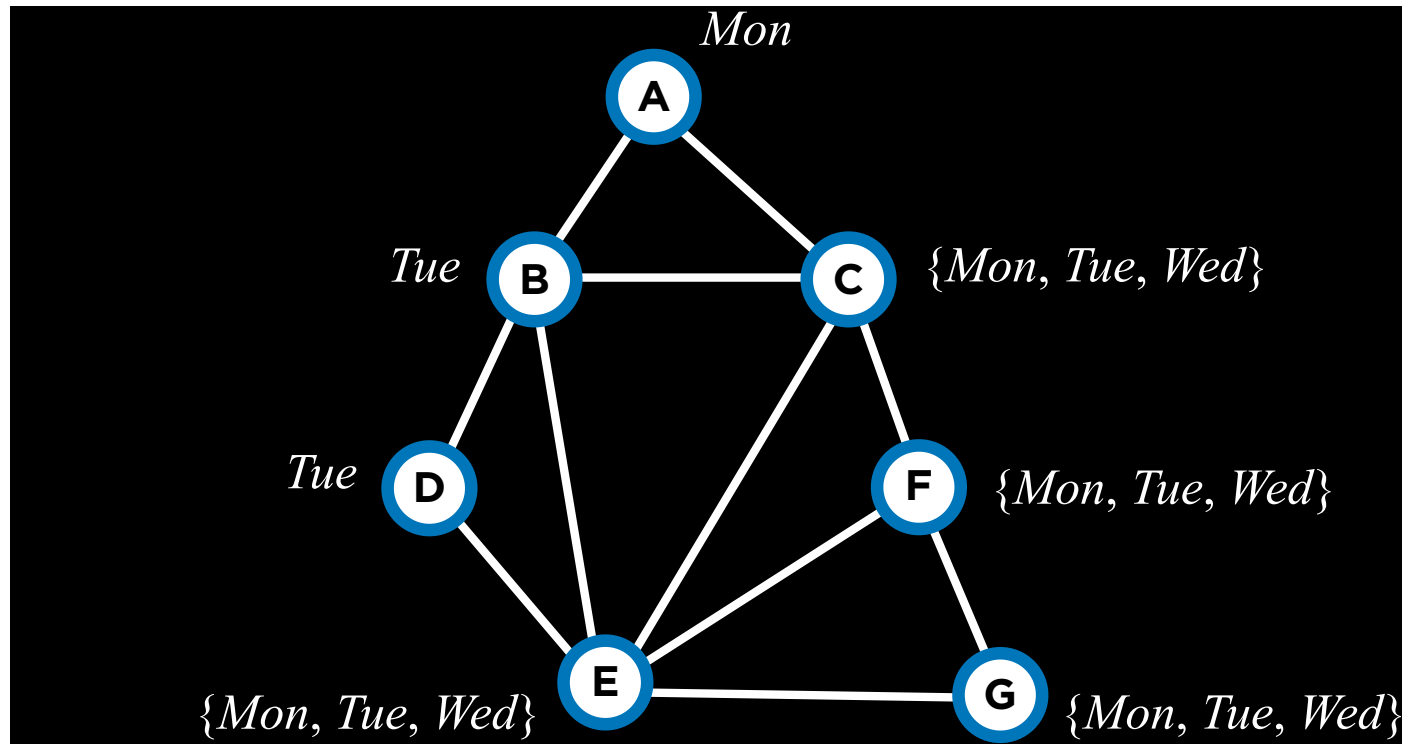
Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice

Backtracking in Practice
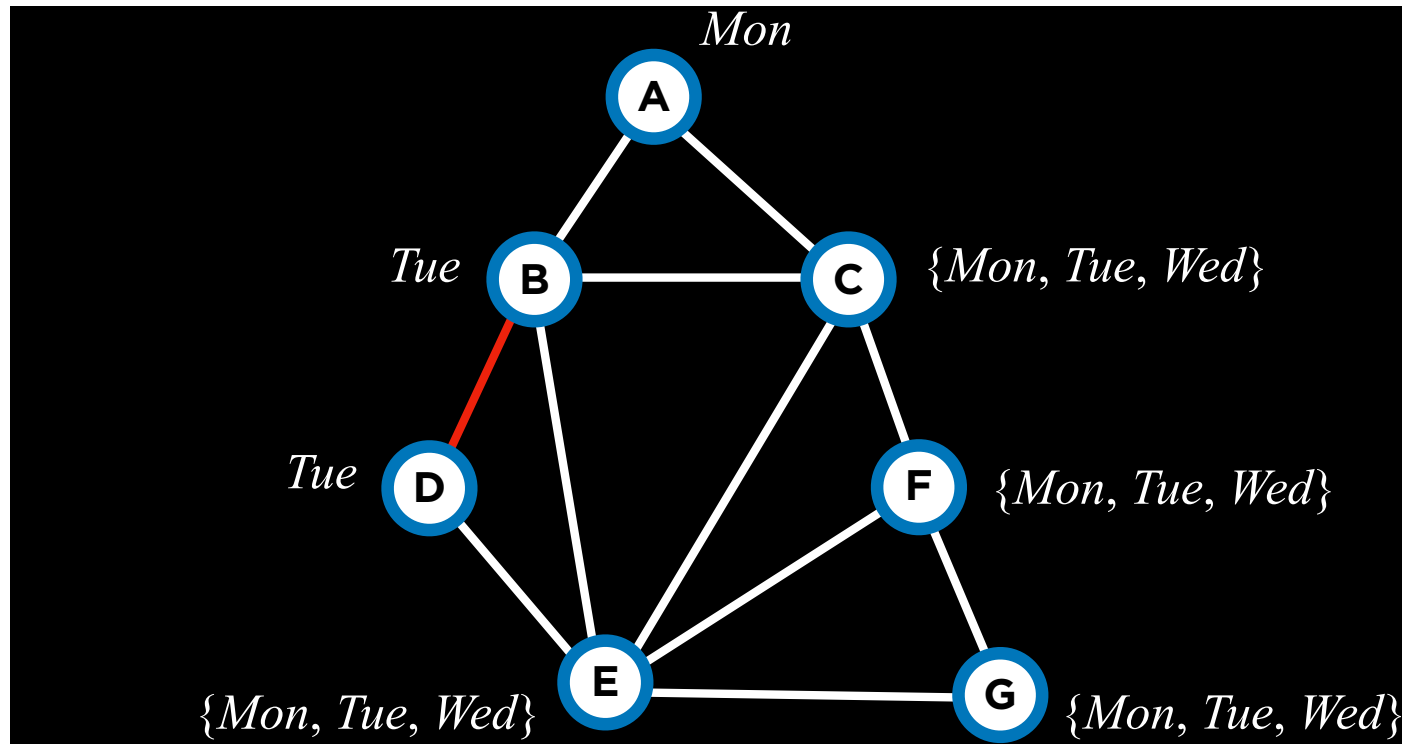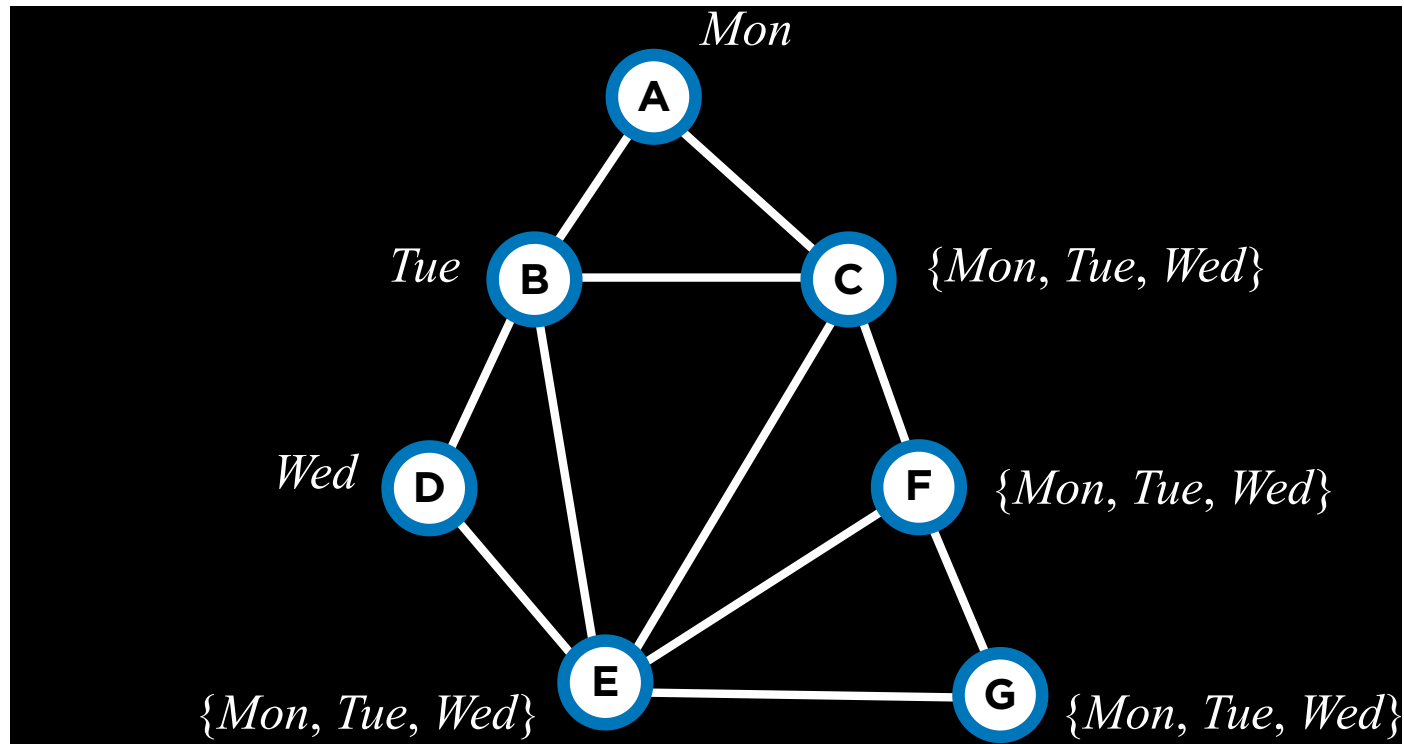
Backtracking in Practice

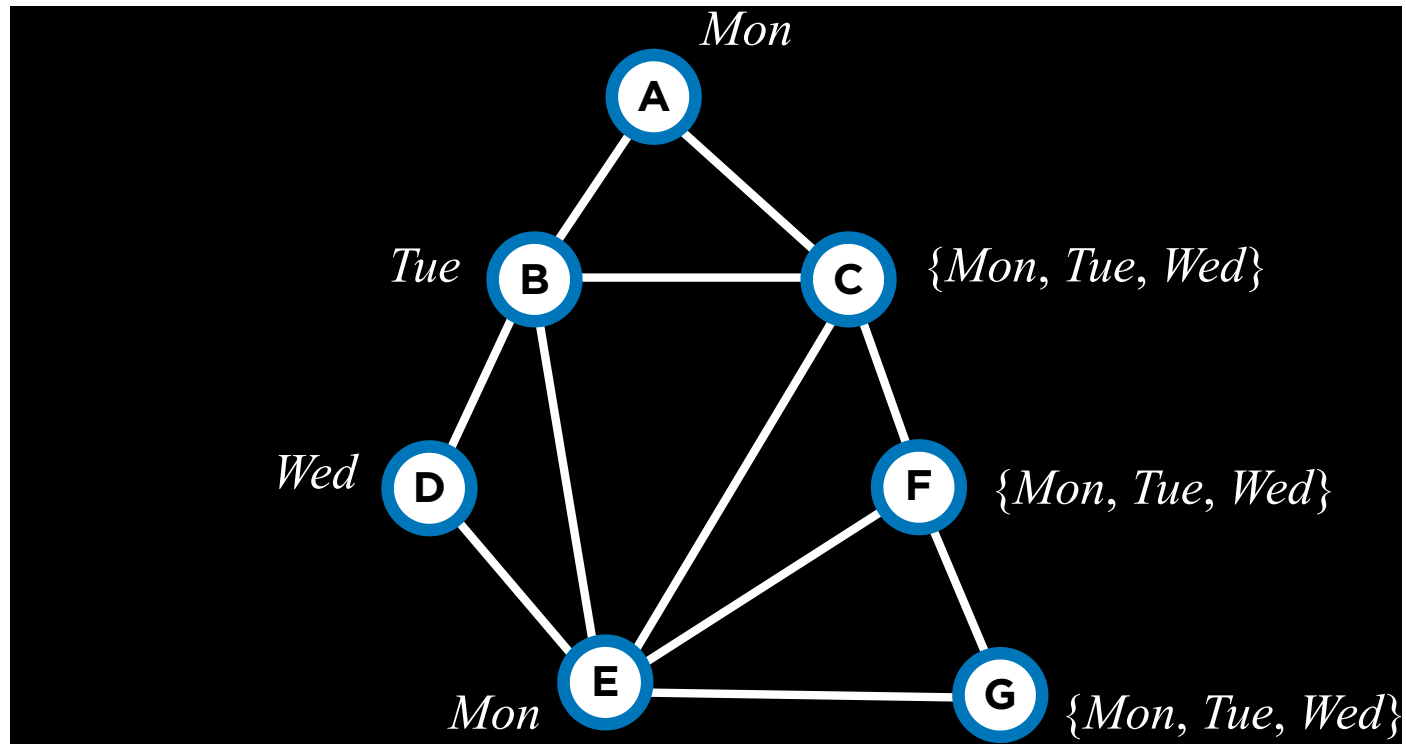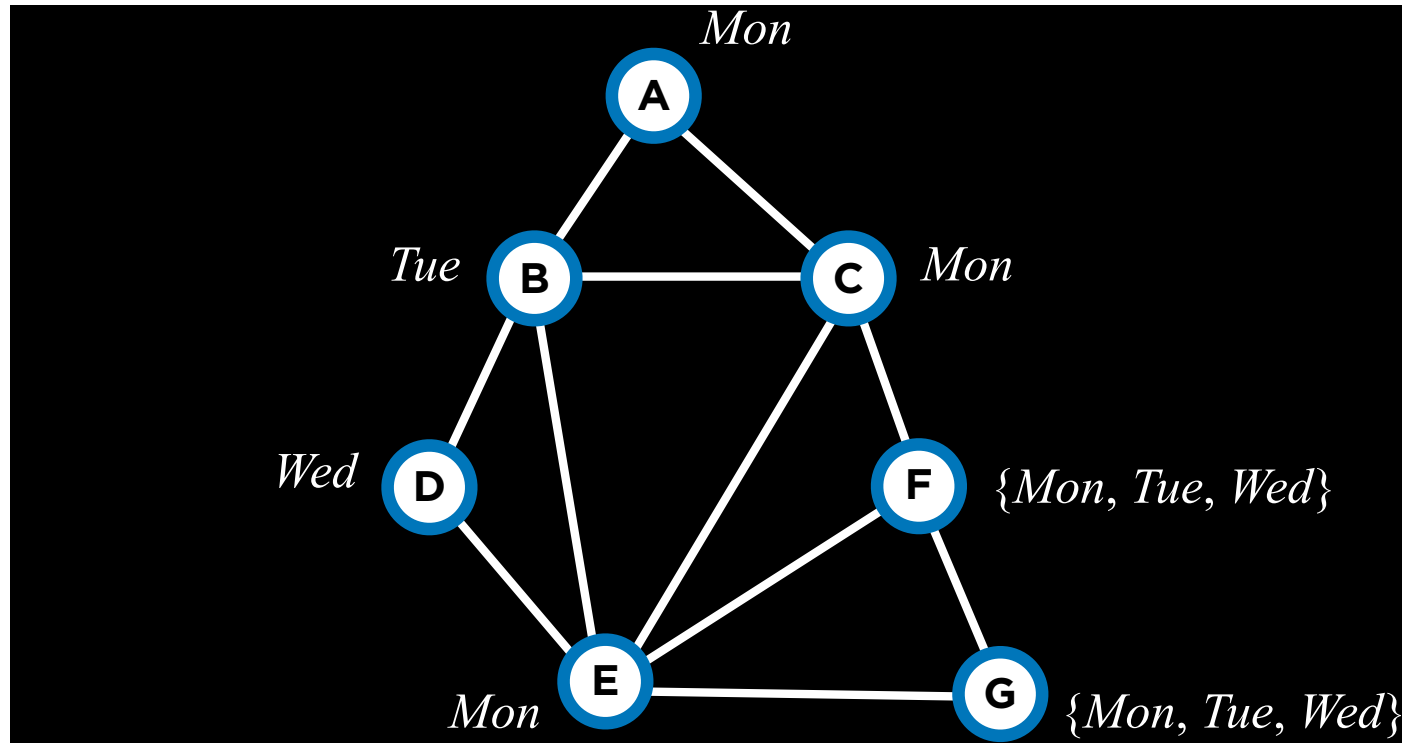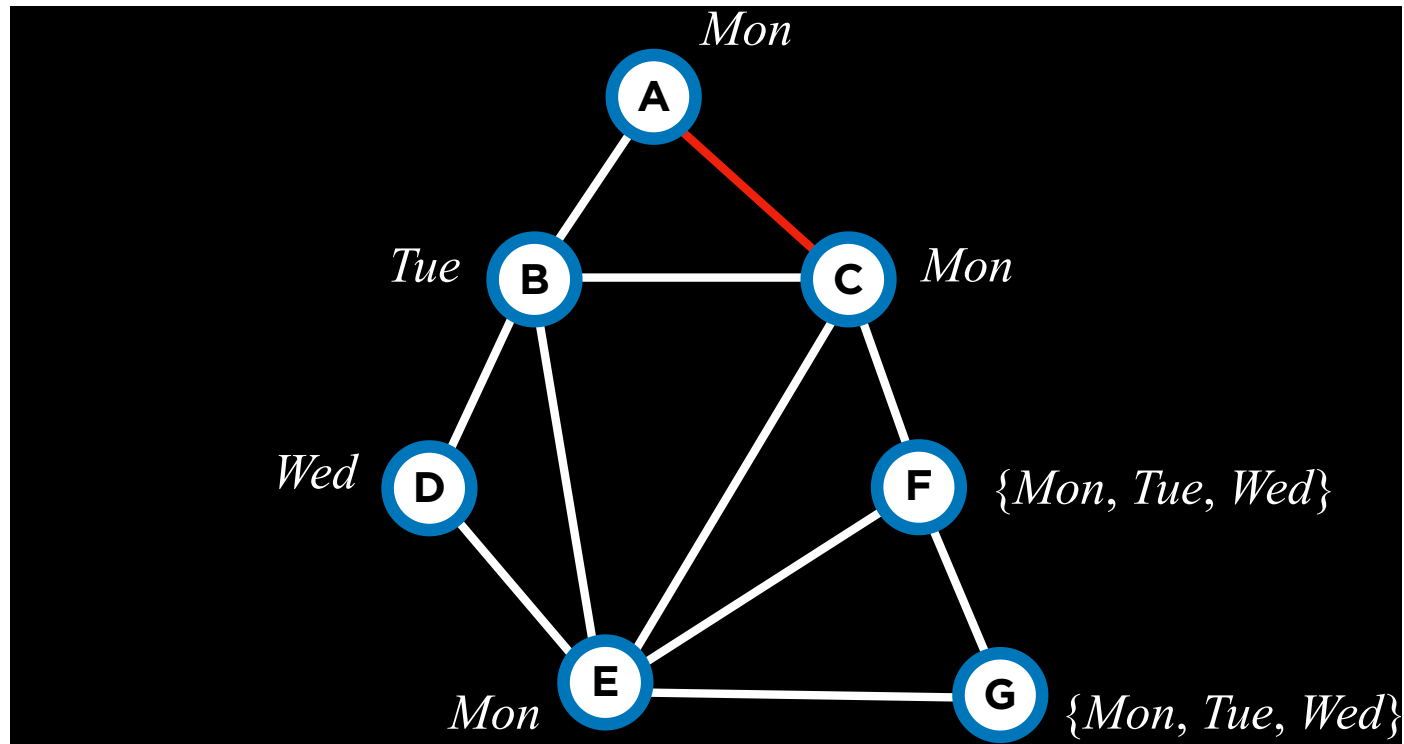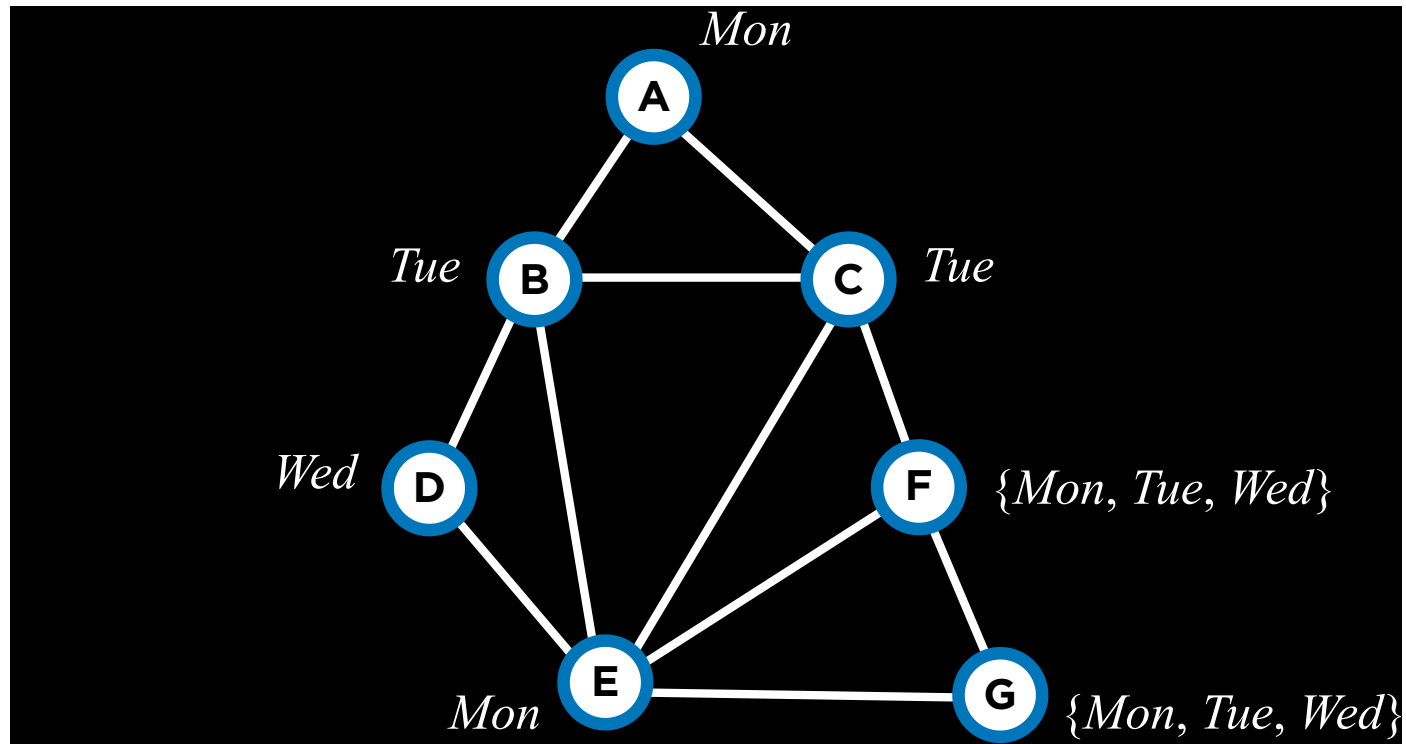# Backtracking in Practice

Inference

- We might be clever in order to improve the efficiency of how we solve these sorts of problems

- The idea is that of inference, using our problem knowledge to draw conclusions in order to make the rest of the problem-solving process easier

- Let's go back to where we got stuck the first time
  - We dealt with B and then we went on to D
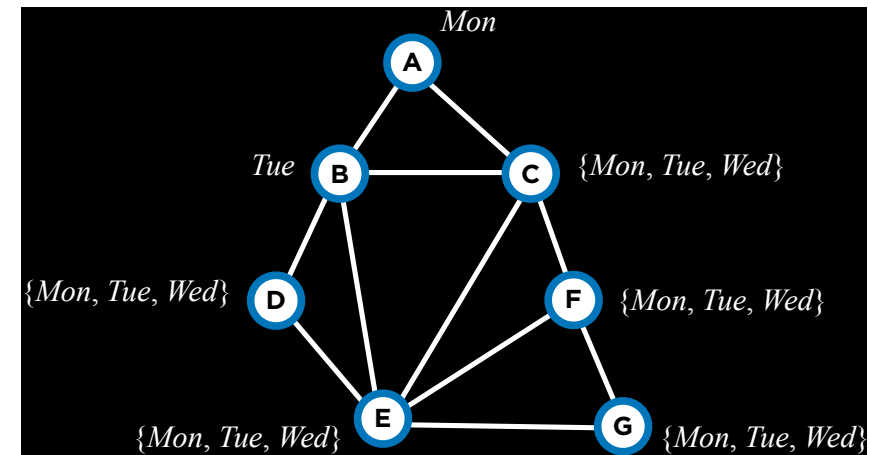
Inference
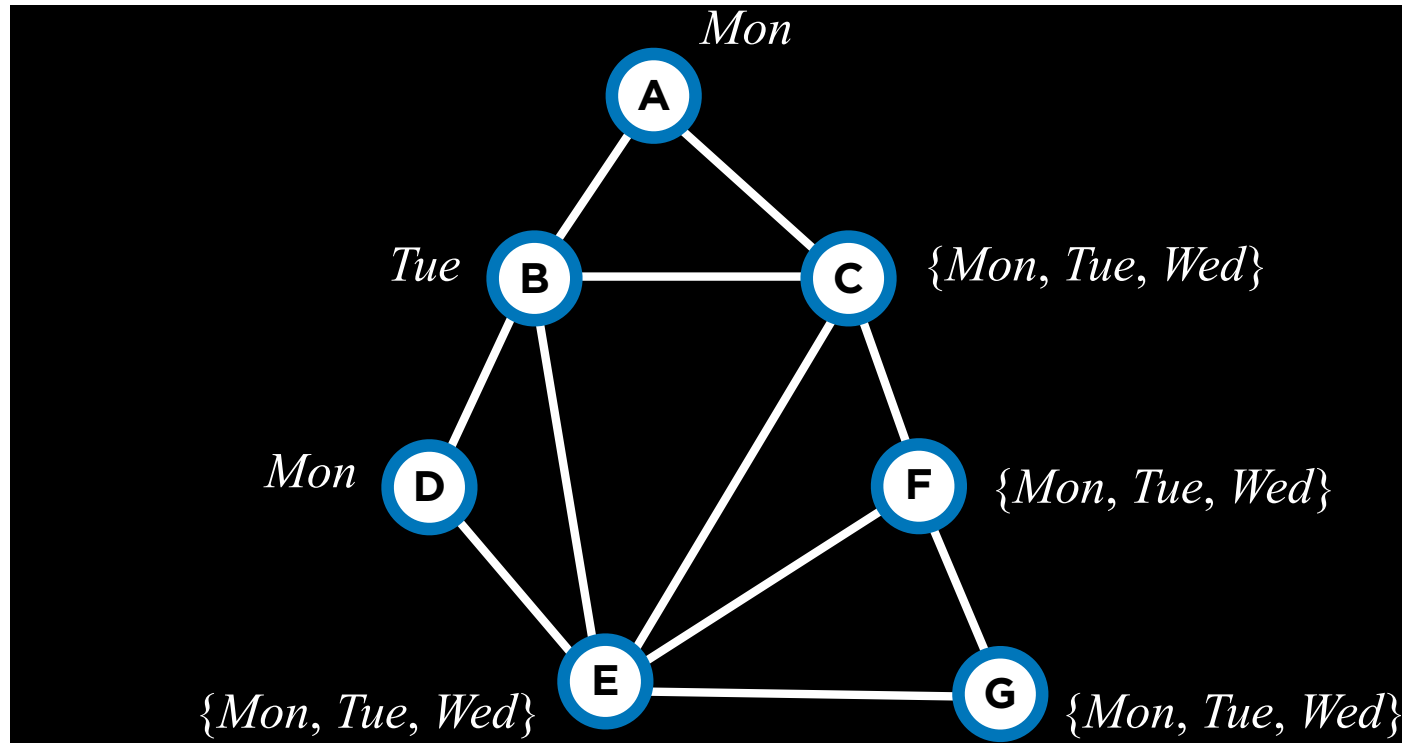
Inference
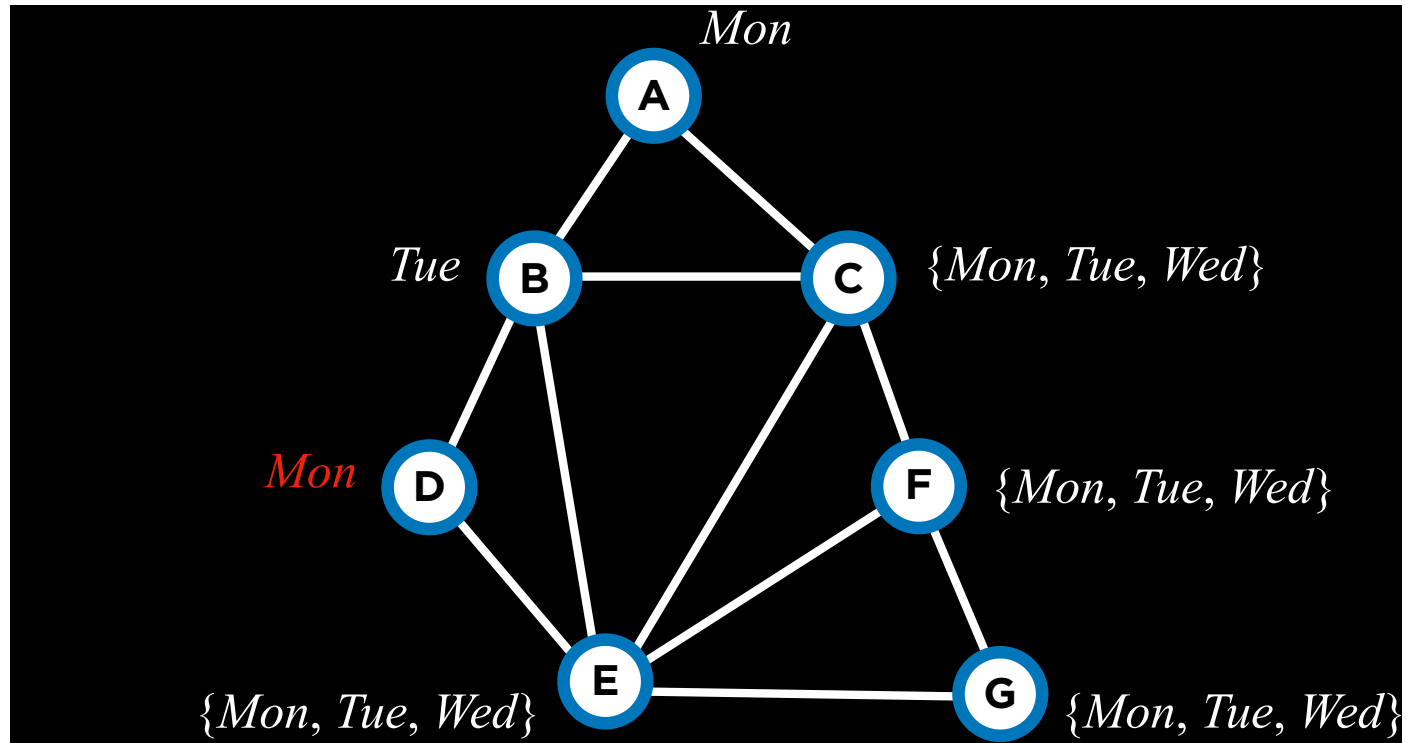
Inference

- We can look at the structure of this graph
  - For example, C's domain contains Monday and Tuesday making it not arc-consistent with A and B
  - Using that information by making C arc-consistent with A and B, we could remove Mon and Tue from C's domain and just leave C with Wed …
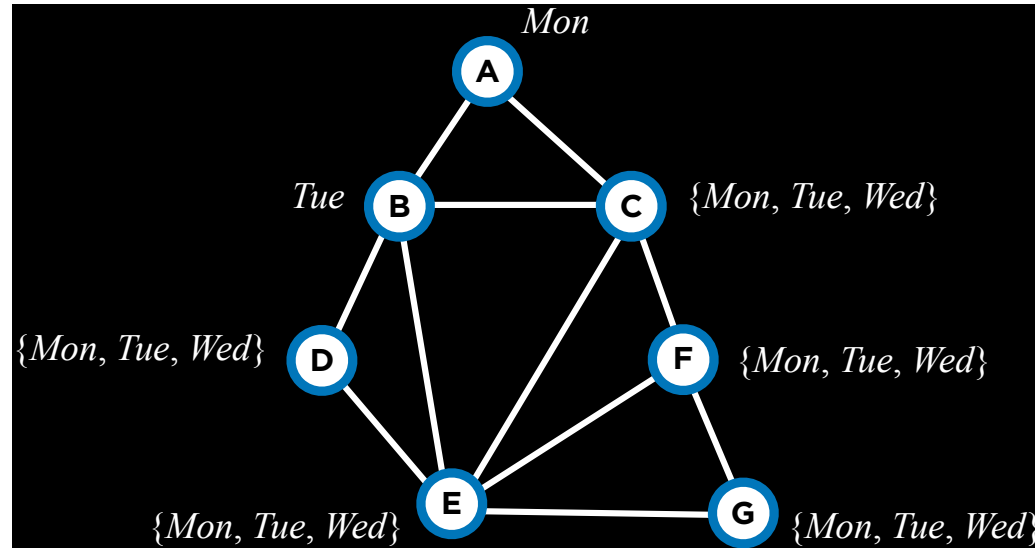
Inference

- Continuing to try and enforce arc consistency, there are some other conclusions we can draw
    - B's only option is Tue and C's only option is Wed
    - if we want to make E arc-consistent, E can't be Tue and Wed because that wouldn't be arc-consistent with B and C…
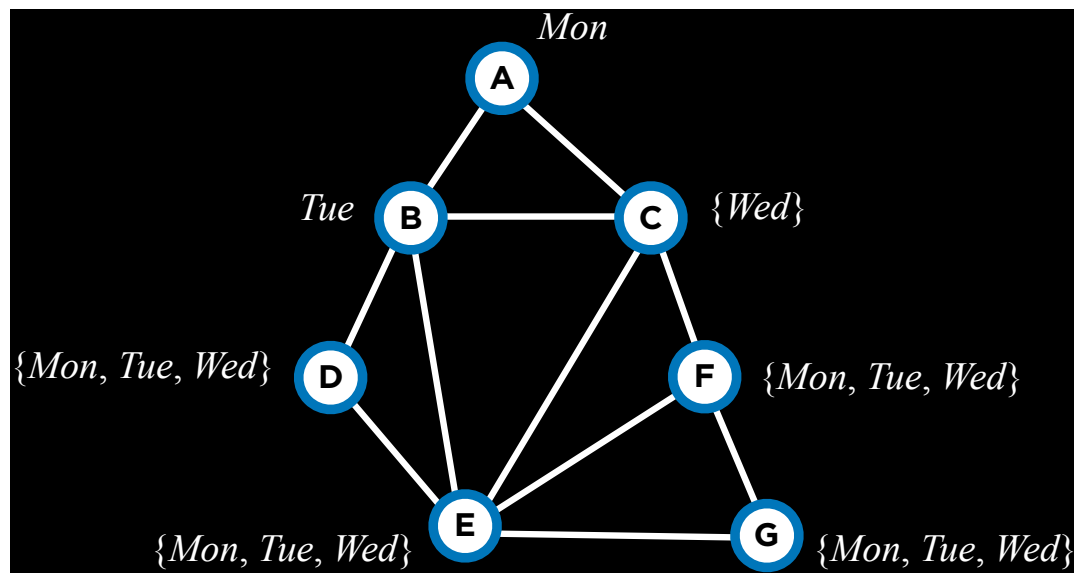
Inference
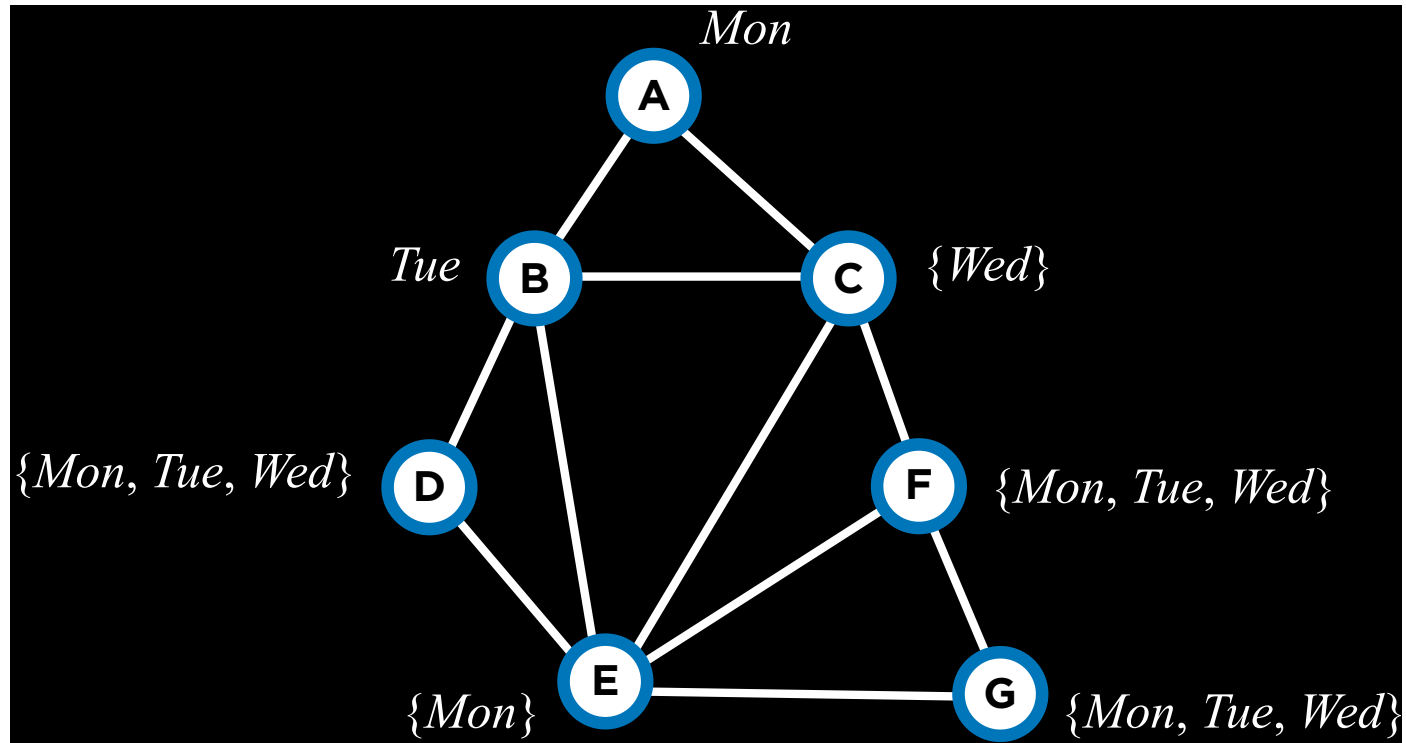
Inference
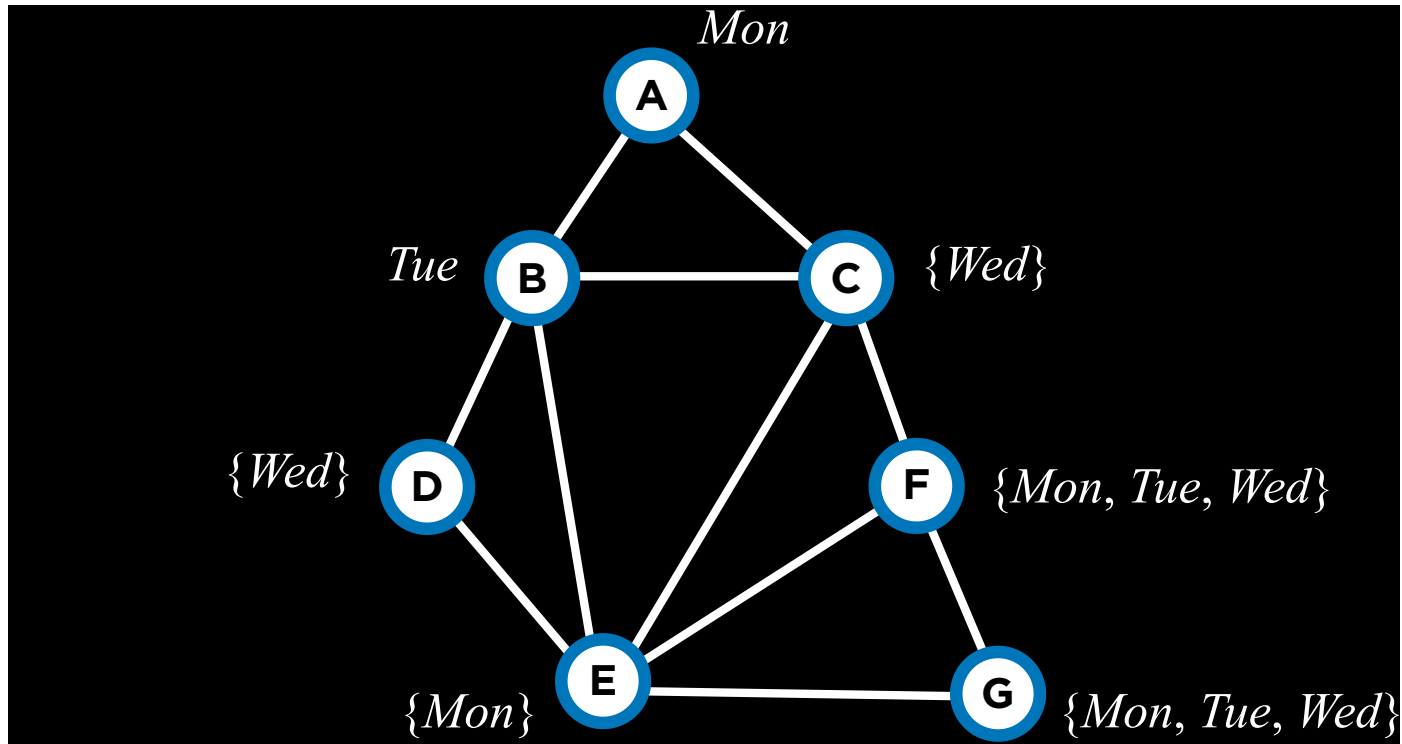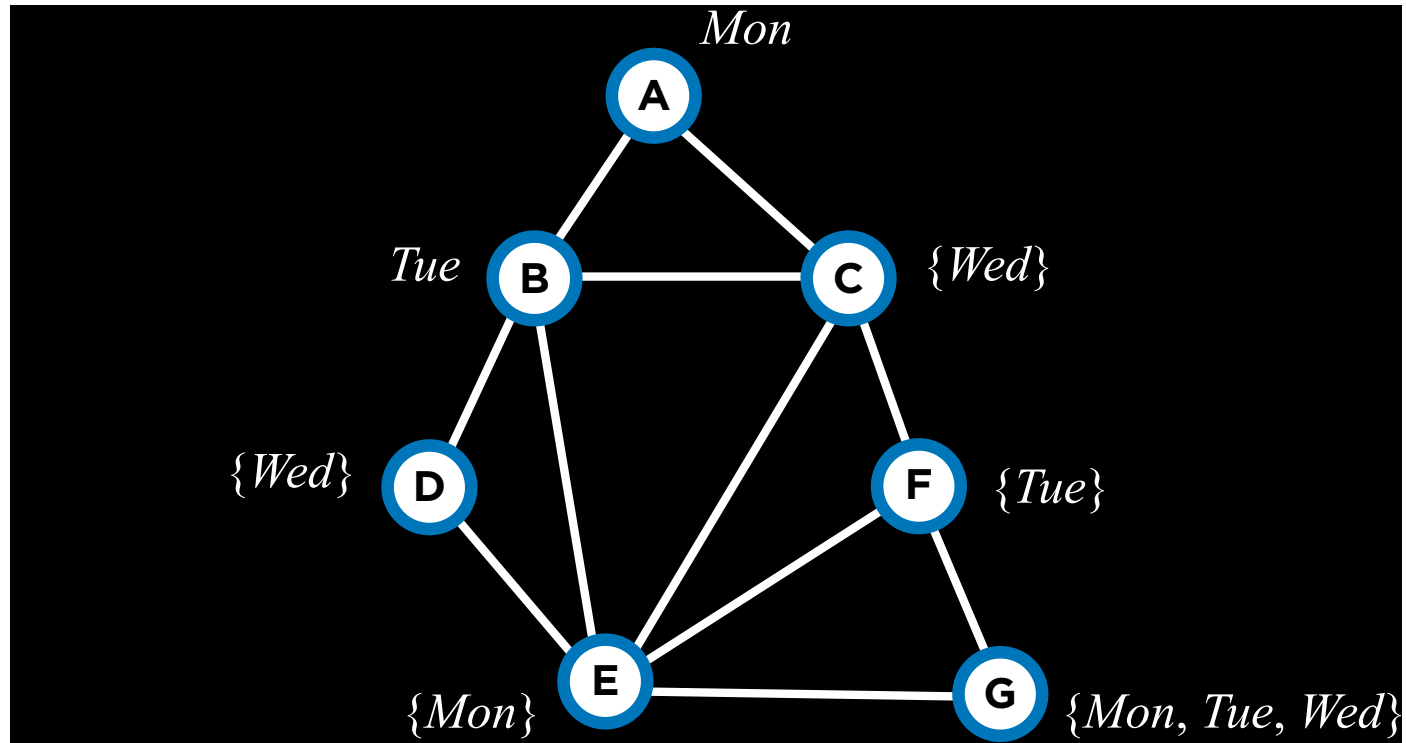
Inference
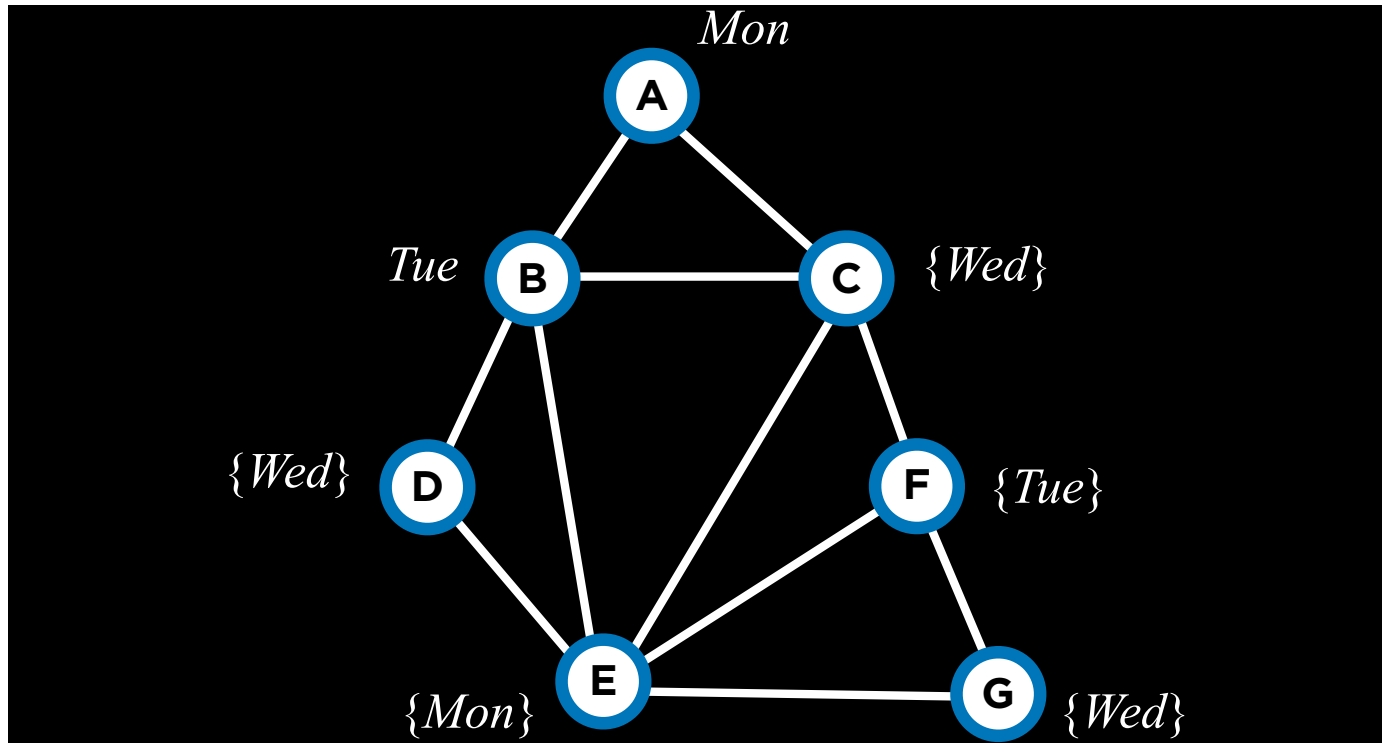
Inference

Inference

- It turns out that without having to do any additional search and backtrack, just by enforcing arc consistency, we were able to figure out what the assignment of all the variables should be
  - We interleave the search process and the inference step in trying to enforce arc consistency
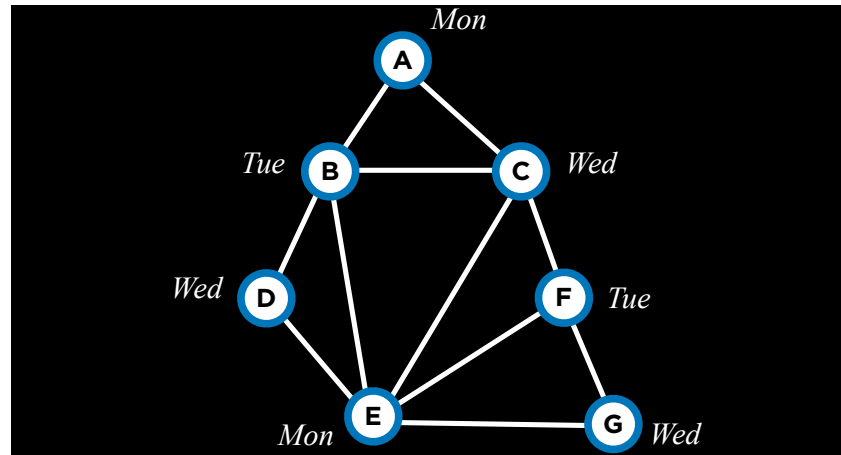
Maintaining Arc-Consistency

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

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

- Sometimes we can run the algorithm at very beginning before we even begin searching to limit the domain of the variables so making it easier to search

Maintaining Arc-Consistency

```
function BACKTRACK(assignment, csp):
    if assignment complete: return assignment
    var = SELECT-UNASSIGNED-VAR(assignment, csp)
    for value in DOMAIN-VALUES(var, assignment):
        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
```

Heuristics

- There are other heuristics that can be used to try to improve the efficiency of the search process
  - it concerns some of the functions employed in the revised backtracking algorithm
- To begin with, let's consider `SELECT-UNASSIGNED-VAR`
  - It selects some variable in the CSP that has not yet been assigned
  - So far, we have been selecting variables at random, but we can do better by using certain heuristics for choosing carefully which variable should be explored next

Using Heuristics Revisited

```
function BACKTRACK(assignment, csp):
    if assignment complete: return assignment
    var = SELECT-UNASSIGNED-VAR(assignment, csp)
    for value in DOMAIN-VALUES(var, assignment):
        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
  - The idea is if there are only two remaining values left, we can discard one of them very quickly to get to the other
    - one of those two has got to be the solution if a solution does exist

- Degree heuristic
  - Select the variable that has the highest degree
  - The idea is that by choosing a variable of high degree, one immediately constraints the rest of the variables more
    - and it's more likely to be able to eliminate large parts of the state-space that we don't need to search trough

Minimum Remaining Values
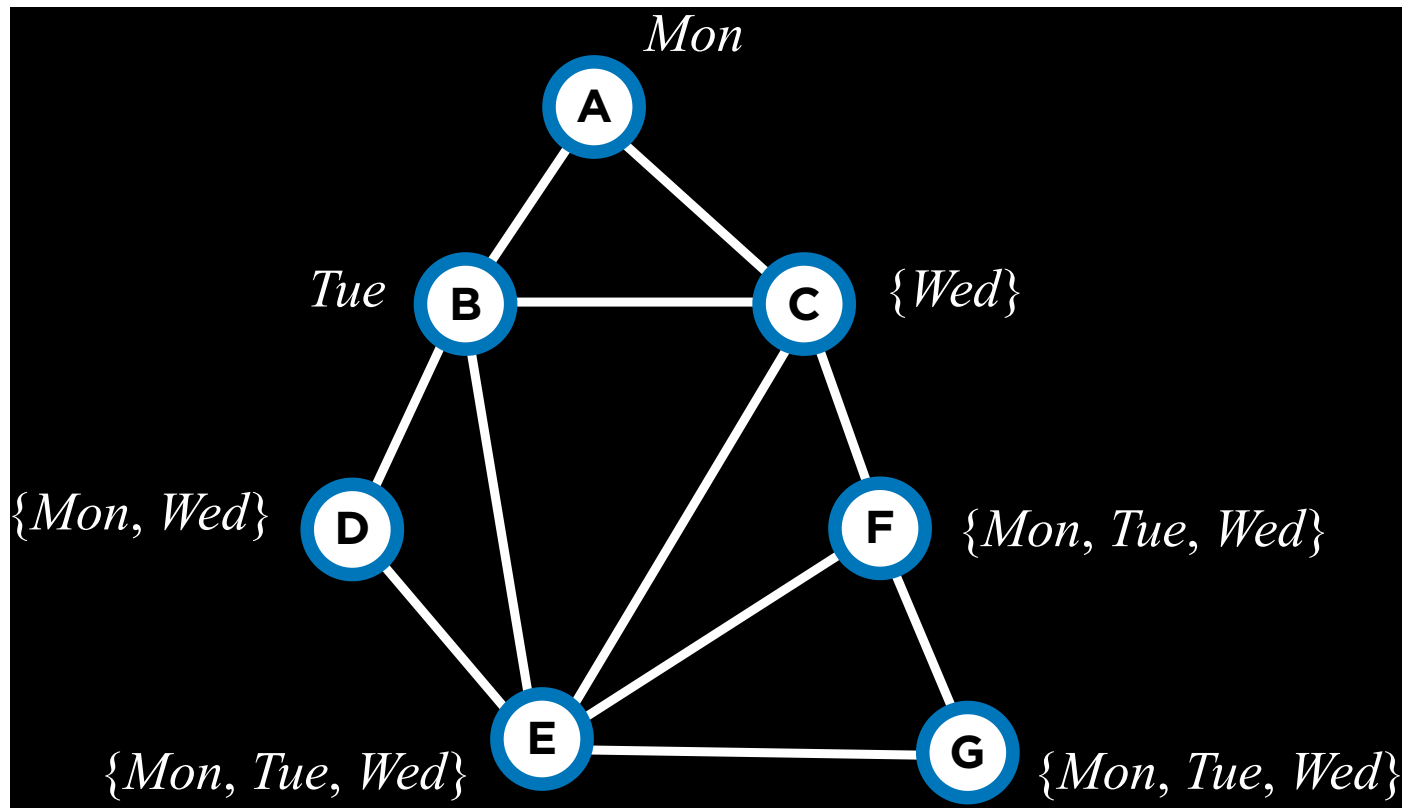
Minimum Remaining Values
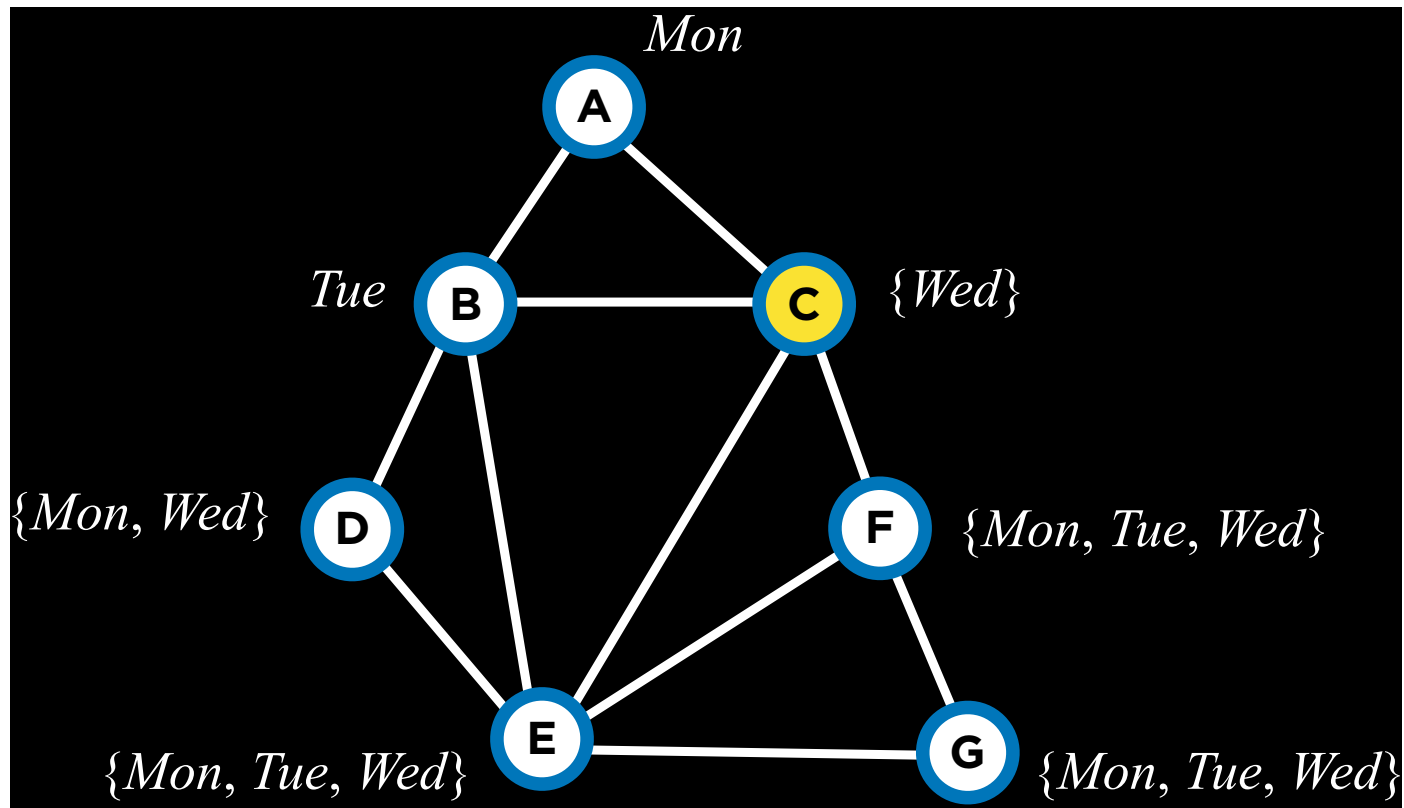
Degree Heuristic

Degree Heuristic
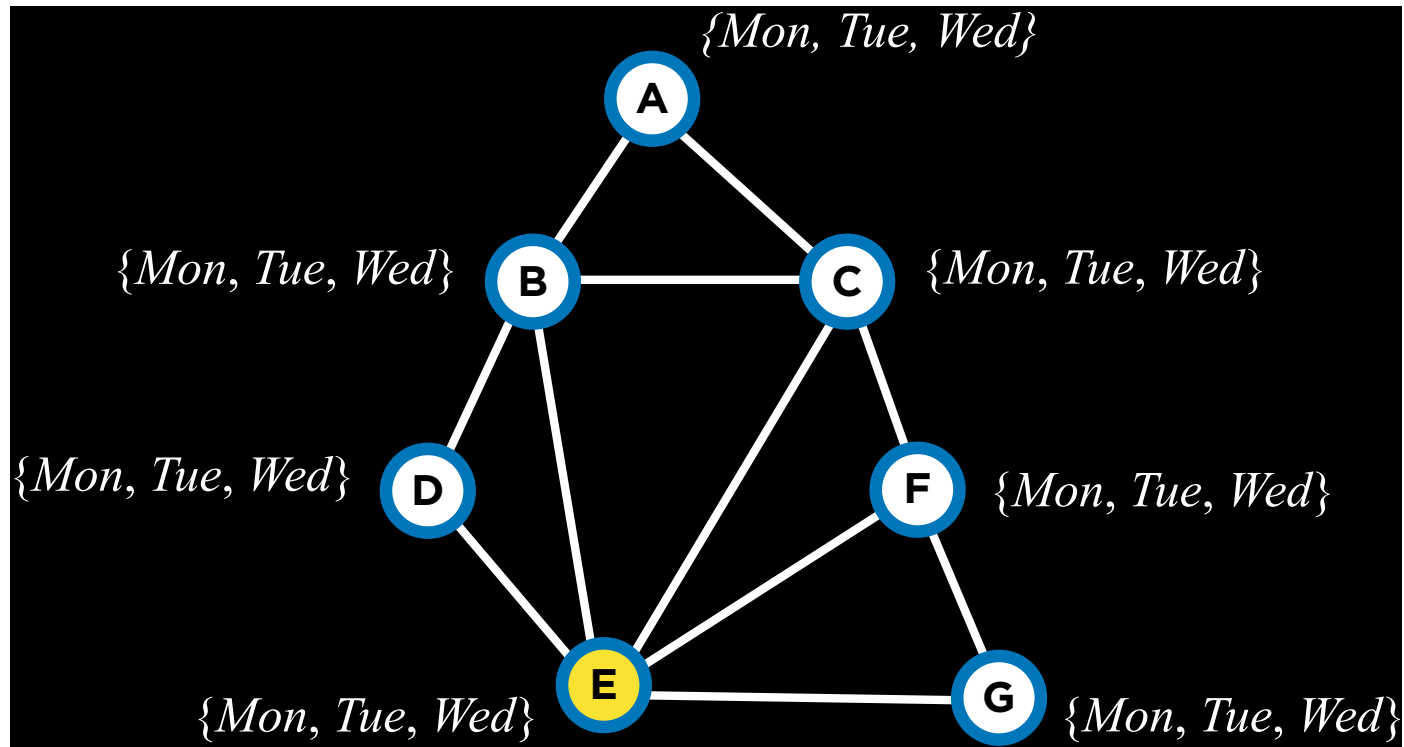
Domain-Values Revisited

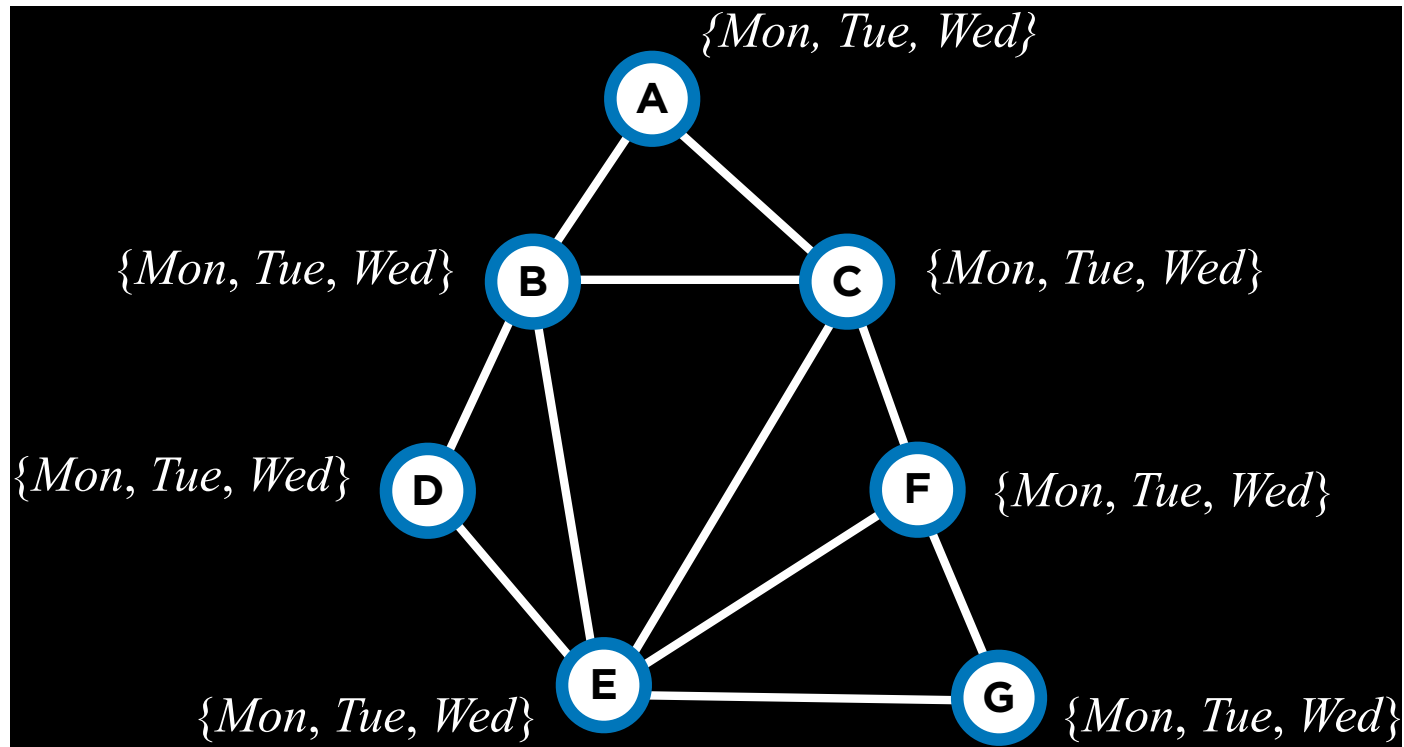- Domain-values takes a domain for a variable and returns a sequence of all the values inside that variable's domain
  - We used a naïve approach where we just go in order Mon, Tue, Wed
- But this order might not be the most efficient one to search in, it might be more efficient to choose values that are likely to be solutions first and then go to other values
- How do we assess whether a value is likely to lead to a solution?
  - We can look at how many things get removed from domains by making this new assignment of a variable to a particular value

# Domain-Values Revisited

```
function BACKTRACK(assignment, csp):
    if assignment complete: return assignment
    var = SELECT-UNASSIGNED-VAR(assignment, csp)
    for value in DOMAIN-VALUES(var, assignment):
        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 value heuristic
  - Returns variables in order by the number of choices that are ruled out for neighboring variables
    - Try least-constraining values first
  - The idea is that if one starts with a value that rules out a lot of other choices, we're ruling out a lot of possibilities likely to lead to a solution

Least-constraining Value Heuristic

- Considering C, what should I choose first, Tue or Wed?
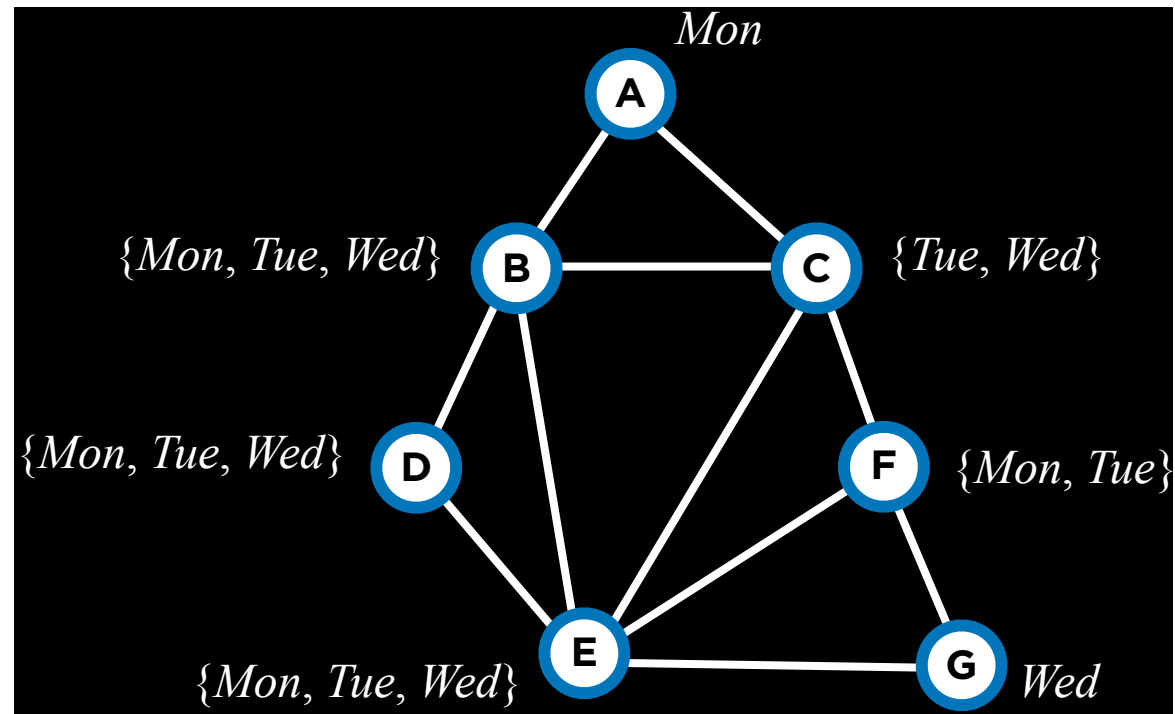
Least-constraining Value Heuristic
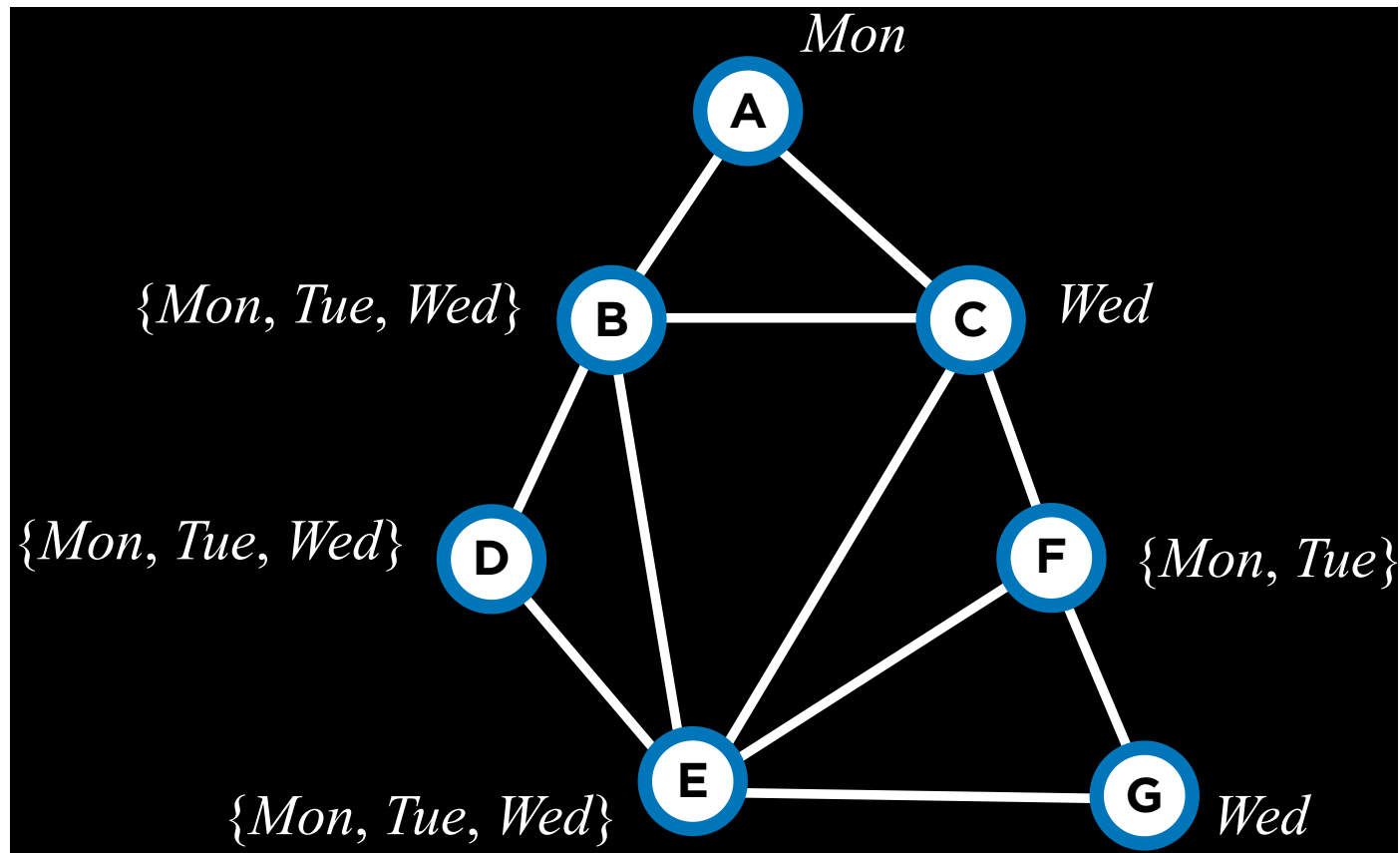
Least-constraining Value Heuristic

- By continuing this process, we will find a solution
  - an assignment of variables to values where each of these classes has an exam date with no conflict