Updated 4/6/2022

# Topic 3: Efficient query evaluation
# Unit 2: Cyclic queries
# Lecture 19

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

https://northeastern-datalab.github.io/cs7240/sp22/

3/29/2022

# Outline: T3-2: Cyclic conjunctive queries

- T3-1: Acyclic conjunctive queries
- T3-2: Cyclic conjunctive queries
  - 2SAT (a detour)
  - Tree decompositions
  - Decompositions of hypertrees
  - Duality in Linear programming (a quick primer)
  - AGM bound (maximal result size for full CQs)
  - Worst-case optimal joins & the triangle query
  - Worst-case optimal joins & the 4-cycle
  - Optimal joins & the 4-cycle

cycles make everything more complicated ☹

# Why cyclic queries (other than social networks)

```
Likes(person, drink)
Frequents(person, bar)
Serves(bar, drink, cost)
```

**2. Specify or choose a Query** Supported grammar

104 Bars: Persons who frequent some bar that serves some drink they like.

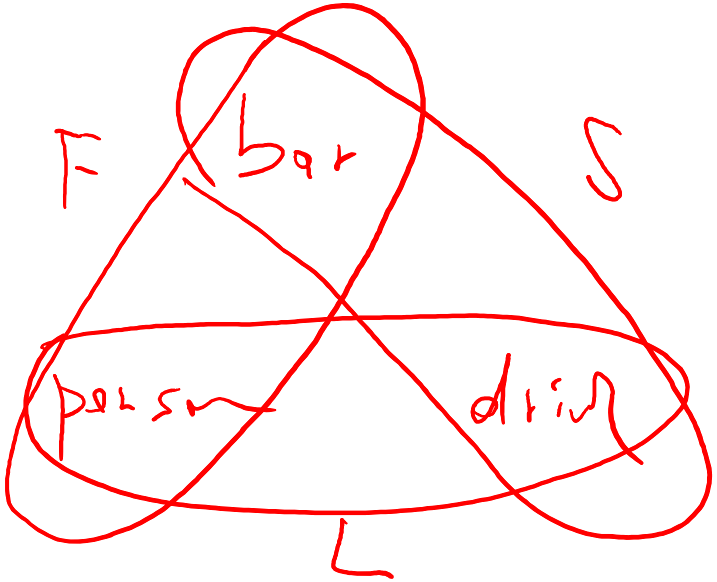
Source: http://demo.queryvis.com

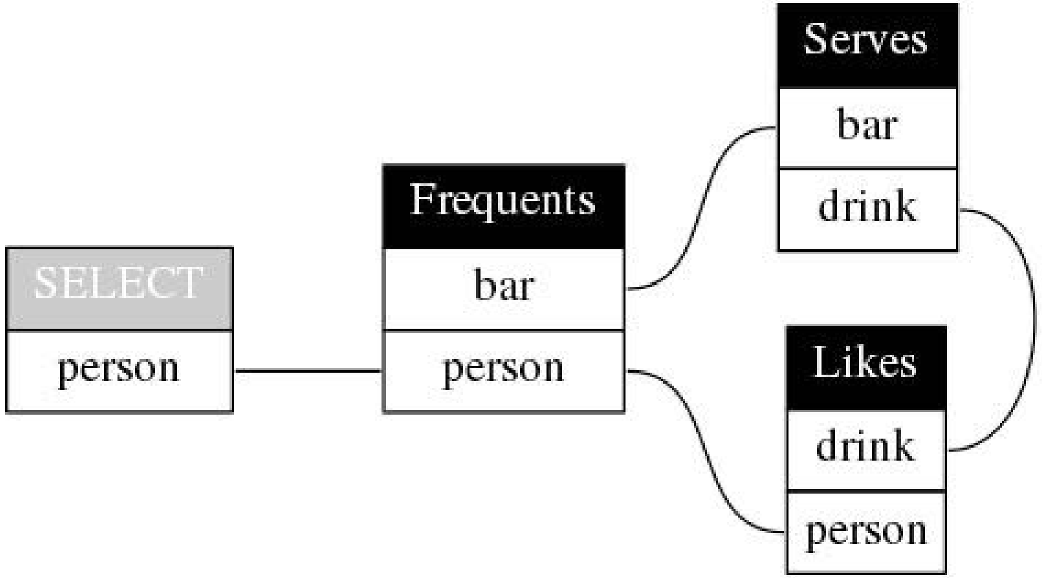
# Why cyclic queries (other than social networks)

```
Likes(person, drink)
Frequents(person, bar)
Serves(bar, drink, cost)
```

**2. Specify or choose a Query** — Supported grammar

104 Bars: Persons who frequent some bar that serves some drink they like.




Source: http://demo.queryvis.com

# Why cyclic queries (other than social networks)

```
Likes(person, drink)
Frequents(person, bar)
Serves(bar, drink, cost)
```

**2. Specify or choose a Query** [Supported grammar]

104 Bars: Persons who frequent some bar that serves some drink they like.

```
SELECT  F1.person
FROM    Frequents F1
WHERE   exists
        (SELECT *
        FROM    Serves S2
        WHERE   S2.bar = F1.bar
        AND     exists
                (SELECT *
                FROM    Likes L3
                WHERE   L3.person = F1.person
                AND     S2.drink = L3.drink))
```

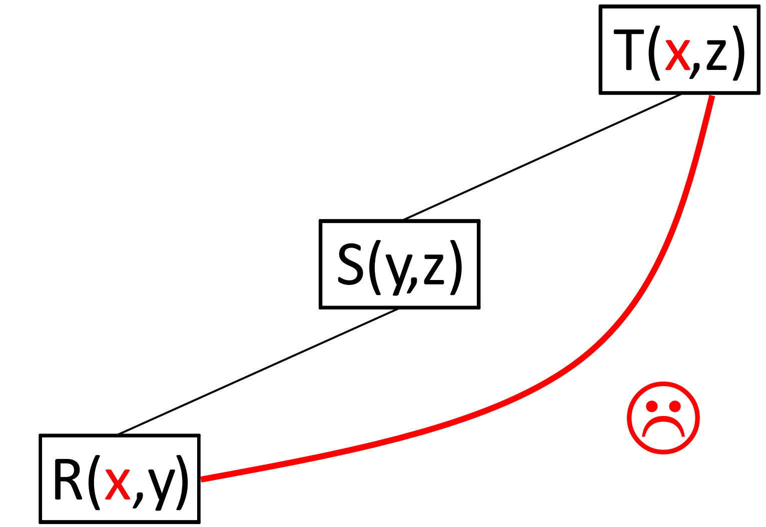
Source: http://demo.queryvis.com

# Joins in databases: one-at-a-time

How can we efficiently process multi-way joins with cycles?

Q(x,y,z) :- R(x,y), S(y,z), T(x,z).

Recall:

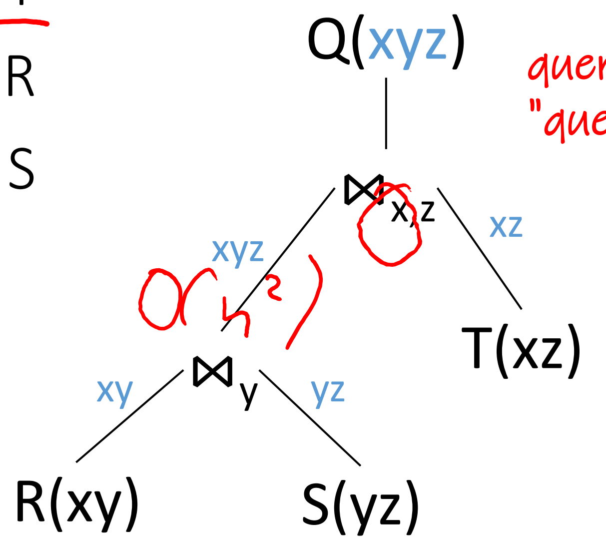



There is no join tree! You can't fulfill the running intersection property...

Three possible plans

- (R ⋈ S) ⋈ T
- (S ⋈ T) ⋈ R
- (T ⋈ R) ⋈ S

query plan as "query tree"




☹

- there is no full semijoin reducer
- intermediate result size bigger than output

Can we do better for cyclic queries? ☺

# Outline: T3-2: Cyclic conjunctive queries

- T3-1: Acyclic conjunctive queries
- T3-2: Cyclic conjunctive queries
  - 2SAT (a detour)
  - Tree decompositions
  - Decompositions of hypertrees
  - Duality in Linear programming (a quick primer)
  - AGM bound (maximal result size for full CQs)
  - Worst-case optimal joins & the triangle query
  - Worst-case optimal joins & the 4-cycle
  - Optimal joins & the 4-cycle

# 2SAT

$\varphi = (x \lor y) \land (\neg y \lor z) \land (\neg x \lor \neg z) \land (z \lor y)$

- Instance: A 2-CNF formula $\varphi$
- Problem: To decide if $\varphi$ is satisfiable

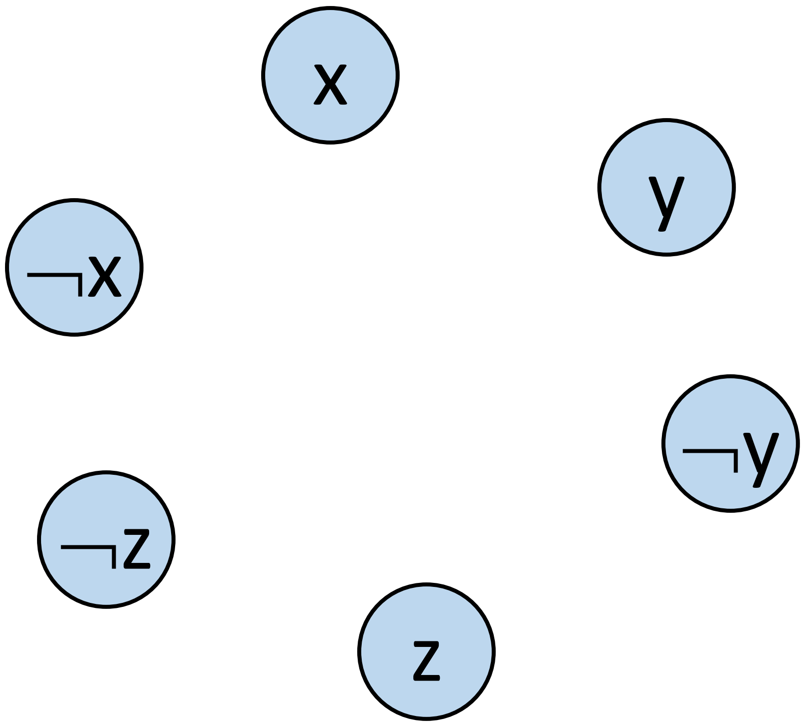
- Theorem: 2SAT is polynomial-time decidable.
  - Proof: We'll show how to solve this problem efficiently using path searches in graphs…

- Background: Given a graph G=(V,E) and two vertices $s,t \in V$, finding if there is a path from s to t in G is polynomial-time decidable. Use some search algorithm (DFS/BFS).

# 2SAT: Graph Construction

$\varphi = (x \lor y) \land (\neg y \lor z) \land (\neg x \lor \neg z) \land (z \lor y)$

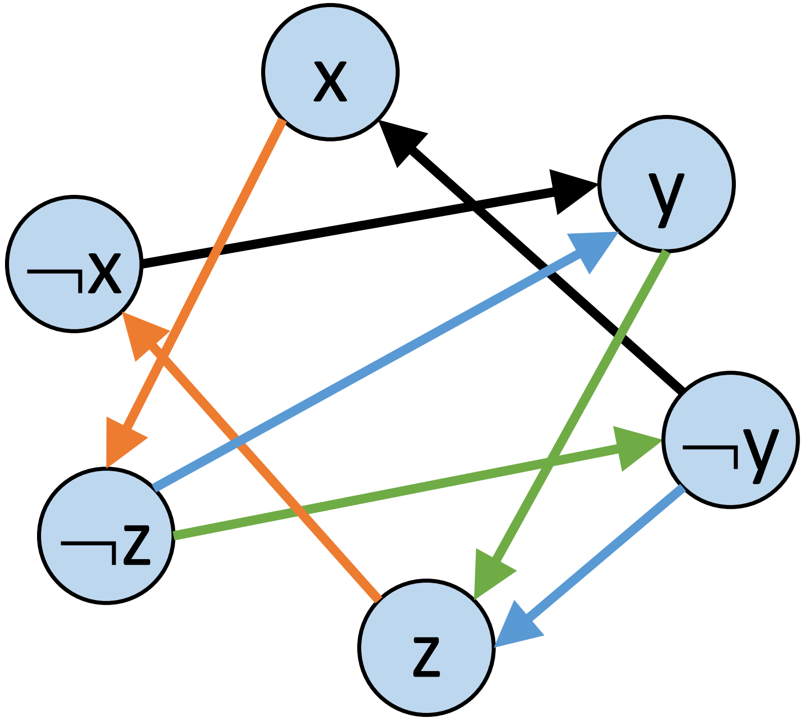
- Vertex for each variable and a negation of a variable



# 2SAT: Graph Construction

$\varphi = (x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg z) \wedge (z \vee y)$

- Vertex for each variable and a negation of a variable
- Edge $(\neg x \rightarrow y)$ iff there exists a clause equivalent to $(x \vee y)$
  - Recall $(x \vee y)$ same as $(\neg x \Rightarrow y)$ and $(\neg y \Rightarrow x)$, thus also $(\neg y \rightarrow x)$

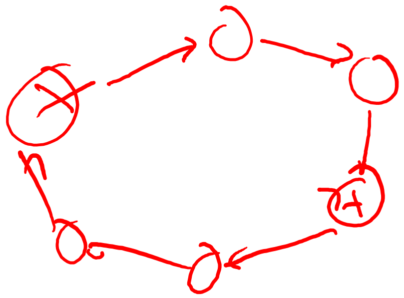


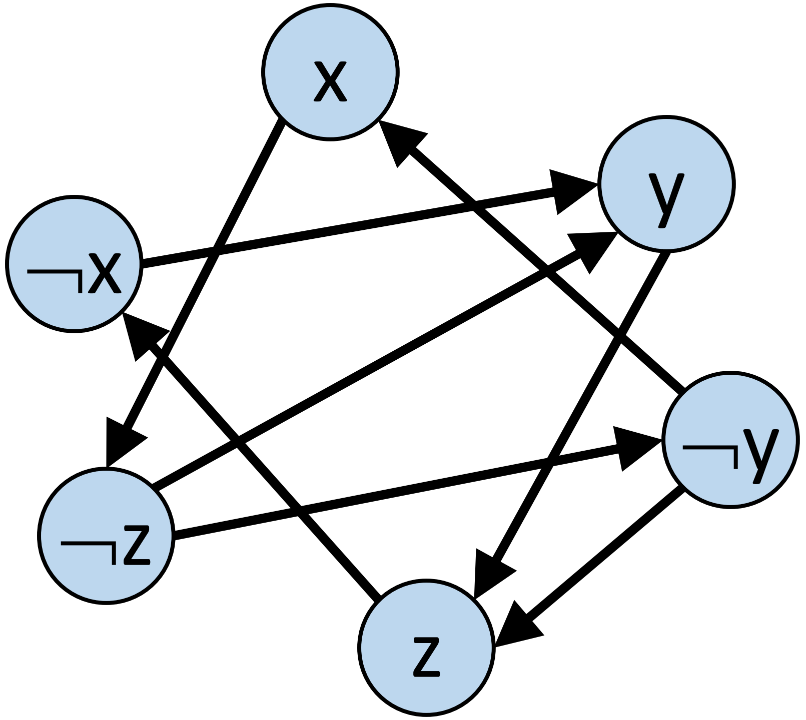
# 2SAT: Graph Construction

φ = (x∨y)∧(¬y∨z)∧(¬x∨¬z)∧(z∨y)

- Vertex for each variable and a negation of a variable
- Edge (¬x→y) iff there exists a clause equivalent to (x∨y)
  - Recall (x∨y) same as (¬x⇒y) and (¬y⇒x), thus also (¬y→x)

- Claim: a 2-CNF formula φ is unsatisfiable iff there exists a variable x, such that:
  - there is a path from x to ¬x in the graph, and
  - there is a path from ¬x to x in the graph



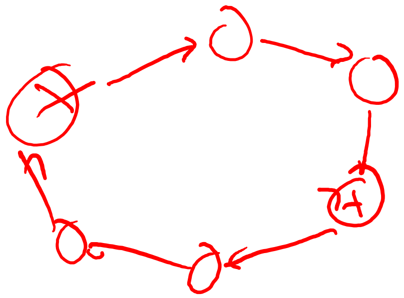


# 2SAT: Graph Construction

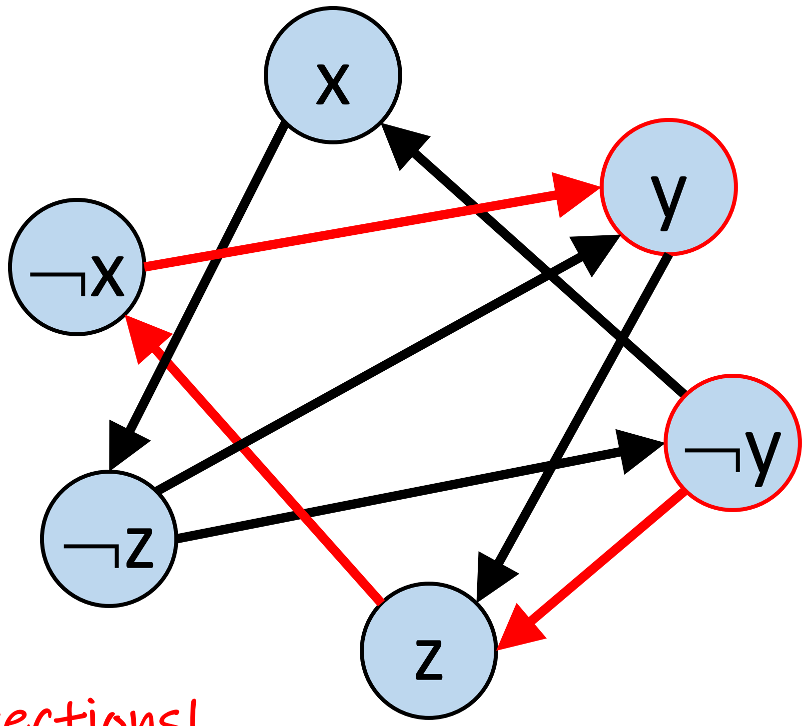
$\varphi = (x \lor y) \land (\neg y \lor z) \land (\neg x \lor \neg z) \land (z \lor y)$

- Vertex for each variable and a negation of a variable
- Edge $(\neg x \rightarrow y)$ iff there exists a clause equivalent to $(x \lor y)$
  - Recall $(x \lor y)$ same as $(\neg x \Rightarrow y)$ and $(\neg y \Rightarrow x)$, thus also $(\neg y \rightarrow x)$

- Claim: a 2-CNF formula $\varphi$ is unsatisfiable iff there exists a variable x, such that:
  - there is a path from x to ¬x in the graph, and
  - there is a path from ¬x to x in the graph



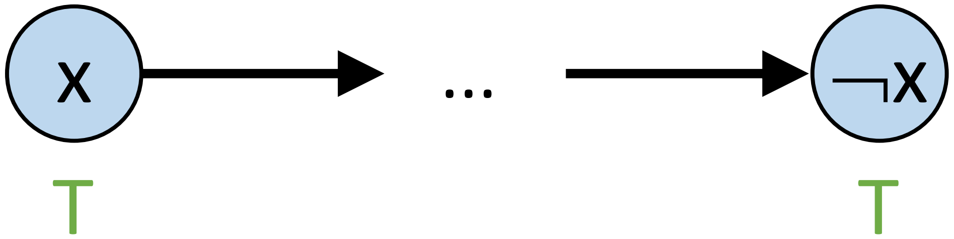
not enough,
needs both directions!



# Correctness (1)

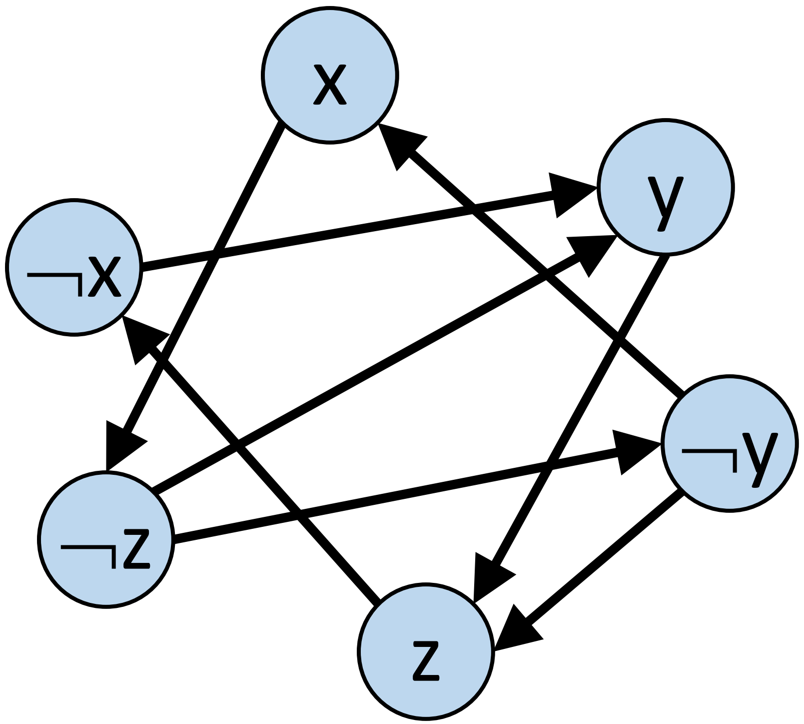
$\varphi = (x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg z) \wedge (z \vee y)$

- Suppose there are paths $x..\neg x$ and $\neg x..x$ for some variable $x$, but there's also a satisfying assignment $\rho$.
  - If $\rho(x)=T$:




  - Similarly for $\rho(x)=F$...

recall, needs to hold in both directions!

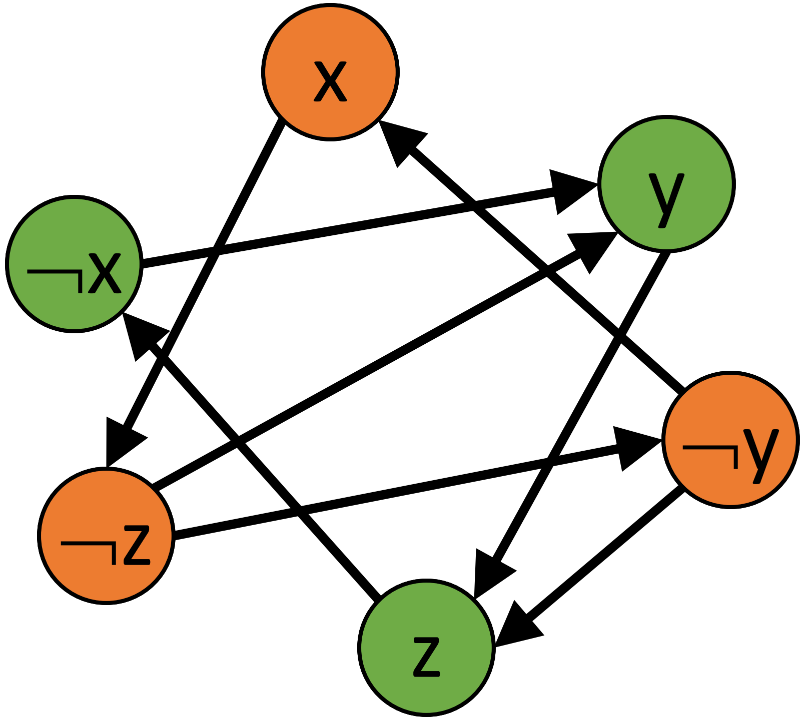


# Correctness (2)

$\varphi = (x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg z) \wedge (z \vee y)$

- Suppose there are no variables with such paths.
- Construct an assignment as follows:

1. pick an unassigned literal $\alpha$, with no path from $\alpha$ to $\neg\alpha$, and assign it T
2. assign T to all reachable vertices
3. assign F to their negations
4. Repeat until all vertices are assigned



# 2SAT is in P

We get the following PTIME algorithm for 2SAT:

- For each variable x find if there is a path from x to $\neg$x and vice-versa.
- Reject if any of these tests succeeded.
- Accept otherwise.

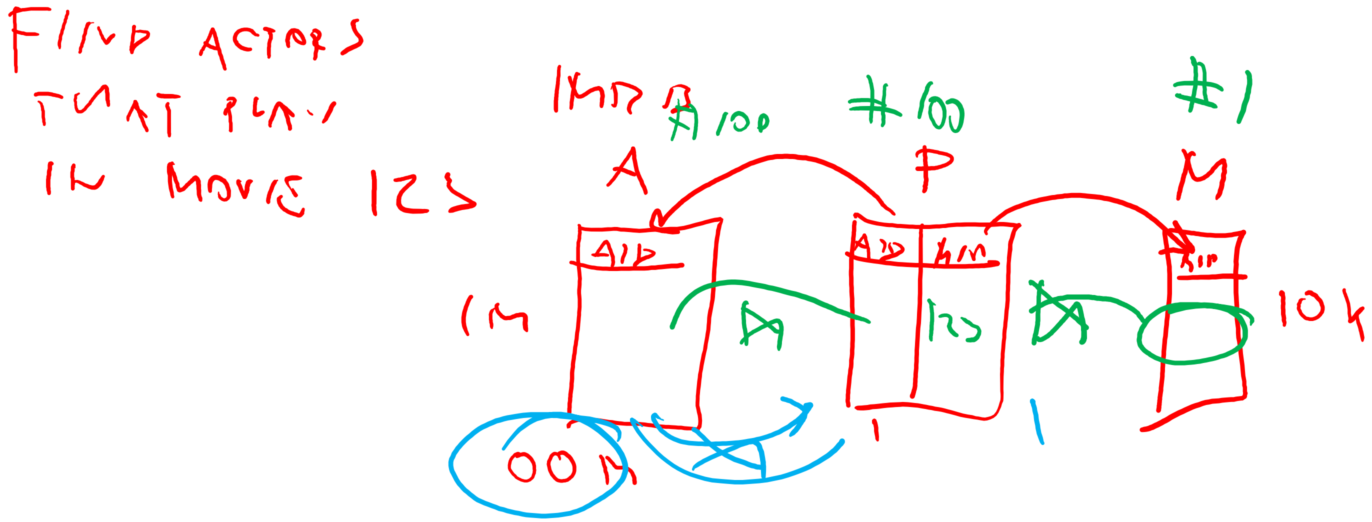
$\Rightarrow$ 2SAT$\in$P. ■

# Outline: T3-2: Cyclic conjunctive queries

- T3-1: Acyclic conjunctive queries
- T3-2: Cyclic conjunctive queries
  - 2SAT (a detour)
  - Tree decompositions
  - Decompositions of hypertrees
  - Duality in Linear programming (a quick primer)
  - AGM bound (maximal result size for full CQs)
  - Worst-case optimal joins & the triangle query
  - Worst-case optimal joins & the 4-cycle
  - Optimal joins & the 4-cycle

# Join Processing: two approaches

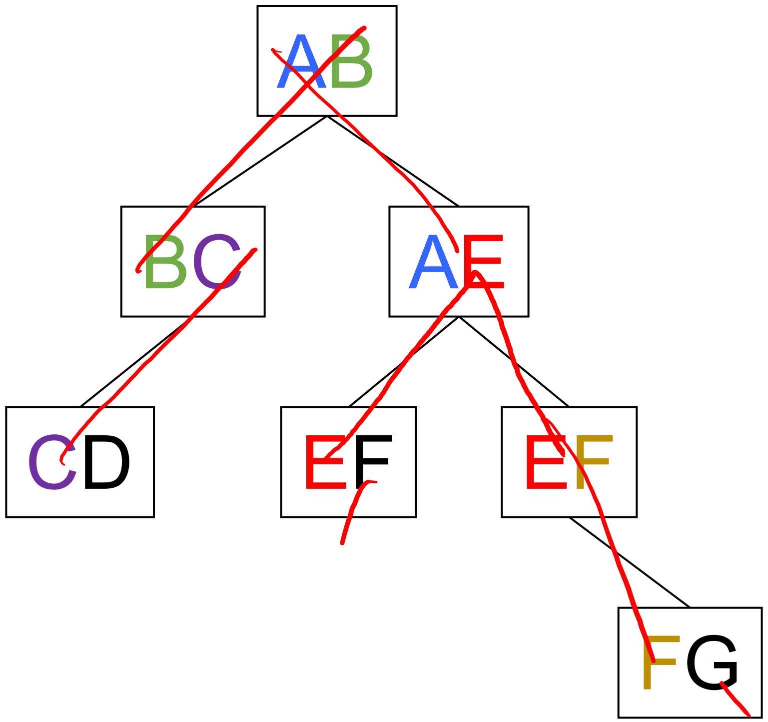
1. Cardinality-based
   - binary joins, consider the sizes of input relations as to reduce the intermediate sizes
   - commercial DBMSs: series of pairwise joins, system R (Selinger), join size estimation
2. Structural approaches (next)
   - acylicity: Yannakakis, GYO algorithm, join tree
   - bounded "width": query width, hypertree width (hw), generalized hw (ghw). All go back to notion of treewidth (work by Robertson & Seymour on graph minors)



AGM: fractional hw (fhw):

- consider both statistics on relations and query structure

# Definition of an attribute-connected tree




DEFINITION: A tree is attribute-connected if the subtree induced by each attribute is connected

Same as the running intersection property from join trees

Also called "coherence"

# Tree decomposition

A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one

# Tree decomposition example 1: a tree

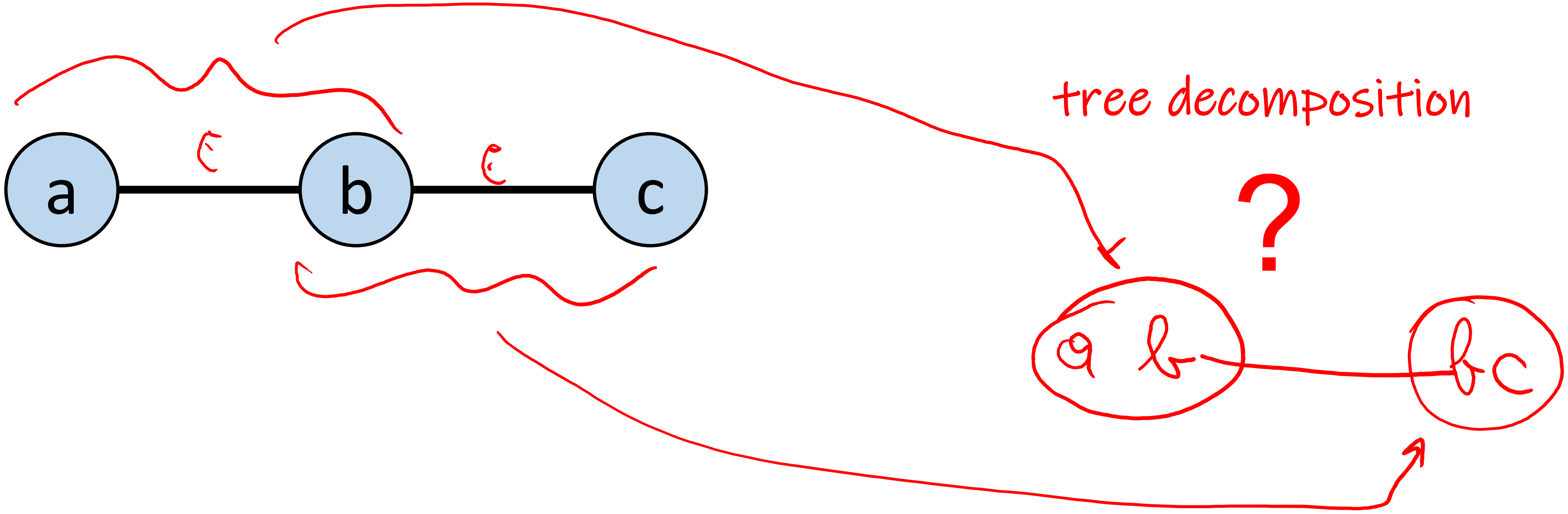
A tree decomposition of graph $\mathrm{G(N, E)}$ is a tree $\mathrm{T(V, F)}$ and a subset $\mathrm{N_v} \subseteq \mathrm{N}$ assigned to each vertex (or "supernode") $\mathrm{v} \in \mathrm{V}$ s.t.:

(1) Node coverage: Every vertex of $\mathrm{G}$ is assigned at least one vertex in $\mathrm{T}$

(2) Edge coverage: For every edge $\mathrm{e}$ of $\mathrm{G}$, there is a vertex in $\mathrm{T}$ that contains both ends of $\mathrm{e}$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one



# Tree decomposition example 1: a tree

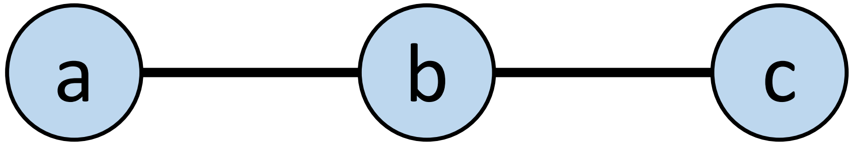
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

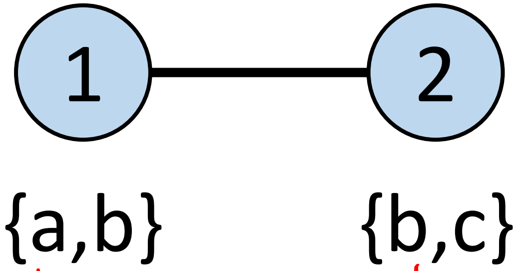
(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one




That's why treewidth defined as max cardinality - 1

# Tree decomposition example 2

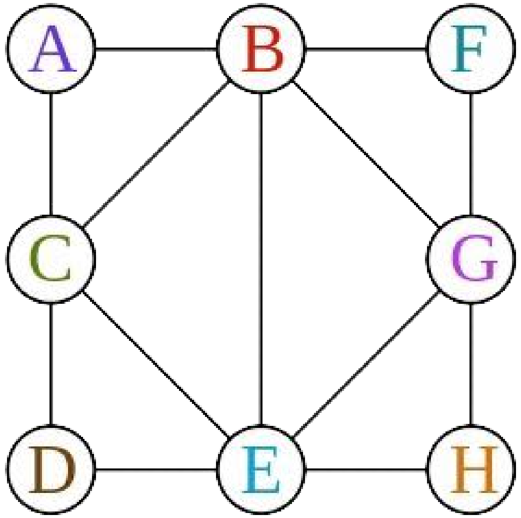
A tree decomposition of graph $\mathrm{G(N, E)}$ is a tree $\mathrm{T(V, F)}$ and a subset $\mathrm{N_v \subseteq N}$ assigned to each vertex (or "supernode") $\mathrm{v \in V}$ s.t.:

(1) Node coverage: Every vertex of $\mathrm{G}$ is assigned at least one vertex in $\mathrm{T}$

(2) Edge coverage: For every edge $\mathrm{e}$ of $\mathrm{G}$, there is a vertex in $\mathrm{T}$ that contains both ends of $\mathrm{e}$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one




tree decomposition

?

Example from: https://en.wikipedia.org/wiki/Tree_decomposition

# Tree decomposition example 2

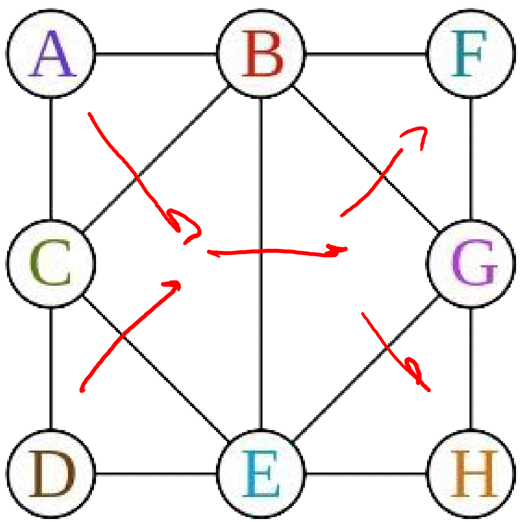
A tree decomposition of graph $\mathrm{G(N, E)}$ is a tree $\mathrm{T(V, F)}$ and a subset $\mathrm{N_v \subseteq N}$ assigned to each vertex (or "supernode") $\mathrm{v \in V}$ s.t.:

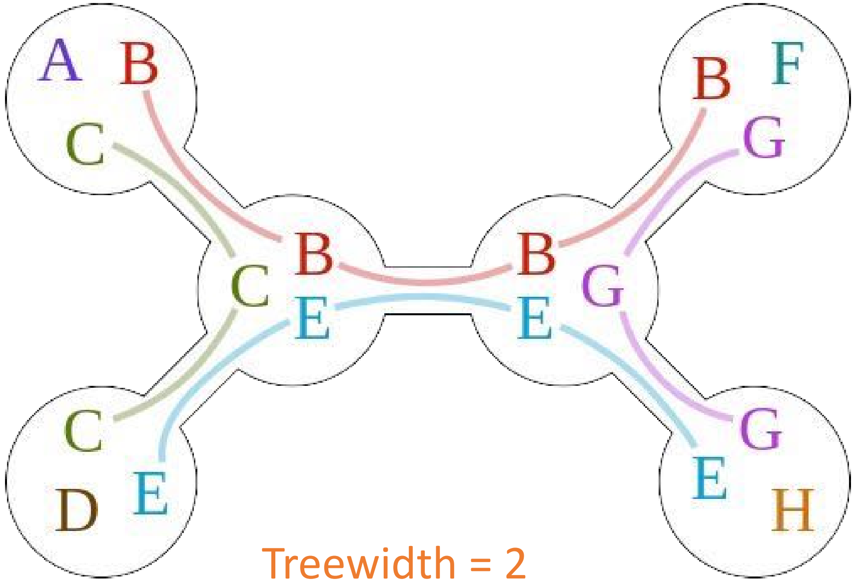
(1) Node coverage: Every vertex of $\mathrm{G}$ is assigned at least one vertex in $\mathrm{T}$

(2) Edge coverage: For every edge $\mathrm{e}$ of $\mathrm{G}$, there is a vertex in $\mathrm{T}$ that contains both ends of $\mathrm{e}$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one




Treewidth = 2

Notice running intersection property

Example from: https://en.wikipedia.org/wiki/Tree_decomposition

# Tree decomposition example 3

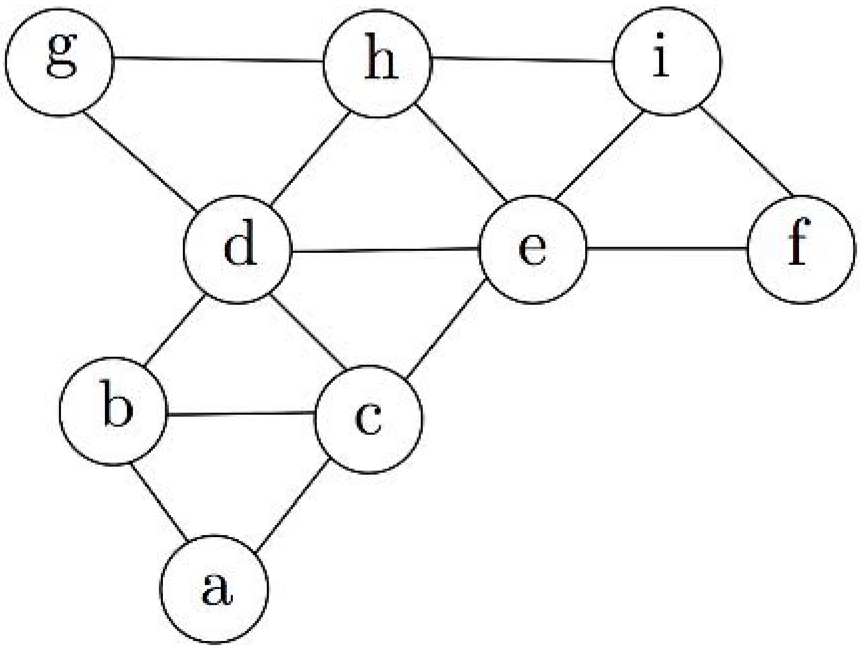
A tree decomposition of graph $\mathrm{G(N, E)}$ is a tree $\mathrm{T(V, F)}$ and a subset $\mathrm{N_v} \subseteq \mathrm{N}$ assigned to each vertex (or "supernode") $\mathrm{v} \in \mathrm{V}$ s.t.:

(1) Node coverage: Every vertex of $\mathrm{G}$ is assigned at least one vertex in $\mathrm{T}$

(2) Edge coverage: For every edge $\mathrm{e}$ of $\mathrm{G}$, there is a vertex in $\mathrm{T}$ that contains both ends of $\mathrm{e}$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one




tree decomposition

?

Example from: https://www.mi.fu-berlin.de/en/inf/groups/abi/teaching/lectures/lectures_past/WS0910/V____Discrete_Mathematics_for_Bioinformatics__P1/material/scripts/treedecomposition1.pdf

# Tree decomposition example 3

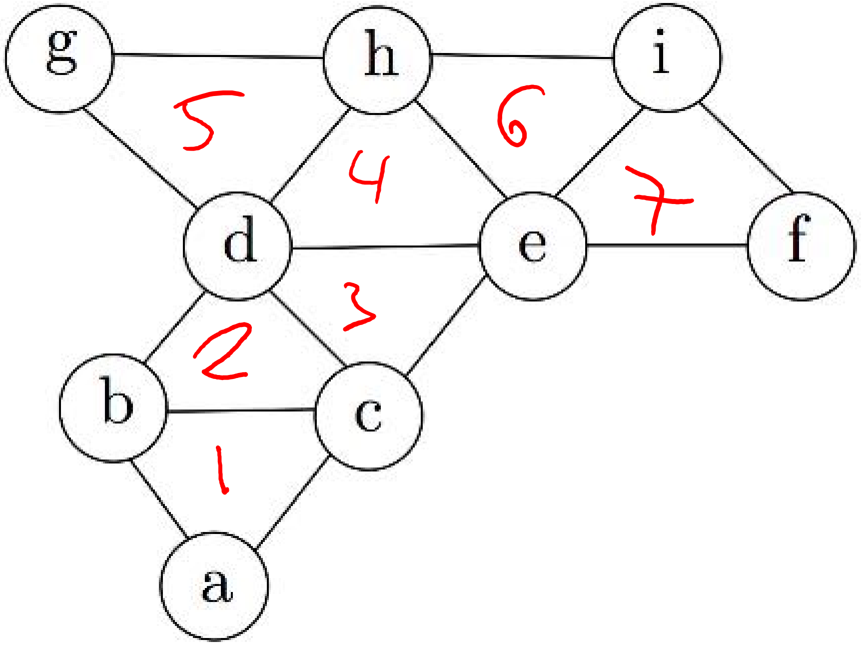
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

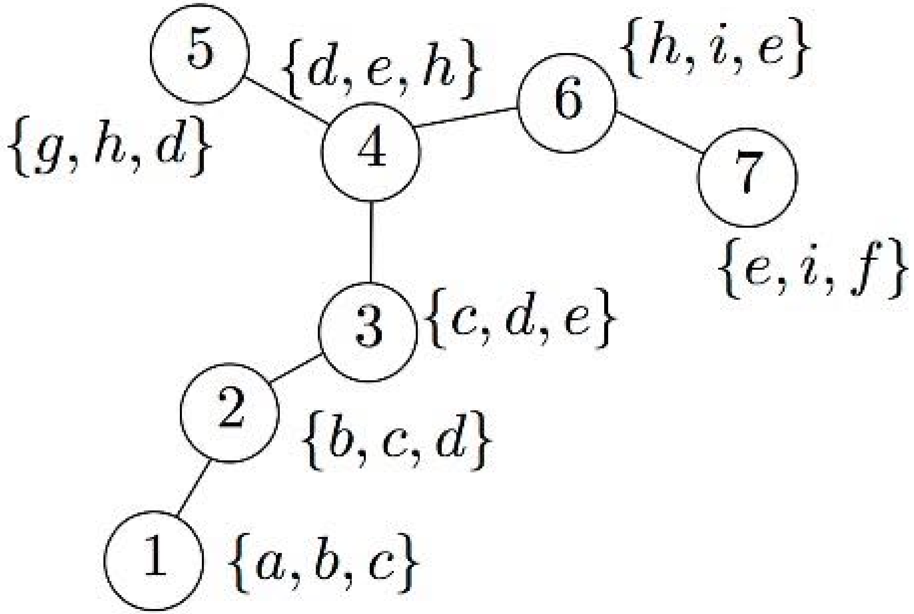
(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one



# Tree decomposition example 4: a cycle

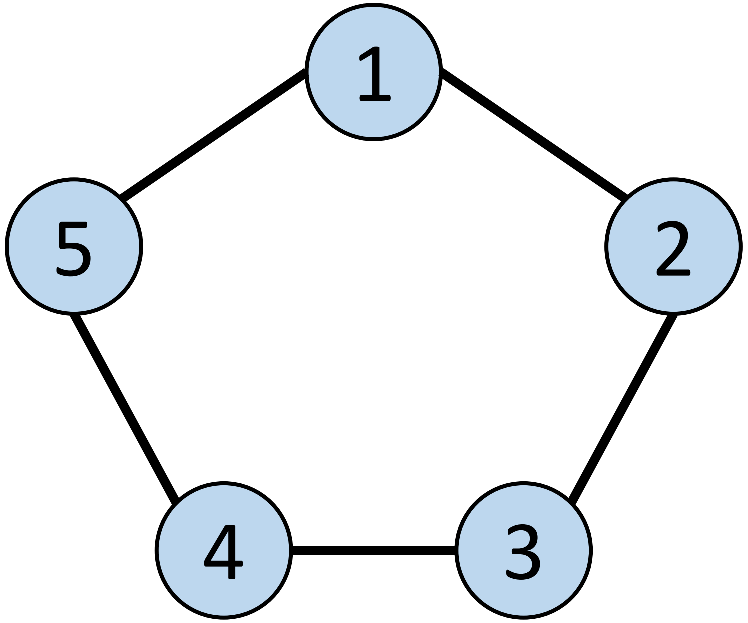
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one




tree decomposition

?

# Tree decomposition example 4: a cycle

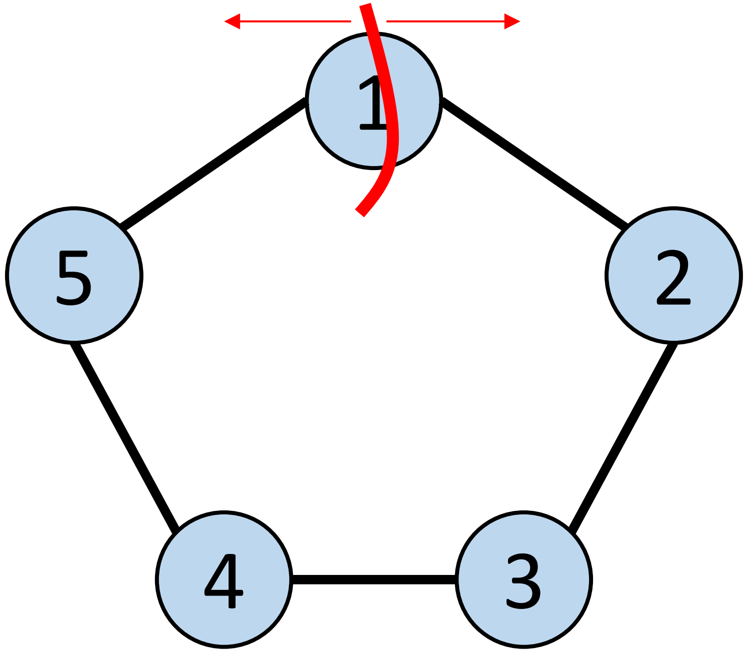
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

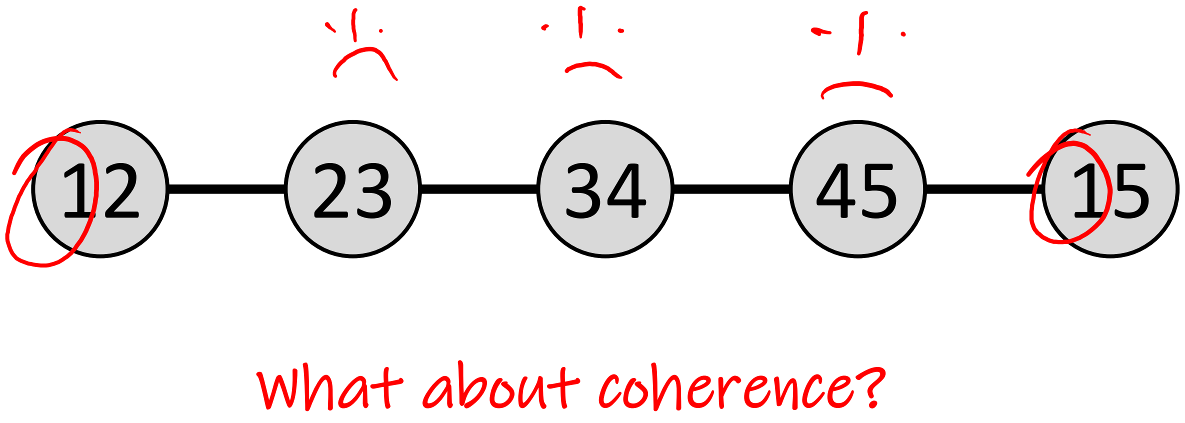
(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one



# Tree decomposition example 4: a cycle

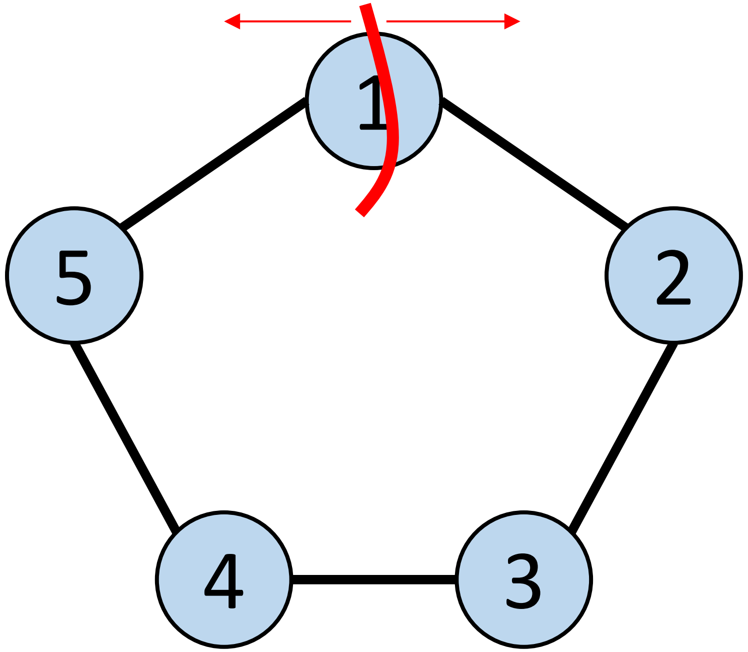
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

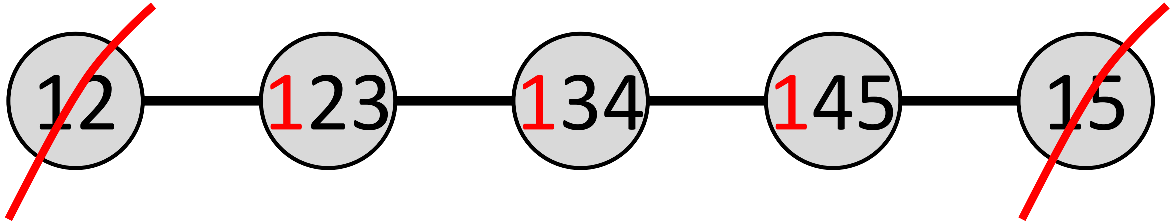
(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one



# Tree decomposition example 4: a cycle

A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one