

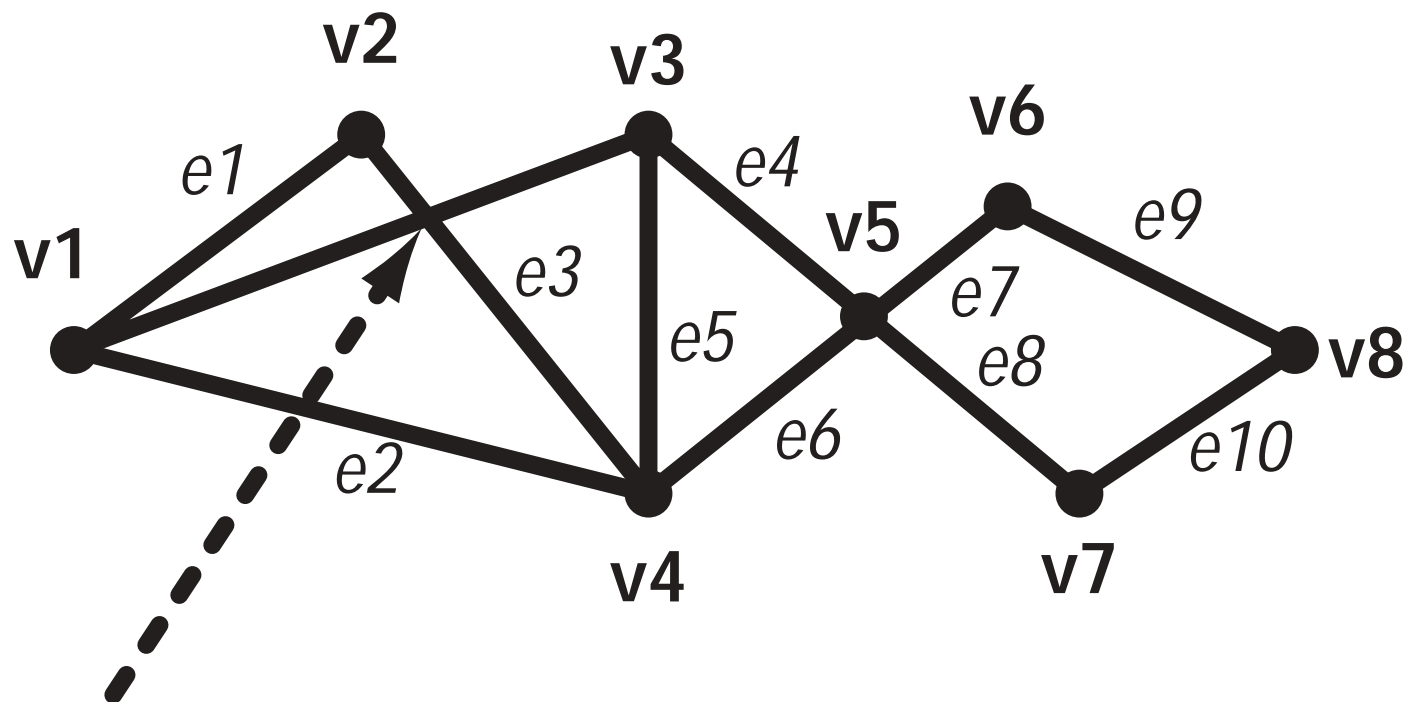
CS20a: NP problems

- Graph theory
 - *Strongly-connected components*



Graph theory

- A *graph* is a set of points (*vertices*) that are interconnected by a set of lines (*edges*)



This is not a vertex



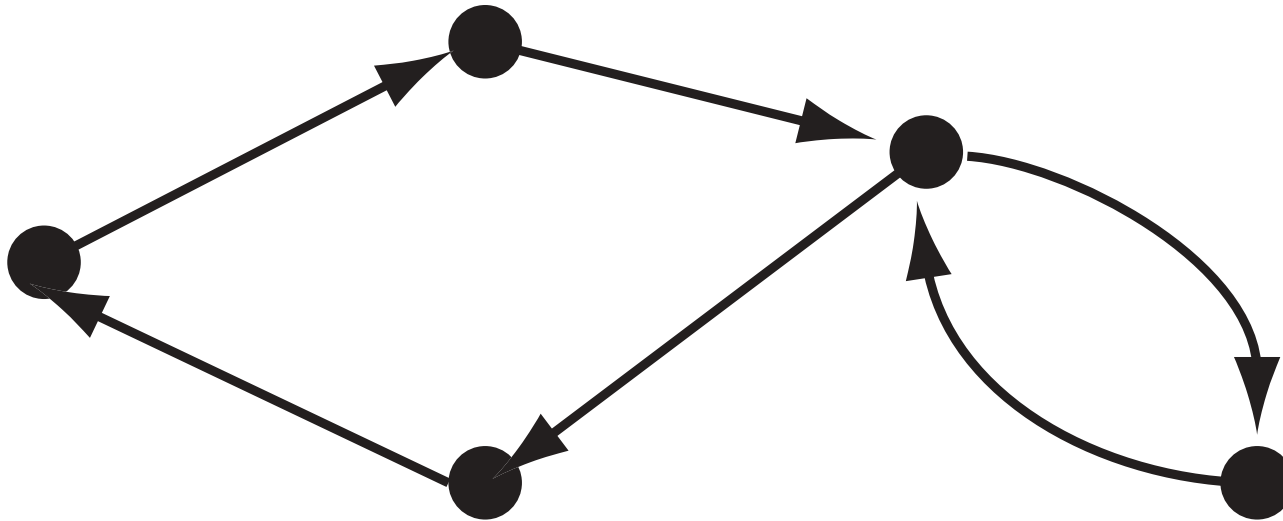
Formal definition of graphs

- A graph G is defined as a pair $G = (V, E)$ where
 - V is a set of vertices
 - E is a set of edges $(v_i, v_j), \dots$
- $n = |V|$ is the size of the graph
- $|E|$ is the number of edges



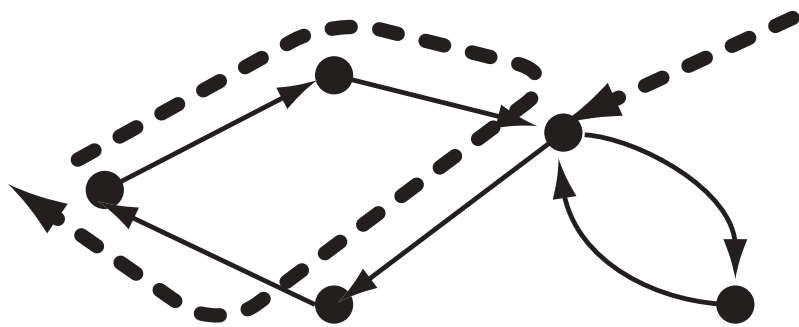
Directed graphs

- A *directed graph* assigns a direction to the edges



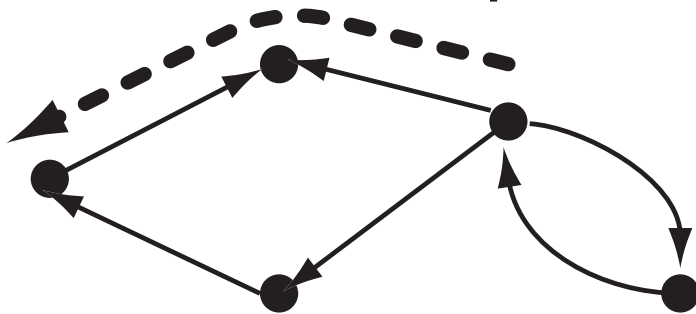
Paths, cycles

A directed cycle

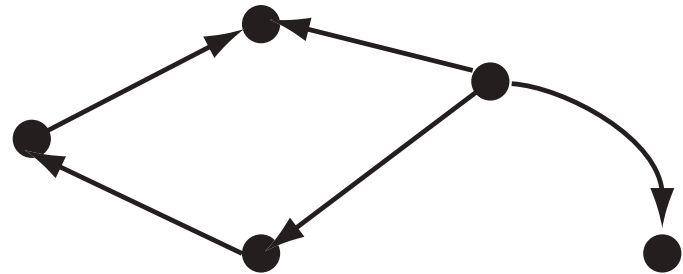


An articulation point

An undirected path



An acyclic graph



Connected components

- A *directed* graph G is defined as a pair $G = (V, E)$ where
 - V is a set of vertices
 - E is a set of edges $(v_i \rightarrow v_j), \dots$
- Two vertices v_i, v_j are *strongly connected* iff there is a directed path from v_i to v_j , **and** a directed path from v_j to v_i .
- Two vertices v_i, v_j are *weakly connected* iff there is an undirected path from v_i to v_j
- Use these to define *strongly-connected* and *weakly-connected* components.

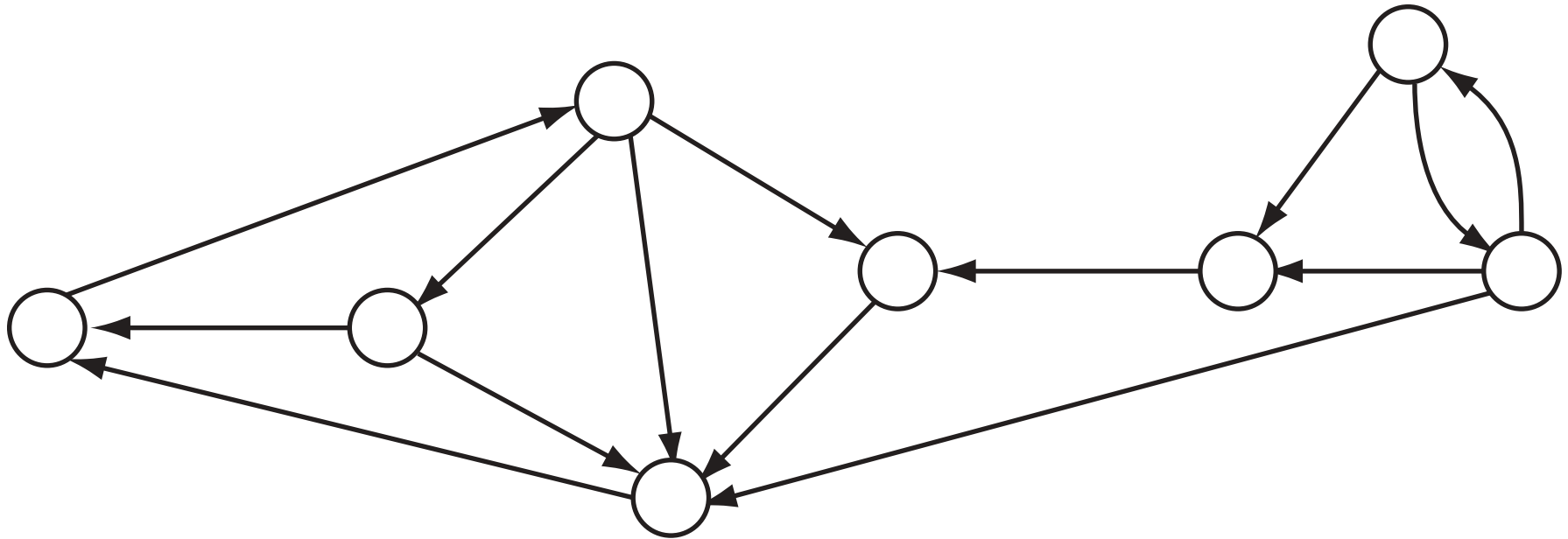


Using DFS to get strongly-connected comp

- Variables:
 - *count*: the current DFS index
 - $dfs[v]$: the DFS number of vertex v
 - $low[v]$: the lowest numbered vertex x such that there is a back-edge from some descendent of v to x
 - *stack* a stack of the collected vertices
- if $low[v] = dfs[v]$, then v is the root of a strongly-connected component

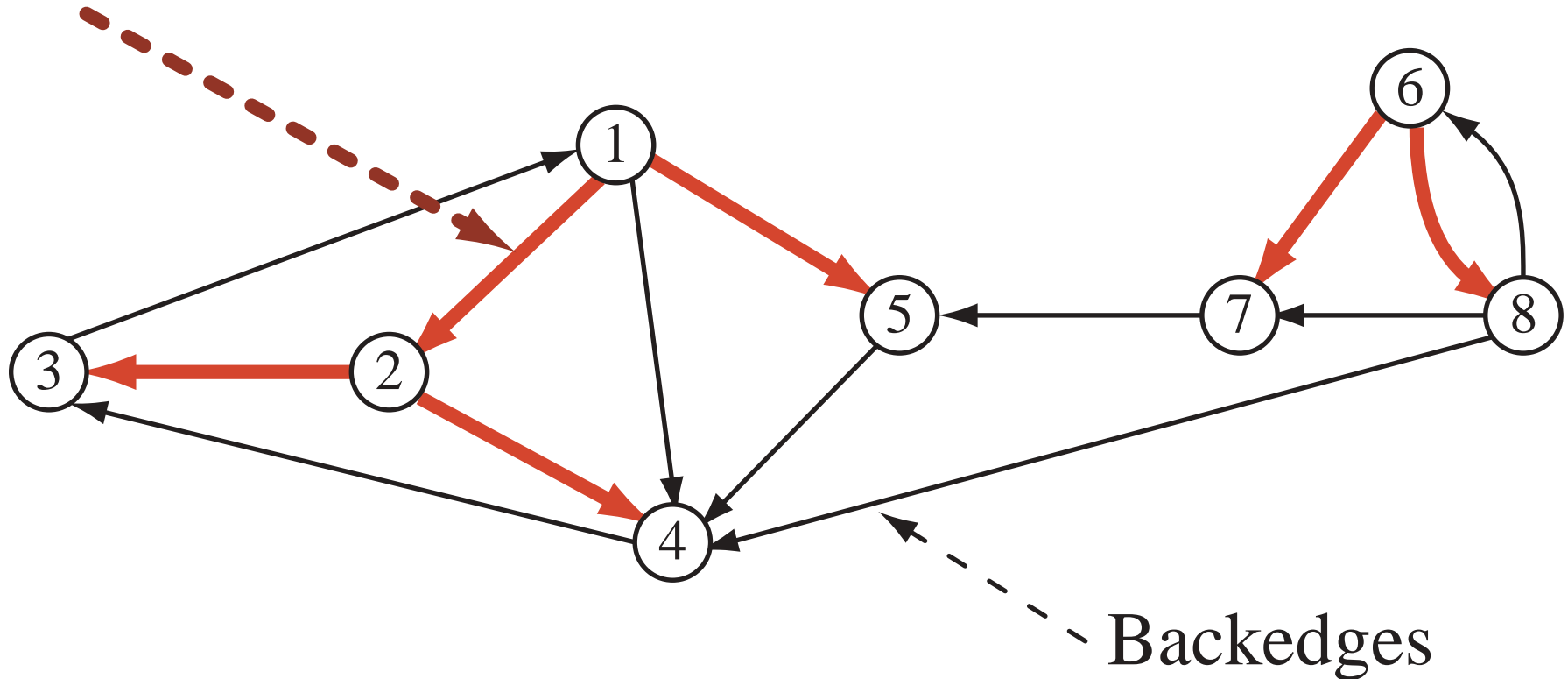


Strongly-connected components

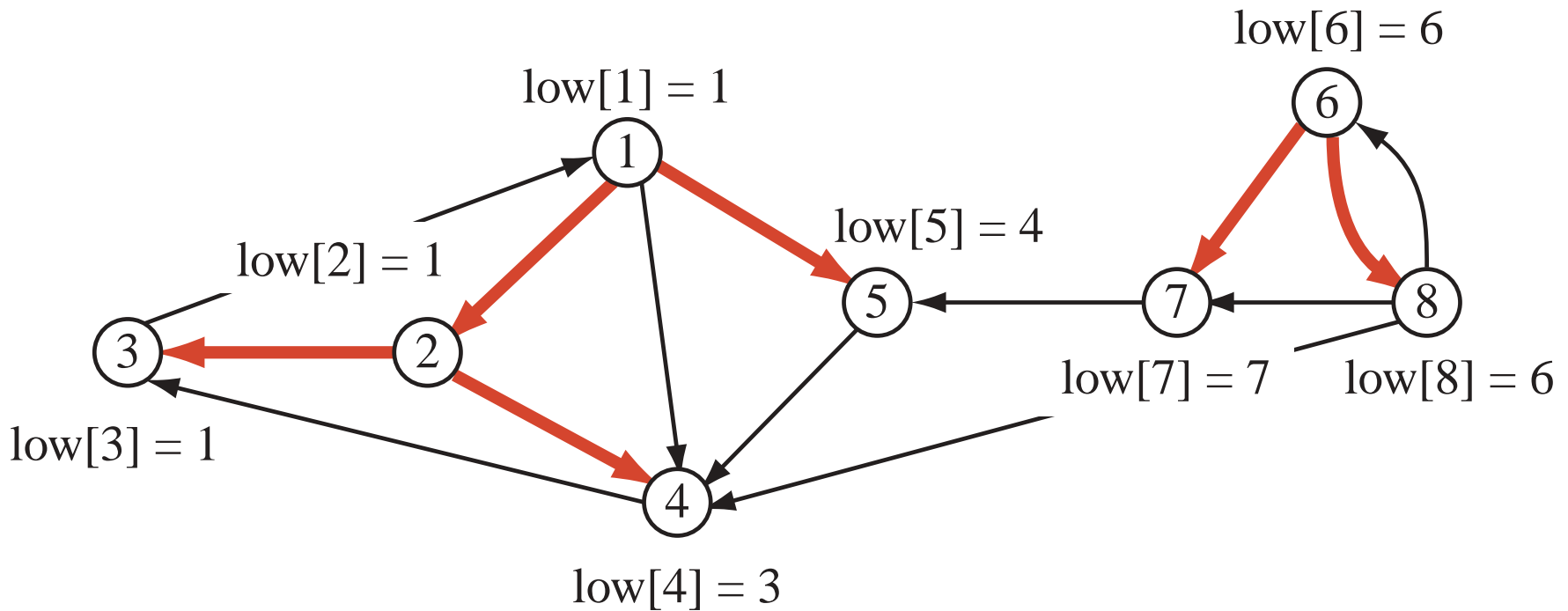


Compute DFS spanning forest

DFS Forest



Compute low values



$x = \min\{\mathbf{low}(w) \mid w \text{ is an immediate descendent of } v\}$

$y = \min\{z \mid z \text{ is reachable by a back-edge from } v\}$

$\mathbf{low}(v) = \min(x, y)$



Using DFS to get strongly-connected comp

- Variables:
 - *count*: the current DFS index
 - $dfs[v]$: the DFS number of vertex v
 - $low[v]$: the lowest numbered vertex x such that there is a back-edge from some descendent of v to x
 - *stack* a stack of the collected vertices
- if $low[v] = dfs[v]$, then v is the root of a strongly-connected component

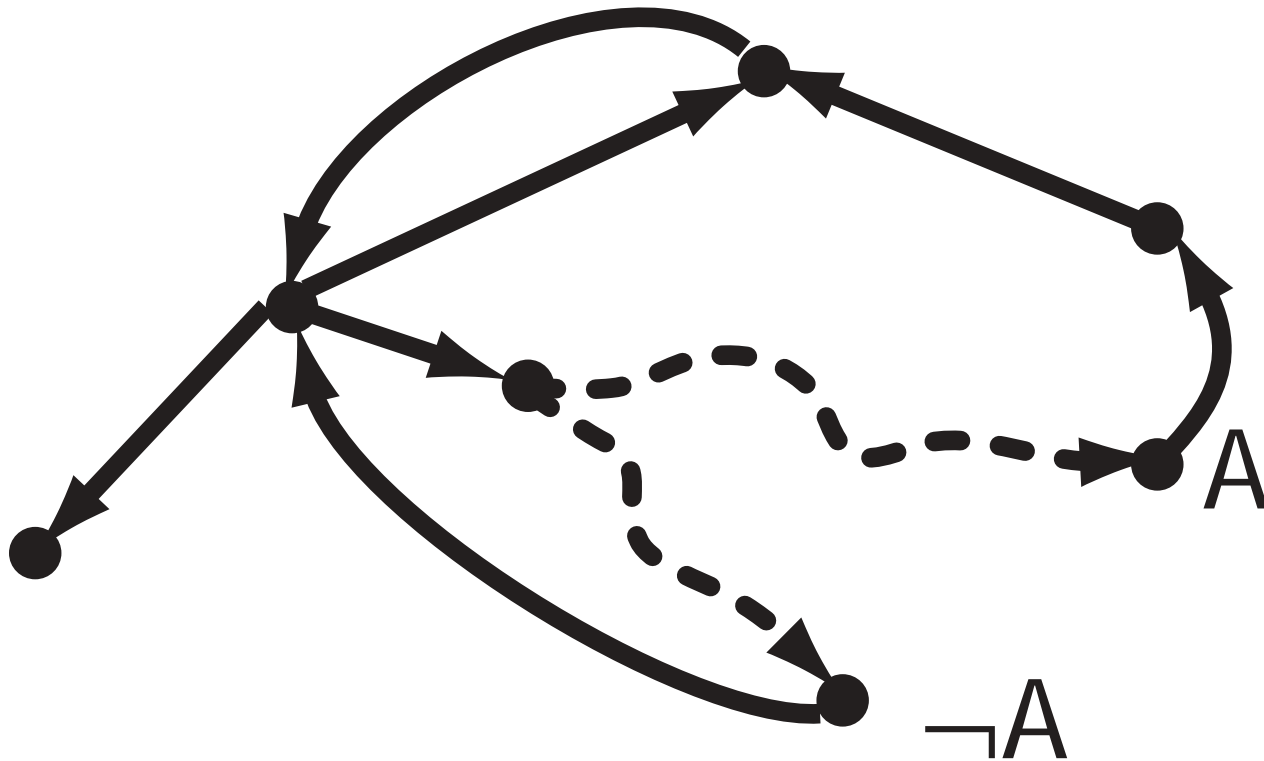


DFS algorithm

```
let rec search  $v$  =  
  mark  $v$  as old;  
   $dfs[v] \leftarrow count$ ;  
   $count \leftarrow count + 1$ ;  
   $low[v] \leftarrow dfs[v]$ ;  
  push  $v$  onto stack;  
  for  $w \in outedges[v]$  do  
    if  $w$  is new then  
      search  $w$ ;  
       $low[v] \leftarrow \min(low[v], low[w])$   
    else if  $dfs[w] < dfs[v]$  and  $w$  on stack then  
       $low[v] \leftarrow \min(dfs[w], low[v])$ ;
```



2SAT graph



2SAT algorithm

Theorem A 2SAT formula \mathcal{B} is satisfiable iff no pair of complementary literals appear in the same strongly-connected component of G .

Proof

- If they do, then $A \Leftrightarrow \neg A$, so \mathcal{B} is not satisfiable.
- Conversely
 - Collapse the strong components of G into single vertices; this new graph is acyclic
 - If A occurs in a strong component before a strong component for $\neg A$, assign $A = \perp$
 - Otherwise assign $A = \top$



2SAT algorithm

- Given formula B
 - Form the directed graph
 - Find the strongly-connected components
 - If any strongly-connected component contains a literal and its negation, report “not satisfiable”
 - Otherwise, build a satisfying assignment
 - Collapse strongly-connected components to get DAG
 - If A occurs before $\text{not } A$, then let $A = \text{false}$
 - If $\text{not } A$ occurs before A , then let $A = \text{true}$



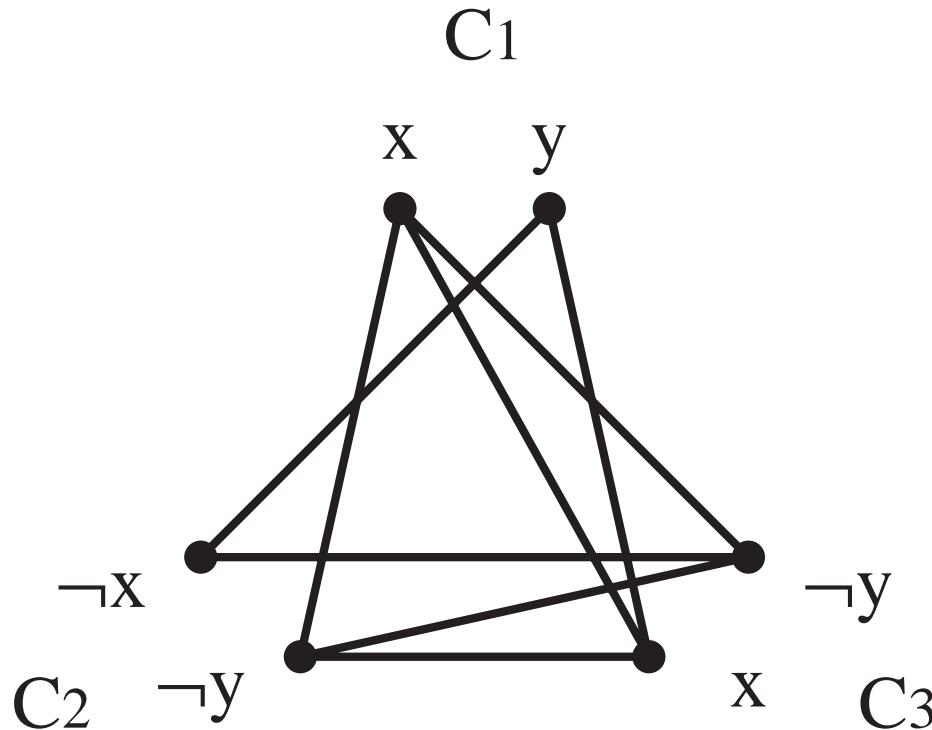
3SAT formulas

- 2SAT is easy
- Let's try to find an algorithm for 3SAT
- Consider a 3SAT formula $(l_1 \vee l_2 \vee l_3) \wedge \dots$
- Build a graph G with a vertex for each *occurrence* of a literal
- Add an edge between two literals if they are in different clauses, and they are not complementary



Conflict graph for 3SAT

$$\begin{array}{ccc} C_1 & C_2 & C_3 \\ (x \vee y) \wedge (\neg x \vee \neg y) \wedge (x \vee \neg y) \end{array}$$



Cliques and satisfiability

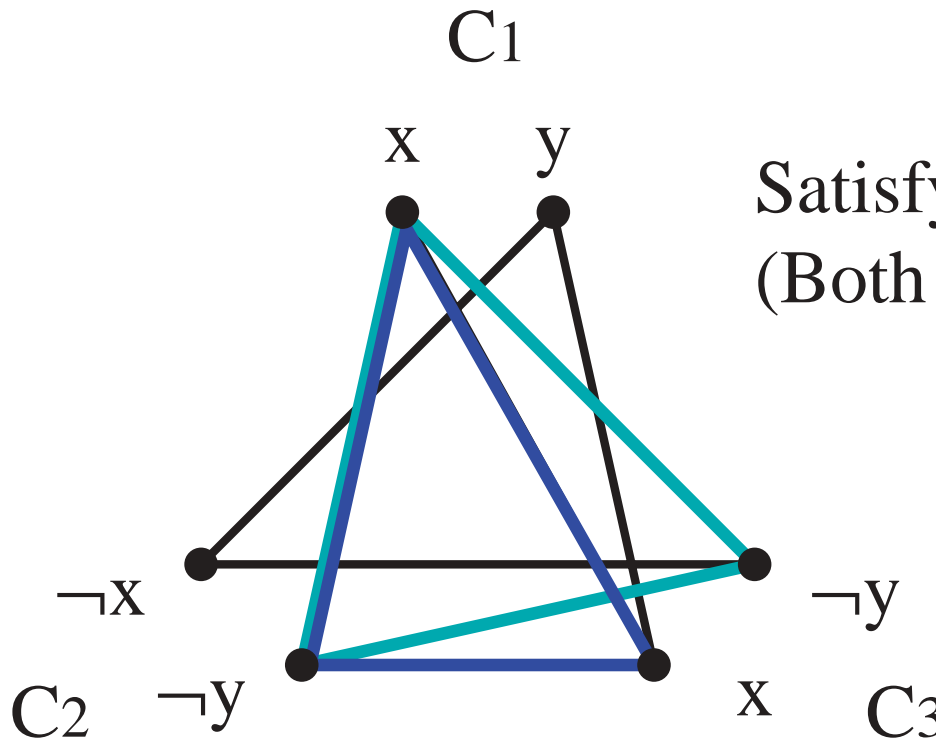
Definition A *k-clique* of graph G is a subgraph with k vertices that is full-connected.

Claim For any 3SAT formula \mathcal{B} with k clauses, the conflict graph G has a k -clique iff \mathcal{B} is satisfiable.



SAT/clique example

$$\begin{array}{ccc} C_1 & C_2 & C_3 \\ (x \vee y) \wedge (\neg x \vee \neg y) \wedge (x \vee \neg y) \end{array}$$



Satisfying assignment:
(Both cases): $(x \wedge \neg y)$



Satisfiable iff k -clique

Claim For any 3SAT formula \mathcal{B} with k clauses, the conflict graph G has a k -clique iff \mathcal{B} is satisfiable.

\Leftarrow Suppose \mathcal{B} is satisfiable.

- At least one literal in each clause must be assigned to \top .
- Choose one literal from each clause.
- The literals are all connected, so the graph has a k -clique.



Converse

⇒ Suppose G has a k -clique.

- It must have one vertex in each clause, since there are no edges within a clause.
- Assign each vertex to \top
- This sets one literal to \top in each clause, so this is a satisfying assignment.



Reductions

- We just *reduced* the SAT problem to the clique-finding problem
- If we have an efficient algorithm to find k-cliques, then we have an efficient algorithm for SAT
 - *Well, we don't know of efficient algorithms for either...*



Reductions

Definition Let $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ be decision problems. We write $A \leq_m^p B$, and say A *reduces to* B in *polynomial time* if there is a function $\sigma : \Sigma^* \rightarrow \Gamma^*$ such that

- σ is computable by a deterministic TM in poly time,
- For all $x \in \Sigma^*$,

$$x \in A \quad \text{iff} \quad \sigma(x) \in B$$

- We say $A \equiv_m^p B$ if $A \leq_m^p B$ and $B \leq_m^p A$



Hardness

Theorem If $A \leq_m^p B$, and B has a poly-time algorithm, then so does A

Proof Given an instance x of problem A , compute $\sigma(x)$ and determine if $\sigma(x) \in B$.

- Note that the size of $\sigma(x)$ is at most polynomial in the size of x (since $\sigma(x)$ runs in poly-time)
- The composition of two polynomials is still polynomial.



More NP problems: Independent set

Definition An *independent set* in an undirected graph $G = (V, E)$ is a subset $U \subseteq V$ such that $U^2 \cap E = \{\}$ (no two vertices of U are connected by an edge in E). The problem is, given a graph $G = (V, E)$ and $k \geq 0$, determine if G has an independent set with k or more vertices.

Reduction k -Independent-set \equiv_m^p k -Clique

- Build the complement $\bar{G} = (V, \bar{E})$

$$\bar{E} = \{(u, v) \mid u \neq v \wedge (u, v) \notin E\}$$

- The \bar{G} has a k -clique iff G has an independent set of size k .



More NP problems: graph coloring

Definition Consider an undirected graph $G = (V, E)$, and a finite set C . A *coloring* is a map $\varphi : V \rightarrow C$ such that $\varphi(u) \neq \varphi(v)$ for $(u, v) \in E$. Given G and a constant k , the *k-coloring problem* is to determine whether there exists a coloring using no more than k colors.

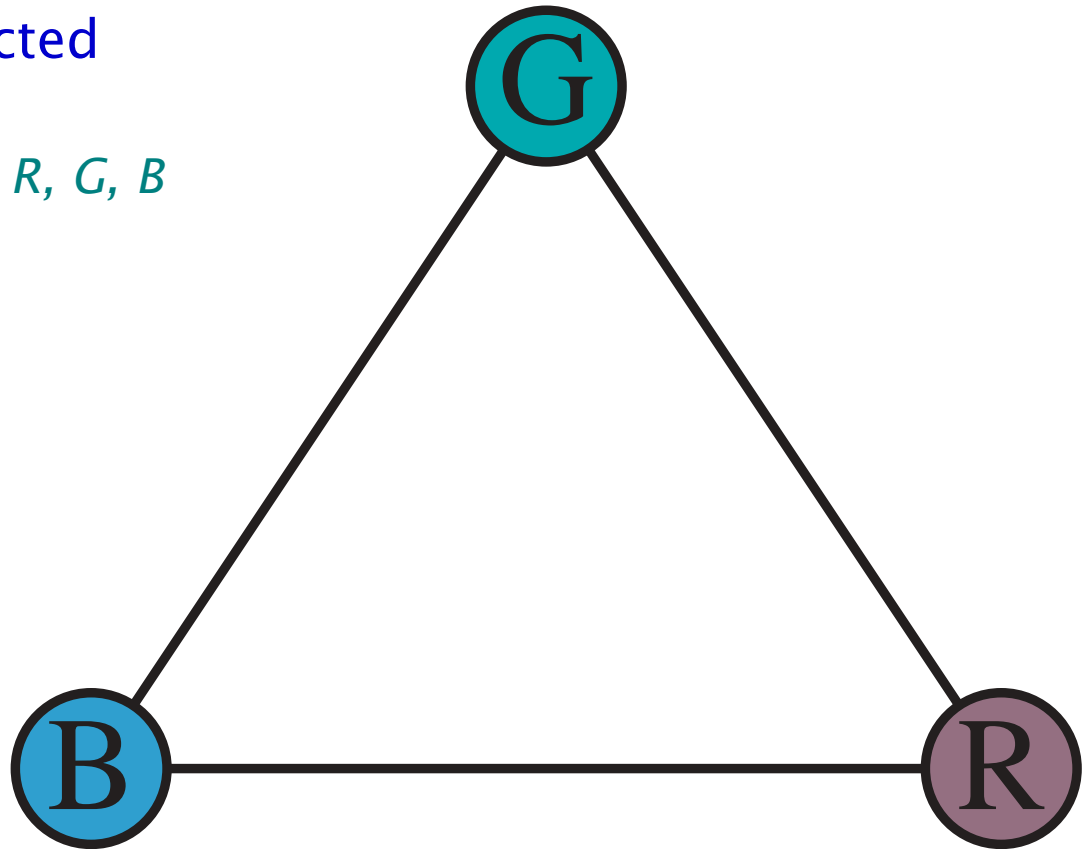
Reduction We show that coloring is hard by showing $\text{SAT} \leq_m^p \text{3-colorability}$.

- Use the colors R, G, B .
- Build a triangle for each literal.



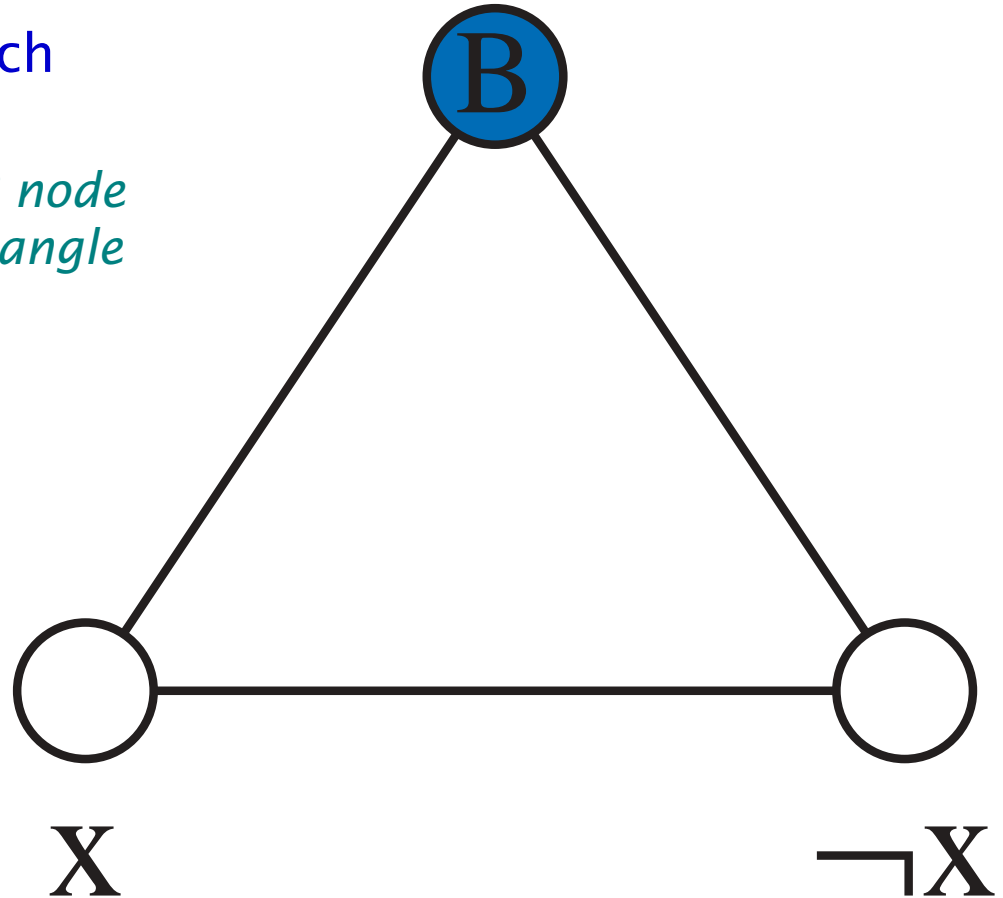
Graph coloring triangle

- Add a fully-connected triangle
 - *Call the vertices R, G, B*



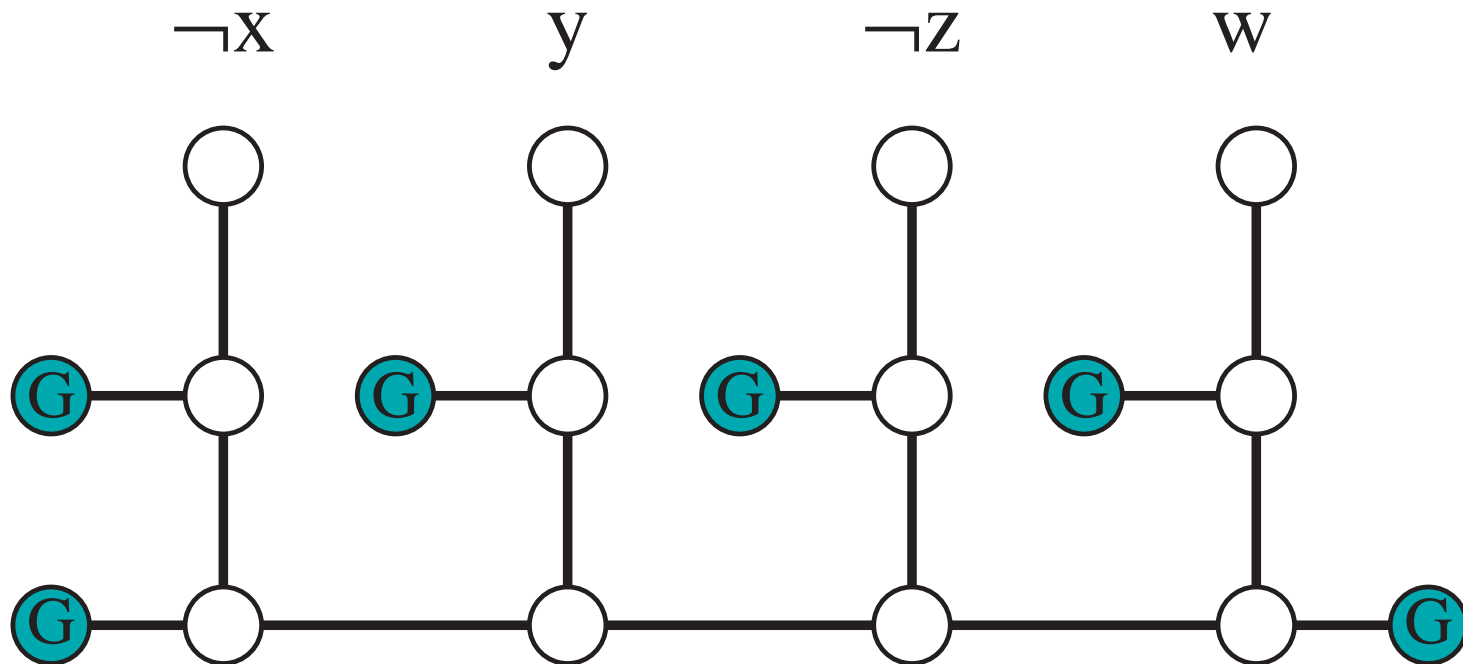
Literal triangle

- Add a triangle for each literal
 - *Upper node is the B node from the 3-color triangle*



Clause graph

- For each clause, add a graph of the following form
 - *The G nodes are all the same vertex from the RGB triangle*



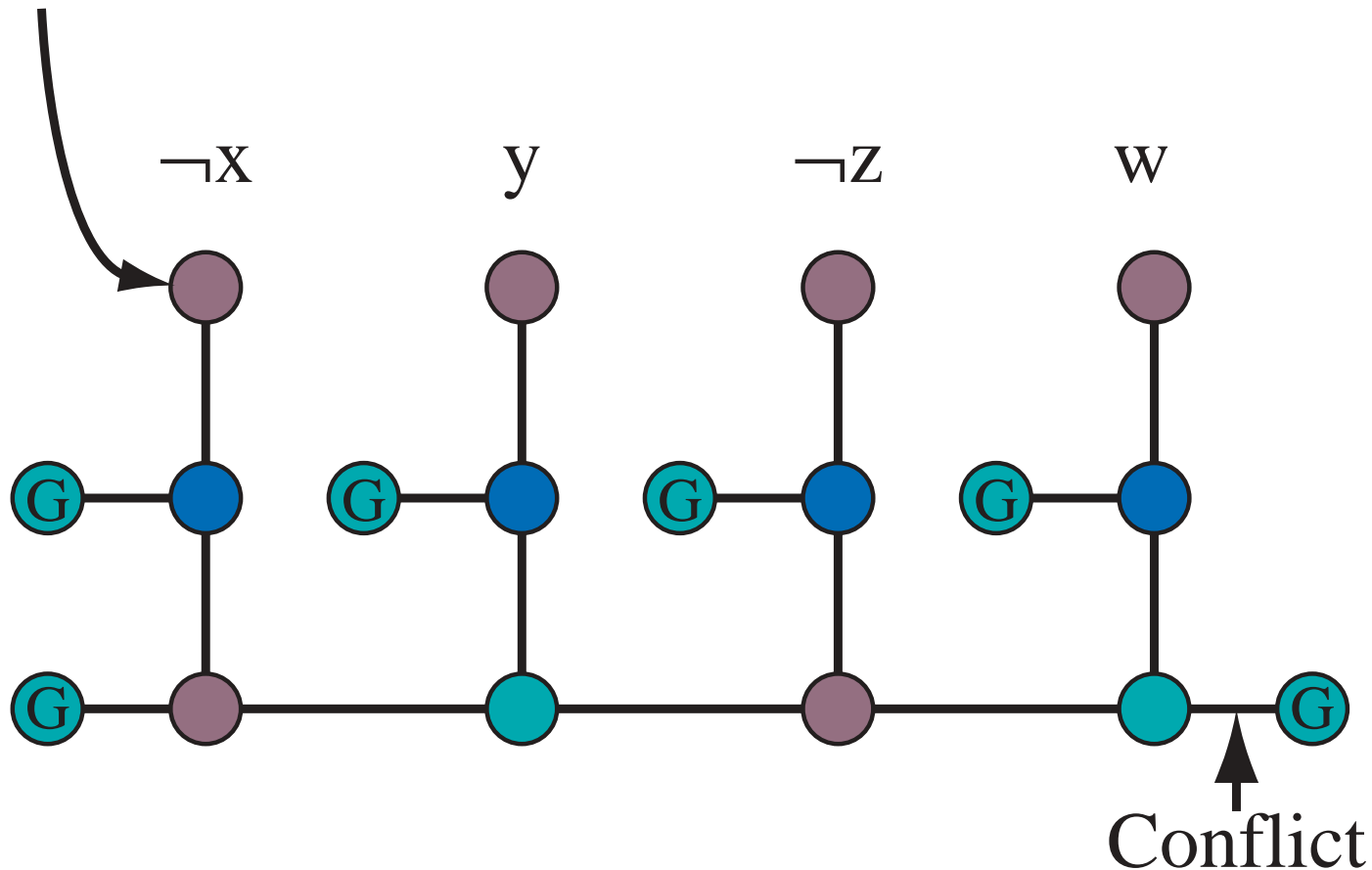
Coloring claim

- The constructed graph is colorable iff one literal can be colored green in each clause
 - *Intuitively, green represents true*
 - *Red represents false*



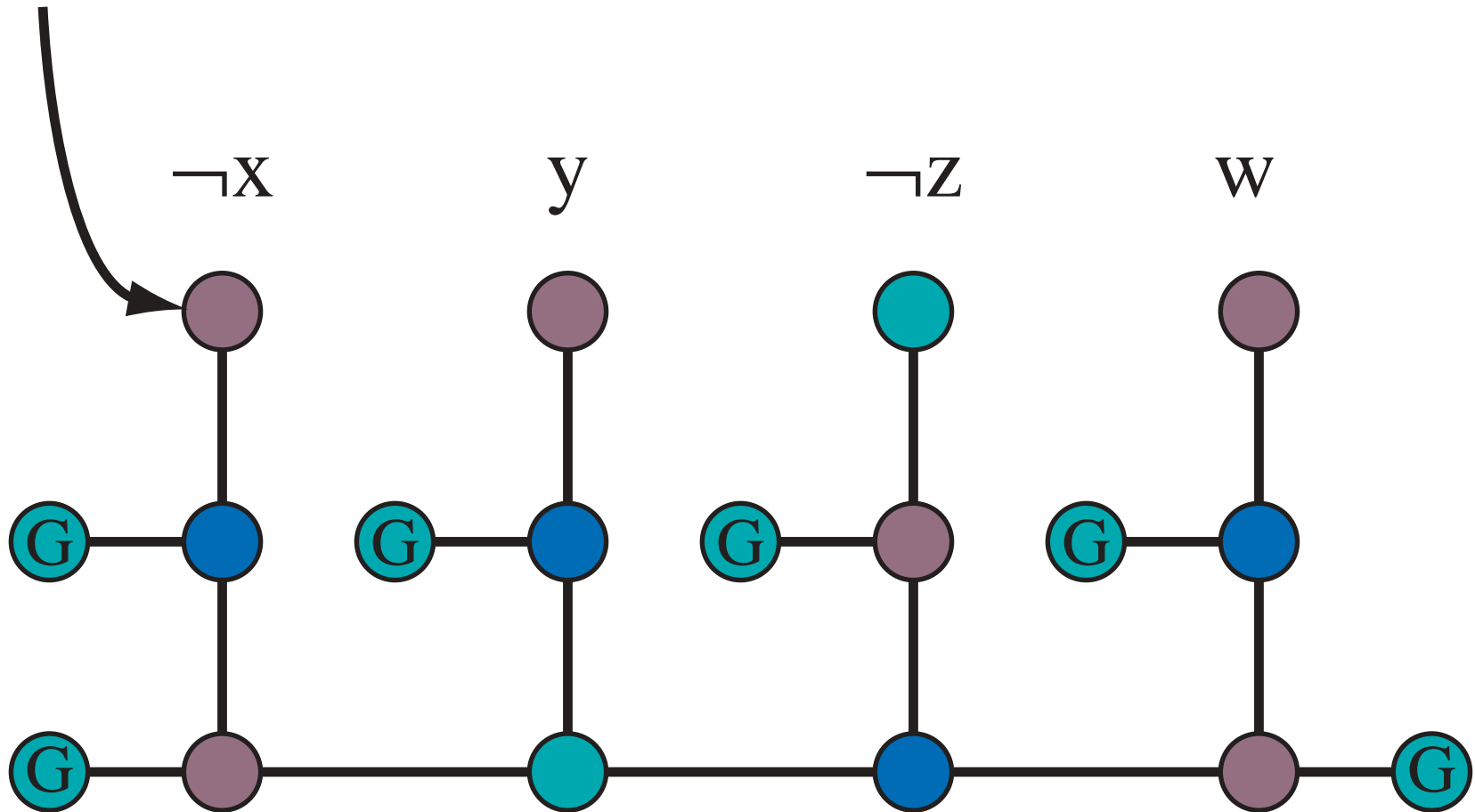
Unsatisfiable clause

Suppose top row is entirely red



Satisfiable clause

Suppose at least one is green



Some additional NP problems

Knapsack problem Given a finite set S , integer weight functions $w : S \rightarrow \mathbb{N}$, benefit function $b : S \rightarrow \mathbb{N}$, weight limit $W \in \mathbb{N}$, and desired benefit $B \in \mathbb{N}$, determine whether there is a subset $S' \subseteq S$ such that

$$\begin{aligned} \sum_{a \in S'} w(a) &\leq W \\ \sum_{a \in S'} b(a) &\geq B \end{aligned}$$



Subset Sum

Subset sum Given a finite set S , integer weight function $w : S \rightarrow \mathbb{N}$, and target $B : \mathbb{N}$, does there exist a subset $S' \subseteq S$ such that

$$\sum_{a \in S'} w(a) = B$$

Partition Given a finite set S and integer weight function $w : S \rightarrow \mathbb{N}$, does there exist a subset $S' \subseteq S$ such that

$$\sum_{a \in S'} w(a) = \sum_{a \in S - S'} w(a)$$



Reductions

- **Partition** → **SubsetSum** by taking $B = \frac{1}{2} \sum_{a \in S} w(a)$
- **SubsetSum** → **Partition**
 - Introduce two new elements of weight $N - B$ and weight $N - (\Sigma - B)$, where N is large

$$\Sigma = \sum_{a \in S} w(a)$$

- Ask whether the new set can be partitioned into two sets of equal weight.
- **Partition** → **Knapsack** take $b = w$ and $W = B = \frac{1}{2} \Sigma$



NP reductions

- Reductions are a fundamental tool
 - *If A reduces to B, then if B can be solved efficiently, so can A*
- Next up
 - *Graph theory*
 - *NP problems in graph theory*
 - *Ptime reductions*
 - *NP-completeness*

