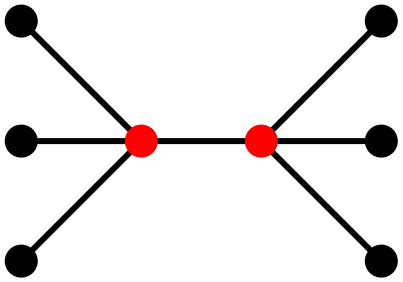
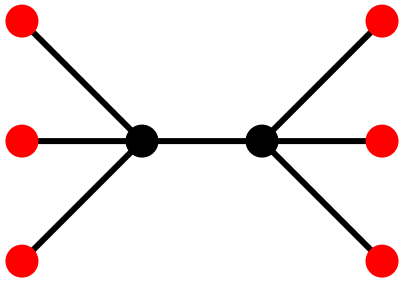
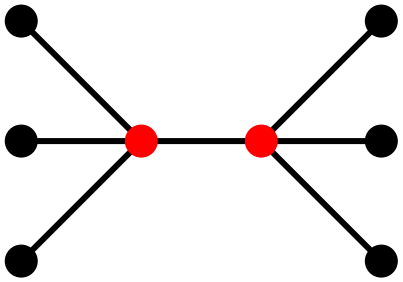
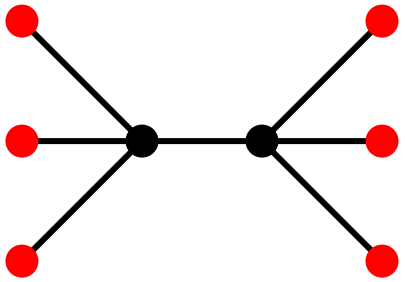
Parameterized complexity

	Vertex Cover	Independent Set
Problem:		
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?
		
Complexity:	NP-complete	NP-complete

Parameterized complexity

	Vertex Cover	Independent Set
Problem:		
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?

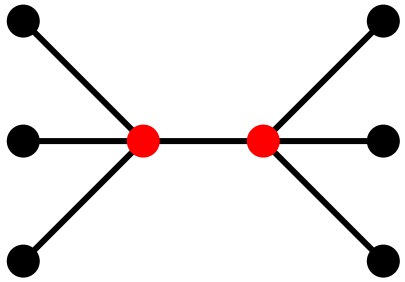


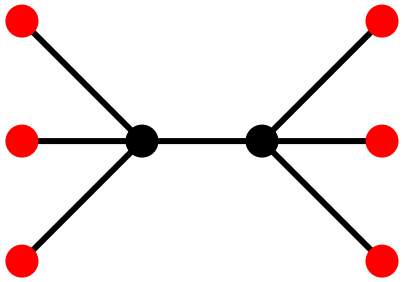


Complexity:	NP-complete	NP-complete
Brute force:	$O(n^k)$ possibilities	$O(n^k)$ possibilities

Parameterized complexity

	Vertex Cover	Independent Set
Problem:		
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?





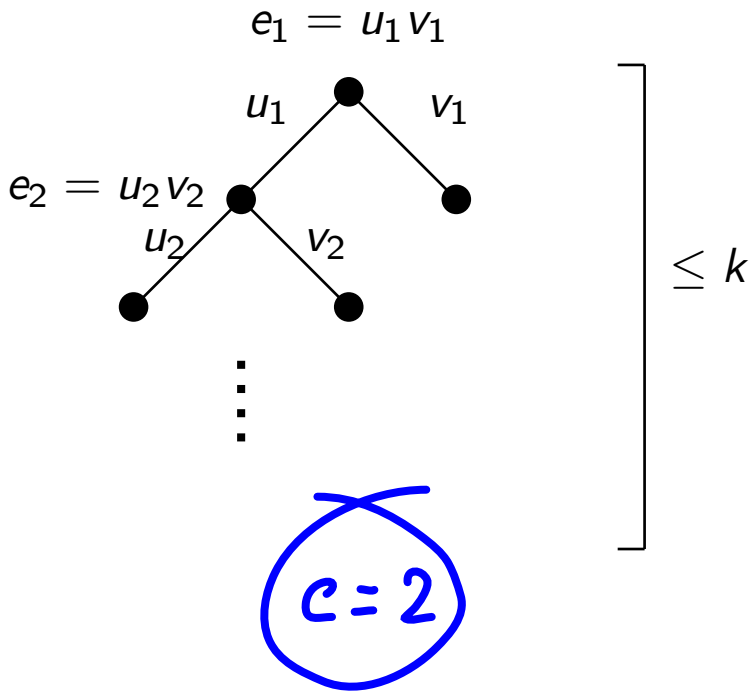
Complexity:	NP-complete	NP-complete
Brute force:	$O(n^k)$ possibilities	$O(n^k)$ possibilities
	$O(2^k n^2)$ **algorithm exists** ☺	**No** $n^{o(k)}$ **algorithm known** ☹

Bounded search tree method

Bounded search tree method

Algorithm for Vertex Cover

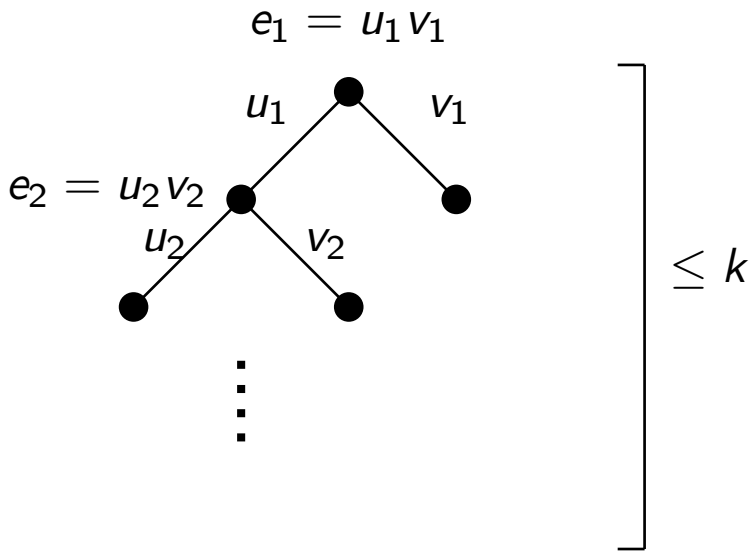
- **Main idea:** reduce problem instance (x, k) to solving a bounded number of instances with parameter $< k$.
- We should be able to solve instance (x, k) in polynomial time using the solutions of the new instances.
- If the parameter strictly decreases in every recursive call, then the depth is at most k.
- Size of the search tree:
  - If we branch into c directions: c^k.
  - If we branch into $O(k)$ directions: $k^{O(k)} = 2^{O(k \log k)}$.
  - (If we branch into $O(\log n)$ directions: $O(n) + 2^{O(k \log k)}$.)



Bounded search tree method

Algorithm for Vertex Cover

- **Main idea:** reduce problem instance (x, k) to solving a bounded number of instances with parameter $< k$.
- We should be able to solve instance (x, k) in polynomial time using the solutions of the new instances.
- If the parameter strictly decreases in every recursive call, then the depth is at most k.
- Size of the search tree:
  - If we branch into c directions: c^k.
  - If we branch into $O(k)$ directions: $k^{O(k)} = 2^{O(k \log k)}$.
  - (If we branch into $O(\log n)$ directions: $O(n) + 2^{O(k \log k)}$.)

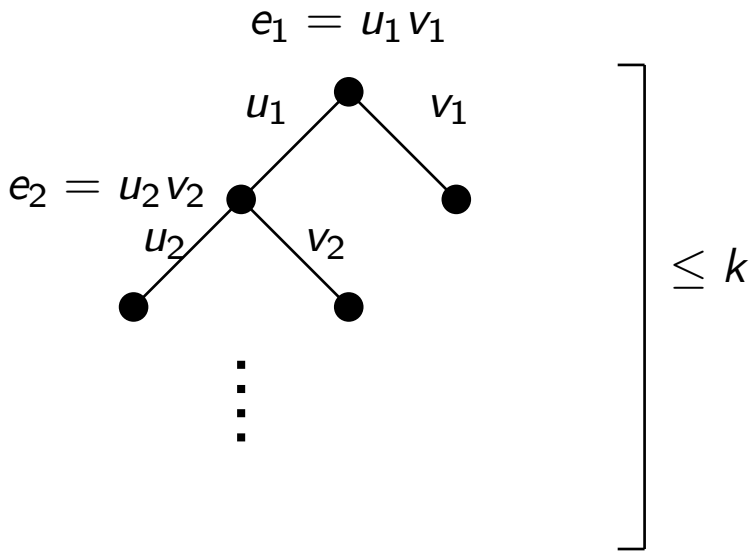


Next: A $1.41^k \cdot n^{O(1)}$ time algorithm for Vertex Cover.

Bounded search tree method

Algorithm for Vertex Cover

- **Main idea:** reduce problem instance (x, k) to solving a bounded number of instances with parameter $< k$.
- We should be able to solve instance (x, k) in polynomial time using the solutions of the new instances.
- If the parameter strictly decreases in every recursive call, then the depth is at most k.
- Size of the search tree:
  - If we branch into c directions: c^k.
  - If we branch into $O(k)$ directions: $k^{O(k)} = 2^{O(k \log k)}$.
  - (If we branch into $O(\log n)$ directions: $O(n) + 2^{O(k \log k)}$.)



Next: A $O^*(1.41^k)$ time algorithm for Vertex Cover.

Improved branching for Vertex Cover

Each component is a path or a cycle!

- If every vertex has degree ≤ 2, then the problem can be solved in polynomial time.
- **Branching rule:**
  If there is a vertex v with at least 3 neighbors, then
  - either v is in the solution,
  - or every neighbor of v is in the solution.

Crude upper bound: $O^*(2^k)$, since the branching rule decreases the parameter.

Improved branching for Vertex Cover

- If every vertex has degree ≤ 2, then the problem can be solved in polynomial time.
- **Branching rule:**
  If there is a vertex v with at least 3 neighbors, then
  - either v is in the solution, $\Rightarrow$ k decreases by 1
  - or every neighbor of v is in the solution. $\Rightarrow$ k decreases by at least 3

Crude upper bound: $O^*(2^k)$, since the branching rule decreases the parameter.

But it is somewhat better than that, since in the second branch, the parameter decreases by at least 3.

Better analysis

Let $T(k)$ be the maximum number of leaves of the search tree if the parameter is at most k (let $T(k) = 1$ for $k \leq 0$).

$$T(k) \leq T(k-1) + T(k-3)$$

There is a standard technique for bounding such functions asymptotically.

Better analysis

Let $T(k)$ be the maximum number of leaves of the search tree if the parameter is at most k (let $T(k) = 1$ for $k \leq 0$).

$$T(k) \leq T(k-1) + T(k-3)$$

There is a standard technique for bounding such functions asymptotically.

We prove by induction that $T(k) \leq c^k$ for some $c > 1$ as small as possible.

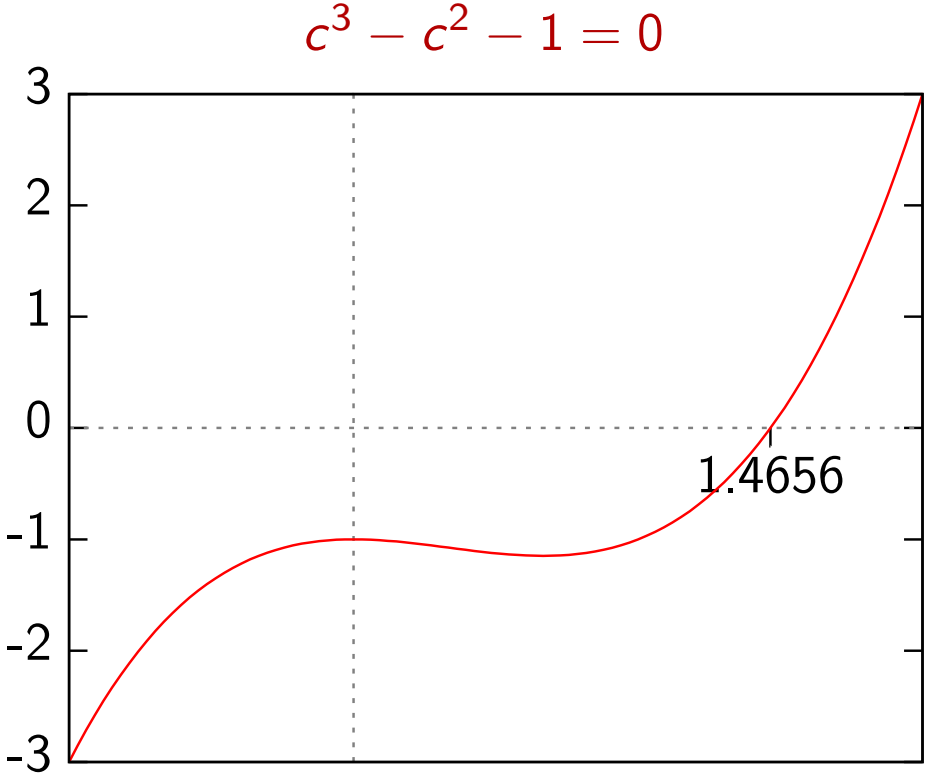
What values of c are good? We need:

$$c^k \geq c^{k-1} + c^{k-3}$$
$$c^3 - c^2 - 1 \geq 0$$

We need to find the roots of the **characteristic equation** $c^3 - c^2 - 1 = 0$.

Note: it is always true that such an equation has a unique positive root.

Better analysis



$c = 1.4656$ is a good value $\Rightarrow$ $T(k) \leq 1.4656^k$

$\Rightarrow$ We have a $O^*(1.4656^k)$ algorithm for Vertex Cover.

Better analysis

We showed that if $T(k) \leq T(k-1) + T(k-3)$, then $T(k) \leq 1.4656^k$ holds.

Is this bound tight? There are two questions:

- Can the function $T(k)$ be that large?
  Yes (ignoring rounding problems).
- Can the search tree of the Vertex Cover algorithm be that large?
  Difficult question, hard to answer in general.

Branching vectors

The **branching vector** of our $O^*(1.4656^k)$ Vertex Cover algorithm was $(1, 3)$.

Example: Let us bound the search tree for the branching vector $(2, 5, 6, 6, 7, 7)$.
(2 out of the 6 branches decrease the parameter by 7, etc.).

Branching vectors

The **branching vector** of our $O^*(1.4656^k)$ Vertex Cover algorithm was $(1, 3)$.

Example: Let us bound the search tree for the branching vector $(2, 5, 6, 6, 7, 7)$.
(2 out of the 6 branches decrease the parameter by 7, etc.).

The value $c > 1$ has to satisfy:

$$c^k \geq c^{k-2} + c^{k-5} + 2c^{k-6} + 2c^{k-7}$$
$$c^7 - c^5 - c^2 - 2c - 2 \geq 0$$

Unique positive root of the characteristic equation: $1.4483 \Rightarrow T(k) \leq 1.4483^k$.

It is hard to compare branching vectors intuitively.

Branching vectors

Example: The roots for branching vector (i,j) $(1 \leq i,j \leq 6)$.

$$T(k) \leq T(k-i) + T(k-j) \Rightarrow c^k \geq c^{k-i} + c^{k-j}$$
$$c^j - c^{j-i} - 1 \geq 0$$

We compute the unique positive root.

	1	2	3	4	5	6
1	2.0000	1.6181	1.4656	1.3803	1.3248	1.2852
2	1.6181	1.4143	1.3248	1.2721	1.2366	1.2107
3	1.4656	1.3248	1.2560	1.2208	1.1939	1.1740
4	1.3803	1.2721	1.2208	1.1893	1.1674	1.1510
5	1.3248	1.2366	1.1939	1.1674	1.1487	1.1348
6	1.2852	1.2107	1.1740	1.1510	1.1348	1.1225

Example: Triangle Free Deletion

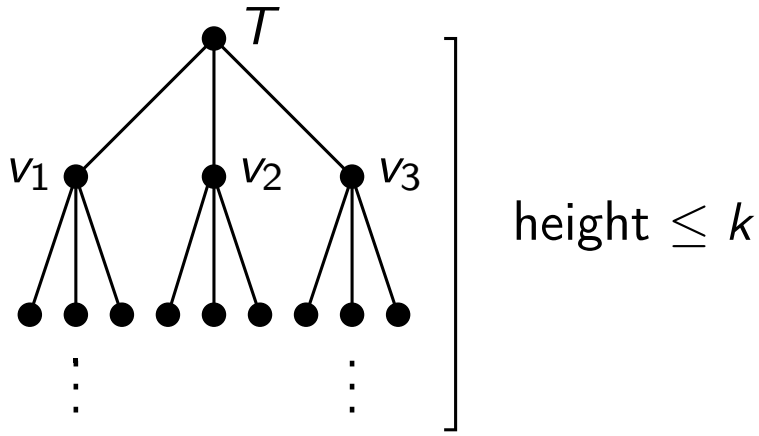
Triangle Free Deletion
Given (G, k), remove at most k vertices to make the graph triangle free.

What is the running time of a simple branching algorithm?

Example: Triangle Free Deletion

Triangle Free Deletion
Given (G, k), remove at most k vertices to make the graph triangle free.

What is the running time of a simple branching algorithm?



The search tree has at most 3^k leaves and the work to be done is polynomial at each step $\Rightarrow O^*(3^k)$ time algorithm.

Note: If the answer is "NO", then the search tree has **exactly** 3^k leaves.

Revisit: How many leaves in a NO instance of VERTEX COVER when the branch vector is (1,1)? How about when it is (1,3)?

Graph modification problems

A general problem family containing tasks of the following type:

Given (G, k), do at most k allowed operations on G to make it have property $\mathcal{P}$.

- Allowed operations: vertex deletion, edge deletion, edge addition, ...
- Property $\mathcal{P}$: edgeless, no triangles, no cycles, planar, chordal, regular, disconnected, ...

Examples:

- Vertex Cover: Delete k vertices to make G edgeless.
- Triangle Free Deletion: Delete k vertices to make G triangle free.
- Feedback Vertex Set: Delete k vertices to make G acyclic (forest).

Hereditary properties

Definition

A graph property $\mathcal{P}$ is **hereditary** or **closed under induced subgraphs** if whenever $G \in P$, every induced subgraph of G is also in $\mathcal{P}$.

"removing a vertex does not ruin the property"
(e.g., triangle free, bipartite, planar)

Hereditary properties

Definition

A graph property $\mathcal{P}$ is **hereditary** or **closed under induced subgraphs** if whenever $G \in P$, every induced subgraph of G is also in $\mathcal{P}$.

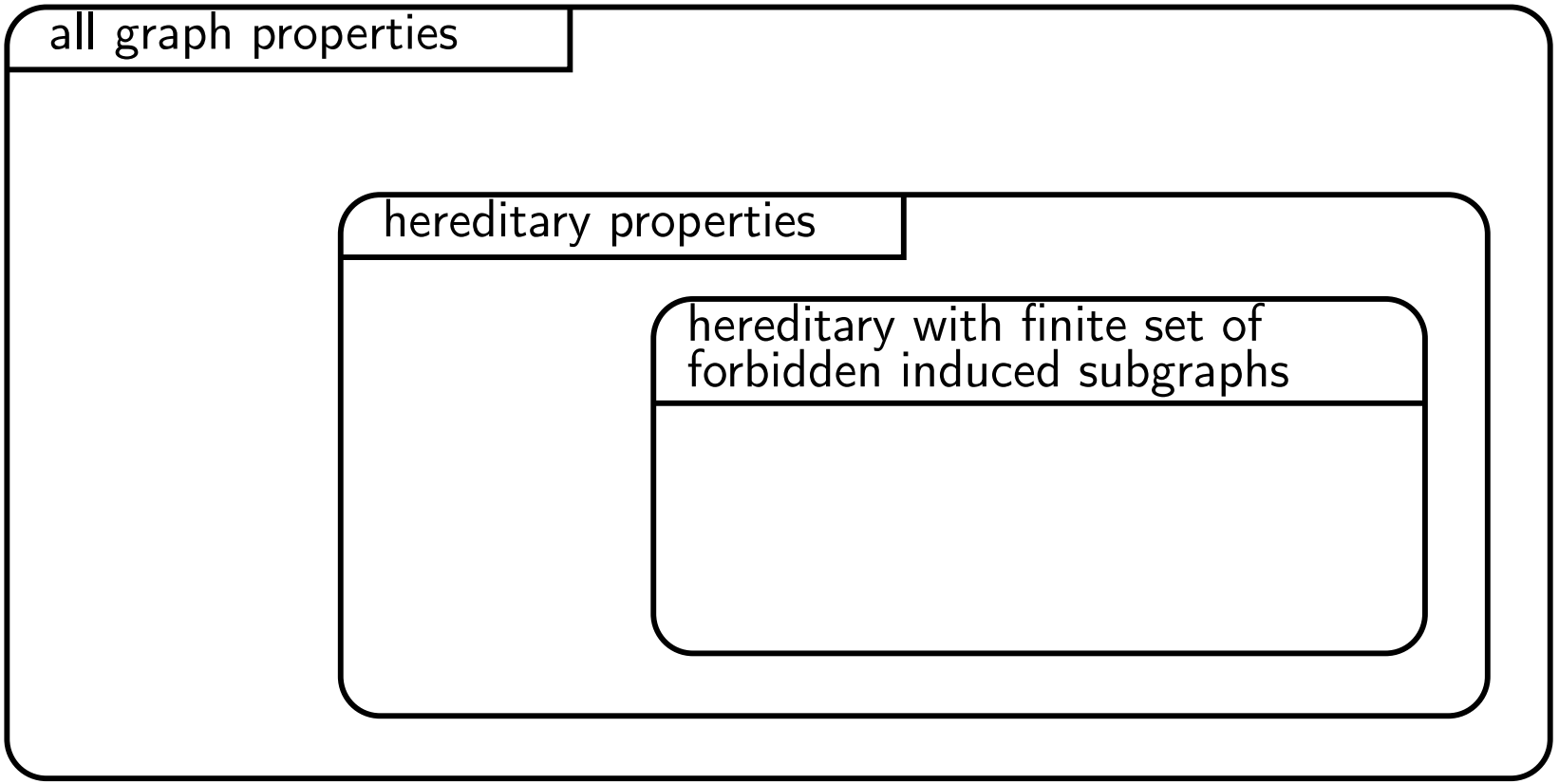
"removing a vertex does not ruin the property"
(e.g., triangle free, bipartite, planar)

Observation

Every hereditary property $\mathcal{P}$ can be characterized by a (finite or infinite) set $\mathcal{F}$ of "minimal bad graphs" or "forbidden induced subgraphs": $G \in \mathcal{P}$ if and only if G **does not** have an induced subgraph isomorphic to a member of $\mathcal{F}$.

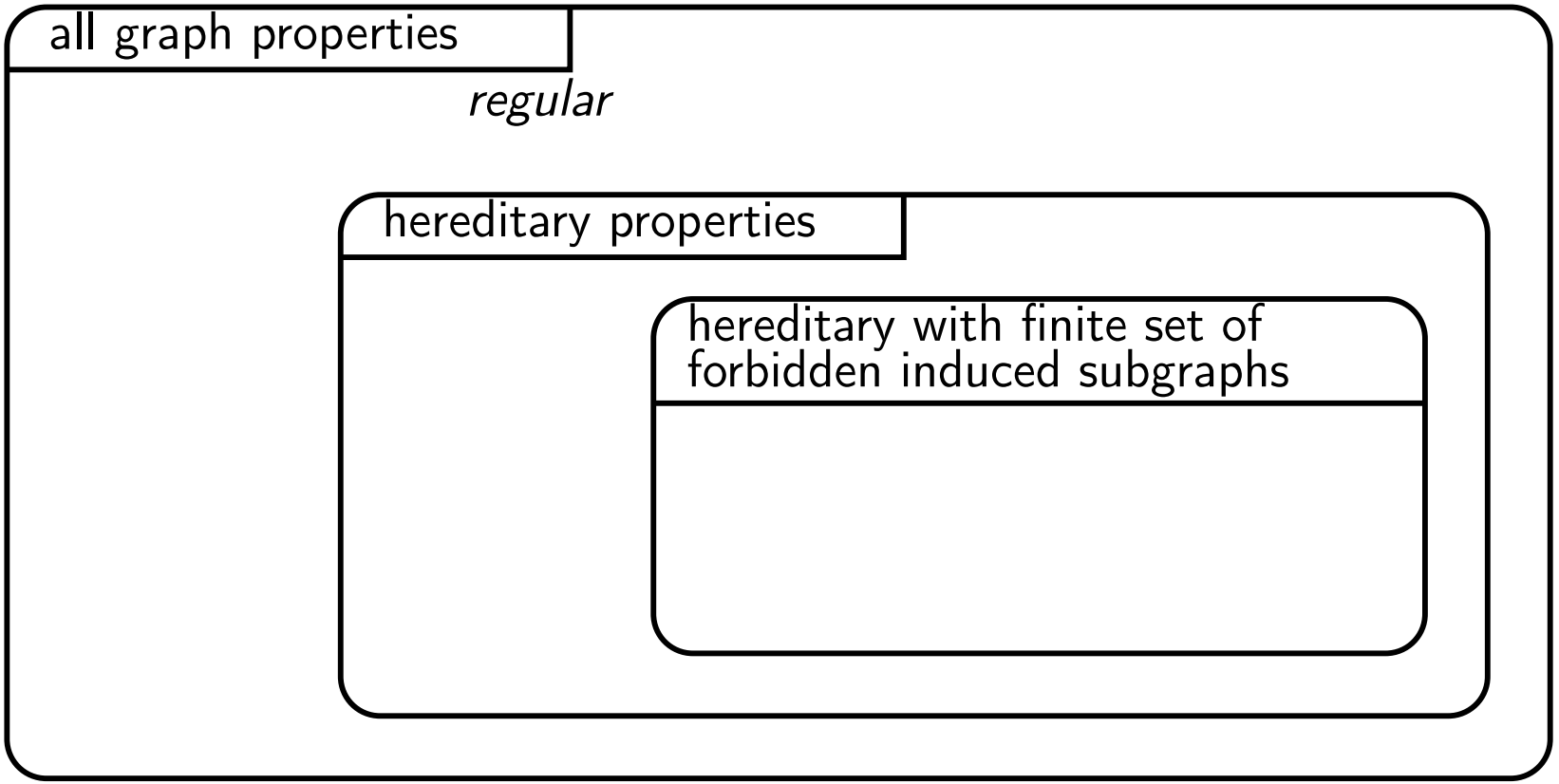
Example: a graph is bipartite if and only if it does not contain an odd cycle as an induced subgraph.

Graph properties

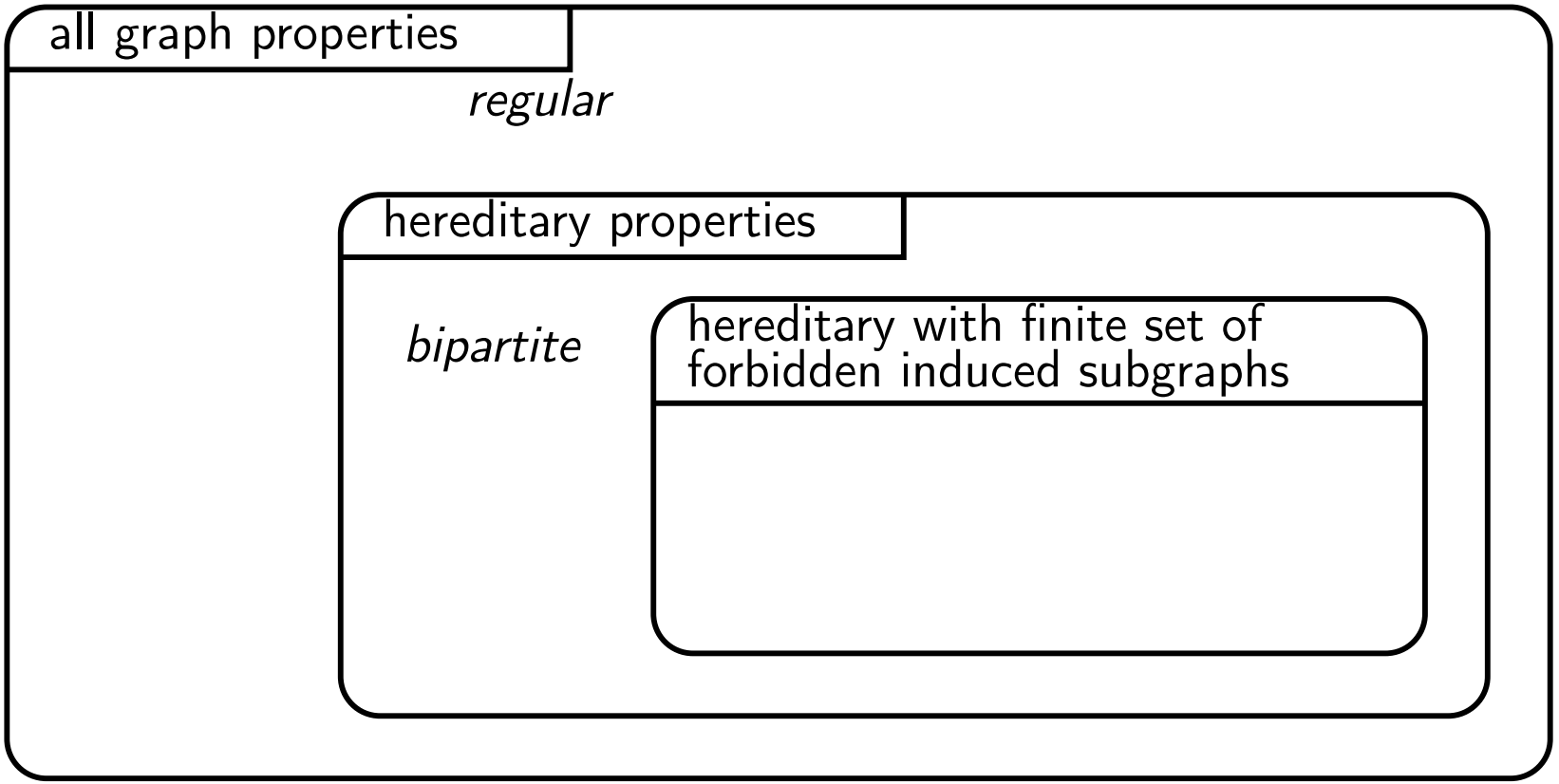


regular *bipartite* *triangle free* *connected*
planar *empty* *complete* *acyclic*

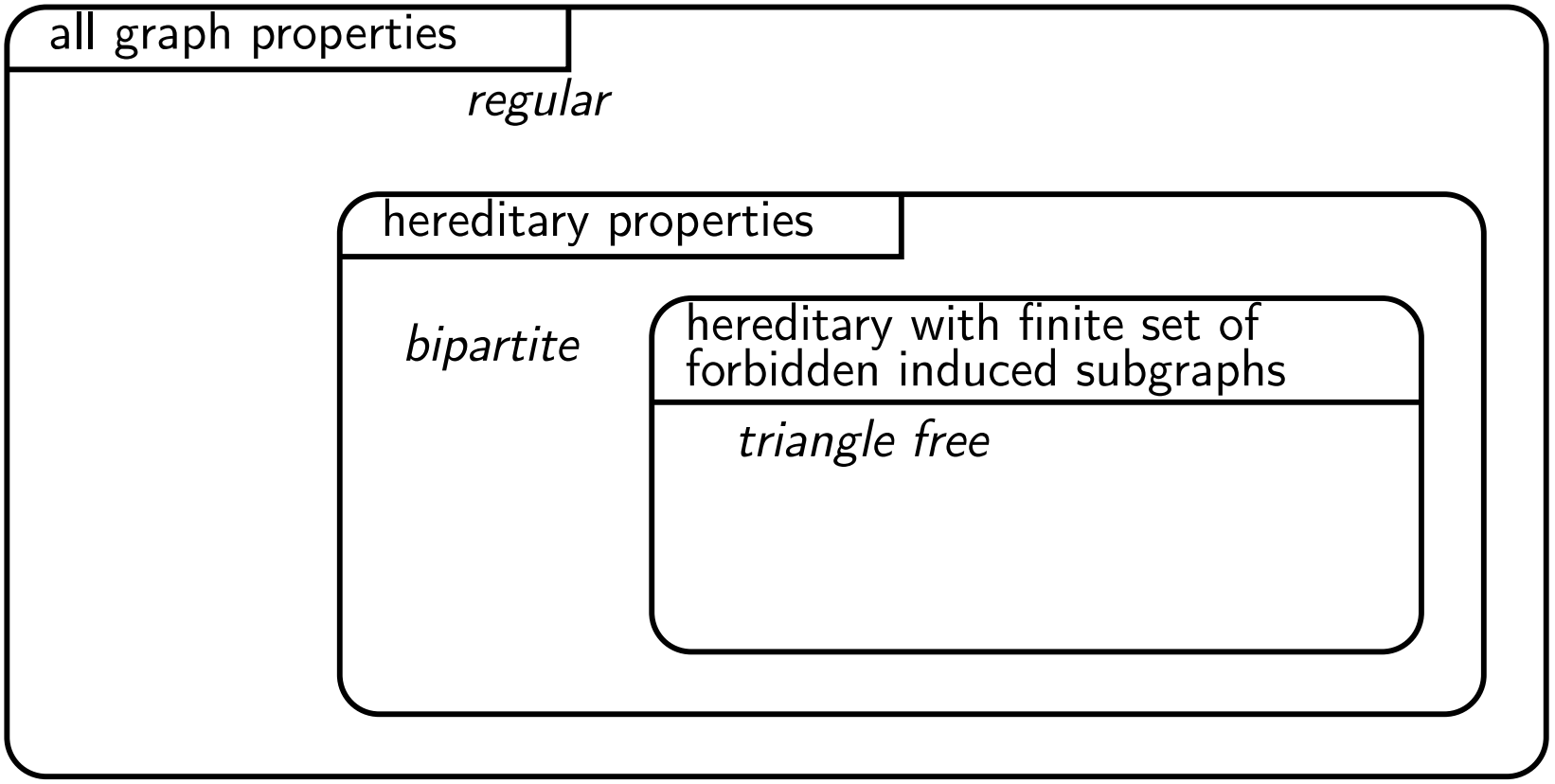
Graph properties



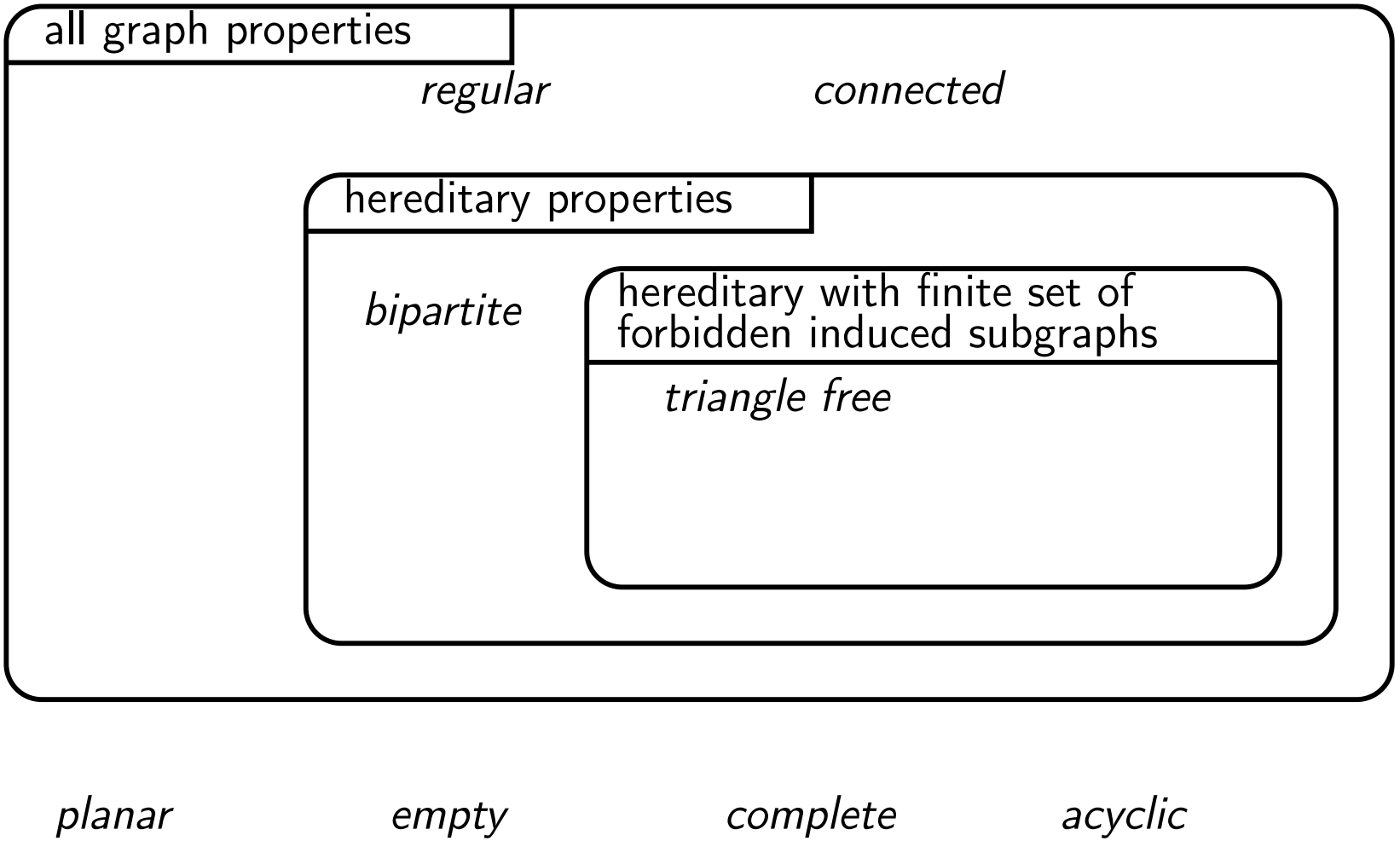
Graph properties



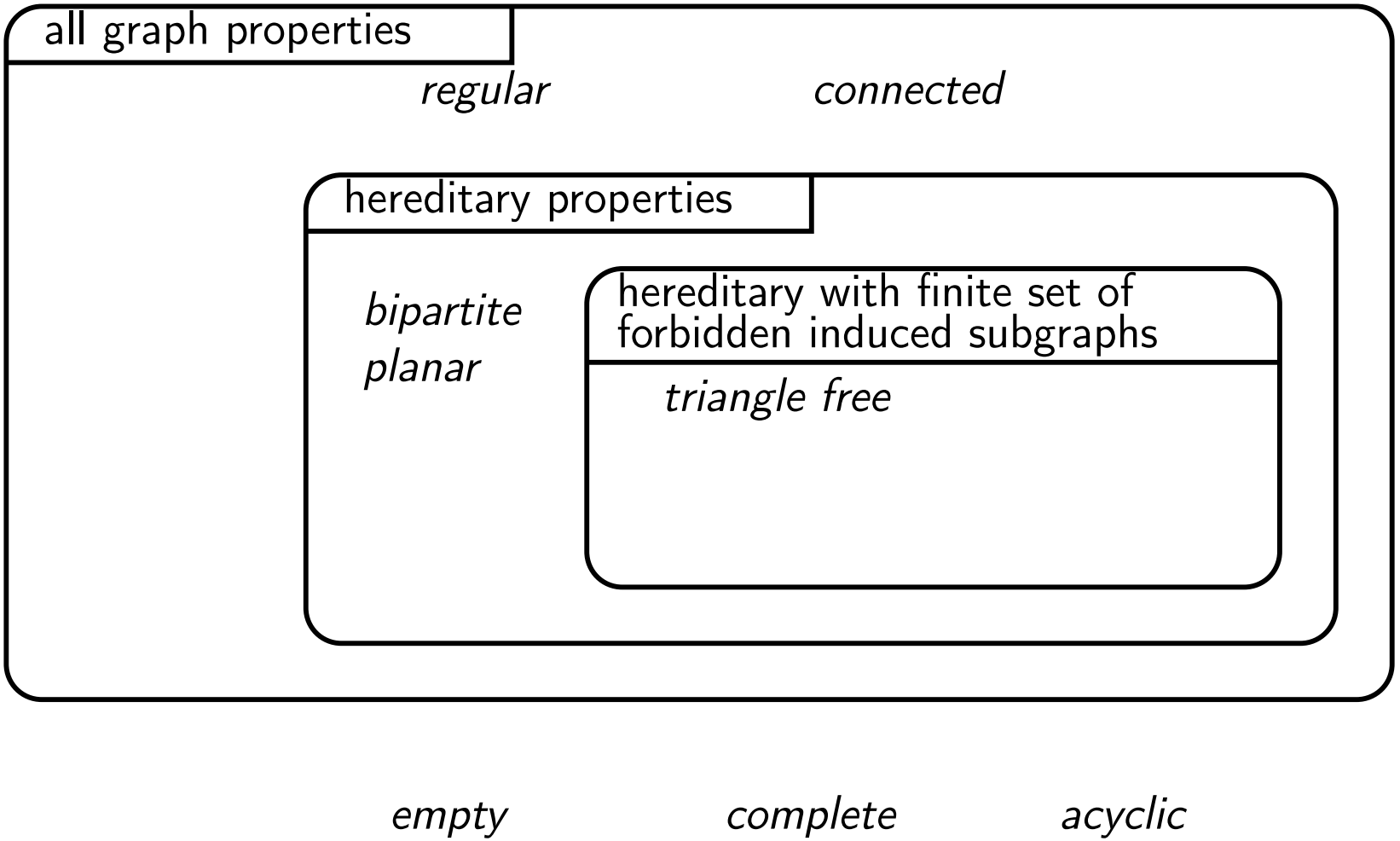
Graph properties



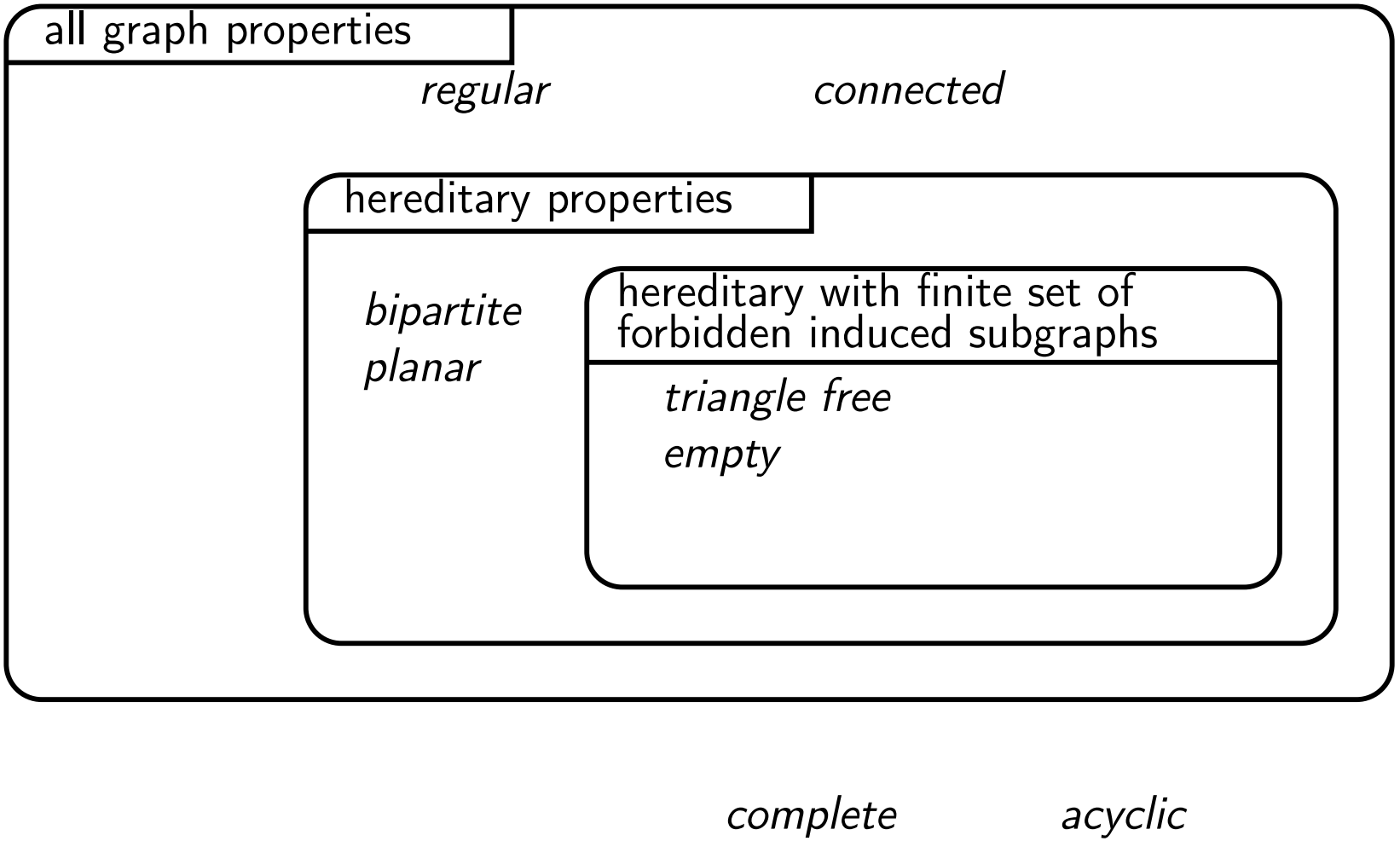
Graph properties



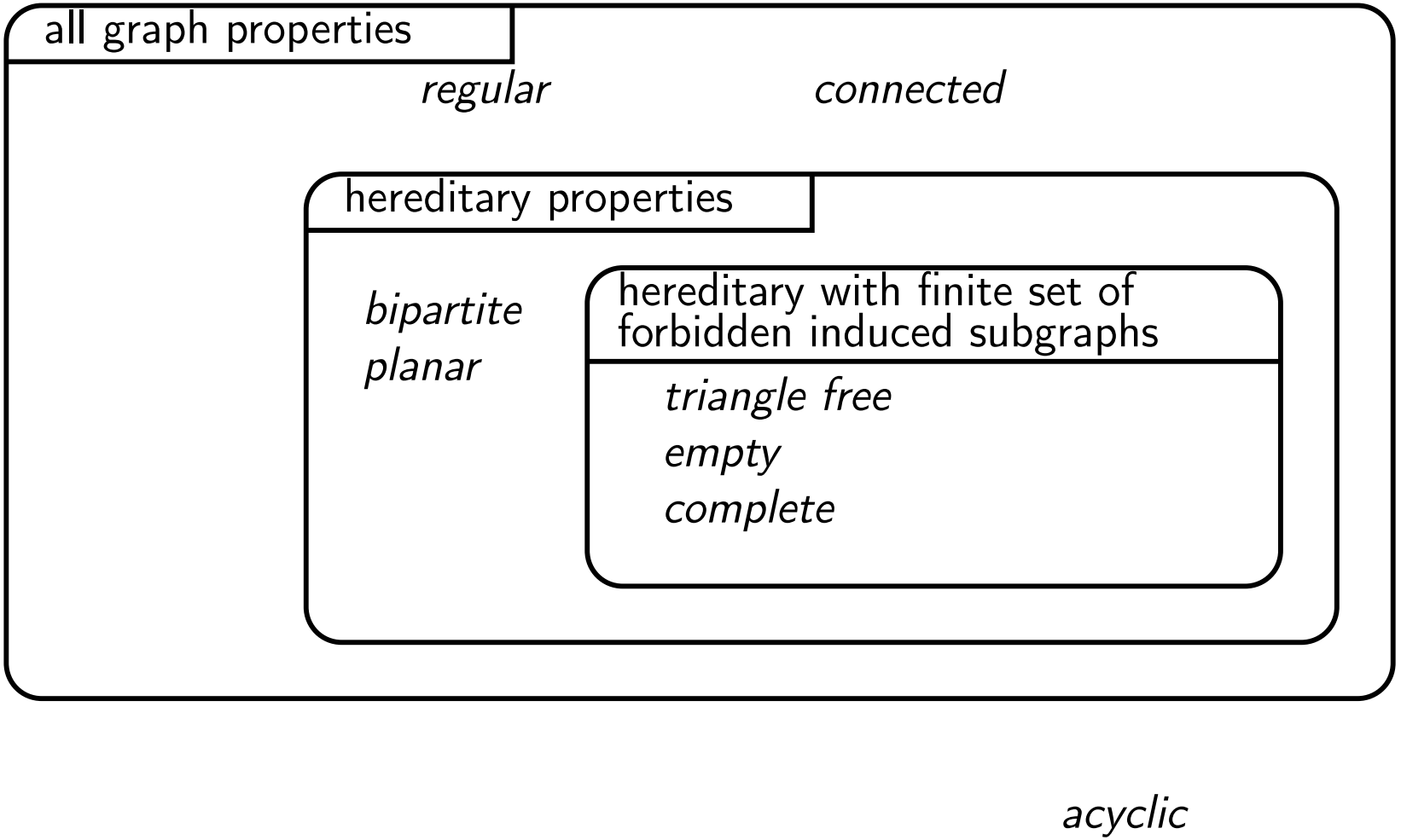
Graph properties



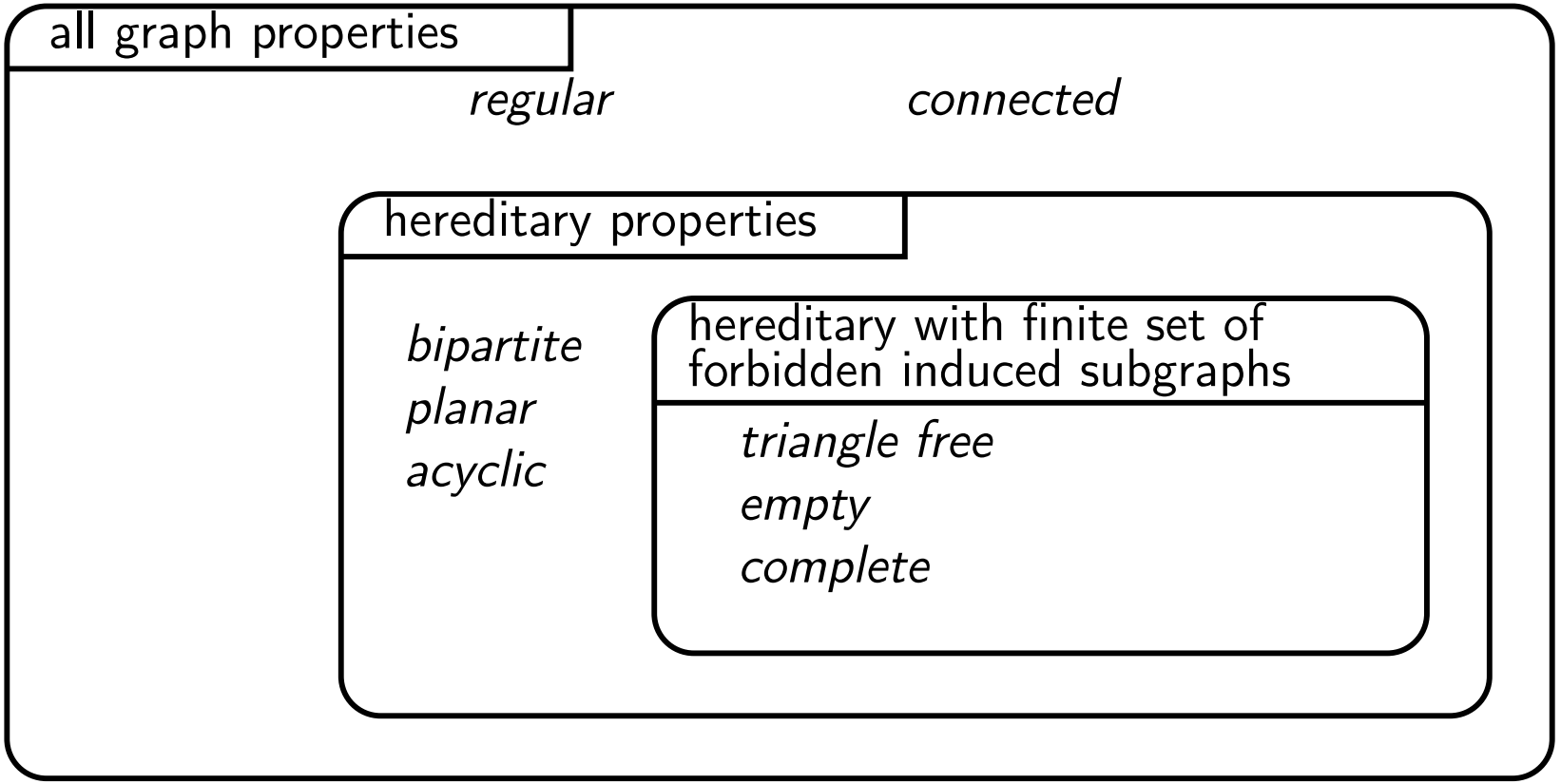
Graph properties



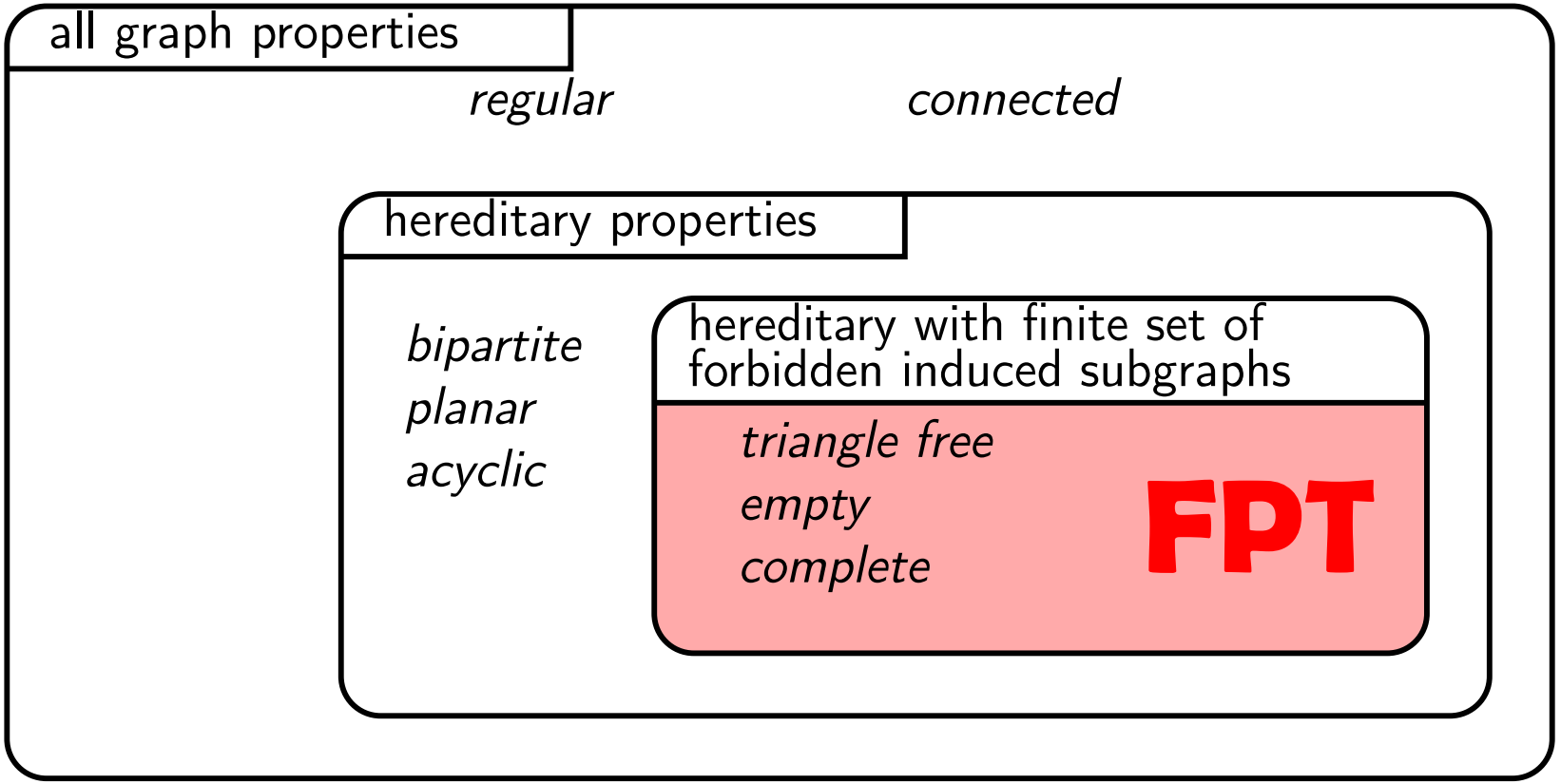
Graph properties



Graph properties



Graph properties



Using finite obstructions

Theorem

If $\mathcal{P}$ is hereditary and can be characterized by a **finite** set $\mathcal{F}$ of forbidden induced subgraphs, then the graph modification problems corresponding to $\mathcal{P}$ are FPT.

Proof:

- Suppose that every graph in $\mathcal{F}$ has at most r vertices. Using brute force, we can find in time $O(n^r)$ a forbidden subgraph (if exists).
- If a forbidden subgraph exists, then we have to delete one of the at most r vertices or add/delete one of the at most $\binom{r}{2}$ edges
  $\Rightarrow$ Branching factor is a constant c depending on $\mathcal{F}$.
- The search tree has at most c^k leaves and the work to be done at each node is $O(n^r)$.

Graph modification problems

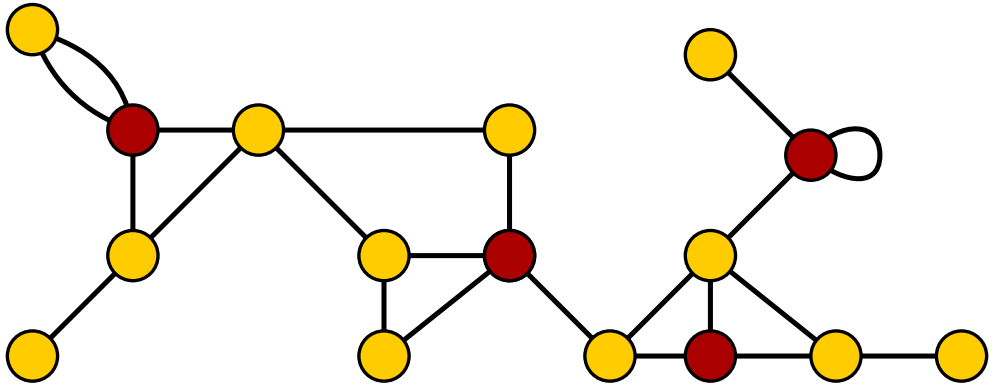
A very wide and active research area in parameterized algorithms.

- If the set of forbidden subgraphs is finite, then the problem is immediately FPT (e.g., Vertex Cover, Triangle Free Deletion). Here the challenge is improving the naive running time.
- If the set of forbidden subgraphs is infinite, then very different techniques are needed to show that the problem is FPT (e.g., Feedback Vertex Set, Bipartite Deletion, Planar Deletion).

Feedback Vertex Set

Feedback Vertex Set:
Given (G, k), find a set S of at most k vertices such that $G - S$ has no cycles.

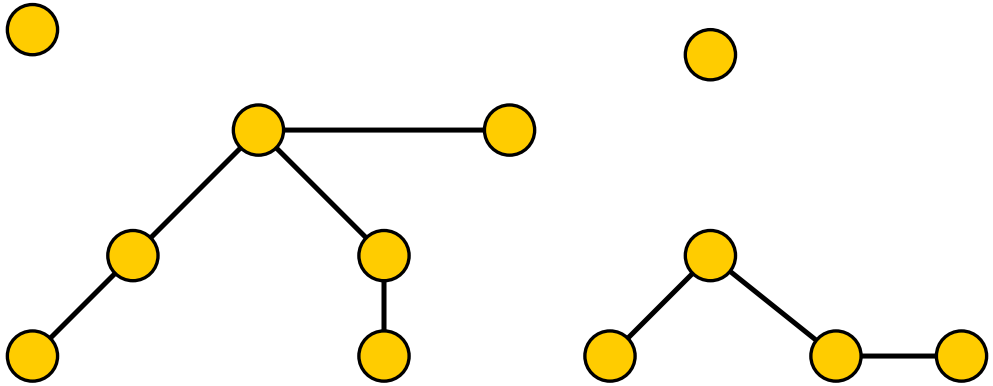
- We allow multiple parallel edges and self loops.
- A **feedback vertex set** is a set that hits every cycle in the graph.



Feedback Vertex Set

Feedback Vertex Set:
Given (G, k), find a set S of at most k vertices such that $G - S$ has no cycles.

- We allow multiple parallel edges and self loops.
- A **feedback vertex set** is a set that hits every cycle in the graph.



Feedback Vertex Set

- If we find a cycle, then we have to include at least one of its vertices into the solution. But the length of the cycle can be arbitrary large!
- **Main idea:** We identify a set of $O(k)$ vertices such that any size-k feedback vertex set has to contain one of these vertices.
- But first: some reductions to simplify the problem.

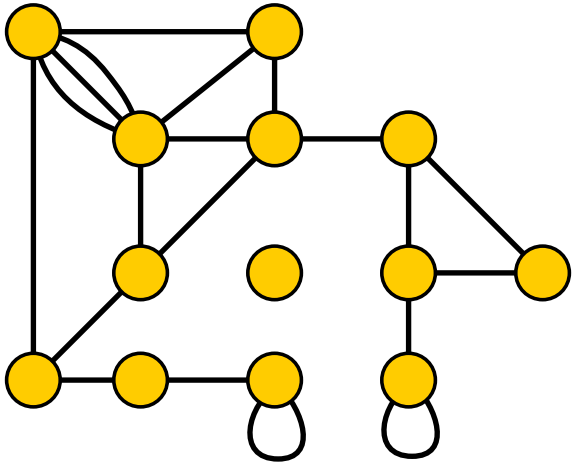
Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.



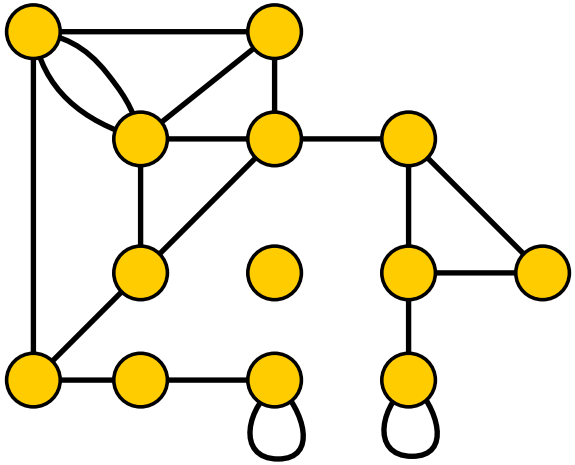
Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.



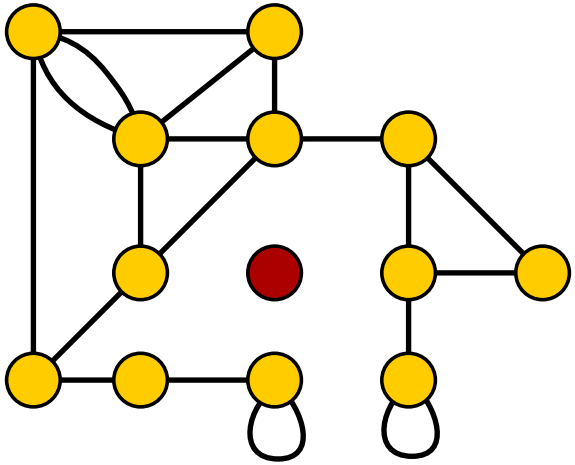
Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.



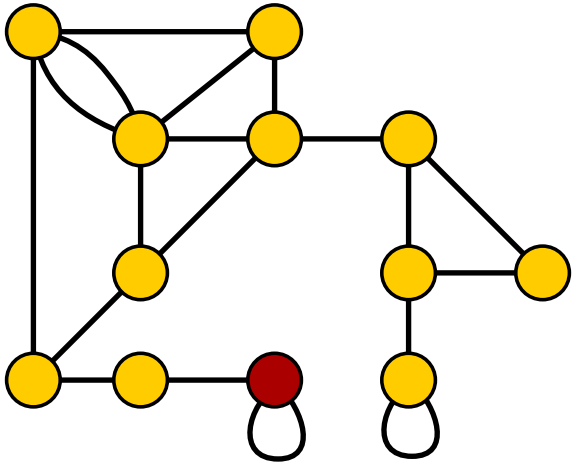
Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.



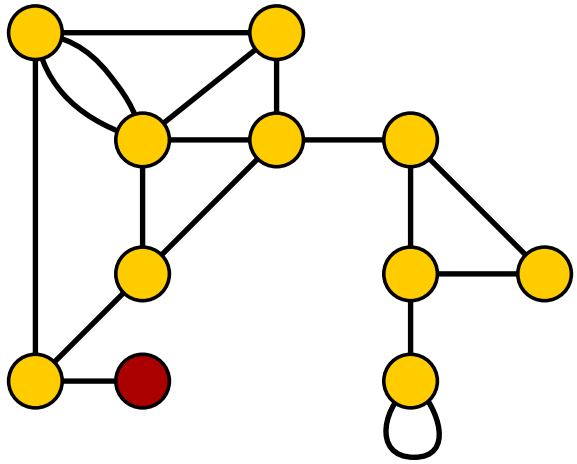
Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.



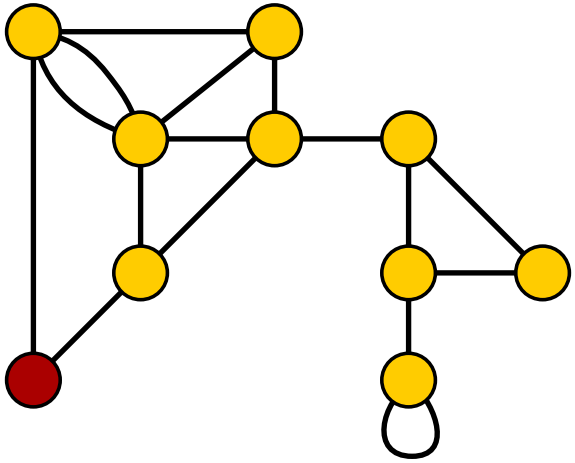
Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.



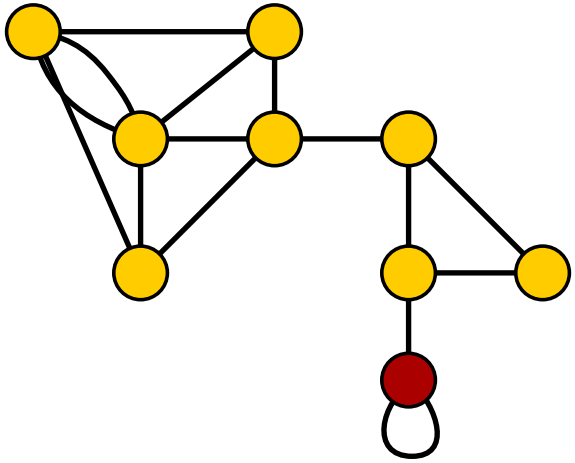
Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.



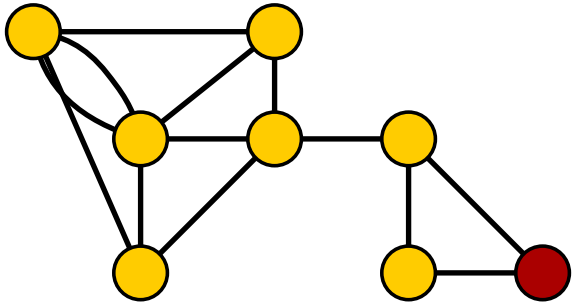
Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.



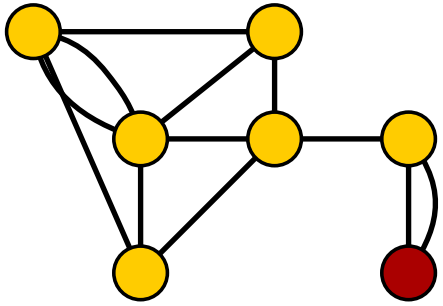
Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.



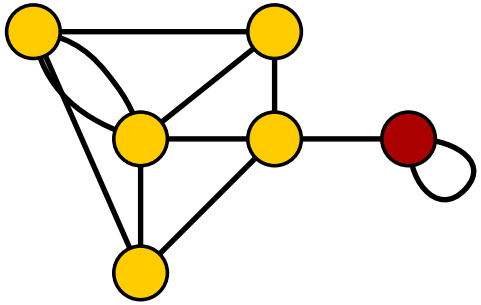
Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.



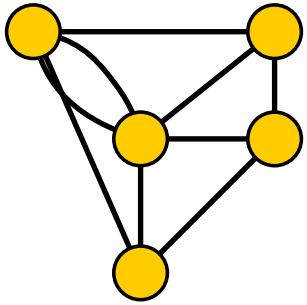
Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.



If the reduction rules cannot be applied, then every vertex has degree at least 3.

Branching

Let G be a graph whose vertices have degree at least 3.

- Order the vertices as v_1, v_2, ..., v_n by **decreasing** degree (breaking ties arbitrarily).
- Let $V_{3k} = \{v_1, \ldots, v_{3k}\}$ be the $3k$ largest-degree vertices.

Lemma

If G has minimum degree at least 3, then every feedback vertex set S of size at most k contains a vertex from V_{3k}.

Branching

Let G be a graph whose vertices have degree at least 3.

- Order the vertices as $v_1, v_2, \ldots, v_n$ by **decreasing** degree (breaking ties arbitrarily).
- Let $V_{3k} = \{v_1, \ldots, v_{3k}\}$ be the $3k$ largest-degree vertices.

Lemma

If G has minimum degree at least 3, then every feedback vertex set S of size at most k contains a vertex from V_{3k}.

Algorithm:

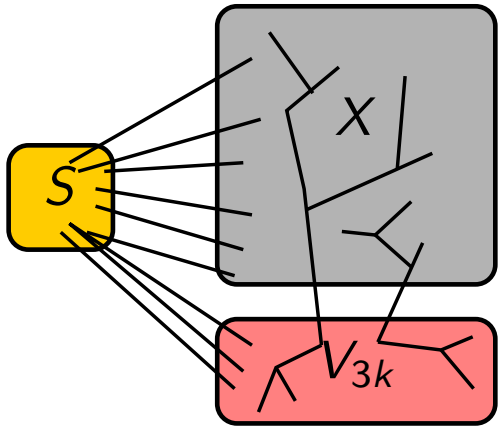
- Apply the reduction rules (poly time) $\Rightarrow$ graph has minimum degree 3.
- For each vertex $v \in V_{3k}$, recurse on the instance $(G - v, k - 1)$.
- Running time $(3k)^k \cdot n^{O(1)} = 2^{O(k \log k)} \cdot n^{O(1)}$.

Proof of the lemma

Lemma

If G has minimum degree at least 3, then every feedback vertex set S of size at most k contains a vertex from V_{3k}.

- $d :=$ minimum degree in V_{3k} ,
  $X = V(G) - (S \cup V_{3k})$.
- Total degree of $V_{3k} \cup X$: $\geq 3kd + 3|X|$
- Edges of $G[V_{3k} \cup X]$: $\leq 3k + |X| - 1$
- Total degree of these edges: $\leq 6k + 2|X| - 2$

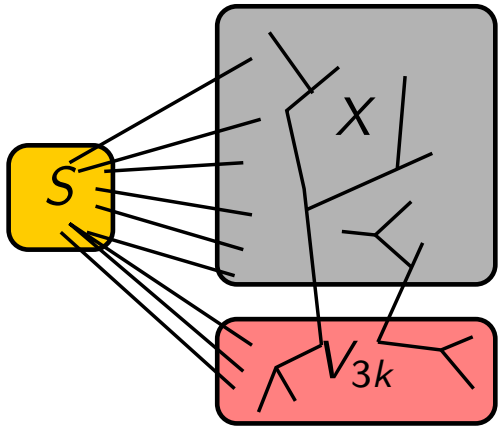


Proof of the lemma

Lemma

If G has minimum degree at least 3, then every feedback vertex set S of size at most k contains a vertex from V_{3k}.

- $d :=$ minimum degree in V_{3k} ,
  $X = V(G) - (S \cup V_{3k})$.
- Total degree of $V_{3k} \cup X$: $\geq 3kd + 3|X|$
- Edges of $G[V_{3k} \cup X]$: $\leq 3k + |X| - 1$
- Total degree of these edges: $\leq 6k + 2|X| - 2$
- Edges between S and $V_{3k} \cup X$:
  - $\leq dk$
  - $\geq 3kd + 3|X| - (6k + 2|X| - 2) > 3(d-2)k$
- As $d \geq 3$, we have $3(d-2) \geq d$, contradiction.



Branching: wrap up

- Branching into c directions: $O^*(c^k)$ algorithms.
- Branching into k directions: $O^*(k^k)$ algorithms.
- Branching vectors and analysis of recurrences of the form

$$T(k) = T(k-1) + 2T(k-2) + T(k-3)$$

- Graph modification problems where the graph property can be characterized by a finite set of forbidden induced subgraphs is FPT.

Reading assignment: Section 3.5 of the Parameterized Algorithms books - CLOSEST STRING