# Graph Canonicalization
# Graph Isomorphism
# Graph Automorphisms

DM840, Week 37

# Motivation

I have two molecules, are they actually 'the same' molecule?

(Graph isomorphism (potentially via canonicalization))

Which symmetries does this molecule have?

("symmetry" $\equiv$ "automorphism", "all symmetries" $\equiv$ "the automorphism group")

I have many molecules, and just got one more, do I have it already?

(Graph isomorphism (likely via canonicalization))

I generate molecules in different ways, and store them. I want each molecule to always be stored in the same way, no matter how it was generated.

(Graph canonicalization)

# Introduction

## The Real World

That messy thing we are trying to study (with computers).

# Introduction

## The Real World

That messy thing we are trying to study (with computers).

## Model

A mathematical object in some class $M$.

# Introduction

## The Real World

That messy thing we are trying to study (with computers).

## Model

A mathematical object in some class $M$.

## Representation

An object of an abstract data type $R$ used to store the model.

# Introduction

## The Real World

That messy thing we are trying to study (with computers).

## Model

A mathematical object in some class $M$.

## Representation

An object of an abstract data type $R$ used to store the model.

## Implementation

An object of a concrete type used to store the model.

# Introduction

## The Real World

That messy thing we are trying to study (with computers).

## Model

A mathematical object in some class $M$.

## Representation

An object of an abstract data type $R$ used to store the model.

## Implementation

An object of a concrete type used to store the model.

**Our Subject**: the relationship between models and representations.

# Fractions

## Model

A mathematical object in some class $M$.

Example: a rational number, $\frac{3}{4}$

## Representation

An object of an abstract data type $R$ used to store the model.

Example: a pair of integers, $(3, 4)$

## Implementation

An object of a concrete type used to store the model.

Example: `std::pair<int, int>(3, 4)`

# Fractions

## Model

A mathematical object in some class $M$.

Example: a rational number, $\frac{3}{4}$

## Representation

An object of an abstract data type $R$ used to store the model.

Example: a pair of integers, $(3, 4)$

## Implementation

An object of a concrete type used to store the model.

Example: `std::pair<int, int>(3, 4)`

When are two fractions 'the same'?
Are $\frac{2}{5}$ and $\frac{4}{10}$ the same thing? yes!

Note: Our representation set $R$ may be redundant.

# Terminology and Notation

"Equal" can mean multiple things:

- Isomorphism, representing the same model.
  Often a relatively computationally expensive to check.
  $(2, 5) \cong (4, 10)$ ("isomorphic to")
- Representational equality, the data structures are equal.
  Usually straight-forward and cheap to check.
  $(2, 5) \overset{r}{=} (2, 5)$ ("representationally equal to")
- Alias, the names refer to the same mathematical object.
  $A = B$

# Canonicalization

Given a representation $G \in R$ find a new representation $C(G)$, a canonical form, such that:

- It represents the same model: $C(G) \cong G$
- All canonicalized isomorphic representations are the same: $\forall G' \in R, G' \cong G : \quad C(G') \overset{r}{=} C(G)$

How do we specify and implement canonicalization in practice?

# Representations

Besides the $\overset{r}{=}$ operation we need:

- A class of operations, $\mathrm{OP}$, that do not change the model.
- A total order $\overset{r}{<}$ among (isomorphic) representations.

# Representations

Besides the $\overset{r}{=}$ operation we need:

- A class of operations, $\mathrm{OP}$, that do not change the model.
- A total order $\overset{r}{<}$ among (isomorphic) representations.

Fraction Example:

$\mathrm{OP}$:

- Multiplying with an integer: $(2,5) \cdot 2 = (2 \cdot 2, 2 \cdot 5) \cong (2,5)$
- Dividing with a common factor: $\frac{(4,10)}{2} = \left(\frac{4}{2}, \frac{10}{2}\right) \cong (4,10)$
- (and compositions of those operations)

# Representations

Besides the $\overset{r}{=}$ operation we need:

- A class of operations, $\mathrm{OP}$, that do not change the model.
- A total order $\overset{r}{<}$ among (isomorphic) representations.

Fraction Example:

$\mathrm{OP}$:

- Multiplying with an integer: $(2,5) \cdot 2 = (2 \cdot 2, 2 \cdot 5) \cong (2,5)$
- Dividing with a common factor: $\frac{(4,10)}{2} = \left(\frac{4}{2}, \frac{10}{2}\right) \cong (4,10)$
- (and compositions of those operations)

$\overset{r}{<}$:

- Prefer both positive over both negative: $(2,5) \overset{r}{<} (-2,-5)$
- Prefer (neg., pos.) over (pos., neg.): $(-2,5) \overset{r}{<} (2,-5)$
- Prefer smaller (absolute) numbers (lexicographically):
  $(2,5) \overset{r}{<} (4,10), \quad (1,2) \overset{r}{<} (2,3)$

# Canonicalization

Given $G \in R$:

- Find an operation $op \in \mathrm{OP}$ that produces the $\overset{r}{<}$-smallest representation.
- Return that representation $op(G)$ as the canonical form.

## Fraction Example:

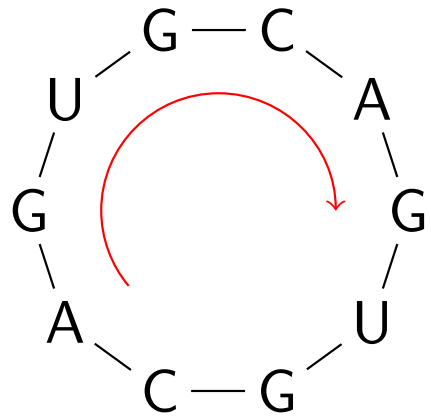
Given $(a, b)$,

- Find $f = GCD(|a|, |b|)$
- If $b < 0$: let $op = \mathrm{DIV}(f) \circ \mathrm{MUL}(\text{-}1)$
  else: let $op = \mathrm{DIV}(f)$
- Return $op((a, b))$

## In Practice:

- Probably return $op$. The user can compute $op(G)$ if needed.
- $\overset{r}{<}$ may be implicitly defined by the canonicalization algorithm.

# Example: Circular RNA (circRNA)



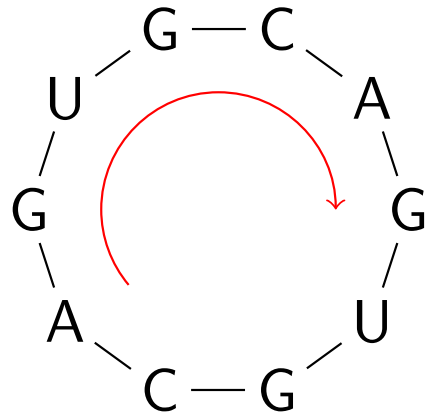

Representation: A sequence of symbols A, C, G, U.
Example: AGUGCAGUGC

Operations: $\textsc{Rotate}(i)$, for $i \in \mathbb{Z}$
Example: $\textsc{Rotate}(2, \text{AGUGCAGUGC}) = \text{UGCAGUGCAG}$

$\overset{r}{=}$ and $\overset{r}{<}$: component-wise and lexicographic comparison

Canonicalization: find the lexicographically smallest rotation
(can be done in linear time)

# Example: Circular RNA (circRNA)




Representation: A sequence of symbols A, C, G, U.
  Example: AGUGCAGUGC

Operations: $\textsc{Rotate}(i)$, for $i \in \mathbb{Z}$
  Example: $\textsc{Rotate}(2, \text{AGUGCAGUGC}) = \text{UGCAGUGCAG}$

$\overset{r}{=}$ and $\overset{r}{<}$: component-wise and lexicographic comparison

Canonicalization: find the lexicographically smallest rotation
(can be done in linear time)

Symmetry Discovery: $op$ is a symmetry if $op(G) \overset{r}{=} G$
  Example: $\textsc{Rotate}(5)$ is a symmetry of AGUGCAGUGC, because
  $\textsc{Rotate}(5, \text{AGUGCAGUGC}) = \text{AGUGCAGUGC}$
  $\overset{r}{=} \text{AGUGCAGUGC}$
  $\textsc{Rotate}(0)$ is a trivial symmetry

# Example: Double Stranded RNA

Representation:

A pair of sequences of symbols A, C, G, U, of equal length.

Example: $\begin{matrix}\mathsf{AGUGC}\\ \mathsf{UCACG}\end{matrix}$

Operations: $\textsc{Reverse} \circ \textsc{Swap}$

Example: $(\textsc{Reverse} \circ \textsc{Swap}) \begin{pmatrix}\mathsf{AGUGC}\\ \mathsf{UCACG}\end{pmatrix} = \begin{matrix}\mathsf{GCACU}\\ \mathsf{CGUGA}\end{matrix}$

$\overset{r}{=}$ and $\overset{r}{<}$: component-wise and lexicographic comparison

Example: $\begin{matrix}\mathsf{AGUGC}\\ \mathsf{UCACG}\end{matrix} \overset{r}{<} \begin{matrix}\mathsf{GCACU}\\ \mathsf{CGUGA}\end{matrix}$

Canonicalization: take the $\overset{r}{<}$-smallest of the two possibilities

# Permutations

Permutation: a 1-to-1 map $\gamma\colon S \to S$.
Usually $S$ is the positions of a list $\{1, 2, \ldots, n\}$.

Example:

Table/matrix notation:

| $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $\gamma(i)$ | 3 | 4 | 1 | 2 |

Cycle notation:
$(1\ 3)(2\ 4)$

We use the notation $i^\gamma$ instead of $\gamma(i)$.
So $\sigma(\gamma(i))$ will be $(i^\gamma)^\sigma = i^{\gamma\sigma}$.

# Permutations

Permutation: a 1-to-1 map $\gamma\colon S \to S$.
Usually $S$ is the positions of a list $\{1, 2, \ldots, n\}$.

Example:

Table/matrix notation:

| $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $\gamma(i)$ | 3 | 4 | 1 | 2 |

Cycle notation:
$(1\ 3)(2\ 4)$

We use the notation $i^\gamma$ instead of $\gamma(i)$.
So $\sigma(\gamma(i))$ will be $(i^\gamma)^\sigma = i^{\gamma\sigma}$.

Permuting a Set:

For a set $X$, we use $X^\gamma$ to mean $\{x^\gamma \mid x \in X\}$.

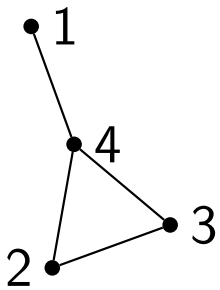
Example:

For $\gamma = (1\ 3)(2\ 4)$ and $X = \{2, 3\}$ we get
$X^\gamma = \{2^\gamma, 3^\gamma\} = \{4, 1\}$

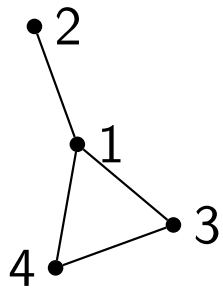
# Graph Representation

$G = (V, E) \quad V = \{1, 2, \ldots, n\}$

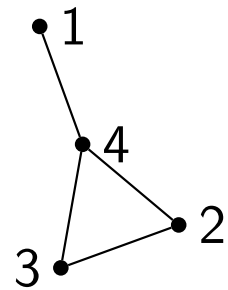
Isomorphic graphs, different representations:




$G_1$




$G$




$G_2$

Adjacency matrix representation:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | 1 |
| 2 | | | 1 | 1 |
| 3 | | 1 | | 1 |
| 4 | 1 | 1 | 1 | |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | 1 | 1 | 1 |
| 2 | 1 | | | |
| 3 | 1 | | | 1 |
| 4 | 1 | | 1 | |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | 1 |
| 2 | | | 1 | 1 |
| 3 | | 1 | | 1 |
| 4 | 1 | 1 | 1 | |

Adjacency list representation (with sorted neighbour lists):

1: 4
2: 3, 4
3: 2, 4
4: 1, 2, 3

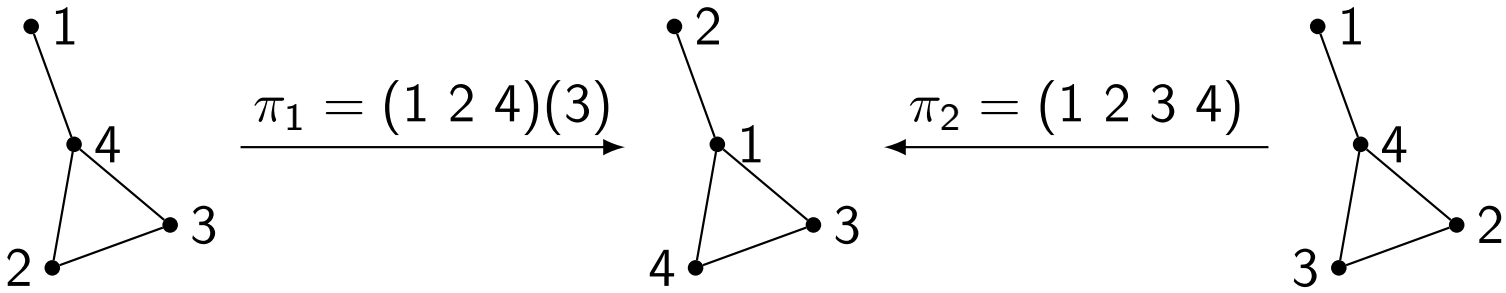
1: 2, 3, 4
2: 1
3: 1, 4
4: 1, 3

1: 4
2: 3, 4
3: 2, 4
4: 1, 2, 3

# Graph Representation

$G = (V, E) \quad V = \{1, 2, \ldots, n\}$

Isomorphic graphs, different representations:




$G_1$ $\qquad$ $G_1^{\pi_1} \overset{r}{=} G \overset{r}{=} G_2^{\pi_2}$ $\qquad$ $G_2$

Adjacency matrix representation:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | 1 |
| 2 | | | 1 | 1 |
| 3 | | 1 | | 1 |
| 4 | 1 | 1 | 1 | |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | 1 | 1 | 1 |
| 2 | 1 | | | |
| 3 | 1 | | | 1 |
| 4 | 1 | | 1 | |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | 1 |
| 2 | | | 1 | 1 |
| 3 | | 1 | | 1 |
| 4 | 1 | 1 | 1 | |

Adjacency list representation (with sorted neighbour lists):

1: 4
2: 3, 4
3: 2, 4
4: 1, 2, 3

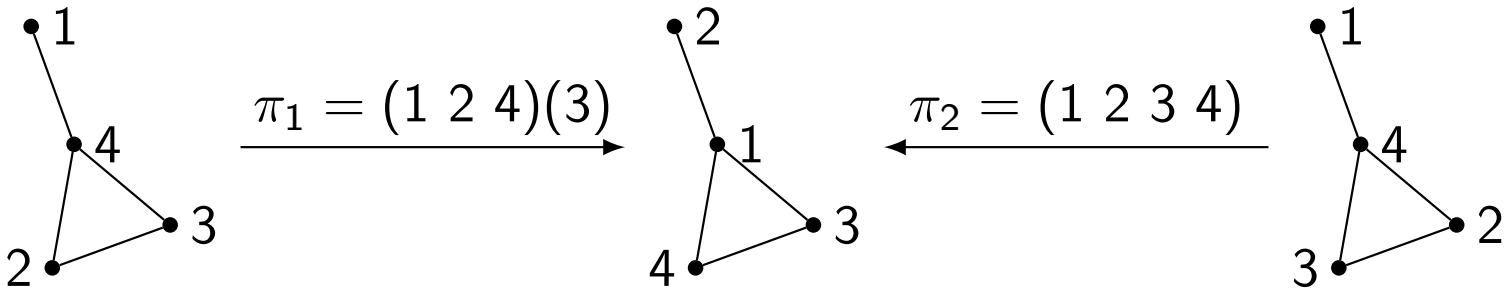
1: 2, 3, 4
2: 1
3: 1, 4
4: 1, 3

1: 4
2: 3, 4
3: 2, 4
4: 1, 2, 3

# Graph Representation

$G = (V, E) \quad V = \{1, 2, \ldots, n\}$

Isomorphic graphs, different representations:




$G_1$ $\qquad$ $G_1^{\pi_1} \overset{r}{=} G \overset{r}{=} G_2^{\pi_2}$ $\qquad$ $G_2$

Adjacency matrix representation:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | 1 |
| 2 | | | 1 | 1 |
| 3 | | 1 | | 1 |
| 4 | 1 | 1 | 1 | |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | 1 | 1 | 1 |
| 2 | 1 | | | |
| 3 | 1 | | | 1 |
| 4 | 1 | | 1 | |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | 1 |
| 2 | | | 1 | 1 |
| 3 | | 1 | | 1 |
| 4 | 1 | 1 | 1 | |

Adjacency list representation (with sorted neighbour lists):

1: 4
2: 3, 4
3: 2, 4
4: 1, 2, 3

1: 2, 3, 4
2: 1
3: 1, 4
4: 1, 3

1: 4
2: 3, 4
3: 2, 4
4: 1, 2, 3

Optional exercise: try permuting an adjacency list/matrix by hand.

# Graphs Canonicalization

Model: A graph $G = (V, E)$.

Representation: An adjacency list/matrix which implicitly assigns $1, 2, \ldots, n$ to $V$.

Operations: $\textsc{Permute}(\gamma)$ for any permutation of $1, 2, \ldots, n$.

$\overset{r}{=}$ and $\overset{r}{<}$: component-wise and lexicographic comparison

Computational Complexity: $\exp\left(O\left(\sqrt{n \log n}\right)\right)$

Brute-Force Algorithm:

1. Construct $G^\gamma$ for all permutations $\gamma \in S_n$.
2. Select the "best" one (for example the $\overset{r}{<}$-smallest).

Generally not feasible to check all $n!$ permutations of $n$ vertices.

---

[Babai and Luks, STOC, 1983]

[Babai, Handbook of Combinatorics, 1996]

# Existing Tools for Canonicalization in Practice

Published Tools: nauty, Traces, Bliss (and Saucy and Conauto)

- All based on the idea of individualization-refinement.
- Different sets of heuristics and variations.
- Many more algorithm variations are possible.
- Which is the best? for a specific class of graphs?
- What if the graph has vertex and edge labels?
- What if those labels are "complicated"? (e.g., stereo-info)

---

[McKay, Congressus Numerantium, 1981] [McKay and Piperno, J. Symb. Comp., 2014] [Junttila and Kaski, ALENEX, 2007] [Darga et al., DAC, 2008] [López-Presa and Fernández Anta, SEA, 2009]

# Existing Tools for Canonicalization in Practice

Published Tools: nauty, Traces, Bliss (and Saucy and Conauto)

- All based on the idea of individualization-refinement.
- Different sets of heuristics and variations.
- Many more algorithm variations are possible.
- Which is the best? for a specific class of graphs?
- What if the graph has vertex and edge labels?
- What if those labels are "complicated"? (e.g., stereo-info)

GraphCanon: [Andersen and Merkle, ALENEX, 2018]

- A generic C++ library for canonization algorithms.
- Code: `https://github.com/jakobandersen/graph_canon`
- Visualizer: `https://jakobandersen.github.io/graph_canon_vis`

---

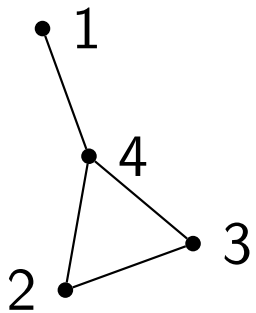
[McKay, Congressus Numerantium, 1981] [McKay and Piperno, J. Symb. Comp., 2014] [Junttila and Kaski, ALENEX, 2007] [Darga et al., DAC, 2008] [López-Presa and Fernández Anta, SEA, 2009]

# Individualization-Refinement

Idea:

- We need an order of the vertices.
- In the beginning, we don't know anything about that order.
- That is, we start with a set of vertices and must create a sequence of vertices.
- We can decide the rules for the ordering.
- Use "easy" rules to introduce order gradually.

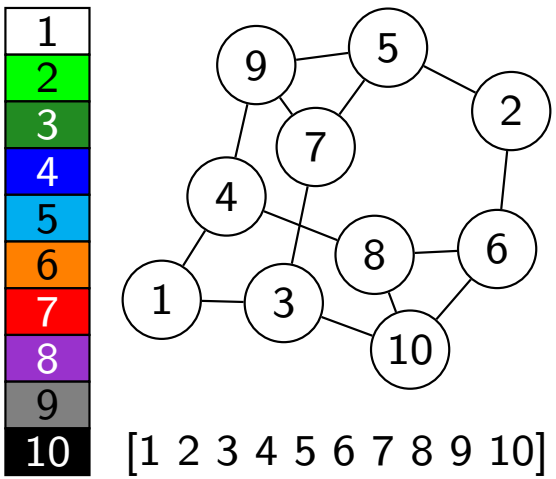
Example: Initially we have a set $V = \{1, 2, 3, 4\}$.




Can we define some (partial) order without looking at the indices?

# Individualization-Refinement Paradigm

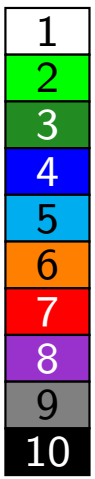
Initially: all vertices are unordered (same colour).

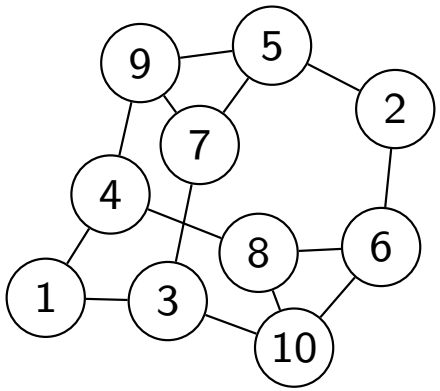


# Individualization-Refinement Paradigm

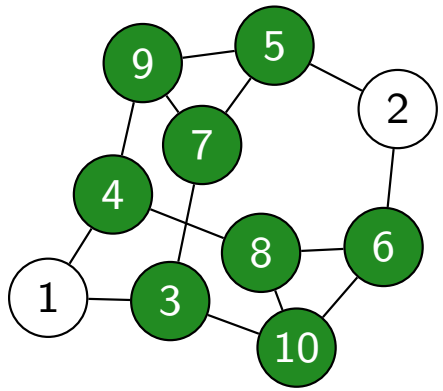
Refine the ordering by propagation of "cheap" local information.

Example: sort and partition by degree (1D Weisfeiler-Leman).




[1 2 3 4 5 6 7 8 9 10]

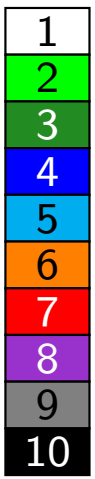


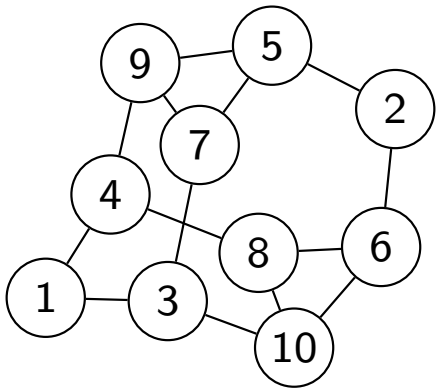

[1 2 | 3 4 5 6 7 8 9 10]

# Individualization-Refinement Paradigm

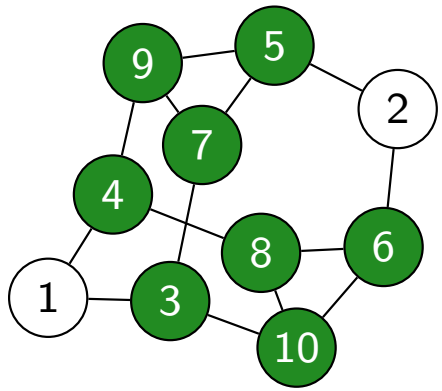
Refine the ordering by propagation of "cheap" local information.

Example: sort and partition by degree (1D Weisfeiler-Leman).

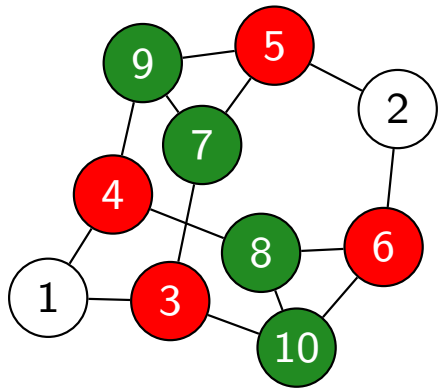



[1 2 3 4 5 6 7 8 9 10]




[1 2 | 3 4 5 6 7 8 9 10]



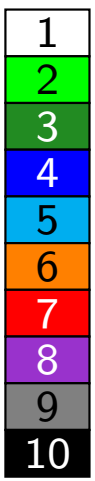

[1 2 | 7 8 9 10 | 3 4 5 6]

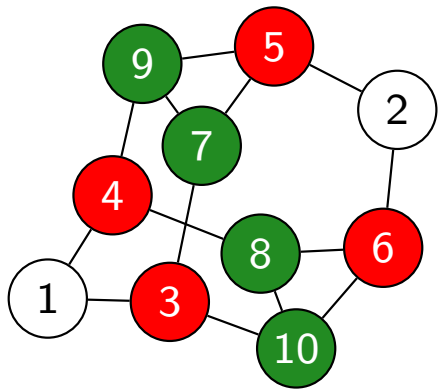
# Individualization-Refinement Paradigm

Let this be the root of a search tree, and select a colour.

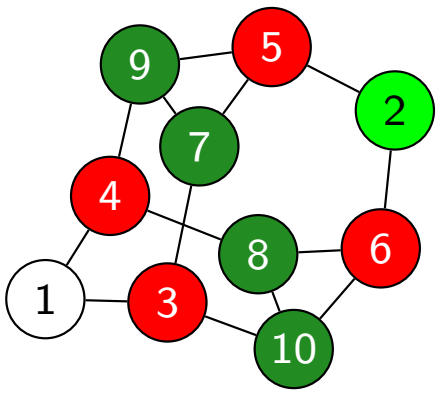
For each vertex of that colour;

create a child with this vertex given a unique new colour.

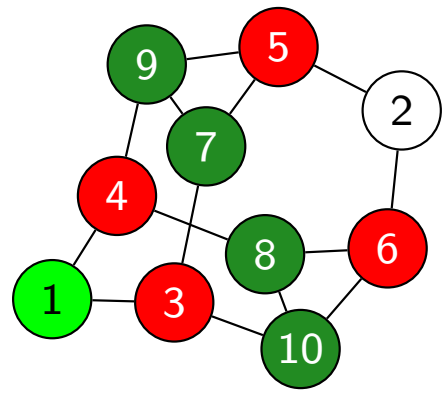



[1 2 | 7 8 9 10 | 3 4 5 6]

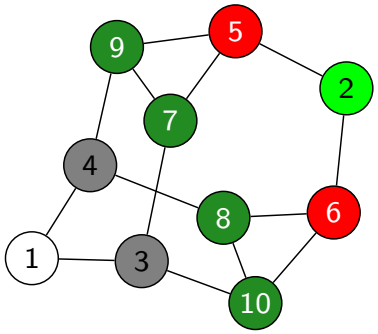



[1 | 2 | 7 8 9 10 | 3 4 5 6]

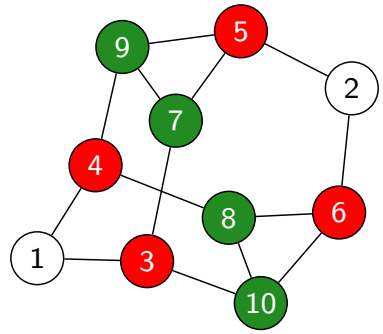



[2 | 1 | 7 8 9 10 | 3 4 5 6]

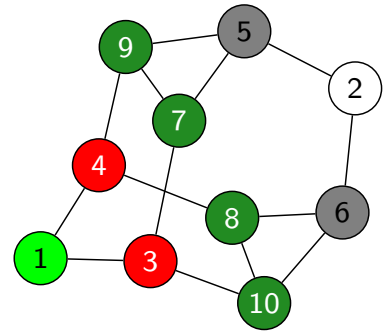
# Individualization-Refinement Paradigm



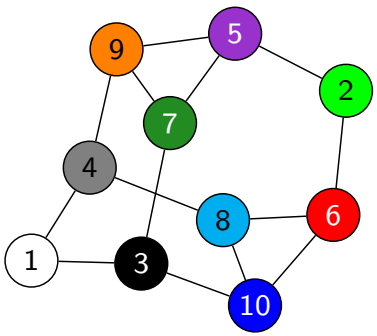

$\pi_{(1)} = [1 \mid 2 \mid 7\ 8\ 9\ 10 \mid 5\ 6 \mid 3\ 4]$



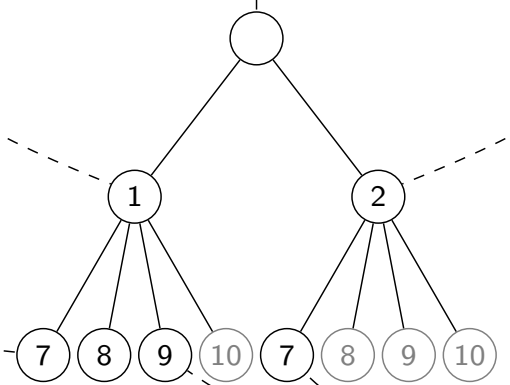

$\pi_{()} = [1\ 2 \mid 7\ 8\ 9\ 10 \mid 3\ 4\ 5\ 6]$

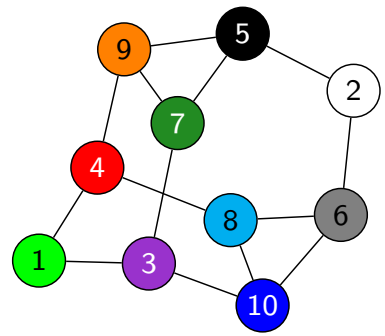



$\pi_{(2)} = [2 \mid 1 \mid 7\ 8\ 9\ 10 \mid 3\ 4 \mid 5\ 6]$

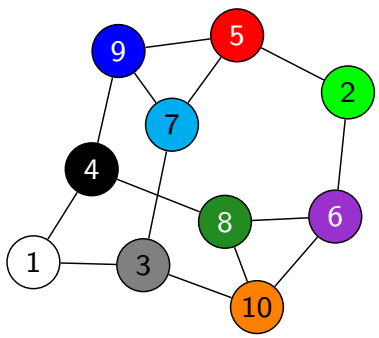



$\pi_{(1,7)} = [1 \mid 2 \mid 7 \mid 10 \mid 8 \mid 9 \mid 6 \mid 5 \mid 4 \mid 3]$



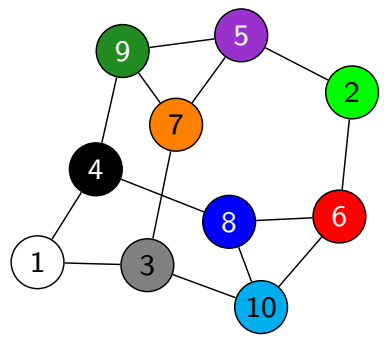




$\pi_{(2,7)} = [2 \mid 1 \mid 7 \mid 10 \mid 8 \mid 9 \mid 4 \mid 3 \mid 6 \mid 5]$




$\pi_{(1,8)} = [1 \mid 2 \mid 8 \mid 9 \mid 7 \mid 10 \mid 5 \mid 6 \mid 3 \mid 4]$

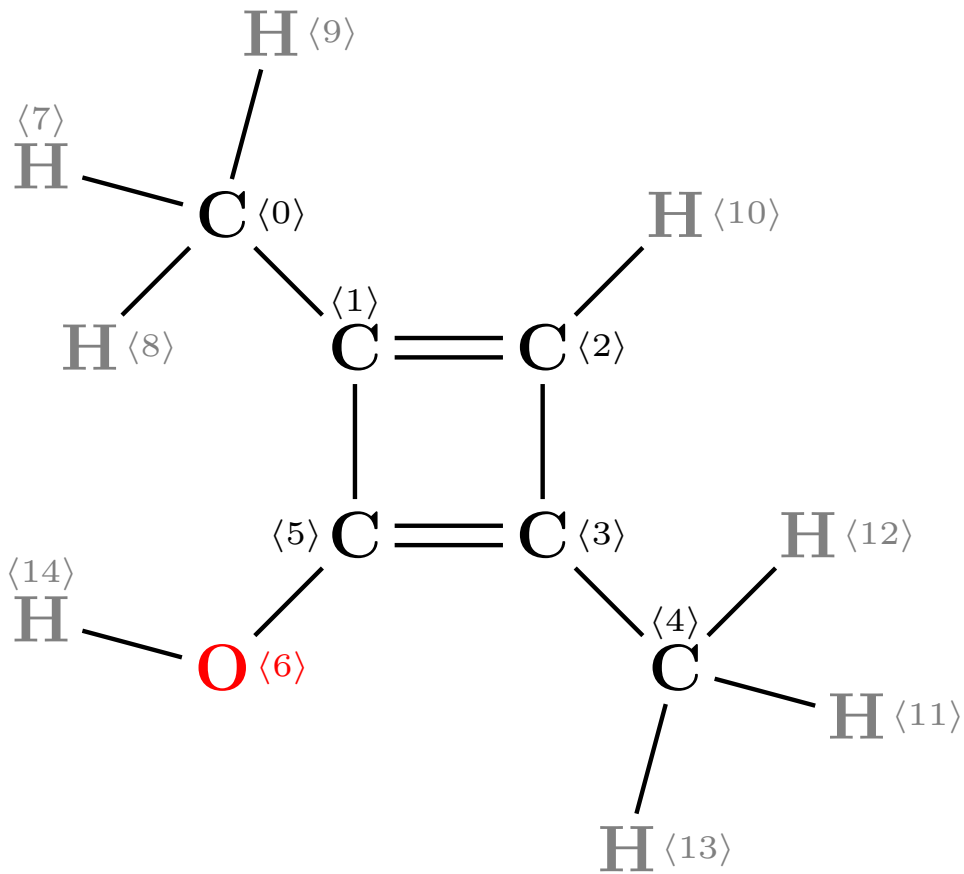

Colour order




$\pi_{(1,9)} = [1 \mid 2 \mid 9 \mid 8 \mid 10 \mid 7 \mid 6 \mid 5 \mid 3 \mid 4]$

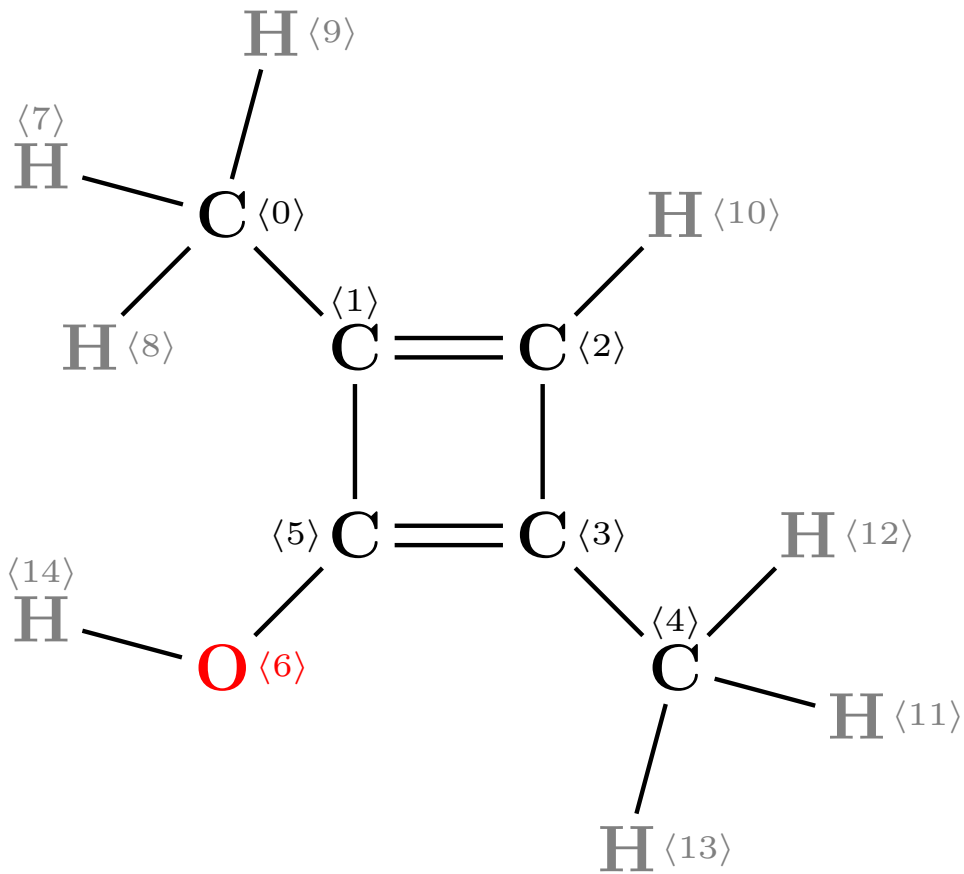
# Labelled Graphs

Consider molecules, what about the atom types?



# Labelled Graphs

Consider molecules, what about the atom types?




Vertex labels: use them to make an initial partial order.

Edge labels: compare them as part of $\overset{r}{<}$.

# Abstract Algorithm

- Construct the search tree.
- For each leaf, construct the permuted graph using the discrete partition as a vertex permutation.
- For two such permuted graphs $G^{\overline{\pi_1}}$ and $G^{\overline{\pi_2}}$,
  - if $G^{\overline{\pi_1}} \overset{r}{<} G^{\overline{\pi_2}}$, discard $\pi_2$,
  - if $G^{\overline{\pi_1}} \overset{r}{=} G^{\overline{\pi_2}}$, yield $\overline{\pi_1}\pi_2$ as automorphism, and discard either $\pi_1$ or $\pi_2$.
- Return the permutation represented by the remaining leaf.

## Pruning Techniques

- Use automorphisms to skip redundant subtrees.
- Use node invariants to remove subtrees (possibly changing which leaves are ever constructed).

# Algorithm Variation

## Categories

- Tree traversal
- Target cell selection
- Refinement
- Pruning with automorphisms
- Detection of implicit automorphisms
- Node invariants

# Final Notes

- Canonicalization is a general concept for representations of models. Note just graphs.

# Final Notes

- Canonicalization is a general concept for representations of models. Note just graphs.
- It is entirely a core Computer Science problem.

# Final Notes

- Canonicalization is a general concept for representations of models. Note just graphs.
- It is entirely a core Computer Science problem.
- It can be non-trivial to create a correct and good algorithm,

# Final Notes

- Canonicalization is a general concept for representations of models. Note just graphs.
- It is entirely a core Computer Science problem.
- It can be non-trivial to create a correct and good algorithm,
- so probably find a good library for it,

# Final Notes

- Canonicalization is a general concept for representations of models. Note just graphs.
- It is entirely a core Computer Science problem.
- It can be non-trivial to create a correct and good algorithm,
- so probably find a good library for it,
- but be very careful: there are unfortunately several bad cheminformatics papers on this topic.

  [Weininger et al., Algorithm for Generation of Unique SMILES Notation, 1988]

  [Schneider et al., Get Your Atoms in Order — An Open-Source Implementation of a Novel and Robust Molecular Canonicalization Algorithm, 2015]

  Note: neither paper has a proof of correctness.

  Rule of thumb: if the so-called algorithm has a "tie-breaking" step, it's probably wrong.