Isomorphism motivation

- Tiny piglet
- Getting strong and backwards
- Reflected in water
- Inspecting its tail
- Painted by abstract artist
- Serialized in Java file
Isomorphism motivation

Should we consider these two schemes to be identical regardless of the pictures geometry?

One triangle and three quadrilaterals inside a triangle.

Two triangles and two quadrilaterals inside a rectangle.

Different representations (views) of the same original structure.
Isomorphism informally

Geometry does not help to confirm the fact that different representations (pictures, descriptions...) correspond to the same structure. On the contrary, it rather seems to obscure the fact quite easily:

G1 and G2 do not depict the same structure, while G1 and G3 do.

The structure, in this context, is the set of nodes and the set of edges between them.
Two graphs are called **isomorphic** to each other when, in fact, they are absolutely the same graph. They only pretend to be different (if they pretend it at all).

\[ G_1 = (\{a,b,c,d\}, \{\{a,b\}, \{b,c\}, \{c,a\}, \{b,d\}\}) \]

// Set of nodes and set of edges

\[ G_2 = (\{a,b,c,d\}, \{\{a,b\}, \{b,c\}, \{c,a\}, \{b,d\}\}) \]

// Set of nodes and set of edges

Clearly, \( G_1, G_2, G_3, G_4, G_5 \) are all pairwise isomorphic to each other.

\[ G_3: \begin{pmatrix}
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 \\
\end{pmatrix} \]

// Adjacency matrix of \( G_3 \)

\[ G_4 \]

// Posh garden
// scheme

\[ G_5 \]

// Posh garden
// scheme

\[ G_5 \]

// Posh garden
// scheme

\[ G_A: \begin{pmatrix}
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 \\
\end{pmatrix} \]

// Adjacency matrix of \( G_5 \)

G5 is an undirected simple graph consisting of 4 nodes and 4 edges. It contains a node of degree 1.

// This defines \( G_5 \) unambiguously.

G5 is an undirected simple graph consisting of 4 nodes and 4 edges. It contains a node of degree 1.

// This defines \( G_5 \) unambiguously.
Two graphs, G1 and G2, are called **isomorphic** to each other when there exists a one-to-one correspondence between the nodes of G1 and the nodes of G2. Additionally, this correspondence between the nodes also completely mirrors the information about the edges in both graphs, in the sense:

There is an edge between x and y in G1 if and only if there is an edge between nodes corresponding to x and y in G2.

There may be more than one such correspondence between the nodes of G1 and G2 when the graphs are isomorphic.

According to informal definition, when G1 and G2 are isomorphic, they both represent the same graph. In effect, the one-to-one correspondence between the nodes of G1 and G2 is a one-to-one correspondence between the nodes of a single graph. Is there any practical sense of studying?
Let $G$ be a graph. A one-to-one correspondence between the nodes of $G$ is called an automorphism of $G$ when

There is an edge between $x$ and $y$ in $G_1$ if and only if there is an edge between nodes corresponding to $x$ and $y$ in $G_2$

There always exists at least one automorphism for any graph. Trivially, it is possible to map each node to itself.

The one-to-one correspondence between the nodes of a graph is a permutation of the nodes.

There are $N!$ different permutations of the nodes. Which of them are automorphisms, and which of them are not, that depends on the graph itself.
Isomorphism formally

On a complete graph on N nodes, there are N! automorphisms, any permutation of the nodes is an automorphisms. For N = 6 there are 720 different automorphisms.

On a complete bipartite graph on M and N nodes, there are M! × N! automorphisms, any permutation which maps a node to another node in the same partition is an automorphisms. For M = 4, N = 3, there are 4! × 3! = 24 × 6 = 144 different automorphisms.

On a path graph N nodes, there are 2 automorphisms. One is identity permutation, the other is the permutation which maps each node to its counterpart on the same place on the “reversed” path.
On many graphs, there is only one automorphism, represented by the identity permutation of the nodes.

In general, the composition of two automorphisms is another automorphism (it is a composition of permutations), and the set of automorphisms of a given graph, under the composition operation, forms a group, the automorphism group of the graph.
Examples of isomorphic and non-isomorphic graphs

\[ G_1 \]
\[ \begin{array}{cccccc}
 a & b & e & f & c & d \\
 c & d & e & f & a & b \\
 b & e & f & a & d & c \\
 e & f & a & d & b & c \\
 f & a & d & c & e & b \\
 a & b & e & f & c & d \\
\end{array} \]

\[ G_2 \]
\[ \begin{array}{cccccc}
 1 & 2 & 6 & 5 & 4 & 3 \\
 4 & 3 & 5 & 6 & 1 & 2 \\
 1 & 4 & 6 & 3 & 2 & 5 \\
 2 & 3 & 5 & 1 & 6 & 4 \\
 5 & 6 & 1 & 2 & 4 & 3 \\
 3 & 5 & 6 & 4 & 1 & 2 \\
\end{array} \]

- \(|\text{nodes}| = 6\)
- \(|\text{edges}| = 9\)
- \(\text{is regular} = \text{true}\)
- \(\text{max degree} = 3\)
- \(\text{diameter} = 2\)
- \(\text{no. of triangles} = 0\) (triangles 1-4-6 and 2-3-5)

- \(|\text{nodes}| = 6\)
- \(|\text{edges}| = 9\)
- \(\text{is regular} = \text{true}\)
- \(\text{max degree} = 3\)
- \(\text{diameter} = 2\)
- \(\text{no. of triangles} = 2\)
Examples of isomorphic and non-isomorphic graphs

\[ \left| V(G_1) \right| = 5 \]
\[ \left| E(G_1) \right| = 6 \]
\text{min degree} = 2
\text{max degree} = 3
\text{degree sequence} = [3\ 3\ 3\ 3\ 2]

\[ \left| V(G_2) \right| = 5 \]
\[ \left| E(G_2) \right| = 6 \]
\text{min degree} = 2
\text{max degree} = 3
\text{degree sequence} = [3\ 3\ 3\ 3\ 2]

The question remains:

Are \( G_1 \) and \( G_2 \) isomorphic to each other?
Examples of isomorphic and non-isomorphic graphs

Both sets of edges are the same. \( G_1 \) and \( G_2 \) are isomorphic. 
(Verification: Sort all sets, compare items one by one.)
Are $G_1$ and $G_2$ isomorphic to each other?
Examples of isomorphic and non-isomorphic graphs

multisets of degrees of neighbours of nodes with degree 3:  
\{\{3 \ 2 \ 2\}\} // d  
\{3 \ 2 \ 2\} // f  
\{3 \ 3 \ 2\} // g  
\{3 \ 3 \ 2\} } // j

Another Invariant:  
\(G_1\) -- nodes of degree 3 form a connected subgraph.  
\(G_2\) -- nodes of degree 3 form two mutually unconnected subgraphs.

More invariants: Try yourself....
Is there a **fixed set of properties** which values can be calculated for any graph, no matter how effectively (linearly, polynomially, exponentially), and which values would decide whether two given graphs $G_1, G_2$ are isomorphic? In the sense:

$$\text{values calculated on } G_1 == \text{ values calculated on } G_2$$

if and only if

$$G_1 \text{ is isomorphic to } G_2$$

So far, no such set of properties is known.

It is also unknown whether it is **NP-hard/complete** to check if two graphs are isomorphic.

**Partial solution**

Advanced heuristical approaches solve the problem in many practical settings:

**SW:** nauty and Traces: https://pallini.di.uniroma1.it/

based on papers by Brendan D.McKay and Adolfo Piperno: *Practical graph isomorphism I and II.*

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.169.6684

https://arxiv.org/abs/1301.1493
Examples of isomorphic and non-isomorphic graphs

Isomorphism is difficult to confirm/reject when the graphs are highly symmetric. Informally, symmetry means that a graph "looks the same" in the vicinity of each node. The number of candidate bijections is then difficult to reduce when there are no obvious invariants which values would help to distinguish between different nodes. As a simple example, consider the following pair of graphs.

![Graphs](https://sagecell.sagemath.org)

```python
# Code for the graphs

g1 = graphs.CirculantGraph(19, [1,5,8]); g1.show()
g2 = graphs.CirculantGraph(19, [1,4,7]); g2.show()
```
Isomorphism of directed graphs

In these slides, term graph always refers to an undirected graph, if not specified otherwise.

All isomorphism properties, algorithms, notions, etc. defined for undirected graphs, can be analogously defined and analyzed/solved in analogous manner for directed graphs.

Two directed graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ are isomorphic if there is a bijection $f: V_1 \rightarrow V_2$ such that

$$\forall x, y \in V_1 : (f(x), f(y)) \in E_2 \iff (x, y) \in E_1$$

Example:

Graphs $G_1$ and $G_2$ are isomorphic, $G_3$ is not isomorphic to any of $G_1$, $G_2$. 
<table>
<thead>
<tr>
<th>N</th>
<th>Number f(N) of graphs on N nodes (incl. unconnected ones)</th>
<th><a href="https://oeis.org/A000088">https://oeis.org/A000088</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1044</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>12346</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>274668</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>12005168</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>31426485969804308768</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>645490122795799841856164638490742749440 ~ 6.5 \times 10^{38}</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>334494316309257669249439569928080028956631479935393064329967834887217734534880582749030521599504384 ~ 3.3 \times 10^{98}</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>7793841167914977954582550817575177766066055272533160501864210580719699592280766598762108507458913936081932965352037372886593259286753883857016383307981863462449691949358853053120648183808 \sim 7.8 \times 10^{186}</td>
<td></td>
</tr>
</tbody>
</table>

The formula approximates f(N) tightly, in the sense:

\[
\lim_{{N \to \infty}} \frac{f(N)}{\frac{N!}{2^{\binom{N}{2}}}} = 1
\]

Applying brute force and checking all graphs for would be a hopeless effort.
<table>
<thead>
<tr>
<th>N</th>
<th>Number $f'(N)$ of connected graphs on N nodes</th>
<th><a href="https://oeis.org/A001349">https://oeis.org/A001349</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>853</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1117</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>261080</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11716571</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>31397381142761241960</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>645465483198722799426731128794502283004</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>3344942976179029274740625889887714205924003404484971757354867875739197630926 64433461017585013705594</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>7793841167347901373159586190645563996131177435680973666982243627070377497235 4174178748323987582425416768805527046107079810797229883124475331332011126406 04192083672776028633590109166374659</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>asymptotically same as all graphs, in the sense: $\lim { N \to \infty, \ f'(N) / f(N) } = 1$</td>
<td></td>
</tr>
</tbody>
</table>

Applying brute force and checking all graphs for would be a hopeless effort.
<table>
<thead>
<tr>
<th>N</th>
<th>Number $f''(N)$ of undirected trees on N nodes</th>
<th><a href="https://oeis.org/A001349">https://oeis.org/A001349</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>7741</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>823065</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>$14830871802 \sim 1.5 \cdot 10^{10}$</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>$363990257783343 \sim 3.6 \cdot 10^{14}$</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>$630134658347465720563607281977639527019590$</td>
<td></td>
</tr>
</tbody>
</table>

Formula is too complex to fit here, see the OEIS reference above

Applying brute force and checking all trees would be a hopeless effort.
Examples of more graph invariants (a tiny! selection):

(.) Maximum/maximum node degree
(.) Degree sequence (sequence of all node degrees sorted in non-increasing order)
(.) Connected - yes/no
(.) Number of edges
(.) Bipartite - yes/no
(.) Regular - yes/no (the degree of all nodes is the same)
(.) Tree - yes/no
(.) Planar - yes/no (can be drawn in a plane without edges crossing)
(X) Diameter, radius, eccentricity, number of centers
(X) Number of triangles
(X) Length of the shortest cycle (so called girth of the graph)
(.) Number of bridges/cutvertices/blocks
(X) Hamiltonian - yes/no (Hamilton path or cycle exists in the graph)
(X) Spectrum (= multiset of eigenvalues) of adjacency (Laplacian) matrix of the graph
(X) Number of automorphisms
(X) Chromatic/independence/dominancy/clique numbers (see respective definitions...)
...
...
(.) $O(E+V)$, (X) more complex than $O(E+V)$, polynomial or exponential.
Two random graphs are extremely(!) probably NOT isomorphic

When two graphs $G_1, G_2$ are selected randomly from the set of all graphs on $N$ nodes or when they are generated randomly, then

A. The probability that $G_1$ and $G_2$ are isomorphic is very close to 0. *)
B. The probability that the values of some (in fact, of many) of invariants in $G_1$ and $G_2$ are different is very close to 1.

A. $\equiv$ Very probably, $G_1$ and $G_2$ are not isomorphic.
B. $\equiv$ Very probably, it is (relatively) easy to verify $G_1$ and $G_2$ are not isomorphic.

Conclusion:
When the graphs are not isomorphic,
checking the values of various (easy to compute, preferentially! ) invariants in both graphs,
quickly confirms the fact in majority of (random) cases.

*) How close? The probability $p$ is in the order of $\frac{n!}{2^{\binom{n}{2}}}$.
For example, $n = 10, p = 10! / 2^{45} \approx 10^{-7}; \quad n = 100, p = 100! / 2^{4950} \approx 10^{-1332}$. 

PAL 2020/04 Graph isomorphism  notes
Tree certificate example

[Diagram of graphs with nodes and edges labeled with bit strings, showing nondecreasing lexicographic order.]

PAL 2020/04 Graph isomorphism notes
Tree certificate example

\[ \begin{array}{cccc}
01 & 01 & 01 & 01 \\
\text{d} & \text{c} & \text{b} & \text{a} \\
01 & 01 & 01 & 01 \\
\text{g} & \text{f} & \text{e} & \text{i} \\
01 & 01 & 01 & 01 \\
\text{k} & \text{j} & \text{h} & \text{i} \\
01 & 01 & 01 & 01 \\
0000101110001101100111 \\
\end{array} \]
Perform DFS from the root == center of the tree. Always expand DFS into that subtree which certificate is lexicographically the smallest.

Output 0 when the node is being open and output 1 when the node is being closed.

The output sequence is the tree certificate, it is obvious by induction.

Drawback: DFS cannot know the subtrees certificates in advance.

The idea can be used only for reconstructing the tree from the certificate.
**proc** reconstructTree( certificate )

nodesList = emptyList()
edgesList = emptyList()
centers = emptyList()  // one or two centers
stack = emptyStack()

for digit in certificate
  if digit == '0'
    create node X
    nodesList.add( X )
    if stack.isEmpty()
      centers.add( X )
    else
      edgesList.add( pair(stack.top(),X) )
      stack.push( X )
  else  // digit == '1'
    stack.pop()

if centers.size() == 2  // two centers
  edgesList.add( pair(centers[0],centers[1]) )
return  nodesList, edgesList, centers
def reconstruct( certificate ):
    nodes, edges, stack = [], [], []
    centers = [] # 1 or 2 centers
    newNode = 0 # nodes are integers

    for digit in certificate:
        if digit == '0':
            newNode += 1 # 'create' new node
            nodes.append( newNode )
            if len( stack ) == 0: # empty
                centers.append( newNode )
            else:
                edges.append( [newNode, stack[-1]] )
                stack.append( newNode )
        else: # digit == '1':
            stack.pop()

        if len( centers ) == 2:
            edges.append( [centers[0], centers[1]] )

    return nodes, edges, centers

cer = "0000101110001101110000111011"
nodes, edges, centers = reconstruct( cer )