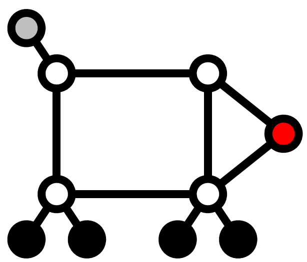
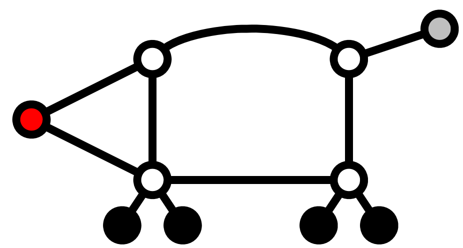
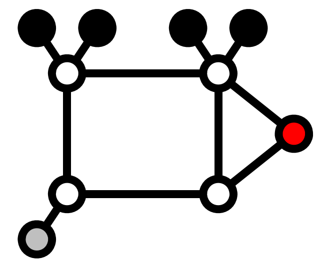
Isomorphism motivation



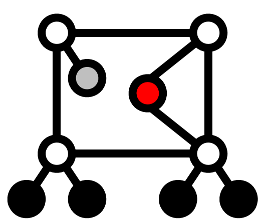
Tiny piglet



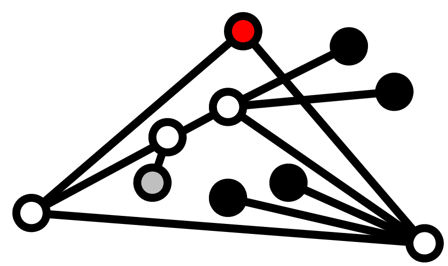
Getting strong and backwards



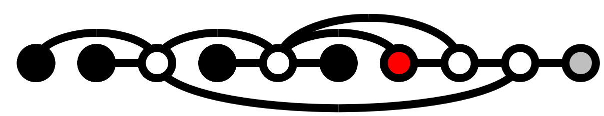
Reflected in water



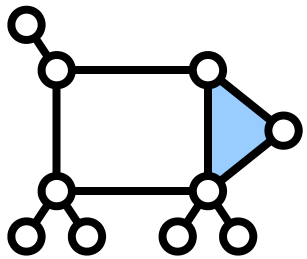
Inspecting its tail



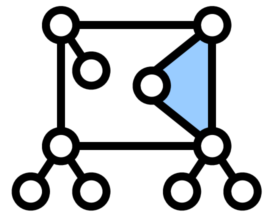
Painted by abstract artist



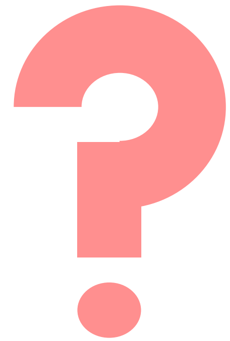
Serialized in Java file



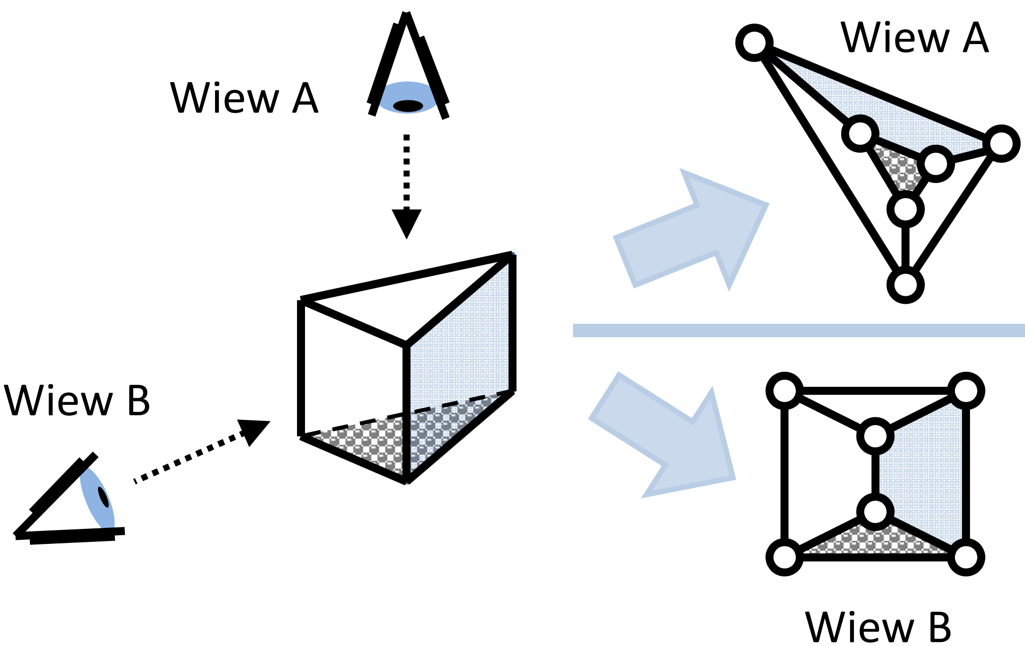
Triangle **outside** the square



Triangle **inside** the square



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

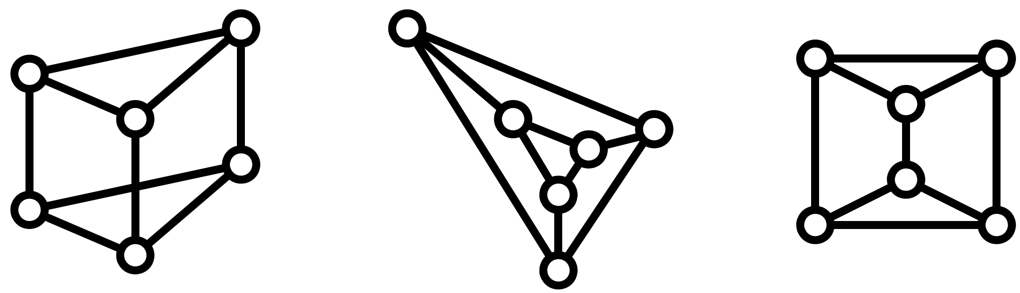


Wiew A
One triangle and **three** quadrilaterals inside a **triangle**.

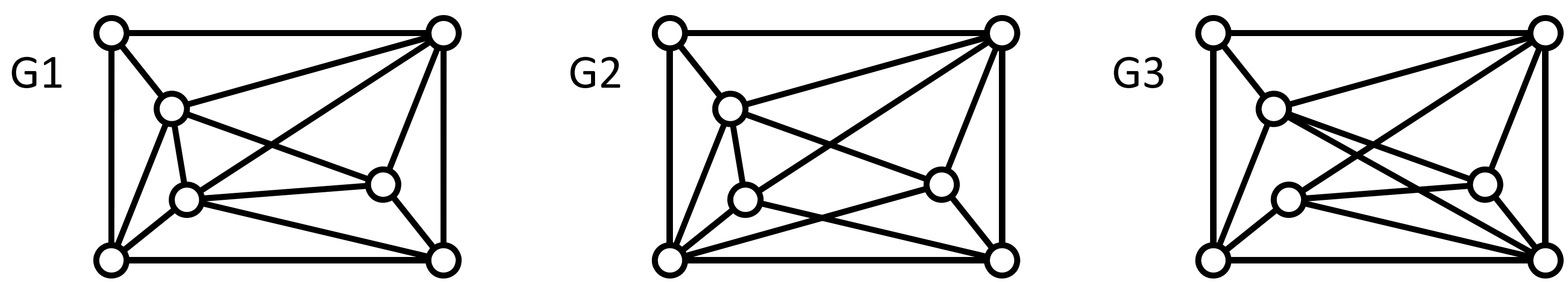
Different representations (views) of **the same** original structure.

Wiew B
Two triangles and **two** quadrilaterals inside a **rectangle**.

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:



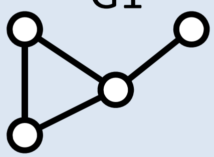
G1 and G2 do not depict the same structure, while G1 and G3 do.

The structure, in this context, is the set of nodes and the set of edges between them.

Isomorphism informally

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).

G1



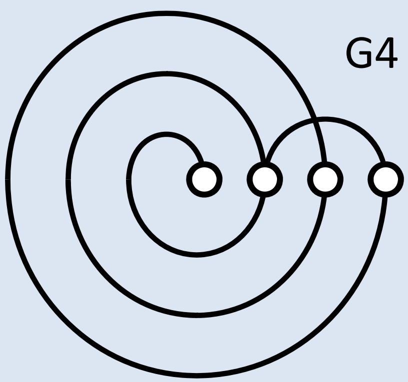
// Plain vanilla
// scheme

G2 = ({a,b,c,d},
{ {a,b}, {b,c}, {c,a}, {b,d} })
// Set of nodes and set of edges

G3: $\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

// Adjacency matrix of G3

G4



// Posh garden
// scheme

Clearly, G1, G2, G3, G4, G5, are all pairwise isomorphic to each other.

Each time it is the same graph.

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

Isomorphism formally

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 and 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 then nodes of G! 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 ?

Isomorphism formally

Let G be a graph. A one-to-one correspondence between the nodes of G is called a automorphism of G when

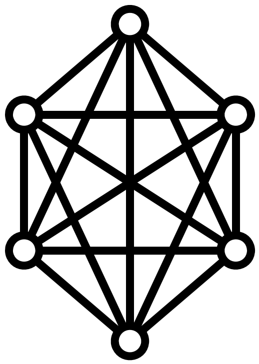
There is and edge between x and y in G1
if and only if
there is an edge between nodes corresponding to x and y in G2

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

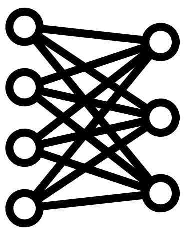
The one-to-one correspondence between the nodes of a graph is a permutation of the nodes.

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

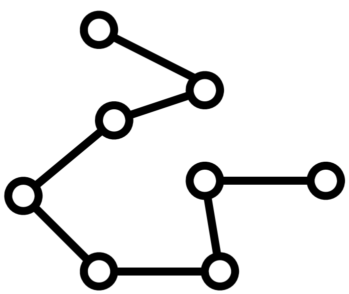
Isomorphism formally



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.

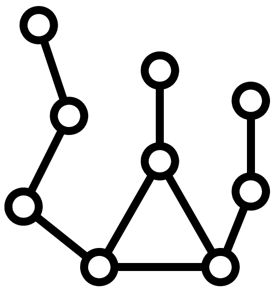


On a complete bipartite graph on M and N nodes, there are M! × N! automorphisms, any permutation which maps a node to another node in the same partition is an automorphisms. For M = 4, N = 3, there are 4! × 3! = 24 × 6 = 144 different automorphisms.



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.

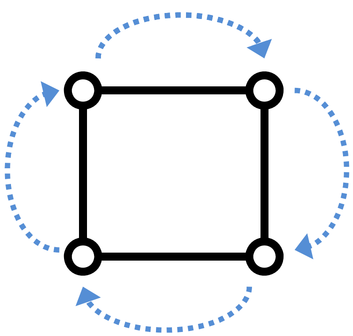
Isomorphism formally



On many graphs, there is only one automorphism, represented by the identity permutation of the nodes.

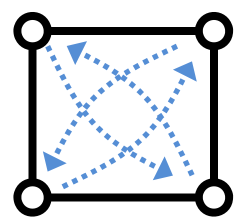
In general, the composition of two automorphisms is another automorphism (it is a composition of permutations), and the set of automorphisms of a given graph, under the composition operation, forms a group, the automorphism group of the graph.

one automorphism



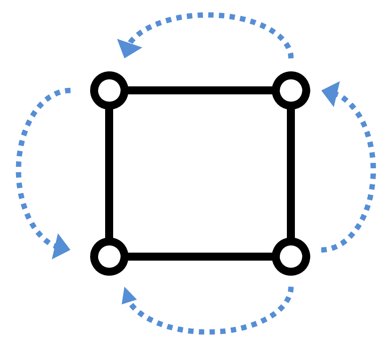
composed with

another automorphism

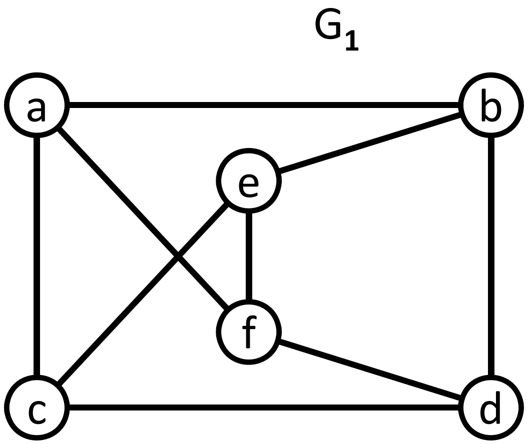


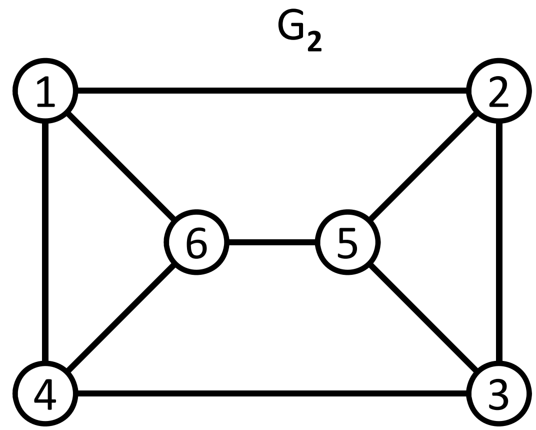
yields

a third automorphism



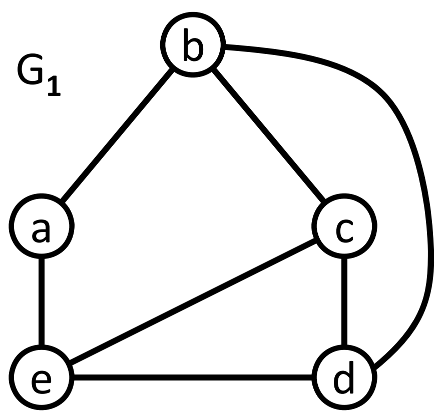
Examples of isomorphic and non-isomorphic graphs

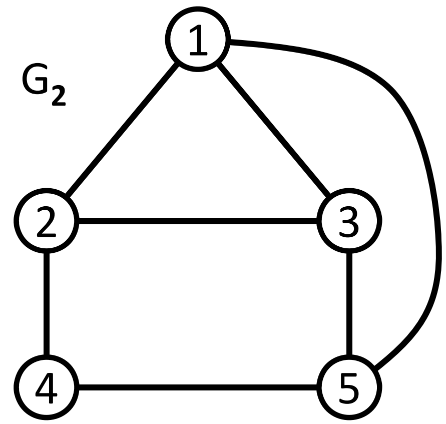




G_1	G_2
|nodes| = 6	|nodes| = 6
|edges| = 9	|edges| = 9
is regular = true	is regular = true
max degree = 3	max degree = 3
diameter = 2	diameter = 2
no. of triangles = 0	no. of triangles = 2 (triangles 1-4-6 and 2-3-5)

Examples of isomorphic and non-isomorphic graphs





$\lvert V(G_1)\rvert = 5$	$\lvert V(G_2)\rvert = 5$
$\lvert E(G_1)\rvert = 6$	$\lvert E(G_2)\rvert = 6$
min degree = 2	min degree = 2
max degree = 3	max degree = 3
degree sequence = [3 3 3 3 2]	degree sequence = [3 3 3 3 2]
...	...
etc.	etc.

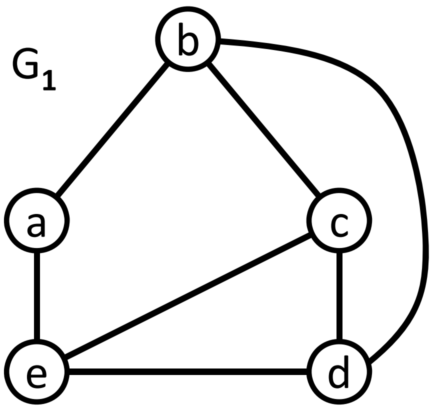
The question remains: ?

Are G_1 and G_2 isomorphic to each other?

?

Examples of isomorphic and non-isomorphic graphs

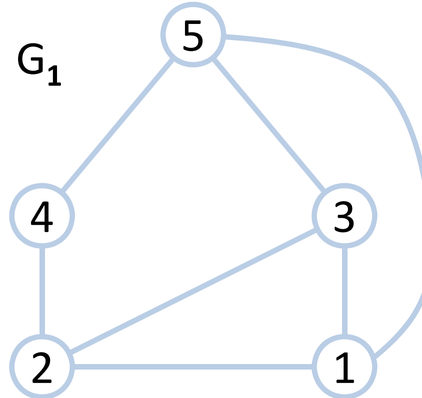
G_1



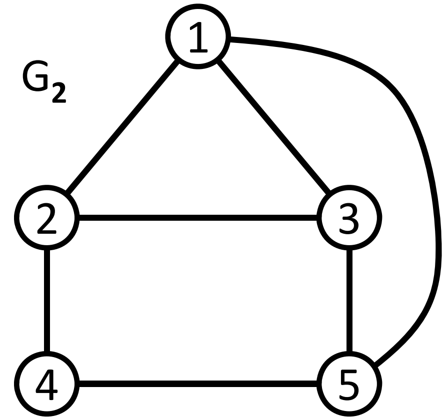
Nodes mapping

a --- 4
b --- 5
c --- 3
d --- 1
e --- 2

G_1



G_2



G_1: Set of edges:

{ {a, e}
{a, b}
{b, c}
{c, e}
{e, d}
{c, d}
{b, d} }

Nodes mapping

a --- 4
b --- 5
c --- 3
d --- 1
e --- 2

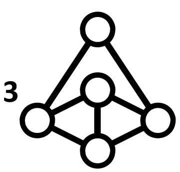
G_1: Set of mapped edges:

{ {4, 2}
{4, 5}
{5, 3}
{3, 2}
{2, 1}
{3, 1}
{5, 1} }

G_2: Set of edges:

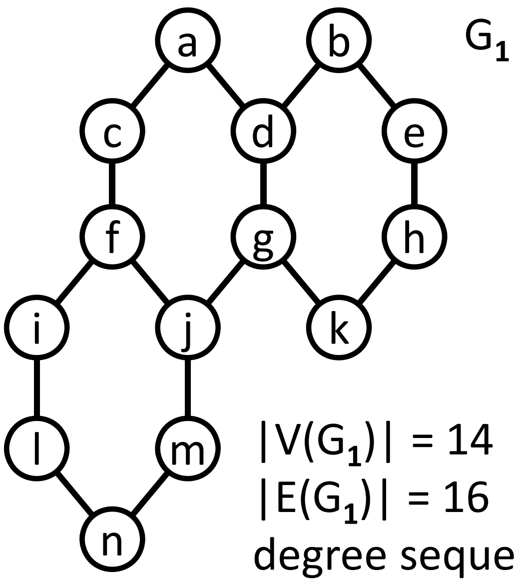
{ {1, 2}
{1, 3}
{1, 5}
{2, 3}
{2, 4}
{3, 5}
{4, 5} }

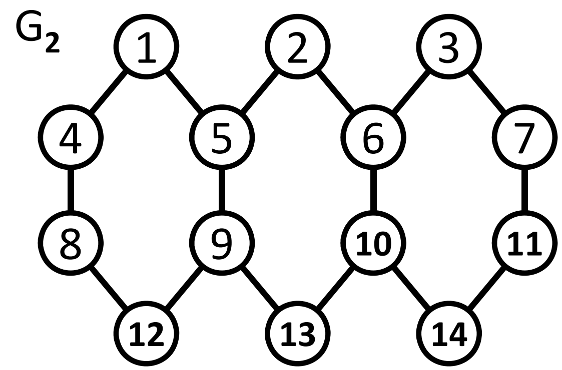
G_3



Both sets of edges are the same. G_1 and G_2 are isomorphic.
(Verification: Sort all sets, compare items one by one.)

Examples of isomorphic and non-isomorphic graphs





$\lvert V(G_1)\rvert = 14$	$\lvert V(G_2)\rvert = 14$
$\lvert E(G_1)\rvert = 16$	$\lvert E(G_2)\rvert = 16$
degree sequence = [3 3 3 3 2 2 2 2 2 2 2 2 2 2]	degree sequence = [3 3 3 3 2 2 2 2 2 2 2 2 2 2]
diameter = 7, (distance(l, e))	diameter = 7, (distance(4, 11))
isBipartite = yes	isBipartite = yes
...	...

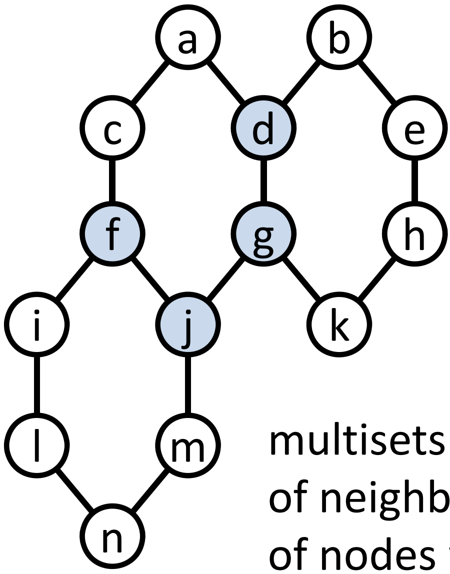
The question remains:

?

Are G_1 and G_2 isomorphic to each other?

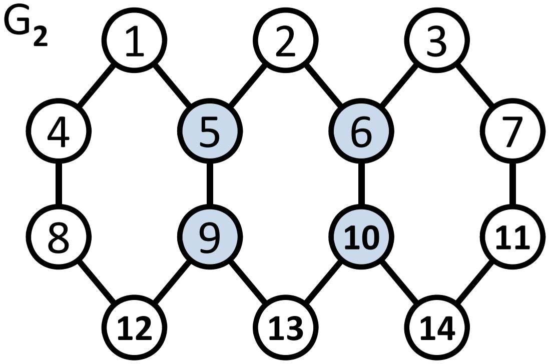
?

Examples of isomorphic and non-isomorphic graphs



G_1

G_2



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

multisets of degrees
of neighbours
of nodes with degree 3:
{ {3 2 2} // 5
{3 2 2} // 6
{3 2 2} // 9
{3 2 2} } // 10

G_1 and G_2 are not isomorphic to each other.

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....

Difficulty

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:

values calculated on G_1 == values calculated on G_2
if and only if
G_1 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.

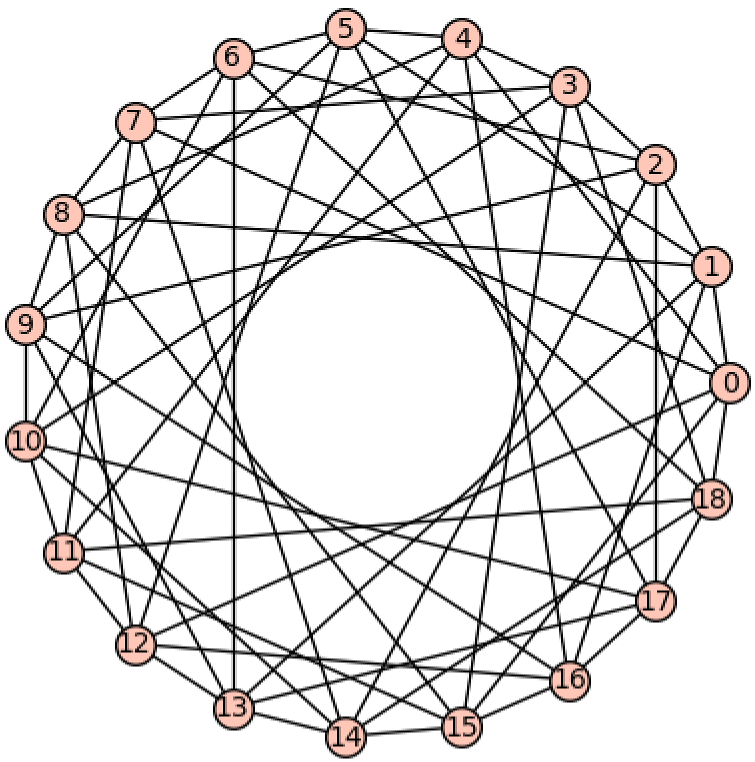
Partial solution

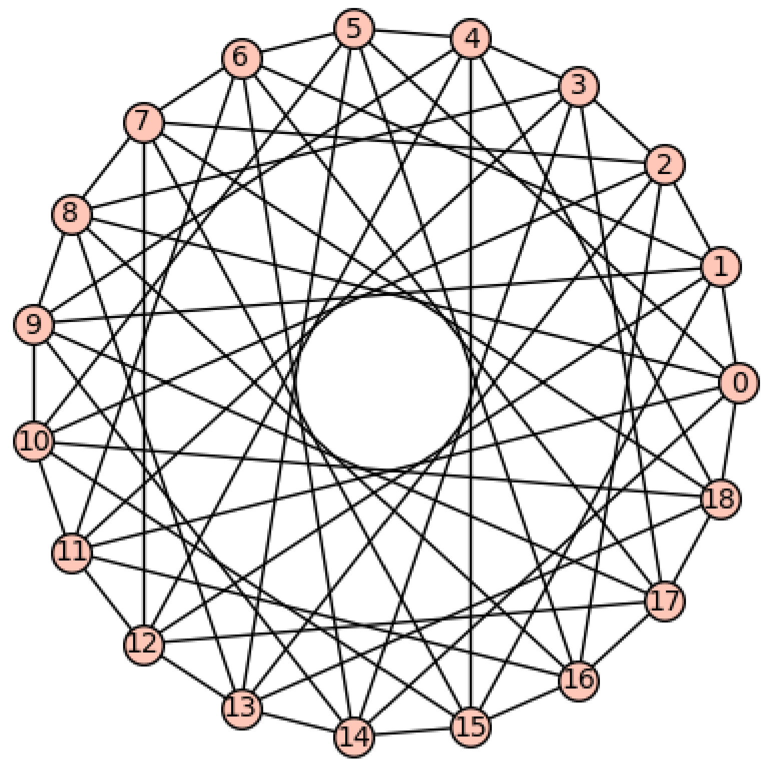
Advanced heuristical approaches solve the problem in many practical settings:
SW: **nauty** and **Traces:** https://pallini.di.uniroma1.it/

based on papers by Brendan D.McKay and AdolfoPiperno: *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.





Picture credit to https://sagecell.sagemath.org and code

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

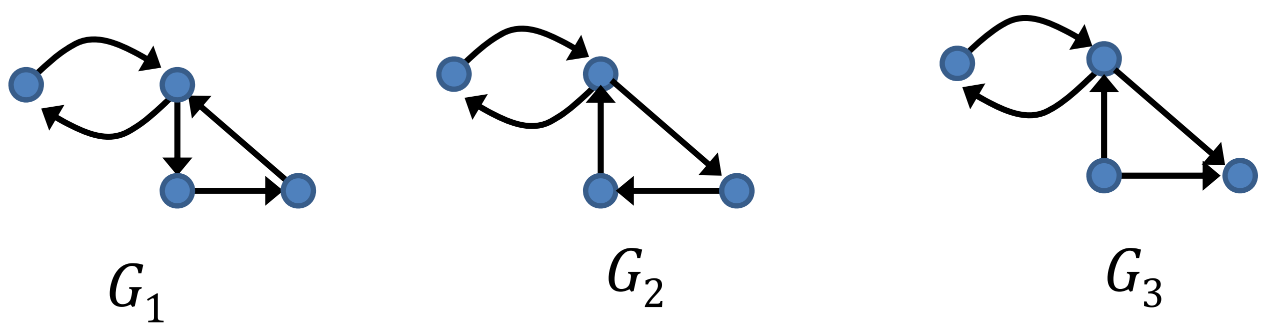
In these slides, term graph always refers to an undirected graph, if not specified otherwise.

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

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

Example:



Graphs G_1 and G_2 are isomorphic, G_3 is not isomorphic to any of G_1, G_2.

N	Number f(N) of graphs on N nodes (incl. unconnected ones) https://oeis.org/A000088
1	1
2	2
3	4
4	11
5	34
6	156
7	1044
8	12346
9	274668
10	12005168
15	31426485969804308768
20	645490122795799841856164638490742749440 ~ $6.5 \cdot 10^{38}$
30	33449431630925766924943956992808002895663147993539306432996783488721773453488058274903052159950 4384 ~ $3.3 \cdot 10^{98}$
40	779384116791497795458255081757517776606605527253316050186421058071969959228076659876210850745891393608193296535203737288659325928675388385701638330798186346244969194935885305312064818380 8 ~ $7.8 \cdot 10^{186}$
N	see inset

Approximation of the number of graphs:

$$f(N) \le \frac{2^{\binom{N}{2}}}{N!}$$

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

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

Applying brute force and checking all graphs for would be a hopeless effort.

N	Number f'(N) of *connected* graphs on N nodes https://oeis.org/A001349
1	1
2	1
3	2
4	6
5	21
6	112
7	853
8	11117
9	261080
10	11716571
15	31397381142761241960
20	645465483198722799426731128794502283004
30	334494297617902927474062588988771420592400340448497175735486787573919763092664433461017585013705594
40	7793841167347901373159586190645563996131177435680973666982243627070377497235417417874832398758242541676880552704610707981079722988312447533133201112640604192083672776028633590109166374659
N	asymptotically same as all graphs, in the sense: $\lim \{ N \to \infty ,\ f'(N) / f(N) \} = 1$

Applying brute force and checking all graphs for would be a hopeless effort.

N	Number f''(N) of undirected trees on N nodes https://oeis.org/A001349
1	1
2	1
3	1
4	2
5	3
6	6
7	11
8	23
9	47
10	106
15	7741
20	823065
30	14830871802 ~ $1.5 \cdot 10^{10}$
40	363990257783343 ~ $3.6 \cdot 10^{14}$
100	5 6301346583474657205636072819776395270195 90
N	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 G1, G2 are selected randomly from the set of all graphs on N nodes or when they are generated randomly, then

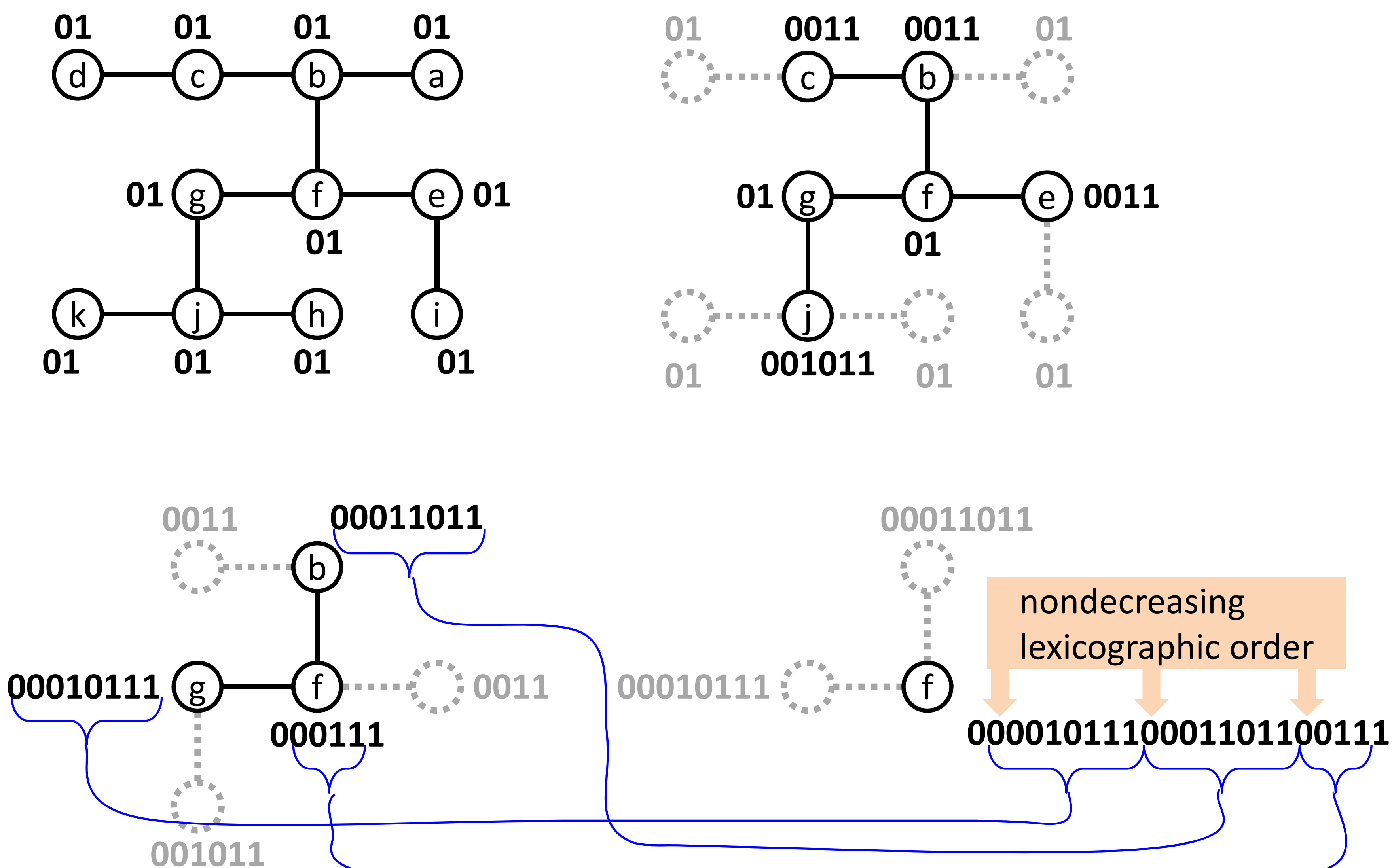
A. The probability that G1 and G2 are isomorphic is very close to 0. *)
B. The probability that the values of some (in fact, of many) of invariants in G1 and G2 are different is very close to 1.

A. $\equiv$ Very probably, G1 and G2 are not isomorphic.
B. $\equiv$ Very probably, it is (relatively) easy to verify G1 and G2 are not isomorphic .

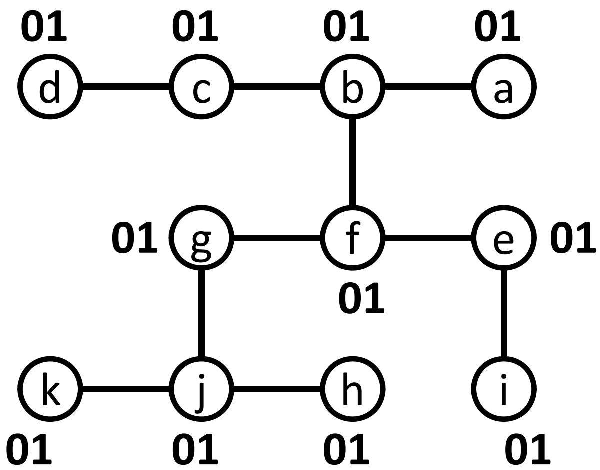
Conclusion:
When the graphs are not isomorphic,
checking the values of various (easy to compute, preferentially!) invariants in both graphs, quickly confirms the fact in majority of (random) cases.

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

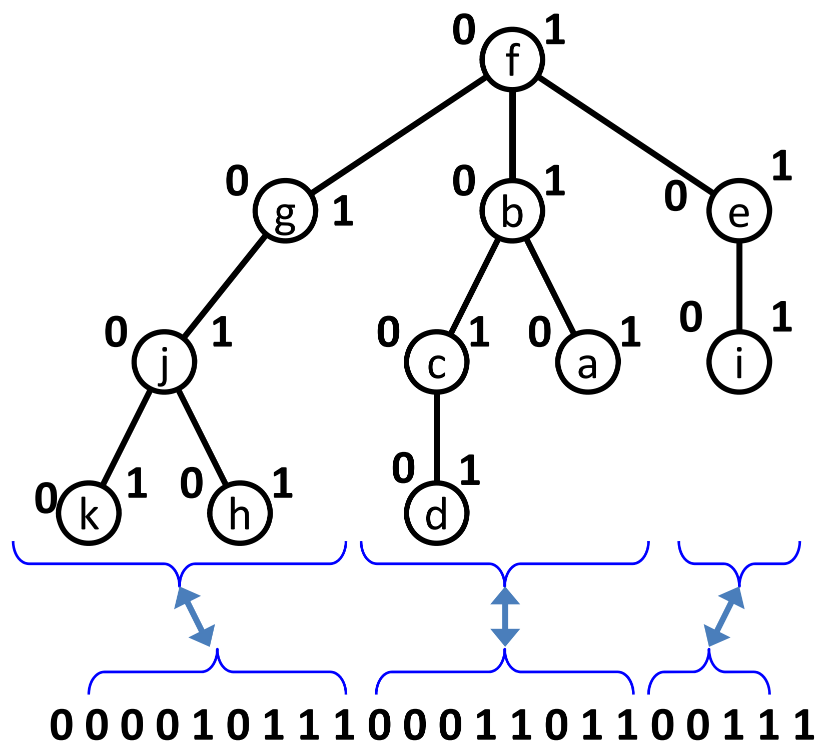
Tree certificate example



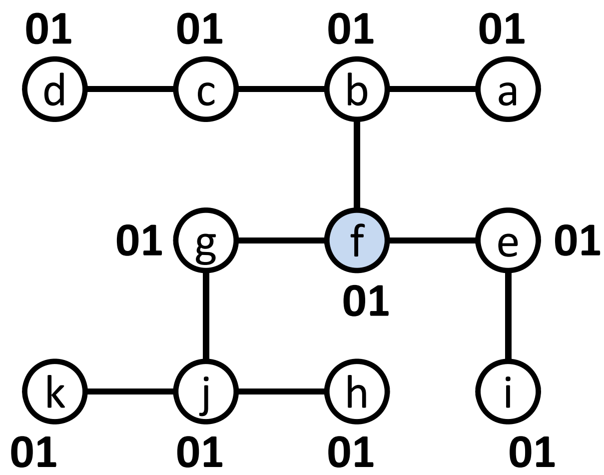
Tree certificate example



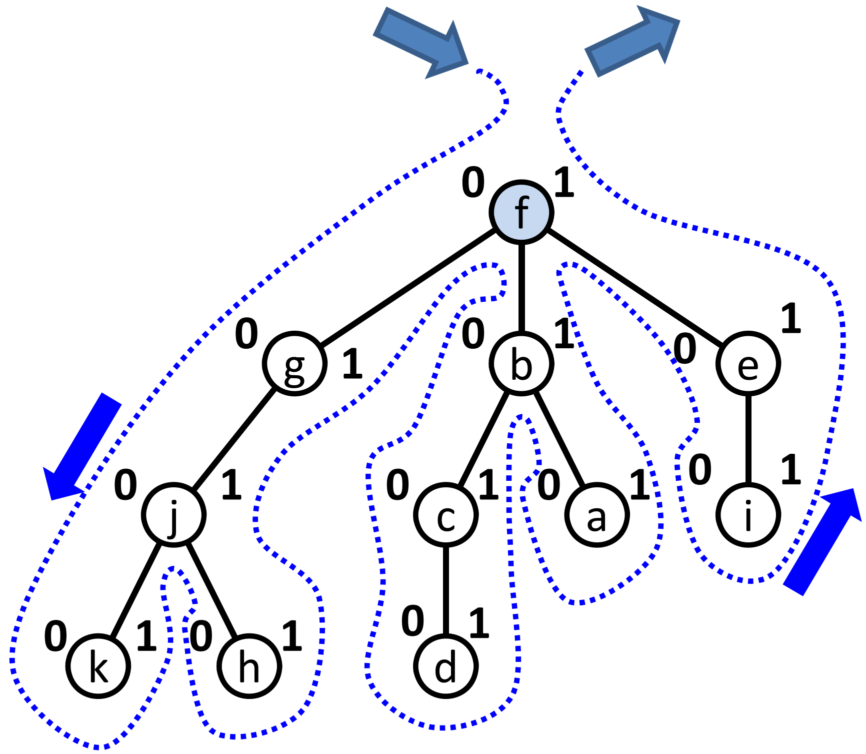
0000101110001101100111



Tree certificate example



0000101110001101100111



0 0 0 0 1 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 1 1

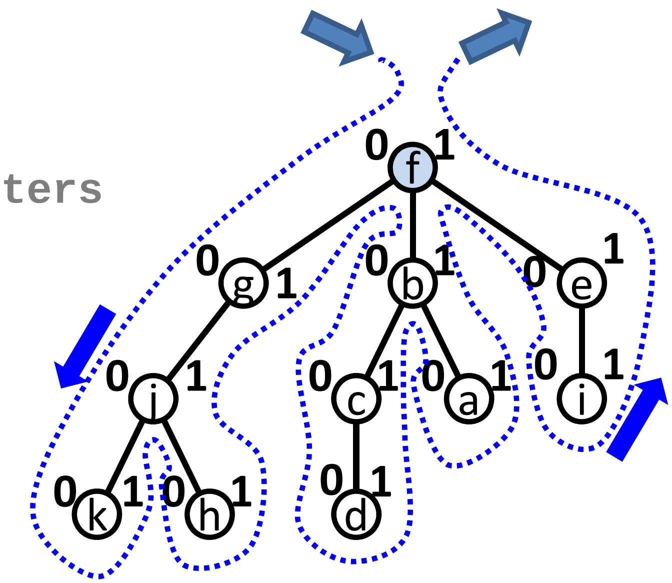
- Perform DFS from the root == center of the tree. Always expand DFS into that subtree which certificate is lexicographically the smallest.
- Output 0 when the node is being open and output 1 when the node is being closed.
- The output sequence is the tree certificate, it is obvious by induction.
- Drawback: DFS cannot know the subtrees certificates in advance.
- The idea can be used only for reconstructing the tree from the certificate.

Tree certificate example

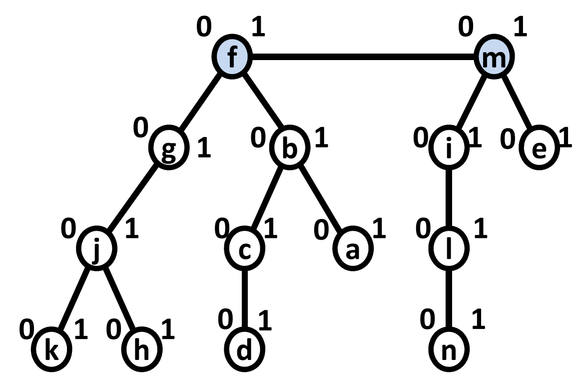
```
proc reconstructTree( certificate )
 nodesList = emptyList()
 edgesList = emptyList()
 centers = emptyList()  // one or two centers
 stack = emptyStack()

 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
```



0 0 0 0 1 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 1 1



0000101110001101110000111011

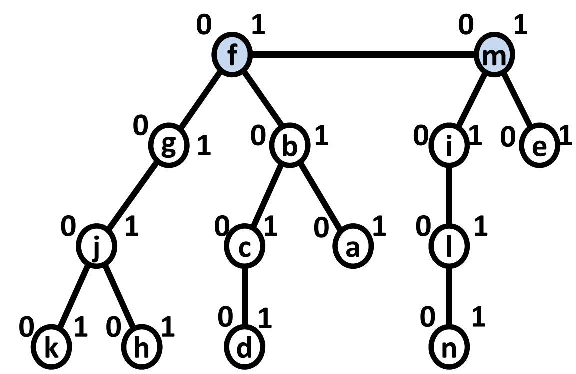
Tree certificate example - Python reconstruction

```python
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 )
```



0000101110001101110000111011