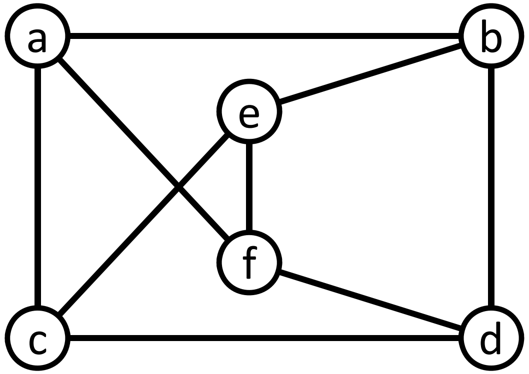
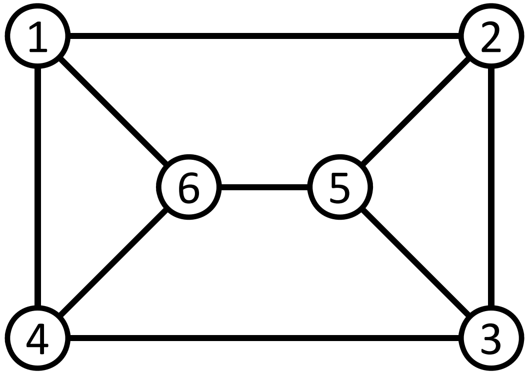
Examples of isomorphic and non-isomorphic graphs

G_1

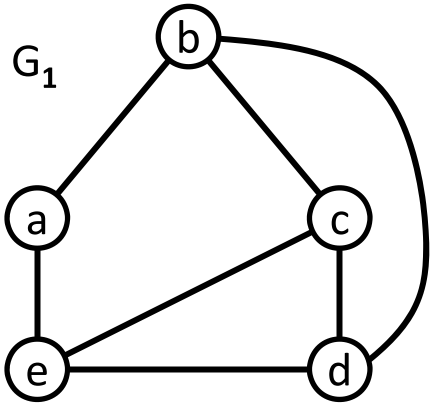


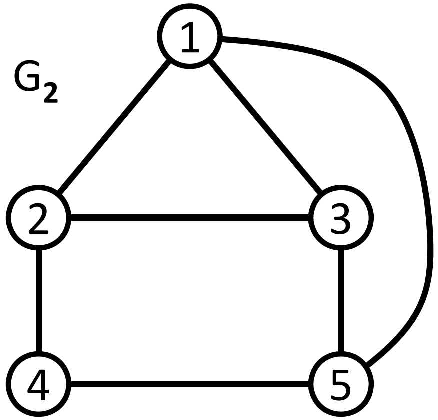
G_2



G_1	G_2
$|V(G_1)| = 6$	$|V(G_2)| = 6$
$|E(G_1)| = 9$	$|E(G_2)| = 9$
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 a-e-c and b-d-f)

Examples of isomorphic and non-isomorphic graphs





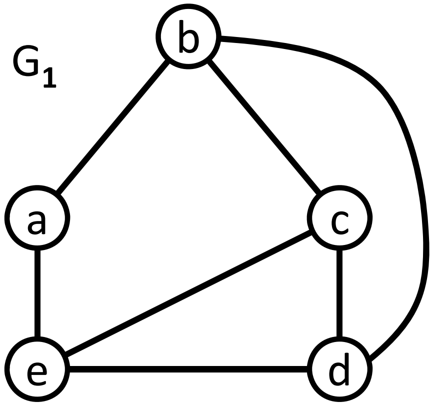
G_1		G_2
$\lvert V(G_1) \rvert = 5$	↔	$\lvert V(G_2) \rvert = 5$
$\lvert E(G_1) \rvert = 6$	↔	$\lvert E(G_2) \rvert = 6$
min degree = 2	↔	min degree = 2
max degree = 3	↔	max degree = 3
degree sequence = [3 3 3 3 2]	↔	degree sequence = [3 3 3 3 2]
...		...
etc.		etc.

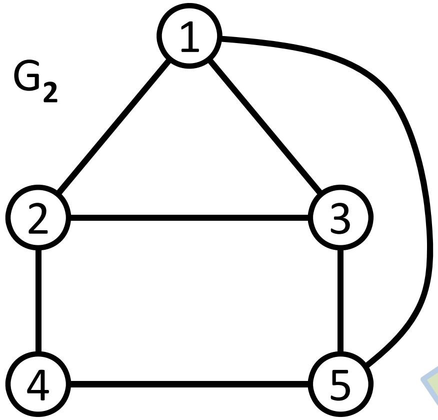
The question remains: ?

Are G_1 and G_2 isomorphic to each other?

?

Examples of isomorphic and non-isomorphic graphs





G_1: 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

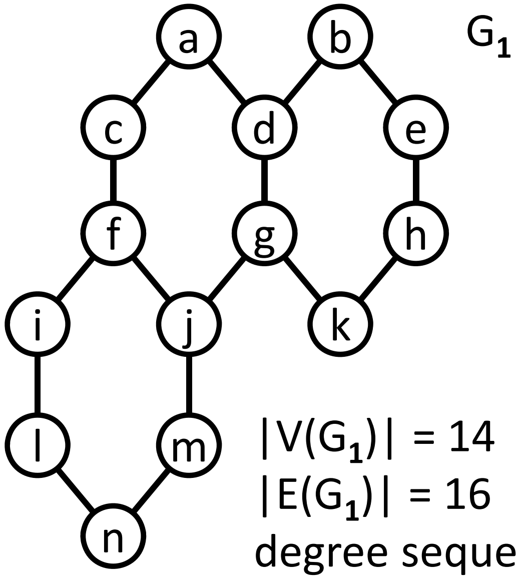
G_1: Set of mapped edges:

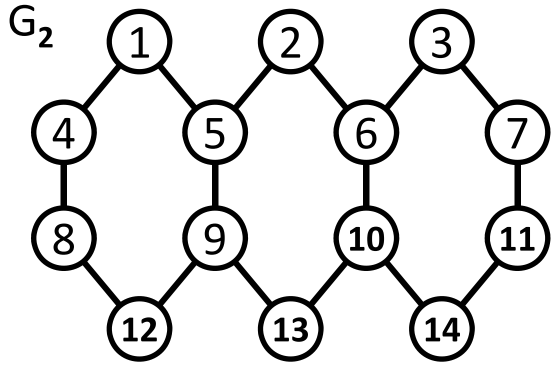
{ {4 2}
{4 5}
{5 3}
{3 2}
{2 1}
{3 1}
{5 1} }

G_2: Set of edges:

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

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





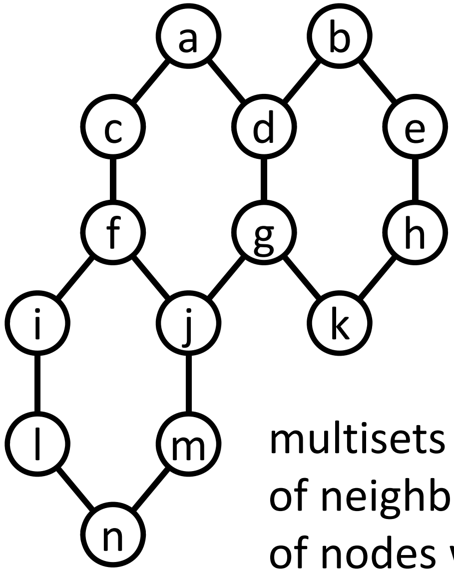
G_1		G_2
$\lvert V(G_1)\rvert = 14$	↔	$\lvert V(G_2)\rvert = 14$
$\lvert E(G_1)\rvert = 16$	↔	$\lvert E(G_2)\rvert = 16$
degree sequence = [3 3 3 3 2 2 2 2 2 2 2 2 2 2]	↔	degree sequence = [3 3 3 3 2 2 2 2 2 2 2 2 2 2]
diameter = 7, (distance(l, e))	↔	diameter = 7, (distance(4, 11))
isBipartite = yes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

The question remains: ?

Are G_1 and G_2 isomorphic to each other?

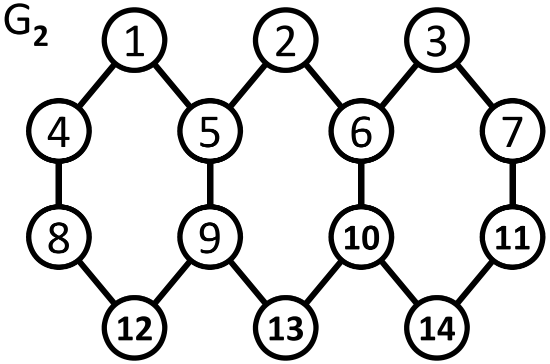
?

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:

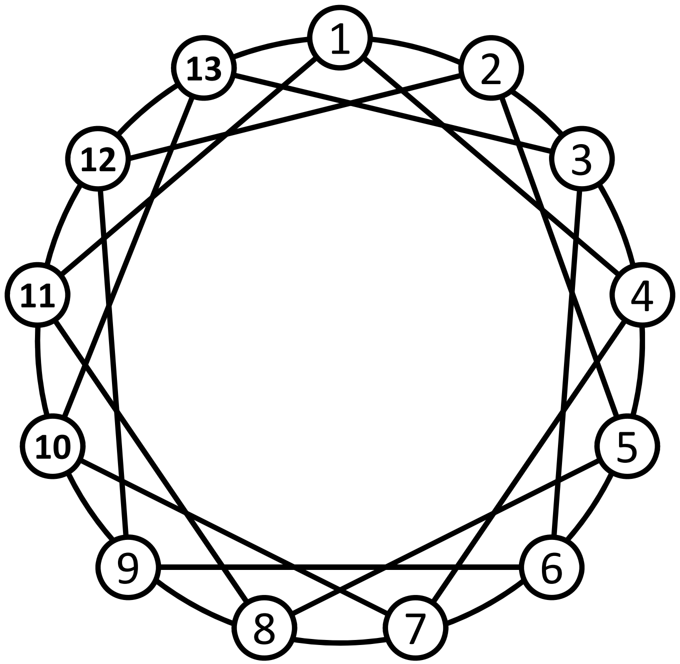
G_1 -- nodes of degree 3
form a connected subgraph.

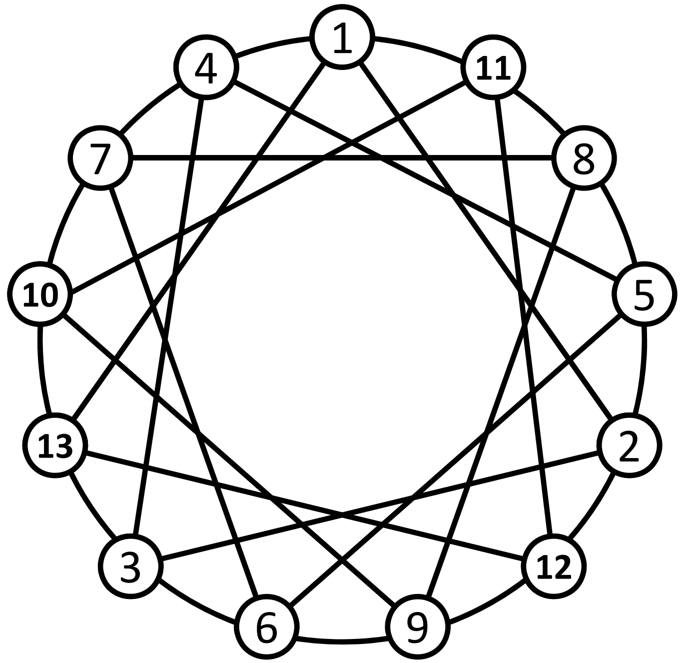
G_2 -- nodes of degree 3
form two mutually unconnected subgraphs.

More invariants: Try yourself....

Examples of isomorphic and non-isomorphic graphs

Isomorphism is difficult to confirm/reject when the graphs are highly symmetric. Informally, it means that the graphs "look the same", both globally and also locally in the vicinity of any particular node. The number of candidate bijections is then difficult to reduce as there is no obvious invariant which values would help to distinguish between different nodes. As a simple example, consider the following pair of isomorphic graphs.





Isomorphism of directed graphs

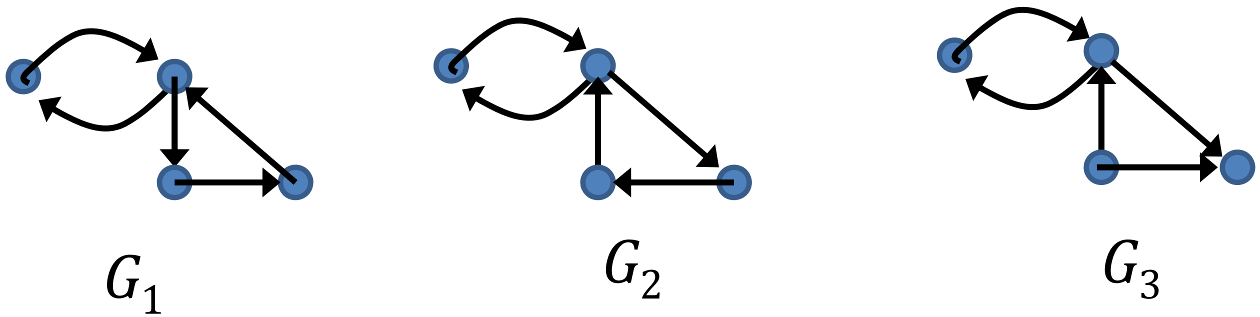
In these slides, term graph always refers to an undirected graph, if not specified otherwise.

All isomrphism 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 $f: V_1 \to V_2$ 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.

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
30	334494316309257669249439569928080028956631479935393064329967834887217734534880582749030521599504384
40	7793841167914977954582550817575177766066055272533160501864210580719699592280766598762108507458913936081932965352037372886593259286753883857016383307981863462449691949358853053120648183808
N	f(N) ~ (2^comb(N, 2)) / N!, in the sense: lim { N→∞ , f(N) / ((2^COMB(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 8 9 10
7	853
8	11117
9	261080
10	11716571
15	31397381142761241960
20	645465483198722799426731128794502283004
30	334494297617902927474062588988771420592400340448497175735486787573919763092 64433461017585013705594
40	7793841167347901373159586190645563996131177435680973666982243627070377497235 4174178748323987582425416768805527046107079810797229883124475331332011126406 04192083672776028633590109166374659
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
40	363990257783343
100	6301346583474657205636072819776395270195 90
N	too complex to fit here, see reference above

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

Examples of more graph invariants (a small selection):

(.) Connected - yes/no
(.) 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) Hamiltonian - yes/no (Hamilton path or cycle exists in the graph)
(.) Number of edges
(.) Maximum node degree
(.) Minimum node degree
(.) Number of nodes with maximum (minimum degree)
(.) Degree sequence (sequence of all node degrees sorted in non-increasing order)
(X) Spectrum (= multiset of eigenvalues) of adjacency (Laplacian) matrix of the graph
(X) Length of the shortest cycle (so called *girth* of the graph)
(X) Number of triangles
(.) Number of bridges/cutvertices/blocks
(X) Number of automorphisms
(X) Chromatic/independence/dominancy/clique numbers (see respective definitions...)
(X) Diameter/excentricity/number of centers
(X) Bandwidth
...

(.) *O(E+V),* (X) *more complex than O(E+V), polynomial or exponential.*

When two graphs G1, G2 are selected randomly form the set of all graphs on N nodes or when they are generated randomly, then

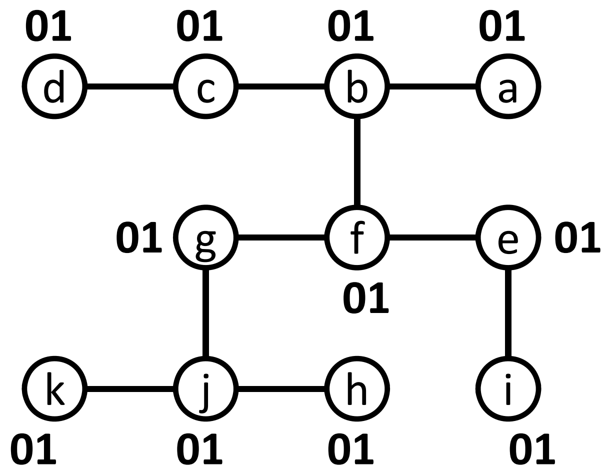
A. The probability that G1 and G2 are isomorphic is very close to 0. *)
B. The probability that the values of some (in fact, of many) of invariants in G1 and G2 are different is very close to 1.

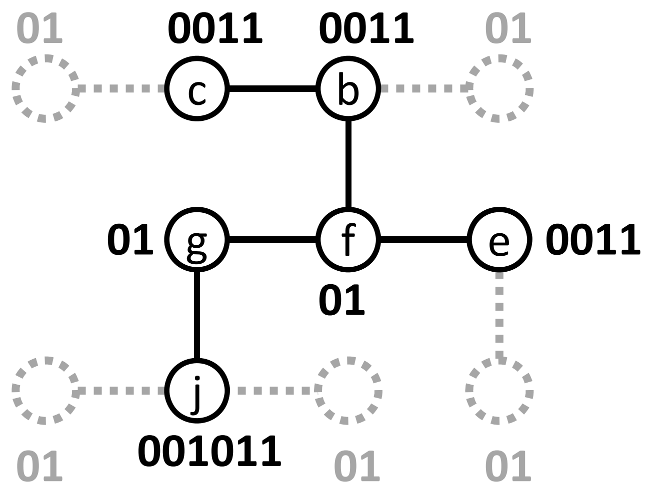
A. $\equiv$ Very probably, the G1 and G2 are not isomorphic.
B. $\equiv$ Very probably, it is (relatively) easy to verify G1 and G2 are not isomorphic .

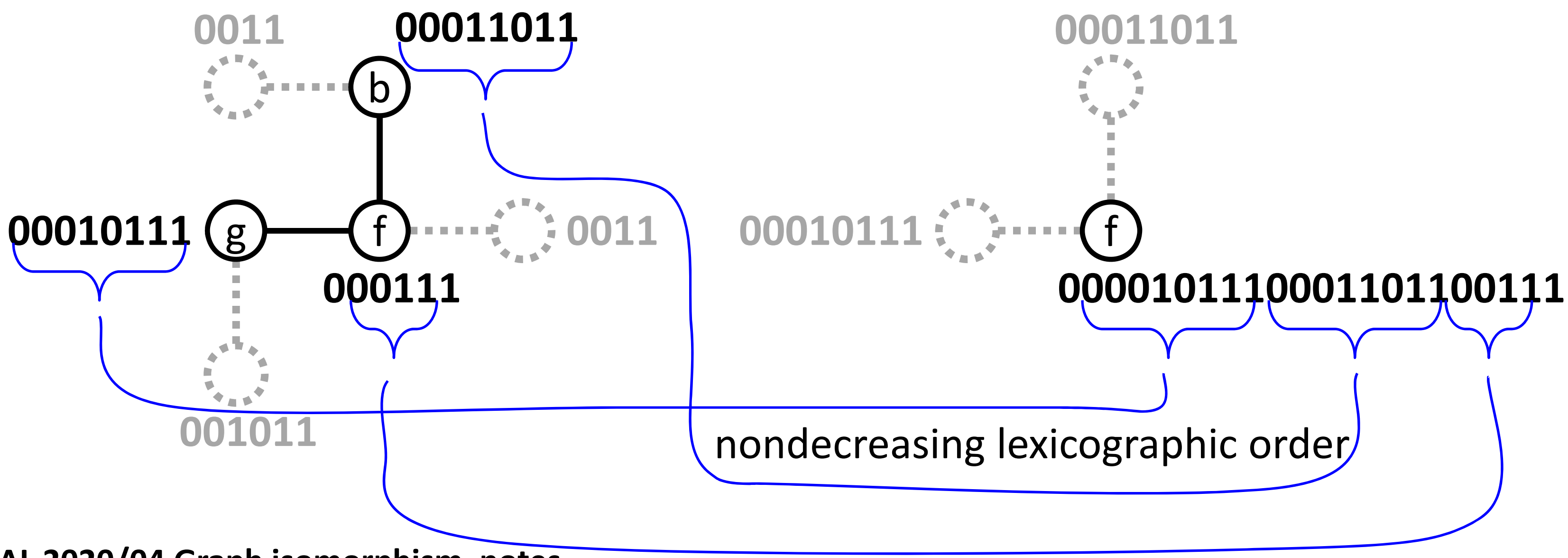
Conclusion:
When the graphs are not isomorphic,
checking the values of various (easy to compute, preferentially!) invariants in both graphs, quickly confirms the fact in majority of (random) cases.

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

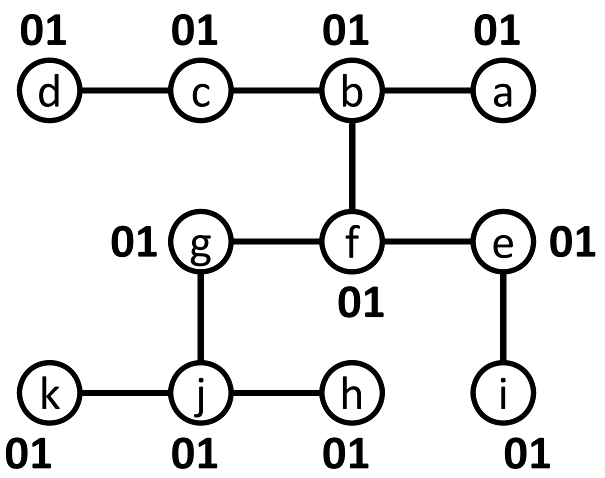
Tree certificate example



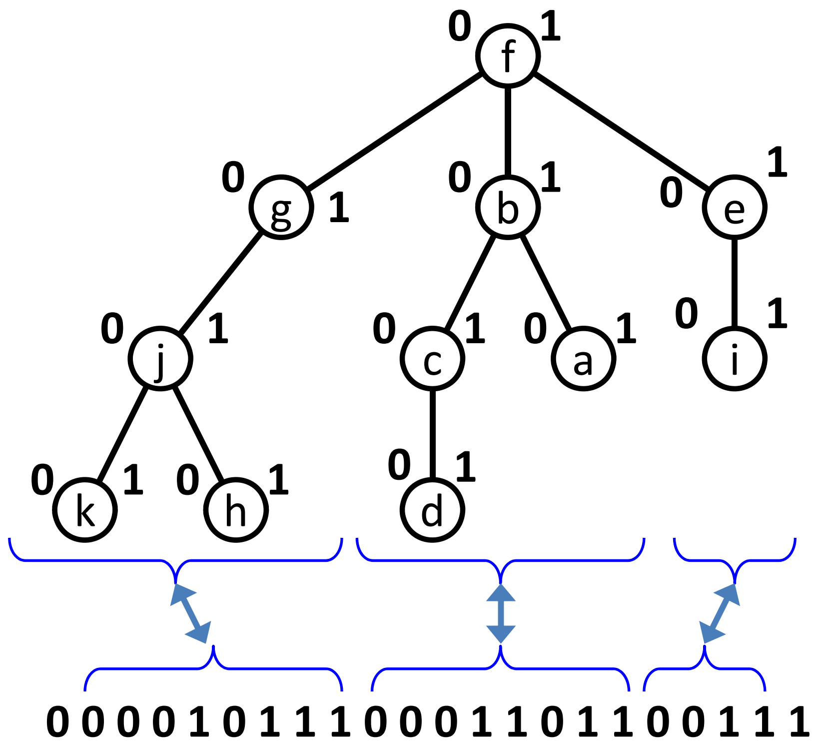




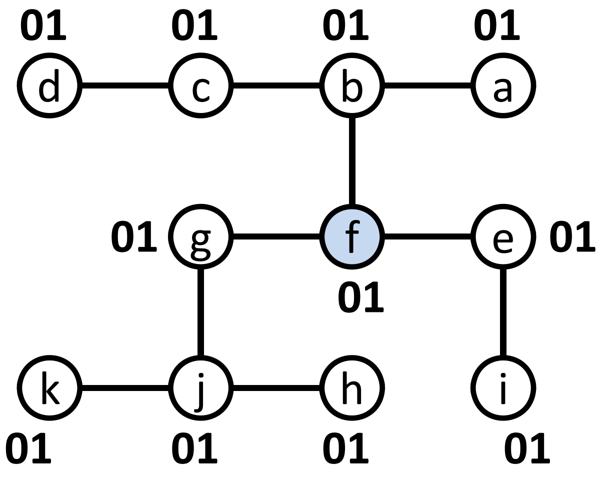
Tree certificate example



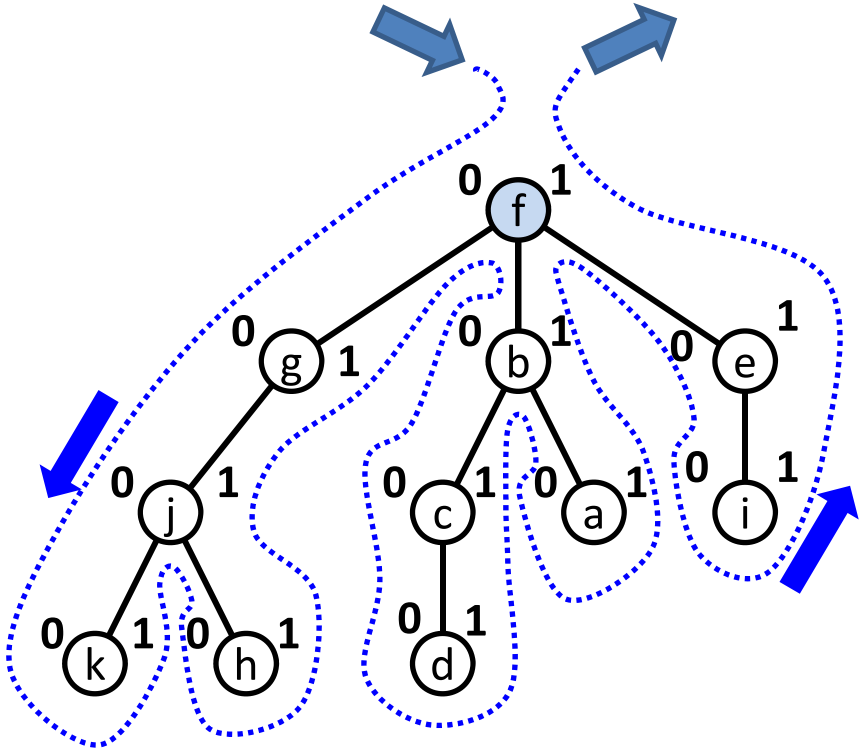
000010111000110110011



Tree certificate example



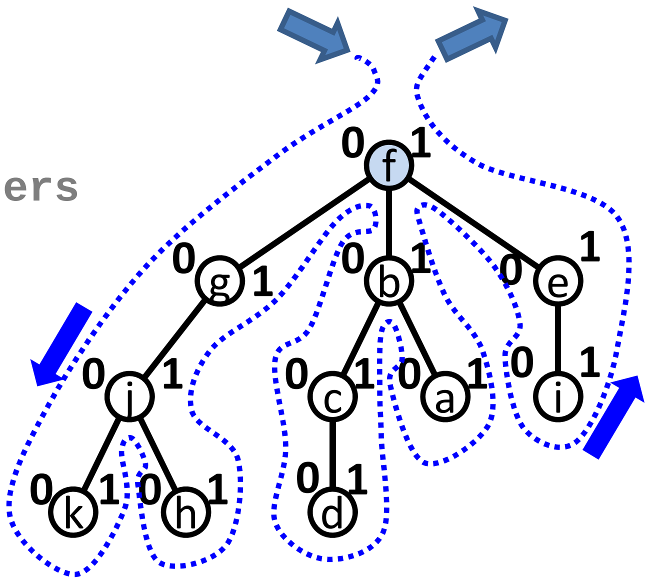
000010111000110110011 1



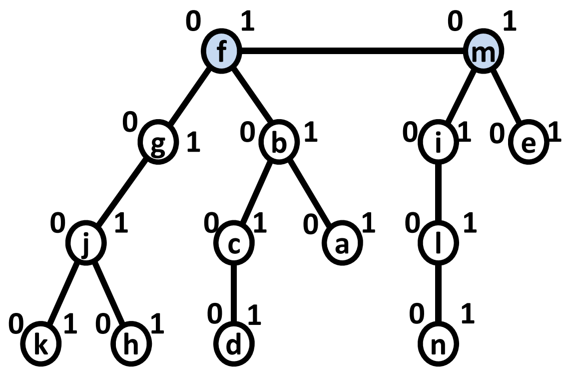
0 0 0 0 1 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 1 1

- 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.

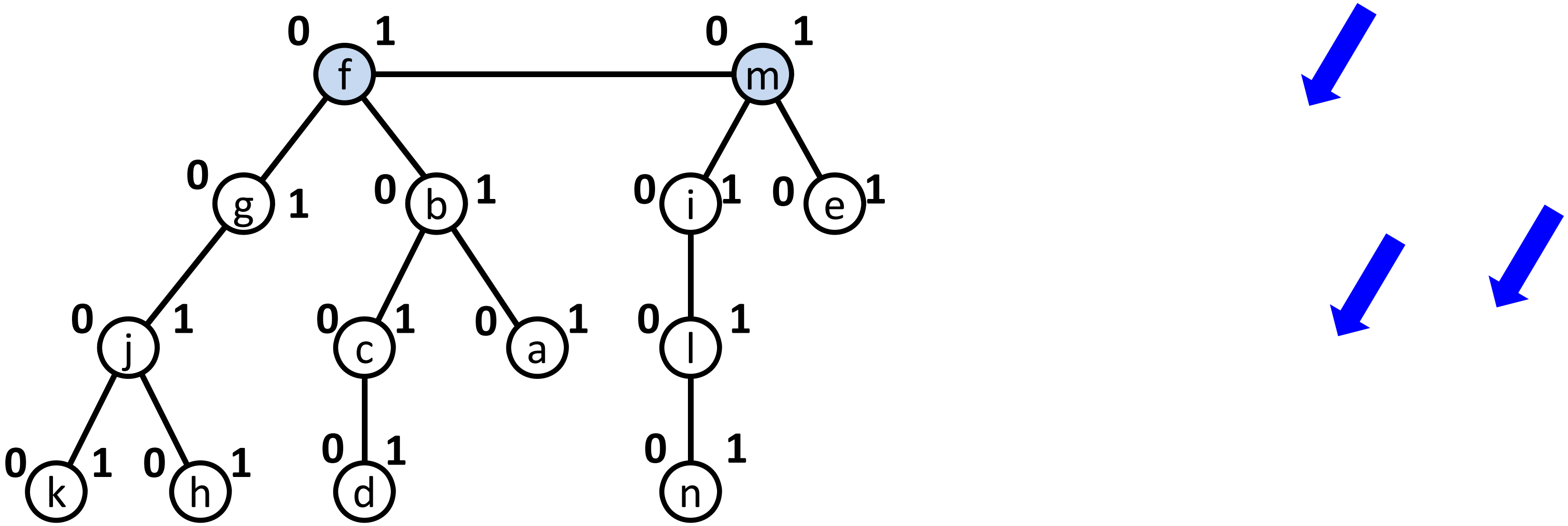
```
proc reconstructTree( certificate )
 nodesList = emptyList()
 edgesList = emptyList()
 centers = emptyList()  // one or two centers
 stack = emptyStack()

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

 if centers.size() == 2  // two centers
   edgesList.add( pair(centers[0],centers[1]) )
return  nodesList, edgesList
```



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



0000101110001101110000111011



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 the subtree which certificate is lexicographically the smallest.
- Output 0 when the node is open and output 1 one the node is 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.