Combinatorial algorithms

computing graph isomorphism,
computing tree isomorphism

Jiří Vyskočil, Radek Mařík

2013

Computing Graph Isomorphism

- **definition:**

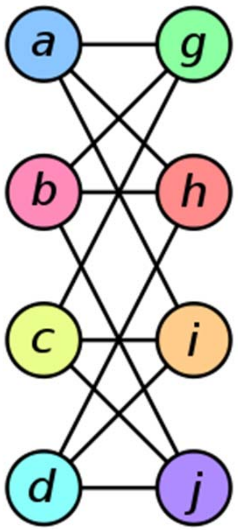
Two graphs G_1=(V_1,E_1) and G_2=(V_2,E_2) are *isomorphic* if there is a bijection $f: V_1 \rightarrow V_2$ such that

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

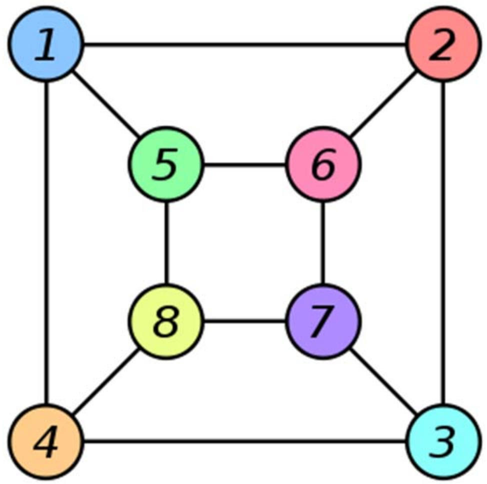
The mapping f is said to be an *isomorphism* between G_1 and G_2.

- **example:**

G_1 :



G_2 :



f :

$f(a) = 1$
$f(b) = 6$
$f(c) = 8$
$f(d) = 3$
$f(g) = 5$
$f(h) = 2$
$f(i) = 4$
$f(j) = 7$

Computing Graph Isomorphism

- **problem:**

The *graph isomorphism problem* is the computational problem of determining whether two finite graphs are isomorphic.

- The graph isomorphism problem is one of a very small number of problems belonging to NP neither known to be solvable in polynomial time nor NP-complete.
- However, there is a number of important special cases of the graph isomorphism problem that have efficient, polynomial-time solutions: trees, planar graphs, some bounded-parameter graphs, etc.

Computing Graph Isomorphism

- **definition of invariant:**

Let $\mathcal{F}$ be a family of graphs. An *invariant* on $\mathcal{F}$ is a function Φ with domain $\mathcal{F}$ such that

$$\forall\, G_1, G_2 \in \mathcal{F} \;:\; \Phi(G_1) = \Phi(G_2) \;\Leftarrow\; G_1 \text{ is isomorphic to } G_2$$

- **example:**
  - $|V|$ for graph $G=(V, E)$ is an invariant.
  - The following degree sequence $[\deg(v_1), \deg(v_2), \deg(v_3), \ldots, \deg(v_n)]$ is not an invariant.
  - However, if the degree sequence is sorted in non-decreasing order, then it is an invariant.

Computing Graph Isomorphism

```python

isomorphisms = [] # list of found isomorphisms
v = 0 # index of the vertex in g1 to be mapped
f = [-1] * len(G1.vertices) # isomorphism mapping, f[index of G1 vertex] = index of G2 vertex

collect_isomorphisms(G1, G2, v, f, isomorphisms)

def collect_isomorphisms(G1, G2, v, f, isomorphisms):

    N = len(G1.vertices)
    if v == N:
        return (f,) # return from the inner-most recursion

    for y in G2.vertices:
        if y in f[:v]: # vertex y has already been mapped. We can't use it twice.
            continue

        # check that the structure mapped so far is the same:
        OK = True
        for u in range(v):
            if (u in G1['neighbors'][v]) ^ (f[u] in G2['neighbors'][y]):
                OK = False
                break
        if OK: # structure is fine (all edges are there), we have a candidate:
            f[v] = y
            result = collect_isomorphisms(G1, G2, v+1, f, isomorphisms)
            if type( result ) is tuple: # we have a isomorphism
                isomorphisms.append(result[0])
    return None
```

Computing Graph Isomorphism

- **definition :**

Let $\mathcal{F}$ be a family of graphs on vertex set V and let D be a function with domain ($\mathcal{F} \times V$). Then the *partition* B_G *of* V *induced* by D is

$$B_G = [\ B_G[0], B_G[1], \ldots , B_G[n-1]\ ]$$

where

$$B_G[i] = \{\ v \in V \ : \ D\,(G,v) = i\ \}$$

If the function

$$\Phi_D(G) = [\ |B_G[0]|, |B_G[1]|, \ldots , |B_G[n-1]|\ ]$$

is an invariant, then we say that D is an *invariant inducing function*.

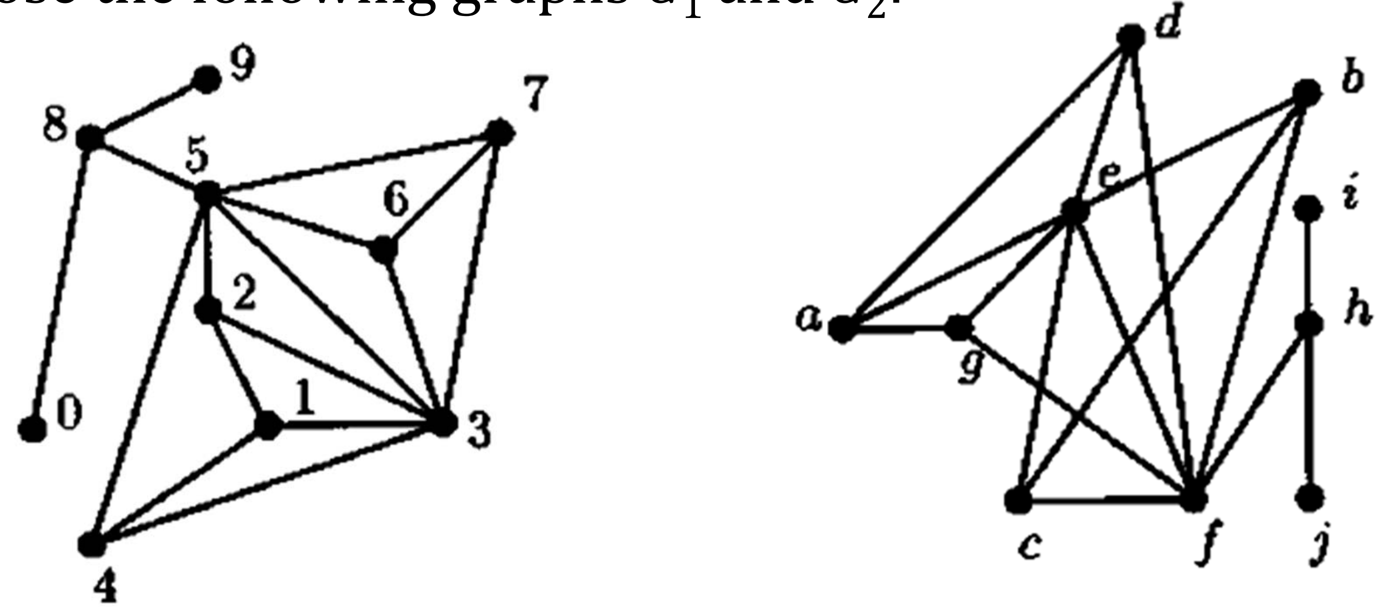
Computing Graph Isomorphism - Example

Let

- $D_1(G,x)=\deg_G(x)$
- $D_2(G,x)=[d_j(x) : j = 1, 2, \ldots, \max\{\deg_G(x) : x \in V(G)\}]$

  where $d_j(x)=|\{y : y \text{ is adjacent to } x \text{ and } \deg_G(y) = j\}|$

Suppose the following graphs G_1 and G_2:



Computing Graph Isomorphism - Example

$$X_0(\mathcal{G}_1) = \{0,1,2,3,4,5,6,7,8,9\}.$$

$$X_0(\mathcal{G}_2) = \{a,b,c,d,e,f,g,h,i,j\}.$$

x	0	1	2	3	4	5	6	7	8	9
$D_1(\mathcal{G}_1,x)$	1	3	3	6	3	6	3	3	3	1

$$\Downarrow$$

$$X_1(\mathcal{G}_1) = \{0,9\},\{1,2,4,6,7,8\},\{3,5\}$$

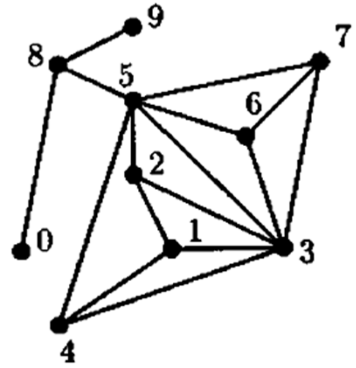
(1) (3) (6)

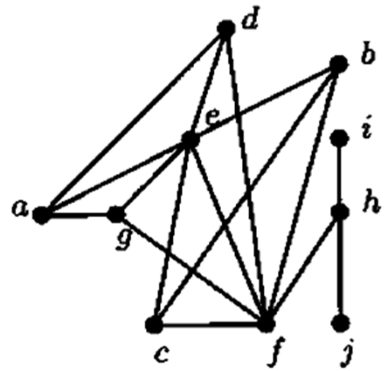
$\bar{x}$	a	b	c	d	e	f	g	h	i	j
$D_1(\mathcal{G}_2,\bar{x})$	3	3	3	3	6	6	3	3	1	1

$$\Downarrow$$

$$X_1(\mathcal{G}_2) = \{i,j\},\{a,b,c,d,g,h\},\{e,f\}.$$

(1) (3) (6)



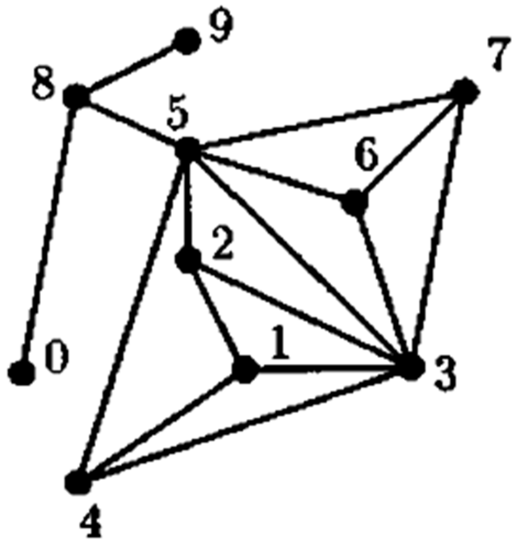


Computing Graph Isomorphism - Example

$$
\begin{aligned}
D_2(\mathcal{G}_1,0) &= (0,0,1,0,0,0,0,0,0)\\
D_2(\mathcal{G}_1,1) &= (0,0,2,0,0,1,0,0,0)\\
D_2(\mathcal{G}_1,2) &= (0,0,1,0,0,2,0,0,0)\\
D_2(\mathcal{G}_1,3) &= (0,0,5,0,0,1,0,0,0)\\
D_2(\mathcal{G}_1,4) &= (0,0,1,0,0,2,0,0,0)\\
D_2(\mathcal{G}_1,5) &= (0,0,5,0,0,1,0,0,0)\\
D_2(\mathcal{G}_1,6) &= (0,0,1,0,0,2,0,0,0)\\
D_2(\mathcal{G}_1,7) &= (0,0,1,0,0,2,0,0,0)\\
D_2(\mathcal{G}_1,8) &= (2,0,0,0,0,1,0,0,0)\\
D_2(\mathcal{G}_1,9) &= (0,0,1,0,0,0,0,0,0)
\end{aligned}
$$

$$\Downarrow$$

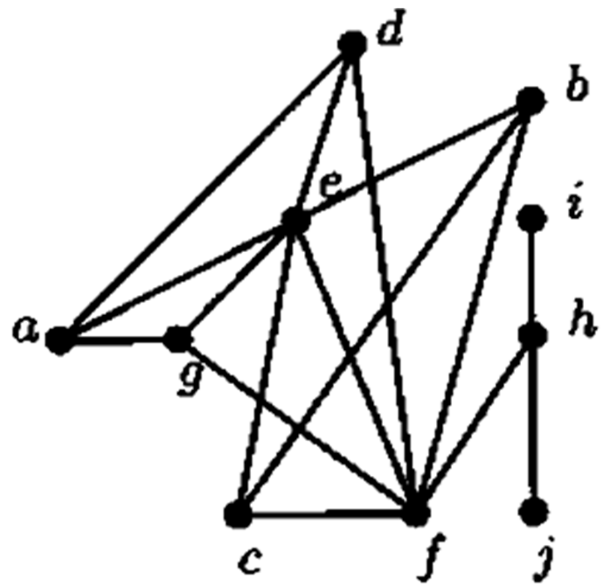
$$X_2(\mathcal{G}_1) = \{0,9\}, \{8\}, \{2,4,6,7\}, \{1\}, \{3,5\}.$$



$$
\begin{aligned}
D_2(\mathcal{G}_2,a) &= (0,0,2,0,0,1,0,0,0)\\
D_2(\mathcal{G}_2,b) &= (0,0,1,0,0,2,0,0,0)\\
D_2(\mathcal{G}_2,c) &= (0,0,1,0,0,2,0,0,0)\\
D_2(\mathcal{G}_2,d) &= (0,0,1,0,0,2,0,0,0)\\
D_2(\mathcal{G}_2,e) &= (0,0,5,0,0,1,0,0,0)\\
D_2(\mathcal{G}_2,f) &= (0,0,5,0,0,1,0,0,0)\\
D_2(\mathcal{G}_2,g) &= (0,0,1,0,0,2,0,0,0)\\
D_2(\mathcal{G}_2,h) &= (2,0,0,0,0,1,0,0,0)\\
D_2(\mathcal{G}_2,i) &= (0,0,1,0,0,0,0,0,0)\\
D_2(\mathcal{G}_2,j) &= (0,0,1,0,0,0,0,0,0)
\end{aligned}
$$

$$\Downarrow$$

$$X_2(\mathcal{G}_2) = \{i,j\}, \{h\}, \{b,c,d,g\}, \{a\}, \{e,f\}.$$

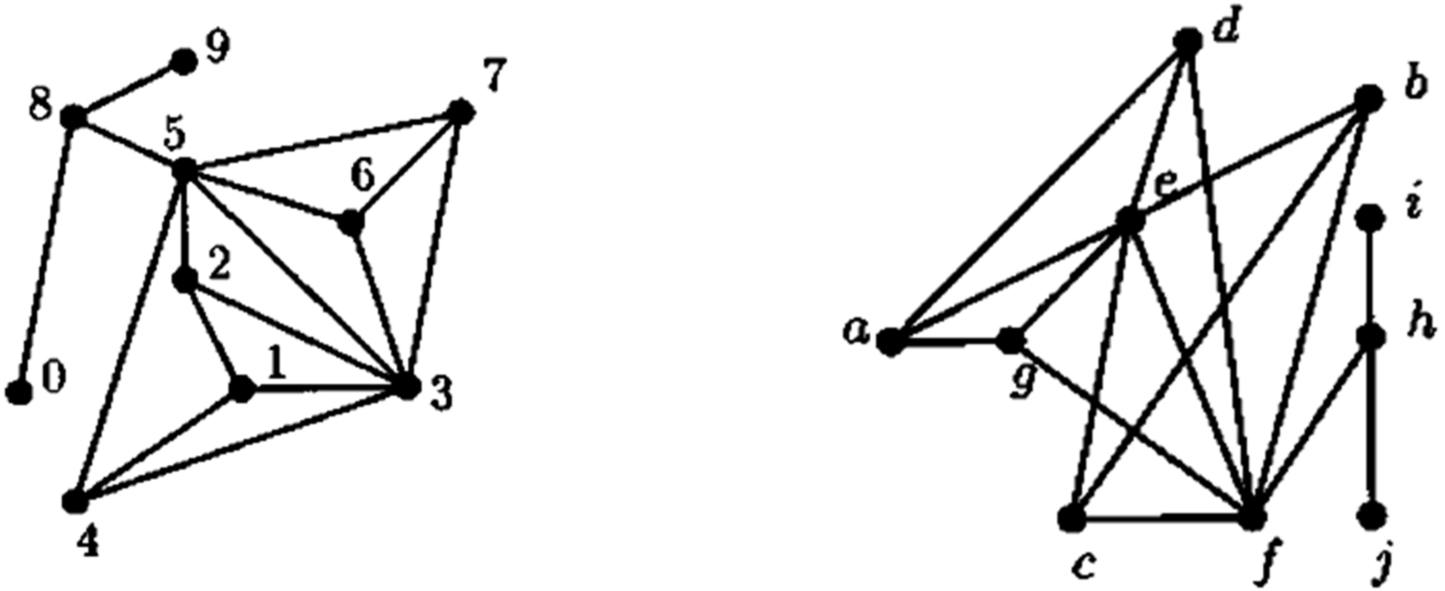


Computing Graph Isomorphism - Example

This restricts a possible isomorphism to bijections between the following sets:

$$\begin{aligned}
\{0,9\} &\longleftrightarrow \{i,j\} \\
\{8\} &\longleftrightarrow \{h\} \\
\{2,4,6,7\} &\longleftrightarrow \{b,c,d,g\} \\
\{1\} &\longleftrightarrow \{a\} \\
\{3,5\} &\longleftrightarrow \{e,f\}
\end{aligned}$$

There are $96 = (2!)(1!)(4!)(1!)(2!)$ bijections giving the possible isomorphisms. Examination of each of these possible isomorphisms shows that only the following eight bijections are isomorphisms.



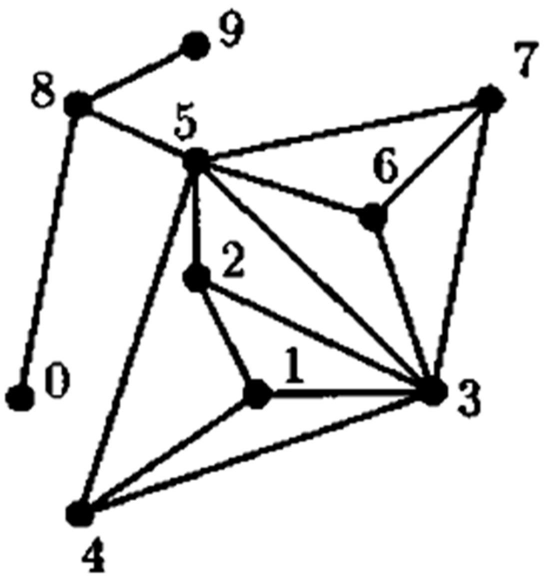
Computing Graph Isomorphism - Example

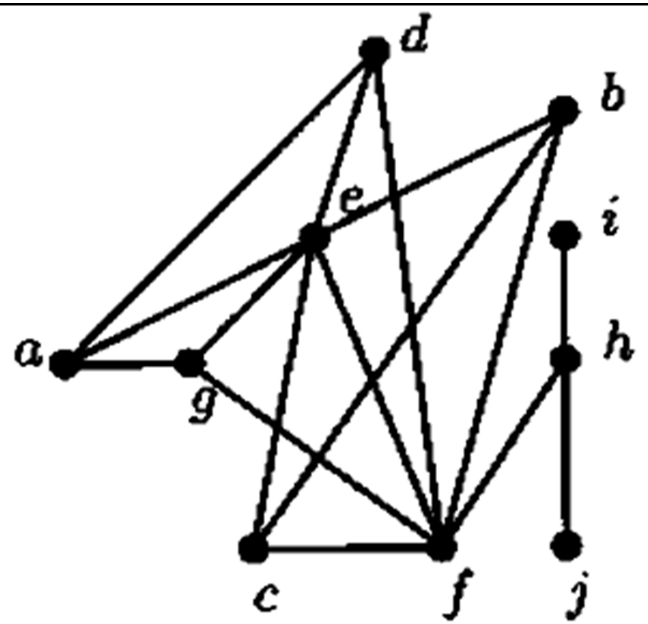
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ i & a & d & e & g & f & b & c & h & j \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ j & a & d & e & g & f & b & c & h & i \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ i & a & d & e & g & f & c & b & h & j \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ j & a & d & e & g & f & c & b & h & i \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ i & a & g & e & d & f & b & c & h & j \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ j & a & g & e & d & f & b & c & h & i \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ i & a & g & e & d & f & c & b & h & j \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ j & a & g & e & d & f & c & b & h & i \end{pmatrix}$$





Computing Graph Isomorphism

Function FINDISOMORPHISM (set of invariant inducing functions I; graph G_1, G_2) : set of isomorphisms

try {

    (*partitions*, X, Y) = GETPARTITIONS (I, G_1, G_2) ;

}

catch ("G_1 and G_2 are not isomorphic! ") { **return** $\emptyset$; }

for $i = 0$ **to** $partitions - 1$ **do** {

    for each $x \in X[i]$ **do** {

        $W[x] = i$;

    }

}

return COLLECTISOMORPHISMS(G_1, G_2, 0, Y, W, f)

Computing Graph Isomorphism

Function GETPARTITIONS $\begin{pmatrix} \text{set of invariant inducing functions } I; \\ \text{graph } G_1; \\ \text{graph } G_2 \end{pmatrix} : \begin{pmatrix} \text{number of partitions } N, \\ \text{paritítions of } G_1\ X, \\ \text{paritítions of } G_2\ Y \end{pmatrix}$

$N = 1$; $X[0] =$ vertices of G_1; $Y[0] =$ vertices of G_2;

for each $D \in I$ **do** {

  $P = N$;

  for $i = 0$ **to** $P - 1$ **do** {

    Partition $X[i]$ into sets $X_1[i], X_2[i], X_3[i], \dots, X_m[i]$ where $x,y \in X_j[i] \Leftrightarrow D(G_1,x) = D(G_1,y)$;

    Partition $Y[i]$ into sets $Y_1[i], Y_2[i], Y_3[i], \dots, Y_n[i]$ where $x,y \in Y_j[i] \Leftrightarrow D(G_2,x) = D(G_2,y)$;

    if $n \neq m$ **then throw exception** "G_1 and G_2 are not isomorphic!";

    Order $Y[i]$ into sets $Y_1[i], Y_2[i], Y_3[i], \dots, Y_n[i]$ so that

      $\forall x \in X[i], \forall y \in Y[i] : D(G_1,x) = D(G_2,y) \Leftrightarrow x \in X_j[i]$ and $y \in Y_j[i]$;

    if ordering is not possible **then throw exception** "G_1 and G_2 are not isomorphic!";

    $N = N + m - 1$;

  }

  Reorder the partitions so that: $|X[i]| = |Y[i]| \leq |X[i+1]| = |Y[i+1]|$ for $0 \leq i < N - 1$;

}

return (N, X, Y)

Computing Graph Isomorphism

Function CollectIsomorphisms $\left(\begin{array}{c} \text{graph } G_1, G_2\,; \\ \text{starting vertex of } G_1 \;\; v\,; \\ \text{parititions of } G_2 \;\; Y\,; \end{array} \begin{array}{c} \text{partition mapping } W \text{ as} \\ \textbf{array } [\text{vertices of } G_1] \textbf{ of} \\ \text{indices of partitions of } G_1 \end{array} ; \begin{array}{c} \text{current isomorphism } f \text{ as} \\ \textbf{array } [\text{vertices of } G_1] \textbf{ of} \\ \text{vertices of } G_2 \end{array}\right)$ **:** set of isomorphisms

if v = number of vertices of G_1 **then return** $\{f\}$;

$R = \emptyset$;

$p = W[v]$;

for each $y \in Y[p]$ **do {**

    OK = **true** ;

    for $u = 0$ **to** $v - 1$ **do {**

        if $\{u, v\} \in$ edges of G_1 **xor** $\{f[u], y\} \in$ edges of G_2 **then { ** OK = **false** ; **break** ; **}**

    }

    if OK **then {**

        $f[v] = y$;

        $R = R \cup$ CollectIsomorphisms$(G_1, G_2, v{+}1, Y, W, f)$;

    }

}

return R

Certificate

- A certificate $Cert$ for family $\mathcal{F}$ of graphs is a function such that

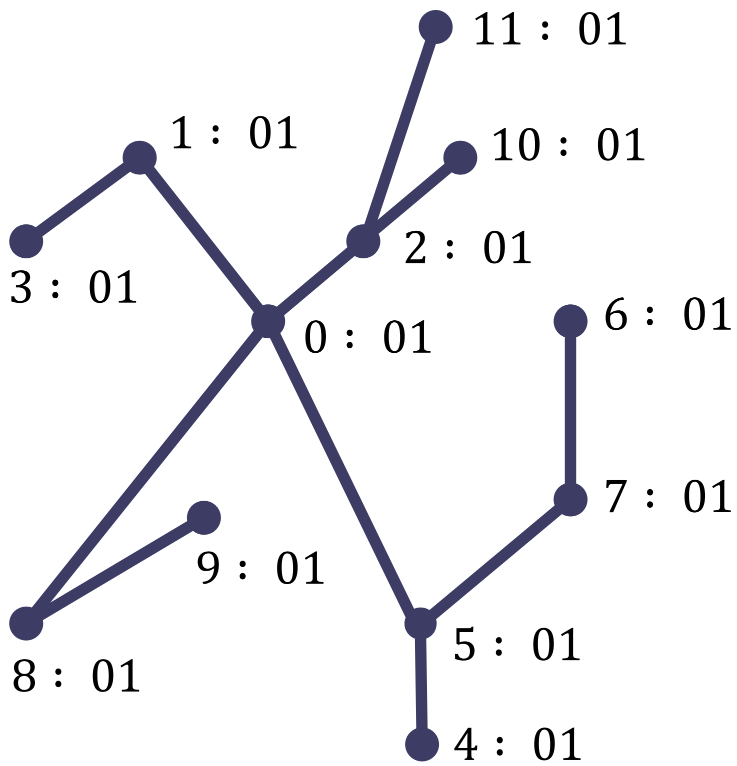
$$\forall\, G_1, G_2 \in \mathcal{F} \;:\quad Cert(G_1) = Cert(G_2) \;\Leftrightarrow\; G_1 \text{ is isomorphic to } G_2$$

- Currently, the fastest general graph isomorphism algorithms use methods based on computing of certificates.
- Computing of certificates works not only for general graphs but it can be also applied on some classes of graphs like trees.

Computing Tree Certificate

1) Label all the vertices of G with the string 01.
2) While there are more than two vertices of G do:
   For each non-leaf x of G:
   a) Let Y be the multi-set of labels of the leaves adjacent to x and the label of x, with the initial 0 and trailing 1 deleted from x;
   b) Replace the label of x with concatenation of the labels in Y sorted in increasing lexicographic order, with 0 prepended and a 1 appended;
   c) Remove all leaves adjacent to x.
3) If there is only one vertex left, report the label of x as certificate.
4) If there are two vertices x and y left, then report the labels of x and y, concatenated in increasing lexicographic order, as the certificate.

Computing Tree Certificate - Example

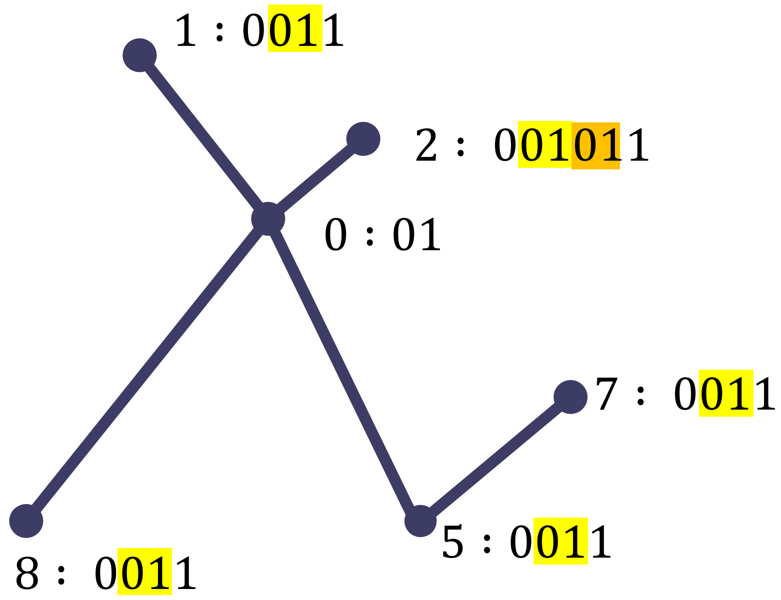


number of vertices: 12

non−leaves vertices:

$0: Y = \langle \quad \rangle$
$1: Y = \langle 01 \rangle$
$2: Y = \langle 01, 01 \rangle$
$5: Y = \langle 01 \rangle$
$7: Y = \langle 01 \rangle$
$8: Y = \langle 01 \rangle$

Computing Tree Certificate - Example



number of vertices: 6

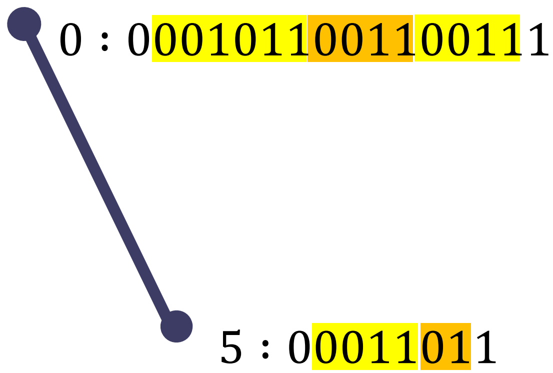
non−leaves vertices:

$$0 : \; Y = \begin{pmatrix} 001011, \\ 0011, \\ 0011 \end{pmatrix}$$

$$5 : \; Y = \begin{pmatrix} 0011, \\ 01 \end{pmatrix}$$

Computing Tree Certificate - Example

number of vertices: 2



Certificate=000101100110011100011011

Computing Tree Certificate

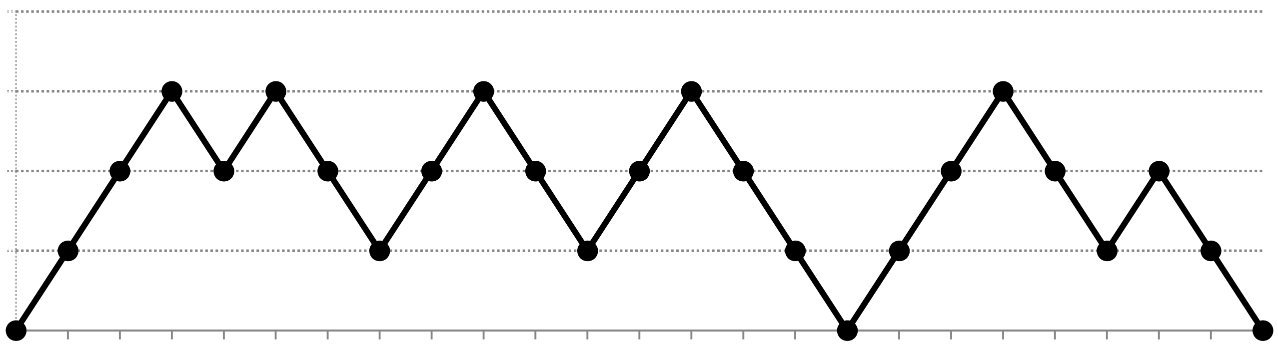
- **properties of certificate:**
  - the length is $2 \cdot |V|$
  - the number of 1s and 0s is the same
  - furthermore, the number of 1s and 0s is the same for every partial subsequence that arise from any label of vertex (during the whole run of the algorithm)

Reconstruction of Tree from Certificate - Example

$$f(0) = 0$$

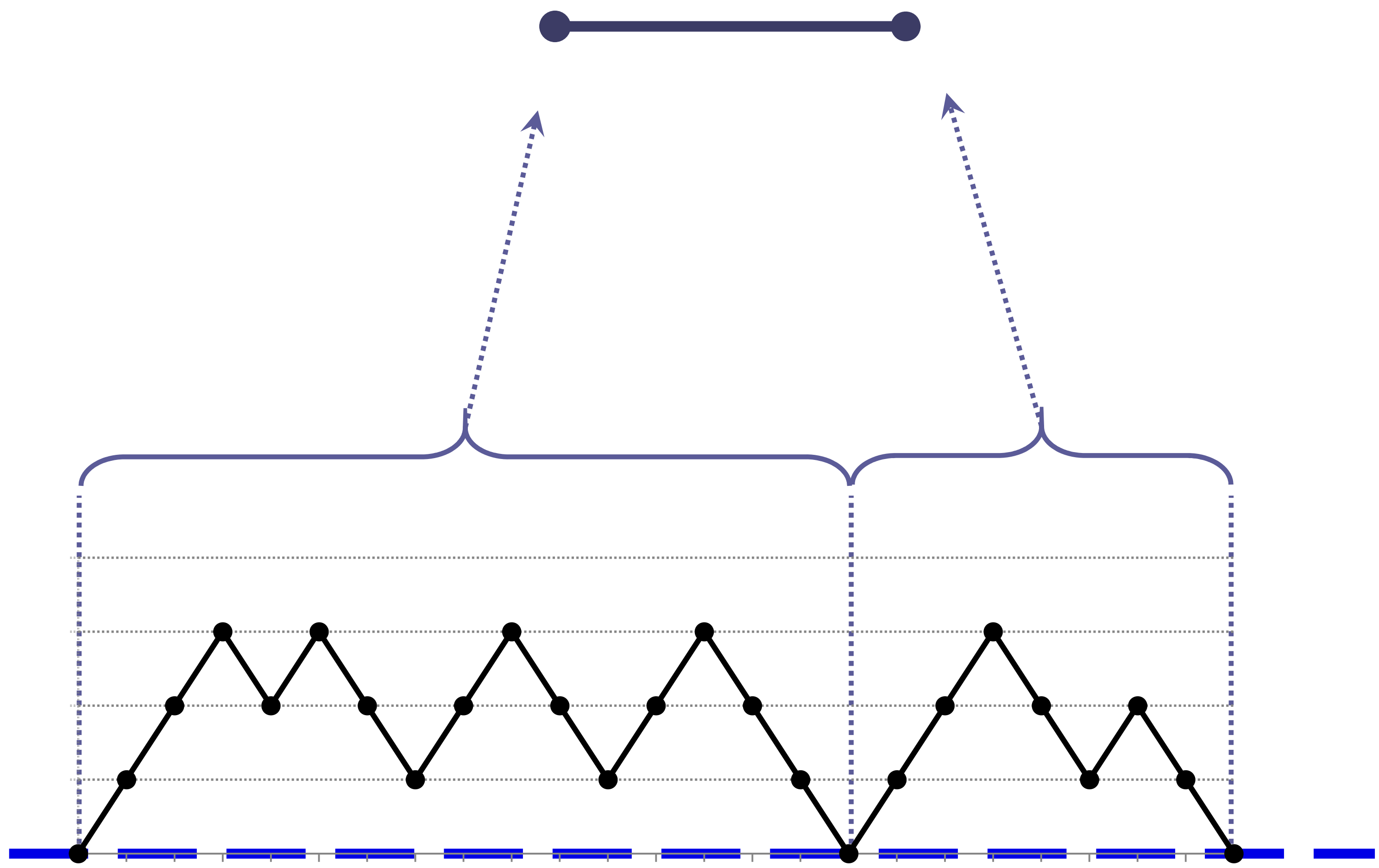
$$f(x+1) = \begin{cases} f(x) + 1, & Cert(G)[x] = 0 \\ f(x) - 1, & Cert(G)[x] = 1 \end{cases}$$

f:



$$Cert(G) = 000101100110011100011011$$

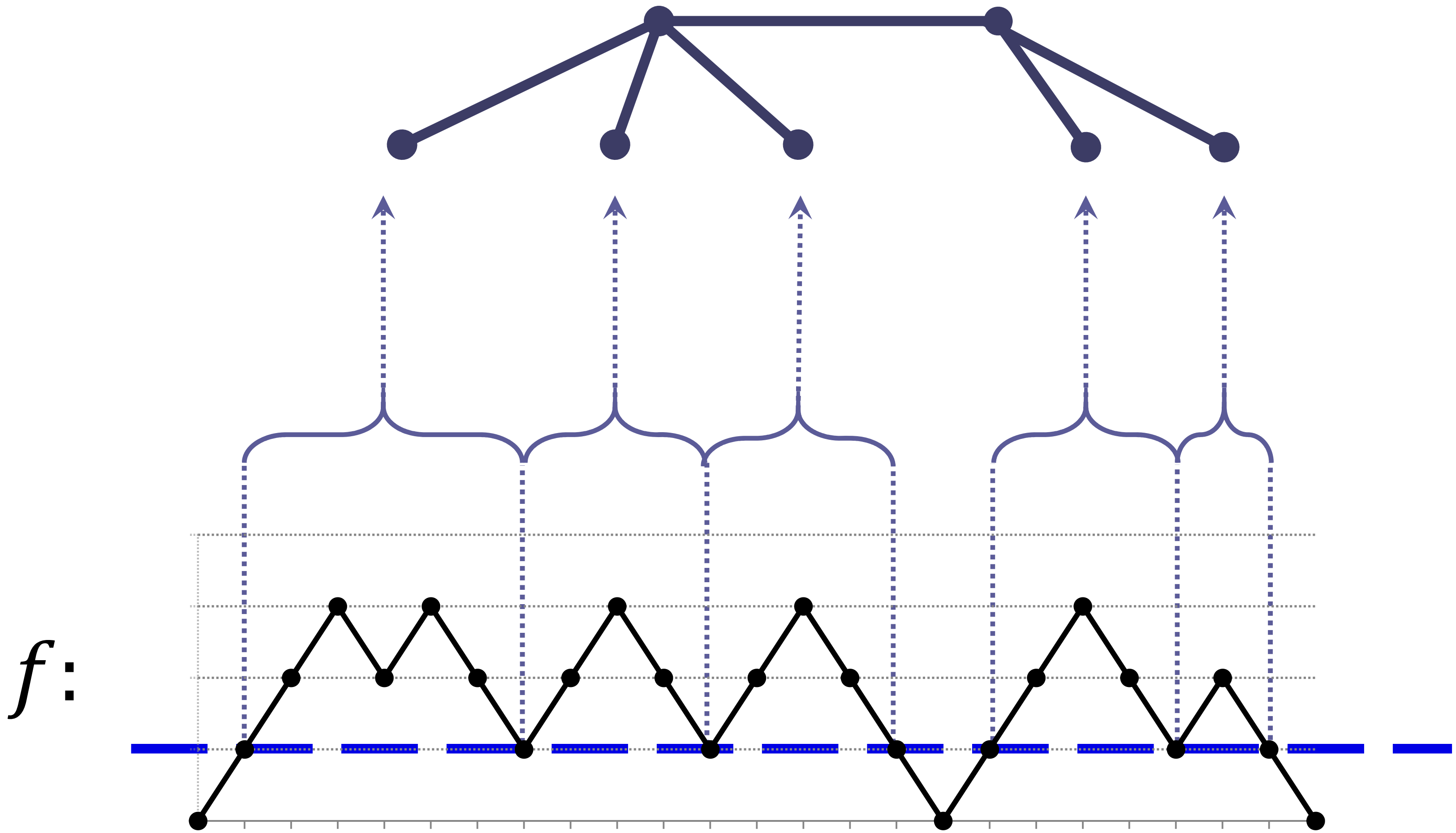
Reconstruction of Tree from Certificate - Example



f:

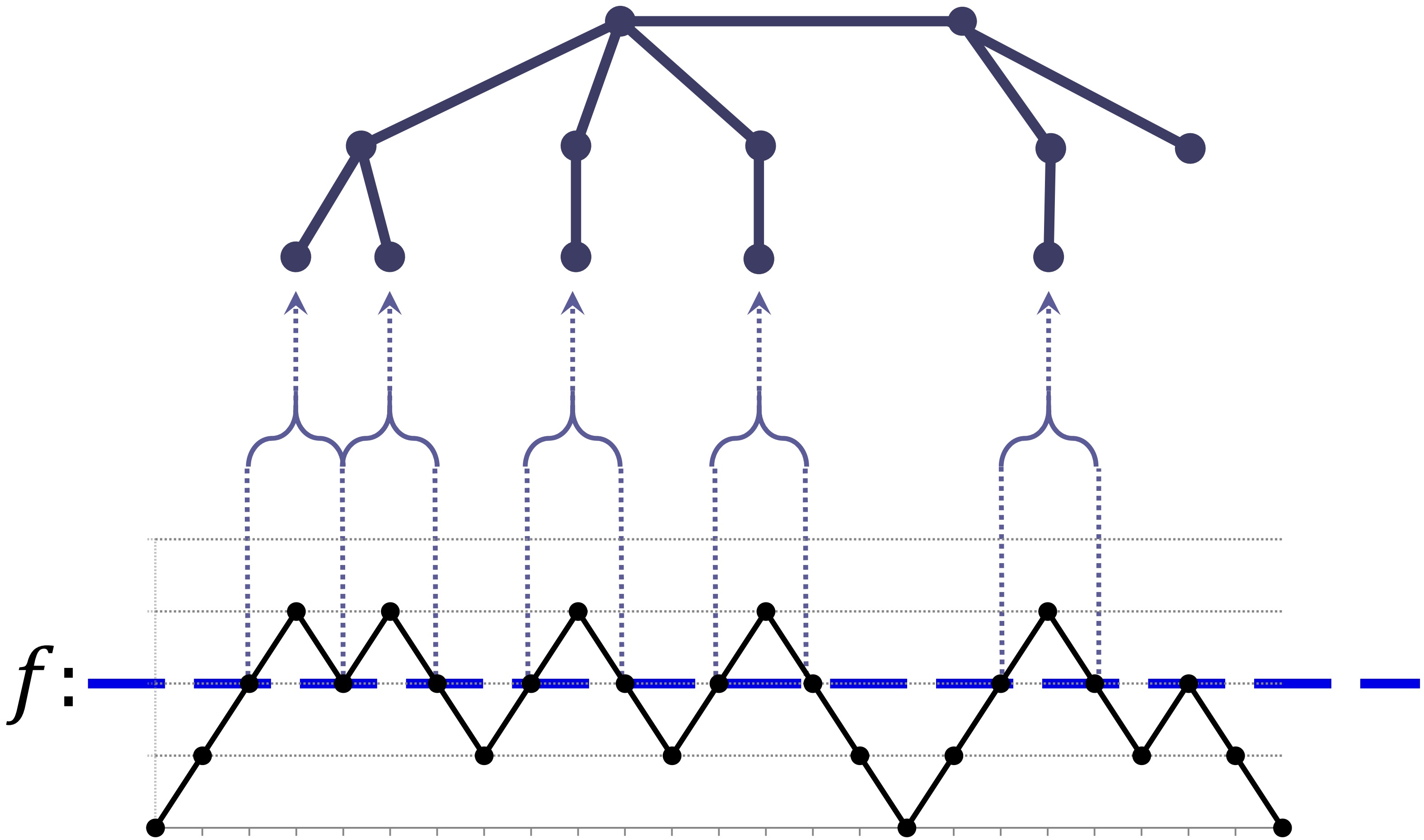
$Cert(G) = 0001011001100111000011011$

Reconstruction of Tree from Certificate - Example



$$Cert(G) = 000101100110011100011011$$

Reconstruction of Tree from Certificate - Example



$$Cert(G) = 000101100110011100011011$$

Reconstruction of Tree from Certificate

Function FINDSUBMOUNTAINS(integer l, certificate as string C) : number of submountines in C
$k = 0$; $M[0] =$ the empty string; $f = 0$;
for $x = l - 1$ **to** $|C| - l$ **do** {
  if $C[x] = 0$ **then** { $f = f + 1$; } **else** { $f = f - 1$; }
  $M[k] = M[k] \cdot C[x]$;
  if $f = 0$ **then** { $k = k + 1$; $M[k] =$ the empty string; $f = 0$; }
}
return k;

Function CERTIFICATETOTREE (certificate as string C) : tree as $G = (V, E)$
$n = \frac{|C|}{2}$; $v = 0$; $(V, E) =$ empty graph of order n; $V = \{0, \dots, n-1\}$;
$k =$ FINDSUBMOUNTAINS$(1, C)$;
if $k = 1$ **then** { $Label[v] = M[0]$; $v = v + 1$; }
  else { $Label[v] = M[0]$; $v = v + 1$; $Label[v] = M[1]$; $v = v + 1$; $E = E \cup \{\{0,1\}\}$; }
for $i = 0$ **to** $n - 1$ **do** {
if $|Label[i]| > 2$ **then** {
  $k =$ FINDSUBMOUNTAINS$(2, Label[i])$; $Label[i] =$ "01";
  for $j = 0$ **to** $k - 1$ **do** { $Label[v] = M[j]$; $E = E \cup \{\{i, v\}\}$; $v = v + 1$; }
}
return $G = (V, E)$;

$O(|C|^2)$

Reconstruction of Tree from Certificate

Function $\textsc{FastCertificateToTree}$ (certificate as string C) : tree as $G = (V, E)$

(V, E) = empty digraph of order $\frac{|C|}{2}$; $V = \left\{0, \dots, \frac{|C|}{2}\right\}$;

$n = 0$;

$p = n$;

for $x = 1$ **to** $|C| - 2$ **do** {

    if $C[x] = 0$ **then** {

        $n = n + 1$;

        $E = E \cup \{(p, n)\}$;

        $p = n$;

    } **else** {

        $p = parent^{\dagger}(p)$;

    }

}

return $G = (V, remove\_orientation(E))$;

† $parent(x)$ returns the parent of a node x. It returns x in the case where x has no parent.

$O(|C|)$

References

- D.L. Kreher and D.R. Stinson , *Combinatorial Algorithms: Generation, Enumeration and Search* , CRC press LTC , Boca Raton, Florida, 1998.