



# Combinatorial algorithms

computing graph isomorphism,  
computing tree isomorphism

Jiří Vyskočil, Radek Mařík

2012

# Computing Graph Isomorphism

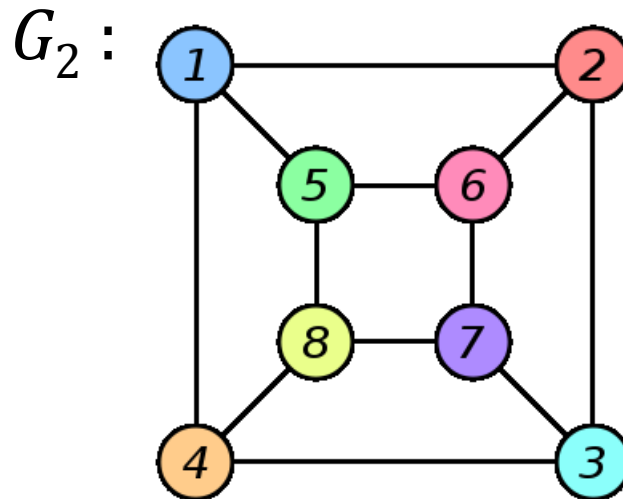
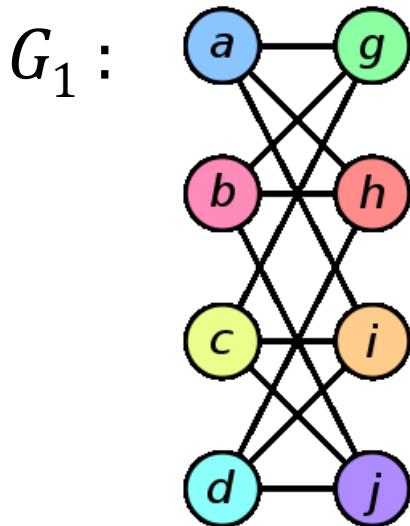
## ■ definition:

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

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

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

## ■ example:



$f$  :

$$\begin{aligned} f(a) &= 1 \\ f(b) &= 6 \\ f(c) &= 8 \\ f(d) &= 3 \\ f(g) &= 5 \\ f(h) &= 2 \\ f(i) &= 4 \\ f(j) &= 7 \end{aligned}$$

# Computing Graph Isomorphism

## ■ definition of invariant:

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

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

## ■ example:

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

# Computing Graph Isomorphism

## ■ 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 induced* by  $D$  is

$$B = [ |B[0]|, |B[1]|, \dots, |B[n-1]| ]$$

where

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

If the function

$$\Phi_D(G) = [ |B[0]|, |B[1]|, \dots, |B[n-1]| ]$$

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

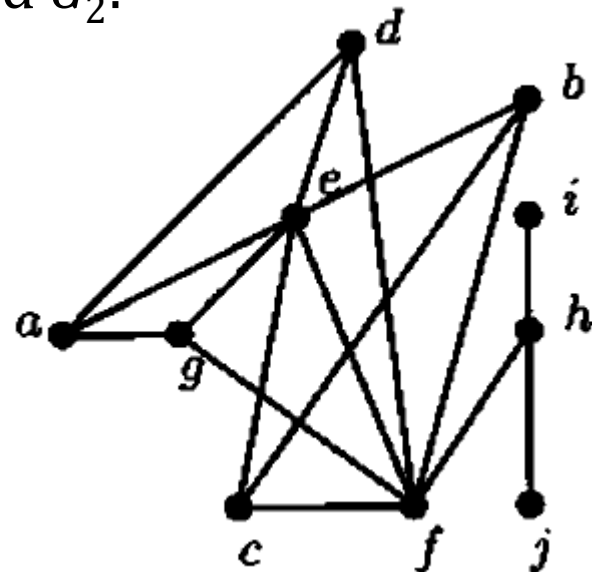
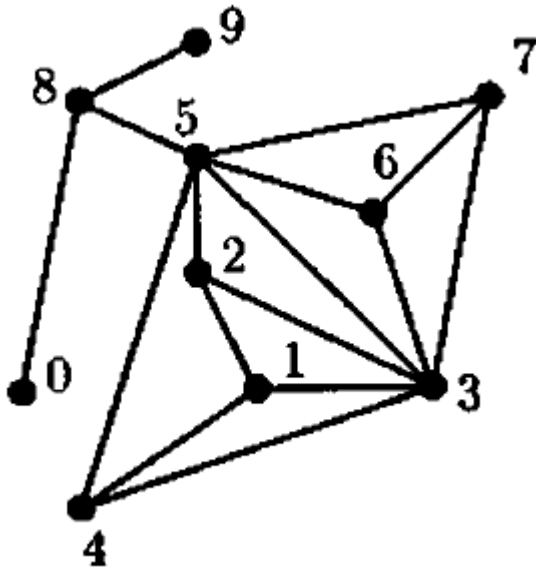
# Computing Graph Isomorphism - Example

Let

- $D_1(G,x) = \deg_G(x)$
- $D_2(G,x) = [d_j(x) : j = 1, 2, \dots, d_{n-1}]$

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

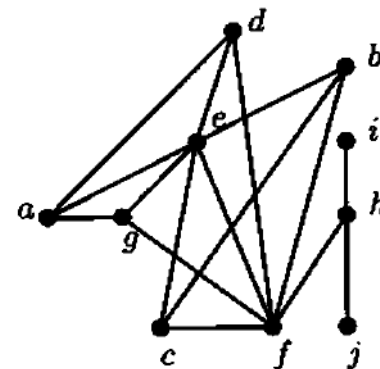
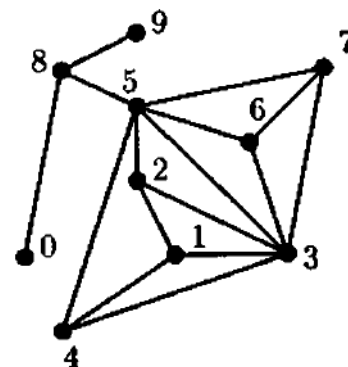


$$X_1(\mathcal{G}_1) = \{0, 9\}, \{1, 2, 4, 6, 7, 8\}, \{3, 5\}$$

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



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



# Computing Graph Isomorphism - Example

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

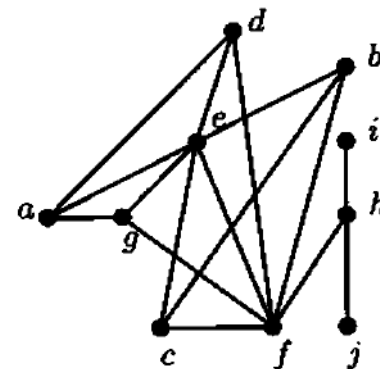
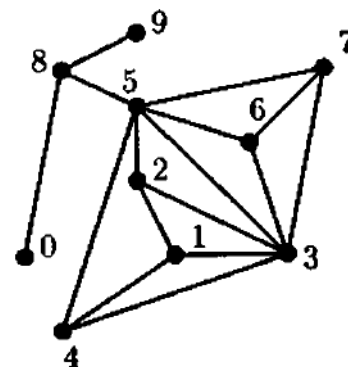
↓

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

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

↓

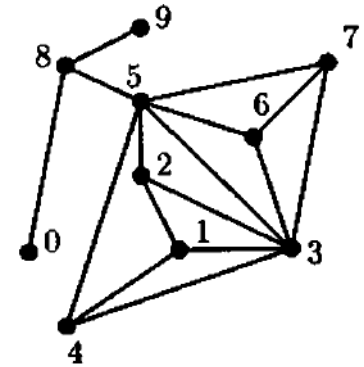
$$X_2(\mathcal{G}_2) = \{i, j\}, \{h\}, \{b, c, d, g\}, \{a\}, \{e, f\}.$$



# Computing Graph Isomorphism - Example

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

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



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.

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

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

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

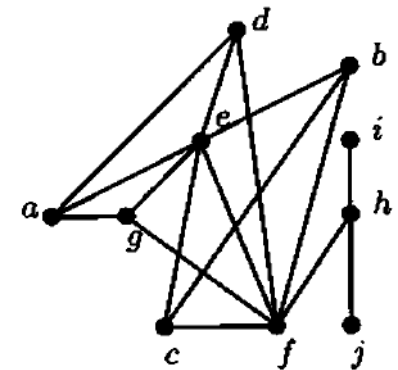
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ j & a & d & e & g & f & c & b & h & i \end{pmatrix}$$

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

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

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

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





# Computing Graph Isomorphism

- 1) **Function** FINDISOMORPHISM (set of invariant inducing functions  $I$ ; graph  $G_1, G_2$ ) : set of isomorphisms
- 2) **try** {
- 3)      $(N, X, Y) = \text{GETPARTITIONS}(I, G_1, G_2)$ ;
- 4) }
- 5) **catch** (“ $G_1$  and  $G_2$  are not isomorphic!”) { **return**  $\emptyset$ ; }
- 6) **for**  $i = 0$  **to**  $N - 1$  **do** {
- 7)     **for each**  $x \in X[i]$  **do** {
- 8)          $W[x] = i$ ;
- 9)     }
- 10) }
- 11) **return** COLLECTISOMORPHISMS( $G_1, G_2, 0, Y, W, f$ )

# Computing Graph Isomorphism

- 1) **Function** GETPARTITIONS  $\left( \begin{array}{l} \text{set of invariant inducing functions } I; \\ \text{graph } G_1; \\ \text{graph } G_2 \end{array} \right) : \left( \begin{array}{l} \text{number of partitions,} \\ \text{partitions of } G_1, \\ \text{partitions of } G_2 \end{array} \right)$
- 2)  $X[0] = \text{vertices of } G_1; Y[0] = \text{vertices of } G_2; N = 1;$
- 3) **for each**  $D \in I$  **do** {
- 4)      $P = N;$
- 5)     **for**  $i = 0$  **to**  $P - 1$  **do** {
- 6)         Partition  $X[i]$  into sets  $X_1[i], X_2[i], X_3[i], \dots, X_m[i]$  where  $x, y \in X_j[i] \Leftrightarrow D(G_1, x) = D(G_1, y);$
- 7)         Partition  $Y[i]$  into sets  $Y_1[i], Y_2[i], Y_3[i], \dots, Y_n[i]$  where  $x, y \in Y_j[i] \Leftrightarrow D(G_2, x) = D(G_2, y);$
- 8)         **if**  $n \neq m$  **then throw exception** “ $G_1$  and  $G_2$  are not isomorphic!” ;
- 9)         Order  $Y[i]$  into sets  $Y_1[i], Y_2[i], Y_3[i], \dots, Y_n[i]$  so that
- 10)              $\forall x \in X[i], \forall y \in Y[i] : D(G_1, x) = D(G_2, y) \Leftrightarrow x \in X_j[i] \text{ and } y \in Y_j[i];$
- 11)         **if** ordering is not possible **then throw exception** “ $G_1$  and  $G_2$  are not isomorphic!” ;
- 12)          $N = N + m - 1;$
- 13)     }
- 14)     Reorder the partitions so that:  $|X[i]| = |Y[i]| \leq |X[i+1]| = |Y[i+1]|$  for  $0 \leq i < N - 1;$
- 15) }
- 16) **return**  $(N, X, Y)$

# Computing Graph Isomorphism

1) **Function**  $\text{COLLECTISOMORPHISMS}$  ( graph  $G_1, G_2$ ; partition mapping  $W$  as starting vertex of  $G_1$   $v$ ; array [vertices of  $G_1$ ] of partitions of  $G_2$   $Y$ ; indices of partitions of  $G_1$  ; array [vertices of  $G_1$ ] of vertices of  $G_2$   $f$  as ) : set of isomorphisms

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

3)  $R = \emptyset$ ;

4)  $p = W[v]$ ;

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

6)      $OK = \text{true}$ ;

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

8)         **if**  $\left( \begin{array}{l} (\{u, v\} \in \text{edges of } G_1 \text{ and } \{f[u], y\} \notin \text{edges of } G_2) \\ \text{or} \\ (\{u, v\} \notin \text{edges of } G_1 \text{ and } \{f[u], y\} \in \text{edges of } G_2) \end{array} \right)$  **then**  $OK = \text{false}$ ;

9)     }

10)    **if**  $OK$  **then** {

11)        $f[v] = y$ ;

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

13)    }

14) }

15) **return**  $R$

# Certificate

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

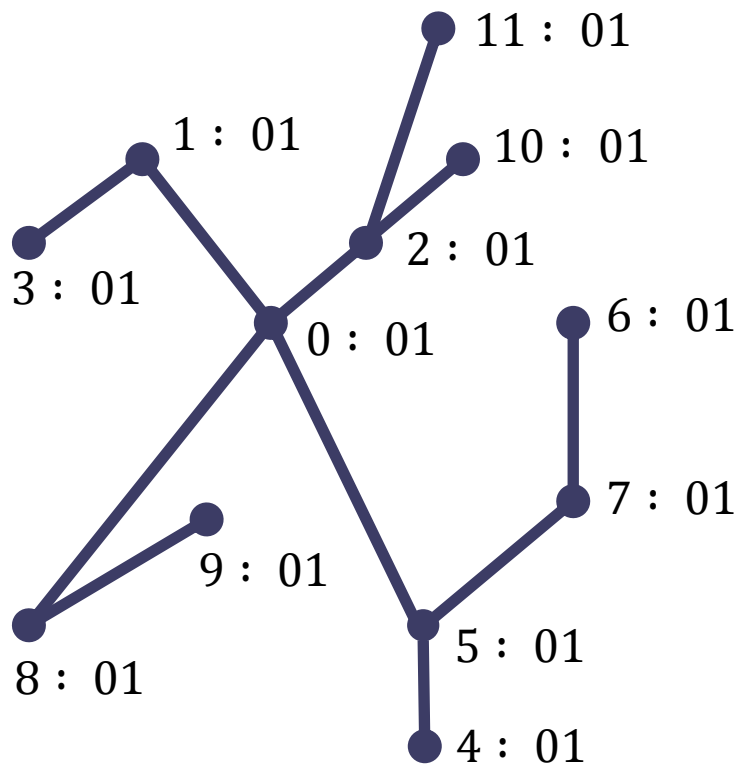
$$\forall G_1, G_2 \in \mathcal{F} : Cert(G_1) = Cert(G_2) \Leftrightarrow G_1 \text{ is isomorphic to } G_2$$

- 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 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 :  $Y = \langle \ \rangle$

1 :  $Y = \langle 01 \rangle$

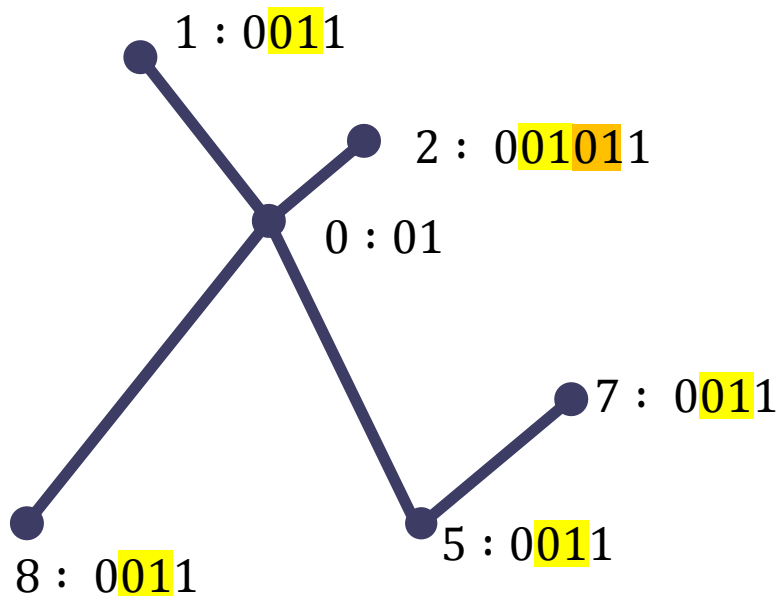
2 :  $Y = \langle 01, 01 \rangle$

5 :  $Y = \langle 01 \rangle$

7 :  $Y = \langle 01 \rangle$

8 :  $Y = \langle 01 \rangle$

# Computing Tree Certificate - Example



number of vertices: 6

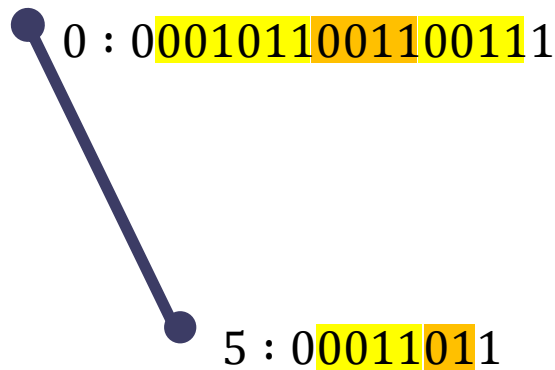
non-leaves vertices:

$$0 : Y = \left\langle \begin{array}{l} 001011, \\ 0011, \\ 0011 \end{array} \right\rangle$$

$$5 : Y = \left\langle \begin{array}{l} 0011, \\ 01 \end{array} \right\rangle$$

# Computing Tree Certificate - Example

number of vertices: 2



Certificate=000101100110011100011011



# Computing Tree Certificate

## ■ properties of certificate:

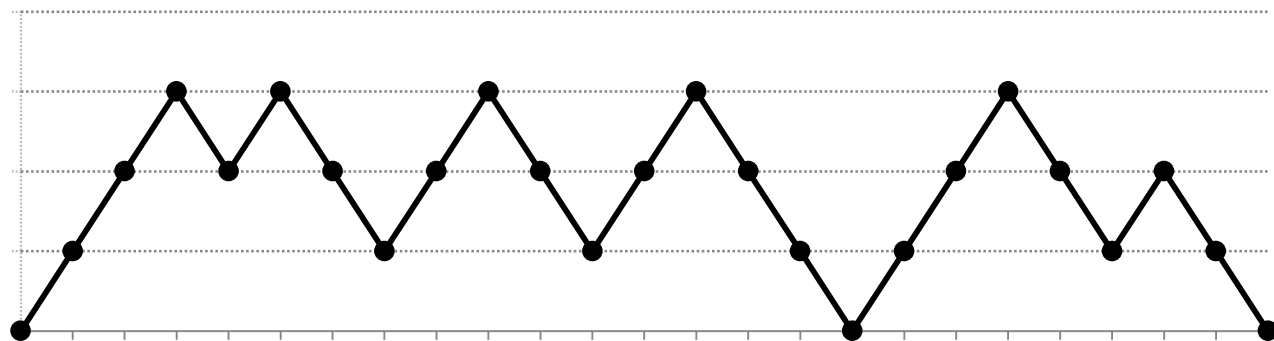
- the length is  $2 \cdot |V|$
- the number of 1s and 0s is the same
- furthermore, the number of 1s and 0s is the same for every partial subsequence that arise from any label of vertex (during the whole run of the algorithm)

# Reconstruction of Tree from Certificate - Example

$$f(0) = 0$$

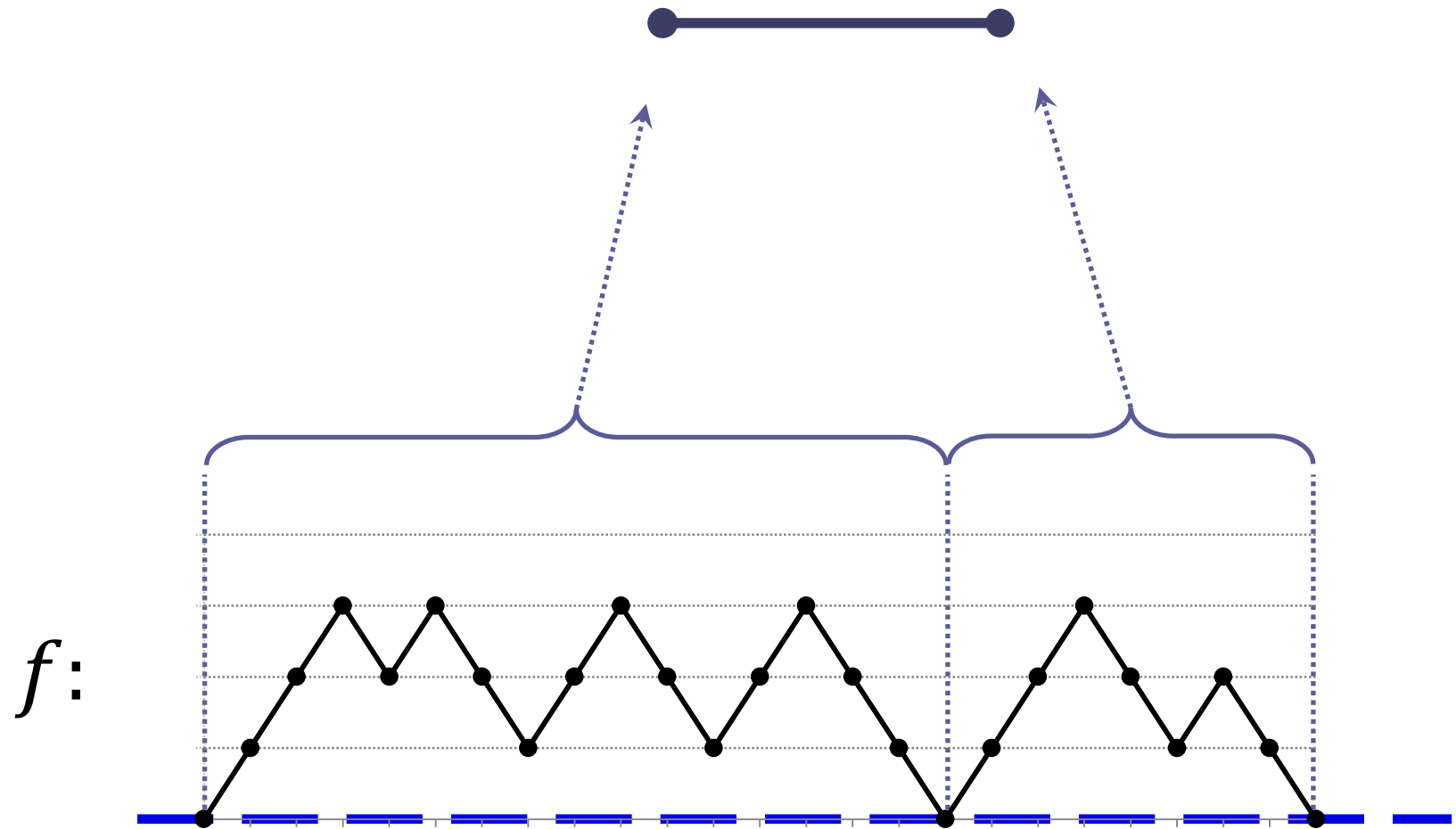
$$f(x + 1) = \begin{cases} f(x) + 1, & \text{Cert}(G)[x] = 0 \\ f(x) - 1, & \text{Cert}(G)[x] = 1 \end{cases}$$

$f$ :



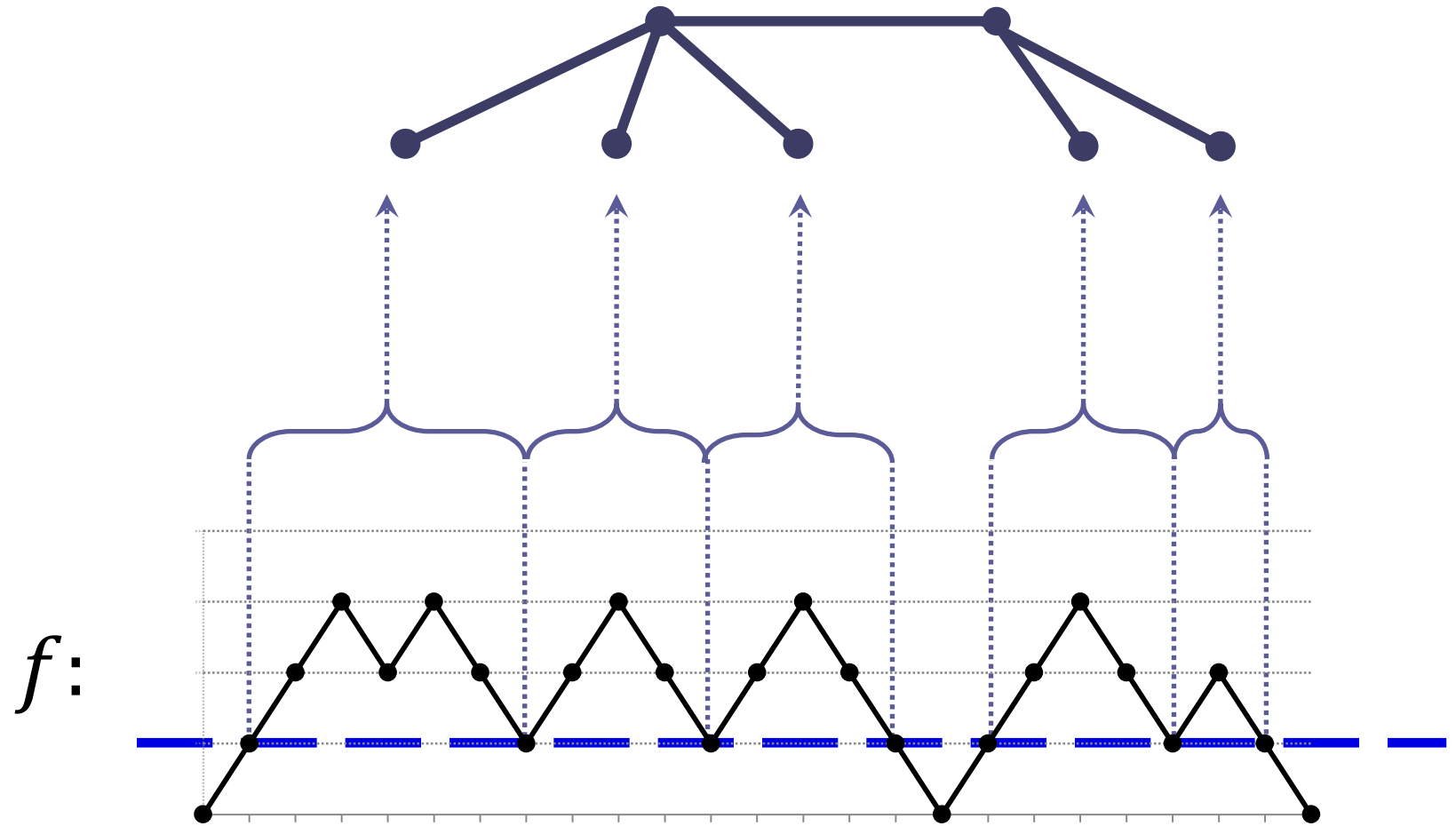
$$\text{Cert}(G) = 000101100110011100011011$$

# Reconstruction of Tree from Certificate - Example



$Cert(G) = 000101100110011100011011$

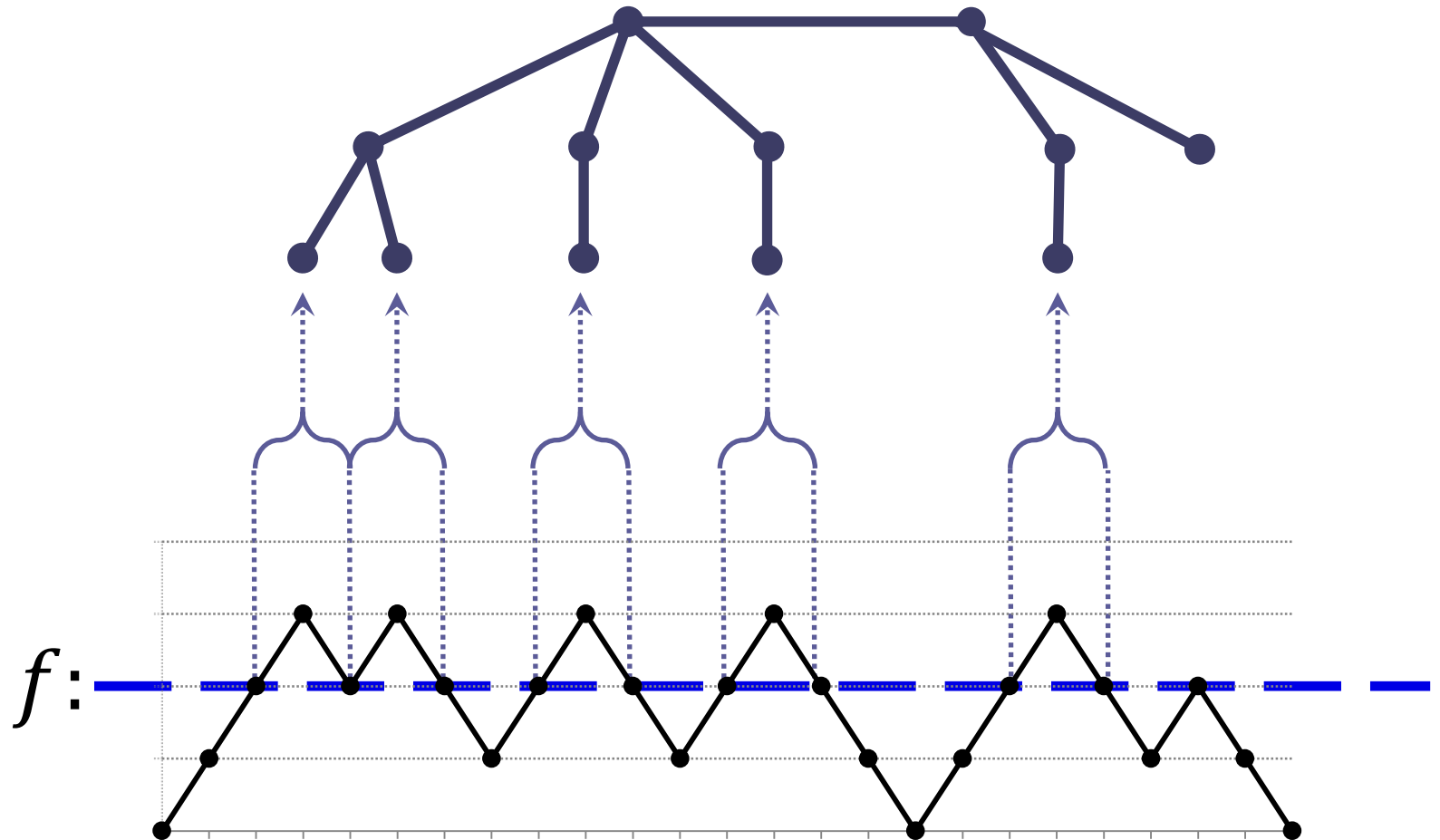
# Reconstruction of Tree from Certificate - Example



$f:$

$Cert(G) = 000101100110011100011011$

# Reconstruction of Tree from Certificate - Example



$Cert(G) = 000101100110011100011011$

# Reconstruction of Tree from Certificate

```
1) Function FINDSUBMOUNTAINS (integer  $l$ , certificate as string  $C$ ) : number of submountines in  $C$ 
2)  $k = 0$ ;  $M[0] =$  the empty string;  $f = 0$ ;
3) for  $x = l - 1$  to  $|C| - l$  do {
4)     if  $C[x] = 0$  then {  $f = f + 1$ ; } else {  $f = f - 1$ ; }
5)      $M[k] = M[k] \cdot C[x]$ ;
6)     if  $f = 0$  then {  $k = k + 1$ ;  $M[k] =$  the empty string;  $f = 0$ ; }
7) }
8) return  $k$ ;
```

---

```
1) Function CERTIFICATETOTREE (certificate as string  $C$ ) : tree as  $G = (V, E)$ 
2)  $n = \frac{|C|}{2}$ ;  $v = 0$ ;  $(V, E) =$  empty graph of order  $n$ ;  $V = \{0, \dots, n - 1\}$ ;
3)  $k =$  FINDSUBMOUNTAINS(1,  $C$ );
4) if  $k = 1$  then {  $Label[v] = M[0]$ ;  $v = v + 1$ ; }
5)     else {  $Label[v] = M[0]$ ;  $v = v + 1$ ;  $Label[v] = M[1]$ ;  $v = v + 1$ ;  $E = E \cup \{\{0,1\}\}$ ; }
6) for  $i = 0$  to  $n - 1$  do {
7)     if  $|Label[i]| > 2$  then {
8)          $k =$  FINDSUBMOUNTAINS(2,  $Label[i]$ );  $Label[i] = "01"$ ;
9)         for  $j = 0$  to  $k - 1$  do {  $Label[v] = M[j]$ ;  $E = E \cup \{\{i, v\}\}$ ;  $v = v + 1$ ; }
10)    }
11) return  $G = (V, E)$ ;
```

$O(|C|^2)$

# Reconstruction of Tree from Certificate

```
1) Function FASTCERTIFICATETOTYPE (certificate as string  $C$ ) : tree as  $G = (V, E)$ 
2)  $(V, E) =$  empty digraph of order  $\frac{|C|}{2}$ ;  $V = \{0, \dots, \frac{|C|}{2}\}$ ;
3)  $n = 0$ ;
4)  $p = n$ ;
5) for  $x = 1$  to  $|C| - 2$  do {
6)     if  $C[x] = 0$  then {
7)          $n = n + 1$ ;
8)          $E = E \cup \{(p, n)\}$ ;
9)          $p = n$ ;
10)    } else {
11)        if  $\text{parent}(p)$  does not exist then {
12)             $n = n + 1$ ;
13)             $E = E \cup \{(p, n)\}$ ;
14)             $p = n$ ;
15)        } else {
16)             $p = \text{parent}(p)$ ;
17)        }
18)    }
19) }
```

20) **return**  $G = (V, \text{remove\_orientation}(E))$ ;

$O(|C|)$

# References

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