Lecture
8_6.1

Alberi (Trees)

cini
Cybersecurity
National Lab

Paolo PRINETTO
Politecnico di Torino (Italy)
Univ. of Illinois at Chicago, IL (USA)
CINI Cybersecurity Nat. Lab. (Italy)

Paolo.Prinetto@polito.it

www.consorzio-cini.it
www.comitato-girotondo.org

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Rel. 03.03.2019

Goal

– **This lecture aims at presenting the ADT *Tree*, the related operations, and the visiting techniques.**

Rel. 03.03.2019

Prerequisites

- **none**

Rel. 03.03.2019

Further readings

- **Students interested in a deeper look at the covered topics can refer, for instance, to the books listed at the end of the lecture.**

Rel. 03.03.2019

Outline

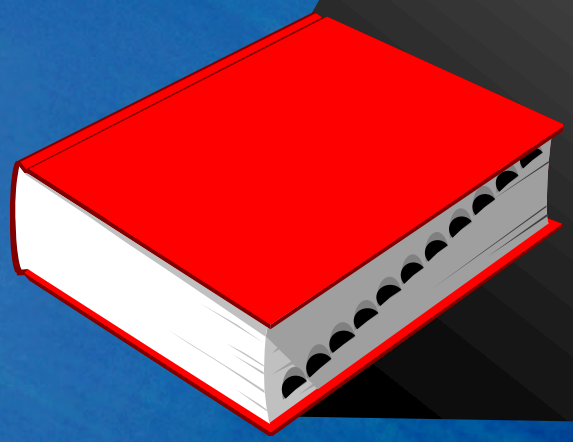
- Concetto di albero
- Operazioni sugli alberi
- Rappresentazione degli alberi
- Alberi binari
- Visita di un albero binario

Rel. 03.03.2019

Outline

- Concetto di albero
- Operazioni sugli alberi
- Rappresentazione degli alberi
- Alberi binari
- Visita di un albero binario

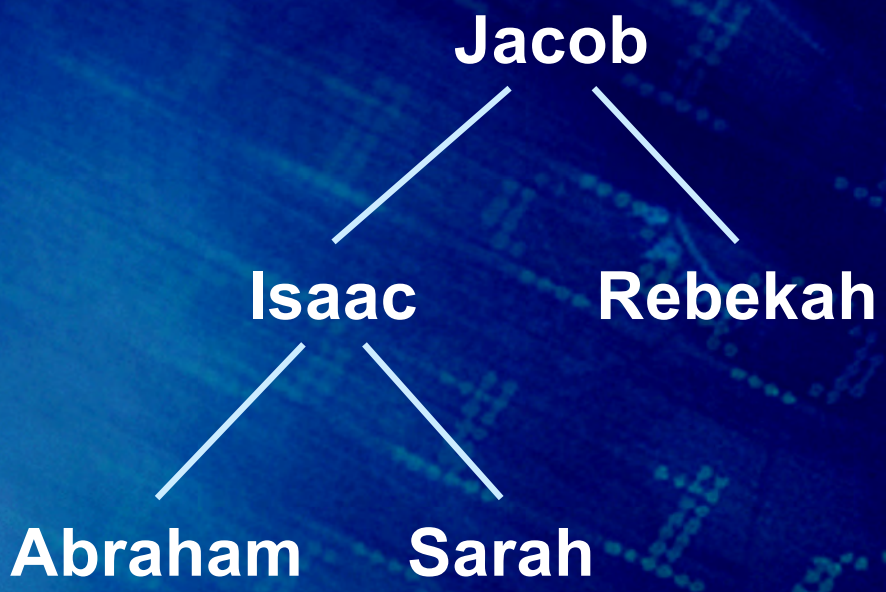
Rel. 03.03.2019



Albero - Tree

Struttura dati utilizzata per memorizzare un insieme di elementi tra i quali sia possibile stabilire una relazione gerarchica

Esempio: Alberi Genealogici



Rel. 03.03.2019

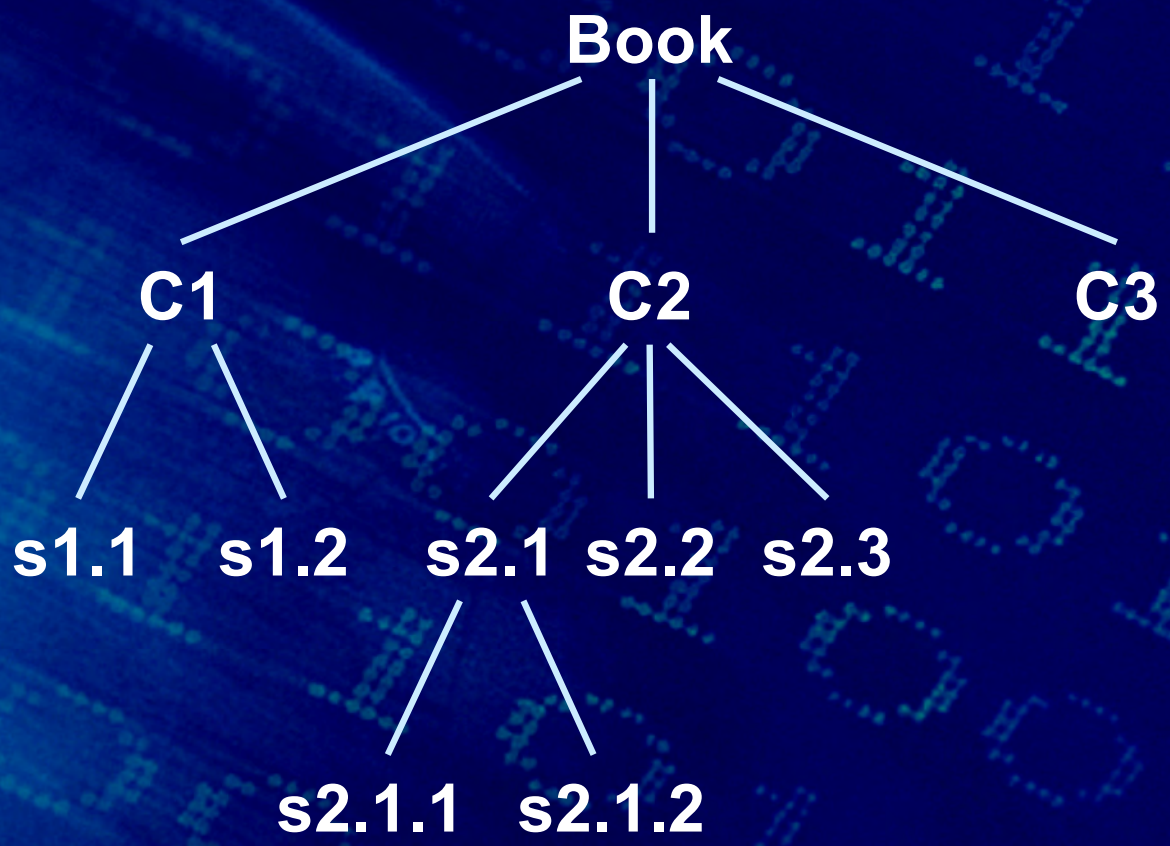
Esempio: Indici

- Book
  - C1
    - s1.1
    - s1.2
  - C2
    - s2.1
      - s2.1.1
      - s2.1.2
    - s2.2
    - s2.3
  - C3

Rel. 03.03.2019

Esempio: Indici

- Book
  - C1
    - s1.1
    - s1.2
  - C2
    - s2.1
      - s2.1.1
      - s2.1.2
    - s2.2
    - s2.3
  - C3



Rel. 03.03.2019

Albero: Definizione Formale

Un albero è una coppia ordinata (V, E) di insiemi:

- V è un insieme finito e non vuoto di oggetti, denominati nodi, tra i quali esiste una qualche relazione (parentela).
  Un nodo viene distinto dagli altri e prende il nome di radice.
- E è un insieme di archi. Ciascun arco unisce due nodi.

Rel. 03.03.2019

Struttura dei nodi

In ogni nodo vengono normalmente memorizzati:

- uno degli elementi dell'insieme
- informazioni necessarie a memorizzare la struttura dell'albero.

Rel. 03.03.2019

Albero: Definizione Assiomatica

- Un singolo nodo è di per sé un albero
- Supponendo che n sia un nodo e che $T_1, T_2, \ldots, T_k$ siano alberi con radici $n_1, n_2, \ldots, n_k$, è possibile costruire un nuovo albero facendo in modo che n sia il padre di $n_1, n_2, \ldots, n_k$

Rel. 03.03.2019

Albero: Definizione Assiomatica

- Un singo
- Suppon , T_2, … , T_k siano all ossibile costruire do che n sia il pad

In questo nuovo albero:

- *n è la radice*
- *T_1, T_2, …, T_k sono i sottoalberi della radice.*

Rel. 03.03.2019

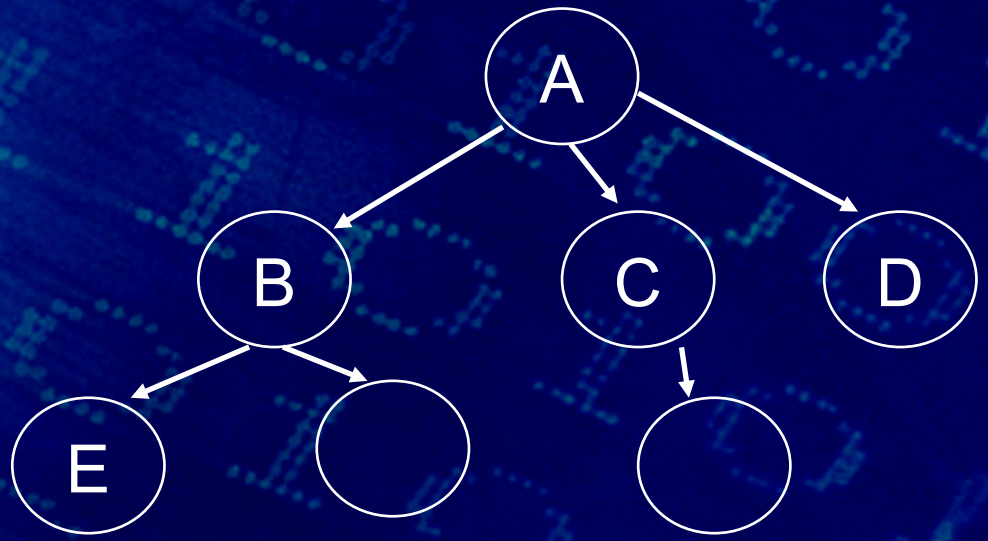
Padre - Figlio

- **Se esiste un arco che collega il nodo *a* e il nodo *b*, allora:**
  - ***a* è il padre di *b***
  - ***b* è un figlio di *a*.**

Rel. 03.03.2019

Padre - Figlio

- **Se esiste un arco che collega il nodo *a* e il nodo *b*, allora:**
  - ***a* è il padre di *b***
  - ***b* è un figlio di *a*.**



A ha 3 figli: B, C, D

A è il padre di B

Rel. 03.03.2019

Cammino (Path)

Cammino da un nodo n_1 a un nodo n_k **è una sequenza di nodi:**

$$n_1, n_2, \ldots, n_k$$

tali che

- n_{j+1} **è un figlio di** n_j $\quad \forall\, j : 1 \leq j < k$

Ascendente - Discendente

- **Se esiste un cammino tra il nodo *a* e il nodo *b*, allora:**
  - ***a* è un ascendente (ancestor) di *b***
  - ***b* è un discendente (descendant) di *a*.**

Rel. 03.03.2019

Teorema

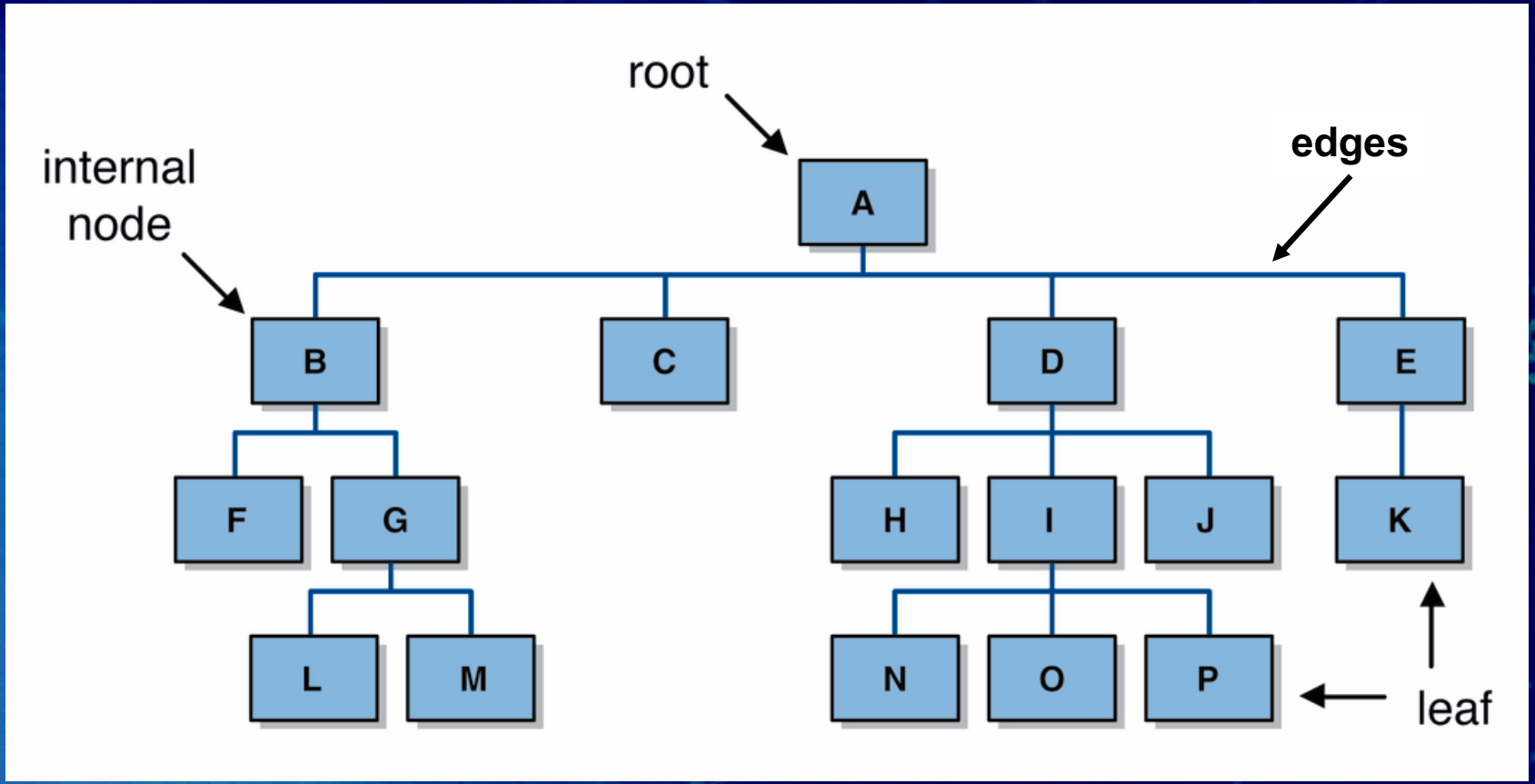
Se esiste un cammino tra il nodo a e il nodo b, tale cammino è unico

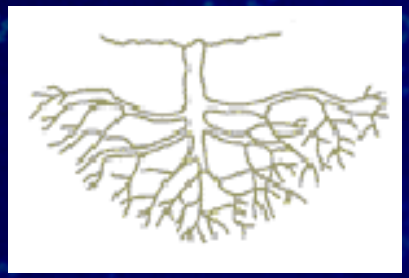
Ulteriori Definizioni

- **Foglia (leaf)**
  - Un nodo senza discendenti è detto foglia, o nodo terminale
- **Nodo interno**
  - Un nodo con discendenti è detto nodo interno, o intermedio

Rel. 03.03.2019

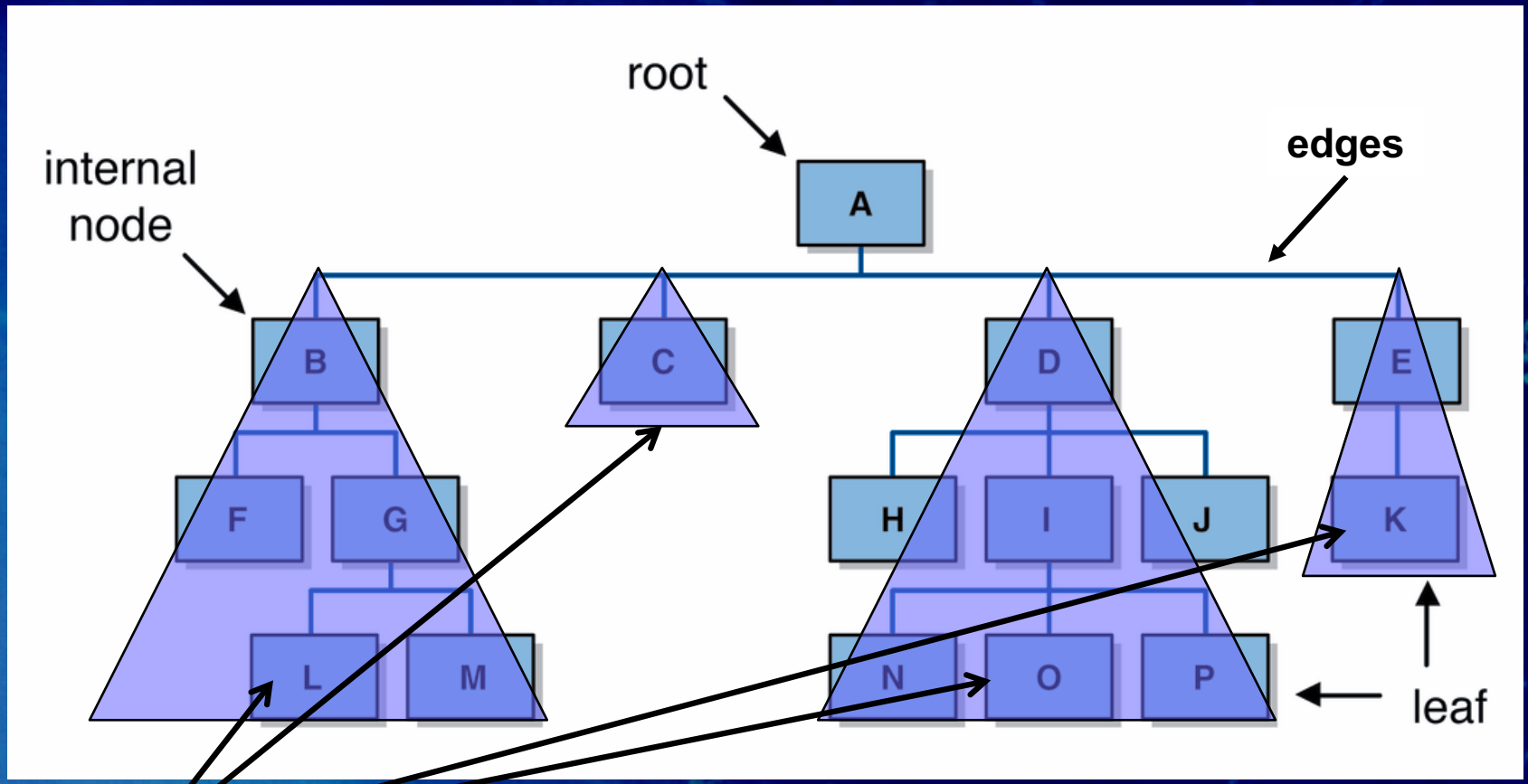
Tree diagram



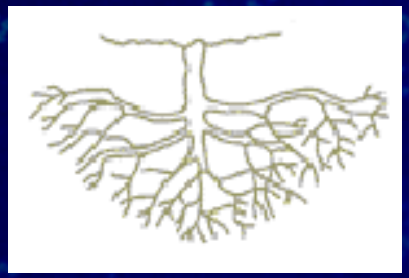


Rel. 03.03.2019

Tree diagram



Subtrees



Rel. 03.03.2019

Profondità (Depth)

La *profondità* di un nodo è la lunghezza dell'unico cammino esistente tra la radice e il nodo

La radice ha profondità 0

Rel. 03.03.2019

Altezza (height)

- **Altezza di un nodo** è la lunghezza del massimo cammino da quel nodo a una foglia.
- **Altezza di un albero** è l'altezza della radice.

Nel caso in cui tutte le foglie abbiano la stessa profondità e l'altezza dell'albero sia *H*, per ogni nodo *i* si ha:

$$p(i) + h(i) = H$$

dove:

- *p(i)* : profondità del nodo i
- *h(i)* : altezza del nodo i.

Rel. 03.03.2019

Grado

- **Grado di un nodo** è il numero dei discendenti diretti (figli) di quel nodo.
- **Grado di un albero** è il massimo grado dei suoi nodi.

- Un albero di grado 2 viene detto **albero binario**.

Rel. 03.03.2019

Ordine

Solitamente i figli di un nodo sono *ordinati* da sinistra a destra

Esempio di alberi diversi:

Rel. 03.03.2019

Outline

- Concetto di albero
- Operazioni sugli alberi
- Rappresentazione degli alberi
- Alberi binari
- Visita di un albero binario

Rel. 03.03.2019

Operazioni sugli alberi

Tra le molteplici operazioni che si possono eseguire sugli alberi, le più significative sono le seguenti:

Rel. 03.03.2019

parent(n,T)

Ritorna:

- **il padre del nodo *n* nell'albero *T***
- **il nodo nullo se *n* è la radice**

leftmost_child(n,T)

Ritorna:

- **il figlio più a sinistra del nodo *n* nell'albero *T***
- **il nodo nullo se *n* è una foglia**

right_sibling(n,T)

Ritorna:

- **il fratello di destra del nodo *n* nell'albero *T***
- **il nodo nullo se non esiste**

root(T)

Ritorna:

- **il nodo radice dell'albero *T***
- **nodo nullo se *T* è un albero nullo**

makenull(T)

Trasforma l'albero *T* in un albero nullo

Create_node($v, T_1, T_2, \ldots, T_k$)

Crea un nuovo nodo *r* con label *v* e *k* figli, radici dei sottoalberi $T_1, T_2, \ldots, T_k$.

Ritorna l'albero avente radice *r*

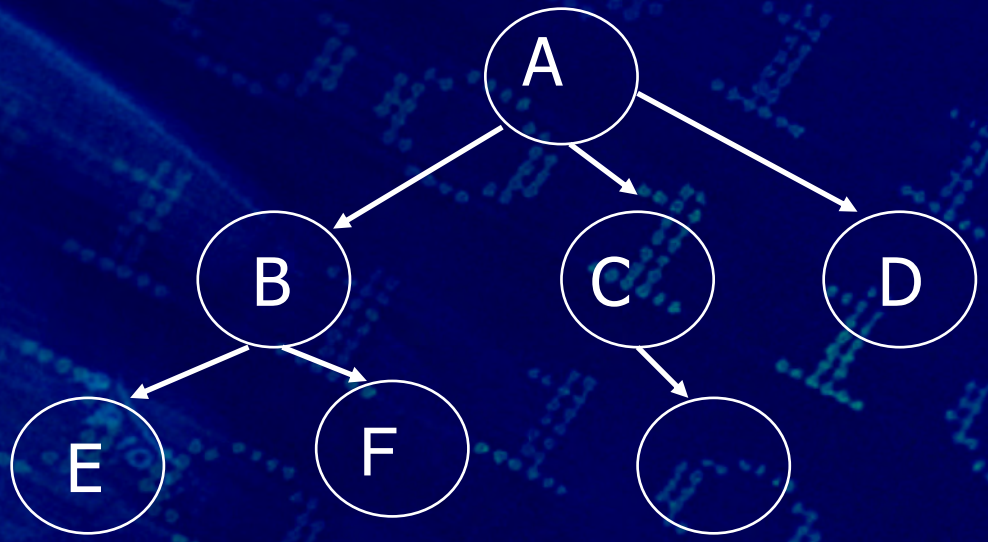
Delete_node(n,T)

Cancella il nodo *n* nell'albero *T*

Ritorna:
Il nuovo albero T privo del nodo n se la cancellazione è stata possibile;
Altrimenti ERROR

Deleting a node

Deletion of a child B

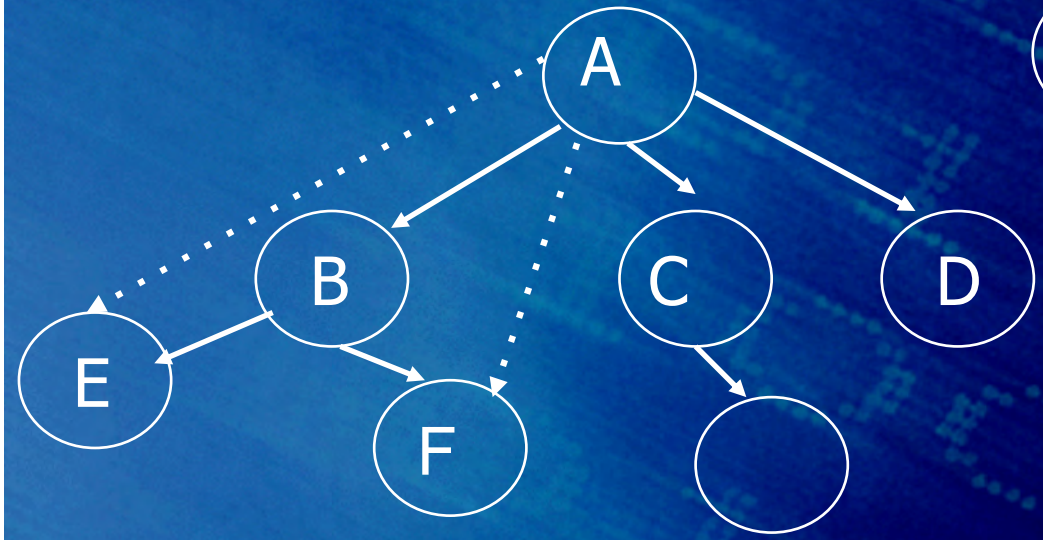


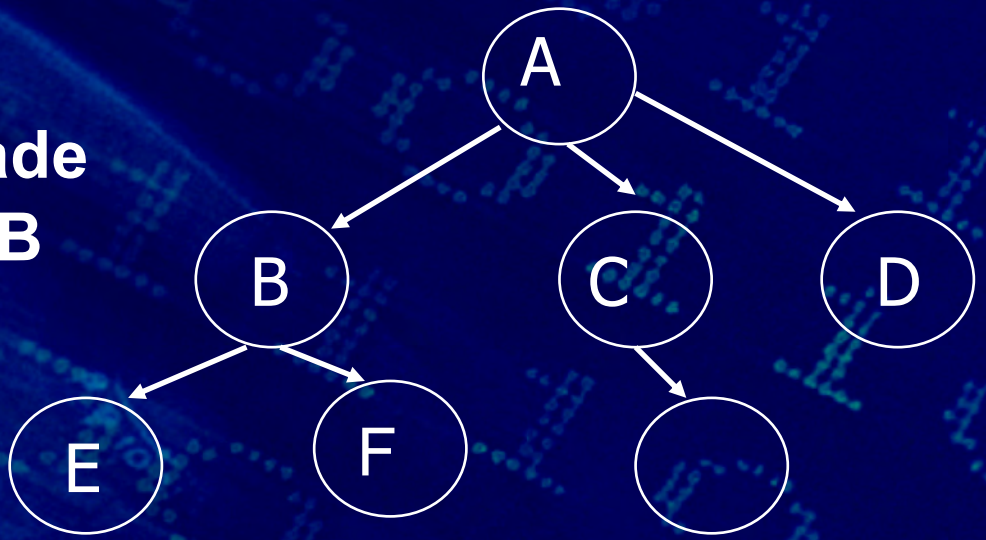
Rel. 03.03.2019

Deleting a node

Deletion of a child B:

- **Children of B are first made children of the parent of B**



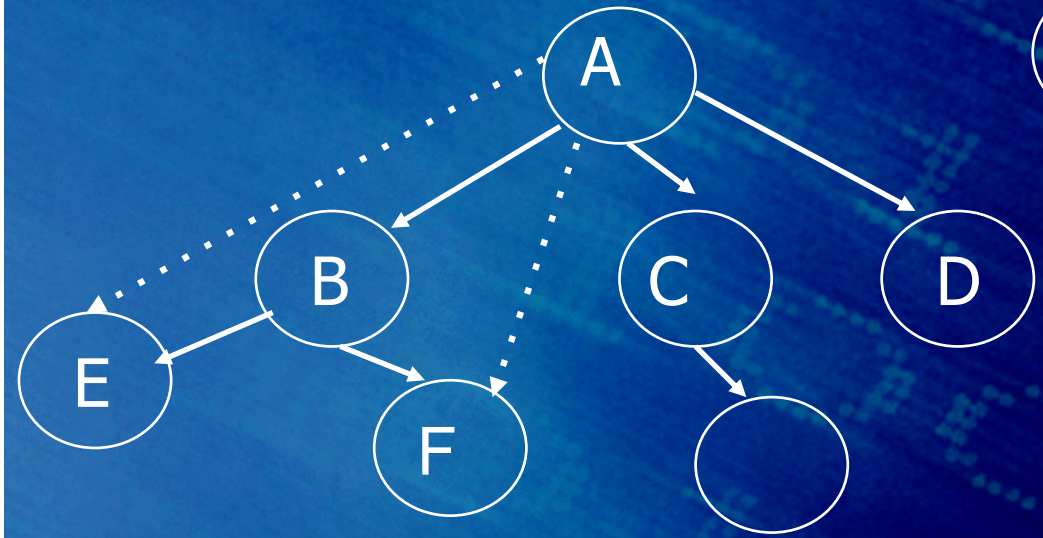


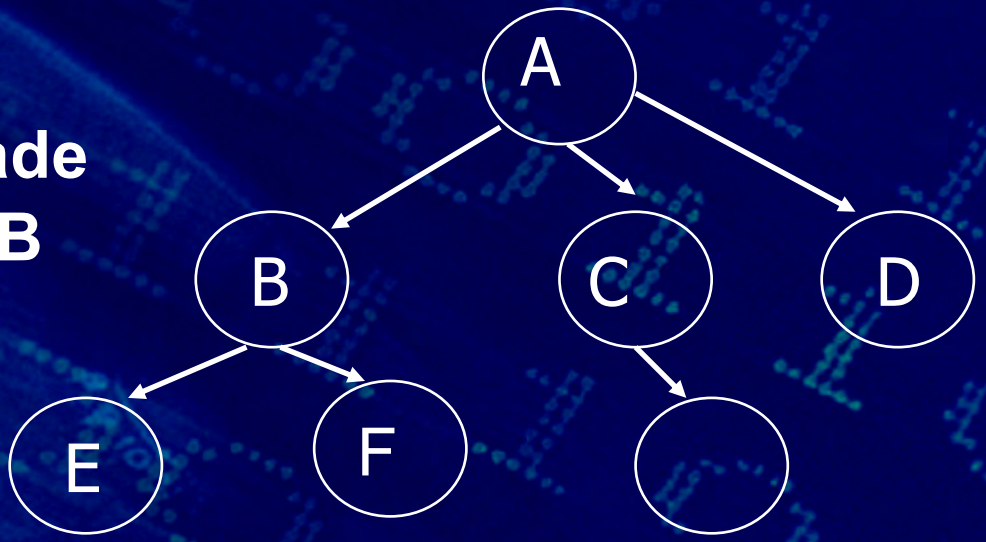
Rel. 03.03.2019

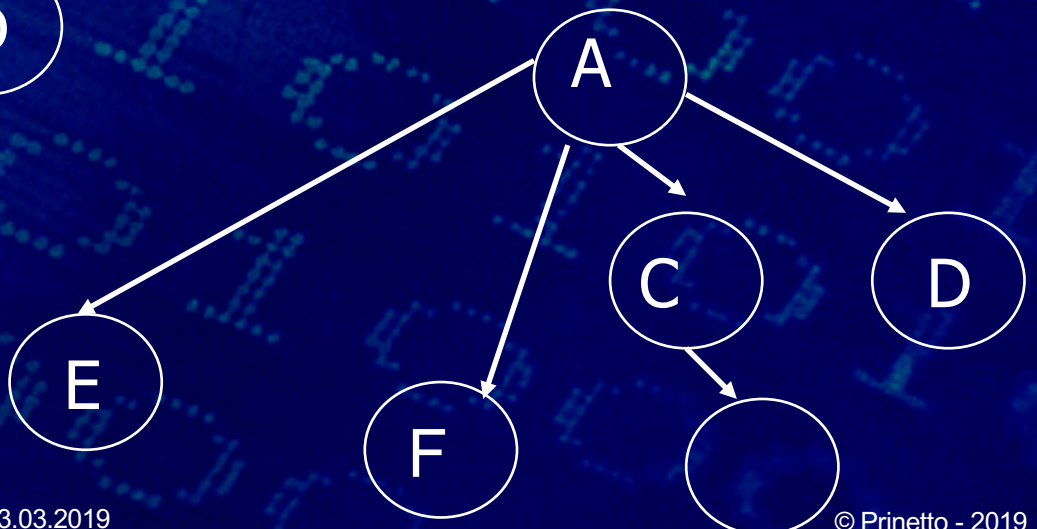
Deleting a node

Deletion of a child B:

- **Children of B are first made children of the parent of B**
- **Node B is deleted.**





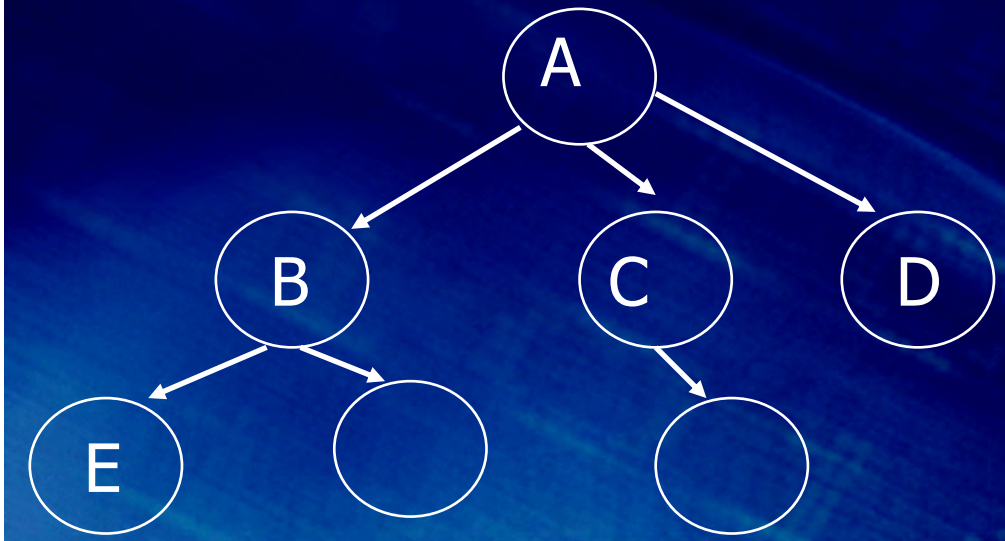


Rel. 03.03.2019

Deleting the root

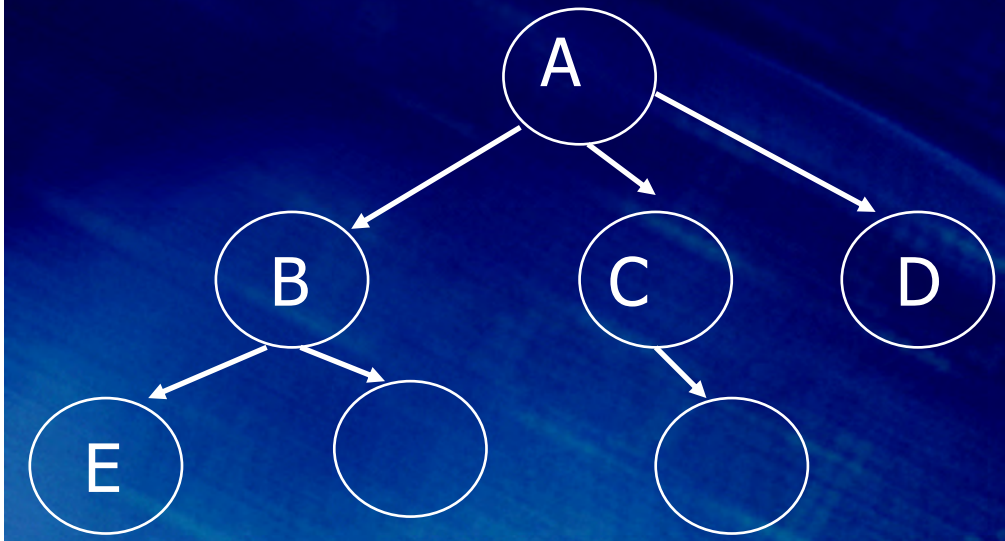
- One of the children becomes the new root
- Other children of old root become the children of the new root

Rel. 03.03.2019



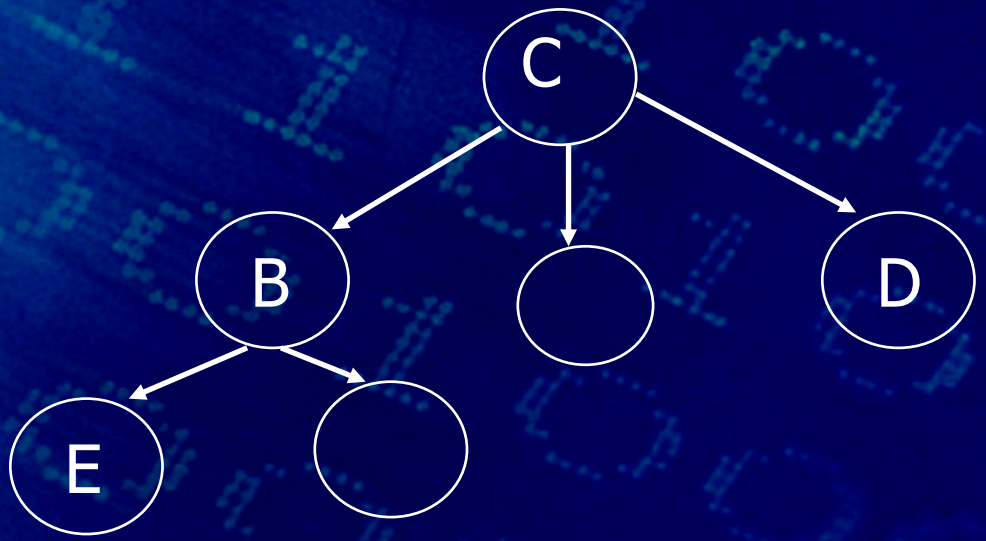
C becomes new root

B and D are children of C in addition to its original child



C becomes new root

B and D are children of C in addition to its original child



Rel. 03.03.2019

Outline

- Concetto di albero
- Operazioni sugli alberi
- Rappresentazione degli alberi
- Alberi binari
- Visita di un albero binario

Rel. 03.03.2019

Rappresentazione degli Alberi

Sono possibili soluzioni diverse:

- tramite vettore
- tramite liste dei figli
- tramite liste multiple.

La scelta di una determinata rappresentazione sarà principalmente determinata dall'insieme delle operazioni da effettuare.

Rel. 03.03.2019

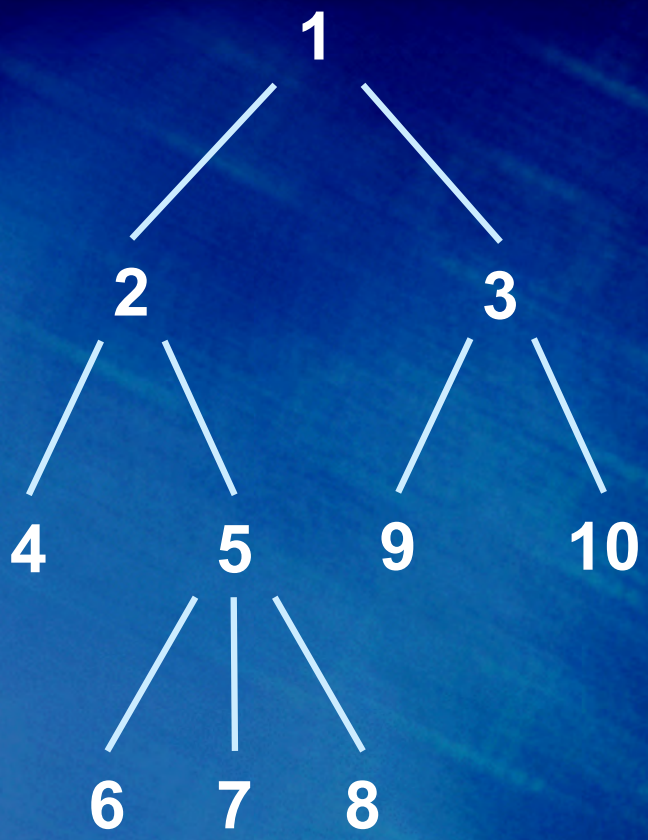
Rappresentazione tramite vettore

Se il numero *m* di nodi è noto a priori, è possibile memorizzare in ciascuna cella di un vettore di *m* elementi il puntatore al padre del nodo cui l'elemento è associato.

I nodi vengono solitamente numerati per livello, dalla radice verso le foglie e da sinistra verso destra.

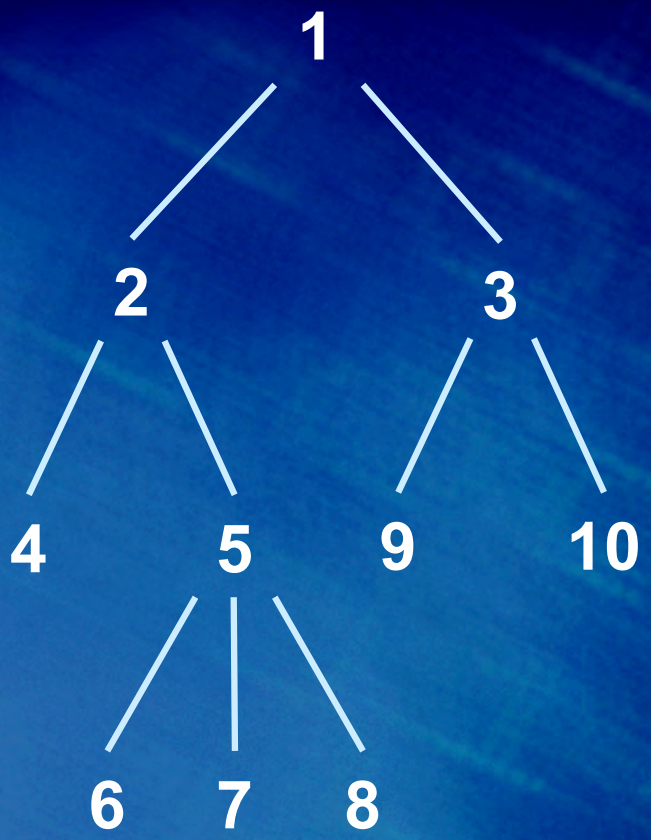
Rel. 03.03.2019

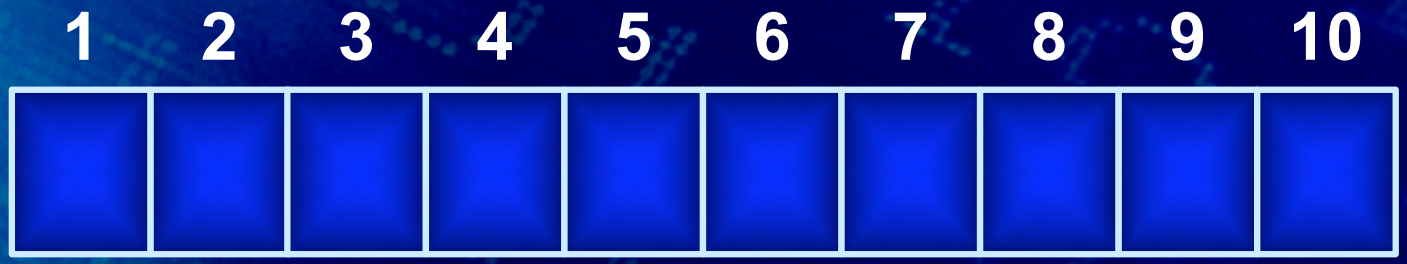
Rappresentazione tramite vettore: Esempio



Rel. 03.03.2019

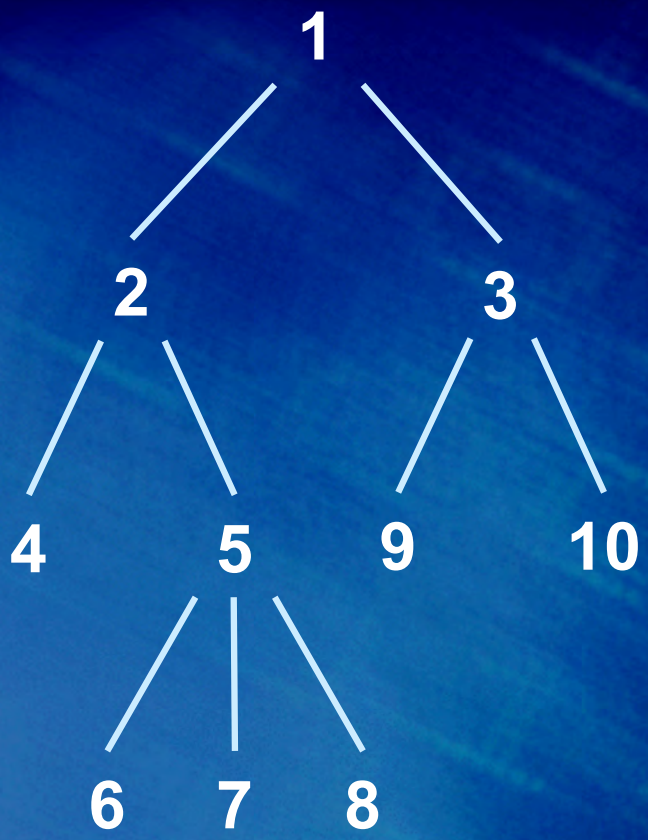
Rappresentazione tramite vettore: Esempio





Rel. 03.03.2019

Rappresentazione tramite vettore: Esempio



1	2	3	4	5	6	7	8	9	10
0	1	1	2	2	5	5	5	3	3

Rel. 03.03.2019

Rappresentazione tramite vettore: Vantaggi e Svantaggi

Dato un nodo,

- **la determinazione del padre è immediata**
- **risulta particolarmente disagevole la determinazione sia dei figli sia dei fratelli.**

Rel. 03.03.2019

Rappresentazione tramite liste dei figli

Un vettore, avente tante celle quanti sono i nodi dell'albero, contiene i puntatori alle liste dei figli di ciascun nodo.

Rel. 03.03.2019

Rappresentazione tramite liste dei figli: Esempio

Rel. 03.03.2019

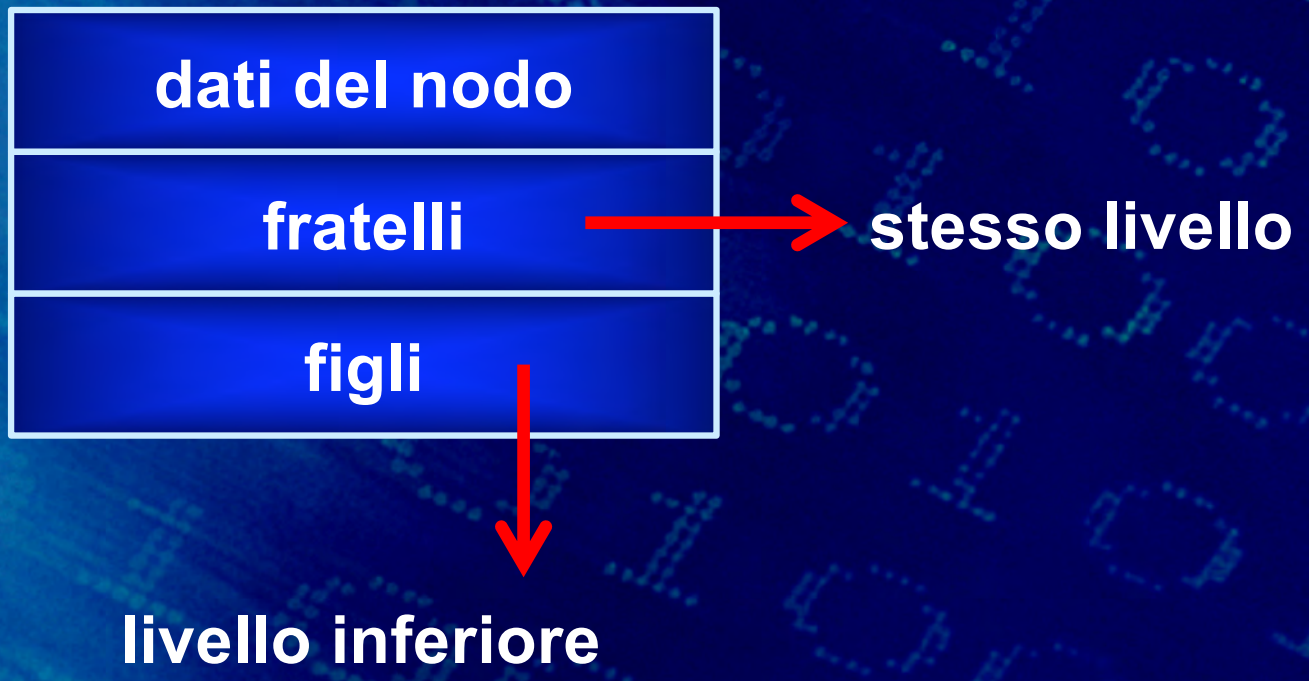
Rappresentazione tramite liste dei figli: Vantaggi e Svantaggi

Dato un nodo, risulta particolarmente disagevole la determinazione del padre.

Rel. 03.03.2019

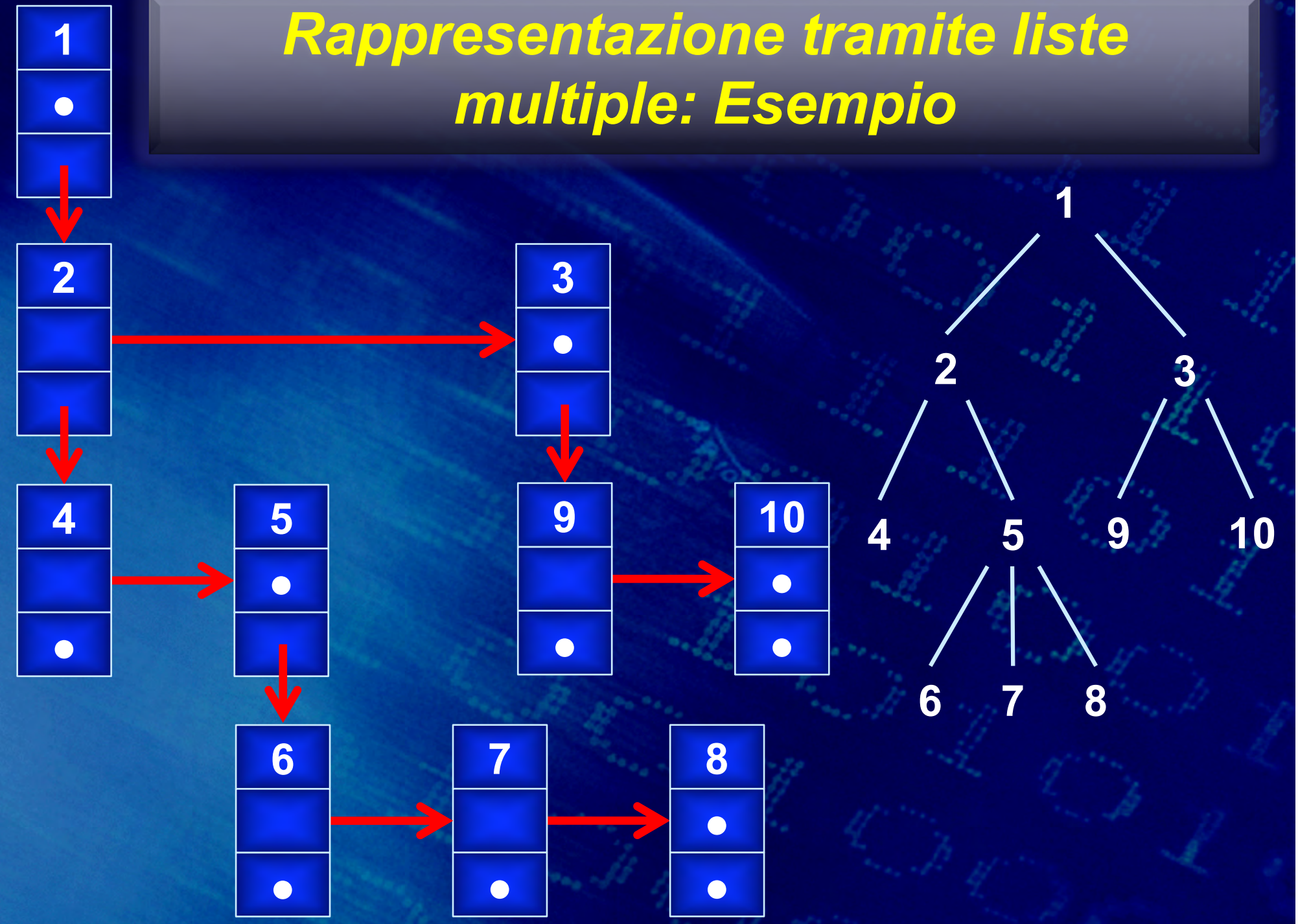
Rappresentazione tramite liste multiple

- Si può ricorrere a una lista multipla, in cui ciascun elemento contiene due puntatori: uno al figlio più a sinistra e uno al fratello di destra.



Rel. 03.03.2019

Rappresentazione tramite liste multiple: Esempio



Rel. 03.03.2019

Rappresentazione tramite liste multiple: Vantaggi e Svantaggi

Dato un nodo, risulta particolarmente disagevole la determinazione del padre.

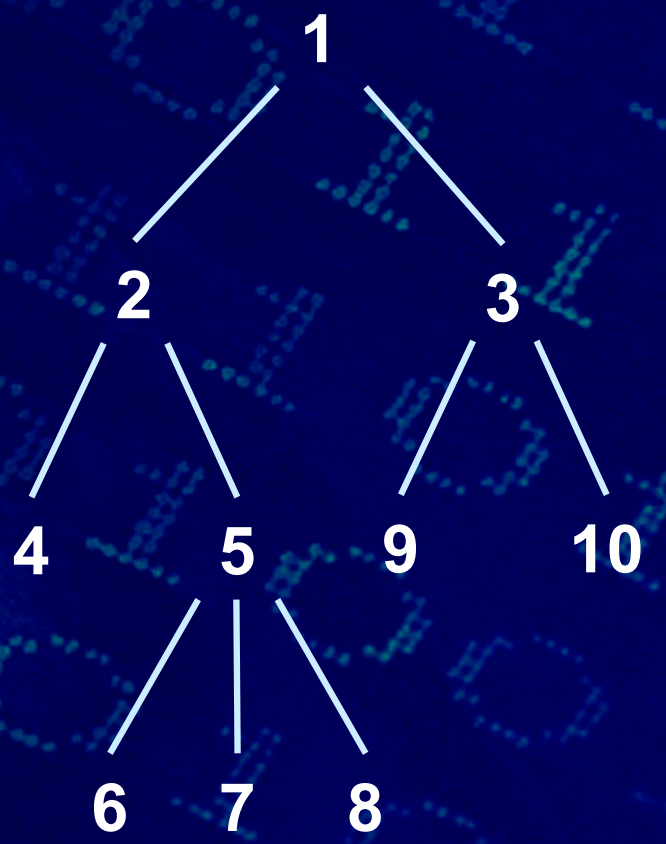
Rel. 03.03.2019

Rappresentazione completa

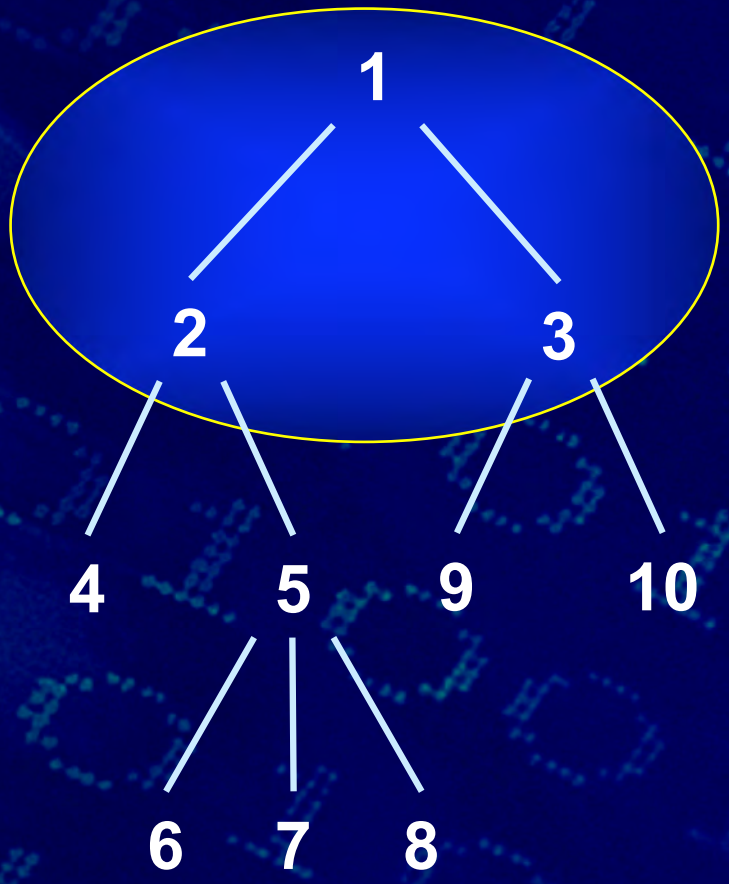
Per agevolare la determinazione del padre è possibile aggiungere a ciascun elemento della lista un puntatore al nodo padre.

Rel. 03.03.2019

Rappresentazione completa: Esempio

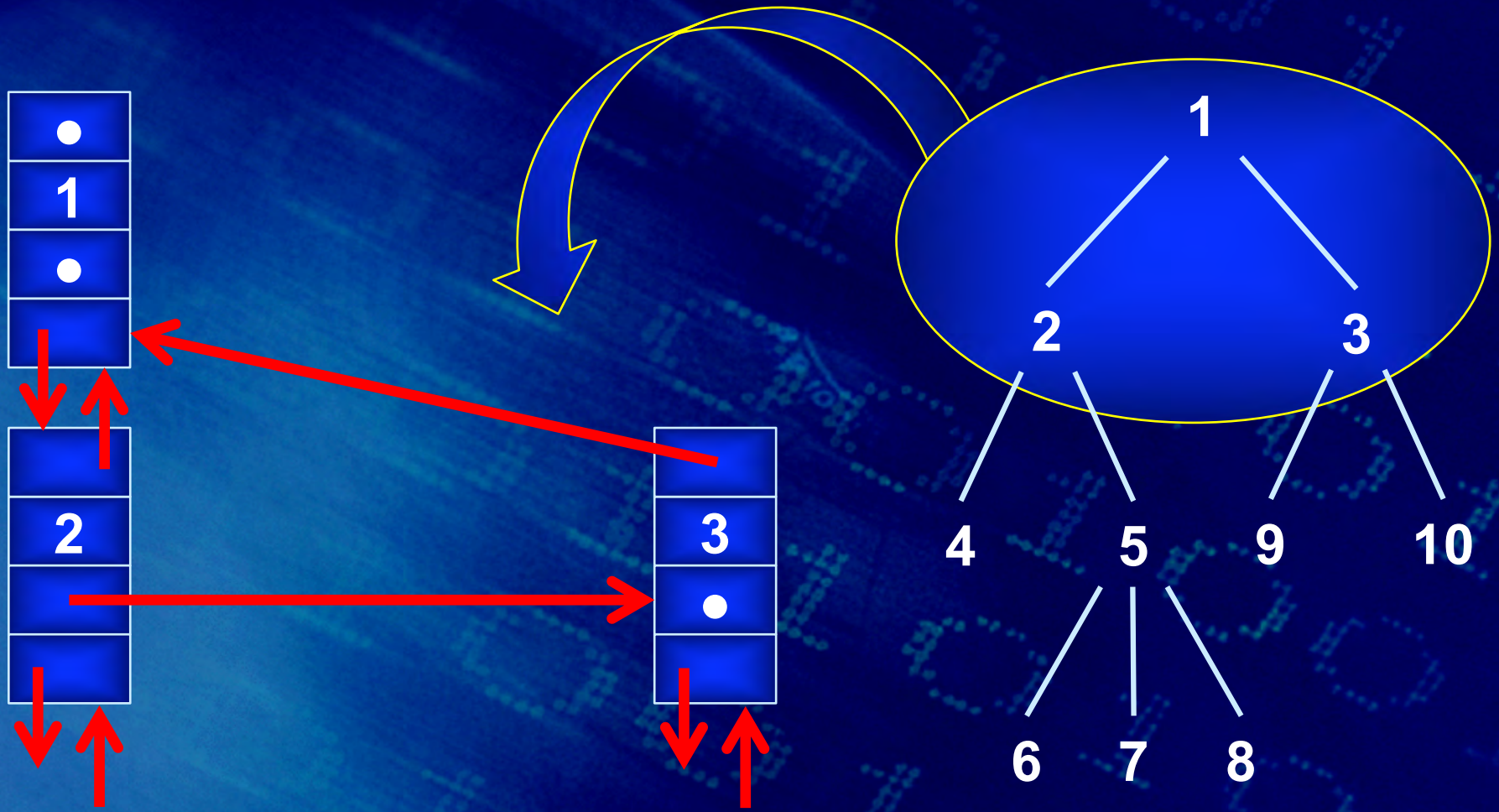


Rappresentazione completa: Esempio



Rel. 03.03.2019

Rappresentazione completa: Esempio



Rel. 03.03.2019

Outline

- Concetto di albero
- Operazioni sugli alberi
- Rappresentazione degli alberi
- **Alberi binari**
- Visita di un albero binario

Rel. 03.03.2019

Alberi Binari

Gli alberi binari (o binary tree) sono alberi di grado 2, vale a dire tali che ciascun nodo può avere:

- **nessun figlio**
- **un figlio *sinistro***
- **un figlio *destro***
- **sia un figlio *sinistro* sia un figlio *destro*.**

Rel. 03.03.2019

Alberi Binari *(cont'd)*

La distinzione tra sinistra e destra per i figli di un nodo può essere estesa a nodi che non sono legati da relazione ancestor-descendant.

Regola fondamentale:

- **Se *a* e *b* sono fratelli e *a* è alla sinistra di *b*, allora tutti i discendenti di *a* sono alla sinistra di tutti i discendenti di *b*.**

Rel. 03.03.2019

Alberi Binari *(cont'd)*

Regola pratica:

- **Dato un nodo *n*, per trovare i nodi alla sua sinistra e alla sua destra basta percorrere il cammino che va da *n* alla radice; tutti i nodi che si dipartono dalla sinistra (e i loro eventuali discendenti) si trovano a sinistra di *n*.**

Discorso analogo vale per la destra.

Rel. 03.03.2019

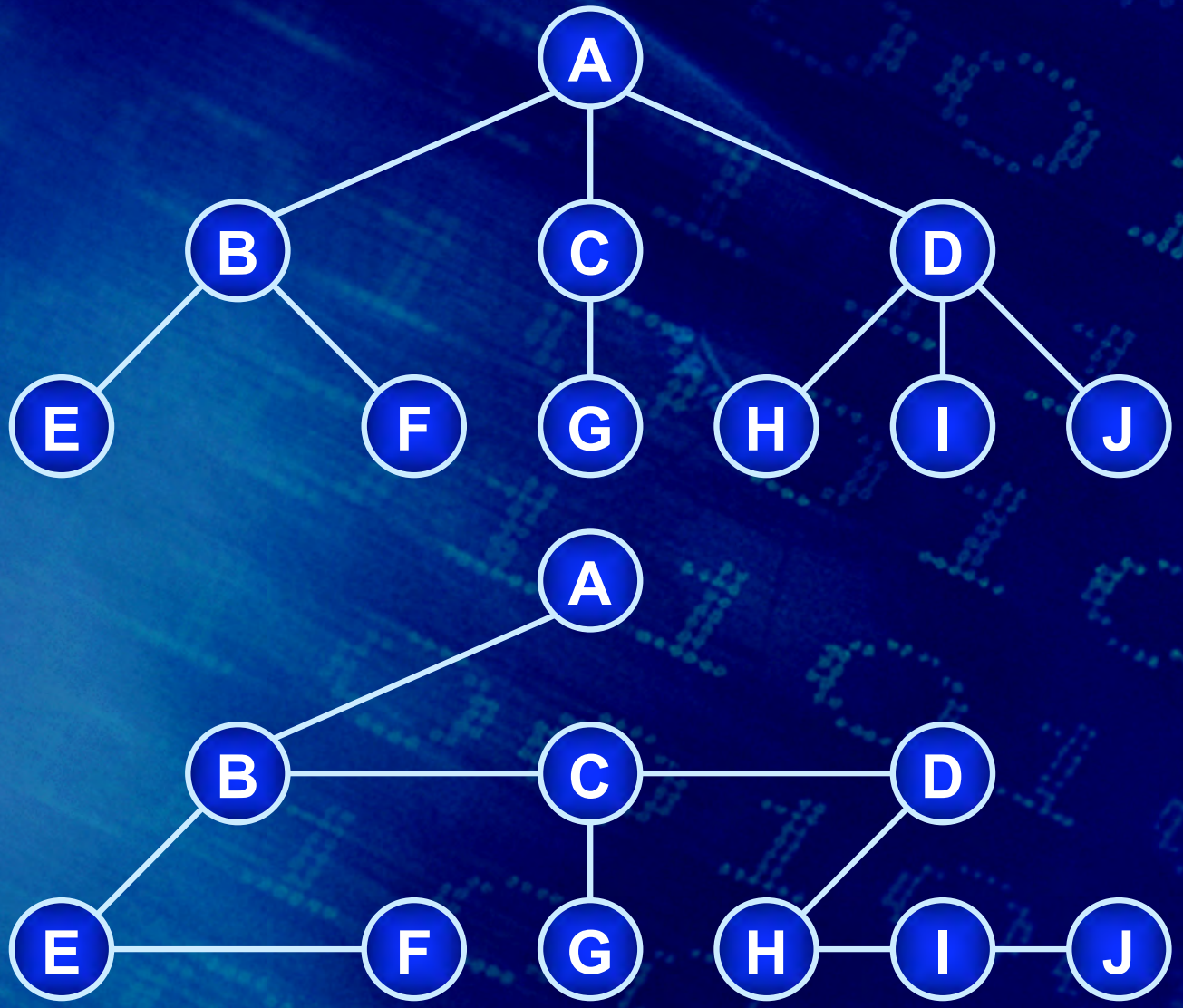
Trasformazione alberi binari/alberi normali

Qualunque albero può essere trasformato in un albero binario equivalente, e viceversa.

- **Trasformazione normale → binario**
  - **per ogni nodo, si collegano tutti i figli di tale nodo**
  - **si cancellano tutti i rami dal nodo in esame ai suoi figli, eccetto quello con il figlio più a sinistra.**

Rel. 03.03.2019

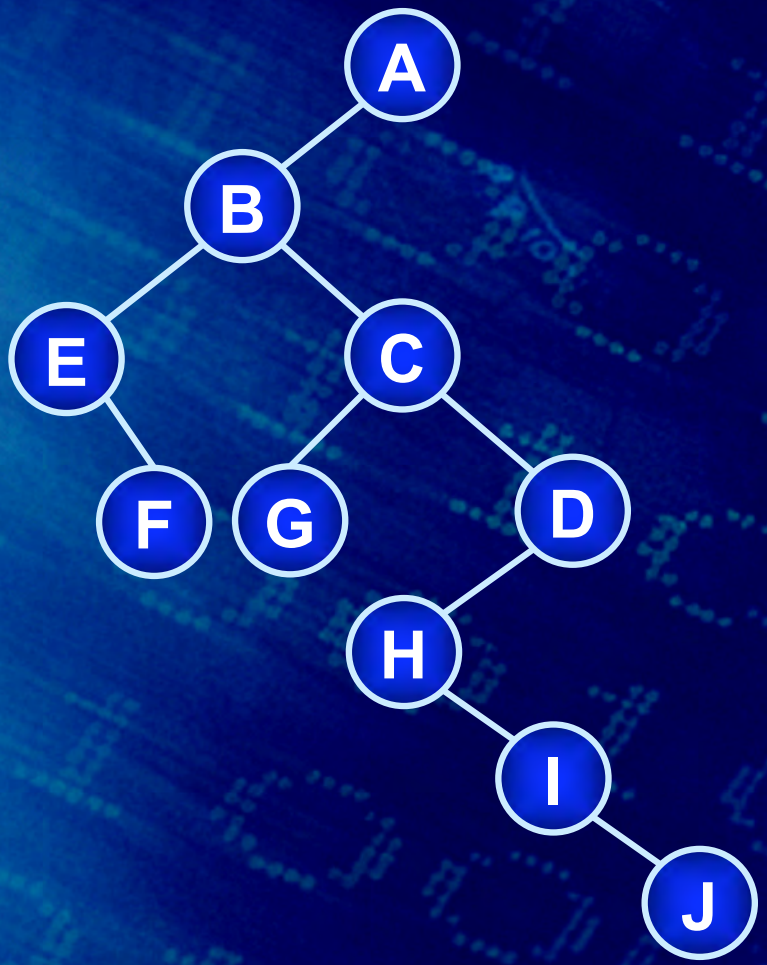
Da alberi normali ad alberi binari: esempio



Rel. 03.03.2019

Da alberi normali ad alberi binari: esempio
(cont'd)

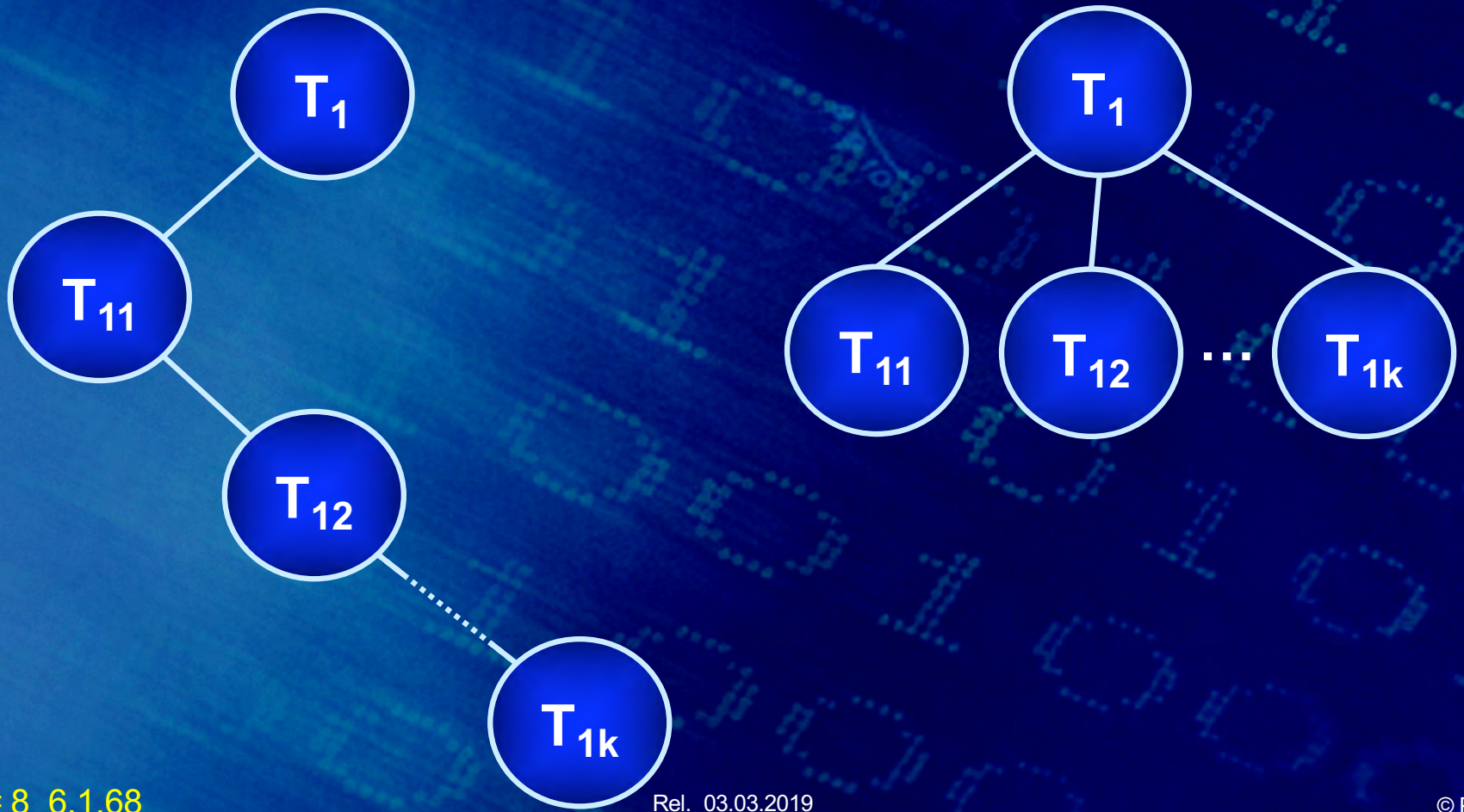
L'albero ottenuto non sembra un albero binario; ruotando di 45° in senso orario si ottiene:



Rel. 03.03.2019

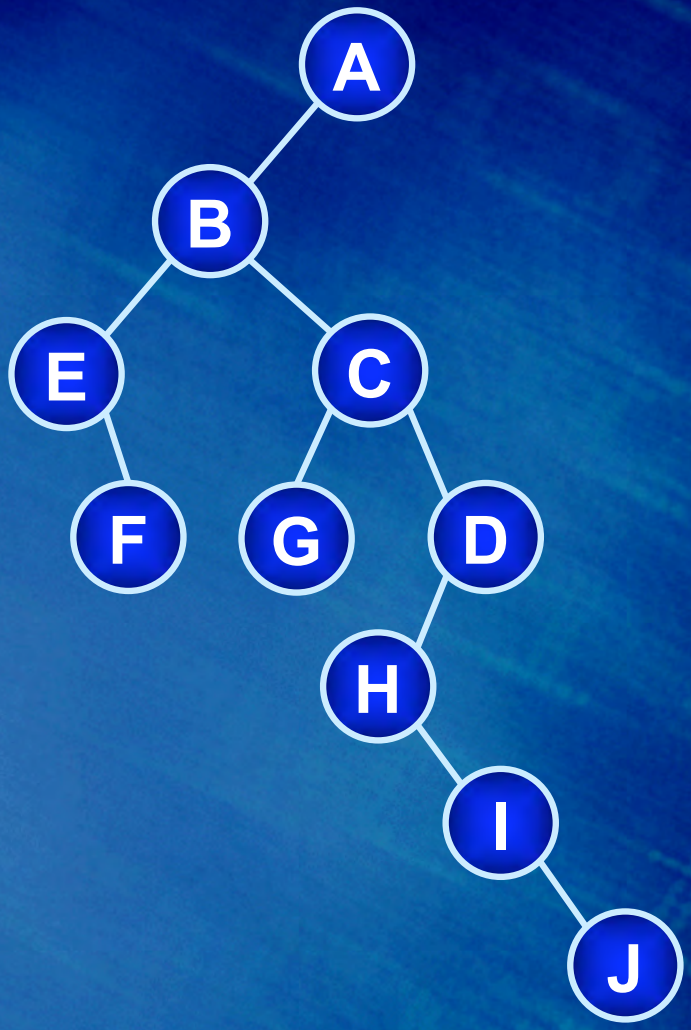
Trasformazione binario → normale

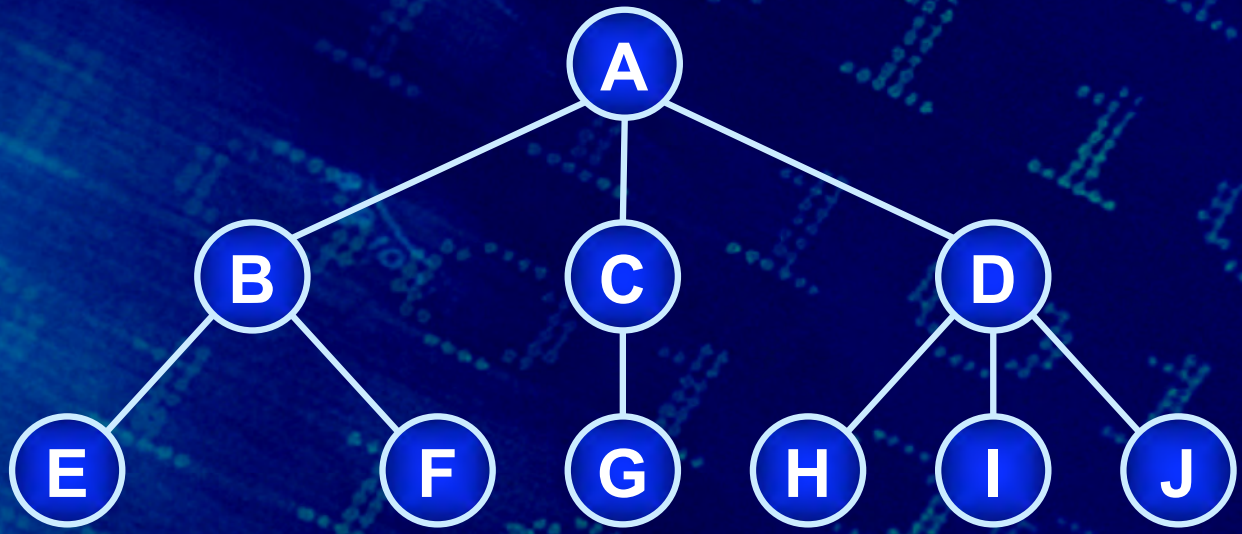
- i figli sinistri rappresentano i figli
- i figli destri rappresentano i fratelli



Rel. 03.03.2019

Trasformazione binario → normale: esempio





Rel. 03.03.2019

Numero di nodi per livello

- **Numero massimo di nodi per livello**
  - **Il numero massimo di nodi aventi profondità *i* in un albero binario è pari a 2^i**

Rel. 03.03.2019

Numero di nodi per livello

- **Numero massimo di nodi complessivi**
  - **Il numero massimo $N_{max}(2,h)$ di nodi in un albero binario di altezza (o profondità, visto che in questo caso coincidono) h è dato da:**

$$N_{max}(2,h) = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$

  - **Ne consegue che l'altezza h e il numero di nodi N sono legati dalla seguente relazione:**

$$h \geq \log_2(N+1) - 1$$

Rel. 03.03.2019

Albero binario completo

- Un albero binario di altezza *h* che contenga esattamente $2^{h+1} - 1$ nodi è detto **completo**.

Rel. 03.03.2019

Albero binario bilanciato

Un albero binario si dice bilanciato (balanced) se, per ogni nodo, *le altezze* del sottoalbero di sinistra e del sottoalbero di destra non differiscono per più di un'unità.

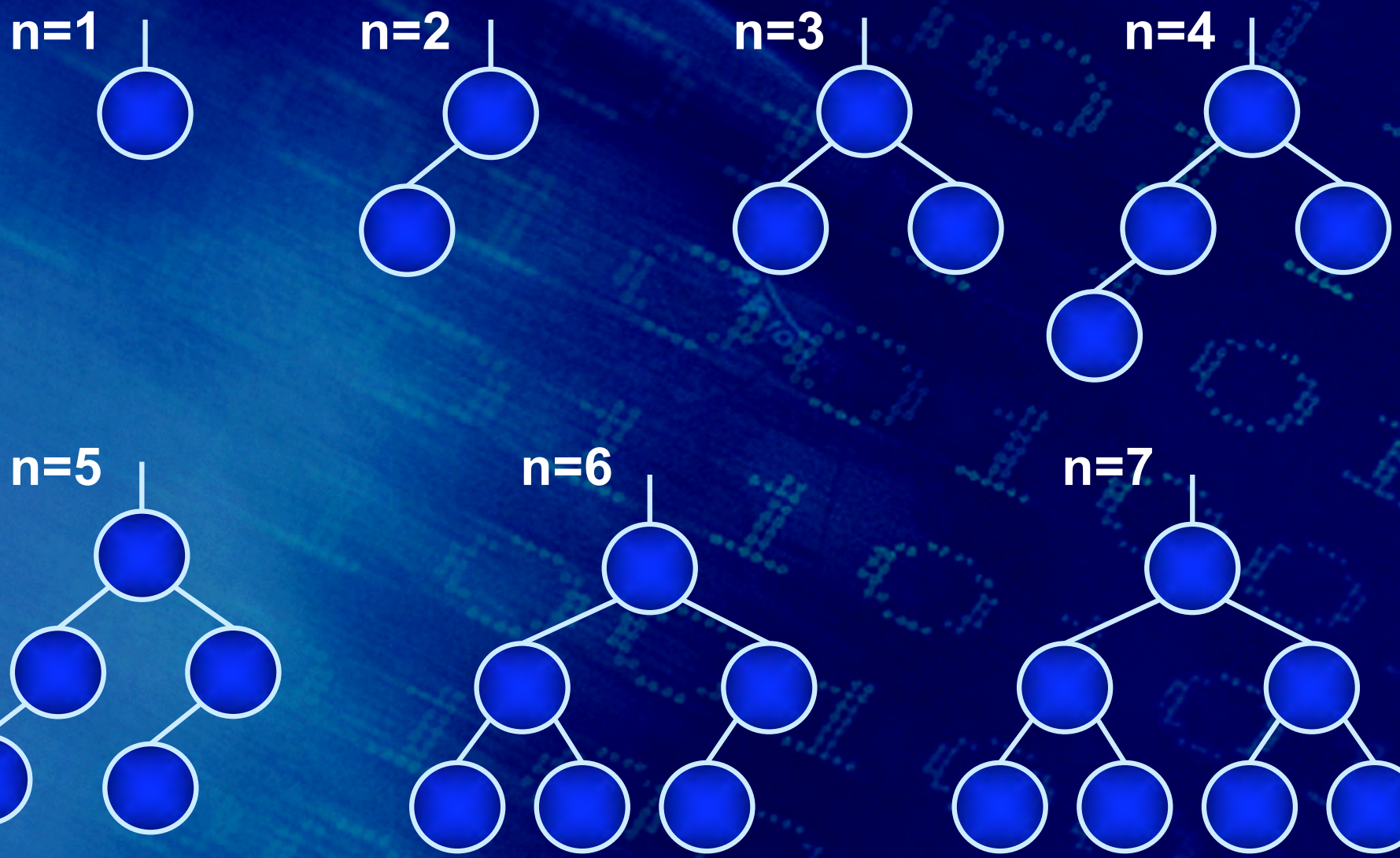
Rel. 03.03.2019

Albero binario perfettamente bilanciato

Un albero binario bilanciato si dice perfettamente bilanciato se, per ogni nodo, il *numero di nodi* del sottoalbero di sinistra e il numero di nodi del sottoalbero di destra non differiscono per più di un'unità.

Rel. 03.03.2019

Albero binario perfettamente bilanciato: Esempi



Rel. 03.03.2019

Rappresentazione degli alberi binari

Sono possibili soluzioni diverse, tutte interpretabili come variazioni di quelle già viste per gli alberi generici.

Esamineremo in particolare le seguenti:

- tramite vettore
- tramite liste multiple.

Rel. 03.03.2019

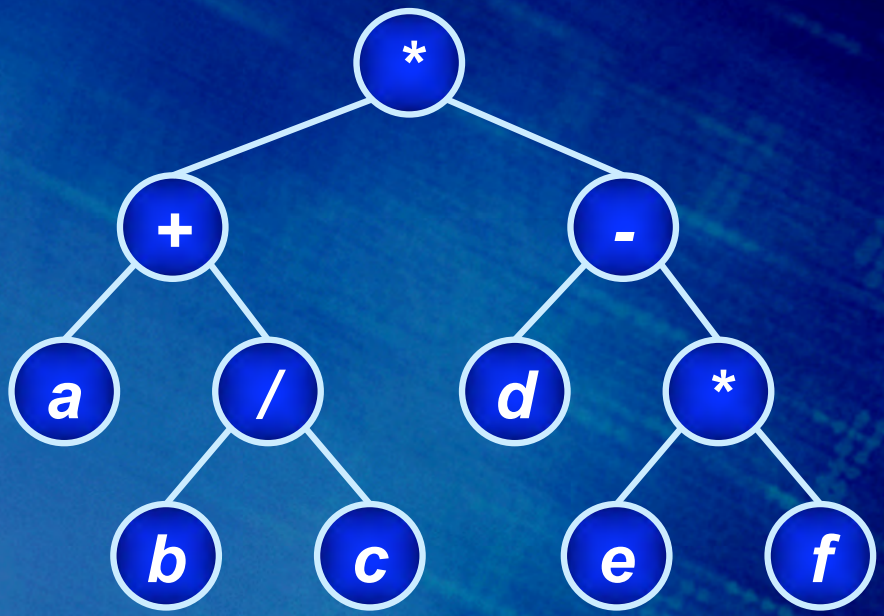
Rappresentazione tramite vettore

- **Se il numero *m* di nodi è noto a priori, è possibile ricorrere a un vettore di *m* record, ciascuno dei quali contiene, oltre all'*i*-esimo nodo, anche gli indirizzi dei suoi due figli.**

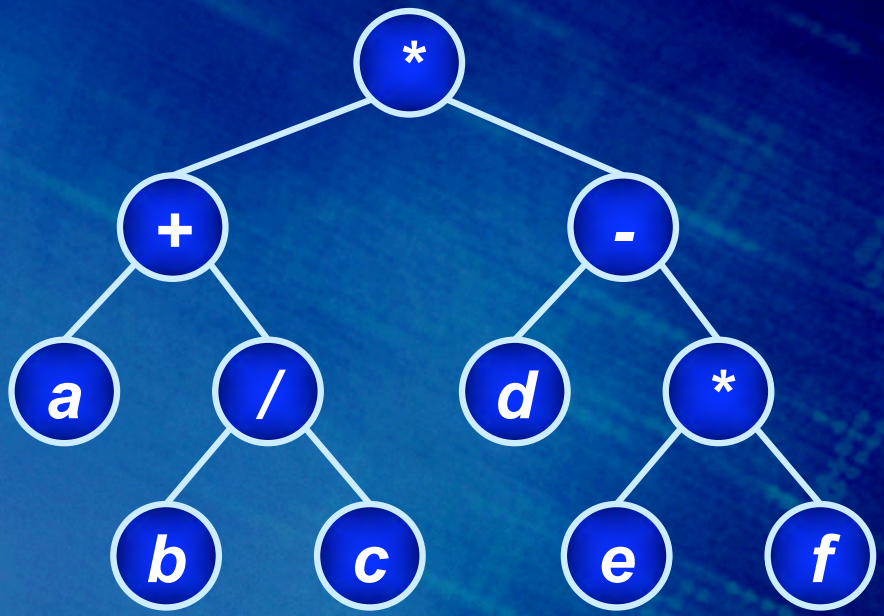
- **Nel caso in cui risulti particolarmente frequente la necessità di risalire al padre di un determinato nodo, è conveniente aggiungere a ciascun record un quarto campo contenente l'indirizzo del padre del nodo in esame.**

Rel. 03.03.2019

Rappresentazione tramite vettore: esempio



Rappresentazione tramite vettore: esempio



1	*	2	3
2	+	6	4
3	-	9	5
4	/	7	8
5	*	10	11
6	a	0	0
7	b	0	0
8	c	0	0
9	d	0	0
10	e	0	0
11	f	0	0

Memorizzazione di alberi binari completi

La memorizzazione di alberi binari completi di *N* nodi può essere effettuata in modo particolarmente agevole, come segue:

- **si numerano i nodi in ordine progressivo, per livello crescente, da sinistra a destra:**

Rel. 03.03.2019

Memorizzazione di alberi binari completi
(cont'd)

- si impiega un vettore di N celle, memorizzando l'*i*-esimo nodo nella cella *i*.

Ne consegue che:

- la radice ha indice 1;
- se un nodo ha indice *k*, il figlio di sinistra ha indice *2k* e quello di destra *2k + 1*
- il nodo *k*-esimo si trova al livello $\lfloor \log_2 k \rfloor$
- il primo nodo al livello *i* è quello di indice 2^i
- il nodo *k*-esimo è il $(k - 2^{\lfloor \log_2 k \rfloor})$-esimo sul suo livello.

Rel. 03.03.2019

Memorizzazione di alberi binari completi Vantaggi e Svantaggi

Risulta particolarmente agevole l'implementazione di alcune operazioni, quali:

- ***parent(i,T)*:**
  - **se $i \neq 1$ ritorna $\lfloor i/2 \rfloor$**
  - **altrimenti *i* è la radice;**
- ***left_child(i,T)*:**
  - **se $2i \leq N$ ritorna $2i$**
  - **altrimenti *i* non ha figli sinistri;**
- ***right_child(i,T)*:**
  - **se $2i + 1 \leq N$ ritorna $2i + 1$**
  - **altrimenti i non ha figli destri.**

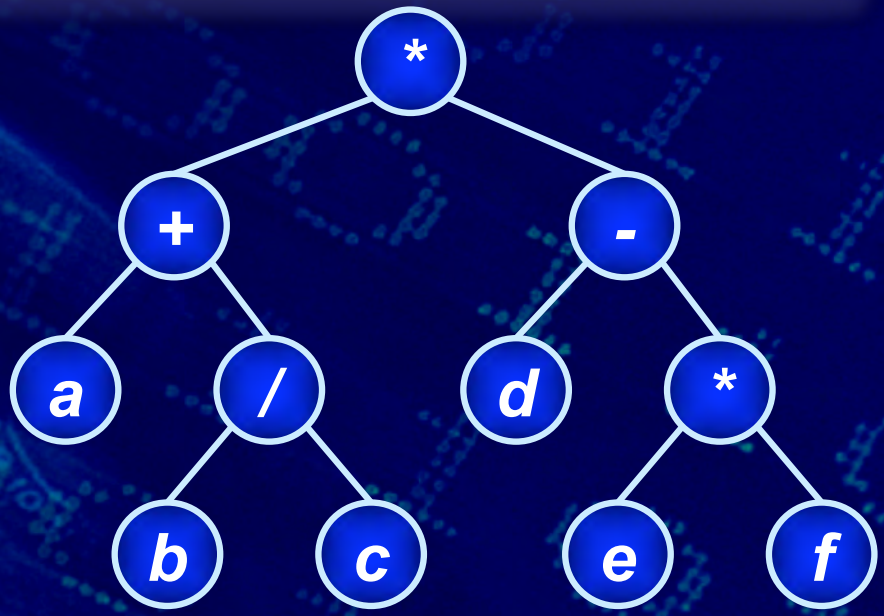
Rel. 03.03.2019

Rappresentazione tramite liste multiple

Ciascun record contiene, oltre all'*i*-esimo nodo, anche i puntatori ai suoi due figli.

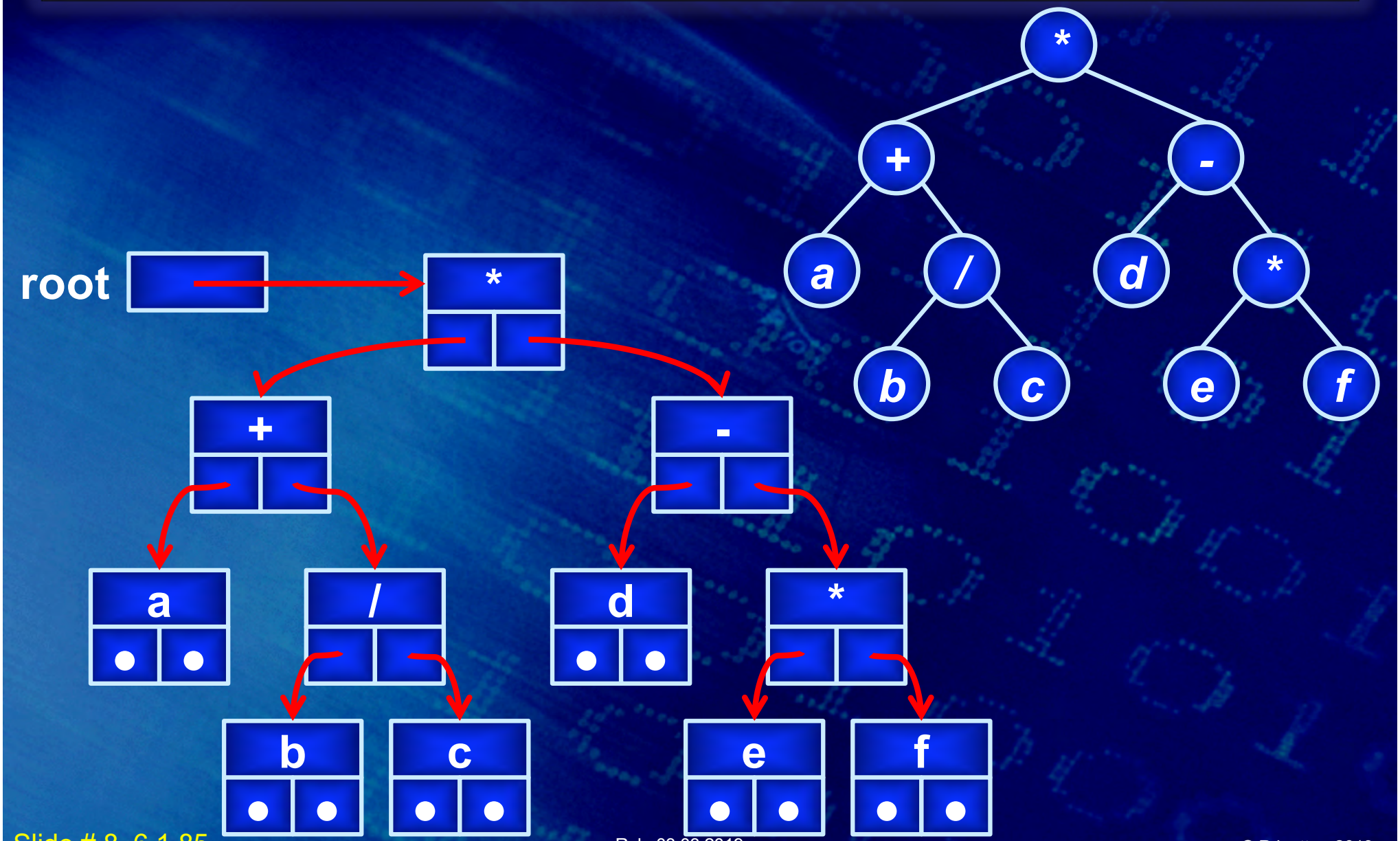
Rel. 03.03.2019

Rappresentazione tramite liste multiple: esempio



Rel. 03.03.2019

Rappresentazione tramite liste multiple: esempio



Rel. 03.03.2019

Outline

- Concetto di albero
- Operazioni sugli alberi
- Rappresentazione degli alberi
- Alberi binari
- Visita di un albero binario

Rel. 03.03.2019

Visita di un albero binario

Per visita (spanning) o attraversamento (traversal) di un albero binario si intende l'esame di tutti i nodi dell'albero, in un certo ordine.

A seconda delle specifiche esigenze vengono solitamente effettuati tre tipi di visita, noti come:

- **visita in ordine anticipato (preorder traversal)**
- **visita in ordine misto (inorder traversal)**
- **visita in ordine differito (postorder traversal).**

Rel. 03.03.2019

Preorder

Per ogni sottoalbero si visita prima la radice, poi il sottoalbero di sinistra, poi quello di destra.

In pratica, il padre prima di tutti i figli.

Preorder

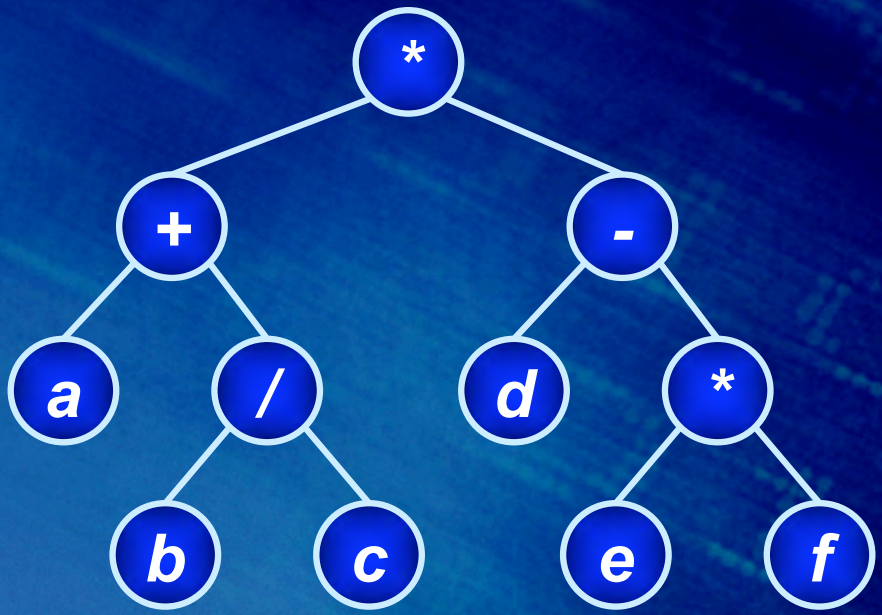
```
Preorder (x)
if x ≠ NIL
    then
        print key[x]
        Preorder ( left[x] )
        Preorder ( right[x] )
```

Hints for Preorder

1. Draw a line along the tree
2. Walk through it counterclockwise
3. Visit a node the 1st time you reach it

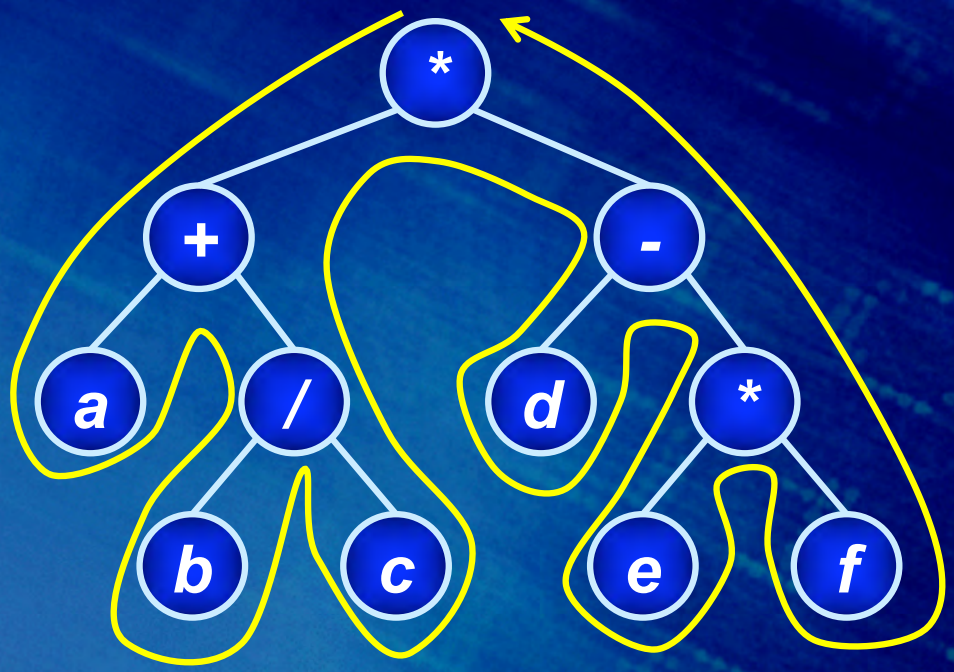
Rel. 03.03.2019

Visita preorder: Esempio



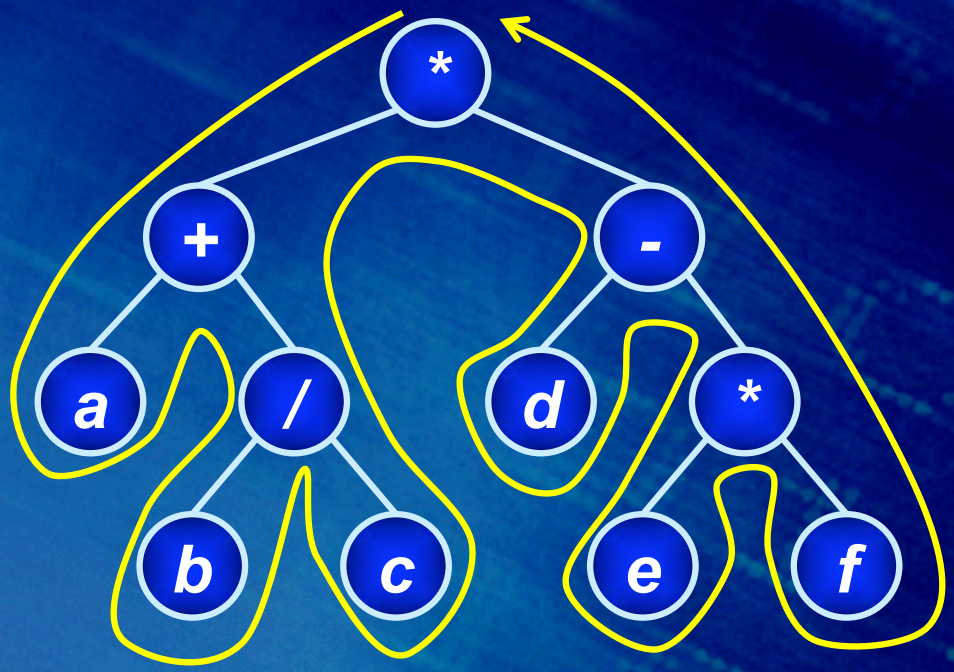
Rel. 03.03.2019

Visita preorder: Esempio



Rel. 03.03.2019

Visita preorder: Esempio



Ordine di visita: * + a / b c - d * e f

Visita inorder

Per ogni sottoalbero si visita prima il sottoalbero di sinistra, poi la radice, poi il sottoalbero di destra.

In pratica, il padre dopo il figlio sinistro, ma prima di quello destro.

Rel. 03.03.2019

Inorder

```
Inorder (x)
    if x ≠ NIL
        then
            Inorder ( left[x] )
            print key[x]
            Inorder ( right[x] )
```

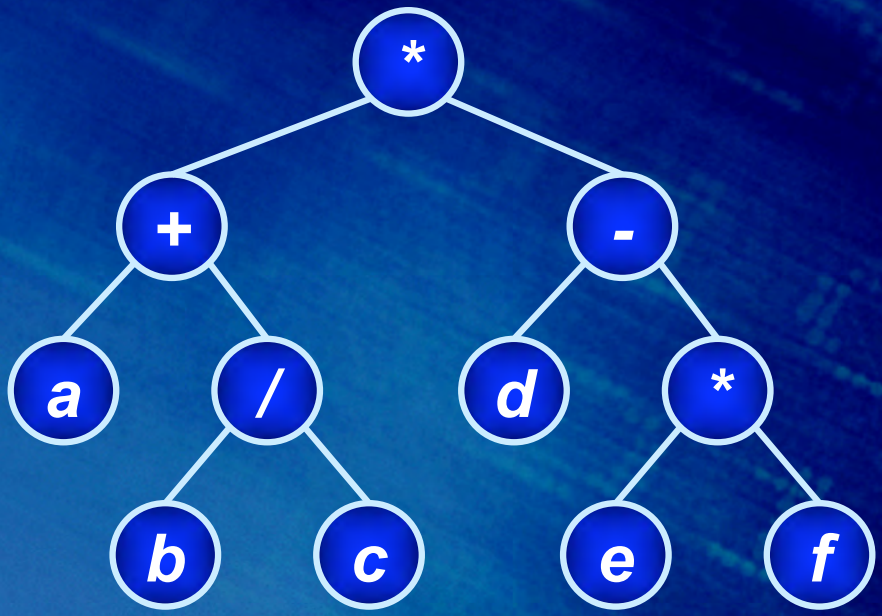

Rel. 03.03.2019

Hints for Inorder

1. Draw a line along the tree
2. Walk through it counterclockwise
3. Visit a node the 2nd time you reach it

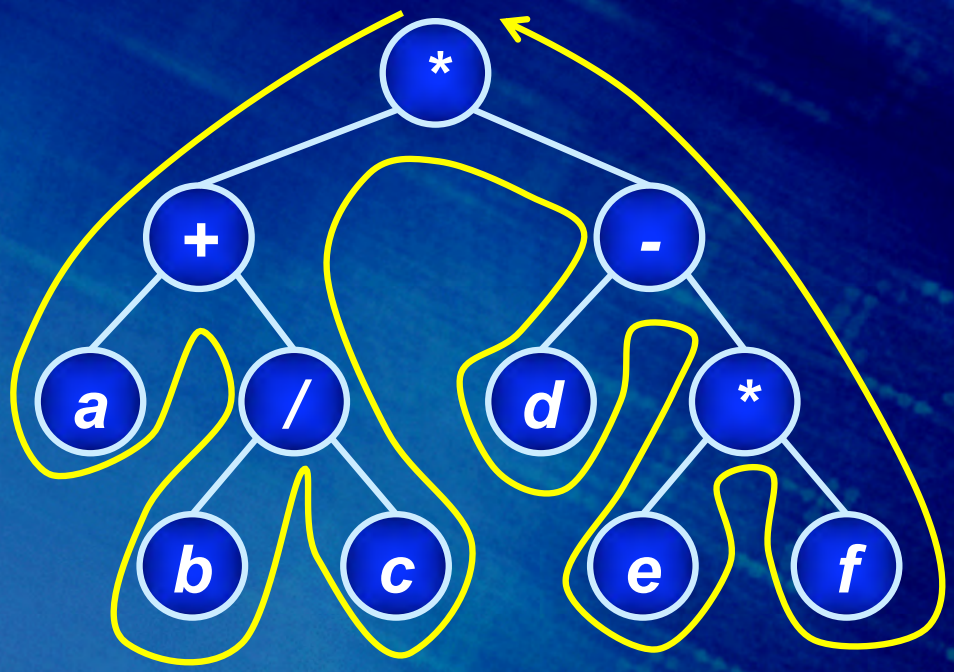
Rel. 03.03.2019

Visita inorder: Esempio 1



Rel. 03.03.2019

Visita inorder: Esempio 1



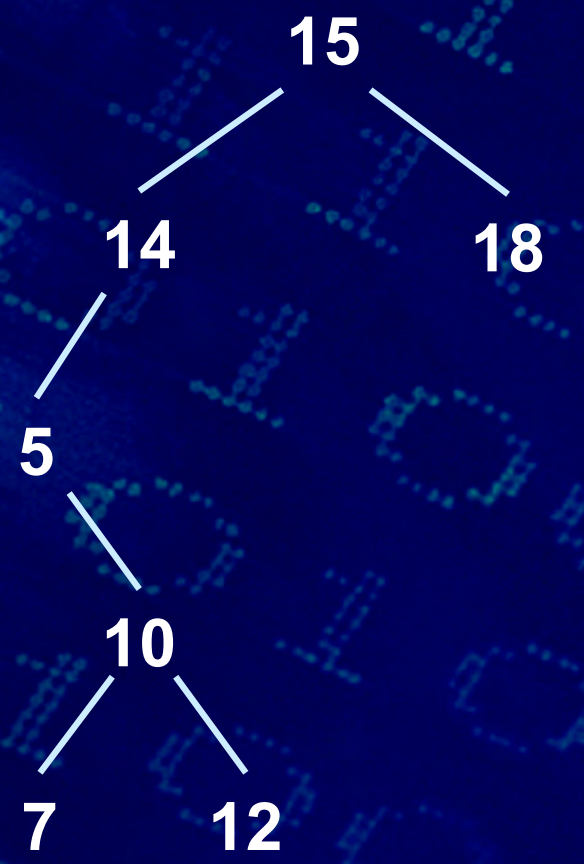
Ordine di visita: a + b / c * d - e * f

Rel. 03.03.2019

Visita inorder: Esempio 2

Rel. 03.03.2019

Visita inorder: Esempio 3



Rel. 03.03.2019

Visita postorder

Per ogni sottoalbero si visita prima il sottoalbero di sinistra, poi quello di destra, poi la radice.

In pratica, il padre dopo tutti i figli.

Rel. 03.03.2019

Postorder

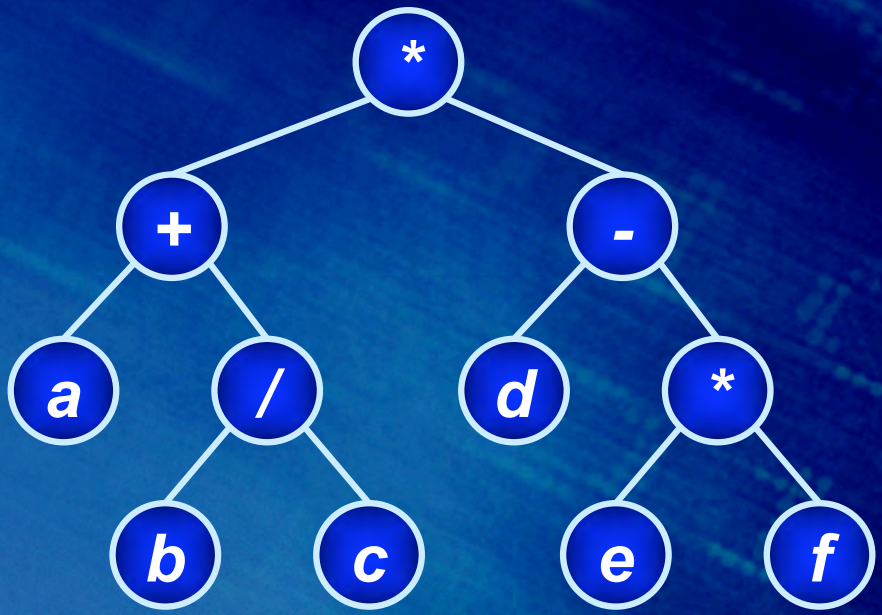
```
Postorder (x)
    if x ≠ NIL
    then
        Postorder ( left[x] )
        Postorder ( right[x] )
        print key[x]
```

Hints for Postorder

1. Draw a line along the tree
2. Walk through it counterclockwise
3. Visit a node the last time you reach it

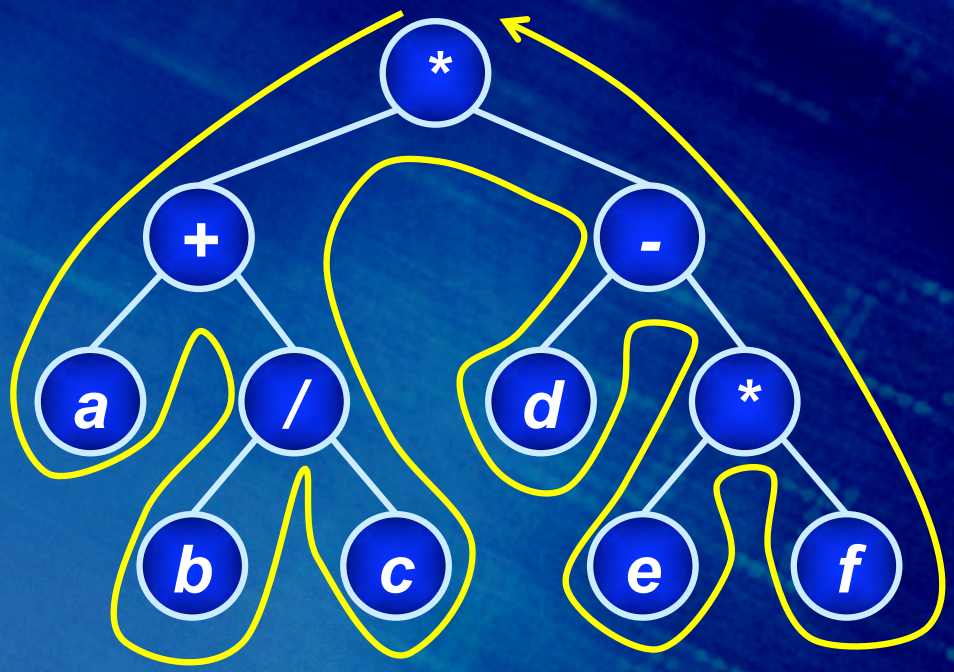
Rel. 03.03.2019

Visita postorder: Esempio



Rel. 03.03.2019

Visita postorder: Esempio



Ordine di visita: a b c / + d e f * - *

Rel. 03.03.2019

Outline

- Concetto di albero
- Operazioni sugli alberi
- Rappresentazione degli alberi
- Alberi binari
- Visita di un albero binario
  - Esempio di applicazione: Espressioni Aritmetiche

Rel. 03.03.2019

Esempio di applicazione: Espressioni Aritmetiche

La rappresentazione avviene secondo i seguenti criteri:

- le foglie rappresentano gli operandi
- i nodi intermedi rappresentano gli operatori
- i sottoalberi di ogni nodo intermedio rappresentano i termini cui si deve applicare l'operatore.

Rel. 03.03.2019

Esempi di applicazione: Espressioni Aritmetiche

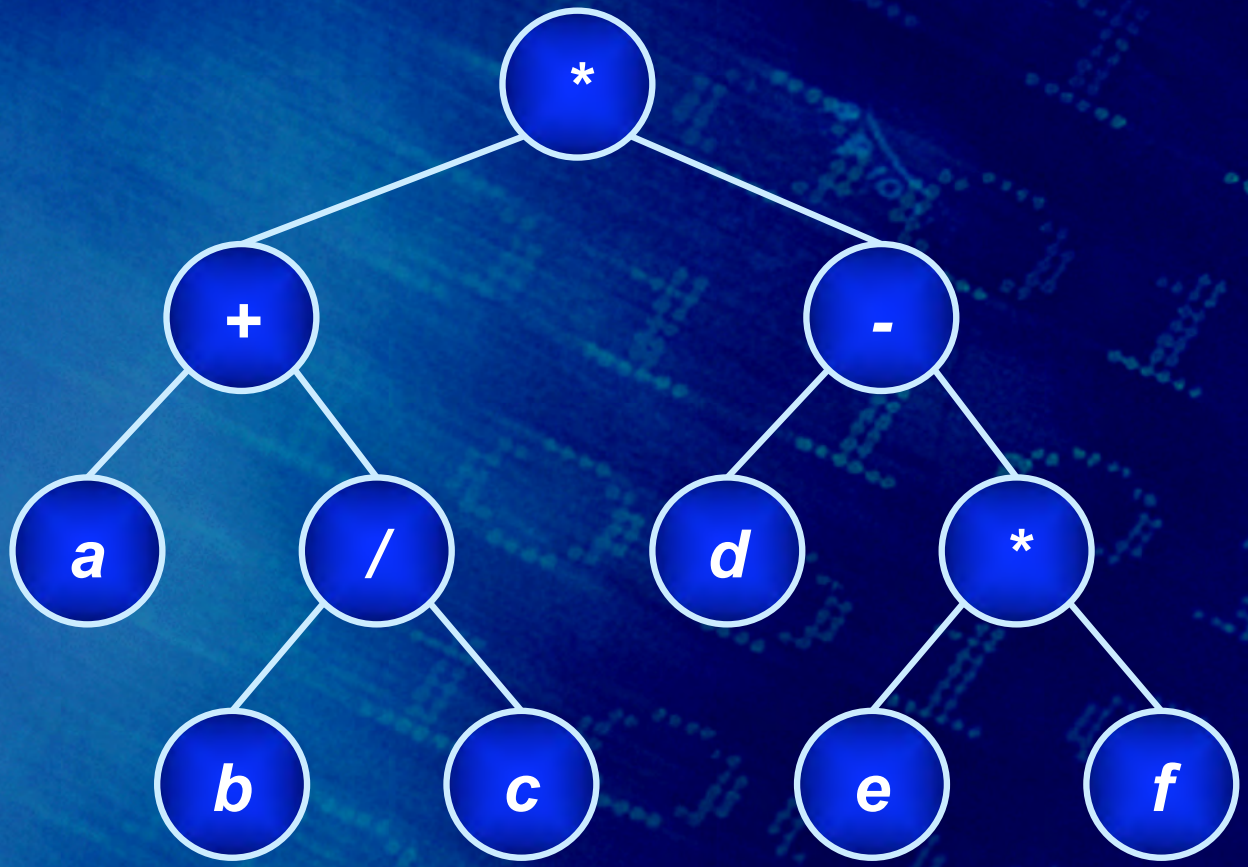
Rappresentazione ad albero dell'espressione:

(a+b/c)*(d-e*f)

Esempi di applicazione: Espressioni Aritmetiche

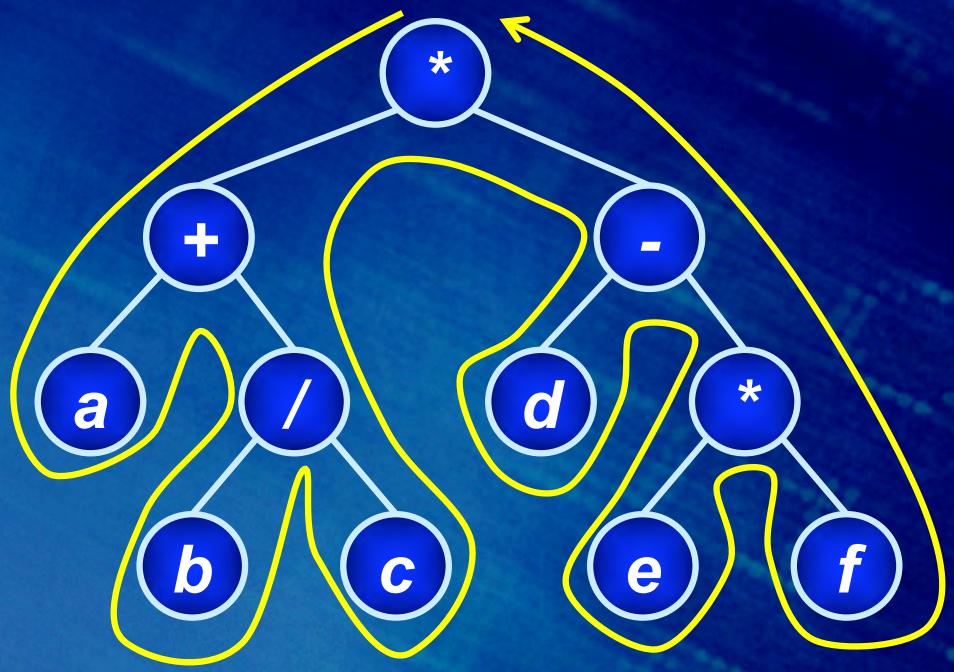
Rappresentazione ad albero dell'espressione:

(a+b/c)*(d-e*f)



Rel. 03.03.2019

Visita preorder



Ordine di visita: * + a / b c - d * e f

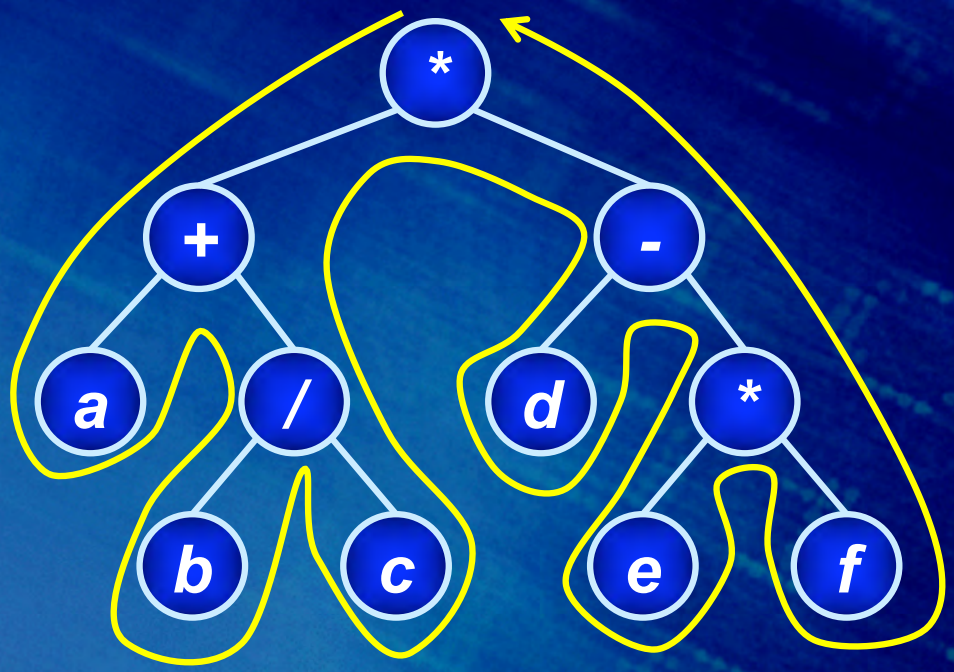
Rel. 03.03.2019

Visita preorder

Una visita preorder di un albero in cui sia memorizzata un'espressione aritmetica fornisce una rappresentazione della stessa nella cosiddetta *notazione prefissa*.

Rel. 03.03.2019

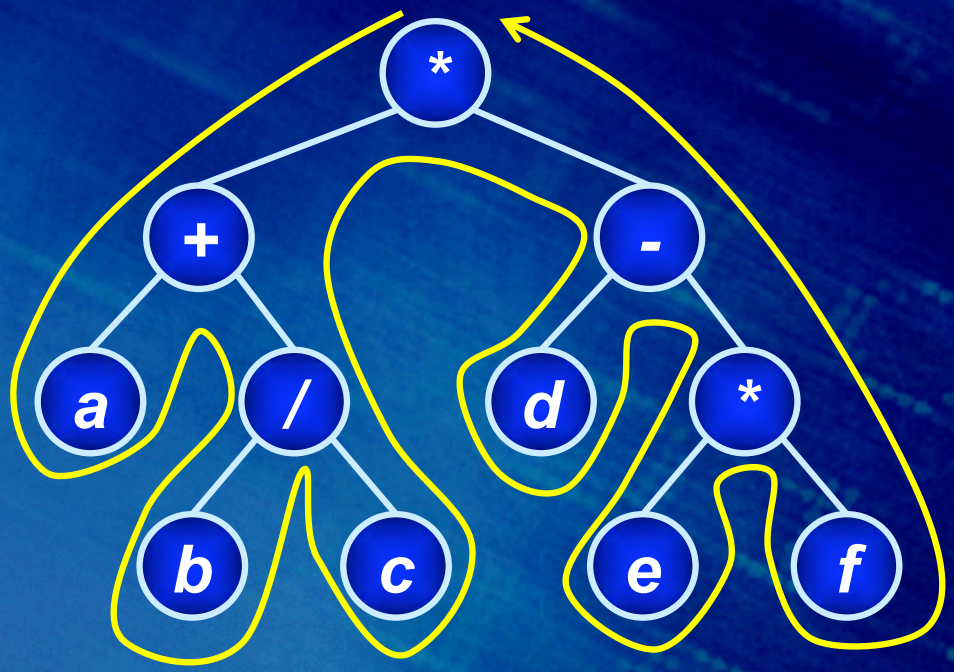
Visita inorder



Ordine di visita: a + b / c * d - e * f

Rel. 03.03.2019

Visita postorder



Ordine di visita: a b c / + d e f * - *

Rel. 03.03.2019

Visita postorder

Una visita postorder di un albero in cui sia memorizzata un'espressione aritmetica fornisce una rappresentazione della stessa nella cosiddetta *notazione postfissa*, o *notazione polacca inversa RPN.*

Reverse Polish notation (RPN)

- ***Reverse Polish notation (RPN)*** **is a mathematical notation in which every operator follows all of its operands, in contrast to Polish notation, which puts the operator in the prefix position.**
- **It is also known as postfix notation and is parenthesis-free as long as operator arities (i.e., the numbers of their operators) are fixed.**
- **The description "Polish" refers to the nationality of logician Jan Łukasiewicz, who invented the (prefix) Polish notation in the 1920s.**

Rel. 03.03.2019

References

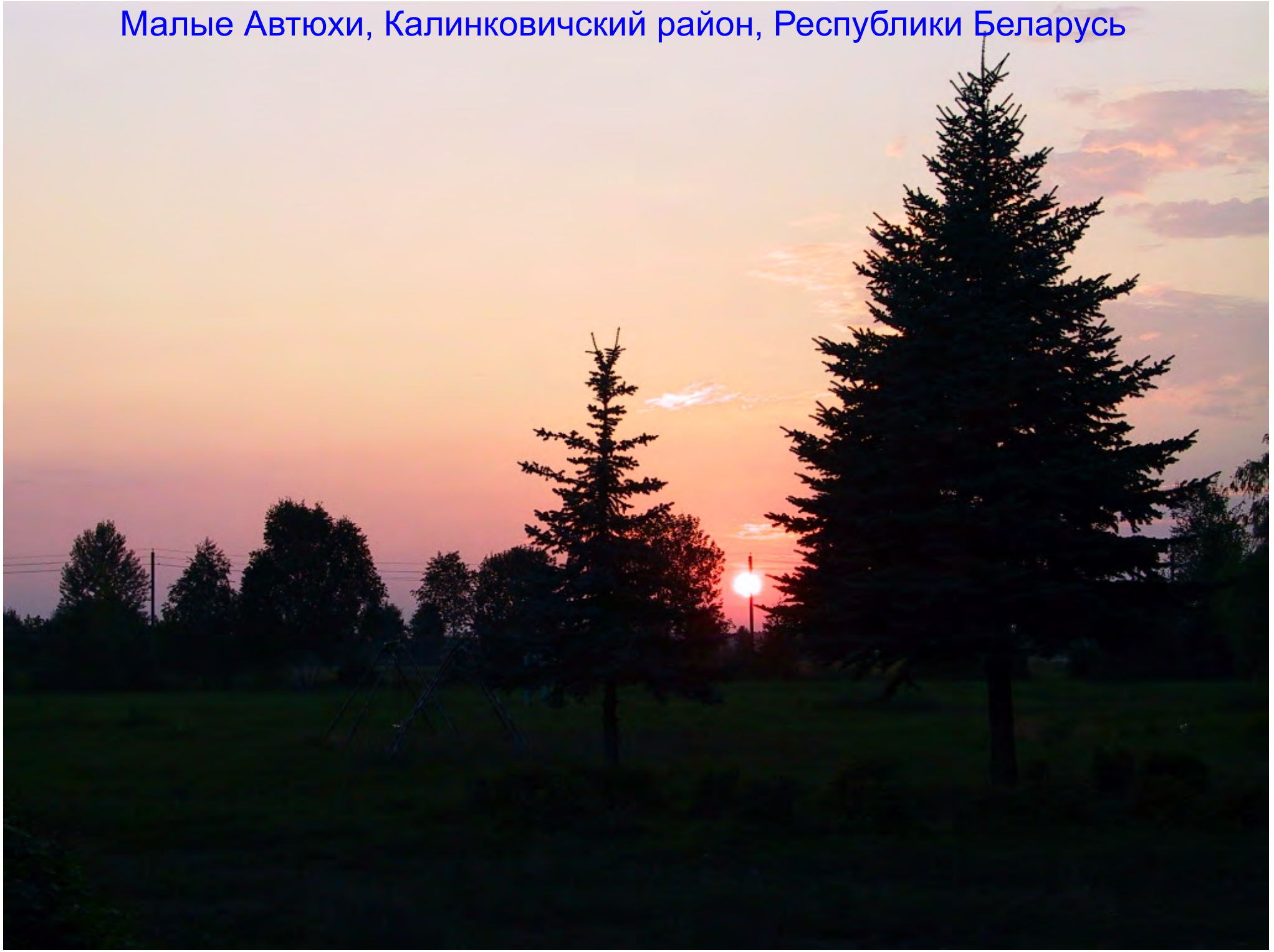
- A.V. Aho, J.E. Hopcroft, J.D. Ullman:
  "Data Structures and Algorithms,"
  Addison Wesley, Reading MA (USA), 1983
  pp. 75-106
- G.H. Gonnet:
  "Handbook of Algorithms and Data Structures,"
  Addison Wesley, Reading MA (USA), 1984, pp. 69-117
- J. Esakow. T. Weiss
  "Data structure: an advanced approach using C,"
  Prentice Hall, Englewood Cliffs NJ (USA), 1982, pp. 38-59

References

- E. Horowitz, S. Sahni:
  "Fundamentals of Computer Algorithms,"
  Pittman, London (UK), 1978
  pp. 203-271
- R. Sedgewick:
  "Algorithms in C,"
  Addison Wesley, Reading MA (USA), 1990
  pp. 35-50
- C.J. Van Wyk:
  "Data Structures and C Programs," Addison Wesley, Reading MA (USA), 1988
  pp 159-176

References

- M.A. Weiss:
  "Data Structures and Algorithm Analysis,"
  The Benjamin/Cummings Publishing Company,
  Redwood City, CA (USA), 1992, pp. 87-98
- R.J. Wilson:
  "Introduzione alla teoria dei grafi,"
  Cremonese, Roma 1978, pp. 57-76
- N. Wirth:
  "Algorithms + Data Structures = Programs,"
  Prentice Hall, Englewood Cliffs NJ (USA), 1976
  pp. 169-263


Малые Автюхи, Калинковичский район, Республики Беларусь