Lecture
8_8.3

Search 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. 07.04.2019

Goal

– **This lecture aims at presenting an ADT suitable for searching: *Search Tree,* the related operations, and the visiting techniques.**

Rel. 07.04.2019

Prerequisites

- **Lectures:**
  - **8_2.1 Introduction to ADT**
  - **8_6.1 Trees**

Rel. 07.04.2019

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. 07.04.2019

Outline

- Alberi binari di ricerca
- Altri tipi di Alberi:
  - HBT
  - B tree
  - Varianti di B tree

Outline

- Alberi binari di ricerca
- Altri tipi di Alberi:
  - HBT
  - B tree
  - Varianti di B tree

Rel. 07.04.2019

Alberi binari di ricerca

- Gli **alberi binari di ricerca** o **Binary Search Tree (BST)** sono alberi binari in cui in ciascun nodo è memorizzato un elemento *x* dell'insieme.
- In particolare, per ogni nodo:
  - tutti gli elementi memorizzati nel suo sottoalbero sinistro hanno una chiave < *x.key*
  - tutti gli elementi memorizzati nel suo sottoalbero destro hanno una chiave > *x.key*

Rel. 07.04.2019

Alberi binari di ricerca

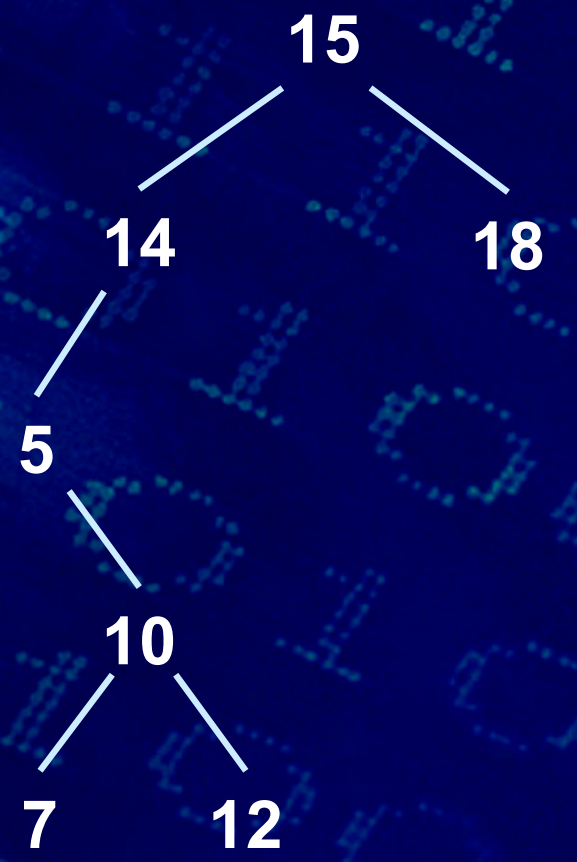
- Gli **alberi bi** **Tree (BST)** sono **odo è memorizzat**
- **In particolar**
  - **tutti gli el** **toalbero sinistro ha**
  - **tutti gli ele** **toalbero destro han**

Ne consegue che una visita inorder fornisce tutti gli elementi ordinati in modo crescente

Rel. 07.04.2019

Alberi binari di ricerca (cont'd)

- **Per un determinato insieme esistano diversi possibili BST**



Il BST come ADT

Le operazioni più significative che si possono definire sui BST sono le stesse già introdotte a proposito dei dizionari.

Ricerca

La caratteristica costruttiva di un BST facilita le operazioni di ricerca: per verificare se *x* appartiene all'insieme, si compara *x.key* con la chiave dell'elemento *r* memorizzato nella radice.

Rel. 07.04.2019

Ricerca *(cont'd)*

- Si può procedere come segue:
  - se *x.key* < *r.key* : *x* può solo essere trovato nel sottoalbero di sinistra della radice
  - se *x.key* = *r.key* : *x* coincide con la radice
  - se *x.key* > *r.key* : *x* può solo essere trovato nel sottoalbero di destra della radice
  - in modo iterativo, si ripete il test sul sottoalbero di destra o di sinistra

Rel. 07.04.2019

Ricerca *(cont'd)*

- Si può proce
  - se *x.key* <
  - se *x.key* =
  - se *x.key* >
  - in modo it
    di destra o

ato nel sottoalbe

e

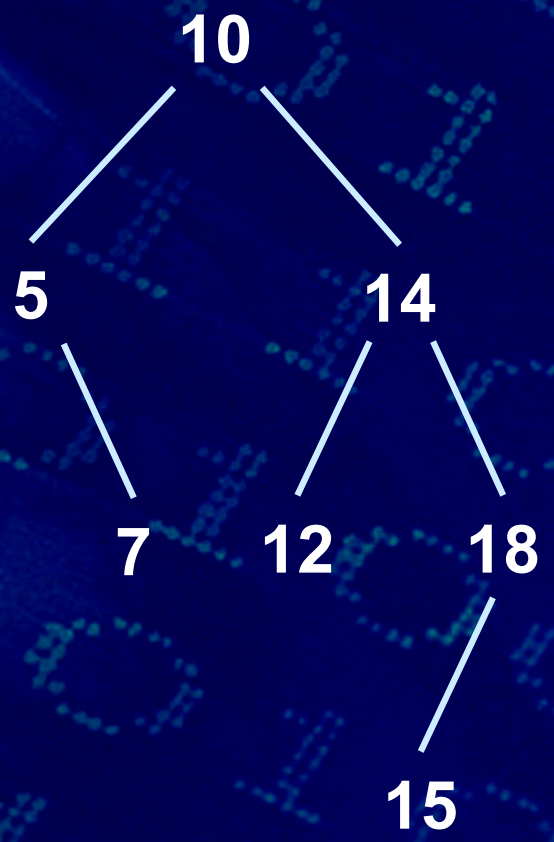
ato nel sottoalber

oalbero

Ne consegue che la ricerca di un dato comporta al più un numero di confronti pari all'altezza dell'albero

Ricerca *(cont'd)*

- **Ricerca del *Minimo*:**
  - **È sempre nell'elemento più a sinistra del BST**
- **Ricerca del Massimo:**
  - **È sempre nell'elemento più a destra del BST**



Rel. 07.04.2019

Ricerca *(cont'd)*

Una possibile implementazione in C può essere la seguente:

```
search(r,key){
   if (r == null)
      return(ERROR)
   if (key < r.key)
      return(search(r.left,key))
   if (key > r.key)
      return(search(r.right,key))
   return(r)
}
```


Rel. 07.04.2019

Inserimento

- Presuppone una ricerca `search()`
- Se la chiave da inserire non esiste ancora, si crea un nuovo nodo e lo si aggancia come figlio dell'ultimo nodo visitato dalla `search()`.

Rel. 07.04.2019

Inserimento : Esempio

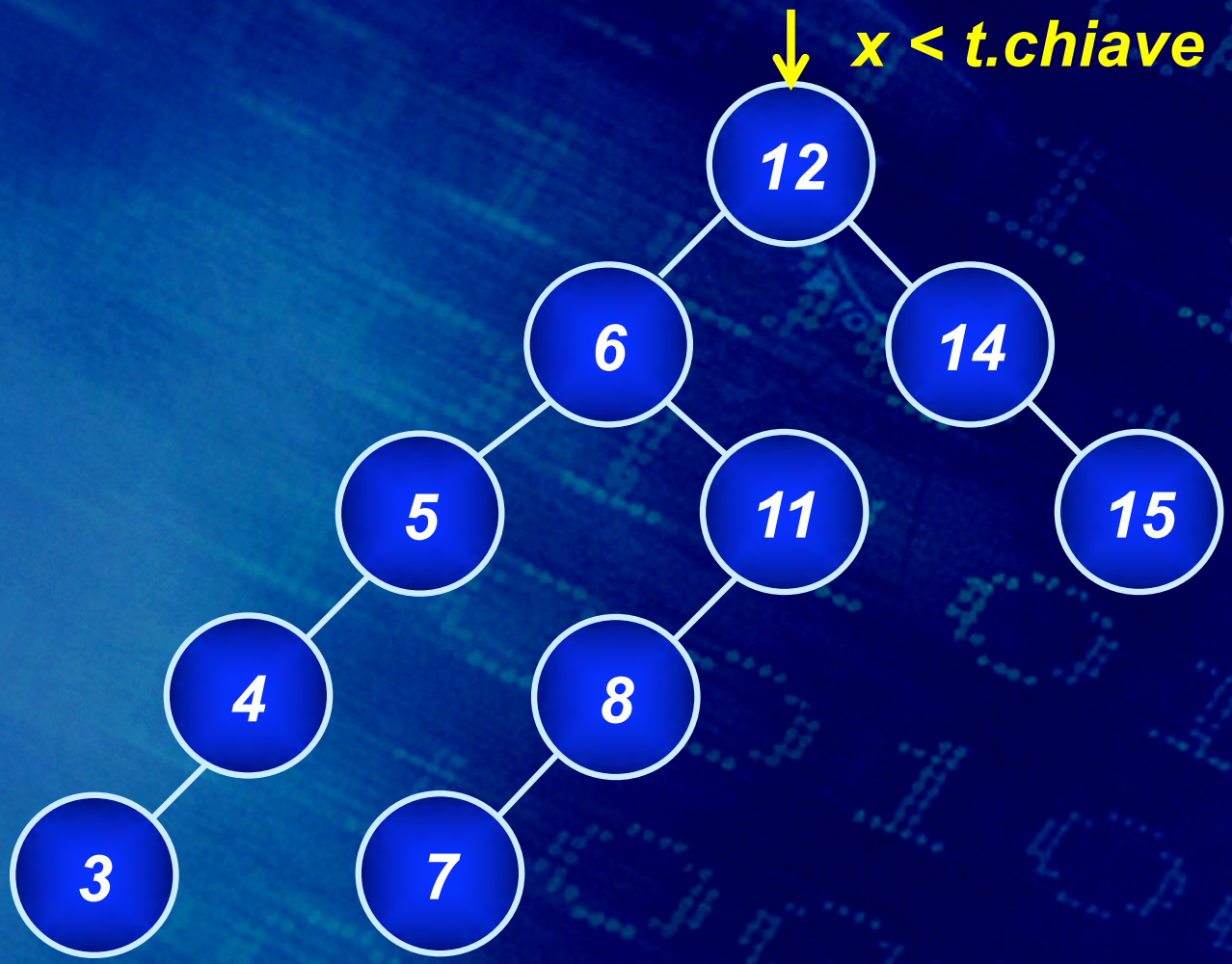
- Inserimento dell'elemento *x* con chiave 10

Rel. 07.04.2019

Inserimento : Esempio

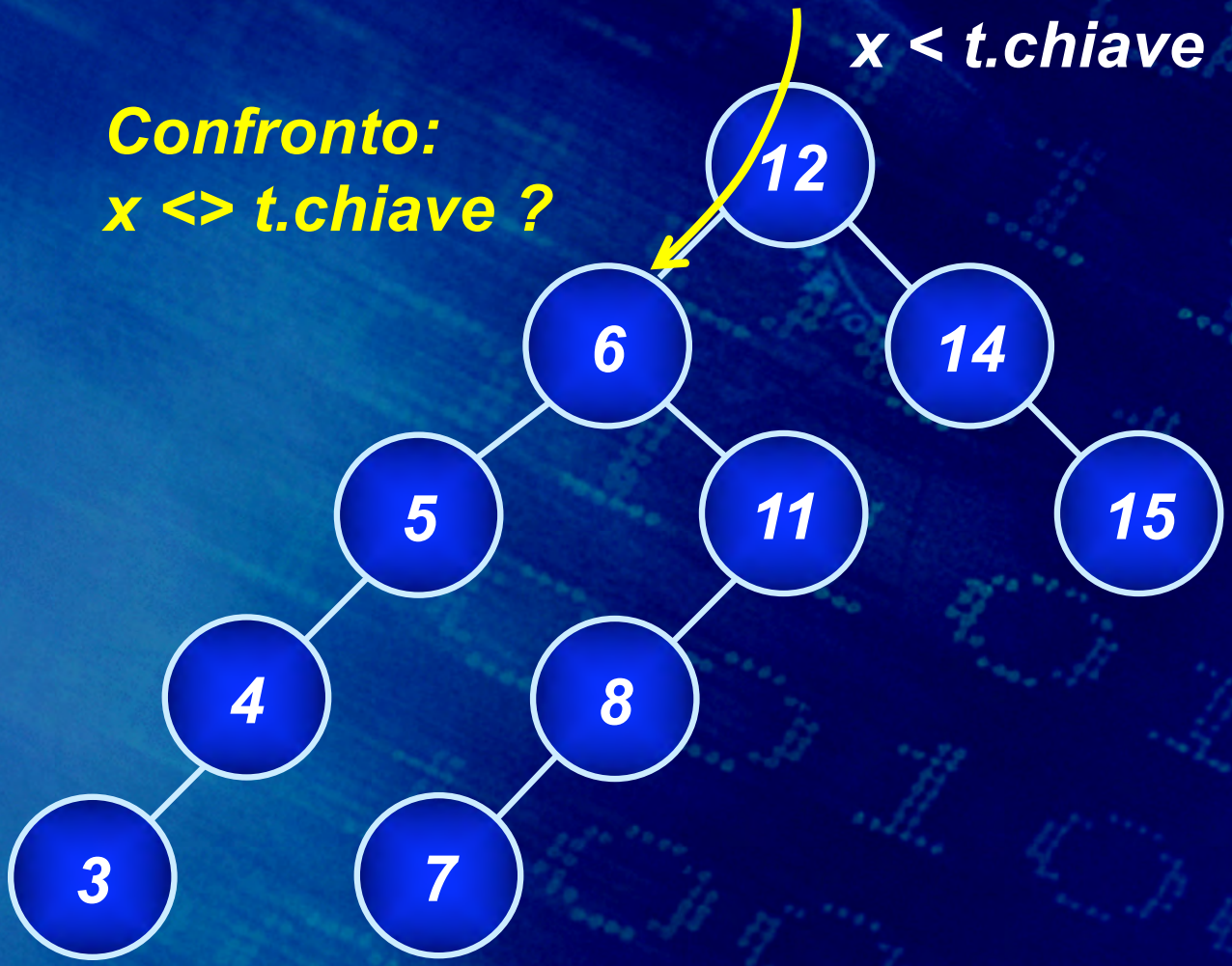
Rel. 07.04.2019

Inserimento : Esempio



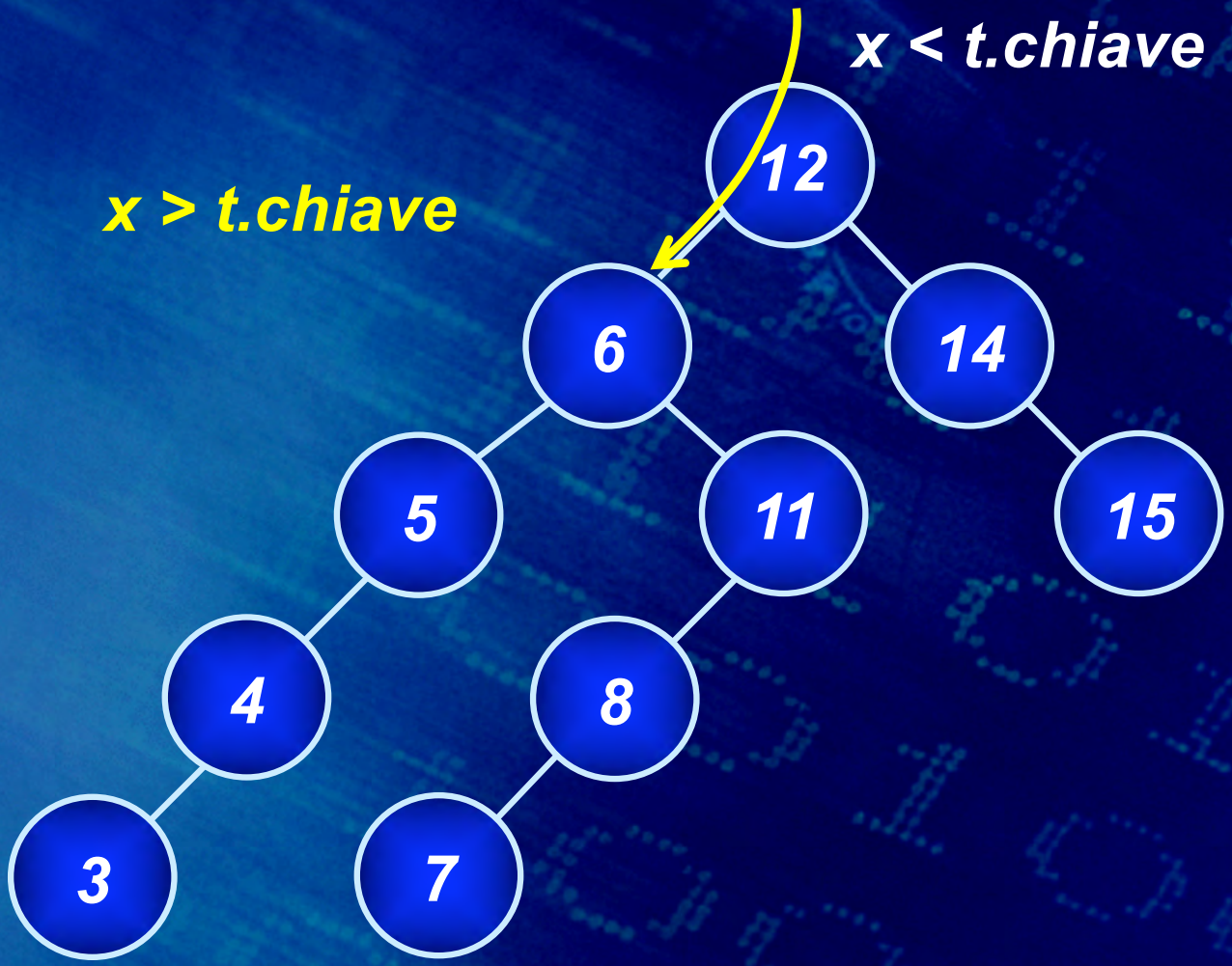
Rel. 07.04.2019

Inserimento : Esempio



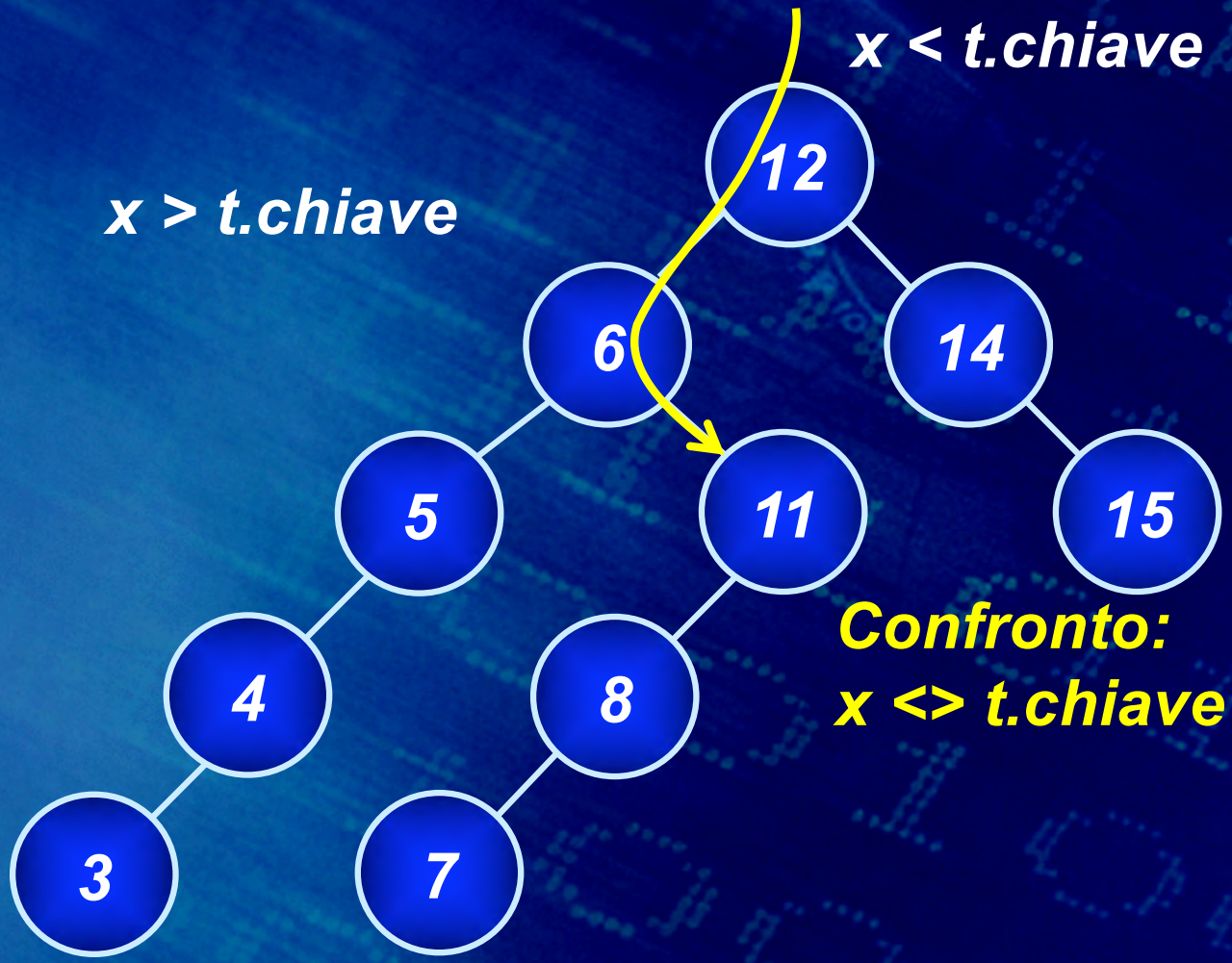
Rel. 07.04.2019

Inserimento : Esempio



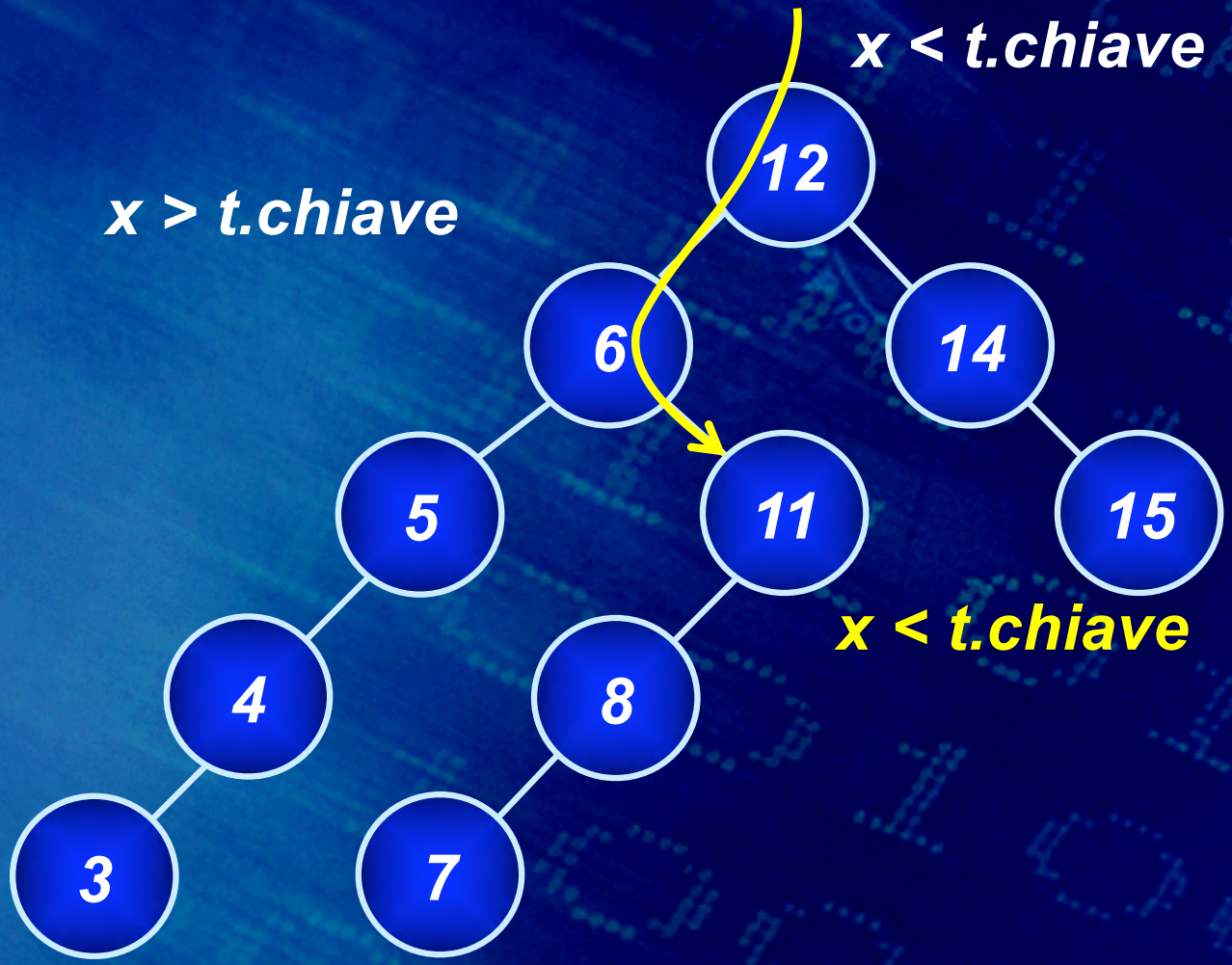
Rel. 07.04.2019

Inserimento : Esempio



Rel. 07.04.2019

Inserimento : Esempio

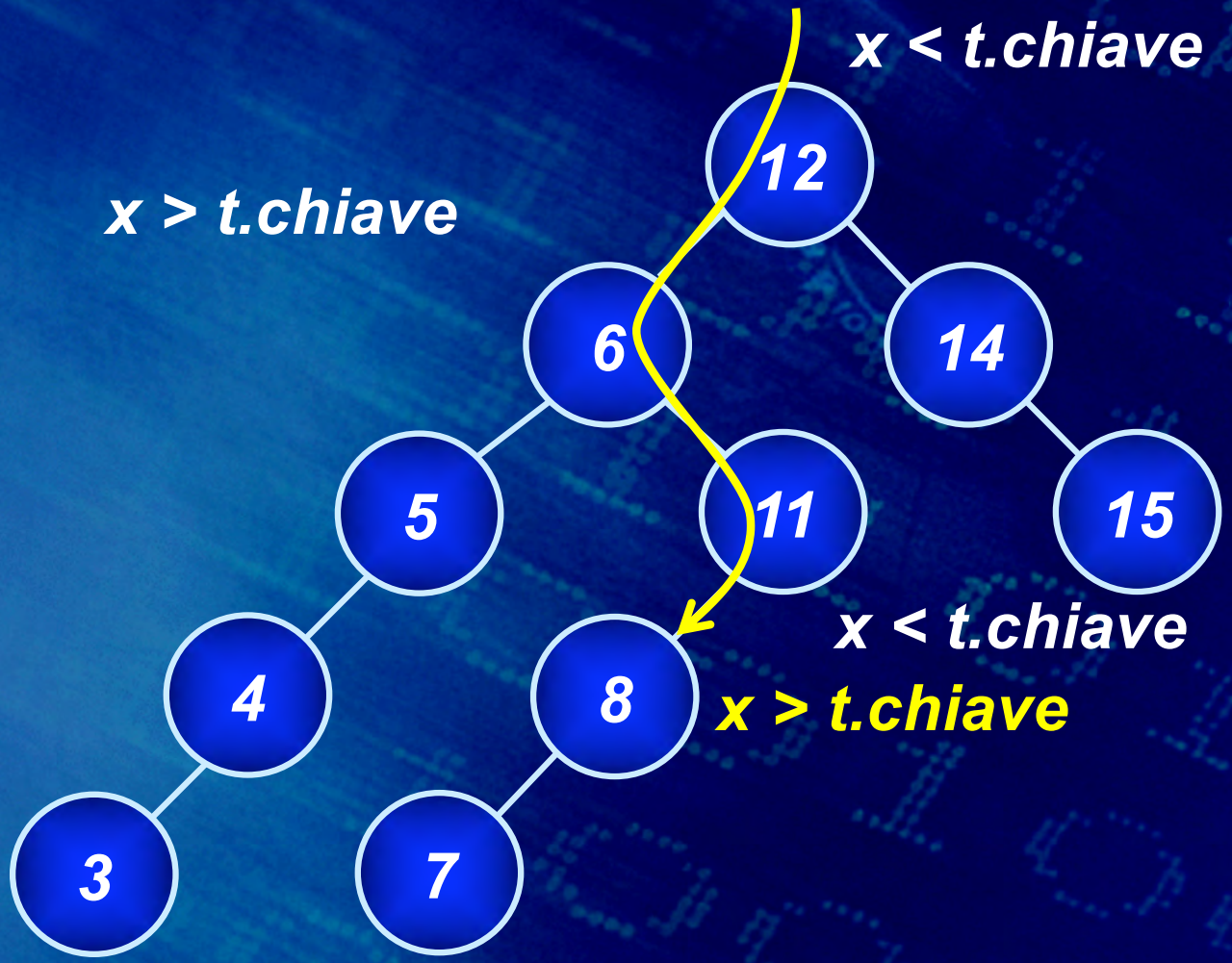


Rel. 07.04.2019

Inserimento : Esempio

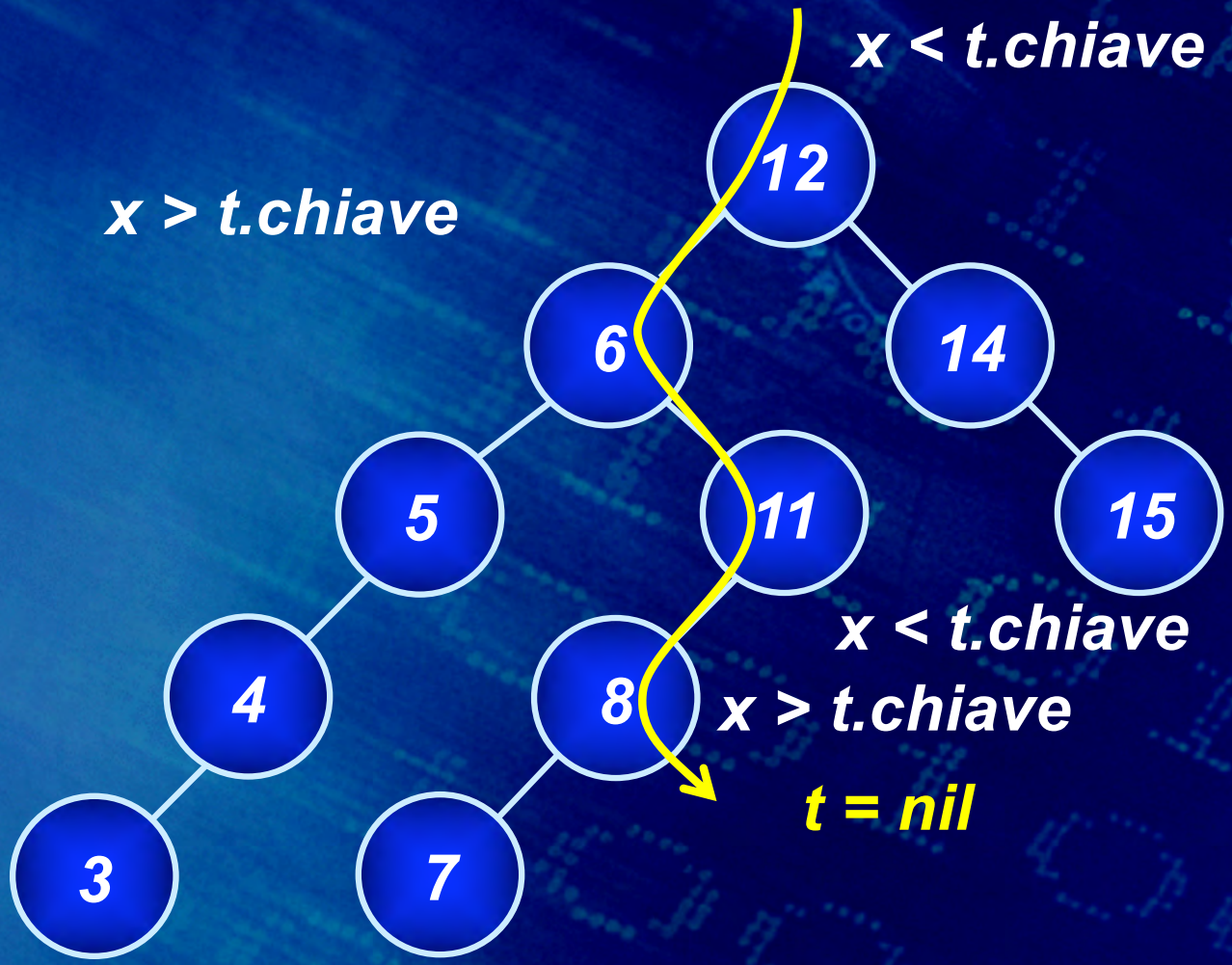
Rel. 07.04.2019

Inserimento : Esempio



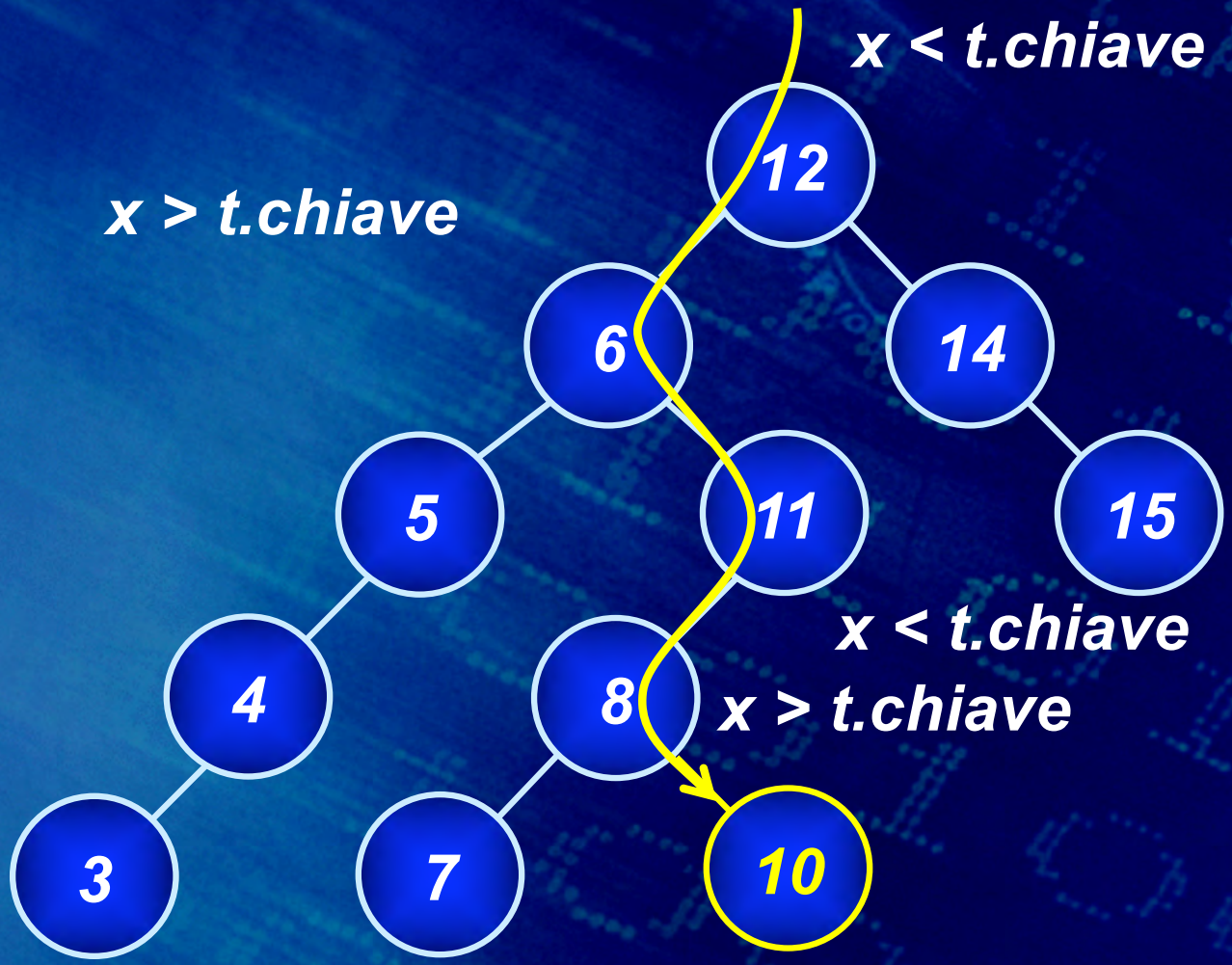
Rel. 07.04.2019

Inserimento : Esempio



Rel. 07.04.2019

Inserimento : Esempio



Rel. 07.04.2019

Sbilanciamento

- L'inserimento casuale in un BST può sbilanciare l'albero.
- In particolare, se gli inserimenti avvengono in ordine (crescente o decrescente) di chiave, l'albero si trasforma in una lista.

Rel. 07.04.2019

Sbilanciamento: Esempio

- **Ordine d'inserimento: 7 6 5 4 3 2 1**

Rel. 07.04.2019

Sbilanciamento: Esempio

- **Ordine d'inserimento: 7 6 5 4 3 2 1**

Ne consegue che record diversi possono avere tempi di accesso molto diversi

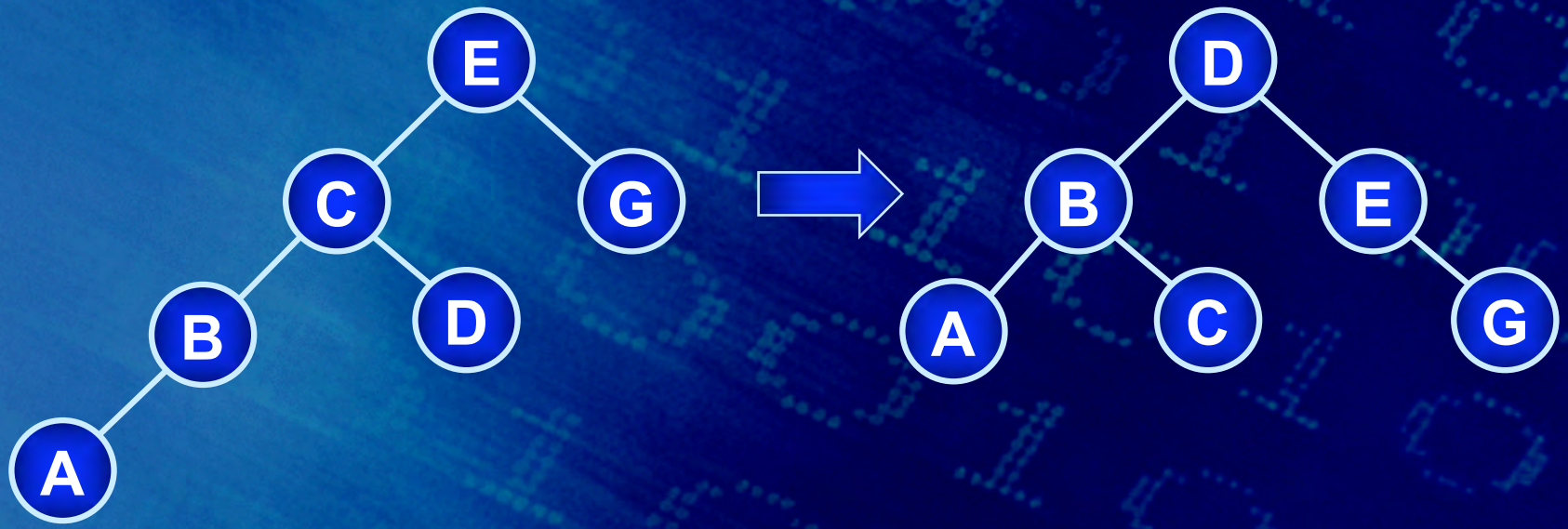
Sbilanciamento: Complessità

- Nel caso peggiore di un albero di *n* record, completamente sbilanciato, coincidente cioè con una lista lineare, il numero di confronti richiesti per ogni ricerca è *O(n)*.
- Tale complessità può essere ridotta solo facendo sì che l'albero sia sempre **completamente bilanciato**. In tal caso, infatti, si esegue un numero di confronti al più pari all'altezza dell'albero, vale a dire: $O(\log_2 (n+1))$.

Rel. 07.04.2019

Costo del bilanciamento

Mantenere il bilanciamento di un albero generico può essere molto costoso, in quanto lo spostamento di un elemento può implicare lo spostamento di tutti gli altri.



Rel. 07.04.2019

Costo del bilanciamento *(cont'd)*

- **Occorre garantire che il risparmio ottenibile nelle operazioni di ricerca grazie alla garanzia di bilanciamento sia superiore all'overhead necessario per garantire il bilanciamento stesso.**
- **A livello pratico, è conveniente riorganizzare l'albero dopo ogni inserzione per garantire il bilanciamento solo qualora la frequenza delle inserzioni sia molto minore della frequenza delle ricerche.**

Rel. 07.04.2019

Costo del bilanciamento *(cont'd)*

- **Si può comunque dimostrare che, nell'ipotesi che tutti i valori di chiave abbiano la stessa probabilità, per grandi valori di *n*, la complessità media dell'operazione di ricerca in BST in cui non viene fatto il ri-bilanciamento, è $O(1.4 \cdot \log_2 n)$.**

Rel. 07.04.2019

Cancellazione

Si distinguono tre casi:

- **l'elemento da cancellare non ha figli**
- **l'elemento da cancellare ha un figlio**
- **l'elemento da cancellare ha due figli.**

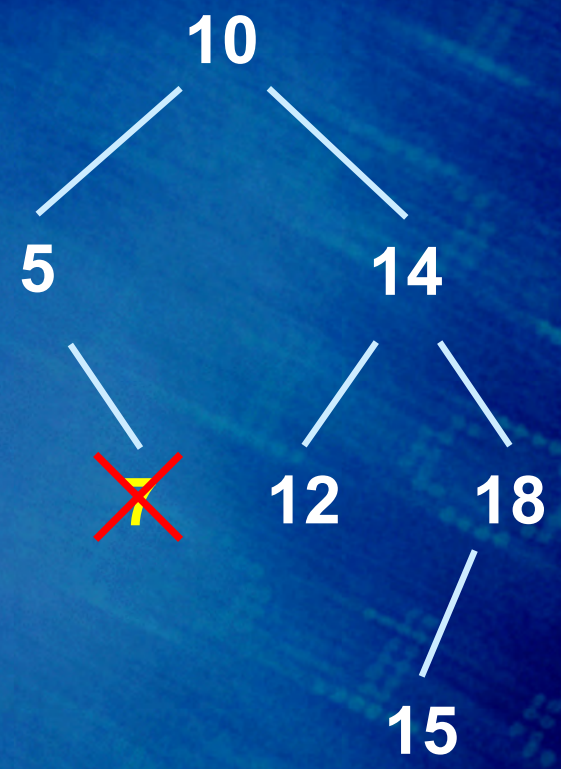
Rel. 07.04.2019

Cancellazione *(cont'd)*

- **l'elemento da cancellare non ha figli:**
  - **nessun problema: lo si cancella e basta.**

Cancellazione *(cont'd)*

- **l'elemento da cancellare non ha figli:**
  - **nessun problema: lo si cancella e basta.**



Cancellazione *(cont'd)*

- **l'elemento da cancellare non ha figli:**
  - **nessun problema: lo si cancella e basta.**

Rel. 07.04.2019

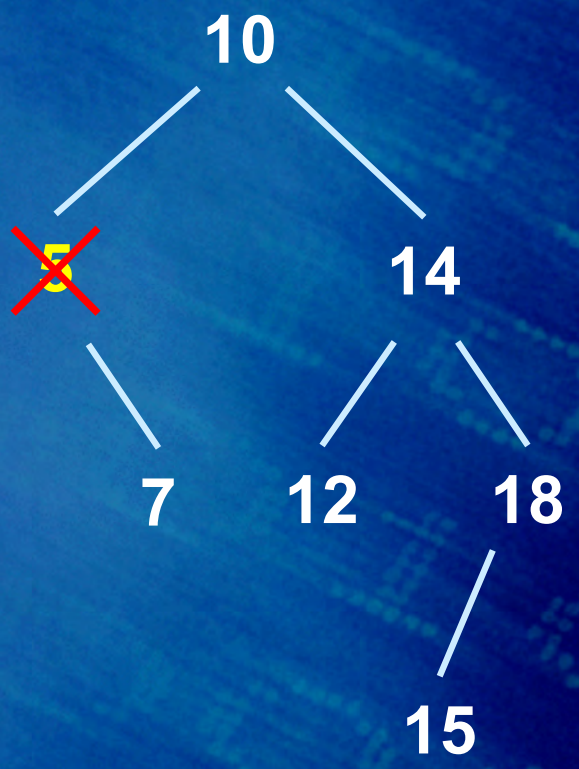
Cancellazione *(cont'd)*

- **l'elemento da cancellare ha un figlio:**
  - **si elimina l'elemento e lo si sostituisce con l'unico figlio. In pratica significa memorizzare nel padre il puntatore all'unico figlio.**

Rel. 07.04.2019

Cancellazione *(cont'd)*

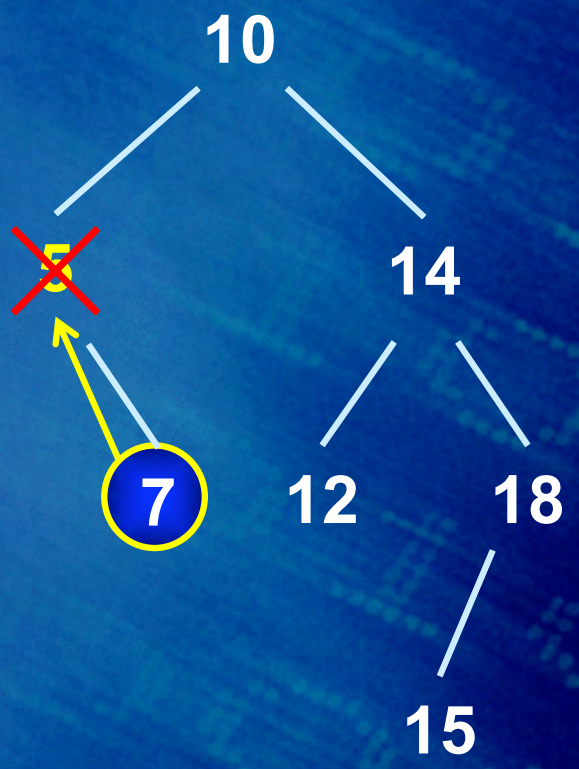
- **l'elemento da cancellare ha un figlio:**
  - **si elimina l'elemento e lo si sostituisce con l'unico figlio. In pratica significa memorizzare nel padre il puntatore all'unico figlio.**



Rel. 07.04.2019

Cancellazione *(cont'd)*

- **l'elemento da cancellare ha un figlio:**
  - **si elimina l'elemento e lo si sostituisce con l'unico figlio. In pratica significa memorizzare nel padre il puntatore all'unico figlio.**



Rel. 07.04.2019

Cancellazione *(cont'd)*

- **l'elemento da cancellare ha un figlio:**
  - **si elimina l'elemento e lo si sostituisce con l'unico figlio. In pratica significa memorizzare nel padre il puntatore all'unico figlio.**

Rel. 07.04.2019

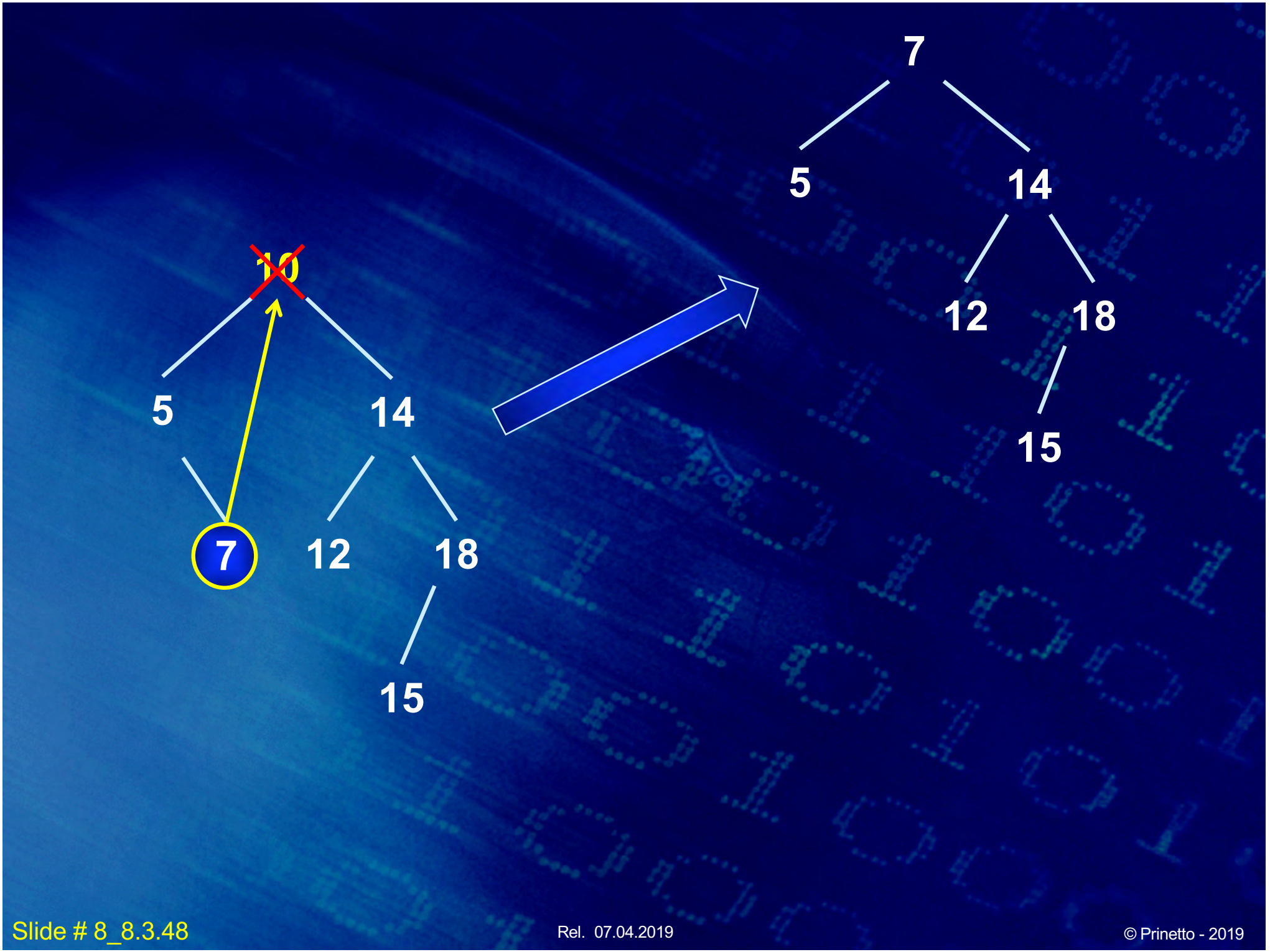
Cancellazione *(cont'd)*

- **l'elemento da cancellare ha due figli:**
  - **l'elemento viene sostituito o dall'elemento immediatamente inferiore (si trova nel nodo più a destra del sottoalbero di sinistra) o immediatamente superiore (si trova nel nodo più a sinistra del sottoalbero di destra)**

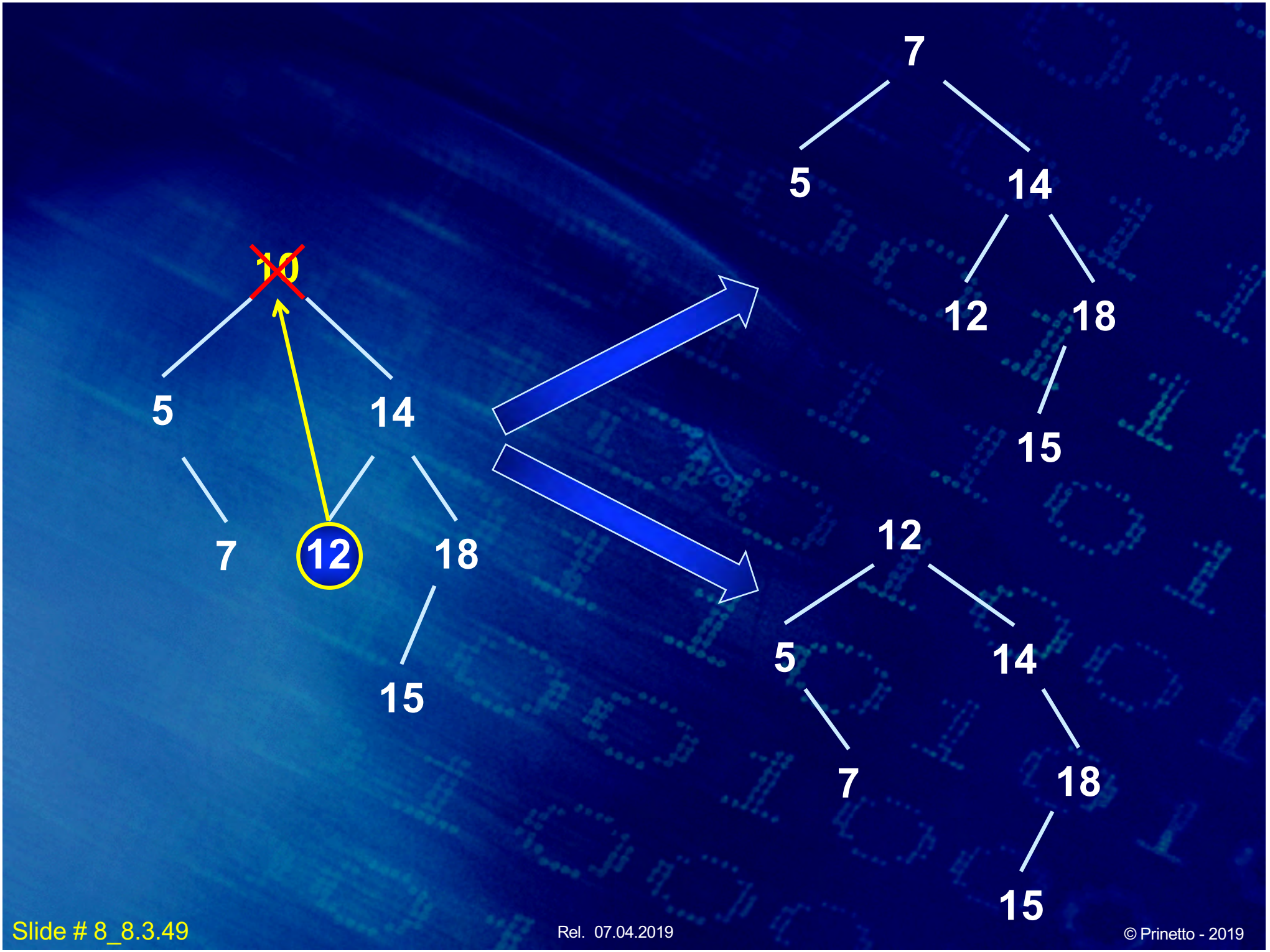
Rel. 07.04.2019

Rel. 07.04.2019

Rel. 07.04.2019



Rel. 07.04.2019



Rel. 07.04.2019

Outline

- Alberi binari di ricerca
- Altri tipi di Alberi:
  - HBT
  - B tree
  - Varianti di B tree

Rel. 07.04.2019

Altri tipi di Alberi

- Come precedentemente evidenziato, il ribilanciamento di un BST può comportare lo spostamento di tutti i nodi dell'albero.
- In letteratura sono stati definiti altri tipi di alberi, quali:
  - HBT
  - M-Way Search Tree
  - B-Tree

  caratterizzate dalla proprietà che le eventuali operazioni necessarie per il ribilanciamento sono "locali", vale a dire limitate al cammino dal nodo interessato alla radice.

Altri tipi di Alberi

- **Ne consegue che per tali alberi tutte le operazioni di ricerca, inserimento e cancellazione hanno, al più, complessità *O(log n)*.**
- **La trattazione di questi tipi di alberi esula dagli scopi del presente corso.**

Rel. 07.04.2019

References

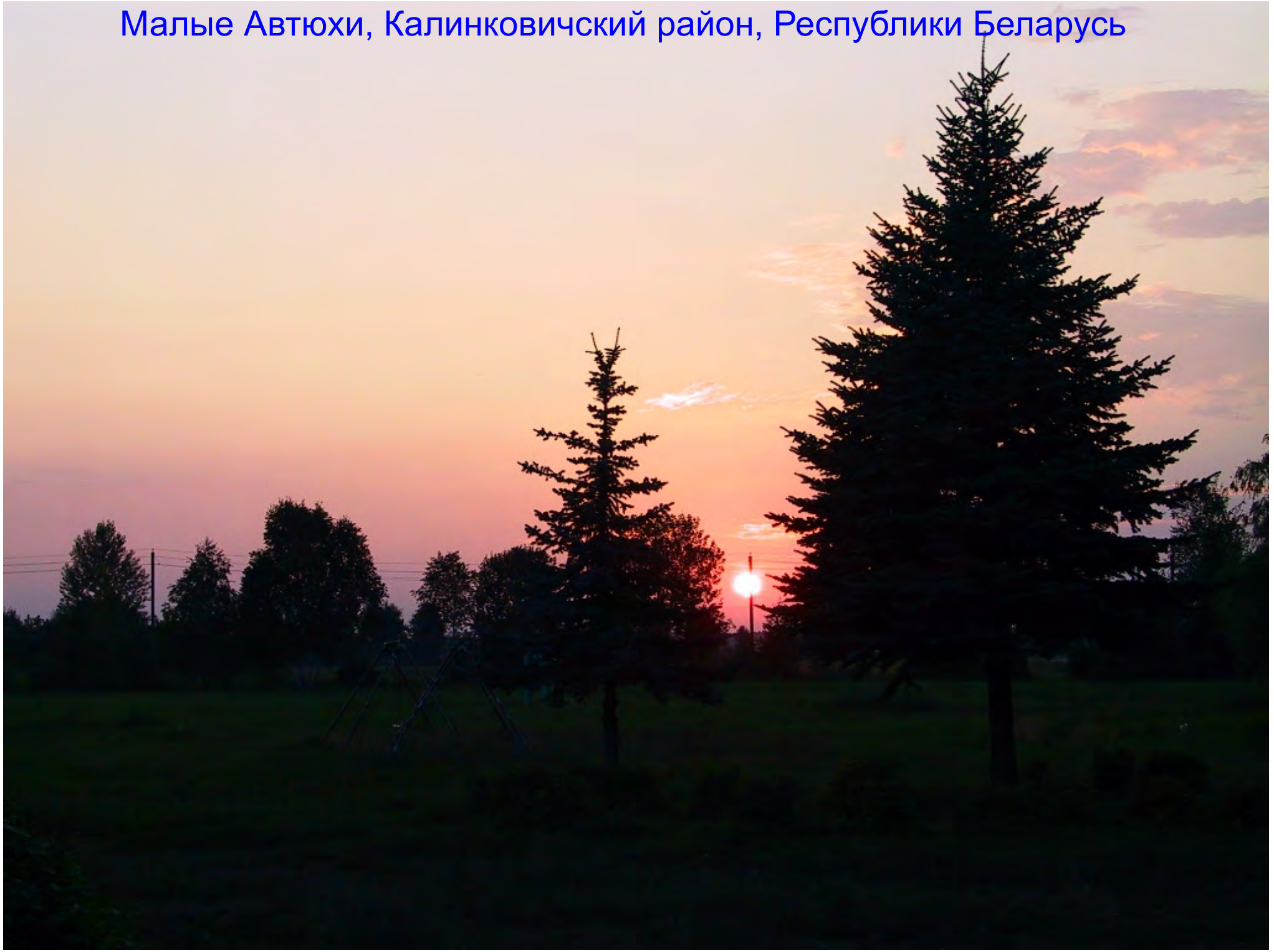
- A.V. Aho, J.E. Hopcroft, J.D. Ullman:
  "Data Structures and Algorithms,"
  Addison Wesley, Reading MA (USA), 1983
  pp. 155-197
- G. Ausiello, A. Marchetti-Spaccamela, M. Protasi:
  "Teoria e Progetto di Algoritmi Fondamentali,"
  Ed. Franco Angeli, Milano, 1985, pp. 186-220
- D. Comer:
  "The ubiquitous B-Treee,"
  Computing Surveys, Vol. 11, No. 2, June 1979,
  pp. 121-137

References

- G.H. Gonnet:
  "Handbook of Algorithms and Data Structures,"
  Addison Wesley, Reading MA (USA), 1984, pp. 69-117
- R. Sedgewick:
  "Algorithms in C,"
  Addison Wesley, Reading MA (USA), 1990
  pp. 215-230, 245-258
- C.J. Van Wyk:
  "Data Structures and C Programs," Addison Wesley, Reading MA (USA), 1988
  pp. 193-224

References

- M.A. Weiss:
  "Data Structures and Algorithm Analysis,"
  The Benjamin/Cummings Publishing Company,
  Redwood City, CA (USA), 1992, pp. 98-146
- N. Wirth:
  "Algorithms + Data Structures = Programs,"
  Prentice Hall, Englewood Cliffs NJ (USA), 1976
  pp. 169-263
- R.J. Wilson:
  "Introduzione alla teoria dei grafi,"
  Cremonese, Roma 1978, pp. 57-76


Малые Автюхи, Калинковичский район, Республики Беларусь