Laboratorio di Algoritmi e Strutture Dati I

Esercitazione 8

martedì 5 maggio

Lezione 8: alberi (definizioni generali)

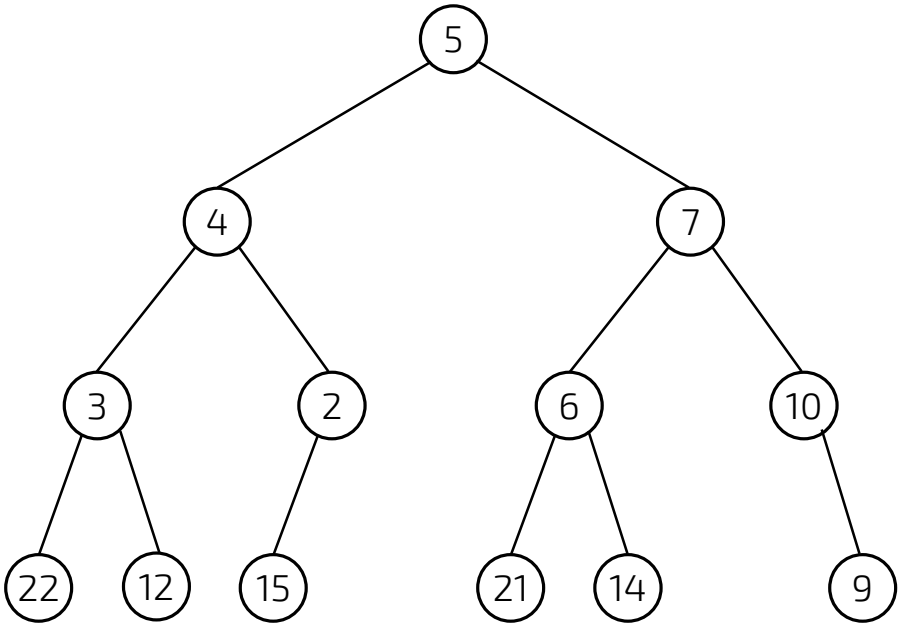
- Nodo: rappresenta sia le informazioni che i rami di collegamento ai sottoalberi;
- Grado di un nodo: numero di rami del nodo (quanti in un albero binario?);
- Grado di un albero: massimo valore dei gradi di ciascun nodo;
- Foglia: nodo avente grado nullo;
- Padre: nodo avente grado non nullo e padre delle radici dei suoi sottoalberi;
- Fratelli: nodi aventi lo stesso padre;
- Antenati di un nodo: nodi che si trovano nel percorso dalla radice al nodo;
- Discendenti di un nodo: nodi che si trovano nei sottoalberi del nodo di cui è radice;
- **Livello** di un nodo: è pari al livello del padre aumentato di 1 (la radice ha livello 1);
- **Altezza** (o **profondità**) di un albero: livello massimo di un nodo dell'albero.

Lezione 8: alberi binari

Definiamo un albero binario come una struttura dati astratta, caratterizzata da un insieme finito di nodi (anche vuoto) composto da:

- **un nodo radice;**
- **un sottoalbero sinistro (anch'esso binario);**
- **un sottoalbero destro (anch'esso binario).**

In altre parole, ogni nodo può avere **al massimo** due figli (grado due).



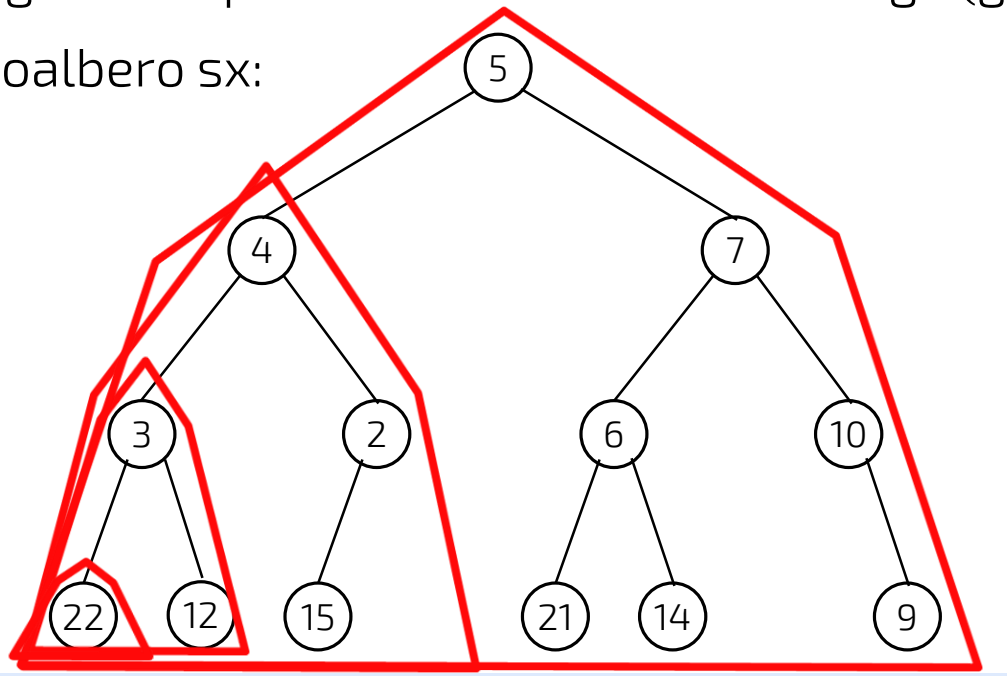
Lezione 8: alberi binari

Definiamo un albero binario come una struttura dati astratta, caratterizzata da un insieme finito di nodi (anche vuoto) composto da:

- **un nodo radice;**
- **un sottoalbero sinistro (anch'esso binario);**
- **un sottoalbero destro (anch'esso binario).**

In altre parole, ogni nodo può avere **al massimo** due figli (grado due).

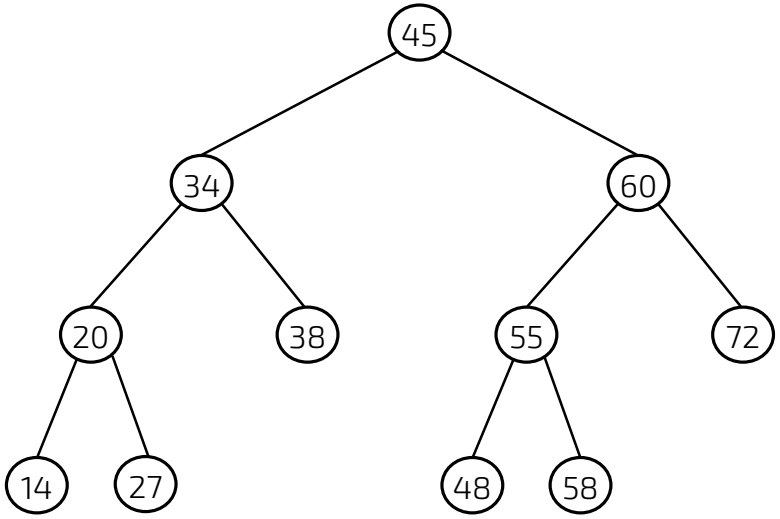
- Esempio sottoalbero sx:



Lezione 8: alberi binari di ricerca (ABR)

Definiamo un albero **binario di ricerca** come un albero binario con le seguenti caratteristiche:

- le chiavi sono uniche;
- le chiavi del sottoalbero sinistro non vuoto sono più piccole della chiave della radice;
- le chiavi del sottoalbero destro non vuoto sono più grandi della chiave della radice;
- I sottoalberi sinistro e destro sono anch'essi alberi binari di ricerca.



Lezione 8: rappresentazione degli ABR

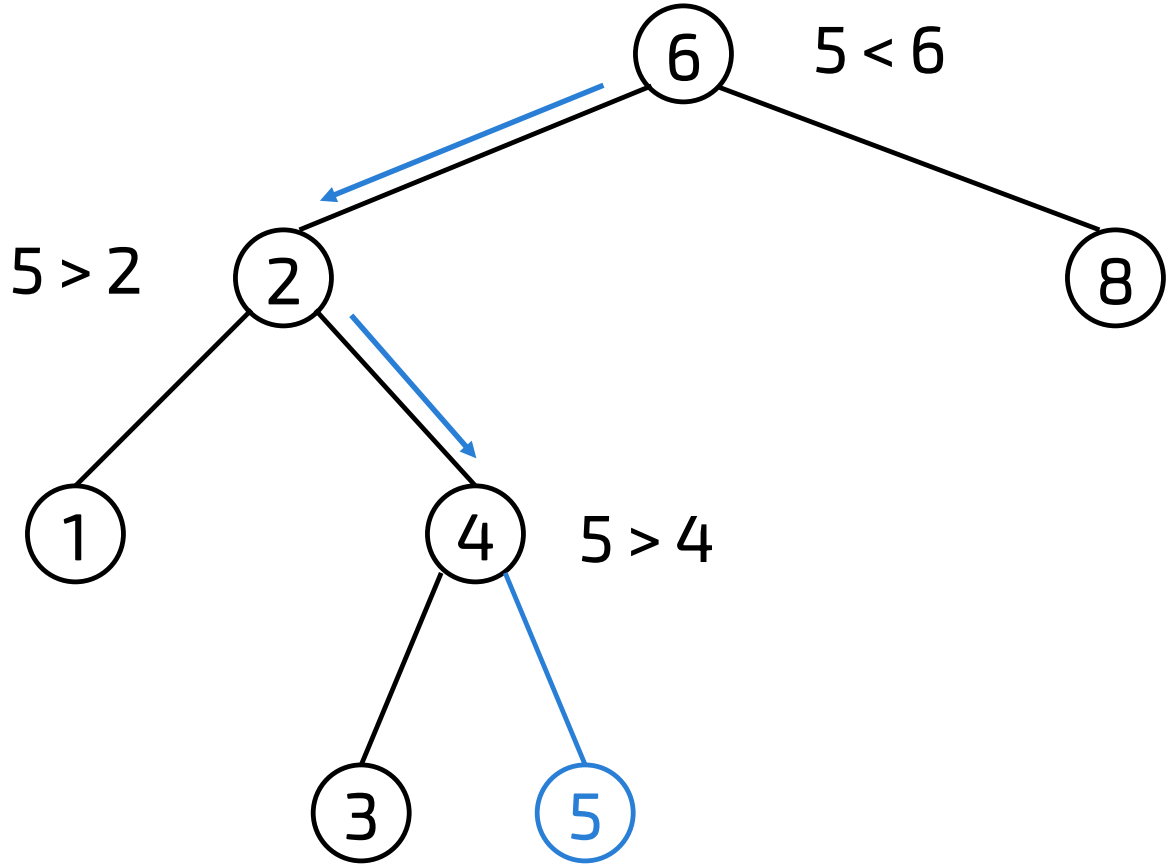
Un albero binario di ricerca è rappresentato mediante nodi collegati tramite puntatori:

- ogni nodo contiene un campo chiave;
- ogni nodo ha un riferimento al suo figlio sinistro, destro e al padre.

```
typedef struct nodo
{
   Type key;
   struct nodo *parent;
   struct nodo *left;
   struct nodo *right;
}Nodo;
```

Lezione 8: Inserimento in un ABR

Inserimento del valore 5



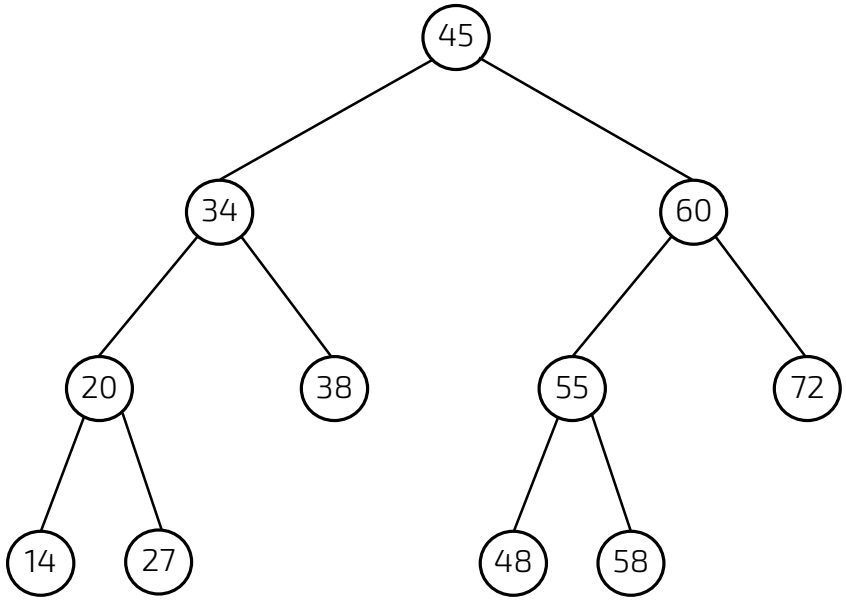
Lezione 8: Inserimento in ABR

```
algoritmo insert_nodo(puntatore a radice, puntatore a nuovo_nodo) → puntatore a Nodo

y ← NULL
x ← puntatore a radice
while (x ≠ NULL) do  // L'algoritmo cerca un cammino discendente dalla radice
     y ← x           // fino ad una foglia; x segue il cammino, y punta al padre di x
     if (key di nuovo_nodo < key di x) then
          x ← figlio sx di x
     else
          x ← figlio dx di x

// usciti da questo ciclo y è il puntatore al padre del nuovo nodo
padre di nuovo_nodo ← y

if (y == NULL) then
     radice ← nuovo_nodo
else if (key di nuovo_nodo < key di y) then
     figlio sx di y ← nuovo_nodo
else
     figlio dx di y ← nuovo_nodo

return puntatore a radice
```

Lezione 8: visite in profondità

Le visite in profondità permettono di visitare i nodi una sola volta durante l'attraversamento e sono:

- Inorder (SVD);
- Preorder (VSD);
- Postorder (SDV).



Inorder: 14, 20, 27, 34, 38, 45, 48, 55, 58, 60, 72.

Preorder: 45, 34, 20, 14, 27, 38, 60, 55, 48, 58, 72.

Postorder: 14, 27, 20, 38, 34, 48, 58, 55, 72, 60, 45.

Lezione 8: visite in profondità

Implementare gli algoritmi ricorsivi per le visite in profondità appena definite.

```
algoritmo inorder(puntatore a Nodo r) → void
if(r == NULL) then
   return
inorder(figlio sx di r)
visita il nodo r        // stampa contenuto di r
inorder(figlio dx di r)
```

```
algoritmo preorder(puntatore a Nodo r) → void
if(r == NULL) then
   return
visita il nodo r        // stampa contenuto di r
preorder(figlio sx di r)
preorder(figlio dx di r)
```

```
algoritmo postorder(puntatore a Nodo r) → void
if(r == NULL) then
   return
postorder(figlio sx di r)
postorder(figlio dx di r)
visita il nodo r        // stampa contenuto di r
```

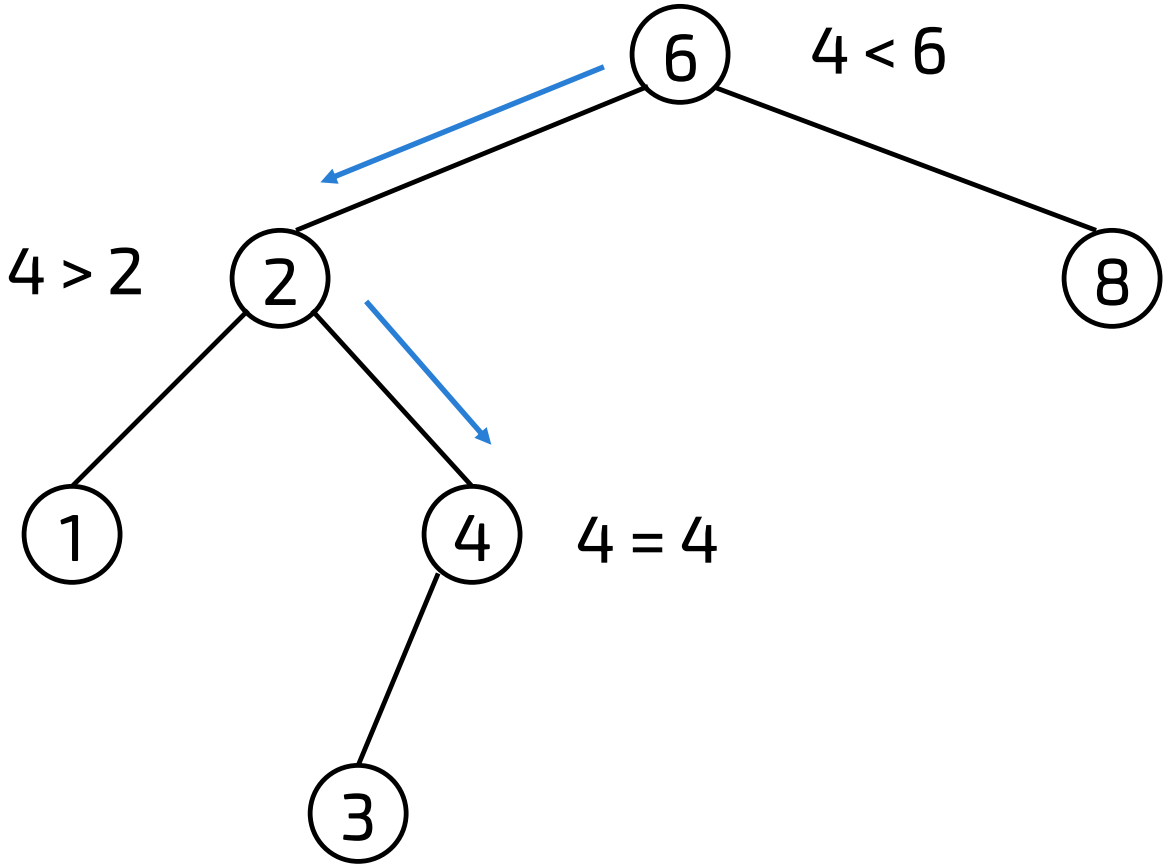
Lezione 8: profondità di un albero binario

- Scrivere un algoritmo che calcoli la profondità di un albero binario.
- Reminder. **Altezza (o profondità) di un albero: livello massimo di un nodo dell'albero.**
- NB: potrebbe esservi utile una funzione *max* che calcoli il massimo tra due interi dati.

```
algoritmo profondità(puntatore a Nodo r) → intero
if(r == NULL) then
    return 0

sx ← profondità(figlio sx di r)
dx ← profondità(figlio dx di r)

return 1 + max(sx, dx)
```

Lezione 8: ricerca di un valore in un ABR

Ricerca del valore 4



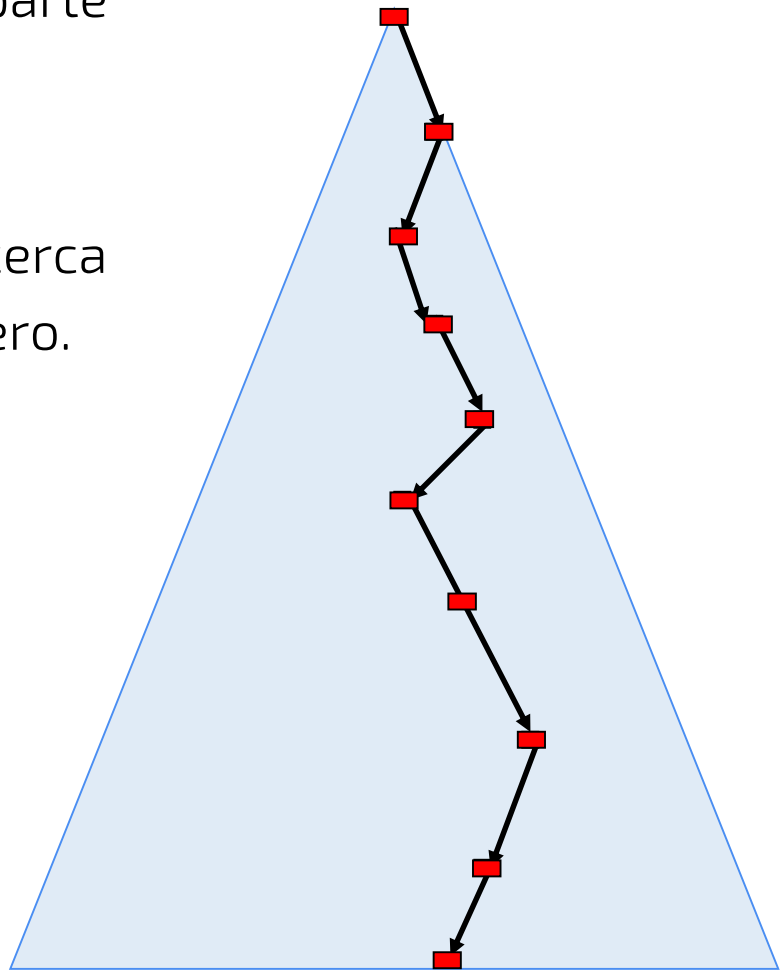
Lezione 8: ricerca in un ABR

La ricerca di un valore all'interno dell'albero si articola su un certo percorso che parte dalla radice e si conclude in una foglia.

La complessità computazionale della ricerca è, quindi, O(h), dove **h** è l'altezza dell'albero.

Un confronto:

- Quanto può valere, al massimo, h? n
- E se avessi un AVL? $\log(n)$



Lezione 8: ricerca in un ABR

Se la radice è NULL, la ricerca non ha successo.

Altrimenti, si confronta la chiave da ricercare con la chiave della radice:

- se i due valori sono uguali, la ricerca termina con successo;
- se la chiave è minore della chiave della radice, allora si effettua la ricerca nel sottoalbero sinistro della radice;
- se la chiave è maggiore della chiave della radice, allora si effettua la ricerca nel sottoalbero destro della radice.

Lezione 8: ricerca in un ABR

```
algoritmo ric_abr_ric(puntatore a Nodo radice, Type chiave) → puntatore a Nodo

if(radice == NULL) then
     return NULL
if(chiave == key di radice) then
     return radice
if(chiave < key di radice) then
     return ric_abr_ric(figlio sx di radice, chiave)

return ric_abr_ric(figlio dx di radice, chiave)
```

```
algoritmo ric_abr_iter(puntatore a Nodo radice, Type chiave) → puntatore a Nodo

while(radice ≠ NULL) do
     if(chiave == key di radice) then
          return radice
     if(chiave < key di radice) then
          radice ← figlio sx di radice
     else
          radice ← figlio dx di radice
return NULL
```

Lezione 8: min e max di un ABR

Per cercare il valore minimo in un ABR, si segue il seguente criterio, a partire da un nodo x:

- se x ha un figlio sinistro, allora il minimo si trova nel suo sottoalbero sinistro;
- se x non ha un figlio sinistro, allora x è il minimo del sottoalbero con radice x.

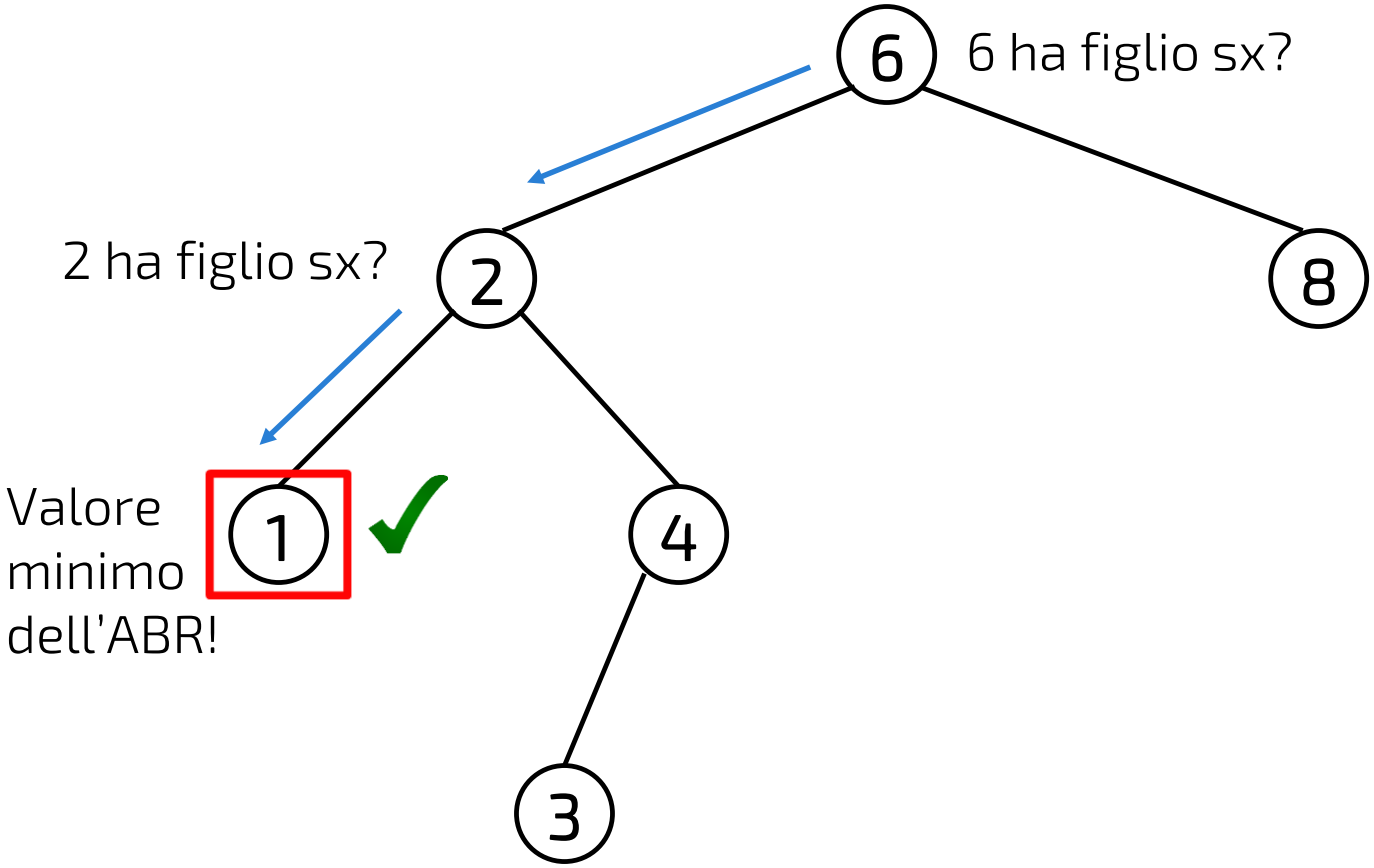
Per cercare il valore massimo si applica il ragionamento simmetrico.

```
algoritmo abr_min(puntatore a Nodo radice) → puntatore a Nodo

 nodo ← radice
 while(figlio sinistro di nodo ≠ NULL) do
      nodo ← figlio sinistro di nodo
 return nodo
```

```
algoritmo abr_max(puntatore a Nodo radice) → puntatore a Nodo

 nodo ← radice
 while(figlio destro di nodo ≠ NULL) do
      nodo ← figlio destro di nodo
 return nodo
```

Lezione 8: min e max di un ABR

Esempio: ricerca del valore minimo del sottoalbero con radice x=6



Lezione 8: successore e predecessore in ABR

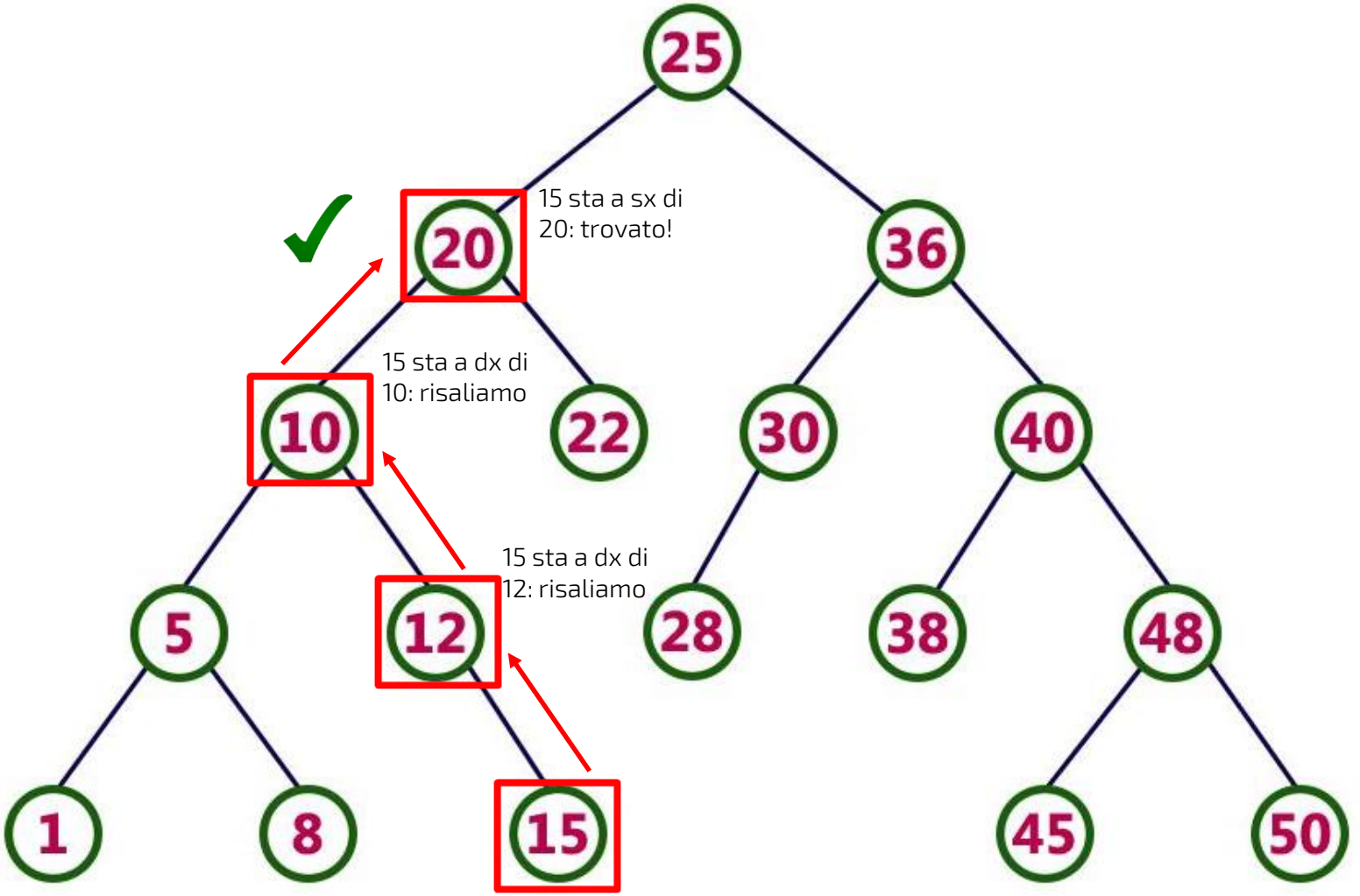
Per cercare il successore di un nodo dato si procede come segue:

- se il nodo ha un figlio destro, il successore è il minimo del suo sottoalbero destro;
- se il nodo non ha un figlio destro, si risale l'albero finché il nodo di provenienza non sta a sinistra.

Per cercare il predecessore di un nodo dato, si procede simmetricamente.

Lezione 8: successore e predecessore in ABR

Esempio: vogliamo trovare il successore di 15.



Lezione 8: successore e predecessore in ABR

```
algoritmo abr_pred(puntatore a Nodo x) → puntatore a Nodo

if(figlio sx di x ≠ NULL) then
     return abr_max(figlio sx di x)
y ← padre di x
while(y ≠ NULL and x == figlio sx di y) do
      x ← y
      y ← padre di y
return y
```

```
algoritmo abr_succ(puntatore a Nodo x) → puntatore a Nodo

if(figlio dx di x ≠ NULL) then
     return abr_min(figlio dx di x)
y ← padre di x
while(y ≠ NULL and x == figlio dx di y) do
      x ← y
      y ← padre di y
return y
```

Lezione 8: cancellazione per copiatura in un ABR

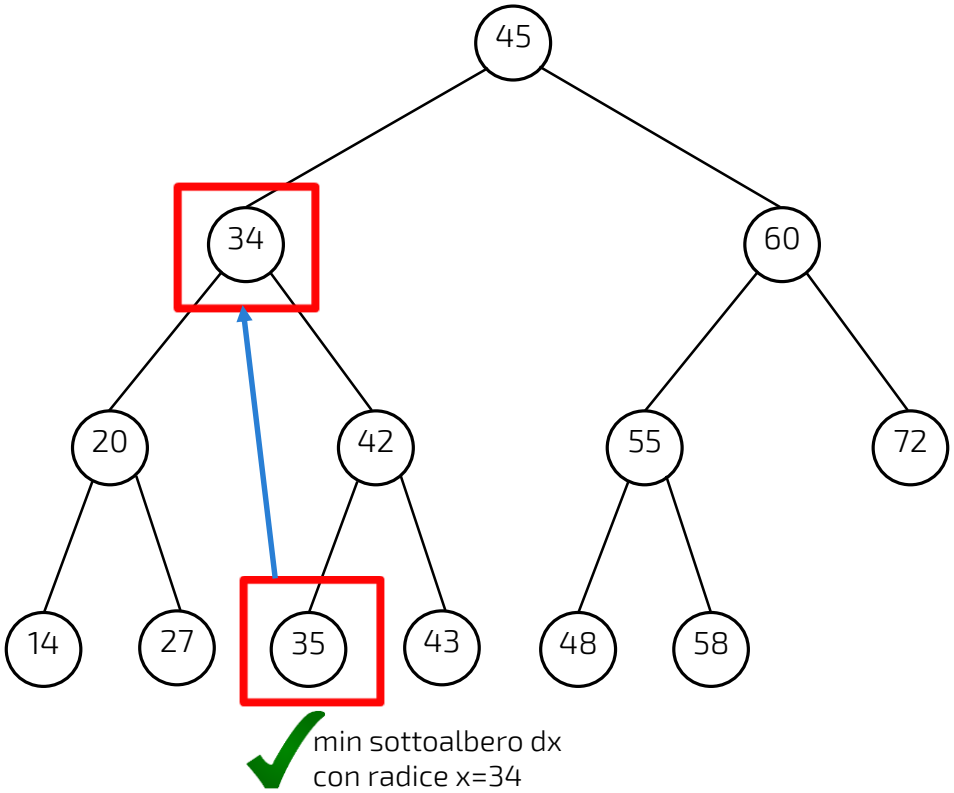
Per cancellare un nodo da un ABR, si devono prevedere tre differenti casi:

- cancellazione di un nodo di grado 0 (foglia): si imposta a NULL il campo figlio (sx o dx, a seconda del caso) del padre del nodo cancellato;
- cancellazione di un nodo di grado 1: si collega il sottoalbero del nodo da cancellare con il campo figlio del padre;
- cancellazione di un nodo di grado 2: si sostituisce il nodo da cancellare con l'elemento più piccolo del suo sottoalbero destro (oppure con il più grande del suo sottoalbero sinistro), infine si cancella anche tale nodo.

Lezione 8: cancellazione per copiatura in un ABR

Cancellazione di un nodo di grado 2: si sostituisce il nodo da cancellare con l'elemento più piccolo del suo sottoalbero destro (oppure con il più grande del suo sottoalbero sinistro), infine si cancella anche tale nodo.

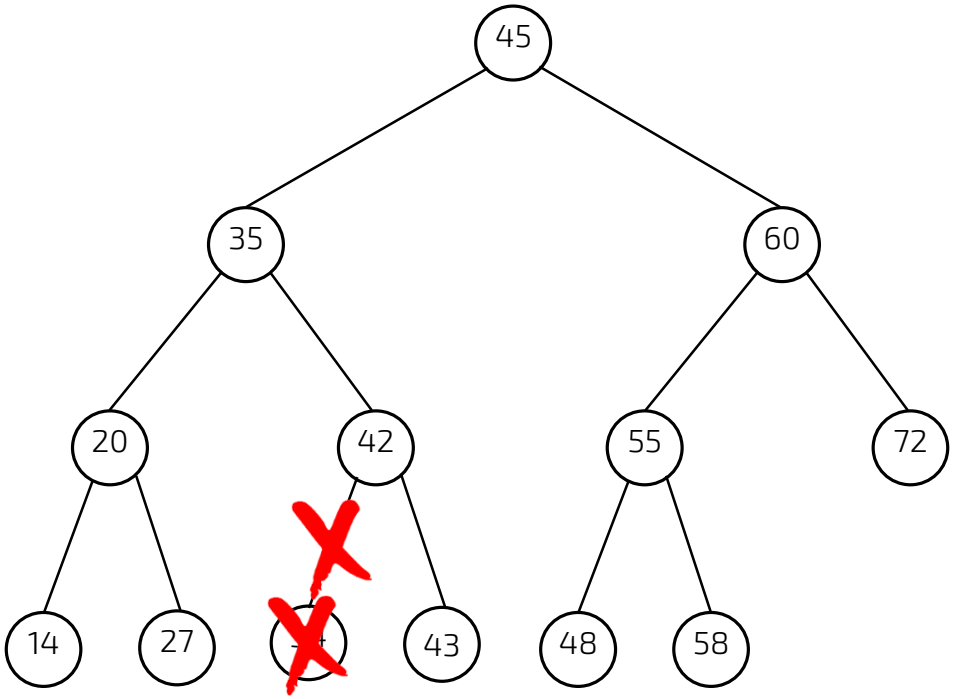
Esempio: cancelliamo il valore 34.



Lezione 8: cancellazione per copiatura in un ABR

Cancellazione di un nodo di grado 2: si sostituisce il nodo da cancellare con l'elemento più piccolo del suo sottoalbero destro (oppure con il più grande del suo sottoalbero sinistro), infine si cancella anche tale nodo.

Esempio: cancelliamo il valore 34.



Lezione 8: cancellazione per copiatura in un ABR

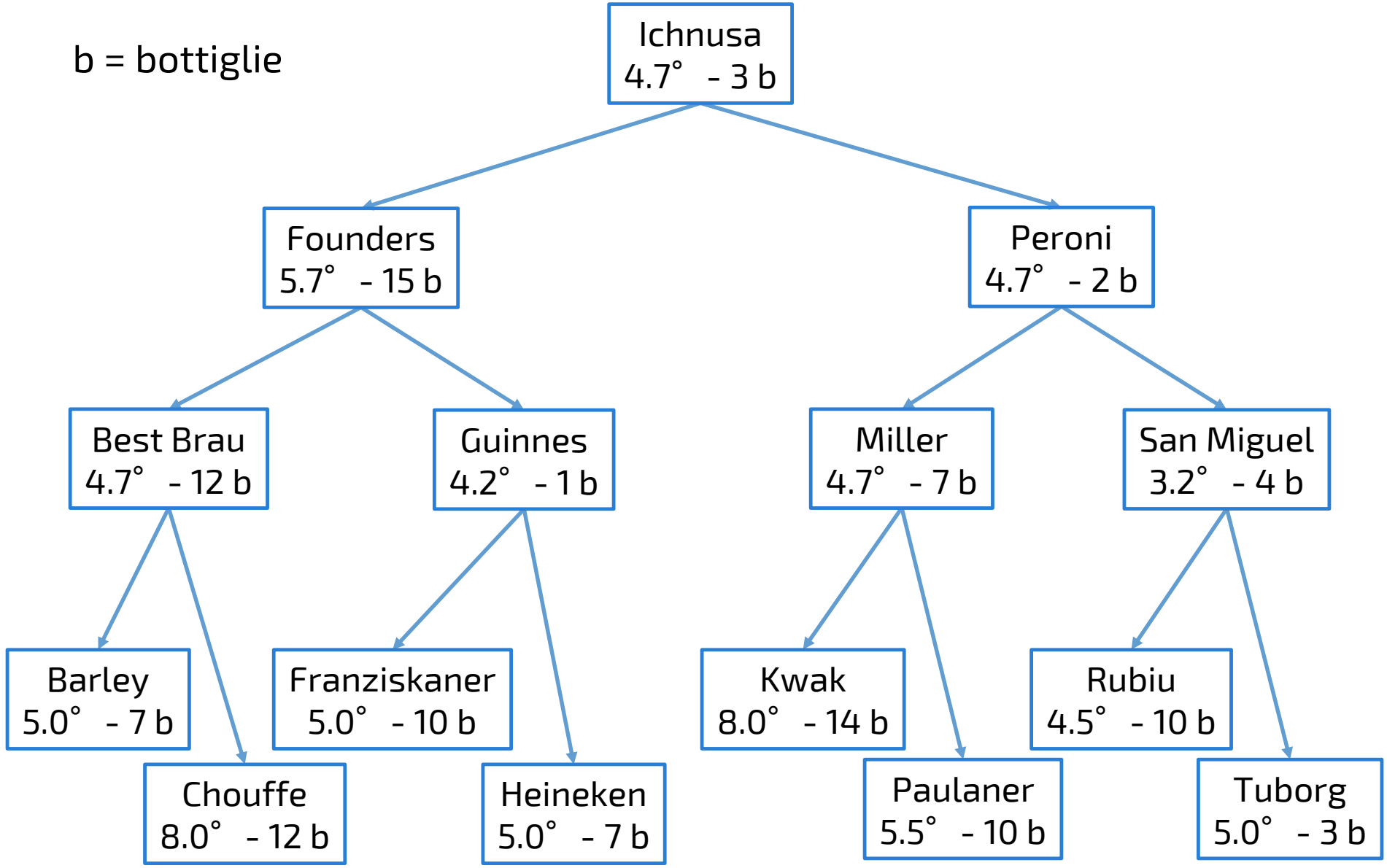
```
algoritmo delete_nodo(puntatore a radice, puntatore a nodo) → puntatore a Nodo

if(figlio sx di nodo == NULL or figlio dx di nodo == NULL) then
     y ← nodo
else
     y ← abr_succ(nodo)

if(figlio sx di y ≠ NULL) then
     x ← figlio sx di y
else
     x ← figlio dx di y

if(x ≠ NULL) then
     padre di x ← padre di y

if(padre di y == NULL) then
     radice ← x
else if(figlio sx di padre di y == y) then
     figlio sx di padre di y ← x
else
     figlio dx di padre di y ← x
if(y ≠  nodo) then
     key di nodo ← key di y

dealloca y
return radice
```

Esercizio 8: A-*Beer*-R

Esercizio 8: A-Beer-R

Esercizio 8: A-Beer-R (astemio version)

b = bottiglie



Esercizio 8: Strutture necessarie

- Rappresentiamo una birra con una struttura così definita:

```
typedef struct
{
    char nome[DIM];
    float gradi;
    int bottiglie;
} Birra;
```

- Un nodo dell'ABR è quindi strutturato come segue:

```
typedef struct nodo
{
   Birra key;
   struct nodo *parent;
   struct nodo *left;
   struct nodo *right;
}Nodo;
```

Esercizio 8: Progetto base

- Andate su eLearning e copiate il codice del progetto base per l'esercitazione odierna.
- Create un nuovo progetto su Clion e incollateci il codice.
- Troverete già definite le varie strutture e diverse funzioni, tra cui:
  - *acquisisci_birra()* → *Birra* chiede all'utente i dati di una Birra e la restituisce in una struttura di tipo Birra.
  - *stampa_birra(Birra b)* → *void* prende in input una struttura di tipo Birra e ne stampa le sue informazioni sul terminale.
  - *crea_nodo (Birra b)* → *Nodo** prende in input una struttura di tipo Birra e la inserisce in un Nodo, che viene restituito come output (pronto per essere inserito nell'ABR).
  - *carica_test()* → *Nodo** carica in un ABR (restituito come output) delle birre di test. Questa funzione crea l'albero che trovate nelle slide precedenti e la potete utilizzare per testare le funzioni che avete implementato. NB: ovviamente, funzionerà adeguatamente solo dopo aver implementato la *insert_nodo*.
  - Un **menu** con varie voci, che richiamano le varie funzioni da implementare.
  - I **prototipi** (firme) di tutte le funzioni da implementare.

Esercizio 8: implementazione e gestione di un ABR

Implementare:

- inserimento di una nuova birra in un ABR;
- visite in profondità (SVD, VSD, SDV);
- calcolo della profondità di un albero ABR;
- ricerca di una birra (scegliere una versione tra la ricorsiva oppure l'iterativa); NB: passare il nome della birra (stringa) come parametro per la ricerca.
- ricerca del minimo e del massimo valore di un ABR (i valori sono birre);
- ricerca del predecessore e del successore di un nodo di un ABR;
- cancellazione per copiatura di una birra (quindi del suo nodo) di un ABR .

Hint: ogni volta che vengono richiamate le funzioni di inserimento o cancellazione, il valore da esse restituito deve essere assegnato alla radice dell'albero dichiarata nel main(). Esempio: *root = insert_nodo(root, nuovo_nodo);* e così via per tutte quelle funzioni che modificano l'albero.

Lezione 8

end().