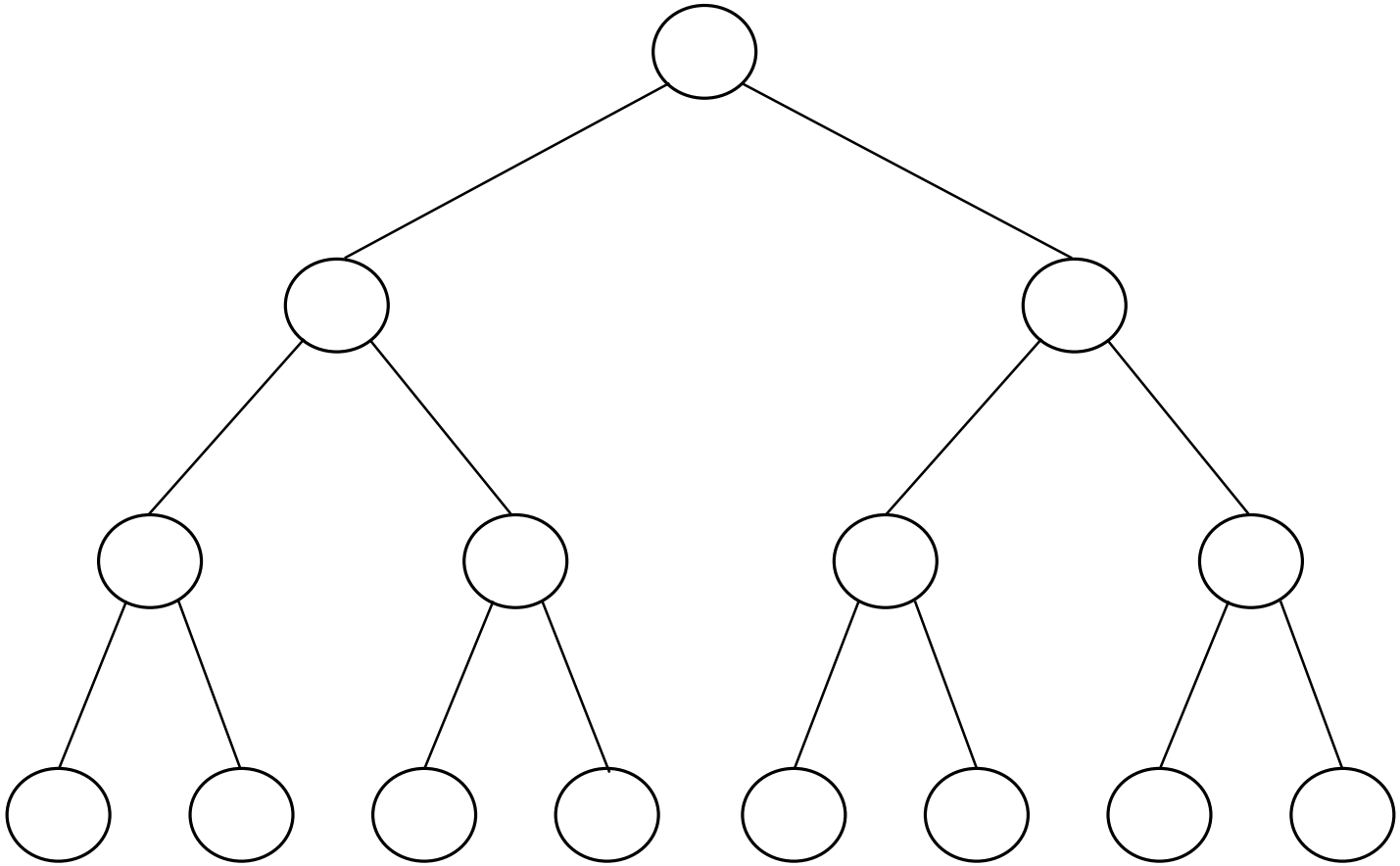
Laboratorio di Algoritmi e Strutture Dati I

Esercitazione 7

martedì 28 aprile

Lezione 7: Alberi, Heap e Coda con Priorità

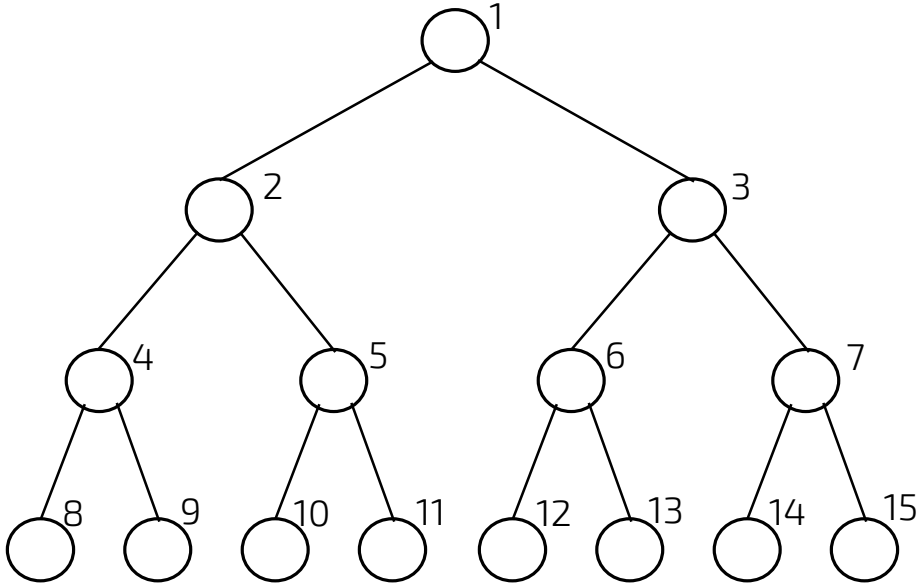


Lezione 7: Alberi (definizioni generali)

- **Nodo**: rappresenta sia le informazioni che i link di collegamento con i 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 7: Alberi binari (definizioni generali)

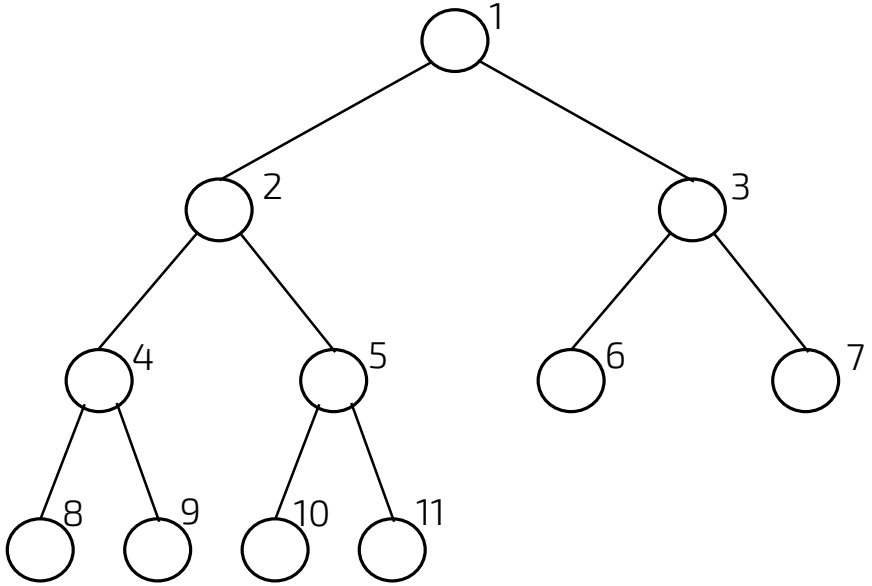
- **Albero pieno:** un **albero binario** di profondità k si dice **pieno** quando è costituito da 2^k-1 nodi



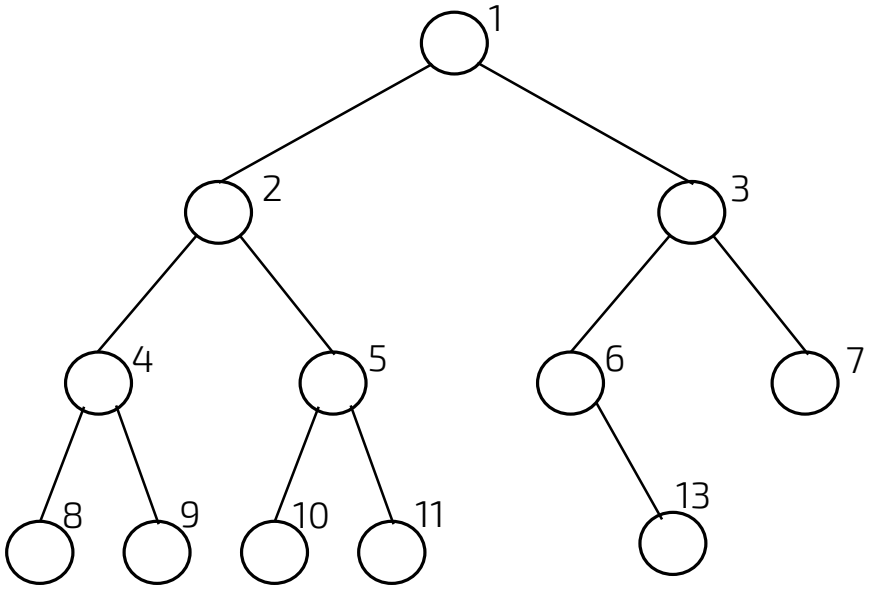
albero binario pieno di profondità 4

Lezione 7: Alberi binari (definizioni generali)

- Albero completo: Un albero binario di profondità k e n nodi si dice **completo** se i suoi nodi corrispondono ai nodi da 1 a n dell'albero pieno.



albero binario completo di profondità k = 4,
con n = 11 nodi



albero binario NON completo
(manca il nodo 12)

Lezione 7: Heap

- **Heap massimo:** un albero binario **completo** di n elementi organizzati in modo che il valore in un nodo qualsiasi sia **maggiore o uguale** di quelli contenuti nei figli.

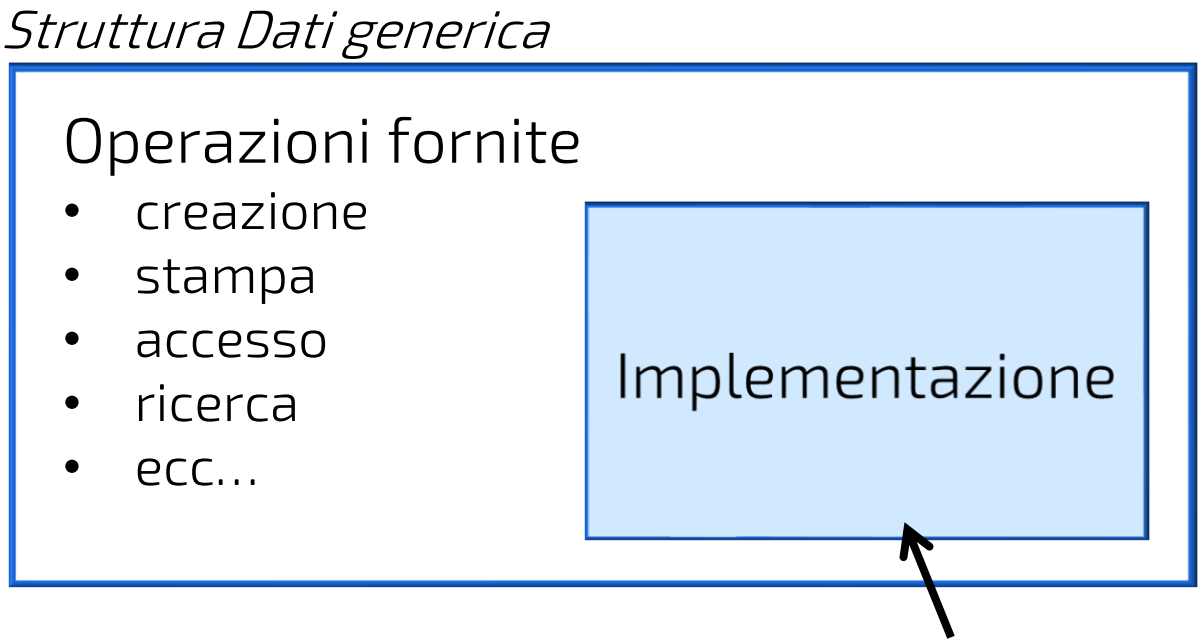
- **Heap minimo:** un albero binario **completo** di n elementi organizzati in modo che il valore in un nodo qualsiasi sia **minore o uguale** di quelli contenuti nei figli.

- Se rappresentiamo l'heap usando array, le posizioni dei nodi sono le seguenti:
  - posizione **1**: nodo **radice**;
  - posizione **i**: nodo **padre**;
  - posizione **2i**: **figlio sinistro** del nodo **i**;
  - posizione **2i+1**: **figlio destro** del nodo **i.**

- Perché la radice è in posizione 1 e non in posizione 0?

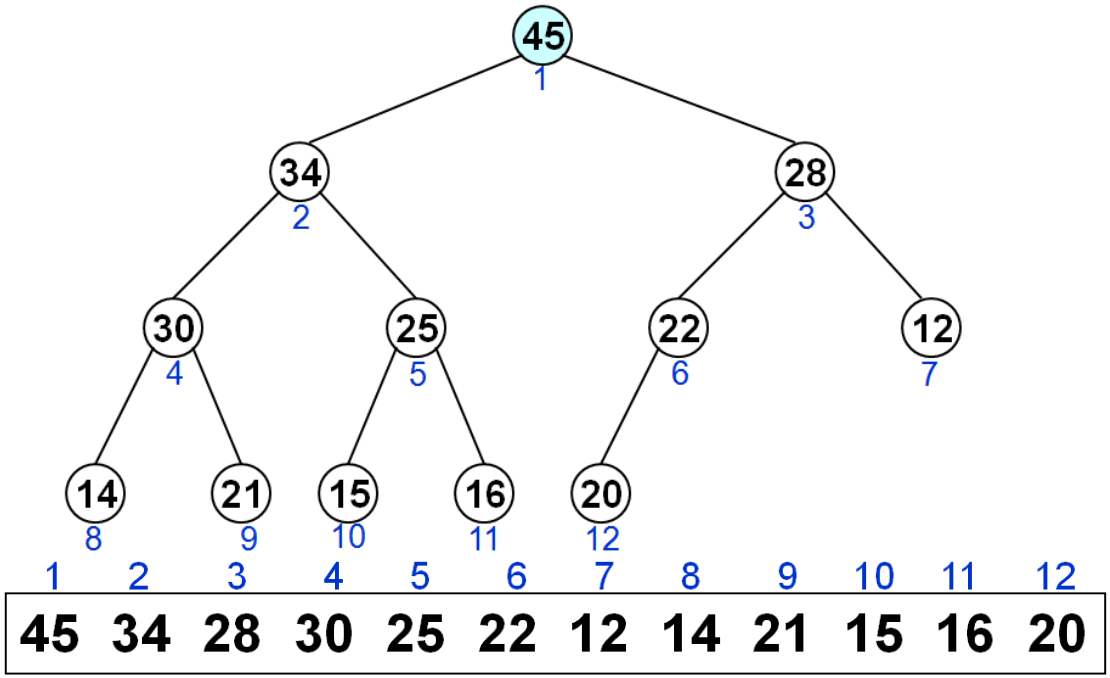
Lezione 7: Strutture Dati Astratte (ancora)

- L'implementazione della Struttura dati deve essere trasparente per l'utente.



Lezione 7: Code con priorità

- Le code di priorità rappresentano una delle applicazioni più efficienti della struttura dati **Heap**
- Una coda con priorità è una struttura dati utilizzata per mantenere un insieme di elementi a ciascuno dei quali è associato un valore chiamato **chiave** e una **priorità** (determinata dalla posizione)
- Essendo l'heap un albero binario **completo**, si può rappresentare utilizzando un array, strutturato come spiegato nella slide precedente



Lezione 7: Inserimento nello Heap

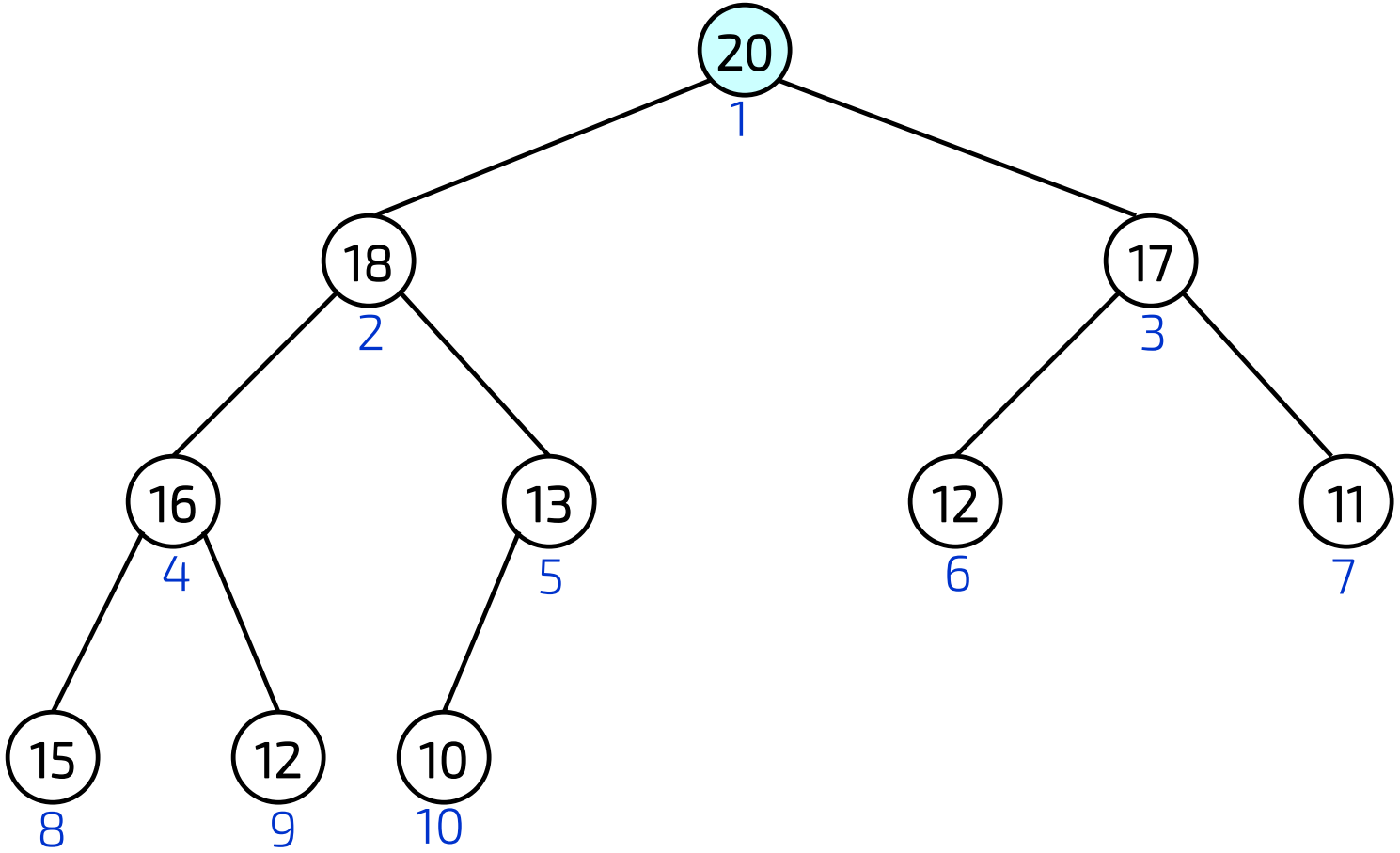
L'inserimento di un nuovo nodo nello Heap avviene seguendo la seguente strategia:

- Si inserisce in fondo allo Heap (come foglia);
- Per determinarne la corretta posizione nello heap, si percorre l'albero verso l'alto finché:
  - si raggiunge il nodo radice
  - oppure si raggiunge una posizione i tale che il valore nella posizione del padre i/2 sia almeno pari al valore da inserire.

```
algoritmo insertHeap(array heap, elemento item, puntatore a intero n) → void
// inserisce un nuovo item in un heap di n elementi

 if (n == DIM_HEAP-1) then
         stampa "L'heap e' pieno"
         return
 n ← n + 1
 i ← n
 while (i ≠ 1 and item ha priorità superiore di heap[i/2]) do
         heap[i] ← heap[i/2]
         i ← i/2
 heap[i] ← item
```

Lezione 7: Inserimento nello Heap (esempio)

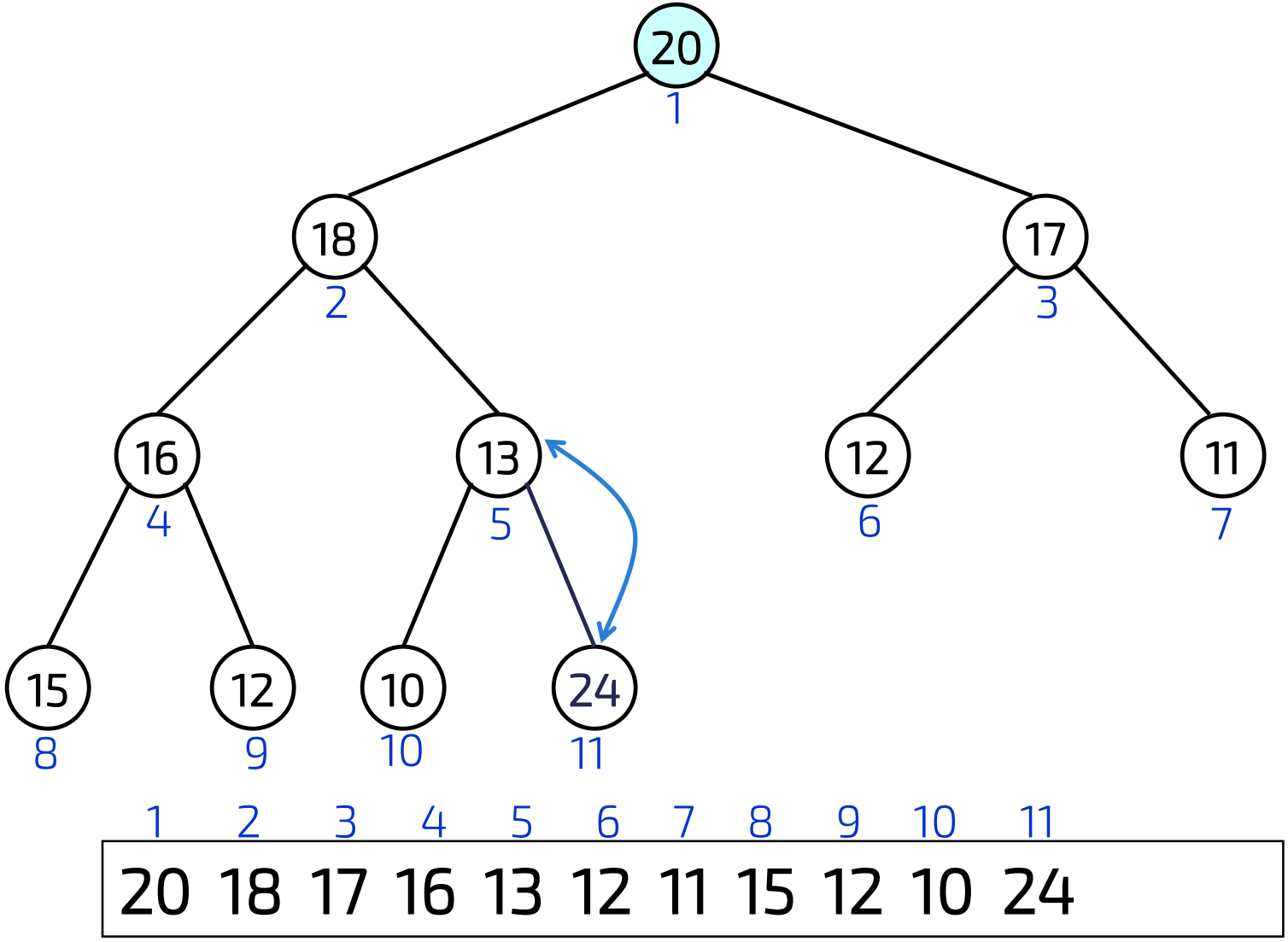
Supponiamo di voler inserire il valore 24.



1	2	3	4	5	6	7	8	9	10
20	18	17	16	13	12	11	15	12	10

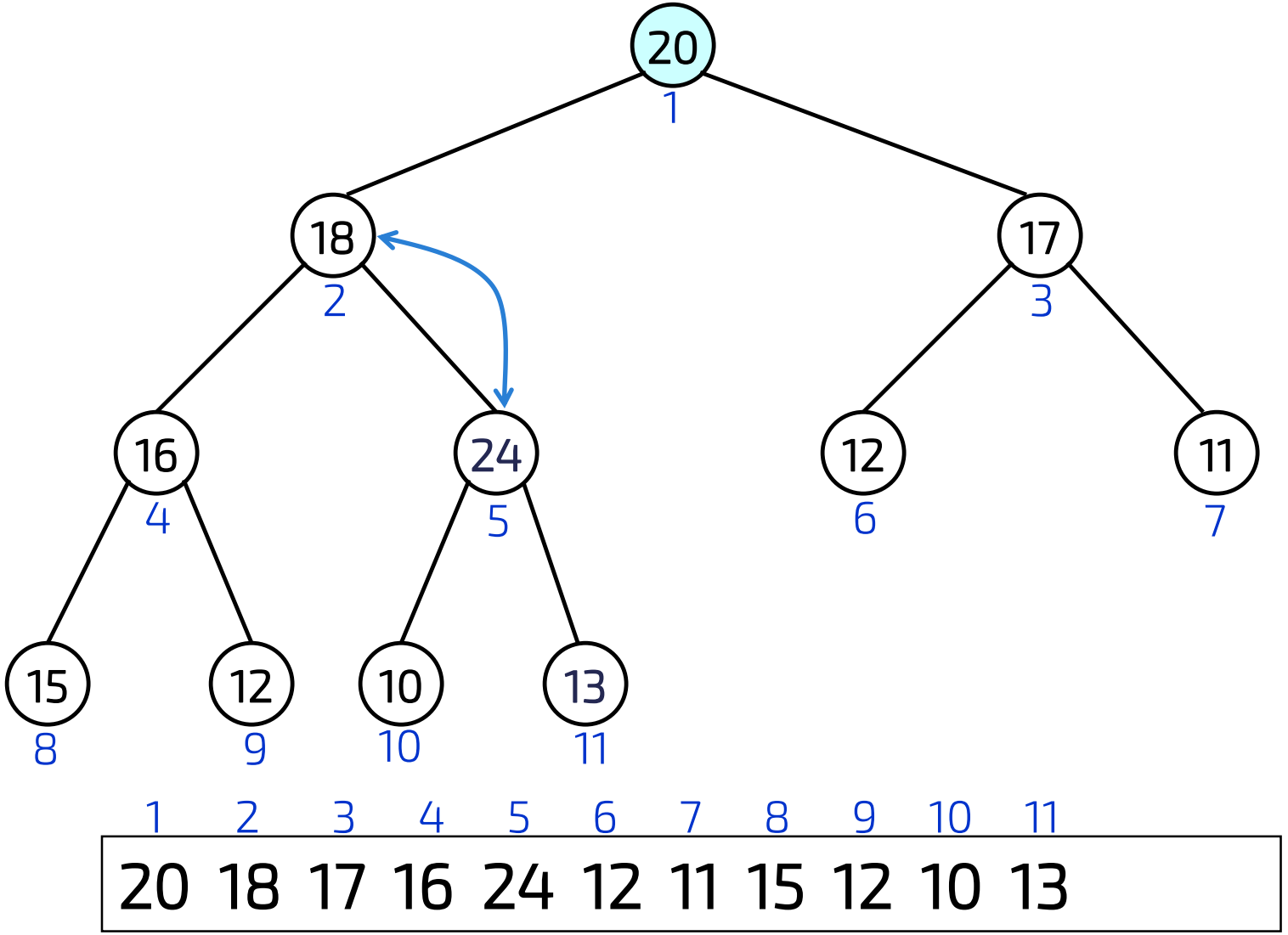
Lezione 7: Inserimento nello Heap (esempio)

Supponiamo di voler inserire il valore 24.



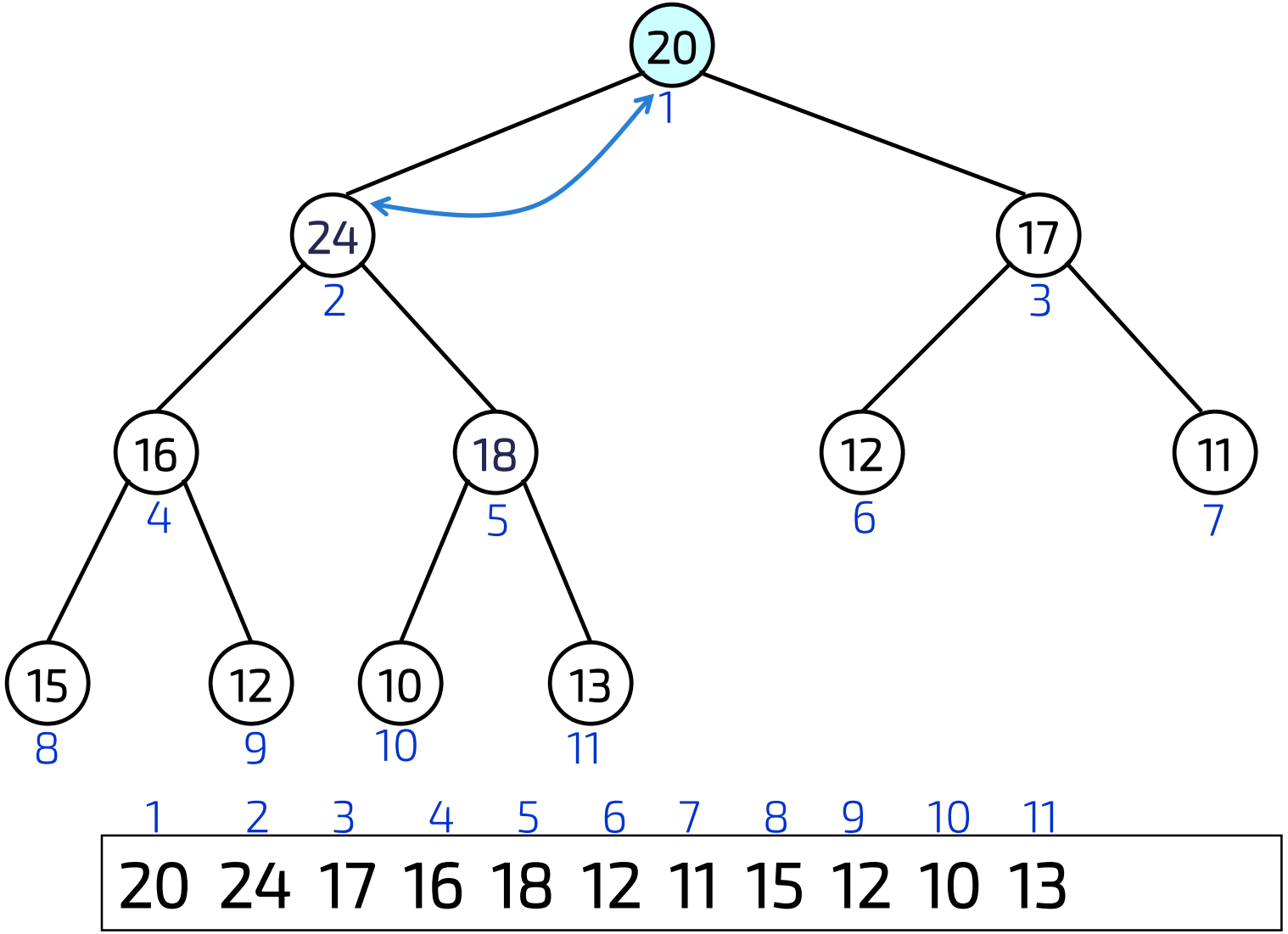
Lezione 7: Inserimento nello Heap (esempio)

Supponiamo di voler inserire il valore 24.



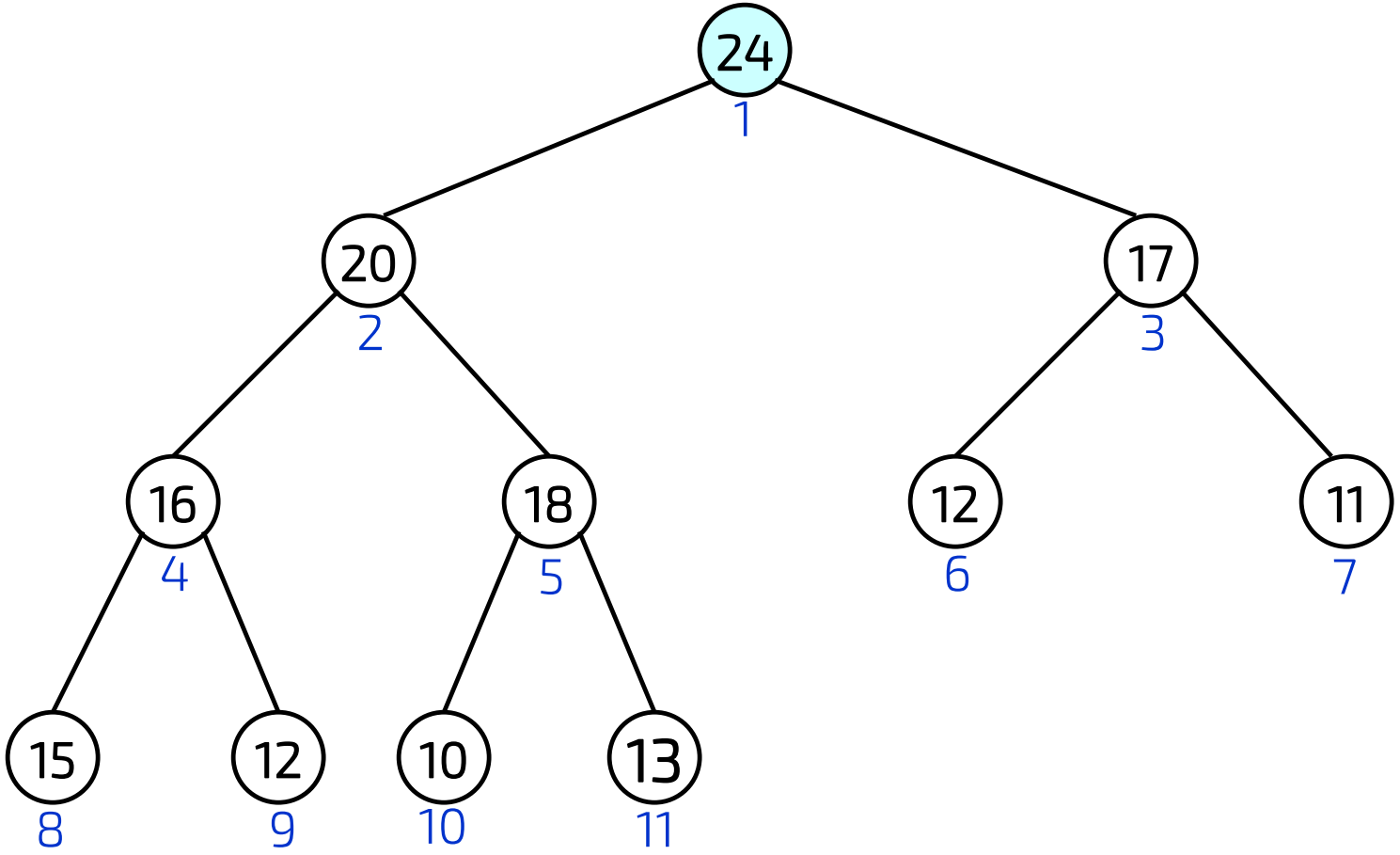
Lezione 7: Inserimento nello Heap (esempio)

Supponiamo di voler inserire il valore 24.



Lezione 7: Inserimento nello Heap (esempio)

Supponiamo di voler inserire il valore 24.



1	2	3	4	5	6	7	8	9	10	11
24	20	17	16	18	12	11	15	12	10	13

Lezione 7: Cancellazione dallo Heap

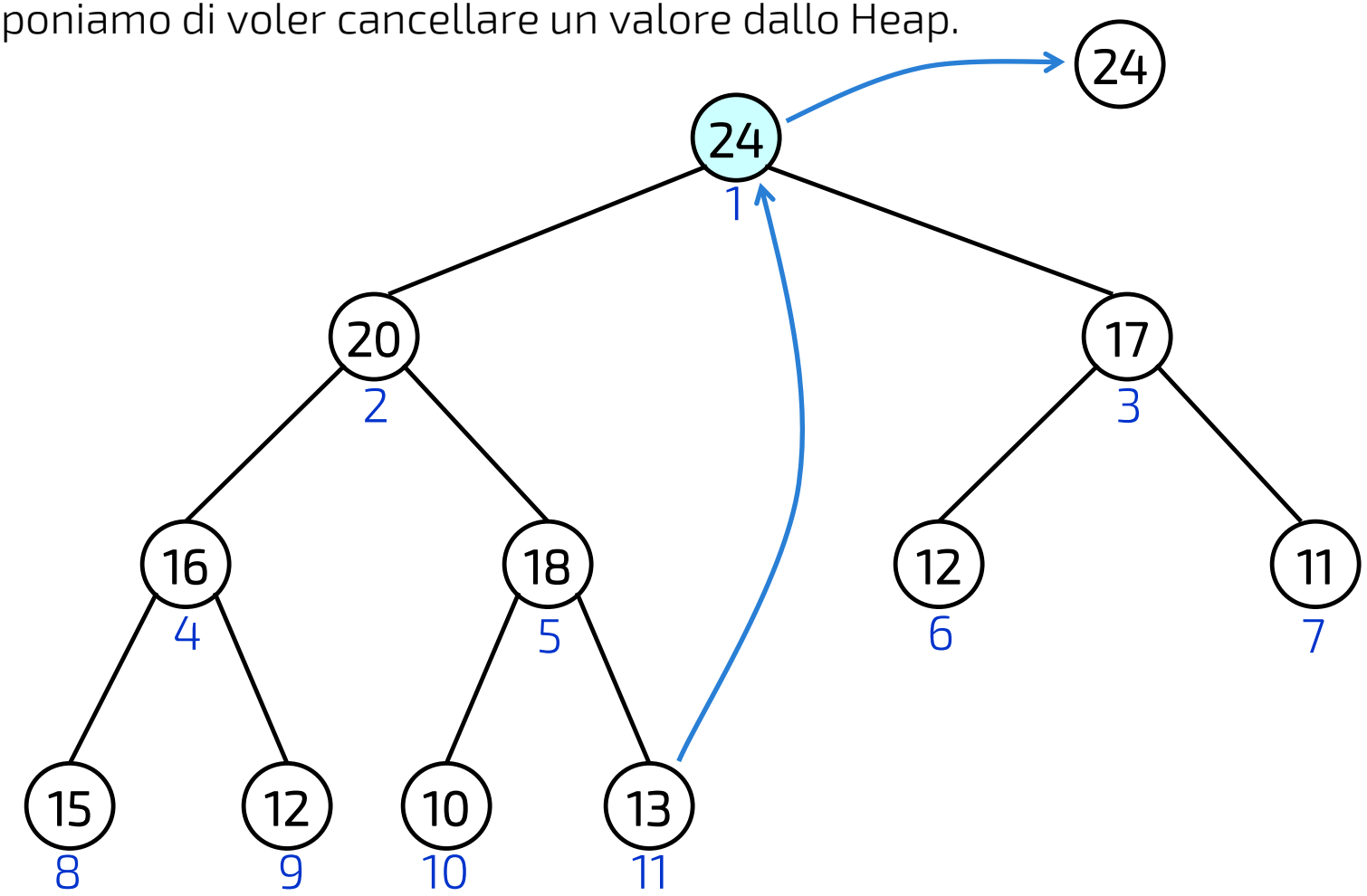
La cancellazione di un nodo dallo Heap avviene secondo la seguente strategia:

- si "sostituisce" il nodo radice con l'ultimo nodo dell'Heap;
- per ripristinare lo Heap, si percorre l'albero verso il basso:
  - si confronta il nodo padre con i suoi figli;
  - si scambiano gli elementi fuori posto.

```
algoritmo deleteHeap(array heap, puntatore a intero n) → elemento
 // cancella e restituisce l'elemento radice in un heap di n elementi
item ← heap[1]
temp ← heap[n]
n ← n-1
padre ← 1
figlio ← 2

while(figlio ≤ n) do
    if (figlio < n and heap[figlio] ha priorita' inferiore di heap[figlio+1]) then
        figlio ← figlio+1

    if  (temp ha priorita' superiore o uguale di heap[figlio]) then
        break

    heap[padre] ← heap[figlio]
    padre ← figlio
    figlio ← 2 × figlio
heap[padre] ← temp

return item
```

Lezione 7: Cancellazione dallo Heap (esempio)

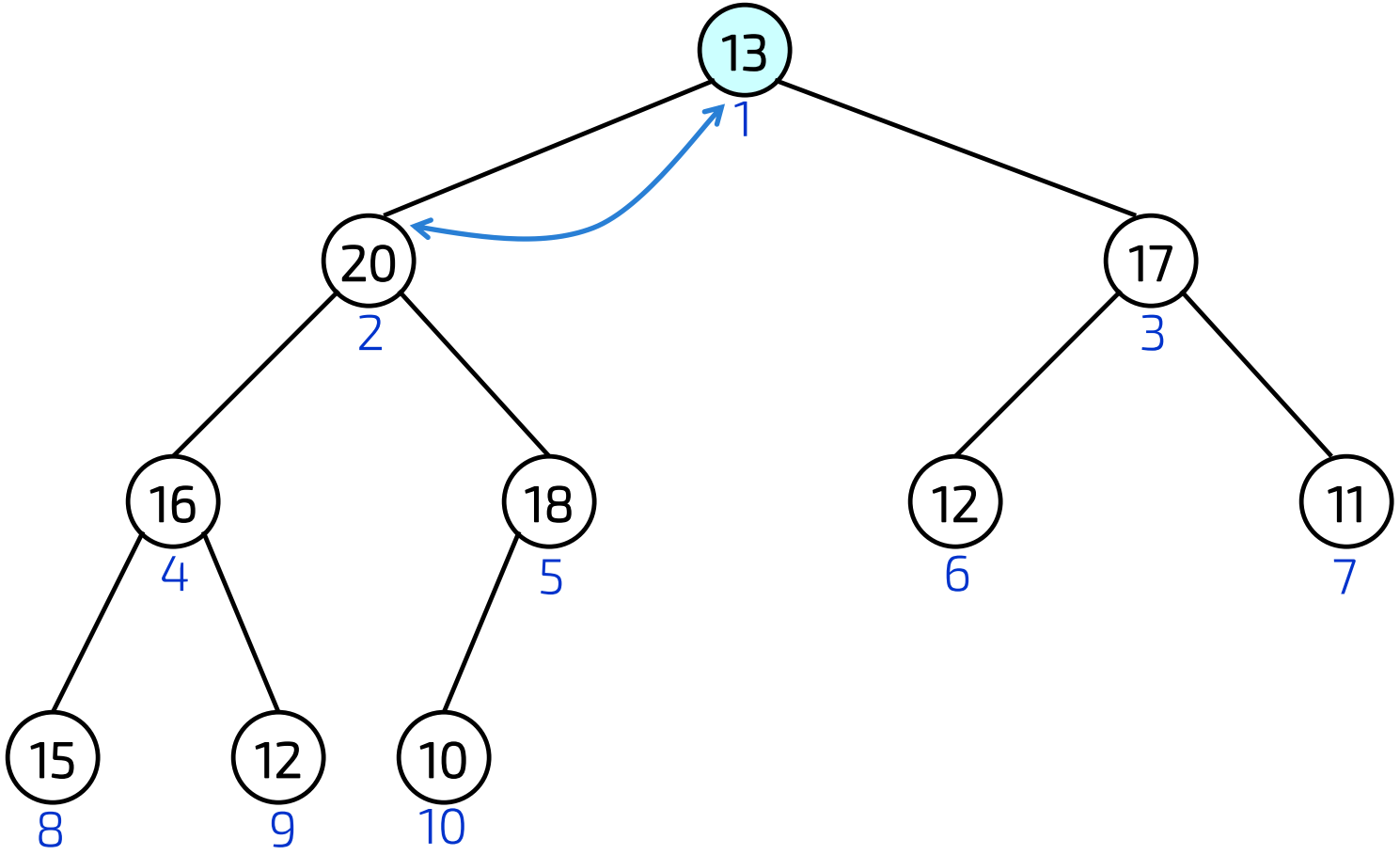
Supponiamo di voler cancellare un valore dallo Heap.



1	2	3	4	5	6	7	8	9	10	11
24	20	17	16	18	12	11	15	12	10	13

Lezione 7: Cancellazione dallo Heap (esempio)

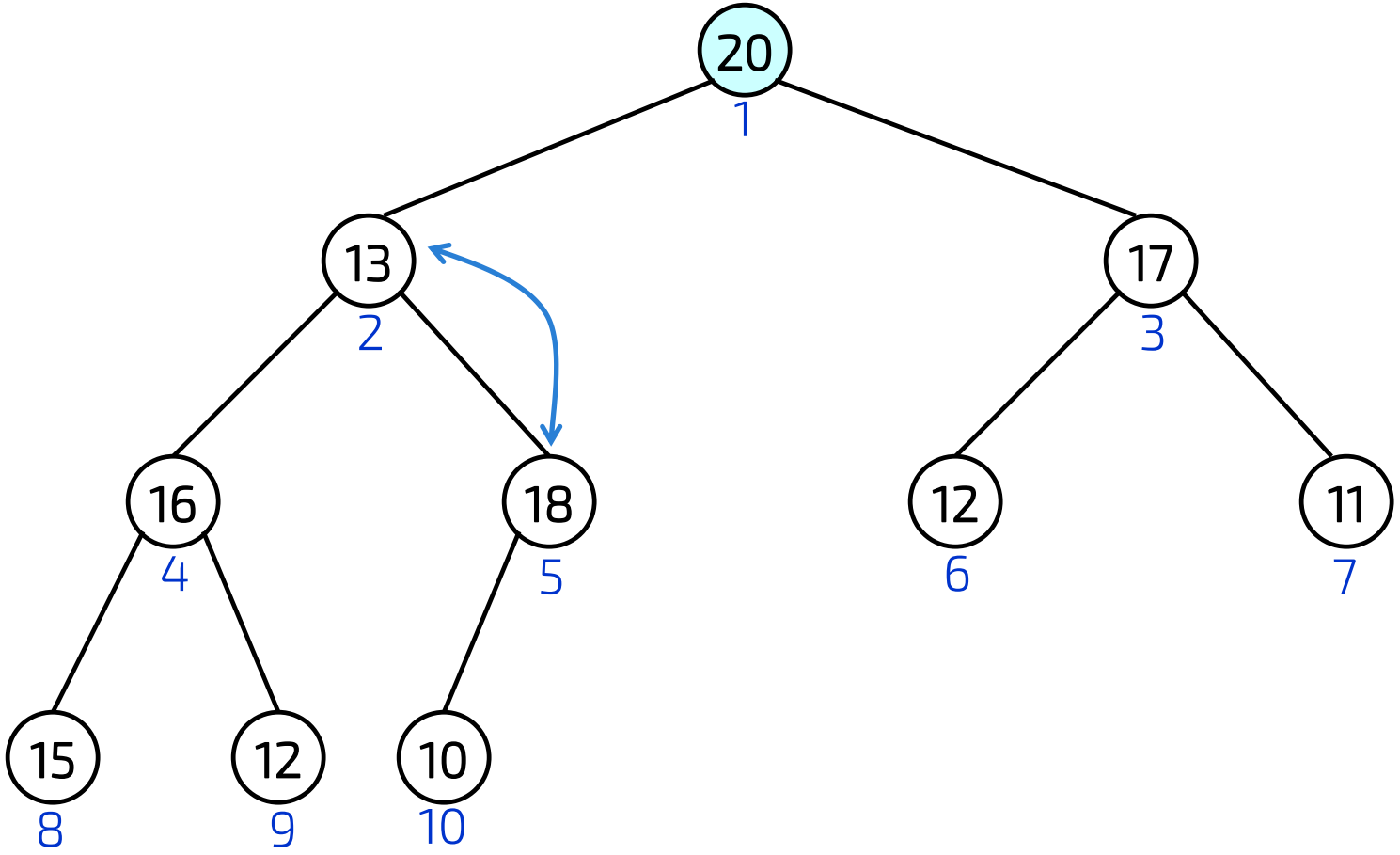
Supponiamo di voler cancellare un valore dallo Heap.



1	2	3	4	5	6	7	8	9	10
13	20	17	16	18	12	11	15	12	10

Lezione 7: Cancellazione dallo Heap (esempio)

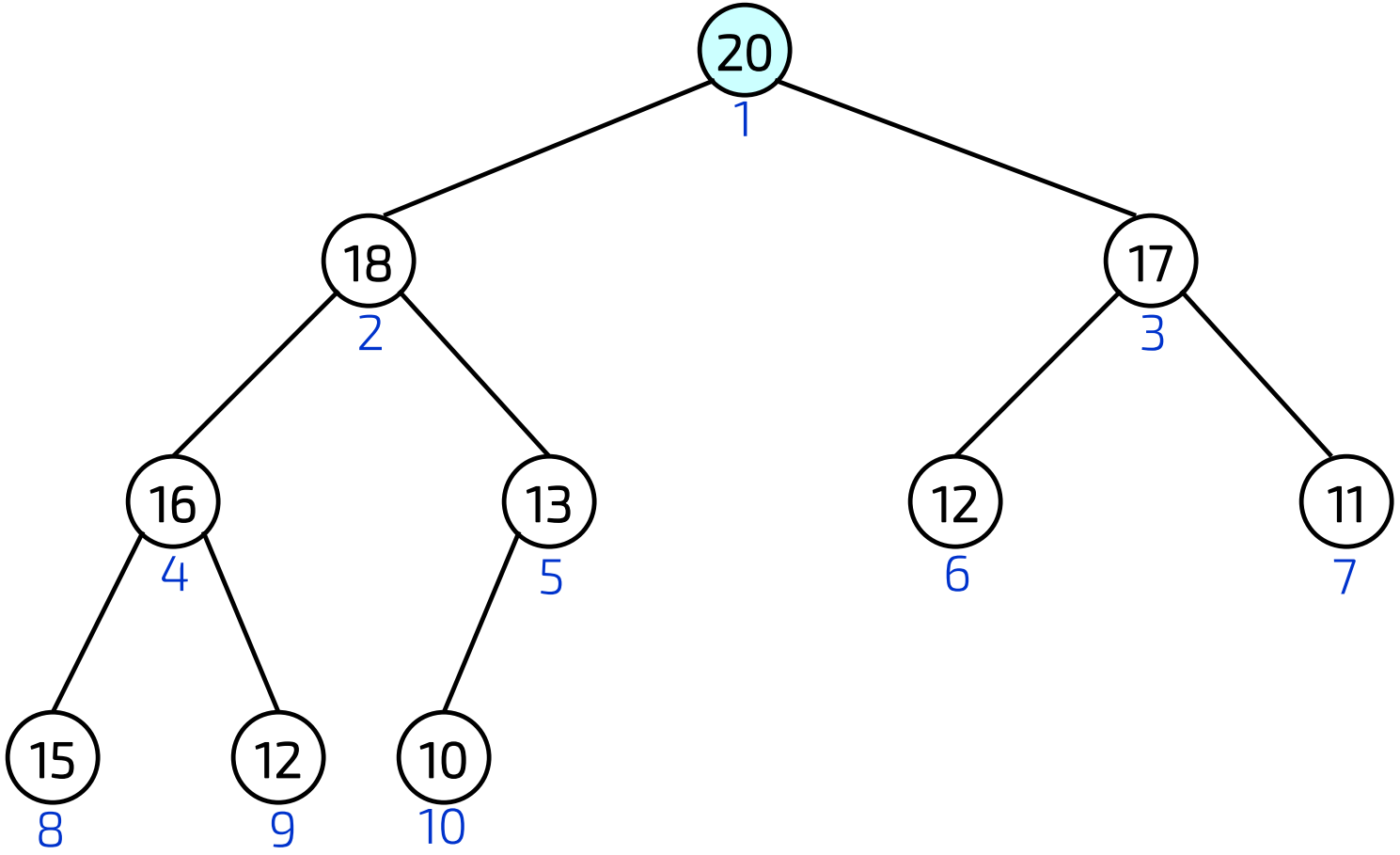
Supponiamo di voler cancellare un valore dallo Heap.



1	2	3	4	5	6	7	8	9	10
20	13	17	16	18	12	11	15	12	10

Lezione 7: Cancellazione dallo Heap (esempio)

Supponiamo di voler cancellare un valore dallo Heap.



1	2	3	4	5	6	7	8	9	10
20	18	17	16	13	12	11	15	12	10

Esercizio 7: Pronto Soccorso

Esercizio 7: Gestione di una Coda con priorità

- Si vuole simulare la gestione di una **coda di attesa al pronto soccorso**, gestendo un insieme di pazienti in attesa come una **coda con priorità**;
- la priorità è data dai seguenti fattori:
  - codice colore (bianco < verde < giallo < rosso);
  - ordine di arrivo (1 > 2 > 3 > 4 ... > n).
- Ad esempio, un paziente Pinco Pallino con **codice rosso e ordine di arrivo 2** ha priorità **superiore** rispetto al paziente John Doe con **codice verde e ordine di arrivo 1**.

Esercizio 7: Strutture e funzioni

Rappresentiamo il **Pronto Soccorso** con la seguente struttura:

```
typedef struct
{
    int numeroB;
    int numeroG;
    int numeroV;
    int numeroR;
} ProntoSoccorso;
```

I campi **numeroB**, **numeroG**, **numeroV** e **numeroR** rappresentano il numero di pazienti con i relativi codici (bianco, giallo, verde e rosso) inseriti in coda durante il giorno. Servono per tenere traccia dell'ordine d'arrivo dei vari pazienti (**non li dovete gestire voi**). Tali campi sono usati dalla funzione *nuovoPaziente(...)* per attribuire al paziente il relativo ordine di arrivo (che userete poi, insieme col colore, per stabilire la priorità).

I **codici** del Pronto Soccorso sono rappresentati da un'enumerazione così definita:

```
typedef enum {BIANCO, VERDE, GIALLO, ROSSO} Codice;
```

Esercizio 7: Strutture e funzioni

Rappresentiamo un **Paziente** con la seguente struttura:

```
typedef struct
{
     char nome[DIM_NOME];
     Codice codice;
     int ordine;
}Paziente;
```

I campi **codice** e **ordine** devono essere utilizzati per stabilire la priorità del paziente (come spiegato in precedenza).

La **Coda con Priorità (coda del pronto soccorso)** viene rappresentata mediante uno Heap (dichiarato nel main), strutturato come un **array di Pazienti**.

```
Paziente codaPrio[DIM_HEAP];
int dim_coda = 0;
```

Esercizio 7: progetto base

- Scaricate da eLearning il progetto base con il codice di partenza per l'esercitazione odierna, e copiatelo in un nuovo progetto Clion.
- Il file comprende le seguenti strutture:
  - Pronto Soccorso
  - Paziente
  - Codice (enumerazione)

- Il file comprende la funzione che **crea** e restituisce un nuovo paziente (chiedendo i dati da terminale all'utente). Tale funzione restituisce un elemento di tipo Paziente (struttura) pronto per essere inserito nella coda con priorità nella posizione data dalla combinazione del suo colore e del suo ordine di arrivo.

```
algoritmo nuovoPaziente(puntatore a ProntoSoccorso ps) → Paziente
```

- **NON** modificate le funzioni e le strutture già presenti!

Esercizio 7: Inserimento e cancellazione

Si richiede:

1. l'inserimento di nuovi pazienti
2. la visualizzazione in ordine posizionale di tutti i pazienti con i relativi dati (ordine in cui sono memorizzati nella coda con priorità: prima la posizione 1, poi la 2, e così via…); *NB: questa non è la posizione data dalla priorità*
3. la possibilità di eliminare un elemento dalla coda (ovvero, ogni volta, l'elemento con priorità massima della coda d'attesa) e di restituirlo.

Dal momento che il confronto tra due pazienti è basato su due possibili parametri (codice colore e ordine d'arrivo) è necessario implementare una funzione che, dati due pazienti, stabilisca quale dei due ha priorità maggiore (utile nella *insertHeap* e nella *deleteHeap*). Tale funzione restituisce un intero come segue:

- 1 se il paziente p1 ha priorità maggiore del paziente p2;
- 2 se il paziente p2 ha priorità maggiore del paziente p1.

```
algoritmo priorita(Paziente p1, Paziente p2) → intero
```

Esercizio 7: Confronto tra pazienti (esempio)

P1

nome: Lello
codice: verde
ordine: 2

priorita(p1, p2) = 1

P2

nome: Sonia
codice: verde
ordine: 3

Esercizio 7: Confronto tra pazienti (esempio)

P1

nome: Paolo
codice: giallo
ordine: 1

priorita(p1, p2) = 2

P2

nome: Sonia
codice: rosso
ordine: 3

Esercizio 7: Ricapitolando

Realizzare una coda con priorità per gestire i pazienti di un Pronto Soccorso.

Devono essere presenti le seguenti funzioni:

- **nuovoPaziente** (per acquisire i dati relativi ad un nuovo paziente)
  - NB: questa funzione è già implementata.
- **priorità** (per confrontare le priorità tra pazienti)
- **insertHeap** (per inserire un nuovo paziente nella coda)
- **deleteHeap** (per rimuovere il paziente con priorià massima dalla coda)
- **printHeap** (per stampare lo stato della coda, in ordine posizionale)

end().