

# Algoritmi e Strutture Dati

## Lezione 11



[www.iet.unipi.it/a.virdis](http://www.iet.unipi.it/a.virdis)

Antonio Virdis

[antonio.virdis@unipi.it](mailto:antonio.virdis@unipi.it)

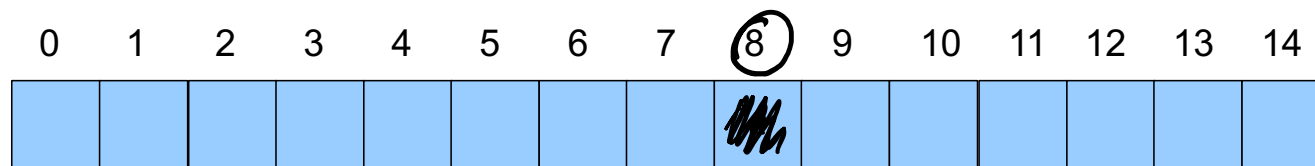


# Sommario

- Hashing
- Hashing e tipi di input 
- *Esome*
- Esercizi 

|

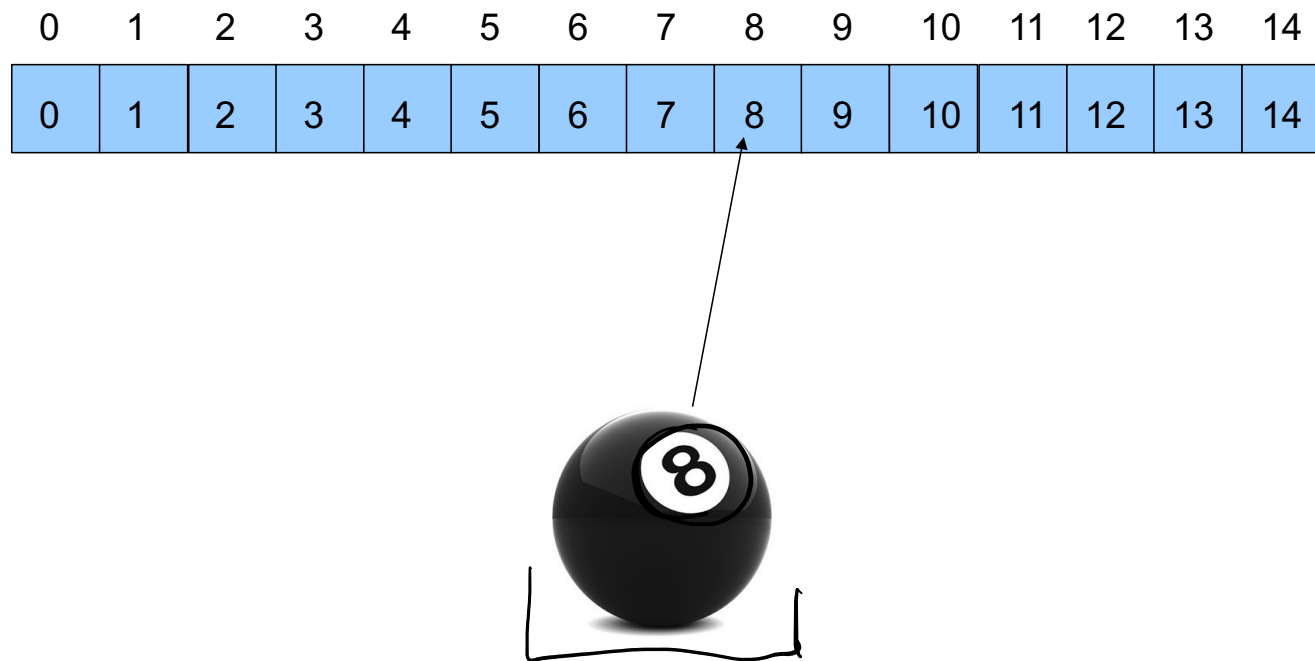
# Array ++



# Indirizzamento diretto



# Indirizzamento Diretto

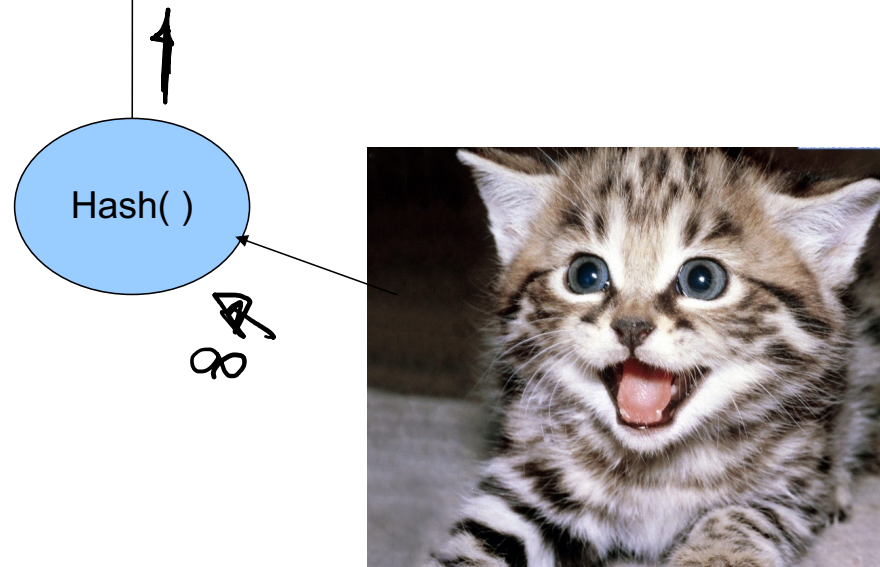


# HASHING



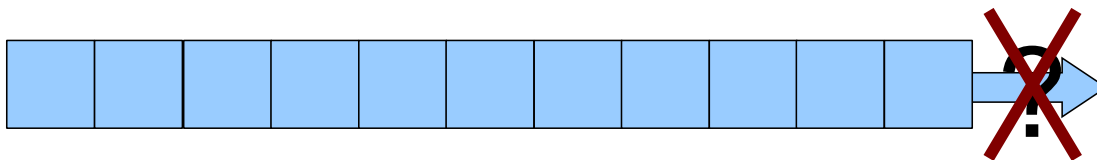
Antonio Virdis - 2022-23

# Hashing



# Strutture Dati

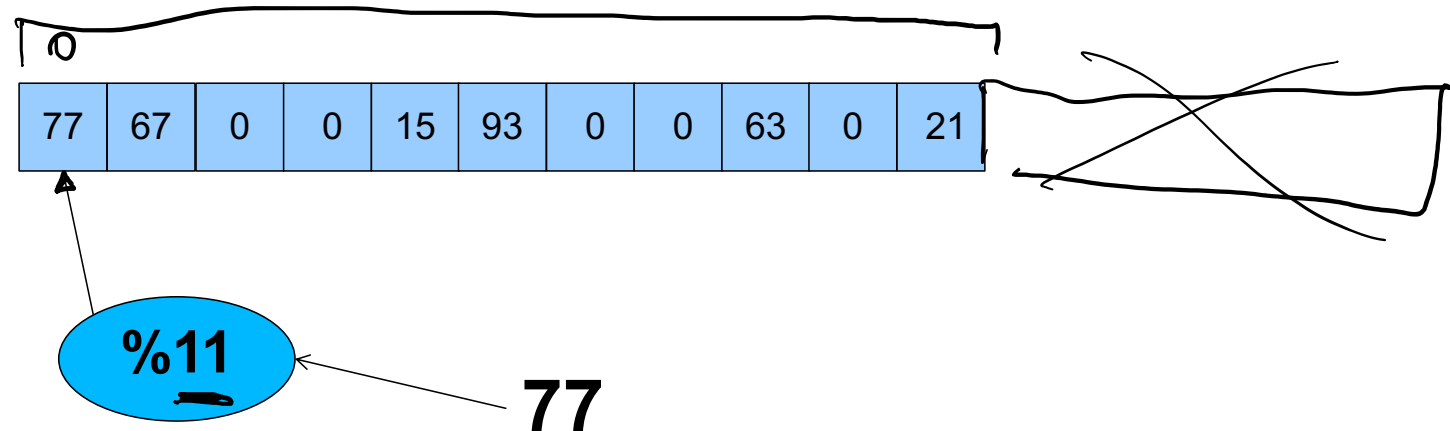
- Array
- Vector
- ...





# Simple Hash Table

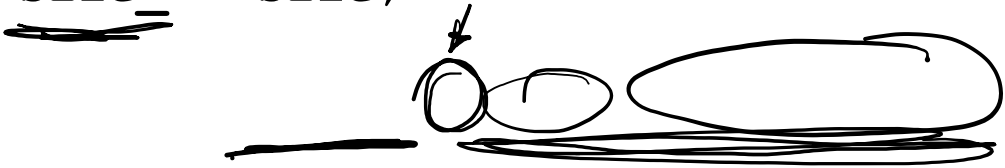
- Trattiamo interi  $> 0$
- Chiave coincide con valore
- La funzione HASH e' la funzione modulo
- Convenzione 0 per vuoto



# Class

```
1 class HashTable
2 {
3     int * table_;
4     int size_;
5
6
7 public:
8     HashTable( int size );
9
10    bool insert( int key );
11
12    void print();
13
14    int hash( int key );
15
16 };
17
18
```

# Costruttore

```
1  HashTable::HashTable( int size )
2  {
3      table_ = new int[size];
4      size_ = size;
5      
6  }
7
8  memset( address , value , size );
9
10
11
12
13
14
15
16
17
18
```

# Hashing

```
1  int HashTable::hash( int key )
2  {
3
4      return key % size_;
5
6  }
```

# Insert

```
1 bool HashTable::insert( int key )
2 {
3     // trova indice tramite hashing
4
5
6
7
8
9
10
11
12
13
14
15
16
17 }
18
```

# Insert

```
1  bool HashTable::insert( int key )
2  {
3      int index = hash(key) ;
4
5
6
7
8
9
10
11
12
13
14 }
15
```

# Insert

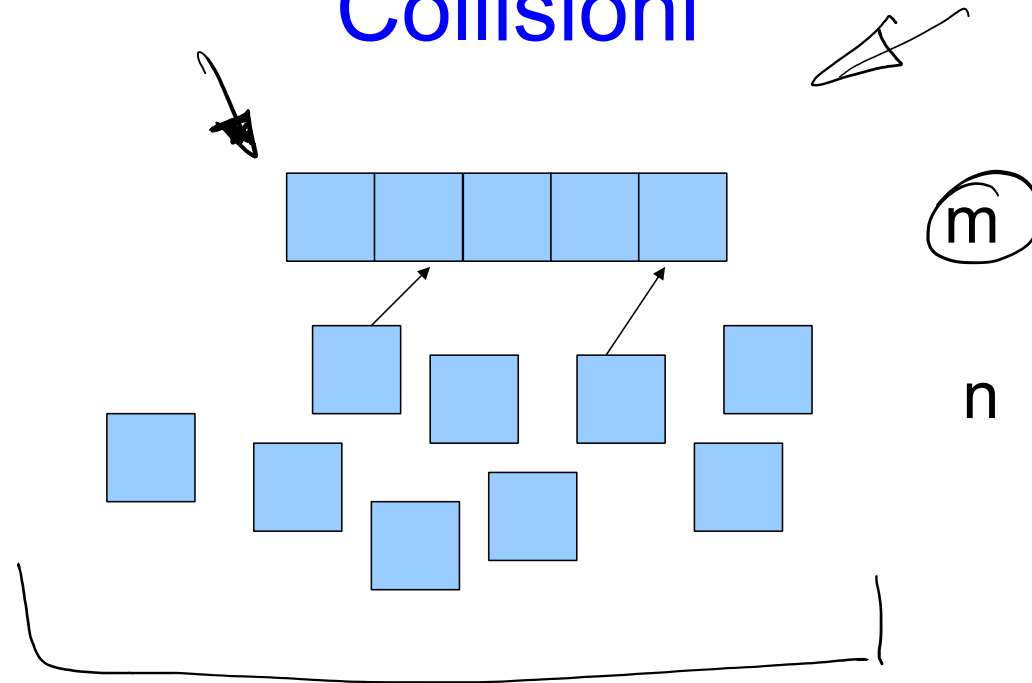
```
1  bool HashTable::insert( int key )
2  {
3      int index = hash(key);
4
5
6
7
8
9
10  → table_[index] = key;
11
12  ↵ cout << "key stored" << endl;
13  return true;
14  }
15
```

# Insert

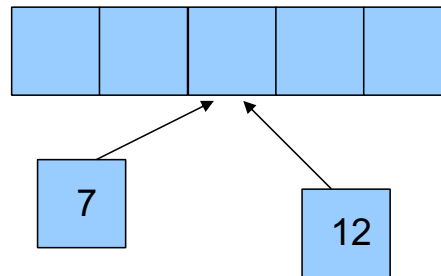
```
1  bool HashTable::insert( int key )
2  {
3      int index = hash(key);
4
5      ➔ if( table_[index] != 0 )
6      {
7          cout << "already occupied" << endl;
8          return false;
9      }
10     table_[index] = key;
11
12     cout << "key stored" << endl;
13     return true;
14 }
15
```



# Collisioni



# Collisioni



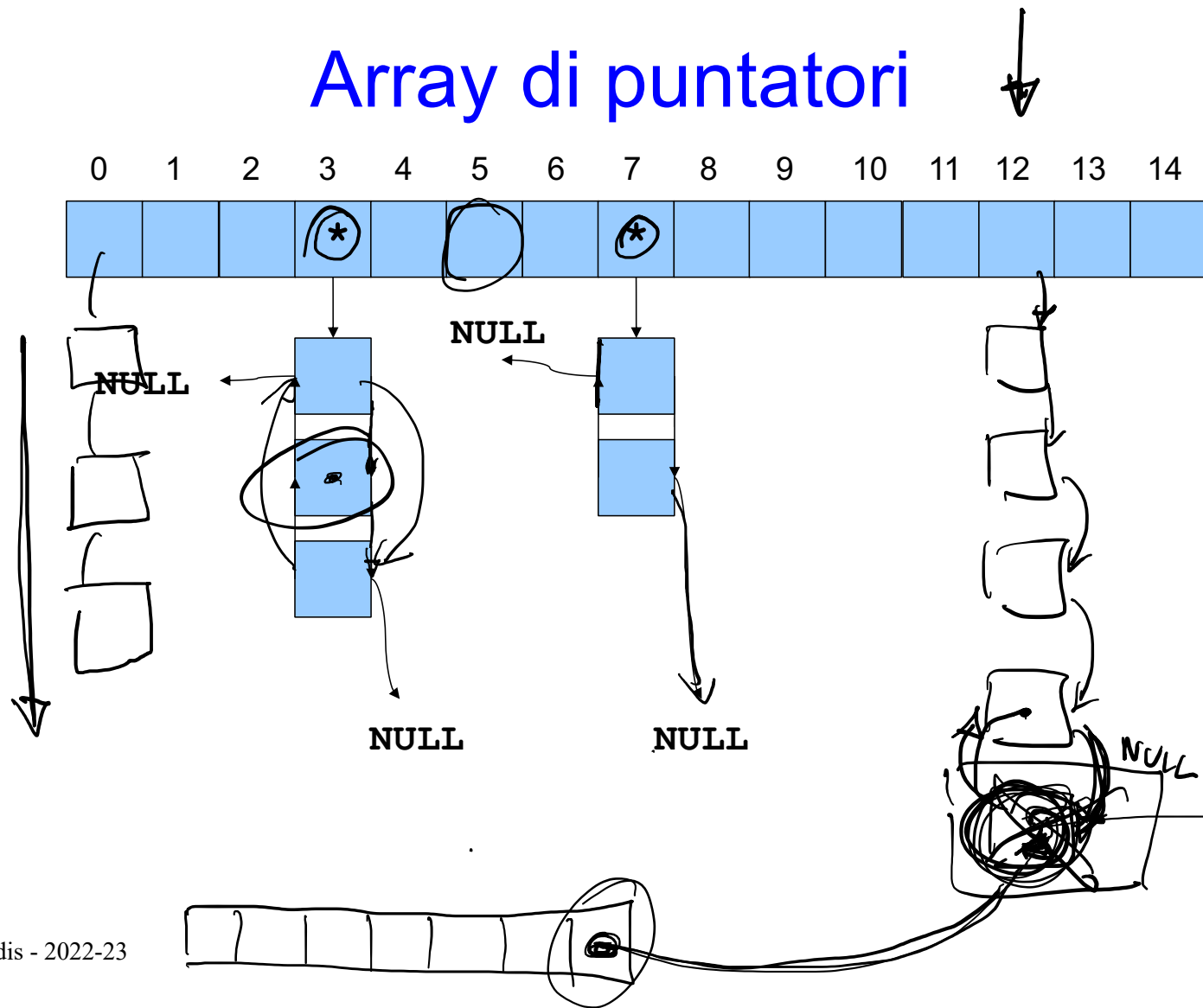
m

$\nearrow$   
 $\nwarrow$   
n

- Liste di trabocco
- Indirizzamento aperto

# Array di puntatori

m



# Elem

```
1 struct Elem
2 {
3     int key;
4     Elem * next;
5     Elem * prev;
6
7     Elem(): next(NULL) , prev(NULL) {}
8 };
9
10
11
12
13
14
15
16
17
18
```

# Hash con trabocco

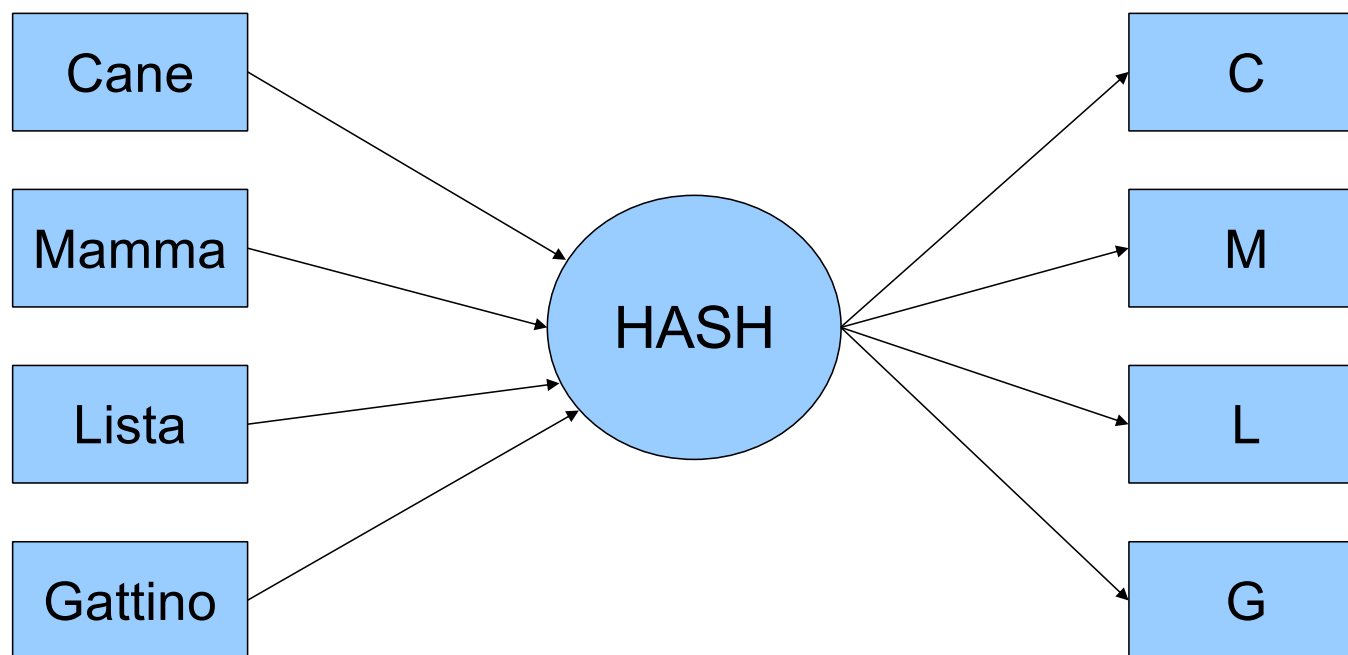
```
1 class HashTable
2 {
3     Elem** table_;
4     int size_;
5
6 public:
7
8
9
10
11
12
13
14
15
16
17
18 };
```

The diagram illustrates a hash table with 4 slots. The first slot contains a pointer to a vertical list of 5 elements, representing a linked list. The other three slots are empty. Arrows indicate the mapping from the table slots to the elements in the list.

# Implementazione

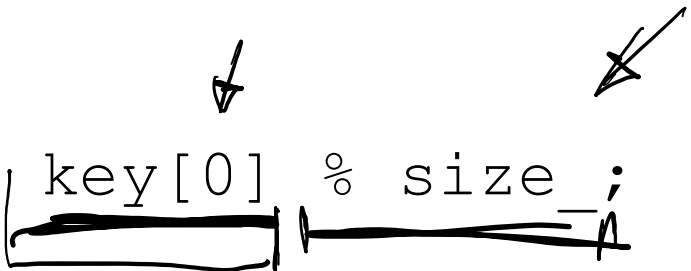
- Insert / Print / Find
- Stiamo trattando liste I
- Facciamo inserimento in testa I

# Hashing Stringhe



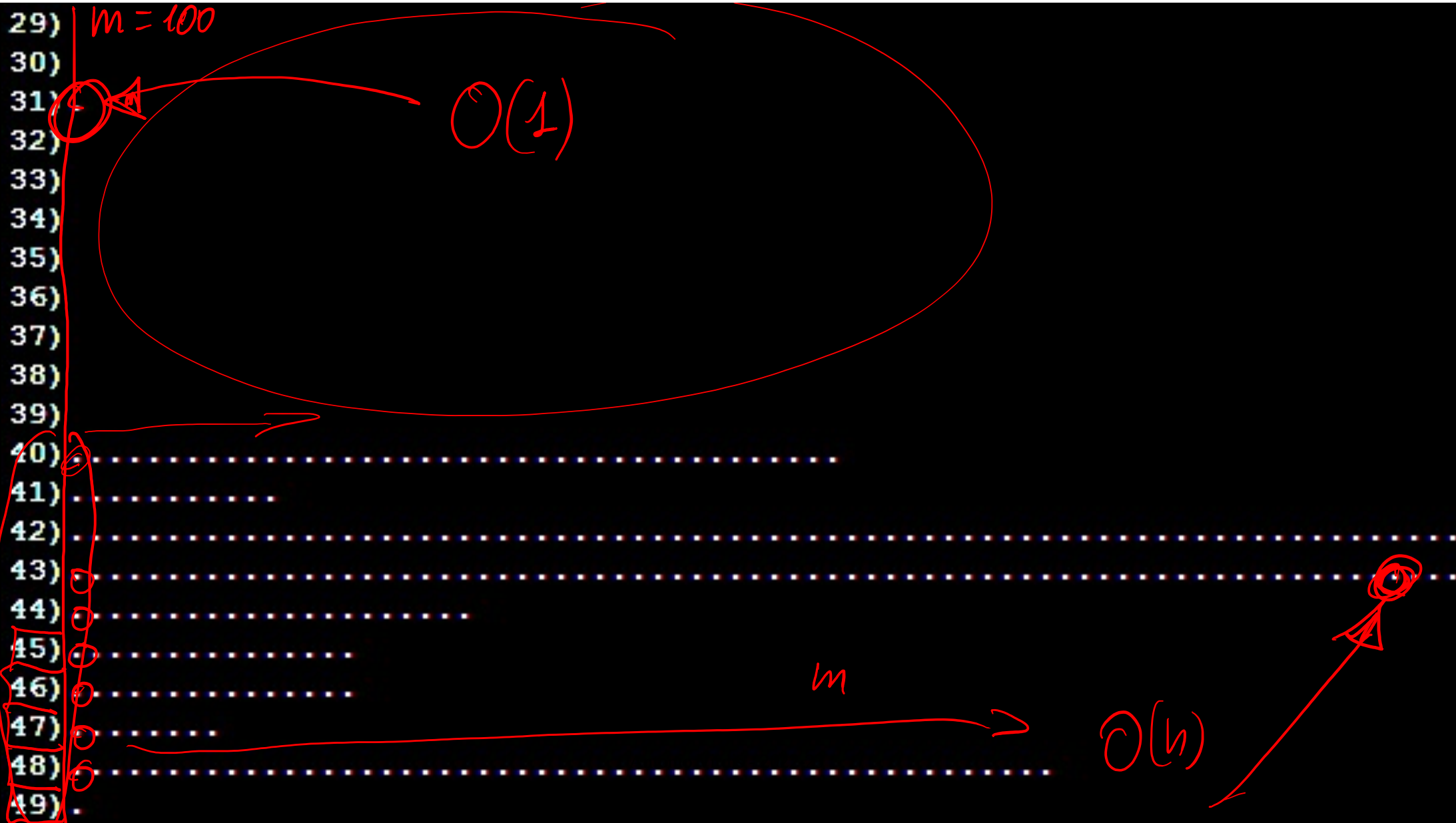
# Prima lettera

```
int hash(string key)
{
    int index = key[0] % size;
}
```



?





# Somma caratteri

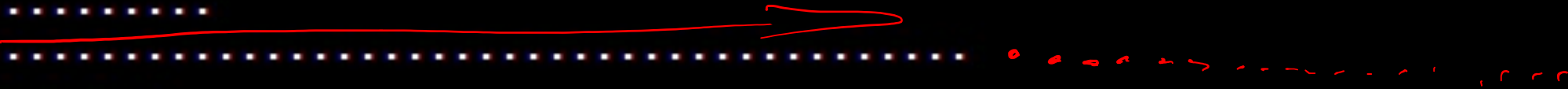
magh()

index = 0

```
for( int i = 0 ; i < key.length() ; ++i )  
{  
    index = ( index + key[i] ) % size_ ;  
}
```

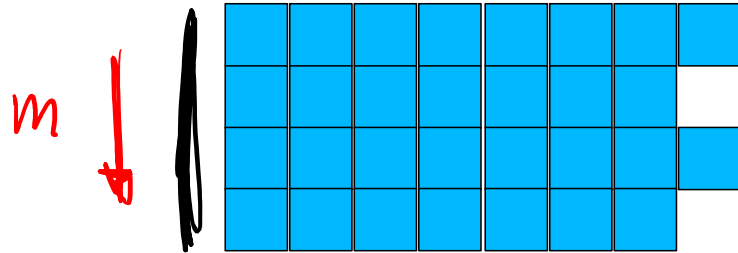
?

28) .....  
29) .....  
30) .....  
31) .....  
32) .....  
33) .....  
34) .....  
35) .....  
36) ...  
37) .....  
38) .....  
39) .....  
40) .....  
41) .....  
42) .....  
43) .....  
44) .....  
45) .....  
46) .....  
47) .....  
48) .....  
49) .....



# Good HASH

- Dipende fortemente dal tipo di applicazione
- Per applicazioni di indexing e' fondamentale l'uniformità



- Lavorano sulla **rappresentazione binaria**
- E.g. MurmurHash,  
CityHash, FarmHash ...



## std::map

```
std::map <key_T, obj_t> table;
```

Tipo chiave

Tipo dati

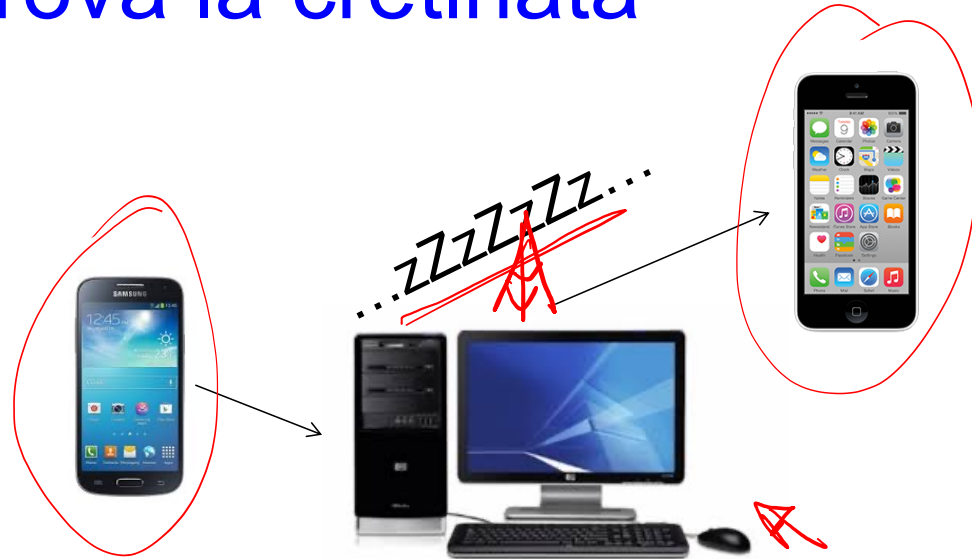
```
table['uno'] = "valore uno";  
table.find('uno');
```

# Open Addressing

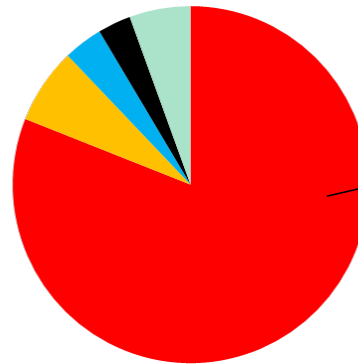


# Quiz: trova la cretinata

- Software di emulazione



valgrind



pow(x,y)

# Trova la cretinata

value =

$v_1 * \text{pow}(x, 6) + v_2 * \text{pow}(x, 5) + v_3 * \text{pow}(x, 4) \dots;$

$$v_1 \cdot x^6 + v_2 \cdot x^5 + v_3 \cdot x^4 + v_4 \cdot x^3 + v_5 \cdot x^2$$

$$x^2 = x \cdot x$$

$$x^3 = x \cdot x \cdot x = x \cdot x^2$$

$$x^4 = x^3 \cdot x$$






# Esercizio 1

- Sorting e tabelle hash
- Classifica videogame online
  - Salvo coppie <nome , punteggio>
- Interrogazioni
  - Sapere i primi K
  - Sapere posizione di Pippo



## Esercizio 2


- Motore di ricerca tematico:
  - Ogni sito ha un 
    - Tipo ( sport , news , musica ...)
    - Nome ( gazzetta.it , lercio.it , amucidimaria.it )
    - Dati accessori (#pagine, statistiche...)
    - Numero di accessi
  - Operazioni
    - Accesso ad un sito
    - Dato un tipo, ottenere il nome e i dati del sito con più accessi

# ESAME

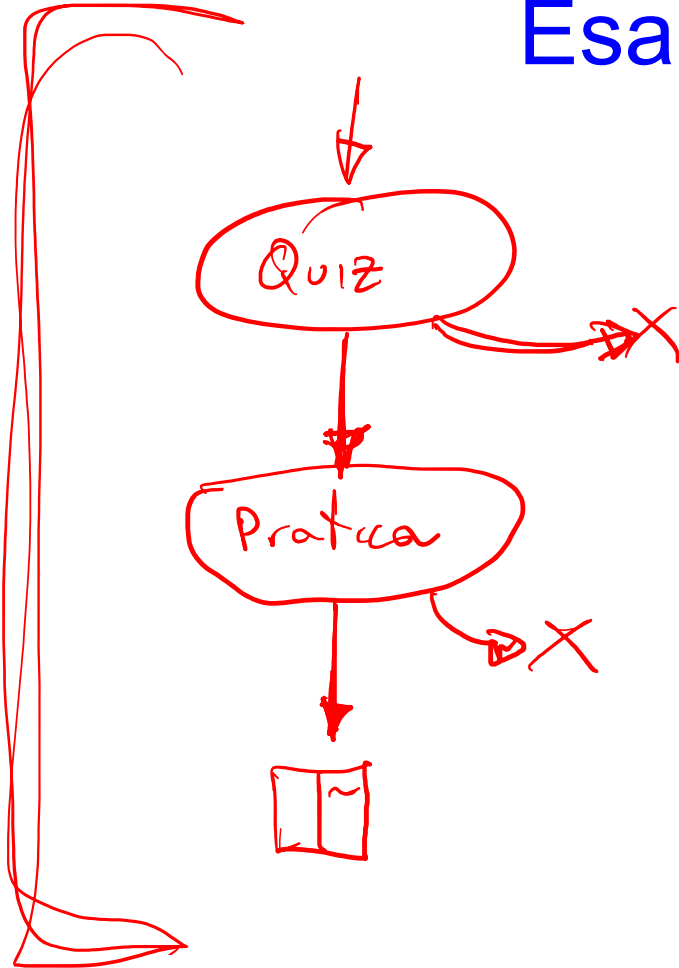
- Prova a quiz (al PC)
- Prova pratica (al PC)
- Iscrizioni chiudono 1 settimana prima della prova
- Nessuna iscrizione verrà accettata dopo la chiusura delle stesse



# Requisiti prova pratica

- Produrre un solo file cpp completo di main
- Il vostra programma deve compilare, eseguire e produrre l'output correttamente su tutti i casi di input
- Il vostra programma deve rispettare la complessità target 
- Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile.

## Esame: timeline




# Istruzioni Esame

- Risolvete gli esercizi prestando particolare attenzione:
  - alla formattazione dell'input e dell'output -
  - e alla complessità target indicata per ciascuna funzionalità. ✓
- E' possibile verificare la correttezza del vostro programma utilizzando i file di input/output contenuti nel file zip. !
- Per effettuare i test dovrete utilizzare il comando del terminale per la redirectione dell'input.

# Istruzioni Esame

- Esempio

  
./compilato <input0.txt |

- Dovete aspettarvi che l'output coincida con quello contenuto nel file output0.txt. Per effettuare un controllo automatico sul primo file input0.txt potete eseguire la sequenza di comandi

./compilato < input0.txt | diff - output0.txt 

- Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto

# Regole prova pratica

- Potete tenere materiale didattico
- NON potete collaborare
- NO domande su come compilare/eseguire/fare debug sul codice
- È vostra responsabilità salvare con regolarità
- Consegna tassativa entro l'orario stabilito





# Altezza Figli

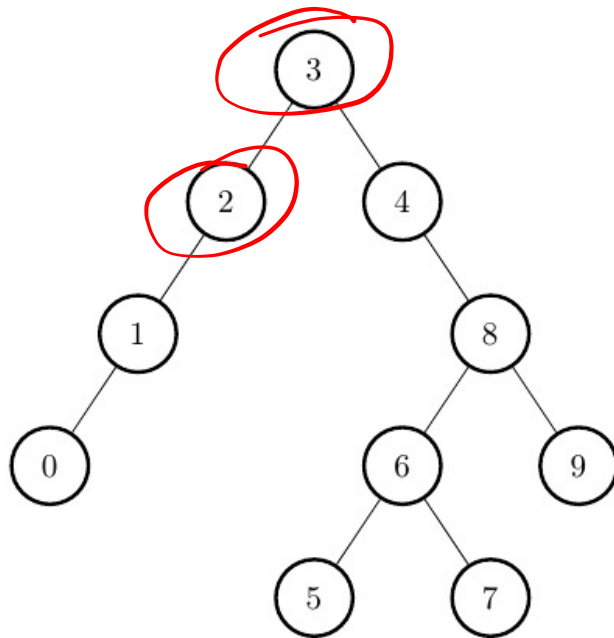
## Input:

- . N interi da inserire in un albero binario di ricerca

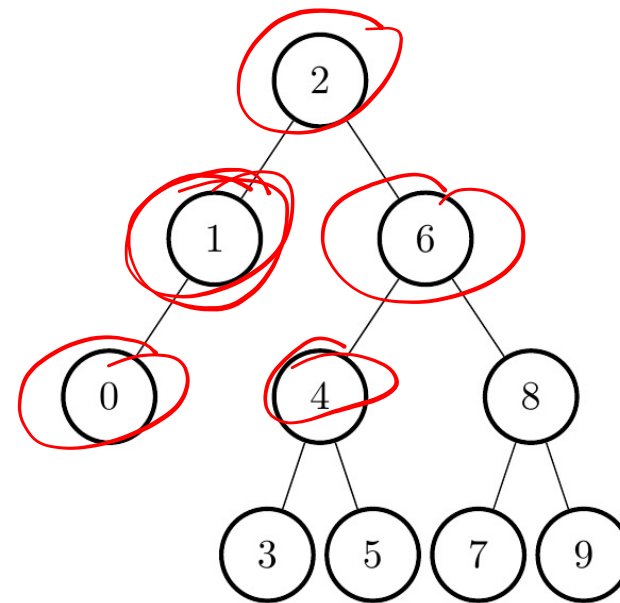
## Verificare che :

- . Per ciascun nodo, l'altezza dei suoi sottoalberi sinistro e destro deve differire al massimo di uno

# Esempi



no



si

```
1 bool wrongSol( Node * tree )
2 {
3     // Controllo se ho raggiunto una foglia
4     return true;
5
6
7
8
9
10
11
12
13
14
15
16
17
18 }
```

**Soluzione  
Sbagliata!**

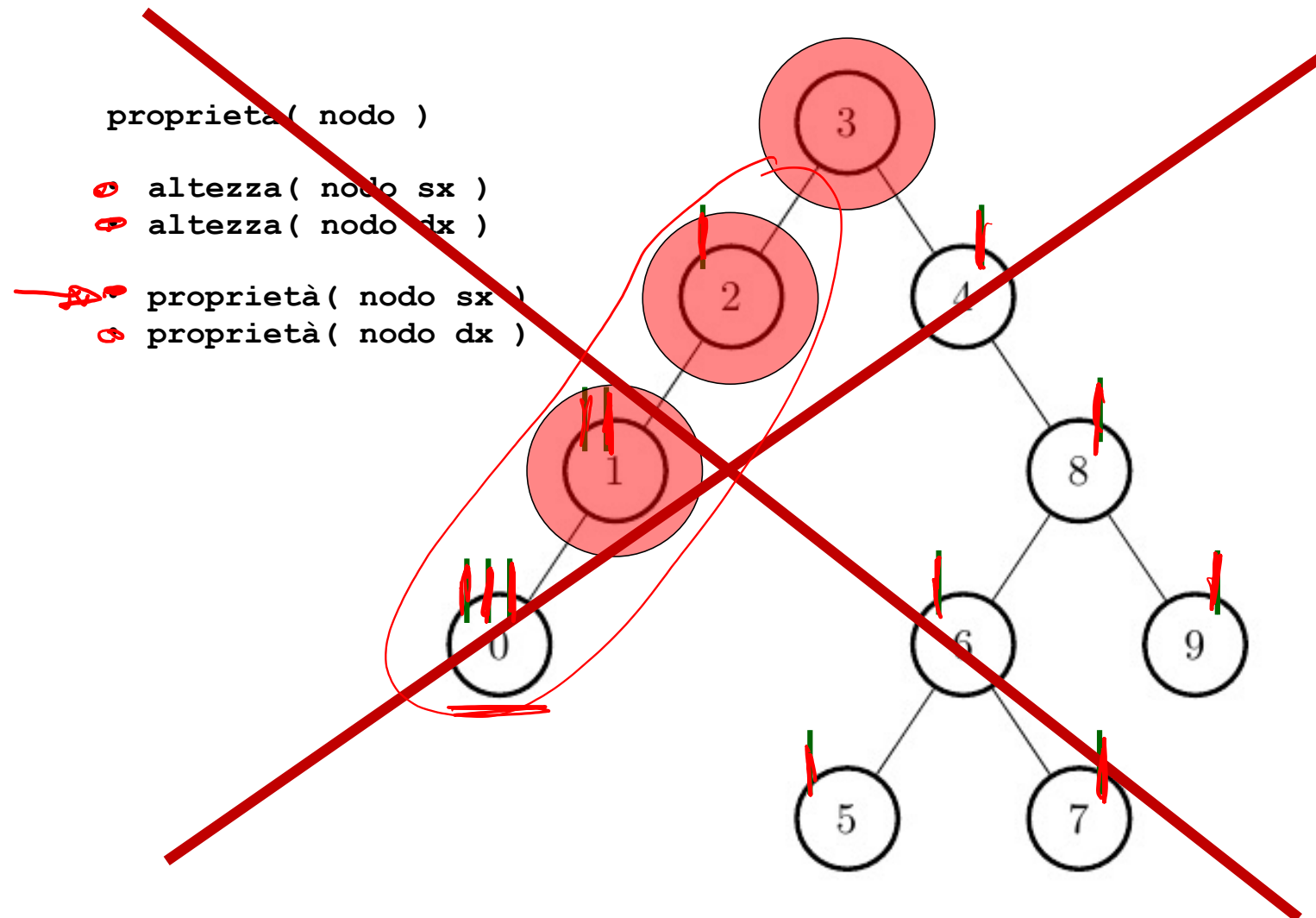


```
1 bool wrongSol( Node * tree )
2 {
3     int hl,hr;
4
5     // Controllo altezza dei figli sx e dx
6     hl = height(tree->left);
7     hr = height(tree->right);
8
9
10
11
12
13
14
15
16 }
17
18
```

Soluzione  
Sbagliata!

```
1 bool wrongSol( Node * tree )
2 {
3     int hl,hr;
4
5     // Controllo altezza dei figli sx e dx
6     hl = height(tree->left);
7     hr = height(tree->right);
8
9
10    // se la proprietà e' verificata da:
11    // nodo corrente, nodo sx, nodo dx
12    return true;
13 else
14     return false;
15
16 }
17
18
```

**Soluzione  
Sbagliata!**

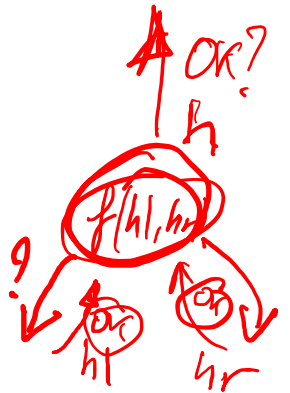


```
1  bool isOk( Node * tree, int & maxH )
2  {
3      // Controllo se ho raggiunto una foglia
4      ↪ return true;
5
6
7
8
9
10
11
12
13
14
15
16
17
18 }
```

```

1 bool isOk( Node * tree, int & maxH )
2 {
3     // Controllo se ho raggiunto una foglia
4     return true;
5
6     // Controllo i figli sinistro e destro
7     bool propL = isOk(tree->left, h1);
8     bool propR = isOk(tree->right, hr);
9
10
11
12
13
14
15
16
17
18 }

```





```
1 bool isOk( Node * tree, int & maxH )
2 {
3     // Controllo se ho raggiunto una foglia
4     return true;
5
6     // Controllo i figli sinistro e destro
7     bool propL = isOk(tree->left,hl);
8     bool propR = isOk(tree->right,hr);
9
10    // ottengo l'altezza del nodo corrente
11    // .. il massimo tra quella sx e dx
12
13
14
15
16
17
18 }
```

```
1 bool isOk( Node * tree, int & maxH )
2 {
3     // Controllo se ho raggiunto una foglia
4     return true;
5
6     // Controllo i figli sinistro e destro
7     bool propL = isOk(tree->left,hl);
8     bool propR = isOk(tree->right,hr);
9
10    // ottengo l'altezza del nodo corrente
11    // .. il massimo tra quella sx e dx
12
13    // se la proprietà e' verificata da:
14    // nodo corrente, nodo sx, nodo dx
15    return true;
16    else
17        return false;
18 }
```



# Esercizi

- Esperimenti
  - Test dimensione table
  - Test tipi di hash
  - Confronto map vs hash
- Esercizi
  - Implementare open addressing
  - Classifica videogame online
  - Motore di ricerca tematico

