Algoritmi e Strutture Dati

Lezione 5

www.iet.unipi.it/a.virdis

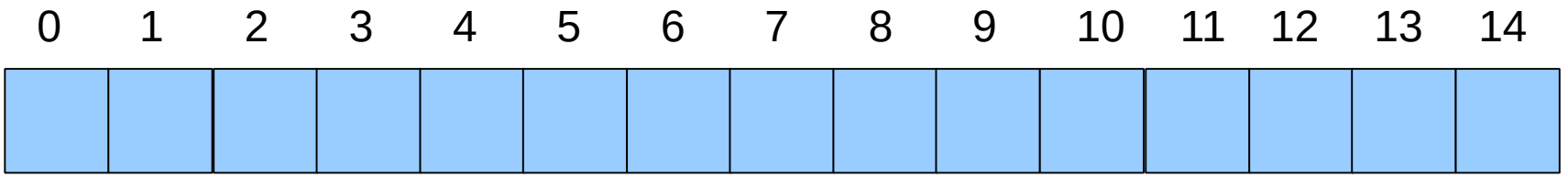
Antonio Virdis

antonio.virdis@unipi.it

Sommario

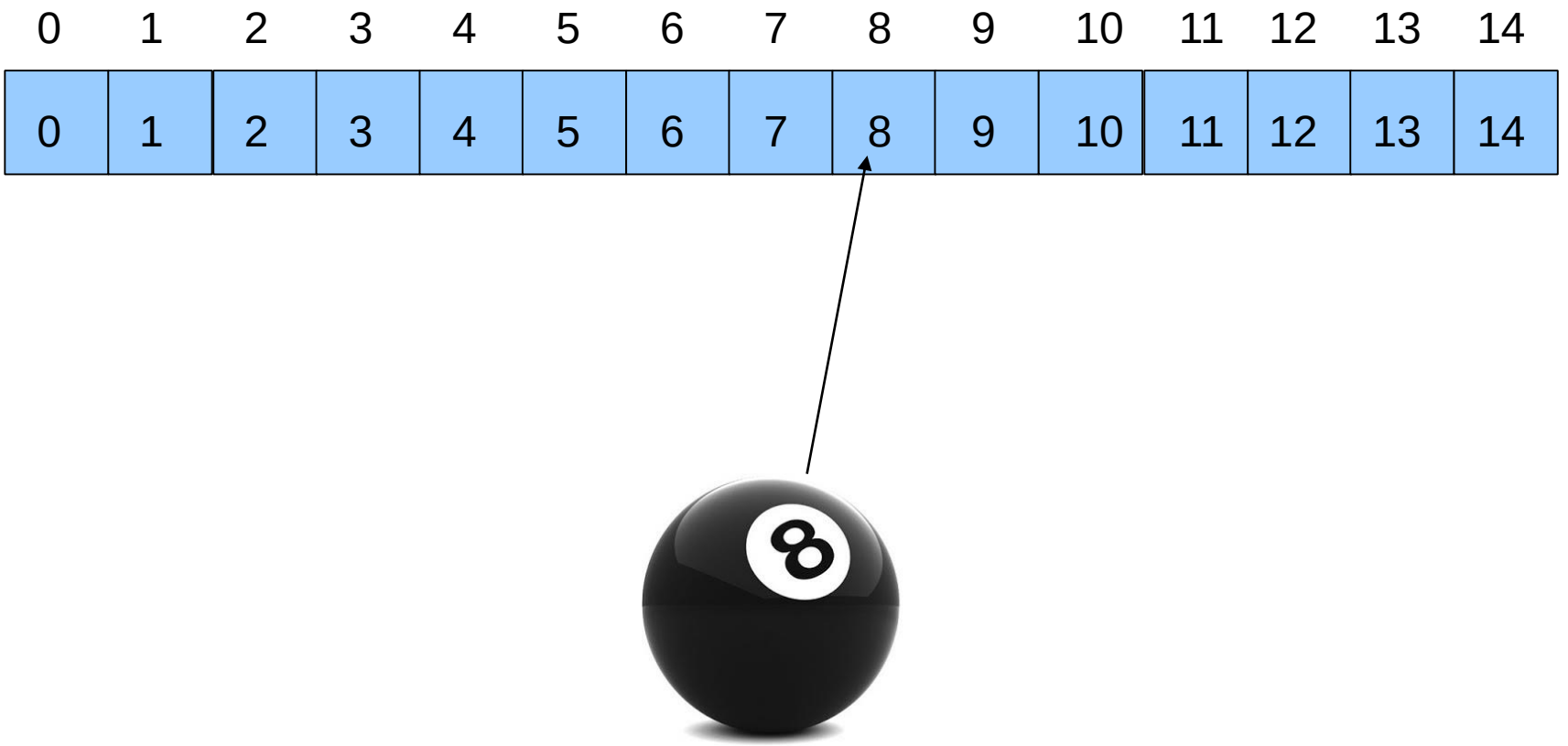
- Hashing
- Hashing e tipi di input
- Esercizi

Array ++



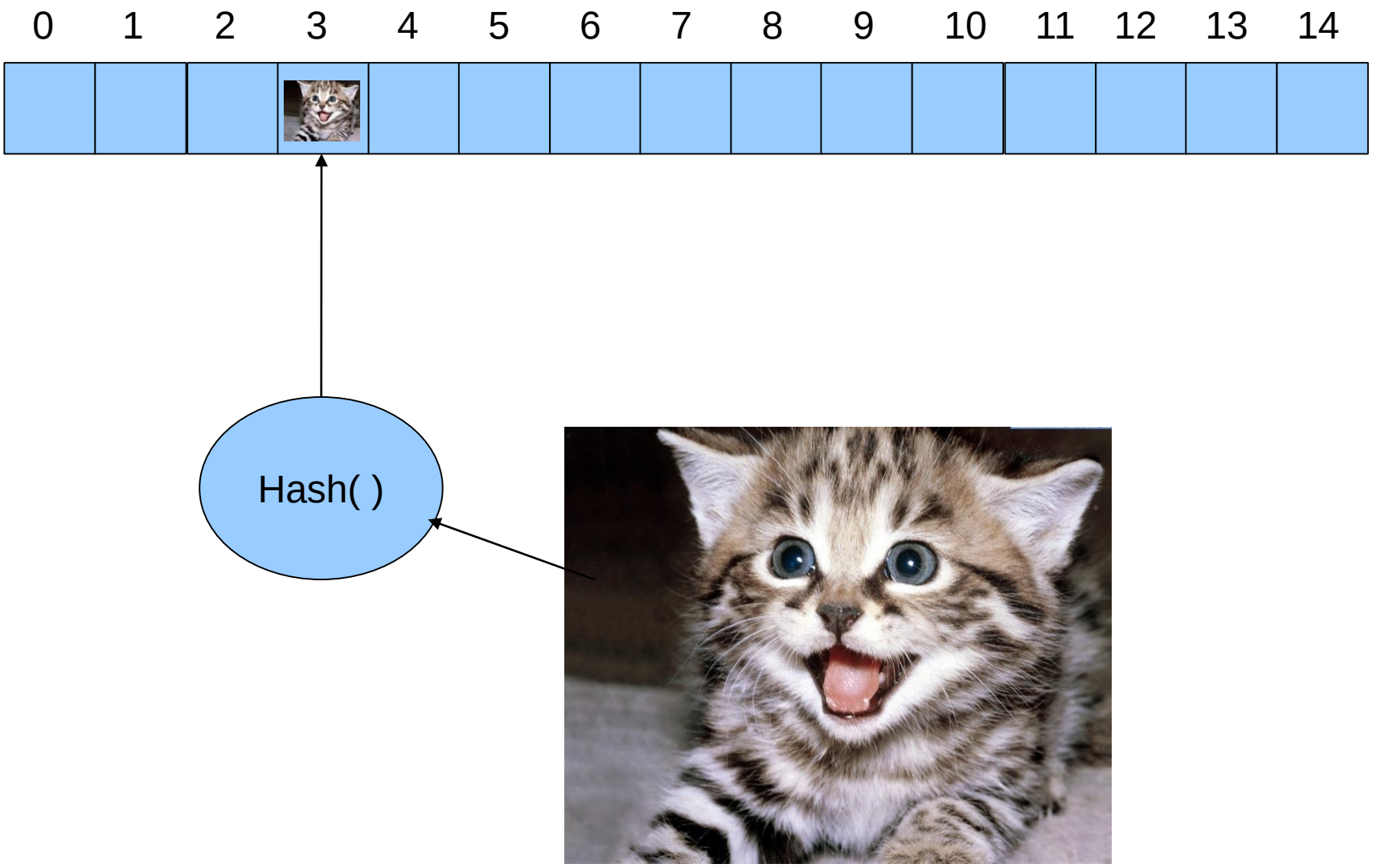
Indirizzamento diretto

Indirizzamento Diretto



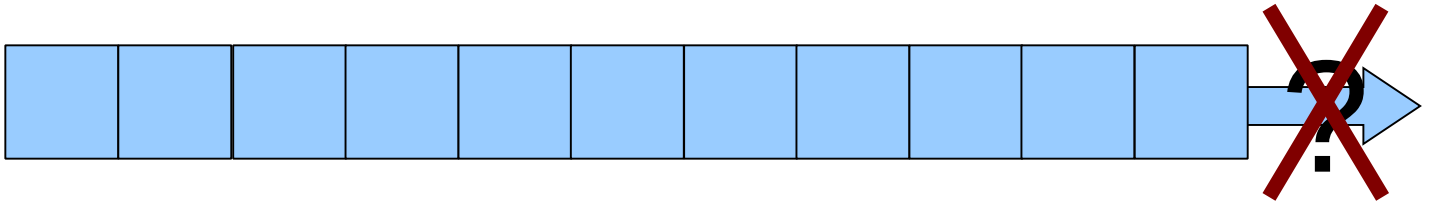
HASHING

Hashing



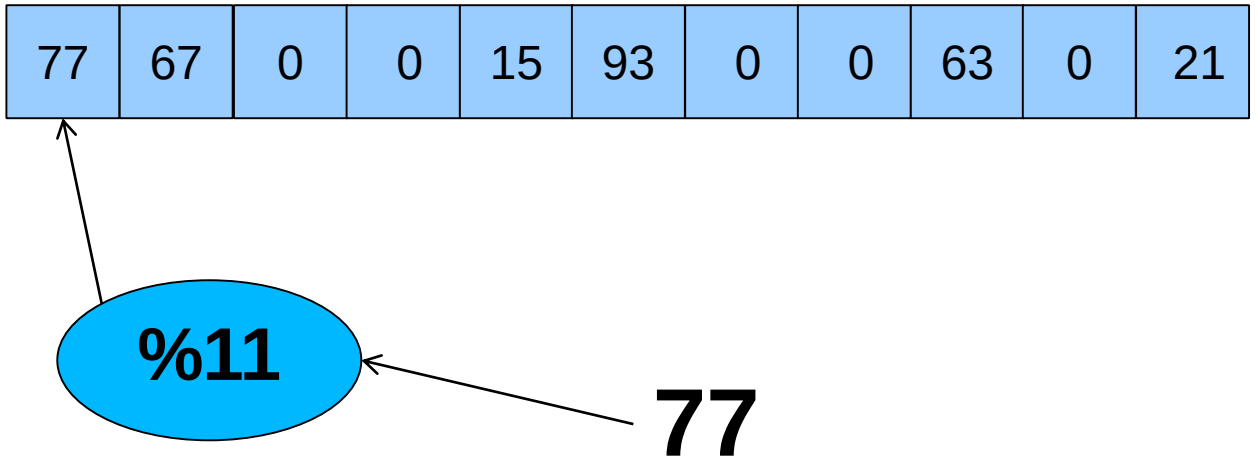
Strutture Dati

- 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   public:
7       HashTable( int size );
8
9       bool insert( int key );
10
11      void print();
12
13      int hash( int key );
14  };
15
16
17
18
```

Costruttore

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

Hashing

```
1   int HashTable::hash( int key )
2   {
3
4       return key % size_;
5
6   }
7
8
9
10
11
12
13
14
15
16
17
18
```

Insert

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

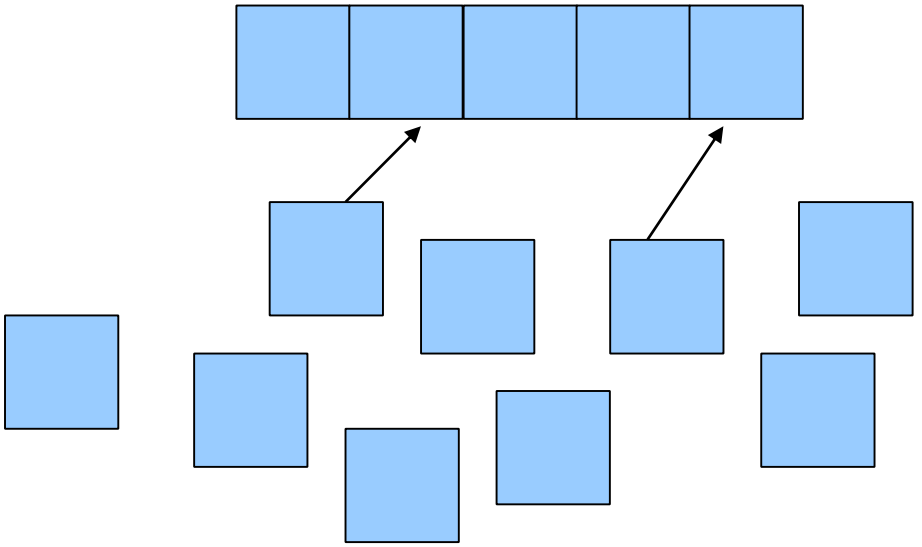
Insert

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

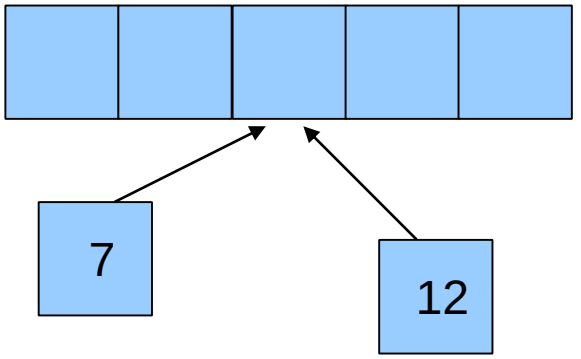
Insert

```
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

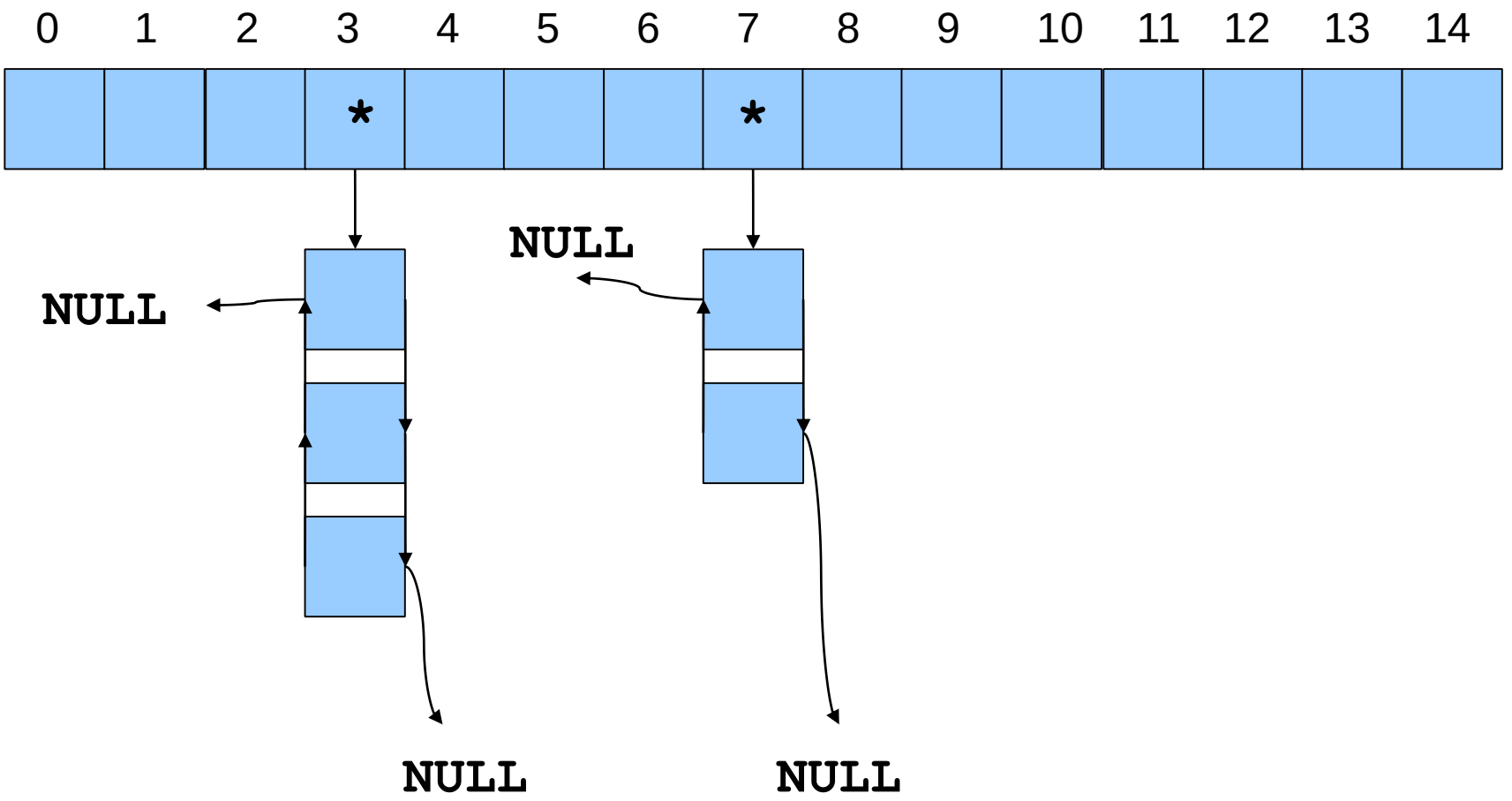


m

n

- Liste di trabocco
- Indirizzamento aperto

Array di puntatori

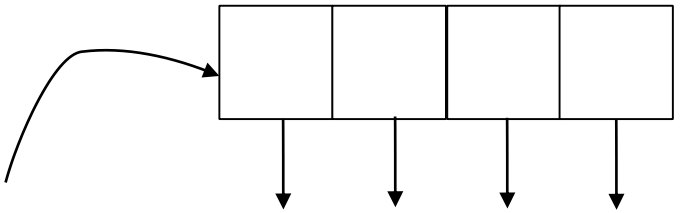


Elem

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

Hash con trabocco

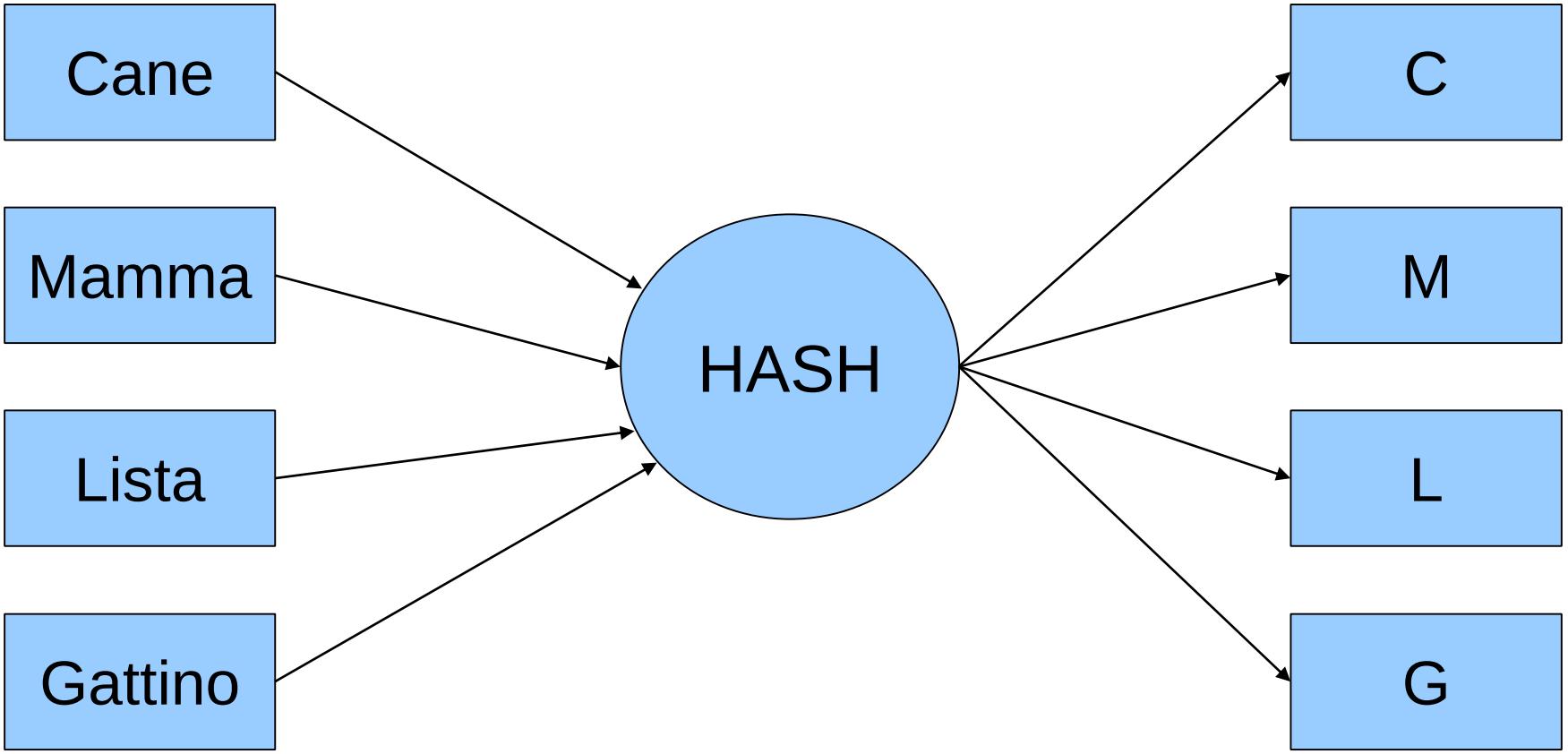
```
1   class HashTable
2   {
3       Elem * * table_;
4       int size_;
5
6   public:
7
8
9
10
11
12
13
14
15
16
17
18  };
```



Implementazione

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

Hashing Stringhe



Prima lettera

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

?

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

Somma caratteri

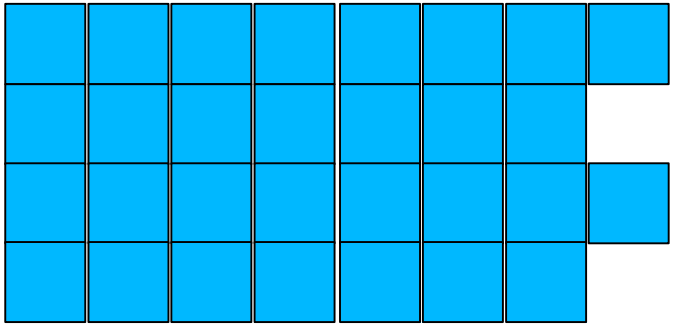
```
for( int i = 0 ; i < key.length() ; ++i )
{
  index = ( index + key[i] )% size_;
}
```

?

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 …

stl::map

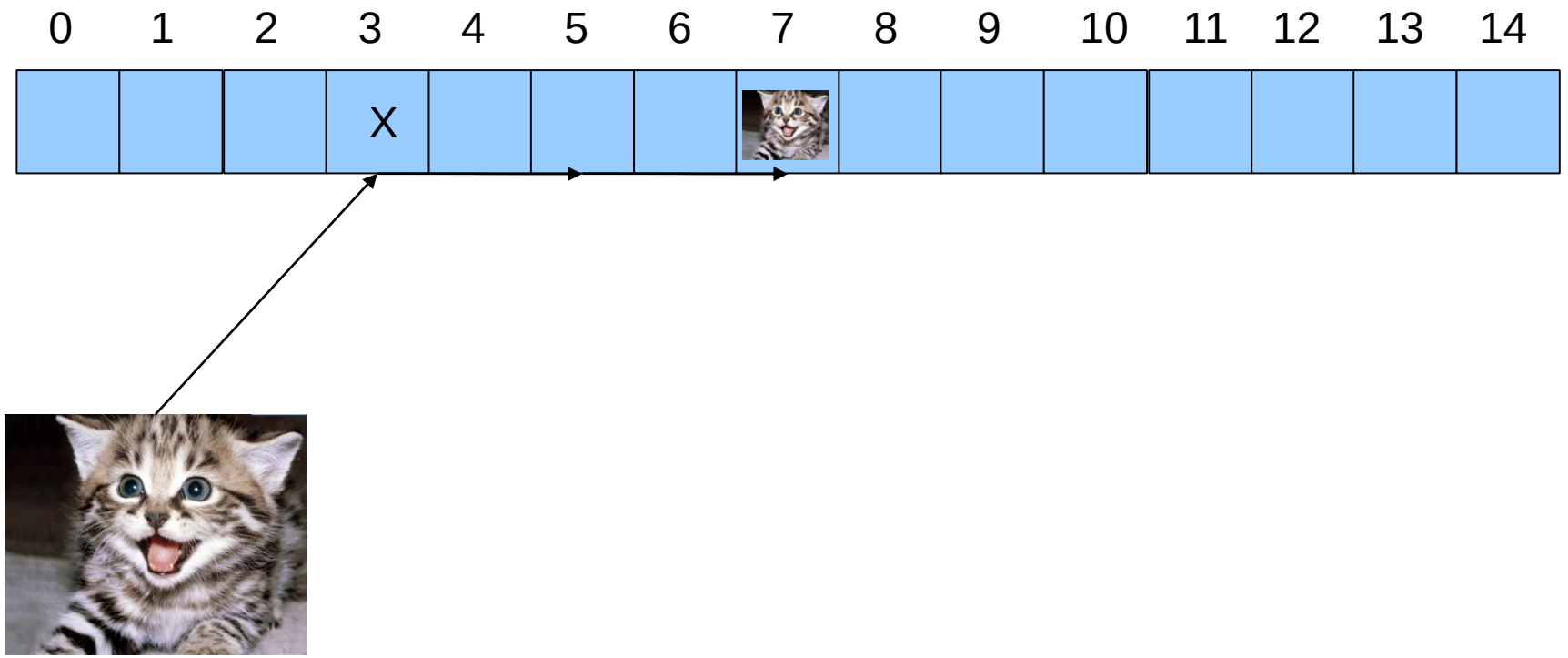
```
std::map < key_T , obj_t > table;
```

Tipo chiave **Tipo dati**

```
table['uno']="valore uno";

table.find('uno');
```

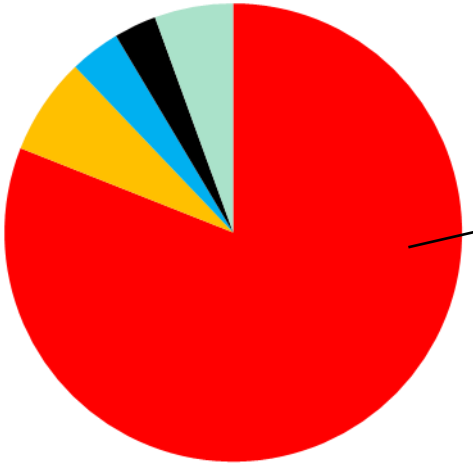
Open Addressing



Quiz: trova la cretinata

- Software di emulazione

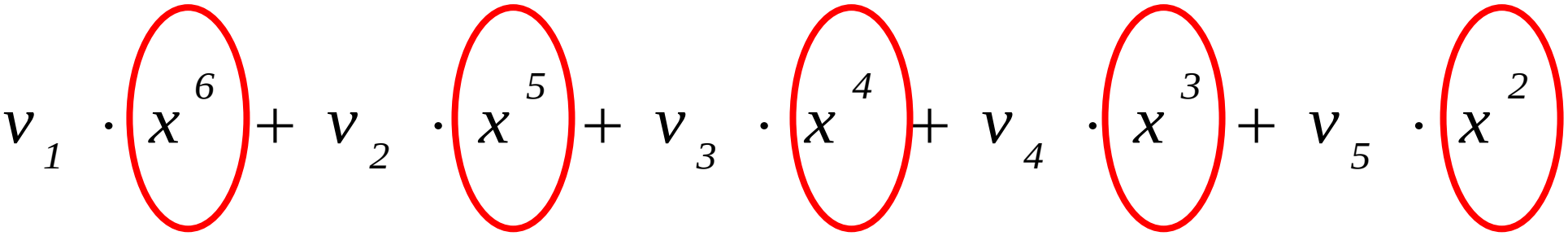
valgrind →



→ **pow(x,y)**

Trova la cretinata

```
value =
v1*pow(x,6) + v2*pow(x,5) + v3*pow(x,4)…;
```



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 , amicidimaria.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

Altezza Figli

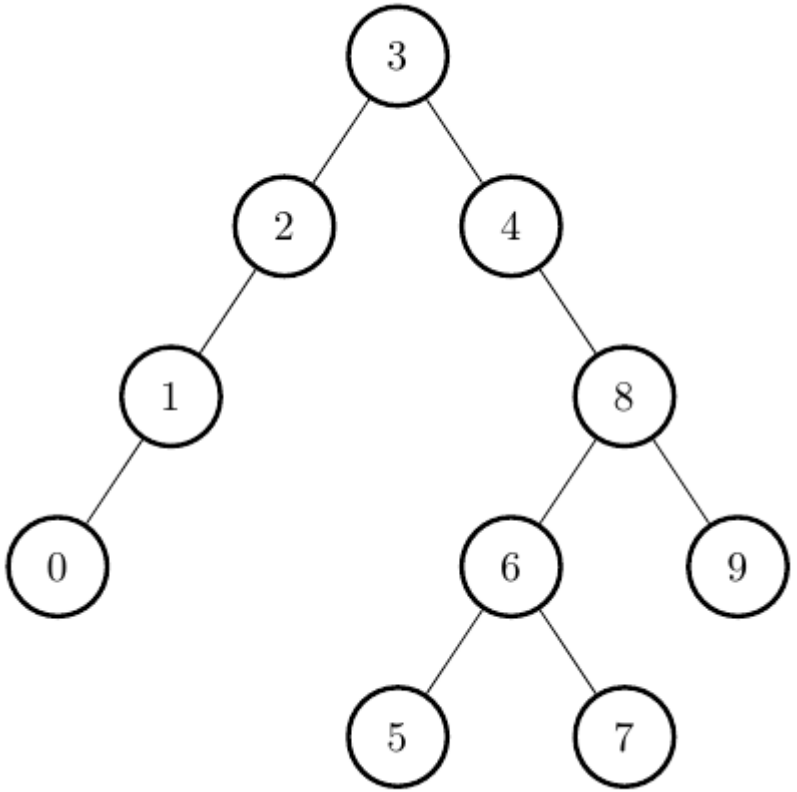
Input:

- N interi da inserire in un albero binario di ricerca

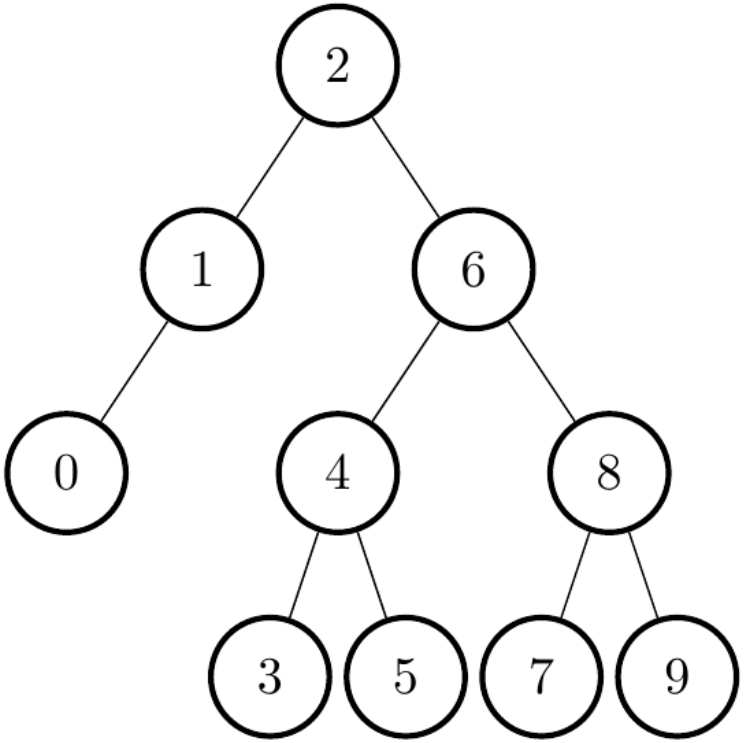
Verificare che :

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

Esempi



no



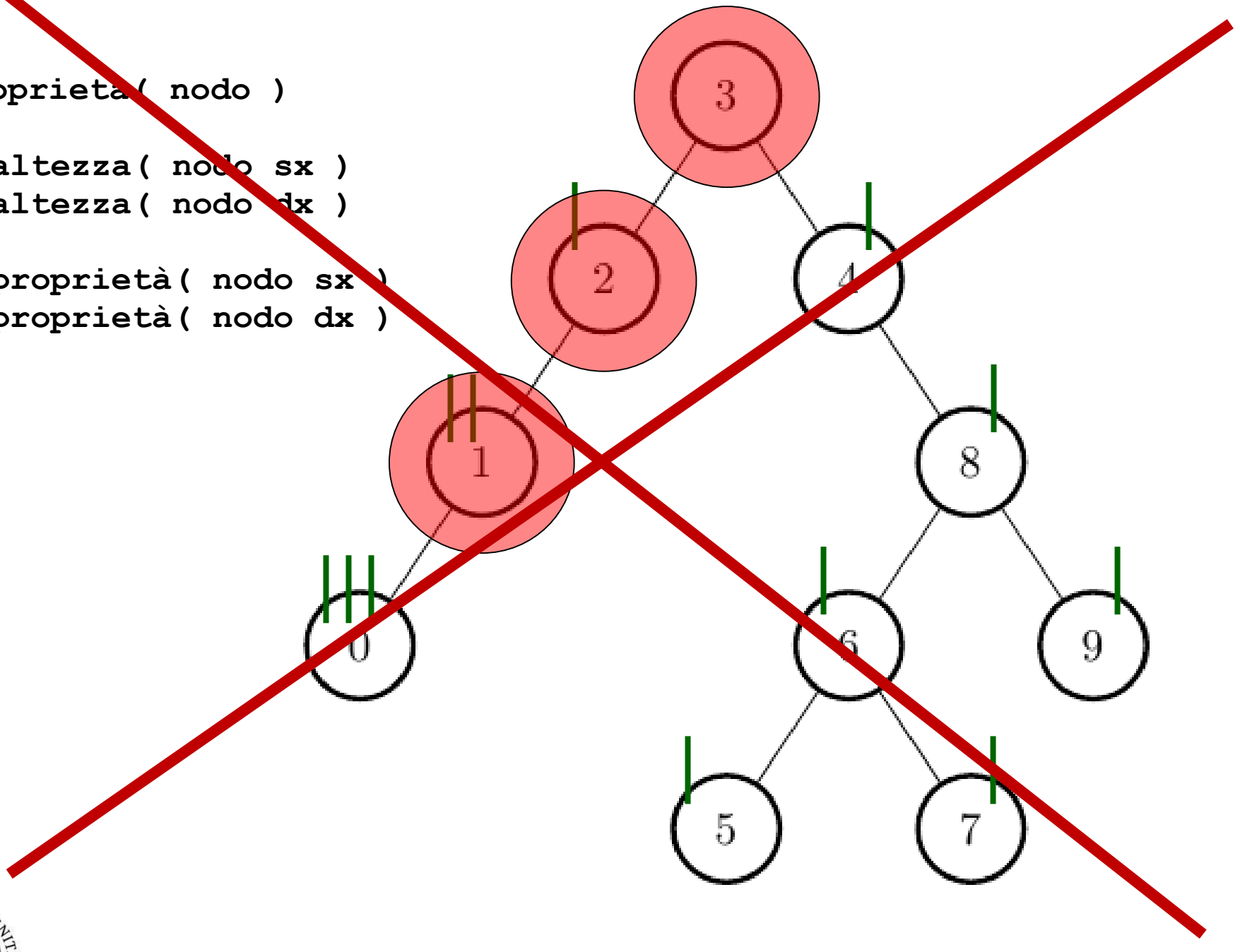
si

```
1   bool wrongSol( Node * tree )
2   {
3         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
```

proprieta(nodo)

- **altezza(nodo sx)**
- **altezza(nodo dx)**

- **proprietà(nodo sx)**
- **proprietà(nodo dx)**



```
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