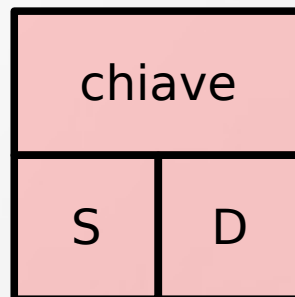


# Alberi binari di ricerca

In un **albero binario di ricerca** ogni **nodo**:

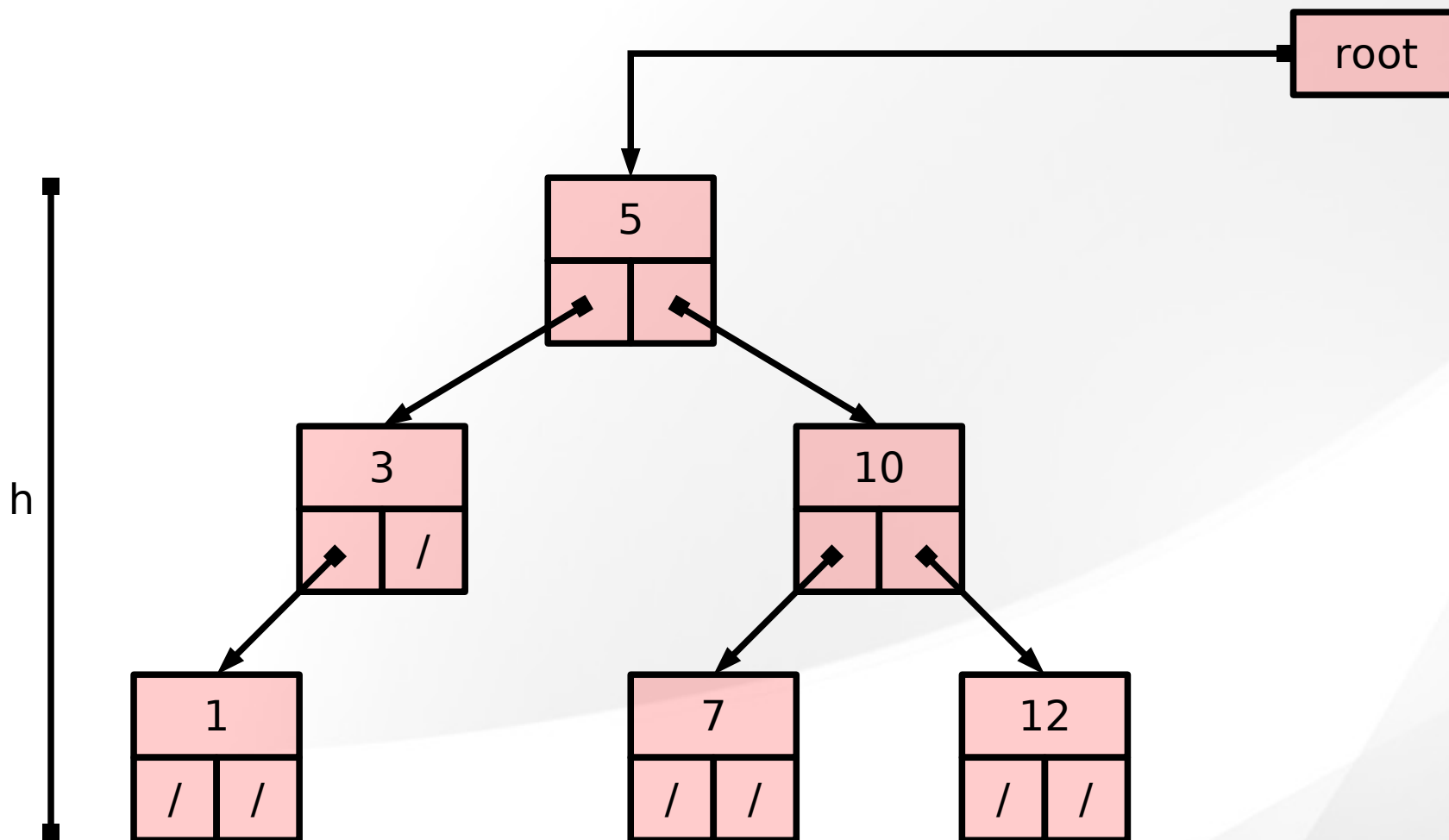
- ha associato un **valore**
- può avere al più due nodi figli (**sinistro** e **destro**)



In un albero binario di ricerca valgono le proprietà:

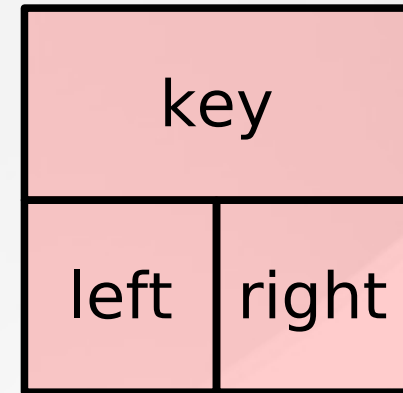
- I nodi del sottoalbero sinistro hanno un valore **minore o uguale** a quello della radice
- I nodi del sottoalbero destro hanno valori **maggiori** della radice

# Alberi binari di ricerca

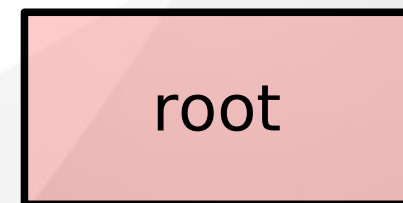


# Implementazione

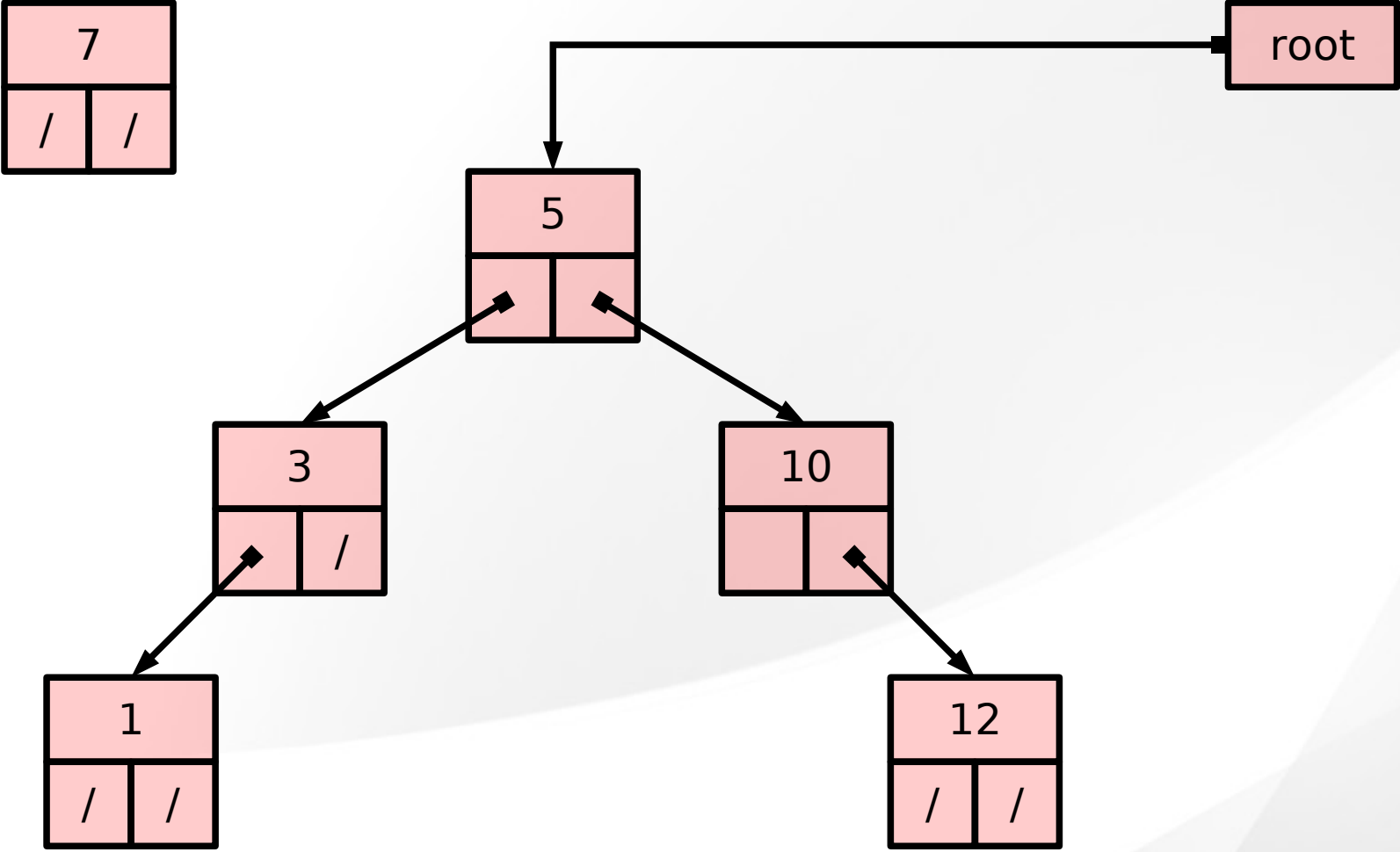
```
struct Nodo{  
    int key;  
    struct Nodo* left;  
    struct Nodo* right;  
};
```



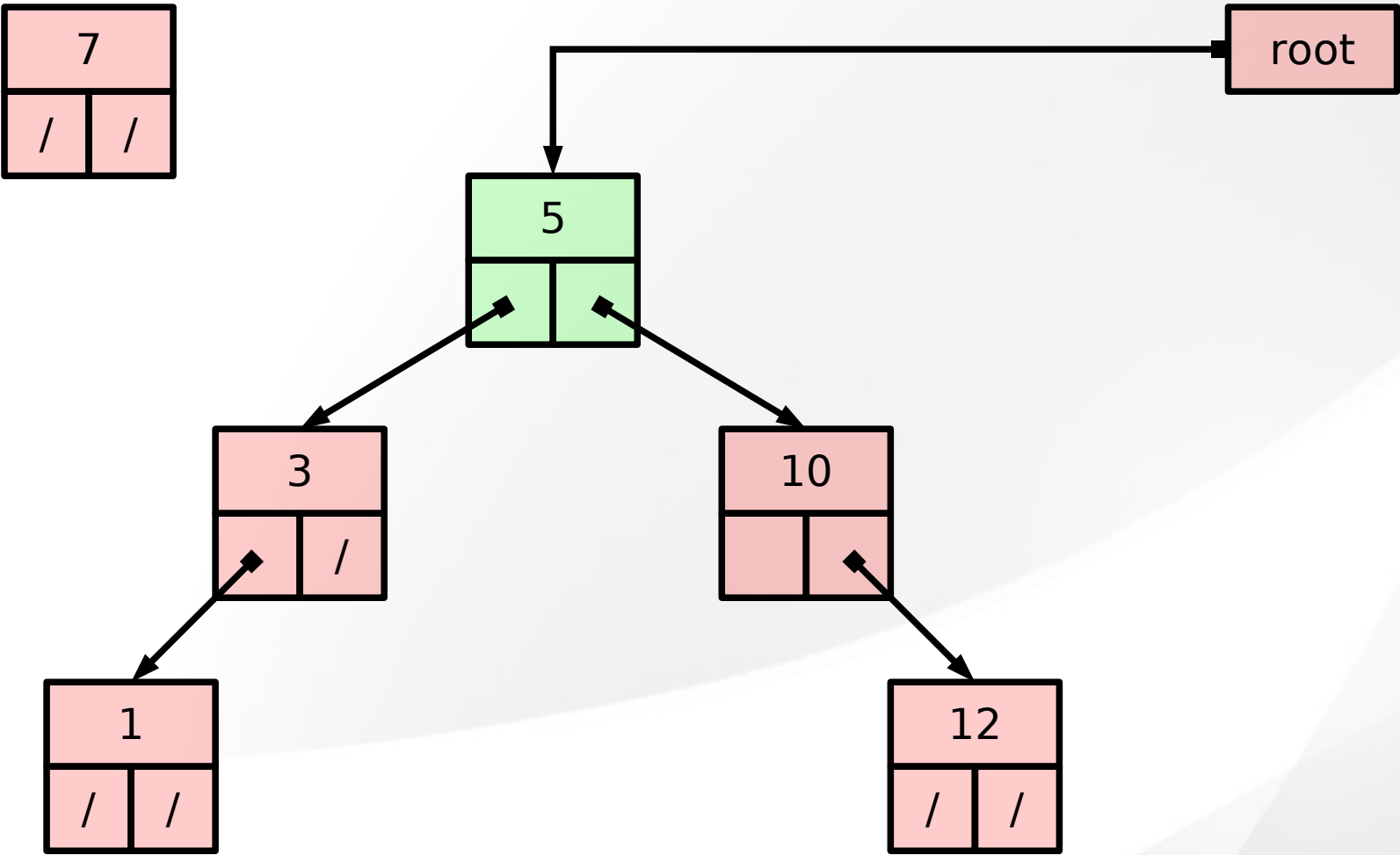
```
typedef Nodo* albero;
```



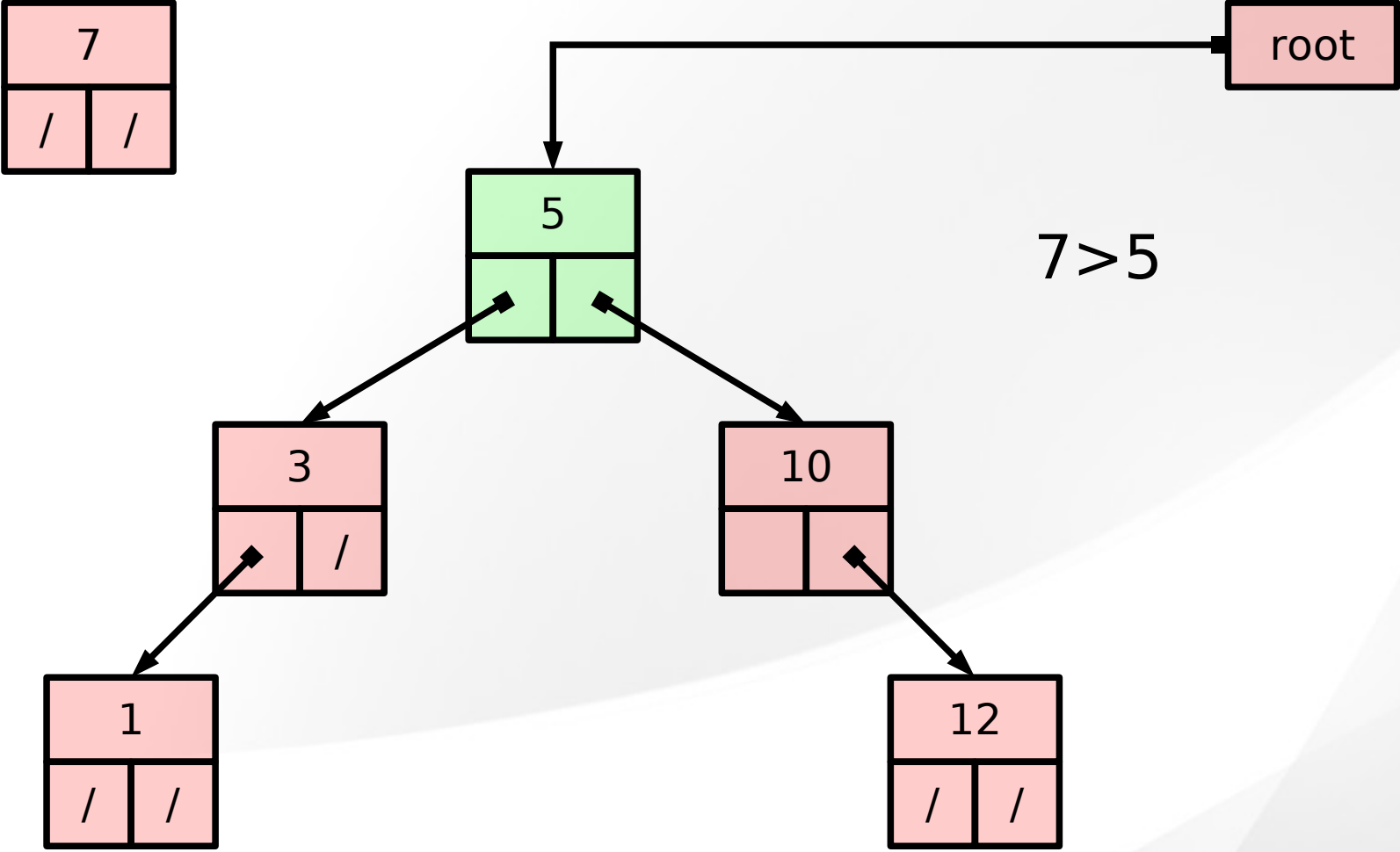
# Inserimento



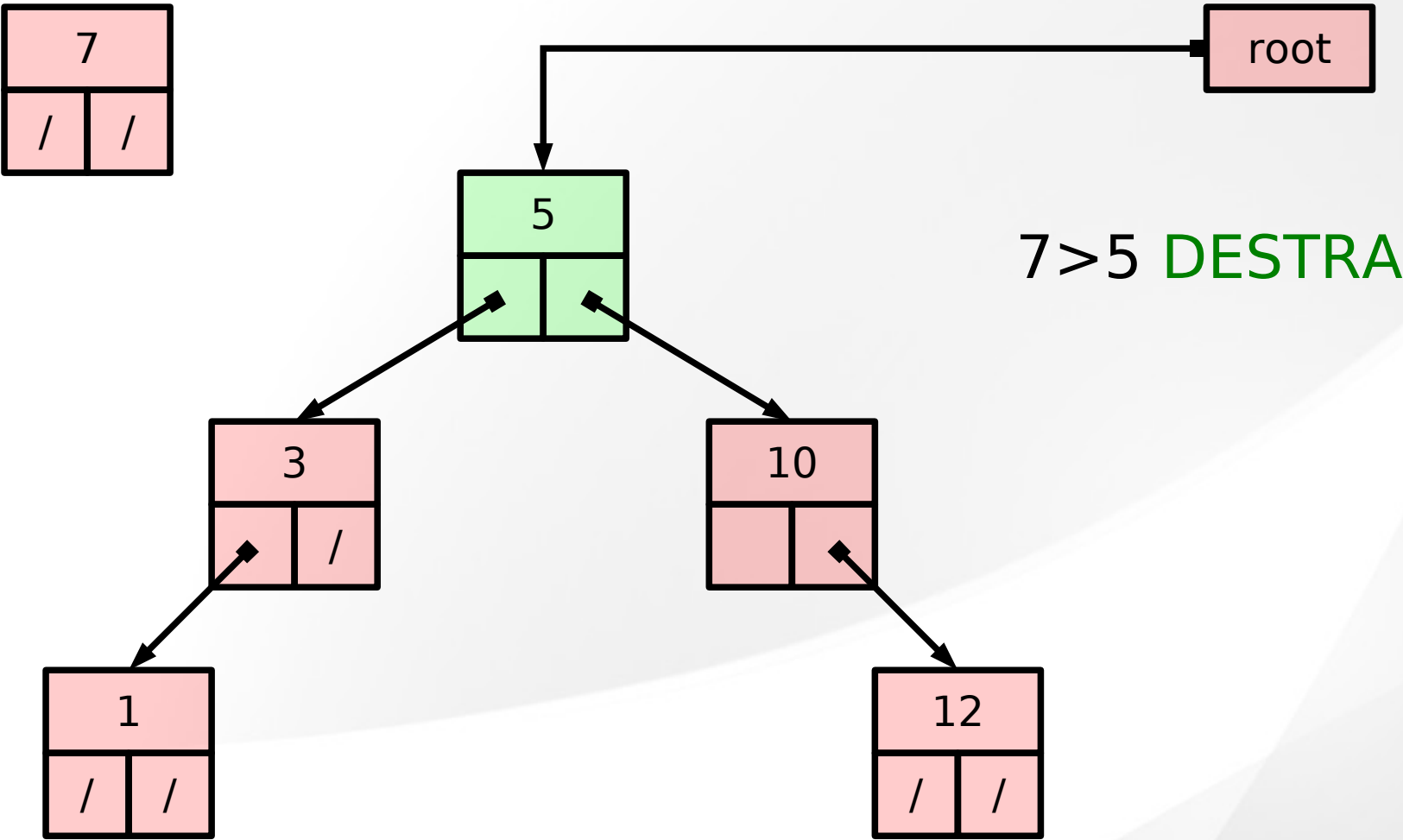
# Inserimento



# Inserimento

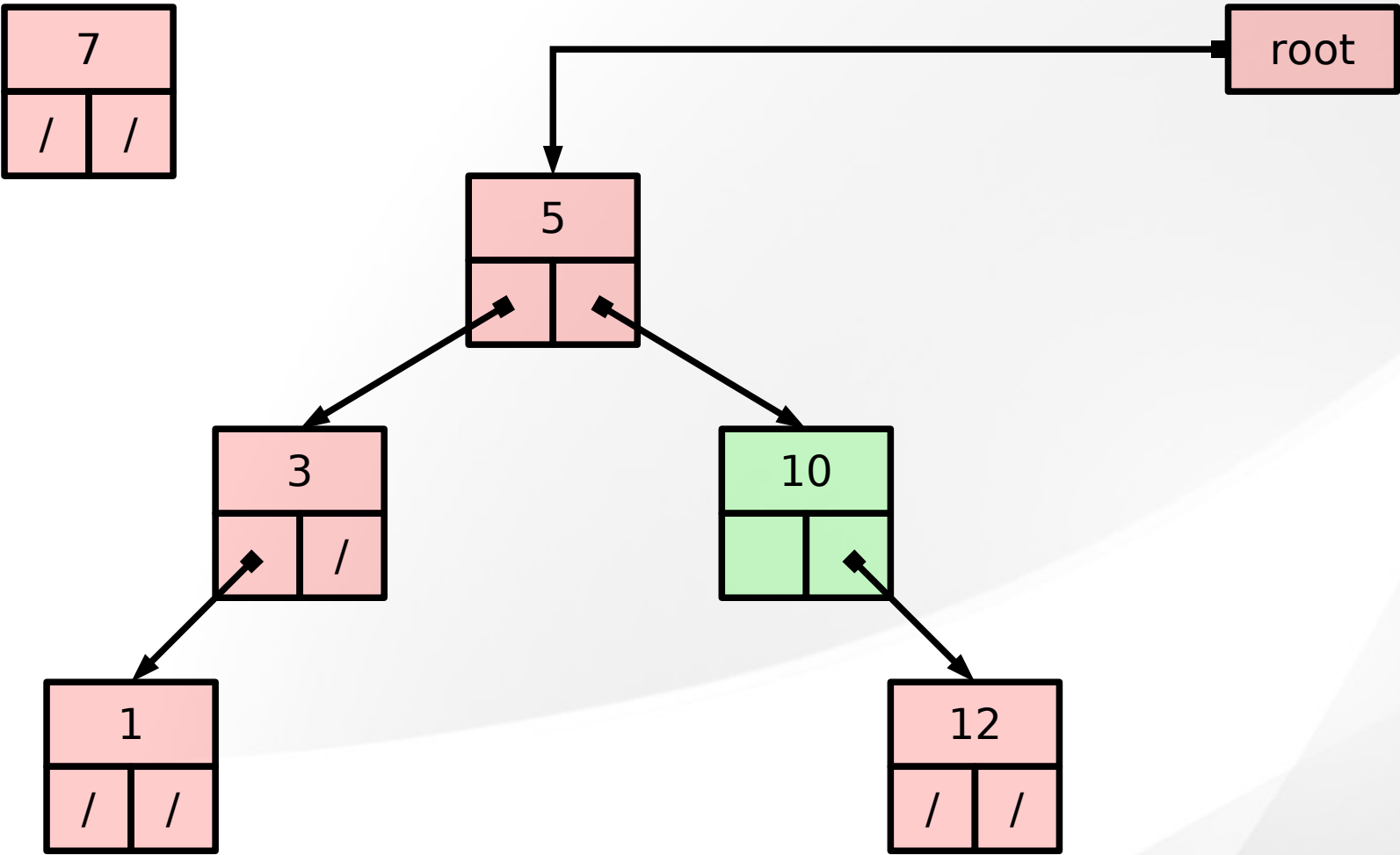


# Inserimento

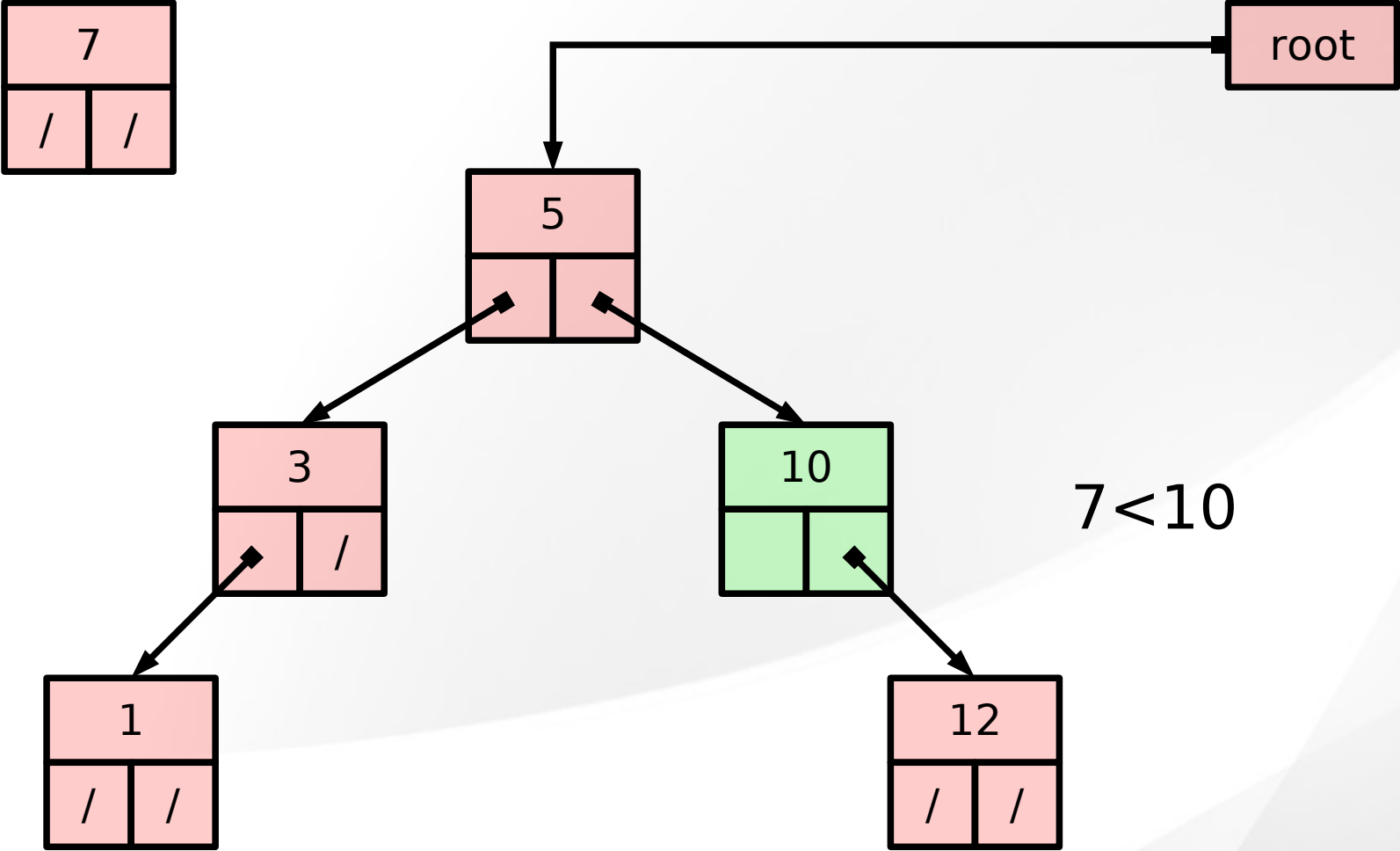




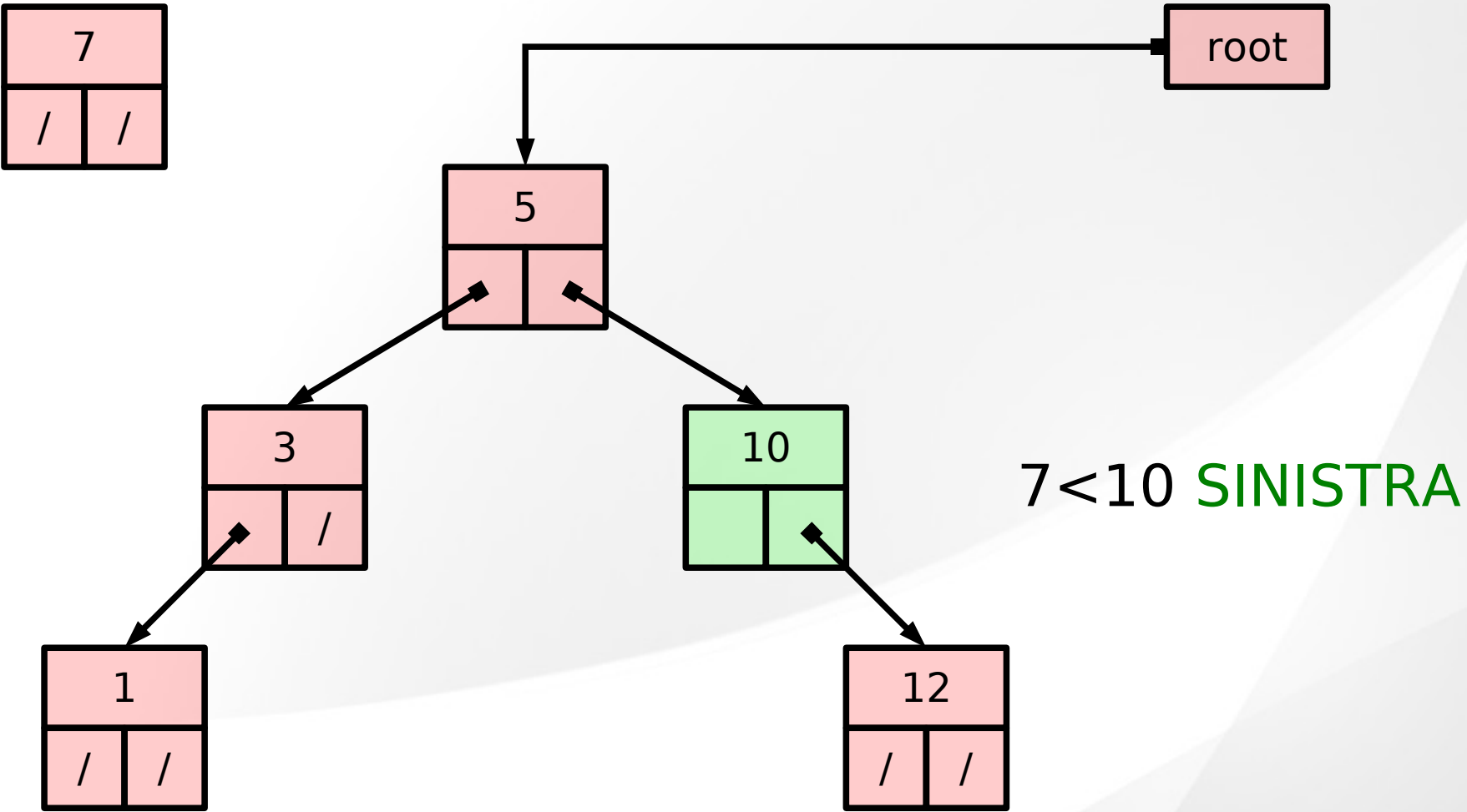
# Inserimento



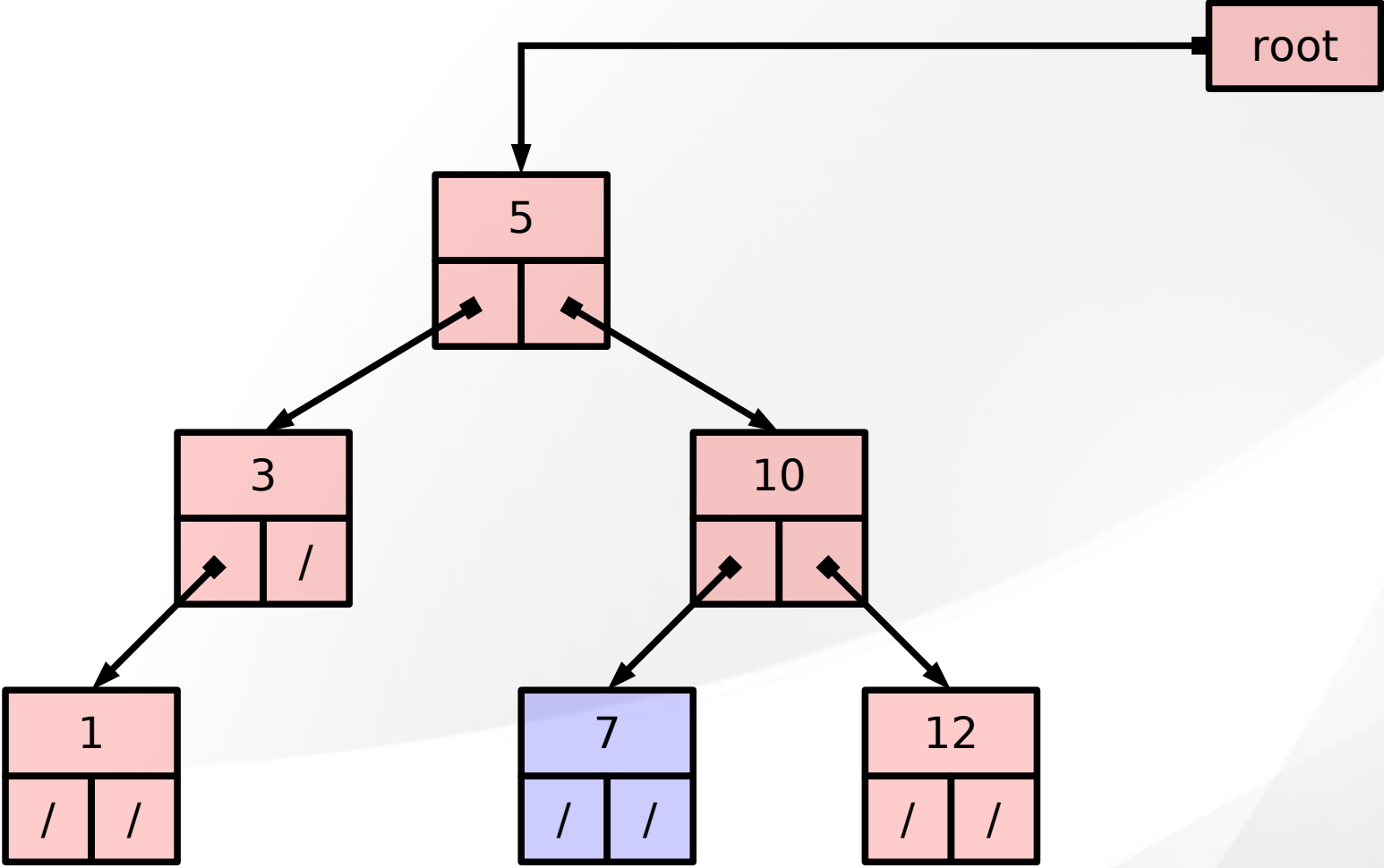
# Inserimento



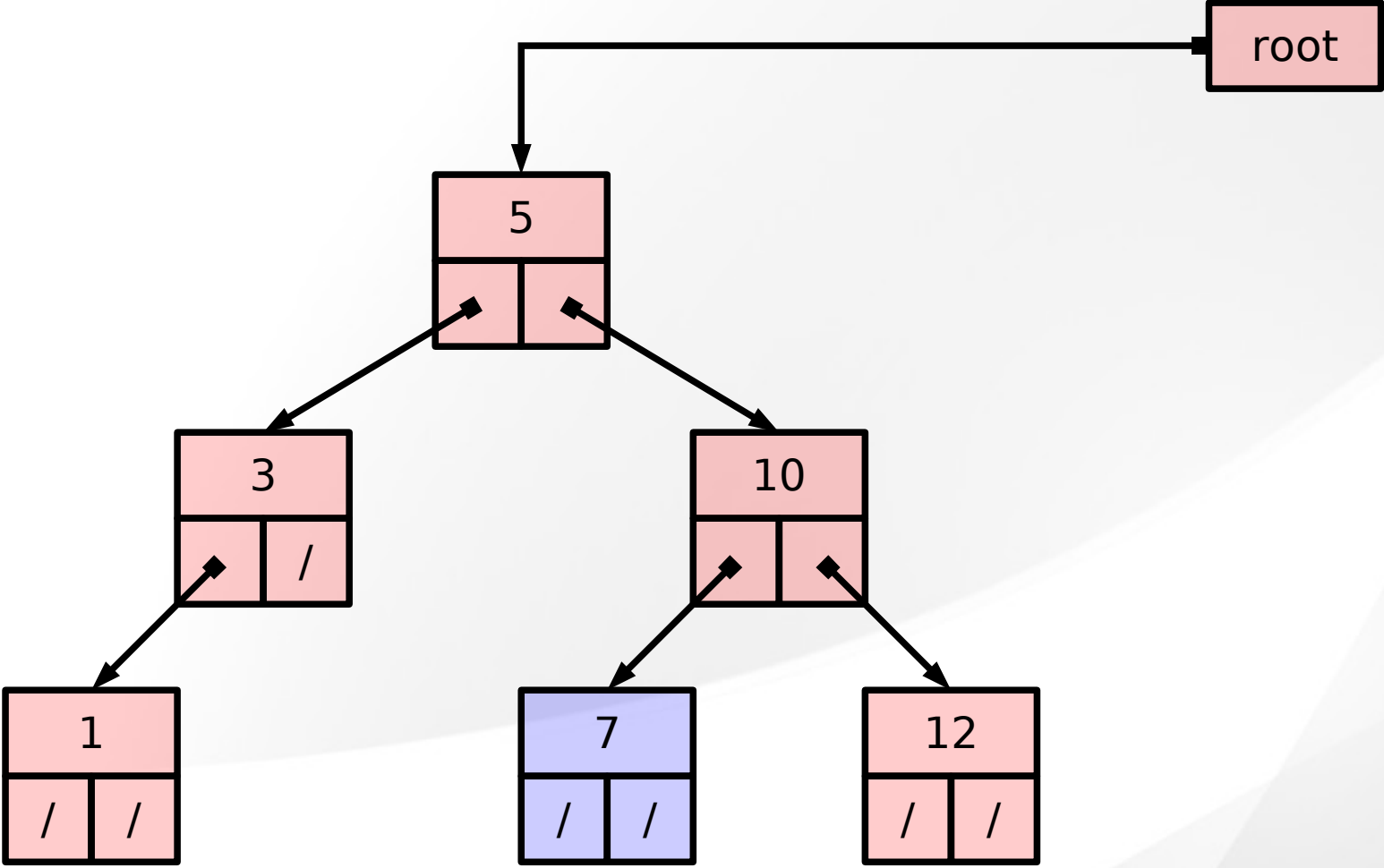
# Inserimento



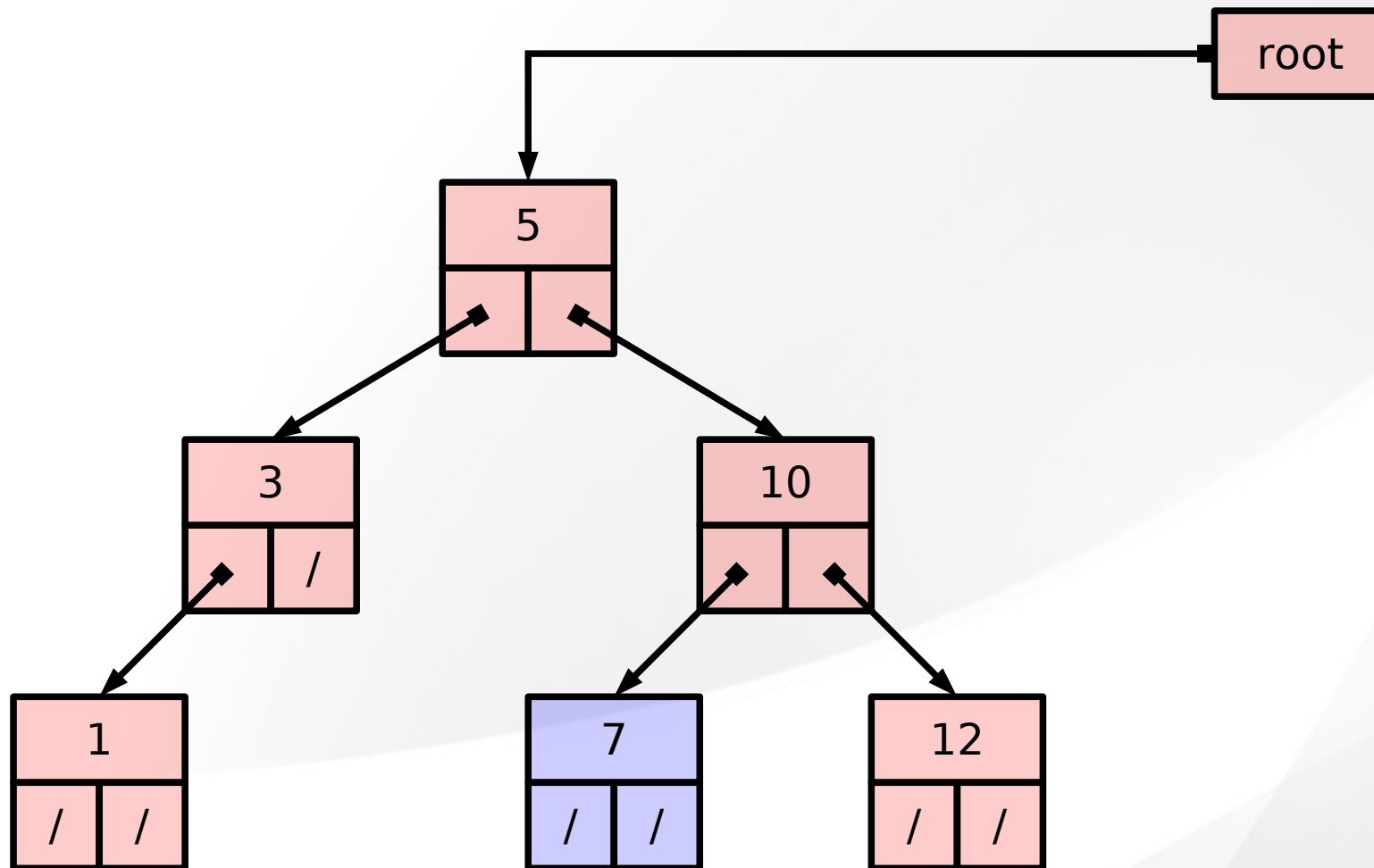
# Inserimento



# Inserimento

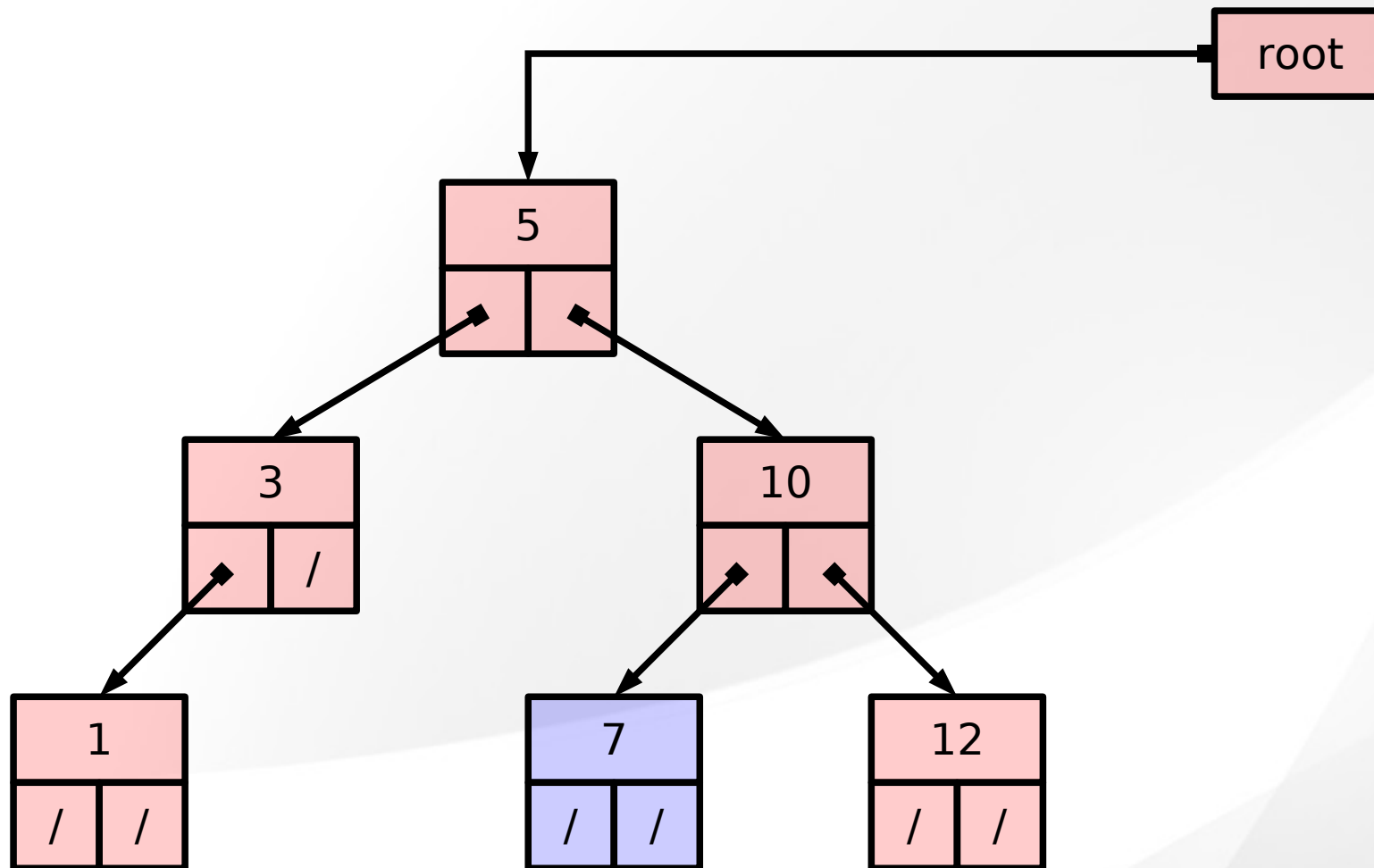


# Inserimento



complessità  $O(h)$

# Inserimento



complessità  $O(h)$

# Inserimento (implementazione)

```
struct Nodo* inserisci(albero t, int key){
    struct Nodo* new = malloc(sizeof(struct Nodo));
    new->key=key;
    new->right=NULL; new->left=NULL;
    if(t==NULL){ return new; }
    struct Nodo* parent;
    struct Nodo* current=t;
    while(current!=NULL){
        parent=current;
        if(current->key<key) current=current->right;
        else current=current->left;
    }
    if(parent->key<key) parent->right=new;
    else parent->left=new;
    return t;
}
```



# Ricerca (implementazione)

```
int cerca(albero t, int key){
    int depth=0;
    struct Nodo* current=t;
    while(current!=NULL){
        if(key==current->key) return depth;
        if(current->key < key) current=current->right;
        else current=current->left;
        depth++;
    }
    return -1;
}
```

# Ricerca (implementazione)

```
int cerca(albero t, int key){
    int depth=0;
    struct Nodo* current=t;
    while(current!=NULL){
        if(key==current->key) return depth;
        if(current->key < key) current=current->right;
        else current=current->left;
        depth++;
    }
    return -1;
}
```

complessità  $O(h)$