



AA 2014-2015

PROGRAMMAZIONE 2

3a. Verso Java

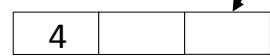
1



**Una implementazione in C degli
alberi binari di ricerca**

1

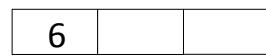
un albero binario di ricerca



```
typedef struct _nodo {
    int key;
    struct _nodo *left;
    struct _nodo *right;
} nodo;
```

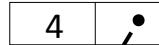
3

un albero binario di ricerca



4

Inserimento iterativo



```

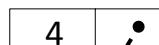
nodo* inserisci(nodo *t, int key) {
    nodo* new = (nodo *) malloc(sizeof(nodo));
    new->key = key;
    new->right = new->left = NULL;
    if (t == NULL) { return new; }

    nodo* parent;
    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;
}

```

5

Inserimento ricorsivo



```

nodo* inserisci(nodo *t, int key) {
    if (t == NULL) {
        nodo* new = (nodo *) malloc(sizeof(nodo));
        new->key = key;
        new->left = NULL;
        new->right = NULL;
        return new;
    }

    if (t->key < key)
        t->right = inserisci(t->right, key);
    else
        t->left = inserisci(t->left, key);
    return t;
}

```

6

ricerca iterativa



4

4

```
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;
}
```

7

ricerca ricorsiva



4

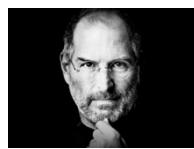
4

```
int cerca(albero t, int key) {
    if (t == NULL) return -1;
    if (t->key == key) return 0;
    int found = -1;
    if (t->key < key)
        found = cerca(t->right, key);
    else
        found = cerca(t->left, key);

    if (found >= 0) return 1+found;
    else return -1;
}
```

8

scenario: ABR modulo condiviso



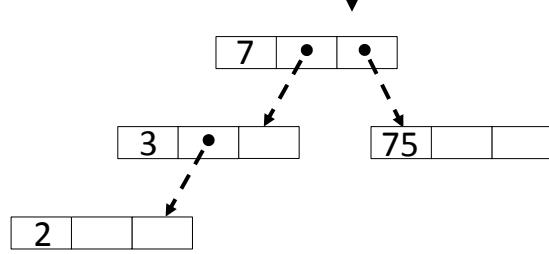
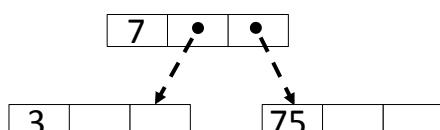
ABR



Programmatori
condividono
la struttura ABR

Goofy's code

```
:
G_node = inserisci(t, 2);
:
G_node >key = 18;
:
```



Goofy's code



:
 G_node = inserisci(t, 2);
 :
 G_node >key = 18;
 :

Viene distrutta l'invariante di rappresentazione

Come mai?

Diagram illustrating a bug in Goofy's code. It shows a sequence of memory blocks and their pointers. In the first state, there are three blocks: one with value 7, one with value 3, and one with value 75. Arrows point from the 7 and 3 blocks to the 75 block. In the second state, the 75 block has been deallocated, leaving a dotted arrow pointing to it. In the third state, the 7 and 3 blocks now both point to the same deallocated space. A fourth block with value 18 is shown at the bottom.

invarianti e rappresentazione

RAPPRESENTAZIONE

(nodo ->left)->key < nodo->key && nodo->key < (nodo ->right)->key

PUBBLICA: visibile a tutti

```
typedef struct _nodo {
    int key;
    struct _nodo *left;
    struct _nodo *right;
} nodo;
```

Diagram illustrating the relationship between representation and public interface. It shows a binary search tree structure with nodes containing keys 10, 4, and 15. Dashed arrows point from the tree structure to a box labeled "RAPPRESENTAZIONE". Below the tree, a condition is given: *(nodo ->left)->key < nodo->key && nodo->key < (nodo ->right)->key*. At the bottom left, it says "PUBBLICA: visibile a tutti". A box on the right contains the C code for the `_nodo` structure definition.

Scenario 2: ABR e dizionario



- ☞ ABR per implementare un dizionario
 - l'estensione richiede di avere una chiave per effettuare la ricerca e una stringa per codificare l'informazione
- ☞ Riuso del codice: utilizziamo il vecchio modulo aggiungendo le opportune modifiche per realizzare il dizionario

13

Scenario 2: ABR e dizionario



```
typedef struct _nodo {
    int key;
    string info;
    struct _nodo *left;
    struct _nodo *right;
} nodo;
```

L'invariante è ora una proprietà delle chiavi

14

ricerca...



```
nodo* inserisci(nodo *t, int key, string info) {
    if (t == NULL) {
        nodo* new = (nodo *) malloc(sizeof(nodo));
        new->key = key;
        new->info = info;
        new->left = NULL;
        new->right = NULL;
        return new;
    }

    if (t->key < key)
        t->right = inserisci(t->right, key, info);
    else
        t->left = inserisci(t->left, key, info);
    return t;
}
```

15

valutazione della soluzione



- ☞ Non abbiamo strumenti linguistici (ovvero previsti nel linguaggio) per estendere il codice alle nuove esigenze
- ☞ Cut&Paste Reuse
 - codice debole
 - difficile da mantenere
 - difficile evitare errori

16

sull'astrazione



“Abstraction arises from a recognition of similarities between certain objects, situations, or processes in the real world, and the decision to concentrate upon those similarities and to ignore for the time being the differences.”

[Tony Hoare]

▲ Astrazione: separare le funzionalità offerte dalla loro implementazione

17

funzionalità vs. implementazione



18

encapsulation



“Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation.”

[Grady Booch]



19

sull'ereditarietà



- Passare dal *Cut&Paste reuse* a un metodo supportato da strumenti linguistici nel quale una nuova funzionalità è ottenuta estendendo esplicitamente del codice già implementato
- La nuova implementazione estende la vecchia con funzionalità aggiuntive ma conservando quelle esistenti

20