

Laboratorio di Algoritmi e Strutture Dati

2020/2021 — Seconda parte

Mattia Bonaccorsi — 124610 – bonaccorsi.mattia@spes.uniud.it

Muhammed Kouate — 137359 – kouate.muhammed@spes.uniud.it

Enrico Stefanel — 137411 – stefanel.enrico@spes.uniud.it

Andriy Torchanyyn — 139535 – torchanyyn.andriy@spes.uniud.it

21 aprile 2021

Indice

1	Alberi binari di ricerca semplici	2
1.1	Definizione di <i>BST</i>	2
1.2	Implementazione della struttura dati	2
1.3	Difetti dei <i>BST</i>	3
2	Alberi binari di ricerca di tipo AVL	4
2.1	Definizione di Albero <i>AVL</i>	4
2.2	Implementazione della struttura dati	4
2.3	Miglioramenti degli Alberi AVL rispetto ai <i>BST</i>	6
3	Alberi binari di ricerca di tipo Red-Black	6
3.1	Definizione di Albero <i>AVL</i>	6
3.2	Implementazione della struttura dati	7

1 Alberi binari di ricerca semplici

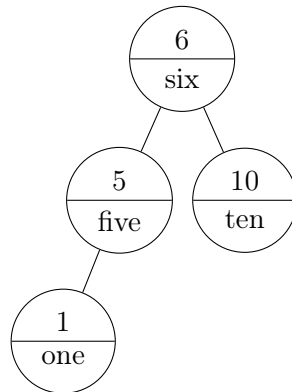
1.1 Definizione di *BST*

Un *albero binario di ricerca* (o *BST*) T è una struttura dati ad albero, in cui valgono le seguenti proprietà:

$$\begin{aligned} \forall x \in T, \forall y \in \text{left}(T) &\rightarrow y.\text{key} < x.\text{key} \\ \forall x \in T, \forall z \in \text{right}(T) &\rightarrow z.\text{key} > x.\text{key} \end{aligned} \quad (*)$$

dove $k.\text{key}$ indica il valore della chiave di k , e $\text{left}(B)$ (rispettivamente $\text{right}(B)$) indica il sottoalbero sinistro (rispettivamente destro) di B .

Esempio Un BST in cui ogni nodo contiene una chiave numerica tra $\{1, 5, 6, 10\}$ e un campo alfanumerico di tipo stringa, è il seguente:



1.2 Implementazione della struttura dati

```
1 class Node():
2     def __init__(self, value, str_name):
3         self.key = value
4         self.name = str_name
5         self.left = None
6         self.right = None
7
8
9 def bst_insert(root, value, str_name):
10     """
11     insert a key value in a tree
12     :param root: BSTNode object that represent the root of the tree
13     :param value: an integer representing the value to insert
14     :param str_name: a string corresponding to the literal format of the value
15     :return: a BSTNode object
16     """
17     if root is None:
```

```

18         return Node(value, str_name)
19
20     if value < root.key:
21         root.left = bst_insert(root.left, value, str_name)
22     else:
23         root.right = bst_insert(root.right, value, str_name)
24
25     return root
26
27
28 def bst_find(root, value):
29     """
30     print the found value in a literal format
31     :param root: BSTNode object that represent the root of the tree
32     :param value: an integer representing the value to find
33     """
34     if root is None:
35         return
36
37     if root.key == value:
38         print(root.name)
39
40     if root.key < value:
41         return bst_find(root.right, value)
42
43     return bst_find(root.left, value)
44
45
46 def bst_show(root):
47     """
48     print the tree in a preorder visit
49     :param root: BSTNode object that represent the root of the tree
50     """
51     if root:
52         print(str(root.key) + ":" + root.name, end=" ")
53         bst_show(root.left)
54         bst_show(root.right)
55     else:
56         print("NULL", end=" ")

```

sources/bst.py

1.3 Difetti dei *BST*

Questa struttura dati ha alcuni difetti ...

2 Alberi binari di ricerca di tipo AVL

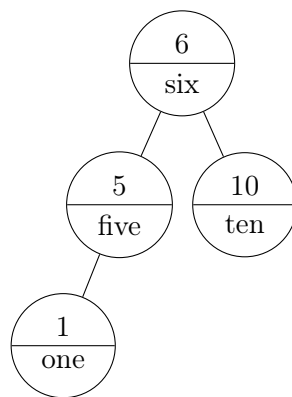
2.1 Definizione di Albero AVL

Un albero AVL T è un BST (\star), in cui vale la seguente proprietà:

$$\forall x \in T \rightarrow |h(\text{left}(x)) - h(\text{right}(x))| \leq 1 \quad (*)$$

dove $h(k)$ indica il valore dell'altezza dell'albero radicato in k , e $\text{left}(B)$ (rispettivamente $\text{right}(B)$) indica il sottoalbero sinistro (rispettivamente destro) di B .

Esempio Un BST in cui ogni nodo contiene una chiave numerica tra $\{1, 5, 6, 10\}$ e un campo alfanumerico di tipo stringa, è il seguente:



, dove $\text{left}(T)$ ha altezza 3, mentre $\text{right}(T)$ ha altezza 2.

2.2 Implementazione della struttura dati

```
1 class AVLNode():
2     def __init__(self, value, str_name):
3         self.key = value
4         self.name = str_name
5         self.left = None
6         self.right = None
7         self.height = 1
8
9
10 def avl_insert(root, value, str_name):
11     """
12     insert a key value in a tree
13     :param root: AVLNode object that represent the root of the tree
14     :param value: an integer representing the value to insert
15     :param str_name: a string corresponding to the literal format of the value
16     :return: an AVLNode object
17     """
```

```

18     if root is None:
19         return AVLNode(value, str_name)
20
21     if value < root.key:
22         root.left = avl_insert(root.left, value, str_name)
23     else:
24         root.right = avl_insert(root.right, value, str_name)
25
26     root.height = 1 + max(getHeight(root.left), getHeight(root.right))
27
28     balance = getBalance(root)
29
30     # LL
31     if balance > 1 and value < root.left.key:
32         return rightRotate(root)
33
34     # RR
35     if balance < -1 and value > root.right.key:
36         return leftRotate(root)
37
38     # LR
39     if balance > 1 and value > root.left.key:
40         root.left = leftRotate(root.left)
41         return rightRotate(root)
42
43     # RL
44     if balance < -1 and value < root.right.key:
45         root.right = rightRotate(root.right)
46         return leftRotate(root)
47
48     return root
49
50
51 def avl_show(root):
52     """
53     print the tree in a preorder visit
54     :param root: AVLNode object that represent the root of the tree
55     """
56     if root:
57         print(str(root.key) + ":" + root.name + ":" + str(root.height), end="
58         avl_show(root.left)
59         avl_show(root.right)
60     else:
61         print("NULL", end=" ")
62
63
64 def avl_find(root, value):
65     """
66     print the found value in a literal format

```

```

67     :param root: AVLNode object that represent the root of the tree
68     :param value: an integer representing the value to find
69     """
70     if root is None:
71         return
72
73     if root.key == value:
74         print(root.name)
75
76     if root.key < value:
77         return avl_find(root.right, value)
78
79     return avl_find(root.left, value)

```

sources/avl.py

2.3 Miglioramenti degli Alberi AVL rispetto ai *BST*

Questa struttura dati migliora alcuni aspetti dei *BST* ...

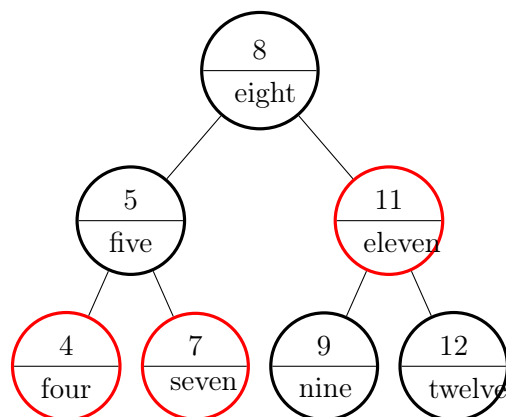
3 Alberi binari di ricerca di tipo Red-Black

3.1 Definizione di Albero *AVL*

Un *albero di tipo Red-Black* (o *RB Tree*) T è un *BST* (\star), in cui vale la seguente proprietà:

... (•)

Esempio



3.2 Implementazione della struttura dati

```
1 class RBTNode():
2     def __init__(self, value, str_name):
3         self.key = value
4         self.name = str_name
5         self.parent = None
6         self.left = None
7         self.right = None
8         self.color = "red"
9
10
11 class RedBlackTree():
12     def __init__(self):
13         self.TNIL = RBTNode(None, None)
14         self.TNIL.color = "black"
15         self.TNIL.left = None
16         self.TNIL.right = None
17         self.root = self.TNIL
18
19     def left_rotate(self, x):
20         y = x.right
21         x.right = y.left
22
23         if y.left != self.TNIL:
24             y.left.parent = x
25
26         y.parent = x.parent
27
28         if x.parent == self.TNIL:
29             self.root = y
30         elif x == x.parent.left:
31             x.parent.left = y
32         else:
33             x.parent.right = y
34
35         y.left = x
36         x.parent = y
37
38     def right_rotate(self, x):
39         y = x.left
40         x.left = y.right
41
42         if y.right != self.TNIL:
43             y.right.parent = x
44
45         y.parent = x.parent
46
47         if x.parent == self.TNIL:
```

```

48         self.root = y
49     elif x == x.parent.right:
50         x.parent.right = y
51     else:
52         x.parent.left = y
53
54     y.right = x
55     x.parent = y
56
57     def rbt_insert(self, value, str_name):
58
59         z = RBTNode(value, str_name)
60         z.left = self.TNIL
61         z.right = self.TNIL
62
63         y = self.TNIL
64         x = self.root
65
66         while x != self.TNIL:
67             y = x
68             if z.key < x.key:
69                 x = x.left
70             else:
71                 x = x.right
72
73         z.parent = y
74
75         if y == self.TNIL:
76             self.root = z
77         elif z.key < y.key:
78             y.left = z
79         else:
80             y.right = z
81
82         self.insert_fix_up(z)
83
84     def insert_fix_up(self, z):
85         while z.parent.color == "red":
86             if z.parent == z.parent.parent.right:
87                 y = z.parent.parent.left
88                 if y.color == "red":
89                     y.color = "black"
90                     z.parent.color = "black"
91                     z.parent.parent.color = "red"
92                     z = z.parent.parent
93             else:
94                 if z == z.parent.left:
95                     z = z.parent
96                     self.right_rotate(z)

```



```

97         z.parent.color = "black"
98         z.parent.parent.color = "red"
99         self.left_rotate(z.parent.parent)
100     else:
101         y = z.parent.parent.right
102
103         if y.color == "red":
104             y.color = "black"
105             z.parent.color = "black"
106             z.parent.parent.color = "red"
107             z = z.parent.parent
108         else:
109             if z == z.parent.right:
110                 z = z.parent
111                 self.left_rotate(z)
112                 z.parent.color = "black"
113                 z.parent.parent.color = "red"
114                 self.right_rotate(z.parent.parent)
115             if z == self.root:
116                 break
117         self.root.color = "black"
118
119
120 def rbt_show(root):
121     """
122     print the tree in a preorder visit
123     :param root: RBTNode object that represent the root of the tree (accessed
124     """
125     if root.key is None:
126         print("NULL", end=" ")
127         return
128     if root.name is None:
129         print("NULL", end=" ")
130         return
131     if root:
132         print(str(root.key) + ":" + root.name + ":" + str(root.color), end=" ")
133         rbt_show(root.left)
134         rbt_show(root.right)
135     else:
136         print("NULL", end=" ")
137
138
139 def rbt_find(root, value):
140     """
141     print the found value in a literal format
142     :param root: RBTNode object that represent the root of the tree (accessed
143     :param value: an integer representing the value to find
144     """
145     if root.key is None:

```

```
146         return
147     if root.name is None:
148         return
149     if root is None:
150         return
151
152     if root.key == value:
153         print(root.name)
154
155     if root.key < value:
156         return rbt_find(root.right, value)
157
158     return rbt_find(root.left, value)
```

sources/rbt.py