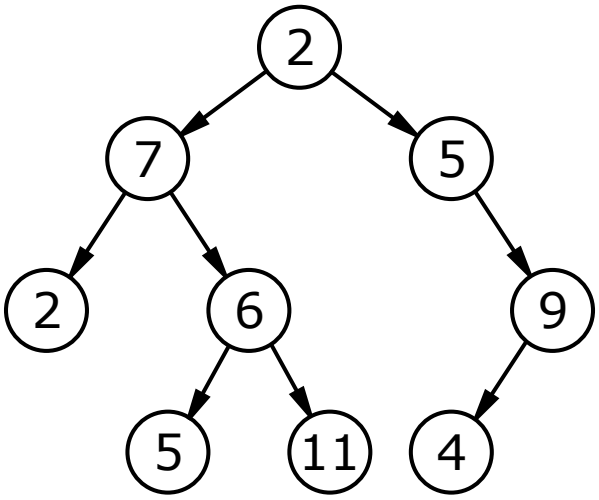
Δομές Δεδομένων και Αλγόριθμοι - Εργαστήριο 9
Δένδρα - Δυαδικά δένδρα αναζήτησης

Τ.Ε.Ι. Ηπείρου, Τμήμα Μηχανικών Πληροφορικής Τ.Ε.
Χρήστος Γκόγκος - Αναπληρωτής Καθηγητής

1 Εισαγωγή

Τα δένδρα όπως και τα γραφήματα είναι μη γραμμικές δομές δεδομένων που αποτελούν συλλογές κόμβων. Τα δένδρα επιτρέπουν ιεραρχική οργάνωση των δεδομένων όπως φαίνεται στο Σχήμα 1. Αυτό το στοιχείο τους επιτρέπει να έχουν καλύτερες επιδόσεις προσπέλασης των επιμέρους στοιχείων σε σχέση με τις γραμμικές λίστες. Με κατάλληλη διευθέτηση των στοιχείων ενός δένδρου καθώς και με εφαρμογή προχωρημένων μηχανισμών εισαγωγής και διαγραφής στοιχείων ο χρόνος εκτέλεσης των περισσότερων λειτουργιών σε ένα δένδρο (ισοζυγισμένο δυαδικό δένδρο αναζήτησης) γίνεται $O(\log n)$. Στην STL τα δένδρα χρησιμοποιούνται στην υλοποίηση των containers std::map και std::set.



Σχήμα 1: Ένα απλό δένδρο [1]

2 Δένδρα

Ένα δένδρο (tree) αποτελείται από κόμβους (nodes) που συνδέονται μεταξύ τους με κατευθυνόμενες ακμές (edges). Ο πρώτος (υψηλότερος) κόμβος του δένδρου ονομάζεται ρίζα (root) ενώ οι κόμβοι που βρίσκονται στα άκρα του δένδρου λέγονται φύλλα (leaves). Οι κόμβοι με τους οποίους συνδέεται απευθείας ένας κόμβος ονομάζονται παιδιά (children) του κόμβου. Αντίστοιχα, ένας κόμβος που έχει παιδιά ονομάζεται γονέας (parent) των αντίστοιχων παιδιών-κόμβων. Απόγονοι (descendants) ενός κόμβου είναι οι κόμβοι για τους οποίους υπάρχει διαδρομή-μονοπάτι (path) πραγματοποιώντας διαδοχικές μεταβάσεις από γονείς σε παιδιά. Αντίστοιχα ορίζεται και η έννοια των προγόνων (ancestors) ενός κόμβου με τη ρίζα να είναι ο μοναδικός κόμβος που δεν έχει προγόνους.

Τα δένδρα είναι αναδρομικές δομές από τη φύση τους. Κάθε κόμβος ενός δένδρου ορίζει έναν αριθμό από μικρότερα δένδρα, ένα για κάθε παιδί του. Σε ένα δένδρο με N κόμβους υπάρχουν πάντα $N-1$ ακμές καθώς όλοι οι κόμβοι εκτός από τον κόμβο ρίζα έχουν μια ακμή η οποία τους συνδέει με τον γονέα τους.

Το μήκος ενός μονοπατιού ανάμεσα σε δύο κόμβους είναι ίσο με το πλήθος των ακμών του μονοπατιού. Εφόσον υπάρχει μονοπάτι μέσω του οποίου συνδέονται δύο κόμβοι το μονοπάτι αυτό είναι μοναδικό. Για κάθε κόμβο ορίζεται ως **βάθος του κόμβου** (depth) το μήκος του μονοπατιού από τη ρίζα του δένδρου μέχρι τον ίδιο τον κόμβο. Αντίστοιχα, **ύψος ενός κόμβου** (height) είναι το μήκος του μακρύτερου μονοπατιού από τον κόμβο προς ένα από τα φύλλα του δένδρου για τα οποία υφίσταται μονοπάτι με αφετηρία τον κόμβο.

3 Δυαδικά δένδρα

Δυαδικό δένδρο είναι ένα δένδρο για το οποίο ισχύει ότι κάθε κόμβος έχει το πολύ δύο παιδιά [2]. Ένα δένδρο μπορεί να διανυθεί με διαφορετικούς τρόπους. Ορισμένοι βασικοί τρόποι διάσχισης (traversal) του δένδρου παρουσιάζονται στη συνέχεια [5].

3.1 Αναζήτηση κατά βάθος

Η αναζήτηση κατά βάθος (DFS = Depth First Search) διανύει το δένδρο αναζήτησης εξαντλώντας μονοπάτια από τη ρίζα προς τα φύλλα του δένδρου. Ένας τρόπος για να επιτευχθεί αυτό είναι η χρήση αναδρομής.

3.1.1 Προ-διατακτική αναζήτηση κατά βάθος

Στη διάσχιση του δένδρου προ-διατακτικά (pre-order) πρώτα πραγματοποιείται η επίσκεψη στη ρίζα και μετά καλείται αναδρομικά η ίδια συνάρτηση πρώτα για το αριστερό υποδένδρο και μετά για το δεξιό υποδένδρο. Συνηθισμένες χρήσεις της pre-order διάσχισης είναι η δημιουργία αντιγράφων ενός δένδρου καθώς και η λήψη της prefix μορφής ενός expression tree [3].

3.1.2 Ένδο-διατακτική αναζήτηση κατά βάθος

Στη διάσχιση του δένδρου ένδο-διατακτικά (in-order) καλείται αναδρομικά η συνάρτηση για το αριστερό υποδένδρο, μετά πραγματοποιείται επίσκεψη στη ρίζα και μετά καλείται αναδρομικά η συνάρτηση για το δεξιό υποδένδρο. Εφόσον το δένδρο είναι δυαδικό δένδρο αναζήτησης, η in-order διάσχιση επιστρέφει τους κόμβους σε μη φθίνουσα σειρά. Σχετικά με το τι είναι τα δυαδικά δένδρα αναζήτησης δείτε την παράγραφο 4).

3.1.3 Μέτα-διατακτική αναζήτηση κατά βάθος

Στη διάσχιση του δένδρου μέτα-διατακτικά (post-order) πρώτα καλείται αναδρομικά η συνάρτηση για το αριστερό υποδένδρο, μετά καλείται αναδρομικά για το δεξιό υποδένδρο και τέλος πραγματοποιείται η επίσκεψη στη ρίζα. Συνηθισμένες χρήσεις της post-order διάσχισης είναι η διαγραφή ενός δένδρου καθώς και η λήψη της postfix μορφής ενός expression tree [4].

3.2 Αναζήτηση κατά πλάτος

Στην αναζήτηση κατά πλάτος (BFS=Breadth First Search) οι κόμβοι του δένδρου διανύονται κατά επίπεδα ξεκινώντας από τη ρίζα και μεταβαίνοντας από πάνω προς τα κάτω. Σε κάθε επίπεδο η προσπέλαση στους κόμβους γίνεται από αριστερά προς τα δεξιά. Για να επιτευχθεί αυτό το είδος διάσχισης του δένδρου χρησιμοποιείται μια ουρά (queue) στην οποία μόλις εξετάζεται ένα στοιχείο προστίθενται στο πίσω άκρο της ουράς τα παιδιά του.

```
#include <cstddef>
#include <string>

struct node {
    std::string key;
    node *left;
    node *right;
};

node* insert(node *root_node, std::string key);
void print_level_order(node *root_node);
void print_pre_order(node *root_node);
void destroy(node *root_node);
void print_in_order(node *root_node);
void print_post_order(node *root_node);
```

Κώδικας 1: header file για το δυαδικό δένδρο (binary_tree.hpp)

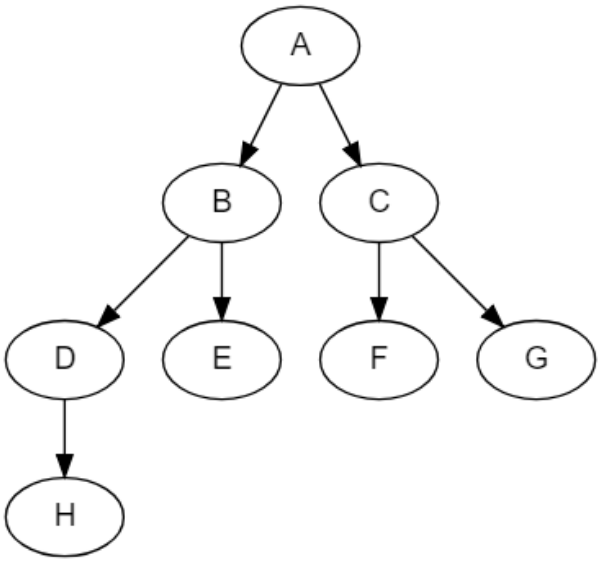
```
#include "binary_tree.hpp"
#include <queue>
#include <iostream>
using namespace std;

node* insert(node *root_node, string key)
{
    if (root_node == NULL)
    {
        cout << "key " << key << " inserted (root of the tree)" << endl;
        return new node{key, NULL, NULL};
    }
    else
    {
        queue<node *> q;
        q.push(root_node);
        while (!q.empty())
        {
            node *anode = q.front();
            q.pop();
            if (anode->left != NULL && anode->right != NULL)
            {
                q.push(anode->left);
                q.push(anode->right);
            }
            else
            {
                if (anode->left == NULL)
                {
                    anode->left = new node{key, NULL, NULL};
                }
                else
                {
                    anode->right = new node{key, NULL, NULL};
                }
                cout << "key " << key << " inserted" << endl;
                return anode;
            }
        }
    }
    return NULL;
}
```

```
43
44 void print_level_order(node *root_node)
45 {
46     if (root_node == NULL)
47     {
48         return;
49     }
50     queue<node *> q;
51     q.push(root_node);
52     while (!q.empty())
53     {
54         node *node = q.front();
55         q.pop();
56         cout << node->key << " ";
57         if (node->left != NULL)
58         {
59             q.push(node->left);
60         }
61         if (node->right != NULL)
62         {
63             q.push(node->right);
64         }
65     }
66 }
67
68 void print_pre_order(node *root_node)
69 {
70     if (root_node != NULL)
71     {
72         cout << root_node->key << " ";
73         if (root_node->left != NULL)
74         {
75             print_pre_order(root_node->left);
76         }
77         if (root_node->right != NULL)
78         {
79             print_pre_order(root_node->right);
80         }
81     }
82     else
83     {
84         cout << "EMPTY";
85     }
86 }
87
88 void print_in_order(node *root_node)
89 {
90     if (root_node != NULL)
91     {
92         if (root_node->left != NULL)
93         {
94             print_in_order(root_node->left);
95         }
96         cout << root_node->key << " ";
97         if (root_node->right != NULL)
98         {
99             print_in_order(root_node->right);
100        }
101    }
102    else
103    {
```

```
104             cout << "EMPTY";
105         }
106 }
107
108 void print_post_order(node *root_node)
109 {
110     if (root_node != NULL)
111     {
112         if (root_node->left != NULL)
113         {
114             print_post_order(root_node->left);
115         }
116         if (root_node->right != NULL)
117         {
118             print_post_order(root_node->right);
119         }
120         cout << root_node->key << " ";
121     }
122     else
123     {
124         cout << "EMPTY";
125     }
126 }
127
128 void destroy(node *root_node)
129 {
130     if (root_node != NULL)
131     {
132         destroy(root_node->left);
133         destroy(root_node->right);
134         delete root_node;
135     }
136 }
```

Κώδικας 2: source file για το δυαδικό δένδρο αναζήτησης (binary_tree.cpp)

Ο ακόλουθος κώδικας δημιουργεί το δυαδικό δένδρο του Σχήματος 2.



Σχήμα 2: Δυαδικό δένδρο με λεκτικά ως τιμές κλειδιών στους κόμβους

```
#include "binary_tree.hpp"
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    node *root_node = NULL;
    vector<string> v = {"A", "B", "C", "D", "E", "F", "G", "H"};
    for (string x : v)
    {
        if (root_node == NULL)
            root_node = insert(root_node, x);
        else
            insert(root_node, x);
    }

    cout << "Level-order Traversal ";
    print_level_order(root_node);
    cout << endl;
    cout << "Pre-order Traversal ";
    print_pre_order(root_node);
    cout << endl;
    cout << "In-order Traversal ";
    print_in_order(root_node);
    cout << endl;
    cout << "Post-order Traversal ";
    print_post_order(root_node);
    cout << endl;
}
```

Κώδικας 3: Δοκιμή των συναρτήσεων του δυαδικού δένδρου (binary_tree_ex1.cpp)

Η μεταγλώττιση και η εκτέλεση του κώδικα γίνεται με τις ακόλουθες εντολές:

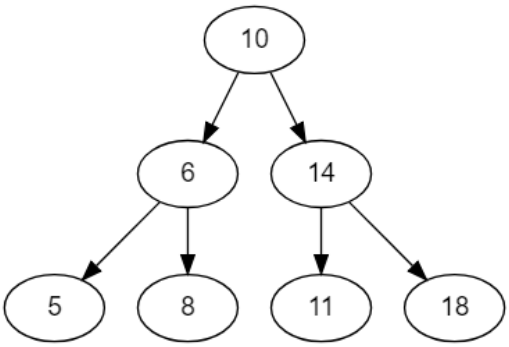
```
$ g++ -Wall -std=c++11 binary_tree.cpp binary_tree_ex1.cpp -o binary_tree_ex1
$ ./binary_tree_ex1
```

Η δε έξοδος που παράγεται και για τους 4 τρόπους διάσχισης του δένδρου είναι η ακόλουθη:

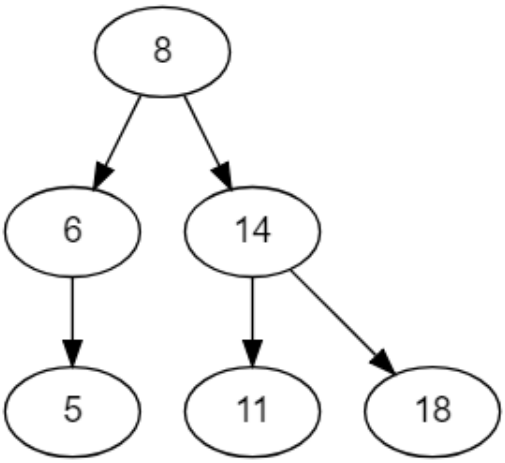
```
key A inserted (root of the tree)
key B inserted
key C inserted
key D inserted
key E inserted
key F inserted
key G inserted
key H inserted
Level-order Traversal A B C D E F G H
Pre-order Traversal A B D H E C F G
In-order Traversal H D B E A F C G
Post-order Traversal H D E B F G C A
```

4 Δυαδικά δένδρα αναζήτησης

Σε ένα δυαδικό δένδρο αναζήτησης θα πρέπει να ισχύει ότι για κάθε κόμβο όλες οι τιμές κλειδιών στο δένδρο αριστερά του κόμβου θα πρέπει να είναι μικρότερες από την τιμή κλειδιού του κόμβου. Αντίστοιχα, όλες οι τιμές κλειδιών στο δένδρο δεξιά του κάθε κόμβου θα πρέπει να είναι μεγαλύτερες από την τιμή κλειδιού του κόμβου.



Σχήμα 3: Δυαδικό δένδρο αναζήτησης



Σχήμα 4: Το δυαδικό δένδρο αναζήτησης μετά τη διαγραφή της ρίζας

4.1 Υλοποίηση δυαδικού δένδρου αναζήτησης

Ιδιαίτερη προσοχή θα πρέπει να δοθεί στην υλοποίηση της διαγραφής ενός κόμβου από το δένδρο έτσι ώστε το δένδρο και μετά τη διαγραφή να εξακολουθεί να είναι δυαδικό δένδρο αναζήτησης [6].

```
#include <cstddef>
#include <iostream>

struct node
{
    int key;
    node *left;
    node *right;
};

node *insert(node *tree, int key);
node *search(node *tree, int key);
node *remove(node *tree, int key);
void destroy(node *tree);
node *max(node *tree);
node *remove_max_node(node *tree, node *max_node);
node *min(node *tree);
void print_in_order(node *tree);
```

Κώδικας 4: header file για το δυαδικό δένδρο αναζήτησης (bst.hpp)

```
#include "bst.hpp"

using namespace std;

node *insert(node *tree, int key)
{
    if (tree == NULL)
    {
        node *new_tree = new node;
        new_tree->left = NULL;
        new_tree->right = NULL;
        new_tree->key = key;
        return new_tree;
```

```
14      }
15      if (key < tree->key)
16      {
17          tree->left = insert(tree->left, key);
18      }
19      else
20      {
21          tree->right = insert(tree->right, key);
22      }
23      return tree;
24  }
25
26  node *search(node *tree, int key)
27  {
28      if (tree == NULL)
29      {
30          return NULL;
31      }
32      else if (tree->key == key)
33      {
34          return tree;
35      }
36      else if (key < tree->key)
37      {
38          return search(tree->left, key);
39      }
40      else
41      {
42          return search(tree->right, key);
43      }
44  }
45
46  node *remove(node *tree, int key)
47  {
48      if (tree == NULL)
49      {
50          return NULL;
51      }
52      if (tree->key == key)
53      {
54          if (tree->left == NULL)
55          {
56              node *right_subtree = tree->right;
57              delete tree;
58              return right_subtree;
59          }
60          if (tree->right == NULL)
61          {
62              node *left_subtree = tree->left;
63              delete tree;
64              return left_subtree;
65          }
66          node *max_node = max(tree->left);
67          max_node->left = remove_max_node(tree->left, max_node);
68          max_node->right = tree->right;
69          delete tree;
70          return max_node;
71      }
72      else if (tree->key > key)
73      {
74          tree->left = remove(tree->left, key);
```

```
75      }
76      else
77      {
78          tree->right = remove(tree->right, key);
79      }
80      return tree;
81  }
82
83  void destroy(node *tree)
84  {
85      if (tree != NULL)
86      {
87          destroy(tree->left);
88          destroy(tree->right);
89          delete tree;
90      }
91  }
92
93  node *max(node *tree)
94  {
95      if (tree == NULL)
96      {
97          return NULL;
98      }
99      else if (tree->right == NULL)
100     {
101         return tree;
102     }
103     return max(tree->right);
104 }
105
106 node *remove_max_node(node *tree, node *max_node_tree)
107 {
108     if (tree == NULL)
109     {
110         return NULL;
111     }
112     if (tree == max_node_tree)
113     {
114         return max_node_tree->left;
115     }
116     tree->right = remove_max_node(tree->right, max_node_tree);
117     return tree;
118 }
119
120 node *min(node *tree)
121 {
122     if (tree == NULL)
123     {
124         return NULL;
125     }
126     else if (tree->left == NULL)
127     {
128         return tree;
129     }
130     return min(tree->left);
131 }
132
133
134 void print_in_order(node *tree)
135 {
```

```
136     if (tree != NULL)
137     {
138         if (tree->left != NULL)
139         {
140             print_in_order(tree->left);
141         }
142         cout << tree->key << " ";
143         if (tree->right != NULL)
144         {
145             print_in_order(tree->right);
146         }
147     }
148     else
149     {
150         cout << "EMPTY";
151     }
152 }
```

Κώδικας 5: source file για το δυαδικό δένδρο αναζήτησης (bst.cpp)

```
1  #include "bst.hpp"
2  #include <vector>
3  using namespace std;
4
5  void test_search(node *root_node, int key)
6  {
7      cout << "Searching for key " << key << ": ";
8      node *p = search(root_node, key);
9      if (p != NULL)
10     {
11         cout << "found (" << p->key << ")" << endl;
12     }
13     else
14     {
15         cout << "not found " << endl;
16     }
17 }
18
19 void test_min_max(node *root_node)
20 {
21     cout << "Minimum " << min(root_node)->key << " Maximum " << max(root_node)->key << endl;
22 }
23
24 int main()
25 {
26     node *root_node = NULL;
27     vector<int> v = {10, 6, 5, 8, 14, 11, 18};
28     for (int x : v)
29     {
30         if (root_node == NULL)
31         {
32             root_node = insert(root_node, x);
33         }
34         else
35         {
36             insert(root_node, x);
37         }
38     }
39     cout << "In-order Traversal ";
40     print_in_order(root_node);
41     cout << endl;
```

```
42      test_search(root_node, 11);
43      test_search(root_node, 13);
44      test_min_max(root_node);
45      cout << "Remove node 10 ";
46      root_node = remove(root_node, 10);
47      cout << endl << "In-order Traversal ";
48      print_in_order(root_node);
49      cout << endl;
50      destroy(root_node);
51  }
```

Κώδικας 6: Δοκιμή των συναρτήσεων του δυαδικού δένδρου αναζήτησης (bst_ex1.cpp)

Η μεταγλώττιση και η εκτέλεση του κώδικα γίνεται με τις ακόλουθες εντολές:

```
$ g++ -Wall -std=c++11 bst.cpp bst_ex1.cpp -o bst_ex1
$ ./bst_ex1
```

Η δε έξοδος που παράγεται είναι η ακόλουθη:

```
In-order Traversal 5 6 8 10 11 14 18
Searching for key 11: found (11)
Searching for key 13: not found
Minimum 5 Maximum 18
Remove node 10
In-order Traversal 5 6 8 11 14 18
```

Για να πραγματοποιηθεί η διαγραφή του κόμβου 10, εντοπίζεται ο κόμβος με τη μεγαλύτερη τιμή στο αριστερό υποδένδρο του κόμβου 10, που είναι ο 8 και ο κόμβος αυτός αφαιρείται από το δένδρο αντικαθιστώντας τον κόμβο 10.

5 Ισοζυγισμένα δυαδικά δένδρα αναζήτησης

Οι καλές επιδόσεις ενός δυαδικού δένδρου αναζήτησης χάνονται όταν το δένδρο δεν είναι ισοζυγισμένο (balanced), δηλαδή όταν υπάρχουν μονοπάτια από τη ρίζα προς τα φύλλα με μεγάλα βάθη ενώ άλλα μονοπάτια έχουν μικρά βάθη. Υπάρχουν διάφορες μορφές ισοζυγισμένων δένδρων με πλέον δημοφιλή τα κόκκινα-μαύρα δένδρα (red black trees) και τα AVL (Adelson, Velskii και Landis) δένδρα. Σε αυτά τα δένδρα πραγματοποιούνται ειδικές λειτουργίες (περιστροφές) έτσι ώστε κατά την εισαγωγή νέων τιμών στο δένδρο και τη διαγραφή τιμών από το δένδρο, τα βάθη των φύλλων του δένδρου εγγυημένα να διατηρούνται σε κοντινές τιμές μεταξύ τους. Ισχύει ότι τα AVL δένδρα είναι καλύτερα ισοζυγισμένα από τα κόκκινα-μαύρα δένδρα αλλά έχουν το μειονέκτημα της υψηλότερης υπολογιστικής επιβάρυνσης κατά την εισαγωγή και τη διαγραφή κόμβων.

6 Παραδείγματα

6.1 Παράδειγμα 1

Δεδομένου ενός δυαδικού δένδρου ζητείται η εκτύπωση όλων των διαδρομών από τη ρίζα του δένδρου μέχρι κάθε φύλλο. Για παράδειγμα, για το δένδρο του Σχήματος 2 το πρόγραμμα θα πρέπει να επιστρέψει ABDH, ABE, ACF και ACG.

```
1  #include "binary_tree.hpp"
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5
6  void print_vector(vector<string> previous_nodes)
7  {
8      for (string s : previous_nodes)
```

```
    {
        cout << s << " ";
    }
    cout << endl;
}

void print_tree(node *root_node, vector<string> previous_nodes)
{
    if (root_node == NULL)
    {
        print_vector(previous_nodes);
    }
    else
    {
        // cout << "call root node=" << root_node->key << " path=";
        // print_vector(previous_nodes);
        previous_nodes.push_back(root_node->key);
        if (root_node->left == NULL && root_node->right == NULL)
        {
            print_vector(previous_nodes);
        }
        else
        {
            if (root_node->left != NULL)
                print_tree(root_node->left, previous_nodes);
            if (root_node->right != NULL)
                print_tree(root_node->right, previous_nodes);
        }
    }
}

int main()
{
    node *root_node = NULL;
    vector<string> v = {"A", "B", "C", "D", "E", "F", "G", "H"};
    for (string x : v)
    {
        if (root_node == NULL)
            root_node = insert(root_node, x);
        else
            insert(root_node, x);
    }

    cout << "Level-order Traversal ";
    print_level_order(root_node);
    cout << endl;

    vector<string> path;
    print_tree(root_node, path);
}
```

Κώδικας 7: Λύση παραδείγματος 1 (lab09_ex1.cpp)

Η μεταγλώττιση και η εκτέλεση του κώδικα γίνεται με τις ακόλουθες εντολές:

```
$ g++ -Wall -std=c++11 binary_search.cpp lab09_ex1.cpp -o lab09_ex1
$ ./lab09_ex1
```

Η έξοδος που παράγεται είναι η ακόλουθη:

```
key A inserted (root of the tree)
key B inserted
key C inserted
```

```
key D inserted
key E inserted
key F inserted
key G inserted
key H inserted
Level−order Traversal A B C D E F G H
A B D H
A B E
A C F
A C G
```

6.2 Παράδειγμα 2

Δεδομένου ενός δυαδικού δένδρου ζητείται να πραγματοποιείται έλεγχος σχετικά με το εάν το δένδρο είναι δυαδικό δένδρο αναζήτησης ή όχι.

```
#include ”bst.hpp”
#include <vector>
using namespace std;

int is_bst(node *node)
{
    if (node == NULL)
    {
        return true;
    }
    if (node−>left != NULL && min(node−>left)−>key > node−>key)
    {
        return false;
    }
    if (node−>right != NULL && max(node−>right)−>key <= node−>key)
    {
        return false;
    }
    if (!is_bst(node−>left) || !is_bst(node−>right))
    {
        return false;
    }
    return true;
}

int main()
{
    node *root_node = NULL;
    vector<int> v = {10, 6, 5, 8, 14, 11, 18};
    for (int x : v)
    {
        if (root_node == NULL)
        {
            root_node = insert(root_node, x);
        }
        else
        {
            insert(root_node, x);
        }
    }
    cout << (is_bst(root_node) ? ”The tree is a BST” : ”The tree is not a BST”) << endl;
    // replacing root node with zero
    root_node−>key = 0;
    cout << (is_bst(root_node) ? ”The tree is a BST” : ”The tree is not a BST”) << endl;
}
```

Κώδικας 8: Λύση παραδείγματος 2 (lab09_ex2.cpp)

Η μεταγλώττιση και η εκτέλεση του κώδικα γίνεται με τις ακόλουθες εντολές:

```
$ g++ -Wall -std=c++11 bst.cpp lab09_ex2.cpp -o lab09_ex2
$ ./lab09_ex2
```

Η έξοδος που παράγεται είναι η ακόλουθη:

```
The tree is a BST
The tree is not a BST
```

7 Ασκήσεις

1. Να γράψετε πρόγραμμα που να εμφανίζει τους κόμβους ενός δυαδικού δένδρου κατά επίπεδα από κάτω προς τα πάνω και από αριστερά προς τα δεξιά. Δηλαδή στο δένδρο του Σχήματος 2 θα πρέπει οι κόμβοι να εμφανιστούν ως D,E,F,G,B,C,A.

2. Να γράψετε πρόγραμμα που να δημιουργεί από έναν ταξινομημένο πίνακα ακεραίων ένα δυαδικό δένδρο αναζήτησης. Να χρησιμοποιηθεί ο ακόλουθος αλγόριθμος:

   (α΄) Εύρεση του μεσαίου στοιχείου του πίνακα και ορισμός του ως ρίζα του δένδρου

   (β΄) Αναδρομική εκτέλεση για το αριστερό και το δεξιό μισό

      i. Εύρεση του μεσαίου στοιχείου του αριστερού μέρους και ορισμός του ως αριστερό παιδί της ρίζας του βήματος α'

      ii. Εύρεση του μεσαίου στοιχείου του δεξιού μέρους και ορισμός του ως δεξί παιδί της ρίζας του βήματος α'

Αναφορές

[1] Wikipedia, Tree (data structure), https://en.wikipedia.org/wiki/Tree_(data_structure)

[2] Binary Trees by Nick Parlante, http://cslibrary.stanford.edu/110/BinaryTrees.html

[3] Wikipedia, Polish Notation, https://en.wikipedia.org/wiki/Polish_notation

[4] Wikipedia, Reverse Polish Notation, https://en.wikipedia.org/wiki/Reverse_Polish_notation

[5] Tree Traversals (Inorder, Preorder and Postorder), https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/

[6] Alex Allain, Jumping into C++, cprogramming.com, Chapter 17 - Binary Trees, 2013