Δομές Δεδομένων και Αλγόριθμοι - Εργαστήριο 9
Δένδρα

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

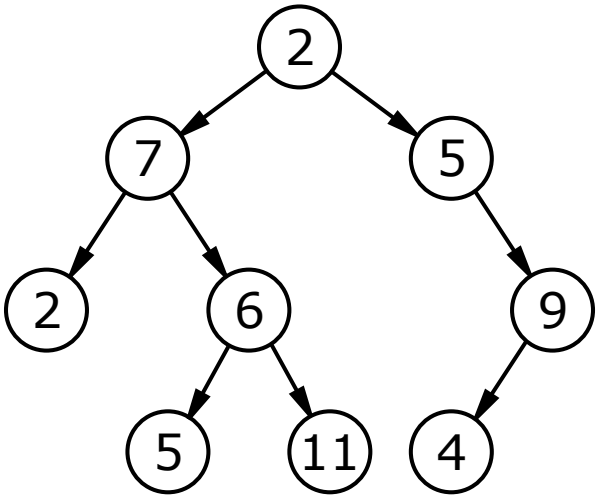
1 Εισαγωγή

Τα δένδρα όπως και τα γραφήματα είναι μη γραμμικές δομές δεδομένων. Τα δένδρα επιτρέπουν ιεραρχική οργάνωση των δεδομένων όπως φαίνεται στο Σχήμα 1.

2 Δένδρα

Ένα δένδρο αποτελείται από κόμβους που συνδέονται μεταξύ τους με ακμές. Ο πρώτος (υψηλότερος) κόμβος του δένδρου ονομάζεται ρίζα ενώ οι κόμβοι που βρίσκονται στα άκρα του δένδρου λέγονται φύλλα. Οι κόμβοι με τους οποίους συνδέεται απευθείας ένας κόμβος ονομάζονται παιδιά του κόμβου. Αντίστοιχα, ένας κόμβος που έχει παιδιά ονομάζεται γονέας των αντίστοιχων παιδιών-κόμβων. Απόγονοι ενός κόμβου είναι οι κόμβοι για τους οποίους υπάρχει διαδρομή πραγματοποιώντας διαδοχικές μεταβάσεις από γονείς σε παιδιά.

Τα δένδρα είναι αναδρομικές δομές από τη φύση τους. Κάθε κόμβος ενός δένδρου ορίζει έναν αριθμό από μικρότερα δένδρα, ένα για κάθε παιδί του.



Σχήμα 1: Ένα απλό δένδρο [1]

Ύψος δένδρου, Βάθος κορυφής

3 Δυαδικά δένδρα

Δυαδικό δένδρο είναι ένα δένδρο για το οποίο ισχύει ότι κάθε κόμβος έχει το πολύ δύο παιδιά [2].

3.1 Αναζήτηση κατά βάθος

3.1.1 pre-order DFS

3.1.2 in-order DFS

3.1.3 post-order DFS

3.2 Αναζήτηση κατά πλάτος

```
#include <cstddef>

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

void insert(node *&root_node, int 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;

void insert(node *&root_node, int key)
{
    if (root_node == NULL)
    {
        root_node = 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};
                }
```

```
35                      cout << "key " << key << " inserted" << endl;
36                      break;
37                  }
38              }
39          }
40      }
41
42      void print_level_order(node *root_node)
43      {
44          if (root_node == NULL)
45          {
46              return;
47          }
48          queue<node *> q;
49          q.push(root_node);
50          while (!q.empty())
51          {
52              node *node = q.front();
53              q.pop();
54              cout << node->key << " ";
55              if (node->left != NULL)
56              {
57                  q.push(node->left);
58              }
59              if (node->right != NULL)
60              {
61                  q.push(node->right);
62              }
63          }
64      }
65
66      void print_pre_order(node *root_node)
67      {
68          if (root_node != NULL)
69          {
70              cout << root_node->key << " ";
71              if (root_node->left != NULL)
72              {
73                  print_pre_order(root_node->left);
74              }
75              if (root_node->right != NULL)
76              {
77                  print_pre_order(root_node->right);
78              }
79          }
80          else
81          {
82              cout << "EMPTY";
83          }
84      }
85
86      void print_in_order(node *root_node)
87      {
88          if (root_node != NULL)
89          {
90              if (root_node->left != NULL)
91              {
92                  print_in_order(root_node->left);
93              }
94              cout << root_node->key << " ";
95              if (root_node->right != NULL)
```

```
96              {
97                  print_in_order(root_node->right);
98              }
99      }
100     else
101     {
102         cout << "EMPTY";
103     }
104 }
105
106 void print_post_order(node *root_node)
107 {
108     if (root_node != NULL)
109     {
110         if (root_node->left != NULL)
111         {
112             print_post_order(root_node->left);
113         }
114         if (root_node->right != NULL)
115         {
116             print_post_order(root_node->right);
117         }
118         cout << root_node->key << " ";
119     }
120     else
121     {
122         cout << "EMPTY";
123     }
124 }
125
126 void destroy(node *root_node)
127 {
128     if (root_node != NULL)
129     {
130         destroy(root_node->left);
131         destroy(root_node->right);
132         delete root_node;
133     }
134 }
```

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

```
1  #include "binary_tree.hpp"
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5
6  int main()
7  {
8      node *root_node = NULL;
9      vector<int> v = {10, 6, 5, 8, 14, 11, 18};
10     for (int x : v)
11     {
12         insert(root_node, x);
13     }
14
15     cout << "Level-order Traversal ";
16     print_level_order(root_node);
17     cout << endl;
18     cout << "Pre-order Traversal ";
19     print_pre_order(root_node);
```

```
20      cout << endl;
21      cout << "In-order Traversal ";
22      print_in_order(root_node);
23      cout << endl;
24      cout << "Post-order Traversal ";
25      print_post_order(root_node);
26      cout << endl;
27  }
```

Κώδικας 3: Δοκιμή των συναρτήσεων του δυαδικού δένδρου (lab09_ex1.cpp)

Η μεταγλώττιση και η εκτέλεση του κώδικα γίνεται με τις ακόλουθες εντολές:
Η δε έξοδος που παράγεται είναι η ακόλουθη:

4 Δυαδικά δένδρα αναζήτησης

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

4.1 Υλοποίηση δυαδικού δένδρου αναζήτησης

```
1  #include <cstddef>
2  #include <iostream>
3
4  struct node
5  {
6      int key;
7      node *left;
8      node *right;
9  };
10
11 node *insert(node *tree, int key);
12 node *search(node *tree, int key);
13 node *remove(node *tree, int key);
14 void destroy(node *tree);
15 node *max(node *tree);
16 node *remove_max_node(node *tree, node *max_node);
17 node *min(node *tree);
18 void print_in_order(node *tree);
```

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

```
1  #include "bst.hpp"
2
3  using namespace std;
4
5  node *insert(node *tree, int key)
6  {
7      if (tree == NULL)
8      {
9          node *new_tree = new node;
10         new_tree->left = NULL;
11         new_tree->right = NULL;
12         new_tree->key = key;
13         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 *tree, int key)
6  {
7      cout << "Searching for key " << key << ": ";
8      node *p = search(tree, 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 *tree)
20 {
21     cout << "Minimum " << min(tree)->key << " Maximum " << max(tree)->key << endl;
22 }
23
24 int main()
25 {
26     node *tree = NULL;
27     vector<int> v = {10, 6, 5, 8, 14, 11, 18};
28     for (int x : v)
29     {
30         if (tree == NULL)
31         {
32             tree = insert(tree, x);
33         }
34         else
35         {
36             insert(tree, x);
37         }
38     }
39     cout << "In-order Traversal ";
40     print_in_order(tree);
41     cout << endl;
42     test_search(tree, 11);
```

```
43      test_search(tree, 13);
44      test_min_max(tree);
45      destroy(tree);
46  }
```

Κώδικας 6: Δοκιμή των συναρτήσεων του δυαδικού δένδρου αναζήτησης (lab09_ex1.cpp)

Η μεταγλώττιση και η εκτέλεση του κώδικα γίνεται με τις ακόλουθες εντολές:
Η δε έξοδος που παράγεται είναι η ακόλουθη:

5 Παραδείγματα

5.1 Παράδειγμα 1

Δεδομένου ενός δυαδικού δένδρου και μιας τιμής SUM ζητείται να βρεθεί το εαν υπάρχει διαδρομή από τη ρίζα του δένδρου μέχρι κάποιο κόμβο που να έχει άθροισμα κλειδιών ίσο με την τιμή SUM.

5.2 Παράδειγμα 2

6 Ασκήσεις

1.

2.

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