Δομές Δεδομένων και Αλγόριθμοι (Γ' εξάμηνο)

Τμήμα Μηχανικών Πληροφορικής ΤΕ, ΤΕΙ Ηπείρου

Γκόγκος Χρήστος

04/12/2014

Άσκηση εργαστηρίου #6 (Υλοποίηση δυαδικού δένδρου αναζήτησης)

Στην εργασία αυτή παρουσιάζεται η υλοποίηση ενός δυαδικού δένδρου αναζήτησης. Θα χρησιμοποιηθεί η δομή `struct node`.

```
// Δομή κόμβου δένδρου
struct node {
	int key;
	struct node* left;
	struct node* right;
};
```

Η δημιουργία ενός νέου κόμβου γίνεται με τη συνάρτηση `new_node` και η προσθήκη του κόμβου στο δένδρο γίνεται με τη συνάρτηση `insert_node` που καλεί τη συνάρτηση `new_node`. Ο δε εντοπισμός ενός κόμβου στο δένδρο γίνεται με τη συνάρτηση `find`.

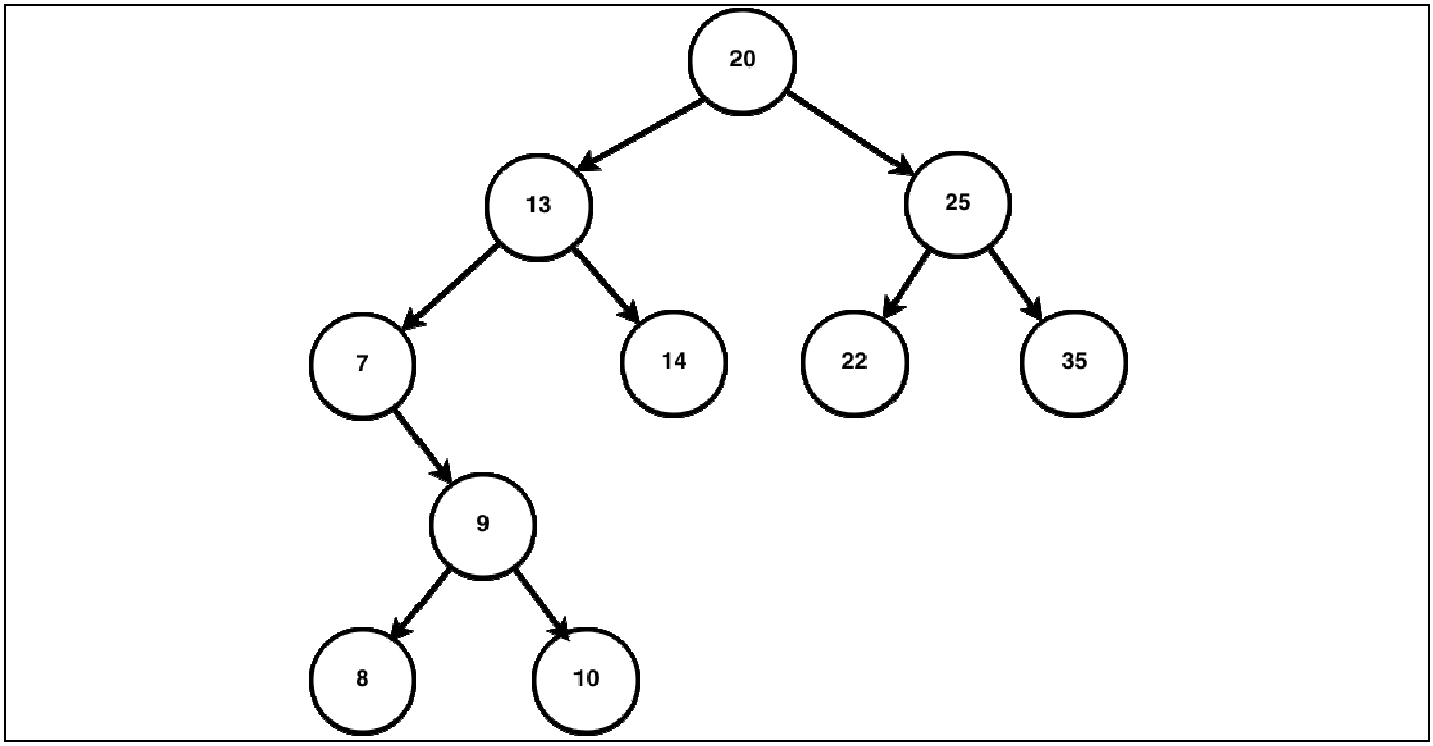
```
// Δημιουργία ενός νέου κόμβου
struct node* new_node(int key) {
	struct node* node = new (struct node);
	node->key = key;
	node->left = NULL;
	node->right = NULL;
	return (node);
}

// Προσθήκη του κόμβου στο δένδρο
struct node* insert_node(struct node* node, int key) {
	if (node == NULL) {
		return (new_node(key));
	} else {
		if (key == node->key) {
			cerr << "only distinct values are accepted!" << endl;
			return node;
		} else if (key < node->key)
			node->left = insert_node(node->left, key);
		else
			node->right = insert_node(node->right, key);
		return (node);
	}
}

// Εύρεση ενός κόμβου στο δένδρο
struct node* find(struct node* node, int key) {
	if (node == NULL) {
		return (NULL);
	} else {
		if (key == node->key)
			return (node);
		else if (key < node->key)
			return (find(node->left, key));
		else
```

```
                return (find(node->right, key));
        }
}
```

Για τη δημιουργία του ακόλουθου δένδρου και εν συνεχεία για τον εντοπισμό της τιμής 9 μπορεί να χρησιμοποιηθεί ο κώδικας που ακολουθεί.




```
#include <iostream>
#include <list>
using namespace std;

// ...

int main(int argc, char **argv) {
        list<int> data = { 20, 13, 7, 14, 9, 8, 10, 25, 22, 35 };
        struct node* root = NULL;
        for (int x : data) {
                root = insert_node(root, x);
        }

        struct node* a_node = find(root, 9);
        if (a_node == NULL)
                cout << "Δε βρέθηκε " << endl;
        else
                cout << "Βρέθηκε " << a_node->key << endl;
        // ...
}
```

Ερώτημα 1

Να προστεθούν οι ακόλουθες συναρτήσεις οι οποίες πραγματοποιούν διάσχιση του δένδρου preorder (προδιατεταγμένα), postorder (μεταδιατεταγμένα) και levelorder (διατεταγμένα κατά επίπεδα). Στη δεύτερη στήλη παρουσιάζεται η σειρά με την οποία θα εμφανιστούν τα δεδομένα για καθένα από τους τρόπους διάχισης του δένδρου.

`void preorder_traversal(struct node* node) {`
`  if (node == NULL)`
`  return;`
`  printf("%d ", node->key);`
`  preorder_traversal(node->left);`
`  preorder_traversal(node->right);`
`}`
	20 13 7 9 8 10 14 25 22 35

`void postorder_traversal(struct node* node) {`
`  if (node == NULL)`
`  return;`
`  postorder_traversal(node->left);`
`  postorder_traversal(node->right);`
`  printf("%d ", node->key);`
`}`
	8 10 9 7 14 13 22 35 25 20

`void levelorder_traversal(struct node* node) {`
`  queue<struct node*> q;`
`  q.push(node);`
`  while (!q.empty()) {`
`  struct node* tmp = q.front();`
`  printf("%d ", tmp->key);`
`  if (tmp->left != NULL)`
`  q.push(tmp->left);`
`  if (tmp->right != NULL)`
`  q.push(tmp->right);`
`  q.pop();`
`  }`
`}`
	20 13 25 7 14 22 35 9 8 10

Παρατήρηση: Για τη συνάρτηση `levelorder_traversal` θα χρειαστεί να δηλωθεί η βιβλιοθήκη `queue`.

Να υλοποιηθεί η συνάρτηση της ενδοδιατεταγμένης (inorder) διάσχισης του δένδρου. Τι παρατηρείτε για τον τρόπο με τον οποίο εμφανίζονται τα δεδομένα του δένδρου.

`void inorder_traversal(struct node* node) {`
 	
`  // να συμπληρωθεί`
 	
`}`	

Ερώτημα 2

Να προστεθούν οι ακόλουθες συναρτήσεις οι οποίες υπολογίζουν το ύψος του δένδρου και τη μικρότερη τιμή που είναι αποθηκευμένη στο δένδρο.

```
int height(struct node* node) {
	if (node == NULL) {
		return (-1);
	} else {
		int l_height = height(node->left);
		int r_height = height(node->right);
		if (l_height > r_height)
			return (l_height + 1);
		else
			return (r_height + 1);
	}
}
```

```
int min_value(struct node* node) {
	struct node* current = node;
	while (current->left != NULL) {
		current = current->left;
	}
	return (current->key);
}
```

Να υλοποηθεί η συνάρτηση υπολογισμού του πλήθους των κόμβων του δένδρου και η συνάρτηση υπολογισμού της μεγαλύτερης τιμής `max_value` που είναι αποθηκευμένη στο δένδρο.

```
int size(struct node* node) {

	// να συμπληρωθεί

}
```

```
int max_value(struct node* node) {

	// να συμπληρωθεί

}
```

Ερώτημα 3

Να προστεθεί η ακόλουθη συνάρτηση η οποία επιτρέπει τη διαγραφή ενός στοιχείου από το δυαδικό δένδρο αναζήτησης.

```
void delete_node(struct node* tree, int key) {
	struct node* ptr = tree;
	struct node* parent = NULL;
	bool turn_left = false;
	while (ptr != NULL && ptr->key != key) {
		parent = ptr;
		if (key < ptr->key) {
			turn_left = true;
			ptr = ptr->left;
		} else {
			turn_left = false;
			ptr = ptr->right;
		}
	}
	if (ptr == NULL) {
		cout << "Value " << key << " not found" << endl;
	} else if (ptr->left == NULL && ptr->right == NULL) {
		// case 1: leaf
		if (turn_left)
			parent->left = NULL;
		else
			parent->right = NULL;
		delete ptr;
	} else if (ptr->left == NULL) {
		// case 2: single subtree
		if (turn_left)
			parent->left = ptr->right;
		else
			parent->right = ptr->right;
		delete ptr;
	} else if (ptr->right == NULL) {
		if (turn_left)
			parent->left = ptr->left;
		else
			parent->right = ptr->left;
		delete ptr;
	} else {
		// case 3: two subtrees
		// replace node to be deleted with the largest value of its left subtree
		int max_value_left_subtree = max_value(ptr->left);
		struct node* tmp = ptr->left;
```

```
            while (tmp->right->key != max_value_left_subtree) {
                  tmp = tmp->right;
            }
            ptr->key = max_value_left_subtree;
            delete tmp->right;
            tmp->right = NULL;
      }
}
```

Να κληθεί η συνάρτηση `delete_node` από το κύριο πρόγραμμα και να συμπληρωθεί στον ακόλουθο πίνακα το τι θα εμφανίζεται στη διάσχιση του δένδρου κατά επίπεδα αν συμβεί η κάθε μία από τις ακόλουθες διαγραφές ξεχωριστά.

Διαγραφή του κλειδιού 22	`7 8 9 10 13 14 20 25 35`
Διαγραφή του κλειδιού 7	`20 13 25 9 14 22 35 8 10`
Διαγραφή του κλειδιού 13	`20 10 25 7 14 22 35 9 8`

Αναφορές

- http://cslibrary.stanford.edu/110/BinaryTrees.html#s1