Εργασία 4η - AVL Tree - Ατομική Εργασία

Εισαγωγή 1

Η εσωτερική κλάση Node 2

Η κλάση του δένδρου AVL 3

Η εσωτερική κλάση Iterator 6

Μέθοδοι της κλάση AVL που συνδέονται με την κλάση AVL::Iterator 7

Γενικές Παρατηρήσεις 7

Έλεγχος του προγράμματος σας 7

Τρόπος Αποστολής 8

Ατομική Εργασία, Καταληκτική ημερομηνία υποβολής: Κυριακή 13 Ιουνίου

Εισαγωγή

Σε αυτή την εργασία θα υλοποιήσετε ένα δένδρο AVL, το οποίο αποθηκεύει αντικείμενα τύπου **std::string**. Το header file της κλάσης **AVL** δίνεται παρακάτω:

```
#ifndef __AVL_HPP_
#define __AVL_HPP_

#include <iostream>
#include <fstream>

using namespace std;

class AVL {
private:
  class Node {
    Node *parent, *left, *right;
    int height;
    string element;

  public:
    Node(const string& e, Node *parent, Node *left, Node *right);

    Node*  getParent() const;
    Node*  getLeft() const;
    Node*  getRight() const;
    string getElement() const;
    int    getHeight() const;

    void setLeft(Node *);
    void setRight(Node *);
    void setParent(Node *);
    void setElement(string e);
```

```
    bool isLeft() const;
    bool isRight() const;
    int  rightChildHeight() const;
    int  leftChildHeight() const;
    int  updateHeight();
    bool isBalanced();
  };
private:

  int   size;
  Node* root;

public:

    class Iterator {

    public:
      Iterator& operator++();
      Iterator operator++(int a);
      string operator*();
      bool operator!=(Iterator it);
      bool operator==(Iterator it);
    };

  Iterator begin() const;
  Iterator end() const;

  static const int MAX_HEIGHT_DIFF = 1;
  AVL();
  AVL(const AVL& );
  bool contains(string e);
  bool add(string e);
  bool rmv(string e);
  void print2DotFile(char *filename);
  void pre_order(std::ostream& out);

  friend std::ostream& operator<<(std::ostream& out, const AVL& tree);
  AVL& operator  =(const AVL& avl);
  AVL  operator  +(const AVL& avl);
  AVL& operator +=(const AVL& avl);
  AVL& operator +=(const string& e);
  AVL& operator -=(const string& e);
  AVL  operator  +(const string& e);
  AVL  operator  -(const string& e);
};

#endif
```

Η εσωτερική κλάση Node

Προκειμένου να υλοποιήσει τον κόμβο του δένδρου, η κλάση **AVL** περιέχει την ***private*** εσωτερική κλάση **Node**, η οποία αντιπροσωπεύει τον κόμβο του δένδρου. Η κλάση **Node** έχει τα εξής *private* πεδία:

1. **AVL::Node* parent:** Δείκτης προς τον πατέρα του κόμβου.

2. **AVL::Node* left:** Δείκτης προς το αριστερό παιδί.
3. **AVL::Node* right:** Δείκτης προς το δεξί παιδί.
4. **int height:** Αντιπροσωπεύει το ύψος του κόμβου στο δένδρο. Κόμβοι που που είναι φύλλα του δένδρου έχουν ύψος 1.
5. **std::string element:** Ένα αντικείμενο τύπου **std::string** που αντιπροσωπεύει την πληροφορία που αποθηκεύεται στον κόμβο.

Η κλάση έχει κατ' ελάχιστο τους εξής κατασκευαστές και ***public*** μεθόδους (**μπορείτε να προσθέσετε και δικές σας μεθόδους και κατασκευαστές**)

Μέθοδος	Περιγραφή
`Node(const string& str, Node *parent, Node *left, Node *right);`	Κατασκευαστής της κόμβου. Αναθέτει στο πεδίο **element** την τιμή της παραμέτρου **str** και τα πεδία **parent**, **left**, **right** στις αντίστοιχες τιμές των παραμέτρων **parent**, **left**, **right**.
`Node *getParent() const;`	Επιστρέφει ένα δείκτη προς τον πατέρα του κόμβου.
`Node *getLeft() const;`	Επιστρέφει ένα δείκτη προς το αριστερό παιδί του κόμβου.
`Node *getRight() const;`	Επιστρέφει ένα δείκτη προς το δεξιό παιδί του κόμβου.
`string *getElement() const;`	Επιστρέφει το περιεχόμενο του κόμβου.
`int getHeight() const;`	Επιστρέφει το ύψος του κόμβου.
`void setLeft(Node *);`	Θέτει τον δείκτη προς το αριστερό παιδί του κόμβου.
`void setRight(Node *);`	Θέτει τον δείκτη προς το δεξί παιδί του κόμβου.
`void setParent(Node *);`	Θέτει τον δείκτη προς τον πατέρα παιδί του κόμβου.
`void setElement(string );`	Θέτει το πεδίο **element**.
`bool isLeft() const;`	Επιστρέφει ***true*** εάν ο κόμβος είναι αριστερό παιδί, διαφορετικά ***false***. Εάν πρόκειται για τη ρίζα επιστρέφει ***false***.
`bool isRight() const;`	Επιστρέφει ***true*** εάν ο κόμβος είναι δεξί παιδί, διαφορετικά ***false***. Εάν πρόκειται για τη ρίζα επιστρέφει ***false***.
`int rightChildHeight() const;`	Επιστρέφει την τιμή του ύψους του δεξιού παιδιού του κόμβου.
`int leftChildHeight() const;`	Επιστρέφει την τιμή του ύψους του αριστερού παιδιού του κόμβου.
`int updateHeight();`	Ενημερώνει το ύψος του κόμβου, με βάση το ύψος των δύο παιδιών του. Το ύψος του κόμβου προσδιορίζεται ως το μέγιστο εκ των υψών των δύο παιδιών του + 1. Επιστρέφει το νέο ύψος του κόμβου.
`bool isBalanced();`	Επιστρέφει **true**, εάν ο κόμβος είναι ισοζυγισμένος με βάση τη συνθήκη των AVL δένδρων.

Η κλάση του δένδρου AVL

Η κλάση του δένδρου έχει κατ' ελάχιστο τα εξής ***private*** πεδία (**μπορείτε να προσθέσετε και δικά σας**):

1. **int size:** Αντιπροσωπεύει τον αριθμό των κόμβων του δένδρου
2. **Node* root:** Δείκτης προς τη ρίζα του δένδρου

Η κλάση έχει κατ' ελάχιστο τους εξής κατασκευαστές και ***public*** μεθόδους (**μπορείτε να προσθέσετε και δικές σας μεθόδους και κατασκευαστές**)

Μέθοδος	Περιγραφή
`AVL()`	Κατασκευαστής της κλάσης χωρίς παραμέτρους.
`AVL(AVL& );`	Copy constructor. Η σειρά εισαγωγής των στοιχείων στο δένδρο είναι η PreOrder διάτρεξη του δένδρου που δίνεται ως παράμετρος.
`bool contains(string);`	Επιστρέφει ***true*** εάν υπάρχει το **string** στο δένδρο.
`bool add(string);`	Δημιουργεί ένα νέο κόμβο τον οποίο και ενθέτει στο δένδρο, εφόσον το **string** δεν υπάρχει ήδη σε αυτό. Στο τέλος της ένθεσης το δένδρο ισοζυγίζεται κατά AVL. Επιστρέφει **true** εάν η εισαγωγή ήταν επιτυχής. Εάν το περιεχόμενο του **string** υπήρχε στο δένδρο, δεν γίνεται εισαγωγή και επιστρέφει **false**.
`bool rmv(string);`	Εάν ο κόμβος υπάρχει διαγράφεται από το δένδρο. **Κατά τη διαγραφή, ο κόμβος προς διαγραφή ανταλλάσσεται με το αριστερότερο στοιχείο του δεξιού υποδένδρου.** Στο τέλος της διαγραφής το δένδρο ισοζυγίζεται κατά AVL. Εάν ο κόμβος που περιέχει την πληροφορία του δοθέντος **std::string** δεν υπάρχει, δεν γίνεται καμία ενέργεια και επιστρέφεται **false**. Εάν ο κόμβος βρεθεί και διαγραφεί με επιτυχία επιστρέφεται **true**.
`void print2DotFile(char *filename);`	Εκτυπώνει το περιεχόμενο του δένδρου σε αρχείο. Η εκτύπωση θα πρέπει να γίνει με τρόπο ώστε το περιεχόμενο του αρχείου να μπορεί να μετατραπεί σε εικόνα από το πρόγραμμα **dot** της σουίτας **graphviz**. Η σειρά εκτύπωσης δεν μας ενδιαφέρει. Η συνάρτηση θα χρησιμοποιηθεί για οπτικό έλεγχο των αποτελεσμάτων σας.
`void pre_order(std::ostream& out);`	Εκτυπώνει τα περιεχόμενα του δένδρου κατά **PreOrder**, στο **std::ostream** που δίνεται ως όρισμα. Η εκτύπωση για κάθε κόμβο έχει ως εξής: Εκτυπώνεται το περιεχόμενο του κόμβου ακολουθούμενο από κενό χαρακτήρα (εκτυπώνεται κενός χαρακτήρας, ακόμη και μετά τον τελευταίο προς εκτύπωση κόμβο).
`friend std::ostream& operator<<(std::ostream& out, const AVL& tree);`	**Φιλική συνάρτηση υπερφόρτωσης του τελεστή <<**. Εκτυπώνει στο **std::ostream** το περιεχόμενο του δένδρου κατά τη διάτρεξη **PreOrder**. Η εκτύπωση για κάθε κόμβο έχει ως εξής: Εκτυπώνεται το περιεχόμενο του κόμβου ακολουθούμενο από κενό χαρακτήρα (εκτυπώνεται κενός χαρακτήρας, ακόμη και μετά τον τελευταίο προς εκτύπωση κόμβο).
`AVL& operator =(const AVL& avl);`	**Υπερφόρτωση του τελεστή =**. Το δένδρο στα αριστερά του τελεστή περιέχει τα ίδια στοιχεία με το δένδρο στα δεξιά του τελεστή (όχι

	απαραίτητα με την ίδια σειρά). Εάν στο δένδρο στα αριστερά του τελεστή υπάρχουν στοιχεία, αυτά θα πρέπει να αφαιρεθούν. Η σειρά εισαγωγής των στοιχείων στο δένδρο στα αριστερά του τελεστή είναι η **PreOrder** διάτρεξη του δένδρου στα δεξιά. Επιστρέφεται μία αναφορά στο δένδρο στα αριστερά του τελεστή.
`AVL operator +(const AVL& avl);`	**Υπερφόρτωση του τελεστή +**. Δημιουργείται ένα νέο δένδρο που αποτελείται από το σύνολο των στοιχείων του δένδρου στα αριστερά του τελεστή με το δένδρο στα δεξιά του τελεστή. Η σειρά εισαγωγής των στοιχείων στο τελικό δένδρο είναι η εξής: 1. **PreOrder** διάτρεξη του δένδρου στα αριστερά του τελεστή. 2. **PreOrder** διάτρεξη του δένδρου στα δεξιά του τελεστή. Επιστρέφεται το νέο δένδρο.
`AVL& operator+=(const AVL& avl);`	**Υπερφόρτωση του τελεστή +=.** Στο δένδρο στα αριστερά του τελεστή προστίθενται τα στοιχεία του δένδρου στα δεξιά του τελεστή. Η σειρά εισαγωγής είναι η **PreOrder** διάτρεξη του δένδρου στα δεξιά του τελεστή. Επιστρέφεται μία αναφορά στο υφιστάμενο δένδρο.
`AVL& operator+=(const string& e);`	**Υπερφόρτωση του τελεστή +=**. Στο δένδρο στα αριστερά του τελεστή προστίθενται το **std::string e**, εφόσον αυτό δεν υπάρχει. Επιστρέφεται μία αναφορά στο υφιστάμενο δένδρο.
`AVL& operator-=(const string& e);`	**Υπερφόρτωση του τελεστή -=**. Από το δένδρο στα αριστερά του τελεστή αφαιρείται το **std::string e**, εφόσον αυτό υπάρχει. Κατά τη διαγραφή, ο κόμβος προς διαγραφή ανταλλάσσεται με το αριστερότερο στοιχείο του δεξιού υποδένδρου. Επιστρέφεται μία αναφορά στο υφιστάμενο δένδρο.
`AVL  operator +(const string& e);`	**Υπερφόρτωση του τελεστή +**. Δημιουργείται ένα νέο δένδρο που περιέχει τα στοιχεία του τρέχοντος δένδρου και το **std::string e**. Επιστρέφεται ένα νέο δένδρο που περιέχει τα στοιχεία του αρχικού δένδρου εισηγμένα κατά *pre-order*. Μετά την εισαγωγή όλων των στοιχείων, προσθέτουμε στο νέο δένδρο το στοιχείο **e**.
`AVL  operator -(const string& e);`	**Υπερφόρτωση του τελεστή -.** Δημιουργείται ένα νέο δένδρο που περιέχει τα στοιχεία του τρέχοντος δένδρου αφαιρώντας όμως το **std::string e**. Κατά τη διαγραφή, ο κόμβος προς διαγραφή ανταλλάσσεται με το αριστερότερο στοιχείο του δεξιού υποδένδρου. Επιστρέφεται ένα νέο δένδρο που περιέχει τα στοιχεία του αρχικού δένδρου εισηγμένα κατά *pre-order*. Μετά την εισαγωγή όλων των στοιχείων, αφαιρούμε από το νέο δένδρο το στοιχείο **e**.

Η εσωτερική κλάση Iterator

Η κλάση AVL διαθέτει την εσωτερική ***public*** κλάση **Iterator** για την διάτρεξη των κόμβων του δένδρου. Η διάτρεξη των κόμβων γίνεται κατά τη σειρά **PreOrder**. Τα πεδία και τους κατασκευαστές της κλάσης **Iterator** θα τα επιλέξετε εσείς.

Η κλάση έχει κατ' ελάχιστο τις εξής ***public*** μεθόδους (**μπορείτε να προσθέσετε και δικές σας private μεθόδους και κατασκευαστές**)

Μέθοδος	Περιγραφή
`Iterator& operator++();`	Προχωρά τον **Iterator** στην επόμενη θέση. Επιστρέφει μία αναφορά στον υφιστάμενο Iterator. Η επιλογή της επόμενης θέσης γίνεται κατά τη διάτρεξη **PreOrder**.
`Iterator operator++(int a);`	Προχωρά τον **Iterator** στην επόμενη θέση. Επιστρέφει ένα νέο Iterator που δείχνει στην προηγούμενη θέση. Η επιλογή της επόμενης θέσης γίνεται κατά τη διάτρεξη **PreOrder**.
`string operator*();`	Λαμβάνει το περιεχόμενο του κόμβου στον οποίο δείχνει ο Iterator.
`bool operator!=(Iterator it);`	Εξετάζει εάν ο Iterator στα αριστερά του τελεστή ΔΕΝ δείχνει στον ίδιο κόμβο με τον Iterator στα δεξιά του τελεστή.
`bool operator==(Iterator it);`	Εξετάζει εάν ο Iterator στα αριστερά του τελεστή δείχνει στον ίδιο κόμβος με τον Iterator στα δεξιά του τελεστή.

Ιδέες για το πως να υλοποιήσετε τη διάτρεξη του Iterator μπορείτε να βρείτε **εδώ** και **εδώ**. Θα χρειαστείτε ένα αντικείμενο τύπου **std::stack**.

Μέθοδοι της κλάση AVL που συνδέονται με την κλάση AVL::Iterator

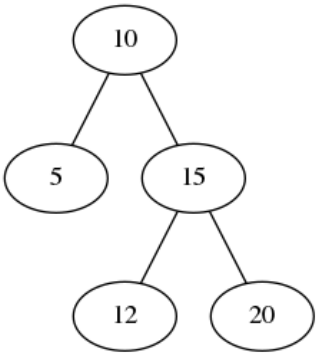
Μέθοδος	Περιγραφή
`AVL::Iterator begin() const;`	Επιστρέφει έναν Iterator που δείχνει στο 1ο στοιχείο του δένδρου κατά τη διάτρεξη **PreOrder**.
`AVL::Iterator end() const;`	Επιστρέφει έναν Iterator που δείχνει αμέσως μετά το τελευταίο στοιχείο του δένδρου κατά τη διάτρεξη **PreOrder**.

Γενικές Παρατηρήσεις

Θα πρέπει να έχετε υπόψη σας η διαπέραση ενός δένδρου κατά pre-order δίνει μοναδικό αποτέλεσμα για κάθε δυαδικό δένδρο. Δηλαδή, δεν μπορείτε να έχετε δύο διαφορετικά δέντρα με την ίδια pre-order διάτρεξη.

Όταν εισάγετε τα στοιχεία ενός δένδρου κατά τη διαπέραση pre-order σε ένα νέο αρχικά κενό AVL δένδρο, το δένδρο που θα προκύψει θα έχει τα ίδια στοιχεία, αλλά με μεγάλη πιθανότητα θα είναι διαφορετικό από το αρχικό και θα είναι πάντα το ίδιο όσες φορές και εάν επαναλάβετε τη διαδικασία.

1. Κατά τη διαγραφή ενός κόμβου, όταν αυτός έχει δύο παιδιά, επιλέγουμε ΠΑΝΤΑ να αντικαταστήσουμε τον κόμβο προς διαγραφή με το αριστερότερο στοιχείο του δεξιού υποδένδρου.
2. Κατά τη διαγραφή ενός κόμβου, όταν έχουμε την επιλογή να κάνουμε είτε απλή είτε διπλή περιστροφή, το είδος της περιστροφής που κάνουμε είναι ΠΑΝΤΑ απλή περιστροφή και όχι διπλή. Για παράδειγμα, στο διπλανό δένδρο η διαγραφή του 5 οδηγεί στην απλή περιστροφή (συμμετέχουν οι κόμβοι 10, 15, 20) και ΟΧΙ στη διπλή (συμμετέχουν οι κόμβοι 10, 15, 12). Τα test cases υπακούουν στο συγκεκριμένο κανόνα.



Έλεγχος του προγράμματος σας

Για τον έλεγχο του προγράμματος σας παρέχονται τα παρακάτω tests:

Test	Περιγραφή	Εξαρτάται από
test1	Ελέγχει τη συνάρτηση add και τη φιλική συνάρτηση υπερφόρτωσης του τελεστή **<<** `std::ostream& operator<<(std::ostream& out, const AVL& tree);`	-
test2	Ελέγχει τη συνάρτηση contains	**test1**
test3	Ελέγχει τη συνάρτηση rmv	**test1**
test4	Ελέγχει τις λειτουργίες της εσωτερικής κλάσης Iterator	**test1**
test5	Ελέγχει τον copy constructor	**test1**
test6	Ελέγχει τον τελεστή =	**test1, test3**
test7	Ελέγχει τον τελεστή + `AVL operator +(const AVL& avl);`	**test1, test6**
test8	Ελέγχει τον τελεστή += `AVL& operator+=(const AVL& avl);`	**test1**

test9	Ελέγχει τους τελεστές += και -=
	`AVL& operator+=(const string& e);`
	`AVL& operator-=(const string& e);`
		test1

test10	Ελέγχει τους τελεστές + και -
	`AVL operator +(const string& e);`
	`AVL operator -(const string& e);`
		test1

Τρόπος Αποστολής

Η εργασία θα εξεταστεί μέσω του εργαλείου αυτόματης υποβολής και διόρθωσης autolab. Οι οδηγίες αποστολής είναι οι εξής:

1. Πηγαίνετε στη σελίδα του autolab και συνδεθείτε με το username και το password σας (απαιτείται σύνδεση VPN).
2. Επιλέξτε το μάθημα **CE326_2021 (S21)** και στη συνέχεια την εργασία **HW4.**

3. Κατασκευάστε το φάκελο **hw4submit** τοπικά στον υπολογιστή σας.
4. Μέσα στον φάκελο αντιγράψτε τα αρχεία **AVL.cpp** και **AVL.hpp** της εργασίας σας.
5. Συμπιέστε και πακετάρετε τον κατάλογο **hw4submit** σε μορφή .tar.gz, κάνοντας δεξί click στον κατάλογο **hw4submit** και στη συνέχεια επιλέγοντας από το pop-up menu την επιλογή **Compress here as tar.gz**.