|  |
| --- |
| Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης – Τμήμα Πληροφορικής |
| Website links |
| Δομές Δεδομένων – Εργασία 1η |

|  |  |  |
| --- | --- | --- |
| Αναγνώστου Αντώνης | 2268 | [anagnoad@csd.auth.gr](mailto:anagnoad@csd.auth.gr) |
| Λασκαρίδης Στέφανος | 2315 | [laskstef@csd.auth.gr](mailto:laskstef@csd.auth.gr) |

# ­­­­

Περιεχόμενα

[Περιγραφή προβλήματος 2](#_Toc373937751)

[Υλοποίηση 3](#_Toc373937752)

[Sorted Array 3](#_Toc373937753)

[AVL Tree 3](#_Toc373937754)

[Περιγραφή κλάσεων 4](#_Toc373937755)

[TreeNode class 4](#_Toc373937756)

[Λειτουργικότητα κλάσης 4](#_Toc373937757)

[Υπολογιστικό κόστος 4](#_Toc373937758)

[AVL class 5](#_Toc373937759)

[Λειτουργικότητα κλάσης 5](#_Toc373937760)

[Node class 6](#_Toc373937761)

[Λειτουργικότητα κλάσης 6](#_Toc373937762)

[Database class 7](#_Toc373937763)

[Λειτουργικότητα κλάσης 7](#_Toc373937764)

[IO class 8](#_Toc373937765)

[Λειτουργικότητα κλάσης 8](#_Toc373937766)

[Input/Output 9](#_Toc373937767)

[Μορφή εντολών 9](#_Toc373937768)

[Μορφή input 10](#_Toc373937769)

[Μορφή output 10](#_Toc373937770)

# Περιγραφή προβλήματος

Η εργασία αυτή αφορά στην κατασκευή ενός αντεστραμμένου καταλόγου με χρήση ταξινομημένου πίνακα και δυναμικής διαχείρισης μνήμης. Συγκεκριμένα, δοθέντος ενός αρχείου εισόδου της μορφής

<ακέραιος a> <ακέραιος b>

το πρόγραμμα θα πρέπει να παράγει με έναν αποδοτικό αλγόριθμο ένα αρχείο εξόδου, που θα έχει την ακόλουθη μορφή:

<ακέραιος α>, <πλήθος k συνδεδεμένων κόμβων>, <ακέραιος 1>, <ακέραιος 2>, …. ,<ακέραιος k>

Μία ρεαλιστική εφαρμογή ενός τέτοιου προγράμματος, όπως αυτή περιγράφεται και στην εκφώνηση της παρούσας εργασίας, είναι η απεικόνιση του πλήθους και των συνδέσμων μιας σελίδας στο διαδίκτυο. Για παράδειγμα, αν ορίσουμε ένα μοναδικό αναγνωριστικό (ακέραιο αριθμό – ID) σε κάθε σελίδα, μπορούμε να αποτυπώσουμε με ακμές, σε ένα γράφημα το πλήθος των εξερχόμενων συνδέσμων. Για παράδειγμα, αν θεωρήσουμε σελίδες με id =1, 2 και 3, και θέλουμε να συνδέσουμε τις σελίδες 1 και 3 και τις σελίδες 2 και 3, το αρχείο input θα είχε την μορφή

1 3  
2 3

ενώ το παραγόμενο αρχείο θα είχε την ακόλουθη μορφή:

1, 1, 3  
2, 1, 3

Για την αρχικοποίηση αυτή του δένδρου διασυνδεδεμένων κόμβων, αλλά και την μετέπειτα επεξεργασία των ακμών, ορίζουμε τις εντολές READ\_DATA, INSERT\_LINK και DELETE\_LINK, η λειτουργία των οποίων θα εξεταστεί λεπτομερώς στην συνέχεια.

Γίνεται κατανοητό ότι το πρόγραμμα θα πρέπει να λειτουργεί για μεγάλο όγκο δεδομένων, καθώς και να παράγει το αρχείο εξόδου, με όσο το δυνατόν λιγότερους υπολογιστικούς πόρους.

# Υλοποίηση

Για την υλοποίηση του προβλήματος, αποφασίσαμε να χρησιμοποιήσουμε δύο δομές δεδομένων.

## Sorted Array

Για την αποθήκευση των κόμβων από τα οποία «φεύγουν» links προς άλλες σελίδες, χρησιμοποιείται ένα sorted array, δυναμικό σε μέγεθος. Το μέγεθός του διπλασιάζεται, κάθε φορά που γεμίζει.

Υλοποιείται με την κλάση Database, η οποία στο εσωτερικό της χρησιμοποιεί την κλάση TreeNode.

Ο λόγος για το οποίο επιλέχθηκε αυτή η δομή δεδομένων είναι διότι στις λειτουργίες που υλοποιούνται, επιτυγχάνει τον καλύτερο συνδυασμό, όσον αφορά τα κόστη. Οι υποστηριζόμενες λειτουργίες είναι οι παρακάτω:

|  |  |
| --- | --- |
| Λειτουργία: | Υπολογιστικό κόστος: |
| Αναζήτηση |  |
| Εισαγωγή νέου κόμβου |  |

**Σημείωση:** Στη συγκεκριμένη εργασία χρησιμοποιείται η κλάση με όνομα Database, αντί για όνομα ClassInvertedIndex, που ορίζει η εκφώνηση.

## AVL Tree

Κάθε γραμμή του array αυτού, περιέχει το id του κόμβου από το οποίο εξέρχονται links, το πλήθος των links το οποίο περιέχει καθώς και ένα δένδρο AVL το οποίο αποθηκεύει τους κόμβους στους οποίους οδηγούμαστε από την παραπάνω σελίδα.

Υλοποιείται με την κλάση AVL, η οποία στο εσωτερικό της χρησιμοποιεί τις κλάσεις Node, AVL, TreeNode.

Ο λόγος για τον οποίο επιλέχθηκε η συγκεκριμένη δομή για την αποθήκευση των συνδεδεμένων κόμβων, είναι για την επίτευξη των ελαχίστων χρόνων στις εξής λειτουργίες:

|  |  |
| --- | --- |
| Λειτουργία: | Υπολογιστικό κόστος: |
| Αναζήτηση |  |
| Εισαγωγή |  |
| Διαγραφή |  |
| Εκτύπωση inorder |  |

# Περιγραφή κλάσεων

## TreeNode class

|  |  |
| --- | --- |
| TreeNode | |
| public: | |
| TreeNode() | *Default constructor* |
| TreeNode(int value) | *Constructor που δέχεται αρχική τιμή για τον κόμβο του δένδρου* |
| virtual ~TreeNode() | *Destructor* |
| Int getHeight() | *Accessor που επιστρέφει την τιμή height του κόμβου* |
| TreeNode\* getLeft()  TreeNode\* getRight() | *Accessors που επιστέφουν pointer σε leftChild και rightChild αντίστοιχα* |
| void setLeft(TreeNode \* left)  void setRight(TreeNode \* right) | *Mutators που θέτουν τους pointers σε κάποιο TreeNode.* |
| void setValue(int idOfConnectedNode) | *Mutator που θέτει τιμή στο πεδίο value* |
| void fixHeight() | *Καλείται όταν χρειάζεται ανανέωση του ύψους του συγκεκριμένου κόμβου. (σε κάποιες εισαγωγές/διαγραφές)* |
| private: | |
| TreeNode\* leftChild | *Pointer τύπου treeNode που δείχνει στο αριστερό παιδί του κόμβου. Εάν δεν υπαρχει παιδί, έχει τιμή nullptr.* |
| TreeNode\* rightChild | *Pointer τύπου treeNode που δείχνει στο δεξί παιδί του κόμβου. Εάν δεν υπαρχει παιδί, έχει τιμή nullptr.* |
| int value | *Το id του ίδιου του κόμβου. Default value = -1.* |
| int height | *Πεδίο τύπου int στο οποίο αποθηκεύεται το ύψος του υποδένδρου το οποίο ορίζει ο συγκεκριμένος κόμβος. Default value = 1.* |

### Λειτουργικότητα κλάσης

Η κλάση ΤreeNode δημιουργήθηκε με σκοπό να αποθηκεύονται οι επιμέρους κόμβοι του δένδρου AVL ως αντικείμενά της.

Συνδέεται με την κλάση AVL με σχέση (has-a), δηλαδή ένα αντικείμενο AVL έχει αντικείμενα τύπου TreeNode.

Θεωρούμε πως η υλοποίηση αυτή συνδυάζει περισσότερα πλεονεκτήματα, έναντι ενός struct, καθώς υπάρχουν μέθοδοι οι οποίες ορίζονται και λειτουργούν πάνω στα δεδομένα των κόμβων. Επιπλέον, τηρείται, με αυτό τον τρόπο και η αρχή αντικειμενοστρεφούς προγραμματισμού της ενθυλάκωσης (encapsulation).

## AVL class

|  |  |
| --- | --- |
| AVL | |
| Public: | |
| AVL() | Default constructor |
| virtual ~AVL | Destructor |
| friend ostream& operator<< (ostream& myStream, AVL &obj) | Operator overload για την αποθήκευση του δένδρου στο αρχείο (καλεί αναδρομικά την printTree(treeNode\* root, ostream& myStream)). |
| Int getNumberOfLeaves() | Accessor για την επιστροφή του numberOfLeaves. |
| void insertTreeNode(int idOfConnectedNode) | Εισαγωγή νέου κόμβου στο δένδρο. Καλεί την insertTreeNode(int idOfConnectedNode, treeNode \* root) με root το head. |
| void deleteTreeNode(int idOfConnectedNode) | Διαγραφή κόμβου από το δένδρο. Καλεί την deleteTreeNode(int idOfConnectedNode, treeNode \* root) με root το head. |
| void printTree(treeNode\* root, ostream& myStream) | Καλείται αναδρομικά από το operator<< για την τύπωση του δένδρου (inorder) |
| Private: | |
| TreeNode \* head | Pointer σε object τύπου TreeNode το οποίο δείχνει στη ρίζα του δένδρου. Εάν το δένδρο είναι κενό, η τιμή του είναι nullptr. |
| int numberOfLeaves | Μεταβλητή που αποθηκεύει το πλήθος των κόμβων που περιέχει το δένδρο |
| TreeNode \* insertTreeNode(int idOfConnectedNode, treeNode \* root) | Συνάρτηση που καλείται αναδρομικά για την εισαγωγή νέου κόμβου στο δένδρο. |
| TreeNode \* deleteTreeNode(int idOfConnectedNode, treeNode \* root) | Συνάρτηση που καλείται αναδρομικά για τη διαγραφή ενός κόμβου από το δένδρο |
| Static int calculateBf(treeNode \* node) | Υπολογίζει και επιστρέφει το bf ενός κόμβου του δένδρου AVL. |
| Static TreeNode\* rotateLL(TreeNode \* node) | LL-rotation στο δένδρο με ρίζα node. |
| Static TreeNode\* rotateRR(TreeNode \* node) | RR-rotation στο δένδρο με ρίζα node. |
| Static TreeNode\* rotateLR(TreeNode \* node) | LR-rotation στο δένδρο με ρίζα node. |
| Static TreeNode\* rotateRL(TreeNode \* node) | RL-rotation στο δένδρο με ρίζα node. |
| Static TreeNode\* fixTree(TreeNode\* node) | Καλείται από την insertTreeNode για να φτιάξει τα ύψη των υποδένδρων, εκεί που χρειάζεται μετά την εισαγωγή νέου κόμβου. |

### Λειτουργικότητα κλάσης

Η συγκεκριμένη κλάση δημιουργήθηκε για να αποθηκεύει του κόμβους που συνδέονται με μία ιστοσελίδα. Υποστηρίζει εισαγωγή και διαγραφή κόμβων, όπως και print inOrder.

Η κλάση αυτη συνδέεται με σχέση (has-a) με την κλάση Node, δηλαδή ένα αντικείμενο τύπου Node έχει ένα αντικείμενο τύπου AVL ως πεδίο του.

Θεωρούμε πως η αποθήκευση των συνδεδεμένων κόμβων με AVL είναι η βέλτιστη, όσον αφορά το context της συγκεκριμένης εργασίας, λόγω της ελαχιστοποίησης του κόστους των υποστηριζόμενων λειτουργιών.

## Node class

|  |  |
| --- | --- |
| Node | |
| public: | |
| Node(); | *Default constructor. Δεσμεύεται χώρος για το AVL, και αρχικοποιείται το numberOfConnectedNodes στο 0. Ορίζεται σαν κατάσταση λάθους το id = -1, καθώς δεν έχει δοθεί κάτι άλλο από τον χρήστη* |
| Node(int id); | *Δεσμεύεται χώρος για το AVL, και αρχικοποιείται το numberOfConnectedNodes στο 0. Το id του node γίνεται ίσο με το όρισμα του constructor.* |
| Node(int id, AVL\* avlTree); | *Στο νέο node, περνιούνται με ορίσματα στον constructor ένας δείκτης σε υπάρχον AVL, καθώς και το id. Το numberOfConnectedNodes γίνεται ίσο με τα φύλλα του AVL που δόθηκε.* |
| virtual ~Node(); | *Αποδεσμεύει τον χώρο του AVL.* |
| int getID(); | *Επιστρέφει το id του node.* |
| int getNumberOfConnectedNodes(); | *Επιστρέφει τον αριθμό των συνδεδεμένων κόμβων.* |
| bool addNewNode(int idOfNewNode); | *Προσθέτει μία νέα σύνδεση, από το Node που βρισκόμαστε, στο node με το id που δίνεται σαν όρισμα.* |
| bool deleteNode(int idOfExisitingNode); | *Σβήνει μία υπάρχουσα σύνδεση με το node που έχει το id του ορίσματος (αν υπάρχει).* |
| friend ostream &operator<<(ostream &mystream, Node& obj); | *Operator overload.* |
| private: | |
| int id; | *To id του κόμβου που βρισκόμαστε.* |
| int numberOfConnectedNodes; | *Ο αριθμός των συνδέσεων του κόμβου αυτού με άλλους.* |
| AVL\* avlTree; | *Το AVL tree στο οποίο αποθηκεύονται οι συνδέσεις με άλλα Nodes.* |

### Λειτουργικότητα κλάσης

Πρόκειται για την κλάση των στοιχείων που βρίσκονται στο ταξινομημένο array της Database. Κάθε τέτοιο στοιχείο έχει ένα δένδρο AVL, για τις συνδέσεις με άλλα στοιχεία, ένα μοναδικό πρωτεύον κλειδί (id) καθώς και τον αριθμό των διασυνδεδεμένων κόμβων. Οι μέθοδοι addNewNode και deleteNode καλούνται από την Database, και αφαιρούν ή προσθέτουν ένα σύνδεσμο στο δένδρο AVL.

#### Υπολογιστικό κόστος

Καθώς η κλάση αυτή έχει μόνο δύο μεθόδους (εξαιρουμένων των get accessors) όποιο υπολογιστικό κόστος θα βρίσκεται σε αυτές τις δύο μεθόδους. Έτσι, για την εισαγωγή ενός συνδέσμου ή την διαγραφή αντίστοιχα, προκύπτει το αντίστοιχο κόστος για την εισαγωγή/διαγραφή στο avlTree, που είναι O(log(numberofConnectedNodes)).

## Database class

|  |  |
| --- | --- |
| Database | |
| public: | |
| Database(); | *Default constructor. Δεσμεύει αρχικά 16000 θέσεις στον πίνακα theDatabase, αρχικοποιεί το size στο 0 και το capacity στις 16.000.* |
| virtual ~Database(); | *Αποδεσμεύεται όλος ο χώρος της theDatabase.* |
| bool insertNewLink(int leftId, int rightId); | *Εισάγει νέο link μεταξύ της σελίδας με id= leftId και της σελίδας με id= rightId.* |
| bool deleteExistingLink(int leftId, int rightId); | *Σβήνει ένα υπάρχον link μεταξύ της σελίδας του leftId και της σελίδας του rightId.* |
| friend ostream &operator<<(ostream &mystream, Database& obj);  friend istream &operator>>(ostream &mystream, Database& obj); | *Operators overload (insertion, extraction)* |
| private: | |
| struct resultOfBinary  {  bool found;  int position;  };  typedef struct resultOfBinary; | *Τοπικός τύπος δεδομένων που εξυπηρετεί εσωτερικά στην κλάση, στην συνάρτηση binarySearch, προκειμένου να ξέρουμε αν και που βρέθηκε ένα συγκεκριμένο id μέσα στην database μας.* |
| Node\*\* theDatabase; | *Στην ουσία πρόκειται για ένα sorted array.* |
| int size; int capacity; | *Το size αποτυπώνει το “load” της database, δηλαδή το πόσα στοιχεία έχουν εισαχθεί μία δεδομένη στιγμή, ενώ το capacity αποτελεί την συνολική χωρητικότητα που υποστηρίζει η δομή δεδομένων.* |
| bool myRealloc(); | *Συνάρτηση για δέσμευση νέου χώρου, μεταφορά των στοιχείων και αποδέσμευση των παλιών θέσεων μνήμης για την database. Καλείται όταν size==capacity, δηλαδή όταν η database έχει γεμίσει.* |
| bool insertNode(Node nodeToBeInserted); | *Μέθοδος που καλείται τοπικά από την insertNewLink κατά την είσοδο ενός στοιχείου που δεν υπάρχει στο database.* |
| resultOfBinary binarySearch(int idToSearch); | *Δυαδική αναζήτηση στην database.* |
| Node\* searchNodeByID(int idToSearch); | *Αναζήτηση κόμβου στην database. Καλεί εσωτερικά την binarySearch, και επιστρέφει το αντικείμενο Node\* που αντιστοιχεί σε εκείνη την θέση του πίνακα.* |

### Λειτουργικότητα κλάσης

Πρόκειται στην ουσία για την κύρια κλάση της εφαρμογής. Αυτή διαχειρίζεται όλες τις εισαγωγές και τις διαγραφές των συνδέσμων των σελίδων και τον κόμβων. Με διαδοχικά abstractions, καλεί συναρτήσεις της κλάσης Node, και της AVL.

Διαχειρίζεται αποτελεσματικά και τις καταστάσεις λάθους, όπως για παράδειγμα εισαγωγή ενός συνδέσμου πολλές φορές, ή διαγραφή συνδέσμου που δεν υπάρχει.

## IO class

|  |  |
| --- | --- |
| IO | |
| public: | |
| IO();  virtual ~IO(); | *Default constructor/destructor. Καθώς πρόκειται για μία βοηθητική κλάση, δεν έχει πρακτικό νόημα η δημιουργία αντικειμένων.* |
| bool readCommands(char \* filename, Database \* db); | *Μέθοδος που καλείται προκειμένου να διαβαστεί το αρχείο commands.txt και να γίνει το parsing στις εντολές που δόθηκαν.* |
| bool readInput(char\* filename, Database \* db); | *Μέθοδος που καλείται προκειμένου να διαβαστεί το αρχείο input.txt και να φορτωθούν στο αντικείμενο της Database τα αρχικά links μεταξύ των node.* |
| bool writeIndex(char \* filename, Database \* db); | *Μέθοδος που καλείται προκειμένου να παραχθεί το αρχείο output.txt, που είναι και το τελικό αποτέλεσμα του προγράμματος.* |
| private: | |
| struct command  {  char\* commandName;  int argc;  char\* argv[2];  };  typedef struct command command; | *Struct που χρησιμοποιείται από την κλάση, για την αναπαράσταση μίας εντολής. Περιέχει μία συμβολοσειρά με το όνομα της εντολής, έναν ακέραιο αριθμό με το πλήθος των ορισμάτων (1 όταν πρόκειται για την WRITE\_INDEX και READ\_FILE, 2 για τις INSERT\_LINK και DELETE\_LINK). Τα ορίσματα αποθηκεύονται στον πίνακα argv.* |
| command parseLine(char input[]); | *Μέθοδος που κάνει parsing σε μία συμβολοσειρά, προκειμένου να αναγνωρίσει και να παράγει ένα αντικείμενο command.* |

### Λειτουργικότητα κλάσης

Πρόκειται για την κλάση που παρέχει την διασύνδεση των δομών μας με τον χρήστη. Μέσα από αυτή την κλάση, διαβάζουμε το αρχείο των εντολών (readCommands), εισάγουμε τα αρχικά δεδομένα της database (readInput) και εξάγουμε το τελικό αποτέλεσμα (writeIndex). Κάθε εντολή που διαβάζεται από το αρχείο κειμένου commands.txt, «ερμηνεύεται» από την parseLine, προκειμένου να εκτελεστεί η κατάλληλη λειτουργία που ορίζετε από την Database.

#### Υπολογιστικό κόστος

Το κόστος των συναρτήσεων της κλάσης αυτής, δεν παρουσιάζει ιδιαίτερο ενδιαφέρον, αφού δεν υπάρχει κάποιο αξιοποιήσιμο περιθώριο βελτίωσης ή ελαχιστοποίησης αυτού. Στις μεθόδους readCommands, readInput και writeIndex, το εκάστοτε κόστος θα είναι γραμμικό ( O(n) ). Πχ. για τη readCommands θα διαβαστούν n γραμμές από το commands.txt, για την readInput θα διαβαστούν n γραμμές από το input.txt ενώ για το writeIndex θα αποθηκευτούν τα δεδομένα από n στοιχεία που βρίσκονται στην database την στιγμή κλήσης της.

# Input/Output

## Μορφή εντολών

Οι εντολές που δίνονται από τον χρήστη, για την εκτέλεση του προγράμματος, βρίσκονται αποθηκευμένες σε ένα αρχείο με όνομα commands.txt στο φάκελο του εκτελέσιμου αρχείου. Κατά την εκκίνηση του προγράμματος, προσπελαύνονται σειρά ανά σειρά οι εντολές αυτές, και εκτελείται ο κατάλληλος κώδικας.

Οι εντολές που υποστηρίζονται από το πρόγραμμα είναι οι ακόλουθες:

* READ\_DATA <όνομα αρχείου>

Πρόκειται ουσιαστικά για την εντολή αρχικοποίησης με δεδομένα των δομών του προγράμματος. Στο αρχείο που δίνεται σαν όρισμα από τον χρήστη, βρίσκονται αποθηκευμένες, οι ακμές των συνδεδεμένων κόμβων. Το όνομα αρχείου δίνεται ως relative path και αναλόγως του λειτουργικού, ο χρήστης πρέπει να θέσει στο filename το σωστό path. (π.χ. windows: .\myFiles\input.txt, linux: ./myFiles/input.txt, os x: ./myFiles/input.txt)

* INSERT\_LINK <ακέραιος α> <ακέραιος β>

Με την εντολή αυτή, ο χρήστης προσθέτει μία ακόμα ακμή – σύνδεσμο στην δομή, από την σελίδα με μοναδικό αναγνωριστικό α στην σελίδα με μοναδικό αναγνωριστικό β.   
  
Αν η ακμή υπάρχει ήδη, τότε το πρόγραμμα συνεχίζει την εκτέλεση του, χωρίς να έχει πραγματοποιήσει καμία παραπάνω εντολή. Αν πρόκειται για έναν κόμβο α που δεν υπάρχει στην δομή του Database, τότε δημιουργείται και εισάγεται στον ταξινομημένο πίνακα.

* DELETE\_LINK <ακέραιος α> <ακέραιος β>  
    
  Με την εντολή αυτή, ο χρήστης αφαιρεί μία ακμή – σύνδεσμο στην δομή. Συγκεκριμένα αφαιρεί τον σύνδεσμο που ξεκινάει από την σελίδα με id = α και καταλήγει στην σελίδα με id = β. Αν η ακμή αυτή δεν υπάρχει, τότε το πρόγραμμα συνεχίζει κανονικά την εκτέλεση του, χωρίς να έχει πραγματοποιήσει καμία παραπάνω εντολή.
* WRITE\_INDEX <όνομα αρχείου>  
    
  Αποτελεί ίσως την πιο σημαντική εντολή στην εκτέλεση του προγράμματος, αφού χωρίς αυτή την εντολή, οποιαδήποτε δομή και αν δημιουργήθηκε στο πρόγραμμα, καταστρέφεται, προτού δοθεί η ζητούμενη έξοδος στον χρήστη.  
  Σε αντιστοιχία με την READ\_DATA, το όνομα του αρχείου προς αποθήκευση των δεδομένων δίνεται ως relative path.

Αξίζει στο σημείο αυτό, να σημειωθεί ότι για λόγους απόδοσης παραλήφθηκαν οι έλεγχοι εγκυρότητας για την μορφή των παραπάνω εντολών. Για τον λόγο αυτό, οι εντολές INSERT\_LINK και DELETE\_LINK θα πρέπει πάντα να ακολουθούνται από 2 ακέραιους αριθμούς, χωρισμένων με ένα κενό, ενώ οι εντολές READ\_DATA και WRITE\_INDEX θα πρέπει να ακολουθούνται από ένα όνομα αρχείου προσβάσιμο κατά την εκτέλεση του προγράμματος.

**Σημείωση**: Η διάκριση των εντολών (READ\_DATA, INSERT\_LINK, DELETE\_LINK, WRITE\_INDEX) από τις παραμέτρους τους, πρέπει να γίνεται με χρήση **ενός** κενού διαστήματος. *(delimeter = ‘ ’)*

## Μορφή input

Το κείμενο στο αρχείο input.txt που δίνεται σαν όρισμα όταν καλείται η λειτουργία READ\_DATA του προγράμματος έχει, όπως προείπαμε την μορφή:

*<ακέραιος a1> <ακέραιος α2>  
<ακέραιος β1> <ακέραιος β2>  
……*

*<ακέραιος κ1> <ακέραιος κ2>*

όπου κ οι γραμμές του αρχείου.

Όσον αφορά την δομή των γραμμών, ο χρήστης θα πρέπει να εισάγει άρτιο πλήθος αριθμών (id σελίδων), καθώς το πρόγραμμα αντιλαμβάνεται δυάδες από id, προκειμένου να εισάγει την αντίστοιχη ακμή. Ανάμεσα στους αριθμούς, δεν έχει καμία ιδιαίτερη βαρύτητα ο αριθμός των white spaces που θα χρησιμοποιηθούν από τον χρήστη. Ωστόσο, η εισαγωγή μη αριθμητικών χαρακτήρων, θα έχει ως αποτέλεσμα την αστάθεια του προγράμματος.

Σελίδες με αρνητικά id είναι επίσης αποδεκτές από το πρόγραμμα (δηλαδή όταν προκύπτει κάποιος αρνητικός ακέραιος στην προσπέλαση του αρχείου), χωρίς ωστόσο να γίνεται προφανής η χρήση και το νόημα μίας τέτοιας ακμής.

## Μορφή output

Για την αποθήκευση των δεδομένων του προγράμματος, χρησιμοποιούνται οι υπερφορτωμένοι τελεστές των κλάσεων Database, Node και AVL.

Αναλόγως με το filename το οποίο έχει δοθεί στο commands.txt, αποθηκεύει τα δεδομένα της Database στο αρχείο “filename”.

Η μορφή του αρχείου κειμένου είναι ως εξής

*<Node.Id\_1>, <NumberOfConnections>, <AVL\_in\_order>*

*<Node.Id\_2>, <NumberOfConnections>, <AVL\_in\_order>*

*…*

*<Node.Id\_N>, <NumberOfConnections>, <AVL\_in\_order>*

Όπου N το πλήθος των κόμβων που έχουμε αποθηκεύσει στη Database.