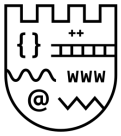
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Σχολή Θετικών Επιστημών



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δομές Δεδομένων

Εργασία 2021-2022

Ονόματα	*ΑΕΜ*
Δημήτριος Αθανασιάδης	*3724*
Νικόλαος Μπασδάνης	*3458*

Διδάσκων

Απόστολος Παπαδόπουλος

Περιεχόμενα

Προδιαγραφές

Για την εργασία ζητήθηκε αρχικά να υλοποιηθούν σε ***C++*** οι παρακάτω δομές δεδομένων:

- μη ταξινομημένος πίνακας
- ταξινομημένος πίνακας
- απλό δυαδικό δένδρο αναζήτησης
- δυαδικό δένδρο αναζήτησης τύπου AVL
- πίνακας κατακερματισμού με ανοικτή διεύθυνση

Σκοπός της εργασίας είναι οι παραπάνω δομές να αποθηκεύουν τα ζεύγη συνεχόμενων λέξεων ενός αρχείου κειμένου που δίνεται ως είσοδος, καθώς και το πλήθος εμφανίσεων αυτών. Στη συνέχεια ζητείται να συγκριθεί η απόδοση τους κατά την αναζήτηση ενός συνόλου που αποτελείται από ζεύγη συνεχόμενων λέξεων, τα οποία δεν είναι κατ' ανάγκη διαφορετικά. Οι πληροφορίες που πρέπει να αποθηκεύονται σε ένα αρχείο εξόδου για την κάθε δομή είναι ο χρόνος κατασκευής, ο χρόνος αναζήτησης του συνόλου ζευγών λέξεων και το πλήθος εμφανίσεων του κάθε ζεύγους που αναζητήθηκε.

Σύντομη περιγραφή λειτουργίας

Για εύκολο compilation σε unix-based λειτουργικά συστήματα, δημιουργήθηκε το *shell script* ***compile.sh***, το οποίο όταν εκτελεστεί παράγει το εκτελέσιμο αρχείο ***project***. Το ***project*** είναι ένα *CLI (Command Line Application)* το οποίο δέχεται σαν είσοδο ένα αρχείο κειμένου *filename* ως *command line argument*. Ακολουθεί η ενδεικτική χρήση του ***project***:

usage: ./project [filename]

Κατά την εκτέλεση του ***project*** όλα τα ζεύγη λέξεων του αρχείου εισόδου εισάγονται σε μία προσωρινή απλή βοηθητική δομή δεδομένων, την ***SimpleArray***, η οποία περιγράφεται αναλυτικά παρακάτω. Πριν από την εισαγωγή κάθε ζεύγους λέξεων στην ***SimpleArray***, προηγείται μια προεπεξεργασία των λέξεων, η οποία περιλαμβάνει την μετατροπή όλων των γραμμάτων από κεφαλαία σε πεζά και την αφαίρεση των σημείων στίξης. Αυτή η προεπεξεργασία γίνεται μέσα στη main, μέσω της συνάρτησης *format_word****(...)***, και έχει ως σκοπό την αποφυγή αποθήκευσης όμοιων όρων. Αφού εισαχθούν όλα τα ζεύγη στην ***SimpleArray*** (η ***SimpleArray*** επιτρέπεται να περιλαμβάνει και διπλότυπα ζεύγη), δημιουργούνται οι πέντε ζητούμενες δομές μία-μία, προσπελάζοντας σειριακά την ***SimpleArray*** και εισάγωντας τα υπάρχοντα ζεύγη συνεχόμενων λέξεων σε αυτές (οι ζητούμενες δομές δεδομένων περιέχουν μόνο τα μοναδικά ζεύγη συνεχόμενων λέξεων, δηλαδή όχι διπλότυπα, και το πλήθος εμφανίσεων τους). Χρησιμοποιώντας τη βιβλιοθήκη ***chrono*** υπολογίζεται ο χρόνος κατασκευής της κάθε δομής. Αφού κατασκευαστούν οι δομές δεδομένων, επιλέγονται με τυχαίο τρόπο ***Q_SIZE*** (το ***Q_SIZE*** ορίζεται στην αρχή του αρχείου *main.cpp)* ζεύγη λέξεων (όχι κατ' ανάγκη διαφορετικά) από την πρώτη βοηθητική δομή ***SimpleArray***, και δημιουργείται με αυτά μια νέα απλή βοηθητική δομή ***SimpleArray*** για να εκτελεστούν οι αναζητήσεις. Στη συνέχεια αναζητείται το καθέ ένα από τα τυχαία ζεύγη της νέας ***SimpleArray*** στην κάθε κατασκευασμένη δομή, ενώ ταυτόχρονα και πάλι με τη βοήθεια της βιβλιοθήκης ***chrono*** υπολογίζεται ο χρόνος που απαιτείται. Τέλος, αφού ολοκληρωθούν και οι αναζητήσεις, αποθηκεύονται τα αποτελέσματα στο αρχείο εξόδου ***OUTFILE*** (το ***OUTFILE*** ορίζεται στην αρχή του αρχείου *main.cpp*) και το πρόγραμμα τερματίζει την λειτουργία του.

Βοηθητικές κλάσεις

Κλάση ζεύγους (Pair)

Για την αποθήκευση του κάθε ζεύγους συνεχόμενων λέξεων χρησιμοποιήθηκε η κλάση ***Pair*** (η υλοποίηση βρίσκεται στα αρχεία *Pair.h* και *Pair.cpp*). Η κλάση ***Pair*** περιέχει τις παρακάτω ιδιότητες και μεθόδους:

Ιδιότητες	***Περιγραφή***
string word1	*Πρώτη λέξη του ζεύγους*
string word2	*Δεύτερη λέξη του ζεύγους*
int occurrences	*Πλήθος εμφανίσεων του ζεύγους*

Μέθοδοι	***Περιγραφή***
Pair(string word1, string word2)	*Constructor της κλάσης*
string getWord1( )	*Getter της word1*
string getWord2( )	*Getter της word2*
int get_occurrences( )	*Getter της occurrences*
void setWord1(string word1)	*Setter της word1*
void setWord2(string word2)	*Setter της word2*
void increment_occurrences( )	*Αύξηση του πλήθους εμφανίσεων του ζεύγους*
bool operator==(Pair pair)	*Τελεστής **==** για έλεγχο ισότητας δύο ζευγών*
bool operator>(Pair pair)	*Τελεστής **>** για έλεγχο ανισότητας δύο ζευγών (μεγαλύτερο)*
bool operator<(Pair pair)	*Τελεστής **>** για έλεγχο ανισότητας δύο ζευγών (μικρότερο)*
friend ostream &operator<<(ostream &os, Pair &pair)	*Τελεστής **<<** για εκτύπωση ιδιοτήτων ενός ζεύγους στο out stream με τη μορφή: word1 word2 [occurences]*

Κλάση βοηθητικού πίνακα (SimpleArray)

Η βοηθητική κλάση ***SimpleArray*** (η υλοποίηση βρίσκεται στα αρχεία *SimpleArray.h* και *SimpleArray.cpp*), όπως αναφέρθηκε και παραπάνω, χρησιμοποιείται για την προσωρινή αποθήκευση όλων των ζευγών (περιλαμβάνει διπλότυπα), όπως επίσης και για την αποθήκευση των τυχαίων ζευγών λέξεων που πρόκειται να αναζητηθούν. Η κλάση ***SimpleArray*** περιέχει τις παρακάτω ιδιότητες και μεθόδους:

Ιδιότητες	***Περιγραφή***
*Pair** array*	*Πίνακας που περιέχει pointers για αντικείμενα της κλάσης Pair*
long size	*Το μέγεθος του array*
long next_free_slot	*Το index της επόμενης ελεύθερης θέσης στο array*

Μέθοδοι	***Περιγραφή***
SimpleArray(long size)	*Constructor της κλάσης, κάνει initialize ένα νέο array μεγέθους size*
~SimpleArray( );	*Destructor της κλάσης*
bool insert(Pair pair);*	*Προσθέτει το ζεύγος pair στο array*
void print( )	*Εμφανίζει όλα τα ζέυγη*

Κατά την εκτέλεση του ***project***, *δημιουργείται* στο μέσω του constructor της ***SimpleArray*** ένα νέο *array* αρχικού μεγέθους 1000 *pointers* για αντικείμενα τύπου ***Pair***. Σε περίπτωση που το *array* είναι γεμάτο και χρειάζεται να προστεθεί ένα νέο ζεύγος λέξεων, μέσω της ***insert(…)*** δημιουργείται ένα νέο *array* διπλάσιου μεγέθους, αντιγράφονται τα περιεχόμενα του παλιού *array* στο νέο, και στη συνέχεια προστίθεται το καινούργιο ζεύγος στο νέο *array*.

Κλάσεις δομών δεδομένων

Μη ταξινομημένος πίνακας (UnorderedArray)

Η υλοποίηση της δομής δεδομένων του μη ταξινομημένου πίνακα έγινε μέσω της κλάσης ***UnorderedArray*** και βρίσκεται στα αρχεία *UnorderedArray.h* και *UnorderedArray.cpp*. Στην κλάση ***UnorderedArray*** παρατηρούμε κοινά στοιχεία με την κλάση ***SimpleArray*** που περιγράφηκε παραπάνω. Η ***UnorderedArray***, όπως και όλες οι ζητούμενες δομές δεδομένων, περιέχουν μοναδικά ζεύγη λέξεων. Συγκεκριμένα, η κλάση ***UnorderedArray*** έχει τις παρακάτω ιδιότητες και μεθόδους:

Ιδιότητες	***Περιγραφή***
*Pair** array*	*Αταξινόμητος πίνακας που περιέχει pointers για αντικείμενα της κλάσης Pair*
long size	*Το μέγεθος του array*
long next_free_slot	*Το index της επόμενης ελεύθερης θέσης στο array*

Μέθοδοι	***Περιγραφή***
UnorderedArray(long size)	*Constructor της κλάσης, κάνει initialize ένα νέο array μεγέθους size*
~UnorderedArray( )	*Destructor της κλάσης*
bool insert(Pair pair)*	*Προσθέτει το ζεύγος pair στο array*
long search(Pair pair)*	*Αναζητεί σειριακά το ζεύγος pair στο array*
bool remove(Pair pair)*	*Αφαιρεί το ζεύγος pair από το array*
Pair get_pair(long index)*	*Επιστρέφει ένα δείκτη για το Pair που βρίσκεται στη θέση index του array*
void print( )	*Εμφανίζει όλα τα ζεύγη*

insert(…): Κατά την εισαγωγή ενός νέου ζεύγους στη δομή, αρχικά ελέγχεται εάν το ζεύγος έχει ήδη εισαχθεί αναζητώντας το σειριακά. Σε περίπτωση που το ζεύγος βρίσκεται ήδη στο *array*, αυξάνεται το πλήθος εμφανίσεων του κατά ένα, αλλιώς προστίθεται στο *array* ως καινούργιο.

search*(…)*: Αναζητεί σειριακά το ζεύγος στο αταξινόμητο array. Σε περίπτωση που βρεθεί επιστρέφει την θέση του στο *array*, αλλιώς επιστρέφεται *-1*.

Ταξινομημένος πίνακας (OrderedArray)

Η υλοποίηση της δεύτερης ζητούμενης δομής του ταξινομημένου πίνακα βρίσκεται στην κλάση **OrderedArray** (στα αρχεία *OrderedArray.h* και *OrderedArray.cpp*). Η **OrderedArray** έχει κοινά στοιχεία με την ***UnorderedArray***, με την βασική διαφορά ότι το *array* της **OrderedArray** είναι ταξινομημένο. Η κλάση **OrderedArray** έχει τις παρακάτω ιδιότητες και μεθόδους:

Ιδιότητες	***Περιγραφή***
*Pair** array*	*Ταξινομημένος πίνακας που περιέχει pointers για αντικείμενα της κλάσης Pair*
long size	*Το μέγεθος του array*
long used_slots	*Αριθμός χρησιμοποιημένων θέσεων του array*

Μέθοδοι	***Περιγραφή***
OrderedArray(long size)	*Constructor της κλάσης, κάνει initialize ένα νέο array μεγέθους size*
~OrderedArray( )	*Destructor της κλάσης*
bool insert(Pair pair)*	*Προσθέτει το ζεύγος pair στο array*
long search(Pair pair, long start, long end)*	*Αναζητεί δυαδικά το ζεύγος pair στο array*
bool remove(Pair pair)*	*Αφαιρεί το ζεύγος pair από το array*
Pair get_pair(long index)*	*Επιστρέφει ένα δείκτη για το Pair που βρίσκεται στη θέση index του array*
long get_used_slots( )	*Getter της used slots*
void print( )	*Εμφανίζει όλα τα ζεύγη*

insert(…): Κατά την εισαγωγή ενός νέου ζεύγους στη δομή, αρχικά ελέγχεται εάν το ζεύγος έχει ήδη εισαχθεί αναζητώντας το δυαδικά. Σε περίπτωση που το ζεύγος βρίσκεται ήδη στο *array*, αυξάνεται το πλήθος εμφανίσεων του κατά ένα, αλλιώς προστίθεται στην κατάλληλη θέση του *array* (λεξικογραφικά) ως καινούργιο.

search*(…)*: Εκτελεί δυαδική αναζήτηση του ζεύγους στο *array*. Σε περίπτωση που το ζεύγος βρεθεί επιστρέφεται η θέση του, αλλιώς επιστρέφεται *-1*.

Απλό δυαδικό δέντρο αναζήτησης (BSTree)

Το *απλό δυαδικό δέντρο αναζήτησης* ή αλλιώς *binary search tree* αποτελεί μια ειδική περίπτωση δυαδικού δέντρου το οποίο χρησιμοποιείται κυρίως για αναζήτηση. Επιπλέον, υποστηρίζει εισαγωγές και διαγραφές κλειδιών - ζευγών λέξεων. Κάθε εσωτερικός κόμβος του δυαδικού δέντρου έχει από μηδέν εώς δύο (κανένα, μόνο αριστερό, μόνο δεξί ή και τα δύο) παιδιά - υπόδεντρα. Το κάθε node του δέντρου περιέχει ένα ζεύγος λέξεων το οποίο είναι λεξικογραφικά μεγαλύτερο από όλα τα ζεύγη λέξεων στο αριστερό υπόδεντρο (αν υπάρχει) και λεξικογραφικά μικρότερο από όλα τα ζεύγη λέξεων στο δεξί υπόδεντρο (αν υπάρχει). Η δομή του απλού δυαδικού δέντρου αναζήτησης υλοποιήθηκε στην κλάση ***BSTree*** (στα αρχεία *BSTree.h* και *BSTree.cpp)*. Η κλάση ***BSTree*** έχει τις παρακάτω ιδιότητες και μεθόδους:

Ιδιότητες	***Περιγραφή***
Pair pair*	*Ζεύγος λέξεων του συγκεκριμένου node*
BSTree left*	*Δείκτης για το αριστερό υπόδεντρο*
BSTree right*	*Δείκτης για το δεξί υπόδεντρο*

Μέθοδοι	***Περιγραφή***
BSTree(Pair pair)*	*Constructor της κλάσης, δημιουργεί ένα νέο node με το ζεύγος pair*
BSTree insert(BSTree* node, Pair* pair)*	*Προσθέτει το ζεύγος pair στο δέντρο*
BSTree search(BSTree* node, Pair* pair)*	*Αναζητεί το ζεύγος pair στο δέντρο*
BSTree remove(BSTree* node, Pair* pair)*	*Αφαιρεί το ζεύγος pair από το δέντρο*
BSTree inorder_successor(BSTree* node)*	*Επιστρέφει το node του δέντρου με το λεξικογραφικά μικρότερο ζεύγος λέξεων*
Pair get_pair(BSTree* node)*	*Επιστρέφει ένα δείκτη για το ζεύγος που βρίσκεται στο συγκεκριμένο node*
void print(BSTree node)*	*Εμφανίζει όλα τα ζεύγη σε αλφαβητική σειρά*

insert(…): Πριν εισαχθεί ένα νέο ζεύγος στο δέντρο ελέγχεται εάν υπάρχει ήδη, αναζητώντας το δυαδικά με αναδρομή. Σε περίπτωση που υπάρχει ήδη αυξάνεται το πλήθος εμφανίσεων. Διαφορετικά, η εισαγωγή ενός νέου ζεύγους γίνεται στο κατάλληλο *node* του δέντρου - υποδέντρου ώστε να διατηρηθούν οι ιδιότητες του.

search*(…)*: Εκτελεί δυαδική αναζήτηση του ζεύγους μέσα στο δέντρο με αναδρομή. Εάν το ζεύγος βρεθεί, επιστρέφεται ο *pointer* του *node* που το περιέχει.

remove*(…)*: Εντοπίζει και αφαιρεί το *node* που περιέχει το συγκεκριμένο ζεύγος μέσα από το δέντρο (εάν υπάρχει). Ανάλογα με το πλήθος των παιδιών του *node* που αφαιρείται γίνονται οι κατάλληλοι μετασχηματισμοί ώστε να διατηρηθούν οι ιδιότητες του δυαδικού δέντρου αναζήτησης.

Δυαδικό δένδρο αναζήτησης τύπου AVL (AVLTree)

Το *δυαδικό δέντρο αναζήτησης τύπου AVL* είναι ένα *self-balancing binary search tree* και έχει αρκετά κοινά στοιχεία με το *απλό δυαδικό δέντρο αναζήτησης*, διατηρώντας τις ιδιότητες του. Η επιπλέον ιδιότητα ενός *AVL tree* είναι ότι η διαφορά ύψους του αριστερού και του δεξιού υποδέντρου κάθε κόμβου είναι ≤*1 (balanced)*. Για να διατηρείται αυτή η ιδιότητα, μετά από κάθε εισαγωγή - αφαίρεση ενός node πρέπει (εάν είναι απαραίτητο) να εκτελεστούν οι κατάλληλες περιστροφές, ώστε το *AVL tree* να ξαναγίνει *balanced*. Η δομή του *δυαδικού δέντρου αναζήτησης τύπου AVL* υλοποιήθηκε στην κλάση ***AVLTree*** (στα αρχεία *AVLTree.h* και *AVLTree.cpp)*. Η κλάση ***AVLTree*** έχει τις παρακάτω ιδιότητες και μεθόδους:

Ιδιότητες	***Περιγραφή***
Pair pair*	*Ζεύγος λέξεων του συγκεκριμένου node*
AVLTree left*	*Δείκτης για το αριστερό υπόδεντρο*
AVLTree right*	*Δείκτης για το δεξί υπόδεντρο*
long height	*Το ύψος του συγκεκριμένου node*

Μέθοδοι	***Περιγραφή***
AVLTree(Pair pair)*	*Constructor της κλάσης, δημιουργεί ένα νέο node με το ζεύγος pair*
AVLTree insert(AVLTree* node, Pair* pair)*	*Προσθέτει το ζεύγος pair στο δέντρο*
AVLTree search(AVLTree* node, Pair* pair)*	*Αναζητεί το ζεύγος pair στο δέντρο*
AVLTree remove(AVLTree* node, Pair* pair)*	*Αφαιρεί το ζεύγος pair από το δέντρο*
AVLTree inorder_successor(AVLTree* node)*	*Επιστρέφει το node του δέντρου με το λεξικογραφικά μικρότερο ζεύγος λέξεων*

AVLTree rotate_left(AVLTree* node)*	*Εκτελεί αριστερή περιστροφή στο node*
AVLTree rotate_right(AVLTree* node)*	*Εκτελεί δεξιά περιστροφή στο node*
AVLTree rotate_left_right(AVLTree* node)*	*Εκτελεί αριστερή και στη συνέχεια δεξιά περιστροφή στο node*
AVLTree rotate_right_left(AVLTree* node)*	*Εκτελεί δεξιά και στη συνέχεια αριστερή περιστροφή στο node*
Pair get_pair(AVLTree* node)*	*Επιστρέφει ένα δείκτη για το ζεύγος που βρίσκεται στο συγκεκριμένο node*
int get_height(AVLTree node)*	*Επιστρέφει το ύψος του node*
int balance_factor(AVLTree node)*	*Επιστρέφει τον balance factor του node*
void increment_height(AVLTree node)*	*Αυξάνει το ύψος ενός node κατά ένα*
void update_height(AVLTree node)*	*Ανανεώνει το ύψος του node*
void print(AVLTree node)*	*Εμφανίζει όλα τα ζεύγη σε αλφαβητική σειρά*

insert(…): Πριν εισαχθεί ένα νέο ζεύγος στο δέντρο ελέγχεται εάν υπάρχει ήδη, αναζητώντας το δυαδικά με αναδρομή. Σε περίπτωση που υπάρχει ήδη αυξάνεται το πλήθος εμφανίσεων. Διαφορετικά, η εισαγωγή ενός νέου ζεύγους γίνεται στο κατάλληλο *node* του δέντρου - υποδέντρου ώστε να διατηρηθούν οι ιδιότητες του. Εάν μετά την εισαγωγή το *node* γίνει *unbalanced*, εκτελούνται οι κατάλληλες περιστροφές ώστε να ξαναγίνει balanced.

search*(…)*: Εκτελεί δυαδική αναζήτηση του ζεύγους μέσα στο δέντρο με αναδρομή. Εάν το ζεύγος βρεθεί, επιστρέφεται ο *pointer* του *node* που το περιέχει.

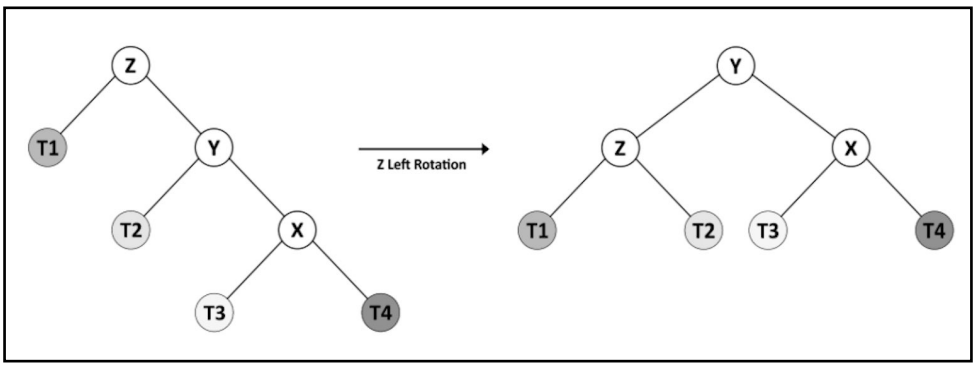
remove(…): Εντοπίζει και αφαιρεί το *node* που περιέχει το συγκεκριμένο ζεύγος μέσα από το δέντρο (εάν υπάρχει). Ανάλογα με το πλήθος των παιδιών του *node* που αφαιρείται γίνονται οι κατάλληλοι μετασχηματισμοί ώστε να διατηρηθούν οι ιδιότητες του δυαδικού δέντρου αναζήτησης. Εάν μετά την αφαίρεση το *node* γίνει *unbalanced*, εκτελούνται οι κατάλληλες περιστροφές ώστε να ξαναγίνει *balanced*.

balance_factor(…): Ο *balance factor* ενός *node* ορίζεται ως η διαφορά ύψους μεταξύ του αριστερού υπόδεντρου και του δεξιά υπόδεντρου αυτού του node. Εάν ο *balance factor* ενός *node* είναι *<-1,* σημαίνει πως το node αυτό είναι *right-heavy*, ενώ αν είναι *>1* σημαίνει πως το node είναι *left-heavy.*

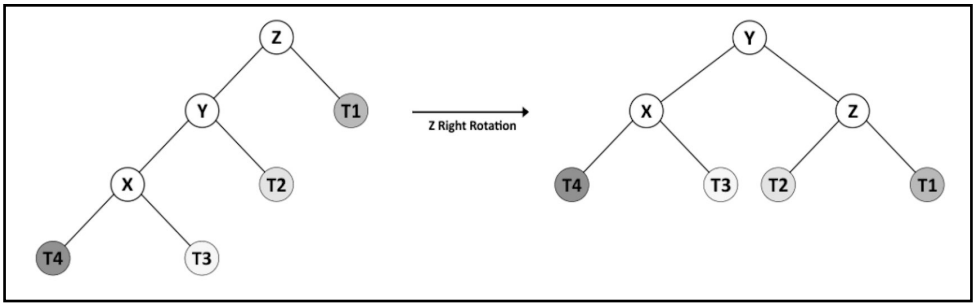
update_height(…): Το ύψος ενός node ορίζεται ως το μεγαλύτερο από τα ύψη του αριστερού και του δεξία υπόδεντρου αυξημένο κατά *1*.

Οι περιστροφές που μπορούν να εκτελεστούν σε ένα κόμβο είναι οι εξής:

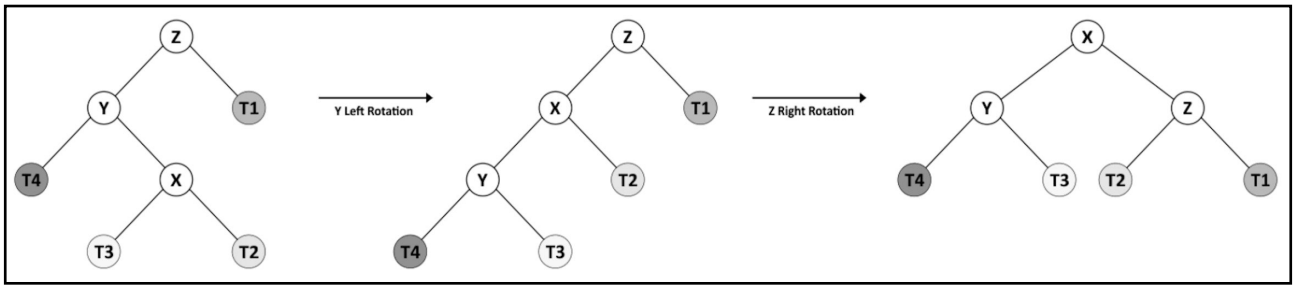
Αριστερή περιστροφή του κόμβου Z (*right-right case*)



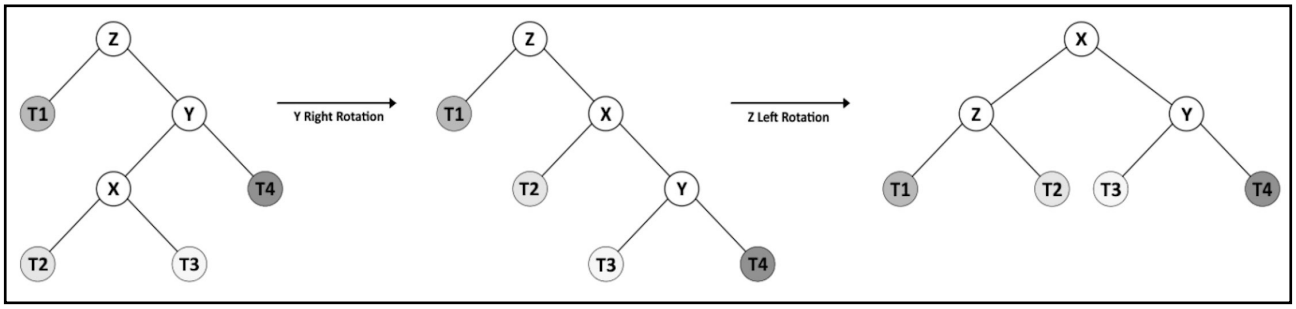
Δεξιά περιστροφή του κόμβου Z (*left-left case*)



Αριστερή περιστροφή του κόμβου Y και δεξιά περιστροφή του κόμβου Z (*left-right case*)



Δεξιά περιστροφή του κόμβου Y και αριστερή περιστροφή του κόμβου Z (*right-left case*)



Πίνακας κατακερματισμού με ανοικτή διεύθυνση (HashTable)

Ο πίνακας κατακερματισμού με ανοικτή διεύθυνση είναι η τελευταία δομή δεδομένων που ζητήθηκε και υλοποιήθηκε με την κλάση ***HashTable*** (στα αρχεία *HashTable.h* και *HashTable.cpp*). Βασική ιδιότητα του ***HashTable*** είναι ότι μπορεί να εκτελέσει σε σταθερό χρόνο -με πολυπλοκότητα O(1)- τις λειτουργίες εισαγωγής, αναζήτησης και διαγραφής ενός ζεύγους λέξεων. Η κλάση ***HashTable*** έχει τις παρακάτω ιδιότητες και μεθόδους:

Ιδιότητες	***Περιγραφή***
*Pair** hashtable*	*Πίνακας κατακερματιμού που περιέχει pointers για αντικείμενα της κλάσης Pair*
long size	*Το μέγεθος του hashtable*
long used_slots	*Δείκτης για το δεξί υπόδεντρο*

Μέθοδοι	***Περιγραφή***
HashTable(long size)	*Constructor της κλάσης, κάνει initialize ένα νέο πίνακα κατακερματισμού μεγέθους size*
~HashTable( )	*Destructor της κλάσης*
bool insert(Pair pair)*	*Προσθέτει το ζεύγος pair στο hash table*
long search(Pair pair)*	*Αναζητεί το ζεύγος pair στο hash table*
bool remove(Pair pair)*	*Αφαιρεί το ζεύγος pair από το hash table*
unsigned long pair_hash(Pair pair)*	*Επιστρέφει το hash του ζεύγους pair*
Pair get_pair(long index)*	*Επιστρέφει ένα δείκτη για το Pair που βρίσκεται στη θέση index του hash table*
void print( )	*Εμφανίζει όλα τα ζέυγη*

pair_hash(…): Υπολογίζει το *hash* του ζεύγους λέξεων που έχει ως όρισμα. Η συνάρτηση κατακερματισμού που χρησιμοποιήθηκε ονομάζεται *djb2* και βρίσκεται υλοποιημένη εδώ.

insert(…): Πριν εισαχθεί ένα νέο ζεύγος στο *hash table* υπολογίζεται το *hash* του με τη μέθοδο *pair_hash(…)*. Εάν το *hash table* στη θέση *hash* είναι ελεύθερο, το ζεύγος

εισάγεται εκεί. Σε περίπτωση που πρόκειται για διπλότυπο, αυξάνεται το πλήθος εμφανίσεων του ζεύγους στο *hash table*, αλλιώς συνεχίζει στην επόμενη θέση (*hash+1*), όπου επαναλαμβάνεται η διαδικασία.

search*(…)*: Πριν αναζητηθεί ένα ζεύγος στο *hash table* υπολογίζεται και πάλι το *hash* του με τη μέθοδο *pair_hash(…)*. Εάν το *hash table* στη θέση *hash* είναι περιέχει το ζητούμενο ζεύγος, επιστρέφεται η θέση του. Διαφορετικά, συνεχίζει στην επόμενη θέση (*hash+1*) όπου και επαναλαμβάνει τη διαδικασία.

remove(…): Πριν αφαιρεθεί ένα ζεύγος από το *hash table* υπολογίζεται το *hash* του και αναζητείται στο *hash table* -η λογική αναζήτησης είναι ίδια με αυτή της *search(…)*-. Σε περίπτωση που βρεθεί, αφαιρείται από το *hash table*.

Αποτελέσματα

Τα ζεύγη κάθε δομής εμφανίζονται στο αρχείο εξόδου με τη μορφή:

word1 word2 [occurrences]

όπου *word1* και *word2* η πρώτη και δεύτερη λέξη του ζεύγους, ενώ ο αριθμός *occurrences* αντιστοιχεί στο πλήθος εμφανίσεων του ζεύγους στο κείμενο εισόδου.

Οι χρόνοι κατασκευής και αναζήτησης κάθε δομής εμφανίζονται με τη μορφή:

δομή_δεδομένων initialization time -> init_time seconds
δομή_δεδομένων search time -> search_time seconds

όπου init_time ο χρόνος κατασκευής και *search_time* ο χρόνος αναζήτησης για την κάθε *δομή_δεδομένων*.

Ακολουθούν οι χρόνοι κατασκευής και αναζήτησης για *Q_SIZE=1000* ζεύγη για το μικρό αρχείο *small-file.txt*:

small-file.txt		
Δομή δεδομένων	***Χρόνος κατασκευής***	***Χρόνος αναζήτησης***
UnorderedArray	*8 min 42 sec*	*0 min 2.208 sec*
OrderedArray	*10 min 20 sec*	*0 min 0.010 sec*
BSTree	*0 min 1.44 sec*	*0 min 0.011 sec*
AVLTree	*0 min 1.38 sec*	*0 min 0.010 sec*
HashTable	*0 min 0.17 sec*	*0 min 0.005 sec*

Τα αναλυτικά αποτελέσματα μαζί με το πλήθος εμφανίσεων του κάθε ζεύγους για το αρχείο ***small-file.txt*** βρίσκονται στο αρχείο ***output-small-file.txt***. Επιπλέον, έγινε μία τελική δοκιμή του προγράμματος με το μεγάλο αρχείο των *~2.21GB* ***gutenberg.txt*** ως είσοδο, μόνο για την δομή του πίνακα κατακερματισμού ***HashTable***. Ακολουθούν τα αποτελέσματα:

gutenberg.txt		
Δομή δεδομένων	***Χρόνος κατασκευής***	***Χρόνος αναζήτησης***
HashTable	*58 min 42 sec*	*0 min 1.237 sec*

Τα αναλυτικά αποτελέσματα για τη δομή του πίνακα κατακερματισμού ***HashTable*** μαζί με το πλήθος εμφανίσεων του κάθε ζεύγους για το αρχείο ***gutenberg.txt*** βρίσκονται στο αρχείο ***output-gutenberg-hashtable.txt***.