Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα
Μέρος Α'

Μεντζέλος Ηλίας ***1115201400106***
Πλακιάς Φώτιος-Απόστολος ***1115201400161***
Πολύδωρος Αθανάσιος ***1115201400164***

```
typedef struct trie{
        trie_node* root;
}trie;
```

Αποτελεί το interface της εφαρμογής μας το οποίο σε αυτήν την φάση περιέχει μόνο τον δείκτη στην ρίζα του tree. Στο μέλλον, ενδέχεται τα περιεχόμενα του να αυξηθούν ώστε να υλοποιηθούν άλλες λειτουργίες.

```
typedef struct trie_node{
        char* word;
        char is_final;
        int current_children;
        int max_children;
        struct trie_node* children;
}trie_node;
```

Είναι ένας κόμβος του trie. Αποτελείται από την λέξη, ένα αναγνωριστικό για να μπορούμε να δούμε αν είναι τερματική λέξη για κάποιο n-gram, τον τρέχων αριθμό παιδιών της λέξεις, τον μέγιστο αριθμό παιδιών που μπορούν να δημιουργηθούν πριν χρειαστεί να αυξήσουμε εκθετικά το μέγεθος του πίνακα (με χρήση realloc) και τέλος των πίνακα με τα παιδιά του κόμβου.

trie_node * init_trie():

Δημιουργεί το root του trie καλόντας την create_trie_node και βάζει σαν word του root την κενή συμβολοσειρά.

trie_node* create_trie_node():

Δεσμεύει έναν trie_node, αρχικοποιεί τις τιμές του, δεσμεύει των πίνακα των παιδιών του και επιστρέφει το trie_node. Χρησιμοποιείται στην create_trie() για την δημιουργία του root.

void init_trie_node(trie_node * node):

Παίρνει ένα tree_node, αρχικοποιεί τις τιμές του, δεσμεύει των πίνακα των παιδιών του και επιστρέφει το trie_node. Χρησιμοποιείται για την δημιουργία ενός νέου παιδιού για ένα tree_node.

int binary_search_kid(trie_node* master_node,char* word,int* spot_ptr_arg):

Η συγκεκριμένη συνάρτηση χρησιμοποιείται σε κάθε λειτουργία του πρόγραματος. Ουσιαστικά αναζητεί στα παιδιά του trie_node μας για μία συγκεκριμένη λέξη και γυρνάει την θέση της στην μεταβλητή spot_opt_arg. Σε περίπτωση που βρει την λέξη γυρνάει την τιμή 1 η συνάρτηση, ενώ σε περίπτωση αποτυχίας γυρνάει την τιμή -1. Η αναζήτηση γίνεται με binary search, αφού ο πίνακας με τα παιδιά είναι ταξινομημένος Ακόμα και σε περίπτωση που δεν βρεθεί η λέξη, η συνάρτηση μας γυρνάει την θέση που πρέπει να εισαχθεί στον πίνακα των παιδιών.

OK_SUCCESS insert_ngram_to_node(trie_node * node, char * ngram):

Παίρνει ένα n-gram και το εισάγει στα κατάλληλα trie_nodes. Σαν όρισμα δέχεται το root trie_node και έπειτα σπάει το n-gram σε λέξεις μέσω της strtok. Ξεκίναει από την 1η λέξη, όπου ψάχνουμε μέσω της binary_search_kid αν υπάρχει ήδη. Αν υπάρχει, τότε προχωράμε στα παιδιά της και ψάχνουμε στα παιδιά της για την επόμενη λέξη. Διαφορετικά, μας επιστρέφει σε ποια θέση πρέπει να εισαχθεί η λέξη μας, την εισάγουμε και πάμε να εισάγουμε στα παιδιά της τις υπόλοιπες λέξεις. Η διαδικασία αυτή συνεχίζεται μέχρι να έχουμε εισάγει όλες τις λέξεις του n-gram. Όταν φτάσουμε στο τέλος τότε κάνουμε την τελευταία μας εισαγωγή final, αλλάζοντας την μεταβλητή του is_final σε 'Y'.

OK_SUCCESS insert_ngram(trie * my_trie, char * ngram):

Στέλνει στην συνάρτηση insert_ngram_to_node το root του trie μας καθώς και το n-gram που θέλουμε να εισάγουμε.

result_of_search* search(trie* my_trie,char* the_ngram):

Για την υλοποίηση της search αρχικά είχαμε χρησιμοποιήσει την strtok, άλλα επειδή επηρεάζει το string εσωτερικά, υλοποιήσαμε σχεδόν την strtok.
Πιο συγκεκριμένα, κατά την αναζήτηση μιας πρότασης ακολουθούνται τα εξής βήματα αφού πρώτα αντιγράφουμε το query

1. Βρίσκουμε το πρώτο κενό και το αντικαθιστούμε με \0
2. Ελέγχουμε αν η λέξη ανήκει στο επίπεδο που βρισκόμαστε με binary search, αν οχι σταματάμε την αναζήτηση και πηγαίνουμε στο βήμα 1 προχωρώντας το string μια λέξη δεξιά
    a. αν ναι, ελέγχουμε αν είναι final και την προσθέτουμε στο result
    b. συνεχίζουμε βαθύτερα(αν μπορούμε) και προχωράμε στην επόμενη λέξη (αν υπάρχει) επαναλαμβάνοντας το βημα 2.
3. επαναφέρουμε την λέξη στην αρχική της κατάσταση, από εκεί που σταματήσαμε μέχρι εκει που ξεκινήσαμε (κάνουμε όλα τα '\0' ίσα με Κεν,εκ τος του τελευταίου)
4. θέτουμε ως αρχή την επόμενη λέξη από αυτή που ξεκινήσαμε πριν (στην αρχή ξεκινάμε από την πρώτη) και συνεχίζουμε από το βήμα ενεά αν η λέξη μας δεν είναι κενή

Η προσθήκη στο result γίνεται μέσω μιας struct που ουσιαστικά δεσμεύει αρχικά ένα string και το αυξάνει δυναμικά όποτε χρειάζεται καθ ως βρίσκουμε καινούρια αποτελέσματα

Δεν κατασκευάζουμε το string που θέλουμε να προσθέσουμε άλλα το προσδιορίζουμε με χρήση 2 pointer στο string που έχουμε αντιγράψει κατά την αναζήτηση. Το struct αυτό δημιουργεί κάθε φορά που καλείται η search και το καταστρέφει η main.

Έναν στην αρχή του string που θέλουμε να προσθέσουμε και έναν στην αρχή της τελευταίας λέξης που θέλουμε να προσθέσουμε. Αυτό σημαίνει ότι υπάρχει περίπτωση να υπάρχουν συνεχόμενα κενά οπότε και αποφεύγεται η χρήση της strstr.
Κατά την προσθήκη στο result γίνεται έλεγχος αν υπάρχει ήδη το ngram με την χρήση της strstr().

OK_SUCCESS delete_ngram(trie * , char *):

Η delete_ngram καλείται από τη main παίρνοντας το trie και το ngram που πρέπει να διαγραφεί. Το μόνο που κάνει είναι να δίνει το root του trie και το ngram στη trie_delete.

OK_SUCCESS trie_delete(trie_node*,char*):

Το σκεπτικό της συνάρτησης είναι να κατεβαίνει το trie κόμβο κόμβο ελέγχοντας αν η λέξεις του ngram υπάρχουν στο trie και αν υπάρχουν όλες (συνεπώς και το ngram) τότε ανεβαίνει το δέντρο (πάλι κόμβο κόμβο) και διαγράφει όσους κόμβους πρέπει . Αρχικά φτιάχνουμε ένα πίνακα(στοίβα) , μεγέθους όσου και του ngram και περιέχει stack_node structs .Το struct είναι το εξής :

```
typedef struct stack_node
{
        trie_node* node;
        int position;
}stack_node;
```

Με αυτό τον τρόπο καθώς κατεβαίνω το trie αποθηκεύω τους pointers των κόμβων όπου βρίσκεται κάθε λέξη του ngram και τη θέση του στον πίνακα του από πάνω του κόμβου. Έτσι έχω μια while που κάνει το κατέβασμα στο trie και άμα δεν υπάρχει το ngram τερματίζει η συνάρτηση. Όταν φτάσει στη τελευαία λέξη του ngram κάνει break η while και ελέγχει αν είναι τελικό . Άμα είναι σημαίνει ότι το ngram υπάρχει. Έπειτα το κάνει μη τερματικό (δηλαδή οτί δεν είναι πια ngram) και ελέγχει αν ο κόμβος έχει παιδιά . Άμα δεν έχει διαγράφει τον κόμβο . Μετά αν διαγράφτηκε ο κόμβος παίρνουμε τα struct από
τη “στοίβα” και ανεβαίνουμε το trie όσο οι κόμβοι με τη σειρά του δεν έχουν πλέον κάποιο παιδί . Μόλις βρεθεί κάποιος κόμβος που παραμένει να έχει παιδία τότε σταματάμε γιατί δεν μπορεί να γίνει άλλη διαγραφή . Η διαγραφή των κόμβων γίνεται με την delete_node_child .

OK_SUCCESS delete_node_child(trie_node* ,int):

Παίρνει ένα trie_node και τη θέση του παιδιού στον πίνακα του trie_node που θέλουμε να διαγράψουμε. Κάνει τα κατάλληλα free και έπειτα αν χρειάζεται κάνει αριστερό shift μια θέση τα παιδιά που βρίσκονται δεξιά από αυτό που μόλις διαγράφηκε. Τέλος μειώνουμε στο κόμβο τον αριθμό των παιδιών του.

OK_SUCCESS trie_node_clean(trie_node*):

Η trie_node_clean κατεβαίνει το trie αναδρομικά (dfs) και κάνει τα κατάλληλα free καθαρίζοντας τη allocated μνήμη.