Κ23α - Ανάπτυξη Λογισμικού Για Πληροφοριακά Συστήματα

Χειμερινό Εξάμηνο 2021– 2022

Καθηγητής Ι. Ιωαννίδης
Άσκηση 1 – Παράδοση: Δευτέρα 15 Νοεμβρίου 2021

Το θέμα της φετινής εργασίας είναι μία παραλλαγή του προγραμματιστικού διαγωνισμού SIGMOD 2013.

Ανεστραμμένη Μηχανή Αναζήτησης (inverted search engine)

Μια συνηθισμένη μηχανή αναζήτησης έχει ένα μεγάλο και ως επί το πλείστον στατικό πλήθος από κείμενα στη διάθεσή της. Ανά τακτά χρονικά διαστήματα γίνονται στη μηχανή διαφορετικά ερωτήματα με στόχο την εύρεση και ανάκτηση των σχετικών κειμένων με αυτά. Κλασικό παράδειγμα μιας τέτοιας εφαρμογής είναι η μηχανή αναζήτησης της Google, στην οποία οι χρήστες γράφουν ερωτήματα σχετικά με την πληροφορία που αναζητούν, και η μηχανή επιστρέφει τα κείμενα που θεωρεί ότι σχετίζονται.

Μια ανεστραμμένη μηχανή αναζήτησης λειτουργεί με τον ίδιο τρόπο, αλλά προϋποθέτει ότι τα ερωτήματα που της δίνονται είναι ως επί το πλείστον στατικά, και τα διαθέσιμα κείμενα είναι αυτά που έρχονται δυναμικά. Σχετικό παράδειγμα μιας τέτοιας εφαρμογής είναι το Twitter, όπου τα κείμενα εδώ αντιστοιχούν σε tweets. Νέα tweets, διαφορετικού περιεχομένου έρχονται συνεχώς στην εφαρμογή και πρέπει να ανακτηθούν όταν γίνει σχετική αναζήτηση.

Στα πλαίσια αυτής της εργασίας θέλουμε να αναπτύξετε μια εφαρμογή η οποία θα δέχεται ένα πλήθος από κείμενα που φτάνουν με συνεχή ροή (stream of documents) και ερωτήματα (queries) που πρέπει να απαντηθούν, εφόσον σχετίζονται με κάποιο κείμενο. Ένα κείμενο αναπαριστάται ως μια ακολουθία από λέξεις που χωρίζονται μεταξύ τους με κενά, ενώ ένα ερώτημα ως ένα σύνολο από λέξεις. Τόσο τα ερωτήματα όσο και τα κείμενα αναπαριστώνται από ένα σύνολο από λέξεις (keywords). Όποτε φτάνει ένα νέο κείμενο, η εφαρμογή σας θα πρέπει γρήγορα να βρει όλα τα ερωτήματα που σχετίζονται με αυτό. Για να θεωρήσουμε ότι ένα κείμενο απαντά σε ένα ερώτημα πρέπει το κείμενο να περιέχει λέξεις που ταιριάζουν με όλες τις λέξεις του ερωτήματος. Η διαδικασία με την οποία εξετάζουμε αν μια λέξη ταιριάζει με μια άλλη λέγεται keyword matching και μπορεί να έχει διάφορες παραλλαγές:

- να είναι οι δύο λέξεις απολύτως ίδιες μεταξύ τους
- να υπάρχει ανοχή να διαφέρουν σε κάποια σημεία.

  Σε αυτήν την εργασία θέλουμε να υποστηρίζουμε τα παρακάτω είδη keyword matching:
- ακριβές ταίριασμα (exact matching)
- προσεγγιστικό ταίριασμα (approximate matching) με περιορισμούς στην απόσταση απόκλισης
    - απόσταση απόκλισης επεξεργασίας (edit distance)
    - απόσταση απόκλισης Hamming (Hamming distance)

Στόχος είναι να μειωθεί όσο το δυνατό περισσότερο ο χρόνος απόκρισης του συστήματος στις απαντήσεις των ερωτημάτων. Περιορισμοί σχετικά με το μέγεθος των λέξεων, των ερωτημάτων, κτλ, βρίσκονται στο αρχείο core.h, ενώ περισσότερες λεπτομέρειες μπορείτε να βρείτε και στη σελίδα του διαγωνισμού http://sigmod.kaust.edu.sa/task-details.html

Περιγραφή Εργασίας

Utilities

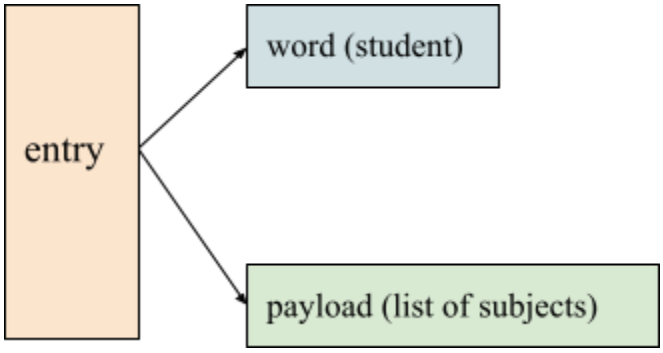
Σε αυτό το κομμάτι θα υλοποιήσετε κάποια απαραίτητα εργαλεία που πρέπει να έχει η εφαρμογή να αξιοποιήσει ως τμήμα της λειτουργικότητά της.

- **Δομές δεδομένων.** Σε αυτό το τμήμα του παραδοτέου θα πρέπει να αποφασίσετε και να υλοποιήσετε τις δομές που θα χρησιμοποιήσετε στον κώδικά σας. Ο τρόπος αναπαράστασης των ερωτημάτων και των κειμένων είναι τα κύρια ζητούμενα εδώ.
- **Keyword Matching.** Αυτό το σύνολο συναρτήσεων θα υλοποιεί τις απαιτούμενες τεχνικές για την εξακρίβωση της ομοιότητας δύο λέξεων. Θα υλοποιήσετε μεθόδους για exact matching, approximate matching με edit distance και approximate matching με hamming distance.
- **Deduplication.** Αυτή η συνάρτηση θα είναι υπεύθυνη για την απαλοιφή των διπλότυπων των λέξεων στα κείμενα. Επειδή τα κείμενα περιέχουν μεγάλο πλήθος λέξεων, είναι εξαιρετικά απίθανο να μην υπάρχουν πολλές ίδιες, γεγονός που πρέπει να το αντιμετωπίσετε, για να μην σπαταλάτε άδικα μνήμη κρατώντας την ίδια πληροφορία.

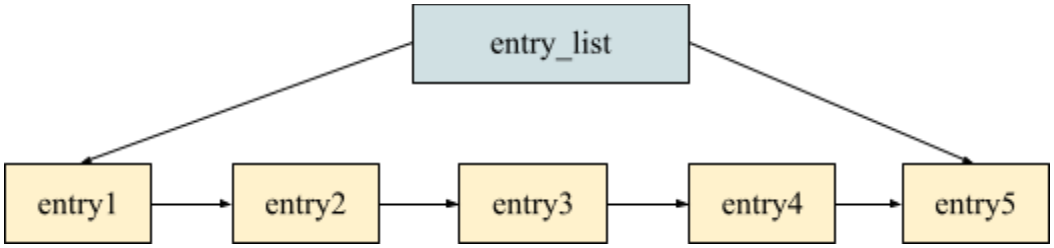
Ανεστραμμένη Μηχανή Αναζήτησης - Επίπεδο Ευρετηρίου (index)

Στο επίπεδο αυτό θα υλοποιήσετε τα απαραίτητα ευρετήρια που θα χρησιμοποιήσει η εφαρμογή σας για να βελτιώσει την ταχύτητα με την οποία ψάχνει ανάμεσα σε ερωτήματα και σε κείμενα και εξακριβώνει αν ταιριάζουν ή όχι. Τα ευρετήρια χρησιμοποιούνται για τη μεθοδολογική αποθήκευση των δεδομένων, με στόχο την ταχεία ανάκτηση πληροφοριών. Χωρίς τη χρήση ευρετηρίων, η μηχανή αναζήτησης θα έπρεπε να ελέγξει κάθε λέξη από το σύνολο των ερωτημάτων του συστήματος, έτσι ώστε να εντοπίσει τα ερωτήματα τα οποία απαντούν ένα κείμενο.

Αρχικά θα ορίσουμε τις βασικές οντότητες του επιπέδου αυτού. Η πρώτη οντότητα είναι το **entry**. Το entry αποτελεί τη δομική οντότητα των ευρετηρίων και περιλαμβάνει δύο πεδία, το **word** και το **payload**. Το word αναπαριστά την λέξη, πάνω στην οποία χτίστηκε το entry και το payload είναι ένα generic πεδίο, το οποίο ορίζει την πληροφορία, η οποία σχετίζεται με το word αυτό. Ένα παράδειγμα αναπαράστασης της πληροφορίας που θα μπορούσε να αντιπροσωπεύει ένα entity είναι το σύνολο των μαθημάτων στα οποία έχει εγγραφεί ένα φοιτητής, όπου το word θα αντιστοιχεί στο αναγνωριστικό του φοιτητή και το payload στη λίστα των μαθημάτων που έχει εγγραφεί.



Η δεύτερη βασική οντότητα του επιπέδου αυτού είναι το **entry_list**. Το entry_list είναι ένα σύνολο από entries, τα οποία χαρακτηρίζονται από μια κοινή ιδιότητα. Στο προηγούμενο παράδειγμα, όπου το entry αναπαραστούσε την αντιστοιχία φοιτητών και μαθημάτων, το entry_list θα μπορούσε να αποτελείται από το σύνολο των φοιτητών, οι οποίοι εισήχθησαν στο ίδιο έτος. Επομένως για κάθε χρόνια, στην οποία υπάρχει τουλάχιστον ένας ενεργός φοιτητής, θα αντιστοιχεί ένα μοναδικό entry_list. Το entry_list θα αποτελεί την είσοδο ενός ευρετηρίου, το οποίο θα χτιστεί πάνω στα entries που αντιστοιχούν στο entry_list αυτό.



Περιγραφή interface

Στην ενότητα αυτή γίνεται μια γενική περιγραφή των διεπαφών που καλείστε να υλοποιήσετε.

```
enum error_code create_entry(const word* w, entry* e)
```

Η ρουτίνα αυτή δημιουργεί και επιστρέφει ένα νέο entry, το οποίο αναπαριστά τη λέξη w. Τέλος θα επιστρέφει κατάλληλο μήνυμα τερματισμού.

```
enum error_code destroy_entry(entry *e)
```

Η ρουτίνα αυτή θα καταστρέφει το δοσμένο entry και θα επιστρέφει κατάλληλο μήνυμα τερματισμού.

```
enum error_code create_entry_list(entry_list* el)
```

Η ρουτίνα αυτή δημιουργεί και θα αρχικοποιεί ένα νέο entry_list και θα επιστρέφει κατάλληλο μήνυμα λάθους.

```
unsigned int get_number_entries(const entry_list* el)
```

Η ρουτίνα αυτή επιστρέφει τον αριθμό των entries που περιέχει η δοσμένη entry_list.

`enum error_code add_entry(entry_list* el,, const entry* e)`
Η ρουτίνα αυτή προσθέτει ένα νέο entry σε μία υπάρχουσα entry_list και θα επιστρέφει κατάλληλο μήνυμα τερματισμού.

`entry* get_first(const entry_list* el)`
Η ρουτίνα αυτή επιστρέφει το πρώτο entry που αντιστοιχεί στη δοσμένη entry_list.

`entry* get_next(const entry_list* el)`
Η ρουτίνα αυτή επιστρέφει το επόμενο entry, το οποίο ανήκει στη δοσμένη entry_list.

`enum eror_code destroy_entry_list(entry_list* el)`
Η ρουτίνα αυτή καταστρέφει τη δοσμένη entry_list και επιστρέφει κατάλληλο μήνυμα.

`enum error_code build_entry_index(const entry_ist* el, enum match_type type, index* ix)`
Η ρουτίνα αυτή δημιουργεί ένα νέο ευρετήριο. Ως είσοδο δίνεται το entry_list, στα στοιχεία του οποίου θα χτιστεί το ευρετήριο, ο τύπος του ταιριάσμος που θα εφαρμοστεί και ένας pointer που μετά το τέλος της ρουτίνας θα πρέπει να δείχνει στο ευρετήριο που δημιουργήθηκε. Τέλος θα επιστρέφει κατάλληλο μήνυμα τερματισμού.

`enum error_code lookup_entry_index(const word* w, index* ix, int threshold, entry_list* result)`
Η ρουτίνα αυτή επιστρέφει ένα σύνολο από entries, τα οποία ταιριάζουν με τη δοσμένη λέξη, δεδομένου του τύπου ταιριάσματος του ευρετηρίου και του threshold που έχει δοθεί ως παράμετρος. Τέλος θα επιστρέφει κατάλληλο μήνυμα τερματισμού.

`enum error_code destroy_entry_index(index* ix)`
Η ρουτίνα αυτή καταστρέφει το ευρετήριο και επιστρέφει κατάλληλο μήνυμα τερματισμού.

BK-tree

Ένα BK-tree είναι μία δενδρική δομή δεδομένων, που ειδικεύεται ως ευρετήριο δεδομένων σε ένα μετρικό χώρο. Ένας μετρικός χώρος είναι ουσιαστικά ένα σύνολο αντικειμένων, μαζί με μία μετρική συνάρτηση απόστασης $d(x, y)$ που ορίζεται για κάθε ζεύγος αντικειμένων στο σύνολο. Η μετρική συνάρτηση πρέπει να ικανοποιεί ένα σύνολο αξιωμάτων προκειμένου να είναι καλώς ορισμένη, όπως θα αναλυθεί παρακάτω.

Η δομή δεδομένων BK-tree προτάθηκε από τους Burkhard και Keller [1] για το πρόβλημα της αναζήτησης σε ένα σύνολο από κλειδιά, προκειμένου να βρεθεί το κλειδί που είναι εγγύτερα σε ένα συγκεκριμένο κλειδί ερώτησης. Ο απλοϊκός τρόπος επίλυσης του προβλήματος αυτού είναι απλά η

σύγκριση του κλειδιού της ερώτησης με κάθε στοιχείο του συνόλου. Ένα BK-tree επιτρέπει να εξαιρεθεί ένα μεγάλο τμήμα του συνόλου των κλειδιών από τις συγκρίσεις.

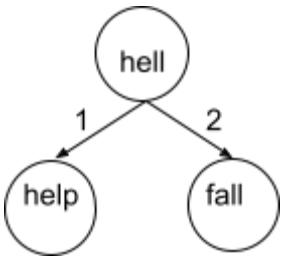
Κατασκευή του δένδρου

Το BK-tree κατασκευάζεται ως εξής. Ένα αυθαίρετο στοιχείο a επιλέγεται ως ρίζα. Η ρίζα μπορεί να έχει μηδέν ή περισσότερα υποδένδρα. Το k-οστό υποδένδρο περιέχει όλα τα στοιχεία b, τέτοια ώστε $d(a, b) = k$.

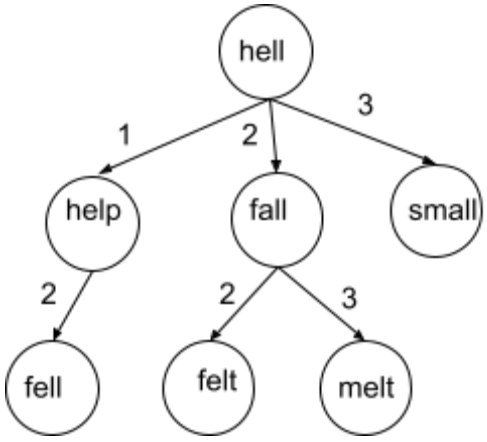
Ας δούμε ένα παράδειγμα κατασκευής ενός BK-tree. Έστω ένα σύνολο από λέξεις από τις οποίες θέλουμε να βρούμε αυτές που είναι πιο όμοιες με μία συγκεκριμένη λέξη ερώτησης. Προκειμένου να εξετάσουμε πόσο όμοιες είναι δύο λέξεις, μπορούμε να χρησιμοποιήσουμε την απόσταση Levenshtein (ή αλλιώς απόσταση επεξεργασίας). Η απόσταση Levenshtein μεταξύ δύο λέξεων προκύπτει από τον ελάχιστο αριθμό επεξεργασίας ενός μόνο χαρακτήρα (η επεξεργασία μπορεί να είναι προσθήκη, διαγραφή ή αντικατάσταση), που απαιτείται προκειμένου να μεταμορφωθεί η μία λέξη στην άλλη. Για παράδειγμα η απόσταση μεταξύ των λέξεων 'hell' και 'small' είναι 3, επειδή για να πάμε από το 'hell' στο 'small', μπορούμε να προσθέσουμε το **s** στην αρχή και στη συνέχεια να αντικαταστήσουμε το **h** με το **m** και το **e** με το **a**.

Έστω το σύνολο λέξεων `['hell', 'help', 'fall', 'felt', 'fell', 'small', 'melt']`

Για να κατασκευαστεί το δένδρο επιλέγουμε ως ρίζα αυθαίρετα μία λέξη, έστω τη 'hell' και στη συνέχεια προσθέτουμε τις υπόλοιπες υπολογίζοντας την απόστασή τους από τη ρίζα. Τότε για τις πρώτες 3 λέξεις, το δένδρο θα έχει την παρακάτω μορφή:



Η λέξη 'help' απέχει 1 από τη 'hell' (αντικατάσταση **p** με **l**), ενώ η λέξη 'fall' απέχει 2 (αντικατάσταση **h** και **e** με **f** και **a**. Στη συνέχεια προσθέτουμε τη λέξη 'felt'. Η λέξη αυτή απέχει 2 από τη 'hell', επομένως θα προστεθεί στο δεξί υποδένδρο ως παιδί του 'fall', χρησιμοποιώντας μία ακμή που να περιέχει την απόσταση μεταξύ 'fall' και 'felt', δηλαδή 2. Μετά την προσθήκη όλων των λέξεων, το δένδρο μας θα έχει την εξής μορφή:



Αναζήτηση

Το αρχικό πρόβλημα ήταν η εύρεση όλων των λέξεων που είναι "κοντά" σε μία δεδομένη λέξη ερώτησης. Έστω n η μέγιστη επιτρεπόμενη απόσταση (που ονομάζεται επίσης και ακτίνα). Ο αλγόριθμος έχει τα εξής βήματα:

1. Δημιούργησε μία λίστα υποψήφιων λέξεων και πρόσθεσε τη ρίζα σε αυτή τη λίστα.
2. Βγάλε μία υποψήφια λέξη από τη λίστα, υπολόγισε την απόστασή της d από τη λέξη της ερώτησης και σύγκρινε την απόσταση με την ακτίνα. Αν είναι μικρότερη, πρόσθεσε την στη λίστα ευρεθεισών λέξεων.
3. Κριτήριο επιλογής: πρόσθεσε στη λίστα υποψήφιων λέξεων όλα τα παιδιά του συγκεκριμένου κόμβου που έχουν απόσταση από τον γονέα στο διάστημα $[d - n,\ d + n]$.

Έστω ότι θέλουμε να βρούμε όλες τις λέξεις που δεν απέχουν περισσότερο από 2 από τη λέξη-κλειδί 'henn'. Η μόνη υποψήφια λέξη στη λίστα είναι η λέξη της ρίζας, δηλαδή 'hell'. Ξεκινάμε υπολογίζοντας την απόσταση μεταξύ 'hell' και 'henn', δηλαδή $d = 2$. Αφού η απόσταση είναι μικρότερη της ακτίνας, προσθέτουμε τη λέξη 'hell' στη λίστα των ευρεθεισών λέξεων.

Στη συνέχεια επεκτείνουμε τη λίστα των υποψήφιων λέξεων με εκείνες τις λέξεις εκείνων των κόμβων/παιδιών που η απόσταση τους από τη ρίζα είναι μεταξύ $[d - n, d + n] = [0, 4]$. Όλα τα υποδένδρα ικανοποιούν αυτό τον περιορισμό, επομένως όλες οι λέξεις του πρώτου υποεπιπέδου προστίθενται στη λίστα υποψήφιων λέξεων ['help', 'fall', 'small']. Οι αποστάσεις των λέξεων αυτών είναι 2, 4 και 5 αντίστοιχα από τη λέξη κλειδί. Επομένως μόνο η λέξη 'help' προστίθεται στη λίστα ευρεθεισών λέξεων. Οι άλλες 2 λέξεις δεν προστίθενται στα αποτελέσματα. Τα υποδένδρα τους εξετάζονται κανονικά..

Στη συνέχεια προστίθεται στη λίστα υποψήφιων λέξεων το μοναδικό υποδένδρο-παιδί της 'help', δηλαδή η λέξη 'fell', και εξετάζονται τα 2 παιδιά της 'fall', δηλαδή οι λέξεις 'felt' και 'melt'. Αυτά τα 2 παιδιά έχουν απόσταση 3 από τη λέξη κλειδί και θα πρέπει να βρίσκονται στο διάστημα [4-2, 4+2]=[2,6], επομένως προστίθενται στη λίστα με τις υποψήφιες λέξεις. Η λέξη 'small' δεν έχει παιδιά.

Τέλος εξετάζονται με τη σειρά τους όλες οι υποψήφιες λέξεις στη λίστα ['fell', 'felt', 'melt'], καμμία εκ των οποίων δεν ικανοποιεί την ελάχιστη απόσταση και καμμία εκ των οποίων δεν έχει παιδιά.

Σε αυτό το σημείο ολοκληρώθηκε η αναζήτηση και η λίστα των ευρεθεισών λέξεων αποτελείται από τις λέξεις ['hell', 'help'], που απέχουν και οι δύο λέξεις απόσταση 2 από τη λέξη κλειδί 'henn'.

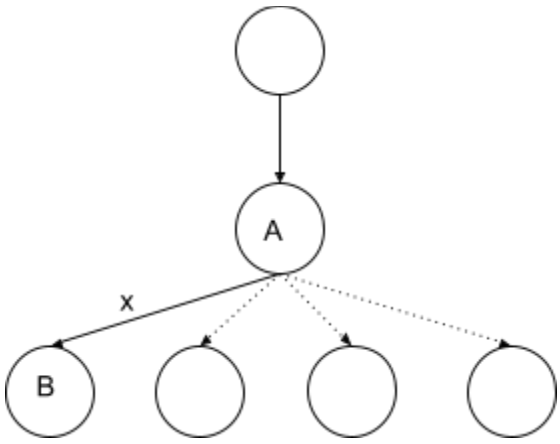
Ανάλυση

Έχει ενδιαφέρον να δούμε γιατί μπορούμε να αποκλείσουμε όλα τα υποδένδρα που δεν καλύπτουν το κριτήριο επιλογής 3. Είχαμε δει ότι η μετρική συνάρτηση απόστασης d πρέπει να ικανοποιεί ένα σύνολο αξιωμάτων προκειμένου να μπορέσουμε να εκμεταλλευτούμε τη δομή του μετρικού χώρου. Τα αξιώματα αυτά είναι τα ακόλουθα. Για όλα τα στοιχεία a, b, c του μετρικού χώρου πρέπει να ισχύουν τα εξής:

1. μη-αρνητικότητα: $d(a, b) \geq 0$.
2. $d(a, b) = 0$ σημαίνει ότι $a = b$ (και το αντίστροφο)
3. συμμετρία: $d(a, b) = d(b, a)$
4. τριγωνική ανισότητα: $d(a, b) \leq d(a, c) + d(c, b)$

Όπως προκύπτει, η απόσταση Levenshtein ικανοποιει και τα 4 αξιώματα, επομένως μπορεί να χρησιμοποιηθεί ως μετρική συνάρτηση.

Έστω $\bar{x}$ το κλειδί της ερώτησης, και ότι αξιολογούμε το παιδί B ενός οποιουδήποτε κόμβου A, για τον οποίο έχουμε καθορίσει ότι απέχει απόσταση $D = d(\bar{x}, A)$ από το κλειδί της ερώτησης. Η γενική δομή του δένδρου μπορεί να αποτυπωθεί στο παρακάτω σχήμα:



Αφού η απόσταση Levenshtein που χρησιμοποιούμε είναι μετρική συνάρτηση, ισχύει ότι

$$d(A, B) \leq d(\bar{x}, A) + d(\bar{x}, B)$$

Επομένως: $d(\bar{x}, B) \geq d(A, B) - d(A, \bar{x}) = x - D$

Αφού ενδιαφερόμαστε μόνο σε κόμβους που βρίσκονται σε απόσταση το πολύ n από το κλειδί της ερώτησης $\overline{x}$, θα πρέπει $d(\overline{x}, B) \leq n$. Αυτό σημαίνει ότι:

$x - D \leq n$ και $D - x \leq n$

που ισοδυναμεί με

$D - n \leq x \leq D + n$

Επομένως, αν η απόσταση d είναι η μετρική συνάρτηση, μπορούμε με ασφάλεια να απορρίψουμε τους κόμβους που δεν ικανοποιούν τα παραπάνω κριτήρια. Τέλος, *κάθε* παιδί του B, θα είναι σε μια απόσταση x από τον A, και συνεπώς μπορεί να αγνοηθεί το συνολικό υποδένδρο αν ο B από μόνος του δεν ικανοποιεί το κριτήριο.

Παράδοση εργασίας

Η εργασία είναι ομαδική, **2 ή 3 ατόμων**.
Προθεσμία παράδοσης: 15/11/2021
Γλώσσα υλοποίησης: C / C++ χωρίς χρήση stl.
Περιβάλλον υλοποίησης: Linux (gcc > 5.4+).

Παραδοτέα: Η παράδοση της εργασίας θα γίνει με βάση το τελευταιο commit πριν την προθεσμία υποβολής στο git repository σας. **Η χρήση git είναι υποχρεωτική**.
Επιπλέον, εκτός από τον πηγαίο κώδικα, θα παραδώσετε μια σύντομη αναφορά, με τις σχεδιαστικές σας επιλογές καθώς και να εφαρμόσετε ελέγχους ως προς την ορθότητα του λογισμικού με τη χρήση ανάλογων βιβλιοθηκών (Software testing).

Παραπομπές
[1] W. Burkhard and R. Keller. Some approaches to best-match file searching, CACM, 1973
[2] https://signal-to-noise.xyz/post/bk-tree/