ΗΥ240: Δομές Δεδομένων
Εαρινό Εξάμηνο – Ακαδημαϊκό Έτος 2017
Διδάσκουσα: Παναγιώτα Φατούρου
Προγραμματιστική Εργασία - 2ο Μέρος

Ημερομηνία Παράδοσης: Δευτέρα, 15 Μαΐου 2017, ώρα 23:59.

Τρόπος Παράδοσης: Χρησιμοποιώντας το πρόγραμμα turnin. Πληροφορίες για το πώς λειτουργεί το πρόγραμμα turnin παρέχονται στην ιστοσελίδα του μαθήματος.

Γενική Περιγραφή

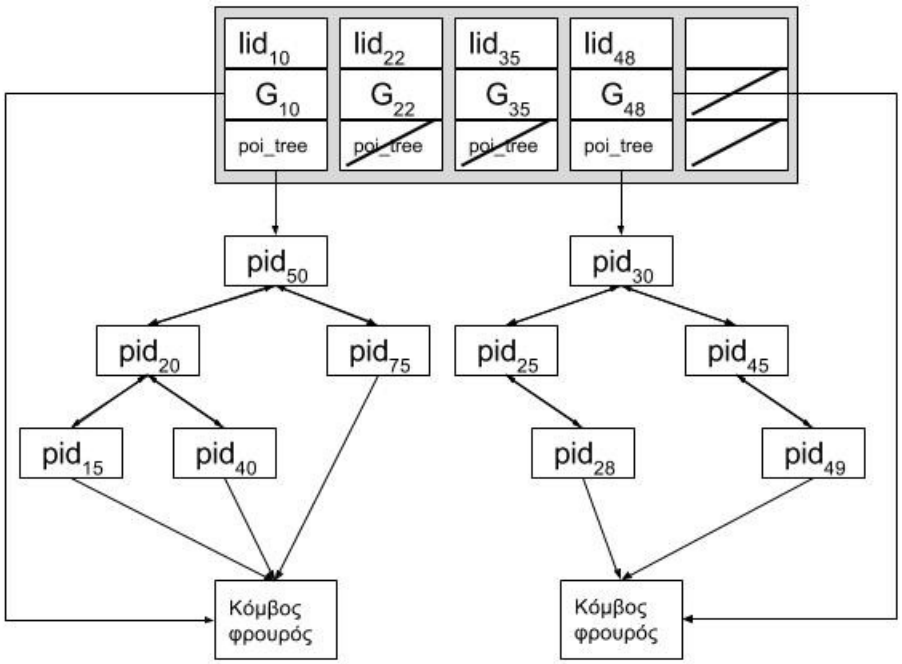
Στην 2η φάση της εργασίας αυτής, καλείστε να υλοποιήσετε και πάλι ένα πρόγραμμα που προσομοιώνει τη λειτουργία ενός συστήματος καταγραφής «σημείων ενδιαφέροντος» (π.χ. ξενοδοχεία, μουσεία, εστιατόρια, αξιοθέατα), από τουριστικής απόψεως, για ένα σύνολο τοποθεσιών. Οι χρήστες του συστήματος μπορούν να αντλήσουν πληροφορίες από αυτό για τις τοποθεσίες που σκοπεύουν να επισκεφτούν. Η διαφορά με την 1η φάση, είναι οι δομές που θα χρησιμοποιήσετε για την υλοποίηση αυτού του συστήματος.

Αναλυτική Περιγραφή Ζητούμενης Υλοποίησης

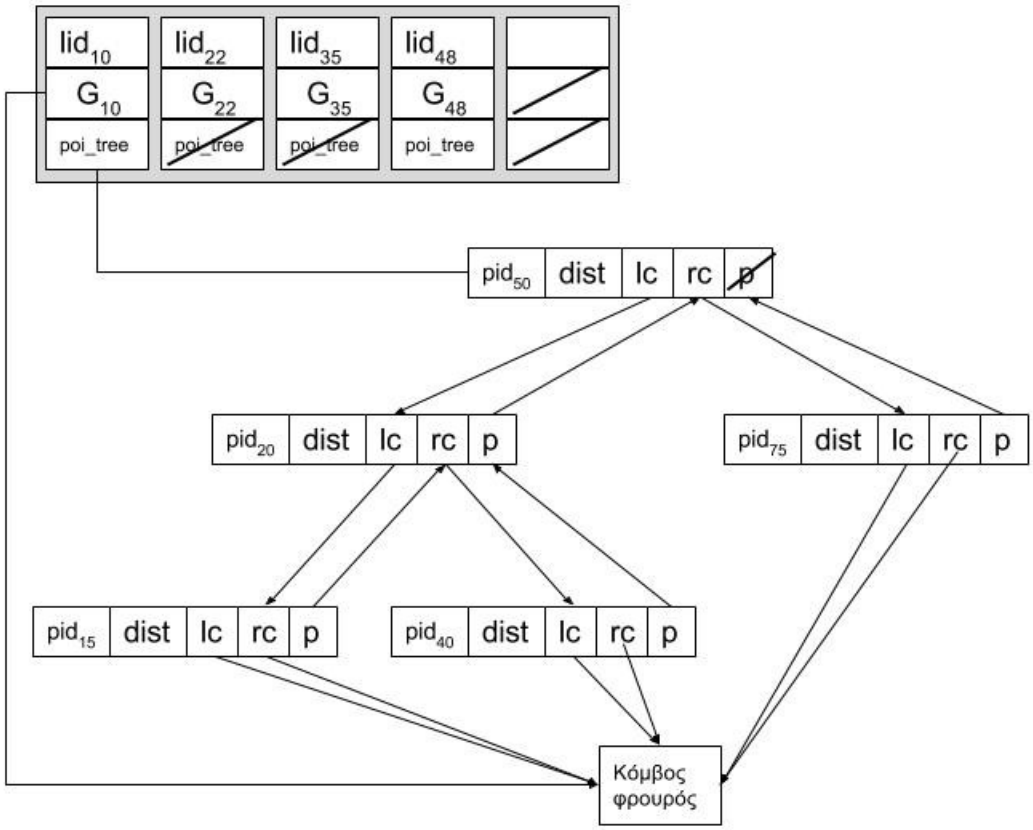
Το σύστημα αποτελείται από ένα σύνολο «σημείων ενδιαφέροντος» (points of interest POI) τα οποία είναι οργανωμένα ανά τοποθεσία. Για κάθε τοποθεσία, θεωρείστε δεδομένο ένα σημείο αναφοράς (π.χ. αυτό μπορεί να είναι κάποιο κεντρικό σημείο της τοποθεσίας). Οι διαθέσιμες τοποθεσίες αποθηκεύονται σε ένα πίνακα του οποίου το κάθε στοιχείο αποτελεί μία εγγραφή τύπου location (struct location). Τα περιεχόμενα αυτού του πίνακα είναι ταξινομημένα, σε αύξουσα διάταξη, με βάση το αναγνωριστικό της κάθε τοποθεσίας (location). Το μέγεθος του πίνακα είναι αρκετά μεγάλο (βλέπε MAX_LOCATIONS), έτσι ώστε να χωράει όλες τις πιθανές τοποθεσίες του συστήματος. Ο πίνακας είναι αρχικά κενός (δηλαδή δεν περιέχει πληροφορίες για καμία τοποθεσία). Η δομή αυτή ονομάζεται **πίνακας τοποθεσιών**. Η κάθε εγγραφή (struct) τύπου location αποτελείται από τα εξής πεδία:

- **lid**: Αναγνωριστικό (τύπου int) που χαρακτηρίζει μοναδικά την τοποθεσία.
- **poi_tree**: Ένας δείκτης (τύπου poi) στη ρίζα ενός δυαδικού δέντρου με κόμβο φρουρό, που ονομάζεται δέντρο σημείων ενδιαφέροντος της τοποθεσίας. Το δέντρο αυτό είναι ταξινομημένο βάσει του αναγνωριστικού pid του σημείου ενδιαφέροντος της κάθε τοποθεσίας. Κάθε κόμβος αυτού του δέντρου, αποτελεί μία εγγραφή τύπου poi (struct poi) με τα παρακάτω πεδία:
    - **pid**: Αναγνωριστικό (τύπου int) που χαρακτηρίζει μοναδικά το σημείο ενδιαφέροντος
    - **distance**: Ένας αριθμός (τύπου int) που αντιστοιχεί στην απόσταση του σημείου ενδιαφέροντος από το σημείο αναφοράς της τοποθεσίας. **ΠΡΟΣΟΧΗ**: Σε αντίθεση με την 1η φάση της εργασίας, η τιμή του πεδίου distance, δεν υπολογίζεται με βάση το προηγούμενο σημείο ενδιαφέροντος της τοποθεσίας. Είναι απλά η απόλυτη απόσταση από το σημείο αναφοράς.
    - **type**: Αναγνωριστικό (τύπου int) που αντιστοιχεί στην κατηγορία του σημείου ενδιαφέροντος. Η μεταβλητή αυτή λαμβάνει τιμές από 1 μέχρι 4, με κάθε τιμή να αντιστοιχεί σε μία κατηγορία (1: ξενοδοχείο, 2: μουσείο, 3: εστιατόριο, 4: αξιοθέατο).
    - **lc**: Ένας δείκτης (τύπου poi) που δεικτοδοτεί το αριστερό παιδί του κόμβου.
    - **rc**: Ένας δείκτης (τύπου poi) που δεικτοδοτεί το δεξί παιδί του κόμβου.
    - **p**: Ένας δείκτης (τύπου poi) που δεικτοδοτεί τον πατέρα του κόμβου.
- **poi_sentinel**: Ένας δείκτης (τύπου poi) που δεικτοδοτεί τον κόμβο φρουρό της τοποθεσίας.

Παρακάτω φαίνονται οι δομές που διατηρούν όλες τις πληροφορίες σχετικά με τις τοποθεσίες και τα σημεία ενδιαφέροντος που περιέχει η καθεμία:



Σχήμα 1: Στο σχήμα αυτό φαίνεται ο ταξινομημένος, ως προς το αναγνωριστικό, πίνακας τοποθεσιών και το δέντρο σημείων ενδιαφέροντος της κάθε τοποθεσίας.

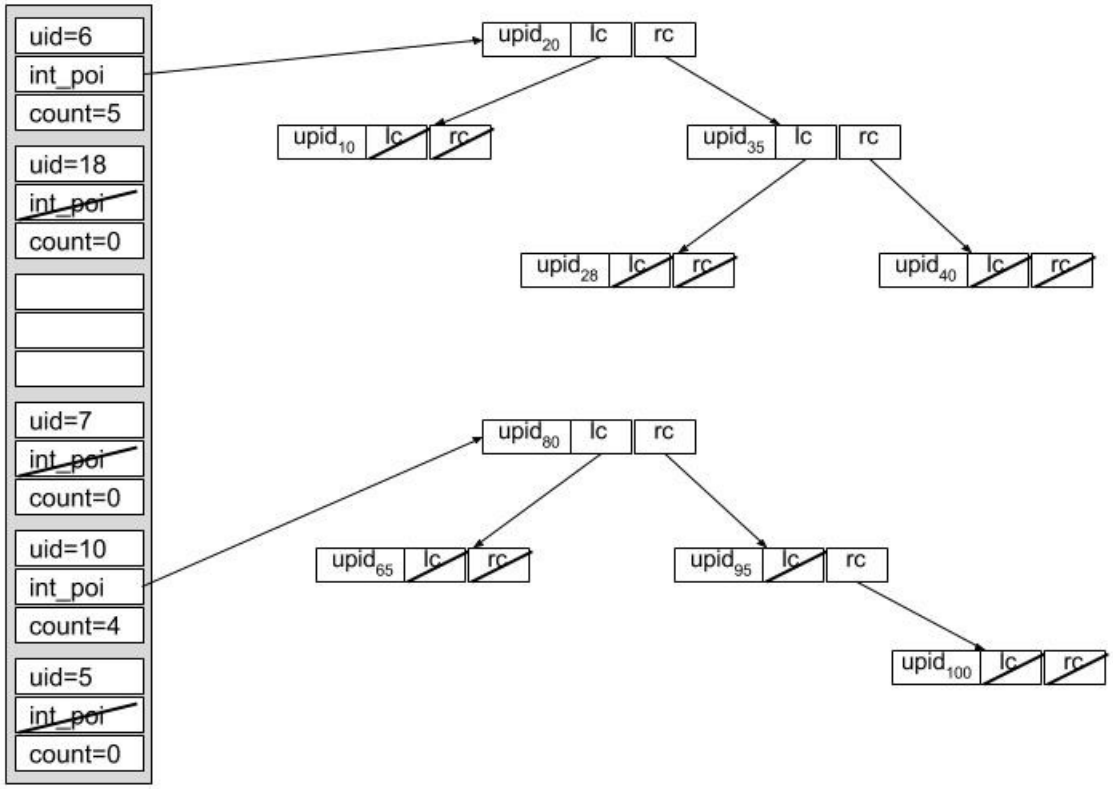


Σχήμα 2: Το δέντρο σημείων ενδιαφέροντος της τοποθεσίας με lid=10 με μεγαλύτερη λεπτομέρεια. Παρατηρούμε πως οι κόμβοι του δέντρου είναι ταξινομημένοι ως προς το πεδίο pid και όλοι οι δείκτες lc, rc, κόμβων που θα ήταν NULL δείχνουν στον κόμβο φρουρό.

Αντίστοιχα με τη 1η φάση, το σύστημα το χρησιμοποιούν ένα σύνολο από χρήστες. Οι χρήστες αυτοί θα αποθηκεύονται σε ένα πίνακα κατακερματισμού. Ο πίνακας αυτός ονομάζεται **πίνακας κατακερματισμού χρηστών** και για την επίλυση των συγκρούσεων χρησιμοποιεί την τεχνική του διπλού κατακερματισμού με ανοικτή διευθυνσιοδότηση (double hashing). Το μέγεθος του πίνακα κατακερματισμού θα πρέπει να επιλέγεται από εσάς προσεχτικά και θα πρέπει να είστε σε θέση να δικαιολογήσετε την επιλογή σας. Για την υλοποίηση των δύο συναρτήσεων κατακερματισμού, θα πρέπει να χρησιμοποιήσετε καθολικό κατακερματισμό (universal hashing). Για την υλοποίηση του καθολικού κατακερματισμού, παρέχεται ένας πίνακας primes_g[] με πρώτους αριθμούς σε αύξουσα σειρά, το μέγιστο πλήθος χρηστών, μέσω της μεταβλητής max_users_g και το μέγιστο αναγνωριστικό μεταξύ των χρηστών, μέσω της μεταβλητής max_uid_g. Ο πίνακας και οι δύο αυτές μεταβλητές είναι global και έχουν δηλωθεί στο αρχείο geo.h. Ο κάθε χρήστης αποτελεί μία εγγραφή τύπου user (struct user) με τα ακόλουθα πεδία:

- **uid**: Αναγνωριστικό (τύπου int) που χαρακτηρίζει μοναδικά το χρήστη.
- **interesting_poi**: Δείκτης (τύπου struct user *) στη ρίζα ενός ταξινομημένου δυαδικού δέντρου, το οποίο ονομάζεται δέντρο σημείων ενδιαφέροντος χρήστη. Το δέντρο αυτό θα περιέχει τα σημεία ενδιαφέροντος που σκοπεύει να επισκεφτεί ο χρήστης και θα είναι ταξινομημένο ως προς το αναγνωριστικό του σημείου ενδιαφέροντος του χρήστη (upid). Κάθε κόμβος αυτού του δέντρου αποτελεί μία εγγραφή τύπου user_poi με τα εξής πεδία:
  - **upid**: Αναγνωριστικό (τύπου int) που χαρακτηρίζει μοναδικά το σημείο ενδιαφέροντος του χρήστη.
  - **lc**: Ένας δείκτης (τύπου user_poi) που δεικτοδοτεί το αριστερό παιδί του κόμβου.
  - **rc**: Ένας δείκτης (τύπου user_poi) που δεικτοδοτεί το δεξί παιδί του κόμβου.
- **upoi_count**: Το πλήθος των κόμβων του δέντρου σημείων ενδιαφέροντος χρήστη.

Ένα παράδειγμα φαίνεται στο Σχήμα 3:



Σχήμα 3: Ο πίνακας κατακερματισμού χρηστών καθώς και τα δέντρα σημείων ενδιαφέροντός τους. Σημειώνεται ότι τα δέντρα ενδιαφέροντος χρήστη είναι ταξινομημένα ως προς το πεδίο upid και δεν υπάρχει κόμβος φρουρός.

Τρόπος Λειτουργίας Προγράμματος

Το πρόγραμμα που θα δημιουργηθεί θα πρέπει να εκτελείται καλώντας την ακόλουθη εντολή:

<executable> <input-file>

όπου **<executable>** είναι το όνομα του εκτελέσιμου αρχείου του προγράμματος (π.χ. a.out) και **<input-file>** είναι το όνομα του αρχείου εισόδου (π.χ. testfile) το οποίο περιέχει γεγονότα των ακόλουθων μορφών:

- L <lid>
Γεγονός που υποδηλώνει την εισαγωγή μιας νέας τοποθεσίας με αναγνωριστικό **<lid>** στο σύστημα. Κατά το γεγονός αυτό θα γίνεται εισαγωγή μιας νέας εγγραφής τύπου location στον πίνακα τοποθεσιών. Μετά από κάθε εισαγωγή, ο πίνακας τοποθεσιών πρέπει να παραμένει ταξινομημένος. Συγκεκριμένα, αν έχουν εισαχθεί k εγγραφές για τοποθεσίες στον πίνακα, οι εγγραφές αυτές καταλαμβάνουν τις πρώτες k θέσεις του πίνακα, σε αύξουσα διάταξη ως προς το πεδίο lid. Οι υπόλοιπες θέσεις του πίνακα είναι κενές. Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

L <lid>
Locations = <lid_1>, <lid_2>, ... ,<lid_n>
DONE

όπου n είναι ο αριθμός των εγγραφών στον πίνακα τοποθεσιών και για κάθε i∈{1,.., n}, <lid_i> είναι το αναγνωριστικό της τοποθεσίας που αντιστοιχεί στην i-οστή εγγραφή του πίνακα αυτού.

- P <pid> <type> <distance> <lid>
Γεγονός που υποδηλώνει την εισαγωγή ενός νέου σημείου ενδιαφέροντος με αναγνωριστικό **<pid>** στην τοποθεσία με αναγνωριστικό **<lid>**. Η παράμετρος αντιστοιχεί στον τύπο του σημείου ενδιαφέροντος όπως περιγράφεται παραπάνω στο πρώτο σκέλος της εκφώνησης. Η παράμετρος περιγράφει την απόσταση του νέου σημείου ενδιαφέροντος από το σημείο αναφοράς της τοποθεσίας . Το γεγονός αυτό προσθέτει έναν κόμβο που αντιστοιχεί στο νέο σημείο ενδιαφέροντος στο δέντρο σημείων ενδιαφέροντος της τοποθεσίας . Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

P <pid> <type> <distance> <lid>
Location = <pid_1:$type_1$:$distance_1$>, <pid_2:$type_2$:$distance_2$> , ... ,<pid_n:$type_n$:$distance_n$>
DONE

όπου n είναι ο αριθμός των κόμβων στο δέντρο σημείων ενδιαφέροντος της τοποθεσίας με αναγνωριστικό <lid> και για κάθε i ∈ {1, ..., n}, <pid_i>,<$type_i$> και <$distance_i$> είναι το αναγνωριστικό, ο τύπος και η απόσταση από το σημείο αναφοράς της τοποθεσίας,

αντίστοιχα, του σημείου ενδιαφέροντος που αντιστοιχεί στον i-οστό κόμβο του δέντρου αυτού (βάσει της ενδοδιατεταγμένης διάσχισης).

- A <pid> <lid>

Γεγονός που υποδηλώνει ότι ένα σημείο ενδιαφέροντος με αναγνωριστικό <pid> που βρίσκεται στην τοποθεσία με αναγνωριστικό <lid> δεν θα είναι πλέον διαθέσιμο στους επισκέπτες. Κατά το γεγονός αυτό αρχικά θα αναζητήσετε στο πίνακα τοποθεσιών την τοποθεσία με αναγνωριστικό <lid> και στη συνέχεια θα διαγράψετε από το δέντρο σημείων ενδιαφέροντός της, τον κόμβο με αναγνωριστικό <pid>. Μετά από τη διαγραφή, το δέντρο των σημείων ενδιαφέροντος της τοποθεσίας πρέπει να παραμείνει ταξινομημένο. Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

A <pid> <lid>
Location = <pid_1:$type_1$:$location_1$>, <pid_2:$type_2$:$location_2$> , ... ,<pid_n:$type_n$:$location_n$>
DONE

όπου n είναι ο αριθμός των κόμβων στο δέντρο σημείων ενδιαφέροντος της τοποθεσίας με αναγνωριστικό <lid> και για κάθε i ∈ {1, ..., n}, <pid_i>,<$type_i$> και <$distance_i$> είναι το αναγνωριστικό, ο τύπος και η απόσταση από το σημείο αναφοράς της τοποθεσίας, αντίστοιχα, του σημείου ενδιαφέροντος που αντιστοιχεί στον i-οστό κόμβο του δέντρου αυτού (βάσει της ενδοδιατεταγμένης διάσχισης).

- R <uid>

Γεγονός που υποδηλώνει την εγγραφή ενός νέου χρήση με αναγνωριστικό <uid> στο σύστημα. Κατά το γεγονός αυτό θα γίνεται εισαγωγή ενός νέου κόμβου τύπου user στο πίνακα κατακερματισμού χρηστών, ακολουθώντας την τεχνική του διπλού κατακερματισμού με ανοιχτή διευθυνσιοδότηση (όπως έχει περιγραφεί στις διαλέξεις του μαθήματος). Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

R <uid>
Users = <uid_1>, <uid_2> , ... ,<uid_n>
DONE

όπου n είναι ο αριθμός των έγκυρων εγγραφών στον πίνακα χρηστών και για κάθε i ∈ {1, ..., n}, <uid_i> είναι το αναγνωριστικό του χρήστη που αντιστοιχεί στην i-οστή έγκυρη εγγραφή του πίνακα αυτού.

- I <uid> <upid>

Γεγονός που υποδηλώνει ότι ο χρήστης με αναγνωριστικό <uid> ενδιαφέρεται για το σημείο ενδιαφέροντος με αναγνωριστικό <upid>. Κατά το γεγονός αυτό θα γίνεται πρώτα η αναζήτηση του χρήστη στον πίνακα κατακερματισμού χρηστών και στη συνέχεια η εισαγωγή του σημείου ενδιαφέροντος με αναγνωριστικό <upid> στο δέντρο σημείων ενδιαφέροντος του χρήστη. Τέλος, θα πρέπει να αυξηθεί κατά ένα ο μετρητής

upoi_count που αντιστοιχεί στο χρήστη αυτό. Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
I <uid> <upid>
        Users POI = <upid_1>, <upid_2> , ... ,<upid_n>
DONE
```

όπου n είναι ο αριθμός των κόμβων στο δέντρο σημείων ενδιαφέροντος του χρήστη με αναγνωριστικό <uid> και για κάθε i ∈ {1, ..., n}, $<upid_i>$ είναι το αναγνωριστικό του σημείου ενδιαφέροντος που αντιστοιχεί στον i-οστό κόμβο του δέντρου αυτού (βάσει της ενδοδιατεταγμένης διάσχισης).

- G <uid$_1$> <uid$_2$>

Γεγονός που υποδηλώνει τη δημιουργία μιας ομάδας δύο χρηστών με αναγνωριστικά $<uid_1>$και $<uid_2>$ οι οποίοι θα περιηγηθούν μαζί στα κοινά σημεία που τους ενδιαφέρουν. Κατά το γεγονός αυτό, θα πρέπει να εντοπίσετε τα κοινά σημεία ενδιαφέροντος από τα δέντρα των δύο χρηστών. Η χρονική πολυπλοκότητα της διαδικασίας αυτής (τομή των δύο δέντρων) πρέπει να είναι O(n + m), όπου n και m είναι το πλήθος των κόμβων των δέντρων σημείων ενδιαφέροντος του πρώτου και του δεύτερου χρήστη, αντίστοιχα. Για να το πετύχετε αυτό, θα πρέπει πρώτα να διασχίσετε το δέντρο σημείων ενδιαφέροντος του πρώτου χρήστη και να αποθηκεύσετε τα σημεία ενδιαφέροντός του σε ένα βοηθητικό πίνακα. Το μέγεθος του πίνακα αυτού είναι γνωστό και ίσο με upoi_count (δηλαδή ίσο με το πλήθος των κόμβων του δέντρου σημείων ενδιαφέροντος του χρήστη). Έπειτα θα διασχίσετε το δέντρο σημείων ενδιαφέροντος του δεύτερου χρήστη και θα ελέγχετε για κοινά σημεία με τον παραπάνω πίνακα, τα οποία και θα τυπώσετε, όπως φαίνεται παρακάτω. Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

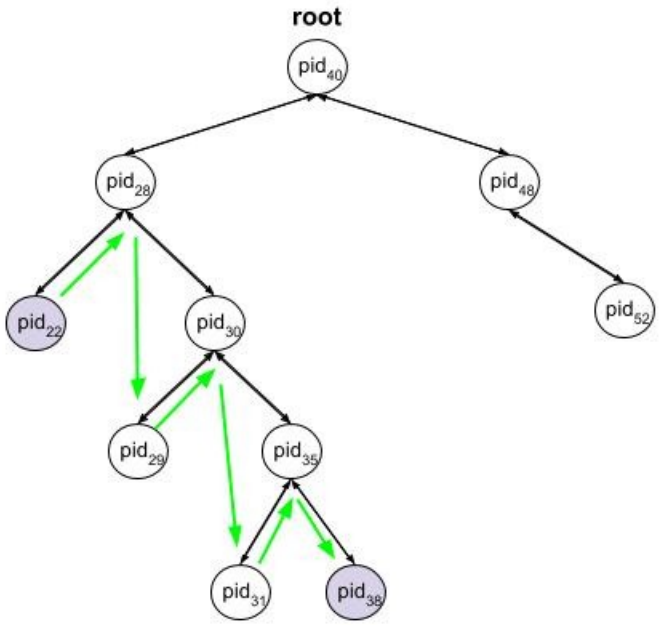
```
G <uid_1> <uid_2>
        USER 1 = <pid_1,1>, <pid_1,2>, ... <pid_1,n1>
        USER 2 = <pid_2,1>, <pid_2,2>, ... <pid_2,n2>
        Intersection = <pid_1>, <pid_2>, ..., <pid_m>
DONE
```

για κάθε j, 1≤ j ≤ 2, n_j είναι το πλήθος των κόμβων του δέντρου σημείων ενδιαφέροντος του χρήστη που αντιστοιχεί στην j-οστή παράμετρο του γεγονότος, και για κάθε i ∈ {1, ..., n_j}, pid_j είναι το αναγνωριστικό του σημείου ενδιαφέροντος που αντιστοιχεί στον i-στο κόμβο του δέντρου σημείων ενδιαφέροντος (βάσει της ενδοδιατεταγμένης διάσχισης) του j-οστού χρήστη και m είναι το πλήθος των κοινών σημείων ενδιαφέροντος των δύο χρηστών.

- B <lid> <pid$_1$> <pid$_2$>

Γεγονός που υποδηλώνει τη διάσχιση του δέντρου σημείων ενδιαφέροντος της τοποθεσίας με αναγνωριστικό <lid> , ξεκινώντας από τον κόμβο με αναγνωριστικό ίσο με το μικρότερο από τα $<pid_1>$ και $<pid_2>$, έως τον κόμβο με αναγνωριστικό ίσο με το

μεγαλύτερο από τα <pid_1> και <pid_2>. Για τη μετάβαση από τον έναν κόμβο στον άλλο, θα πρέπει να χρησιμοποιήσετε αλγόριθμο (InorderTreeSuccessor) που θα βρίσκει τον επόμενο ενός κόμβου στην ενδοδιατεταγμένη διάσχιση (τον οποίο θα υλοποιήσετε). Κατά το γεγονός αυτό, θα πρέπει πρώτα να εντοπίσετε την τοποθεσία με αναγνωριστικό <lid> στον πίνακα τοποθεσιών. Έπειτα, θα επιλέξετε το μικρότερο από τα <pid_1> και <pid_2> (έστω ότι είναι το <pid_1>) και θα εντοπίσετε τον κόμβο με αναγνωριστικό <pid_1> στο δέντρο σημείων ενδιαφέροντος της τοποθεσίας <lid>. Στη συνέχεια, θα καλείτε την InorderTreeSuccessor() επαναληπτικά, έως ότου φτάσετε στον κόμβο με αναγνωριστικό <pid_2> (τυπώνοντας στην πορεία τα αναγνωριστικά των κόμβων τους οποίους διασχίσατε). Στο παρακάτω σχήμα παρουσιάζεται ένα παράδειγμα εκτέλεσης αυτού του γεγονότος:



Σχήμα 4: Στο σχήμα αυτό παρουσιάζεται το δέντρο σημείων ενδιαφέροντος μιας τοποθεσίας. Τα αναγνωριστικά pid_{22} και pid_{38} δόθηκαν ως ορίσματα κατά το γεγονός Β. Τα πράσινα βελάκια δηλώνουν τη σειρά με την οποία γίνεται η διάσχιση των κόμβων, βρίσκοντας σε κάθε βήμα τον επόμενο στην ενδοδιατεταγμένη διάσχιση του τρέχοντος κόμβου. Επομένως, η σειρά επίσκεψης των κόμβων είναι: (22, 28, 29, 30, 31, 35, 38).

Κατά την εκτέλεση ενός τέτοιου γεγονότος, το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

B <lid> <pid_1> <pid_2>
Location: lid
Inorder Traversal: <pid_1:$type_1$:$distance_1$>, ... , <pid_n:$type_n$:$distance_n$>
DONE

όπου n είναι ο αριθμός των κόμβων του δέντρου σημείων ενδιαφέροντος της τοποθεσίας με αναγνωριστικό <lid> που πρέπει να τυπωθούν και για κάθε i ∈ {1,...,n}, <pid_i>,<$type_i$> και <$distance_i$> είναι το αναγνωριστικό, ο τύπος και η απόσταση από το σημείο αναφοράς της τοποθεσίας, αντίστοιχα, του σημείου ενδιαφέροντος που αντιστοιχεί στον i-οστό κόμβο που πρέπει να τυπωθεί.

- **X**

Γεγονός τύπου print locations το οποίο σηματοδοτεί την εκτύπωση όλων των τοποθεσιών από τον πίνακα τοποθεσιών. Για την κάθε τοποθεσία θα πρέπει να εκτυπώνονται όλα τα σημεία ενδιαφέροντός της. Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

X
LOCATIONS:
$Location_1$: <$pid_{1,1}$:$type_{1,1}$:$location_{1,1}$>, ... ,<$pid_{1,n1}$:$type_{1,n1}$:$location_{1,n1}$>
$Location_2$: <$pid_{2,1}$:$type_{2,1}$:$location_{2,1}$>, ... ,<$pid_{2,n2}$:$type_{2,n2}$:$location_{2,n2}$>
. . .
$Location_k$: <$pid_{k,1}$:$type_{k,1}$:$location_{k,1}$>, ... ,<$pid_{k,nk}$:$type_{k,nk}$:$location_{k,nk}$>
DONE

όπου k είναι ο αριθμός των έγκυρων εγγραφών στο πίνακα τοποθεσιών, για κάθε j, $1 \le j \le k$, n_j είναι το πλήθος των κόμβων του δέντρου σημείων ενδιαφέροντος της j-οστής τοποθεσίας, και για κάθε i ∈ {1 ... n_j}, <$pid_{j,i}$> , <$type_{j,i}$> και <$distance_{j,i}$> είναι το αναγνωριστικό, το είδος και η απόσταση (από στο σημείο αναφοράς της τοποθεσίας), του σημείου ενδιαφέροντος που αντιστοιχεί στον i-στο κόμβο του δέντρου της j-οστής τοποθεσίας (βάσει της ενδοδιατεταγμένης διάσχισης).

- **Y**

Γεγονός τύπου print users το οποίο σηματοδοτεί εκτύπωση όλων των χρηστών του πίνακα κατακερματισμού χρηστών. Για τον κάθε χρήστη θα πρέπει να εκτυπώνονται όλα τα σημεία ενδιαφέροντός του. Κατά την εκτέλεση ενός τέτοιου γεγονότος, το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

Y
USERS:
$User_1$ = <$upid_{1,1}$>, <$upid_{1,2}$>, ... <$upid_{1,n1}$>
$User_2$ = <$upid_{2,1}$>, <$upid_{2,2}$>, ... <$upid_{2,n2}$>
. . .
$User_k$ = <$upid_{k,1}$>, <$upid_{k,2}$>, ... <$upid_{k,nk}$>
DONE

όπου k είναι το πλήθος των έγκυρων εγγραφών στον πίνακα χρηστών, για κάθε j, $1 \leq j \leq k$, n_j είναι το πλήθος των κόμβων του δέντρου των σημείων ενδιαφέροντος του j-οστού χρήστη, και για κάθε $i \in \{1 \ldots n_j\}$, $<upid_{j,i}>$, είναι το αναγνωριστικό του σημείου ενδιαφέροντος που αντιστοιχεί στον i-στο κόμβο του δέντρου (βάσει της ενδοδιατεταγμένης διάσχισης) του j-οστού χρήστη.

- Z <uid>

Γεγονός τύπου search user το οποίο σηματοδοτεί την αναζήτηση του χρήστη με αναγνωριστικό uid στο πίνακα χρηστών. Όταν ο χρήστης εντοπιστεί, θα πρέπει να εκτυπώνεται το δέντρο με τα σημεία ενδιαφέροντός του. Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος το πρόγραμμα θα πρέπει να τυπώνει την ακόλουθη πληροφορία:

```
Z <uid>
        Interesting_poi: <upid1>, <upid2>, ... , <upidn>
DONE
```

όπου n είναι το πλήθος των κόμβων του δέντρου σημείων ενδιαφέροντος του χρήστη με αναγνωριστικό <uid> και για κάθε $i \in \{1, ..., n\}$, $<upid_i>$ είναι το αναγνωριστικό του σημείου ενδιαφέροντος που αντιστοιχεί στον i-οστό κόμβο του δέντρου αυτού (βάσει της ενδοδιατεταγμένης διάσχισης).

Βαθμολογία γεγονότων

L	**10**
P	**15**
A	**15**
R	**12**
I	**13**
G	**13**
B	**10**
X	**4**
Y	**4**
Z	**4**

Δομές Δεδομένων

Στην υλοποίησή σας δεν επιτρέπεται να χρησιμοποιήσετε έτοιμες δομές δεδομένων (πχ., ArrayList) είτε η υλοποίηση πραγματοποιηθεί στη C είτε στη Java. Στη συνέχεια παρουσιάζονται οι δομές σε C που πρέπει να χρησιμοποιηθούν για την υλοποίηση της παρούσας εργασίας.

```
struct location {
  int lid;
  struct poi *poi_tree;
  struct poi *poi_sentinel;
};

struct poi {
  int pid;
  int distance;
  int type;
  struct poi *lc;
  struct poi *rc;
  struct poi *p;
};

struct user {
  int uid;
  int upoi_count;
  struct user_poi *interesting_poi;
};

struct user_poi {
  int upid;
  struct user_poi *lc;
  struct user_poi *rc;
};
```

```
/* maximum number of locations */
#define MAX_LOCATIONS 10000

/* global variable, array of locations*/
struct location locations_array[MAX_LOCATIONS];

/* global variable, users' hashtable */
struct user *users_hashtable;

/* global variable, array of primes */
extern int primes_g[160];

/* global variable, maximum number of users */
extern unsigned int max_users_g;

/* global variable, maximum user id */
extern unsigned int max_uid_g;
```