Ανάπτυξη Λογισμικού Πληροφορικής
Χειμερινό Εξάμηνο 2016-2017

“Εύρεση συντομότερων μονοπατιών σε γράφο”

Υπέυθυνος Καθηγητής	**Συνεργάτες Μαθήματος**
Ιωάννης Ιωαννίδης	Μαίλης Θεόφιλος
	Γιαννακόπουλος Αναστάσιος
	Γαβάνας Χρήστος
	Γαλούνη Κωνσταντίνα
	Λιβισιανός Τσαμπίκος
	Μιχαηλίδης Θοδωρής

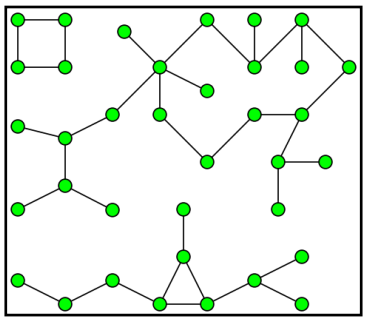
Β' Μέρος της Άσκησης

1. Περιγραφή Λειτουργικότητας και Δομών

Στο 2ο επίπεδο της εργασίας, θα υλοποιηθούν δομές με στόχο τη ταχύτερη απάντηση σε ερωτήματα εύρεσης συντομότερου μονοπατιού μεταξύ δύο κόμβων στο γράφο. Οι τρεις βασικές λειτουργίες που θα υλοποιηθούν είναι i) ένας αλγόριθμος για την "Εύρεση Συνεκτικών Συνιστωσών" στο γράφο , ii) ο αλγόριθμος του "**Tarjan**" για την εύρεση των "Ισχυρά Συνεκτικών Συνιστωσών Προσανατολισμένου Γραφήματος" (Strongly connected components), καθώς και iii) το ευρετήριο "**Grail**", το όποιο είναι μια δομή που χρησιμοποιείται για απάντηση ερωτημάτων μεταβατικότητας μεταξύ δύο κόμβων (reachability queries).

Εύρεση Συνεκτικών Συνιστωσών Γραφήματος (Connected Components)

Θεωρούμε ένα μη κατευθυνόμενο γράφημα. Συνεκτικό γράφημα ονομάζεται ένα υποσύνολο κόμβων του αρχικού γράφου, για το οποίο ισχύει ότι για κάθε δύο κόμβους του υποσυνόλου υπάρχει μονοπάτι που τους συνδέει. Σε ένα γράφο μπορεί να υπάρχουν περισσότερα από ένα συνεκτικά γραφήματα.



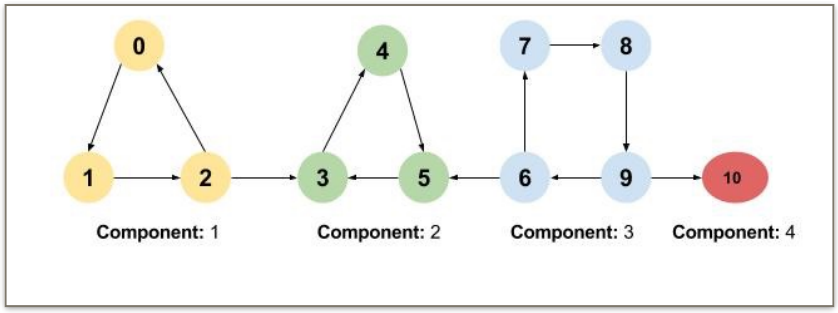
Σχήμα 1. *Γράφος και τα (weakly) connected components που ορίζονται*

Τα connected components ενός γράφου μπορούν να υπολογιστούν με την χρήση αναζήτησης κατά βάθος ή πλάτος (BFS ή DFS). Ως αρχή της αναζήτησης μπορεί να οριστεί οποιοσδήποτε κόμβος, η αναζήτηση θα συνεχιστεί μέχρι να επισκεφτούν όλοι οι κόμβοι που συνδέονται με την αφετηρία. Οι κόμβοι που επισκέφτηκαν ανήκουν στην ίδια συνιστώσα. Περαιτέρω αναζητήσεις πρέπει να πραγματοποιηθούν μέχρι να έχουν επισκεφτούν όλοι οι κόμβοι του γράφου και συνεπώς να έχουμε ανακαλύψει όλες τις συνεκτικές συνιστώσες του.

Εύρεση Ισχυρά Συνεκτικών Συνιστωσών Προσανατολισμένου Γραφήματος (Strongly Connected Components)

Σε ένα προσανατολισμένο γράφο, "Ισχυρά Συνεκτική Συνιστώσα" ονομάζεται κάθε υποσύνολο κόμβων του αρχικού γράφου, για το οποίο ισχύει ότι για κάθε δύο κόμβους του υποσυνόλου υπάρχει μονοπάτι που τους συνδέει. Σε ένα προσανατολισμένο γράφημα

μπορεί να υπάρχουν περισσότερες από μία συνεκτικές συνιστώσες. Στο παρακάτω σχήμα όπως φαίνεται, υπάρχουν 4 ισχυρά συνεκτικές συνιστώσες.



Σχήμα 2. *Γράφος και τα strongly connected components που ορίζονται*

Για την ανεύρεση των Strongly Connected Components υπάρχουν διάφοροι αλγόριθμοι στη βιβλιογραφία. Στη συγκεκριμένη περίπτωση θα χρησιμοποιηθεί ο Αλγόριθμος του Tarjan[1] όπως περιγράφεται στο σχετικό σύνδεσμο. Ο αλγόριθμος εκτελείται σε γραμμικό χώρο και βασίζεται σε μια κατά βάθος διάσχιση (Depth First Search) ενός κατευθυνόμενου γραφήματος. Αναφορικά με την υλοποίηση του αλγορίθμου είναι σημαντικό να χρησιμοποιηθούν δομές οι οποίες θα εξασφαλίσουν τη γρήγορη εκτέλεση του αλγορίθμου, όπως για παράδειγμα η γρήγορη εξακρίβωση των ήδη επισκεπτόμενων κόμβων, η αντικατάσταση των αναδρομικών κλήσεων με δομές επανάληψης κλπ.

```
struct SCC{
      Component* components; // Components index - a vector which stores
                                the components information
      uint32_t components_count;
      uint32_t id_belongs_to_component[N]; //inverted index
      …
};
```

Η δομή SCC αποθηκεύει την πληροφορία των components και χρησιμοποιείται για την απάντηση των σχετικών ερωτημάτων. Πιο συγκεκριμένα, αποθηκεύει για κάθε component ποιους κόμβους του γραφήματος αυτό εμπεριέχει.

[1] https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm

```
struct Component{

      uint32_t component_id; //current component id

      uint32_t included_nodes_count; //number of nodes in component

      uint32_t* included_node_ids;  //ids of included nodes

      …

};
```

Η δομή Component αποθηκεύει το id του component και τα ids των κόμβων που εμπεριέχονται σε αυτό.

Το πρόγραμμα σας πρέπει να υποστηρίζει τις παρακάτω λειτουργίες.

```
SCC* estimateStronglyConnectedComponents(NodeIndex* graph);
```

Η παραπάνω συνάρτηση δέχεται ως είσοδο ένα γράφημα και υπολογίζει τα strongly connected components. Το κάθε component έχει ως αναγνωριστικό έναν αύξων θετικό αριθμό και εμπεριέχει ως δεδομένα τα ids των κόμβων που αποτελείται.

```
int findNodeStronglyConnectedComponentID(SCC* components, uint32_t nodeId);
```

Η συνάρτηση αυτή για τον κόμβο (nodeId) του γραφήματος ως όρισμα, επιστρέφει το id του component το οποίο ανήκει. Η απάντηση κάθε επερώτησης πρέπει να εκτελείται γρήγορα σε σταθερό χρόνο.

```
OK_SUCCESS iterateStronglyConnectedComponentID(SCC* components,
ComponentCursor* cursor);
```

Η συγκεκριμένη συνάρτηση εκκινεί μια διαδικασία σάρωσης για όλα τα components που εμπεριέχονται στη δομή SCC και αρχικοποιεί κατάλληλα τη δομή τύπου “ComponentCursor”. Η δομή αυτή πρέπει να κρατάει κατά τη διάρκεια μιας διάσχισης των components σε ποιο component βρίσκεται τη τρέχουσα χρονική στιγμή.

```
struct ComponentCursor{

      Component* component_ptr; // pointer to current’s iteration component

      …     // Any other necessary information in order to move to next
               component in the vector

};
```

```
bool next_StronglyConnectedComponentID(SCC* components, ComponentCursor*
cursor);
```

Σκοπός της συνάρτησης αυτής είναι να προχωρήσει τον cursor στο επόμενο component. Όσο η διάσχιση δεν έχει ολοκληρωθεί, αφού ενημερωθεί κατάλληλα το περιεχόμενο του curosr, επιστρέφει true. Όταν ολοκληρωθεί η διάσχιση επιστρέφει false.

```
int estimateShortestPathStronglyConnectedComponents(SCC* components,
NodeIndex* graph, uint32_t source_node, uint32_t target_node);
```

Η συνάρτηση αυτή υπολογίζει το συντομότερο μονοπάτι μεταξύ δύο κόμβων που ανήκουν στο ίδιο strongly connected component, εκτελώντας Bidirectional BFS. Επιστρέφει είτε την ελάχιστη απόσταση μεταξύ τους, είτε -1 αν οι κόμβοι δεν ανήκουν στο ίδιο component.
Παρατήρηση: Είναι σημαντικό κατά την εκτέλεση του Bidirectional BFS, να αναπτύσσονται μόνο οι κόμβοι που ανήκουν στο συγκεκριμένο component που πραγματοποιείται η τρέχουσα αναζήτηση.

```
OK_SUCCESS destroyStronglyConnectedComponents(SCC* components);
```

Η παραπάνω συνάρτηση καταστρέφει τη δομή SCC και αποδεσμεύει τη μνήμη που καταναλώνει.

Ευρετήριο ανίχνευσης μεταβατικότητας: GRAIL

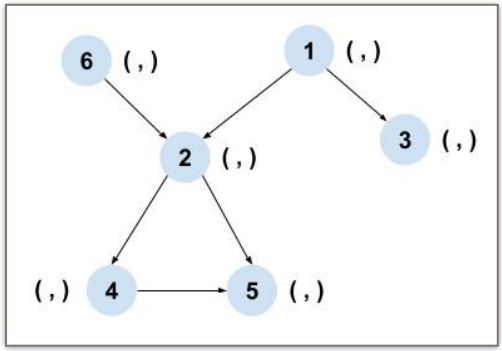
Το ευρετήριο GRAIL[2] είναι μια πιθανοτική δομή δεδομένων, η οποία ελέγχει εάν υπάρχει κάποιο μονοπάτι μεταξύ δύο κόμβων σε ένα γράφημα. Αν ένα ερώτημα μεταβατικότητας στο ευρετήριο απαντήσει “No” , τότε σίγουρα δεν υπάρχει μονοπάτι μεταξύ των κόμβων που μετέχουν στο ερώτημα. Σε περίπτωση που απαντήσει “Maybe” τότε δεν υπάρχει ξεκάθαρη απάντηση για το αν υπάρχει τέτοιο μονοπάτι στο γράφημα, δηλαδή υπάρχουν ενδεχομένως false-positives. Συνεπώς, το GRAIL χρησιμοποιείται για την άμεση απάντηση των ερωτημάτων “No”. Το ευρετήριο αυτό παρόλο που δεν απαντάει όλα τα ερωτήματα με βεβαιότητα, απαιτεί μικρή πολυπλοκότητα σε χώρο και σε χρόνο και είναι πολύ χρήσιμο σε σχέση με τις ακραίες περιπτώσεις, δηλαδή να μην είχαμε καθόλου ευρετήριο για τα ερωτήματα ή να είχαμε προϋπολογίσει και αποθηκεύσει όλα τα ζεύγη μεταβατικότητας του γράφου.

Δημιουργία ευρετηρίου

Ο αλγόριθμος λειτουργεί για γράφους που δεν εμπεριέχουν κύκλους. Θα εξηγήσουμε αργότερα πως διαχειριζόμαστε γράφους που έχουν κύκλους. Για να δημιουργήσουμε το ευρετήριο επιλέγουμε τυχαία έναν κόμβο που δεν έχουμε ήδη επισκεφθεί και εκκινούμε μια post order διάσχισης στο γράφο με αφετηρία τον κόμβο αυτό. Αρχικά, κάθε κόμβος του γράφου αποθηκεύει δύο τιμές “min_rank, rank” ως *labels.* Το label “rank” αποτελεί τη σειρά επίσκεψης ενός συγκεκριμένου κόμβου κατά την πραγματοποίηση της post order διάσχισης

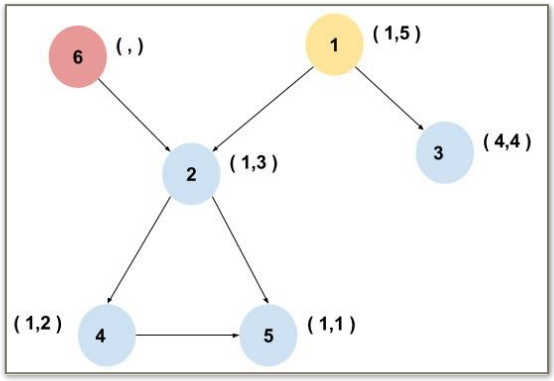
[2] http://www.vldb.org/pvldb/vldb2010/papers/R24.pdf

στο γράφο. Το label “min rank” σε έναν κόμβο αποτελεί την ελάχιστη τιμή των min_rank των παιδιών του εφόσον υπάρχουν. Άμα ο κόμβος δεν έχει καθόλου εξερχόμενες ακμές, τότε ισχύει ότι min_rank = rank. Αφότου ολοκληρωθεί η διάσχιση του συγκεκριμένου κόμβου, επιλέγουμε έναν άλλο κόμβο από το γράφο που δεν έχουμε επισκεφθεί, (δηλαδή δεν έχει τιμές στα πεδία “min_rank, rank”) και εκτελούμε την ίδια διαδικασία. Σε περίπτωση όπου ο κόμβος που συναντάμε έχει ήδη τιμή από προηγούμενο βήμα, τότε προχωρούμε την εκτέλεση σαν να είχαμε μόλις υπολογίσει τις τιμές του. Η δημιουργία του ευρετηρίου ολοκληρώνεται όταν όλοι οι κόμβοι του γράφου έχουν εξερευνηθεί.



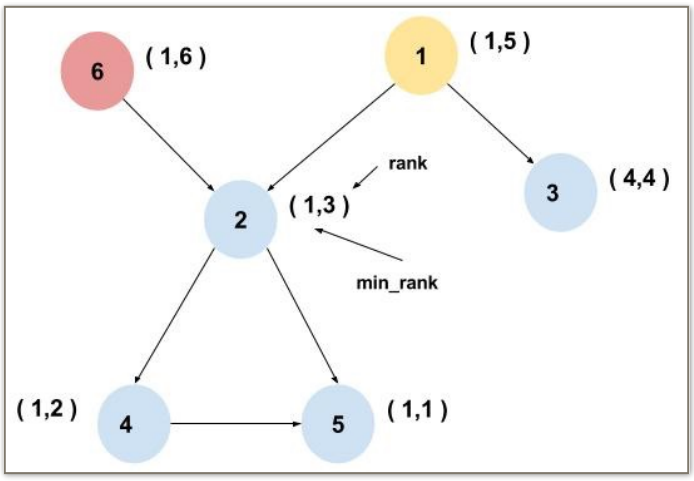
Σχήμα 3. *Ο DAG που θα δημιουργήσουμε το ευρετήριο GRAIL*

Στο παραπάνω σχήμα απεικονίζεται ένας κατευθυνόμενος ακυκλικός γράφος (**D**irected **A**cyclic **G**raph), για τον οποίο θα δημιουργήσουμε ένα ευρετήριο GRAIL. Έστω ότι καταναλώνουμε τους κόμβους σε αύξουσα σειρά με βάση τα ids τους, τότε θα υπολογίσουμε τα labels των κόμβων που μπορούμε να φθάσουμε εκκινώντας από τον κόμβο “1”. Με βάση τον αλγόριθμο δημιουργίας, μόλις τελειώσει η διάσχιση του κόμβου “1” τότε ο γράφος θα βρίσκεται στην παρακάτω κατάσταση.



Σχήμα 4. *Κατάσταση του γράφου έπειτα από την ανάπτυξη του κόμβου “1”*

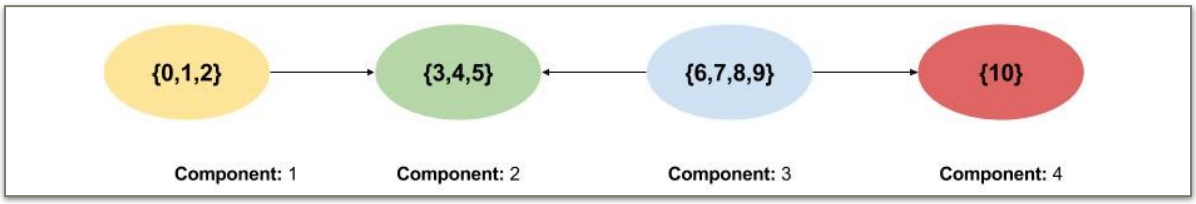
Στη συνέχεια διαπιστώνουμε ότι οι κόμβοι με id [2-5] έχουν ήδη υπολογιστεί, επομένως επιλέγουμε τον κόμβο “6” προς υπολογισμό. Ο τελευταίος, εφόσον όλοι οι εξερχόμενοι κόμβοι του έχουν ήδη υπολογιστεί θα διαμορφώσει τη τιμή του με βάση τα labels του κόμβου “2”. Η τελική κατάσταση του γράφου μετά την ολοκλήρωση του ευρετηρίου είναι η εξής.



Σχήμα 5. *Τέλικη κατάσταση του γράφου έπειτα από την ανάπτυξη όλων των κόμβων*

Διαχείριση κύκλων

Σε έναν πραγματικό γράφο είναι πολύ πιθανό να υπάρχουν κύκλοι. Ωστόσο, το ευρετήριο GRAIL λειτουργεί για ακυκλικούς γράφους, άρα είναι αναγκαίο να εξαλειφθούν οι κύκλοι. Η αφαίρεση των κύκλων επιτυγχάνεται μέσω της έρευσης των strongly connected components. Αρχικά, θα εκτελεστεί ο αλγόριθμος του Tarzan ώστε να εντοπιστούν τα components ώστε να δημιουργηθεί ένας hypergraph. Σε έναν hypergraph, οι κόμβοι ενός component ορίζουν έναν “νέο” κόμβο ο οποίος διατηρεί ως ακμές μόνο εκείνες τις ακμές οι οποίες τον συνδέουν με άλλους κόμβους του hypergraph. Για παράδειγμα, ο hypergraph που ορίζεται από το γράφο του σχήματος 1 είναι ο ακόλουθος.



Σχήμα 6. *Ο hypergraph που προκύπτει από τα SCCs*

Επερωτήσεις

Εφόσον δημιουργηθεί το ευρετήριο, ήρθε η στιγμή να εκτελέσουμε ερωτήματα. Ένα ερώτημα στο GRAIL είναι της μορφής:

X —> Y ? , όπου X,Y ο κόμβος έναρξης / τερματισμού αντίστοιχα

Για να απαντήσουμε το ερώτημα ακολουθούμε την εξής διαδικασία. Αρχικά ελέγχουμε αν τα X,Y ανήκουν στο ίδιο component. Αν ανήκουν τότε η απάντηση είναι “Yes” ολοκληρώνεται η αναζήτηση. Ειδάλλως, χρησιμοποιούμε το ευρετήριο GRAIL και ελέγχουμε την εξής συνθήκη:

Αν Y{[min_rank,rank]} ⊄ X{[min_rank,rank]}
Τότε “No”
Αλλιώς “Maybe”

Αν η απάντηση είναι “No” τότε δεν υπάρχει μεταβατικότητα. Συνεπώς, αν η απάντηση είναι “Yes” τότε εκτελούμε ένα ερώτημα υπολογισμού συντομότερου μονοπατιού εσωτερικά στο strongly connected component που ανήκουν οι 2 κόμβοι. Τέλος αν η απάντηση είναι “Maybe”, τότε ο υπολογισμός συντομότερου μονοπατιού θα πραγματοποιηθεί σε ολόκληρο το γράφο. Κατά τη διάρκεια του υπολογισμού στο Bidirectional BFS, θα ελέγχετε αν οι τρέχοντες κόμβοι που αναπτύσσονται ανήκουν στο ίδιο SCC. Σε περίπτωση που δεν ανήκουν, τότε ξαναρωτάμε το GRAIL με βάση το ids τους στο hypergraph και σε περίπτωση που αυτό απαντήσει σε κάποιο στάδιο “No”, τότε σταμάτα η αναζήτηση.

Για παράδειγμα στο γράφο του σχήματος 6 η ερώτηση **2 —> 3** απαντά “**No”**, ενώ οι ερωτήσεις **1 —> 5** και **6 —> 3** απαντούν **“Maybe”** και πρέπει να υπολογίσουμε την πραγματική απάντηση. Όπως παρατηρούμε, η απάντηση στην 1η ερώτηση ειναι **“Yes”** ενώ στη 2η ερώτηση είναι **“No”**.

Βελτίωση πιθανότητας σφάλματος με χρήση πολλαπλών labels

Η διαδικασία αναζήτησης στο ευρετήριο GRAIL εμφανίζει false positives, τα οποία έχουν ως συνέπεια την εκτέλεση BFS ερωτημάτων ώστε να πάρουμε το σωστή αποτέλεσμα. Είναι δυνατό να βελτιώσουμε τη πιθανότητα των απαντήσεων “Maybe” χρησιμοποιώντας παραπάνω από 1 labels σε κάθε κόμβο. Σε αυτήν την περίπτωση εκτελείται η δημιουργία του ευρετηρίου τόσες φορές, όσες ο αριθμός των labels που έχουμε επιλέξει. Κατά την εκτέλεση των ερωτημάτων πρέπει να ελέγχονται ξεχωριστά οι αντίστοιχες θέσεις των labels στους δύο κόμβους που μετέχουν στην αναζήτηση. Αν έστω και ένα ζευγάρι από labels επιστρέψει “No”, τότε η συνολική απάντηση του ερωτήματος είναι “No”. Τέλος, για ακόμα μεγαλύτερη αποτελεσματικότητα είναι καλό κάθε φορά που επιλέγεται ένας κόμβος για ανάπτυξη μεταξύ των πολλαπλών παιδιών ενός κόμβου να επιλέγεται τυχαία.

Λειτουργίες ευρετηρίου

```
GrailIndex* buildGrailIndex(NodeIndex* graph, SCC* components);
```

Η παραπάνω συνάρτηση δέχεται ως είσοδο ένα γράφημα, την ήδη δημιουργημένη δομή των strongly connetcted components και δημιουργεί το ευρετήριο GRAIL. Επιστρέφει τη δομή GrailIndex η οποία αποθηκεύει ολή την πληροφορία και την οργάνωση του ευρετηρίου.

```
GRAIL_ANSWER isReachableGrailIndex(GrailIndex* index, uint32_t source_node,
uint32_t target_node);
```

Η συγκεκριμένη συνάρτηση πραγματοποιεί ένα ερώτημα στο ευρετήριο GRAIL και επιστρέφει ως απάντηση το enumaration GRAIL_ANSWER = {NO=0, MAYBE=1, YES=2}.

```
OK_SUCCESS destroyGrailIndex(GrailIndex* index);
```

Η παραπάνω συνάρτηση καταστρέφει ο ευρετήριο GRAIL και αποδεσμεύει τη μνήμη που χρησιμοποιεί.

2. Ενσωμάτωση των ευρετηρίων στην επίλυση του προβλήματος

Στο 2ο επίπεδο της εργασίας το πρόγραμμα θα δέχεται ως είσοδο τόσο δυναμικούς γράφους, δηλαδή έχουν προσθήκες ακμών όσο και στατικούς οι οποίοι δε μεταβάλλονται κατά τη διάρκεια εκτέλεσης των εισερχόμενων ερωτημάτων. Στην αρχή του αρχείου του γράφου θα υπάρχει το αναγνωριστικό DYNAMIC ή STATIC GRAPH, το οποίο θα δηλώνει αν ο γράφος είναι δυναμικός ή στατικός αντίστοιχα.

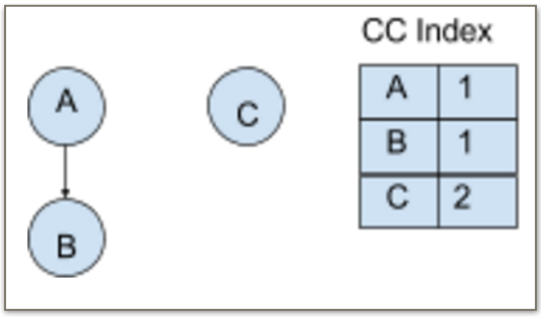
Στατικός Γράφος

Σε περίπτωση που ο γράφος είναι στατικός, τότε μόλις ολοκληρωθούν όλες οι εισαγωγές θα χτίσετε ένα ευρετήριο GRAIL με τη διαδικασία όπως περιγράφηκε στην προηγούμενη ενότητα. Σε κάθε ερώτημα εύρεσης συντομότερου μονοπατιού θα το χρησιμοποιείτε στην αρχή, ώστε να διαπιστώσετε άμα μπορεί να απαντηθεί άμεσα η μη μεταβατικότητα (απάντηση *“No”*). Σε περίπτωση που επιστρέψει απάντηση “Yes” ή “Maybe”, τότε θα εκτελέσετε την αναζήτηση κανονικά.

Δυναμικός Γράφος

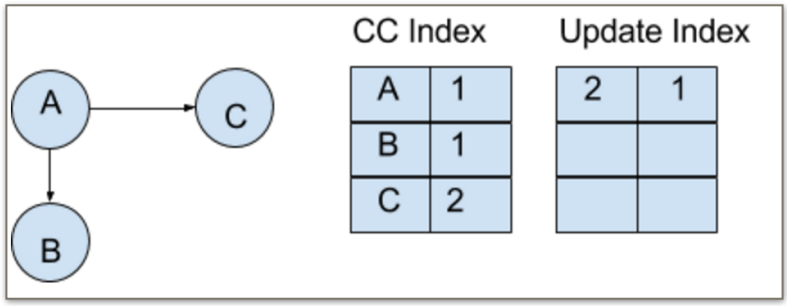
Στους δυναμικούς γράφους, ένα δημιουργημένο ευρετήριο είναι πιθανό να μην απαντήσει σωστά, όταν προκύψει μια αλλάγη στη δομή του γράφου (π.χ. εισαγωγή-διαγραφή μιας ακμής). Σε αυτή την περίπτωση θα εντοπίζετε τα connected components στον αρχικό γράφο θεωρώντας τον μη κατευθυνόμενο. Με αυτό το τρόπο μπορούμε να μειώσουμε τις αναζητήσεις καθώς είμαστε σίγουροι ότι αν 2 κόμβοι δεν βρίσκονται στο ίδιο connected component δεν πρόκειται να συνδέονται. Αντίθετα αν 2 κόμβοι ανήκουν στο ίδιο connected component τότε είναι πιθανόν να υπάρχει μονοπάτι μεταξύ τους.

Μετά την εισαγωγή του γράφου θα υπολογίζετε τα connected components και για καθε κόμβο θα αποθηκεύεται σε ποια συνιστώσα ανήκει.



Σχήμα 7. *Παράδειγμα γράφου με 3 κόμβους*

Όταν έρχεται μια καινούργια ακμή θα πρέπει να ενημερώσουμε την αποθηκευμένη πληροφορία. Ας υποθέσουμε ότι στο παράδειγμα του προηγούμενου σχήματος εισάγουμε την ακμή A->C. Τότε θα δημιουργηθεί ένα μεγάλο connected component. Η ενημέρωση όμως του CC Index είναι αργή για να γίνεται για κάθε εισαγωγή που προκαλει μεταβολή στα connected components. Για αυτό το λόγο θα δημιουργήσουμε μια δεύτερη δομή που θα αποθηκεύει τυχόν ενώσεις συνεκτικών συνιστωσών.



Σχήμα 8. *Ο γράφος και οι δομές μετα την εισαγωγή της ακμής A->C*

Για την απάντηση ερωτημάτων θα πρέπει να συμβουλεύεστε το CC Index για να βρίσκετε τις συνιστώσες που ανήκουν οι 2 κόμβοι ενός query. Αν ανήκουν στο ίδιο τότε θα πρέπει να εκτελεστεί αναζήτηση για να ανακαλύψουμε αν οι κόμβοι συνδέονται. Αν δεν ανήκουν στο ίδιο τότε θα πρέπει να συμβουλευτείτε και τον Upate Index διοτι μπορεί οι δυο αρχικές συνιστώσες να έχουν ενωθεί μέσω μιας καινούργιας ακμής. Στην περίπτωση που έχει εισαχθεί τέτοια ακμή θα πρέπει πάλι να εκτελείται αναζήτηση στο γράφο. Το Update Index θα είναι δυναμική δομή που σε χρόνο O(1) θα απαντάει αν κάποιο cc έχει ενωθεί με κάποιο άλλο.

Είναι προφανές ότι αυτή η διαδικασία μπορεί να γίνει αργή μετά από κάποιον αριθμό εισαγωγών. Θα υλοποιήσετε την ανακατασκευή του CC Index. Για να αποφασίσετε πότε θα γίνεται η ανακατασκευή θα χρησιμοποείτε μια μετρική που θα υπολογίζεται ανά ριπή και θα τη συγκρίνετε με ένα κατώφλι που θα υπολογίσετε.

$$metric = \frac{\#queries\ that\ used\ the\ Update\ Index}{\#queries}$$

Για τη δημιουργία και τη λειτουργία των ευρετηρίων θα υλοποιήσετε τις παρακάτω δομές και συναρτήσεις:

```
struct CC{
      uint32_t ccindex[]; //CCIndex
      UpdateIndex* updateIndex;
      uint32_t metricVal;
      …
};

CC* estimateConnectedComponents(NodeIndex* graph);

OK_SUCCESS insertNewEdge(CC* components, uint32_t nodeIdS, uint32_t
nodeIdE);

int findNodeConnectedComponentID(CC* components, uint32_t nodeId);

OK_SUCCESS rebuildIndexes(CC* components);

OK_SUCCESS destroyConnectedComponents(CC* components);
```

Παρατηρήσεις

- Τα πρότυπα των συναρτήσεων δίνονται σε C, στην περίπτωση υλοποίησης σε κάποια αντικειμενοστραφή γλώσσα το πρώτο όρισμα παραλείπετε.
- Επιπλέον ορίσματα μπορούν να προστεθούν στα πρότυπα των συναρτήσεων.
- Οι διαδικασίες επίλυσης του προβλήματος για δυναμικούς και στατικούς γράφους είναι διακριτές και μπορούν να μοιραστούν εύκολα.
- Ίδιες δομές και λειτουργικότητες που χρησιμοποιούνται σε πολλά σημεία δεν θα πρέπει να υλοποιούνται ξανά.