ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Εξεταστική Περίοδος Χειμερινού Εξαμήνου
2014 - 2015

Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

Υλοποίηση - Επίπεδο 1-2-3

ΟΝΟΜΑ: Γαλούνη Κωνσταντίνα **Α.Μ:** 1115201000034
ΟΝΟΜΑ: Γιαννακέλος Κωνσταντίνος **Α.Μ:** 1115201000029

ΔΙΔΑΣΚΩΝ: Ιωάννης Ιωαννίδης

ΑΘΗΝΑ – 2015

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ 2

Περιγραφή Υλοποίησης 3

Δομές Δεδομένων 6

Συναρτήσεις Part 1 11

Συναρτήσεις Part 2-Metrics 12

Συναρτήσεις Part 2-Queries 12

Συναρτήσεις Part 3-Communities 13

Συναρτήσεις Part 3-CPM 14

Βοηθητικές Συναρτήσεις & Συναρτήσεις για τη main 15

Περιγραφή Υλοποίησης

Η υλοποίηση αποτελείται από τα εξής αρχεία :

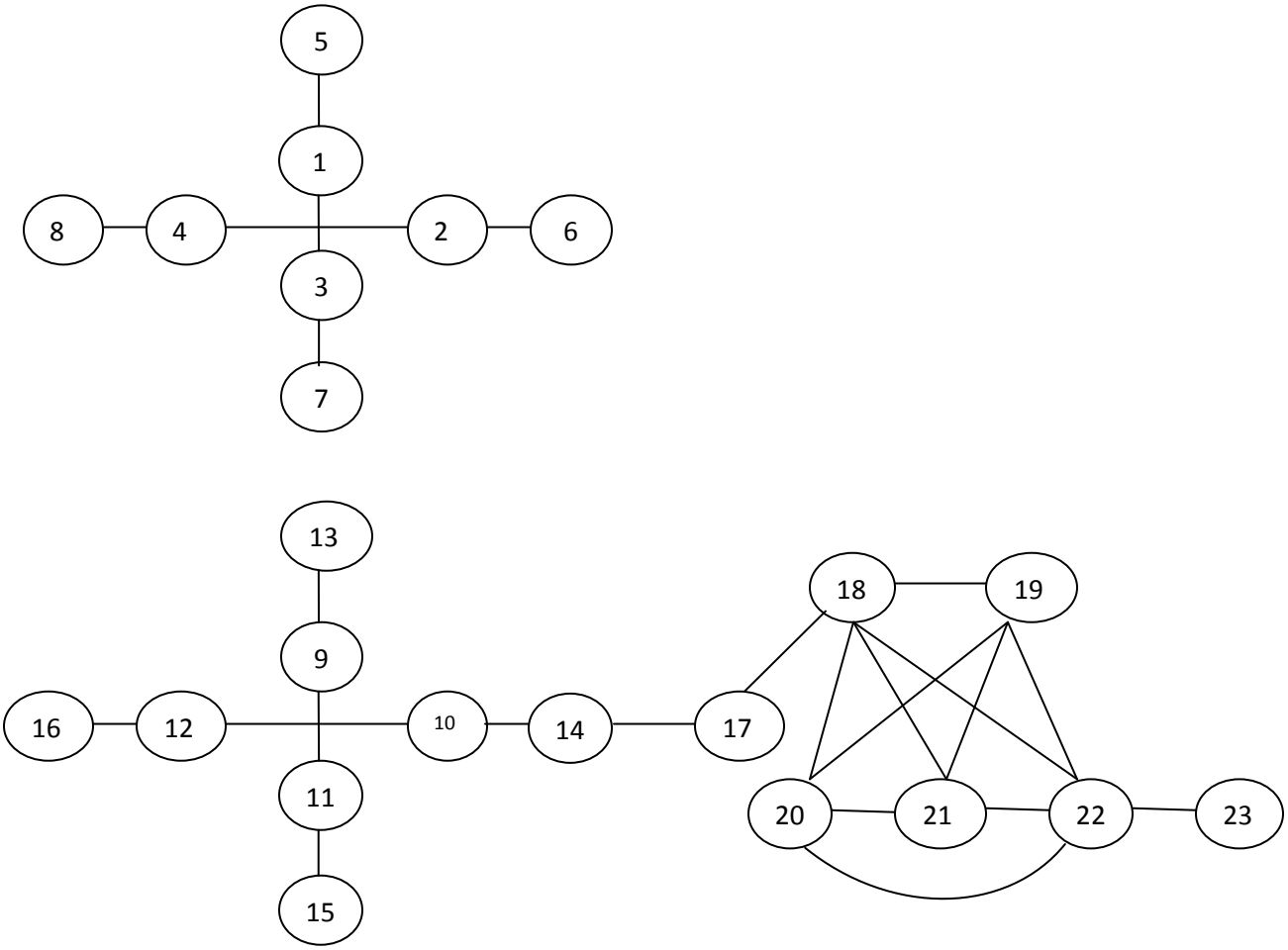
Φάκελος graph:	**graph.c**	**graph.h**	**makefile**
Φάκελος list:	**list.c**	**list.h**	**makefile**
Φάκελος metrics:	**metrics.c**	**metrics.h**	**makefile**
Φάκελος queries:	**queries.c**	**queries.h**	**makefile**
Φάκελος algorithms:	**communities.c**	**communities.h**	
	threadpool.c	**threadpool.h**	
	CPM.c	**CPM.h**	
	GN.c	**GN.h**	
	makefile		

Εχούν υλοποιηθεί τα αρχεία για τον έλεγχο της υλοποίησης :

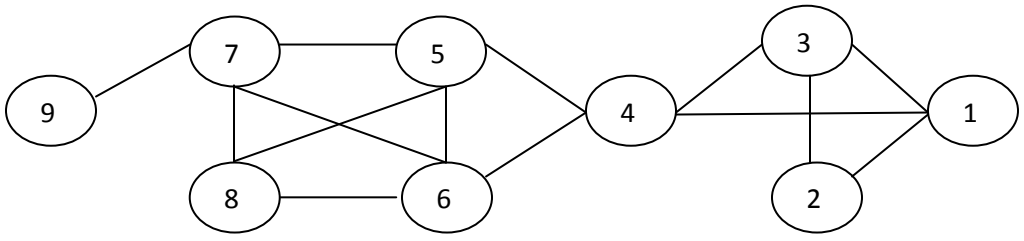
Φάκελος main:	**main1.c**	**main_functions.c**
	main2.c	**main_functions.h**
	main3.c	

Παρατηρήσεις:

- Η main1 βρίσκει τις κοινότητες μέσω του αλγορίθμου Clique Percolation Method (CPM), σε έναν τυχαίο γράφο που δημιουργήσαμε, ο οποίος μεταξύ άλλων σχηματίζει δύο ανεξάρτητες μεταξύ του 4-κλίκες, και μία 5-κλίκα που συνδέεται με μία από τις 4-κλίκες. Όλοι οι κόμβοι ανήκουν σε ένα forum. Ο γράφος αναπαριστάται στο παρακάτω σχήμα:



- Η main2 βρίσκει τις κοινότητες μέσω του αλγορίθμου Girvan-Newman (GN), στο γράφο που αναφέρεται στην εκφώνηση ως παράδειγμα. Όλοι οι κόμβοι ανήκουν σε ένα forums. Ο γράφος αναπαριστάται στο παρακάτω σχήμα:



- Η main3 βρίσκει τις κοινότητες μέσω και των 2 αλγορίθμων και έπειτα ελέγχει τα αποτελέσματα με αυτά των αρχείων που δίνονται. Ο αρχικός γράφος δίνεται από τα αρχεία της εκφώνησης.

- Η μεταγλώττιση μέσω make (περιλαμβάνεται και εξωτερικό makefile) θα μεταγλωττίσει όλα τα αρχεία οπότε η χρήση της κάθε main γίνεται άμεσα (ανάλογα με τη main της επιλογής μας, εκτελούμε ./main1 , ./main2 ή ./main3).
- Περιλαμβάνονται ακόμα τα απαραίτητα αρχεία για τον έλεγχο των αποτελεσμάτων της main3.

- Η υλοποίηση αναπτύχθηκε και δοκιμάστηκε σε Ubuntu 12.04.

  Παρακάτω θα γίνει μία περιγραφή των δομών δεδομένων που χρησιμοποιήθηκαν καθώς και των σημαντικότερων συναρτήσεων που υλοποιήθηκαν.

Δομές Δεδομένων

Graph : Η δομή αυτή αναπαριστά τον γράφο του κοινωνικού δικτύου. Περιέχει τον πίνακα κατακερματισμού ο οποίος αποθηκεύει τις εγγραφές, καθώς και όλα τα απαραίτητα δεδομένα για τη χρήση του γραμμικού κατακερματισμού. Ο πίνακας κατακερματισμού αποτελεί έναν πίνακα με δείκτες προς **Bucket.** Η δομή είναι η παρακάτω:

```
typedef struct Graph{
    void **ht;                  // hash table
    int edges;                  // number of total edges
    int allrecs;                // number of total records
    int m;                      // initial hash table size
    int c;                      // bucket size
    int p;                      // pointer to split bucket
    int i;                      // splitting round
    int size;                   // size of hash table
}Graph;
```

Bucket : Η δομή αυτή αναπαριστά τους κάδους κατακερματισμού. Αποτελείται από έναν πίνακα με void* , δηλαδή έναν πίνακα με δείκτες προς τις εγγραφές – κόμβους, το μέγεθος του και τον αριθμό των εγγραφών τον οποίο περιέχει. Η δομή είναι η παρακάτω:

```
typedef struct Bucket{
    void **cells;
    int size;
    int nrecs;
}Bucket;
```

Properties : Η δομή αυτή αναπαριστά τις ιδιότητες είτε για μία ακμή είτε για μία εγγραφή. Αποτελεί έναν πίνακα από void* ώστε να είναι δυνατή η αποθήκευση δεδομένων κάθε τύπου. Η δομή είναι η παρακάτω:

```
typedef struct Properties{
    void **properties;
}Properties;
```

Edge : Η δομή αυτή αναπαριστά τις ακμές ανάμεσα στους κόμβους. Αποτελεί μία λίστα η οποία κρατάει το id του κόμβου στο άλλο άκρο, καθώς και μία δομή **Properties** η οποία αναπαριστά τη σχέση της ακμής. Η δομή είναι η παρακάτω:

```
typedef struct Edge{
        int id;
        Properties      *props;
        struct Edge     *next;
}Edge;
```

Record : Η δομή αυτή αναπαριστά τον κόμβο – εγγραφή του γράφου. Αποτελείται από το id της εγγραφής, μία λίστα ακμών **Edge** και μία δομή ιδιοτήτων **Properties.** Η δομή είναι η παρακάτω:

```
typedef struct Record{
    int id;
    int degree;
    Edge  *edges;
    Properties *props;
    Graph *likes;                       // likes from record to other people
    Graph *forums;                      // forums record belongs to
    Graph *interests;                   // interests a record has
    Graph *generic;                     // multipurpose generic graph
}Record;
```

Pair : Αποτελεί τη δομή που επιστρέφει η συνάρτηση **next.** Περιέχει τον **κόμβο – target** και την απόσταση μεταξύ του **source – target** .

```
struct pair {
    int ID;
    int distance;
};
typedef struct pair Pair;
```

ResultSet : Αποτελεί τη δομή που επιστρέφει η συνάρτηση **reachNodeN**, η οποία κρατάει τα κατάλληλα δεδομένα για τον υπολογισμό των αποστάσεων μέσω της συνάρτησης next. Συγκεκριμένα κρατάει το id του **κόμβου – source**, τον αριθμό του κάδου και τον αριθμό της εγγραφής (για το συγκεκριμένο κάδο) η οποία είναι η επόμενη προς υπολογισμό.

```
typedef struct ResultSet{
    Graph *graph;
    int from;
    int hash;
    int record;
}ResultSet;
```

List_Node : Αποτελεί τη δομή που αναπαριστά τους κόμβους για τις ουρές του Bidirectional Breadth First Search, η οποία περιέχει ένα δείκτη σε void (στα properties φυλάσσεται η απόσταση του κόμβου από τον **κόμβο - source** ή **target** αντίστοιχα) και ένα δείκτη next για τη σύνδεση του ενός κόμβου της ουράς με τον επόμενό του.

```
typedef struct List_Node{
    void *node;
    struct List_Node *next;
}List_Node;
```

List : Αποτελεί τη δομή που αναπαριστά τη γενική λίστα που χρησιμοποιείται.

```
typedef struct List{
    List_Node *first;
    List_Node *last;
}List;
```

Matches,Stalkers: Η δομή που χρησιμοποιείται για την αποθήκευση των αποτελεσμάτων για τα queries 1 , 2 .

```
typedef struct Matches{
    long id;
    double score;
}Matches,Stalkers;
```

ForumPopulation: Η δομή που κρατάει τα στοιχεία των top-N forum.

```
typedef struct ForumPopulation{
    int id;
    char *name;
    int population;
}ForumPopulation;
```

Forums: Η δομή που κρατάει τα top-N forums με τους αντίστοιχους γράφους που δημιουργήθηκαν.

```
typedef struct Forums{
    ForumPopulation *nForums;
    Graph **gForums;
}Forums;
```

ArgsA,ArgsB: Οι δομές που αποτελούν τα ορίσματα για τις συναρτήσεις των thread (pthread_create).

```
typedef struct ArgsA{
    Graph *forums;
    int N;
}ArgsA;
```

```
typedef struct ArgsB{
    Graph *g;
    int index;
}ArgsB;
```

Community: Η δομή του κάθε community.

Communities: Η λίστα με τα communities.

```
typedef struct Community{
    int id;
    Graph *members;
}Community;
typedef struct Communities{
    int numberOfCommunities;
    List *communities;
}Communities;
```

Threadpool

```
typedef struct threadpool {
    pthread_cond_t empty;
    pthread_mutex_t lock1;
    pthread_mutex_t lock2;
    pthread_t *wt;                  /* working threads */
    List *queue;                    /* job queue */
    int threadpoolexit;
    int current;
}threadpool;
```

Treenode: Η δενδρική δομή για την ανακάλυψη των κλικών στη μέθοδο CPM.

```
typedef struct Treenode{
    int id;
    Graph *visited;
    List *children;
    struct Treenode *parent;
}Treenode;
```

Συναρτήσεις Part 1

```
int reachNode1(int start, int end, Graph *g)
```

Η συνάρτηση αυτή βρίσκει την ελάχιστη απόσταση μεταξύ δύο κόμβων start και end. Σε περίπτωση που υπάρχει μονοπάτι σύνδεσης επιστρέφει την απόσταση του συντομότερου μονοπατιού, ενώ αν οι κόμβοι δε συνδέονται επιστρέφει INFINITY_REACH_NODE το οποίο έχει οριστεί ως ο μεγαλύτερος ακέραιος στο αρχείο επικεφαλίδας. Για την υλοποίηση δημιουργεί δύο δομές **Graph**, για την αποθήκευση των κόμβων που έχει συναντήσει κάθε μονοπάτι (εκμεταλλεύεται τον ήδη υλοποιημένο πίνακα κατακερματισμού για βέλτιστη αναζήτηση) και δύο δομές **List**, για την αποθήκευση των κόμβων που πρόκειται να εξερευνηθούν. Συγκεκριμένα, υλοποιείται Bidirectional search , με εκτέλεση Breadth First Search από τον **κόμβο – source** στον **κόμβο – target** και αντίστροφα και γι 'αυτό καλείται η συνάρτηση bfs (αρχείο queue.c).

```
void bfs(List **queue,Graph *visited1,Graph *visited2,Record
**node1,Record **node2,Graph *g,int *min)
```

Η συνάρτηση αυτή υλοποιεί τον BFS αλγόριθμο, λαμβάνοντας υπόψη και τις συνθήκες τερματισμού, δεδομένου ότι χρησιμοποιείται στα πλαίσια bidirectional bfs. Τα ορίσματα queue, visited1 και node1 αναφέρονται στο μονοπάτι που εξετάζεται κάθε φορά και είναι η ουρά των υπό εξέταση κόμβων, ο πίνακας κατακερματισμού στον οποίο βρίσκονται οι κόμβοι που έχει συναντήσει το μονοπάτι και ο κόμβος που θα «επιστρέψει» η συνάρτηση ως ο τελευταίος που εξερευνήθηκε. Τα ορίσματα visited2 και node2 αφορούν στο άλλο μονοπάτι. Το g είναι ο αρχικός μας γράφος με όλους τους κόμβους και τα χαρακτηριστικά τους και τέλος, το min είναι ένας ακέραιος ώστε να καθίσταται δυνατή η σύγκριση της απόστασης πιθανών λύσεων. Εσωτερικά, εξάγει τον πρώτο κόμβο από την ουρά, αν αυτός υπάρχει, και για κάθε ακμή του, αν δεν υπάρχει στη δομή visited1 ο κόμβος που καταλήγει η ακμή, τον δημιουργεί και τον εισάγει τόσο στη visited1, όσο και στην ουρά. Παράλληλα, αν ο κόμβος αυτός υπάρχει στη δομή visited2 (το άλλο μονοπάτι τον έχει συναντήσει), τότε έχει βρεθεί μία πιθανή λύση. Για να

```
ResultSet* reachNodeN(int id, Graph *g)
```

Η συνάρτηση δέχεται ένα id και επιστρέφει μία δομή **ResultSet,** την οποία θα χρησιμοποιήσει η συνάρτηση next.

```
int next(ResultSet *r, Pair *p)
```

Η συνάρτηση δέχεται μία δομή **ResultSet** και κάθε κλήση της επιστρέφει μία απόσταση από έναν άλλο διαφορετικό κόμβο στο γράφο. Συγκεκριμένα χρησιμοποιεί την reachNode1 για τον υπολογισμό της απόστασης, όπου τα ορίσματά της είναι εύκολα προσπελάσιμα μέσω της δομής **ResultSet** η οποία κρατάει τη θέση του επόμενου κόμβου στο hash_table.

Συναρτήσεις Part 2-Metrics

```
void degreeDistribution(Graph* g)
```

Αρχικά δημιουργείται ένα αρχείο Degree Distribution με τους διαφορετικούς βαθμούς και τα ποσοστά τους στο γράφο. Στη συνέχεια το αρχείο ταξινομείται μέσω script στο κέλυφος και δίνεται ως παράμετρος στο gnuplot. Το διάγραμμα παραμένει ανοιχτό εως ώτου ο χρήστης επιλέξει να κλείσει το παράθυρο.

```
double betweennessCentrality(Record* n, Graph* g)
```

Έχει υλοποιηθεί ο αλγόριθμος του Brande:

http://www.inf.uni-konstanz.de/algo/publications/b-fabc-01.pdf

Συναρτήσεις Part 2-Queries

```
Matches* matchSuggestion(Record* n, int k, int h, int x, int
limit, Graph* g)
```

Αρχικά χρησιμοποιείται μία αναζήτηση κατα πλάτος (BFS) στον γράφο **person knows person** μέχρι βάθος k και οι εγγραφές αποθηκεύονται σε ένα γράφο (εκτός των ήδη φίλων) . Ύστερα

φιλτράρουμε το γράφο αυτό ανάλογα με τα κριτήρια γένους,ενδιαφερόντων,ηλικίας και χώρου εργασίας-διαμονής. Τέλος ταξινομούνται οι εγγραφές και αποθηκεύονται στη δομή Matches που επιστρέφεται.

```
Graph* getTopStalkers(int k, int x, int centralityMode, Graph*
g,Stalkers* st)
```

Για κάθε εγγραφή , βρίσκονται οι γείτονες-φίλοι οι οποίοι αποθηκεύονται σε ένα γράφο friends. Στη συνέχεια ελέγχονται τα likes που έχει κάνει η συγκεκριμένη εγγραφή σε εγγραφές οι οποίοι δεν είναι φίλοι και αν ο αριθμός ξεπερνά τον x αριθμό likes τότε αυτή η εγγραφή θεωρείται stalker. Έπειτα δημιουργούνται οι ακμές του γράφου stalkers. Τέλος υπολογίζεται το centrality όλων των stalkers ενώ παράλληλα τοποθετούνται στη δομή `Stalkers` αυτοί με το μεγαλύτερο centrality.

```
void findTrends(int k, Graph *g, char **womenTrends, char
**menTrends)
```

Αρχικά χωρίζεται ο γράφος **person knows person** σε δύο υπογράφους men-women. Ύστερα για κάθε tag , βρίσκουμε στους δύο υπογράφους τα μεγαλύτερα connected component, μέσω της find_maxCC, του συγκεκριμένου tag και τα αποθηκεύουμε ανάλογα με το αν χωράνε στα k-top trends.

```
void buildTrustGraph(int forumID, Graph *g)
```

Αρχικά δημιουργείται ο γράφος του συγκεκριμένου forumID και στη συνέχεια εισάγονται “trust” ακμές στο γράφο αυτό σύμφωνα με τον υπολογισμό της εμπιστοσύνης ανάμεσα στα μέλη. Ο υπολογισμός της εμπιστοσύνης πραγματοποιείται από τη συνάρτηση estimateTrust.

Συναρτήσεις Part 3-Communities

```
ForumPopulation** computeTopNForums(Graph *g, int n)
```

Αρχικά πραγματοποιείται διάβασμα για το αρχείο forum_hasMember_person και αποθηκεύεται σε μία ουρά του threadpool όλα τα forumID και το polulation του καθενός. Στη συνέχεια δημιουργούνται 8 threads, καθένα από τα οποία παίρνει από την ουρά ένα forum και υπολογίζει τη θέση του στα top-N (`void calculatePolulation(void *args)`).

```
Graph** computeTopNGraphs(Graph *g, int n)
```

Έχοντας ήδη βρεθεί τα top-N forums, δημιουργούνται N threads, ώστε να υπολογιστεί για κάθε ένα από αυτά ο γράφος του (`void createForumGraph(void *args)`).

Συναρτήσεις Part 3-CPM

```
Communities *cliquePercolationMethod(int k, Graph *g)
```

Αρχικά για κάθε εγγραφή ανακαλύπτονται οι γείτονές της (που δεν έχουν ήδη εξερευνηθεί), και ελέγχουμε αν κάθε γείτονας συνδέεται με τουλάχιστον k-2 από τους υπόλοιπους. Στην περίπτωση αυτή, οι γείτονες που πληρούν τη συνθήκη εισάγονται σε ένα γράφο (neighbors), ο οποίος δίνεται ως όρισμα στη συνάρτηση explore. Η συνάρτηση αυτή είναι υπεύθυνη να εξερευνήσει αναδρομικά κάθε πιθανό μονοπάτι μεταξύ των κόμβων αυτών μέχρι βάθος k και να αποθηκεύσει τους κόμβους φύλλα που ανήκουν σε k-κλίκα. Ύστερα μέσω backtracking στους κόμβους αυτούς, ανακαλύπτονται όλες οι k-κλίκες, οι οποίες είναι οι υπερ-κόμβοι ενός νέου γράφου. Τέλος, ανακαλύπτονται όλα τα communitites (connected components) μέσω της συνάρτησης getCC.

```
Communities* GirvanNewman(double modularity, Graph *g)
```

Στο γράφο κάθε forum υπολογίζεται το edgeBetweenness όλων των ακμών, και επιστρέφεται η ακμή με το μεγαλύτερο betweenness (`int* EdgeBetweenness(Graph *g)`), ώστε να αφαιρεθεί από το γράφο μέσω της συνάρτησης removeEdge. Έπειτα υπολογίζεται το modularity σύμφωνα με τον τύπο που δόθηκε και αν μειωθεί σε σχέση με τη μέγιστη τιμή του, η ακμή που αφαιρέθηκε ξαναεισάγεται και υπολογίζονται τα communitites που προκύπτουν.

Βοηθητικές Συναρτήσεις & Συναρτήσεις για τη main

```
Graph* graph_person(int bucketsNumber, int bucketSize)
```

Η συνάρτηση αυτή επιστρέφει το γράφο με τις εγγραφές χωρίς ακμές μεταξύ τους.

```
void update_person_knows_person(Graph *g)
```

Η συνάρτηση αυτή, προσθέτει τις ακμές στον ήδη υπάρχοντα γράφο από το αρχείο person_knows_person.

ΠΑΡΑΤΗΡΗΣΗ: Οι συναρτήσει που ξεκινούν με graph επιστρέφουν πάντα γράφο, σε αντίθεση με αυτές που ξεκινούν με update, οι οποίες ανανεώνουν το γράφο με τις εγγραφές-ανθρώπους.