Εργασία 3 – Βιβλιοθήκη για λειτουργίες σε γράφους

Αναπτύξτε μια βιβλιοθήκη για την δημιουργία / διαχείριση γράφων και την υποστήριξη βασικών λειτουργιών αναζήτησης σε αυτούς. Η βιβλιοθήκη πρέπει να υποστηρίζει μια κατάλληλη διεπαφή προγραμματισμού (application programming interface -- API) μέσω της οποίας η παρεχόμενη λειτουργικότητα μπορεί να χρησιμοποιηθεί από άλλα προγράμματα/εφαρμογές.

Τα βασικά βήματα της εργασίας είναι:

1. Ορίστε την διεπαφή της βιβλιοθήκης μέσω ενός header file (**graph.h**) που περιέχει τους ορισμούς για τους τύπους δεδομένων (data types) και τα πρωτότυπα των συναρτήσεων (function prototypes) που προσφέρει η βιβλιοθήκη.
2. Σε ξεχωριστό αρχείο (**graph.c**), υλοποιήστε τις λειτουργίες που ορίζονται στο header file.
3. Αναπτύξτε ένα πρόγραμμα δοκιμής (**project3.c**), που επιτρέπει στον χρήστη να ορίζει γράφους και να πραγματοποιεί λειτουργίες αναζήτησης σε αυτούς, δίνοντας κατάλληλες εντολές.
4. Γράψτε κατάλληλο **Makefile** στο οποίο δημιουργείται μια στατική βιβλιοθήκη libgraph.a που υλοποιεί τις λειτουργίες γράφων και συνδέεται (link) με το κυρίως πρόγραμμα ώστε να παράγει εκτελέσιμο με όνομα project3. Ο στόχος που παράγει το εκτελέσιμο πρέπει να είναι ο πρώτος στόχος στο Makefile.

Ορισμός τύπων

Η βιβλιοθήκη πρέπει να ορίζει τον τύπο **graph_t** για έναν γράφο. Ο τύπος **graph_t** πρέπει να υλοποιηθεί ως αφηρημένος τύπος δεδομένων (abstract data type) χωρίς να φανερώνεται η υλοποίηση του (π.χ., δομές δεδομένων και αλγόριθμοι που χρησιμοποιεί εσωτερικά η βιβλιοθήκη για να υποστηρίζει τις λειτουργίες που σχετίζονται με αυτόν τον τύπο). Ενδεικτικά, η υλοποίηση θα μπορούσε να χρησιμοποιεί έναν πίνακα ή μια λίστα γειτνίασης (adjacency matrix/list), περισσότερα για αυτό στο φροντιστήριο.

Παρομοίως, η βιβλιοθήκη πρέπει να ορίζει τον τύπο **path_t** για μονοπάτια/διαδρομές μέσα στον γράφο, πάλι ως αφηρημένο τύπο δεδομένων, χωρίς να φανερώνεται η εσωτερική υλοποίηση.

Για μεγαλύτερη ευελιξία, οι κόμβοι του γράφου πρέπει να ορίζονται με γενικό/αφηρημένο τρόπο, ως **void***. Αυτό επιτρέπει στα προγράμματα που χρησιμοποιούν την βιβλιοθήκη να φτιάχνουν γράφους έχοντας ως κόμβους αντικείμενα οποιουδήποτε τύπου (με κατάλληλο type casting).

Ο έλεγχος για το αν δύο κόμβοι είναι ίδιοι/ταυτόσημοι πρέπει να πραγματοποιείται με γενικό/αφηρημένο τρόπο, μέσω μιας συνάρτησης η οποία παίρνει ως παραμέτρους δύο κόμβους (void *) και επιστρέφει την ακέραια τιμή 1 εάν αυτοί θεωρούνται ίσοι/ταυτόσημοι, διαφορετικά 0. Η υλοποίησή της είναι ευθύνη του εκάστοτε προγράμματος που χρησιμοποιεί την βιβλιοθήκη, ενώ η “εγκατάσταση” αυτής της συνάρτησης “μέσα” στην βιβλιοθήκη γίνεται κατά την δημιουργία/αρχικοποίηση του γράφου.

Αντίστοιχα, η καταστροφή ενός κόμβου πραγματοποιείται μέσω μιας συνάρτησης η οποία παίρνει ως παράμετρο τον κόμβο (void *) και ελευθερώνει τη μνήμη που έχει δεσμευτεί

δυναμικά γι αυτόν. Η υλοποίησή της είναι ευθύνη του προγράμματος που χρησιμοποιεί τη βιβλιοθήκη και η εγκατάστασή της γίνεται κατά τη δημιουργία/αρχικοποίηση του γράφου.

Λειτουργίες διαχείρισης γράφου

Πρέπει να παρέχονται οι εξής λειτουργίες διαχείρισης γράφου:

graph_t graph_new (int (*cmp)(void*, void*), void (*destroy_vertex(void*))): Δημιουργία και επιστροφή ενός νέου (κενού) γράφου, με εγκατάσταση της συνάρτησης ελέγχου ταυτότητας κόμβων και της συνάρτησης καταστροφής κόμβου (βλέπε παραπάνω).

int graph_add_vertex (graph_t g, void *vertex): Προσθήκη κόμβου στον γράφο. Αν ο κόμβος (ένας ταυτόσημος κόμβος) υπάρχει ήδη στον γράφο, επιστρέφεται κωδικός λάθους -1. Σε επιτυχία επιστρέφεται 0.

int graph_rmv_vertex (graph_t g, void *vertex): Αφαίρεση κόμβου από τον γράφο, με αυτόματη αφαίρεση όλων των ακμών που συμπεριλαμβάνουν τον κόμβο. Αν ο κόμβος δεν υπάρχει στον γράφο, επιστρέφεται κωδικός λάθους -1. Σε επιτυχία επιστρέφεται 0.

int graph_add_edge (graph_t g, void *vertex1, void *vertex2, int weight): Προσθήκη ακμής ανάμεσα στον κόμβο vertex1 και vertex2. Η παράμετρος weight προσδιορίζει το βάρος της ακμής. Σε επιτυχία επιστρέφεται 0. Αν έστω κι ένας κόμβος δεν υπάρχει στο γράφο επιστρέφεται κωδικός λάθους -1. Αν η συγκεκριμένη ακμή υπάρχει ήδη, επιστρέφεται κωδικός λάθους -2.

int graph_rmv_edge (graph_t g, void *vertex1, void *vertex2): Αφαίρεση ακμής ανάμεσα σε δύο κόμβους του γράφου. Σε επιτυχία επιστρέφεται 0. Αν έστω κι ένας κόμβος δεν υπάρχει στο γράφο επιστρέφεται κωδικός λάθους -1. Αν η συγκεκριμένη ακμή δεν υπάρχει, επιστρέφεται κωδικός λάθους -2.

void graph_destroy (graph_t g): Καταστροφή γράφου, με απελευθέρωση της μνήμης που χρησιμοποιείται για όλες τις εσωτερικές δομές.

Λειτουργίες διαχείρισης μονοπατιού

Πρέπει να παρέχονται οι εξής λειτουργίες διαχείρισης μονοπατιού:

int path_len (path_t p): Επιστροφή του αριθμού των ακμών του μονοπατιού.

void path_traverse_init (path_t p): Αρχικοποίηση διάσχισης μονοπατιού.

void *path_traverse_nxt (path_t p): Επιστροφή του επόμενου κόμβου στο μονοπάτι. Αν δεν υπάρχει επόμενος κόμβος, επιστρέφεται NULL.

void path_destroy (path_t p): Καταστροφή μονοπατιού, με απελευθέρωση της μνήμης που χρησιμοποιείται για όλες τις εσωτερικές δομές.

Λειτουργίες αναζήτησης στον γράφο

Πρέπει να παρέχονται οι εξής λειτουργίες αναζήτησης στον γράφο:

path_t find_minhops_path (graph_t g, void *vertex1, void *vertex2): Επιστροφή του μονοπατιού με τον μικρότερο αριθμό ακμών ανάμεσα σε δύο κόμβους. Αν δεν βρεθεί μονοπάτι, επιστρέφεται ένα κενό μονοπάτι (μήκους 0).

path_t find_shortest_path (graph_t g, void *vertex1, void *vertex2): Επιστροφή του μονοπατιού με το μικρότερο συνολικό βάρος ακμών ανάμεσα σε δύο κόμβους. Αν δεν βρεθεί μονοπάτι, επιστρέφεται ένα κενό μονοπάτι (μήκους 0).

Για την υλοποίηση των παραπάνω λειτουργιών, μπορείτε να βασιστείτε στον “κλασικό” αλγόριθμο του Dijkstra[1] (χρησιμοποιήστε όποιες παραλλαγές θεωρείτε κατάλληλες για κάθε περίπτωση).

Πρόγραμμα δοκιμής -- social network model

Δοκιμάστε την βιβλιοθήκη σας μέσω ενός προγράμματος που επιτρέπει στον χρήστη να ορίζει έναν γράφο κοινωνικής δικτύωσης και να πραγματοποιεί αναζητήσεις σε αυτόν, χρησιμοποιώντας την παραπάνω βιβλιοθήκη.

Οι κόμβοι που δημιουργεί το πρόγραμμα αντιστοιχούν σε ανθρώπους (για απλότητα περιέχουν μόνο το όνομα, το οποίο δεν περιέχει κενά), οι ακμές συμβολίζουν σχέσεις φιλίας, και το βάρος μιας ακμής συμβολίζει την περιοδικότητα της επικοινωνίας ανάμεσα σε δύο φίλους (μεγαλύτερες τιμές αντιστοιχούν σε πιο σπάνια επικοινωνία). Οπότε το μονοπάτι με τον μικρότερο αριθμό ακμών ανάμεσα σε δύο κόμβους του γράφου αντιστοιχεί στην πιο άμεση σύνδεση δύο ατόμων μέσω των γνωριμιών τους, ενώ το μονοπάτι με το μικρότερο συνολικό βάρος ακμών ανάμεσα σε δύο κόμβους του γράφου αντιστοιχεί στην αλυσίδα που θα οδηγήσει με μεγαλύτερη πιθανότητα σε μια συνάντηση δύο ατόμων. Το πρόγραμμα πρέπει να υλοποιεί την συνάρτηση ελέγχου ταυτοποίησης ανάμεσα σε δύο κόμβους (έλεγχος αν το όνομα είναι ίδιο), και να την περνά ως παράμετρο κατά την δημιουργία του γράφου.

Συγκεκριμένα πρέπει να υποστηρίζονται οι εξής εντολές, με την μορφή που δίνεται παρακάτω:

Εντολή	**Είσοδος**	**Έξοδος stdout (επιτυχία)**	**Έξοδος stdout (αποτυχία)**
create	c	C-OK	C-NOK
destroy	d	D-OK	-
add person	ap <name>	AP-OK	AP-NOK
rmv person	rp <name>	RP-OK	RP-NOK

[1] https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

add friendship	af <name> <name> <com. period>	AF-OK	AF-NOK P ή AF-NOK F
rmv friendship	rf <name> <name>	RF-OK	RF-NOK P ή RF-NOK F
find closest connection	fc <name> <name>	## FC-OK <len> <name>…<name>	FC-NOK
find likely connection	fl <name> <name>	## FL-OK <len> <name>…<name>	FL-NOK
print social network	p	<name> <name> <com. period> … <name> <name> <com. period>	
quit	q		

Αν μια εντολή αποτελείται από περισσότερα τμήματα (συνοδεύεται από μια ή περισσότερες παραμέτρους), αυτά διαχωρίζονται με έναν ή περισσότερους χαρακτήρες ‘ ‘ (space). Κάθε εντολή τερματίζει με τον χαρακτήρα ‘\n’.

Πριν και μετά από κάθε μήνυμα εξόδου βρίσκεται χαρακτήρας ‘\n’. Αν το μήνυμα αποτελείται από πολλαπλά τμήματα ανά γραμμή, αυτά διαχωρίζονται από έναν μοναδικό χαρακτήρα ‘ ‘ (space).

Στην έξοδο αποτυχίας των add/rmv friendship, P σημαίνει ότι αυτά τα άτομα δεν υπάρχουν κι F ότι είναι/δεν είναι ήδη φίλοι.

Σημειώσεις:

- Αν έχει ήδη δημιουργηθεί ένας γράφος στην εφαρμογή, δε θα επιτρέπεται η δημιουργία άλλου πριν καταστραφεί ο προηγούμενος (εκτυπώνεται C-NOK).
- Το len στην περίπτωση fc αναφέρεται σε πλήθος ακμών του μονοπατιού ενώ στην περίπτωση fl σε συνολικό άθροισμα βαρών των ακμών του μονοπατιού.
- Σε περίπτωση αποτυχίας της λειτουργίας af, υπεύθυνος για την απελευθέρωση της μνήμης που δεσμεύτηκε για τα δεδομένα του κόμβου είναι ο χρήστης.

Προσοχή: Για όλες τις εντολές, εκτός της p(rint), το πρόγραμμα πρέπει να ακολουθεί ακριβώς τις παραπάνω προδιαγραφές, διαφορετικά θα αποτύχει ο αυτοματοποιημένος έλεγχος της λειτουργίας το προγράμματος, όπου η είσοδος και η έξοδος του προγράμματος ανακατευθύνεται σε αρχεία, όπου τα αρχεία εισόδου περιέχουν τις εντολές ακριβώς με βάση την παραπάνω μορφή, ενώ τα αρχεία εξόδου συγκρίνονται, byte προς byte, με άλλα αρχεία που έχουν ακριβώς την αναμενόμενη έξοδο. Η λειτουργία p(rint) δεν θα χρησιμοποιηθεί κατά την

διαδικασία του αυτοματοποιημένου ελέγχου, όμως πρέπει να υλοποιηθεί αφενός για να μπορεί να γίνονται πιο εύκολα έλεγχοι κατά την εξέταση/επίδειξη της εργασίας αφετέρου για να διευκολύνει εσάς τους ίδιους στο debugging κατά την ανάπτυξη.

Εναλλακτική υλοποίηση της print: Μπορείτε να υλοποιήσετε την print ώστε να παράγει έξοδο σε μορφή που μπορεί να μετατραπεί σε γραφική αναπαράσταση. Συστήνουμε η έξοδος να σταλεί στο stderr ώστε να είναι εύκολο να αποθηκευτεί σε αρχείο με κατάλληλη ανακατεύθυνση.

Η μορφή της εξόδου είναι:

- Η συμβολοσειρά “graph G {“
- Για κάθε κόμβο, μια γραμμή με το όνομα του κόμβου
- Για κάθε ακμή από κόμβο X σε κόμβο Y με βάρος Z, μια γραμμή με το string “X -- Y [label=Z]”
- Στην τελευταία γραμμή ο χαρακτήρας ‘}’ και αλλαγή γραμμής.

Παράδειγμα:

Εκτελείτε το πρόγραμμά σας και ανακατευθύνετε το stderr στο αρχείο mygraph.dot ως εξής:

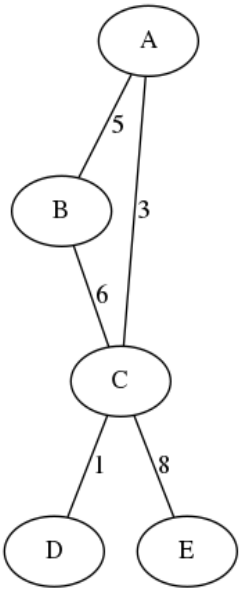
./project3 2> mygraph.dot

Ας υποθέσουμε ότι τα περιεχόμενα του mygraph.dot είναι:

```
graph G {
A
B
C
D
E
A -- B [label=5]
A -- C [label=3]
B -- C [label=6]
C -- D [label=1]
C -- E [label=8]
}
```

Γράφετε σε τερματικό την εντολή dot -Tpng < mygraph.dot > mygraph.png

Αν ανοίξετε το mygraph.png θα δείτε:



Εγκατάσταση του προγράμματος dot στον υπολογιστή σας

Το πρόγραμμα dot βρίσκεται μέσα στη σουίτα εργαλείων graphviz για εκτύπωση γράφων. Αναζητήστε το πακέτο **graphviz** στη διανομή Linux του υπολογιστή σας.

Για Ubuntu: `$> sudo apt-get install graphviz`

Περιεχόμενα αρχείου graph.h

```
#ifndef _GRAPH_H
#define _GRAPH_H

typedef struct private_graph * graph_t;
typedef struct private_path * path_t;

graph_t graph_new(int (*cmp)(void*, void*), void (*destroy_vertex(void*));
void graph_destroy(graph_t g);

int graph_add_vertex(graph_t graph, void *vertex);
int graph_rmv_vertex(graph_t graph, void *vertex);

int graph_add_edge(graph_t graph, void *vertex1, void *vertex2, int weight) ;
int graph_rmv_edge(graph_t graph, void *vertex1, void *vertex2) ;

int path_len(path_t path);
void path_traverse_init(path_t path);
void *path_traverse_nxt(path_t path);
void path_destroy(path_t path);
```

```
path_t find_minhops_path (graph_t graph, void *vertex1, void *vertex2);
path_t find_shortest_path (graph_t graph, void *vertex1, void *vertex2);

#endif
```

Παρατηρήστε πως οι τύποι graph_t, path_t είναι δείκτες σε struct. Οι εσωτερικές υλοποιήσεις των struct private_graph, private_path θα βρίσκονται στο graph.c ώστε να είναι κρυμμένες από τον χρήστη της βιβλιοθήκης. Ο συγκεκριμένος τρόπος σχεδιασμού αφηρημένων τύπων δεδομένων καλύφθηκε στη διάλεξη "Βιβλιοθήκες".

Στάδια ανάπτυξης του κώδικα

Στάδιο 0:

Αποφασίστε πώς ακριβώς θα υλοποιήσετε το γράφο σας (adjacency list/matrix) και τι πληροφορίες θα αποθηκεύσετε στις εκάστοτε δομές.

Στάδιο 1:

Υλοποιήστε τις λειτουργίες διαχείρισης γράφου (στα πλαίσια της βιβλιοθήκης). Γράψτε τη main με όλες τις λειτουργίες εκτός των d, fc, fl. Χρησιμοποιήστε τη για να κατασκευάσετε ένα γράφο και ελέγξτε συστηματικά κάθε μία λειτουργία.

Στάδιο 2:

Υλοποιήστε τις λειτουργίες του μονοπατιού. Γράψτε μια προσωρινή βοηθητική συνάρτηση που απλά κατασκευάζει ένα μονοπάτι και χρησιμοποιεί τις λειτουργίες που υλοποιήσατε για να εκτυπώσει τα περιεχόμενά του, και καλέστε τη από τη main.

Στάδιο 3:

Υλοποιήστε τον αλγόριθμο του Dijkstra και τις λειτουργίες fc, fl. Ελέγξτε την ορθότητά τους.

Στάδιο 4:

Υλοποιήστε τη λειτουργία d.