

Εργασία 5η - Γράφος - Ατομική Εργασία

Ο Γράφος	1
Το κυρίως πρόγραμμα	4
Έλεγχος του προγράμματος σας	5
Makefile και αρχεία ελέγχου	5
Τρόπος Αποστολής	5
Παράρτημα 1 - Ενδεικτικά γραφήματα	6

Ατομική Εργασία, Καταληκτική ημερομηνία υποβολής: Κυριακή 24 Δευτέρα 29 Ιουνίου

Σε αυτή την εργασία θα υλοποιήσετε την παραμετρική (*templated*) κλάση ενός γράφου. Ο γράφος μπορεί να είναι κατευθυνόμενος ή μη. Για τα μη κατευθυνόμενα γραφήματα μπορείτε να υποθέσετε ότι αυτά θα είναι πάντα συνεκτικά. Η παραπάνω υπόθεση δεν ισχύει για τα κατευθυνόμενα γραφήματα.

Ο Γράφος

Το header file της κλάσης δίνεται παρακάτω:

```
#ifndef _GRAPH_HPP_
#define _GRAPH_HPP_
#include <list>

template<typename T>
struct Edge {
    T from;
    T to;
    int dist;
    Edge(T f, T t, int d): from(f), to(t), dist(d);
    bool operator<(const Edge<T>& e) const;
    bool operator>(const Edge<T>& e) const;

    template<typename U>
    friend std::ostream& operator<<(std::ostream& out, const Edge<U>& e);
};

template<typename T>
std::ostream& operator<<(std::ostream& out, const Edge<T>& e) {
    out << e.from << " -- " << e.to << " (" << e.dist << ")";
    return out;
}

template <typename T>
class Graph {

public:
    Graph(bool isDirected = true);
    bool addVtx(const T& info);
```

```

bool rmvVtx(const T& info);
bool addEdg(const T& from, const T& to, int distance);
bool rmvEdg(const T& from, const T& to);
std::list<T> dfs(const T& info) const;
std::list<T> bfs(const T& info) const;
std::list<Edge<T>> mst();

bool print2DotFile(const char *filename) const;
std::list<T> dijkstra(const T& from, const T& to);
std::list<T> bellman_ford(const T& from, const T& to);
};

```

Το γράφημα μπορεί να υλοποιηθεί μέσω πίνακα γειτνιάσεως ή λίστας γειτνιάσεως. Εφόσον υλοποιήσετε το γράφημα μέσω λίστας γειτνιάσεως, συνιστάται η λίστα γειτνιάσεως να υλοποιείται μέσω ενός πίνακα μιας διάστασης που περιέχει τις επιμέρους λίστες. Κάθε θέση του πίνακα της λίστας αντιστοιχεί σε ένα κόμβο. Σε οποιονδήποτε γράφημα (είτε υλοποιείται μέσω λίστας είτε μέσω πίνακα) δεν είναι απαραίτητο όλες οι θέσεις του πίνακα να είναι κατειλημμένες. Οι συναρτήσεις της κλάσης και ο κατασκευαστής δίνονται παρακάτω:

Μέθοδος	Περιγραφή
<code>Graph(bool isDirected)</code>	Η παράμετρος προσδιορίζει εάν το γράφημα είναι κατευθυνόμενο ή όχι..
<code>bool contains(const T& info);</code>	Επιστρέφει true εάν ο γράφος περιέχει το συγκεκριμένο κόμβο, διαφορετικά false .
<code>bool addVtx(const T& info);</code>	Ενθέτει τον κόμβο στον γράφο, εφόσον ο συγκεκριμένος κόμβος δεν υπάρχει ήδη σε αυτόν. Επιστρέφει true σε περίπτωση επιτυχίας, διαφορετικά false . Εάν πριν την ένθεση ο πίνακας γειτνιάσεως ή ο πίνακας της λίστας γειτνιάσεως δεν έχει διαθέσιμες θέσεις προς πλήρωση αυξάνεται η χωρητικότητα του πίνακα.
<code>bool rmvVtx(const T& info);</code>	Διαγράφει τον κόμβο από τον γράφο, εφόσον ο συγκεκριμένος κόμβος υπάρχει ήδη σε αυτόν. Επιστρέφει true σε περίπτωση επιτυχίας, διαφορετικά false .
<code>bool addEdg(const T& from, const T& to, int cost);</code>	Ενθέτει την ακμή με αφετηρία τον κόμβο <i>from</i> και προορισμό τον κόμβο <i>to</i> και κόστος <i>cost</i> . Απαραίτητες προϋποθέσεις για την επιτυχή ένθεση της ακμής είναι τα εξής: <ul style="list-style-type: none"> - οι κόμβοι <i>from</i> και <i>to</i> θα πρέπει να υπάρχουν στο γράφημα.. - η ακμή με αφετηρία τον κόμβο <i>from</i> και προορισμό τον κόμβο <i>to</i> δεν πρέπει να υπάρχει στο γράφημα. Σημείωση: Σε ένα μη κατευθυνόμενο γράφημα η σειρά δήλωσης των κόμβων <i>from</i> και <i>to</i> από τους οποίους προσδιορίζεται η ακμή δεν έχει σημασία.
<code>bool rmvEdg(const T& from, const T& to);</code>	Διαγράφει την ακμή με αφετηρία τον κόμβο <i>from</i> και προορισμό τον κόμβο <i>to</i> και κόστος <i>cost</i> . Για την επιτυχή διαγραφή της ακμής, αυτή θα πρέπει να υπάρχει στο γράφημα. Σημείωση: Σε ένα μη κατευθυνόμενο γράφημα η σειρά δήλωσης των κόμβων <i>from</i> και <i>to</i> από τους οποίους προσδιορίζεται η ακμή δεν έχει σημασία.

<pre>std::list<T> dfs(const T& info) const;</pre>	<p>Επιστρέφει μία λίστα των κόμβων του γραφήματος, διατρέχοντας το γράφημα κατά βάθος (depth first search traversal), με αφετηρία τον κόμβο info. Ο αλγόριθμος εφαρμόζεται σε κατευθυνόμενα και μη κατευθυνόμενα γραφήματα.</p> <p>Η σειρά επίσκεψης των γειτονικών κόμβων ενός κόμβου είναι με βάση τη σειρά εισαγωγής τους στο γράφημα (αυτός που εισήχθη πρώτος, εκτυπώνεται πρώτος, αυτός που εισήχθη δεύτερος εκτυπώνεται δεύτερος κ.ο.κ.).</p>
<pre>std::list<T> bfs(const T& info) const;</pre>	<p>Επιστρέφει μία λίστα των κόμβων του γραφήματος, διατρέχοντας το γράφημα κατά πλάτος (breadth first search traversal), με αφετηρία τον κόμβο info. Ο αλγόριθμος εφαρμόζεται σε κατευθυνόμενα και μη κατευθυνόμενα γραφήματα.</p> <p>Η σειρά επίσκεψης των γειτονικών κόμβων ενός κόμβου είναι με βάση τη σειρά εισαγωγής τους στο γράφημα (αυτός που εισήχθη πρώτος, εκτυπώνεται πρώτος, αυτός που εισήχθη δεύτερος εκτυπώνεται δεύτερος κ.ο.κ.).</p>
<pre>std::list<Edge<T>> mst();</pre>	<p>Υλοποιεί το ελάχιστο επικαλυπτόμενο δέντρο για ένα <u>μη κατευθυνόμενο και συνεκτικό</u> γράφημα. Εάν το γράφημα είναι <u>κατευθυνόμενο</u> επιστρέφει μία άδεια λίστ.. Η λίστα των ακμών θα πρέπει να έχει τα εξής χαρακτηριστικά:</p> <ul style="list-style-type: none"> - Η σειρά των κόμβων κάθε ακμής είναι <u>πρώτα ο κόμβος που προστέθηκε πρώτος στο γράφημα και μετά ο έτερος κόμβος</u>. Ακολουθεί το κόστος της ακμής. - Η σειρά των ακμών είναι από την ακμή χαμηλότερου κόστους προς την ακμή υψηλότερου κόστους. Μεταξύ δύο ακμών ίδιου κόστους δεν υπάρχει συγκεκριμένη προτεραιότητα.
<pre>list<T> dijkstra(const T& from, const T& to);</pre>	<p>Εφαρμόζεται ο αλγόριθμος dijkstra για τη εύρεση του συντομότερου μονοπατιού με αφετηρία τον κόμβο from και προορισμό τον κόμβο to. Επιστρέφεται μία λίστα με τους κόμβους του μονοπατιού από from έως και to. Εάν δεν υπάρχει μονοπάτι μεταξύ from και to επιστρέφεται μία άδεια λίστα.</p>
<pre>list<T> bellman_ford(const T& from, const T& to);</pre>	<p>Εφαρμόζεται ο αλγόριθμος bellman-ford για τη εύρεση του συντομότερου μονοπατιού με αφετηρία τον κόμβο from και προορισμό τον κόμβο to. Επιστρέφεται μία λίστα με τους κόμβους του μονοπατιού από from έως και to. Εάν δεν υπάρχει μονοπάτι μεταξύ from και to επιστρέφεται μία άδεια λίστα.</p> <p>Επιπλέον, ο αλγόριθμος θα πρέπει να μπορεί να ανιχνεύσει εάν υπάρχει βρόγχος αρνητικού βάρους εντός του γράφου. Σε αυτή την περίπτωση παράγει μία εξαίρεση του τύπου NegativeGraphCycle που είναι απόγονος της κλάσης std::exception. Η μέθοδος what() της παραπάνω εξαίρεσης επιστρέφει το αλφαριθμητικό "Negative Graph Cycle!".</p>
<pre>bool print2DotFile(const char *filename) const;</pre>	<p>Εκτυπώνει το γράφημα σε ένα αρχείο κειμένου κατάλληλο για ανάγνωση και μετασχηματισμό σε εικόνα από το πρόγραμμα graphviz/dot. Επιστρέφει true εάν η εγγραφή ήταν επιτυχής διαφορετικά false. Εάν το αρχείο έχει περιεχόμενο κατά το άνοιγμα, το υφιστάμενο περιεχόμενο διαγράφεται.</p> <p>Η συνάρτηση δεν θα ελεγχθεί από autolab και συνιστάται η χρήση της μόνο για δική σας αποσφαλμάτωση του κώδικα και την επιβεβαίωση της ορθής κατασκευής του γράφου.</p>

Το κυρίως πρόγραμμα

Σας δίνεται ο σκελετός της παραμετρικής συνάρτησης **graphUI** η οποία αποτελεί της υλοποίηση του κυρίως προγράμματος. Η συνάρτηση διαβάζει πάντα από το **stdin** και εκτυπώνει στο **stdout**. Το πρόγραμμα υποθέτει ότι κάθε εντολή καταλαμβάνει πάντα μία ακριβώς γραμμή. Κάθε γραμμή που διαβάζεται από το **stdin** αποθηκεύεται σε ένα **std::stringstream**. Στη συνέχεια το **std::stringstream** χρησιμοποιείται για να κατασκευάσει τα αντικείμενα που αντιπροσωπεύουν το περιεχόμενο των κόμβων του γράφου. Η κλάση των αντικειμένων αυτών αποτελεί την παράμετρο της συνάρτησης και την παράμετρο της κλάσης του γράφου.

Κάθε κλάση που δημιουργεί αντικείμενα τα οποία αποτελούν την πληροφορία των κόμβων του γράφου, θα πρέπει να έχει ένα κατασκευαστή που διαβάζει από **std::istream**, ώστε να μπορεί να κατασκευάσει αντικείμενα διαβάζοντας από το παραπάνω **std::stringstream**.

Το κυρίως πρόγραμμα έχει τις εξής εντολές:

Εντολή	Περιγραφή	Εκτύπωση επιτυχίας	Εκτύπωση αποτυχίας
av node	Εντολή εισαγωγής ενός νέου κόμβου με πληροφορία node στο γράφημα. Καλεί τη συνάρτηση addVtx .	av node OK	av node NOK
rv node	Εντολή διαγραφής ενός υφιστάμενου κόμβου με πληροφορία node από το γράφημα. Καλεί τη συνάρτηση rmvVtx .	rv node OK	rv node NOK
ae from to	Εντολή εισαγωγής μιας ακμής στο γράφημα μεταξύ των κόμβων from και to . Καλεί τη συνάρτηση addEdg .	ae from to OK	ae from to NOK
re from to	Εντολή διαγραφής της υφιστάμενης ακμής μεταξύ των κόμβων from και to από το γράφημα. Καλεί τη συνάρτηση rmvEdg .	re from to OK	re from to NOK
dot file	Εντολή εκτύπωσης του γραφήματος σε μορφή dot στο αρχείο με pathname file .	dot file OK	dot file NOK
dfs node	Με αφετηρία τον κόμβο node εκτυπώνει στο stdout η κατά βάθος διαπέραση του γραφήματος. Οι κόμβοι εκτυπώνονται με τη σειρά που επιστρέφονται από τη συνάρτηση dfs . <ul style="list-style-type: none"> - Αρχικά εκτυπώνεται το αλφαριθμητικό "\n----- DFS Traversal -----\n". - Στη συνέχεια εκτυπώνεται η διαπέραση του γράφου με αφετηρία τον κόμβο node. Κάθε κόμβος του γράφου διαχωρίζεται από τον προηγούμενο με το αλφαριθμητικό " -> ". - Στο τέλος εκτυπώνεται το αλφαριθμητικό "\n-----\n". 		
bfs node	Με αφετηρία τον κόμβο node εκτυπώνει στο stdout η κατά πλάτος διαπέραση του γραφήματος. Οι κόμβοι εκτυπώνονται με τη σειρά που επιστρέφονται από τη συνάρτηση bfs . <ul style="list-style-type: none"> - Αρχικά εκτυπώνεται το αλφαριθμητικό "\n----- BFS Traversal -----\n". - Στη συνέχεια εκτυπώνεται η διαπέραση του γράφου με αφετηρία τον κόμβο node. Κάθε κόμβος του γράφου διαχωρίζεται από τον προηγούμενο με το αλφαριθμητικό " -> ". - Στο τέλος εκτυπώνεται το αλφαριθμητικό "\n-----\n". 		
dijkstra from to	Εκτυπώνεται το αλφαριθμητικό "Dijkstra: " ακολουθούμενο από το μονοπάτι των κόμβων από τον κόμβο from έως τον κόμβο to . Κάθε κόμβος διαχωρίζεται από τον προηγούμενο με το αλφαριθμητικό ", " (κόμμα και κενό).		
bellman-ford from to	Εκτυπώνεται το αλφαριθμητικό "Bellman-Ford: " . Εάν δεν υπάρχει βρόγχος αρνητικού βάρους μέσα στον γράφο εκτυπώνεται το μονοπάτι των κόμβων από τον κόμβο from έως τον κόμβο to . Κάθε κόμβος διαχωρίζεται από τον προηγούμενο με το αλφαριθμητικό ", " (κόμμα και κενό). Στην περίπτωση που υπάρχει βρόγχος αρνητικού βάρους, θα πρέπει να εκτυπώνεται το μήνυμα "Negative Graph Cycle!" ακολουθούμενο από χαρακτήρα αλλαγής γραμμής.		
mst	Εκτυπώνεται το αλφαριθμητικό "\n--- Min Spanning Tree ---\n" . Στη συνέχεια εκτυπώνονται οι ακμές του ελάχιστου επικαλυπτόμενου δέντρου, με τη σειρά που επιστρέφονται από τη συνάρτηση mst . Τέλος εκτυπώνεται το συνολικό κόστος διαπέρασης του δέντρου που αποτελεί το άθροισμα του κόστους		

	των επιμέρους ακμών. Στο τέλος εκτυπώνεται το αλφαριθμητικό " MST Cost: x " ακολουθούμενο από χαρακτήρα αλλαγής γραμμής όπου x το συνολικό κόστος του δέντρου.
#	Μία εντολή που ξεκινά με τον χαρακτήρα '#' και ακολουθεί κενός χαρακτήρας δεν λαμβάνεται υπόψη από το πρόγραμμα.
α	Μία εντολή που ξεκινά με τον χαρακτήρα 'α' και ακολουθεί κενός χαρακτήρας σηματοδοτεί τον τερματισμό του προγράμματος.

Έλεγχος του προγράμματος σας

Για τον έλεγχο του προγράμματος σας παρέχονται τα παρακάτω tests:

`bfs1, bfs2, bfs3, dfs1, dfs2, dfs3, mst1, mst2, mst3, dijkstra1, dijkstra2, dijkstra3, dijkstra4, bellman-ford1, bellman-ford2, bellman-ford3.`

[Τις εικόνες από τα γραφήματα που αντιστοιχούν στα αρχεία ελέγχου μπορείτε να τις κατεβάσετε εδώ.](#) (είναι πλέον προσβάσιμος ο σύνδεσμος)

Makefile και αρχεία ελέγχου

Στα αρχεία του e-class σας παρέχεται ο αρχείο `hw5_skeleton.tar.gz` το οποίο διαθέτει τους σκελετούς των αρχείων που σας δίνονται έτοιμα, κατάλληλο Makefile και τα αρχεία ελέγχου (βρίσκονται στον υποκατάλογο **tests**). Για να εκτελέσετε όλα τα test γράφετε στο terminal `$> make run`, ενώ για να εκτελέσετε ένα μεμονωμένο test (π.χ. `dijkstra1`) γράφετε `$> make dijkstra1`.

Μέσα στο αρχείο `hw5_skeleton.tar.gz` υπάρχει και το αρχείο `UnionFind.hpp`, το οποίο παρέχεται κενό. Σε περίπτωση που θέλετε να υλοποιήσετε ένωση-εύρεση σε ξένα σύνολα αποθηκεύστε τον κώδικα σας στο αρχείο αυτό. Εάν δεν υλοποιήσετε ένωση-εύρεση αφήστε το κενό, αλλά υποβάλλετε το μαζί με τα άλλα δύο αρχεία (δείτε σχετικά στην ενότητα Τρόπος Αποστολής)

Τρόπος Αποστολής

Η εργασία θα εξεταστεί μέσω του εργαλείου αυτόματης υποβολής και διόρθωσης autolab. Οι οδηγίες αποστολής είναι οι εξής:

1. Πηγαίνετε στη σελίδα του [autolab](#) και συνδεθείτε με το username και το password σας (απαιτείται σύνδεση VPN).
2. Επιλέξτε το μάθημα **CE325_2019-2020 (S20)** και στη συνέχεια την εργασία **HW5**.
3. Κατασκευάστε το φάκελο **hw5submit** τοπικά στον υπολογιστή σας.
4. Μέσα στον φάκελο αντιγράψτε τα αρχεία **Graph.hpp**, **GraphUI.hpp** της εργασίας σας και το αρχείο **UnionFind.hpp**. Το αρχείο **UnionFind.hpp** σας δίνεται κενό. Εάν θέλετε να υλοποιήσετε μία κλάση ένωσης-εύρεσης σε ξένα σύνολα μπορείτε να το κάνετε στο αρχείο αυτό. Σε κάθε περίπτωση το αρχείο πρέπει να υποβληθεί (ακόμη και κενό περιεχομένου).
5. Συμπιέστε και πακετάρετε τον κατάλογο **hw5submit** σε μορφή **.tar.gz**, κάνοντας δεξί click στον κατάλογο **hw5submit** και στη συνέχεια επιλέγοντας από το pop-up menu την επιλογή **Compress here as tar.gz**.

Παράρτημα 1 - Ενδεικτικά γραφήματα

Τα παρακάτω γραφήματα είναι ενδεικτικά για τις δοκιμές σας. Σε καμία περίπτωση δεν αποτελούν τα μοναδικά γραφήματα τα οποία θα εξεταστούν στα test cases.



