Εργασία 5η - Γράφος

Ατομική Εργασία, Καταληκτική ημερομηνία υποβολής: Σάββατο 9 Ιουλίου

Σημείωση: Μην αντιγράφετε αλφαριθμητικά με copy-paste από την εκφώνηση. Μπορεί να εισαχθούν στον κώδικα σας χαρακτήρες που θα δημιουργήσουν πρόβλημα στην μεταγλώττιση ή στην συμβατότητα της εξόδου σας με την έξοδο του autolab.

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;   // σύγκριση ακμών με βάση το κόστος τους
  bool operator>(const Edge<T>& e) const;   // σύγκριση ακμών με βάση το κόστος τους

  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);
  ~Graph();
  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);
};
```

Το γράφημα μπορεί να υλοποιηθεί αποκλειστικά μέσω λίστας γειτνιάσεως. Οι συναρτήσεις της κλάσης και ο κατασκευαστής δίνονται παρακάτω:

Μέθοδος	Περιγραφή
`Graph(bool isDirected)`	Η παράμετρος προσδιορίζει εάν το γράφημα είναι κατευθυνόμενο ή όχι.
`~Graph()`	Καταστροφέας της κλάσης
`bool contains(const T& info);`	Επιστρέφει **true** εάν ο γράφος περιέχει το συγκεκριμένο κόμβο, διαφορετικά **false**.
`bool addVtx(const T& info);`	Ενθέτει τον κόμβο στον γράφο, εφόσον ο συγκεκριμένος κόμβος δεν υπάρχει ήδη σε αυτόν. Επιστρέφει ***true*** σε περίπτωση επιτυχίας, διαφορετικά ***false***.
`bool rmvVtx(const T& info);`	Διαγράφει τον κόμβο από τον γράφο, εφόσον ο συγκεκριμένος κόμβος υπάρχει ήδη σε αυτόν. Επιστρέφει ***true*** σε περίπτωση επιτυχίας, διαφορετικά ***false***.
`bool addEdg(const T& from, const T& to, int cost);`	Ενθέτει την ακμή με αφετηρία τον κόμβο *from* και προορισμό τον κόμβο *to* και ακέραιο κόστος *cost*. Απαραίτητες προϋποθέσεις για την επιτυχή ένθεση της ακμής είναι τα εξής: - οι κόμβοι *from* και το θα πρέπει να υπάρχουν στο γράφημα. - η ακμή με αφετηρία τον κόμβο *from* και προορισμό τον κόμβο *to* δεν πρέπει να υπάρχει στο γράφημα. **Σημείωση:** Σε ένα μη κατευθυνόμενο γράφημα η σειρά δήλωσης των κόμβων *from* και το από τους οποίους προσδιορίζεται η ακμή δεν έχει σημασία.
`bool rmvEdg(const T& from, const T& to);`	Διαγράφει την ακμή με αφετηρία τον κόμβο *from* και προορισμό τον κόμβο *to*. Για την επιτυχή διαγραφή της ακμής, αυτή θα πρέπει να υπάρχει στο γράφημα. **Σημείωση:** Σε ένα μη κατευθυνόμενο γράφημα η σειρά δήλωσης των κόμβων *from* και το από τους οποίους προσδιορίζεται η ακμή δεν έχει σημασία.
	Επιστρέφει μία λίστα των κόμβων του γραφήματος, διατρέχοντας το γράφημα

`std::list<T> dfs(const T& info) const;`	κατά βάθος (***depth first search*** traversal), με αφετηρία τον κόμβο ***info***. Ο αλγόριθμος εφαρμόζεται σε κατευθυνόμενα και μη κατευθυνόμενα γραφήματα. 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).
`std::list<T> bfs(const T& info) const;`	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. 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).
`std::list<Edge<T>> mst();`	Υλοποιεί το ελάχιστο επικαλυπτόμενο δέντρο για ένα μη κατευθυνόμενο και συνεκτικό γράφημα. Εάν το γράφημα είναι κατευθυνόμενο επιστρέφει μία άδεια λίστα. Η λίστα των ακμών θα πρέπει να έχει τα εξής χαρακτηριστικά: - Η σειρά των κόμβων κάθε ακμής είναι πρώτα ο κόμβος που προστέθηκε πρώτος στο γράφημα και μετά ο έτερος κόμβος. Ακολουθεί το κόστος της ακμής. - Η σειρά των ακμών είναι από την ακμή χαμηλότερου κόστους προς την ακμή υψηλότερου κόστους. Μεταξύ δύο ακμών ιδίου κόστους δεν υπάρχει συγκεκριμένη προτεραιότητα.

`list<T> dijkstra(`
`const T& from,`
`const T& to);`
	Εφαρμόζεται ο αλγόριθμος **dijkstra** για την 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.

`bool print2DotFile(const char *filename) const;`	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. Η συνάρτηση δε θα ελεγχθεί από το autolab και συνιστάται η χρήση της μόνο για δική σας αποσφαλμάτωση του κώδικα και την επιβεβαίωση της ορθής κατασκευής του γράφου.

Το κυρίως πρόγραμμα

Σας δίνεται ο σκελετός της παραμετρικής συνάρτησης **graphUI** η οποία αποτελεί τμήμα της υλοποίηση του κυρίως προγράμματος και την οποία καλείστε να συμπληρώσετε κατάλληλα. Η συνάρτηση διαβάζει πάντα από το **stdin** και εκτυπώνει στο **stdout**. Το πρόγραμμα υποθέτει ότι κατά την ανάγνωση κάθε εντολή καταλαμβάνει μία ακριβώς γραμμή (διαβάζουμε επομένως γραμμή-γραμμή). Κάθε γραμμή που διαβάζεται από το **stdin** αποθηκεύεται σε ένα ***std::stringstream***. Στη συνέχεια το ***std::stringstream*** χρησιμοποιείται για να κατασκευαστούν τα αντικείμενα που αντιπροσωπεύουν τους κόμβους του γράφου. Η κλάση των αντικειμένων

αυτών αποτελεί την παράμετρο της *templated* συνάρτησης **graphUI** (δείτε το αρχείο GraphUI.hpp) και την παράμετρο της *templated* κλάσης **Graph** που αντιπροσωπεύει τον γράφο.

Η πρώτη γραμμής της εισόδου ξεκινά υποχρεωτικά με μία από τις λέξεις **`graph`** (μη κατευθυνόμενο γράφημα) ή **`digraph`** (κατευθυνόμενο γράφημα). Οι επόμενες εντολές του προγράμματος είναι μία από τις ακόλουθες:

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**. - 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**. - 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).		
`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.		
`#`	Μια εντολή που ξεκινά με τον χαρακτήρα **`'#'`** και ακολουθεί κενός χαρακτήρας δεν λαμβάνεται υπόψη από το πρόγραμμα.		
`q`	Μια εντολή που ξεκινά με τον χαρακτήρα **`'q'`** 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`.

Τις εικόνες από τα γραφήματα που αντιστοιχούν στα αρχεία ελέγχου μπορείτε να τις κατεβάσετε εδώ.

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 αποστολή της εργασίας θα γίνει μέσω της πλατφόρμας autolab. Για την υποβολή ακολουθήστε τα εξής βήματα:

Συμπιέστε σε μορφή zip, από το project σας τα περιεχόμενα του καταλόγου που περιέχει τον πηγαίο κώδικα της εργασίας σας, δηλαδή 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).

Το αρχείο που προκύπτει μετά τη συμπίεση, πρέπει να έχει όνομα **`hw5.zip`**.

Συνδέεστε στο autolab (μέσω VPN) και επιλέγετε το μάθημα **ECE326_2022 (S22)** και από αυτό την εργασία **HW5**.

Για να υποβάλετε την εργασία σας κάνετε click στην επιλογή “I affirm that I have compiled with this course academic integrity policy…” και πατάτε submit. Στη συνέχεια επιλέγετε το αρχείο hw1.zip που δημιουργήσατε

Παράρτημα - 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.