Εργασία 5η - Γράφος - Ατομική Εργασία

Ο Γράφος 1

Το κυρίως πρόγραμμα 4

Έλεγχος του προγράμματος σας 5

Τρόπος Αποστολής 5

Ατομική Εργασία, Καταληκτική ημερομηνία υποβολής: Κυριακή 21 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 {
  // ορίστε τα πεδία της κλάσης και private/protected συναρτήσεις
  void expand_table();
  void shrink_table();

public:
  Graph(bool isDirected = true, int capacity = 2);
  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
`void expand_table()`	Διπλασιάζει το μέγεθος του αρχικού πίνακα. Κάθε φορά που διπλασιάζεται το μέγεθος του πίνακα εκτυπώνεται στο **stdout** το μήνυμα "`Capacity: X -> Y`", όπου **X** η αρχική χωρητικότητα και **Y** η νέα χωρητικότητα του πίνακα.
`void shrink_table()`	Υπό-διπλασιάζει το μέγεθος του αρχικού πίνακα. Κάθε φορά που υπό-διπλασιάζεται το μέγεθος του πίνακα εκτυπώνεται στο **stdout** το μήνυμα "`Capacity: X -> Y`", όπου **X** η αρχική χωρητικότητα και **Y** η νέα χωρητικότητα του πίνακα.
`Graph(bool isDirected, int capacity)`	Η πρώτη παράμετρος προσδιορίζει εάν το γράφημα είναι κατευθυνόμενο ή όχι και η δεύτερη παράμετρος προσδιορίζει το μέγεθος του αρχικού πίνακα γειτνιάσεως ή του αρχικού πίνακα που περιέχει τις λίστες γειτνιάσεως για κάθε κόμβο.
`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***. Εάν πριν την ένθεση ο πίνακας γειτνιάσεως ή ο πίνακας της λίστας γειτνιάσεως δεν έχει διαθέσιμές θέσεις προς πλήρωση καλείται η συνάρτηση ***expand_table*** για τον διπλασιασμό του μεγέθους του αρχικού πίνακα.
`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***. Εάν μετά τη διαγραφή ο πίνακας γειτνιάσεως ή ο πίνακας της λίστας γειτνιάσεως έχει συνολικές θέσεις περισσότερες από τέσσερις (4) φορές τον αριθμό των κόμβων του γραφήματος ο πίνακας υποδιπλασιάζεται καλώντας τη συνάρτηση ***shrink_table*** για τον υπο-διπλασιασμό του μεγέθους του.
`bool addEdg(const T& from, const T& to, int cost);`	Ενθέτει την ακμή με αφετηρία τον κόμβο from και προορισμό τον κόμβο to και κόστος cost. Για την επιτυχή ένθεση της ακμής θα πρέπει να ισχύουν τα εξής:

	- οι κόμβοι from και to θα πρέπει να υπάρχουν στο γράφημα.. - η ακμή δεν πρέπει να υπάρχει στο γράφημα. **Σημείωση:** Σε ένα μη κατευθυνόμενο γράφημα η σειρά των κόμβων from και to της ακμής δεν έχει σημασία.
`bool rmvEdg(const T& from, const T& to);`	Διαγράφει την ακμή με αφετηρία τον κόμβο from και προορισμό τον κόμβο to και κόστος cost. Για την επιτυχή διαγραφή της ακμής, αυτή θα πρέπει να υπάρχει στο γράφημα. **Σημείωση:** Σε ένα μη κατευθυνόμενο γράφημα η σειρά των κόμβων from και to από τους οποίους προσδιορίζεται η ακμή δεν έχει σημασία.
`std::list<T> dfs(const T& info) const;`	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. Η σειρά επίσκεψης τον 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 τον 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 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.

`list<T> bellman_ford(`
`  const T& from,`
`  const T& to);`
	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.
	
	Επιπλέον, ο αλγόριθμος θα πρέπει να μπορεί να ανιχνεύσει εάν υπάρχει βρόγχος αρνητικού βάρους εντός τους 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!"**.
`bool print2DotFile(const char *filename) const;`	Εκτυπώνει το γράφημα σε ένα αρχείο κειμένου κατάλληλο για ανάγνωση και μετασχηματισμό σε εικόνα από το πρόγραμμα dot. Επιστρέφει true εάν η εγγραφή ήταν επιτυχής διαφορετικά false. Εάν το αρχείο έχει περιεχόμενο κατά το άνοιγμα, το υφιστάμενο περιεχόμενο διαγράφονται. 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.

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**. - 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).
`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.
`q`	Μία 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:

Test	Περιγραφή	Εξαρτάται από
test1		
test2		
test3		
test4		
test5		
test6		
test7		
test8		
test9		
test10		

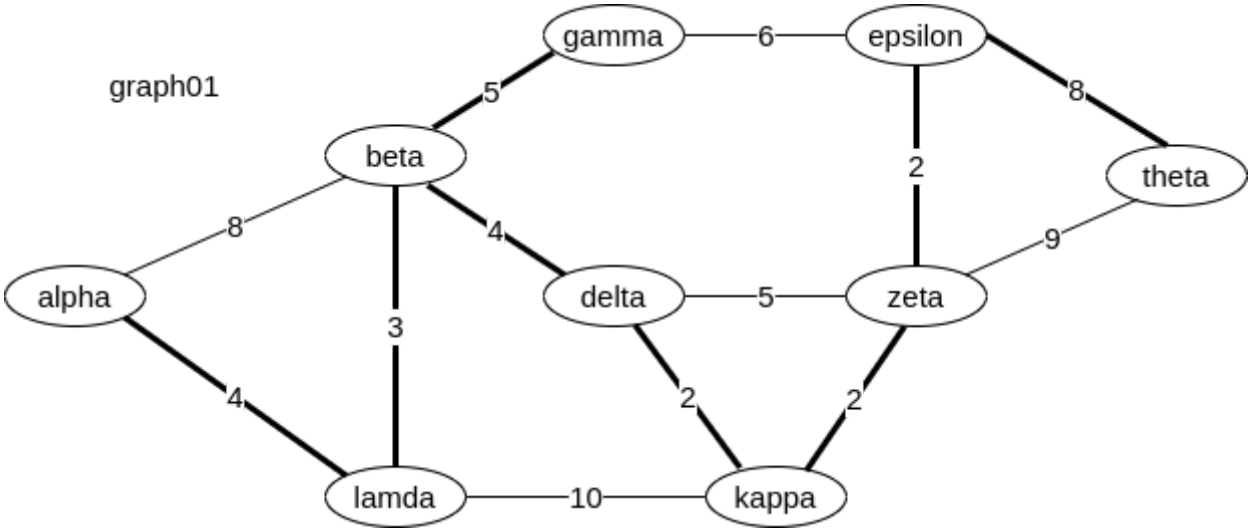
Τρόπος Αποστολής

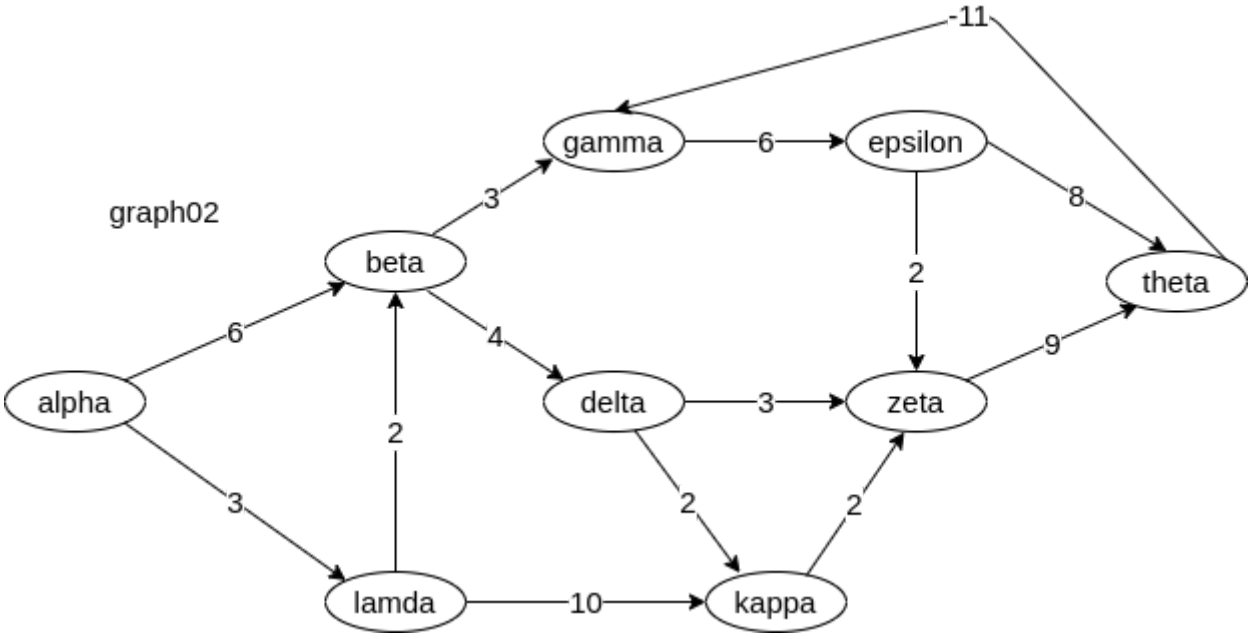
Η εργασία θα εξεταστεί μέσω του εργαλείου αυτόματης υποβολής και διόρθωσης autolab. Οι οδηγίες αποστολής είναι οι εξής:

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** και **GraphUI.hpp** της εργασίας σας.
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**.

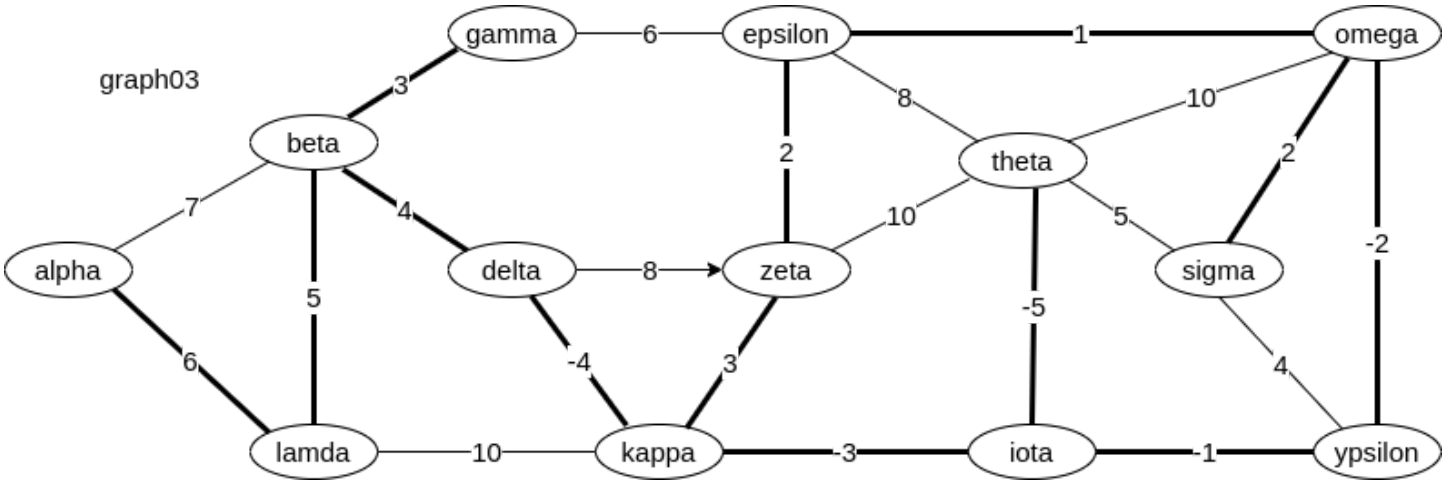
Παράρτημα 1 - Ενδεικτικά γραφήματα

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

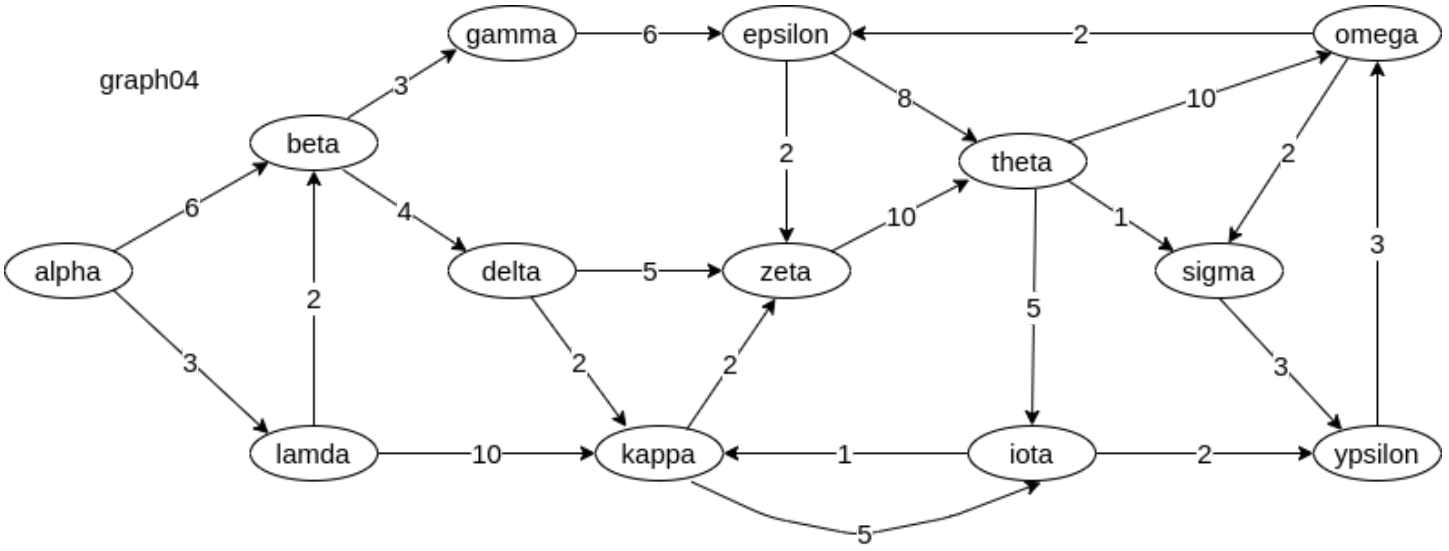




graph03



graph04



graph05