ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΤΕΙ ΗΠΕΙΡΟΥ	Σχολή Τεχνολογικών Εφαρμογών (ΣΤΕΦ) Τμήμα Μηχανικών Πληροφορικής Τ.Ε. Διδάσκων: Γκόγκος Χρήστος Μάθημα: Τεχνητή Νοημοσύνη (εργαστήριο Δ' εξαμήνου)	Ακαδημαϊκό έτος 2016-2017 εαρινό εξάμηνο

ΑΣΚΗΣΕΙΣ ΓΙΑ ΤΟ ΕΡΓΑΣΤΗΡΙΟ 1

Άσκηση 1

Σχεδιάστε ένα μη κατευθυνόμενο γράφημα με 5 κόμβους και 7 ακμές στο χαρτί. Δημιουργήστε ένα αρχείο κειμένου που να περιέχει τη πληροφορία του γραφήματος που σχεδιάσατε όπως περιγράφεται στο εργαστήριο. Χρησιμοποιώντας τον κώδικα lab01_01.cpp του εργαστηρίου εμφανίστε τα περιεχόμενα του γραφήματος.

Άσκηση 2

Γράψτε πρόγραμμα που να δέχεται ως όρισμα γραμμής εντολών ένα όνομα αρχείου γραφήματος και να υπολογίζει τις ακόλουθες πληροφορίες:

1. Μέσος όρος βάρους ακμών για όλο το γράφημα
2. Το βαθμό κάθε κορυφής και το μέσο όρο βαθμού κορυφών για όλο το γράφημα
3. Ονόματα κορυφών με το μεγαλύτερο βαθμό
4. Όνομα κορυφής που βρίσκεται στη μεγαλύτερη απόσταση από όλες τις γειτονικές της

Άσκηση 3

Τροποποιώντας τις υλοποιήσεις των BFS, DFS και UCS του εργαστηρίου έτσι ώστε να επιστρέφουν το μήκος της διαδρομής να υπολογιστούν τα ακόλουθα για κάθε πιθανό συνδυασμό δύο πόλεων στο πρόβλημα tour_romania:

1. Πόσες φορές ο αλγόριθμος BFS δίνει συντομότερη διαδρομή από τον DFS;
2. Πόσες φορές ο αλγόριθμος DFS δίνει συντομότερη διαδρομή από τον BFS;
3. Πόσες φορές οι αλγόριθμοι DFS και BFS εντοπίζουν διαδρομή με το ίδιο μήκος;
4. Πόσες φορές ο καθένας από τους αλγορίθμους DFS και BFS δίνει τη βέλτιστη σε σχέση με το μήκος της διαδρομής λύση;

Ας σημειωθεί ότι καθώς οι πόλεις είναι 20 οι συνδυασμοί των πόλεων ανά 2 είναι choose(20,2)=190.

Αριθμός ζευγών πόλεων που ο DFS εντοπίζει συντομότερη διαδρομή από τον BFS	
Αριθμός ζευγών πόλεων που ο BFS εντοπίζει συντομότερη διαδρομή από τον DFS	
Αριθμός ζευγών πόλεων που ο DFS και ο BFS εντοπίζει διαδρομή με το ίδιο μήκος	
Αριθμός ζευγών πόλεων που ο BFS εντοπίζει τη βέλτιστη διαδρομή όπως υπολογίζεται από τον UCS	
Αριθμός ζευγών πόλεων που ο DFS εντοπίζει τη βέλτιστη διαδρομή όπως υπολογίζεται από τον UCS	

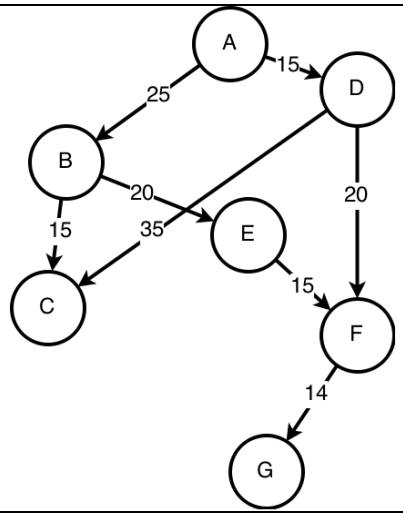
Άσκηση 4

Τροποποιήστε τον αλγόριθμο depth_first_search_base του εργαστηρίου έτσι ώστε να επιστρέφει το πλήθος των περιπτώσεων που ο αλγόριθμος οδηγήθηκε σε βρόχο. Γράψτε πρόγραμμα που να καλεί τον αλγόριθμο για το πρόβλημα tour_romania και να εμφανίζει το πλήθος των βρόχων για τη μετάβαση από όλους τους κόμβους προς τον κόμβο Β.

Άσκηση 5

Για το γράφημα toy1.txt με τα ακόλουθα δεδομένα να εφαρμοστούν οι αλγόριθμοι DFS, BFS και UCS για την μετάβαση από την κορυφή A στην κορυφή G και να συμπληρωθούν οι ακόλουθοι πίνακες

```
[VERTICES:7]
A
B
C
D
E
F
G
[EDGES:8]
A,B,25
A,D,15
B,C,15
B,E,20
D,C,35
D,F,20
E,F,15
F,G,14
```



toy1.txt

Για τον αλγόριθμο BFS (Αφετηρία: A, Στόχος: G)

Μέτωπο αναζήτησης	Κλειστό σύνολο	Τρέχουσα Κατάσταση	Παιδιά

Διαδρομή και κόστος:

Για τον αλγόριθμο DFS (Αφετηρία: A, Στόχος: G)

Μέτωπο αναζήτησης	Κλειστό σύνολο	Τρέχουσα Κατάσταση	Παιδιά

Διαδρομή και κόστος:

Για τον αλγόριθμο UCS (Αφετηρία: A, Στόχος: G)

Μέτωπο αναζήτησης	Κλειστό σύνολο	Τρέχουσα Κατάσταση	Παιδιά

Διαδρομή και κόστος:

Άσκηση 6

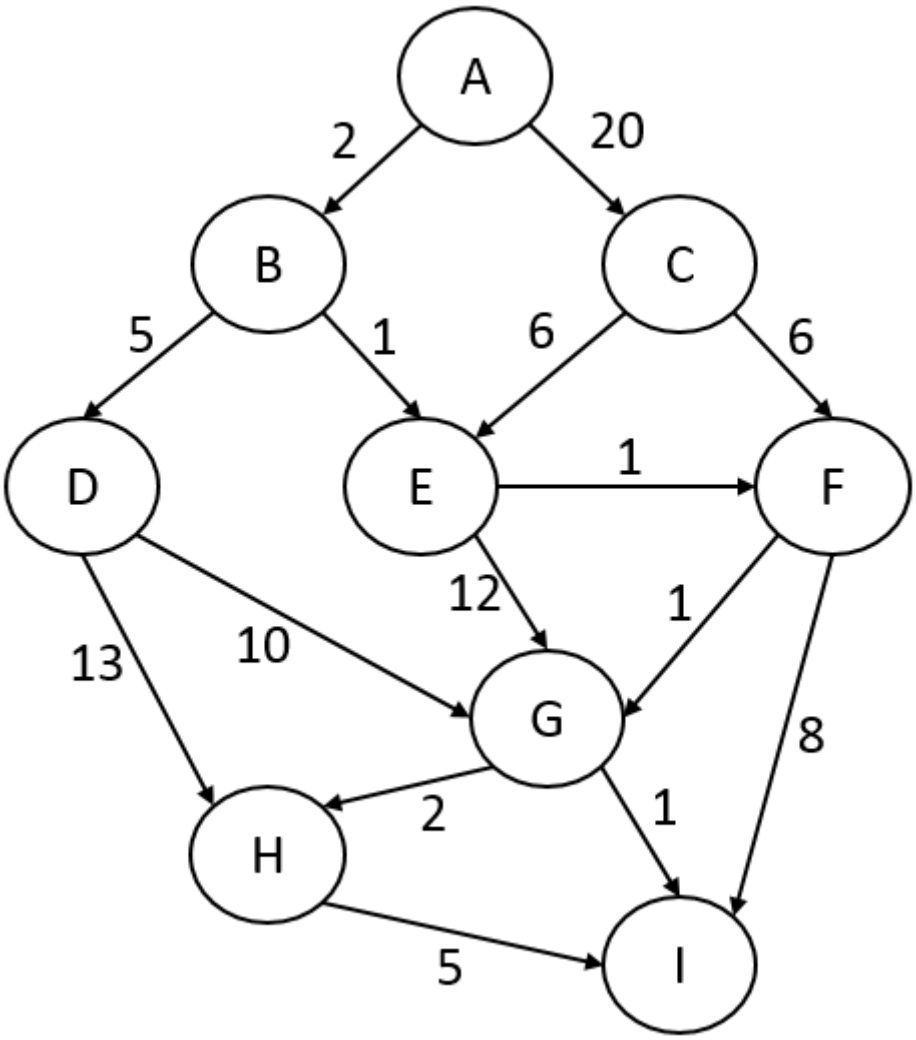
Μια εναλλακτική υλοποίηση του αλγορίθμου αναζήτησης κατά πλάτος είναι η ακόλουθη:

```
/**
 * εναλλακτική υλοποίηση του bfs
 *
 */
void breadth_first_search_alt(struct di_graph graph, string start_vertex,
  string goal_vertex) {
 cout << "BREADTH FIRST SEARCH" << endl;
 set<string> closed { };
 queue<search_node> frontier { }; // FIFO
 frontier.push(to_search_node(start_vertex));
 bool found { false };
 while (true) {
  if (frontier.empty())
   break;
  search_node current_state = frontier.front();
  string current_state_back = current_state.path.back();
  print_status(closed, queue_to_list(frontier), current_state_back,
    get_successors(graph, current_state_back));
  frontier.pop();
  if (goal_vertex.compare(current_state_back) == 0) {
   cout << "Path to goal node found: "
     << search_node_as_string(current_state) << endl;
   found = true;
   break;
  }
  bool is_in { closed.find(current_state_back) != closed.end() };
  if (!is_in) {
   closed.insert(current_state_back);
   for (string v : get_successors(graph, current_state_back))
    frontier.push(to_search_node(graph, current_state, v));
  }
 }
 if (!found)
  cout << "Goal not found!" << endl;
}
```

Κατασκευάστε ανάλογες υλοποιήσεις για τον αλγόριθμο αναζήτησης κατά βάθος (DFS) και για τον αλγόριθμο ομοιόμορφης αναζήτησης (UCS). Ελέγξτε την ορθότητα των υλοποιήσεων συγκρίνοντας τα αποτελέσματα από τις δικές σας υλοποιήσεις με τα αποτελέσματα των ήδη διατυπωμένων αλγορίθμων. Θα πρέπει να σημειωθεί ότι κατά την εισαγωγή νέων καταστάσεων στο μέτωπο αναζήτησης χρησιμοποιείται η σύμβαση ότι ανάμεσα στις καταστάσεις που προστίθενται προτεραιότητα θα πρέπει να αποκτούν εκείνες που προηγούνται στην αλφαβητική σειρά (δηλαδή ο κόμβος A εφόσον εισήλθε μαζί με τον κόμβο B στο μέτωπο αναζήτησης θα πρέπει να εξέλθει πριν τον κόμβο B). Άρα στην περίπτωση του DFS η συνάρτηση **`get_successors`** θα πρέπει να κληθεί ως εξής: **`get_successors(graph, current_state_back, false)`**. Ορίζοντας την τελευταία παράμετρο της συνάρτησης ως false οι κόμβοι επιστρέφονται σε φθίνουσα σειρά και ωθούνται στη στοίβα. Κατά την απώθηση τους στη συνέχεια εξέρχονται σε αύξουσα αλφαβητική σειρά.

Άσκηση 7

Για το ακόλουθο γράφημα να υπολογιστούν οι διαδρομές που υπολογίζουν οι αλγόριθμοι BFS, DFS και UCS για τη μετάβαση από την κορυφή A στην κορυφή I χωρίς να κατασκευαστούν οι πίνακες μετώπου αναζήτησης, κλειστού συνόλου κλπ.



Άσκηση 8 (προαιρετική)

Υλοποιήστε αναδρομικούς αλγορίθμους για τον αλγόριθμο αναζήτησης πρώτα κατά πλάτος και τον αλγόριθμο αναζήτησης πρώτα κατά βάθος. Ελέγξτε την ορθότητα των αλγορίθμων συγκρίνοντας τα αποτελέσματα που επιστρέφουν με τα αποτελέσματα που επιστρέφουν οι μη αναδρομικοί αλγόριθμοι του εργαστηρίου.