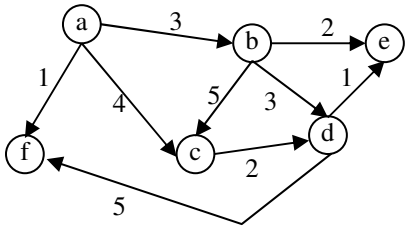
ΑΤΕΙ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΜΑΘΗΜΑ: ΕΡΓΑΣΤΗΡΙΟ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ (Prolog)
ΕΞΑΜΗΝΟ: Δ΄- Εαρινό 2018-19
ΚΑΘΗΓΗΤΕΣ: Δ.ΣΤΑΜΑΤΗΣ, Μ.ΒΟΖΑΛΗΣ, Κ.ΚΡΑΒΑΡΗ

ΑΣΚΗΣΗ 8:
ΓΡΑΦΗΜΑΤΑ ή ΓΡΑΦΟΙ (GRAPHS)

Οι γράφοι απαντώνται σε όλους τους κλάδους της πληροφορικής (μεταφραστές, βάσεις δεδομένων, λειτουργικά συστήματα, παράλληλη επεξεργασία, δίκτυα υπολογιστών, κλπ) και φυσικά και στην τεχνητή νοημοσύνη. Οι γράφοι είναι δομές δεδομένων που αποτελούνται από κόμβους συνδεδεμένους με ακμές. Χωρίζονται σε δύο κατηγορίες: (α) κατευθυνόμενους γράφους, όπου οι ακμές έχουν συγκεκριμένη κατεύθυνση και (β) μη-κατευθυνόμενους γράφους όπου οι ακμές θεωρούνται ότι έχουν και τις δύο κατευθύνσεις. Για παράδειγμα, ο παρακάτω γράφος είναι κατευθυνόμενος.



Συχνά οι ακμές των γράφων συνδέονται με κάποιο κόστος "διαδρομής". Θα μπορούσαμε για παράδειγμα να φανταστούμε ότι ο παραπάνω γράφος παριστάνει το οδικό δίκτυο μεταξύ πόλεων. Στην περίπτωση αυτή το κόστος κάθε ακμής θα μπορούσε να αντιστοιχεί στη χιλιομετρική απόσταση μεταξύ των πόλεων. Έτσι σύμφωνα με το γράφο αυτό το κόστος που πληρώνουμε για να πάμε από τον κόμβο a στον κόμβο b είναι 3 (μετρούμενο σε οποιοδήποτε σύστημα εμείς έχουμε επιλέξει).

Στην Prolog οι γράφοι όπως ο παραπάνω μπορούν πολύ εύκολα να παρασταθούν με γεγονότα που δηλώνουν την γειτνίαση των κόμβων. Αν **δε μας ενδιαφέρει το κόστος** των ακμών τότε ο γράφος μπορεί να περιγραφεί με τα παρακάτω γεγονότα:

```
next_to(a,b).
next_to(a,c).
next_to(a,f).
next_to(b,c).
next_to(b,d).
next_to(b,e).
next_to(c,d).
next_to(d,e).
next_to(d,f).
```

Επειδή ο γράφος είναι **κατευθυνόμενος**, η σειρά των ορισμάτων στο `next_to` έχει σημασία. Με άλλα λόγια το `next_to(a,b)` είναι διαφορετικό από το `next_to(b,a)` καθώς το πρώτο δηλώνει την ύπαρξη μιας ακμής από το a στο b ενώ το δεύτερο δηλώνει μια ακμή από το b στο a.

Αν θέλαμε να περιγράψουμε ένα **μη-κατευθυνόμενο** γράφο τότε έχουμε δύο επιλογές:

(α) να γράψουμε δύο γεγονότα για κάθε ακμή περιγράφοντας τις δύο κατευθύνσεις. Π.χ. για την ακμή που συνδέει τους κόμβους a και του b θα γράφαμε

```
next_to(a,b).
next_to(b,a).
```

(β) να γράψουμε μια μόνο κατεύθυνση και να χρησιμοποιήσουμε ένα νέο κατηγόρημα, το `next` το οποίο θα μας λέει ότι αν οι κόμβοι X, Y, είναι γείτονες τότε και οι Y, X, είναι γείτονες

```
next(X,Y) :-
    next_to(X,Y).
next(X,Y) :-
    next_to(Y,X).
```

Αν **μας ενδιαφέρει το κόστος** των ακμών τότε θα μπορούσαμε να χρησιμοποιήσουμε ένα άλλο κατηγόρημα, π.χ. το `link` με τρία ορίσματα όπου τα δύο πρώτα αντιστοιχούν στους συνδεόμενους κόμβους και το τρίτο στο κόστος της ακμής:

```
link(a,b,3).          link(b,e,2).
link(a,c,4).          link(c,d,2).
link(a,f,1).          link(d,e,1).
link(b,c,5).          link(d,f,5).
link(b,d,3).
```

Επεξεργασία ενός κατευθυνόμενου γράφου

Ο λόγος για τον οποίο αναπαριστούμε ένα γράφο είναι για να τον χρησιμοποιήσουμε κατόπιν με σκοπό να απαντήσουμε σε κάποιες ερωτήσεις. Μια σχετικά απλή ερώτηση είναι αν υπάρχει μονοπάτι μεταξύ δύο κόμβων X και Y

```
existspath(X,Y) :-
    next_to(X,Y).
existspath(X,Y) :-
    next_to(X,Z),
    existspath(Z,Y).
```

Το κατηγόρημα `existspath` απαντάει Yes ή No ανάλογα με το αν υπάρχει μονοπάτι ή όχι. Ο πρώτος κανόνας είναι η οριακή συνθήκη που δηλώνει ότι υπάρχει μονοπάτι αν οι X και Y γειτνιάζουν. Ο δεύτερος κανόνας είναι αναδρομικός και δηλώνει ότι υπάρχει μονοπάτι μεταξύ X και Y αν υπάρχει ένας γείτονας Z του X και υπάρχει μονοπάτι μεταξύ του Z και του Y.

ΤΙ ΠΡΕΠΕΙ ΝΑ ΚΑΝΕΤΕ

Χρησιμοποιήστε το γράφο που δώσαμε σαν παράδειγμα και γράψτε τα παρακάτω κατηγορήματα

(α) `path(X,Y,Path)`. Το κατηγόρημα αυτό θα βρίσκει το μονοπάτι που συνδέει τους κόμβους X και Y και θα το επιστρέφει στη λίστα Path. Παράδειγμα:

```
?- path(a,e,Path).
Path = [a,b,e];
Path = [a,b,d,e];
Path = [a,c,d,e]; ...κλπ
```

(β) `pathlength(X,Y,Path,Length)`. Το κατηγόρημα αυτό θα είναι μια επέκταση του `path` που θα βρίσκει και το μονοπάτι που συνδέει τους κόμβους X και Y αλλά και το μήκος του μονοπατιού (δηλαδή πόσες ακμές περιέχει αυτό). Παράδειγμα:

```
?- pathlength(a,e,Path,Length).
Path = [a,b,e]
Length = 2;
Path = [a,b,d,e]
Length = 3;
Path = [a,c,d,e]
Length = 3; ...κλπ
```

(γ) `pathcost(X,Y,Path,Cost)`. Το κατηγόρημα αυτό θα είναι μια άλλη επέκταση του `path` που θα βρίσκει και το μονοπάτι που συνδέει τους κόμβους X και Y αλλά και το κόστος του μονοπατιού, δηλαδή το άθροισμα των επί μέρους κοστών των ακμών που το αποτελούν. Παράδειγμα:

```
?- pathcost(a,e,Path,Cost).
Path = [a,b,e]
Cost = 5;
Path = [a,b,d,e]
Cost = 7;
Path = [a,c,d,e]
Cost = 7; ...κλπ
```

(δ) `pathloop(X,Y,Visited,Path)`. Το κατηγόρημα αυτό θα είναι μια άλλη επέκταση του `path` που θα βρίσκει και το μονοπάτι που συνδέει τους κόμβους X και Y αλλά θα φροντίζει να μη περνάει από τους ίδιους κόμβος για δεύτερη φορά, χρησιμοποιώντας την αρχικά είναι κενή βοηθητική λίστα Visited για να αποθηκεύει τους κόμβους που επισκέφτηκε . Παράδειγμα:

```
?- pathloop(a,e,[],Path).
```

Τι ψάχνει η κλήση

```
?- pathloop(a,e,[b],Path).
```