Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
https://courses.softlab.ntua.gr/progtech/

Προγραμματιστικές Τεχνικές

Άσκηση 10 Δυαδικά Δένδρα

Προθεσμία υποβολής στον grader: 22/5/2020

Υλοποιήστε την κλάση `lexicon` για την αναπαράσταση λεξιλογίων κειμένων με τη μορφή δυαδικών δένδρων αναζήτησης. Η κλάση θα πρέπει να υποστηρίζει τις παρακάτω λειτουργίες, τις οποίες πρέπει να υλοποίησετε:

```
class lexicon {
public:
  lexicon();
  ~lexicon();

  void insert(const string &s);
  int  lookup(const string &s) const;
  int  depth(const string &s) const;
  void replace(const string &s1, const string &s2);

  friend ostream & operator << (ostream &out, const lexicon &l);
};
```

Θεωρήστε δεδομένο ότι οι λέξεις που θα αποθηκεύονται σε ένα λεξικό δεν θα είναι κενές και θα περιέχουν μόνο μικρά λατινικά γράμματα. Κάθε λέξη θα αντιστοιχεί σε έναν κόμβο στο δένδρο. Ο κόμβος θα περιέχει τη λέξη και τη συχνότητα εμφάνισής της στο λεξικό (δηλαδή έναν μετρητή που θα θυμάται πόσες φορές έχει εισαχθεί αυτή η λέξη). Το αριστερό παιδί του κόμβου θα περιέχει αλφαβητικά (λεξικογραφικά) μικρότερες λέξεις, ενώ το δεξιό παιδί αλφαβητικά (λεξικογραφικά) μεγαλύτερες. Καμία λέξη δεν θα πρέπει να εμφανίζεται σε περισσότερους από έναν κόμβους στο δένδρο. Επίσης, το δένδρο θα πρέπει να είναι ένα απλό BST: δεν απαιτείται και δεν πρέπει να είναι ισοζυγισμένο (π.χ., AVL).

Η μέθοδος `insert(s)` θα εισάγει τη λέξη `s` στο δένδρο.

Η μέθοδος `lookup(s)` θα αναζητά τη λέξη `s` στο δένδρο και θα επιστρέφει τη συχνότητα εμφάνισής. Το αποτέλεσμα θα είναι 0 (μηδέν) αν η λέξη δεν υπάρχει στο δένδρο.

Η μέθοδος `depth(s)` θα αναζητά τη λέξη `s` στο δένδρο, όπως και η `lookup`. Αν δεν υπάρχει, θα επιστρέφει 0 (μηδέν). Αν όμως υπάρχει, θα επιστρέφει το *βάθος* στο οποίο βρίσκεται ο κόμβος που την περιέχει στο δένδρο, δηλαδή το μήκος του μονοπατιού από τη ρίζα έως αυτόν τον κόμβο. Θεωρούμε ότι η ρίζα του δένδρου βρίσκεται σε βάθος 1 (ένα).

Η μέθοδος `replace(s1, s2)` Θα αντικαθιστά όλες τις εμφανίσεις της λέξης `s1` με ισάριθμες εμφανίσεις της λέξης `s2`. Αν η `s1` δεν υπάρχει στο δένδρο, τότε δε θα γίνεται τίποτα. Αν υπάρχει και το πλήθος εμφανίσεών της είναι $k > 0$, τότε η `s1` θα διαγράφεται και θα αναζητάται η λέξη `s2`. Αν αυτή δεν υπάρχει, θα εισάγεται με συχνότητα εμφάνισης k. Αν όμως υπάρχει, τότε η συχνότητα εμφάνισής της θα ενημερώνεται κατάλληλα (θα αυξάνει κατά k).

Κατά τη διαγραφή μίας λέξης από το λεξικό (που γίνεται έμμεσα με τη μέθοδο `replace`), αν διαγράφεται κόμβος που έχει δύο μη κενά παιδιά, τότε ο κόμβος που διαγράφεται αντικαθίσταται από αυτόν

που περιέχει την αμέσως μικρότερη λέξη. Αν διαγράφεται κόμβος που έχει ένα μη κενό παιδί, τότε αντικαθίσταται από το παιδί του.

Η εκτύπωση των λεξικών πρέπει να γίνεται σε αλφαβητική σειρά, με τις λέξεις να ακολουθούνται από τη συχνότητα εμφάνισής τους.

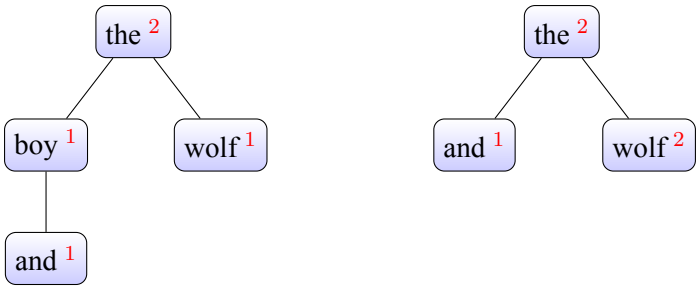
Μπορείτε να δοκιμάσετε την υλοποίησή σας με προγράμματα όπως το εξής:

```
int main() {
  lexicon l;
  l.insert("the");
  l.insert("boy");
  l.insert("and");
  l.insert("the");
  l.insert("wolf");
  cout << "The word 'the' is found " << l.lookup("the") << " time(s)" << endl;
  cout << "The word 'and' is found at depth " << l.depth("and") << endl;
  cout << l;
  l.replace("boy", "wolf");
  cout << "After replacement:\n";
  cout << l;
  cout << "Now the word 'and' is found at depth " << l.depth("and") << endl;
}
```

Η εκτέλεσή του θα πρέπει να εμφανίζει:

```
The word 'the' is found 2 time(s)
The word 'and' is found at depth 3
and 1
boy 1
the 2
wolf 1
After replacement:
and 1
the 2
wolf 2
Now the word 'and' is found at depth 2
```

Η μορφή του δένδρου πριν (αριστερά) και μετά (δεξιά) την κλήση της `replace` δίνεται παρακάτω.



Στο αρχείο που θα ανεβάσετε στον grader θα πρέπει να συμπεριλάβετε (μόνο) τη δήλωση της κλάσης `lexicon` και τις υλοποιήσεις των μεθόδων της.

Άσκηση 11 Μονοπάτια και κύκλοι Euler

Προθεσμία υποβολής στον grader: 22/5/2020

Δίνεται ένας μη κατευθυνόμενος γράφος N κορυφών (αριθμημένων από 0 έως $N-1$) και M ακμών. Θα είναι $2 \leq N \leq 10.000$ και $0 \leq M \leq 100.000$. Είναι πιθανό κάποια ακμή να υπάρχει πολλές φορές, όπως επίσης και να υπάρχουν κυκλικές ακμές — δηλαδή ακμές της μορφής (u, u). Θεωρήστε δεδομένο ότι ο γράφος θα είναι συνεκτικός, δηλαδή ότι θα είναι δυνατή η μετακίνηση από οποιαδήποτε κορυφή σε οποιαδήποτε άλλη, μέσω κάποιου μονοπατιού αποτελούμενου από ακμές.

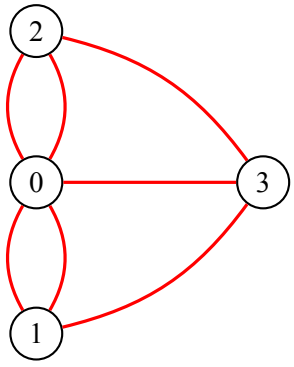
Γράψτε ένα πρόγραμμα που να διαβάζει τον γράφο από το standard input και να ελέγχει υπάρχει κύκλος ή μονοπάτι Euler στον γράφο. Συγκεκριμένα, το πρόγραμμά σας πρέπει να εκτυπώνει στο standard output μία γραμμή που να περιέχει ένα από τα εξής τρία:

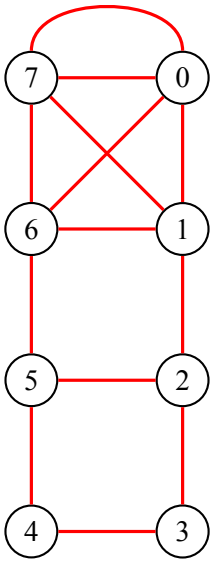
- "`CYCLE`", αν υπάρχει κύκλος Euler.
- "`PATH u v`", αν υπάρχει δεν υπάρχει κύκλος Euler, υπάρχει όμως μονοπάτι Euler από την κορυφή u στην κορυφή v — φροντίστε να είναι $u < v$.
- "`IMPOSSIBLE`", αν δεν υπάρχει ούτε κύκλος ούτε μονοπάτι Euler.

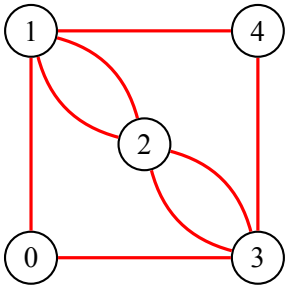
Η είσοδος θα περιέχει τα εξής. Η πρώτη γραμμή θα περιέχει δύο αριθμούς, χωρισμένους μεταξύ τους με ένα κενό διάστημα: το πλήθος των κορυφών N και το πλήθος των ακμών M. Οι επόμενες M γραμμές της εισόδου θα περιγράφουν τις ακμές του γράφου. Κάθε μία θα περιέχει δύο αριθμούς, χωρισμένους μεταξύ τους με ένα κενό διάστημα: η γραμμή που περιέχει τους αριθμούς u και v θα παριστάνει μία ακμή μεταξύ των κορυφών u και v του γράφου.

Μπορείτε να προσαρμόσετε κατάλληλα τους γνωστούς τρόπους αναπαράστασης γράφων ώστε να καλύπτουν την περίπτωση γράφων με πολλαπλές και με κυκλικές ακμές.

Παραδείγματα γράφων:







Είσοδος:

```
4 7
0 1
0 1
0 2
0 2
3 0
3 1
3 2
```

```
8 13
0 7
0 7
0 1
0 6
1 7
7 6
6 1
1 2
6 5
2 5
2 3
3 4
5 4
```

```
5 8
0 1
1 2
0 3
1 4
1 2
2 3
3 2
4 3
```

Έξοδος:

```
IMPOSSIBLE
```

```
PATH 2 5
```

```
CYCLE
```

Άσκηση 12 Κύκλος σε κατευθυνόμενο γράφο

Προθεσμία υποβολής στον grader: 22/5/2020

Υλοποιήστε την κλάση `Graph` για την αναπαράσταση κατευθυνόμενων γράφων. Χάριν ευκολίας, θα θεωρήσουμε ότι ένας γράφος περιέχει V κορυφές αριθμημένες από 0 έως $V-1$. Η κλάση σας πρέπει να υποστηρίζει τις εξής λειτουργίες, τις οποίες πρέπει να υλοποιήσετε:

```
class Graph {
 public:
  Graph(int V);
  ~Graph();
  void addEdge(int u, int v);
};
```

Στη συνέχεια, προσθέστε την παρακάτω μέθοδο στην κλάση σας:

```
  bool cycle(vector<int> &path) const;
```

Η μέθοδος αυτή πρέπει να επιστρέφει **`true`** αν υπάρχει κύκλος στο γράφο, διαφορετικά **`false`**. Στην πρώτη περίπτωση, η μέθοδος πρέπει να προσθέτει στο `path` τους κόμβους που απαρτίζουν τον κύκλο. Δηλαδή, αν στο γράφο υπάρχουν οι ακμές $1 \to 4$, $4 \to 2$, $2 \to 7$ και $7 \to 1$, τότε η μέθοδός σας μπορεί να προσθέσει στο `path` τις τιμές $1, 4, 2, 7$ με αυτή τη σειρά. Εξίσου καλά μπορεί να προσθέσει τις τιμές $2, 7, 1, 4$, ή οποιονδήποτε άλλον έγκυρο κύκλο βρει.

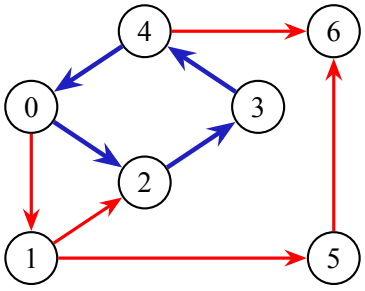
Στο αρχείο που θα ανεβάσετε στον grader θα πρέπει να συμπεριλάβετε (μόνο) τη δήλωση της κλάσης `Graph` και τις υλοποιήσεις των μεθόδων της.

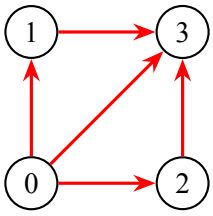
Μπορείτε να δοκιμάσετε την υλοποίησή σας με προγράμματα όπως το εξής (όπως ήδη γνωρίζετε από προηγούμενες ασκήσεις, μπορείτε να προσθέσετε τη συνάρτηση `main` ανάμεσα σε "**`#ifndef CONTEST`**" και "**`#endif`**", αν θέλετε να υποβάλλετε τον κώδικά σας στον grader χωρίς να τη σβήνετε):

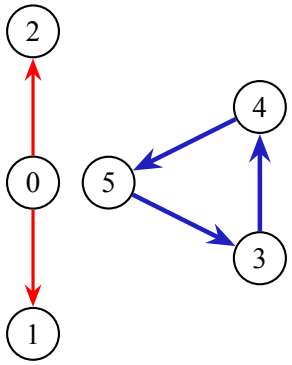
```cpp
int main() {
  int V, E;
  cin >> V >> E;
  Graph g(V);
  for (int i = 0; i < E; ++i) { int u, v; cin >> u >> v; g.addEdge(u, v); }
  vector<int> path;
  bool c = g.cycle(path);
  if (c) {
    cout << "CYCLE: ";
    for (int i = 0; i < path.size(); ++i)
      cout << path[i] << (i == path.size()-1 ? "\n" : " ");
  } else {
    cout << "NO CYCLE" << endl;
  }
}
```

Η εκτέλεση του παραπάνω προγράμματος θα πρέπει να έχει την εξής συμπεριφορά, ανάλογα με την είσοδο που θα του δίνεται και που θα περιγράφει το γράφο:

Παραδείγματα γράφων:







Είσοδος:

```
7 9
0 1
0 2
1 2
2 3
3 4
4 0
4 6
1 5
5 6
```

```
4 5
0 1
0 2
1 3
2 3
0 3
```

```
6 5
0 1
0 2
3 4
4 5
5 3
```

Έξοδος:

```
CYCLE: 0 2 3 4
```

```
NO CYCLE
```

```
CYCLE: 3 4 5
```