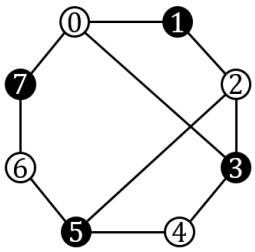
Πανεπιστήμιο Ιωαννίνων – Τμήμα Μηχανικών Η/Υ και Πληροφορικής
Δομές Δεδομένων [ΜΥΥ303] – Χειμερινό Εξάμηνο 2020

4η Εργαστηριακή Άσκηση
Διερεύνηση Γραφήματος
Παράδοση έως Πέμπτη 19/11, 12:00 από το eCourse

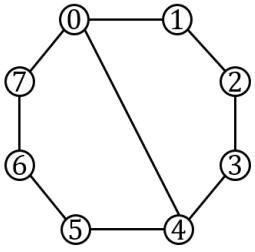
ΠΡΟΣΟΧΗ: Γράψτε σε κάθε αρχείο που παραδίδετε τα ονόματα και τους Α.Μ. των μελών της ομάδας σας. Συμπεριλάβετε όλα τα αρχεία σας (κώδικας Java και lab4results.txt) σε ένα zip αρχείο το οποίο να έχει το όνομα ενός μέλους της ομάδας σας με λατινικούς χαρακτήρες.

Μας δίνεται ένα ημιτελές πρόγραμμα BipartiteGraph.java το οποίο πρέπει να συμπληρώσουμε ώστε να επιτελεί την ακόλουθη λειτουργία. Το πρόγραμμα δέχεται στην είσοδο ένα γράφημα $G = (V, E)$ με $|V| = N$ κόμβους και $|E| = K$ ακμές, όπου οι κόμβοι είναι αριθμημένοι από 0 έως $N - 1$. Σκοπός του προγράμματος είναι να ανακαλύψει εάν το γράφημα είναι *διμερές*, δηλαδή αν το σύνολο των κόμβων μπορεί να διαμεριστεί σε δύο υποσύνολα, το άσπρο A και το μαύρο M, τέτοια ώστε κάθε ακμή να συνδέει ένα άσπρο κόμβο u ($u \in A$) με ένα μαύρο κόμβο v ($v \in M$).

Είναι γνωστό ότι ένα γράφημα είναι διμερές αν και μόνο αν δεν έχει κύκλους περιττού μήκους (δηλαδή με περιττό πλήθος κόμβων/ακμών). Στο παρακάτω σχήμα το γράφημα G είναι διμερές, καθώς δίνεται ένας κατάλληλος χρωματισμός των κόμβων σε άσπρους και μαύρους όπου κάθε ακμή ενώνει κόμβους διαφορετικού χρώματος. Αντίθετα, το G' δεν είναι διμερές καθώς έχει κύκλους περιττού μήκους 5, τους (0,1,2,3,4,0) και (0,7,6,5,4,0).

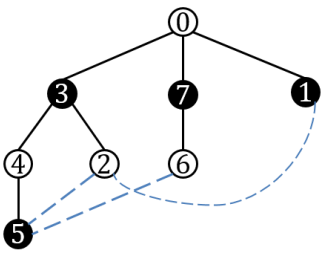


G

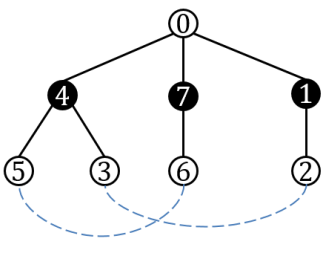


G'

Η οριζόντια διερεύνηση (breath-first search – BFS) δίνει έναν αποτελεσματικό τρόπο για να ελέγξουμε αν ένα γράφημα είναι διμερές. Ξεκινάμε τη διερεύνηση από ένα αυθαίρετο κόμβο, π.χ. τον 0, και χρωματίζουμε τους κόμβους ανά επίπεδο του BFS δένδρου, έτσι ώστε οι κόμβοι στα άρτια επίπεδα να είναι άσπροι και στα περιττά επίπεδα μαύροι. Αν βρούμε μια ακμή που να συνδέει κόμβους του ίδιου χρώματος τότε το γράφημα έχει κύκλο περιττού μήκους και επομένως δεν είναι διμερές. Για παράδειγμα, η εκτέλεση της παραπάνω διαδικασίας στα γραφήματα G και G' δίνει το ακόλουθο αποτέλεσμα:



BFS δένδρο του G



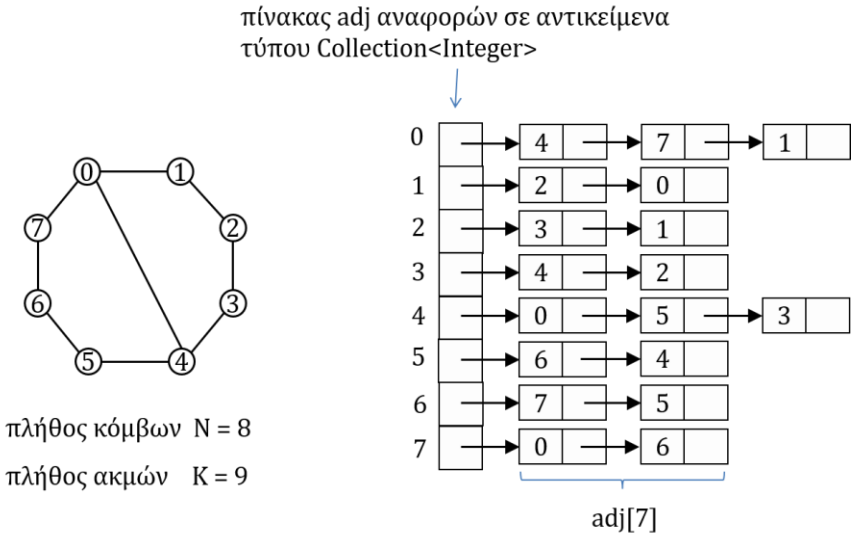
BFS δένδρο του G'

Στο G' ανακαλύπτουμε δύο ακμές, {2,3} και {5,6}, οι οποίες συνδέουν κόμβους ίδιου χρώματος και ορίζουν αντίστοιχα τους κύκλους περιττού μήκους (2,1,0,4,3,2) και (5,4,0,7,6,5).

Η main() μέθοδος του BipartiteGraph.java διαβάζει ένα γράφημα από την είσοδο, με τη βοήθεια του βοηθητικού αρχείου In.java. Το γράφημα οποίο δίνεται ως μια ακολουθία ακεραίων. Οι δύο πρώτοι ακέραιοι δίνουν τον αριθμό των κόμβων N και τον αριθμό των ακμών K. Στη συνέχεια, το πρόγραμμα διαβάζει K ζεύγη ακεραίων, όπου κάθε ζεύγος αντιστοιχεί σε μια ακμή και αποτελείται από δύο ακέραιους από 0 έως $N-1$. Το γράφημα G' του παραπάνω παραδείγματος δίνεται ως

```
8
9
0 1
1 2
2 3
3 4
4 5
5 6
6 7
7 0
0 4
```

Μετά την ανάγνωση του γραφήματος, το πρόγραμμα κατασκευάζει τις αντίστοιχες λίστες γειτνίασης χρησιμοποιώντας ένα πίνακα adj[], όπως φαίνεται στο παρακάτω σχήμα.



Η προσπέλαση των στοιχείων μιας λίστας γίνεται με τη βοήθεια επαναληπτών (iterators). Για να προσπελάσουμε όλους τους γείτονες w του κόμβου v γράφουμε

```
for (int w : adj(v)) { ... }
```

Δείτε την ενδεικτική μέθοδο printGraph().

Για να εκτελέσουμε το πρόγραμμα με είσοδο ένα αρχείο exampleGraph δίνουμε την εντολή java BipartiteGraph < exampleGraph.

Ζητούμενες Μέθοδοι

Συμπληρώστε στο αρχείο BipartiteGraph.java τις ακόλουθες μεθόδους:

void bfs(int s)	Εκτελεί την οριζόντια διερεύνηση με αφετηρία ένα κόμβο s και χρωματίζει τους κόμβους ανά επίπεδο, όπως περιγράψαμε παραπάνω. Θα χρειαστεί να υλοποιήσετε στο αρχείο Queue.java μια FIFO ουρά η οποία αποθηκεύει τους κόμβους του γραφήματος καθώς τους ανακαλύπτει η οριζόντια διερεύνηση.
Queue<Integer> findOddCycle(int v, int u)	Εκτελείται όταν το γράφημα *δεν είναι διμερές*. Δέχεται ως όρισμα μια ακμή (v, u) που συνδέει κόμβους του *ίδιου χρώματος* και επιστρέφει μια FIFO ουρά C η οποία περιέχει ένα κύκλο περιττού μήκος που ορίζει αυτή η ακμή στο δένδρο της οριζόντιας διερεύνησης. ***Σημείωση:*** *Οι κόμβοι του κύκλου πρέπει να αποθηκεύονται σε μια σωστή (κυκλική) σειρά στην ουρά.*
void printCycle (Queue <Integer> C)	Τυπώνει τον κύκλο που περιέχει η ουρά C.

Για παράδειγμα, η κλήση $C = \text{findOddCycle}(6,5)$ στο παραπάνω παράδειγμα μπορεί να επιστρέψει μια ουρά $C = (5, 4, 0, 7, 6, 5)$.

Τα αρχεία In.java και Collection.java σας δίνονται έτοιμα και δε χρειάζονται κάποια αλλαγή. Θα πρέπει να υλοποιήσετε τις μεθόδους σας στα αρχεία BipartiteGraph.java και Queue.java.

Εκτέλεση Προγραμμάτων

Εκτελέστε το συμπληρωμένο πρόγραμμα σας BipartiteGraph.java με είσοδο τα αρχεία smallGraph1.txt, smallGraph2.txt, largeGraph1.txt και largeGraph2.txt. Αποθηκεύστε τα αποτελέσματα της κάθε εκτέλεσης στο αρχείο lab4results. txt.

Παραδοτέα

Ανεβάστε στο eCourse ένα zip αρχείο με τα τελικά προγράμματα σας BipartiteGraph.java και Queue.java, τα έτοιμα προγράμματα Collection.java και In.java, καθώς και με το αρχείο των αποτελεσμάτων lab4results. txt. Το zip αρχείο πρέπει να έχει το όνομα ενός μέλους της ομάδας σας με λατινικούς χαρακτήρες.