Graf Programming

Building Teams

Feladat:

Uolevi osztályában **n diák** van, és közöttük **m barátság** áll fenn. A feladat az, hogy a diákokat **két csapatra osszuk** úgy, hogy két barát ne kerüljenek ugyanabba a csapatba. A csapatok méretét szabadon megválaszthatjuk.

Bemenet:

- Az első sor tartalmazza az **n** (diákok száma) és **m** (barátságok száma) egész számokat.
- A következő **m** sorban két szám (**a**, **b**) szerepel, ami azt jelzi, hogy az **a** és **b** diákok barátok.

Kimenet:

- Ha lehetséges a diákokat két csapatra osztani, akkor minden diákhoz ki kell írni, hogy melyik csapatba tartozik: 1 vagy 2.
- Ha nem lehetséges a diákokat két csapatra osztani, akkor a kimenet: IMPOSSIBLE.

- Input:
- 5 3
- 1 2
- 1 3
- 4 5

5 diák van (1, 2, 3, 4, 5).
3 barátságot ismerünk:
Az 1-es és 2-es diák barátok.
Az 1-es és 3-as diák barátok.
A 4-es és 5-ös diák barátok.

- Output:
- 1 2 2 1 2

Az 1-es diákot az **1-es csapatba** tesszük.
Az 1-es diák barátait (2-est és 3-ast) a **2-es csapatba** tesszük.
A 4-es diákot az **1-es csapatba**, az 5-öst a **2-es csapatba** tesszük.
Így két csapatot kapunk, ahol nincs két barát ugyanabban a csapatban.

Programkód:

```java
import java.util.*;

public class BuildingTeams {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Bemenet beolvasása
        int n = sc.nextInt(); // Diákok száma
        int m = sc.nextInt(); // Barátságok száma
        List<List<Integer>> graf = new ArrayList<>();
        for (int i = 0; i <= n; i++) {
            graf.add(new ArrayList<>());
        }

        // Gráf építése
        for (int i = 0; i < m; i++) {
            int a = sc.nextInt();
            int b = sc.nextInt();
            graf.get(a).add(b);
            graf.get(b).add(a);
        }

        // Csapatok inicializálása (0: nincs csapatban, 1: első csapat, 2: második csapat)
        int[] csapat = new int[n + 1];

        // BFS függvény a csapatok szétosztására
        boolean lehetseges = true;

        for (int i = 1; i <= n; i++) {
            if (csapat[i] == 0) { // Ha még nincs csapatban
                if (!bfs(i, graf, csapat)) {
                    lehetseges = false;
                    break;
                }
            }
        }

        // Eredmény kiírása
        if (!lehetseges) {
            System.out.println("IMPOSSIBLE");
        } else {
            for (int i = 1; i <= n; i++) {
                System.out.print(csapat[i] + " ");
            }
        }
    }

    // BFS széleségi keresés megvalósítása
    private static boolean bfs(int kezd, List<List<Integer>> graf, int[] csapat) {
        Queue<Integer> queue = new LinkedList<>();
        queue.add(kezd);
        csapat[kezd] = 1; // Kezdjük az első csapattal

        while (!queue.isEmpty()) {
            int jelenlegi = queue.poll();
            int jelenlegiCsapat = csapat[jelenlegi];

            for (int szomszed : graf.get(jelenlegi)) {
                if (csapat[szomszed] == 0) {
                    csapat[szomszed] = 3 - jelenlegiCsapat; // Atvaltas (1 -> 2, 2 -> 1)
                    queue.add(szomszed);
                } else if (csapat[szomszed] == jelenlegiCsapat) {
                    return false; // Konfliktus: ugyanabban a csapatban van két barát
                }
            }
        }

        return true;
    }
}
```

Részletesen:

```java
// Bemenet beolvasása
int n = sc.nextInt(); // Diákok száma
int m = sc.nextInt(); // Barátságok száma
List<List<Integer>> graf = new ArrayList<>();
for (int i = 0; i <= n; i++) {
    graf.add(new ArrayList<>());
}
```

```
graf = [
    [],     // Index 0 - nem használjuk
    [],     // Index 1 - az 1-es diák szomszédainak listája
    [],     // Index 2 - a 2-es diák szomszédainak listája
    [],     // Index 3 - a 3-as diák szomszédainak listája
    [],     // Index 4 - a 4-es diák szomszédainak listája
    []      // Index 5 - az 5-ös diák szomszédainak listája
]
```

```java
// Gráf építése
for (int i = 0; i < m; i++) {
    int a = sc.nextInt();
    int b = sc.nextInt();
    graf.get(a).add(b);
    graf.get(b).add(a);
}
```

```
graf = [
    [],     // Index 0 - Nem használjuk (diákok 1-től számozottak)
    [2, 3], // Index 1 - Az 1-es diák barátai: 2 és 3
    [1],    // Index 2 - A 2-es diák barátja: 1
    [1],    // Index 3 - A 3-as diák barátja: 1
    [5],    // Index 4 - A 4-es diák barátja: 5
    [4]     // Index 5 - Az 5-ös diák barátja: 4
]
```

```java
// BFS széleségi keresés megvalósítása
private static boolean bfs(int kezd, List<List<Integer>> graf, int[] csapat) {
    Queue<Integer> queue = new LinkedList<>();
    queue.add(kezd);
    csapat[kezd] = 1; // Kezdjük az első csapattal

    while (!queue.isEmpty()) {
        int jelenlegi = queue.poll();
        int jelenlegiCsapat = csapat[jelenlegi];

        for (int szomszed : graf.get(jelenlegi)) {
            if (csapat[szomszed] == 0) {
                csapat[szomszed] = 3 - jelenlegiCsapat; // Atvaltas (1 -> 2, 2 -> 1)
                queue.add(szomszed);
            } else if (csapat[szomszed] == jelenlegiCsapat) {
                return false; // Konfliktus: ugyanabban a csapatban van két barát
            }
        }
    }

    return true;
```

Ez a bfs (Breadth-First Search) függvény egy gráfban végzett szélességi keresést valósít meg, hogy eldöntse, a gráf két csapatba osztható-e úgy, hogy két barát ne kerüljön ugyanabba a csapatba.

Kezdés: A kezd csúcsot az 1-es csapatba teszi, és elindítja a BFS-t.
Szélességi keresés: Minden feldolgozott csúcs szomszédait (barátait) az ellenkező csapatba helyezi:
Ha egy szomszéd még nincs csapatban, hozzáadja a megfelelő csapathoz.
Ha egy szomszéd már ugyanabban a csapatban van, konfliktust észlel, és false-t ad vissza.

Sikeres bejárás: Ha nincs konfliktus, a függvény true-val tér vissza, jelezve, hogy a gráf részben két csapatba osztható.