Round Trip

A program célja:

Ez a program egy **gráf** ciklusait keresi, és ha talál egyet, akkor visszaépíti és kiírja a ciklus városait. Ha nem talál ciklust, akkor **"IMPOSSIBLE"** üzenetet ír ki.

Program lépései:

1. **Bemeneti adatok beolvasása:**
    - A program a n számú csúcsot (városokat) és az m számú élt (utakat) várja bemenetként.
    - A gráf egy **szomszédsági listát** használ, hogy tárolja az összes város és azok közvetlen szomszédai közötti kapcsolatokat.
2. **DFS (Mélységi keresés) alkalmazása a gráf bejárásához:**
    - A program **DFS algoritmust** használ a gráf bejárásához.
    - A latogat tömb nyilvántartja, hogy mely városokat látogattunk már meg.
    - Az os tömb tárolja minden város előző városát (szülőjét), hogy megakadályozza az önciklusok (visszafelé irányuló élek) téves felismerését.
    - A ut lista a keresési verem, amely a mélységi keresés aktuális útvonalát tartalmazza.
3. **Ciklus keresése:**
    - Ha egy olyan szomszédos városra találunk, amely már látogatott (és nem a szülő), akkor **ciklust találtunk**.
    - A ciklus visszaépítése: A program végigmegy a ut listán, és visszafelé rekonstruálja a ciklust.
4. **Ciklus kiírása:**
    - A megtalált ciklust kiírjuk a képernyőre.
    - Ha nincs ciklus, akkor a program **"IMPOSSIBLE"** üzenetet ír ki.

Részletes magyarázat lépésről lépésre:

1. Bemenet beolvasása és szomszédsági lista létrehozása:

A program először beolvassa a városok és utak számát:

python

Kód másolása

```
n, m = map(int, input().split())
```

Ezután létrehozza a szomszédsági listát, amelyben minden városhoz hozzáadja annak közvetlen szomszédjait:

```
adj = [[] for _ in range(n + 1)]  # Szomszédsági lista, +1, mert 1-indexelt a gráf
for _ in range(m):
    a, b = map(int, input().split())
    adj[a].append(b)
    adj[b].append(a)
```

Rajz a szomszédsági lista alapján:

Tegyük fel, hogy az alábbi gráfot kapjuk (ahol a városokat számokkal jelöljük):

```
1 -- 2
|    |
3 -- 4
```

A szomszédsági lista így néz ki:

1: [2, 3]

2: [1, 4]

3: [1, 4]

4: [2, 3]

2. DFS keresés (Mélységi keresés):

A DFS algoritmus célja, hogy végig járja a gráf összes városát, és megpróbálja megtalálni a ciklust. A következő részlet tartalmazza a DFS-t:

```
def dfs(v):
    latogat[v] = True
    ut.append(v)

    for szomszed in adj[v]:
        if not latogat[szomszed]:
            os[szomszed] = v
            if dfs(szomszed):
                return True
        elif szomszed != os[v]:  # Ha találunk egy visszafelé irányuló élt (ciklus)
```

```
        kor = []
        kor.append(szomszed)
        idx = len(ut) - 1
        while ut[idx] != szomszed:
            kor.append(ut[idx])
            idx -= 1
        kor.append(szomszed)
        kor.reverse()
        print(len(kor))
        print(" ".join(map(str, kor)))
        return True
    ut.pop()
    return False
```

- A latogat tömb segít megakadályozni, hogy ugyanazt a várost többször is felkeressük.
- Az os tömb tárolja, hogy a városokat milyen sorrendben látogattuk meg, hogy ne térjünk vissza az előző városba véletlenül.
- A ut lista a keresési verem, amely végig követi a DFS folyamatát.

3. Ciklus felismerése:

A ciklust akkor találjuk meg, amikor a DFS visszafelé érkezik egy olyan szomszédhoz, amely már szerepel a keresési veremben (ut listában), de nem az aktuális szülőváros.

```
elif szomszed != os[v]:  # Visszafelé élt találunk
    kor = []
    kor.append(szomszed)
    idx = len(ut) - 1
    while ut[idx] != szomszed:
        kor.append(ut[idx])
        idx -= 1
    kor.append(szomszed)
    kor.reverse()
    print(len(kor))
```

print(" ".join(map(str, kor)))

return True

Rajz a ciklus felismerésére:

Ha egy ilyen utat találunk:

1 -- 2 -- 3 -- 4

|____|

A program azt látja, hogy amikor a 4-es városból visszafelé eljutunk a 3-as városhoz, egy ciklus található.

4. Ciklus visszaépítése:

Amikor megtalálunk egy ciklust, az ut listát használjuk a ciklus városainak visszaépítésére. A ciklus a következőképpen épül vissza:

1. Kezdjük a visszafelé irányuló él szomszédjával.
2. Addig megyünk vissza a veremben, amíg el nem érjük a ciklus kezdő városát.
3. Miután megtaláltuk a ciklus összes városát, visszafordítjuk őket, hogy a helyes sorrendben jelenjenek meg.

Rajz a ciklus visszaépítésére:

Visszaépítendő ciklus:

3 -- 4 -- 2 -- 3

5. Kimenet:

Ha sikerült ciklust találni, kiírjuk annak hosszát és városait:

print(len(kor))

print(" ".join(map(str, kor)))

***Példa kimenet*:**

4

3 4 2 3

Ha nincs ciklus, akkor:

print("IMPOSSIBLE")

Rajz a kimenet példájához:

Ciklus: 3 -> 4 -> 2 -> 3

Összegzés:

1. **Szomszédsági lista létrehozása**: A városok közötti kapcsolatok nyilvántartása.
2. **DFS algoritmus alkalmazása**: A mélységi keresés végigjárja a gráfot.
3. **Ciklus felismerése**: Ha egy visszafelé irányuló él található, akkor ciklus van.
4. **Ciklus visszaépítése**: A ciklust a keresési verem alapján visszaépítjük.
5. **Kimenet**: Kiírjuk a ciklust, ha létezik, vagy "IMPOSSIBLE" üzenetet, ha nem találtunk ciklust.

Ez a program kiválóan demonstrálja a gráfok bejárását és a ciklusok keresését. A diákok számára az ilyen típusú programok segítenek megérteni az algoritmusok működését és alkalmazását gráfokkal kapcsolatos problémák megoldásában. Tantárgyi koncentráció-matematika.