Structuri de Date
Tema 3 - Grafuri

Responsabil de temă:
Florin Dumitrescu
E-mail: florin.dumitrescu997@gmail.com
Teams: florin.dumitrescu97@upb.ro (ID: 78424)

Deadline: 22.05.2022

1 Introducere

Gică, patron al unei firme de logistică nou înființate, a reușit să semneze un contract cu un mare lanț de magazine. Printre atribuțiile sale se află aprovizionarea fiecărui magazin din lanț de la depozitele acestora, cât și livrarea de produse către clienți. Pentru a reduce timpul petrecut de fiecare șofer să își planifice ruta, Gică a apelat la tine pentru a-l ajuta la dezvoltarea unui algoritm care să automatizeze acest lucru.

1.1 Descriere problemă

- Magazinele/Clienții și depozitele sunt reprezentate prin noduri într-un graf orientat ponderat;
- Fiecare nod va avea ca ID un număr natural. Într-un graf ce conține **n** noduri, acestea vor avea asignate ID-uri în intervalul **[0, n - 1]**;
- Costul parcurgerii drumului dintre 2 locații (noduri) adiacente (care au o legătură între ele) va fi întotdeauna o valoare reală, pozitivă și nenulă cu o precizie de o zecimală;

1.2 Formatul datelor de intrare

Se vor citi de la **stdin**:

- Prima linie va conține numerele naturale **n**, **m** și **d** ce reprezintă numărul de noduri, numărul de arce și numărul de depozite din graf ;
- **m** linii ce reprezintă arcele dintre noduri. Acestea vor avea formatul **u v w**, unde **u** și **v** sunt ID-urile nodurilor și **w** este costul parcurgerii drumului dintre acestea;
- O linie ce conține ID-urile nodurilor ce reprezintă depozite;
- O linie ce reprezintă numărul de cerințe ce trebuiesc rezolvate;
- Liniile ce vor urma sunt specifice fiecărei cerințe și vor începe mereu cu ID-ul task-ului ce trebuie rezolvat.

De exemplu, pentru input-ul de mai jos, se va obține graful din figura 1.

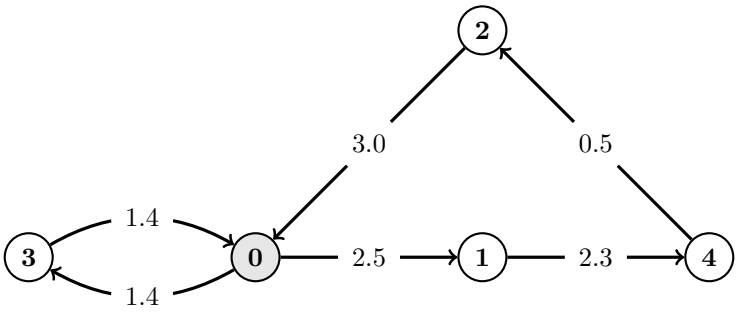

Figure 1

```
5 6 1
0 1 2.5
0 3 1.4
1 4 2.3
2 0 3.0
3 0 1.4
4 2 0.5
0
```

2 Cerințe

2.1 Cerința 1

Pentru a transporta marfa de la depozit către un magazin, este nevoie să se ocupe întreaga capacitate a mașinii. Acest lucru înseamnă că după fiecare aprovizionare a unui magazin, șoferul trebuie să se întoarcă la depozit pentru a prelua marfa pentru următorul transport. Să se determine traseul cel mai scurt de la depozit către fiecare magazin care trebuie aprovizionat.

Specificații

- Există un singur depozit de la care se ridică marfa (chiar dacă în graf există mai multe depozite, ridicarea de marfă se va face doar din depozitul specificat)
- Traseul se va începe și încheia întotdeauna la depozitul specificat
- Există o singură mașină care transportă marfa
- Se vor aproviziona doar magazinele care sunt specificate

Formatul datelor de intrare

După ce s-a construit graful, se vor citi de la **stdin** pe câte o linie:

- ID-ul task-ului: **e1**
- Depozitul din care se încarcă marfa: **s**
- Numărul de magazine ce vor fi aprovizionate: **k**
- ID-urile magazinelor ce vor fi aprovizionate: $\{x_0, x_1, \ldots, x_k\}$, în ordine crescătoare.

Formatul datelor de ieșire

Se va scrie la **stdout**, pe câte o linie:

- Pentru fiecare magazin, în ordinea în care s-au primit ID-urile acestora:
  - ID-ul magazinului
  - Costul drumului parcurs de la depozit până la magazin și cel de la magazin înapoi la depozit
  - ID-urile nodurilor în ordinea parcurgerii, incluzând nodul de start și destinația
- Costul total al traseului

Exemplu

Input:

```
5 6 1
0 1 2.5
0 3 1.4
1 4 2.3
2 0 3.0
3 0 1.4
4 2 0.5
0
1
e1
0
2
3 4
```

Output:

```
3
1.4 1.4
0 3 0
4
4.8 3.5
0 1 4 2 0
11.1
```

Explicație:

- Graful construit este cel din figura 1
- Depozitul din care se va încărca marfa este cel din nodul cu ID-ul **0**
- Pe lista de livrări există 2 magazine, acestea având ID-urile **3** și **4**
- Se începe cu magazinul cu ID-ul **3**. Există un drum bidirecțional direct între depozit și magazin, deci traseul ce trebuie parcurs este **0 3 0** (depozit → magazin → depozit). Costul parcurgerii acestui drum este de 1.4 și este identic în ambele sensuri.
- Fiind înapoi la depozit, se poate continua procesul cu următorul magazin din listă. De această dată nu mai avem o legătură directă, ceea ce înseamnă că trebuie parcurs graful până când se găsește traseul cu cel mai mic cost către nodul destinație. În acest caz, drumul parcurs este **0 1 4** cu un cost de 4.8. Se repetă același proces de la nodul **4** până la depozit, găsindu-se drumul **4 2 0** cu un cost de 3.5. Concatenând drumul de la depozit la magazin cu cel de la magazin la depozit, se obține drumul total parcurs **0 1 4 2 0**.
- După ce toate livrările au fost făcute, se calculează costul total al traseului **0 3 0 1 4 2 0**, acesta fiind suma costurilor tuturor drumurilor parcurse, adică $1.4 + 1.4 + 2.5 + 2.3 + 0.5 + 3.0 = 11.1$

2.2 Cerința 2

Spre deosebire de marfa transportată pentru aprovizionarea magazinelor, comenzile clienților nu sunt la fel de voluminoase, un singur șofer putând livra mai multe comenzi odată. De asemenea, deoarece nu sunt necesare mașini speciale și se dorește ca produsele să fie livrate cât mai repede, mai mulți curieri pot fi trimiși pe traseu în aceeași zi. Să se determine câți curieri ar trebui să livreze pachete și în ce zone. Altfel spus, să se determine componentele tare conexe (*strongly connected components*) din graful dat.

Specificații

- Nodurile de tip depozit nu vor face parte din componentele tare conexe
- Nu este relevant costul de livrare
- Pot fi trimiși pe traseu mai mulți curieri
- O componentă conexă a unui graf este un sub-graf în care există o cale între oricare 2 noduri. Denumirea de componentă conexă este asociată grafurilor neorientate. În cazul grafurilor orientate, acestea se numesc componente tare conexe.

Formatul datelor de intrare

După ce s-a construit graful, se vor citi de la **stdin** pe câte o linie:

- ID-ul task-ului: **e2**

Formatul datelor de ieșire

Se va scrie la **stdout**, pe câte o linie:

- Numărul de componente tare conexe găsite
- Pe câte o linie, pentru fiecare componentă tare conexă, nodurile care fac parte din aceasta în ordine crescătoare după ID. Componentele vor fi afișate în ordine crescătoare după ID-ul primului nod.

Exemplu

Input:

```
9 15 1
0 1 8.0
0 3 5.0
0 8 8.0
1 0 1.0
1 2 3.0
2 4 6.0
2 5 2.0
3 6 8.0
4 3 4.0
4 7 3.0
5 1 9.0
6 0 6.0
6 4 4.0
7 6 9.0
8 0 8.0
0
1
e2
```

Output:

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

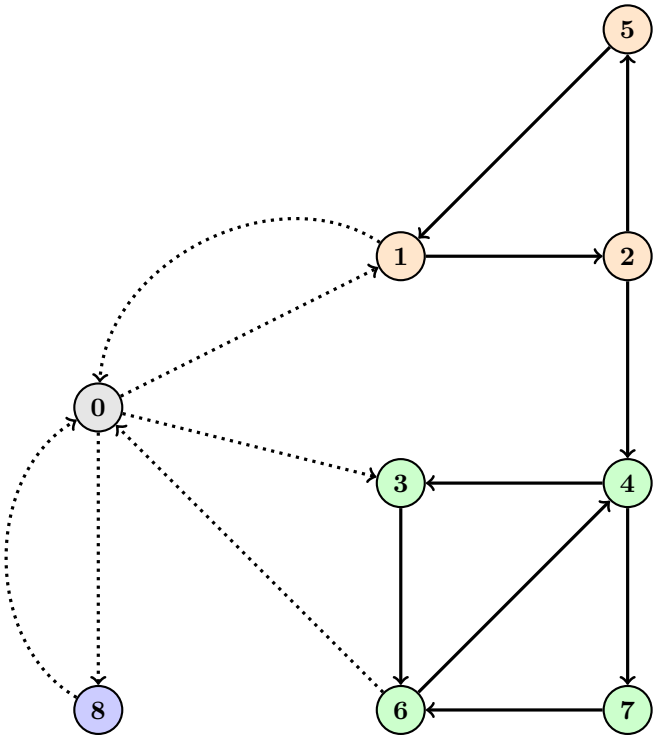

Figure 2

Explicație:

- Graful construit este cel din figura 2. Pentru simplitate, nu s-au mai trecut ponderile, iar toate legăturile ce conectau nodul depozit de restul grafului sunt desenate cu linie punctată pentru a indica faptul că acesta nu este inclus în nicio componentă
- Nodul 8 este o componentă conexă, deoarece orice sub-graf format din mai puțin de 2 noduri este implicit conex
- Sub-graful **1 2 5** este un ciclu elementar, ceea ce înseamnă cu este conex.
- Sub-graful **3 4 6 7** este format din 2 cicluri și există conexiuni între oricare 2 noduri. De la nodul **4** se poate ajunge la nodul **6** atât prin nodul **3**, cât și prin **4**, iar legătura de la nodul **6** înapoi la nodul **4** asigură conexitatea acestui sub-graf.
- Deși există o conexiune de la nodul **2** la nodul **4**, nu există nicio legătură de la un nod din sub-graful **3 4 6 7** la cel puțin un nod din sub-graful **1 2 5**.
- Avem astfel 3 componente tare conexe pe care le vom afișa în ordine crescătoare după ID-ul primului nod.

2.3 Cerința 3

Odată ce au fost stabilite zonele în care va face livrări fiecare curier, ar trebui să se calculeze și câte un traseu pe care aceștia să îl urmeze pentru a duce pachetele la destinație în cel mai eficient mod posibil.

Specificații

- Curierul va pleca și se va întoarce la același nod de tip depozit
- Se garantează că există conexiuni directe între nodul depozit și zona asociată curierului. Altfel spus, curierul nu va fi nevoit să traverseze zone asignate altor curieri pentru a ajunge la nodul de tip depozit
- Pot exista mai multe depozite care au legături cu aceeași zonă. Curierul poate trece printr-un alt nod depozit decât cel din care a plecat
- Curierul poate trece de mai multe ori prin același nod și poate folosi de mai multe ori același drum
- Pot exista mai multe trasee cu același cost de parcurgere

Formatul datelor de intrare

După ce s-a construit graful, se vor citi de la **stdin** pe câte o linie:

- ID-ul task-ului: **e3**
- Numărul de zone: **r**
- Pentru fiecare zonă se vor da numărul de noduri din aceasta, urmat de ID-ul nodurilor ce se află în componența sa: $\{k, x_0, x_1, \ldots, x_k\}$

Formatul datelor de ieșire

Se va scrie la **stdout**, pe câte o linie pentru fiecare zonă:

- Costul traseului

Exemplu

Input:

```
9 15 1
0 1 8.0
0 3 5.0
0 8 8.0
1 0 1.0
1 2 3.0
2 4 6.0
2 5 2.0
3 6 8.0
4 3 4.0
4 7 3.0
5 1 9.0
6 0 6.0
6 4 4.0
7 6 9.0
8 0 8.0
0
1
e3
1
4 3 4 6 7
```

Output:

```
35.0
```

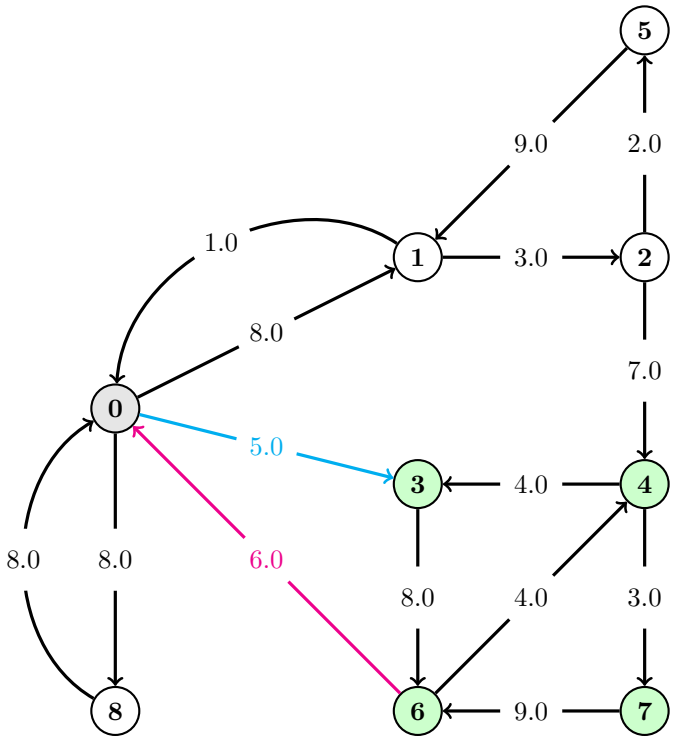

Figure 3

Explicație:

- Graful construit este cel din figura 3. Nodurile pe care curierul trebuie să le viziteze sunt colorate cu verde
- Plecând din nodul **0**, se poate ajunge în zonă folosind legătura cu nodul **3**
- Se determină că nodul prin care curierul se poate întoarce la depozit este **6**
- Se determină că cel mai scurt drum prin care se poate ajunge de la nodul **3** la nodul **6** parcurgând toate celelalte noduri din zone este **3 6 4 7 6** cu un cost de 24.0.
- Se adună la acest cost și costurile drumurilor de la depozit până în zonă și invers, traseul final fiind **0 3 6 4 7 6 0** cu un cost de 35.0

HINT:

Se poate folosi o variantă modificată a algoritmului lui Dijkstra care, pe lângă nodurile vizitate, va reține și o stare a traseului până în acel moment. Acest lucru este util deoarece:

- Dacă se face o parcurgere în lățime sau în adâncime, fără a reține nodurile vizitate, nu vor fi detectate ciclurile, algoritmul rulând la nesfârșit
- Dacă se rețin doar nodurile vizitate, algoritmul de parcurgere se va opri la primul nod deja vizitat

Starea poate fi codificată folosind o mască binară, fiecare bit din mască indicând dacă nodul acela a fost deja vizitat sau nu. De exemplu, nodul **0** ar fi codificat cu masca [0 0 ... 0 1], nodul **1** cu masca [0 0 ... 1 0] și așa mai departe.

Pentru exemplul din figura 4 vrem să pornim din nodul **0** și să ajungem în nodul **3** trecând prin nodurile **1** și **2**. Altfel spus, vrem să ajungem în nodul **3** cu starea [1 1 1 1]. Procesul este următorul:

- Vom marca drept vizitată starea inițială, care este perechea (**0**, [0 0 0 1]) și vom introduce în coada de vizitare nodul **3**, care este singurul la care se poate ajunge din nodul curent, împreună cu starea ce urmează: (**3**, [1 0 0 1])
- Extragem din coadă (**3**, [1 0 0 1]). Deși **3** este nodul la care vrem să ajungem, nu am traversat toate nodurile din graf, ceea ce înseamnă că vom continua explorarea. Se introduce în coadă nodul **1** și starea ce va urma: (**1**, [1 0 1 1]). Perechile vizitate până în acest moment sunt: [(**0**, [0 0 0 1]), (**3**, [1 0 0 1])]
- Extragem din coadă (**1**, [1 0 1 1]). Nu este destinația finală, deci continuăm explorarea. Din nodul **1** se poate ajunge la nodurile **0** și **2**. Vom introduce în coadă perechile (**0**, [1 0 1 1]) și (**2**, [1 1 1 1]). Perechile vizitate până în acest moment sunt: [(**0**, [0 0 0 1]), (**3**, [1 0 0 1]), (**1**, [1 0 1 1])]
- Extragem din coadă (**0**, [1 0 1 1]). Am ajuns iar în nodul 0, dar este posibil să mai existe soluții, deci vom continua explorarea. Se introduce în coadă perechea (**3**, [1 0 1 1]). Perechile vizitate până în acest moment sunt: [(**0**, [0 0 0 1]), (**3**, [1 0 0 1]), (**1**, [1 0 1 1]), (**0**, [1 0 1 1])]
- Extragem din coadă (**2**, [1 1 1 1]). Starea este cea așteptată, dar nu ne aflăm în nodul terminal, deci continuăm explorarea. Se introduce în coadă perechea (**3**, [1 1 1 1]). Perechile vizitate până în acest moment sunt: [(**0**, [0 0 0 1]), (**3**, [1 0 0 1]), (**1**, [1 0 1 1]), (**0**, [1 0 1 1]), (**2**, [1 1 1 1])]
- Extragem din coadă (**3**, [1 0 1 1]). Nu este destinația finală, deci continuăm explorarea. Se introduce în coadă perechea (**1**, [1 0 1 1]). Perechile vizitate până în acest moment sunt: [(**0**, [0 0 0 1]), (**3**, [1 0 0 1]), (**1**, [1 0 1 1]), (**0**, [1 0 1 1]), (**2**, [1 1 1 1]), (**3**, [1 0 1 1])]
- Extragem din coadă (**3**, [1 1 1 1]). Aceasta este o soluție, deci o vom salva. Perechile vizitate până în acest moment sunt: [(**0**, [0 0 0 1]), (**3**, [1 0 0 1]), (**1**, [1 0 1 1]), (**0**, [1 0 1 1]), (**2**, [1 1 1 1]), (**3**, [1 0 1 1])]. Soluții posibile: **0 3 1 2 3**.
- Extragem din coadă (**1**, [1 0 1 1]). Am mai vizitat odată această pereche, deci se abandonează traseul. Perechile vizitate până în acest moment sunt: [(**0**, [0 0 0 1]), (**3**, [1 0 0 1]), (**1**, [1 0 1 1]), (**0**, [1 0 1 1]), (**2**, [1 1 1 1]), (**3**, [1 0 1 1])]. Soluții posibile: **0 3 1 2 3**.
- Nu mai există elemente în coadă. Printre soluțiile posibile, se caută acelea ce au costul minim. În cazul de față, singura soluție este **0 3 1 2 3**.

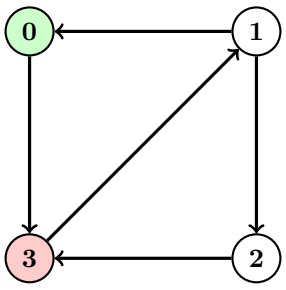

Figure 4

3 Precizări

- Temele vor fi încărcate pe vmchecker, **NU** se acceptă teme trimise pe e-mail sau altfel decât prin intermediul vmchecker-ului
- O rezolvare constă dintr-o arhivă de tip *zip* care conţine toate fişierele sursă alaturi de un ***Makefile***, ce va fi folosit pentru compilare, şi un fisier ***README***, în care se vor preciza detaliile implementării. Toate fișierele se vor afla în rădăcina arhivei
- Makefile-ul trebuie să aibă obligatoriu regulile pentru *build* şi *clean*. Regula build trebuie să aibă ca efect compilarea surselor şi crearea binarului *tema3*
- Programul vostru va citi de la **stdin** graful ce trebuie construit împreună cerințele ce vor fi rezolvate și va afișa la **stdout** rezultatele rulării
- Costurile traseelor se vor afișa ca numere reale cu precizia de **1** zecimală
- Graful va fi implementat folosind liste sau matrice de adiacență
- După fiecare test, toată memoria alocată trebuie eliberată pentru a obține punctajul
- Atenție la afișare, întrucât checker-ul va apela funcția **diff** pentru a compara output-ul programului vostru cu fișierul de referință aferent testului
  - Elementele de pe un rând sunt separate printr-un singur spațiu
  - Nu se va afișa spațiu la final de rând
  - Nu vor exista rânduri goale

4 Punctaj

Cerința	**Punctaj**
Cerința 1	25 p
Cerința 2	30 p
Cerința 3	35 p
Readme, Coding Style și warning-uri	10 p

Table 1: Caption

5 Grafuri

În cadrul temei sunt folosite următoarele 5 grafuri: **G0**, **G1**, **G2**, **G3**, **G4**.

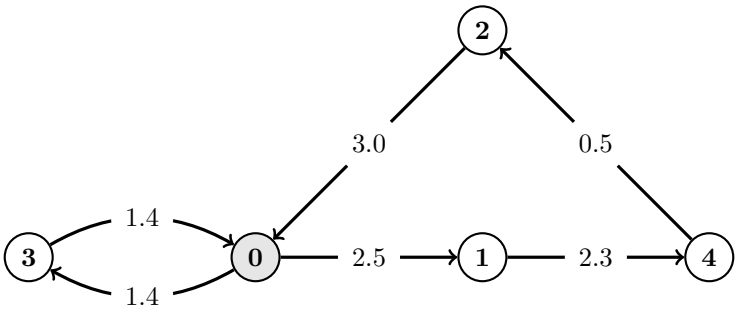

Figure 5: **Graful 0**

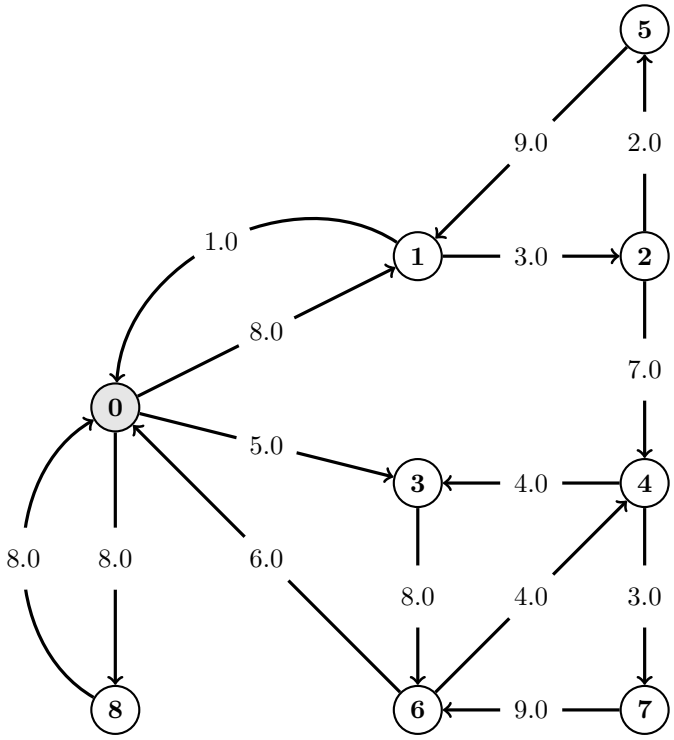

Figure 6: **Graful 1**

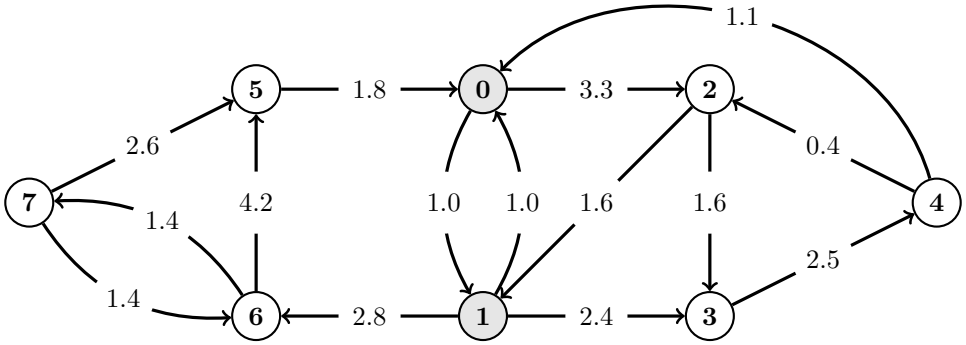

Figure 7: **Graful 2**

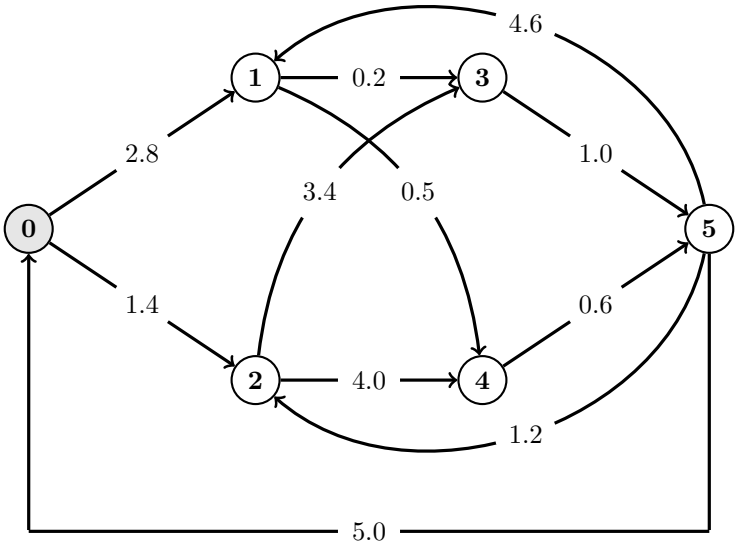

Figure 8: **Graful 3**

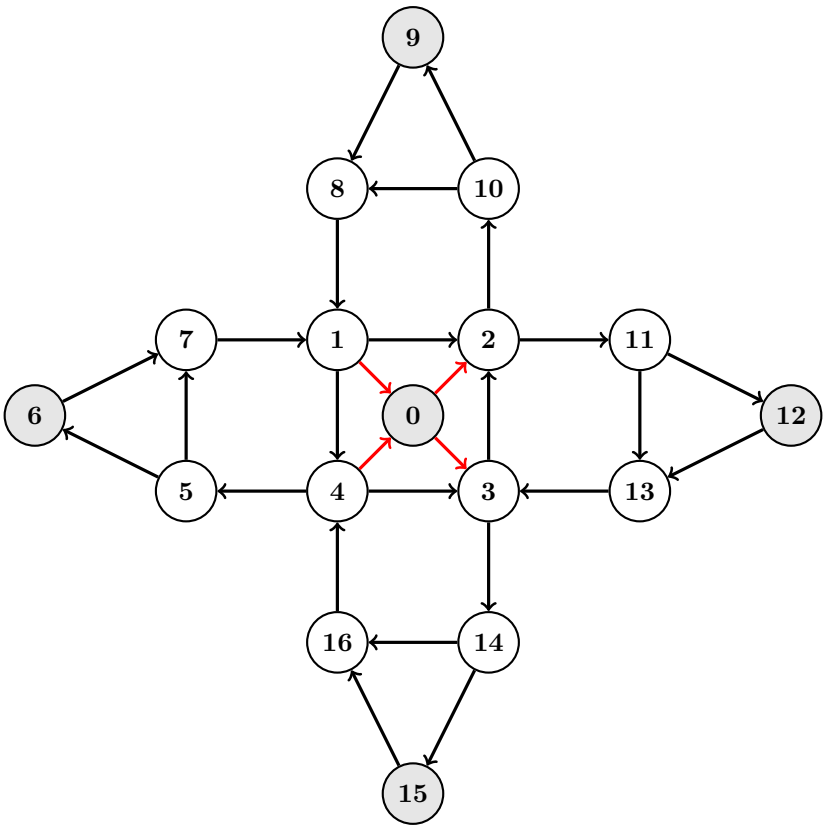

Figure 9: **Graful 4**
Legăturile roșii au o pondere de 0.1, iar cele negre de 1.0