Puncte de articulație

Capitolul 6

- **Considerații teoretice**
- **Un algoritm simplu**
- **Algoritmul eficient**
- **Rezumat**
- **Implementări sugerate**
- **Probleme propuse**
- **Soluțiile problemelor**

În cadrul acestui capitol ne vom ocupa de punctele de articulație ale unui graf. Vom defini acest concept, iar apoi vom descrie un algoritm simplu, dar ineficient, care poate fi utilizat pentru determinarea acestora. Ulterior, vom descrie algoritmul optim care poate fi utilizat pentru a determina astfel de puncte. De asemenea, vom analiza complexitatea algoritmilor descrişi.

6.1. Considerații teoretice

Această secțiune este dedicată prezentării conceptului de punct de articulație într-un graf neorientat. Vom presupune, fără a restrânge generalitatea, că grafurile luate în considerare sunt conexe. În cazul în care această condiție nu este respectată, algoritmii care vor fi descrişi pot fi aplicați pentru fiecare componentă conexă în parte.

6.1.1. Dispariția conexității

Grafurile conexe pot fi utilizate pentru a descrie o mulțime de sisteme care apar în viața reală. Câteva exemple în acest sens sunt o rețea de calculatoare, o rețea de străzi, un sistem de securitate etc. Uneori, aceste sisteme au puncte "nevralgice" sau vulnerabile, "căderea" cărora poate duce la întreruperea funcționării corecte. De exemplu, dacă într-o rețea de calculatoare se defectează server-ul principal, atunci este foarte posibil ca rețeaua să nu mai funcționeze corect (evident, depinde de modul în care este configurată rețeaua, dar situația este plauzibilă). Defectarea server-ului a dus la întreruperea sau limitarea posibilităților de comunicare. Foarte probabil, vor exista perechi de calculatoare care nu vor mai putea comunica. În termenii teoriei grafurilor, graful care modelează rețeaua şi-a pierdut conexitatea.

Există mai multe moduri prin care un graf îşi poate pierde conexitatea. Unul dintre acestea îl reprezintă eliminarea unui nod şi a tuturor muchiilor adiacente acestuia.

6.1.2. Puncte de articulație în grafuri conexe

Nodurile unui graf a căror eliminare duce la dispariția conexității poartă denumirea de ***noduri*** sau ***puncte de articulație***. Uneori, mai sunt utilizați şi termenii de *punct critic* sau *nod critic*.

În cazul în care graful nu este unul conex, punctele de articulație pot fi definite ca fiind noduri a căror eliminare duce le creşterea numărului componentelor conexe ale grafului.

6.2. Un algoritm simplu

Cea mai simplă modalitate de determinare a punctelor de articulație dintr-un graf conex este eliminarea (pe rând) a câte unui nod (împreună cu muchiile adiacente) şi verificarea conexității grafului rămas după eliminarea nodului respectiv.

6.2.1. Implementarea algoritmului

Versiunea în pseudocod a algoritmului descris anterior, este prezentată în continuare:

```
Subalgoritm Puncte_de_articulație(G,n):
                                                    { G – graful }
                                          { n – numărul nodurilor din graf }
  pentru i ← 1, n execută:
    dacă nu Conex(G\{i}) atunci
      scrie i
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

În cadrul subalgoritmului anterior prin `G\{i}` s-a notat graful obținut prin eliminarea nodului `i` din graful `G`. Se foloseşte un subalgoritm `Conex` care verifică dacă un anumit graf este sau nu conex.

6.2.2. Analiza complexității

Se observă că în cadrul algoritmului anterior fiecare nod este luat în considerare o singură dată. Nodurile sunt eliminate pe rând din graf, operație ce poate fi realizată în timp constant dacă se utilizează un algoritm particular de verificare a conexității (un algoritm asemănător cu cel clasic, dar care nu ia în considerare nodul eliminat).

Pentru fiecare nod eliminat se verifică dacă graful rămas este sau nu conex. Ordinul de complexitate al operației de verificare a conexității este **O(M)**.

Aşadar, ordinul de complexitate al acestui algoritm simplu de determinare a punctelor de articulație este **O(M · N)**.

6.3. Algoritmul eficient

În cadrul acestei secțiuni vom prezenta un algoritm cu ordinul de complexitate **O(M + N)** care poate fi utilizat pentru a determina punctele de articulație ale unui graf.

6.3.1. Preliminarii

Algoritmul pe care îl vom prezenta se bazează pe parcurgerea *DF* a grafurilor. Reamintim faptul că în arborele obținut printr-o parcurgere *DF* nu pot exista muchii de traversare, ci doar muchii de înaintare şi de întoarcere.

În aceste condiții, rădăcina arborelui *DF* va fi punct de articulație dacă şi numai dacă are cel puțin doi descendenți. Într-adevăr, prin eliminarea rădăcinii se va rupe conexitatea grafului deoarece nu există muchii care să treacă de la un subarbore al rădăcinii la altul.

Un vârf *i* al grafului, diferit de rădăcină, va fi punct de articulație dacă şi numai dacă are cel puțin un fiu *j* cu proprietatea că nu există nici un nod în arborele de rădăcină *j* care să fie legat printr-o muchie de întoarcere de un predecesor al vârfului *i*. Cu alte cuvinte, dacă din fiecare subarbore "se poate ajunge deasupra nodului considerat", atunci nodul respectiv nu este critic.

6.3.2. Prezentarea algoritmului

Pentru a verifica dacă dintr-un anumit nod se poate ajunge deasupra unui alt nod vom păstra, pentru fiecare nod în parte, nivelul pe care se află acesta în arborele *DF*.

Datorită faptului că, în cadrul parcurgerii *DF*, pentru fiecare nod va trebui să luăm în considerare toate muchiile care pornesc din nodul respectiv, cu excepția celei care îl uneşte de părintele său, va trebui să cunoaştem părinții nodurilor.

Pentru fiecare nod vom determina nivelul minim la care se poate ajunge parcurgând o muchie de întoarcere care pleacă din subarborele care are rădăcina în nodul respectiv. Pentru aceasta va trebui să păstrăm şi nivelurile nodurilor.

Varianta în pseudocod a algoritmului este:

```
Subalgoritm Puncte_Critice(k,părinte,G,niv,niv_min):
                                                    { k – nodul curent }
                                          { părinte – părintele nodului curent }
                                                { G – graful,  niv – nivelul curent }
                                  { niv_min – nivelul minim la care se poate ajunge parcurgând }
                                          { o muchie care pleacă de la un nod din subarborele de }
                                                { rădăcină k; valoare transmisă prin referință }
  dacă vizitat_k atunci
    val_min ← nivel_k
  altfel
    nrfii_părinte ← nrfii_părinte + 1
```

```
    vizitat_k ← adevărat
    nivel_k ← niv
    val_min ← niv
    pentru toți vecinii i ai lui k execută
      dacă i ≠ părinte atunci
        Puncte_Critice(i,nod,G,niv+1,aux)
      sfârşit dacă
      dacă aux < val_min atunci
        val_min ← aux
      sfârşit dacă
      dacă aux ≥ niv atunci
        scrie nod
      sfârşit dacă
    sfârşit pentru
  sfârşit dacă
sfârşit subalgoritm

Algoritm Determinare_Puncte_Critice(G)
  PuncteCritice(1,-1,G,1,aux)
  dacă nrfii_1 > 1 atunci
    scrie 1
  sfârşit dacă
sfârşit algoritm
```

În cadrul algoritmului anterior se consideră nodul `1` ca fiind rădăcină a arborelui *DF*. Prin apelul subalgoritmului `Puncte_Critice` se determină toate nodurile critice fără a fi luat în considerare şi nodul `1`. În urma apelului se verifică numărul fiilor nodului `1` şi dacă numărul acestora este mai mare decât `1` acesta este afişat ca fiind punct de articulație.

6.3.2. Analiza complexității

Datorită faptului că algoritmul constă într-o variantă modificată a unei parcurgeri în adâncime a unui graf, ordinul său de complexitate este $O(M + N)$; aşadar, acest algoritm este mult mai rapid decât cel care constă în eliminarea succesivă a nodurilor.

6.4. Rezumat

În cadrul acestui capitol am prezentat noțiunea de punct de articulație, precum şi doi algoritmi care pot fi utilizați pentru a determina punctele de articulație ale unui graf neorientat conex.

De asemenea, am arătat motivele pentru care algoritmul bazat pe parcurgerea *DF* este mai performant decât cel simplu, bazat pe simpla eliminare a nodurilor.

6.5. Implementări sugerate

Pentru a vă familiariza cu modul în care trebuie implementate rezolvările problemelor în care apar punctele de articulație, vă sugerăm să încercați să implementați algoritmi pentru:

1. verificarea eficientă a conexității unui graf din care a fost eliminat un nod;
2. identificarea punctelor de articulație dintr-un graf neorientat conex prin eliminări succesive ale nodurilor;
3. identificarea punctelor de articulație dintr-un graf neorientat neconex prin eliminări succesive ale nodurilor;
4. identificarea punctelor de articulație dintr-un graf neorientat conex prin intermediul unui parcurgeri în adâncime;
5. identificarea punctelor de articulație dintr-un graf neorientat neconex prin intermediul unui parcurgeri în adâncime.

6.6. Probleme propuse

În continuare vom prezenta enunțurile câtorva probleme pe care vi le propunem spre rezolvare. Toate aceste probleme pot fi rezolvate folosind informațiile prezentate în cadrul acestui capitol. Cunoştințele suplimentare necesare sunt minime.

6.6.1. Grevă

Descrierea problemei

Într-un județ sunt N localități legate între ele printr-un număr total de M şosele pe care se poate circula în ambele sensuri. Rețeaua de şosele este construită în aşa fel încât să se poată circula între oricare două localități. Din nefericire, toți locuitorii județului doresc să participe la o grevă generală. Până la urmă s-a decis că se permite participarea la grevă a locuitorilor dintr-o singură localitate. Prefectul dorește să aleagă localitatea astfel încât să se poată circula în continuare între oricare două dintre celelalte localități, eventual folosindu-se alte trasee. Va trebui să determinați care sunt opțiunile prefectului. Evident, prin localitatea celor care participă la grevă nu se va mai putea circula.

Date de intrare

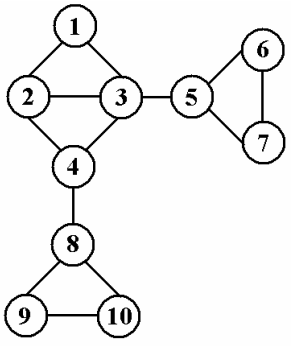
Prima linie a fişierului de intrare `GREVA.IN` conține numărul N al localităților din județ şi numărul M al şoselelor directe dintre localități. Aceste numere vor fi separate printr-un spațiu. Fiecare dintre următoarele M linii va conține câte două numere întregi x şi y cu semnificația: există o şosea care leagă direct localitățile identificate prin x şi y.

Date de ieşire
Prima linie a fişierului de ieşire **GREVA.OUT** va conține numărul K al localităților în care ar putea fi permisă participarea la greva generală. Pe următoarea linie se vor afla K numere întregi care reprezintă numerele de ordine ale acestor localități. Aceste numere vor fi separate printr-un spațiu.

Restricții şi precizări
- $1 \le N \le 100$;
- $1 \le M \le 1000$;
- localitățile sunt identificate prin numere întregi cuprinse între 1 şi N;
- numerele de ordine ale localităților pot fi scrise în fişierul de ieşire în orice ordine;
- există cel mult o şosea între oricare două localități.

Exemplu

GREVA.IN	GREVA.OUT
10 13	6
1 2	1 2 6 7 9 10
1 3	
2 3	
2 4	
3 4	
3 5	
4 8	
5 6	
5 7	
6 7	
8 9	
8 10	
9 10	



Timp de execuție: 1 secundă/test

6.6.2. Atac

Descrierea problemei
O companie deține o rețea formată din N calculatoare. Între acestea există un număr total de M legături directe astfel încât este posibilă comunicarea directă sau indirectă între oricare două calculatoare. Din surse demne de încredere se ştie că în scurt timp un hacker va ataca un calculator, dar nu se ştie exact pe care dintre acestea. Din fericire, hacker-ul nu cunoaşte punctele vulnerabile ale rețelei de calculatoare. Din aceste motive, probabilitea ca el să aleagă un anumit calculator este aceeaşi pentru fiecare calculator. După atac, calculatorul respectiv nu va mai putea fi folosit pentru comunicare.

Directorul companiei doreşte să cunoască probabilitatea ca reţeaua să nu mai funcţioneze corect după atacul inevitabil al hacker-ului. Reţeaua nu va mai funcţiona corect dacă va exista cel puţin o pereche de calculatoare care nu vor mai putea comunica.

Date de intrare
Prima linie a fişierului de intrare **ATAC.IN** conţine numărul N al calculatoarelor din reţea şi numărul M al legăturilor directe dintre calculatoare. Aceste numere vor fi separate printr-un spaţiu. Fiecare dintre următoarele M linii va conţine câte două numere întregi x şi y cu semnificaţia: există o legătură directă între calculatoarele identificate prin x şi y.

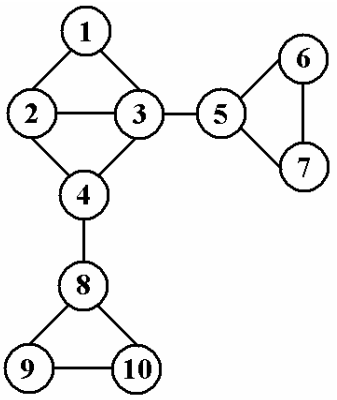
Date de ieşire
Fişierul de ieşire **ATAC.OUT** va conţine o singură linie pe care se va afla un singur număr care reprezintă probabilitatea ca reţeaua să nu mai funcţioneze corect după atacul hacker-ului. Probabilitatea va fi exprimată în procente şi numărul va fi scris cu două zecimale exacte.

Restricţii şi precizări
- $1 \le N \le 100$;
- $1 \le M \le 1000$;
- calculatoarele sunt identificate prin numere întregi cuprinse între 1 şi N;
- există cel mult o legătură directă între oricare două calculatoare.

Exemplu

ATAC.IN
```
10 13
1 2
1 3
2 3
2 4
3 4
3 5
4 8
5 6
5 7
6 7
8 9
8 10
9 10
```

ATAC.OUT
```
40.00
```



Timp de execuţie: 1 secundă/test

6.6.3. Neutroni

Un cristal este format din mai mulți ioni uniți prin legături cristaline. Se consideră un număr de N ioni şi un număr de M legături cristaline. Fiecare legătură este între doi ioni. Vom spune că un ion face parte dintr-un cristal dacă există o legătură cristalină între ionul respectiv şi un alt ion care face parte din cristalul respectiv.

La un moment dat, cristalele sunt bombardate cu un flux de neutroni. În momentul bombardării cristalelor este posibil ca unii ioni să fie eliminați din cristal. Să se determine numărul cristalelor şi ionii care, dacă părăsesc structura cristalină, cristalul care conține ionul respectiv se sparge.

Date de intrare
Prima linie a fişierului de intrare **NEUTRONI.IN** conține numărul N al ionilor din rețea şi numărul M al legăturilor cristaline dintre aceştia. Aceste numere vor fi separate printr-un spațiu. Fiecare dintre următoarele M linii va conține câte două numere întregi x şi y cu semnificația: există o legătură cristalină între ionii identificați prin x şi y.

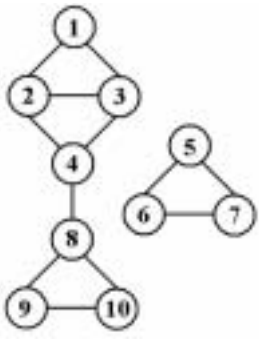
Date de ieşire
Prima linie a fişierului de ieşire **NEUTRONI.OUT** va conține numărul C al cristalelor şi numărul K al ionilor a căror eliminare duce la spargerea unui cristal. Cele două numere vor fi separate printr-un spațiu. Fiecare dintre următoarele K linii corespunde unui astfel de ion; pe o astfel de linie va fi scris numărul de ordine ale ionului respectiv.

Restricții şi precizări
- $1 \le N \le 100$;
- $1 \le M \le 1000$;
- ionii sunt identificați prin numere întregi cuprinse între 1 şi N;
- există posibilitatea ca un cristal să fie format dintr-un singur ion;
- există cel mult o legătură cristalină între oricare doi ioni.

Exemplu

NEUTRONI.IN
```
10 12
1 2
1 3
2 3
2 4
3 4
4 8
5 6
5 7
6 7
```

NEUTRONI.OUT
```
2 2
4 8
```



```
8 9
8 10
9 10
```

Timp de execuție: 1 secundă/test

6.7. Soluțiile problemelor

Vom prezenta acum soluțiile problemelor propuse în cadrul secțiunii precedente. Pentru fiecare dintre acestea va fi descrisă metoda de rezolvare şi va fi analizată complexitatea algoritmului prezentat.

6.7.1. Grevă

Rețeaua de şosele din județ poate fi privită ca fiind un graf neorientat în care localitățile sunt reprezentate de noduri, iar şosele dintre localități de muchii. În aceste condiții problema se reduce la determinarea tuturor punctelor necritice ale acestui graf.

Evident, vom determina nodurile critice ale grafului şi vom scrie în fişierul de ieşire numerele de identificare ale tuturor nodurilor care nu sunt critice.

Vom utiliza algoritmul de determinare a nodurilor critice şi, în momentul detectării unui astfel de nod, vom marca acest nod ca fiind critic.

În final, vom număra nodurile necritice şi vom scrie numerele de ordine ale nodurilor necritice.

Analiza complexității

Citirea datelor de intrare implică citirea celor M muchii ale grafului, aşadar ordinul de complexitate al acestui algoritm este **O(M)**. În paralel cu citirea se realizează crearea structurii de date în care se memorează graful, ordinul de complexitate al acestei operații fiind tot **O(M)**.

Algoritmul de determinare a nodurilor critice ale unii graf are ordinul de complexitate **O(M + N)**.

Datorită faptului că un graf poate conține cel mult N noduri critice, ordinul de complexitate al operației de scriere a datelor de ieşire are ordinul de complexitate **O(N)**.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(M) + O(M) + O(M + N) + O(N) =$ **O(M + N)**.

6.7.2. Atac

Rețeaua de calculatoare a companiei poate fi privită ca fiind un graf neorientat în care calculatoarele sunt reprezentate de noduri, iar legăturile directe dintre acestea de muchii. Rețeaua nu va mai funcționa corect dacă hacker-ul va ataca un calculator corespunzător unui nod critic. În aceste condiții problema se reduce la determinarea procentului de noduri critice din acest graf.

Vom utiliza algoritmul de determinare a nodurilor critice şi după determinarea lor le vom număra.

După determinarea numărului *NC* al nodurilor critice vom scrie în fişierul de ieşire raportul $100 \cdot NC / M$.

Analiza complexității

Citirea datelor de intrare implică citirea celor *M* muchii ale grafului, aşadar ordinul de complexitate al acestui subalgoritm este **O(M)**. În paralel cu citirea se realizează crearea structurii de date în care este memorat graful, ordinul de complexitate al acestei operații fiind tot **O(M)**.

Algoritmul de determinare a nodurilor critice ale unui graf are ordinul de complexitate **O(M + N)**.

Datele de ieşire constau într-un singur număr, aşadar ordinul de complexitate al operației de scriere a datelor de ieşire are ordinul de complexitate **O(1)**.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(M) + O(M) + O(M + N) + O(1) = \mathbf{O(M + N)}$.

6.7.3. Neutroni

Ionii pot fi priviți ca fiind nodurile unui graf neorientat ale cărui muchii reprezintă legăturile cristaline dintre ioni. Un cristal va fi reprezentat de o componentă conexă a grafului. În aceste condiții problema se reduce la determinarea numărului componentelor conexe ale unui graf şi la determinarea nodurilor critice ale acestui graf.

Vom utiliza algoritmul de determinare a nodurilor critice şi, în momentul detectării unui astfel de nod, îl vom insera într-un vector care conține rezultatele. Vom număra nodurile critice pe parcursul determinării lor.

Deoarece va trebui să determinăm nodurile critice pentru fiecare componentă conexă, vom număra componentele conexe pe parcursul executării algoritmului de determinare a nodurilor critice.

După determinarea vectorului care conține nodurile critice vom scrie în fişierul de ieşire numărul elementelor acestuia, numărul componentelor conexe identificate, precum şi numerele de ordine ale nodurilor critice.

Analiza complexității

Citirea datelor de intrare implică citirea celor *M* muchii ale grafului, aşadar ordinul de complexitate al acestui subalgoritm este **O(M)**. În paralel cu citirea se realizează crearea structurii de date în care este memorat graful, ordinul de complexitate al acestei operații fiind tot **O(M)**.

Algoritmul de determinare a nodurilor critice ale unii graf are ordinul de complexitate **O(M + N)**.

Datorită faptului că un graf poate conține cel mult N noduri critice, ordinul de complexitate al operației de scriere a datelor de ieşire are ordinul de complexitate **O(N)**.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(M) + O(M) + O(M + N) + O(N) = \mathbf{O(M + N)}$.