Biconexitate

Capitolul 8

- Considerații teoretice
- Determinarea componentelor biconexe
- Rezumat
- Implementări sugerate
- Probleme propuse
- Soluțiile problemelor

În cadrul acestui capitol vom prezenta noțiunea de biconexitate a unui graf. De asemenea, va fi introdusă noțiunea de componentă biconexă şi va fi prezentat un algoritm cu ajutorul căruia pot fi determinate componentele biconexe ale unui graf.

8.1. Considerații teoretice

Aşa cum am arătat în capitolul 6, prin eliminarea unui nod al unui graf, conexitatea grafului poate dispărea. Apare în mod natural problema determinării subgrafurilor maximale care nu conțin nici un astfel de nod.

Un graf care nu îşi poate pierde conexitatea în urma eliminării unuia dintre nodurile sale poartă denumirea de ***graf biconex***. Cu alte cuvinte, un graf biconex este un graf care nu conține nici un punct de articulație.

Verificarea biconexității unui graf este o operație foarte simplă, deoarece implică doar verificarea existenței unui punct de articulație. Dacă graful nu conține nici un astfel de nod, atunci el este biconex, iar dacă există cel puțin un astfel de nod, atunci el nu este biconex.

8.1.1. Componente biconexe

Noțiunea de componentă biconexă are o semnificație asemănătoare celei de componentă conexă. Dacă o componentă conexă este un subgraf maximal conex, atunci componenta biconexă este un graf maximal biconex.

Cu alte cuvinte, o componentă biconexă este un subgraf care nu îşi pierde conexitatea în urma eliminării oricărui nod din componența sa.

8.1.2. Triconexitate

Noțiunea de biconexitate poate fi extinsă, dar practic conceptele pe care le vom prezenta în continuare sunt foarte rar utilizate.

Un graf este considerat a fi ***triconex*** dacă nu îşi pierde conexitatea prin eliminarea a oricare două dintre nodurile sale. Cu alte cuvinte, un graf este triconex dacă prin eliminarea unuia dintre nodurile sale el rămâne biconex.

Evident, poate fi introdusă şi noţiunea de componentă triconexă; aceasta reprezintă un graf maximal triconex al unui graf.

În general, putem spune că un graf este ***k-conex*** dacă el nu îşi pierde conexitatea prin eliminarea a oricare $k - 1$ noduri din componenţa sa, sau dacă îşi menţine (k-1)-conexitatea prin eliminarea unuia dintre nodurile sale. Noţiunea de componentă k-conexă poate fi şi ea introdusă.

Observăm faptul că un graf ***k-conex*** este totodată şi un graf ***(k-1)-conex***. Deşi acest lucru este evident, pe baza prorietăţii enunţate deducem că un graf biconex este întotdeauna şi un graf conex.

8.2. Determinarea componentelor biconexe

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

8.2.1. Preliminarii

Algoritmul de determinare a componentelor biconexe ale unui graf se bazează şi el pe parcurgerea *DF* a grafului respectiv. De fapt, datorită faptului că o componentă biconexă nu conţine nici un nod critic, o astfel de componentă poate fi identificată în momentul în care este găsit un astfel de nod.

Practic, vom insera într-o stivă muchiile de înaintare şi de întoarcere în ordinea în care acestea sunt parcurse. În momentul în care identificăm un punct de articulaţie, toate muchiile aflate/inserate în stivă după vizitarea acelui nod vor face parte dintr-o componentă biconexă. Toate aceste muchii vor fi ulterior eliminate din stivă. Nodurile care fac parte din componenta conexă sunt reprezentate de extremităţile muchiilor respective.

8.2.2. Prezentarea algoritmului

Singura diferenţă faţă de algoritmul de determinare a punctelor critice ale unui graf o reprezintă păstrarea unei stive care conţine muchii. Această stivă poate fi păstrată static sau dinamic.

Datorită faptului că, în cadrul parcurgerii *DF*, pentru fiecare nod va trebui să luăm în considerare toate muchiile care pornesc din nodul respectiv, cu excepţia celei care îl uneşte de părintele său, va trebui să cunoaştem părinţii 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 vom avea nevoie să păstrăm şi nivelurile nodurilor.

În momentul în care vizităm un nod, vom memora vârful stivei, deoarece o eventuală componentă biconexă va fi dată doar de muchiile care vor fi inserate în stivă ulterior.

Varianta în pseudocod a algoritmului care utilizează o stivă memorată static este:

```
Subalgoritm ScrieComponenta(i,k,G,stiva):
                                        { i, k – extremitățile ultimei muchii din }
                                        { stivă care face  parte din componentă }
                                                                  { G – graful }
                                                          { stiva – stiva de muchii }
  noduri ← ∅
  repetă
    stiva ⇒ (x,y)                              { eliminarea unei muchii din stivă }
    noduri ← nivel ← {x,y}                  { se adaugă în mulțime nodurile x, y }
  până când (x,y) = (i,k)
  Scrie elementele mulțimii noduri
sfârşit subalgoritm

Subalgoritm Componente_Biconexe(k,părinte,G,niv,niv_min):
                                                                  { k – nodul curent }
                                        { G – graful, părinte – părintele nodului curent }
                                                                { 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
    vizitat_k ← adevărat
    nivel_k ← niv
    val_min ← niv
    pentru toți vecinii i ai lui k execută:
      dacă i ≠ părinte atunci
        stiva ⇐ (i,k)                          { inserarea muchiei în stivă }
        dacă vizitat_i atunci
          Componente_Biconexe(i,nod,G,niv+1,aux)
        altfel
          Componente_Biconexe(i,nod,G,niv+1,aux)
          dacă niv ≤ aux atunci
            ScrieComponenta(i,k)
          sfârşit dacă
        sfârşit dacă
```

```
      dacă aux < val_min atunci
        val_min ← aux
      sfârşit dacă
      Elimină din graf muchia (i,k)
    sfârşit pentru
  sfârşit dacă
sfârşit subalgoritm

Algoritm Determinare_Componente_Biconexe(G)
  Componente_Biconexe(1,-1,G,1,aux)
sfârşit algoritm
```

Am considerat nodul `1` ca fiind rădăcina arborelui *DF*. Prin apelul subalgoritmului `Componente_Biconexe` se determină toate punțile din graful considerat.

8.2.3. Analiza complexității

Algoritmul constă într-o variantă modificată a unei parcurgeri în adâncime a unui graf, parcurgere care are ordinul de complexitate $O(M + N)$. Pe lângă această parcurgere, se creează o stivă cu muchiile grafului. Fiecare muchie va fi inserată în stivă exact o dată şi va fi eliminată din stivă exact o dată. Ca urmare, timpul total necesar operațiilor cu stiva are ordinul de complexitate $O(M)$. Pentru identificarea nodurilor care fac parte dintr-o anumită componentă conexă se pot utiliza algoritmi eficienți cu ordinul total de complexitate $O(M)$. Ca urmare, algoritmul de determinare a componentelor conexe ale unui graf are ordinul de complexitate $O(M + N)$.

8.2.4. Partiționarea muchiilor

Este evident faptul că pentru un graf, componentele conexe maximale reprezintă o partiționare a nodurilor acestuia. Cu alte cuvinte, fiecare nod face parte din exact o componentă conexă maximală.

În cazul biconexității, această afirmație nu mai este adevărată. Există noduri care pot face parte simultan din componente biconexe diferite.

După cum s-a arătat în momentul prezentării algoritmului, o componentă biconexă este identificată pe baza muchiilor care o formează. Aşadar, numai despre muchii se poate spune că fac parte din exact o componentă biconexă.

Aşadar, componentele biconexe reprezintă o partiționare a muchiilor unui graf.

8.3. Rezumat

În cadrul acestui capitol am prezentat noțiunea de biconexitate în graf, precum şi un algoritm eficient care poate fi utilizat pentru a determina componentele biconexe ale unui graf.

De asemenea, am generalizat noțiunea de biconexitate şi am arătat că, spre deosebire de componentele conexe, componentele biconexe reprezintă o partiționare a muchiilor grafului.

8.4. Implementări sugerate

Pentru a vă familiariza cu modul în care trebuie implementate rezolvările problemelor ale căror soluții necesită cunoştințe referitoare la noțiunea de biconexitate vă sugerăm să încercați să implementați algoritmi pentru:

1. determinarea componentelor biconexe folosind metoda backtracking pentru a genera toate posibilitățile de partiționare a muchiilor unui graf;
2. determinarea componentelor biconexe ale unui graf neorientat conex;
3. determinarea componentelor biconexe ale unui graf neorientat neconex;
4. verificarea triconexității unui graf prin eliminări succesive ale unei perechi de noduri şi verificarea conexității grafului obținut la fiecare pas;
5. verificarea triconexității unui graf prin eliminări succesive ale unui nod şi verificarea biconexității grafului obținut la fiecare pas.

8.5. 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 şi în cadrul capitolelor dedicate muchiilor şi punctelor critice. Cunoştințele suplimentare necesare sunt minime.

8.5.1. Tolani

Descrierea problemei

Pe fiecare dintre cele *N* planete locuite de tolani se află câte o poartă stelară. Din nefericire, datorită consumului mare de energie necesar activării unei găuri de vierme între anumite porți nu se poate călători de la o planetă la oricare alta în mod direct. Totuşi există un număr total de *M* perechi de planete între care se poate călători direct, iar cele *M* perechi au fost alese în aşa fel încât să se poată circula (direct sau cu "escale" pe planete intermediare) între oricare două planete. Datorită conflictului dintre tolani şi *Lorzii Sistemului* există posibilitatea ca, în urma unui atac, anumite porți stelare să fie distruse, ceea ce ar putea duce la imposibilitatea unei călătorii între oricare două planete.

Consiliul doreşte să cunoască grupurile de planete care nu vor fi afectate chiar dacă o poartă stelară de pe o planetă din grupul respectiv este distrusă. Un grup de planete nu este afectat de distrugere dacă se poate călători în continuare între oricare două planete, cu excepția celei a cărei poartă stelară a fost distrusă. O planetă poate face parte din mai multe grupuri, dar dacă există posibilitatea de a călători direct între două pla-

nete, atunci trebuie să existe cel puțin un grup care conține cele două planete. În plus, un grup de planete nu poate fi subgrup al unui alt grup. Cu alte cuvinte, grupurile formate trebuie să fie maximale.

Date de intrare
Prima linie a fişierului de intrare **TOLANI.IN** conține numărul N al planetelor locuite de tolani şi numărul M al perechilor de planete între care se poate călători direct. Fiecare dintre următoarele M linii va conține câte două numere întregi x şi y cu semnificația: există posibilitatea de a călători direct între planetele identificate prin x şi y.

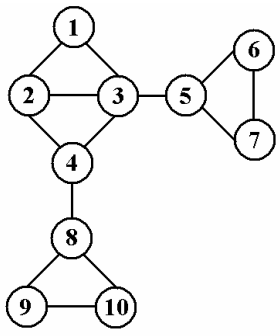
Date de ieşire
Fişierul de ieşire **TOLANI.OUT** va conține un număr de linii egal cu numărul grupurilor formate. Fiecare dintre aceste linii va conține numerele de identificare ale planetelor din grupul corespunzător, separate printr-un spațiu.

Restricții şi precizări
- $1 \le N \le 100$;
- $1 \le M \le 1000$;
- planetele sunt identificate prin numere întregi cuprinse între 1 şi N;
- grupurile vor conține cel puțin două planete;
- numerele de ordine ale planetelor dintr-un grup pot fi scrise în fişierul de ieşire în orice ordine;
- planetele aparținând unui grup pot fi descrise în fişierul de ieşire în orice ordine;
- dacă există posibilitatea călătoriei directe între două planete x şi y, în fişierul de ieşire va exista o singură linie pe care se va afla perechea x şi y.

Exemplu

TOLANI.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
```

TOLANI.OUT
```
1 2 3 4
3 5
4 8
5 6 7
8 9 10
```



Timp de execuție: 1 secundă/test

8.5.2. Egipt

Descrierea problemei
Cleopatra a comandat construirea unei noi piramide în care doreşte să îşi petreacă eternitatea. Arhitectul i-a adus planurile piramidei care are *N* camere şi *M* culoare care unesc câte două camere. Cleopatra doreşte să ştie dacă, în cazul în care accesul într-una din camere este blocat, se poate ajunge din fiecare cameră în oricare alta (evident, fără a o lua în considerare pe cea blocată) indiferent care dintre camerele piramidei a fost blocată.

Date de intrare
Prima linie a fişierului de intrare **EGIPT.IN** conţine numărul *N* al camerelor piramidei şi numărul *M* al culoarelor. Fiecare dintre următoarele *M* linii va conţine câte două numere întregi *x* şi *y* cu semnificaţia: există un culoar prin care se poate trece din camera identificată prin *x* în camera identificată prin *y* şi din camera identificată prin *y* în camera identificată prin *x*.

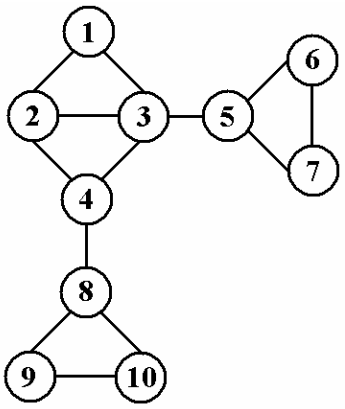
Date de ieşire
Fişierul de ieşire **EGIPT.OUT** va conţine o singură linie pe care se va afla mesajul `DA` în cazul în care se poate ajunge din fiecare cameră în oricare alta, chiar dacă accesul în una dintre camere va fi blocat şi mesajul `NU` în caz contrar.

Restricţii şi precizări
- $1 \le N \le 100$;
- $1 \le M \le 1000$;
- camerele sunt identificate prin numere întregi cuprinse între 1 şi *N*;
- există cel mult un culoar între oricare două camere.

Exemplu

EGIPT.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
```

EGIPT.OUT
```
NU
```



Timp de execuţie: 1 secundă/test

8.5.3. Vulcan

Descrierea problemei
Vulcanienii doresc să construiască un nou sistem de comunicaţii pe planeta lor. Din nefericire lucrările nu sunt încă terminate, dar *Federaţia Planetelor Unite* cere detalii referitoare la fiabilitatea sistemului în acest moment. Au fost construite un număr total de N staţii de emisie şi au fost configurate un număr total de M legături directe între perechi de staţii. Prin intermediul unei legături pot fi transmise informaţii în ambele sensuri.

O porţiune a sistemului este fiabilă dacă în cazul în care o staţie nu mai funcţionează există posibilitatea unei comunicări directe sau indirecte între oricare două dintre celelalte staţii ale porţiunii.

Vulcanienii trebuie să precizeze care porţiuni ale sistemelor sunt fiabile, indicând staţiile aparţinând fiecărei porţiuni. În cel mai rău caz, o porţiune fiabilă este formată din două staţii între care a fost configurată o legătură directă. O staţie poate face parte din mai multe porţiuni fiabile, dar o legătură trebuie să facă parte din exact o astfel de porţiune. O porţiune fiabilă trebuie să conţină cât mai multe staţii de emisie.

Date de intrare
Prima linie a fişierului de intrare **`VULCAN.IN`** conţine numărul N al staţiilor de emisie şi numărul M al legăturilor configurate. Fiecare dintre următoarele M linii va conţine câte două numere întregi x şi y cu semnificaţia: a fost configurată o legătură directă între staţiile de emisie identificate prin x şi y.

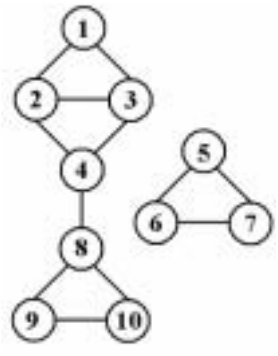
Date de ieşire
Fişierul de ieşire **`VULCAN.OUT`** va conţine un număr de linii egal cu numărul porţiunilor fiabile. Fiecare dintre aceste linii va conţine numerele de identificare ale staţiilor de emisie din porţiunea corespunzătoare, separate printr-un spaţiu.

Restricţii şi precizări
- $1 \le N \le 100$;
- $1 \le M \le 1000$;
- staţiile sunt identificate prin numere întregi cuprinse între 1 şi N;
- numerele de ordine ale staţiilor dintr-o porţiune pot fi scrise în fişierul de ieşire în orice ordine;
- porţiunile pot fi descrise în fişierul de ieşire în orice ordine;
- nu se garantează posibilitatea comunicării între oricare două staţii chiar dacă toate staţiile funcţionează la parametri normali.

Exemplu

VULCAN.IN
```
10 12
1 2
1 3
2 3
2 4
3 4
4 8
5 6
5 7
6 7
8 9
8 10
9 10
```

VULCAN.OUT
```
1 2 3 4
4 8
5 6 7
8 9 10
```



Timp de execuție: 1 secundă/test

8.5.4. Informații

Descrierea problemei

Serviciul Imperial de Informații al *Împăratului Palpatine* este format din *N* agenți imperiali identificați prin numere cuprinse între 1 şi *N*. Din motive de securitate nu oricare doi agenți pot comunica în mod direct, dar există posibilitatea ca oricare doi agenți să poată comunica, eventual cu ajutorul unor intermediari.

Un serviciu de informații este considerat a fi sigur dacă eliminarea unui agent nu afectează comunicarea dintre ceilalți agenți.

Va trebui să verificați dacă serviciul de informații este sigur. În cazul în care serviciul nu este sigur va trebui să identificați agenții cheie, legăturile de comunicație cheie, precum şi toate grupurile de agenți care formează subservicii sigure. Un subserviciu sigur este maximal, în sensul că nu poate exista un subserviciu care să fie format din toți agenții unui alt subserviciu la care se adaugă alți agenți.

Un agent cheie este un agent a cărui eliminare duce la imposibilitatea comunicării între cel puțin doi dintre agenții rămaşi.

O legătură de comunicație cheie este dată de doi agenți care pot comunica în mod direct, dar întreruperea legăturii directe dintre cei doi agenți duce la imposibilitatea comunicării directe între cel puțin doi agenți.

Date de intrare

Prima linie a fişierului de intrare **INFO.IN** conține numărul *N* al agenților şi numărul *M* al legăturilor de comunicație dintre aceştia. Fiecare dintre următoarele *M* linii va

conține câte două numere întregi x şi y cu semnificația: agenții identificați prin x şi y pot comunica direct.

Date de ieşire

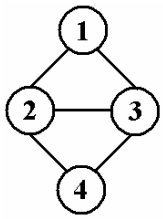
Prima linie a fişierului de ieşire **INFO.OUT** va conține mesajul DA în cazul în care serviciul este sigur şi mesajul NU în caz contrar. Cea de-a doua linie a fişierului va conține numărul agenților cheie, iar următoarea linie va conține numerele de ordine ale agenților cheie, separate printr-un spațiu. Următoarea linie a fişierului va conține numărul l al legăturilor cheie. Fiecare dintre următoarele l linii va conține câte două numere x şi y, separate printr-un spațiu, cu semnificația: legătura de comunicație dintre agentul x şi agentul y este o legătură cheie. Următoarea linie va conține numărul k al subserviciilor sigure. Fiecare dintre următoarele k linii va conține numerele de identificare ale agenților care fac parte din subserviciul respectiv.

Restricții şi precizări

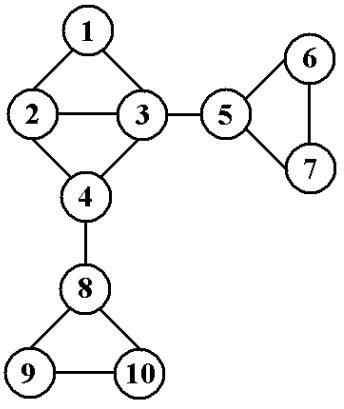
- $1 \le N \le 100$;
- $1 \le M \le 1000$;
- în cazul în care serviciul este sigur nu vom avea nici un agent cheie, nici o legătură cheie şi va exista un singur subserviciu sigur (format din toți agenții);
- fiecare agent cheie va apărea pe linia corespunzătoare o singură dată;
- fiecare legătură cheie va fi descrisă o singură dată;
- subserviciile sigure pot fi descrise în fişierul de ieşire în orice ordine;
- Numerele de ordine ale agenților care fac parte dintr-un subserviciu sigur pot fi scrise în orice ordine.

Exemple

INFO.IN	**INFO.OUT**
4 5	DA
1 2	0
1 3	0
2 3	1
2 4	1 2 3 4
3 4	



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



Timp de execuție: 1 secundă/test

8.6. 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.

8.6.1. Tolani

Planetele tolanilor vor reprezenta nodurile unui graf neorientat; între două noduri ale grafului va exista o muchie doar dacă între cele două planete corespunzătoare se va putea călători în mod direct.

Astfel, practic, va trebui doar să identificăm componentele biconexe ale acestui graf şi, pe măsura detectării lor, să le scriem în fişierul de ieşire.

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 componentelor biconexe ale unui graf are ordinul de complexitate **O(M + N)**. Pe măsura determinării acestora, ele vor fi descrise în fişierul de ieşire; aşadar, nu se va consuma timp suplimentar pentru crearea acestui fişier.

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

8.6.2. Egipt

Piramida Cleopatrei poate fi considerată a fi un graf neorientat în care nodurile reprezintă camerele, iar muchiile reprezintă culoarele.

Astfel, problema se reduce la simpla verificare a biconexității unui graf neorientat. Există două posibilități: fie determinăm componentele biconexe şi dacă obținem o singură astfel de componentă deducem că graful este biconex, fie verificăm dacă graful conține puncte de articulație. După verificare, în fişierul de ieşire va fi scris un mesaj corespunzător.

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 este memorat graful, ordinul de complexitate al acestei operații fiind tot **O(M)**.

Algoritmul de verificare a biconexității unui graf are ordinul de complexitate **O(M + N)**, indiferent de metoda aleasă pentru verificare.

După verificare, vom scrie mesajul corespunzător în fişierul de ieşire, operație al cărei ordin de complexitate este **O(1)**.

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

8.6.3. Vulcan

Sistemul de comunicație vulcanian poate fi caracterizat printr-un graf ale cărui vârfuri reprezintă stațiile de emisie şi ale cărui muchii reprezintă legăturile configurate.

O porțiune fiabilă a sistemului va fi dată de o componentă biconexă maximală a acestuia. Ca urmare, problema se reduce la determinarea componentelor biconexe ale unui graf neorientat.

Pe măsură ce aceste componente sunt determinate se vor scrie în fişierul de ieşire nodurile care le formează.

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 este memorat graful, ordinul de complexitate al acestei operații fiind tot **O(M)**.

Algoritmul de determinare a componentelor biconexe ale unui graf are ordinul de complexitate **O(M + N)**. Pe măsura determinării acestora, ele vor fi descrise în fişierul de ieşire; aşadar, nu se va consuma timp suplimentar pentru crearea acestui fişier.

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

8.6.4. Informații

Serviciul Imperial de Informații poate fi reprezentat printr-un graf neorientat ale cărui vârfuri reprezintă agenții imperiali. Între două vârfuri va exista o muchie dacă şi numai dacă cei doi agenți pot comunica direct între ei.

În aceste condiții, un agent cheie reprezintă un punct de articulație al grafului, o legătură cheie reprezintă o punte a grafului, iar un subserviciu sigur reprezintă o componetă biconexă a grafului.

Ca urmare, problema se reduce la determinarea punctelor critice, a muchiilor critice şi a componentelor biconexe ale unui graf neorientat.

Toate acestea pot fi determinate printr-o singură parcurgere în adâncime a grafului dar, pentru aceasta, la fiecare pas al parcurgerii trebuie efectuate toate operațiile corespunzătoare celor trei algoritmi (de determinare a punctelor de articulație, de determinare a muchiilor şi de determinare a componentelor biconexe).

Datorită ordinii în care trebuie scrise datele de ieşire, fişierul nu mai poate fi creat pe parcurs. Din acest motiv, datele obținute trebuie memorate şi scrise în fişier doar la sfârşit.

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 este memorat graful, ordinul de complexitate al acestei operații fiind tot **O(M)**.

Parcurgerea în adâncime care unifică cei trei algoritmi are ordinul de complexitate **O(M + N)** deoarece numărul operațiilor suplimentare introduse la fiecare pas este constant. Practic, la fiecare pas, vom verifica dacă am obținut un punct critic, dacă am obținut o muchie critică şi dacă am obținut o nouă componentă biconexă. Datele care vor fi scrise în fişierul de ieşire vor fi memorate pe măsură ce vor fi determinate; nu se va consuma timp suplimentar pentru memorarea lor.

După determinarea punctelor critice, a muchiilor critice şi a componentelor biconexe, datele corespunzătoare trebuie scrise în fişierul de ieşire. Vom avea cel mult $N - 2$ puncte de articulație, deci ordinul de complexitate al operației de scriere a acestora este $O(N)$. Numărul punților este de cel mult $N - 1$, deci şi operația de scriere a acestora are ordinul de complexitate $O(N)$. Componentele biconexe reprezintă o partiționare a muchiilor grafului; din acest motiv, în cel mai defavorabil caz, s-ar putea ca pentru descrierea componentelor să fie necesară descrierea fiecărei muchii. Ca urmare, operația de scriere a datelor referitoare la componentele biconexe are ordinul de complexitate $O(M)$. Aşadar, ordinul de complexitate al operației de scriere a datelor de ieşire este $O(N) + O(N) + O(M) = \mathbf{O(M + N)}$.

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