Cicluri

Capitolul 5

- **Etajarea grafurilor**
- **O clasificare a muchiilor**
- **Determinarea unui ciclu**
- **Cicluri disjuncte**
- **Rezumat**
- **Implementări sugerate**
- **Probleme propuse**
- **Soluțiile problemelor**

În cadrul acestui capitol ne vom ocupa de ciclurile unui graf neorientat. Pentru început, vom prezenta modul în care poate fi etajat pe niveluri un graf. În continuare vom clasifica muchiile grafului în funcție de rolul lor în cadrul etajării pe niveluri. Vom prezenta apoi modul în care poate fi determinat un ciclu al unui graf, precum şi noțiunea de cicluri disjuncte şi modul de determinare a acestora.

5.1. Etajarea grafurilor

Această secțiune este dedicată prezentării conceptului de etajare pe niveluri a unui graf neorientat. Acest concept va fi utilizat pentru a realiza o clasificare a muchiilor grafului, precum şi pentru a descrie o modalitate simplă de determinare a ciclurilor unui graf.

În cele ce urmează vom presupune, fără a restrânge generalitatea, că grafurile considerate 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.

5.1.1. Existența ciclurilor

Este cunoscut faptul că un arbore parțial al unui graf conex conține exact $n - 1$ muchii, unde n este numărul nodurilor grafului. De asemenea, este cunoscut faptul că orice altă muchie adăugată unui astfel de arbore va "închide" un ciclu.

Ca urmare, pentru ca un graf să fie conex şi să nu conțină nici un ciclu, este necesar ca el să conțină exact $n - 1$ muchii (să fie un arbore). Un graf care conține mai multe

muchii va conține cel puțin un ciclu, iar un graf cu mai puține muchii nu mai poate fi conex.

Despre arborii parțiali se poate spune că sunt ***maximali în raport cu aciclicitatea***, deoarece adăugarea unei muchii duce la apariția unui ciclu şi minimali în raport conexitatea, deoarece eliminarea unei muchii dă naştere la două componente conexe.

5.1.2. Grafuri etajate pe niveluri

Etajarea grafurilor pe niveluri se realizează cu ajutorul unei parcurgeri în adâncime. Vom numi ***nivel*** al unui nod "adâncimea" la care este acesta vizitat în timpul parcurgerii în adâncime. Să luăm în considerare graful din figura 5.1 şi să realizăm o parcurgere în adâncime începând cu nodul 1. Acesta va fi primul nod vizitat şi va fi singurul nod aflat pe primul nivel.

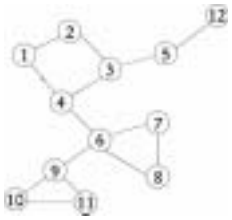


Figura 5.1: Un graf

Dacă vecinii unui nod sunt luați în considerare în ordinea dată de numerele lor de ordine, atunci următorul nod vizitat va fi 2. Acest nod se va afla pe al doilea nivel. În acest moment am ajuns la nodul 3; următorul nod vizitat va fi 3, care se va afla pe al treilea nivel. În continuare va fi vizitat nodul 4 care se va afla pe al patrulea nivel, nodul 6 care se va afla pe al cincilea nivel, nodul 7 care se va afla pe al şaselea nivel şi nodul 8 care se va afla pe al şaptelea nivel.

În acest moment se va reveni la nodul 6 (care se află, după cum am arătat anterior, pe al cincilea nivel). Următorul nod vizitat este 9 care se va afla pe al şaselea nivel. Vor urma nodurile 10 şi 11 care se vor afla pe al şaptelea, respectiv al optulea nivel.

Acum se revine până la nodul 3 (aflat pe al treilea nivel); vor fi vizitate în continuare nodurile 5 şi 12 care se vor afla pe nivelurile al patrulea, respectiv al cincilea.

În figura 5.2 este ilustrată reprezentarea pe niveluri a grafului considerat. După cum se poate vedea, a fost stabilit un sens pentru fiecare muchie, acestea devenind arce.

După cum puteți observa, muchiilor care au fost "parcurse" pentru a vizita noi noduri li s-a asociat un sens de la nodul de pe nivelul superior spre nodul de pe nivelul inferior. Tuturor celorlalte muchii li s-a asociat un sens de la un nod de pe un nivel inferior spre un nivel superior. Evident, am considerat un nivel ca fiind superior dacă are numărul de ordine mai mic.

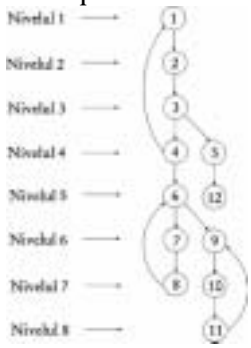


Figura 5.2: Reprezentarea pe niveluri a grafului din figura 5.1

Mai trebuie remarcat că, datorită faptului că s-a realizat o parcurgere în adâncime, în arborele DF obținut, orice muchie leagă două noduri dintre care unul este descendent al celuilalt. Nu va exista nici o muchie care să unească două noduri din subarbori diferiți.

5.1.3. Algoritmul de stabilire a nivelurilor

Pentru a cunoaşte nivelul pe care se află un nod în graful etajat se poate păstra un vector `nivel` care să conţină această informaţie pentru fiecare nod.

Pentru determinarea acestui vector vom realiza o simplă parcurgere în adâncime dar, în momentul în care vom vizita un nou nod, vom păstra adâncimea la care se află nodul. Algoritmul este prezentat în continuare:

```
Subalgoritm Stabilire_Niveluri(k,G,niv)
                                                    { k – nodul curent }
                                                    { G – graful }
                                                    { niv – nivelul curent }
  vizitat_k ← adevărat
  nivel_k ← niv
  pentru toţi vecinii i ai lui k execută
    dacă nu vizitat_i atunci
      Stabilire_Niveluri(i,G,niv+1)
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

5.2. O clasificare a muchiilor

În cadrul acestei secţiuni vom prezenta, pe scurt, o modalitate de clasificare a muchiilor unui graf în funcţie de rolul lor în cadrul unei parcurgeri a grafului. Noţiunile prezentate vor fi utilizate atât în cadrul acestui capitol, cât şi în cele care urmează.

5.2.1. Muchii de înaintare

Aşa cum am afirmat anterior, în cadrul etajării pe niveluri se stabileşte un sens al muchiilor. Aşadar, muchiilor care unesc un nod de pe un anumit nivel de un nod de pe nivelul imediat următor li se asociază un sens dinspre nivelul superior spre nivelul inferior. Muchiile de acest tip poartă denumirea de ***muchii de înaintare*** deoarece sunt utilizate pentru a "avansa" în timpul determinării arborelui *DF*. Aceste muchii mai sunt cunoscute şi sub denumirile de *muchii de avansare* sau *muchii înainte*.

Proprietatea cea mai importantă a muchiilor de înaintare este faptul că diferenţa absolută dintre nivelurile celor două noduri unite printr-o astfel de muchie este întotdeauna 1.

5.2.2. Muchii de întoarcere

Pentru muchiile care unesc două noduri aflate pe niveluri neconsective se stabileşte un sens dinspre nodul de pe nivelul inferior spre cel de pe nivelul superior. Aceste muchii poartă denumirea de ***muchii de întoarcere*** deoarece, dacă ar fi traversate, s-ar ajunge

la un nod care a fost deja vizitat. Aceste muchii mai sunt cunoscute şi sub denumirea de *muchii înapoi*.

Proprietatea cea mai importantă a muchiilor de întoarcere este faptul că diferența absolută dintre nivelurile celor două noduri unite printr-o astfel de muchie este întotdeauna cel puțin egală cu 2.

5.2.3. Clasificarea

În cazul arborilor *DF* nu există alte tipuri de muchii. Aşadar, se poate spune că orice muchie care nu este de înaintare este muchie de întoarcere şi orice muchie care nu este de întoarcere este muchie de înaintare.

Pentru graful din figurile 5.1 şi 5.2 avem unsprezece muchii de avansare (1 – 2, 2 – 3, 3 – 4, 3 – 5, 4 – 6, 5 – 12, 6 – 7, 6 – 9, 7 – 8, 9 – 10 şi 10 – 11) şi trei muchi de întoarcere (1 – 4, 6 – 8 şi 9 – 11).

O proprietatea interesantă este dată de faptul că muchiile de înaintare formează întotdeauna un arbore parțial al grafului.

5.2.4. Muchii de traversare

Aşa cum am spus, în cadrul arborilor *DF* nu există decât două tipuri de muchii. Cu toate acestea, există posibilitatea de a determina şi alte tipuri de arbori parțiali ai grafului (care sunt obținuți printr-o metodă diferită față de parcurgerea în adâncime).

Dacă etajăm pe niveluri graful în funcție de un astfel de arbore (pentru graful din figura 5.1, o astfel de etajare este prezentată în figura 5.3), există posibilitatea de a obține muchii care unesc noduri aflate în doi subarbori diferiți ai unui nod. Aceste muchii poartă denumirea de ***muchii de traversare***.

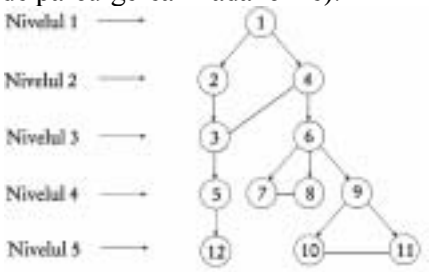


Figura 5.3: O altă etajare a grafului din figura 5.1

Pentru muchiile de traversare nu poate fi întotdeauna stabilit un sens, deoarece există posibilitatea să unească noduri aflate pe acelaşi nivel.

Pentru etajarea din figura 5.3 avem trei muchii de traversare şi anume: 3 – 4, 7 – 8 şi 10 – 11.

5.3. Determinarea unui ciclu

Există o mulțime de posibilități de identificare a unui ciclu într-un graf. Datorită faptului că muchiile de înaintare formează întotdeauna un arbore parțial, oricare muchie de întoarcere sau de traversare va închide un ciclu. În cadrul acestei secțiuni vom prezenta câteva modalități prin care pot fi determinate nodurile care fac parte dintr-un anumit ciclu.

5.3.1. Şirul părinţilor

Pentru fiecare nod al unui graf etajat ***părintele*** este nodul de pe nivelul anterior de la care porneşte o muchie de înaintare spre nodul considerat. Evident, toate nodurile vor avea un părinte, cu excepţia celui de pe primul nivel.

Şirul părinţilor poate fi determinat foarte simplu în timpul parcurgerii; pentru o parcurgere *DF*, algoritmul este prezentat în continuare:

```
Subalgoritm Stabilire_Părinţi(k,G)
                                              { k – nodul curent }
                                                  { G – graful }
  vizitat_k ← adevărat
  pentru toţi vecinii i ai lui k execută
    dacă nu vizitat_i atunci
      părinte_i ← k
      Stabilire_Părinţi(i,G)
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

De obicei, pentru a arăta faptul că nodul de pe primul nivel (rădăcina) nu are nici un părinte, se foloseşte o valoare specială în poziţia corespunzătoare a vectorului părinţilor (în majoritatea cazurilor valoarea aleasă este 0 sau -1).

5.3.2. Identificarea unui ciclu în cadrul parcurgerii DF

În cazul parcurgerii *DF*, în momentul în care o muchie ajunge la un nod deja vizitat, aceasta nu mai este muchie de înaintare, ci muchie de întoarcere, aşadar închide un ciclu. Datorită faptului că această muchie leagă un nod de un predecesor (strămoş) al acestuia, se pot identifica toate nodurile care fac parte din ciclul respectiv folosind şirul părinţilor.

Un aspect important este faptul că vom găsi o muchie între nodul curent şi părintele său (care este deja vizitat). Aceasta nu este o muchie de întoarcere şi nu va închide un ciclu, deci va trebuie ignorată.

Aşadar, pentru a identifica un ciclu într-un graf poate fi utilizat următorul algoritm:

```
Subalgoritm ScrieCiclu(k,i,părinte)
                                    { k – nodul de pe nivelul inferior }
                                  { i – nodul de pe nivelul superior }
                                      { părinte - şirul părinţilor }
  nod ← k
  scrie nod
  cât timp i ≠ nod execută
    nod ← părinte_nod
```

```
    scrie nod
  sfârşit cât timp
sfârşit subalgoritm

Subalgoritm Identificare_Ciclu(k,G)
                                                    { k – nodul curent }
                                                    { G – graful }
  vizitat_k ← adevărat
  pentru toți vecinii i ai lui k execută
    dacă nu vizitat_i atunci
      părinte_i ← k
      Identificare_Ciclu(i,G)
    altfel
      dacă i ≠ părinte_k atunci
        ScrieCiclu(k,i,părinte)
        * întrerupe execuția
      sfârşit dacă
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

5.3.3. Închiderea unui ciclu cu o muchie de traversare

Determinarea unui ciclu este puțin mai dificilă în cazul în care graful conține muchii de traversare. Să presupunem că dorim să determinăm nodurile care fac parte dintr-un ciclu închis de o astfel de muchie.

Evident, extremitățile acestei muchii vor avea un strămoş comun (eventual rădăcina arborelui). Pornind de la cele două extremități, va trebui să "urcăm în arbore" până vom ajunge la strămoşul comun. Operația este îngreunată de faptul că cele două extremități s-ar putea afla pe niveluri diferite.

O variantă este să determinăm drumurile spre rădăcină şi apoi să eliminăm eventuala porțiunea comună. Vom păstra aceste două drumuri în doi vectori şi apoi vom afişa ultimul nod din porțiunea comună, precum şi nodurile din porțiunile distincte.

Algoritmul este prezentat în continuare:

```
Subalgoritm DrumSpreRădăcină(x,părinte,drum)
                                                    { x – nodul de pornire }
                                                    {părinte - şirul părinților }
                                                    { drum – variabilă transmisă prin referință }
                                                    { va conține drumul spre rădăcină }
  nod ← x
  nr ← 0
  cât timp nod ≠ -1 execută
```

```
    nr ← nr + 1
    drum_nr ← nod
    nod ← părinte_nod
  sfârşit cât timp
  returnează nr                          { se returnează lungimea drumului}
sfârşit subalgoritm

Subalgoritm IdentificăPorţiuneaComună(drumx,nrx,drumy,nry)
                                  { drumx, drumy – drumurile spre rădăcină }
                                   { nrx, nry – lungimile celor două drumuri }
  i ← nrx
  j ← nry
  nr ← 0
  cât timp drum_i = drum_j execută
    nr ← nr + 1
    i ← i - 1
    j ← j - 1
  sfârşit cât timp
  returnează nr                  { se returnează lungimea porţiunii comune }
sfârşit subalgoritm

Subalgoritm ScriePorţiune(drum,nr,nrcom)
                                              { drum – drumul spre radacină }
                                                  { nr – lungimea drumului }
                                          { nrcom – lungimea porţiunii comune }
  pentru i ← 1, nr - nrcom execută
    scrie drum_i
  sfârşit pentru
sfârşit subalgoritm

Algoritm Identificare_Ciclu(x,y,părinte)
                                              { x, y – nodurile care închid ciclul }
                                                  { părinte - şirul părinţilor }
  nrx ← DrumSpreRădăcină(x,părinte,drumx)
  nry ← DrumSpreRădăcină(y,părinte,drumy)
  nrcom ← IdentificăPorţiuneaComună(drumx,nrx,drumy,nry)
  scrie drumx_nrx-nrcom+1
  ScriePorţiune(drumx,nrx,nrcom)
  ScriePorţiune(drumy,nry,nrcom)
sfârşit algoritm
```

În cel mai defavorabil caz va trebui să urcăm de la două noduri aflate pe ultimul nivel, până la rădăcină. Este posibil ca această urcare să necesite parcurgerea majorită-

ții nodurilor grafului (până la $n - 1$). Aşadar, ordinul de complexitate al operației de parcurgere a drumului până la rădăcină este $O(n)$.

Practic vom traversa cele două drumuri în întregime (o parte pentru a determina porțiunea comună şi cealaltă parte pentru a scrie nodurile care formează ciclul), deci aceste două operații au şi ele ordinul de complexitate $O(n)$.

În concluzie, în cel mai defavorabil caz, dacă avem un arbore parțial al unui graf, operația de determinare a nodurilor care fac parte dintr-un ciclu închis de o muchie se realizează în timp liniar, în funcție de numărul nodurilor grafului.

Să considerăm etajarea din figura 5.3; presupunem că dorim să determinăm ciclul închis de muchia 7 – 8. Vom determina drumurile spre rădăcină 7 – 6 – 4 – 1, respectiv 8 – 6 – 4 – 1. Porțiunea comună a acestora este 6 – 4 – 1, deci va fi eliminată. Se va scrie nodul 6 (primul element al porțiunii comune care reprezintă punctul de intersecție al celor două drumuri) şi nodurile care nu se află în porțiunea comună: 7 din primul drum şi 8 din al doilea. Aşadar, ciclul închis de muchia 7 – 8 va conține nodurile 6, 7 şi 8.

5.3.4. O îmbunătățire a algoritmului de închidere a ciclului

În cazul algoritmului prezentat anterior, parcurgerea porțiunii comune spre rădăcină este inutilă. Pentru a o evita va trebui să găsim un algoritm care să poată determina momentul în care cele două drumuri se intersectează.

Pentru aceasta, avem nevoie de nivelurile pe care se află extremitățile muchiei care închide ciclul. Există posibilitatea ca acestea să se afle pe acelaşi nivel dar, în majoritatea cazurilor, unul dintre cele două noduri se va afla pe un nivel inferior celui pe care se află celălalt. În această situație, pornind din acest nod vom urca în arbore până vom ajunge la nivelul celuilalt nod. Apoi, vom urca pe ambele drumuri până în momentul în care vom ajunge la un nod comun.

Aşadar, pentru acest algoritm avem nevoie atât de nivel, cât şi de părintele fiecărui nod. O variantă a acestui algoritm este următoarea:

```
Algoritm Identificare_Ciclu(x,y,părinte,nivel)
                                          { x, y – nodurile care închid ciclul }
                                          { părinte - şirul părinților }
                                          { nivel – şirul nivelurilor }
  nod1 ← x
  nod2 ← y
  cât timp nivel_nod1 < nivel_nod2 execută
    scrie nod1
    nod1 ← părinte_nod1
  sfârşit cât timp
  cât timp nivel_nod1 > nivel_nod2 execută
    scrie nod2
```

```
    nod2 ← părinte_nod2
  sfârşit cât timp
  cât timp nod1 ≠ nod2 execută
    scrie nod1, nod2
    nod1 ← părinte_nod1
    nod2 ← părinte_nod2
  sfârşit cât timp
  scrie nod1
sfârşit algoritm
```

Se observă foarte clar faptul că se execută cel mult unul dintre primele două cicluri **cât timp**. În cazul în care cele două extremități se află pe acelaşi nivel, nici unul dintre aceste cicluri nu se va executa.

În cazul acestui algoritm vom urca în arbore doar până la punctul de intersecție a drumurilor. Bineînțeles, în cazul cel mai defavorabil, vom urca tot de pe ultimul nivel până la rădăcină dar, de această dată, fiecare nod va fi vizitat cel mult o dată. Ordinul de complexitate este tot $O(n)$, dar în marea majoritate a cazurilor se execută mai puține operații.

Totuşi, acest algoritm necesită păstrarea vectorului nivelurilor care nu era necesar pentru algoritmul prezentat anterior.

Pentru a ilustra modul în care funcționează algoritmul descris, vom arăta modul în care se determină ciclul închis de muchia 3 – 4 în graful din figura 5.3. Nodul 3 se află pe al treilea nivel, iar nodul 4 se află pe al doilea. Aşadar, vom urca în arbore începând din nodul 3 până vom ajunge pe al doilea nivel. În acest caz se urcă un singur nivel, deci este afişat doar nodul 3. În acest moment, ne aflăm pe nivelul al doilea, " în dreptul" nodurilor 2 şi 4. Vom urca simultan pe ambele drumuri până ajungem la un nod comun. Pentru acest exemplu vom urca un singur nivel, iar nodul comun va fi rădăcina. Vor fi scrise nodurile parcurse pe cele două drumuri (2 pentru primul drum, respectiv 4 pentru al doilea), iar apoi va fi scris nodul comun: 1. În concluzie, ciclul închis de muchia 3 – 4 conține nodurile 1, 2, 3 şi 4.

5.4. Cicluri disjuncte

În cadrul acestei secțiuni vom introduce noțiunea de cicluri disjuncte şi vom prezenta un algoritm care poate fi utilizat pentru a determina un număr maxim de cicluri disjuncte într-un graf neorientat.

5.4.1. Noțiunea de cicluri disjuncte

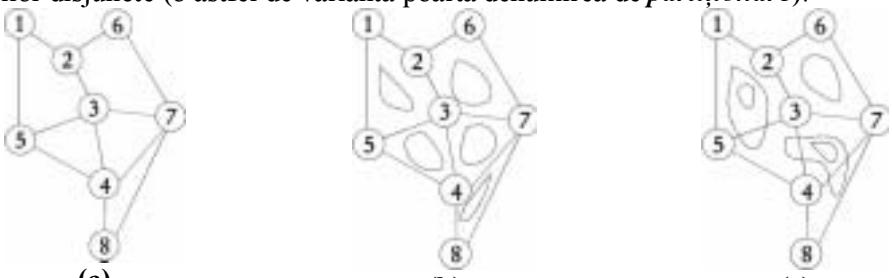
De obicei, se spune despre două sau mai multe entități (concepte) că sunt disjuncte dacă au o caracteristică care le diferențiază complet. Cu alte cuvinte, oricare entitate are

o caracteristică pe care nu o are nici una dintre celelalte. În cazul ciclurilor unui graf caracteristica este o muchie care nu apare în nici un alt ciclu.

Se numesc ***cicluri disjuncte*** o mulțime de cicluri care conțin, fiecare, cel puțin o muchie care nu apare în nici unul dintre celelalte cicluri din mulțime. Muchia respectivă poartă denumirea de ***muchie proprie*** a ciclului.

Datorită faptului că fiecare muchie care nu este de înaintare închide un ciclu, vom avea $m - n + 1$ cicluri disjuncte în oricare graf (m este numărul muchiilor, iar n este cel al nodurilor). În acest moment toate cele m muchii ale grafului aparțin cel puțin unui ciclu; aşadar, orice nou ciclu conține muchii care aparțin deja cel puțin unui alt ciclu, deci nu va putea avea o muchie proprie.

În concluzie, numărul maxim al ciclurilor disjuncte care pot fi identificate este $m - n + 1$. Evident, există mai multe variante de alegere a ciclurilor. Pentru graful din figura 5.4(a) sunt ilustrate în figurile 5.4 (b) şi 5.4(c) două posibilități diferite de alegere a ciclurilor disjuncte (o astfel de variantă poartă denumirea de ***partiționare***).



(a) **(b)** **(c)**

Figura 5.4: Două posibilități de partiționare a unui graf în cicluri disjuncte

5.4.2. Un algoritm de partiționare în cicluri disjuncte

Pentru a partiționa un graf în cicluri disjuncte va trebui, mai întâi, să determinăm un arbore parțial. Apoi vom lua în considerare toate muchiile care nu fac parte din acest arbore şi vom închide ciclurile corespunzătoare.

Cea mai convenabilă modalitate este folosirea arborelui *DF*, deoarece operația de închidere a ciclurilor este mai simplă. Toate muchiile care nu fac parte din arborele DF vor fi muchii de întoarcere. În momentul în care se va identifica o astfel de muchie, va fi determinat şi ciclul pe care îl închide. Versiunea în pseudocod a algoritmului este următoarea:

```
Subalgoritm ScrieCiclu(k,i,părinte)
                                          { k – nodul de pe nivelul inferior }
                                        { i – nodul de pe nivelul superior }
                                              { părinte - şirul părinților }
  nod ← k
  scrie nod
```

```
  cât timp i ≠ nod execută
    nod ← părinte_nod
    scrie nod
  sfârşit cât timp
sfârşit subalgoritm

Subalgoritm Identificare_Ciclu(k,G)
                                                { k – nodul curent }
                                                    { G – graful }
  vizitat_k ← adevărat
  pentru toţi vecinii i ai lui k execută
    dacă nu vizitat_i atunci
      părinte_i ← k
      Identificare_Ciclu(i,G)
    altfel
      dacă i ≠ părinte_k atunci
        ScrieCiclu(k,i,părinte)
      sfârşit dacă
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

Se observă că singura diferență față de algoritmul care identifică un ciclu folosind parcurgerea *DF* este eliminarea instrucțiunii de întrerupere a algoritmului.

5.4.3. Analiza complexității

Ordinul de complexitate al algoritmului de determinare a arborelui *DF* este $O(m + n)$. În plus, pentru fiecare dintre cele $m - n + 1$ muchii de întoarcere va trebuie să închidem un ciclu, operație al cărei ordin de complexitate este $O(n)$.

Aşadar, ordinul de complexitate al operației de determinare a unei mulțimi maximale de cicluri disjuncte este $O(m + n) + O(m - n) \cdot O(n) = O(m \cdot n - n^2)$.

5.5. Rezumat

În cadrul acestui capitol am prezentat modul în care pot fi etajate pe niveluri grafurile, folosind diferite parcurgeri. De asemenea, pentru o etajare am arătat modul în care pot fi determinate şirul părinților şi şirul nivelurilor.

În continuare am prezentat modul în care pot fi clasificate muchiile grafurilor în funcție de rolul lor în cadrul parcurgerii. Am definit muchiile de avansare, de întoarcere şi de traversare.

De asemenea, am prezentat modul în care pot fi identificate ciclurile unui graf, precum şi o modalitate de partiționare a grafului în cicluri disjuncte.

5.6. Implementări sugerate

Pentru a vă familiariza cu modul în care trebuie implementate rezolvările problemelor care se reduc la efectuarea unor operații care implică ciclurile unui graf vă sugerăm să încercați să implementați algoritmi pentru:

1. verificarea ciclicității unui graf;
2. determinarea unui ciclu folosind parcurgerea în adâncime a unui graf;
3. determinarea unui ciclu folosind parcurgerea în lățime a unui graf;
4. determinarea şirului părinților în cadrul unei parcurgeri în adâncime;
5. determinarea şirului nivelurilor în cadrul unei parcurgeri în adâncime;
6. determinarea şirului părinților în cadrul unei parcurgeri în lățime;
7. determinarea şirului nivelurilor în cadrul unei parcurgeri în lățime;
8. determinarea listei muchiilor de înaintare pentru o parcurgere în adâncime;
9. determinarea listei muchiilor de întoarcere pentru o parcurgere în adâncime;
10. determinarea listei muchiilor de înaintare pentru o parcurgere în lățime;
11. determinarea listei muchiilor de întoarcere şi a listei muchiilor de traversare pentru o parcurgere în lățime;
12. determinarea unui ciclu închis de o muchie de întoarcere;
13. determinarea unui ciclu închis de o muchie de traversare;
14. determinarea unei partiționări în cicluri disjuncte folosind parcurgerea în adâncime;
15. determinarea unei partiționări în cicluri disjuncte folosind parcurgerea în lățime.

Va trebui să găsiți implementări care să funcționeze atât pentru grafuri conexe, cât şi pentru grafuri neconexe.

5.7. 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.

5.7.1. Erathia

Descrierea problemei

În Erathia există un număr total de *N* castele între care se află un număr total de *M* drumuri. Regina *Catherine* doreşte să creeze un circuit format din cel puțin trei castele astfel încât oricare două castele consecutive în circuit să fie legate printr-un drum.

Date de intrare

Prima linie a fişierului de intrare **`ERATHIA.IN`** conține numărul *N* al castelelor şi numărul *M* al perechilor de castele între care există drumuri. Fiecare dintre următoarele

M linii va conține câte două numere întregi *x* şi *y* cu semnificația: există un drum între castelele identificate prin *x* şi *y*.

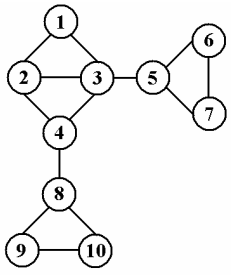
Date de ieşire
Fişierul de ieşire **ERATHIA.OUT** va conține o singură linie pe care se vor afla numerele de ordine ale castelelor care fac parte dintr-un circuit, separate printr-un spațiu, în ordinea în care sunt ele parcurse pentru a realiza circuitul.

Restricții şi precizări
- $1 \le N \le 100$;
- $1 \le M \le 1000$;
- castelele sunt identificate prin numere întregi cuprinse între 1 şi *N*;
- fiecare castel poate apărea o singură dată în cadrul circuitului;
- pe un drum se poate circula în ambele sensuri;
- va exista întotdeauna cel puțin un circuit;
- dacă există mai multe soluții trebuie generată doar una dintre ele.

Exemplu

ERATHIA.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
```

ERATHIA.OUT
```
1 2 3
```



Timp de execuție: 1 secundă/test

5.7.2. Copii

Descrierea problemei
Într-o şcoală există *K* clase formate dintr-un anumit număr de copii (numărul copiilor nu este acelaşi pentru toate clasele). Pentru fiecare clasă se cunosc toate perechile de prieteni. Se ştie că dacă un copil primeşte o informație, atunci el o va transmite

imediat tuturor prietenilor săi, cu excepția celui de la care a primit-o. În plus, oricare ar fi copilul care intră primul în posesia informației, se ştie că aceasta va ajunge, până la urmă, la toți elevii din clasa respectivă. Pentru fiecare clasă trebuie să se verifice dacă există cel puțin un elev care, dacă primeşte primul o informație (de la un profesor), atunci informația respectivă se va "întoarce" la el.

Date de intrare
Prima linie a fişierului de intrare **COPII.IN** conține numărul K al claselor. În continuare sunt prezentate informațiile referitoare la cele K clase. Prima linie corespunzătoare unei clase conține numărul N al elevilor şi numărul M al perechilor de prieteni din clasa respectivă. Fiecare dintre următoarele M linii va conține câte două numere întregi x şi y cu semnificația: copiii identificați prin x şi y sunt prieteni.

Date de ieşire
Fişierul de ieşire **COPII.OUT** va conține K linii, câte una pentru fiecare clasă. Linia corspunzătoare unei clase va conține mesajul DA în cazul în care există cel puțin un elev care respectă condiția dată sau mesajul NU în caz contrar.

Restricții şi precizări
- $1 \le K \le 100$;
- $1 \le N \le 100$;
- $1 \le M \le 1000$;
- copiii sunt identificați prin numere întregi cuprinse între 1 şi N;
- dacă x şi y sunt prieteni, atunci x poate transmite informații lui y, dar şi y poate transmite informații lui x;
- informațiile din fişierul de ieşire vor respecta ordinea în care sunt descrise clasele în fişierul de intrare.

Exemplu

COPII.IN	COPII.OUT
2 3 3 1 2 1 3 2 3 4 3 1 2 2 3 2 4	DA NU

Timp de execuție: 1 secundă/test

5.7.3. Formula 1

Descrierea problemei
Din nefericire, s-a descoperit faptul că echipele participante la concursurile de Formula 1 au "datorii" unele față de altele. "Datoriile" sunt cauzate de anumite favoruri pe care şi le-au acordat echipele (una alteia) în anii precedenți.

Evident, dacă o echipă A are o datorie față de o echipă B şi echipa B are o datorie față de echipa C, atunci datoria se poate transfera de la A la C. Ziariştii doresc să afle dacă s-a ajuns în situație hilară în care, datorită numărului mare de favoruri, o echipă ajunge în situația să fie datoare ei însăşi. În acest scop ei au ales, pentru un studiu de caz, echipa *Ferrari*.

Aşadar, va trebui să verificați dacă există un "lanț al datoriilor" care porneşte de la *Ferrari* şi ajunge la aceeaşi echipă. În cazul în care un astfel de lanț există, el va trebui determinat.

Date de intrare
Prima linie a fişierului de intrare **`FORMULA1.IN`** conține numărul N al echipelor participante şi numărul M al datoriilor aduse la cunoştința ziariştilor. Fiecare dintre următoarele M linii va conține câte două şiruri de caractere x şi y cu semnificația: echipa x este datoare echipei y. Cele două şiruri de caractere de pe o linie vor fi separate printr-un spațiu.

Date de ieşire
Fişierul de ieşire **`FORMULA1.OUT`** va conține mesajul `NU` în cazul în care nu există un lanț cu proprietatea descrisă şi `DA` în caz contrar. În cazul în care un astfel de lanț există, numele echipelor care îl formează (fără a include echipa *Ferrari*) vor fi scrise pe următoarele linii, în ordinea în care apar acestea în lanț.

Restricții şi precizări
- $2 \le N \le 100$;
- $1 \le M \le 1000$;
- echipa *Ferrari* va fi întotdeauna prezentă în fişier;
- echipele sunt identificate prin şiruri formate din cel mult 100 de caractere printre care nu se află spații; se face distincție între literele mici şi cele mari;
- dacă o echipă x este datoare unei echipe y, atunci este posibil (dar nu obligatoriu) ca şi echipa y să fie datoare echipei x.

Exemplu

`FORMULA1.IN`
```
5 8
Ferrari Williams
```

`FORMULA1.OUT`
```
DA
Williams
```

```
Williams Jordan                Jordan
Williams McLaren               Minardi
McLaren Jordan
Jordan Minardi
Minardi Williams
Minardi Ferrari
Minardi McLaren
```

Timp de execuție: 1 secundă/test

5.7.4. Trasee

Descrierea problemei
Într-un oraş există N obiective strategice legate printr-un număr total de M străzi pe care se poate circula în ambele sensuri. Primăria doreşte să organizeze un serviciu de pază în cadrul căruia se stabilesc traseele parcurse de "oamenii legii".

Un traseu va fi format din mai multe obiective distincte (cel puțin trei) astfel încât două obiective aflate pe poziții consecutive sunt legate direct printr-o stradă. În plus, primul şi ultimul obiectiv al traseului trebuie să fie şi ele legate printr-o stradă.

Se doreşte stabilirea unui număr maxim de astfel de trasee în aşa fel încât pentru parcurgerea fiecărui traseu să fie necesară utilizarea unei străzi care nu este utilizată pentru nici unul dintre celelalte trasee.

Date de intrare
Prima linie a fişierului de intrare **`TRASEE.IN`** conține numărul N al obiectivelor strategice şi numărul M al străzilor. Fiecare dintre următoarele M linii va conține câte două numere x şi y cu semnificația: între obiectivele x şi y există o stradă. Numerele de pe o linie vor fi separate printr-un spațiu.

Date de ieşire
Prima linie a fişierului de ieşire **`TRASEE.OUT`** va conține numărul maxim K al traseelor care pot fi stabilite. Pe fiecare dintre următoarele K linii va fi descris câte un traseu. Primul număr de pe o astfel de linie reprezintă numărul t al obiectivelor care fac parte din traseul respectiv, iar următoarele t numere reprezintă obiectivele de pe traseu, în ordinea în care acestea ar trebui parcurse. Numerele de pe o linie vor fi separate prin câte un spațiu.

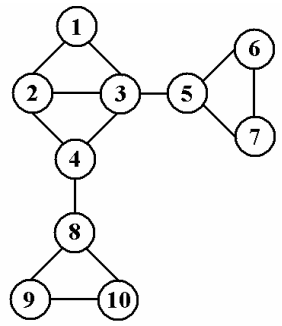
Restricții şi precizări
- $3 \le N \le 100$;
- $1 \le M \le 1000$;
- există cel mult o stradă între oricare două obiective;

- traseele pot fi descrise în orice ordine;
- dacă există mai multe soluții va fi generată doar una dintre ele;
- obiectivele strategice sunt identificate prin numere întregi cuprinse între 1 şi *N*.

Exemplu

TRASEE.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
```

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



Timp de execuție: 1 secundă/test

5.8. 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.

5.8.1. Erathia

Castelele din Erathia pot fi considerate a fi nodurile unui graf neorientat ale cărui muchii sunt date de drumurile dintre acestea.

Astfel, problema se reduce la identificarea unui ciclu într-un graf. Pentru aceasta putem utiliza algoritmul de determinare a ciclurilor şi ne vom opri în momentul în care este identificat primul ciclu. Enunțul problemei ne asigură că graful va conține întotdeauna cel puțin un ciclu.

După identificarea ciclului vom scrie în fişierul de ieşire numerele de ordine ale nodurilor care formează ciclul identificat.

Analiza complexității

Citirea datelor de intrare implică citirea extremităților celor M muchii ale grafului, operație al cărei ordin de complexitate este **O(M)**. Pe parcursul citirii vom crea şi reprezentarea grafului, operație care nu consumă timp suplimentar.

În continuare va trebui să identificăm un ciclu, operație care implică, în cel mai defavorabil caz, efectuarea unei parcurgeri complete în adâncime a grafului. Aceasta are ordinul de complexitate **O(M + N)**.

Datele de ieşire constau în numerele de ordine ale nodurilor care formează un ciclu. Ciclul poate fi format, în cel mai defavorabil caz, din toate cele N noduri ale grafului, deci operația de scriere a rezultatului în fişierul de ieşire are ordinul de complexitate **O(N)**.

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

5.8.2. Copii

Putem privi fiecare clasă ca fiind un graf neorientat în care nodurile reprezintă elevii, iar muchiile reprezintă perechile de prieteni. Aşadar, vom avea K grafuri, câte unul pentru fiecare clasă.

O informație se va putea întoarce la un anumit elev dacă şi numai dacă nodul corespunzător elevului face parte dintr-un ciclu.

Ca urmare, problema se reduce la verificarea ciclicității celor K grafuri care caracterizează clasele.

Vom construi pe rând cele K grafuri unul după altul (nu este necesar să le păstrăm pe toate în memorie deoarece operațiile efectuate asupra unui graf sunt independente de cele efectuate asupra celorlalte).

Pentru a verifica ciclicitatea unui graf este suficient să utilizăm algoritmul prezentat în cadrul acestui capitol. După verificare vom scrie în fişierul de ieşire mesajul corespunzător.

Analiza complexității

Citirea datelor de intrare implică citirea, pentru fiecare dintre cele K grafuri, a extremităților celor M muchii ale grafului. Aşadar, pentru un anumit graf, operația de citire a datelor referitoare la acesta are ordinul de complexitate $O(M)$. Pe parcursul citirii vom crea şi reprezentarea grafului; această operație nu consumă timp suplimentar. Datorită faptului că vom efectua, în total, K astfel de citiri şi vom crea K astfel de grafuri, ordinul de complexitate al operației de citire este **O(K · M)**.

În continuare va trebui să verificăm, pentru fiecare graf, dacă acesta este sau nu ciclic. Verificarea ciclicității unui graf se realizează, aşa cum am arătat în cadrul acestui capitol, într-un timp având ordinul de complexitate $O(M + N)$. Vom realiza K astfel de verificări, deci ordinul de complexitate al operației de verificare a ciclicității tuturor grafurilor este **O(K · (M + N))**.

Datele de ieşire constau într-un singur mesaj pentru fiecare dintre grafuri. Ordinul de complexitate al operației va fi $O(1)$ pentru fiecare graf şi, datorită faptului că vom scrie K astfel de mesaje, ordinul de complexitate al operației de scriere a datelor de ieşire va fi **O(K)**.

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

5.8.3. Formula 1

Denumirile echipelor de Formula 1 vor fi considerate vârfurile unui graf orientat. Va exista un arc de la un vârf x la un vârf y dacă şi numai dacă echipa corespunzătoare nodului x are o datorie față de echipa corespunzătoare nodului y.

În aceste condiții este suficient să verificăm dacă nodul corespunzător echipei *Ferrari* face sau nu parte dintr-un circuit. În cazul în care un astfel de circuit există, vor fi scrise în fişierul de ieşire denumirile echipelor cărora le corespund nodurile care formează circuitul.

Pentru a determina un circuit într-un graf orientat putem aplica acelaşi algoritm (bazat pe o parcurgere în adâncime) folosit pentru determinarea ciclurilor în grafuri orientate.

Datorită faptului că echipele nu sunt identificate prin numere, ci prin denumirile lor, pentru a putea construi graful, va trebui să realizăm o codificare a acestora. Prima echipă întâlnită în fişierul de intrare va fi identificată prin 1, cea de-a doua prin 2 etc. În momentul în care se citeşte denumirea unei echipe, se verifică dacă există deja o codificare pentru ea; în cazul în care există o codificare (denumirea a mai apărut în fişier) se va returna codul stabilit anterior; în caz contrar se va genera un nou cod.

Analiza complexității

Citirea datelor de intrare implică citirea extremităților celor M muchii ale grafului; pentru fiecare muchie citită va trebui să identificăm codurile corespunzătoare denumirilor. Această operației constă în cel mai defavorabil caz, în parcurgerea întregii liste de coduri care va conține cel mult N elemente. În cazul în care nu a fost încă stabilit un cod, acesta este generat în timp constant. Aşadar, vom efectua două astfel de parcurgeri, pentru o muchie, deci operația de codificare pentru o muchie are ordinul de complexitate $O(N) + O(N) = O(N)$. Pe parcursul citirii vom crea şi reprezentarea grafului, operație care nu consumă timp suplimentar. Datorită faptului că vom codifica M muchii, ordinul de complexitate al tuturor operațiilor necesare construirii grafului este $\mathbf{O(M \cdot N)}$.

Teoretic, există posibilitatea de a păstra codurile folosind o structură avansată de date care permite regăsirile codurilor şi adăugările lor în timp logaritmic. Totuşi, folosirea unei astfel de structuri nu este necesară în cazul de față.

După construirea grafului va trebui să identificăm nodul corespunzător echipei *Ferrari* şi să verificăm dacă acesta face sau nu parte dintr-un circuit. Această operaţie are ordinul de complexitate **O(M + N)**.

După eventuala identificare a circuitului, va trebui să scriem denumirile echipelor corespunzătoare nodurilor care îl formează. Pe baza codului, vom putea regăsi denumirea unei echipe în timp constant (un cod reprezintă poziţia denumirii într-un vector). Ca urmare, operaţia de scriere a datelor de ieşire are ordinul de complexitate **O(N)** în cazul în care este identificat un circuit şi **O(1)** în caz contrar (în fişier se va scrie doar mesajul `NU`).

În concluzie, pentru cazul cel mai defavorabil, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(M \cdot N) + O(M + N) + O(N) = \mathbf{O(M \cdot N)}$.

5.8.4. Trasee

Cele N obiective strategice vor reprezenta nodurile unui graf neorientat ale cărui muchii vor fi date de cele M străzi.

Astfel, problema se reduce la determinarea unei partiţionări în cicluri disjuncte pentru graful construit. Numărul ciclurile disjuncte va fi întotdeauna $M - N + 1$.

Pe măsură ce identificăm aceste cicluri le vom scrie în fişierul de ieşire.

Analiza complexităţii

Citirea datelor de intrare implică citirea extremităţilor celor M muchii ale grafului, operaţie al cărei ordin de complexitate este **O(M)**. Pe parcursul citirii vom crea şi reprezentarea grafului, operaţie care nu consumă timp suplimentar.

În continuare va trebui să identificăm ciclurile disjuncte ale grafului, operaţie al cărei ordin de complexitate este $\mathbf{O(M \cdot N - N^2)}$. Pe măsura identificării acestora, ele vor fi scrise în fişierul de ieşire, deci generarea fişierului de ieşire nu va consuma timp suplimentar.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(M) + O(M \cdot N - N^2) = \mathbf{O(M \cdot N - N^2)}$.