# Noțiuni elementare

Capitolul 3

- **Definiții**
- **Reprezentarea grafurilor**
- **Traversarea grafurilor**
- **Arbori parțiali minimi**
- **Drumuri minime**
- **Rezumat**
- **Implementări sugerate**
- **Probleme propuse**
- **Soluțiile problemelor**

În cadrul acestui capitol vom recapitula unele noțiuni elementare referitoare la teoria grafurilor. Nu vom prezenta detaliat aceste noțiuni deoarece ele au fost studiate în clasa a X-a; vom descrie succint doar elementele necesare înțelegerii informațiilor prezentate în capitolele anterioare.

## 3.1. Definiții

Această secțiune este dedicată prezentării definițiilor noțiunilor elementare ale teoriei grafurilor. Vom prezenta noțiunile de graf neorientat şi orientat, drum, lanț, ciclu, circuit, traversare a grafurilor, graf parțial, subgraf, conexitate, componentă conexă, arbore parțial minim, drum minim etc.

### 3.1.1. Grafuri

Un ***graf neorientat*** este definit, din punct de vedere matematic, ca fiind o pereche de mulțimii $G = (U, V)$ unde $U$ este mulțimea vârfurilor, iar $V$ este mulțimea muchiilor. Elementele mulțimii $V$ sunt perechi neordonate de elemente din mulțimea $U$.

Grafic, nodurile unui graf sunt reprezentate prin cercuri, iar muchiile prin linii care unesc aceste cercuri. Un graf neorientat cu 10 noduri şi 13 muchii este prezentat în figura 3.1.

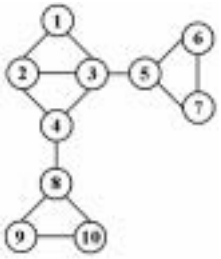



**Figura 3.1:** Graf neorientat

Un ***graf orientat*** este definit ca fiind o pereche de mulțimii $G = (U, V)$ unde $U$ este mulțimea vârfurilor, iar $V$ este mulțimea arcelor. Elementele mulțimii $V$ sunt perechi ordonate de elemente din mulțimea $U$.

Grafic, nodurile unui graf sunt reprezentate prin cercuri, iar arcele prin linii care unesc aceste cercuri. Un graf orientat cu 10 noduri şi 13 muchii este prezentat în figura 3.2.

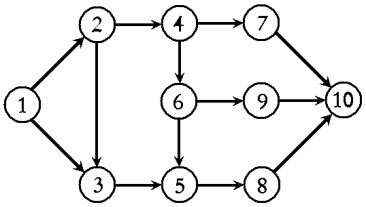


**Figura 3.2:** Graf orientat

Cea mai importantă diferență dintre grafurile orientate şi cele neorientate este dată, practic, de faptul că în cazul grafurilor orientate arcele sunt direcționate. Despre o muchie a unui graf neorientat vom spune că ***uneşte*** două vârfuri, iar pentru un arc al unui graf orientat vom spune că ***pleacă*** de la un vârf şi ***ajunge*** la un alt vârf.

Vârfurile grafurilor poartă şi denumirea de ***noduri***, cele două noțiuni fiind echivalente.

Pentru grafurile neorientate este definită noțiunea de ***grad*** al unui vârf care reprezintă numărul de muchii care unesc vârful respectiv de alte vârfuri. De exemplu, nodul 3 din graful neorientat din figura 3.1. are gradul 3.

Pentru grafurile orientate sunt definite noțiunile de ***grad interior*** şi ***grad exterior***. Gradul interior al unui vârf indică numărul de arce care ajung la vârful respectiv, iar gradul exterior indică numărul de arce care pleacă de la vârful respectiv. Aşadar, despre nodul 3 al grafului orientat din figura 3.2 se poate spune că are gradul interior 2 şi gradul exterior 1.

Atât pentru grafurile neorientate cât şi pentru cele orientate este definită noțiunea de ***graf parțial***. Un graf parțial al unui graf este obținut prin eliminarea unor muchii ale grafului inițial. În figura 3.3 este ilustrat un graf parțial al grafului din figura 3.1 obținut prin eliminarea muchiilor dintre nodurile 1 şi 2, 5 şi 7, respectiv 9 şi 10.

De asemenea, poate fi definită noțiunea de ***subgraf*** ca fiind un graf obținut prin eliminarea unor noduri şi a tuturor muchiilor (arcelor) adiacente cu acestea. În figura 3.4 este prezentat subgraful obținut prin eliminarea vârfurilor 2, 7 şi 10 şi a tuturor muchiilor adiacente.

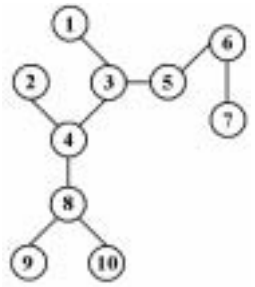


**Figura 3.3:** Graf parțial

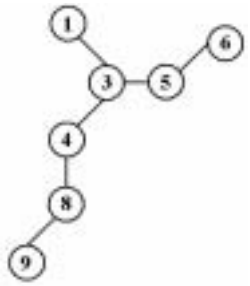


**Figura 3.4:** Subgraf

## 3.1.2. Alte elemente

Un ***lanț*** într-un graf neorientat este definit ca o succesiune de noduri, oricare două noduri consecutive fiind unite printr-o muchie. Pentru graful din figura 3.1 unul dintre lanțuri este 1 – 3 – 4 – 2 – 3 – 5. Un ***lanț elementar*** este un lanț în care fiecare nod apare o singură dată. Pentru acelaşi graf, un lanț elementar este 1 – 2 – 4 – 8 – 10.

Un ***ciclu*** poate fi definit ca fiind un lanț în care primul şi ultimul nod al succesiunii sunt identice. Pentru graful considerat un ciclu ar putea fi 1 – 2 – 3 – 4 – 2 – 3 – 1.
Un ***ciclu elementar*** este un ciclu în care fiecare nod apare o singură dată cu excepția primului nod care apare întotdeauna de două ori (pe prima şi pe ultima poziție). Un ciclu elementar al grafului considerat este 5 – 6 – 7 – 5.

Pentru grafurile orientate există noțiunile de *drum*, *drum elementar*, *circuit* şi *circuit elementar* definite asemănător.

Un graf neorientat care nu conține nici un ciclu poartă denumirea de ***arbore***. Grafurile orientate care nu conțin circuite se numesc ***grafuri orientate aciclice***.

Un ***arbore parțial*** este un graf parțial care este arbore.

O ***traversare*** a unui graf neorientat poate fi definită ca fiind o parcurgere a grafului pornind de la un anumit nod şi vizitând toate celelalte noduri parcurgând muchiile grafului.

Un graf neorientat este ***conex*** dacă şi numai dacă pentru fiecare pereche de noduri există un lanț care are ca extremități cele două noduri.

O ***componentă conexă*** a unui graf este un subgraf care este conex.

## 3.1.3. Costuri asociate muchiilor şi arcelor

Există posibilitatea de a asocia ***costuri*** muchiilor unui graf orientat sau neorientat. În acest caz muchiile şi arcele vor fi caracterizate prin cele două extremități şi un cost. Pentru costuri mai este folosită denumirea de ponderi. În figura 3.5 este ilustrat graful orientat din figura 3.2 după asocierea unor costuri pentru fiecare arc.

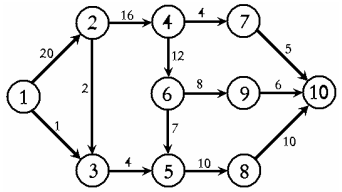



**Figura 3.5:** Graf orientat cu costuri

După asocierea costurilor pentru muchii sau arce pot fi definite noțiunile de cost al unui lanț, drum, ciclu, circuit, graf parțial subgraf etc. ca fiind suma costurilor muchiilor care fac parte din elementele respective. De exemplu, pentru graful din figura 3.5 costul drumului 1 – 2 – 4 – 6 – 9 – 10 este 62.

Pot fi definite şi noțiunile de drum minim, arbore parțial minim etc. ca fiind elementul din categoria respectivă care are cel mai mic cost. Pentru graful considerat drumul minim de la nodul 1 la nodul 10 este 1 – 3 – 5 – 8 – 10 şi are costul 25.

Dacă nu se asociază costuri muchiilor sau arcelor, în funcție de situație, se poate considera că toate costurile sunt 1, toate costurile sunt 0 etc.

# 3.2. Reprezentări ale grafurilor

Există numeroase posibilități de reprezentare a grafurilor. Cea mai simplă dintre ele este cea geometrică, folosită până acum în cadrul acestui capitol. În această secțiune vom prezenta cele mai folosite moduri de reprezentare a grafurilor în memoria calculatorului cum ar fi matricea de adiacență, lista muchiilor, matricea succesorilor şi lista succesorilor.

## 3.2.1. Matricea de adiacență

Cea mai simplă şi mai comodă posibilitate de reprezentare a grafurilor neorientate este cu ajutorul ***matricei de adiacență***.

Pentru un graf cu $N$ noduri, matricea de adiacență $A$ are $N$ linii şi $N$ coloane. Elementul $A_{ij}$ va avea valoarea 1 dacă există o muchie între nodurile $i$ şi $j$ şi 0 în caz contrar. În figura 3.6 este prezentată matricea de adiacență corespunzătoare grafului din figura 3.1.

```
0 1 1 0 0 0 0 0 0 0
1 0 1 1 0 0 0 0 0 0
1 1 0 1 1 0 0 0 0 0
0 1 1 0 0 0 0 1 0 0
0 0 1 0 0 1 1 0 0 0
0 0 0 0 1 0 1 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 0 1 0 0 0 0 1 1
0 0 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 1 1 0
```

**Figura 3.6:** Matrice de adiacență pentru un graf neorientat

Se observă că matricea este simetrică față de diagonala principală şi că toate elementele de pe această diagonală au valoarea 0.

Pentru un graf orientat un element $A_{ij}$ al matricei de adiacență va avea valoarea 1 dacă există un arc de la nodul $i$ la nodul $j$ şi 0 în caz contrar. Matricea de adiacență corespunzătoare grafului din figura 3.2 este prezentată în figura 3.7.

În anumite situații, dacă există un arc de la nodul $i$ la nodul $j$, atunci valoarea elementului este -1. Această variantă este posibilă dacă nu există două noduri $i$ şi $j$ astfel încât să existe atât un arc de la nodul $i$ la nodul $j$, cât şi un arc de la nodul $j$ la nodul $i$. Dacă există o astfel de pereche, atunci elementul $A_{ij}$ ar trebui să aibă atât valoarea 1 cât şi valoarea -1, ceea ce este imposibil.

```
0 1 1 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
```

**Figura 3.7:** Matrice de adiacență pentru un graf orientat

Se observă că, în cazul grafurilor orientate matricea de adiacență nu mai este simetrică, dar elementele de pe diagonala principală au şi în acest caz valoarea 0.

În cazul în care muchiile sau arcele au costuri asociate matricea de adiacență va fi transformată într-o ***matrice a costurilor***. Valoarea unui element $A_{ij}$ nu va mai fi 1, ci va fi egală cu ponderea muchiei sau arcului corespunzător.

Elementele corespunzătoare muchiilor sau arcelor inexistente vor avea o valoare aleasă în aşa fel încât să nu afecteze operațiile efectuate asupra matricei. Cele mai folosite valori sunt 0 şi $\infty$ (această valoare este teoretică, în practică se foloseşte un număr suficient de mare).

În figura 3.8 este prezentată matricea costurilor corespunzătoare grafului din figura 3.5.

Se observă că o matrice de adiacență poate fi privită ca fiind o matrice a costurilor în care toate muchiile sau arcele existente au costul 1 şi se foloseşte valoarea 0 pentru muchiile şi arcele inexistente.

Principalul avantaj al utilizării matricelor de adiacență, respectiv al utilizării matricelor costurilor, îl reprezintă faptul că poate fi verificată foarte uşor existența unei muchii între două noduri sau a unui arc de la un nod la altul.

```
0 20  1  0  0  0  0  0  0  0
0  0  2 16  0  0  0  0  0  0
0  0  0  0  4  0  0  0  0  0
0  0  0  0  0 12  4  0  0  0
0  0  0  0  0  0  0 10  0  0
0  0  0  0  7  0  0  0  8  0
0  0  0  0  0  0  0  0  0  5
0  0  0  0  0  0  0  0  0 10
0  0  0  0  0  0  0  0  0  6
0  0  0  0  0  0  0  0  0  0
```

**Figura 3.8:** O matrice a costurilor

Principalul dezavantaj al acestei metode de reprezentare îl reprezintă faptul că determinarea tuturor vecinilor unui nod necesită parcurgerea unei întregi linii a matricei.

## 3.2.2. Lista de muchii

O altă posibilitatea de reprezentare a grafurilor o constituie păstrarea unei liste care conține muchiile sau arcele grafului. Pentru un graf cu $M$ muchii, de obicei, sunt folosiți doi vectori cu $M$ elemente $x$ şi $y$ cu semnificația *există o muchie care uneşte vârfurile $x_i$ şi $y_i$, respectiv există un arc care pleacă de la nodul $x_i$ şi ajunge la nodul $y_i$.*

De exemplu, pentru graful neorientat din figura 3.1 vom avea şirurile `x=(1,1,2,2,3,3,4,5,5,6,8,8,9)` şi `y=(2,3,3,4,4,5,8,6,7,7,9,10,10)`, iar pentru graful neorientat din figura 3.2 vom avea şirurile `x=(1,1,2,2,3,4,4,5,6,6,7,8,9)` şi `y=(2,3,3,4,5,6,7,8,5,9,10,10,10)`.

În cazul în care muchiile sau arcele au costuri asociate, trebuie păstrat un vector suplimentar $c$ ale cărui elemente vor fi aceste costuri. De exemplu, pentru graful din figura 3.5 vom avea şirul `c=(20,1,2,16,4,12,4,10,7,8,5,6,10)`.

Există şi varianta folosirii unui vector ale cărui elemente să fie articole care conțin informații referitoare la muchii (arce). Dacă nu există costuri, articolele vor avea două câmpuri care vor conține extremitățile unui muchii sau ale unui arc. În cazul în care există şi costuri, articolele vor avea un câmp suplimentar care va conține costul muchiei (arcului).

## 3.2.3. Liste de vecini

O a treia posibilitate de reprezentare a grafurilor îl reprezintă listele de vecini pentru grafurile neorientate, respectiv listele de succesori pentru grafurile orientate.

În cazul grafurilor orientate vom păstra pentru fiecare nod în parte o listă care va conține nodurile care sunt unite printr-o muchie de nodul respectiv. De obicei se utilizează o matrice ale cărei linii reprezintă listele de vecini ale nodurilor. Datorită faptului că într-un graf cu $N$ noduri un vârf poate avea cel mult $N - 1$ vecini, matricea va

avea $N - 1$ coloane. Evident, nu toate nodurile vor avea $N - 1$ vecini; ca urmare, liniile corespunzătoare unor astfel de noduri vor fi completate cu zerouri. Deseori este necesară cunoaşterea numărului de vecini ai unui anumit nod, motiv pentru care, de obicei, se păstrează un vector suplimentar care va conține, practic, gradele nodurilor.

Principalul dezavantaj al folosirii unei astfel de liste îl reprezintă irosirea spațiului de memorie în cazurile în care gradele nodurilor sunt mici. Pentru a elimina acest inconvenient în locul folosirii unei astfel de matrice se utilizează un vector de liste alocate dinamic. În figura 3.9 sunt prezentate listele de vecini corespunzătoare grafului din figura 3.1.

În cazul grafurilor neorientate, pentru fiecare nod în parte va fi păstrată o listă a nodurilor spre care pleacă un arc de la nodul respectiv. Din nou poate fi folosită o matrice sau un vector de liste. Listele succesorilor pentru graful din figura 3.2 sunt prezentate în figura 3.10.

*Nodul* `1` *are doi vecini*: `2 3`
*Nodul* `2` *are trei vecini*: `1 3 4`
*Nodul* `3` *are patru vecini*: `1 2 4 5`
*Nodul* `4` *are trei vecini*: `2 3 8`
*Nodul* `5` *are trei vecini*: `3 6 7`
*Nodul* `6` *are doi vecini*: `5 7`
*Nodul* `7` *are doi vecini*: `5 6`
*Nodul* `8` *are trei vecini*: `4 9 10`
*Nodul* `9` *are doi vecini*: `8 10`
*Nodul* `10` *are doi vecini*: `8 9`

**Figura 3.9:** Liste de vecini

*Nodul* `1` *are doi succesori*: `2 3`
*Nodul* `2` *are doi succesori*: `3 4`
*Nodul* `3` *are un succesor*: `5`
*Nodul* `4` *are doi succesori*: `6 7`
*Nodul* `5` *are un succesor*: `8`
*Nodul* `6` *are doi succesori*: `5 9`
*Nodul* `7` *are un succesor*: `10`
*Nodul* `8` *are un succesor*: `10`
*Nodul* `9` *are un succesor*: `10`
*Nodul* `10` *nu are nici un succesor.*

**Figura 3.10:** Liste de succesori

În cazul în care muchiile sau arcele au costuri trebuie păstrat o matrice suplimentară sau un vector de liste suplimentar care conțin, în locul vecinului sau al succesorului, costul muchiei sau al arcului către vecin sau către succesor. De exemplu, pentru nodul 1 al grafului din figura 3.5 lista succesorilor va fi (2, 3), iar cea a costurilor va fi (20, 1).

Uneori, în cazul grafurilor orientate este utilă păstrarea unei liste a predecesorilor. Pentru fiecare nod $i$ această listă va conține toate nodurile de la care pleacă un arc spre nodul $i$.

## 3.2.4. Şirul vecinilor

Ultima posibilitate de reprezentare a grafurilor pe care o vom prezenta constă în şirul vecinilor, respectiv şirul succesorilor.

Acest şir se obține prin concatenarea listelor de vecini sau succesori. Se păstrează un vector suplimentar $p$ (numit ***vectorul pozițiilor***) care conține indicii din şirul vecinilor (succesorilor) la care încep vecinii (succesorii) fiecărui nod. Aşadar, vecinii (succesorii) unui nod i se vor afla pe pozițiile cuprinse între $p_i$ şi $p_{i+1} - 1$.

Pentru un graf neorientat cu $N$ noduri şi $M$ muchii şirul vecinilor va avea $2 \cdot N$ elemente deoarece pentru fiecare muchie dintre două noduri $i$ şi $j$, $i$ va apărea ca vecin al lui $j$ şi $j$ va apărea ca vecin al lui $i$. Aşadar, pentru fiecare muchie vom avea două elemente în şirul vecinilor.

Vectorul $p$ va avea $N + 1$ elemente, ultimul dintre acestea având valoarea $2 \cdot M + 1$ (elementul suplimentar apare pentru a indica faptul că succesorii nodului $N$ se află pe poziţii cuprinse între $p_N$ şi $2 \cdot M$; nu este absolut necesară adăugarea acestui element, dar folosirea sa duce la o mai mare uşurinţă în utilizarea şirului vecinilor).

Dacă un nod $i$ nu are nici un vecin (este un nod) izolat atunci vom avea $p_i = p_{i+1}$. Cu alte cuvinte vecinii nodului $i$ se află între poziţiile $p_i$ şi $p_{i-1}$, adică nu există nici un vecin.

Pentru grafurile orientate şirul succesorilor va avea doar $M$ elemente deoarece pentru fiecare arc există un singur succesor. Din aceste motive, valoarea elementului suplimentar $p_{N+1}$ va fi $M + 1$.

De exemplu, pentru graful din figura 3.1 şirul vecinilor este:

```
2 3 1 3 4 1 2 4 5 2 3 8 3 6 7 5 7 5 6 4 9 10 8 10 8 9,
```

în timp ce vectorul poziţiilor este:

```
1 3 6 10 13 16 18 20 23 25 27.
```

Pentru graful din figura 3.2 şirul succesorilor este:

```
2 3 3 4 5 6 7 8 5 9 10 10 10,
```

în timp ce vectorul poziţiilor este:

```
1 3 5 6 8 9 11 12 13 14 14.
```

Şi pentru aceste şiruri, în cazul în care muchiile sau arcele au costuri există posibilitatea de a păstra şiruri suplimentare care conţin aceste costuri.

De asemenea, pentru grafurile orientate există posibilitatea de a păstra un şir al predecesorilor, obţinut prin concatenarea listelor predecesorilor.

# 3.3. Parcurgerea grafurilor

Deşi există o mulţime de modalităţi de traversare a nodurilor grafurilor, două dintre acestea sunt mai importante datorită numeroaselor situaţii în care este necesară utilizarea lor. În cadrul acestei secţiuni vom prezenta succint aceste două tipuri de parcurgeri (traversare în lăţime şi traversare în adâncime) şi vom descrie algoritmi eficienţi care pot fi utilizaţi pentru realizarea acestor tipuri de parcurgeri.

## 3.3.1. Parcurgere în lăţime

Cunoscută mai ales sub denumirea de parcurgere ***BF*** (***B****readth* ***F****irst*) parcurgerea lăţime constă în alegerea unui nod de pornire, vizitarea sa şi apoi vizitarea tuturor vecinilor săi care nu au fost vizitaţi. După vizitarea vecinilor fiecare dintre aceştia devine, pe rând, nod de pornire şi se aplică acelaşi algoritm pentru nodul respectiv.

O posibilitate de implementare constă în păstrarea unei liste de tip coadă care va conține nodurile traversate. Inițial coada va conține doar nodul de pornire. La fiecare pas, în coadă sunt adăugați vecinii nevizitați ai nodului din capul cozii şi apoi este eliminat capul cozii. Teoretic, algoritmul se opreşte în momentul în care coada este vidă. Practic, algoritmul poate fi oprit în momentul în care au fost vizitate toate nodurile grafului.

Parcurgerea în lățime poate fi utilizată atât pentru grafuri orientate cât şi pentru grafuri neorientate.

Pentru grafurile neorientate, în cazul în care acestea nu sunt conexe, utilizarea algoritmului descris nu poate duce la vizitarea tuturor nodurilor. În această situație, în momentul în care coada devine vidă şi există noduri care nu au fost vizitate încă, unul dintre nodurile respective devine nod de pornire şi algoritmul este executat din nou. Astfel este traversată o nouă componentă conexă. Procedura continuă până în momentul în care au fost vizitate toate nodurile (aşadar, sunt vizitate toate componentele conexe). Aşadar, o astfel de parcurgere poate fi utilizată pentru determinarea componentelor conexe ale unui graf.

## 3.3.2. Parcurgere în adâncime

Diferența esențială dintre parcurgerea în lățime şi cea în adâncime constă în modul şi momentul în care sunt vizitați vecinii nodului de pornire. Dacă în cazul parcurgerii BF sunt vizitați toți vecinii şi abia apoi este modificat nodul de pornire, în cazul parcurgerii DF (Depth First) este vizitat un vecin şi acesta devine nod de pornire. După încheierea execuției algoritmului pentru acest fiu, este vizitat următorul fiu (dacă acesta nu a fost vizitat anterior) şi acesta devine nod de pornire.

Şi în cazul parcurgerii în adâncime, algoritmul trebuie apelat pentru fiecare componentă conexă în parte.

Dacă în cazul parcurgerii în lățime cea mai comodă implementare consta în utilizarea unei cozi, pentru parcurgerea în adâncime se va utiliza o stivă. Inițial stiva va conține doar nodul de pornire. La fiecare pas, este eliminat nodul din capul acesteia şi sunt adăugați vecinii nevizitați ai acestuia. Algoritmul se opreşte în momentul în care stiva este vidă.

Acest algoritm poate şi el fi folosit atât pentru grafuri orientate, cât şi pentru grafuri neorientate.

## 3.3.3. Lista părinților

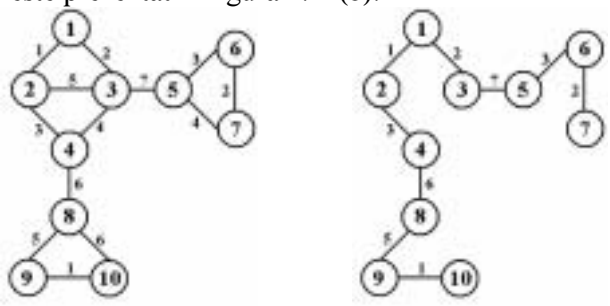
Pentru fiecare tip de parcurgere ***părintele*** unui nod este definit ca fiind vecinul din care s-a ajuns la nodul respectiv. Evident, nodul inițial de pornire nu vor avea nici un părinte, dar toate celelalte noduri din aceeaşi componentă conexă vor avea unul.

Pe parcursul traversării grafului pentru fiecare nod poate fi păstrat părintele său, obținându-se astfel o listă a părinților care va avea $N$ elemente. De obicei, pentru pă-

rintele nodului inițial de pornire se foloseşte o valoare specială; sunt utilizate mai ales valorile -1 şi 0.

# 3.4. Arbori parțiali minimi

Un ***arbore parțial minim*** al unui graf neorientat este *arborele parțial* care are cea mai mică sumă a costurilor muchiilor. De exemplu, pentru graful din figura 3.11(a), arborele parțial minim este prezentat în figura 3.11(b).



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

**Figura 3.11: (a)** Un graf neorientat cu costuri;
**(b)** Arborele parțial minim corespunzător

Există mai mulți algoritmi de determinare a arborelui parțial minim al unui graf conex; dintre aceştia cei mai cunoscuți şi mai des utilizați sunt algoritmul lui ***Kruskal*** şi algoritmul lui ***Prim***. În cadrul acestei secțiuni vom prezenta pe scurt aceşti doi algoritmi.

## 3.4.1. Algoritmul lui Kruskal

Pentru a determina un arbore parțial minim, *Kruskal* propune un algoritm în care inițial, se consideră că avem $N$ (numărul de noduri ale grafului) arbori, fiecare format dintr-un singur nod.

La fiecare pas se alege muchia de cost minim care uneşte două noduri aflate în doi arbori diferiți. Aşadar, doi dintre arbori sunt "uniți", deci numărul acestora descreşte cu 1 la fiecare pas. După $N$ - 1 paşi vom rămâne cu un singur arbore care este chiar arborele parțial de cost minim.

În figura 3.12 sunt prezentați cei nouă paşi ai algoritmului pentru graful din figura 3.11(a). În final, după cum se poate vedea în figura 3.12(j), se obține arborele din figura 3.11(b).

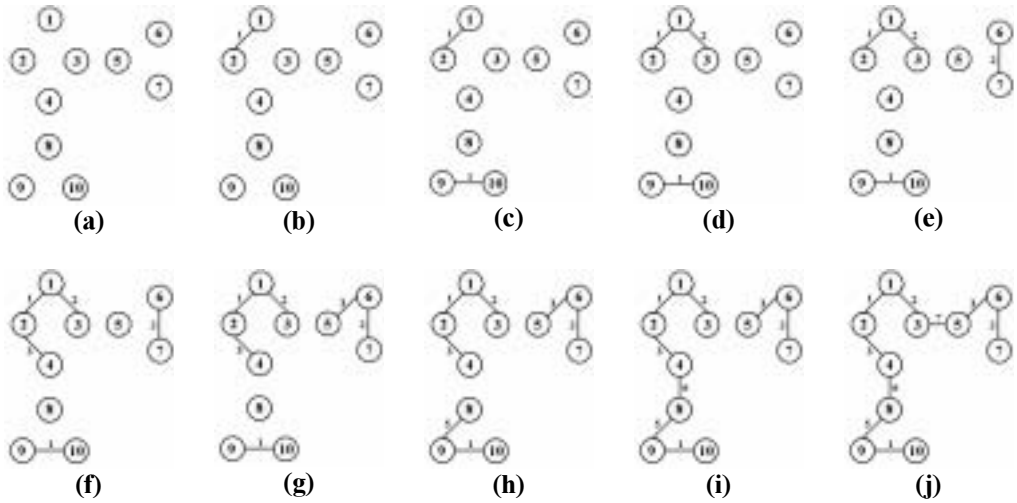



**Figura 3.12:** Paşii algoritmului lui *Kruskal*

Acest algoritm este foarte asemănător celui de determinare a claselor de echivalenţă. Existenţa unei muchii arată faptul că cele două extremităţi ale sale fac parte din acelaşi arbore (aceeaşi clasă de echivalenţă). În final vom rămâne cu o singură clasă de echivalenţă deoarece graful este conex. Algoritmul poate fi adaptat şi pentru cazul în care graful nu este conex. În acest caz numărul paşilor va fi mai mic (depinde de numărul componentelor conexe; pentru $k$ componente conexe vom avea $N$ - $k$ paşi) şi în final se va obţine o "pădure" de arbori.

## 3.4.2. Algoritmul lui Prim

Algoritmul propus de Prim este foarte asemănător cu cel propus de Kruskal. Singura diferenţă constă în faptul că la fiecare pas $i$ vom avea un arbore format din $i$ noduri şi $N - i$ arbori formaţi dintr-un singur nod.

De data aceasta, la fiecare pas vom alege muchia de cost minim care leagă un nod din arborele "mai mare" cu unul dintre nodurile care nu sunt incluse în arbore în acel moment.

Practic vom porni cu un arbore format dintr-un singur nod şi vom adăuga noduri până în momentul în care toate nodurile grafului vor face parte din acest arbore. După $N$ - 1 paşi toate cele N noduri vor face parte din acest arbore care va fi arborele parţial de cost minim al grafului.

În figura 3.13 sunt prezentaţi cei nouă paşi ai algoritmului pentru graful din figura 3.11(a). În final, după cum se poate vedea în figura 3.13(j), se obţine tot arborele din figura 3.11(b). De asemenea, se observă că pentru paşii intermediari configuraţia arborilor este diferită faţă de cea de la algoritmul lui *Kruskal*.

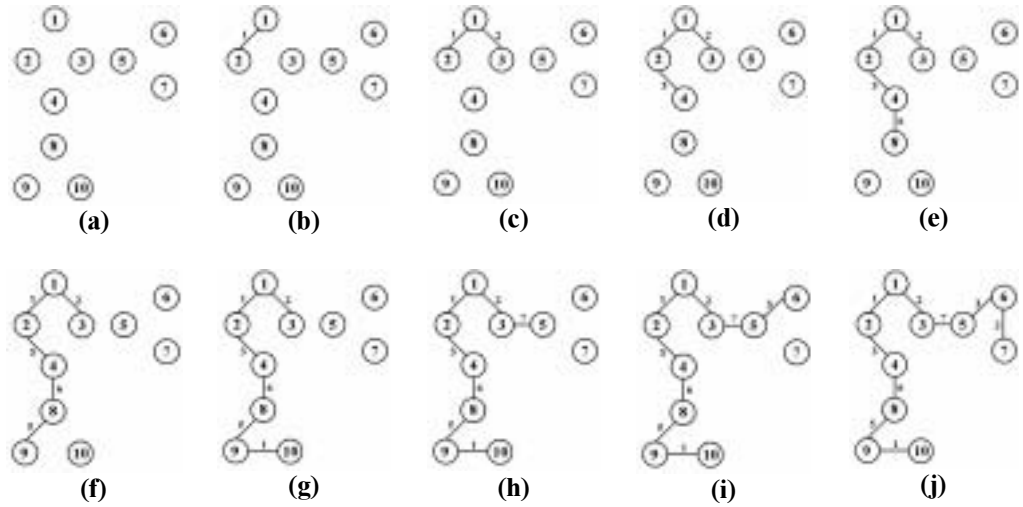



**Figura 3.13:** Paşii algoritmului lui *Prim*

Datorită faptului că la fiecare pas se adaugă încă un nod unui subarbore, algoritmului lui Prim nu poate fi adaptat foarte uşor pentru a funcționa şi în cazul grafurilor neconexe. În momentul în care este determina arborele parțial minim al primei componente conexe nu va mai fi găsit nici un nod care să fie legat printr-un muchie de unul dintre nodurile arborelui.

Aşadar, pentru a determina arborii parțiali minimi ai celorlalte componente conexe va trebui să aplicăm acest algoritm pornind de la un nod care face parte dintr-o altă componentă conexă.

# 3.5. Drumuri minime

În cadrul acestei secțiuni vom prezenta pe scurt detalii referitoare la modul în care pot fi determinate drumurile minime în grafuri. În principiu, un drum (lanț) este o succesiune de muchii între două noduri. Primul dintre acestea este nodul ***sursă***, iar celălalt este nodul ***destinație***.

Există patru categorii de algoritmi care pot fi utilizați pentru a determina astfel de drumuri:

- algoritmi pentru determinarea drumului minim de la o sursă dată la o destinație dată (sursă unică, destinație unică);
- algoritmi pentru determinarea drumului minim de la o sursă dată la toate celelalte noduri ale grafului (sursă unică, destinații multiple);
- algoritmi pentru determinarea drumului minim spre o destinație dată de la toate celelalte noduri (surse multiple, destinație unică);

- algoritmi pentru determinarea tuturor drumurilor minime între toate perechile de noduri (surse multiple, destinații unice).

Practic, algoritmii pentru primele trei categorii sunt aceeaşi deoarece dacă rezolvăm problema drumurilor de la un nod sursă la toate celelalte noduri atunci rezolvăm şi problema drumurilor de la un nod sursă la un nod destinație dat (deşi poate părea bizar, algoritmii pentru determinarea drumului către o singură destinație nu sunt asimptotic mai rapizi decât cei pentru determinarea drumurilor către toate destinațiile posibile). În plus, dacă rezolvăm problema drumurilor minime pentru o sursă unică şi destinații multiple, atunci am rezolvat şi problema drumurilor minim pentru surse multiple şi o destinație unică transformând nodul destinație în sursă şi aplicând acelaşi algoritm. Eventual, în cazul în care graful este orientat, va trebui să inversăm sensurile arcelor (vezi şi secțiunea 4.??).

Se observă faptul că este folosită noțiunea de *drum* şi nu cea de *lanț*. Aceasta se datorează faptului că aceşti algoritmi sunt concepuți pentru a opera asupra grafurilor orientate. Totuşi, ei pot opera şi asupra grafurilor neorientate în cazul în care considerăm că o muchie între două noduri $x$ şi $y$ reprezintă un arc de la nodul $x$ la nodul $y$ şi un arc de la nodul $y$ la nodul $x$.

## 3.5.1. Drumuri minime de la o sursă unică

Vom prezenta în cadrul acestei secțiuni doi dintre cei mai cunoscuți algoritmi care pot fi utilizați pentru determinarea drumurilor de cost minim de la o sursă la toate celelalte noduri ale grafului. Primul dintre ei, algoritmul lui ***Dijkstra***, poate fi utilizat doar dacă în graf nu există arce cu costuri negative. Cel de-al doilea, algoritmul ***Bellman-Ford***, poate fi utilizat chiar dacă avem arce cu costuri negative.

Algoritmul lui *Dijkstra* utilizează metoda *Greedy* pentru a determina costurile drumurilor spre toate nodurile grafului. Se porneşte de la nodul sursă şi se consideră că drumul până la acest nod are costul 0. În continuare, la fiecare pas se alege un arc care pleacă de la unul dintre nodurile pentru care s-a determinat deja distanța şi care ajunge la un nod pentru care nu a fost determinată încă distanța şi suma dintre costul arcului şi distanța până la nodul ales este minimă. Distanța până la noul nod va fi dată de această sumă. Pentru fiecare nod se va păstra distanța precum şi nodul anterior, în vederea reconstituirii drumului.

În figura 3.14 sunt prezentați cei cinci paşi ai algoritmului lui *Dijkstra* pentru un graf cu şase noduri. Pentru fiecare pas este evidențiată muchia aleasă. Primul număr din dreptul unui nod reprezintă distanța până la nodul respectiv, iar cel de-al doilea reprezintă nodul anterior.

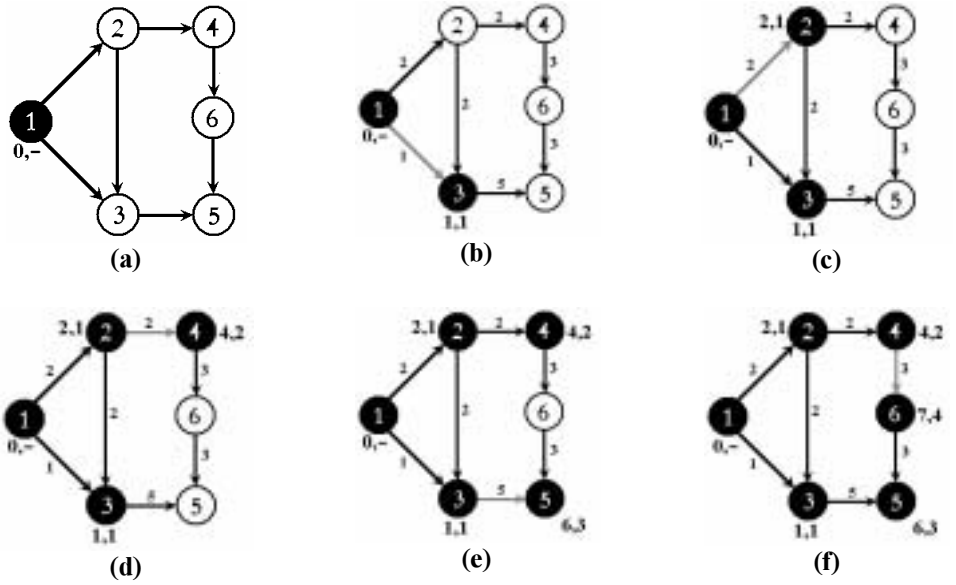



**Figura 3.14:** Paşii algoritmului lui *Dijkstra*

Din figura 3.14(f) rezultă că în final se obțin următoarele drumuri:

- 1 – 2 (costul este 2);
- 1 – 3 (costul este 1);
- 1 – 2 – 4 (costul este 4);
- 1 – 3 – 5 (costul este 6);
- 1 – 2 – 4 – 6 (costul este 7).

Aceasta nu este ordinea în care au fost obținute drumurile (ele sunt ordonate în funcție de numărul de ordine al destinației). De fapt, se observă că drumurile sunt obținute în ordinea crescătoare a costurilor lor.

După cum am afirmat anterior, algoritmul nu va duce la obținerea rezultatelor dorite în cazul în care există arce ale căror costuri sunt negative. Vom prezenta în continuare un algoritm care poate fi utilizat în cazul în care există arce cu ponderi negative, dar nu există circuite ale căror costuri sunt negative. Practic nici nu are sens să căutăm drumurile minime în cazul în care avem circuite cu ponderi negative deoarece aceste circuite ar putea fi parcurse la nesfârşit, costurile drumurilor devenind $-\infty$.

Pentru algoritmul Bellman-Ford, vom stabili din nou faptul că drumul până la nodul sursă are costul 0. În plus, vom stabili faptul că toate celelalte drumuri au costul $\infty$. Apoi la fiecare pas vom lua în considerare toate arcele. În cazul în care suma dintre costul unui drum la nodul de la care pleacă arcul şi costul arcului este mai mică decât

costul drumului la nodul la care ajunge arcul, vom actualiza costul acestui drum şi predecesorul nodului.

În figura 3.15 sunt prezentați cei cinci paşi ai algoritmului lui *Bellman-Ford* pentru un graf cu şase noduri care conține şi arce cu ponderi negative. Pentru fiecare pas sunt evidențiate muchiile care au dus la scăderea anumitor costuri. Primul număr din dreptul unui nod reprezintă distanța până la nodul respectiv, iar cel de-al doilea reprezintă nodul anterior.

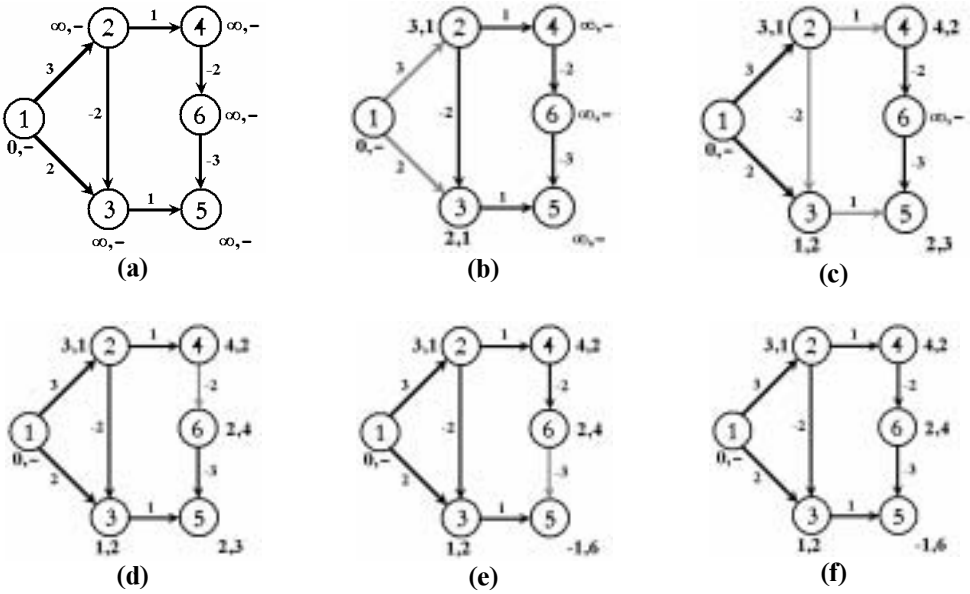



**Figura 3.15:** Paşii algoritmului *Bellman-Ford*

Se observă că la ultimul pas nu a existat nici o "îmbunătățire". Există posibilitatea să nu mai apară îmbunătățiri chiar înaintea acestui pas. Practic, algoritmul poate fi oprit în momentul în care se detectează faptul că un pas nu a adus din o îmbunătățire.

Din figura 3.15(f) rezultă că, în final, se obțin următoarele drumuri:

- 1 – 2 (costul este 3);
- 1 – 2 – 3 (costul este 1);
- 1 – 2 – 4 (costul este 4);
- 1 – 2 – 4 – 6 – 5 (costul este -1);
- 1 – 2 – 4 – 6 (costul este 2).

### 3.5.2. Drumuri minime între toate perechile de vârfuri

Există posibilitatea de a determina drumurile minime pentru toate perechile de vârfuri folosind algoritmii prezentați anterior. Vom considera, pe rând, fiecare nod ca fiind sursă şi vom determina astfel drumurile de la fiecare astfel de sursă la toate celelalte noduri. În final vom obține drumurile minime între toate perechile de vârfuri.

Cu toate acestea, există un algoritm uşor de implementat care poate fi folosit în acest scop. El poartă denumirea de algoritmul ***Floyd-Warshall***, dar datorită simplității sale mai este cunoscut şi sub numele de "algoritmul cu trei for-uri".

Pentru a nu complica prea mult prezentarea vom reaminti doar versiunea în pseudocod a acestui algoritm. Considerăm că $A$ este matricea costurilor arcelor; în cazul în care nu există un arc de la un nod $i$ la un nod $j$, valoarea corespunzătoare va fi 0. Algoritmul este următorul:

```
pentru k ← 1, n
  pentru i ← 1, n
    pentru j ← 1, n
      dacă aik + akj < aij atunci
        aij ← aik + akj
      sfârşit dacă
    sfârşit pentru
  sfârşit pentru
sfârşit pentru
```

Vă reamintim faptul că "ordinea" celor trei for-uri este ($k$, $i$, $j$); dacă aceasta nu este respectată rezultatele nu vor fi cele aşteptate.

## 3.6. Rezumat

În cadrul acestui capitol am prezentat noțiunile de bază referitoare la teoria grafurilor. Am prezentat definițiile noțiunilor mai importante introduse în clasa a X-a, precum şi diferite modalități de reprezentare a grafurilor.

De asemenea au fost descrişi pe scurt mai mulți algoritmi de bază care sunt vor fi folosiți în cadrul capitolelor care urmează. Pentru parcurgerea unui graf au fost descrişi algoritmii de parcurgere în lățime (*BF*) şi adâncime (*DF*). Pentru determinarea arborelui parțial minim al unui graf am prezentat algoritmul lui *Kruskal* şi algoritmul lui *Prim*.

Au fost prezentate şi câteva aspecte referitoare la determinarea drumurilor minime în grafurile orientate. Pentru drumurile de sursă unică am prezentat algoritmul lui *Dijkstra* şi algoritmul *Bellman-Ford*, iar pentru drumurile între toate perechile de vârfuri am prezentat algoritmul *Floyd-Warshall*.

## 3.7. Implementări sugerate

Pentru a vă familiariza cu modul în care trebuie implementate rezolvările problemelor în cadrul cărora trebuie folosite noțiuni de teoria grafurilor vă sugerăm să încercați să implementați algoritmi pentru:

- crearea listei de muchii pe baza matricei de adiacență;
- crearea listelor de vecini pe baza matricei de adiacență;
- crearea şirului succesorilor pe baza matricei de adiacență;
- crearea matricei de adiacență pe baza listei de muchii;
- crearea listelor de vecini pe baza listei de muchii;
- crearea şirului succesorilor pe baza listei de muchii;
- crearea matricei de adiacență pe baza listelor de vecini;
- crearea listei de muchii pe baza listelor de vecini;
- crearea şirului succesorilor pe baza listelor de vecini;
- crearea matricei de adiacență pe baza şirului succesorilor;
- crearea listei muchiilor pe baza şirului succesorilor;
- crearea listelor de vecini pe baza şirului succesorilor;
- parcurgerea în lățime a unui graf neorientat;
- parcurgerea în adâncime a unui graf neorientat;
- parcurgerea în lățime a unui graf orientat;
- parcurgerea în adâncime a unui graf orientat;
- verificarea conexității unui graf neorientat;
- determinarea componentelor conexe ale unui graf neorientat;
- determinarea unui arbore parțial minim într-un graf neorientat;
- determinarea unui drum de cost minim între două noduri ale unui graf neorientat;
- determinarea drumurilor de cost minim de la o sursă dată la toate celelalte noduri într-un graf neorientat;
- determinarea drumurilor de cost minim între toate perechile de noduri ale unui graf neorientat;
- determinarea unui drum de cost minim între două noduri ale unui graf orientat;
- determinarea drumurilor de cost minim de la o sursă dată la toate celelalte noduri într-un graf orientat;
- determinarea drumurilor de cost minim între toate perechile de noduri ale unui graf orientat.

## 3.8. Probleme propuse

În continuare vom prezenta enunțurile câtorva probleme pe care vi le propunem spre rezolvare. Acestea necesită anumite cunoştințe prezentate în clasa a X-a, dar care au fost amintite pe scurt şi în cadrul acestui capitol.

## 3.8.1. Rețea

**Descrierea problemei**
Se consideră o rețea de calculatoare care conține $N$ noduri (calculatoare) identificate prin numere cuprinse între 1 şi $N$. Nodul identificat prin 1 este server-ul. Un mesaj trebuie transmis de la server la toate celelalte noduri ale rețelei. La momentul 0 mesajul pleacă de la server spre toate nodurile cu care acesta este legat direct. Dacă un calculator primeşte mesajul la un moment t, atunci la momentul $t + 1$ el transmite mesajul respectiv spre toate calculatoarele cu care acesta este legat direct şi care nu au primit mesajul la un moment anterior. Să se determine momentele de timp la care ajunge mesajul la fiecare dintre nodurile rețelei.

**Date de intrare**
Prima linie a fişierului de intrare **`RETEA.IN`** conține numărul $N$ al nodurilor rețelei şi numărul $M$ al legăturilor directe dintre calculatoare. 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 directă între calculatoarele identificate prin $x$ şi $y$.

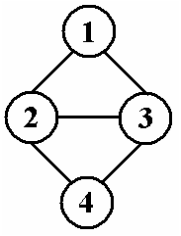
**Date de ieşire**
Fişierul de ieşire **`RETEA.OUT`** va conține o singură linie pe care se vor afla $N - 1$ numere. Al $i$-lea număr de pe această linie reprezintă momentul de timp la care ajunge mesajul la calculatorul identificat prin numărul $i + 1$. Aceste numere vor fi separate prin spații.

**Restricții şi precizări**
- $3 \leq N \leq 100$;
- $1 \leq M \leq 1000$;
- există cel mult o legătură directă între oricare două calculatoare;
- mesajul va putea ajunge întotdeauna la toate nodurile rețelei;
- mesajul poate fi transmis folosind o legătură directă între două calculatoare în ambele sensuri.

**Exemplu**

**RETEA.IN**

```
4 5
1 2
1 3
2 3
2 4
3 4
```




**RETEA.OUT**

```
1 1 2
```

**Timp de execuție: 1 secundă/test**

## 3.8.2. Galia

**Descrierea problemei**

După moartea eroilor mitici *Asterix* şi *Obelix*, *Imperiul Roman* a reuşit, în sfârşit, să cucerească întreaga *Galie*. După terminarea luptelor, a fost creată o structură administrativă care să conducă noua provincie romană. Cea mai mare problemă era comunicația între diferitele sate care era greoaie deoarece galezii nu aveau nevoie de drumuri (puteau ajunge foarte repede oriunde cu ajutorul poțiunii magice). Din aceste motive, Imperiul a decis construirea unor drumuri astfel încât să se poate ajunge dintr-un sat în oricare altul.

Ca urmare, pentru fiecare pereche de sate se ştie dacă poate fi construit un drum între satele respective şi, în caz afirmativ, care este costul necesar construirii drumului respectiv. Din nefericire, războiul cu *Asterix*, *Obelix* şi cățelul *Idefix* a secătuit vistieria romană, motiv pentru care costul total al construirii drumurilor trebuie să fie minim.

Pentru a nu avea surprize (moartea druidului *Panoramix* nu a fost încă dovedită) Împăratul a cerut şi un plan de rezervă. Pentru acest plan costul total al lucrării trebuie să fie cât mai apropiat de cel al lucrării corespunzătoare planului principal (eventual costurile pot fi egale).

**Date de intrare**

Prima linie a fişierului de intrare **GALIA.IN** conține numărul $N$ al satelor din *Galia* şi numărul $M$ al drumurilor care pot fi construite. Aceste numere vor fi separate printr-un spațiu.

Fiecare dintre următoarele $M$ linii va conține câte trei numere întregi $x$, $y$ şi $c$ cu semnificația: poate fi construit un drum între satele $x$ şi $y$, iar costul acestui drum este $c$.

**Date de ieşire**

Prima linie a fişierului de ieşire **GALIA.OUT** va conține numărul *p* al drumurilor care trebuie construite potrivit planului principal. Fiecare dintre următoarele *p* linii va conține câte două numere *x* şi *y* cu semnificația: potrivit planului principal trebuie construit un drum între satele *x* şi *y*.

Următoarea linie a fişierului va conține numărul *q* al drumurilor care trebuie construite potrivit planului de rezervă. Fiecare dintre următoarele *q* linii va conține câte două numere *x* şi *y* cu semnificația: potrivit planului de rezervă trebuie construit un drum între satele *x* şi *y*.

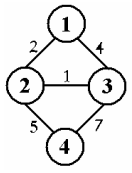
**Restricții şi precizări**

- $3 \le N \le 100$;
- $N \le M \le 1000$;
- costul unui drum este un număr întreg cuprins între 1 şi 10000;
- satele sunt identificate prin numere cuprinse între 1 şi *N*;
- poate fi construit cel mult un drum între oricare două sate;
- va exista întotdeauna cel puțin o soluție;
- pe un drum se poate circula în ambele sensuri;
- configurația de drumuri corespunzătoare planului principal trebuie să difere de cea corespunzătoare planului de rezervă;
- dacă există două sau mai multe planuri de rezervă, atunci poate fi ales oricare dintre ele;
- dacă există cel puțin două planuri pentru care costul este minim, atunci oricare dintre ele poate fi plan principal şi oricare dintre ele poate fi plan de rezervă.

**Exemplu**

**GALIA.IN**

4 5
1 2 2
1 3 4
2 3 1
2 4 5
3 4 7




**GALIA.OUT**

3
1 2
2 3
2 4

3
1 3
2 3
2 4

**Explicație**
Configurația corespunzătoare planului principal are costul total 8. Planul de rezervă are costul total 10. Există încă un plan care are costul total tot 10 şi care putea fi şi el ales ca fiind plan de rezervă. Acesta constă în alegerea perechilor de sate 1 şi 2, 2 şi 3, 3 şi 4.

**Timp de execuție: 1 secundă/test**

## 3.8.3. Săgeți

**Descrierea problemei**
*Mickey* şi *Minnie* au ajuns în fața unui perete pe care erau desenate mai multe cercuri şi mai multe săgeți, fiecare săgeată pornind dintr-un cerc şi ajungând la un alt cerc. Desenul era foarte complicat, dar *Minnie* l-a întrebat pe *Mickey* dacă există vreun cerc în care intră săgeți de care pornesc de la toate celelalte cercuri, dar din care nu pleacă nici o săgeată. Pentru a o impresiona pe *Minnie*, *Mickey* a spus că va studia problema, dar se pare că nu se prea descurcă, motiv pentru care vă cere ajutorul. Din nefericire, nu puteți comunica decât prin intermediul unui telefon. *Mickey* vă spune la început numărul $N$ al cercurilor, dar nu are răbdare să vă descrie toate săgețile. Datorită nerăbdării lui *Mickey*, îl veți putea întreba doar de $2 \cdot N$ ori dacă există o săgeată care pleacă de la un anumit cerc şi ajunge la un anumit cerc. Din fericire, ați auzit-o la telefon pe *Minnie* strigând "*L-am găsit*!", motiv pentru care ştiți cu siguranță că există un astfel de cerc.

Pentru a simula discuția cu *Mickey* aveți la dispoziție o bibliotecă externă numită **SAGETI.PAS** (pentru programatorii în Pascal) sau **SAGETI.H** (pentru programatorii în C/C++). Rutinele pe care le aveți la dispoziție sunt prezentate în continuare:

```
procedure Init;
void Init(void);
```

- este folosită pentru inițializarea bibliotecii;
- trebuie apelată înaintea apelării oricărei alte rutine a bibliotecii;
- întrerupe execuția programului dacă este apelată a doua oară.

```
function GetN:Integer;
int GetN(void);
```

- returnează numărul de cercuri de pe perete.

```
function ExistaSageata(x,y:Integer):Boolean;
int ExistaSageata(int x, int y);
```

- returnează true (valoare nenulă) dacă există o săgeată de la cercul *x* la cercul *y* şi fals în caz contrar;
- întrerupe execuția programului dacă este apelată a $2 \cdot N + 1$ – a oară.

```
procedure Rezultat(cerc:Integer);
void Rezultat(int cerc);
```

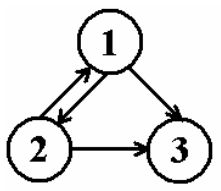
- acceptă ca rezultat faptul că `cerc` este cercul căutat;
- validează rezultatul;
- întrerupe execuția programului.

**Restricții şi precizări**

- programul vostru nu va citi date din nici un fişier de intrare şi nu va scrie data în nici un fişier de ieşire; toate operațiile de intrare/ieşire vor fi realizate prin intermediul bibliotecii;
- $1 \leq N \leq 100$;
- cercurile sunt identificate prin numere cuprinse între 1 şi *N*;
- există cel mult o săgeată de la un cerc *x* la un cerc *y*;
- nu există săgeți care pleacă de la un cerc *x* şi ajung la acelaşi cerc *x*;
- dacă există o săgeată de la un cerc *x* la un cerc *y*, atunci este posibil să existe şi o săgeată de la cercul *y* la cercul *x*.

**Exemplu de utilizare a bibliotecii**

| *Rutină apelată* | *Valoare returnată* |
|---|---|
| `Init` | – |
| `GetN` | `3` |
| `ExistaSageata(1,2)` | `true (1)` |
| `ExistaSageata(1,3)` | `true (1)` |
| `ExistaSageata(2,3)` | `true (1)` |
| `ExistaSageata(3,1)` | `false (0)` |
| `ExistaSageata(2,1)` | `true (1)` |
| `ExistaSageata(3,2)` | `false (0)` |
| `Rezultat(3)` | – |




**Timp de execuție: 1 secundă/test**

# 3.9. 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.

## 3.9.1. Rețea

Rețeaua de calculatoare poate fi privită ca fiind un graf neorientat în care nodurile reprezintă calculatoarele rețelei, iar muchiile reprezintă legăturile directe între acestea.

În aceste condiții problema se reduce la determinarea, pentru fiecare nod, a distanței față de nodul identificat prin 1 (cel care reprezintă serverul). Distanța dintre două noduri este dată de numărul muchiilor din care este format cel mai scurt drum dintre cele două noduri.

Pentru a determina aceste distanțe vom folosi algoritmul de parcurgere în lățime (***B**readth **F**irst - **BF***) a unui graf. La fiecare pas vom determina, pentru toate nodurile aflate pe nivelul curent, nodurile de pe nivelul următor. Nodurile de pe nivelul următor sunt acele noduri care au legături directe cu cel puțin un nod de pe nivelul curent şi nu se află pe un nivel anterior. Este evident faptul că toate nodurile de pe un anumit nivel se vor afla la aceeaşi distanță față de nodul identificat prin 1. Aşadar, distanța față de acest nod va fi dată de nivelul pe care se află nodul respectiv în urma parcurgerii în lățime.

După determinarea acestor distanțe vom scrie rezultatele în fişierul de ieşire.

**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 parcurgere în lățime a unui graf are ordinul de complexitate **O(M)**, deci acesta este ordinul de complexitate al operației de determinare a distanțelor la care se află nodurile față de nodul identificat prin 1.

Scrierea datelor în fişierul de ieşire implică scrierea a $N - 1$ valori, aşadar ordinul de complexitate al acestei operații este **O(N)**.

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

## 3.9.2. Galia

Satele din *Galia* pot fi privite ca fiind nodurile unui graf ale cărui muchii sunt date de drumurile care pot fi construite. Costurile muchiilor reprezintă costurile necesare construirii unui drum între două sate.

În aceste condiții problema se reduce la determinarea unui arbore parțial de cost minim şi a celui de-al doilea arbore parțial de cost minim într-un graf neorientat. Prin al doilea arbore parțial minim înțelegem arborele cu cel mai mic cost care este diferit de arborele parțial de cost minim.

Pentru a determina arborele parțial de cost minim putem folosi unul dintre algoritmii clasici: *Kruskal* sau *Prim*. Datorită faptului că graful este specificat prin lista muchiilor, am ales folosirea algoritmului lui *Kruskal*.

Pentru a utiliza algoritmul lui *Kruskal* muchiile trebuie sortate în funcție de costul lor. Datorită faptului că sunt cel mult 1000 de muchii, nu trebuie neapărat folosit un algoritm performant de sortare, fiind suficient unul care funcționează în timp pătratic.

După construirea listei ordonate a muchiilor, arborele parțial de cost minim este determinat folosind algoritmul amintit. Vom crea un vector de indici în care un element va indica numărul de ordine (în lista sortată) al unei muchii care face parte din arborele parțial de cost minim.

Vom demonstra în continuare că al doilea arbore parțial de cost minim diferă printr-o singură muchie de arborele parțial de cost minim. În acest scop vom utiliza metoda reducerii la absurd. Vom presupune că acest arbore diferă prin două sau mai multe muchii. Aşadar, au fost eliminate cel puțin două muchii care au fost înlocuite cu altele. Evident, costurile muchiilor adăugate sunt mai mari sau egale cu costurile muchiilor eliminate deoarece, în caz contrar ele ar fi făcut parte din arborele parțial de cost minim.

Reintroducând în arbore una dintre muchiile eliminate şi eliminând-o pe cea cu care aceasta a fost înlocuită vom obține un arbore parțial minim care are un cost mai mic sau egal. În plus, acest arbore diferă de arborele parțial de cost minim datorită prezenței celeilalte muchii nou introduse. Ca urmare, am obținut un arbore care are un cost mai mic sau egal decât "al doilea arbore parțial de cost minim" şi un cost mai mare sau egal decât cel al arborelui parțial de cost minim. În concluzie, ipoteza este falsă, deci al doilea arbore parțial de cost minim diferă printr-o singură muchie de arborele parțial de cost minim.

Pentru a determina al doilea arbore parțial de cost minim vom considera că oricare dintre muchiile care nu fac parte din arborele parțial de cost minim este o *muchie candidată* pentru inserare.

Prin inserarea unui astfel de muchii se creează un ciclu; aşadar, una dintre muchiile din acest ciclu trebuie eliminată pentru a păstra proprietatea de arbore. Pentru ca arborele obținut să aibă un cost cât mai mic vom elimina acea muchie care are cea mai mică diferență de cost față de muchia introdusă.

Pentru a regăsi soluția vom păstra diferența minimă obținută la un moment dat, muchia adăugată în momentul în care a fost determinată această diferență minimă şi muchie eliminată în momentul respectiv.

Pentru a determina ciclul format prin adăugarea unei muchii vom crea o listă de părinți ai nodurilor din arborele parțial de cost minim. Cu excepția nodului 1 (pe care

îl vom considera ca fiind rădăcina arborelui) pentru celelalte noduri în listă se va păstra numărul de ordine al părinților nodurilor în arborele parțial de cost minim care are rădăcina în nodul 1.

Lista părinților poate fi determinată folosind următorul algoritm: atât timp cât nu au fost stabiliți părinții tuturor nodurilor se parcurge lista muchiilor arborelui parțial de cost minim; în momentul în care este identificată o muchie pentru care se cunoaşte părintele uneia dintre extremități şi nu se cunoaşte părintele celeilalte extremități se poate afirma că părintele acestei a doua extremități este prima extremitate.

Cunoscând lista părinților, se poate determina un ciclu format prin adăugarea unei muchii dacă se parcurg traseele de la extremitățile muchiilor spre rădăcină. După determinarea acestor două trasee se elimină porțiunile comune (acestea nu fac parte din ciclu). Muchiile de pe cele două trasee (după eliminarea porțiunii comune) sunt muchiile care, împreună cu muchia adăugată, formează un ciclu. În acest moment se poate verifica dacă prin eliminarea uneia dintre muchiile de pe trasee şi adăugarea muchiei considerate se obține o diferență mai mică decât minimul din acel moment. Pentru a obține mai rapid costul unei muchii va trebui să avem la dispoziție o matrice a costurilor. Aceasta poate fi creată pe măsura citirii datelor de intrare.

În final vom cunoaşte muchiile care fac parte din primul arbore parțial minim, muchia care este eliminată în vederea obținerii celui de-al doilea arbore de cost minim şi muchia care trebuie inserată în vederea obținerii acestui al doilea arbore de cost minim.

Este cunoscut faptul că numărul de muchii dintr-un arbore parțial al unui graf cu $N$ noduri este $N - 1$. Pentru a scrie în fişierul de ieşire muchiile arborelui parțial de cost minim va trebui să parcurgem doar lista muchiilor şi să scriem extremitățile corespunzătoare. Pentru a scrie în fişierul de ieşire muchiile celui de-al doilea arbore parțial de cost minim vom scrie mai întâi extremitățile muchiei adăugate. În continuare vom parcurge din nou lista muchiilor şi vom scrie extremitățile corespunzătoare numai dacă acestea nu sunt cele ale muchiei eliminate.

## 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 şi crearea matricei costurilor, ordinul de complexitate al acestei operații fiind tot **O(M)**.

Algoritmul de sortare a muchiilor grafului are ordinul de complexitate $\mathbf{O(M^2)}$ deoarece nu este necesară folosirea unui algoritm mai performant.

Algoritmul de determinare a muchiilor care fac parte din arborele parțial minim implică o parcurgere a listei sortate a muchiilor şi, la fiecare pas, actualizarea claselor de echivalență corespunzătoare nodurilor grafului. Ca urmare, ordinul de complexitate al acestei operații este $\mathbf{O(M \cdot N)}$.

Pentru determinarea listei părinților va trebui parcursă de mai multe ori lista muchiilor. Aşadar, fiecare parcurgere are ordinul de complexitate $O(M)$. Datorită faptului că

avem $N$ noduri şi la fiecare parcurgere se va determina cel puţin părintele unui nod, ordinul de complexitate al operaţiei de determinare a listei părinţilor este **O(M · N)**.

Pentru a determina cel de-al doilea arbore parţial de cost minim va trebui să luăm în considerare toate cele $M - N + 1$ muchii care nu fac parte din arborele parţial de cost minim. Pentru fiecare astfel de muchie vom determina traseele până la rădăcină (nodul 1). Un astfel de traseu va conţine cel mult $N - 1$ noduri, deci ordinul de complexitate al operaţiei de determinare a unui trasei este $O(N)$. Cele două trasee conţin cel mult $N - 1$ noduri, deci operaţia de eliminare a porţiunii comune are ordinul de complexitate tot $O(N)$. Considerarea muchiilor care fac parte din ciclu are acelaşi ordin de complexitate $\mathbf{O}(N)$ deoarece un ciclu este format din cel mult $N$ muchii. În concluzie, operaţia de determinare a celui de-al doilea arbore parţial de cost minim are ordinul de complexitate $O(M - N + 1) \cdot (\mathbf{O}(N) + \mathbf{O}(N) + \mathbf{O}(N)) = O(M - N) \cdot O(N) = \mathbf{O(M \cdot N - N^2)}$.

Scrierea datelor în fişierul de ieşire implică două traversări a listei muchiilor arborelui parţial de cost minim, deci are ordinul de complexitate $O(N)$.

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

## 3.9.3. Săgeţi

Cercurile de pe perete pot fi privite ca fiind nodurile unui graf orientat ale cărui arce sunt date de săgeţile dintre cercuri; sensurile arcelor sunt date de sensurile săgeţilor.

În aceste condiţii problema se reduce la determinarea unui nod al grafului care are gradul interior $N - 1$ (la el ajung arce de la toate celelalte noduri) şi gradul exterior 0 (nu există nici un arc care pleacă de la nodul respectiv).

Datorită limitei de $2 \cdot N$ interogări (apeluri ale funcţiei `ExistaSageata()`) pe care le avem la dispoziţie va trebui să găsim un algoritm eficient de determinare a unui astfel de nod.

Iniţial vom presupune că nodul căutat este identificat prin 1. Vom verifica respectarea condiţiilor parcurgând celelalte noduri. Atât timp cât nu există o săgeată de la nodul 1 la nodul curent şi există o săgeată de la nodul curent la nodul 1, vom continua parcurgerea.

Să presupunem că identificăm un nod $i$ pentru care condiţia nu mai este satisfăcută. În această situaţie putem presupune că nici unul dintre nodurile cuprinse între 2 şi $i - 1$ nu reprezintă soluţia deoarece de la ele pleacă arce spre nodul 1. Evident nici nodul 1 nu este soluţia căutată deoarece fie există o săgeată de la el la nodul $i$, fie nu există o săgeată de la nodul $i$ la el. Ca urmare, problema se reduce la determinarea unei soluţii pentru subgraful format din nodurile cu numere de ordine cuprinse între $i$ şi $N$.

Vom aplica acest algoritm până în momentul în care ajungem să luăm în considerare şi nodul $N$. În acest moment soluţia va fi dată fie de "candidatul" curent, în cazul în care se respectă condiţiile pentru acel nod şi nodul $N$, fie de nodul $N$, în caz contrar.

În final, soluția obținută este furnizată ca rezultat.

Trebuie observat faptul că enunțul ne asigură că soluția există şi este unică. Deoarece, parcurgem $N - 1$ noduri şi la fiecare pas efectuăm una sau două interogări (dacă rezultatul primei interogări implică nerespectarea condiției, atunci nu are sens să realizăm şi a doua interogare), numărul total al interogărilor va fi de cel mult $2 \cdot N - 2$, deci condiția impusă asupra numărului maxim de interogări este respectată.

**Analiza complexității**

Preluarea datelor şi furnizarea rezultatului se realizează în timp constant. Datorită faptului că realizăm cel mult $2 \cdot N - 2$ interogări şi fiecare astfel de interogare este realizată în timp constant, ordinul de complexitate al algoritmului este **O(N)**.

În această analiză nu am luat în considerare timpul necesar inițializării bibliotecii deoarece concurentul nu poate influența modul de creare a acesteia. În cazul de față inițializarea constă în citirea matricei de adiacență a grafului, deci ar avea ordinul de complexitate $O(N^2)$.

Cu toate acestea, rezolvarea propriu-zisă a problemei necesită un timp de execuție care variază liniar față de numărul cercurilor de pe perete (cel al nodurilor grafului).

**Discuție**

Datorită faptului că din enunț rezultă clar faptul că întotdeauna există un cerc cu proprietatea cerută, există şi posibilitatea de a-l determina folosind numai $N - 1$ interogări. Vă propunem să încercați să demonstrați acest lucru şi să implementați o soluție în care numărul maxim al interogărilor permise este $N$.

În cazul în care enunțul nu ar fi asigurat existența soluției am avea, într-adevăr, nevoie de până la $2 \cdot N - 2$ interogări. Vă propunem să încercați să rezolvați problema şi pentru acest caz. (În cazul în care nu există nici un cerc cu proprietatea cerută transmiteți ca rezultat valoarea 0.)