[section]

Introducere în Teoria Grafurilor

Daniel GROSU
Liceul Academiei de Științe
daniel.grosu@hotmail.com

Martie 2015

Cuprins

Capitolul 1

Introducere în teoria grafurilor

1.1 Introducere

Teoria grafurilor este o ramură a matematicii discrete. Matematica discretă—studiul structurilor discrete (a mulțimilor finite) și proprietățile acestora—cuprinzând combinatorica (studiul combinării și enumerării obiectelor), algoritmii de calcul a proprietăților mulțimilor de obiecte și *teoria grafurilor* (studiul obiectelor și a relațiilor acestora).

Multe probleme în matematica discretă se reduc la o problemă de grafuri. Deci, teoria grafurilor, este considerată una din cele mai importante și studiate ramuri ale matematicii discrete.

În această lucrare voi prezenta clasificarea grafurilor într-o cheie teoretică și defini un șir de algoritmi de căutare și de determinare a drumului de cost minim într-un graf ponderat.

1.2 De ce să studiem grafurile?

O ramură științifică își justifică importanța prin aplicativitatea sa, dar și prin frumusețea estetică a definiției structurilor abstracte. Teoria grafurilor se aplică în multe domenii, și dacă să numim puține dintre ele, în biologie, economie, inginerie, informatică, lingvistică, matematică, medicină și științe sociale. Vom observa că graful este o unealtă bună de modelare a problemelor. Să introducem câteva probleme clasice.

Figura 1.1: Podurile din Kőnigsberg

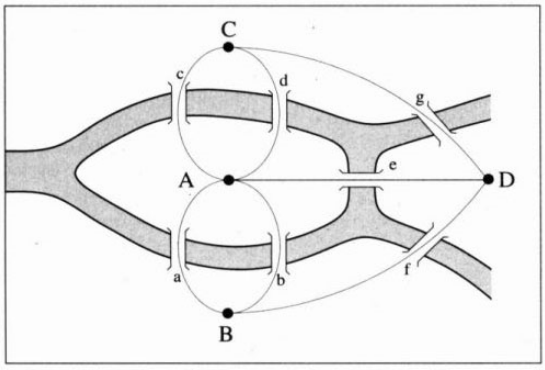


Figura 1.2: Graful podurilor din Kőnigsberg

Podurile din Kőnigsberg Figura 1.1 reprezintă centrul orașului Kőnigsberg. Râul Pregel are două maluri, de nord și de sud (C și B) iar ramificarea lui formează două insule (A și D).

Malurile și insulele sunt conectate prin intermediul a șapte poduri la fel ca în Figura 1.2. Se pune problema găsirii unei călătorii prin centrul Kőnigsberg-ului, traversând fiecare pod o singură dată.

Euler a simplificat această problemă, reducând-o la o diagramă (un graf) cum e reprezentat în Figura 1.2. Punctele negre (nodurile) sunt țărmurile iar liniile (muchiile) reprezintă podurile. Deci, începând din orice nod, trebuie să parcurgem toate muchiile o singură dată și să ajungem în nodul de unde am pornit. Euler a dat o explicație concisă când un astfel de drum există.

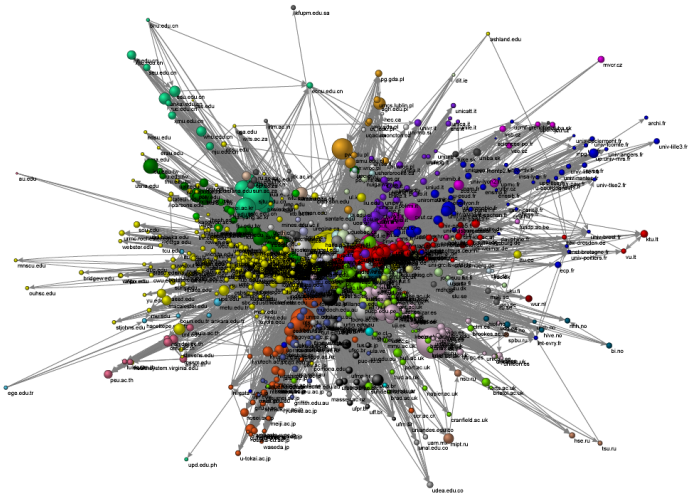


Figura 1.3: Un subgraf a WWW-ului

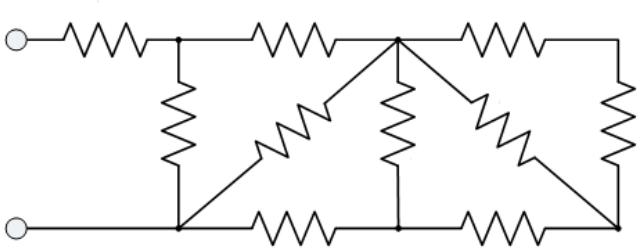


Figura 1.4: O rețea de rezistențe

World Wide Web WWW-ul poate fi modelat ca un graf în care paginile sunt puncte sau noduri iar hyperlink-urile între ele sunt reprezentate de linii sau muchii ce le unesc în graf. Îm Figura 1.3 este reprezentată un *subgraf* foarte mic a grafului WWW.

Rețele electrice În figura 1.4 fiecare muchie reprezintă o rezistență de $R = 1\Omega$. Cum putem să calculăm rezistența totală a acestui circuit? Poate părea ușor un astfel de circuit și putem să calculăm simplu rezistența totală cu legile legării în serie și paralel a conductoarelor, însă e complicat să facem operații pe scheme mari.

Până aici am enumerat câteva exemple de probleme din teoria grafurilor. Putem să modelăm o mulțime de noi probleme cu ajutorul grafurilor.

Pentru a modela probleme și mai interesante, și de a forma o teorie coerentă despre grafuri, este necesar de a formaliza noțiunile din aceste exemple în

definiții generale, dar și concrete în același timp.

1.3 Definiția grafului

Am analizat grafurile informal și am schițat o anumită idee despre ele. O idee intuitivă este bună însă nu mereu aplicabilă în practică. Informal, graful este o mulțime de noduri și muchii cu o legitate a conectării nodurilor prin muchii.

Următoarea definiție este o generalizare, iar în continuare vom prezenta particularități ale grafurilor cu diferite proprietăți.

Definiție 1.3.1 *Un graf sau un graf general este un triplet ordonat* $G = (V, E, \phi)$, *unde*

1. $V \neq \emptyset$,
2. $V \cap E = \emptyset$,
3. $\phi : E \to \boldsymbol{P}(V)$ *este o funcție de ordin superior astfel încât* $|\phi(e)| \in \{1, 2\}$ *pentru* $\forall e \in E$

Elementele lui V sunt *nodurile* lui G iar elementele lui E sunt *muchiile* lui G. Funcția ϕ este o funcție de incidență, iar nodurile din $\phi(e)$ se numesc *noduri de capăt* a muchiei e. Când tripletul unui graf nu este definit, prin mulțimea $V(G)$ se subînțelege mulțimea nodurilor lui G iar prin $E(G)$ — mulțimea muchiilor.

Majoritatea grafurilor reprezentate prin diagrame constau în puncte, ce reprezintă nodurile, și din curbe între puncte (vârfurile muchiilor), ce reprezintă muchiile. În cazul în care diagrama nu creează ambiguități, ea însăși poate fi numită un graf, căci toată informația poate fi citită direct din diagramă și poate fi construită o unică definiție abstractă a grafului.

Analizăm un exemplu pentru a ilustra definiția.

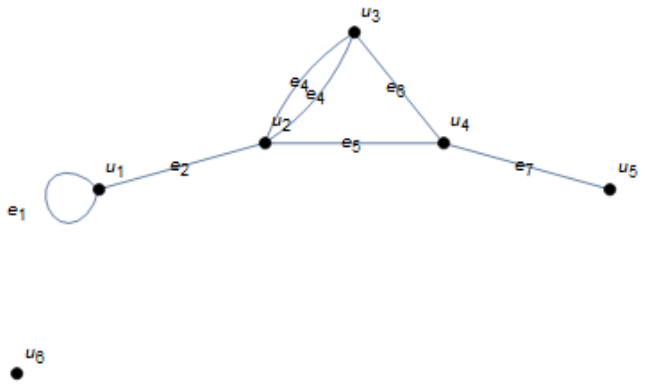


Figura 1.5: Un graf cu 6 noduri și 7 muchii

Considerăm graful din Figura 1.5. Observăm graful $G = (V, E, \phi)$, unde mulțimea nodurilor $V = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, mulțimea muchiilor $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ și funcția $\phi : E \to \mathbf{P}(V)$ ia valorile

$$\begin{aligned}
&\phi(e_1) = \{u_1\},\\
&\phi(e_2) = \{u_1, u_2\},\\
&\phi(e_3) = \phi(e_4) = \{u_2, u_3\},\\
&\phi(e_5) = \{u_2, u_4\},\\
&\phi(e_6) = \{u_3, u_4\},\\
&\phi(e_7) = \{u_4, u_5\},\\
&\nexists e \in E(V), u_6 \in \phi(e)
\end{aligned}$$

Următoarea definiție definește terminologia de bază a grafurilor.

Definiții 1.3.2 *Fie $G = (V, E, \phi)$ un graf, atunci*

1. *Nodurile u și v din $V(G)$ sunt **noduri vecine** sau **noduri adiacente** dacă sunt vârfurile unei muchii $e \in E$. Adică, sunt **adiacente** dacă există o muchie $e \in E$, astfel încât $\phi(e) = \{u, v\}$.*

2. *Două muchii distincte $e, f \in E(G)$ sunt **muchii adiacente** dacă au un nod-vârf comun. Adică, $\phi(e) \cap \phi(f) \neq \emptyset$.*

3. *Asemeni, un nod v și o muchie e sunt **incidente** dacă $u \in \phi(e)$—adică, u este un vârf a muchiei e.*

4. *O **buclă** este o muchie e, a cărei noduri de vârf sunt identice, $|\phi(e)| = 1$.*

5. *Un nod este numit un **nod izolat** dacă nu este incident cu nici o muchie, așadar, nu este vârful niciunei muchii. Deci, $u \notin \phi(e), \forall e \in E$.*

1.4 Grafuri simpli

Definiție 1.4.1 *Un graf simplu este perechea $G = (V, E, \phi)$, unde V este o mulțime nevidă de noduri, iar E este o mulțime de submulțimi de 2 elemente din V.*

$$E \subseteq \{X : X \subseteq V, |V| = 2\} = \{\{u, v\} : u, v \in V, u \neq v\}.$$

În următoarele exemple, definim câteva tipuri de grafuri simple.

Definiție 1.4.2 *Un **graf nul** de n noduri este un graf simplu $G = N_n$, unde*

$$V(N_n) = \{u_1, u_2, \ldots, u_n\},$$
$$E(N_n) = \emptyset.$$

Figura 1.6: Un graf nul N_n

Definiție 1.4.3 *Un **graf-drum** de $n \geq 2$ noduri este un graf simplu $G = P_n$, unde*

$$V(P_n) = \{u_1, u_2, \ldots, u_n\},$$
$$E(P_n) = \{\{u_1, u_2\}, \{u_2, u_3\}, \ldots, \{u_(n-1), u_n\}\}.$$

Din convenție, $P_1 = N_1$.

Figura 1.7: Un graf-drum P_n

Definiție 1.4.4 *Un **graf-ciclu** sau **ciclu** de $n \geq 3$ noduri afișat în Figura 1.8 este un graf simplu $G = C_n$, unde*

$$V(C_n) = \{u_1, u_2, \ldots, u_n\},$$
$$E(C_n) = \{\{u_1, u_2\}, \{u_2, u_3\}, \ldots, \{u_{n-1}, u_n\}, \{u_n, u_1\}\}.$$

Observăm că putem obține C_n conectând u_1 și u_n în P_n (legând capetele drumului).

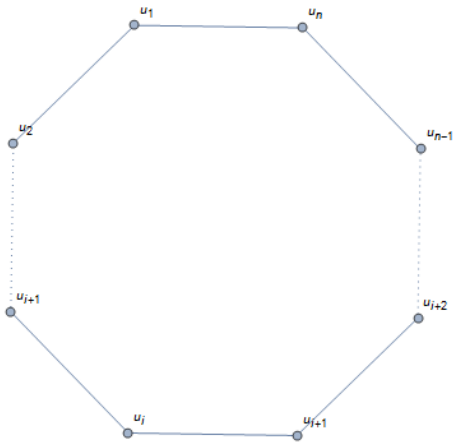


Figura 1.8: Un ciclu C_n

Definiție 1.4.5 *Un **graf complet** de n noduri reprezentat în Figura 1.9 este graful simplu $G = K_n$, unde*

$$V(K_n) = \{u_1, u_2, \ldots, u_n\},$$
$$E(K_n) = \{\{u_i, u_j\} : 1 \leq i < j \leq n\}.$$

În graful complet, fiecare pereche de noduri este conectată printr-o muchie.

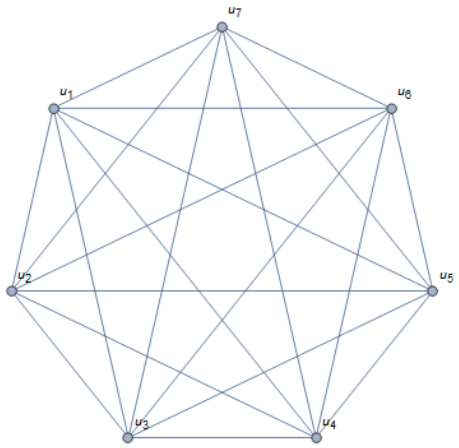


Figura 1.9: Un graph complet K_7

Definiție 1.4.6 *Un* ***(m,n)-graf bipartit complet*** *de $m+n$ noduri, în Figura 1.10, este un graf simplu $K_{m,n}$, unde*

$$V(K_{m,n}) = \{u_1, u_2, \ldots, u_m\} \cup \{v_1, v_2, \ldots, v_n\},$$
$$E(K_{m,n}) = \{\{u_i, u_j\} : 1 \leq i \leq m, 1 \leq j \leq n\}.$$

În graful bipartit complet sunt două tipuri de noduri și fiecare pereche de noduri diferite este adiacentă, însă nici o pereche de noduri de același tip nu este adiacentă.

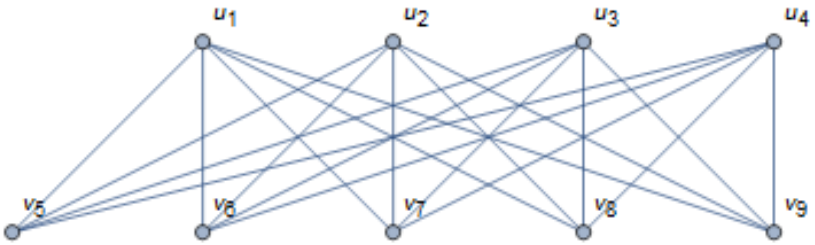


Figura 1.10: Un graph complet $K_{4,5}$

În diferite resurse, grafurile care nu sunt simpli (care au noduri izolate sau bucle) se mai numesc *multigrafuri*, cu referire la multitudinea de muchii. În genere, și grafurile simple și multigrafurile se numesc grafuri.

Când reprezentăm un graf într-o diagramă, din punct de vedere teoretic, informația ce contează este faptul dacă două noduri sunt conectate sau nu. Fie punctele ce reprezintă nodurile — mici, mari, ovale sau pătrate este irelevant. La fel și muchiile, pot fi curbe, în zig-zag sau linii drepte, iarăși, nu contează. În practică, însă, este un obicei comun de a reprezenta graful cu un număr minim de intersectări ale muchiilor.

Capitolul 2

Concepte de bază în Teoria Grafurilor

2.1 Gradul nodurilor și grafuri regulate

Multe proprietăți ale grafurilor pot fi exprimate în valori numerice asociate cu graful sau unele componente ale sale. Prima însușire ce o definim este *gradul* unui nod.

Definiție 2.1.1 *Fie $G = (V, E, \phi)$ un graf și $u \in V$, un nod din G.* ***Gradul nodului*** *u, notat $d_G(u)$, sau $d(u)$ când nu este pericol de ambiguitate, este definit de*

$$\begin{aligned} d_G(u) =& |\{e \in E : u \in \phi(e), |\phi(e)| = 2\}| \\ &+2|\{e \in E : u \in \phi(e), |\phi(e)| = 1\}|. \end{aligned}$$

Informal, $d_G(u)$ numără câte muchii își au capetele conectate la u. În acest caz, buclele le numărăm de două ori deoarece amândouă capete sunt unite la același nod.

Definiție 2.1.2 *Fie $G = (V, E, \phi)$ un graf și $u \in V$ un nod din G. Se numește* ***vecinătatea deschisă*** *a lui u, notată $N_G(u)$ sau doar $N(u)$, mulțimea tuturor vecinilor lui u în G. Asemănător,* ***vecinătatea închisă*** *a lui u, notată $N_G[u]$ sau $N[u]$ este mulțimea vecinilor lui u împreună cu el însuși.*

$$\begin{aligned} N_G(u) &= \{v \in V(G) : \phi = \{u, v\}, e \in E(G)\}, \\ N_G[u] &= N_G(u) \cup u. \end{aligned}$$

Putem defini vecinătatea unei submulțimi de noduri $S \subseteq V(G)$:

$$N_G(S) = \{v : v \in N_G(s), s \in S\},$$
$$N_G[S] = N_G(S) \cup S.$$

Vecinătatea deschisă(închisă) a unei submulțimi de noduri S este egală cu reuniunea vecinătăților deschise(închise) a fiecărui nod $u \in S$.

Se observă că numărul de vecini a unui nod u într-un graf G, niciodată nu este mai mare ca $d_G(u)$. Cu atât mai mult, un graf este un *graf simplu*, dacă și numai dacă, $d_G(u) = N_G(u)$ pentru fiecare $u \in V(G)$.

Observație 2.1.3 *Pentru orice graf G, este adevărată inegalitatea*

$$|N_G(u)| \leq d_G(u)$$

pentru fiecare $u \in V(G)$.

Definiție 2.1.4 *Un graf G se numește **k-regulat** dacă $d_G = k$ pentru orice $u \in V(G)$. Un graf este **regulat** dacă este k-regulat pentru un oarecare $k \geq 0$.*

Fie $m, n \in \mathbb{N}$.

1. Graful nul N_n este un graf 0-regulat.
2. Ciclul C_n este un graf 2-regulat, precum $d_{C_n}(u) = 2$ pentru fiecare $u \in V(C_n)$.
3. Graful complet K_n este un graf (n-1)-regulat.
4. Graful (m-n)-bipartit complet este regulat dacă și numai dacă, $m = n$, caz în care el este m-regulat.

2.2 Subgrafuri

În studiul grafurilor, adesea avem nevoie să ne focusăm atenția asupra unei porțiuni din graf, poate la un graf mai mic ce se află în graful mare. Această necesitate justifică utilizarea *subgrafurilor*.

Definiție 2.2.1 *Pentru grafurile $G' = (V', E', \phi')$ și $G = (V, E, \phi)$, spunem că G' este un **subgraf** a lui G dacă*

1. *$V' \subseteq V$,*
2. *$E' \subseteq E$,*
3. *$\phi'(e) = \phi(e)$ pentru toate $e \in E'$.*

*Indicăm relația de **subgraf** prin semnul includerii unei submulțimi $\subseteq$. Deci, dacă G' este un subgraf a lui G, scriem $G' \subseteq G$. Pentru grafurile simple $G' = (V', E')$ și $G = (V, E)$, $G' \subseteq G$ dacă $V' \subseteq V$ și $E' \subseteq E$.*

Relația de *subgraf* admite următoarele relații:

1. $G \subseteq G$, fiecare graf este și un subgraf al său.
2. dacă $G \subseteq G'$ și $G' \subseteq G$, atunci $G = G'$
3. dacă $G' \subseteq G''$ și $G'' \subseteq G$, atunci $G' \subseteq G$

2.3 Graf orientat

Până aici am definit două noțiuni importante în teoria grafurilor: graful (general) și graful simplu; și pe scurt am discutat câteva proprietăți. În unele cazuri, însă, pe lângă existența unui drum între două puncte, trebuie să ținem cont și de direcția acestui drum, ca de exemplu, străzile într-un oraș pot fi parcurse în ambele sensuri sau doar în *sens unic*.

Definiție 2.3.1 *Un **graf direcționat** sau un **digraf** este un triplet ordonat $vecG = (V, E, \eta)$, unde*

1. $V \neq \emptyset$,
2. $V \cap E = \emptyset$,
3. $\eta : E \to V \times V$ *este o aplicație (regulă).*

Mulțimea V este mulțimea de noduri, iar mulțimea E este mulțimea de textitmuchii direcționate. Dacă $\eta(e) = (u, v)$, atunci u se numește textitcoada muchiei e și v este *capul* (*vârful*) muchiei e. La fel, v este numit *succesorul* lui u, iar u este *predecesorul* lui v. Două muchii e și e' sunt *muchii paralele* dacă $\eta(e) = \eta(e')$, adică muchiilor le sunt asociate aceeași pereche ordonată de noduri.

Digraful este reprezentat în diagramă asemenea grafului, doar că muchiile nu sunt linii ci săgeți de la un punct (coada muchiei) la alt punct (vârful muchiei). Ilustrăm definiția printr-un exemplu.

Considerăm digraf-ul $\vec{G} = (V, E, \eta)$ cu 6 noduri și 7 muchii direcționate reprezentat în Figura 2.1. Observăm că $\vec{G} = (V, E, \eta)$, unde $V = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $E = e_1, e_2, e_3, e_4, e_5, e_6, e_7$ și funcția $\eta : E \to V \times V$ ia valorile

$$\begin{aligned}
\eta(e_1) &= (u_1, u_1),\\
\eta(e_2) &= (u_1, u_2),\\
\eta(e_3) &= (u_2, u_3),\\
\eta(e_4) &= (u_3, u_2),\\
\eta(e_5) &= (u_2, u_4),\\
\eta(e_6) &= (u_3, u_4),\\
\eta(e_7) &= (u_4, u_5).
\end{aligned}$$

Dacă $\vec{G} = (V, E, \eta)$ este un graf direcționat, atunci putem să îi creăm *graful* bază $G = (V, E, \phi)$, unde

$$\eta(e) = (u, v) \implies \phi(e) = \{u, v\}.$$

În schimb, crearea unui digraf dintr-un graf general dat nu poate fi efectuat într-o modalitate unică. Defapt, dintr-un graf general, putem crea 2^m digrafuri (m muchii a câte două direcții distincte).

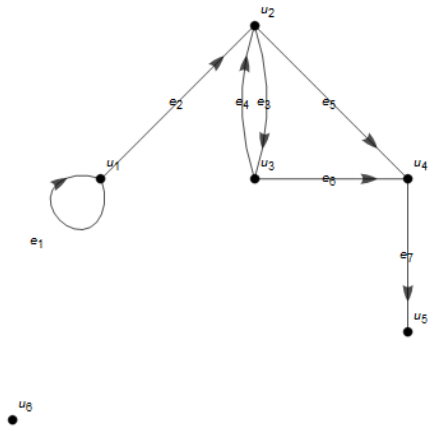


Figura 2.1: Un graf direcțtionat de 6 noduri și 7 muchii.

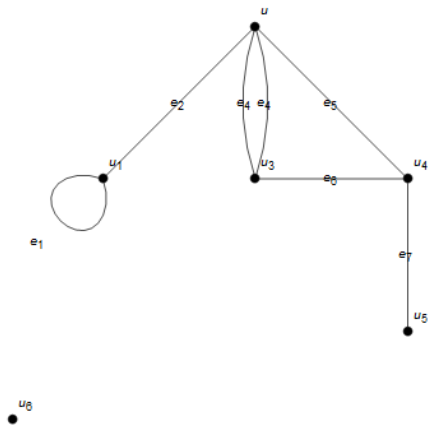


Figura 2.2: Graful de bază a grafului din Figura 2.1

2.4 Drumuri și cicluri

Din definiția grafului din Capitolul 1, esența unui graf este legătura între nodurile acestuia. În unele cazuri însă, nu avem nevoie de o conexiune *directă* prin o muchie, ci un traseu prin care am putea ajunge de la un nod la altul prin mai mulți pași. Chiar dacă două noduri pot să nu fie conectate printr-o muchie, ele pot avea un vecin comun sau, la general, pot fi conectate printr-un șir de muchii. Următoarele definiții captează această idee și unele varietăți ale acesteia.

Definiții 2.4.1 *1. Un **drum** într-un graf $G = (V, E, \phi)$ este o secvență*

$$(u_0, e_1, u_1, e_2, \ldots, e_k, u_k)$$

*de noduri și muchii ce începe și se termină cu un nod. Nodul u_0 este nodul inițial a drumului. Nodul n_k este **nodul final** al drumului. Primul și ultimul nod a drumului se mai numesc noduri de capăt a drumului. Numărul natural k este **lungimea drumului**.*

*2. Un **traseu** în G este un drum cu toate muchiile $e_1, e_2, \ldots, e_k$ distincte.*

*3. O **cale** în G este un drum cu toate nodurile $u_0, u_1, \ldots, u_k$ distincte.*

*4. Pentru nodurile u și v în G, un **u,v-drum** (u,v-traseu sau u,v-cale) este un drum (respectiv traseu, cale) cu nodul inițial u și nodul final v.*

*5. Un drum sau traseu de lungime $k \geq 1$ este **închis** dacă nodul inițial și nodul final este identic. Un traseu închis mai este numit și **circuit**.*

*6. Un **ciclu** este un drum închis cu noduri distincte, cu excepția celor inițiale și finale, care sunt identice.*

Din convenție, un singur nod îl putem considera un drum, un traseu și o cale de lungime 0. Cu toate aceste, ciclul mereu are lungime pozitivă și singurul ciclu de lungime $k = 1$ este bucla. La fel, mulțime de noduri și muchii ce formeaza un drum, traseu, ciclu sau cale în G este, evident, un subgraf din G.

Dacă G este un graf simplu, atunci un drum, traseu, cale sau ciclu poate fi definit printr-un șir de noduri $\{u_0, u_1, \ldots, u_k\}$ în loc de $\{u_0, e_1, u_1, e_2, u_2, \ldots, e_k, u_k\}$ astfel încât orice pereche de noduri adiacente u_{i-1} și u_i este determinată de o singură muchie $e = \{u_{i-1}, u_i\}$.

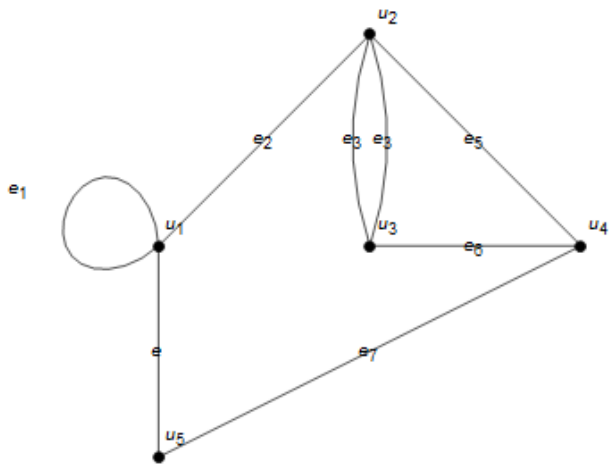


Figura 2.3: Un graf pentru a ilustra drumuri, trasee, căi și cicluri.

Ilustrăm Definițiile 2.4.1 pe graful din Figura 2.3

1. $w = (u_5, e_7, u_4, e_6, u_3, e_4, u_2, e_3, u_3, e_6, u_4)$
   Aici, w este un drum de lungime $k = 5$. Observați că w nu este un traseu, precum muchia e_6 o parcurgem de două ori. Dacă un drum nu este un traseu, cu atât mai mult nu este o cale. Deci, w nu este o cale.

2. $t = (u_1, e_2, u_2, e_3, u_3, e_4, u_2, e_5, u_4)$
   Drumul t este un traseu de lungime $k = 4$. Drumul t nu este o cale deoarece trecem pein nodul u_2 de două ori, însă prin muchii diferite — e_3 și e_4.

3. $p = (u_5, e_8, u_1, e_2, u_2, e_4, u_3, e_6, u_4)$
   Drumul p este o cale de lungimea $k = 4$.

4. $c_1 = (u_5, e_7, u_4, e_6, u_3, e_3, u_2, e_7, u_1, e_8, u_5)$
   Drumul c_1 este un drum închis de lungime $k = 5$.

5. $c_2 = (u_5, e_8, u_1, e_2, u_2, e_5, u_4, e_7, u_5)$.
   Drumul c_2 este un ciclu.

2.5 Grafuri conexe

Intuitiv, ne dăm seama ce este un graf conex, și anume faptul că putem începe în orice punct și să parcurgem un drum spre orice al nod de-a lungul muchiilor. Această idee ne conduce la următoarea definiție.

Definiție 2.5.1 *Un graf G este conex dacă pentru orice pereche de noduri distincte $u, v \in V(G)$, graful are un u, v-drum. Altminteri, graful nu este conex sau este deconectat.*

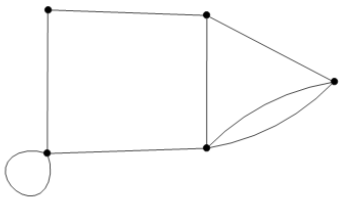
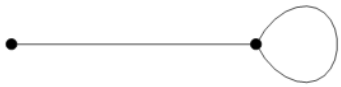


Figura 2.4: G



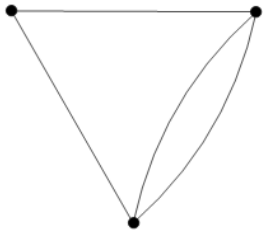


Figura 2.5: G'

Considerăm Figura 2.4 și Figura 2.5. Observăm că G este conex (conectat) iar G' este deconectat. Graful G' este format din două componente. Fiecare componentă este un subgraf lui G' și fiecare componentă în parte este conexă. Această parte conectată într-un graf se numește *componentă* sau *componentă conexă* a grafului.

Definiție 2.5.2 *Fie G un graf. Fie $H_1, \ldots, H_k$ subgrafuri conexe a lui G a căror mulțime de noduri și muchii sunt disjuncte și astfel încât ele acoperă întreg graful G. Adică,*

$$V(G) = V(H_1) \cup \cdots \cup V(H_k),$$
$$E(G) = E(H_1) \cup \cdots \cup E(H_k).$$

unde $V(H_i) \cap V(H_j) = E(H_i) \cap E(H_j) = \emptyset$ pentru fiecare i și j distincte. Fiecare subgraf H_i se numește componentă sau componentă conexă a lui G.

Capitolul 3

Reprezentarea grafurilor

Există diferite modalități de a stoca un graf în memorie și este important să înțelegem cum fiecare influențează complexitatea diferitor algoritmi. Reprezentările diferă prin modalitatea stocării și accesării informației. Deci, o reprezentare poate să aibă o trăsătură X favorabilă, însă o trăsătură Y ce ar avea complexitate mare, și invers, o reprezentare poate favoriza trăsătura Y în defavoarea trăsăturii X.

3.1 Matricea de adiacență Nod-Muchie

Definiție 3.1.1 *Matricea de adiacență Nod-Muchie este o matrice* $A = (a_{ij}) \in M_{n,m}(\{-1, 1, 0\})$*, în care fiecărei linii îi corespunde un nod și fiecărei coloane o muchie. Elementele matricei sunt date de formula:*

$$a_{ij} = \begin{cases} 1 & \text{dacă nodul } i \text{ este coada muchiei} \\ -1 & \text{dacă nodul } i \text{ este capul muchiei} \\ 0 & \text{dacă } \nexists e = \{i, j\} \end{cases}$$

3.2 Matricea de adiacență Nod-Nod

Definiție 3.2.1 *Matricea de adiacență Nod-Nod este o matrice* $A = (a_{ij}) \in M_{n,n}(\{1, 0\})$*, în care fiecare linie și coloană reprezintă un nod* $1 \leq i, j \leq n$*, iar intersecția unei coloane cu o linie indică o muchie direcționată* $e = (i, j)$*. Elementele matricei sunt date de formula:*

$$a_{ij} = \begin{cases} 1 & \text{dacă } \exists (i, j) \in E \\ 0 & \text{dacă nu există} \end{cases}$$

Putem observa că matricea de adiacență Nod-Nod a unui graf nedirecționat este simetrică față de diagonala principală.

Este oare această reprezentare optimă pentru o căutare în lățime? Pentru a determina dacă un nod este adiacent altor noduri, trebuie să efectuăm n operații pentru fiecare n linii. Pentru un graf conectat cu n noduri și m muchii, $n-1 \leq m \leq n(n-1)/2$, în total sunt $O(n^2)$ căutări.

3.3 Liste de adiacență Nod-Muchie

Fie un arbore care este foarte rar (are puține muchii) cu $|E| = |V| - 1$. Dacă l-am reprezenta prin matrici, am utiliza multă memorie. O reprezentare mai compactă este de a scrie fiecare nod și lista sa de adiacență.

Definiție 3.3.1 *Listele de adiacență reprezintă n liste înlănțuite, ce corespund fiecărui nod $v \in V$ și conțin toate nodurile $u \in V$ pentru care există o muchie direcționată $e = (u, v) \in E(G)$.*

Această reprezentare este cea mai eficientă din punct de vedere a utilizării memoriei și folosită în cei mai eficienți algoritmi. Listele de adiacență ocupă $2|E| + |V|$ memorie, pe când matricile de adiacență ocupă $|E| * |V|$ sau $|V|^2$ memorie.

3.4 Compararea reprezentărilor

3.4.1 Consumul de memorie

Reprezentarea	Memorie	Densitatea	Observații
Matrice Nod-Muchie	$O(mn)$	$2/n$	Ineficientă
Matrice Nod-Nod	$O(n^2)$	m/n^2	Eficientă pentru grafuri dense
Liste de adiacență	$O(m+n)$	1	Eficientă pentru grafuri rare

3.4.2 Avantaje și dezavantaje

Reprezentare	Avantaje	Dezavantaje
Matrice Nod-Muchie	Bună pentru reprezentări teoretice	Nu este eficientă în practică
Matrice Nod-Nod	Existenței muchiei în $O(1)$	Lista vecinilor în timp $O(n)$
Liste de adiacență	Lista vecinilor în timp $O(d_G(i))$	Existența muchiei în timp $O(d_G(i))$

Capitolul 4

Algoritmi de parcurgere

Fie dat un graf $G = (V, E)$ și nodul sursă $s \in V$. Punem problema determinării conexității grafului.

4.1 Algoritmul general de parcurgere

Algoritmul general de căutare începe cu un nod s, îl face rădăcina arborelui și îl adaugă în lista L. Algoritmul apoi alege la fiecare iterare un nod din listă și caută un nod adiacent acestuia, dacă găsește, îl adaugă în listă. Procesul continuă până când toate nodurile sunt vizitate. Mecanismul de selecție a nodului din listă diferă de la o căutare la alta.

Următoarea procedură demonstrează acest algoritm în pseudocod

Data: $L = \{s\}$, $V = V \setminus \{s\}$.
Result: Parcurgerea întregului graf
while $L \neq \emptyset$ **do**
  Alege $v \in L$;
  if *Există* $N_E(v) = u, u \in V$ **then**
    $L = L \cup \{u\}$;
    $V = V \setminus \{u\}$;
    $E = E \setminus \{\{v, u\}\}$;
  else
    $L = L \setminus \{v\}$;
  end
end

Algorithm 1: Parcurgerea generică

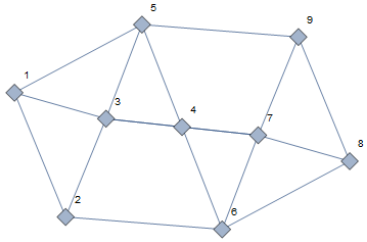


Figura 4.1: Graful G de parcurs.

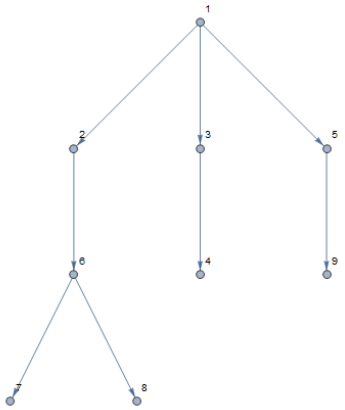


Figura 4.2: Arborele de căutare în lățime (BFS).

4.2 Parcurgere în lățime (BFS)

Teoremă 4.2.1 ***Parcurgerea în lățime*** *sau* ***BFS*** *este o parcurgere generică ce folosește strategia first-in-first-out(FIFO) și este implementată deobicei cu o* ***coadă****. Parcurgerea în lățime are complexitatea $O(m+n)$ în timp.*

Pentru a obține pseudocodul *BFS*, este deajuns să modificăm mecanismul de alegere a nodului în pseudocodul parcurgerii generice, astfel încât să eliminăm nodurile **de la începutul listei** L.

La căutarea în lățime, graful este parcurs într-o manieră completă. Adică, toți vecini neparcurși a unui nod sunt adăugați la capătul listei L.

Ținem cont de faptul că arborii de parcurgere în lățime nu este unic și depinde de nodul sursă și de ordinea în care vecinii sunt vizitați. Nu toate muchiile din pot fi prezente în arbore. Cu toate acestea, știm că în graful original nu poate exista o muchie ce ar uni două nivele neconsecutive din arbore.

Definiție 4.2.2 *Nivelul nodului, d(u), este distanța de la s la u.*

Mai întâi de toate, este clar că toate nodurile accesibile din nodul s va fi vizitat și va avea un $d(v)$. Prin inducție pe $d(v)$, coada Q rămâne ordonată crescător

după d și conține noduri cu $d = k$ sau $d = k + 1$. Desigur, nodurile $d = k$ sunt eliminate primele din coadă și adaugă la sfârșitul cozii noduri de nivel $d = k+1$. Doar atunci când au fost eliminate toate nodurile din coadă, încep a fi introduse noduri de nivel $d = k + 2$.

Și iarăși, prin inducție, la începutul nivelului k, toate nodurile de distanță $s < k$ au fost deja introduse și eliminate din coadă, doar acelea își au nivelul bine determinat. Să cercetăm cazul pentru un nivel k. Fie un nod v de distanță $k+1$. Acesta nu putea fi introdus în coadă la începutul nivelului k, deci presupunem că $d(v) = \infty$. Însă graful este conex, deci există cel puțin un nod u la distanța k care este adicent nodului v. Nodul $u \in Q$, și presupunem că $d(u) = k$. Nodul u din Q va fi urmat de nodul v, deci $d(v) = d(u) + 1 = k + 1$. Nodurile de distanță $d > k + 1$ nu au un nod adiacent la distanța $d = k$ și nu vor fi inserați în coadă. Această proprietate este comună pentru orice parcurgere în lățime.

Corolară 4.2.3 *Fie un graf G parcus în lățime. Nu există o muchie $\{i, j\}$ în care $|d(i) - d(j)| > 1$. Adică, vecinii trebuie să se afle în nivele adiacente.*

4.3 Parcurgere în adâncime (DFS)

Definiție 4.3.1 ***Parcurgerea în adâncime*** *sau* ***DFS*** *este un o parcurgere generică ce folosește strategia last-in-first-out(FIFO) și este implementată deobicei cu o* ***stivă****. Parcurgerea în lățime are complexitatea $O(m + n)$ în timp.*

Parcurgerea în adâncime poate fi la fel obținută din psedocodul parcurgerii generice, modificând mecanismul de eliminare a nodurilor din listă, astfel încât nodurile să fie eliminate de la sfârșitul listei.

Căutând în adâncime, căutăm un vecin nevizitat a unui nod. Dacă un astfel de nod este găsit, este introdus la sfârșitul stivei și parcurgerea continuă imediat cu el, altminteri, dacă nici un nod nevizitat nu este găsit, îl eliminăm din vârful stivei și selectăm următorul nod de la sfârșitul listei.

4.4 Aplicările parcurgerilor

4.4.1 Verificarea conexității unui graf

Fie un graf G și pornind de la un nod $s \in V$, îl parcurgem. Dacă parcurgând-ul am vizitat toate noduril, deci toate nodurile pot fi accesate din nodul s, deci este un graf conex. Dacă însă o parte din noduri nu au fost vizitate, există o *ruptură* în graf ce nu permite algoritmului să treacă de la o *componentă* a grafului la alta.

Avem nevoie doar de o singură parcurgere DFS sau BFS alegând un nod arbitrar s, căci dacă toate nodurile $v \in V$ pot fi accesate din s, atunci putem creea minim un traseu dintr-un nod în oricare altul care va conține nodul s. Astfel, verificarea conexității unui graf ia $O(m + n)$ timp.

4.4.2 Verificarea existenței ciclurilor în graf

Definiție 4.4.1 *Un graf este* ***aciclic*** *dacă acesta nu conține nici un ciclu.*

Verificăm această proprietate la fel prin aplicarea parcurgerii în lățime sau adâncime. În acest caz însă, parcurgerea în lățime găsește un ciclu mai rapid deoarece aceasta se va depărta mai repede de rădăcină și, posibil că, spre un cilcu. Cu atât mai mult, dacă ciclul este un ciclu C_k, atunci un BFS ar fi trebuit să parcurgă cel puțin $kdiv2$ nivele.

4.4.3 Verificăm dacă graful este bipartit

Știm că dacă un graf este bipartit, dacă și numai dacă acesta nu conține un ciclu de lungime impară. Deci, putem face o parcurgere în lățime și dacă aceasta găsește muchii între nodurile de același nivel, graful conține un ciclu de lungime impară, deci graful nu este bipartit.

Bibliografie

[1] Raymond Greenlaw Geir Agnarsson. *Graph Theory — Modeling, Applications, and Algorithms.* Pearson Prentice Hall, 2007.

[2] Eva Tardos Jon Kleinberg. *Algorithm Design.* Pearson Education, Inc., 2006.

[3] Dorit Hochbaum. *Lecture Notes for IEOR 266: Graph Algorithms and Network Flows.* Department of Industrial Engineering and Operations Research, University of California, Berkeley., 2014.

[4] Leiserson Charles E. Rivest Ronald L. Cormen, Thomas H. *Introduction to Algorithms (1st ed.).* MIT Press and McGraw-Hill., 1990.