

UNIVERSITATEA „ALEXANDRU IOAN CUZA”, IAȘI
FACULTATEA DE MATEMATICĂ



ALGORITMI PENTRU GRAFURI ȘI APLICAȚII

Lucrare de licență

Conducător științific:
Lect.Dr. Ana-Maria Moșneagu

Student:
Galbiniță Sebastian

Septembrie, 2020
Iași

Cuprins

Introducere	1
1 Noțiuni introductive	2
1.1 Graf	2
1.2 Reprezentarea unui graf	3
1.3 Gradul unui nod	5
1.4 Drumuri si cicluri	5
1.5 Conexitate	6
2 Drumuri minime de sursă unică	7
2.1 Reprezentarea drumurilor minime	7
2.2 Relaxare	9
2.3 Algoritmul Dijkstra	10
2.3.1 Descrierea algoritmului	10
2.3.2 Implementare	11
3 Drumuri minime între toate perechile de vârfuri	13
3.1 Drumuri minime	14
3.2 Algoritmul Floyd-Warshall	16
3.2.1 Descrierea algoritmului	16
3.2.2 Implementare	18
4 Flux maxim	19
4.1 Fluxuri și rețele de transport	19
4.2 Metoda lui Ford-Fulkerson	21
4.2.1 Implementare	22
5 Aplicație	24
Bibliografie	28

Introducere

Multe aplicații computaționale invocă nu numai o pereche de obiecte dar și un set de conexiuni care să lege aceste informații între ele. Relația impusă dintre aceste conexiuni a condus la mai multe întrebări precum: Există posibilitatea să ajungi de la un obiect la altul urmărind conexiunile? La câte obiecte pot să ajung pornind de la unul deja prestabilit? Care ar fi cel mai scurt/lung drum parcurs pentru a ajunge la destinația propusă? Există o legătură între toate tipurile de date? Pentru a ilustra diversitatea aplicațiilor ce folosesc proprietăți și metode, care au ca fundament grafurile, enumerăm următoarele exemple: Hărțile, Hypertexts (documente ce conțin referințe către alte pagini web), Circuite, Tranzacții, Rețea de internet etc.

Pentru a putea răspunde la întrebările de mai sus, vom folosi obiecte abstracte cum ar fi grafurile. Grafurile sunt structuri de date extrem de răspândite în știința calculatoarelor, iar algoritmi pentru grafuri sunt esențiali în acest domeniu.

În capitolul 1 sunt prezentate câteva noțiuni introductive legate de grafuri, cum ar fi: reprezentarea grafurilor pe calculator, noțiunea de grad și drum și aspecte legate de conexitatea unui graf.

În capitolul 2 se tratează problema de determinare a drumurilor de lungime minimă de la un nod fixat la toate celelalte noduri, atunci când fiecărei muchii îi este asociată o lungime pozitivă. Se va prezenta algoritmul de relaxare aplicat unei muchii iar la final se va descrie și implementa algoritmul lui Dijkstra pentru problema de drum minim.

În capitolul 3 se va rezolva problema determinării a tuturor drumurilor de cost minim între oricare două noduri. Pentru determinarea unei soluții optime se va studia structura unui drum minim și se vor prezenta algoritmi necesari pentru rezolvarea acestei probleme.

În final, în capitolul 4 se va rezolva problema de flux maxim care se referă la determinarea celei mai mari cantități de material care poate fi transportată de la sursă la destinație ținând cont de restricțiile de capacitate într-un graf orientat ponderat. Definim formal problema fluxului maxim, noțiunile de rețea de transport și flux de rețea iar la final se va descrie metoda lui Ford-Fulkerson pentru găsirea fluxului maxim.

Capitolul 1

Noțiuni introductive

1.1 Graf

Un *graf* $G = (V, E)$ este o pereche de mulțimi astfel încât $E \subseteq V \times V$. Elementele din mulțimea V se numesc *noduri* ale grafului G iar elementele mulțimii E se numesc *muchii*. Modul obișnuit de a ilustra un graf este prin desenarea unui punct pentru fiecare nod și unirea celor două puncte printr-o linie, dacă cele două vârfuri corespunzătoare formează o muchie.

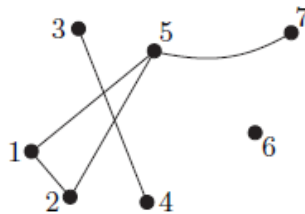


Figure 1.1: Graful pe $V = \{1, 2, 3, \dots, 7\}$ cu setul de muchii
 $E = \{\{1, 2\}, \{1, 5\}, \{2, 5\}, \{3, 4\}, \{5, 7\}\}$

Se spune că un graf cu setul de noduri V este un graf pe V . Setul de noduri al unui graf G este notat cu $V(G)$, muchiile sale fiind $E(G)$. Aceste convenții sunt independente de orice alte denumiri ale acestor două seturi: setul de noduri W a grafului $H = (W, F)$ este tot notat prin $V(H)$, nu prin $W(H)$. Numărul de noduri ale unui graf G dă *ordinul* acestuia, notat prin $|G|$; numărul său de muchii este notat cu $||G||$. Grafurile pot fi de ordin *finit*, *infini* sau *numărabil*. Pentru *graful null* (\emptyset, \emptyset) scriem pur și simplu \emptyset . Un graf de ordin 0 sau 1 se numește *trivial*.

Un nod v este *incident* cu o muchie e dacă $v \in e$; atunci e este o muchie la v . O muchie $\{u, v\}$ este de obicei scrisă sub forma (u, v) . Vârfurile u și v se numesc *extremitățile muchiei* (u, v) .

Două noduri ale grafului G sunt *adiacente* sau *vecine* dacă (x, y) formează o muchie pe G . Două muchii care au o extremitate comună, se numesc *muchii incidente*. Dacă toate vârfurile lui G sunt perechi adiacente, atunci G este *complet*.

Fie $G = (V, E)$ și $G' = (V', E')$ două grafuri. Numim G și G' **izomorfe** și notăm $G \simeq G'$ dacă există o bijecție $\varphi : V \rightarrow V'$ cu $(x, y) \in E \Leftrightarrow (\varphi(x), \varphi(y)) \in E'$ pentru orice $x, y \in V$.

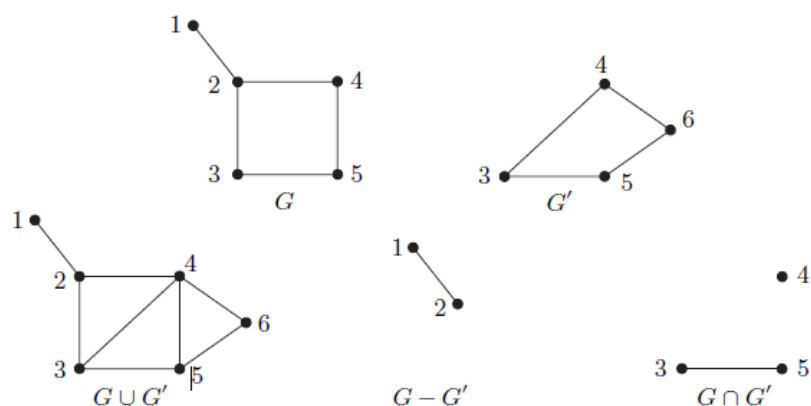


Figure 1.2: Reuniunea, diferența și intersecția; vârfurile 2,3,4 formează un triunghi în $G \cup G'$ dar nu în G

Stabilim că $G \cup G' = (V \cup V', E \cup E')$ și $G \cap G' = (V \cap V', E \cap E')$. Dacă $G \cap G' = \emptyset$, atunci G și G' sunt *disjuncte*. Dacă $V' \subseteq V$ și $E' \subseteq E$, atunci G' este un *subgraf* al lui G (și G este un *supergraf* pentru G'), scris ca $G' \subseteq G$. Mai puțin formal, spunem că G îl conține pe G' .

1.2 Reprezentarea unui graf

Un *graf neorientat* [3] G este o pereche (V, E) , unde V reprezintă o mulțime finită numită *mulțimea nodurilor* lui G , iar E este o mulțime de perechi neordonate $(u, v) \in V \times V$, numită *mulțimea muchiilor* lui G . Prin urmare, fiecare muchie este o pereche neordonată de noduri $(u, v) \in E$, prin care este stabilită o relație de vecinătate între $u, v \in V$. Figura 1.3(a) este o reprezentare grafică a unui graf neorientat.

Într-un *graf orientat* [3] $G = (V, E)$, *mulțimea arcelor* E este constituită din perechi de vârfuri ordonate și nu din perechi neordonate. Arcul $(u, v) \in E$ este reprezentat printr-o săgeată de la nodul u la v cu semnificația că există o relație de la u la v . Figura 1.4(a) este o reprezentare grafică a unui graf orientat.

Există două moduri de reprezentare a unui graf $G = (V, E)$: ca o colecție de liste de adiacență sau ca o matrice de adiacență. Oricare dintre aceste

două modalități de reprezentare, se aplică atât grafurilor orientate cât și celor neorientate.

Reprezentarea prin liste de adiacență [2] a grafului $G = (V, E)$ constă în realizarea unui tablou M cu $|V|$ liste, o listă pentru fiecare vârf din V . Pentru fiecare $u \in V$, $M[u]$ conține toate vârfurile v astfel încât există o muchie $(u, v) \in E$. Figura 1.3(b) este o reprezentare prin liste de adiacență a grafului neorientat din Figura 1.3(a). Atât pentru grafurile orientate cât și pentru cele neorientate, reprezentarea lor prin liste de adiacență au o dimensiunea de memorie de $O(V + E)$.

Listele de adiacență pot fi adaptate și pentru reprezentarea unor **grafuri ponderate** astfel, fiecărei muchii a grafului G i se asociază o funcție numită **funcție de cost** $f : E \rightarrow \mathbb{R}$, unde costul $f(u, v)$ al muchiei (u, v) este memorat împreună cu v în $M[u]$.

În majoritatea cazurilor, listele de adiacență se folosesc atunci când vrem să gestionăm memoria programului cât mai eficient. De exemplu, dacă avem un graf *rar* ($|E|$ este mult mai mic decât $|V|(|V| - 1)/2$) și am folosi reprezentarea grafului cu ajutorul matricelor, atunci majoritatea elementelor din matrice ar rămâne nefolosite, producând astfel o risipă de memorie.

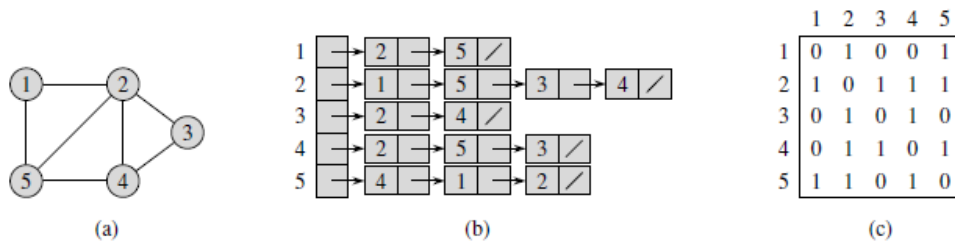


Figure 1.3: Două reprezentări a unui graf neorientat. (a) Graf neorientat. (b) Listă de adiacență pentru G . (c) Matricea de adiacență a lui G .

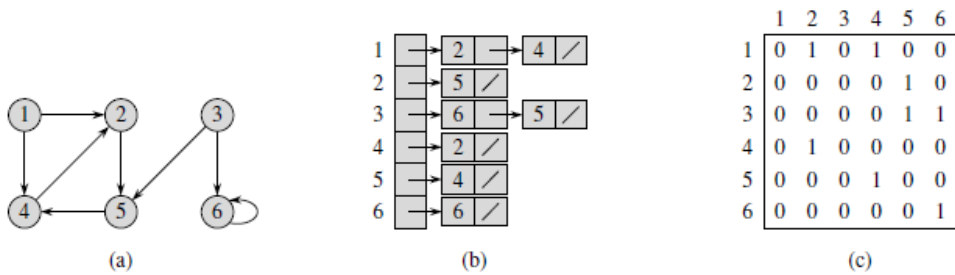


Figure 1.4: Două reprezentări a unui graf orientat. (a) Graf orientat. (b) Listă de adiacență pentru G . (c) Matricea de adiacență a lui G .

Pentru **reprezentarea prin matrice de adiacență** [2], presupunem că vârfurile sunt numerotate arbitrar. Reprezentarea matricii de adiacență

a grafului $G = (V, E)$ constă într-o matrice $A_{|V| \times |V|} = (a_{ij})$ a.î.:

$$a_{ij} = \begin{cases} 1, (i, j) \in E \\ 0, (i, j) \notin E \end{cases}$$

Figurile 1.3(c) și 1.4(c) sunt reprezentările matricilor de adiacență a grafurilor 1.3(a), respectiv 1.4(a). Necesarul de memorie este de $\Theta(|V|^2)$ și nu depinde de numărul de muchii a grafului. În plus, când se face implementarea cu ajutorul matricelor, verificarea dacă este o muchie între cele două vârfuri durează $\Theta(1)$ timpi, în timp ce cu ajutorul listelor de adiacență ar putea avea un ordin de complexitate liniar $\Theta(n)$, unde n este numărul de noduri.

Matricile de adiacență pot fi folosite și pentru grafuri ponderate. Astfel, fie un graf orientat ponderat $G = (V, E)$ și funcția de cost f de mai sus. Costul $f(u, v)$ al unei muchii $(u, v) \in E$ este memorat ca un element din matrice. În cazul în care o muchie nu există, elementul corespunzător din matrice poate fi *NIL*.

Un avantaj pentru folosirea matricilor în locul listelor este acela că dacă graful este *dens* ($|E|$ este aproximativ egal cu $|V|^2$) atunci numărul de muchii este aproape de $n(n-1)/2$, unde $n = |V|$.

1.3 Gradul unui nod

Fie $G = (V, E)$ un graf neorientat nenul. Mulțimea vecinilor nodului v în G este notată cu $N_G(v)$ sau pe scurt $N(v)$. Mai general, pentru $U \subseteq V$, vecinii din $V \setminus U$ ai nodurilor din U se numesc vecini lui U ; mulțimea lor este notată cu $N(U)$.

Gradul $d_G(v) = d(v)$ a unui nod v este numărul de muchii $|E|$ la v . Dacă gradul unui nod este 0 atunci acesta se numește *nod izolat*. Numărul $\delta(G) = \min \{d(v) \mid v \in V\}$ este *gradul minim* [1] a lui G , iar numărul $\Delta(G) = \max \{d(v) \mid v \in V\}$ este *gradul maxim* [1] a lui G . Dacă toate nodurile lui G au același grad k , atunci G este *k-regulat*, sau simplu *regulat*. Un graf 3-regulat se numește *cub*.

Numărul

$$d(G) = \frac{1}{|V|} \sum_{v \in V} d(v)$$

se numește *gradul mediu* [1] a lui G . Deasemena, are loc relația,

$$\delta(G) \leq d(G) \leq \Delta(G)$$

1.4 Drumuri si cicluri

Drumul este un graf orientat nenul $= (V, E)$ de forma

$$V = \{v_1, v_2, \dots, v_k\} \quad E = \{(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)\}$$

unde toate nodurile v_i sunt distincte. Numărul de muchii a unui drum reprezintă *lungimea* lui. Un *drum elementar* este un drum în care nodurile sunt distincte două câte două.

Fiind date două mulțimi A, B de noduri, spunem că $P = [x_0, x_1, \dots, x_k]$ este un *drum* A - B dacă $V(P) \cap A = \{x_0\}$ și $V(P) \cap B = \{x_k\}$. Două sau mai multe drumuri sunt *independente* dacă nici unul dintre ele nu conține un nod interior al altuia.

Dacă $P = [x_0, x_1, \dots, x_{k-1}]$ este un drum și $k \geq 3$, atunci graful $C = P + (x_{k-1}, x_0)$ este un *ciclu*. Ca și la drumuri, vom nota ciclul după secvența de noduri pe care o are; ciclul de mai sus C poate fi scris sub forma $C = [x_0, \dots, x_{k-1}, x_0]$.

Distanța dintre două noduri în G , $d_G(x, y)$ este lungimea celui mai scurt drum $x - y$ în G ; dacă nu există un astfel de drum vom nota $d(x, y) = \infty$. Cea mai mare distanță dintre oricare două noduri în G este *diametrul* lui G , notată cu *diam* G .

1.5 Conexitate

Un graf neorientat nenul $G = (V, E)$ se numește **conex** dacă pentru orice $u, v \in V$, $u \neq v$ există cel puțin un drum de la u la v . Un subgraf maximal conex la G se numește *componentă conexă* la G . Mai general, pentru orice subgraf $S = (V_1, E_1)$ la G , S este *convex* și nu există un alt subgraf la G , $S' = (V_2, E_2)$ cu $V_1 \subset V_2$ care să fie conex. Un graf cu un singur nod este graf conex.

Pentru grafurile orientate, vom evidenția două noțiuni asociate cu noțiunea de conexitate. Un graf orientat se numește *slab conex* [3] dacă înlocuirea tuturor muchiilor orientate, cu muchii ale unui graf neorientat produce un graf conex (neorientat). Un graf orientat este *tare conex* [3] dacă oricare ar fi două noduri $u, v \in E$, există drum și de la u la v , și de la v la u . Subgraful $C = (V_1, E_1)$ a grafului orientat G , este *componentă tare conexă* dacă este tare conexă și nu există un alt subgraf al lui G care să fie tare conex.

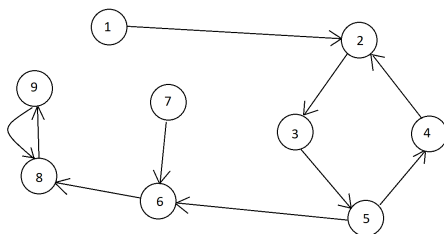


Figure 1.5: Acest graf nu este tare conex pentru că nu există drum de la 4 la 1, dar are 5 componente tare conexe: subgraful $\{2, 3, 4, 5\}$, subgraful $\{8, 9\}$ și 3 subgrafuri cu câte un singur nod: $\{1\}$, $\{6\}$ și $\{7\}$.

Capitolul 2

Drumuri minime de sursă unică

Să presupunem că un ciclist dorește să parcurgă drumul de la Iași la Bacău utilizând o hartă rutieră a României, unde sunt indicate distanțele între fiecare două intersecții adiacente.

O posibilă rezolvare a acestei probleme este aceea de a înșirui toate drumurile de la Iași la Bacău și, pe baza lungimilor acestora, de a alege cel mai scurt drum dintre ele. Este vizibil de observat faptul că numărul de variante posibile este un număr foarte mare chiar și în cazul în care avem drumuri care nu conțin cicluri.

În acest capitol vom arăta cum poate fi rezolvată problema în mod eficient. Într-o **problemă de drum minim**, avem în ipoteză un graf orientat ponderat $G = (V, E)$, iar funcția cost $f : E \rightarrow \mathbb{R}$ repartizează fiecărei muchii un cost exprimat printr-un număr real. **Costul** drumului $p = [\alpha_0, \alpha_1, \dots, \alpha_k]$ reprezintă suma costurilor corespunzătoare muchiilor componente :

$$f(p) = \sum_{i=1}^k f(\alpha_{i-1}, \alpha_i)$$

Așadar, **costul unui drum minim** de la u la v este dat de

$$\delta(u, v) = \begin{cases} \min \{f(p) : u \rightsquigarrow v\}, & \text{dacă există drum de la } u \text{ la } v \\ \infty, & \text{altfel} \end{cases}$$

În cazul exemplului de mai sus, putem modela harta rutieră ca un graf: vârfurile constituie punctele de intersecție, muchiile reprezintă segmentele de drum iar costurile, distanțele între intersecții.

2.1 Reprezentarea drumurilor minime

Fiind dat un graf $G = (V, E)$, se va reține pentru fiecare nod $v \in V$ un **predecesor** $\omega[v]$ care este fie un nod, fie NULL. Pentru determinarea dru-

murilor minime, algoritmi prezentați în acest capitol determină ω așa încât pentru orice vârf v , lanțul de predecesori care pornește de la v să coincidă unei traversări în ordinea inversă a unui drum de valoare minimă de la vârful sursă s la v .

Pe durata execuției a unui algoritm pentru determinarea unui drum minim, valorile lui ω nu arată în mod necesar drumurile minime. Astfel, vom considera **subgraful predecesor** $G_\omega = (V_\omega, E_\omega)$, unde V_ω reprezintă mulțimea vârfurilor din G cu proprietatea că au predecesor diferit de NULL, reunită cu mulțimea constituită din vârful s :

$$V_\omega = \{v \in V : \omega[v] \neq NULL\} \cup \{s\}$$

Mulțimea de muchii E_ω este mulțimea de muchii impusă de valorile lui ω pentru vârfurile din V_ω :

$$E_\omega = \{(\omega[v], v) \in E : v \in V_\omega \setminus \{s\}\}$$

Valorile ω determinate de algoritmi ce vor fi prezentați, asigură că la terminare, G_ω să fie un "arbore al drumurilor minime". Mai precis, există un arbore cu rădăcină care conține câte un drum minim de la sursa s la orice nod al grafului care este accesibil din nodul sursă s .

Fie $G = (V, E)$ un graf orientat ponderat, având funcția de cost $f : E \rightarrow \mathbb{R}$, presupunem că graful G nu conține cicluri de cost negativ, disponibile din vârful sursă s , așadar drumurile minime sunt bine definite. Un **arbore al drumurilor minime** de rădăcină s este subgraful orientat $G' = (V', E')$ unde $V' \subseteq V$, $E' \subseteq E$ a.î. următoarele condiții sunt îndeplinite:

1. G' arbore cu rădăcină, având pe s ca rădăcină.
2. V' mulțimea vârfurilor accesibile din s în G .
3. Pentru orice $v \in V'$ unicul drum de la s la v în G' este un drum minim de la s la v în G .

Drumurile minime nu sunt întotdeauna unice și în consecință există mai mulți arbori de drumuri minime.

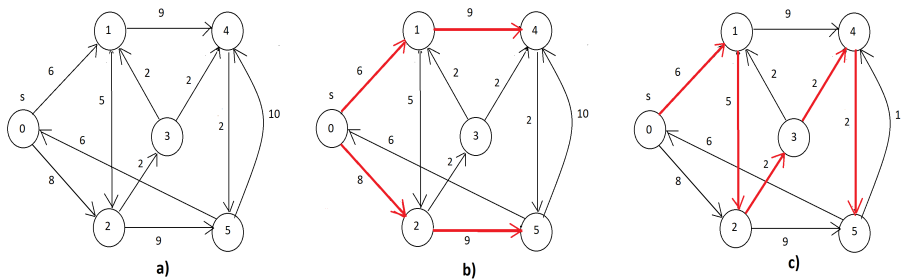


Figure 2.1: **a)** Graf orientat ponderat. **b)** Muchiile reprezentate cu roșu formează un arbore de drumuri minime având ca rădăcină sursa s . **c)** Un alt exemplu de arbore de drumuri minime având aceeași rădăcină.

2.2 Relaxare

Algoritmi pentru determinarea drumurilor minime de sursă unică sunt bazați pe o tehnică care poartă numele de **relaxare**. Pentru fiecare nod $v \in V$, conservăm un atribut $d[v]$, care reprezintă o margine superioară a costului de drum minim de la s la v . Numim acest atribut $d[v]$ o **estimare a drumului minim**. Estimările predecesorilor și a drumurilor minime sunt inițializate prin următorul algoritm:

INIȚIALIZEAZĂ-SURSĂ-UNICĂ (G, s)

1: **for** fiecare vârf $v \in V(G)$

2: $d[v] \leftarrow \infty$

3: $\omega[v] \leftarrow NULL$

4: $d[s] \leftarrow 0$

După inițializare $\omega[v] = NULL$ pentru orice vârf $v \in V$, $d[v] = 0$ pentru $v = s$ și $d[v] = \infty$ pentru $v \in V \setminus \{s\}$

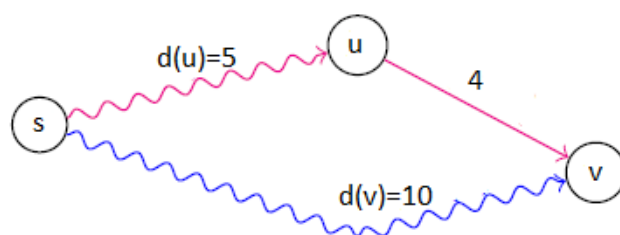


Figure 2.2: Are loc procesul de relaxare a unei muchii (u, v) cu costul $f(u, v) = 4$. Pentru orice vârf $u, v \in V$ este prezentată estimarea drumului minim. Înainte de relaxare, $d[v] > d[u] + f(u, v)$, valoarea lui $d[v]$ scade și va fi egală cu 9. În cazul în care $d[v] \leq d[u] + f(u, v)$, valoarea lui $d[v]$ va rămâne neschimbată.

Acest proces numit **relaxare** aplicat unei muchii (u, v) verifică dacă drumul minim la v , poate fi îmbunătățit pe baza lui u , și în caz afirmativ se reactualizează $d[v]$ și $\omega[v]$. Codul de mai jos realizează un pas de relaxare a unei muchii (u, v) .

RELAXEAZĂ (u, v, f)

1: **if** $d[v] > d[u] + f(u, v)$

2: $d[v] \leftarrow d[u] + f(u, v)$

3: $\omega[v] \leftarrow u$

În figura 2.1 este aplicat algoritmul de mai sus, astfel estimarea drumului minim scade.

Toți algoritmi apelează INIȚIALIZEAZĂ-SURSĂ UNICĂ după care apelează relaxarea repetată a muchiilor. În algoritmul Dijkstra fiecare muchie

este relaxată doar o singură dată iar în cazul algoritmului Bellman-Ford, fiecare dintre muchii este relaxată de mai multe ori.

2.3 Algoritmul Dijkstra

Algoritmul lui Dijkstra este cel mai utilizat algoritm de căutare pentru problema de drum minim. Algoritmul a fost propus de olandezul Edsger Dijkstra în anul 1959. Algoritmul Dijkstra calculează cel mai scurt drum selectând vârful nevizitat cu cea mai mică distanță față de fiecare vecin nevizitat. Pentru un graf cu n noduri având costuri negative pe muchii, metoda calculează calea cu cel mai mic cost între o pereche de noduri cu o complexitate de $O(n^2)$. Algoritmul lui Dijkstra calculează cel mai scurt drum de la nodul sursă până la destinație calculând iterativ cele mai scurte drumuri de la nodul sursă la toate celelalte noduri din graf.

2.3.1 Descrierea algoritmului

Fie un tablou $d[\]$ unde pentru fiecare vârf v stocăm lungimea curentă a celui mai scurt drum de la s la v în $d[v]$. Inițial $d[s] = 0$, iar pentru toate celelalte vârfuri această lungime este egală cu INT_MAX . În implementare, un număr suficient de mare (care este garantat a fi mai mare decât orice lungime posibilă) este ales ca infinit.

$$d[v] = \infty, v \neq s$$

În plus menținem un tablou boolean $u[\]$ care stochează pentru fiecare vârf v dacă este marcat. Inițial toate vârfurile sunt nemarcate $u[v] = \text{false}$.

Algoritmul lui Dijkstra rulează pentru n iterații. Evident, în prima iterație, vârful de pornire s va fi selectat. Vârful selectat v este marcat. În continuare, de la vârful v se realizează relaxări: toate muchiile de forma (v, i) sunt luate în considerare și pentru fiecare vârf i , algoritmul încearcă să îmbunătățească valoarea $d[i]$. Dacă lungimea muchiei curente este egală cu $f(v, i)$, atunci:

$$d[i] = \min(d[i], d[v] + f(v, i))$$

După ce toate aceste muchii sunt luate în considerare, iterația curentă se termină. În cele din urmă după n iterații, toate vârfurile vor fi marcate și algoritmul se încheie. Astfel, valorile găsite $d[v]$ sunt lungimile celor mai scurte drumuri de la s la toate vârfurile v . O observație ar fi că, dacă unele vârfuri nu pot fi atinse din cele de început, valorile $d[v]$ pentru ele vor rămâne infinite. Evident, ultimele câteva iterații ale algoritmului vor alege acele vârfuri, dar nu se va lucra pentru ele. Prin urmare, algoritmul poate fi oprit imediat ce vârful selectat are o distanță infinită față de s .

În esență, acest algoritm rezolvă eficient problema drumurilor minime de sursă unică într-un graf orientat ponderat $G = (V, E)$, muchiile fiind nenegative. Vom presupune că $f(u, v) \geq 0$ pentru fiecare $(u, v) \in E$.

```

DIJKSTRA ( $G, f, s$ )
1: ÎNȚĂLIZEAZĂ-SURSĂ-UNICĂ( $G, s$ )
2:  $S \leftarrow \emptyset$ 
3:  $Q \leftarrow V(G)$ 
4: while  $Q \neq \emptyset$ 
5:    $u \leftarrow \text{EXTRAGE-MIN}(Q)$ 
6:    $S \leftarrow S \cup \{u\}$ 
7:   for fiecare vârf  $v \in \text{Adj}[u]$ 1
8:     RELAXEAZĂ( $u, v, f$ )

```

Algoritmul Dijkstra aplică metoda relaxare pentru fiecare muchie în modul prezentat din figura 2.2.

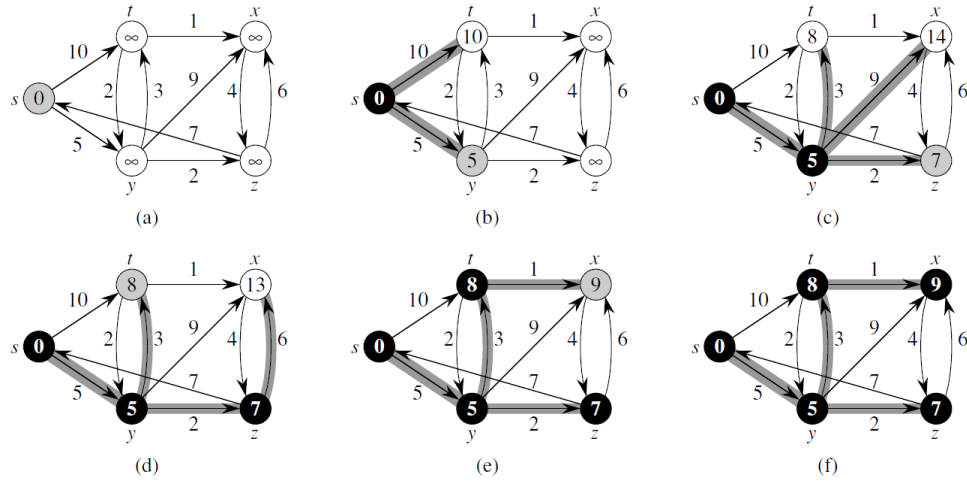


Figure 2.3: [2] Algoritmul Dijkstra pe etape. Vârful sursă este 0. Muchiile hașurate reprezintă valorile predecesorilor: dacă (u, v) este hașurat atunci $\omega[v] = u$. Vârfurile marcate cu negru sunt din S iar cele marcate cu alb aparțin cozii $Q = V - S$. (a) Configurația există înaintea primei iterații a repetiției **while**. Vârful hașurat este u din linia 5 și are valoarea minimă. (b)-(f) Configurația după fiecare iterație **while**.

2.3.2 Implementare

Algoritmul Dijkstra efectuează n iterații. La fiecare iterație, se selectează un vârf nemarcat v cu cea mai mică valoare $d[v]$, îl marchează și verifică toate marginile (v, i) încercând să îmbunătățească valoarea $d[i]$. Fie un graf ponderat orientat sau neorientat cu n noduri și m muchii.

¹Listă de adiacență care conține toate nodurile v pentru care există o muchie $(u, v) \in E$.

Durata de rulare a acestui algoritm constă în :

- n căutări a unui nod cu cea mai mică valoare $d[v]$ printre nodurile nemarcate.
- m încercări de relaxări.

Pentru cea mai simplă implementare a acestor operații pe fiecare vârf de iterație, căutarea necesită $O(n)$ operații și fiecare relaxare poate fi efectuată în $O(1)$. Prin urmare, comportamentul asimptotic rezultat al algoritmului este:

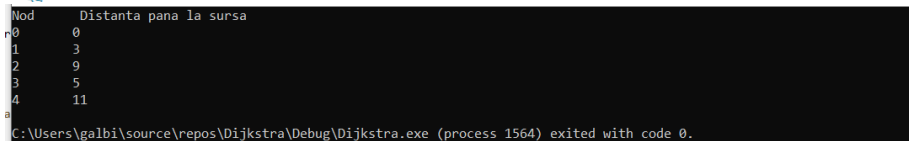
$$O(n^2)^2$$

Această complexitate este optimă pentru un graf ponderat, atunci când $m \approx n^2$. Cu toate acestea, în grafurile rare, când m este mult mai mic decât numărul maxim de muchii n^2 , problema poate fi rezolvată în complexitate $O(n \log(n) + m)$.

```

1  #include<iostream>
2  #define Varfuri 5
3  using namespace std;
4  int distantaMinima(int distanta[], bool inclus[]) {
5      int min = INT_MAX, min_index;
6      for (int v = 0; v < Varfuri; v++)
7          if (inclus[v] == false && distanta[v] <= min) min = distanta[v], min_index = v;
8      return min_index;
9  }
10 void AfiseazaSolutie(int distanta[]) {
11     cout<<("Nod \t Distanța pana la sursa\n");
12     for (int i = 0; i < Varfuri; i++) cout << i << "\t" << distanta[i] << endl;
13 }
14 void Dijkstra(int graf[Varfuri][Varfuri], int sursa) {
15     int distanta[Varfuri];
16     bool inclus[Varfuri];
17     for (int i = 0; i < Varfuri; i++) { distanta[i] = INT_MAX;
18                                     inclus[i] = false;
19     }
20     distanta[sursa] = 0;
21     for (int count = 0; count < Varfuri - 1; count++) {
22         int u = distantaMinima(distanta, inclus);
23         inclus[u] = true;
24         for (int v = 0; v < Varfuri; v++)
25             if (graf[u][v] && distanta[u] != INT_MAX && distanta[u] + graf[u][v] < distanta[v])
26                 distanta[v] = distanta[u] + graf[u][v];
27     }
28     AfiseazaSolutie(distanta);
29 }
30 int main() {
31     int graf[Varfuri][Varfuri] = { {0,3,0,5,0},
32                                     {0,0,6,2,0},
33                                     {0,0,0,0,2},
34                                     {0,2,4,0,6},
35                                     {3,0,7,0,0}, };
36     Dijkstra(graf,0);
37     return 0;
38 }

```



```

Nod      Distanța pana la sursa
0        0
1        3
2        9
3        5
4        11

```

C:\Users\galbi\source\repos\Dijkstra\Debug\Dijkstra.exe (process 1564) exited with code 0.

Figure 2.4: Algoritmul afișează toate costurile drumurilor de la i la sursă.

² Alte rezultate cu privire la costul diverselor implemetări a algoritmului lui Dijkstra, [4]

Capitolul 3

Drumuri minime între toate perechile de vârfuri

În acest capitol vom pune problema studiului determinării drumurilor de lungime minimă între toate perechile de vârfuri ale unui graf G . Problema poate fi dacă dorim să construim un tabel al distanțelor între toate perechile de magazine. Ipotezele sunt aceleași ca în capitolul anterior, avem un graf orientat $G = (V, E)$, cu costuri și o funcție de costuri $f : E \rightarrow \mathbb{R}$ aplicată arcelor grafului. Dorim să determinăm, pentru fiecare $u, v \in V$, un **drum de cost minim** de la u la v , unde acest rezultat este suma costurilor acelor arce care formează acest drum. Rezultatul obținut este de preferat să fie sub forma unui tabel: linia u , coloana v și conținutul drumului minim de la u la v .

Majoritatea algoritmilor din cadrul acestui capitol vor avea reprezentarea prin matrici de adiacență. Input-ul este o matrice A , având dimensiunea $n \times n$, reprezentând costurile arcelor unui graf $G = (V, E)$ orientat cu n noduri. Mai pe scurt $A = (a_{ij})$ unde

$$a_{ij} = \begin{cases} 0, & \text{dacă } i = j, \\ f(i, j), & \text{dacă } i \neq j \text{ și } (i, j) \in E, \\ \infty, & \text{dacă } i \neq j \text{ și } (i, j) \notin E. \end{cases}$$

Output-ul este o matrice $D = (d_{ij})$ de dimensiune $n \times n$ al căror elemente reprezintă costul minim de la i la j . Notând cu $\delta(i, j)$ costul minim drumului de la i la j , vom avea $d_{ij} = \delta(i, j)$.

Pentru rezolvarea problemei, trebuie să calculăm costurile drumurilor minime și **matricea predecesorilor** pe care o notăm cu $P = (\pi_{ij})$, unde π_{ij} este NIL pentru $i = j$ sau dacă nu este un drum de la i la j . Altfel, elementul π_{ij} este predecesorul lui j având un drum minim de la i . Pentru orice vârf $i \in V$, fixăm **subgraful predecesorilor** lui G pentru i ca $G_{\pi, i} = (V_{\pi, i}, E_{\pi, i})$,

unde

$$V_{\pi,i} = \{j \in V : \pi_{ij} \neq NIL\} \cup \{i\}$$

și

$$E_{\pi,i} = \{(\pi_{ij}, j) : j \in V_{\pi,i} \text{ și } \pi_{ij} \neq NIL\}.$$

Dacă $G_{\pi,i}$ îndeplinește condiția de a fi un arbore de drum minim, atunci are loc următoarea procedură, aceea de a aplica metoda AFIȘEAZĂ-DRUM, care afișează drumul minim de la i la j .

AFIȘEAZĂ-DRUM (P, i, j)

```

1: if  $i = j$ 
2:   afișează  $i$ 
3: else
4:   if  $\pi_{ij} = NIL$ 
5:     afișează "Nu este drum de la  $i$  la  $j$ "
6:   else
7:     AFIȘEAZĂ-DRUM ( $P, i, \pi_{ij}$ )
8:   afișează  $j$ 

```

3.1 Drumuri minime

Studiem mai întâi **structura unui drum minim** pentru caracterizarea unei soluții optime. Presupunem că graful $G = (V, E)$ este reprezentat printr-o matrice de adiacență $A = (a_{ij})$. Considerăm un drum p de lungime minimă de la vârful i la j și m numărul de arce conținute în p . Dacă $i = j$ atunci costul lui p este 0. Dacă $i \neq j$ atunci putem descompune drumul p în $i \xrightarrow{p'} k \rightarrow j$ unde p' conține $m - 1$ arce. În final avem următoarea egalitate

$$\delta(i, j) = \delta(i, k) + f(k, j).$$

Definim $d_{ij}^{(m)}$ ca fiind costul minim a unui drum de la i la j care este alcătuit din cel mult m arce.

$$d_{ij}^{(0)} = \begin{cases} 0, & \text{dacă } i = j, \\ \infty, & \text{dacă } i \neq j. \end{cases}$$

Pentru $m \geq 1$, determinăm $d_{ij}^{(m)}$ ca minimumul între $d_{ij}^{(m-1)}$ și costul minim al fiecărui drum de la i la j cu cel mult m arce, luând în considerare toți predecesorii k ai lui j .

$$d_{ij}^{(m)} = \min \left(d_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \left\{ d_{ik}^{(m-1)} + a_{kj} \right\} \right) = \min_{1 \leq k \leq n} \left\{ d_{ik}^{(m-1)} + a_{kj} \right\} \quad (3.1)$$

iar costurile $\delta(i, j)$ ale drumurilor minime sunt date de

$$\delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \dots$$

Dacă graful G nu are nici un ciclu de cost negativ, putem deduce că toate drumurile minime sunt elementare și conțin măcar, $n - 1$ arce. Un drum de la nodul i la j cu mai mult de $n - 1$ arce nu poate avea un cost mai mic decât cel mai scurt drum de la nodul i la j .

Avem următoarea problemă, dorim să determinăm în mod ascendent costurile drumurilor minime. Considerăm ca input o matrice $A = (a_{ij})$ și vom determina o listă de matrici $D^{(1)}, D^{(2)}, \dots, D^{(n-1)}$, unde pentru orice $m = 1, 2, \dots, n - 1$ avem $D^{(m)} = (d_{ij}^{(m)})$. Matricea $D^{(n-1)}$ va conține costurile drumurilor minime. O observație importantă ar fi că dacă $d_{i,j}^{(1)} = a_{ij}$ pentru orice $i, j \in V$, obținem $D^{(1)} = A$. Cu alte cuvinte, dându-se matricele $D^{(m-1)}$ și A , se va obține matricea $D^{(m)}$ care reprezintă extinderea drumurilor minime cu încă un arc.

EXTINDE(D, A)

```

1:  $n \leftarrow \text{linii}[D]$ 
2: fie  $B = (b_{ij})$  matrice cu dimensiunea  $n \times n$ 
3: for  $i \leftarrow 1, n$ 
4:   for  $j \leftarrow 1, n$ 
5:      $b_{ij} \leftarrow \infty$ 
6:   for  $k \leftarrow 1, n$ 
7:      $b_{ij} \leftarrow \min(b_{ij}, d_{ik} + a_{kj})$ 
8: return B

```

Timpul de execuție al acestei funcții este de $O(n^3)$ datorită celor 3 bucle pe care le conține. Funcția returnează matricea $B = (b_{ij})$, acest lucru realizându-se cu ajutorul ecuației (3.1) pentru orice i, j utilizând D pentru $D^{(m-1)}$ și B pentru $D^{(m)}$.

Acum, după toată această discuție, putem observa legătura cu înmulțirea matricilor. Dorim să calculăm produsul dintre două matrici A și B de dimensiune $n \times n$, $C = A * B$. Vom calcula pentru orice $i, j = 1, 2, \dots, n$

$$^1 c_{ij} = \sum_{k=1}^n a_{ik} * b_{kj}. \quad (3.2)$$

ÎNMULȚEȘTE-MATRICILE(A, B)

```

1:  $n \leftarrow \text{linii}[A]$ 
2: fie  $C = (c_{ij})$  matrice cu dimensiunea  $n \times n$ 
3: for  $i \leftarrow 1, n$ 

```

¹Se poate arăta că pornind de la (3.2) se ajunge la (3.1). [2]

```

4:   for  $j \leftarrow 1, n$ 
5:        $c_{ij} \leftarrow 0$ 
6:       for  $k \leftarrow 1, n$ 
7:            $c_{ij} \leftarrow c_{ij} + a_{ik} * b_{kj}$ 
8: return C

```

Întorcându-ne la problema propriu zisă, determinăm costul drumurilor minime extinzând arc cu arc. Notând cu $A * B$ matricea returnată de EXTINDE(A,B), determinăm șirul de $n - 1$ matrice

$$\begin{aligned}
 D^{(1)} &= D^{(0)} * A = A, \\
 D^{(2)} &= D^{(1)} * A = A^2, \\
 &\vdots \\
 D^{(n-1)} &= D^{(n-2)} * A = A^{n-1}.
 \end{aligned}$$

Acum că am determinat șirul de $n - 1$ matrice, putem transpune tot ce am scris mai sus într-o funcție.

```

DRUMURI-MINIME(A)
1:  $n \leftarrow \text{linii}[A]$ 
2:  $D^{(1)} \leftarrow A$ 
3: for  $i \leftarrow 2, n - 1$ 
4:      $D^{(i)} \leftarrow \text{EXTINDE}(D^{(i-1)}, A)$ 
5: return  $D^{(n-1)}$ 

```

Acestă funcție este o versiune mai lentă, timpul de execuție al determinării acestui șir fiind de $O(n^4)$.

3.2 Algoritmul Floyd-Warshall

În informatică, algoritmul Floyd-Warshall (cunoscut și sub numele de algoritmul lui Floyd, algoritmul Roy-Warshall, algoritmul Roy-Floyd) este un algoritm pentru găsirea drumurilor celor mai scurte într-un graf ponderat cu cost pozitiv sau negativ. O singură execuție a algoritmului va găsi lungimile ale celor mai scurte drumuri între toate perechile de vârfuri. Deși acest algoritm nu întoarce detalii ale drumurilor în sine, este posibilă reconstrucția drumurilor cu modificări simple ale algoritmului.

3.2.1 Descrierea algoritmului

Ideea principală a acestui algoritm este de a partiționa procesul de găsire a celui mai scurt drum între oricare două noduri, la mai multe faze. Astfel, algoritmul ia în considerare nodurile "intermediare" ale unui drum minim,

unde un *nod intermediar* al unui drum elementar $p = [v_1, v_2, \dots, v_n]$ este orice nod din mulțimea $\{v_2, v_3, \dots, v_{n-1}\}$.

Fie $V = \{1, 2, \dots, n\}$ mulțimea nodurilor lui G și matricea $D = (d_{ij})$ ale căror elemente reprezintă distanța de la i la j , pentru orice $i, j = \overline{1, 2, \dots, n}$. Înainte de pasul k ($k = 1, \dots, n$), $d[i][j]$ stochează lungimea celui mai scurt drum de la nodul i la j care conține doar vârfurile $\{1, 2, \dots, k\}$ ca vârfuri interne. Pentru orice pereche $(i, j) \in V \times V$, considerăm toate drumurile de la i la j ale căror noduri intermediare fac parte din mulțimea $\{1, 2, \dots, k\}$. Fie p drumul de cost minim dintre aceste drumuri.

Este ușor de arătat că proprietatea are loc pentru primul pas. Pentru $k = 0$, putem încărcă matricea cu $d[i][j] = f(i, j)$ dacă există muchie de la i la j cu costul $f(i, j)$ și $d[i][j] = \infty$ dacă nu există nici o muchie. În principiu, în aplicații, aproximăm ∞ cu un număr foarte mare.

Să presupunem că ne aflăm la pasul k , și vrem să construim matricea $d[\][\]$ astfel încât să îndeplinească cerințele pentru pasul $(k + 1)$. Trebuie să remediem distanțele pentru unele perechi de noduri (i, j) . [2] Există două cazuri fundamentale care depind de statutul lui k :

- Dacă nodul k nu este nod intermediar al drumului p , atunci cel mai scurt drum de la nodul i la j cu nodurile interne $\{1, 2, \dots, k\}$ coincide cu cel mai scurt drum cu nodurile interne din $\{1, 2, \dots, k - 1\}$. În acest caz, $d[i][j]$ va rămâne neschimbat în timpul tranziției.
- Dacă nodul k este nod intermediar al drumului p , atunci drumul poate fi descompus în două drumuri, fiecare folosind nodurile din $\{1, 2, \dots, k - 1\}$ pentru a compune un drum care folosește toate nodurile din $\{1, 2, \dots, k\}$. Asta înseamnă că putem împărți drumul de la i la j în două drumuri: un drum de la i la k iar cel de-al doilea de la k la j . Prin urmare am calculat deja lungimile acestor drumuri înainte și putem calcula lungimea celui mai scurt drum de la i la j ca fiind $d[i][k] + d[k][j]$.

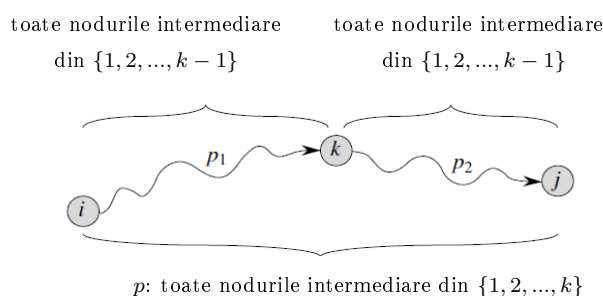


Figure 3.1: Drumul p este unul de lungime minimă de la nodul i la j iar k este nodul intermediar al lui p cu numărul cel mai mare. Drumul p_1 reprezintă porțiunea din drumul p de la i la k cu toate nodurile intermediare $\{1, 2, \dots, k - 1\}$. Aceeași procedură este valabilă și pentru drumul p_2 .

Combinând aceste două cazuri, aflăm că putem recalcula lungimea tuturor perechilor (i, j) la pasul k în felul următor:

$$d_{nou}[i][j] = \min(d[i][j], d[i][k] + d[k][j])$$

Astfel, tot ce este necesar la pasul k este de a itera peste toate perechile de vârfuri și de a recalcula lungimea celui mai scurt drum dintre ele. Prin urmare, după pasul n , valoarea $d[i][j]$ din matricea cost D este lungimea celui mai scurt drum dintre i și j sau ∞ dacă nu există drum de la i la j .

O ultimă remarcă ar fi că nu ar trebui să creăm o altă matrice D_{nou} pentru stocarea temporală a celui mai scurt drum la pasul k , toate modificările pot avea loc în matricea D la orice pas.

3.2.2 Implementare

Fie $D \in M_{n \times n}$ matricea cost care este construită la pasul $k = 0$ cum am menționat mai sus. Deasemenea vom fixa elementele de pe diagonala principală $d[i][i] = 0$ pentru orice $i \in V$ la pasul 0.

Algoritmul este implementat astfel:

```

1  #include <iostream>
2  using namespace std;
3  #define Nod 6
4  #define INF 9999999
5  void AfiseazaSolutie(int distanta[][Nod]);
6  void FloydMarshall(int graf[][Nod]){
7      int distanta[Nod][Nod], i, j, k;
8      for (i = 0; i < Nod; i++)
9          for (j = 0; j < Nod; j++) distanta[i][j] = graf[i][j];
10     for (k = 0; k < Nod; k++)
11         for (i = 0; i < Nod; i++)
12             for (j = 0; j < Nod; j++)
13                 if (distanta[i][k] + distanta[k][j] < distanta[i][j]) distanta[i][j] = distanta[i][k] + distanta[k][j];
14     AfiseazaSolutie(distanta);
15 }
16
17 void AfiseazaSolutie(int distanta[][Nod]){
18     cout << "Matricea urmatoare arata cele mai scurte drumuri intre toate perechile de noduri (i,j)\n";
19     for (int i = 0; i < Nod; i++) {
20         for (int j = 0; j < Nod; j++) {
21             if (distanta[i][j] == INF) cout << "INF" << " ";
22             else cout << distanta[i][j] << " ";
23         }
24         cout << endl;
25     }
26 }
27
28 int main(){
29     int graph[Nod][Nod] = { {0,    5, INF, 10, 3, 2},
30                             {INF,  0,  3, INF, INF, INF},
31                             {INF, INF,  0,  1, 3, 10},
32                             {INF, INF, INF,  0, INF, 4},
33                             {INF, INF,  1, INF, 0, 4},
34                             {INF,  2, 10, 11, INF, 0} };
35     FloydMarshall(graph);
36     return 0;
37 }

```

```

Matricea urmatoare arata cele mai scurte drumuri intre toate perechile de noduri (i,j)
0  4  4  5  3  2
INF 0  3  4  6  8
INF 7  0  1  3  5
INF 6  9  0 12  4
INF 6  1  2  0  4
INF 2  5  6  8  0

```

C:\Users\galbi\source\repos\Roy-Floyd\Debug\Roy-Floyd.exe (process 880) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

Figure 3.2: Algoritmul afișează toate drumurile de cost minim de la nodul i la j în k pași.

Capitolul 4

Flux maxim

Problema fluxului maxim, la fel ca alte probleme formulate pe grafuri, își are originea în economia modernă, mai precis în optimizarea economică. Cele mai recente aplicații sunt în domeniul rețelelor și fluxurilor informaționale. Dacă este pusă problema cercetării unei rețele de transport a unui material dintr-un punct din care acesta este fabricat, acest punct îl numim sursă, într-un punct de depozitare pe care îl numim stoc sau destinație folosind canale de transport cu anumite capacități, iminent se ivește problema determinării capacității de transportare a materialului din sursă către stoc raportându-se la toată rețeaua. Cu alte cuvinte problema fluxului este aceea de a determina cantitatea cea mai mare de material care poate fi transportată pornind de la sursă și ajungând la destinație ținând cont de restricțiile de capacitate. Canalele de transport și materialele pot avea diverse forme precum: pachete de date, piese și transportatoare, conducte, cisterne petroliere, etc.

În ceea ce urmează vom defini cu ajutorul grafurilor, rețelele de transport, vom discuta anumite proprietăți, vom defini problema fluxului maxim și vom introduce câteva notații utile.

4.1 Fluxuri și rețele de transport

O **rețea de transport** este în principiu un graf orientat $G = (V, E)$ în care fiecărei muchii $(u, v) \in E$ cu $u, v \in V$ îi este atașată o **capacitate** nenegativă $c(u, v) \geq 0$. Vom denumi două vârfuri din rețea: unul **sursă** s și celălalt **destinație** d . Dacă $(u, v) \notin E$ atunci considerăm că $c(u, v) = 0$. Presupunem că pentru orice nod $v \in V$ există un drum $s \rightarrow v \rightarrow d$. Graful fiind conex avem următoarea relație $|E| \geq |V| - 1$.

Fie o rețea de transport $G = (V, E)$ cu o funcție de capacitate c . Fixăm nodul sursă s și nodul destinație d . Denumim **fluxul** G ca fiind o funcție $f : V \times V \rightarrow \mathbb{R}$ care satisface următoarele condiții:

1. **Restricția de capacitate:** Pentru orice $u, v \in V$, $f(u, v) \leq c(u, v)$.

2. **Antisimetrie:** Pentru orice $u, v \in V$, $f(u, v) = -f(v, u)$.
3. **Conservarea fluxului:** Pentru orice $u \in V \setminus \{s, d\}$ avem

$$\sum_{v \in V} f(u, v) = 0.$$

Cantitatea $f(u, v)$ care poate fi negativă sau pozitivă se numește **fluxul** pe arcul (u, v) . Totuși un flux negativ de la u la v reprezintă unul virtual, acesta nu reprezintă un transport profitabil, ci doar sugerează că există un transport fizic de la u la v . Definim valoarea fluxului ca fiind :

$$|f| = \sum_{v \in V} f(s, v),$$

mai precis fluxul total care pleacă din sursă. O prima observație ar fi că notația pe care am definit-o mai sus $|\cdot|$ nu înseamnă valoarea absolută sau cardinalul unei mulțimi, ci **valoarea fluxului**.

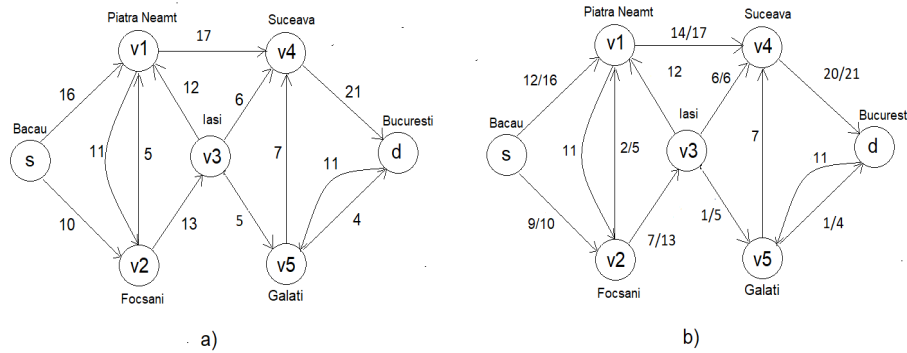


Figure 4.1: **a)** O rețea G pentru problema de transport de persoane a firmei BRONSON. Plecarea microbuzului din orașul Bacău reprezintă sursa s iar destinația este dată de nodul d . Persoanele sunt transportate prin localități intermediare, dar numai $c(u, v)$ persoane pot fi luate din orașul u în orașul v . Toate arcurile au o anumită capacitate. **b)** Valoarea fluxului $|f| = 21$. Pe arce sunt marcate numai fluxurile de rețea pozitivă. Dacă $f(u, v) > 0$ rezultă că arcul (u, v) se poate marca astfel $f(u, v)/c(u, v)$ (scrierea pe care am folosit-o este doar de notație, aceea de a separa cele două valori: fluxul și capacitatea). Dacă $f(u, v) \leq 0$ atunci (u, v) este marcat numai cu capacitatea.

O primă observație ar fi că fluxul între oricare două vârfuri care nu sunt legate prin nici un arc nu poate fi decât 0. Astfel, pentru $(u, v) \notin E$ și $(v, u) \notin E$ avem $c(u, v) = c(v, u) = 0$. Impunându-se restricția de capacitate avem că $f(u, v) \leq 0$ și $f(v, u) \leq 0$. Folosind antisimetria $f(u, v) = -f(v, u)$

rezultă că $f(u, v) = f(v, u) = 0$. Prin urmare, existența fluxului nenul între u și v implică $(v, u) \in E$ sau $(u, v) \in E$ sau ambele.

Acum apare următoarea problemă. Fiind dat G o rețea de transport cu sursa s și destinația d , problema ne cere să aflăm găsirea unui flux de valoare maximă de la s la d . Cu alte cuvinte, **problema fluxului maxim** impune găsirea unui flux maxim de la s la d .

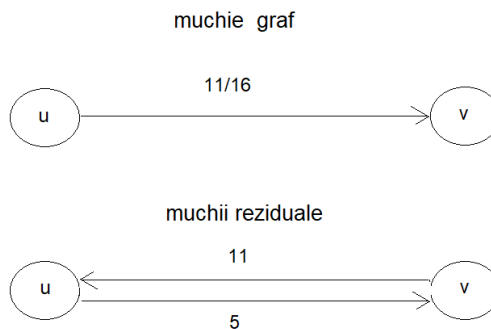
4.2 Metoda lui Ford-Fulkerson

Este denumită "metodă" deoarece cuprinde mai multe implementări, fiecare dintre acestea având timpul ei de execuție. Metoda lui Ford-Fulkerson se bazează pe trei idei principale: **rețele reziduale**, **drumuri de ameliorare** și **tăieturi**.

Metoda lui Ford-Fulkerson este una iterativă. Pentru orice $u, v \in V$ vom avea fluxul $f(u, v) = 0$. La fiecare iterație vom mări fluxul prin găsirea unui "drum de ameliorare". Se va repeta acest procedeu până nu se va mai găsi nici un drum de ameliorare.

Fie $G = (V, E)$ o rețea de transport având o sursă s și destinația d . Considerăm un flux f în G și o pereche de vârfuri $u, v \in V$. Denumim **capacitatea reziduală** a arcului (u, v) ca fiind cantitatea de flux adițională care poate fi transportată de la u la v , fără a depăși capacitatea $c(u, v)$.

$$c_f(u, v) = c(u, v) - f(u, v)$$



Dacă avem arcul $(u, v) \in V$ cu $f(u, v) = 11$ și $c(u, v) = 16$ atunci se pot transporta $c_f(u, v) = 5$ unități suplimentare. Deși arcul $(v, u) \notin V$, aplicând antisimetria, vom putea avea totuși o capacitate reziduală $c_f(v, u) = c(v, u) - f(v, u) = 0 - (-11) = 11$. Astfel putem transporta 11 unități în sens opus care să le anuleze pe cele 11 ale fluxului $f(u, v)$.

Fiind dat G și un flux f , numim **rețeaua reziduală** a lui G de f ca fiind $G_f = (V, E_f)$, unde

$$E_f = \{(u, v) \in V \times V \mid c_f(u, v) = c(u, v) - f(u, v) > 0\}$$

Denumim **drum de ameliorare** p ca fiind un drum simplu de la s la t în G_f . Mai general, drumul de ameliorare este un drum $[u_1, u_2, \dots, u_k]$, unde $u_1 = s$ și $u_k = t$, în graful rezidual cu $c_f(u_i, u_{i+1}) > 0$ pentru orice $i = 1, 2, \dots, k-1$. Spunem că **capacitatea reziduală** a lui p reprezintă cantitatea maximă a fluxului f care poate fi transportată de-a lungul drumului p , cu următoarea formulă

$$c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$$

4.2.1 Implementare

La fiecare iterație căutăm un drum aleator de ameliorare p la care mărim fluxul f de-a lungul lui p cu capacitatea reziduală $c_f(p)$. Metoda următoare calculează fluxul maxim a grafului $G = (V, E)$, actualizând fluxul $f(u, v)$. Dacă u și v nu formează un arc atunci vom presupune că $f(u, v) = 0$. Presupunem că între vârfurile u și v valoarea capacității este dată de funcția $c(u, v)$, calculabilă în timp constant și că $c(u, v) = 0$ dacă $(u, v) \notin E$.

METODA-FORD-FULKERSON(s, d, G)

- 1: **for** fiecare arc $(u, v) \in E[G]$
- 2: $f(u, v) \leftarrow 0$
- 3: $f(v, u) \leftarrow 0$
- 4: **while** există un drum de la s la d în rețeaua reziduală G_f
- 5: $c_f(p) \leftarrow \min\{c_f(u, v) \mid (u, v) \in p\}$
- 6: **for** fiecare (u, v) din p
- 7: $f(u, v) \leftarrow f(u, v) + c_f(p)$
- 8: $f(u, v) \leftarrow -f(u, v)$

Timpul de execuție al acestui algoritm depinde de felul cum se calculează drumul de ameliorare p pe linia 4 a algoritmului. De cele mai multe ori, problemele de flux maxim apar cu capacități care sunt numere întregi. În cazul în care capacitățile sunt numere raționale, le putem înlocui cu numere întregi făcând o operație de scalare corespunzătoare.

O implementare directă al acestui algoritm durează $O(E|f_m|)$, unde f_m este fluxul maxim găsit de algoritm. Motivul pentru acest rezultat ar fi că bucla **while** se execută de cel mult $|f_m|$ ori deoarece valoarea fluxului crește cu cel puțin o unitate de fiecare dată.

Complexitatea algoritmului poate fi îmbunătățită dacă p de pe linia 4 se calculează folosind o căutare în lățime, care acest lucru garantează că va fi calea *cea mai scurtă* de la s la t în rețeaua reziduală, unde fiecarei muchii îi corespunde o distanță unitară (greutate). Vom numi această implementare a metodei ca fiind **algoritmul lui Edmons-Karp**.

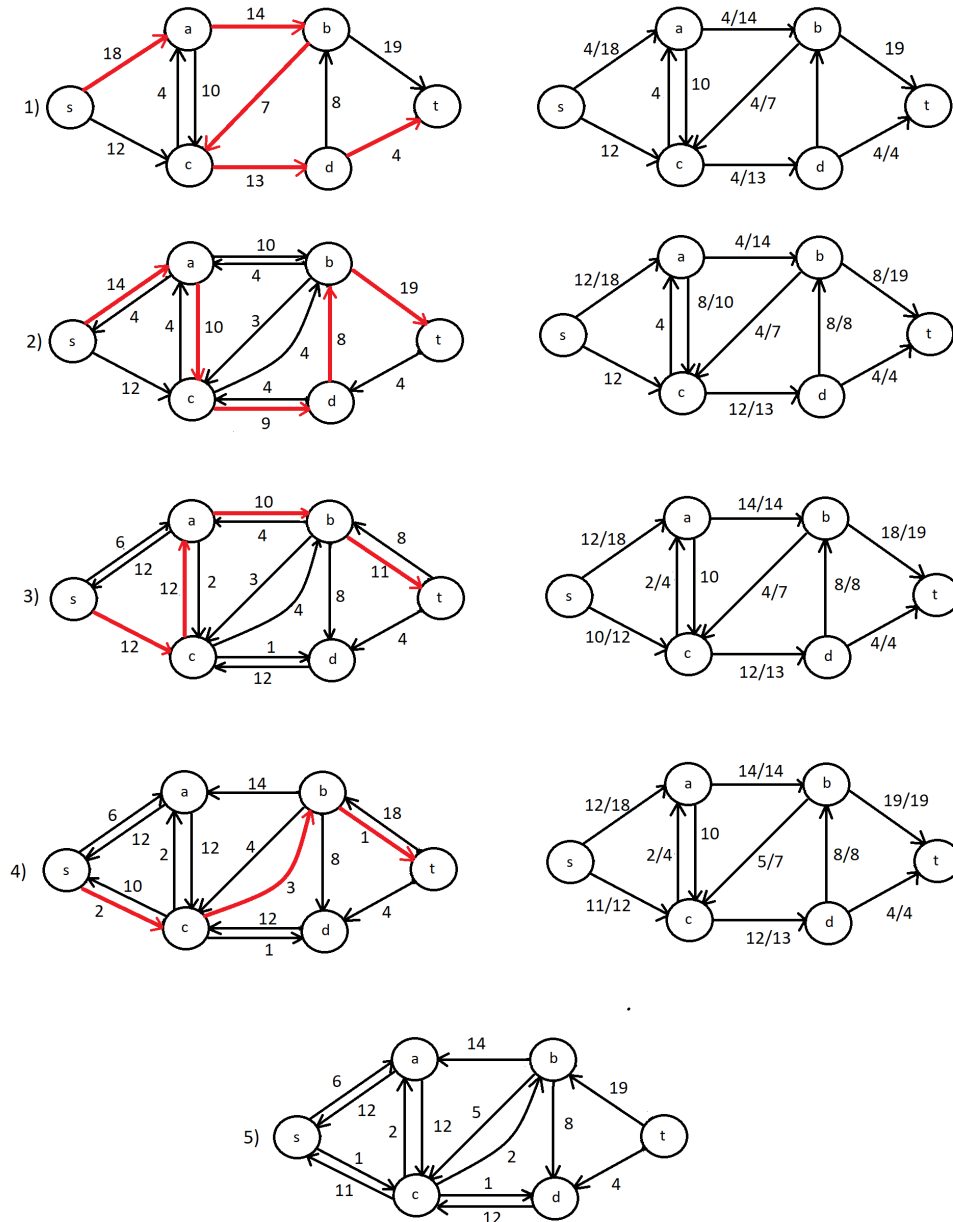


Figure 4.2: Exemplificarea algoritmului al lui For-Fulkerson. Rețeaua reziduală (1) este rețeaua de intrare $G(1)$ -(4) Iterațiile succesive ale instrucțiunii repetitive **while**. În partea stângă se găsesc rețelele reziduale G_f ; drumurile de ameliorare p sunt trasate cu roșu iar în partea dreaptă se găsesc noile fluxuri f . (5) Rețeaua reziduală la ultimul test, aceasta nu mai conține nici un drum de ameliorare, așadar fluxul f de la (4) este unul maxim.

Capitolul 5

Aplicație

GPS-ul este un sistem de navigație sub forma unui dispozitiv compact atașat unei mașini, avion sau a unei nave. GPS-ul primește informații de navigare în timp real sub formă de coordonate din sateliți. În mașină, GPS-ul îl ajută pe șofer să urmeze cea mai scurtă cale de la sursă la destinație. GPS-ul de astăzi are caracteristici suplimentare cum ar fi: furnizarea de căi alternative la cea mai scurtă cale pentru a evita traficul sau construcția drumurilor pe această cale. Această informație este importantă deoarece cel mai scurt drum nu garantează întotdeauna sosirea într-un timp minim. Prin urmare, una sau două drumuri alternative sunt disponibile cu ușurință în dispozitiv.

Astfel, un cetățean străin care nu cunoaște harta orașului, dorește să ajungă în timp util la o anumită locație, folosind transportul public. Timpul alocat pentru a ajunge la destinație este de preferat să fie minim. Prin urmare, cu ajutorul unei aplicații de navigare, acesta poate afla ce autobuze parcurg drumul dorit într-un interval minim de timp. Cetățeanul poate selecta locația curentă și destinația pentru a afla rutele autobuzelor. Făcând acest lucru, aplicația generează 3 drumuri optime și autobuzele care se deplasează pe aceste drumuri. Dacă acesta află că pe drumul optim generat de aplicație sunt probleme ce țin de drum sau alte diverse motive care împiedică parcurgerea drumului, acesta are la dispoziție alte două drumuri alternative pentru a ajunge la destinație într-un timp minim.

Implementare

Aplicația GPS are la bază algoritmului lui Dijkstra. Proiectul ilustrează un model GPS simplu pentru afișarea a trei rute diferite între sursă și destinație. Proiectul produce versiunea offline a GPS-ului în care nu există date în timp real cu privire la coordonatele actuale ale șoferului.

Figura de mai jos (Figure 5.1) afișează un simplu output pentru GPS. Rezultatul constă în afișarea unui graf cu noduri și muchii generate aleator și o fereastră de vizualizare în care se găsesc informații de rutare. Programul

afișează deasemenea și alte informații despre autobuze cum ar fi: costul total, numărul de stații precum și drumul.

Se crează mai întâi o copie a grafului G pentru a se putea realiza operațiunile de reducere a grafului G . Primul drum este obținut din graful original G cu 30 de noduri. Odată ce drumul a fost obținut, muchiile de-a lungul drumului sunt îndepărtate pentru a reduce G la G' . Algoritmul lui Dijkstra este aplicat din nou pentru G' pentru a produce cel de-al doilea drum, care are un set de muchii diferit față de primul. Respectând algoritmul, muchiile celui de-al doilea autobuz sunt îndepărtate pentru a reduce G' la G'' . Analog se aplică același procedeu și pentru cel de-al treilea drum. După ce s-au generat toate cele 3 drumuri, copia grafului se va înlocui cu originalul pentru a nu rămâne eliminate muchiile de mai sus.

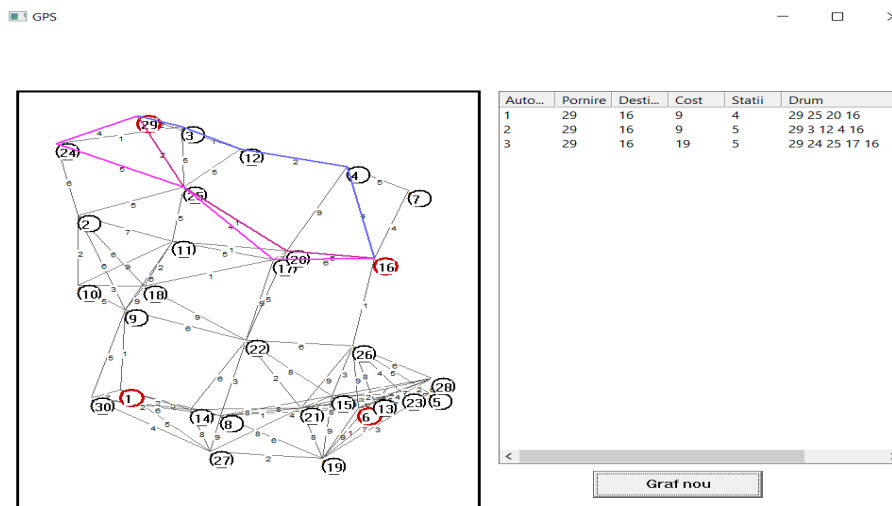


Figure 5.1: Output-ul codului GPS care afișează ruta a trei autobuze de la sursă la destinație.

Sistemul GPS pornește de la constructorul $GPS()$, care creează o ferestă și butonul *Graf nou*. Această funcție apelează deasemenea funcțiile *AfisareSetup()* și *ResetareGraf()* care stabilesc variabilele de afișare comune și creează graful. *ResetareGraf()* apelează *Initializare()* care creează graful prin alocarea coordonatelor aleatorii la noduri și costurile aleatorii la muchiile grafului. Graful inițial este afișat și reactualizat prin *OnPaint()*.

Evenimentul *OnLButtonDown()* detectează click-ul din stânga mouse-ului a cărui poziție se află în Windows returnat de obiectul *CPoint punct*. Valoarea obiectului este verificată cu $v[i].rect$ folosind *PtInRect()* (această funcție determină dacă punctul specificat se află sau nu în dreptunghi). Nodul sursă este identificat prin $bFlag=1$ iar destinația prin $bFlag=2$.

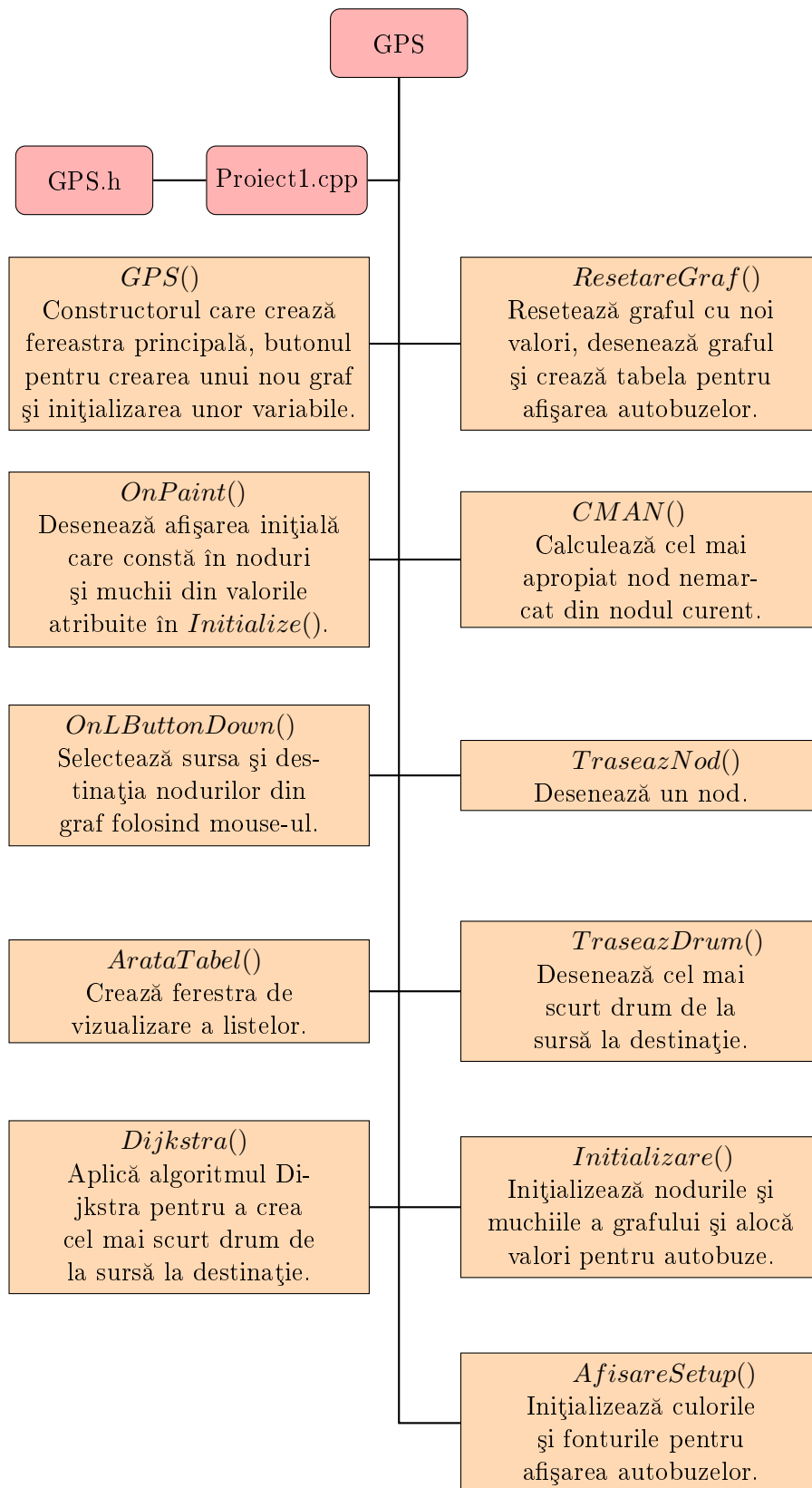
Dijkstra() calculează cel mai scurt drum folosind algoritmul lui Dijkstra. Funcția este apelată atunci când nodul secund a fost selectat în timp ce au-

tobuzul de la sursă la nodul destinație este desenat folosind *TraseazăDrum()*. Odată ce primul autobuz a fost finalizat, informațiile sale sunt actualizate și afișate în fereastra de vizualizare. Deasemenea, muchiile de la primul autobuz sunt șterse din graful original. Ștergerea muchiilor este îndeplinită de *Initialize()* înlocuind valoarea muchiei cu 99.

Cel de-al doilea autobuz este o repetare a primului autobuz prin referire la graful redus G' . Prin urmare, calculul pentru noul drum dintre cele două noduri, va lua în considerare faptul că primul drum are noduri care nu sunt adiacente dealungul parcurgerii. Programul calculează noul drum care în mod cert evită primul drum. Similar, se aplică aceeași metodă și pentru drumul cu numărul 3.

Variabile și obiecte importante din clasa *GPS* și descrierea lor:

GPS		
Variabile	Tipul	Descriere
<i>bgGraf</i>	<i>CButton</i>	Buton pentru generarea unui nou graf
<i>Autobuz[i].drum[k]</i>	<i>int</i>	Drumul k în autobuzul i
<i>Autobuz[i].nNod</i>	<i>int</i>	Numărul de noduri prin care autobuzul i a trecut
<i>Autobuz[i].cTotal</i>	<i>int</i>	Costul total al autobuzului i
<i>home</i>	<i>CPonit</i>	Colțul din stânga sus al zonei grilei dreptunghiului
<i>v[i].cost[j]</i>	<i>int</i>	Costul dintre (v_i, v_j)
<i>v[i].sp[j]</i>	<i>int</i>	Drumul cel mai scurt dintre (v_i, v_j)
<i>v[i].rct</i>	<i>CRect</i>	Definește un dreptunghi prin coordonatele colțurilor din stânga sus și dreapta jos
<i>NodPrecedent</i>	<i>int</i>	Nodul precedent al nodului curent
<i>Sursa, Destinatia</i>	<i>int</i>	Sursa și destinația
<i>nAutobuz</i>	<i>int</i>	Numarul de autobuze de succes
<i>tabel</i>	<i>CListCtrl</i>	Tabela care afișează autobuzele de succes
<i>pAutobuz[i]</i>	<i>CPen</i>	Culoarea autobuzului i
<i>Noduri</i>	<i>constant</i>	Numărul de noduri în graf
<i>LinkRange</i>	<i>constant</i>	Valoarea pragului intervalului pentru adiacența dintre două noduri din grafic
<i>stareVar f[i]</i>	<i>bool</i>	Starea vârfului v_i



Bibliografie

- [1] A. Bondy, U.S.R. Murty, *Graph theory*, Springer (2008), Graduate texts in mathematics 244.
- [2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introducere în Algoritmi*, Computer Libris Agora, Cluj-Napoca, 2000.
- [3] A.M.Moşneagu, *Structuri de date*, Note de curs, Iaşi, 2019.
- [4] R. Sedgewick, *Algorithms in C++ Part 5: Graph Algorithms (3rd Edition)*, Addison-Wesley Professional 2008.