Universitatea ,,Alexandru Ioan Cuza", Iaşi
Facultatea de Matematică

Algoritmi pentru grafuri şi aplicaţii

Lucrare de licenţă

Conducător ştiinţific:
Lect.Dr. Ana-Maria Moşneagu

Student:
Galbiniţă Sebastian

Iulie, 2020
Iaşi

Cuprins

Introducere

Multe aplicaţii computaţionale invocă nu numai o pereche de obiecte dar si un set de conexiuni care să lege aceste informatii între ele. Relaţia impusă de 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 destinaţia propusă? Există o legătura între toate tipurile de date? Pentru a ilustra diversitatea aplicaţiilor ce folosesc proprietăţi şi metode, care au ca fundament grafurile, enumerăm următoarele exemple: Hărţile, Hypertexts(documente ce conţin referinţe către alte pagini web), Circuite, Planificări, Tranzacţii, Reţea de internet etc.

Pentru a putea răspunde la intrebările de mai sus, vom folosi obiecte abstracte cum ar fi grafurile. Grafurile sunt structuri de date extrem de răspândite în ştiinţa calculatoarelor, iar algoritmii de grafuri sunt esenţiali in acest domeniu.

În capitolul 1 se tratează problema de a determina toate drumurile de cost minim dintre oricare doua noduri şi determinarea drumurilor minime de la un nod fixat la toate celelalte, când fiecare muchie are asociată o lungime.

În capitolul 2 se v-a descrie determinarea unui arbore de acoperire minimă al unui graf. Acest arbore este introdus ca fiind cea mai "ieftină" cale de conectare a tuturor vârfurilor atunci când fiecare dintre muchie are un cost asociat. Algoritmii de acoperire minimă a unui graf sunt exemple concrete de algoritmi greedy.

În final, în capitolul 3 se v-a discuta despre modul de a putea calcula fluxul maxim de material dintr-o reaţea(graf orientat) având in ipoteza ipoteza sursa de material, destinaţia şi cantitaţile de material care pot traversa muchia.

Pentru evaluarea timpului de execuţie al unui algoritm pe un graf fixat $G = (V, E)$, de obicei vom măsura dimensiunile intrării in funcţie a numărului de vârfuri $|V|$ şi a numărului de muchii $|E|$ ale grafului. Astfel există doi parametrii relevanţi care descriu dimensiunea intrării.

Capitol 1

Noţiuni introductive

Vom incepe studiul cu o introducere in acest capitol a câtorva concepte de baza în teoria grafurilor. Se vor stabili câteva rezultate ca implică aceste concepte. Aceste rezultate, vor servi, în introducerea cititorului la anumite tehnici utilizate frecvent in dovedirea teoremelor în teoria grafurilor.

1.1 Graf

Un *graf* $G = (V, E)$ este o pereche de mulţimi astfel încât $E \subseteq [V]^2$, astfel elementele din E sunt perechi de două subseturi de elmente ale lui V. Pentru a evita ambiguitaţile notaţionale , vom presupune tacit că $V \cap E = \emptyset$. Elementele din mulţimea V se numesc *vârfuri*(sau *noduri* sau *points*) ale grafului G, elementele multimii E se numesc *muchii*(sau *linii*). Modul obişnuit de a ilustra u graf este prin desenarea unui punct pentru fiecare vârf şi unirea celor două puncte printr-o linie daca cele doua varfuri corespunzătoare formeaza o muchie.

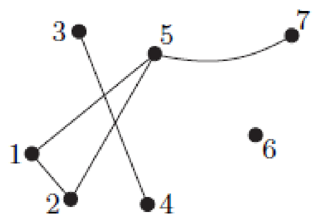


Figure 1.1: Graful pe $V = \{1, 2, 3...7\}$ cu setul de muchii $E = \{\{1, 2\}, \{1, 5\}, \{2, 5\}, \{3, 4\}, \{5, 7\}\}$

Se spune că un graf cu setul de noduri V este un graf pe V. Setul de noduri al unui graf G este denumit $V(G)$, muchia s-a fiind $E(G)$. Aceste convenţii sunt independente de orice alte denumiri ale acestor două seturi:

setul de noduri W a grafului $H = (W, F)$ este tot definit ca fiind $V(H)$, nu ca $W(H)$. Numărul de noduri ale unui graf G este *ordinea* sa, scrisă ca $|G|$; numărul sau de muchii este notat cu $||G||$. Graficele sunt *finite*,*infinite* sau *numărabile*. Pentru *graful null* $(\emptyset, \emptyset)$ scriem pur şi simplu $\emptyset$. Un graf de ordin 0 sau 1 se numeste *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). Dacă $x \in X$ si $y \in Y$ atunci (x, y) este o muchie $X - Y$. Mulţimea tuturor marginilor $X - Y$ dintr-un set E este notat cu $E(X, Y)$.

Doua noduri ale grafului G sunt *adiacente*, sau *vecine* dacă (x, y) formează o muchie pe G. Dacă toate vârfurile lui G sunt perechi adiacente, atunci G este *complet*. Un graf complet de n vârfuri este un K^n; K^3 se numeşte triunghi.

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

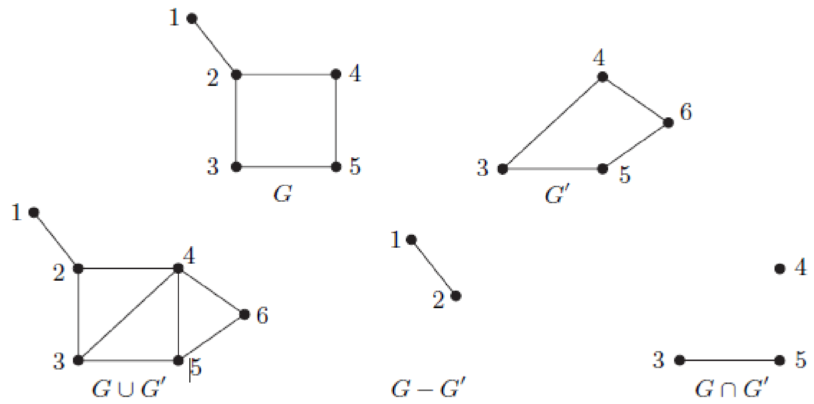


Figure 1.2: Reuniunea, diferenţa si intersecţia; vârfurile 2,3,4 formează un triunghi în $G \cup G'$ dar nu in G

Stabilim că $G \cup G' = (V \cup V', E \cup E')$ şi $G \cap G' = (V \cap V', E \cap E')$. Dacă $G \cap G' = \emptyset$, atunci G şi G' sunt *disjuncte*. Dacă $V' \subseteq V$ şi $E' \subseteq E$, atunci G' este un *subgraf* al ui G (şi G este un *supergraf* pentru G'), scris ca $G' \subseteq G$. Mai puţin formal , spunem ca G il conţine pe G'.

1.2 Reprezentarea unui graf

Există două moduri de reprezentare a unui graf $G = (V, E)$: ca o colectie de listă de liste de adiacenţaăsau ca o matrice de adiacenţă. Oricare dintre aceste două modalităţi de reprezentare se aplică atât grafurilor orientate cât şi celor neorientate.

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

Listele de adiacenţă pot fi adaptate şi pentru realizara unor **grafuri cu cost** astfel, pentru fiecare muchie a grafulu G i se asociază o funcţie numită **funcţie de cost** $\varphi : E \to \mathbb{R}$, unde costul $\varphi(u, v)$ al muchiei (u, v) este memorat împreună cu v în $AD[u]$.

În majoriattea cazurilor, liste de adiacenţă se folosesc atunci când vrem sa gestionăm memoria programului cât mai eficient. De exemplu, dacă avem un graf *rar* ($|E|$ este mult ma mic decât $|V| \times |V|$) şi am folosi reprezentarea grafului cu ajutorul matricilor, atunci majoritatea elementelor din matrice ar rămîne nefolosite producând astfel o risipă de memorie.

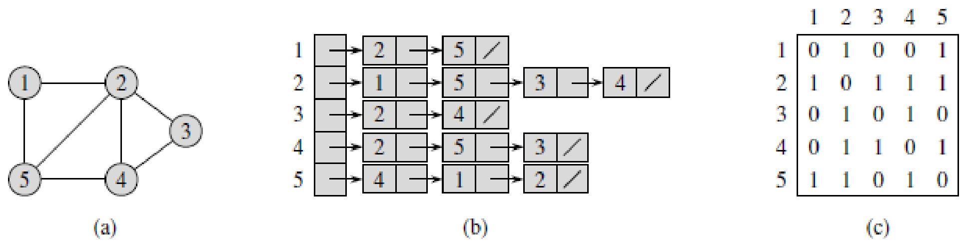


Figure 1.3: Două reprezentări a unui graf neorientat.(**a**) Graf neorientat. (**b**) Listă de adiacenţa pentru G.(**c**) Matricea de adiacenţă a lui G.

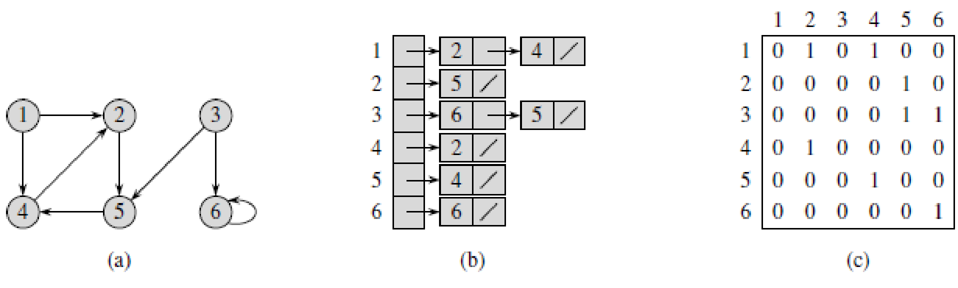


Figure 1.4: Două reprezentări a unui graf orientat.(**a**) Graf orientat. (**b**) Listă de adiacenţa pentru G.(**c**) Matricea de adiacenţă a lui G.

Pentru **reprezentarea prin matrice de adiaceţă**, presupune că v arfurile sunt numerotate arbitrar. Reprezentarea matricii de adiacenţă a grafului $G = (V, E)$ constă într-o matrice $A_{|V| \times |V|} = (a_{ij})$ a.î.:

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

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

La fel ca şi la listele de adiacenţă, pot fi folosite şi pentru grafuri cu cost. Astfel, fie un graf cu cost $G = (V, E)$ şi fie funcţia de cost φ de mai sus , costul $\varphi(u, v)$ 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 sau în majoritatea cazurilor 0 sau∞.

Un avantaj pentru folosirea matricilor in locul listelor este acela că dacă graful este *dens* ($|E|$ este aproxiativ egal cu $|V| \times |V|$) atunci numărul de muchii este aproape de (complet) $n(n-1)/2$ sau de n^2, unde $n = |V|$, dacă graful este orientat ş ciclic.

1.3 Gradul unui vârf

Reprezentarea unui graf $G = (V, E)$ prin liste de adiacenţ a constă intr-un tablou

Fie $G = (V, E)$ un graf nenul. Mulţimea vecinilor nodurilor 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; mulţimea lor este notata cu $N(U)$.

Gradul $d_G(v) = d(v)$ a unui vârf v este numarul de muchii $|E(v)|$ la v. Dacă gradul unui nod este 0 atunci aceste se numeşte *nod izolat*. Numărul $\delta(G) = min\,\{\, d(v) \,|\, v \in V \,\}$ este *gradul minim* lui G, iar numarul $\Delta(G) = max\,\{\, d(v) \,|\, v \in V \,\}$ este *gradul maxim* lui G. Dacă toate nodurile lui G au acelaşi grad k, atunci G este *k-regulat*, sau simplu *regulat*. Un graf 3-regulat se numeste *cub*.

Numărul

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

se numeşte *gradul mediu* a lui G. Deasemena, este indusă de relaţia,

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

1.4 Dumuri si cicluri

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

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

unde toate nodurile v_i sunt distincte. Nodurile v_0 şi v_j sunt *legate* de P si se numesc *capete*; nodurile $v_1, v_2, ..., v_{j-1}$ sunt nodurile interioare ale lui P. Numărul de muchii a unui drum este *lungimea* lui.

Fiind date doua mulţimi A, B de noduri, spunem ca $P = [\,x_0, x_1, .., x_k\,]$ este un *drum* A-B dacă $V(P) \cap A = \{x_0\}$ şi $V(P) \cap B = \{x_k\}$. Două sau mai multe drumuri sunt *independente* dacă nici unul dintre ele nu conţine un nod interior al altuia. De exemplu, două drumuri $a - b$ sunt independente d.d a şi b sunt singurele lor noduri comune.

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

Distanţa dintre doă 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 distanţa dintre oricare două noduri în G este *diametrul* lui G, notată cu diam G.

1.5 Conexitate

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

Pentru grafurile orientate, vom evidenţia două noţiuni asociate cu noţiunea de conexitate. Un graf orientat se numește *slab conex* dacă înlocuirea tuturor muchiilor orienatte cu muchii ale unu graf neorientat produce un graf conex (neorientat).

Fie G un graf orientat, se spune că nodurile x şi y sunt *tare conexe* dacă există simultan un drum de la x la y, şi de la y la x, unde cele două vaârfuri sunt distincte între ele.

Capitol 2

Drumuri minime de sursă unică

Să presupunem că un ciclist doreşte să parcurga drumul de la Iaşi la Bacău acestă utilizând o hartă rutieră a României, unde sunt indicate distanţele între fiecare două intersecţii adiacente.

O posibilă rezolvare a acestei probleme este aceea de a înşirui toate drumurile de la Iaşi la Bacău şi, 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 şi în cazul în care avem drumuri care nu conţin cicluri.

În acest capitol vom demonstra 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 funcţia cost $w : E \longrightarrow \mathbb{R}$ repartizează fiecărei muchii un cost exprimat intr-un numar real. **Costul** drumului $p = \langle \alpha_0, \alpha_1, ..., \alpha_k \rangle$ reprezintă suma costurilor corespunzătoare muchiilor componente :

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

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

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

In cazul exemplului de mai sus, putem modela harta rutieră ca un graf: vârfurile constitue punctele de intersecţie, muchiile reprezintă segmentele de drum iar costurile distanţele între intersecţii.

2.1 Reprezentarea drumurilor minime

Find dat un graf $G = (V, R)$, se v-a reţine pentru fiecare vârf $v \in V$ un **predecesor** $\omega[v]$ care este fie un vârf, fie NULL. Pentru determinarea drumurilor minime, algoritmii prezentaţi in acest capitol determină ω aşa încât pentru orice vârf v, lanţul de predecesori care porneşte de la v să coincidă unei tranversări in ordinea inversă unui drum de valoare minimă de la s la v.

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

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

Mulţimea de muchii E_ω este mulţimea de muchii impusă de valorile lui ω pentru vârfurile din V_ω :

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

Fie $G = (V, E)$ un graf orientat cu muchii cost, având funcţia de cost $\psi : E \longrightarrow R$, presupunem ca graful G nu conţine cicluri de cost negativ, disponible din vârful sursă s, asadar drumurile minime sunt bine definite. Un **arbore al drumurilor minime** de radăcină s este subgraful orientat $G' = (V', E')$ unde $V' \subseteq V$, $E' \subseteq E$ a.î. următoarele condiţii sunt îndeplinite:

1. G' arbore orientat cu rădăcină, având pe s ca rădăcină.

2. V' mulţimea vârfurilor accesibile din s în G.

3. pentru orice $v \in V'$ unicul drum de la s la v in G' este un drum minim de la s la v in G.

Drumurile minime nu sunt întotdeauna unice şi în consecinţă există mai mulţi arbori de drumuri minime.

2.2 Relaxare

Algoritmi pentru determinarea drumurilor minime de sursă unică sunt bazaţi pe o tehnică care poartă numele de **relaxare**. Pentru fiecare vârf $v \in V$, 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 şi a drumorilor minime sunt iniţializate prin următorul algoritm:

INIŢIALIZEAZĂ-SURSĂ-UNICĂ (G,s)

1: **pentru** fiecare vârf $v \in V[G]$ **execută**
2: $\quad d[v] \longleftarrow \infty$
3: $\quad \omega[v] \longleftarrow NULL$
4: $d[s] \longleftarrow 0$

După iniţializare $\omega[v] = NULL$ onetru orice vârf $v \in V$, $d[v] = 0$ pentru $v = 2$, $d[v] = \infty$ pentru $v \in V \setminus \{s\}$

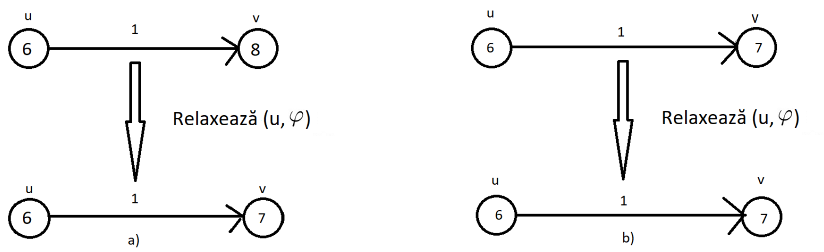


Figure 2.1: Are loc procesul de relaxare a unei muchii (u, v) cu costul $\varphi(u, v) = 1$. Pentru orice vârf $u, v \in V$ est prezentată estimarea drumului minim .**(a)** Înainte de relaxare, $d[v] > d[u] + \varphi(u, v)$, valoarea lui d[v] descreşte. **(b)** Înainte de relaxare, $d[v] \leq d[u] + \varphi(u, v)$, prin urmare valoarea lui $d[v]$ rămâne neschimbată.

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

RELAXEAZĂ (u, v, φ)
dacă $d[v] > d[u] + \varphi(u, v)$ **execută**
    $d[v] \longleftarrow d[u] + \varphi(u, v)$
    $\omega[v] \longleftarrow u$

În figura 2.1 este aplicat algoritmul de mai sus astfel, în **(a)** estimarea drumului minim descreşte iar în **(b)** estimarea nu este modificată.

Toţi algoritmi apelează INIŢIALIZARE-SURSĂ UNICĂ dupa care apelează relaxarea repetată a mchiilor. În algoritmul Dijkstra fiecare muchie este relaxata doar o singură dată iar în cazul aloritmului 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 in anul 1959. Algoritmul Dijkstra calculează cel mai scurt drum prin recursivitate selectând vârful nevizitat cu cea mai mică distanţă faţă 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 cea mai scurtă cale de la nodul sursă până la destinaţie calculând recursiv cele mai scurte căi de la nodul sursă la toate celelalte noduri din grafic.

2.3.1 Algoritmul

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

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

În plus menţinem un tablou boolean $u[\,]$ care stochează pentru fiecare vârf v indiferent dacă este marcat. Iniţial toate vârfurile sunt marcate:

$$u[v] = false$$

Algoritmul lui Dijkstra rulează pentru n iteraţii. La fiecare iteraţie, un vârf v este ales ca vârf nemarcat care are cea mai mică valoare $d[v]$. Evident, prima iteraţie vârful de pornire s va fi selectat. Vârful selectat v este marcat. În continuare, de la vârful v se realizează relaxări: toate marginile formei (v, i) sunt luate in considerare şi pentru fiecare vârf i algoritmul încearcă să îmbunătăţeascăvaloarea $d[v]$. După ce toate aceste margini sunt luate in considerare, iteraţia curentă se termină. În cele din urma după n iteraţii, toate vârfurile vor fi marcate si algoritmul se încheie. Susţinem ca valorile gasite $d[v]$ sunt lungimile celor mai scurte căi de la s la toate vârfurile.

O observaţie ar fi că, dacă unele vârfuri nu pot fi atinse din cele de început, valorile $d[v]$ pentru ele vor rămâne infinit. Evident, ultimele câteva iteraţii 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 distanţă infinită de acesta.

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

DIJKSTRA (G, w, s)
1: INIŢIALIZEAZĂ-SURSĂ-UNICĂ(G, s)
2: $S \longleftarrow \emptyset$
3: $Q \longleftarrow V[G]$
4: **cât timp** $Q \neq \emptyset$ **execută**
5: $\quad u \longleftarrow$ EXTRAGE-MIN(Q)
6: $\quad S \longleftarrow S \cup \{u\}$
7: $\quad$ **pentru** fiecare vârf $v \in Adj[u]$ **execută**
8: $\quad\quad$ RELAXEAZĂ(u, v, w)

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

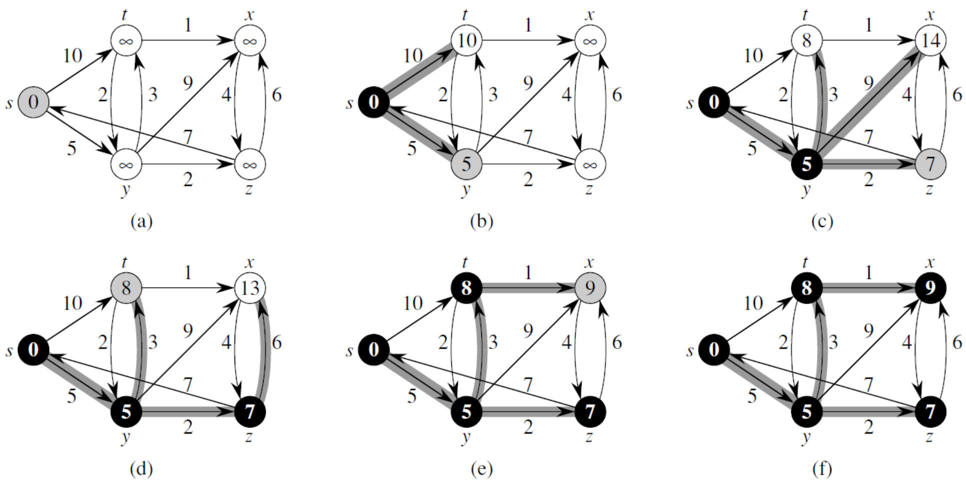


Figure 2.2: Algoritmul Dijkstra pe etape. Vârful sursă este 0. Muchiile haşurate reprezintă valorile predecesorilor: dacă (u, v) este haşurat atunci $\pi[v] = u$. Vârfurile marcate cu negru sunt din S iar cele marcate cu alb aparţin cozii $Q = V - S$.**(a)** Configuraţia există înaintea primei iteraţii a repetiţii **cât timp**. Vârful haşurat este u din linia 5 şi are valoarea minimă.**(b)-(f)** Configuraţia după fiecare iteraţie **cât timp**.

2.3.2 Implementare

Algoritmul Dijkstra efectuează n iteraţii. La fiecare iteraţie selecteazăun vârf nemarcat v cu cea mai mică valoare $d[v]$, îl marchează şi verifică toate marginile (v, i) încercând să îmbunătăţească valoarea $d[i]$.

Durata de rulare a acestui algoritm este de :

- n caută un vârf cu cea mai mică valoare $d[v]$ printre $O(n)$ vârfuri nemarcate.
- m încercări relaxări.

Pentru cea mai simplă implementare a acestor operaţii pe fiecare vârf de iterac tie căutarea necesită$O(n)$ operac tii şi fiecare relaxare poate fi efectuată în $O(1)$. Prin urmare, comportamentul asimptotic rezultat al algoritmului este:

$$O(n^2 + m)$$

Această complexitate este optimă pentru un graf cu costuri, adicăatunci 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 $O(nlog(n) + m)$.

```cpp
#include <iostream>
#define Varfuri 9
using namespace std;
//distanta minima din setul de noduri
int distantaMinima(int distanta[], bool inclus[]){
    int min = INT_MAX, min_index;
    for (int v = 0; v < Varfuri; v++)
        if (inclus[v] == false && distanta[v] <= min) min = distanta[v], min_index = v;
    return min_index;
}

void printSolution(int dist[]){
    cout<<("Nod \t Distanta pana la sursa\n");
    for (int i = 0; i < Varfuri; i++) cout<<i<<"\t"<< dist[i]<<endl;
}

void Dijkstra(int graph[Varfuri][Varfuri], int src){
    int distanta[Varfuri]; //Va tine cel mai scurt drum de la sursa la i
    bool inclus[Varfuri]; //Va fi true daca va fi inclus in cel mai scurt drum de la src la i
    for (int i = 0; i < Varfuri; i++) {
        distanta[i] = INT_MAX;
        inclus[i] = false;
    }
    distanta[src] = 0;
    // Gaseste cel mai scurt drum
    for (int count = 0; count < Varfuri - 1; count++) {
        int u = distantaMinima(distanta, inclus);//ia distanta minima din setul de varduri  care nu a fost deja procesat
        inclus[u] = true;//il marcam
        // Update
        for (int v = 0; v < Varfuri; v++)
            if (graph[u][v] && distanta[u] != INT_MAX && !inclus[v] && distanta[u] + graph[u][v] < distanta[v])
                distanta[v] = distanta[u] + graph[u][v];
    }
    printSolution(distanta);
}

// driver program to test above function
int main(){
    int graph[Varfuri][Varfuri] = { { 0, 5, 0, 0, 0, 0, 0, 9, 0 },
                                    { 5, 0, 8, 0, 0, 0, 0, 12, 0 },
                                    { 0, 8, 0, 8, 0, 5, 0, 0, 3 },
                                    { 0, 0, 8, 0, 10, 15, 0, 0, 0 },
                                    { 0, 0, 0, 10, 0, 12, 0, 0, 0 },
                                    { 0, 0, 5, 15, 12, 0, 3, 0, 0 },
                                    { 0, 0, 0, 0, 0, 3, 0, 2, 7 },
                                    { 9, 12, 0, 0, 0, 0, 2, 0, 8 },
                                    { 0, 0, 3, 0, 0, 0, 7, 8, 0 } };
    Dijkstra(graph, 0);
    return 0;
```

Microsoft Visual Studio Debug Console

```
Nod      Distanta pana la sursa
0       0
1       5
2       13
3       21
4       26
5       14
6       11
7       9
8       16
```

Figure 2.3: Algoritmul afişează toate costurile drumurilor de la i la sursă.

Capitol 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 distanţelor între toate perechile de magazine. Ipotezele sunt aceleaşi ca în capitolul anterior, avem un graf orientat $G = (V, E)$, cu costuri şi o functie de costuri $w : E \longrightarrow \mathbb{R}$ aplicată arcelor grafului. Dorim să determinăm , pentru fiecare $u, v \in V$, un drum cu cost minim de la u la v, unde acest rezultat este suma costurilor acelor arce care formează acest drum. Rezultatul obţinut este de preferat să fie sub forma unui tabel: linia u, coloana v şi conţinutul drumului minim de la u la v.

Spre deosebire de algoritmii folosiţi anteriori, majoritatea algoritmilor din cadrul acestui capitol vor avea reprezentarea prin matrici de adiacenţă. 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

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

Output-ul este o matrice $D = (d_{ij})$ de dimensiune $n \times n$ ale 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 şi **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

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

şi

$$E_{\pi,i} = \{(\pi_{ij}, j) : j \in V_{\pi,i} \ si \ \pi_{ij} \neq NUL\}.$$

Dacă $G_{\pi,i}$ îndeplineşte condiţia de a fi un arbore de drum minim, atunci are loc următoarea procedura, aceea de a aplica metoda AFIŞEAZĂ-DRUM, care afişează drumul minim de la i la j.

AFIŞEAZĂ-DRUMURILE-MINIME (P, i, j)
1: **dacă** $i = j$ atunci
2: afişează ste i
3: **altfel**
4: **dacă** $\pi_{ij} = NIL$ **atunci**
5: afişează "Nu este drum de la i la j"
6: **altfel**
7: AFIŞEAZĂ-DRUMURILE-MINIME (P, i, π_{ij})
8: afişează j

3.1 Drumuri minime şi înmultirea matricelor

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

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

Definim $d_{ij}^{(m)}$ ca fiind costul minim a unui drum de la i la j care este alcătuit din cel mul 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 minimul între $d_{ij}^{(m-1)}$ şi costul minim al fiecarui drum de la i la j cu cel mult m arce , luând în considerare toţi predecesorii k ai lui j.

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

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

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

Avem urmatoarea problema, dorim sa determinăm in mod ascendent costurile drumurilor minime. Considerăm ca input o matrice $A = (a_{ij})$ şi vom determina o listă de matrici $D^{(1)}, D^{(2)}, ..., D^{(n-1)}$, unde pentru orice $m = 1, 2, .., n-1$ avem $D^{(m)} = \left(d_{ij}^{(m)}\right)$. Matricea $D^{(n-1)}$ va contine costurile drumurilor minime. O observaţie importantă ar fi că dacă $d_{i,j}^{(1)} = a_{ij}$ pentru orice $i, j \in V$, obţinem $D^{(1)} = A$. Cu alte cuvinte, dându-se matricele $D^{(m-1)}$ şi A se va obtine matricea $D^{(m)}$ care reprezintă extinderea drumurilor minime cu încă un arc.

EXTINDE(D, A)
$n \leftarrow linii[D]$
fie $B = (b_{ij})$ matrice cu dimensiunea $n \times n$
pentru $i \leftarrow 1, n$ **execută**
    pentru $j \leftarrow 1, n$ **execută**
        $b_{ij} \leftarrow \infty$
        pentru $k \leftarrow 1, n$ **execută**
            $b_{ij} \leftarrow \min(b_{ij}, d_{ik} + w_{kj})$
returnează B

Timpul de execuţie al acestei funcţii este de $O(n^3)$ datorită celor 3 bucle pe care le conţine. Funcţia returnează matricea $B = (b_{ij})$, acest lucru realizânduse cu ajutorul ecuţiei (3.1) pentru orice i, j utilizând D pentru $D^{(m-1)}$ şi B pentru $D^{(m)}$.

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

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

ÎNMULŢEŞTE-MATRICILE(A, B)
$n \leftarrow linii[A]$
fie $C = (c_{ij})$ matrice cu dimensiunea $n \times n$
pentru $i \leftarrow 1, n$ **execută**
    pentru $j \leftarrow 1, n$ **execută**
        $c_{ij} \leftarrow 0$
        pentru $k \leftarrow 1, n$ **execută**
            $c_{ij} \leftarrow c_{ij} + a_{ik} * b_{kj}$
returnează C

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

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

Mini-GPS

GPS este un sitem de navigaţie în forma unui mic dispozitiv ataşat unei maşini,avion sau a unei nave. Cu tehnoligia din ziua de azi, GPS este deasemenea folosit şi în alte dispozitive electronice cum ar fi: camera, telefonul, si calculatorul. GPS primeşte informaţii utile de navigare în timp real sub formă de coordonate din sateliţi la fiecare câteva minute. Un dispozitiv GPS afişează deasemenea o mapă detaliată unei regiuni cu oraşele învecinate şi reţeaua de drumuri care leagă oraşele. În maşina, GPS-ul îl ajută pe şofer să urmeze cea mai scurtă cale de la sursă la destinaţie. GPS-ul de astăzi are caracteristici suplimentare cum ar fi: furnizarea de căi alternattive la cea mai scurtă cale pentru a evita traficul sau construc ctia drumurilot pe această cale. Această informaţie 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 uşurinţă în dispozitiv.

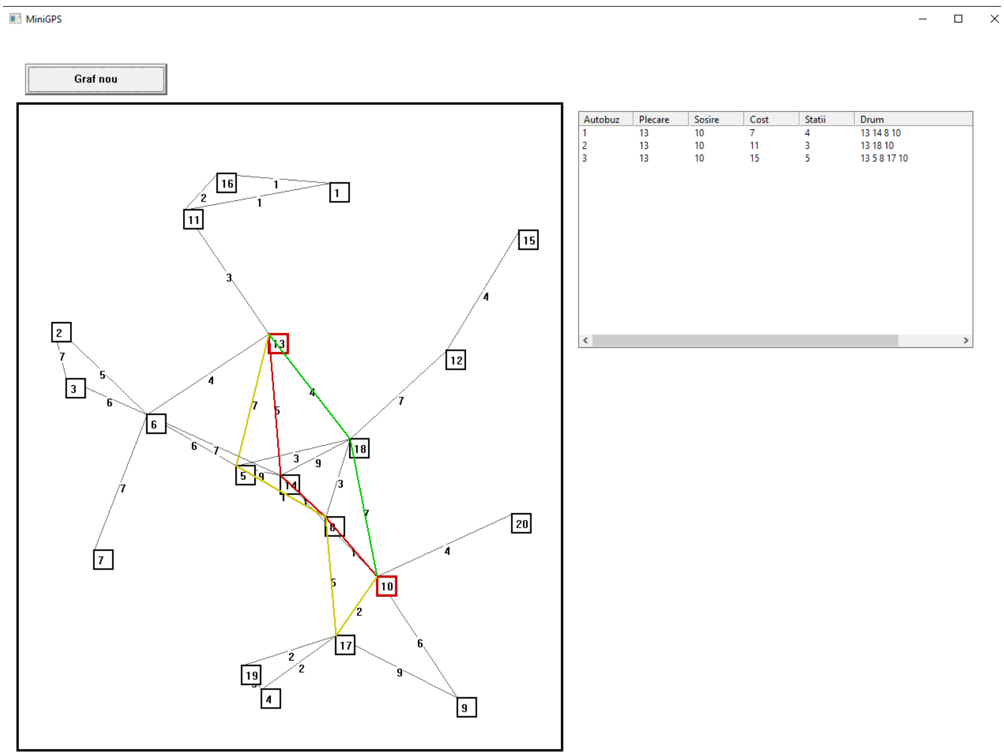


Figure 3.1: Output-ul codului MiniGPS care arată trei autobuze de la sursă la destinaţie.

Implementare

MiniGPS este implementarea algoritmului lui Dijkstra pentru problema de k drumuri scurte, care este aproximativ găsirea k diferite drumuri mai scurte între o pereche de noduri. Proiectul ilustrează un model GPS simplu pentru afiąrea a trei rute diferite între sursă și nodurile de destinaţie. Proiectul produce versiunea offline a GPS-ului în care nu există date în timp real cu privire la coordonatele actuale ale şoferului.

Figura de mai sus afişează un simplu output pentru MiniGPS. Rezultatul constă în afişarea unui graf cu noduri si muchii generate aleatori şi o fereastra de vizualizare în care se găsesc informaţii de rutare. În zona de desen sunt 20 de noduri, nodul v_{13} este nodul sursă iar v_{10} nodul destinaţie. Programul produce trei diferite drumuri cu cost minim de la nodul sursă la destinaţie. Fiecare drum din graf este numit *bus*, care este definit ca un unic drum de la sursă la destinaţie. Programul afişează deasemenea şi alte informaţii despre autobuze cum ar fi: costul total, numarul de staţii precum şi drumul.

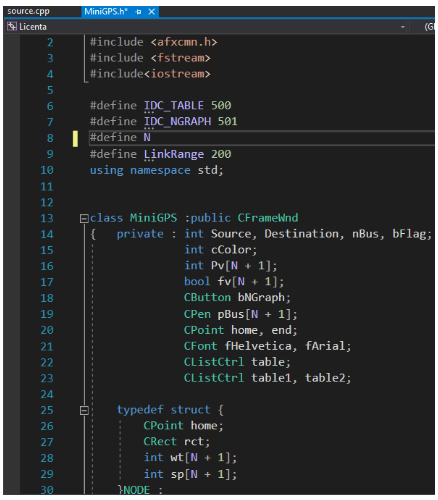
MiniGPS implementează algoritmul lui Dijkstra pentru gasirea celor 3 drumuri de cost minim. Primul drum este obţinut din graful original G cu 20 de noduri. Odată ce drumul a fost obtinut, muchiile ḑea lungul drumului sunt & indepărtate pentru a reduce G la G'. Algoritmul lui Dijkstra este aplciat din nou pentru G' pentru a produce cel de-al doilea drum, care are un set de muchii diferit faţa de primul. Respectând algoritmul, muchiile celui de-al doilea autobuz sunt îndepartate pentru a reduce G' la G''.Analog se aplică acelaşi procedeu şi pentru cel de-al treilea drum. Programul are o singură clasă numită $MiniGPS$ cu $MiniGPS.h$ ca header şi *Source.cpp* ca sursă.

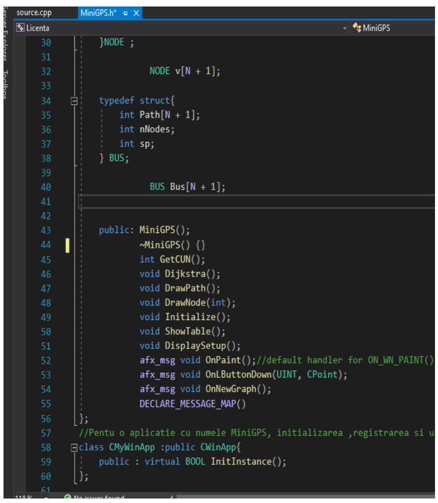
Tabelul prezintă câteva variabile şi obiecte din $MiniGPS$. O structură numită BUS care include diferite autobuze într-un tablou numit Bus pentru a reprezenta drumurile drumurile cu succes între nodurile sursă şi destinaţie. În acest program, Bus este conectat la membrii din BUS şi anume de mulţimea $Path$, care reprezintă numărul de noduri $nNodes$ de- a lungul drumului şi *sp* care este costul total.

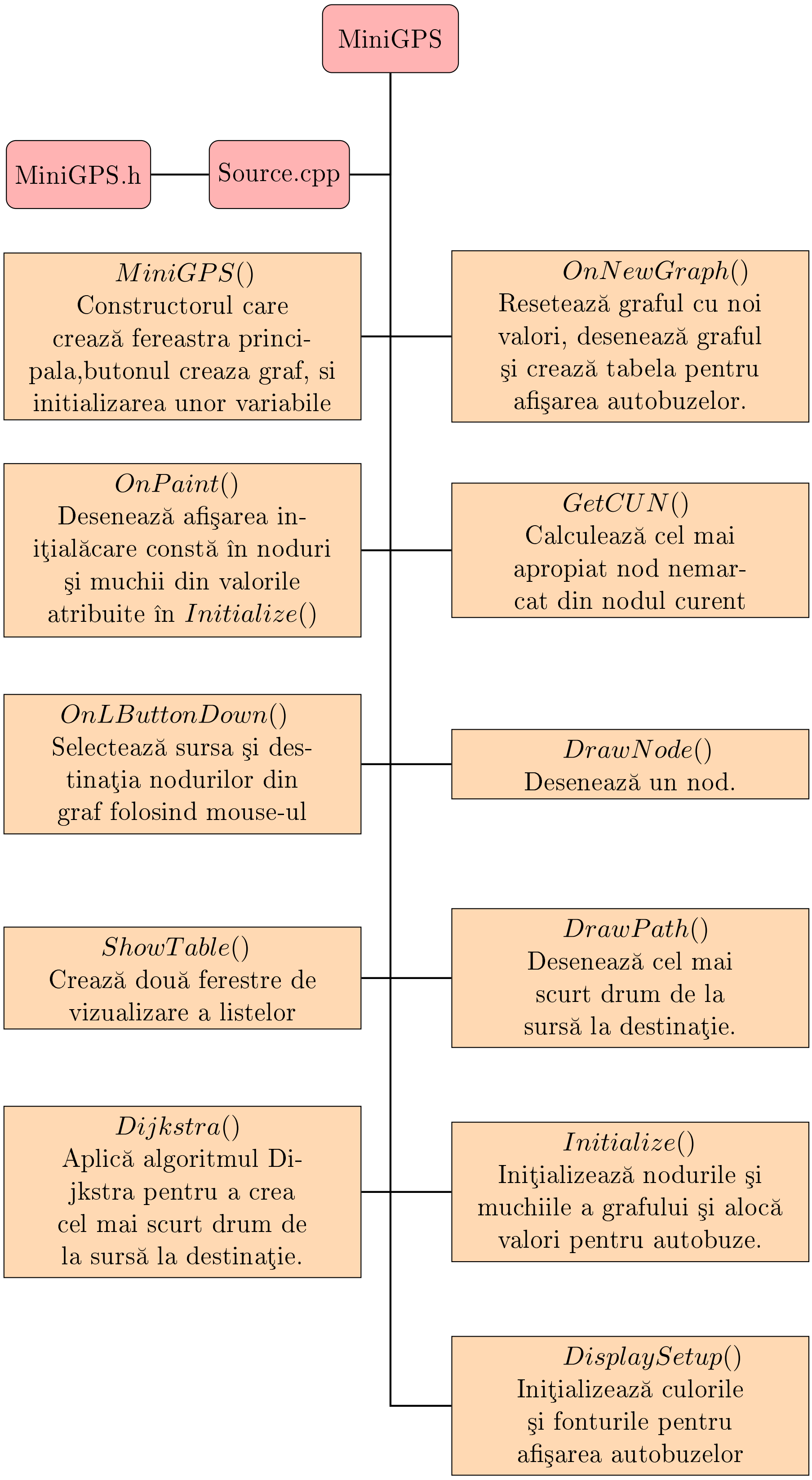
Table

Variabile şi obiecte importante din *MiniGPS* şi descrierea lor.

MiniGPS		
Variabile	Tipul	Descriere
$bNGraph$	$CButton$	Buton pentru generarea unui nou graf
$BCompute$	$CButton$	Compune cel mai scurt drum
$Bus[i].path[k]$	int	Drumul k în autobuzul i
$Bus[i].nNodes$	int	Numărul de noduri în autobuzul i
$Bus[i].sp$	int	Costul total al autobuzului i
$home$	$CPonit$	Colţul din stânga sus al zonei grilei dreptunghiului
$v[i].wt[j]$	int	Costul dintre (v_i, v_j)
$v[i].sp[j]$	int	Drumul cel mai scurt dintre (v_i, v_j)
Pv	int	Nodul precedent al nodului curent
$Source$, $Destination$	int	Sursa, nodul destinaţie
$nBus$	int	Numarul de autobuze de succes
$table$	$CListCtrl$	Tabela care afişează autobuzele de succes
$pBus[i]$	$CPen$	Culoarea autobuzului i
N	$constant$	Numărul de noduri în graf
$LinkRange$	$constant$	Valoarea pragului intervalului pentru adiacenţa dintre douănoduri din grafic
$fv[i]$	$bool$	Starea vârfului v_i






MiniGPS
MiniGPS.h
Source.cpp
MiniGPS()
Constructorul care crează fereastra princi-pala,butonul creaza graf, si initializarea unor variabile
OnNewGraph()
Resetează graful cu noi valori, desenează graful şi crează tabela pentru afişarea autobuzelor.
OnPaint()
Desenează afişarea in-iţialăcare constă în noduri şi muchii din valorile atribuite în Initialize()
GetCUN()
Calculează cel mai apropiat nod nemar-cat din nodul curent
OnLButtonDown()
Selectează sursa şi des-tinaţia nodurilor din graf folosind mouse-ul
DrawNode()
Desenează un nod.
ShowTable()
Crează două ferestre de vizualizare a listelor
DrawPath()
Desenează cel mai scurt drum de la sursă la destinaţie.
Dijkstra()
Aplică algoritmul Di-jkstra pentru a crea cel mai scurt drum de la sursă la destinaţie.
Initialize()
Iniţializează nodurile şi muchiile a grafului şi alocă valori pentru autobuze.
DisplaySetup()
Iniţializează culorile şi fonturile pentru afişarea autobuzelor

Aceasta este schema pentru aplicaţia MiniGPS. Sistemul mini-GPS pornc ste de la constructorul *MiniGps()*, care creează o fereastră şi butonul *Graf nou.* Această funcţie apelează deasemenea funcţiile *DisplaySetup()* şi *OnNewGraph* care stabileşte variabilele de afişare comune şi creează graful. *OnNewGraph()* apelează *Initialize* care creează graful prin alocarea coordonatelor aleatorii la noduri şi costul aleatorii la muchiile grafului. Graful initial este afişat şi updatat prin *OnPaint()*.

Evenimentul *ON_ WM_ LBUTTONDOWN()* detectează clik-ul din stânga mous-ului a că rui poziţie se află în Windows returnat de obiectul *CPoint pt.* Valoarea obiectului este verificată cu *v[i].rct* folosind *PtInRect()*. Nodul sursă este identificat prin *bFlag=1* iar destinaţia prin *bFlag=2*.

Dijkstra() calculează cel mai scurt drum folosing algoritmul lui Dijkstra. Funcţia este apelată atunci când nodul secund a fost selectat în timp ce autobuzul de la sursă la nodul destinaţie este desenat folosind *DrawPath()*. Odata ce primul autobuz a fost finalizat, informaţiile sale sunt actualizate şi afişate în fereastra de vizualizare. Deasemenea , muchiile de la primul autobuz sunt şterse din graful original. Ştergerea 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 neadiacente dealungul parcurgerii. Programul calculează noul drum care în mod cert avită primul drum. Similar, se aplică aceeaşi metodă şi pentru drumul cu numă rul 3.