

Graf w Scilabie reprezentowany jest poprzez określoną listę. Nazywamy ją listą danych grafu.. Lista grafu składa się z 33 elementów. Pierwsze pięć musi posiadać wartość w liście. Pozostałe mogą mieć puste wartości i mogą być nie używane.

Tych pięć elementów wymaganych to:

name – nazwa grafu,

directed – oznaczenie typu grafu; graf skierowany = 1, graf nieskierowany = 0,

node_number – liczba węzłów w grafie,

tail – wektor jednowymiarowy zawierający początki krawędzi (jego rozmiar jego jest równy ilości krawędzi w grafie),

head – wektor jednowymiarowy zawierający końce krawędzi (jego rozmiar jego jest równy ilości krawędzi w grafie),

Na przykład, można zdefiniować listę grafu w następujący sposób:

```
g=make_graph('min',1,1,[1],[1]);
```

który jest najprostszym grafem, jaki można utworzyć (graf jest skierowany, posiada jeden wierzchołek i jedną pętlę na tym wierzchołku).

make_graph - tworzy listę danych grafu.

Wywołanie funkcji:

```
g=make_graph(name,directed,node_number,tail,head)
```

Opis:

make_graph tworzy listę danych grafu zgodnie z jej argumentami które są odpowiednio nazwą grafu, znacznikiem dla grafu skierowanego lub nieskierowanego, liczbą węzłów oraz wektorami jednowymiarowymi początku i końca. Są one minimalnymi danymi potrzebnymi do zbudowania grafu. Jeżeli n jest równe 0, graf g nie ma izolowanych węzłów a ich liczbę wylicza się z tail oraz head. Nazwy węzłów brane są z numerów w tail i head.

5.8.1. Zastosowanie biblioteki Metanet: graph and network toolbox

Biblioteka Metanet: graph and network toolbox posiada bardzo dużo funkcji służących do tworzenia, modyfikacji, obliczeń i różnego typu działań na grafach, sieciach. W owej pracy zostaną przedstawione przykładowe funkcje.

- **Macierz incydencji oraz lista sąsiedztwa**

Opis funkcji służącej do wyświetlania macierzy incydencji dla grafu (węzeł-łuk, węzeł-węzeł):

graph_2_mat – macierz incydencji dla grafu (węzeł-łuk, węzeł-węzeł).

Wywołanie:

```
a = graph_2_mat(z,mat)
```

gdzie:

z – graf (lista danych);

mat – opcjonalnie – ciąg znakowy, macierz ‘węzeł-łuk’ albo ‘węzeł-węzeł’;

a – macierz incydencji;

Opis:

graph_2_mat wylicza macierz incydencji dla grafu z odpowiadającej macierzy a typu węzeł-węzeł lub węzeł-łuk. Jeżeli opcjonalny argument mat zostanie pominięty albo zostanie zapisany jako ciąg znakowy 'node-arc', musi to być macierz węzeł-łuk. Jeżeli mat jest ciągiem 'node-node', musi to być macierz węzeł-węzeł.

Opis funkcji służącej o wyświetlenie listy sąsiedztwa:

adj_lists – wyświetlanie list sąsiedztwa

Wywołanie:

```
[lp,la,ls] = adj_lists(z)
```

`[lp,la,ls] = adj_lists(czy_skierowany,n,tail,head)`

gdzie:

`z` – graf (lista danych);

`czy_skierowany` – liczba całkowita, 0 (graf nie skierowany) albo 1 (graf skierowany);

`n` – liczba całkowita, liczba węzłów grafu;

`tail` – wektor jednowymiarowy zawierający początki krawędzi;

`head` - wektor jednowymiarowy zawierający końce krawędzi;

`lp` – tablica, tablica wskaźników z list danych opisujących graf ;

`la` – tablica, tablica krawędzi z list danych opisujących graf;

`ls` – tablica, tablica węzłów z list danych opisujących graf ;

Opis:

`adj_lists` wyliczają wektory jednowymiarowe sąsiedztwa opisujących graf `z`. Jest także możliwe podanie przez `adj_lists` opisu grafu po liczbie węzłów `n` oraz wektorach `tail` i `head`.

Przykładowy skrypt:

`clear;`

`head=[2 3 5 4 5 1 6 2 5];`

`tail=[1 1 2 3 3 4 4 6 6];`

`z=make_graph('graf',1,6,tail,head);`

`z.node_x=[40 230 240 200 470 500];`

`z.node_y=[220 350 200 50 280 160];`

`z.node_name=[1 2 3 4 5 6];`

`z.node_type=[2 0 0 0 0 1];`

`z.node_color=[1 1 1 1 1 1];`

`z.node_diam=[50 50 50 50 50 50];`

`z.node_border=[3 3 3 3 3 3];`

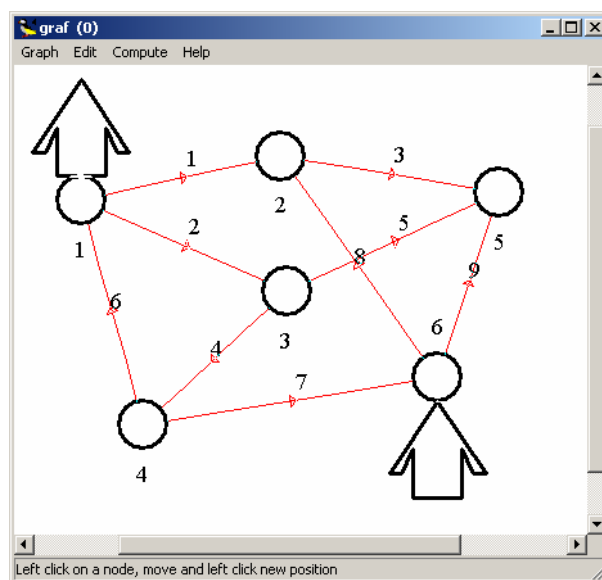
`z.edge_name=[1 2 3 4 5 6 7 8 9];`

`z.edge_color=[5 5 5 5 5 5 5 5 5];`

`z.edge_cost=[2 4 4 2 3 1 2 2 1];`

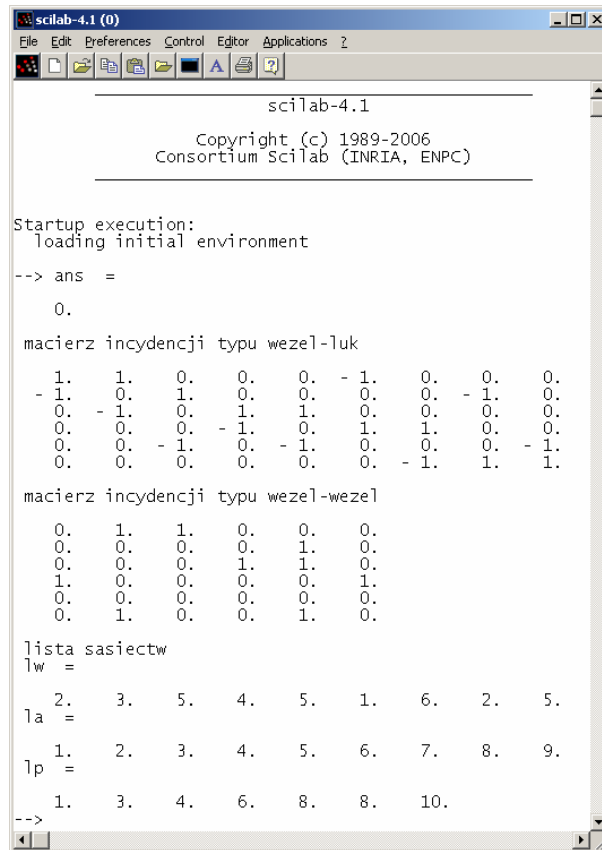
`z.node_name=string([1 2 3 4 5 6]);`

Po zapisaniu przykładowego skryptu z rozszerzeniem .sce z menu okienka SciPada należy wybrać ‘Execute’ a następnie ‘Load into Scilab’. Rysunek 6.1. przedstawia edytor graficzny wraz z grafem skierowanym utworzonym za pomocą przykładowego skryptu.



Rys. 6.1. Graf „z” utworzony za pomocą przykładowego skryptu

W oknie głównym Scilaba (rysunek 6.2.) zostanie wyświetlona macierz incydencji typu węzeł-łuk i węzeł- węzeł oraz lista sąsiedztwa.



```

scilab-4.1 (0)
File Edit Preferences Control Editor Applications ?
scilab-4.1
Copyright (c) 1989-2006
Consortium Scilab (INRIA, ENPC)

Startup execution:
loading initial environment

--> ans =
0.

macierz incydencji typu wezel-luk
1. 1. 0. 0. 0. - 1. 0. 0. 0.
- 1. 0. 1. 0. 0. 0. 0. - 1. 0.
0. - 1. 0. 1. 1. 0. 0. 0. 0.
0. 0. 0. - 1. 0. 1. 1. 0. 0.
0. 0. - 1. 0. - 1. 0. 0. 0. - 1.
0. 0. 0. 0. 0. 0. - 1. 1. 1.

macierz incydencji typu wezel-wezel
0. 1. 1. 0. 0. 0.
0. 0. 0. 0. 1. 0.
0. 0. 0. 1. 1. 0.
1. 0. 0. 0. 0. 1.
0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0.

lista sasiectw
lw =
2. 3. 5. 4. 5. 1. 6. 2. 5.
la =
1. 2. 3. 4. 5. 6. 7. 8. 9.
lp =
1. 3. 4. 6. 8. 8. 10.
-->

```

Rys. 6.2. Macierz incydencji oraz lista sąsiedztwa dla przykładowego grafu

- **Maksymalny przepływ pomiędzy dwoma węzłami**

Funkcja **max_flow** służy do obliczania maksymalnego przepływu pomiędzy dwoma węzłami.

Wywołanie funkcji:

$[v, \phi, \text{flag}] = \text{max_flow}(i, j, g)$

gdzie:

i – liczba całkowita, numer węzła początkowego;

j – liczba całkowita, numer węzła końcowego;

g – graf (lista danych);

v – wartość maksymalnego przepływu jeżeli istnieje;

ϕ – wektor jednowymiarowy zawierający wartości przepływów poszczególnych łuków;

flag – znacznik wykonania problemu (0 albo 1);

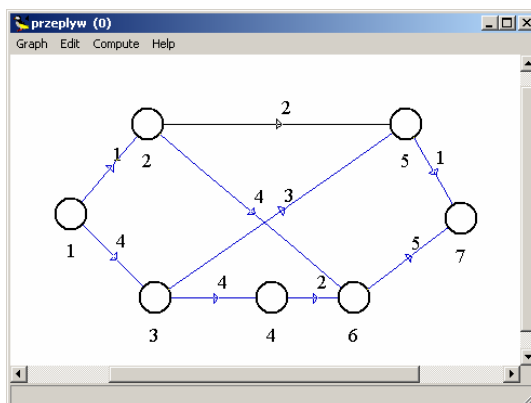
Opis:

`max_flow` zwraca wartość największego przepływu v od węzła i do węzła j jeżeli istnieje oraz wartość przepływu w poszczególnych łukach jako wektor ϕ . Wszystkie wyliczenia wykonywane są na liczbach całkowitych. Graf musi być skierowany. Jeżeli problem nie jest rozwiązywalny, znacznik `flag` równy jest 0, w przeciwnym wypadku 1. Granice przepływu dane są poprzez elementy `edge_min_cap` oraz `edge_max_cap` na podstawie listy danych grafu. Wartość maksymalnej pojemności musi być większa albo równa minimalnej wartości pojemności. Jeżeli nie są dane wartości `edge_min_cap` albo `edge_max_cap` (pusty wektor `[]`), zakłada się że są równe 0 na każdej krawędzi.

Przykładowy skrypt:

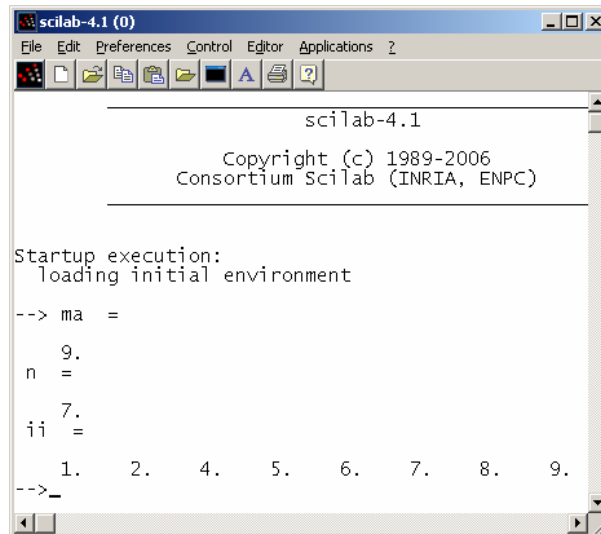
```
g=load_graph('przeplyw.graph');
s=1;t=7;
[v,phi,ierr]=max_flow(s,t,g);
ma=edge_number(g)
n=node_number(g)
ii=find(phi<>0)
edgecolor=phi; edgecolor(ii)=11*ones(ii);
g.edge_color=edgecolor;
g.edge_label=string(phi);
show_graph(g);
```

Rysunek 6.3. przedstawia edytor graficzny wraz z grafem skierowany utworzonym za pomocą przykładowego skryptu.



Rys. 6.3. Graf „g” utworzony za pomocą przykładowego skryptu

W oknie głównym Scilaba (rysunek 6.4) zostanie wyświetlony maksymalny przepływ.



Rys. 6.4. Maksymalny przepływ dla grafu „g”

6. Implementacja oraz wizualizacja algorytmów w Scilabie.

6.1. Minimalna droga - algorytm Dijkstry

- Implementacja algorytmu Dijkstry w Scilabie.

```
function [] = fdijkstra(h)

n=h.node_number;
w=zeros(n,n);
for i=1:arc_number(h),
w(h.tail(i),h.head(i)) = h.edge_cost(i);
end;
for i=1:n,
    for j=1:n, if i<>j then if w(i,j)==0 then w(i,j)=%inf;end,end,
end,end;
s=evstr(x_dialog('Podaj wierzchołek początkowy',' '))
t=evstr(x_dialog('Podaj wierzchołek końcowy',' '))
for v=1:n
    droga(v)=%inf; koniec(v)=%f; poprzednik(v)=-1
end;
droga(s)=0; koniec(s)=%t;
p=%t; cecha=s;
while ~koniec(t)
    for v=1:n
        if w(cecha,v) < %inf & ~koniec(v) then
            NEWLABEL=droga(cecha)+w(cecha,v);
            if NEWLABEL < droga(v) then
                droga(v)=NEWLABEL; poprzednik(v)=cecha;end;
            end;
        end;
    end;
    TEMP=%inf;
    for v=1:n
        if droga(v) < TEMP & ~koniec(v) then
            TEMP=droga(v); t=v;
        end;
    end;
end;
```



```

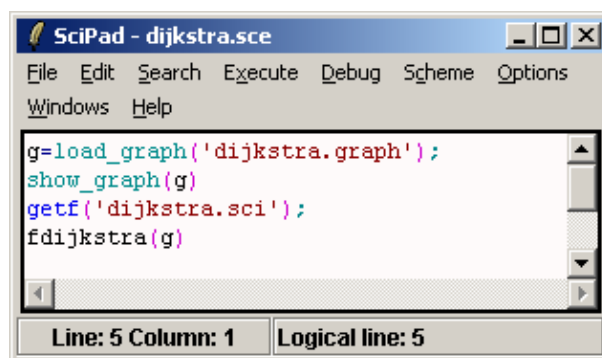
for i=1:n
    if ~koniec(i) & droga(i) < TEMP then
        Y=i; TEMP=droga(i)
    end;end;
    if TEMP < %inf then
        koniec(Y)=%t; cecha=Y;
    else
        p=%f; koniec(t)=%t;
    end;
end;

disp(droga)
show_graph(h)
endfunction;

```

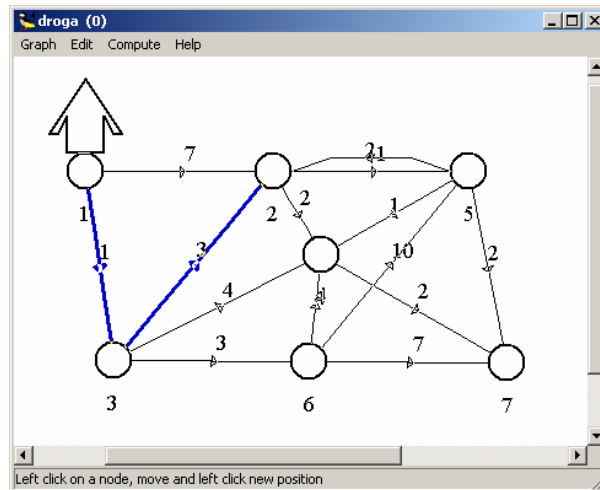
- Wizualizacja działania algorytmu Dijkstra w Scilabie.

Funkcje uruchamiamy za pomocą dodatkowego skryptu (rys. 7.1.) który powoduje załadowanie grafu 'dijkstra.graph' i funkcji 'dijkstra.sci'.



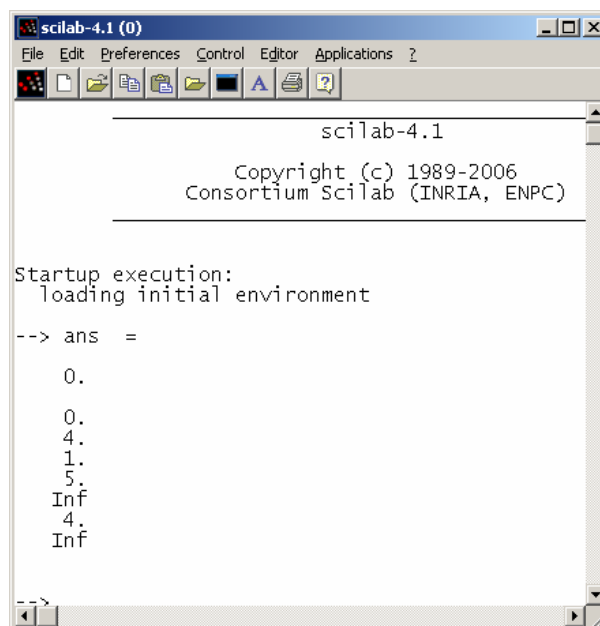
Rys. 7.1. Dodatkowy skrypt

Po wybraniu „Execute” a następnie „Load into Scilab” wyświetla się graf 'dijkstra.graph' (rysunek 7.2.) oraz okienko w którym wpisujemy wierzchołek pierwszy. Po wciśnięciu „ok” pojawia się drugie okienko w którym wpisujemy wierzchołek końcowy i zatwierdzamy „ok”.



Rys. 7.2. Graf dijkstra.graph

Na rysunku 7.2. pogrubioną niebieską linią została pokazana najkrótsza droga z wierzchołka początkowego o numerze 1 do wierzchołka końcowego o numerze 2.



Rys. 7.3. Tablica minimalnych dróg

Na rysunku 7.3. została wyświetlona tablica minimalnych dróg. Od wierzchołka numer 1 do wierzchołka numer 2 minimalna droga wynosi 4.

6.2. Minimalne drzewo rozpinające - algorytm Prima

- Implementacja algorytmu Prima w Scilabie

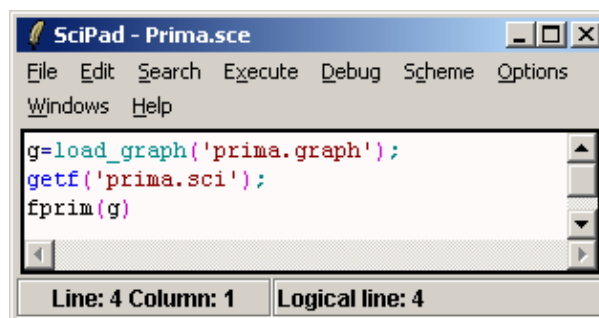
```
function [] = fprim(Graf)
    WektorTail=Graf.tail;
    WektorHead=Graf.head;
    WektorKosztow=Graf.edge_cost;
    MaksymalnyKoszt=max(WektorKosztow);
    LiczbaKrawedzi=arc_number(Graf);
    licz=0;
    n=Graf.node_number;
    w=zeros(1,n);
    for i=1:LiczbaKrawedzi/2,
        [Min,Idx]=mini(WektorKosztow); //wyszukanie krawędzi o minimalnym koszcie,
        Idx wskazuje na której pozycji w wektorze jest krawędź o minimalnym koszcie
        WektorKosztow(Idx)=MaksymalnyKoszt+1; //wpisanie wartości większej o jeden
        od maksymalnego kosztu w miejsce Idx (oznaczenie krawędzi już sprawdzonej)
        w1=WektorTail(Idx); //w1 oraz w2 zawierają nr węzłów między którymi jest
        krawędź o minimalnym koszcie
        w2=WektorHead(Idx);
        p = nodes_2_path([w1,w2],g); //ścieżka utworzona na podstawie dwóch węzłów
        if (w(w1)/w(w2))==0 then
            w(w1)=i;w(w2)=i;
            disp(w);
            licz=licz+1;
            DoWytłuszczenia(licz)=p;
        else
            if w(w1)<>w(w2) then
                KopiaW1=w(w1);
                KopiaW2=w(w2);
                if w(w1)==0 then w(w1)=i; KopiaW1=w(w1); KopiaW2=w(w2); end;
                if w(w2)==0 then w(w2)=i; KopiaW1=w(w2); KopiaW2=w(w1); end;
            end
        end
    end
end
```

```

for j=1:n,
    if w(j)==KopiaW1 then w(j)=KopiaW2; end,
    end;
    disp(w);
    licz=licz+1;
    DoWytluszczenia(licz)=p;
end;
end;
end;
disp(DoWytluszczenia);
show_graph(g);
for j=1:licz,
    x_message(['Wytuszcza kolejne krawędzie po naciśnięciu klawisza Enter'],['ok']);
    halt; //Wytuszcza kolejne krawędzie po naciśnięciu klawisza Enter
    ShowTime(j)=DoWytluszczenia(j);
show_arcs(ShowTime);
end;
endfunction;

```

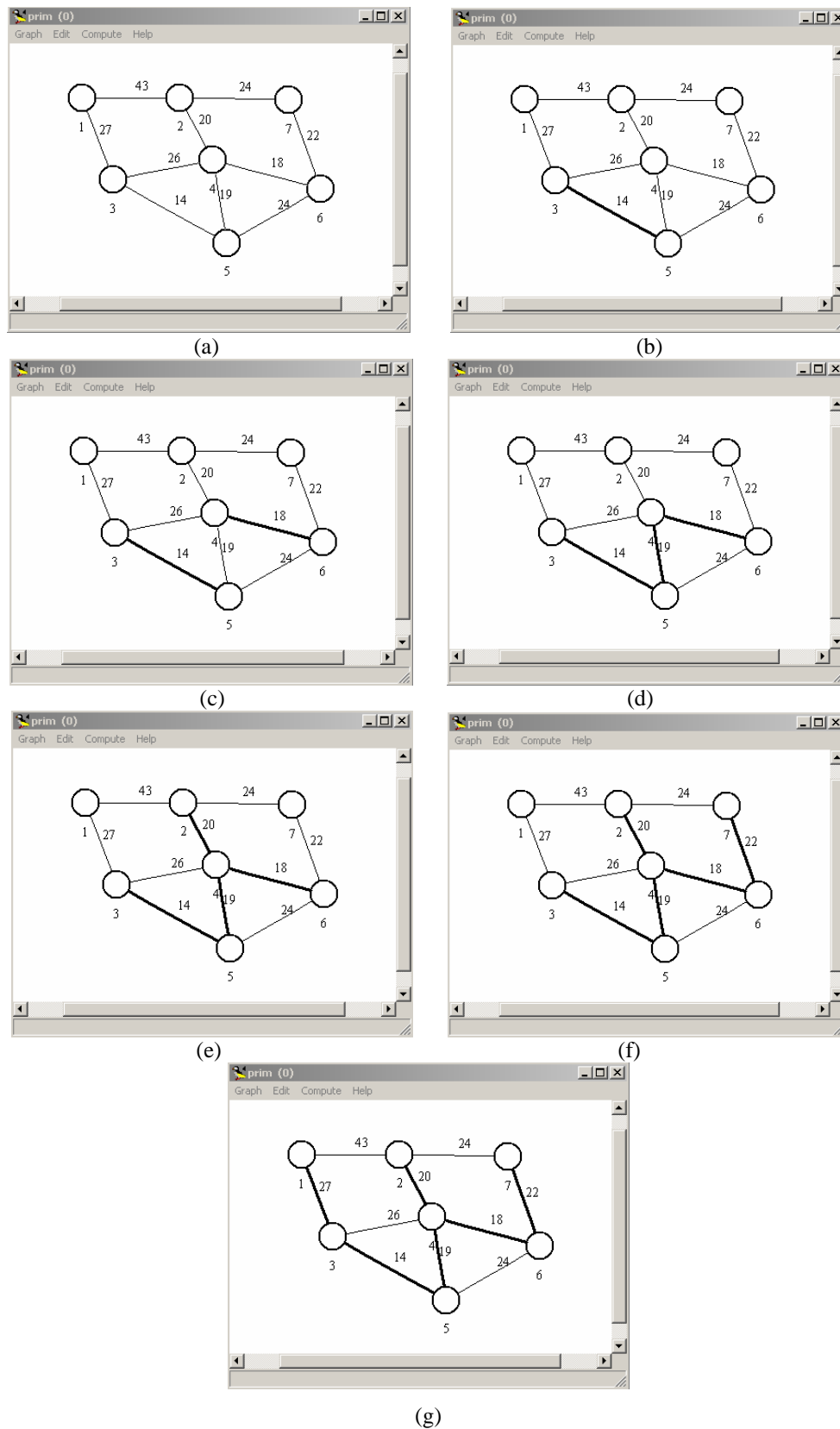
- Wizualizacja działania algorytmu Prima w Scilabie.



Rys. 7.4. Dodatkowy skrypt

Po uruchomieniu dodatkowego skryptu (który powoduje załadowanie grafu 'prima.graph' i funkcji 'prima.sci'.) z rysunku 7.4. pojawia się okienko z wiadomością która informuje nas o tym że każdorazowe wciśnięcie klawisza 'enter' będzie powodowało wytłuszczenie się kolejnych krawędzi tworzących minimalne drzewo rozpinające.

Na rysunku 7.5. a-g pokazano kolejne etapy tworzenia drzewa minimalnego.



Rys. 7.5. Działanie algorytmu Prima (rys. a-g)