STUDIA INFORMATICA 2011
Volume 32 Number 1A (94)

Marcin SZOŁTYSEK, Jacek WIDUCH
Politechnika Śląska, Instytut Informatyki

ALGORYTM DEWIACYJNY DLA PROBLEMU WYZNACZANIA *K* NAJKRÓTSZYCH ŚCIEŻEK

Streszczenie. Jednym z badanych problemów teorii grafów jest problem wyznaczenia najkrótszej ścieżki. Rozszerzeniem tego problemu jest problem wyznaczenia *K* (*K* > 1) najkrótszych ścieżek. Badane są dwa warianty tego problemu. W pierwszym dopuszcza się wyznaczenie ścieżek zawierających cykle, a w drugim są wyznaczane wyłącznie ścieżki acykliczne. W pracy został przedstawiony algorytm dewiacyjny, umożliwiający wyznaczenie *K* najkrótszych ścieżek, które mogą zawierać cykle.

Słowa kluczowe: skierowany graf ważony, ścieżka, podścieżka, waga ścieżki, najkrótsza ścieżka, *K* najkrótszych ścieżek, drzewo ścieżek, algorytm Dijkstry

DEVIATION ALGORITHM FOR SOLVING THE *K* SHORTEST PATH PROBLEM

Summary. One of the studied problems of the graph theory is the shortest path problem. An extension of this problem is the problem of finding *K* (*K* > 1) shortest paths. Two variants of this problem are studied. In the first case path containing cycles are allowed, and in the second are determined only loopless paths. In the paper a deviation algorithm for solving *K* shortest paths, which may contain cycles is shown.

Keywords: directed weighted graph, path, subpath, weight of path, shortest path, *K* shortest paths, tree of paths, Dijkstra algorithm

1. Wprowadzenie

Jednym z badanych zagadnień teorii grafów jest problem wyznaczenia najkrótszej ścieżki w grafie ważonym. Opracowanych zostało wiele algorytmów, umożliwiających jego rozwiązanie, a najbardziej znane są algorytmy Bellmana–Forda, Floyda–Warshalla, Dijkstry, Johnsona [3, 5, 8, 9, 11, 23]. Interpretacja ścieżki w grafie zależna jest od zastosowania grafu.

Przykładowo, jeżeli graf opisuje sieć dróg, gdzie wierzchołki reprezentują miasta lub skrzyżowania, a krawędzie (łuki) grafu reprezentują łączące je drogi, to najkrótsza ścieżka w grafie może opisywać drogę między dwoma miastami lub skrzyżowaniami o najmniejszej długości, koszcie lub czasie przejazdu.

Problem wyznaczenia najkrótszej ścieżki w grafie może być rozszerzony do problemu wyznaczenia K ($K > 1$) najkrótszych ścieżek. W przypadku zastosowania grafu do opisu sieci dróg wiedza uzyskana w ten sposób może okazać się niezbędna, w sytuacji gdy najkrótsza droga jest niedostępna (roboty drogowe, wypadek, zamknięcie z jakiegoś powodu pewnych odcinków drogi itp.) i zachodzi konieczność wyboru alternatywnej drogi. Znajomość kilku najkrótszych ścieżek w grafie umożliwia także porównanie alternatywnych rozwiązań i ewentualny wybór ścieżki nieznacznie dłużej niż najkrótsza, ale spełniającej inne kryteria, istotne z punktu widzenia użytkownika. Przykładowo, użytkownik może mieć informację odnośnie do korków występujących na pewnych odcinkach najkrótszej drogi pomiędzy dwoma miastami, dzięki czemu czas przejazdu tą drogą będzie dłuższy od czasu przejazdu drogą o większej długości, ale na której nie występują korki.

W literaturze są omawiane dwa warianty problemu wyznaczania K najkrótszych ścieżek. W pierwszym z nich są wyznaczane wyłącznie ścieżki acykliczne [10, 13, 16, 18, 22], natomiast w drugim dopuszcza się wyznaczenie ścieżek zawierających cykle [1, 2, 6, 7, 15, 17, 20]. W niniejszej pracy skupiono się nad drugim wariantem problemu i zaproponowano algorytm umożliwiający jego rozwiązanie. Na podstawie przeprowadzonych badań eksperymentalnych dokonano porównania zaproponowanego algorytmu z wybranymi istniejącymi rozwiązaniami.

2. Problem wyznaczania *K* najkrótszych ścieżek

Niech będzie dany skierowany ważony graf $G = (V, E, w)$, gdzie $V = \{v_1, \ldots, v_N\}$ jest zbiorem zawierającym N wierzchołków, $E = \{e_1, \ldots, e_M\}$ jest zbiorem zawierającym M łuków, a w jest funkcją wagową odwzorowującą zbiór łuków w zbiór liczb rzeczywistych nieujemnych, tj. $w : E \rightarrow R_0^+$ [4, 12, 14, 19, 21]. Każdy łuk $e_i \in E$, gdzie $i = 1, \ldots, M$, składa się z uporządkowanej pary wierzchołków $v_j, v_k \in V$, tj. $e_i = (v_j, v_k)$. Wierzchołek, z którego wychodzi łuk e_i jest oznaczony przez $tail(e_i)$, a przez $head(e_i)$ wierzchołek, do którego wchodzi łuk e_i. Tak więc dla łuku e_i zachodzą zależności: $tail(e_i) = v_j$ i $head(e_i) = v_k$.

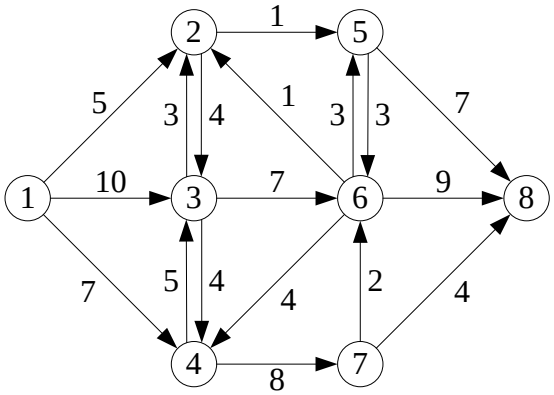
Ścieżką p z wierzchołka początkowego v_s do wierzchołka końcowego v_e jest nazywany ciąg wierzchołków i łuków, tj.: $p = \langle v_0; (v_0, v_1); v_1; (v_1, v_2); \ldots, v_{m-1}; (v_{m-1}, v_m); v_m \rangle$, przy czym $v_0 = v_s$, a $v_m = v_e$. Przez P zostanie oznaczony zbiór wszystkich ścieżek z wierzchołka v_s do

wierzchołka v_e. Wierzchołek v_j jest osiągalny z wierzchołka v_i, jeżeli istnieje ścieżka z wierzchołka v_i do wierzchołka v_j. Podścieżką ścieżki p jest ciąg kolejnych wierzchołków i łuków ścieżki p. Długość ścieżki jest równa liczbie łuków należących do ścieżki. Waga $W(p)$ ścieżki p jest równa sumie wag łuków należących do ścieżki, tj.:

$$W(p)=\sum_{i=1}^{m} w(v_{i-1}, v_i).$$

Najkrótszą ścieżką z wierzchołka v_s do wierzchołka v_e jest ścieżka o minimalnej wadze W. Niech wierzchołek v_i należy do najkrótszej ścieżki z v_s do v_e. Można wykazać, że ścieżka ta składa się z dwóch najkrótszych ścieżek: najkrótszej ścieżki z v_s do v_i i najkrótszej ścieżki z v_i do v_e [4]. Problem wyznaczenia K ($K > 1$) najkrótszych ścieżek polega na wyznaczeniu zbioru $P_K \subseteq P$, zawierającego K ścieżek, tj. $P_K = \{p_1, \ldots, p_K\}$. Ścieżki ze zbioru P_K mają następujące własności:

1. Waga ścieżki p_i nie jest większa od wagi ścieżki p_{i+1}, tj. $W(p_i) \leq W(p_{i+1})$; $i = 1, \ldots, K–1$.
2. Wagi wszystkich ścieżek, które nie należą do zbioru P_K, nie są mniejsze od wagi ścieżki p_K, tj. $\forall p \in P \setminus P_K\ W(p_K) \leq W(p)$.
3. Dopuszcza się występowanie cykli w ścieżkach.



Rys. 1. Przykładowy skierowany graf ważony
Fig. 1. A sample directed weighted graph

Tabela 1

Wyznaczone $K = 5$ najkrótszych ścieżek w grafie z rys. 1 wraz z ich wagami

Ścieżka	Waga ścieżki
$p_1 = \langle 1; (1, 2); 2; (2, 5); 5; (5, 8); 8\rangle$	$W(p_1) = 13$
$p_2 = \langle 1; (1, 2); 2; (2, 5); 5; (5, 6); 6; (6, 8); 8\rangle$	$W(p_2) = 18$
$p_3 = \langle 1; (1, 2); 2; (2, 5); 5; (5, 6); 6; (6, 2); 2; (2, 5); 5; (5, 8); 8\rangle$	$W(p_3) = 18$
$p_4 = \langle 1; (1, 2); 2; (2, 5); 5; (5, 6); 6; (6, 5); 5; (5, 8); 8\rangle$	$W(p_4) = 19$
$p_5 = \langle 1; (1, 4); 4; (4, 7); 7; (7, 8); 8\rangle$	$W(p_5) = 19$

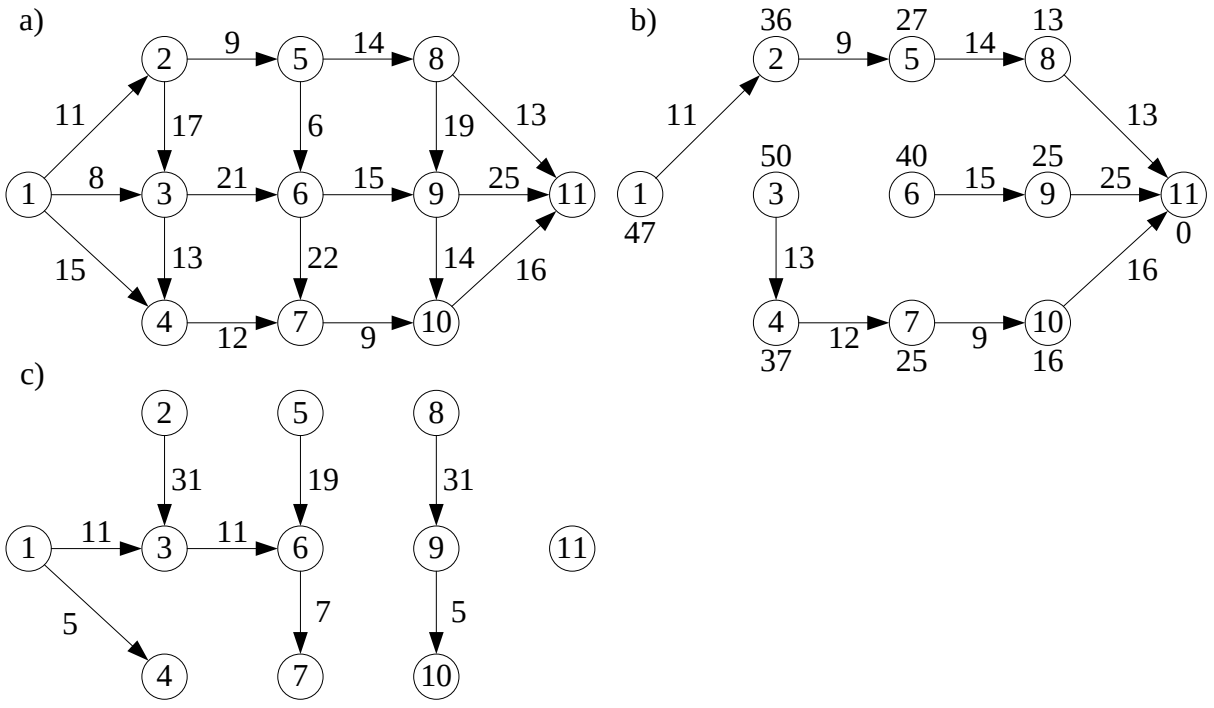
Problem wyznaczania $K = 5$ najkrótszych ścieżek zostanie zilustrowany na przykładzie wyznaczania ścieżek z wierzchołka $v_s = 1$ do wierzchołka $v_e = 8$ w grafie przedstawionym na

rys. 1. Wyznaczone ścieżki wraz z ich wagami przedstawiono w tabeli 1. Każda ścieżka, która nie należy do zbioru P_K, ma więc wagę nie mniejszą niż 19.

Wśród algorytmów wyznaczania K najkrótszych ścieżek można wyróżnić algorytmy odchyleniowe/dewiacyjne (ang. *deviation algorithms*) [7] i algorytmy etykietowania (ang. *labelling algorithms*) [1, 2, 6, 15, 17, 20]. Istotą algorytmów dewiacyjnych jest wyznaczenie drzewa najkrótszych ścieżek z wszystkich wierzchołków grafu do ustalonego wierzchołka końcowego v_e. Wszystkie łuki grafu, które nie należą do drzewa, są oznaczane jako stratne oraz jest obliczana strata w wadze ścieżki (zwiększenie wagi ścieżki), wynikająca z przejścia wzdłuż takiego łuku. Strata $\Delta w(v_i, v_j)$ dla łuku (v_i, v_j) jest równa:

$$\Delta w(v_i, v_j) = W_{\min}(p_j) - W_{\min}(p_i) + w(v_i, v_j) \tag{1}$$

gdzie $W_{\min}(p_j)$ jest wagą najkrótszej ścieżki z wierzchołka v_j do wierzchołka v_e, a $W_{\min}(p_i)$ jest wagą najkrótszej ścieżki z wierzchołka v_i do wierzchołka v_e. W ogólnym przypadku, mając dane drzewo ścieżek, łuki stratne oraz informację o stratach wprowadzanych przez nie, są podejmowane próby modyfikacji (dewiacji, odchylenia) najkrótszej ścieżki przez dodanie do niej łuków stratnych. Każda unikalna sekwencja łuków stratnych dodanych do ścieżki definiuje nową ścieżkę [7].



Rys. 2. Tworzenie drzewa najkrótszych ścieżek: a) skierowany graf ważony, b) drzewo najkrótszych ścieżek z wszystkich wierzchołków do wierzchołka końcowego $v_e = 11$, c) łuki nienależące do drzewa wraz z zaznaczonymi stratami

Fig. 2. A construction of shortest path tree: a) a weighted directed graph, b) a shortest path tree from all vertices to the final vertex $v_e = 11$, c) arcs not belonging to the tree with marked losses

Niech będzie dany graf (rys. 2a), w którym należy wyznaczyć *K* najkrótszych ścieżek z wierzchołka $v_s = 1$ do wierzchołka $v_e = 11$. Drzewo najkrótszych ścieżek z wszystkich wierzchołków grafu do wierzchołka $v_e = 11$ przedstawiono na rys. 2b, przy wierzchołkach zaznaczono wagi najkrótszych ścieżek. Łuki stratne, które nie należą do drzewa ścieżek, wraz ze stratami zostały przedstawione na rys. 2c. Rozpatrzmy przykładowo łuk (5, 6) o wadze $w(5, 6) = 6$. Istnieje możliwość utworzenia ścieżki p' z wierzchołka $v_s = 1$ do wierzchołka $v_e = 11$ zawierającej ten łuk. Ścieżka p' składa się z najkrótszej ścieżki z $v_s = 1$ do wierzchołka 5 (jest ona podścieżką najkrótszej ścieżki z $v_s = 1$ do $v_e = 11$), łuku (5, 6) oraz najkrótszej ścieżki z wierzchołka 6 do wierzchołka $v_e = 11$, tj.:

$$p' = \langle 1; (1, 2); 2; (2, 5); 5; (5, 6); 6; (6, 9); 9; (9, 11); 11\rangle.$$

Waga $W(p')$ jest większa o wartość $\Delta w(5, 6)$ od wagi najkrótszej ścieżki z $v_s = 1$ do $v_e = 11$, która jest równa 47. Wagi najkrótszych ścieżek z wierzchołków 5 i 6 do wierzchołka $v_e = 11$ są równe odpowiednio $W_{\min}(5) = 27$ i $W_{\min}(6) = 40$. Tak więc na podstawie zależności (1) strata łuku (5, 6) wynosi $\Delta w(5, 6) = 19$, a waga ścieżki p' jest równa $W(p') = 66$.

W algorytmach etykietowania podczas wyznaczania rozwiązań nie korzysta się z drzewa najkrótszych ścieżek. Ich głównym elementem jest odwiedzanie poszczególnych wierzchołków grafu i zapisywanie etykiet w wektorze o rozmiarze *K*, skojarzonym z danym wierzchołkiem v_i. Etykieta zawiera długość ścieżki z wierzchołka początkowego v_s do wierzchołka v_i. Wektor zawiera więc długości *K* najkrótszych wyznaczonych do danej chwili ścieżek. Algorytmy etykietowania można podzielić na dwie grupy: algorytmy korygowania (ang. *label correcting algorithms*) [20] oraz algorytmy nadawania etykiet (ang. *label setting algorithms*) [1, 2, 6, 15, 17].

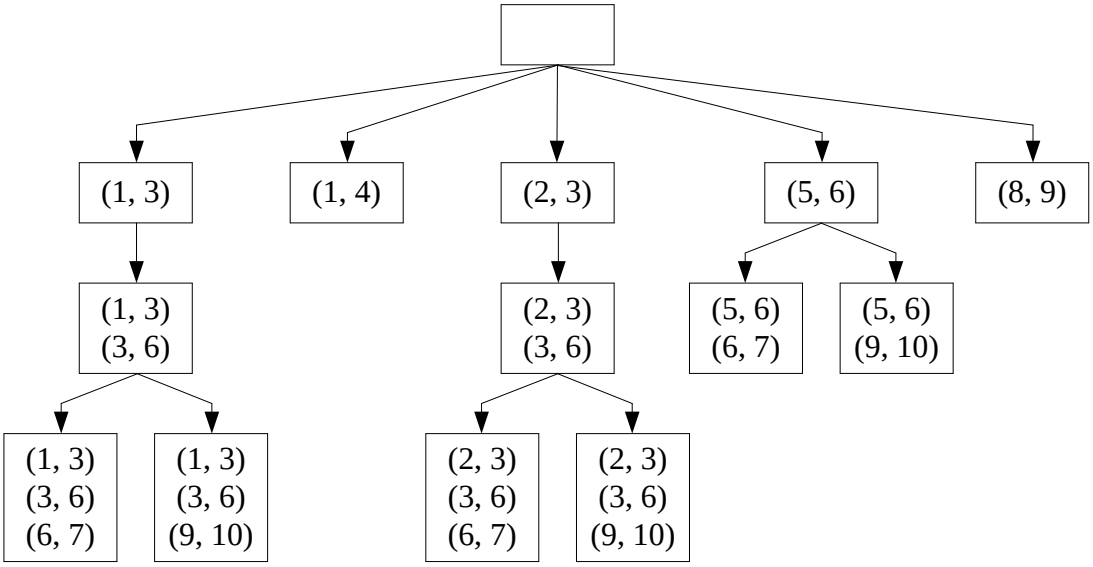
3. Opis dewiacyjnego algorytmu wyznaczania *K* najkrótszych ścieżek

3.1. Ogólne założenia

W zaproponowanym algorytmie podczas wyznaczania rozwiązań, tj. *K* najkrótszych ścieżek z wierzchołka początkowego v_s do wierzchołka końcowego v_e, korzysta się z drzewa najkrótszych ścieżek z wszystkich wierzchołków grafu do wierzchołka v_e, którego ogólna koncepcja została przedstawiona w pracy [7][1]. Na podstawie drzewa najkrótszych ścieżek następuje próba modyfikacji najkrótszej ścieżki z v_s do v_e przez dodanie do niej łuku stratnego, co powoduje przejście do nowego wierzchołka v_j. Z wierzchołka v_j w kierunku wierzchołka v_e można podążać wzdłuż najkrótszej ścieżki lub dodać kolejny łuk stratny.

[1] W pracy [7] przedstawiono wyłącznie ogólny opis drzewa najkrótszych ścieżek bez algorytmu jego wyznaczania.

Sekwencja łuków stratnych dodanych do ścieżki definiuje nową ścieżkę z wierzchołka v_s do wierzchołka v_e, a wszystkie tak utworzone ścieżki można przedstawić w postaci drzewa. Każdy węzeł takiego drzewa zawiera unikalny ciąg łuków stratnych dodawanych do ścieżki, co oznacza, że reprezentuje on w sposób jednoznaczny nową ścieżkę z wierzchołka v_s do wierzchołka v_e. Korzeń drzewa zawiera ciąg pusty, co może być interpretowane jako reprezentacja najkrótszej ścieżki. Wszystkie węzły na poziomie *i*–tym reprezentują ścieżki zawierające *i* łuków stratnych, przy czym na poziomie 0 znajduje sie korzeń drzewa.



Rys. 3. Drzewo zawierające ścieżki opisane łukami stratnymi
Fig. 3. A tree containing paths described by arcs of loss

Na rysunku 3 zostało przedstawione drzewo zawierające wszystkie ścieżki z wierzchołka początkowego $v_s = 1$ do wierzchołka końcowego $v_e = 11$ w grafie z rys. 2a, jakie można utworzyć na podstawie drzewa najkrótszych ścieżek (rys. 2b) i łuków stratnych (rys. 2c). Przykładowo, ścieżka *p* zawierająca łuki stratne (5, 6) i (9, 10) powstaje z trzech najkrótszych ścieżek: z wierzchołka $v_s = 1$ do wierzchołka 5, z wierzchołka 6 do wierzchołka 9, z wierzchołka 10 do wierzchołka końcowego $v_e = 11$, które są połączone tymi łukami. Najkrótsze ścieżki są odczytywane z drzewa najkrótszych ścieżek. Ścieżka *p* ma postać:

$$p = \langle 1; (1, 2); 2; (2, 5); 5; (5, 6); 6; (6, 9); 9; (9, 10); 10; (10, 11); 11\rangle.$$

Wyznaczanie *K* najkrótszych ścieżek odbywa się w trzech etapach:

- wyznaczenie drzewa najkrótszych ścieżek z wszystkich wierzchołków grafu do wierzchołka końcowego v_e oraz wyznaczenie łuków stratnych i określenie strat dla tych łuków,
- wyznaczenie *K* najkrótszych ścieżek na podstawie drzewa najkrótszych ścieżek i łuków stratnych,
- odtworzenie wyznaczonych *K* najkrótszych ścieżek.

3.2. Wyznaczenie drzewa najkrótszych ścieżek oraz łuków stratnych

Drzewo najkrótszych ścieżek z wszystkich wierzchołków grafu do wierzchołka końcowego v_e jest wyznaczane za pomocą zmodyfikowanego algorytmu Dijkstry. Podczas jego wyznaczania dla każdego łuku należy stwierdzić, czy należy on do drzewa lub jest łukiem stratnym i wyznaczyć stratę dla takiego łuku zgodnie z zależnością (1). Jak się okazuje, nie wszystkie łuki stratne muszą być analizowane w kolejnych etapach algorytmu, gdyż część z nich może zostać usunięta z grafu i niepoddawana dalszej analizie. Z grafu mogą zostać usunięte także niektóre wierzchołki. Wierzchołek v_i może zostać usunięty z grafu, jeżeli nie jest z niego osiągalny wierzchołek końcowy v_e. W przypadku usunięcia z grafu wierzchołka v_i są usuwane wszystkie łuki wchodzące do tego wierzchołka oraz wychodzące z niego.

Drzewo najkrótszych ścieżek może zostać wyznaczone z użyciem algorytmu Dijkstry przez wprowadzenie prostych modyfikacji w algorytmie. Modyfikacja polega na zmianie sposobu analizy łuków i wierzchołków grafu. Nie będą analizowane łuki wychodzące z wierzchołków, tylko łuki wchodzące, a analiza wierzchołków zostanie rozpoczęta od wierzchołka końcowego v_e. W momencie ustalenia minimalnej odległości dla wierzchołka v_i, będącej minimalną odległością wierzchołka v_i od wierzchołka v_e, zostanie dokonana relaksacja wszystkich łuków wchodzących do wierzchołka v_i. Dzięki temu zostaną wyznaczone najkrótsze ścieżki z wszystkich wierzchołków grafu do wierzchołka końcowego v_e, a w efekcie zostanie wyznaczone szukane drzewo najkrótszych ścieżek[2].

W	Δw	*arc*

Rys. 4. Zawartość rekordu skojarzonego z wierzchołkiem v_i podczas wyznaczania drzewa najkrótszych ścieżek

Fig. 4. A contents of a record associated with the vertex v_i during the computation of a shortest path tree

W algorytmie Dijkstry z każdym wierzchołkiem v_i grafu są skojarzone dwie wartości: waga najkrótszej ścieżki z wierzchołka v_s oraz wierzchołek poprzedzający w najkrótszej ścieżce wierzchołek v_i. Pamiętanie wymienionych wartości umożliwia wyłącznie wyznaczenie drzewa najkrótszych ścieżek, natomiast nie pozwala na wyznaczenie łuków stratnych i ich strat. Stąd też wprowadzono modyfikację polegającą na skojarzeniu z każdym wierzchołkiem v_i listy rekordów zawierających następujące wartości (rys. 4):

- łuk *arc* wychodzący z wierzchołka v_i,
- wagę *W* ścieżki zawierającej łuk *arc*,
- stratę Δw w wadze ścieżki wynikającą z włączenia do ścieżki łuku *arc*.

[2] W algorytmie Dijkstry analiza wierzchołków jest rozpoczynana od wierzchołka początkowego v_s, a dla wierzchołka v_i, dla którego ustalono minimalną odległość z wierzchołka v_s, jest dokonywana relaksacja łuków wychodzących z niego, w efekcie czego jest wyznaczane drzewo najkrótszych ścieżek z wierzchołka v_s do pozostałych wierzchołków grafu.

Rekordy w listach opisują łuki drzewa najkrótszych ścieżek oraz łuki stratne. Pierwszym rekordem w liście wierzchołka v_i jest rekord zawierający opis łuku (v_i, v_j), należącego do najkrótszej ścieżki z wierzchołka v_i do wierzchołka v_e, a więc jest to rekord o minimalnej wadze W, a strata dla tego łuku jest równa 0, tj. $\Delta w = 0$.

Wyznaczanie drzewa najkrótszych ścieżek oraz łuków stratnych po wprowadzeniu opisanych modyfikacji w algorytmie Dijkstry jest realizowane przez procedurę *ComputeShortestPathTree* o następującej treści:

```
   Wejście: G = (V, E, w) - skierowany graf ważony z funkcją wagową w
            ve            - wierzchołek końcowy
   Wyjście: T             - listy rekordów opisujących łuki drzewa najkrótszych
                            ścieżek i łuki stratne
 1: procedure ComputeShortestPathTree(G, ve, T)
 2: begin
 3:   Q := ∅;                  // pusta kolejka zawierająca wierzchołki
 4:   for all vi ∈ V do
 5:     T[vi] := ∅;            // pusta lista rekordów dla wierzchołka vi
 6:   end for
 7:   r.W := 0;
 8:   r.Δw := 0;
 9:   r.arc := ∅;
10:   T[ve].Insert(r);         // wstawienie rekordu r do listy rekordów T[ve]
11:   Q.Insert(ve);            // wstawienie wierzchołka ve do kolejki Q
12:   while Q ≠ ∅ do
13:     vi := Q.Pop();         // pobranie pierwszego wierzchołka z kolejki
14:     for all (vj, vi) ∈ in[vi] do  // dla wszystkich łuków wchodzących do vi
15:       r.arc := (vj, vi);               // łuk z vj do vi
16:       r.W := T[vi][0].W + w(vj, vi);   // waga ścieżki z vj do ve
17:       T[vj].Insert(r);     // wstawienie rekordu r do listy T[vj]
18:       Q.Insert(vj);        // wstawienie wierzchołka vj do kolejki Q
19:     end for
20:   end while
21:   for all vi ∈ V do
22:     if T[vi] ≠ ∅ then         // wyznaczono najkrótszą ścieżkę z vi do ve
23:       for all r ∈ T[vi] do    // wyznaczenie strat dla łuków w liście T[vi]
24:         r.Δw := r.W - T[vi][0].W;   // wyznaczenie straty dla łuku r.arc
25:       end for
26:     end if
27:   end for
28:   return T;
29: end
```

Drzewo najkrótszych ścieżek wyznaczone w procedurze *ComputeShortestPathTree* jest zapisane niejawnie w postaci rekordów opisujących łuki (rys. 4). Z każdym wierzchołkiem v_i jest skojarzona lista rekordów $T[v_i]$, gdzie pierwszy rekord opisuje łuk należący do drzewa najkrótszych ścieżek, natomiast kolejne rekordy opisują łuki stratne. W zapisie algorytmu rekordy w listach są indeksowane, przy czym pierwszy rekord listy ma indeks 0. Przed rozpoczęciem wyznaczania drzewa listy są zerowane (wiersze 4–6), wyjątkiem jest lista dla wierzchołka końcowego v_e, do której jest wstawiany rekord inicjalizacyjny (wiersz 10). Rekord ten nie zawiera żadnego łuku *arc* (wiersz 9), natomiast waga W ścieżki oraz strata Δw są równe 0 (wiersze 7–8). Podczas wyznaczania drzewa jest użyta kolejka priorytetowa Q

zawierająca wierzchołki, dla których wyznaczono ścieżkę do wierzchołka końcowego v_e. Wierzchołki w kolejce są uporządkowane niemalejąco według wagi najkrótszej z dotychczas wyznaczonych ścieżek do wierzchołka v_e, tj. według wartości W pierwszego rekordu znajdującego się w liście T danego wierzchołka. Początkowo kolejka zawiera wierzchołek końcowy v_e (wiersz 11).

Wyznaczanie drzewa najkrótszych ścieżek i łuków stratnych odbywa się w iteracji **while** (wiersze 12–20), której wykonanie jest przerywane, w chwili kiedy kolejka Q nie zawiera żadnego wierzchołka. W każdej iteracji jest pobierany z kolejki pierwszy wierzchołek v_i (wiersz 13) i zgodnie z działaniem algorytmu Dijkstry dla wierzchołka tego zostaje ustalona waga najkrótszej ścieżki do wierzchołka v_e. W kolejnym kroku są analizowane wszystkie łuki (v_j, v_i) wchodzące do wierzchołka v_i (wiersze 14–19). Dla każdego łuku jest tworzony rekord r, do którego jest zapisywana waga ścieżki z wierzchołka v_j do wierzchołka v_e, zawierająca łuk (v_j, v_i) (wiersz 16) oraz jest zapisywany łuk (v_j, v_i) (wiersz 15). Utworzony rekord jest umieszczany w liście rekordów wierzchołka v_j, tj. $T[v_j]$ (wiersz 17), a wierzchołek v_j jest umieszczany w kolejce Q (wiersz 18)[3].

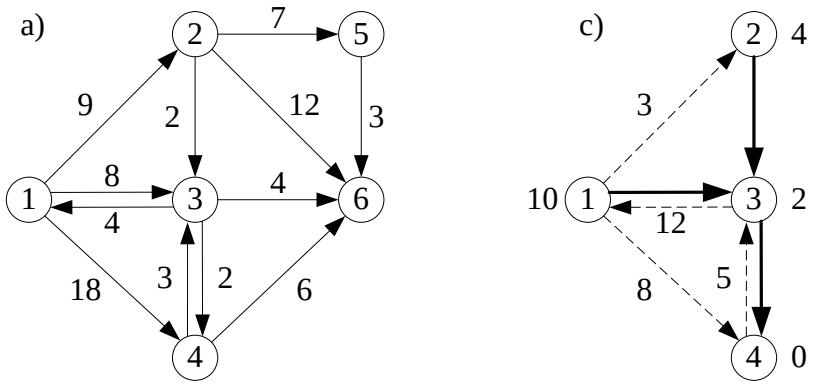
Po wyznaczeniu drzewa najkrótszych ścieżek oraz łuków stratnych następuje wyznaczenie strat dla łuków (wiersze 21–27). Straty są wyznaczane jedynie dla łuków (v_j, v_i), dla których został utworzony rekord w liście $T[v_j]$, a nie dla wszystkich łuków stratnych. Obecność rekordu w liście oznacza, że istnieje ścieżka z wierzchołka v_j do wierzchołka v_e oraz została wyznaczona najkrótsza ścieżka między tymi wierzchołkami. Strata dla łuku jest równa różnicy między wagą ścieżki, zawierającej ten łuk, a wagą najkrótszej ścieżki (wiersz 24).

Jako wynik wykonania procedury *ComputeShortestPathTree* są zwracane listy utworzonych rekordów. W kolejnych etapach algorytmu będą analizowane wyłącznie łuki, dla których został utworzony rekord oraz wierzchołki, z których istnieje ścieżka do wierzchołka końcowego.

Jako przykład wykonania procedury *ComputeShortestPathTree* zostanie rozpatrzone wyznaczenie w grafie (rys. 5a) drzewa najkrótszych ścieżek z wszystkich wierzchołków grafu do wierzchołka końcowego $v_e = 4$ oraz łuków stratnych. Listy utworzonych rekordów przedstawiono na rys. 5b. Dla wierzchołków 5 i 6 nie został utworzony żaden rekord, ponieważ z wierzchołków tych nie jest osiągalny wierzchołek końcowy, stąd w kolejnych etapach algorytmu wierzchołki te nie będą analizowane oraz nie będą analizowane łuki wychodzące z tych wierzchołków i wchodzące do nich. Warto zwrócić uwagę na rekord dla łuku (3, 1). Włączenie go do ścieżki odpowiada utworzeniu cyklu ⟨3; (3, 1); 1; (1, 3); 3⟩ o wadze równej 12, stąd strata dla tego łuku wynosi 12, a waga utworzonej w ten sposób ścieżki ⟨3; (3, 1); 1; (1, 3); 3; (3, 4); 4⟩ jest równa 14. Rysunek 5c zawiera graficzne przedsta-

[3] Jeżeli kolejka Q zawiera wierzchołek v_j, to nie jest on ponownie wstawiany.

wienie drzewa najkrótszych ścieżek oraz łuki stratne, dla których zostały utworzone rekordy. Łuki należące do drzewa ścieżek zaznaczono w sposób pogrubiony, a łuki stratne strzałką przerywaną. Dodatkowo, zostały zaznaczone straty dla łuków stratnych.



b)

v_j	$T[v_j]$ (listy rekordów opisujących łuki)								
1	10	0	(1, 3)	13	3	(1, 2)	18	8	(1, 4)
2	4	0	(2, 3)						
3	2	0	(3, 4)	14	12	(3, 1)			
4	0	0	-	5	5	(4, 3)			
5									
6									

Rys. 5. Ilustracja wykonania procedury *ComputeShortestPathTree*: a) skierowany graf ważony, b) listy rekordów opisujących łuki drzewa najkrótszych ścieżek oraz łuków stratnych, c) drzewo najkrótszych ścieżek oraz łuki stratne wraz z zaznaczonymi stratami

Fig. 5. An illustration of the *ComputeShortestPathTree* procedure execution: a) a weighted directed graph, b) lists of records described arcs of a shortest path tree and arcs of loss, c) a shortest path tree and arcs of loss with marked losses

3.3. Wyznaczenie *K* najkrótszych ścieżek na podstawie drzewa najkrótszych ścieżek i łuków stratnych

W celu wyznaczenia *K* najkrótszych ścieżek są analizowane łuki grafu, dla których zostały wyznaczone rekordy za pomocą procedury *ComputeShortestPathTree*. Rekordy reprezentują łuki stratne oraz łuki należące do drzewa najkrótszych ścieżek, które można nazwać łukami bezstratnymi. Analizowanie łuków jest analogiczne do przeszukiwania drzewa ścieżek, którego koncepcja została opisana w punkcie 3.1 i przedstawiona na rys. 3. W rzeczywistości drzewo nie jest budowane, a celem analizy łuków jest utworzenie unikalnych sekwencji łuków stratnych opisujących ścieżki w sposób niejawny. Analiza jest rozpoczynana z wierzchołka początkowego v_s, a strategia analizy jest połączeniem przeszukiwania grafu wszerz i przeszukiwania w głąb [4, 14]. Podczas analizy łuków są tworzone rekordy,

z których każdy opisuje aktualnie przeanalizowaną ścieżkę z wierzchołka początkowego do danego wierzchołka v_i i zawiera następujące wartości (rys. 6):

- wierzchołek v_i, do którego została przeanalizowana ścieżka z wierzchołka początkowego,
- listę *arcs* zawierającą łuki stratne należące do ścieżki,
- wagę *W* ścieżki z wierzchołka początkowego do wierzchołka końcowego, zawierającą łuki stratne *arcs*,
- stratę Δw w wadze ścieżki wynikającą z włączenia do ścieżki łuków stratnych *arcs*.

Rekordy opisujące ścieżki są przechowywane w liście *P*, która zawiera maksymalnie *K* rekordów, a rekordy są uporządkowane niemalejąco według wagi *W* ścieżki. W przypadku dodania nowego rekordu do listy, która zawiera już *K* rekordów, z listy jest usuwany rekord o największej wadze *W*. Przed rozpoczęciem analizy łuków grafu jest tworzony rekord odpowiadający najkrótszej ścieżce z wierzchołka v_s do wierzchołka v_e i jest on wstawiany do listy. Rekord ten zawiera pustą listę *arcs*, wagę *W* najkrótszej ścieżki z wierzchołka v_s do wierzchołka v_e, stratę $\Delta w = 0$ oraz wierzchołek $v_i = v_s$. Na jego podstawie będzie podejmowana próba tworzenia kolejnych ścieżek przez rozszerzenie ścieżki o łuki stratne. Analiza łuków jest wykonywana iteracyjnie. W każdej iteracji są analizowane łuki wychodzące z wierzchołka v_i, znajdującego się w aktualnie analizowanym rekordzie opisującym ścieżki, co przypomina przeszukiwanie grafu wszerz. Jako pierwsze zostaną więc przeanalizowane łuki wychodzące z wierzchołka v_s.

W przypadku analizy łuku stratnego (v_i, v_j) jest tworzony nowy rekord, gdyż odpowiada to utworzeniu nowej ścieżki przez dodanie tego łuku do ścieżki reprezentowanej przez aktualnie badany rekord. W nowo utworzonym rekordzie w liście łuków *arcs* są umieszczane wszystkie łuki stratne z listy *arcs* aktualnie analizowanego rekordu oraz łuk (v_i, v_j), a v_j jest zapamiętywany jako wierzchołek, do którego została przeanalizowana ścieżka z wierzchołka początkowego. Waga *W* ścieżki oraz strata Δw w wadze ścieżki, wynikająca z dodania do ścieżki łuku (v_i, v_j), są przepisywane z rekordu opisującego łuk (v_i, v_j) do nowo utworzonego rekordu opisującego ścieżkę. Nowo utworzony rekord jest wstawiany do listy rekordów *P*.

Analiza bezstratnego łuku (v_i, v_j) nie powoduje utworzenia nowego rekordu reprezentującego ścieżkę. W takim przypadku w aktualnie analizowanym rekordzie reprezentującym ścieżkę jest dokonywane zapamiętanie wierzchołka v_j, do którego została przeanalizowana ścieżka z wierzchołka początkowego. Ponadto, w kolejnej iteracji będzie kontynuowana analiza bieżącego rekordu, a nie kolejnego rekordu znajdującego się w liście *P*. Taki sposób postępowania, przypominający przeszukiwanie grafu w głąb, wynika z tego, że kolejne rekordy znajdujące się w liście *P* reprezentują ścieżki o nie mniejszej stracie w wadze ścieżki niż ścieżka reprezentowana przez aktualnie analizowany rekord. Zatem, jeśli jest możliwość przejścia do kolejnego wierzchołka v_j za pomocą łuku bezstratnego (v_i, v_j), jest uzasadnione

wykonanie takiego przejścia i zbadanie możliwości utworzenia nowych ścieżek przez dołączenie do ścieżki łuków wychodzących z wierzchołka v_j. Tym sposobem, przed rozpoczęciem analizy kolejnego rekordu w liście P, nastąpi dojście po łukach bezstratnych do wierzchołka końcowego v_e, co wiąże się z wyznaczeniem najkrótszej ścieżki z wierzchołka v_s do wierzchołka v_e. Jednocześnie zostanie dokonana analiza wszystkich wierzchołków sąsiadujących z wierzchołkami należącymi do tej ścieżki, oraz zostaną utworzone dla nich rekordy, które będą wstawione do listy P. Dopiero po stwierdzeniu, że z wierzchołka v_e nie wychodzi żaden łuk bezstratny, zostanie dokonana analiza kolejnego rekordu w liście P i próba utworzenia na jego podstawie nowych ścieżek. Zakończenie obliczeń następuje po przeanalizowaniu wszystkich rekordów w liście P.

Wyznaczenie K najkrótszych ścieżek odbywa się w procedurze *ComputeKShortestPath* o następującej treści:

```
   Wejście: T   - listy rekordów opisujących łuki drzewa najkrótszych
                  ścieżek i łuki stratne
            vs     - wierzchołek początkowy
            K   - liczba najkrótszych ścieżek, które należy wyznaczyć
   Wyjście: P   - lista rekordów opisujących K najkrótszych ścieżek
 1: procedure ComputeKShortestPath(T, vs, K, P)
 2: begin
 3:   P := ∅;          // pusta lista zawierająca rekordy opisujące ścieżki
 4:   rp.arcs := ∅;
 5:   rp.W := T[vs][0].W;
 6:   rp.Δw := 0;
 7:   rp.vi := vs;
 8:   P.Insert(rp);     // wstawienie rekordu rp do listy P
 9:   i := 0;           // indeks aktualnie analizowanego rekordu w liście P
10:   finish := false;
11:   while not finish do
12:     vi := P[i].vi;         // aktualnie analizowany wierzchołek
13:     all := true;
14:     for all ra ∈ T[vi] do  // dla wszystkich rekordów w liście wierzchołka vi
15:       if ra ≠ T[vi][0] then   // ra reprezentuje łuk stratny
16:         rp.arcs := P[i].arcs ⊕ ra.arc;
17:         rp.W := ra.W;
18:         rp.Δw := ra.Δw;
19:         rp.vi := head(ra.arc);
20:         P.Insert(rp);      // wstawienie rekordu rp do listy P
21:       else                 // ra reprezentuje łuk bezstratny
22:         P[i].vi := head(ra.arc);
23:         all := false;
24:       end if
25:     end for
26:     if all then      // nie analizowano łuku bezstratnego dla wierzchołka vi
27:       i := i + 1;          // przejście do następnego rekordu w liście P
28:       if i >= count(P) then             // osiągnięto koniec listy rekordów
29:         finish := true;                 // zakończenie obliczeń
30:       end if
31:     end if
32:   end while
33:   return P;
34: end
```

W części inicjalizacyjnej (wiersze 3–10) jest tworzony rekord r_p (wiersze 4–7) odpowiadający najkrótszej ścieżce z wierzchołka początkowego do wierzchołka końcowego i jest on wstawiany do listy *P* (wiersz 8). W zapisie algorytmu przyjmuje się, że dostęp do rekordów listy odbywa się za pomocą indeksu *i*, przy czym pierwszy rekord ma indeks równy 0. W części właściwej procedury, jaką jest iteracja **while** (wiersze 11–32), następuje wyznaczenie *K* najkrótszych ścieżek na podstawie rekordu utworzonego w części inicjalizacyjnej i rekordów opisujących łuki, znajdujących się w strukturze *T*. W każdej iteracji są analizowane łuki wychodzące z wierzchołka v_i reprezentowane przez rekordy r_a (wiersze 14–25), gdzie wierzchołek v_i jest wierzchołkiem zapisanym w aktualnie analizowanym rekordzie listy *P* (wiersz 12). Podczas analizy został użyty znacznik *all* informujący o tym, czy z wierzchołka v_i nastąpiło przejście po łuku bezstratnym, a co za tym idzie, czy w kolejnej iteracji należy kontynuować analizę bieżącego rekordu.

arcs	*W*	Δw	v_i

Rys. 6. Zawartość rekordu opisującego ścieżkę z wierzchołka początkowego v_s do wierzchołka v_i
Fig. 6. A contents of a record described a path from the start vertex v_s to the vertex v_i

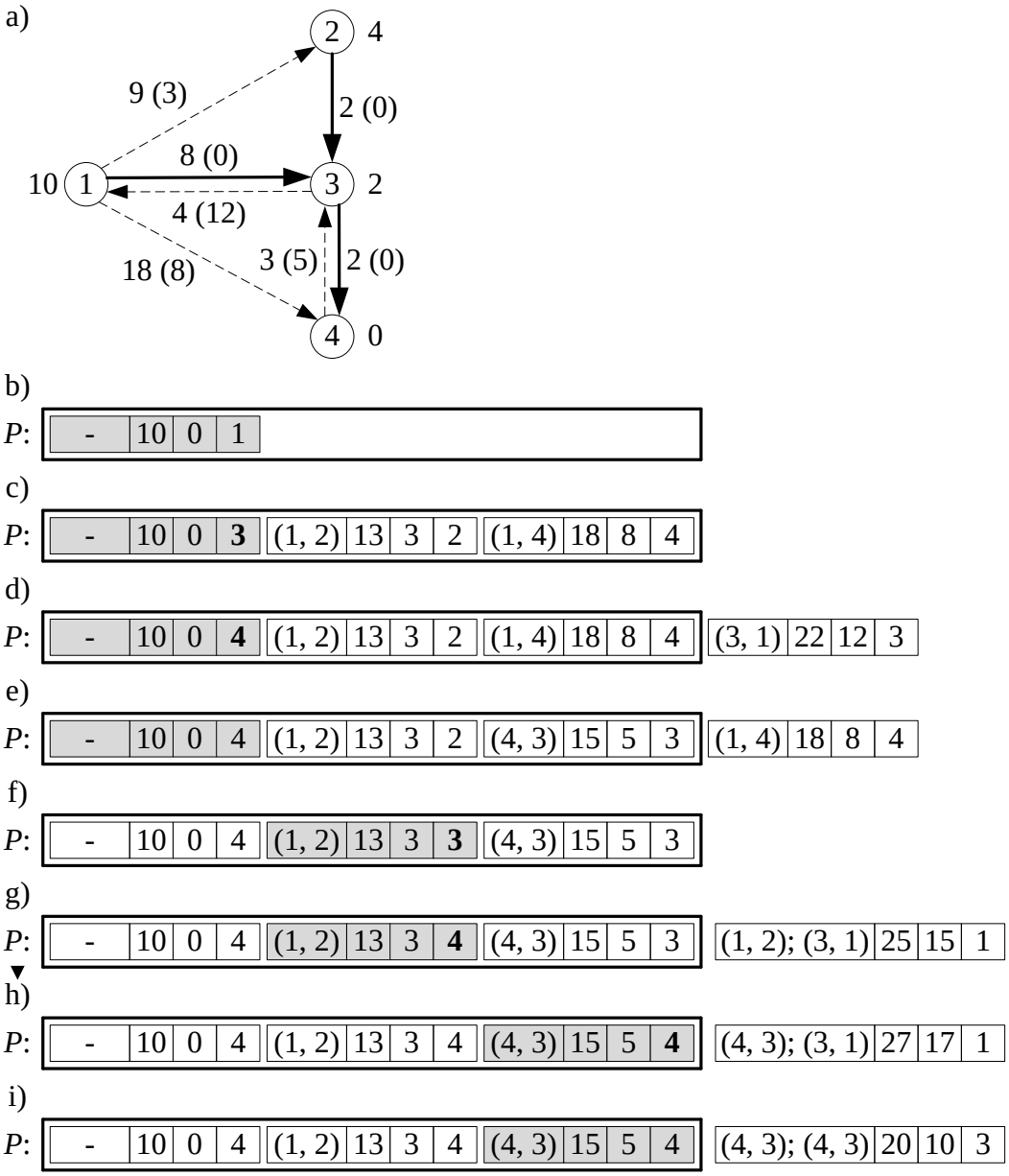
W przypadku analizy łuku bezstratnego następuje aktualizacja wierzchołka v_i w bieżącym rekordzie *P*[*i*] oraz zmiana wartości znacznika *all* (wiersze 22–23). Jeżeli jest analizowany łuk stratny, to jest tworzony nowy rekord r_p reprezentujący nową ścieżkę i jest on wstawiany do listy *P* (wiersze 16–20). Zapis $P[i].arcs \oplus r_a.arc$ (wiersz 16) należy rozumieć jako utworzenie listy łuków będącej kopią listy $P[i].arcs$ oraz dodanego na jej końcu łuku $r_a.arc$.

Przejście do analizy kolejnego rekordu listy *P* (wiersz 27) następuje wyłącznie w przypadku, kiedy dla wierzchołka v_i nie był analizowany łuk bezstratny. Zakończenie obliczeń, przez nadanie znacznikowi *finish* wartości **true** (wiersz 29), następuje po przeanalizowaniu wszystkich rekordów w liście *P* (*count*(*P*) oznacza liczbę rekordów w liście *P*). Wynikiem działania procedury jest lista rekordów opisujących ścieżki w sposób niejawny za pomocą sekwencji łuków stratnych.

Działanie procedury *ComputeKShortestPath* zostanie prześledzone na przykładzie wyznaczania w grafie (rys. 5a) $K = 3$ najkrótszych ścieżek z wierzchołka $v_s = 1$ do wierzchołka $v_e = 4$ na podstawie wyznaczonych list rekordów opisujących łuki grafu, przedstawionych na rys. 5b. Na rysunku 7a w sposób pogrubiony przedstawiono łuki należące do drzewa najkrótszych ścieżek, a strzałką przerywaną łuki stratne. Przy łukach zaznaczono wagę łuku oraz w nawiasie stratę, a przy wierzchołkach wagę najkrótszej ścieżki do wierzchołka $v_e = 4$. Na rysunkach 7b–7i przedstawiono kolejne iteracje wyznaczania rekordów opisujących $K = 3$ najkrótszych ścieżek, kolorem szarym zaznaczono aktualnie analizowany rekord.

W części inicjalizacyjnej jest tworzony rekord odpowiadający najkrótszej ścieżce (rys. 7b), jest on wstawiany do listy *P* i staje się rekordem, od którego rozpocznie się

wyznaczanie kolejnych rekordów opisujących ścieżki. Jako pierwsze są analizowane łuki wychodzące z wierzchołka $v_i = 1$, w efekcie dla łuków stratnych (1, 2) i (1, 4) są tworzone nowe rekordy, a dla łuku bezstratnego (1, 3) następuje jedynie aktualizacja wierzchołka v_i w bieżącym rekordzie (rys. 7c).



Rys. 7. Ilustracja wykonania procedury *ComputeKShortestPath*: a) drzewo najkrótszych ścieżek oraz łuki stratne, b) – i) kolejne iteracje wyznaczania rekordów opisujących $K = 3$ najkrótszych ścieżek

Fig. 7. An illustration of the *ComputeKShortestPath* procedure execution: a) a shortest path tree and arcs of loss, b) – i) a successive iterations of calculating records described $K = 3$

W kolejnej iteracji analizie są poddawane łuki wychodzące z wierzchołka $v_i = 3$, dla łuku bezstratnego (3, 4) następuje aktualizacja wierzchołka v_i w bieżącym rekordzie, a dla łuku stratnego (3, 1) jest tworzony nowy rekord (rys. 7d). Rekord ten nie jest dodawany do listy, ponieważ opisuje ścieżkę o wadze $W = 22$, która jest większa od dotychczas wyznaczonych

trzech ścieżek. Kolejnym analizowanym wierzchołkiem jest $v_i = 4$, z którego wychodzi jeden łuk (4, 3) będący łukiem stratnym. Dla łuku tego jest tworzony nowy rekord, który „wypiera" z listy rekord dla łuku (1, 4) (rys. 7e). Z wierzchołka $v_i = 4$ nie wychodzi łuk bezstratny, dlatego nastąpi analiza kolejnego rekordu w liście *P*. Rekord ten oraz następny są analizowane w podobny sposób, w efekcie czego zostaną utworzone trzy rekordy opisujące $K = 3$ najkrótsze ścieżki, których wagi są równe odpowiednio: $W_1 = 10$, $W_2 = 13$ i $W_3 = 15$ (rys. 7i). Pierwsza ścieżka jest ścieżką najkrótszą, nie zawiera więc łuku stratnego, natomiast dwie kolejne zawierają jeden łuk stratny.

3.4. Odtworzenie *K* najkrótszych ścieżek

Rekordy wyznaczane za pomocą procedury *ComputeKShortestPath* opisują ścieżki w sposób niejawny za pomocą sekwencji łuków stratnych. W celu odtworzenia pełnej ścieżki należy ją uzupełnić brakującymi wierzchołkami oraz łukami bezstratnymi. Operacja ta jest wykonywana w procedurze *ConstructKShortestPath*, o następującej treści:

```
   Wejście: T   - listy rekordów opisujących łuki drzewa najkrótszych
                  ścieżek i łuki stratne
            P   - lista rekordów opisujących K najkrótszych ścieżek
            vs     - wierzchołek początkowy
            ve     - wierzchołek końcowy
   Wyjście: PK  - lista K najkrótszych ścieżek
 1: procedure ConstructKShortestPath(T, P, vs, ve, PK)
 2: begin
 3:   for i := 0 to count(P)-1 do    // dla wszystkich ścieżek
 4:     p := ⟨vs⟩;                   // aktualnie odtwarzana ścieżka
 5:     for all arc ∈ P[i].arcs do   // dla wszystkich łuków stratnych w P[i]
 6:       while last(p) ≠ tail(arc) do     // dodanie do p łuków niestratnych
 7:         p := p ⊕ T[last(p)][0].arc ⊕ head(T[last(p)][0].arc);
 8:       end while
 9:       p := p ⊕ arc ⊕ head(arc);        // dodanie do p łuku stratnego
10:     end for
11:     while last(p) ≠ ve do     // dodanie do p łuków niestratnych
12:       p := p ⊕ T[last(p)][0].arc ⊕ head(T[last(p)][0].arc);
13:     end while
14:     PK := PK ⊕ p;             // dodanie ścieżki p do listy PK
15:   end for
16:   return PK;
17: end
```

Ścieżki są odtwarzane w iteracji **for** (wiersze 3–15), której liczba wykonań jest równa co najwyżej *K*. Jako pierwszy do odtwarzanej ścieżki *p* jest wstawiany wierzchołek początkowy v_s (wiersz 4), a następnie są przeglądane wszystkie łuki stratne należące do ścieżki (wiersze 5–10). Łuk stratny *arc* i wierzchołek, do którego on wchodzi, mogą być dodane do ścieżki *p*, jeżeli ostatni wierzchołek należący do ścieżki *p*, oznaczony przez *last*(*p*), jest identyczny jak wierzchołek, z którego wychodzi łuk *arc*, tj. *last*(*p*) = *tail*(*arc*). Jeżeli nie jest spełniona ta zależność, wówczas przed dodaniem do ścieżki łuku stratnego i wierzchołka *head*(*arc*)

(wiersz 9) następuje dodanie do ścieżki łuków bezstratnych i wierzchołków, które utworzą podścieżkę z wierzchołka *last*(*p*) do wierzchołka *tail*(*arc*) (wiersze 6–8). Jeżeli po dodaniu do ścieżki *p* wszystkich łuków stratnych ostatnim wierzchołkiem należącym do ścieżki nie jest wierzchołek końcowy v_e, to dodawane są do ścieżki wierzchołki i łuki bezstratne w celu utworzenia podścieżki z wierzchołka *last*(*p*) do wierzchołka v_e (wiersze 11–13). Utworzona ścieżka *p* jest dodawana do listy ścieżek P_K (wiersz 14), która jest wynikiem wykonania procedury (wiersz 16).

Wykonując procedurę dla listy rekordów *P* przedstawionej na rys. 7i, zostanie odtworzonych *K*= 3 następujących ścieżek:

$p_1 = \langle 1; (1, 3); 3; (3, 4); 4\rangle$,

$p_2 = \langle 1; \mathbf{(1, 2)}; 2; (2, 3); 3; (3, 4); 4\rangle$,

$p_3 = \langle 1; (1, 2); 3; (3, 4); 4; \mathbf{(4, 3)}; 3; (3, 4); 4\rangle$.

W zapisie ścieżek wyróżniono łuki stratne.

3.5. Pseudokod algorytmu wyznaczania *K* najkrótszych ścieżek

Algorytm wyznaczania *K* najkrótszych ścieżek można zapisać w postaci procedury *KShortestPath*, której treść przedstawia się następująco:

```
   Wejście: K             - liczba najkrótszych ścieżek, które należy wyznaczyć
            G = (V, E, w) - skierowany graf ważony z funkcją wagową w
            vs               - wierzchołek początkowy
            ve               - wierzchołek końcowy
   Wyjście: PK            - lista K najkrótszych ścieżek
 1: procedure KShortestPath(K, G, vs, ve, PK)
 2: begin
 3:   PK := ∅;             // pusta lista wyznaczonych ścieżek
 4:   ComputeShortestPathTree(G, ve, T);
 5:   if T[vs] ≠ ∅ then    // wierzchołek ve jest osiągalny z wierzchołka vs
 6:     ComputeKShortestPath(T, vs, K, P);
 7:     ConstructKShortestPath(T, P, vs, ve, PK);
 8:   end if
 9:   return PK;
10: end
```

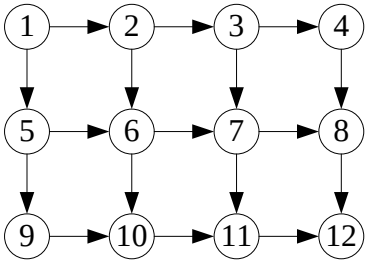
Zgodnie z opisem algorytmu przedstawionym w punkcie 3.1, jako pierwsze jest wyznaczane drzewo najkrótszych ścieżek z wszystkich wierzchołków grafu do wierzchołka końcowego v_e (wiersz 4). Jeżeli wierzchołek v_e jest osiągalny z wierzchołka v_s, to następuje wyznaczenie rekordów opisujących w sposób niejawny *K* najkrótszych ścieżek (wiersz 6), a następnie odtworzenie pełnych ścieżek (wiersz 7). Wynikiem działania algorytmu jest zbiór P_K zawierający *K* najkrótszych ścieżek (wiersz 9).

Do wyznaczenia drzewa najkrótszych ścieżek (wiersz 4) został użyty zmodyfikowany algorytm Dijkstry, przy czym modyfikacje nie wpływają na złożoność czasową algorytmu. Drzewo jest więc wyznaczane w czasie $O((N + M) \log(N))$ [4]. Drugim etapem algorytmu jest

wyznaczenie *K* rekordów opisujących w sposób niejawny *K* najkrótszych ścieżek (wiersz 6). Pierwszy rekord opisuje najkrótszą ścieżkę, która zawiera co najwyżej *N* wierzchołków. Podczas jego wyznaczania są analizowane wszystkie łuki wychodzące z wierzchołków należących do ścieżki. Z każdego wierzchołka może wychodzić co najwyżej $N - 1$ łuków, co oznacza, że jego wyznaczenie w przypadku pesymistycznym odbywa się w czasie $O(N^2)$. Wraz z rekordem opisującym najkrótszą ścieżkę zostanie wyznaczonych $K - 1$ rekordów opisujących kolejne ścieżki z wierzchołka początkowego v_s do wierzchołka v_i, każdy z nich zawiera jeden łuk stratny. Każdy z wyznaczonych rekordów jest poddawany analizie w celu wyznaczenia ścieżki z wierzchołka v_i do wierzchołka końcowego v_e. Warto zauważyć, że zostanie przeanalizowana najkrótsza ścieżka i wszystkie łuki wychodzące z wierzchołków, które do niej należą. Operacja ta jest analogiczna do wyznaczania rekordu reprezentującego najkrótszą ścieżkę i odbywa się w czasie $O(N^2)$. Tak więc pesymistyczna złożoność czasowa drugiego etapu algorytmu jest równa $O(K\ N^2)$. W ostatnim etapie algorytmu następuje odtworzenie pełnych ścieżek (wiersz 7), co jest realizowane w identycznym czasie jak wyznaczanie rekordów w drugim etapie. Oznacza to, że pesymistyczna złożoność czasowa algorytmu wyznaczania *K* najkrótszych ścieżek jest równa $O((N + M) \log(N) + K\ N^2)$.

4. Wyniki badań eksperymentalnych

Badania eksperymentalne algorytmu opisanego w rozdziale 3, oznaczonego w dalszej części jako DA (*Deviation Algorithm*), zostały przeprowadzone dla trzech rodzajów grafów: grafu pełnego, grafu o topologii kratownicy i grafu o losowej topologii. Graf o topologii kratownicy jest grafem, w którym wierzchołki są ułożone w n_k kolumn i n_w wierszy. Z wierzchołka o współrzędnych (w, k), gdzie w jest numerem wiersza, a k numerem kolumny, w której się znajduje, wychodzą łuki do dwóch wierzchołków, odpowiednio o współrzędnych $(w, k + 1)$ i $(w + 1, k)$. Przykład takiego grafu został przedstawiony na rys. 8. Oprócz badań dla algorytmu DA, dodatkowo przeprowadzono badania dla trzech algorytmów etykietowania: LS (*Label Setting*), DS (*Double Sweep*), SL (*Sequence List*) [76].



Rys. 8. Przykładowy graf o topologii kraty
Fig. 8. A sample graph of mesh–connected topology

Wyniki przeprowadzonych badań zostały przedstawione w tabelach 2–5. Każda seria badań obejmowała pomiar czasu wyznaczania K najkrótszych ścieżek między pięcioma parami wierzchołków z użyciem każdego z czterech algorytmów. Czasy prezentowane w tabelach stanowią wartość średnią z pięciu pomiarów i są wyrażone w milisekundach. Zostały przeprowadzone dwa etapy badań. Celem pierwszego etapu było zbadanie zależności czasu wyznaczania K najkrótszych ścieżek od liczby wierzchołków grafu N, przy stałej wartości parametru K. Celem drugiego etapu badań było zbadanie zależności czasu wyznaczania K najkrótszych ścieżek od parametru K, przy stałej liczbie wierzchołków N.

Tabela 2

Zależność czasu wyznaczania $K = 5$ i $K = 20$ najkrótszych ścieżek w grafie pełnym od liczby wierzchołków grafu N

N	DS [ms]		SL [ms]		LS [ms]		DA [ms]	
	$K = 5$	$K = 20$	$K = 5$	$K = 20$	$K = 5$	$K = 20$	$K = 5$	$K = 20$
50	2,0	7,8	1,6	6,6	0,4	4,2	2,6	4,6
100	7,6	19,6	8,6	27,6	3,8	17,2	9,6	18,2
200	40,8	66,2	28,6	58,8	14,0	66,2	58,6	67,2
300	73,2	117,2	65,8	97,2	27,8	105,4	179,6	183,2
500	128,4	182,8	96,4	158,8	53,8	220,2	596,2	672,8

Tabela 3

Zależność czasu wyznaczania $K = 20$ najkrótszych ścieżek w grafie o topologii kratownicy od liczby wierzchołków grafu N

N	DS [ms]	SL [ms]	LS [ms]	DA [ms]
$15 \times 15 = 225$	1,4	1,6	2,6	0,1
$90 \times 5 = 450$	4,2	3,6	5,2	0,2
$60 \times 15 = 900$	13,0	11,2	14,2	0,6
$50 \times 50 = 2500$	31,6	28,6	36,4	6,6

Tabela 4

Zależność czasu wyznaczania $K = 20$ najkrótszych ścieżek w grafie o losowej topologii od liczby wierzchołków grafu N

N	DS [ms]	SL [ms]	LS [ms]	DA [ms]
100	10,6	11,0	8,0	4,4
200	17,8	17,4	12,2	4,8
300	32,8	36,6	23,0	10,8
400	28,8	31	19,8	6,8
500	38,8	38	22,8	10,2
750	50,0	48,8	34,0	11,4
1000	61,4	61,4	43,2	17,6

W tabelach 2–4 przedstawiono zależność czasu wyznaczania K najkrótszych ścieżek od liczby wierzchołków grafu N. Dla grafu o topologii kratownicy w tabeli 3 podano liczbę n_k kolumn i n_w wierszy tworzących kratownicę. Dla wszystkich grafów badania przeprowadzono dla $K = 20$, a dla grafu pełnego dodatkowo dla $K = 5$. Dla grafu pełnego i liczby ścieżek $K = 5$

największe czasy wyznaczania rozwiązania uzyskano dla algorytmu DA (tabela 2). Dla największego przebadanego grafu, zawierającego $N = 500$ wierzchołków, czas obliczeń jest ok. 11 razy większy od najmniejszego czasu obliczeń, który został uzyskany dla algorytmu LS. Strata w stosunku do najlepszego algorytmu LS rośnie wraz ze wzrostem liczby wierzchołków grafu. Jak się okazuje, strata algorytmu DA do najlepszego algorytmu maleje wraz ze wzrostem liczby K wyznaczanych ścieżek. Dla $K = 20$ czas wyznaczania rozwiązań z użyciem algorytmu DA w najgorszym przypadku jest ok. 4 razy większy od minimalnego czasu, jaki uzyskano dla algorytmu SL.

Dla grafu o topologii kratownicy i losowej topologii najmniejszy czas obliczeń został uzyskany dla opracowanego algorytmu DA. Dla grafu o topologii kratownicy największy czas obliczeń uzyskano dla algorytmu LS i jest on ok. 26 razy większy od czasu obliczeń dla algorytmu DA (tabela 3). Najmniejszy czas obliczeń może wynikać z braku cykli w przetwarzanych grafach. Ścieżki są wyznaczane przez przemieszczanie się w kratownicy wyłącznie „w prawo" i „w dół", dzięki czemu część łuków i wierzchołków grafu na pewno nie będzie należeć do ścieżki. W algorytmie DA takie wierzchołki i łuki są wykrywane podczas wyznaczania drzewa najkrótszych ścieżek i łuków stratnych (pierwszy etap algorytmu) i nie są badane w drugim i trzecim etapie. Z tego samego powodu algorytm DA uzyskuje najmniejsze czasy dla grafów o losowej topologii (tabela 4). Warto zwrócić uwagę na to, że wzrost liczby wierzchołków nie powoduje znaczącego wzrostu czasu obliczeń jak w przypadku grafu pełnego. Wynika to z faktu, że wzrost liczby wierzchołków nie pociąga wzrostu łącznej liczby łuków grafu. Wszystkie grafy o losowej topologii miały łączną liczbę łuków równą ok. 3000, a różnice w grafach występowały w liczbie łuków wychodzących z wierzchołków.

Tabela 5

Zależność czasu wyznaczania K najkrótszych ścieżek w grafie pełnym, grafie o topologii kratownicy, grafie o losowej topologii, zawierającym N wierzchołków, od parametru K. Jednostką czasu jest [ms].

K	Pełny ($N = 300$)				Kratownica ($N = 50 \times 50$)				Losowy ($N = 500$)			
	DS	SL	LS	DA	DS	SL	LS	DA	DS	SL	LS	DA
5	73	65	27	179	7	9	11	4	11	8	4	8
20	117	97	105	183	31	28	36	6	38	38	22	10
50	182	194	316	252	76	72	104	8	96	114	69	14
100	336	406	663	269	173	182	206	8	267	271	192	25
200	887	1108	1698	380	388	394	423	6	655	679	489	37

W drugim etapie badań zbadano zależność czasu wyznaczania K najkrótszych ścieżek od parametru K, przy stałej liczbie wierzchołków grafu (tabela 5). Podatność algorytmu DA na zmianę wartości parametru K nie jest tak znaczna jak w przypadku pozostałych algorytmów. Dla tego algorytmu, niezależnie od topologii grafu, został uzyskany najmniejszy czas

obliczeń. Zwiększenie dla grafu pełnego o rząd wartości K (z 20 do 200) powoduje jedynie dwukrotny wzrost czasu obliczeń z użyciem algorytmu DA, podczas gdy dla pozostałych algorytmów jest on ośmio- (DS), jedenasto- (SL), a nawet szesnastokrotny (LS). W przypadku grafu o topologii kratownicy zmiana wartości parametru K wpływa nieznacznie na czas obliczeń z użyciem algorytmu DA, podczas gdy dla pozostałych algorytmów można zaobserwować znaczny wzrost czasu obliczeń.

5. Podsumowanie

Problem wyznaczania najkrótszej ścieżki w grafie ważonym jest jednym z badanych problemów teorii grafów. Jego rozszerzeniem jest problem wyznaczania K ($K > 1$) najkrótszych ścieżek, który został omówiony w niniejszej pracy. Rozpatrywane są dwa warianty problemu wyznaczania K najkrótszych ścieżek. W pierwszym wariancie są wyznaczane wyłącznie ścieżki acykliczne, natomiast w drugim wariancie dopuszcza się wyznaczenie ścieżek zawierających cykle. Wśród algorytmów rozwiązywania niniejszego problemu można wyróżnić algorytmy etykietowania, dzielące się na dwie grupy: nadawania lub korygowania etykiet. Alternatywą dla algorytmów etykietowania jest algorytm dewiacyjny, w którym ścieżki są wyznaczane wykorzystaniu drzewa najkrótszych ścieżek.

W niniejszej pracy został zaproponowany algorytm dewiacyjny umożliwiający wyznaczenie K najkrótszych ścieżek, w których jest dopuszczalne występowanie cykli. Rozwiązania są wyznaczane w trzech etapach. W pierwszym etapie jest wyznaczane drzewo najkrótszych ścieżek z wszystkich wierzchołków grafu do wierzchołka końcowego oraz łuki stratne. Dodatkowo, są eliminowane z dalszej analizy wierzchołki i łuki, do których wiadomo, że nie będą należeć do ścieżki. W drugim etapie są wyznaczane niejawne reprezentacje K najkrótszych ścieżek. Ścieżki te są opisane wyłącznie sekwencją łuków stratnych. Pełne ścieżki są odtwarzane w ostatnim etapie algorytmu. Pesymistyczna złożoność czasowa opracowanego algorytmu jest równa $O((N + M) \log(N) + K N^2)$.

Dla opracowanego algorytmu oraz dodatkowo dla trzech algorytmów etykietowania zostały przeprowadzone badanie eksperymentalne. Celem badań było określenie zależności czasu obliczeń od liczby wierzchołków grafu przy stałej wartości parametru K oraz zależności czasu obliczeń od parametru K przy stałej liczbie wierzchołków grafu. Badania przeprowadzono dla trzech rodzajów grafów: grafów pełnych, grafów o topologii kratownicy i grafów o losowej topologii. Przeprowadzone badania wykazały, że dla dużych wartości parametru K najmniejszy czas obliczeń, niezależnie od topologii grafu, został uzyskany dla opracowanego algorytmu. Fakt ten wynika z pomijania w drugim i trzecim etapie algorytmu części wierzchołków i łuków, co nie jest wykonywane w pozostałych badanych algorytmach.

Dodatkowo na czas obliczeń wpływa zastosowana strategia odwiedzania wierzchołków, będąca połączeniem przeszukiwania grafu w głąb i wszerz.

BIBLIOGRAFIA

1. Azevedo J. A., Costa M. E. O. S., Madeira J. J. R. E. S., Martins E. Q. V.: An algorithm for the ranking of shortest paths. European Journal of Operational Research, Vol. 69, 1993, s. 97÷106.
2. Azevedo J. A., Madeira J. J. R. E. S., Martins E. Q. V., Pires F. M. A.: A computational improvement for a shortest paths ranking algorithm. European Journal of Operational Research, Vol. 73, 1994, s. 188÷191.
3. Bellman R.: On a routing problem. Quarterly of Applied Mathematics, Vol. 16, No. 1, 1958, s. 87÷90.
4. Cormen T., Leiserson C., Rivest R., Stein C.: Wprowadzenie do algorytmów. WNT, Warszawa 2004.
5. Dijkstra E. W.: A note on two problems in connexion with graphs. Numerical Mathematics, Vol. 1, 1959, s. 269÷271.
6. Dreyfus S. E.: An appraisal of some shortest-path algorithms. Operations Research, Vol. 17, 1969, s. 395÷412.
7. Eppstein D.: Finding the k shortest paths. SIAM Journal on Computing, Vol. 28, No. 2, 1998, s. 652÷673.
8. Floyd R. W.: Algorithm 97: Shortest path. Communications of the ACM, Vol. 5, No. 6, 1962, s. 345.
9. Ford Jr L. R., Fulkerson D. R.: Flows in networks. Princeton University Press, 1962.
10. Hershberger J., Maxel M., Suri S.: Finding the k Shortest Simple Paths: A New Algorithm and Its Implementation. In Proceedings of ALENEX'03, 2003, s. 26÷36.
11. Johnson D. B.: Efficient algorithms for shortest paths in sparse networks. Journal of the ACM, Vol. 24, No. , 19771, s. 1÷13.
12. Jungnickel D.: Graphs, networks and algorithms. Springer, Berlin 1999.
13. Katoh N., Ibaraki T., Mine H.: An efficient algorithm for k shortest simple paths. Networks, Vol. 12, 1982, s. 411÷427.
14. Lipski W.: Kombinatoryka dla programistów. WNT, Warszawa 1989.
15. Martins E. Q. V., Pascoal M. M. B., Santos J. L. E.: An algorithm for ranking paths that may contain cycles. European Journal of Operational Research, Vol. 18, 1984, s. 123÷130.

16. Martins E. Q. V., Pascoal M. M. B., Santos J. L. E.: Deviation algorithms for ranking shortest paths. International Journal of Foundations of Computer Science, Vol. 10, No. 3, 1999, s. 247÷261.
17. Martins E. Q. V., Santos J. L. E.: A new shortest paths ranking algorithm. Investigação Operacional, Vol. 20, No. 1, 2000, s. 47÷62.
18. Perko A.: Implementation of algorithms for k shortest loopless paths. Networks, Vol. 16, 1986, s. 149÷160.
19. Reingold E. M., Nievergelt J., Deo N.: Algorytmy kombinatoryczne. PWN, Warszawa 1985.
20. Shier D. R.: On algorithms for finding the k shortest paths in a network. Networks, Vol. 9, 1979, s. 195÷214.
21. Sysło M., Deo N., Kowalik J.: Algorytmy optymalizacji dyskretnej z programami w języku Pascal. PWN, Warszawa 1999.
22. Yen J. Y.: Finding the k shortest loopless paths in a network. Management Science, Vol. 17, 1971, s. 712÷716.
23. Warshall S.: A theorem on boolean matrices. Journal of the ACM, Vol. 9, No. 1, 1962, s. 11÷12.

Recenzent: Dr hab. inż. Mariusz Boryczka

Wpłynęło do Redakcji 5 stycznia 2011 r.

Abstract

The shortest path problem is one of the studied problems of graph theory. The extension of this problem is the problem of determining K ($K > 1$) shortest paths, which is discussed in the paper. Two cases of the K shortest paths problem are considered. In the first case only loopless paths are determined [10, 13, 16, 18, 22], while in the second case determination of paths containing cycles are allowed [1, 2, 6, 7, 15, 17, 20]. The first group of algorithms for solving the K shortest paths problem are labeling algorithms which are divided into label setting [1, 2, 6, 15, 17] and label correcting algorithms [20]. Another types of algorithms are deviation algorithms where the paths are determined on the basis of the shortest–path tree [7].

In the paper deviation algorithm (DA) for solving the K shortest paths problem is proposed. There is considered problem of determination paths containing cycles. Determination of the paths consists of three phases. In the first phase a shortest path tree from

all vertices to the final vertex is determined. Arcs that do not belong to the shortest path tree are marked as arcs of loss. For those arcs there is determined increasing the weight of the path, defined by formula (1), resulting from adding this arc to the path. In addition, vertices from which cannot be reached the final vertex are determined. These vertices and arcs that are adjacent to them are not analyzed in next phases of the algorithm. In the second phase *K* shortest paths are determined but the paths are described by sequence of arcs of loss. All arcs and vertices belonging to the paths are determined in the last phase of the algorithm.

The DA algorithm and three labeling algorithms were tested using three kinds of graphs: complete graphs, graphs with mesh–connected topology and graphs with random topology. The aim of the tests was determination relationship between computation time and number of vertices and determination relationship between computation time and the *K* parameter. The tests showed that for large values of the *K* parameter smallest computation time has been obtained for DA algorithm. It is results of omitting during computation mentioned part of vertices and arcs, which is not included on the other studied algorithms.

Adresy

Marcin SZOŁTYSEK: marcin.szoltysek@gmail.com.
Jacek WIDUCH: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44-100 Gliwice, Polska, jacek.widuch@polsl.pl .