Uniwersytet Zielonogórski
Instytut Sterowania i Systemów Informatycznych

Algorytmy i struktury danych

Laboratorium Nr 8

Grafy

1 Cel ćwiczenia

Ćwiczenie ma na celu zapoznanie studentów z wybranymi zagadnieniami dotyczącymi grafów zarówno skierowanych jak i nieskierowanych.

2 Wstęp

W informatyce występuje wiele zastosowań dla grafów. Najbardziej popularne z nich to problemy: reprezentacji grafu w pamięci komputera; efektywnego przeglądania grafu; wyznaczenia najkrótszej ścieżki z ustalonego wierzchołka do innego, oraz wiele innych.

3 Grafy skierowane i grafy nieskierowane

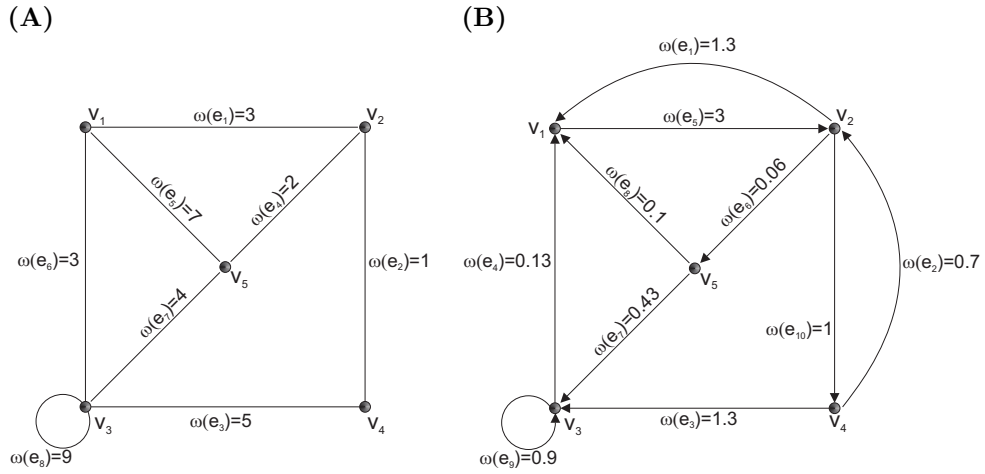
Każdy graf składa się ze zbioru wierzchołków V (ang. *vertex* – wierzchołek) oraz zbioru krawędzi E (ang. *edge* – krawędź), co symbolicznie zapisujemy $G = (V, E)$. W dalszej części tego opracowania będziemy zajmować się grafami, dla których oba zbiory są zbiorami zawierającymi skończoną liczbę elementów t.j. $|V|, |E| < \infty$ ($|V|$ – liczba wierzchołków w grafie; $|E|$ – liczba krawędzi w grafie). Istnieje wiele rodzajów grafów, przy czym, biorąc pod uwagę samą strukturę grafu możemy podzielić je na grafy skierowane i grafy nieskierowane. Formalnie, graf definiuje się jako czwórkę $G = (V, E, \gamma, \omega)$, gdzie V jest zbiorem wierzchołków, E - zbiorem krawędzi, $\omega : E \to \mathbb{R}_+$ jest funkcją przyporządkowującą każdej krawędzi odpowiednią, nieujemną wagę. Różnica pomiędzy definicją grafu skierowanego (digraf) i nieskierowanego wyraża się w postaci funkcji γ, która definiuje strukturę grafu. Dla grafu funkcja ta przedstawia się następująco: $\gamma : E \to \{\{u, v\} : u, v \in V\}$, natomiast dla digrafu funkcja ta jest postaci: $\gamma : E \to V \times V$. Zapis ten oznacza dokładanie tyle, że w przypadku digrafu możemy dla każdej krawędzi wyróżnić jej początek oraz koniec, natomiast dla grafu nieskierowanego takie rozróżnienie jest niemożliwe.

Rozważmy grafy przedstawione na Rys. 3.1. Matematycznie zapis grafu nieskierowanego **(A)** możemy przedstawić następująco:

- $G = (V, E, \gamma, \omega)$
- $V = \{v_1, v_2, v_3, v_4, v_5\}$
- $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$
- Funkcje γ, ω mogą być zaprezentowane tak jak podano w Tab. 3.1.

Analogicznie zdefiniujemy graf skierowany **(B)** jako czwórkę:

- $G = (V, E, \gamma, \omega)$
- $V = \{v_1, v_2, v_3, v_4, v_5\}$
- $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\}$
- Funkcje γ i ω mogą być zaprezentowane tak jak podano w Tab. 3.2.



Rys. 3.1: Graf **(A)** oraz graf skierowany (digraf) **(B)**.

E	$\gamma : E \to \{\{u, v\} : u, v \in \boldsymbol{V}\}$
e_1	$\{v_1, v_2\}$
e_2	$\{v_2, v_4\}$
e_3	$\{v_3, v_4\}$
e_4	$\{v_2, v_5\}$
e_5	$\{v_1, v_5\}$
e_6	$\{v_1, v_3\}$
e_7	$\{v_5, v_3\}$
e_8	$\{v_3, v_3\}$

E	$\omega : E \to \mathbb{R}_+$
e_1	3
e_2	1
e_3	5
e_4	2
e_5	7
e_6	3
e_7	4
e_8	9

Tab. 3.1: Funkcje γ i ω dla grafu nieskierowanego.

$\boldsymbol{E}$	$\gamma : \boldsymbol{E} \to \boldsymbol{V} \times \boldsymbol{V}$
e_1	(v_2, v_1)
e_2	(v_4, v_2)
e_3	(v_4, v_3)
e_4	(v_3, v_1)
e_5	(v_1, v_2)
e_6	(v_2, v_5)
e_7	(v_5, v_3)
e_8	(v_5, v_1)
e_9	(v_3, v_3)
e_{10}	(v_1, v_4)

$\boldsymbol{E}$	$\omega : \boldsymbol{E} \to \mathbb{R}_+$
e_1	1.3
e_2	0.7
e_3	1.3
e_4	0.13
e_5	3
e_6	0.06
e_7	0.43
e_8	0.1
e_9	0.9
e_{10}	1

Tab. 3.2: Funkcje γ i ω dla grafu skierowanego.

4 Reprezentacje grafów

Grafy w pamięci komputera najczęściej reprezentuje się za pomocą *listy sąsiedztw* oraz za pomocą *macierzy sąsiedztw*. Każda z tych metod ma oczywiście swoje wady i zalety, stąd też wybór odpowiedniej reprezentacji powinien być oparty na podstawie wiedzy o problemie, który chcemy sformułować w postaci grafu. Podejmując decyzję dotyczącą reprezentacji, grafy można podzielić na dwie kategorie: grafy rzadkie oraz grafy gęste. Pierwsza klasa charakteryzuje się małym stosunkiem liczby krawędzi do liczby wierzchołków t.j. $|\boldsymbol{E}| \ll |\boldsymbol{V}|^2$. Dla grafów gęstych stosunek ten jest bliski jedności tzn.: $|\boldsymbol{E}| \approx |\boldsymbol{V}|^2$.

4.1 Reprezentacja macierzowa

Jeśli chcemy posłużyć się reprezentacją macierzową grafu, musimy najpierw nadać etykiety poszczególnym wierzchołkom grafu. Można to zrobić na wiele sposobów, np.:

- nadać etykiety literowe (np.: $a, b, c, d, \ldots$);

- nadać nazwy miejscowości (jeśli graf reprezentuje szlaki komunikacyjne między miastami);
- ponumerować je kolejnymi liczbami (np. $1, 2, 3, \ldots, |V|$ lub z literą $v_1, v_2, v_3, \ldots, v_{|V|}$);
- lub w jakikolwiek inny adekwatny sposób.

Reprezentacja macierzowa sprowadza się do rozważania macierzy $A = (a_{i,j})$ wymiaru $|V| \times |V|$ takiej że:

$$a_{i,j} = \begin{cases} \omega\Big(\gamma(i,j)\Big), & \text{jeżeli } \gamma(i,j) \in E \\ 0 \quad \text{lub} \quad \infty, & \text{w przeciwnym razie} \end{cases}$$

czyli element o indeksach (i, j) w powyższej macierzy zawiera wagę połączenia pomiędzy wierzchołkiem i oraz wierzchołkiem j (jeśli odpowiednia krawędź istnieje), oraz 0 lub ∞ (w zależności od rozważanego problemu), gdy odpowiednia krawędź nie należy do zbioru E.

Rozważmy grafy przedstawione na Rys. 3.1. Macierze sąsiedztw przedstawiają się dla tych grafów odpowiednio:

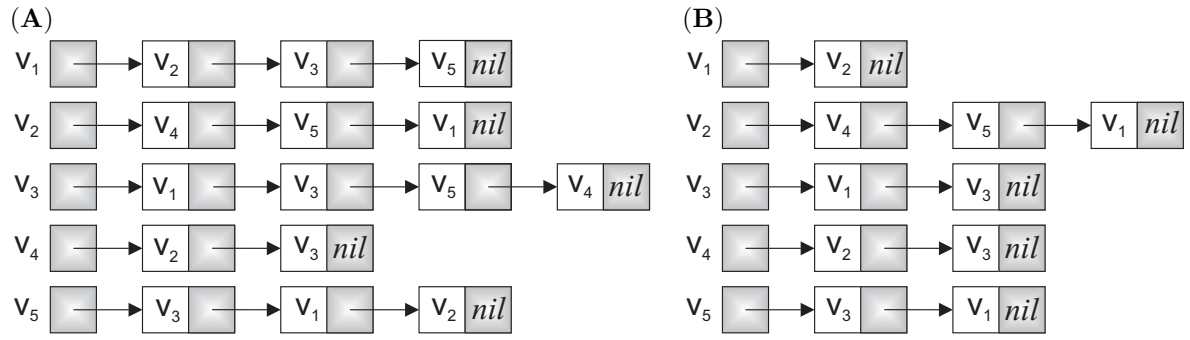
A	v_1	v_2	v_3	v_4	v_5
v_1	0	3	3	0	7
v_2	3	0	0	1	2
v_3	3	0	9	5	4
v_4	0	1	5	0	0
v_5	7	2	4	0	0

B	v_1	v_2	v_3	v_4	v_5
v_1	0	3	0	0	0
v_2	1.3	0	0	1	0.06
v_3	0.13	0	0.9	0	0
v_4	0	0	1.3	0	0
v_5	0.1	0	0.43	0	0

4.2 Lista sąsiedztw

W reprezentacji grafu za pomocą listy sąsiedztw, dana jest tablica L, zawierająca $|V|$ list, po jednej dla każdego wierzchołka. Dla każdego wierzchołka $v \in V$ elementami jego listy sąsiedztw $L[v]$ są wszystkie wierzchołki u_i takie, że istnieje krawędź z wierzchołka v do u_i. Kolejność wierzchołków na listach sąsiedztw, zazwyczaj jest dowolna (może jednakże być ułożona według jakiegoś porządku np. numeracji wierzchołków).

Rozważmy ponownie grafy z Rys. 3.1:



Rys. 4.1: Listy sąsiedztw dla grafów przedstawionych na Rys. 3.1

5 Przeszukiwanie grafów

W tej sekcji zajmiemy się omówieniem algorytmów służących do przeszukiwania grafów.

5.1 Przeszukiwanie w głąb (ang. *depth-first search*)

Przy przeszukiwaniu grafu w głąb, badane są krawędzie ostatnio odwiedzonego wierzchołka, z którego jeszcze wychodzą dotychczas niezbadane krawędzie. Wybieramy spośród nich pierwszą nieodwiedzoną krawędź i przechodzimy do następnego wierzchołka. Proces ten jest kontynuowany dopóki jest jeszcze jakaś nieodwiedzona krawędź. Gdy wszystkie krawędzie wierzchołka zostaną zbadane, przeszukiwanie

wraca do wierzchołka, z którego wierzchołek ten był odwiedzony i bada pozostałe (jeśli są) niezbadane krawędzie. Jeśli takich krawędzi nie ma znów cofamy się do poprzedniego wierzchołka. Procedura ta jest rekurencyjna, zatem przeglądanie wierzchołków jest kontynuowane, dopóki wszystkie wierzchołki osiągalne z początkowego wierzchołka nie zostaną odwiedzone. Może się zdarzyć, że w wyniku tak sformułowanego przeszukiwania, nie wszystkie wierzchołki w grafie zostaną odwiedzone. Wówczas należy wybrać nieodwiedzony jeszcze wierzchołek jako nowy wierzchołek startowy i proces przeszukiwania kontynuować do momentu odwiedzenia wszystkich wierzchołków grafu.

Algorytm przeglądania grafu w głąb został przedstawiony w Tab. 5.1.

```
     Procedura DFS(G)
1.   Dla każdego v ∈ V wykonaj
2.       K[v] ← BIAŁY
3.       P[v] ← NIL
4.   czas ← 0
5.   Dla każdego v ∈ V wykonaj
6.       Jeśli K[v] =BIAŁY to wykonaj
7.           OdwiedźDFS(v)
```

Tab. 5.1: Procedura DFS.

```
     Procedura OdwiedźDFS(u)
1.   K[u] ← SZARY
2.   czas ← czas +1
3.   T_p[u] ← czas
4.   Dla każdego v ∈ L[u] wykonaj        ▷ Zbadaj krawędź (u, v)
5.       Jeśli K[v] =BIAŁY to wykonaj
6.           P[v] ← u
7.           OdwiedzDFS(v)
8.   K[u] ← CZARNY
9.   czas ← czas +1
10.  T_k[u] ← czas
```

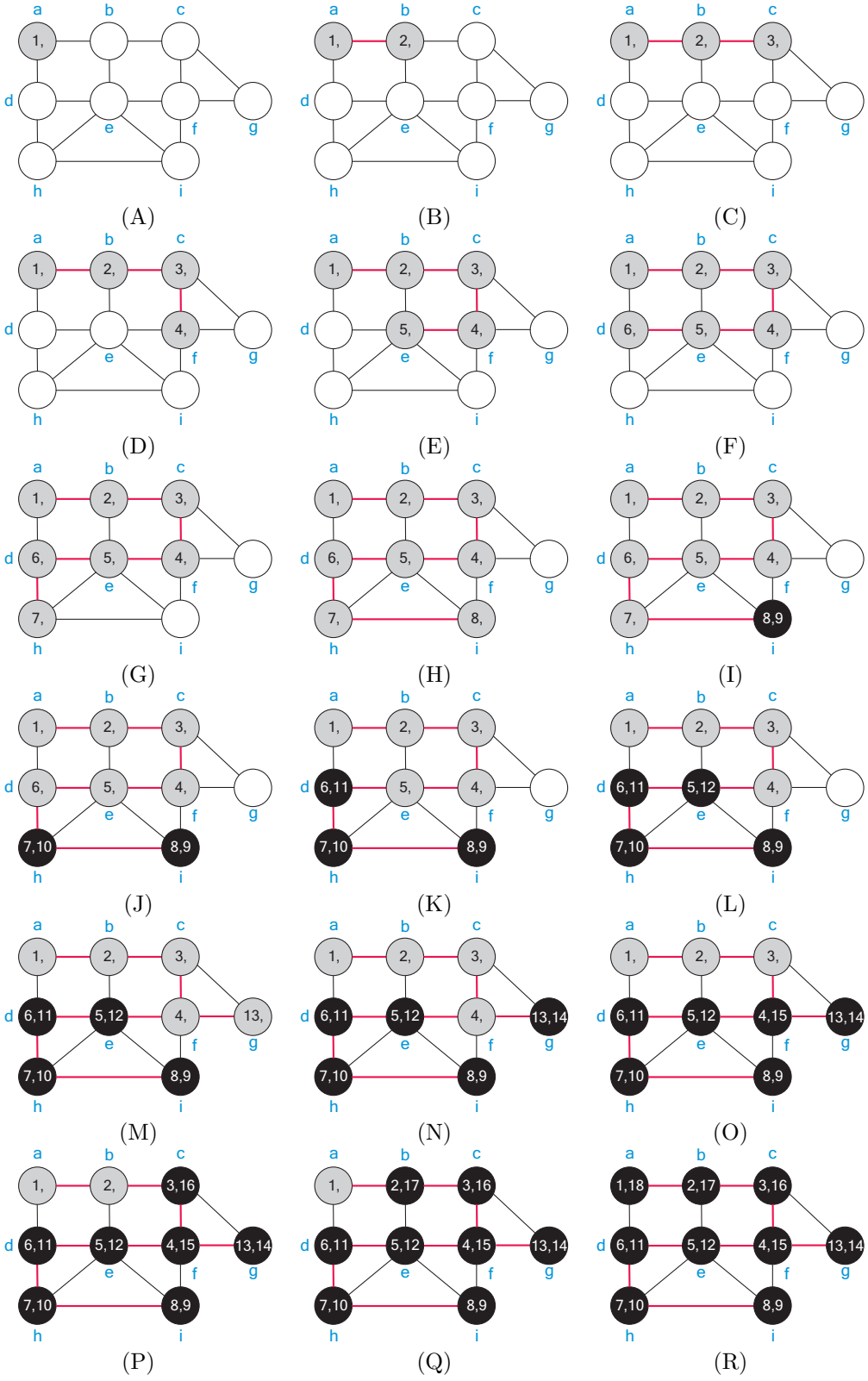
Tab. 5.2: Procedura OdwiedźDFS.

Oznaczenia przyjęte w tej procedurze są następujące:

- G – rozważany graf;
- V – wierzchołki grafu G;
- K – wektor, w którym dla każdego wierzchołka przechowywana jest informacja o jego kolorze;
- czas – zmienna, która służy do odnotowania, w którym kroku algorytmu został odwiedzony wierzchołek;
- T_p – wektor, w którym dla każdego wierzchołka jest przechowywany czas jego odwiedzenia;
- T_k – wektor, w którym dla każdego wierzchołka jest przechowywany czas jego przetworzenia;
- P – wektor, w której pamiętany jest poprzednik danego wierzchołka.

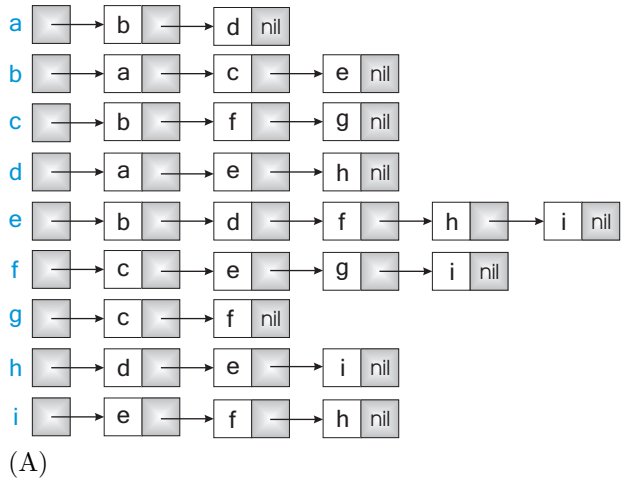
Algorytm DFS rozpoczyna swe działanie od pokolorwania wszystkich wierzchołków grafu G kolorem białym i ustawieniu poprzedników danego wierzchołka v na wartość NIL (linie 1–3 algorytmu). Wartość zmiennej *czas* ustawiana jest na zero. Następnie dla każdego wierzchołka w grafie wykonana jest procedura *OdwiedźDFS* jeśli badany wierzchołek jest biały.

Procedura *OdwiedźDFS* rozpoczyna swe działanie od pokolorwania wierzchołka białego na kolor szary. Zwiększamy jego czas o jeden i w wierzchołku, już szarym, wpisujemy czas odwiedzenia. Następnie badani są jego sąsiedzi (linia 4). Jeśli jakikolwiek jego sąsiad nie został jeszcze odwiedzony (ma kolor biały) to ustawiany jest poprzednik wierzchołka v na swego rodzica u oraz zostaje wywołana procedura

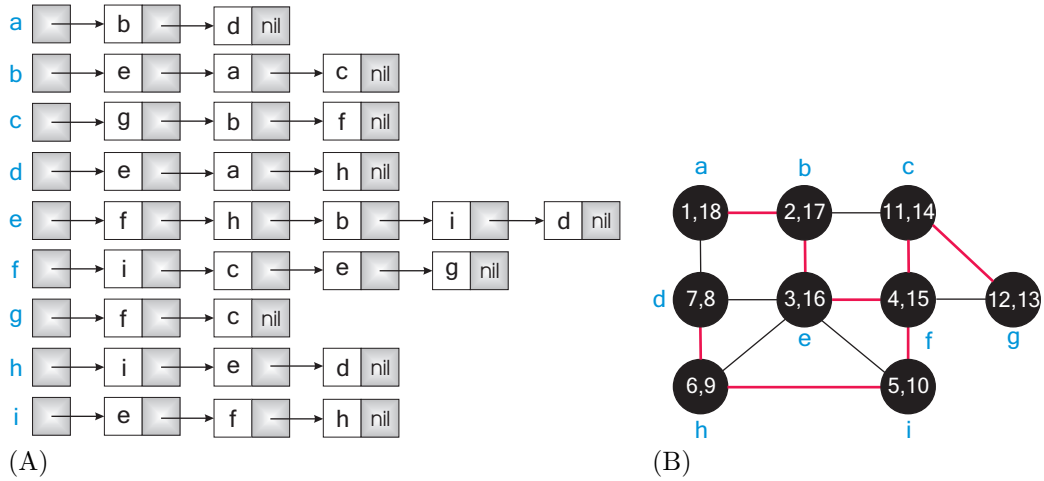


Rys. 5.1: Ilustracja działania algorytmu przeszukiwania grafu w głąb (DFS). W wierzchołkach są wpisane czasy odwiedzin. Pierwsza liczba oznacza czas odwiedzenia wierzchołka, druga liczba (po przecinku) oznacza czas przetworzenia wierzchołka. Krawędzie czerwone rozpinają drzewo DFS na podstawie listy poprzedników P. Wierzchołek szary oznacza, że został on wybrany do przetworzenia ale nie wszyscy jego sąsiedzi zostali jeszcze odwiedzeni. Wierzchołek czarny oznacza, że wszyscy sąsiedzi tego wierzchołka, zostali już odwiedzeni i więcej ten wierzchołek nie będzie brany pod uwagę w algorytmie.

rekurencyjnie dla wierzchołka v. Gdy przeglądanie krawędzi względem u się zakończy, wykonywany jest



Rys. 5.2: Dla Rys. 5.1: (A) Lista sąsiedztw, (B) poprzednicy przechowani w zmiennej P w chwili zakończenia algorytmu.



Rys. 5.3: (A) Lista sąsiedztw – z innym uporządkowaniem wierzchołków na listach – dla grafu przedstawionego na Rys. 5.1. Dla takiej listy sąsiedztw uzyskano inne drzewo przeszukiwań DFS (B).

kod w liniach 8 – 10, gdzie wierzchołek u zostaje pokolorowany na czarno, czas zostaje zwiększony o jeden oraz wpisany zostaje czas przetworzenia wierzchołka u w wektorze T_k.

Przykład działania algorytmu został przedstawiony na Rys. 5.1 dla listy sąsiedztw grafu przedstawionego na Rys. 5.2 (A). W tej liście sąsiedztw wierzchołki uporządkowane są alfabetycznie, stąd algorytm przeszukiwania wyznaczył drzewo DFS takie jak pokazano na Rys. 5.1 (krawędzie zaznaczone czerwonym kolorem). Natomiast, jeśli byśmy zmienili uporządkowanie wierzchołków w liście sąsiedztw na przedstawione na Rys. 5.3 (A), to otrzymalibyśmy drzewo DFS przedstawione na Rys. 5.3 (B). Można zatem zauważyć, że wynik przeszukiwania w głąb zależy od kolejności przeglądania wierzchołków (linia 5) w procedurze *DFS*, a także od kolejności przeglądania sąsiadów (linia 4) w procedurze *OdwiedźDFS*. Mimo, że uzyskamy różne drzewa DFS, dla różnego uporządkowania wierzchołków i kolejności przeglądania ich sąsiadów, to nie powoduje to zazwyczaj problemów w późniejszych zastosowaniach tak uzyskanych drzew DFS (dają one zazwyczaj równoważne rezultaty dla różnych drzew DFS).

Na Rys. 5.2 (B) przedstawiono poprzedników dla każdego z wierzchołków grafu przedstawionego na Rys. 5.1 (R). Korzystając z poprzedników umieszczonych w zmiennej P można podać procedurę wypisującą najkrótszą ścieżkę z danego wierzchołka startowego s (jeśli byłoby ich kilka, to dla każdego wierzchołka startowego należałoby wywołać tę procedurę osobno) do innego wierzchołka v w grafie G. Procedurę tę przedstawiono w Tab. 5.3.

W Tab. 5.4 zostały pokazane najkrótsze ścieżki i ich czasy odwiedzin, względem wierzchołka startowego dla przykładu przedstawionego na Rys. 5.1.

```
    Procedura WypiszŚcieżkę(G, P, s, v)
1.  Jeśli v = s to wykonaj
2.      wypisz s
3.  w przeciwnym razie jeśli P[v] =NIL to wykonaj
4.      wypisz "Nie ma ścieżki z " s " do " v
5.  w przeciwnym razie
6.      WypiszŚcieżkę(G, P, s, P[v])
7.      wypisz v
```

Tab. 5.3: Procedura wypisująca ścieżkę z wierzchołka startowego (s) do zadanego (v) w grafie (G).

Wierzchołek		Uzyskana	Czas	Czas
źródłowy	docelowy	ścieżka	$T_p[v]$	$T_k[v]$
a	a	a	1	18
a	b	a, b	2	17
a	c	a, b, c	3	16
a	f	a, b, c, f	4	15
a	e	a, b, c, f, e	5	12
a	d	a, b, c, f, e, d	6	11
a	h	a, b, c, f, e, d, h	7	10
a	i	a, b, c, f, e, d, h, i	8	9
a	g	a, b, c, f, g	13	14

Tab. 5.4: Wyznaczenie najkrótszych ścieżek przy pomocy procedury *WypiszŚcieżkę* oraz czasy odwiedzin wierzchołków.

5.2 Przeszukiwanie wszerz (ang. *breadth-first search*)

Przeszukiwanie w szerz jest jednym z najprostszych algorytmów przeszukiwania grafu. Algorytm rozpoczyna swe działanie od ustalonego wierzchołka źródłowego s i systematycznie przegląda wszystkie krawędzie wychodzące z tego wierzchołka, dodając odkryte w ten sposób wierzchołki, do kolejki Q. Następnie gdy wierzchołek źródłowy nie ma już więcej krawędzi do zbadania, pobierany jest wierzchołek z kolejki Q i procedura ta powtarza się od początku, tak, jak by to było wykonywane dla wierzchołka źródłowego aż do przetworzenia w ten sposób wszystkich wierzchołków umieszczonych w kolejce Q.

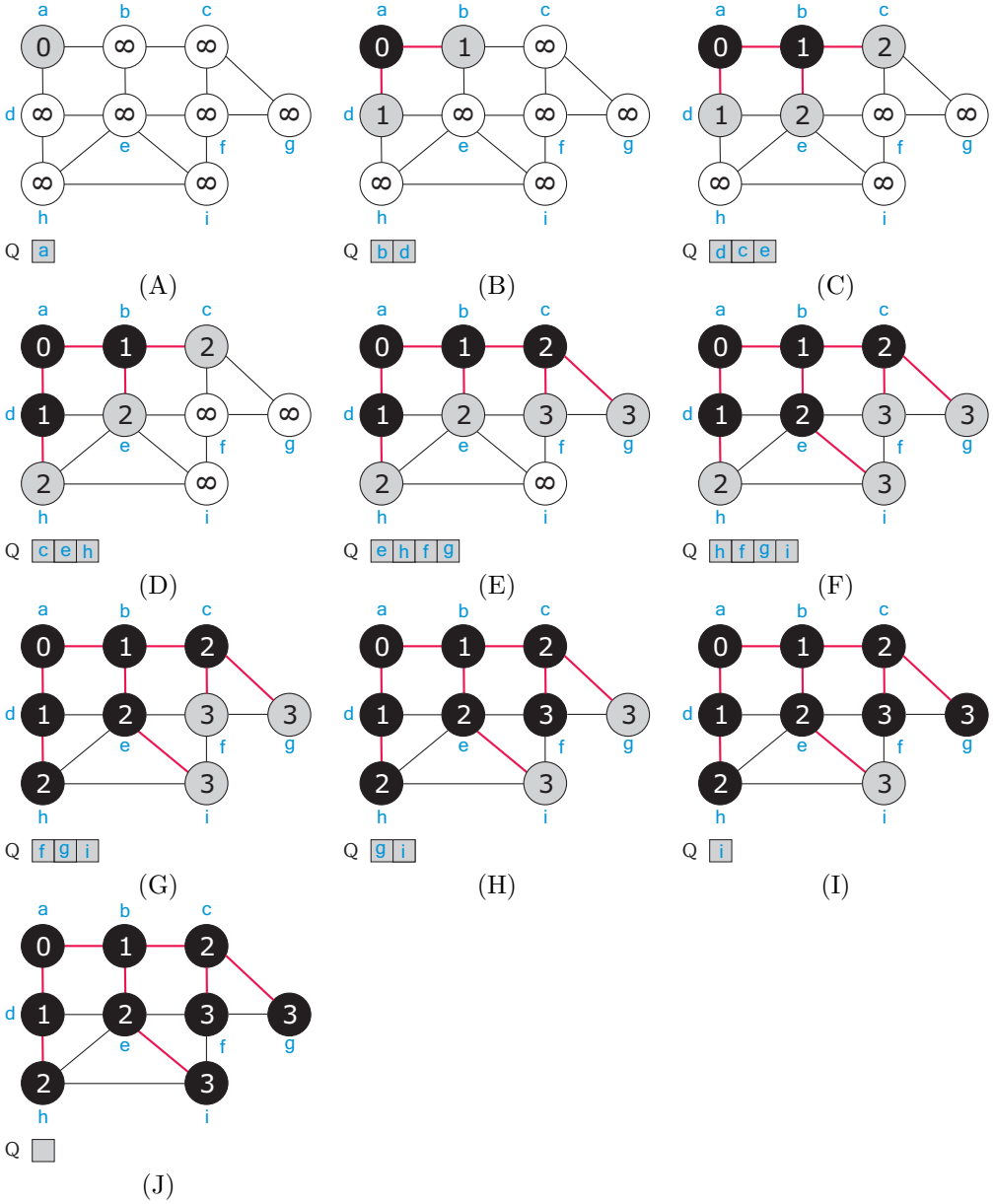
Ten typ przeszukiwania grafu charakteryzuje się tym, że wierzchołki w odległości h od źródła są przeszukiwane wcześniej niż wierzchołki w odległości większej niż h. Możemy zatem powiedzieć, że najpierw przeglądane są wierzchołki w odległości 1, od wierzchołka źródłowego, następnie wierzchołki w odległości 2 oddalonej od wierzchołka źródłowego, itd.

Algorytm przeglądania grafu wszerz (BFS) możemy opisać przy pomocy procedury zamieszczonej w Tab. 5.5.

W algorytmie zostały przyjęte następujące oznaczenia:

- D – wektor, w którym przechowywane są dla każdego wierzchołka odległości od wierzchołka źródłowego s;
- Q – kolejka typu FIFO (elementy są przetwarzane w takiej kolejności w jakiej się pojawiły);
- pozostałe parametry są podobne jak przy opisie procedury przeglądania w głąb (zobacz Tab. 5.1).

Procedura *BFS* rozpoczyna swe działanie od nadania wartości początkowych swym parametrom (linie 1 – 8). Następnie do kolejki Q zostaje dodany wierzchołek źródłowy s. W linii 10 rozpoczyna się główna pętla procedury, która będzie tak długo wykonywana, dopóki kolejka Q nie będzie pusta. Następnie z tej kolejki pobierany jest wierzchołek u. Dla tak pobranego wierzchołka sprawdzani są wszyscy jego sąsiedzi $L[u]$. Jeśli sąsiedni wierzchołek v, nie był jeszcze zbadany (ma kolor biały), to zostaje pokolorowany na szaro; zostaje wpisana odległość od wierzchołka źródłowego w $D[v]$; zostaje wpisany poprzednik dla wierzchołka v i na sam koniec wierzchołek v trafia do kolejki Q typu FIFO (w kolejce przechowywane



Rys. 5.4: Ilustracja działania algorytmu przeszukiwania grafu wszerz (BFS). W wierzchołkach są wpisane odległości od wierzchołka źródłowego ($D[s]$). Stan kolejki Q, w poszczególnych krokach algorytmu, został przedstawiony poniżej każdego z rysunków. Krawędzie czerwone rozpinają drzewo BFS. Wierzchołek szary oznacza, że został on dodany do kolejki Q i według niej będzie przetwarzany w celu zbadania wszystkich swoich sąsiadów, którzy jeszcze nie zostali odwiedzeni. Wierzchołek czarny oznacza, że wszyscy sąsiedzi tego wierzchołka zostali już odwiedzeni (wierzchołek został usunięty z kolejki Q).

```
     Procedura BFS(G, s)
1.   Dla każdego v ∈ V wykonaj
2.       K[v] ← BIAŁY
3.       D[v] ← ∞
4.       P[v] ← NIL
5.   K[s] ←SZARY
6.   D[s] ← 0
7.   P[s] ← NIL
8.   Q ← ∅
9.   DodajDoKolejki(Q, s)
10.  Dopóki Q ≠ ∅ wykonaj
11.      u ← UsuńZKolejki(Q)
12.      Dla każdego v ∈ L[u] wykonaj        ▷ Zbadaj krawędź (u, v)
13.          Jeśli K[v] =BIAŁY to wykonaj
14.              K[v] ←SZARY
15.              D[v] ← D[v] + 1
16.              P[v] ← u
17.              DodajDoKolejki(Q, v)
18.      K[u] ←CZARNY
```

Tab. 5.5: Procedura BFS.

są wierzchołki o kolorze szarym). Po przetworzeniu wszystkich sąsiadów wierzchołka u, wierzchołek ten zostaje pokolorowany na czarno i tym samym nie będzie już więcej brany pod uwagę.

W Tab. 5.6 zostały pokazane najkrótsze ścieżki i ich odległości, względem wierzchołka startowego dla przykładu przedstawionego na Rys. 5.4.

Wierzchołek		Najkrótsza	Odległość
źródłowy	docelowy	ścieżka	$D[v]$
a	a	a	0
a	b	a, b	1
a	d	a, d	1
a	c	a, b, c	2
a	e	a, b, e	2
a	h	a, d, h	2
a	f	a, b, c, f	3
a	g	a, b, c, g	3
a	i	a, b, e, i	3

Tab. 5.6: Wyznaczenie najkrótszych ścieżek przy pomocy procedury *WypiszŚcieżkę* oraz czasy odwiedzin wierzchołków.

W Tab. 5.7 została pokazana wartość poprzedników dla przykładu przedstawionego na Rys. 5.4 w chwili zakończenia działania algorytmu.

Wierzchołek v	a	b	c	d	e	f	g	h	i
P[v]	NIL	a	b	a	b	c	c	d	e

Tab. 5.7: Poprzednicy dla drzewa przeszukiwań BFS.

6 Problem najkrótszej ścieżki

Jedno z podstawowych zagadnień związanych z grafami skierowanymi dotyczy znajdowania najkrótszej ścieżki pomiędzy dwoma wybranymi wierzchołkami. Można znaleźć szereg praktycznych problemów, gdzie zagadnienie odnalezienia najkrótszej ścieżki ma ogromne znaczenie. Możemy wyobrazić sobie, że musimy

dotrzeć z miasta A do miasta B w sposób możliwie najtańszy. Jeśli założymy, że koszt przejazdu jest związany z ilością przebytych kilometrów, rzeźbą terenu (jeśli pokonujemy wiele wzniesień to samochód spali więcej paliwa), wielkością opłat za pokonywane odcinki autostrad, wielkością natężenia ruchu itp., to zadanie najtańszego przejazdu możemy sprowadzić do postaci grafu skierowanego. Wierzchołki tego grafu, w tym wypadku, rozgraniczają odcinki pokonywanej trasy o stałych kosztach przejazdu. Koszty te można utożsamiać z wagami na krawędziach, pomiędzy poszczególnymi wierzchołkami. Oczywiście jeśli popatrzymy na atlas samochodowy, widzimy (załóżmy nietrywialny przypadek), że istnieje bardzo wiele możliwości pokonania interesującej nas trasy. Którą z nich należałoby wybrać? W jaki sposób? I czy sposób wyboru, faktycznie zagwarantuje nam optymalny wybór? Szukając odpowiedzi na te pytania, zapoznamy się z dwoma podstawowymi algorytmami, służącymi do rozwiązywania podobnych problemów grafowych: algorytmu Dijkstry oraz algorytm Warshalla. Problem opisany powyżej można prawidłowo zamodelować na podstawie grafu skierowanego, bez pętli i krawędzi wielokrotnych, tak więc nasze rozważania będą dotyczyć jedynie grafów, dla których $E \subset V \times V$. Funkcję ω można zdefiniować za pomocą macierzy sąsiedztw W, tak że $W(i,j)$ oznacza wagę połączenia pomiędzy wierzchołkiem v_i a wierzchołkiem v_j. Algorytmy poszukiwania minimalnych wag (zarówno Dijkstry jaki i algorytm Warshalla), najpierw przeglądają drogi o długości 1, a następnie systematycznie sprawdzają coraz to dłuższe drogi między wierzchołkami. W czasie działania, algorytmy te znajdują drogi o coraz to mniejszych wagach, tak że w chwili zakończenia znamy najmniejsze wagi połączeń.

6.1 Algorytm Dijkstry

Pierwszy z przedstawianych algorytmów znajduje wagi minimalne dróg z wybranego wierzchołka źródłowego s do wszystkich innych wierzchołków grafu skierowanego G.

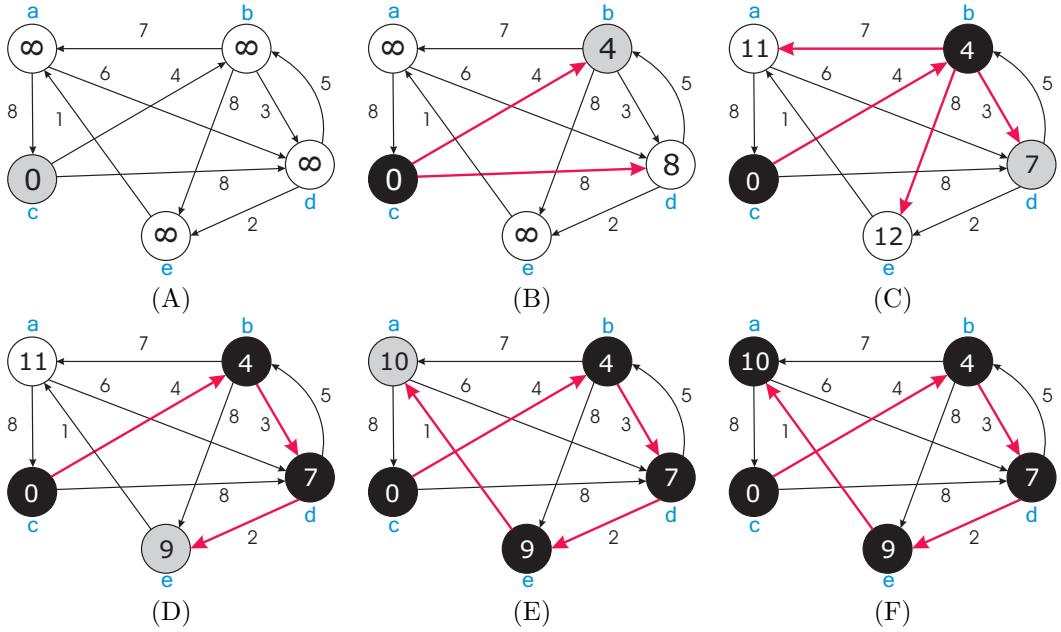
Algorytm działania Dijkstry możemy opisać przy pomocy procedury zamieszczonej w Tab. 6.1.

```
     Procedura Dijkstra(G, W, s)
1.   Dla każdego v ∈ V wykonaj
2.       D[v] ← ∞
3.       P[v] ← NIL
4.   D[s] ← 0
5.   S ← ∅
6.   Q ← V
7.   Dopóki Q ≠ ∅ wokonaj
8.       u ← WybierzMin(Q)
9.       S ← S ∪ {u}
10.      Dla każdego v ∈ L[u] wykonaj          ▷ Zbadaj krawędź (u, v)
11.          Jeśli D[u] + W(u, v) < D[v] to wykonaj
12.              D[v] ← D[v] + W(u, v)
13.              P[v] ← u
```

Tab. 6.1: Procedura Dijkstra.

W algorytmie przyjęto następujące oznaczenia:

- W — macierz sąsiedztw grafu G;
- s — wierzchołek źródłowy, od którego zaczynamy szukać najkrótszych ścieżek do pozostałych wierzchołków grafu G;
- S — zbiór zawierający wierzchołki, dla których wagi najkrótszych ścieżek ze źródła s zostały już obliczone;
- D — wektor $|V|$ elementowy odpowiadający ilości wierzchołków w grafie G, w którym przechowywane są wartości oszacowań wag dla najkrótszej ścieżki wyznaczonej z wierzchołka źródłowego s do danego wierzchołka v. Zapis $D[v]$ oznacza, wartość oszacowanej ścieżki w wierzchołku v z wierzchołka źródłowego s;
- Q — kolejka priorytetowa z operacją minimum względem wartości wektora D;



Rys. 6.1: Ilustracja działania algorytmu Dijkstry. Oszacowania najkrótszych ścieżek znajdują się w wierzchołkach. Czarne wierzchołki należą do zbioru S. Szary wierzchołek oznacza, że wierzchołek ten został wybrany do przetwarzania w następnym kroku algorytmu. Białe wierzchołki należą do kolejki priorytetowej Q z operacją minimum względem wartości D. Krawędzie czerwone wskazują wartości poprzedników.

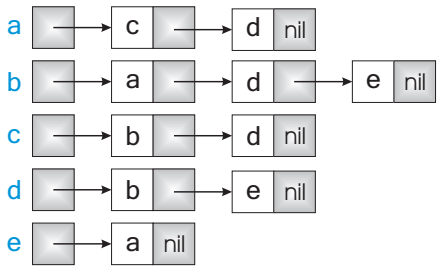
- $L[u]$ — lista sąsiedztw dla wierzchołka u.
- P – wektor, w której pamiętany jest poprzednik danego wierzchołka.

Algorytm rozpoczyna swe działanie od przypisania wagi o wartości nieskończonej, każdemu wierzchołkowi należącemu do grafu G, w wektorze D oraz ustawienia poprzedników w liście P na wartość NIL. Następnie wierzchołkowi źródłowemu s w tablicy D nadana jest wartość zero. W linii nr 5 inicjalizujemy zbiór wierzchołków S na zbiór pusty, a w linii nr 6 dodajemy wszystkie wierzchołki (łącznie ze źródłowym) do kolejki priorytetowej Q (z operacją minimum względem wartości oszacowań wag najkrótszej ścieżki zawartej w wektorze D).

W liniach 7–13 wykonywana, jest pętla tak długo, dopóki kolejka Q nie jest pusta. Pierwszym krokiem w tej pętli jest wybranie wierzchołka u z kolejki Q. Zapis, $WybierzMin(Q)$ oznacza wybranie wierzchołka z kolejki priorytetowej Q. Z własności, tejże kolejki, wynika, że zostanie wybrany wierzchołek o najmniejszej wadze (najkrótszej ścieżce, najmniejszej wartości) z wektora D, spośród wszystkich wierzchołków będący w tej kolejce. Następnie dodajemy tak wybrany wierzchołek u do zbioru S (co oznacza, że waga dla tak wybranego wierzchołka jest już minimalna) i wierzchołek ten nie będzie już dalej rozpatrywany. Drugim krokiem tej pętli jest wyliczenie nowych wartości wag (oszacowanie najkrótszej ścieżki) w wektorze D, dla każdego wierzchołka należącego do listy sąsiedztwa wierzchołka u oraz wpisanie poprzednika dla wierzchołka v w liście P, jeśli warunek w linii 11 jest spełniony.

Rozważany przykład, gdzie macierz sąsiedztw i lista sąsiedztw jest dana na Rys. 6.2. Załóżmy, że

W	a	b	c	d	e
a	∞	∞	8	6	∞
b	7	∞	∞	3	8
c	∞	4	∞	8	∞
d	∞	5	∞	∞	2
e	1	∞	∞	∞	∞



Rys. 6.2: Macierz sąsiedztw oraz lista sąsiedztw rozważanego grafu.

wierzchołkiem startowym będzie wierzchołek c. Algorytm rozpoczyna swe działanie od przypisania wartości początkowych w liniach 1 – 6. Wierzchołki oznaczone kolorem białym należą do kolejki Q (zobacz

Rys. 6.1. Uwaga: dla zwiększenia czytelności szarym kolorem oznaczono wierzchołki, które w danym kroku algorytmu mają najmniejszą wartość). Następnie algorytm wybiera z kolejki Q wierzchołek o najmniejszej wartości (będzie to, w tym przypadku, wierzchołek startowy c) i dodaje go do zbioru S (na rysunku (B) oznaczony jest kolorem czarnym). Następnie algorytm przegląda, dla tak wybranego wierzchołka, czy przechodząc przez listę swoich sąsiadów, można znaleźć krótszą drogę od obecnie znalezionej (aktualne oszacowanie drogi jest zapisane w wektorze D — na Rys. 6.1 oszacowanie to jest wpisane w wierzchołkach). Przeglądając sąsiadów wierzchołka źródłowego (wierzchołki b i d) uaktualniamy oszacowanie drogi dla tych wierzchołków, gdyż w tych wierzchołkach wartości oszacowań były nieskończone, zatem warunek w linii 11 został spełniony. Pojawiły się wartości 4 i 8 odpowiednio dla wierzchołka b i d, oraz lista ich poprzedników wskazuje na wierzchołek źródłowy c (krawędzie czerwone na rysunku (B)). Następnie algorytm rozpoczyna główną pętlę od nowa, wybierając z kolejki Q wierzchołek o najmniejszej wartości – będzie to wierzchołek b. Procedura ta powtarza się iteracyjne dla każdego wierzchołka, za każdym razem wybierając wierzchołek o najmniejszej wartości oszacowania drogi z wierzchołka źródłowego. Przebieg całego algorytmu został zilustrowany na Rys. 6.1.

W tabeli 6.1 została pokazana zmiana wartości oszacowań najkrótszej ścieżki w wektorze D dla poszczególnych iteracji algorytmu. Można także zauważyć na podstawie tej tabeli, że wartości umieszczone w wektorze D stanowią ciąg nierosnący.

Nr iteracji	D[a]	D[b]	D[c]	D[d]	D[e]
0	∞	∞	**0***	∞	∞
1	∞	**4***	0	8	∞
2	11	4	0	**7***	12
3	11	4	0	7	**9***
4	**10***	4	0	7	9
5	10	4	0	7	9

Tab. 6.2: Zmiana wartości oszacowań najkrótszej ścieżki w wektorze D. Czerwonym kolorem oraz gwiazdką oznaczono znalezione rozwiązanie (została znaleziona najkrótsza ścieżka z wierzchołka źródłowego do rozpatrywanego wierzchołka). Kolorem niebieskim oznaczono te wierzchołki, które należą do zbioru S jako wierzchołki, dla których znaleziono już optymalne rozwiązanie (są to wierzchołki pokolorowane na czarno – zobacz Rys. 6.1).

W tabeli 6.3 zostały pokazane najkrótsze uzyskane ścieżki i ich oszacowania, względem wierzchołka źródłowego.

Wierzchołek		Najkrótsza	Oszacowanie
źródłowy	docelowy	ścieżka	ścieżki
c	c	c	0
c	b	c, b	4
c	d	c, b, d	7
c	e	c, b, d, e	9
c	a	c, b, d, e, a	10

Tab. 6.3: Wyznaczenie najkrótszej ścieżki i jej oszacowania z wierzchołka źródłowego do docelowego.

W Tab. 6.4 została pokazana wartość poprzedników dla przykładu przedstawionego na Rys. 6.1 w chwili zakończenia działania algorytmu.

Wierzchołek v	a	b	c	d	e
P[v]	e	c	NIL	b	d

Tab. 6.4: Poprzednicy dla drzewa przeszukiwań BFS.

6.2 Algorytm Warshalla

Algorytm Dijkstry znajduje wagi dróg minimalnych prowadzących z danego wierzchołka. Aby znaleźć minimalne drogi dla wszystkich par wierzchołków można po prostu wykonać ten algorytm n razy, wybierając za każdym razem inny wierzchołek startowy. Istnieje jednak inny algorytm, który daje w wyniku wszystkie możliwe drogi o najmniejszej wadze, a przy tym jest bardzo łatwy do zaprogramowania.

Problem opisany przy pomocy algorytmu Warshalla można prawidłowo zamodelować na podstawie grafu skierowanego, bez pętli i krawędzi wielokrotnych, tak więc nasze rozważania będą dotyczyć jedynie grafów, dla których $E \subset V \times V$. Funkcję ω można zdefiniować za pomocą macierzy sąsiedztw W, tak że $W(u, v)$ oznacza wagę połączenia pomiędzy wierzchołkiem u a wierzchołkiem v

$$W(u,w) = \begin{cases} 0, & u = v \\ \infty, & \text{nie ma krawędzi między } u \text{ a } v; \\ w_{uv}, & \text{istnieje krawędź między } u \text{ a } v \text{ o wadze } w_{uv} \end{cases}$$

Podobnie jak algorytm Dijkstry, algorytm Warshalla, przegląda krawędzie grafu szukając najkrótszej drogi przechodzącej przez wierzchołki v, k, u, dla aktualnie pobranego wierzchołka k z kolejki Q o najmniejszej wartości. Algorytm ten opiera się na iteracyjny przetwarzaniu macierzy wag oszacowań najkrótszych ścieżek D oraz systematycznym umieszczaniu odwiedzonych wierzchołków w zbiorze S. Początkowo macierzy D zostaje przypisana wartość macierz sąsiedztw W, macierz P przechowująca poprzedników wierzchołków v i u jest wypełniona wartością pustą (NIL). Reszta algorytmu przebiega w podobny sposób jak miało to miejsce w przypadku algorytmu Dijkstry.

Algorytm działania Warshalla możemy opisać przy pomocy procedury zamieszczonej w Tab. 6.5.

```
     Procedura Warshall(W)
1.   Dla każdego v, u ∈ V wykonaj
2.       P(v, u) ← NIL
3.       D(v, u) ← W(v, u)
4.   S ← ∅
5.   Q ← V
6.   Dopóki Q ≠ ∅ wokonaj
7.       k ← WybierzMin(Q)
8.       S ← S ∪ {k}
9.       Dla każdego v, u ∈ V wykonaj
10.          Jeśli D(v, k) + W(k, u) < D(v, u) to wykonaj
11.              D(v, u) ← D(v, k) + W(k, u)
12.              P(v, u) ← k
```

Tab. 6.5: Procedura Warshall.

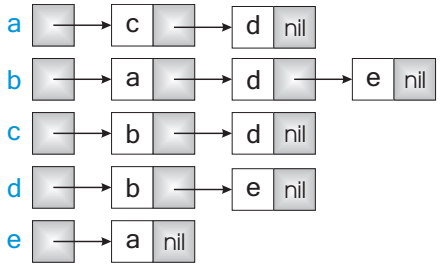
Algorytm ten może zostać zapisany w bardzo prostej, macierzowej formie pokazanej w Tab 6.6.

```
     Procedura WarshallMac(W)
1.   D ← W
2.   n ← rozmiar(W)
3.   Dla k = 1 do n wykonaj
4.       Dla v = 1 do n wykonaj
5.           Dla u = 1 do n wykonaj
6.               Jeśli D(v, k) + W(k, u) < D(v, u) to wykonaj
7.                   D(v, u) ← D(v, k) + W(k, u)
```

Tab. 6.6: Procedura WarshallMac – wersja macierzowa.

Rozważany przykład, gdzie macierz sąsiedztw i lista sąsiedztw jest dana na Rys. 6.2.
Sesja obliczeniowa algorytmu Warshalla dla takiego grafu skierowanego, przedstawia się w postaci

W	a	b	c	d	e
a	0	∞	8	6	∞
b	7	0	∞	3	8
c	∞	4	0	8	∞
d	∞	5	∞	0	2
e	1	∞	∞	∞	0



Rys. 6.3: Macierz sąsiedztw oraz lista sąsiedztw rozważanego grafu.

kolejnych modyfikacji macierzy wag:

$$D_0 = \begin{bmatrix} 0 & \infty & 8 & 6 & \infty \\ 7 & 0 & \infty & 3 & 8 \\ \infty & 4 & 0 & 8 & \infty \\ \infty & 5 & \infty & 0 & 2 \\ 1 & \infty & \infty & \infty & 0 \end{bmatrix}, \quad D_1 = \begin{bmatrix} 0 & \infty & 8 & 6 & \infty \\ 7 & 0 & 15 & 3 & 8 \\ \infty & 4 & 0 & 8 & \infty \\ \infty & 5 & \infty & 0 & 2 \\ 1 & \infty & 9 & 7 & 0 \end{bmatrix}, \quad D_2 = \begin{bmatrix} 0 & \infty & 8 & 6 & \infty \\ 7 & 0 & 15 & 3 & 8 \\ 11 & 4 & 0 & 7 & 12 \\ 12 & 5 & 20 & 0 & 2 \\ 1 & \infty & 9 & 7 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 12 & 8 & 6 & 20 \\ 7 & 0 & 15 & 3 & 8 \\ 11 & 4 & 0 & 7 & 12 \\ 12 & 5 & 20 & 0 & 2 \\ 1 & 13 & 9 & 7 & 0 \end{bmatrix}, \quad D_4 = \begin{bmatrix} 0 & 11 & 8 & 6 & 8 \\ 7 & 0 & 15 & 3 & 5 \\ 11 & 4 & 0 & 7 & 9 \\ 12 & 5 & 20 & 0 & 2 \\ 1 & 12 & 9 & 7 & 0 \end{bmatrix}, \quad D_5 = \begin{bmatrix} 0 & 11 & 8 & 6 & 8 \\ 6 & 0 & 14 & 3 & 5 \\ 10 & 4 & 0 & 7 & 9 \\ 3 & 5 & 11 & 0 & 2 \\ 1 & 12 & 9 & 7 & 0 \end{bmatrix}.$$

Prześledzimy sesję obliczeniową dla ścieżki pomiędzy wierzchołkami c i a (wiersz 3, kolumna 1 w macierzy sąsiedztw $D_i, i = 0, 1, \ldots, k$, gdzie $k = n$). Początkowo widzimy że waga tej ścieżki jest równa ∞, co oznacza, że na tym etapie algorytmu (k=0) nie istnieje ścieżka między tymi wierzchołkami (D_0). W kroku $k = 1$ nadal nic się nie zmienia, dopiero w drugim kroku $k = 2$ w macierzy sąsiedztw pojawia się połączenie między wierzchołkiem c i a. W krokach 3 i 4 nic się nie zmienia dla tego połączenia. Dopiero w kroku 5 pojawia się lepsze połączenie między tymi wierzchołkami o wartości 10. Dla porównania uzyskanych wyników, proszę porównać wyniki jakie uzyskano dla tego samego grafu w przypadku algorytmu Dijkstry (zobacz Tab. 6.1).

Literatura

[1] L.Banachowski i in. *Algorytmy i struktury danych*; WNT, 1996.

[2] T.H.Cormen i in. *Wprowadzenie do algorytmów*, WNT, 2000

[3] A.Drozdek *Struktury danych w jezyku C*, WNT, 1996

[4] D.E.Knuth *Sztuka programowania*, WNT, 2002

[5] R.Sedgewick *Algorytmy w C++*, Wydawnictwo RM, 1999

[6] P.Wróblewski, *Algorytmy, struktury danych i techniki programowania*, HELION, 1996

[7] N. Wirth, *Algorytmy + struktury danych = programy*, WNT, 2001