Matematyka Dyskretna

Wykład 10

Algorytmy grafowe

Wojciech Kordecki
wojciech.kordecki@collegiumwitelona.pl

Collegium Witelona
Wydział Nauk Technicznych i Ekonomicznych
Zakład Informatyki

Semestr letni 2023/24

Przedstawione w tym wykładzie algorytmy grafowe są jedynie wybranymi algorytmami, które są wykorzystane w dalszej części. Pominięto wiele spośród ważnych i szeroko stosowanych. Jest jednakże do dyspozycji szeroki wybór książek poświęconych algorytmom, a zwłaszcza algorytmom grafowym. Są to między innymi T. H. Cormen, C. E. Leiserson, R. L. Rivest i C. Stein *Wprowadzenie do algorytmów* [CLRS] i R. Sedgewick *Algorytmy w C++. Część 5. Grafy* [Se] .

[CLRS] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein,
Wprowadzenie do algorytmów, wydanie 2,
WNT, Warszawa 2004.

[L] W. Lipski,
Kombinatoryka dla programistów,
WNT, Warszawa 2009.

[Se] R. Sedgewick,
Algorytmy w C++. Część 5. Grafy,
Wyd. RM, Warszawa 2003

W głąb i wszerz

Wiele algorytmów grafowych polega na systematycznym przeszukiwaniu wierzchołków grafu tak, by każdy wierzchołek został odwiedzony dokładnie raz. Najpopularniejsze to algorytm przeszukiwania w głąb (DBF) i algorytm przeszukiwania wszerz (BFS). Możemy je stosować dla dowolnych grafów – skierowanych i nieskierowanych. W dalszych rozważaniach ograniczymy się jednak tylko do grafów nieskierowanych. W wyniku działania algorytmu dostajemy drzewo przeszukiwań. Dwie takie procedury są dobrze znane – metoda przeszukiwania w głąb i metoda przeszukiwania wszerz. W obydwu metodach odwiedzamy wszystkie wierzchołki grafu spójnego, ale na ogół w różnej kolejności, przechodząc po różnych krawędziach.
W książce W. Lipskiego *Kombinatoryka dla programistów* [L] można znaleźć ścisłe, a jednocześnie przystępne omówienie algorytmów przeszukiwania.

FIFO – LIFO

Kolejka – First In First Out, FIFO.

Element, który pierwszy jest pobrany do kolejki, pierwszy jest z niej usunięty.

FIFO – LIFO

Kolejka – First In First Out, FIFO.

Element, który pierwszy jest pobrany do kolejki, pierwszy jest z niej usunięty.

Stos – First In Last Out, LIFO.

Element, który pierwszy jest położony na stos, ostatni jest z niego usunięty.

BFS

Algorytm przeszukiwania grafu wszerz rozpoczynamy od ustalonego wierzchołka u. Wierzchołek u, który zostaje oznaczony jako odwiedzony, umieszczamy w kolejce, a następnie przechodzimy do wszystkich nieodwiedzonych wierzchołków z nim sąsiednich, umieszczamy je w kolejce i oznaczamy jako odwiedzone. Jeśli takich nieodwiedzonych wierzchołków nie ma, to usuwamy z kolejki pierwszy wierzchołek i przechodzimy z niego do wszystkich wierzchołków z nim sąsiednich. Procedurę powtarzamy, aż kolejka będzie pusta. Wtedy wszystkie wierzchołki są odwiedzone.

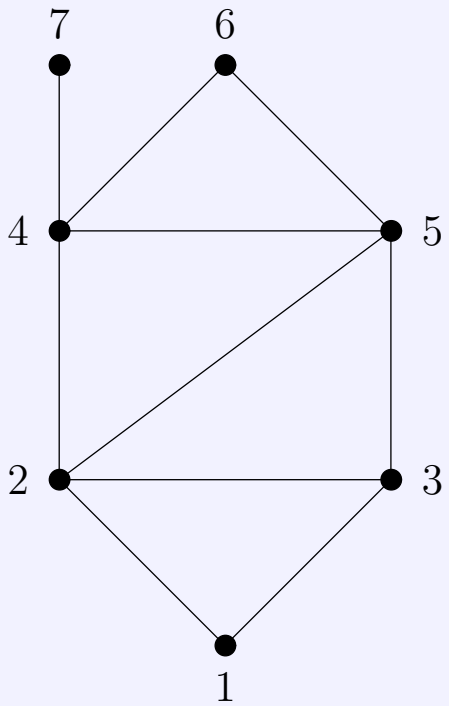
BFS

Algorytm przeszukiwania grafu wszerz rozpoczynamy od ustalonego wierzchołka u. Wierzchołek u, który zostaje oznaczony jako odwiedzony, umieszczamy w kolejce, a następnie przechodzimy do wszystkich nieodwiedzonych wierzchołków z nim sąsiednich, umieszczamy je w kolejce i oznaczamy jako odwiedzone. Jeśli takich nieodwiedzonych wierzchołków nie ma, to usuwamy z kolejki pierwszy wierzchołek i przechodzimy z niego do wszystkich wierzchołków z nim sąsiednich. Procedurę powtarzamy, aż kolejka będzie pusta. Wtedy wszystkie wierzchołki są odwiedzone.

Krawędzie grafu, po których przechodzimy do kolejnych wierzchołków, tworzą drzewo przeszukiwań grafu wszerz. Algorytm ten nosi nazwę algorytmu przeszukiwania wszerz (ang. *Breadth First Search* – BFS).

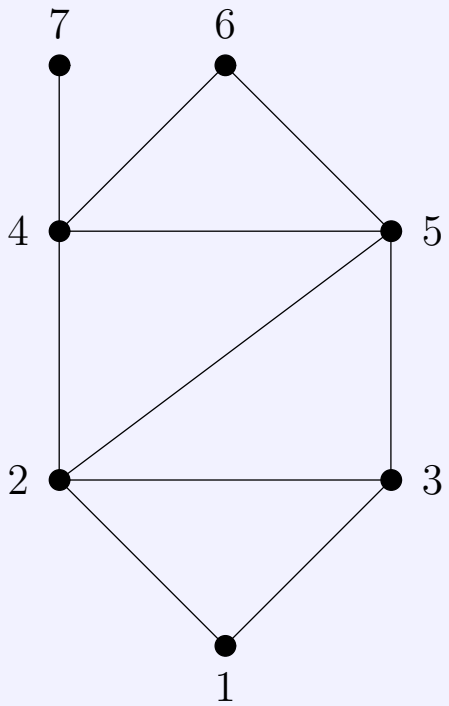
Przykład

Przeszukajmy wszerz graf G.
Dla ustalenia uwagi przyjmijmy, że gdy mamy możliwość przejścia w kolejnym kroku do więcej niż jednego wierzchołka, to wybieramy wierzchołek o niższym numerze.



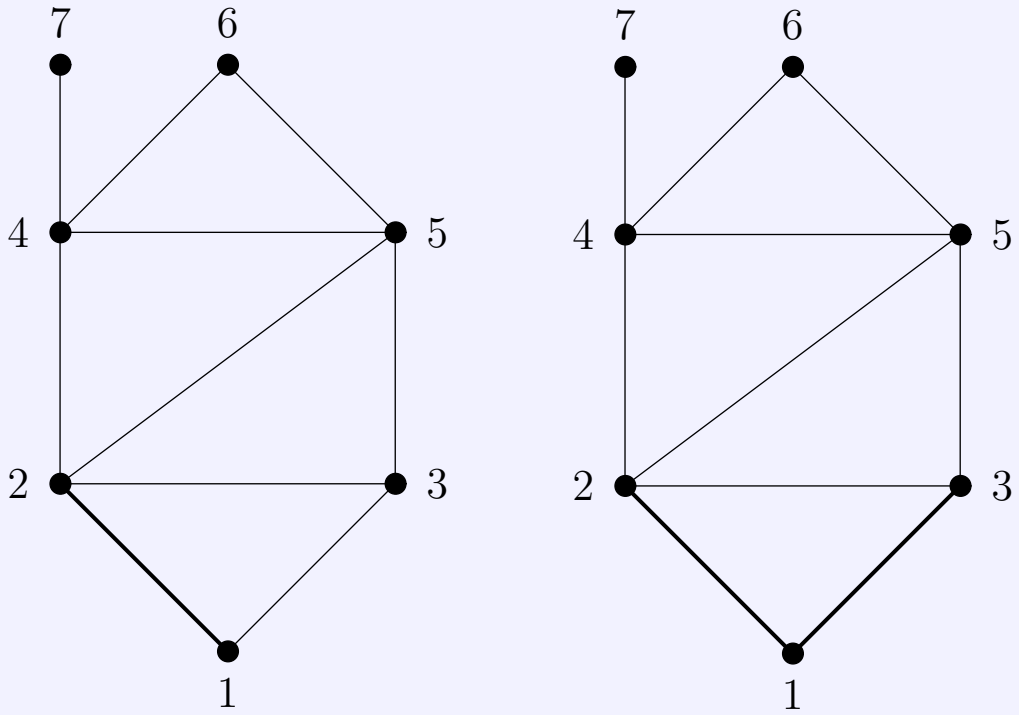
Krok 1

Zaczynamy z wierzchołka 1, wstawiamy go do kolejki $Q = (1)$ i zaznaczamy go jako odwiedzony: $W = \{1\}$.



Krok 2

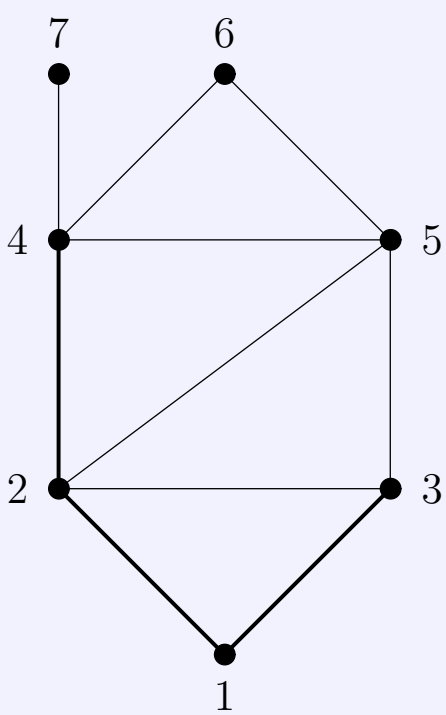
Odwiedzamy wierzchołki sąsiednie 2 oraz 3, wtedy $Q = (1, 2, 3)$, $W = \{1, 2, 3\}$,
krawędzie drzewa $T = \{(1, 2), (1, 3)\}$.

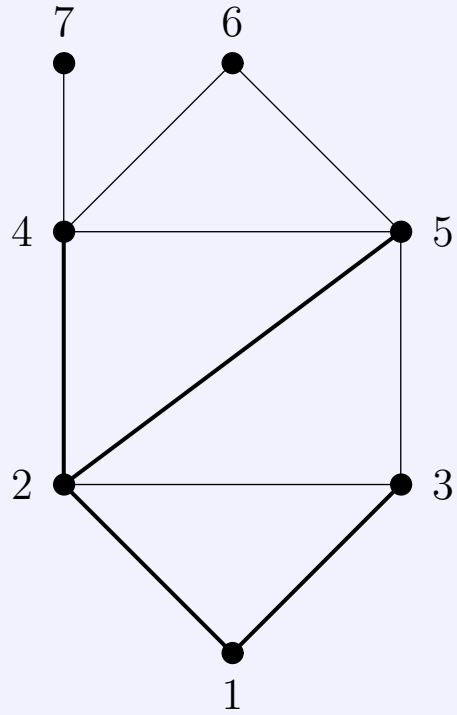


Nie ma innych nieodwiedzonych wierzchołków sąsiednich z 1, więc usuwamy go z kolejki: $Q = (2, 3)$.

Krok 3

Odwiedzamy nieodwiedzone wierzchołki sąsiednie z 2: wtedy $Q = (2, 3, 4, 5)$, $W = \{1, 2, 3, 4, 5\}$,
krawędzie drzewa $T = \{(1, 2), (1, 3), (2, 4) . (2, 6)\}$.

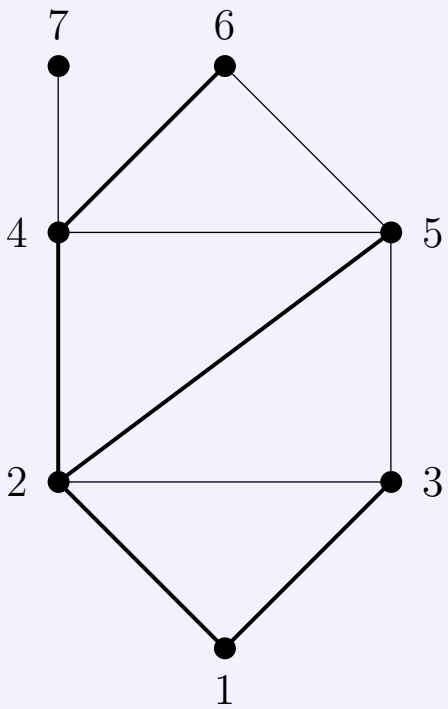


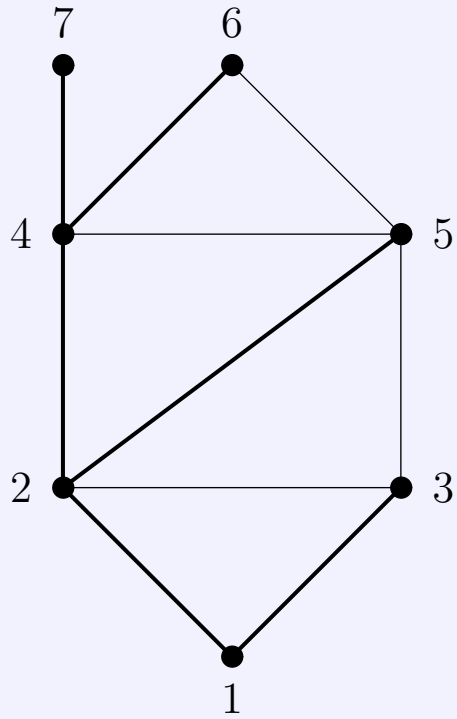


Nie ma innych nieodwiedzonych wierzchołków sąsiednich ani z 2 ani z 3, więc usuwamy je z kolejki: $Q = (4, 5)$

Krok 4

Odwiedzamy nieodwiedzone wierzchołki sąsiednie z 4: wtedy
$Q = (4, 5, 6, 7)$ $W = \{1, 2, 3, 4, 5, 6, 7\}$,
$T = \{(1, 2), (1, 3), (2, 4), (2, 5), (4, 6), (4, 7)\}$.

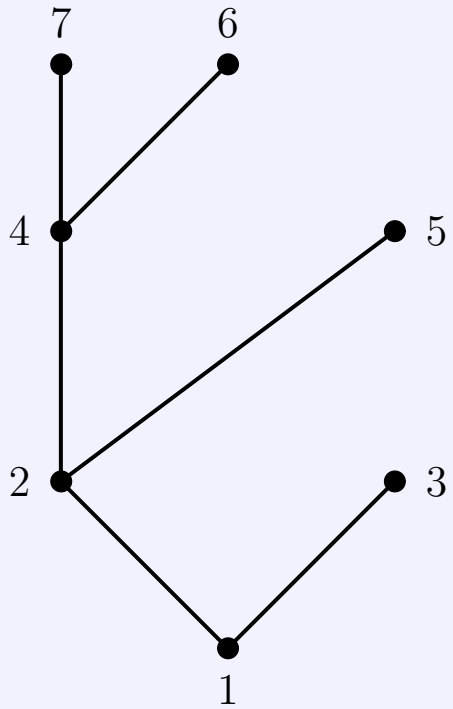




Nie ma innych nieodwiedzonych wierzchołków sąsiednich ani z 4 ani z 5, więc usuwamy je z kolejki: $Q = (6, 7)$

Krok 5

Nie ma innych wierzchołków sąsiednich ani z 6 ani z 7, więc usuwamy je z kolejki: $Q = \emptyset$. Koniec algorytmu.



DFS

Algorytm przeszukiwania grafu w głąb rozpoczynamy od ustalonego wierzchołka u. Wierzchołek u zostaje oznaczony jako odwiedzony i umieszczony na stosie, po czym przechodzimy do nieodwiedzonego wierzchołka sąsiedniego (powiedzmy, wierzchołka v). Jeśli takiego wierzchołka nie ma, to usuwamy u ze stosu i powtarzamy procedurę, rozpoczynając od wierzchołka ze szczytu stosu. Procedurę powtarzamy, aż stos będzie pusty.

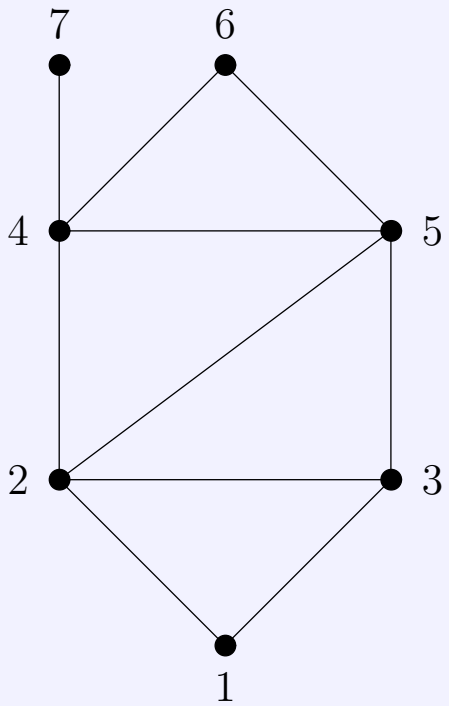
DFS

Algorytm przeszukiwania grafu w głąb rozpoczynamy od ustalonego wierzchołka u. Wierzchołek u zostaje oznaczony jako odwiedzony i umieszczony na stosie, po czym przechodzimy do nieodwiedzonego wierzchołka sąsiedniego (powiedzmy, wierzchołka v). Jeśli takiego wierzchołka nie ma, to usuwamy u ze stosu i powtarzamy procedurę, rozpoczynając od wierzchołka ze szczytu stosu. Procedurę powtarzamy, aż stos będzie pusty.

Krawędzie grafu, po których przechodzimy do kolejnych wierzchołków, tworzą drzewo przeszukiwań grafu w głąb. Algorytm ten nosi nazwę algorytmu przeszukiwania w głąb (ang. *Depth First Search* – DFS).

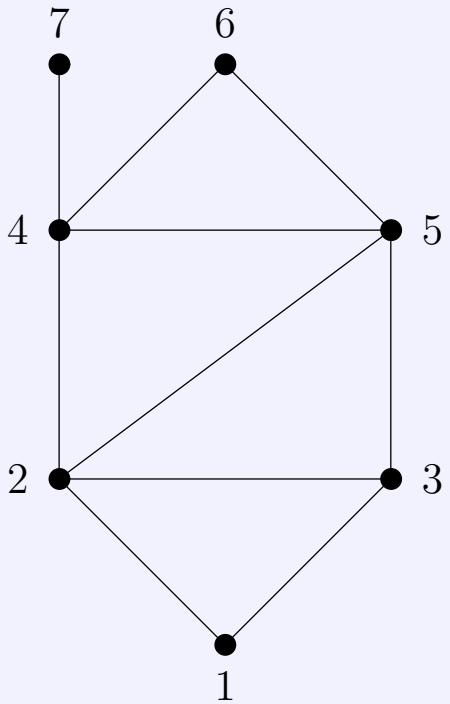
Przykład

Przeszukajmy w głąb graf G.
Dla ustalenia uwagi przyjmijmy, że gdy mamy możliwość przejścia w kolejnym kroku do więcej niż jednego wierzchołka, to wybieramy wierzchołek o niższym numerze.



Stos

W kolejnych krokach:



stos	odwiedzone
1	1
1,2	1,2
1,2,3	1,2,3
1,2,3,5	1,2,3,5
1,2,3,5,4	1,2,3,5,4
1,2,3,5,4,6	1,2,3,4,5,6
1,2,3,5,4	1,2,3,4,5,6
1,2,3,5,4,7	1,2,3,4,5,6,7

Uwagi

Algorytmy BFS i DFS są jednymi z najważniejszych algorytmów w teorii grafów, gdyż są szkieletem wielu innych. Z tego też względu są dokładnie omówione w wielu podręcznikach.

Uwagi

Algorytmy BFS i DFS są jednymi z najważniejszych algorytmów w teorii grafów, gdyż są szkieletem wielu innych. Z tego też względu są dokładnie omówione w wielu podręcznikach.

Zauważmy, że wysokość drzewa otrzymanego algorytmem BFS jest nie większa niż wysokość drzewa otrzymanego algorytmem DFS. Skrajnym przypadkiem jest graf pełny K_n. Wynikiem działania BFS jest gwiazda S_n, a DFS – droga P_n.

Sformułowanie problemu

Niech $G = (V, E)$ – graf nieskierowany, prosty, bez pętli, spójny. Określona jest funkcja wagi w na zbiorze E.

$$w : E \to \mathbb{R}^+$$

czyli $w(e) > 0$ dla każdej krawędzi $e \in E$.

Sformułowanie problemu

Niech $G = (V, E)$ – graf nieskierowany, prosty, bez pętli, spójny. Określona jest funkcja wagi w na zbiorze E.

$$w : E \to \mathbb{R}^+$$

czyli $w(e) > 0$ dla każdej krawędzi $e \in E$.

Szukamy drzewa spinającego T takiego, że waga

$$W(G) = \sum_{e \in T} w(e) = \min .$$

Drzewo spinające o minimalnej wadze nazywamy minimalnym drzewem spinającym.

Sformułowanie problemu

Niech $G = (V, E)$ – graf nieskierowany, prosty, bez pętli, spójny. Określona jest funkcja wagi w na zbiorze E.

$$w : E \to \mathbb{R}^+$$

czyli $w(e) > 0$ dla każdej krawędzi $e \in E$.

Szukamy drzewa spinającego T takiego, że waga

$$W(G) = \sum_{e \in T} w(e) = \min .$$

Drzewo spinające o minimalnej wadze nazywamy minimalnym drzewem spinającym.

Jeżeli graf G nie jest spójny, to szukamy minimalnego lasu spinającego.

Algorytm Kruskala

W grafie z wagami sortujemy krawędzie wg rosnących wag.
Do drzewa T dołączamy kolejne krawędzie tak, aby nie utworzyć cyklu. Tak utworzone drzewo ma minimalną wagę $W(G)$.

Algorytm Kruskala

W grafie z wagami sortujemy krawędzie wg rosnących wag.
Do drzewa T dołączamy kolejne krawędzie tak, aby nie utworzyć cyklu. Tak utworzone drzewo ma minimalną wagę $W(G)$.

Uwaga 1. Jeśli dwie krawędzi mają takie same wagi, sortujemy je w dowolnej kolejności.

Uwaga 2. Jeśli istnieją krawędzie o tych samych wagach, to minimalne drzewo spinające nie jest wyznaczone jednoznacznie.

Algorytm Prima

Inna nazwa: algorytm najbliższego sąsiada.

1. Niech $T = \emptyset$.
2. Wybieramy dowolny wierzchołek początkowy $v_1 \in T$ i dołączamy go do T.
3. Wybieramy dowolną krawędź incydentną z v_1 o najniższej wadze i drugi koniec tej krawędzi dołączamy do zbioru T.
4. Kolejno dla $i = 2, \ldots, n-1$ wybieramy dowolną krawędź e_i incydentną z dowolnym wierzchołkiem ze zbioru T o najniższej wadze taką, że jej drugi koniec nie należy do T, i dołączamy ten koniec do zbioru T.
5. Algorytm kończy się, gdy $T = V$.

Uwagi

Algorytm Prima wyznaczania minimalnego drzewa spinającego grafu, nie wymaga ani sortowania krawędzi ani sprawdzania istnienia cykli w każdym kroku.

Uwagi

Algorytm Prima wyznaczania minimalnego drzewa spinającego grafu, nie wymaga ani sortowania krawędzi ani sprawdzania istnienia cykli w każdym kroku.

Algorytm Prima wyznacza minimalne drzewo spinające zawierające krawędzie $e_1, e_2, \ldots, e_{n-1}$.

Uwagi

Algorytm Prima wyznaczania minimalnego drzewa spinającego grafu, nie wymaga ani sortowania krawędzi ani sprawdzania istnienia cykli w każdym kroku.

Algorytm Prima wyznacza minimalne drzewo spinające zawierające krawędzie $e_1, e_2, \ldots, e_{n-1}$.

Najgorszy przypadek dla powyższego algorytmu pojawia się, gdy graf G jest grafem pełnym. W takiej sytuacji w każdym kroku algorytmu musimy przeprowadzić maksymalną liczbę porównań, aby znaleźć najbliższy sąsiedni wierzchołek.

Uwagi

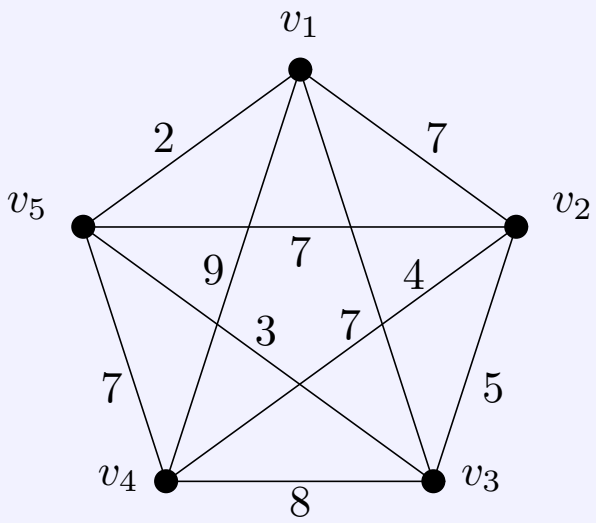
Algorytm Prima wyznaczania minimalnego drzewa spinającego grafu, nie wymaga ani sortowania krawędzi ani sprawdzania istnienia cykli w każdym kroku.

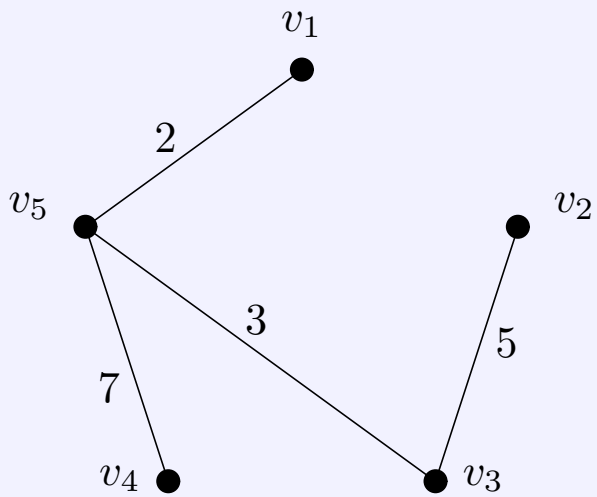
Algorytm Prima wyznacza minimalne drzewo spinające zawierające krawędzie $e_1, e_2, \ldots, e_{n-1}$.

Najgorszy przypadek dla powyższego algorytmu pojawia się, gdy graf G jest grafem pełnym. W takiej sytuacji w każdym kroku algorytmu musimy przeprowadzić maksymalną liczbę porównań, aby znaleźć najbliższy sąsiedni wierzchołek.

Jeśli G jest grafem niespójnym, to dla wyznaczenia minimalnego lasu należy algorytm Prima zastosować dla każdej składowej oddzielnie.

Przykład





Algorytmy zachłanne

Algorytm zachłanny (ang. greedy algorithm) – w każdym kroku algorytmu musimy dokonać jednego z wielu możliwych wyborów. Stosując strategię zachłanną, dokonujemy wyboru, który jest w danej chwili najlepszy.

Algorytmy zachłanne

Algorytm zachłanny (ang. greedy algorithm) – w każdym kroku algorytmu musimy dokonać jednego z wielu możliwych wyborów. Stosując strategię zachłanną, dokonujemy wyboru, który jest w danej chwili najlepszy.

Algorytm Kruskala i algorytm Prima są przykładami zachłannych algorytmów optymalizacyjnych

Zachłanność może być szkodliwa

Dana jest macierz:

$$\begin{bmatrix} 1 & 3 & 3 \\ 2 & 49 & 51 \\ 4 & 51 & 50 \end{bmatrix}$$

Mamy wybrać trzy elementy, wszystkie z różnych wierszy i kolumn.

Zachłanność może być szkodliwa

Dana jest macierz:

$$\begin{bmatrix} 1 & 3 & 3 \\ 2 & 49 & 51 \\ 4 & 51 & 50 \end{bmatrix}$$

Mamy wybrać trzy elementy, wszystkie z różnych wierszy i kolumn.

Zachłannie:

$$a_{11} + a_{22} + a_{33} = 100.$$

Zachłanność może być szkodliwa

Dana jest macierz:

$$\begin{bmatrix} 1 & 3 & 3 \\ 2 & 49 & 51 \\ 4 & 51 & 50 \end{bmatrix}$$

Mamy wybrać trzy elementy, wszystkie z różnych wierszy i kolumn.

Zachłannie:

$$a_{11} + a_{22} + a_{33} = 100.$$

Optymalnie:

$$a_{21} + a_{12} + a_{33} = 55.$$

Historia

Algorytm zachłanny generujący minimalne drzewo spinające zawdzięczamy J. B. Kruskalowi (1956).
Algorytm wyznaczający minimalne drzewo spinające metodą najbliższego sąsiada został odkryty w 1930 roku przez czeskiego matematyka V. Jarníka, a następnie ponownie odkryty w 1957 roku przez R. C. Prima oraz niezależnie w 1959 roku przez holenderskiego informatyka E. Dijkstrę.

Długość ważona drogi

Długością ważoną drogi $(u_0, u_1, \ldots, u_n)$ w grafie $G(V, E)$ o nieujemnych wagach krawędzi (łuków) $e \in E$ jest suma wag krawędzi tej drogi.

Długość ważona drogi

Długością ważoną drogi $(u_0, u_1, \ldots, u_n)$ w grafie $G(V, E)$ o nieujemnych wagach krawędzi (łuków) $e \in E$ jest suma wag krawędzi tej drogi.

Do znajdowania najkrótszej drogi ważonej z ustalonego wierzchołka u w grafie $G(V, E)$ o nieujemnych wagach krawędzi $e \in E$ służy algorytm opracowany przez E. Dijkstrę.

Długość ważona drogi

Długością ważoną drogi $(u_0, u_1, \ldots, u_n)$ w grafie $G(V, E)$ o nieujemnych wagach krawędzi (łuków) $e \in E$ jest suma wag krawędzi tej drogi.

Do znajdowania najkrótszej drogi ważonej z ustalonego wierzchołka u w grafie $G(V, E)$ o nieujemnych wagach krawędzi $e \in E$ służy algorytm opracowany przez E. Dijkstrę.

Algorytm ten znajduje w grafie wszystkie najkrótsze drogi pomiędzy wyróżnionym wierzchołkiem u a wszystkimi pozostałymi, dodatkowo wyliczając długość każdej z tych dróg. Algorytm Dijkstry jest również przykładem algorytmu zachłannego.

Sformułujemy go dla grafów nieskierowanych.

Algorytm Dijkstry

Tworzymy dwa zbiory wierzchołków Q i S. W trakcie algorytmu będziemy tworzyć funkcję $d(v) \geq 0$ określoną na V i funkcję $p(v)$ określoną na $V \setminus u$ o wartościach w V. Wierzchołek u jest jedynym wierzchołkiem początkowym.

Algorytm Dijkstry

Tworzymy dwa zbiory wierzchołków Q i S. W trakcie algorytmu będziemy tworzyć funkcję $d(v) \geq 0$ określoną na V i funkcję $p(v)$ określoną na $V \setminus u$ o wartościach w V. Wierzchołek u jest jedynym wierzchołkiem początkowym.

1. Niech $Q = V$, $S = \emptyset$.

Algorytm Dijkstry

Tworzymy dwa zbiory wierzchołków Q i S. W trakcie algorytmu będziemy tworzyć funkcję $d(v) \geq 0$ określoną na V i funkcję $p(v)$ określoną na $V \setminus u$ o wartościach w V. Wierzchołek u jest jedynym wierzchołkiem początkowym.

1. Niech $Q = V$, $S = \emptyset$.
2. Dla każdego $v \in V \setminus u$ przyjmujemy $d(v) = \infty$ oraz $d(u) = 0$.

Algorytm Dijkstry

Tworzymy dwa zbiory wierzchołków Q i S. W trakcie algorytmu będziemy tworzyć funkcję $d(v) \geq 0$ określoną na V i funkcję $p(v)$ określoną na $V \setminus u$ o wartościach w V. Wierzchołek u jest jedynym wierzchołkiem początkowym.

1. Niech $Q = V$, $S = \emptyset$.
2. Dla każdego $v \in V \setminus u$ przyjmujemy $d(v) = \infty$ oraz $d(u) = 0$.
3. Funkcja $p(v)$ jest niezdefiniowana dla wszystkich $v \in V$.

Algorytm Dijkstry

Tworzymy dwa zbiory wierzchołków Q i S. W trakcie algorytmu będziemy tworzyć funkcję $d(v) \geq 0$ określoną na V i funkcję $p(v)$ określoną na $V \setminus u$ o wartościach w V. Wierzchołek u jest jedynym wierzchołkiem początkowym.

1. Niech $Q = V$, $S = \emptyset$.
2. Dla każdego $v \in V \setminus u$ przyjmujemy $d(v) = \infty$ oraz $d(u) = 0$.
3. Funkcja $p(v)$ jest niezdefiniowana dla wszystkich $v \in V$.
4. Powtarzamy poniższe kroki algorytmu, aż $Q = \emptyset$.

Algorytm Dijkstry

Tworzymy dwa zbiory wierzchołków Q i S. W trakcie algorytmu będziemy tworzyć funkcję $d(v) \geq 0$ określoną na V i funkcję $p(v)$ określoną na $V \setminus u$ o wartościach w V. Wierzchołek u jest jedynym wierzchołkiem początkowym.

1. Niech $Q = V$, $S = \emptyset$.
2. Dla każdego $v \in V \setminus u$ przyjmujemy $d(v) = \infty$ oraz $d(u) = 0$.
3. Funkcja $p(v)$ jest niezdefiniowana dla wszystkich $v \in V$.
4. Powtarzamy poniższe kroki algorytmu, aż $Q = \emptyset$.
    1. Wybieramy w Q wierzchołek o najmniejszym $d(v)$, usuwamy go z Q i dodajemy do S.

Algorytm Dijkstry

Tworzymy dwa zbiory wierzchołków Q i S. W trakcie algorytmu będziemy tworzyć funkcję $d(v) \geq 0$ określoną na V i funkcję $p(v)$ określoną na $V \setminus u$ o wartościach w V. Wierzchołek u jest jedynym wierzchołkiem początkowym.

1. Niech $Q = V$, $S = \emptyset$.
2. Dla każdego $v \in V \setminus u$ przyjmujemy $d(v) = \infty$ oraz $d(u) = 0$.
3. Funkcja $p(v)$ jest niezdefiniowana dla wszystkich $v \in V$.
4. Powtarzamy poniższe kroki algorytmu, aż $Q = \emptyset$.
    1. Wybieramy w Q wierzchołek o najmniejszym $d(v)$, usuwamy go z Q i dodajemy do S.
    2. Dla każdego $t \in \Gamma(v) \cap Q$, jeśli $d(t) > d(v) + w(v, t)$, to $d(t) = d(v) + w(v, t)$.

Algorytm Dijkstry

Tworzymy dwa zbiory wierzchołków Q i S. W trakcie algorytmu będziemy tworzyć funkcję $d(v) \geq 0$ określoną na V i funkcję $p(v)$ określoną na $V \setminus u$ o wartościach w V. Wierzchołek u jest jedynym wierzchołkiem początkowym.

1. Niech $Q = V$, $S = \emptyset$.
2. Dla każdego $v \in V \setminus u$ przyjmujemy $d(v) = \infty$ oraz $d(u) = 0$.
3. Funkcja $p(v)$ jest niezdefiniowana dla wszystkich $v \in V$.
4. Powtarzamy poniższe kroki algorytmu, aż $Q = \emptyset$.
    1. Wybieramy w Q wierzchołek o najmniejszym $d(v)$, usuwamy go z Q i dodajemy do S.
    2. Dla każdego $t \in \Gamma(v) \cap Q$, jeśli $d(t) > d(v) + w(v, t)$, to $d(t) = d(v) + w(v, t)$.
    3. Określamy $p(t) = v$.

Uwagi

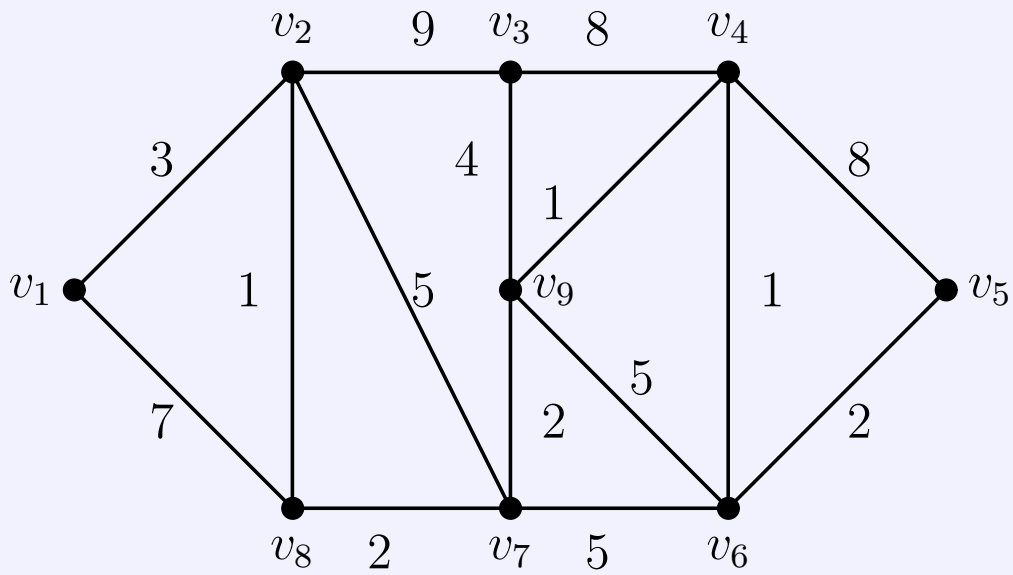
W wyniku działania tego algorytmu ciąg wierzchołków $v, v_1 = p(v), v_2 = p(v_1), \ldots, u = p(v_m)$ jest najkrótszą drogą z wierzchołka v do początkowego wierzchołka u.

Uwagi

W wyniku działania tego algorytmu ciąg wierzchołków $v, v_1 = p(v), v_2 = p(v_1), \ldots, u = p(v_m)$ jest najkrótszą drogą z wierzchołka v do początkowego wierzchołka u.

Dla wybrania z Q wierzchołka v o najmniejszym $d(v)$ najlepiej przyjąć, że Q jest uporządkowany tak jak przy przeszukiwaniu grafu metodą BFS.

Przykład

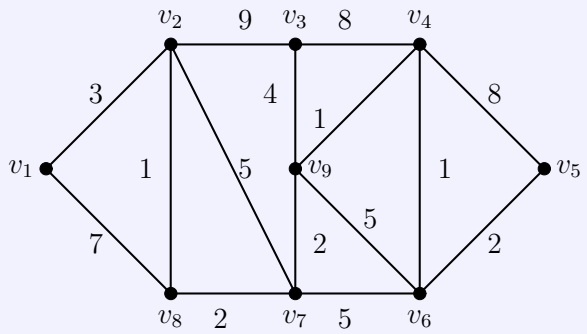


Najkrótszą drogą ważoną od v_1 do v_5 jest ciąg $(v_1, v_2, v_8, v_7, v_9, v_4, v_6, v_5)$. Ważona długość tej drogi jest równa 12, a długość nieważona jest równa 7.
Długość zaś najkrótszej drogi nieważonej jest równa 4, a taką drogą jest na przykład ciąg v_1, v_2, v_3, v_4, v_5. Długość ważona tej drogi wynosi 28.

Przykład (1)

Aby zastosować algorytm Dijkstry, ustalamy najpierw algorytmem BFS zbiór (kolejkę) Q (wiersz pierwszy) i nadajemy wartości z kroku 2 w algorytmie (wiersz drugi).
W kolejnych krokach algorytmu, idąc z kolejnych wierzchołków v z kolejki Q, wierzchołki $t \in V^+(v)$ grafu otrzymują wartości $d(t)$.

Przykład (2)



	v_1	v_2	v_8	v_3	v_7	v_4	v_9	v_6	v_5
	0	∞	∞	∞	∞	∞	∞	∞	∞
v_1	0	3	7	∞	∞	∞	∞	∞	∞
v_2	0	3	4	12	8	∞	∞	∞	∞
v_8	0	3	4	12	6	∞	∞	∞	∞
v_3	0	3	4	12	6	20	16	∞	∞
v_7	0	3	4	12	6	20	8	11	∞
v_9	0	3	4	12	6	9	8	11	∞
v_4	0	3	4	12	6	9	8	10	17
v_6	0	3	4	12	6	7	6	10	12
		v_1	v_2	v_2	v_8	v_9	v_7	v_4	v_6