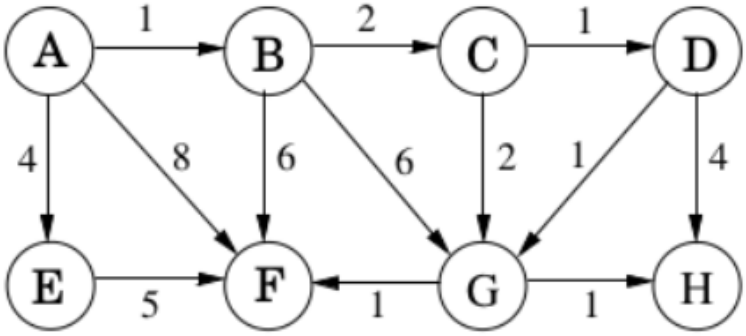
Lista 8

Zadanie 1

W zadaniu należy wykonać algorytm Dijkstry na poniższym grafie zaczynając od węzła A.



1. Zasada działania algorytmu Dijkstry

Zasadę działania pokażę na podstawie poniższego pseudokodu.

```
1  Dijkstra(G, s):
2      dla każdego wierzchołka v w V[G] wykonaj:
3          v.dist := nieskończoność
4          v.prev := null
5      s.dist := 0
6      Q := MakeQueue(V)
7      dopóki Q niepuste wykonuj:
8          u := pop_min(Q)
9          dla każdego wierzchołka v - sąsiada u wykonaj:
10             if v.dist > u.dist + l(u, v):
11                 v.dist := u.dist + l(u, v)
12                 v.prev := u
13                 DecreaseKey(Q, v)
```

Linie 1 – 5:

- Na początku dla każdego wierzchołka grafu $v \in V(G)$, jego odległość od źródła (od wierzchołka s, który podajemy jako argument) ustawiana jest na nieskończoność. Za wyjątkiem źródła, gdyż odległość od źródła do niego samego wynosi 0 (patrz 5 linijka).

Linia 6:

- Następnie ze zbioru będącego zbiorem wierzchołków grafu tworzona jest kolejka priorytetowa Q, przy czym jako priorytet będzie traktowana odległość od źródła (tj. v.dist). Jest to kolejka oparta na kopcu minimalnym – najniższa wartość klucza =

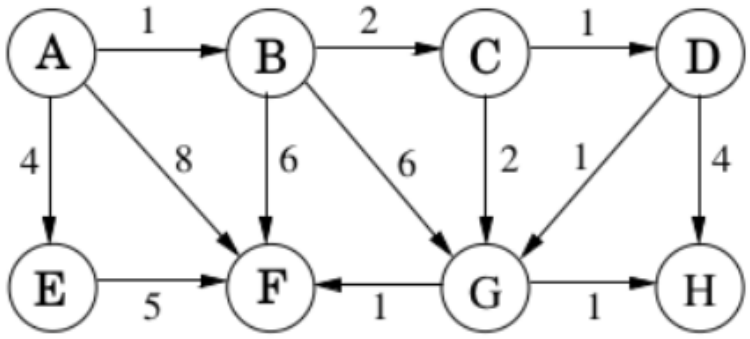
najwyższy priorytet. Czyli wierzchołek o najmniejszej odległości od źródła ma największy priorytet.

Linie 7 – 13:

- Teraz dopóki kolejka priorytetowa wierzchołków jest niepusta pobieramy (i jednocześnie usuwamy) z niej element o największym priorytecie, czyli element o najmniejszej odległości od źródła (stąd pop_min(Q) – patrz linia 8). Po pobraniu wierzchołka o najmniejszej odległości od źródła obserwujemy wszystkich jego sąsiadów. Jeśli aktualna odległość dla danego sąsiada jest większa niż ta, którą możemy uzyskać docierając do niego bezpośrednio z wierzchołka, który przed chwilą zdjęliśmy z kolejki (w pseudokodzie wierzchołek u), to odległość od źródła (v.dist) dla tego sąsiada otrzymuje nową wartość (równą sumie odległości aktualnego wierzchołka od źródła i kosztu przejścia z aktualnego wierzchołka na tego sąsiada – patrz linia 11), zaś jako poprzednika tego sąsiada ustawiamy aktualny wierzchołek (ten, który został przed chwilą zdjęty z kolejki). Na samym końcu pozostaje nam zaktualizowanie priorytetu tego sąsiada, gdyż kluczami w kolejce Q są odległości od źródła, więc skoro ta wartość się zmieniła, to może zajść konieczność zmiany kolejności wierzchołków w kolejce priorytetowej (patrz linia 13).

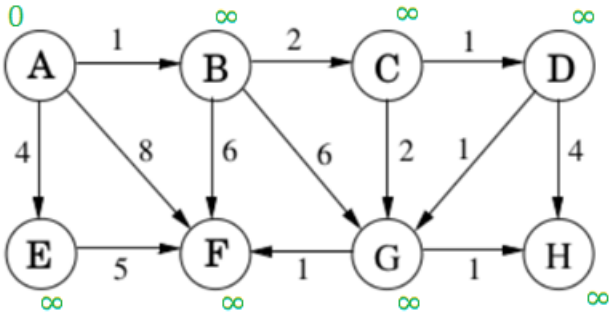
Warto zauważyć, że powyższy algorytm gwarantuje, że jeżeli do pewnego wierzchołka nie da się dojść, to na końcu jego działania odległość dla tego wierzchołka będzie równa nieskończoność.

Teraz wykonamy algorytm Dijkstry na poniższym grafie.

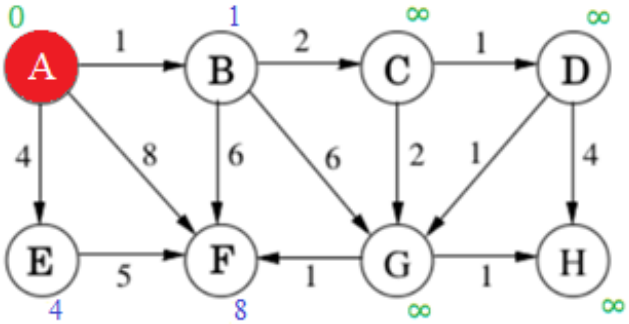


Przyjmiemy następujące oznaczenia:

- Aktualna wartość pola *dist* (odległość od źródła) dla każdego z wierzchołków będzie reprezentowana za pomocą zielonej liczby obok tego wierzchołka
- Odległości, które uległy zmianie w danej iteracji będą reprezentowane za pomocą niebieskiej liczby obok wierzchołka
- Wierzchołek, który został już zdjęty z kolejki priorytetowej będzie koloru czerwonego (wierzchołki znajdujące się w kolejce nie będą miały koloru)
- Kolejkę priorytetową, na której umieszczone będą wierzchołki nazywać będziemy Q, zaś wierzchołek od którego zaczynamy, czyli A będziemy nazywali źródłem

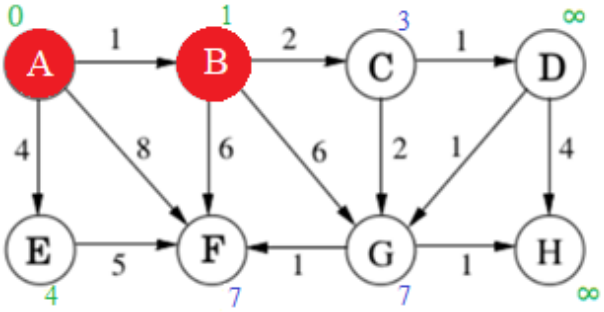


Zgodnie z algorytmem, na samym początku odległość od źródła dla wszystkich wierzchołków ustawiana jest na nieskończoność (za wyjątkiem wierzchołka od którego zaczynamy – źródła).



Przed pierwszą iteracją z Q zostaje pobrany wierzchołek A – dlatego jest zaznaczony kolorem czerwonym (on ma najmniejszą odległość od wierzchołka startowego = 0). Jego sąsiadami są wierzchołki B, E oraz F. Dla każdego z nich aktualna odległość wynosiła nieskończoność, była więc większa od tej, którą możemy uzyskać dochodząc do nich bezpośrednio z wierzchołka A. Ustawiamy więc sąsiadom A odpowiednią wartość *dist* (są to liczby oznaczone kolorem niebieskim), oraz jako poprzednika tych wierzchołków ustawiamy A.

Zauważmy, że wartość *dist* jest teraz najmniejsza dla wierzchołka B, więc w następnej iteracji to on zostanie pobrany z Q.

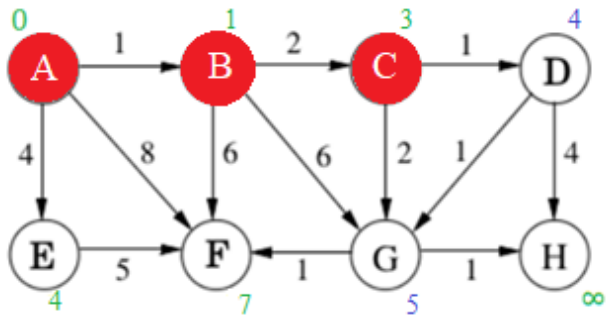


Przechodząc po sąsiadach wierzchołka B możemy zauważyć, że odległość dla C i G wynosiła nieskończoność:

inf > 1 + 2, więc C.dist = 3 i C.prev = B

inf > 1 + 6, więc G.dist = 7 i G.prev = B

Zauważmy, że poprzednio F.dist = 8. Podróżując do wierzchołka F przez wierzchołek B koszt podróży do tego wierzchołka jest mniejszy, niż gdy idziemy do niego bezpośrednio z wierzchołka A (8 > 1 + 6). Zatem F.dist = 7 i F.prev = B.

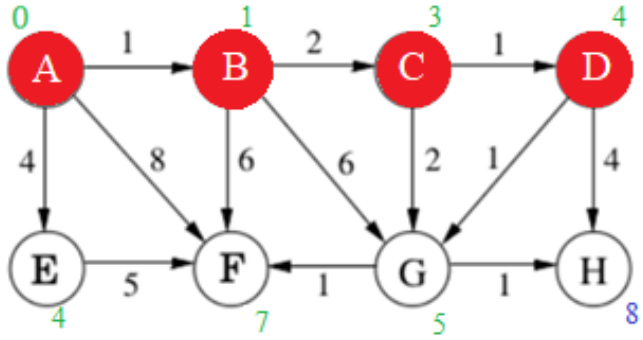


Teraz najmniejsza odległość od A występuje dla wierzchołka C, więc w następnej iteracji to on zostanie pobrany z Q. Z wierzchołka C możemy dotrzeć jedynie do wierzchołków D oraz G:

inf > 3 + 1, więc D.dist = 4 i D.prev = C

7 > 3 + 2, więc G.dist = 5 i G.prev = C

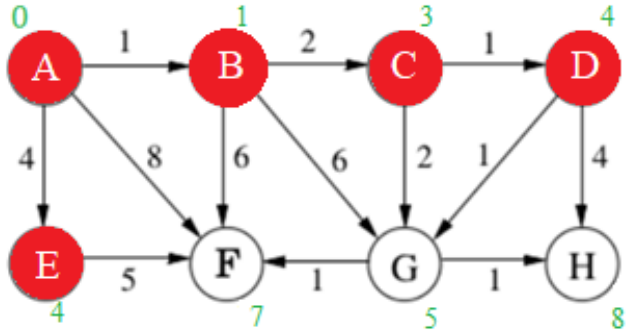
Zauważmy, że wartość *dist* jest teraz najmniejsza dla D.dist = E.dist = 4. Nie ma znaczenia, który z tych wierzchołków zostanie pobrany pierwszy z Q. Przyjmijmy, że weźmiemy wierzchołek D.



Z wierzchołka D możemy dotrzeć do G oraz H. Zauważmy, że dla wierzchołka G otrzymujemy taką samą wartość dist podróżując przez wierzchołek D, jak gdy docieraliśmy do niego bezpośrednio z C, więc G.dist pozostaje niezmienione. Dla wierzchołka H zaś:

inf > 4 + 4, więc H.dist = 8 i H.prev = D.

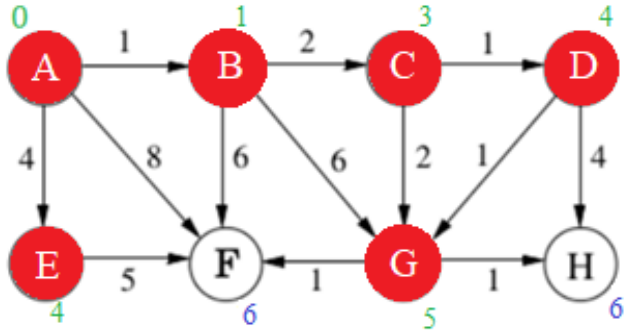
Teraz wierzchołkiem, który zostanie pobrany z kolejki priorytetowej będzie wierzchołek E, gdyż ma największy priorytet (najmniejszą odległość od wierzchołka A).



Z wierzchołka E możemy dojść jedynie do wierzchołka F. Ale ścieżka prowadząca od A do F, która przechodzi przez wierzchołek E nie będzie krótsza niż ta, która prowadzi do F poprzez wierzchołek B, gdyż:

4 + 5 > 7, więc wszystkie odległości pozostają niezmienione.

Następny wierzchołek, jaki zostanie zdjęty z kolejki priorytetowej, to G.



Z wierzchołka G możemy dotrzeć do wierzchołków H oraz F. Zauważmy, że zarówno do H jak i F korzystniej będzie dotrzeć wcześniej odwiedzając G, gdyż:

7 > 5 + 1, więc F.dist = 6 i F.prev = G

8 > 5 + 1, więc H.dist = 6 i H.prev = G

W tym momencie zostaną pobrane z kolejki priorytetowej wierzchołek F, a następnie H (lub w odwrotnej kolejności – nie ma znaczenia, gdyż mają ten sam priorytet). Oba te wierzchołki są incydentne tylko z krawędziami skierowanymi w stronę tych wierzchołków (nie ma sąsiadów, do których moglibyśmy się udać bezpośrednio z tych wierzchołków). Nie zajdzie więc już żadna zmiana w wartościach dist wierzchołków tego grafu, zaś algorytm skończy działanie – kolejka priorytetowa Q jest pusta.

Warto również zauważyć, że dla każdego wierzchołka v w polach v.prev zapisywany był wierzchołek, z którego bezpośrednio docieramy do wierzchołka v podróżując od źródła do wierzchołka v. Pozwala nam to na odtworzenie najkrótszej ścieżki do dowolnego wierzchołka, przy czym jeśli do danego wierzchołka nie ma ścieżki (nie da się do niego dotrzeć), to wartość prev pozostanie *null'em*.

W poniższej tabeli przedstawiam dystans do każdego węzła grafu po każdej iteracji algorytmu.

Uwaga: 5_X – oznacza, że po danej iteracji odległość od źródła dla danego wierzchołka wynosiła 5, zaś X oznacza wierzchołek, który był bezpośrednio poprzedzającym

Uwaga: na niebiesko zaznaczam te wartości dist, której już na pewno się nie zmienią, ponieważ wierzchołki te zostały już zdjęte z kolejki priorytetowej.

Iteracja	Wierzchołek usunięty z Q w tej iteracji	A.dist	B.dist	C.dist	D.dist	E.dist	F.dist	G.dist	H.dist
Po pierwszej iteracji	A	0_A	1_A	∞	∞	4_A	8_A	∞	∞
Po drugiej iteracji	B	0_A	1_A	3_B	∞	4_A	7_B	7_B	∞
Po trzeciej iteracji	C	0_A	1_A	3_B	4_C	4_A	7_B	5_C	∞
Po czwartej iteracji	D	0_A	1_A	3_B	4_C	4_A	7_B	5_C	8_D
Po piątej iteracji	E	0_A	1_A	3_B	4_C	4_A	7_B	5_C	8_D
Po szóstej iteracji	G	0_A	1_A	3_B	4_C	4_A	6_G	5_C	6_G
Po siódmej iteracji	F	0_A	1_A	3_B	4_C	4_A	6_G	5_C	6_G
Po ósmej iteracji	H	0_A	1_A	3_B	4_C	4_A	6_G	5_C	6_G

Poniżej przedstawiam drzewo najkrótszych ścieżek powstałe po wykonaniu algorytmu.

Poniższy graf jest drzewem, gdyż jest grafem spójnym oraz acyklicznym (zawiera on najkrótsze ścieżki wyznaczone przez algorytm Dijkstry).