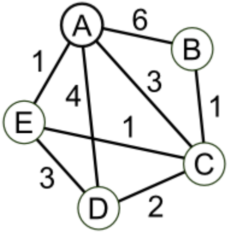
Znajdywanie najkrótszej drogi w grafie - metoda Dijkstry

Igor Nowicki

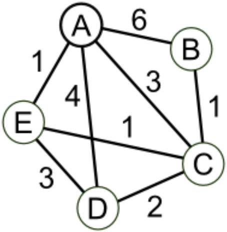
WIT - Wyższa Szkoła Informatyki Stosowanej i Zarządzania

31 stycznia 2020

Dany jest graf ważony krawędziowo o nieujemnych wagach.

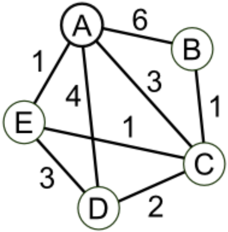


Dany jest graf ważony krawędziowo o nieujemnych wagach.



Jaki jest minimalny koszt przejścia pomiędzy wybranymi dwoma węzłami grafu?

Dany jest graf ważony krawędziowo o nieujemnych wagach.



Jaki jest minimalny koszt przejścia pomiędzy wybranymi dwoma węzłami grafu? Czy możemy znaleźć stowarzyszoną z tym kosztem optymalną trasę?

Metoda Dijkstry wykorzystuje *zachłanne* podejście do problemu - na każdym kroku ustalamy koszty alternatywnych ścieżek i wybieramy tę o najniższej wartości.

Metoda Dijkstry wykorzystuje *zachłanne* podejście do problemu - na każdym kroku ustalamy koszty alternatywnych ścieżek i wybieramy tę o najniższej wartości.

Algorytm możemy opisać następująco:

Metoda Dijkstry wykorzystuje *zachłanne* podejście do problemu - na każdym kroku ustalamy koszty alternatywnych ścieżek i wybieramy tę o najniższej wartości.

Algorytm możemy opisać następująco:

1. Wybierz nieodwiedzony jeszcze węzeł o najniższej przypisanej wartości.

Metoda Dijkstry wykorzystuje *zachłanne* podejście do problemu - na każdym kroku ustalamy koszty alternatywnych ścieżek i wybieramy tę o najniższej wartości.

Algorytm możemy opisać następująco:

1. Wybierz nieodwiedzony jeszcze węzeł o najniższej przypisanej wartości.
2. Zaktualizuj wartości sąsiadów - jeśli ich przypisana wartość jest wyższa, zastąp ją sumą bieżącego węzła i drogi. Zapamiętaj skąd wyszło połączenie.

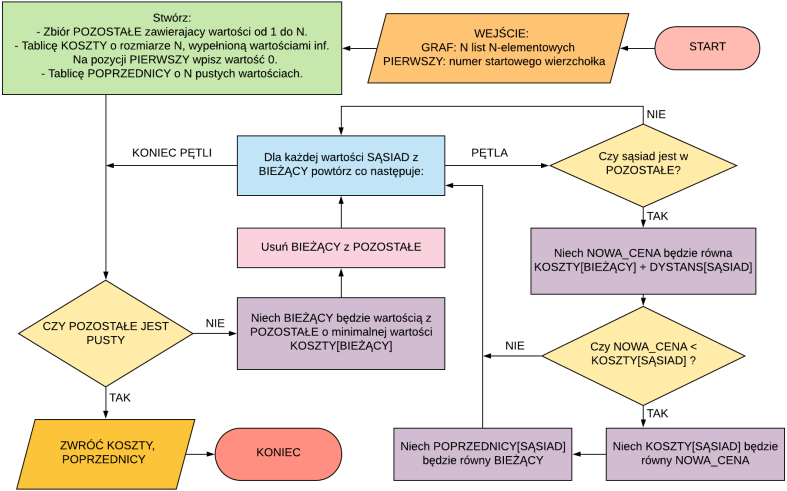
Metoda Dijkstry wykorzystuje *zachłanne* podejście do problemu - na każdym kroku ustalamy koszty alternatywnych ścieżek i wybieramy tę o najniższej wartości.

Algorytm możemy opisać następująco:

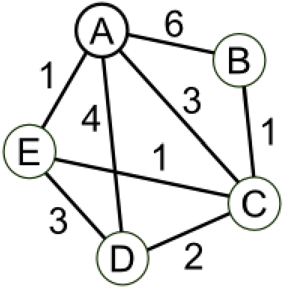
1. Wybierz nieodwiedzony jeszcze węzeł o najniższej przypisanej wartości.
2. Zaktualizuj wartości sąsiadów - jeśli ich przypisana wartość jest wyższa, zastąp ją sumą bieżącego węzła i drogi. Zapamiętaj skąd wyszło połączenie.
3. Powtarzaj kroki 1 i 2 dopóki nie zostaną odwiedzone wszystkie węzły.

Schemat blokowy:

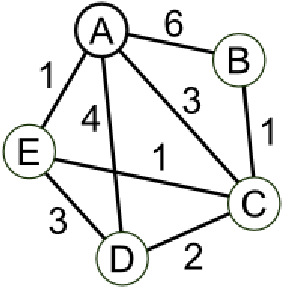
Schemat blokowy:



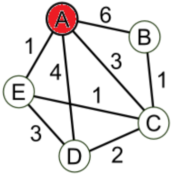
Weźmiemy graf z początku prezentacji:



Weźmiemy graf z początku prezentacji:

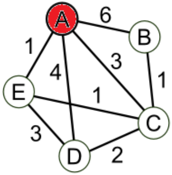


Będziemy poszukiwać optymalnych tras z punktu A. Jako test bojowy sprawdzimy, czy algorytm poprawnie znajduje najniższy koszt przejścia z A do B.



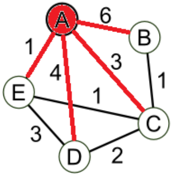
	A	B	C	D	E
POZOSTAŁE	+	+	+	+	+
KOSZTY	∞	∞	∞	∞	∞
POPRZEDNICY	-	-	-	-	-

Inicjalizacja. Wszystkie węzły traktujemy jako nieodwiedzone. Bieżący koszt dotarcia do każdego z węzłów jest nieskończony, poza węzłem A, z którego zaczynamy.



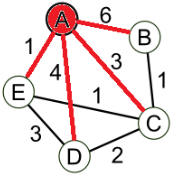
	A	B	C	D	E
POZOSTAŁE	+	+	+	+	+
KOSZTY	0	∞	∞	∞	∞
POPRZEDNICY	-	-	-	-	-

Inicjalizacja. Wszystkie węzły traktujemy jako nieodwiedzone. Bieżący koszt dotarcia do każdego z węzłów jest nieskończony, poza węzłem A, z którego zaczynamy.



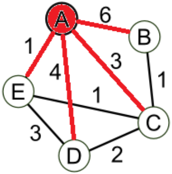
	A	B	C	D	E
POZOSTAŁE	+	+	+	+	+
KOSZTY	0	∞	∞	∞	∞
POPRZEDNICY	-	-	-	-	-

Pierwszy krok. Kasujemy A ze zbioru POZOSTAŁE. Uzupełniamy koszty B, C, D oraz E, ustawiamy ich poprzednika na A. Ustalamy E jako bieżący węzeł o najniższej wartości z jeszcze nieodwiedzonych.



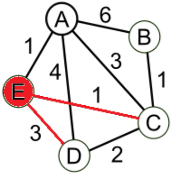
	A	B	C	D	E
POZOSTAŁE	-	+	+	+	+
KOSZTY	0	∞	∞	∞	∞
POPRZEDNICY	-	-	-	-	-

Pierwszy krok. Kasujemy A ze zbioru POZOSTAŁE. Uzupełniamy koszty B, C, D oraz E, ustawiamy ich poprzednika na A. Ustalamy E jako bieżący węzeł o najniższej wartości z jeszcze nieodwiedzonych.



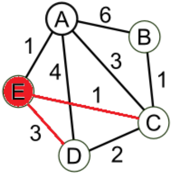
	A	B	C	D	E
POZOSTAŁE	-	+	+	+	+
KOSZTY	0	6	3	4	1
POPRZEDNICY	-	A	A	A	A

Pierwszy krok. Kasujemy A ze zbioru POZOSTAŁE. Uzupełniamy koszty B, C, D oraz E, ustawiamy ich poprzednika na A. Ustalamy E jako bieżący węzeł o najniższej wartości z jeszcze nieodwiedzonych.



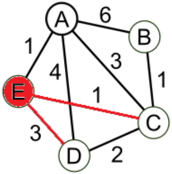
	A	B	C	D	E
POZOSTAŁE	-	+	+	+	+
KOSZTY	0	6	3	4	1
POPRZEDNICY	-	A	A	A	A

Drugi krok. Kasujemy E z listy pozostałych, aktualizujemy wartości połączeń z C oraz D. C jest kolejną wartością.



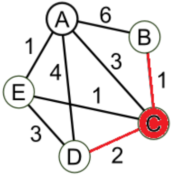
	A	B	C	D	E
POZOSTAŁE	-	+	+	+	-
KOSZTY	0	6	3	4	1
POPRZEDNICY	-	A	A	A	A

Drugi krok. Kasujemy E z listy pozostałych, aktualizujemy wartości połączeń z C oraz D. C jest kolejną wartością.



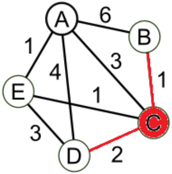
	A	B	C	D	E
POZOSTAŁE	-	+	+	+	-
KOSZTY	0	6	2	4	1
POPRZEDNICY	-	A	E	A	A

Drugi krok. Kasujemy E z listy pozostałych, aktualizujemy wartości połączeń z C oraz D. C jest kolejną wartością.



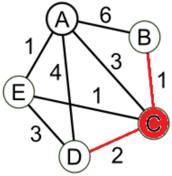
	A	B	C	D	E
POZOSTAŁE	-	+	+	+	-
KOSZTY	0	6	2	4	1
POPRZEDNICY	-	A	E	A	A

Trzeci krok. Kasujemy C z pozostałych, aktualizujemy koszty dróg do B oraz D. Następcą będzie B.



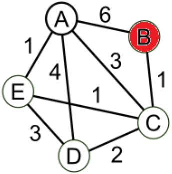
	A	B	C	D	E
POZOSTAŁE	-	+	-	+	-
KOSZTY	0	6	2	4	1
POPRZEDNICY	-	A	E	A	A

Trzeci krok. Kasujemy C z pozostałych, aktualizujemy koszty dróg do B oraz D. Następcą będzie B.



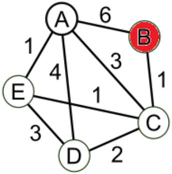
	A	B	C	D	E
POZOSTAŁE	-	+	-	+	-
KOSZTY	0	3	2	4	1
POPRZEDNICY	-	C	E	A	A

Trzeci krok. Kasujemy C z pozostałych, aktualizujemy koszty dróg do B oraz D. Następcą będzie B.



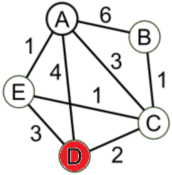
	A	B	C	D	E
POZOSTAŁE	-	+	-	+	-
KOSZTY	0	3	2	4	1
POPRZEDNICY	-	C	E	A	A

Czwarty krok. Usuwamy B z listy pozostałych. Nie aktualizujemy żadnych dróg. Następnym węzłem będzie D.



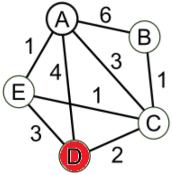
	A	B	C	D	E
POZOSTAŁE	-	-	-	+	-
KOSZTY	0	3	2	4	1
POPRZEDNICY	-	C	E	A	A

Czwarty krok. Usuwamy B z listy pozostałych. Nie aktualizujemy żadnych dróg. Następnym węzłem będzie D.



	A	B	C	D	E
POZOSTAŁE	-	-	-	+	-
KOSZTY	0	3	2	4	1
POPRZEDNICY	-	C	E	A	A

Piąty krok. Usuwamy D z listy pozostałych. Ponieważ wszystkie możliwe węzły zostały już uwzględnione, nie aktualizujemy żadnych ścieżek. Algorytm zwraca wyniki i kończy zadanie.



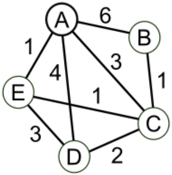
	A	B	C	D	E
POZOSTAŁE	-	-	-	-	-
KOSZTY	0	3	2	4	1
POPRZEDNICY	-	C	E	A	A

Piąty krok. Usuwamy D z listy pozostałych. Ponieważ wszystkie możliwe węzły zostały już uwzględnione, nie aktualizujemy żadnych ścieżek. Algorytm zwraca wyniki i kończy zadanie.

	A	B	C	D	E
KOSZTY	0	3	2	4	1
POPRZEDNICY	-	C	E	A	A

Dostaliśmy dwie tablice wyników: KOSZTY i POPRZEDNICY. Na podstawie tego możemy wyczytać, że najniższy koszt podróży z A do B wynosi 3 i następuje według trasy:

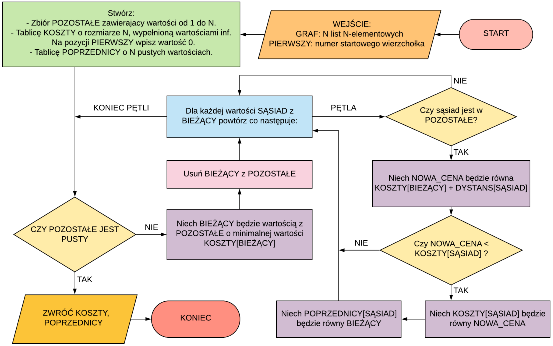
$$A \to E \to C \to B.$$



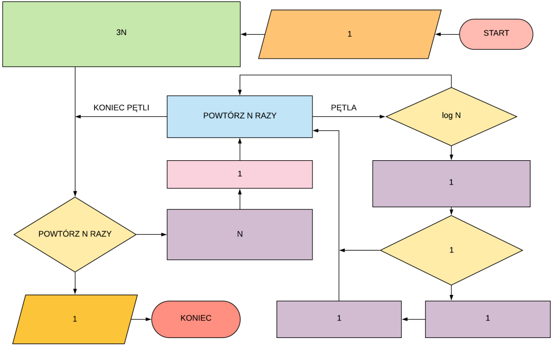
Zaimplementujemy teraz ten algorytm w Pythonie (wersja 3.8).

```python
from math import inf
import sys

def dijkstra(graph, start):
    n = len(graph)
    pozostale = set([i for i in range(n)])  #zbiór nieodwiedzonych węzłów
    poprzednicy = [None for i in range(n)] #lista poprzedzajacych węzłów
    koszty = [inf for i in range(n)] #lista kosztów dojścia do węzła
    koszty[start] = 0 #koszt dojścia do węzła startowego jest 0

    while len(pozostale) > 0:
        #znajdujemy wartość minimalną kosztów dla i spośród pozostałych
        #i zwracamy związany z nią indeks
        biezacy = min([(koszty[i], i) for i in pozostale])[1]
        #usuwamy bieżący indeks z listy nieodwiedzonych węzłów
        pozostale.remove(biezacy)
        #pobieramy listę dystansów z bieżącego węzła do sąsiadów
        dystans = graph[biezacy]
        for sasiad in range(n):
            #jeśli sąsiad został odwiedzony to zostawiamy go w spokoju
            #zerowy dystans do sąsiada oznacza brak połączenia
            if sasiad not in pozostale or dystans[sasiad] == 0:
                continue
            #nowa cena to bieżący koszt połączeń + koszt przejścia
            nowa_cena = koszty[biezacy] + dystans[sasiad]
            #jeśli nowa cena jest niższa, aktualizujemy koszty sąsiada
            if nowa_cena < koszty[sasiad]:
                poprzednicy[sasiad] = biezacy
                koszty[sasiad] = nowa_cena

    return koszty, poprzednicy
```

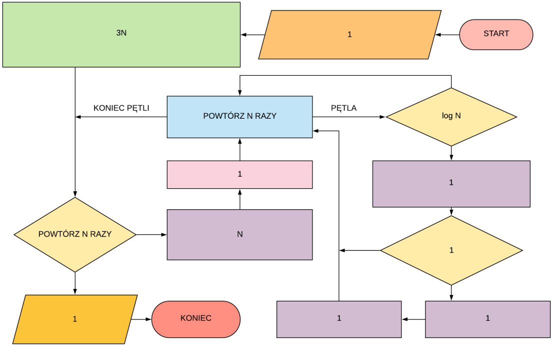
Oszacowanie złożoności czasowej:



Oszacowanie złożoności czasowej:

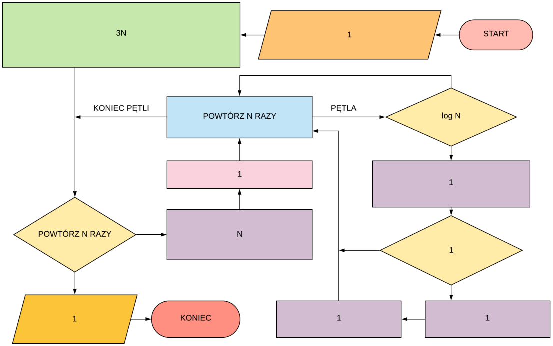


Oszacowanie złożoności czasowej:



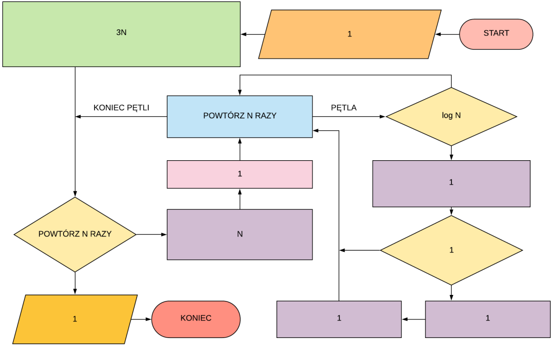
$$T(N) = 1 + 3N + N \cdot \Big(N + 1 + N \cdot \big(\log(N) + 1 + 1 + 1 + 1 \big) \Big) + 1$$

Oszacowanie złożoności czasowej:



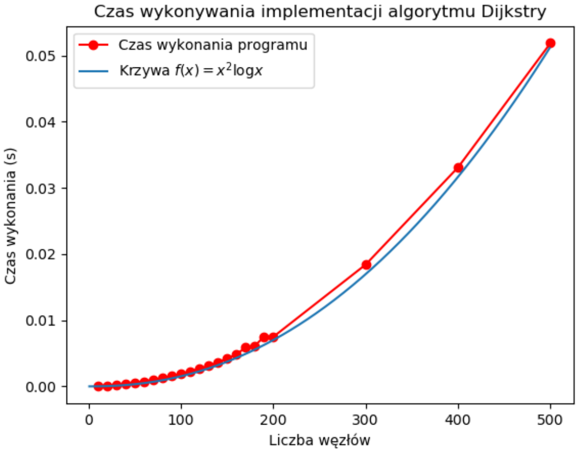
$$T(N) = 2 + 4N + 5N^2 + N^2 \log(N)$$

Oszacowanie złożoności czasowej:



$$T(N) \approx O\left(N^2 \log(N)\right)$$

Przedstawiona złożoność na wykresie:



Dziękuję za uwagę!