Wprowadzenie do teorii grafów w języku Python

Źródło: Anthony levlev, domena publiczna.

Grafy służą do modelowania wielu typów relacji i procesów w systemach fizycznych, biologicznych, społecznych i informacyjnych. Za pomocą wierzchołków oraz krawędzi można przedstawić wiele praktycznych sytuacji. Teoria grafów pozwala spojrzeć na problem z nieco innej perspektywy, opisać sytuację za pomocą połączeń lub powiązać relacje z pewnymi właściwościami. Więcej informacji o teorii grafów znajdziesz w e-materiale Wprowadzenie do teorii grafów.

W tym e-materiale przyjrzymy się zastosowaniu teorii grafów w języku Python.

Wyjaśnienie tego zagadnienia w kontekście pozostałych języków programowania znajdziesz w e-materiałach:

- Wprowadzenie do teorii grafów w języku C++,
- Wprowadzenie do teorii grafów w języku Java.

Twoje cele

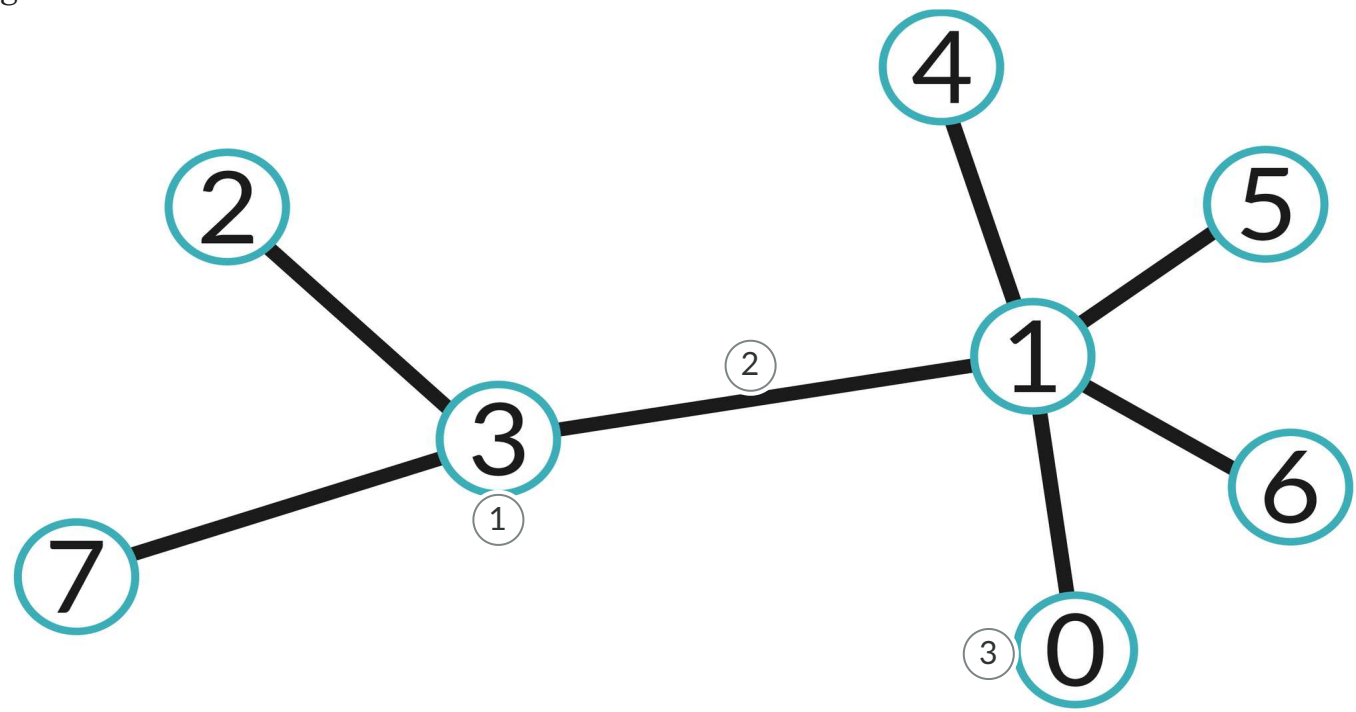
- Zaimplementujesz rozwiązania prostych problemów dotyczących teorii grafów w języku Python.
- Zapoznasz się z przykładową implementacją grafu w języku Python.
- Zaimplementujesz klasę grafu, zawierającą wszystkie potrzebne dane na temat wierzchołków i krawędzi.

- Rozwiążesz problemy związane z sąsiedztwem wierzchołków grafów.

Przeczytaj

Struktura grafu

Graf jest strukturą składającą się z obiektów (wierzchołków) oraz z powiązań między nimi (krawędzi). Zazwyczaj przedstawiany jest w formie diagramu jako zbiór punktów połączonych odcinkami, gdzie punkty przedstawiają wierzchołki, a odcinki – krawędzie grafu.



1

Wierzchołek oznaczony jako „3”

2

Krawędź łącząca wierzchołki „3” i „1”

3

Sąsiad wierzchołka „1”

Wierzchołki i krawędzie

Podczas implementacji potrzebna jest możliwość rozróżnienia wierzchołków. Osiągnąć to można, na przykład oznaczając poszczególne wierzchołki kolejnymi liczbami naturalnymi, zaczynając od 0. Przechowywać je będziemy za pomocą listy.

Warto zaznaczyć, że wierzchołki mogą być reprezentowane również za pomocą liter (taki przypadek zaprezentowano w **Ćwiczeniu 1** na dole tej sekcji).

```
wierzcholki = []
```

Podobnie w przypadku krawędzi:

```
krawedzie = []
```

Listy są parametrami klasy `Graf`. Klasa ta ma z kolei dwa pola – listę wierzchołków grafu oraz listę jego krawędzi. Domyślnymi parametrami konstruktora tej klasy będą puste listy:

```
class Graf:
    def __init__(self, wierzcholki=[], krawedzie=[]):
        self.wierzcholki = wierzcholki
        self.krawedzie = krawedzie
```

Wierzchołki grafu reprezentowane będą liczbami naturalnymi.

Utworzony w ten sposób obiekt `graf` klasy `Graf` ma trzy różne wierzchołki oznaczone numerami 0, 1 oraz 2, jednak nie ma żadnych krawędzi:

```
graf = Graf([0, 1, 2])
```

By wyświetlić wierzchołki obiektu `graf` klasy `Graf`, wykorzystamy polecenie print:

```
print(graf.wierzcholki)
```

Każda krawędź reprezentowana jest przez dwuelementową listę zawierającą numery wierzchołków, które łączy.

Dodajmy do utworzonego obiektu krawędzie łączące wierzchołki:

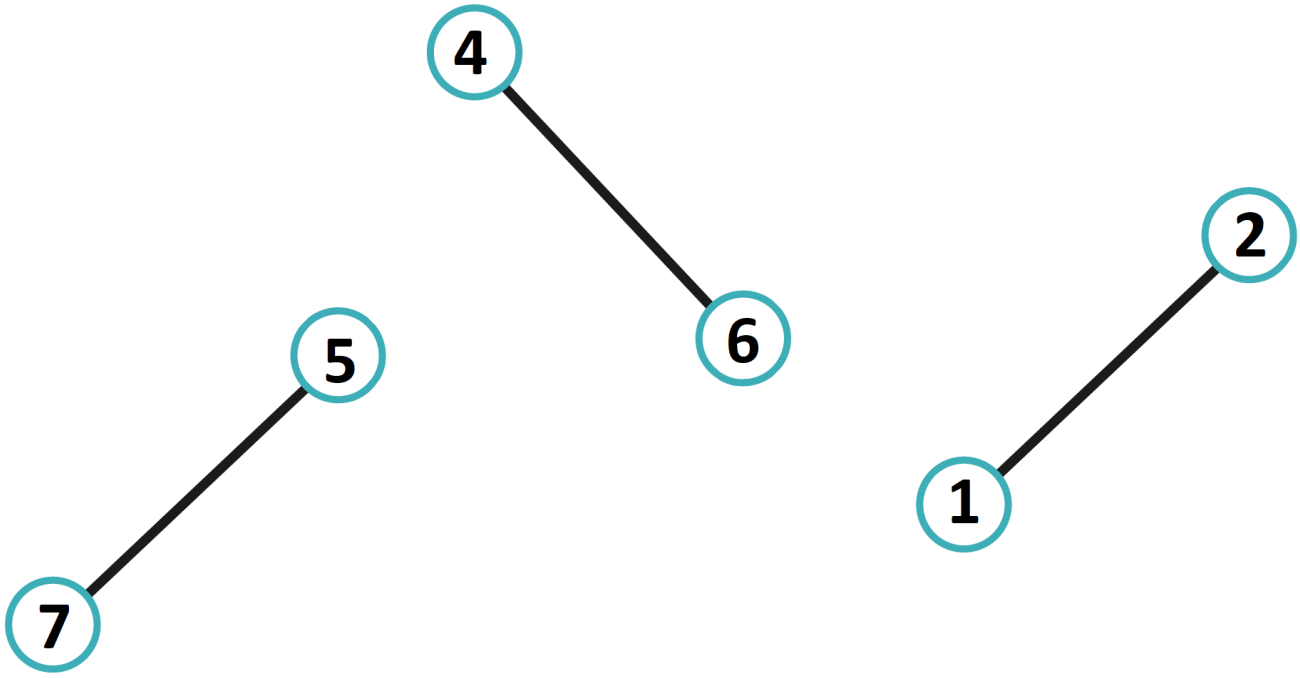
```

```

```
graf = Graf([0, 1, 2], [[0, 1], [0, 2]])
```

Graf opisany za pomocą poniższej klasy:

```
graf = Graf([7, 5, 4, 5, 1, 2], [[7, 5], [4, 6], [1, 2]])
```

wyglądałby następująco:



> **Polecenie 1**
>
> Jak nazywamy graf, którego nie wszystkie wierzchołki połączone są ze sobą krawędziami?

Przykład

W jaki sposób moglibyśmy za pomocą kodu zaprezentować graf zamieszczony na początku tej sekcji?

Zacznijmy od wierzchołków.

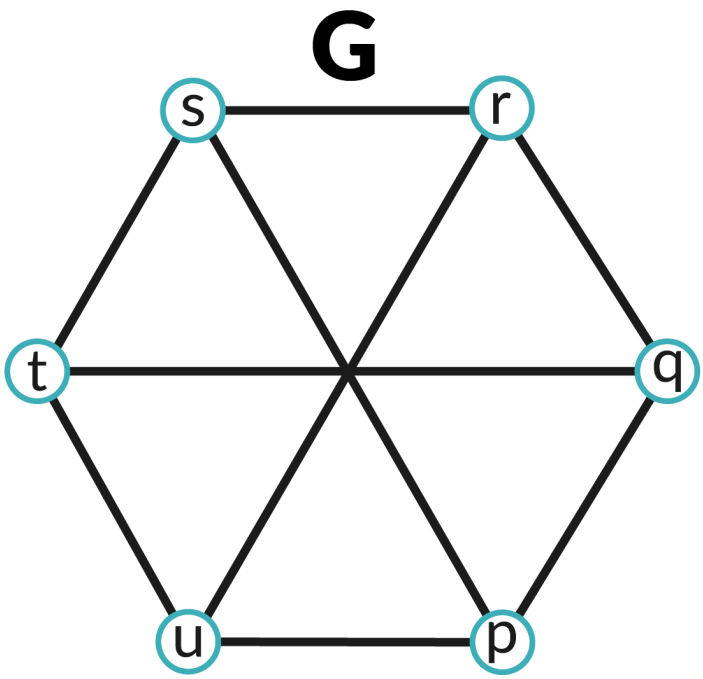
```
graf = Graf([7, 2, 3, 1, 4, 5, 6, 0])
```

Mamy do czynienia z grafem spójnym, więc jego wierzchołki połączone są krawędziami.

```
graf = Graf([7, 2, 3, 1, 4, 5, 6, 0], [[7, 3], [2, 3], [3, 1], [4
```

Ćwiczenie 1

Dany jest graf G.



Opisz go za pomocą klasy `Graf` zawierającej listę krawędzi oraz listę wierzchołków.

```
1
```

Słownik

droga

ciąg wierzchołków, w którym wszystkie wyrazy (z wyjątkiem pierwszego i ostatniego, które mogą być równe) są różne

graf

struktura składająca się z niepustego zbioru wierzchołków oraz ze zbioru krawędzi; dla grafu G zbiór wierzchołków oznaczamy $V(G)$ (ang. *verticies* – wierzchołki), a zbiór krawędzi – $E(G)$ (ang. *edges* – krawędzie); liczbę elementów zbioru $V(G)$ oznaczamy literą n, zaś liczbę elementów zbioru $E(G)$ – m

graf spójny

graf, w którym każda para wierzchołków połączona jest drogą

izomorfizm

właściwość grafów polegająca na tym, że liczba krawędzi łączących dane dwa wierzchołki jednego grafu jest równa liczbie krawędzi łączących odpowiadające im wierzchołki drugiego grafu; jeśli graf G jest izomorficzny z grafem H, to gdy jakieś dwa wierzchołki są połączone krawędzią w jednym z grafów, odpowiadające im wierzchołki w drugim grafie również łączy krawędź; izomorfizm grafów zachowuje takie własności jak: liczba wierzchołków, liczba krawędzi, stopnie wierzchołków, czy spójność grafu

klasa generyczna

klasa w języku Java pozwalająca stworzyć ogólną, uniwersalną klasę, która będzie wykonywała działania z różnymi typami danych, umożliwiając ponowne użycie kodu bez konieczności redefiniowania klasy dla każdego typu

krawędź

nieuporządkowana para (niekoniecznie różnych) elementów zbioru wierzchołków, tj. takich, które są ze sobą połączone; w reprezentacji graficznej jest to linia łącząca te wierzchołki; krawędź może łączyć ze sobą jeden wierzchołek (wtedy nazywana jest pętlą)

para nieuporządkowana

zbiór złożony jedynie z dwóch **różnych** elementów: $\{a, b\}$
stopień wierzchołka

liczba krawędzi incydentnych z danym wierzchołkiem
wierzchołek

inaczej: punkt, węzeł; element niepustego zbioru, który wraz ze zbiorem krawędzi tworzy graf

Prezentacja multimedialna

Polecenie 1

Zapoznaj się z prezentacją przedstawiającą przykład implementacji grafu w języku Python z wykorzystaniem klasy `Graf` zawierającej listę wierzchołków i listę krawędzi. Zastanów się, jakiego typu problemy programistyczne (ale nie tylko) można rozwiązywać za pomocą algorytmów grafowych.

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18BrngzT

Aby utworzyć reprezentację grafu w języku Python w postaci listy krawędzi, powinniśmy zacząć od implementacji klasy `Graf`.

```
1 class Graf:
2     def __init__(self, wierzcholki=[], krawedzie=[]):
3         self.wierzcholki = wierzcholki
4         self.krawedzie = krawedzie
5
```

Wiemy już, za pomocą jakich struktur przechowywana jest lista krawędzi oraz wierzchołków.

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18BrngzT

Możemy utworzyć przykładowy obiekt grafu zawierający trzy wierzchołki: 0, 1 i 2. Następnie wypiszemy w konsoli listę wierzchołków grafu.

```
class Graf:
    def __init__(self, wierzcholki=[], krawedzie=[]):
        self.wierzcholki = wierzcholki
        self.krawedzie = krawedzie

graf = Graf([0, 1, 2])
print(graf.wierzcholki) # [0, 1, 2]
```

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18BrngzT

Do klasy `Graf` dodamy funkcję `dodaj_wierzcholek()`, za pomocą której można będzie dodać nowy wierzchołek do grafu. Konieczne jest sprawdzenie, czy wierzchołek, który będziemy chcieli dodać, nie znajduje się już na liście `wierzcholki` – aby wierzchołki dało się rozróżnić, muszą być unikatowe.

4

```
class Graf:
    def __init__(self, wierzcholki=[], krawedzie=[]):
        self.wierzcholki = wierzcholki
```

```
        self.krawedzie = krawedzie

    def dodaj_wierzcholek(self, wierzcholek):
        if wierzcholek in self.wierzcholki:
            print("Nie można dodać wierzchołka - wierzchołek już dodano do grafu")
            return

        self.wierzcholki.append(wierzcholek)

graf = Graf([0, 1, 2])
print(graf.wierzcholki) # [0, 1, 2]
```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18BrngzT

5

Działanie funkcji `dodaj_wierzcholek()` możemy zaobserwować, próbując dodać wierzchołki o identyfikatorach 0 i 3 do istniejącego grafu. Po każdej operacji wypiszemy aktualną listę wierzchołków grafu, aby prezentować zmiany.

```
class Graf:
    def __init__(self, wierzcholki=[], krawedzie=[]):
        self.wierzcholki = wierzcholki
        self.krawedzie = krawedzie

```

```
    def dodaj_wierzcholek(self, wierzcholek):
        if wierzcholek in self.wierzcholki:
            print("Nie można dodać wierzchołka - wierzchołek już dodano do grafu")
            return

        self.wierzcholki.append(wierzcholek)

graf = Graf([0, 1, 2])
print(graf.wierzcholki) # [0, 1, 2]
graf.dodaj_wierzcholek(0) # Nie można dodać wierzchołka - wierzchołek już dodano do grafu
print(graf.wierzcholki)  # [0, 1, 2]
graf.dodaj_wierzcholek(3)
print(graf.wierzcholki)  # [0, 1, 2, 3]
```

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18BrngzT

Następnie do klasy `Graf` dodamy funkcję `dodaj_krawedz()`, za pomocą której można będzie dodać nową krawędź do grafu.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18BrngzT

7

Krawędzie muszą łączyć istniejące wierzchołki, więc podczas dodawania krawędzi konieczne jest sprawdzenie, czy wierzchołki podane w dwuelementowej liście znajdują się na liście `wierzcholki`.

```
class Graf:
    def __init__(self, wierzcholki=[], krawedzie=[]):
        self.wierzcholki = wierzcholki
        self.krawedzie = krawedzie

    def dodaj_wierzcholek(self, wierzcholek):
        if wierzcholek in self.wierzcholki:
            print("Nie można dodać wierzchołka - wierzchołek już dodano do grafu")
            return

        self.wierzcholki.append(wierzcholek)

    def dodaj_krawedz(self, krawedz):
        if krawedz[0] not in self.wierzcholki or krawedz[1] not in self.wierzcholki:
            print("Krawędź zawiera wierzchołek nie dodany do grafu")
            return
        # miejsce na dalszy ciąg funkcji


graf = Graf([0, 1, 2])
```

```
print(graf.wierzcholki) # [0, 1, 2]
graf.dodaj_wierzcholek(0)  # Nie można dodać wierzchołka - wierzchołek już dodano do grafu
print(graf.wierzcholki)  # [0, 1, 2]
graf.dodaj_wierzcholek(3)
print(graf.wierzcholki)  # [0, 1, 2, 3]
```

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18BrngzT

Przed dodaniem krawędzi należy sprawdzić (tak jak w przypadku wierzchołków), czy dana krawędź nie została już dodana do grafu.
W naszej implementacji graf nie będzie miał krawędzi skierowanych, co oznacza, że kolejność wierzchołków jednej krawędzi nie ma znaczenia – krawędzie `[0, 1]` i `[1, 0]` powinny być traktowane jako takie same. Aby to zrobić, możemy przekształcić porównywane krawędzie do postaci zbioru (ang. *set*) – struktury danych, w której elementy nie mają zachowanej kolejności – i porównać tak uzyskane struktury.

9

```
class Graf:
    def __init__(self, wierzcholki=[], krawedzie=[]):
        self.wierzcholki = wierzcholki
```

```
        self.krawedzie = krawedzie

    def dodaj_wierzcholek(self, wierzcholek):
        if wierzcholek in self.wierzcholki:
            print("Nie można dodać wierzchołka - wierzchołek już dodano do grafu")
            return

        self.wierzcholki.append(wierzcholek)

    def dodaj_krawedz(self, krawedz):
        if krawedz[0] not in self.wierzcholki or krawedz[1] not in self.wierzcholki:
            print("Krawędź zawiera wierzchołek niedodany do grafu")
            return
        for k in self.krawedzie:
            if set(krawedz) == set(k):
                print("Nie można dodać krawędzi - krawędź już dodano do grafu")
                return

        self.krawedzie.append(krawedz)


graf = Graf([0, 1, 2])
print(graf.wierzcholki) # [0, 1, 2]
```

```
graf.dodaj_wierzcholek(0)
 # Nie można dodać
wierzchołka - wierzchołek
już dodano do grafu
print(graf.wierzcholki)  #
[0, 1, 2]
graf.dodaj_wierzcholek(3)
print(graf.wierzcholki)  #
[0, 1, 2, 3]
```

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18BrngzT

Następnie przetestujemy dodawanie krawędzi – do konstruktora dodamy krawędzie łączące wierzchołki `[0, 1]` oraz `[0, 2]`. Działanie funkcji `dodaj_krawedz()` przetestujemy dla argumentów `[2, 3]` oraz `[0, 4]`.

```
class Graf:
    def __init__(self,
wierzcholki=[], krawedzie=
[]):
        self.wierzcholki =
wierzcholki
        self.krawedzie =
krawedzie

    def
dodaj_wierzcholek(self,
wierzcholek):
        if wierzcholek in
self.wierzcholki:
            print("Nie
można dodać wierzchołka -
wierzchołek już dodano do
grafu")
            return

 self.wierzcholki.append(w
```

```
ierzcholek)

    def dodaj_krawedz(self, krawedz):
        if krawedz[0] not in self.wierzcholki or krawedz[1] not in self.wierzcholki:
            print("Krawędź zawiera wierzchołek niedodany do grafu")
            return
        for k in self.krawedzie:
            if set(krawedz) == set(k):
                print("Nie można dodać krawędzi - krawędź już dodano do grafu")
                return

 self.krawedzie.append(krawedz)


graf = Graf([0, 1, 2], [[0, 1], [0, 2]])
print(graf.wierzcholki)  # [0, 1, 2]
graf.dodaj_wierzcholek(0)  # Nie można dodać wierzchołka - wierzchołek już dodano do grafu
print(graf.wierzcholki)  # [0, 1, 2]
graf.dodaj_wierzcholek(3)
print(graf.wierzcholki)  # [0, 1, 2, 3]
print(graf.krawedzie)  # [[0, 1], [0, 2]]
graf.dodaj_krawedz([2, 3])
print(graf.krawedzie)  # [[0, 1], [0, 2], [2, 3]]
```

```
graf.dodaj_krawedz([0, 4])
 # Krawędź zawiera
wierzchołek niedodany do
grafu
print(graf.krawedzie)  #
[[0, 1], [0, 2], [2, 3]]
graf.dodaj_krawedz([3, 2])
# Nie można dodać krawędzi
- krawędź już dodano do
grafu
print(graf.krawedzie)  #
[[0, 1], [0, 2], [2, 3]]
```

11

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18BrngzT

Wynik działania programu:

```
[0, 1, 2]
Nie można dodać
wierzchołka - wierzchołek
już dodano do grafu
[0, 1, 2]
[0, 1, 2, 3]
[[0, 1], [0, 2]]
[[0, 1], [0, 2], [2, 3]]
Krawędź zawiera
wierzchołek niedodany do
grafu
[[0, 1], [0, 2], [2, 3]]
Nie można dodać krawędzi -
krawędź już dodano do
grafu
[[0, 1], [0, 2], [2, 3]]
```

Polecenie 2

Opracuj notatkę podsumowującą najważniejsze informacje przedstawione w prezentacji.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Do pojęć z zakresu teorii grafów przyporządkuj elementy, które można za ich pomocą przedstawić.

Wierzchołek

Krawędź

Graf

miasto jako sieć ulic

połączenie drogowe

relacja znajomości

wiązanie chemiczne

schemat elektroniczny

miasto jako punkt na mapie

osoba

cząsteczka

mapa połączeń drogowych

Ćwiczenie 2

Wskaż, z ilu wierzchołków (reprezentowanych liczbami naturalnymi) składa się graf spójny o podanej liście krawędzi:

```
krawedzie= [[1, 0], [1, 2], [1, 3], [2, 3], [3, 4], [5, 1], [5, 3]
```

Graf spójny to graf, w którym dowolne dwa wierzchołki są połączone drogą.

- ◯ 6
- ◯ 7
- ◯ 8
- ◯ 5

Ćwiczenie 3

Napisz funkcję sprawdzającą, czy w grafie reprezentowanym przez listę krawędzi istnieje przekazana jako parametr droga. Działanie programu przetestuj dla następujących danych:

```
krawedzie = [[1, 2], [3, 1], [1, 4], [2, 4], [3, 4]]
droga1 = [1, 2, 4, 3]
droga2 = [1, 2, 3, 4]
```

Specyfikacja problemu:

Dane:

- `krawedzie` – tablica krawędzi w grafie składająca się z par liczb całkowitych
- `droga1` – tablica liczb całkowitych
- `droga2` – tablica liczb całkowitych

Wynik:

Program wypisuje True, jeśli droga istnieje, lub False, jeśli nie istnieje.

Przykładowe wyjście:

```
True
False
```

Twoje zadania

1. Zdefiniowanie funkcji `znajdz_droge()` sprawdzającej, czy w grafie reprezentowanym przez listę krawędzi istnieje dana droga w postaci tablicy `droga` indeksów wierzchołków. Funkcja ma zwrócić odpowiednio `True` lub `False`. Zastosuj wcześniej zaimplementowaną funkcję `znajdz_krawedz()`.

```python
def znajdz_krawedz(graf, p, q):
    for i, k in enumerate(graf):
        if k == [p, q] or k == [q, p]:
            return i
    return -1

def znajdz_droge(graf, droga):

# uzupełnij kod

    return True

krawedzie = [[1, 2], [3, 1], [1, 4], [2, 4], [3, 4]]
```

```python

```

Ćwiczenie 4

Uzupełnij brakujący kod metody `znajdz_krawedz(wierzcholek1, wierzcholek2)` w klasie `Graf`, która sprawdzi, czy w grafie istnieje krawędź łącząca dane wierzchołki. Działanie programu przetestuj dla podanego w kodzie grafu `graf` i trzech par wierzchołków `wierzcholek1` i `wierzcholek2`:

- 1, 3
- 5, 4
- 2, 4

Specyfikacja problemu:

Dane:

- `graf` – podany graf; obiekt klasy `Graf`
- `wierzcholek1`, `wierzcholek2` – wierzchołki grafu; liczby naturalne

Wynik:

Program wypisuje indeksy szukanych krawędzi, jeśli zostanie znaleziona krawędź łącząca dwa wierzchołki, lub -1, jeśli nie ma między nimi połączenia.

Przykładowe wyjście:

```
1
5
-1
```

Twoje zadania

1. Program powinien wypisać w kolejnych liniach wartości uzyskane dla przykładowych wierzchołków przez funkcję `znajdz_krawedz(wierzcholek1,`

`wierzcholek2`), która zwraca indeks znalezionej krawędzi lub -1, jeśli szukana krawędź nie istnieje.

```
class Graf:
    def __init__(self, wierzcholki=[], krawedzie=[]):
        self.wierzcholki = wierzcholki
        self.krawedzie = krawedzie

    def dodaj_wierzcholek(self, wierzcholek):
        if wierzcholek not in self.wierzcholki:
            self.wierzcholki.append(wierzcholek)

    def dodaj_krawedz(self, krawedz):
        if krawedz[0] in self.wierzcholki and krawedz[1] in self.wierzcholki:
            self.krawedzie.append(krawedz)

```

```

```

Ćwiczenie 5

Napisz funkcję, która wypisze listę wierzchołków grafu wraz ze wszystkimi ich sąsiadami. Działanie programu przetestuj dla następujących danych:

```
krawedzie = [[1, 2], [1, 3], [2, 3], [3, 4], [5, 1], [5, 3], [5
wierzcholki = [1, 2, 3, 4, 5, 6]
```

Specyfikacja problemu:

Dane:

- `krawedzie` – tablica krawędzi w grafie składająca się z par liczb całkowitych
- `wierzcholki` – tablica wierzchołków w grafie; tablica liczb całkowitych

Wynik:

Program wypisuje w kolejnych wierszach indeksy wierzchołków oraz po dwukropku indeksy sąsiadów danego wierzchołka.

Przykładowe wyjście:

```
1: 2 3 5
2: 1 3
3: 1 2 4 5
4: 3
5: 1 3 6
6: 5
```

Twoje zadania

1. Zdefiniowanie funkcji `wypisz_graf()`, która wypisze listę wszystkich wierzchołków wraz z listą sąsiadów każdego wierzchołka grafu reprezentowanego przez listę krawędzi. Zastosuj gotową funkcję `lista_sasiadow()`. Indeksy sąsiadów, jak i kolejność wierzchołków, dla których listy wypisujemy, muszą być posortowane rosnąco.

```python
def lista_sasiadow(graf, wierzcholek):
    sasiedzi = []
    for krawedz in graf:
        if krawedz[0] == wierzcholek:
            sasiedzi.append(krawedz[1])

        if krawedz[1] == wierzcholek:
            sasiedzi.append(krawedz[0])
    sasiedzi.sort()
    return sasiedzi

def wypisz_graf(graf):
    # Dokończ kod tutaj

```

```
1
```

Ćwiczenie 6

Zmodyfikuj funkcję `wypisz_graf` tak, aby program wypisywał stopnie kolejnych wierzchołków (stopień wierzchołka jest równy liczbie wszystkich krawędzi, dla których dany wierzchołek jest końcem lub początkiem). Graf reprezentowany jest przez listę krawędzi. Każdy wierzchołek może pojawić się tylko raz. Działanie programu przetestuj dla następujących danych:

```
krawedzie = [[0, 6], [1, 2], [1, 3], [2, 6], [3, 4], [5, 1], [5
wierzcholki = [0, 1, 2, 3, 4, 5, 6]
```

Specyfikacja problemu:

Dane:

- `krawedzie` – tablica krawędzi w grafie składająca się z par liczb całkowitych
- `wierzcholki` – tablica wierzchołków w grafie; tablica liczb całkowitych

Wynik:

Program wypisuje stopnie kolejnych wierzchołków.

Przykładowe wyjście:

```
0 : 1
1 : 3
2 : 2
3 : 3
4 : 1
5 : 3
6 : 3
```

Twoje zadania

1. Zmodyfikuj funkcję `wypisz_graf` tak, aby program wypisywał stopnie kolejnych wierzchołków.

```python
def lista_sasiadow(graf, wierzcholek):
    sasiedzi = []
    for krawedz in graf:
        if krawedz[0] == wierzcholek:
            sasiedzi.append(krawedz[1])

        if krawedz[1] == wierzcholek:
            sasiedzi.append(krawedz[0])
    sasiedzi.sort()
    return sasiedzi

# wprowadź modyfikację do funkcji poniżej tego wiersza
def wypisz_graf(graf):
    for w in wierzcholki:
```

```python

```

Dla nauczyciela

Autor: Jakub Tarkowski
Przedmiot: Informatyka

Temat: Wprowadzenie do teorii grafów w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) w zależności od problemu rozwiązuje go, stosując metodę wstępującą lub zstępującą;

2) do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;

3) objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) projektuje i tworzy rozbudowane programy w procesie rozwiązywania problemów, wykorzystuje w programach dobrane do algorytmów struktury danych, w tym struktury dynamiczne i korzysta z dostępnych bibliotek dla tych struktur;

2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

j) grafy (do przedstawiania abstrakcyjnego modelu sytuacji problemowych).

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Zaimplementujesz rozwiązania prostych problemów dotyczących teorii grafów w języku Python.
- Zapoznasz się z przykładową implementacją grafu w języku Python.
- Zaimplementujesz klasę grafu, zawierającą wszystkie potrzebne dane na temat wierzchołków i krawędzi.
- Rozwiążesz problemy związane z sąsiedztwem wierzchołków grafów.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

Przebieg lekcji

Przed lekcją:

1. Uczniowie wyszukują informacje na temat problemów, w których rozwiązaniu można zastosować algorytmy grafowe.
2. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Wprowadzenie do teorii grafów w języku Python". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Uczniowie odpowiadają na pytanie o to, w jakich sytuacjach może mieć zastosowanie graf.
2. Nauczyciel wyświetla i odczytuje temat lekcji oraz cele zajęć. Prosi uczniów o sformułowanie kryteriów sukcesu.
3. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające, prosi wybraną osobę o przedstawienie najważniejszych informacji z sekcji „Przeczytaj".
   Uczniowie analizują przykład z sekcji „Przeczytaj". Następnie indywidualnie wykonują Ćwiczenie 1 z tej sekcji. Nauczyciel sprawdza poprawność rozwiązań.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Prezentacja multimedialna". Uczniowie wspólnie zapoznają się z treścią zawartego w niej multimedium. Analizują przedstawiony program, a następnie testują go na swoich komputerach. Zapisują ewentualne problemy i pytania. Po czym następuje dyskusja, w trakcie której nauczyciel wyjaśnia niezrozumiałe treści.

3. W kolejnym etapie uczniowie dobierają się w pary i wykonują ćwiczenia nr 1–5 z sekcji „Sprawdź się". Następnie konsultują swoje rozwiązania z inną parą uczniów i ustalają jedną wersję odpowiedzi.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Wybrany uczeń podsumowuje zajęcia z programowania w Pythonie, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie wykonują Polecenie 1 z sekcji „Przeczytaj".
2. Uczniowie wykonują Ćwiczenie 6 z sekcji „Sprawdź się".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.
- Robin J. Wilson, *Wprowadzenie do teorii grafów*, PWN, Warszawa 2007.

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści w sekcjach: „Przeczytaj", „Prezentacja multimedialna", „Sprawdź się" jako materiał do lekcji powtórkowej.