Wprowadzenie do teorii grafów w języku Java

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 intormacji na ten temat znajdziesz w e-materiale Wprowadzenie do teorii grafów.

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

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 Python.

Twoje cele

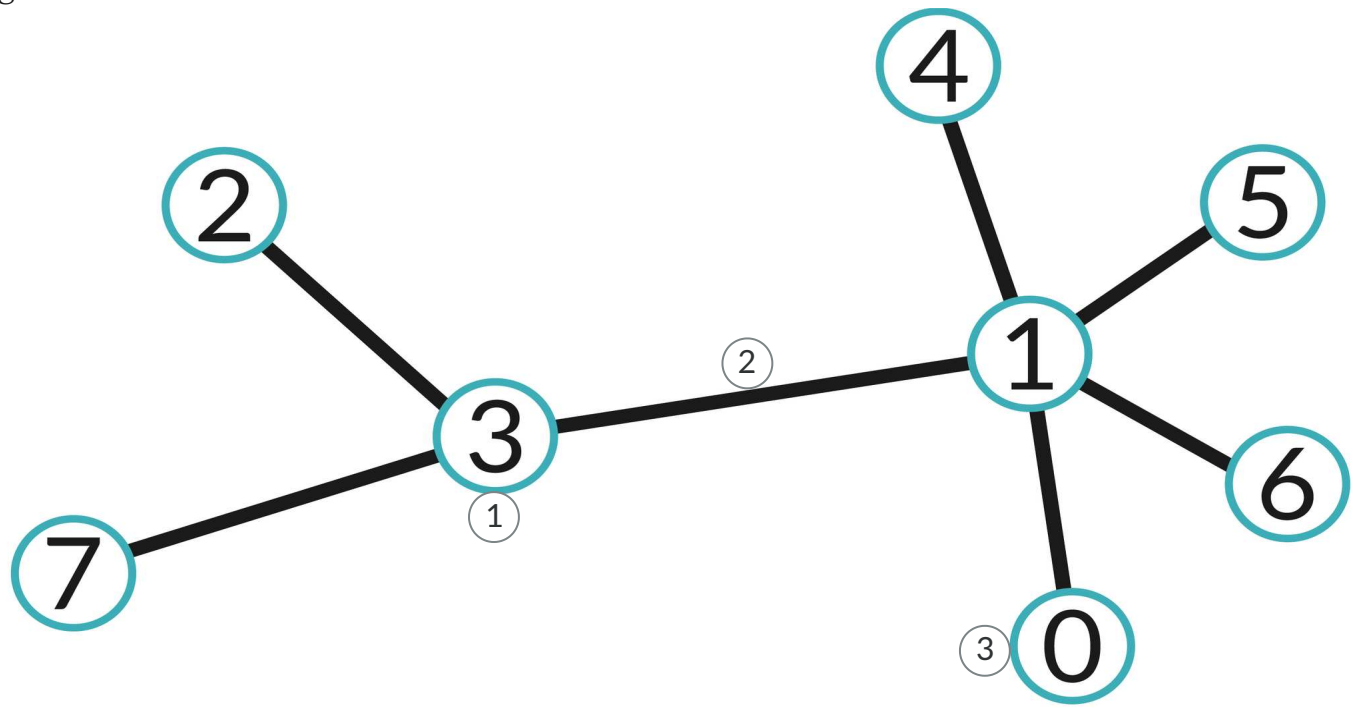
- Zaimplementujesz rozwiązania prostych problemów dotyczących teorii grafów w języku Java.
- Zapoznasz się z przykładową implementacją grafu w języku Java.
- Zaimplementujesz klasę grafu, zawierającą wszystkie potrzebne dane na temat wierzchołków i krawędzi.

- 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.

```
List<Wierzcholek> wierzcholki
```

Podobnie w przypadku krawędzi:

```
List<Krawedz> krawedzie
```

Klasa grafu będzie wyglądać następująco:

```
class Graf {
    private List<Integer> wierzcholki;
    private List<List<Integer>> krawedzie;

    public Graf() {
        this.wierzcholki = new ArrayList<>();
        this.krawedzie = new ArrayList<>();
    }

    public Graf(List<Integer> wierzcholki) {
        this.wierzcholki = wierzcholki;
        this.krawedzie = new ArrayList<>();
    }

    public Graf(List<Integer> wierzcholki, List<List<Integer>> kr
        this.wierzcholki = wierzcholki;
        this.krawedzie = krawedzie;
    }

    public List<Integer> getWierzcholki() {
        return wierzcholki;
    }

    public List<List<Integer>> getKrawedzie() {
        return krawedzie;
    }
}
```

Podaliśmy w listach również typ danych, które będą za ich pomocą przechowywane.

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:

By wyświetlić wierzchołki obiektu `graf` klasy `Graf`, wykorzystamy polecenie `System.out.println`:

```
public class Main {
    public static void main(String[] args) {
        Graf graf = new Graf(Arrays.asList(0, 1, 2));
        System.out.println(graf.getWierzcholki());
    }
}
```

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

Załóżmy, że nasz graf ma dwie krawędzie – jedna łączy wierzchołek 0 oraz 1, druga łączy wierzchołek 0 oraz 2.

Dodajmy do utworzonego obiektu krawędzie łączące wierzchołki:

```
public class Main {
    public static void main(String[] args) {
        Graf graf = new Graf(Arrays.asList(0, 1, 2));
        System.out.println(graf.getWierzcholki());

        List<List<Integer>> krawedzie = new ArrayList<>();
        krawedzie.add(Arrays.asList(0, 1));
        krawedzie.add(Arrays.asList(0, 2));

        graf = new Graf(Arrays.asList(0, 1, 2), krawedzie);
        System.out.println(graf.getKrawedzie());
    }
}
```

Kod wyświetlający wierzchołki oraz krawędzie:

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

class Graf {
    private List<Integer> wierzcholki;
    private List<List<Integer>> krawedzie;

    public Graf() {
        this.wierzcholki = new ArrayList<>();
        this.krawedzie = new ArrayList<>();
    }

    public Graf(List<Integer> wierzcholki) {
        this.wierzcholki = wierzcholki;
        this.krawedzie = new ArrayList<>();
    }

    public Graf(List<Integer> wierzcholki, List<List<Integer>> kr
        this.wierzcholki = wierzcholki;
        this.krawedzie = krawedzie;
    }

    public List<Integer> getWierzcholki() {
        return wierzcholki;
    }

    public List<List<Integer>> getKrawedzie() {
        return krawedzie;
    }
}

public class Main {
    public static void main(String[] args) {
        Graf graf = new Graf(Arrays.asList(0, 1, 2));
        System.out.println(graf.getWierzcholki());

        List<List<Integer>> krawedzie = new ArrayList<>();
        krawedzie.add(Arrays.asList(0, 1));
        krawedzie.add(Arrays.asList(0, 2));

        graf = new Graf(Arrays.asList(0, 1, 2), krawedzie);
```

```
43          System.out.println(graf.getKrawedzie());
44      }
45  }
```

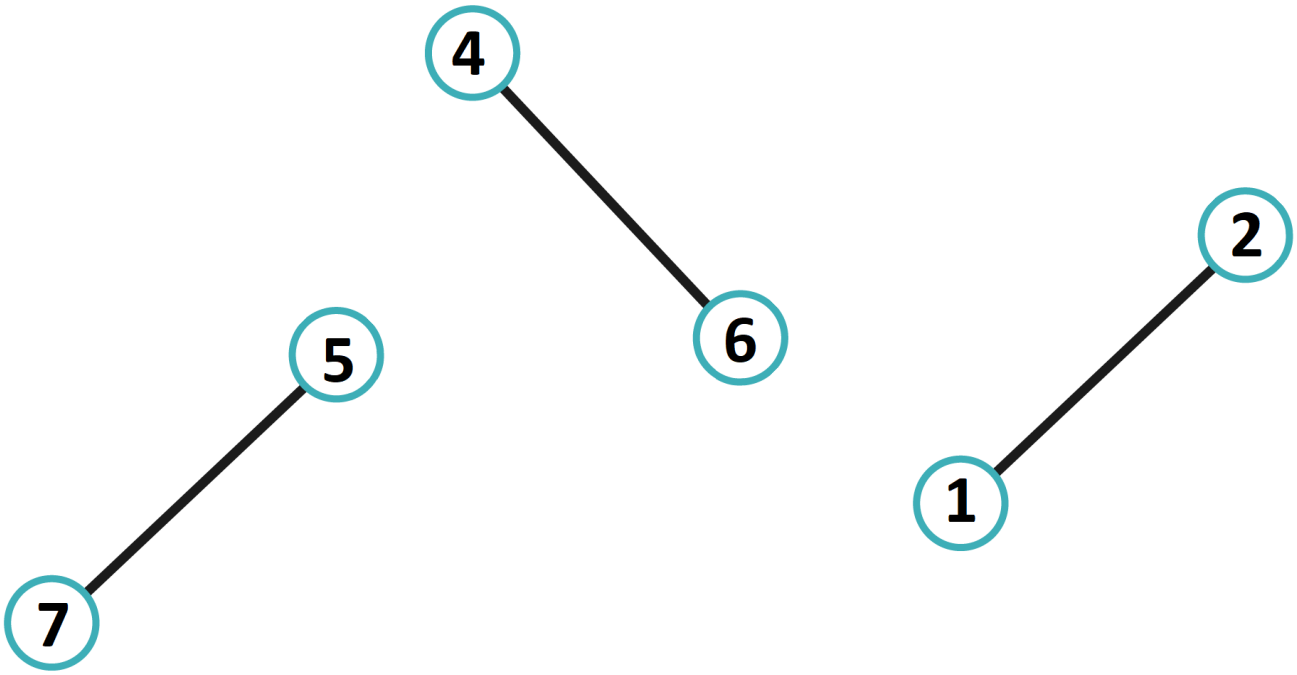
Wynik jego działania:

```
1  [0, 1, 2] // wierzchołki
2  [[0, 1], [0, 2]] // krawędzie
```

Graf opisany za pomocą poniższej klasy:

```
1  public class Main {
2      public static void main(String[] args) {
3          List<Integer> wierzcholki = Arrays.asList(1, 2, 4, 5, 6,
4
5          List<List<Integer>> krawedzie = new ArrayList<>();
6          krawedzie.add(Arrays.asList(7, 5));
7          krawedzie.add(Arrays.asList(4, 6));
8          krawedzie.add(Arrays.asList(1, 2));
9
10         Graf graf = new Graf(wierzcholki, krawedzie);
11     }
12 }
```

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.

```
import java.util.Arrays;
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Integer> wierzcholki = Arrays.asList(0, 1, 2, 3, 4,

        Graf graf = new Graf(wierzcholki, krawedzie);
    }
}
```

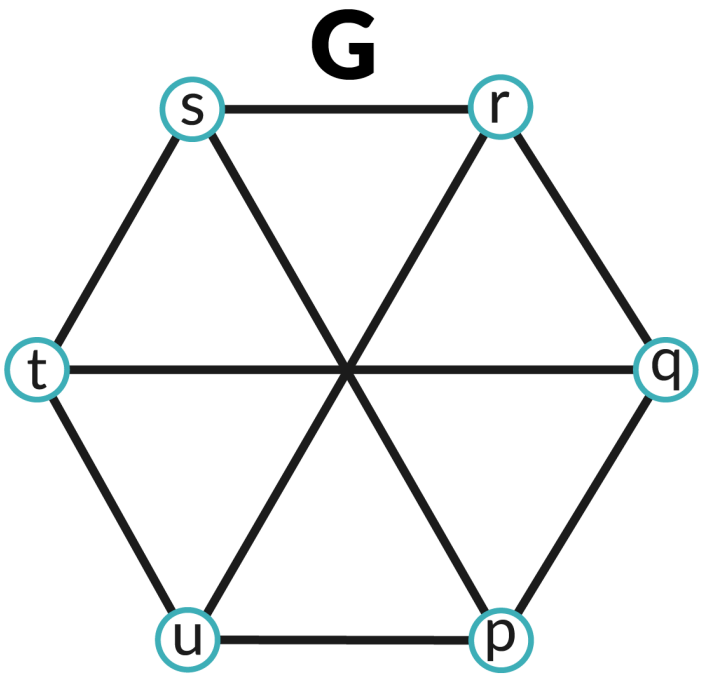
Mamy do czynienia z grafem spójnym, więc jego wierzchołki połączone są krawędziami.

```
import java.util.Arrays;
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Integer> wierzcholki = Arrays.asList(0, 1, 2, 3, 4,

        List<List<Integer>> krawedzie = new ArrayList<>();
        krawedzie.add(Arrays.asList(7, 3));
        krawedzie.add(Arrays.asList(2, 3));
        krawedzie.add(Arrays.asList(3, 1));
        krawedzie.add(Arrays.asList(4, 1));
        krawedzie.add(Arrays.asList(5, 1));
```

```
        krawedzie.add(Arrays.asList(6, 1));
        krawedzie.add(Arrays.asList(0, 1));

        Graf graf = new Graf(wierzcholki, krawedzie);
    }
}
```

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.

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

Zacznijmy od zaimportowania niezbędnych klas z pakietu `java.util`. Będą potrzebne do tworzenia list i manipulowania nimi.

```
1 import
  java.util.ArrayList;
2 import java.util.Arrays;
3 import
  java.util.Collections;
4 import java.util.List;
```

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

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

Zacznijmy od jej deklaracji.

```
1 class Graf {
2
3 }
```

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

Zadeklarujemy również dwa prywatne pola tej klasy. Pierwsze z nich to pole `wierzcholki`, czyli lista przechowująca wierzchołki grafu. Drugie z nich to pole `krawedzie`, czyli lista, która przechowuje pary wierzchołków połączonych krawędzią.

```
1 class Graf {
2     private List<Integer>
  wierzcholki;
3     private
  List<List<Integer>>
  krawedzie;
4 }
```

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

W kolejnym kroku zdefiniujemy domyślny konstruktor klasy `Graf`, który zainicjalizuje listy `wierzcholki` i `krawedzie` jako puste listy.

```
1 class Graf {
2     private List<Integer>
  wierzcholki;
3     private
  List<List<Integer>>
  krawędzie;
4
5     public Graf() {
6         this.wierzcholki =
  new ArrayList<>();
```

```
        this.krawedzie =
new ArrayList<>();
    }
```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

5

Definiujemy drugi konstruktor, który przyjmuje listy `wierzcholki` i `krawedzie` jako argumenty, a następnie przypisuje je do odpowiednich pól klasy.

```
class Graf {
    private List<Integer>
wierzcholki;
    private
List<List<Integer>>
krawedzie;

    public Graf() {
        this.wierzcholki =
new ArrayList<>();
        this.krawedzie =
new ArrayList<>();
    }

    public
Graf(List<Integer>
wierzcholki,
List<List<Integer>>
krawedzie) {
        this.wierzcholki =
wierzcholki;
        this.krawedzie =
krawedzie;
    }
```

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

Dodajemy odpowiednie metody.

Metoda `dodajWierzcholek()` służy do dodawania wierzchołka do grafu. Sprawdza, czy dany wierzchołek jest już w grafie. Jeśli tak, wyświetla komunikat. Jeśli nie, dodaje go do listy wierzchołków.

Metoda `dodajKrawedz()` służy do dodawania krawędzi do grafu. Sprawdza, czy wierzchołki krawędzi są już w grafie. Jeśli nie, wyświetla komunikat. Jeśli tak, sprawdza, czy taka krawędź jest już w grafie. Jeśli ten warunek jest prawdziwy, wyświetlany jest komunikat. W przeciwnym wypadku krawędź jest dodawana do listy krawędzi.

```
1  class Graf {
2      private List<Integer>
   wierzcholki;
3      private
   List<List<Integer>>
   krawedzie;
4
5      public Graf() {
6          this.wierzcholki =
   new ArrayList<>();
7          this.krawedzie =
   new ArrayList<>();
8      }
9
10     public
   Graf(List<Integer>
   wierzcholki,
   List<List<Integer>>
   krawedzie) {
11         this.wierzcholki =
   wierzcholki;
12         this.krawedzie =
   krawedzie;
13     }
```

```
14
15      public void dodajWierzcholek(Integer wierzcholek) {
16          if (this.wierzcholki.contains(wierzcholek)) {
17              System.out.println("Nie można dodać wierzchołka - wierzchołek już dodano do grafu");
18          } else {
19              this.wierzcholki.add(wierzcholek);
20          }
21      }
22
23      public void dodajKrawedz(List<Integer> krawedz) {
24          if (!this.wierzcholki.containsAll(krawedz)) {
25              System.out.println("Krawędź zawiera wierzchołek niedodany do grafu");
26          } else {
27              boolean isExists = this.krawedzie.stream().anyMatch(k -> {
28                  Collections.sort(k);
29                  Collections.sort(krawedz);
30                  return k.equals(krawedz);
31              });
32
33              if (isExists) {
```

```
34
                System.out.println("Nie
                można dodać krawędzi -
                krawędź już dodano do
                grafu");
35              } else {
36
                this.krawedzie.add(krawed
                z);
37              }
38          }
39      }
```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

7

Definiujemy metody dostępowe (`getters`) dla pól `wierzcholki` i `krawedzie`.

Metody dostępowe zwracają wartość odpowiednich pól klasy (w tym wypadku wierzchołki i krawędzie).

```
1  class Graf {
2      private List<Integer>
       wierzcholki;
3      private
       List<List<Integer>>
       krawedzie;
4
5      public Graf() {
6          this.wierzcholki =
       new ArrayList<>();
7          this.krawedzie =
       new ArrayList<>();
8      }
9
10     public
       Graf(List<Integer>
       wierzcholki,
```

```
List<List<Integer>>
krawedzie) {
11          this.wierzcholki =
wierzcholki;
12          this.krawedzie =
krawedzie;
13      }
14
15      public void
dodajWierzcholek(Integer
wierzcholek) {
16          if
(this.wierzcholki.contains
(wierzcholek)) {
17
 System.out.println("Nie
można dodać wierzchołka -
wierzchołek już dodano do
grafu");
18          } else {
19
 this.wierzcholki.add(wier
zcholek);
20          }
21      }
22
23      public void
dodajKrawedz(List<Integer>
krawedz) {
24          if
(!this.wierzcholki.contain
sAll(krawedz)) {
25
 System.out.println("Krawę
dź zawiera wierzchołek
niedodany do grafu");
26          } else {
27              boolean
isExists =
this.krawedzie.stream().an
yMatch(k -> {
28
 Collections.sort(k);
29
 Collections.sort(krawedz)
```

```
;
30                  return
k.equals(krawedz);
31              });
32
33              if (isExists)
{
34
 System.out.println("Nie
można dodać krawędzi -
krawędź już dodano do
grafu");
35              } else {
36
 this.krawedzie.add(krawed
z);
37              }
38          }
39      }
40
41      public List<Integer>
getWierzcholki() {
42          return
this.wierzcholki;
43      }
44
45      public
List<List<Integer>>
getKrawedzie() {
46          return
this.krawedzie;
47      }
```

8

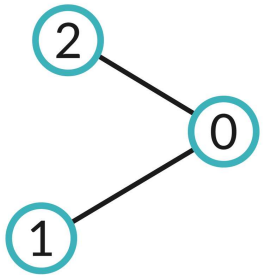
Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

W kolejnym kroku zadeklarujemy publiczną klasę `Main`, która będzie zawierać metodę `main`, czyli punkt startowy dla programu w języku Java.

Tworzymy instancję klasy `Graf`, dodajemy wierzchołki i krawędzie, a potem wyświetlamy wyniki na ekranie.

```
1    public static void
main(String[] args) {
2
3 }
```



Przykładowy graf o podanych wierzchołkach i krawędziach.
Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

Metoda `main` jest punktem startowym dla każdego programu w języku Java. Argumenty linii poleceń są przekazywane do metody `main` jako tablica łańcuchów znaków.

Zaczniemy od utworzenia list wierzchołków `wierzcholki` oraz krawędzi `krawedzie`. Dodajemy do drugiej listy dwie krawędzie: `(0, 1)` i `(0, 2)`.

```
1    public static void
main(String[] args) {
2        List<Integer>
wierzcholki = new
ArrayList<>
(Arrays.asList(0, 1, 2));
3
 List<List<Integer>>
```

```
krawedzie = new
ArrayList<>();
4
 krawedzie.add(Arrays.asLi
st(0, 1));
5
 krawedzie.add(Arrays.asLi
st(0, 2));
6
7 }
```

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

Tworzymy nową instancję klasy `Graf`, przekazując do konstruktora wcześniej stworzone listy wierzchołków i krawędzi.

```
1    public static void
main(String[] args) {
2         List<Integer>
wierzcholki = new
ArrayList<>
(Arrays.asList(0, 1, 2));
3
 List<List<Integer>>
krawedzie = new
ArrayList<>();
4
 krawedzie.add(Arrays.asLi
st(0, 1));
5
 krawedzie.add(Arrays.asLi
st(0, 2));
6
7         Graf graf = new
Graf(wierzcholki,
krawedzie);
8
9 }
```

11

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

Wyświetlamy listę wierzchołków grafu na standardowe wyjście.

```
1     public static void
main(String[] args) {
2         List<Integer>
wierzcholki = new
ArrayList<>
(Arrays.asList(0, 1, 2));
3
 List<List<Integer>>
krawedzie = new
ArrayList<>();
4
 krawedzie.add(Arrays.asLi
st(0, 1));
5
 krawedzie.add(Arrays.asLi
st(0, 2));
6
7         Graf graf = new
Graf(wierzcholki,
krawedzie);
8
9
 System.out.println(graf.g
etWierzcholki());
10
11 }
```

12

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

Spróbujemy dodać wierzchołek o wartości 0 do grafu.

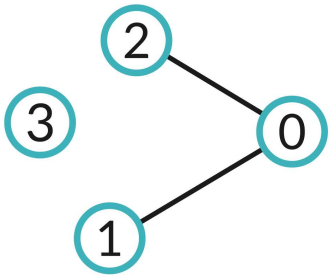
Wierzchołek o tej wartości już istnieje, więc metoda wyświetla odpowiedni komunikat.

```
1     public static void
main(String[] args) {
2         List<Integer>
wierzcholki = new
ArrayList<>
(Arrays.asList(0, 1, 2));
3
 List<List<Integer>>
krawedzie = new
ArrayList<>();
4
 krawedzie.add(Arrays.asLi
st(0, 1));
5
 krawedzie.add(Arrays.asLi
st(0, 2));
6
7         Graf graf = new
Graf(wierzcholki,
krawedzie);
8
 System.out.println(graf.g
etWierzcholki());
9
 graf.dodajWierzcholek(0);
10
11 }
```

Ponownie wyświetlamy listę wierzchołków grafu. Lista nie zmieniła się.

```
1     public static void
main(String[] args) {
2         List<Integer>
wierzcholki = new
ArrayList<>
(Arrays.asList(0, 1, 2));
3
 List<List<Integer>>
```

```
krawedzie = new
ArrayList<>();
4
 krawedzie.add(Arrays.asLi
st(0, 1));
5
 krawedzie.add(Arrays.asLi
st(0, 2));
6
        Graf graf = new
Graf(wierzcholki,
krawedzie);
8
 System.out.println(graf.g
etWierzcholki());
9
 graf.dodajWierzcholek(0);
10
 System.out.println(graf.g
etWierzcholki());
11
12 }
```

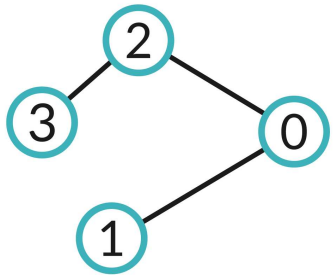
Przykładowy graf o podanych wierzchołkach i krawędziach po dodaniu do niego wierzchołka oznaczonego jako 3.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

Dodajemy wierzchołek o wartości 3 do grafu. Wyświetlamy zaktualizowaną listę wierzchołków grafu oraz krawędzi.

```
    public static void main(String[] args) {
        List<Integer> wierzcholki = new ArrayList<> (Arrays.asList(0, 1, 2));

 List<List<Integer>> krawedzie = new ArrayList<>();

 krawedzie.add(Arrays.asList(0, 1));

 krawedzie.add(Arrays.asList(0, 2));

        Graf graf = new Graf(wierzcholki, krawedzie);

 System.out.println(graf.getWierzcholki());

 graf.dodajWierzcholek(0);

 System.out.println(graf.getWierzcholki());

 graf.dodajWierzcholek(3);

 System.out.println(graf.getWierzcholki());

 System.out.println(graf.getKrawedzie());

}
```



Przykładowy graf o podanych wierzchołkach i krawędziach po dodaniu krawędzi.
Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

Wykonujemy serię operacji dodawania krawędzi do grafu i wyświetlania listy krawędzi po każdej operacji. Niektóre z nich nie powiodą się ze względu na zasady zdefiniowane w metodzie `dodajKrawedz`. Na przykład próba dodania krawędzi `(0, 4)` nie powiedzie się, ponieważ nie ma wierzchołka o wartości 4 w grafie. Próba dodania krawędzi `(3, 2)` również się nie powiedzie, ponieważ krawędź `(2, 3)` już istnieje (kierunek nie ma znaczenia).

```
1    public static void
main(String[] args) {
2        List<Integer>
wierzcholki = new
ArrayList<>
(Arrays.asList(0, 1, 2));
3
 List<List<Integer>>
krawedzie = new
ArrayList<>();
4
 krawedzie.add(Arrays.asLi
st(0, 1));
5
 krawedzie.add(Arrays.asLi
st(0, 2));
6
```

```
7         Graf graf = new
Graf(wierzcholki,
krawedzie);
8
 System.out.println(graf.g
etWierzcholki());
9
 graf.dodajWierzcholek(0);
10
 System.out.println(graf.g
etWierzcholki());
11
 graf.dodajWierzcholek(3);
12
 System.out.println(graf.g
etWierzcholki());
13
 System.out.println(graf.g
etKrawedzie());
14
 graf.dodajKrawedz(Arrays.
asList(2, 3));
15
 System.out.println(graf.g
etKrawedzie());
16
 graf.dodajKrawedz(Arrays.
asList(0, 4));
17
 System.out.println(graf.g
etKrawedzie());
18
 graf.dodajKrawedz(Arrays.
asList(0, 4));
19
 System.out.println(graf.g
etKrawedzie());
20
 graf.dodajKrawedz(Arrays.
asList(3, 2));
21
 System.out.println(graf.g
etKrawedzie());
22     }
23 }
```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P18cMKly8

Zapisaliśmy cały program.

```
1  import
   java.util.ArrayList;
2  import java.util.Arrays;
3  import
   java.util.Collections;
4  import java.util.List;
5
6  class Graf {
7      private List<Integer>
   wierzcholki;
8      private
   List<List<Integer>>
   krawedzie;
9
10     public Graf() {
11         this.wierzcholki =
   new ArrayList<>();
12         this.krawedzie =
   new ArrayList<>();
13     }
14
15     public
   Graf(List<Integer>
   wierzcholki,
   List<List<Integer>>
   krawedzie) {
16         this.wierzcholki =
   wierzcholki;
17         this.krawedzie =
   krawedzie;
18     }
19
20     public void
   dodajWierzcholek(Integer
```

```
wierzcholek) {
21          if
(this.wierzcholki.contains
(wierzcholek)) {
22
 System.out.println("Nie
można dodać wierzchołka -
wierzchołek już dodano do
grafu");
23          } else {
24
 this.wierzcholki.add(wier
zcholek);
25          }
26      }
27
28      public void
dodajKrawedz(List<Integer>
krawedz) {
29          if
(!this.wierzcholki.contain
sAll(krawedz)) {
30
 System.out.println("Krawę
dź zawiera wierzchołek
niedodany do grafu");
31          } else {
32              boolean
isExists =
this.krawedzie.stream().an
yMatch(k -> {
33
 Collections.sort(k);
34
 Collections.sort(krawedz)
;
35                  return
k.equals(krawedz);
36              });
37
38              if (isExists)
{
39
 System.out.println("Nie
można dodać krawędzi -
```

```
krawędź już dodano do grafu");
            } else {
 this.krawedzie.add(krawedz);
            }
        }
    }

    public List<Integer> getWierzcholki() {
        return this.wierzcholki;
    }

    public List<List<Integer>> getKrawedzie() {
        return this.krawedzie;
    }

    public static void main(String[] args) {
        List<Integer> wierzcholki = new ArrayList<>(Arrays.asList(0, 1, 2));

 List<List<Integer>> krawedzie = new ArrayList<>();

 krawedzie.add(Arrays.asList(0, 1));

 krawedzie.add(Arrays.asList(0, 2));

        Graf graf = new Graf(wierzcholki, krawedzie);

 System.out.println(graf.g
```

```
etWierzcholki());
62
 graf.dodajWierzcholek(0);
// Nie można dodać
wierzchołka - wierzchołek
już dodano do grafu
63
 System.out.println(graf.g
etWierzcholki());  // [0,
1, 2]
64
 graf.dodajWierzcholek(3);
65
 System.out.println(graf.g
etWierzcholki());  // [0,
1, 2, 3]
66
 System.out.println(graf.g
etKrawedzie());  // [[0,
1], [0, 2]]
67
 graf.dodajKrawedz(Arrays.
asList(2, 3));
68
 System.out.println(graf.g
etKrawedzie());  // [[0,
1], [0, 2], [2, 3]]
69
 graf.dodajKrawedz(Arrays.
asList(0, 4));  // Krawędź
zawiera wierzchołek
niedodany do grafu
70
 System.out.println(graf.g
etKrawedzie());  // [[0,
1], [0, 2], [2, 3]]
71
 graf.dodajKrawedz(Arrays.
asList(0, 4));  // Krawędź
zawiera wierzchołek
niedodany do grafu
72
 System.out.println(graf.g
etKrawedzie());  // [[0,
1], [0, 2], [2, 3]]
```

```
73
 graf.dodajKrawedz(Arrays.
asList(3, 2)); // Nie
można dodać krawędzi -
krawędź już dodano do
grafu
74
 System.out.println(graf.g
etKrawedzie());  // [[0,
1], [0, 2], [2, 3]]
75     }
76 }
```

16 Wynik działania programu:

```
1  [0, 1, 2]
2  Nie można dodać
   wierzchołka - wierzchołek
   już dodano do grafu
3  [0, 1, 2]
4  [0, 1, 2, 3]
5  [[0, 1], [0, 2]]
6  [[0, 1], [0, 2], [2, 3]]
7  Krawędź zawiera
   wierzchołek niedodany do
   grafu
8  [[0, 1], [0, 2], [2, 3]]
9  Krawędź zawiera
   wierzchołek niedodany do
   grafu
10 [[0, 1], [0, 2], [2, 3]]
11 Nie można dodać krawędzi -
   krawędź już dodano do
   grafu
12 [[0, 1], [0, 2], [2, 3]]
```

Polecenie 2

Opracuj notatkę podsumowującą najważniejsze informacje przedstawione w prezentacji.

Polecenie 3

Zastanów się, jakiego typu problemy programistyczne (ale nie tylko) można rozwiązywać za pomocą algorytmów grafowych. Podaj ich przykłady.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

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

Wierzchołek

Krawędź

Graf

wiązanie chemiczne

relacja znajomości

połączenie drogowe

miasto jako sieć ulic

schemat elektroniczny

cząsteczka

osoba

mapa połączeń drogowych

miasto jako punkt na mapie

Ćwiczenie 2

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

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

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

- ◯ 5
- ◯ 6
- ◯ 8
- ◯ 7

Ćwiczenie 3

Uzupełnij brakujący kod metody `znajdzKrawedz(int wierzcholek1, int 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:

Dane:

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

Wynik:

Program, na standardowe wyjście, wypisuje indeks krawędzi, jeśli zostanie znaleziona krawędź łącząca dwa wierzchołki, lub -1, jeśli nie ma między nimi połączenia.

Twoje zadania

1. Program powinien wypisać indeksy znalezionych krawędzi lub −1, jeśli szukana krawędź nie istnieje.

```
import java.util.List;
import java.util.ArrayList;

class Graf{
    List<Integer> wierzcholki;
```

```
6     List<List<Integer>> krawedzie;
7
8     Graf(){
9         wierzcholki = new ArrayList<>();
10        krawedzie = new ArrayList<>();
11    }
12
13    public void dodajWierzcholek(int nowyWierzcholek) {
14        for (int wierzcholek : this.wierzcholki)
```

```
1
```

Ćwiczenie 4

Uzupełnij brakujący kod metody `listaSasiadow(int wierzcholek)` w klasie `Graf`, która zwróci posortowaną rosnąco listę sąsiadów danego wierzchołka `wierzcholek`. Działanie programu przetestuj dla podanego w kodzie grafu `graf` i wierzchołka `wierzcholek = 1`.

Specyfikacja:

Dane:

- `graf` – podany graf; obiekt klasy `Graf`
- `wierzcholek` – wierzchołek grafu; liczba naturalna

Wynik:

Program, na standardowe wyjście, wypisuje posortowaną rosnąco listę sąsiadów wierzchołka `wierzcholek`.

Twoje zadania

1. Program powinien wypisać listę indeksów wierzchołków sąsiadujących z wierzchołkiem `wierzcholek`, zwróconą przez funkcję `listaSasiadow(int wierzcholek)`. Indeksy sąsiadów muszą być posortowane rosnąco.

```
import java.util.List;
import java.util.ArrayList;

class Graf{
    List<Integer> wierzcholki;
    List<List<Integer>> krawedzie;

    Graf(){
        wierzcholki = new ArrayList<>();
        krawedzie = new ArrayList<>();
    }

```

```
13      public void dodajWierzcholek(int nowyWierzcholek) {
```

```
1
```

Ćwiczenie 5

Uzupełnij brakujący kod metody `wypiszGraf()` w klasie `Graf`, która dla każdego wierzchołka grafu wypisze posortowaną rosnąco listę jego sąsiadów. Działanie programu przetestuj dla podanego w kodzie grafu `graf`.

Specyfikacja:

Dane:

- `graf` – podany graf; obiekt klasy `Graf`

Wynik:

Program, na standardowe wyjście, wypisuje – w kolejnych liniach – posortowane rosnąco listy sąsiadów kolejnych wierzchołków grafu `graf`.

Twoje zadania

1. Program powinien wypisać w konsoli posortowane rosnąco listy sąsiadów wszystkich wierzchołków grafu `graf`. Listy sąsiadów kolejnych wierzchołków wypisz od nowej linii. W rozwiązaniu zastosuj gotową funkcję `listaSasiadow(int wierzcholek)`.

```
1  import java.util.List;
2  import java.util.ArrayList;
3  import java.util.Collections;
4
5  class Graf{
6      List<Integer> wierzcholki;
7      List<List<Integer>> krawedzie;
8
9      Graf(){
10         wierzcholki = new ArrayList<>();
11         krawedzie = new ArrayList<>();
12     }
13
14     public void dodajWierzcholek(int nowyWierzcholek) {
```

```
1
```

Dla nauczyciela

Autor: Jakub Tarkowski
Przedmiot: Informatyka

Temat: Wprowadzenie do teorii grafów w języku Java

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 Java.
- Zapoznasz się z przykładową implementacją grafu w języku Java.
- 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 Java SE 8 (lub nowszej wersji), w tym Eclipse 4.4 (lub nowszej wersji).

Przebieg lekcji

Przed lekcją:

1. Uczniowie przygotowują prezentacje dotyczące przykładów problemów z życia codziennego, do których rozwiązania można wykorzystać teorię grafów.
2. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Wprowadzenie do teorii grafów w języku Java". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć. Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Jeżeli przygotowanie uczniów do lekcji jest niewystarczające, nauczyciel prosi o indywidualne zapoznanie się z treścią zawartą w sekcji „Przeczytaj". Każdy uczestnik zajęć podczas cichego czytania wynotowuje najważniejsze kwestie poruszane w tekście. W kolejnym kroku nauczyciel analizuje z uczniami implementacje zawarte w tej sekcji i wyjaśnia ich działanie.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Prezentacja multimedialna". Uczniowie wspólnie zapoznają się z treścią polecenia 1, a następnie przechodzą do analizy prezentacji. Po czym następuje dyskusja, w trakcie której nauczyciel wyjaśnia niezrozumiałe treści. W kolejnym kroku uczniowie realizują polecenie 2.

3. **Ćwiczenie umiejętności.** Uczniowie indywidualnie wykonują ćwiczenia 1–4 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.
4. Uczniowie prezentują przygotowane przez siebie przykłady problemów grafowych z życia codziennego.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń z programowania w języku Java.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 5 z sekcji „Sprawdź się".
2. Uczniowie wykonują polecenie 3 z sekcji „Prezentacja multimedialna".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Java SE 8 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Eclipse 4.4 (lub nowszej wersji).

Wskazówki metodyczne:

- Treści w sekcji „Prezentacja multimedialna" można wykorzystać na lekcji jako podsumowanie i utrwalenie wiedzy uczniów.