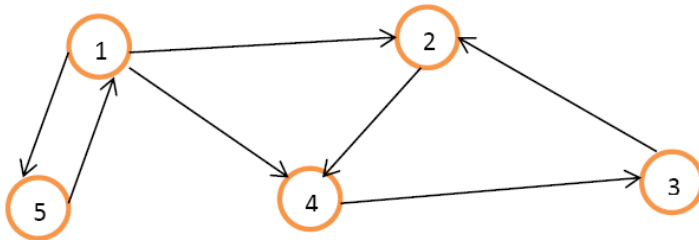


Laboratorium 3: Grafy.

1. Grafy – wprowadzenie.

Graf jest strukturą składającą się ze zbioru wierzchołków (węzłów) i zbioru krawędzi łączących wierzchołki ze sobą. Zbiór wierzchołków będziemy oznaczać przez V (od ang. vertex), a zbiór krawędzi przez E (ang. edge). Najbardziej intuicyjną reprezentacją grafu jest reprezentacja graficzna. Węzły reprezentowane są najczęściej jako koła, natomiast krawędzie jako linie (w grafach nieskierowanych) lub strzałki (w grafach skierowanych) łączące ze sobą węzły. Poniższy rysunek przedstawia graf skierowany składający się z 5 wierzchołków i 7 krawędzi:



Ten graf jest wyznaczony przez zbiór wierzchołków $V=\{1, 2, 3, 4, 5\}$ oraz następujący zbiór krawędzi: $E=\{ (1, 5), (5, 1), (1, 4), (1, 2), (2, 4), (4,3), (3, 2) \}$. Zwróćmy uwagę, że krawędź jest wyznaczona przez parę wierzchołków, a zbiór krawędzi to zbiór par wierzchołków. Formalnie E jest podzbiorem produktu kartezjańskiego $V \times V$.

2. Macierzowa reprezentacja grafu.

Reprezentacja graficzna jest intuicyjna i czytelna dla człowieka (zwłaszcza dla grafów o niewielkich rozmiarach), ale nie nadaje się do automatycznego przetwarzania przez programy komputerowe. Jedną z reprezentacji grafów ułatwiającą przetwarzanie grafów przez algorytmy jest macierz sąsiedztwa.

Macierz sąsiedztwa to macierz kwadratowa o rozmiarze równym liczbie wierzchołków grafu. Wiersze i kolumny macierzy są etykietowane kolejnymi wierzchołkami grafu. Element a_{ij} macierzy zawiera informację o krawędzi (lub jej braku) pomiędzy wierzchołkiem i a wierzchołkiem j . W najprostszym przypadku (dla grafów nieetykietowanych) możemy przyjąć, że jeśli wierzchołki i oraz j są połączone krawędzią, to $a_{ij} = 1$. Jeśli taka krawędź nie istnieje, to $a_{ij} = 0$. Poniżej macierz sąsiedztwa dla grafu przedstawionego na rysunku:

	1	2	3	4	5
1	0	1	0	1	1
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	0
5	1	0	0	0	0

3. Przeszukiwanie grafu wszerz (Breadth-First Search, BFS), wersja podstawowa.

Algorytm przeszukiwania grafu wszerz rozpoczyna działanie od wyznaczonego wierzchołka s . Wierzchołek ten jest oznaczany jako odwiedzony, żeby zapobiec powtórnemu przetwarzaniu tego wierzchołka przez algorytm, a następnie dodawany do listy Q . Na tej liście znajdują się wierzchołki, które będą odwiedzone w kolejnych krokach algorytmu. W danym kroku algorytmu pobierany jest pierwszy wierzchołek z listy Q , oznaczony przez v . Następnie odwiedzane są wszystkie wierzchołki sąsiadujące z wybranym wierzchołkiem v , które nie są jeszcze oznaczone jako odwiedzone (odwiedzenie wierzchołka polega na oznaczeniu go jako odwiedzonego oraz dodaniu go na koniec listy Q). Następnie wierzchołek v zostaje oznaczony jako przeszukany (gdy wszyscy sąsiedzi v są już odwiedzeni). Algorytm kończy działanie, gdy lista Q jest pusta.

4. Przeszukiwanie grafu w głąb (Depth-first search, DFS), wersja podstawowa.

Idea algorytmu DFS jest bardzo prosta – sięgamy w grafie „głębiej” o ile jest to możliwe. Najpierw wybieramy wierzchołek, z którego rozpoczniemy przeszukiwanie (nazwijmy go wierzchołkiem startowym). Następnie przechodzimy z wybranego wierzchołka do któregośkolwiek z jego sąsiadów, który nie został jeszcze odwiedzony. Powtarzamy powyższy krok do momentu, gdy wszyscy sąsiedzi wierzchołka v , w którym się aktualnie znajdujemy, zostali już odwiedzeni. Cofamy się wtedy do wierzchołka, z którego doszliśmy do v i szukamy innego nieodwiedzonego jeszcze wierzchołka. Proces ten jest kontynuowany do momentu, kiedy odwiedziliśmy wszystkie wierzchołki osiągalne z wierzchołka startowego. Po wykonaniu jednego takiego przejścia, jeśli w grafie pozostały jeszcze jakieś wierzchołki nieodwiedzone, to znów wybieramy wierzchołek startowy i przeszukujemy dalej. W przeciwnym wypadku kończymy działanie algorytmu, gdyż wszystkie wierzchołki zostały już odwiedzone.

PSEUDOKOD

```
funkcja odwiedzaj( $u$ ):
    kolor[ $u$ ] = SZARY
    czas = czas + 1
    poczatek[ $u$ ] = czas
    dla każdego wierzchołka  $v$  na liście sąsiedztwa  $u$ :
        jeżeli  $v$  jest biały:
            rodzic[ $v$ ] =  $u$ 
            odwiedzaj( $v$ )
    kolor[ $u$ ] = CZARNY
    czas = czas + 1
    koniec[ $u$ ] = czas

funkcja DFS(Graf  $G$ ):
    dla każdego wierzchołka  $u$  z grafu  $G$ :
        kolor[ $u$ ] = BIAŁY
        poczatek[ $u$ ] = koniec[ $u$ ] = 0
        rodzic[ $u$ ] = NIL
    czas = 0
    dla każdego wierzchołka  $u$  z grafu  $G$ :
        jeżeli  $u$  jest biały:
            odwiedzaj( $u$ )
```

5. Klasy potrzebne do rozwiązania zadań.

Dana jest klasa abstrakcyjna AGraph, którą należy pobrać bezpośrednio ze strony WWW.

Zadania.

1. Utworzyć własną klasę dziedziczącą z klasy AGraph. Zaimplementować metody abstrakcyjne check, connect, writeMatrix oraz writelist. [2p]
2. Zaimplementować metodę wypisującą wierzchołki grafu, które mają dokładnie k sąsiadów. [1p]
3. Zaimplementować metodę wypisującą wierzchołki grafu, które mają najwięcej incydenctnych krawędzi (wchodzące + wychodzące). [1p]
4. Zaimplementować metodę **void bfs(int v)**, która przeszuka graf za pomocą metody BFS, wypisze wierzchołki w kolejności odwiedzania, zaczynając od v , oraz odległości od v do wszystkich pozostałych wierzchołków. [3p]
5. Zaimplementować metody **void dfs()** oraz **void odwiedzaj(int v)** realizujące rekurencyjne przeszukiwanie grafu metodą DFS i wypisujące wierzchołki w kolejności odwiedzania. Na koniec należy wypisać tablice *poczatek*, *koniec*, *rodzic*. [3p]

Praca domowa:

1. Zaimplementować wypisywanie tablicy incydencji grafu.
2. Zaimplementować metodę *equals* sprawdzającą, czy dwa grafy są identyczne.

Na następnych zajęciach: Drzewa binarne. Drzewa BST.