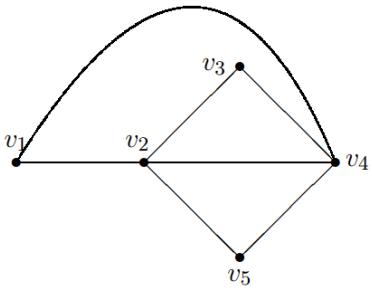
**Algorytmy i złożoność obliczeniowa**

**Laboratorium 7: Grafy. Listowa reprezentacja grafów. Algorytm DFS.**

1. **Listowa reprezentacja grafów.**

Dla każdego wierzchołka v budujemy listę, której elementami są wierzchołki połączone krawędzią z v. Lista dla grafu z poniższego rysunku wygląda następująco:




$v_1$ : $v_2$, $v_4$
$v_2$ : $v_1$, $v_3$, $v_4$, $v_5$
$v_3$ : $v_2$, $v_4$
$v_4$ : $v_1$, $v_2$, $v_3$, $v_5$
$v_5$ : $v_2$, $v_4$

2. **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 kolejnego sąsiada z listy, 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        // Szary - odwiedzony
       czas = czas + 1
       poczatek[u] = czas      // czas wejścia do wierzchołka
       dla każdego wierzchołka v na liście sąsiedztwa u:
           jeżeli v jest biały:
               rodzic[v] = u   // poprzednik - zapamiętujemy drzewo przeszukiwania
               odwiedzaj(v)
       kolor[u] = CZARNY       // Czarny - przeszukany
       czas = czas + 1
       koniec[u] = czas        // czas przeszukania wszystkich sąsiadów wierzchołka

   funkcja DFS(Graf G):
       dla każdego wierzchołka u z grafu G: // inicjalizacja
           kolor[u] = BIAŁY    // Biały - nieodwiedzony
           poczatek[u] = koniec[u] = 0
           rodzic[u] = NIL
       czas = 0
       dla każdego wierzchołka u z grafu G: // uruchomienie przeszukiwania
           jeżeli u jest biały:
               odwiedzaj(u)
```

## 3. Klasy potrzebne do rozwiązania zadań

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

Do dyspozycji mamy również interfejs List(**java.util.List**.) oraz jego implementacje. Wśród nich najważniejsze dla nas:

- **ArrayList -** implementacja tablicowa
- **LinkedList -** implementacja wiązana

Obie mają swoje wady i zalety jednak najprościej mówiąc klasę ArrayList ykorzystujemy w przypadku, gdy najważniejszy jest czas dostępu do danych, natomiast LinkedList, gdy wiemy, że często będą wykonywane operacje usuwania, dodawania itp. elementów gdzieś w środku struktury, a sprawą podrzędną są operacje wyszukiwania i szybkości dostępu do elementów listy.

Podstawowe metody interfejsu List (niezależne od implementacji) to:

- add(Object) – dodaje element do listy
- remove(Object) – usuwa pierwsze wystąpienie podanego obiektu z listy
- remove(int) – usuwa z listy element o wskazanym indeksie
- size() – odpowiednik własności length w przypadku tablic – zwraca rozmiar listy
- get(int) – pozwala odczytać element o wskazanym indeksie
- contains(Object) – sprawdza, czy dany element znajduje się w liście

## 4. Zadania.

1. Utworzyć własną klasę dziedziczącą z klasy LGraph. Zaimplementować metody abstrakcyjne: addVertex, addEdge, writeList, sasiedzi, check. Napisać krótki program wykorzystujący ww. metody utworzonej klasy, który zaprezentuje ich działanie. (Na razie wstawić puste ciało do pozostałych metod abstrakcyjnych) **[3p]**
2. Zaimplementować metodę transpose w klasie LGraph realizującą transpozycję grafu. **[1p]**
3. Zaimplementować metodę writeMatrix w klasie LGraph wypisującą graf w postaci macierzy sąsiedztwa. **[1p]**
4. Zaimplementować rekurencyjne przeszukiwanie DFS dla klasy LGraph (metody dfs i odwiedzaj) **[3p]**
5. Zaimplementować iteracyjne przeszukiwanie DFS, z wykorzystaniem stosu. **[2p]**

**Praca domowa:**

- Zaimplementować algorytm wyszukujący silnie spójne składowe grafu.
- Zaimplementować metodę usuwającą wierzchołek z grafu razem z wszystkimi krawędziami.

**Na następnych zajęciach:**

Silnie spójne składowe. Znajdowanie najkrótszej drogi w grafie ważonym. Algorytm Bellmana-Forda, algorytm Dijkstry, algorytm Floyda-Warshalla