Trzecia praca domowa
(Przeszukiwanie grafów skierowanych)

Należy napisać pełny projekt w Javie, który będzie zawierał program testujący algorytmy przeszukiwania grafów skierowanych wszerz i w głąb.

W szczególności program powinien zawierać własne klasy:

a) **GrafSkierowany** – abstrakcyjna klasa z abstrakcyjnymi metodami umożliwiającymi:
   - dodawanie krawędzi skierowanej do grafu w postaci 2 liczb (wierzchołek startowy i wierzchołek końcowy)
   - przeszukiwania grafu wszerz, przeszukana powinna zostać spójna część grafu, którą jesteśmy w stanie osiągnąć z wierzchołka startowego
     - parametrem wejściowym powinien być wierzchołek startowy algorytmu,
     - wyjściem lista wierzchołków w kolejności odwiedzania
   - przeszukiwania grafu w głąb, założenia jak na wykładzie - odwiedzamy wszystkie składowe spójne

Zakładamy, że graf składa się z N wierzchołków, które są numerowane kolejnymi liczbami całkowitymi od 0 do N-1 (aby prosto można było realizować reprezentacje z użyciem tablic).

b) implementacja klasy GrafSkierowany przy pomocy macierzy sąsiedztwa (bez wag, jedynie liczby 1 lub 0). Poza dostarczeniem ciała metodom wydziedziczonym należy zdefiniować:
   - konstruktor przyjmujący w parametrze liczbę naturalną – ilość wierzchołków tworzonego grafu
   - standardową metodę `toString()` konwertującą do napisu macierz sąsiedztwa

c) implemetacja klasy GrafSkierowany przy pomocy tablicy/listy list sąsiedztwa. Metody jak w klasie powyżej.

Dodatkowo należy stworzyć mechanizm tworzenia odpowiedniego grafu na podstawie pliku. Najlepiej zrobić to w oddzielnej klasie/klasach. Wejściem do metody tworzącej graf może być ścieżka do pliku zapisana tekstowo (ewentualnie klasa File). Niech taki plik ma postać:

```
<liczba_wierzchołków> [m/s]
<liczba_krawędzi>
<nr_wierzch_start_krawędzi_1> <nr_wierzch_końc_krawędzi_1>
<nr_wierzch_start_krawędzi_2> <nr_wierzch_końc_krawędzi_2>
...
```

Program testujący powinien działać następująco:

1. Ustalić ścieżkę do pliku z grafem (rozwiązanie dowolne – dialog, parametr programu, ostatecznie nawet zaszycie w kodzie)
2. Na podstawie pliku stworzyć graf odpowiedniego typu (**`m`** – reprezentacja macierzowa, **`s`** – reprezentacja listowa): użyć konstruktora oraz metody dodawania linii
3. Wyświetlić postać każdego z grafów (używając metody toString())
4. Ustalić z użytkownikiem wierzchołek startowy do przeszukiwania wszerz
5. Użyć każdego z dwóch przeszukiwań dla utworzonego grafu (wynik powinien automatycznie wypisać się na ekran)

Uwagi:

1. Wewnątrz metod przeszukiwania można wykorzystać kolejkę i stos (inna struktura dla innej metody). Można użyć typów standardowych lub własnej implementacji.
2. Lista sąsiadów jednego wierzchołka może być dowolnym z typów: kolejka, stos, lista liniowa.
3. Proszę przetestować programy na większych grafach. Najlepiej narysować grafy, których przeszukiwanie będzie łatwe do śledzenia, stworzyć odpowiednie pliki i uruchomić swoje programy na tych plikach.
   Warto sprawdzić czy program nie wpadnie w pętlę nieskończoną, gdy w grafach istnieją cykle.

Przykłady:

1. Plik postaci:

```
4 m
5
1 2
1 3
1 0
0 2
0 1
```

powinien umożliwiać produkcję grafu o macierzy sąsiedztwa:

```
0 1 1 0
1 0 1 1
0 0 0 0
0 0 0 0
```

2. Ten sam plik zaczynający się od linii

```
4 s
```

powinien umożliwiać produkcję grafu o liście sąsiedztwa (format wyświetlania może być inny, byle czytelny):

```
0: 1 -> 2
1: 0 -> 2 -> 3
2:
3:
```

3. Przeszukiwania powyższego grafu

a) wszerz od wierzchołka 0: 0 1 2 3 lub 0 2 1 3
b) wszerz od wierzchołka 2: 2
c) w głąb dla tego grafu ma wiele możliwości. Co jest istotne, nawet jeśli zaczniemy przeszukiwać od ostatnich wierzchołków (które są „izolowane" – nie mają sąsiadów), to powinniśmy odwiedzić wszystkie. Np. 3 0 1 2 (najpierw wierzchołek 3, a potem w głąb od 0)