# Bonusová domáca úloha č. 2

(Termín odovzdania úlohy: **do utorka 15. apríla 2025, 9:50**, t. j. do začiatku deviatych cvičení.)

Cieľom tejto úlohy je preprogramovať prehľadávanie grafov do hĺbky a do šírky do podoby iterátorov cez vrcholy grafu.

Archív priložený k tomuto zadaniu obsahuje balík `graphs` a v ňom triedy pre grafy z prednášky, ako aj kostry tried `DepthFirstSearchIterator` a `BreadthFirstSearchIterator`, ktoré by mali implementovať rozhranie `Iterator<Integer>`. Vašou úlohou bude dokončiť implementáciu týchto dvoch tried tak, aby poskytovali konštruktor, ktorý dostane orientovaný graf `g` a štartovací vrchol `start`, pričom:

- Trieda `DepthFirstSearchIterator` bude reprezentovať iterátor postupne vracajúci vrcholy grafu `g` dosiahnuteľné z vrcholu `start` v poradí, v akom sú objavované *prehľadávaním do hĺbky* z vrcholu `start`.
- Trieda `BreadthFirstSearchIterator` bude reprezentovať iterátor postupne vracajúci vrcholy grafu `g` dosiahnuteľné z vrcholu `start` v poradí, v akom sú objavované *prehľadávaním do šírky* z vrcholu `start`.

Objavenie vrcholu možno pri prehľadávaní do hĺbky z prednášky opísať ako volanie metódy `search` pre daný vrchol a pri prehľadávaní do šírky ako pridanie daného vrcholu do radu. Vaše iterátory by mali vrcholy grafu `g` dosiahnuteľné z vrcholu `start` vracať v tom poradí, v akom boli v práve opísanom zmysle objavované algoritmami z prednášky – samotné metódy vašich tried sa však, samozrejme, budú od tých z prednášky líšiť.
Pri oboch prehľadávaniach je teda prvým objaveným vrcholom vždy vrchol `start`. Následníci každého vrcholu `v` by sa mali spracúvať v poradí podľa výstupu volania `g.outgoingEdgesDestinations(v)`. Každý vrchol dosiahnuteľný z vrcholu `start` by mal byť iterátorom vrátený práve raz.

V obidvoch triedach implementujte:

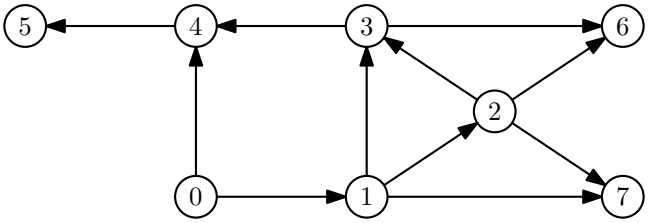
- Konštruktor s argumentmi `DirectedGraph g` a `int start` (v tomto poradí). V prípade, že je za `g` dosadená referencia `null` alebo vrchol `start` v grafe `g` neexistuje, vyhodí tento konštruktor výnimku typu `IllegalArgumentException`. V opačnom prípade sa korektne inicializuje iterátor zodpovedajúci danému prehľadávaniu grafu `g` z vrcholu `start`.
- Metódu `next`, ktorá „objaví“ a vráti na výstupe prvý ešte neobjavený vrchol grafu `g` dosiahnuteľný z vrcholu `start`. Poradie objavovania vrcholov je pre jednotlivé iterátory opísané vyššie. V prípade, že už neexistuje žiaden ďalší neobjavený vrchol, malo by mať volanie metódy `next` za následok vyhodenie výnimky typu `NoSuchElementException`.
- Metódu `hasNext`, ktorá je s metódou `next` konzistentná, t. j. vráti `true` práve vtedy, keď nasledujúce volanie metódy `next` nevyhodí výnimku.

Úlohu *neriešte* spustením celého prehľadávania hneď v konštruktore – naopak pri volaniach metódy `next` zakaždým pokračujte v už načatom prehľadávaní. Pri prehľadávaní do hĺbky sa môže zísť jeho odrekurzívnenie do podoby algoritmu využívajúceho zásobník.
Na testovači budú vaše iterátory spúšťané aj na veľmi veľkých grafoch, ktoré síce sú reprezentované inštanciami tried implementujúcich rozhranie `DirectedGraph`, avšak ich prípadná reprezentácia pomocou zoznamov následníkov alebo matíc susednosti by už zabrala príliš veľa pamäte.[1] Pri implementácii iterátorov si teda dajte pozor na to, aby bol objem dát uložený v ich inštanciách vždy najviac úmerný počtu uskutočnených volaní metódy `next` – a nie počtu vrcholov grafu. Špeciálne si teda informáciu o navštívených vrcholoch grafu nepamätajte v poli o veľkosti úmernej počtu vrcholov grafu, ale použite napríklad množinu, do ktorej budete ukladať len reálne navštívené vrcholy.

Na testovač odovzdávajte ZIP archív obsahujúci priečinok `graphs` a v ňom všetky triedy balíka `graphs` – vrátane tých z prednášky. Pri tvorbe kódu rešpektujte konvencie jazyka Java a elementárne zásady objektovo orientovaného programovania.

[1] Hrany v týchto grafoch sú namiesto toho reprezentované programami, ktoré na základe danej dvojice vrcholov vypočítajú, či medzi nimi vedie hrana.

**Príklad 1.** Uvažujme graf na nasledujúcom obrázku a inštanciu triedy `DepthFirstSearchIterator` vytvorenú pre tento graf a štartovací vrchol 0. Predpokladajme, že výstupy metódy `outgoingEdgesDestinations` vždy obsahujú následníkov jednotlivých vrcholov vo *vzostupnom* poradí.




Volania metódy `next` tohto iterátora potom budú postupne vracať vrcholy `0, 1, 2, 3, 4, 5, 6, 7`. Prípadné ďalšie volanie metódy `next` vyústi vo vyhodenie výnimky typu `NoSuchElementException`.

**Príklad 2.** Uvažujme pre rovnaký graf a štartovací vrchol iterátor typu `BreadthFirstSearchIterator`. Volania metódy `next` tohto iterátora budú postupne vracať vrcholy `0, 1, 4, 2, 3, 7, 5, 6`. Prípadné ďalšie volanie metódy `next` bude mať za následok vyhodenie výnimky typu `NoSuchElementException`.