Általános tudnivalók

Ebben az ismertetésben az osztályok, valamint a minimálisan szükséges metódusok leírásai szerepelnek. A feladat megoldása során be kell tartani a félév során megismert kódolási konvenciókat. Ennek megfelelően ügyeljünk arra, hogy a megadottakon kívül egyetlen osztály se tartalmazzon más publikus metódust vagy adattagot, illetve egyik csomag se tartalmazzon más osztályokat! Ha az implementáció megköveteli, akkor az osztályok rejtett adattagokkal és metódusokkal szabadon bővíthetők. Törekedjünk arra, hogy az osztályok belső reprezentációját a lehető legjobban védjük, tehát csak akkor engedjünk meg, és csak olyan hozzáférést, amelyre a feladat felszólít, vagy amit az osztályt használó kódrészlet megkíván!

*Figyelem!* Az a metódus, amely fordítási hibát tartalmaz, automatikusan *nulla* pontot ér!

Használható segédanyagok:

[Java dokumentáció](https://bead.inf.elte.hu/files/java/api/index.html), legfeljebb egy üres lap és toll.

A feladathoz tartozik egy [letölthető segédlet] (people.inf.elte.hu/bonnie/melleklet2.zip), amely a JUnit tesztelő jar fájljait, minta bemeneti és kimeneti fájlokat, egy adatbázist (a szavakat tartalmazó szótárt), a kijavítandó és kiegészítendő Dictionary, valamint a futtató Main osztályt tartalmazza.

Ha bármilyen kérdés, észrevétel felmerül, azt a felügyelőknek kell jelezni, *NEM* a diáktársaknak!

A beadás módja

* A megoldást egyetlen .zip állományként kell feltölteni, amely tartalmazza a csomagnak megfelelő könyvtárszerkezetben az összes forráskódot.
* A .zip állomány *NE* tartalmazzon semmilyen egyéb mappát!
* A fordítás során keletkező .class állományokat és a JUnit futtatásához használatos két .jar fájlt sem szabad mellékelni!

A feladat összefoglaló leírása

A feladat a Word ladder (Szólétra) játék szimulálása a megadott követelmények szerint. A szólétra lényegében szavak sorozata, amely egy megadott szóval kezdődik és egy megadott szóval végződik. A játék folyamán egy lépés során pontosan egy betűt szabad megváltoztatni. A létra mindegyik foka egy értelmes angol szó kell, hogy legyen. Mindegyik szó hossza ugyanakkora. Például, ha a clock szóból el szeretnénk jutni a break szóba, akkor az a következő létrán kereszül lehetséges:

[clock, crock, croak, creak, break]

A játék célja a legrövidebb ilyen létra meghatározása.

A feladat részletes ismertetése

1. rész (12 pont)

Dictionary

Hozzuk létre az utils.Dictionary osztályt, amely egy szótárt reprezentál!

Hibajavítás *(8 pont)*

* Az osztálynak egy rejtett adattagja van, amely a szótárban szereplő szavak (a szavak szöveges típusúak) halmazát reprezentálja. A halmaz rendezett (TreeSet típusú).
* Az osztálynak legyen egy nyilvános konstruktora, amely egy fájlnevet kap paraméterként. A fájl a szótárban szereplő szavakat tartalmazza. Minden sorban pontosan egy szó van. A metódus dolgozza fel a fájlt és tárolja el a szavakat az előbbi halmazban. Feltesszük, hogy a fájl nem tartalmaz hibás sorokat.

A kiadott kód nyolc darab hibát tartalmaz, mindegyik hiba megtalálása *(1 pont)*ot ér.

Eddig tart a Hibajavítás rész

* Az osztálynak legyen egy getWords nevű lekérdező metódusa, amely visszaadja a szótárban szereplő szavak halmazát. Ügyeljünk arra, hogy a belső állapotot ne szivárogtassuk ki! *(1 pont)*
* Az osztályban definiáljuk felül az Object-től örökölt equals metódust. Két Dictionary típusú objektum akkor egyenlő, ha a bennük szereplő szavak halmaza megegyezik. *(2 pont)*
* Az Object-től örökölt hashCode metódust is felüldefiniáljuk. Megköveteljük, hogy azonos szótárakra a hashCode azonos értéket adjon vissza. *(1 pont)*

2. rész (18 pont)

Graph (3 pont)

Hozzuk létre a graph.Graph generikus interfészt. Adjunk neki egy V típusparamétert (a gráf csúcsai V típusúak)!

* Az interfésznek legyen egy hasEdge nevű metódusa, amely két V típusú értéket (csúcsot) kap paraméterként. A metódus egy logikai értéket ad vissza, amely azt mondja meg, hogy a megadott két csúcs között van-e él. *(1 pont)*
* Rendelkezzen az interfész egy visszatérési érték nélküli, addNode nevű metódussal is. A metódus segítségével egy, a paraméterként kapott V típusú csúcsot tudunk a gráfhoz hozzáadni. *(1 pont)*
* Tartalmazzon a Graph interfész egy visszatérési érték nélküli, addEdge nevű metódust. Az addEdge metódus segítségével, a paraméterként kapott V típusú végpontok által megadott élt adhatjuk a gráfhoz. *(1 pont)*

UndirectedGraph (13 pont)

Hozzuk létre a graph.undirected.UndirectedGraph nevű osztályt, amely implementálja a graph.Graph generikus interfészt. Az UndirectedGraph osztály irányítatlan gráfok reprezentálására szolgál. Adjunk az UndirectedGraph osztálynak is egy V típusparamétert, amely megegyezik ősének típusparaméterével!

*Megjegyzés*: irányítatlan gráf segítségével fogjuk ábrázolni a szavak gráfját. Két szó között a gráfban van él, ha hosszuk megegyezik és pontosan egy betűben térnek el egymástól.

* A gráfot ún. szomszédsági lista segítségével ábrázoljuk. A szomszédsági listás reprezentációban a gráf minden csúcsához egy listát rendelünk. Ezen listában tartjuk nyilván az adott csúcs szomszédait. Ennek megfelelően az osztálynak legyen egy rejtett adattagja, amely a szomszédsági lista tárolására szolgál.

*Segítség*: a szomszédsági lista ábrázolása történhet hasításon alapuló adatszerkezet, pl. HashMap segítségével, amelyben a csúcsok felelnek meg a kulcsoknak, az értékek pedig az adott csúcsokkal szomszédos csúcsok listái / halmazai (feltehető, hogy a gráfban nincs többszörös él). Az adattag típusának pontos kiválasztásában a feladatot megoldó szabad kezet kap.

*Megjegyzés*: a szomszédsági lista 'szimmetrikus'. Tehát a "creak" és a "break" szomszédosak, hiszen pontosan egy betűben térnek el egymástól. Vagyis a "creak" szomszédsági listájába fel kell venni a "break" szót, de a "break" szomszédsági listájába is fel kell venni "creak" szót (a gráf irányítatlan).

* Legyen az osztálynak két további rejtett adattagja: (1) A prev, amely nyilvántartja az összes csúcsot és minden egyes csúcsra azok közvetlen megelőzőjét a kiindulási csúcstól az adott csúcsig vezető úton, vagyis csúcs-csúcs párokat. (2) A dist, amely nyilvántartja az összes csúcsot és azok távolságát a kiindulási csúcstól (vagyis csúcs-egész párokat fogunk eltárolni).

*Segítség*: A prev ábrázolása történhet hasításon alapuló rendezett adatszerkezet, pl. TreeMap segítségével, amelyben a csúcsok felelnek meg a kulcsoknak, az értékek pedig az adott csúcsokat közvetlenül megelőző csúcsoknak. A dist ábrázolására szintén alkalmazhatunk TreeMap-et, amelyben a csúcsok felelnek meg a kulcsoknak, az értékek pedig az adott csúcsoknak a kiindulási csúcstól való legrövidebb távolságát (a kiindulási csúcstól az adott csúcsig vezető legrövidebb út hosszát) adják meg.

* Az osztály konstruktora publikus, egy üres irányítatlan gráfot hoz létre.
* Az interfészspecifikációban megadott metódusok megvalósítása a következőképpen történjen:
  + hasEdge: ha a gráf nem tartalmazza a két végpontként megadott csúcsot, akkor dobjunk java.util.NoSuchElementException kivételt, "Nonexistent node." hibaüzenettel. Egyébként vizsgáljuk meg, hogy a mindkét végponthoz tartozó listában / halmazban szerepel-e a metódus paraméterként megadott (másik) végpont (mivel irányítatlan gráfról van szó, ezért ha a gráfban az i-dik és a j-dik csúcs között van él, akkor a j-dik és az i-dik között is, és fordítva). *(2 pont)*
  + addNode: ha a gráf tartalmazza a paraméterként megadott csúcsot, akkor nem történik változás. Ellenkező esetben tároljuk el a csúcsot a szomszédsági listában, és hozzunk létre a szomszédok későbbi tárolására egy üres listát. *(1 pont)*
  + addEdge: ha a gráf nem tartalmazza a paraméterként megadott végpontokat, akkor vegyük fel ezen csúcsokat a szomszédsági listába. Amennyiben valamelyik végpont szomszédai között nem szerepelt eddig a másik végpont, akkor adjuk hozzá a hiányzó végpontot az adott végpont szomszédainak listájához. *(2 pont)*
* Rendelkezzen az osztály egy nyilvános, a szélességi bejárást megvalósító, bfs nevű metódussal. A bfs metódus egy V típusú csúcsot kap paraméterként, amelyből a szélességi keresést indítjuk és egy listában a bejárás sorrendjében visszaadja a bejárt csúcsokat. *(4 pont)*

A szélességi bejárás (breadth first search, bfs) a következőképpen működik: a bfs a gráf egy pontjából (csúcsából) indul ki és veszi annak szomszédait, majd ezeknek a szomszédoknak a szomszédait, és így tovább, egészen addig, amíg el nem érjük az összes csúcsot. A kiindulási csúcsot nevezzük most s-nek (ezt kapjuk paraméterül). Tehát az algoritmus a következő lépéseket hajtja végre:

1. A bejárást az s csúcsnál kezdjük. Használjunk egy tárolósort: mindig ennek a tárolósornak a végéhez fogunk újabb gráfbeli csúcsokat hozzáadni, az elejéről pedig kivesszük a következő vizsgálandó csúcsot (az s csúcsot berakjuk a sorba, és ki is vesszük onnan).
2. Keressük meg az s csúcshoz közvetlenül kapcsolódó csúcsokat (az s csúcs szomszédait a gráfban)! Rakjuk ezeket a csúcsokat a sor végére!
3. Vegyük ki a sorból az éppen a sor elején lévő csúcsot! Soroljuk fel ennek a csúcsnak a szomszédait és ezek közül a szomszédok közül válasszuk ki azokat, amelyekben még nem jártunk! Rakjuk be ezeket az új csúcsokat a sor végére!
4. Ha a tárolósor nem üres, akkor ismételjük meg a 3. lépést, ellenkező esetben az algoritmus befejeződik, mert minden csúcsot bejártunk.

Példa: Definiáljuk a G=(V,E) irányítatlan gráfot, ahol V jelöli a csúcsok, E az élek halmazát a következőképpen. V={v1,v2,v3,v4,v5,v6,v7,v8,v9}, E={e12,e13,e14,e56,e54,e76,e78,e79}, hol eij jelöli azt, hogy a gráfban az i. és j. csúcs között irányítatlan él van. Tegyük fel, hogy a szélességi bejárást a 4. (v4) csúcsból indítjuk. Ekkor a fenti algoritmus a következőképpen hajtódik végre: elsőként berakjuk a v4 csúcsot a sorba, majd megkeressük v4 szomszédait, v1-et és v5-öt, és ezeket a sor végére rakjuk, v4-et pedig kivesszük a sorból. A következő lépésben v1-gyel folytatjuk, és berakjuk a sor végére v1 még meg nem látogatott szomszédait, vagyis v2-t és v3-at, v1-et pedig kivesszük a sorból. Ezután a sor elején álló v5 következik, először a még meg nem látogatott szomszédját, vagyis v6-ot rakjuk a sorba, v5-öt pedig kivesszük a sorból. Ezután a sor elején álló v2-t, majd v3-at vesszük ki a sorból egymás után, mert ezen csúcsoknak nincs további meg nem látogatott szomszédja. Majd v6 következik, berakjuk a sorba v7-et, a meg nem látogatott szomszédját, v6-ot pedig kivesszük. A következő lépésben v7-tel folytatjuk és berakjuk a sor végére v7 még meg nem látogatott szomszédait, vagyis v8-at és v9-et, v7-et pedig kivesszük a sorból. Mivel v8-nak és v9-nek nincsenek meg nem látogatott szomszédai, ezért ezeket a csúcsokat egymás után kivesszük a sorból és ezzel a bejárás befejeződik. Tehát a csúcsokat a következő sorrendben látogatjuk meg: [v4,v1,v5,v2,v3,v6,v7,v8,v9].

*Segítség*: a metódusban a sort láncolt listával is ábrázolhatjuk (LinkedList) és használhatjuk ezen osztály [poll()] metódusát is, amely visszaadja és eltávolítja a lista első elemét (fejét).

A szélességi bejárás során, mikor az egyes csúcsokat a sorba rakjuk, akkor aktualizálni kell a prev és dist adatszerkezeteket is. Vagyis, ha a sorba rakott csúcs még nem szerepel a nyilvántartásban, akkor a prev-be berakjuk a csúcsot (mint kulcsot) és annak megelőzőjét (mint értéket), a dist-be pedig ugyanazon csúcsot (mint kulcsot) és a kiindulási csúcstól való távolságát (a megelőző csúcs kiindulási csúcstól való távolsága+1, feltesszük, hogy az élek hossza egységnyi). Figyeljünk arra, hogy a kiindulási csúcsnak nincsen megelőzője, önmagától vett távolsága pedig 0!

* Továbbá legyen az osztálynak egy nyilvános hasPathTo nevű metódusa, amely egy V típusú csúcsot kap paraméterként és egy logikai értékkel tér vissza, amely megadja, hogy létezik-e út az aktuális (a kiindulási) csúcs és a paraméterként megadott csúcs között. *(1 pont)*

*Segítség*: a metódusban felhasználhatjuk a dist TreeMap-et, amelyben a paraméterként megadott csúcsnak mint kulcsnak kell szerepelnie.

* Legyen az osztálynak egy nyilvános distTo nevű metódusa is, amely egy V típusú csúcsot kap paraméterként és egy egész számmal tér vissza, amely megadja az adott csúcsnak a kiindulási csúcstól való távolságát. Ha nincs út a két csúcs között, akkor a távolság legyen végtelen (Integer.MAX\_VALUE) *(1 pont)*
* Definiáljuk a nyilvános pathTo metódust is, amely egy V típusú csúcsot kap paraméterként és csúcsok listájával tér vissza. A csúcslista a kiindulási csúcs és a paraméterként megadott csúcs közötti utat tartalmazza. Ha ilyen út nincs, akkor a metódus null-t adjon vissza. *(2 pont)*

DisconnectedException *(2 pont)*

Hozzunk létre egy saját kivételosztályt, a utils.DisconnectedException osztályt. Ez a kivétel jelzi, ha nincs út két csúcs között. A kivétel származzon az Exception kivételosztályból. Legyen egy paramétert nem váró konstruktora és hívja meg az ősének üzenetszöveget váró konstruktorát. A megadott üzenet legyen a következő: "Not connected!".

3. rész -- Tesztelés (10 pont)

Az UndirectedGraph osztályhoz készüljön JUnit alapú tesztelő, test.UndirectedGraphTest osztályban.

A tesztelésnek alaposnak kell lennie, a tesztek megalkotásakor gondoljunk az alábbi esetekre:

* null paraméter (gráf),
* üres gráf,
* nem létező / létező élt tartalmazó gráf,
* nem létező / létező csúcsot tartalmazó gráf,
* nincs út két csúcs között,
* két csúcs távolsága 0, pozitív vagy végtelen.

Emlékeztető: a JUnit futtatása, ha a tesztesetek osztálya a névtelen csomagba tartozó SimpleTest:

javac -cp .:junit-4.12.jar:hamcrest-core-1.3.jar SimpleTest.java

java -cp .:junit-4.12.jar:hamcrest-core-1.3.jar org.junit.runner.JUnitCore SimpleTest

4. rész (7 pont)

WordLadder (7 pont)

Hozzuk létre a ladder.WordLadder osztályt, amely a játék szimulációját hajtja végre!

* Legyen az osztálynak egy osztályszintű, rejtett isNeighbour nevű metódusa, amely két szöveges típusú adatot kap paraméterként (két szót) és egy logikai értéket ad vissza. A két szónak egyforma hosszúnak kell lennie. A metódus igazat ad vissza, ha a két szó pontosan egy betűben tér el egymástól, egyébként hamisat. *(2 pont)*
* Definiáljuk az osztályszintű, rejtett readInput metódust is, amely egy fájlnevet kap paraméterként és egy listában azt a két szót adja vissza, amelyekkel a szólétra kezdődik és végződik. Ezt a két szót az inputfájlból olvassuk be. Dobjunk RuntimeException kivételt, ha az alábbi esetek valamelyike fordul elő: *(2 pont)*

a.) az inputfájl nem két szót tartalmaz ("The number of words in the input file in a line differs from two." hibaüzenettel), b.) a szavak hossza nem egyforma ("Words have different lengths." hibaüzenettel) és c.) az inputfájl nem egy sorból áll ("The input file does not consist of one line." hibaüzenettel).

* Hozzunk létre egy ugyancsak osztályszintű és rejtett, visszatérési érték nélküli writePath nevű metódust is, amely szöveges típusú paraméterként kapja a fájl nevét és a fájlba írandó szöveget. *(1 pont)*
* Legyen az osztálynak egy osztályszintű, nyilvános buildLadder nevű metódusa, amely három szöveges típusú paramétert kap: az inputfájlt, az adatbázist (a szavakat) tartalmazó szótárt, valamint az outputfájl nevét. A metódus a szótár alapján létrehozza a gráfot. *(2 pont)*

Majd megnézi, hogy az inputfájlban szereplő két szó benne van-e a szótárban. Ha valamelyik nincsen, akkor RuntimeException kivételt dob. Egyébként végrehajtja a szélességi keresést a gráfon. Ha a két szó között van út, akkor kiírjuk az út hosszát, valamint magát az utat a következőképpen (l. az output\_expected\_3.txt fájlt):

"The length of the path is 18. [white, whine, chine, chink, clink, clank, flank, flask, flash, slash, slosh, sloth, sooth, south, sough, rough, rouge, rouse, house]"

Ha nincs út a két szó között, akkor a "DISCONNECTED" szó kerüljön az outputfájlba.

Minden más esetben, amikor kivétel váltódik ki, írassuk ki az outputfájlba a kivétel típusát és a hibaüzenetet, amelyet kapunk (l. a Wordladder readInput és buildLadder metódusait) a következőképpen (l. az output\_expected\_4.txt fájlt):

"class java.lang.RuntimeException doggy is not in the dictionary."

*Segítség*: az elkapott kivétel típusát a getClass(), a hibaüzenetet a getMessage() metódussal kérdezhetjük le.

A Main futtatása

A main.Main fájl tartalmazza a főprogramot, amely három parancssori argumentumot vár:

* a két szót tartalmazó fájl nevét (a szólétra kezdetét és végét),
* a szótárt tartalmazó fájl nevét és
* a kimeneti fájl nevét.

$ javac main/Main.java

$ java main.Main words\_1.txt dictionary.txt output\_1.txt

Ekkor az output\_1.txt tartalmát kell az output\_expected\_1.txt-vel összehasonlítani.

Pontozás

* 0 -- 14: elégtelen (1)
* 15 -- 22: elégséges (2)
* 23 -- 30: közepes (3)
* 31 -- 38: jó (4)
* 39 -- 47: jeles (5)

Jó munkát! :-)