Általános tudnivalók

Ebben az ismertetésben az osztályok, valamint a minimálisan szükséges metódusok leírásai szerepelnek. A feladatmegoldás során fontos betartani az elnevezésekre és típusokra vonatkozó megszorításokat, illetve a szövegek formázási szabályait. Segédfüggvények is létrehozhatók, a feladatban nem megkötött adattagok és elnevezéseik is a feladat megoldójára vannak bízva. 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!

A beadott megoldásnak működnie kell a mellékelt tesztprogrammal, de ez nem elégséges feltétele az elfogadásnak. Törekedjünk arra, hogy a megírt forráskód kellően általános és újrafelhasználható legyen!

Az egyes feladatrészeknél két pontszám látható (pl. 5 + 1), ebből az első az alappont, mely a helyességért, a feladatleírásban szereplő követelmények teljesítéséért adható. A második egy pluszpont, mely az oktató megítélése alapján akkor jár, ha a hallgató az adott feladatrészt a félévben tanult alapvető programozási irányelvek betartásával készítette el, így levonandó például optimalizálatlan, kódismétlésekkel teli vagy konvenciókat erősen sértő kód esetén.

Használható segédanyagok: [Java dokumentáció](https://bead.inf.elte.hu/files/java/api/index.html), legfeljebb egy üres lap és toll. Ha bármilyen kérdés, észrevétel felmerül, azt a felügyelőknek kell jelezni, *NEM* a diáktársaknak!

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

Tesztelés

Az egyes részfeladatokhoz tartoznak külön tesztesetek, amelyeket a feladatok végén jelöltünk meg. Ezek önállóan is fordítható és futtatható .java állományok a mellékelt .jarsegítségével. Például Windows alatt az első feladathoz tartozó tesztesetek így fordíthatók és futtathatók:

> javac -cp .;tests-Bacon.jar tests/Part1.java

> java -cp .;tests-Bacon.jar tests/Part1

Ugyanezeket a teszteseteket használja a komplett feladathoz tartozó tesztelést végző Test osztály is. Ezt Windows alatt így lehet futtatni:

> java -cp .;tests-Bacon.jar Test

Linux alatt mindent ugyanúgy lehet fordítani és futtatni, csak a -cp paraméterében a pontosvesszőt kell kettőspontra cserélni.

A feladat összefoglaló leírása

A feladat a Hat lépés távolság (Six Degrees of Separation) című játék szimulálása a megadott követelmények szerint. A játékot filmek és a filmekben szereplő színészek gráfján fogjuk játszani. A cél annak kiszámítása, hogy egy adott színész hány lépés távolságra van a Kevin Bacon nevű színésztől (Six Degrees of Kevin Bacon - az elnevezés onnan ered, hogy ez a távolság általában 6, vagy annál kisebb egész szám). A távolságot az ún. Bacon-szám segítségével adjuk meg. A Bacon-számot a következőképpen definiáljuk: Kevin Bacon Bacon-száma 0; azoknak a színészeknek, akik Kevin Baconnel szerepeltek közös filmben, a Bacon-száma 1; azoknak a színészeknek, akik szerepeltek közös filmben azon színészekkel, akikkel Kevin Bacon játszott együtt, a Bacon-száma 2, stb. (Ha a gráfban egy színészhez több út vezet Kevin Bacontól, akkor mindig a legrövidebbet tekintjük.)

A programhoz tartozik egy [egységtesztelő](https://bead.inf.elte.hu/files/java/Bacon.zip), amely az egyes osztályok funkcionalitását teszteli.

A feladat részletes ismertetése

1. rész (12 + 2 pont)

Entity (1 pont)

Hozzuk létre az entities.Entity absztrakt osztályt, amely a játékban résztvevő entitásoknak a közös ősosztálya!

* Az osztálynak egy védett adattagja van, a szöveges típusú name, amely az entitás nevét (címét) reprezentálja és létrehozás után nem változik.
* Az osztálynak legyen egy nyilvános konstruktora, amely paraméterként megkapja az entitás nevét, és beállítja a megfelelő adattagot!
* Az osztálynak egy lekérdező metódusa van, amelynek segítségével az entitás nevét tudjuk lekérdezni.

Actor (4 pont)

Hozzuk létre az entities.Actor osztályt, amely az entities.Entity osztály egyik leszármazottja és egy színészt reprezentál!

* Az osztálynak legyen egy nyilvános konstruktora, amely paraméterként megkapja a színész nevét! A konstruktor meghívja a szülőosztály konstruktorát, és beállítja a megfelelő adattagot.
* Az osztályban definiáljuk felül az Object-től örökölt equals metódust. Két Actor típusú objektum akkor egyenlő, ha a nevük megegyezik (a name értéke). *(1 pont)*
* Az Object-től örökölt hashCode metódust is felüldefiniáljuk. Megköveteljük, hogy azonos színészekre a hashCode azonos értéket adjon vissza. *(1 pont)*
* Az Actor osztályban felüldefiniáljuk az Object-től örökölt toString metódust. A felüldefiniált toString metódus az alábbi szöveget adja vissza: "[name]", ahol [name] a színész nevét jelöli. Pl. "Kevin Bacon" *(1 pont)*
* Az Actor osztálynak meg kell valósítania a Comparable<Actor> interfészt, hogy a színészek rendezhetőek legyenek. A színészeket a nevük szerint rendezzük. *(1 pont)*

Genre (1 pont)

A utils.Genre felsorolási típus segítségével a különböző filmfajtákat reprezentáljuk.

* Vegyük fel a filmfajtákat (ACTION, ADVENTURE, COMEDY, CRIME, DRAMA) tartalmazó felsorolási típust!

Movie (6 pont)

Hozzuk létre az entities.Movie osztályt, amely az entities.Entity osztály egy másik leszármazottja és egy filmet reprezentál!

* Az osztálynak két rejtett adattagja van: (1) egy Genre felsorolási típusú genre, amely az adott film fajtáját, és (2) egy egész típusú year, amely a film elkészítésének évét adja meg. Létrehozás után egyik adattag sem változik.
* Az osztálynak legyen egy nyilvános konstruktora, amely paraméterként megkapja a film címét (nevét), a fajtáját és az elkészítésének évét! A konstruktor beállítja a megfelelő adattagokat.
* Definiáljuk felül az Object-től örökölt equals metódust. Két Movie típusú objektum akkor egyenlő, ha a name, a genre és a year értékeik rendre megegyeznek. *(1 pont)*
* Az Object-től örökölt hashCode metódust is felüldefiniáljuk. Megköveteljük, hogy azonos filmekre a hashCode azonos értéket adjon vissza. *(2 pont)*
* A Movie osztályban felüldefiniáljuk az Object-től örökölt toString metódust. A felüldefiniált toString metódus az alábbi szöveget adja vissza: "[name] ([genre],[year])", ahol [name] a film címét, [genre] a film fajtáját, [year] pedig az elkészítésének évét jelöli. Pl. "Animal House (COMEDY, 1978)" *(1 pont)*
* A Movie osztálynak meg kell valósítania a Comparable<Movie> interfészt, hogy a filmek rendezhetőek legyenek. Elsődlegesen a filmek címe, másodlagosan a fajtája, harmadlagosan pedig az elkészítési éve szerint rendezünk. Pl. ("Titanic",DRAMA,1997) = ("Titanic",DRAMA,1997); ("Titanic",DRAMA,1997) > ("Titanic",COMEDY,1997); ("Titanic",DRAMA,2000) > ("Titanic",DRAMA,1997). *(2 pont)*

Tesztelő: tests.Part1

2. rész (14 + 2 pont)

Graph (2 pont)

Hozzuk létre a graph.Graph generikus interfészt. Adjunk neki egy V típusparamétert (a gráf csúcsai V típusúak), azzal a kikötéssel, hogy a paraméter csak Entity típusú lehet!

* 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.
* 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.
* 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.

UndirectedGraph (12 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! Itt is meg kell ismételnünk a Vtípusparaméterre vonatkozó kikötést.

*Megjegyzés*: irányítatlan gráf segítségével fogjuk ábrázolni a színész-film gráfot. A gráf csúcsai színész vagy film típusúak lehetnek. Egy színész és egy film között a gráfban van él, ha az adott színész szerepelt a megadott filmben.

* 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 ha "Kevin Bacon" szerepelt az "Animal House" c. filmben, akkor "Kevin Bacon" szomszédsági listájába fel kell venni az "Animal House" c. filmet, de az "Animal House" c. film szomszédsági listájába is fel kell venni "Kevin Bacon"-t (a gráf irányítatlan).

* Legyen az osztálynak két további rejtett adattagja: (1) A prev, amely nyilvántartja az összes csúcsot (pontosabban azok szöveges reprezentációját, String típusú adatok) és minden egyes csúcsra azok közvetlen megelőzőjét (pontosabban azok szöveges reprezentációját, String típusú adatok) a gyökértől a csúcsig vezető úton, vagyis String-String párokat. (2) A dist, amely nyilvántartja az összes csúcsot és azok távolságát a gyökértől. A csúcsoknak ismét a szöveges reprezentációját használjuk (vagyis String-Integer 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 gyökértől való legrövidebb távolságát (a gyökértől az adott csúcsig vezető legrövidebb út hosszát) adják meg. A szöveges reprezentáció előállítására majd alkalmazható a toString metódus.

* 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 visszatérési érték nélküli, bfs nevű metódussal. A bfs metódus egy V típusú csúcsot kap paraméterként, amelyből a szélességi keresést indítjuk. *(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 kiinduló 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}, ahol 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-gyel 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 (annak szöveges reprezentációja) 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 gyökértől való távolságát (a megelőző csúcs gyökértől való távolsága+1, feltesszük, hogy az élek hossza egységnyi). Figyeljünk arra, hogy a gyökérnek nincsen megelőzője, önmagától vett távolsága pedig 0!

* Továbbá legyen az osztálynak egy distTo nevű metódusa, amely egy V típusú csúcsot kap paraméterként és egy egész számmal tér vissza, amely megadja az adott csúcsnak a gyökértől való távolságát. *(1 pont)*
* Definiáljuk a pathTo metódust is, amely egy V típusú csúcsot kap paraméterként és szöveges típusú adattal tér vissza. A szöveg megadja, hogy egy adott színész milyen messze van egy másiktól (a gyökértől, jelen esetben Kevin Bacontól), valamint az adott színészig vezető útvonalat is, a következő formában: *(2 pont)*

"Nicole Kidman's number is 2.Bacon->Animal House (COMEDY, 1978)->Donald Sutherland->Cold Mountain (ADVENTURE, 2003)->Nicole Kidman"

*Megjegyzés*: Kevin Bacon az Animal House c. filmben szerepelt együtt Donald Sutherlanddel, aki viszont a Cold Mountain c. filmben játszott együtt Nicole Kidmannel. Kevin Bacon Bacon-száma 0, Donald Sutherlandé 1, Nicole Kidmané 2.

"Kevin Bacon's number is 0.Bacon"

*Megjegyzés*: a pathTo esetében nem a tényleges távolságát kell visszaadni a gráfban a csúcsnak és a kiinduló csúcsnak, hanem annak a felét, hiszen egy "lépés" két él a gráfban: egy a színésztől a filmig és egy másik a filmtől a másik színészig.

Tesztelő: tests.Part2

3. rész (5 + 1 pont)

Bacon (5 pont)

Hozzuk létre a bacon.Bacon osztályt, amely a játék szimulációját hajtja végre!

* Az osztálynak két rejtett adattagja van: (1) egy UndirectedGraph típusú g, amely a színész-film gráfot adja meg, valamint egy osztályszintű, Actor típusú konstans, amely a "Kevin Bacon" nevű színészt reprezentálja.
* Az osztálynak legyen egy nyilvános konstruktora, amely egy fájlnevet kap paraméterként. A fájl a szimulációban szereplő filmeket és színészeket tartalmazza. A metódus dolgozza fel a fájlt és tárolja el a filmeket és azok szomszédait (a filmekben szereplő színészeket) az irányítatlan gráfban. Az inputfájl minden sora egy filmet tartalmaz filmjellemző@szomszédok listája formában, ahol filmjellemző megadja a film címét, fajtáját és elkészítésének évét (vesszővel elválasztva), a szomszédok listája pedig a filmben szereplő színészeket (vesszővel elválasztva). Pl. a fájlban előfordulhat a következő sor: Animal House,COMEDY,1978@John Belushi,Karen Allen,Donald Sutherland,Kevin Bacon. Feltesszük, hogy minden sor a megadott szerkezetű, a fájl nem tartalmaz hibás sorokat. *(4 pont)*
* Definiáljuk az osztályban a simulate metódust, amely egy Actor típusú értéket kap paraméterként és szöveges típusú adattal tér vissza. A szimuláció során szélességi kereséssel bejárjuk a gráfot "Kevin Bacon" nevű színészből indítva, és megadjuk a paraméterként megadott színészig vezető utat (ehhez alkalmazzuk az előzőekben definiált pathTo metódust). *(1 pont)*

Tesztelő: tests.Part3

A tesztelő által adott pontszám csak becslésnek tekinthető, a gyakorlatvezető levonhat pontokat, vagy adhat részpontokat.

Pontozás

* 0 -- 8: elégtelen (1)
* 9 -- 15: elégséges (2)
* 16 -- 22: közepes (3)
* 23 -- 29: jó (4)
* 30 -- 36: jeles (5)

Jó munkát! :-)