Hibajegyzék

2017. 03. 15.

* Kiemelve, hogy melyek a nyilvános metódusok és adattagok, melyek nem.
* A tesztelő hibát jelez, ha az adattagok nem rejtettek.

2017. 03. 17.

* Pótolva a Packet.getData() hiányzó leírása.

2017. 03. 22.

* A tesztelő hibát jelez, ha egy Node objektum az IPAddress objektumok egyenlőségének vizsgálatára nem az isTheSame() metódust használja.
* Új teszteset hozzádva a route()-hoz, a traceRoute()-nak eggyel hosszabb útvonalat kell felderítenie.

Számítógépes hálózat szimulációja

Ebben a feladatban egy számítógépes hálózat szimulációjának egyszerűsített implementációját kell elkészítenünk. A részfeladatok megoldása során ügyeljünk arra, hogy a megadottakon kívül egyetlen osztály se tartalmazzon más publikus metódust vagy adattagot! A megoldás minden osztályát tegyük a router csomagba!

Tesztelés

A feladathoz tartozik egy [letölthető segédlet](http://people.inf.elte.hu/poor_a/hu/java/router/Router.zip), amelyben megtaláljuk a feladat tesztelőjét.

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

> javac -cp .;tests-Router.jar tests/IPAddressTest.java

> java -cp .;tests-Router.jar tests/IPAddressTest

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

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

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

A hálózatban levő számítógépek címeinek ábrázolása

A hálózaton összekapcsolt gépeket ún. IP-címekkel fogjuk azonosítani. Az IP-címeknek többféle szabványa is létezik, ezek közül mi most az IPv4-et fogjuk választani, ahol a címek négy, pontokkal elválaszott, 00 és 255255 közti egész számból állnak, például 127.0.0.1, 8.8.4.4 vagy 255.255.255.255.

Ezeket az IP-címeket reprezentáljuk egy külön IPAddress osztállyal! Az osztály tartalmazzon:

* egy konstruktort, amely paraméterként kap egy egészekből álló tömböt, amely tartalmazza az IP-cím egyes komponenseit, és ennek elemeit másoljuk is le.

A paraméterül kapott nem feltétlenül négy elemű. Ha rövidebb, egészítsük ki a végén 0-kal. Ha hosszabb, akkor csak az első négy számot tároljuk el.

* egy osztályszintű fromString() metódust, amely egy, csak az IP-címek szöveges leírásának megfelelő, például "12.100.32.76" String értékből állít elő egy IPAddressobjektumot. (Segítségül: figyeljünk arra, hogy ha reguláris kifejezéseket használunk a megoldásban, akkor pontra illeszkedő reguláris kifejezés a "\\.", míg a "." egy tetszőleges karakterre illeszkedő kifejezés.) Ha a paraméter nem bontható fel négy komponensre vagy négynél több komponensből áll, úgy a metódus adjon vissza nullreferenciát.
* egy logikai értékkel visszatérő isTheSame() metódust, amely eldönti, hogy a paramétereként átadott, IP-címet ábrázoló objektum megegyezik az objektumban tárolt IP-címmel.
* egy logikai értékkel visszatérő insideRng() függvényt, amely eldönti, hogy az adott IP-cím a paraméterként megadott másik két cím közé esik-e. Akkor mondjuk, hogy két cím közé esik egy harmadik, ha lexikografikus rendezés szerint az elsőtől nagyobb vagy egyenlő és a másodiktól kisebb vagy egyenlő.

Példák:

- 127.0.0.0 - 127.0.0.255 tartományon belül: 127.0.0.1

- 127.0.0.0 - 127.0.0.255 tartományon kívül: 10.0.0.1

- 0.0.0.1 - 63.127.127.127 tartományon belül: 23.0.0.0

- 0.0.0.1 - 63.127.127.127 tartományon kívül: 64.0.0.2

* egy String visszatérési értékű toString() metódust, amely formázott szövegként jeleníti meg az adott IP-címet. Ez formátum a korábbiak szerint négy, ponttal elválasztott számot fog jelenteni, például: "12.100.32.76".

A hálózaton utazó csomagok ábrázolása

A hálózaton utazó csomagokat reprezentáljuk a Packet osztály egy objektumával! Egy ilyen Packet típusú objektum tartalmazza a csomag által továbbított szöveges adatot, a küldő IP-címét, a csomag célpontjának IP-címét, a csomag által meglátogatható csomópontok felső korlátját (Time to Live, TTL)! A felső korlátra azért lesz szükségünk, hogy a csomag, ha esetleg véletlenül hurokba kerülne, ne keringjen a végtelenségig a hálózaton. Ezt minden csomópont csökkenteni fogja eggyel.

A meglátogatható csomópontok alapértelmezett felső korlátját tároljuk el egy osztályszintű INIT\_TTL konstansban, értéke legyen 50.

Egy csomagnak megfelelő Packet objektum létrehozása kétféle konstruktorral történhessen:

* egy konstruktorral, amely paraméterként megkapja az általa továbbítandó szöveges adatot, a meglátogatható csomópontok felső korlátját, a küldő és a célpont IP-címét,
* egy konstruktorral, amely paraméterként megkapja az általa továbbítandó szöveges adatot, a küldő és a célpont IP-címét. A meglátogatható csomópontok felső korláta legyen az alapértelmezett INIT\_TTL.

A csomagokat leíró osztály tartalmazzon:

* egy getData() metódust, amely a csomag által hordozott adatot segít lekérdezni,
* egy getDestination() metódust, amely a csomag célpontját segít lekérdezni,
* egy getSource() metódust, amely a csomag küldőjét segít lekérdezni,
* egy decreaseTTL() metódust, mely a csomag által meglátogatható csomópontok számát csökkenti eggyel,
* egy logikai értékkel visszatérő isLive() metódust, amely el tudja dönteni, hogy az adott csomag továbbküldhető még, vagy már nem tehet meg több ugrást. A csomag továbbküldhető még, ha a csomópontok felső korlátja pozitív,
* egy toString() metódust, amely egy String értékben szövegesen visszaadja a cél IP-címét és a hordozott üzenetet a következő formátumban:

küldő cím -> cél cím , üzenet

Például:

10.0.0.5 -> 192.168.0.3 , ping

Buffer

A beérkező, feldolgozásra váró csomagokat egy buffer adatszerkezetben tároljuk. Egy buffert egy csomagokból álló tömbbel ábrázolunk, ahol számon tartjuk az első csomag indexét és a tárolt csomagok számát. A bufferból az első csomagot tudjuk kivenni és a buffer végére tudunk csomagot betenni.

Az osztály tartalmazzon:

* egy konstruktort, mely megkapja a buffer kapacitását, azaz az eltárolható csomagok számát,
* egy isEmpty() metódust, mely logikai értéket ad eredményül: igazat, ha üres a buffer, különben hamisat.
* egy isFull() metódust, mely logikai értéket ad eredményül: igazat, ha nem fér több csomag a bufferbe, különben hamisat.
* egy append() metódust, mely egy csomagot vár paraméterül, és nincs visszatérési értéke. A metódus a buffer végére szúrja be a paraméterül kapott csomagot, ha a buffer még nincs tele.

Vigyázat: az új elem indexe akár kisebb is lehet, mint a buffer első elemének indexe. A tömböt itt körkörösen töltjük föl, az utolsó index után a nulladik következik.

* egy removeFirst() metódust, mely eltávolítja az első elemet a bufferből és visszaadja azt eredményül, ha a buffer nem üres. Ha üres, az eredmény null.

Vigyázat: itt frissíteni kell az első elem indexét és a hosszt is. Az utolsó index után a nulladik következik.

A hálózat csomópontjainak ábrázolása

A számítógépes hálózat egy csomópontját a Node osztály egy példánya fogja reprezentálni. A hálózaton minden egyes csomópont ismeri a szomszédos csomópontjait, továbbá minden csomópont nyilvántartja a saját IP-címét és annak az IP-tartománynak a határait, amiért felelős. Minden csomópontnak maximum 4 csatlakozója van, ahova a szomszédokhoz vezető kábeleket illeszthetjük, azaz 4 szomszédot ismerhet. Ha egy csatlakozóban nincs kábel, akkor azon a helyen null referencia szerepel, különben a szomszédos Node objektum. Emellett egy csomópontnak van egy buffere, ahova a beérkezett, feldolgozatlan csomagok kerülnek. A buffert úgy hozzuk létre, hogy legalább 10 csomag férjen el benne.

Az osztály tartalmazzon:

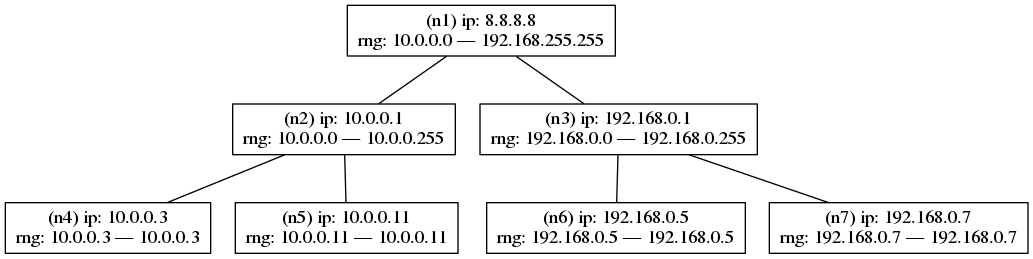
* egy konstruktort, mely paraméterül várja a saját IP-címét, és a felelősségére bízott IP-címtartomány alsó és felső határát,
* egy osztályszintű connect() metódust, mely egy kábel csatlakoztatását végzi el. A metódus paraméterül vár egy Node típusú node1-et, ahhoz tartozó egész típusú node2index indexet, egy másik Node típusú node2-őt és az ahhoz tartozó egész típusú node1index indexet. A connect() összekapcsolja a két csomópontot a hálózatban azáltal, hogy a node1 csomópont node2index indexű szomszédjának beállítja node2-t, és fordítva, ha az indexek érvényesek (nem túl nagyok és nem negatívak). Ha nem érvényesek, a metódus nem tesz semmit. Az indexelés 0-val kezdődik.
* egy getPackets() metódust, mely a bufferbe érkező összes csomagot sortörés karakterrel elválasztva egy String értékben adja vissza.

Csomagok szimulált továbbítása a hálózaton

A szimuláció során egy csomagot úgy fogunk továbbítani, hogy a csomópontokat leíró Node osztályt kiegészítjük még egy route() nevű metódussal. Ez a metódus egy csomagot kap paraméterként és nincs visszatérési értéke A metódus működése legyen az alábbi:

* Ellenőrizzük, hogy a csomag céljának címe megegyezik-e a csomópont saját címével.
  + Ha a csomópont címe egyezik a csomag célpontjával, akkor a csomag célbaért, ekkor a csomagot tegyük a bufferbe.
  + Ha nem egyezik meg, akkor nézzük meg, hogy él-e még a csomag.
    - Ha él még, akkor keressünk olyan szomszédot, amely azért a IP-címtartományért felelős, amelybe a csomag céljának címe tartozik. Feltételezhetjük, hogy ezek a tartományok diszjunktak.
      * Ha találtunk, akkor csökkentsük csomag által meglátogatható csomópontokat és a továbbítsuk az adott rákövetkezőnek a csomagot.
      * Ha nem találtunk, akkor küldjünk vissza a csomag küldőjének egy csomagot "Destination is unreachable" adattal. A visszaküldött csomagnál a küldő az aktuális csomópont, nem az eredeti cél csomópont.
    - Ha nem él már a csomag, akkor küldjünk vissza a csomag küldőjének egy csomagot "Time is exceeded" adattal. A visszaküldött csomagnál a küldő az aktuális csomópont, nem az eredeti cél csomópont.

Például:



Ebben a gráfban az alábbi lekérdezések a következő eredményeket adják:

// ...

Node n4 = new Node(IPAddress.fromString("10.0.0.3"),

IPAddress.fromString("10.0.0.3"),

IPAddress.fromString("10.0.0.3"));

// ...

Packet p1 = new Packet("alma",

4,

IPAddress.fromString("10.0.0.3"),

IPAddress.fromString("192.168.0.5"));

n4.route(p1);

// n6 bufferének tartalma: 10.0.0.3 -> 192.168.0.5 , alma

Packet p2 = new Packet("alma",

4,

IPAddress.fromString("10.0.0.3"),

IPAddress.fromString("1.1.1.1"));

n4.route(p2);

// n4 bufferének tartalma: 10.0.0.1 -> 10.0.0.3 , "Destination is unreachable"

Packet p3 = new Packet("alma",

4,

IPAddress.fromString("10.0.0.3"),

IPAddress.fromString("10.0.0.2"));

n4.route(p3);

// n4 bufferének tartalma: 8.8.8.8 -> 10.0.0.3 , "Time is exceeded"

Packet p4 = new Packet("alma",

5,

IPAddress.fromString("10.0.0.3"),

IPAddress.fromString("10.0.0.2"));

n4.route(p4);

// n4 bufferének tartalma: 10.0.0.1 -> 10.0.0.3 , "Time is exceeded"

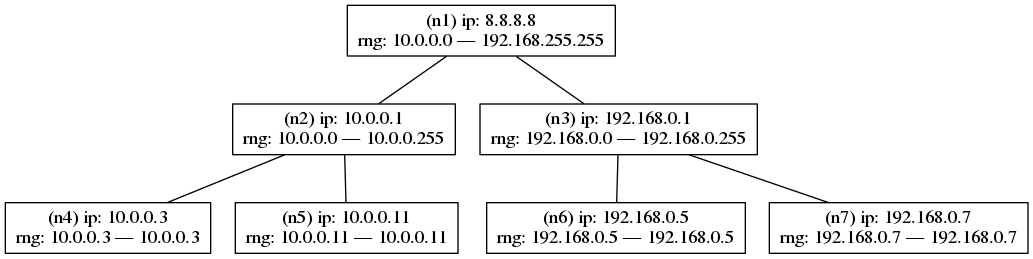
Csomagok útjának nyomon követése a hálózatban

Egy csomag útját úgy fogjuk nyomon követni, hogy a csomópontokat leíró Node osztályt kiegészítjük még egy traceRoute() nevű metódussal. Ez a metódus egy IP-címet, a cél címét kapja paraméterként és a visszatérési érték egy String objektum.

A metódus 1-től kezdve egyre nagyobb felső korláttal rendelkező csomagot küld el a cél felé, amíg egy csomag el nem éri a cél csomópontot. Az elküldendő adat tetszőleges, lehet például "traceroute". A nyomkövetés úgy történik, hogy a csomag életének lejártakor a csomópont, amelyhez eljuttott a csomag visszaküld egy "Time is exceeded" csomagot. Ennek a "Time is exceeded" csomagnak a küldője rajta van az eredeti célhoz vezető úton, ezért ezt letároljuk. A következő csomagnál eggyel megnöveljük a csomag által meglátogatható csomópontok felső korlátját, és ezt is elküldjük az eredeti (paraméterként kapott) célhoz. Ezt addig ismételjük, amíg nem kapunk választ az elküldött csomagra, mert az eredeti cél nem fog "Time is exceeded" csomagot visszaküldeni.

Feltesszük, hogy az eredeti cél megtalálható a hálózatban, tehát a visszakapott csomagok mind "Time is exceeded" tartalmúak, ezért ezt nem kell ellenőrizni.

A végeredmény egy String, mely vesszővel elválasztva sorolja fel a célig vezető csomópontok IP-címét.



Például, maradva az előző hálózatunknál:

// ...

Node n4 = new Node(IPAddress.fromString("10.0.0.3"),

IPAddress.fromString("10.0.0.3"),

IPAddress.fromString("10.0.0.3"));

// ...

System.out.println(n4.traceRoute(IPAddress.fromString("192.168.0.1")));

// "10.0.0.3 , 10.0.0.1 , 8.8.8.8 , 192.168.0.1"

Jó munkát!