El?z? - Fel - Következ?

Tartalomjegyzék

- 1 Generics
  - 1.1 Feladat
  - 1.2 Wildcards
- 2 Collections
  - 2.1 List
    - 2.1.1 ArrayList
    - 2.1.2 LinkedList
  - 2.2 Set
    - 2.2.1 HashSet
    - 2.2.2 TreeSet
  - 2.3 Map
    - 2.3.1 HashMap
    - 2.3.2 TreeMap
- 3 Feladat
  - 3.1 Naív
  - 3.2 Map
  - 3.3 Generic

Generics

tutorial

C++ **template** megfelel?je.

Tetsz?leges metódust vagy osztályt megírhatunk úgy, hogy m?ködjön több osztályra is. Például:

```
public class Main
{
    public static <T> boolean IsIn(T[] a, T v)
    {
        for( int i = 0; i < a.length; ++i)
        {
             if (a[i].equals(v))
                 return true;
        }
        return false;
    }
}
```

Írhatunk generikus osztályt is:

```
public class A <T>
```

```
{
    . . .
}
```

Feladat

Írjunk generikus `Ring` interface-t, ami a három alapm?veletet definiálja. Ezt implementálja a `Real` és `Int` osztályok.

Ezután megírhatjuk a generikus `Complex` osztályt, amit így példányosíthatunk:

```
Complex<Real>
Complex<Int>
```

```
public interface  Ring<T> {
public T add(T other);
public T mul(T other);
public T sub(T other);
public T clone();
}
```

Wildcards

Collections

Kollekciókban lehet objektumokat tárolni.

List

Sorrendet ?rz? kollekció. Lehet beszúrni, kitörölni elemet, tetsz?leges pozícióban. Le lehet kérdezni az adott pozícióban lév? elemet.

ArrayList

A List-nek olyan megvalósítása, ahol az elemek egy összefügg? memóriatartományban (tömb) helyezkednek el.

- Id?ben **konstans** költsége van egy elem elérésének
- Id?ben **lineáris** egy elem beszúrása vagy törlése, mert az összes többi elemet is mozgatni kell hozzá.

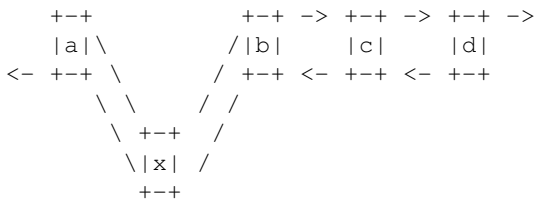
```
|a|b|c|d|e|f|g| -> |a|b|x|c|d|e|f|g|
    ^
    x
```

LinkedList

Láncolt lista, az elemek helye nem rögzített a mamóriában, de mindegyik eltárol egy-egy referenciát ez el?tte és a mögötte álló elemre, így ?rzi a sorrendet.

```
   +-+ -> +-+ -> +-+ -> +-+ ->
   |a|    |b|    |c|    |d|
<- +-+ <- +-+ <- +-+ <- +-+
```

- Egy adott pozíció elérése id?ben **lineáris**, mert az elejét?l lépkedve lehet eljutni minden elemhez.
- Egy elem beszúrása id?ben **konstans**, mert csak a szomszédos elemeket érinti a beszúrás



Set

A Set olyan tároló, amiben egyedi elemek vannak, nem lehet benne kétszer ugyan az (halmaz). De cserébe nem beszélhetek adott index? elemr?l, mert az elemet sorrendje nem kötött. Egy elem beszúrása mind az el?tte, mind az utána lév?ket átrendezheti. S?t nem is lehet egy adott pozícióba beszúrni elemet, csak *beszúrni* lehet, ha már benne volt az elem, akkor nem történik semmi, ha nem, akkor benne lesz, de nem tudjuk, hogy hol.

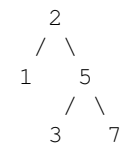
HashSet

Hash függvény segítségével határozza meg, hogy hova kerüljön egy elem. Az elemek sorrendje látszólag véletlen.

- Egy elem beszúrása, megkeresése és törlése is id?ben **várhatóan konstans**!
  - de legrosszabb esetben lineáris!
  - ez azt jelenti, hogy általában gyorsak a fent említett m?veletek, de néha lehet lassú, várható értékben még mindig nagyon gyors.

TreeSet

Elemeit rendezett fában tárolja. Beszúrás/törlés hatására a sorrend garantáltan nem romlik el. A lenti példában az [1,2,3,5,7] elemeket tároljuk.



A fa kiegyensúlyozott, ami azt jelenti, hogy nem engedi, hogy bármelyik ága is túl hosszú legyen. Széles és sekély fára törekszik az implementáció.

Ennek következménye, hogy:

- a beszúrás, keresés, törlés id?ben **logaritmikus**

Map

A Set-hez hasonló, csak még értékeket is tudunk az elemekhez tárolni, hasonlít a **python dict**-hez.

```
["http://wiki.math.bme.huanna"http://wiki.math.bme.hu: 555, "http://wiki.math.bme.hubeatrice"http:
```

A kulcsok azok, helyek a Set-hez hasonlóan vannak tárolva, az értékek egyértelm?ek egy adott kulcshoz, de lehet két kulcsnak ugyan az az értéke.

HashMap

Lásd #HashSet, csak Map.

TreeMap

Lásd #TreeSet, csak Map.

Feladat

Implementáljunk egy **gráf** osztályt!

- A gráf csúcsait jelöljék egész számok
- két egész szám jelöljön egy irányított élet a két csúcs között
- Tudjunk csúcsot hozzáadni és törölni
    - **void add(Integer a)**
    - **void remove(Integer a)**
- Tudjunk élet hozzáadni és törölni
    - **void add(Integer a, Integer b)**
    - **void remove(Integer a, Integer b)**
- Írjuk meg a **toString** metódust, hogy lássunk is valamit a gráfból.
- Legyenek **GetV** és **GetE** metódusok, melyekkel a csúcsok és élek számát tudjuk lekérdezni.

Naív

Kezdetben a csúcsokat tároljuk listában, az éleket pedig egy list-of-list adatstruktúrában, ami kett? hosszú listák listája.

```
private List<Integer> nodes_;
private List<List<Integer>> edges_;
```

Ha így teszünk, mennyi az egyes metódusok m?veletigénye? Konstans, lineáris, logaritmikus?

Map

Az éleket hatékonyabb egy adott csúcshoz eltárolni: legyen minden csúcshoz egy lista, ami a bel?le kiinduló éleket tárolja.

```
private Map<Integer, List<Integer>> edges_;
```

Ekkor mennyi az egyes metódusok m?veletigénye?

Generic

Változtassuk meg az osztályt úgy, hogy a csúcsokat tetsz?leges objektum jelölje, például Integer vagy String.

```
public class Graph <Type>
{
   . . .
```

```
}
```

El?z? - Fel - Következ?