Midterm 2015 Meno a priezvisko: _____
obsahuje 5 príkladov, spolu 31 bodov skupina: _____

1) [7 bodov] Zistite, čo počíta nasledujúca funkcia **foo** pre **a>=0 a b=0**. Hint: **foo(10,0) = 2**.

```
static int foo(int a, int b) {
   if (a == 0)
      return b;
   else
      return foo(a/2, b + a%2);
}
```

[1 bod] Pre istotu to vyjadrite pár slovami, a že tomu rozumiete, tak vypočítajte **foo(35,0)** = ?

[2 body] Napíšte nerekurzívnu definíciu funkcie **foo** tak, aby počítala to isté, teda:

```
static int foo(int a, int b) {          // píšte rovno sem...
```

```
}
```

Zistite, čo počíta nasledujúca funkcia **goo** pre **a>=0**. Hint: **goo(10) = 2, goo(15) = 0**.

```
static int goo(int a) {
   if (a == 0)
      return 0;
   else
      return (~a&1) + goo(a>>1);
}
```

[2 body] Vyjadrite pár slovami, a že tomu rozumiete, tak vypočítajte **goo(35)** = ?

[2 body] Nájdite najväčšie 3-ciferné číslo **x** také, že **goo(x) = foo(x,0)**:

Hint:

```
2^0  =     1,           1
2^1  =     2,          10
2^2  =     4,         100
2^3  =     8,        1000
2^4  =    16,       10000
2^5  =    32,      100000
2^6  =    64,     1000000
2^7  =   128,    10000000
2^8  =   256,   100000000
2^9  =   512,  1000000000
.......  999   1111100111
2^10  = 1024, 10000000000
```

————— riešenia príkladu 1 píšte na tento list (aj z druhej strany) alebo si vypýtajte ďalší list

2) [7 bodov] Tieto dve triedy popisujú známu implementáciu binárneho vyhľadávacieho stromu:

```
public class BVSNode<E extends Comparable<E>> {
    BVSNode<E> left;
    E key;
    BVSNode<E> right;
}
public class BVSTree<E extends Comparable<E>>  {
    BVSNode<E> root;
}
```

Pomocou týchto tried vieme vytvoriť objekt typu `BVSTree<E>`, ktorý predstavuje binárny vyhľadávací strom, t.j. hodnoty v ľavom podstome sú <= ako vrchol a hodnoty v pravom podstrome sú > vrchol, a to platí pre každý vrchol stromu určený referenciou `root`.

[1 bod] V triede `BVSTree` (prípadne aj `BVSNode`) definujte metódu `E findMax()`, ktorá nájde maximálny prvok, ak strom spĺňa uvedenú podmienku vyhľadávacieho stromu. Ak žiaden nie je maximálny, vráti null.

[3 body] V triede `BVSTree` (prípadne aj `BVSNode`) definujte metódu `int depth(E k)`, ktorá nájde hĺbku vrchola s hodnotu `k` oproti koreňu stromu. Hĺbka samotného koreňa je 0, hĺbka jeho synov 1,... Ak sa vrchol s hodnotou `k` nenachádza v strome, výsledkom funkcie je -1.

Predpokladajte, že máte utriedený zoznam rôznych prvkov v štruktúre typu `ArrayList<E>`, z ktorých idete skonštruovať vyvážený binárny vyhľadávací strom nasledujúcim spôsobom:

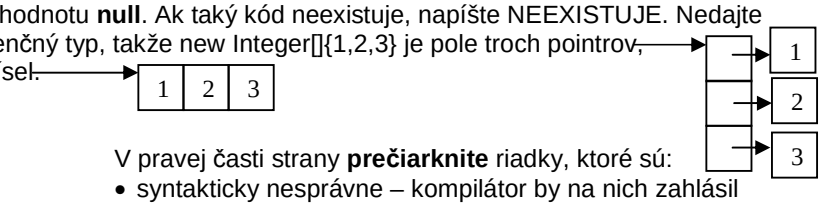
- zo zoznamu vyberiete median (prostredný), a urobíte z neho koreň stromu,
- z menších ako median rekurzívne vytvoríte ľavý podstrom, a z väčších ako median pravý podstrom koreňa.

Medián je prostredný prvok, nie aritmetický priemer ! Ak sú prostredné dva, lebo pár, tak medián je ktorýkoľvek z nich.

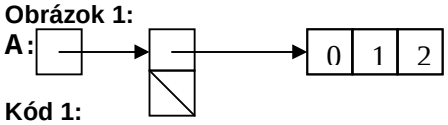
[3 body] V triede `BVSNode` definujte metódu `BVSNode<E> makeTree(List<E> zoz)` vytvorí vyvážený binárny vyhľadávací strom z usporiadaného zoznamu `zoz` (že je ozaj usporiadaný nemusíte testovať).

——————— riešenia príkladu 2 píšte na tento list alebo si vypýtajte ďalší list

3) [5 bodov = 0.5 bodu za každú správnu odpoveď] Ku každému obrázku napíšte kód, ktorý vytvorí znázornené pole v pamäti. Prečiarknutá bunka prestavuje hodnotu **null**. Ak taký kód neexistuje, napíšte NEEXISTUJE. Nedajte sa nachytať ! Uvedomte si, že Integer je referenčný typ, takže new Integer[]{1,2,3} je pole troch pointrov, ale new int[]{1,2,3} je pole troch 32-bitových čísel.

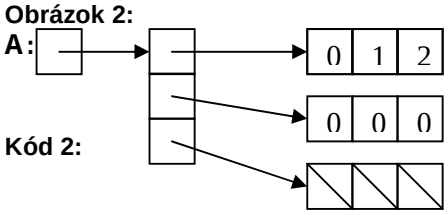


Obrázok 1:



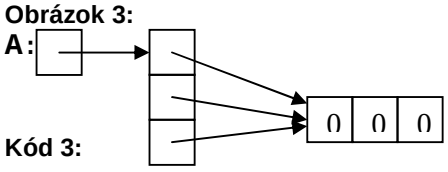
Kód 1:

Obrázok 2:



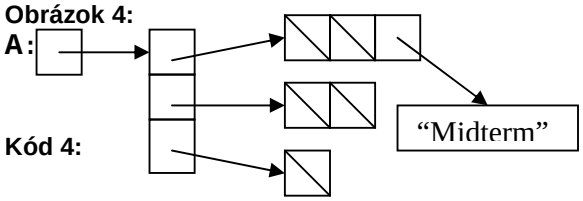
Kód 2:

Obrázok 3:



Kód 3:

Obrázok 4:



Kód 4:

V pravej časti strany **prečiarknite** riadky, ktoré sú:

- syntakticky nesprávne – kompilátor by na nich zahlásil chybu (tie **prečiarknite** a označte S),
- spôsobia chybu počas behu programu (tie označte R).

```
//Kód 5 -------- pole s null-mi
Integer[] pole1 =
       new Integer[]{1,2,null,null};
for(Integer p:pole1) {
  System.out.println(p+p);
}

// Kód 6 ------- priradenie poľa
Integer[] pole3 = { 0 };
Integer[] pole4 = pole3;

pole4 = new Integer[]{1,2};
System.out.println(pole3[1]);

pole3 = pole4;
pole4[1] = 99;

if (pole3[1] == 99)
  System.out.println(pole3[1]);


// Kód 7 ---------- dvojrozmerné pole
Integer[][] pole2 = new Integer[3][];
pole2[0] = new Integer[]{1,2,3};
pole2[2] = new Integer[]{4,5,6};
```

nakreslite obrázky pamäte pre tieto kódy:

```
// Kód 8 --------
int [][] A = new int[4][];
A[1] = new int[4];
A[3] = new int[2];


// Kód 9 --------
int [][] A = new int[4][3];

// Kód 10 --------
int [] A = {1,1,1,1};
int [] B = {2,2,2};
System.arraycopy(A,0,B,1,2);
System.arraycopy(B,0,A,0,3);
```

Help: arraycopy(Object src, int srcPos, Object dest, int destPos, int length) kopíruje od srcPos po srcPos+length-1 z poľa src na destPos až destPos+length-1 v poli dest.

———— riešenia príkladu 3 píšte na tento list, aj zozadu, alebo si vypýtajte ďalší list

4) [5 bodov] Nasledujúci kód sa spustí s argumentami predstavujúcimi jednosmerné cestovné poriadky:

```
java Cities BA-ZA-KE BA-NR-BB-KE PO-SK-BJ NR-ZA
```

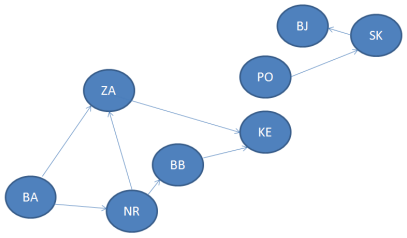
```
public class Cities {
static TreeMap<String, TreeSet<String>> cudo=new TreeMap<String, TreeSet<String>>();

public static void main(String[] args) {
   for (String arg : args) {
      String[] edges = arg.split("-");
      String vertex = null;
      for (String v : edges) {
         if (vertex != null) {
            TreeSet<String> set = cudo.get(vertex);
            if (set == null)
                 set = new TreeSet<String>();
            set.add(v);
            cudo.put(vertex, set);
         }
         vertex = v;
      }
   }
System.out.println(cudo);
```

výpis premennej *cudo* vypíše podľa Vás **[označte správnu možnosť, 2 body]**:

1. {BA=NR,BB=KE,BJ=SK,KE=BB,NR=BA,PO=SK,SK=BJ,ZA=BA}
2. {PO=[SK],SK=[BJ],ZA=[KE],BA=[ZA,NR],BB=[KE],NR=[ZA,BB]}
3. {BA=[ZA,NR],BB=[KE],NR=[ZA,BB],PO=[SK],SK=[BJ],ZA=[KE]}
4. {0=[NR,ZA],1=[KE,NR],2=[SK],3=[BB,ZA],4=[BA,BB],5=[SK],6=[BJ,PO],7=[BA,KE]}
5. {BA=[NR,ZA],BB=[KE],NR=[BB,ZA],PO=[SK],SK=[BJ],ZA=[KE]}
6. {BA=[NR,ZA],BB=[KE,NR],BJ=[SK],KE=[BB,ZA],NR=[BA,BB],PO=[SK],SK=[BJ,PO],ZA=[BA,KE]}
7. {BB=[NR],BJ=[SK],KE=[BB,ZA],NR=[BA],SK=[PO],ZA=[BA]}
8. niečo ine:

Napíšte kus kódu, ktorý vychádzajúc z hodnoty premennej cudo z predošlého kódu nájde a vypíše všetky mestá, **odkiaľ nevedie žiaden spoj**, v uvedenom príklade sú to KE a BJ. **[3 body]**



———— riešenia príkladu 4 píšte na tento list, aj zozadu, alebo si vypýtajte ďalší list

5) [7 bodov] Multi-množina je sťaby množina, v ktorej sa môžu opakovať prvky viackrát. Nezáleží na ich poradí. Kým v množine sa prvok nachádza alebo nie, v multi-množine sa nachádza n-krát (n>0), alebo nenachádza. Navrhnite vašu implementáciu multi-množiny v triede `MultiSet` implementujúcej metódy:

```
public void add(E elem);          // pridanie prvku do multi-množiny [2 body]
public int count(E elem);         // počet výskytov prvku v multi-množine [1 bod]
public boolean subset(MultiSet<E> ms);// ms je obsiahnutá v multi-množine this [2body]
public List<E> toList();//zoznam všetkých prvkov multi-množiny aj s opakovaním [2body]
```

Kód, ktorý musí zbehnúť:

```
MultiSet<Integer> ms1 = new MultiSet<Integer>();
ms1.add(1);ms1.add(3);ms1.add(2);ms1.add(1);ms1.add(3);ms1.add(4);
System.out.println(ms1.toList()); // vypíše [1, 1, 2, 3, 3, 4] v ľubovoľnom poradí
for(int i=0; i<5; i++)
  System.out.println(i+" je tam "+ms1.count(i)+".krat");
MultiSet<Integer> ms2 = new MultiSet<Integer>();
ms2.add(1); ms2.add(3); ms2.add(2); // ms2 = { 1,2,3 }
System.out.println(ms1.subset(ms2)); // true, {1,2,3} je podmnožina {1,1,2,3,3,4}
ms2.add(2);     // ms2 = { 1,2,3,2 }
System.out.println(ms1.subset(ms2));//false,{1,2,2,3} nie je podmnožina {1,1,2,3,3,4}
```

———— riešenia príkladu 5 píšte na tento list, aj zozadu, alebo si vypýtajte ďalší list