Midterm 2017 Meno a priezvisko: _____

obsahuje 5 príkladov, spolu 6+6+4+6+5 = 27 bodov+bonusy

1) [6 bodov] Dve triedy popisujú implementáciu binárneho stromu s hodnotami typu Integer vo vnútorných vrcholoch

```
public class Node {
    Node left;      // ľavý podstrom
    Integer key;    // hodnota vrchola
    Node right;   } // pravý podstrom, konštruktor: public Node(Node left, Integer k, Node right)
public class Tree {//                  konštruktor: public Tree(Node root)
    Node root;    }
```

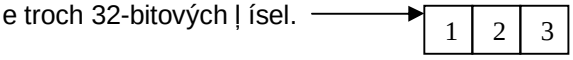
[2 body] V triede **Tree** (prípadne aj **Node**) definujte metódu **public boolean isBVS()**, ktorá vráti True, ak strom spĺňa podmienku vyhľadávacieho stromu **[+1 bod za efektívne riešenie,nelineárne riešenie max.2 body]**

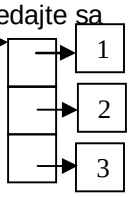
[2 body] V triede **Tree** definujte metódu **public boolean String path(Integer k)**,ktorá vráti reťazec obsahujúci písmená **L** a **R**, ktoré vás dovedú v binárnom vyhľadávacom strome od koreňa stromu k vrcholu s hodnotou **k**. **L** znamená „choň do ľavého podstromu“ **R a** analogicky „do pravého“ Hodnota **path(Integer k)**, pre koreň stromu je prázdny reťazec, teda **""**. Hodnota **path(Integer k)**, ak **k** sa nenachádza v strome je **null**, pozor **null != ""**.

[2 body] V Triede **Tree** definujte konštruktor **public Tree(int from, int to)**, ktorý vytvorí binárny vyhľadávací strom obsahujúci vo vrcholoch stromu **všetky prirodzené čísla z intervalu [from,to], teda [from..to]** a každú z nich práve raz. Nie je dôležité ako strom vyzerá, ale musí spĺňať podmienku **isBVS()** a obsahovať všetky čísla práve raz. **[Bonus od cvičiaceho: ak predsa len zostrojíte „pekný“ vyvážený strom +1bod].**

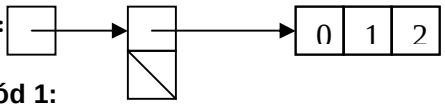
ïïïïïïïïïï riešenia príkladu 1 píšte na tento list alebo si vypýtajte ďalší list

2) [6 bodov = 0.5 bodu za kaġdú správnu odpoveĎ] Ku kaġdému obrázku napíġte kód, ktorý vytvorí znázornené pole v pamäti. Preļ iarknutá bunka prestavuje hodnotu **null**. Ak taký kód neexistuje, napíġte NEEXISTUJE. Nedajte sa nachytaŠ! Uvedomte si, ġe Integer je referenļ ný typ, takġe new Integer[]{1,2,3} je pole troch pointrov, ale new int[]{1,2,3} je pole troch 32-bitových ļ ísel.



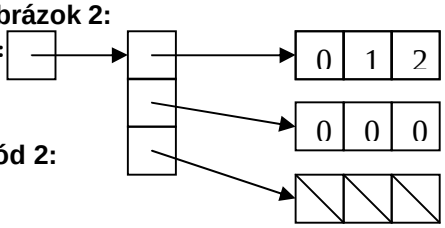


Obrázok 1:



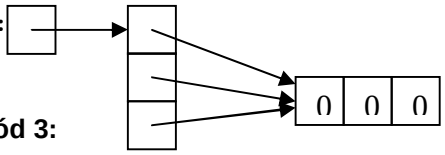
Kód 1:

Obrázok 2:



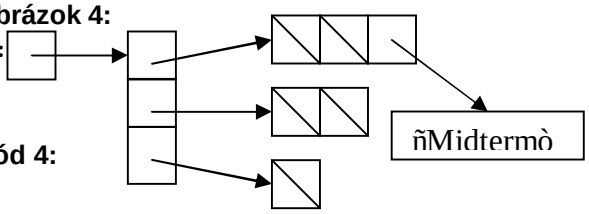
Kód 2:

Obrázok 3:



Kód 3:

Obrázok 4:



Kód 4:

Kód 5: dvojrozmerný clone

```
Integer[][]p ={{0,1},{2,3},null,{4,5}};
Integer[][] clon =
            (Integer[][])p.clone();
p[0][0] = 4;
```

ļ o vypíġe následujúci print ?

```
System.out.println(16/ clon[0][0]);
```

Kód 6:nakreslite obrázky pamäte pre premenné A..C

```
String [][] A = new String[4][];
A[0] = new String[4];
A[2] = new String[2];
String[][] B = (String [][])A;

Integer[][] C = new Integer[3][];
C[0] = new Integer[]{1,2,3};
C[2] = new Integer[]{4,5,6};
```

V pravej ļ asti strany **preļ iarknite** riadky, ktoré sú:

- syntakticky nesprávne ï kompilátor by na nich zahlásil chybu (tie **preļ iarknite** a oznaļ te S),
- spôsobia chybu poļ as behu programu (tie oznaļ te R).

Kód 7: pole s null-mi

```
Integer[] pole1 =
        new Integer[]{1,2,null,null};
for(Integer p:pole1) {
  System.out.println(p+p);
}
```

Kód 8: priradenie poĎa

```
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 9: podtrieda HashSet

```
class MySet extends HashSet<String> {}

MySet ms = new HashSet<String>();
HashSet<String> hs = new MySet();
```

Kód 10: pole podtried

```
MySet[] pole5 = new HashSet<String>[5];
HashSet<String>[] pole6 = new MySet[6];
```

Kód 11: list podtried

```
ArrayList<MySet> al1 =
        new ArrayList<HashSet<String>>();
ArrayList<HashSet<String>> al2 =
        new ArrayList<MySet>();
```

Kód 12: prepíġte kód bez pouġitia (%, /)

```
int a = ......;
if (a % 2 > 0) a++;
int b = (a / 16) % 16;
```

ï ï ï ï ï ï ï rieġenia príkladu 2 píġte na tento list, aj zozadu, alebo si vypýtajte Ņalġí list

3) [4 body] 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) {                                  // sem pride graf…
          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);
```

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

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. nieľ o ine:

[2 body] Napíšte kus kódu, ktorý nájde v štruktúre ***cudo*** všetky mestá, odkiaľ nevedie žiaden spoj, v uvedenom príklade sú to KE a BJ.

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

4) [6 bodov] Tento príklad je o vzťahoch medzi osobami, zjednodušene osoba X môže *lajkovať* osobu Y z dôvodu Z. Vzťahy nie sú symetrické ani reflexívne (samoľúbosť). Okrem toho, X môže *lajkovať* Y z viacerých dôvodov Z_1-Z_2...-Z_n. Osoby v zadaní sú vymyslené a skutoční dôvody radšej nahradené prirodzenými číslami. Príklad vstupu do programu je nižšie na strane. Každý argument má tvar X-Y{-i}$^+$arg, kde X,Y sú reťazce a -i je niekoľko (nenula) čísel dôvodov, prečo X *lajkuje* osobu Y. Štruktúra, v ktorej reprezentujeme tento svet je nasledujúca:
`TreeMap<String, TreeMap<String, TreeSet<Integer>>> likes = new … ();`
Kľúč prvého zobrazenia (*mapy*) je **`kto`**, kľúč druhého zobrazenia je **koho** a množina dôvodov, pre ktoré **`kto`** *lajkuje* **koho**.

[2 body] V konštruktore vytvorte zo vstupu štruktúru `likes`. Pre účely, čísla programu rozbíjajúca argumenty je známa, dopíšte len telo cyklu:

```
public Like(String[] args) { // konštruktor
  likes = new TreeMap<String, TreeMap<String, TreeSet<Integer>>>();
  for (String arg : args) {
    String[] parts = arg.split("-");
    String kto = parts[0];
    String koho = parts[1];
    for (int i = 2; i < parts.length; i++) {
      String preco = parts[i];
      // dopíšte kód, ktorý doplní do likes informáciu, že kto lajkuje koho z dôvodu prečo









    }
  }
}
```

Po zavolaní konštruktora musí byť hodnota `likes` nasledujúca:

```
{Bohumila={Charlotte=[1, 2], Xaver=[2]}, Colette={Chalotte=[2]}, Matilda={Bohumila=[1]}, Paula={Rexana=[1, 3],
Sara=[1, 2, 3]}, Rexana={Saxana=[1, 2]}, Sara={Xaver=[1, 3]}, Xaver={Matilda=[1, 3], Rexana=[2]}}
```

[2 body] Keď máte vyrobenú štruktúru `likes` (resp. 1.podúlohu ste preskočili), definujte metódu **`public void unfriended()`**, ktorá vypíše všetky osoby, ktorých nikto *nelajkuje*. Na poradí nezáleží. V príklade je to `Colette` a `Paula`.

[2 body] Štruktúra `likes` vlastne reprezentuje multigraf, kde vrcholmi sú osoby, a viacnásobné hrany medzi vrcholmi sú označené dôvodom *lajkovania*. V tejto podúlohe ideme otočiť orientované šípky v tomto grafe. Definujte metódu **`public TreeMap<String, TreeMap<String, TreeSet<Integer>>> liked()`**, ktorá vytvorí a vráti ako výsledok štruktúru inverznú k `likes`, teda kľúč prvého zobrazenia je **koho**, kľúč druhého zobrazenia je **kto** a množina dôvodov, pre ktoré **`kto`** *lajkuje* **koho**. V našom príklade jej obsah bude:

```
{Bohumila={Matilda=[1]}, Chalotte={Colette=[2]}, Charlotte={Bohumila=[1, 2]}, Matilda={Xaver=[1, 3]}, Rexana={Paula=[1,
3], Xaver=[2]}, Sara={Paula=[1, 2, 3]}, Saxana={Rexana=[1, 2]}, Xaver={Bohumila=[2], Sara=[1, 3]}}
```

Ilustračný príklad vstupu:

```
Xaver-Matilda-1-3
Paula-Sara-2-3
Sara-Xaver-1-3
Xaver-Rexana-2
Rexana-Saxana-2-1
Bohumila-Xaver-2-2
Bohumila-Charlotte-2-1
Colette-Chalotte-2
Matilda-Bohumila-1
Paula-Rexana-1-3
Paula-Sara-1
```

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

5) [5 bodov] V celom príklade predpokladajte, že pracujeme len s nezápornými číslami, lebo nie je dobré na midterme zisťovať, či vie študent predstaviť si binárnom zápise záporných čísel. Toto je krátka rekurzívna funkcia, ktorá má na vstupe dve nezáporné čísla:

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

Máte zistiť, čo počíta. Pre overenie hypotéz, pár overených výsledkov: **foo(5,6)=53, foo(6,5)=46, foo(5,5)=63**.
[1 bod] Zistite, čo daná funkcia počíta pre kladné vstupné hodnoty **int a,b>0** a vyjadrite to niekoľkými (<17) slovami. Pre istotu, vypočítajte tiež **foo(7,31)=?** Výsledky ďalších podúloh môžete riešiť aj keď vás toto ešte nenapadlo...

[1 bod] Je zrejmé, že **foo(0,b)=b.** Dokážte/zdôvodnite napr. indukciou, že aj **foo(a,0)=a**, pre ľubovoľné kladné **a**.
Hint: pre tých, čo sa boja indukcie: a môže byť párne alebo nepárne...

[2 body] Definujte vlastnú nerekurzívnu verziu funkcie **foo** a nazvite ju **myfoo:**
Hint: dole je tabuľka...

[1-2 body] Funkcia **goo** počíta to isté? Teda platí pre ľubovoľné a, b nezáporne, že **foo(a,b) =?= goo(a,b)** ?
Vyberte si a označte jednu z možností, ak zle zvolíte, ste bez bodov:

- **[1 bod]** áno, bez zdôvodnenia...
- **[1 bod]** nie, bez zdôvodnenia...
- **[2 body]** nie, ale treba kontrapríklad, teda **a =____ b = ____**, také, že **foo(a,b) != goo(a,b).**

```
public static int goo(int a, int b) {
	if ((a | b) == 0)
		return 0;
	else if (a == 0)
		return goo(b, a);
	else
		return (goo(a >> 1, b) << 1) + (a&1);
}
```

Hint:

i	Integer.toBinaryString(i)
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010
19	10011
20	10100

s	Integer.parseInt(s,2)
"0"	0
"1"	1
"10"	2
"11"	3
"110101"	53

ďalšie riešenia príkladu 5 píšte na tento list (aj z druhej strany) alebo si vypýtajte ďalší list