Midterm 2014 Meno a priezvisko: ____________________
obsahuje 5 príkladov, spolu 31>25 bodov skupina: ______

**1) [8 bodov]** Zistite, čo počíta nasledujúca funkcia **foo** pre **n>=0**. Hint: **foo(1000) = 1**.

```
static long foo(long n) {
      return rekurzia(n, 0);
}
static long rekurzia(long n, long r) {
      if (n == 0)
            return r;
      else
            return rekurzia(n / 10, 10 * r + n % 10);
}
```

**[1 bod]** Pre istotu to vyjadrite pár slovami, a že tomu rozumiete, tak vypočítajte **foo(911775039)** = ?

**[1 bod]** Nájdite najväčšie 3-ciferné číslo také, že **foo(x) == x**:

**[2 body]** Napíšte nerekurzívnu definíciu pre funkciu **rekurzia**, aby počítala to isté, teda:

```
static long nerekurzia(long n, long r) {       // píšte rovno sem...




}
```

Zistite, čo počíta nasledujúca funkcia **goo** pre **n>=0**. Hint: **goo(13) = 11, goo(11) = 13**.

```
static long goo(long n) {
      long vysledok = 0;
      while (n != 0) {
            vysledok <<= 1;
            vysledok +=  n & 1 ;
            n >>= 1;
      }
      return vysledok;
}
```

**[2 body]** Vyjadrite pár slovami, a že tomu rozumiete, tak vypočítajte **goo(31)** = ?

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

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
2^10  = 1024, 10000000000
```

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

**2) [6 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 však nespĺňa podmienku vyhľadávacieho stromu, t.j. že hodnoty v ľavom podstome sú <= ako vrchol a hodnoty v pravom podstrome sú > vrchol. To musí platiť pre každý vrchol. V triede BVSTree (prípadne aj BVSNode) definujte metódu boolean isBVS(), ktorá platí, ak strom spĺňa uvedenú podmienku vyhľadávacieho stromu. **[3 body za efektívne riešenie, inak max. 2 body]**

AVL strom je taký binárny vyhľadávací strom, v ktorom hĺbky ľavého a pravého podstromu ľubovoľného vrchola sa líšia najviac o 1. Definujte v triede BVSTree metódu boolean isAVL(), ktorá zistí, či strom je AVL stromom. **[3 body, ak prejdete celý strom len raz, inak max. 2 body]**

—————— 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 priraďte číslo obrázku, ktorý graficky vystihuje situáciu v pamäti po vykonaní série príkazov. V prípade, že nastane chyba, napíšte CHYBA. Prečiarknutá bunka prestavuje hodnotu **null**. Ak taký obrázok neexistuje, napíšte NEEXISTUJE. Nedajte sa nachytať ! `zdroj: http://pages.cs.wisc.edu/~hasti/cs368/JavaTutorial/NOTES/Java_vs.html`

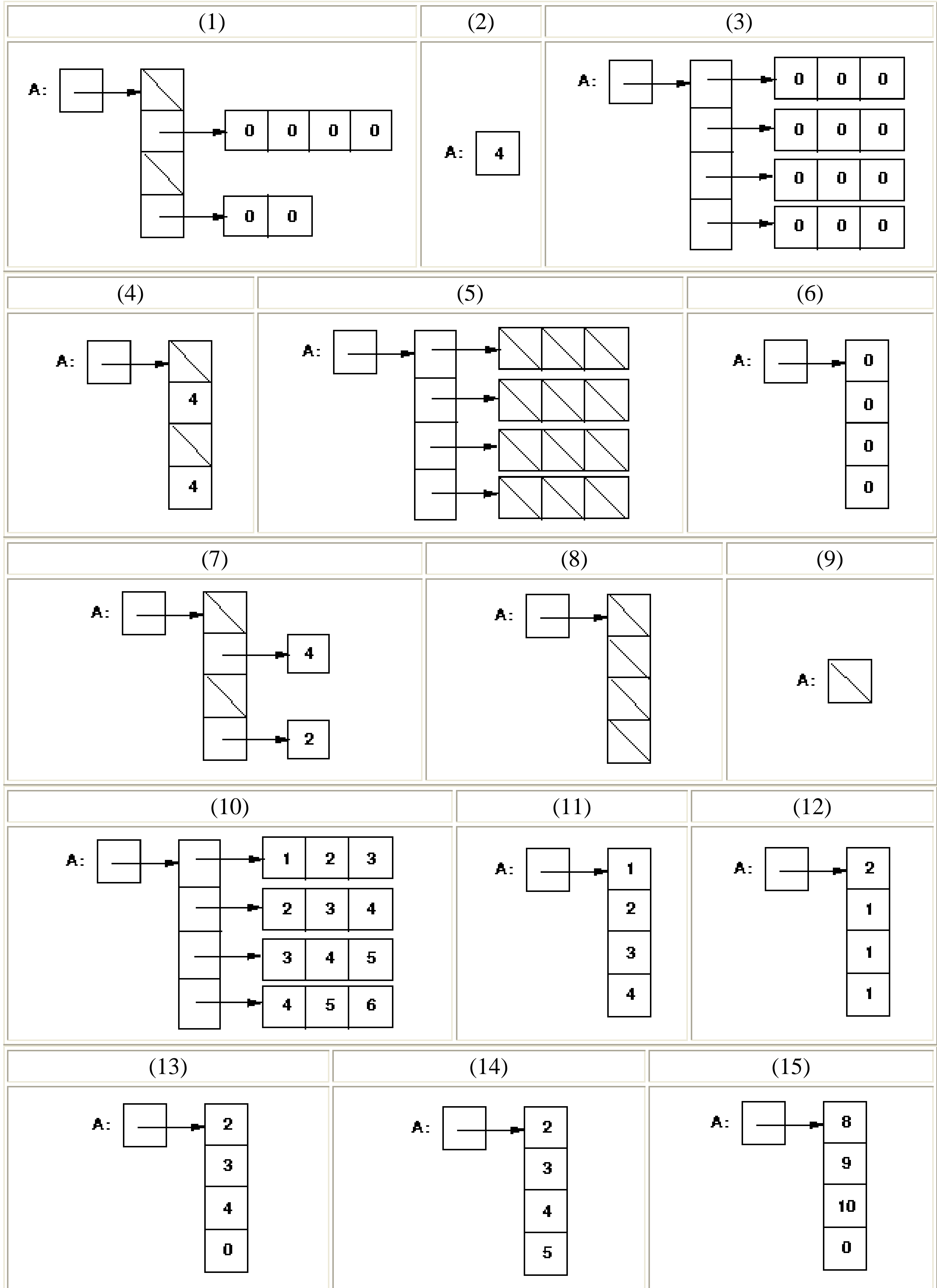
| *kód* | *Číslo obrázku, chyba, alebo neexistuje obrázok* |
|---|---|
| `int [] A;` | |
| `int [] A = new int [4];` | |
| `int [][] A = new int[4][3];` | |
| `int [][] A = new int[4][];`<br>`A[1] = new int[4];`<br>`A[3] = new int[2];` | |
| `int [] A = new int[4];`<br>`int [] B = {0,1,2,3,4,5,6,7,8,9};`<br>`System.arraycopy(B,2,A,0,4);` | |
| `int [] A = new int[4];`<br>`int [] B = {2,3,4};`<br>`System.arraycopy(B,0,A,0,4);` | |
| `int [] A = new int[4];`<br>`int [] B = {0,1,2,3,4,5,6,7,8,9};`<br>`System.arraycopy(B,8,A,0,4);` | |
| `int [] A = {1,1,1,1};`<br>`int [] B = {2,2,2};`<br>`System.arraycopy(A,0,B,1,2);`<br>`System.arraycopy(B,0,A,0,3);` | |
| `int [] A = new int[4];`<br>`int [] B = {0,1,2,3,4,5,6,7,8,9};`<br>`System.arraycopy(B,0,A,0,10);` | |
| `int [][] A = new int[4][3];`<br>`int [] B = {1,2,3,4,5,6,7,8,9,10};`<br>`System.arraycopy(B,0,A[0],0,3);`<br>`System.arraycopy(B,1,A[1],0,3);`<br>`System.arraycopy(B,2,A[2],0,3);`<br>`System.arraycopy(B,3,A[3],0,3);` | |

**Help:**

```
public static void arraycopy(Object src,
               int srcPos,
               Object dest,
               int destPos,
               int length)
```

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array. A subsequence of array components are copied from the source array referenced by `src` to the destination array referenced by `dest`. The number of components copied is equal to the `length` argument. The components at positions `srcPos` through `srcPos+length-1` in the source array are copied into positions `destPos` through `destPos+length-1`, respectively, of the destination array.

## Obrázky



**4) [5 bodov]** V tomto príklade reprezentujeme orientovaný ohodnotený graf medzi mestami SR. Intuitívne, hodnota hrany je vzdialenosť z mesta do druhého mesta. Orientovaný znamená, že k hrane z A->B nemusí existovať z B->A. Najprv definujeme triedu Dvojica, ktorá popisuje cieľovú stanicu a jej vzdialenosť:

```
public class Dvojica implements Comparable<Dvojica> {
  String kam;               // meno mesta - cieľová stanica
  int vzdialenost;
  public Dvojica(String kam, int vzdialenost) {  // konštruktor
       this.kam = kam; this.vzdialenost = vzdialenost;
  }
  @Override
  public String toString() { return "("+kam+":"+vzdialenost+")"; }




}
```

**[1 bod]** V definícii triedy Dvojica chýba metóda `compareTo`. Doplňte ju do voľného miesta tak, aby menšia dvojica bola tá, ktorá je bližšie, t.j. má menšiu vzdialenosť. Nasledujúci kód spustíme s argumentami v príkazovom riadku

```
java Cities BA:KE:401 NI:KE:307 NI:PP:234 PP:KE:119 PP:BA:327 KE:BA:401 BB:PP:115 NI:BB:118
```

kde hrany grafu sú zadané v trojiciach, vo formáte odkiaľ:kam:vzdialenosť.

```
public class Cities {
  static TreeMap<String, TreeSet<Dvojica>> cudo = new TreeMap<String, TreeSet<Dvojica>>();
  public static void main(String[] args) {
       for (String arg : args) {        // arg má tvar odkiaľ:kam:vzdialenosť
              String[] hrana = arg.split(":");
              String odkial = hrana[0];
              String kam = hrana[1];
              int vzdialenost = Integer.parseInt(hrana[2]);
              TreeSet<Dvojica> set = cudo.get(odkial);
              if (set == null)
                     set = new TreeSet<Dvojica>();
              set.add(new Dvojica(kam, vzdialenost));
              cudo.put(odkial, set);
       }
       System.out.println(cudo);
```

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

1. `{[BA,BB,KE,NI,PP]=[(KE:401)],[(PP:115)],[(BA:401)],[(BB:118),(PP:234),(KE:307)],[(KE:119),(BA:327)]}`
2. `{BB=[(PP:115)], NI=[(BB:118),(PP:234),(KE:307)], PP=[(KE:119),(BA:327)],BA=[(KE:401)],KE=[(BA:401)]}`
3. `{BA=[(KE:401)],BB=[(PP:115)],KE=[(BA:401)],NI=[(BB:118),(KE:307),(PP:234)],PP=[(BA:327),(KE:119)]}`
4. `{BA=[(KE:401)],BB=[(PP:115)],KE=[(BA:401)],NI=[(BB:118)],NI=[(PP:234)],NI=[(KE:307)],PP=[(KE:119)], PP=[(BA:327)]}`
5. `{BA=[(KE:401)],BB=[(PP:115)],KE=[(BA:401)],NI=[(BB:118),(PP:234),(KE:307)],PP=[(KE:119),(BA:327)]}`
6. `{(BA:KE:401),(NI:KE:307),(NI:PP:234),(PP:KE:119),(PP:BA:327),(KE:BA:401),(BB:PP:115),(NI:BB:118)}`
7. niečo ine:

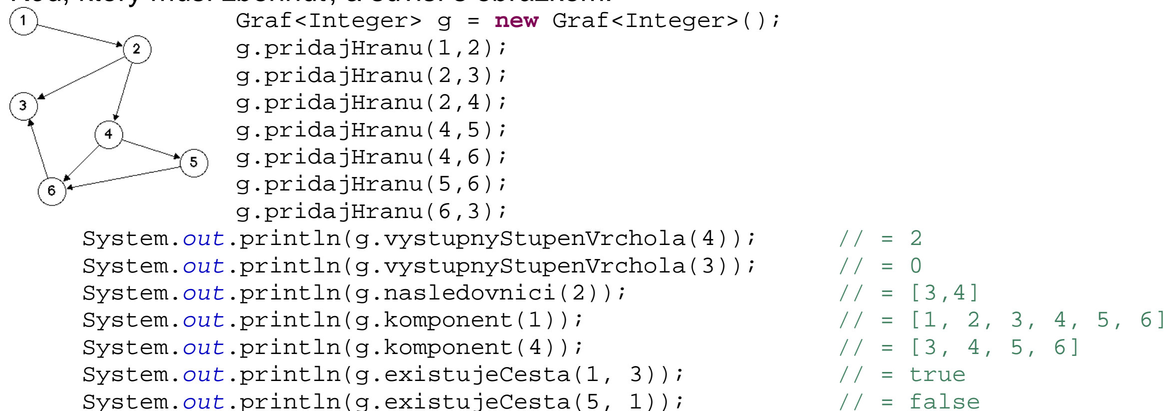
Napíšte kus kódu, ktorý z hodnoty premennej `cudo` nájde niektoré dve najvzdialenejšie mestá, v uvedenom príklade sú to KE a BA **[2 body]**

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

**5) [7 bodov]** Orientovaný graf pozostáva z hrán (jednosmerných šípiek) medzi vrcholmi parametrizovateľ-ného typu E. V celom zadaní môžete predpokladať, že graf **neobsahuje cyklus**. Vašou úlohou je navrhnúť vlastnú reprezentáciu grafu v triede `Graf<E extends Comparable<E>>`, ktorá implementuje nasledujú-ci interface:

```
public interface GrafI<E extends Comparable<E>> {
  public void pridajHranu(E a, E b);    // zapamätá si hranu z vrchola a do b       [1bod]
  public int vystupnyStupenVrchola(E a);// počet hrán vychádzajúcich z vrchola a    [1bod]
  public HashSet<E> nasledovnici(E a);// množina vrcholov,do ktorých vedie hrana z a [1bod]
  public HashSet<E> komponent(E a); // množina vrcholov dosiahnuteľných z vrchola a [2body]
  public boolean existujeCesta(E a, E b); // existuje cesta z a do b               [2body]
```

Kód, ktorý musí zbehnúť, a súvisí s obrázkom:




```
                Graf<Integer> g = new Graf<Integer>();
                g.pridajHranu(1,2);
                g.pridajHranu(2,3);
                g.pridajHranu(2,4);
                g.pridajHranu(4,5);
                g.pridajHranu(4,6);
                g.pridajHranu(5,6);
                g.pridajHranu(6,3);
    System.out.println(g.vystupnyStupenVrchola(4));      // = 2
    System.out.println(g.vystupnyStupenVrchola(3));      // = 0
    System.out.println(g.nasledovnici(2));               // = [3,4]
    System.out.println(g.komponent(1));                  // = [1, 2, 3, 4, 5, 6]
    System.out.println(g.komponent(4));                  // = [3, 4, 5, 6]
    System.out.println(g.existujeCesta(1, 3));           // = true
    System.out.println(g.existujeCesta(5, 1));           // = false
```

Ak metódy `komponent` a `existujeCesta` implementujete tak, že fungujú aj na cyklickom grafe (t.j. existuje v ňom cyklus po orientovaných hranách), môžete získať +0.5 bodu za každú z nich. Ak však napíšete explicitne (a slovom), že vaše metódy fungujú na acyklickom grafe, tak získate za každú +1 bod, ak funguje, a -1 bod, ak nefunguje. Svet patrí odvážnym...

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