**Midterm 2020 - ONLINE**

**Inštrukcie**

všetky zdrojové súbory k príkladom, s testami, sú tu [src.zip](https://list.fmph.uniba.sk/index.php/tasks/download_hidden_file/2369/c3JjLnppcA__)

**Important! Čas: 2:10 min, koniec 14:45.**

**Komunikácia počas Midtermu:**

* **otázky píšte do MS Teams (do kanála PROG4JAVA/Cvičenia), alebo na mail prog4java@lists.dai.fmph.uniba.sk**
* **publikovaná otázka môže byť čokolvek, ale nesmie zobrazovať kó ani inú časť riešenia úlohy, na tomto ani iných kanáloch**
* **V prípade problémov nám (individuálne) nazdieľajte obrazovku v MSTeams, a budeme sa snažiť vám pomôcť.**
* **organizačné veci a prípadné chyby v zadaniach budeme publikovať na MS Teams/PROG4JAVA/Cvičenia**

**Ďakujeme za pochopenie**

**1. Rekurzia**

Táto rekurzívna funkcia je neefektívne napísaná, lebo nevypočíta hodnotu ani pre a=b=15. Vašou úlohou je ju prepísať tak, aby počítala efektívnejšie. Nie je podstatné, čo počíta. Možno vám to pripomína rekurentný vzťah pre kombinačné čísla, ale to je falošná stopa.

public static long foo(int a, int b) {  
 if (a == 0 || b == 0)  
 return 1;  
 else  
 return foo(a-1,b) + foo(a-1, b-1) + foo(a, b-1);  
}

**Úlohy:** V triede **Rekurzia**

* **[2 body]** skúste použiť pole (memoizáciu) na to, aby ste prepísali funkciu tak, aby vypočítala všetky hodnoty v rozsahu **long**. Predefinujte telo funkcie **public static long foo(int a, int b)**.
* **[2 body]** definujte funkciu **public static boolean zOboruHodnot(long n)**, ktorá zistí pre vstupné **n**, či existujú a, b také, že foo(a,b) = n.  
  Hint: skôr, než dvakrát vnoríte cyklus, vypíšte si tabuľku hodnôt, stačí malý kúsok ľavého horného rohu. Aké čísla určite budú v tabuľke? Aké sa tam určite neobjavia? ...

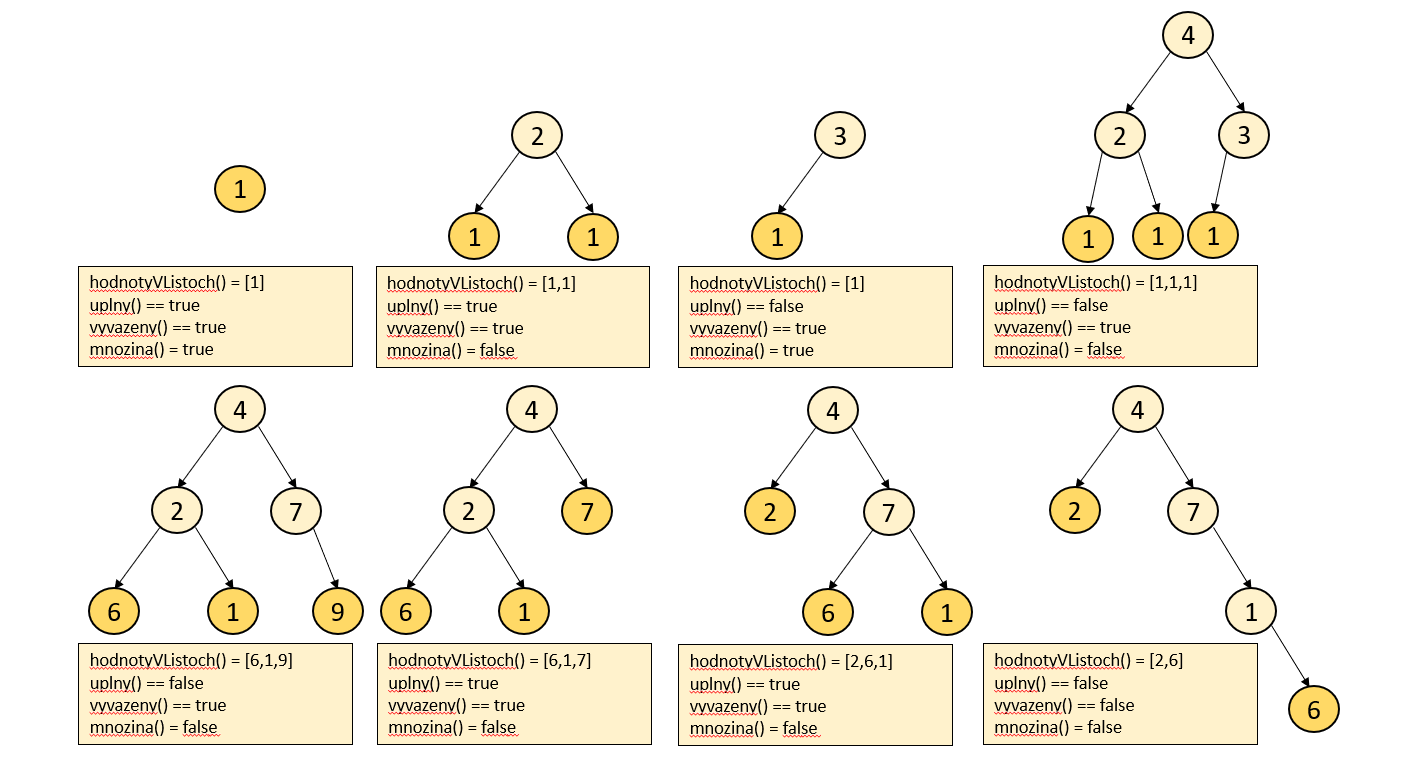
Body: 4

Autor: Peter Borovanský

**2. Stromový**

Vrchol binárneho stromu je definovaný v parametrizovanej triede **Node**. List stromu (na obrázku tmavožltá) je vrchol, ktorý nemá žiadneho syna. Vrchol, ktorý nie je listom, môže mať jedného alebo dvoch synov.

public class Node<E> {  
 private E value;  
 private Node<E> left, right;  
 public Node(Node<E>left, E value, Node<E> right) { ... }



**Úlohy**: V triede **Node**dodefinujte telá metód

* **[1 bod]** **public List<E> hodnotyVListoch()**, ktorá vráti zoznam hodnôt v **listoch** stromu (v ľubovoľnom poradí)
* **[1 bod] public boolean uplny()**, ktorá vráti true, ak žiaden vrchol nemá jediného syna, teda všetky (vnútorné) vrcholy majú dvoch synov a listy žiadneho
* **[2 body] public boolean vyvazeny()**, ktorá vráti true práve vtedy keď vzdialenosť všetkých listov od koreňa stromu sa líši najviac o 1
* **[2 body] public boolean mnozina()**, ktorá vráti true práve vtedy keď strom reprezentuje množinu uloženú v Binárnom vyhľadávacom strome. To znamená že zároveň platí:
  + strom je BVS: pre každý vrchol ***v*** platí, že všetky prvky v ľavom podstrome sú menšie ako hodnota ***v***, a všetky prvky v pravom podstrome sú väčšie ako hodnota ***v***
  + prvky sa v strome neopakujú

Body: 6

Autor: Peter Borovanský

**3. Oprav ma**

Príklad obsahuje 5 nasúvisiach podúloh:

* **[1 bod]**Kód, ktorý chcel utriediť pole nejakým sortom, ale netriedi. Opravte ho!  
  Upresnenie: máte nájsť chybu v tomto kóde, opraviť ju, a nie napísať Arrays.sort(s).  
  Upresnenie2: Kolekcie ani streamy nie sú v tomto príklade dovolené.

public static void Uloha1(int[] s) {  
 for(int i = 0; i < s.length; i++)  
 for(int j = 0; j < s.length; j++)  
 if (s[i] > s[j]) {   
 int temp = s[i]; s[i] = s[j]; s[j] = temp;  
 }  
}

* **[1 bod]**Kód vráti maticu 2x2 s hodnotami a, teda {{a,a}, {a,a}} , lenže test padá. Preštudujte si test a opravte funkciu Uloha2, tak aby test nepadal.

public static String[][] Uloha2(String a) {  
 String[] row = new String[]{a,a};  
 return new String[][]{row,row};  
}  
test:  
String[][] matrix = OpravMa.Uloha2(randomString());  
assertEquals("[0][0]=[1][0]", matrix[0][0], matrix[1][0]); // ok  
matrix[0][0] = randomString();  
assertNotEquals("[0][0] a [1][0] musia byt rozne, lenze nie su :(", matrix[0][0], matrix[1][0]); // padne

* **[1 bod]**dodefinujte triedy Zviera, Psicek, Macicka, aby prešiel tento test

assertEquals("psicek poslucha", Psicek.a, Zviera.a); // ==  
assertNotEquals("macicka neposlucha", Macicka.a, Zviera.a); // !=  
Zviera[] zoo = new Zviera[]{ new Psicek(), new Macicka() };  
assertEquals("zvuky", 2, Stream.of(zoo).map(z -> z.sound()).distinct().count());

* **[1 bod]** dodefinujte triedu Zajac, aby zbehol tento test:

List<Zajac> ls = List.of(new Zajac(1), new Zajac(2), new Zajac(1), new Zajac(2));  
TreeSet<Zajac> ts = new TreeSet<>(ls);  
HashSet<Zajac> hs = new HashSet<>(ls);  
assertEquals("len dvaja", 2, ts.size());  
assertEquals("mensi", new Zajac(1), ts.first());  
assertEquals("vacsi", new Zajac(2), ts.last());  
assertEquals("len dvaja", 2, hs.size());

* **[1 bod]** dodefinujte Uloha5, ktorá ak dostane ako argument null, alebo pole nulovej dĺžky, vyhodí Exception("nieco zle")

public static void Uloha5(int[] a) { }

Body: 5

Autor: Peter Borovanský

**4. Kolekcie**

V tomto príklade modelujeme orientovaný graf s vrcholmi, ktoré majú mená typu String, a hrany majú mená typu Character.

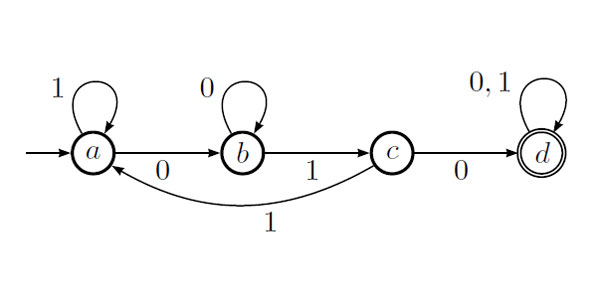
**Pre tých, čo mali UTI:** Ide vlastne o deterministický konečný automat. Jediná zmena je v tom, že pokiaľ zo stavu nie je zadaná hrana s nejakým písmenom, tak táto hrana ide automaticky do reject stavu.

**Pre tých, čo nevedia čo je konečný automat:** V tomto orientovanom grafe sa chodí na povely - písmenká, ktoré musia súhlasiť s označením hrany, po ktorej ideme. Takáto prechádzka po grafe je potom postupnosť písemen, t.j. slovo, čiže celú cestu vieme zapísať ako jeden String. Je kľudne možné, že ste v nejakom stave (vrchole) a nasledujúce písmenko cesty vám velí vydať sa po hrane, ktorá v danom vrchole neexistuje - vtedy končíte prechádzku a presúvate sa do stavu null.

Graf reprezentujeme štruktúrou **Map<String, Map<Character, String>> graf**. Kľúče mapy sú mená stavov (typ String). Hodnotou je mapa, ktorá zobrazuje meno hrany (Character) do vrchola, kam hrana s daným názvom vedie.

Požičaný obrázok z domácej úlohy 3 zobrazuje automat, ktorý akceptuje všetky slová z {0,1}∗

ktoré obsahujú substring "010":



Graf graf1 = new Graf(Map.of( "a", Map.of('0', "b", '1', "a"), // graf z obrazku vyssie  
 "b", Map.of('0', "b", '1', "c"),  
 "c", Map.of('0', "d", '1', "a"),  
 "d", Map.of('0', "d", '1', "d") ));  
  
Graf graf2 = new Graf(Map.of( "X", Map.of('1', "XXX", '2', "Y"),  
 "Y", Map.of('1', "Z"),  
 "Z", Map.of('1', "ZZZ", '8', "ZZZ") ));

**Úlohy**: V triede **Graf** dodefinujte:

* **[1.5 bodu]** metódu **public Set<String> menaVrcholov()**, ktorá pozbiera a vráti mená všetkých vrcholov (stavov) daného grafu (automatu). Ale pozor, stavy sú nie len v **graf.keySet()**, ale aj v mapách jednotlivých hrán, a nemusí to byť rovnaká množina. Vrátiť máte všetky mená vrcholov, ktoré sa v grafe nachádzajú.  
  Príklad: **graf1.menaVrcholov == {"a","b","c","d"};  graf2.menaVrcholov == {"X","XXX","Y","Z","ZZZ"}.**
* **[1.5 bodu]** metódu **public Set<Character> menaHran()**, ktorá vráti mená všetkých hrán (abeceda) daného grafu (automatu).  
  Príklad: **graf1.menaHran == {'0','1'};  graf2.menaHran == {'1','2','8'}.**
* **[2 body]** metódu **public String simuluj(String vychodisko, String cesta)**, ktorá vráti meno vrchola (stavu), kam sa z vrchola **vychodisko** dostanete postupným nasledovaním hrán podľa písmenok z **cesta**. Ak počas cesty nastane okamih, že z vrchola neexistuje cesta na predpísané písmenko, tak výsledkom je **null.**

**Príklad**:

graf1.simuluj("a","0010").equals("d"),   
graf1.simuluj("a","").equals("a"),   
graf1.simuluj("a","").equals("a"),   
graf1.simuluj("a","2")==null,  
graf1.simuluj("f","0")==null.

Body: 5

Autor: Peter Borovanský

**5. Streamový**

* **[1 bod]** dodefinujte metódu striedavo(int n), ktorá vráti IntStream celých čísel *-n..n* v poradí *0,-1,1,-2,2, ..., -n,n*

public static IntStream striedavo(int n) { ... }

* **[1 bod]** dodefinujte metódu prvocisla(IntStream input), ktorá prefiltruje input stream a ponechá v ňom len prvočísla (v tom istom poradí). Nezabudnite že prvé prvočíslo je 2, nie 1.

public static IntStream prvocisla(IntStream input) { ... }

* **[2 body]** dodefinujte metódu najvyssiaCifra(List<Integer> vstup), ktorá vráti frekvenčnú tabuľku najvyšších cifier v zozname vstup. Najvyššia cifra = prvá cifra v desiatkovom zápise čísla.  
  Príklad: ak vstupný zoznam obsahuje čísla **1**0,**4**31,**2**1,**1**21,**2**1,**1**4, tak výsledná frekvenčná tabuľka je { 1: 3, 2: 2, 4: 1 }.

public static Map<Integer, Long> najvyssiaCifra(List<Integer> vstup) { ... }

* **[2 body]** dodefinujte metódu delitele(IntStream input), ktorá pre prvky x streamu input vyrobí mapu, v ktorej x je kľúčom a hodnotou je zoznam jeho deliteľov z intervalu *1..x*.  
  Poznámka: 1 je určite prvý prvok tohoto zoznamu deliteľov, a *x* je určite posledný prvok v zozname.

public static Map<Integer, List<Integer>> delitele(IntStream input) { ... }

Body: 5

Autor: Peter Borovanský