**Midterm 2021 - ONLINE**

**Inštrukcie**

**POSUNUTÝ DEADLINE na 20:55**

všetky zdrojové súbory k príkladom spolu s testami, sú tu [src.zip](https://list.fmph.uniba.sk/index.php/tasks/download_hidden_file/2641/c3JjLnppcA__) (na rozbehanie testov možno potrebujete pridať juni4 do classpath::Alt+Enter na junit, ok)

test obsahuje 5 úloh za 4+6+5+6+6=27 bodov.

**Čas: 2:15 min, začiatok 18:10, koniec 20:25.**

**Komunikácia počas Midtermu:**

* **otázky píšte alebo sa pýtajte v MS Teams (v kanáli PROG4JAVA/Cvičenia)**
* **publikovaná otázka môže byť čokolvek, ale nesmie zobrazovať kód 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 - M1**

Táto rekurzívna funkcia na výpočet hodnoty foo(a,b) potrebuje sčítať všetky hodnoty foo(i,j) pre i<=a, j<=b (samozrejme okrem foo(a,b)). Je napísaná neefektívne, lebo nevypočíta už ani hodnotu pre a=b=15. A tú hodnotu nevypočíta len preto, že už dávno **vypočítané hodnoty pre menšie (i,j) ako (a,b) si nepamätá**. Vašou úlohou je ju prepísať tak, aby počítala hodnoty funkcie efektívnejšie.

public static long foo(int a, int b) {  
 if (a == 0 && b == 0)  
 return 1;  
 else {  
 long sum = 0L;  
 for (int i = 0; i <= a; i++)  
 for (int j = 0; j <= b; j++) {  
 if (i == a && j == b) continue;  
 sum += *foo*(i, j);  
 }  
 return sum;  
 }  
}

**Úlohy:** V triede **Rekurzia**

* **[1 bod]** skúste použiť pole (možno dvojrozmerné) či inú dátovú štruktúru na memoizáciu (po slovensky, pamätanie si) výsledkov, ktoré ste už počítali, a prepíšte funkciu public static long foo(int a, int b)  tak, aby vypočítala hodnotu aj pre a=b=15. Inak..., nám to vyšlo 731420783788032L, len pre vašu kontrolu. A v teste nájdete aj iné, menšie benchmarky. Predefinujte priamo telo funkcie public static long foo(int a, int b) na efektívnejšiu verziu.

...keď už máte efektívnejšiu verziu funkcie...

* **[1 bod]** nájdite najväčšiu hodnotu funkcie **long foo(int a, int b)** v rozsahu typu **long.** Definujte funkciu public static long maxHodnota(), ktorá vráti maximálnu hodnotu funkcie foo, ktorá je ešte v rozsahu typu **long**. Očakáva sa riešenie/kód, ktorý max.hodnotu hľadá. Riešenie napr. typu return 545684246L; bude diskvalifikované pri posudzovaní, ak aj konštanta je prípadne dobrá a prejde testom.
  + **Hint1:** Uvedomne si pri tom, že **všetky korektné hodnoty funkcie sú kladné čísla.** Záporý výsledok je teda podozrivý, a naznačuje, že ste zažili overflow v longu. Taktiež test **long hodnota = ....; if (hodnota <= Long.MAX\_VALUE)** je nezmysel (automaticky **true**), ktorý vám **ofarbí aj IntelliJ**, lebo každá hodnota long je najviac MAX\_VALUE.
  + **Hint2:** Príklad naozaj nie je na to, aby ste objavili BigInteger, stačí sa len zamyslieť, ako vyzerá aritmetické pretečenie v longu.
* **[1 bod]** overte, či platí vzťah foo(a,b) = 2\*(foo(a-1,b) + foo(a,b-1) - foo(a-1,b-1)) pre 0 < a,b <= 15. Definujte funkciu public static int kontrapriklad1(), ktorá vráti a+b pre také 0 < a,b <= 15, **pre ktoré vzťah neplatí**, teda foo(a,b) != 2\*(foo(a-1,b) + foo(a,b-1) - foo(a-1,b-1)). **Inak** (ak také a,b neexistujú) funkcia **vráti -1**.
* **[1 bod]** definujte analógiu predchádzajúcej funkcie public static int kontrapriklad2(), skúma ten istý vzťah, ale a, b berie **z iného intervalu 1 < a, b < 1000000**. Všimnite, že sa zmenila nie len horná ale aj dolná hranica intervalu pre a,b.  A milion\*milion je dosť času na premýšľanie... Ak také a,b nájdete, funkcia kontrapriklad2 vráti a+b. Inak (ak také a,b neexistujú), vráti -1.

Body: 4

Autor: Peter Borovanský

**2. Stromový - M2**

Parametrický strom s ľubovoľným počtom synov (Rhododendron) ste mali predstavený v domácej úlohe. Jeho vrchol Node<T> obsahuje hodnotu value typu T, zoznam podstromov sons typu Node<T>, a je definovaný v parametrizovanej triede **Node<T extends Comparable<T>>**. List stromu je taký vrchol stromu, ktorý nemá žiadneho syna, teda zoznam synov sons je prázdny, alebo null. Môžete predpokladať, že hodnoty value vo vrcholoch testovacích stromov sú rôzne. Aj preto, **T extends Comparable<T>**.

public class Node<T extends Comparable<T>> implements RTree<T> {  
 private final T value;  
 private final List<Node<T>> sons;  
  
 public Node(T value, List<Node<T>> sons) {... }

Inteface, ktorý idete implementovať, obsahuje nasledovné metódy:

interface RTree<T extends Comparable<T>> {  
 public int size(); *// počet vrcholov stromu* public int depth(); *// hĺbka stromu, pre list je to 0, inak je to 1 + maximum hĺbok podstromov* public Set<T> leafs(); *// množina listov stromu, pričom list je vrchol, ktorý nemá synov* public boolean isHeap(); *// pre každý vrchol stromu platí, že je väčší ako všetky vrcholy jeho podstromoch (synoch)* public boolean atSameLevel(T e1, T e2); *// hodnoty e1 a e2 sa nachádzajú niekde v strome v rovnakej hĺbke od koreňa stromu*}

**Úlohy: V triede Node definujte telá metód (resp. implementujte interface):**

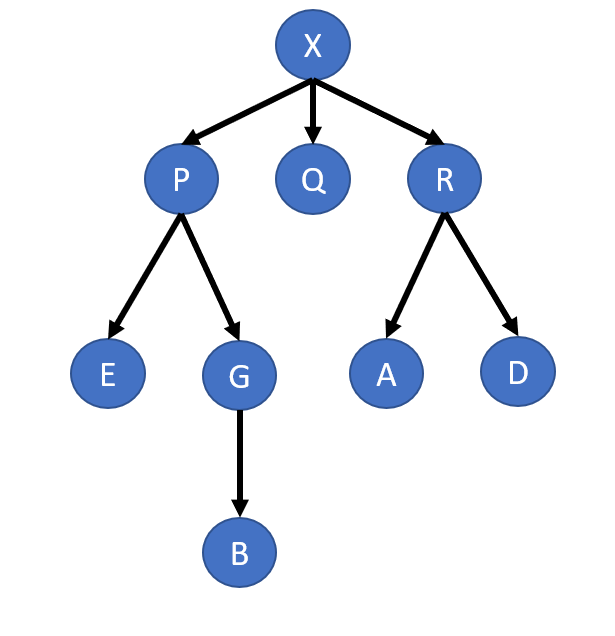
* **[1 bod]** public int size() je počet vrcholov stromu,
* **[1 bod]** public int depth() je hĺbka stromu, pre list je to 0, inak je to 1+max.hĺbok podstromov
* **[1 bod]** public Set<T> leafs()  je množina listov stromu, list je vrchol, ktorý nemá synov,
* **[1 bod]** public boolean isHeap() pre každý vrchol stromu platí, že je väčší ako všetky hodnoty vrcholov jeho podstromoch (synoch),
* **[2 body]** public boolean atSameLevel(T e1, T e2)  hodnoty e1 a e2 sa nachádzajú niekde v strome v rovnakej hĺbke od koreňa stromu.

**Príklad**:

strom vytvorí takéto volanie konštruktorov

Node<String> rt = new Node<>("X", List.*of*(  
 new Node<>("P", List.*of*(  
 new Node<>("E",List.*of*()),  
 new Node<>("G",List.*of*(  
 new Node<>("B",List.*of*())))  
 )),  
 new Node<>("Q", List.*of*()),  
 new Node<>("R", List.*of*(  
 new Node<>("A", List.*of*()),  
 new Node<>("D", List.*of*())  
 ))  
 )  
 );

* size() je 9,
* depth() je 3,
* leafs() sú vrcholy A, B, D, E, Q, teda Set.of("A", "B", "D", "E", "Q"),
* isHeap platí, lebo "X">"P", "X">"Q", "X">"R", "P">"E", "P">"G", "G">"B",...
* atSameLevel(E,D), atSameLevel(E,G), atSameLevel(E,E), atSameLevel(P,R) platí,
* atSameLevel(G,Q), atSameLevel(E,B), atSameLevel(B,X) neplatí.



Body: 6

Autor: Peter Borovanský

**3. Oprav ma - M3**

Príklad obsahuje 5 nesúvisiacich podúloh, riešenie realizujte do priloženej triedy **OpravMa**.

* **[1 bod]**Kód, ktorý chcel utriediť pole nejakým sortom, ale netriedi. Opravte ho!
  + Upresnenie1: máte nájsť chybu v tomto kóde, opraviť ju, a nie napísať Arrays.sort(s). Také riešenie sa diskvalifikuje.
  + Upresnenie2: Kolekcie ani streamy nie sú v tomto príklade dovolené.
  + Upresnenie3: V prípade hraničných nejasností, pýtajte sa.
  + public static void Uloha1(int[] a) {  
     for (int i = 0; i < a.length ; i++) {  for (int j = i+1; j < a.length; j++) {  if (a[j-1] > a[j]) {  
     int temp = a[j];  
     a[j] = a[j-1];  
     a[j-1] = temp;  
     }  }  }  
    }

* **[1 bod]**Kód vráti zoznam zoznamov Integerov, a jednotlivé zoznamy by mali byť [0], [0,1], [0,1,2]...[0,1,2...n-1], lenže test neprejde. Preštudujte si test a opravte funkciu Uloha2, tak aby test nepadal.

public static List<List<Integer>> Uloha2(int n) {  
 List<Integer> list = new LinkedList<>();  
 List<List<Integer>> result = new ArrayList<>();  
 for(int i = 0; i < n; i++) {  
 list.add(i); *// list ma tvar 0,1, ... , i* result.add(list); *// do result prida dalsi, este dlhsi riadok 0,1, ..., i* }  
 return result;  
}  
... test

for(int i = 0; i<list.size(); i++) {  
 *assertEquals*("U2-1 i+1 == list.get(i).size()", i+1, list.get(i).size());  
 *assertEquals*("U2-2 list.get(i) == {0..i}",   
 IntStream.*range*(0,i+1).boxed().collect(Collectors.*toList*()), list.get(i));  
}

* **[1 bod]**definujte triedy Matfyzak, Bakalar, Magister a Profak (nájdete ich ako súčasť src.zip) tak, aby prešiel tento test

var matfyz = List.*of*(new Bakalar(), new Magister());  
*assertTrue*("U3-1", matfyz.stream()  
 .map(Matfyzak::goo).collect(Collectors.*toSet*()).size() > 1); *// true  
assertTrue*("U3-2",Matfyzak.*foo*() **!=** Bakalar.*foo*()); *// not // true  
assertTrue*("U3-3",Matfyzak.*foo*().equals(Bakalar.*foo*())); *// true  
assertTrue*("U3-4",new Magister().goo() **!=** new Bakalar().goo()); *// not // true  
assertTrue*("U3-5",**!**new Magister().*foo*().equals(new Bakalar().goo())); *// not // true  
assertTrue*("U3-6",Profak.*zoo*() **!=** Profak.*zoo*()); *// not // true  
assertTrue*("U3-7",**!**Profak.*zoo*().equals(Profak.*zoo*())); *// not // true*

* **[1 bod]** definujte triedy  Bakalar, Magister tak, aby zbehol tento test:

var b1 = new Bakalar("Palko");  
var b2 = new Bakalar("Ferko");  
var m1 = new Magister("Lukas");  
var m2 = new Magister("Jozko");  
*assertEquals*("U4-1", 2, new TreeSet<>(List.*of*(m1, m2)).size());  
*assertEquals*("U4-2", m2, new TreeSet<>(List.*of*(m1, m2)).first());  
*assertEquals*("U4-3", 1, new TreeSet<>(List.*of*(b1, b2)).size());  
*assertEquals*("U4-4", 3, new HashSet<>(List.*of*(b1, b2, m1, m2)).size());

* **[1 bod]** dodefinujte Uloha5, ktorá **skončí výnimkou Exception("zly vstup"), ak** argumentom metódy Uloha5(String [][] a)
* nie je dvojrozmerné pole reťazcov, ktorého všetky riadky sú rovnako dlhé,
* alebo nejaký reťazec v poli je null,
* alebo argumentom je null.

Pre obdĺžnikové polia (všetky riadky rovnako dlhé) a plné skutočných (nie-null) reťazcov nemusí robiť nič, ale nesmie skončiť výnimkou.

public static void Uloha5(String [][] a) ...

Body: 5

Autor: Peter Borovanský

**4. Kolekcie - M4**

V tomto príklade pracujeme s orientovaným grafom s vrcholmi parametrického typu. Graf pozostáva z vrcholov a orientovaných hrán. Orientácia pri hrane A -> B znamená, že z vrcholu A vedie hrana vrcholu B. Graf reprezentujeme ako zobrazenie/mapu/dictionary z vrcholov do množiny vrcholov, do ktorých vedie orientovaná hrana. Pracujete s triedou **Graph**

public class Graph<V extends Comparable<V>> {  
 Map<V, Set<V>> g;  
 public Graph(Map<V, Set<V>> g) {  
 this.g = g;  
 }

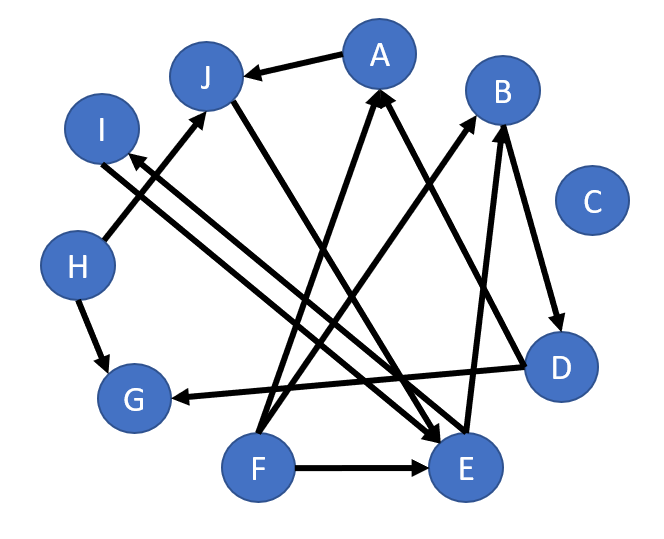
a toto je príklad hodnoty aj s volaním konštuktora, ktorý je dole zobrazený ako graf: *//g={A=[J], B=[D], C=[], D=[A, G], E=[B, I], F=[A, B, E], G=[], H=[G,J], I=[E], J=[E]}* var g = new Graph<Character>(Map.*of*(   
 'A',Set.*of*( 'J'),  
 'B',Set.*of*( 'D'),  
 'C',Set.*of*(),  
 'D',Set.*of*( 'A','G'),  
 'E',Set.*of*( 'B','I'),  
 'F',Set.*of*( 'A','B','E'),  
 'G',Set.*of*(),  
 'H',Set.*of*( 'G','J'),  
 'I',Set.*of*( 'E'),  
 'J',Set.*of*( 'E')  
));

**Úlohy:** Všetky funkcie definujte v priloženej triede **Graph**:

* **[1 bod]** prameň v grafe je vrchol, **do ktorého nevedie žiadna hrana**. Definujte metódu public boolean jePramen(V pramen), ktorá rozhodne, či vrchol je prameň grafu. Následne definujte metódu public Set<V> pramene(), ktorá vráti všetky pramene grafu.
* **[1 bod]** ústie v grafe je vrchol, **z ktorého nevedie žiadna hrana**. Definujte metódu public boolean jeUstie(V pramen), ktorá rozhodne, či vrchol je ústie grafu. Následne definujte metódu public Set<V> ustia(), ktorá vráti všetky ústia grafu.
* **[2 body]** cesta v grafe z vrchola do vrchola musí rešpektovať orientáciu hrán. Definujte metódu public boolean cesta(V from, V to), ktorá rozhodne, či existuje v grafe orientovaná cesta z vrchola from do vrcholu to. **Rada od Ariadny**: graf môže obsahovať cykly.
* **[2 body]** graf je cyklický, ak obsahuje orientovaný cyklus. Definujte metódu public boolean cyklicky(), ktorá rozhodne o celom grafe, či v ňom existuje orientovaný cyklus z niektorého vrcholu naspäť do toho istého vrcholu.

**Príklad:** V tomto grafe:

* C,F,H sú pramene a žiadne iné nie sú,
* C,G sú ústia a žiadne iné nie sú,
* existuje cesta(J,B), (J-E-B)
* existuje cesta(F,D), (F-A-J-E-B-D)
* neexistuje cesta(J,H),
* a graf je cyklický, lebo existuje v ňom cyklus, napr. I-E-I, ale aj A-J-E-B-D-A



Body: 6

Autor: Peter Borovanský

**5. Streamový - M5**

**Úlohy:** V priloženej triede **Streams**

* **[1 bod]** definujte metódu IntStream pyramida(int n), ktorá pre n >= 0 vráti IntStream celých čísel v  poradí *n, n,n-1, n,n-1,n-2, ... , n,n-1,..,1*

public static IntStream pyramida(int n)  
Príklad: *pyramida*(5) vyzerá takto: Stream.of(  
 5,   
 5, 4,   
 5, 4, 3,   
 5, 4, 3, 2,   
 5, 4, 3, 2, 1  
).

* **[1 bod]** definujte metódu IntStream palindromy(IntStream input), ktorá prefiltruje input stream nazáporných čísel a ponechá v ňom len čísla, ktorých dekadický zápis je symetrický, tzv. palindromy (a nechá ich v tom istom poradí ako boli vo vstupe). Napríklad, 11, 121, 4224 sú palindromy a 12, 110 nie sú palindromy.
* public static IntStream palindromy(IntStream input)  
  Príklad: *palindromy*(IntStream.*range*(0,100)) vyzerajú takto:   
  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99]
* **[2 body]** definujte metódu Map<Integer, Integer> pocetDelitelov(List<Integer> vstup), ktorá vráti zobrazenie/mapu/dictionary, ktoré zobrazuje každé zložené číslo zo vstupu (teda nie prvočíslo) na počet jeho deliteľov. Zložené číslo je číslo > 1, ktoré nie je prvočíslo, napr. 12 je zložené číslo, 13 nie je.

public static Map<Integer, Integer> pocetDelitelov(List<Integer> vstup)  
Príklad: ak je na vstupe zoznam 0..99, potom výsledok vyzerá takto: {4=1, 6=2, 8=2, 9=1, 10=2, 12=4,   
14=2, 15=2, 16=3, 18=4, 20=4, 21=2, 22=2, 24=6, 25=1, 26=2, 27=2, 28=4, 30=6, 32=4, 33=2, 34=2,   
35=2, 36=7, 38=2, 39=2, 40=6, 42=6, 44=4, 45=4, 46=2, 48=8, 49=1, 50=4, 51=2, 52=4, 54=6, 55=2,   
56=6, 57=2, 58=2, 60=10, 62=2, 63=4, 64=5, 65=2, 66=6, 68=4, 69=2, 70=6, 72=10, 74=2, 75=4, 76=4,   
77=2, 78=6, 80=8, 81=3, 82=2, 84=10, 85=2, 86=2, 87=2, 88=6, 90=10, 91=2, 92=4, 93=2, 94=2, 95=2,   
96=10, 98=4, 99=4}

* **[2 body]** definujte metódu Map<Integer, Integer> rozklad(Integer vstup), ktorá pre vstupné číslo vyrobí zobrazenie/mapu/dictionary, ktoré zobrazuje všetky prvočíselné delitele vstupu na exponent, ktorý má tento deliteľ v prvočíselom rozklade.

public static Map<Integer, Integer> rozklad(Integer vstup)

Príklad: pre 967032 = 23 \* 33 \* 112 \* 37, mapa bude Map.of(2,3,  3,3,  11,2,  37,1) = {2=3, 3=3, 37=1, 11=2}.

Body: 6

Autor: Peter Borovanský