Midterm 2022 Meno a priezvisko: _____ ID:
test obsahuje 5 úloh za 5+6+4+6+5=26 bodov.
Čas: 2h min, začiatok 18:10, koniec 20:10.

1. Rekurzia - M1

Táto rekurzívna funkcia na výpočet hodnoty **foo(a,b)** potrebuje sčítať viaceré hodnoty **foo(i,j)** pre **i+j<a+b** (samozrejme okrem foo(a,b)). Všimnite si, že robí to po uhlopriečkach, kde argumenty spolu majú súčet **s**, pričom **s** sa postupne zväčšuje. Je ale napísaná tak neefektívne, že už nevypočíta ani hodnotu pre a=b=9. A tú hodnotu nevypočíta len preto, že už dávno **vypočítané hodnoty pre menšie (i,j) ako (a,b) si nepamätá**, a teda ich počíta mnohokrát a opakovane. Vašou úlohou je ju prepísať tak, aby počítala hodnoty funkcie **foo** efektívnejšie.

```
public static long foo(int a, int b) {
    if (a == 0 && b == 0) return 1;
    else {
        long sum = 0L;
        for(int s = 0; s < a+b; s++)
            for(int i = 0; i <= s; i++)
                sum += foo(i, s-i);              // i + (s-i) je vzdy s...
        return sum;
    }
}
```

Úlohy: V tejto triede **`Rekurzia`**

• **[1.5 bodu]** skúste použiť pole (možno dvojrozmerné) či inú dátovú štruktúru na memoizáciu (po slovensky, pamätanie si) výsledkov, ktoré ste už počítali, a prepíšte funkciu **foo** na novú, na efektívnejšiu verziu **public static long foo1(int a, int b)** tak, aby vypočítala hodnotu aj pre a=b=9. Inak..., nám to vyšlo 60822550204416000L, len pre vašu kontrolu. Toto sú hodnoty pôvodnej funkcie foo:

foo	0	1	2	3	4	5	6
0	**1**	**1**	3	12	60	360	2520
1	**1**	3	12	60	360	2520	20160
2	3	12	60	360	2520	20160	181440
3	12	60	360	2520	20160	181440	1814400
4	60	360	2520	20160	181440	1814400	19958400
5	360	2520	20160	181440	1814400	19958400	239500800
6	2520	20160	181440	1814400	19958400	239500800	3113510400

• **[1.5 bodu]** prepíšte funkciu **foo** na novú **public static long foo2(int a, int b)** tak, aby **nebola rekurzívna**. **Hint**: Rekuzia na implementáciu používa zásobník, preto očakávané riešenie bez rekurzie bude asi používať váš zásobník.

Všimnite si, že funkcia je symetrická, teda **foo(a,b) = foo(b,a)**. A nie len to! P**re a+b > 2** (to sú hodnoty, ktoré nie sú bold fontom) platia vzťahy: **foo(a,b) = (a+b+1)*foo(a-1,b)**, ak 0 < a, resp. **foo(a,b) = (a+b+1)*foo(a,b-1)**, ak 0 < b.

• **[1 bod]** Na základe vlastnosti prepíšte funkciu **foo** na ešte efektívnejšiu verziu **public static long foo3(int a, int b)**. Na výpočet foo(a,b) vám bude stačiť približne a+b operácií, preto zrejme nemusíte riešiť rekurziu ani memoizáciu.

• **[1 bod]** vami definované funkcie foo1, foo2, foo3 počítajú hodnotu typu long. Akú najväčšiu hodnotu tejto funkcie v type **long** (teda bez pretečenia) viete vyrátať?
Hint: to chce asi kalkulačku, ktorú nemáte. Preto dôležitá je táto tabuľka faktoriálov a fakt, že Long.MAX_VALUE je 9223372036854775807 (pre istotu napísané aj dole pod tabuľkou).

1+ Rekurzia - M1 Meno a priezvisko: _____ ID:

1!=1
2!=2
3!=6
4!=24
5!=120
6!=720
7!=5040
8!=40320
9!=362880
10!=3628800
11!=39916800
12!=479001600
13!=6227020800
14!=87178291200
15!=1307674368000
16!=20922789888000
17!=355687428096000
18!=6402373705728000
19!=121645100408832000
20!=2432902008176640000 ...
9223372036854775807 = Long.MAX_VALUE
21!=too long for long

RIEŠENIA TOHOTO PRÍKLADU PÍŠTE NA TENTO LIST

2. Stromy – M2 Meno a priezvisko: _____ ID:

Binárny generický strom je najjednoduchšie definovaný ako data-class, teda record v Jave17 takto:

```
public record Node<E>(Node<E> left, E value, Node<E> right) { ... }
```

Každý vrchol má teda hodnotu typu E, pointer na ľavý a pravý podstrom, ktoré môžu byť null. Príklad konštanty typu **Node<Integer>t**, ktorú neskôr použijeme v príkladoch, je:

```
Node<Integer> t = new Node<>(
        new Node<>(new Node<>(null, 2, null),4,new Node<>(null, 9, new Node<>(null, 10, null))),
        13,
        new Node<>(new Node<>(null, 22, null),27,null));
```

Keď si predstavíte strom nakreslený po úrovniach, tak na najvrchnejšej úrovni 0 je jediný vrchol, to je koreň stromu, za predpokladu, ak strom nie je prázdny. O úroveň nižšie už môžu byť 2 vrcholy, ale aj jeden, aj dokonca žiaden (napr. ak celý strom vyzerá new Node<>(null, 22, null)). Takže na rôznych úrovniach má strom rôzny počet vrcholov. Strom **Node<Integer>t** uvedený v príklade má na úrovni 0 jeden vrchol 13, na úrovni 1 dva vrcholy 4 a 27, na úrovni 2 má tri vrcholy 2,9,22, a na úrovni 3 len jeden vrchol 10. Takže neplatí, že čím hlbšie, tým viac !

Úlohy: V generickej triede **Node<E>** definujte **triedne metódy**:

- **[2 body] public Integer najviacVrcholov()**, ktorá vráti **počet vrcholov na úrovni s maximálnym počtom vrcholov**. V uvedenom príklade sú to 3 vrcholy,

- **[1 bod] public Integer maximalnaUroven()**, ktorá vráti index ktorejkoľvek úrovne, ktorá obsahuje maximálny počet vrcholov z predošlej časti. Je evidentné, že to nie je jednoznačné, a takých úrovní môže byť viac, preto vráťte index **ktorejkoľvek, čo má maximálny počet vrcholov** na tej úrovni. Úrovne indexujeme od 0, v nej je vždy len koreň stromu. V uvedenom príklade je maximálna úroveň 2, lebo tá obsahuje spomínané 3 vrcholy. Keďže obe sú to metódy triedy Node, je jasné, že strom nemôže byť prázdny, keď ich aplikujete.

- **[1.5 bodu] public <T> Node<T> map(Function<E,T> f)**, ktorá na každú hodnotu E value v celom strome typu Node<E> aplikuje funkciu Function<E,T> f. Výsledkom bude **nový strom iného typu** Node<T> (pôvodný bol Node<E> a ten zostane nezmenený). Vlastne mate urobiť kópiu stromu s tým, že hodnoty value premapujete funkciou f. **Hint**: pripomíname, že aplikácia funkcie nevyzerá matematicky f(value), ale f.apply(value).

Príklad, ktorý vám má zbehnúť na strome Node<Integer>**t** definovanom na príklade vyššie je t.map("*"::repeat), a tento výsledný strom má v koreni hodnotu "**********" - číslo 13 sa zmenilo na 13 hviezdičiek, to robí mapovaná funkcia x -> "*".repeat(x), v skratke zapísaná ako "*"::repeat. Keď si uvedomíte je to funkcia typu Function<Integer, String>, preto mapovanie takejto funkcie zmení strom Node<Integer> na Node<String>, ktorého hodnoty value už nie sú Integer, ale sú String.

- **[1.5 bodu] public Node<E> filter(Predicate<E> p)** , ktorá vráti nový strom rovnakého typu Node<E>, ktorý sa od pôvodného líši tak, že [boli odrezané/neboli prekopírované] všetky podstromy, ktorých hodnota vo vrchole E value **nespĺňa predikát p**, teda p.test(value) == false. To znamená, že ak vrchol nespĺňa podmienku, celý podstrom sa vo výsledku nenachádza, aj keď niekde v ňom môžu byť vrcholy, ktoré podmienku spĺňajú. V uvedenom príklade t.filter(x -> x % 2 > 0) vráti Node[left=null, value=13, right=Node[left=null, value=27, right=null]], ale t.filter(x -> x % 2 == 0) vráti null, lebo v koreni je nepárne číslo 13.

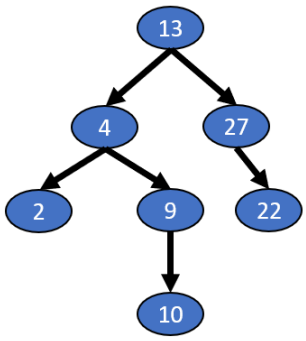
Hint: pripomíname, že aplikácia predikátu nevyzerá p(value), ale p.test(value).

Ilustrácia:

```
t.najviacVrcholov()// 3
t.maximalnaUroven()// 2
t.map("*"::repeat) // Node[left=Node[left=Node[left=null, value=**, right=null], value=****,
right=Node[left=null, value=*********, right=Node[left=null, value=**********, right=null]]],
value=*************, right=Node[left=Node[left=null, value=**********************, right=null],
value=*************************, right=null]]
t.filter(x -> x % 2 > 0)  // Node[left=null, value=13, right=Node[left=null, value=27, right=null]]
t.filter(x -> x % 2 == 0) // null
```

RIEŠENIA TOHOTO PRÍKLADU PÍŠTE NA TENTO, NASLEDUJÚCI LIST

2+ Stromy – M2 Meno a priezvisko: _____ ID:



3. Oprav ma – M3 Meno a priezvisko: _____ ID:

Úloha 1: **[1 bod]** Tento kód vyzerá ako binárne vyhľadávanie, ale v poli new int[]{1,3,4,5,6,7} nenájde ani 3, ani 7. Opravte kód tak, aby fungoval ako správne binárne vyhľadávanie, ktoré nájde každý prvok nachádzajúci sa v utriedenom poli.

```
public static boolean search(int[] arr, int element) {
   int bot = 0;
   int top = arr.length-1;
   while (bot < top) {
      int mid = (bot+top)/2;
      if (arr[mid] == element) return true;
      if (arr[mid] < element) bot = mid+1; else top = mid-1;
   }
   return false;
}
```

Úloha 2: **[1 bod]** Jednotková matica je štvorcová matica, ktorá má na uhlopriečke 1, všade mimo 0. Tento kód chce vytvoriť jednotkovú maticu pre vstupný rozmer matice

```
public static int[][] jednotkova(int n) {
   int[] vektor = new int[n];                  // je vynulovany vdaka Jave
   int[][] matica = new int[n][];
   IntStream.range(0,n).forEach(i -> {
      vektor[i] = 1;
      matica[i] = vektor;
   });
   return matica;
}
```

Máme za to, že kód nevytvorí jednotkovú maticu.
Koľko jednotiek obsahuje matica 4x4 pre volanie *jednotkova*(4)? Inak povedané, ak maticu vypíšem cez Arrays.*deepToString*(*jednotkova*(4)), koľko jednotiek vidím ?
Opravte metódu tak, aby skutočne generovala jednotkovú maticu, pre prípad n=4 to je [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]].

Úloha 3: **[1 bod]** Definujte premenné s1, s2, s3 tak, aby ste dostali zodpovedajúce výpisy:

```
System.out.println(s1 == s2);                    // false
System.out.println(s1.equalsIgnoreCase(s2));     // true
System.out.println(s2.equals(s1));               // false
System.out.println(s2.equals(s3));               // false
System.out.println(s3.equals(s2));               // Null Pointer Exception
```

Úloha 4: [1 bod] Definujte **A,B,X** tak, aby ste dostali zodpovedajúce výpisy:

```
System.out.println(List.of(new A(), new B()).size());                    // 2
System.out.println(new HashSet<>(Arrays.asList(new A(), new B())).size()); // 1

System.out.println(List.of(X.name, A.name, B.name));                     // [X, A, B]
System.out.println(List.of(new A().getName(), new B().getName()));       // [A, X]

System.out.println(new TreeSet<>(Arrays.asList(new A(), new B())).size());          // 2
System.out.println(new TreeSet<>(Arrays.asList(new A(), new B())).first().getName()); // A
System.out.println(new TreeSet<>(Arrays.asList(new A(), new B())).last().getName());  // X
```

RIEŠENIA TOHOTO PRÍKLADU PÍŠTE NA TENTO LIST

4. Relácia – M4 Meno a priezvisko: _____ ID:

V tomto príklade sa vyskytujú pojmy, ktoré ste počuli na Diskrétke, všetko znova vysvetlíme, pochopíte a naprogramujete. Nepreskakujte zadanie, len pre pár cudzích pojmov.

Relácia je daná definičným oborom, oborom hodnôt a vzťahmi medzi dvojicami prvkov z oboch množín. *Asi ste počuli, že to je podmnožina kartézskeho súčinu, ale tak zlé to nebude :)* Navrhli sme nasledujúcu reprezentáciu dvojíc (*z kartézskeho súčinu*) a relácie:

```
record Dvojica<A,B>(A a, B b) {
   @Override
    public String toString() { return a + "->" + b; }
}
public record Relacia<A,B>(Set<A> defObor, Set<B> oborHodnot, Set<Dvojica<A,B>> dvojice) { ... }
```

Programátorsky teda vidíte, že je to parametrizovaná trieda typmi A, B, a je určená definičným oborom Set<A>, oborom hodnôt Set<B>, a množinou dvojíc. Dvojica<A,B> dvojica, ktorej **prvá** zložka je dvojica.a() je z definičného oboru, teda typu A, a **druhá** zložka je dvojica.b() je z oboru hodnôt, teda typu B, je opäť parametrizovaná trieda typmi definičného oboru a oboru hodnôt a disponuje dvomi gettermi dvojica.a(), dvojica.b(). Idete naprogramovať základné vlastnosti relácie.

Úlohy [každá 1 bod]: V triede **Relacia** definujte triedne metódy:

• **public boolean jeZobrazenie()** - je test, ktorý platí, ak žiaden prvok definičného oboru sa nezobrazuje na dva rôzne prvky, obrázok 1 ilustruje zobrazenie, aj to, čo nie je zobrazenie.

• **public Map<A,B> zobrazenie()** - ak je relácia zobrazením (predošlý test), tak vráti reprezentáciu jeho dvojíc pomocou mapy, inak vráti null. Toto je kostra kódu, do ktorej dopíšte v časti **...** if (jeZobrazenie()) return **...** else return null;

• **public boolean jeInjektivne()** - každý prvok odboru hodnôt je obrazom **najviac jedného prvku** z definičného oboru. Obrázok 2 ilustruje injektívne, aj to, čo nie injektívne.

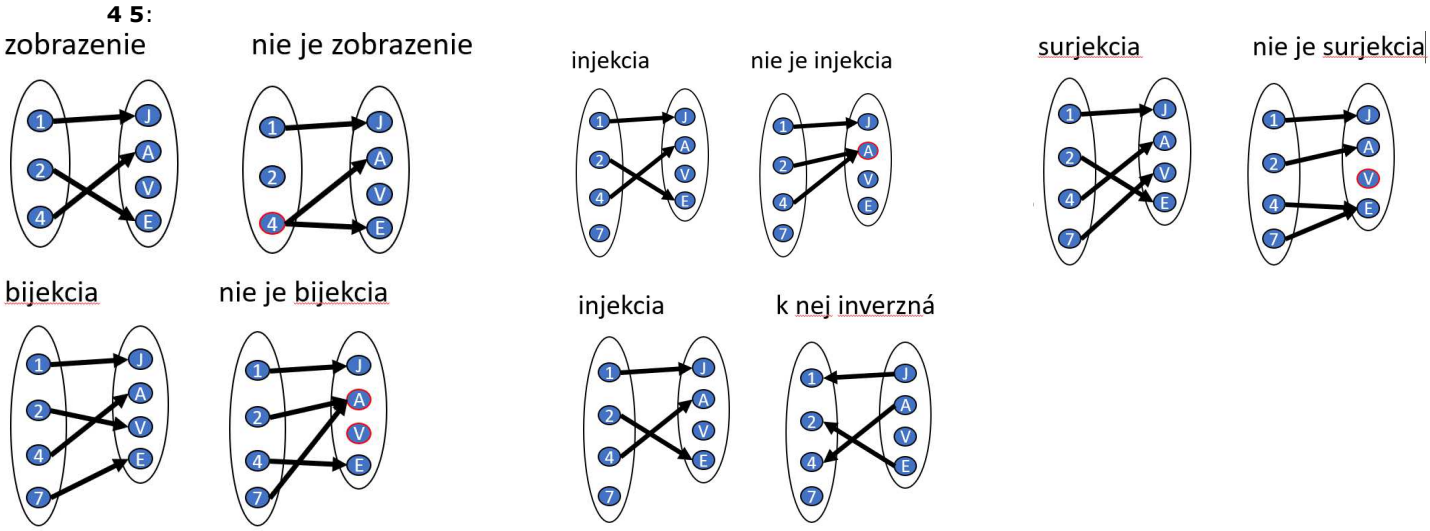
• **public boolean jeSurjektivne()** - každý prvok oboru hodnôt je obrazom **aspoň nejakého prvku** z definičného oboru. Obrázok 3 ilustruje surjektívne, aj to čo nie surjektívne.

• **public boolean jeBijektivne()** - každý prvok oboru hodnôt je obrazom **práve jedného prvku** z definičného oboru. Obrázok 4 ilustruje bijektívne, aj nie bijektívne. **Hint:** ak je možné, tak to vymyslite bez copy-paste.

• **public Relacia<B,A> inverzne()** - ak je zobrazenie injektívne, existuje k nemu **inverzné zobrazenie/relácia**. To znamená, že definičný obor a obor hodnôt sa vymenia, a dvojice sa otočia. Obrázok 5 ilustruje inverziu.

Dopisujte do kostry kódu v časti **...** if (jeInjektivne()) return **...** else return null;

Obrázky 1 2 3 4 5:



4+ Relácia – M4 Meno a priezvisko: _____ ID:

Legenda: Z – jeZobrazenie, I – jeInjektívne, S – jeSurjektívne, B - jeBijektívne

obrázok1_L Relacia[defO=[4, 2, 1], oborH=[V, E, J, A], dvojice=[4->A, 1->J, 2->E]] Z: true, I: true, S: false, B: false
obrázok1_R Relacia[defO=[4, 2, 1], oborH=[V, E, J, A], dvojice=[4->A, 1->J, 4->E]] Z: false, I: false, S: false, B: false
obrázok2_L Relacia[defO=[4, 2, 1, 7], oborH=[V, E, J, A], dvojice=[4->A, 1->J, 2->E]] Z: true, I: true, S: false, B: false
obrázok2_R Relacia[defO=[4, 2, 1, 7], oborH=[V, E, J, A], dvojice=[4->A, 1->J, 2->A]] Z: true, I: false, S: false, B: false
obrázok3_L Relacia[defO=[4, 2, 1, 7], oborH=[V, E, J, A], dvojice=[4->A, 2->E, 1->J, 7->V]] Z: true, I: true, S: true, B: true
obrázok3_R Relacia[defO=[4, 2, 1, 7], oborH=[V, E, J, A], dvojice=[7->E, 4->E, 1->J, 2->A]] Z: true, I: false, S: true, B: false
obrázok4_L Relacia[defO=[4, 2, 1, 7], oborH=[V, E, J, A], dvojice=[7->E, 4->A, 2->V, 1->J]] Z: true, I: true, S: true, B: true
obrázok4_R Relacia[defO=[4, 2, 1, 7], oborH=[V, E, J, A], dvojice=[7->A, 4->E, 1->J, 2->A]] Z: true, I: false, S: true, B: false
obrázok5_L Relacia[defO=[4, 2, 1, 7], oborH=[V, E, J, A], dvojice=[4->A, 1->J, 2->E]] Z: true, I: true, S: false, B: false
obrázok5_R inverzne = Relacia[defObor=[A, V, E, J], oborHodnot=[1, 7, 4, 2], dvojice=[A->4, J->1, E->2]]

RIEŠENIA TOHOTO PRÍKLADU PÍŠTE NA TENTO LIST

5. Streamy – M5 Meno a priezvisko: _____ ID:

Odporúčame použite StreamAPI, dostanete tak najkompaktnejšie riešenia.

Úlohy [každá 1bod]:

- definujte metódu **public static IntStream jednotkovaMatica(int n)**, ktorá pre vstupné **n** vyrobí IntStream s **n*n** prvkami, ktorý keby ste nakrájali po **n** prvkoch a dali pod seba, tak dostanete jednotkovú maticu (jednotková matica je vysvetlená v 3.príklade).

  Príklad: jednotkovaMatica(int n) vyrobí IntStream obsahujúci 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1. Medzery sú v ňom umiestnené pre zvýraznenie riadkov jednotkovej matice 4x4. Váš kód musí fungovať pre nezáporné n.

- definujte metódu **public static IntStream cifSum9(IntStream vstup)**, ktorá prefiltruje čísla vstupného streamu vstup, a vo výslednom streame **nechá len tie**, ktorých **ciferný súčet je deliteľný 9**. Môžete predpokladať, že vstup obsahuje len nezáporné čísla.

- definujte metódu **public static IntStream cifry1_9(IntStream vstup)**, ktorá prefiltruje čísla vstupného streamu vstup, a vo výslednom streame **nechá len tie**, ktoré obsahujú všetky cifry 1..9, každú práve raz. Na poradí cifier nezáleží. Príklad: 987654321 zostane, 112345 vypadne, 1023456789 vypadne.

- dokonalé číslo je číslo, ktorého súčet vlastných deliteľov sa rovná číslu samotnému. Napríklad, 6 je dokonalé, lebo jeho delitele sú 1+2+3=6. Pripomíname, že 6 nie je vlastným deliteľom čísla 6. Podobne 28 je ďalšie dokonalé číslo, lebo súčet jeho vlastných deliteľov je 28=1+2+4+7+14.

  Definujte static **IntPredicate *dokonale*** = ..., ktorý testuje vlastnosť dokonalého čísla.

- spriatelené čísla sú **dvojice rôznych** čísel (a,b) také, že súčet deliteľov a je b, a súčet deliteľov b je a. Príklad, delitele 220 sú 1+2+4+5+10+11+20+22+44+55+110=284, a delitele 284 sú 1+2+4+71+142=220. Takže (220, 284) sú takto spriatelené čísla. Číslo môže byť spriatelené s najviac jedným iným číslom. Ako ho nájsť ? Je to predsa súčet deliteľov čísla. A ako zistím, či je x číslo spriatelené ? Súčet deliteľov x má súčet deliteľov, ktorý musí byť x. Keďže súčet deliteľov už asi máte naprogramovaný z predošlej úlohy, tak definujte metódu **public static IntStream spriatelene(IntStream vstup)**, ktorá prefiltruje vstupný stream a nechá len čísla, ktoré sú spriatelené s nejakým iným, teda sú jedno z dvojice (a,b) spriatelených čísel. Z čísel 1..30000 sú spriatelené tieto:

```
spriatelene(IntStream.range(0,30_000)) 220, 284, 1184, 1210, 2620, 2924, 5020, 5564, 6232,
6368, 10744, 10856, 12285, 14595, 17296, 18416.
```

RIEŠENIA TOHOTO PRÍKLADU PÍŠTE NA TENTO LIST