Započetí testu	Pátek, 12. červen 2020, 09.37
Stav	Dokončeno
Dokončení testu	Pátek, 12. červen 2020, 12.33
Délka pokusu	2 hodin 56 min.
Známka	Dosud nehodnoceno

Úloha **1**

Hotovo

Počet bodů z 10,00

Profesor Hammerstein definoval predikat **setrid/2** takto:

```
% setrid(+Xs,-Ys) :- Ys je seznam přirozených čísel ze seznamu Xs setříděný vzestupně
setrid(Xs,Ys) :- append(A,[H1,H2|B],Xs), H1 > H2, !, append(A,[H2,H1|B],Xs1),
setrid(Xs1,Ys).
```

zapomněl však na klauzuli, která definuje bázi rekurze.

(a) Doplňte jednu (opravdu jen jednu) chybějící klauzuli za uvedené pravidlo tak, aby výsledná procedura korektně setřídila vstupní seznam přirozených čísel.
Na výstupu bychom měli obdržet jen jediné řešení.

(b) Doplňte jednu (opravdu jen jednu) chybějící klauzuli před uvedené pravidlo tak, aby výsledná procedura korektně setřídila vstupní seznam přirozených čísel.
Na výstupu bychom měli obdržet jen jediné řešení.

(c) V definici pravidla je použit řez (!). Jde o zelený (nemění deklarativní význam) či červený řez (mění d.v.) ? Vysvětlete!
Obsahuje některá z vašich klauzulí, (doplněná v(a) nebo (b)) zelený či červený řez?

(d) Jaký známý třídící algoritmus výše uvedený kód implementuje? Pokud neznáte název, můžete alespoň slovně popsat, jak **setrid/2** funguje.

(e) VOLITELNE: Lze u procedury **setrid/2** obrátit směr výpočtu?

```
% setrid(-Xs,+Ys) :- Xs je seznam přirozených čísel ze seznamu Ys setříděný vzestupně
```

Pokud ne, šel by kód jednoduše upravit tak, aby se výsledný predikát (pojmenovaný třeba **setrid2/2**) dal korektně volat oběma způsoby?

```
% Excercise 1 - Bastian Lukas

% setrid(+Xs,-Ys) :- Ys je seznam přirozených čísel ze seznamu Xs setříděný vzestupně
setrid(Xs,Ys) :- append(A,[H1,H2|B],Xs), H1 > H2, !, append(A,[H2,H1|B],Xs1), setrid(Xs1,Ys).

% a) doplnit za:
setrid(X, X) :- !.

% b) doplnit pred:

% missing

% c)
% Jde o cerveny rez, ktery po usporadani zajisti, ze se znovu nebude brat mensi prefix seznamu nez A a A bude uz setrizene

% d)
% melo by jit o bubble sort - postupne se swapuji prvky, ktere nejsou ve spravnem poradi
```

Úloha **2**

Hotovo

Počet bodů z 10,00

Do země Mobilia, v níž je každý občan vybaven chytrým telefonem, přicestoval *Cestovatel*, nakažený virovým onemocněním. Všichni ostatní byli přitom ještě zdraví.
Můžeme předpokládat, že virus se přenese z jedné osoby na druhou, pokud spolu strávili ve vzdálenosti menší než $2m$ alespoň čas K, kde K ja známá kritická hodnota.
Díky chytrým telefonům máme pro každého občana Mobilie seznam záznamů jeho kontaktů, kde každý takový záznam pro osobu A obsahuje

- identifikaci osoby B, která se k němu přiblížila do vzdálenosti $< 2m$
- čas setkání
- a délku setkání

Cílem je sestavit program, který na základě takových záznamů vrátí seznam infikovaných osob.

(a) V jazyce Prolog popište datovou strukturu pro reprezentaci jednoho záznamu kontaktu občana Mobilie popsaného výše.

(b) V jazyce Prolog navrhněte reprezentaci položek *VstupníhoSeznamu,* přičemž každá položka bude obsahovat indentifikaci občana Mobilie a seznam záznamů jeho kontaktů.

(c) Sestavte predikát **inf/4**, který obdrží

- *VstupníSeznam*
- identifikaci *Cestovatele*
- kritickou hodnotu *K*
- a vrátí seznam infikovaných

U každého pomocného predikátu prosím v poznámce popište jeho význam.

Volitelné: výstupní seznam můžete uspořádat dle délky kontaktu s infikovanými do nerostoucí posloupnosti.

(d) Odhadněte časovou složitost vašeho řešení.

(e) Je některý z vašich predikátů *koncově rekurzivní* ? Pokud ano, vysvětlete, jaký to má význam. Pokud ne , dal by se některý takto upravit?

```
% Excercise 2 - Bastian Lukas

% a)
% record bude array vzdy o 3 polozkach [jmeno, cas, doba]

% b)
% vstupni seznam bude array dvojic s polozkoku jmeno a seznam zaznamu
% (jmeno, [arrayOfRecords])

% c) inf/4
inf(VstupniSeznam, Cestovatel, Krit, Result) :- najdiKontakty(VstupniSeznam, Cestovatel, Kontakty)
% Najde kontakty pro nejakeho cestovatele - infikovaneho
najdiKontakty([V|StupniSeznam], Cestovatel, Kontakty) :- V is [Jmeno, Records], (
                                                    (
                                                        Jmeno = Cestovatel,
                                                        Kontakty is Records
                                                    );
                                                        najdiKontakty(StupniSeznam, Cestovatel, Kontakty)
                                                )
% Dale bych podle krit hodnoty a hodnot v recordech porovnaval, kdo je infikovany a dale tranzitivne hledal, kdo je nakazeny od




% d)


% e)
% Vyznam je hlavne ve spotrebe pameti - pokud se rekurze vola az jako posledni tak vim, ze mi staci stejny blok pameti na vypoc
```

Úloha **3**

Hotovo

Počet bodů z 10,00

Rozdělte zadaný *binární vyhledávací strom T* na $n+1$ binárních vyhledávacích stromů $T_0, \ldots, T_n$

- podle zadaných vstupních hodnot k_i, $1 \le i \le n$
- tak, že ve stromě T_i jsou hodnoty x, $k_i \le x < k_{i+1}$, pokud jsou nerovnosti aplikovatelné.

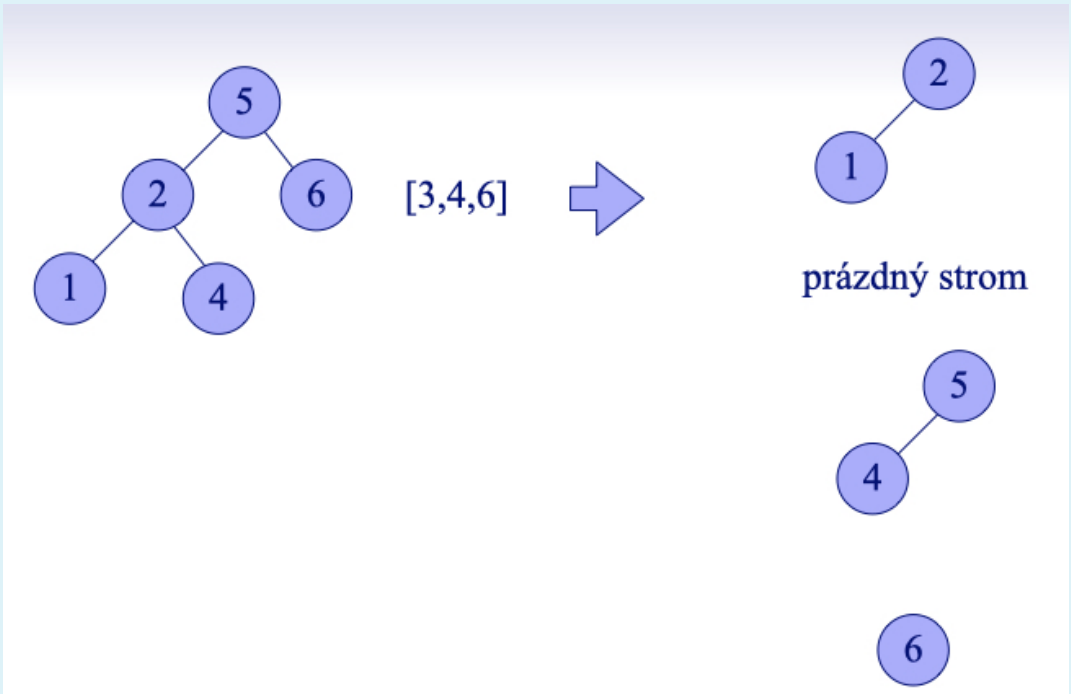
Snažte se o efektivitu, celé podstromy patřící do jednoho pruhu zpracujte najednou.

(a) Definujte datový typ pro reprezentaci binárních vyhledávacích stromů. Snažte se o co nejobecnější definici.

(b) Definujte *typovou signaturu* funkce **pruhy**, včetně typových tříd.

(c) Funkci **pruhy** definujte. Budete-li používat pomocné funkce, u každé popište její význam.

(d) Pokuste se stručně zdůvodnit korektnost vaší defnice.




```
-- Excercise 3 - Bastian Lukas


-- a)
-- Binarni Vyhledavaci Strom je bud:
-- End ~ konec ~ ekvivalent null
-
- Node Left Value Right ~ kdy jde o nejaky vrchol ve stromu, ktery ma levy podstrom, svou hodnotu a pravy podstrom. Podrstror
data BVS a = End | Node (BVS a) a (BVS a)
 deriving (Show)

-- b) && c)
-- pruhy (Node (Node (Node End 1 End) 2 (Node End 4 End)) 5 (Node End 6 End)) [3, 4, 6]
pruhy :: BVS Int -> [Int] -> [BVS Int]
pruhy bvs seznam = internalpruhy bvs (0:seznam)

internalpruhy :: Ord a => BVS a -> [a] -> [BVS a]
internalpruhy bvs (a:[]) = [getBVSLast bvs a]
internalpruhy bvs (a:b:rest) = ((getBVS bvs a b):internalpruhy bvs (b:rest))

getBVS :: Ord a => BVS a -> a -> a -> BVS a
getBVS (End) _ _ = (End)
getBVS (Node l v r) a b
    | a <= v && v < b = (Node leftSub v rightSub)
    | otherwise = newBest
  where
    leftSub = getBVS l a b
    rightSub = getBVS r a b
    newBest = getBest leftSub rightSub

-
- za zadani by se asi dala nejak odvodit horni hranice a pouzit getBVS nebo proste passovat predicate, podle ktereho se to vyhod
getBVSLast :: Ord a => BVS a -> a -> BVS a
getBVSLast (End) _ = (End)
getBVSLast (Node l v r) a
    | a <= v = (Node leftSub v rightSub)
    | otherwise = newBest
  where
```

```
    leftSub = getBVSLast l a
    rightSub = getBVSLast r a
    newBest = getBest leftSub rightSub

-- spojovani dvou vetvy vypoctu
getBest :: BVS a -> BVS a -> BVS a
getBest (End) (End) = (End)
getBest (End) r = r
getBest l (End) = l
getBest l r = getLeftMostInRight l r

-
- zpusob napojeni dvou podstromu, pokud jsou v nich validni hodnoty - i nejlevejsi hodnota praveho podstromu by mela byt vet
getLeftMostInRight :: BVS a -> BVS a -> BVS a
getLeftMostInRight l (Node (End) v r) = (Node l v r)
getLeftMostInRight l (Node rl v r) = (Node (getLeftMostInRight l rl) v r)

-- d) korektnost
-- postupne beru okenka/intervaly, pricemz ja jsem pro zacatek pridal okenko 0 az prvni uzivatelem zadane cislo
-
- pokud se v prubehu vypoctu stane, ze pro nejaky (Node l v r) pro v podminka neplati, ale pro neco z l nebo r ano, tak se vytvor
-
- samotne vyhodnoceni je o kontrole podminky pro v ~ value, jestli spada do okenka a pokud ano, tak si ji necham a rekurzivne v
-- pokud ne, tak se kontroluji podstromy, ktere se vsam musi spojit do validniho stromu (popsano vyse)
```

Úloha **4**

Hotovo

Počet bodů z 10,00

Definujte funkce **rle** a **rld**, které realizují *run-length encoding* a *decoding*. Funkce

```
rle :: Eq a => [a] -> [Either a (a,Int)]
```

zakóduje co nejdelší úseky stejných prvků ve vstupním seznamu do dvojice (prvek, počet) typu `Either` s datovým konstruktorem `Right`.

Pokud je prvek v úseku sám, kóduje se pouze prvek vnořený do typu `Either` s datovým konstruktorem `Left`.

Příklad:

```
> rle ”abbcccda”
[Left 'a', Right ('b',2), Right ('c',3), Left 'd', Left 'a']
```

(a) Definujte funkci **rle** s využitím rekurze, ale bez použití stručných seznamů či funkcí vyšších řádů (funkce s funkcionálními parametry).

(b) Definujte funkci **rle** bez explicitního využití rekurze, ale za použití stručných seznamů či funkcí vyšších řádů.

(c) Definujte typovou signaturu funkce **rld**, která realizuje dekompresi, tj. převod ze seznamu úseků na původní seznam prvků.

(d) Definujte funkci **rld**. Použijte přitom funkci `map` či `concat`.

(e) Bude některá z funkcí fungovat i na nekonečných seznamech? Proč ano nebo proč ne?

```
-- Excercise 4 - Bastian Lukas

-- a) rle recursion
rle :: Eq a => [a] -> [Either a (a,Int)]
rle [] = []
rle (x:xs)
    | length == 1 = ((Left x) : rle newXs)
    | otherwise = ((Right (x, length)) : rle newXs)
  where
    length = isSame x (x:xs)
    newXs = drop length (x:xs)
-- not sure I can use takewhile - this is the eqvivalent
isSame :: Eq a => a -> [a] -> Int
isSame x ([]) = 0
isSame x (y:ys)
  | x == y = 1 + isSame x ys
  | otherwise = 0

-- b) rlef - rle folding
-- nejdrive iterace pres cely array a pocitani toho, kolikrat se co objevilo za sebou
-- pote namapovat vsechno do spravne podoby ((char, 1) => Left x; (char,i), i > 1 => Right (char, i))
rlef:: Eq a => [a] -> [Either a (a,Int)]
rlef array = map countToEither (foldl count [] array)

count:: Eq a => [(a,Int)] -> a -> [(a,Int)]
count x c
    | x == [] = [(c, 1)] --start
    | otherwise = (take (len - 1) x ) ++ (incOrAdd (last x) c) -
- vzit vsechno krome posledni a k poslednimu zkusit pridat dalsi znak
  where
    len = length x

incOrAdd:: Eq a => (a,Int) -> a -> [(a,Int)]
incOrAdd (c, n) newC
    | c == newC = [(c,n+1)] -- pokud je znak stejny, zvysit pocet
    | otherwise = [(c,n), (newC, 1)] -- pokud ne, tak vratit posledni jak byl a pridat novy znak s vyskytem 1

-- pro namapovani poctu vyskytu na spravnou formu
countToEither:: (a, Int) -> Either a (a,Int)
countToEither (c, n)
  | n == 1 = Left c
  | otherwise = Right (c, n)

-- c) && d) rld - types + function
rld :: Eq a => [Either a (a,Int)] -> [a]
rld = concat . map getArrayFromEither
-- this generates the required number of occurances based on the either
```

```
getArrayFromEither :: Either a (a,Int) -> [a]
getArrayFromEither (Left x) = [x]
getArrayFromEither (Right (x, i)) = take i (repeat x)

-- e)
-
- jak rle a rlef potrebuji konec seznamu pro zastaveni se vydani vysledku, ale rld mapuje a spojuje postupne co prave vypocitalo t
bude fungovat => bude fungovat na nekonecnych
```

◄ Požadavky na termín ústní zkoušky (pondělí 15.6. od 9:00)

Přejít na...