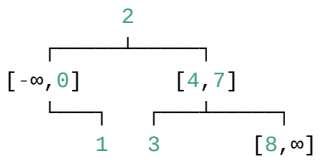
Hrobečkové stromy

Termín: 27. listopadu 2019 23.59; řešení odevzdávejte opět do odevzdávárny.

V této domácí úloze si budete hrát s binárními vyhledávacími stromy a datovými typy. Pokuste se o co nejelegantnější řešení. Snáze pak odhalíte chyby a spokojeně si odnesete 2 body.



Obrázek 1: Takto vypadá hrobečkový strom.

Hrobečkový strom slouží k uchovávání a vyhledávání celých čísel (značíme $\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$). Jako hodnoty si drží buď *živá* čísla, nebo *hrobečky* – intervaly *mrtvých* čísel, která nejsou *živá* a slouží k vyplnění stromu. Hrobeček je dán dvojicí čísel a obsahuje všechna čísla od jeho levé po pravou hodnotu. Například hrobeček `Grave 0 0` obsahuje pouze číslo 0, hrobeček `Grave 6 9` obsahuje čísla $6, 7, 8$ a 9.

Pokud přidáme navíc k číslům kladné a záporné nekonečno, pak jsme schopni mít ve stromě všechny hodnoty! Například `Grave NegInf PosInf` v sobě pohřbívá všechna celá čísla. Nyní můžeme dosáhnout toho, že prostor mezi každými dvěma živými čísly v hrobečkovém stromě bude vyplněn příslušným hrobečkem. Při hledání čísla v korektním (viz dále) hrobečkovém stromě tak nebude nikdy třeba jít až k listům (`Empty`), ale skončíme, jakmile nalezneme odpovídající živou hodnotu, nebo hrobeček, ve kterém je číslo pohřbené.

Ukázkový strom z obrázku 1 má hodnoty uspořádané a je korektní. Jedná se tedy o variantu vyhledávacího stromu: pokud v něm budeme hledat např. číslo 1, pak začneme nahoře v kořeni v živé hodnotě 2 (v Haskellu `Alive 2`, na obrázku pouze `2`) a půjdeme doleva, protože $1 < 2$ a všechny hodnoty menší než hodnota v uzlu se nachází v jeho levém podstromě. Protože $1 > 0$, půjdeme v uzlu obsahujícím `Grave NegInf 0` doprava, kde už hledanou 1 nalezneme – `Alive 1`. Přitom listů a více než poloviny stromu jsme se vůbec nedotkli. Obdobně, pokud hledáme například hodnotu 42, začneme znovu v kořeni stromu a protože $42 > 2$, víme, že se tato hodnota musí nacházet v pravém podstromu kořene, kde najdeme uzel s hodnotou `Grave 4 7`. Jelikož $42 > 7$, půjdeme znovu doprava, kde narazíme na uzel s hodnotou `Grave 8 PosInf`. Jelikož hodnota 42 spadá do intervalu $[8, \infty]$, zjistili jsme, že se hodnota 42 v tomto stromě nenachází (jako živá).

Použité datové typy

Pro reprezentaci hrobečkových stromů a hodnot v nich budeme používat následující datové typy. Jejich definici najdete i v kostře řešení. Tyto typy nesmíte modifikovat.

```
data ZEx = NegInf | Z Integer | PosInf deriving (Eq, Ord, Show)
```

Datový typ `ZEx` popisuje celá čísla rozšířená o záporné a kladné nekonečno ($\mathbb{Z}^\infty = \mathbb{Z} \cup \{-\infty, \infty\}$). V kostře mají definovanou instanci `Num` (tedy s nimi lze pracovat jako s normálními čísly, například je sčítat, násobit, a pokud napíšete číselný literál, například `42`, Haskell z něj automaticky dokáže vytvořit hodnotu `Z 42`, pokud jeho typ má být `ZEx`). Tato čísla budeme používat při popisu intervalů.

```
data Hanged = Alive Integer | Grave ZEx ZEx deriving (Eq, Show)
```

Datový typ `Hanged` popisuje živá čísla (hodnoty `Alive x`) nebo intervaly mrtvých čísel (hrobečky, `Grave x y`). Intervaly jsou uzavřené, tedy reprezentují rozsah celých čísel obsahující i meze intervalu.

```
data BinTree a = Empty | Node a (BinTree a) (BinTree a) deriving (Eq, Show)
type HTree = BinTree Hanged
```

Binární stromy, jak je znáte ze cvičení, reprezentujeme pomocí datového typu `BinTree`. Hrobečkový strom (`HTree`) pak vznikne použitím hodnot typu `Hanged` v uzlech `BinTree`.

Korektní hrobečkový strom

Korektní hrobečkový strom má v sobě uložena všechna celá čísla (ne nutně všechna živá). Uložené hodnoty jsou navíc uspořádané zleva doprava. Přesněji, hrobečkový strom je korektní právě tehdy, když splňuje následující čtyři podmínky.

1. Levá hodnota každého hrobu je *menší nebo rovna* jeho pravé hodnotě, tedy pro korektní `Grave a b` platí `a <= b`.
2. Pokud hrob obsahuje nekonečno, pak je jeho levá hodnota *ostře menší* než jeho pravá hodnota. Tedy hrob nemůže reprezentovat samotné nekonečno. Např. nechceme `Grave NegInf NegInf`.
3. Posloupnost čísel při průchodu zleva doprava (viz níže) *neklesá*.
4. Každá hodnota ze $\mathbb{Z}^\infty = \mathbb{Z} \cup \{-\infty, \infty\}$ se nachází ve stromě právě jednou, a to buď jako živá, nebo v hrobě.

Z podmínek vyplývá, že posloupnost čísel při průchodu zleva doprava roste s výjimkou případů, kdy se objevují jednoprvkové hroby. To je jediné místo, kde se mohou čísla v zápisu korektního hrobečkového stromu opakovat.

Průchod zleva doprava

Průchodem binárního stromu `Node v l r` zleva doprava (`inorder`) rozumíme posloupnost hodnot, v níž jsou nejdříve hodnoty z levého podstromu (`inorder l`), potom teprve hodnota `v` a nakonec hodnoty z pravého podstromu (`inorder r`). Funkce `inorder` tak prochází celý strom zleva doprava a přitom vkládá jeho prvky do seznamu.

```
inorder :: BinTree a -> [a]
inorder  Empty       = []
inorder (Node v l r) = inorder l
                     ++ [v]
                     ++ inorder r
```

```
> pprint inorderExample
        3
      ┌─┴─┐
      1   [4,∞]
    ┌─┴─┐
[-∞,0] 2
> inorder inorderExample
[Grave NegInf 0,Alive 1,Alive 2,Alive 3,Grave 4 PosInf]
```

Funkce k implementaci

Následuje popis funkcí, které máte implementovat. V příkladech se držíme zavedeného formátu, v němž řádky začínající znakem „>“ označují vstup interpretu a ostatní řádky jeho výstup. Pro lepší čitelnost používáme funkci `pprint`, kterou máte schovanou v kostře spolu se stromy ze zadání.

- **`cmpHanged :: ZEx -> Hanged -> Ordering`**

  Standardní typ `Ordering` má tři hodnoty: `LT`, `EQ`, `GT` a slouží k porovnání dvou hodnot např. pomocí standardní funkce `compare :: Ord a => a -> a -> Ordering`. Výsledek `compare x y` nám říká, zda je `x` menší, rovno, nebo větší než `y`.

  Pro dané číslo `x` a `Hanged` hodnotu `h` rozhodněte, zda bude hledaná hodnota `x` v levém podstromě uzlu obsahujícího `h` (`LT`), zde (`EQ`) nebo v pravém podstromě tohoto uzlu (`GT`). Pro *živá* čísla půjde o jednoduché porovnání. U hrobů se rozhodněte na základě jejich krajních hodnot, o kterých můžete předpokládat, že levá je *menší nebo rovna* pravé. Krajní hodnoty považujeme za *mrtvé* a do hrobů patří. Tato funkce se vám bude hodit při průchodu stromem, když budete potřebovat porovnávat hledanou hodnotu s hodnotou v uzlu (`Node`).

```
> cmpHanged 1 (Alive 2)
LT
> cmpHanged 0 (Grave (-1) 1)
EQ
> cmpHanged 1 (Alive 1)
EQ
```

```
> cmpHanged 0 (Grave NegInf 0)
EQ
> cmpHanged PosInf (Grave NegInf 0)
GT
> cmpHanged NegInf (Grave NegInf 0)
EQ
```

- **`isAlive :: Integer -> HTree -> Bool`**

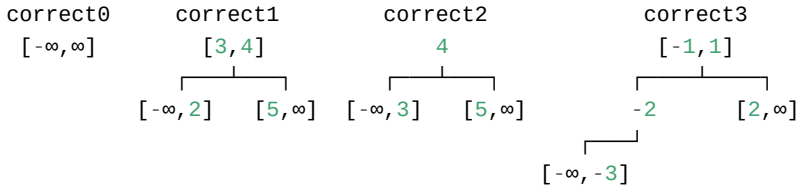
  Rozhodněte, zda je číslo v korektním hrobečkovém stromě naživu. Díky uspořádání hodnot ve stromě zleva doprava nemusíte ani procházet celý strom, ale můžete jít přímo k hodnotě. To proto, že všechny hodnoty menší než hodnota v daném uzlu se nachází v jeho levém podstromě, pro větší pak v pravém. V úvodní ukázce (obrázek 1) jsme mohli při hledání 1 projít pouze uzly obsahující hodnoty 2, $[-\infty, 0]$ a 1 právě díky této vlastnosti.

  Pokud chcete, aby vaše implementace této funkce byla hodnocena plným počtem bodů, nesmíte procházet celý strom, ale musíte jen následovat jednu větev stromu, na které se daná hodnota nachází. Kam se vydat z daného uzlu, lze vždy určit porovnáním hodnoty v uzlu s hledanou hodnotou. Pro ty z vás, kteří znají pojem časové složitosti, to znamená, že časová složitost funkce `isAlive` je lineární vzhledem k délce nejdelší větve stromu. Speciálně to tedy znamená, že v implementaci `isAlive` byste neměli použít funkci `inorder`. Splnění této podmínky bude kontrolovat cvičící (automatické testy vám přidělí jen část bodů na základě toho, zda funkce vrací správné výsledky).

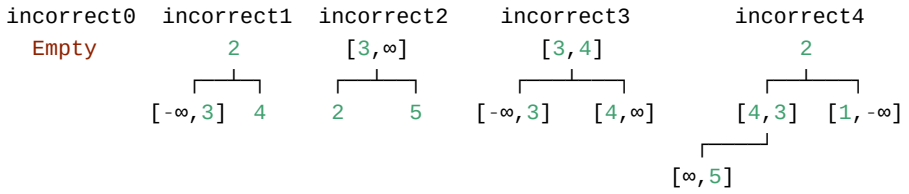
```
> isAlive 0 correct0
False
> isAlive 4 correct1
False
> isAlive 4 correct2
True
> isAlive 4 correct3
False
> isAlive (-2) correct3
True
> isAlive 1 exampleCorrect -- úvodní strom
True
```

- **`isCorrect :: BinTree Hanged -> Bool`**

  Napište funkci, která pro libovolný hrobečkový strom rozhodne, zda je korektní. Pokud si nevíte rady, zapátrejte ve sbírce příkladů nebo na internetu. Rozmyslete si však, čím se hrobečkové stromy odlišují od ostatních vyhledávacích stromů!



Obrázek 2: Příklady korektních hrobečkových stromů.



Obrázek 3: Příklady nekorektních hrobečkových stromů.

Můžete si úkol výrazně zjednodušit funkcí `inorder` i vlastními pomocnými funkcemi, ty však nejsou vyžadovány. Například by se vám mohla hodit funkce `isSuccHanged :: Hanged -> Hanged -> Bool`.

- **`livingMembers :: BinTree Hanged -> [Integer]`**

  Zachraňte všechny přeživší! Tedy vyberte všechna čísla `i`, která jsou ve stromě jako `Alive i`. Na pořadí ve výsledném seznamu nezáleží a vstupní strom nemusí být korektní (a může tedy obsahovat i duplicitní živá čísla, která se musí ve výsledku objevit tolikrát, kolikrát jsou ve vstupním stromě).

```
> pprint incorrect5
          2
    ┌─────┴─────┐
  [5,4]       [0,-∞]
  ┌─┴─┐       ┌─┴─┐
[∞,6]   3 1

> pprint incorrect6
  2
┌─┴─┐
2   3
    ┌┘
    3

> pprint treeOfDeath
                [12,12]
      ┌──────────┴──────────┐
    [1,10]                 17
    ┌─┴─┐              ┌────┴────┐
[-∞,0]   11        [13,16]   [18,∞]
```

```
> livingMembers incorrect5
[3,2,1]

> livingMembers incorrect6
[2, 2, 3, 3]

> livingMembers treeOfDeath
[11,17]
```

- **`moveBy :: HTree -> Integer -> HTree`**

  Věšet čísla je náročné, a proto si radši vytvoříme další stromečky pomocí posunů. Projděte celý strom a ke každému číslu přičtěte danou hodnotu. Můžete využít instance `Num` pro `ZEx`, ve které přičítání obyčejných čísel k nekonečnům nemá žádný vliv a nic nepokazí (tedy například $4 + \infty = \infty + 4 = \infty$). Můžete předpokládat, že vstupní strom je korektní. Výsledek musí být taktéž korektní hrobečkový strom lišící se pouze posunutými hodnotami, tedy musí mít stejný tvar jako vstupní strom.

```
> pprint threeSeven
          7
    ┌─────┴─────┐
    3         [8,∞]
  ┌─┴─┐
[-∞,2]  [4,6]
```

```
> pprint $ threeSeven `moveBy` (-7) -- moveBySeven
          0
    ┌─────┴─────┐
   -4         [1,∞]
  ┌─┴─┐
[-∞,-5]  [-3,-1]
```

  Pomoci vám může například funkce `binTreeMap` (tedy obdoba funkce `map` pro binární stromy), kterou však kostra neobsahuje a budete si ji případně muset naprogramovat sami. Takové čáry[1] se ale v tomto kurzu nepožadují ani netestují.

- **`mirror :: HTree -> HTree -> HTree`**

  Poslední zoufalý pokus, jak zachránit planetu sázením více stromů, bude vytvořit zrcadlové stromy. Nahraďte čísla v korektním stromě jim opačnými (tedy vynásobte je číslem -1) a upravte hrobečkový strom tak, aby byl opět korektní. Zároveň chceme, aby druhá aplikace funkce vrátila původní strom, tedy `mirror (mirror tree) == tree`.

```
> pprint reflected
        [-1,1]
      ┌──┴──┐
     -2     2
   ┌─┘      └─┐
[-∞,-3]     [3,∞]
```

```
> pprint $ mirror reflected
        [-1,1]
      ┌──┴──┐
     -2     2
   ┌─┘      └─┐
[-∞,-3]     [3,∞]
```

```
> pprint exampleCorrect
      2
  ┌───┴───┐
[-∞,0]  [4,7]
  └┐   ┌──┴──┐
   1 3      [8,∞]
```

```
> pprint $ mirror exampleCorrect
            -2
      ┌──────┴──────┐
   [-7,-4]        [0,∞]
  ┌──┴──┐        ┌──┘
[-∞,-8]  -3     -1
```

Poznámky a tipy

- Importovat smíte libovolné moduly z balíku `base`, pohodlně se bez nich ale obejdete.
- Neduplikujte kód! Snažte se vždy využít funkce, které jste již naprogramovali. Pokud to nejde přímo, ale přesto vidíte v řešení podobu, vytkněte podobnou část do pomocné funkce.
- Využijte poskytnuté instance typových tříd, zejména **`instance`** `Ord ZEx` a `Num ZEx`.
- Uveďte typy všech globálních funkcí, které definujete.

[1] Nebo dokonce instance `Functor` pro `BinTree` a `Hanged`, pokud si chcete opravdu vyzkoušet Haskell.

- Nestihnete-li implementovat některou funkci, můžete ji nechat `undefined`, zakomentovat či smazat. Nezapomeňte pak zakomentovat či smazat i její typovou signaturu.
- Nejste-li si jisti nějakou částí zadání, zeptejte se v diskusním fóru.
- Přebíráte-li kód odjinud, uveďte zdroj, jinak bude na vaši práci pohlíženo jako na plagiát.
- Nezapomeňte, že **opisování je zakázáno a bude postihováno podle disciplinárního řádu.**

Kostra

Kostra řešení obsahuje potřebné typy, hlavičky funkcí k implementaci a pomocné funkce (vykreslování stromů, instance `Num` pro `ZEx`, …). Kostra je komentovaná, žádné další povinnosti vám však z komentářů nevyplývají. **V kostře, za hlavičkami funkcí k implementaci, je opravdu hodně kódu navíc. Nic z toho však nemusíte číst**. Pokud nechcete, ani si ho tam nemusíte nechat. K řešení vám stačí definice typů a signatury zadaných funkcí z tohoto zadání. **Zadané typy nesmíte měnit.**

Směle můžete použít vše, co v kostře naleznete. Užitečnější funkce jsou výše a jednotlivé sekce jsou vizuálně oddělené. Podobna Danteho peklu, i kostra je postupně děsivější a je naprosto v pořádku, pokud sekcím dále za `instance Num ZEx` nebudete rozumět. Více se o nich dočtete dále.

Odevzdání

Tento domácí úkol se neodevzdává přes odpovědník, nýbrž přes odevzdávárny v Informačním systému. O tom, kterou odevzdávárnu máte použít, rozhoduje číslo vaší seminární skupiny. Do odevzdávárny vkládejte **jediný** soubor s příponou `.hs` obsahující vaši implementaci zadaných funkcí. Ty samozřejmě musí mít požadovanou signaturu a požadované jméno. **Všechny globální funkce musí mít typovou signaturu!** Odevzdaný soubor musí být přeložitelný překladačem GHC 8.6 na Aise nebo Nymfe.

```
$ module add ghc
$ ghci -Werror=missing-signatures hw08.hs
```

Pro zajištění funkčnosti je také třeba, aby byl odevzdaný soubor uložen v kódování UTF-8 (ve kterém je poskytnuta kostra).

V odevzdaném souboru **neuvádějte hlavičku `module`!** Pokud nevíte, o čem je řeč, vůbec to nevadí.

Vyhodnocení po nahrání souboru **není** okamžité; automatický testovací nástroj kontroluje soubory v odevzdávárně několikrát denně. Podle času odevzdání a vytížení vyhodnocovacího serveru může vyhodnocení trvat několik desítek minut. Po vyhodnocení se získané body a případný výpis neprošedších testů objeví v poznámkovém bloku. U nesprávně implementovaných funkcí se dozvíte příklad vstupu, na němž se váš výsledek neshoduje s očekávaným.

Máte **pět možností odevzdání**, započítává se nejlepší z nich. Další odevzdání provedete tak, že do odevzdávárny nahrajete novou verzi, jíž přepíšete odevzdaný soubor. Vzhledem k prodlevám při vyhodnocování neodkládejte práci na poslední chvíli, ať možnost vícenásobného odevzdání v případě potřeby vůbec stihnete využít. S blížícím se termínem uzavření odevzdáváren očekávejte větší (i několikahodinové) prodlevy.

S odevzdávárnou zacházejte s rozvahou, abyste nepřišli o možnosti odevzdání. I když nahrajete nové řešení ještě před zveřejněním výsledku v poznámkovém bloku, vyhodnocovací nástroj už může mít vaše dřívější odevzdání ve frontě. Z jeho pohledu tak došlo ke dvěma odevzdáním a vy si vyplýtváte jeden pokus. Také pokud odevzdáte více souborů naráz, můžete ztratit více odevzdání.

Hodnocení

Za funkčnost můžete od automatických testů obdržet **až 1,6 bodu** podle toho, které funkce se vám podařilo správně implementovat. Za částečně implementované funkce (např. nefunguje některý okrajový případ) žádné body nezískáte. Plnou funkčnost funkce `isAlive` budou dále hodnotit také vaši cvičící, kteří zkontrolují, jestli funkce neprochází celý strom. Za správnou funkci `isAlive` vám cvičící mohou udělit další **0,2** bodu.

Cvičící vám dále poskytnou zejména zpětnou vazbu na kód, ale také vám za úhlednost a pochopitelnost řešení mohou udělit další **0,2** bodu. Neočividné či zajímavé části řešení proto stručně komentujte v kódu.

V součtu tedy můžete za úlohu získat **až 2 body**.

Testování

Této části nemusíte rozumět, může se vám však hodit při vašem testování. Zejména kreslítka využívají kromě vstupu a výstupu i `newtype`, což je nad rámec předmětu. Kód naleznete v kostře a můžete si s ním dělat, co chcete, například ho smazat nebo **sdílet své testy**, samozřejmě **pouze pokud v nich neprozrazujete své řešení**.

Kreslítka

K dispozici vám je funkce `pprint`, která stromy vypíše pomocí Unicode znaků obdobně jako v příkladech tu. Pokud jste na unixovém operačním systému (macOS, GNU Linux), pak stačí spouštět GHCi v terminálu s podporou UTF-8, a pokud není správně nastaven, nastavit jazyk pomocí proměnné prostředí:

```
export LANG=en_US.UTF-8  # případně cs_CZ.UTF-8 nebo sk_SK.UTF-8
```

Pokud se vám nepodaří nastavit Unicode font v terminálu, nebo jste na OS bez uživatelské podpory, nezoufejte a použijte náhradní[2] horizontální kreslítko – `pprint (HorizontalT tree)`. Pro trvalou změnu na ASCII výpis upravte tuto instanci v kostře:

```
instance PointAtBox a => Pretty (BinTree a) where
    pprint t = case t of
        Empty -> putStrLn "Empty"
        branch -> pprint $ VerticalT {- změňte na HorizontalT -} branch
    pshow = pshow . VerticalT {- také tady -}
```

Stromečky

V kostře najdete sekci se stromečky, můžete si zde zasadit i své vlastní.

```
-- | First presented 'HTree'.
-- >          2
-- >     ┌────┴─────┐
-- > [-∞,0]       [4,7]
-- >     └──┐    ┌──┴──────┐
-- >        1    3       [8,∞]
exampleCorrect :: HTree
exampleCorrect = Node (Alive 2)
                      (Node (Grave NegInf 0) Empty (root (Alive 1))
                      (Node (Grave 4 7) (root (Alive 3)) (root (Grave 8 PosInf)))
```

Jednoduché testy

Ve vyznačené sekci je zakomentována jednoduchá šablona pro testování. Zakomentovaná je proto, abyste ji mohli ignorovat a nezpůsobovala vám problémy, pokud umažete některou z funkcí ze zadání. Šabloně nemusíte rozumět, použitá `do`-notace se procvičuje až na posledním cvičení, ale je poměrně přirozená.

V kostře už jsou jednoduché testy všech příkladů ze zadání. Odkomentujte ty, které si chcete otestovat.

```
main :: IO ()
main = do
    putStrLn "RUNNING TESTS"
    putStrLn "Testing compare:"
    testCompare
testCompare :: IO ()
testCompare = do
    test EQ (cmpHanged 1  (Alive 1))
    test EQ (cmpHanged 42 (Grave NegInf PosInf))
    test LT (cmpHanged 0  (Grave 6 9))
```

```
> main
RUNNING TESTS
Trying compare:
    FAIL:
     expected: EQ
     got: LT
    OK
    OK
```

Nenechte se zmást podobností s imperativním programováním, dokud však budete používat pouze jednoduché funkce jako `putStrLn :: String -> IO ()` nebo `test`, vše se bude chovat očekávatelně.

[2] Toto kreslítko fuguje lépe pro velké stromy a má také nastavitelný Unicode mód.