IB015 – Domácí úkol 8: Databáze modulů

Termín: do 7. 11. 23.59; 5 pokusů odevzdání; způsob odevzdání je podrobně popsán níže.

V prvním velkém domácím úkolu budeme pracovat s databází modulů, pro kterou si naprogramujeme několik užitečných funkcí.

S moduly jsme se již setkali několikrát. Například na Aise jsme si přidávali modul s novější verzí GHCi. Moduly jsme viděli i v Haskellu, kdy jsme importovali některé moduly, protože jsme chtěli využít nějakou knihovní funkci, kterou modul obsahoval. Modul je tedy jakýsi softwarový balíček, který v sobě sdružuje nějakou ucelenou funkcionalitu (GHC v případě modulu na Aise, funkce pro práci se seznamy v Haskellu v případě modulu `Data.List`).

Kostru zadání si můžete stáhnout z ISu.

Reprezentace databáze

Pro snazší orientaci v kódu si na začátku zavedeme několik typových aliasů. V této úloze budeme používat odpovídající alias vždy, když bude typ významově odpovídat názvu aliasu.

```
type ModuleName = String
type Version = String
type ModuleId = Int
type UserName = String
type Timestamp = Int
```

Databáze je potom jednoduchým aliasem pro trojici.

```
type Database = ([Module], [Dependency], [Usage])
```

První složkou této trojice je seznam modulů v databázi, druhou složkou je seznam závislostí a třetí složkou je seznam záznamů o používání modulů.

```
type Module = (ModuleId, ModuleName, Version)
```

Modul je opět jen typovým aliasem pro trojici, kde je první složkou id modulu (toto id je jednoznačné v rámci celé databáze), druhou složkou je název modulu a třetí je jeho verze.

```
type Dependency = (ModuleId, ModuleId)
```

Závislosti mezi moduly reprezentujeme dvojicí id modulů, kdy první složkou je modul, který závisí na modulu určeném druhou složkou.

Například závislost `(42, 66)` říká, že modul s id `42` závisí na modulu s id `66`.

Můžete předpokládat, že v databázi se nevyskytují cyklické závislosti.

```
type Usage = (ModuleId, UserName, Timestamp)
```

Konečně, záznamy o používání modulů opět reprezentujeme trojicí, jejíž první složkou je zase id modulu, druhou složkou je jméno uživatele, který modul použil, a třetí složkou je časová známka odpovídající času, kdy byl daný modul uživatelem použit.

Všimněte si, že `Timestamp` je jen typovým aliasem pro `Int`, což může být na první pohled zvláštní, ale ve skutečnosti se tak čas uchovává běžně (např. unixový čas počítá počet vteřin, které uběhly od 1. ledna 1970). Naše časové známky budou fungovat stejně včetně toho, že čím novější čas známka určuje, tím vyšší je její hodnota.

Funkce k implementaci

Vaším úkolem je naprogramovat následující sadu funkcí.

Nejprve budeme chtít implementovat několik tzv. *getterů*, což jsou funkce typicky nad nějakou datovou strukturou / datovým typem, které zpřístupňují jednotlivé (pod)složky dané struktury. Getter zpravidla používáme, pokud k nějaké složce přistupujeme často, proto, abychom zbytečně neduplikovali kód a zároveň jej učinili čitelnějším.

Všimněte si, že z názvu funkcí a jejich typové signatury je obzvláště zřejmé, co funkce dělá. Popíšeme tak pouze funkcionalitu prvního getteru, zbytek naprogramujte podle typové signatury

Poznámka: Tyto gettery mohou přijít vhod při implementaci ostatních funkcí. Řešení, která budou duplikovat jejich funkcionalitu bez jejich využití, nemohou dostat plný počet bodů za styl (viz níže).

- `getModuleId :: Module -> ModuleId`

  Tato funkce dostane jako argument modul a vrátí jeho id.

- `getModuleName :: Module -> ModuleName`
- `getVersion :: Module -> Version`
- `getUsageMId :: Usage -> ModuleId`
- `getTimestamp :: Usage -> Timestamp`
- `getUser :: Usage -> UserName`

Dále chceme naprogramovat funkce s lehce sofistikovanější funkcionalitou.

- `moduleInfo :: ModuleId -> [Module] -> (ModuleName, Version)`

  Funkce vrátí dvojici tvořenou jménem modulu a jeho verzí, která odpovídá modulu v seznamu modulů s id odpovídajícím tomu na vstupu.

  Můžete předpokládat, že modul s id odpovídající vstupnímu id v databázi existuje a je jediný.

- `moduleId :: ModuleName -> Version -> [Module] -> ModuleId`

  Funkce na vstupu dostane název modulu, jeho verzi a seznam modulů. Vrátí první id modulu, který odpovídá vstupní kombinaci.

  Opět můžete předpokládat, že takový modul v databázi existuje.

- `versionsOf :: ModuleName -> [Module] -> [(ModuleId, Version)]`

  Výstupem funkce je seznam všech dvojic *id modulu* × *verze*, pro které ve vstupním seznamu existuje modul, jehož název se shoduje se jménem modulu, které funkce dostane na vstupu.

  Pořadí dvojic musí odpovídat pořadí modulů ve vstupním seznamu.

- `lastUsedModule :: [Usage] -> ModuleId`

  Tato funkce vrátí id modulu, který byl podle vstupního seznamu použití použitý jako poslední. Pokud existuje více záznamů s nejnovějším časem, funkce vrátí ten, který je v seznamu nejblíže konci.

  Můžete předpokládat, že vstupní seznam je neprázdný. Naopak o jeho (ne)seřazenosti nemůžete předpokládat nic.

- `lastUserOf :: ModuleId -> [Usage] -> UserName`

  Výsledkem funkce je jméno uživatele, který modul s id, které dostane na vstupu, použil jako poslední. Stejně jako u předchozí funkce, existuje-li takových jmen více, vraťte to, jemuž odpovídající záznam byl v seznamu nejblíž konci.

  Opět můžete předpokládat, že pro takový modul záznam ve vstupním seznamu existuje.

- `insertUsage :: ModuleId -> UserName -> Timestamp -> [Usage] -> [Usage]`

  Tato funkce dostane id modulu, jméno uživatele, časovou známku a seznam záznamů použití modulů. Funkce aktualizuje záznam odpovídající kombinaci modulu a jménu uživatele, pokud je časová známka novější než ta ve vstupním seznamu – tato aktualizace nesmí změnit pořadí záznamů v seznamu. V případě, že žádný takový záznam neexistuje, jej funkce vytvoří a vloží na konec seznamu. Výstupem je aktualizovaný/rozšířený seznam záznamů o použití modulů.

  Můžete předpokládat, že pro každou kombinaci *id modulu* x *jméno uživatele* existuje nejvýše jeden záznam.

```
insertUsage 42 "Max" 1423 [] ~>* [(42,"Max",1423)]
insertUsage 42 "Max" 16 [(42, "Max", 1423)] ~>*  [(42,"Max",1423)]
insertUsage 42 "Max" 1699 [(42, "Max", 1423)] ~>*  [(42,"Max",1699)]
insertUsage 1 "Joe" 1529 [(42, "Max", 1423)] ~>* [(42,"Max",1423),(1,"Joe",1529)]
```

- `dependenciesOf :: ModuleId -> [Dependency] -> [ModuleId]`

  Tato funkce vrátí seznam všech id modulů, na kterých modul se vstupním id přímo závisí.

  Výstupní seznam nesmí obsahovat duplikáty.

```
dependenciesOf 1  [(1, 0), (1, 42), (32, 66)] ~>* [0,42]
dependenciesOf 42 [(1, 0), (1, 42), (32, 66)] ~>* []
dependenciesOf 32 [(1, 0), (1, 42), (32, 66)] ~>* [66]
dependenciesOf 1  [(1, 0), (1, 42), (32, 66), (66, 42)] ~>* [0,42]
dependenciesOf 32 [(1, 0), (1, 42), (32, 66), (66, 42)] ~>* [66]
```

- `dependingOn :: ModuleId -> [Dependency] -> [ModuleId]`

  Stejně jako u předchozí funkce, výsledkem je seznam modulů, tentokrát takových, které přímo závisí na modulu se vstupním id.

  Opět platí, že výstupní seznam nesmí obsahovat duplikáty.

  Poznámka: Funkci ***`dependingOn`*** *lze implementovat bez zbytečné duplikace kódu funkce* ***`dependenciesOf`****. Řešení duplikující kód nemohou dostat plný počet bodů za styl (viz níže).*

- `activeSince :: Timestamp -> Database -> [ModuleId]`

  Pro danou časovou známku a databázi je výstupem funkce seznam všech id modulů a jejich přímých závislostí, které někdo použil alespoň jednou od zadané vstupní časové známky. Výstupní seznam nesmí obsahovat duplikáty.

```
activeSince 42  ([(0, "ghc", "8.4.3"), (1, "xmonad", "0.15")], [(1, 0)]
                , [(0, "Joe", 42), (1, "Max", 10)]) ~>* [0]
activeSince 0   ([(0, "ghc", "8.4.3"), (1, "xmonad", "0.15")], [(1, 0)]
                , [(0, "Joe", 42), (1, "Max", 10)]) ~>* [0, 1]
activeSince 420 ([(0, "ghc", "8.4.3"), (1, "xmonad", "0.15")], [(1, 0)]
                , [(0, "Joe", 42), (1, "Max", 10)]) ~>* []
```

- `purgeModule :: ModuleId -> Database -> Database`

  Výstupem této funkce je opět databáze, která ale neobsahuje modul se vstupním id, ani žádný modul, který na něm závisí ať už přímo, nebo nepřímo (tj. může mezi nimi existovat libovolný počet modulů, které jeden na druhém závisí). Databáze také nesmí obsahovat žádné záznamy o použití takto odstraněných balíků.

```
purgeModule 1 ([(0, "ghc", "8.4.3"), (1, "xmonad", "0.15")], [(1, 0)], [(0, "Joe", 42)])
    ~>* ([(0,"ghc","8.4.3")],[],[(0,"Joe",42)])
purgeModule 0 ([(0, "ghc", "8.4.3"), (1, "xmonad", "0.15")], [(1, 0)], [(0, "Joe", 42)])
    ~>* ([], [], [])
purgeModule 6 ([(0, "ghc", "8.4.3"), (1, "xmonad", "0.15")], [(1, 0)], [(0, "Joe", 42)])
    ~>* ([(0,"ghc","8.4.3"),(1,"xmonad","0.15")],[(1,0)],[(0,"Joe",42)])
purgeModule 0 ([(0, "ghc", "8.4.3"), (1, "xmonad", "0.15"), (2, "xmobar", "0.11")]
              , [(1, 0), (2, 1)], [])
    ~>* ([], [], [])
```

Poznámky a tipy

- Importovat smíte moduly z balíku `base`.[1]
- 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.

[1] Výjimku tvoří moduly, které jsou „Unsafe“, ty však určitě nebudete potřebovat.

- Nevynalézejte znovu kolo! Snažte se využívat knihovních funkcí, především nemá smysl je zbytečně reimplementovat. Důrazně doporučujeme použít vyhledávač `hoogle` ve fakultní verzi, která automaticky vyhledává pouze v `base`. Pozornost věnujte především modulu `Data.List`.
- Pomocné funkce definujte lokálně, pokud jsou využívány jedinou funkcí.
- Funkce jsou v kostře zadefinovány jako `undefined`, takže projdou překladem, ale jejich zavolání způsobí chybu.
- Nejste-li si jisti nějakou částí zadání, zeptejte se v diskusním fóru.
- Nezapomeňte, že **opisování je zakázáno** a bude postihováno podle disciplinárního řádu.
- Přebíráte-li kód odjinud, uveďte zdroj, jinak bude na vaši práci pohlíženo jako na plagiát.
- Než řešení odevzdáte, **pečlivě si přečtěte následující sekci** a ujistěte se, že váš kód splňuje všechny náležitosti. Neztrácejte body jen kvůli nepozornému čtení pokynů.

Odevzdání

Tento domácí úkol se neodevzdává přes odpovědník, nýbrž přes příslušnou odevzdávárnu v Informačním systému.

- Do odevzdávárny vkládejte **jediný** soubor s příponou `.hs` obsahující vaši implementaci **všech** požadovaných funkcí.
  - Pokud chcete odevzdat znovu, můžete soubor přepsat či přidat nový, nezáleží na tom – vyhodnocuje se vždy nejnovější odevzdaný soubor.
  - Žádné jiné soubory nevkládejte.
  - Již vložené soubory nepřejmenovávejte, mohou se pak vyhodnotit dvakrát.
- Vaše řešení **musí zachovat všechny definice datových typů tak, jak jsou v zadání**, jediná povolená modifikace je přidání instance typové třídy.
- Řešení musí jít přeložit překladačem GHC 9.0.1. Je možné, že na svých počítačích máte starší verzi, což by nemělo vadit, jde-li o verzi 8.4 a vyšší. I přesto vám doporučujeme, abyste si před odevzdáním svůj kód zkusili zkompilovat a spustit na Aise (nezapomeňte přidat modul s novým GHC), kde máte k dispozici GHC ve verzi 9.0.1. V případě selhání testů už při kompilaci nepřijdete o žádný pokus.
- **Všechny globální funkce musí mít typovou signaturu**.
- V odevzdaném souboru neuvádějte hlavičku `module` (pokud nevíte, o co se jedná, vůbec to nevadí).

Vyhodnocování

- Vyhodnocení po nahrání souboru **není** okamžité.
  - Automatický testovací nástroj kontroluje soubory v odevzdávárně v pravidelných intervalech několika minut. Podle času odevzdání a vytížení vyhodnocovacího serveru může vyhodnocení trvat několik desítek minut.
  - Pokud se vám výsledky neobjevily cca do půl hodiny a neblíží se zatím čas deadline, dejte nám vědět ve fóru.
- Po vyhodnocení se získané body a případný výpis testů, které na vašem řešení selhaly, **objeví v poznámkovém bloku**.
  - U nesprávně implementovaných funkcí se dozvíte příklad vstupu, na němž se váš výsledek neshoduje s očekávaným.
  - Nejprve jsou v bloku uvedeny vstupy, následně výstup učitelského řešení a váš výstup.
- Máte **pět možností odevzdání**, započítává se nejlepší z nich.
  - Pokud při odevzdání neprojde kontrola syntaxe, tak se do tohoto limitu nepočítá, pokud však kontrola projde, je automaticky započítáno a nelze to zvrátit.
- Další odevzdání provedete tak, že do odevzdávárny nahrajete novou verzi.
- 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.
  - Není žádná garance rychlosti vyhodnocování (může se tedy stát, že výsledky řešení odevzdaného v neděli ve 21.00 neuvidíte do konce deadline).

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 (a pravděpodobně má) vaše dřívější odevzdání ve frontě. Z jeho pohledu tak došlo ke dvěma odevzdáním a vy si vyplýtváte jeden pokus. Podobně se vám mohou započítat odevzdání navíc, pokud do odevzdávárny

omylem vložíte více než jeden soubor nebo pokud soubor přejmenujete (každý soubor se vezme jako samostatné odevzdání).

Hodnocení

Za funkčnost můžete od automatických testů obdržet **až 2,5 bodu** podle toho, které funkce (nebo jejich části) se vám podařilo správně implementovat (viz tabulka níže).

Funkce	Body
`gettery`	0.1
`moduleInfo`	0.1
`moduleId`	0.1
`versionsOf`	0.1
`lastUsedModule`	0.1
`lastUserOf`	0.2
`insertUsage`	0.2
`dependenciesOf`	0.2
`dependingOn`	0.2
`activeSince`	0.2
`purgeModule`	1.0

Následně vám cvičící dají zpětnou vazbu na kód, a také vám za úhlednost a pochopitelnost řešení mohou udělit dalšího 0,5 bodu. Snažte se proto kód psát hezký a případné neočividné, či zajímavé části řešení stručně komentujte v kódu.

Tipy k psaní hezkého kódu naleznete ve sbírce. Hodnotit na kvalitu se bude poslední odevzdání s maximem bodů ze všech vašich odevzdání. Pokud vám tedy po dosažení plného počtu automaticky přidělovaných bodů ještě zbývají pokusy, můžete bezpečně zkusit svůj kód zkrášlit.

V součtu tedy můžete za úlohu získat **až 3 body**.