

# FLP - Haskell

## Úvod

Toto je zadání funkcionálního projektu do předmětu Funkcionální a logické programování 2024/25. Za projekt zodpovídá Ing. Daniel Poliakov (poliakov@fit.vut.cz).

## Obecné požadavky

Svá řešení odevzdávejte v zip archivu prostřednictvím VUT IS. Odevzdaný projekt musí obsahovat zdrojové texty v Haskellu. V hlavní složce musí být soubor **Makefile**, který program přeloží a výsledný binární soubor (pojmenovaný **flp-fun**) umístí také do hlavní složky. Dále se doporučuje přiložit stručný popis všeho, co nebylo dořešeno nebo naopak bylo implementováno nad rámec zadání.

Pro překlad použijte kompilátor **ghc** s volbou **-Wall**. Můžete využít standardní knihovny z balíku base, případně knihovny z balíků containers, parsec, vector, split, directory, a random. Za referenční verzi Haskellu je považována verze dostupná na serveru **merlin**. Před odevzdáním si ověřte, zda lze zip rozbalit a program přeložit a spustit.

## Hodnocení

Za vypracovaný projekt lze získat až 12 bodů (+ až 3 bonusové body, ad laboratorní cvičení). Hodnocena bude míra splnění zadání, kvalita řešení, a kvalita kódu. Vyvarujte se nestandardních funkcí, které obcházejí bezpečné otypování nebo obcházejí zapouzdření vedlejších efektů - nepoužívejte nic, co má ve svém jméně slovo unsafe. Snažte se psát čistý kód respektující různé varování a připomínky, které zazněly na přednáškách a cvičeních.

Prosím také o dodržení, ať: úvodní řádky zdrojových kódů obsahují název projektu, login, jméno autora a rok řešení; součástí definic na globální úrovni jsou typové anotace a stručný a výstižný komentář; je zřejmé, co a z jakých modulů importujete ([https://wiki.haskell.org/Import\\_modules\\_properly](https://wiki.haskell.org/Import_modules_properly)).

Projekt vypracovávejte **samostatně**. Všechna odevzdaná řešení budou testována na původnost jak mezi sebou, tak s řešeními a open source knihovnami dostupnými na internetu. V případě **plagiátorství** jakéhokoliv druhu bude projekt hodnocen nulovým počtem bodů, navíc nebude udělen zápočet a bude zvaženo zahájení disciplinárního řízení. Podobně bude přistupováno při zjištění **využití generativních modelů** (jako ChatGPT nebo GitHub Copilot).

K zadání budou dostupná malá podmnožina testů, zejména pro jistotu dodržení formátu a základní funkcionality. Některé testy budou označeny jako **nutné pro splnění minima (4**

**bodů) na zápočet.** Jako studenti si budete moct dopředu ověřit, že máte jistý zápočet (hodnocení 4 a více bodů). Zadání, které tyto dopředu známé nutné testy nesplní jsou **automaticky hodnoceny nulou**. Pro splnění minima je potřeba mít vyřešen alespoň podúkol 1 (načtení rozhodovacího stromu a klasifikace).

## Rozhraní programu

Program bude možné spustit (podúkol 1):

```
flp-fun -1 <soubor obsahující strom> <soubor obsahující nove data>
```

nebo (podúkol 2)

```
flp-fun -2 <soubor obsahující trenovací data>
```

Příklady vstupu a výstupu jsou uvedeny u detailů podúkolů. Vstupy (můžete předpokládat, že jsou všechny ve správném formátu) dostupné přes argumenty jsou vždy dostupné přes soubory. Výstup je vždy na **stdout**.

## Podúkol 1: Načtení rozhodovacího stromu a klasifikace

### Cíl

Navrhnout a definovat interní datové typy pro rozhodovací strom. Zpracovat zadaný textový formát stromu ze souboru. Načíst nová data ze souboru. Použít načtený strom pro klasifikaci nových dat.

### Popis

Načtení stromu:

Implementujte část, která načte rozhodovací strom ve specifikovaném textovém formátu. Textový formát je definován jako:

1. uzel je označen jako: `Node: index\_příznaku, práh`,
2. list je označen jako: `Leaf: třída`
3. odsazení (dvěma mezerami) určuje hierarchii stromu

Příklad vstupu:

```
Node: 0, 5.5  
  Leaf: TridaA  
  Node: 1, 3.0
```

```
Leaf: TridaB
```

```
Leaf: TridaC
```

## Klasifikace

Implementujte část, která pro načtený strom a nové data určí výslednou třídu. Nové data jsou definované po řádcích. Každá hodnota příznaku je oddělená čárkou. Počet příznaků není omezen (ale v testech neočekávám více jak 1024 příznaků).

Příklad vstupu:

```
2.4,1.3
```

```
6.1,0.3
```

```
6.3,4.4
```

Příklad výstupu (pro zadaný strom výše):

```
TridaA
```

```
TridaB
```

```
TridaC
```

## Podúkol 2: Trénování rozhodovacího stromu

### Cíl

Implementovat zjednodušený algoritmus pro trénování rozhodovacího stromu inspirovaný metodou CART.

### Popis

Implementujte část, která načte trénovací data a vygeneruje nový strom. Detail implementace algoritmu je ponechán volbě studentů. Očekává se alespoň nějaká zjednodušená verze inspirovaná algoritmem CART (výpočet Gini index, výpočet split score/thresholdů). Pro nějaké pokročilejší řešení možné implementovat i pruning (ale stačí základní funkční implementace).

Příklad vstupu (tři tečky značí další neuvedené řádky):

```
2.4,1.3,TridaA
```

```
6.1,0.3,TridaB
```

```
6.3,4.4,TridaC
```

2.9,4.4,TridaA

3.1,2.9,TridaB

.

.

.

Výstup:

Node: 0, 5.5

Leaf: TridaA

Node: 1, 3.0

Leaf: TridaB

Leaf: TridaC