**Programiranje I: 1. izpit**

28. avgust 2023

Čas reševanja je 120 minut. Veliko uspeha!

### 1. naloga

**a)** Napišite funkcijo `vzporedna : int * int -> int * int -> bool`, ki preveri, ali sta podana vektorja vzporedna.

**b)** Napišite funkcijo `zlozi_pocez : 'a list -> 'b list -> ('a * 'b) list`, ki sprejme dva seznama in vrne nov seznam, kjer so elementi prekrižani pari elementov. Predpostavite lahko, da sta seznama enakih in sodih dolžin.

```
# zlozi_pocez [1; 2; 3; 4] [1.1; 2.2; 3.3; 4.4];;
- : (int * float) list = [(1, 2.2); (2, 1.1); (3, 4.4); (4, 3.3)]
```

**c)** Napišite funkcijo `kompozitumi : ('a -> 'a) -> int -> ('a -> 'a) list`, ki sprejme funkcijo in nenegativno število ter vrne seznam zaporednih kompozitumov te funkcije s samo sabo, torej `potenciraj f n` = [$f^1$;$f^2$;...;$f^n$], kjer je $f^i = \underbrace{f \circ \cdots \circ f}_{i}$

**d)** Napišite funkcijo `repi : 'a list -> 'a list list`, ki sprejme seznam in vrne seznam vseh njegovih repov.

```
# repi [1;2;3];;
- : int list list = [[1; 2; 3]; [2; 3]; [3]; []]
```

**e)** Vsoto $A + B$ lahko predstavimo s tipom

```
type ('a, 'b) sum = Left of 'a | Right of 'b
```

Definirajte preslikavi

```
iso1 : ('a * ('b, 'c) sum) -> (('a * 'b), ('a * 'c)) sum
iso2 : (('a * 'b), ('a * 'c)) sum -> ('a * ('b, 'c) sum)
```

ki ustrezata izomorfizmu $A \times (B + C) \cong A \times B + A \times C$.

**2. naloga**

S tipom `game_tree` predstavimo odločitveno drevo igre. Igra je lahko bodisi zaključena z zmagovalcem ali z neodločenim izidom bodisi je na potezi igralec, ki mora izbrati med nepraznim seznamom možnosti. Vsaka možnost je opisana z verjetnostjo, da jo igralec izbere, in z novim odločitvenim drevesom, ki opisuje nadaljevanje igre po izbiri.

```
type player = White | Black

type game_tree =
  | Winner of player
  | Tie
  | Decision of player * (float * game_tree) list
```

Na primer, drevo

```
let primer =
  Decision ( White, [
    (0.3, Decision (Black, [ (0.5, Winner White); (0.5, Winner Black) ]));
    (0.7, Decision (Black, [ (0.5, Tie);          (0.5, Winner Black) ]));
  ] )
```

predstavlja igro, v kateri beli z verjetnostjo `0.3` izbere možnost, po kateri črni z enako verjetnostjo izbere svoj poraz ali zmago, z verjetnostjo `0.7` pa možnost, po kateri črni izbira med neodločenim izidom in zmago.

Drevo je *veljavno*, če so vse verjetnosti v seznamu možnosti števila med `0.0` in `1.0`, ki se seštejejo v `1.0`. Pri tem lahko predpostavite, da ne bo prišlo do napak pri seštevanju števil s plavajočo vejico.

**a)** Napišite funkcijo `prestej_zmage: game_tree -> int`, ki prešteje število listov, v katerih zmaga beli igralec.

**b)** Napišite funkcijo `rezultat: game_tree -> result`, ki izračuna verjetnosti, da zmaga beli igralec, da zmaga črni igralec in da se igra konča z neodločenim izidom. Pri tem je tip `result` podan kot:

```
type result = { white_wins : float; black_wins : float; ties : float }
```

Predpostavite lahko, da je drevo igre veljavno.

```
  # rezultat primer;;
  - : result = {white_wins = 0.15; black_wins = 0.5; ties = 0.35}
```

**c)** Napišite funkcijo `je_veljavno: game_tree -> bool`, ki sprejme drevo igre in preveri, če je veljavno.

**d)** Napišite funkcijo `odigraj_igro: game_tree -> int list -> (player * float) option`, ki sprejme drevo igre in seznam odločitev, ki jih zaporedoma ubirata igralca, ter izračuna, kdo bo zmagal, ter kolikšna je verjetnost danega poteka igre. Vsaka odločitev je predstavljena z indeksom v seznamu možnosti, ki so na voljo na trenutni potezi. Če je seznam odločitev prekratek, ne vodi do zmagovalca, ali pa kakšen indeks ni v skladu s seznamom možnosti, naj funkcija vrne `None`. Predpostavite lahko, da je drevo igre veljavno.

**3. naloga**

*Nalogo lahko rešujete v Pythonu ali OCamlu.*

Dedek Peter je skuhal gigantski lonec marmelade, zdaj pa bi z njo napolnil kozarce različnih velikosti, pri tem pa mu je na pomoč priskočilo $k$ vnukov. Dedek vsakič z začetka police vzame nekaj zaporednih kozarcev, pokliče naslednjega vnuka ter mu jih izroči. Ko ostane samo še en vnuk, mu da vse preostale kozarce. Kljub različni starosti so vsi vnuki enako spretni pri polnjenju kozarcev in sicer za vsak kozarec potrebujejo toliko časa, kot je velik, čas za menjavo med dvema kozarcema pa je zanemarljiv.

Napišite funkcijo

```
polnjenje_marmelade(polica: list[int], k: int) -> tuple[int, list[list[int]]]
```

oziroma

```
polnjenje_marmelade : int list -> int -> int * int list list
```

ki vzame velikosti kozarcev na polici in število vnukov ter izračuna najkrajši čas, v katerem lahko napolnijo kozarce, ter razdelitev kozarcev med vnuke. Če je takih razdelitev več, naj funkcija vrne poljubno.

Na primer, če so na polici kozarci velikosti $1,2,6,5,3,4$ in ima dedek 3 vnuke, lahko kozarce najhitreje napolnijo v 9 enotah časa, če si jih razdelijo kot $(1,2,6),(5),(3,4)$ ali kot $(1,2,6),(5,3),(4)$.