

## Programiranje I: 3. izpit

24. avgust 2021

Čas reševanja je 120 minut. Veliko uspeha!

### 1. naloga

**a)** Napišite predikat `je_urejena : int * int * int -> bool`, ki pove, ali je podana trojica celih števil urejena strogo naraščajoče.

**b)** Napišite funkcijo `poskusi_deljenje : float option -> float option -> float option`, ki sprejme morebitni deljenec in morebitni delitelj ter vrne rezultat deljenja, če se to da, ali pa `None`, če ne (kadar kakšnega argumenta ni ali pa bi prišlo do deljenja z nič).

```
# poskusi_deljenje (Some 1.0) (Some 2.0);;  
- : float option = Some 0.5  
# poskusi_deljenje (Some 1.0) (Some 0.0);;  
- : float option = None  
# poskusi_deljenje None (Some 2.0);;  
- : float option = None
```

**c)** Definirajte funkcijo `zavrti : 'a list -> int -> 'a list`, ki seznam zavrti za dano število mest v levo (v vsaki rotaciji se prvi element prestavi na konec seznama).

```
# zavrti [1; 2; 3; 4; 5] 2;;  
- : int list = [3; 4; 5; 1; 2]
```

**d)** Napišite funkcijo `razdeli : ('a -> int) -> 'a list -> ('a list * 'a list * 'a list)`, ki sprejme cenilno funkcijo in seznam elementov. Vrne naj trojico, kjer so na prvem mestu vsi elementi za katere je cenilna funkcija negativna, na drugem vsi, kjer je enaka 0, na tretjem pa vsi preostali elementi. Elementi naj v seznamih nastopajo v enakem vrstnem redu kot v prvotnem seznamu. Za vse točke naj bo funkcija repno rekurzivna.

```
# razdeli ((-) 3) [1; 2; 3; 4; 5; 6];;  
- : int list * int list * int list = ([4; 5; 6], [3], [1; 2])
```

## 2. naloga

Pri tej nalogi bomo za slovar uporabili kar enostavno implementacijo z asociativnim seznamom, ki smo jo spoznali na predavanjih. S spodaj definiranimi funkcijami si lahko pomagata pri vseh podnalogah.

```
type ('a, 'b) slovar = ('a * 'b) list

let prazen_slovar : ('a, 'b) slovar = []

let velikost (m : ('a, 'b) slovar) = List.length m

let vsebuje (x : 'a) (m : ('a, 'b) slovar) = List.mem_assoc x m

(* Vrne vrednost, ki pripada ključu ali None *)
let najdi x (m : ('a, 'b) slovar) = List.assoc_opt x m

(* Doda vrednost v slovar in poveži prejšnjo, če obstaja *)
let dodaj (k, v) (m : ('a, 'b) slovar) = (k, v) :: List.remove_assoc k m
```

Matematične izraze predstavimo z dvojiškimi drevesi, v katerih vozlišča predstavljajo aritmetične operacije, listi pa števila ali spremenljivke, predstavljene z nizi. Izraz v drevo pretvorimo tako, da pri operaciji levi podizraz vzamemo za levo poddrevo, desni podizraz za desno, v vozlišče pa zapišemo operator.

```
type operator = Plus | Minus | Krat | Deljeno

type 'a izraz =
  | Spremenljivka of string
  | Konstanta of 'a
  | Operacija of ('a izraz * operator * 'a izraz)
```

Izrazu  $(x - 3) - (y * (z/x))$  pripada drevo

```
let primer =
  Operacija
    ( Operacija (Spremenljivka "x", Minus, Konstanta 3),
      Minus,
      Operacija
        ( Spremenljivka "y",
          Krat,
          Operacija (Spremenljivka "z", Deljeno, Spremenljivka "x") ) )
```

**a)** Napišite funkcijo `prestej : izraz -> int`, ki vrne število vseh *različnih* spremenljivk v izrazu.

**b)** Napišite funkcijo `izlusci : 'a izraz -> (string * int) slovar`, ki sprejme izraz in vrne slovar, ki pove, kolikokrat se posamezna spremenljivka pojavi v izrazu. Vrstni red v slovarju ni pomemben.

**c)** Napišite funkcijo `izracunaj : (string * int) slovar -> int izraz -> option int`, ki sprejme izraz in slovar vrednosti spremenljivk ter poskuša izračunati vrednost izraza. Če to ni mogoče (deljenje z 0 ali manjkajoča definicija spremenljivke), naj bo rezultat `None`.

```
# izracunaj [("x",3); ("y", 4); ("z",5)] primer;;
- : int option = Some (-4)
```

**d)** Ocenite časovno zahtevnost funkcije `izracunaj` v odvisnosti od velikosti izraza  $n$  (torej števila vseh vozlišč in listov v drevesu) ter števila različnih spremenljivk  $m$ . Kako se časovna zahtevnost spremeni, če bi za slovar uporabili uravnoteženo iskalno drevo?

### 3. naloga

*Nalogo lahko rešujete v Pythonu ali OCamlu.*

Ker ne zna ničesar koristnega, se je Miha odločil, da bo postal vplivnež. Priskrbel si je zemljevid plaže, ki za vsako mesto na plaži pove, koliko sledilcev dobi (ali izgubi), če objavi fotografijo s tega mesta. Plažo predstavimo s pravokotno mrežo dimenzije  $M \times N$ , kjer za vsako celico povemo, koliko sledilcev bo Miha dobil, če se na poti prek te celice slika. Miha svojo pot začne v točki  $(0, 0)$ , konča v točki  $(M - 1, N - 1)$ , na svoji poti do cilja pa bi rad nabral čim več sledilcev, pri čemer med potjo nikoli ne sme zaiti izven plaže. Miha se lahko običajno premika na tri načine: korak desno, korak navzdol, korak desno-navzdol in pri tem objavi slike iz vseh lokacij na svoji poti (tudi če so njihove vrednosti negativne). Poleg osnovnih korakov lahko *največ enkrat* na svoji poti naredi tudi korak nazaj (torej se vrne na polje, kjer je bil trenutek prej). Ker sledilci nimajo dobrega spomina, se lahko Miha večkrat slika na isti lokaciji in vsakič dobi (ali izgubi) podano število sledilcev.

Definirajte funkcijo, ki sprejme zemljevid plaže in vrne maksimalno število sledilcev, ki jih Miha lahko nabere na podani plaži. Miho zanima zgolj končna sprememba sledilcev, zato je ta lahko skupno tudi negativna.

Na spodnji mreži je najvplivnejši sprehod  $(1, 2, 5, 30, 5, 30, -1, 5)$  vreden 77 sledilcev.

```
[
    [1, 2, -3, -10, 9],
    [0, 0, 5, 5, 2],
    [1, 2, 30, -1, 0],
    [4, 3, -20, -1, 5],
]
```