Programiranje I: 2. izpit

9. februar 2021

Čas reševanja je 120 minut. Veliko uspeha!

1. naloga (30 točk)

a) (4 točke) Napišite funkcijo, ki za trojico celih števil preveri, ali tvorijo *pitagorejsko trojico*. Trojica (a, b, c) je pitagorejska, če je $a^2 + b^2$ enako c^2.

```
pitagorejska_trojica : int * int * int -> bool

# pitagorejska_trojica (3, 4, 5);;
- : bool = true
# pitagorejska_trojica (5, 4, 3);;
- : bool = false
```

b) (5 točk) Napišite funkcijo, ki za celo število x vrne celo število a, kjer velja $\sqrt{x} \in [a, a+1)$.

```
priblizek_korena : int -> int

# priblizek_korena 9;;
- : int = 3
# priblizek_korena 17;;
- : int = 4
```

c) (7 točk) Definirajte funkcijo, ki sprejme seznam celih števil in najprej **izpiše** vsa soda števila v seznamu, nato pa **izpiše** še vsa liha števila v seznamu. Števila naj bodo izpisana v isti vrstici in med njimi ne želimo presledkov.

```
izpisi_soda_liha : int list -> unit

# izpisi_soda_liha [3; 1; 4; 1; 5; 9; 2];;
4231159- : unit = ()
# izpisi_soda_liha [2; 7; 1; 8; 2; 8; 1];;
2828711- : unit = ()
```

d) (7 točk) Napišite funkcijo, ki sprejme seznam elementov tipa `option` in preveri, da si v seznamu izmenično sledijo konstruktorji `None` in `Some`.

```
alternirajoci_konstruktorji : 'a option list -> bool

# alternirajoci_konstruktorji [None; Some 1; None; Some 100; None];;
- : bool = true
# alternirajoci_konstruktorji [None; Some 1; Some 10];;
- : bool = false
# alternirajoci_konstruktorji [Some 1; None; Some 10; None];;
- : bool = true
```

e) (7 točk) Funkcija `najmanjsi_rezultat` naj za element `x` in seznam funkcij `fs` vrne **indeks** funkcije, ki ima pri argumentu `x` najmanjšo vrednost izmed vseh funkcij v seznamu `fs`. Ker je seznam morda prazen, naj bo rezultat tipa `option`.

Za vse točke naj bo funkcija repno rekurzivna.

```
najmanjsi_rezultat : 'a -> ('a -> 'b) list -> int option

# najmanjsi_rezultat (-10) [(fun x -> 10 * x); succ; (fun y -> -10 * y)];;
- : int option = Some 0
# najmanjsi_rezultat 10 [(fun x -> 10 * x); succ; (fun y -> -10 * y)];;
- : int option = Some 2
# najmanjsi_rezultat 30 [];;
- : int option = None
```

2. naloga (30 točk)

Za učinkovitejše iskanje po leksikografsko urejenih parih bomo uporabili *leksikografska drevesa*, ki jih ustvarimo s pomočjo binarnih dreves.

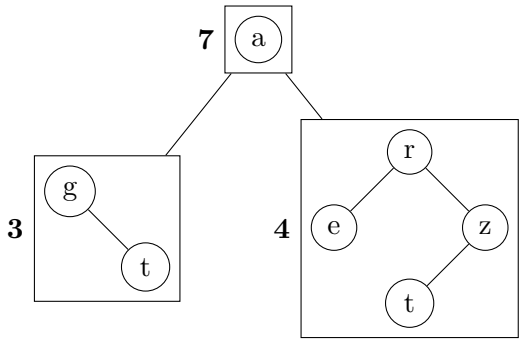
```
type 'a tree = Empty | Node of 'a tree * 'a * 'a tree
```

Leksikografsko drevo za pare tipa `'a * 'b` je binarno drevo, ki ima v vozlišču element tipa `'a` (da lahko primerjamo po prvi komponenti) in pa drevo tipa `'b tree` (za primerjanje po drugi komponenti).

```
type ('a, 'b) lexi_tree = ('a * 'b tree) tree
```

Par `(a, b)` se nahaja v leksikografskem drevesu, če imamo v drevesu vozlišče s parom `(a, subtree)` in se `b` nahaja v `subtree`.

Primer drevesa za pare `(3, "g")`, `(3, "t")`, `(7, "a")`, `(10, "e")`, `(10, "r")`, `(10, "t")` in `(10, "z")` je:



a) (3 točke) V OCamlu definirajte primer, ki ustreza zgornjemu leksikografskemu drevesu.

b) (7 točk) Napišite funkcijo, ki preveri ali je par prisoten v leksikografskem drevesu.

c) (7 točk) Napišite funkcijo za vstavljanje elementov v leksikografsko drevo. Če je element že v drevesu vrnite nespremenjeno drevo.

d) (7 točk) Napišite funkcijo `lexi_fold`, ki sprejme funkcijo `f` in začetno vrednost akumulatorja, nato pa funkcijo `f` zloži preko leksikografskega drevesa. Vrstni red zlaganja je določen z leksikografsko urejenostjo.

```
lexi_fold : ('a -> 'b -> 'c -> 'a) -> 'a -> ('b, 'c) lexi_tree -> 'a
```

e) (6 točk) Definirajte funkcijo, ki vrne urejen seznam vseh elementov, ki se nahajajo v leksikografskem drevesu.

3. naloga (20 točk)

Nalogo lahko rešujete v Pythonu ali OCamlu.

Psička Nara po njivi preganja krokarje. Opazila je, da jo lastnik čaka na drugem koncu polja, zato hiti k njemu, pri tem pa hoče prestrašiti kar se da veliko ubogih ptičev.

Njivo predstavimo kot matriko, ki v vsakem polju vsebuje število krokarjev, ki jih pasja navihanka prežene, če teče preko tega polja.

$$\begin{bmatrix} 2 & 3 & 0 & 2 & 9 \\ 8 & 3 & 5 & 1 & 2 \\ 1 & 2 & 7 & 2 & 0 \\ 4 & 3 & 6 & 5 & 5 \end{bmatrix}$$

a) (10 točk) Nara se nahaja v zgornjem levem kotu njive (polje $(0, 0)$). Ker se ji mudi k lastniku, se vztrajno premika desno. Na vsakem koraku se lahko premakne:

- desno
- diagonalno desno-gor
- diagonalno desno-dol

Pregon krokarjev zaključi na poljubnem skrajno desnem polju njive. Napišite funkcijo, ki izračuna največje število krokarjev, ki jih lahko nagajivka prežene.

b) (10 točk) Funkcijo iz točke (a) prilagodite tako, da ji dodatno podate indeks vrstice, v kateri Nara začne, in indeks vrstice, v kateri Nara konča. Funkcija naj vrne seznam **vseh** optimalnih poti, kjer pot predstavimo s seznamom indeksov polj, preko katerih Nara teče.