Programiranje I: 1. izpit

27. avgust 2019

Čas reševanja je 150 minut. Veliko uspeha!

1. naloga

a) Napišite funkcijo, ki po komponentah odšteje dve trojici.

```
odstej_trojici : int * int * int -> int * int * int -> int * int * int
```

b) Napišite funkcijo, ki sprejme funkcijo `f` in naravno število `n` ter vrne največjega od rezultatov izračuna funkcije `f` na naravnih številih od 0 do `n` (vključno z obema mejama).

```
max_rezultat_do_n : (int -> 'a) -> int -> 'a
```

c) Napišite funkcijo, ki sprejme seznam elementov tipa `'a option` in vrne seznam elementov tipa `'a`, tako da odstrani vse pojavitve `None`.

```
pocisti_seznam : 'a option list -> 'a list
```

Funkcija naj bo repno rekurzivna.

d) Napišite funkcijo, ki za dani seznam celih števil preveri, ali v njem podzaporedje sodih števil narašča in podzaporedje lihih števil pada. Kot primer: `[5; 2; 4; 1; 6]` je pravilno urejen, medtem ko `[3; 2; 4; 5; 6]` ni, saj liha števila ne padajo.

```
preveri_urejenost : int list -> bool
```

2. naloga

Vsi elementi seznama morajo imeti enak tip, zaradi česar nam jeziki s tipi ne dovolijo izrazov kot je `[1; 2; [3; [4]; []]; [5]]`. Ker želimo uporabljati gnezdene sezname, konstruiramo nov tip elementov, ki predstavlja element ali pa ugnezden podseznam (ki lahko ponovno vsebuje elemente ali gnezdenja):

```
type 'a gnezdenje =
    | Element of 'a
    | Podseznam of 'a gnezdenje list
```

a) Definirajte `gnezdenje_primer`, ki modelira `[1; 2; [3; [4]; []]; [5]]`.

b) Napišite funkcijo

```
najvecja_globina : 'a gnezdenje list -> int
```

ki vrne največjo globino gnezdenja v seznamu. Na zgornjem primeru torej vrne 3, saj je `4` gnezden tri sezname globoko. Prazen seznam naj ima globino gnezdenja 1.

c) Napišite funkcijo

```
preslikaj : ('a -> 'b) -> 'a gnezdenje list -> 'b gnezdenje list
```

ki v gnezdenih seznamu vse elemente preslika s podano funkcijo.

d) Napišite funkcijo

```
splosci : 'a gnezdenje list -> 'a list
```

ki splošči gnezdene sezname. Iz zgornjega primera tako dobimo `[1; 2; 3; 4; 5]`.

e) Napišite funkcijo

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

ki preveri, ali v zunanjem seznamu (in zgolj v zunanjem seznamu!) konstruktorja `Element` in `Podseznam` nastopata izmenično. V zgornjem primeru vrne `false`, saj ima na začetku dva zaporedna elementa (ne zanima pa nas ali imajo podseznami tudi takšno strukturo).

f) Napišite funkcijo

```
zlozi_preko_gnezdenja :
    ('acc -> 'a -> 'acc) -> 'acc -> 'a gnezdenje list -> 'acc
```

ki na gnezdenih seznamih deluje tako kot `List.fold_left`. Za vse točke naj bo funkcija repno rekurzivna, vendar bodite pazljivi, saj operator `@` v standardnem OCamlu ni repno rekurziven (ali so vgrajene funkcije repno rekurzivne lahko preverite na spletu).

3. naloga

Nalogo lahko rešujete v Pythonu ali OCamlu.

Vaš sošolec Mortimer se je med potovanjem po Finski spravil v krepko godljo. Po divjem poskušanju lokalne vodke se je namreč stepel s kravo, zaradi česar ga sedaj lovi finska govedorejska mafija. Na srečo so za njegovo hrabro bitko slišale vse rokovske in metalske skupine, ki so mu pripravljene ponuditi prevoz.

Ker je Mortimer pridno poslušal predavanja iz finančne matematike, med potjo uspe prislužiti nekaj denarja, s katerim bo lahko plačal prevoz. Finci, navdušeni nad Mortimerjevim pogumom, mu dovolijo, da se med potjo zadolži, dokler na koncu pobega vse stroške povrne.

Mesta na poti predstavimo kot seznam, katerega elementi so seznami vseh možnih nadaljnjih poti. Pot je par `(indeks_cilja, denar)`. Kot primer,

```
[[(1, 10), (3, -10)],    # 0
 [(2, 10), (5, -20)],    # 1
 [(3, -10)],             # 2
 [(4, 15)],              # 3
 [(5, 0)]]               # 4
```

pomeni, da lahko v mestu 1 Mortimer izbere med prevozom v mesto 2, kjer dodatno zasluži 10 evrov, ali pa prevoz v mesto 5, ki ga stane 20 evrov. Ker beži pred mafijo, lahko predpostavite, da bodo možne zgolj poti na mesta z višji indeksom (torej ni ciklov).

Pobeg je uspešen, čim lahko odpotuje v mesto, ki ni več na seznamu (torej skok na indeks, ki preseže seznam) in ima po koncu zadnjega skoka 0 ali več evrov. Napišite program, ki nam vrne pot z najmanjšim številom skokov, predstavljeno kot seznam indeksov mest na poti. Ker pobeg morda ni možen, naj v tem primeru funkcija vrne `None`.

Na primeru je optimalna pot `[0, 3, 4, 5]`, kjer se Mortimer sicer zadolži, vendar v skoku iz 3 v 4 zasluži dovolj, da konča z 5 evri. Hitrejša pot bi bila `[0, 1, 5]`, vendar v tem primeru Mortimer na koncu dolguje še 10 evrov.

Mortimer pot vedno začne v mestu z indeksom 0 in ima 0 evrov (saj je vse zapil). Funkcija (oz. program) sprejme seznam, ki predstavlja finska mesta in vrne seznam indeksov mest, v katerih se Mortimer ustavil. Funkcija naj bo memoizirana.