Programiranje I: 2. izpit

12. februar 2024

Čas reševanja je 120 minut. Veliko uspeha!

1. naloga

a) Napišite funkcijo `je_pravokotni : int -> int -> int -> bool`, ki preveri, ali podane stranice (v poljubnem vrstnem redu) ustrezajo Pitagorovemu izreku.

b) Napišite funkcijo `geometrijsko : int -> int -> int -> int list`, ki sprejma naravna števila `a`, `q` in `n`, ter vrne prvih `n` členov geometrijske vrste s prvim členom `a` in kvocientom `q`.

c) Napišite funkcijo `premesaj : ('a * 'b * ('c * 'd * 'e)) -> ('c * 'd * ('a * 'e * 'c))`.

d) Napišite funkcijo `odberi : 'a list -> int list -> 'a list`, ki sprejme seznam elementov in seznam indeksov ter vrne seznam elementov, ki se nahajajo na mestih, ki so podani z indeksi. Seznam indeksov bo urejen naraščajoče, ne bo vseboval ponovitev in vsi indeksi bodo veljavni glede na dolžino seznama elementov. Vaša funkcija mora delovati učinkovito (torej v času, ki je linearen v dolžini vhodnega seznama elementov).

```
# odberi [ 1; 2; 3; 4; 5; 6; 7; 8; 9; 10 ] [ 0; 2; 4; 6; 8 ];;
- : int list = [1; 3; 5; 7; 9]
```

2. naloga

```
type element = Variable of char | Constant of char
type result = element list
(* Seznam pravil podamo s pari, ki spremenljivkam priredijo sliko  *)
(* Za vse preslikave bomo predpostavili, da so enolične *)
type rules = (char * result) list
type l_tree = Node of element * l_tree list option

let example0 =
  Node
    ( Variable 'A',
      Some
        [
          Node
            ( Variable 'A',
              Some [ Node (Variable 'A', None); Node (Variable 'B', None) ] );
          Node (Variable 'B', Some [ Node (Variable 'A', None) ]);
        ] )

let rules0 = [ ('A', [ Variable 'A'; Variable 'B' ]); ('B', [ Variable 'A' ]) ]

let example1 =
  Node
    ( Variable '0',
      Some
        [
          Node (Variable '1', None);
          Node (Constant '[', None);
          Node (Variable '0', None);
          Node (Constant ']', None);
          Node (Variable '0', None);
        ] )

let rules1 =
  [
    ('1', [ Variable '1'; Variable '1' ]);
    ( '0',
      [ Variable '1'; Constant '['; Variable '0'; Constant ']'; Constant '0' ]
    );
  ]
```

a) Element drevesa je `veljaven`, če nima praznega seznama poddreves (če njegovo poddrevo ni enako `Some []`). Element drevesa je `list`, če ima za poddrevo `None`. Drevo je `veljavno`, če so vsi njegovi elementi veljavni, so globine vseh poddreves enake in so vsa poddrevesa veljavna. Napišite funkcijo `veljavno : l_tree -> bool`, ki preveri, ali je dano drevo veljavno. Oba podana primera dreves sta veljavna.

b) Napišite funkcijo `preslikaj : rules -> element -> element list`, ki za podan seznam pravil in element vrne rezultat, v katerega se element preslika. Lahko predpostavite, da je element vedno možno preslikati. Če je element konstanta, naj funkcija vrne seznam, ki vsebuje samo ta element, ne glede na seznam pravil.

c) Napišite funkcijo `zadnja_vrstica : l_tree -> char list`, ki vrne seznam elementov zadnje vrstice drevesa. Lahko predpostavite, da je drevo vedno veljavno. Elementi v rezultatu naj bodo v istem vrstnem redu, kot se pojavijo v drevesu, če bi ga brali od leve proti desni. Za vse točke naj bo funkcija repno rekurzivna.

d) Napišite funkcijo `preveri_pravila : rules -> bool`, ki za dani seznam pravil preveri, ali so:

1. enolična, torej ali se vsaka spremenljivka pojavi največ enkrat na levi strani pravila,
2. ponovljiva, torej, ali je vsaka spremenljivka, ki se pojavi na desni strani pravila, tudi na levi.

e) Napišite funkcijo `preslikaj_drevo : rules -> l_tree -> l_tree`, ki za dano drevo in seznam pravil vrne preslikano drevo - drevo, pri katerem smo na spodnjem nivoju enkrat uporabili seznam pravil. Preslikano drevo je takšno, da so otroci listov izvirnega drevesa zamenjani listi, kjer so vrednosti novih listov take, kot jih določajo pravila (enako, kot pri funkciji `preslikaj`). Lahko predpostavite, da je drevo vedno veljavno. Drevo iz primera `example0` smo dobili tako, da smo dvakrat preslikali drevo `Node (Variable 'A', None)` z uporabo `rules0`. Drevo iz primera `example1` smo dobili tako, da smo enkrat preslikali drevo `Node (Variable '0', None)` z uporabo `rules1`.

3. naloga

Nalogo lahko rešujete v Pythonu ali OCamlu.

Kurent mora po dolgi in naporni pustni soboti obiskati še Ameriško mesto (zaradi kulturne izmenjave), ki ga predstavimo z matriko `N x M`. Na vsakem mestu v matriki je celo (lahko negativno) število, ki predstavlja količino `bobov`. Ker je Kurent, seveda ne sme iti kar tako, ampak mora upoštevati nekaj pravil:

- Pot po mreži se začne v zgornjem levem kotu in konča v spodnjem desnem kotu.
- Pot začne z 0 bobi in med potjo nikoli ne sme imeti negativnega števila bobov (četudi bi ob pristanku na lokaciji število bobov spet postalo nenegativno to ni sprejemljivo).
- Po mreži se lahko premika le en korak v desno, en krak navzdol ali en korak v desno in en korak navzdol (diagonalno) in ob vsakem takem koraku poje 1 bob.
- Enkrat v sprehodu lahko skoči na poljubno mesto (tudi nazaj na mesto, kjer stoji) v mreži, vendar ob takem skoku poje 3 bobe.
- Ko se kurent nahaja na nekem mestu v mreži pobere vse bobe, ki so na tem mestu (če je število bobov negativno se to odšteje od skupnega števila bobov).
- Če se kurent večkrat znajde na istem mestu, bo vsakič pobral vse bobe, ki so na tem mestu (medtem, ko je bil odsoten se bobi obnovijo).

Pomagajte kurentu poiskati pot, ob kateri konča z največjim številom bobov. Napišite funkcijo, ki vrne največje število bobov, ki jih lahko kurent pobere in seznam ukazov, ki jih mora izvesti, da jih pobere, pri tem `D` pomeni korak navzdol, `R` korak desno in `DI` korak diagonalno, `JUMP i j` pa skok na mesto i, j.

```
pustni_torek = [
    [10, 20, -30, -100, 9],
    [0, 0, 50, 50, 2],
    [10, 20, 300, -10, 0],
    [40, 30, -200, -10, 5],
]

print(najvec_bobov(pustni_torek))
# Izhod: (744, ['R', 'DI', 'D', 'JUMP 0 0', 'R', 'DI', 'D', 'R', 'DI'])
```