README

Nume: Gogea

Prenume: Mihail

Grupa: 324CC

Tema 2 Paradigme de programare (Haskell)

Titlu tema: **Învățare instrumentală**

Dificultate: 8.5/10

Timpul alocat acestei teme: 5-6 ore

Observatii:

Arhiva Gogea_Mihail_324CC.zip contine urmatoarele fisiere:

RL.hs

README

Tema terminata complet (100%) . Pe checker rezultatul este 75 asa cum spune si in enuntul temei, urmand ca la laborator sa mentionez si sa arat functionalitatea task-ului 3.

Am lucrat cu functionale , toate cele din Hint-uri + alte functii cautate si gasite pe hogle sau alte site-uri haskell, de asemenea am lucrat cu multe alte functii din biblotecile Data.List si Data.Array.

In interiorul sursei RL.hs sunt comentarii pentru a ajuta la intelegerea mai buna a functiilor auxiliare in a ma ajuta a rezolva functile principale.

Task I - Generarea cailor

Am rezolvat acest task complet , urmand informatiile din enuntul temei de la task-ul 1, comentariile asupra codului avand o explicatie mai explicita.

Functia auxiliara getVecin am creato pentru a ma ajuta a extrage un vecin in functie de cel de-al doilea argument care este un numar random , am impartit acest numar la lungimea de vecini a starii curente , urmand sa extrag un vecin random.

Task 2 + Task 3

Am rezolvat toate functile acestor task-uri , prima oara am rezolvat task2 in care nu trebuia sa diminuez rata de invatare , urmand apoi sa updatez datorita task-ului 3 si explicatilor de pe forum , ca trebuie sa updatez in functile de la task2 in asa fel sa se retina numarul de treceri printr-o stare , pentru a putea schimba rata de invatare .

La aceste functii am folosit ca se poate de mult toate functionalele invatate de la laborator , curs sau din documentatia de pe internet(foldl,map,accum,mapAccum etc.). Am exploatat de asemenea biblotecile Data.List si Data.Array.

Explicatii mai amanuntite sunt in interiorul sursei , comentariile fiind deasupra aproximativ oricarei functie auxiliara.

Am sa introduc in urmatoarele linii codul de la task2 , cel initial in care rata de invatare nu se diminua, deoarece in codul sursa RL.hs este task2 updatat la cerintele de la task3 :

```
--  === 2. Estimarea utilităților fără diminuarea ratei de învățare ===

{-
   *** TODO ***

   Array cu conscințele specifice fiecărei stări.
-}
reinforcements :: Array State Float
```

```
reinforcements = array (1,nStates) [(x, (if x == 8 then -1 else (if x == 12 then 1 else 0))) | x <- [1..nStates]]


{-
   *** TODO ***

   Valorile inițiale ale stărilor, înaintea rulării algoritmului.
   Se construiesc pe baza array-ului de consecințe.
-}
initialEstimation :: Estimation
initialEstimation = array (1,nStates) [(x, reinforcements ! x) | x <- [1..nStates]]


{-
   *** TODO ***

   Lista de utilități provenite dintr-o estimare.
-}
values :: Estimation -> [Float]
values est = [(est ! y) | y <- [1..nStates]]


{-
   *** TODO ***

   Reprezentarea sub formă de șir de caractere a unei estimări.
   Se va întrebuința forma bidimensională, ca în imaginile din enunț.
   De asemenea, utilitățile vor fi rotunjite la 2 zecimale, și vor
```

avea semnul inclus.

Hint: `Text.Printf`.

Exemplu de rezultat:

-0.07 +0.06 +0.20 +1.00

-0.20 +0.00 -0.43 -1.00

-0.32 -0.45 -0.56 -0.78

Pentru a vizualiza corect caracterele de linie nouă, aplicați

în interpretor funcția `putStrLn` asupra șirului obținut.

-}

functie1 es nr = (reverse (tail (reverse (foldl (₩x y -> x ++ (printf "%+.2f " (es ! y))) "" [nr..(nr + (width - 1))]))))

showEstimation :: Estimation -> String

showEstimation es = (reverse (tail (reverse (foldl (₩x y -> x ++ (functie1 es y) ++ "₩n") "" (reverse [1,(width + 1)..nStates])))))

{-

*** TODO ***

Actualizează o estimare în urmare parcurgerii unei căi.

```
    Hint: `Data.Array.accum`.
-}

formula v ve = (v +  (learningRate * (ve - v)))



updateEstimation :: Estimation -> Path -> Estimation
updateEstimation e cale = if (length cale) == 1 then e
 else (updateEstimation (accum (₩x y -> (formula x y)) e [(head cale, e ! (head (tail cale)))]) (tail cale))

{-
    *** TODO ***

    Obține un flux infinit de estimări rafinate succesiv, pe baza unui flux
    infinit de căi finite, încheiate în stări terminale.

    Hint: `Data.List.mapAccumL`.
-}
estimations :: [Path] -> [Estimation]
estimations infinitCai = snd (mapAccumL (₩acc x -> let acc1 = (updateEstimation acc x) in (acc1,acc1)) initialEstimation infinitCai)

{-
    *** TODO ***
```

```
    Determină estimarea de rang dat ca parametru, pe baza unui generator.
-}
--trunchiez o lista infinita de cai infinite la o lista infinita de cai terminate in stari terminale
trunchiere lista = [(terminatePath x) | x <- lista]

estimate :: RandomGen g => Int -> g -> Estimation
estimate rang g = (last (take rang (estimations (trunchiere (randomPaths g)))))

{-
    *** TODO ***

    Pentru o stare, determină vecinul cu cea mai mare valoare estimată.

    Hint: `Data.Function.on`.
-}
--perechi pentru o stare (stare,valoare)
--perechiStare :: Estimation -> State -> [(State,Float)]
perechiStare est stare = [(x, est ! x) | x <- (neighborsOf stare)]

bestNeighborOf :: State -> Estimation -> State
bestNeighborOf stare estimare = fst (foldl (₩x y -> if ((snd x) > (snd y)) then  x else  y) (head (perechiStare estimare stare)) (perechiStare estimare stare))

{-
    *** TODO ***
```

```
    Contruieşte o cale începută în starea inițială, pe principiul alegerii
    vecinului cu utilitata maximă.
-}

drumBun es start acc
 | (elem (bestNeighborOf start es) terminalStates) == True =  [(bestNeighborOf start es),start] ++ acc
 | otherwise = (drumBun es (bestNeighborOf start es) ([start] ++ acc))

bestPath :: Estimation -> Path
bestPath estim = (reverse (drumBun estim 1 []))
```