Paradigme de Programare
Tema 1 - Haskell

Termen de predare: **21.04** (soft)
28.04 (hard)

Responsabili temă: Sergiu Weisz
Adrian Dinu
Mihai Dumitru

Data publicării: *28.03*
Ultima actualizare: *22.04*

1 Introducere

Vișî încearcă să se deplaseze prin Europa, pentru a ajunge într-un oraș anume. El are la dispoziție o hartă pe care sunt marcate mai multe orașe, drumurile dintre acestea, precum și lungimea drumurilor. Pentru a trece printr-un oraș, Vișî trebuie să plătească din buzunar o taxă de intrare. În cadrul acestei teme, va trebui să-l ajutați pe Vișî să găsească cel mai scurt traseu pe care și-l permite financiar, de la orașul sursă la orașul destinație.

2 Cerințe

Considerând două variații ale problemei găsirii celui mai scurt traseu, se cere să implementați câte un algoritm eficient pentru rezolvarea acestora:

- În primul caz se ignoră costul de intrare în fiecare oraș, precum și suma inițială de bani. Trebuie deci să determinați cel mai scurt traseu de la orașul sursă la orașul destinație.
- În al doilea caz trebuie luate în considerare costurile de intrare în fiecare oraș și suma inițială de bani. Astfel, nu se poate trece printr-un oraș decât cu o sumă de bani mai mare sau egală cu taxa de intrare, care apoi se scade din suma deținută. Ținând cont de această restricție, trebuie să determinați cel mai scurt traseu **posibil** de la orașul sursă la orașul destinație. Dacă există mai multe astfel de trasee, trebuie identificat cel mai ieftin.

Orașele sunt reprezentate prin indici de la `1` la `n` (numărul total de orașe). Orașul sursă este mereu cel cu indicele `1`, iar cel destinație este mereu cel cu indicele `n`.

Important! Dacă există mai multe traseuri cu aceeași lungime (și același cost în cel de-al doilea caz), este de preferat primul în ordinea lexicografică a indicelor orașelor. Altfel spus, dacă într-un nod puteți ajunge în mod echivalent din mai multe noduri, salvați ca părinte nodul cu indicele mai mic.

Mai specific, trebuie să implementați următoarele două funcții:

```
solveSimple :: (Int, [(Int, Int, Int)])
               -> Maybe ([Int], Int)

solveCosts :: (Int, Int, [Int], [(Int, Int, Int)])
              -> Maybe ([(Int, Int)], Int)
```

1. Prima funcție, `solveSimple`, primește ca argument un tuplu, care constă în:

- **n**, numărul de orașe
- o listă de drumuri, sub forma unor triplete formate din orașele aflate la capete și de lungimea drumului

În cazul în care există o soluție, aceasta întoarce un constructor `Just` care încapsulează o pereche de:

- lungimea totală a traseului
- lista indicilor orașelor din cale, în ordinea în care apar în aceasta, de la orașul sursă la orașul destinație

2. Cea de-a doua funcție, `solveCosts`, primește ca argument un tuplu, care constă în:

- **n**, numărul de orașe
- **m**, suma inițială de bani
- o listă de **n** taxe de intrare în orașe (al i-lea element e taxa de intrare în orașul cu indicele i)
- o listă de drumuri, sub forma unor triplete formate din orașele aflate la capete și de lungimea drumului

În cazul în care există o soluție, aceasta întoarce un constructor `Just` care încapsulează o pereche de:

- lungimea totală a traseului
- o listă de perechi, formate din indicele unui oraș din cale și suma de bani rămasă după trecerea prin acesta (ordonate de la orașul sursă, la orașul destinație)

În cazul în care nu există soluție, ambele funcții trebuie să întoarcă o valoare `Nothing`.

3 Programare Dinamică Leneșă

O abordare bună pentru rezolvarea temei, este utilizarea tehnicii de programare dinamică, [1] profitând de principiile evaluării leneșe din Haskell. Această tehnică este descrisă și exemplificată aici [2]. O astfel de implementare este necesară pentru a obține punctajul maxim, după cum este specificat și în secțiunea Punctaj.

4 Teste

Următoarele două subsecțiuni descriu formatul fișierelor de input/output din setul de teste date. Acest format este relevant doar pentru a vă ajuta să înțelegeți în ce constau testele. Checkerul se ocupă de citirea și scrierea în și din fișiere; nu trebuie să implementați partea de IO.

4.1 Input

Există două tipuri de input, unul pentru fiecare din variațiile menționate.

Pentru primul caz, fără taxe de intrare în orașe, formatul fișierelor de input este următorul:

- prima linie: **n**, numărul de orașe
- a doua linie: **e**, numărul de drumuri inter-orașe
- următoarele **e** linii: câte trei întregi **(i, j, c)** reprezentând indicii orașelor aflate la capete și lungimea drumului

Pentru al doilea caz, cu taxe de intrare în orașe, formatul fișierelor de input este următorul:

- prima linie: **n**, numărul de orașe
- a doua linie: **e**, numărul de drumuri inter-orașe
- a treia linie: **m**, suma inițială de bani
- următoarele **n** linii: linia **i** conține taxa de intrare pentru orașul cu indicele **i**
- următoarele **e** linii: câte trei întregi **(i, j, c)** reprezentând indicii orașelor aflate la capete și lungimea drumului

4.2 Output

Există două tipuri de output, unul pentru fiecare din variațiile menționate.

Pentru primul caz, fără taxe de intrare, formatul fișierelor de output este următorul:

- primul rând: lungimea totală a traseului găsit
- pe fiecare din următoarele rânduri, indicele unui oraș din cale (ordonate de la orașul sursă la orașul destinație)

Pentru al doilea caz, cu taxe de intrare, formatul fișierelor de output este următorul:

- primul rând: lungimea totală a traseului găsit
- pe fiecare din următoarele rânduri, o pereche constând din indicele unui oraș din cale și suma de bani rămasă după trecerea prin acesta (ordonate de la orașul sursă, la orașul destinație)

În ambele cazuri, dacă nu există un traseu posibil, fișierul de output va conține doar șirul "Nothing".

5 Schelet de cod

Dacă doriți să rulați programul cu testele proprii, vă oferim un schelet de cod ce are implementată deja partea de citire și afișare. Pentru rularea scheletului de cod puteți folosi comanda de mai jos. Primul parametru reprezintă fișierul din care se face citirea, al doilea parametru este numele fișierului de ieșire al programului. Al treilea parametru al programului reprezintă pe ce tip de problemă vreți sa testați:

- simple - varianta fără costuri pe orașe
- costs - varianta ce include costuri pe orașe

Exemplu de rulare:

```
runhaskell Skel.hs test.in test.out simple
```

6 Resurse

În arhiva `"1-haskell.zip"`, se găsesc:

- fișierul "Checker.hs" folosit pentru testare
- folderul "tests", care conține testele publice, precum și rezultatele de referință
- fișierul "Tema1.hs", care conține scheletul de cod și în care trebuie să implementați rezolvările problemelor
- fișierul "Skel.hs", care conține funcții de ajutor pentru afișare

6.1 Checker

Pentru a vă testa tema, puteți rula fișierul "Checker.hs" cu unul din parametrii: `simple`, `costs`, `both`

Pentru a rula checkerul, puteți folosi `runghc` sau `runhaskell`. Exemplu:

```
runghc Checker.hs both
runhaskell Checker.hs simple
```

7 Trimitere

Tema trebuie trimisă sub forma unei arhive zip, care să conțină:

- fișierul "Tema1.hs" (cu definiția funcției `solveSimple` și a funcției `solveCosts`)
- fișier "README"
- orice alt fișier sursă Haskell de care aveți nevoie

8 Punctaj

Tema valorează în total **1.75 puncte** din nota finală, dintre care:

- 0.5 pentru testele simple, fără taxe de intrare
- 0.5 pentru testele cu taxe de intrare
- 0.25 pentru implementare folosind evaluare lenesă
- 0.25 pentru implementare folosind o structură de date cu acces arbitrar (e.g. `Data.Array`)
- 0.25 pe structura codului și conținut README

Testarea se va face automat.
Pentru fiecare zi de întârziere după deadline-ul soft există o depunctare de 0.1 puncte.

9 Linkuri utile

[1] Dynamic Programming
https://en.wikipedia.org/wiki/Dynamic_programming

[2] Lazy Dynamic Programming
http://jelv.is/blog/Lazy-Dynamic-Programming/

[3] Maybe and Either (Learn You a Haskell for Great Good)
http://learnyouahaskell.com/making-our-own-types-and-typeclasses

[4] Error handling in Haskell (School of Haskell)
https://www.schoolofhaskell.com/school/starting-with-haskell/basics-of-haskell/10_Error_Handling

[5] Error handling in Haskell (Real World Haskell)
http://book.realworldhaskell.org/read/error-handling.html#errors.maybe