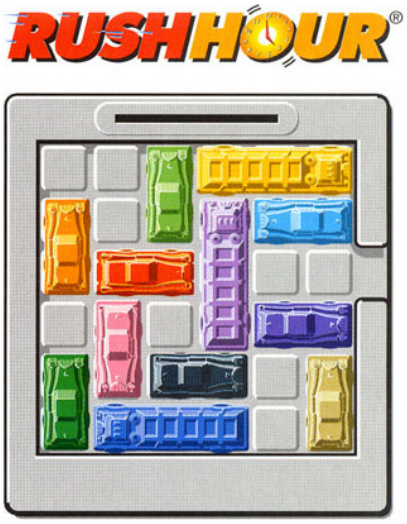
Ημερομηνία Ανάρτησης: 08/1/2018
Ημερομηνία Παράδοσης: -
Αρχές Γλωσσών Προγραμματισμού

Περιγραφή Προβλήματος

Στην εργασία αυτή καλείστε να υλοποιήσετε ένα πρόγραμμα σε Haskell που θα επιλύει το παιγνίδι *Rush Hour*. Το *Rush Hour* είναι ένα puzzle με συρόμενα blocks το οποίο παίζεται σε ένα ταμπλό 6×6. Τα blocks, τα οποία αναπαριστούν αμαξίδια, είναι διαστάσεων 1×2, 1×3, 2×1 ή 3×1. Τα αμαξίδια μπορούν να μετακινηθούν μπρος και πίσω παράλληλα στην μεγάλη τους διάσταση. Μια νόμιμη κίνηση αποτελείται από την μετακίνηση οποιουδήποτε αμαξιδίου οσοδήποτε μεγάλο πλήθος θέσεων, αρκεί να μην παρεμποδίζεται από άλλο αμαξίδιο και να μην βγει εκτός ταμπλό. Στόχος του παιγνιδιού είναι το κόκκινο αμαξίδιο να απεγκλωβιστεί και να βγει μέσω της εξόδου στο δεξί άκρο του ταμπλό. Το παιγνίδι μπορεί εύκολα να γενικευτεί σε ταμπλό $w \times h$ και αμαξίδια διαστάσεων $1 \times k$ για κάθε k, $2 \leq k \leq h$ και $k \times 1$ για κάθε k, $2 \leq k \leq w$.



Σχήμα 1. Rush Hour

Αναπαράσταση του παιγνιδιού σε ASCII

Για την αναπαράσταση μιας κατάστασης του παιγνιδιού σε ταμπλό $w \times h$ χρησιμοποιούμε μια συμβολοσειρά h γραμμών, η κάθε μία από τις οποίες περιέχει w χαρακτήρες. Σε κάθε θέση που δεν υπάρχει αμαξίδιο, στην συμβολοσειρά δηλώνεται με μία τελεία ('.'). Σε κάθε θέση που υπάρχει αμαξίδιο, στην συμβολοσειρά εμφανίζεται ένας χαρακτήρας ο οποίος ταυτοποιεί μοναδικά το αμαξίδιο. Για την ταυτοποίηση των αμαξιδίων μπορεί να χρησιμοποιηθεί οποιοσδήποτε χαρακτήρας ASCII εκτός από αυτόν της αλλαγής γραμμής ('\n') και της τελείας ('.'). Ειδικότερα το κόκκινο αμαξίδιο ταυτοποιείται πάντα από τον χαρακτήρα της ισότητας ('='). Η διεύθυνση του κόκκινου αμαξιδίου είναι πάντα οριζόντια και θεωρούμε την έξοδο του ταμπλό στο δεξί άκρο της γραμμής που βρίσκεται αυτό. Για παράδειγμα η κατάσταση της εικόνας αναπαρίσταται ως εξής:

```
"..abbb
 c.adee
 c==d
 .g.dhh
 igjj.k
 illl.k"
```

Σχήμα 2. Αναπαράσταση σε ASCII

Ζητούμενοι τύποι δεδομένων

Καλείστε να ορίσετε τους εξής τύπους:

- **State**

Ο τύπος δεδομένων *State* θα αποτελεί μια κατάσταση του παιγνιδιού. Θα πρέπει να εμπεριέχει τις διαστάσεις του ταμπλό και τις θέσεις όλων των αμαξιδίων.

- **Move**

Ο τύπος δεδομένων *Move* θα κωδικοποιεί μια κίνηση. Θα αποθηκεύεται ποιο αμαξίδιο και πως κινείται.

Έχετε πλήρη ελευθερία στον τρόπο που θα αποθηκεύουν εσωτερικά οι παραπάνω τύποι τις απαραίτητες πληροφορίες. Πρέπει όμως να σκεφτείτε πως οι δομές που θα χρησιμοποιήσετε θα βοηθήσουν την αποδοτική υλοποίηση των παρακάτω συναρτήσεων.

Ενδέχεται να χρειαστεί να ορίσετε σχέση διάταξης και ισότητας για τους παραπάνω τύπους δεδομένων.

Ζητούμενες συναρτήσεις

Καλείστε να ορίσετε τις εξής συναρτήσεις:

- **readState** :: **String -> State**

Η συνάρτηση *readState* θα δέχεται μια συμβολοσειρά με το συντακτικό που ορίζεται εδώ και θα επιστρέφει την κατάσταση του παιγνιδιού.

- **writeState** :: **State -> String**

Η συνάρτηση *writeState* είναι η "αντίστροφη" της *readState*. Δεδομένου μιας κατάστασης παιγνιδιού επιστρέφει την αντίστοιχη συμβολοσειρά.
Παράδειγμα: `writeState (readState "..abbb\nc.adee\nc==d..\n.g.dhh\nigjj.k\n illl.k\n") = "..abbb\nc.adee\nc==d..\n.g.dhh\nigjj.k\nilll.k\n"`

- **successorMoves** :: **State -> [(Move, Int)]**

Η συνάρτηση *successorMoves* βρίσκει όλες τις νόμιμες κινήσεις που μπορούν να γίνουν στην δεδομένη κατάσταση παιγνιδιού. Επιστρέφει μια λίστα με τις κινήσεις και τα κόστη τους. Για το συγκεκριμένο παιγνίδι θεωρούμε το κόστος κάθε κίνησης ίσο με 1.

- **makeMove** :: **State -> Move -> State**

Η συνάρτηση *makeMove* θα πραγματοποιεί μια κίνηση στην κατάσταση παιγνιδιού που δέχεται και θα επιστρέφει την νέα κατάστασή που προκύπτει. Στην περίπτωση που η δοθείσα κίνηση δεν είναι νόμιμη, η τιμή της συνάρτησης είναι απροσδιόριστη.
Παραδείγμα:

```
> state = readState "...a\n==.a\n....\n....\n"
> moves = [move|(move,cost)<-successorMoves state]
> length moves
3
> successorStates = map (makeMove state) moves
> map writeState successorStates
["...a\n.==a\n....\n....\n","....\n==.a\n...a\n....\n",
"....\n==..\n...a\n...a\n"]
```

• **finalState :: State -> Bool**

Η συνάρτηση *finalState* ελέγχει αν μια κατάσταση είναι τελική ή όχι, δηλαδή αν το κόκκινο αμαξίδιο έχει φτάσει στην έξοδο.
Παραδείγματα: `finalState (readState "...\n==.\n...")=False` και
`finalState (readState "...\n.==\n...")=True`

• **solve :: State -> [Move]**

Η συνάρτηση *solve* επιλύει το παιγνίδι επιστρέφοντας μια ακολουθία κινήσεων που πρέπει να γίνουν ώστε από την δεδομένη κατάσταση να βρεθούμε σε μια τελική. Θα πρέπει να πραγματοποιεί απληροφόρητη (uninformed) αναζήτηση, δηλαδή να μην χρησιμοποιεί καμία επιπλέον πληροφορία για τις καταστάσεις, πέρα από αυτές που παρέχονται από τον ορισμό του προβλήματος (αρχική κατάσταση, τελικές καταστάσεις και συνάρτηση διαδόχων καταστάσεων). Επιπλέον η συνάρτηση θα πρέπει να δουλεύει ανεξάρτητα της εσωτερικής δομής των τύπων *State* και *Move*. Αυτό σημαίνει ότι επανορίζοντας τους τύπους δεδομένων *State* και *Move* και τις συναρτήσεις *successorMoves*, *makeMove* και *finalState*, η *solve* θα πρέπει να είναι σε θέση να λύσει οποιοδήποτε άλλο πρόβλημα αναζήτησης.
Παραδείγματα:

```
> :{
| printSolution s [] = putStrLn (writeState s)
| printSolution s (m:ms) = do {putStrLn (writeState s);
|                               printSolution (makeMove s m) ms}
| :}
> state = readState "...a\n==.a\n....\n....\n"
> printSolution state (solve state)
...a
==.a
....
....

....
==..
...a
...a

....
..==
...a
...a
```

```
> state = readState "..abbb\n..a.c.\n..==c.\n....d.\n....d.\n....d.\n"
> printSolution state (solve state)
..abbb
..a.c.
```

```
..==c.
....d.
....d.
....d.

..abbb
..a.c.
==..c.
....d.
....d.
....d.

...bbb
....c.
==..c.
....d.
..a.d.
..a.d.

.bbb..
....c.
==..c.
....d.
..a.d.
..a.d.

.bbbc.
....c.
==....
....d.
..a.d.
..a.d.

.bbbc.
....c.
....==
....d.
..a.d.
..a.d.
```

Bonus Ζητούμενα (+20%)

• **heuristic :: State -> Int**

Η *heuristic* θα είναι μια συνάρτηση εκτίμησης του κόστους (του πλήθους των κινήσεων) της φθηνότερης διαδρομής προς μια τελική κατάσταση. Μια καλή και γρήγορη αξιολόγηση της κάθε κατάστασης μπορεί να βελτιώσει σημαντικά τον χρόνο της αναζήτησης, καθώς επεκτείνονται πρώτα οι κόμβοι που πλησιάζουν σε μια τελική κατάσταση. Πρέπει να ισχύει `finalState s=True` ⇔ `heuristic s=0`.

• **solve_astar :: State -> [Move]**

Η συνάρτηση *solve_astar* θα επιλύει το παιγνίδι, αντίστοιχα με την *solve*, αλλά αντί για απληροφόρητη αναζήτηση θα εκτελεί τον αλγόριθμο A*, με ευρετική συνάρτηση την *heuristic*. Για την υλοποίηση του A* θα χρειαστείτε να ορίσετε έναν νέο τύπο δεδομένων *PriorityQueue* που θα υλοποιεί μια ουρά προτεραιότητας με την χρήση της δομής pairing heap.

Παρατηρήσεις

1. Στην περίπτωση που υλοποιήσετε το bonus τμήμα της εργασίας η *solve* μπορεί να υλοποιηθεί ακριβώς όπως η *solve_astar* με ευρετική συνάρτηση την $(\lambda x.0)$.
2. Ενθαρρύνεται η χρήση βιβλιοθηκών που παρέχει ο μεταγλωττιστής ghc και ειδικά δομών δεδομένων όπως η *Data.Set* και η *Data.Map*. Απαγορεύεται η χρήση πακέτων της Haskell που δεν περιλαμβάνει ο ghc.
3. Στην περίπτωση που έχετε χρησιμοποιήσει κάποιο τμήμα κώδικα από το Internet, από κάποιο βιβλίο κλπ, πρέπει να αναφέρετε ρητά την πηγή του.

Παράδοση Άσκησης

Η άσκηση μπορεί να γίνει από ένα άτομο ή μια ομάδα δύο ατόμων (προτείνεται). Στη δεύτερη περίπτωση θα παραδοθεί **μόνο** από το ένα άτομο της ομάδας.

Τα ονόματα των συναρτήσεων που θα δημιουργήσετε πρέπει να είναι **ακριβώς** τα ίδια με αυτά που καθορίζονται από την παραπάνω εκφώνηση.

Θα πρέπει να παραδώσετε ένα αρχείο της μορφής sdiYY00NNN_sdiYY00NNN.zip με όνομα τους αριθμούς μητρώου σας. Το zip θα περιέχει **ένα** αρχείο rush_hour.hs με τις λύσεις σας για τα παραπάνω ζητούμενα και ένα αρχείο README.pdf που θα περιγράφει την υλοποίηση σας, θα επεξηγεί τις επιλογές σας για τον τρόπο που δομήσατε τους τύπους *State* και *Move* και θα αναφέρει τυχόν παραδοχές που κάνατε.

Η ημερομηνία παράδοσης θα ανακοινωθεί σύντομα. Το παραδοτέο αρχείο .zip θα το στείλετε με email στις παρακάτω διευθύνσεις: sdi1400016@di.uoa.gr, sdi1400221@di.uoa.gr και prondo@di.uoa.gr. Ερωτήσεις σχετικά με την άσκηση θα πρέπει να απευθύνονται στα πρώτα δύο emails (Σπύρος Αυλωνίτης και Γιάννος Χατζηαγάπης).