Σταματελόπουλος Νικόλαος
Α.Μ. : 03116138

Γλώσσες Προγραμματισμού ΙΙ
Άσκηση 3 (Erlang για ραλίστες)

Στο αρχείο rally.erl έχει υλοποιηθεί μια απλή λύση της άσκησης με πολυπλοκότητα O(Units*min((Acceleration+Brake)/10, 24)*24^2) όπου 24 είναι η μέγιστη δυνατή ταχύτητα που μπορεί να αναπτύξει ένα όχημα, δια 10 (δεδομένου ότι όλα τα μεγέθη είναι πολλαπλάσια του 10 αυτή η μετατροπή απλουστεύει την διαδικασία επίλυσης ως προς τις πράξεις), και αφού ο όρος min(_,_) είναι φραγμένος στο 24, προκύπτει O(Units) γραμμικός ως προς το μήκος της εισόδου/πίστας.

- ***make_track(Track)***

Παίρνει ως είσοδο μια λίστα από τούπλες της μορφής {πλήθος units, speed limit}, που περιγράφουν την πίστα ως μια σειρά από units/τμήματα με ένα αντίστοιχο ασφαλές όριο ταχύτητας. Έξοδος είναι μια λίστα με elements ακεραίους αριθμούς που αντιστοιχούν στα όρια ταχύτητας για κάθε unit με σειρά όμως από το τέλος προς την αρχή της πίστας (αντιστροφή γίνεται αλλού).

`prop_track_length()`
Ορίζουμε λοιπόν ως property test ότι μήκος της τελικής λίστας εξόδου πρέπει να ισούται με το άθροισμα του πλήθους units σε όλες τις τούπλες της λίστας εισόδου. Ουσιαστικά για τυχαία Inputs βλέπουμε αν το μήκος του αποτελέσματος ισούται με το άθροισμα των πρώτων στοιχείων όλων των tuples της λίστας εισόδου.
(το τελευταίο υλοποιείται με χρήση της `lists:foldl(fun({X,_}, Sum) -> X+Sum end, 0, Track)`)

- ***perform_move(Track, Speed)***

Λαμβάνει ως είσοδο μία πίστα/track (λίστα από διαδοχικά speed limits) και προσπαθεί να προχωρήσει κατά Speed units αν αυτό είναι εφικτό επιστρέφοντας το υπόλοιπο τμήμα του track μετά την κίνηση αυτή. Αν στην πορεία για τη δοσμένη ταχύτητα παραβιάζεται κάποιο όριο ταχύτητας τότε επιστρέφει ‘error’.

`prop_perform_move_length()`
Για τυχαίους συνδυασμούς εισόδου λίστας-ταχύτητας ελέγχουμε ότι η συνάρτηση είτε επιστρέφει error είτε μια λίστα με μήκος μικρότερο της εισόδου κατά Speed (ή επιστρέφει κενή λίστα, δηλαδή τερμάτησε). Δεδομένου ότι δε μας εξασφαλίζει ότι η επιλογή μεταξύ των δύο εξόδων είναι σωστή θα δημιουργήσουμε άλλα δύο tests.

`prop_check_perform_move_success()`
Για μια τυχαία ταχύτητα ως είσοδο (από 1 ως 24 δεδομένου ότι έχει διαιρεθεί δια του 10) δίνουμε επίσης μια λίστα τυχαίου μήκους με όλα τα στοιχεία να είναι speed limits μεγαλύτερα ή ίσα του Speed, οπότε και περιμένουμε ως έξοδο τη δεύτερη περίπτωση που αναφέραμε στο προηγούμενο τεστ.

`prop_check_perform_move_fail()`
Για μια τυχαία ταχύτητα ως είσοδο δημιουργούμε μια λίστα από speed limits μεγαλύτερα ή ίσα της ταχύτητας αυτής και μήκους Speed-1. Έπειτα προσθέτουμε ένα limit ίσο με Speed-1, ανακατεύουμε την τελική λίστα και την τροφοδοτούμε ως είσοδο στη perform_move(Track, Speed) περιμένοντας να αποτύχει επιστρέφοντας error δεδομένου ότι ένα από τα επόμενα Speed units που θα δει έχει όριο μικρότερο της τρέχουσας ταχύτητας.

- ***check_speed(Track, Speed, B)***

Δέχεται μια λίστα με όρια ταχύτητας, μια τρέχουσα ταχύτητα και μέγιστη δυνατή επιβράδυνση. Αναδρομικά κάνει βήματα με χρήση της perform_move() μειώνοντας την ταχύτητα κατά B κάθε φορά. Αν φτάσει η ταχύτητα μικρότερη ή ίση του 1 τελικά επιστρέφει true αλλιώς αν κάποια στιγμή επιστρέψει error η perform_move() τότε επιστρέφει false. Εφαρμόζει ένα απλό κριτήριο για έλεγχο αν μια κίνηση από ένα σημείο της πίστας με μια συγκεκριμένη ταχύτητα είναι έγκυρη (αν φτάσει ταχύτητα 1 τότε μπορεί να τερματίσει προφανώς διατηρώντας αυτή).

Out of ideas or time or both…

- ***find_max_possible_speed(Track, Speed, A, B)***

Για μια δεδομένη πίστα, μια ταχύτητα που αντιστοιχεί στην τελευταία κίνηση που πραγματοποιήθηκε επιτυχώς στο προηγούμενο βήμα και μέγιστη επιτάχυνση και επιβράδυνση, βρίσκει και επιστρέφει τη μέγιστη δυνατή ταχύτητα που μπορεί να χρησιμοποιηθεί στο επόμενο βήμα.

`prop_max_possible_speed()`

Για τυχαίες αρχικές συνθήκες (τυχαίος συνδυασμός των 4 μεταβλητών που δέχεται) ελέγχουμε ότι δεδομένου ότι υπάρχει λύση, δηλαδή εφόσον η κίνηση με ταχύτητα Speed-B που είναι και η ελάχιστη δυνατή είναι και έγκυρη (`?IMPLIES(check_speed(Track,Speed-B,B),....)`) θα πρέπει να ισχύουν οι παρακάτω συνθήκες για την έξοδο την έξοδο της συνάρτησης:

1. η ταχύτητα που επιστρέφει να είναι εντός του [Speed-B, Speed+A] αφού έγκειται στους περιορισμούς των κινήσεων όπως ορίζονται από την άσκηση.
2. η ταχύτητα που επιστρέφει να είναι έγκυρη ως επόμενη κίνηση, δηλαδή αν δοθεί ως είσοδο στην check_speed(Track, Speed, B) να επιστραφεί true.
3. η αμέσως μεγαλύτερη ταχύτητα από αυτή που επιστρέφει (Speed+1) να παραβιάζει τουλάχιστον μία από τις παραπάνω δύο συνθήκες, καθυστώντας την μη έγκυρη ως επόμενη κίνηση.

- ***race(Track, Speed, A, B, Moves)***

Αναδρομική συνάρτηση που καλεί την find_max_possible_speed() και την perform_step() και αυξάνοντας το πλήθος των βημάτων που εκτελούνται από τη μία αναδρομική κλήση της στην επόμενη. Δεν έχει κάποια σημαντική ανεξάρτητη από το υπόλοιπο πρόγραμμα αξία οπότε και δεν ορίσαμε κάποιο property based test.

- ***rally(Acc, Br, Track)***

Από τα παραπάνω εξασφαλίσαμε ότι τα διάφορα βήματα που ακολουθούνται εντός της δοσμένης πίστας/αγώνα είναι έγκυρα και δεν παραβιάζουν τα όρια που ορίζονται στα διαφορετικά units. Δεν αιτιολογήσαμε όμως πως η greedy επιλογή του επόμενου βήματος οδηγεί πάντα στο βέλτιστο δυνατό αποτέλεσμα.

Για το σκοπό αυτό μπορούμε να τεστάρουμε το τελικό μας πρόγραμμα με ένα άλλο το οποίο πραγματοποιεί εξαντλητική αναζήτηση για τη βέλτιστη λύση επαληθεύοντας έτσι ότι το αποτέλεσμα που επιστρέφει είναι πάντα το βέλτιστο δυνατό.
Είχε προηγηθεί η ιδέα για υλοποίηση ενός Dynamic Programming Algorithm που εκτελεί εξαντλητική αναζήτηση με χρήση πίνακα (Units+24) x 24 όπου κάθε κελί DP[i][j] θα είναι ο ελάχιστος αριθμός κινήσεων που απαιτείται για να τερματίσει την πίστα αν βρίσκεσαι στο i-οστό unit με στο οποίο κατέληξες με ταχύτητα j από το προηγούμενο βήμα. Οι τελευταία 24 σειρές είναι οι 24 θέσεις μετά το τελευταίο unit της πίστας δεδομένου ότι δεν έχουμε συγκεκριμένο όριο ταχύτητας στον τερματισμό και μπορούμε να τον υπερβούμε στο τελευταίο βήμα. Έτσι το τελευταίο 24 x 24 κελιά του πίνακα αρχικοποιούνται σε 0 κινήσει αφού είναι ήδ ηστο τέλος και μετά αρχίζει να σκανάρει κάθε unit από το τέλος προς την αρχή ο αλγόριθμος συμπληρώνοντας

κατάλληλα τις κινήσεις για το unit προς μελέτη και για όλες τις πιθανές ταχύτητες άφιξης σε αυτό. Για λόγους απλότητας δεν εκτελέστηκαν test σε αυτό πέραν του μερικών eunit test για επιβεβαίωση της ορθότητας του. Η μόνη αλλαγή πυο χρειαστηκε στον πίνακα για λόγους επίδοσης σε erlang είναι να χρησιμοποιηθεί μια λίστα από 24 λίστες/σειρές (δεδομένου ότι ερευνούνται το πολύ τα 24 επόμενα unit σε κάθε κίνηση) που η κάθε μία έχει με τη σειρά και ένα λιγότερο στοιχείο (από ένα unit θες S ταχύτητα για να φτάσεις ακριβώς S units πιο μετά όπου και θα έχεις πλέον ταχύτητα άφιξης τουλάχιστον S για κάθε επόμενο προς έρευνα unit καταλήγοντας σε ένα αυτοσχέδιο πίνακα με σχήμα τριγώνου). Ο αλγόριθμος μπορεί να είναι O(Units*24^2) αλλά η δημιουργία, ανανέωση και πέρασμα του πίνακα στις αναδρομικές κλήσεις φαίνεται στη γενική περίπτωση να έχει μεγαλύτερο χρόνο εκτέλεσης από τον greedy αλγόριθμο κι ας είναι O(Units) τελικά και οι δύο.

`prop_check_rally()`
Για τυχαίο συνδυασμό εισόδου {Acceleration, Brake, Track} ελέγχουμε ότι και οι δύο αλγόριθμοι επιστρέφουν τελικά το ίδιο αποτέλεσμα. Ο τελευταίος είναι εξαντλητικής αναζήτησης μέσω DP οπότε και αρκεί για επαλήθευση της ορθότητας του πρώτου.

Όλα τα παραπάνω μπορούν να εκτελεστούν κατόπιν μεταγλώττισης του rally.erl τρέχοντας
proper:quickcheck(rally:<property_test_name>()).
ενώ μπορεί προφανώς να δοθεί ως δεύτερο όρισμα στην quickcheck και ο αριθμός των επιθυμητών test που θέλουμε να γίνουν.

Το αρχείο επιπλέον περιέχει μερικά ‘demo’ eunit tests χωρίς κάποια ιδιαίτερη δυσκολία ή αξία αναφοράς στο καθένα ξεχωριστά καθώς είναι απλά ένα σύνολο από συγκρίσεις επιθυμητής εξόδου με πραγματική έξοδο μιας συνάρτησης για κάποιες συγκεκριμένες παραμέτρους. Μπορούν να εκτελεστούν όλα μαζί με eunit:test(rally).