Σπίθας Ευάγγελος: 1115201500147

Χαραλάμπους Παναγιώτης: 1115201500174

Οδηγίες για να τρέξετε το πρόγραμμα:

- ghci –XFlexibleInstances rush_hour.hs
- let s=readState(είσοδος της επιλογής σας)
- printSolution s (solve s)\ printSolution s (solve_astar s)

State:

>>Ο τύπος State αποτελείται από έναν ακέραιο που δηλώνει τον αριθμό των γραμμών του (h-height),έναν ακέραιο που δηλώνει τον αριθμό των στηλών του(w-width), μια λίστα από strings που αποτελεί ουσιαστικά μια οπτικοποίηση της κατάστασης του παιχνιδιού(grid),έναν ακόμη ακέραιο (pinRow), που δηλώνει την γραμμή του grid στην οποία βρίσκεται το αμαξίδιο "=" και μας διευκολύνει στον έλεγχο τερματισμού του παιχνιδιού, και τέλος, ένα Map(Car-Map) που αντιστοιχίζει ονόματα αμαξιδίων με αμαξίδια. Έχουν οριστεί επίσης και οι συναρτήσεις για την ανάκτηση των παραπάνω στοιχείων-μελών ενός state.

>>Η συνάρτηση **finalState** χρησιμοποιεί το προαναφερθέν στοιχείο pinRow ενός State, και απλά τσεκάρει αν στην τελευταία θέση του βρίσκεται το σύμβολο "=".

Πριν την συνέχεια επεξήγησης του State,είναι χρήσιμο εδώ να αναλυθεί το datatype Car:

Car:

>>Ο τύπος Car αποτελείται από έναν χαρακτήρα που είναι το όνομά του αμαξιδίου(name), ένα tuple ακεραίων που δείχνει το σημείο πάνω στο grid στο οποίο βρίσκεται η "αρχή" του αμαξιδίου (δηλαδή το πίσω μέρος του-(row,col)) ένας ακέραιος που δηλώνει το μέγεθος του αμαξιδίου στην μεγάλη του διάσταση (size) και έναν χαρακτήρα που δηλώνει αν το αμαξίδιο κινείται οριζόντια ή κάθετα (direction). Χρειαζόμαστε έναν μόνο ακέραιο για το μέγεθος του αμαξιδίου, αφού αυτό είτε θα είναι διαστάσεων 1*K είτε K*1, με K>=2. Έχουν οριστεί επίσης και οι συναρτήσεις για την ανάκτηση των παραπάνω στοιχείων-μελών ενός Car.

>>Η συνάρτηση CreateCarsList δημιουργεί από το αρχικό δοσμένο από τον χρήστη String, μια λίστα με όλα τα name των αμαξιδίων.

>>Η συνάρτηση traceCar, μαζί με την dirSize και την CarSize, δημιουργεί ουσιαστικά ένα Car,δοσμένου του name του, ανακτώντας τις απαραίτητες πληροφορίες από το grid ενός state.

>>Η συνάρτηση correlateCars δημιουργεί μια λίστα με tuples αντιστοίχισης ονόματος αυτοκινήτου με αυτοκίνητο.

>>Η συνάρτηση ChangePos, δοσμένων 2 car A,B αντικαθιστά στο πρώτο την πληροφορία του δεύτερου σχετικά με το σημείο που βρίσκεται η "αρχή" ενός Car (row-col tuple).

>>Η συνάρτηση changeCar (χρησιμοποιείται από την makeMove), δοσμένου ενός Car και ενός ακεραίου stp που αναπαριστά των αριθμό των βημάτων που θα εκτελέσει το Car, τροποποιεί κατάλληλα την πληροφορία (row,col) του αυτοκινήτου ,ανάλογα με το αν το αυτοκίνητο είναι οριζόντιο ή κάθετο στον χώρο του παιχνιδιού (direction).

Πλέον, μπορούμε να συνεχίσουμε την επεξήγηση του State:

>>Η συνάρτηση **readState** δέχεται ένα string από τον χρήστη και δημιουργεί ένα state με τα στοιχεία που αφορούν αμιγώς το state(μέγεθος,grid,pinRow) καθώς και με τις πληροφορίες τις σχετικές με τα αμαξίδια, χρησιμοποιώντας τις createCarsList, traceCar, correlateCars και δημιουργώντας τελικά ένα Map.

>>Η συνάρτηση **writeState** αντίθετα, ανακτά το grid ενός state,προσθέτει στο τέλος κάθε γραμμής τον χαρακτήρα \n και αναδρομικά συνενώνει και επιστρέφει το αποτέλεσμα.

>>Η συνάρτηση writeDots γενικά, αντικαθιστά την εμφάνιση ενός Car πάνω σε ένα grid. Δηλαδή, οπουδήποτε εμφανίζεται ο χαρακτήρας που αντιστοιχεί στο εν λόγω Car,αντικαθίσται από μια τελεία ('.'). Ορίζεται για οριζοντίως κινούμενα Car(ξεκίνα να αντικαθιστάς χαρακτήρες σε μια γραμμή μόλις συναντήσεις το ζητούμενο χαρακτήρα και σταμάτα μόλις αντικαταστάσεις Size(Car) χαρακτήρες), καθώς και για καθέτως κινούμενα Car (η διαφορά εδώ είναι πως αντικαθιστούμε χαρακτήρες σε συγκεκριμένη θέση από κάποιες γραμμές του grid, αντί για αντικατάσταση σε μία γραμμή μόνο.

>>Η συνάρτηση WriteChar αντίθετα, όταν ορίζεται για οριζοντίως κινούμενα Car,ξεκινά από την αρχή μιας γραμμής και φτάνοντας σε μια συγκεκριμένη θέση, ξεκινά να αντικαθιστά ένα συγκεκριμένο αριθμό χαρακτήρων (που έχουμε φροντίσει να είναι όλοι ίσοι με '.')με το name ενός Car. Όταν πάλι ορίζεται για καθέτως κινούμενα Car, αντικαθιστά σε συγκεκριμένη θέση κάποιων γραμμών του grid το '.' με το name.

>>Η συνάρτηση ChangeStateList(χρησιμοποιείται από την makeMove) ,μεταβάλλει το grid μιας κατάστασης αναλόγως της κίνησης ενός Car. Εξετάζει το direction του Car και καλεί τις συναρτήσεις writeChar και writeDots με τα κατάλληλα ορίσματα. Αν πρόκειται για κίνηση οριζόντιου Car, καλείται αμέσως μετά η συνάρτηση replace για την αντικατάσταση της νέας γραμμής πάνω στο grid. Αν πρόκειται για κάθετη κίνηση, το grid έχει ήδη υποστεί επεξεργασία από τις writeDots,writeChar.

MOVE:

>>Ο τύπος Move μπορεί να είναι είτε Nil (δες constructSolution) είτε ένας χαρακτήρας και ένας ακέραιος. Ο χαρακτήρας δείχνει το αυτοκίνητο που κινείται, ενώ ο ακέραιος δείχνει τον αριθμό των βημάτων, τα οποία γίνονται στην μεγάλη διάσταση του αυτοκινήτου. **Κατά σύμβαση, η κίνηση των κατακόρυφων αυτοκινήτων κωδικοποιείται με αρνητικούς ακεραίους για προς τα πάνω κινήσεις και με θετικούς για προς τα κάτω**. **Αντίστοιχα, στα**

οριζόντια θετικοί για προς τα δεξιά και αρνητικοί για προς τα αριστερά κινήσεις. Ο τύπος Move, έχει κατ'αυτόν τον τρόπο αποθηκευμένα μόνο τα απολύτως απαραίτητα στοιχεία. Αν δεν υπήρχε είτε ο χαρακτήρας είτε ο ακέραιος, δεν θα μπορούσε να προσδιοριστεί επαρκώς το Move.

>>Η συνάρτηση CheckMoveRight, για ένα δεδομένο state και ένα δεδομένο χαρακτήρα (ο οποίος έχουμε διασφαλίσει ότι αντιστοιχεί σε Car που κινείται οριζόντια) βρίσκει το πλήθος των "θετικών" κινήσεων που μπορεί να εκτελέσει το Car στην γραμμή του.

>>Αντίστοιχα για "αρνητικές" κινήσεις η checkMoveLeft.

>>Η checkMoveHorizontal δέχεται σαν όρισμα ένα state και ένα Car που κινείται οριζόντια και τσεκάρει για αυτό το to Car το πλήθος των κινήσεων που μπορεί να εκτελέσει, συνενώνοντας σε μια λίστα τα αποτελέσματα των checkMoveRight και CheckMoveLeft.

>>Οι συναρτήσεις CheckMoveUp και CheckMoveDown βρίσκουν αντίστοιχα όλες τις δυνατές κινήσεις για τα αυτοκίνητα που κινούνται κάθετα.

>> Η checkMoveVertical δέχεται σαν όρισμα ένα state και ένα Car που κινείται κάθετα και τσεκάρει για αυτό το to Car το πλήθος των κινήσεων που μπορεί να εκτελέσει, συνενώνοντας σε μια λίστα τα αποτελέσματα των checkMoveUp και CheckMoveDown.

>>Η validCarMoves παίρνει σαν ορίσματα ένα State και μια λίστα από Car και επιστρέφει μια λίστα με όλες τα δυνατά Moves και τα κόστη τους πάνω στο συγκεκριμένο State.

>>Η **successorMoves** επιτελεί την λειτουργία που αναφέρεται στην εκφώνηση, χρησιμοποιώντας την validCarMoves. **Εδώ ,φαίνεται η χρησιμότητα του τρόπου δόμησης του State,καθώς πρέπει δοσμένου ενός State να μπορούμε να ανακτήσουμε πληροφορία σχετικά με τις διαστάσεις του, αλλά και σχετικά με τα αυτοκίνητα που υπάρχουν σε αυτό, μαζί με τα μεγέθη, την θέση και τον τόπο κίνησής τους.**

>>Η συνάρτηση takeSteps ανακτά τον αριθμό των βημάτων ενός Move.

>>Η συνάρτηση takeCarFromMove ανακτά τον χαρακτήρα που αναπαριστά το κινούμενο Car.

>>Η **makeMove** επιτελεί την λειτουργία που αναφέρεται στην εκφώνηση. Δοσμένου ενός State, δημιουργεί ένα νέο state το οποίο έχει διαφορετικό grid (χρήση της changeStateList που αναφέρεται πιο πάνω) και διαφορετικό Car-map (χρήση της ChangeCar που αναφέρεται πιο πάνω και στην συνέχεια εισαγωγή του νέου Car στο Car-Map. Το ήδη υπάρχων Car με το ίδιο key αντικαθίσταται από το νέο). Το αυτοκίνητο που κινείται και ο αριθμός των βημάτων μαθαίνονται με τις takeSteps και takeCarFromMove από το δοσμένο Move.

Περνάμε τώρα, στην περιγραφή του τρόπου επίλυσης του παιχνιδιού:

>>Η συνάρτηση makeMap δέχεται σαν ορίσματα ένα State s1,μια λίστα από States [s] και μια λίστα από Moves [m] ίσου μεγέθους. Δημιουργεί και επιστρέφει ένα Map με Key το s[i] και value ένα tuple (s1,m[i]). Το λογικό νόημα του Map που δημιουργεί η makeMap είναι η αντιστοίχιση ενός State s' (key) με το State s από το οποίο προήλθε καθώς και με το Move από το οποίο προέκυψε η s' από την s.

>>Η συνάρτηση constructSolution δέχεται σαν όρισμα ένα State s1 που είναι τελικό (finalState) και ένα Map που έχει δημιουργηθεί από την makeMap (path).Η συνάρτηση αυτή ψάχνει στο path το State s1 ως key,προσαρτά στην λίστα επιστροφής την κίνηση που οδηγεί στην s1,και καλείται αναδρομικά με όρισμα την s0 που οδήγησε στην s1. **Η constructSolution τερματίζει όταν βρει κατάσταση στην οποία οδηγηθήκαμε από Move Nil, το οποίο έχουμε ορίσει κατά σύμβαση ότι αντιστοιχεί σε μια αρχική κατάσταση.** Η constructSolution, επιστρέφει τελικά την λίστα των κατάλληλων κινήσεων μέχρι την λύση ανεστραμμένη.

>>Η συνάρτηση solver δέχεται σαν όρισμα μια λίστα(unexplored) που αναπαριστά τα μη εξερευνημένα States και αρχικά περιέχει μόνο το αρχικό State του παιχνιδιού , ένα set(explored) που αναπαριστά τα εξερευνημένα States και αρχικά είναι κενό και ένα Map από State σε (State,Move) (το path που δημιουργεί η makeMap) το οποίο αρχικά περιέχει την αρχική κατάσταση σε αντιστοίχιση με τον εαυτό της και την κίνηση Nil. Αν η λίστα unexplored είναι άδεια, σημαίνει πως δεν μπορούμε να εξερευνήσουμε άλλα States και επιστρέφεται η κενή λίστα. **Πρόκειται για την περίπτωση που το παιχνίδι είναι μη επιλύσιμο και σαν αποτέλεσμα επίλυσης επιστρέφει μόνο το αρχικό State.** Σε διαφορετική περίπτωση παίρνουμε το πρώτο στοιχείο του unexplored,έστω cs, και αν αυτό είναι finalState επιστρέφουμε το αποτέλεσμα της constructSolution. Εάν το cs δεν είναι τελική κατάσταση, και ανήκει στο set explored απλά απορρίπτουμε το cs ως ήδη εξερευνημένο και αναδρομικά καλούμε την solver για τα υπόλοιπα States(tail unexplored). Αν δεν ανήκει στο explored,παράγουμε όλες τα States παιδιά του cs μέσω των συναρτήσεων successorMove και makeMove, τοποθετούμε τα παιδιά στο path με την makeMap, τοποθετούμε επίσης τα παιδιά στο τέλος της λίστας unexplored, και το cs στο set explored. Η solver καλείται αναδρομικά μέχρι να συναντήσουμε κάποιο finalState. Ουσιαστικά, υλοποιούμε μια αναζήτηση bfs σε γράφο.

>>Η συνάρτηση **solve** καλεί την solver με τα κατάλληλα ορίσματα τα οποία αυτή πρέπει αρχικά να έχει και αναφέρονται παραπάνω, και αντιστρέφει την λίστα που της επιστρέφεται.

Για την πιο αποδοτική επίλυση του παιχνιδιού υλοποιείται και ο αλγόριθμός solve_astar και η ευρετική συνάρτηση heuristic.

>>Η συνάρτηση **heuristic** εκτιμά το κόστος από ένα δεδομένο State μέχρι ένα finalState, συνυπολογίζοντας α)την απόσταση του Car '=' από την άκρη του Grid 2)τον αριθμό των Car που παρεμβάλλονται μεταξύ του '=' και της άκρης, δίνοντας μεγαλύτερη βαρύτητα στο α).

>>Η συνάρτηση astar_solver διαφέρει από την solver ως προς την υλοποίηση του unexplored.Εδώ, αντί για λίστα χρησιμοποιείται ένα pairing heap. Μέσα στον σωρό φυλάσσονται tuples καταστάσεων μαζί με κόστος μονοπατιού(g). Τα tuples ταξινομούνται

εντός του σωρού με βάση το συνολικό τους κόστος(heuristic +g). Όταν εξετάζεται ένα State, έστω cs, ισχύουν τα ίδια με την solve,σχετικά με το άδειο unexplored και το finalState. Αν το cs ανήκει στο explored καλούμε αναδρομικά την astar_solver έχοντας κάνει pop στον σωρό. Αν δεν ανήκει στο explored, παράγουμε όλες τα States παιδιά του cs μέσω των συναρτήσεων successorMove και makeMove και υπολογίζουμε τα συνολικά κόστη τους. **Εδώ φαίνεται η χρησιμότητα της αποθήκευσης και του κόστους g μαζί με τα States. Έχοντας αποθηκεύσει το κόστος g για το cs,το κόστος μονοπατιού για τα παιδιά του είναι απλά (g+1)**. Η costAppend δημιουργεί τα tuples (State,g), τα οποία στην συνέχεια εισάγονται στον σωρό με την συνάρτηση heapFromList. Για την εισαγωγή ενός State στο path γίνεται ένας επιπλέον έλεγχος με την notInPath. Αν το State ανήκει στο path, τότε το path δεν μεταβάλλεται, αλλιώς εισάγεται κατάλληλα το State**. Έτσι, διασφαλίζουμε πως εντός του path φτάνουμε σε ένα State με το ελάχιστο συνολικό κόστος, αφού οποτεδήποτε ξανασυναντήσουμε το ίδιο State, θα έχει σίγουρα μεγαλύτερο g. Αντίθετα, η εισαγωγή του State στον σωρό για δεύτερη φορά δεν μας ενοχλεί** αφού, πάλι λόγο κόστους, το State θα εξαχθεί με το μικρότερο δυνατό κόστος και θα μπει στο explored. Η astar_solver καλείται αναδρομικά μέχρι την εύρεση ενός finalState.

>>Η συνάρτηση **solve_astar** καλεί την astar_solver με τα ορίσματα που αυτή πρέπει αρχικά να έχει (ίδια με αυτά που πρέπει να έχει η solver,με την διαφορά πως το unexplored είναι Heap με την αρχική κατάσταση) και αντιστρέφει την λίστα επιστροφής.

>>Το μεγαλύτερο μέρος του κώδικα για τον ορισμό του τύπου **Heap** πάρθηκε από εδώ.