REPORT

LAB 1

ΧΑΣΑΝΗΣ ΕΥΑΓΓΕΛΟΣ cs05058

ΣΑΚΕΛΛΑΡΙΟΥ ΙΩΑΝΝΑ cs05125

ΜΟΥΣΕΛΙΜΗ ΑΜΑΛΙΑ-ΓΕΩΡΓΙΑ cs05074

Η κατάσταση του προβλήματος θεωρείται ως ArrayList<ArrayList<Cube>>, μέσα στο οποίο κάθε σειρά είναι ένα ArrayList<Cube> το οποίο έχει το αντίστοιχο μέγεθος(row[1] size = 4 * K, row[2] & row[3] size = K).

Main είναι η class CubeGame.java

Αρχικά θα αναλυθεί η class GameBoard.java. Σε αυτή δημιουργείται η αρχική κατάσταση των κύβων. Με την getValidInput() αποτρέπουμε να υπάρχει input που να μην είναι integer και με την isValid() ελέγχουμε αν η θέση στην οποία προσπαθεί να βάλει τον κύβο ο χρήστης είναι επιτρεπτή. Η getK() επιστρέφει το K και η getState() την κατάσταση αφού έχει οριστεί επιτυχώς από τον χρήστη. Η printState() τυπώνει την κατάσταση.

Η class Cube.java είναι αναπαράσταση του κάθε κύβου κρατώντας και επιστρέφοντας πληροφορία για την θέση και το id του κύβου.

Οι classes Node.java & NodeAStar έχουν τον ίδιο κώδικα αλλά το Override compareTo, έχοντας κάνει implement Comparable<Node>, είναι διαφορετικό, καθώς στην NodeAStar χρησιμοποιούμε και την ευρετική συνάρτηση.

Η ευρετική συνάρτηση που έχει χρησιμοποιηθεί έχει ως g(n) το κόστος μέχρι να φτάσουμε στην κατάσταση και ως h(n) πόσοι κύβοι δεν έχουν μπει ακόμα σε σωστή θέση. Για να μπει ένας κύβος σε σωστή θέση είναι προφανές ότι χρειάζεται τουλάχιστον 1 κίνηση, ενώ σε άλλες περιπτώσεις παραπάνω, οπότε η h(n) είναι πάντα μικρότερη ή ίση του πλήθους κινήσεων.

Οι κλάσσεις αυτές κρατούν και επιστρέφουν πληροφορίες για τον γονιό(κατάσταση από όπου προήλθε), το K, την κατάσταση των κύβων, το κόστος, το βάθος και τα παιδιά(επόμενες καταστάσεις).

Η isTerminal() επιστρέφει true αν όλοι οι κύβοι είναι σε σωστή θέση.

Η isValid() επιστρέφει true αν μπορεί να μετακινηθεί ο κύβος σε αυτή τη θέση.

Η isFree() επιστρέφει true αν ο κύβος μπορεί να μετακινηθεί(δεν υπάρχει κάποιος άλλος πάνω του).

Η makeMove() αλλάζει θέση στον κύβο και υπολογίζει το κόστος μεταφοράς με την κλήση της calculateCost().

Η isInPlace() επιστρέφει true αν ο κύβος βρίσκεται στη σωστή θέση.

Η isBlocking() επιστρέφει true αν ο κύβος εμποδίζει την μεταφορά κάποιου άλλου κύβου.

Η willBlock() επιστρέφει true αν η τοποθέτηση κάποιου κύβου σε αυτή την θέση θα μπλόκαρε μελλοντική μετακίνηση άλλου κύβου.

Η copyStates() κάνει deep copy του state ώστε να μπορούμε να δημιουργήσουμε καταστάσεις παιδιά.

Η removeCube() χρησιμοιείται στην getNextStates() όταν η μεταφορά του κύβου έχει ολοκληρωθεί και θέλουμε να τον αφαιρέσουμε από την παλιά του θέση.

Η getNextStates() επιστρέφει τις καταστάσεις παιδιά. Για να αποφύγουμε την δημιουργία περιττών καταστάσεων αν ένας κύβος είναι σε σωστή θέση και δεν μπλοκάρει μετακίνηση άλλου τότε δεν παίρνουμε καταστάσεις στις οποίες αυτός μετακινείται.

Αν ένας κύβος μπλοκάρει την μετακίνηση άλλου τότε τον μετακινούμε

1 φορά σε κάθε επίπεδο(στο επιπεδο 1 ακόμα και σε μικρό K δημιουργούνται πολλές καταστάσεις με ίδιο αποτέλεσμα. Για K = 2 αν υποθέσουμε ότι από την θέση 3 ως και 8 δεν υπάρχει άλλος κύβος θα είχαμε 5 καταστάσεις παιδιά με το ίδιο ακριβώς αποτέλεσμα. Μετακινώντας το 1 φορά σε κάθε επίπεδο αποφεύγουμε να γίνει αυτό)

Η UCS και η AStar υλοποιούνται με τον ίδιο τρόπο, με ένα PriorityQueue ως frontier και ένα HashSet για να αποθηκεύει τους κόμβους που

έχουμε ελέγξει ήδη. Ο τρόπος που επιλέγονται οι κόμβοι από το μέτωπο αναζήτησης όμως είναι διαφορετικός λόγω του τρόπου σύγκρισης των κόμβων Node & NodeAStar.

Έχει δημιουργηθεί και μια extra class DFS.java που αποδεικνύει την σωστή λειτουργία της getNextStates() των nodes, αφού μετά το K = 3 η UCS και η AStar δεν μπορούν να βρουν λύση λόγω αποθηκευτικού χώρου ή αργούν πολύ να βρουν. Με τη DFS βλέπουμε ότι το κλάδεμα καταστάσεων που έγινε δεν οδηγεί σε αδιέξοδο αφού βρίσκουμε αποτέλεσμα και με λίγες κινήσεις.

Η AStar όπως βλέπουμε υπερτερεί της UCS καθώς για πιο σύνθετες καταστάσεις η $2^{η}$ αδυναμεί να καταφέρει να βρει λύση λόγω χώρου ή βρίσκει με πολύ περισσότερες επεκτάσεις.

Example(χωρίς τα path) :

Initial State:

Level 3 : 3 4

Level 2 : 1 2

Level 1 : 6 5 X X X X X X

>>>>>>>>>>>>>>> A-STAR <<<<<<<<<<<<<<<<<<

Found terminal

Cost : 12.0

Depth : 12

Nodes explored : 3334

>>>>>>>>>>>>>>> UCS <<<<<<<<<<<<<<<<<<

Found terminal

Cost : 12.0

Depth : 12

Nodes explored : 7652