Τεχνητή Νοημοσύνη

1η Σειρά Ασκήσεων

Δημήτριος Αναστασόπουλος 3180010

Μιχαήλ Δικαιόπουλος 3180050

Κωνσταντίνος Καρράς 3180076

Αναπτύξαμε το πρόβλημα της διάσχισης της γέφυρας σε γλώσσα προγραμματισμού Java. Στον φάκελο src υπάρχουν 3 αρχεία (Main.java, SpaceSearcher.java, State.java).

- Main.java
  Η Main κλάση περιέχει μία και μόνο συνάρτηση, τη main. Έχουν γίνει import και τρεις βιβλιοθήκες της (ArrayList, Collections, Scanner). Ζητάμε από το χρήστη να δώσει τον αριθμό των ανθρώπων που θα διασχίσουν τη γέφυρα και σε περίπτωση που δώσει αριθμό μικρότερο ή ίσο με το μηδέν τον ενημερώνουμε και ξαναζητάμε τιμή. Ύστερα, κάνουμε ακριβώς το ίδιο και για τους χρόνους του κάθε ατόμου, αλλά και για τη μέγιστη επιτρεπτή τιμή. Μετράμε και τον χρόνο που έκανε να επιστρέψει το terminalState από τον αλγόριθμο SpaceSearcher. Εμφανίζουμε τη λύση, παίρνοντας τον πατέρα κάθε κόμβου που βρίσκεται στο μονοπάτι της λύσης αναδρομικά και εμφανίζοντας το Arraylist αντεστραμμένο, τα βήματα που έκανε ο αλγόριθμος, καθώς και τον χρόνο σε second. Αλλιώς, σε περίπτωση που το πρόβλημα δεν έχει λύση, εμφανίζουμε αντίστοιχο μήνυμα. Τέλος, ρωτάμε το χρήστη αν θέλει να συνεχίσει το πρόγραμμα και ανάλογα με την απάντησή του είτε συνεχίζουμε από την αρχή είτε σταματάμε.

- SpaceSearcher.java
  - ❖ Κατασκευαστής SpaceSearcher όπου αρχικοποιούνται ως null οι λίστες states και closedSet. Η states περιέχει όλες τις καταστάσεις που παράγονται ταξινομημένες, ενώ το closedSet περιέχει όλους του κόμβους που έχουν εξεταστεί.

- ❖ Static συνάρτηση AstarClosedSet που επιστρέφει state(το terminalState ή αν δεν υπάρχει αυτό τότε null) και η οποία παίρνει σαν όρισμα το initialState και το μέγιστο επιτρεπτό χρόνο που έχει δώσει σαν είσοδο ο χρήστης. Εκχωρεί το initialState στη λίστα των states και όσο υπάρχει state εξετάζει το κόστος του, την περίπτωση που είναι τερματική κατάσταση (οπότε και επιστρέφει) και τέλος αν δεν υπάρχει στο κλειστό σύνολο, τότε προσθέτει και τον ίδιο τον κόμβο αλλά και τα παιδιά του. Τέλος, ταξινομεί τη λίστα των states.

- State.java
  - ❖ Κατασκευαστής State που παίρνει σαν ορίσματα τη λίστα των ανθρώπων που θέλουν να διασχίσουν τη γέφυρα, τη λίστα αυτών που την έχουν ήδη διασχίσει, τη λάμπα(boolean, εάν είναι false τότε είναι στη μεριά με τους ανθρώπους που θέλουν να διασχίσουν τη γέφυρα, αλλιώς αν είναι true βρίσκεται στην άλλη μεριά, με τους ανθρώπους που έχουν ήδη διασχίσει τη γέφυρα), το extraTime που είναι το κόστος μετάβασης στον επόμενο κόμβο και τέλος τον πατέρα του κόμβου του τωρινού state.
  - ❖ Δεύτερος κατασκευαστής, ο οποίος χρησιμοποιείται για το initialState όπου το μόνο όρισμα που δίνεται είναι η λίστα με τους ανθρώπους που θέλουν να διασχίσουν τη γέφυρα ενώ όλα τα υπόλοιπα ορίσματα είναι Crossed = null, lamp = false, extraTime = 0, father = null.
  - ❖ Συνάρτηση getChildren, η οποία επιστρέφει ένα Arraylist με όλα τα παιδιά. Η συνάρτηση αυτή όμως μία άλλη συνάρτηση(την findChildren). Ανάλογα με τη τιμή της λάμπας δίνει και τα αντίστοιχα ορίσματα. Αν η λάμπα είναι false (δηλαδή στη μεριά με τα άτομα όπου θέλουν να διασχίσουν τη γέφυρα αλλά δεν την έχουν διασχίσει ακόμα) τότε δίνει σαν όρισμα το Arraylist με αυτούς που δεν έχουν περάσει ακόμα και τη τιμή της λάμπας. Ειδάλλως, δίνει σαν όρισμα την άλλη λίστα(με αυτούς που έχουν περάσει) και πάλι τη τιμή της λάμπας. Αυτό γίνεται, διότι θέλουμε να παράγουμε κάθε φορά τα παιδιά της κατάστασης της οποίας βρισκόμαστε.

- Συνάρτηση findChildren, η οποία παίρνει από την προαναφερθείσα δύο ορίσματα. Το πρώτο ουσιαστικά όρισμα μπορεί να είναι μία από τις δύο λίστες, ενώ το δεύτερο(η λάμπα) είναι αυτό που καθορίζει τις ενέργειες που θα γίνουν μέσα. Αρχικά, ορίζουμε δύο προσωρινές λίστες(temp και temp1) για τις λίστες crossed και toCross και ακόμη μία λίστα (children) για τα παιδιά που θα παραχθούν.
  - Εάν η λάμπα είναι false και το μήκος της λίστας είναι ίσο με 1 και ο πατέρας του state είναι ίσος με null (δηλαδή είναι το initialState) τότε κάνουμε deep copy χρησιμοποιώντας iterator στον temp το toCross και στον temp1 το crossed. Στον temp1(προσωρινός crossed) προσθέτουμε την πρώτη και μοναδική τιμή του δοθέντος Arraylist, την οποία ύστερα την αφαιρούμε από το temp(προσωρινός toCross). Τέλος, δημιουργούμε ένα child με ορίσματα toCross = temp, crossed = temp, lamp = true(έτσι ώστε αν υπερισχύσει αυτό το state να κοιτάξουμε από την άλλη μεριά μετά), extraTime = child_extraTime(που είναι ίσο με το χρόνο που κάνει να περάσει από την άλλη μεριά το άτομο) και father = this. Προσθέτουμε το child αυτό στο Arraylist children και επιστρέφουμε το children.
  - Εάν η λάμπα είναι πάλι false αλλά αυτή τη φορά δεν είναι το μέγεθος της λίστας toCross ίσο με 1, τότε και αφού υπάρχει η δυνατότητα περνάμε στην απέναντι μεριά δύο άτομα. Για να γίνει, όμως, αυτό θα πρέπει να εξετάσουμε όλα τα πιθανά ζευγάρια μεταξύ των ατόμων που βρίσκονται στην λίστα toCross. Αυτό το επιτυγχάνουμε με διπλό for loop το οποίο εξετάζει μόνο διαφορετικά μεταξύ τους ζευγάρια. Θέτουμε εκ νέου τα temp και temp1 Arraylist ως κενά και αντιγράφουμε όπως και πριν με χρήση iterator για deep copy το toCross και το crossed αντίστοιχα. Ομοίως, προσθέτουμε δύο στοιχεία αυτή τη φορά στο temp1 και τα αφαιρούμε από το temp. Παίρνουμε ως child_extraTime τη μέγιστη από τις δύο τιμές που αντίστοιχα προσθαφαιρέσαμε(καθώς ο χρόνος του πιο αργού μετράει πάντα για τη διάσχιση σε ένα ζευγάρι) και

ορίζουμε ένα νέο child με τα ακριβώς ίδια ορίσματα όπως και πριν και το προσθέτουμε στο Arraylist children.

- Εάν τώρα η λάμπα είναι true, πράγμα το οποίο σημαίνει ότι πρέπει να γυρίσει κάποιος πίσω για να πάρει και τους υπόλοιπους που δεν έχουν διασχίσει τη γέφυρα(γνωρίζουμε ότι σίγουρα υπάρχουν άτομα πίσω αλλιώς θα σταματούσε στον έλεγχο για το terminalState μέσα στον AstarClosedSet στον SpaceSearcher). Θέτουμε εκ νέου τα temp και temp1 Arraylist ως κενά και αντιγράφουμε όπως και πριν με χρήση iterator για deep copy το toCross και το crossed αντίστοιχα. Ομοίως, προσθέτουμε ένα στοιχεία αυτή τη φορά στο temp και το ίδιο το αφαιρούμε από το temp1. Παίρνουμε ως child_extraTime το χρόνο του εκάστοτε στοιχείου που προσθαφαιρούμε και δημιουργούμε με τον γνωστό πλέον τρόπο το child. Τέλος, προσθέτουμε στο child στο children Arraylist.

- Συνάρτηση trueCost, που παίρνει ως όρισμα τον έξτρα χρόνο που κάνει ο εκάστοτε κόμβος για να μετακινηθεί από τον πατέρα στο παιδί. Αν ο πατέρας είναι null τότε επιστρέφει μηδέν, καθώς πρόκειται για το initialState. Αλλιώς, θέτει το νέο κόστος ίσο με το κόστος του πατέρα και την μετάβαση στο παιδί, μέσω της void συνάρτησης setCost(setter) και το επιστρέφει μέσω της getCost(getter).
- Ακολουθεί η ευρετική συνάρτηση heuristic που επιστρέφει έναν ακέραιο αριθμό, ο οποίος είναι μία εκτίμηση για την περίπτωση όπου πρόκειται να περάσουν δύο άτομα που δεν έχουν διασχίσει τη γέφυρα ακόμη. Αν ο αριθμός των ατόμων που περιμένουν είναι ζυγός, τότε στο heuristic_score προσθέτουμε τους ανθρώπους που βρίσκονται στις θέσεις 2, 4, 6 κ.λπ. Αυτό, διότι οι χρόνοι των ατόμων είναι σε αύξουσα σειρά και αν υποθέσουμε ότι πάνε όλοι μαζί κατευθείαν σε ζευγάρια(εννοούμε χωρίς να απαιτείται η επιστροφή της λάμπας) τότε ο ελάχιστος χρόνος είναι το άθροισμα των ατόμων που βρίσκονται σε ζυγές θέσεις. Γι’ αυτό, λοιπόν, και στο heuristic_score την προσθέτουμε το toCross(++i), καθώς ο prefix τελεστής πρώτα αυξάνει και μετά χρησιμοποιείται, και ύστερα ακόμη προσθέτουμε άλλη μία μονάδα στο i. Επίσης, η ευρετική μας υποεκτιμά το συνολικό κόστος, καθώς δεν λαμβάνει υπόψη της, τις επιστροφές της

λάμπας. Αντίστοιχα, λοιπόν, και με το ίδιο σκεπτικό αν ο αριθμός των ατόμων που περιμένουν να περάσουν απέναντι είναι περιττός, τότε ο ελάχιστος χρόνος είναι το άθροισμα των χρόνων των ατόμων που βρίσκονται στις θέσεις 1, 3, 5 κ.λπ. Αυτό, διότι οι χρόνοι των ατόμων είναι σε αύξουσα σειρά και αν υποθέσουμε ότι πάνε όλοι μαζί κατευθείαν σε ζευγάρια(εννοούμε χωρίς να απαιτείται η επιστροφή της λάμπας) τότε ο ελάχιστος χρόνος είναι το άθροισμα των ατόμων που βρίσκονται σε μονές θέσεις, καθώς περισσεύει άλλος ένας που δεν έχει ζευγάρι. Γι’ αυτό, λοιπόν, και στο heuristic_score την προσθέτουμε το toCross(i) και ύστερα αυξάνουμε τον τελεστή i κατά 2 μονάδες.

- Συνάρτηση isTerminal, η οποία επιστρέφει true μόνο εάν το μέγεθος της λίστας toCross είναι ίσο με το μηδέν, αλλιώς επιστρέφει false.
- Συνάρτηση print**,!!**
- Επιπλέον, γίνεται override η συνάρτηση equals που συγκρίνει δύο καταστάσεις και αν αυτές είναι ίδιες, τότε εμφανίζει true. Override γίνεται επίσης και η συνάρτηση compareTo, η οποία επιστρέφει μηδέν όταν οι δύο τιμές είναι ίδιες, αρνητικό όταν το this.score < s.score και θετικό όταν this.score > s.score.
- Τέλος, χρησιμοποιούνται και κάποιοι getters και setters για μπορούμε να θέσουμε, αλλά και να πάρουμε τις τιμές των μεταβλητών από άλλες συναρτήσεις για το ίδιο αντικείμενο.