Τεχνητή Νοημοσύνη (Κύβος του Ρούμπικ)

Περιγραφή

Με αυτό το πρόγραμμα προσπαθούμε να λύσουμε έναν κύβο Ρούμπικ, χρησιμοποιώντας τον αλγόριθμο A*. Το πρόγραμμα περιέχει τρεις κλάσεις, την Cube, την CubeSolver και την Main.

Cube.java

Αρχικά, κάνουμε implement την κλάση Comparable για να μπορέσουμε να κάνουμε την συνάρτηση compareTo. Έχουμε δηλώσει τον δισδιάστατο πίνακα int με όνομα squares, ο οποίος έχει διαστάσεις 6 (οι πλευρές) x 9 (κουτάκια σε κάθε πλευρά). Επίσης δηλώσαμε τις integer μεταβλητές cost, που αντιπροσωπεύει το κόστος για να φτάσουμε στην συγκεκριμένη κατάσταση, και h_cost, που είναι το κόστος που προβλέπεται μέχρι την τελική κατάσταση. Με την μεταβλητή father τύπου Cube, η οποία αντιπροσωπεύει κάθε φορά τον γονέα κόμβο. Για κάθε μεταβλητή έχουμε getter και setter.

Με τον constructor δημιουργούμε έναν κύβο όπου δεν είναι ανακατεμένος και αρχικοποιούμε το κόστος. Στην void print() τυπώνουμε τον κύβο σε ξεδιπλωμένη μορφή.

Στις γραμμές 144 μέχρι 488 έχουμε φτιάξει τις κινήσεις που μπορεί να κάνει ένας κύβος. Οι κινήσεις που περιέχουν στο τέλος του ονόματος την λέξη «anti», είναι αριστερόστροφες, ενώ οι υπόλοιπες είναι δεξιόστροφες. Στις δεξιόστροφες κινήσεις το cost αυξάνεται κατά 1, ενώ στις αριστερόστροφες μειώνεται αρχικά κατά 2 αφού μετά καλούμε την αντίστοιχη δεξιόστροφη κίνηση και το αυξάνει κατά 3. Άρα θα αυξηθεί κατά 1.

Έπειτα η void shuffle(int moves) ανακατεύει τον κύβο με τυχαίες κινήσεις. Το όρισμα moves αντιπροσωπεύει τον αριθμό κινήσεων που θέλουμε να κάνουμε για να ανακατέψουμε τον κύβο. Αναλόγως με την moves επιλέγει τυχαίο συνδυασμό κινήσεων από τις κινήσεις που κάναμε παραπάνω χρησιμοποιώντας την κλάση Random. Τέλος, επαναφέρουμε το κόστος στην τιμή 0 με τη βοήθεια του setter.

Η συνάρτηση boolean checkSide(int x) επιστρέφει true, αν η x πλευρά του κύβου είναι λυμένη. Η συνάρτηση boolena isFinal(int k) που παίρνει ως όρισμα το πόσες πλευρές πρέπει να λύσουμε, με την χρήση ενός counter μετράει πόσες πλευρές είναι λυμένες, με τη βοήθεια της checkSide για τον έλεγχο. Εάν οι πλευρές είναι τουλάχιστον όσες έχει ζητήσει το k, επιστρέφει true.

Η συνάρτηση countRotations παίρνει ως ορίσματα 2 μεταβλητές, την x και y, όπου είναι η πλευρά και το κουτάκι που θέλουμε να ελέγξουμε αντίστοιχα. Υπολογίζει τον ελάχιστο αριθμό κινήσεων για να πάει το συγκεκριμένο κουτάκι στη σωστή πλευρά. Η συνάρτηση countSumRotations υπολογίζει το άθροισμα όλων των κινήσεων χρησιμοποιώντας την countRotations για κάθε κουτάκι. Επιστρέφει το άθροισμα δια 9, γιατί μόνο έτσι η ευρετική θα είναι αποδεκτή, με τη λογική ότι με κάθε κίνηση μετακινείς περισσότερα από ένα κουτάκια.

Κάναμε override την συνάρτηση compareTo ώστε να συγκρίνει δύο κύβους ανάλογα με το συνολικό κόστος που είναι το άθροισμα του cost και του h_cost.

Η συνάρτηση equals παίρνει ως όρισμα έναν κύβο και ελέγχει αν είναι ίδιος με αυτόν που την καλούμε.

Η συνάρτηση findCombinations επιστρέφει όλους τους συνδυασμούς πλευρών ανάλογα με το πόσες πλευρές θέλουμε να λύσουμε (k).

Η συνάρτηση countByColor παίρνει ως όρισμα τον αριθμό πλευρών που θέλουμε να λύσουμε. Εάν θέλουμε να λύσουμε όλες τις πλευρές, χρησιμοποιείται η countSumRotations. Αλλιώς, υπολογίζουμε το προβλεπόμενο κόστος για κάθε συνδυασμό χρωμάτων με τη βοήθεια της findCombinations και επιστρέφουμε το ελάχιστο.

Η συνάρτηση evaluate που έχει ως όρισμα το k, απλά καλεί την countByColor για τον συγκεκριμένο κύβο έτσι ώστε να δώσουμε τιμή στην h_cost.

Η συνάρτηση getChildren επιστρέφει μια λίστα με όλα τα παιδιά του κύβου. Κάθε παιδί δημιουργείται κάνοντας μια κίνηση στον γονέα κόμβο και σε κάθε παιδί καλείται η evaluate καθώς και η set_father.

CubeSolver.java

Με την CubeSolver φτιάχνουμε τον αλγόριθμο A* με κλειστό σύνολο. Ο A* αναζητεί το καλύτερο μονοπάτι επεκτείνοντας τον κόμβο με το μικρότερο συνολικό κόστος με βάση την ευρετική.

Η συνάρτηση closedSetContains ελέγχει αν στο κλειστό σύνολο υπάρχει κύβος ίδιος με τον κύβο c, τον οποίο παίρνει ως όρισμα.

Main.java

Στην main δημιουργούμε έναν initialCube και τον ανακατεύουμε. Ακολούθως καλούμε τον αλγόριθμο Astar_ClosedSet ώστε να φτάσει σε τελική κατάσταση terminalCube. Τέλος, τυπώνουμε το μονοπάτι προς τη λύση.

Παραδείγματα Εκτέλεσης του προγράμματος

A) K=1, shuffle = 8

Cube to solve:

```
-----------------------------
      3 5 3
      0 1 0
      3 5 3


0 1 0   5 1 5   4 3 4   2 3 2
0 0 0   5 2 5   4 4 4   2 5 2
0 1 0   5 1 5   4 3 4   2 3 2


      1 2 1
      4 3 4
```

1 2 1

Path to solved cube (3 rotations):

---

3 5 3

0 1 0

3 5 3

0 1 0 5 1 5 4 3 4 2 3 2

0 0 0 5 2 5 4 4 4 2 5 2

0 1 0 5 1 5 4 3 4 2 3 2

1 2 1

4 3 4

1 2 1

---

5 5 5

5 1 5

5 5 5

0 0 0 1 1 1 4 4 4 3 3 3

1 0 1 4 2 4 3 4 3 0 5 0

0 0 0 1 1 1 4 4 4 3 3 3

2 2 2

2 3 2

222

010

515

010

202 141 545 303

202 121 545 353

202 141 545 303

434

232

434

111

111

111

222 444 555 000

000 222 444 555

222 444 555 000

333

333

333

B) K=2, shuffle = 8

Cube to solve:

---

0 1 3

5 1 3

5 1 4

2 0 0 1 2 5 1 4 4 2 5 3

2 0 0 1 2 3 5 4 4 2 5 3

5 4 4 2 0 5 3 0 0 1 4 3

1 3 4

1 3 2

0 5 2

Path to solved cube (5 rotations):

---

0 1 3

5 1 3

5 1 4

2 0 0 1 2 5 1 4 4 2 5 3

2 0 0 1 2 3 5 4 4 2 5 3

5 4 4 2 0 5 3 0 0 1 4 3

1 3 4

1 3 2

0 5 2

013

513

514

200 125 144 253

200 123 544 253

205 300 143 544

422

335

110

013

513

514

200 125 144 253

200 123 544 253

300 143 544 205

250

231

431

113

113

114

000 225 144 255

000 223 544 255

223 443 544 200

550

331

331

112

112

112

000 223 444 155

000 223 444 155

223 444 155 000

555

333

333

112

112

112

0 0 0 2 2 3 4 4 4 1 5 5

0 0 0 2 2 3 4 4 4 1 5 5

0 0 0 2 2 3 4 4 4 1 5 5

3 3 5

3 3 5

3 3 5

Γ) K=6, shuffle = 6

Cube to solve:

---

5 1 1

5 1 4

0 1 2

2 1 1 4 4 3 4 5 3 1 0 5

5 0 2 1 2 2 4 4 3 4 5 3

5 0 2 1 0 0 2 2 5 4 5 3

4 3 3

2 3 3

0 0 0

Path to solved cube (8 rotations):

511

514

012

211 443 453 105

502 122 443 453

502 100 225 453

433

233

000

411

114

112

122 443 453 100

100 222 443 455

255 000 225 455

333

333

500

211

114

112

022 443 451 055

000 222 441 055

555 000 224 144

333

333

335

114

114

112

222 443 455 100

100 222 443 455

155 000 223 455

333

333

005

442

111

111

100 222 443 455

100 222 443 455

155 000 223 455

333

333

005

111

111

111

500 222 442 555

000 222 444 555

055 000 224 444

333

333

333

244

111

111

100 222 443 455

100 222 443 455

155 000 223 455

333

333

500

111

111

111

000 222 444 555

000 222 444 555

555 000 222 444

333

333

333

111

111

111

000 222 444 555

000 222 444 555

000 222 444 555

333

333

333
