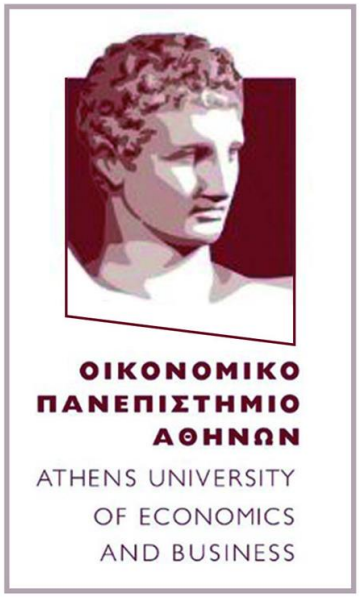


Τεχνητή Νοημοσύνη - INF153

I. Ανδρουτσόπουλος

2022-23

1η Εργασία

Κύβος του Rubik

Αναστάσιος Σπυρίδων Μπουρτσουκλής
3190138

Αντώνιος Προμπονάς
3190179

Περιεχόμενα

Τρόπος χρήσης

Αρχικά, ο χρήστης του προγράμματος δίνει ως είσοδο στο πρόγραμμα τον αριθμό των πλευρών του κύβου που θα πρέπει να έχουν ενιαίο χρώμα.

```
How many sides do you want to be completed in the cube?
```

Στην συνέχεια, ο χρήστης επιλέγει και τον τύπο του κύβου Rubik που θα κατασκευαστεί, ανάμεσα στον πλήρης τυχαιοποιημένο και δυο ενδεικτικές αρχικές καταστάσεις.

```
Choose the version of the cube you want to create:
Press 0 to create a randomized cube.
Press 1 to create the 1st test cube.
Press 2 to create the 2nd test cube.
```

Το πρόγραμμα εκτυπώνει τις κινήσεις που γίνονται προκειμένου ο κύβος να «ανακατευτεί» ή να διαμορφωθεί στην αντίστοιχη ενδεικτική κατάσταση και την αρχική κατάσταση.

```
Moves made to randomize:
```

Τέλος, το πρόγραμμα εκτυπώνει τις κινήσεις που γίνονται προκειμένου ο κύβος να «φτάσει» σε τελική κατάσταση και την τελική κατάσταση.

```
Moves made to solve:
```

Δυνατότητες

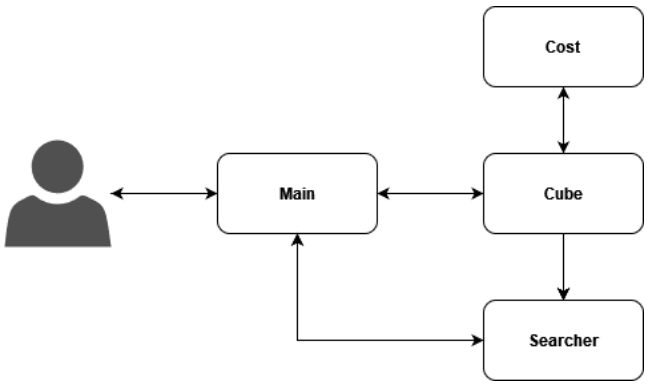
Αρχικά, το πρόγραμμα έχει την δυνατότητα να δέχεται από τον χρήστη τον αριθμό των πλευρών του κύβου που θα πρέπει να έχουν ενιαίο χρώμα, ώστε μία κατάσταση να θεωρείται τελική, και εν τέλει, να τερματίζει την αναζήτηση και να επιστρέψει την αντίστοιχη κατάσταση.

Επιπρόσθετα, το πρόγραμμα, ξεκινώντας από μία αρχική κατάσταση στον κύβο του Rubik, αναζητεί και εκτελεί, κάθε φορά, την βέλτιστη κίνηση που οδηγεί πιο αποδοτικά, δηλαδή με τις λιγότερες δυνατές συνολικές κινήσεις, σε τελική κατάσταση.

Σε κάθε κατάσταση, εφόσον δεν αποτελεί τελική κατάσταση, προκειμένου να επιλεχθεί η κατάλληλη κίνηση, χρησιμοποιείται ο αλγόριθμος A*, με κλειστό σύνολο. Αναλυτικότερα, επιλέγεται η κίνηση που οδηγεί στην κατάσταση n, με το μικρότερο άθροισμα $f(n) = h(n) + g(n)$, όπου $h(n)$ το κόστος που ενδείκνυται από την ευρετική συνάρτηση για την κατάσταση n και $g(n)$ το σύνολο των κινήσεων από την αρχική κατάσταση μέχρι την κατάσταση n, συγκριτικά με όλες τις δυνατές κινήσεις.

Τέλος, υπάρχει η δυνατότητα να μεταβληθεί το μέγεθος του κύβου, μεταβάλλοντας την τιμή της μεταβλητής cubeSize. Το μέγεθος του κύβου είναι προκαθορισμένο, βέβαια, σε 3x3, ωστόσο αν η μεταβλητή cubeSize, για παράδειγμα, έχει ως τιμή 4, το πρόγραμμα λειτουργεί με κύβο μεγέθους 4x4, δίχως προβλήματα.

Αρχιτεκτονική



Main

Η επικοινωνία του χρήστη και του προγράμματος πραγματοποιείται μέσω της **Main** κλάσης. Αναλυτικότερα, γίνεται είσοδος του είδους αρχικής κατάστασης του κύβου και στην περίπτωση που επιλέγεται η τυχαία αρχική κατάσταση, γίνεται είσοδος και του πλήθους πλευρών που είναι αναγκαίο για να θεωρείται μία κατάσταση του κύβου ως τελικ. Μέσω της main, δημιουργείται κατάλληλο αντικείμενο κλάσης Cube, χρησιμοποιώντας τον constructor της, και εκτυπώνεται η αρχική του κατάσταση. Στην συνέχεια, δημιουργείται αντικείμενο κλάσης Searcher και εκτελείται η μέθοδος aStar της, με παράμετρο την αρχική κατάσταση του κύβου. Τέλος, η τελική κατάσταση, που επιστρέφεται από την εκτελεσμένη μέθοδο, αποθηκεύεται στον κύβο και εκτυπώνεται.

Cube

Οι καταστάσεις του κύβου Rubik, όπως και όλες οι λειτουργίες τους, κατασκευάζονται και υλοποιούνται στην **Cube** κλάση. Συγκεκριμένα, περιλαμβάνει τον constructor του κύβου, μέσω του οποίου διαμορφώνεται κατάλληλα η αρχική κατάσταση, σύμφωνα με τα ζητούμενα του χρήστη, και εκτυπώνονται οι «κινήσεις» που πραγματοποιήθηκαν για να «ανακατευθεί» ο κύβος, και τον copy constructor, μέσω του οποίου κατασκευάζονται όλες οι επόμενες καταστάσεις του κύβου. Επίσης, περιλαμβάνονται όλοι οι αναγκαίοι getters και setters, για την λειτουργία του κύβου και σε public εύρος, συμπεριλαμβανόμενης και της μεθόδου getChilden. Οι μέθοδοι print και isFinal, που περιέχονται στην Cube, εκτυπώνουν την κατάσταση και ελέγχουν αν αποτελεί τελική κατάσταση, αντίστοιχα. Η σύγκριση μεταξύ δύο καταστάσεων του κύβου υλοποιείται με το override της μεθόδου compareTo, σύμφωνα με τα κριτήρια του αλγόριθμου A*. Τέλος, η κλάση Cube περιλαμβάνει 18 μεθόδους, εκ των οποίων κάθε μία υλοποιεί από μία «κίνηση» του κύβου, και τις μεθόδους της τεχνητής νοημοσύνης, που χρησιμοποιούνται εσωτερικά κάθε κατάστασης του κύβου.

Cost

Το ευρετικό κόστος μίας κατάστασης του κύβου Rubik υπολογίζεται μέσω της **Cost** κλάσης. Αναλυτικότερα, περιλαμβάνει τον constructor, μέσω του οποίου ορίζεται η πλευρά του κύβου, το χρώμα και το πλήθος των τετραγώνων με το συγκεκριμένο χρώμα στη

συγκεκριμένη πλευρά του κύβου. Επίσης, περιλαμβάνει και τα αναγκαία getters και setters.

Searcher

Ο αλγόριθμος αναζήτησης A* υλοποιείται στην **Searcher** κλάση. Ειδικότερα, περιλαμβάνει τον constructor, ο οποίος δημιουργεί το ανοιχτό και το κλειστό σύνολο, και την μέθοδο aStar, η οποία δέχεται ως είσοδο μία κατάσταση του κύβου Rubik και επιστρέφει την τελική κατάσταση, ενώ εμφανίζει και τις «κινήσεις» που πραγματοποιηθήκαν για να φτάσει σε αυτήν.

Μέθοδοι Τεχνητής Νοημοσύνης

getChilden

Η μέθοδος **getChildren** επεκτείνει μία κατάσταση.

Ειδικότερα, δημιουργείται ένα ArrayList με χωρητικότητα 18 αντικειμένων, και γεμίζεται με όλες τις δυνατές κινήσεις που μπορούν να γίνουν από την κατάσταση για την οποία έγινε κλήση της μεθόδου.

Αυτό επιτυγχάνεται με την αντιγραφή της κατάστασης σε ένα νέο αντικείμενο, χρησιμοποιώντας τον copy constructor, την εκτέλεση της αντίστοιχης κίνησης, τον υπολογισμό της ευρετικής τιμής της νέας πλέον κατάστασης και την προσθήκη της νέας κατάστασης στην λίστα. Το παραπάνω επαναλαμβάνεται 18 φορές, όσες και οι δυνατές κινήσεις σε έναν κύβο Rubik.

Τέλος, επιστρέφεται η λίστα.

! Η δομή δεδομένων που χρησιμοποιείται είναι το **ArrayList**, καθώς η προσθήκη και η προσπέλαση κάθε κατάστασης έχουν πολυπλοκότητα O(1), αφότου το μέγεθος της είναι προκαθορισμένο και αποφεύγονται τα περιττά resizes.

! Οι ονομασίες αντιστοιχούν στις εξής κινήσεις:

Πηγή: jperm.net

heuristic

Η μέθοδος **heuristic** αποτελεί την ευρετική συνάρτηση και την βάση του αλγορίθμου αναζήτησης A*.

Συγκεκριμένα, η συνάρτηση αυτή δημιουργεί ένα ArrayList αντικειμένων Cost και το συμπληρώνει ως εξής:

1. Δημιουργείται ένα for loop, το οποίο επαναλαμβάνεται 6 φορές, όσες δηλαδή είναι και οι πλευρές του κύβου. Σε κάθε επανάληψη, μηδενίζονται 6 counters, κάθε ένας εκ των

οποίων αντιστοιχεί σε ένα μοναδικό χρώμα του κύβου. Συνοπτικά, αυτό το loop δίνει τη δυνατότητα προσπέλασης κάθε πλευράς του κύβου.

2. Μέσα στο ίδιο loop, δημιουργούνται 2 εμφωλευμένα for loops, κάθε ένα εκ των οποίων επαναλαμβάνεται όσο και η διάσταση του πίνακα blocks, για τις γραμμές και τις στήλες του κύβου αντίστοιχα. Μέσω των εμφωλευμένων loops, κάθε φόρα υπολογίζεται το πλήθος των χρωμάτων που περιέχει η εκάστοτε πλευρά.
3. Μετά το πέρας των δύο εμφωλευμένων loops, δίνονται τιμές στα αντικείμενα της κλάσης Cost και προσθέτονται στο arraylist που έχει δημιουργηθεί. Όπως προαναφέρθηκε, η μεταβλητή Side προσδιορίζει την πλευρά του κύβου, η μεταβλητή Colour προσδιορίζει το χρώμα και η μεταβλητή Counter προσδιορίζει το πλήθος των εναπομεινάντων τετραγώνων που χρειάζονται, προκειμένου να γεμίσει μία πλευρά με ένα συγκεκριμένο χρώμα. Αυτό επιτυγχάνεται, κάνοντας ανάθεση στη μεταβλητή αυτή, τη πράξη '9-«colour»_counter', όπου «colour»_counter είναι μία εκ των αρχικοποιημένων μεταβλητών στην αρχή του πρώτου loop, πλέον περιέχει το πλήθος του συγκεκριμένου χρώματος στη συγκεκριμένη πλευρά.

Αξίζει να αναφερθεί ότι επιλέγεται το 9, καθώς αφαιρώντας από το 9 τον αριθμό των φορών που εμφανίζεται στη πλευρά αυτή το συγκεκριμένο χρώμα, βρίσκουμε πόσα «τετράγωνα» αυτού του χρώματος απαιτούνται ώστε να γεμίσει η πλευρά αυτή, με το χρώμα αυτό.

Πραγματοποιώντας τη παραπάνω διαδικασία, λοιπόν, δημιουργείται μία λίστα, η οποία δείχνει σε κάθε πλευρά του κύβου, πόσα «τετράγωνα» απομένουν για να γεμίσει η συγκεκριμένη πλευρά, με το συγκεκριμένο χρώμα.

Τέλος, η λίστα επιστρέφεται.

countHeuristicCost

Η μέθοδος **countHeuristicCost** εφαρμόζει την ευρετική συνάρτηση μίας κατάστασης, δηλαδή υπολογίζει το h(n).

Αρχικά, καλείται η ευρετική συνάρτηση heuristic() και αποθηκεύεται η επιστρεφόμενη λίστα.

Στην συνέχεια, δηλώνονται 3 μεταβλητές, οι οποίες αντιπροσωπεύουν τα αντικείμενα της κλάσης Cost. Με αυτόν τον τρόπο, για κάθε πλευρά αποθηκεύεται το χρώμα που απαιτεί τα λιγότερα «τετράγωνα» για να γεμίσει η συγκεκριμένη πλευρά και το πλήθος τους. Αναλυτικότερα, η μεταβλητή minSide αντιπροσωπεύει τις πλευρές του κύβου, η μεταβλητή minCost αντιπροσωπεύει το μικρότερο κόστος που υπάρχει σε μία πλευρά του κύβου και η μεταβλητή minColour δείχνει σε ποιο χρώμα ανήκει το μικρότερο κόστος της συγκεκριμένης πλευράς του κύβου

Και τις 3 μεταβλητές αρχικοποιούνται έτσι ώστε να εξασφαλιστεί ότι στη πρώτη σύγκριση το κόστος του 1ου στοιχείου της λίστας "Cost" θα είναι μικρότερο από την τιμή που θα περιέχει η μεταβλητή minCost. Οι τιμές των μεταβλητών θα ενημερώνονται κατάλληλα

κάθε φορά που θα βρίσκεται χρώμα σε πλευρά με κόστος μικρότερο της τιμής που θα περιέχει η μεταβλητή minCost.

Εκτελείται ένα διπλό εμφωλευμένο loop, στο οποίο η 1η επανάληψη αντιπροσωπεύει τις πλευρές του κύβου, ενώ η 2η επανάληψη αντιπροσωπεύει τα αντικείμενα της λίστας Cost. Σε κάθε επανάληψη του 2ου loop , συγκρίνεται το κόστος κάθε αντικειμένου με το minCost και έτσι βρίσκεται τη πλευρά και το χρώμα το οποίο έχει το μικρότερο κόστος προκειμένου να ολοκληρωθεί μία πλευρά του κύβου.

Τέλος, το μικρότερο κόστος αποθηκεύεται στο συνολικό ευρετικό κόστος.

aStar

Η μέθοδος **aStar** υλοποιεί τον αλγόριθμος αναζήτησης A*.

Αναλυτικότερα, προσθέτει την αρχική κατάσταση, η οποία δόθηκε ως παράμετρος, στο ανοικτό σύνολο και, μέχρις ότου το ανοικτό σύνολο να είναι κενό, ταξινομεί το ανοικτό σύνολο με βάση τις ευρετικές τιμές κάθε κατάστασης και ελέγχει αν η 1η κατάσταση της αποτελεί τελική κατάσταση. Αν είναι, τότε εμφανίζονται οι «κινήσεις» που πραγματοποιήθηκαν και επιστρέφεται η τελική κατάσταση, διαφορετικά προστίθεται στο κλειστό σύνολο και επεκτείνεται, χρησιμοποιώντας την μέθοδο getChildren.

Κάθε παιδί ελέγχεται αν βρίσκεται στο κλειστό σύνολο και αντίστοιχα, παραλείπεται ή ελέγχεται αν αποτελεί τελική κατάσταση. Αν είναι, τότε εμφανίζονται οι «κινήσεις» που πραγματοποιήθηκαν και επιστρέφεται, διαφορετικά προστίθεται στο ανοικτό σύνολο.

! Η δομή δεδομένων που χρησιμοποιείται για το ανοικτό σύνολο είναι το **PriorityQueue**, καθώς η ταξινόμηση του με βάση την ευρετική και η εξαγωγή της κατάστασης με την μικρότερη τιμή έχει πολυπλοκότητα O(logn), βελτιώνοντας αισθητά την απόδοση, παρότι η προσθήκη μίας κατάστασης έχει πολυπλοκότητα O(logn).

! Η δομή δεδομένων που χρησιμοποιείται για το κλειστό σύνολο είναι το **HashSet**, καθώς η προσθήκη και ο εντοπισμός μίας κατάστασης έχουν πολυπλοκότητα O(1).

Παραδείγματα Χρήσης

Κατασκευή κύβου Rubik με τυχαία αρχική κατάσταση και K=6.

```
Moves made to randomize: B S M R
Side 1  | Side 2  | Side 3  | Side 4  | Side 5  | Side 6
--------|---------|---------|---------|---------|--------
|W|G|Y| |G|G|G| |O|B|R| |O|Y|B| |G|R|W| |Y|W|R|
|W|B|G| |O|O|O| |B|G|R| |O|Y|B| |B|R|W| |G|W|R|
|W|O|B| |Y|Y|Y| |G|Y|R| |O|Y|B| |O|R|W| |B|W|R|
--------|---------|---------|---------|---------|--------
```

```
Moves made to solve:  R M S B
```

Side 1	Side 2	Side 3	Side 4	Side 5	Side 6
|W|W|W|	|G|G|G|	|R|R|R|	|B|B|B|	|O|O|O|	|Y|Y|Y|
|W|W|W|	|G|G|G|	|R|R|R|	|B|B|B|	|O|O|O|	|Y|Y|Y|
|W|W|W|	|G|G|G|	|R|R|R|	|B|B|B|	|O|O|O|	|Y|Y|Y|

Κατασκευή κύβου Rubik με αρχική κατάσταση ελέγχου για K=1.

```
Moves made to randomize: M U S' D
```

Side 1	Side 2	Side 3	Side 4	Side 5	Side 6
|G|G|G|	|R|Y|R|	|B|B|B|	|W|R|W|	|O|O|O|	|Y|B|Y|
|W|O|W|	|G|W|G|	|R|Y|R|	|B|R|B|	|Y|G|G|	|W|B|W|
|B|R|B|	|W|O|W|	|G|Y|G|	|R|Y|R|	|O|O|O|	|Y|O|Y|

```
Moves made to solve:  U' D' S M
```

Side 1	Side 2	Side 3	Side 4	Side 5	Side 6
|W|W|W|	|G|O|G|	|R|G|R|	|B|O|B|	|O|R|O|	|Y|Y|Y|
|W|W|W|	|G|G|G|	|R|R|R|	|B|B|B|	|R|O|B|	|Y|Y|G|
|W|W|W|	|G|O|G|	|R|Y|R|	|B|B|B|	|O|O|O|	|Y|Y|Y|

Κατασκευή κύβου Rubik με αρχική κατάσταση ελέγχου για K=2.

```
Moves made to randomize: U' E S'
```

Side 1	Side 2	Side 3	Side 4	Side 5	Side 6
|B|B|B|	|W|Y|W|	|G|G|G|	|R|O|R|	|O|O|O|	|Y|Y|Y|
|B|B|B|	|W|Y|W|	|G|G|G|	|R|O|R|	|W|W|G|	|R|R|B|
|W|W|W|	|G|Y|G|	|R|R|R|	|B|O|B|	|O|O|O|	|Y|Y|Y|

```
Moves made to solve:  S
```

Side 1	Side 2	Side 3	Side 4	Side 5	Side 6
|B|B|B|	|W|W|W|	|G|G|G|	|R|R|R|	|O|O|O|	|Y|Y|Y|
|B|B|B|	|W|W|W|	|G|G|G|	|R|R|R|	|O|O|O|	|Y|Y|Y|
|W|W|W|	|G|G|G|	|R|R|R|	|B|B|B|	|O|O|O|	|Y|Y|Y|

Κατασκευή κύβου Rubik με αρχική κατάσταση ελέγχου για K=4.

```
Moves made to randomize: F' D S F
```

Side 1	Side 2	Side 3	Side 4	Side 5	Side 6
|B|W|W|	|G|O|G|	|R|R|R|	|B|B|B|	|O|O|O|	|W|Y|Y|
|B|W|W|	|G|O|G|	|R|R|R|	|B|Y|Y|	|R|B|B|	|W|G|G|
|O|W|W|	|G|O|W|	|Y|G|G|	|R|Y|Y|	|R|O|O|	|B|Y|Y|

```
Moves made to solve:  B L F
```

Side 1	Side 2	Side 3	Side 4	Side 5	Side 6
|W|W|W|	|O|O|O|	|R|R|R|	|B|Y|Y|	|B|B|B|	|G|G|G|
|W|W|W|	|O|O|O|	|R|R|R|	|B|Y|Y|	|B|B|B|	|G|G|G|
|W|W|W|	|O|O|O|	|R|R|R|	|B|Y|Y|	|Y|Y|Y|	|G|G|G|

Κατασκευή κύβου Rubik με αρχική κατάσταση ελέγχου για K=6.

```
Moves made to randomize: U' E S' F' D F
```

Side 1	Side 2	Side 3	Side 4	Side 5	Side 6
|B|B|B|	|W|Y|W|	|G|G|G|	|R|O|B|	|O|O|O|	|B|Y|Y|
|O|B|B|	|W|Y|W|	|G|G|G|	|R|O|B|	|W|W|G|	|R|R|Y|
|O|W|W|	|G|B|W|	|Y|Y|G|	|R|R|Y|	|R|O|O|	|R|R|Y|

```
Moves made to solve:  F D' B R
```

Side 1	Side 2	Side 3	Side 4	Side 5	Side 6
|B|B|B|	|Y|Y|Y|	|G|G|G|	|O|O|O|	|W|W|W|	|R|R|R|
|B|B|B|	|Y|Y|Y|	|G|G|G|	|O|O|O|	|W|W|W|	|R|R|R|
|B|B|B|	|Y|Y|Y|	|G|G|G|	|O|O|O|	|W|W|W|	|R|R|R|