ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

Ο κόσμος των κύβων

(Blocks World)

Φρέγκος Περικλής (dai17033)

https://github.com/leajian

fregkos@gmail.com

Διδάσκων: Γιάννης Ρεφανίδης

1η Εργασία

12/5/2019

Περιεχόμενα

Εισαγωγή

Ο κόσμος των κύβων είναι πολύ απλό πρόβλημα διάταξης, όπου δίνεται μια αρχική κατάσταση και μια τελική και με διάφορους αλγορίθμους (bfs, dfp, best, a-star), καλούμαστε να βρούμε τα βήματα που απαιτούνται για να φτάσουμε από την αρχική, στην τελική κατάσταση. Η λύση μου, υλοποιήθηκε σε Python 3, μια γλώσσα πολύ απλή και εύχρηστη, που επιτρέπει γρήγορη ανάπτυξη μιας ιδέας, χωρίς να δίνει μεγάλη σημασία σε λεπτομέρειες. Επίσης είναι πολύ όμορφη και φορητή. Για τους λόγους αυτούς, επιλέχθηκε ως γλώσσα για την εργασία αυτή.

Είσοδος προβλήματος

Τα αρχεία εισόδου είναι τα ίδια που χρησιμοποιήθηκαν στο διαγωνισμό AIPS 2000 Planning και βρίσκονται όλα εδώ.

Μορφή αρχείων pddl

Η ολοκληρωμένη περιγραφή αυτής της μορφής αρχείων βρίσκεται εδώ.

```
(define (problem BLOCKS-4-1)
(:domain BLOCKS)
(:objects A C D B )
(:INIT (CLEAR B) (ONTABLE D) (ON B C) (ON C A) (ON A D) (HANDEMPTY))
(:goal (AND (ON D C) (ON C A) (ON A B)))
)
```

Παρατηρούμε ότι περιέχει πληροφορίες που δε θα χρειαστούμε, αφού εμείς θέλουμε μόνο τα ονόματά τους *(:objects ...)* και την αρχική *(:INIT ...)* και τελική *(:goal ...)* κατάσταση των κύβων. Προφανώς, τα αρχεία *pddl* δεν προορίζονταν για τον σκοπό αυτής της εργασίας, έτσι, θα αγνοήσουμε τα περιττά στοιχεία και θα εστιάσουμε μόνο στις πληροφορίες που μας είναι απαραίτητες.

Προσπέλαση (υλοποίηση parser)

Η προσπέλαση επιτυγχάνεται με τη βοήθεια της βιβλιοθήκης **re** της Python *3* που έχει χρήσιμα εργαλεία για *regular expressions*. Ο τρόπος που υλοποιήθηκε, ακολουθεί την εξής πορεία :

1. Διόρθωσε όλες τις ασυνέπειες χαρακτήρων κενού και νέας γραμμής που παρατηρήθηκαν μεταξύ αρκετών αρχείων εισόδου. Δηλαδή, αντικατέστησε πολλαπλές εμφανίσεις τους με μία αντίστοιχα.
2. Χώρισε το αρχείο σε κομμάτια που καθορίζονται από τις λέξεις κλειδιά που μας ενδιαφέρουν (objects, INIT, goal), χωρίς να μας ενδιαφέρει η διάκριση πεζών-κεφαλαίων, και τοποθέτησέ τα σε μία λίστα.

3. Στο κομμάτι *objects*, βρες όλα τα ονόματα των κύβων, ακολουθώντας ένα συγκεκριμένο μοτίβο και τοποθέτησέ τα σε μία λίστα.
4. Στο κομμάτι *INIT*, βρες όλες τις αρχικές καταστάσεις των κύβων, ακολουθώντας ένα συγκεκριμένο μοτίβο και τοποθέτησέ τα σε μία λίστα.
5. Στο κομμάτι *goal*, βρες όλες τις τελικές καταστάσεις των κύβων, ακολουθώντας ένα συγκεκριμένο μοτίβο και τοποθέτησέ τα σε μία λίστα.

Στα σχόλια του κώδικα, στο αρχείο *bwIO.py*, περιγράφεται λεπτομερώς η λογική πορεία του κάθε μοτίβου.

Έξοδος προβλήματος

Η έξοδος του προβλήματος είναι πληροφορίες σχετικές με το πρόβλημα στο τερματικό και η αναλυτική περιγραφή των βημάτων λύσης του προβλήματος σε ένα απλό αρχείο κειμένου.

Έξοδος στο τερματικό

```
| Problem name:          probBLOCKS-4-1.pddl
| Algorithm used:        best
| Number of cubes:       4
| Cubes:                 A C D B
| Solved in:             0.0009935139987646835
| Algorithm iterations:  14
| Moves:                 5
| Solution:              Found!
```

Μορφή αρχείων λύσης

```
1. Move(B, C, table)
2. Move(C, A, table)
3. Move(A, D, B)
4. Move(C, table, A)
5. Move(D, table, C)
```

Περιγράφεται αναλυτικά η σειρά των βημάτων καθώς και η μετακίνηση:

(κύβος, βρίσκεται πάνω, να πάει πάνω)

Μοντελοποίηση προβλήματος

Τρόπος προσέγγισης

Για να μπορέσουν να αναζητήσουν οι αλγόριθμοι τη σωστή κατάσταση κύβων, θα πρέπει με κάποιο τρόπο να αναπαρασταθεί η κάθε κατάσταση. Οι πληροφορίες που χρειάζονται σε μια κατάσταση είναι να γνωρίζουμε για κάθε κύβο πού βρίσκεται και αν έχει πάνω του άλλο κύβο (δηλαδή αν είναι καθαρός από πάνω ή όχι). Ωστόσο, χρειάζεται να ξέρουμε τον γονέα της κάθε κατάστασης καθώς και την κίνηση που έγινε για να μεταβεί από την κατάσταση του γονέα στην δική του. Όλα αυτά μας οδηγούν στην υλοποίηση ενός αντικειμενοστραφούς μοντέλου.

Περιγραφή υλοποίησης αντικειμένου καταστάσεων κύβων

Κατασκευαστής

parent	Το αντικείμενο κατάστασης του γονέα του. Αν δεν υπάρχει, σημαίνει ότι είναι ρίζα του προβλήματος.
moveToForm	Μια δομή δεδομένων λίστας (list) που είναι η κίνηση που χρειάστηκε για να μεταβεί από την κατάσταση του γονέα στη δική του. Αν δεν υπάρχει, σημαίνει ότι είναι ρίζα του προβλήματος.
description	Μια δομή δεδομένων λεξικό (dictionary) που περιγράφει την κατάσταση των κύβων. Το κλειδί αναφέρεται στον κύβο και τα περιεχόμενά της κάθε καταχώρησης είναι μια λίστα που στην πρώτη θέση έχει το αλφαριθμητικό όνομα του κύβου που βρίσκεται πάνω και στη δεύτερη μια μπουλιανή μεταβλητή που περιγράφει αν έχει από πάνω του άλλον ή όχι. Η περιγραφή της κατάστασης, αν δε δοθεί, αρχικοποιείται αυτόματα στον κατασκευαστή του αντικειμένου, χρησιμοποιώντας την κίνηση που χρειάστηκε.

Μέθοδοι αντικειμένου

generateStateChildren	Υπολογίζει όλες τις πιθανές κινήσεις που προκύπτουν από την κατάσταση του αντικειμένου. Επιστρέφει μια λίστα με αντικείμενα όλων των πιθανών καταστάσεων από την τωρινή.
move	Μετακινεί έναν κύβο από μία θέση, σε μία άλλη και επιστρέφει την κίνηση που έγινε σε μορφή λίστας [κύβος, παλιά θέση, καινούρια θέση]. Δίνεται η δυνατότητα να επιστραφεί μόνο η κίνηση, χωρίς πραγματικά να γίνει. Αυτό είναι χρήσιμο σε περίπτωση που θέλουμε να παράγουμε όλα τα πιθανά παιδιά μέσω όλων των πιθανών κινήσεων, αφού η κατάσταση αρχικοποιείται αυτόματα στον κατασκευαστή του αντικειμένου, χρησιμοποιώντας την κίνηση που χρειάστηκε.
tracePath	Επιστρέφει μια λίστα με όλες τις λίστες των κινήσεων που χρειάστηκε η κατάσταση για να φτάσει στην μορφή που βρίσκεται, ψάχνοντας τις κινήσεις του γονέα του γονέα κ.ο.κ.

Υπερκαλυμμένοι μέθοδοι

eq	Όταν γίνεται σύγκριση ενός αντικειμένου κατάστασης με ένα άλλο, συγκρίνουμε μόνο τις περιγραφές καταστάσεων σε μορφή δεδομένων dictionary μεταξύ τους.
repr	Η αναπαράσταση της κατάστασης είναι η ίδια η περιγραφή της, δηλαδή, σε μορφή δεδομένων dictionary.
hash	Στους αλγόριθμους αναζήτησης θέλουμε να ελέγχουμε αν έχουμε δει ή όχι μια κατάσταση πολύ γρήγορα και το hashing του αντικειμένου βοηθάει. Για το λόγο αυτό, το hash του προκύπτει από τη μέθοδο hash() της Python που χρησιμοποιεί μία αλφαριθμητική μεταβλητή που προκύπτει από το όνομα του κύβου, τη θέση του και αν είναι καθαρός από πάνω του ή όχι, για όλους τους κύβους. Π.χ. *ABTBtableFCtableT*, για την κατάσταση όπου ο Α είναι πάνω στον Β και καθαρός *T(rue)*, ο Β πάνω στο τραπέζι και όχι καθαρός *F(alse)* και ο C πάνω στο τραπέζι και καθαρός *T(rue)*.

Υπολογιστική μελέτη

Δομές δεδομένων για κάθε αλγόριθμο

Το αντικείμενο των καταστάσεων είναι *hashable* που σημαίνει ότι μια αναζήτηση σε ένα σύνολο (**set**) μπορεί να γίνει πολύ γρήγορα με πολυπλοκότητα **O(1)** στη μέση περίπτωση. Μια ουρά διπλού τέλους **deque** (double-ended queue) είναι πολύ πιο γρήγορη σε εξαγωγές/εισαγωγές στοιχείων στα άκρα, ενώ μια **λίστα** είναι πολύ αργή στις εισαγωγές στην αρχή, συγκεκριμένα **O(n)** πάντα, αφού πρέπει να ενημερώσει τις σχετικές θέσεις όλων των στοιχείων.

Πληροφορημένης αναζήτησης

Breadth First Search	• Για τους κόμβους που έχουν επισκεφθεί, χρησιμοποιήθηκε set. • Για το μέτωπο αναζήτησης deque ως ουρά.
Depdth First Search	• Για τους κόμβους που έχουν επισκεφθεί, χρησιμοποιήθηκε set. • Για το μέτωπο αναζήτησης deque ως στοίβα. Ο λόγος που δε χρησιμοποιήθηκε list ως στοίβα είναι για λόγους ομοιότητας με τον BFS και αναγνωσιμότητας.

Απληροφόρητης αναζήτησης

Best First Search A-Star Search	• Για τους κόμβους που έχουν επισκεφθεί, χρησιμοποιήθηκε set. • Για το μέτωπο αναζήτησης list, αφού οι το αντικείμενο που εξέρχεται από το μέτωπο, καθορίζεται από ευρετικές μεθόδους που ενδέχεται να επιλέξουν στοιχεία από οποιοδήποτε σημείο.

Χρονική απόδοση αλγορίθμων

Να σημειωθεί ότι οι δοκιμές έγιναν στον προσωπικό μου υπολογιστή με CPU Intel i5 Haswell, οπότε και είναι υποκειμενικές.

Παρακάτω παρατίθενται οι χρόνοι εύρεσης (με το πολύ 5 δεκαδικά ψηφία) κάθε προβλήματος από κάθε αλγόριθμο, μαζί με τον αριθμό βημάτων των λύσεων με μέγιστο όριο εκτέλεσης 1 λεπτό.

	Breadth	**Depth**	**Best**	**A***
4-0	0.00395, 3	0.00183, 13	0.00085, 3	0.00102, 3
4-1	0.00348, 5	0.00279, 18	0.00084, 5	0.00080, 5
4-2	0.00303, 3	0.00266, 3	0.00036, 3	0.00044, 3
5-0	0.03709, 6	0.03325, 170	0.00306, 5	0.00555, 6
5-1	0.03303, 5	0.03927, 91	0.00627, 10	0.00475, 5
5-2	0.03869, 8	0.01393, 86	0.00382, 16	0.01715, 8
6-0	0.30849, 6	0.22071, 1177	0.00174, 6	0.00239, 6
6-1	0.41516, 5	0.42859, 1491	0.00611, 7	0.00698, 5
6-2	0.47021, 10	0.13351, 692	0.00477, 15	0.43548, 10
7-0	4.9334, 10	3.12618, 13185	0.12622, 24	0.10268, 10
7-1	5.68184, 11	5.46296, 18661	0.03185, 18	25.05289, 11
7-2	5.75031, 10	6.10961, 3262	0.03294, 19	17.94924, 11
8-0	-	-	0.59846, 32	2.31351, 9
8-1	-	60.52310, 216055	0.29925, 35	34.59941, 10
8-2	-	-	0.17347, 22	0.34275, 8
9-0	-	-	0.83511, 62	-
9-1	-	-	0.37008, 39	11.02363, 14
9-2	-	-	1.05395, 50	38.54760, 13
10-0	-	-	0.04948, 28	-
10-1	-	-	2.39613, 51	-
10-2	-	-	0.45544, 42	-
11-0	-	-	2.90378, 49	-
11-1	-	-	1.49171, 58	-
11-2	-	-	2.06283, 51	-
12-0	-	-	43.39735, 62	-
12-1	-	-	16.47943, 82	-

Παρατηρήσεις

Παρατηρούμε πως ο γρηγορότερος αλγόριθμος είναι ο **Best First Search**, αφού συνεχίζει να βρίσκει λύσεις όσο το μέγεθος του προβλήματος μεγαλώνει, ενώ οι υπόλοιποι έχουν ήδη ξεπεράσει το χρονικό όριο για να βρουν λύση.

Ο **Breadth First Search** βρίσκει την βέλτιστη λύση, όμως δεν είναι αποδοτικός σε μεγάλα προβλήματα.

Ο **Depth First Search** δεν βρίσκει απαραίτητα την βέλτιστη λύση και μάλιστα απέχει πολύ. Επίσης όσο μεγαλώνουν τα προβλήματα, αδυνατεί να βρει λύση, σχετικά νωρίς.

Ο **A*** έχει ανάμικτους χρόνους και δεν μπορούμε να πούμε ότι εξαρτάται απαραίτητα από το μέγεθος του προβλήματος. Σε κάποιες περιπτώσεις είναι πιο γρήγορος από τους υπόλοιπους.