Artificial Intelligence Project1

Σκορδούλης Κωνσταντίνος 1115 2016 00155

Σημείωση: Στις ερωτήσεις 1,2 όπου υλοποιούμε DFS και BFS αντίστοιχα ο autograder διαφωνεί με τον αλγόριθμο των διαλέξεων (υπάρχουν κάποιες μικροδιαφορές ➔ αναλυτικά στα σχόλια του κώδικα). Ωστόσο δοκίμασα τις «εντολές» που αναφέρει (για να ελέγξουμε ότι βγαίνουν τα σωστά αποτελέσματα) ➔ και τα 2 μου βγάζουν σωστά αποτελέσματα (οι αλγόριθμοι των διαλέξεων είναι πιο αποδοτικοί ➔ λιγότερα expanded nodes). Για αυτό θα χρειαστεί κατάλληλο commenting/uncommenting κομματιών του κώδικα (δείχνω ξεκάθαρα ποια είναι).

Σημείωση2: Στα προβλήματα search.py, θεωρούσα σαν **state** τις **συντεταγμένες** του pacman (x,y).

Question1:

Χρησιμοποιήσα LIFO queue ➔ **Stack** και υλοποίησα τον **Graph Search** αλγόριθμο.

Η κύρια διαφορά μεταξύ των 2 είναι ότι, ο autograder δεν ελέγχει αν βρίσκεται ο successor ήδη στο fringe .

Question2:

Χρησιμοποιήσα FIFO queue ➔ **Queue** και υλοποίησα τον προτεινόμενο αλγόριθμο (Διαφάνειες lecture 3)

Η κύρια διαφορά μεταξύ των 2 είναι , πότε ελέγχουν αν η κατάσταση είναι **Goal state**. Ο autograder κοιτάζει όταν κάνουμε POP() από το queue, ενώ ο αλγόριθμος κατά την «δημιουργία» του node.

Question 3, 4:

Χρησιμοποίησα και για τα 2 ερωτήματα ➔ **PriorityQueue** και υπολόγιζα **h(x)** και **g(x)+h(x)** αντίστοιχα , για **priority**. Εδώ δεν είχαμε διαφωνίες με τον autograder.

Question 5:

Στο **CornersProblem**, σκοπός του agent είναι να περάσει και από τις 4 γωνίες του προβλήματος. Άρα κάπως θα πρέπει να διατηρούμε την πληροφορία ➔ ποιες γωνίες έχει επισκεφτεί. Σκέφτηκα λοιπόν να χρησιμοποιήσω την **corner_state** , η οποία είναι μια **λίστα** από **flags**(0 ή 1) ➔ **[0,0,0,0]** , την οποία θα κάνω modify κατά την διάρκεια του προγράμματος (1 αν επισκεφτήκαμε την γωνία).

State: Η κατάσταση του προβλήματος ορίζεται από δύο πράγματα:

α) **(x,y)** συντεταγμένες του agent

β) **corner_state ➔ [_,_,_,_]** που δείχνει ποιες κορυφές έχουμε επισκεφτεί.

Initial State: Η αρχική κατάσταση είναι, η αρχική θέση του agent και η λίστα [0,0,0,0]

Goal State: Μας ενδιαφέρει η λίστα να αποτελείται μόνο από 1 ➔ ((xi,yi) , **[1,1,1,1]**)

Question6:

Το heuristic μου έχει ως κύριο component ➔ **Manhattan_Distance()**, δηλαδή των υπολογισμό Manhattan αποστάσεων του agent με όλα τα corners του Grid.

Συγκεκριμένα, σε κάθε βήμα/new_state:

1. Υπολογίζω τις Manhattan αποστάσεις του **agent** με τα **unvisited corners**.
2. Από αυτές επιλέγω την **maximum**, η οποία αποτελεί και το heuristic value.

Η ιδέα προέκυψε , επιλύοντας το **relaxed problem** (χαλαρωμένη έκδοση του προβλήματος) ➔ Αφαιρούμε όλα τα Walls.

Σκέφτηκα , πως για να φτάσουμε στο goal state πρέπει να περάσουμε από όλα τα corners. Επειδή όλες οι κινήσεις μας θα έχουν cost = 1 , καθοριστικό παράγοντα θα έχει το heuristic. Σε κάθε new state, υπολογίζω τον Manhattan distance όλων των **unvisited corners,** και αυτό που έχει τη μεγαλύτερη απόσταση θα το επισκεφτεί τελευταίο. Εννοείται πως το corner που είναι το πιο «μακρινό» δεν είναι σταθερό, και αλλάζει κατά τη διάρκεια του αλγορίθμου.

Για την **συνέπεια** της ευρετικής συνάρτησης, στην εκφώνηση έλεγε πως για ένα συγκεκριμένο πρόβλημα, είτε χρησιμοποιήσω A* είτε UFS ➔ πρέπει να έχω το ίδιο πλήθος expanded nodes ➔ το οποίο ευτυχώς έχω (106) !

Από τις διαφάνειες, «Αν μια συνάρτηση είναι συνεπής τότε είναι και παραδεκτή» ➔ All ok.

Question7:

Παρόμοιο σκεπτικό, μόνο που τώρα χρησιμοποιούμε **mazeDistance()** (για πιο ακριβή αποτελέσματα από ότι με Manhattan Distance).

Επιπλέον, θεωρώ πως αν ένας λαβύρινθος έχει παραπάνω από 15 **Walls ,** τότε θεωρείται **πολύπλοκος**, με αποτέλεσμα να μην χρειάζεται να υπολογίζω πολλές mazeDistance() αποστάσεις (καθορίζεται από το **counter>3**).