Εργασία 1 στο Μάθημα Τεχνητή Νοημοσύνη

Αθανασία Τουρνάκη
ΑΜ: 1115201600172

Πρόβλημα 6 (Pacman Project 1):

Σε αυτήν την εργασία υλοποιήθηκαν όλα τα ερωτήματα, δηλαδή πραγματοποιήθηκαν όλες οι απαραίτητες αλλαγές στα αρχεία search,py, searchAgents.py. Οι υλοιποιήσεις έγιναν με βάση τις διαλέξεις και τη θεωρία που είχαμε διδαχτεί στο μάθημα και γι'αυτό τον λόγο υπάρχουν κάποιες διαφορές με αυτές που ζητούσε το Project, ονομαστικά στα ερωτήματα 1, 2 και 5, οι οποίες γίνονται φανερές στα αποτέσματα του Autograder και τις οποίες θα αναλύσω παρακάτω. Αναλυτικά, για κάθε ερώτημα.

Question 1: Finding a Fixed Food Dot Using Depth First Search

Υλοποιήθηκε η συνάρτηση depthFirstSearch στο search.py με βάση την Graph-Search των διαφανειών που μας δόθηκαν.

Χρησιμοποιήθηκε η έτοιμη δομή Στοίβας απο το util.py(util.Stack()) σαν σύνορο, δημιουργήθηκε μια λίστα explored η οποία κρατάει τους κόμβους που έχουν ήδη επισκεφθεί, ενώ για διευκόλυνση στον εντοπισμό του μονοπατιού προς τον κόμβο στόχο, έγινε χρήση λεξικού parents το οποίο κρατάει για κάθε state που συναντάει ένα tuple που περιέχει τον πατέρα του και την οδηγία που οδηγεί απο τον πατέρα στο παιδί, όπως δίνεται απο την συνάρτηση problem.getSuccessors(-state-). Για το Start State το tuple αυτό θα είναι (0, 0).

Για να ανοικοδομήσει τη διαδρομή, ο αλγόριθμος μου χρησιμοποιεί μια προσωρινή λίστα στην οποία αποθηκεύει την κατεύθυνση που οδηγεί από τον πατέρα στο παιδί(με τη χρήση του λεξικού) ξεκινώντας απο το Goal State και καταλήγοντας στο Start State.

Σε αυτήν την συνάρτηση, ο Autograder επιστρέφει αποτέλεσμα 0/3. Αυτό συμβαίνει διότι η υλοποίηση της συνάρτησης DFS με βάση την Graph-Search ορίζει την προσθήκη στο σύνορο όλων των διαδόχων του state και στην επόμενη επανάληψη απομάκρυνση εκείνου που προστέθηκε πιο πρόσφατα, ενώ ο Autograder για να λειτουργήσει σωστά περιμένει να αναπτύσσεται ο κάθε διάδοχος αναδρομικά μέχρι να βρεθεί το Goal-State.

Question 2: Breadth First Search

Υλοποιήθηκε η συνάρτηση breadthFirstSearch στο search.py με βάση την δοσμένη BFS στις διαφάνειες του μαθήματος.

Χρησιμοποιήθηκε η έτοιμη δομή Ουράς από το util.py(util.Queue()) σαν σύνορο, δημιουργήθηκε μια λιστα explored για να κρατάει τους κόμβους που έχουμε επισκεφτεί και έγινε και πάλι χρήση λεξικού parents για την διευκόλυνση στην ανοικοδόμηση του μονοπατιού. Για κάθε state που συναντάμε αποθηκεύται στο λεξικό ένα tuple που περιέχει τον πατέρα του και την οδηγία από τον πατέρα στο παιδί, όπως αυτή δίνεται απο το problem.getSuccesors(-state-). Για το Start State το tuple αυτό θα είναι (0, 0).

Η ανοικοδόμηση του μονοπατιού γίνεται με τον ίδιο τρόπο που περιγράφηκε παραπάνω για την DFS.

Να σημειωθεί ότι, όπως ορίζουν οι διαφάνειες, ο έλεγχος για το εάν ένα state είναι Goal State γίνεται πρωτού αυτό εισαχθεί στο σύνορο, επομένως ο Autograder επιστρέφει αποτέλεσμα 0/3, αφού περιμένει μία παραπάνω επέκταση κόμβου.

Question 3: Varying the Cost Function

Υλοποιήθηκε η συνάρτηση uniformCostSearch στο search.py με βάση την δοσμένη UCS στις διαφάνειες του μαθήματος.

Χρησιμοποιήθηκε η έτοιμη δομή Ουράς Προτεραιότητας από το util.py(util.priorityQueue()) σαν σύνορο, δημιουργήθηκε μια λιστα explored για να κρατάει τους κόμβους που έχουμε επισκεφτεί και έγινε χρήση λεξικού info που κρατάει για κάθε state που συναντάμε ένα tuple που περιέχει τον πατέρα του, την οδηγία από τον πατέρα στο παιδί, όπως αυτή δίνεται απο το problem.getSuccesors(-state-), καθώς επίσης και το κόστος της διαδρομής έως και αυτό το state και μια boolean μεταβλητή που δίνει την πληφορία για το εάν ένα state βρίσκεται ή όχι μέσα στο σύνορο. Για το Start State, μόλις αυτό μπει στο σύνορο το tuple αυτό θα είναι (0, 0, 0, True).

Η ανοικοδόμηση του μονοπατιού γίνεται με παρόμοιο τρόπο όπως για τις δύο παραπάνω συναρτήσεις, μόνο που γίνεται χρήση του λεξικού info για την αντιστοίχιση πατέρα-παιδιού-οδηγίας.

Ο Autograder σε αυτή την συνάρτηση επιστρέφει αποτέλεσμα 3/3.

Question 4: A* Search

Υλοποιήθηκε η συνάρτηση aStarSearch στο search.py με παρόμοιο τρόπο όπως η uniformCostSearch παραπάνω με τροποποιήσεις για τον υπολογισμό της τιμής f που περιλαμβάνει το κόστος της διαδρομής μέχρι και το παιδί(κόστος μέχρι τον πατέρα συν το κόστος της ακμής πατέρα-παιδιού), όπως στη UCS συν επιπλέον την τιμή της συνάρτησης heuristic που δίνει σαν όρισμα ο χρήστης.

Χρησιμοποιήθηκε η έτοιμη δομή Ουράς Προτεραιότητας από το util.py(util.priorityQueue()) σαν σύνορο, δημιουργήθηκε μια λιστα explored για να κρατάει τους κόμβους που έχουμε επισκεφτεί και έγινε χρήση λεξικού info που κρατάει για κάθε state που συναντάμε ένα tuple που περιέχει τον πατέρα του, την οδηγία από τον πατέρα στο παιδί, όπως αυτή δίνεται απο το problem.getSuccesors(-state-), καθώς επίσης και το κόστος της διαδρομής έως και αυτό το state και μια boolean μεταβλητή που δίνει την πληφορία για το εάν ένα state βρίσκεται ή όχι μέσα στο σύνορο. Για το Start State, μόλις αυτό μπει στο σύνορο το tuple αυτό θα είναι (0, 0, 0, True).

Η ανοικοδόμηση του μονοπατιού γίνεται με παρόμοιο τρόπο όπως για τις δύο παραπάνω συναρτήσεις, μόνο που γίνεται χρήση του λεξικού info για την αντιστοίχιση πατέρα-παιδιού-οδηγίας.

Ο Autograder σε αυτή την συνάρτηση επιστρέφει αποτέλεσμα 3/3.

Question 5: Finding All the Corners

Τροποποιήθηκε η κλάση CornersProblem στο searchAgents.py και συγκεκριμένα οι συναρτήσεις της getStartState, isGoalState, getSuccessors. Πλέον το state δεν θα είναι μόνο η θέση στην οποία βρισκόμαστε, αλλά ένα tuple που θα περιέχει τη θεση και ένα tuple με το πόσες γωνίες έχουμε επισκεφτεί. Πιο αναλυτικα οι συναρτήσεις:

1) getStartState: Εφόσον το ζητούμενο έχει αλλάξει και πλέον πρέπει να ανακαλύψουμε και τις 4 γωνίες, το νέο Start State θα είναι ένα tuple που θα περιέχει την αρχική θέση του προβλήματος(startingPosition) και ενα κενό tuple με όνομα visitedCorners.

2) isGoalState: Το ζητούμενο της άσκησης είναι να επισκεφτούμε και τις τέσσερις γωνίες του λαβυρίνθου. Επομένως, το Goal State μας ανεξάρτητα από τη θέση του θα είναι αυτό που θα περιέχει ένα tuple μήκους 4.

3) getSuccessors: Για κάθε διάδοχο state του τρέχοντος, προστίθεται στη λιστα με τους successors ένα tuple της μορφης((position, tuple(visitedCorners)), action, cost). Το visitedCorners εξαρτάται από το εάν ο διάδοχος κόμβος είναι γωνία που δεν έχουμε ξαναεπισκεφτεί. Εαν είναι, τότε το tuple την περιλαμβάνει, εάν όχι τότε είναι ίδιο με αυτό του πατέρα του. Το cost είναι πάντα ένα.

Σημείωση: Εάν το επόμενο state ανήκει στις γωνίες που δεν έχουμε επισκεφτεί, τότε αφού το προσθέσω στη λίστα με τα visitedCorners κάνω μία ταξινόμηση στη λίστα για να έχουν κάθε φορά ίδια σειρά οι γωνίες που περιέχει. Εάν αυτό δεν γίνει και ο αλγόριθμος ξανασυναντήσει το ίδιο state το οποίο περιέχει διαφορετική σειρά στο visitedCorners, θα το θεωρήσει διαφορετικό από αυτό που είναι αποθηκευμένο στην λίστα explored σε όλους τους παραπάνω αλγορίθμους αναζήτησης και θα κάνει περιττες επεκτάσεις.

Ο Autograder σε αυτό το ερώτημα δε βγάζει κάποιο αποτέλεσμα αφού στηρίζεται στο ερώτημα 2, το οποίο θεωρεί λάθος, όπως ανέφερα παραπάνω.

Question 6: Corners Problem: Heuristic

Υλοποιήθηκε η συνάρτηση cornersHeuristic στο searchAgents.py. Από το state το οποίο μελετάμε πληροφορούμαστε για τη θέση μας στο πρόβλημα και τη λιστα των γωνιών που έχουμε ήδη επισκεφτεί. Έχοντας τη λίστα με τις γωνίες που έχουμε επισκεφτεί και γνωρίζοντας όλες τις γωνίες του προβλήματος, μπορούμε να κατασκευάσουμε μια λίστα με τις γωνίες που δεν έχουμε επισκεφτεί ακόμα. Εάν αυτή η λίστα είναι κενή, τότε βρισκόμαστε στο Goal State, οπότε η ευρετική συνάρτηση επιστρέφει 0. Αλλιώς, κατασκευάζουμε με τη χρήση της συνάρτησης permutations της Python όλα τα πιθανά σενάρια όσον αφορά τη σειρά με την οποία θα επισκεπτούμε τις υπόλοιπες γωνίες και τα αποθηκεύουμε σε μία λίστα options. Για κάθε επιλογή που έχουμε υπολογίζουμε την απόσταση Manhattan μεταξύ της θέσης μας στο πρόβλημα και της πρώτωνίας σε αυτό το σενάριο, έπειτα την απόσταση Manhattan μεταξυ αυτής της γωνίας και της επόμενης κ.ο.κ. και αποθηκεύουμε τα αποτελέσματα σε μία λίστα totalDistances. Έτσι, αυτή η λίστα θα περιέχει μια προσέγγιση για τα κόστη των διαδρομών για κάθε πιθανό σενάριο που μπορούμε να ακολουθήσουμε. Ο αλγόριθμος επιστρέφει το μικρότερο από αυτά τα κόστη.

Η ευρετική συνάρτηση cornersHeuristic επιστρέφει την μικρότερη διαδρομή που μπορούμε να κάνουμε με οριζόντιες και κάθετες κινήσεις έτσι ώστε να καλύψουμε και τις τέσσερις γωνίες. Επομένως, θα επιστρέφει πάντα λιγότερα ή ίσα βήματα με αυτά που χρειάζονται στο αρχικό πρόβλημα και άρα είναι παραδεκτή. Πρόκειται για τη λύση χαλαρωμένης μορφής του cornersProblem η οποία δεν λαμβάνει υπόψη της τους τοίχους.

Εφόσον με τα Manhattan Distances κινούμαστε οριζοντια και κάθετα, η μετακίνηση απο το state στο οποίο βρισκόμαστε σε ένα γειτονικό του(κόστος 1) θα μας “γλιτώσει” ένα βήμα από την συνολική διαδρομή που θα κάνει. Επομένως, η ευρετικη του state μας μας επιστρέφει έναν αριθμό βημάτων που δεν είναι μεγαλύτερος από αυτόν που επιστρέφει η ευρεστική του παιδιού του συν το κόστος της μεταξύ τους ακμής. Άρα, η ευρετική είναι και συνεπής.

Question 7: Eating All The Dots

Υλοποιήθηκε η συνάρτηση foodHeuristic στο searchAgents.py. Υπολογίζεται η πραγματική απόσταση μέσα στον λαβύρινθο(mazeDistance) μεταξύ του state στο οποίο βρισκόμαστε και κάθε state που περιέχει φαγητό και επιστρέφεται η μεγαλύτερη από αυτές.

Πρόκειται για χαλαρωμένη έκδοση του foodSearchProblem, κατά την οποία αναζητούμε το πιο απομακρυσμένο food state. Είναι παραδεκτή, αφού για να καλύψουμε όλα τα food states από το αρχικό μας state θα πρέπει να κάνουμε τουλάχιστον όσα βήματα χρειάζονται για να πάμε στο πιο μακρινό, άρα δεν υπερεκτιμάται το αποτέλεσμα.

Ξέρουμε ότι όλα τα states έχουν μεταξύ τους απόσταση 1. Έστω x η απόσταση μεταξύ του state στο οποίο βρισκόμαστε με το πιο μακρινό του food state. Για ένα γειτονικό του state, υπάρχουν οι εξης περιπτώσεις:

1) Εαν το πιο μακρινό food state από αυτό είναι το ίδιο και το γειτονικό state είναι πιο μακρια από το food state συγκριτικά με το state μας, η απόσταση μεταξυ τους θα ειναι x + 1. Θα ισχτεί ότι x <= x + 1 + 1.

2) Εαν το γειτονικό state είναι πιο κοντά στο το food state από το state μας η απόσταση μεταξυ τους θα ειναι x – 1. Θα ισχύει ότι x <= x – 1 + 1.
3) Εαν το γειτονικό state έχει διαφορετικό πιο μακρινό food state από το state μας, τότε το state μας θα έχει από αυτό το food state απόσταση το πολύ x – 1. Θα ισχύει, x <= x – 1 + 1.

Συμπεραίνω ότι σε κάθε περίπτωση ισχύει η συνθήκη της συνέπειας. Επομένως, η foodHeuristic ειναι συνεπής.

Question 8: Suboptimal Search

Συμπληρώθηκε η συνάρτηση isGoalState της κλάσης anyFoodSearchProblem του searchAgents.py και τροποιποιήθηκε η συνάρτηση findPathToClosestDot της κλάσης ClosestDotSearchAgent του searchAgents.py. Πιο αναλυτικα:

1) isGoalState: Εάν το state στο οποίο βρισκόμαστε περιέχει φαγητό, είναι Goal State
2)findPathToClosestDot: Επιστρέφει το αποτέλεσμα της αναζήτησης με bfs για το anyFoodSearchProblem, δηλαδή θα επιστρέψει το πιο σύντομο μονοπάτι προς οποιοδηποτε food state ή αλλιώς το μονοπάτι προς το κοντινότερο food state.