Τεχνητή Νοημοσύνη
Εργασία 1 - Σχόλια Κώδικα Προβλήματος 6
Project 1: Search in Pacman

Λάμπρου Ιωάννης

7 Νοεμβρίου 2016

Για τις ανάγκες του Project 1: Search in Pacman από τα Pacman Projects, υλοποίησα την παρακάτω κλάση κόμβου (SearchNode):

```
class SearchNode:

    def __init__(self, state, action = None, parent = None, cost = 0):
        self.state = state
        self.action = action
        self.parent = parent
        self.cost = cost

    # Recursively get the actions that led to this state (Node)
    def get_solution(self):
        solution = []

        if self.parent is not None:
            solution = self.parent.get_solution()
            solution.append(self.action)

        return solution
```

Ο κόμβος (SearchNode) αποθηκεύει μια κατάσταση, την πράξη, ενέργεια που είχε ως αποτέλεσμα αυτή τη κατάσταση, δείχνει σε έναν κόμβο ¨πατέρα¨, στον οποίο είναι αποθηκευμένη μια προηγούμενη κατάσταση και αποθηκεύει το κόστος όλων των πράξεων που έγιναν για να φτάσουμε στην κατάσταση αυτή. Η συνάρτηση get_solution παίρνει ως όρισμα έναν κόμβο, βρίσκει αναδρομικά τις πράξεις που έγιναν για να φτάσουμε στον κόμβο αυτόν, (μέσω των κόμβων ¨πατέρων¨) και επιστρέφει μια λίστα με αυτές. Ο παραπάνω κώδικας της κλάσης SearchNode βρίσκεται στο τέλος του αρχείου search.py.

Οι αλγόριθμοι στα ερωτήματα 1, 2, 3, 4 είναι παρόμοιοι (Graph Search) Το μόνο που αλλάζει είναι το πως (με ποιά δομή) υλοποιείται το σύνορο «fringe).

- Στο ερώτημα 1, το σύνορο υλοποιείται με μια δομή σωρού.
- Στο ερώτημα 2, το σύνορο υλοποιείται με μια δομή ουράς.
- Στο ερώτημα 3, το σύνορο υλοποιείται με μια δομή ουράς προτεραιότητας ενώ, ο κάθε κόμβος που εισάγεται παίρνει ως προτεραιότητα το κόστος των πράξεων για να φτάσουμε σε αυτόν τον κόμβο.

- Στο ερώτημα 4, το σύνορο υλοποιείται με μια δομή ουράς προτεραιότητας ενώ, ο κάθε κόμβος που εισάγεται παίρνει ως προτεραιότητα το κόστος των πράξεων για να φτάσουμε σε αυτόν τον κόμβο, συν την τιμή που μας δίνει η ευρετική συνάρτηση που χρησιμοποιούμε, για τον κόμβο αυτόν.

Για την επίλυση του ερωτήματος 5, θεώρησα την κατάσταση state του pacman ως ένα tuple, που περιέχει το που βρίσκεται ο pacman αλλά και ένα tuple με 4 boolean τιμές, μία για κάθε γωνία του προβλήματος, με τη σειρά που υπάρχουν στο self.corners, οι οποίες γίνονται true όταν περάσει από αυτές ο pacman άρα και δείχνουν σε μία κατάσταση του pacman, πόσες γωνίες έχει εξερευνήσει.

Για το ερώτημα 6, η ευρετική συνάρτηση που χρησιμοποίησα βρίσκει αρχικά την απόσταση Μανχάταν από την κατάσταση του pacman προς την κοντινότερη γωνία που δεν έχει εξερευνήσει, έπειτα προσθέτει σε αυτή την απόσταση Μανχάταν από αυτή τη γωνία προς την κοντινότερη γωνία που δεν έχει εξερευνήσει, και συνεχίζει μέχρις ότου έχει υπολογίσει τις μικρότερες αποστάσεις όλων των γωνιών που δεν έχει εξερευνήσει ο pacman μεταξύ τους, και τέλος τις επιστρέφει.

Για το ερώτημα 7, η ευρετική συνάρτηση που χρησιμοποίησα, στην κατάσταση που δέχεται, θεωρεί έναν γράφο με κόμβους όλα τα φαγητά που δεν έχει φάει ο pacman, με ακμές όλες τις δυνατές συνδέσεις των κόμβων αυτών μεταξύ τους, και βάρος της κάθε ακμής, την μικρότερη πραγματική απόσταση μεταξύ των δύο κόμβων - άκρων της ακμής μέσα στον λαβύρινθο (υπολογίζεται με την έτοιμη συνάρτηση mazeDistance). Έπειτα, εφαρμόζει τον αλγόριθμο Kruskal και βρίσκει το ελάχιστο επικαλυπτικό δένδρο για τον γράφο αυτό. Προσθέτει τα βάρη όλων των ακμών του δέντρου αυτού και τέλος, στο άθροισμα αυτό προσθέτει και την μικρότερη πραγματική απόσταση μεταξύ του pacman και του κοντινότερου φαγητού (που δεν έχει φάει) και το επιστρέφει. Ο αλγόριθμος Kruskal έχει υλοποιηθεί χρησιμοποιώντας δομές της python.

Για το ερώτημα 8, στην συνάρτηση findPathToClosestDot δημιουργείται ένα πρόβλημα τύπου AnyFoodSearchProblem και χρησιμοποιείται ο αλγόριθμος BFS για την επίλυσή του. Στη κλάση AnyFoodSearchProblem, η συνάρτηση isGoalState ελέγχει μόνο αν η κατάσταση (θέση) που δέχεται, έχει φαγητό. Αν έχει φαγητό, τότε και θα είναι κατάσταση στόχου. (Αφού ο ClosestDotSearchAgent ψάχνει επαναληπτικά για το κοντινότερο φαγητό)