Project 1 - Τεχνητή Νοημοσύνη

Παναγιώτα Γύφτου , Α.Μ.: 1115201900318

Νοέμβριος 2022

---

*Question*1

Η αναζήτηση DFS ξεκινά με την λήψη του κόμβου της αρχικής κατάστασης και ελέγχοντας τον, αν είναι και κόμβος στόχου του προβλήματος.Εάν η υπόθεση αυτή είναι αληθής τότε ως μονοπάτι λύσης επιστρέφεται μια κενή λίστα, διαφορετικά ο αρχικός κόμβος αποθηκεύεται στο σύνορο, όπου για την υλοποίηση του συνόρου επιλέχθηκε η χρήση της δομής στοίβας. Αφού πραγματοποιηθεί η αποθήκευση στην στοίβα, ο αλγόριθμος μέσα σε έναν $while$ βρόγχο αναζητεί τον κόμβο στόχου μέχρι η στοίβα να μείνει κενή, δηλαδή να έχουν διερευνηθεί όλοι οι κόμβοι.Το πρώτο βήμα που γίνεται στο σώμα του βρόχου είναι η εξαγωγή του τελευταίου στοιχείου που μπήκε στην στοίβα, έπειτα ελέγχεται αν αυτός ο κόμβος είναι ο κόμβος στόχου.Στην περίπτωση που δεν είναι αποθηκεύουμε τον κόμβο στο σετ με τους επισκεπτόμενους κόμβους ώστε να αποφευχθεί τυχόν δεύτερη μελλοντική επίσκεψη.Στην συνέχεια γίνεται με την βοήθεια της μεθόδου $getSuccessors()$ η λήψη των διαδόχων του τρέχοντος κόμβου, οι $successors$ και οι πληροφορίες τους, αποθηκεύονται στην μεταβλητή $listSuccessors$.Ακολουθεί ένα $while\ loop$, στο οποίο ελέγχονται ένας ένας οι διάδοχοι, εάν υπάρχουν "καταγεγραμμένοι" στο σετ $expanded$ ως κόμβοι που έχουν επισκεφθεί στο παρελθόν.Στην περίπτωση αυτή τον προσθέτουμε στο σύνορο (στο $stack$) και μέσα σε ένα $dictionary$ αποθηκεύουμε τον τρέχον διάδοχο αντιστοιχίζοντας του το μονοπάτι "την ακμή" πατέρας που είναι ο τρέχον κόμβος και παιδί που είναι ο τρέχον διάδοχος.Σε αυτό το λεξικό αποθηκεύονται όλοι οι έγκυροι διαδόχοι, όταν τελειώσει ο έλεγχος όλων των διαδόχων τότε στην λίστα $parentKids$ αποθηκεύουμε το ζεύγος πατέρα - παιδιά δηλαδή αποθηκεύουμε τον τρέχοντα κόμβο (προκάτοχος) με τους έγκυρους διαδόχους του, όπου τα παιδιά με τις πληροφορίες τους είναι το λεξικό που δημιουργήσαμε προηγουμένως.Εάν βρισκόμαστε στην περίπτωση όπου είναι ο κόμβος στόχου τότε καλείται η συνάρτση $pathConstruction()$, ώστε να λάβει το μονοπάτι λύσης και να το επιστρέψει.Η $pathConstruction()$ έχοντας λάβει ως όρισμα την λίστα $parentKids$, αντεστραμμένη διότι θέλουμε να ανατρέξουμε τα πρόσφατα δεδομένα προς τα παλαιότερα, ώστε να υλοποιήσουμε μια υλοποίηση οπισθοδρόμησης.

*Question*2

Ο αλγόριθμος αναζήτησης BFS είναι ίδιος με τον αλγόριθμο του DFS με τις διαφορές, ότι αρχικά χρησιμοποιείται η δομή της ουράς και δεύτερον ότι προστίθεται ο έλεγχος, αν υπάρχει αποθηκευμένος ο τρέχον εξετάσιμος διάδοχος και στο σύνορο, στο σημείο προσθήκης των διαδόχων στην ουρά.

*Question*3

Η λογική είναι ίδια με αυτή τον προηγούμενων αλγορίθμων.Η δομή που χρησιμοποιείται εδώ που αναπαραστεί το σύνορο είναι μια ουρά προτεραιότητας.Σε αυτόν το αλγόριθμο δεν χρησιμοποιούμε την συνάρτηση $pathConstruction$, έχω ένα λεξικό ($path$) στο οποίο αποθηκεύω το μονοπάτι των διαδόχων, δηλαδή ποια διαδρομή ακολουθήθηκε για να φτάσουμε στους διαδόχους, όταν βρεθεί ο κόμβος στόχου τότε βρίσκω το μονοπάτι που αντιστοιχεί στον συγκεκριμένο κόμβο που είναι αποθηκευμένος στο λεξικό αυτό.Σαν προτεραιότητα έχω ορίσει να είναι το μήκος της διαδρομής προέλευσης του κάθε κόμβου που εισέρχεται στο σύνορο.Επομένως αν κάποιος διάδοχος ελεγχθεί ξανά και είναι ήδη στο σύνορο συγκρίνουμε το κόστος των διαδρομών (παλιού-νέου), και κρατάμε αυτό με την μικρότερη προτεραιότητα, το ίδιο συμβαίνει και στο λεξικό $path$,κρατάμε το βέλτιστο μονοπάτι.

*Question*4

Ο αλγόριθμος αναζήτησης $A - star$ είναι ίδιος με τον αλγόριθμο του UCS με την διαφορά ότι τώρα λαμβάνουμε υπόψη και την ευρετική συνάρτηση.Δηλαδή για τον υπολογισμό του κόστους ενός μονοπατιού χρησιμοποιούμε και την πραγματική τιμή του κόστους του μονοπατιού αλλά και την τιμή της ευρετικής συνάρτησης. $pathCost = getCostOfActions() + heuristic(, problem)$

*Question*5

• *init method* :

Έχει προστεθεί ένα $tuple$ που περιέχει 4 $tuples$, αυτό λειτουργεί ως κατάλογος κατάστασης φαγωμένων μπαλών, δηλαδή περιέχονται οι 4 γωνίες συνοδευόμενες με την πληροφορία αν έχουν φαγωθεί.Στην πρώτη θέση των εσωτερικών $tuple$ βρίσκεται η συντεταγμένη ενός $corner$ και στην δεύτερη θέση ένας σημαφόρος που δείχνει, αν η μπάλα στην αντίστοιχη θέση είναι φαγωμένη.Αρχικοποιούμε όλους τους σημαφόρους με 0, καθώς καμία μπάλα στην αρχή δεν έχει φαγωθεί.

• *getStartState method* :

Επιστρέφει την αρχική θέση του *pacman* και το *tuple* κατάστασης φαγωμένων μπαλών, το οποίο ορίστηκε παραπάνω στην μέθοδο *init*.

• *isGoalState method* :

Αρχικά ελέγχει αν η θέση της κατάστασης εξέτασης είναι μια θέση *corner*, αυτή η πληροφορία αποθηκεύεται σε μια μεταβλητή *boolean*. Στην συνέχεια μετράει μέσω του *tuple* κατάστασης φαγωμένων μπαλών, πόσες μπάλες έχουν φαγωθεί.Αφού τελειώσει με την καταμέτρηση και αποθηκεύσει σε μια μεταβλητή και αυτή την πληροφορία, έπειτα ελέγχει αν η τρέχουσα θέση εξέτασης είναι θέση *corner* και αν έχουν φαγωθεί και οι 4 μπάλες τότε η κατάσταση εξέτασης είναι κατάσταση στόχου και επιστρέφεται *boolean* τιμή $True$ διαφορετικά $False$.

• *getSuccessors method* :

Με την βοήθεια της υπόδειξης που μας παρέχετε από τον κώδικα, βρίσκουμε τις πιθανές θέσεις για την επόμενη κατάσταση.Για κάθε μια θέση εξετάζεται αν είναι τοίχος.Στην περίπτωση που δεν είναι εμπόδιο τότε καλούμε την:

$$updateEatenCornerBalls()$$

η οποία ενημερώνει το *tuple* κατάστασης φαγωμένων μπαλών, ώστε αν η θέση του διαδόχου είναι θέση *corner* τότε να θέσει τον αντίστοιχο σημαφόρο ίσο με 1 αν δεν είναι. Συνεχίζοντας φτιάχνει την κατάσταση του διαδόχου και την προσθέτει σε μία λίστα (*successors*) μαζί με τους άλλους έγκυρους διαδόχους.Όταν τελειώσει με την εύρεση των διαδόχων τότε επιστρέφει την λίστα (*successors*) με αυτούς.

*Question*6

Η ιδέα αυτου του ευρετικού μηχανισμού είναι να βρίσκει από τη θέση της τρέχουσας κατάστασης την κοντινότερη απόσταση $Manhattan$ από τις γωνίες που δεν έχουν φαγωθεί οι μπάλες.Αρχικά επιλέγεται η μικρότερη απόσταση, η τρέχουσα θέση πλέον είναι η θέση που επιλέχθηκε και με την ίδια διαδικασία επιλέγεται ξανά η κοντινότερη απόσταση $Manhattan$ από τις γωνίες που δεν έχουν φαγωθεί οι μπάλες.Η επανάληψη αυτής της διαδικασίας συνεχίζεται μέχρι να φαγωθούν όλες οι γωνιακές μπάλες.

Αρχικά στο σώμα της ευρετικής συνάρτησης ελέγχεται εάν η τρέχουσα θέση εξέτασης είναι κατάσταση στόχου.Αν είναι επιστρέφεται μηδενικό κόστος, διαφορετικά, καλείται η $subGoal()$ συνάρτηση, η οποία βρίσκει και επιστρέφει τον επόμενο υπο-στόχο με την μικρότερη απόσταση. Ο υπολογισμός της απόστασης γίνεται με την βοήθεια της μεθόδου $Manhattan$.Στην συνέχεια εισερχόμαστε σε έναν ατέρμονο βρόγχο, στον οποίο ελέγχεται αρχικά αν η τρέχουσα θέση είναι θέση *corner*. Σε αυτή την περίπτωση ανανεώνεται το *tuple* κατάστασης φαγωμένων

μπαλών μέσω της μεθόδου $updateEatenCornerBalls()$, έπειτα εξετάζεται εάν η τρέχουσα κατάσταση είναι και κατάσταση στόχου, εάν είναι επιστρέφεται το συνολικό κόστος, διαφορετικά, καλείται η $subGoal()$, για να λάβουμε τον επόμενο υπό-στόχο και ανανεώνουμε τα δεδομένα μας δηλαδή το καινούριο υποστόχο μας και το συνολικό κόστος. Διαφορετικά,αν η τρέχουσα θέση δεν είναι θέση $corner$, ανανεώνουμε μόνο τα δεδομένα μας, δηλαδή, το καινούριο υπό-στόχο μας και το συνολικό κόστος.

*Question*7

Η ιδέα αυτου του ευρετικού μηχανισμού είναι να βρίσκει αρχικά την απόσταση του μακρυνότερου φαγητού από την τρέχουσα θέση στην οποία βρίσκεται ο $pacman$. Αφού βρεθεί η πιο ακριβή διαδρομή με την βοήθεια της μεθόδου $mazeDistance$, η οποία είναι μια ακριβής μέθοδος όσον αφορά το υπολογισμό απόστασης με εμπόδια (δηλαδή με τους τοίχους), προσθέτουμε στην μέγιστη απόσταση και το πλήθος των μπαλών που βρίσκονται κοντά στην τρέχουσα θέση του πακμαν.Συγκεκριμένα αν η μπάλα με την μέγιστη απόσταση είναι δεξιά του $pacman$ και βρίσκονται κοντινές μπάλες στα αριστερά του πακμαν τότε τις τρώει, αντίστοιχα αν ο πακμαν βρίσκεται στα δεξιά της μπάλας με την μεγαλύτερη απόσταση και υπάρχουν μπάλες κοντά στα δεξιά του πακμαν τις τρώει.Τέλος αν ο πακμαν βρίσκεται στην ίδια στήλη με την μπάλα με την μέγιστη απόσταση και υπάρχουν δεξιά και αριστερά μπάλες κοντά του τις τρώει.

Παρατήρησα ότι στο τεστ $foodheuristicgradetricky.test$ το πρόγραμμα αργεί με αποτέλεσμα να τρέχει σε 26 δευτερόλεπτα

*Question*8

• $isGoalState\ method$:

Στην μέθοδο αυτή, ελέγχουμε αν στην θέση της τρέχουσας κατάστασης υπάρχει φαγητό.Εάν υπάρχει επιστρέφουμε τιμή $boolean\ True$, αν η λίστα με τις συντεταγμένες των μπαλών είναι κενή τότε πάλι επιστρέφουμε $boolean\ True$, διαφορετικά $boolean\ False$.

• $findPathToClosestDot\ method$:

Για την εύρεση της κοντινότερης μπάλας κάνω λήψη της λίστας συντεταγμένων των μπαλών. Με την βοήθεια της μεθόδου $Manhattan$ βρίσκω την πιο κοντινή σε απόσταση μπάλα.Ορίζω ως στόχο του προβλήματος τις συντεταγμένες της μπάλας που βρήκα ότι απέχει λιγότερη απόσταση.Τέλος καλώ την $aStarSearch()$ με ορίσματα, το τροποποιημένο $problem$ και με ευρετικό μηχανισμό την συνάρτηση $manhattanHeuristic$.