Απόστολος Φλωράκης
1115201400217

-Έχει τοτποθετηθεί στα παραδοτέα και το αρχείο **util.py** καθώς σε αυτό έχει προστεθεί μια κλάση Node που όρισα ώστε να ακολουθήσω το format των διαφανειών ως προς την αναπαράσταση κάθε κόμβου του γράφου, μια συνάρτηση existOnFrontier στις κλάσεις Stack,Queue και PriorityQueue για την εύρεση του άμα ένας κόμβος υπάρχει στο αντίστοιχο frontier καθώς και έχει τροποποιηθεί η συνθήκη της update συνάρτησης της δομής PriorityQueue (γραμμή 234) ωστέ να είναι συνεπής και να εκτελεί σωστό έλεγχο για τις δομή Node που χρησιμοποιώ. Στη δομή Node αποθηκεύω τις συντεταγμένες του κόμβου (state), τον πατέρα του (parent), την κίνηση που πραγματοποιήθηκε ώστε από τον πατέρα να προκύψει ο κόμβος αυτός (action), καθώς και το κόστος του μονοπατιού μέχρι τον κόμβο αυτό Cost. Αφού βρούμε τον τελικό κόμβο και καλώντας την συνάρτηση getPath() δημιουργούμε το μονοπάτι της λύσης μας καθώς για κάθε κόμβο γνωρίζουμε τόσο την κίνηση από την οποία προήλθε τόσο και τον πατέρα του, οπότε και μπορούμε επαναληπτικά να οδηγηθούμε από τον GoalState κόμβο προς τον αρχικό.
-Όσον αφορά τους αλγορίθμους **DFS,BFS,UCS,A*** έχει ακολουθηθεί το format των διαφανειών του μαθήματος όμως για να λαμβάνω από τον autograder σωστά αποτελέσματα δεδομένου πως τους χρησιμοποιώ και σε άλλα ερώτήματα, έχω κάνει ορισμένες αλλαγές παραδοχές. Ο έλεγχος για GoalState ειναι σχολιασμένος μεσα στον έλεγχο για το άμα ο κόμβος δεν ανήκει στο explored set η στο frontier οπότε και δεν εκτελείται εκεί όπως στις διαφάνειες και πραγματοποιείται εκτός και στους τρεις αλγορίθμους. Επίσης στον DFS αλγόριθμο ο έλεγχος πριν γινει το push ενός successor περιέχει μόνο το άμα αυτός δεν ανήκει στο explored set και όχι την ύπαρξη του στο frontier όπως θα έπρεπε (γραμμή 115), αναφέρεται και σε σχόλιο (γραμμη 114).
-Για τον ορισμό του **CornersProblem** ως αρχική κατάσταση ορίζω τις αρχικές συντεταγμένες καθώς και μια λίστα που περιέχει τις συντεταγμένες των κόμβων corner που θα πρέπει να επισκεφθώ. Αντίστοιχα ο έλεγχος για GoalState απαιτεί η λίστα αυτή με τα corners να είναι άδεια δεδομένου πως κάθε φορά που επισκέπτομαι ένα corner το αφαιρώ απο τη λίστα. Για τους succesors ενός κόμβου άμα οι συντεταγμένες αυτών (φυσικά δεν συμπίπτουν με κάποιο wall) υπάρχουν στη λίστα με τα corners, αφαιρείται από τη λίστα του state του successor το αντίστοιχο corner που συμπίπτει με αυτόν. Άρα για παράδειγμα έχοντας επισκεφτεί 3 corners και και άμα ο succesor του κόμβου που εξετάζουμε είναι το 4ο και τελευταίο corner πριν γίνει push στο frontier αφαιρείται το corner αυτό από τη λίστα του state του, ώστε όταν γίνει ο έλεγχος για goalstate λίστα να βρεθεί κενή οπότε και να διαπίστωθεί η λύση του προβλήματος.
- Στην **cornersHeuristic** για τα corners που δεν έχω επισκεφθεί υπολόγίζω επαναληπτικά ένα άθροισμα των μικρότερων manhattan αποστάσεων μεταξύ σημείου και corner θέτοντας αρχικά ως σημείο τις συντεταγμένες του state μου και σε κάθε μια επανάληψη θέτοντας το ως το corner στο οποίο αντιστοιχεί η μικρότερη επιμέρους απόσταση. Πχ για τρία εναπομείναντα corners βρίσκω την απόσταση σημείου-corner3 ως μικρότερη την προσθέτω στο αποτέλεσμα μου. Συνεχίζω βρίσκοντας για παράδειγμα ως μικρότερη την απόσταση corner3-corner1 κοκ.
-Στη **foodHeuristic** χρησιμοποιώ την συνάρτηση mazeDistance . Η συνάρτηση αυτη χρησιμοποιώντας την bfs επιστρέφει την πραγματική απόσταση σε αριθμό κόμβων δυο δεδομένων σημείων έτσι και μπορώ να υπολογίσω τη μεγαλύτερη απόσταση του σημείου μου από κάποιο food που θα είναι και το αποτέλεσμα που θα επιστρέψει. Η συνάρτηση αργεί αρκετά καθώς εμπεριέχει πάρα πολλές bfs κλήσεις κάνει όμως expand λίγους κόμβους οπότε και λαμβάνει καλή βαθμολογία απο τον autograder.
-Στο **findPathToClosestDot** ερώτημα αρχικά τίθεται ως goalstate του AnyFoodSearchProblem ο έλεγχος για τον αν ο κόμβος αυτός περιέχει φαγητό, και στο findPathToClosestDot απλά καλείται η bfs συνάρτηση. Ο τρόπος εύερεσης του αποτελέσματος είναι είναι επαναλαμβανόμενες bfs κλήσεις καθώς για μια δεδομενο κόμβο και μια bfs έχοντας ως goalstate παραπάνω, βρίσκουμε το μονοπάτι προς το πλησιέστερο food . Η πρώτη bfs βρίσκει το κοντινότερο food στην αρχική μας θέση, η επόμενη bfs το κοντινότερο food από την καινούρια θέση που είμαστε κοκ. Η bfs βρίσκει για την ακρίβεια ένα από τα κοντινότερα food άμα αυτά είναι περισσότερα από ένα αλλά αυτό δεν μας

πειράζει καθώς θέλουμε κάθε φορά να κάνουμε μια greedy επιλογή ενός κοντινότερου από τη θέση που είμαστε. Η σύνθεση των επιμέρους paths που επιστρέφουν κάθε μια bfs κλήση μέχρι να επισκεφτούμε όλα τα foods μας δίνει τη λύση.