Σχεδιαστικές επιλογές

Ονοματεπώνυμο : Ψαθόπουλος Φανούριος Στυλιανός

Αριθμός Μητρώου : 1115201400320

Question 1

Απο τη στιγμή που θέλω να υλοποιήσω DFS δηλαδή να ψάξω σε βάθος η δομή του frontier θα πρέπει να είναι Στοίβα (LIFO) . Η δομή του αλγόριθμου είναι η εξής :

- Αρχικοποίηση με το position της αρχής και δίπλα το μονοπάτι για μέχρι εκείνο το σημείο. (frontier.push([problem.getStartState(),[]]))
- Αρχικοποίηση μιας δομής με τους κόμβους που ήδη έχουν επεκταθεί.
- Στη συνέχεια σε ένα infinite loop βγάζω από το frontier κόμβους είτε μέχρι να αδειάσει είτε να φτάσω στο στόχο .
- Αν ο κόμβος που έχει επιλεχθεί δεν υπάρχει στην δομή με όσους έχουν επεκταθεί τον επεκτείνω.
- Τα παιδιά του τα βάζω όλα στο frontier. Θα μπορούσα να κάνω τον έλεγχο να μην τα βάζω στο frontier αν υπάρχουν ήδη εκεί ή αν έχουν γίνει σαν κόμβοι expand , αφού ξέρω εξ'ορισμού ότι οποιοδήποτε κόμβος υπάρχει ήδη στο frontier ή έχει γίνει expand θα έχει μικρότερο κόστος από την επανεμφάνιση του. Για λόγους autograder τα έβαλα όλα και επέτρεψα στον αλγόριθμο να τα απορρίψει εκ νέου αφού πχ ο κόμβος Β θα έχει επεκταθεί με το μικρότερο κόστος μονοπατιού προς αυτόν άρα ένα επόμενο κόστος απλά θα αγνοηθεί.
- Σε κάθε κόμβο που επεκτείνω και βάζω τους απογόνους στο frontier φροντίζω να αποθηκεύω το path μέχρι εκεί , έτσι ώστε όταν φτάσω στον κόμβο στόχο να ξέρω τη διαδρομή για να την κάνω return.

Question 2

Απο τη στιγμή που θέλω να υλοποιήσω BFS δηλαδή να κάνω αναζήτηση κατα πλάτος η δομή του frontier θα πρέπει να είναι Ουρά (FIFO) . Η δομή του αλγόριθμου είναι η εξής :

- Αρχικοποίηση με το position της αρχής και δίπλα το μονοπάτι για μέχρι εκείνο το σημείο. (frontier.push([problem.getStartState(),[]]))
- Αρχικοποίηση μιας δομής με τους κόμβους που ήδη έχουν επεκταθεί.
- Στη συνέχεια σε ένα infinite loop βγάζω από το frontier κόμβους είτε μέχρι να αδειάσει είτε να φτάσω στο στόχο .
- Αν ο κόμβος που έχει επιλεχθεί δεν υπάρχει στην δομή με όσους έχουν επεκταθεί τον επεκτείνω.
- Τα παιδιά του τα βάζω όλα στο frontier. Θα μπορούσα να κάνω τον έλεγχο να μην τα βάζω στο frontier αν υπάρχουν ήδη εκεί ή αν έχουν γίνει σαν κόμβοι expand , αφού ξέρω εξ'ορισμού ότι οποιοδήποτε κόμβος υπάρχει ήδη στο frontier ή έχει γίνει expand θα έχει μικρότερο κόστος από την επανεμφάνιση του. Για λόγους autograder τα έβαλα όλα και επέτρεψα στον αλγόριθμο να τα απορρίψει εκ νέου αφού πχ ο κόμβος Β θα έχει επεκταθεί με το μικρότερο κόστος μονοπατιού προς αυτόν άρα ένα επόμενο κόστος απλά θα αγνοηθεί.
- Σε κάθε κόμβο που επεκτείνω και βάζω τους απογόνους στο frontier φροντίζω να αποθηκεύσω το path μέχρι εκεί , έτσι ώστε όταν φτάσω στον κόμβο στόχο να ξέρω τη διαδρομή για να την κάνω return.

Question 3

Προσπαθώντας να κάνω αποδοτικά με dictionary την αναζήτηση σε κόμβους που ήδη έχουν ανοιχτεί χρειάστηκε να χωρίσω την A* ανάλογα. Ίδια λογική ακολουθείται και στα 2 if ανάλογα με την τιμή του var απλά ήταν λίγο διαφοροποιημένος ο κώδικας για να δουλέψει έτσι όπως το ήθελα.

- Αρχικοποίηση με βάση την αρχική κατάσταση.
- Αποθήκευση σε Priority Queue την αρχικής κατάστασης.
- Αποθήκευση με κλειδί το (x,y) του κόμβου.
- Βγάζω από το frontier τον κόμβο με το μικρότερο f(n) και τον επεκτείνω.
- Κάθε του απόγονος αποθηκεύεται με το f(n) του σαν προτεραιότητα εφόσον δεν έχει ήδη επεκταθεί. Αφού αν έχει ήδη επεκταθεί ξέρω ότι το μονοπάτι που τον επέκτεινε ήταν το βέλτιστο.
- Η f(n) είναι ουσιαστικά η ευρετική h(n) και η g(n).
- Όταν βγάλει κόμβο στόχου ξέρω ότι θα έχω goal state και ταυτόχρονα βέλτιστη λύση.

Question 4-5

Στο συγκεκριμένο ανταποκρίνεται η A* όπου var==2. Ουσιαστικά αυτό που κάνω είναι να αποθηκεύω τις καταστάσεις που έχουν γίνει expand ανάλογα και με το πως βρισκόμαστε σε σχέση με την κατάσταση στόχου. Δηλαδή ο κομβος (4,5),[Univisited,Univisited,Univisited,Univisited] είναι διαφορετικός από τον κόμβο (4,5),[Univisited, Univisited, Univisited ,Visited] γιατί παρόλο που έχουν ίδια συντεταγμένη δεν έχουν ίδια κατάσταση υλοποιήσει. Η διαδικασία A* βασίζεται σε αυτή τη λογική.

Επίσης στο dictionary αποθηκεύω για κάθε κόμβο ποιες καταστάσεις έχουν υπάρξει με λίστα Π.χ. (4,5) - [[Univisited,Univisited , Univisited,Visited],[Univisited,Univisited,Univisited,Univisited]] για να κάνω τους ελέγχους.

Η ευρετική που έχω ακολουθήσει βασίζεται στο Manhattan Distance. Ουσιαστικά για αρχή βρίσκω τη κοντινότερη άκρη με βάση αυτό και από εκεί με loop βρίσκω απο ποια γωνία να πάω σε ποια. Η ευρετική επειδή τα corners είναι σταθερά και στις άκρες και επειδή χρησιμοποιεί Manhattan Distance καταλαβαίνω ότι είναι και συνεπής και ευρετική αφού ισχύουν και οι 2 συνθήκες , δηλαδή η ανισότητα για το βήμα και ότι δεν υπερεκτιμά.

Το goalstate είναι όταν έχω [“Visited’,’Visited’, “Visited’,’Visited].

Question 6

Η τακτική που ακολούθησα εδώ πέρα είναι να χρησιμοποιήσω σαν ευρετική την πραγματική απόσταση μεταξύ του κοντινότερου κόμβου απο το τωρινό position και την απόσταση μεταξύ των 2 πιο μακρινών κόμβων(min-max στις αντίστοιχες λίστες στο κώδικα). Καταλαβαίνω ότι ανα πάσα στιγμή η πραγματική απόσταση θα είναι τουλάχιστον η απόσταση από τον ένα κόμβο στον άλλον + την απόσταση των 2 ακραίων. Άρα συνεπής και ευρετική.

Question 7

Προφανώς η greedy στρατηγική μπορεί να γίνει με ένα bfs ή ucs και goal να αποτελεί η κατάσταση εφόσον βρίσκεται στο foodlist.asList().

.Ο autograder μου βγάζει 26/25. Για τυχόν απορίες/διευκρινήσεις σχετικά με τον κώδικα είμαι στη διάθεση των εξεταστών.