```
##############################################
##    Λεπτομέρειες Υλοποίησης Project 1:     ##
##         Κανελλόπουλος Στέφανος            ##
##              sdi1200050                    ##
##############################################


*****************************************************************
*  Question 1: Finding a Fixed Food Dot using Depth First Search  *
*****************************************************************

Αρχικά , χρησιμοποίησα την δοσμένη απο το φροντιστήριο κλαση Node ( χωρις
καμία αλλαγή στον κωδικά της ).

Με βάση το τυφλό αλγόριθμο αναζήτησης DFS που διδαχτήκαμε στο μάθημα
κατασκεύασα το σώμα της συνάρτησης
depthFirstSearch.
Πιο συγκεκριμένα:
 - Φτιάχνω ενα κομβο node που αρχικοποιείται με την αρχική κατάσταση του
προβλήματος (εδώ το startingPosition).
 - Ορίζω το σύνορο μου το οποίο για την υλοποίηση του DFS θα'ναι μια
Stack.
 - Βάζω στο σύνορο μου τον αρχικό κόμβο.
 - Δημιουργώ μια λίστα όπου θα κρατώ του εξερευνημένους κόμβους.
 - Και για όσο το σύνορο μου δεν ειναι άδειο :
    - Κάνω pop απο το σύνορο ( το pop θα αλλάζει με βάση τη δομή που
χρησιμοποίησα ) εδώ έχω μια δομή LIFO.
    - Αν έχω φτάσει σε μια κατάσταση στόχου τότε επιστρέφω ( μέσω της
δοσμένης path() ) μια λίστα με actions.
      απο τον αρχική θέση μέχρι τη θέση του goal.
    - Ελέγχω έπειτα αν το state του δεδομένου κόμβου βρίσκεται στο
explored , κι αν δεν ειναι τότε το βάζω.
    - Βρίσκω όλους τους διαδόχους του δεδομένου κόμβου και τους βάζω στο
σύνορο.

************************************************
*  Question 2 (3 points): Breadth First Search  *
************************************************

Ίδιος ακριβώς είναι κι εδώ ο αλγόριθμος αναζήτησης της BFS με πριν , με
τη μόνη διαφορά , αντί για Stack (LIFO)
να χρησιμοποιώ μια Queue ( FIFO ) . Έτσι διαφοροποιείται πρακτικά η pop
συνάρτηση του συνόρου και μου πετάει
άλλους κόμβους σε σχέση με πριν.

*****************************************************
*  Question 3 (3 points): Varying the Cost Function  *
*****************************************************

Όμοια κι εδώ ο αλγόριθμος αναζήτησης της UCS με πριν , με τη μόνη διαφορά
, αντί για Stack (LIFO) , ή αντί για Queue
( FIFO ), να χρησιμοποιώ μια PriorityQueue . Έτσι διαφοροποιείται
πρακτικά η pop συνάρτηση του συνόρου και μου
πετάει άλλους κόμβους σε σχέση με πριν (πιο συγκεκριμένα αυτούς με το
μεγαλύτερο priority - δηλαδή τους πιο κοντινούς
κόμβους).

************************************
*  Question 4 (3 points): A* search  *
************************************
```

Ίδια ακριβώς λογική με τα προηγούμενα , μονο που εδώ πιο συγκεκριμένα χρειαζόμαστε την PriorityQueue και τα στοιχεία
που θα της κάνουμε push θα παίρνουν κατά το push επιπρόσθετα στο κόστος τους, μια τιμή που θα επιστρέφει η εκάστοτε
ευριστική συνάρτηση (f(n) = g(n) + h(n)).

****
* Question 5 (3 points): Finding All the Corners *
****

Στο CornersProblem έχω κάνει τα εξής:
- Αρχικά το έβαλα στην __init__ μια κενή λίστα στην οποία θέλω να κρατά κάθε κόμβος ποιές γωνίες έχει επισκευθέι.
- Η getStartState μου επιστρέφει ένα tuple της αρχικής θέσης,κόμβων που έχει επισκευθεί (κενή λίστα κατά την
  αρχικοποίηση δηλαδή) .
- Η συνάρτηση isGoalState επιστρέφει True όταν "len(state[1]) == 4" που αυτό σημαίνει ότι ο συγκεκριμένος κόμβος
  θα έχει επισκευθεί και τις 4 γωνίες. Σε οποιαδήποτε άλλη περίπτωση θα επιστρέφει False.
- Και τέλος, στη getSuccessors ελέγχω για κάθε action (SOUTH,EAST,WEST,NORTH) παίρνοντας τις κατάλληλες μετατοπίσεις
  dx,dy και βάζοντας τες στο (x,y) του δεδομένου κόμβου , αν πέφτει πάνω σε τοίχο. Αν αυτό δε συμβαίνει ελέγχω αν το
  παιδί αποτελεί γώνια και αν δεν ειναι στη λίστα των γωνιών που έχει επισκεφθεί ο γονέας , το βάζω.
  Η getSuccessors, μου επιστρέφει μια λίστα με triples ((x,y),visited_corner) , action , cost=1)

****
* Question 6 (3 points): Corners Problem: Heuristic *
****

Η βασική λογική που χρησιμοποίησα εδώ είναι η εξής:
- Όταν φτάσω σε κατάσταση στόχου (δηλαδή μπει κομβός ο οποιός να έχει επισκεφθεί και τις 4 γωνίες) επιστρέφω 0.
- Για το πλήθος των κόμβων που δεν έχω επισκευθεί (τις πρώτες φορές δλδ "4-len(state[1]) = 4"):
   - Θέτω ως την πιο κοντινή απόσταση τον MAXINT
   - Και έπειτα για κάθε γωνία που ΔΕΝ έχω επισκευθεί , υπολογίζω την αποστασή προς αυτή και διαλέγω τη μικρότερη
   - Θέτω αυτή τη γωνία να'ναι η επόμενη θέση μου,την βάζω στις γωνίες που έχω επισκευθεί και προσθέτω στο συνολικό
     άθροισμα των αποστάσεων , την απόσταση προς αυτήν.

- Στο τέλος επιστρέφω το άθροισμα της διαδρομής.

****
* Question 7 (4 points): Eating All The Dots *
****

Στη συνάρτηση foodHeuristic έχω κάνει τα εξής:
- Ελέγχω αν το foodCount είναι μηδέν και τότε πρακτικά είμαι σε goal state οπότε η ευριστική μου επιστρέφει 0.
- Στη περίπτωση που εμείνε ένα φαγητάκι,απλώς επιστρέφω την απόσταση από τη θέση μου σ'σαυτό.
- Για το πλήθος των φαγητών:
   - Αντιγράφω τη λίστα των εναπομείναντων φαγητών.

- Θεωρώ το πρώτο φαγητό της λίστας (κάθε φορά) το πιο κοντινό απο τη θέση που είμαι " closest = lst[i] ".
- Υπολογίζω την απόσταση απο αυτό (ευκλείδεια) και τη βάζω στη λίστα με τα αθροίσματα S " S[i] = S[i] + closestDist ** 0.5 "
- Βγάζω απο τη λίστα αυτό το φαγητό.
  -Οσο η λίστα με τα φαγητά δεν είναι άδεια:
    - Παίρνω το "νέο" πρώτο φαγητό της λίστας ως το πιο κοντινό .
    - Ελέγχω για όλα τα υπόλοιπα φαγητά τις αποστάσεις αν υπάρχει κάποιο πιο κοντινό
    - Το πιο κοντινό τελικά στο προηγούμενο το βγάζω απο τη λίστα.
    - Και προσθέτω την κοντινότερη απόσταση στο άθροισμα για αυτό το φαγητάκι.

(Με λίγα λόγια ελέγχω για όλα τα φαγήτα τα αθροίσματα των κοντινότερων αποστάσεων απο αυτά και τα κρατώ στη λίστα S)

- Και επιστρέφω το μικρότερο από αυτά τα αθροίσματα.

```
*********************************************
*  Question 8 (3 points): Suboptimal Search  *
*********************************************
```

- Στο τελικό ερώτημα αυτό που χρειάστηκε ουσιαστικά ήταν να συμπληρώσω το σώμα της isGoalState , όπου σαν goal θεωρώ
  το να βρισκόμαι σε φαγητό ("if state in self.food.asList()").
  Οπότε το μόνο που χρειάζεται στο τέλος να προσθέσω στο σώμα της findPathToClosestDot() είναι η κλήση του A* ,
  όπως αυτόν τον υλοποίησα στο αρχείο search.py , δοσμένου ενός προβλήματος problem.

```
##########################################################################
##########################################################################
```

Τέλος Pdf