PACMAN PROJECT 1
ΔΗΜΗΤΡΙΟΣ ΓΕΩΡΓΑΝΤΟΠΟΥΛΟΣ - SDI1900036

QUESTION 1:
Θέλουμε να υλοποιήσουμε αναζήτηση κατά βάθος, έτσι ώστε ο χαρακτήρας πακμαν να πάει από την αρχική του θέση, στο μοναδικό goal state. Όπως είναι γνωστό, στο DFS χρησιμοποιείται στοίβα, η οποία βρίσκεται ήδη υλοποιημένη στο αρχείο util.
Στη στοίβα αποθηκεύουμε τις πληροφορίες σε μορφή (start state,[]), η οποία αρχικά κενή λίστα, θα χρησιμοποιηθεί για την απομνημόνευση των κόμβων που θα έχουμε επισκεφθεί, μέχρις ώτου φτάσουμε στο goal state(το path δηλαδη).
Όσο εξερευνούμε το γράφο, θα κάνουμε push στη στοίβα τις διάφορες νέες θέσεις που θα επισκεπτόμαστε, με τον επόμενο κόμβο να τον μετατρέπουμε σε tuple ([action]) για να τον προσθέσουμε στη λίστα με τους κόμβους που έχουμε επισκεφθεί, γιατί αλλιώς θα ήταν unhashable.
Όταν φτάσουμε στο goal, επιστρέφουμε τη λίστα από τις επισκεφθείσες θέσεις.
Σε περίπτωση που ο αρχικός μας κόμβος ήταν το goal, η στοίβα ειναι κενή, το while δεν εκτελείται καν, και επιστρέφεται μια κενή λίστα, εφόσον δεν υπήρξε path για να βρεθούμε στο goal.

QUESTION 2:
Η αναζήτηση κατά πλάτος ακολουθεί πανομοιότυπη λογική με τη DFS, αλλά αντιθέτως υλοποιείται με χρήση ουράς. Η εισαγωγή δεδομένων στην ουρά ακολουθεί την ίδια λογική και format με το προηγούμενο ερώτημα.

QUESTION 3:
Το κόστος κάθε κίνησης υπολογίζεται από την heuristic function όπου κάθε φορά θα δίνουμε. Καθώς θέλουμε πλέον να αποθηκεύουμε το συνολικό κόστος από την heuristic, ο τρόπος αποθήκευσης των πληροφοριών στη ουρά ακολουθεί τη λογική ((start,[]),σύνολο heuristic μέχρι τώρα). Αυτό το σύνολο, είναι το σύνολο κόστους των ήδη επισκεφθέντων χώρων, συν του νέου που επέστρεψε η heuristic.

QUESTION 4:
getStartState: Το state εκτός των αρχικών συντεταγμένων, θα εμπεριέχει ενα αρχικά κενό tuple στο οποίο θα αποθηκεύουμε ποιες γωνίες θα έχουμε επισκεφθεί (οχι list για να μην προκύψει το unhashable error)

isGoalState: Βρίσκουμε το μέγεθος του tuple. Αν αυτό είναι 4, όλεςε οι γωνίες προφανώς έχουν επισκεφθεί, και γυρνάμε True. Σε αντίθετη περίπτωση, γυρνάμε False.

expand: Για την προσθήκη των διαθέσιμων παιδιών στη λίστα children, απαιτείται ο υπολογισμός των συντεταγμένων του, του κόστους μετακίνησης σε αυτό, αλλά και το “action” για να φτάσουμε σε αυτό. Το καθ’ένα από αυτά εύκολα μας είναι διαθέσιμο από αντίστοιχες συναρτήσεις, που είναι ήδη υλοποιημένες.

getNextState: Δημιουργώ νέα λίστα, στην οποια καταχωρούνται τα περιεχόμενα του set των ηδη επισκεφθέντων γωνιών. Η λίστα αυτή δημιουργείται εφόσον εκ φύσεώς του, τα set είναι immutable. Προστίθενται οι συντεταγμένες nextx,nexty αν αποτελούν γωνιές. Κάνω return αυτές, και την λίστα που δημιουργήσαμε ως tuple (καθώς είπαμε τα states έχουν μορφή ((x,y),set()).

(σημείωση: το init παραμένει απαράλλαχτο, εφόσον σε κάθε εκτέλεση της getNextState δημιουργώ μια νέα λίστα στην οποία καταχωρώ τις επισκεφθείσες γωνίες.)

QUESTION 5:

Ευρετική συνάρτηση η οποία, για κάθε corner το οποίο δεν έχουμε ήδη επισκεφθεί, υπολογίζω την απόσταση manhattan του από τις τωρινές μας συντεταγμένες, τοποθετώντας το αποτέλεσμα σε αντίστοιχη λίστα. Με ενα decreasing order
sort-άρισμα της λίστας, επιστρέφω πάντοτε το corner με τη μεγαλύτερη απόσταση.(μεγαλύτερη απόσταση διαλέγεται αντί της μικρότερης εφόσον η μεγαλύτερη είναι η πιο κοντινή στη πραγματική απόσταση εξαιτίας του manhattan distance), Η συνάρτηση αρχικά είναι παραδεκτή, εφόσον ποτέ της δεν υπερεκτιμά το κόστος της άφιξης του χαρακτήρα μας στην επιλεχθείσα γωνία (προφανώς λόγω manhattan με τις ακριβείς συντεταγμένες και decreasing order sorting). Ταυτόχρονα, είναι και συνεπής, εφόσον ισχύει η σχέση h(n) <= c(n,a,n')+h(n'), δηλαδή το κόστος μεταφοράς στο goal state από τον κόμβο n, είναι μικρότερο η ίσο του αθροίσματος του κόστους μεταφοράς στο goal state από τον διάδοχο του, n' και το κόστος μεταφοράς στο n'. Αυτό θα γίνεται πάντοτε αφού κάθε φορά επιλέγουμε το μεγαλύτερο δυνατό από τις διαθέσιμες προσεγγίσεις.

QUESTION 6:

Παρόμοια λογική με cornersHeuristic. Αξιοποιώντας τη συνάρτηση .asList(), λαμβάνουμε τα φαγητα του grid σε μορφή λίστας (όπως προτάθηκε στις αντίστοιχες διαφάνειες φροντιστηρίου). Για καθ'ένα από αυτά υπολογίζω την manhattan απόστασή του από τις τωρινές μας συντεταγμένες. Με απλή χρήση μιας αριθμητικής μεταβλητής ανιχνεύω αυτή με το μεγαλύτερο κόστος μετακίνησης. Η σνάρτηση προφανώς είναι παραδεκτή και δεν υπερεκτιμά το κόστος του pacman να πάει κάπου αφού βρίσκει το max κόστος και αξιοποιεί manhattanDistance για την εύρεση του μεγαλύτερου δυνατού κόστους για τα φαγητά. Ακόμα, είναι συνεπής εφόσον ισχύει η ίδια σχέση με το question 5, εφόσον πάλι κάθε φορά επιλέγουμε το μεγαλύτερο δυνατό από τις προσεγγίσεις.

QUESTION 7:
isGoalState: Εφόσον έχουμε συνάρτηση που μετατρέπει σε μορφή λίστας όλα τα φαγητά του grid, απλώς ελέγχουμε αν οι τωρινές μας συντεταγμένες, συμπέφτουν με αυτές της λίστας. Αν ναι, τότε επιστρεφουμε "επιτυχής". Εφόσον στόχος μας είναι η εύρεση και καθοδήγηση στο κοντινότερο φαγητό,

δεν απαιτείται νεα υλοποίηση αλλά απλώς καλείται η bfs εφόσον επιτελεί ίδιο έργο.