Τεχνητή Νοημοσύνη - 1η Εργασία

Ονοματεπώνυμο: Στεργιανάς Αχιλλέας

ΑΕΜ: 3351

Για την μοντελοποίηση του προβλήματος δημιούργησα μια κλάση “State” η οποία αναπαριστά μια κατάσταση του πίνακα των πλακιδίων.

Για την κάθε κατάσταση αποθηκεύω το ύψος, το πλάτος, το μέγεθος του πίνακα, την διάταξη των αριθμών, την κατάσταση «γονέα», το αριθμό κινήσεων που προηγήθηκαν από την αρχική κατάσταση, την οριζόντια και την κατακόρυφη θέση του κενού πλακιδίου.

Για την δημιουργία ενός αντικειμένου της κλάσης “State” χρησιμοποιώ 3 κατασκευαστές:

* έναν χωρίς ορίσματα που δίνει την τιμή μηδέν (0) σε κάθε μεταβλητή του αντικειμένου και κενή τιμή στις μεταβλητές για τη διάταξη των πλακιδίων και για την κατάσταση «γονέα».
* έναν με ορίσματα έναν ακέραιο με το ύψος του πίνακα, έναν ακέραιο με το πλάτος του πίνακα, έναν μονοδιάστατο πίνακα ακεραίων με τη διάταξη των αριθμών των πλακιδίων
* και έναν με όρισμα μια σταθερά τύπου State που κάνει αναπαραγωγή μια ήδη υπάρχουσας κατάστασης

και έναν καταστροφέα.

Στην κλάση επίσης υπάρχουν οι μέθοδοι:

* void setG(int), που θέτει το πλήθος κινήσεων που προηγήθηκαν από την αρχική κατάσταση σε αυτή που δίνεται από το όρισμα
* int getG(), που επιστρέφει το πλήθος κινήσεων που προηγήθηκαν από την αρχική κατάσταση
* string toString(), που επιστρέφει την διάταξη των αριθμών σε μορφή κειμένου
* bool isSolvable(), που επιστρέφει τιμή true αν η διάταξη των αριθμών είναι επιλύσιμη και τιμή false αν δεν είναι
* bool isFinished(), που επιστρέφει τιμή true αν η διάταξη των αριθμών είναι η τελική διάταξη
* int manhattan(), που επιστρέφει το άθροισμα της απόστασης του κάθε πλακιδίου από την επιθυμητή του θέση
* vector<State> expand(), που επιστρέφει ένα vector τύπου State με τις καταστάσεις «παιδιά»
* void moveBlank(int,int), που μετακινεί το κενό πλακίδιο στη θέση που ορίζετε από τα 2 ορίσματα
* void moveX(int), που καλεί την μετακίνηση του κενού πλακιδίου 1 θέση κατακόρυφα
* void moveY(int), που καλεί την μετακίνηση του κενού πλακιδίου 1 θέση οριζόντια
* bool moveUp(State&), που επιστρέφει true αν μπορεί να μετακινηθεί το κενό πλακίδιο προς τα πάνω και δημιουργεί την αντίστοιχη κατάσταση «παιδί» η οποία είναι καλεσμένη με αναφορά αλλιώς επιστρέφει την τιμή false
* bool moveDown(State&), που επιστρέφει true αν μπορεί να μετακινηθεί το κενό πλακίδιο προς τα κάτω και δημιουργεί την αντίστοιχη κατάσταση «παιδί» η οποία είναι καλεσμένη με αναφορά αλλιώς επιστρέφει την τιμή false
* bool moveLeft(State&), που επιστρέφει true αν μπορεί να μετακινηθεί το κενό πλακίδιο προς τα αριστερά και δημιουργεί την αντίστοιχη κατάσταση «παιδί» η οποία είναι καλεσμένη με αναφορά αλλιώς επιστρέφει την τιμή false
* bool moveRight(State&), που επιστρέφει true αν μπορεί να μετακινηθεί το κενό πλακίδιο προς τα δεξιά και δημιουργεί την αντίστοιχη κατάσταση «παιδί» η οποία είναι καλεσμένη με αναφορά αλλιώς επιστρέφει την τιμή false
* vector<string> getPath(), που επιστρέφει ένα vector τύπου string με όλες τις καταστάσεις από την αρχική μέχρι την τελική

Στο αρχείο main.cpp υπάρχουν 4 μέθοδοι που εκτελούν τους 4 αλγόριθμους της εργασίας. Και οι 4 είναι τύπου int και επιστρέφουν τον αριθμό των καταστάσεων με τις οποίες αλληλεπίδρασαν. Επίσης και οι 4 έχουν από 3 ορίσματα με τα 2 πρώτα να είναι η τελική κατάσταση με αναφορά και ένα set καταστάσεων με τις καταστάσεις που έχουν ελεγχθεί. Το 3ο όρισμα αφορά την ανοιχτή λίστα καταστάσεων που μένουν να ελεγχθούν και ο τύπος του διαφέρει ανάλογα με τον αλγόριθμο:

* για τον αλγόριθμο dfs χρησιμοποιώ τη δομή stack τύπου State διότι είναι της λογικής LIFO (last in first out) και εξυπηρετεί στον συγκεκριμένο αλγόριθμο που οι καταστάσεις «παιδιά» μπαίνουν στην αρχή της ανοιχτής λίστας
* για τον αλγόριθμο bfs χρησιμοποιώ τη δομή queue τύπου State διότι είναι της λογικής FIFO (first in first out) και εξυπηρετεί στον συγκεκριμένο αλγόριθμο που οι καταστάσεις «παιδιά» μπαίνουν στο τέλος της ανοιχτής λίστας
* για τους αλγορίθμους best first και A\* χρησιμοποιώ τη δομή priority\_queue τύπου State που αποθηκεύει τα δεδομένα σε ένα vector τύπου State με κριτήριο ταξινόμησης τα struct compareStates1 και compareStates2 αντίστοιχα που υλοποίσα στο ίδιο αρχείο
  + στο compareStates1 το κριτήριο είναι ο ελάσσων αριθμός Manhattan
  + στο compareStates2 το κριτήριο είναι το ελάσσων άθροισμα του αριθμού Manhattan με το πλήθος κινήσεων που προηγήθηκαν από την αρχική κατάσταση

Στην μέθοδο main το πρόγραμμα ρωτάει τον χρήστη αν θέλει να χρησιμοποιήσει την προκαθορισμένη από την εκφώνηση της εργασίας αρχική κατάσταση ή αν θέλει να εισάγει μια δική του.

Αν επιλέξει το 2ο του ζητάει το ύψος του πίνακα, το πλάτος του πίνακα και την σειρά των αριθμών που θέλει. Στη συνέχεια το πρόγραμμα ελέγχει αν η δοθείσα αρχική κατάσταση είναι επιλύσιμη και αν όχι ζητάει από τον χρήστη να δώση μία διαφορετική.

Η τελική κατάσταση παράγεται αυτόματα και είναι η αύξουσα σειρά των αριθμών με το κενό πλακίδιο στο τέλος.

Ο χρήστης επιλέγει με ποιον αλγόριθμο θέλει να βρεθεί η λύση και το πρόγραμμα του εμφανίζει με την σειρά τις καταστάσεις που πρέπει να ακολουθήσει για να φτάσει από την αρχική κατάσταση στην τελική καθώς και σε πόσα βήματα και με πόσες αλληλεπιδράσεις μεταξύ καταστάσεων.

Τέλος ρωτάει τον χρήστη αν θέλει να δοκιμάσει έναν διαφορετικό αλγόριθμο για την ίδια αρχική κατάσταση, να αλλάξει αρχική κατάσταση ή να τερματίσει το πρόγραμμα.

**ΔΟΚΙΜΕΣ**

Για την αρχική κατάσταση της εκφώνησης της εργασίας οι 4 αλγόριθμοι εκτελέστηκαν και έδωσαν τα εξής αποτελέσματα:

* **dfs:** 59760 βήματα, 97932 αλληλεπιδράσεις
* **bfs:** 10 βήματα, 721 αλληλεπιδράσεις
* **best first:** 10 βήματα, 19 αλληλεπιδράσεις
* **A\*:** 10 βήματα, 19 αλληλεπιδράσεις