ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
Τμήμα Πληροφορικής

ΕΠΛ 232 – Προγραμματιστικές Τεχνικές & Εργαλεία

Άσκηση 3 – Αυτόματη Επίλυση του Παιχνιδιού N-Puzzle

Διδάσκων: Δημήτρης Ζεϊναλιπούρ
Υπεύθυνοι Εργαστηρίων: Παύλος Αντωνίου και Πύρρος Μπράτσκας

Ημερομηνία Ανάθεσης: Παρασκευή, 12 Οκτωβρίου 2018
Ημερομηνία Παράδοσης: Δευτέρα, 29 Οκτωβρίου 2018, Ώρα 12:00 μεσημέρι (17 ημέρες)

https://www.cs.ucy.ac.cy/courses/EPL232

I. Στόχοι

Στην εργασία αυτή θα ασχοληθούμε με **δυναμικές δομές δεδομένων** σε συνδυασμό με αναδρομικές συναρτήσεις. Συγκεκριμένα, καλείστε να υλοποιήσετε ένα σύνολο τυφλών και ευρετικών αλγορίθμων αναζήτησης (blind and heuristic search algorithms) για να επιλύσετε το γνωστό παιχνίδι n-Puzzle (γνωστό και ως Sliding-Block ή Tile-Puzzle). Για την υλοποίηση της άσκησης θα χρειαστεί να χρησιμοποιήσετε τα ακόλουθα στοιχεία:

1. Διάσπαση του προγράμματος σε πολλαπλά αρχεία .c και .h με χρήση generic **makefile** και ένα βασικό αρχείο as3-npuzzle.c, μαζί με τα σχετικά αρχεία κεφαλίδας.
2. Το πρόγραμμα πρέπει να μεταγλωττίζεται τόσο μέσω του **IDE** όσο και μέσω της γραμμής εντολής με το Makefile.
3. **Κάθε αντικείμενο (module)** πρέπει να συμπεριλαμβάνει τον **σχετικό οδηγό χρήσης** (**driver functions**, δείτε διάλεξη 12).
4. **Σχόλια** και οδηγό σχολίων με χρήση του **doxygen** αλλά και διάγραμμα εξαρτήσεων αντικειμένων με χρήση του **`graphviz`** διαθέσιμο στις μηχανές των εργαστηρίων.

Θα ξεκινήσουμε με την περιγραφή του αλγόριθμου και στη συνέχεια θα δώσουμε τα ζητούμενα.

II. Περιγραφή παιχνιδιού N-Puzzle

Έστω το παιχνίδι 8-παζλ, όπου στόχος είναι να βρούμε μια ακολουθία κινήσεων έτσι ώστε από μια δοθείσα αρχική διάταξη των πλακιδίων στο ταμπλό του παιχνιδιού να καταλήξουμε σε μία προκαθορισμένη τελική διάταξη. Στο παρακάτω σχήμα φαίνεται ένα τέτοιο πρόβλημα.

4	1	3
2		6
7	5	8

Αρχική διάταξη

1	2	3
4	5	6
7	8	

Τελική διάταξη

Σημειώστε ότι στην τελική διάταξη το κενό βρίσκεται **πάντα** στην κάτω δεξιά γωνία του ταμπλό (πίνακα) του παιχνιδιού. Η λύση του προβλήματος μπορεί να περιγραφεί ως μια ακολουθία μετακινήσεων της κενής θέσης (δηλ. αντικατάστασης της με ένα από τα γειτονικά της πλακίδια). Ειδικότερα, η λύση του παραπάνω προβλήματος είναι η εξής:

αριστερά, επάνω, δεξιά, κάτω, κάτω, δεξιά

που δηλώνει ότι η κενή θέση μετακινήθηκε έξι φορές, πρώτα προς τα αριστερά, μετά προς τα πάνω κλπ. Σημειώστε ότι η μετακίνηση της κενής θέσης ουσιαστικά προκύπτει από τη μετακίνηση κάποιου πλακιδίου προς την αντίθετη κατεύθυνση. Για παράδειγμα, όταν λέμε ότι η κενή θέση μετακινείται προς τα αριστερά αυτό που πραγματικά συμβαίνει είναι ότι το πλακίδιο που βρισκόταν αριστερά της κενής θέσης μετακινήθηκε προς τα δεξιά. Παρόμοια με το πρόβλημα 8-παζλ, μπορεί να οριστεί το πρόβλημα 15-παζλ, 24-παζλ κλπ (γενικότερα (N^2-1) πάζλ).

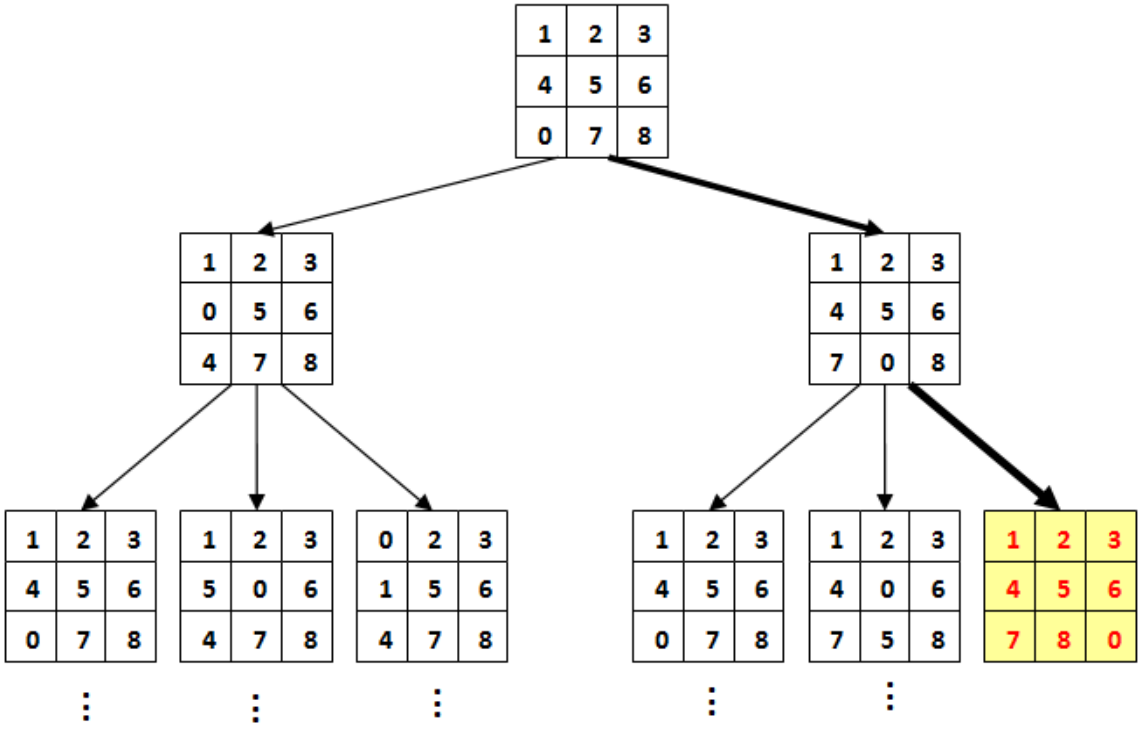
Ζητούμενο αυτής της άσκησης είναι η κατασκευή ενός προγράμματος στη γλώσσα προγραμματισμού C που θα λύνει προβλήματα (N^2-1) πάζλ, **για τα διάφορα δυνατά N όπως αυτά προσδιορίζονται από το πρόβλημα εισόδου,** χρησιμοποιώντας κάποιους δοσμένους αλγορίθμους αναζήτησης οι οποίοι θα περιγραφούν πιο κάτω.

Θεωρήστε το Διάγραμμα 1, το οποίο θα ονομάσουμε Δένδρο Παιχνιδιού (Game Tree). Σε αυτό το δένδρο, κάθε κόμβος αναπαριστά μία κατάσταση του παιχνιδιού και παιδιά ενός κόμβου είναι οι καταστάσεις στις οποίες μπορούμε να μεταβούμε από την κατάσταση-πατέρα εκτελώντας μία μετακίνηση του κενού τετραγώνου. Πιο συγκεκριμένα, η δομή ενός δένδρου παιχνιδιού ορίζεται ως εξής:

1. Η ρίζα του δένδρου περιέχει την αρχική κατάσταση του πάζλ.
2. Κάθε κόμβος του δένδρου αναπαριστά μία κατάσταση του πάζλ.

Τα παιδιά κάθε κόμβου είναι οι καταστάσεις στις οποίες μπορούμε να μεταβούμε από τον κόμβο εκτελώντας μία μετακίνηση του κενού τετραγώνου, *δεδομένου ότι δεν έχουμε ήδη επισκεφθεί τη συγκεκριμένη κατάσταση.* Κάθε κόμβος-πατέρας μπορεί να έχει τόσα παιδιά όσες είναι όλες οι πιθανές κινήσεις του κενού τετραγώνου από την κατάσταση του κόμβου-πατέρα με μέγιστο 4 (πάνω, κάτω, δεξιά, αριστερά)

Στο πιο κάτω σχήμα δείχνεται μέρος του δένδρου παιχνιδιού για κάποιο απλό 8-πάζλ με αρχική κατάσταση την ρίζα του δένδρου και τελική κατάσταση το δεξιότερο κόμβο του δένδρου:



Διάγραμμα 1: Ενδεχόμενα Μονοπάτια Επίλυσης του Παιχνιδιού N-Puzzle (το μονοπάτι από την ρίζα προς το δεξιότερο-κάτω κόμβο τυγχάνει να είναι μια επίλυση του πιο πάνω παιχνιδιού)

III. Ζητούμενα Άσκησης

Γράψτε ένα πρόγραμμα το οποίο θα υλοποιεί τις ακόλουθες λειτουργίες όπως περιγράφονται πιο κάτω:

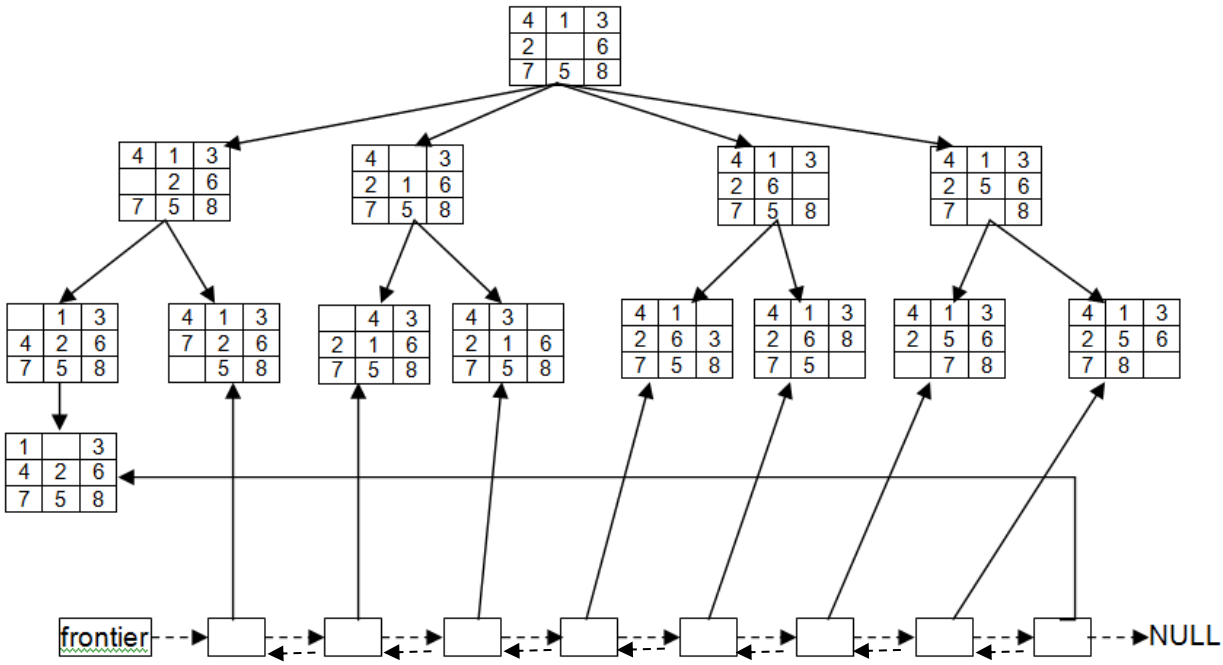
- Θέμα 1: Αλγόριθμοι απληροφόρητης (ή τυφλής) αναζήτησης
- Θέμα 2: Αλγόριθμοι πληροφορημένης (ή ευρετικής) αναζήτησης

Θέμα 1ο : Αλγόριθμοι απληροφόρητης (ή τυφλής) αναζήτησης

Γράψτε ένα πρόγραμμα που να λύνει προβλήματα (N^2-1)-πάζλ χρησιμοποιώντας τον αλγόριθμο ***Αναζήτηση κατά Πλάτος (breadth-first search),*** *στον οποίο η βασική ιδέα είναι να εισάγουμε τις εναλλακτικές κινήσεις στο τέλος μιας ουράς (η οποία για λόγους υλοποίησης που θα περιγραφούν στο τέλος θα είναι διπλά συνδεδεμένη), ενώ η επόμενη διερευνώμενη κίνηση θα εξάγεται από την κεφαλή της ουράς, δηλ.,:*

A. Τα παιδιά ενός κόμβου (το πολύ μέχρι τέσσερα για το συγκεκριμένο πρόβλημα) εισάγονται στο τέλος της ουράς (την οποία θα ονομάσουμε μέτωπο) με την εξής σειρά: Πρώτα αυτό που προκύπτει από τη μετακίνηση του κενού προς τα δεξιά, μετά αυτό που προκύπτει από τη μετακίνηση του κενού προς τα κάτω, μετά αυτό που προκύπτει από τη μετακίνηση του κενού προς τα αριστερά και τέλος αυτό που προκύπτει από τη μετακίνηση του κενού προς τα πάνω. Η σειρά αυτή δεν είναι υποχρεωτική, το πρόγραμμά σας μπορεί να τοποθετεί τους νέους κόμβους στο μέτωπο αναζήτησης υιοθετώντας διαφορετική μεταξύ τους σειρά.

B. Κάθε φορά που δημιουργείται ένα παιδί, γίνεται έλεγχος για βρόχο (κύκλο) σε όλους τους προγόνους του (το μονοπάτι μέχρι την ρίζα του δένδρου). Εάν το παιδί ταυτίζεται με κάποιον από τους προγόνους του (η ίδια κίνηση δηλαδή), τότε δεν εισάγεται ούτε στο δένδρο ούτε στο μέτωπο. Στο παράδειγμα, παρατηρήστε ότι κάθε ένα από τα παιδιά της ρίζας έχει δύο παιδιά, αντί για τρία. Το τρίτο παιδί που δεν εμφανίζεται θα ήταν ίδιο με τη ρίζα**. Ο έλεγχος μόνο στους προγόνους και όχι σε όλο το δένδρο είναι ένας συμβιβασμός:** Είναι δυνατόν να προκαλέσει επανάληψη κόμβων σε διαφορετικά κλαδιά, ωστόσο είναι πιο γρήγορος και πετυχαίνει το βασικό ζητούμενο που είναι η αποφυγή των ατέρμονων βρόχων που προκύπτουν από το ίδιο μονοπάτι.



Διάγραμμα 2: Αναζήτηση Κατά Πλάτος

Στο παραπάνω διάγραμμα με συνεχή γραμμή φαίνονται τα βελάκια-δείκτες προς κόμβους του δένδρου και με διακεκομμένη γραμμή φαίνονται τα βελάκια-δείκτες που σχηματίζουν το μέτωπο αναζήτησης (ουρά αναζήτησης). Σε κάθε επανάληψη εξάγεται από το μέτωπο ο πρώτος κόμβος και εισάγονται στο μέτωπο τα παιδιά αυτού. Ο επόμενος αλγόριθμος **πληροφορημένης (ή ευρετικής) αναζήτησης** διαφοροποιείται ανάλογα με το σε ποια θέση του μετώπου εισάγονται τα παιδιά του εξαγόμενου κόμβου, όπως θα εξηγήσουμε σε λίγο.

Υλοποίηση

Το πρόγραμμα θα δέχεται ως παραμέτρους τη μέθοδο επίλυσης, το όνομα του αρχείου περιγραφής του προβλήματος και το όνομα του αρχείου στο οποίο θα γραφεί η λύση. Για παράδειγμα, εάν το όνομα του προγράμματός σας είναι `puzzle`, θέλετε να χρησιμοποιήσετε αναζήτηση κατά πλάτος, το αρχείο εισόδου είναι το `input.txt` και θέλετε η λύση να γραφεί στο αρχείο `solution.txt`, θα πρέπει να καλέσετε το πρόγραμμά σας με την εντολή:

```
./puzzle breadth input.txt solution.txt
```

Σε σχέση με το αρχείο εισόδου `input.txt`, έστω ότι `N=3` (το οποίο πρέπει να βρίσκετε μόνοι σας από το ίδιο το αρχείο) και ότι θέλουμε να λύσουμε το πρόβλημα του αρχικού παραδείγματος. Τα περιεχόμενα του αρχείου εισόδου έχουν την ακόλουθη μορφή:

```
4	1	3
2	0	6
7	5	8
```

Σημειώστε ότι στην παραπάνω γραμμογράφηση οι αριθμοί της ίδιας σειράς διαχωρίζονται με tab.

Σε σχέση με το αρχείο εξόδου `solution.txt`, το περιεχόμενό του θα πρέπει να είναι της παρακάτω μορφής (ακολουθεί ως παράδειγμα η λύση του τρέχοντος προβλήματος):

```
6
left
up
right
down
down
right
```

όπου στην πρώτη γραμμή αναγράφεται το πλήθος των κινήσεων, ενώ οι λέξεις up, down, left και right προσδιορίζουν την εκάστοτε μετακίνηση της κενής θέσης.

Το πρόγραμμά σας μπορεί να τυπώνει περιορισμένης έκτασης μηνύματα στην οθόνη, όπως π.χ. το πλήθος των βημάτων που έχει η λύση, κλπ.

Δώστε ιδιαίτερη προσοχή στον τρόπο κλήσης του προγράμματός σας, καθώς και στη μορφή των αρχείων εισόδου και εξόδου, σύμφωνα με όσα περιγράφηκαν παραπάνω (συμπεριλαμβανομένων των λύσεων που αυτό παράγει), διότι το πρόγραμμα σας ενδέχεται να ελεγχτεί με αυτόματο τρόπο πάνω σε ένα μεγάλο σύνολο δεδομένων εισόδου (black-box testing).

Θέμα 2ο : Αλγόριθμοι πληροφορημένης (ή ευρετικής) αναζήτησης

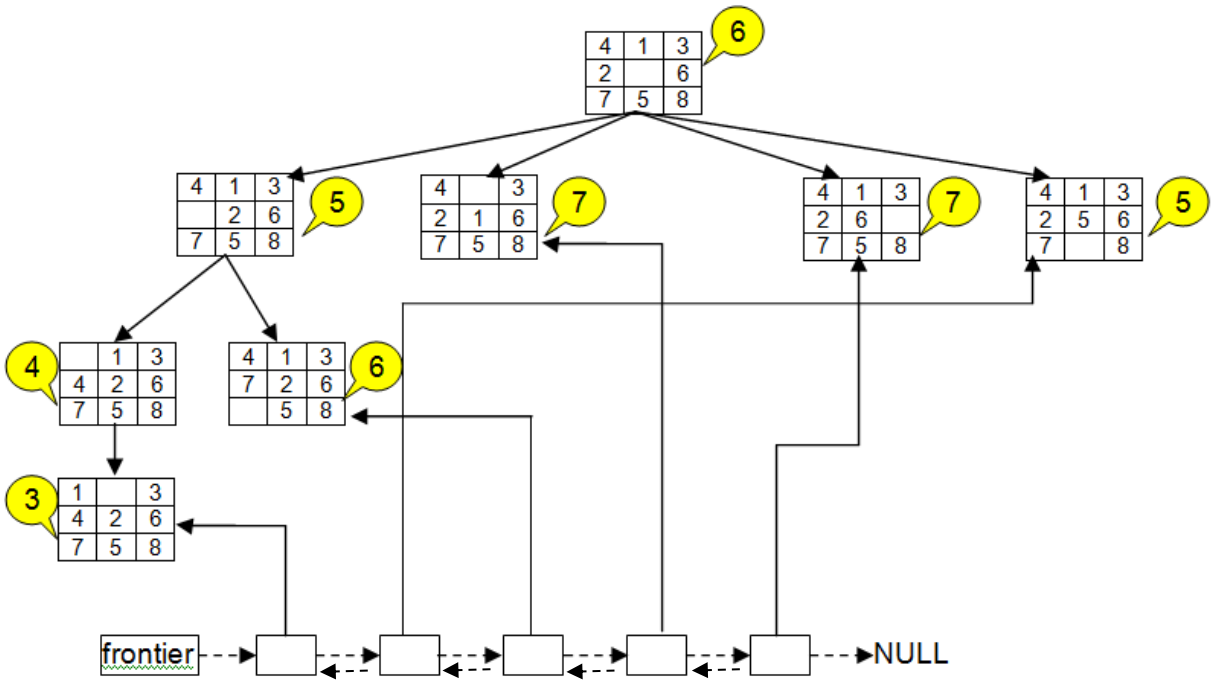
Ο αλγόριθμος αναζήτησης *κατά πλάτος* τοποθετεί τους νέους κόμβους στο τέλος του μετώπου (frontier) αναζήτησης. Υπάρχουν ωστόσο και άλλες επιλογές, οι οποίες οδηγούν σε ολόκληρη κατηγορία νέων αλγορίθμων. Η βασική εναλλακτική επιλογή είναι οι νέοι κόμβοι να μην τοποθετούνται ούτε στην αρχή, ούτε στο τέλος του μετώπου αναζήτησης, αλλά ενδιάμεσα. Ειδικότερα, κάθε νέος κόμβος βαθμολογείται

με μια ευρετική συνάρτηση, η οποία παρέχει μια εκτίμηση της απόστασης (πλήθος κινήσεων) του κόμβου από την επιθυμητή τελική κατάσταση, ενώ το μέτωπο αναζήτησης διατηρείται ταξινομημένο, με τους κόμβους που έχουν μικρότερη ευρετική τιμή πρώτους.

Μια δυνατότητα βαθμολόγησης των κόμβων (κάθε κόμβος αντιστοιχεί σε μια διάταξη των πλακιδίων του παιχνιδιού) για το συγκεκριμένο παιχνίδι είναι η εξής: *Μετράμε για κάθε πλακίδιο πόσες θέσεις απέχει από την τελική του θέση στην οριζόντια και στην κατακόρυφη διεύθυνση (απόσταση Manhattan ή αλλιώς η απόσταση μεταξύ δύο σημείων κατά μήκος αξόνων). Σε ένα επίπεδο με το σημείο p_1 στο (x_1, y_1) και p_2 στο (x_2, y_2), η απόσταση Manhattan είναι $| x_1 - x_2 | + | y_1 - y_2 |$. Στην αριστερή διάταξη του παραδείγματος της πρώτης σελίδας, το πλακίδιο με αριθμό 2 απέχει δύο θέσεις από την τελική του θέση (μία στον οριζόντιο άξονα και μία στον κατακόρυφο άξονα). Παρόμοια, το πλακίδιο με αριθμό 4 απέχει μία θέση από την τελική του θέση, ενώ τέλος το πλακίδιο 7 απέχει μηδέν θέσεις από την τελική του θέση. Αφού μετρήσουμε όλες αυτές τις αποστάσεις, τις προσθέτουμε και έχουμε μια εκτίμηση του πλήθους των κινήσεων που πρέπει να γίνουν για να πάμε από την τρέχουσα διάταξη στην τελική (στο συγκεκριμένο παράδειγμα η εκτίμηση αυτή είναι ίση με 6). Παρατηρούμε ότι η εκτίμηση που παίρνουμε με αυτό τον τρόπο είναι πάντα μικρότερη ή ίση του πραγματικού πλήθους των κινήσεων που απαιτούνται για να λύσουμε το πρόβλημα (υποεκτίμηση).*

α) Αναζήτηση πρώτα στο καλύτερο (best-first search)

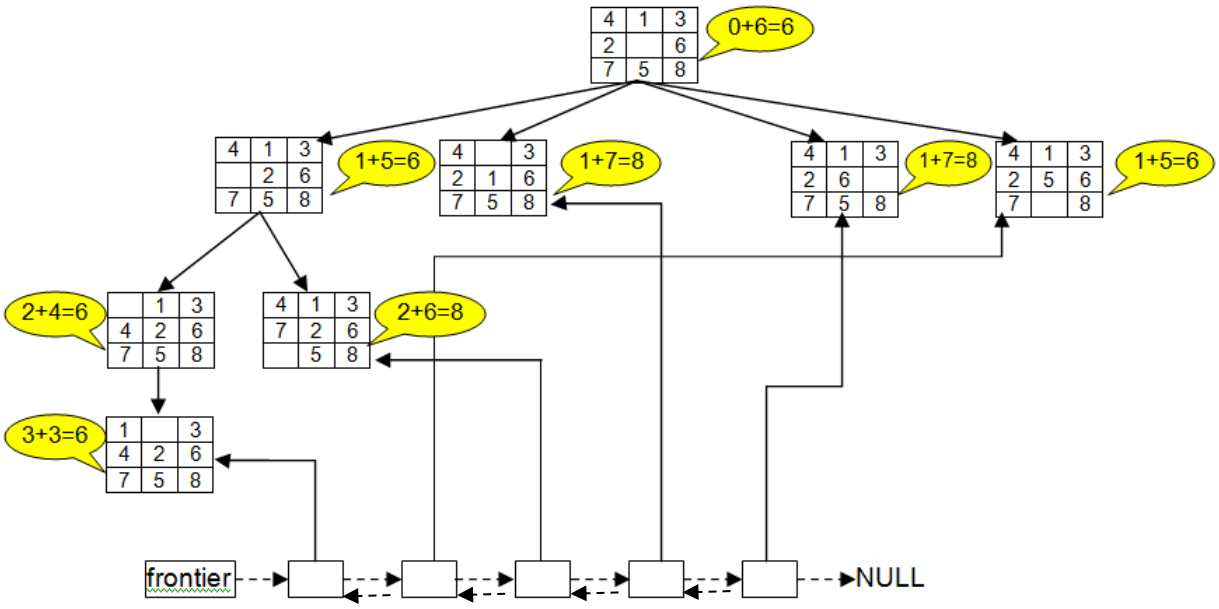
Στην αναζήτηση πρώτα στο καλύτερο κάθε κόμβος πρέπει να βαθμολογείται βάση μιας ευρετικής συνάρτησης που περιγράψαμε πιο πάνω και η οποία μετράει την απόσταση του κόμβου από την τελική επιθυμητή διάταξη (δείτε διάγραμμα 3). Οι νέοι κόμβοι εισέρχονται στο μέτωπο αναζήτησης με τέτοιον τρόπο ώστε να διατηρείται μια αύξουσα ταξινόμηση. Σε περίπτωση ισοβαθμίας μεταξύ των δύο κόμβων, προηγείται αυτός που εισήχθη νωρίτερα στο δένδρο, ενώ σε περίπτωση ισοβαθμίας δύο κόμβων που εισάγονται ταυτόχρονα, ισχύει η παραδοχή (Α) του αλγορίθμου αναζήτησης πρώτα σε πλάτος. Στο διάγραμμα που ακολουθεί φαίνεται η αρχή του δένδρου αναζήτησης του αλγορίθμου πρώτα στο καλύτερο, συμπεριλαμβανομένων και των βαθμών των κόμβων. Υπόδειγμα εκτέλεσης: `./puzzle best input.txt solution.txt`



Διάγραμμα 3: Αναζήτηση Πρώτα σε Καλύτερο

β) Αναζήτηση A* (A* search)

Ισχύουν ακριβώς όσα ισχύουν στην αναζήτηση πρώτα στο καλύτερο με μοναδική διαφορά ότι στον βαθμό κάθε κόμβου του δένδρου αναζήτησης συνυπολογίζεται και η "απόστασή" του από τη ρίζα (όπως αυτό παρουσιάζεται στο διάγραμμα 4) Δηλαδή η μοναδική διαφορά της αναζήτησης A* είναι ότι στον βαθμό κάθε κόμβου συνυπολογίζεται (προστίθεται) και το πόσες κινήσεις εκτελέσατε από την αρχική κατάσταση για να φτάσετε στον τρέχοντα κόμβο. Για παράδειγμα, εάν ένας κόμβος έχει προκύψει ύστερα από 4 μετακινήσεις του κενού (σε σχέση με την αρχική διάταξη), ενώ η ευρετική απόσταση του από την τελική διάταξη είναι 7, ο συνολικός βαθμός του κόμβου είναι 11. Στην αναζήτηση A* το μέτωπο αναζήτησης διατηρείται ταξινομημένο με βάση το παραπάνω άθροισμα.



Διάγραμμα 4: Αναζήτηση A*

Μια δεύτερη διαφορά αφορά τις ισοβαθμίες: Σε περίπτωση ισοβαθμίας μεταξύ δύο κόμβων, το βασικό κριτήριο επίλυσής της είναι ότι προηγείται στο μέτωπο αναζήτησης ο κόμβος με μεγαλύτερη απόσταση από την αρχή (ή ισοδύναμα ο κόμβος με μικρότερη ευρετική απόσταση από την επιθυμητή τελική διάταξη). Εάν υπάρχει και εδώ ισοβαθμία, τότε ισχύουν τα κριτήρια του αλγορίθμου πρώτα στο καλύτερο.

Υπόδειγμα εκτέλεσης:

```
./puzzle a-star input.txt solution.txt
```

IV. Θέματα Υλοποίησης

Για την κατασκευή του δένδρου αναζήτησης θα πρέπει να ορίσετε μια δομή για τον κόμβο του δένδρου. Η δομή αυτή πρέπει να περιέχει τουλάχιστον τα εξής πεδία:

- πίνακας ακεραίων NxN, για την αναπαράσταση της τρέχουσας κατάστασης του παιχνιδιού
- απόσταση από την αρχή (χρειάζεται για τον αλγόριθμο A*)
- ευρετική τιμή απόστασης από το τέλος (χρειάζεται για τους αλγορίθμους πρώτα στο καλύτερο και A*)
- δείκτης προς τον γονέα (χρειάζεται ώστε όταν βρεθεί ένας κόμβος που είναι λύση του προβλήματος, να μπορούμε να υπολογίσουμε την ακολουθία βημάτων που οδήγησαν στη λύση)
- μέχρι τέσσερις δείκτες προς τα παιδιά (χρειάζονται για να είναι δυνατή η ελευθέρωση της μνήμης μετά την ολοκλήρωση του αλγορίθμου)

Ένα υπόδειγμα κατασκευής της παραπάνω δομής ακολουθεί:

```
typedef struct tree_node
{
      int **puzzle;
      int g;
      int h;
      struct tree_node *parent;
      struct tree_node *children[4];
} TREE_NODE;
```

Για το μέτωπο αναζήτησης θα χρειαστείτε μια **διπλά συνδεδεμένη ουρά**, ικανή να έχει δείκτες προς κόμβους του δένδρου αναζήτησης σε κάθε κόμβο της λίστας:

```
typedef struct frontier_node
{
      TREE_NODE *leaf;
      struct frontier_node *next;
      struct frontier_node *previous;
} FRONTIER_NODE;
```

Φυσικά θα χρειαστείτε μεταβλητές/δείκτες που να δείχνουν στη ρίζα του δένδρου και στην αρχή του μετώπου αναζήτησης (εναλλακτικά συμπεριλάβετε τα σε ένα struct QUEUE):

```
TREE_NODE *search_tree;
FRONTIER_NODE *frontier_head;
FRONTIER_NODE *frontier_tail;
```

Από κει και πέρα, το πρόγραμμά σας θα πρέπει να είναι ικανό μεταξύ άλλων να εκτελέσει τις εξής λειτουργίες:

- Δοθέντος ενός κόμβου αναζήτησης, να βρίσκει όλα τα παιδιά του και να τα τοποθετεί στο δένδρο.
- Να ελέγχει εάν δύο διατάξεις των πλακιδίων ταυτίζονται (χρειάζεται στον έλεγχο για βρόχους καθώς και στον έλεγχο για εύρεση της λύσης).
- Να εισάγει κόμβους στο μέτωπο αναζήτησης, είτε στην αρχή, είτε στο τέλος, είτε σε θέση ανάλογα με το άθροισμα g+h του κόμβου (το g θα είναι πάντα μηδέν για την αναζήτηση πρώτα στο καλύτερο).

V. Κριτήρια αξιολόγησης

Θέμα 1 (Τυφλοί αλγόριθμοι αναζήτησης) Αναζήτηση πρώτα σε πλάτος (40 μονάδες)	40
Θέμα 2 (Ευρετικοί αλγόριθμοι αναζήτησης) α) Αναζήτηση πρώτα στο καλύτερο (30 μονάδες) β) Αναζήτηση A* (20 μονάδες)	50
Διάσπαση Προγράμματος σε Πολλαπλά Αρχεία, Makefile, eClipse & Γενική εικόνα (ευανάγνωστος κώδικας, σχόλια, δομή, αποδοτικότητα κλπ.)	10
ΣΥΝΟΛΟ	**100**

Παρακαλώ όπως μελετηθούν ξανά οι κανόνες υποβολής εργασιών όπως αυτοί ορίζονται στο συμβόλαιο του μαθήματος.

VI. Γενικές Οδηγίες

Το πρόγραμμα σας θα πρέπει να συμβαδίζει με το πρότυπο ISO C, να περιλαμβάνει εύστοχα και περιεκτικά σχόλια, να έχει καλή στοίχιση και το όνομα κάθε μεταβλητής, σταθεράς, ή συνάρτησης να

είναι ενδεικτικό του ρόλου της. **Να χρησιμοποιήσετε το λογισμικό τεκμηρίωσης doxygen** έτσι ώστε να μπορούμε να μετατρέψουμε τα σχόλια του προγράμματός σας σε HTML αρχεία και να τα δούμε με ένα browser. Η συστηματική αντιμετώπιση της λύσης ενός προβλήματος περιλαμβάνει στο παρόν στάδιο τη διάσπαση του προβλήματος σε μικρότερα ανεξάρτητα προβλήματα που κατά κανόνα κωδικοποιούμε σε ξεχωριστές συναρτήσεις. Για αυτό τον λόγο σας καλούμε να κάνετε χρήση συναρτήσεων και άλλων τεχνικών δομημένου προγραμματισμού που διδαχτήκατε στο ΕΠΛ131. Επίσης, σας θυμίσουμε ότι κατά την διάρκεια της εκτέλεσης του προγράμματος σας αυτό θα πρέπει να δίνει τα κατάλληλα μηνύματα σε περίπτωση λάθους. Τέλος το πρόγραμμα σας θα πρέπει να μεταγλωττίζεται στις μηχανές του εργαστηρίου. **Παρακαλώ όπως μελετηθούν ξανά οι κανόνες υποβολής εργασιών όπως αυτοί ορίζονται στο συμβόλαιο του μαθήματος.** Επίσης να ακολουθήσετε τα πιο κάτω βήματα όταν υποβάλετε την άσκηση σας στο Moodle:

1. Δημιουργείστε ένα κατάλογο με το όνομά σας π.χ., PyrrosBratskas/ χωρίς να αφήνετε κενά στο όνομα του καταλόγου.
2. Βάλτε μέσα στον κατάλογο αυτό όλα τα αρχεία της εργασίας (κώδικας, doxygen configuration file, README.md) που πρέπει να υποβάλετε.
3. Συμπιέστε (zip) τον κατάλογο (και όχι τα αρχεία ξεχωριστά) χρησιμοποιώντας την εντολή *zip –c PyrrosBratskas.zip PyrrosBratskas/**
4. **Βεβαιωθείτε ότι κάνατε σωστά τα τρία προηγούμενα βήματα**
5. Ανεβάστε στο Moodle το συμπιεσμένο αρχείο π.χ., PyrrosBratskas.zip

Καλή Επιτυχία!