ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ
6ο εξάμηνο

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΕΡΓΑΣΙΑ – N-puzzle

Εισαγωγικά:

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) παζλ).

Ζητούμενο αυτής της άσκησης είναι η κατασκευή ενός προγράμματος στη γλώσσα προγραμματισμού C που θα λύνει προβλήματα (N^2-1) παζλ, για τα διάφορα δυνατά N, χρησιμοποιώντας αλγορίθμους αναζήτησης σε γράφους. Στην προκειμένη περίπτωση ο γράφος θεωρείται ότι έχει ως κόμβους το σύνολο των δυνατών διατάξεων των πλακιδίων του παιχνιδιού και ακμές που συνδέουν κόμβους μεταξύ των οποίων μπορούμε να μεταβούμε με μία κίνηση πλακιδίου.

Θεωρείστε το N ως σταθερά του προγράμματος (π.χ. #define N 3)

Θέμα 1ο : Αλγόριθμοι απληροφόρητης (ή τυφλής) αναζήτησης

Κατασκευάστε ένα πρόγραμμα που να λύνει προβλήματα (N^2-1)-παζλ με τους αλγορίθμους πρώτα σε πλάτος (breadth-first search) και πρώτα σε βάθος (depth-first search). 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.exe` (μπορείτε φυσικά να το ονομάσετε όπως αλλιώς θέλετε), 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.exe  breadth  input.txt  solution.txt
```

Εάν αντίθετα θέλετε να χρησιμοποιήσετε τον αλγόριθμο πρώτα σε βάθος, χρησιμοποιείστε τη λέξη `depth` αντί της λέξης `breadth` στην παραπάνω κλήση.

Σε σχέση με το αρχείο εισόδου `input.txt`, έστω ότι N=3 και ότι θέλουμε να λύσουμε το πρόβλημα του αρχικού παραδείγματος. Τα περιεχόμενα του αρχείου εισόδου πρέπει να είναι τα εξής:

```
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 χρόνο που χρειάστηκε για να λύσει το πρόβλημα, το πλήθος των βημάτων που έχει η λύση, ενδεχόμενα μηνύματα λάθους (π.χ. αδυναμία επίλυσης του προβλήματος μέσα σε συγκεκριμένα χρονικά όρια, π.χ. 5 mins κλπ).

Δώστε ιδιαίτερη προσοχή στον τρόπο κλήσης του προγράμματός σας, καθώς και στη μορφή των αρχείων εισόδου και εξόδου, σύμφωνα με όσα περιγράφηκαν παραπάνω, ώστε το πρόγραμμά (συμπεριλαμβανομένων των λύσεων που αυτό παράγει) να μπορεί να **ελεγχθεί αυτόματα**.

Προσοχή: Για την αποφυγή ατέρμονων βρόχων (αφορά κατά κύριο λόγο την αναζήτηση πρώτα σε βάθος), το πρόγραμμά σας θα πρέπει να ελέγχει για κύκλους τουλάχιστον στο τρέχον μονοπάτι.

Θέμα 2ο : Αλγόριθμοι πληροφορημένης (ή ευρετικής) αναζήτησης

Η μοναδική διαφορά (σε επίπεδο υλοποίησης) των αλγορίθμων αναζήτησης πρώτα σε πλάτος και πρώτα σε βάθος είναι ότι ο πρώτος τοποθετεί τους νέους (γκρι) κόμβους στο τέλος του μετώπου (frontier) αναζήτησης ενώ ο δεύτερος στην αρχή. 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 (γκρι) 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 πλακιδιών 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). 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).

Υλοποιείστε λοιπόν τους παρακάτω δύο αλγορίθμους. Χρησιμοποιείστε τις λέξεις `best` και `astar` στην είσοδο για να τους δηλώσετε.

Αναζήτηση πρώτα στο καλύτερο (best-first search)

Βαθμολογείστε τους κόμβους με βάση το άθροισμα των αποστάσεων Manhattan των πλακιδίων τους από την τελική τους θέση. Εισάγετε τους νέους κόμβους στην κατάλληλη θέση του μετώπου αναζήτησης, ώστε αυτό να παραμένει ταξινομημένο. Εξάγετε από το μέτωπο αναζήτησης προς εξέταση πάντα τον πρώτο κόμβο.

Αναζήτηση A* (A* search)

Παρόμοια με την αναζήτηση πρώτα στο καλύτερο, 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.

Το πρόγραμμα `puzzle.c` υλοποιεί τα προηγούμενα 2 θέματα. Μέσα στο πρόγραμμα υπάρχουν αναλυτικά σχόλια. Ενδεικτικά αναφέρεται εδώ ότι οι τέσσερις αλγόριθμοι υλοποιούνται από την ίδια συνάρτηση (search()), η οποία ανάλογα με τον εκάστοτε αλγόριθμο εισάγει τα παιδιά στο μέτωπο αναζήτησης με διαφορετικό τρόπο.

Θέμα 3ο : Πειραματική αξιολόγηση των αλγορίθμων

Δοκιμάστε τους τέσσερις αλγορίθμους αναζήτησης που υλοποιήσατε σε διάφορα προβλήματα (συμπεριλαμβανομένων προβλημάτων διαφορετικών διαστάσεων, π.χ. για N=3 και για N=4). Συγκρίνετε τους χρόνους επίλυσης και το πλήθος των βημάτων σε κάθε λύση. Εκθέστε τα συμπεράσματά σας όσον αφορά (συγκριτικά) το χρόνο που απαιτείται από κάθε αλγόριθμο για να λύσει τα κάθε πρόβλημα και το μήκος της λύσης που αυτός βρίσκει.

Με τη γεννήτρια προβλημάτων κατασκευάστηκαν 10 τυχαία προβλήματα διάστασης 3x3 (επισυνάπτονται). Κάθε πρόβλημα επιχειρήθηκε να λυθεί και με τους τέσσερις αλγόριθμους αναζήτησης. Τέθηκε χρονικό όριο 1 λεπτό για κάθε αλγόριθμο. Στον παρακάτω πίνακα φαίνεται για κάθε πρόβλημα και για κάθε αλγόριθμο ο χρόνος που χρειάστηκε για την επίλυσή του καθώς και το μήκος της λύσης που βρέθηκε. Όπου υπάρχει παύλα σημαίνει ότι δεν κατέστη δυνατή η επίλυση του προβλήματος εντός του προκαθορισμένου χρονικού ορίου.

	breadth		depth		best		astar	
	Χρόνος (sec)	Βήματα	Χρόνος (sec)	Βήματα	Χρόνος (sec)	Βήματα	Χρόνος (sec)	Βήματα
#1	2,52	20	0,49	2232	0,02	82	0,00	20
#2	16,46	23	-	-	0,01	119	0,06	23
#3	-	-	54,59	22507	0,03	133	0,10	25
#4	0,51	17	-	-	0,02	113	0,00	17
#5	-	-	-	-	0,00	31	0,18	25
#6	-	-	-	-	0,01	98	0,07	26
#7	1,09	18	-	-	0,00	60	0,00	18
#8	17,18	23	53,41	22259	0,00	57	0,03	23
#9	3,65	20	-	-	0,00	52	0,02	20
#10	53,73	25	0,05	715	0,00	65	0,01	25

Με βάση τα παραπάνω αποτελέσματα κάνουμε τις εξής παρατηρήσεις:

- Οι πληροφορημένοι αλγόριθμοι αναζήτησης (best και astar) είναι 2 με 3 τάξεις μεγέθους γρηγορότεροι από τους απληροφόρητους.
- Οι αλγόριθμοι breadth (όταν βρίσκει λύση) και astar βρίσκουν λύσεις ίδιου μήκους. Οι δύο αυτοί αλγόριθμοι είναι βέλτιστοι, δηλαδή η πρώτη λύση που βρίσκουν είναι αποδεδειγμένα η συντομότερη.
- Ο αλγόριθμος depth βρίσκει υπερβολικά μεγάλες σε μήκος λύσεις. Αυτό οφείλεται στο ότι κατά την κατασκευή του δένδρου αναζήτησης επιλέγει ένα κλαδί και δεν το αλλάζει παρά μόνο όταν φτάσει σε αδιέξοδο. Με δεδομένο ότι το πλήθος των δυνατών διατάξεων του παιχνιδιού είναι 9!, το μήκος των κλαδιών μπορεί να γίνει ιδιαίτερα μεγάλο.

- Ο αλγόριθμος breadth φαίνεται ότι βρήκε λύσεις έως 25 βήματα. Τα προβλήματα που δεν έλυσε είχαν μήκος λύσης μεγαλύτερο ή ίσο από 25 βήματα. Φαίνεται λοιπόν ότι το χρονικό όριο που τέθηκε (1 min) του επέτρεψε να κατασκευάζει το δένδρο αναζήτησης μόνο μέχρι αυτό το βάθος.

Αντίστοιχα συμπεράσματα προκύπτουν και για προβλήματα 4x4, ωστόσο εκεί τα μήκη των λύσεων προκύπτουν ιδιαίτερα μεγάλα (της τάξης των μερικών εκατοντάδων βημάτων) με αποτέλεσμα αφενός να είναι αδύνατο για τους απληροφόρητους αλγορίθμους αναζήτησης να λύσουν οποιοδήποτε εξ αυτών, ενώ οι πληροφορημένοι αλγόριθμοι μπορεί να χρειαστούν κατά περίπτωση ιδιαίτερα πολλή ώρα.

Δίνονται:

Σας δίνονται δύο προγράμματα:

α) Το πρόγραμμα `generator.c` παράγει τυχαία αρχεία προβλημάτων συγκεκριμένης διάστασης (ορίζεται από την σταθερά N μέσα στο πρόγραμμα). Εάν το εκτελέσιμο ονομάζεται generator.exe, τότε το καλείτε με:

```
generator.exe <filaneme> <id1> <id2>
```

όπου <filename> το πρόθεμα των ονομάτων των αρχείων που θα δημιουργηθούν, και <id1> και <id2> δύο αριθμοί που δηλώνουν το εύρος των καταλήξεων των ονομάτων των αρχείων. Για παράδειγμα, εάν καλέσουμε το πρόγραμμα με:

```
generator.exe test 21 30
```

θα δημιουργηθούν 10 αρχεία προβλημάτων με ονόματα από `test21.txt` έως `test30.txt`.

Προσοχή: Δεν έχουν λύση όλα τα προβλήματα (N^2-1)-παζλ. Για παράδειγμα, το παρακάτω πρόβλημα δεν έχει λύση:

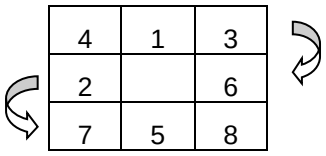
2	1	3
4		6
7	5	8

Αρχική διάταξη

1	2	3
4	5	6
7	8	

Τελική διάταξη

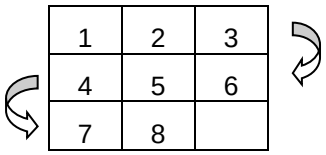
Ένας τρόπος για να ελέγξετε εύκολα εάν ένα πρόβλημα έχει λύση είναι ο εξής: Θεωρείστε μια γραμμική αναπαράσταση της διάταξης των πλακιδίων πάνω στο παιχνίδι, η οποία κατασκευάζεται βάζοντας τις γραμμές του ταμπλό του παιχνιδιού τη μία μετά την άλλη, διατρέχοντάς τες όμως εναλλάξ προς αντίθετη κατεύθυνση, όπως φαίνεται στο παρακάτω σχήμα:



Από το παραπάνω σχήμα προκύπτει η εξής σειρά (αγνοείται το μηδέν):

4 1 3 6 2 7 5 8

Κατασκευάστε την αντίστοιχη σειρά για την τελική διάταξη του παιχνιδιού:



1 2 3 6 5 4 7 8

Μετρήστε τέλος πόσα ζευγάρια αριθμών έχουν διαφορετική σειρά εμφάνισης στις δύο διατάξεις. Στο τρέχον παράδειγμα τα ζευγάρια αυτά είναι τα (4,1), (4,3), (4,2), (4,5), (4,6), (3,2), (6,2) και (7,5). Εάν το πλήθος των ζευγαριών είναι μονός αριθμός, το πρόβλημα δεν έχει λύση. Αντίθετα εάν το πλήθος των ζευγαριών είναι ζυγός αριθμός, το πρόβλημα έχει λύση. Στη συγκεκριμένη περίπτωση έχουμε οκτώ αλλαγές σειράς, άρα το πρόβλημα έχει λύση.

Το πρόγραμμα generate.c, κάθε φορά που δημιουργεί ένα πρόβλημα ελέγχει εάν αυτό είναι επιλύσιμο. Εάν δεν είναι, το αγνοεί και δημιουργεί νέο. Έτσι είναι εξασφαλισμένο πως όλα τα προβλήματα που θα παραχθούν είναι επιλύσιμα.

β) Το πρόγραμμα `verify.c` δέχεται ως είσοδο το όνομα ενός αρχείου περιγραφής προβλήματος και ενός αρχείου με τη λύση του προβλήματος και ελέγχει εάν η λύση είναι έγκυρη ή όχι, τυπώνοντας στην οθόνη τα αντίστοιχα μηνύματα. Εάν το εκτελέσιμο ονομαστεί `verify.exe`, μπορείτε να το καλέσετε ως εξής:

```
verify.exe input.txt  solution.txt
```

όπου η γραμμογράφηση των αρχείων `input.txt` και `solution.txt` έχει περιγραφεί στο 1ο θέμα της εργασίας.

Κριτήρια αξιολόγησης:

Θέμα 1ο : Τυφλοί αλγόριθμοι αναζήτησης	
Αναζήτηση πρώτα σε πλάτος	20
Αναζήτηση πρώτα σε βάθος	20
Θέμα 2ο : Ευρετικοί αλγόριθμοι αναζήτησης	
Αναζήτηση πρώτα στο καλύτερο	20
Αναζήτηση A*	10
Θέμα 3ο : Υπολογιστική μελέτη	20
Γενική εικόνα (ευανάγνωστος κώδικας, σχολιασμός, κλπ.)	10
ΣΥΝΟΛΟ	**100**

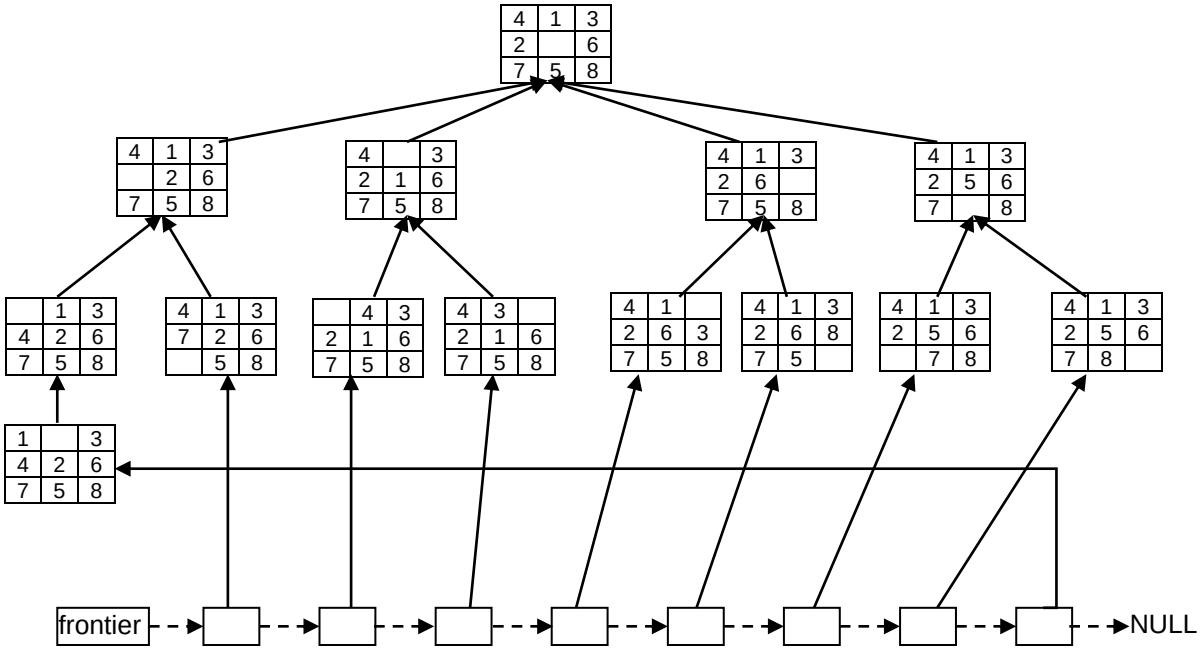
Ο συνολικός βαθμός θα διαιρεθεί δια 100, ώστε να προκύψει ο τελικός βαθμός της εργασίας (με μέγιστη τιμή το 1).

Υποδείξεις

Θα δείξουμε τα πρώτα βήματα εφαρμογής των αλγορίθμων για το παράδειγμα της πρώτης σελίδας. Όλοι οι αλγόριθμοι, καθώς αναζητούν τη λύση, κατασκευάζουν ένα δένδρο αναζήτησης, του οποίου η ρίζα είναι η αρχική κατάσταση και τα διάφορα κλαδιά αντιστοιχούν σε εναλλακτικές ακολουθίες κινήσεων του παίκτη. Σημειώστε ότι το δένδρο αναζήτησης δεν είναι δυαδικό, αλλά κάθε κόμβος μπορεί να έχει μέχρι 4 παιδιά. Επίσης όλοι οι αλγόριθμοι αναζήτησης πρέπει να διατηρούν μία λίστα από δείκτες προς τα φύλλα του δένδρου (το λεγόμενο μέτωπο αναζήτησης – frontier).

Αναζήτηση πρώτα σε πλάτος

Παρακάτω φαίνεται η αρχή του δένδρου αναζήτησης που κατασκευάζει ο αλγόριθμος αναζήτησης πρώτα σε 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. Σε κάθε επανάληψη εξάγεται από το μέτωπο ο πρώτος κόμβος και εισάγονται στο μέτωπο τα παιδιά αυτού. Οι επιμέρους αλγόριθμοι διαφοροποιούνται ανάλογα με το σε ποια θέση του μετώπου εισάγονται τα παιδιά του εξαγόμενου κόμβου.

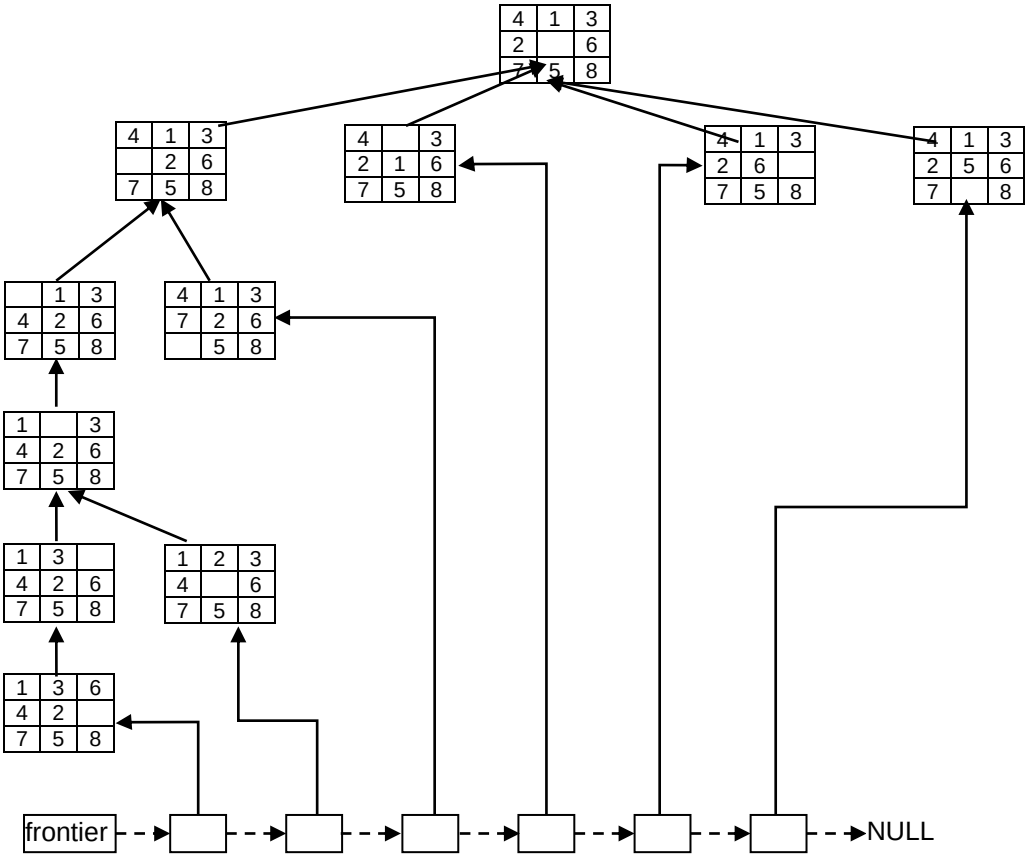
Στο παραπάνω διάγραμμα που αφορά τον αλγόριθμο αναζήτησης πρώτα σε πλάτος έγιναν οι εξής δύο παραδοχές:

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

β) Κάθε φορά που δημιουργείται ένα παιδί, γίνεται έλεγχος για βρόχο σε όλους τους προγόνους του. Εάν το παιδί ταυτίζεται με κάποιον από τους προγόνους του, 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.

Αναζήτηση πρώτα σε βάθος

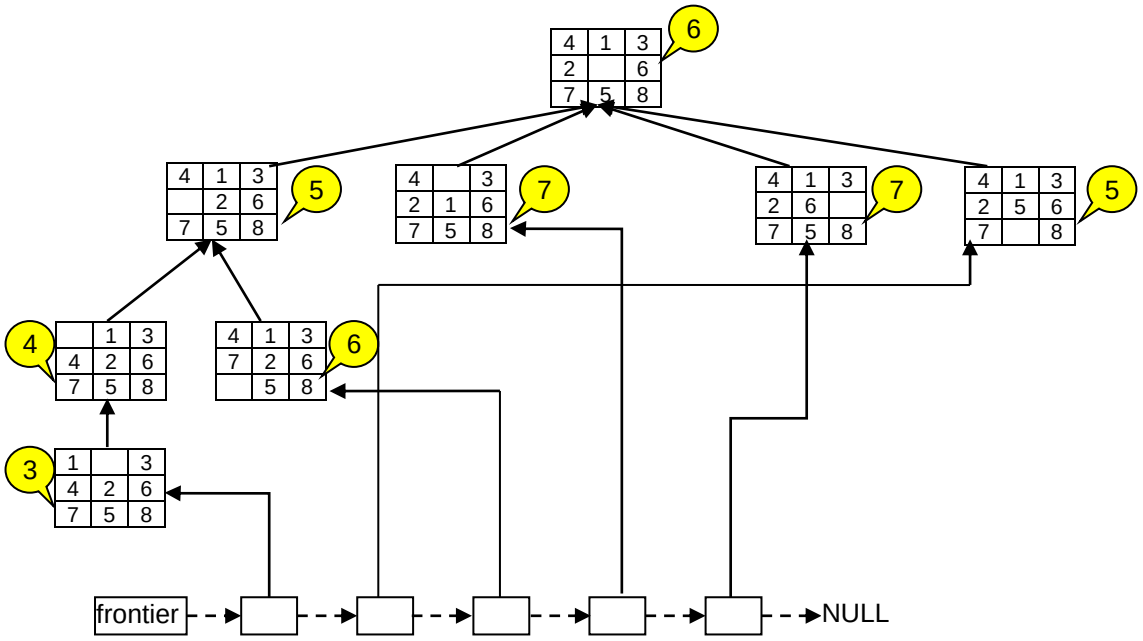
Παρακάτω φαίνεται η αρχή του δένδρου αναζήτησης που κατασκευάζει ο αλγόριθμος αναζήτησης πρώτα σε βάθος.



Ισχύουν και εδώ οι παραδοχές (α) και (β) του αλγορίθμου πρώτα σε πλάτος. Προσέξτε τη διαφορετική προτεραιότητα με την οποία εισάγονται στο μέτωπο αναζήτησης τα παιδιά του εξαγόμενου κόμβου. Στην αναζήτηση πρώτα σε βάθος εισάγονται στην αρχή του μετώπου αναζήτησης, με αποτέλεσμα ο αλγόριθμος να επιλέγει ένα κλαδί και να το ακολουθεί μέχρι τέλους, πριν ελέγξει τα γειτονικά κλαδιά.

Αναζήτηση πρώτα στο καλύτερο

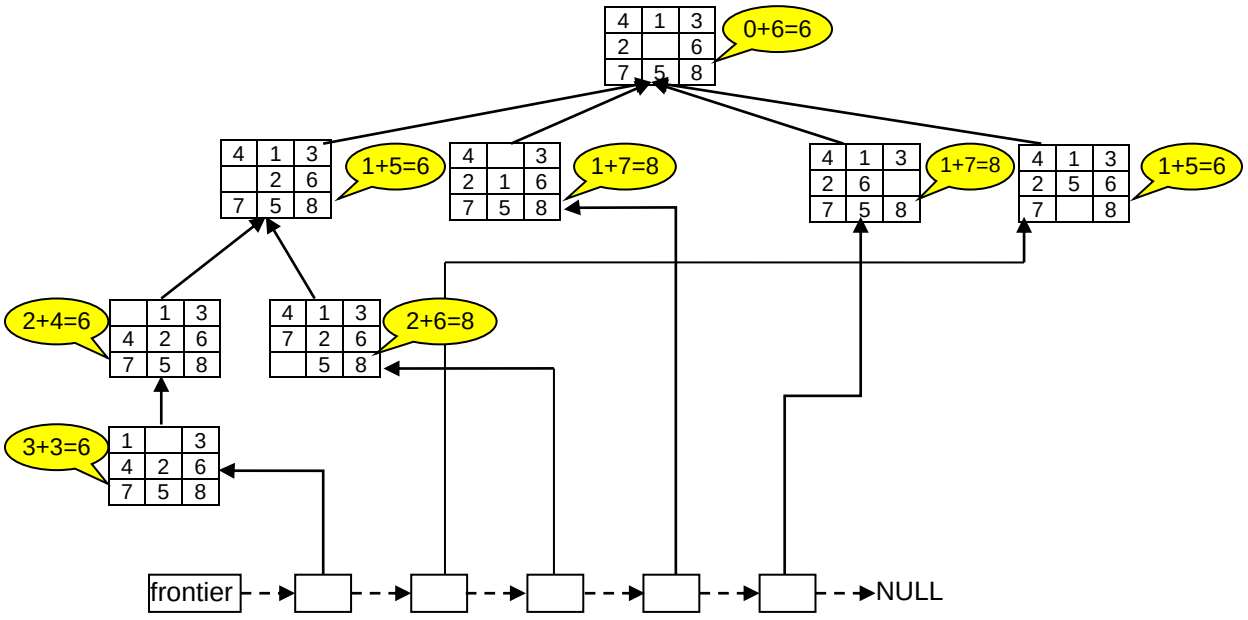
Στην αναζήτηση πρώτα στο καλύτερο κάθε κόμβος πρέπει να βαθμολογείται βάση μιας ευρετικής συνάρτησης που μετράει την απόσταση του κόμβου από την τελική επιθυμητή διάταξη. Οι νέοι κόμβοι εισέρχονται στο μέτωπο αναζήτησης με τέτοιον τρόπο ώστε να διατηρείται μια αύξουσα ταξινόμηση. Σε περίπτωση ισοβαθμίας 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.



Αναζήτηση Α*

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.

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. Στο διάγραμμα που ακολουθεί φαίνεται η αρχή του δένδρου αναζήτησης του αλγορίθμου Α*, συμπεριλαμβανομένων και των βαθμών των κόμβων.



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 είναι το εξής:

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

Για το μέτωπο αναζήτησης θα χρειαστείτε μια δομή ικανή να δημιουργήσει διπλά συνδεδεμένες λίστες, με δείκτη προς κόμβους του δένδρου αναζήτησης σε κάθε κόμβο της λίστας:

```
struct frontier_node
{
      struct tree_node *leaf;
      struct frontier_node *next;
      struct frontier_node *previous;
};
```

Φυσικά θα χρειαστείτε μεταβλητές/δείκτες που να δείχνουν στη ρίζα του δένδρου και στην αρχή του μετώπου αναζήτησης:

```
struct tree_node *search_tree;
struct frontier_node *frontier_head;
struct frontier_node *frontier_tail;
```

Από κει και πέρα, το πρόγραμμά σας θα πρέπει να είναι ικανό 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).