|  |
| --- |
|  |
| Αλγόριθμοι Αναζήτησης |
| Τυφλή και Στοχαστική αναζήτηση |
|  |
| **ΚΙΡΚΑΛΑΣ ΠΑΝΑΓΙΩΤΗΣ / ics22025** |
|  |

|  |
| --- |
|  |

Περιεχόμενα

[Επεξήγηση κώδικα 3](#_Toc156564398)

[Depth: 3](#_Toc156564399)

[Breadth: 3](#_Toc156564400)

[Best: 3](#_Toc156564401)

[Astar: 3](#_Toc156564402)

[Αποτελέσματα 5](#_Toc156564403)

[Σύγκριση & σχολιασμός αλγόριθμων 5](#_Toc156564404)

Επεξήγηση κώδικα

Υλοποίησα τους αλγόριθμους χρησιμοποιώντας την γλώσσα προγραμματισμού C++.

Όρισα μια κλάση να αναπαριστά κάθε αλγόριθμο, μια που να αναπαριστά έναν κόμβο στο δέντρο αναζήτησης και ένα Main.cpp αρχείο όπου βρίσκεται η main συνάρτηση.

Κάθε μια από τις τέσσερεις κλάσεις αλγορίθμων εφαρμόζει τον αλγόριθμο που της αντιστοιχεί και επιστρέφει τον κόμβο-λύση, αν τον βρει μέσα στο χρονικό όριο των εξήντα δευτερολέπτων.

Η κλάση που αναπαριστά έναν κόμβο περιέχει τα δεδομένα του κόμβου, έναν πίνακα με δείκτες προς τα παιδιά του, διάφορες πληροφορίες για αυτόν, όπως για παράδειγμα, ποιας πράξης (+, -, \*, /, ^, τετραγωνική ρίζα) αποτέλεσμα είναι, ή αν είναι root και μεθόδους για την διαχείριση του κόμβου, όπως την δημιουργία στιγμιότυπων κόμβων που θα αποθηκευτούν ως παιδιά του.

Το αρχείο Main.cpp αναλύει τα terminal arguments που δόθηκαν στο πρόγραμμα κατά την κλήση του, καλεί την ανάλογη κλάση βάσει αυτών, και αν βρεθεί κόμβος-λύση, γράφει το «μονοπάτι» προς τη λύση που βρέθηκε στο αρχείο που αναφέρθηκε στα terminal arguments.

## Depth:

Τον υλοποίησα με ένα stack και κάθε φορά που προχωρούσε σε έναν κόμβο, πρόσθετε μια αναφορά στον γονιό του σε αυτό, εκτός αν δεν ήταν η πρώτη φορά που προχωρούσε σε παιδί του γονιού. Έτσι, αν βρισκόταν σε αδιέξοδο, απλά θα πήγαινε στον κόμβο που βρίσκεται στην κορυφή του stack. Αν για δυο συνεχόμενες φορές, το «μονοπάτι» που ακολούθησε μεγάλωσε την διαφορά μεταξύ του αριθμού του κόμβου που βρίσκεται τώρα και του αριθμού-λύση θεωρεί ότι βρέθηκε σε αδιέξοδο. Για να αποφύγει να ξαναβρεθεί σε αδιέξοδο, δεν ξαναχρησιμοποιεί την πράξη που τον οδήγησε τελευταία σε ένα, για όλα τα παιδιά του γονιού κόμβου, και τα παιδιά τους. Έκανα μια ακόμα κλάση, που καλείται όμως μόνο από την κλάση του Depth, που ενθυλακώνει τους κόμβους του δέντρου ώστε να αποθηκεύει ποιες πράξεις έχουν αποκλειστεί για τον καθένα.

## Breadth:

Τον υλοποίησα με έναν δυναμικό πίνακα, vector όπως λέγεται στην C+ +. Προσθέτει όλους τους διαθέσιμους κόμβους στον πίνακα (στην αρχή θα είναι μόνο το root), ελέγχει αν κάποιος από αυτούς είναι ο κόμβος-λύση, αν είναι εννοείται τον επιστρέφει και τελειώνει, αν όχι, τους αφαιρεί από τον πίνακα και προσθέτει τα παιδιά όλων τους. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να βρεθεί ο κόμβος-λύση ή να λήξει το χρονικό όριο.

## Best:

Τον υλοποίησα με έναν δυναμικό πίνακα. Προσθέτει όλους τους διαθέσιμους κόμβους στον πίνακα (στην αρχή θα είναι μόνο το root), ελέγχει ο αριθμός ποιανού από αυτούς έχει την λιγότερη απόσταση από τον αριθμό λύση (αν είναι ο κόμβος-λύση θα έχει απόσταση μηδέν), αφαιρεί αυτόν τον κόμβο από τον πίνακα και προσθέτει τα παιδιά του. Αν βρει τον κόμβο-λύση τον επιστρέφει. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να βρεθεί ο κόμβος-λύση ή να λήξει το χρονικό όριο.

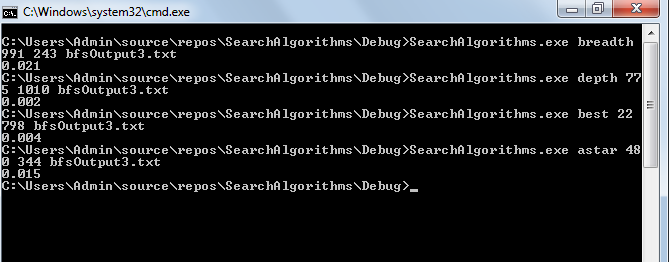
## Astar:

Τον υλοποίησα με έναν δυναμικό πίνακα. Προσθέτει όλους τους διαθέσιμους κόμβους στον πίνακα (στην αρχή θα είναι μόνο το root), ελέγχει ποιος κόμβος έχει το καλύτερο (λιγότερο) σκορ, αφαιρεί αυτόν τον κόμβο από τον πίνακα και προσθέτει τα παιδιά του. Αν βρει τον κόμβο-λύση τον επιστρέφει. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να βρεθεί ο κόμβος-λύση ή να λήξει το χρονικό όριο. Το σκορ υπολογίζεται από την απόσταση από τον αριθμό λύση + το κόστος του να φτάσουμε στον νυν κόμβο. Έκανα μια ακόμα κλάση, που καλείται όμως μόνο από την κλάση του Astar, που ενθυλακώνει τους κόμβους του δέντρου ώστε να αποθηκεύει το σκορ του καθενός.

Η απόσταση που αναφέρεται στην περιγραφή του Best και του Astar, υπολογίζεται από μια ευρετική συνάρτηση η οποία υπολογίζει απλά την απόλυτη τιμή της διαφοράς μεταξύ δυο αριθμών. Θεώρησα ότι μια τέτοια ευρετική συνάρτηση θα είναι επαρκής επειδή δεν προσπαθούμε να κάνουμε τίποτα παραπάνω από το να πάμε από τον ένα αριθμό στον άλλον.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Breadth | | Depth | | Best | | AStar | |
|  | | Χρόνος (sec) | Βήματα | Χρόνος (sec) | Βήματα | Χρόνος (sec) | Βήματα | Χρόνος (sec) | Βήματα |
| #1 | 779 - 55 | 0.001 | 4 | 0.008 | 726 | 0.00 | 11 | 0.00 | 8 |
| #2 | 991 - 243 | 0.002 | 4 | 0.009 | 751 | 0.00 | 10 | 0.00 | 10 |
| #3 | 985 - 835 | - | - | 0.001 | 152 | 0.009 | 150 | 0.012 | 150 |
| #4 | 257 - 999 | 1.044 | 8 | 0.007 | 742 | 0.001 | 31 | 0.00 | 31 |
| #5 | 22 - 798 | 0.008 | 5 | 0.005 | 776 | 0.011 | 172 | 0.06 | 143 |
| #6 | 480 - 344 | 0.06 | 6 | 0.002 | 138 | 0.003 | 105 | 0.031 | 136 |
| #7 | 77 - 400 | 0.002 | 4 | 0.003 | 323 | 0.003 | 94 | 0.007 | 94 |
| #8 | 182 - 261 | 0.067 | 6 | 0.001 | 79 | 0.004 | 79 | 0.002 | 79 |
| #9 | 775 - 1010 | - | - | 0.001 | 235 | 0.013 | 235 | 0.027 | 235 |
| #10 | 640 - 771 | - | - | 0.002 | 131 | 0.002 | 131 | 0.005 | 131 |

# Αποτελέσματα



Τα αποτελέσματα στον πίνακα και στο στιγμιότυπο οθόνης παράχθηκαν σε διαφορετικές ημέρες. Σε αυτό ίσως να οφείλεται η διαφορά στους χρόνους εκτέλεσης.

# Σύγκριση & σχολιασμός αλγόριθμων

Ο BFS καταφέρνει πάντα να βρίσκει την πιο σύντομη λύση, συνήθως με πολύ μεγάλη διαφορά από τους υπόλοιπους αλγόριθμους, αλλά είναι πιο χρονοβόρος. Επίσης καταλαμβάνει σημαντικά περισσότερη μνήμη, με αποτέλεσμα σε κάποιες περιπτώσεις να αποτυχαίνει, πριν λήξει το χρονικό περιθώριο που του δίνεται.

Ο DFS είναι αρκετά γρήγορος και απαιτεί ελάχιστη μνήμη, συγκριτικά με τους υπόλοιπους, αλλά συνήθως, η λύση που βρίσκει περιέχει τα περισσότερα βήματα και είναι πολύ πιθανό να «παγιδευτεί» σε ατέρμονες βρόχους, αν δεν παραλλαχθεί για να αποφευχθεί κάτι τέτοιο.

Ο Best είναι αρκετά καλός σε κάθε τομέα και ο Α\* μου φαίνεται απλά σαν μια πιο βέλτιστη εκδοχή του. Και οι δύο αλγόριθμοι δεν απαιτούν όση μνήμη όση ο BFS, αλλά κάνουν περισσότερα βήματα από αυτόν, όχι όμως όσα ο DFS. Οι λύσεις που βρίσκουν κοστίζουν λιγότερο από τον DFS και πολλές φορές από τον BFS ακόμα. Η υλοποίησή τους είναι λίγο πιο δύσκολη από τον BFS, αλλά σημαντικά πιο εύκολη από τον DFS. Αν μπορούμε να κάνουμε μια ικανοποιητική ευρετική συνάρτηση, δε «βλέπω» λόγο να μην υλοποιήσουμε οποιονδήποτε από αυτούς.