ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

1η Εργασία

Peg Solitaire Solver

Όνομα: Δημήτρης

Επίθετο: Μανωλάκης

A.M.: it1423

Περιεχόμενα

Εισαγωγή

Στόχος της παρούσας εργασίας είναι η ανάπτυξη λογισμικού το οποίο επιλύει Peg Solitaire puzzles κανοντας χρήση των αλγορίθμων: Αναζήτηση πρώτα σε βάθος και Αναζήτηση πρώτα στο καλύτερο. Συνολικά δημιουργήθηκαν 4 διαφορετικοί Solvers, τρεις από τους οποίους αφορούν τον αλγόριθμο Αναζήτησης πρώτα στο καλύτερο χρησιμοποιώντας ο καθένας διαφορετική ευρετική συνάρτηση, ενώ ο τελευταίος αφορά τον αλγόριθμο Αναζήτησης πρώτα σε βάθος.

Όλες οι υλοποιήσεις χρονομετρήθηκαν στην επίλυση προβλημάτων Peg Solitaire διαφόρων μεγεθών και δυσκολίας προκειμένου να αναδειχτεί η ιδανική τεχνική για την αποτελεσματική αντιμετώπιση του puzzle αυτου. Τέλος η επικύρωση των λύσεων που παράγονται από το πρόγραμμα γίνεται μέσω αλλού προγράμματος που αναπτύχθηκε για το σκοπό αυτό και επεξηγείται αναλυτικά παρακάτω.

Τεκμηρίωση Κώδικα

1. Γενικές Πληροφορίες

Προτού γίνει μια πιο ειδική επεξήγηση της λειτουργίας τμημάτων (π.χ. Solvers) του προγράμματος θεωρήθηκε

συνετό να δοθούν κάποιες γενικές πληροφορίες σχετικά με τεχνικές και δομές δεδομένων που καθολικά χρησιμοποιούνται από όλες τις διαφορετικές υλοποιήσεις.

Γλώσσα Προγραμματισμού και Βιβλιοθήκες

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την ανάπτυξη του λογισμικού είναι η **Python**. Το πρώτο σημαντικό ζήτημα που προέκυψε ήταν η επιλογή της κατάλληλης δομής δεδομένων που θα αναπαριστά το αντίστοιχο Peg Solitaire board, το οποίο το πρόγραμμα καλείται να επιλύσει. Η δομή που κρίθηκε κατάλληλη ήταν αυτή του **πίνακα** λόγω της ομοιότητας της με το αντίστοιχο board ενός παιχνιδιού, επιτρέποντας μια άμεση και ξεκάθαρη αναπαράσταση του, διευκολύνοντας έτσι την διαδικασία του προγραμματισμού και τον μετέπειτα έλεγχο της λειτουργίας του προγράμματος. Επιπλέον, οι πίνακες στην Python υλοποιούνται από την δημοφιλής βιβλιοθήκη **NumPy** η οποία προσφέρει βελτιστοποιημένη επεξεργασία επί των δομών αυτών, οι οποίες παραδοσιακά επιβαρύνουν την λειτουργία ενός προγράμματος. Οι πίνακες συμπληρώνονται με ακέραιους από το 0 έως το 2 ακολουθώντας την ιδιά ακριβώς κωδικοποίηση για το board όπως αυτή δόθηκε από την εκφώνηση της εργασίας.

Τέλος, για την ανάγνωση, την δημιουργία αλλά και την εγγραφή σε αρχεία χρησιμοποιήθηκε η βιβλιοθήκη sys καθώς και η βιβλιοθήκη time για την χρονομέτρηση των αλγορίθμων.

Βασική προγραμματιστική τεχνική: Αναδρομή

Η βασική προγραμματιστική τεχνικη που εφαρμόζεται από όλους του Solvers είναι η **αναδρομή**. Υστέρα από την κάθε κίνηση που εκτελείται στο board του puzzle, ο Solver αναδρομικα καλεί τον εαυτό του στο ανανεωμένο board μέχρις ότου εντοπιστεί μια λύση. Η τεχνική επιλέχτηκε για την προγραμματιστική λιτότητα που προσφέρει καθώς και γιατί θεωρήθηκε πως είναι ιδανική για προβλήματα αυτού του είδους.

Επιπροσθέτως, δεν έγινε κάποια υλοποίηση δέντρου αναζήτησης αν και το πρόγραμμα ‘μιμείται’ την αναζήτηση σε μια δομή αυτού του είδους. Ξεκινώντας από την ριζά ενός δέντρου αναζήτησης (το αρχικό board), επιλέγεται μια από τις νόμιμες κινήσεις και ουσιαστικά από το σημείο αυτό, εξετάζονται όλες οι πιθανές πορείες (ανάλογα πάντα με τον αλγόριθμο που υλοποιείται), σημειώνοντας παράλληλα σε λίστα τις κινήσεις που πραγματοποιούνται όσο αναδρομικά προχωρά σε χαμηλότερα επίπεδα του δέντρου. Στις περιπτώσεις οπού δημιουργούνται αδιέξοδα το πρόγραμμα εκτελεί **backtracking** στην αμέσως επόμενη νόμιμη κίνηση.

Έλεγχος ομοιότητας

Στην αρχική υλοποίηση του προγράμματος κάνεις από τους Solvers δεν πραγματοποιούσε **έλεγχο ομοιότητας** στα boards που συναντώνται καθώς πραγματοποιούνται κινήσεις με τα πουλιά. Μια τέτοιου είδους τεχνική όμως προστέθηκε αργότερα προκειμένου να αυξήσει την χρονική απόδοση των αλγορίθμων. Κάθε board που συναντάται και δεν καταλήγει σε λύση αποθηκεύεται καταλληλά και σε περίπτωση που επανεμφανίζεται, το πρόγραμμα άμεσα πραγματοποιεί backtracking σε επόμενη κίνηση ανάλογα με τον εκάστοτε αλγόριθμο. Η προσθήκη αυτή στους Solvers μείωσε ουσιαστικά κατά το ήμισυ (κατά μέσο ορό) την χρονική διάρκεια της επίλυσης των προβλημάτων.

Σημείωση: Μολονότι ο έλεγχος ομοιότητας αναφέρεται σε προγονούς μια κατάστασης, στην προκείμενη περίπτωση αναφέρεται σε οποιαδήποτε κατάσταση έχει προηγουμένως συναντηθεί από το πρόγραμμα. Ίσως μια καλύτερη αναφορά στη τεχνική θα ήταν πως όλοι οι αλγόριθμοι έχουν εφοδιαστεί με μνήμη.

2. Depth-First Solver

Ο λυτής αυτός υλοποιείται μέσω της συνάρτησης ***depth_first_solver*** η οποία δέχεται 3 εισόδους: τον πίνακα (NumPy Array) που αποθηκεύει την τρέχουσα κατάσταση του board και στον οποίο θα πραγματοποιήσει μια κίνηση, τη λίστα path (Python List) οπού είναι αποθηκευμένες με την μορφή **συμβολοσειρών** οι κινήσεις που έχουν πραγματοποιηθεί μέχρι το σημείο αυτό (π.χ. 3 3 3 1) και την λίστα memory οπού είναι αποθηκευμένα όλα τα boards του puzzle που έχει συναντήσει ο λυτής στην εκτέλεση του (σε μορφή ακέραιου μέσω της ***ConvertToInt***). Η συνάρτηση δεν έχει κάποια έξοδο και παίζει ουσιαστικά το ρολό του επόπτη, ενημερώνει δηλαδή σε κάθε βήμα όλες τις παραμέτρους που λαμβάνει ως είσοδο όσο αναδρομικά διανύει το κλαδί του δέντρου αναζήτησης που έχει επιλέξει. Επομένως, ο σωστός ορός (με τον οποίο θα αναφέρεται από εδώ και στο εξής) είναι η **διαδικασία**.

Το πρώτο βήμα της διαδικασίας αυτής είναι η σάρωση του πίνακα που αποθηκεύει το board μέχρι να συναντήσει το πρώτο πουλί στο οποίο εκτελεί άμεσα μια από τις νόμιμες κινήσεις (αν αυτό επιτρέπεται) σε ένα αντίγραφο του πίνακα που έλαβε ως είσοδο (μέσω της συνάρτησης ***makeTheMove***). Στην συνέχεια, ελέγχεται η περίπτωση να έχει ήδη συναντήσει τον ανανεωμένο πίνακα κάνοντας χρήση της λίστας

memory και προχωράει είτε σε επόμενη κίνηση είτε σε επόμενο πουλί αν το board αυτό έχει εμφανιστεί ξανά. Στην αντίθετη περίπτωση αποθηκεύει το νέο αυτό board, όπως και την κίνηση που εκτέλεσε με το πουλί στην λίστα path και ελέγχει αν το πρόβλημα λύθηκε (μέσω της ***solutionFound***).

Συναρτήσεις

- ***ConvertToStr:*** Μετατρέπει ένα πίνακα σε συμβολοσειρά, σαρώνοντας τον και αποθηκεύοντας κάθε ψηφίο του, εκτός αν ισούται με 0 γιατί οι θέσεις αυτές στο board δεν αλλάζουν ποτέ.

- ***makeTheMove:*** Δέχεται ως είσοδο ένα πίνακα, την τοποθεσία ενός πουλιού στον πίνακα αυτό και μια ‘νόμιμη’ κίνηση την οποία εκτελεί και επιστρέφει τον ανανεωμένο πίνακα.

- ***solutionFound:*** Δέχεται ως είσοδο ένα πίνακα και στην περίπτωση που περιέχει μόνο έναν άσσο επιστρέφει Αληθές.

3. Best-First Solver

Οι διαδικασίες-solvers που υλοποιούν τον αλγόριθμο πρώτα στο καλύτερο λειτουργούν παρόμοια με την υλοποίηση του αλγορίθμου πρώτα σε βάθος. Η ουσιαστική διαφορά προέρχεται από το γεγονός ότι τα πουλιά που θα κινηθούν πρώτα δεν επιλέγονται με βάση την θέση τους στο πίνακα αλλά εξαρτάται από την τιμή της αντίστοιχης ευρετικής συνάρτησης.

Προκειμένου να γίνεται σε κάθε επίπεδο της αναζήτησης η σύγκριση αναμεσά στα διαθέσιμα πουλιά και τις διαθέσιμες κινήσεις, αποθηκεύεται σε **λεξικό**, η τιμή της ευρετικής συνάρτησης ως κλειδί και η αντίστοιχη κίνηση (θέση του πουλιού + κατεύθυνση της κίνησης) ως τιμή. Επειδή συχνά πολλές κινήσεις έχουν την ιδιά τιμή στην ευρετική συνάρτηση, οι τιμές του λεξικού αποθηκεύονται ως λίστες και στις περιπτώσεις που υπάρχει ισοπαλία, οι λίστες αυτές εξετάζονται με προτεραιότητα (πάντα) από την αρχή της, προς το τέλος. Στην συνέχεια το λεξικό ταξινομείται και το πρόγραμμα επιλεγεί μια προς μια τις υποψήφιες κινήσεις, από το καλύτερο προς το χειρότερο, κάνοντας πάντα έλεγχο ομοιότητας αναμεσά στους νέους πίνακες που προκύπτουν.

Α) Ευρετική Συνάρτηση: Rating

Η ονομασία της ευρετικής συνάρτησης αυτή προκύπτει από το γεγονός πως ουσιαστικά ‘βαθμολογεί’ τον αντίστοιχο πινακα-board με βάση δυο του χαρακτηριστικά:

- Το πλήθος των νόμιμων κινήσεων που υπάρχουν στον πίνακα αυτό.
- Το πλήθος των απομονωμένων πουλιών (αυτά που δεν συνορεύουν με άλλο πούλι από πάνω, δεξιά, αριστερά ή κάτω).

Με βάση τα δυο κριτήρια αυτά προκύπτει ένα άθροισμα-βαθμολογία για κάθε πίνακα που εξετάζεται, με στόχο να υπάρχουν όσο το δυνατόν περισσότερες κινήσεις σε αυτόν που θα επιλεχθεί.

Β) Ευρετική Συνάρτηση: Manhattan

Η ευρετική συνάρτηση αυτή προκύπτει από τον υπολογισμό του Manhattan Distance αναμεσά σε κάθε πουλί του πίνακα που προκύπτει από την αντίστοιχη κίνηση που εξετάζεται. Ο υπολογισμός της αποτελεί τον πιο δαπανηρό από τις 3 ευρετικές συναρτήσεις που υλοποιήθηκαν.

Γ) Ευρετική Συνάρτηση: Area

Το κριτήριο της συνάρτησης αυτής είναι ο μικρότερος τετραγωνικός πίνακας που περιέχει όλα τα εναπομείναντα πούλια. Σε αντίθεση με την προηγουμένη ευρετική συνάρτηση, ο υπολογισμός της τιμής της για ένα πίνακα είναι ιδιαίτερα ελαφρύς και επηρέασε θετικά την απόδοση της.

4.Πρόγραμμα ελέγχου λύσεων

Για τον έλεγχο των λύσεων που προέκυψαν από κάθε Solver δημιουργήθηκε κατάλληλο πρόγραμμα, το οποίο δέχεται ως είσοδο το Input.txt ώστε να διαβάσει τον αρχικό πίνακα και σε αυτόν εκτελεί κάθε κίνηση που διαβάζει στο Output.txt. Ανάλογα με την ύπαρξη ενός ή περισσότερων άσσων στον πίνακα που προκύπτει εκτυπώνεται το κατάλληλο μήνυμα σχετικά με το αν αποτελεί λύση του προβλήματος.

Υπολογιστική Μελέτη

Στον παρακάτω πίνακα καταγράφονται τα αποτελέσματα από την επίλυση μιας συλλογής 13 προβλημάτων Peg Solitaire από κάθε Solver που υλοποιήθηκε:

	Depth-First	Best-First (Rating)	Best-First (Area)	Best-First (Manhattan)
Cross	0.001994 sec	0.002992 sec	0.002991 sec	0.004984 sec
Triangle	0.300736 sec	0.105698 sec	0.028921 sec	8.360190 sec
Pointer	0.336869 sec	0.209994 sec	0.191487 sec	2.013663 sec
Flag	1.286104 sec	0.135871 sec	0.043909 sec	3.763476 sec
Ring	0.468884 sec	0.033880 sec	22.78338 sec	3 min 8 sec
Square 6x6	1 min 27 sec	1.175808 sec	6.907615 sec	1.533894 sec
English (4)[1]	5.016579 sec	---	2.946135 sec	---
Target	---	---	49.97341 sec	3 min 51 sec
Letter X	---	0.423349 sec	12 min 41 sec	28.794959 sec
French (1)	---	---	---	---
German (2)	---	---	---	---
Asymmetrical (3)	---	22 min 56 sec	10 min 49 sec	---
Diamond (5)	---	---	---	---

Σε συνδυασμό με τα 6 προβλήματα που δόθηκαν από την εκφώνηση της άσκησης (Cross, English, French κ.α.) προστέθηκαν και επιπλέον 7 προβλήματα κυρίως μικρής δυσκολίας ώστε να υπάρχει μια ισορροπία στην δυσκολία των προβλημάτων. Τα προβλήματα είναι ταξινομημένα από τα πιο ευκολά στα δύσκολα. Οπού υπάρχουν τρεις παύλες (---) σημαίνει ότι ο αντίστοιχος αλγόριθμος δεν κατάφερε σε 30 λεπτά να λύσει το πρόβλημα.

Παρατηρωντας τις χρονομετρήσεις, ένα πρώτο και σημαντικό συμπέρασμα είναι πως η χρήση της

[1] Τα αριθμημένα προβλήματα αποτελούν τα προβλήματα που δόθηκαν στην εκφώνηση της άσκησης και έχουν αριθμηθεί με τον ίδιο τρόπο για εύκολη αναγνώριση. Το απλό πρόβλημα (Cross) που δινόταν στην διαφάνεια δεν έχει αριθμηθεί.

αναζήτησης πρώτα στο καλύτερο, η ύπαρξη δηλαδή μια ευρετική συνάρτησης έχει ιδιαίτερα θετική επίδραση στην ταχύτητα με την οποία το πρόγραμμα λύνει τα προβλήματα. Αν και η υλοποίηση που έγινε με την αναζήτηση πρώτα σε βάθος λύνει 7 από τα 13 προβλήματα και σε ανταγωνιστικά χρονικά πλαίσια είναι εμφανές πως όσο αυξάνεται το μέγεθος των προβλημάτων, η επίλυση τους χωρίς ευρετική συνάρτηση έχει μεγάλη χρονική διάρκεια. Η επιβάρυνση που επιφέρει ο υπολογισμός της ευρετική συνάρτησης φαίνεται να μην επηρεάζει στις περισσότερες περιπτώσεις την απόδοση του προγράμματος, σε σχέση ειδικά με τον χρόνο που εξοικονομείται λόγω της ‘κατεύθυνσης’ που δίνει στην αναζήτηση. Το φαινόμενο αυτό είναι ιδιαίτερα εμφανές στην απόδοση των ευρετικών συναρτήσεων Rating και Area ενώ η ευρετική συνάρτηση Manhattan αναδεικνύει το γεγονός πως το βάρος του υπολογισμού της σε κάθε βήμα δεν είναι ισορροπείται με την εξοικονόμηση χρόνου που προσφέρει. Ακόμα και έτσι όμως, η υλοποίηση Best-First (Manhattan) είχε καλύτερη απόδοση από την Depth-First υλοποίηση καθώς έλυσε 8 από τα 13 προβλήματα (αν και με μικρότερη ταχύτητα κατά μέσο ορό).

Συγκρίνοντας τις τρεις διαφορετικές ευρετικές συναρτήσεις είναι εμφανές πως η συνάρτηση Area έχει την καλύτερη απόδοση λύνοντας 10 από τα 13 προβλήματα και μάλιστα 6 από αυτά, με την μεγαλύτερη ταχύτητα. Οι δυο συναρτήσεις Area και

Rating έχουν παρόμοια πολυπλοκότητα στον υπολογισμό που απαιτούν καθώς πρόκειται ουσιαστικά για μια σάρωση του υπό εξέταση πίνακα. Επομενως, είναι ασφαλές να συμπεράνουμε ότι η ευρετική συνάρτηση Area εκφράζει με καλύτερο τρόπο την πιθανότητα μια από τις διαθέσιμες κινήσεις να οδηγήσουν στη λύση του προβλήματος.

Συμπεράσματα

Το πιο σημαντικό συμπέρασμα σχετικά με την υλοποίηση προγραμμάτων που επιλύουν τέτοιου είδους προβλήματα, είναι πως όσο αποτελεσματική και να είναι η υπό χρήση ευρετική συνάρτηση, είναι υψίστης σημασίας να γίνεται εξοικονόμηση χρόνου χρησιμοποιώντας ιδανικές προγραμματιστικές τεχνικές και τύπους δεδομένων. Για παράδειγμα, η παρούσα εργασία πέτυχε την δημιουργία ενός Solver που λύνει μεγάλη ποικιλία προβλημάτων αλλά σιγουρά υπάρχουν τρόποι για περαιτέρω βελτίωση, όπως η αποθήκευση του κάθε Board με κάποια αποτελεσματική κωδικοποίηση και όχι σε NumPy array.

Μια επιπλέον πρόταση για το μέλλον σχετικά με την επίλυση τέτοιων προβλημάτων θα ήταν η χρήση παραλληλισμού, οπού η εργασία της αναζήτησης μοιράζεται σε νήματα με ιδιαίτερη προσοχή να ελέγχει το καθένα διαφορετικά κλαδιά του δέντρου αναζήτησης.