**Εργασία στη Τεχνητή Νοημοσύνη**

**Παυλίδης Παύλος dai16035**

Περιεχόμενα

[Peg Solitaire 2](#_Toc512799694)

[BFS – find\_solution\_BFS 2](#_Toc512799695)

[DFS – find\_solution\_DFS 4](#_Toc512799696)

[Επαλήθευση λύσης – verify\_solution.py 5](#_Toc512799697)

[Εκτελέσεις των 2 αλγορίθμων 6](#_Toc512799698)

[Παράρτημα 8](#_Toc512799699)

[1. Συναρτήσεις προγραμμάτων: 8](#_Toc512799700)

[1.1 pegsol.py 8](#_Toc512799701)

[1.2 verify\_solution.py 10](#_Toc512799702)

[2. Αρχεία με δοκιμών προγράμματος 11](#_Toc512799703)

# Peg Solitaire

H παρακάτω η εργασία αφορά την χρήση των αλγορίθμων DFS (Depth – First- Search) και BFS (Best – First – Search) για την εύρεση λύσης του παιχνιδιού Peg Solitaire. Και το πρόγραμμα εύρεσης λύσης (pegsol.py) αλλά και το πρόγραμμα επαλήθευσης λύσης (verify\_solution.py) έχουν γραφτεί σε Python. Σε κανέναν από τους 2 αλγορίθμους δε γίνεται έλεγχος αν βρεθεί σε ατέρμον βρόχο καθώς θεωρήθηκε αδύνατο. Το σκεπτικό ήταν όπως αφαιρώντας έναν πάσαλο αλλάζει η μορφή του ταμπλό συνεπώς δεν υπάρχει περίπτωση να βρεθούμε σε ακριβώς ίδια κατάσταση με αυτή που έχουμε προηγουμένως βρεθεί. Από την στιγμή που κάθε φορά ελέγχεται και γίνεται 1 μοναδική κίνηση η οποία αν απορριφθεί για την τρέχουσα κατάσταση δεν πρόκειται να ξαναεμφανιστεί τότε είναι αδύνατο να ξανά παρθεί η ίδια λάθος απόφαση και να βρεθούμε σε ατέρμον βρόχο. Υπάρχει όμως περίπτωση να γίνει κίνηση που περιλαμβάνει 2 φορές την ίδια μεταφορά. Δηλαδή κάποιος πάσαλο από το (1,1) να μετακινηθεί στο (1,3) και μετά από κάποια βήματα κάποιος πάσσαλος που έχει βρεθεί στο (1,1) να χρειαστεί να μετακινηθεί επίσης στο (1,3). Από τη στιγμή που κατά την διαδικασία αυτών των βημάτων έχουν αφαιρεθεί πάσσαλοι, οι καταστάσεις αυτές θεωρούνται διαφορετικές.

Για την εκτέλεση του pegsol απαιτούνται τα 3 παρακάτω ορίσματα:

1. Μέθοδος επίλυσης : depth ή best
2. Αρχείο δεδομένων : μπορεί να πάρει οποιοδήποτε όνομα έχει το αρχείο του χρήστη
3. Αρχείο εξόδου : μπορεί να πάρει οποιοδήποτε όνομα επιθυμεί ο χρήστης

Το πρόγραμμα τερματίζει γράφοντας την λύση στο αρχείο εξόδου και εκτυπώνοντας στη κονσόλα το συνολικό χρόνο εκτέλεσης

Για το verify\_solution απαιτούνται 2 ορίσματα :

1. Αρχείο δεδομένων 1 : μπορεί να πάρει οποιοδήποτε όνομα έχει το αρχείο του χρήστη. Πρόκειται για αρχείο που περιέχει το πρόβλημα
2. Αρχείο δεδομένων 2 : μπορεί να πάρει οποιοδήποτε όνομα έχει το αρχείο του χρήστη. Πρόκειται για αρχείο που περιέχει τη λύση του προβλήματος

Το πρόγραμμα τερματίζει εκτυπώνοντας στη κονσόλα αν η λύση είναι σωστή ή όχι.

# BFS – find\_solution\_BFS

Για την επίλυση με τον αλγόριθμο BFS χρησιμοποιείται η συνάρτηση find\_solution\_BFS :

Δέχεται ως ορίσματα:

* game\_board : την τωρινή κατάσταση του ταμπλό
* counter: την τωρινή τιμή του μετρητή βημάτων. Στην αρχή του προγράμματος (στην 1η κλήση δηλαδή της συνάρτησης) παίρνει την τιμή μηδέν.

Κατά την κλήση της ορίζεται μια τοπική μεταβλητή flag ίση με τον μετρητή βημάτων και μια κενή λίστα που θα τοποθετηθούν οι τιμές τις ευρετικής συνάρτησης. Στην συνέχεια γίνεται έλεγχος αν έχει βρεθεί λύση. Αν ναι, γίνεται η εγγραφή των κινήσεων και του μετρητή στο αρχείο εξόδου και επιστρέφει την τιμή -1 ο οποία χρησιμοποιείται σαν τιμή τερματισμού όλων των κλήσεων της find\_solution\_BFS . Πιο συγκεκριμένα επιστρέφοντας την τιμή -1 στην εντολή counter = findsolution(child\_board,counter) θα έχει ως αποτέλεσμα το if(counter == -1) να είναι αληθής και να εκτελεστεί η εντολή break. Έτσι επιστρέφεται το -1 που έχει εκείνη τι στιγμή ο counter και προς τα πίσω κλείνουν όλες οι κλήσεις τις συνάρτησης.

Αν δεν έχει βρεθεί λύση, με την find\_child βρίσκονται και αποθηκεύονται στο λεξικό moves όλες τις πιθανές κινήσεις που μπορούν να γίνουν στην τρέχουσα κατάσταση του game\_board (Σημείωση : Για να βρεθούν τις πιθανές κινήσεις, βρίσκονται πρώτα τις κενές θέσεις και έπειτα για κάθε μια από αυτές ελέγχονται προς όλες τις κατευθύνσεις οι δυνατές κινήσεις που μπορούν να γίνουν προς αυτήν). Για κάθε μια από αυτές τις κινήσεις υπολογίζεται η τιμή της ερευετικής συνάρτησης και αποθηκεύεται στη λίστα heuristics\_values. Στην συνέχεια ταξινομείται αυτή η λίστα με βάση την τιμή της ευρετικής συνάρτησης σε αύξουσα σειρά. Χρησιμοποιώντας αυτή τη λίστα επιλέγεται κάθε φορά η καλύτερη τιμή της ευρετικής συνάρτησης ( όσο μικρότερη τόσο το καλύτερο ) και καλείται η make\_move η οποία επιστρέφει στην μεταβλητή child\_board το νέο ταμπλό μετά την επιλεγμένη κίνηση από την heuristics\_values λίστα. Έπειτα αυξάνεται ο μετρητής κατά ένα , ελέγχεται αν το πρόγραμμα τρέχει πάνω από 5 λεπτά προκειμένου να τερματιστεί και μέσω της line = adjust\_coordinats\_to\_write(position,direction) αποθηκεύεται στην μεταβλητή line μια λίστα με αλφαριθμητικά που περνά και αποθηκεύεται στην μεταβλητή movehistory η οποία κρατά το ιστορικό των κινήσεων που έχουν γίνει. Στη συνέχεια καλείται η counter = findsolution(child\_board,counter). Όταν επιστρέψει η κλήση της συνάρτησης αυτής, αν δεν αληθεύει το if(counter == -1) τότε σημαίνει πως φτάσαμε σε αδιέξοδο επομένως το πρόγραμμα θα συνεχίσει με την εντολή del movehistory[-1] που διαγράφει την τελευταία κίνηση που προστέθηκε και στη συνέχεια με την επιλογή του επόμενου καλύτερου βήματος. H if (flag == counter): return (counter - 1) θα εκτελεστεί σε περίπτωση που η find\_child επιστρέψει κενή λίστα και ο counter αυξηθεί άδικα.

Η ευρετική συνάρτηση def heuristic(game\_board, peg\_position, direction) αφού υπολογίσει την νέα μορφή του ταμπλό, υπολογίζει και επιστρέφει το γινόμενο του πλήθους των πασάλων που υπάρχουν στο ταμπλό μετά την επιλεγμένη κίνηση, επί την ορθογώνια περιοχή των πασάλων ( υπολογίζεται και αποθηκεύεται στην μεταβλητή area από την συνάρτηση calculate\_pegs\_area(temp\_board) ) και επί την απόσταση μανχάταν.

# DFS – find\_solution\_DFS

Για την επίλυση με τον αλγόριθμο DFS χρησιμοποιείται η συνάρτηση find\_solution\_DFS :

Δέχεται ως ορίσματα:

* game\_board : την τωρινή κατάσταση του ταμπλό
* counter : την τωρινή τιμή του μετρητή βημάτων. Στην αρχή του προγράμματος (στην 1η κλήση δηλαδή της συνάρτησης) παίρνει την τιμή μηδέν.

Το πρώτο μέρος της συνάρτησης είναι ίδιο με της find\_solution\_BFS . Δηλαδή, κατά την κλήση της ορίζεται μια τοπική μεταβλητή flag ίση με τον μετρητή βημάτων και μια κενή λίστα που θα τοποθετηθούν οι τιμές τις ευρετικής συνάρτησης. Στην συνέχεια γίνεται έλεγχος αν έχει βρεθεί λύση. Αν ναι γίνεται η εγγραφή των κινήσεων και του μετρητή στο αρχείο εξόδου και επιστρέφει την τιμή -1 ο οποία χρησιμοποιείται σαν τιμή τερματισμού όλων των κλήσεων της find\_solution\_DFS . Πιο συγκεκριμένα επιστρέφοντας την τιμή -1 στην εντολή counter = findsolution(child\_board,counter) θα έχει ως αποτέλεσμα το if(counter == -1) να είναι αληθής και να εκτελεστεί η εντολή break. Έτσι επιστρέφεται το -1 που έχει εκείνη τι στιγμή ο counter και προς τα πίσω κλείνουν όλες οι κλήσεις τις συνάρτησης.

Έπειτα για κάθε κελί – στοιχείο του ταμπλό – game\_board μέσω ενός for που σαρώνει τον πίνακα κατά πλάτος, συμβαίνουν τα εξής:

1. Ελέγχεται πρώτα αν έχουμε φτάσει σε λύση προκειμένου να τερματιστεί το for και να μην συνεχίσει άδικα το πρόγραμμα επιβαρύνοντας το με περισσότερο χρόνο εκτέλεσης.
2. Αν δεν τερματιστεί, ακολουθούν 4 if με μόνη διαφορά την κατεύθυνση (left, right, up, under αντίστοιχα). Το if ελέγχει (μέσω της valid\_move(game\_board, item, κατεύθυνση)) αν η κίνηση προς την κατεύθυνση που του αντιστοιχεί είναι εφικτή, δηλαδή αν η κίνηση δεν θα βγει εκτός ορίου του πίνακα, αν το κελί που βρίσκομαι είναι κατειλημμένο, το γειτονικό του προς αυτή την κατεύθυνση είναι επίσης κατειλημμένο και αν το αμέσως επόμενο προς την ίδια κατεύθυνση είναι ελεύθερο. Για παράδειγμα, βρισκόμαστε στο (1,1) με κατεύθυνση δεξιά. Αν το (1,2) έχει peg και το (1,3) είναι εντός ορίων του πίνακα και είναι κενή θέση – δεν έχει peg τότε η κίνηση θεωρείται εφικτή και το if αληθές.
3. Δημιουργείται ένα αντίγραφο του ταμπλό στο οποίο εφαρμόζεται η κίνηση αυτή και αποθηκεύεται στην μεταβλητή new\_board( new\_board = make\_move (game\_board, item, κατεύθυνση ) ). Αυτή θα χρησιμοποιηθεί και για την επόμενη κλήση της συνάρτησης και του αλγορίθμου. Ο λόγος που χρησιμοποιείται είναι για να μην τροποποιηθεί η παράμετρος της συνάρτησης που έχει περαστεί έτσι ώστε σε περίπτωση που απορριφθεί λόγο αδιεξόδου το βήμα αυτό να μην έχει χαθεί η προηγούμενη κατάσταση .
4. Αύξηση του μετρητή βημάτων κατά 1.
5. Έλεγχος αν το πρόγραμμα τρέχει περισσότερο από 5 λεπτά προκειμένου να τερματιστεί εμφανίζοντας ανάλογο μήνυμα στην κονσόλα.
6. Mέσω της line = adjust\_coordinats\_to\_write(position,κατεύθυνση) αποθηκεύεται στην μεταβλητή line μια λίστα με αλφαριθμητικά που έπειτα περνά και αποθηκεύεται στην μεταβλητή movehistory η οποία κρατά το ιστορικό των κινήσεων που έχουν γίνει.
7. Καλείται ξανά η find\_solutuon\_DFS για το επόμενο βήμα.
8. Αν η τιμή του counter γίνει -1 από το βήμα 7, τότε φτάσαμε σε λύση του προβλήματος και τερματίζεται το for item in game\_board .
9. Διαφορετικά σημαίνει ότι φτάσαμε σε αδιέξοδο και διαγράφεται από τη movehistory η τελευταία κίνηση που της καταχωρήθηκε ( η line ) .

Αν κανένα από τα 4 if σε ένα κελί δεν βγει αληθές τότε απλά ελέγχουμε το αμέσως επόμενο κελί του. Αν στο τέλος του for item in game\_board η τιμή του counter είναι ίδια με την αρχική του τότε σημαίνει πως η κίνηση αυτή έφτασε σε αδιέξοδο και επιστρέφει τον counter-1 επειδή όταν επιλέξαμε αυτό το βήμα αυξήσαμε τον counter κατά 1.

# Επαλήθευση λύσης – verify\_solution.py

Το σκεπτικό αυτού του προγράμματος είναι απλό. Δέχεται το αρχείο με το πρόβλημα, το αρχείο με την λύση και τα απαιτούμενα βήματα που υπολογίστηκαν και εφαρμόζει τα βήματα αυτά στο πρόβλημα προκειμένου να δει αν φτάνει σε λύση, αν δηλαδή στο τέλος τα ταμπλό έχει μόνο έναν πάσσαλο. Χρησιμοποιεί τις 2 συναρτήσεις που χρησιμοποιεί και το pegsol.py για την ανάγνωση του αρχείου του προβλήματος ( readfile(gameboard) ) και τον έλεγχο αν φτάσαμε σε λύση ( checkforsolution(game\_board) ). Στη συνέχεια για κάθε μετακίνηση πασσάλου, τσεκάροντας την αρχική και την τελική του θέση, εντοπίζει ποια κατεύθυνση ακολούθησε προκειμένου να κάνει τον ενδιάμεσο πάσσαλο 1. Για παράδειγμα αν ξέρει πως από το (2,2) ο πάσσαλος πήγε στο (2,0), καταλαβαίνει πως έγινε κίνηση προς αριστερά και έτσι κάνει το κελί (2,1) ίσο με άσσο. Έπειτα κάνει τις ανάλογες αλλαγές και στα κελιά (2,2) και (2,0) και συνεχίζει για το επόμενο δοσμένο βήμα. Στο τέλος εκτυπώνει στην κονσόλα αν η λύση που του δόθηκε λύνει ή όχι το πρόβλημα που επίσης του δόθηκε.

# Εκτελέσεις των 2 αλγορίθμων

Παρακάτω παρουσιάζονται τα 16 αρχεία που χρησιμοποιήθηκαν για τον έλεγχο των αλγορίθμων και οι χρόνοι που χρειάστηκαν στον καθένα. Με το όρισμα stopped χαρακτηρίζονται όσα ξεπέρασαν το χρονικό όριο των 5 λεπτών και τα νούμερα μετρούν δευτερόλεπτα. Κάθε αρχείο έχει εκτελεστεί 3 φορές με τον κάθε αλγόριθμο και τα νούμερα που παρουσιάζονται είναι ο μέσος όρος αυτών των εκτελέσεων. Τέλος πρέπει να αναφερθεί πως οι δοκιμές έγιναν με την χρήση του source file pegsol.py και όχι με το εκτελέσιμο pegsol.exe καθώς το 2ο ήταν αρκετά πιο αργό λόγο του τύπου του και συνεπώς κρίθηκε πιο χρήσιμο να χρησιμοποιηθεί το .py αρχείο . Χρησιμοποιήθηκαν επεξεργαστής Intel i5 7400 3,00 GHz και RAM Corsair Vengeance 16 GB DDR4-2400 MHz.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Όνομα αρχείου** | **Χρόνος εκτέλεσης DFS** | **Χρόνος εκτέλεσης BFS** | **Αριθμός πασσάλων** | **Αριθμός ελεύθερων θέσεων** |
| input\_5x5.txt | Stopped | Stopped | 16 | 1 |
| input\_5x5\_b.txt | 0,007 | 0,020 | 14 | 3 |
| input\_5x6.txt | 0,005 | 0,022 | 12 | 10 |
| input\_5x7.txt | 13,754 | 0,031 | 17 | 10 |
| input\_6x5.txt | 2,361 | 0,040 | 21 | 1 |
| input\_6x5\_b.txt | 2,279 | 0,027 | 21 | 1 |
| input\_6x6.txt | 130,539 | 9,001 | 16 | 10 |
| input\_6x7.txt | 64.841 | 0.628 | 16 | 10 |
| input\_7x5.txt | Stopped | Stopped | 24 | 1 |
| input\_7x5\_b.txt | Stopped | Stopped | 20 | 5 |
| input\_7x5\_c.txt | 1,572 | 0,166 | 17 | 10 |
| input\_7x6.txt | 39,844 | 0,205 | 15 | 10 |
| input\_7x7.txt | 10,895 | Stopped | 32 | 1 |
| input\_7x7\_b.txt | Stopped | Stopped | 26 | 7 |
| input\_7x7\_c.txt | Stopped | Stopped | 21 | 12 |
| input\_7x7\_d.txt | Stopped | 7,760 | 20 | 13 |
| Συνολικός χρόνος εκτέλεσης αλγορίθμου | 266,097 | 17,9 |  |  |

Αρχικά, είναι εμφανέστατο από τα δεδομένα, πως η βελτίωση που φέρει η ευρετική συνάρτηση του BFS είναι αρκετά σημαντική, ειδικά όσο μεγαλώνει το μέγεθος του πίνακα. Στους πίνακες 5x5\_b και 5x5\_c ο DFS εμφανίζεται γρηγορότερος καθώς οι πάσσαλοι είναι λίγοι και τελικά οι χρόνοι υπολογισμού των τιμών της ευρετικής συνάρτησης και η χρήση τους επιβαρύνει ελάχιστα το πρόγραμμα.

Επιπλέον, έγιναν έλεγχοι για το πώς επηρεάζεται ο χρόνος εκτέλεσης από το μέγεθος του πίνακα και το πλήθος των πασσάλων και των κενών. Όπως μπορεί πολύ εύκολα να διακρίνει κανείς δεν επηρεάζεται εξολοκλήρου από έναν παράγοντα αλλά από συνδυασμό αυτών. Δοκιμάζοντας διάφορες παραλλαγές και συνδυασμούς αυτών των παραγόντων ( σταθερές διαστάσεις, σταθερό πλήθος πασσάλων ή ελεύθερων θέσεων κτλ ) καταλήγουμε στον συμπέρασμα πως σημαντικό ρόλο παίζει και η διάταξη και η θέση που έχουν τοποθετηθεί οι πάσσαλοι και οι ελεύθερες θέσεις. Για παράδειγμα:

* Από τον πίνακα αυτό μπορούμε να παρατηρήσουμε πως όσο μεγαλώνει το μέγεθος του ταμπλό τόσο πιο αργοί γίνονται οι αλγόριθμοι γεγονός που είναι λογικό. Αυτό δεν σημαίνει ωστόσο ότι είναι ο μοναδικός παράγοντας επιρροής καθώς και μικροί πίνακες ήταν αργοί. Όπως για παράδειγμα για τον 5x5 που δεν βρέθηκε λύση ή στο 5χ7 που ήταν πιο αργός από τον 7x5 και από το 7x7.
* Στις δοκιμές δημιουργήθηκαν αρχεία με ελάχιστες διαφορές προκειμένου να μελετηθούν οι πιθανές βελτιώσεις που μπορούν να υπάρξουν . Για παράδειγμα οι πίνακες 6x5 και 6x5\_b έχουν ως διαφορά ότι η 3η γραμμή του 1ου μετατέθηκε στην 5η δημιουργώντας τον b. Αυτή η τόσο μικρή αλλαγή , όπου δεν άλλαξε το πλήθος πασσάλων και κενών θέσεων, είχε ως αποτέλεσμα η εκτέλεση των 2 αλγορίθμων να βελτιωθεί.
* Δοκιμές έγιναν και στις διαστάσεις πάλι κρατώντας σταθερό πλήθος πασσάλων και ελεύθερων θέσεων. Για παράδειγμα οι πίνακες 6x6 και 6x7 όπου ο 2ος χρειάστηκε σχεδόν τον μισό χρόνο στον DFS και μερικά κλάσματα του δευτερολέπτου για τον BFS. Ή οι πίνακες 5x7 και 7x5\_c όπου η αύξηση των γραμμών και η μείωση των στηλών εν τέλει είχε ως αποτέλεσμα να αυξηθεί ο χρόνος εκτέλεσης του BFS και να μην φτάσει σε λύση ο DFS.
* Τέλος κρατώντας το πλήθος των κενών θέσεων σταθερό και αυξάνοντας τις διαστάσεις και το πλήθος των πασσάλων ( πχ τα αρχεία 5x6 , 5x7 , 6x6, 6x7 , 7x5\_x, 7x6) δεν είχαμε ως αποτέλεσμα πάντα να αυξάνονται και οι χρόνοι εκτέλεσης του προγράμματος. Σε παρόμοιο συμπέρασμα οδηγηθήκαμε, κρατώντας σταθερό το πλήθος των πασσάλων και μεταβάλλοντας τις διαστάσεις, το πλήθος και τη θέση των ελεύθερων θέσεων.

# Παράρτημα

## 1. Συναρτήσεις προγραμμάτων:

Ακολουθούν μερικά σχόλια από τα προγράμματα, στα αγγλικά, για τις συναρτήσεις που χρησιμοποιήθηκαν.

### 1.1 pegsol.py

def readfile(gameboard):  
 # This function reads the data from the input file and stores them in a dictionary  
 # Dictionary board has: key : x,y and value 0, 1 or 2  
 # 0 -> invalid position  
 # 1 -> position with peg  
 # 2 -> position without peg  
 # 0 <= x <= width - 1  
 # 0 <= y <= height - 1  
 # inputs : game\_board -> an empty dictionary  
 # output : returns a dictionary created by file's data

def checkforsolution(game\_board):  
 # This function checks if we have found a solution  
 # inputs : game\_board : the current game board  
 # output : true if we have solve the problem , else return false

def calculate\_pegs\_area(game\_board):  
 # This function finds the corners of the orthogonal area which surrounds the pegs  
 # inputs : game\_board : the current game board  
 # output : returns the area

def heuristic(game\_board, peg\_position, direction):  
 # This function calculates the heuristic value for a possible move  
 # inputs : game\_board : the current game board  
 # : peg\_position : The pegs' coordinates that we will move  
 # : direction : The direction which the peg will take  
 # output : Returns the orthogonal area multiplied by manhattan distance and multiplied by  
 # the remained number of pegs

def valid\_move(game\_board, item, direction):  
 # This function check if a peg at position 'item' can move to a specific direction  
 # inputs : game\_board : the current game board  
 # : item : The pegs' coordinates that we will move  
 # : direction : The direction which the peg will take  
 # output : Returns True if we can make that move , else returns False

def make\_move(game\_board, position, direction):  
 # This function return the new board after we move the peg at given position and towards a given direction  
 # inputs : game\_board : the current game board  
 # : position : The pegs' coordinates that we will move  
 # : direction : The direction which the peg will take  
 # output : Returns the new game board after peg's movement

def get\_possible\_moves(game\_board):  
 # This function detects all possible moves  
 # inputs : game\_board : the current game board  
 # output : Returns a dictionary with all possible moves and their direction

def find\_child(game\_board):  
 # This Function finds the node's children  
 # inputs : game\_board : the current game board  
 # output : Return node's children

def adjust\_coordinats\_to\_write(position, direction):  
 # This function is used to adjust the coordinates for our output file.  
 # It adjusts our coordinates increasing them by 1  
 # inputs : position : peg's coordinates  
 # : direction: peg's movement direction  
 # output : Returns a string with the peg's adjusted coordinates before and after its movement

def find\_solution\_BFS(game\_board, counter):  
 # This function uses BFS algorithm to find the moves that we need to solve our problem.  
 # inputs : game\_board : the current game board  
 # : counter : the current counter's value. At the beginning of our program it is equal with zero  
 # output : Returns the new counter's value . If we reach a dead end path, it will returns the value that  
 # it got when we called this function

def find\_solution\_DFS(game\_board, counter):  
 # This function uses DFS algorithm to find the moves that we need to solve our problem.  
 # inputs : game\_board : the current game board  
 # : counter : the current counter's value. At the beginning of our program it is equal with zero  
 # output : Returns the new counter's value . If we reach a dead end path, it will returns the value that  
 # it got when we called this function

### 1.2 verify\_solution.py

def readfile(gameboard):  
 # This function reads the data from the input file and stores them in a dictionary  
 # Dictionary board has: key : x,y and value 0, 1 or 2  
 # 0 -> invalid position  
 # 1 -> position with peg  
 # 2 -> position without peg  
 # 1 <= x <= width  
 # 1 <= y <= height  
 # inputs : game\_board -> an empty dictionary  
 # output : returns a dictionary created by file's data

def read\_solution():  
 # This function reads the data from the input file and stores them in a list  
 # output : returns the number of moves and a list containing those moves

def adjust\_coordinates\_to\_write(solution\_board):  
 # This function is used to adjust the coordinates for our input file.  
 # It adjusts our coordinates reducing them by 1  
 # After this function coordinates values are :  
 # 0 <= x <= width - 1  
 # 0 <= y <= height - 1  
 # inputs : solution\_board : a list with our moves. Every value represents a coordinate

def verify\_solution(game\_board, moves, history):  
 # This function is used to check if we have been given a correct solution  
 # inputs : game\_board : the initial game board  
 # : moves = the number of moves that had been made to solve the problem  
 # : history : a list with our moves  
 # output : Prints a message to console whether a solution it correct or not

def checkforsolution(game\_board):  
 # This function checks if we have found a solution  
 # inputs : game\_board : the current game board  
 # output : true if we have solve the problem , else return false

## 2. Αρχεία με δοκιμών προγράμματος

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input\_5x5.txt | Input\_5x5\_b.txt | Input\_5x6.txt | Input\_5x7.txt | Input\_6x5.txt |
| 5 5  0 1 1 1 0  0 1 1 1 0  1 1 2 1 1  0 1 1 1 0  0 1 1 1 0 | 5 5  0 1 1 1 0  0 1 1 1 0  2 1 2 1 2  0 1 1 1 0  0 1 1 1 0 | 5 6  0 2 1 1 2 0  0 2 1 1 2 0  2 1 1 1 1 2  0 2 1 1 2 0  0 2 1 1 2 0 | 5 7  0 2 1 1 1 2 0  0 2 1 1 1 2 0  2 1 1 1 1 1 2  0 2 1 1 1 2 0  0 2 1 1 1 2 0 | 6 5  0 1 1 1 0  0 1 1 1 0  0 1 1 1 0  1 1 2 1 1  1 1 1 1 1  0 1 1 1 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input\_6x5\_b.txt | Input\_6x6.txt | Input\_6x7.txt | Input\_7x5.txt | Input\_7x5\_b.txt |
| 6 5  0 1 1 1 0  0 1 1 1 0  1 1 2 1 1  1 1 1 1 1  0 1 1 1 0  0 1 1 1 0 | 6 6  0 0 2 2 2 0  0 1 1 1 1 0  2 1 1 1 1 2  2 1 1 1 1 2  0 1 1 1 1 0  0 0 2 2 2 0 | 6 7  0 0 2 2 2 0 0  0 0 1 1 1 0 0  2 1 1 1 1 1 2  2 1 1 1 1 1 2  0 0 1 1 1 0 0  0 0 2 2 2 0 0 | 7 5  0 1 1 1 0  0 1 1 1 0  0 1 1 1 0  1 1 2 1 1  1 1 1 1 1  0 1 1 1 0  0 1 1 1 0 | 7 5  0 1 1 1 0  0 1 1 1 0  0 1 1 1 0  2 1 2 1 2  2 1 1 1 2  0 1 1 1 0  0 1 1 1 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input\_7x5\_c.txt | Input\_7x6.txt | Input\_7x7.txt | Input\_7x7\_b.txt | Input\_7x7\_6.txt |
| 7 5  0 2 2 2 0  0 1 1 1 0  2 1 1 1 2  1 1 1 1 1  2 1 1 1 2  0 1 1 1 0  0 2 2 2 0 | 7 6  0 0 2 2 0 0  0 0 1 1 0 0  2 1 1 1 1 2  2 1 1 2 1 2  2 1 1 1 1 2  0 0 1 1 0 0  0 0 2 2 0 0 | 7 7  0 0 1 1 1 0 0  0 0 1 1 1 0 0  1 1 1 1 1 1 1  1 1 1 2 1 1 1  1 1 1 1 1 1 1  0 0 1 1 1 0 0  0 0 1 1 1 0 0 | 7 7  0 0 1 1 1 0 0  0 0 1 1 1 0 0  2 1 1 1 1 1 2  2 1 1 2 1 1 2  2 1 1 1 1 1 2  0 0 1 1 1 0 0  0 0 1 1 1 0 0 | 7 7  0 0 2 2 2 0 0  0 0 1 1 1 0 0  2 1 1 1 1 1 2  2 1 1 1 1 1 2  2 1 1 1 1 1 2  0 0 1 1 1 0 0  0 0 2 2 2 0 0 |

|  |
| --- |
| Input\_7x7\_d.txt |
| 7 7  0 0 2 2 2 0 0  0 0 1 1 1 0 0  2 1 1 1 1 1 2  2 1 1 2 1 1 2  2 1 1 1 1 1 2  0 0 1 1 1 0 0  0 0 2 2 2 0 0 |