

ΕΘΝΙΚΟ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

Καθηγητής : *Κουμπάρακης Μανόλης*

Ημ/νία παράδοσης: *09/12/2011*

Όνομ/μο φοιτητή : *Μπεγέτης Νικόλαος*
A.M.: *1115200700281*

Σχόλια και παρουσίαση αποτελεσμάτων για Πρόβλημα 1:

Όπως ζητείται κατά την εκκίνηση του προγράμματος ορίζουμε ένα αντικείμενο *game* το οποίο κατά την αρχικοποίηση του λαμβάνει τις εισόδους για τα δεδομένα μέσω ενός εκ των αρχείων *input.txt*, *input1.txt*, και *input2.txt* τα οποία περιέχουν το πλέγμα.

Έπειτα αφού γίνει η αρχικοποίηση του βασικού αντικειμένου του προβλήματος ο χρήστης καλείται να επιλέξει ανάμεσα σε 5 ευρεστικές συναρτήσεις για να βρει βέλτιστη λύση στο παιχνίδι (απλή *manhattan*, βελτιωμένη *manhattan*, *ευκλείδεια*, *τετραγωνική ευκλείδεια-χωρίς τον υπολογισμό ρίζας*- και *διαγώνια*). Αφού επιλέξει ο χρήστης ξεκινάει η χρονομέτρηση και σταματάει μόλις θα έχει βρει βέλτιστη λύση χρησιμοποιώντας τον A^* αλγόριθμο. Έπειτα εκτυπώνονται τα αποτελέσματα:

- *συνολικοί κόμβοι που εξερευνήθηκαν,*
- *μονοπάτι καταστάσεων βέλτιστης λύσης και περιγραφικός τρόπος κινήσεων*
- *κόστος λύσης*
- *συνολικός χρόνος εκτέλεσης αλγορίθμου.*

Στην κλάση *Game* που δέχεται ως όρισμα το πρόβλημα το οποίο χρησιμοποιεί ο αλγόριθμος A^* γίνονται *override* οι συναρτήσεις *actions*, *result* και *path_cost* του αρχείου *search.py* τις οποίες έχουμε ξανα-ορίζει εμείς στο αρχείο *problem1_1.py* .

Η συνάρτηση *actions* επιστρέφει τις έγκυρες ενέργειες-κινήσεις που μπορούν να επιτευχθούν από κάθε θέση κόμβου. Αυτές οι ενέργειες είναι οι “*up*”, “*down*”, “*right*” και “*left*” και έχουν οριστεί έτσι ώστε να είναι πιο αντιληπτές στο χρήστη.

Η συνάρτηση *result* επιστρέφει την επόμενη θέση κόμβου από τον οποίο πρέπει να περάσει το ανθρωπάκι του παιχνιδιού ανάλογα με την ενέργεια-κίνηση που έγινε κατά την εκτέλεση του αλγορίθμου A^* .

Η συνάρτηση *path_cost* επιστρέφει το κόστος ανάμεσα στην τρέχουσα και στην επόμενη θέση κόμβου απ’ όπου πρέπει να περάσει το ανθρωπάκι σύμφωνα με τον αλγόριθμο A^* .

Χρησιμοποιήθηκαν 5 ευρεστικές συναρτήσεις που θεωρούνται ως οι πιο γνωστές για πλέγματα χαρτών σύμφωνα με το πανεπιστήμιο [Stanford](http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html).

- Η ευρεστική συνάρτηση *manhattan_simple* υπολογίζει την απόσταση των κόμβων από τις τρέχουσες θέσεις τους προς την κατάσταση στόχου.
- Η ευρεστική συνάρτηση *manhattan_improved* υπολογίζει την απόσταση των κόμβων από τις τρέχουσες θέσεις τους προς την κατάσταση στόχου χρησιμοποιώντας επιπλέον έναν πολλαπλασιαστικό παράγοντα που επιλέχθηκε με βάση το μέγεθος του μέγιστου προβλεπόμενου μήκους μονοπατιού ($1 / \langle \text{expected maximum path cost} \rangle$). Αυτό βελτιώνει την ευρεστική συνάρτηση γιατί με αυτό τον τρόπο ο αλγόριθμος προτιμά να εξερευνά κόμβους που βρίσκονται όλο και πιο κοντά στον κόμβο στόχου, αφού το κόστος θα μεγαλώνει συνεχώς και έτσι προτιμώνται οι τιμές των καταστάσεων που είναι πιο κοντά στο στόχο.
- Η ευρεστική συνάρτηση *euclidean* υπολογίζει την ευκλείδεια απόσταση των κόμβων από τις τρέχουσες θέσεις τους προς την κατάσταση στόχου, θεωρώντας ότι επιτρέπονται και οι διαγώνιες κινήσεις. Με αυτό τον τρόπο πολλές παγίδες και εμπόδια μπορούν θεωρητικά να ξεπερνιούνται. Έτσι η ευρεστική συνάρτηση θεωρητικά θα βρίσκει μία κοντινότερη απόσταση από τη συνάρτηση *manhattan*, πρακτικά όμως αφού δεν επιτρέπονται οι διαγώνιες κινήσεις αν και μπορεί τελικά να βρεθούν καλύτερα μονοπάτια μιας αφού εξερευνούνται περισσότεροι κόμβοι αυτό θα έχει και αρνητική επίπτωση στο χρόνο αν υποθέσουμε ότι ο χρόνος εξαρτάται μόνο από το πλήθος των κόμβων που πρέπει να εξερευνηθούν.
- Η ευρεστική συνάρτηση *euclidean_squared* υπολογίζει την τετραγωνική ευκλείδεια απόσταση των κόμβων από τις τρέχουσες θέσεις τους προς την κατάσταση στόχου, χωρίς να χρησιμοποιεί το 'χρονοβόρο' υπολογισμό της sqrt . Η χρήση αυτής της συνάρτησης σε πλέγματα χωρίς πολλά και μεγάλα εμπόδια πολλές φορές αποβαίνει ταχύτερη από την απλή ευκλείδεια ευρεστική, όμως σε περίπτωση που υπάρχουν πολλά εμπόδια, δεν συνίσταται γιατί κατανατάει να λειτουργεί σαν τον άπληστο αλγόριθμο με άσχημες επιπτώσεις στο χρόνο και στο μονοπάτι που αναζητείται.
- Η ευρεστική συνάρτηση *diagonal* γνωστή και ως *Chebyshev* υπολογίζει την καλύτερη διαγώνια απόσταση των κόμβων από τις τρέχουσες θέσεις τους προς την κατάσταση στόχου και την απόσταση *manhattan*, και έπειτα συνδυάζει τις παραπάνω δύο.
- Περισσότερες λεπτομέρειες για τις παραπάνω ευρεστικές συναρτήσεις μπορούν να βρεθούν στο σύνδεσμο:
<http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>

Ακολουθούν 5 εκτυπώσεις για το πλέγμα που μας δόθηκε με καθεμιά ευρεστική συνάρτηση:

```
Python Shell
File Edit Shell Debug Options Windows Help

>>> ===== RESTART =====
>>>
Type the name of the file with the grid(eg."input.txt") : "input.txt"
The game map is:

00000000
02220000
02400000
00111110
01000000
00132010
00002000
00000000

The initial and goal positions are :
((3, 5), (2, 2))
Switch from the following 3 heuristic functions :
1. manhattan_simple
2. manhattan_improved
3. euclidean
4. euclidean_squared
5. diagonal

Type the heuristic function you want to use : 1
Manhattan simple heuristic algorithm is selected

Total nodes explored : 25
Game solution:

||No. ||State ||Action ||Cost ||
||=====||
|| 0 || (3, 5) || || 0 ||
|| 1 || (3, 6) || down || 1 ||
|| 2 || (2, 6) || left || 2 ||
|| 3 || (1, 6) || left || 3 ||
|| 4 || (1, 5) || up || 4 ||
|| 5 || (0, 5) || left || 5 ||
|| 6 || (0, 4) || up || 6 ||
|| 7 || (0, 3) || up || 7 ||
|| 8 || (1, 3) || right || 8 ||
|| 9 || (1, 2) || up || 10 ||
|| 10 || (2, 2) || right || 11 ||
||=====||

Execution time : 0.026 seconds
>>> |
```

```
Python Shell
File Edit Shell Debug Options Windows Help

|| 10 || (2, 2) || right || 11 ||
||=====||

Execution time : 0.026 seconds
>>> ===== RESTART =====
>>>
Type the name of the file with the grid(eg."input.txt") : "input.txt"
The game map is:

00000000
02220000
02400000
00111110
01000000
00132010
00002000
00000000

The initial and goal positions are :
((3, 5), (2, 2))
Switch from the following 3 heuristic functions :
1. manhattan_simple
2. manhattan_improved
3. euclidean
4. euclidean_squared
5. diagonal

Type the heuristic function you want to use : 2
Improved Manhattan heuristic algorithm is selected

Total nodes explored : 24
Game solution:

||No. ||State ||Action ||Cost ||
||=====||
|| 0 || (3, 5) || || 0 ||
|| 1 || (3, 6) || down || 1 ||
|| 2 || (2, 6) || left || 2 ||
|| 3 || (1, 6) || left || 3 ||
|| 4 || (1, 5) || up || 4 ||
|| 5 || (0, 5) || left || 5 ||
|| 6 || (0, 4) || up || 6 ||
|| 7 || (0, 3) || up || 7 ||
|| 8 || (1, 3) || right || 8 ||
|| 9 || (1, 2) || up || 10 ||
|| 10 || (2, 2) || right || 11 ||
||=====||

Execution time : 0.035 seconds
>>>
```

```
Python Shell
File Edit Shell Debug Options Windows Help
=====
|| 10 | (2, 2) | right | 11 ||
=====
Execution time : 0.035 seconds
>>> ===== RESTART =====
>>>
Type the name of the file with the grid(eg."input.txt") : "input.txt"
The game map is:

00000000
02220000
02400000
00111110
01000000
00132010
00002000
00000000

The initial and goal positions are :
((3, 5), (2, 2))
Switch from the following 3 heuristic functions :
1. manhattan_simple
2. manhattan_improved
3. euclidean
4. euclidean_squared
5. diagonal

Type the heuristic function you want to use : 3
Euclidean heuristic algorithm is selected

Total nodes explored : 29
Game solution:
=====
||No. |State |Action |Cost ||
=====
|| 0 | (3, 5) | | 0 ||
|| 1 | (3, 6) | down | 1 ||
|| 2 | (2, 6) | left | 2 ||
|| 3 | (1, 6) | left | 3 ||
|| 4 | (1, 5) | up | 4 ||
|| 5 | (0, 5) | left | 5 ||
|| 6 | (0, 4) | up | 6 ||
|| 7 | (0, 3) | up | 7 ||
|| 8 | (1, 3) | right | 8 ||
|| 9 | (1, 2) | up | 10 ||
|| 10 | (2, 2) | right | 11 ||
=====
Execution time : 0.032 seconds
>>>
```

Ln: 132 Col: 4

```
Python Shell
File Edit Shell Debug Options Windows Help
=====
|| 10 | (2, 2) | right | 11 ||
=====
Execution time : 0.032 seconds
>>> ===== RESTART =====
>>>
Type the name of the file with the grid(eg."input.txt") : "input.txt"
The game map is:

00000000
02220000
02400000
00111110
01000000
00132010
00002000
00000000

The initial and goal positions are :
((3, 5), (2, 2))
Switch from the following 3 heuristic functions :
1. manhattan_simple
2. manhattan_improved
3. euclidean
4. euclidean_squared
5. diagonal

Type the heuristic function you want to use : 4
Euclidean squared (without sqrt computation) heuristic algorithm is selected

Total nodes explored : 15
Game solution:
=====
||No. |State |Action |Cost ||
=====
|| 0 | (3, 5) | | 0 ||
|| 1 | (3, 6) | down | 1 ||
|| 2 | (2, 6) | left | 2 ||
|| 3 | (1, 6) | left | 3 ||
|| 4 | (1, 5) | up | 4 ||
|| 5 | (0, 5) | left | 5 ||
|| 6 | (0, 4) | up | 6 ||
|| 7 | (0, 3) | up | 7 ||
|| 8 | (1, 3) | right | 8 ||
|| 9 | (1, 2) | up | 10 ||
|| 10 | (2, 2) | right | 11 ||
=====
Execution time : 0.034 seconds
>>>
```

Ln: 175 Col: 4

```

Python Shell
File Edit Shell Debug Options Windows Help
|| 10 | (2, 2) | right | 11 ||
=====
Execution time : 0.034 seconds
>>> ===== RESTART =====
>>>
Type the name of the file with the grid(eg."input.txt") : "input.txt"
The game map is:

00000000
02220000
02400000
00111110
01000000
00132010
00002000
00000000

The initial and goal positions are :
((3, 5), (2, 2))
Switch from the following 3 heuristic functions :
1. manhattan_simple
2. manhattan_improved
3. euclidean
4. euclidean_squared
5. diagonal

Type the heuristic function you want to use : 5
Diagonal heuristic algorithm is selected

Total nodes explored : 30
Game solution:

||No. |State |Action |Cost ||
=====
|| 0 | (3, 5) | | 0 ||
|| 1 | (3, 6) | down | 1 ||
|| 2 | (2, 6) | left | 2 ||
|| 3 | (1, 6) | left | 3 ||
|| 4 | (1, 5) | up | 4 ||
|| 5 | (0, 5) | left | 5 ||
|| 6 | (0, 4) | up | 6 ||
|| 7 | (0, 3) | up | 7 ||
|| 8 | (1, 3) | right | 8 ||
|| 9 | (1, 2) | up | 10 ||
|| 10 | (2, 2) | right | 11 ||
=====
Execution time : 0.045 seconds
>>> |
Ln: 218 Col: 4

```

Όπως φαίνεται από τις παραπάνω εκτυπώσεις και οι 5 ευρεστικές συναρτήσεις βρήκαν τη βέλτιστη λύση που έχει κόστος 11 και το μονοπάτι αποτελείται από 11 κόμβους συμπεριλαμβανομένων του αρχικού και του κόμβου στόχου. Εδώ αυτό που αξίζει να προσέξουμε είναι ότι σε αυτό το πλέγμα η καλύτερη ευρεστική συνάρτηση ως προς το χρόνο εκτέλεσης είναι η ευρεστική *manhattan_simple* ακολουθούμενη από την *euclidean* ενώ η καλύτερη ευρεστική συνάρτηση ως προς την ‘ευφυΐα’ της εξερεύνησης είναι η *euclidean_squared* ακολουθούμενη από την *manhattan_improved*. Παρατηρούμε ότι η ευρεστική που εξερεύνησε τους περισσότερους κόμβους και επακολούθως καθυστέρησε και περισσότερο είναι η *diagonal*.

Σημείωση: Όλες οι ευρεστικές συναρτήσεις έχουν τον πολλαπλασιαστικό παράγοντα που είναι 0.05 υποθέτοντας ότι το προβλεπόμενο μονοπάτι μπορεί να βρεθεί περίπου σε 20 εξερευνήσεις. Η πραγματική όμως χρήση των ευρεστικών προϋποθέτει ότι δεν χρειάζεται απαραίτητα να χρησιμοποιείται σε όλες τις ευρεστικές συναρτήσεις ο ίδιος πολ/στικός παράγοντας.

Παρακάτω παρατίθεται πίνακας με τους χρόνους εκτελέσεων και τους εξερευνημένους κόμβους των παραπάνω ευρεστικών συναρτήσεων:

| Heuristic function used | Total nodes explored | Algorithm execution time |
|-------------------------|----------------------|--------------------------|
| Manhattan_simple | 25 | 0.026 sec |
| Manhattan_improved | 24 | 0.035 sec |
| Euclidean | 29 | 0.032 sec |
| Euclidean_squared | 15 | 0.034 sec |
| Diagonal | 30 | 0.045 sec |

Για να αποδείξουμε ότι η καλύτερη και πιο αποδοτική ευρεστική συνάρτηση ουσιαστικά δεν υπάρχει αλλά πάντα εξαρτάται από το μέγεθος και τη μορφή του πλέγματος, θα τρέξουμε άλλα 2 παραδείγματα όπου στο ένα θα χρησιμοποιήσουμε ένα πολύ μεγάλο και πολύπλοκο πλέγμα όπως πρότεινε ένας συμφοιτητής στη λίστα του μαθήματος και ένα ακόμα μικρό πλέγμα με λίγα εμπόδια και αρκετές παγίδες.

Για το “*input1.txt*” έχουμε τα εξής αποτελέσματα όπως εμφανίστηκαν στις εκτυπώσεις:

| <i>Heuristic function used</i> | <i>Total nodes explored</i> | <i>Algorithm execution time</i> | <i>Size (and cost) of path followed</i> |
|--------------------------------|-----------------------------|---------------------------------|---|
| <i>Manhattan_simple</i> | 8188 | 2.343 sec | 433(472) |
| <i>Manhattan_improved</i> | 8172 | 2.363 sec | 433(472) |
| <i>Euclidean</i> | 8233 | 2.022 sec | 433(472) |
| <i>Euclidean_squared</i> | 4531 | 2.008 sec | 529(591) |
| <i>Diagonal</i> | 8247 | 2.012 sec | 433(472) |

Όπως παρατηρούμε εδώ τα αποτελέσματα έχουν αρκετά διαφορετική μορφή από το προηγούμενο πλέγμα που χρησιμοποιήσαμε. Κάποια αξιοσημείωτα είναι τα εξής:

1. Η σχέση μεταξύ κόμβων που εξερευνήθηκαν και κόμβων του μονοπατιού είναι 16:1, που σημαίνει ότι αν 16 κόμβους που εξερευνούνταν μόνο ο ένας ανήκε στο μονοπάτι. Αντίστοιχα στο προηγούμενο πλέγμα “*input.txt*” η αναλογία ήταν περίπου 2:1 ή 3:1. Από αυτά μπορούμε να συμπεράνουμε ότι το δεύτερο πλέγμα είναι πολύ πιο πολύπλοκο από το πρώτο, αφού στο πρώτο σχεδόν σε 2 κόμβους ήμασταν σίγουροι ότι ο ένας ανήκε στο βέλτιστο μονοπάτι.
2. Ένα ακόμα αξιοσημείωτο πριν περάσουμε στη σύγκριση των αλγορίθμων είναι ότι αυτή τη φορά δεν βρήκαν και οι 5 αλγόριθμοι βέλτιστη λύση. Συγκεκριμένα η ευρεστική *euclidean_squared* βρήκε ως βέλτιστο ένα μονοπάτι με μέγεθος 529 και κόστος 591, που όμως δεν είναι βέλτιστο όπως φαίνεται από τα αποτελέσματα των υπόλοιπων τεσσάρων ευρεστικών. Το λάθος αυτό μπορούμε να το βασίσουμε σε αλληλένδετους 2 λόγους. Πρώτον, η συγκεκριμένη ευρεστική για να επιτύχει καλύτερη ταχύτητα υπολόγιζε την τετραγωνική απόσταση των σημείων και όχι την τετραγωνική, με αποτέλεσμα να εξερευνήσει λιγότερους κόμβους και δεύτερον η συγκεκριμένη ευρεστική ελέγχει τα μονοπάτια προς όλες τις κατευθύνσεις (και τις διαγώνιες) και αυτό όπως γράψαμε και παραπάνω στην περιγραφή της μπορεί να λειτουργήσει αρνητικά όταν οι μόνες επιτρεπτές κατευθύνσεις είναι οι οριζόντιες και οι κάθετες. Συνεπώς καλό είναι να αποφεύγεται η χρήση της σε μεγάλα και πολύπλοκα πλέγματα, αφού η εύρεση βέλτιστης λύσης είναι προτιμότερη από την εξοικονόμηση χρόνου.
3. Περνώντας πλέον στην σύγκριση των ευρεστικών παρατηρούμε ότι σε αυτό το πλέγμα η καλύτερη ευρεστική συνάρτηση ως προς το χρόνο εκτέλεσης αυτή τη φορά είναι η ευρεστική *euclidean_squared* και αυτό οφείλεται κυρίως όπως γράψαμε παραπάνω στο ότι εξερεύνησε λιγότερους κόμβους. Επειδή όμως δεν είναι βέλτιστη η ταχύτερη ευρεστική που προσφέρει βέλτιστη λύση με πολύ μικρή διαφορά από την *euclidean_squared* αυτή τη φορά είναι η *diagonal* η οποία αξίζει να σημειωθεί ότι στο προηγούμενο πλέγμα ήταν η χειρότερη και στο χρόνο και στην ‘ευφυΐα’. Ως προς την επιλογή των κόμβων

που εξερευνήθηκαν η πιο ευφυής φαίνεται ότι είναι και πάλι η *euclidean_squared* αλλά όπως είπαμε τελικά δεν δίνει βέλτιστη λύση οπότε η ευφυέστερη ευρετική που δίνει βέλτιστη λύση είναι η *manhattan_improved* ακολουθούμενη από την *manhattan_simple*. Εδώ παρατηρούμε ότι οι παραπάνω δύο ευρετικές παρόλο που είναι οι δύο ευφυέστερες είναι και οι δύο χειρότερες ως προς το χρόνο εκτέλεσης.

Να σημειωθεί πως για δύσκολα πλέγματα με πολλούς εσωτερικούς λαβυρίνθους και με επιτρεπόμενες κινήσεις μόνο τις οριζόντιες και τις κάθετες οι ευρεστικές *manhattan* είναι οι καλύτερες.

Τέλος για το “*input2.txt*” έχουμε τα εξής αποτελέσματα όπως εμφανίστηκαν στις εκτυπώσεις:

| <i>Heuristic function used</i> | <i>Total nodes explored</i> | <i>Algorithm execution time</i> | <i>Size (and cost) of path followed</i> |
|--------------------------------|-----------------------------|---------------------------------|---|
| <i>Manhattan_simple</i> | 22 | 0.009 sec | 08(13) |
| <i>Manhattan_improved</i> | 20 | 0.009 sec | 08(13) |
| <i>Euclidean</i> | 25 | 0.009 sec | 08(13) |
| <i>Euclidean_squared</i> | 10 | 0.009 sec | 08(13) |
| <i>Diagonal</i> | 25 | 0.009 sec | 08(13) |

Όπως παρατηρούμε εδώ τα αποτελέσματα είναι σχεδόν όλα ίδια με πολύ μικρές διαφορές. Το αξιοσημείωτο εδώ είναι ότι ενώ το μονοπάτι αποτελείται από 8 κόμβους το εύρος των κόμβων που εξερευνήθηκαν από τις ευρεστικές είναι 15 κόμβοι(10 - 25) δηλαδή 2 φορές επιπλέον το εύρος του μονοπατιού ενώ όλες έκαναν τον ίδιο χρόνο. Αυτό συνέβη γιατί οι περισσότερες ευρεστικές ακολουθούσαν 3 διαφορετικά μονοπάτια(~24 κόμβοι) μέχρι να αποφασίσουν το βέλτιστο(*manhattan_simple*, *manhattan_improved*) ή μέχρι να πέσουν πάνω στον κόμβο στόχο(*euclidean*, *diagonal*), ενώ η *euclidean_squared* αν και φέρθηκε άπληστα και ακολούθησε μόνο ένα μονοπάτι τελικά είχε το ίδιο αποτέλεσμα. Όμως αυτή τη φορά δεν είχε κάποιο πλεονέκτημα στο χρόνο εκτέλεσης.