ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ
«ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ»

ΠΡΟΒΛΗΜΑ

ΙΕΡΑΠΟΣΤΟΛΩΝ-ΚΑΝΙΒΑΛΩΝ

ΑΛΓΟΡΙΘΜΟΙ

(A*, BF)

ΟΜΑΔΑ

ΜΑΝΕΤΤΑ ΝΙΚΟΛΙΤΣΑ (Α.Μ.: 3967)

ΧΙΩΤΗΣ ΓΕΩΡΓΙΟΣ (Α.Μ.: 4038)

Πάτρα 2009-2010

1. ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ.

Υπάρχουν τρεις ιεραπόστολοι., τρεις κανίβαλοι και μια βάρκα στη μια όχθη ενός ποταμού. Θέλουμε όλοι να μεταφερθούν στην απέναντι όχθη με τη βάρκα. Όμως, η βάρκα για να κινηθεί χρειάζεται τουλάχιστον ένα άτομο, χωρά μόνο μέχρι δυο άτομα και σε καμία περίπτωση δεν πρέπει ο αριθμός των κανιβάλων να είναι μεγαλύτερος από αυτόν των ιεραποστόλων σε κάποια όχθη.

2. ΑΝΑΠΑΡΑΣΤΑΣΗ ΚΑΤΑΣΤΑΣΗΣ.

Η αναπαράσταση μιας τυχαίας κατάστασης του προβλήματος μπορεί να είναι η εξής:

(ML,CL,MR,CR,Side)

όπου:
ML: ο αριθμός των ιεραπόστολων στην αριστερή όχθη του ποταμού
CL: ο αριθμός των κανίβαλων στην αριστερή όχθη του ποταμού
MR: ο αριθμός των ιεραπόστολων στην δεξιά όχθη του ποταμού
CR: ο αριθμός των κανίβαλων στην δεξιά όχθη του ποταμού
Side: left ή right ανάλογα με το σε ποια όχθη βρίσκεται η βάρκα.

Στο γενικό πρόβλημα η αρχική κατάσταση είναι:

(3,3,0,0,left)

Και η τελική κατάσταση είναι:

(0,0,3,3,right).

Στον αλγόριθμο best first που έχει γίνει παραμετροποίηση δίνουμε στον χρήστη να καθορίζει αυτός την αρχική κατάσταση του προβλήματος. Όμως σε κάθε περίπτωση θεωρούμε σαν τελική κατάσταση αυτή όπου όλοι οι ιεραπόστολοι και οι κανίβαλοι έχουν μεταφερθεί στην δεξιά όχθη του ποταμού όπου βρίσκεται και η βάρκα.

3. ΠΕΡΙΓΡΑΦΗ ΤΕΛΕΣΤΩΝ ΜΕΤΑΒΑΣΗΣ

Έχουμε δέκα τελεστές μετάβασης για την μετακίνηση των κανιβάλων και των ιεραποστόλων από την αριστερή στην δεξιά όχθη του ποταμού και αντίστροφα. Οι τελεστές είναι αυτοί που φαίνονται στον παρακάτω πίνακα:

ΤΕΛΕΣΤΗΣ	ΠΡΟΥΠΟΘΕΣΕΙΣ	ΑΠΟΤΕΛΕΣΜΑ
T1:Μετέφερε 1 ιεραπόστολο από αριστερά στα δεξιά	ML>0, ML≥CL+1 ή ML=1, Side=left	(CL,ML-1,CR,MR+1,right)
T2:Μετέφερε 1 κανίβαλο από αριστερά στα δεξιά	CL>0, MR>CR ή MR=0 ή CR=0, Side=left	(CL-1,ML,CR+1,MR, right)
T3:Μετέφερε 2 ιεραπόστολους από αριστερά στα δεξιά	ML>1, ML≥CL+2 ή ML=2, Side=left	(CL,ML-2,CR,MR+2, right)
T4:Μετέφερε 2 κανίβαλους από αριστερά στα δεξιά	CL>1, MR>CR+1 ή MR=0, Side=left	(CL-2,ML,CR+2,MR, right)
T5:Μετέφερε 1 ιεραπόστολο και 1 κανίβαλο από αριστερά στα δεξιά	CL>0 και ML>0, MR=0 ή CR>0, Side=left	(CL-1,ML-1,CR+1,MR+1, right)
T6:Μετέφερε 1 ιεραπόστολο από δεξιά στα αριστερά	MR>0, MR≥CR+1 ή MR=1, Side=right	(CL,ML+1,CR,MR-1,left)
T7:Μετέφερε 1 κανίβαλο από δεξιά στα αριστερά	CR>0, ML>CL ή ML=0 ή CL=0, Side=right	(CL+1,ML,CR-1,MR, left)
T8:Μετέφερε 2 ιεραπόστολους από δεξιά στα αριστερά	MR>1, MR≥CR+2 ή MR=2, Side=right	(CL,ML+2,CR,MR-2, left)
T9:Μετέφερε 2 κανίβαλους από δεξιά στα αριστερά	CR>1, ML>CL+1 ή ML=0, Side=right	(CL+2,ML,CR-2,MR, left)
T10:Μετέφερε 1 ιεραπόστολο και 1 κανίβαλο από δεξιά στα αριστερά	CR>0 και MR>0, ML=0 ή CL>0, Side=right	(CL+1,ML+1,CR-1,MR-1, left)

4. ΑΛΓΟΡΙΘΜΟΣ BEST FIRST (ΜΑΝΕΤΤΑ ΝΙΚΟΛΙΤΣΑ)

4.1. Ο αλγόριθμος best first για να υλοποιηθεί χρειάζεται μόνο μια ευρετική συνάρτηση. Σαν ευρετική συνάρτηση χρησιμοποιούμε μια συνάρτηση που μετρά την απόσταση κάθε κατάστασης από τον στόχο(τελική κατάσταση). Οπότε ορίζουμε σαν ευρετική συνάρτηση την

$$h(n)=ML+CL$$

όπου

ML: ο αριθμός των ιεραπόστολων στην αριστερή όχθη του ποταμού

CL: ο αριθμός των κανίβαλων στην αριστερή όχθη του ποταμού.

Ουσιαστικά με την παραπάνω ευρετική βρίσκουμε το άθροισμα των ιεραποστόλων και των κανιβάλων που βρίσκονται στην αριστερή όχθη του ποταμού και επομένως απομένουν ακόμα να μεταφερθούν στην δεξιά όχθη

4.2. Στη συνέχεια ακολουθεί η περιγραφή του αλγορίθμου best first σε φυσική γλώσσα:

1. Βάλε την αρχική κατάσταση στο μέτωπο αναζήτησης.
2. Αν το μέτωπο αναζήτησης είναι κενό σταμάτα (Αποτυχία).
3. Πάρε την πρώτη σε σειρά κατάσταση από το μέτωπο αναζήτησης.
4. Αν η κατάσταση είναι μέλος του κλειστού συνόλου τότε πήγαινε στο 2.
5. Αν η κατάσταση είναι μια τελική ανέφερε τη λύση και σταμάτα (Επιτυχία).
6. Εφάρμοσε τους τελεστές μετάβασης για να παράγεις τις καταστάσεις παιδία
7. Εφάρμοσε την ευρετική συνάρτηση σε κάθε παιδί.
8. Βάλε τις καταστάσεις παιδία στο μέτωπο αναζήτησης
9. Αναδιάταξε το μέτωπο αναζήτησης έτσι ώστε η κατάσταση με την καλύτερη ευρετική τιμή να είναι πρώτη
10. Βάλε την κατάσταση γονέα στο κλειστό σύνολο
11. Πήγαινε στο βήμα 2.

Σύμφωνα με τον παραπάνω αλγόριθμο το κλειστό σύνολο περιέχει τις καταστάσεις που ήδη έχουμε επισκεφτεί ενώ τα παιδία τους

βρίσκονται στο ανοικτό σύνολο. Ουσιαστικά ότι έχουμε στο κλειστό σύνολο δεν το ξανά επισκεπτόμαστε. Αντίθετα στο μέτωπο αναζήτησης είναι όλες οι κορυφές που έχουμε επισπευτεί αλλά δεν έχουμε αναπτύξει ακόμα. Πρόκειται για ένα διατεταγμένο σύνολο

4.3. Ο ψευδοκώδικας για τον αλγόριθμο best first είναι:

```
algorithm bestfs(InitialState, FinalState)
begin
        Closed←Ø;
        EvaluatedInitialState←Heuristic(<InitialState>)
        Frontier←<EvaluatedInitialState>;
        CurrentState←best(Frontier);
        while CurrentState ∉ FinalState      do
                Frontier←delete(CurrentState,Frontier);
                if CurrentState ∉ ClosedSet then
                        begin
                                Children ←Expand(CurrentState);
                                EvaluatedChildren←Heuristic(Children);
                                Frontier←Frontier ^ EvaluatedChildren;
                                Closed←Closed∪{CurrentState};
                        end;
                if Frontier= Ø then exit;
                CurrentState←best(Frontier);
        endwhile;
end.
```

Σύμφωνα με τον παραπάνω ψευδοκώδικα ο αλγόριθμος best first παίρνει σαν ορίσματα την αρχική και την τελική κατάσταση. Αρχικά θέτει το κλειστό σύνολο ίσο με το κενό. Και στη συνέχεια υπολογίζει την ευρετική συνάρτηση για την αρχική κατάσταση, την οποία τοποθετεί στο μέτωπο αναζήτησης. Την αρχική κατάσταση την ορίζει σαν τρέχουσα κατάσταση(CurrentState) και εκτελεί για αυτήν τον αλγόριθμο best first. Για όσο η τρέχουσα κατάσταση είναι διαφορετική από την τελική κατάσταση(κατάσταση στόχος) σβήνει από το μέτωπο αναζήτησης την τρέχουσα κατάσταση. Στη συνέχεια ελέγχει αν η κατάσταση αυτή βρίσκεται στο κλειστό σύνολο. Αν δεν ανήκει στο κλειστό σύνολο βρίσκει τα παιδία της κατάστασης αυτής με βάση τους τελεστές μετάβασης του προβλήματος. Βρίσκει την ευρετική κατάσταση των παιδιών αυτών και τα προσθέτει στο

μέτωπο αναζήτησης κατά αύξουσα σειρά με βάση την ευρετική συνάρτηση, διατηρώντας τις ήδη υπάρχουσες καταστάσεις που βρίσκονται εκεί. Τέλος βάζει την τρέχουσα κατάσταση στο κλειστό σύνολο (χωρίς να σβήσει τις καταστάσεις που ήδη βρίσκονται εκεί). Στη συνέχεια εκτελεί και πάλι τον αλγόριθμο best first για την κατάσταση του μετώπου αναζήτησης με την μικρότερη τιμή ευρετικής συνάρτησης. Η παραπάνω διαδικασία εκτελείται μέχρι η εκάστοτε τρέχουσα κατάσταση να είναι ίδια με την κατάσταση στόχος οπότε έχουμε επιτυχία ή μέχρι το μέτωπο αναζήτησης να μείνει κενό οπότε έχουμε αποτυχία.

4.4. Η υλοποίηση του αλγορίθμου σε prolog ακολουθεί τον ψευδοκώδικα του ερωτήματος 4.3. Πιο συγκεκριμένα καλώντας τον αλγόριθμο best first να εκτελεστεί αρχικά παίρνω την αρχική κατάσταση της οποίας υπολογίζω την ευρετική συνάρτηση και στη συνέχεια εκτελώ τον αλγόριθμο best first για αυτή την κατάσταση, όπως φαίνεται παρακάτω:

```
gobfs(Solution):-          in_state(IS),
                           heuristic(IS,V),
                           bfs([V-[IS]],Solution).
```

Κάθε κατάσταση για την οποία καλώ τον αλγόριθμο best first αρχικά την συγκρίνω με την τελική κατάσταση έτσι ώστε να ελέγχω αν έχω φτάσει στον στόχο οπότε και να τερματίσω με επιτυχία. Αυτό γίνεται από το παρακάτω κομμάτι κώδικα στο οποία gl_state(State) είναι η τελική κατάσταση που είναι πάντα η ίδια. Δηλαδή όλοι οι κανίβαλοι και οι ιεραπόστολοι έχουν μεταφερθεί στην δεξιά όχθη του ποταμού όπου βρίσκεται και η βάρκα.

```
bfs([_-[State|Path]|_],[State|Path]):-    gl_state(State).
```

Αν ο παραπάνω κανόνας δεν μας δώσει true η prolog συνεχίζει εκτελώντας τον αλγόριθμο best first για να φτάσει στην λύση και να μας δώσει true. Ο αλγόριθμος best first εκτελείται με βάση το ακόλουθο κομμάτι κώδικα.

```
bfs([V-[State|Path]|RestPaths],Solution):-
      nl,
      write('Efarmogi toy best first:'),
      write(V-State),
      nl,
      expand(V-[State|Path],NewPaths),
      append(NewPaths,RestPaths,Frontier),
      keysort(Frontier,OrFrontier),
      bfs(OrFrontier,Solution).
```

Στο παραπάνω κομμάτι κώδικα καλούμε τον κανόνα expand(V-[State|Path],NewPaths) μέσω του οποίου βρίσκουμε τις καταστάσεις παιδία της τρέχουσας κατάστασης. Πιο συγκεκριμένα ο

κανόνας expand φαίνεται παρακάτω. Σύμφωνα με αυτό το κομμάτι κώδικα βρίσκουμε όλες τις καταστάσεις παιδία της τρέχουσας κατάστασης. Για να βρούμε τις καταστάσεις παιδία εφαρμόζουμε τους τελεστές μετάβασης του προβλήματος και κάθε νέα κατάσταση που προκύπτει την ονομάζουμε NewState. Για κάθε νέα κατάσταση-παιδί κοιτάμε να μην ανήκει ήδη στο μέτωπο αναζήτησης από την εντολή `not(member(NewState,Path))` και στη συνέχεια υπολογίζουμε την τιμή της ευρετικής συνάρτησης για αυτή τη νέα κατάσταση καλώντας τον κανόνα heurestic. Ο κανόνας heuristic φαίνεται στη συνέχεια.

```
heuristic(State,V):-          findall(Value,
                              h(State,Value),
                              AllValues),
                              min(AllValues,V),!.
```

Σύμφωνα με τον παραπάνω κανόνα υπολογίζουμε την τιμή της ερευτική συνάρτησης για κάθε κατάσταση και στην συνέχεια καλούμε τον κανόνα min για να βρούμε την κατάσταση με βάση την μικρότερη τιμή της ευρετικής.
Στη συνέχεια την τιμή της ευρετικής συνάρτησης την περνάμε στην μεταβλητή Value και τέλος αποθηκεύουμε όλες τις νέες καταστάσεις στο NewPaths.

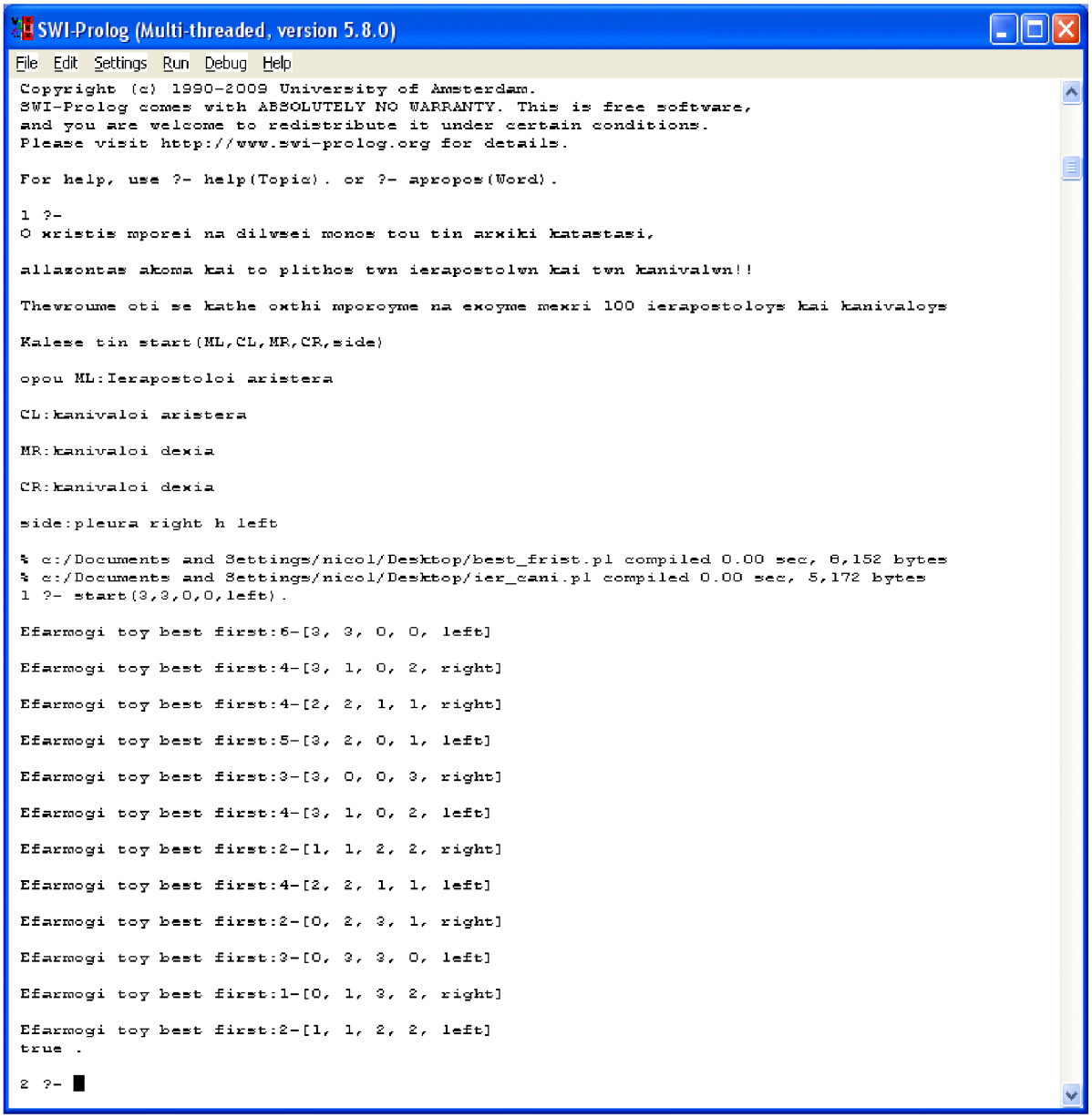
```
expand(_-[State|Path],NewPaths):-   findall(
                                    Value-[NewState,State|Path],
                                    (operator(State,NewState),
                                    not(member(NewState,Path)),
                                    heuristic(NewState,HV),
                                    Value is HV),
                                    NewPaths).
```

Στη συνέχεια περάμε τις καταστάσεις παιδία στο μέτωπο αναζήτησης κρατώντας τις καταστάσεις που ήδη βρίσκονται εκεί με την `append(NewPaths,RestPaths,Frontier).`
Τέλος ταξινομούμε το μέτωπο αναζήτησης με βάση την `keysort(Frontier,OrFrontier)` και εκτελούμε και πάλι τον best first για την πρώτη κατάσταση του ταξινομημένου μετώπου αναζήτησης δηλαδή της κατάστασης με την μικρότερη τιμή ευρετικής συνάρτησης με την `bfs(OrFrontier,Solution).` Το παραπάνω κομμάτι κώδικα εκτελείται μέχρι η τρέχουσα κάθε φορά κατάσταση να είναι ίδια με την τελική κατάσταση ή μέχρι το μέτωπο αναζήτησης να μην έχει πλέον άλλες καταστάσεις για να εφαρμόσουμε τον αλγόριθμο.

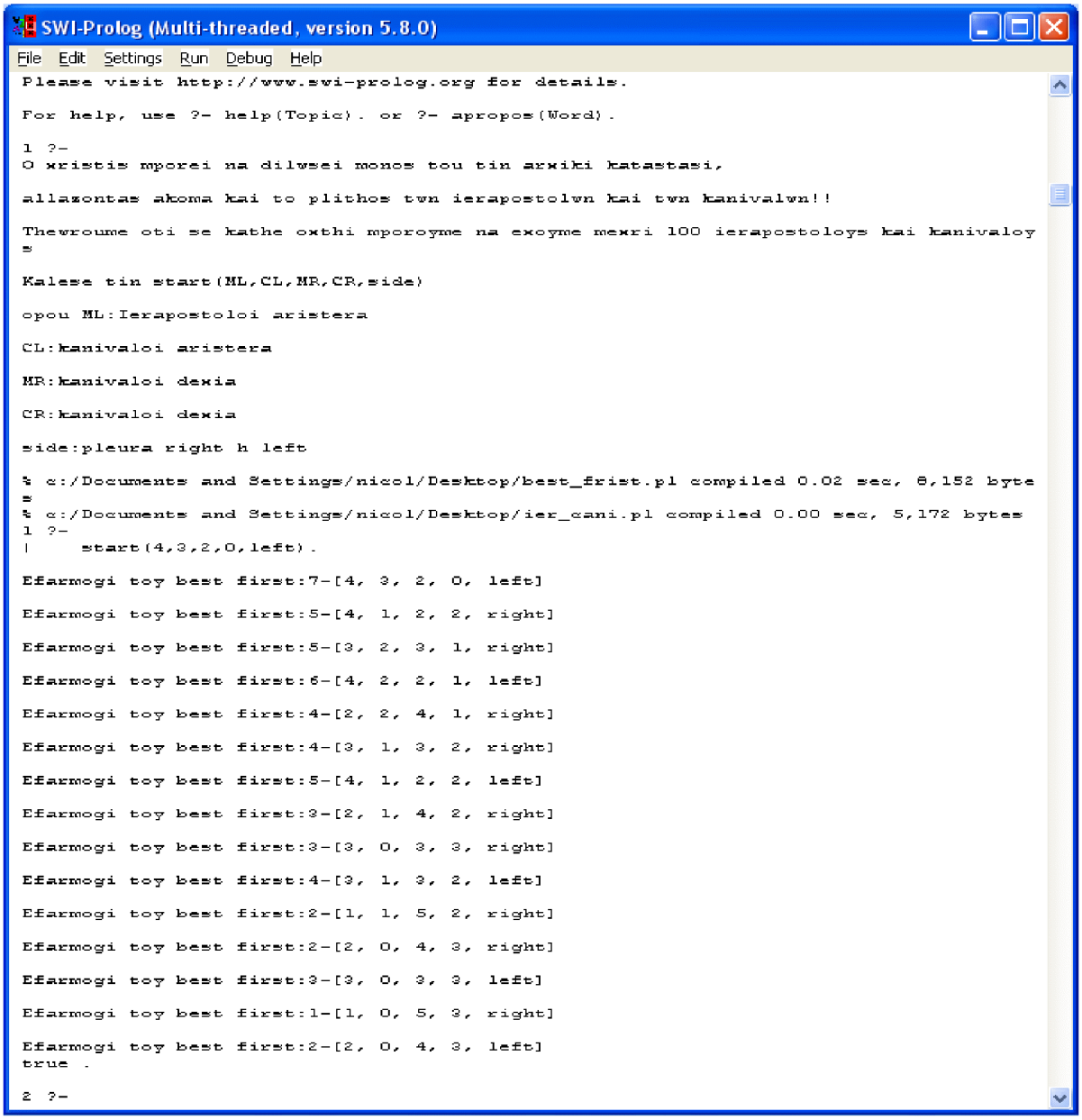
4.5. Έχουμε παραμετροποιήσει τον αλγόριθμό μας έτσι ώστε ο χρήστης να μπορεί να καθορίσει μόνος του το πλήθος των κανιβάλων και των ιεραποστόλων αλλά και τον αριθμό των κανιβάλων και των

ιεραποστόλων τόσο στη δεξιά όσο και στην αριστερή όχθη του ποταμού, δηλαδή ο χρήστης μπορεί να καθορίζει μόνος του την αρχική κατάσταση του προβλήματος . Έχουμε θέσει βέβαια τον περιορισμό ότι μπορεί να έχει μέχρι 100 ιεραπόστολους και 100 κανίβαλους σε κάθε όχθη για να μπορούμε να κάνουμε έλεγχο ότι αυτό που μας δίνει ο χρήστης σαν αριθμό των ιεραποστόλων και των κανιβάλων είναι αριθμός. Βέβαια όλα τα παραπάνω θα πρέπει να συμφωνούν με τους περιορισμούς του προβλήματός μας. Δηλαδή ότι ο συνολικός αριθμός ιεραποστόλων να είναι μεγαλύτερος ή ίσος με τον συνολικό αριθμό κανιβάλων όπως και το ότι σε κάθε όχθη του ποταμού ο αριθμός των κανιβάλων δεν ξεπερνά τον αριθμό των ιεραποστόλων.
Στη συνέχεια παραθέτουμε τρία τρεξίματα του προβλήματός μας. Σε κάθε περίπτωση όποια και αν είναι η αρχική μας κατάσταση θεωρούμε σαν τελική κατάσταση την (0,0,m,n,right) οπού m είναι το πλήθος των ιεραπόστολων και n το πλήθος των κανιβάλων. Δηλαδή θεωρούμε σαν τελική κατάσταση αυτή στην οποία έχουν μεταφερθεί όλοι οι ιεραπόστολοι και όλοι η κανίβαλοι στην δεξιά όχθη του ποταμού.

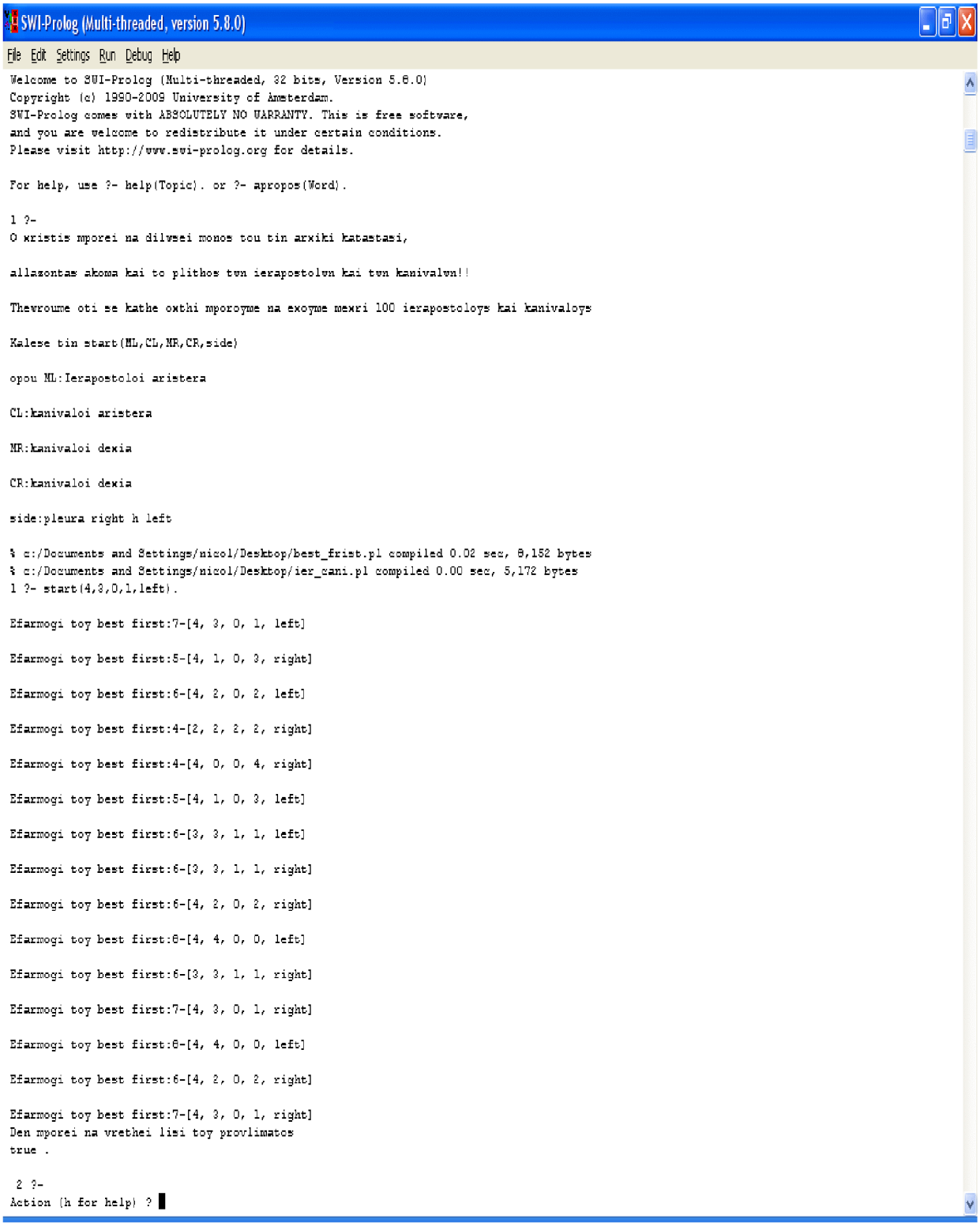
1. Στο παρακάτω θεωρούμε ότι έχουμε σαν αρχική κατάσταση την (3,3,0,0,left). Δηλαδή έτσι όπως λύνεται το προβλημά μας πριν την παραμετροποιηση.



2. Έστω τώρα ότι έχουμε την τυχαία κατάσταση (4,3,2,0,left). Οπότε είναι:



3. Στο παρακάτω τρέχουμε το πρόγραμμα με αρχική κατάσταση την (4,3,0,1,right). Σε αυτή την περίπτωση ο αλγόριθμος best first δεν μπορεί να βρεί λύση οπότε μας εμφανίζει ανάλογο μήνυμα.



4.6. Στο σημείο αυτό θα αναφερθώ στην παραμετροποίηση του αλγορίθμου. Πιο συγκεκριμένα όπως αναφέραμε και πιο πάνω

έχουμε παραμετροποιήσει τον αλγόριθμό μας έτσι ώστε ο χρήστης να μπορεί να καθορίσει μόνος του το πλήθος των κανιβάλων και των ιεραποστόλων αλλά και τον αριθμό των κανιβάλων και των ιεραποστόλων τόσο στη δεξιά όσο και στην αριστερή όχθη του ποταμού, δηλαδή ο χρήστης μπορεί να καθορίζει μόνος του την αρχική κατάσταση του προβλήματος . Έχουμε θέσει βέβαια τον περιορισμό ότι μπορεί να έχει μέχρι 100 ιεραπόστολους και 100 κανίβαλους σε κάθε όχθη για να μπορούμε να κάνουμε έλεγχο ότι αυτό που μας δίνει ο χρήστης σαν αριθμό των ιεραποστόλων και των κανιβάλων είναι αριθμός. Ο έλεγχος αυτός γίνεται με βάση το κομμάτι κώδικα που ακολουθεί. Στο κομμάτι αυτό κάνουμε έλεγχο για τον αριθμό των ιεραποστόλων στην αριστερή όχθη του ποταμού. Όμοια με αυτό ελέγχουμε και τους κανίβαλους και στις δυο όχθες του ποταμού και τους ιεραπόστολους την δεξιά όχθη του ποταμού.

```
mnuml(X):-        X = 0;
                  (X > 0, X < 101);
                  (write('Oi ierapostoloi stin aristeri oxthi den
                  exoyn parei swsti timi'),nl,nl, fail).
```

Επιπλέον θα πρέπει να ελέγχουμε ότι η πλευρά που μας δίνει ο χρήστης είναι σωστή αφού δεκτές είναι μόνο οι left και right. Αυτό ελέγχεται από το παρακάτω κομμάτι κώδικα:

```
side(X):-         X = right;
                  X = left;
                  write('H pleura pou dwsate den einai
                  apodekti'),nl,nl,fail.
```

Βέβαια όλα τα παραπάνω θα πρέπει να συμφωνούν με τους περιορισμούς του προβλήματός μας. Δηλαδή ότι ο συνολικός αριθμός ιεραποστόλων να είναι μεγαλύτερος ή ίσος με τον συνολικό αριθμό κανιβάλων. Αυτό το ελέγχουμε με βάση το παρακάτω κομμάτι κώδικα όπου X είναι ο συνολικός αριθμός των ιεραποστόλων σύμφωνα με την κατάσταση που μας έδωσε ο χρήστης και Y είναι ο συνολικός αριθμός των κανιβάλων:

```
cs(X,Y):-   X>=Y;
            write('Oi ierapostoloi einai pio ligoi se athroisma apo
            toys kanivaloys.Mi epitreptei katastasi'),nl,nl,fail.
```

Τέλος σε κάθε όχθη του ποταμού ο αριθμός των κανιβάλων δεν ξεπερνά τον αριθμό των ιεραποστόλων. Αυτός ο περιορισμός ελέγχεται από το ακόλουθο τμήμα κώδικα. Στον παρακάτω κώδικα κάνουμε τον έλεγχο για την αριστερά πλευρά του ποταμού. Όμοια ελέγχουμε την δεξιά πλευρά του ποταμού.

```
cl(0,_).

cl(X,Y):-   X>=Y;
```

```
write('oi ierapostoloi einai pio ligoi apo toys
kanivalouss sti aristeri oxthi.Mi epitreptei
katastasi'),nl,nl,fail.       %plithos ierapostolwn
pio megalo apo plithos kanivalwn
```

Στον αλγοριθμό μας θεωρούμε πάντα ότι η τελική κατάσταση είναι η (0,0,m,n,right) οπού m είναι το πλήθος των ιεραπόστολων και n το πλήθος των κανιβάλων. Δηλαδή θεωρούμε σαν τελική κατάσταση αυτή στην οποία έχουν μεταφερθεί όλοι οι ιεραπόστολοι και όλοι η κανίβαλοι στην δεξιά όχθη του ποταμού ανεξάρτητα από την κατάσταση που μας έχει δώσει ο χρήστης σαν αρχική.

Όσον αφορά την απόδοση του αλγορίθμου παρατηρούμε ότι όσο αυξάνεται το πλήθος των ιεραπόστολων και των κανίβαλων τόσο πιο πολλές καταστάσεις περνά ο αλγόριθμος μας για να βρει την λύση. Ενώ σε πολλές περιπτώσει δεν υπάρχει λύση δηλαδή δεν υπάρχει τρόπος για να μεταφέρουμε όλους τους κανίβαλους και τους ιεραπόστολους στην δεξιά όχθη του ποταμού.

Για να τρέξετε το πρόγραμμα

1. Ανοίγεται το πρόγραμμα prolog
2. Πηγαίνετε file->consult και αναζητάτε το αρχείο best_first
3. Όμοια αναζητάτε το αρχείο ier_cani
4. Πατάτε start(ML,CL,MR,CR,Side)
   Όπου
   ML: ο αριθμός των ιεραπόστολων στην αριστερή όχθη του ποταμού
   CL: ο αριθμός των κανίβαλων στην αριστερή όχθη του ποταμού
   MR: ο αριθμός των ιεραπόστολων στην δεξιά όχθη του ποταμού
   CR: ο αριθμός των κανίβαλων στην δεξιά όχθη του ποταμού
   Side: left ή right ανάλογα με το σε ποια όχθη βρίσκεται η βάρκα.

5. ΑΛΓΟΡΙΘΜΟΣ Β (ΧΙΩΤΗΣ ΓΕΩΡΓΙΟΣ)

5.1. Ο αλγόριθμος A* για να υλοποιηθεί χρειάζεται μια ευρετική συνάρτηση και μια συνάρτηση κόστους.
Σαν ευρετική συνάρτηση χρησιμοποιούμε μια συνάρτηση που μετρά την απόσταση κάθε κατάστασης από τον στόχο(τελική κατάσταση).
Οπότε ορίζουμε σαν ευρετική συνάρτηση την

$$h(n)=ML+CL$$

όπου
ML: ο αριθμός των ιεραπόστολων στην αριστερή όχθη του ποταμού
CL: ο αριθμός των κανίβαλων στην αριστερή όχθη του ποταμού.
Ουσιαστικά με την παραπάνω ευρετική βρίσκουμε το άθροισμα των ιεραποστόλων και των κανιβάλων που βρίσκονται στην αριστερή όχθη του ποταμού και επομένως απομένουν ακόμα να μεταφερθούν στην δεξιά όχθη
Σαν συνάρτηση κόστους θεωρούμε ότι το κόστος μετάβασης είναι πάντα 1 αφού δεν υπάρχει κάτι που να διαφοροποιεί το κόστος στις διάφορες μεταβάσεις. Οπότε η συνάρτηση κόστους είναι g(n)=1+g(n-1) για κάθε κατάσταση n όπου g(n-1) είναι το κόστος της κατάστασης γονέα.

5.2. Στη συνέχεια ακολουθεί η περιγραφή του αλγορίθμου A* σε φυσική γλώσσα:

1. Δημιούργησε μία ουρά (queue) που αρχικά έχει τη ρίζα (root).
2. Έως ότου η ουρά είναι άδεια ή ο στόχος έχει βρεθεί, έλεγξε αν το πρώτο στοιχείο είναι ο στόχος.

   2α. Αν είναι μην κάνεις τίποτα.

   2β. Αν δεν είναι, τότε:

   - βγάλε το πρώτο στοιχείο
   - βάλε τα παιδιά του πρώτου στοιχείου **πίσω** στην ουρά
   - **ταξινόμησε** όλη την ουρά με βάση την f(s) (ο καλύτερος κόμβος μπροστά)

3. Αν έχει βρεθεί ο στόχος, τότε έχουμε επιτυχία, αλλιώς, αποτυχία

5.3. Ο ψευδοκώδικας για τον αλγόριθμο A* είναι:

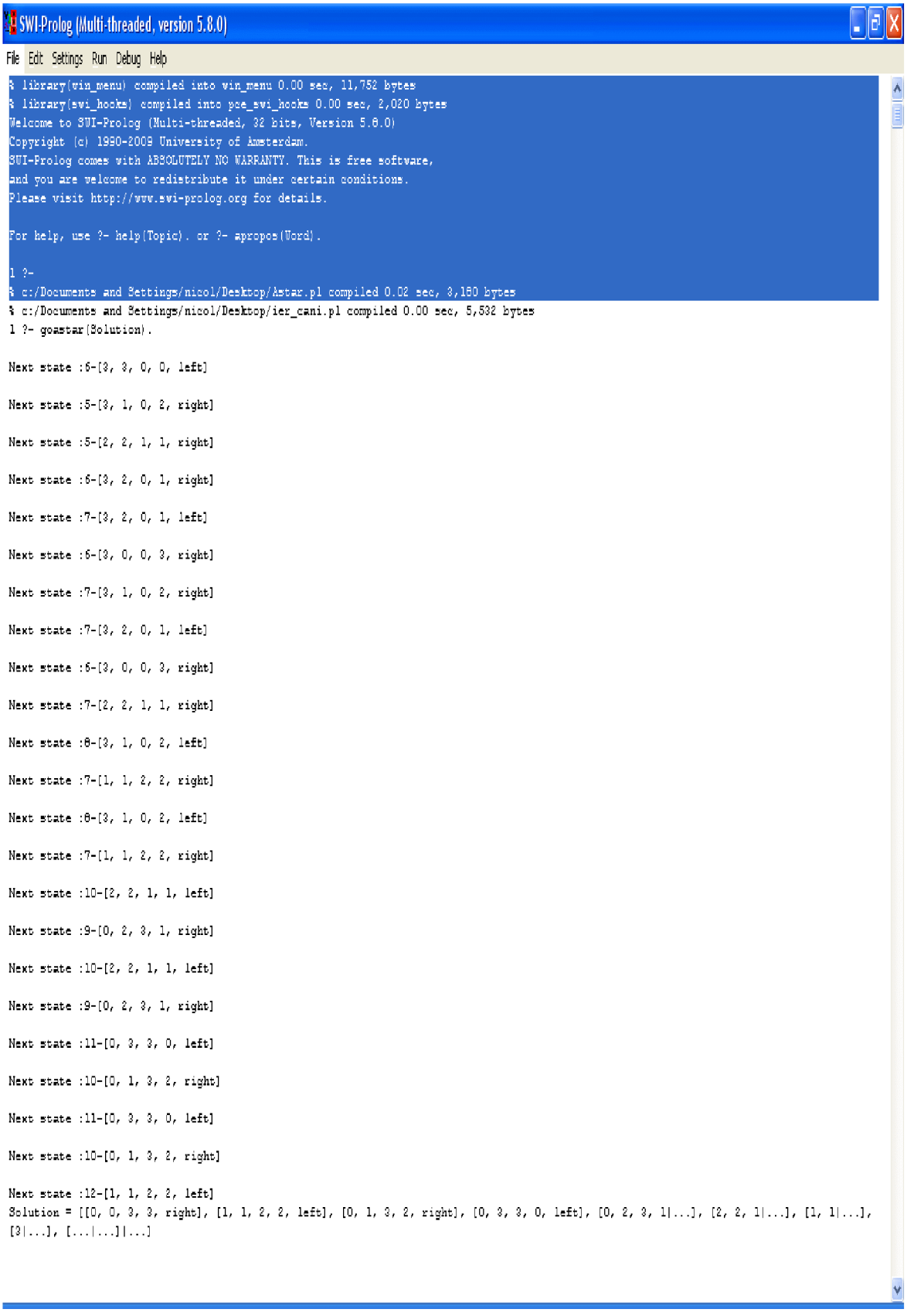
```
algorithm astar(InitialState, FinalState)
begin
    queue←root;
```

```
    while ( root == Ø OR currentSTATE == FinalState )
    goalstate = queue[1];

    if ( currentSTATE == FinalState )
    Solution Found;
else
        swap( queue[1] , queue[ endOFqueue ] );
        sort ( queue, by( A*_Heuristic ) );

end
end
```

5.4.Στη συνεχεια φαίνεται πως τρέχει ο αλγοριθμός μας για αρχική κατάσταση την (3,3,0,0,left).


SWI-Prolog (Multi-threaded, version 5.8.0)
File Edit Settings Run Debug Help
% library(win_menu) compiled into win_menu 0.00 sec, 11,752 bytes
% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 2,020 bytes
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.8.0)
Copyright (c) 1990-2009 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.
For help, use ?- help(Topic). or ?- apropos(Word).
1 ?-
% c:/Documents and Settings/nicol/Desktop/Astar.pl compiled 0.02 sec, 3,180 bytes
% c:/Documents and Settings/nicol/Desktop/ier_cani.pl compiled 0.00 sec, 5,532 bytes
1 ?- goastar(Solution).
Next state :6-[3, 3, 0, 0, left]
Next state :5-[3, 1, 0, 2, right]
Next state :5-[2, 2, 1, 1, right]
Next state :6-[3, 2, 0, 1, right]
Next state :7-[3, 2, 0, 1, left]
Next state :6-[3, 0, 0, 3, right]
Next state :7-[3, 1, 0, 2, right]
Next state :7-[3, 2, 0, 1, left]
Next state :6-[3, 0, 0, 3, right]
Next state :7-[2, 2, 1, 1, right]
Next state :8-[3, 1, 0, 2, left]
Next state :7-[1, 1, 2, 2, right]
Next state :8-[3, 1, 0, 2, left]
Next state :7-[1, 1, 2, 2, right]
Next state :10-[2, 2, 1, 1, left]
Next state :9-[0, 2, 3, 1, right]
Next state :10-[2, 2, 1, 1, left]
Next state :9-[0, 2, 3, 1, right]
Next state :11-[0, 3, 3, 0, left]
Next state :10-[0, 1, 3, 2, right]
Next state :11-[0, 3, 3, 0, left]
Next state :10-[0, 1, 3, 2, right]
Next state :12-[1, 1, 2, 2, left]
Solution = [[0, 0, 3, 3, right], [1, 1, 2, 2, left], [0, 1, 3, 2, right], [0, 3, 3, 0, left], [0, 2, 3, 1|...], [2, 2, 1|...], [1, 1|...], [3|...], [...|...]|...]

7. ΣΥΓΚΡΙΣΗ ΤΩΝ ΑΛΓΟΡΙΘΜΩΝ-ΣΥΜΠΕΡΑΣΜΑΤΑ

Τόσο ο αλγόριθμος Best first όσο και ο A* μπορούν να επιστρέψουν σε μια κατάσταση αν το μονοπάτι που ακολουθεί βρίσκεται σε χειρότερο δρόμο.

Ο Best first σταματά συνήθως την αναζήτηση αν βρεθεί μια λύση. Γενικά έχει το πλεονέκτημα ότι μπορεί να βρίσκει περισσότερες λύσεις αλλά αν το πρόβλημα έχει μία μόνο λύση χάνεται αυτό το πλεονέκτημα. Ο Best first προσπαθεί να δώσει μια γρήγορη λύση σε ένα πρόβλημα. Ωστόσο αν το καταφέρνει ή όχι εξαρτάται από την ευρετική συνάρτηση που χρησιμοποιείται. Μια κακή ευρετική συνάρτηση θα υποχρεώσει τον αλγόριθμο σε χρονοβόρα αναζήτηση μονοπατιών που δεν αποτελούν μέρος μιας λύσης. Το μειονέκτημά του είναι ότι το μέτωπο αναζήτησης μεγαλώνει με υψηλό ρυθμό και μαζί με αυτό μεγαλώνει και ο χώρος που χρειάζεται για την αποθήκευση του καθώς και ο χρόνος για την επεξεργασία των στοιχείων του. Ο Best first είναι πλήρης αλλά δεν εγγυάται ότι η λύση που θα βρει είναι η βέλτιστη.

Ο A* είναι ένας άπληστος αλγόριθμος, μια εξειδίκευση του best first, που βρίσκει τη συντομότερη διαδρομή από μια αρχική κατάσταση σε μια τελική. Ενώ στον Best first χρησιμοποιείται μόνο μια ευρετική συνάρτηση που δίνει μόνο μια εκτίμηση της απόστασης μιας κατάστασης από την τελική, στον A* στην τιμή αυτή προστίθεται η απόσταση που έχει διανυθεί από την αρχική κατάσταση μέχρι την τρέχουσα κατάσταση.
Δηλαδή: $f(s)=h(s)+g(s)$
όπου: s: η τρέχουσα κατάσταση
g(s): δίνει την απόσταση της s από την αρχική κατάσταση IS
h(s):δίνει την εκτίμηση της απόστασης της s από την τελική κατάσταση μέσω μιας ευρετικής συνάρτησης.

Γενικά από τους δυο παραπάνω αλγόριθμους με τον A* θα οδηγηθώ στην κατάσταση στόχος έχοντας πληρώσει το λιγότερο κόστος. Παρότι ο αλγόριθμος A* μπορεί και να περνά από περισσότερες καταστάσεις σε σχέση με τον Best first για να φτάσει στην λύση.