ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Τεχνητή Νοημοσύνη Ι

Εργαστηριακή Άσκηση 4-6

Σγάρμπας Κυριάκος

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστων

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ - ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ & ΤΕΧΝΟΛΟΓΙΑΣ ΤΗΣ ΠΛΗΡΟΦΟΡΙΑΣ - ΕΡΓΑΣΤΗΡΙΟ ΕΝΣΥΡΜΑΤΗΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΣ

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ Ι - ΕΡΓΑΣΤΗΡΙΟ

ΕΝΟΤΗΤΑ Β: "Αλγόριθμοι Αναζήτησης"
ΑΣΚΗΣΕΙΣ #4, #5, #6

Οι αλγόριθμοι αναζήτησης που θα εξετάσουμε στην παρούσα ενότητα ασκήσεων είναι γενικοί. Δηλαδή λύνουν οποιοδήποτε πρόβλημα, αρκεί να τους προσθέσουμε το κατάλληλο interface: τα δεδομένα του προβλήματος μέσω της συνάρτησης διαδόχων.

ΑΣΚΗΣΗ #4
(BFS, UCS, DFS)

1. Ο παρακάτω κώδικας Prolog υλοποιεί τον αλγόριθμο **αναζήτησης πρώτα σε πλάτος** (**Breadth-First Search = BFS**).

```
% Breadth-First Search
% Επειδή δεν έχει μνήμη επισκεφθέντων καταστάσεων,
% ..κλειδώνει σε loop αν δεν υπάρχει λύση.
% Αν υπάρχει λύση βρίσκει αυτήν που βρίσκεται σε μικρότερο βάθος,
% ..όχι κατ'ανάγκη αυτή με το μικρότερο κόστος.
% Καλείται ως: find_BFS(+State1,+State2,-Node)
search_BFS([CurrentNode|_],GoalNode):-
    GoalNode=node(C,State,A,D,P),
    CurrentNode=node(C,State,A,D,P), !.
% Εναλλακτικά: search_BFS([G|_],G):- !.
search_BFS([CurrentNode|Fringe],GoalNode):-
    findall(SuccNode,successor(CurrentNode,SuccNode),NList),
    append(Fringe,NList,NewFringe), search_BFS(NewFringe,GoalNode).
find_BFS(Initial_State,Goal_State,GoalNode):-
    GoalNode=node(_,Goal_State,_,_,_),
    search_BFS([node(0,Initial_State,move,0,null)],GoalNode).
```

   Και ο επόμενος υλοποιεί τον αλγόριθμο **αναζήτησης ομοιόμορφου κόστους** (**Uniform Cost Search**).

```
% Uniform Cost Search
% Καλείται ως: find_UCS(+State1,+State2,-Node)
search_UCS([CurrentNode|_],GoalNode):-
    GoalNode=node(C,State,A,D,P), CurrentNode=node(C,State,A,D,P), !.
search_UCS([CurrentNode|Fringe],GoalNode):-
    findall(SuccNode,successor(CurrentNode,SuccNode),NList),
    append(Fringe,NList,NewFringe), msort(NewFringe,SNFringe),
    search_UCS(SNFringe,GoalNode).
find_UCS(Initial_State,Goal_State,GoalNode):-
    GoalNode=node(_,Goal_State,_,_,_),
    search_UCS([node(0,Initial_State,move,0,null)],GoalNode).
```

   Μελετήστε τον κώδικα και αφού καταλάβετε πώς λειτουργεί, προσθέστε τις κατάλληλες συναρτήσεις ώστε να λύνει το πρόβλημα του χάρτη της Ρουμανίας και με τους δύο αλγορίθμους.
   Υπόδειξη: Θα χρειαστεί να προσθέσετε τις συναρτήσεις **link/3** και **nextd/3** από την προηγούμενη ενότητα ασκήσεων, καθώς και μια νέα μη ντετερμινιστική συνάρτηση **successor/2** που θα εκφράζει την συνάρτηση διαδόχων του προβλήματος και θα είναι κοινή και για τους δυο αλγορίθμους. Η κλήση της είναι: **successor(+Node,-NextNode)**. Θεωρήστε

ότι κάθε κόμβος έχει τη δομή: **node(PathCost, State, Action, Depth, Parent)**.

2. Ο παρακάτω κώδικας Prolog υλοποιεί τον αλγόριθμο **αναζήτησης πρώτα σε βάθος** (**Depth-First Search = DFS**).

```
% Depth-First Search
% Επικίνδυνη γιατί μπορεί να κλειδώσει σε ατέρμονο loop.
% Κλήση: find_DFS(+State1,+State2,-Node)
search_DFS(CurrentNode,GoalNode):-
    GoalNode=node(C,State,A,D,P), CurrentNode=node(C,State,A,D,P), !.
search_DFS(CurrentNode,GoalNode):-
    successor(CurrentNode,SuccNode), search_DFS(SuccNode,GoalNode).
find_DFS(Initial_State,Goal_State,GoalNode):-
    GoalNode=node(_,Goal_State,_,_,_),
    search_DFS(node(0,Initial_State,move,0,null),GoalNode), !.
```

Μελετήστε τον κώδικα και αφού καταλάβετε πώς λειτουργεί, χρησιμοποιήστε τον ως πρότυπο για να φτιάξετε τον αλγόριθμο **αναζήτησης περιορισμένου βάθους** (**Depth-Limited Search = DLS**) με πρότυπο κλήσης: **find_DLS(+State1,+State2,+Limit,-Node)**. Ελέγξτε τη σωστή λειτουργία του στο πρόβλημα του χάρτη της Ρουμανίας.

ΑΣΚΗΣΗ #5
(IDS, Συναρτήσεις Διαδόχων)

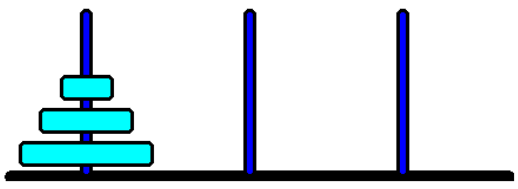
3. Με βάση την εμπειρία σας από τους παραπάνω αλγορίθμους, υλοποιήστε τον αλγόριθμο **αναζήτησης με επαναληπτική εμβάθυνση** (**Iterative-Deepening Search = IDS**) με πρότυπο κλήσης: **find_IDS(+State1,+State2,-Node)**. Μπορείτε να χρησιμοποιείτε κλήσεις προς τις συναρτήσεις των προηγούμενων αλγορίθμων, αν θέλετε. Ελέγξτε τη σωστή λειτουργία του στο πρόβλημα του χάρτη της Ρουμανίας.

4. Φτιάξτε μια συνάρτηση με πρότυπο κλήσης **detailed_print(+Node)** η οποία θα δέχεται έναν κόμβο και θα τυπώνει όλο το μονοπάτι από τον αρχικό κόμβο προς αυτόν, με τα ενδιάμεσα κόστη και τις λοιπές πληροφορίες που καταχωρούνται στους κόμβους. Για παράδειγμα, θα πρέπει να τρέχει κάπως έτσι:

```
run_romania_map:- find_IDS(a,b,R), detailed_print(R).
?- run_romania_map.
0 0 move:a
1 140 move:s
2 239 move:f
3 450 move:b
true.
```

5. Τροποποιήστε κατάλληλα την **successor/2** ώστε να γίνει ανεξάρτητη από το πρόβλημα αναζήτησης.
   <u>Υπόδειξη:</u> Μεταφέρετε τα στοιχεία που εξαρτώνται από το πρόβλημα σε νέα συνάρτηση.

6. Προσθέστε κατάλληλο κώδικα ώστε να λύσετε με αναζήτηση (προτιμήστε τον αλγόριθμο IDS) το πρόβλημα των Πύργων του Ανόι:
   Τρεις δίσκοι είναι τοποθετημένοι στον αριστερό από τρεις στύλους όπως στο σχήμα, και πρέπει να μετακινηθούν όλοι στον δεξιό στύλο. Επιτρέπεται να μετακινούμε μόνο έναν δίσκο κάθε φορά και ποτέ δεν μπορούμε να τοποθετήσουμε έναν μεγαλύτερο πάνω σε έναν μικρότερο. Το πρόβλημα γενικεύεται για οποιοδήποτε πλήθος δίσκων, αλλά πάντα οι στύλοι είναι τρεις.



7. Ο παρακάτω κώδικας λύνει το πρόβλημα των Πύργων του Ανόι χωρίς αναζήτηση:

```
hanoi(0,_,_,_) :- !.
hanoi(N,A,B,C) :-
    M is N-1, hanoi(M,A,C,B),
    write([move,from,A,to,B]), nl,
    hanoi(M,C,B,A).
```

Τρέξτε τον με:

```
?- hanoi(3,left,right,middle).
```

Συγκρίνετε το αποτέλεσμα με αυτό που βρίσκει η αναζήτηση. Σχολιάστε τα πλεονεκτήματα και τα μειονεκτήματα των δύο προσεγγίσεων.

8. Ο παρακάτω κώδικας λύνει το πρόβλημα του αγρότη:
Ένας αγρότης βρίσκεται στην αριστερή όχθη ενός ποταμού μαζί με ένα λύκο, μια κατσίκα και ένα μεγάλο μαρούλι και πρέπει να περάσουν όλοι απέναντι με μια βάρκα που χωρά μόνο τον αγρότη και έναν επιπλέον επιβάτη, χωρίς ποτέ να βρεθεί η κατσίκα στην ίδια όχθη με το μαρούλι χωρίς να είναι παρών ο αγρότης (γιατί η κατσίκα θα φάει το μαρούλι), ούτε ο λύκος μαζί με την κατσίκα χωρίς τον αγρότη (γιατί ο λύκος θα φάει την κατσίκα).

```
% Το πρόβλημα του Αγρότη
% initial state: s([a,k,l,m],[])    final state: s([], [a,k,l,m])
farmer_ps(_,State,SAction,SState,Cost):-
    farmer_next_move(State,SState,SAction), Cost=1.
farmer_next_move(s([a|LT],R),s(LT,[a|R]),((a),right)):- is_safe(LT).
farmer_next_move(s(L,[a|RT]),s([a|L],RT),((a),left)):- is_safe(RT).
farmer_next_move(s([a|LT],R),s(Ln,[a|Rn]),((a,Passenger),right)):-
    member(Passenger,LT), remove(LT,Passenger,Ln), is_safe(Ln),
    msort([Passenger|R],Rn).
farmer_next_move(s(L,[a|RT]),s([a|Ln],Rn),((a,Passenger),left)):-
    member(Passenger,RT), remove(RT,Passenger,Rn), is_safe(Rn),
    msort([Passenger|L],Ln).
is_safe(L):- member(l,L),member(k,L),!,fail.
is_safe(L):- member(m,L),member(k,L),!,fail.
is_safe(_).
remove(L,X,L2):- append(A,[X|B],L), append(A,B,L2).
run_farmer_puzzle:-
    find_IDS(s([a,k,l,m],[]),s([],[a,k,l,m]),R), detailed_print(R).
```

Μελετήστε τον κώδικα και προσπαθήστε να καταλάβετε πώς λειτουργεί.

9. Προσθέστε κατάλληλο κώδικα ώστε να λύσετε με αναζήτηση (με IDS) το puzzle των 8 πλακιδίων:
Έστω 8 πλακίδια, τοποθετημένα σε διάταξη 3x3 με μια θέση κενή. Σε κάθε κίνηση επιτρέπεται να μετακινηθεί οριζόντια ή κατακόρυφα μόνο ένα πλακίδιο γειτονικό της κενής θέσης ώστε να την καλύψει. Ο στόχος είναι να φτάσετε από μια αρχική κατάσταση στην τελική.

ΑΡΧΙΚΗ

1	2	3
4		5
6	7	8

ΤΕΛΙΚΗ

1	2	3
4	5	6
7	8	

Χρονομετρήστε τον αλγόριθμό σας.
Μετά δοκιμάστε ως αρχική κατάσταση αυτήν:

1	2	
3	4	5
6	7	8

Υπόδειξη: Μπορείτε να συμβολίσετε κάθε κατάσταση είτε ως λίστα είτε ως ένα σύνθετο αντικείμενο, πχ. s(1,2,3,4,5,6,7,8,0) όπου 0 συμβολίζει το κενό τετράγωνο.

ΑΣΚΗΣΗ #6
(BestFS, A*)

10. Υλοποιήστε τον αλγόριθμο **Best-First Search** (**Αναζήτηση Πρώτα στο Καλύτερο**) με πρότυπο κλήσης: **find_BestFS(+State1,+State2,-Node)** και χρησιμοποιήστε τον για να λύσετε το puzzle των 8 πλακιδίων.
    Υπόδειξη: Ως ευρετική συνάρτηση χρησιμοποιήστε το άθροισμα των αποστάσεων Manhattan των αριθμών από τις τελικές τους θέσεις. (Η συνάρτηση αυτή είναι consistent.)
    Προσοχή: Ο αλγόριθμος κινδυνεύει από ατέρμονα loops.

11. Προσθέστε μνήμη στον προηγούμενο αλγόριθμο ώστε να μην αναπτύσσει καταστάσεις που έχει ξανασυναντήσει (και να μην πέφτει σε ατέρμονα loops). Προσέξτε πόσο πιο γρήγορα βρίσκει λύση στις περιπτώσεις που ο IDS καθυστερεί πολύ, αλλά δεν βρίσκει τόσο καλές λύσεις όπως ο IDS.

12. Υλοποιήστε τον αλγόριθμο **A*** με πρότυπο κλήσης **find_AstarS(+State1,+State2,-Node)** και χρησιμοποιήστε τον για να λύσετε το puzzle των 8 πλακιδίων. Προσέξτε πόσο πιο γρήγορα βρίσκει τη βέλτιστη λύση σε σύγκριση με τον IDS.

Σημειώματα

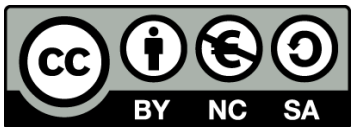
Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0

- Έκδοση 1.0 διαθέσιμη εδώ.

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης



Διατήρηση Σημειωμάτων

- Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:
- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Σημείωμα Χρήσης Έργων Τρίτων

Το Έργο αυτό κάνει χρήση των ακόλουθων έργων:

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στοπλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.