4

% A\* SEARCH ALGORITHM.

:- use\_module(library(clpfd)).

% Define the search problem as a predicate with a start state and a goal state

search(Start, Goal, Path) :-

    astar(Start, Goal, Path, \_).

% Define the A\* search algorithm

astar(Start, Goal, Path, Cost) :-

    heuristic(Start, H),

    astar\_search(Start, Goal, [Start], H, Path, Cost).

astar\_search(Goal, Goal, \_, \_, [], 0).

astar\_search(Current, Goal, Visited, H, [Current|Path], Cost) :-

    \+ member(Current, Visited),

    findall(Next, successor(Current, Next), Nexts),

    Nexts \= [],

    maplist(heuristic, Nexts, Hs),

    minimum(Hs, Hmin),

    select(Hmin, Hs, Hrest),

    select(Next, Nexts, Nrest),

    Cost1 is Cost + Hmin,

    astar\_search(Next, Goal, [Next|Visited], Hrest, Path, Cost1).

% Define the successor function for the search problem

successor(State, Next) :-

    append([State, [X]], Next),

    X in 1..9,

    all\_distinct(Next).

% Define the heuristic function for the A\* search algorithm

heuristic(State, H) :-

    H is sum(State) - 3 \* sum(State, X is X mod 3 = 0) - sum(State, X is X mod 3 = 1) - sum(State, X is X mod 3 = 2).

% Example query

?- search([1,2,3,4,5,6,7,8,9], [4,5,6,7,8,9,1,2,3], Path).

1

%  WRITE A PROGRAM IN PROLOG TO IMPLEMENT TOWER OF HANOI WHERE N REPRESENTS THE NUMBER OF DISKS.

move(0,\_,\_,\_).

move(1,A,\_,C):- inform(A,C),!. % base condition

move(N,A,B,C):- M is N-1,

move(M,A,C,B),

inform(A,C),

move(M,B,A,C).

inform(A,B):- write('Move one disk from tower '),

write(A), write(' to tower '), write(B), nl.

2

% HILL CLIMB SEARCH ALGORITHM.

state(a).

state(b).

state(c).

state(d).

goal(d).

move(a, b).

move(b, a).

move(b, c).

move(c, b).

move(c, d).

move(d, c).

heuristic(a, 10).

heuristic(b, 5).

heuristic(c, 2).

heuristic(d, 0).

hill\_climbing(State, Goal) :-

    move(State, NewState),

    (   Goal = NewState

    ;   heuristic(NewState, H),

        hill\_climbing(NewState, Goal, H)

    ).

hill\_climbing(\_, Goal, \_) :-

    goal(Goal).

hill\_climbing(State, Goal, CurrentH) :-

    move(State, NewState),

    heuristic(NewState, NewH),

    NewH < CurrentH,

    hill\_climbing(NewState, Goal, NewH).

% Example

% ?- hill\_climbing(a, Goal).

% Goal = d.

3

% BEST FIRST SEARCH ALGORITHM.

best\_first\_search(Start, Goal, Solution) :-

    best\_first\_search([[Start]], Goal, Solution).

best\_first\_search([[Goal|Path]|\_], Goal, [Goal|Path]).

best\_first\_search([[Current|Path]|Paths], Goal, Solution) :-

    findall([Next, Current|Path],

            (move(Current, Next), \+ member(Next, [Current|Path])),

            Successors),

    append(Paths, Successors, NewPaths),

    sort\_paths(NewPaths, SortedPaths),

    best\_first\_search(SortedPaths, Goal, Solution).

move(a, b).

move(b, c).

move(c, d).

move(d, e).

sort\_paths(Paths, SortedPaths) :-

    predsort(compare\_paths, Paths, SortedPaths).

compare\_paths(Order, [Goal1|\_], [Goal2|\_]) :-

    heuristic(Goal1, H1),

    heuristic(Goal2, H2),

    compare(Order, H1, H2).

heuristic(a, 4).

heuristic(b, 3).

heuristic(c, 2).

heuristic(d, 1).

heuristic(e, 0).

% Sample query:

% ?- best\_first\_search(a, e, Solution).

5

state(a).

state(b).

state(c).

state(d).

move(a, b).

move(b, a).

move(b, c).

move(c, b).

move(c, d).

move(d, c).

terminal(d).

utility(d, 10).

minimax(State, Depth, MaximizingPlayer, Value) :-

    Depth = 0,

    terminal(State),

    utility(State, Value).

minimax(State, Depth, MaximizingPlayer, Value) :-

    Depth > 0,

    \+ terminal(State),

    (   MaximizingPlayer = true

    ;   MaximizingPlayer = false

    ),

    findall(NewState, move(State, NewState), NewStates),

    (   MaximizingPlayer = true

    ->  max\_value(NewStates, Depth, Value)

    ;   min\_value(NewStates, Depth, Value)

    ).

max\_value([], \_, Max) :-

    Max is -infinity.

max\_value([State|Rest], Depth, Max) :-

    minimax(State, Depth, false, Value),

    max\_value(Rest, Depth, NewMax),

    (   Value > NewMax

    ->  Max is Value

    ;   Max is NewMax

    ).

min\_value([], \_, Min) :-

    Min is infinity.

min\_value([State|Rest], Depth, Min) :-

    minimax(State, Depth, true, Value),

    min\_value(Rest, Depth, NewMin),

    (   Value < NewMin

    ->  Min is Value

    ;   Min is NewMin

    ).

% Example query

% ?- minimax(a, 3, \_, Value).

% Value = 10.

6

water\_jug(Start, Goal, Path) :-

    astar(Start, Goal, Path, \_).

astar(Start, Goal, Path, Cost) :-

    heuristic(Start, H),

    astar\_search(Start, Goal, [Start], H, Path, Cost).

astar\_search(Goal, Goal, \_, \_, [], 0).

astar\_search(Current, Goal, Visited, H, [Current|Path], Cost) :-

    \+ member(Current, Visited),

    findall(Next, successor(Current, Next), Nexts),

    Nexts \= [],

    maplist(heuristic, Nexts, Hs),

    minimum(Hs, Hmin),

    select(Hmin, Hs, Hrest),

    select(Next, Nexts, Nrest),

    Cost1 is Cost + Hmin,

    astar\_search(Next, Goal, [Next|Visited], Hrest, Path, Cost1).

successor(State, Next) :-

    State = [J1, J2],

    Next = [J11, J22],

    (J11 = J1, J22 = J2; J11 is min(J1, max(0, J2 - J1)), J22 is J2 + J1 - J11),

    (J11 =< 10, J22 =< 10),

    J11 >= 0,

    J22 >= 0.

heuristic(State, H) :-

    State = [J1, J2],

    H is abs(Goal - J1) + abs(Goal - J2).

% Example query

?- water\_jug([0, 0], [7, 0], Path).

7

:- use\_module(library(clpfd)).

sudoku(Rows) :-

    length(Rows, 9),

    maplist(same\_length(Rows), Rows),

    append(Rows, Vs),

    Vs ins 1..9,

    maplist(all\_distinct, Rows),

    transpose(Rows, Columns),

    maplist(all\_distinct, Columns),

    Boxes = [A,B,C,D,E,F,G,H,I],

    maplist(box\_constraint(Rows), Boxes),

    label(Vs).

box\_constraint(Rows, Box) :-

    append([As,Bs,Cs], Rows),

    append(As, A),

    append(Bs, B),

    append(Cs, C),

    all\_distinct(A),

    all\_distinct(B),

    all\_distinct(C),

    all\_distinct(Box).

% Example query

?- sudoku([

    [5,3,\_, \_,7,\_, \_,\_,\_],

    [6,\_,\_, 1,9,5, \_,\_,\_],

    [\_,9,8, \_,\_,\_, \_,6,\_],

    [8,\_,\_, \_,6,\_, \_,\_,3],

    [4,\_,\_, 8,\_,3, \_,\_,1],

    [7,\_,\_, \_,2,\_, \_,\_,6],

    [\_,6,\_, \_,\_,\_, 2,8,\_],

    [\_,\_,\_, 4,1,9, \_,\_,5],

    [\_,\_,\_, \_,8,\_, \_,7,9]

]).

8

parent(john, mary).

parent(john, jim).

parent(mary, ann).

parent(mary, beth).

male(john).

male(jim).

female(mary).

female(ann).

female(beth).

age(john, 50).

age(mary, 45).

age(jim, 30).

age(ann, 25).

age(beth, 20).

spouse(X, Y) :-

    parent(X, C1),

    parent(Y, C2),

    C1 \= C2.

child(X, Y) :-

    parent(Y, X).

sibling(X, Y) :-

    parent(Z, X),

    parent(Z, Y),

    X \= Y.

brother(X, Y) :-

    sibling(X, Y),

    male(X).

sister(X, Y) :-

    sibling(X, Y),

    female(X).

aunt(X, Y) :-

    sister(X, Z),

    parent(Z, Y).

uncle(X, Y) :-

    brother(X, Z),

    parent(Z, Y).

grandparent(X, Y) :-

    parent(X, Z),

    parent(Z, Y).

% Example queries

% ?- spouse(john, X).

% X = mary.

% ?- child(ann, X).

% X = mary.

% ?- sibling(jim, X).

% X = ann.

% ?- brother(jim, X).

% false.

% ?- sister(ann, X).

% X = beth.

% ?- aunt(ann, X).

% X = beth.

% ?- uncle(john, X).

% X = jim.

% ?- grandparent(john, X).

% X = ann.

9

frame(john, [

    [name, 'John Doe'],

    [age, 30],

    [gender, male],

    [occupation, engineer]

]).

frame(jane, [

    [name, 'Jane Smith'],

    [age, 28],

    [gender, female],

    [occupation, doctor]

]).

frame(car1, [

    [make, toyota],

    [model, 'Corolla'],

    [year, 2018],

    [color, blue]

]).

frame(car2, [

    [make, honda],

    [model, 'Civic'],

    [year, 2020],

    [color, red]

]).

get\_frame\_property(Frame, Property, Value) :-

    frame(Frame, Properties),

    member([Property, Value], Properties).

% Example queries:

% Get the name of John

% ?- get\_frame\_property(john, name, Name).

%

% Get the color of car2

% ?- get\_frame\_property(car2, color, Color).

10

conc(L1, L2, L3) :-

    append(L1, L2, L3).

% Example query

% ?- conc([1, 2], [3, 4], L3).

% L3 = [1, 2, 3, 4].

11

reverse([], []).

reverse([H|T], R) :-

    reverse(T, RevT),

    append(RevT, [H], R).

% Example query

% ?- reverse([1, 2, 3, 4], R).

% R = [4, 3, 2, 1].

12

s(s(NP, VP)) --> np(NP), vp(VP).

np(np(Det, N)) --> det(Det), n(N).

vp(vp(V, NP)) --> v(V), np(NP).

vp(vp(V)) --> v(V).

det(the) --> [the].

det(a) --> [a].

n(cat) --> [cat].

n(dog) --> [dog].

v(chases) --> [chases].

v(sleeps) --> [sleeps].

parse(Tree, Sentence) :-

    phrase(s(Tree), Sentence).

% Example query

% ?- parse(Tree, [the, cat, chases, a, dog]).

13

rule(a, [a]).

rule(b, [b]).

rule(S, [a, b]).

start(S).

recognize(W) :-

    start(S),

    expand(S, W, []).

expand(X, [], W) :-

    rule(X, W).

expand(X, [H|T], W) :-

    rule(X, Y),

    append(Y, [H], Z),

    expand(Z, T, W).

% Example query

% ?- recognize([a, b]).

% true.