JPS

Przemysław Kopański, Mateusz Forc

Baza wiedzy

```
% sciezka ab, z a do b i koszt 2
succ(a,ab,2,b).
succ(b,bf,3,f).
succ(a,ac,3,c).
succ(b,bg,4,g).
succ(g,gm,2,m).
succ(c,cd,2,d).
succ(d,dm,2,m).
succ(c,ce,3,e).
succ(e,em,5,m).

% punkt docelowy
goal(m).

% wartosci f heurystycznej dla danego punktu
hScore(a,4).
hScore(b,4).
hScore(f,7).
hScore(g,1).
hScore(m,0).
hScore(c,3).
hScore(d,1).
hScore(e,4).
```

Modyfikowany kod

```
startSample(StepCounter, MaxCounter, NFirstCounter, PathCost) :-
  start_A_star(a, StepCounter, MaxCounter, NFirstCounter, PathCost).

% program
% StepCounter - licznik zaglebien
start_A_star(InitState, StepCounter, MaxCounter, NFirstCounter, PathCost) :-
  StepCounter =< MaxCounter,
  score(InitState, 0, 0, InitCost, InitScore) ,
  writeln(StepCounter / MaxCounter),
  search_A_star([node(InitState, nil, nil, InitCost, InitScore)], [],
                StepCounter, NFirstCounter, PathCost) .

start_A_star(InitState, StepCounter, MaxCounter, NFirstCounter, PathCost) :-
  StepCounter =< MaxCounter,
  NewStepCounter is StepCounter + 1,
  start_A_star(InitState, NewStepCounter, MaxCounter, NFirstCounter, PathCost).

start_A_star(_, StepCounter, MaxCounter, _, _) :-
```

```
  StepCounter > MaxCounter,
  print("Solution not found").

continue(node(State, Action, Parent, Cost, _), _, ClosedSet, _, _,
     path_cost(Path, Cost) ) :-
  goal(State),
  !,
  build_path(node(Parent, _ , _, _, _) , ClosedSet, [Action/State], Path) .

continue(_, _, _, 0, _, _)   :-
  writeln("Licznik wyczerpany"),
  fail.

continue(Node, RestQueue, ClosedSet, StepCounter, NFirstCounterMax, Path) :-
  StepCounter > 0,
  NewStepCounter is StepCounter - 1,
  expand(Node, NewNodes),
  insert_new_nodes(NewNodes, RestQueue, NewQueue),
  search_A_star(NewQueue, [Node | ClosedSet], NewStepCounter,
                NFirstCounterMax, Path).


fetch(node(State, Action,Parent, Cost, Score),
      [node(State, Action,Parent, Cost, Score) |RestQueue],
      ClosedSet, y, NFirstCounter, RestQueue) :-
  NFirstCounter > 1,
  \+ member(node(State, _, _, _, _) , ClosedSet).

fetch(node(State, Action, Parent, Cost, Score),
      [node(State, Action, Parent, Cost, Score)|RestQueue],
      ClosedSet, UserInput, NFirstCounter, RestQueue) :-
  NFirstCounter == 1,
  \+ member(node(State, _, _, _, _) , ClosedSet),
  writeln("Aktualne stany:"),
  show_states([node(State, Action, Parent, Cost, Score)|RestQueue]),
  writeln("Czy kontynuowac? y/n"),
  read(UserInput), UserInput == y.

fetch(_, _, _, UserInput, _, _) :-
  UserInput == y.

fetch(Node, [ _ |RestQueue], ClosedSet, UserInput, NFirstCounter, NewRest) :-
  NFirstCounter > 0,
  NewNFirstCounter is NFirstCounter - 1,
  fetch(Node, RestQueue, ClosedSet, UserInput, NewNFirstCounter, NewRest).

show_states([]) :- writeln("Stan: nil").
show_states([node(State, _, _, _, Score)|Rest]) :-
  format('Stan: ~w\tOcena: ~w\n', [State, Score]),
  show_states(Rest).
```

Tablica ósemkowa

```
% znajduje zbiorowa roznice miedzy zbiorami rozwiazania
getDiffs(PosTable1, PosTable2, Result) :-
  findall(newpos(Id, Position, NewPosition),
    (member(pos(Id, Position), PosTable1),
```

```
     (member(pos(Id, NewPosition), PosTable2)),
     (Position \== NewPosition)
    ),
    Result).

succ([pos(0, EmptyPos)|TilePositions], Diffs, Cost,
     [pos(0, NewEmptyPos)|NewTilePositions]) :-
  find_neighbour(EmptyPos, TilePositions, NewEmptyPos, NewTilePositions),
  sum_of_distances([pos(0,EmptyPos)|TilePositions],
                   [pos(0,NewEmptyPos)|NewTilePositions], Cost),
  getDiffs([pos(0, EmptyPos)|TilePositions],
           [pos(0, NewEmptyPos)|NewTilePositions], Diffs).
```

Przykładowe wywołanie dla grafu

```
?- start_A_star(a,2,4,3,Path).
2/4
Licznik wyczerpany
Licznik wyczerpany
Aktualne stany:
Stan: f Ocena: 12
Stan: nil
Czy kontynuowac? y/n
|: n.
Licznik wyczerpany
Licznik wyczerpany
3/4
Licznik wyczerpany
Licznik wyczerpany
Aktualne stany:
Stan: e Ocena: 10
Stan: f Ocena: 12
Stan: nil
Czy kontynuowac? y/n
|: y.
Licznik wyczerpany
Path = path_cost([nil/a, ab/b, bg/g, gm/m], 8) .
```

Przykładowe wywołanie dla tablicy ósemkowej

```
start_A_star(
    [pos(0 , 2/2), pos(1 , 1/3), pos(2 , 2/3),
    pos(3 , 3/3), pos(4 , 1/2), pos(5 , 2/1),
    pos(6 , 3/2), pos(7 , 1/1), pos(8 , 3/1) ], Path).
path_cost([nil/[pos(0,2/2),pos(1,1/3),pos(2,2/3),
                pos(3,3/3),pos(4,1/2),pos(5,2/1),
                pos(6,3/2),pos(7,1/1),pos(8,3/1)],
          [newpos(0,2/2,2/1),newpos(5,2/1,2/2)]/
          [pos(0,2/1),pos(1,1/3),pos(2,2/3),
           pos(3,3/3),pos(4,1/2),pos(5,2/2),
           pos(6,3/2),pos(7,1/1),pos(8,3/1)],
          [newpos(0,2/1,3/1),newpos(8,3/1,2/1)]/
          [pos(0,3/1),pos(1,1/3),pos(2,2/3),
           pos(3,3/3),pos(4,1/2),pos(5,2/2),
           pos(6,3/2),pos(7,1/1),pos(8,2/1)]],4)
```

Etap 2

```
search_A_star(Queue, ClosedSet, PathCost) :-
  fetch(Node, Queue, ClosedSet, RestQueue, NewClosedSet),
  continue(Node, RestQueue, NewClosedSet, PathCost).

continue(node(State, Action, Parent, Cost, _), _, ClosedSet, path_cost(Path, Cost)) :-
  goal(State), ! ,
  build_path(node(Parent, _, _, _, _) , ClosedSet, [Action/State], Path) .

continue(Node, RestQueue, ClosedSet, Path) :-
  expand(Node, NewNodes),
  insert_new_nodes(NewNodes, RestQueue, NewQueue),
  search_A_star(NewQueue, [Node | ClosedSet], Path).


fetch(node(State, Action,Parent, Cost, Score),
        [node(State, Action,Parent, Cost, Score) |RestQueue],
        ClosedSet, RestQueue, ClosedSet) :-
  \+ member(node(State, _, _, _, _), ClosedSet),  !.

fetch(Node, [node(State, Action, Parent, Cost, Score) | RestQueue],
  ClosedSet, FinalQueueRest, FinalClosedSet) :-
  member(node(State, _, _, Cost1, _), ClosedSet),
  Cost < Cost1,
  replace_node(node(State, Action, Parent, Cost, Score), ClosedSet, NewClosedSet),
  Diff is Cost1 - Cost,
  update_nodes_p_queue(State, Diff, ClosedSet, QueueRest, [], NewQueueRest),
  update_nodes(State, Diff, ClosedSet, NewClosedSet, NewClosedSetUpdated),
  fetch(Node, NewQueueRest, NewClosedSetUpdated, FinalQueueRest, FinalClosedSet),
  !.

fetch(Node, [ _ | RestQueue], ClosedSet, NewRest, NewClosedSet):-
  fetch(Node, RestQueue, ClosedSet, NewRest, NewClosedSet).

replace_node(node(State, Action, Parent, Cost, Score),
            [node(State, _, _, _, _)|Set],
            [node(State, Action, Parent, Cost, Score)|Set]):-
    !.

replace_node(Node, [N|Set], [N|NewSet]):-
    replace_node(Node, Set, NewSet).

update_nodes(_, _, _, [], []).

update_nodes(RootState, Diff, ClosedSet, [Node|Set], [UpdatedNode|NewSet]):-
    update_node(RootState, Diff, ClosedSet, Node, UpdatedNode),
    update_nodes(RootState, Diff, ClosedSet, Set, NewSet).

update_nodes_p_queue(_, _, _, [], Result, Result).

update_nodes_p_queue(RootState, Diff, ClosedSet, [Node|QueueRest], PartialResult, Result):-
    update_node(RootState, Diff, ClosedSet, Node, UpdatedNode),
    insert_p_queue(UpdatedNode, PartialResult, PartialResult1),
    update_nodes_p_queue(RootState, Diff, ClosedSet, QueueRest, PartialResult1, Result).

update_node(RootState, Diff, ClosedSet,
        node(State,Action,Parent,Cost,Score),
```

```
        node(State,Action,Parent,NewCost,NewScore)):-
    is_ancestor(node(State, Action, Parent, Cost, Score), RootState, ClosedSet),
    NewCost is Cost - Diff,
    NewScore is Score - Diff, !.

update_node(_, _, _, Node, Node).

is_ancestor(node(_, _,RootState, _, _), RootState, _):- !.
is_ancestor(node(_, _, Parent, _, _), RootState, ClosedSet):-
    member(node(Parent, Ac, Pa, Co, Sc), ClosedSet),
    is_ancestor(node(Parent, Ac, Pa, Co, Sc), RootState, ClosedSet).
```