11. Graphs Search Algorithms

This session covers graph search algorithms in Prolog: depth-first search, breadth-first search and best-first search.

11.1 Depth-first search

As in the previous session, we shall employ the *edge-clause* form for graph representation. Since *depth-first search* is the mechanism employed by the Prolog engine, all path predicates from the previous session employed a DFS search strategy. Below you have the predicates which implement a DFS search from a source node (it explores the connected component of the source node):

```
%d_search(Start, Path)

d_search(X,_):- df_search(X,_).
d_search(_,L):- collect_v([],L).

df_search(X,L):-
                asserta(vert(X)),
                edge(X,Y),
                \+(vert(Y)),
                df_search(Y,L).

collect_v(L,P):-retract(vert(X)),!, collect_v([X|L],P).
collect_v(L,L).
```

Exercise 11.1: Trace the execution of the d_search/2 predicate on an example graph.
Exercise 11.2: Add a stopping condition to predicate df_search/2, and trace its execution on an example graph. What do you observe?

11.2 Breadth-first search

The *BFS* strategy employs a queue to keep track of the expansion order of the nodes. In each step, a new node is read from the queue and is expanded – i.e. all its unvisited neighbors are added to the queue, in turn. Below you may find the specification for the necessary predicates:

```
%do_bfs(Start, Path)

do_bfs(X, Path):-assertz(q(X)), asserta(vert(X)), bfs(Path).

bfs(Path):- q(X), !, expand(X), bfs(Path).
bfs(Path):- collect_v([],Path).
expand(X):-     edge(X,Y),
                \+(vert(Y)),
                asserta(vert(Y)),
                assertz(q(Y)),
                fail.
expand(X):- retract(q(X)), !.
```

Exercise 11.3: Trace the execution of the do_bfs/2 predicate on an example graph.

11.3 Best-first search

The *Best-first search* algorithm is an informed greedy search strategy, in that it employs a heuristic to estimate the cost of the path from the current node to the target node. In each step, the algorithm selects the node having the smallest estimated distance to the target node (via the heuristic function).

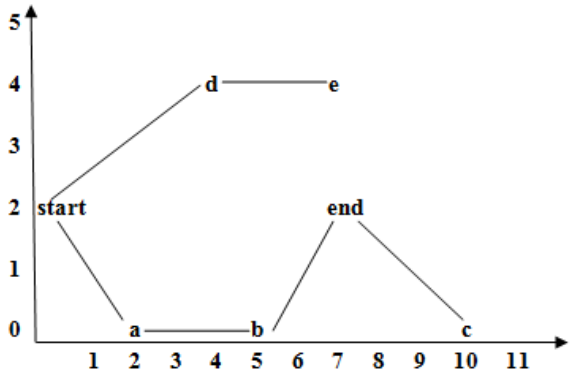

Figure 11.1: An example graph for the best-first search algorithm

The graph above can be represented using a variation of the *neighbor list-clause* form, as:

```
pos_vec(start,0,2,[a,d]).
pos_vec(a,2,0,[start,b]).
pos_vec(b,5,0,[a,c,end]).
pos_vec(c,10,0,[b,end]).
pos_vec(d,3,4,[start,e]).
pos_vec(e,7,4,[d]).
pos_vec(tinta,7,2,[b,c]).

is_target(end).
```

The end node is specified as being the target node, using a predicate clause. The predicate specifications are presented below:

```
dist(Nod1,Nod2,Dist):-pos_vec(Nod1,X1,Y1,_),pos_vec(Nod2,X2,Y2,_),
                      Dist is (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2).

order([Nod1|_],[Nod2|_]):- is_target(Target),
                      dist(Nod1,Target,Dist1),
                      dist(Nod2,Target,Dist2),
                      Dist1<Dist2.

best([],[]):-!.
best([[Target|Rest]|_],[Target|Rest]):-is_target (Target),!.
best([[H|T]|Rest],Best):-pos_vec(H,_,_,Vec),
```

```
                    expand(Vec,[H|T],Rest,Exp),
                    q(Exp,SortExp,[]),
                    best(SortExp,Best).

expand([],_,Exp,Exp):-!.
expand([E|R],Cale,Rest,Exp):-\+(member(E,Cale)),!,
                    expand(R,Cale,[[E|Cale]|Rest],Exp).
expand([_|R],Cale,Rest,Exp):-expand(R,Cale,Rest,Exp).

partition(H,[A|X],[A|Y],Z):-
            order(A,H),!,partition(H,X,Y,Z).
partition(H,[A|X],Y,[A|Z]):-partition(H,X,Y,Z).
partition(_,[],[],[]).

q([H|T],S,R):-
        partition(H,T,A,B),
        q(A,S,[H|Y]),
        q(B,Y,R).
q([],S,S).
```

Exercise 11.4: Trace the execution of the following query:

```
?- best([[start]], Best).
```

11.3 Quiz exercises

11.3.1 Write a predicate which perform DLS – Depth-Limited Search on a graph.