Search (4A)

Young W. Lim
12/26/13

Please send corrections (or suggestions) to youngwlim@hotmail.com.

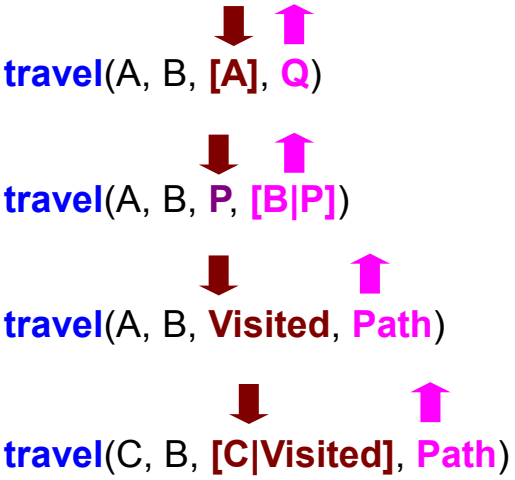
This document was produced by using OpenOffice.

Graph Structures and Paths

```
edge(1,2).
edge(1,4).
edge(1,3).
edge(2,3).
edge(2,5).
edge(3,4).
edge(3,5).
Edge(4,5).


edge(X,Y) :- edge(Y,X).

connected(X,Y) :- edge(X,Y) ; edge(Y,X).

connected(X,Y) :- edge(X,Y).
connected(X,Y) :- edge(Y,X).
```

Graph Structures and Paths

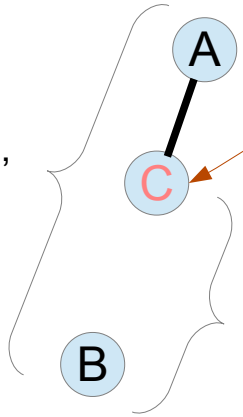
```
path(A, B, Path) :-
    travel(A, B, [A], Q),
    reverse(Q, Path).

travel(A, B, P, [B|P]) :-
    connected(A, B).

travel(A, B, Visited, Path) :-
    connected(A, C),
    C \== B,
    \+member(C, Visited),
    travel(C, B, [C|Visited], Path).
```



Graph Structures and Paths

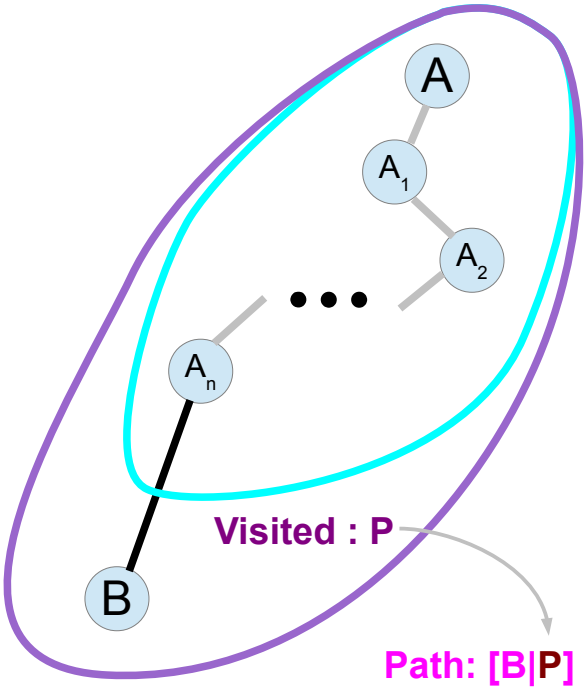
```
path(A, B, Path) :-
        travel(A, B, [A], Q),
        reverse(Q, Path).

travel(A, B, P, [B|P]) :-
        connected(A, B).

travel(A, B, Visited, Path) :-
        connected(A, C),
        C \== B,
        \+member(C, Visited),
        travel(C, B, [C|Visited], Path).
```

travel(A, B, [A], Q),



connected(A, C),
C \== B,
\+member(C, Visited),

travel(C, B, [C|Visited], Path).



Search

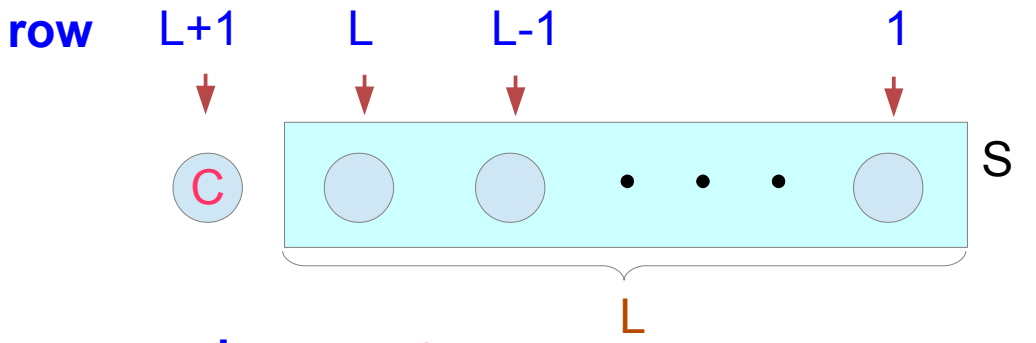
```
solve(P) :-
    start(Start),
    search(Start, [Start], Q),
    reverse(Q, P).

search(S, P, P) :- goal(S), !.          /* done                */

search(S, Visited, P) :-
    next_state(S, Nxt),                 /* generate next state */
    safe_state(Nxt),                    /* check safety        */
    no_loop(Nxt, Visited),              /* check for loop      */
    search(Nxt, [Nxt|Visited], P).      /* continue searching... */
```

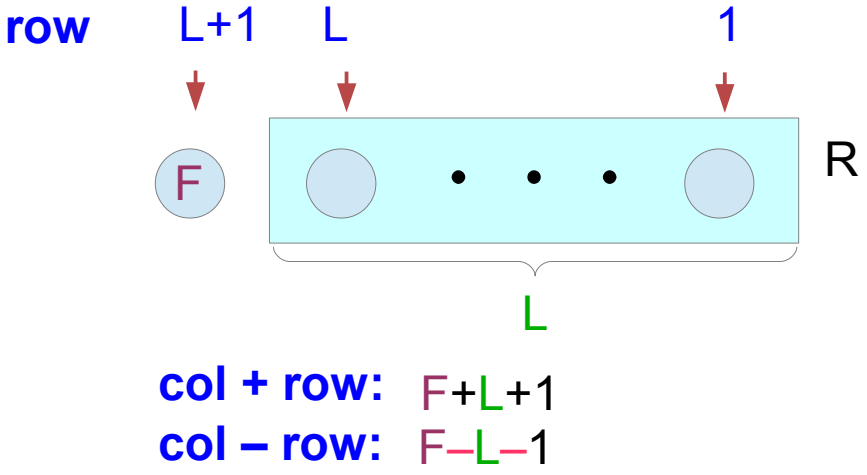
Search

```
start([]).

goal(S) :- length(S,8).

next_state(S, [C|S]) :-
        member(C, [1,2,3,4,5,6,7,8]),
        not member(C, S).


no_loop(Nxt, Visited) :-
       \+member(Nxt, Visited).
```

Search

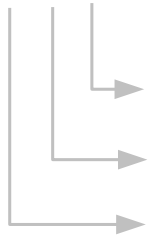
```
safe_state([C|S]) :-
                length(S,L),
                Sum is C+L+1,
                Diff is C–L–1,
                safe_state(S, Sum, Diff).

safe_state([],_,_) :- !.

safe_state([F|R],Sm,Df) :-
                  length(R,L),
                  X is F+L+1,
                  X \= Sm,
                  Y is F–L–1,
                  Y \= Df,
                 safe_state(R, Sm, Df).
```





8 Queen Puzzle

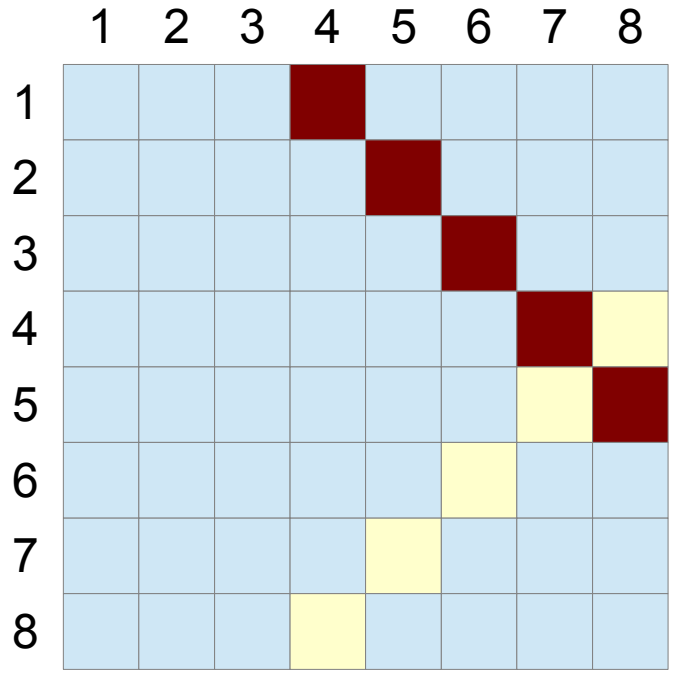
L=[7,4,2] already chosen for k rows (length L = 3)



column 2 for the 3rd row
column 4 for the 2nd row
column 7 for the 1st row

[C|L] choosing C for the 1st row

8 Queen Puzzle



on the same / diagonal iff
the **sum** of the row and column is the same

on the same \ diagonal iff
the **difference** of the row and column is the same

A* Algorithm

```
solve(Start,Soln) :- f_function(Start,0,F),
                     search([Start#0#F#[]],S),
                     reverse(S,Soln).

f_function(State,D,F) :- h_function(State,H),
                         F is D + H.


 search([State#_#_#Soln | _], Soln) :- goal(State).
 search([B|R],S) :- expand(B, Children),
                    insert_all(Children, R, NewOpen),
                    search(NewOpen,S).

 expand(State#D#_#A, All_My_Children) :-
          bagof(Child#D1#F#[Move|A],
               ( D1 is D + 1,
                 move(State,Child,Move),
                 f_function(Child,D1,F) ) ,
               All_My_Children).
```

A* Algorithm

```
insert_all([F|R],Open1,Open3) :- insert(F,Open1,Open2),
                                 insert_all(R,Open2,Open3).
insert_all([],Open,Open).

insert(B,Open,Open) :- repeat_node(B,Open), ! .
insert(B,[C|R],[B,C|R]) :- cheaper(B,C), ! .
insert(B,[B1|R],[B1|S]) :- insert(B,R,S), !.
insert(B,[],[B]).

repeat_node(P#_#_#_, [P#_#_#_|_]).

cheaper( _#_#H1#_ , _#_#H2#_ ) :- H1 <  H2.
```

References

[1] en.wikipedia.org
[2] en.wiktionary.org
[3] U. Endriss, “Lecture Notes : Introduction to Prolog Programming”
[4] http://www.learnprolognow.org/ Learn Prolog Now!
[5] http://www.csupomona.edu/~jrfisher/www/prolog_tutorial
[6] www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/intro.html
[7] www.cse.unsw.edu.au/~billw/dictionaries/prolog/negation.html