A* and Graph Search

CS 367

Search & Tree/Graph Search

- IDA* is always implemented as a tree search algorithm. Why?
- A* is almost always implemented as a graph search algorithm. Why?
- (What is the difference between a tree search and a graph search?)

Open List

- A* always needs an Open List
- Open List is used for selecting which node to expand next.
- Usually implemented as a priority queue (aka heap)

Closed List

- Closed list used for keeping track of best paths found so far (so can return the solution path).
- Closed list used to detect duplicate states.
- Closed list usually implemented as hash table (key being the state description).

Node List

- In a tree search, there may be many nodes with the same state.
- In a graph search, only one node can have a particular state.
- It may be possible to combine the open and closed lists into a new list (the Node list).

Adding Nodes to Node List

define in Prolog the *updated* relationship between a node list, NL, along with a node, N, and a node list, UNL. This relationship is defined formally as follows:

Given that a node list never has two nodes that have identical states, that N is never an element of NL, that NL and UNL are sets, and that *state* and *fValue* are functions that take a node as an argument and return (respectively) the state description or the *f* value of that node

$$
\begin{aligned}
&updatedNodeList(NL, N, UNL) \Leftrightarrow \\
&\quad ((\forall N1 \in NL\ state(N) \neq state(N1)) \\
&\qquad \rightarrow UNL = NL \cup \{N\}) \\
&\vee\ ((\exists N1 \in NL\ state(N) = state(N1)\ \&\ \\
&\quad fValue(N) \geq fValue(N1)) \\
&\qquad \rightarrow UNL = NL) \\
&\vee\ ((\exists N1 \in NL\ state(N) = state(N1)\ \&\ \\
&\quad fValue(N) < fValue(N1)) \\
&\qquad \rightarrow UNL = \{N\} \cup NL - \{N1\})
\end{aligned}
$$

```
:- use_module(library(lists)). % for member/2 and delete/3 predicates
not(P) :- P, !, fail.
not(_).

/* node accessor and setter predicates
   ================================

   node data structure: node(State, ShouldExpand, FValue, ParentState)
*/
node(node(State, ShouldExpand, FValue, ParentState),
     State, ShouldExpand, FValue, ParentState).
state(node(State, _,_,_), State).
shouldExpand(node(_, ShouldExpand,_,_), ShouldExpand).
fValue(node(_,_, FValue,_), FValue).
parentState(node(_,_,_,ParentState), ParentState).
```

updatedNodeList/3

```
/* updatedNodeList(+NodeList, +Node, -UpdatedNodeList)

    ==========================================
    ===
*/
% put your code here, you only need to write a clause per case
% you will only need the predicates defined above
% and the member/2 and delete/3 predicates defined in the lists
% library loaded above
```

member/2

- member(Element, List)

delete/3

- delete(List1, Element, List2)