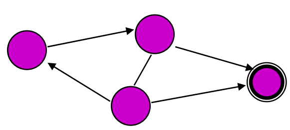
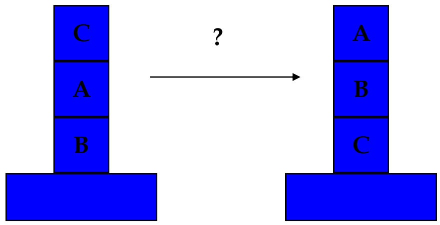
State Space Search in Prolog

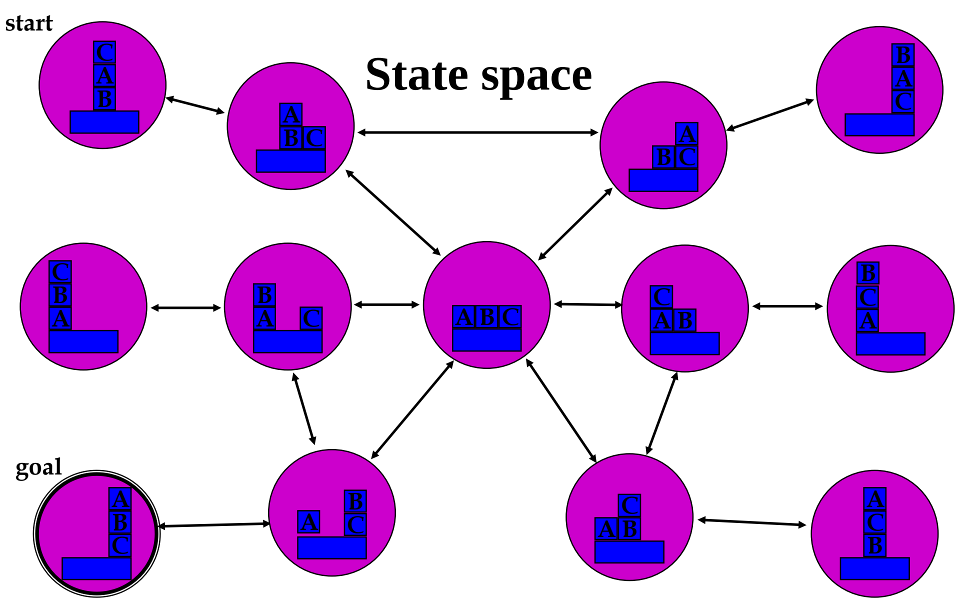
Introduction

- One of the most powerful approaches to problem solving in AI
- Represent problem as
  - a set of states
  - a start state
  - a set of legal moves between states
  - a set of goal states or a goal condition
- Convenient to represent as a directed graph

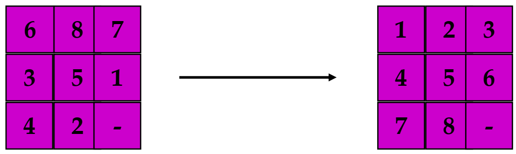


Block Stacking





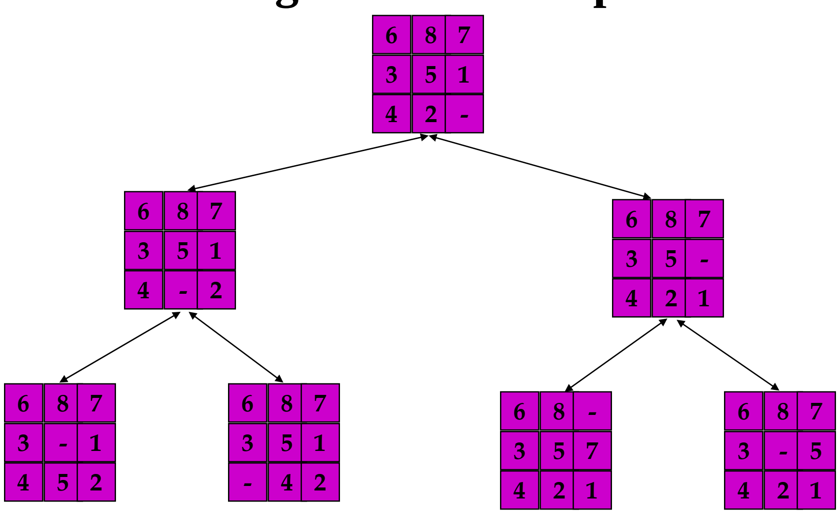
Sliding Tiles Problem
8-Puzzle



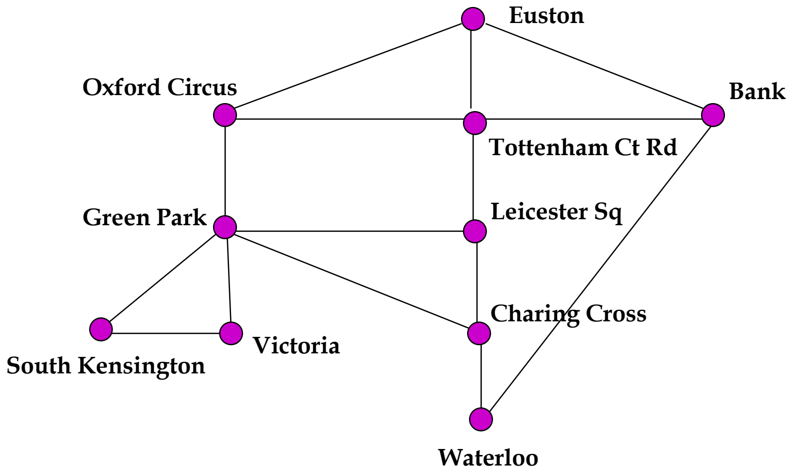
start state **goal state**

362,880 possible states!

A fragment of state space



Route Finding



Get from Euston to Waterloo

Other Examples

Problem	Nodes/States	Arcs/Moves
chess	board positions	legal moves
transport scheduling	candidate schedules	operators modifying schedules

State space search using Prolog

- Represent each arc by a relation:

```
s(X,Y)   or   move(X,Y)
```

  meaning there is a legal move from node X to node Y or state X to state Y. Can say Y is a successor state to X.

- Remember to include moves in both directions.

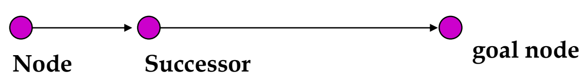
- Assume a predicate

```
goal(X)
```

  that is true if X is a goal node.

The solve predicate

- Want predicate solve(Node, List) that takes a start node Node and returns a list of nodes List on the route to a goal node, if such a route exists.
- Two cases
  - Node is already a goal node
  - if a route exists to a goal node from a successor of Node, then tacking Node onto this route gives a solution.



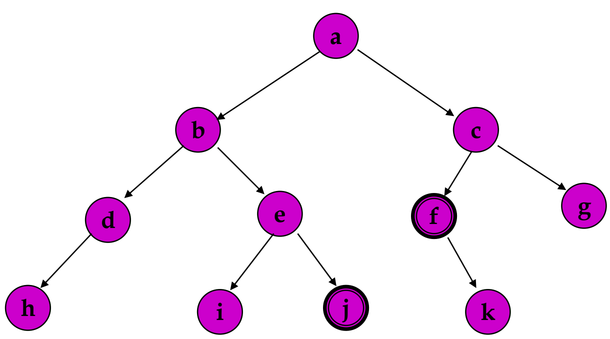
Writing this in Prolog

- `solve(Node,[Node]) :-`
  `goal(Node).`

- `solve(Node,[Node|Sol]) :-`
  `s(Node,Successor),`
  `solve(Successor,Sol).`

- Note: instead s(Node, Successor) we usually write
  something like: move(State, NextState)

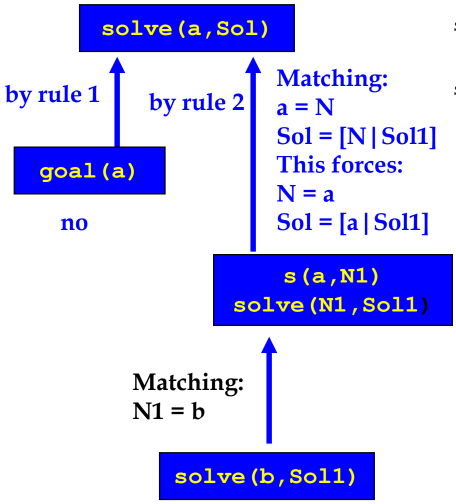
An example




```
s(a,b).
s(a,c).
s(b,d).
s(b,e).
s(c,f).
etc...

goal(j).
goal(f).
```

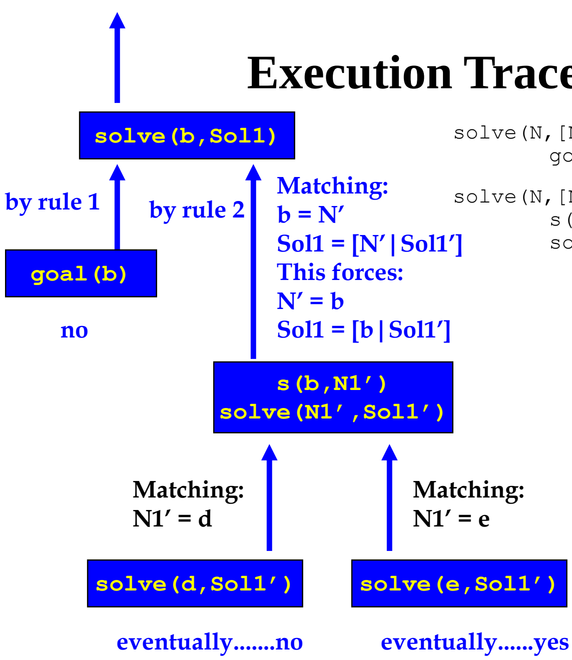
?- `solve(a,Sol).`

Execution Trace - 1



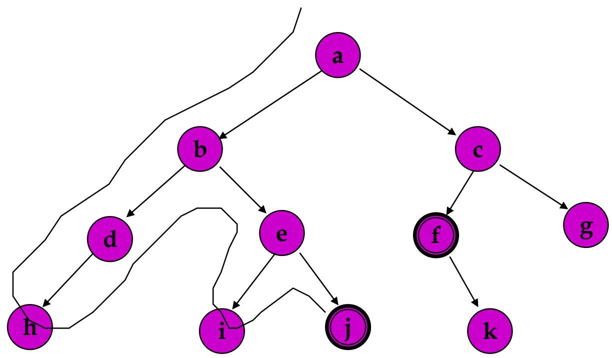

```
solve(N,[N]) :-                     rule 1
        goal(N).

solve(N, [N|Sol1]) :-               rule 2
        s(N, N1),
        solve(N1, Sol1 ).
```

Execution Trace - 2




```
solve(N,[N]) :-                     rule 1
        goal(N).

solve(N,[N|Sol1]) :-                rule 2
        s(N,N1),
        solve(N1,Sol1).
```

Search path



visits nodes in order: a, b, d, h, e, i, j

depth-first search

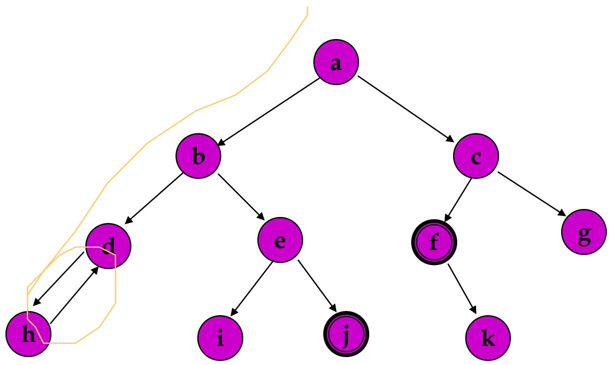
First attempt in Prolog

```
% Naive approach
move(a,b).
move(a,c).
move(b,d).
move(b,e).
move(c,f).
move(c,g).
move(d,h).
move(e,i).
move(e,j).
move(f,k).

goal(j).
goal(f).
```

```
solve(State, [State]) :-
    goal(State).

solve(State, [State|Sol]) :-
    move(State, NewState),
    solve(NewState, Sol).
```

```
1 ?- solve(a, Sol).
Sol = [a, b, e, j] ;
Sol = [a, c, f] ;
false.

2 ?- solve(d, Sol).
false.

3 ?-
```

Circular paths



Deal with this by recording where we've been

Our Prolog code goes into infinite loop with circular path

```
% Naive approach

move(a,b).
move(a,c).
move(b,d).
move(b,e).
move(c,f).
move(c,g).
move(d,h).
move(h,d).
move(e,i).
move(e,j).
move(f,k).

start(a).
goal(j).
goal(f).
```

```
solve(State, [State]) :-
    goal(State).

solve(State, [State|Sol]) :-
    move(State, NewState),
    solve(NewState, Sol).
```

```
1 ?- solve(a, Sol).
ERROR: Out of global stack
2 ?-
```

Prolog code to prevent looping

No looping, Sol not instantiated until solve/3 succeeds. Sol is then Path.

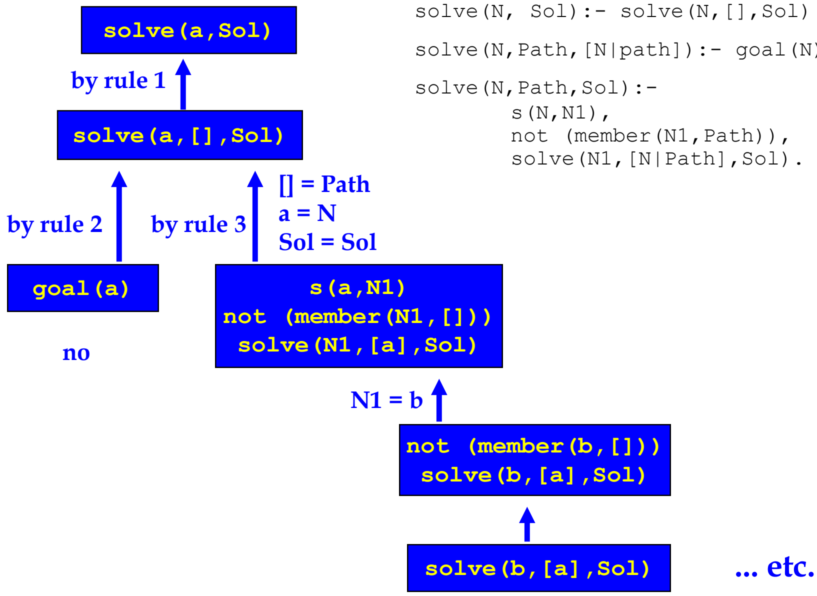
```
% solve(State, Path, Sol)
solve(State, Path, [State|Path]) :-
    goal(State).

solve(State, Path, Sol) :-
    move(State, NewState),
    not(member(NewState, Path)),
    solve(NewState, [State|Path], Sol).

solve(X, Soln) :-
    solve(X, [], Sol),
    reverse(Sol, Soln).
```

Prolog code to prevent looping

```
solve(N,Sol):- solve(N,[],Sol).

solve(N, Path,[N|Path]):- goal(N).

solve(N,Path,Sol):-
      s(N,N1),
      not (member(N1,Path)),
      solve(N1,[N|Path],Sol).
```

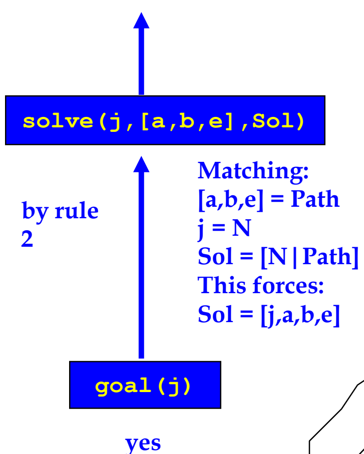
Execution Trace - 1

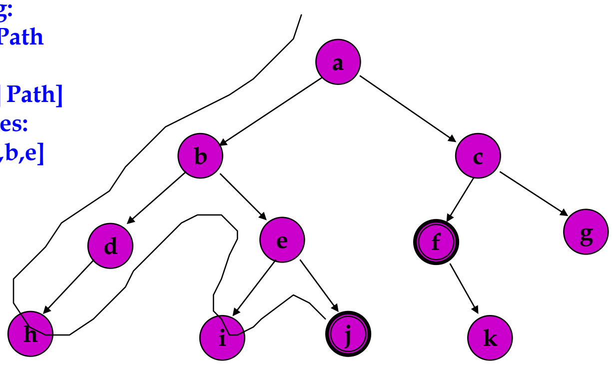
```
solve(N, Sol):- solve(N,[],Sol).              1
solve(N,Path,[N|path]):- goal(N).             2
solve(N,Path,Sol):-                           3
        s(N,N1),
        not (member(N1,Path)),
        solve(N1,[N|Path],Sol).
```



Execution Trace - 2

```
solve(N, Sol):- solve(N,[],Sol).              1
solve(N,Path,[N|path]):- goal(N).             2
solve(N,Path,Sol):-                           3
        s(N,N1),
        not (member(N1,Path)),
        solve(N1,[N|Path],Sol).
```





Lab Work

- Use Prolog state space search code outlined here to find solution paths for
  - farmer, wolf goat and cabbage problem
  - missionaries and canibals problem
- You only need to write the correct code for s(N,N1) which we write as move(N,N1).

- Hint: Instead of

```
move(a,b).
```

use code like

```
move(state(X,X,G,C), state(Y,Y,G,C)) :-
        opp(X,Y), not(unsafe(Y,Y,G,C)).
```