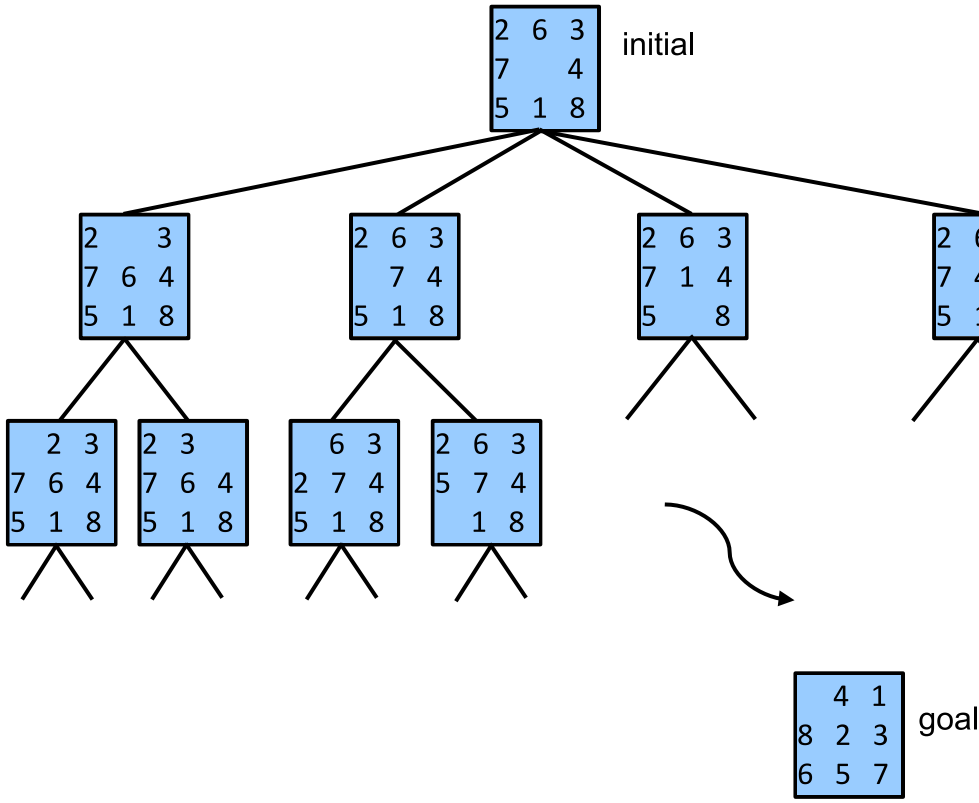
State space search



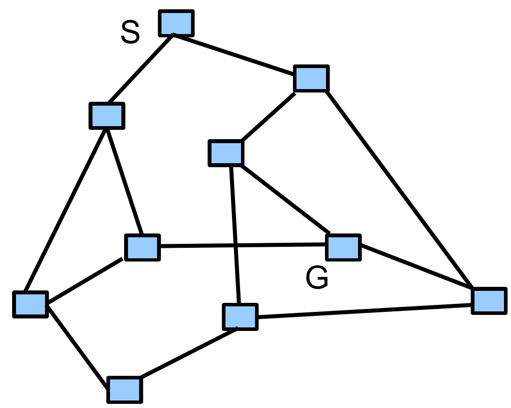
- How to find a path from initial board to goal board

- State space search
    - start at the initial state
    - search the states in a systematic order
    - stop when the goal state is found
    - display the path from initial state to goal state

- Two types of state space search
    - uninformed / blind search
    - informed / heuristic search

Applications of state space search

- State space search may be used in problems that involve
    - multiple states
    - actions that change states
    - initial state and goal state
    - finding path from initial to goal



- Some applications
    - board puzzles
    - two player board games
    - automous vehicle driving
    - robot task planning

Uninformed / blind search

- Search is done blindly

- Search does not use any problem specific information

- Search may take long time

- Uninformed search algorithms
    - breadth first search
    - depth first search
    - depth limited, iterated, bidirectional search
    - uniform cost search

Informed / heuristic search

- Search is done intelligently

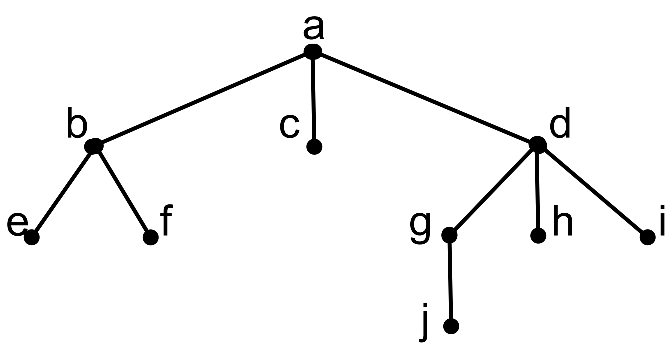
- Search uses some problem specific information

- Search is likely to run fast

- Informed search algorithms
    - best first search
    - A* search

Breadth first search on tree

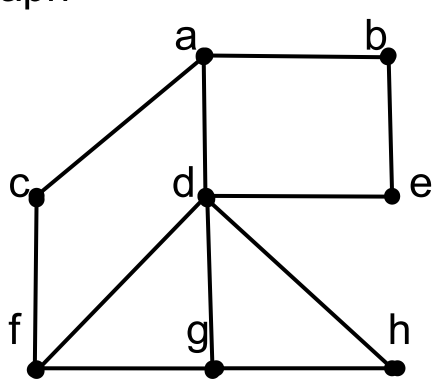
- Tree



- Breadth first search order - a b c d e f g h i j

Breadth first search on graph

- Graph



- Initial state - a

- Breadth first search order - a c d b f g h e

Breadth first search algorithm

```
- open list has initial state
  closed list is empty
  while open list is not empty
  {
        S = remove the first state from open list
        insert S into closed list
        if S is goal state
             display path from initial state to S
             stop search
        else
        {
              generate successors of S
              for each successor C
                    if C is not in open list and C is not in closed list
                          insert C at the end of open list
        }
  }
  say there is no path
```

- Open list is a queue, operates in first in first out order, FIFO

- Closed list is not needed if search is done on tree

- Run time $O(b^d)$, space usage $O(b^d)$, b is branching factor, d is depth of goal state

Open/closed lists in breadth first search

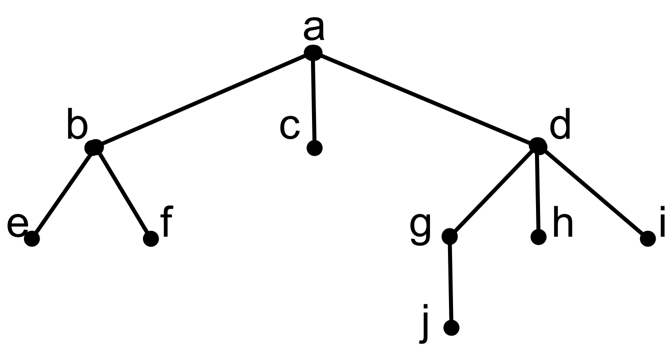
- Open and closed lists for the tree shown above

open list	look	closed list
a	a	

b c d	b	a
c d e f	c	a b
d e f	d	a b c
e f g h i	e	a b c d
f g h i	f	a b c d e
g h i	g	a b c d e f
h i j	h	a b c d e f g
i j	i	a b c d e f g h
j	j	a b c d e f g h i
		a b c d e f g h i j

Depth first search on tree

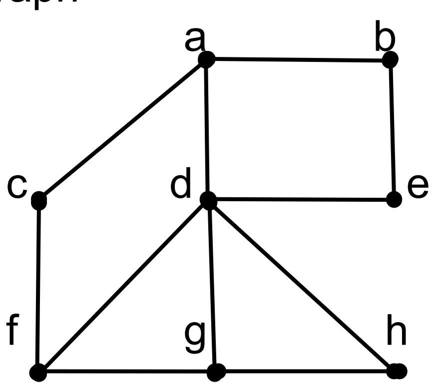
- Tree



- Depth first search order - a b e f c d g j h i

Depth first search on graph

- Graph



- Initial state - a

- Depth first search order - a c f d e b h g

Depth first search algorithm

- open list has initial state

```
closed list is empty
while open list is not empty
{
      S = remove the first state from open list
      insert S into closed list
      if S is goal state
           display path from initial state to S
           stop search
      else
      {
            generate successors of S
            for each successor C
                  if C is not in open list and C is not in closed list
                        insert C at the beginning of open list
      }
}
say there is no path
```

- Open list is a stack, operates in last in first out order, LIFO

- Closed list is not needed if search is done on tree

- Run time $O(b^m)$, space usage $O(b^m)$ if search is done in graph
  run time $O(b^m)$, space usage $O(bm)$ if search is done in tree
  b is branching factor, m is depth of tree

Open/closed lists in depth first search

- Open and closed lists for the tree shown above

open list	look	closed list
a	a	
b c d	b	a
e f c d	e	a b
f c d	f	a b e
c d	c	a b e f
d	d	a b e f c
g h i	g	a b e f c d
j h i	j	a b e f c d g
h i	h	a b e f c d g j
i	i	a b e f c d g j h
		a b e f c d g j h i

Depth limited, iterated, bidirectional search algorithms

- Depth limited search
  - choose a depth limit
  - when the depth limit is reached, do not create successors
  - depth limit depends on time/space constraints

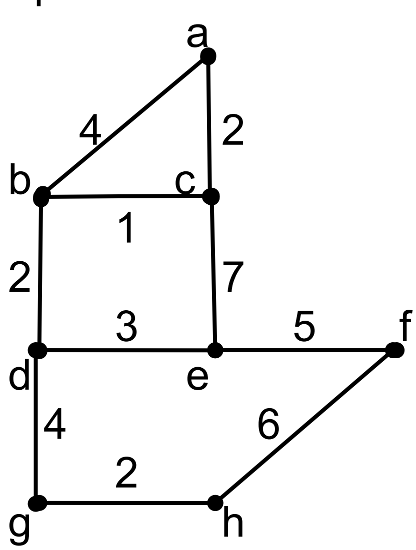
- Iterated depth limited search
  - do depth limited search with limit = 1, limit = 2, limit = 3, limit = 4, and so on
  - number of iterations depends on time/space constraints

- Bidirectional search
  - search forward from initial to goal
  - search backward from goal to initial
  - stop when forward and backward searches meet

Uniform cost search on graph

- Finding shortest path from initial state to goal state

- Graph



- Initial state - a
  goal state - e

- Shortest path - a c b d e
  cost - 8

- open list	look	closed list
a/0	a	
b/4 c/2	c	a
b/3 e/9	b	a c
e/9 d/5	d	a c b
e/8 g/9	e	a c b d
g/9		a c b d e

Uniform cost search algorithm

- open list has initial state with cost = 0

```
closed list is empty
while open list is not empty
{
     S = remove the minimum cost state from open list
     insert S into closed list
     if S is goal state
          display path from initial state to S
          stop search
     else
     {
          generate successors of S
          for each successor C
               cost of C = cost of S + edge cost from S to C
               if C is not in closed list
               {
                    if C is not in open list
                         insert C into open list
                    else
                    {
                         compare costs of new copy of C and
                         old copy of C in open list
                         if new cost < old cost
                              replace old copy of C in open list with
                              new copy of C
                    }
               }
     }
}
say there is no path
```

- If all edges have the same cost then the uniform cost search behaves like the breadth first search