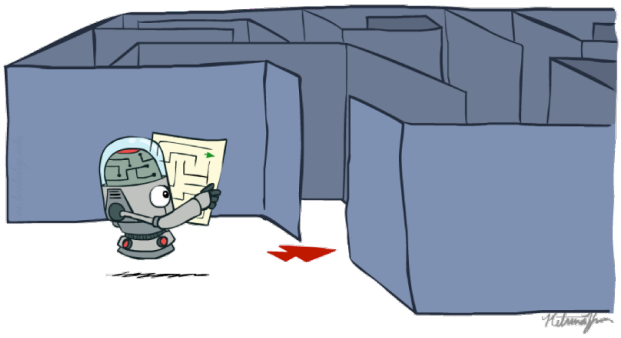
CS440 – Uninformed Search



Purpose: Consider a problem where you can choose from a set of actions in different situations and you want to find a sequence of actions which will solve the problem. An example of this is trying to find a path through a maze. This module will demonstrate techniques for searching for solutions of such problems.

The purpose of this module is to present algorithms for uninformed searches.

Knowledge: This module will help you become familiar with the following content knowledge:

- The components of a Search Problem
- The difference between a World and Search spaces
- Tree and graph search algorithms

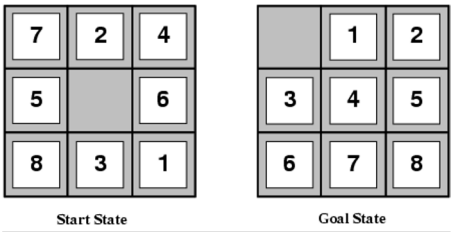
Activity1 - Problems and Search Spaces:
With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 20 minutes.

1. A **Problem** has four components:

   - an initial state
   - actions or successor function S(x) = set of state-action pairs
   - goal test
   - path cost

   A solution is a sequence of actions leading from the initial state to a goal state.

   Consider the 8-puzzle problem where you can slide tiles around on a square grid to go from one configuration to a goal configuration. Clearly you can only slide a tile if there is a blank space next to it.



   What set of actions would we have for the initial state?

   The Pathing Problem determines a path in the Pacman maze from one location to another:

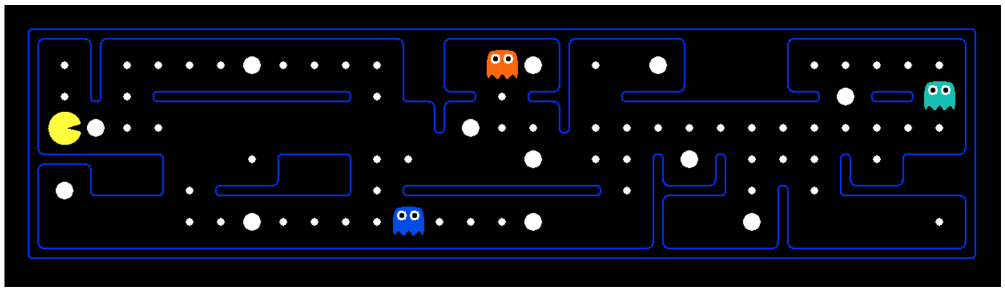
   - States: (x,y) location
   - Actions NSEW (moving North, South, East, West)
   - Successor: update location
   - Goal Test: Is (x,y)=END?

   Complete the pieces for the problem to eat all the dots in the Pacman world:

   - States: _____
   - Actions NSEW
   - Successor: _____
   - Goal Test: _____

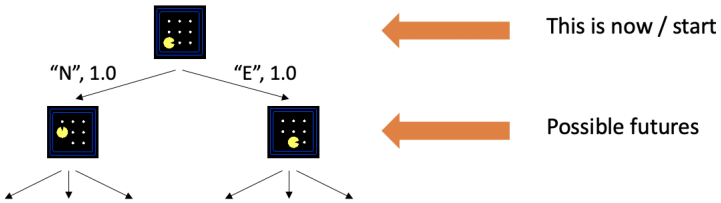
2. A **World Space** includes every last detail of the environment. Whereas a **Search Space** is an abstraction that includes only the details needed for planning.

   Consider the Safe Passage Problem to eat all the dots while keeping the ghosts perma-scared. Eating a power pellet (the large dots) puts the ghosts in scared mode in which they don't chase the Pacman for a specified time period.



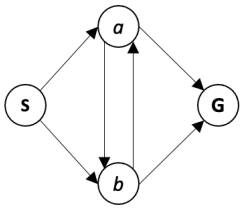
   What does the search state need to include?

3. A **Search Tree** is a "what if" tree of plans and their outcomes. The start state is the root of the tree and children correspond to the successors. Nodes show states, but actually correspond to PLANS that achieve those states.



   Do we want to build the whole tree for Pacman? Why or why not?

   Consider the following 4 state graph. Draw a few levels of the search tree from S which looks for a path to the goal G.



   How big is the search tree from S?

Activity2 - Tree Search Algorithms:
With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 30 minutes.

1. We will examine the **General Tree Search Algorithm** more closely.

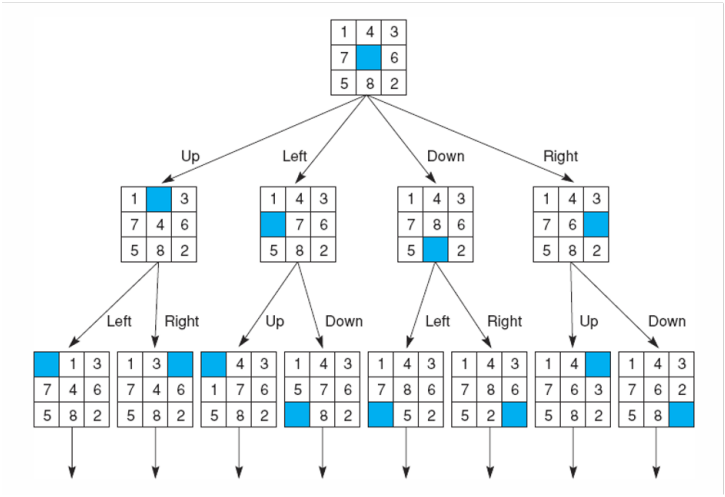
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

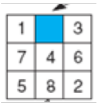
   The important ideas in this algorithm is that we have a *fringe* of nodes in the tree that have not been explored yet. We *expand* a node by generating its successors and we use some *exploration strategy* to decide the order in which fringe nodes are expanded.

   Give two or three ways in which to pick fringe nodes to be explored.

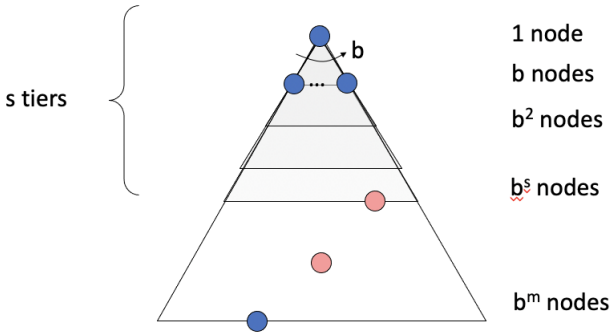
2. Breadth First Search uses a queue to store the fringe in the search.

   Consider a search tree for the 8-puzzle problem.



   In a Breadth-First Search, what state would be expanded after  ?

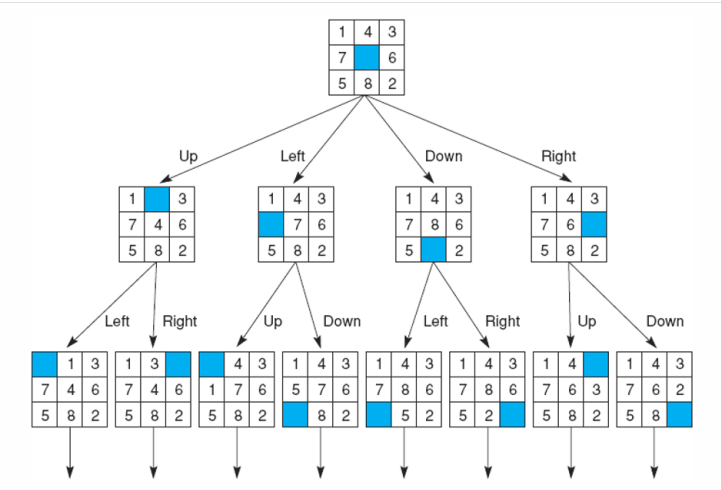
3. A Breadth-First Search processes all nodes above the shallowest solution.

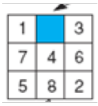


Let the depth of the shallowest solution be s. What would be the order of growth for the search time?
How much space does the fringe take?
Is this search complete (always finds a solution)?
Is this search optimal (always finds the solution with the best cost)?

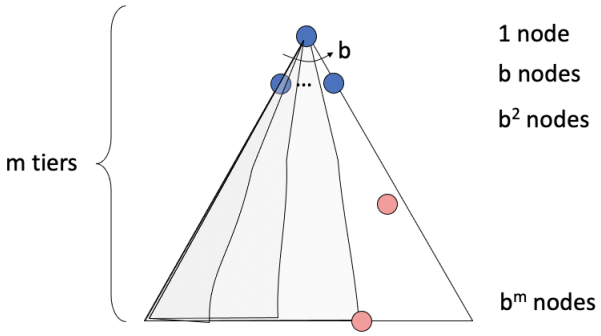
4. Depth-First Search uses a stack to store the fringe in the search.

Consider a search tree for the 8-puzzle problem.



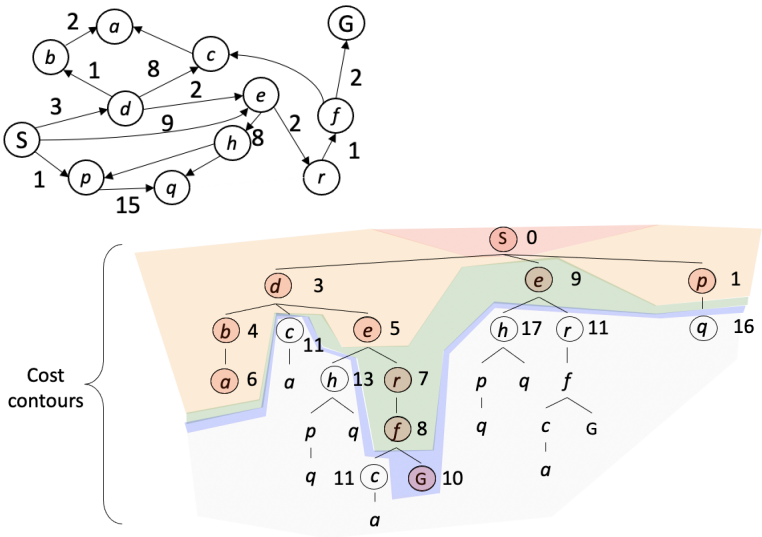
In a Depth-First Search, what state would be expanded after  ?

5. What nodes does Depth-First Search expand?

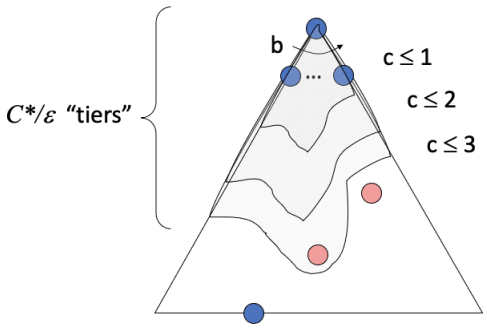


Let m be the greatest depth that is explored. If m is finite, what would be the order of growth for the search time?
How much space does the fringe take?
Is this search complete?
Is this search optimal?

6. Uniform Cost Search uses a priority queue to store the fringe in the search so that the cheapest node is expanded first. This results in cost contours in the search tree:

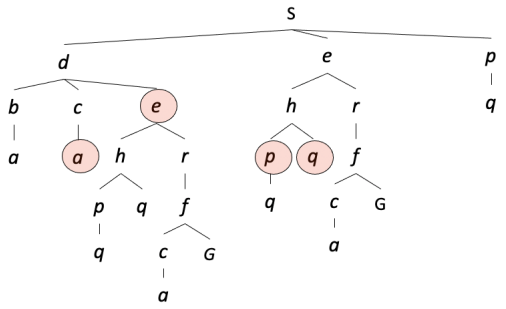


What nodes does Uniform Search expand?



In the diagram C^* is the solution cost and graph edges cost at least ϵ. What is the order of growth for the search time?
Is this search complete?
Do you think this search optimal? (proof will be coming later)

7. A **Graph Search** (as opposed to a Tree Search) is one that does not expand duplicate nodes. Consider this BFS example:



Why should we not bother the expand the circled nodes?
How can a graph search be implemented?