# Search

Agent: entity that perceives its environment and acts upon that environment

State: a configuration of the agent and its environment

Initial state:

Actions: ACTIONS(s) set of actions that can be executed in state s

Transition model: RESULT(s,a) returns the state resulting from performing action a in state s

State space: the set of all states reachable from the initial state by any sequence of actions

State space can be thought of a graph: nodes being state, actions being directional link

Goal test: way to determine whether a given state is a goal state

Path cost: numerical cost associated with a give path

Solution: a sequence of actions that leads from the initial state to a goal state

Optimal solution: a solution that has the lowest path cost

Node: a data structure that keeps track of

* a state
* a parent
* an action
* a path cost

Frontier: like a queue in the BFS, a data structure stores all next steps

Approach: (feels like bfs in graph)(**its almost always good to add a seen**, or we may have loop in graphs)

* start with a frontier that contains the initial state
* repeat:
  + if the frontier is empty, then no solution
  + remove a node from the frontier (if use stack, then dfs, if use queue, then bfs)
  + if node contains goal state, return the solution
  + add the node the explored set.
  + expand the node, add resulting nodes to the frontier

dfs will get a solution if maze is finite, but not always optimal solution

bfs will get optimal solution, but bfs will explore many states that may not be needed

Knowing the direction of the goal **usually** is a good thing, not **always**

Uninformed search: search strategy that uses no problem-specific knowledge: bfs,dfs

Informed search: uses problem-specific knowledge to find solutions more efficiently

Greedy best-first search: search algorithm that expands the node that is closest to the goal, as estimated by a heuristic function h(n): Manhattan distance.

heuristic is not a guaranty, but an estimate

A\* search: search algorithm that expands node with lowest value of g(n) +h(n)

* g(n) = cost to reach node
* h(n) = estimated cost to goal
* optimal if:
  + h(n) is admissible (never overestimates the true cost)
  + H(n) is consistent ( every node n and successor n' with step cost c, h(n)<=h(n') +c
* tends to use lots of memory

Adversary problems:

类似五子棋这种相互对抗的游戏

* minimax:
  + given the goal state is defined as -1,0,1
  + gamer1 wants to max score
  + gamer2 wants to min score so gamer1 lose

Game:

* S0: initial state
* Players(s): returns which player to move in state s
* actions(s): returns legal moves in the state s
* Result(S,A)
* Terminal(s): checks if states s is a terminal state
* Utility(S): final numerical value for terminal state s

Minimax:

* given a states:
  + MAX player will picks action a in ACTIONS(s) that produces highest value of MIN-VALUE(results(s,a));
  + 也就是说，maxplayer要从最小的数字里选最大的选项，因为minplayer会选最小的

Depth-limited minimax: limiting number of depth so won't go too deep into the tree.

* evaluation function: estimates the expected utility of the game