- W01
  - 3.1 Problem-Solving Agents
    - Search problems and solutions
    - Formulating problems
  - 3.2 Example problems
    - Standard problem
      - Grid world
      - Vacuum world
      - Sokoban puzzle
      - Sliding-tile puzzle
      - Knuth's 4
    - Real world problem
      - Route-finding problem
      - Touring problems
        - TSP
      - VLSI problem
      - Robot navigation
      - Automatic assembly sequencing
  - 3.3 Search Algorithms
    - Search data structures
      - Best-first search
      - Priority queue
      - FIFO: queue
      - LIFO: stack
    - Redundant paths
      - Graph search: check redundant
        - BFS
        - Remember all previously reached states
        - Rely on other mechanism to deal with long cycles
        - Check limited ancestor
      - Tree search: does not check redundant
        - Assembly problem
    - Measuring problem-solving performance
      - Completeness
      - Cost optimality
      - Time and Space
        - IVI
        - IEI
        - d: depth
        - m: max actions in any path
        - b: branching factor
  - 3.4 Uniformed Search Strategies
    - BFS
    - Dijkstra or uniform-cost search
    - Backtracking search
      - DFS
    - Depth-limited and iterative deepening search
    - Bidirectional search
    - Comparing

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

  - Informed(Heuristic) Search Strategies
    - **h(n)**: estimated cost of the cheapest path from the state at node n to a goal state
    - Greedy best first search
    - A* Search
      - Admissible heuristic: never overestimates
      - Prunes nodes that are not necessary for optimal
    - Satisficing search
      - Search contours
      - Weighted A*
    - Memory-bounded search
      - Beam search
      - IDA*
      - RBFS
      - SMA*
    - Bidirectional heuristic search
      - Bidirectional A*
  - Heuristic Functions
    - The effect of heuristic accuracy on performance
      - Effective depth
    - Generating heuristics from relaxed problems
    - Generating heuristics from sub-problems: pattern databases
      - Pattern database
      - Disjoint pattern databases
      - Difference to DP problem???
    - Generating heuristics with landmarks
    - Learning to search better
    - Learning heuristics from experience
  - State Space Search
    - All of AI is search
    - 4 key elements
      - States — node
      - Actions
      - Transition model — arc
      - Cost (action cost function)
    - Solution may not unique
    - Problem: Episodic, single agent, fully observable, deterministic, static, discrete, known
  - Generate the state when we need
  - Frame the problem
    - size of the state space graph
    - implementation
      - State: dictionary, list
      - Transition: Function, table, map
      - Location as values
    - Problem to resolve: **planning**, routing, design, sequencing, puzzles, and **games**
  - Uninformed search
    - Agent can not estimate how far it is from the goal
    - DFS (stack)
      - Complete — N
      - Optimal — N
      - Space: O(bm)
        - b: branching factor
        - d: depth of shallowest solution
        - m: max depth of the resulting search tree
      - Time (operations, not CPU): O(b^m)
      - Improvement: iterative deepening => why not BFS?
    - BFS (queue):
      - Complete — Y
      - Optimal — Y (uniform cost)
      - Space: **O(b^m)**
      - Time: (O(b^m)
  - Declarative programming
    - Function, logic, DB approach
    - expresses the logic of a computation without describing its control flow
    - What to do: Red? Blue: red
    - Imperative programming: How to do: if RED, blue. If Blue, red
  - Informed search
    - The cost model may give us important information about good solution and where to look for them
    - UCS, Uniform cost search (Priority queue)
      - Cost: as our key. Examine the node on the frontier with lowest total path cost so far
      - Successor function with cost-calculated annotation
      - BFS++
    - Greedy(Priority queue)
      - Best first search
        - A simplest variant: pick the cheapest action **at the current.**
        - Cost estimates come from heuristic functions
      - Might faster than UCS
      - Doesn't always work — Might deadlock itself
      - Heuristic function
        - **Estimate current node to the goal**
        - Lecture: use current action cost as an estimate
        - c: the shortest cost from previous node, through action node, to the goal
        - **Admissible** heuristic: h(n) <= c — The best h(n) is the c, better is close to c
      - Complete (finite state space), not optimal.
      - Time/space: O(IVI) or some cases O(bm)
    - A* search (Priority queue)
      - UCS+Greedy
      - Extremely useful for path finding
      - f(n) = g(n) + h(n)
        - g(n): cost so far, from UCS
        - h(n): heuristic from greedy — The estimated cost of getting from the current node to the goal
      - Optimal (may or may not) and complete
        - If there is a path to the goal
        - Depends on heuristic function