1. State Space
   - Start state
   - Actions (A function which returns which actions can be performed at s.)
   - Transitions (Specifications of what each action does.)
   - Goal Test (Specifies whether s is a goal state.)
   - Path cost (Cost of a particular transition)

   Can be useful to think in these terms, it is usually unnecessarily complicated.

   Simpler to understand actions as transitions, and the path cost is the cost of a transition.

   - Start state
   - Transitions, including cost
   - Goal test

   Do whatever you feel comfortable with!

   Goal test versus Fitness test (Sometimes we don’t know a goal when we see one!)

2. Draw ‘usual’ example
   Europe map.

   Discuss issue of visibility!

   Goal versus fitness (Has Kremlin, Is Most Beautiful). We will see utilization of fitness functions later in the course.

3. Search tree
   - Nodes/States
   - Expanding nodes
   - Frontier
   - Search strategy (how we go about choosing nodes to expand)
   - Redundant paths
   - Explored Set
4. Criteria

- Completeness: Guaranteed to find a solution (if one exists)?
- Optimality: Will it find the optimal solution (in terms of path cost)?
- Time Complexity: How long will it take to find a solution?
- Space Complexity: How much memory is needed?

5. Breadth First

- Idea (expand each row in order until goal found)
- Example. Use:
  - FIFO
  - Frontier
  - Explored Set
  - Search Tree
- Completeness. Yes - if branching factor finite and solution at finite depth. (When would this fail?)
- Optimality. No. Finds shallowest solution. This is not necessarily the optimal solution.

- Time: $b^d - b + 1 + 1$ if goal test applied when expanding node. Memory: $b^d$ in the frontier (and $b^{d-1}$ in the explored set)

6. Uniform-Cost Search

- Priority queue - Expand node with lowest path cost
- Must test node when it is expanded (first found goal may be sub-optimal)
- Must replace costs of nodes in frontier with lower costs if such are found (on better paths)
- Complete if all step costs exceed some positive constant (and finite b and d). (Compare with continous variables: Hill climbing as dipping in at points)
- Optimal
- Complexity not easy to characterise, look at book.

7. Depth First

- Idea
- LIFO (or recursion)
- Example
- Complete: Tree-search not complete; Graph search is complete in finite search spaces
- Optimality: Non-optimal (Assume depth = cost: If goal on top of right explores left side first, then right. If sub-optimal goal on left, it is found first.)
• Time: Graph search: Size of state space. Tree search: $b^m$, where $m$ is max depth and can be infinite with loops.
  Graph search: No advantage, must store all visited nodes. Tree search: $d^m$—only store a single path from root to lead (plus siblings, though this too can be avoided, see book).

8. Iterated Deepening
   • Run depth first search to depth $d=0,1,2,3,4,5...$
   • Complete: When $b$ and $d$ finite. (note loops do not lead to infinite paths at any step)
   • Optimality: As Depth First
   • Time: $(d)b+(d-1)b^2+(d-2)b^3+...O(b^d)...$ like breadth first. Space: $d^m$, like depth first

9. Bidirectional Search
   • Search forward from origin and backwards from goal and find an intersection.
   • Draw (double) search tree
   • Motivation is that $b^{d/2}+b^{d/2}$ is much less than $b^d$. Need to be able to compute predecessors
   • See book for more details

10. Heuristic Search:
    • Evaluation Function: The function which selects which node is to be expanded next. Cost - select lowest cost.
    • Heuristic function: Estimated cost from node to goal.
      – Example: As the crow flies distance
      – Note that this requires domain knowledge
    • Admissibility: The heuristic must never overestimate the cost to the goal (ie Optimistic - Crow flies is optimistic)
    • Monotonicity: For every node $n$, and every successor of $n$, $t(n)$, the estimated cost of reaching the goal from $n$ is no greater than the step cost from $n$ to $t(n)$ plus the estimated cost of reaching the goal from $t(n)$.
      – $h(n) = c(n\rightarrow t(n)) + h(t(n))$

11. Greedy Best First
    • Use heuristic as evaluation.
    • Complete: Graph: Complete (in finite spaces). Tree: Incomplete (give example where we always return to a dead end, Moscow-Stockholm (StP))
    • Look in book for other criteria

12. A*
    • Priority queue: Expand the node that has minimal $f(n)=g(n)+h(n)$:
      – $g(n)$= actual path cost to node
      – $h(n)$= heuristic estimate of cost to goal
• Complete: Yes, if all step costs are larger than some positive constant and b and d are finite.

• Optimal: If heuristic is admissible then the tree search is optimal. If heuristic is monotonic (and therefore also admissible) then graph search is optimal
  -- Note that f(n) is the actual cost to goal for all goal nodes (since h(n) = 0). Therefore the first goal node expanded is certain to be optimal.
  -- In fact, A* will expand all nodes with f(n) less than that of the optimal goal node, and possible some nodes where f(n) is equal to that of the optimal goal node. Accordingly, it is optimally efficient for a given heuristic: No other algorithm is guaranteed to expand fewer nodes than A* for a given heuristic (ignoring tie breaking methods).

• Time: See book for details.

• Space: We must store all nodes in the frontier (and for graph those visited). Worst case is the size of the state space.

• A* usually runs out of space before it runs out of time.

• We can attempt to improve things. One example is Simplified Memory-Bound A*. Basically:
  -- Proceed as usual until assigned memory is full.
  -- Now when adding a new node, drop worst leaf node (breaking ties with age) and back up value to parent.
  -- See literature for more detail and other algorithms that seek to deal with the memory problems of A*