Concept #1  Elements of a decision

A. Problem

1. Our task is to encode the problem so that a computer can solve it.

2. Components
   Action
   Goal
   State
   Cost

B. Formalization

Goal test
\[ s_f \in G = \{ s : \text{goal condition met} \} \]

Do this for checkers example
\[ s_{tr1} = a(s_6) \]

\[ a \in A : S \rightarrow C \]
\[ c = \{ s_{tr1}, \text{costs} \} \]
C. Do for robot navigation on a grid world.
D. Groups choose & formulate a decision.

Concept #2: Search Strategy

A. State space & search trees
   \( S = \text{states in world} \)
   \( \text{Tree = states in world that have been visited} \)

B. Example: Navigation (Route Finding)

\[
G = \{ (1,3) \}
A = \{ N, S, E, W \}
\]

Initial state, root of tree

\[
\text{Fringe = nodes waiting to be expanded}
\]

- encoded as a queue or priority queue:
  - front of queue is next node expanded

search strategy

C. Objective: Find efficient way to find a sequence of actions from start to goal \(\Rightarrow\) cool way to set-up queue.
Concept #3  Measuring/Evaluating Search Strategy

- Completeness: guaranteed to find a solution
- Time Complexity
- Space Complexity
- Optimality: found solution a high quality one?

Concept #4  Breadth-First Search

A. Use concept #2 example

B. Efficiency
   Complete?  Yes, if path cost is non-decreasing
   Optimal?  Yes, if path cost is non-decreasing
   Function of node depth:
   Time complexity: \( b^d \) (\( b \) nodes expanded, \( 1, b, b^2, \ldots \))
   Space complexity: size of frontier = \( b^d \) (\( b \) nodes in frontier)

In BFS, use FIFO expansion (expand all nodes at level \( k \))
than level \( k+1 \), \ldots

Key concept, Depth of node.
Concept #5 Uniform Cost Search

A. Always expand lowest-cost node on the fringe

B. Edge Costs

\[ w(u,v) \]

\[ g(n) = \text{Edge costs} \]

\[ \text{BFS = UCS when } g(n) = \text{Depth}(n) \]

C. Efficiency

- Optimal if \( g(\text{successor}(n)) \geq g(n) \)
  - "Non-decreasing path cost"

- Complete?
- Time complexity?
- Space complexity?
Concept #6  Depth First Search

A. Always expand the newest unexpanded node on the Fringe

B. Infinite loop

C. Optimal - no
   - Time complexity
   - Space complexity

\( b^m \)  \( m = \text{maximum depth} \)
Concept #7  Avoiding Repeated States

A. Priming the state space
   - saves space
   - makes route finding finite

B. Techniques
   - do not return to parent state
   - do not cycle (don't go to state generate successor node that is a parent ancestor node
   - don't regenerate visited states

\[ O(s) \quad s = \text{size of state space} \]

\[ \uparrow \]

Space complexity
A. Back from goal + forward from start
B. Optimal: yes
   Complete: yes
   Time: $O(d^2)$
   Space: $O(d^2)$
C. Requirements — See text
   - generating predecessors
   - reversible operators
     $$\delta(n) = \delta_2 \quad \delta_1 = \rho(n)$$
   - know goal explicitly (not recognize when we get there)
   - check to see if new node is already in other half of search
     (e.g., associative memory)
   - what search done in each half.
   - $O(d^2)$ assumes testing intersection
done in constant time (e.g., associative mem, not search).
Concept #10  Informed Search
See last year's notes. Talk about tab.

Concept #10  Best-First Search
- This is a generalization of UCS (priority queue)
- UCS uses g(n) (“cost-so-far”)
- BFS can use g(n), h(n), or both.
- Informed search.

Concept #11  Heuristics
= “cost to go”

h(n) = estimated cost of the cheapest path from state at node n to goal
- admissible: a heuristic that never
Concept #11 continued...

- Why admissible heuristics are important.

All paths go around to avoid unrealistic heuristic costs, but if $h(n) \leq h^*(n) = f^*(n) - d$, you will test shortest paths first.
Concept #12: Greedy Search
Ignore $g(n)$, use only $h(n)$.

Example, $h(n) =$ Euclidean distance

$A^*$

$f(n) =$ estimated total cost of cheapest solution that passes through $n$
$= g(n) + h(n)$.
Concept #14. **A**<sup>+</sup> **part 2**

Given a graph and a heuristic function, we can apply **A**<sup>+</sup> algorithm to find the optimal path. The heuristic function must be admissible and consistent for **A**<sup>+</sup> to be guaranteed to find the optimal path.

- **Path**: \( f(n') = \max_n (f(n), g(n') + h(n')) \)
  
  - Example:
    - Given: \( g(n') = 4, h(n') = 2 \)
    - Using \( h(n) = 4 \): \( f(n) = 7 \)
    - Using \( h(n) = 2 \): \( f(n) = 6 \)

- **Optimality**: **Efficient** for **A**<sup>+</sup> when \( h(n) \) is admissible.
  - No other optimal algorithm exists that will always expand fewer nodes than **A**<sup>+</sup>.
Learn that when it is incomplete, 
out of (comparing) partially effective.

1. Justified exploration is partially effective.
2. Evaluation: costs & benefits.
3. Nature's (assumptions) can solve shutdown.
4. Complexity of the branching & the function.
5. High (cost) & low (cost).
6. Expressed less of the language.
7. Tractability of the language.

Appendix A: Properties of and relations between various search strategies.
Concept #15. Proof of $A^*$ optimality

Suppose $G_1$ is optimal

$\Rightarrow$ In, $g(n) = g(n)$

$\rightarrow$ optimal path

\[ 1 \] Suppose that $A^*$ returns a goal node (including linked list of parents $\rightarrow$ path) $G_2$ instead of $G_1$. WLOG, $G_2$ could be the same state as $G_1$, but reached by suboptimal path.

\[ 2 \] By suboptimality of $G_2$

$g(b_2) > g(b_1) = f^*$.

\[ 3 \] Consider some $n$ along true optimal path from $A$ to $G_1$, s.t. $n$ is on Frontier of expansion

- Such a node must exist.
  otherwise, all nodes would have been expanded and $G_1$ would have been found.

- For some such $n$, $G_2$ is expanded over $n$. Meaning

  $f(n) > f(b_2)$.  \[ \Box \]
Concept #15 continued...

4. Since $h(n)$ is admissible, $f(n) \geq f^{*}$

5. Putting 3 b & 4 together
   
   $f^{*} = f(g_{3})$

6. But $f(g_{3}) = h(g_{3}) + g(g_{3})$
   
   $= g(g_{3})$ since $g_{3}$ isgoal
   
   whence $h(g_{3}) = 0$.

   and $f^{*} = g(g_{3})$.

7. Putting 6 into 5 gives

   $g(g_{3}) \geq g(g_{3})$, a contradiction.

8. Therefore, $g_{3}$ cannot be expanded.