Intro

- Tri-city: many ways to go about it this, answer is to do A* except run the three cities in parallel
- Also iterative deepening A* can solve rubik's cubes

What is a problem

- problem has a state (S0)
- actions {a1, a2, ..}
- Result (s,a) which makes a new state (S')
- GoalTest(s) returns True | False if the state is goal or not
- Path Cost, takes sequence of state/action transactions and sums up it's cost, essentiall the sum of the step cost
- Step Cost, takes (s, a, s') and returns the cost

Route Finding

- Explored, the nodes and routes that have been visited
- Frontier, the furthest the nodes have explored (furthers out)
- Unexplored, the routes and nodes outside the fronteier

Tree Search

- Superimposes tree space over search spaces
- steps:
  - frontier is initial state
  - for loop
  - pop curState from frontier
  - check if curState is goal
  - add curStates paths to frontier

Graph Search

- Tree search, except when pop from frontier, add to explored states
- Don't re-expolre states that have already been explored

Breadth First Search (shortest first search)

- considers shortest path first
- if the goal is added to the frontier, still continue search because we're not sure if this is the shortest path
- Goal is checked when poped from frontier
- I mean, we could optmize this though if we're looking for shortest steps

Uniform Cost Search (cheapest first search)

- Steps
  - move curState from frontier to explored
  - add it's paths to frontier
  - pick the path with the lowest total path (summation of all paths that make up paht)

- Can only terminate if the goal is popped off the fronteir (possible to find another path that is cheaper)
- The algo expands out in all directions looking for goal, but not directed at goal
  - typically explore half the space to get to the goal
  - Need to add more knowledge to get better
  - Greedy based search - does exactly this, however, if there's a barrier, we're fucked

Search Comparison

- Breath first search vs Cheapest-First vs Depth-First
  - BFS - optimal for finding shortest, complete
  - UCS - optimal for finding cheapest, complete
  - DFS - not optimal, not complete
- Why woould anyone use DFS?
  - The fronteir will only have n nodes, instead of 2^n nodes
  - sometimes this doesn't matter, like if you keep track of expolored nodes
- Complete means that if there's an infinite tree, then the algo will still find the goal. DFS will continue down infinite path and never find the goal

A* (best estimated total path cost first)

- Need the best of Uniform Cost Search and Greedy Search
- Always expanding the path that is mimimum f where f = g + h
  - g(path) = path cost (mimimize this to keep path short)
  - h(path) = h(s) = estimated distance to goal (mimimize this to keep focused on goal)
- Again, when we pop the goal off, no when we add it to the queue first.
- A* will only find the lowest cost path h if h(s)< true cost
  - if the h overestimate distance to goal (h is optimistic/admissible)

Optimistic Heuristic

- Optimistic h finds lowest cost path
  - if f = g + h and h(s) < true cost, then when f finds shorts path to goal, we're 100% sure that the path is the shortest path

State Spaces

- Example: vaccuum world, vaccuum can be in 2 positions that can be clean or dirty
  - 8 states because (2 x 2 x 2)
- Now vaccuum can be on/off/sleep, cam on/off, bursh 1-5, and 10 positions
  - 3 X 2 X 5 X 10 X 2^10 = 307,200 states :O
- so simple, yet so fucking complex. This is why we need efficient algorithms

Sliding Blocks Puzzle (15 puzzle)

- both heuristic are admissible
  - h1 = #misplaced blocks, every tile has to be moved at least once
  - h2 = sum (distances of blocks), every tile can be moved closer by at least one move
  - h2 is always greater tha h1, so with the exception of breaking ties, A* will always expand less paths with h2 than h1
- therefore the goal is to get the **closest** possible heuristic, but still being admissible

- finding heuristics isn't that bad. For example the sliding block constraints are:
  - A block can move A -> B if (A adjacent to B) and (B is blank)
  - the statement without the 2nd param is h2, and without both param is h1
  - and the we can do max(h1, h2) to make better heuristic (might be more computation)
  - this is called generating a relaxed problem (deconstraints)

Limitations with Search

1. must be fully observable (must know what we start off with)
2. domain must be known (what actions avaialbe)
3. domain must be discrete (finite number of actions)
4. domain must be deterministic (know the results of actions)
5. domain must be static (nothing changes)

Note on Implementation

- nodes is a data structure with 4 fields
  1. state ad the end of path
  2. action it took to get there
  3. cost
  4. pointer to another node (parent)
- frontier
  - remove best and add in new ones
  - priority queue + membership queury
  - so use a SET (hash table or tree)
- Explored
  - add new members
  - check for membership
  - single set (hash table or tree)

Outro

- AI is programming a computer to do the right thing when you don't know what the right thing is
- pay attention to the data, measure how well you're doing
- Pay attention to the people, keep them happy
- These guys like lisp a lot, but then switched to python because more feasible

Lab g

- BFS, DFS, and A* in PacMac