Administrivia

Homework Turn-In:

- Due Tuesday in class
- For Tuesday homework: 7 days
- For Thursday homework: 5 days
- Both include a weekend
- No lates or excuses -- just move on
Online slide policy

- Slides will be posted online
- Hopefully long enough before class for you to have for note-taking
- Slides complement the lecture
- Do no expect them to make sense by themselves
- The syllabus is in terms of textbook material
Administrivia

TA Lab hours:

- Monday 5–6: Majors’ Lab (CSB 633)
- Thursday 6–7: Walter’s Office (CSB625)

CSUG Tutoring (not just 242):

- M 11:30am – 1:30pm (Josh Pawlicki)
- W 5pm – 7pm (Simon Weber)
- F 3pm – 5pm (Darcey Riley)
Problem Solving
Problem Solving
Roger Bacon (c. 1214–1294)
\[ |\vec{F}_{1,2}| = |\vec{F}_{2,1}| = \frac{C_1 M_1 M_2}{d^2} \]
CSC242

We don’t make the computers.*
We make the computers solve problems.

*Or the programming languages, compilers, debuggers, databases, graphics pipelines, network protocols, web servers, ...
Our first problem
Given start city A and destination city B:

Can I get from A to B?

How do I get from A to B?

Minimize tolls

Deadline

Mileage limit

Stay close to rest areas
Romania

Cities

Arad

Timisoara

Zerind

Oradea

Drobeta

Mehadia

Craiova

Sibiu

Rimnicu Vilcea

Pitesti

Bucharest

Urziceni

Hirsova

Eforie

Scenic areas

Elevation

Rest areas

Roads: connect cities

Distances between cities

71

75

118

111

70

75

120

211

101

90

98

86

142

92

75

140

99

211

138

87

97

80

99

85

101

98

92

90

87

86

85

70

75

146

97

118

151

111

140

97

146

120

80

90

101

98

92

90

86

85

85
What problem are we trying to solve?

What aspects of the world are important to solving that problem?
Problem Solving by Computers
Given initial city $src$ and destination city $dst$, find a route from $src$ to $dst$ (if one exists).

**Approach #1**

Precompute, for each $<src,dst>$ pair, the shortest route between them. Then, to get from $src$ to $dst$, just lookup the stored route.

**Does it work?** Yes

**Problems?**

- Lookup table grows as $O(n^2)$
- Computing all-pairs shortest path: $O(n^3)$
- Changes to network: recompute entire table
Given initial city src and destination city dst, find a route from src to dst (if one exists).

Approach #2

If I were at src, and I wanted to get to dst, what could I do?

➡ Move to any adjacent city

➡ Pick an adjacent city

➡ Move to it

➡ Until I’m at dst
find_solution(City src, City dst) {
    City[] solution = [];
    City c = src;
    while (c != dst) {
        City[] neighbors = adjacentCities(c);
        City next_c = selectOne(neighbors);
        solution.add(next_c);
        c = next_c;
    }
    return solution;
}
find_solution(City src, City dst) {
    City[] solution = [];
    City c = src;
    while (c != dst) {
        City[] neighbors = adjacent_cities(c);
        City next_c = select_one(neighbors);
        solution.add(next_c);
        c = next_c;
    }
    return solution;
}

Problems?
State
(Data Structure)
find_solution(City src, City dst) {
    City[] solution = [];
    City c = src;
    while (c != dst) {
        City[] neighbors = adjacent_cities(c);
        City next_c = select_one(neighbors);
        solution.add(next_c);
        c = next_c;
    }
    return solution;
}
State
(Data Structure)

\[
\begin{bmatrix}
7 & & 1 \\
6 & 2 & 8 \\
3 & 4 & 5
\end{bmatrix}
\]

```java
int[][] M = new int[3][3];
M[0][0] = 7;
M[0][1] = 0;
... 
M[2][2] = 5;
```
Approach #1
Precompute, for each $<M^{\text{in}},M^{G}>$ pair, a sequence of moves that transforms $M^{\text{in}}$ into $M^{G}$. Then, to solve $M^{\text{in}}$, just lookup the stored sequence.

Does it work? No

Problems?
- There are $9! = 362880$ ($O(n!)$ in general) cases
- No obvious way to solve the $<M^{\text{in}},M^{G}>$ cases
- This problem is known to be NP-complete
Approach #2
If the puzzle is currently $M$, and I want it to be $M^G$, what could I do?

- Pick an adjacent space
- Move the blank to it

Until $M == M^G$
Actions

- Can be performed in a state
- Change the state to a resulting state
find_solution(City src, City dst) {
    City[] solution = [];
    City c = src;
    while (c != dst) {
        City[] neighbors = adjacent_cities(c);
        City next_c = select_one(neighbors);
        solution.add(next_c);
        c = next_c;
    }
    return solution;
}
<table>
<thead>
<tr>
<th>7</th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
\[
\begin{array}{ccc}
7 & 1 & 8 \\
6 & & 2 \\
3 & 4 & 5 \\
\end{array}
\]

1 - 8 2 - ...
East - South - West - ...
Actions

• For any state and action:
  • Can I apply this action in this state?
    • “Applicability”
  • How do I update the state if this action is performed?
    • “Result” or “transition” function
Approach #2

If the puzzle is currently $M$, and I want it to be $M^G$, what could I do?

➡ Pick an adjacent space
Move the blank to it

Until $M == M^G$
If the situation is like this, and I want it to be something else,
From the things I could do in the current situation,
Pick something to do
Do it (and remember what I did)
Keep going until the situation is what I’m looking for
find_solution(State initial) {
    State s = initial;
    Action[] solution = [];
    while (!is_goal(s)) {
        Action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
Problem (Domain): \((S, A, \text{Actions}, \text{Result})\)

**Actions**: \(s \in S \rightarrow [a \in A : a \text{ can be executed (is applicable) in } s]\)

**Result**: \(s \in S, a \in A \rightarrow s', s' \in S \text{ s.t. } s' \text{ is the result of performing } a \text{ in } s\)

Problem (Instance): \((I \in S, G \subseteq S)\)

Solution: \((a_0, a_1, \ldots, a_n) \in A \text{ s.t. } \text{Result} \circ \text{Result}(\text{Result}(\text{Result}(I, a_0), a_1) \circ \ldots \circ a_n)) \in G\)
“Fly”

Problem-solving state
Transition Model

World state
Action
find_solution(initial) {
    state s = initial;
    action[] solution = [];
    while (!is_goal(s)) {
        action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(initial) {
    state s = initial;
    action[] solution = [];
    while (!is_goal(s)) {
        action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
State-Space Search
States + Actions + Transition Model = State Space

The set of all states reachable from the initial state by some sequence of actions
State Space

\[ V \cup E \vdash \]
\[ V = \{ v_i | s_i \in S \} \]
\[ E = \{ (v_i, v_j, a) | a_j = \text{Result}(s_i, a) \} \]
I'VE LOST YOU SO
WHY SHOUL I CARE

THEDA BARA

By Richard Howard

Daly
MUSIC PUBLISHERS
BOSTON, MASS
State-Space Search

- Start with initial state
- Generate successor states by considering applicable actions
- Until you get to a goal state
Solution treeSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());

    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        for (Node n : node.expand()) {
            frontier.add(n);
        }
    }
}
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return n.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
• General-purpose algorithm for solving any problem that can be represented using states and actions that transition between them

• State-space search framework will allow us to explore and compare alternatives

```java
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return n.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
```
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return n.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
For Next Time:
Finish Chapter 3