CSC242: Intro to AI

Lecture 2
Upper Level Writing

Read → Think → Topic
Read → Think → Write → Mar 1 → Critique
Read → Think → Write → Apr 1 → Critique
Read → Think → Write → May 2
Academic Honesty
PLAGIARISM IS CHEATING

plagiarism |ˈplājərīzəm|
noun
the practice of taking someone else's work or ideas and passing them off as one's own.

ORIGIN early 17th cent.: from Latin plagiarus ‘kidnapper’ (from plagium ‘a kidnapping,’ from Greek plagion) + -ism.
University Policy

- Plagiarism
- Copying papers or allowing others to copy your work
- Copying answers on exams or facilitating copying of your own exam answers
- Any other act that represents someone else's work as your own
- Misuse of library materials such as the removal of books from the libraries without formally checking out the items, or the intentional hiding of materials, or the refusal to return reserve readings to the library, etc.
- Obtaining an exam prior to its administration
- Using unauthorized aid during an examination
- Altering answers on graded exams and submitting them for re-grading
- Copying data from other students' labs or research projects or allowing others to copy your data
- Using labs or assignments from previous semesters or from other students and submitting them for credit
- Copying answers from online answer keys
- Turning in identical work on collaborative assignments
- Giving or receiving inappropriate help or feedback on written assignments.
University Policy

- If found guilty:
  - First offense: “grade change, course failure, or suspension”
  - Second offense: “standard penalty for a second offense is suspension from the University”
University Policy

Academic Honesty: Arts, Sciences, Engineering
http://www.rochester.edu/college/honesty/

Academic Honesty: Center for Academic Support
http://www.rochester.edu/college/ccas/AdviserHandbook/AcadHonesty.html
One More Policy
One More Policy

[Logos representing Facebook, Angry Birds, Twitter, and Farmville with prohibition signs around them]
Problem Solving
Problem Solving
Roger Bacon (c. 1214–1294)
\[ |\vec{F}_{12}| = |\vec{F}_{21}| = \frac{G M_1 M_2}{d^2} \]
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?
Given start city A and destination city B:

Can I get from A to B?

How do I get from A to B?
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

Stay close to rest areas

Mileage limit
Romania

Cities

Roads: connect cities

Distances between cities
What problem are we trying to solve?
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).
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 route.
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?
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
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?
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 $O(n^2)$
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: $O(n^3)$
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
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?
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
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?

➡ Pick an adjacent city
Move to it
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?

- Pick an adjacent city
- Move to it
- Until I’m at $dst$
Romania

Tuesday, February 26, 13
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;
}
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?
Intelligence and Generality

- Intelligence includes the ability to solve many kinds of problems
- Including problems we haven’t seen before
- Every new problem-solving method needs to be designed, implemented, tested, and debugged
Given initial puzzle configuration $start$ and desired configuration $goal$, find a sequence of moves that goes from $start$ to $goal$ (if one exists).
State
(Data Structure)
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)

<table>
<thead>
<tr>
<th>7</th>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
State
(Data Structure)

\[
M_{i,j} = \begin{bmatrix}
7 & 0 & 1 \\
6 & 2 & 8 \\
3 & 4 & 5
\end{bmatrix}
\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
### State (Data Structure)

\[
M_{i,j} = \begin{bmatrix}
7 & 0 & 1 \\
6 & 2 & 8 \\
3 & 4 & 5 \\
\end{bmatrix}
\]

```java
int[][] M = new int[3][3];
M[0][0] = 7;
M[0][1] = 0;
... 
M[2][2] = 5;
```
Actions

• Can be performed in a state
• Change the state to a resulting state
```java
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;
}
```
South
West
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

East - South - West - ...

Tuesday, February 26, 13
Actions

• For any state and action:
  • Can I perform this action in this state?
  • “Applicability”
  • How do I update the state if this action is performed?
  • “Result” or “transition” function
\[ M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \]
\[
M^\text{in} = \begin{bmatrix}
7 & 0 & 1 \\
6 & 2 & 8 \\
3 & 4 & 5
\end{bmatrix} \quad M^G = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 0
\end{bmatrix}
\]
\[ M^{in} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^{G} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \]

Approach #1

Precompute, for each \(<M^{in},M^{G}>\) pair, a sequence of moves that transforms \(M^{in}\) into \(M^{G}\). Then, to solve \(M^{in}\), just lookup the stored sequence.
\[ M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \]

Approach #1

Precompute, for each \(<M^{\text{in}}, M^{\text{G}}\)> pair, a sequence of moves that transforms \(M^{\text{in}}\) into \(M^{\text{G}}\). Then, to solve \(M^{\text{in}}\), just lookup the stored sequence.

Does it work?
\[ M^{in} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^{G} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \]

Approach #1

Precompute, for each \(<M^{in}, M^{G}>\) pair, a sequence of moves that transforms \(M^{in}\) into \(M^{G}\). Then, to solve \(M^{in}\), just lookup the stored sequence.

Does it work? No
$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^G = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$

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?
\[
M^{\text{in}} = \begin{bmatrix}
7 & 0 & 1 \\
6 & 2 & 8 \\
3 & 4 & 5
\end{bmatrix}
\quad
M^G = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 0
\end{bmatrix}
\]

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
\[ M^{in} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^{G} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \]

Approach #1

Precompute, for each \(<M^{in}, M^{G}>\) pair, a sequence of moves that transforms \(M^{in}\) into \(M^{G}\). Then, to solve \(M^{in}\), 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 \(<M^{in}, M^{G}>\) cases
\[
M^{in} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^G = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}
\]

**Approach #1**

Precompute, for each \(<M^{in}, M^G>\) pair, a sequence of moves that transforms \(M^{in}\) into \(M^G\). Then, to solve \(M^{in}\), 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 \(<M^{in}, M^G>\) cases
- This problem is known to be NP-complete
$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^G = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$

Approach #2
Approach #2

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

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^G = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$
Approach #2

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

➡ Move the blank to an adjacent space
$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^{G} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$

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
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 $M == M^G$
\[ M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^{G} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \]

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 \( M == M^{G} \)
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?

➡ Pick an adjacent city
   Move to it

Until I’m at dst
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 problem state is currently $S$, and I want it to be $S^G$, what could I do?

➡ Pick an applicable action $A$

Update $S$ with the result of applying $A$

Until $S == S^G$
```java
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;
}
```
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;
}
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;
}
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;
}
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;
}
```java
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;
}
```
```java
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;
}
```
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;
}
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;
}
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;
}
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

\(\langle V, E \rangle :\)

\(V = \{v_i \mid s_i \in S\}\)

\(E = \{\langle v_i, v_j, a \rangle \mid s_j = \text{RESULT}(s_i, a)\}\)
State-Space Search

• Start with initial state
• Generate successor states by applying applicable actions
• Until you find 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 n.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);
            }
        }
    }
}
Summary

• 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
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