CSE 143

read: 12.5

Lecture 17: recursive backtracking
Exercise: Permutations

• Write a method `permute` that accepts a string as a parameter and outputs all possible rearrangements of the letters in that string. The arrangements may be output in any order.

• Example:
  ```java
  permute("TEAM")
  ```
  outputs the following sequence of lines:

<table>
<thead>
<tr>
<th>TEAM</th>
<th>ATEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMA</td>
<td>ATME</td>
</tr>
<tr>
<td>TAEM</td>
<td>AETM</td>
</tr>
<tr>
<td>TAME</td>
<td>AEMT</td>
</tr>
<tr>
<td>TMEA</td>
<td>AMTE</td>
</tr>
<tr>
<td>TMAE</td>
<td>AMET</td>
</tr>
<tr>
<td>ETAM</td>
<td>MTEA</td>
</tr>
<tr>
<td>ETMA</td>
<td>MTAE</td>
</tr>
<tr>
<td>EATM</td>
<td>META</td>
</tr>
<tr>
<td>EAMT</td>
<td>MEAT</td>
</tr>
<tr>
<td>EMTA</td>
<td>MATE</td>
</tr>
<tr>
<td>EMAT</td>
<td>MAET</td>
</tr>
</tbody>
</table>
Decision tree

<table>
<thead>
<tr>
<th>chosen</th>
<th>available</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEAM</td>
<td></td>
</tr>
</tbody>
</table>

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM

TEAM
Backtracking

- Useful to solve problems that require making decisions
  - Each decision leads to new choices
  - Some (but not all!) sequence(s) of choices will be a solution
  - Insufficient information to make a thoughtful choice

- Systematically prune out infeasible solutions
Exercise: solve maze

- Write a method `solveMaze` that accepts a `Maze` and a starting row/column as parameters and tries to find a path out of the maze starting from that position.

- If you find a solution:
  - Your code should `stop` exploring.
  - You should `mark` the path out of the maze on your way back out of the recursion, using backtracking.

- (As you explore the maze, squares you set as 'explored' will be printed with a dot, and squares you 'mark' will display an X.)
Maze class

Suppose we have a Maze class with these methods:

<table>
<thead>
<tr>
<th>Method/Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Maze(String text)</td>
<td>construct a given maze</td>
</tr>
<tr>
<td>public int getHeight(), getWidth()</td>
<td>get maze dimensions</td>
</tr>
<tr>
<td>public boolean isExplored(int r, int c)</td>
<td>get/set whether you have visited a location</td>
</tr>
<tr>
<td>public void setExplored(int r, int c)</td>
<td></td>
</tr>
<tr>
<td>public void isWall(int r, int c)</td>
<td>whether given location is blocked by a wall</td>
</tr>
<tr>
<td>public void mark(int r, int c)</td>
<td>whether given location is marked in a path</td>
</tr>
<tr>
<td>public void isMarked(int r, int c)</td>
<td></td>
</tr>
<tr>
<td>public String toString()</td>
<td>text display of maze</td>
</tr>
</tbody>
</table>
Decision tree

Position: (row 1, col 7)

Choices: \( \rightarrow \uparrow \downarrow \leftarrow \)

(1, 6) \hspace{1cm} (0, 7) \hspace{1cm} (2, 7) \hspace{1cm} (1, 8)

- (1, 6) \rightarrow (1, 5) \rightarrow (1, 4)
  - wall
  - wall
  - ... (1, 4)

- (0, 6) \rightarrow (0, 5)
  - wall
  - ... (0, 5)

- (2, 6) \rightarrow (2, 5)
  - ... (2, 5)

- (1, 7) \rightarrow (1, 8)
  - visited
  - ... (1, 9)

(0, 7) \rightarrow (0, 8)
- visited
- ... (0, 8)

(2, 7) \rightarrow (2, 8)
- wall
- ... (2, 8)

(1, 8) \rightarrow (1, 9)
- wall
- ... (1, 9)

*These never change*
Recall: Backtracking

A general pseudo-code algorithm for backtracking problems:

Explore(\texttt{choices}):

- if there are no more \texttt{choices} to make: stop.
- else, for each available choice \texttt{C}:
  - Choose \texttt{C}.
  - Explore the remaining \texttt{choices}.
  - Un-choose \texttt{C}, if necessary. (backtrack!)