Recursive Trees

CS 106B

Programming Abstractions
Fall 2016
Stanford University
Computer Science Department
Announcements

▶ Assignment 2 due Saturday

▶ Assignment 3 goes out today (due Oct 24th)

▶ CS for Social Good has a winter class CS 51
Assignment 3

META ACADEMY

Due: Oct 24th
YEAH: Monday 7-8pm
Location: TBA
In order to learn recursion, you will teach recursion.
In order to teach recursion, you must first learn recursion.
Recursion by Definition

\[ x^0 = 1 \]
\[ x^n = x \cdot x^{n-1} \]
Fractals
Today’s Goal

1. Learn to see trees in recursion
2. Be ready for assn #3
Problems with the Same Underlying Form

1. List all **permutations** of a set of elements
2. Output the path of all files in a **folder**.
3. Find all words that are “**reducible**”
4. Play a game of knight’s tour **solitaire**
5. Find the optimal “alignment” of two strands of **DNA**

They have a similar underlying representation and they can all be solved with recursion.
Let's talk about your life decisions
Decision Tree

- Undergrad
  - Oceanography
  - Native Studies
- CS
  - AI
  - HCI
  - Graphics
  - CompBio
- Research
- Startup
Why Tree?
Decision Tree

Undergrad
Oceanography
Native Studies

CS

AI
HCI
Graphics
CompBio

Research
Startup
Decision Tree

Current state

Option 0

Next state 0

Option 1

Next state 1

Option n

Next state n
Decision Tree

Next state 1

Option 0

Option 1

Option m
Recursion Hearts Trees

Recursion
outputAllEndStates("undegrad")?

3. When you make a recursive call it should be to a simpler instance (forward progress towards base case)

Set<string> next = getNextStates(currState);
void outputAllStates(string currState) {
    if (isEndState(currState)) {
        cout << currState << endl;
    } else {
        Set<string> allNextStates = getNextStates(currState);
        for (string next : allNextStates) {
            outputAllStates(next);
        }
    }
}
Recursion on Trees

```cpp
void outputAllStates(string currState) {
    if (isEndState(currState)) {
        cout << currState << endl;
    } else {
        Set<string> allNextStates = getNextStates(currState);
        for (string next : allNextStates) {
            outputAllStates(next);
        }
    }
}
```

```
next = "Oceanography"
```

```
allNextStates: [ "Oceanography", "CS" , "Native Studies"
```
The point: get the right metaphor in your head.
List all Paths in a Folder
A folder (aka directory) is a file system container that can hold files and other folders.

A folder is a tree!
Visualize the Recursion

eexampleFolder

anotherFolder

child1

child2

child3
A Little Word Puzzle
“What nine-letter word can be reduced to a single-letter word one letter at a time by removing letters, leaving it a legal word at each step?”
The Startling Truth

STARTLING
Is there really just one nine-letter word with this property?
Iterative?
Recursive?
Decision Tree?
Reduce Decision Tree

CART

ART

CRT

CAT

CAR

RT  AT  AR

RT  CT  CR

AT  CT  CA

AR  CR  CA
bool search(currentState) {
  if(isSolution(currentState)) {
    return true;
  } else {
    for(option : moves from currentState) {
      nextState = takeOption(curr, option);
      if(search(nextState)){
        return true;
      }
    }
    return false;
  }
}
Let's define a **reducible word** as a word that can be reduced down to one letter by removing one character at a time, leaving a word at each step.

- **Base Cases:**
  - The empty string

- **Recursive Step:**
  - Any multi-letter word is reducible if you can remove a letter (legal move) to form a shrinkable word.
Is there really just one nine-letter word with this property?
Reducible Word

```cpp
bool reducible(Lexicon & lex, string word) {
    if(word.length() == 1 && lex.contains(word)) {
        return true;
    } else {
        for(int i=0; i < word.length(); i++) {
            string copy = word;
            copy.erase(i, 1);
            if(lex.contains(copy)) {
                if(reducible(lex, copy)) {
                    return true;
                }
            }
        }
        return false;
    }
}
```
bool reducable(Lexicon & lex, string word) {
    if(word.length()==1 && lex.contains(word)){
        return true;
    } else {
        for(int i=0; i < word.length(); i++) {
            string copy = word;
            copy.erase(i, 1);
            if(lex.contains(copy)){
                if(reducable(lex, copy)){
                    return true;
                }
            }
        }
        return false;
    }
}
bool reducable(Lexicon & lex, string word) {
    if(word.length()==1 && lex.contains(word)) {
        return true;
    } else {
        for(int i=0; i < word.length(); i++) {
            string copy = word;
            copy.erase(i, 1);
            if(lex.contains(copy)) {
                if(reducible(lex, copy)) {
                    return true;
                }
            }
        }
    }
    return false;
}
The function we have just defined is an example of recursive exploration of a tree. In this case we are looking for any path through the decision tree. For a given state:

- If *any* option leads to succeeds, that's great! We're done.

- If *none* of the options succeed, then this particular problem can't be solved from the state.
Visualize the Recursion
bool reducible(Lexicon & lex, string word) {
    if (word.length() == 1 && lex.contains(word)) {
        return true;
    } else {
        for (int i = 0; i < word.length(); i++) {
            string copy = word;
            copy.erase(i, 1);
            if (lex.contains(copy)) {
                if (reducible(lex, copy)) {
                    return true;
                }
            }
        }
    }
    return false;
}
bool reducible(Lexicon & lex, string word) {
    if(word.length()==1 && lex.contains(word)){
        return true;
    } else {
        for(int i=0; i < word.length(); i++) {
            string copy = word;
            copy.erase(i, 1);
            if(lex.contains(copy)){
                if(!reducible(lex, copy)){
                    return false;
                }
            }
        }
        return true;
    }
}
Ur Doin It Rong!
Generating All Permutations

Generate all permutations of a string

Eg: “abcd”

Eg: “abcdefghijklmnop”
Generating Permutations

<table>
<thead>
<tr>
<th>x_1</th>
<th>x_2</th>
<th>x_3</th>
<th>x_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1</td>
<td>x_2</td>
<td>x_3</td>
<td>x_4</td>
</tr>
<tr>
<td>x_1</td>
<td>x_2</td>
<td>x_4</td>
<td>x_3</td>
</tr>
<tr>
<td>x_1</td>
<td>x_3</td>
<td>x_2</td>
<td>x_4</td>
</tr>
<tr>
<td>x_1</td>
<td>x_3</td>
<td>x_4</td>
<td>x_2</td>
</tr>
<tr>
<td>x_1</td>
<td>x_4</td>
<td>x_2</td>
<td>x_3</td>
</tr>
<tr>
<td>x_1</td>
<td>x_4</td>
<td>x_3</td>
<td>x_2</td>
</tr>
</tbody>
</table>
What is the Permutation Tree?
Problems with the Same Underlying Form

1. List all permutations of a set of elements
2. Output the path of all files in a folder.
3. Find all words that are “reducible”
4. Play a game of knights tour solitaire
5. Find the optimal “alignment” of two strands of DNA

They have a similar underlying representation and they can all be solved with recursion
Your Brain is Recursive
You Can Imagine Many Next Steps
You Can Use the Same “Function” To Imagine Further Futures


Executes the same “function” with different inputs
Happy Friday
Knight’s Tour

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>40</td>
<td>47</td>
<td>44</td>
<td>61</td>
<td>08</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>46</td>
<td>43</td>
<td>36</td>
<td>41</td>
<td>14</td>
<td>11</td>
<td>62</td>
<td>09</td>
</tr>
<tr>
<td>39</td>
<td>34</td>
<td>45</td>
<td>48</td>
<td>07</td>
<td>60</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>50</td>
<td>55</td>
<td>42</td>
<td>37</td>
<td>22</td>
<td>17</td>
<td>10</td>
<td>63</td>
</tr>
<tr>
<td>33</td>
<td>38</td>
<td>49</td>
<td>54</td>
<td>59</td>
<td>06</td>
<td>23</td>
<td>18</td>
</tr>
<tr>
<td>56</td>
<td>51</td>
<td>28</td>
<td>31</td>
<td>26</td>
<td>21</td>
<td>03</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>32</td>
<td>53</td>
<td>58</td>
<td>05</td>
<td>02</td>
<td>19</td>
<td>24</td>
</tr>
<tr>
<td>52</td>
<td>57</td>
<td>30</td>
<td>27</td>
<td>20</td>
<td>25</td>
<td>04</td>
<td>01</td>
</tr>
</tbody>
</table>

Knight's Tour Demo
Knight’s Tour