Boggle YEAH Hours

Brahm Capoor

(Some slides adapted from Spring 2017 YEAH slides)
Road Map

Lecture review
Road Map

Lecture review

Assignment overview
Road Map

- - -

Lecture review
Assignment overview
Q&A!
Recursive Backtracking

Recursion vs backtracking

```cpp
bool subseq(string &s1, string &s2) {
    if (s2 == "") return true;
    if (s1 == "") return false;
    if (s1[0] == s2[0]){
        string r1 = s1.substr(1);
        string r2 = s2.substr(1);
        return subseq(r1, r2);
    } else {
        string r1 = s1.substr(1);
        return subseq(r1, s2);
    }
}
```
Recursion vs backtracking

```cpp
bool subseq(string &s1, string &s2) {
    if (s2 == "") return true;
    if (s1 == "") return false;
    if (s1[0] == s2[0]){
        string r1 = s1.substr(1);
        string r2 = s2.substr(1);
        return subseq(r1, r2);
    } else {
        string r1 = s1.substr(1);
        return subseq(r1, s2);
    }
}
```

- In recursion, you only ever do one recursive call at every level of the recursion
- In recursion, you know that your recursive call will work (it’s the leap of faith!)
Recursion vs backtracking

bool subseq(string &s1, string &s2) {
    if (s2 == "") return true;
    if (s1 == "") return false;
    if (s1[0] == s2[0]){
        string r1 = s1.substr(1);
        string r2 = s2.substr(1);
        return subseq(r1, r2);
    } else {
        string r1 = s1.substr(1);
        return subseq(r1, s2);
    }
}

string LCS(string &s1, string &s2) {
    if (s1 == "" || s2 == "") return "";
    if (s1[0] == s2[0]){
        string r2 = s2.substr(1);
        string r2 = s2.substr(1);
        return s1[0] + LCS(r1, r2);
    } else {
        string r1 = s1.substr(1);
        string r2 = s2.substr(1);
        string p1 = LCS(s1, r2);
        string p2 = LCS(r1, s2);
        if (p1.length() > p2.length()) {
            return p1;
        } else {
            return p2;
        }
    }
}
Recursion vs backtracking

bool subseq(string &s1, string &s2) {
    if (s2 == "") return true;
    if (s1 == "") return false;
    if (s1[0] == s2[0]){
        string r1 = s1.substr(1);
        string r2 = s2.substr(1);
        return subseq(r1, r2);
    } else {
        string r1 = s1.substr(1);
        return subseq(r1, s2);
    }
}

string LCS(string &s1, string &s2) {
    if (s1 == "" || s2 == "") return "";
    if (s1[0] == s2[0]){
        string r2 = s2.substr(1);
        string r1 = s1.substr(1);
        return s1[0] + LCS(r1, r2);
    } else {
        string r1 = s1.substr(1);
        string r2 = s2.substr(1);
        string p1 = LCS(s1, r2);
        string p2 = LCS(r1, s2);
        if (p1.length() > p2.length()) {
            return p1;
        } else {
            return p2;
        }
    }
}
Recursion vs backtracking

- Multiple recursive calls at every level of the function call
- Backtracking is about finding and weighing your options

```c++
string LCS(string &s1, string &s2) {
    if (s1 == "" || s2 == "") return "";
    if (s1[0] == s2[0]) {
        string r1 = s1.substr(1);
        string r2 = s2.substr(1);
        return s1[0] + LCS(r1, r2);
    } else {
        string r1 = s1.substr(1);
        string r2 = s2.substr(1);
        string p1 = LCS(s1, r2);
        string p2 = LCS(r1, s2);
        if (p1.length() > p2.length()) {
            return p1;
        } else {
            return p2;
        }
    }
}
```
Types of recursion & backtracking

Determine whether a solution exists
Types of recursion & backtracking

Determine whether a solution exists

Find a solution
Types of recursion & backtracking

Determine whether a solution exists

Find a solution

Find the best solution
Types of recursion & backtracking

- Determine whether a solution exists
- Find a solution
- Find the best solution
- Count the number of solutions
Types of recursion & backtracking

- Determine whether a solution exists
- Find a solution
- Find the best solution
- Count the number of solutions
- Print/find all the solutions
Types of recursion & backtracking

- Determine whether a solution exists
- Find a solution
- Find the best solution
- Count the number of solutions
- Print/find all the solutions

See midterm review slides for more detail!
Classes
// Person.cpp
Person::Person(string name) {
    this->name = name;
}

Person::string getName()
{
    return this->name;
}
// Person.h

class Person {
    public:
        // constructor(s)
        Person(string name)
            /*
             * Write sick (and public)
             * code prototypes here
             */
    private:
        string name;
        /*
         * Write sick (and secret)
         * code prototypes here
         */
};

// Person.cpp

Person::Person(string name) {
    this->name = name;
}

Person::string getName() {
    return this->name;
}

Another file, far far away (or not)

Person me = Person("Brahm");
cout << me.getName() << endl; //"Brahm"
Boggle!

Logistics:

- Due May 10
- Pair programming allowed!
  - Partner needs to be in the same section
The Rules
Starter code structure

---

boggleplay.cpp

“Client to perform console UI and work with your Boggle class to play a game”

“...not meant to be the place to store the majority of the game’s state, logic or algorithms...”

“...no recursion or backtracking should take place in boggleplay...”

Boggle.h & Boggle.cpp

“files for a Boggle class representing the state of the current Boggle game”

“the majority of your code”

“...required members...”
Starter code structure

---

boggleplay.cpp

“Client to perform console UI and work with your Boggle class to play a game”

“...not meant to be the place to store the majority of the game’s state, logic or algorithms...”

“...no recursion or backtracking should take place in boggleplay...”

Boggle.h & Boggle.cpp

“files for a Boggle class representing the state of the current Boggle game”

“the majority of your code”

“...required members...”

Also boggleui.h, but worry about this last!
Game Setup

- Drawing the board
  - Custom board
  - Shaking the cubes
    - Representing the cubes
    - Representing the board
    - Random locations and faces

#include “shuffle.h”

shuffle(array, length);

#include random.h

randomInteger(0,6);

#include <<cctype>

isalpha(ch);

#include “simpio.h”

getYesOrNo(prompt, reprompt);
Get a word from the user...

Make sure to error check!
Human Word Search

- Find where the word you’re searching for can start
- Recursively explore from this point
  - Is the public method enough, or do you need a helper function?
Types of recursion & backtracking

Determine whether a solution exists

Find a solution

Find the best solution

Count the number of solutions

Print/find all the solutions
Human Word Search

- Find where the word you’re searching for can start
- Recursively explore from this point
  - Is the public method enough, or do you need a helper function?
- An example (courtesy of previous YEAH hours)
**humanWordSearch Demo**

**word = “smart”**

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>M</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>
humanWordSearch Demo

word = “smart”
humanWordSearch  Demo

word = “smart”
humanWordSearch Demo

word = “smart”

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>M</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>
humanWordSearch Demo

word = “smart”

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>M</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
- Mark it as used
  - Why?
- Explore the rest of the word
**humanWordSearch Demo**

**word = “mart”**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T</td>
<td>R</td>
<td>E</td>
</tr>
<tr>
<td>S</td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>M</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
  - Mark it as used
    - Why?
  - Highlight square
  - Look at its neighbors for the second letter.
**humanWordSearch Demo**

**word = “mart”**

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Marked As Used</strong></td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>M</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
  - Mark it as used
  - Why?
- Highlight square
- Look at its neighbors for the second letter.
**humanWordSearch Demo**

**word = “mart”**

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Marked As Used</strong></td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>M</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
  - Mark it as used
    - Why?
  - Highlight square
  - Look at its neighbors for the second letter.
humanWordSearch Demo

word = “mart”

- We found the first letter
- Mark it as used
- Why?
- Highlight square
- Look at its neighbors for the second letter.
**humanWordSearch Demo**

**word = “mart”**

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>M</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
- Mark it as used
  - Why?
- Highlight square
- Look at its neighbors for the second letter.

Marked As Used
**humanWordSearch Demo**

**word = “mart”**

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Marked As Used</strong></td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>M</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
- Mark it as used
  - Why?
- Highlight square
- Look at its neighbors for the second letter.
**humanWordSearch Demo**

**word = “mart”**

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marked As Used</td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>U</th>
<th>M</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
  - Mark it as used
    - Why?
  - Highlight square
  - Look at its neighbors for the second letter.
**humanWordSearch Demo**

**word = “mart”**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T</td>
<td>R</td>
<td>E</td>
</tr>
<tr>
<td>Marked As Used</td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>M</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
  - Mark it as used
  - Why?
  - Highlight square
  - Look at its neighbors for the second letter.
word = “mart”

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>A</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>B</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
- Mark it as used
- Why?
- Highlight square
- Look at its neighbors for the second letter.

- Found it, now do it again.
humanWordSearch Demo

word = “art”

- We found the first letter
- Mark it as used
  - Why?
- Highlight square
- Look at its neighbors for the second letter.
word = “art”

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Marked As Used</strong></td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>B</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
  - Mark it as used
    - Why?
  - Highlight square
  - Look at its neighbors for the next letter.
**humanWordSearch Demo**

**word = “art”**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Marked As Used</td>
<td></td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Marked As Used</td>
<td></td>
<td>B</td>
<td></td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>A</td>
<td></td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
- Mark it as used
- Why?
- Highlight square
- Look at its neighbors for the next letter.

- Found the next letter! Let’s do it again.
**humanWordSearch Demo**

**word = “rt”**

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Marked As Used</strong></td>
<td></td>
<td><strong>Marked As Used</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Marked As Used</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td></td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- We found the first letter
- Mark it as used
  - Why?
- Highlight square
- Look at its neighbors for the next letter.
...a few steps later

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>R</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>N</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>M</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

- How do we know when we are here?
  - That’s our base case

- *What if that first “S” did not work out?*
  - Keep looking
The user ends their turn...
(by pressing enter)
Computer Word Search

- Find all the words on the board
- Also backtracking
Types of recursion & backtracking

Determine whether a solution exists
Find a solution
Find the best solution
Count the number of solutions
Print/find all the solutions
Computer Word Search

- Find all the words on the board
- Also backtracking
- When do you stop?
  - It can’t be when you find a word
  - Once you’ve found “ban”, you can still find “banter”

```javascript
lexicon.containsPrefix(pre);
// pre is a possible string prefix
```
Computer Word Search

- Find all the words on the board
- Also backtracking
- When do you stop?
  - It can’t be when you find a word
  - Once you’ve found “ban”, you can still find “banter”
- An example (courtesy of previous YEAH hours)

```java
lexicon.containsPrefix(pre);
// pre is a possible string prefix
```
computerWordSearch()  Demo

word so far: “E”

Select each neighbor in turn and recurse down.
computerWordSearch() Demo

word so far: “EA”

Select each neighbor in turn and recurse down.

Marked As Used → A

S R A R

U V K H

M E J O
word so far: “EAQ”

Select each neighbor in turn and recurse down.

BUT WAIT! EAQ is not the start of any English word! So should we continue??
computerWordSearch() Demo

word so far: “EA”

Select each neighbor in turn and recurse down.
computerWordSearch() Demo

word so far: “EAS”

Select each neighbor in turn and recurse down.
word so far: “EASR”

Select each neighbor in turn and recurse down.

But wait, no word begins with EASR!
computerWordSearch() Demo

word so far: “EAS”

Select each neighbor in turn and recurse down.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>Q</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>R</td>
<td>A</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>V</td>
<td>K</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>E</td>
<td>J</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>

Marked As Used

Marked As Used
computerWordSearch() Demo

word so far: “EASU”

Select each neighbor in turn and recurse down.

But wait, no word begins with “EASU”!
**computerWordSearch() Demo**

**word so far: “EAS”**

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>R</th>
<th>A</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>V</td>
<td>K</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>E</td>
<td>J</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>

Select each neighbor in turn and recurse down.

Marked As Used | Marked As Used | Q   | E   |

Marked As Used | Marked As Used | Q   | E   |
**computerWordSearch() Demo**

word so far: “EASV”

Select each neighbor in turn and recurse down.

STOP! No words start with “EASV”!
computerWordSearch() Demo

word so far: “EAS”

<table>
<thead>
<tr>
<th>Marked As Used</th>
<th>Marked As Used</th>
<th>Q</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>R</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>V</td>
<td>K</td>
<td>H</td>
</tr>
<tr>
<td>M</td>
<td>E</td>
<td>J</td>
<td>O</td>
</tr>
</tbody>
</table>

Select each neighbor in turn and recurse down.

We have looked at all of S’s neighbors, so we will head back up.
Marked As Used

Select each neighbor in turn and recurse down.
computerWordSearch() Demo

word so far: “EAR”

<table>
<thead>
<tr>
<th>Marked As Used</th>
<th>Marked As Used</th>
<th>Q</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>R</td>
<td>A</td>
<td>R</td>
</tr>
<tr>
<td>U</td>
<td>V</td>
<td>K</td>
<td>H</td>
</tr>
<tr>
<td>M</td>
<td>E</td>
<td>J</td>
<td>O</td>
</tr>
</tbody>
</table>

Select each neighbor in turn and recurse down.

“EAR” is a word, **but it is not 4 letters.**
computerWordSearch() Demo

word so far: “EARS”

Select each neighbor in turn and recurse down.

“EARS”! Hey, that’s a word and it’s 4 letters at least.

Let’s add it to our set, and keep looking!
computerWordSearch() Demo

word so far: “EARSU”

Select each neighbor in turn and recurse down.
Time for the GUI!

Figure out what each function in boglegui.h does and how/when to use it.

BoggleGUI::initialize(row, col) if you want to call initialize(row, col)
endl;

// questions?