Today’s topics:

- Recursion Week Fortnight continues!
- Today:
  - Loops + recursion for *generating sequences and combinations*
- Upcoming:
  - Loops + recursion for *recursive backtracking*
Heads or Tails?

GENERATING SEQUENCES
Heads or Tails?

- You flip a coin 5 times
- What are all the possible heads/tails sequences you could observe?
  - TTTTT
  - HHHHH
  - THTHT
  - HHHHT
  - etc…
- We want to write a program to fill a Vector with strings representing each of the possible sequences.
Generating all possible coin flip sequences

```cpp
void generateAllSequences(int length, Vector<string>& allSequences) {
    string sequence;
    generateAllSequences(length, allSequences, sequence);
}

void generateAllSequences(int length, Vector<string>& allSequences, string sequence) {
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

Generating all possible coin flip sequences
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}

Your Turn: coin flip sequences

Q: Of these sequences (all of which should be included in allSequences), which sequence appears first in allSequences? Last?

- TTTTT, HHHHH, THTHT, HHHHT
Helpful mental models for recursion: the call stack, and the call tree

Remember we used this to help us understand **Factorial** recursion:

```
main()
myfunction(x): 4
xfac: 0
factorial(n): 4
```

Remember we used this to help us understand **Fibonacci** recursion:

```
N=5
  N=4
    N=3
      N=2
        N=1
          N=0
    N=2
      N=1
        N=0
  N=3
    N=2
      N=1
        N=0
    N=1
      N=0
```

Stanford University
Recursive code

```c
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```
Recursive code

```cpp
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```
Recursive code

```cpp
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

Remember how pass-by-value works is that the function being called gets its own copy of the data, so we start with both functions having the same string.
Call stack for our Heads/Tails code

Recursive code

```cpp
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```
Recursive code

```java
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

---

**Call stack for our Heads/Tails code**

<table>
<thead>
<tr>
<th>sequence:</th>
<th>&quot;H&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence:</td>
<td>&quot;HH&quot;</td>
</tr>
<tr>
<td>sequence:</td>
<td>&quot;HH&quot;</td>
</tr>
</tbody>
</table>

---

**Stanford University**
Recursive code

void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
Call stack for our Heads/Tails code

| sequence: | "H" |
| sequence: | "HH" |
| sequence: | "HHH" |
| sequence: | "HHHH" |

Recursive code

```cpp
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```
Call stack for our Heads/Tails code

Recursive code

```c
void generateAllSequences(int length, vector<String>& allSequences, String sequence)
{
    // base case: this sequence is full - length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

Finally hit base case! Add this sequence HHHH to our list of possible coin toss sequences of length 4, and return.

<table>
<thead>
<tr>
<th>sequence</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;H&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;HH&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;HHH&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;HHHH&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;HHHH&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Text, Heap
Call stack for our Heads/Tails code

Recursive code
```cpp
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length)
    {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

Most recent stack frame “popped” off the stack when we returned.

We come back to this next line that says to erase the H we added before the function call.
Recursive code

```cpp
void generateAllSequences(int length, Vector<string>& allSequences, string sequence) {
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

We come back to this next line that says to erase the H we added before the function call.

Erased the 4th H
Recursive code

```plaintext
void generateAllSequences(int length, Vector<string>& allSequences, string sequence) {
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

Call stack for our Heads/Tails code

<table>
<thead>
<tr>
<th>sequence:</th>
<th>&quot;H&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence:</td>
<td>&quot;HH&quot;</td>
</tr>
<tr>
<td>sequence:</td>
<td>&quot;HHH&quot;</td>
</tr>
<tr>
<td>sequence:</td>
<td>&quot;HHHT&quot;</td>
</tr>
</tbody>
</table>

Added a T

Now a second recursive call with this new sequence.
Recursive code

```c
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

At the base case again, we add HHHT to our collection as the second completed sequence, and return.
Recursive code

```c
void generateAllSequences(int length, Vector<string>& allSequences, string sequence) {
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

This function call has reached the end (did both recursive calls), so it is done and it returns.
Recursive code

```cpp
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

This function call now needs to erase its H and add a T and try again.
Recursive code:

```c++
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

Call stack for our Heads/Tails code:

<table>
<thead>
<tr>
<th>sequence</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;H&quot;</td>
</tr>
<tr>
<td>sequence</td>
<td>&quot;HH&quot;</td>
</tr>
<tr>
<td>sequence</td>
<td>&quot;HHT&quot;</td>
</tr>
</tbody>
</table>

Now ends in T.
Call stack for our Heads/Tails code

<table>
<thead>
<tr>
<th>sequence:</th>
<th>&quot;H&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence:</td>
<td>&quot;HH&quot;</td>
</tr>
<tr>
<td>sequence:</td>
<td>&quot;HHT&quot;</td>
</tr>
<tr>
<td>sequence:</td>
<td>&quot;HHT&quot;</td>
</tr>
</tbody>
</table>

Recursive code

```cpp
void generateAllSequences(int length, Vector<string>& allSequences, string sequence) {
  // base case: this sequence is full-length and ready to add
  if (sequence.size() == length) {
    allSequences.add(sequence);
    return;
  }

  // recursive cases: add H or T and continue
  sequence += "H";
  generateAllSequences(length, allSequences, sequence);
  sequence.erase(sequence.size() - 1);
  sequence += "T";
  generateAllSequences(length, allSequences, sequence);
}
```
Recursive code

```java
void generateAllSequences(int length, Vector<String>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```
void generateAllSequences(int length, Vector<string> &allSequences, string sequence)
{
    // base case: this sequence is full length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```c++
void generateAllSequences(int length, Vector<string>& allSequences, string sequence) {
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```
void generateAllSequences(int length, Vector<String>& allSequences, string sequence)
{
    // base case: this sequence is full and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
Recursive code

```cpp
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

We did both H and T recursive calls, so this function is done and also returns.
Call stack for our Heads/Tails code

Recursive code

```cpp
void generateAllSequences(int length, Vector<string>& allSequences, string sequence) {
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

We did both H and T recursive calls, so this function is done and also returns.
Recursive code

```cpp
void generateAllSequences(int length, Vector<string>& allSequences, string sequence) {
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}
```

This function still needs to erase its H, and try T, but we’ll end our animation here. 😊
Call tree for Heads/Tails code

Labels are based on the value of the sequence parameter at the time of the function call (without add/erase/add edits we make inside the function).
void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}

Q: What would happen if we didn’t do the erase (highlighted above)? Which of the following sequences would we NOT generate? Which additional sequences would we generate (that we shouldn’t)?

› TTTTT, HHHHH, THTHT, HHHHT
Roll the Dice!

GENERATING MORE SEQUENCES
Roll the Dice!

- You roll a single die 5 times
- What are all the possible 1/2/3/4/5/6 sequences you could observe?
  - 11111
  - 66666
  - 12345
  - 21655
  - etc…
- We want to write a program to fill a Vector with strings representing each of the possible sequences.
Generating all possible coin-flip die roll sequences

void generateAllSequences(int length, Vector<string>& allSequences) {
    string sequence;
    generateAllSequences(length, allSequences, sequence);
}

void generateAllSequences(int length, Vector<string>& allSequences, string sequence) {
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add H or T and continue
    sequence += "H";
    generateAllSequences(length, allSequences, sequence);
    sequence.erase(sequence.size() - 1);
    sequence += "T";
    generateAllSequences(length, allSequences, sequence);
}

Generating all possible coin-flip die roll sequences

To adapt for die rolls, we need to change this from 2 options (H/T) to 6 options (1-6).
Generating all possible coin-flip die roll sequences

// recursive cases: add 1 or 2 or 3 or 4 or 5 or 6 and continue
sequence += "1";
generateAllSequences(length, allSequences, sequence);
sequence.erase(sequence.size() - 1);
sequence += "2";
generateAllSequences(length, allSequences, sequence);
sequence.erase(sequence.size() - 1);
sequence += "3";
generateAllSequences(length, allSequences, sequence);
sequence.erase(sequence.size() - 1);
sequence += "4";
generateAllSequences(length, allSequences, sequence);
sequence.erase(sequence.size() - 1);
sequence += "5";
generateAllSequences(length, allSequences, sequence);
sequence.erase(sequence.size() - 1);
sequence += "6";
generateAllSequences(length, allSequences, sequence);
}
Generating all possible coin-flip die roll sequences

// recursive cases: add 1 or 2 or 3 or 4 or 5 or 6 and continue
sequence += "1";
generateAllSequences(length, allSequences, sequence);
sequence.erase(sequence.size() - 1);
sequence += "2";
generateAllSequences(length, allSequences, sequence);
sequence.erase(sequence.size() - 1);
sequence += "3";
generateAllSequences(length, allSequences, sequence);
sequence.erase(sequence.size() - 1);
sequence += "4";
generateAllSequences(length, allSequences, sequence);
sequence.erase(sequence.size() - 1);
sequence += "5";
generateAllSequences(length, allSequences, sequence);
sequence.erase(sequence.size() - 1);
sequence += "6";
generateAllSequences(length, allSequences, sequence);

}
Generating all possible coin-flip die roll sequences

```cpp
void generateAllSequences(int length, Vector<string>& allSequences) {
    string sequence;
    generateAllSequences(length, allSequences, sequence);
}

void generateAllSequences(int length, Vector<string>& allSequences, string sequence) {
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add 1-6 and continue
    for (int i = 1; i <= 6; i++) {
        sequence += integerToString(i);
        generateAllSequences(length, allSequences, sequence);
        sequence.erase(sequence.size() - 1);
    }
}
```

Much nicer!!
Generating all possible coin-flip die roll sequences

```cpp
void generateAllSequences(int length, Vector<string>& allSequences) {
    string sequence;
    generateAllSequences(length, allSequences, sequence);
}

void generateAllSequences(int length, Vector<string>& allSequences, string sequence) {
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }
    // recursive cases: add 1-6 and continue
    for (int i = 1; i <= 6; i++) {
        sequence += integerToString(i);
        generateAllSequences(length, allSequences, sequence);
        sequence.erase(sequence.size() - 1);
    }
}
```

Notice that this loop does not replace the recursion. It just controls how many times the recursion launches.
Your Turn: die roll sequences

void generateAllSequences(int length, Vector<string>& allSequences, string sequence)
{
    // base case: this sequence is full-length and ready to add
    if (sequence.size() == length) {
        allSequences.add(sequence);
        return;
    }

    // recursive cases: add 1-6 and continue
    for (int i = 1; i <= 6; i++) {
        sequence += integerToString(i);
        generateAllSequences(length, allSequences, sequence);
        sequence.erase(sequence.size() - 1);
    }
}

- Q: Of these sequences (all of which should be included in allSequences), which sequence appears first in allSequences? Last?
  - 11111, 66666, 12345, 21655