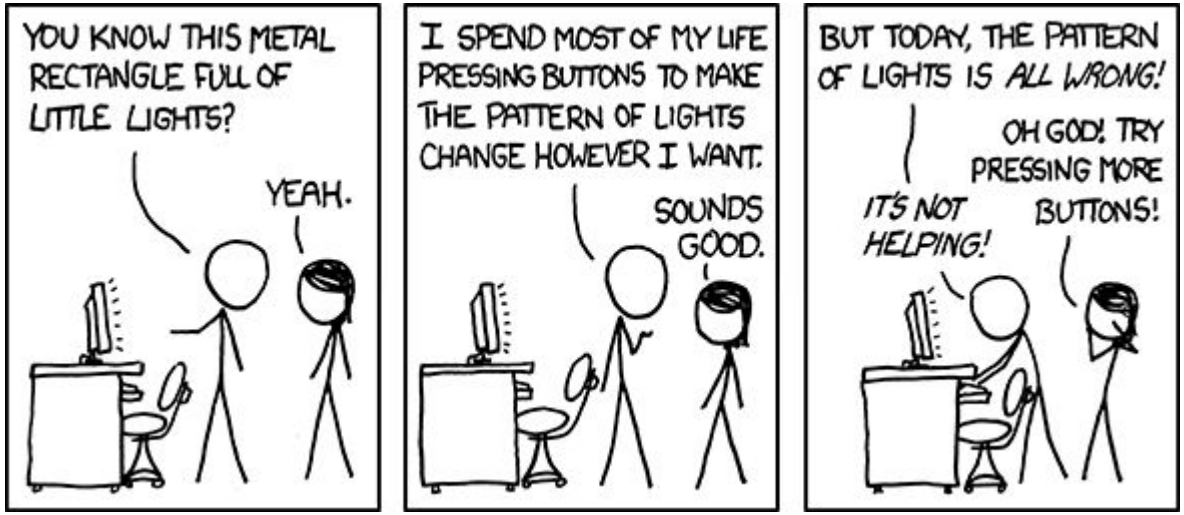
YEAH - ADTs

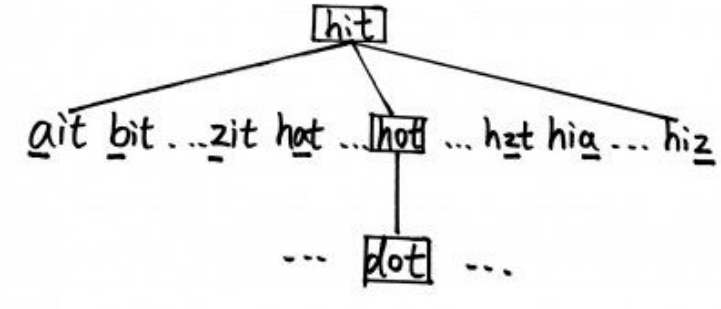
Anton Apostolatos



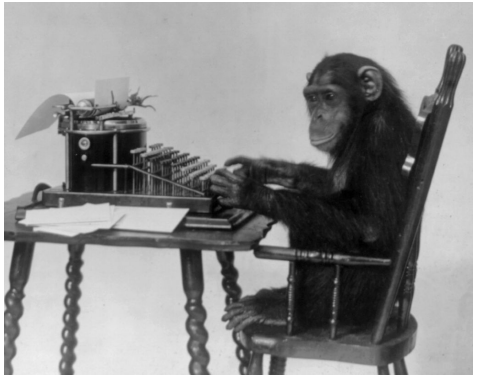
Source: XKCD

A2: ADTs

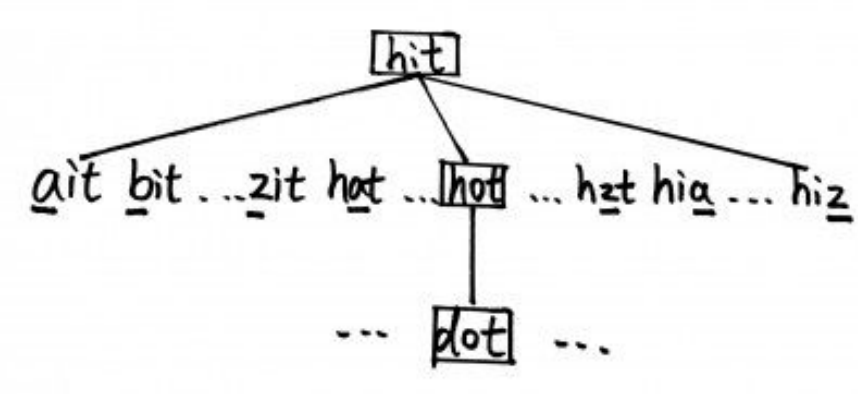
Word Ladders



Random Writer



Word Ladders



A **word ladder** is a connection from one word to another, where:

1) Each word is one character different than the previous

`map` → `mat` ✓ `map` → `sit` ✘

2) Every word in the ladder is valid

`blame` → `bhame` → `shame` ✘

3) Shortest possible!

`bit` → `fit` ✓ `bit` → `sit` → `fit` ✘

Demo!

Pseudocode

```
create an empty queue
add the start word to the end of the queue

while (the queue is not empty):
    dequeue the first ladder from the queue

    if (the final word in this ladder is the destination word):
        return this ladder as the solution


    for (each word in the lexicon of English words that differs by one):
        if (that word has not been already used in a ladder):
            create a copy of the current ladder
            add the new word to the end of the copy
            add the new ladder to the end of the queue


return that no word ladder exists
```

How do we know it's the shortest path?

Starter code - `wordladder.cpp`

```
#include <cctype>
#include <cmath>
#include <fstream>
#include <iostream>
#include <string>
#include "console.h"
using namespace std;

int main() {
   // TODO: Finish the program!
   cout << "Have a nice day." << endl;
   return 0;
}
```

Design Decision

How to store ladder? Seen words?

Steps

1. **Load the dictionary.** The file `EnglishWords.dat`, which is bundled with the starter files, contains just about every legal English word.

2. **Prompt the user for two words to try to connect with a ladder.** For each of those words, make sure to reprompt the user until they enter valid English words. They don't necessarily have to be the same length, though – if they aren't, it just means that your search won't find a word ladder between them.

3. **Find the shortest word ladder.** Use breadth-first search, as described before, to search for a word ladder from the first word to the second.

Steps II

4. **Report what you've found.** Once your breadth-first search terminates:
    a. If you found a word ladder, print it out to the console.
    b. If you don't find a word ladder, print out a message to that effect.
5. **Ask to continue.** Prompt for whether to look for another ladder between a pair of words.

Tips and Tricks

- **Pick data structures wisely:** not all ADTs are made equal

- **Watch out for case sensitivity**

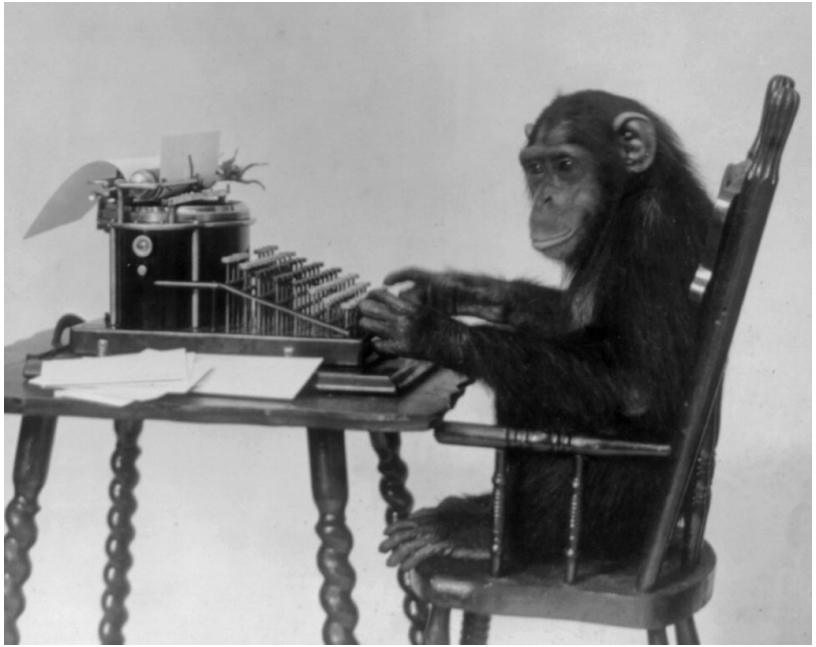
`Work` ⟷ `wOrK`

- **Ties don't matter:** don't worry about multiple ladders of the same length

`bit` → `fit` → `fat` ✓ `bit` → `bat` → `fat` ✓

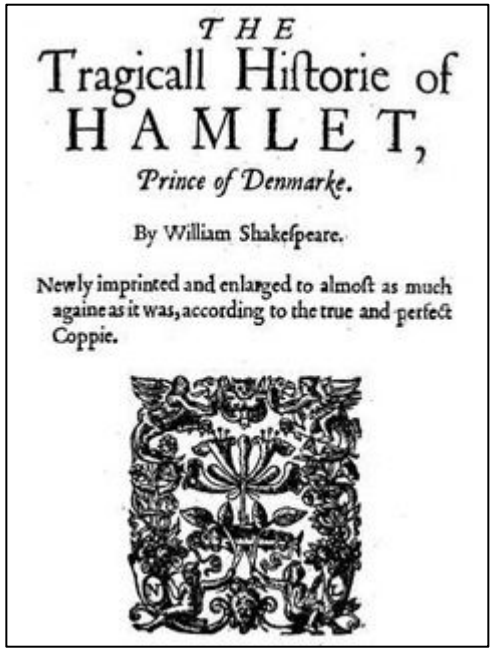
Questions?

Random Writer



Infinite Monkey Theorem

"A monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will almost surely type [...] the complete works of William Shakespeare." - *Wikipedia*


THE
Tragicall Hiſtorie of
HAMLET,
Prince of Denmarke.

By William Shakeſpeare.

Newly imprinted and enlarged to almoſt as much againe as it was, according to the true and perfect Coppie.

Original text

“To be or not to be just be who you want to be or not okay you want okay”

3-grams

```
{ {to, be} : {or, just, or},
  {be, or} : {not, not},
  {or, not} : {to, okay},
  {not, to} : {be},
  {be, just} : {be},
  {just, be} : {who},
  {be, who} : {you},
  {who, you} : {want},
  {you, want} : {to, okay},
  {want, to} : {be},
  {not, okay} : {you},
  {okay, you} : {want},
  {want, okay} : {to},
  {okay, to} : {be} }
```

Made-up text

... chapel. Ham. Do not believe his tenders, as you go to this fellow. Whose grave's ...

Connects a collection of N - 1 words to all Nth words that follow it in the text

Demo!

Step 1: Build Map

```
Map<String, int> phonebook;
```

Key

Value

to be | or not to be just ...

```
map    = {}
window = {to, be}
```

Note that window is of size N-1!

to be or | not to be just ...

```
map    = { {to, be} : {or} }
window = {be, or}
```

to be or not | to be just ...

```
map    = { {to, be} : {or},
           {be, or} : {not} }
window = {or, not}
```

to be or not to | be just ...

```
map    = { {to, be} : {or},
           {be, or} : {not},
           {or, not} : {to} }
window = {not, to}
```

to be or not to be just
be who you want to be
or not okay you want okay

```
map     = { {to, be} : {or, just, or},
            {be, or} : {not, not},
            {or, not} : {to, okay},
            {not, to} : {be},
            {be, just} : {be},
            {just, be} : {who},
            {be, who} : {you},
            {who, you} : {want},
            {you, want} : {to, okay},
            {want, to} : {be},
            {not, okay} : {you},
            {okay, you} : {want},
            {want, okay} : {to},
            {okay, to} : {be} }
```

Wrapping!

How can we implement wrapping...?

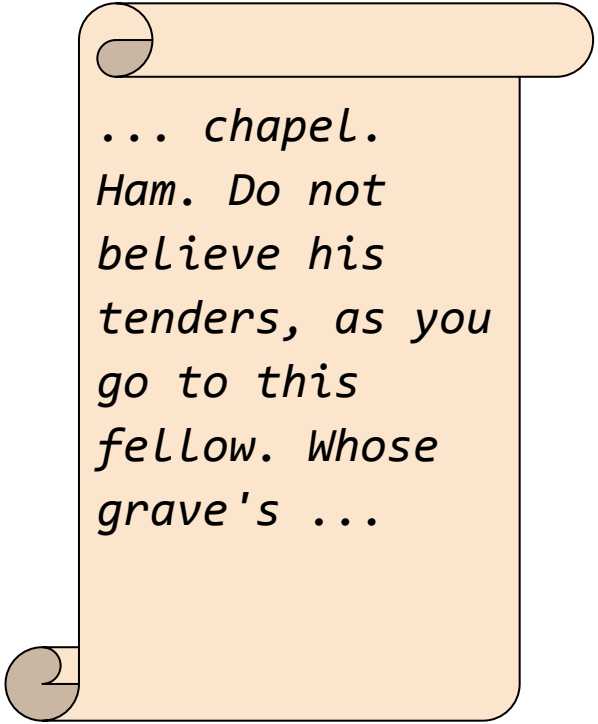
Design Decision

How do we store keys / values in the Map?

Step 2: Generate Random Text

Generating Random Text

1. Pick a random key in your map
2. For each subsequent word randomly choose one using last two words in generated text
3. Repeat (2) until complete!



Tips and Tricks

- Think about the collections you want to use in every case. Plan ahead.
- Test each function with small input (`tiny.txt`)
- To choose a random prefix from a map, consider using the map's keys member function, which returns a Vector containing all of the keys in the map.
- For randomness in general, check out "random.h".
- You can loop over the elements of a vector or set using a for-each loop. A for-each also works on a map, iterating over the keys in the map.

Questions?