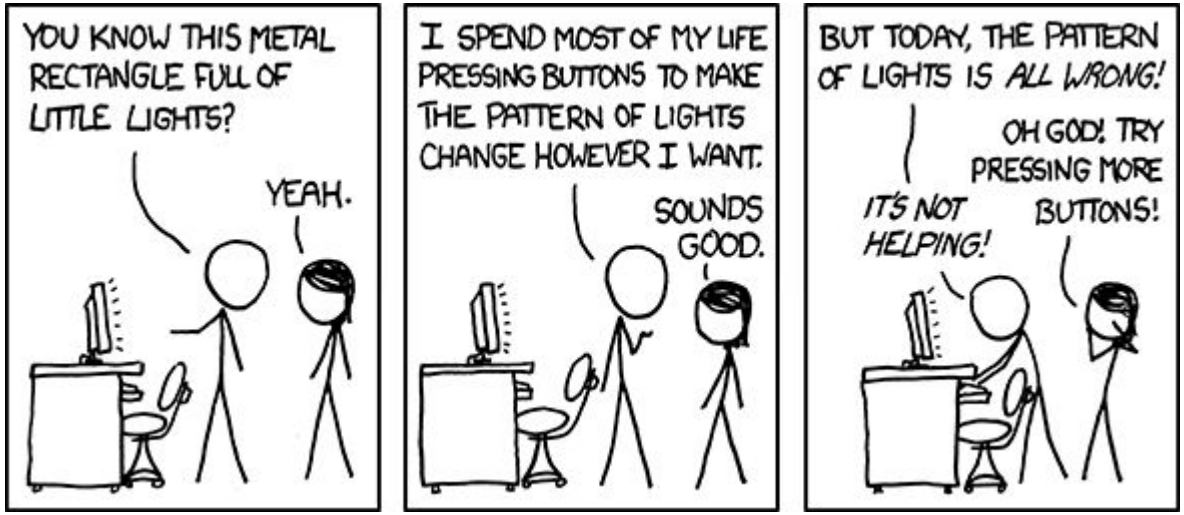
YEAH - Serafini

Jason Chen

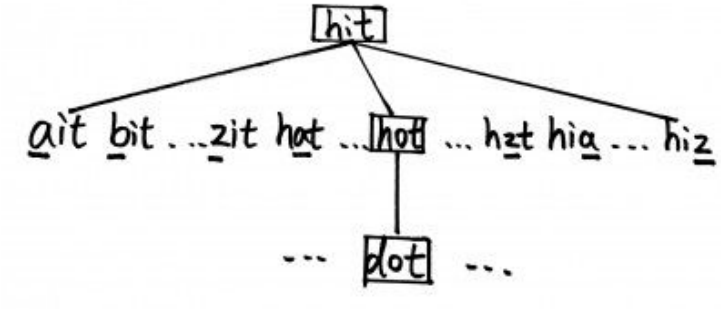
Original slides by: Anton Apostolatos



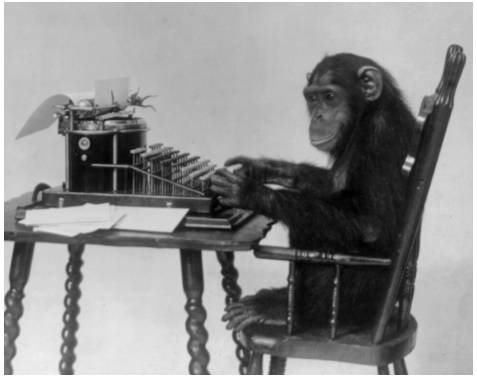
Source: XKCD

A2: Serafini

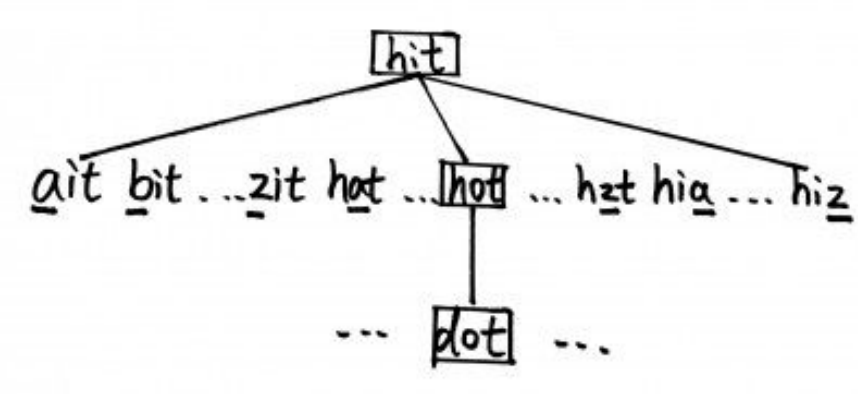
Word Ladders



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!

```
Welcome to CS 106B Word Ladder.
Please give me two English words, and I will change the
first into the second by changing one letter at a time.

Dictionary file name? dictionary.txt

Word #1 (or Enter to quit): code
Word #2 (or Enter to quit): data
A ladder from data back to code:
data date cate cade code

Word #1 (or Enter to quit):
Have a nice day.
```

```
Dictionary file name? notfound.txt
Unable to open that file.  Try again.
Dictionary file name? oops.txt
Unable to open that file.  Try again.
Dictionary file name? smalldict1.txt

Word #1 (or Enter to quit): ghost
Word #2 (or Enter to quit): boo
The two words must be the same length.

Word #1 (or Enter to quit): marty
Word #2 (or Enter to quit): keith
The two words must be found in the dictionary.

Word #1 (or Enter to quit): kitty
Word #2 (or Enter to quit): kitty
The two words must be different.
```

```
Dictionary file name? dictionary.txt

Word #1 (or Enter to quit): metal
Word #2 (or Enter to quit): azure
No word ladder found from azure back to metal.

Word #1 (or Enter to quit): kwyjibo
Word #2 (or Enter to quit): fluxbar
The two words must be found in the dictionary.

Word #1 (or Enter to quit): monkey
Word #2 (or Enter to quit): monkey
The two words must be different.

Word #1 (or Enter to quit): partial
Word #2 (or Enter to quit):
Have a nice day.
```

Pseudocode

```
create an empty queue
add the start word to a ladder. then add the ladder 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 letter):
        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?

Design Decision

How to store ladder? Seen words?

How to store ladder?

Queue<Stack>

Stack<Queue>

Stack<Stack>

Queue<Queue>

A short comparison of stacks vs queues

code => data

{bode, core, mode ...} [1 letter away]

if using a stack:

{bade, bide, bore, core, mode ...}

if using a queue:

{core, mode, bade, bide, bore ...}

How to store seen words?

SET

Finding “neighbors”

1. some measure of distance is implicit

2. for each dimension, explore all options within a certain distance

	Game of Life	Word Ladder
Dimensions:	x / y	word length
All options:	{ -1, 0, 1}	{a - z}

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;
}
```

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

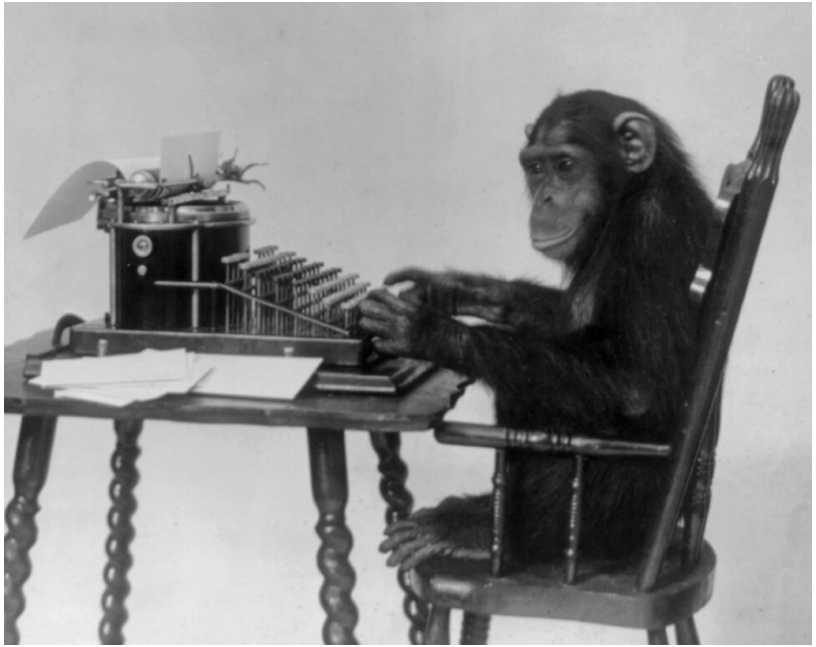
`bit` → `fit` → `fat` ✓ `bit` → `bat` → `fat` ✓

- **Passing variables by reference:** Try passing in the Lexicon by value and by reference and just watch the difference in runtime! Think about what other variables you should be passing by reference.

Questions?

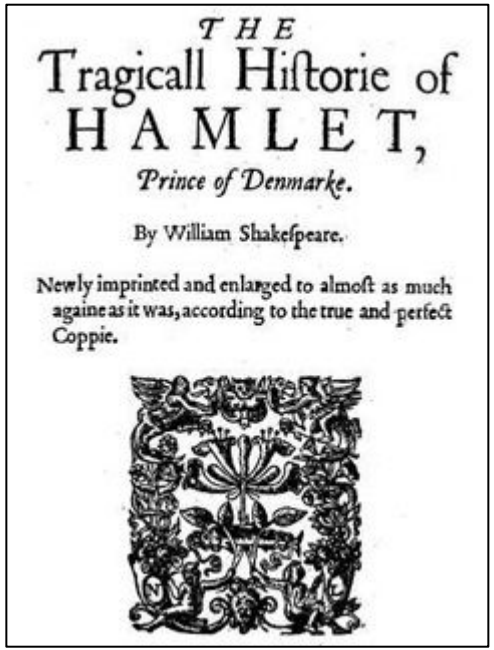
http://web.stanford.edu/class/cs106b/assn/serafini.pdf

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"

Build Map

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} }
```

Generate Random Text

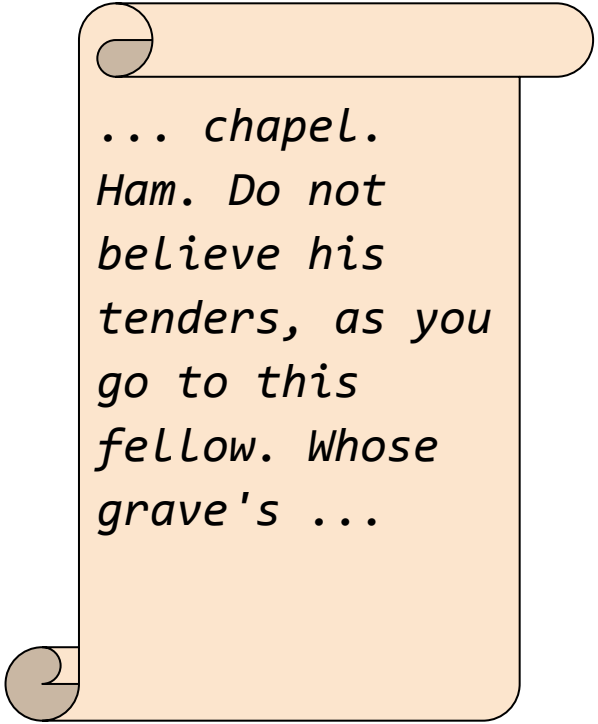
Made-up text

... [fill in during YEAH hours] ...

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



N-Gram Fun Facts

https://books.google.com/ngrams

http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

https://web.stanford.edu/~jurafsky/slp3/4.pdf
[language modeling]

What is the tradeoff between smaller and larger values of N?

Demo!

```
Welcome to CS 106B Random Writer ('N-Grams').
This program makes random text based on a document.
Give me an input file and an 'N' value for groups
of words, and I'll create random text for you.

Input file name? tiny.txt
Value of N? 3

# of random words to generate (0 to quit)? 8
... or not to be or not okay you ...

# of random words to generate (0 to quit)? 20
... be who you want to be or not to be just be who you want to be or not okay
...

# of random words to generate (0 to quit)? 0
Exiting.
```

```
Input file name? badfile
Unable to open that file.  Try again.
Input file name? notfound.txt
Unable to open that file.  Try again.
Input file name? hamlet.txt
Value of N? 0
N must be 2 or greater.
Value of N? -4
N must be 2 or greater.
Value of N? aoeu
Illegal integer format. Try again.
Value of N? 4

# of random words to generate (0 to quit)? xyz
Illegal integer format. Try again.
# of random words to generate (0 to quit)? 2
Must be at least 4 words.
```

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...?

Wrapping - why do we wrap?

1. wrapping gives the user a gracious handling of edge cases

2. you can think of wrapping as essentially an approximation of the truth

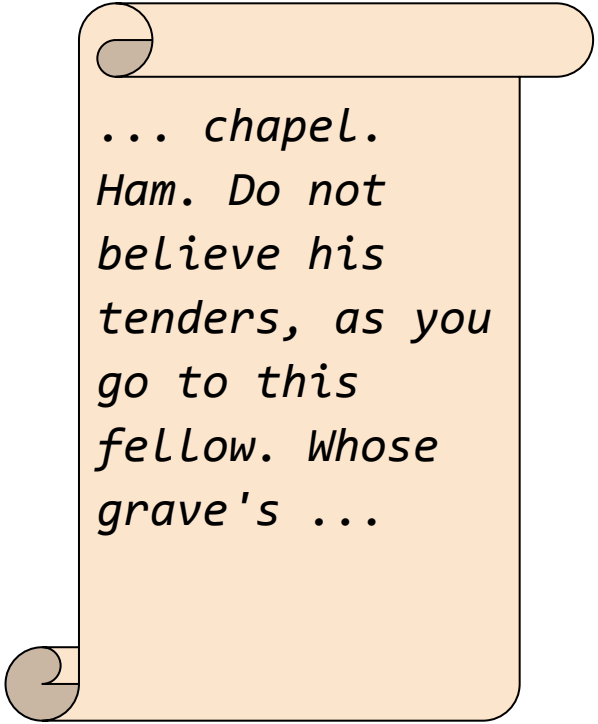
Design Decision

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?

http://web.stanford.edu/class/cs106b/assn/serafini.pdf

filelib.h

set.h

simpio.h

stack.h

map.h

queue.h

spellcheck

Donald J. Trump
@realDonaldTrump

I am honered to serve you, the great American People, as your 45th President of the United States!

2017. 01. 21. 17:57

1 878 RETWEETS **6 980** LIKES

“I am honered to serve you…”

1. word ladder [distance]
   hovered, honeyed, honored [1 letter away]

2. ngram map [frequency]
   {“am” : {tired, honored, honored, hovered} … }


you got this
INEVERYTHING.CA