YEAH!

Serafini

Sahil Chopra - 1.19.2016

Adapted from SLs Rishi Bedi & Audrey Ho

Part 1: Word Ladder

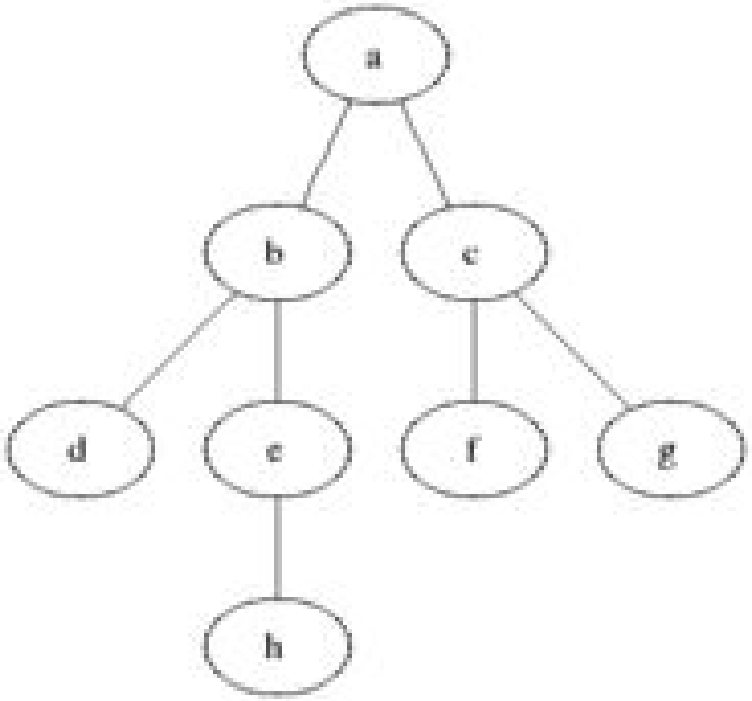
```
Welcome to CS 106B Word Ladder.
Please give me two English words, and I will change the
first into the second by changing one letter at a time.

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.
```

Word Ladder: Overview

- Input: Given start word & destination word
- Output: Create “ladder” of words from start to destination
- Goal: Find shortest ladder (fewest number of changes)
- Ignore Case

```
string lower = toLowerCase(input_string);
```

Word Ladder: BFS (Breadth First Search)



https://commons.wikimedia.org/wiki/File:Animated_BFS.gif

Word Ladder: BFS (Breadth First Search)

	The	Shy
Queue of Paths	Current Path	Neighbors
{the}	{}	{}

Word Ladder: BFS (Breadth First Search)

	The	Shy
Queue of Paths	Current Path	Neighbors
{}	{the}	{tie, she, tee}

Word Ladder: BFS (Breadth First Search)

Queue of Paths	The Current Path	Shy Neighbors
{the, tie}	{}	{}
{the, she}	{}	{}
{the, tee}	{}	{}

Word Ladder: BFS (Breadth First Search)

	The	Shy
Queue of Paths	Current Path	Neighbors
{the, she}	{the, tie}	{lie, tin, ~~the~~}
{the, tee}		

Word Ladder: BFS (Breadth First Search)

	The	Shy
Queue of Paths	Current Path	Neighbors
{the, she}	{}	{}
{the, tee}		
{the, tie, lie}		
{the, tie, tin}		

Word Ladder: BFS (Breadth First Search)

The

Shy

Queue of Paths	Current Path	Neighbors
{the, tee}	{the, she}	{~~the~~, see, shy}
{the, tie, lie}		
{the, tie, tin}		

Word Ladder: BFS (Breadth First Search)

Queue of Paths	Current Path	Neighbors
	The	Shy
{the, tee}		
{the, tie, lie}		
{the, tie, tin}		
{the, she, see}		
{the, she, shy} ← DONE.		

Word Ladder - BFS: Algorithm

```
Finding a word ladder between words w1 and w2:
    Create an empty queue of stacks.
    Create/add a stack containing {w1} to the queue.
    While the queue is not empty:
        Dequeue the partial-ladder stack from the front of the queue.
        For each valid English word that is a "neighbor" (differs by 1 letter)
        of the word on top of the stack:
            If that neighbor word has not already been used in a ladder before:
                If the neighbor word is w2:
                    Hooray! we have found a solution.
                Otherwise:
                    Create a copy of the current partial-ladder stack.
                    Put the neighbor word on top of the copy stack.
                    Add the copy stack to the end of the queue.
```

Word Ladder: Error Checking

- Input Word Restrictions:
  - 2 Valid English Words
  - 2 Different Words
  - Same Length
- Error Examples:
  - Invalid Words: bygh -> asdf
  - Different Words: code -> code
  - Different Length: this -> things

Part 2: N-Grams!

- Input: File & Number (“N” of “N-Grams”)
- Output: “Randomly Generated” Sequence of Words
- Note: N = Size of Chunks “N-Gram”

```
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? hamlet.txt
Value of N? 3

# of random words to generate (0 to quit)? 40
... chapel. Ham. Do not believe his tenders, as you
go to this fellow. Whose grave's this, sirrah?
Clown. Mine, sir. [Sings] O, a pit of clay for to
the King that's dead. Mar. Thou art a scholar; speak
to it. ...

# of random words to generate (0 to quit)? 20
... a foul disease, To keep itself from noyance; but
much more handsome than fine. One speech in't I
chiefly lov'd. ...

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

N-Grams!: Reading Algorithm in Action

N: 3

File: “to be or not to be that is the question.” **Map:** {}

Window: {}

Next Word: “”

Create window of first N-1 words

N-Grams!: Reading Algorithm in Action

N: 3

File: “to be or not be that is the question.” **Maps:** {}

Windows: {“to”, “be”}

Next Word: “or”

Add to map

N-Grams!: Reading Algorithm in Action

N: 3

File: “to be or not to be that is the question” **Map:** {{“to”, “be”} ⇒ {“or”}}

Window: {“be”, “or”}

Next Word: “not”

Slide window over

N-Grams!: Reading Algorithm in Action

N: 3

File: “to be or not to be that is the question”

Window: {“is”, “the”}

Next Word: “question.”

Map: {{“to”, “be”} => {“or”, “that”}

{“be”, “or”} => {“not”}

{{“or”, “not”} => {“to”}

{“not”, “to”} => {“be”}

{“be”, “that”} => {“is”}

{“that”, “is”} => {“the”}

{“is”, “the”} => {“question.”}}

N-Grams!: Reading Algorithm

- What is our map of?
- Read file word by word
- Read first N-1 words and create a window with them
- Store both window and word that follows
- Slide the window across word by word throughout the rest of the file
- If you come across a window that is already a key in the map, add this next word to the list of next words

N-Grams!: Writing Algorithm

- From the user: # of words to generate =>
- Pick a random starting point: a random key from your map:

```
Vector<key_type> keys = map.keys();
```

- Get a random suffix for this randomly chosen prefix
- Get the collection of possible suffixes from the map
- Choose a random number to use as index
- Slide current window over to get the new prefix (adding in the suffix you just randomly selected)
- Repeat until you've outputted however many words they asked for