Column Transpose Decipher - ISI

May 21, 2017

Author: Thamme Gowda thammegowda.n@usc.edu

1 Column Transpose Decipher

This note book is an attempt to solve the text decryption problem posted at http://www.isi.edu/natural-language/people/transpose.html

1.1 Problem statement:

1. There are 10 rows, each of them are independent.
2. There are 50 columns.
3. Cryptographic tenchnique - transpose cipher
4. Traspose technique: Columns are reorded/shuffled
5. Key: the sequence of columns that when the message is read left to right yields english or english like message

Just like the majority of cracking techniques, the goal here is to search for a valid combination.

1.2 Challenges for Search:

1. **Tractability:** There are 50 Columns, meaning that there are $50! = 3.0414093e + 64$ combinations. So, an exhaustive search is expensive in terms of CPU cycles and time.
2. **Stop condition:** How do we know a combination of columns is a valid answer so that we can stop the search or atleast we can mark the state as a solution?

1.3 Approach:

1. **Tractability:**
   The blind search yields $50! = 3.0414093e + 64$ combinations which is not tractable. So, we need to incorporate intelligence search into the search. We may take advantage of english vocabulary to restrict the search.

   a. Detect early when the search is going in wrong/invalid branch and stop -- done in this notebook

   b. Rank the branches based on probability of yielding a valid search. For example, `th --> e` is more likely an answer than the `th --> x` -- *Not done in this notebook, may be it is not needed.*

2. **Stop Condition:**

a. All the columns are used/fit in the solution

b. The text in each row is segmented properly. That is, except the last word, all internal words are valid as per the dictionary. The last word may be incomplete since the text is cut off at 50th column.

c. There can be multiple candidate solutions. So the search continues to produce all possible solutions.

3. **The ranking**: If there are multiple candidates, then each one of them is ranked as per language model and the highest possible candidate is returned.

1.4 External requirements

1. `Dictionary` - dictionary of valid words -- Used in this notebook

    a. Includes **all** the words in given message

    b. Large enough to cover all the words, however minimal enough to restrict the search

2. `Language Model` - probabilities of occurrence of words, bigrams and hopefully trigrams -- *Not Yet used in this notebook* Since the input consists of 10 rows, together all the 10 rows serves the same purpose as a language model. In otherwords, it is extremely unlikely to fit gibberish text for all the 10 rows at the same time!

1.4.1 What could possibly go wrong

1. Solution not reached: Dictionary doesn't include at least one word that is neccessary to check the validity
2. Intractable search: Dictionary includes too many words or invalid words, thus making this search intractable.

2 Input Text and Dictionary

2.1 Input text

```
In [1]: import os, sys
        import numpy as np
        from copy import copy, deepcopy
        import requests
```

```
In [2]: eval_msg = ["dtjmeoftumhbhstehresweseeoatearthkteoouietohfyetri",
        "ehsinaeinctnecfdbouunuetomltoimtonsrsihyeognrcfesi",
        "heldoehwaenwkleomnaegnonwefrimimynhenopngotiwymltt",
        "ikthgatysiowiitdeedrofeeyexonroaneitthtntsnhleofhc",
        "oslmnemjrtoebeftegernesioiweaaksphluethuaraeahafth",
        "nfiehtuohugiandettheeoartttrbeybuiohvsatsdousnwona",
        "sltaeytgnnhaatgoehtkssulttroleulenaofiolgtwanlleli",
        "wmlptepmsttoehwchhieiavoehikrnwthaterecnltewacwele",
        "icnoovslantrebdweoifseundstntewanhioottaeatairetds",
        "cggoarrhsdewarhtipntiiretnowrddcahipwtgneoeboeldwa"]

        ref_msg_orig = ["thepanelisalsoexpectedtorecommendthatthewhitehouse",
        "theiraniangovernmenthasmaintainedthatitknowsnothin",
        "bymatchingthelowestpriceandenhancingservicehewasde",
        "thisfallherneighborhoodinthenortheasternpartofthis",
        "forcollegebasketballfansthisisasgoodaweekasyourego",
        "butaccordingtobarbershopproprietorsthenumberoffema",
        "anintriguingnewstudysuggeststhatwhatreallydrawspeo",
        "alsoonthursdaythelatestwinnersofthelifesciencesand",
        "whatmembersofbothpartiesbemoanedmorethananythingwa",
        "onekeyhurdleforteslainproducingthenewsmallercarwil"]

        ref_msg = ["heeemirhcletshlohttwhsnpuesecetottipoenadeamtoaexd",
        "tnreaaahninsnoototttnhiaihkvnnhimttwmtieadnrisegena",
        "raomnnaytihhgchawbsindcasveedreeipeeenngcethcltswi",
        "rrginhnhrllteantfttpaiashneshoeiehrbettshofodiroho",
        "eaerigtolelyeaarufakoglceesoifwsolsbshsdgoosnkbata",
        "nebtrdpueroribgffbhmsmcaeutaoseprrerprttoocioonbah",
        "aawituendgiriygswarlaernplnotseghydtssttwathgenusu",
        "eotsruilahtnridseaicennoasadnefwhteeenfltcostyslhs",
        "aeoaaebhabmtrnoniwtarwetgnfamthsoryhoedemhmnebspti",
        "mgreironluhrdleraowlniykwafluisreaeecdtehcenpolstn"]

        devT = np.array(list(map(list, ref_msg))) # table
        origT = np.array(list(map(list, ref_msg_orig)))
        evalT = np.array(list(map(list, eval_msg)))
```

2.2 Building the dictionary

To begin with, I used wordnet synsets as dictionary. However, I found that wordnet is not a comprehensive list of english words, (for example - I could not found `'is'`, it is stored as `'be'`).

So I started to look for other comprehensive dictionaries and found these two:

1. http://www-01.sil.org/linguistics/wordlists/english/wordlist/wordsEn.txt
2. https://raw.githubusercontent.com/first20hours/google-10000-english/master/20k.txt

These lists seems comprehensive, however it also includes many short words such as `ab`, `ac`, `ii` etc . These short words doesnt sound like english words, but they are very easily reached during the search. As a result, the search complexity increases exponentially.

As a quick fix, scraped common words from OGDEN's Basic English Combined Word Lists at http://ogden.basic-english.org/word2000.html Then the words that are 2 characters or lesser but doesnt exists in the OGDEN's list are removed.

Then, I used Trie data structure to efficiently store and retrieve all the words in dictionary.

2.2.1 Trie Datastructure

```
In [3]: class TrieNode(object):

            def __init__(self, name, parent):
                self.name = name
                self.parent = parent
                self.kids = {}
                self.is_term = False
                self.count = 0

            def add_word(self, word):
                self._add_child(word, 0)

            def _add_child(self, word, pos):
                self.count += 1
                if pos >= len(word):
                    self.is_term = True
                    return
                ch = word[pos]
                if ch not in self.kids:
                    self.kids[ch] = TrieNode(ch, self)
                self.kids[ch]._add_child(word, pos + 1)

            def is_valid(self, word):
                return self._is_valid(word, 0)

            def _is_valid(self, word, pos):
                if pos == len(word):
                    return self.is_term
                ch = word[pos]
                if ch not in self.kids:
                    return False
                return self.kids[ch]._is_valid(word, pos + 1)

            def get_path(self):
                txt = ""
                n = self
                while n:
                    txt = n.name + txt
                    n = n.parent
                return txt

            def __repr__(self):
                return self.get_path() + ('*' if self.is_term else '')

        def load_set(f_p):
            with open(f_p) as f:
```

```
        return set([line.strip() for line in f])

def write_set(items, f_p):
    with open(f_p, 'w') as f:
        f.write('\n'.join(items))
```

2.2.2 Loading the dictionary

```
In [4]: %%time

        #url = "http://www-01.sil.org/linguistics/wordlists/english/wordlist/wordsEn.txt"
        #url = "https://raw.githubusercontent.com/first20hours/google-10000-english/master/20k
        #en_dict = set(requests.get(url).text.split())
        # The above lists include too many words that the search is intractable
        # especially because of many 2 character short words

        # so I have removed unnecessary short words such as 'fi', 'ab', and stored in this file
        dict_file = "data/wordsEn-filtered.txt"
        en_dict = load_set(dict_file)
        print("%d words" % len(en_dict))

        root = TrieNode("/", None)
        for w in en_dict:
            root.add_word(w)
        print(len(en_dict), root.count)
```

```
109467 words
109467 109467
CPU times: user 1.91 s, sys: 80 ms, total: 1.99 s
Wall time: 1.99 s
```

3 Preparation

3.1 Simple problem at first

Before jumping to decrypt the large input message with a large (110K) dictionary, I formulated a simple problem and a smaller dictionary. The purpose is to prove that the algorithm is correct by manually verifying the solutions.

```
In [6]: # smaller version of this problem to test the correctness

        # lets take a simple message with keylength = 11

        msg = ["iaminoffice","whereareyou", "shallisleep"]
        sml_T_orig = np.array(list(map(list, msg))) # table

        np.random.seed(10)  # to reproduce same shuffle for reporting
        key = np.random.permutation(sml_T_orig.shape[1])
        print("Key:", key)
        sml_T = sml_T_orig[:, key]

        print(sml_T.shape)
        print(sml_T_orig)
        print(sml_T)

        sml_en_dict = ["i", "am", "in", "office", "where",
                       "are", "you", "shall", "sleep", "write"]
        sml_root = TrieNode("/", None)
        for w in sml_en_dict:
            sml_root.add_word(w)
        print(len(sml_en_dict), sml_root.count)
```

```
Key: [ 2  6  8  5  7 10  3  1  0  4  9]
(3, 11)
[['i' 'a' 'm' 'i' 'n' 'o' 'f' 'f' 'i' 'c' 'e']
 ['w' 'h' 'e' 'r' 'e' 'a' 'r' 'e' 'y' 'o' 'u']
 ['s' 'h' 'a' 'l' 'l' 'i' 's' 'l' 'e' 'e' 'p']]
[['m' 'f' 'i' 'o' 'f' 'e' 'i' 'a' 'i' 'n' 'c']
 ['e' 'r' 'y' 'a' 'e' 'u' 'r' 'h' 'w' 'e' 'o']
 ['a' 's' 'e' 'i' 'l' 'p' 'l' 'h' 's' 'l' 'e']]
10 10
```

3.2 Text Segmentation using Greedy technique

Trusting my hypothesis that *greedy segmentation combined with a valid dictionary and reasonable language model may yield valid result*, I created a text segmentation function.

The greediness here is to fit the longest span words/segments. This segmentation function uses dictionary in the trie to quickly lookup valid segments. The effect of language-model / 10-rows-batch isnt evident at this step. Later we can see that if this segmenter fails for any one row, then the split is considered invalid

```
In [7]: def greedy_split(root, txt, node, tails, pos=0):
            #print(pos, txt, node, tails)
            if len(txt) == 0 or pos == len(txt):
                tails.append(node)
                return True
            ch = txt[pos]
            if ch in node.kids:
                # greedy -- try to maximize the current word lengh
                if greedy_split(root, txt, node.kids[ch], tails, pos+1):
                    return True
            if node.is_term and ch in root.kids:
                tails.append(node)
                if greedy_split(root, txt, root.kids[ch], tails, pos+1):
```

```
            return True
        tails.pop()
    return False
```

3.3 Test the greedy splitter

Here I test the splitter on the given message and manually verify if it is a reasonable split (it may not be valid a english sentence since language model is not incorporated at this step, but it should be yielding correct as per grreedy assumptions.)

```
In [8]: %%time
        for txt in ref_msg_orig:
            tails = []
            res = greedy_split(root, txt, root, tails)
            print(res, tails)
```

```
True [/the*, /panel*, /is*, /also*, /expected*, /tore*, /commend*, /that*, /the*, /white*, /hou
True [/their*, /an*, /i*, /an*, /government*, /has*, /maintained*, /that*, /it*, /knows*, /noth
True [/by*, /matching*, /the*, /lowest*, /price*, /and*, /enhancing*, /service*, /hew*, /as*, /
True [/this*, /fall*, /her*, /neighborhood*, /int*, /hen*, /orth*, /eastern*, /part*, /oft*, /l
True [/for*, /college*, /basketball*, /fans*, /this*, /is*, /as*, /good*, /a*, /week*, /as*, /y
True [/but*, /according*, /to*, /barbershop*, /proprietors*, /then*, /umber*, /off*, /ema]
True [/an*, /intriguing*, /new*, /study*, /suggests*, /that*, /what*, /really*, /draws*, /peo]
True [/also*, /on*, /thursday*, /the*, /latest*, /winners*, /oft*, /he*, /life*, /sciences*, /a
True [/what*, /members*, /of*, /both*, /parties*, /bemoaned*, /more*, /than*, /anything*, /wa]
True [/one*, /key*, /hurdle*, /fortes*, /lain*, /producing*, /the*, /new*, /smaller*, /car*, /w
CPU times: user 0 ns, sys: 4 ms, total: 4 ms
Wall time: 3.7 ms
```

4 The Column Transpose Decipher Algorithm

This is a recursive algorithm to decrypt column transpose cipher

Input: >T - a matrix of R x C where R = 10 and C=50
root - a pointer to the root of Trie containing dictionary

Output:
>key - sequence of columns - can be used for decrypting
words - list of lists - each list contains word tails of rows. It can be used to print the state of pointers

State: >nR - max rows allowed in the input, this could be 10
nC - max columns allowed as per input, this could be 50
root - pointer to root of the dictionary trie
words - List of words, one list per row, that are seen to reach to the current state
txts - List of texts, one per row, that are constructed while reaching to the current state
seq - sequence of columns used to reach to the current state
used - set of column indices that are currenlty taken or used for reaching to current state

** Algorithm** >1. Begin with clean state
a. `words` for each row have single pointer pointing to the root of tries b. `used` is an empty set c. `seq` is an empty sequence d. `txts` for each row is empty string 2. Base case:

a. if len(used) == nC, it is a solution. Yield it
b. The problem statement doesnt enforce the ending words to be valid words so we dont need to check it 3. Recursive case: a. enumerate all columns that may be next. This is all columns - `used`
b. for each possible next column check if it could be a valid next column else continue to next column
c. if the possible next column is valid (as per the dictionary, it yields valid words)
i. add this column to `used`, `seq` and update the trie pointers in `words`
ii. recursively call the same algorithm
iii. remove column from `used`, `seq` and revert the trie pointers in `words`

The detailed algorith is below as a python (executable pseudo code)

```
In [9]: class Decipher(object):

            def __init__(self, root):
                self.root = root

            def _make_state(self, T):
                print("Size:", T.shape)
                return State(self.root, *T.shape)

            def decipher(self, T, print_sol=False):
                st = self._make_state(T)
                count = 0
                if print_sol:
                    print("Input:")
                    print('\n'.join(map(lambda x: ''.join(x), T)))
                for key, words in self._decipher(T, st):
                    print(count + 1, "Key:", key)
                    if print_sol:
                        decrypted = T[:, key]
                        print('\n'.join(map(lambda x: ''.join(x), decrypted)))
                        print("Greey Split:\n", '\n'.join(
                            map(lambda x: ''.join(map(lambda n: str(n), x)), words)))
                    count += 1
                print("Found %d candidate solutions" % count)

            def _decipher(self, T, st):
                if st.is_terminal(): # Solution
                    yield copy(st.seq), copy(st.words)
                else:
                    for col in st.get_next_cols():
                        nxt = T[:, col]
                        if st.is_valid_next(nxt):
                            st.move_next(col, nxt)
                            yield from self._decipher(T, st)
                            st.move_back(col)

        class State(object):
```

```
    '''
    An object of this class stores state of search.
    '''

    def __init__(self, root, nR, nC):
        self.root = root
        self.nR = nR
        self.nC = nC
        self.words = [[self.root] for i in range(nR)]
        self.txts = ['' for i in range(nR)]
        self.seq = []
        self.used = set()

    def is_terminal(self):
        '''
        true if this is a terminal state
        '''
        return self.nC == len(self.used)

    def get_next_cols(self):
        '''
        List of columns that can be next
        '''
        #TODO: Rank columns
        return filter(lambda x: x not in self.used,range(self.nC))

    def is_valid_next(self, nxt):
        assert len(nxt) == self.nR
        for i in range(self.nR):
            ch = nxt[i]
            if ch in self.words[i][-1].kids: # word continuation
                continue
            if self.words[i][-1].is_term and ch in self.root.kids: # new word
                continue
            if greedy_split(self.root, self.txts[i] + ch, self.root, []):
                # re adjust word boundaries and fit
                continue
            return False # None of the above cases ==> not possible
        # all rows are ok
        return True

    def move_next(self, col, nxt):
        assert len(nxt) == self.nR
        self.used.add(col)
        self.seq.append(col)
        # update pointers
        for i in range(self.nR):
            ch = nxt[i]
            self.txts[i] = self.txts[i] + ch
            if ch in self.words[i][-1].kids: # continuation of word
                self.words[i][-1] = self.words[i][-1].kids[ch]
            elif self.words[i][-1].is_term and ch in self.root.kids: # new word
                self.words[i].append(self.root.kids[ch])
            else:
                self.words[i] = []
                assert greedy_split(self.root, self.txts[i], self.root, self.words[i])

    def move_back(self, col):
        self.used.remove(col)
        assert col == self.seq.pop()
        # update pointers
        for i in range(self.nR):
            self.txts[i] = self.txts[i][:-1]
            self.words[i][-1] = self.words[i][-1].parent
            if self.words[i][-1].parent is None:
                if self.used:
                    self.words[i].pop()
```

4.0.1 Simple case - smaller message, smaller dictionary

Goal: Test on correctness

```
In [10]: %%time
         print(sml_root.count)
         dec = Decipher(sml_root)
         dec.decipher(sml_T, print_sol=True)
```

```
10
Size: (3, 11)
Input:
mfiofeiainc
eryaeurhweo
aseilplhsle
1 Key: [3, 1, 4, 2, 10, 5, 8, 7, 0, 6, 9]
officeiamin
areyouwhere
isleepshall
Greey Split:
 /office*/i*/am*/in*
/are*/you*/where*
/i*/sleep*/shall*
2 Key: [8, 7, 0, 6, 9, 3, 1, 4, 2, 10, 5]
iaminoffice
whereareyou
shallisleep
Greey Split:
 /i*/am*/in*/office*
/where*/are*/you*
/shall*/i*/sleep*
Found 2 candidate solutions
CPU times: user 4 ms, sys: 4 ms, total: 8 ms
Wall time: 7.24 ms
```

We see that

1. dictionary had 10 words 2. It took 4ms (search, format and print) 3. Input size (3, 11)

For input:

```
camofiieinf
ohearwyuree
ehaissepl ll
```

There are two candidate solutions:

1. Key: [3, 1, 4, 2, 10, 5, 8, 7, 0, 6, 9]

```
/office*/i*/am*/in*
/are*/you*/where*
/i*/sleep*/shall*
```

2. Key: [8, 7, 0, 6, 9, 3, 1, 4, 2, 10, 5]

```
/i*/am*/in*/office*
/where*/are*/you*
/shall*/i*/sleep*
```

The second one is correct. (TODO: use language model with bigrams and trigrams to rank/score these candidates)

4.0.2 Simpler case - Smaller message and larger dictionary

Here we decrypt the same message with a larger dictionary:

```
In [11]: %%time
         print("Dictionary size:", root.count)
         dec = Decipher(root)
         dec.decipher(sml_T, print_sol=False)
```

```
Dictionary size: 109467
Size: (3, 11)
1 Key: [0, 2, 4, 1, 10, 8, 7, 3, 6, 9, 5]
2 Key: [0, 2, 9, 6, 1, 7, 10, 3, 8, 4, 5]
3 Key: [0, 2, 9, 6, 4, 3, 5, 1, 7, 10, 8]
4 Key: [0, 2, 9, 6, 8, 4, 10, 3, 1, 5, 7]
5 Key: [0, 2, 9, 6, 8, 7, 10, 3, 1, 4, 5]
6 Key: [0, 2, 9, 6, 8, 7, 10, 3, 4, 1, 5]
```

```
343 Key: [8, 4, 2, 10, 5, 7, 0, 3, 9, 1, 6]
344 Key: [8, 4, 2, 10, 5, 7, 0, 6, 9, 1, 3]
345 Key: [8, 4, 2, 10, 5, 7, 0, 6, 9, 3, 1]
346 Key: [8, 4, 2, 10, 5, 7, 3, 1, 0, 6, 9]
347 Key: [8, 4, 2, 10, 5, 7, 3, 1, 9, 6, 0]
348 Key: [8, 7, 0, 4, 2, 9, 3, 1, 6, 10, 5]
349 Key: [8, 7, 0, 6, 4, 3, 1, 2, 9, 10, 5]
350 Key: [8, 7, 0, 6, 4, 3, 1, 5, 9, 10, 2]
351 Key: [8, 7, 0, 6, 4, 3, 1, 9, 2, 10, 5]
352 Key: [8, 7, 0, 6, 9, 3, 1, 2, 4, 10, 5]
353 Key: [8, 7, 0, 6, 9, 3, 1, 4, 2, 10, 5]
354 Key: [8, 7, 0, 6, 9, 10, 2, 4, 1, 3, 5]
355 Key: [8, 7, 2, 0, 6, 4, 1, 5, 9, 3, 10]
356 Key: [8, 7, 2, 0, 6, 4, 1, 5, 9, 10, 3]
357 Key: [8, 7, 2, 0, 6, 4, 3, 1, 5, 9, 10]
358 Key: [8, 7, 2, 0, 6, 9, 3, 1, 4, 10, 5]
359 Key: [8, 7, 2, 1, 5, 0, 3, 4, 6, 9, 10]
360 Key: [8, 7, 2, 1, 5, 0, 3, 9, 6, 4, 10]
361 Key: [8, 7, 2, 1, 5, 0, 4, 6, 3, 9, 10]
362 Key: [8, 7, 2, 1, 5, 0, 4, 6, 9, 3, 10]
363 Key: [8, 7, 2, 1, 5, 0, 4, 6, 9, 10, 3]
364 Key: [8, 7, 2, 1, 5, 0, 9, 3, 6, 4, 10]
365 Key: [8, 7, 2, 3, 1, 5, 0, 4, 6, 9, 10]
366 Key: [8, 7, 2, 4, 3, 1, 6, 10, 5, 0, 9]
367 Key: [8, 7, 2, 4, 6, 1, 9, 3, 5, 0, 10]
368 Key: [8, 7, 2, 4, 6, 3, 1, 5, 9, 10, 0]
369 Key: [8, 7, 2, 4, 6, 9, 3, 1, 5, 0, 10]
370 Key: [8, 7, 2, 4, 6, 9, 10, 3, 1, 0, 5]
371 Key: [8, 7, 2, 9, 3, 1, 6, 10, 5, 0, 4]
372 Key: [8, 7, 2, 9, 6, 0, 3, 1, 4, 10, 5]
373 Key: [8, 7, 2, 9, 6, 1, 4, 3, 5, 0, 10]
374 Key: [8, 7, 2, 9, 6, 1, 4, 3, 5, 10, 0]
375 Key: [8, 7, 2, 9, 6, 3, 1, 5, 0, 4, 10]
376 Key: [8, 7, 2, 9, 6, 4, 3, 1, 5, 0, 10]
377 Key: [8, 7, 3, 4, 6, 1, 0, 2, 9, 10, 5]
378 Key: [8, 7, 3, 4, 6, 1, 2, 10, 5, 0, 9]
379 Key: [8, 7, 3, 4, 6, 1, 9, 2, 0, 10, 5]
380 Key: [8, 7, 3, 4, 6, 9, 2, 10, 5, 0, 1]
381 Key: [8, 7, 3, 9, 6, 1, 2, 10, 5, 0, 4]
382 Key: [8, 7, 3, 9, 6, 4, 2, 10, 5, 0, 1]
383 Key: [8, 7, 4, 2, 5, 1, 0, 3, 9, 6, 10]
384 Key: [8, 7, 10, 3, 0, 1, 2, 9, 6, 4, 5]
385 Key: [8, 7, 10, 3, 0, 2, 4, 6, 1, 5, 9]
386 Key: [8, 7, 10, 3, 0, 2, 4, 6, 1, 9, 5]
387 Key: [8, 7, 10, 3, 0, 2, 9, 6, 1, 4, 5]
388 Key: [8, 7, 10, 3, 0, 2, 9, 6, 1, 5, 4]
389 Key: [8, 7, 10, 3, 0, 6, 4, 1, 5, 9, 2]
390 Key: [8, 7, 10, 3, 2, 4, 6, 5, 0, 1, 9]
```

```
391 Key: [8, 7, 10, 3, 2, 4, 9, 6, 0, 5, 1]
392 Key: [8, 7, 10, 3, 2, 9, 6, 5, 0, 1, 4]
393 Key: [8, 7, 10, 3, 2, 9, 6, 5, 0, 4, 1]
394 Key: [8, 7, 10, 3, 6, 4, 1, 2, 9, 0, 5]
395 Key: [8, 7, 10, 3, 6, 4, 1, 5, 0, 2, 9]
396 Key: [8, 7, 10, 3, 6, 4, 1, 5, 0, 9, 2]
397 Key: [8, 7, 10, 3, 6, 4, 1, 5, 9, 2, 0]
398 Key: [8, 7, 10, 3, 6, 4, 9, 2, 0, 1, 5]
399 Key: [8, 7, 10, 3, 6, 4, 9, 2, 0, 5, 1]
400 Key: [8, 7, 10, 3, 6, 9, 1, 5, 0, 2, 4]
401 Key: [8, 7, 10, 3, 6, 9, 1, 5, 0, 4, 2]
402 Key: [8, 7, 10, 3, 9, 4, 6, 2, 0, 5, 1]
403 Key: [8, 7, 10, 3, 9, 6, 2, 1, 5, 0, 4]
404 Key: [8, 7, 10, 3, 9, 6, 4, 0, 5, 1, 2]
405 Key: [8, 9, 2, 10, 5, 0, 6, 4, 1, 3, 7]
406 Key: [8, 9, 2, 10, 5, 0, 6, 4, 1, 7, 3]
407 Key: [8, 9, 2, 10, 5, 1, 3, 7, 0, 4, 6]
408 Key: [8, 9, 2, 10, 5, 1, 3, 7, 0, 6, 4]
409 Key: [8, 9, 2, 10, 5, 3, 4, 6, 0, 7, 1]
410 Key: [8, 9, 2, 10, 5, 3, 4, 6, 1, 7, 0]
411 Key: [8, 9, 2, 10, 5, 7, 0, 3, 4, 1, 6]
412 Key: [8, 9, 2, 10, 5, 7, 0, 6, 4, 1, 3]
413 Key: [8, 9, 2, 10, 5, 7, 0, 6, 4, 3, 1]
414 Key: [8, 9, 2, 10, 5, 7, 3, 1, 0, 6, 4]
415 Key: [8, 9, 2, 10, 5, 7, 3, 1, 4, 0, 6]
416 Key: [8, 9, 2, 10, 5, 7, 3, 1, 4, 6, 0]
417 Key: [9, 2, 0, 1, 7, 10, 6, 5, 4, 3, 8]
418 Key: [9, 2, 0, 3, 4, 6, 1, 7, 10, 8, 5]
419 Key: [9, 2, 0, 3, 4, 6, 1, 8, 7, 10, 5]
420 Key: [9, 2, 0, 6, 1, 7, 10, 3, 8, 4, 5]
421 Key: [9, 2, 0, 6, 4, 3, 5, 1, 7, 10, 8]
422 Key: [9, 2, 0, 6, 8, 7, 10, 3, 1, 4, 5]
423 Key: [9, 2, 0, 8, 3, 1, 7, 10, 5, 6, 4]
424 Key: [9, 2, 0, 8, 7, 3, 4, 6, 1, 10, 5]
425 Key: [9, 2, 0, 8, 7, 10, 3, 6, 4, 1, 5]
426 Key: [9, 2, 0, 8, 7, 10, 3, 6, 4, 5, 1]
427 Key: [9, 3, 5, 0, 1, 6, 2, 10, 8, 7, 4]
428 Key: [9, 3, 5, 0, 4, 6, 2, 1, 10, 8, 7]
429 Key: [10, 6, 7, 3, 2, 1, 5, 0, 8, 4, 9]
430 Key: [10, 6, 7, 3, 2, 1, 5, 0, 8, 9, 4]
Found 430 candidate solutions
CPU times: user 780 ms, sys: 44 ms, total: 824 ms
Wall time: 813 ms
```

4.0.3 Observations:

1. Dictionary had ~110k words

2. time taken = 1.06 seconds
3. Found 430 candidate solutions, candidate 353 is the key

```
353 Key: [8, 7, 0, 6, 9, 3, 1, 4, 2, 10, 5]
```

TODO: 1. rank candidates based on bigram language model -- may be not needed for 10 × 50 eval message

4.1 Large message with larger dictionary

Let us try to decrypt the reference message

```
In [12]: %%time
         dec = Decipher(root)
         dec.decipher(devT, print_sol=True)
```

```
Size: (10, 50)
Input:
heeemirhcletshlohttwhsnpuesecetottipoenadeamtoaexd
tnreaaahninsnootottnhiaihkvnnhimttwmtieadnrisegena
raomnnaytihhgchawbsindcasveedreeipeeenngcethcltswi
rrginhnhrllteantfttpaiashneshoeiehrbettshofodiroho
eaerigtolelyeaarufakoglceesoifwsolsbshsdgoosnkbata
nebtrdpueroribgffbhmsmcaeutaoseprrerprttoocioonbah
aawituendgiriygswarlaernplnotseghydtssttwathgenusu
eotsruilahtnridseaicennoasadnefwhteeenfltcostyslhs
aeoaaebhabmtrnoniwtarwetgnfamthsoryhoedemhmnebspti
mgreironluhrdleraowlniykwafluisreaeecdtehcenpolstn
1 Key: [17, 7, 3, 23, 42, 22, 10, 9, 5, 12, 46, 14, 26, 45, 2, 48, 35, 47, 8, 33, 29, 49, 44, 3
thepanelisalsoexpectedtorecommendthatthewhitehouse
theiraniangovernmenthasmaintainedthatitknowsnothin
bymatchingthelowestpriceandenhancingservicehewasde
thisfallherneighborhoodinthenortheasternpartofthis
forcollegebasketballfansthisisasgoodaweekasyourego
butaccordingtobarbershopproprietorsthenumberoffema
anintriguingnewstudysuggeststhatwhatreallydrawspeo
alsoonthursdaythelatestwinnersofthelifesciencesand
whatmembersofbothpartiesbemoanedmorethananythingwa
onekeyhurdleforteslainproducingthenewsmallercarwil
Greey Split:
 /the*/panel*/is*/also*/expected*/tore*/commend*/that*/the*/white*/house*
/their*/an*/i*/an*/govern*/me*/nth*/as*/maintained*/that*/it*/knows*/nothin
/by*/matching*/the*/lowest*/price*/and*/enhancing*/service*/hew*/as*/de
/this*/fall*/her*/neighborhood*/int*/hen*/orth*/eastern*/part*/oft*/his*
/for*/college*/basketball*/fans*/this*/is*/as*/good*/a*/week*/as*/your*/ego*
/but*/according*/to*/barbershop*/proprietors*/then*/umber*/off*/ema
/an*/intriguing*/new*/study*/suggests*/that*/what*/really*/draws*/peo
/also*/on*/thursday*/the*/latest*/winners*/oft*/he*/life*/sciences*/and*
/what*/members*/of*/both*/parties*/bemoaned*/more*/than*/any*/thing*/wa
```

```
/one*/key*/hurdle*/fortes*/lain*/producing*/the*/new*/smaller*/car*/wil
Found 1 candidate solutions
CPU times: user 880 ms, sys: 4 ms, total: 884 ms
Wall time: 894 ms
```

Observation
1. It found the key in less than 1second!

```
[17, 7, 3, 23, 42, 22, 10, 9, 5, 12, 46, 14, 26, 45, 2, 48, 35, 47, 8,
33, 29, 49, 44, 31, 6, 37, 28, 36, 4, 43, 1, 38, 40, 32, 20, 39, 18, 30,
0, 25, 19, 13, 34, 11, 41, 16, 15, 24, 21, 27]
```

2. Greedy text segmentation has made few mistakes

4.2 Decrypting the evaluation message:

```
In [13]: %%time
         dec = Decipher(root)
         print("On test message")
         dec.decipher(evalT)
```

```
On test message
Size: (10, 50)
Found 0 candidate solutions
CPU times: user 656 ms, sys: 4 ms, total: 660 ms
Wall time: 663 ms
```

Found 0 Candidates!
Possible reasons: Dictionary is missing one or more words that are necessary to construct the evaluation message
Solutions: 1. Get a better dictionary --> I do not know which domain this message is from! 2. Modify search to account for missing words in dictionary

Option 2 sounds like a reasonable plan and fun exercise

Going to work with uncertainities using probabilities.
PState - Probabilistic State
PDecipher - Probabilistic Decipher

```
In [14]: PROB = 2.0 / 3 # probability of correctness

         class PState(State):

             def is_valid_next(self, nxt):
                 assert len(nxt) == self.nR
                 un_fit = 0
                 for i in range(self.nR):
```

```
            ch = nxt[i]
            if ch in self.words[i][-1].kids: # word continuation
                continue
            if self.words[i][-1].is_term and ch in self.root.kids: # new word
                continue
            if greedy_split(self.root, self.txts[i] + ch, self.root, []):
                # re adjust word boundaries and fit
                continue
            # no fit found for this row
            un_fit += 1

        return (1.0 - float(un_fit) / self.nR) > PROB

class PDecipher(Decipher):

    def _make_state(self, T):
        print("Size:", T.shape)
        return PState(self.root, *T.shape)


dec = PDecipher(sml_root)
dec.decipher(sml_T)
```

```
Size: (3, 11)
```

```
    ---------------------------------------------------------------------------

    AssertionError                            Traceback (most recent call last)

    <ipython-input-14-54c9036d0830> in <module>()
     27
     28 dec = PDecipher(sml_root)
---> 29 dec.decipher(sml_T)


    <ipython-input-9-97b63858d804> in decipher(self, T, print_sol)
     14             print("Input:")
     15             print('\n'.join(map(lambda x: ''.join(x), T)))
---> 16         for key, words in self._decipher(T, st):
     17             print(count + 1, "Key:", key)
     18             if print_sol:


    <ipython-input-9-97b63858d804> in _decipher(self, T, st)
     30                 nxt = T[:, col]
     31                 if st.is_valid_next(nxt):
```

```
---> 32                         st.move_next(col, nxt)
     33                         yield from self._decipher(T, st)
     34                         st.move_back(col)


    <ipython-input-9-97b63858d804> in move_next(self, col, nxt)
     89             else:
     90                 self.words[i] = []
---> 91                 assert greedy_split(self.root, self.txts[i], self.root, self.words


    AssertionError:
```

AssertionError:
`assert greedy_split(self.root, self.txts[i], self.root, self.words[i])`

This requires me account for probabilities in state updates and backtracking --> that is going to be messy! From external point of view, bringing down probability of match from 1.00 to a lower value skips some rows. Let us accomplish the same by preprocessing the input. i.e., skip a row and try to find solution!

```
In [16]: %%time
         T = evalT
         nR, nC = T.shape
         dec = Decipher(root)
         for i in range(nR):
             print("i=%d" % i)
             newT = np.delete(T, (i), axis=0) # skip ith row
             dec.decipher(newT)
```

```
i=0
Size: (9, 50)
Found 0 candidate solutions
i=1
Size: (9, 50)
Found 0 candidate solutions
i=2
Size: (9, 50)
Found 0 candidate solutions
i=3
Size: (9, 50)
Found 0 candidate solutions
i=4
Size: (9, 50)
Found 0 candidate solutions
i=5
```

```
Size: (9, 50)
1 Key: [31, 32, 24, 30, 21, 9, 38, 13, 41, 11, 46, 19, 36, 3, 28, 20, 29, 45, 5, 8, 27, 42, 6,
Found 1 candidate solutions
i=6
Size: (9, 50)
Found 0 candidate solutions
i=7
Size: (9, 50)
Found 0 candidate solutions
i=8
Size: (9, 50)
Found 0 candidate solutions
i=9
Size: (9, 50)
Found 0 candidate solutions
CPU times: user 50.7 s, sys: 44 ms, total: 50.8 s
Wall time: 50.8 s
```

```
In [17]: # Hurray! we found a solution
         key = [31, 32, 24, 30, 21, 9, 38, 13, 41, 11, 46, 19, 36, 3,
                28, 20, 29, 45, 5, 8, 27, 42, 6, 10, 4, 17, 35, 22, 26,
                49, 0, 14, 16, 18, 2, 37, 33, 40, 48, 34, 25, 47, 43, 23,
                1, 12, 39, 15, 44, 7]
         decrypted = evalT[:, key]
         print('\n'.join(map(lambda x: ''.join(x), decrypted)))

         msg = list(map(lambda x: ''.join(x), decrypted))
         for txt in msg:
             tails = []
             greedy_split(root, txt, root, tails)
             print(tails)
```

```
theremustbesomewayoutofheresaidthejokertothethieft
toomuchconfusionicantgetnoreliefbusinessmentheydri
mywineplowmendigmyearthnoneofthemalongthelineknoww
anyofitisworthnoreasontogetexcitedthethiefhekindly
spoketherearemanyhereamonguswhofeelthatlifeisbutaj
butyouandiwevebeenthroughthatandthisisnotourfateso
letusnottalkfalselynowthehourisgettinglateallalong
thewatchtowerprinceskepttheviewwhileallthewomencam
andwentbarefootservantstoooutsideinthedistanceawil
catdidgrowltworiderswereapproachingthewindbegantoh
[/there*, /must*, /be*, /someway*, /out*, /of*, /heres*, /aid*, /the*, /joker*, /tot*, /he*, /
[/too*, /much*, /confusion*, /i*, /cant*, /get*, /no*, /relief*, /businessmen*, /they*, /dri]
[/my*, /wine*, /plowmen*, /dig*, /my*, /earth*, /none*, /oft*, /hem*, /along*, /the*, /line*, /
[/any*, /of*, /it*, /is*, /worth*, /no*, /reason*, /to*, /get*, /excited*, /the*, /thief*, /he
[/spoke*, /there*, /are*, /many*, /here*, /among*, /us*, /who*, /feel*, /that*, /life*, /is*, /
```

```
[]
[/let*, /us*, /not*, /talk*, /falsely*, /now*, /the*, /houris*, /getting*, /late*, /all*, /alon
[/the*, /watchtower*, /princes*, /kept*, /the*, /view*, /while*, /all*, /thew*, /omen*, /cam*]
[/and*, /went*, /barefoot*, /servants*, /too*, /outside*, /int*, /he*, /distance*, /a*, /wil]
[/cat*, /did*, /growl*, /two*, /riders*, /were*, /approaching*, /the*, /wind*, /began*, /to*, /
```

Observations

1. One of the word in row 6 is missing in dictionary
2. Greedy split has made mistakes, however the transposition key is uneffected by it.

Manual split:

```
there must be some way out of here said the joker to the thief t
too much confusion i cant get no relief businessmen they dri
my wine plow men dig my earth none of them along the line know w
any of it is worth no reason to get excited the thief he kindly
spoke there are many here among us who feel that life is but aj
but you and i we ve been through that and this is not our fate so
let us not talk falsely now the houris getting late all along
the watch tower princes kept the view while all the women cam
and went bare foot servants too outside in the distance awil
cat did growl two riders were approaching the wind began to h
```

4.3 Scratchpad

These are some experiments on data exloration

```
In [ ]: # check if dictionary contains words
        s ="""
        the panel is also expected to recommend that the white house
        the iranian government has maintained that it knows nothin
        by matching the lowest price and enhancing service he was de
        this fall her neighborhood in the north eastern part of this
        for college basket ball fans this is as good a week as your ego
        but according to barbershop proprietors the number of fema
        an intriguing new study suggests that what really draws peo
        also on thursday the latest winners of the life sciences and
        what members of both parties bemoaned more than anything wa
        one key hurdle for tesla in producing the new smaller car wil"""
        words = set(s.split())
        '''
        root = TrieNode("/", None)
        for w in en_dict:
```

```
                 root.add_word(w)
             print(len(en_dict), root.count)
             print(en_dict)
             '''

             words - en_dict

    In [ ]: # filter words in dictionary to reduce search space
            # the original dictionary has too many small words such as 'ii', 'ab', 'ac', 'eh' etc
            # So filtering words based on http://ogden.basic-english.org/word2000.html
            all_toks = load_set('data/wordsEn.txt')
            basic_words = load_set("data/basic-words.txt")

            filtered = set()
            for tok in all_toks:
                if len(tok) > 2 or tok in basic_words:
                    filtered.add(tok)

            print(len(all_toks), len(filtered))

            write_set(filtered, 'data/wordsEn-filtered.txt')

    In [ ]: from collections import defaultdict

            words = defaultdict(int)
            for name in treebank.fileids():
                for i in treebank.words(name):
                    words[i] += 1

    In [ ]: all_toks = load_set('data/wordsEn.txt')
            subset  = load_set('data/wordsEn-filtered.txt')
            diff = all_toks - subset
            len(diff)
```