CS323-Artificial Intelligence

Text Compression

Course Project

Introduction

There is no such thing as universal compression, recursive compression, or compression of random data. So how can we compress a text file? The truth is we can't compress most files because most strings are random. But most meaningful strings are not. This means that although we can't compress 'all files', we can compress meaningful files!

Dictionary Compression

The basic inspiration behind compression was that many words are repeated throughout a text, and if somehow we can put those words in a list so that every time we encounter that word, we can just point to it. So, let's say the word "would" appears 10 times in a text, we can just put it in our reference-list (which is nothing but a list of most frequented words) and just point to the index of our reference-list that holds our word!

The word "would" takes up the space of 5 bytes, so a text containing the word "would" 10 times will take a space of 50 bytes. Now imagine if we store the word in the reference-list and whenever we encounter the word in the file, we just put the index of the word (2-byte integer), this way we only use 2*10 + 5 = 25 bytes!

From above we can see that the bigger the size of a word in question, the more memory we can save from it! Unfortunately, the English language doesn't have many large words. But why are we restricting ourselves to just words? If someone is writing an essay on graph theory, the phrases "cost of the path" is bound to appear several times! What if we can put the entire phrases into our reference-list and refer it by index?

Say the phrase "cost of the path" appears 5 times in the entire text, the actual space required will be 80 (16*5) bytes, whereas if we use our technique to store the word, we only use 2*5 + 16 = 26 bytes! This makes our algorithm much more effective!

The **focus of this project will be to find phrases** that will help up optimize the compression.

Problem

For a given file, we have to find the phrases that can be substituted in the file by a 2-byte integer specifying it's index in our reference-reference-list. We have to find phrases in such a way that they maximize the overall compression of the file. We'll solve this problem using heuristics, so it is a part of informed searching.

Background Survey

There are many ways to compress text files, but we are going to restrict ourselves to dictionary-based compression. Now, there are many people who have done a dictionary-based compression, where the dictionary was pre-designed using common English words/phrases or was made on the fly.[1]

Dictionary-based compression schemes are the most commonly used data compression schemes since they appeared in the foundational paper of Ziv and Lempel in 1977 and generally referred to as LZ77 [2]. Many compression software utilities such as Zip, gZip, 7-Zip etc use variants of the greedy approach to parse the text into dictionary phrases.

Our algorithm is also a variant of the greedy dictionary parsing with LZ77.

Discussion:

We are going to refer to the text of the file as T, We assume the number of characters in T (or file size in bytes) in N. We make a dictionary of phrases that we will now refer to as reference-list.

We are going to pose the compression as a graph search problem. The start state is the uncompressed file. The goal state is when the text can no longer be compressed. For each intermediate state.

We will use a sliding window technique to get all the phrases that can be substituted in the text by referring to it by the reference list. Since these phrases can interfere with each other we will only substitute one at a time. The future states will be when one of the phrases is added to the reference-list.

Say we have n number of phrases for the given string, we then have n future states where i^{th} state is when we substitute the string with the i^{th} phrases. Our goal state is when no such phrases exist!

We will look at how we can greedily replace phrases from text T to compress the file. We will **not be** using the greedy compression of LZ77, but we'll implement something similar **using search trees**! We will then move on to a **self-developed variant** of the greedy search known as GDFS. We will then improve GDFS using various techniques from search algorithms. These are the two things **novel** about our project: Adapting LZ77 as a graph search and developing a variant for the greedy search.

GDFS is an attempt at improving the efficiency of greedy approach and then we use several other techniques to make it more time-efficient.

Algorithm:

Brute Force Solution

For each state, we are finding the next possible states by finding the phrases that will reduce the size of text T. We find those phrases using a sliding window technique. We then get the next states by replacing the text in the state by phrases. We then repeat the process for each of state. The algorithm is mentioned below.

```
Algorithm 1: Brute Force Phrase Compression

def BrutePhraseCompression(text):
  maxSaved = 0

  for phrase in getPhrases(text):
    memorySaved = countOfPhrase(phrase) * (len(phrase) - 2) - len(phrase)

    if memorySaved > 0:
        newText = text.replace(phrase, "#")
        maxSaved = max(maxSaved, BrutePhraseCompression(newText))

    return maxSaved
```

This DFS approach to find the optimal solution will have a complexity of $O(n^n)$ time complexity, where n is the size of the text (in bytes). From this we can see that DFS is simply not feasible!

Greedy Solution

Instead of trying to find the optimal solution, we go with a “good enough” solution that can be calculated in real-time.

Heuristic Function

Our goal is to compress the file size, so an appropriate heuristic would be the size of the file at a given state! This means, for a given state, our successor function chooses the next state that has the minimum size. This heuristic is admissible as well as consistent*.
`getMemorySaved` is the heuristic function.

* Depends on implementation

Successor Function

The successor function greedily selects the file having the minimum size.

Algorithm

```
Algorithm 2:Greedy Compression

def greedyCompress(texts):
    memSaved = getMemorySaved(texts) # Returns map of memory saved for each phrase
    mx = 0
    bestPhrase = ""

    for phrase in memSaved:
        if(memSaved[phrase] > mx):
            mx = memSaved[phrase]
            bestPhrase = phrase

    if mx > 0:
        newTexts = []
        newTexts = text.replace(bestPhrase, "#")
        saved, phrases = greedyCompress(newTexts)

        return mx + saved, phrases + [bestPhrase]

    return 0, []
```

Challenges

To compress a file, we look for best future state among the N^2 phrases, we repeat this N time, giving the time complexity to compress a file as O(N^3), where N is the size of the file (in bytes). This means that compression won't be possible for files with size larger than 1 MB!

But, let's look at the phrases once, do we need to search for all possible phrases in a text? From the study of the English language, we can estimate that the maximum size of a phrase that repeats in a text is around 15 bytes. For example, the phrase " and the" is 8 bytes long, we don't really see the repetition of phrases that are longer than 15-20 characters. We can use this basic fact to restrict our search of phrases, If we only search for phrases that have length less than 20, we can reduce the overall complexity of the algorithm to O(N^2).

Results

I used greedy algorithm to compress the entire novel “To Kill a Mocking Bird”. Here are the results:

File Size: 166045 Bytes (16KB)
Compression Achieved: 35.7 %
Time taken to compress: 216.9 Seconds (3 Minutes and 36 Seconds)
Number of phrases in reference-list = 965. [List of phrases: at the end of the report]

Interesting observations: A few examples of the phrases in the reference-list : [“Mrs. Dubose”, “Uncle Jack”, “Aunt Alexandra”, “ the “, “ood”]
It is interesting to note:

1. How the algorithm automatically detected that it is better to replace the word “Mrs. Dubose” instead of “Mrs.” and “Dubose”.
2. How the algorithm included the spaces in “ the ”.
3. How the algorithm detected that “ood” is a common recurring phrase in English (like good, wood etc).

These phrases improve compression results.

GDFS Solution:

Finding phrases is an NP-hard problem. DFS was optimal, but not feasible. Greedy was feasible but not optimal. Let’s look at an approach where we can combine the two things to get better results than a simple greedy search with significantly less time than DFS.

There is something interesting with phrases. A text can contain both “because there” and “there is”. Now compressing one before the other can lead to reduction in the overall extent of compression! Greedily selecting one may not lead to optimal results. In GDFS, we will select 20 best states (according to the same heuristic) and expand them in a DFS-like fashion. This will take us closer to optimality.

Successor Function

The successor function selects the best 20 states (according to the same heuristic as above) and expands those 20 states in a DFS like fashion.

Algorithm

Algorithm 3:G-DFS Compression

```
def gdfsCompression(text):
    memSavers = {}
    for phrase in getPhrases(text):
        memSaved = countOfPhrase(phrase) * (len(phrase) - 2) - len(phrase)
        if memSaved > 0:
            memSavers[phase] = memorySaved
    topSavers = sort(memorySavers)[0:20]

    maxSaved = 0
    maxPhrases = []

    for phrase in topSavers:
      memSaved, memPhrases = gdfsCompression(text.replace(phrase, "#"))

      if memSaved>maxSaved:
          maxSaved = memSaved
          maxPhrases = memPhrases + [phrase]

    return maxSaved, maxPhrases
```

Challenges

G-DFS now has a complexity of O(N^{20})! This again is too large to be practical. We need something more practical.

GDFS with "Simulated Annealing" (GDFSA)

In GDFS, we talked about how phrases are often intersecting and selecting a particular phrase before other can lead to change in the overall extent of compression. However, there is another interesting thing to note here: As we go deeper into the tree, the length of the phrases decreases. Now the phrases are not multi-worded, a phrase is now just a word or a substring of a word! Since we no longer face intersecting phrases, ordering of phrases replaced won't affect much and we can resort back to greedy search.

So, In G-DFS with simulated annealing, we can reduce the branching factor as we go down the tree. One way is to reduce the branching factor by 1 (until it is 1) with every step down the tree.

Complexity

If we have an initial branching factor of m, with the branching factor decreasing by 1, we get a complexity of O(m! * n^2). This is quite practical for large files. GDFSA with initial branching factor m is called GDFSA - m.

Results:

I used the GDFSA -5 algorithm (GDFS with simulated annealing and initial branch factor 5) to compress the entire novel “To Kill a Mockingbird”. Here are the results:

File Size: 166045 Bytes (16KB)
Compression Achieved: 36.59 %
Time taken to compress: 5168.35 Seconds (86 Minutes and 8.3 Seconds)
Number of phrases in reference-list = 969. [List of phrases: at the end of the report]

Note that the time taken to compress is 5! times the time taken for greedy compression. (as predicted)

Summary and Conclusions:

To Sum up: We adapted LZ77 to a graph search so as to apply a greedy algorithm using a heuristic giving the file size. We then used a self devised algorithm that combines the quickness of greedy search along with the property of alternative search of DFS.

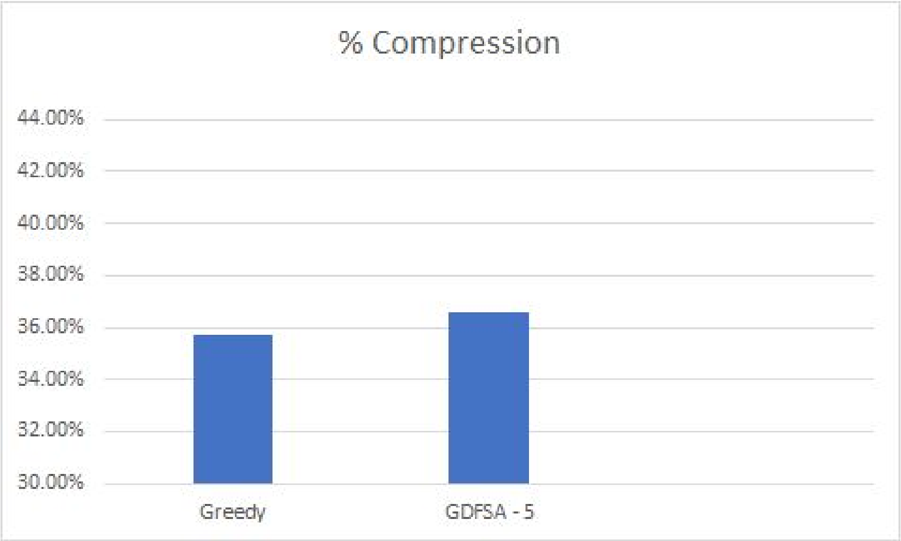
We also observed that the compression algorithm will **work better for larger files**. Also to avoid the time consuming process of finding the phrases, we can have a **predefined reference-lists.**

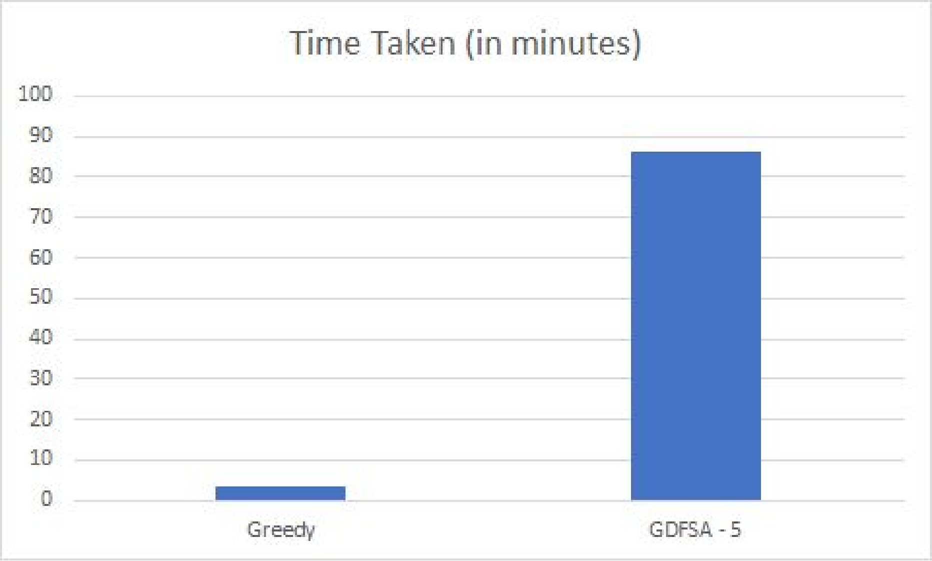
We exploited several constraints of the language such as:

1. The length of phrases that repeat in texts is often less.
2. There are less conflicts among the phrases as we go deeper into the tree.

Using these constraints, we significantly reduced the complexity of our algorithms. We looked for phrases with the help of sliding window, we varied the length of sliding window from size 2 to 20 (because of the first constraint).
Using the second constraint, we can reduce the branching factor for GDFS as we go deeper. GDFS with simulated annealing starts as DFS but becomes Greedy as we go deeper into the tree.

Here is a comparison of the performance of the two algorithms:





GDFSA - 5 is Greedy DFS with Simulated Annealing and branching factor 5

References:

[1]: Constructing Word-Based Text Compression Algorithms

[2]: On parsing optimality for dictionary-based text compression—the Zip case

Other References:

1. Data Compression Explained by Matt Mahoney

Phrase List with GDFSA - 5 on “To Kill a Mocking Bird”:

```
[' the ', ' was ', ' to ', ' and ', 'Atticus', '     ', '.  ', ' that ', ' you', ' said', 'ing ', 'ed ', ' th', "n't ", ' he', ' were ',
'ould ', ' had ', 'er ', 'Aunt Alexandra', ' of ', 'Mr. ', 'his ', 'Miss Maudie', ' with', 'ight', "'s ", ' in', 'Judge
Taylor', 'ing', ' did', 'Tom Robinson', 'Calpurnia', ' a ', ' wh', 'about', 'Mayella', ' for', ' him', ' some', ' I ',
'."\n', ' she', ', but', 'll ', 'Jem', 'ough', 'ere', 'Radley', ' it', 'because', '?"\n"', ' like', ' to', ' do', 'Miss Caroline',
'round', 'have', ' we', ' me', 'out', 'Finch', ' an', 'house', 'st ', 'all', 'one', ' not', 'he ', 'know', ' from', 'Mrs.
Dubose', 'children', 'look', 'at happen', 'ver', ', "', 'en ', 'hat', 'school', 'understand', '.\n"', 'Maycomb', '
go', 'time', 'ould', 'ome', 'Ewell', 'rea', ' be', 'people', 'Miss ', ' wa', 'back', 'tion', ' of', ' as', 'ther', 'Jean
Louise', "in'", 'Dill', '  "', ' ca', 'ent', 'our', ' again', ' on', ' say', 'ear', '\nChapt', ' st', '?"\n', 're ', 'room',
'They', ' ma', 'long', 'side', 'ant', 'ell', 'folks', ' ha', ' so', 'ter', 'You', 'talk', 'family', ' us', ' up', 'Underwood',
'rememb', 'n****r', 'nd Sykes', 'way', 'ind', ' sh', 'with', 'ard', 'ink', 'for', 'see', 'get', ' day', 'not', 'is ', 'little',
'body', 'old', ' at', 'front', '\n\n\n\n\n\n\n\n\n\n\n', 'ree', ' co', 'nce', 'hen', 'defend', 'and', 'Tate', 'ess',
'ice', ' my ', ' si', 'turn', 'pla', 'ake', ' su', ' no', ' found', 'minute', 'Gilmer', 'chiffarobe', ' wo', "n't", 'unt',
'left', 'Stephanie', " 'em", 'tried', 'nder', 'ly ', 'Uncle Jack', 'porch', 'him', ' fa', ' mo', 'She', ' if', 'across',
'like', '-"\n"', 'Raymond', 'ive', ' la', '"  ', 'cha', 'lea', 'say', 'ask', ' Tom', 'ill', ' but', 'an ', 'pretty', ' are', 'own',
'eck', 'Mrs. Merriwea', 'es,', 'ble', 'point', 'Negro', 'oth', 'He ', 'ick', 'ques', 'church', ' knew', 'black', 'ood',
'eir ', 'said', 'Cecil Jacobs', 'jury', 'art', 'ey ', 'let', 'run', 'ite', 'ist', 'in ', 'me ', 'be ', 'probably', '?  ', 'next', ',"  ',
'son', 'just', 'swer', 'est', 'trial', 'Bob Ewe', 'balcony', 'nev', 'scar', '!  ', 'self', 'Cunn', 'face', 'friend', 'man',
'at ', 'day', 'ine', 'try', 'ed,', 'neighbor', ' re', 'ass', 'each', 'even', ' is', 'ose', 'pen', 'ile', 'Boo ', 'from', ' saw',
'tch', ' how', 'age', 'ous', 'fir', 'per', 'But', 'Rachel', 'es ', 'door', 'flower', 'Arthur', ' much', 'also', '...', 'hole', '
put', ' boy', 'mor', 'oll', ' ra', 'decided', 'Francis', 'Dolphus', 'tor', 'got', 'ook', 'walk', 'girl', 'dea', 'quiet', ' ju',
'Mrs. ', 'yth', 'pro', 'aught', "'ve", 'unch', 'wor', 'ked', ', I', 'disorderly', 'ain', 'eep', 'spec', 'act', 'gra', 'judge',
'spoke', 'fee', 'she', 'con', ', Sc', 'believe', 'ense', 'witn', 'Why', ' or', 'und', 'ham', 'you', 'sho', 'We ', 'One',
'ept', 'ate', 'ture', 'rong', 'stop', 'com', 'had', ' hi', ' ev', 'row', 'ame', 'sit', ' pa', 'lized', 'Did', 'guilt', 'arm',
'ies', 'Walt', 'ise', 'tra', 'eye', ' by', 'stay', 'Gilm', 'the', 'ful', ' af', 'Cal', ' ai', 'broke', 'on ', 'People', 'any', 'ath',
'two', 'gave', 'expect', 'felt', 'head', 'ten', 'Burris', 'life', 'yond', 'speak', 'Novemb', 'stume', ' ba', 'use',
'rry', ' dr', ' di', 'ld ', ' ch', ' se', 'off', 'opp', 'Ewe', 'almo', 'ted', ' sc', 'mean', 'How', 'help', 'step', ' bus',
'snow', ' fi', 'med', 'iss', 'tinue', ' mu', 'rif', 'ank', 'Rev', 'Helen', 'color', ' Deas', 'Tom', 'Alabama', 'ner',
'ack', "I'm", 's, ', 's.\n', 'made', 'dden', 'foot', 'danger', 'middle', 'noon', 'lad', 'emselv', 'ish', 'ock', 'int', ' cl',
'new', "'re", ' ex', "'d ", 'my ', ' al', 'her', 'slow', 'ttle', ' trou', ' po', 'eat', ' lo', 'pull', '!"\n', 'irt', '-"\n',
'peculiar', 'Robin', '\n\n\n\n\n', 'of ', ' sp', ' en', 'his', 'liv', 't, ', 'ipp', '.\nT', 'to ', 'air', 'ich', 'ied', 'go ', 'tru',
'tire', 'but', '****', 'wait', ' adv', 'y, ', 'Boo', 'e, ', 'Tribune', 'was', 'gin', 'cry', 'em ', 'piece', ' sm', ' Maudie', '
Craw', 'ese', "ey'", '.\nI', 'Zeebo', ' Papa', 'squa', 'rri', 'too', 'nt ', 'saw', ' my', 'cor', 'pre', 'Bob ', ' Chr',
'hurt', 'else', 'od ', 'Yes', ' br', 'jail', 'hap', 'I c', 'ush', 'who', 'ump', 'nge', '. \n', '"Do', 'ort', 'ict', 'summ', 'it ',
'descri', 'shut', 'set', 'stead', 'twelve', 'I w', 'ans', 'ect', 'view', 'che', 'd.\n', 'It ', ' argu', 'accus', 'lawyer',
'gun', "'em", 'gonna', '?" ', 'Tim John', 'een', 'burst', 'apolog', 'Dr. Reyn', 'uffl', 'at,', 'case', 'Taylor', 'floor',
", '", 'Hitl', 'emocracy', 'mov', 'as ', 'Let', 'Now', 'ity', 'And', 'ugh', ' fu', 'lot', 'uck', 'eir', 'put', 'uch', 'add', ' - ',
'Who', 'fin', 'ild', ' ye', 'Even', 'y.\n', 'stud', 'wn ', 'ambl', 'ad ', ' cu', 'icul', 'ers', 'ion', 'Aft', 'al ', 'imp', 'tinfoil',
'anno', 'led', 'mon', 'did', 'aybe', 'tak', 'ned', 'so ', 'No,', 'car', "'am", 'ruin', ' gr', 'bird', 'dark', 'Co-Cola',
'half', 'few ', 'up ', 'how', 'gan', 'de ', 'cho', 'ry ', 'ast', 'ure', 's a', 'ert', 'pri', 'we ', 'end', ' bl', 'roa', 'sneak',
'cross', 'llars', '.\nW', '"Th', ' de', 'bit', ' hymn', 'climb', 'ors', 'r, ', 'now', 'hot', 'str', ' Charl', 'bett', 'ula', 'arr',
'ove', " I'", ' le', ' Wal', 'hav', 'ret', 'h, ', 'ny ', 'ped', ' Atkin', ' wi', 'sti', ' li', 'Like', 'ord', 'tow', 'only', '.\nS',
'bor', '"He', ' cem', 'teleph', 'ded', 'diff', 'ugly', 'oot', ' job', 'ave', 'ftly', ' Sun', ' sa', ' fe', 'duc', 'Alex', 'eas',
'writ', 'ical', 'raid', 'los', ' circumsta', 'big', 'Our', ' hu', ' ti', 'boy', 'eek', 'rid', 'r m', ' na', 'ran', 'gry', 'cra', '." ',
' qu', 'box', 'men', 'ife', '"I ', 'ywh', 'dis', 'why', 'The', ' sw', 'Are', 'y,"', 'DEMOCRACY', 'ton', 'us ', 'Folks', 'y
c', 'mer', 'ows', 'no ', 'cle', 'God', 'rmur', ' sl', 'supp', 'ima', 'res', ' ov', 'oes', 'I gu', 'spa', 'kids', 'kitc', 'urr',
'low', ' Sc', 'welfa', 'nor', ' Indian-', 'liz', 'With', 'mas', 'ntinu', 'sin', 'pol', '.\nLa', 'd, ', 'figur', '"It', 'visi', ' te',
'by ', 'do ', ' rubb', 'mat', 'tal', 'gla', 'absolute', 'serv', '. "', '.\nA', 'urni', ':  ', 'damn', 'group', 'acqu', ' ag',
'injur', 'okay', ' jerk', 'high', 'cid', 'buy', 'doc', 'law', 'dur', 'rac', 'rap', 'rep', 'app', '  I', 'emb', 'ins', 'unn',
'mil', 'ial', '!" ', 'beh', 'All', ', c', 'ta ', ' Di', 'ves', 't a', 'ts ', '"Wh', 'Not', 's,"', ' Co', 'win', 'rag', ' ne', 'par', 'dog',
'rup', 'isp', 're,', "an'", 'red', 'r,"', 'squeeze', 'rel', ', e', 'six', 'ugg', 'sci', 'itt', 'ime', 'ory', 'rew', ' ta', 'bur', 'ail',
'rigg', 'So ', 'pay', 'load', 'oke', 'val', 'r.\n', ' un', 'des', 'le ', 'owed', 'gum', 'h.\n', 'bed', 'ar ', 'warn', 'd. ', 'r g',
'ssi', 'a p', 'se ', ' fl', 'ust', 'are', 'join', 'thr', '\n\nI', ' wr', 'an,', 'alth', 'l n', 'odd', ' awh', ', p', ' am', 'eca', 'curi',
'lace', '-lo', 'ifi', "I'd", 'ott', ', H', ' da', 'ce,', 'ily', 'esca', 'ide', 'adow', 'Go h', 'edge', ' Braxt', "' k", 'Naw,',
'"Was', 'ung', 'r n', ' Vio', "Long's", " o'", 'public', 'Adolf ', 'My ', 'ays', 'us,', 'ual', '"So', 'a r', ' cr', 'mak', 'eth',
'ow,', 'ool', 'spr', 'Fin', 'ets', 'ift', 'e.\n', 'drugs', 'imm', 'lap', 'uar', 'a s', 'ruler', 'bab', 'ncy', 'ps ', 'nds',
'iskey', 'ult', 'oak t', 'I s', 'hid', 'fun', 'joy', 'hop', 'lie', 'ngs', 'ney', 'Tsk. ', 'bro', 'do,', ' yo', 'At ', 'pee', "'m ",
"at'", 'bul', ' Ma', 'ngl', 'lat', 'a b', ' ga', ', a', 'se,', '"We', 'mpl', 'r r', 'roy', 'ybe', 'umb', 'gisla', 'n.\n', 'mix',
'avoid', 'nee', 'Fir', "we'", 'a g', '1\n\n', 'pic', 'tic', 'rks', 'cut', 'adv', 'uti', 'ian', 'rch', 'nts', 'top', 'lla', ' ap',
'rec', ', A', 'r a', 'vel', ', w', 's w', 'en,', 'surpr', 'jar', 'lor', 'bruis', 'cabin', 'rem', '"Sc', "?' ", 'eld', 'ach', "Id'ja",
'arply', 'jud', 'Ger', 'Hel']
```