Project Proposal for CSE549
Fall 2011

Reconstructing Books using Google NGrams

Himanshu Jindal
Piyush Kansal

Contents:

1. Project proposal report

a. Objective:

The objective is to reconstruct books for a given year using Google NGrams with following factors in mind:

- Reconstruction accuracy
- Extent to which one can reconstruct
- Running time

To start with, we are supposed to pick a book (.txt file) from Project Gutenberg, create a 5grams file out of it just like Google does and then try reconstructing it. If it is able to achieve the desired accuracy, then apply the same on Google NGrams and check how far the reconstruction goes.

b. Implementation Concept:

We have come up with following:

- After getting all the 5gram data sets from Google, filter them and extract 5grams only for a given year
- Then apply one of the following reconstruction strategies, whichever gives better results based on above three factors

We have thought of following reconstruction strategy using 5grams:

Implementation using Hash tables

- Scan all the 5grams and store them in a list (using STLs in C++)
- Read 5gram from the list one by one until just one is left
- Match the last four grams of this 5gram with the first four grams of an another 5gram in the list
- If the maximum match is found, form a super string out of it and store it in the same location from where first 5gram was read
- This way, the length of this super string will increase to 6. So, we will now its take last 4 grams and again above steps till we found a mismatch. At that point we will move

to a different 5gram and repeat the above steps. Finally, our list will have just have 1 superstring left and that will represent the book

- To avoid searching in the whole list (which will be really huge), we will match the very first character of the 4gram (last four grams of current 5gram) with all 5grams starting with the same character in the list. To make this search fast, we have thought of following:
  - Maintain a hash table for 128 ASCII characters. Each entry of this table consists of following:
    - an integer value of the ASCII character
    - iterator or starting address of the list from where all the 5grams whose first character matches with this character are stored
    - a count of all of such 5grams
- The combination of above three will help in localizing the search area
- Although, in this list some of the characters will never show up like non-printable characters. But this is just to make the access to hash table in O(1) time

Implementation using Tries

- We think that the above approach of searching and then matching can be optimized using Tries
- Instead of using hashing, we can store all the 5grams in a word trie
- This can give us very fast lookup time and can save the time which we are spending in matching with each of the 5grams in a given area in the list

c. What has been tried

- To start with, we have taken a book from Project Gutenberg
- We have written a program to shred this book in 5grams, sort it and then create a Google like data set with number of occurrences on each 5gram appended to it
- We passed this data set to our reconstruction program but is not giving us the desired output

The program terminates after running for some time with segmentation fault. We have localized a bug in our hash table implementation. We are working on it and expect to resolve it soon

d. What is to be tried

- We have to evaluate the above program on the three factors listed above
- In parallel, we will develop another program to implement it using Tries and will then compare the final results on the book from Project Gutenberg
- We are expecting issues with the reconstruction accuracy and extent of reconstruction and yet to think of applying any heuristics, if possible

2. Progress report

a. Word Trie lookup

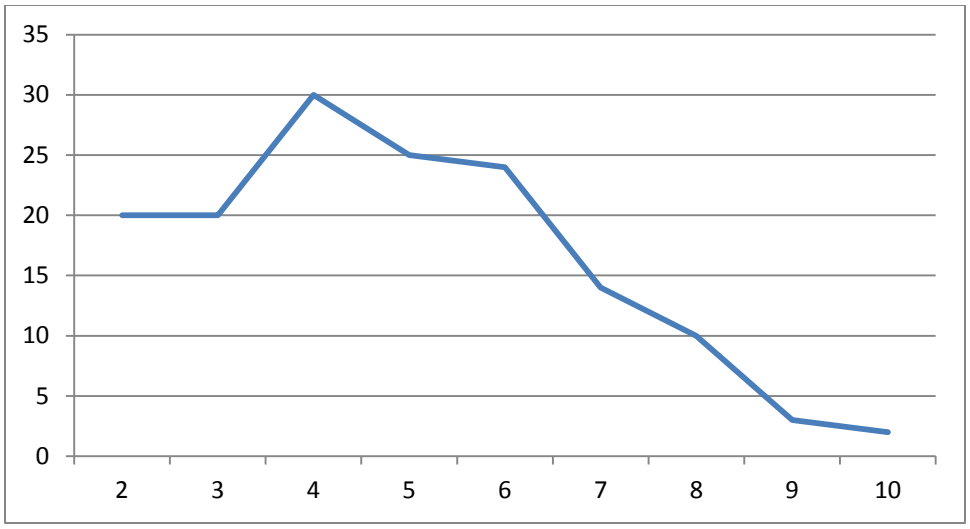
We implemented the look up using Word Trie and then created a List (provided by C++ STL) for storing all the NGrams. After getting the execution result which we showed to the professor, we figured out following shortcomings of this approach:

- The reconstruction time was not impressive. It took around 30 min for reconstructing just one single book
- The reconstruction length was not that good. We could have tried improving it but that would mean increasing the reconstruction time even further

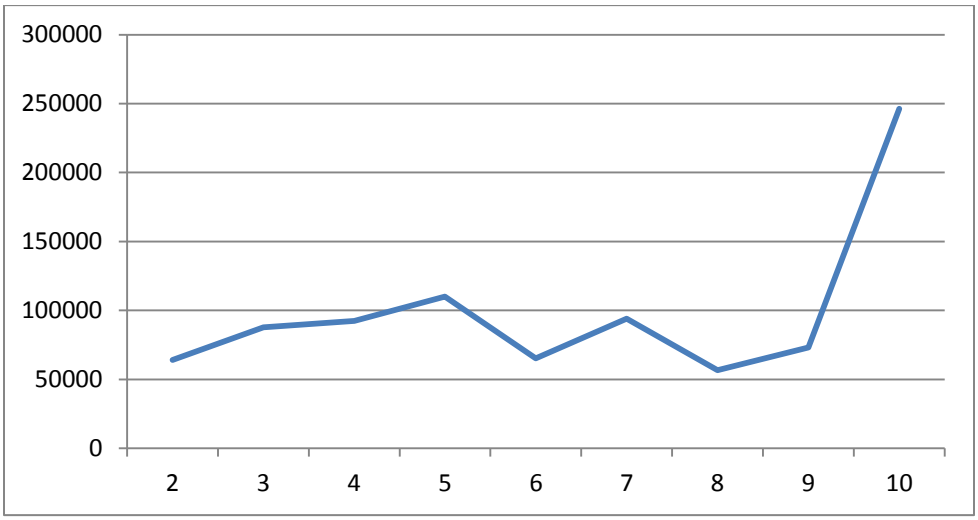
Hence, we looked for other option which could improve the reconstruction time

b. Hash map lookup

- We chose hash map and used FNV hash and stored the NGrams and the hash in a Vector (provided by C++ STL) for O(1) lookup time
- The reconstruction time went down to 2 minutes for the same book
- Then we tested our output to see whether all the NGrams strings are present in the output superstring and they were present
- As suggested by the professor, we then tried testing from N=2 to 10 NGrams to check the effect on the length and frequency of the various fragments. Following are our results:
  - The reconstructed fragment is quite long and the accuracy is very good for smaller values of N
  - When N is large, the reconstruction length is good. For example, we found a fragment of 300 continuous matching words. However, we see that the accuracy is not that good. It is happening if there are 2 strings which have the same matching prefix as our base string. We think that this can be fixed by applying some kind of NLP which can tell us which word to choose, although is some cases like this, even NLP would not help:
    - Steve Jobs, founder of Apple
    - Steve Jobs, founder of Pixar
- Based on the reconstruction, we have plotted following data:



X-axis: NGrams(2<=N<=10), Y-axis: Number of fragments



X-axis: NGrams(2<=N<=10), Y-axis: Length of maximum fragment (in words)

Data corresponding to above graphs

Number of Fragments/N	2	3	4	5	6	7	8	9	10
1	26846	14494	2729	9354	41728	868	56700	29462	2995
2	1746	18015	28451	105	65328	47	63	22112	246185
3	11816	1885	524	164	625	137	1270	73031	
4	677	2214	13	737	404	11966	48		
5	7754	7	96	450	80	954	758		
6	1108	90	53	1643	164	10448	28667		
7	7411	5	13	1529	130	2204	30681		
8	1952	4	58	168	356	3901	5571		
9	15	7	17	17	357	30	19		
10	78	21	8	59	6458	188	868		

11	17	4	26	30	6321	25			
12	3	22	13	15	438	18			
13	5	7	11	33	169	37			
14	577	4	36	14	18	93833			
15	64301	5	12	15	238				
16	64343	8	7	35	167				
17	6	7	11	37	60				
18	64263	4	31	12	15				
19	64268	5	9	16	127				
20	110	87802	19	14	24				
21			62	50	17				
22			13	14	45				
23			7	14	15				
24			20	9	1399				
25			13	110129					
26			7						
27			10						
28			7						
29			12						
30			92361						

c. What is Next

- We have to focus on increasing the length of the accurate strings
- As of now, we have thought of the following:
  - Consider the NGrams as nodes in the graph. Two nodes having similar suffix and prefix as edges. Hence, anytime we have 2 edges leading from a node, we do reconstruction for both the paths and dump both the strings in the output
    - Pros and Cons
      - The output obtained by this method would have many junk strings. However, the accurate strings will be of much larger lengths
      - The running time would increase, as this method uses naïve technique of exploring each option in the graph to find the right string
  - We are also looking at implementing an intelligent way of knowing which string to choose for appending instead of the naïve approach. For example, to get good accuracy, we have to implement a heuristic of some kind that given two choices of words, explore the strings formed with both these words for considerable length and then tell us which path makes more linguistic sense. To achieve this, we explored few options like OpenNLP and WordNet but we don't see them fitting in our context

d. Questions/Suggestions

- We are looking forward for any questions/suggestions from the professor