Final Project Report for CSE549
Fall 2011

Reconstructing Books using Google NGrams

Piyush Kansal
Himanshu Jindal

PREVIOUS WORK

We had prepared a program which performed reconstruction using hashing.

SHORTCOMINGS

1. *Inaccurate reconstruction*
2. *Reconstruction time: about 2 min per book.*

IMPROVEMENTS

We have now optimized our hashing program and have got excellent results.

1. **Reconstruction accuracy is now 100%**
2. Reconstruction time for a single book is **0.48 sec** for **5grams**
3. 250 books were recombined in **3 min 7 seconds** using **5grams**
4. The **n50 value** for recombining 250 books was **19 for 5grams** and rose up to **7007 for 10grams.**

ALGORITHM

Given a document containing 5grams, the reconstruction algorithm that our program follows

1. Store the grams in a vector along with their occurrences.
2. Compute fnv-hash for the prefix containing first 4 word for each gram and the position where it is present and store this pair in a hash map implemented as a simple array.
3. Iterate through the vector from start to end
    1. Compute the hash of the suffix containing last 4 words of the current gram.
    2. Look for position returned from the hash map for this hash value.
    3. Match the suffix and prefix using string function to eliminate the chance of collision.
    4. If there is only 1 position which matches the given suffix with occurrence>0, then merge the current gram with the gram at this position decreasing its occurrence by 1
    5. If more than 1 or 0 positions are returned, then go to the next element in the vector
    6. Else, repeat for the same element.
4. Dump the elements from the vector, with occurrences >= 1, into the output file.

Our reconstruction algorithm performs very well. It is of the order of **O(N)** where n is the sum of the occurrence value for all the grams.

THE BOTTLENECK

However, the bottleneck is the amount of memory required. The program requires

1. Memory for all the ngrams
2. Memory for hash values of all the ngrams.

We ran the tests on our machine with 4GB RAM. For 250 books, the program executed perfectly and in very small time. However, for 500 books, the program reconstructed from 2grams till 7grams, but could not execute any further due to lack of memory.

Computation time would still be in minutes even for tens of thousands of books and the only bottleneck is the memory required.

RECONSTRUCTING BOOKS FROM GOOGLE NGRAMS

- The next step would be to run the algorithm on googles ngrams for a given year. However, the total number of these grams would be huge and would require high amount of system memory.
- Hence, to check the reconstruction accuracy on data resembling google ngrams, we made ngrams out of 250 books and removed grams with occurrences less than k, for k ranging from 1 to 5.
- The results were really good and the n50 value was still respectable.
- This shows that this algorithm would run considerably well on google ngrams as well as they have data from large number of books after removing the grams with low frequency.

RESULTS FORMALIZED IN GRAPHS

We ran the reconstruction algorithm for 1, 10, 50, 100, 150 and 250 books and have plotted the important statistics we obtained from each reconstruction and the statistical inference we made from each.

- Reconstruction length as a function of n50 for 250 books
- Reconstruction time
  - for different ngrams of 250 books and
  - for different number of books
- Fragment Length for reconstructing 250 books.

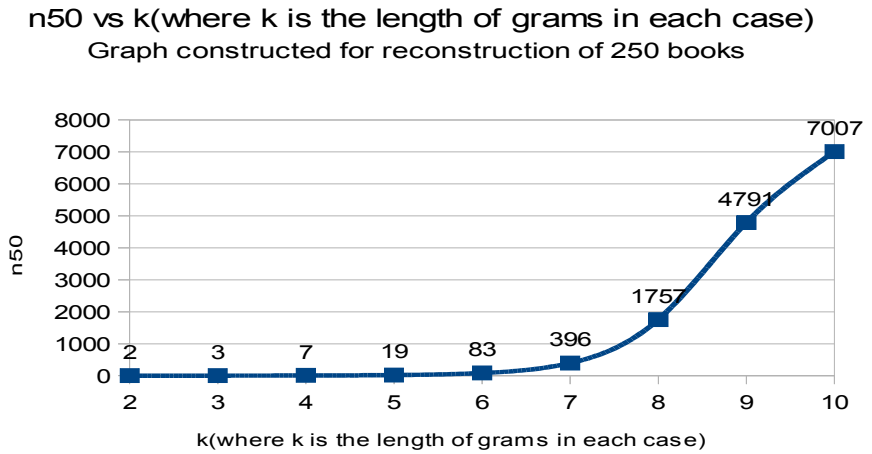
RESULTS FORMALIZED IN GRAPHS

Length of reconstruction

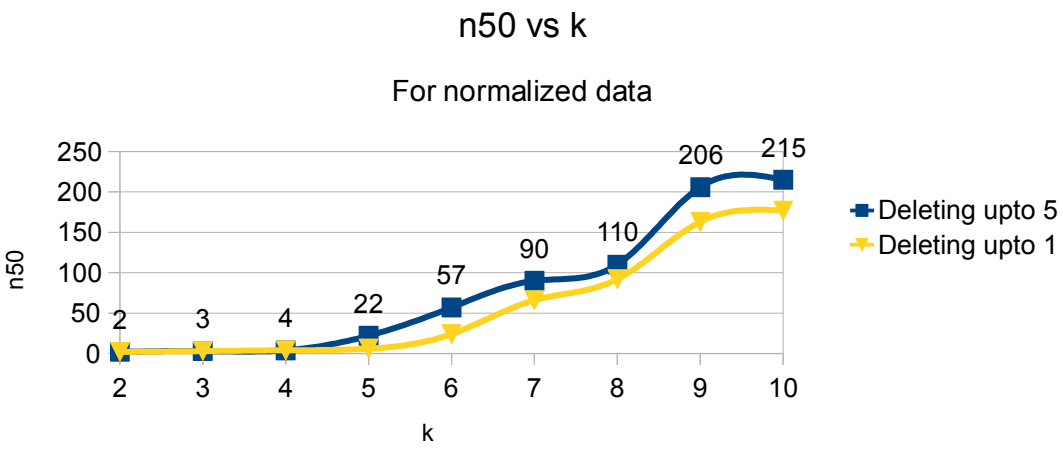
N50 value vs k(number of words in each gram)

In this graph, we have plotted the n50 value vs the kgram we have used for the reconstruction of 250 books. Where, k is the number of words in each gram used for the specific reconstruction.

Inference: n50 value **increases very fast after k>5** and for 10, it is 7007, which implies that we can recover huge contiguous chunks from a jumbled heap of 250 books, if given 6grams or more.



N50 vs k for normalized data

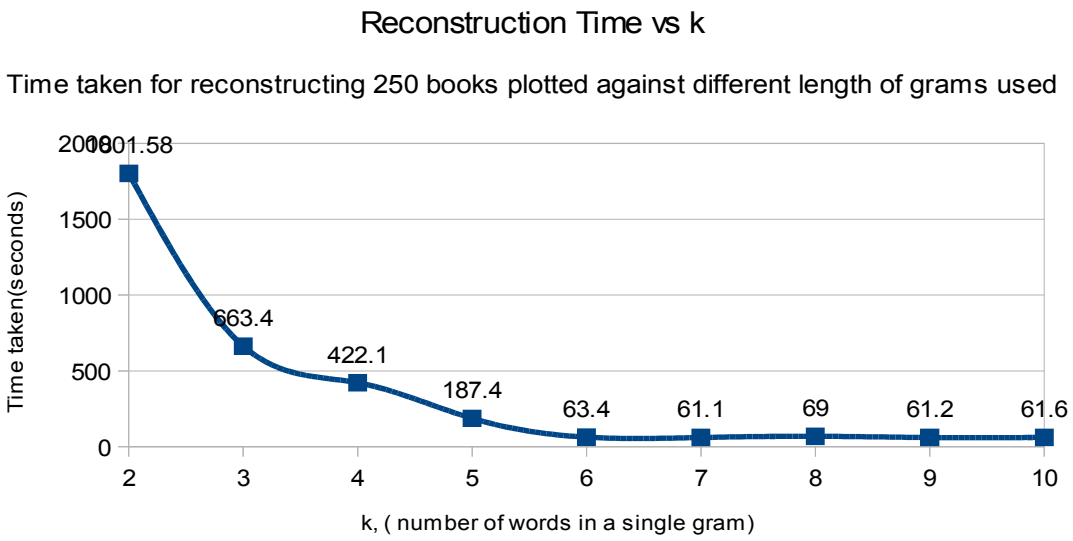


We removed the ngrams with low occurrences and ran our reconstruction algorithm

Inference: **Reconstruction length is still quite long** and stays about the same if we delete all grams with occurrence 1, or we delete all grams with occurrence <= 5.

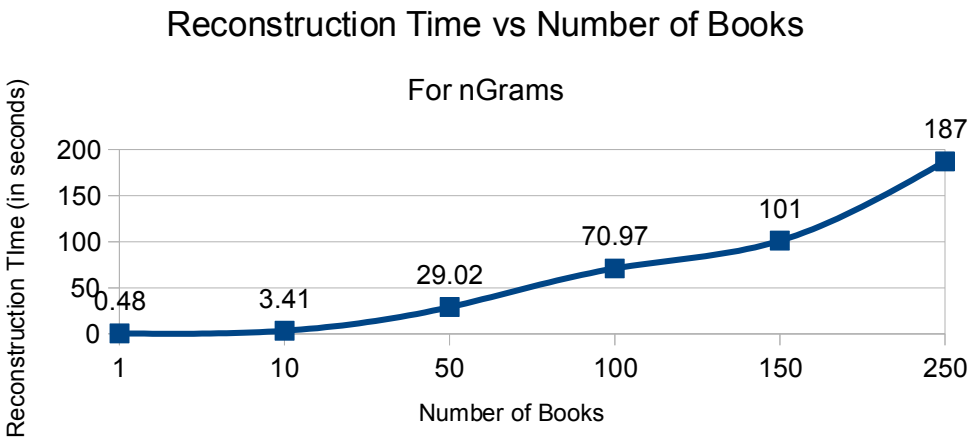
Reconstruction Time

Reconstruction time vs k(number of words in each gram)



Inference: The reconstruction time falls immensely after 5 and stays about constant. **THIS COULD BE THE MAIN REASON why google did not release 6grams or more.**

Reconstruction time for 5grams vs Number of books reconstructed



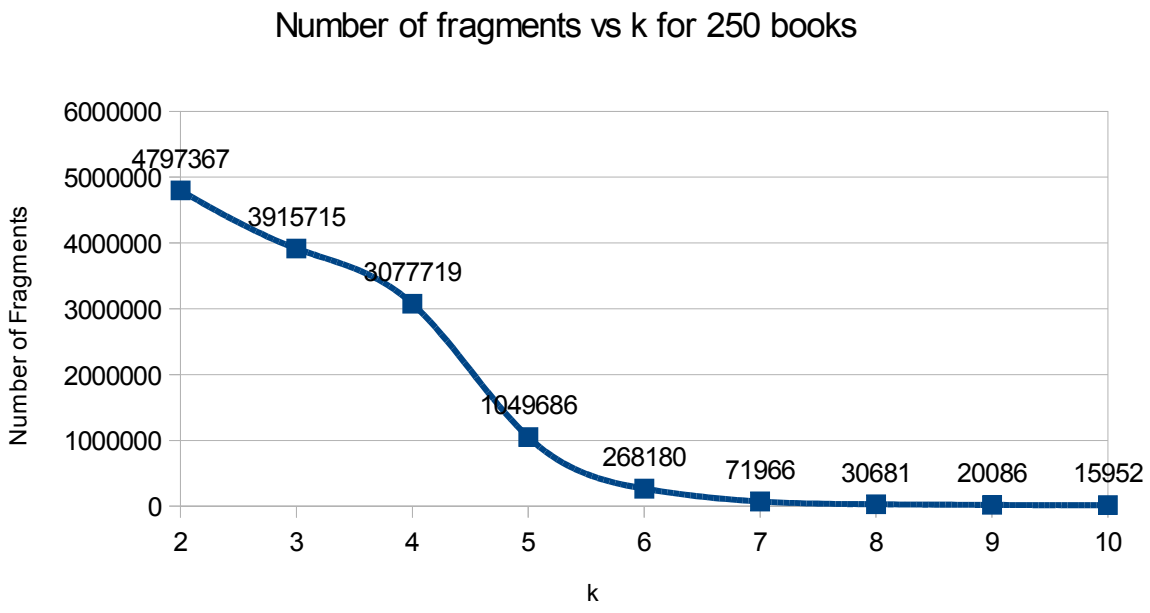
Inference: The time taken increases as number of books are increased. It should be noted that the time taken for reconstructing 100 books with 5grams is more than reconstructing 250 books with 6grams which again reinforces our belief about why google stopped at 5grams.

Number of Fragments formed

Number of fragments vs k

Graph shows number of fragments formed while reconstructing 250 books from 2grams to 10grams.

Inference: The number of fragments decreases by the maximum amount for 5gram to 6gram to 7gram (By a factor of 5). Which reinforces our belief that this could be the main reason google is not giving away the 6grams and 7grams that they have used for their analysis.



CONCLUSION

Hence, we see that implementing the given hashing based algorithm drastically improves the speed and accuracy of reconstruction. And as seen by its results on data resembling google ngrams, we can conclude that good reconstruction could be achieved on google ngrams as well. Moreover, the project was a great learning experience and we learnt many things that were completely unknown to us.

1. "***wget***" shell command: We wrote a script to download large number of books
2. "***tr***" shell command (existing): To remove Ctrl-M characters from the i/p files
3. ***File Splitter*** (new implementation): To split the Gutenberg files into i/p number of kGrams
4. "***sort***" shell command (existing): To sort the data from step 2
5. "***uniq***" shell command: To count the number of occurrences of a given line from step 3 quickly
6. ***Reconstruction Program*** (new implementation): To reconstruct the book on i/p data from step 4
    1. We learned how to implement tries
    2. Implemented FNV hash
7. Finally, we ***integrated*** everything in a shell script and made it quite user friendly to execute. For eg, ./recn.sh <no-of-books> <min-k> <max-k>

CHALLENGES FACED

1. All of the files from Gutenberg had Ctrl-M characters in them. May be because they were created in a Windows based editor. Since our implementation is Linux based, we had to remove all of them. So, we used "tr" command for it.
2. Before using "uniq" as mentioned in step 4 above, we were using a C++ program to count the occurrences. This was quite slow and used to take around 20 min even for 100 books. We then posted it on a forum and got to know about "uniq". It was surprisingly fast. We also learned what made it so fast. It was quite helpful as it reduced our overall testing time (Ref: http://stackoverflow.com/questions/8364447/how-to-compare-all-the-lines-in-a-sorted-file-file-size-1gb-in-a-very-effici)

3. Initially we used a Brute Force algorithm (from Homework 1) and tried reconstructing 1 book from Gutenberg. The program ran for almost 12 hours after which we killed it. We realized a need for a better algorithm
4. We then implemented a Word Trie along with Hash Table. This algorithm reconstructed the the same book in 20 min. It was still not good as reconstructing more books would have taken hours
5. We discarded the above data structure as well and finally implemented FNV hash which did its job in just 30 sec. This was quite satisfactory
6. As per directions from Professor, we were supposed to run reconstruction for kGrams (k=2 to 10). Moreover, we were supposed to experiment with different number of books and with different number of rare grams deleted. Professor also suggested us to create graphs for N50, Number of Fragments and Reconstruction Time. This needed substantial amount of ***effort***. Moreover, doing all this stuff manually was quite ***error prone***. This made us think in a direction to write a code which could automate all the steps required along with report generation. This gave us enough motivation to implement a script which did all the required stuff by just giving 3 i/ps i.e number of books to reconstruct, min-k and max-k. With this our job became quite easy and gave us a good insight into implementing a good flow in the project.

INTEGRATED SCRIPT

Here is the script that we implemented that automated the process of constructing ngrams from books, reconstructing books from them and then dumping the required data for creating report from them.

```
#!/bin/bash

# File: recn.sh
# I/P:  1. Number of books to reconstruct
#       2. Min k
#       3. Max k
# Flow:
# 1. Remove Ctrl-M characters from the i/p file
# 2. If the splitted file exists (from previous run), don't split it again
# 3. Split i/p file into required number of grams
# 4. Merge the requested number of files having kgrams in a single file
# 5. Sort the merged file
# 6. Create grams from the sorted file
# 7. Reconstruct book
```

```
# 8. Calculate statistics
# 9. Remove intermediate files created for nGrams

# Check validity of the arguments
if [ $# -lt 3 ]
then
    echo "Invalid no of arguments"
    echo "Usage: $0 <Number of files> <min no of grams> <max no of grams>"
    exit 1
fi

# Initialize arguments
FILECNT=$1
MINGRAMS=$2
MAXGRAMS=$3

# Set the directory to point to External Hard Disk and save the current
directory in CWD
CWD=$PWD
cd "/media/My Book/CB/"

# Set redirection for the log files
exec 1>$CWD/$FILECNT-$MINGRAMS-$MAXGRAMS-`date +'%Y%m%d%H%M%S'`.log 2>&1

# Sort the splitted file
sortFile() {
    rm -f $1.sort
    sort $1 >> $1.sort

    if [ $? -ne 0 ]
    then
        echo "Some error occured during sorting of $1"
        exit 1
    fi
}

# Generate statistics
genStats() {
    rm -f $1.stat*

    awk '{ print NF }' $1 >> $1.stat

    sort -g $1.stat >> $1.stat.2
    rm -f $1.stat
    mv $1.stat.2 $1.stat

    uniq -c $1.stat | awk '{ for( i = 2 ; i <= NF ; i++ ) printf( "%s%s", $i,
OFS ); print $1 }' >> $1.stat.2
    cat $1.stat.2
```

```
    N50=`cat $1.stat.2 | awk 'BEGIN{N50=0} {N50+=$1*$2} END{print N50/2}'`
    cat $1.stat.2 | awk -v var=$N50 'BEGIN{F=0;FL=0} {FL+=$1*$2; if(F==0 &&
FL>=var) F=$1} END{printf "=====================================\nN50 ::
%d\n", F}'

    echo "No of fragments :: "`wc -l $1`
    echo -e "=====================================\n"

    rm -f $1.stat
    rm -f $1.stat.2
}

minGrams=$MINGRAMS
maxGrams=$MAXGRAMS
curFileCnt=9
for (( i = 0 ; i < FILECNT ; ))
do
    curFileCnt=`expr $curFileCnt + 1`
    fileToSplit=./PGBooks/$curFileCnt.txt

    # If the splitted file already exists (from previous run), the don't
split it again
    if [ -f $fileToSplit ]
    then
        echo "Processing "$fileToSplit" ..."

        # Take care of "^M" characters
        cat $fileToSplit | tr -d '\15' >> $fileToSplit.2
        rm -f $fileToSplit
        mv $fileToSplit.2 $fileToSplit

        for (( minGrams = MINGRAMS ; minGrams <= maxGrams ; minGrams++ ))
        do
            rm -f ./PGBooks/$FILECNT"Books.txt"$minGrams.sort.gram.recn

            if [ ! -f $fileToSplit$minGrams ]
            then
                # Split the file
                $CWD/fileSplitter $fileToSplit $minGrams
                if [ $? -ne 0 ]
                then
                    echo "Some error occured during file splitting for
$minGrams""grams"
                    exit 1
                fi
            fi
        done
```

```
        i=`expr $i + 1`
    fi
done

echo ""

minGrams=$MINGRAMS
for (( ; minGrams <= maxGrams ; minGrams++ ))
do
    mrgFile=./PGBooks/$FILECNT"Books".txt$minGrams
    rm -f $mrgFile

    # Merge the requested number of files having kgrams in a single file
    curFileCnt=9
    for (( i = 0 ; i < FILECNT ; ))
    do
        curFileCnt=`expr $curFileCnt + 1`
        curFileToMrg=./PGBooks/$curFileCnt.txt$minGrams

        if [ -f $curFileToMrg ]
        then
            cat $curFileToMrg >> $mrgFile
            echo "" >> $mrgFile
            i=`expr $i + 1`
        fi
    done

    # This check is really important just to ensure the no of grams
    err=`cat $mrgFile | awk '{ print NF }' | sort -u | wc -l`
    if [ $err -gt 1 ]
    then
        echo "Number of fields in file $mrgFile are more than $minGrams"
        exit 1
    fi

    # Sort the merged file
    sortFile $mrgFile

    # Create grams for the sorted file
    uniq -c $mrgFile.sort | awk '{ for( i = 2 ; i <= NF ; i++ ) printf( "%s
%s", $i, OFS ); print $1 }' >> $mrgFile.sort.gram
    if [ $? -eq 0 ]
    then
        echo `wc -l $mrgFile | awk '{print $1}'` $minGrams"Grams created
successfully from all "$FILECNT" requested books"
    else
        echo "Some error occured while creating $minGrams""Grams"
        exit 1
```

```
    fi

    # Reconstruct book
    time $CWD/recnBook $mrgFile.sort.gram $minGrams
    if [ $? -eq 1 ]
    then
        echo "Some error occured while reconstruction for $minGrams""Grams"
        exit 1
    fi

    # Generate statistics
    echo -e "\nGenerating Statistics on number of words in each fragment and
the count of such matching fragments ..."
    genStats $mrgFile.sort.gram.recn

    # Remove intermediate files created for nGrams
    rm -f $mrgFile
    rm -f $mrgFile.sort
done
```