

# Project #2 – Big Data

## Compressing large collections of web pages

*Federico Conte*

*fconte90@gmail.com*

*<https://github.com/Draxent>*

University of Pisa  
Department of Computer Science  
Master in Computer Science and Networking

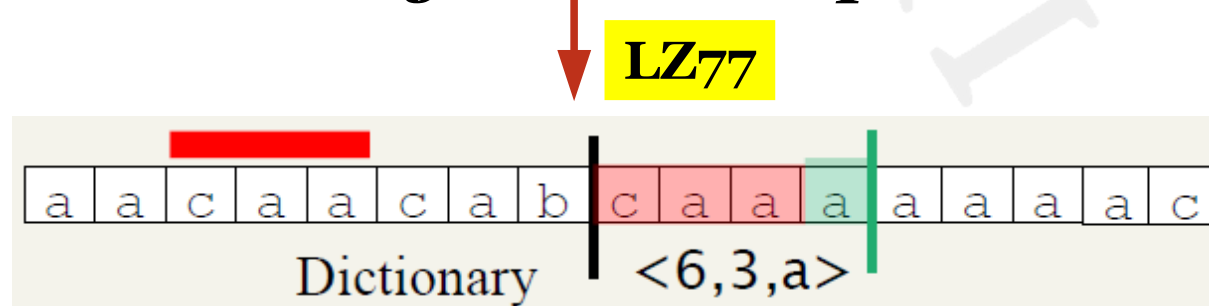
*6<sup>th</sup> April 2015*

# Problem Statement

**INPUT:** A file containing a collection of **Web pages**

**OUTPUT:** A file containing the permuted collection, where the permutation is driven by the **similarity** between pages

**GOAL:** Find the best way of permuting in order to **minimize the compression** of the output file (using Lzma2 compressor)



# Analysis of the Web pages

```
warc/0.9.10757 response http://www.mattenltd.co.uk/0061006192531 message/http :
BUbING-guessed-charset: iso-8859-1
BUbING-content-digest: eceb3de18182b34aedab72fb9a62846

HTTP/1.1 200 OK
content-type: text/html
accept-ranges: bytes
connection: close
server: Apache/1.3.33 (Unix) mod_auth_passthrough/1.8 mod_log_bytes/1.2 mod_bwli
content-length: 10123
last-modified: Thu, 13 Oct 2005 22:41:57 GMT
etag: "4f1df5-278b-434ee2b5"
date: Wed, 05 Jul 2006 07:44:13 GMT
ubi-http-equiv-charset: iso-8859-1

<html>
<head>
<META name="description" content="Matten Press Makers">
<META name="keywords" content="Press, Hand, ...">
<title>Matten Engineering</title>
<script language="JavaScript" type="text/JavaScript">
  <!--
    function MM_openBrWindow(theURL,winName,features) { //v2.0
      window.open(theURL,winName,features);
    }
  //-->
</script>
```

Start new web Page

Length of the Page

URL

Content-Type

Start HTML Page

TAG

Word

Script

# Analysis of the Web pages

## Assumption:

If someone **COPY** some TAGs / WORDs / SCRIPTs from an HTML page, he will **NOT MODIFY** them.

Start new web Page

Length of the Page

URL

Content-Type

Start HTML Page

TAG

Word

Script

```
<html>
<head>
<META name="description" content="Matten Press Makers">
<META name="keywords" content="Press, Hand, ...">
<title>Matten Engineering</title>
<script language="JavaScript" type="text/JavaScript">
  <!--
    function MM_openBrWindow(theURL,winName,features) { //v2.0
      window.open(theURL,winName,features);
    }
  //-->
</script>
```

# Proposed Solution – 1<sup>st</sup> Version

**Algorithm:** *CompressingWebPages(InputFile)*

begin

```
pi <- InitializePermutations(21)
lsh <- new LSH(7, 3)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
```

```
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
```

```
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc
```

```
foreach wp in (categWP union notCategWP)
  write wp in OutputFile
```

end

*Initialize 21 MURMUR HASH function*



# Proposed Solution – 1<sup>st</sup> Version

**Algorithm:** *CompressingWebPages(InputFile)*

begin

```
pi <- InitializePermutations(21)
lsh <- new LSH(7, 3)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
```

```
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
```

```
simDoc <- sort notCategWP by URL
```

```
calculate permutation order of simDoc
```

```
foreach wp in (categWP union notCategWP)
```

```
  write wp in OutputFile
```

end

## LSH-initialization:

- pre-compute sampling of **3** elements from **Sketch**
- create **7** hash tables with chaining

# Proposed Solution – 1<sup>st</sup> Version

**Algorithm:** *CompressingWebPages(InputFile)*

begin

```
pi <- InitializePermutations(21)
lsh <- new LSH(7, 3)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
```

```
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
```

```
simDoc <- sort notCategWP by URL
```

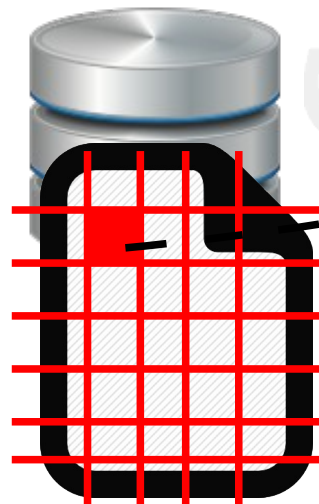
```
calculate permutation order of simDoc
```

```
foreach wp in (categWP union notCategWP)
```

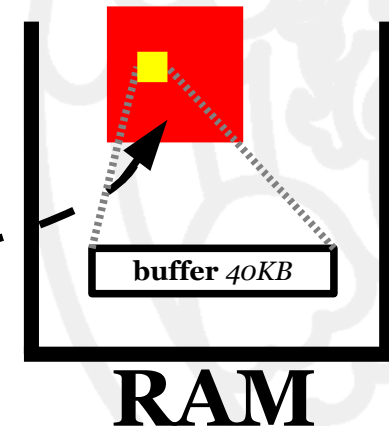
```
  write wp in OutputFile
```

end

Scanning File:



memory  
mapping



# Proposed Solution – 1<sup>st</sup> Version

**Algorithm:** *CompressingWebPages(InputFile)*

begin

```
pi <- InitializePermutations(21)
lsh <- new LSH(7, 3)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()

while(NOT sf.EOF)
{
    wp <- Recognize in sf a Web page
    if (wp.Signature != NULL)
    {
        lsh.AddDocument(wp)
        categWP.Add(wp)
    }
    else notCategWP.Add(wp)
}

foreach wp in categWP
{
    simDoc <- sort lsh.UnionFind(wp) by URL
    calculate permutation order of simDoc
}

simDoc <- sort notCategWP by URL
calculate permutation order of simDoc

foreach wp in (categWP union notCategWP)
    write wp in OutputFile
```

end

## Recognize Web page:

- Divide the Web page in **words** (*TAGs / WORDs / SCRIPTs*) → pair (start, length)
- Karp-Rabin hashing for every **word**
- Shingling of *x*-words, with *x* depending #characters (*Q=25*)
- Karp-Rabin hashing for every **shingle**
- Sketch Vector using **Min-Hashing** (*pi*)
- Return the end of the page → next step will start here



# Proposed Solution – 1<sup>st</sup> Version

**Algorithm:** *CompressingWebPages(InputFile)*

begin

```
pi <- InitializePermutations(21)
lsh <- new LSH(7, 3)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
```

```
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
```

```
simDoc <- sort notCategWP by URL
```

```
calculate permutation order of simDoc
```

```
foreach wp in (categWP union notCategWP)
```

```
  write wp in OutputFile
```

end

## LSH-AddDocument:

- pick 3 elements from the Sketch
- compute the sum
- add result to 1 of 7 buckets
- if there are collision → we list them

**Threshold**  
~52%

# Proposed Solution – 1<sup>st</sup> Version

**Algorithm:** *CompressingWebPages(InputFile)*

begin

```
pi <- InitializePermutations(21)
lsh <- new LSH(7, 3)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc
```

```
foreach wp in (categWP union notCategWP)
  write wp in OutputFile
```

end

- ← **Calculate Permutation:** *For every page:*
- *find all similar pages (union find LSH)*
  - *sort by URL*
  - *calculate the position of the permutation (NewStartPage)*

# Proposed Solution – 1<sup>st</sup> Version

**Algorithm:** *CompressingWebPages(InputFile)*

begin

```
pi <- InitializePermutations(21)
lsh <- new LSH(7, 3)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
```

```
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
```

```
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc
```

```
foreach wp in (categWP union notCategWP)
  write wp in OutputFile
```

end

## Write on OutputFile:

*Since we have calculated the new position that the Web pages will occupy:*

- scan sequentially the **input file**
- write randomly into **output file**

# Proposed Solution – 2<sup>nd</sup> Version

Algorithm: *CompressingWebPages*(InputFile)

begin

```
sf <- new ScanningFile(InputFile)
webpages <- new List()
while(NOT sf.EOF)
    webpages.Add ( Recognize in sf a Web page )

pi <- InitializePermutations(21)
lsh <- new LSH(7 , 3)
swp = new ScanningWebPages(InputFile)
categWP <- new Set(), notCategWP <- new Set()
foreach wp in webpages
{
    if (wp.IsHTML)
    {
        swp.WebPageContent(wp)
        wp.Signature <- CalculateSignature(swp.Container, pi)
        lsh.AddDocument(wp)
        categWP.Add(wp)
    }
    else
        notCategWP.Add(wp)
}
```

Calculate\_Permutation()

```
foreach 25K wp in webpages
{
    swp.WebPageContent(25K wp)
    sort swp.Container by NewStartPage
    write 25K wp in OutputFile
}
```

end

**Do not process entire page:**

We analyse only the “header” information and then we skip to the next pages (*Length of Page*)

# Proposed Solution – 2<sup>nd</sup> Version

**Algorithm:** *CompressingWebPages(InputFile)*

begin

```

sf <- new ScanningFile(InputFile)
webpages <- new List()
while(NOT sf.EOF)
    webpages.Add ( Recognize in sf a Web page )

pi <- InitializePermutations(21)
lsh <- new LSH(7 , 3)
swp = new ScanningWebPages(InputFile)
categWP <- new Set(), notCategWP <- new Set()
foreach wp in webpages
{
    if (wp.IsHTML)
    {
        swp.WebPageContent(wp)
        wp.Signature <- CalculateSignature(swp.Container, pi)
        lsh.AddDocument(wp)
        categWP.Add(wp)
    }
    else
        notCategWP.Add(wp)
}

```

Calculate\_Permutation()

```

foreach 25K wp in webpages
{
    swp.WebPageContent(25K wp)
    sort swp.Container by NewStartPage
    write 25K wp in OutputFile
}

```

end

## Scanning Web Pages:

Similar to *ScanningFile* but the **chunks** are made by **100K** pages for a time



# Proposed Solution – 2<sup>nd</sup> Version

**Algorithm:** *CompressingWebPages(InputFile)*

begin

```
sf <- new ScanningFile(InputFile)
webpages <- new List()
while(NOT sf.EOF)
    webpages.Add ( Recognize in sf a Web page )
```

```
pi <- InitializePermutations(21)
lsh <- new LSH(7 , 3)
swp = new ScanningWebPages(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
foreach wp in webpages
```

```
{
    if (wp.IsHTML)
    {
        swp.WebPageContent(wp)
        wp.Signature <- CalculateSignature(swp.Container, pi)
        lsh.AddDocument(wp)
        categWP.Add(wp)
    }
    else
        notCategWP.Add(wp)
}
```

```
Calculate_Permutation()
```

```
foreach 25K wp in webpages
```

```
{
    swp.WebPageContent(25K wp)
    sort swp.Container by NewStartPage
    write 25K wp in OutputFile
}
```

end

## Web Pages Content:

Depending on **wp**, we store the appropriate **chunk** in memory, and we **copy** the content of **wp** into the **swp.Container**, adapting it if necessary

# Proposed Solution – 2<sup>nd</sup> Version

**Algorithm:** *CompressingWebPages(InputFile)*

begin

```
sf <- new ScanningFile(InputFile)
webpages <- new List()
while(NOT sf.EOF)
    webpages.Add ( Recognize in sf a Web page )

pi <- InitializePermutations(21)
lsh <- new LSH(7 , 3)
swp = new ScanningWebPages(InputFile)
categWP <- new Set(), notCategWP <- new Set()
foreach wp in webpages
{
    if (wp.IsHTML)
    {
        swp.WebPageContent(wp)
        wp.Signature <- CalculateSignature(swp.Container, pi)
        lsh.AddDocument(wp)
        categWP.Add(wp)
    }
    else
        notCategWP.Add(wp)
}
```

Calculate\_Permutation()

```
foreach 25K wp in webpages
{
    swp.WebPageContent(25K wp)
    sort swp.Container by NewStartPage
    write 25K wp in OutputFile
}
```

end

## Calculate Signature:

Knowing exactly the web page content the process is faster and more precise respect the first version

# Proposed Solution – 2<sup>nd</sup> Version

**Algorithm:** *CompressingWebPages*(InputFile)

begin

```
sf <- new ScanningFile(InputFile)
webpages <- new List()
while(NOT sf.EOF)
    webpages.Add ( Recognize in sf a Web page )

pi <- InitializePermutations(21)
lsh <- new LSH(7 , 3)
swp = new ScanningWebPages(InputFile)
categWP <- new Set(), notCategWP <- new Set()
foreach wp in webpages
{
    if (wp.IsHTML)
    {
        swp.WebPageContent(wp)
        wp.Signature <- CalculateSignature(swp.Container, pi)
        lsh.AddDocument(wp)
        categWP.Add(wp)
    }
    else
        notCategWP.Add(wp)
}
```

Calculate\_Permutation()

← Same as before, but we have improved **LSH UnionFind**

```
foreach 25K wp in webpages
{
    swp.WebPageContent(25K wp)
    sort swp.Container by NewStartPage
    write 25K wp in OutputFile
}
```

end

# Proposed Solution – 2<sup>nd</sup> Version

Algorithm: *CompressingWebPages*(InputFile)

begin

```
sf <- new ScanningFile(InputFile)
webpages <- new List()
while(NOT sf.EOF)
    webpages.Add ( Recognize in sf a Web page )

pi <- InitializePermutations(21)
lsh <- new LSH(7 , 3)
swp = new ScanningWebPages(InputFile)
categWP <- new Set(), notCategWP <- new Set()
foreach wp in webpages
{
    if (wp.IsHTML)
    {
        swp.WebPageContent(wp)
        wp.Signature <- CalculateSignature(swp.Container, pi)
        lsh.AddDocument(wp)
        categWP.Add(wp)
    }
    else
        notCategWP.Add(wp)
}
```

Calculate\_Permutation()

```
foreach 25K wp in webpages
{
    swp.WebPageContent(25K wp)
    sort swp.Container by NewStartPage
    write 25K wp in OutputFile
}
```

end

**Sorting in memory 25K pages at a time:**  
In this way we can read and write sequentially

# Results – 1° & 2° phase

**Algorithm:** *CompressingWebPages*(InputFile)

begin

```
sf <- new ScanningFile(InputFile)
webpages <- new List()
while(NOT sf.EOF)
    webpages.Add ( Recognize in sf a Web page )
```

```
pi <- InitializePermutations(21)
lsh <- new LSH(7 , 3)
swp = new ScanningWebPages(InputFile)
categWP <- new Set(), notCategWP <- new Set()
foreach wp in webpages
{
    if (wp.IsHTML)
    {
        swp.WebPageContent(wp)
        wp.Signature <- CalculateSignature(swp.Container, pi)
        lsh.AddDocument(wp)
        categWP.Add(wp)
    }
    else
        notCategWP.Add(wp)
}
```

Calculate\_Permutation()

```
foreach 25K wp in webpages
{
    swp.WebPageContent(25K wp)
    sort swp.Container by NewStartPage
    write 25K wp in OutputFile
}
```

end

#Pages	Time 1°	Time 2°
1	00:00:00	00:00:00
10	00:00:02	00:00:06
100	00:00:14	00:00:42
500	00:02:29	00:05:57
1000	00:05:05	00:13:00
3000	00:15:00	00:36:35



# Results – 1° & 2° phase

**Algorithm:** *CompressingWebPages*(InputFile)

begin

```
sf <- new ScanningFile(InputFile)
webpages <- new List()
while(NOT sf.EOF)
    webpages.Add ( Recognize in sf a Web page )
```

```
pi <- InitializePermutations(21)
lsh <- new LSH(7 , 3)
swp = new ScanningWebPages(InputFile)
categWP <- new Set(), notCategWP <- new Set()
foreach wp in webpages
{
    if (wp.IsHTML)
    {
        swp.WebPageContent(wp)
        wp.Signature <- CalculateSignature(swp.Container, pi)
        lsh.AddDocument(wp)
        categWP.Add(wp)
    }
    else
        notCategWP.Add(wp)
}
```

Calculate\_Permutation()


```
foreach 25K wp in webpages
{
    swp.WebPageContent(25K wp)
    sort swp.Container by NewStartPage
    write 25K wp in OutputFile
}
```

end

## comparison between v1 & v2

#Pages	Version 1	Version 2
1	00:00:01	00:00:00
10	00:00:26	00:00:08
100	00:05:26	00:00:56
500	00:36:58	00:08:26
1000	00:57:16	00:18:05
3000	>> 2 days	00:51:35



	2% CPU	94% Memory	100% Disk
 CompressingWebPages	1,5%	2.971,6 MB	1,3 MB/s

# Results – 3° & 4° phase

**Algorithm:** *CompressingWebPages*(InputFile)

begin

```
sf <- new ScanningFile(InputFile)
webpages <- new List()
while(NOT sf.EOF)
    webpages.Add ( Recognize in sf a Web page )
```

```
pi <- InitializePermutations(21)
lsh <- new LSH(7 , 3)
swp = new ScanningWebPages(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
foreach wp in categWP
{
    simDoc <- sort lsh.UnionFind(wp) by URL
    calculate permutation order of simDoc
}
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc
```



**Calculate\_Permutation()**

```
foreach 25K wp in webpages
{
    swp.WebPageContent(25K wp)
    sort swp.Container by NewStartPage
    write 25K wp in OutputFile
}
```

end

#Pages	Version 1	Version 2
1	00:00:01	00:00:00
10	00:01:07	00:00:00
100	00:27:46	00:06:30
500	02:18:35	01:22:27
1000	05:29:42	04:34:51
3000	-	32:35:03

# Results – 3° & 4° phase

**Algorithm:** *CompressingWebPages*(InputFile)

begin

```
sf <- new ScanningFile(InputFile)
webpages <- new List()
while(NOT sf.EOF)
    webpages.Add ( Recognize in sf a Web page )
```

```
pi <- InitializePermutations(21)
lsh <- new LSH(7 , 3)
swp = new ScanningWebPages(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
foreach wp in categWP
{
    simDoc <- sort lsh.UnionFind(wp) by URL
    calculate permutation order of simDoc
}
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc
```



**Calculate\_Permutation()**

```
foreach 25K wp in webpages
{
    swp.WebPageContent(25K wp)
    sort swp.Container by NewStartPage
    write 25K wp in OutputFile
}
```

end

## comparison between v1 & v2

#Pages	Time 3°	Time 4°
1	00:00:00	00:00:00
10	00:00:00	00:00:00
100	00:01:13	00:05:17
500	00:27:34	00:54:53
1000	03:14:25	01:26:26
3000	25:07:45	07:27:18

# Results – compression

# Pages	Original Compression	Achieved Compression
1	89.75%	99.79%
10	91.41%	93.00%
100	93.53%	95.08%
500	94.57%	96.17%
1000	94.25%	95.99%
3000	93.83%	95.69%

# Future Work

- **Parallelize:** Divides *#pages* in group depending *#CPU*
  - Computes **signature** of each group in **parallel**
  - After found the final permutation, in **parallel**, reads each group of pages and write them into the output file, accordingly the permutation

**Assumption:**

If someone **COPY** some TAGs / WORDs / SCRIPTs from an HTML page, he will **NOT MODIFY** them.

- **Remove initial assumption:**
  - Collects from the **shingle 3** groups of 7 random characters each (**85%**)
  - Compute **Karp-Rabin** hashing of each group → **Sketch  $21 \times 3$**

- **Improve Algorithms:**
  - Union Find → the **3<sup>th</sup> phase** is too slow despite there are no *I/Os*
  - Sorting → use **Multi-Way MergeSort** (**4<sup>th</sup> phase**)



# Questions



***Thanks !***