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

20th March 2015

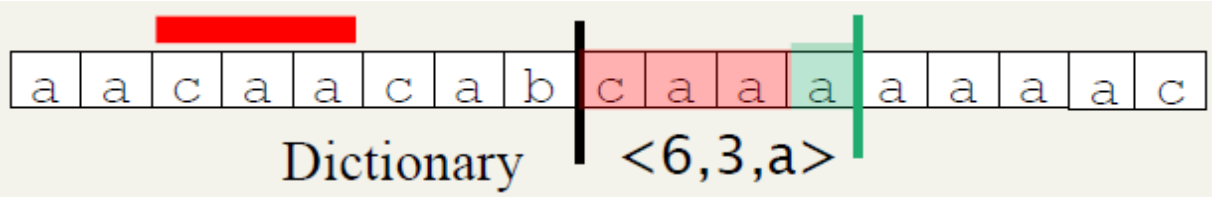
Problem Statement

INPUT: A file containing a collection of Web pages

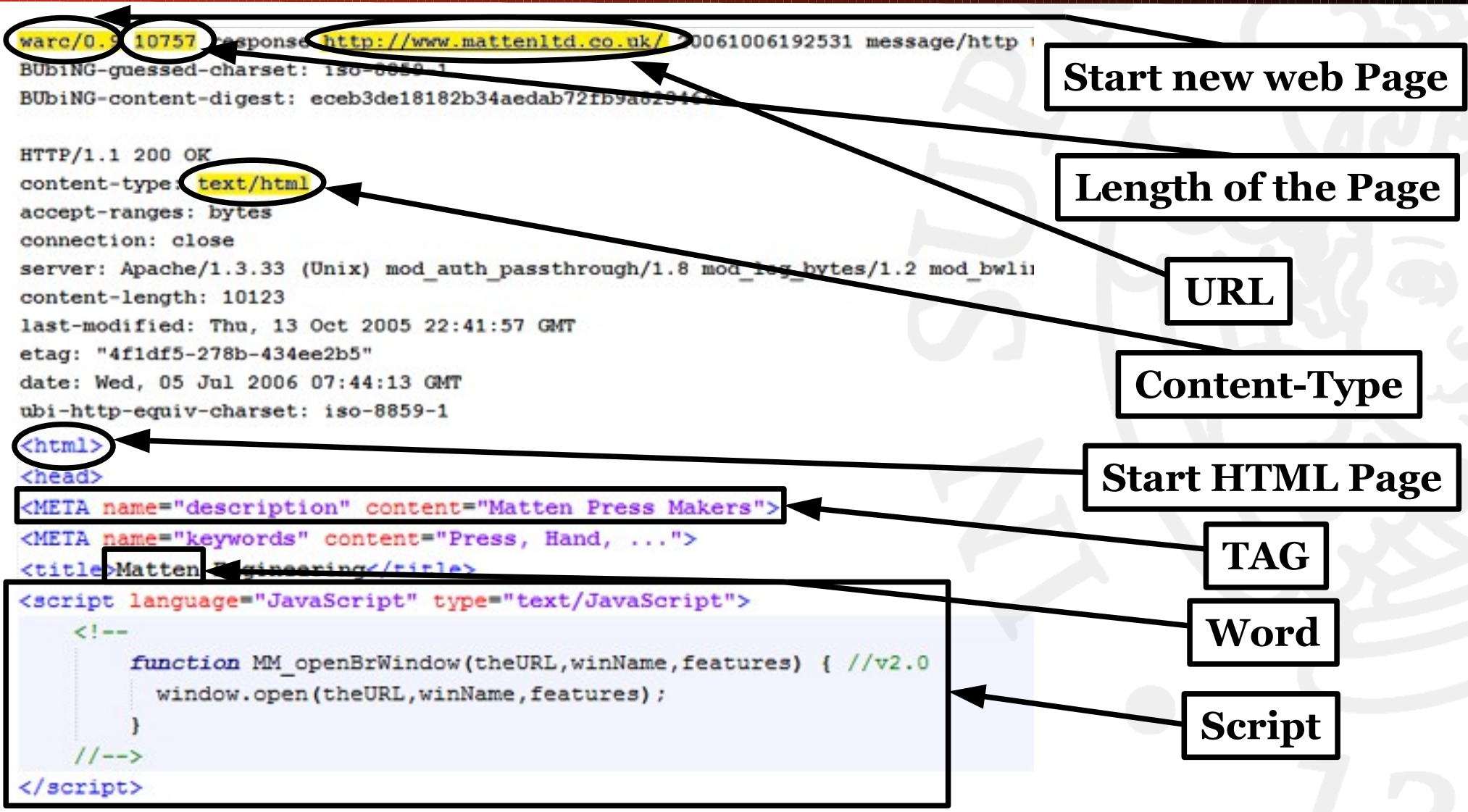
OUTPUT: A file containing the permuted collection, where the permutation is driven by the similarity between pages

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

LZ77



Analysis of the Web pages



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

```
<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 HTML Page

TAG

Word

Script

Proposed Solution

```
Algorithm: CompressingWebPages(InputFile)
begin
    pi <- CalculatePermutations(20)
    lsh <- new LSH(b , r)
    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
```

Min-Hashing: *used to permute the* ***fingerprints***
Initialize 20 **MURMUR HASH** *function*

Proposed Solution

```
Algorithm: CompressingWebPages(InputFile)
begin
    pi <- CalculatePermutations(20)
    lsh <- new LSH(b , r)
    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* ***r*** *elements from* ***Sketch***
- *create* ***b*** *hash tables with chaining*

Proposed Solution

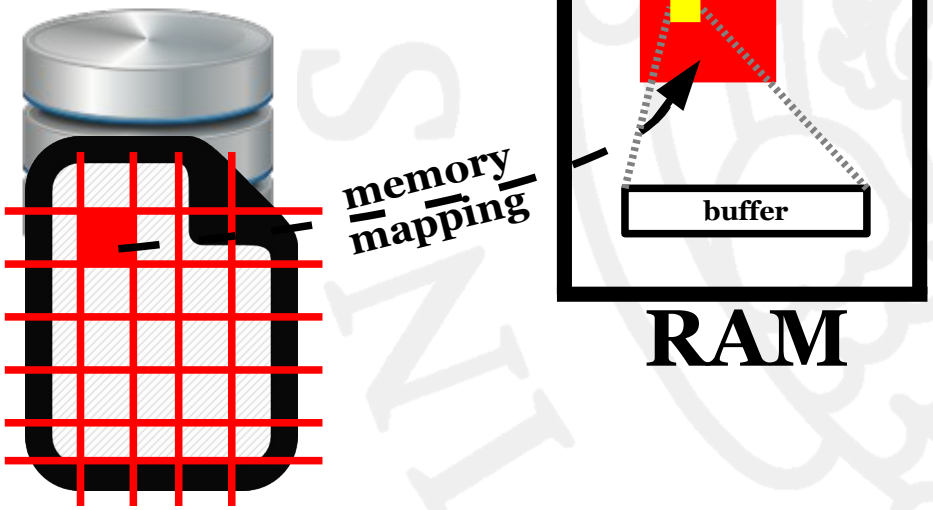
```
Algorithm: CompressingWebPages(InputFile)
begin
    pi <- CalculatePermutations(20)
    lsh <- new LSH(b , r)
    sf <- new ScanningFile(InputFile)
    categWP <- new Set(), notCategWP <- new Set()

    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:



Proposed Solution

```
Algorithm: CompressingWebPages(InputFile)
begin
    pi <- CalculatePermutations(20)
    lsh <- new LSH(b , r)
    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

```
Algorithm: CompressingWebPages(InputFile)
begin
    pi <- CalculatePermutations(20)
    lsh <- new LSH(b , r)
    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 **37** buckets*
- *if there are collision → we list them*

Threshold ~30%

Proposed Solution

```
Algorithm: CompressingWebPages(InputFile)
begin
    pi <- CalculatePermutations(20)
    lsh <- new LSH(b , r)
    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*

Proposed Solution

```
Algorithm: CompressingWebPages(InputFile)
begin
    pi <- CalculatePermutations(20)
    lsh <- new LSH(b , r)
    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***

Results – 1 ° phase

```
Algorithm: CompressingWebPages(InputFile)
begin
    pi <- CalculatePermutations(20)
    lsh <- new LSH(b , r)
    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
```

Dimension (K)	Time
1	00:00:01
10	00:00:26
100	00:05:26
3000	00:00:01

Results – 2 ° phase

```
Algorithm: CompressingWebPages(InputFile)
begin
    pi <- CalculatePermutations(20)
    lsh <- new LSH(b , r)
    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
```

Dimension (K)	Time
1	00:00:02
10	00:01:07
100	**00:27:46**
3000	00:00:01

Results – compression

Dimension (K)	Original Compression	Achieved Compression
1	89.75%	89.83%
10	91.41%	91.97%
100	93.53%	94.64%
3000	93.83%	95.00%

Future Work

- **Parallelize 1° phase:**
  - File Scanning: find ***#pages*** and ***start*** & ***length*** of each page
  - Divides ***#pages*** in group depending ***#CPU***
  - Computes **signature** of each group in **parallel**
- **Save data of Web pages on file:**
  - Avoids stressing the memory
  - Collects information when needed
- **Parallelize 2° phase:**
  - After found the final permutation
  - In **parallel**, reads each group of pages and write them into the output file, accordingly the permutation