EECS 440 System Design of a Search Engine

Winter 2021

Lecture 17: Shingles

Nicole Hamilton
https://web.eecs.umich.edu/~nham/
nham@umich.edu

Agenda

1. Course details.
2. Shingles.

Agenda

1. Course details.
2. Shingles.

details

1. HashBlob. All but two teams finished with 180/180. I was surprised that most teams placed an unnecessary dummy SerialTruple sentinel at the end of every chain.
2. LinuxTinyServer assignment posted. Expression parser is optional.
3. Allowed features pleadings are a little tedious. I really can’t stop you from doing whatever you want but I’m hoping you’ll write as much as possible yourself as part of the learning experience and because it’s a somewhat competitive assignment and these are the “Pinewood Derby”rules.
4. Group presentations Apr 19 & 21.

Agenda

1. Course details.
2. Shingles.

Basic problem

The web is full of duplicate content.

What you don’t want in a search engine is for the top 10 results to all be the same page.

Very easy to identify exact duplicates by comparing checksums, e.g., CRCs, of the files.

But most of the duplicates are only near duplicates, e.g., a last modified date.

Straightforward algorithm

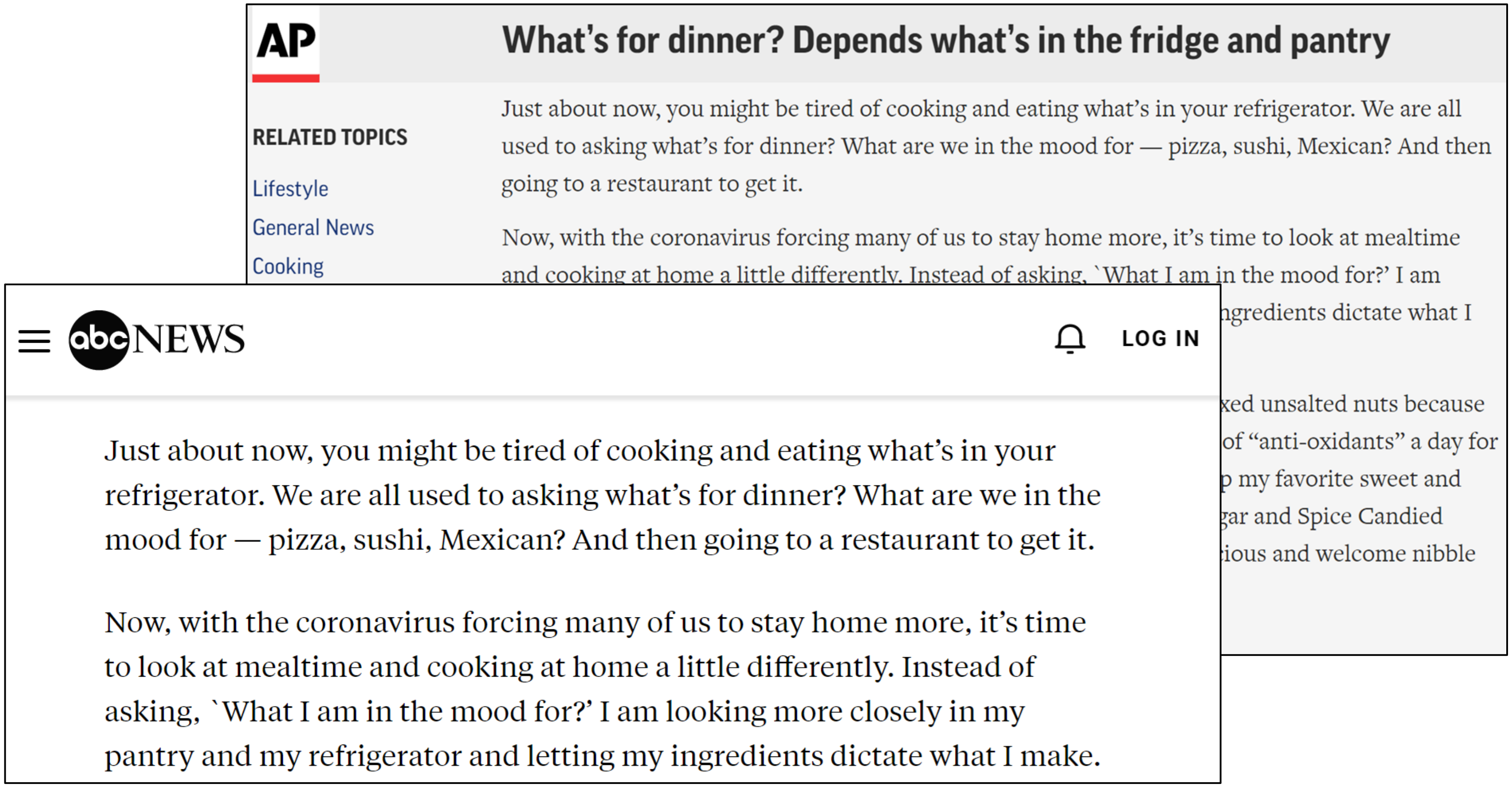
Hash of a document
- Hash("It grows in dry places...") = 8d2f908...

When crawling a new document
- Compute hash
- Look up in hash table
  - If present, it's a duplicate
  - If not, put in the hash table

How many comparisons for N web pages?
- O(N) = O(N) lookups at O(1) per lookup

Why not compare entire document?

Documents can be similar but not exactly the same



AP

What's for dinner? Depends what's in the fridge and pantry

RELATED TOPICS

Lifestyle

General News

Cooking

Just about now, you might be tired of cooking and eating what's in your refrigerator. We are all used to asking what's for dinner? What are we in the mood for — pizza, sushi, Mexican? And then going to a restaurant to get it.

Now, with the coronavirus forcing many of us to stay home more, it's time to look at mealtime and cooking at home a little differently. Instead of asking, `What I am in the mood for?' I am

ngredients dictate what I

xed unsalted nuts because

of "anti-oxidants" a day for

p my favorite sweet and

gar and Spice Candied

ious and welcome nibble

abc NEWS

LOG IN

Just about now, you might be tired of cooking and eating what's in your refrigerator. We are all used to asking what's for dinner? What are we in the mood for — pizza, sushi, Mexican? And then going to a restaurant to get it.

Now, with the coronavirus forcing many of us to stay home more, it's time to look at mealtime and cooking at home a little differently. Instead of asking, `What I am in the mood for?' I am looking more closely in my pantry and my refrigerator and letting my ingredients dictate what I make.

Stripping “chrome”

One strategy is to try to strip out “chrome”, e.g., HTML tags, headers, footers and sidebars.

But still a question of how to quantify similarity.

3.1.1 Jaccard Similarity of Sets

The *Jaccard similarity* of sets S and T is $|S \cap T|/|S \cup T|$, that is, the ratio of the size of the intersection of S and T to the size of their union. We shall denote the Jaccard similarity of S and T by $\text{SIM}(S, T)$.

Example 3.1 : In Fig. 3.1 we see two sets S and T. There are three elements in their intersection and a total of eight elements that appear in S or T or both. Thus, $\text{SIM}(S, T) = 3/8$. □

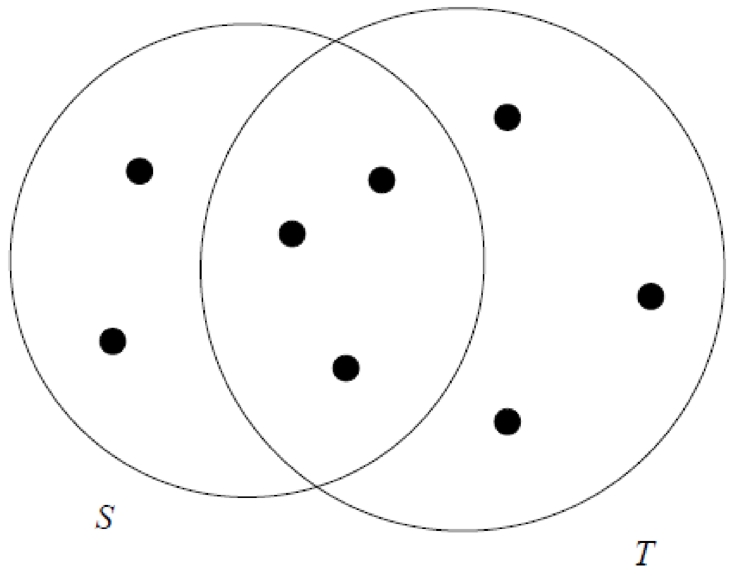


Figure 3.1: Two sets with Jaccard similarity 3/8

Source: http://infolab.stanford.edu/~ullman/mmds/ch3.pdf

k-Shingles

Consider a document as a string of characters or words.

Define a *k*-shingle as any substring of length *k* found in the document.

Suppose our document D is the string `abcdabd`, and we pick *k* = 2. Then the set of 2-shingles for D is `{ab, bc, cd, da, bd}`.

Picking shingle size

k can be any constant you like.

If it's too small, then most sequences will appear in most documents, creating false positives.

k should be large enough that the probability of any given shingle appearing in any given document is low.

k often chosen between 5 and 9 words or perhaps 25 to 50 characters.

Shingles are hashed

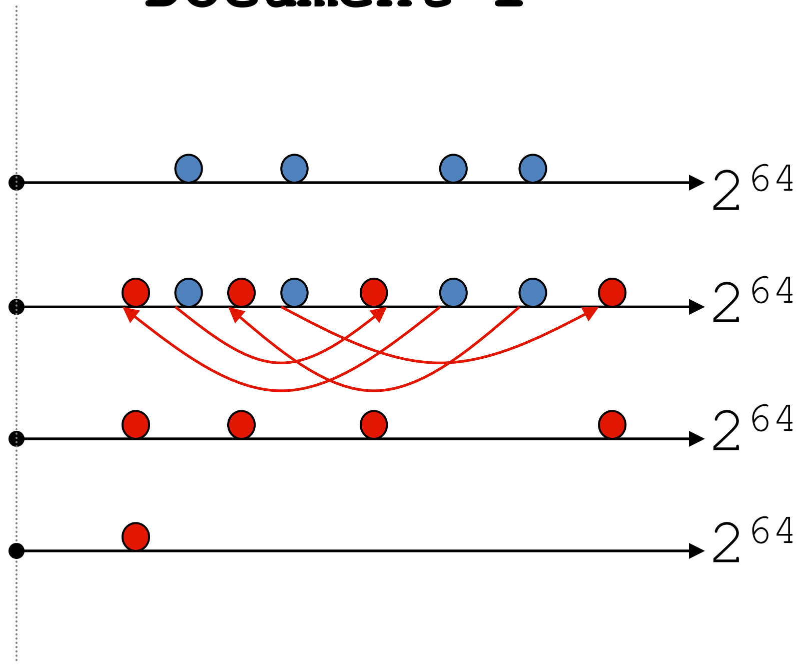
Shingles are then hashed.

Bits are transposed randomly in the same way for all hashes.

Smallest result is saved in a hash table.

To look for a duplicate, look for a match in the hash table.

Document 1

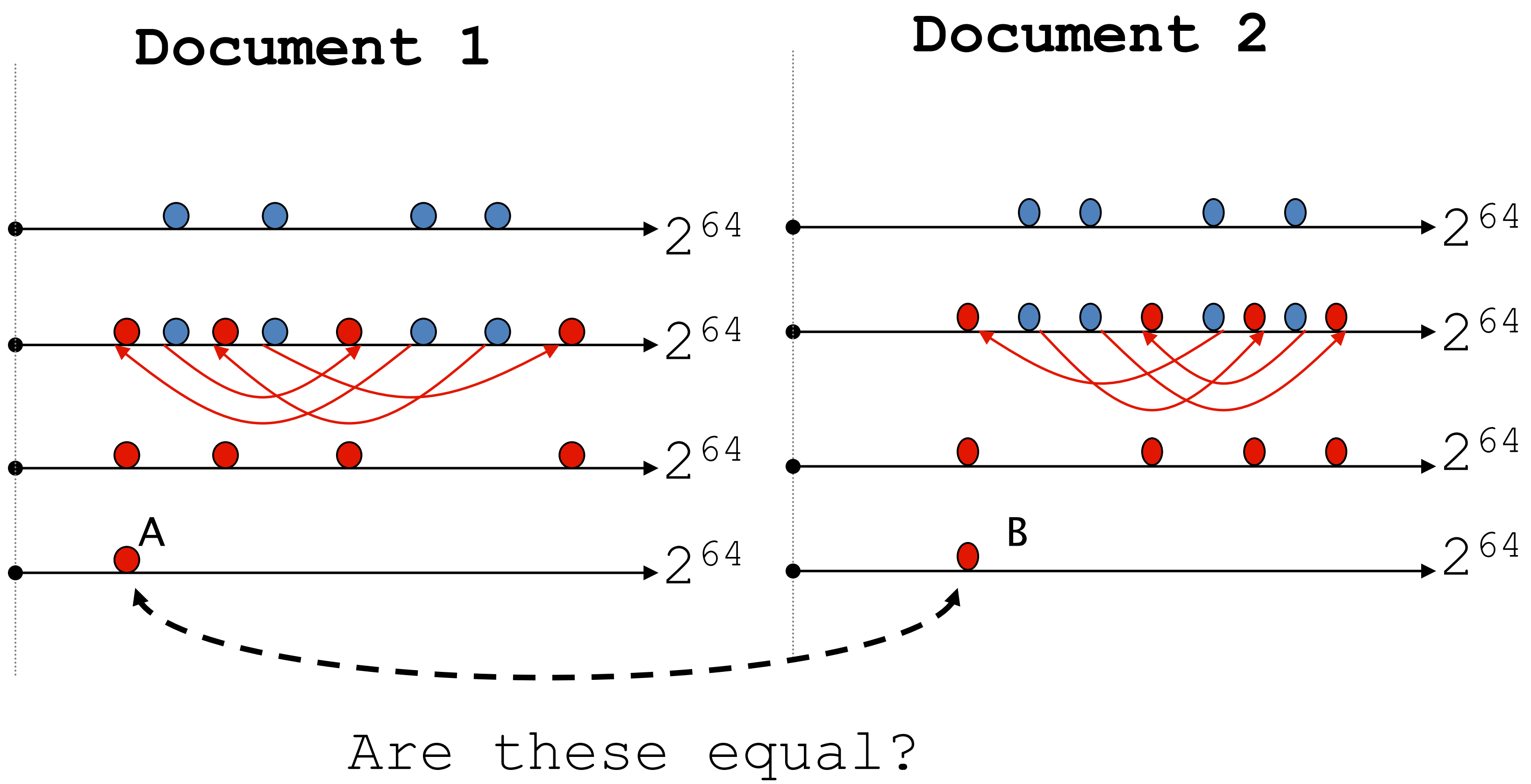


Start with 64-bit *f*(shingles)

Permute on the number line with π_i

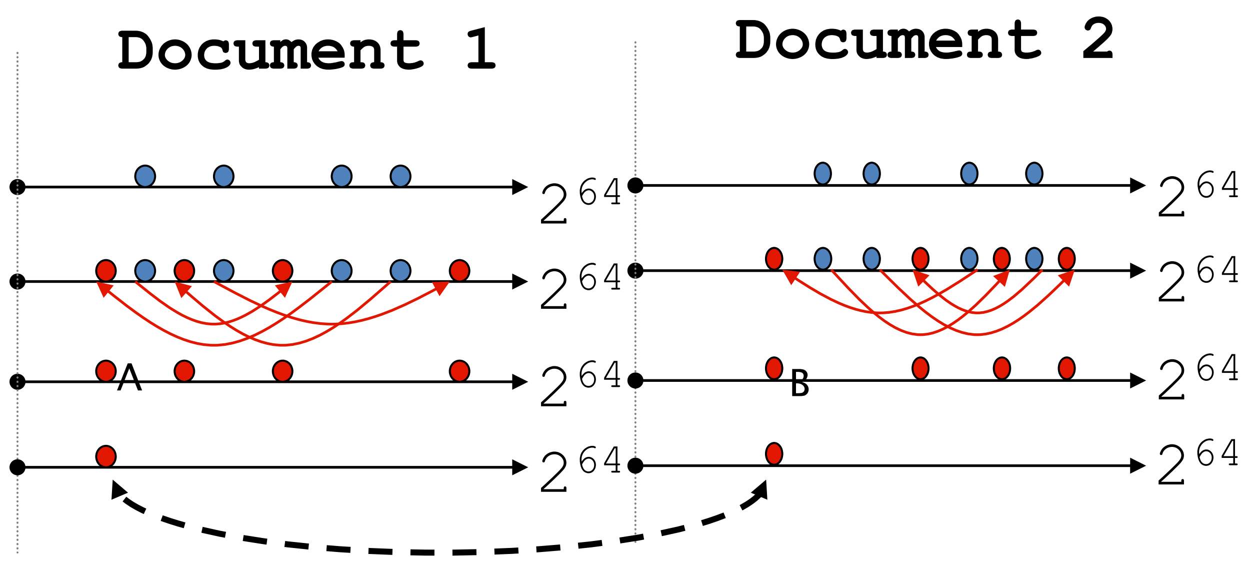
Pick the min value

Test if Doc1.Sketch[i] = Doc2.Sketch[i]



Test for 200 random permutations: $\pi_1, \pi_2, \ldots \pi_{200}$

However…



A = B iff the shingle with the MIN value in the union of Doc1 and Doc2 is common to both (i.e., lies in the intersection)

Claim: This happens with probability

Size_of_intersection / Size_of_union

Observations

Transforms the problem from comparing various sizes of text to comparing 64-bit integers.

But it’s still largely heuristic because you still have to decide how big the shingles should be, whether to keep duplicates and how many shingles to keep for each document.

Unclear why there’s an advantage to doing “200 random permutations” (out of 64! = 1.27e+089 possible) of what are already basically random numbers representing the hashes of all the shingles.

To investigate

I’ve written and uploaded two C++ programs that can memory map a file and calculate a sorted list shingles for comparison.

LinuxShingle.cpp

WindowsShingle.cpp

```
10 C% WindowsShingle.exe
Usage:  Shingle [-d] <shingleSize> <stepSize> <N> <filename>
-d  Report duplicates. By default, only unique shingles arereported
shingleSize = number of characters in a shingle
shingleStep = how many characters to step from one shingle to next
N = number of shingles to report.  If N = -1, all shingles reported
11 C%
```

```
11 C% wc -l LinuxGet{Url,Ssl}.cpp
      188  LinuxGetUrl.cpp
      225  LinuxGetSsl.cpp
      413  total
12 C% diff !$ | head
diff LinuxGet{Url,Ssl}.cpp | head
1c1,3
< // Linux get URL utility that copies the HTTP page to stdout.
---
> // Linux SSL get SSL utility that copies the HTTPS page to stdout
> // using OpenSSL
>
3a6,17
> // Required for build under Cygwin:
> //    openssl
> //    libssl-devel
13 C% diff LinuxGet{Url,Ssl}.cpp | grep '^^[><]' | wc -l
       55
14 C% @ union = 413
15 C% @ difference = 55
16 C% calc percentOverlap = (union - difference)/union
0.866828
18 C%
```

```
101 C% WindowsShingle.exe 25 5 10 linuxGetUrl.cpp
118989564705677488
118993962752190332
130609402609293028
130703960609319174
168981396494694060
177539890157274494
184314166939961877
186826603892655687
195933926610954092
220539590904835157
102 C%
```

```
103 C% WindowsShingle.exe 25 5 10 linuxGetSsl.cpp
4092356117376548O
6052952209442206
118989564705677488
118993962752190332
130609402609293028
168981396494694060
177539890157274494
184314166939961877
220539590904835157
236557822413782588
104 C%
```

7 out of 10 shingles are shared.

```
104 C% set a = LinuxGetUrl.cpp
105 C% set b = LinuxGetSsl.cpp
106 C% WindowsShingle.exe 25 5 10 $a > $a.shingle
107 C% WindowsShingle.exe 25 5 10 $b > $b.shingle
108 C% di {$a,$b}.shingle
4092356117376548O
60529522094422068
118989564705677488
118993962752190332
130609402609293028
130703960609319174
168981396494694060
177539890157274494
184314166939961877
186826603892655687
195933926610954092
220539590904835157
236557822413782588
109 C%
```

Use a script to compare diffs and shingles.

```
110 C% similarity.csh 25 5 10 LinuxGet{Url,Ssl}.cpp
a = LinuxGetSsl.cpp
b = LinuxGetUrl.cpp
25 5 10
diffunion = 413
diffintersection = 358
diffsimilarity = 86.682809
shingleunion = 20
shingleintersection = 14
shinglesimilarity = 70
111 C%
```

Too small a number of shingles can generate a false match.

```
123 C% similarity.csh 25 1 1 LinuxGetUrl.cpp LinuxTinyServer.cpp
a = LinuxTinyServer.cpp
b = LinuxGetUrl.cpp
25 1 1
diffunion = 475
diffintersection = 96
diffsimilarity = 20.210526
shingleunion = 2
shingleintersection = 2
shinglesimilarity = 100
124 C%
```

Tune the heuristics.

```
124 C% similarity.csh 25 5 50 LinuxGetUrl.cpp LinuxTinyServer.cpp
a = LinuxTinyServer.cpp
b = LinuxGetUrl.cpp
25 5 50
diffunion = 475
diffintersection = 96
diffsimilarity = 20.210526
shingleunion = 100
shingleintersection = 22
shinglesimilarity = 22
125 C%
```

Tune the heuristics.

```
125 C% similarity.csh 25 5 50 LinuxGetUrl.cpp LinuxWcMap.cpp
a = LinuxWcMap.cpp
b = LinuxGetUrl.cpp
25 5 50
diffunion = 260
diffintersection = 33
diffsimilarity = 12.692308
shingleunion = 100
shingleintersection = 8
shinglesimilarity = 8
126 C%
```