

EECS 498 System Design of a Search Engine

Winter 2026

Lecture 18: Sitemaps and Shingles

Nicole Hamilton

[https://web.eecs.umich.edu/~nham/
nham@umich.edu](https://web.eecs.umich.edu/~nham/nham@umich.edu)

agenda

1. details
2. tee shirts
3. finish ranking
4. sitemaps
5. shingles
6. instrumenting memory usage
7. exception handling.

agenda

1. details
2. tee shirts
3. finish ranking
4. sitemaps
5. shingles
6. instrumenting memory usage
7. exception handling.

details

1. This is the last technical lecture.
2. Lab this week will be discussion of team strategies.
3. Team presentations and demos.
 Thursday April 16 (4 weeks left!)
 Friday April 17
 Tuesday April 21 (last day of class)
4. Team meetings April 22 to April 28.
5. Final Exam April 29, 4:00 pm.

agenda

1. details
2. tee shirts
3. finish ranking
4. sitemaps
5. shingles
6. instrumenting memory usage
7. exception handling.

Orders will open tomorrow, stay open until next week.
Price will be \$25. We should get them two weeks later.



agenda

1. details
2. tee shirts
3. finish ranking
4. sitemaps
5. shingles
6. instrumenting memory usage
7. exception handling.

basic strategy

Choose rarest word as an anchor and advance that ISR through each occurrence in the document.

At each occurrence of the rarest word, advance the other ISRs to position them as close as possible to the desired position in the flattened list.

Requires only one stage of look-ahead.

spans

1. If there is at least one occurrence of each query term, that's a "span" which can be further distinguished as ordered, short, an exact phrase, etc.
2. Only the rarest word is guaranteed to a unique occurrence.
3. The other words may be reused.
4. Also count doubles and triples, meaning various combinations of just 2 or 3 of the words in the query, one of which must be the rarest.
5. Parameterized threshold for short vs. long spans, frequent vs. infrequent, etc.
6. Most features are binary and either occur or do not occur unambiguously.

Example phrase query "quick brown fox".

What I did was *pick the rarest word* and *then arrange the other ISRs* to as close as possible to their desired locations in the query relative to each occurrence of the rarest word.

Word counts:

quick	10
brown	14
fox	7

This reduces the number of combinations to be scored in this example from $10 \times 14 \times 7 = 957$ to 7 and can be done a single pass with a single stage of lookahead.

Call each combination a *span*.

quick	brown	fox
62	83	284
69	94	423
84	170	580
311	179	612
421	216	796
430	227	912
559	400	958
619	417	
794	422	
952	516	
	795	
	826	
	828	
	957	

Example phrase query "quick brown fox".

Iterate over the rarest word hits, arranging the other ISRs to as close as they could be to their desired locations in the query. Here are the first 3 *spans*.

brown			brown			brown		
	83			83			83	
quick	94			94			94	
62	170			170		quick	170	
69	179		quick	179		62	179	
84	216	fox	62	216		69	216	
311	227	284	69	227		84	227	
421	400	423	84	400	fox	311	400	fox
430	417	580	311	417	284	421	417	284
559	422	612	421	422	423	430	422	423
619	516	796	430	516	580	559	516	580
794	795	912	559	795	612	619	795	612
952	826	958	619	826	796	794	826	796
	828		794	828	912	952	828	912
	957		952	957	958		957	958

Example phrase query "quick brown fox".

For each combination, decide if it's an *exact phrase* or all the words *in order*, or *close together*, incrementing an associated counter. Shown here, there is one exact phrase but none of the rest are in order or particularly close together.

brown			brown			brown		
	83			83			83	
quick	94			94			94	
62	170			170		quick	170	
69	179		quick	179		62	179	
84	216	fox	62	216		69	216	
311	227	284	69	227		84	227	
421	400	423	84	400	fox	311	400	fox
430	417	580	311	417	284	421	417	284
559	422	612	421	422	423	430	422	423
619	516	796	430	516	580	559	516	580
794	795	912	559	795	612	619	795	612
952	826	958	619	826	796	794	826	796
	828		794	828	912	952	828	912
	957		952	957	958		957	958

Example phrase query "quick brown fox".

	brown	
	83	
quick	94	
62	170	
69	179	
84	216	
311	227	fox
421	400	284
430	417	423
559	422	580
619	516	612
794	795	796
952	826	912
	828	958
	957	

	brown	
	83	
quick	94	
62	170	
69	179	
84	216	
311	227	
421	400	fox
430	417	284
559	422	423
619	516	580
794	795	612
952	826	796
	828	912
	958	
	957	

	brown	
	83	
	94	
	170	
quick	179	
62	216	
69	227	
84	400	
311	417	
421	422	fox
430	516	284
559	795	423
619	826	580
794	828	612
952	957	796
		912
		958

Example phrase query "quick brown fox".

This last one is *close together* (say, less than 10 words apart) and *in order* but it's *not an exact phrase*.

	brown	
	83	
	94	
	170	
quick	179	
62	216	
69	227	
84	400	
311	417	fox
		284
421	422	423
430	516	580
559	795	612
619	826	796
794	828	912
952	957	958

Example phrase query "quick brown fox".

Set some thresholds and accumulate some counts, which can be scored at the end.

Thresholds	Values
Max to be short	10
Min to be frequent	?
Min to be most	?
Min to be near the top	?

Heuristic	Count	Weight	Score
Number of short spans	1	5	5
Number of in order spans	1	2	2
Exact phrases	2	10	20
Number of spans near the top	?	?	?
All word are frequent	?	?	?
Most words are frequent	?	?	?
Some words are frequent	?	?	?

Total dynamic rank score ?

brown		
	83	
	94	
	170	
quick	179	
62	216	
69	227	
84	400	
311	417	fox
421	422	284
430	516	423
559	795	580
619	826	612
794	828	796
952	957	912
958		

Example phrase query "quick brown fox".

One reason to decorate words as *anchor*, *URL*, *title* or *body* is so they can be separated in to separate *streams* (separate sets of ISRs) and *scored separately* with the same algorithm but different scoring parameters.

<i>Stream</i>	<i>Weight</i>	<i>Score</i>
Anchor	?	?
URL	?	?
Title	?	?
Body	?	?
<i>Total dynamic rank score</i>		?

Each of the streams can start at location 1 relative to the start of the document. The document length is the length of the longest stream.

Not all streams may contain all the words.

	brown	
	83	
	94	
	170	
quick	179	
62	216	
69	227	
84	400	
311	417	fox
421	422	284
430	516	423
559	795	580
619	826	612
794	828	796
952	828	912
957		958

Example phrase query "quick brown fox".

We do a similar heuristic calculation of the *static rank*, the quality of the page *independent of the query*.

<i>Heuristic</i>	<i>Weight</i>	<i>Score</i>
Short title	?	?
Nice document length	?	?
Short URL	?	?
Lots of anchor text	?	?
edu/gov/com/etc domain	?	?
PageRank if known	?	?
<i>Total static rank score</i>		?

	brown	
	83	
	94	
	170	
quick	179	
62	216	
69	227	
84	400	
311	417	fox
421	422	284
430	516	423
559	795	580
619	826	612
794	828	796
952	957	912
		958

Example phrase query "quick brown fox".

Combine static and dynamic rank to get a final score.

<i>Component</i>	<i>Weight</i>	<i>Score</i>
Static rank	?	?
Dynamic rank	?	?
<i>Total rank score</i>		<i>?</i>

The result is a *linear combination* of features. We're just adding them up.

You should be able to achieve reasonably good results with a reasonably small number of heuristics and simple *hand-tuning* starting from some rough guesses at, e.g., the relative importance of an exact phrase versus lots of individual hits.

	brown	
	83	
	94	
	170	
quick	179	
	62	216
	69	227
	84	400
	311	417
	421	422
	430	516
	559	795
	619	826
	794	828
	952	912
	957	958
	fox	
	284	
	423	
	580	
	612	
	796	

The actual score is calculated as a linear combination of features, which may be thought of as:

$$R = \sum C_i(Q) * A_i(P, F_i) * S_i(F_i)$$

Where:

R = Overall Rank

Q = The Query and its characteristics, e.g., the number of rare vs. common words in the query.

P = The Page and its characteristics, e.g., the number of words in the URL or title.

F_i = An arbitrary feature observation, e.g., an exact phrase in the title, or a raw value, e.g., raw PageRank.

S_i = Scaling for feature F_i from the raw number space of the feature into a nominal 0.0 .. 1.0 range.

A_i = Attribute scaling for feature F_i, possibly dependent on the characteristics of the page P.

C_i = Coefficient for feature F_i, depending on the characteristics of the query Q.

F_i = E_i(Q, P, T)

steps to ranking

1. Decide what information to collect and how to measure it.
2. Decide how to measure the quality of the result.
3. Pick a method for scoring a page based on the inputs.
4. Tune the system by testing it on sample queries and adjusting parameters, collecting more information or changing the scoring algorithm.

measuring quality

At Microsoft, initially, I just ran queries and eyeballed the results and fiddled with the parameters.

Results were surprisingly good.

measuring quality

We were also collecting labeled pages. We scraped queries and results from several engines and then paid people to rate results on a 0 to 5 scale.

- 5 Definitive result. Unlikely any other page could be a better result for this query, e.g., whitehouse.gov for "whitehouse".
- :
- 0 Completely deleterious result, e.g., porno, spam, phishing.

This is similar to what Google is doing with their [Page Quality Rating Guidelines](#), creating specificity in the ratings.

Altogether, I think we had about 50K queries and about 150K labeled results.

measuring quality

Next came a tool allowing us to do side-by-side comparison of results with current best parameters in a frame on the left and with a set you could tweak on the right.

If the pages had been labeled, it reported the quality and attempted to score the resulting set.

We ran a competition to see who could come up with the best settings.

measuring quality

Finally, we added a tuner that could adjust the parameters mechanically by gradient descent: Tweak an individual parameter rerun all the searches and measure whether the results got better (more highly ranked pages for each query.)

But we had lots of problems in the methodology of what to do with unlabeled pages.

measuring quality

Initially unlabeled pages were assumed to be “average”. Later, a “promising proximity” heuristic was added to the tuner bump the estimate for unlabeled pages if the search words were found close together.

The effect was to tune my complex ranker to behave like the tuner’s naïve ranker.

Better strategy would have been to only tune based on labeled results, discarding any unlabeled results or construct an index that only contained labeled pages.

generating labeled results

You can't afford to pay people to label results but you might assume that Google is pretty good, so you might simply try to get pages in the same order as Google.

Your ranker won't be as good as Google's, so it might not be able to discern the difference between the top results on the first page, but perhaps it could distinguish between results on the first page and results on the tenth page.

strategy

1. Create a list of popular short queries.
2. Scrape Google for first 10 pages of results and label them.
3. Create an index of only those pages.
4. Make some initial estimates of the weights of your heuristics.
5. While you're still making progress:
 - a. Run each query and count how many you got right.
 - b. Iterate over each heuristic, adjusting it up or down a little to see if there's any improvement.
 - c. Pick the best as a new starting point.

remarks

It is much easier to tune weights for binary features than for those encoding a range of “goodness”.

It is difficult to estimate the improvement you’ll get from a new feature without actually building it. A lot of it is intuition.

remarks

Think about what “blindspot” is created by not having the new proposed features.

1. When would the ranker not be able to tell the difference between two pages it should be able to distinguish?
2. How egregious would that be and how often would it occur?

What is your ranker getting wrong right now and would a simple heuristic fix it?

This kind of thinking led us to add features like “exact match” for the title.

remarks

Mechanically tuned parameter weights can be counter-intuitive if some of your heuristics overlap.

You can get a negative weight for something that should be good because it overlaps with something better.

The importance of anchor text made Googlebombs possible (and at Microsoft, we checked that they worked on our engine, too.)



[Web](#) [Images](#) [Groups](#) [News](#) [Froogle](#) [Local](#) [more »](#)

miserable failure

Search

[Advanced Search](#)
[Preferences](#)

Web

Results **1 - 10** of about **969,000** for [miserable failure](#). (0.06 seconds)

[Biography of President George W. Bush](#)

Biography of the president from the official White House web site.

www.whitehouse.gov/president/gwbbio.html - 29k - [Cached](#) - [Similar pages](#)

[Past Presidents](#) - [Kids Only](#) - [Current News](#) - [President](#)

[More results from www.whitehouse.gov »](#)

[Welcome to MichaelMoore.com!](#)

Official site of the gadfly of corporations, creator of the film Roger and Me and the television show The Awful Truth. Includes mailing list, message board, ...

www.michaelmoore.com/ - 35k - Sep 1, 2005 - [Cached](#) - [Similar pages](#)

[BBC NEWS | Americas | 'Miserable failure' links to Bush](#)

Web users manipulate a popular search engine so an unflattering description leads to the president's page.

news.bbc.co.uk/2/hi/americas/3298443.stm - 31k - [Cached](#) - [Similar pages](#)

[Google's \(and Inktomi's\) Miserable Failure](#)

A search for **miserable failure** on Google brings up the official George W.

Bush biography from the US White House web site. Dismissed by Google as not a ...

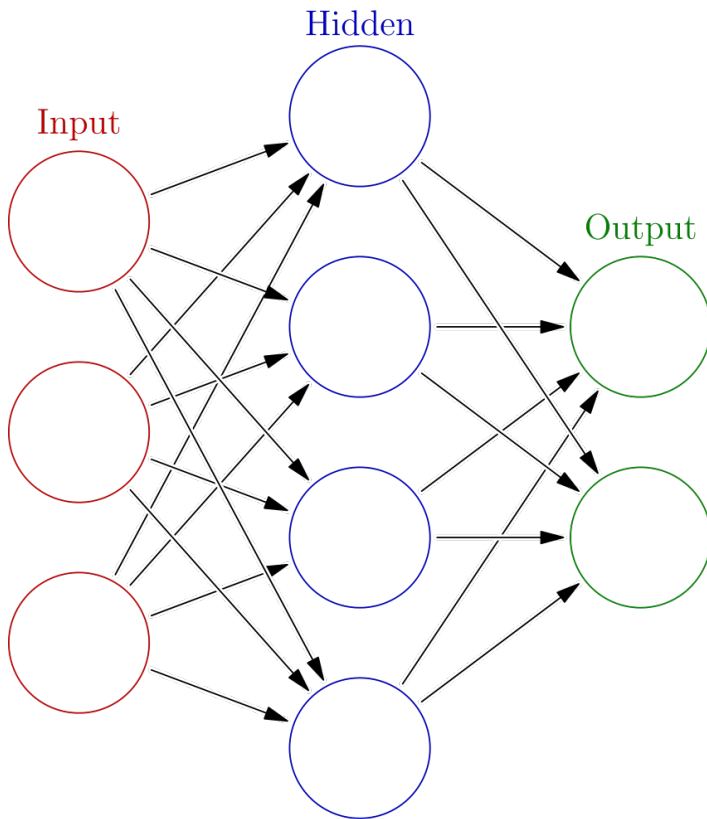
searchenginewatch.com/sereport/article.php/3296101 - 45k - Sep 1, 2005 - [Cached](#) - [Similar pages](#)

how to do better

To do better than heuristics, you will probably need a neural network.

Example phrase query "quick brown fox".

A machine learning strategy would be to simply collect all the lists and pass everything to a neural network, which must be trained. Beyond the scope here.



quick	brown	fox
62	83	284
69	94	423
84	170	580
311	179	612
421	216	796
430	227	912
559	400	958
619	417	
794	422	
952	516	
	795	
	826	
	828	
	957	

agenda

1. details
2. tee shirts
3. finish ranking
4. sitemaps
5. shingles
6. instrumenting memory usage
7. exception handling.

sitemaps

A sitemap is a file that lists all important pages, videos, and files on a website, acting as a roadmap for search engines to find, crawl, and index content efficiently.

sitemap format

The Sitemap protocol format consists of XML tags. All data values in a Sitemap must be *entity-escaped*. The file itself must be UTF-8 encoded.

The Sitemap must:

1. Begin with an opening *<urlset>* tag and end with a closing *</urlset>* tag.
2. Specify the namespace (protocol standard) within the *<urlset>* tag.
3. Include a *<url>* entry for each URL, as a parent XML tag.
4. Include a *<loc>* child entry for each *<url>* parent tag.

All other tags are optional.

Also, all URLs in a Sitemap must be from a single host, such as `www.example.com` or `store.example.com`.

Character		Escape Code
Ampersand	&	&
Single Quote	'	'
Double Quote	"	"
Greater Than	>	>
Less Than	<	<

Attribute		Description
<urlset>	required	Encapsulates the file and references the current protocol standard.
<url>	required	Parent tag for each URL entry. The remaining tags are children of this tag.
<loc>	required	URL of the page. This URL must begin with the protocol (such as http) and end with a trailing slash, if your web server requires it. This value must be less than 2,048 characters.
<lastmod>	optional	The date of last modification of the page. This date should be in W3C Datetime format. This format allows you to omit the time portion, if desired, and use YYYY-MM-DD.
<changefreq>	optional	How frequently the page is likely to change. Valid values are: always, hourly, daily, weekly, monthly, yearly never.
<priority>	optional	The priority of this URL relative to other URLs on your site. Valid values range from 0.0 to 1.0. This value does not affect how your pages are compared to pages on other sites—it only lets the search engines know which pages you deem most important for the crawlers. The default priority of a page is 0.5

Example sitemap

```
<?xml version="1.0" encoding="UTF-8"?>

<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.example.com/</loc>
    <lastmod>2005-01-01</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.8</priority>
  </url>
  <url>
    <loc>http://www.example.com/catalog?item=12&desc=vacation_hawaii</loc>
    <changefreq>weekly</changefreq>
  </url>
  <url>
    <loc>http://www.example.com/catalog?item=73&desc=vacation_new_zealand</loc>
    <lastmod>2004-12-23</lastmod>
    <changefreq>weekly</changefreq>
  </url>
  <url>
    <loc>http://www.example.com/catalog?item=74&desc=vacation_newfoundland</loc>
    <lastmod>2004-12-23T18:00:15+00:00</lastmod>
    <priority>0.3</priority>
  </url>
  <url>
    <loc>http://www.example.com/catalog?item=83&desc=vacation_usa</loc>
    <lastmod>2004-11-23</lastmod>
  </url>
</urlset>
```

A sitemap may also list other sitemaps

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">  
  <sitemap>  
    <loc>http://www.example.com/sitemap1.xml.gz</loc>  
    <lastmod>2004-10-01T18:23:17+00:00</lastmod>  
  </sitemap>  
  <sitemap>  
    <loc>http://www.example.com/sitemap2.xml.gz</loc>  
    <lastmod>2005-01-01</lastmod>  
  </sitemap>  
</sitemapindex>
```

agenda

1. details
2. tee shirts
3. finish ranking
4. sitemaps
5. shingles
6. instrumenting memory usage
7. exception handling.

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

 **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.

ingredients dictate what I

ked unsalted nuts because
of "anti-oxidants" a day for
p my favorite sweet and
gar and Spice Candied
ious and welcome nibble

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, $\text{SIM}(S, T) = 3/8$. $\square$

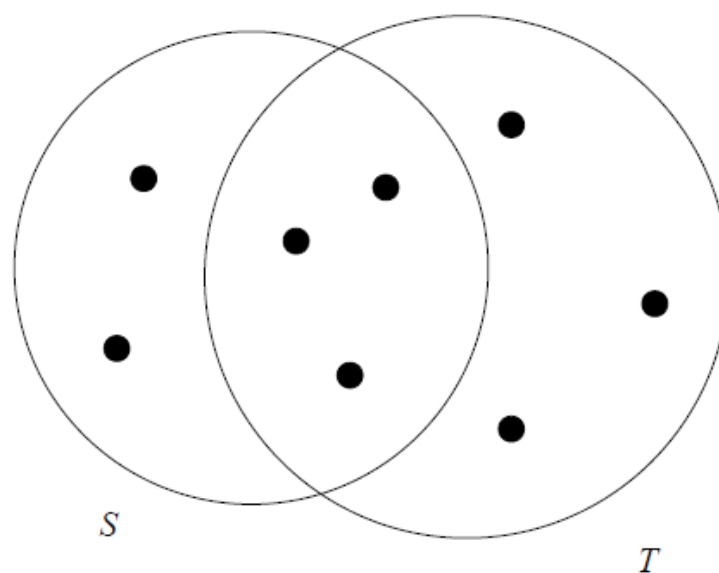


Figure 3.1: Two sets with Jaccard similarity $3/8$

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

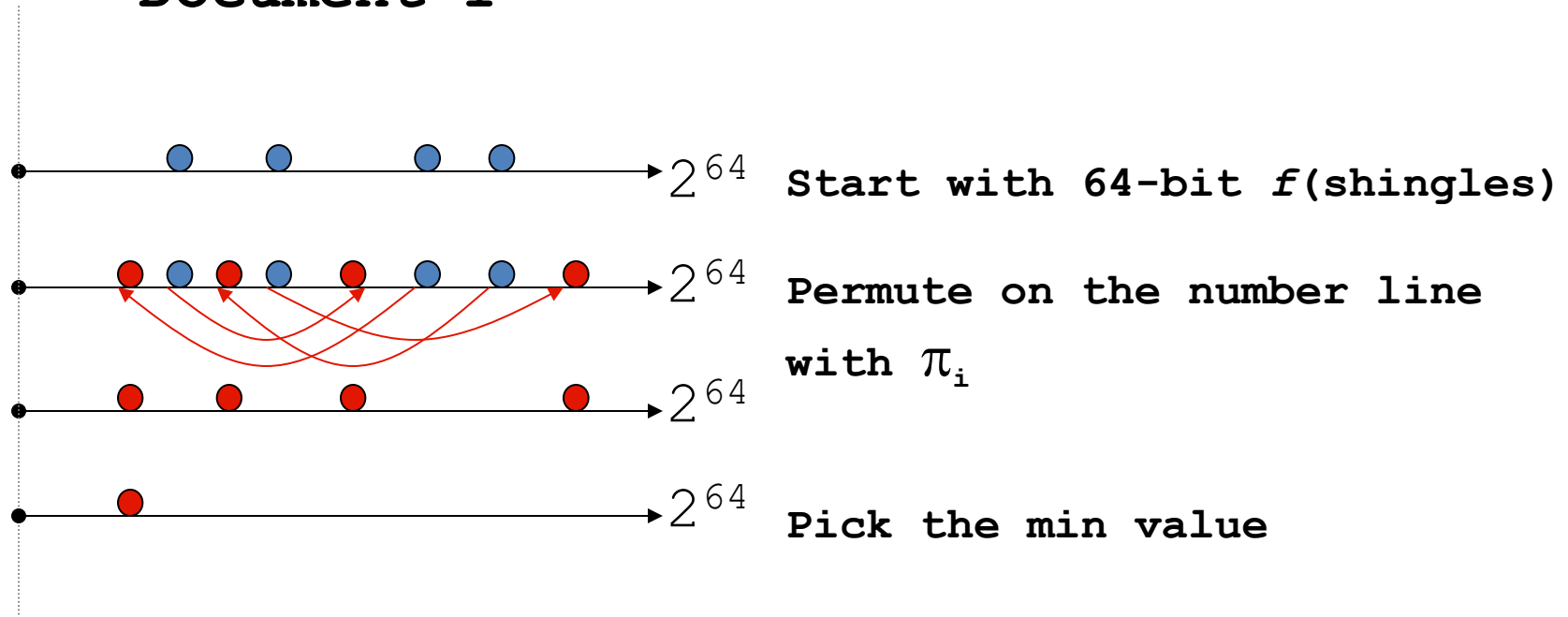
Shingles are then hashed.

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

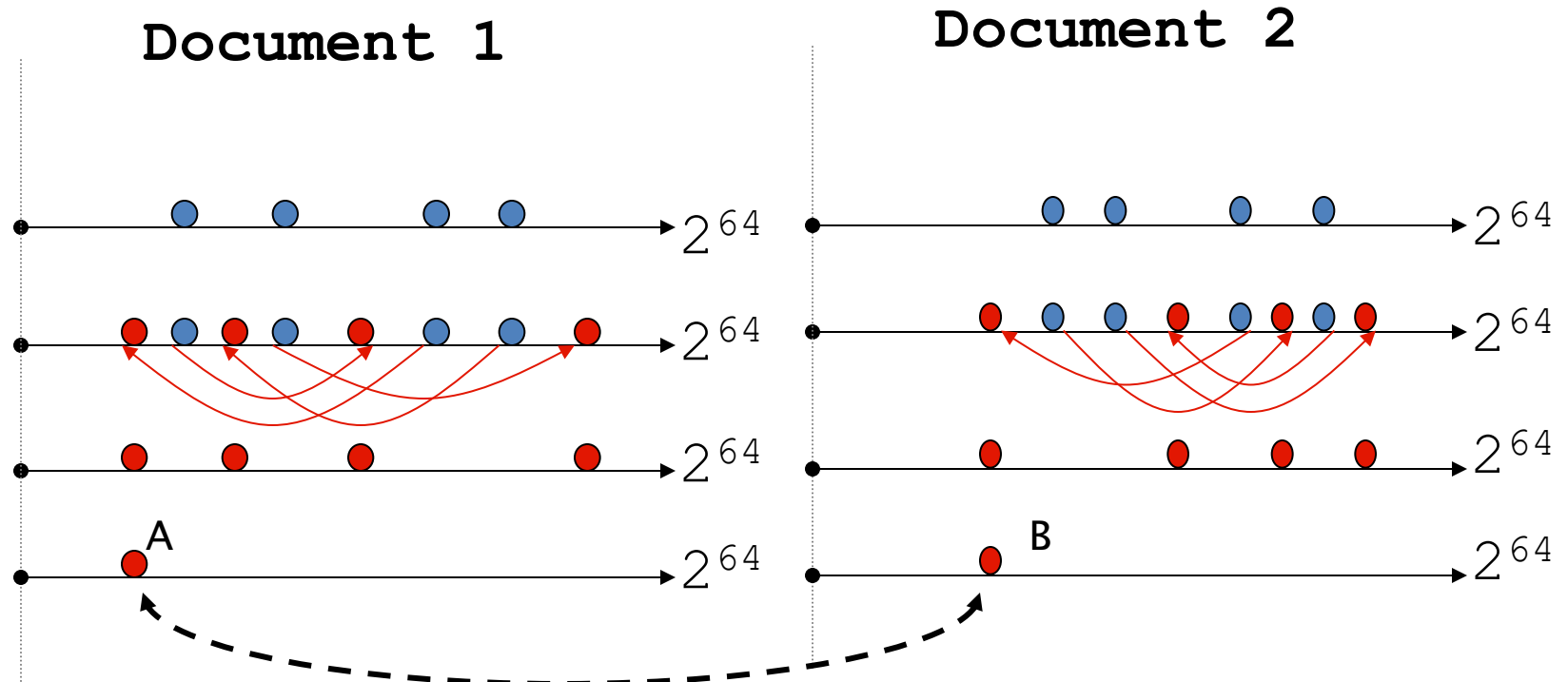
Smallest result is saved in a hash table.

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

Document 1



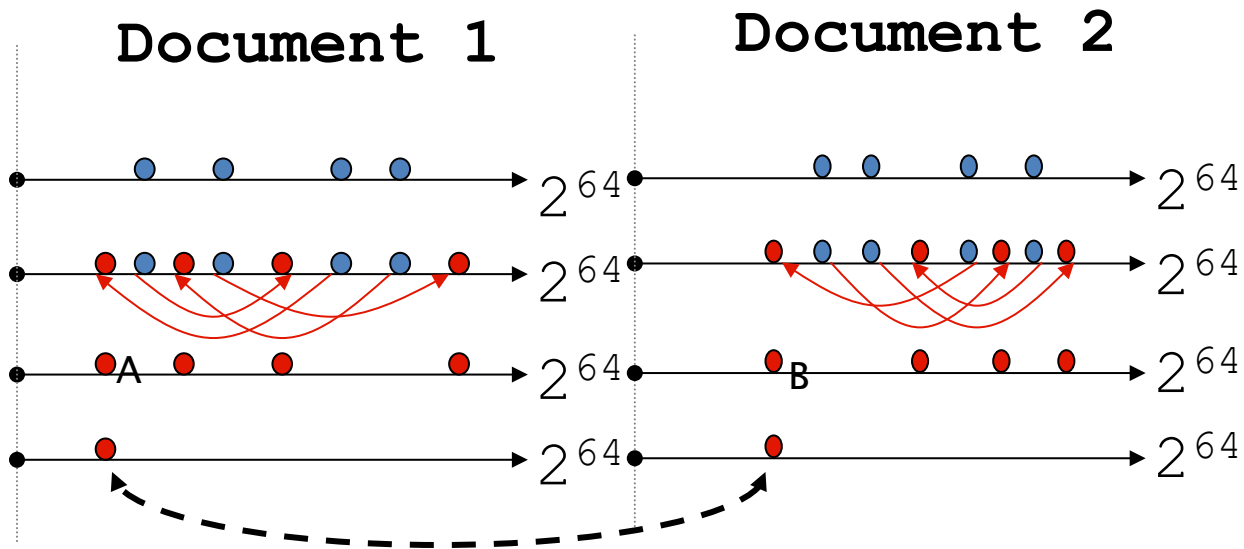
test if $\text{Doc1.Sketch}[i] = \text{Doc2.Sketch}[i]$



Are these equal?

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

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

$$\text{Size_of_intersection} / \text{Size_of_union}$$

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 are reported

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
40923561173765480
60529522094422068
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
40923561173765480
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%
```

agenda

1. details
2. tee shirts
3. finish ranking
4. sitemaps
5. shingles
6. instrumenting memory usage
7. exception handling.

instrumenting memory usage

Sometimes, you would like to know about your own memory usage.

How many objects do you have on the heap, total size, etc.

Problem is to intercept the new and delete operators.

Yes, you can do that.

```
// Toy program to investigate how memory is used by the STL
// string class.

#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;

// Instrument new and delete by overriding the new and delete
// operators (yes, you can do that!) with a wrapper around malloc
// and free, adding a boundary tag to record the size of the memory
// request.

// total reflects the net total amount requested by the user and
// does not include any boundary-tagging overhead.

size_t total;
```

```

void *operator new( size_t size )
{
    total += size;
    size_t *q = ( size_t * )malloc( size + sizeof( size_t ) ),
        *p = q + 1;
    *q = size;
    cout << size << " bytes allocated at " << p <<
        ", total = " << total << endl;
    return ( void *)p;
}

```

```

void operator delete( void *p )
{
    size_t *q = ( size_t * )p - 1;
    total -= *q;
    cout << *q << " bytes freed at " << p <<
        ", total = " << total << endl;
    free( ( void * )q );
};

```

```

int main( )
{
    cout << "1.  starting total on the heap = " << total << endl;
    cout << "    putting a string on the stack" << endl << endl;

    string p;

    cout << endl << "2. sizeof an empty string on the stack = " <<
        sizeof( p ) << endl;
    cout << "    total on the heap = " << total << endl << endl;

    cout << "3.  about to new an empty string" << endl << endl;

    string *s = new string( );

    cout << endl << "4.  empty string has been allocated, total = " <<
        total << endl << endl;

    *s = "hello";

    cout << endl << "5.  string has been set it to hello, total = " <<
        total << endl << endl;
}

```

```

*s += " world how are you i am fine";

cout << endl << "6.  added 28 chars, total = " << total <<
    ", about to c_str" << endl << endl;

cout << "cstr = " << s->c_str( ) << endl;

cout << endl << "7.  about to delete" << endl << endl;

delete s;

cout << endl << "8.  string has been deleting, exiting total = " <<
    total << endl;
}

```

```
51 C% Debug\sizestring.exe | head -20
```

```
1.  starting total on the heap = 0  
    putting a string on the stack
```

```
8 bytes allocated at 00A2FD34, total = 8
```

```
2.  sizeof an empty string on the stack = 28  
    total on the heap = 8
```

```
3.  about to new an empty string
```

```
28 bytes allocated at 00A259AC, total = 36
```

```
8 bytes allocated at 00A2FE84, total = 44
```

```
4.  empty string has been allocated, total = 44
```

```
5.  string has been set it to hello, total = 44
```

```
48 bytes allocated at 00A256A4, total = 92
```

```
52 C%
```

agenda

1. details
2. tee shirts
3. finish ranking
4. sitemaps
5. shingles
6. instrumenting memory usage
7. exception handling.

Failing gracefully

If you're writing a serious application you intend to ship to customers, you have to anticipate *it will fail* in their hands.

If it does, it should *fail gracefully* reporting what happened, freeing up any resources, leaving files in a consistent state, capturing debug information, etc.

Exceptions

Two meanings:

1. *In C++*, it's something you throw, then catch higher up the call stack. You can't catch anything you didn't throw.
2. *To the operating system*, it's a hardware interrupt due to a null pointer deref (a seg fault), a floating point exception, ^C typed at the keyboard, etc.

```
#include <iostream>
#include <string>
using namespace std;

int main( int argc, char **argv )
{
    string s;
    while ( cin >> s )
    {
        try
        {
            cout << "stoi( " << s << " ) = " << stoi( s ) << endl;
        }
        catch ( invalid_argument )
        {
            cout << "stoi( " << s << " ) is invalid" << endl;
        }
        catch ( out_of_range )
        {
            cout << "stoi( " << s << " ) is out of range" << endl;
        }
    }
}
```

```
198 C% cl /EHsc stoi.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.11.25508.2 for x64
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
stoi.cpp
Microsoft (R) Incremental Linker Version 14.11.25508.2
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
/out:stoi.exe
```

```
stoi.obj
```

```
199 C% stoi.exe
```

```
5
```

```
stoi( 5 ) = 5
```

```
hello
```

```
stoi( hello ) is invalid
```

```
1241241241234123412341234123412341234
```

```
stoi( 1241241241234123412341234123412341234 ) is out of range
```

```
^Z
```

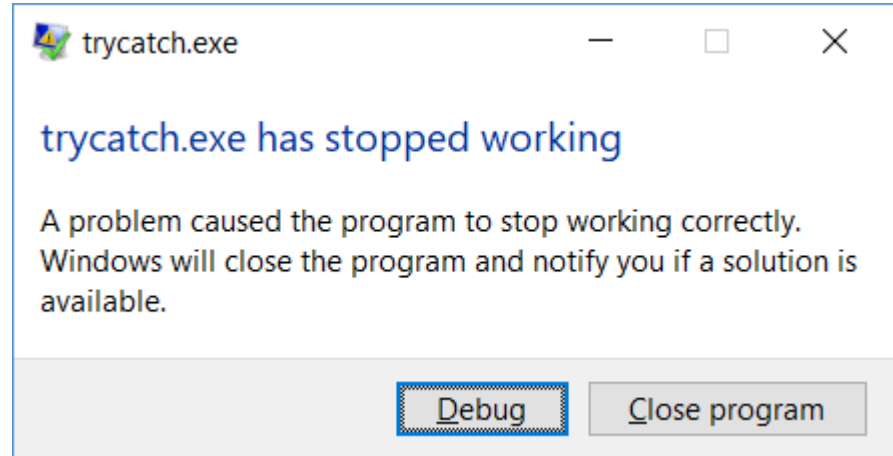
```
// trycatch.cpp
```

```
// Cannot catch a null pointer deref with C++ try/catch.
```

```
// Nicole Hamilton nham@umich.edu
```

```
#include <iostream>
#include <cassert>
#include <stdexcept>
using namespace std;
```

```
int main( void )
{
    int *p = 0;
    try
    {
        *p = 5;
    }
    catch ( ... )
    {
        cout << "Null pointer encountered" << endl;
    }
}
```



```
// throw.cpp

// You could protect a possible null pointer deref
// with a test and a throw, but it's not realistic to do
// to all your derefs.

// Nicole Hamilton nham@umich.edu

#include <iostream>
using namespace std;
```

You could try to protect every pointer refer with a test and a throw, but that's just not realistic.

```

int main( void )
{
    int *p = 0;
    struct NullPointerError
    {
        int **BadPointer;
        NullPointerError( int **p )
        {
            BadPointer = p;
        }
    };

    try
    {
        if ( !p )
            throw NullPointerError( &p );
        *p = 5;
    }
    catch ( NullPointerError &e )
    {
        cout << "Null pointer at " << e.BadPointer << " encountered" << endl;
    }
}

```

```
217 C% cl /EHsc throw.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.11.25508.2 for x64
Copyright (C) Microsoft Corporation. All rights reserved.

throw.cpp
Microsoft (R) Incremental Linker Version 14.11.25508.2
Copyright (C) Microsoft Corporation. All rights reserved.

/out:throw.exe
throw.obj
218 C% throw
Null pointer at 00000033E5CFFE40 encountered
219 C%
```

You really need OS support.

This is the one area where Windows is better than Linux.

Windows added there own keywords, `__try` and `__except`, to the C++ language.

```
// tryexcept.cpp

// Simple Windows __try/__except example, intercepting
// a null pointer dereference using Microsoft try-except
// extension to C and C++.  Compile with /EHsc.

// Choices for the __except filter expression value are:
//
// EXCEPTION_CONTINUE_EXECUTION = -1
// EXCEPTION_CONTINUE_SEARCH    = 0
// EXCEPTION_EXECUTE_HANDLER    = 1

// Nicole Hamilton  nham@umich.edu

#include <windows.h>
#include <iostream>
#include <cassert>
#include <stdexcept>
using namespace std;
```

```
int main( void )
{
    int *p = nullptr;

    __try
    {
        *p = 5;
    }

    __except ( EXCEPTION_EXECUTE_HANDLER ) // Intercept anything
    {
        cout << "Null pointer dereference encountered" << endl;
    }
}
```

```
175 C% cl /EHsc tryexcept.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.11.25508.2 for x64
Copyright (C) Microsoft Corporation. All rights reserved.

tryexcept.cpp
Microsoft (R) Incremental Linker Version 14.11.25508.2
Copyright (C) Microsoft Corporation. All rights reserved.

/out:tryexcept.exe
tryexcept.obj
176 C% tryexcept.exe
Null pointer dereference encountered
177 C%
```

Windows can also give you additional information about the exception.

```
// tryexceptfilter.cpp

// A second Windows try/except example, intercepting
// a null pointer dereference using Microsoft try-except
// extension to C and C++ with an exception filter.
//
// Must be compiled with Microsoft Visual C++ /EHsc options.

// Nicole Hamilton  nham@umich.edu

#include <windows.h>
#include <iostream>
#include <cassert>
#include <stdexcept>
using namespace std;

int ExceptionFilter( DWORD exceptionCode,
                    EXCEPTION_POINTERS *exceptionPointers )
{
    cerr << "Exception code = " << hex << exceptionCode << endl;
    return EXCEPTION_EXECUTE_HANDLER;
}
```

```

int main( void )
{
    // In this example, we'll filter the exception, using
    // GetExceptionCode and GetExceptionInformation to
    // retrieve information about what happened.
    //
    // GetExceptionCode and GetExceptionInformation can
    // only be called from within the __except expression,
    // not from a subroutine.

    int *p = nullptr;

    __try
    {
        *p = 5;
    }
    __except ( ExceptionFilter( GetExceptionCode( ),
                               GetExceptionInformation( ) ) )
    {
        cout << "Null pointer dereference encountered" << endl;
    }
}

```

```
157 C% cl /EHsc tryexceptfilter.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.11.25508.2 for x64
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
tryexceptfilter.cpp
Microsoft (R) Incremental Linker Version 14.11.25508.2
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
/out:tryexceptfilter.exe
tryexceptfilter.obj
158 C% tryexceptfilter.exe
Exception code = c0000005
Null pointer dereference encountered
```

Linux uses more complex “signal handler” model.

If invoked, the signal handler “long jumps” back to earlier place in your code, unwinding the stack.

But note! Destructors are not run on anything popped off the stack by long jump.

```
// LinuxSignalHandler.cpp

// Simple demonstration of how to use the Linux sigaction system call
// along with setjmp and longjmp to intercept a segfault (or any other
// exception.)

// Nicole Hamilton  nham@umich.edu

#include <unistd.h>
#include <setjmp.h>
#include <string.h>
#include <iostream>
using namespace std;

bool errorEncountered = false;
jmp_buf longjmpBuffer;
```

```
void SignalHandler( int signal, siginfo_t *info, void *context )
{
    cerr << "Signal " << signal << " encountered." << endl;

    // Indicate that an error occurred so we don't go down
    // the same path again when we return.

    errorEncountered = true;

    // longjmp back to just before we entered the block
    // failed, returning the signal number.

    longjmp( longjmpBuffer, signal );
}
```

```
int main( int argc, char **argv )
{
    // Create a sigaction structure describing how we'd
    // like to intercept signals.

    struct sigaction action;
    memset( &action, 0, sizeof( action ) );
    action.sa_sigaction = SignalHandler;
    action.sa_flags = SA_SIGINFO;

    // Install the signal handler for seg faults. (We have
    // do this for each signal we'd like to intercept.)

    int sigactionResult = sigaction( SIGSEGV, &action, nullptr );

    // Before entering suspect code, capture a convenient
    // spot to return to. If we encounter an error, this
    // is where we'll pop out again. (Think Groundhog Day
    // or Butterfly Effect.)
    //
    // The first time we setjmp, it returns 0.
    // But if we pop back out because of a longjmp,
    // it'll be with the value passed in the longjmp.

    int setjmpResult = setjmp( longjmpBuffer );
```

```
if ( setjmpResult == 0 )
{
    // Trying for the first time.
    cout << "About to do the nullptr deref." << endl;
    int *p = nullptr;
    *p = 5;
    cout << "Never reached." << endl;
}
else
    cout << "Failed, error = " << setjmpResult << endl;
}
```

```
181 C% g++ LinuxSignalHandler.cpp -o LinuxSignalHandler
182 C% LinuxSignalHandler.exe
About to do the nullptr deref.
Signal 11 encountered.
Failed, error = 11
183 C%
```