

Document Type Detection

Agenda

- Motivation for Automated File Detection
- Use Cases
- File Formats
- Approaches for Automated File Detection
- BFD, BFC, and FHT
- Datasets to use for Document Type Detection
- Future Work

More Use Cases

- Content Repositories**
 - Document Sharing systems, e.g., share point need to automatically identify file types – put files accessed more often (e.g., News such as RSS) on faster servers, rather than less frequently changed (e.g., PDFs)
- Web Crawling**
 - Go after certain types of files for your e.g., food rating review system (e.g., Microformats, Microdata), ignore other file types

Some History in the Space

- Operating Systems**
 - Need to associate applications to open with file types
 - As far back as the DEC operating system
 - Original filenames in File Access Table (FAT) filesystem limited to 8 character letters and 3 char extension (8.3 filename)
- Virus Scanners**
 - Default in my cases to scanning only executable files, so understanding which files are executable and having a high degree of accuracy and rapid response in that detection is necessary – could mean the difference in detecting Virus or not.

File Formats

language of how to read characters

- A file format is a standard way that information is encoded for storage in a computer file. It specifies how bits are used to encode information in a digital storage medium. File formats may be either proprietary or free and may be either unpublished or open.**
- https://en.wikipedia.org/wiki/File_format

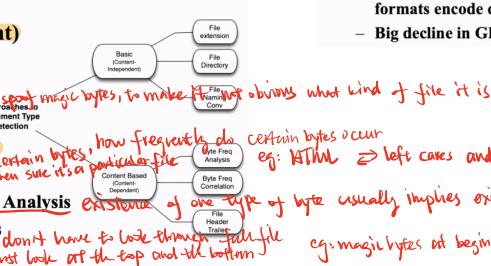
Approaches for automatic detection

Basic (Content-Independent)

- File extension
- File Directory Structure
- File Naming Convention

Content-Based Algorithms

- Byte Frequency Analysis
- Byte Frequency Correlation Analysis
- File Header Trailer Analysis
- /etc/magic



Why automated file detection



- Virus Scanning**
 - Many virus scanning software tools rely on the ability to identify which files may contain viruses in them, and so the ability to automatically detect executable files is key
- Firewalls / Networking / Intrusion Detection Systems**
 - At the network level, allowing e.g., SMTP communication on port 25 with an EXE attachment or Python file is important to detect
- File Forensics**
 - When handed a bunch of disks with data on them, being ability to accurately classify that data for legal purposes, etc., is key
- Scientific Data / Processing / Big Data**
 - About common cloud
 - Sorting level 1 data (raw telemetry) from processed geo-referenced science data (level 2)



Digital Cyber Investigations

file can be intentionally spoofed and work differently when it's not

- File extensions and even MIME magic is no longer enough to detect file types!**
 - eg. love bug, Stuxnet, Flame, Today's file
- Forensic investigators have found critical evidence (images, multimedia, etc.) embedded in otherwise “safe” file types (Alamri & Allen, IEEE SouthEastCon 2015)
- Extensions are often changed to “safe” file types and encoded with bad content (viruses)

8.3 filenames

8 characters...3 characters-extension



An 8.3 filename (also called a short filename or SFN) is a filename convention used by old versions of DOS and versions of Microsoft Windows prior to Windows 95 and Windows NT 3.5. It is also used in modern Microsoft operating systems as an alternate filename to the long filename for compatibility with legacy programs.

8.3 filename - Wikipedia

https://en.wikipedia.org/wiki/8.3_filename

File Formats



Free file formats

- Difference between e.g., Open Office (.odf) and e.g., Microsoft Office (.docx)
- Specification for file header, encoding (compression), etc., provided openly so as to allow many reference implementations
 - LibreOffice, Apache OpenOffice, Apache Open Document Format (ODF) Incubating

Closed file formats

- Protected by trade secrets, non disclosure agreements, specifications aren't public
- Can be reverse engineered using techniques like we will discuss today
- File format patents are not directly permitted under US law, some file formats encode data using patented algorithms but can apply it to encode data
- Big decline in GIF because of this (compression), and PNG increase

try to spot magic bytes, to make it not obvious what kind of file it is
 see certain bytes, how frequently do certain bytes occur
 eg: HTML → left cares and right cares ← b0 < < b2 bytes are seen often
 then sure it's a particular file
 existence of one type of byte usually implies existence of another
 don't have to look through full file just look off the top and the bottom
 eg: magic bytes at beginning helps to detect

Basic approaches

- 1. File Extension based** – can easily be spoofed. Microsoft's Operating Systems typically have used this approach almost exclusively. OS maintains table of associations and must be manually updated by user.
- 2. /etc/magic file** – Special file in UNIX oriented systems that look at first 16 bits of each file and then associates these as a magic number with each file. Table is maintained in /etc/magic file
- 3. Container File Formats** – encapsulate other files

Some limitations of /etc/magic

- Magic numbers must be predefined before the files are generated
- Makes it difficult to evolve and adapt magic numbers over time since it would require having to go back and adapt these signatures in existing files
- Not all file types use magic numbers (we'll see more about this later)
- Since only 16-bits allocated, had to expand to handle #! special operator later (indicates other commands to run on the file and that file is script)

Container file formats

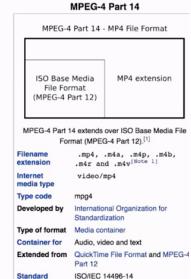
- Standard Archive Format (SAF)**
 - <http://www.fileformat.info/format/saf/cgff.htm>
 - Archival storage of multiple data types in a standard format
 - Proprietary
- OLE (Object Linking and Embedding Compound File)**
 - http://www.forensicswiki.org/wiki/OLE_Compound_File
 - Originally Proprietary (Microsoft), became part of Open Office
- Formats include special information that indicate MIME type of internal files**
- A bigger list is here: https://en.wikipedia.org/wiki/Digital_container_format

Another container file format

mp4 file special extension

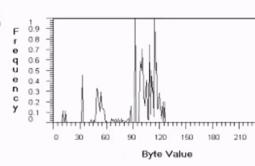
- MPEG Layer 4**
- Nice elements
 - Open standard (ISO)
 - Many proprietary implementations initially didn't all play the same way
 - Has largely been standardized now

https://en.wikipedia.org/wiki/MPEG-4_Part_14



Content-Based File Type Detection

- McDaniel & Heydari, 2003**
 - Investigated three approaches
 - Byte Frequency Analysis (BFA)
 - Byte Frequency Correlation (BFC)
 - File Header Trailer (FHT)
- Some key targets**
 - Accuracy – approach should be highly accurate (overcome filenames and difficult to spoof (security/firewalls/viruses))
 - Automatic – lots of files, big data, need to run fast
 - Fingerprints should be small and easily comparable
 - Flexibility – don't want to have to maintain tables



Format of /etc/magic file

- <https://www.mkssoftware.com/docs/man4/magic.4.asp> - file tool in UNIX
- Lines contain byte sequences that can be compared
- >** indicates a continuation of the previous line and any patterns preceded by this character are checked
- &** indicates continuation of previous line and MUST match this byte sequence.
- First col is the byte offset, then data type, then byte sequence to compare and then descriptive name for the type

EXAMPLES

Here are some sample entries:

```
0     byte  0x80           OMF object file (Microsoft relocatable)
0     short   0x5AD          executable binary executable (.EXE)
>+0x8C short   0x5AD        -- Windows Win32 format
>+54    byte  2             -- Win32 or NT Portable format
>+0x32 string  PKWARE     -- Windows format
>+0x24 string  LZMA       Self extracting Zip
>+0x25 string  ZIP         Self extracting Zip
0     short   0xFFE2        Self extracting Zip
Windows Registry exported text file (.REG)
Windows Registry Editor Version 5.00
```

....

Ongoing research regarding MAGIC

- Research by Beebe, Maddox et al. 2013 on using unigrams (single bytes) and bigrams (byte pairs) to create n-grams (groups of bytes)
- General research area is deriving a representative **fingerprint** (recall Deduplication lecture)
- Most modern approaches are focused on **byte frequencies** (more on this later)

Container file format limitations

- Many began as **Proprietary** – vendor lock in
- Brings together data and **metadata** – usually better to let them evolve separately
- Wrappers are necessary and we've created another level of indirection to get at the internal constituent files

– And a new set of Magic, and need for detecting



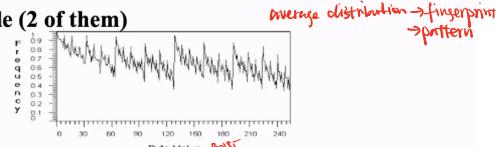
Of course we have seen an example of these: message/* and multipart/* attachments from top level types of MIME taxonomy

Learning from the actual File Data

- Previous approaches were somewhat limited
 - Rely on filenames which are easily spoofed and/or changed
 - Rely on tables that are difficult to discern and must be maintained and updated (Magic)
 - Rely on new proprietary "wrapper formats" (Containers)
- Key Insight:** What if we simply examined the data inside many existing files and tried to develop a "fingerprint" a la Similarity metrics to identify files => Content-based approaches identify through pattern

Byte Frequency Distribution (BFD)

- GIF file (2 of them)



- Use 8-bit (byte) signatures and divide them into bins

9	2	1	2	5	1	...	1	3	9	0	0	1
0	1	2	3	4	5							255

frequency of bytes 2^8 (0-256)

Count freq
in each

1	.22	.11	.22	.55	.1111	.33	1	0	.11	
0	1	2	3	4	5							255

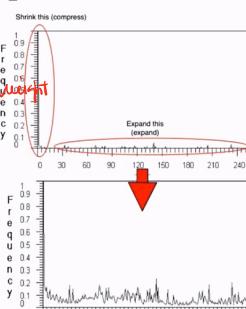
normalized scores 0-1
(divide by largest frequency)

Some issues with BFD

- BFD for an EXE file
- Note it has a large variation between 0 byte and small signatures in rest
- Use a companding function to amplify the smaller frequencies
- Smaller frequencies lead to difficulty in BFD signature accuracy and efficacy

Companding process

- Companding amplifies low signal and compresses high signal
- Results after companding of the ~~file~~ file
- Since original numbers were normalized 0..1 the result companding will be too



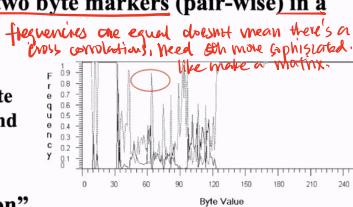
BFD "correlation" strength

- If a byte value occurs with some regular frequency f in a file type then it is an important feature by definition for file identification
- Compute "correlation strength" between a type (e.g., GIF's) BFD and a particular file type
- Difference the BFD from an input file and its overall BFD fingerprint
 - Compute a "similarity" or "correlation" between an input file and its BFD
 - If frequency is close to fingerprint, then file is likely, e.g., a GIF

DSCI 550 CAM-27

Byte Frequency Cross Correlation (BFC)

- In looking at Byte Frequency Correlation, sometimes it is possible to notice e.g., equal sized spikes between two byte markers (pair-wise) in a file type
- Example
 - HTML files, byte values 60 "<" and 62 ">"
- This is called "cross correlation"



File/Header Trailer (FHT)

- Sometimes BFA and BFC are unable to identify file types b/c no distinct patterns
 - Text/plain can have such varied byte patterns, it's hard to specialize
- Also BFA/BFC has to full file scan
- File Headers and Trailers – patterns of bytes that appear at fixed location at beginning and end of file
 - Reduce full file scan
 - Still take into consideration strong byte frequencies (BF)
- Pick H header bytes from beginning file
- Pick T trailing bytes from end of file
- Generate $H \times 256$ matrix; and $T \times 256$ matrix

Companding function

- <https://www.techopedia.com/definition/24121/com-panding>
- From the signal processing domain
- "Compressing" and "Expanding" = companding
- From Wikipedia reduce some singles and increase some other single <https://en.wikipedia.org/wiki/Companding>
 - "mitigating the detrimental effects of a channel with limited dynamic range"
 - In our case, mitigating limited dynamic range of frequencies by compression and expanding

Finalizing BFDs

- Resulting BFDs are generated for file types based on the actual contents of the file – somewhat limiting b/c you have to scan the whole file for traces of the byte fingerprints
- Each file BFD is then averaged across many samples from the same type
 - Take 100 GIFs and generate BFD and then average them
- Resulting averaged BFD is now the generalized indicator or **fingerprint** for a file type

BFD Correlation

- GIF files
- Average BFD fingerprint

Two key pieces of BFC

- Average difference in frequency between all byte pairs
- Correlation strength of any new file and that average frequency difference is how ~~similar~~ ~~different~~ they are
- Cross-correlation array is built 0-255, to represent correlation between byte pairs
 - (i,j) stores the freq of co-occurrence of i and j bytes
 - (i,j) stores the normalized freqs
- Can be compared to new files

General process: FHT

- Compute H and T matrices
- Flag a 1 if the header byte is equal to the col value (0-255), 0 otherwise
- Sparse matrix
 - H byte 0
 - H byte 1
 - \vdots
 - H byte H
- FHT is strong motivation for MIME "magic"
- GIF specifies GIF87a; GIF89a at byte 0

sparse matrix is a 1 if that header byte value (0-255) is present at byte 0, 0 otherwise

average and compute correlation strength across sample set

Evaluation of BFD, BFC and FHT

- 30 file type fingerprints across 120 files
 - ACD, DOC, PPT, XLS
- **BFA accuracy is only 27.5%**
 - Better than random guess, but not by much
 - Speed was 0.010 seconds (fastest; file size may be a factor)
- **BFC accuracy is only 45.83%**
 - Speed was 1.19 seconds (worst)
- **FHT accuracy is 95.85%**
 - Also was 2nd place in speed, not far off (0.015 seconds per file)

The GovDocs dataset

- <http://digitalcorpora.org/corpora/govdocs>
- 1 million files
 - Made available in multiple ways: set of 1000 x 1000 directories
 - 1000 ZIP files
 - Tar files
 - A set of 1000 directories, with 1000 files in each directory, downloadable from our server at <http://digitalcorpora.org/corp/files/govdocs1/>.
 - Subsets of the data
- Metadata to go along with the files
 - URL, download time, search term, md5 hash, dublin core metadata (MIME type)



One of Garfinkel's famous experiments

- In 2003, Garfinkel and Abhi Shelat published an article in IEEE Security & Privacy Magazine reporting on an experiment in which they purchased 158 used hard drives from a variety of sources and checked to see whether they still contained readable data. Roughly one third of the drives appeared to have information that was highly confidential and should have been erased prior to the drive's resale.

Common Crawl Dataset

- <http://commoncrawl.org/the-data/get-started/>
- Data is available in ARC (compressed) file format
- 7 years and petabytes of crawled and collected data
- Made available as part of Amazon's Public Dataset program
 - <http://aws.amazon.com/public-data-sets/>

Tika and Document Type Detection

- Use GovDocs as a release-oriented test
 - Use Tika Batch (we'll discuss later) to iterate over GovDocs on a machine freely provided by RackSpace
 - Compare precision/recall and regressions against previously computed MIME types to test detectors
- <https://wiki.apache.org/tika/TikaBatchOverview>
- <https://wiki.apache.org/tika/VirtualMachine>

FHT promising approach with speed and

accuracy

- Natural evolution is to MIME magic
 - Extend H and T signatures with priority / precedence
 - Simplify by treating H and T the same, simply with offsets
- Common approaches today (Tika does this)
 - File extension hint (allow strong override from MIME magic)
 - Content-Type hint (potentially if available from metadata surrounding file, e.g., if from web server)

GovDocs Dataset

- Garfinkel, Farrell, Roussev and Dinolt, Bringing Science to Digital Forensics with Standardized Forensic Corpora, DFRWS 2009, Montreal, Canada
- Simson Leon Garfinkel
- Worked at NIST, and Naval Postgraduate School
- Expert in computer science forensics and computer security



Bit rot

- How to preserve and assess digital data DECADES or CENTURIES later?

Our cookie policy has changed. Please see [our cookie policy](#) for more details and to change your cookie preferences.
By continuing to browse this site you are agreeing to our use of cookies.

[More from The Economist](#) | [My Reservation](#) | [Subscribe](#) | [Log in or register](#)

Digital Data Bit rot

Apr 26th 2012 | [Comment \(2\)](#) | [Timeline reader reading list](#)

PICTURE yourself as a historian in 2055, trying to make sense of this year's American election. You might be surprised to learn that the records of the campaign will have long since perished. Data stored electronically decays. Many floppy disks from the early digital age are already unusable. If you are lucky, copies of campaign documents may still exist on paper, but even then they are fragile. Even computers that can read such ancient technologies as CD and USB thumb-drives, But even that may not be enough. Computer files are not worth anything without software to open them.

But why not be able to read them? Already NASA has lost data from some of its earliest missions to the moon because the machines used to read the tapes were scrapped and no longer exist. The agency is now trying to find a way to read the data again. And it is not alone. Climate change is amplifying in the Polar Regions. Polar amplification is captured via space and airborne remote sensing, in-situ measurement, and climate modeling. Beyond the rich literature that documents changing Polar regions, each method of Polar data collection presents a unique challenge. One way around this is to collect a broad set of data types, ranging from text-based reports to images and video. Another is to use machine learning techniques (e.g. HDP, LDA, GPC) to identify patterns in the data. This is a primary challenge in scientific discovery. Inclusion of the Polar dataset in TREC-DD would help advance science through research and provide TREC-DD a new challenge in the realm of search relevancy.

Dataset Description

This dataset is a collection of web crawls from three primary sources:

1. Dr. Chris Mattmann's crawl of ADE, performed in the Open Science Codefest and the NSF Data4D Hackathon for Polar Cyberinfrastructure.
2. Dr. Mattmann's CSD 872 Course at URC, students submitted 3 datasets: 2 sets of ACADS and one of NASA AOD.
3. Dr. Mattmann's CSD 872 Course at URC, students submitted 3 individual crawls of NASA ACADS, NSDC, ADE, and AOD.

Each web crawl used Apache Nutch as the core framework for web crawling and Apache Tika as the main content detection and extraction framework. Nutch is a distributed search engine that runs on top of Apache Hadoop. Apache Nutch is an open source framework for metadata extraction, automatic text mining, and information retrieval. Web crawls were focused on three polar data repositories: the National Science Foundation Advanced Cooperative Arctic Data Center (ACADS), the National Snow and Ice Data Center (NSIDC Arctic Data Explorer (ADE)), and the Atmospheric Radiation Measurement (ARM) Data Discovery System (DD).

<http://github.com/chris mattmann/trec-dd-polar/>

TREC Dynamic Domain Polar data

- 1.7M urls
- 93+ different web MIME types
- Collected across three of the most widely used NSF, NASA arctic data repositories from 2015-2016
- Data available on Amazon S3 and at Arcticdata.io

TREC Dynamic Domain Polar Dataset

Purpose of data:

Climate change is amplified in the Polar Regions. Polar amplification is captured via space and airborne remote sensing, in-situ measurement, and climate modeling. Beyond the rich literature that documents changing Polar regions, each method of Polar data collection presents a unique challenge. One way around this is to collect a broad set of data types, ranging from text-based reports to images and video. Another is to use machine learning techniques (e.g. HDP, LDA, GPC) to identify patterns in the data. This is a primary challenge in scientific discovery. Inclusion of the Polar dataset in TREC-DD would help advance science through research and provide TREC-DD a new challenge in the realm of search relevancy.

Dataset Description:

This dataset is a collection of web crawls from three primary sources:

1. Dr. Chris Mattmann's crawl of ADE, performed in the Open Science Codefest and the NSF Data4D Hackathon for Polar Cyberinfrastructure.
2. Dr. Mattmann's CSD 872 Course at URC, students submitted 3 datasets: 2 sets of ACADS and one of NASA AOD.
3. Dr. Mattmann's CSD 872 Course at URC, students submitted 3 individual crawls of NASA ACADS, NSDC, ADE, and AOD.

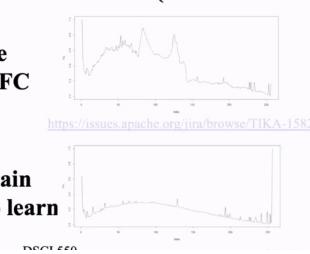
Each web crawl used Apache Nutch as the core framework for web crawling and Apache Tika as the main content detection and extraction framework. Nutch is a distributed search engine that runs on top of Apache Hadoop. Apache Nutch is an open source framework for metadata extraction, automatic text mining, and information retrieval.

Web crawls were focused on three polar data repositories: the National Science Foundation Advanced Cooperative Arctic Data Center (ACADS), the National Snow and Ice Data Center (NSIDC Arctic Data Explorer (ADE)), and the Atmospheric Radiation Measurement (ARM) Data Discovery System (DD).

<http://github.com/chris mattmann/trec-dd-polar/>

Current Research Trends

- BFD and BFC rarely looked at for some of the modern scientific data file formats (and other increasing formats)
- Gridded Binary File Format and BFD/BFC
- GRIB is a popular scientific format
- Question: can we train Neural Networks to learn BFD for file types?



Modern File Type Identification Techniques

- Alamri and Allen, 2015
- Modern ML approaches
 - SVM classifier
 - K-nearest neighbor
 - Neural Networks
 - Linear Discriminant Analysis
- All relies upon what features to select
- How to “featurize” the file type?

Some recent results

- Highest accuracy is 97%
 - K-NN with 8 features
500 byte files
 - NN-MLP with 64 features and 500 byte files
- Feature size isn't dependent
- Sample size likely too low

TABLE III. PARAMETERS THAT PRODUCED THE BEST CLASSIFICATION RATE FOR THE LINEAR DISCRIMINANT ANALYSIS APPROACH

Model	Number of Features	Fragment Size	Training Data Ratio	Classification Rate
LR	8	500 byte	80:20	93%
LR	16	500 byte	80:20	93%
LR	32	500 byte	80:20	93%
LR	64	500 byte	80:20	93%
LR	128	500 byte	80:20	93%
LR	256	500 byte	80:20	93%
LR	512	500 byte	80:20	93%
LR	1024	500 byte	80:20	93%
LR	2048	500 byte	80:20	93%
LR	4096	500 byte	80:20	93%
LR	8192	500 byte	80:20	93%
LR	16384	500 byte	80:20	93%
LR	32768	500 byte	80:20	93%
LR	65536	500 byte	80:20	93%
LR	131072	500 byte	80:20	93%
LR	262144	500 byte	80:20	93%
LR	524288	500 byte	80:20	93%
LR	1048576	500 byte	80:20	93%
LR	2097152	500 byte	80:20	93%
LR	4194304	500 byte	80:20	93%
LR	8388608	500 byte	80:20	93%
LR	16777216	500 byte	80:20	93%
LR	33554432	500 byte	80:20	93%
LR	67108864	500 byte	80:20	93%
LR	134217728	500 byte	80:20	93%
LR	268435456	500 byte	80:20	93%
LR	536870912	500 byte	80:20	93%
LR	1073741824	500 byte	80:20	93%
LR	2147483648	500 byte	80:20	93%
LR	4294967296	500 byte	80:20	93%
LR	8589934592	500 byte	80:20	93%
LR	17179869184	500 byte	80:20	93%
LR	34359738368	500 byte	80:20	93%
LR	68719476736	500 byte	80:20	93%
LR	137438953472	500 byte	80:20	93%
LR	274877856944	500 byte	80:20	93%
LR	549755713888	500 byte	80:20	93%
LR	1099511427776	500 byte	80:20	93%
LR	2199022855552	500 byte	80:20	93%
LR	4398045711104	500 byte	80:20	93%
LR	8796091422208	500 byte	80:20	93%
LR	17592182844416	500 byte	80:20	93%
LR	35184365688832	500 byte	80:20	93%
LR	70368731377664	500 byte	80:20	93%
LR	140737462755328	500 byte	80:20	93%
LR	281474925510656	500 byte	80:20	93%
LR	562949851021312	500 byte	80:20	93%
LR	112589970204264	500 byte	80:20	93%
LR	225179940408528	500 byte	80:20	93%
LR	450359880817056	500 byte	80:20	93%
LR	900719761634112	500 byte	80:20	93%
LR	180143952326824	500 byte	80:20	93%
LR	360287904653648	500 byte	80:20	93%
LR	720575809307296	500 byte	80:20	93%
LR	1441151618614592	500 byte	80:20	93%
LR	2882303237229184	500 byte	80:20	93%
LR	5764606474458368	500 byte	80:20	93%
LR	11529212948916736	500 byte	80:20	93%
LR	23058425897833472	500 byte	80:20	93%
LR	46116851795666944	500 byte	80:20	93%
LR	92233703591333888	500 byte	80:20	93%
LR	184467407182667776	500 byte	80:20	93%
LR	368934814365335552	500 byte	80:20	93%
LR	737869628730671104	500 byte	80:20	93%
LR	1475739257461342208	500 byte	80:20	93%
LR	2951478514922684416	500 byte	80:20	93%
LR	5902957029845368832	500 byte	80:20	93%
LR	11805914059690737664	500 byte	80:20	93%
LR	23611828119381475328	500 byte	80:20	93%
LR	47223656238762950656	500 byte	80:20	93%
LR	94447312477525901312	500 byte	80:20	93%
LR	188894624955051802624	500 byte	80:20	93%
LR	377789249910103605248	500 byte	80:20	93%
LR	755578499820207210496	500 byte	80:20	93%
LR	151115699964041442096	500 byte	80:20	93%
LR	302231399928082884192	500 byte	80:20	93%
LR	604462799856165768384	500 byte	80:20	93%
LR	1208925599712315536768	500 byte	80:20	93%
LR	2417851199424631073536	500 byte	80:20	93%
LR	4835702398849262147072	500 byte	80:20	93%
LR	9671404797698524294144	500 byte	80:20	93%
LR	19342809595397048588288	500 byte	80:20	93%
LR	38685619190794097176576	500 byte	80:20	93%
LR	77371238381588194353152	500 byte	80:20	93%
LR	154742476763176388706304	500 byte	80:20	93%
LR	309484953526352777412608	500 byte	80:20	93%
LR	618969907052705554825216	500 byte	80:20	93%
LR	1237939814105411109650432	500 byte	80:20	93%
LR	2475879628210822219300864	500 byte	80:20	93%
LR	4951759256421644438601728	500 byte	80:20	93%
LR	9903518512843288877203456	500 byte	80:20	93%
LR	19807037025686577754406912	500 byte	80:20	93%
LR	39614074051373155508813824	500 byte	80:20	93%
LR	79228148102746311017627648	500 byte	80:20	93%
LR	15845629620549262203525328	500 byte	80:20	93%
LR	31691259241098524407050656	500 byte	80:20	93%
LR	63382518482197048814101312	500 byte	80:20	93%
LR	12676503696439409762820264	500 byte	80:20	93%
LR	25353007392878819525640528	500 byte	80:20	93%
LR	50706014785757639051281056	500 byte	80:20	93%
LR	101412029571515278102562112	500 byte	80:20	93%
LR	202824059143030556205124224	500 byte	80:20	93%
LR	405648118286061112410248448	500 byte	80:20	93%
LR	811296236572122224820496896	500 byte	80:20	93%
LR	1622592473144244449640993792	500 byte	80:20	93%
LR	3245184946288488899281987584	500 byte	80:20	93%
LR	6490369892576977798563975168	500 byte	80:20	93%
LR	12980739785153955597127550336	500 byte	80:20	93%
LR	25961479570307911194255100672	500 byte	80:20	93%
LR	51922959140615822388510201344	500 byte	80:20	93%
LR	103845918281236444777054002688	500 byte	80:20	93%
LR	207691836562472889554108005376	500 byte	80:20	93%
LR	415383673124945779108216010752	500 byte	80:20	93%
LR	830767346249891558216432021504	500 byte	80:20	93%
LR	1661534892497983116432864043008	500 byte	80:20	93%
LR	3323069784995966232865728086016	500 byte	80:20	93%
LR	6646139569991932465711456172032	500 byte	80:20	93%
LR	1329227913998386493442812344064	500 byte	80:20	93%
LR	2658455827996772986885624688128	500 byte	80:20	93%
LR	5316911655993545973771249376256	500 byte	80:20	93%
LR	1063382331986789194754488873256	500 byte	80:20	93%
LR	2126764663973578389508977746512	500 byte	80:20	93%
LR	4253529327947156779017955493024	500 byte	80:20	93%
LR	8507058655894313558035910986048	500 byte	80:20	93%
LR	17014117311788627116071821972096	500 byte	80:20	93%
LR	34028234623577254232143643944192	500 byte	80:20	93%
LR	68056469247154508464287287888384	500 byte	80:20	93%
LR	13611293849430901692854457577672	500 byte	80:20	93%
LR	27222587698861803385708915155344	500 byte	80:20	93%
LR	54445175397723606771417830310688	500 byte	80:20	93%
LR	10889035079544721354283666062176	500 byte	80:20	93%
LR	21778070159089442708567332124352	500 byte	80:20	93%
LR	43556140318178885417134664248704	500 byte	80:20	93%
LR	87112280636357770834269328497408	500 byte	80:20	93%
LR	174224561272715541668538656994816	500 byte	80:20	93%
LR	348449122545431083337077313989632	500 byte	80:20	93%
LR	696898245090862166674154627979264	500 byte	80:20	93%
LR	139379649018172433334833315958856	500 byte	80:20	93%
LR	278759298036344866669666631917712	500 byte	80:20	93%
LR	557518596072689733339333263835424	500 byte	80:20	93%
LR	111503719214537946667866532767088	500 byte	80:20	93%
LR	223007438429075893335733065534176	500 byte	80:20	93%
LR	446014876858151786671466131068352	500 byte	80:20	93%
LR	892029753716303573342932262136704	500 byte	80:20	93%
LR	1784059507432607146685864524273408	500 byte	80:20	93%
LR	3568119014865214293371729048546816	500 byte	80:20	93%
LR	7136238029730428586743458097093632	500 byte	80:20	93%
LR	14272476059460857173486916194187264	500 byte	80:20	93%
LR	28544952118921714346973832388374528	500 byte	80:20	93%
LR	57089854237843428693947664776749056	500 byte	80:20	93%
LR	114179708475686857387853329553498112	500 byte	80:20	93%
LR	228359416951373714775706659106996224	500 byte	80:20	93%
LR	456718833902747429551413318213992448	500 byte	80:20	93%
LR	91343766780549485910282663642798496	500 byte	80:20	93%
LR	18268753356109897182056531728559696	500 byte	80:20	93%
LR	36537506712219794364113063457119392	500 byte	80:20	93%
LR	73075013424439588728226126914238784	500 byte	80:20	93%
LR	14615002684887917745645225382857568	500 byte	80:20	93%
LR	29230005369775835491290450765715136	500 byte	80:20	93%
LR	58460010739551670982580901531430272	500 byte	80:20	93%
LR	11692002147870341965016180306280544	500 byte	80:20	93%
LR	23384004295740683930032360612561088	500 byte	80:20	93%
LR	4676800859148136786006472122512216	500 byte	80:20	93%
LR	9353601718296273572001284245024432	500 byte	80:20	93%
LR	18707203436592547144002568490048864	500 byte	80:20	93%
LR	37414406873185094288005136980097728	500 byte	80:20	93%
LR	74828813746370188576001027960195456	500 byte	80:20	93%
LR	149657627492740377152020558920390912	500 byte	80:20	93%
LR	299315254985480754304041117840781824	500 byte	80:20	93%
LR	598630509970961			