COMPRESSING DOCUMENTS

Use algorithms to reduce the bit used to represent a document, in order to minimize the resources required to store and transmit data

SPACE-TIME COMPLEXITY TRADEOFF = you have to consume resources in the compression/decompression phase

Types of compression →

LOSSLESS = DATA DOESN'T LOSE INFORMATION. IT EXPLOITS STATISTICAL REDUNDANCY (EXAMPLE: IF I HAVE 200 CONTIGUOS RED PIXEL, I DON'T HAVE TO CODE THEM ALL AS "RED, RED, RED..." BUT I CAN DO INSTEAD "200 RED")

LZ

LOSSY = DROPS NON-ESSENTIAL DETAILS OF THE DATA TO SAVE MORE SPACE (EXAMPLE: ROUND UP THE PIXEL COLORS THAT ARE NOT PERCEIVED WELL BY HUMANS) R:7.2 G:3.222 B:9.54 → R:7 G:3 B:4

JPEG

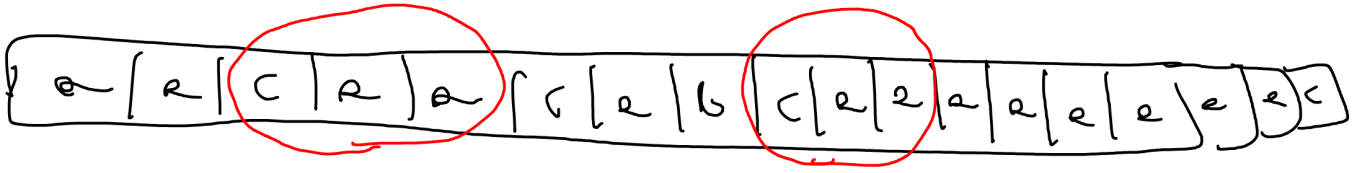
LZ77

WE ALSO REFER TO IT AS GZIP

LOSSLESS DATA COMPRESSION ALG.

IT REPLACES REPEATED OCCURRENCES OF DATA WITH A REFERENCE TO A SINGLE COPY OF THAT DATA. IN ORDER TO FIND MATCHES, THE ALG. KEEPS TRACK OF SOME AMOUNT OF THE RECENT DATA (THE SLIDING WINDOW). A MATCH IS ENCODED AS A LENGTH-DISTANCE PAIR (IN THE EXAMPLE WE ALSO KEEP THE VALUE OF THE NEXT CHAR, BECAUSE IT MAKES DECODING EASIER)

EXAMPLE: LET'S IMAGINE THAT THE WINDOW IS THE WHOLE STRING



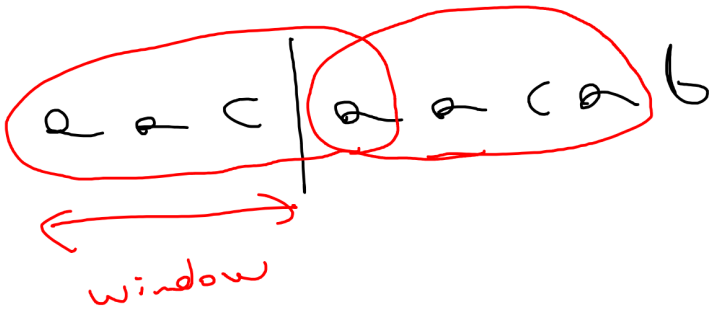
first occurrence

CAN BE ENCODED AS <6,3,a>
DIST LEN

THE NEXT 3 CHARS ARE EQUAL TO THE CHARS YOU SAW "6 CHARS AGO" | THEN THERE IS AN a AFTER

EXAMPLE

IT IS ALSO POSSIBLE TO HAVE PAIRS WITH <LEN> DIST, WHICH IS USED TO SAY TO THE DECODER THAT THE DATA I WANT TO DECOMPRESS NEEDS TO BE PASTED UNTIL NO SPACE REMAINS

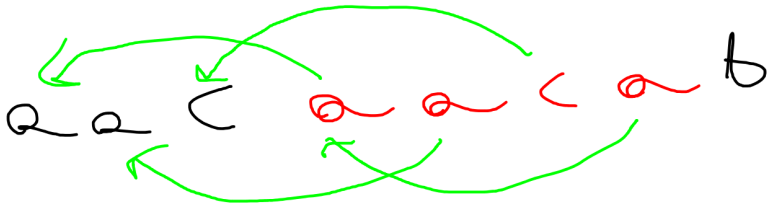


<3,4,b>

THE DECODER WILL FIND THIS:

a a c <3,4,b>

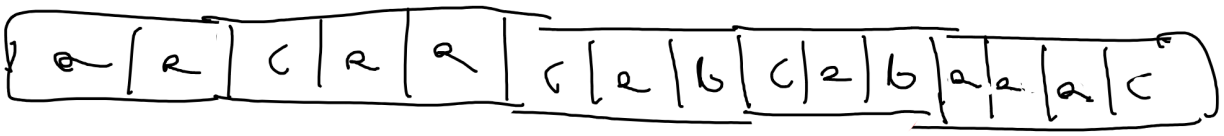
AND WILL DECODE AS:

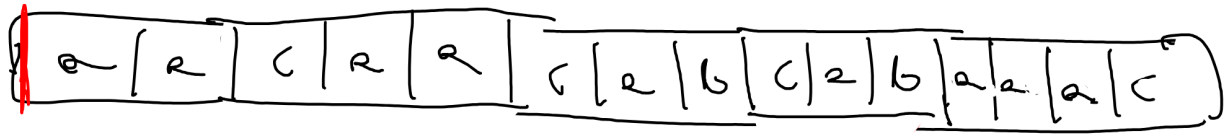


EXAMPLE: WITH WINDOW = 6

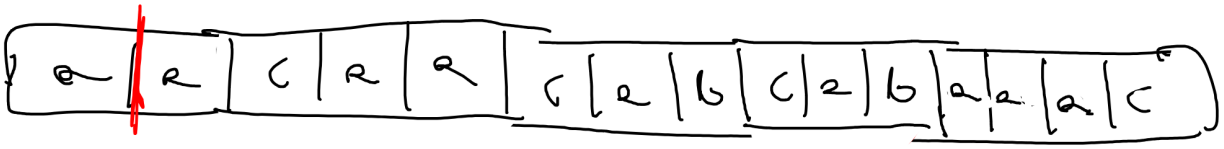
< dist, len, next char >

THE STRING:

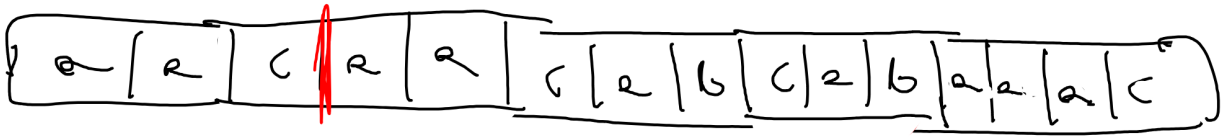




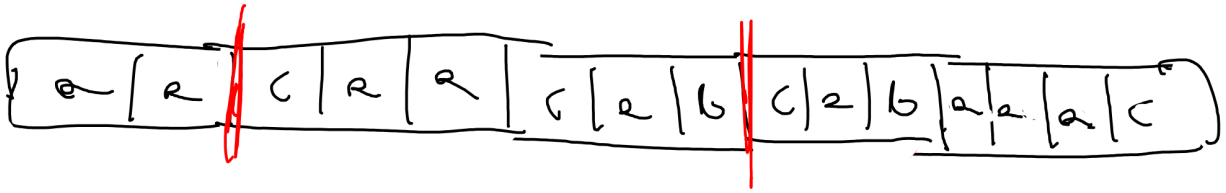
OUTPUT = < 0, 0, a >



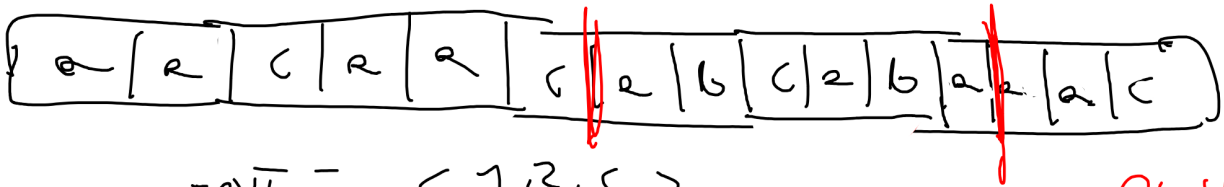
OUTPUT = < 1, 1, c > a matches with a



OUTPUT = < 3, 4, b > aac a matches with aac a



OUTPUT = < 3, 3, a >



OUTPUT = < 1, 2, c > FINISH

FINAL RESULT : < 0, 0, a > < 1, 1, c > < 3, 4, b > < 3, 3, a > < 1, 2, c >

EXAMPLE

DECODE THE RESULT ABOVE

→ TAKE THE TRIPLES, ONE BY ONE

→ DO WHAT THEY ORDER

$\langle 0, 0, a \rangle \langle 1, 1, c \rangle \langle 3, 4, b \rangle \langle 3, 3, a \rangle \langle 1, 2, c \rangle$

$a \langle 1, 1, c \rangle \langle 3, 4, b \rangle \langle 3, 3, a \rangle \langle 1, 2, c \rangle$

$a\ a \langle 3, 4, b \rangle \langle 3, 3, a \rangle \langle 1, 2, c \rangle$

$a\ a \langle a\ a \langle a\ b \langle 3, 3, a \rangle \langle 1, 2, c \rangle$

$a\ a \langle a\ a \langle a\ b \langle a b a \langle 1, 2, c \rangle$

$a\ a \langle a\ a \langle a\ b \langle a\ b\ a\ a\ a \langle$

COMPRESSION & NETWORKING

WE WANT TO MAKE DATA TRANSFER AS TRANSPARENT AS POSSIBLE TO THE USERS.

EVEN IF INTERNET IS NOW FAST, SOMEONE MAY BE STILL USING A SLOW CLIENT (MOBILE) OR BATTERY BASED.

TWO SOLUTIONS

CACHING = AVOID SENDING THE SAME OBJECT AGAIN

IF AN OBJECT CHANGE, EVEN SLIGHTLY, YOU HAVE TO RESEND IT

COMPRESSION = REMOVE REDUNDANCY IN TRANSMITTED DATA

BUT WHAT IF THE RECEIVER HAS NEVER SEEN THE ORIGINAL DATA BEFORE?

→ IF BOTH SENDER & RECEIVER KNOWS THE FILE (UNSTRUCTURED)

↘ DELTA COMPRESSION (diff)

SENDS ONLY THE DIFFERENCES BETWEEN THE REQUESTED PAGE AND THE ONES IN CACHE

FORMALLY

Z-DELTA COMPRESSION

Given two files f_{known}, known to both parties, and f_{new}, known only to sender, we want to compute f_d of minimum size, such that f_{new} can be derived by the receiver using f_{known} and f_d

FIND AN OPTIMAL COVERING OF f_{new} BASED ON f_{known}

OUR APPROACH

COMPRESS THE TEXT AS f_{known} f_{new} USING THE CZ77 SCHEME, STARTING THE ENCODE FROM f_{new}, WHILE f_{known} ACTS AS THE "previously encoded text"

EFFICIENT AND OPTIMAL SOLUTION !!!

USE: ZDELTA USED ALSO IN BACKUPS AND VERSION CONTROL

CLUSTER-BASED DELTA COMPRESSION

ZDELTA CAN BE ALSO USED TO COMPRESS A GROUP OF FILES F (USEFUL ON A DYNAMIC COLLECTION OF WEB PAGES OR BACKUPS...).

THE IDEA IS TO FIND A GOOD REFERENCE FOR EACH FILE $f \in F$ AND APPLY A PAIR-WISE ZDELTA

↓

THE PROBLEM REDUCES TO THE MIN. BRANCHING PROBLEM ON DAGS (DIRECTED ACYCLIC GRAPHS)

- BUILD A (COMPLETE?) WEIGHTED GRAPH G_F (NODES = FILES, WEIGHTS = zdelta size)
  # triples created during encoding
- INSERT A DUMMY NODE CONNECTED TO ALL AND THE WEIGHTS ARE THE gzip SIZE OF THE RESPECTIVE NODES
- COMPUTE THE DIRECTED SPANNING TREE OF MIN TOTAL COST THAT COVERS ALL G'S NODES (A PATH THAT CONNECTS ALL THE NODES)

SEE EXAMPLE

PROBLEM = CONSTRUCTING G_F IS COSTLY, BECAUSE IT REQUIRES n^2 edge calculation (the number of zdelta executions)

↳ SOLUTION

↳ PRUNING ?

- COLLECTION ANALYSIS = CLUSTER THE FILES THAT APPEAR SIMILAR, HENCE GOOD FOR ZDELTA, THEN BUILD A SPARSE WEIGHTED GRAPH G'_F CONTAINING ONLY EDGES BETWEEN PAIRS OF FILES IN THE SAME CLUSTER
- ASSIGN WEIGHTS = ESTIMATE APPROPRIATE EDGE WEIGHTS FOR G'_F, HENCE SAVING ZDELTA EXEC (HOWEVER, STILL n^2 TIMES)

SOME PERFORMANCES...

NO PRUNING

	SPACE	TIME
UNCOMPRESS.	30Mb	...
TGZ	20%	O(n)
CLUSTER ZDELTA	8%	$O(n^2)$

PRUNING

	SPACE	TIME
UNCOMPRESS.	250Mb	...
TGZ	12%	2min
CLUSTER ZDELTA	8%	16min

EXAMPLE

GIVEN 3 STRINGS $s1 =$ "abaco", $s2 =$ "dada" AND $s3 =$ "cobo", SHOW THE COMPRESSION USING Z-DELTA APPLIED TO THE GROUP OF THOSE STRINGS USING GZIP (LZ77) AND DERIVES THEIR CONCATENATION ORDER

(1) Apply gzip to the strings

$gzip(s1) = \langle 0,0,a\rangle\langle 0,0,b\rangle\langle 2,1,c\rangle\langle 0,0,o\rangle$ size 4

$gzip(s2) = \langle 0,0,d\rangle\langle 0,0,a\rangle\langle 2,2,\$\rangle$ size 3

$gzip(s3) = \langle 0,0,c\rangle\langle 0,0,o\rangle\langle 0,0,b\rangle\langle 2,1,\$\rangle$ size 5

(2) Apply zdelta pair-wise (ENCODE ONLY SECOND TERM!!)

$zdelta(s1|s2) = \langle 0,0,d\rangle\langle 4,1,d\rangle\langle 2,1,\$\rangle$ size 3

$zdelta(s1|s3) = \langle 2,2,a\rangle\langle 7,2,\$\rangle$ size 2

$zdelta(s2|s1) = \langle 1,1,b\rangle\langle 2,1,c\rangle\langle 0,0,o\rangle$ size 3

$zdelta(s2|s3) = \langle 0,0,c\rangle\langle 0,0,o\rangle\langle 3,1,b\rangle\langle 2,1,\$\rangle$ size 4

$zdelta(s3|s1) = \langle 3,3,c\rangle\langle 8,1,\$\rangle$ size 2

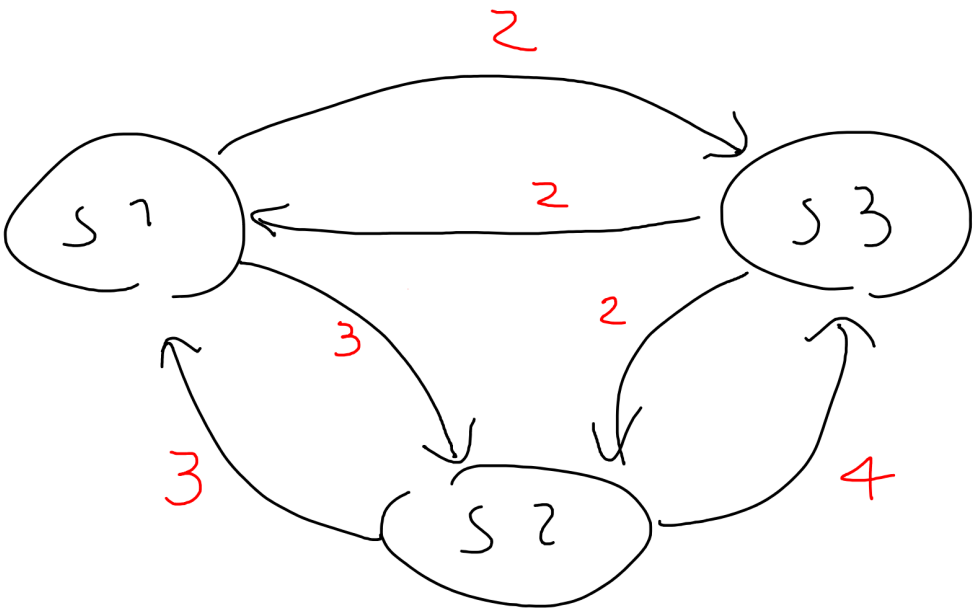
$zdelta(s3|s2) = \langle 0,0,d\rangle\langle 2,3,\$\rangle$ size 2

→ NICE OVERLAP

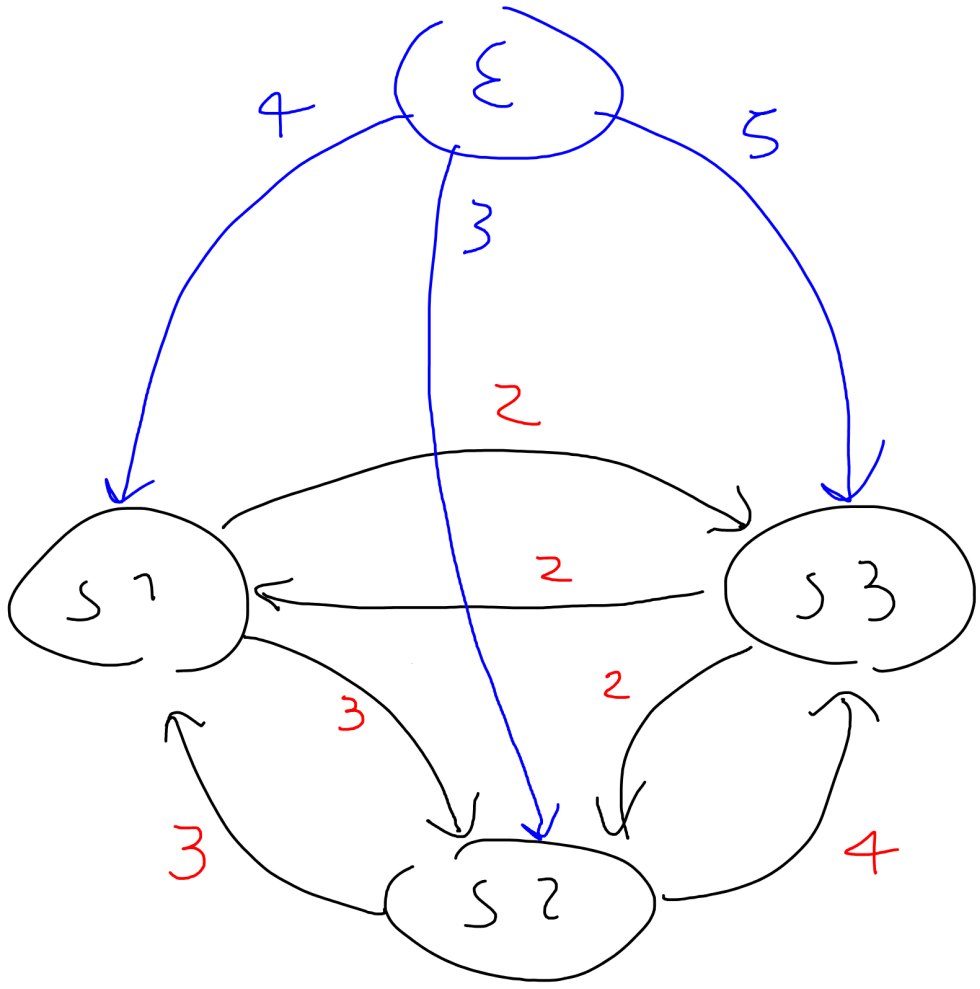
COABADADA

(3) BUILD THE WEIGHTED GRAPH

(nodes = FILES, WEIGHTS = ZDELTA SIZE)



(4) ADD DUMMY NODE ε



(5) FIND THE DIRECTED SPANNING TREE OF MIN TOTAL COST

NOTE = YOU HAVE TO START FROM ε

TWO POSSIBILITIES

- $\varepsilon \longrightarrow S2 \longrightarrow S1 \longrightarrow S3$ COST 8
- $\varepsilon \longrightarrow S1 \longrightarrow S3 \longrightarrow S2$ COST 8

I'm taking the second path

zdelta compression will compress like

gzip(S_1), zdelta(S_1/S_3), zdelta(S_3/S_2)

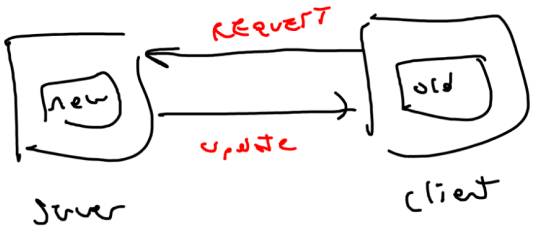
→ IF THERE IS ONLY A PARTIAL KNOWLEDGE BETWEEN SENDER & RECEIVER

UNSTRUCTURED FILES → FILE SYNCRONIZATION (RSYNC, ZSYNC)

RECORD-BASED DATA → SET RECONCILIATION

Client request for a new version of a file. Server sends only the updated parts, without knowing the old file

CLIENT UPDATES STRUCTURED FILE, WHICH IS AVAILABLE ON A SERVER



FILE SYNCRONIZATION WITH RSYNC

1. Client divides old file in block of size B and hashes each of them
2. Client sends $\frac{\text{old size}}{B}$ hashes to the server
3. The server identifies the mismatching parts of the files by hash rolling B chars: from left to right, substitute the hashes where possible
4. Server sends back a concatenation of literals and hashes know by the client, which forms the updated file

- Single round-up
- Not good in theory (granularity problem

EXAMPLE

Server has f_{new} = CABADDACCC and client has f_{old} = ABACCCDAC , $B = 3$

(1) f_{old} = ABA | CCC | DAC
h_1 h_2 h_3

(2) client $\xrightarrow{h_1, h_2, h_3}$ server

(3) HASH ROLLING f_{new} TO COMPUTE f_d

f_d = C ~~ABA~~ D ~~DAC~~ CC
h_1 h_3

(4) Server $\xrightarrow{C\, h_1\, D\, h_3\, CC}$ Client

Client is able to build f_{new}

FILE SYNCHRONIZATION WITH ZSYNC

The idea is to reduce the workload on the server (sort of symmetrical approach to rsync)

(1) Server divides new file in B blocks and hashes each of them

(2) Server sends $\frac{\text{new size}}{B}$ hashes to client

(3) Client identifies the missing parts by hash rolling B chars and then sends to the server a bitmask to request them (size = # hashes received = $\frac{\text{new size}}{B}$)
(B[i]=0 if client doesn't own hash i, else 1)

(4) Using the bitmask, the server is able to understand what the client only and what not, therefore it zdelta compresses the missing blocks according to the owned ones

AND SENDS THE ENCODED FILE.

Example

Server has f_{new} = ABRACALABRIA and client has f_{old} = ABRACADABRA, $B = 3$

(1) f_{new} = ABR | ACA | LAB | RIA = h_1 h_2 h_3 h_4

(2) Server $\xrightarrow{h_1, h_2, h_3, h_4}$ client

(3) HASH ROLLING fold

ABRACADABRA

h_1 h_2 h_1

bitmask = 1100

Client $\xrightarrow{1100}$ Server

(4) Server does zdelta compression of h_3h_4, taking h_1h_2 as known text

$gzip(h_1h_2 \mid h_3h_4) = gzip(\text{ABRACA} \mid \text{LABRIA}) =$

$= \langle 0,0,L \rangle \langle 7,3,1 \rangle \langle 4,1,\$ \rangle$

Sever $\xrightarrow{\langle 0,0,L \rangle \langle 7,3,1 \rangle \langle 4,1,\$ \rangle}$ Client

CLIENT IS NOW ABLE TO BUILD f_{new}