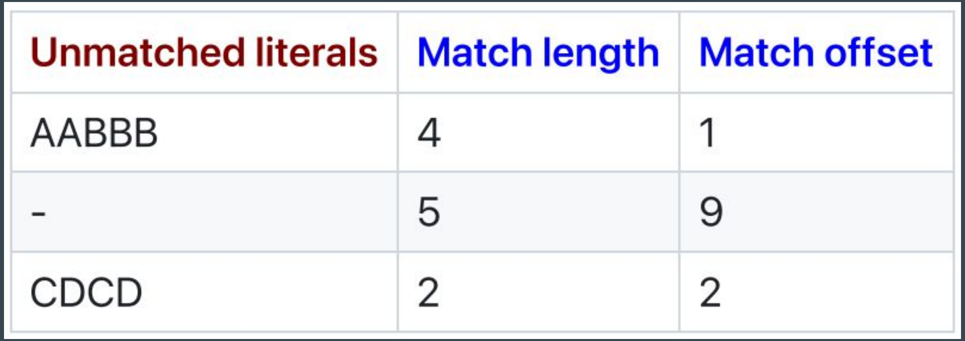
LZ77 Optimization

•••

Pranavi Boyalakuntla and Lycia Tran

About the LZ77 Algorithm

- LZ77 is the basis for some of the most widely used compressors
- Works by replacing later iterations of a previously existing pattern by encoding information about how far back the pattern exists and how long the pattern was in a table
- Unmatched patterns, or "new" patterns are included in the table as is



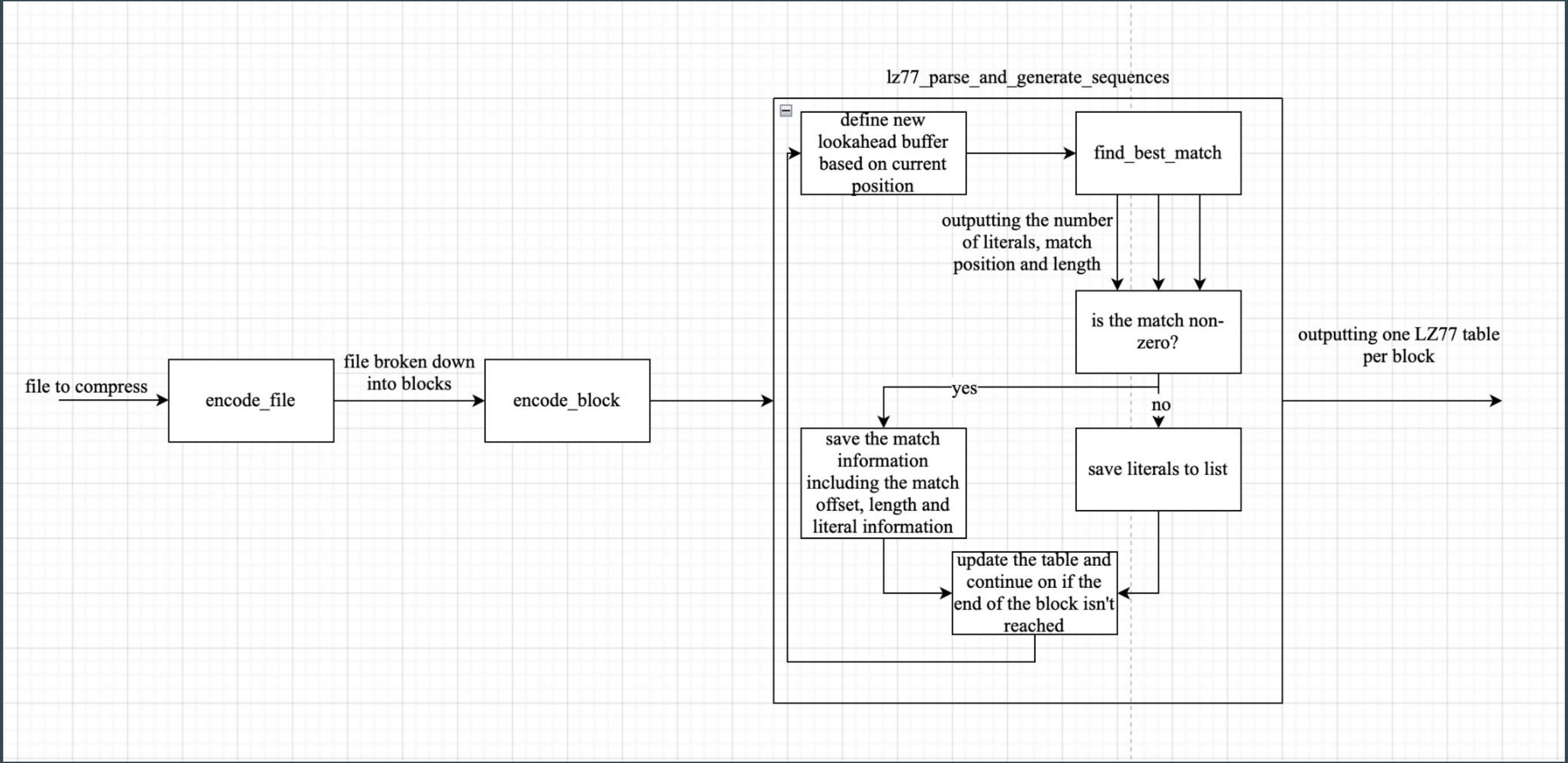
Unmatched literals	Match length	Match offset
AABBB	4	1
-	5	9
CDCD	2	2

Example of LZ77 table

Current LZ77 Implementation in SCL

- Used the LZ77 Sliding Window Implementation in SCL
- Utilizes a custom LZ77Window class provides utilities for a given byte array
    - Keeps track of the current window (start and end pointers)
- Utilizes a custom match finder with options for two types:
    - Base match finder
    - Hash based match finder
- Hash based match finder stores hashes of all the windows
- Uses Huffman coding for entropy coding
- Does not implement repcodes

Current LZ77 Implementation in SCL


lz77_parse_and_generate_sequences
file to compress
encode_file
file broken down into blocks
encode_block
define new lookahead buffer based on current position
find_best_match
outputting the number of literals, match position and length
is the match non-zero?
yes
no
save the match information including the match offset, length and literal information
save literals to list
update the table and continue on if the end of the block isn't reached
outputting one LZ77 table per block

Implemented Optimization Strategies - Entropy Coders

- Change the Huffman entropy coding to rANS, tANS, Arithmetic coding
- rANS
  - Provides a good tradeoff between compression rate and speed
  - Encodes state value and counts
- tANS
  - Provides a good tradeoff between compression rate and speed
  - Variation of cached rANS
  - Encodes state value and counts (as a lookup table)
- Arithmetic coding
  - Data is encoded into a single block
  - No codebook is necessary, since codeword computed on the fly
  - Compression often close to entropy

Implemented Optimization Strategies - Repcodes

- Helps compress structured data more efficiently
  - Works when the same offset is used frequently
- Use repcodes to cheaply determine matches by having a “seed” for the most likely matches
- Example:
  - Start with rep1, rep2, rep3 with 1, 4, and 8 respectively
  - Continuously update the 3 repcodes based on the last 3 used match offsets

Compression Results on Sherlock Holmes File

Uncompressed file size: 362308 bytes

	Huffman		rANS		Arithmetic	
	w/o repcodes	w/ repcodes	w/o repcodes	w/ repcodes	w/o repcodes	w/ repcodes
File Size (bytes)	123010	126643	121291	125992	121262	125962
Encode/ Decode Speed (s)	7.57	7.22	9.77	8.91	192.27	181.01

Additional Results - Github User Data

Uncompressed file size: 243730 bytes

	Huffman		rANS		Arithmetic	
	w/o repcodes	w/ repcodes	w/o repcodes	w/ repcodes	w/o repcodes	w/ repcodes
File Size (bytes)	10475	14800	10283	14744	10261	14722
Encode/ Decode Speed (s)	4.17	3.93	4.69	4.39	131.20	118.99

Additional Results - Random ASCII File

	Huffman		rANS		tANS		Arithmetic	
	w/o repcodes	w/ repcodes	w/o repcodes	w/ repcodes	w/o repcodes	w/ repcodes	w/o repcodes	w/ repcodes
Encode/ Decode Speed (s)	0.55	0.52	0.53	0.51	1639.58 (27.32 min)	1372.97 (22.88 min)	0.57	0.51