Exact string matching algorithms

Branimir Lazarević 3156/2017
Dimitrije Petrović 3165/2017

Content

- Algorithm overview
- Testing and test results
- Algorithm comparison and conclusion

Exact string matching algorithms

- Search for pattern (string) occurrences in a larger string
- Used in genome processing when looking for specific base sequence locations inside the genome - *alignment*
- Tested algorithms (implemented using Python):
  - **Sorted Index**
  - **Hash Table**
  - **Suffix Array**
  - **Suffix Tree**
- Off-line algorithms, preprocess input text, use various index structures

Sorted Index

- Create index:
  - Extract all substrings with specific length from input string and remember their positions
  - Store (substring, location) pairs in a list structure
  - Sort the list alphabetically
- Query for pattern:
  - Use binary search
  - Return all *possible* positions for the pattern in input string

T: CGTGCGTGCTT

<u>5-mer index of T:</u>

CGTGC, 0
CGTGC, 4
GCGTG, 3
GTGCC, 1
GTGCT, 5
TGCCT, 2
TGCTT, 6

Hash Table

- Create index:
  - Extract all substrings with specific length from input string and remember their positions
  - Store substring: [locations] pairs in a dictionary
- A Python dictionary is basically a hash table
- Query for pattern:
  - Direct access to the Python dictionary
  - Return all *possible* positions for the pattern in input string

T: CGTGCGTGCTT

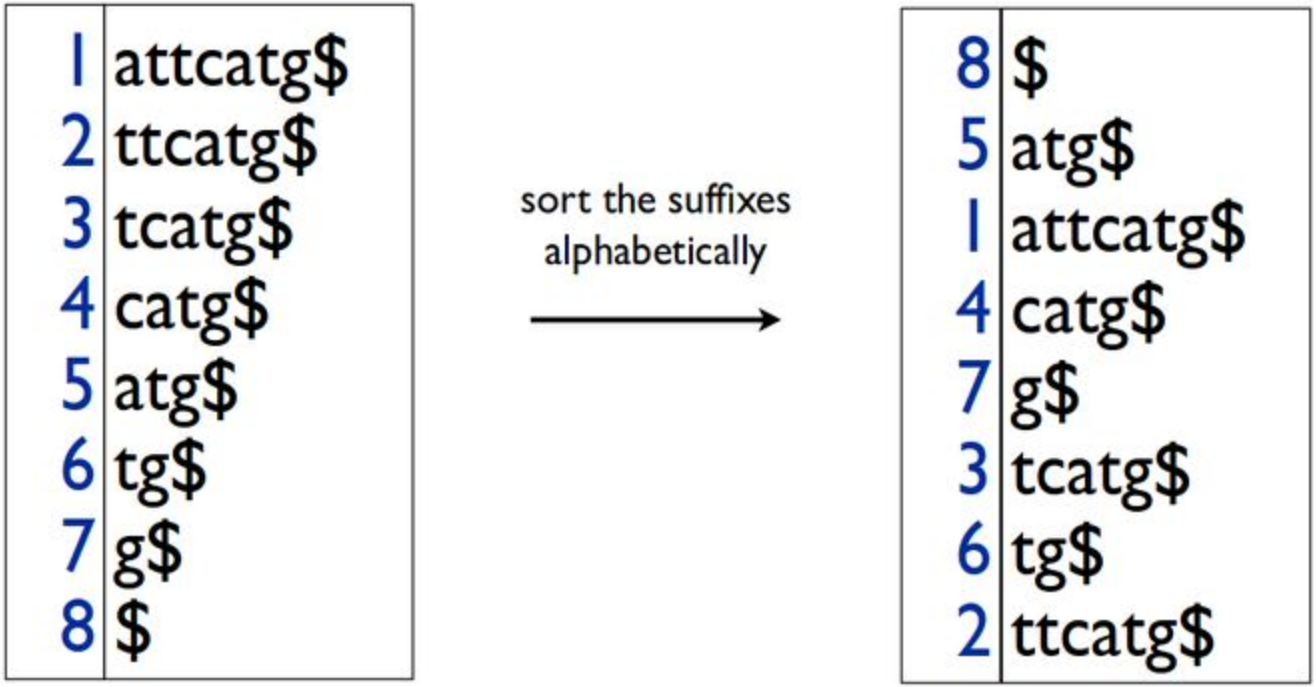
5-mer dictionary of T:

CGTGC: 0, 4
GCGTG: 3
GTGCC: 1
GTGCT: 5
TGCCT: 2
TGCTT: 6

Suffix Array

- Create index:
  - Make list of suffixes from input string
  - Store only suffix positions in a list structure
  - Sort suffix positions according to alphabetical order of suffixes
- Query for pattern:
  - Use binary search
  - Return all positions for the pattern in input string

Suffix Array (ATTCATG)



Suffix Tree

- Compressed trie of all suffixes
- Create index:
  - Generate all suffixes of given text
  - Consider all suffixes as individual words and build a compressed trie
  - Build a suffix tree using McCreight O(n) algorithm
- Query for pattern:
  - Starting from the first character of the pattern and root of Suffix Tree
  - For the current character of pattern, if there is an edge from the current node of suffix tree, follow the edge
  - Return all positions for the pattern in input string

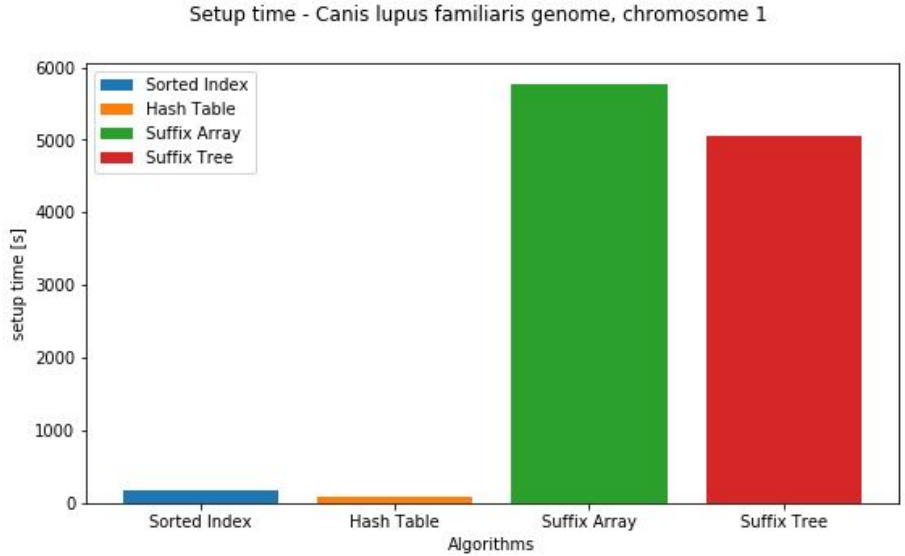
Testing

- Tested algorithms on three different genomes given as FASTA format files:
  - Canis lupus familiaris genome, chromosome 1
  - Phoenix dactylifera genome
  - Ananascomosus genome, chromosome 1
- Time needed to parse each FASTA file:

File 1	File 2	File 3
1.34 s	6.42 s	0.259 s

- Searched for three patterns: ATGATG, CTCTCTA, TCACTACTCTCA

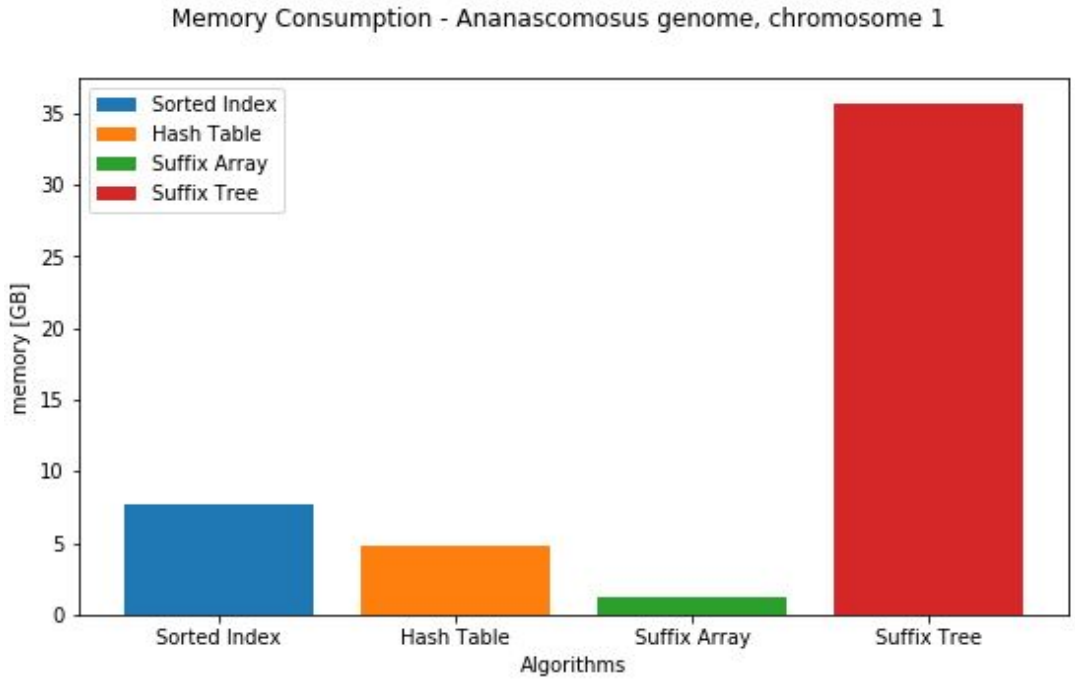
Preprocessing - index creation



Sorted Index	Hash Table	Suffix Array	Suffix Tree
180 s	92 s	5768 s	5054 s

Tested only for file 1

Memory consumption



File 3 shown

Memory consumption

Memory consumption	File 1	File 2	File 3
Sorted Index	19.34 GB	86.75 GB	7.7 GB
Hash Table	7.74 GB	31.02 GB	4.76 GB
Suffix Array	5.47 GB	/	1.26 GB
Suffix Tree	102.51 GB	/	21.59 GB

- File 2 not tested for Suffix Array - creation time too long
- File 2 not tested for Suffix Tree - creation time too long, memory consumption too high

Query time - Sorted Index

Sorted Index	Pattern 1	Pattern 2	Pattern 3
File 1	131 ms	187 ms	110 ms
File 2	1.24 s	955 ms	831 ms
File 3	18.6 ms	19.6 ms	10.2 ms

- File 1 = Canis lupus familiaris genome, chromosome 1
- File 2 = Phoenix dactylifera genome
- File 3 = Ananascomosus genome, chromosome 1

Query time - Hash Table

Hash Table	Pattern 1	Pattern 2	Pattern 3
File 1	81.1 ms	126 ms	84 ms
File 2	660 ms	512 ms	421 ms
File 3	13.7 ms	15.8 ms	7.67 ms

- File 1 = Canis lupus familiaris genome, chromosome 1
- File 2 = Phoenix dactylifera genome
- File 3 = Ananascomosus genome, chromosome 1

Query time - Suffix Array

Suffix Array	Pattern 1	Pattern 2	Pattern 3
File 1	62.6 ms	18 ms	472 µs
File 2	/	/	/
File 3	15 ms	6.1 ms	418 µs

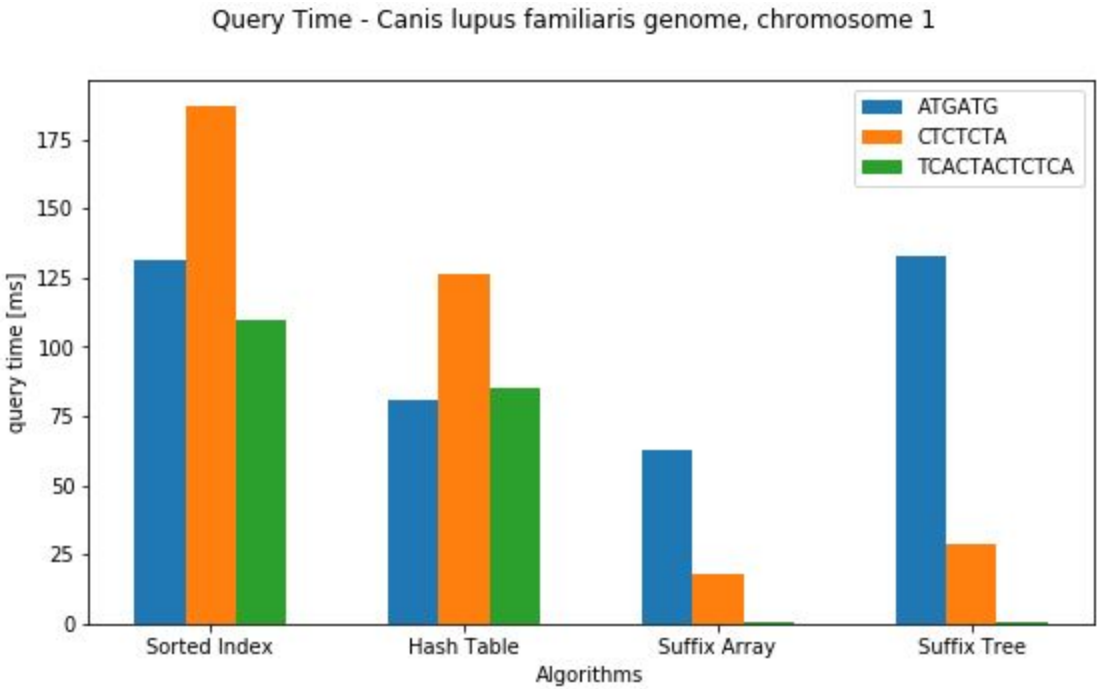
- File 1 = Canis lupus familiaris genome, chromosome 1
- File 2 = *not measured*
- File 3 = Ananascomosus genome, chromosome 1

Query time - Suffix Tree

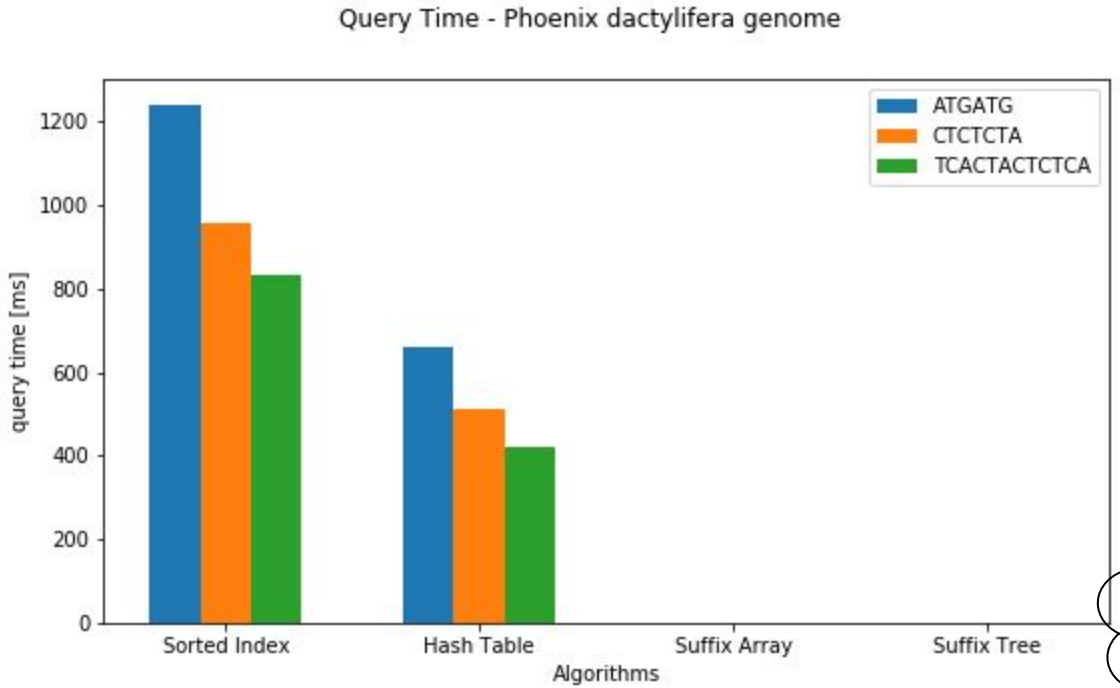
Suffix Tree	Pattern 1	Pattern 2	Pattern 3
File 1	133 ms	28.4 ms	506 µs
File 2	/	/	/
File 3	23.5 ms	8.81 ms	349 µs

- File 1 = Canis lupus familiaris genome, chromosome 1
- File 2 = *not measured*
- File 3 = Ananascomosus genome, chromosome 1

Query comparison - file 1

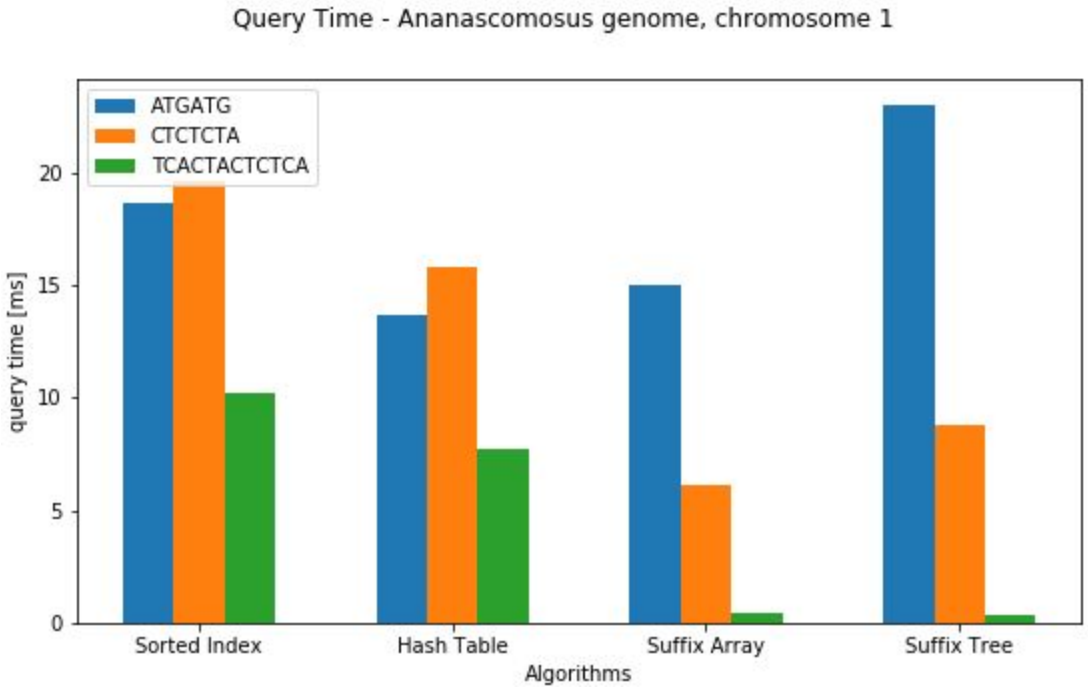


Query comparison - file 2



Suffix algorithms not tested

Query comparison - file 3



Conclusion

- Fast preprocessing (Hash Table and Sorted Index) vs slow preprocessing (Suffix Array and Suffix Tree)
- Hash Table is better than Sorted Index in all aspects - preprocessing is faster, memory consumption is lower and query time is faster for all files and patterns
- Suffix Array is better than Suffix Tree in all aspects, except a slightly slower preprocessing time
- Suffix Tree algorithm requires too much memory
- Suffix algorithms show much better query times, especially with longer patterns and should always be used when the initial preprocessing time isn’t important

Thank you for your attention!