An Improved Query Time for Succinct Dynamic Dictionary Matching

Guy Feigenblat[12]
Ely Porat[1]
Ariel Shiftan[1]

[1]Bar-Ilan University

[2]IBM Research Haifa labs

Agenda

- Dictionary Matching
- Aho-Corasick
- Motivation
- Succintness
- Dynamic Dictionary Matching
- Dynamic Dictionary Matching Algorithm
- Succinct Aho-Corasick automaton construction using a compact working space

Dictionary Matching

- ∑ is the alphabet, |∑| = σ
- Input:

  Text $T = t_1 t_2 \ldots t_m$

  Dictionary of patterns D = $\{P_1, P_2, \ldots, P_d\}$, |D|=n

  All characters in patterns and text belong to ∑
- Output:

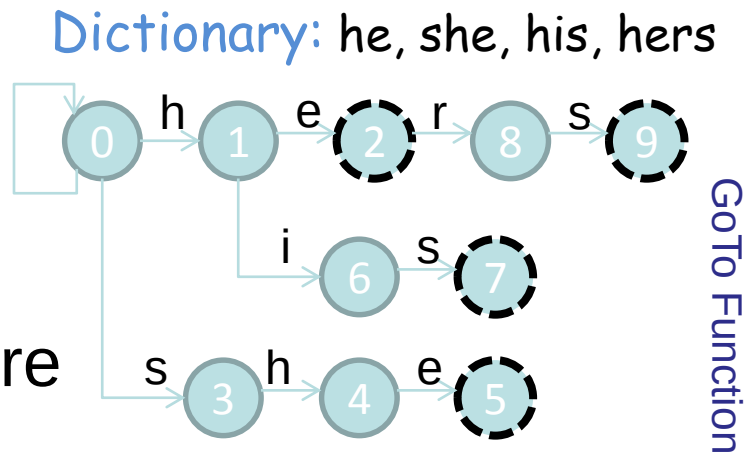
  All occurrences of patterns in T

Dictionary Matching Solution

- Run KMP algorithm d times
  - Very inefficient – run time of O(|T|d) (assuming |T|>|D|)
- Alternative solution- Generalized version of KMP
  - Aho-Corasick
    - ✓Space – O(|D|) words
    - ✓ Query time – O(|T| + occ)

Aho-Corasick (1975)

- Goto function – a trie of patterns (with ≤ n states)
- Failure function – for each pattern's prefix
  the longest suffix which is
  a prefix of any pattern
- Report – for each state,
  list of all patterns which end there



Report Function

i	2	9	7	5	ow
r(i)	1	1	1	1	0

Failure Function

i	1	2	3	4	5	6	7	8	9
f(i)	0	0	0	1	2	0	3	0	3

Research Motivation

Reduce the storage

- O(n) words = O(nlogn) bits
  - Can be up to O(logn) more than patterns length

- For small Alphabet the space can be O(n log n) bits vs. O(n) input.

Succintness

A succinct data structure uses an amount of space that is almost the information-theoretic lower bound.

Say M the information-theoretical number bits need to store some data, then :

Implicit	Uses M + O(1) bits
Succinct	Uses M + o(M) bits
Compact	Uses O(M)
Compressed	Depends on the entropy of the data being stored $H_0 = -\sum_{i=1}^{\sigma} \frac{n_i}{\|s\|} \log \frac{n_i}{\|s\|}$

Static Solution - Belazzougui 2010

- Encode the automaton

  Use 3 kind of succinct data structures:

  – Succinct indexable dictionary (Goto)

  – Succinctly encoded navigable tree (Failure)

  – Succinctly encoded integer array (Report)
- Space – $n(\log \sigma + O(1)) + O(d \log n/d)$ bits
- Optimal query time – $O(|T| + occ)$

Comparison of results

Bits of space	
Hon 08’	$O(n\log(\sigma))$
Hon 08’	$n\log(\sigma)(1+o(1))+O(d\log(n))$
Sadakane 00’	$n\log(\sigma)(2+o(1))+O(d\log(n))$
Belazzougui 10’	$n(\log(\sigma)+3.443+o(1))+O(d\log(\frac{n}{d}))$

Time	
Hon 08’	$O(\lvert T\rvert\log\log(n)+occ)$
Hon 08’	$O(\lvert T\rvert\log^{\varepsilon}(n)+\log(d))+occ)$
Sadakane 00’	$O(\lvert T\rvert\log(d)+\log(\sigma))+occ)$
Belazzougui 10’	$O(\lvert T\rvert+occ)$

Dynamic Dictionary Matching

- Σ is the alphabet, $|\Sigma| = \sigma$
- Input:

  Text $T = t_1 t_2 \ldots t_m$

  Dictionary of patterns $D = \{P_1, P_2, \ldots, P_d\}$, $|D| = n$

  All characters in patterns and text belong to Σ.

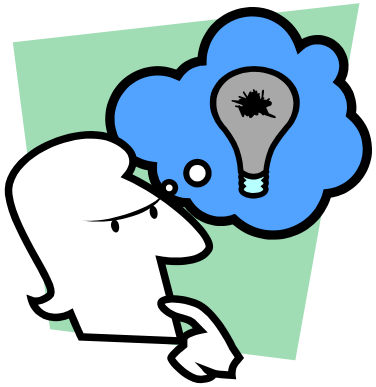
  Patterns can be inserted or removed

- Output:

  All occurrences of patterns in T

Solution for the dynamic case

- Too hard to have one succinct DS
  - insertion and deletion should utilize only succinct working set



Idea:

- Conceptually divide the patterns into hierarchy of groups according to their length.
- On insertion pattern will be added to a group according to its length.
- When a group becomes “full” - merge it to a larger group.

Main Contribution

- Succinct dictionary matching structure with only $O(\log\log n)^2$ slow down in query time (for poly-logarithmic alphabet)
- Insertions and deletions in $O(1/\varepsilon \log n)$ slow down
- Succinct Aho-Corasick automaton construction with only a compact working space

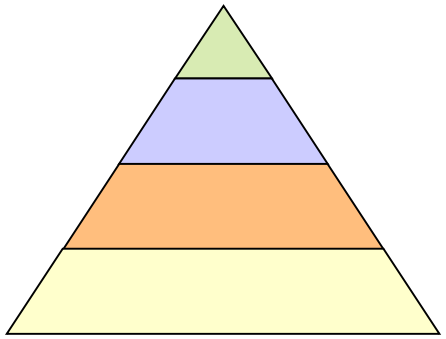
Comparison of results (dynamic case)

	query time	update time	space
[8]	$O((\|T\| + occ)\log^2 n)$	$O(\|P\|\log^2 n)$	$O(n\sigma)$
[7]	$O(\|T\|\log n + occ)$	$O(\|P\|\log\sigma + \log n)$	$O(n\log\sigma)$
[7]	$O(\|T\|\log n + occ)$	$O(\|P\|\log\sigma + \log n)$	$(1+o(1))n\log\sigma + O(d\log n)$
This paper	$O(\|T\|(\log\log n)\log\sigma + occ)$	$O(\frac{1}{\epsilon}\|P\|\log n)$	$(1+\epsilon)(n\log\sigma + 6n) +$ $o(n) + O(d\log n)$
This paper, for $\sigma = polylog(n)$, using $\epsilon = \frac{1}{\log\log n}$	$O(\|T\|(\log\log n)^2 + occ)$	$O(\|P\|\log n\log\log n)$	$(1+o(1))n\log\sigma + O(d\log n)$

where n is the total length of d patterns, σ is the alphabet size and $0<\varepsilon<1$. The space is given in bits

[8] Chan, Hon, Lam, Sadakane: Dynamic dictionary matching and compressed suffix trees. SODA 2005
[7] Hon, Lam, Shah, Tam, Vitter : Succinct index for dynamic dictionary matching. ISAAC 2009

Dynamic Hierarchy



The hierarchy is divided into four groups:

- Group XL (extra large)

$$|\sum \text{patterns\_length}| \geq \frac{n}{\log \log n}$$

- Group L (Large)

$$\frac{n}{\log n \log \log n} \leq |\sum \text{patterns\_length}| < \frac{n}{\log \log n}$$

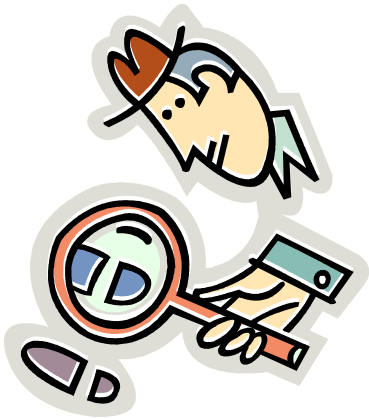
- Group M (Medium)

$$0.5 \log_\sigma n < |\sum \text{patterns\_length}| < \frac{n}{\log n \log \log n}$$

- Group S (Small)

$$|\sum \text{patterns\_length}| \leq 0.5 \log_\sigma n.$$

Some Observations



1) The total length of the patterns in groups "Medium" and "Small" is relatively Small.
   - Not restricted to use only succinct data-structures

2) It is possible to utilize static structures
   - Defer insertions and deletions. Use static DS in each block, reconstruct it every once in a while

3) Reconstruct the structure in segments
   - Not restricted to use succinct DS relative to the size of the segment

Extra Large Group

By definition, there can be at most loglogn such patterns.

utilize the constant space KMP of [Rytter02] for each pattern separately

Query time - O(|T| log log n + occ)

Insertion or deletion- for pattern P, O(|P|)

Large Group

Divide the patterns in this group into loglogn levels:

$$\frac{n}{2^i \log\log n} < |t_i| \leq \frac{n}{2^{i+1} \log\log n}$$

|Ti| is the total length of patterns in level i

Pattern in the i-th level is either:

1. a pattern that was originally inserted into this level.
2. a pattern that was moved into this level from lower levels

Each level contains two static succinct Aho-Corasick automatons

Large Group

Query:

Iterate over all of the levels and search for the pattern.

The total time is $O(|T| \log\log n * \log\sigma)$:

- $\log\log n$ levels
- query each Aho-Corasick in $O(|T| \log \sigma)$

Delete:

First locate the pattern in the sub structures; then, we "remove" the pattern from the Aho-Corasick automaton in O(1) time.

Remove only from the report tree

Large Group

Insertion:

Pattern P is inserted.

1. Construct a succint Aho-Corasick automaton
2. Insert it into level j, chosen according to its size.

More than two structures in level j:

- Merge them by building a new Aho-Corasick automaton that contains all of the patterns.
  - Can be done in compact working space. How?
- Put the resulting structure into an upper level

Large Group

Time analysis:

the amortized insert cost is O(|P| log n)

Space analysis:

The total size is $n \log \sigma + 6n + o(n) + O(d \log n)$

Medium and Small Groups

- Small group cannot exceed o(n)
- Medium group can be moved to smallest level in Large group when it becomes full

- We are not restricted to succinct DS
  - Use “traditional” dynamic algorithm
    - Such as [Sahinalp and Vishkin 96’], which is dynamic and optimal in time

Query time - O(|T| + occ)

Insertion or deletion- for pattern P, O(|P|)

Putting it All Together

Problems:

- Groups boundaries change
- Patterns are not actually deleted

➢ Based on standard amortization argument, we reconstruct the DS every pre defined threshold

- Query time – $O(|T| \log\log n \log\sigma + occ)$
- Insert - $O(1/\varepsilon\ |P| \log n)$
- Delete $O(1/\varepsilon\ |P| \log n)$

Remaining Obstacle

Construct a succinct Aho-Corasick automaton using only compact working space

- A sketch is described in our paper
- Complete construction is left for the full paper

Summary

- Open Questions
  - Can the dynamic case be further improved?
  - Can we reduce insertion/deletion time without affecting the query time?

- Any questions?