# Succinct dictionary matching with no slowdown

Djamal Belazzougui, dbelaz@liafa.jussieu.fr

LIAFA, Univ. Paris Diderot - Paris 7

# Dictionary matching problem

- Set of $d$ patterns (strings): $S = \{s_1, s_2, \ldots, s_d\}$.
- $\sum_{i=1}^{d} |s_i| = n$ characters from an alphabet of size $\sigma$.
- Queries: text $T \longrightarrow$ occurrences of patterns in $S$.

# Aho-Corasick automaton

- Classical solution to dictionary matching $\longrightarrow$ Aho-Corasick automaton with $m \leq n + 1$ states.
- Space usage $\longrightarrow$ $O(m)$ words of memory.
- Query time $\longrightarrow$ $O(|T| + occ)$ for resporting $occ$ occurences.

# Aho-Corasick automaton (Space usage)

- Aho-Corasick automaton $\longrightarrow$ too much space:

  $O(m)$ words$=O(m \log m)$ bits.

- But patterns occupy $n \log \sigma$ bits only.

- If $m = \Omega(n)$ and $\sigma = O(1)$:
  - Automaton uses $\Omega(n \log n)$ bits.
  - While the patterns occupy $O(n)$ bits only.
  - The automaton uses $\log n$ times the space of the patterns.

# Our result

- AC automaton encoded in just $m(\log \sigma + O(1)) + O(d \log(n/d)) \longrightarrow O(n \log \sigma)$ bits in the worst case.
- Query time still $O(|T| + occ)$.
- Use three kinds of succinct data structures:
  - 2 Succinct indexable dictionaries.
  - 2 Succinctly encoded navigable trees.
  - 1 Succinctly encoded integer array.

# Related work

| Algorithm | Query time |
|---|---|
| HLSTV1 | $O(\|T\| \log\log(n) + occ)$ |
| HLSTV2 | $O(\|T\|(\log^{\epsilon}(n) + \log(d)) + occ)$ |
| TWLY | $O(\|T\|(\log(d) + \log\sigma) + occ)$ |
| Ours | $O(\|T\| + occ)$ |

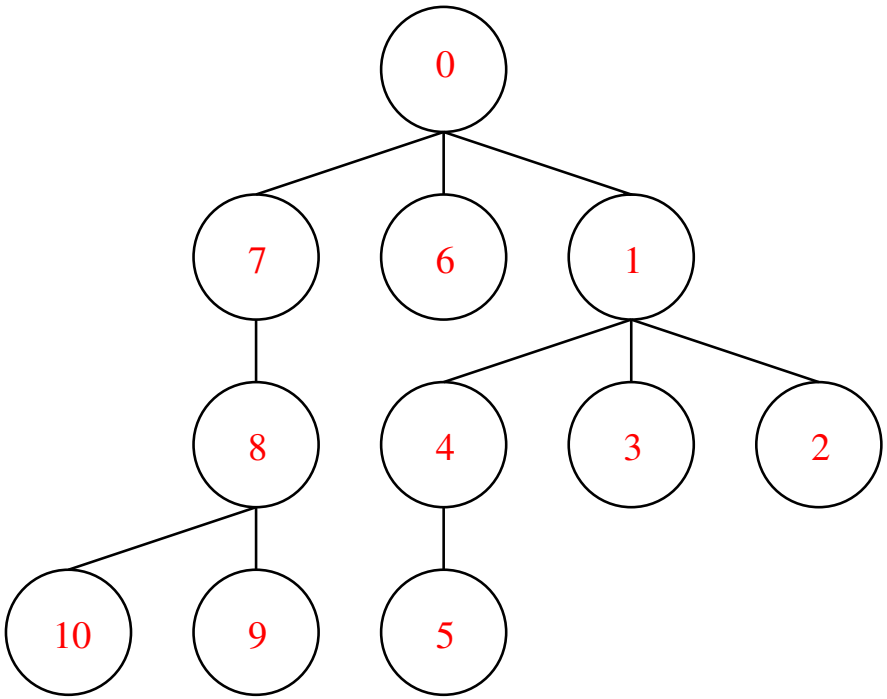
Table: Query time of succinct indexes dictionary matching solutions

| Algorithm | Space usage (in bits) |
|---|---|
| HLSTV1 | $O(n \log\sigma)$ |
| HLSTV2 | $n\log\sigma(1 + o(1)) + O(d\log(n))$ |
| TWLY | $n\log\sigma(2 + o(1)) + O(d\log(n))$ |
| Ours | $n(\log\sigma + 3.443) + O(d\log(n/d))$ |

Table: Space usage of succinct dictionary matching indexes

# Tools: Succinct navigable trees

- A tree of $n$ nodes $\longrightarrow$ $n(2 + o(1))$ bits.
- Many navigation operations in constant time.
- Nodes can be addressed in DFS lexicographic order.
- We need one operation :
  A node of DFS label $i$ $\longrightarrow$ the DFS label of its parent.

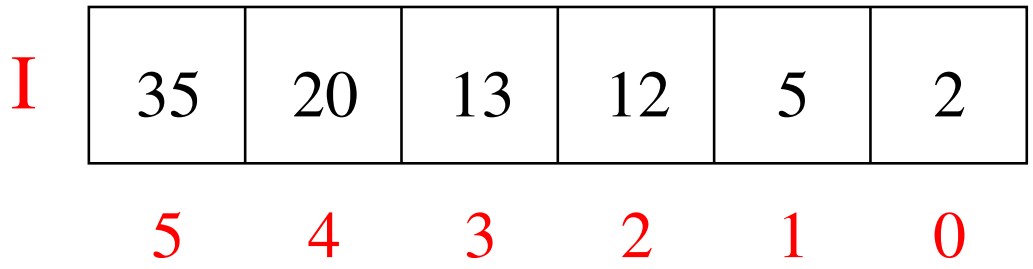


# Tools: succinct indexable dictionary

| I | 35 | 20 | 13 | 12 | 5 | 2 |
|---|---|---|---|---|---|---|
| | 5 | 4 | 3 | 2 | 1 | 0 |

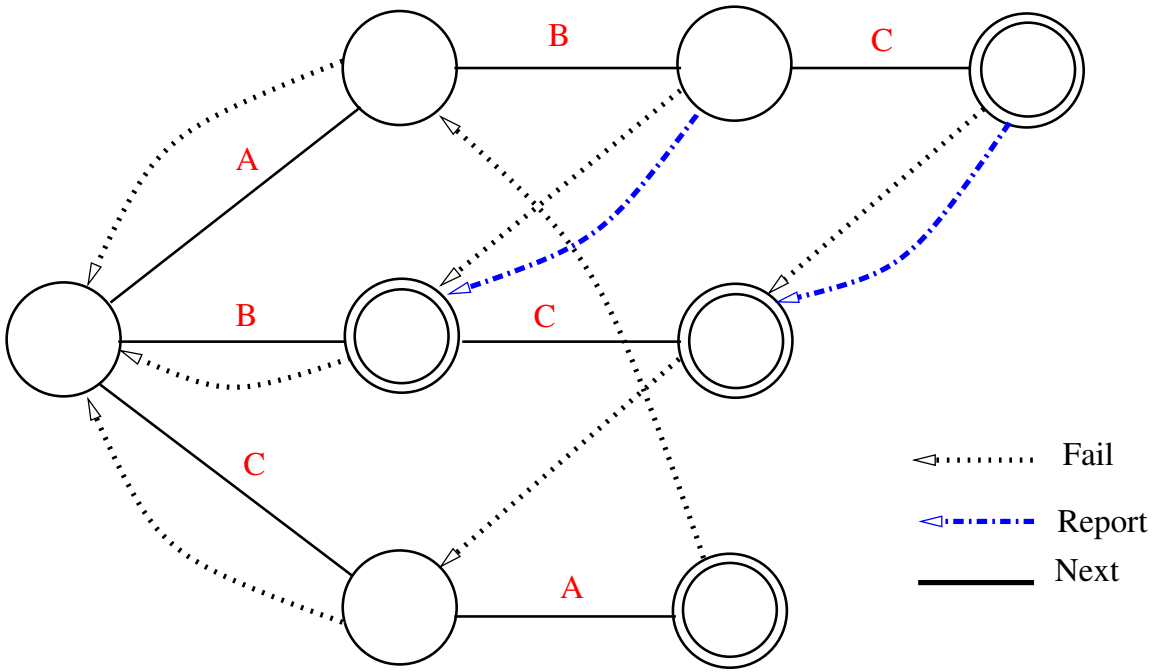
- Given a set of integers $I$, such that $I \subseteq [0, U-1]$ and $|I| = t$.
- $rank(x)$:
  - $x \in I \longrightarrow$ number of elements in $I$ smaller than $x$ (example $rank(13) = 3$).
  - $x \notin I \longrightarrow -1$ (example $rank(16) = -1$)

# Tools: succinct indexable dictionary



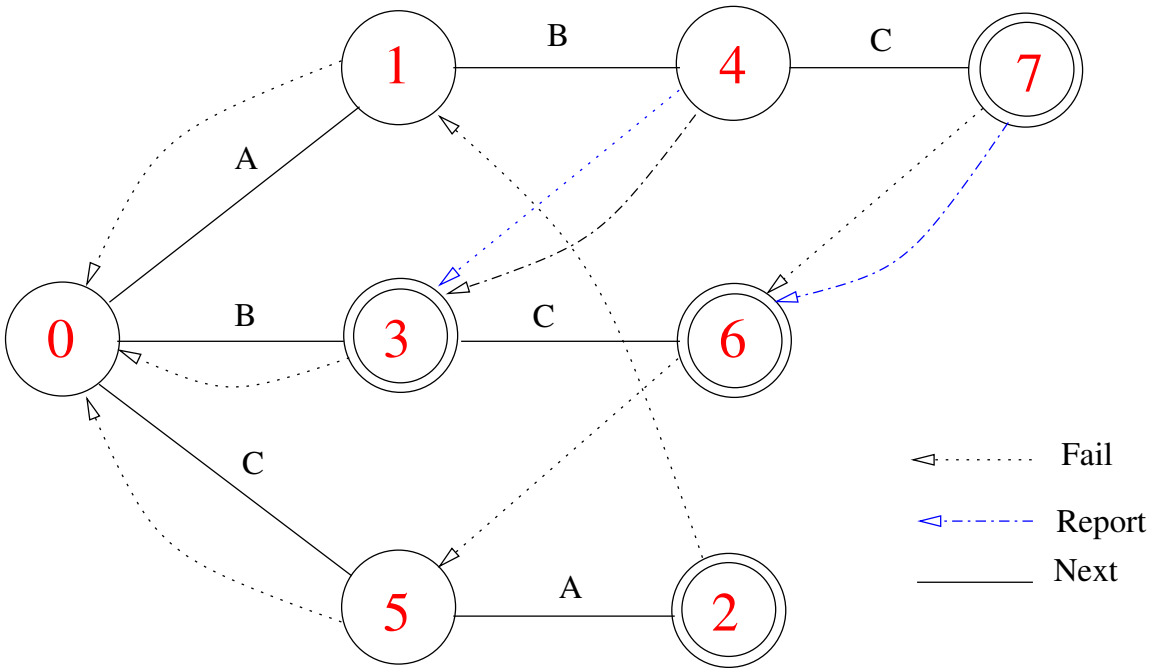

- comparison-based machine model $\longrightarrow$ sorted table: $t \log U$ bits and rank in $O(\log t)$ time (binary search).
- In the RAM (integer) model, a result of RRR02: $t(\log(U/t) + \log_2 e + o(1))$ bits and rank in $O(1)$ time.
- Also select(i) in constant time $\longrightarrow$ find the element of $S$ of rank $i$.

# Aho-Corasick automaton




- $P$: the set of all prefixes of strings in $S$.
- states in the automaton correspond to a prefixes in $P$. Number of states is $m = |P|$.
- Three kinds of transitions: *next*,*failure* and *report*.

# Our result (Overview)




- Each state is represented as a unique number in $[0, m-1]$.
- The number associated with each state represents the suffix-lexicographic order of the prefix corresponding to that state in the set $P$.
- Example: for the set $S = \{"ABC", "B", "BC", "CA"\}$. We have $P = "", "A", "CA", "B", "AB", "C", "BC", "ABC"$.

# Encoding of *next* transition (1)

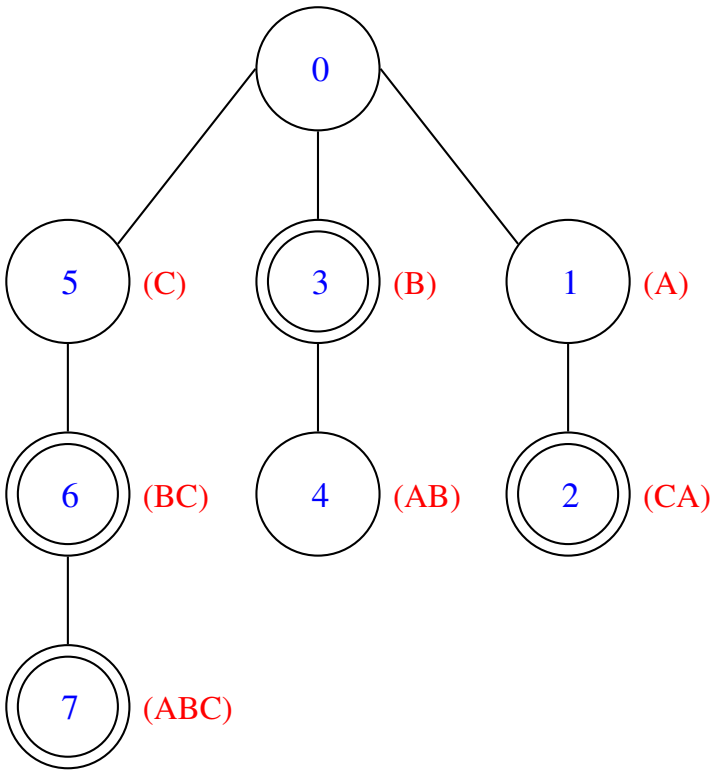
- Transitions are stored in an indexable dictionary.
- A *next* transition labeled with character $c$: $state(p) \longrightarrow_c state(p)$.
- Transition stored as a pair $(c, state(p))$: $(c, state(p)) \longrightarrow \log \sigma + \log m$ bits (bits of $c$ + bits of $state(p)$).
- Each pair corresponds uniquely to one of the prefixes of $P$ (except the empty prefix).

# Encoding of *next* transition (2)

- natural order of stored pairs $\longrightarrow$ order of the prefixes.
- $m$ transitions $\longrightarrow$ $m$ pairs of integers from $U = [0..2^{\log \sigma + m} - 1]$ .
- Dictionary is succinct $\longrightarrow$ space : $m(\log(U/m) + 1.443 + o(1)) = m(\log \sigma + 1.443 + o(1))$ bits.
- Next transition : $state(p)$ to $state(pc) \longrightarrow state(pc) = rank((c, state(p))) + 1$.
- $state(pc) = -1 \longrightarrow$ transition does not exist.

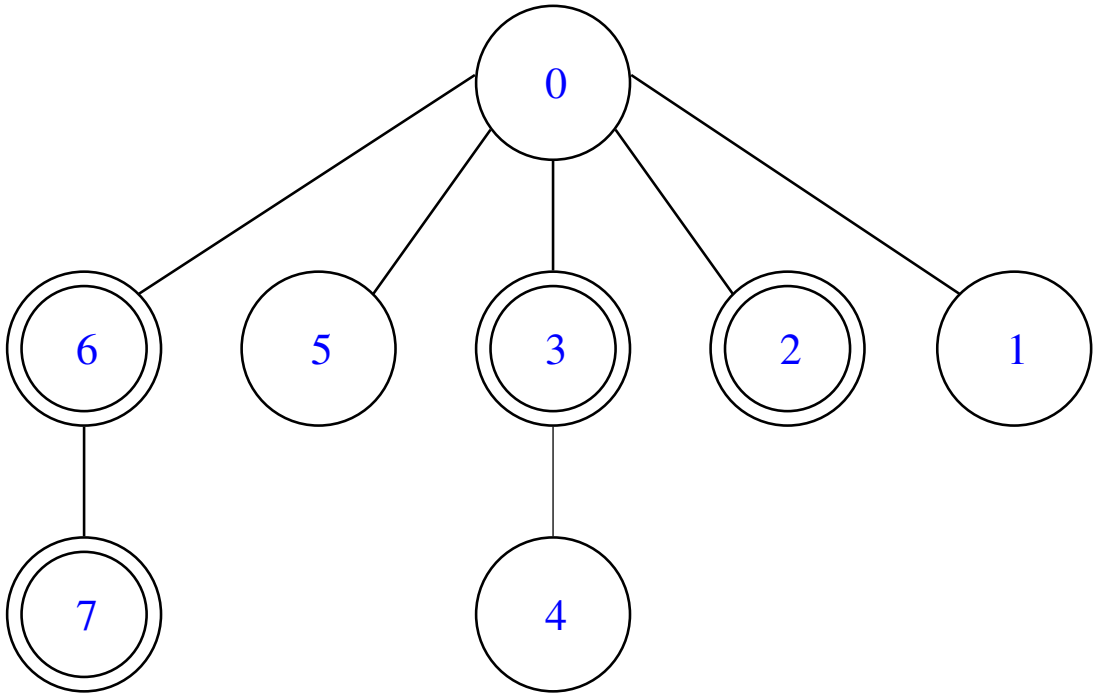
# Encoding of *failure* transitions

- Failure transitions form a tree rooted at state 0 (arrows reversed). The numbers associated with the states correspond to the DFS lexicographic order of the tree.
- Succinct encoding of tree $\longrightarrow$ $m(2 + o(1))$ bits of space.
- Failure transition $\longrightarrow$ parent operation on the tree.



# Encoding of report transitions

- Same as failure tree, but with $\leq d$ internal nodes (only elements of $S$).
- Compressed encoding of the tree in $d(\log(m/d) + O(1))$ bits.
- Report transition $\longrightarrow$ parent operation on report tree.



# Open problems

- Can we remove the $3.443m$ term in the space usage.
- Can query time be improved to $O(|T| \log \sigma / w + occ)$, where $w$ is the size of the processor word. This is obviously the best one could hope for.
- Practical Performance:
  - Query time is asymptotically the same as that of standard Aho-Corasick.
  - But constants in BIG-$O$ notation might be much larger.
  - It would be interesting to explore the practicality of our result.