Bioinformatics

Applications Aho-Corasick(1)

Exact pattern matching with do not cares (‘*’) character

Let T be a string with n characters and P be a string with k-1 do not care ('*') characters of total length m.

Algorithm

0. Let C be an array of integers of length n initialized in zeros.

1. Let P = {p1, p2,...,pk) be the (multi-)set of substrings of P that do not contain wildcard characters. Let l1, 12, . . . , lk be the initial positions in P of each of these substrings (l1=1).

2. Using the Aho-Corasick algorithm find for each string Pi in P, all the initial positions of Pi in T. For each position j of Pi in T, increase the number in cell j - li + 1 of C by one.

3. Scan array C to find cells with value k. There is an appearance of P in T starting from position p if and only if C (p) = k**.**

Applications Aho-Corasick(2)

Two dimensional Pattern Matching

Let T be a two-dimensional text with n=n1 x n2 cells and P a two-dimensional pattern of m=m1 x m2 cells. We want to identify all occurrences of P in T.

The method is divided into two phases.

In the first phase, look for all occurrences of each of the rows of P among the rows of T. To do this, add an end-of-line marker (a character that does not exist in the alphabet) to each line of T, and concatenate these lines into a text string of T' length O(n).

Then, treating each line of P as a separate pattern, use the Aho-Corasick algorithm to search for all occurrences in T' of any row of P.

Therefore, the first phase identifies all occurrences of P and takes time O(n + m).

Applications Aho-Corasick(3)

Whenever an occurrence of line i of P starting with position (p, q) of T is detected, write the number i in position (p, q) of another array M with the same dimensions as T. Because each line of P is considered distinct, and because P is rectangular, at most one number will be written in any cell of M.

in the second phase scan each column of m looking for an occurrence of the string 1.2 . . . . m1 in consecutive single-column cells.

This gives an O(n+m) solution if a linear time pattern matching algorithm is used in each column, regardless of the alphabet (Z-algorithm, Knuth Morris Pratt)

Now suppose that the rows of P are not all discrete. It is enough to identify all identical rows of P and give them a common label.

Techniques for Analysis and Comparison of Biological Data Sequences

- Suffix Tree
- Generalized Suffix Tree
- Applications in Molecular Biology Problems

Definitions

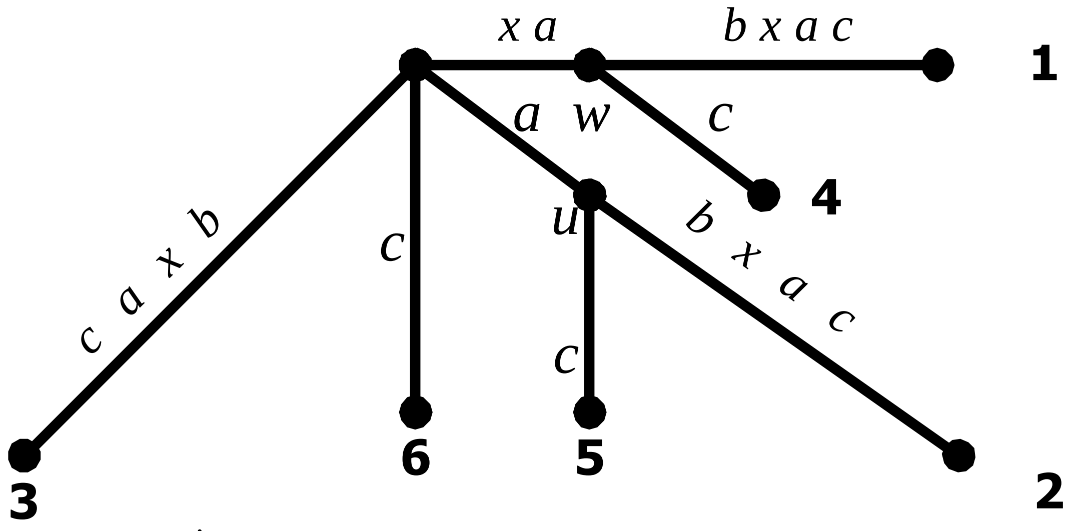
- String: $x=x[1]x[2]\ldots\ldots x[n]$, $x[i]\in\Sigma$ & $|x|=n$
  $x=$ acgttaaaca, $|x|=10$ & $\Sigma=\{a,c,g,t\}$

- Empty string: ε
- Substring w: x=uwv
- Prefix w: x=wu
- Suffix w: x=uw
- Each string S, length |S|=m, has m non-empty suffixes which are the following: S[1…m], S[2…m], …. S[m-1…m] και S[m].
- Example "sequence" : *sequence, equence, quence, uence, ence, nce, ce, e.*

Suffix Tree

Definition: "stores all possible suffixes of a string".

p= xabxac

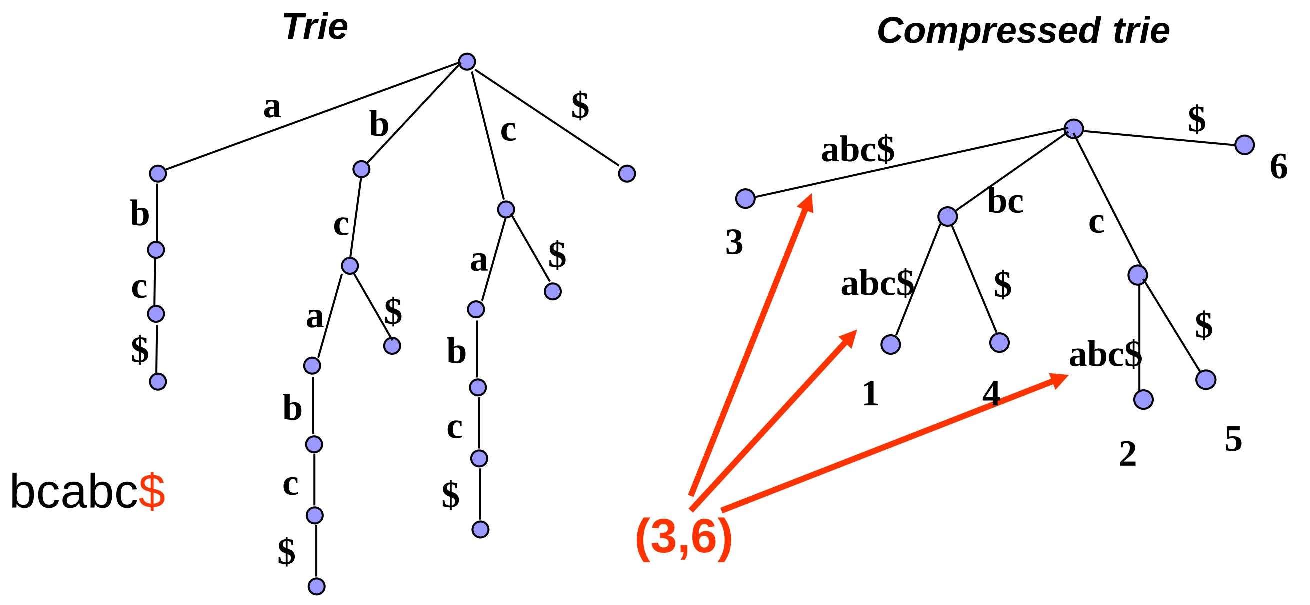


Definition A suffix tree T for a string of n characters is a rooted directed tree with exactly n leaves numbered from 1 to n. Each inner node, except the root, has at least two children, and each edge is marked with a nonempty substring. It is not possible for a node to have two edges starting from the same character. The key feature of the suffix tree is that for any leaf i, the concatenation of edge-labels in the path from root to leaf i is equal to the suffix of the string starting at position i.

Dan Gusfield Algorithms on Strings, Trees and Sequences, Cambridge University Press,

Suffix Tree

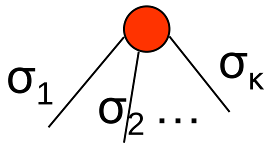
Definition: The suffix tree of a string S[1... n] is a compact trie that contains as keys , all suffixes S[i…n], 1≤i≤n.



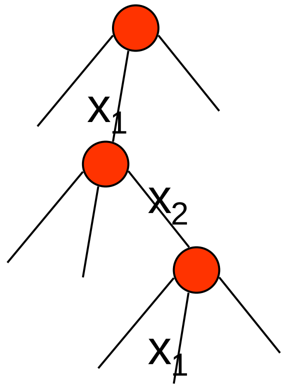
Trie Definition:

Let universe U= $\Sigma^0 \cup \ldots \cup \Sigma^l$ for alphabet Σ and l>0.

$$(x \in U : x = d_1 d_2 \cdots d_l) \; S \subseteq U :$$



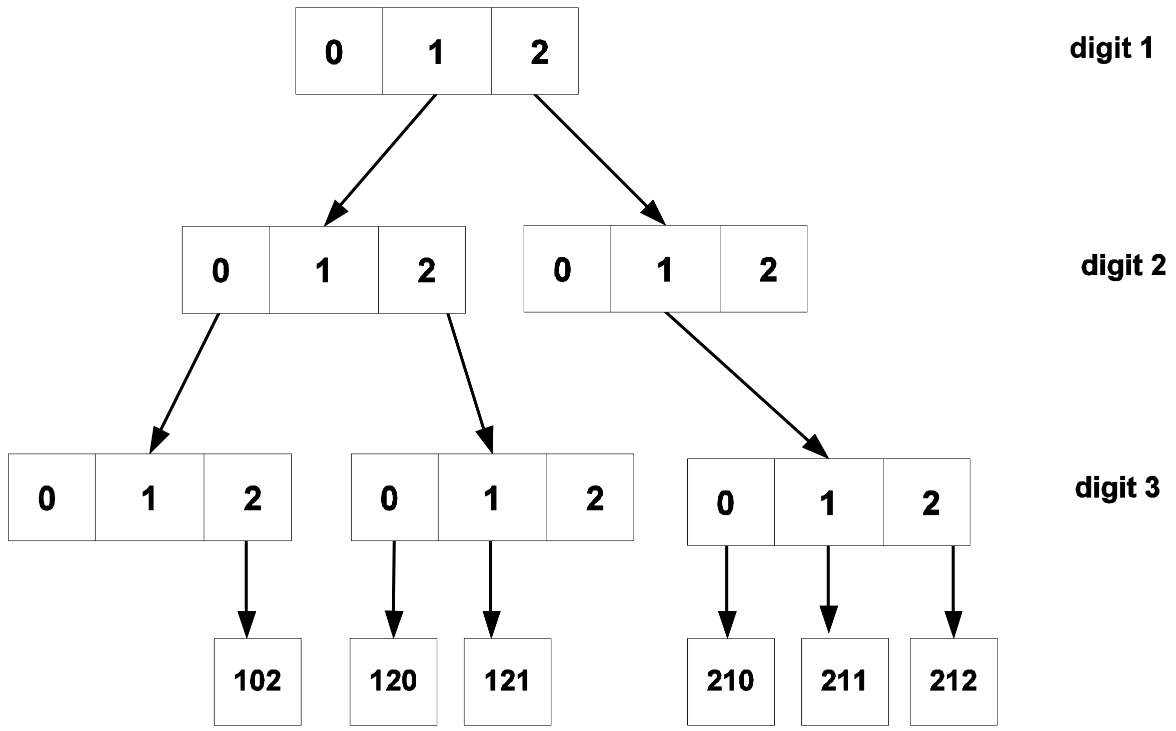
$$(\sigma_i \in \Sigma)$$



Trie (uncompressed)

Trie - example

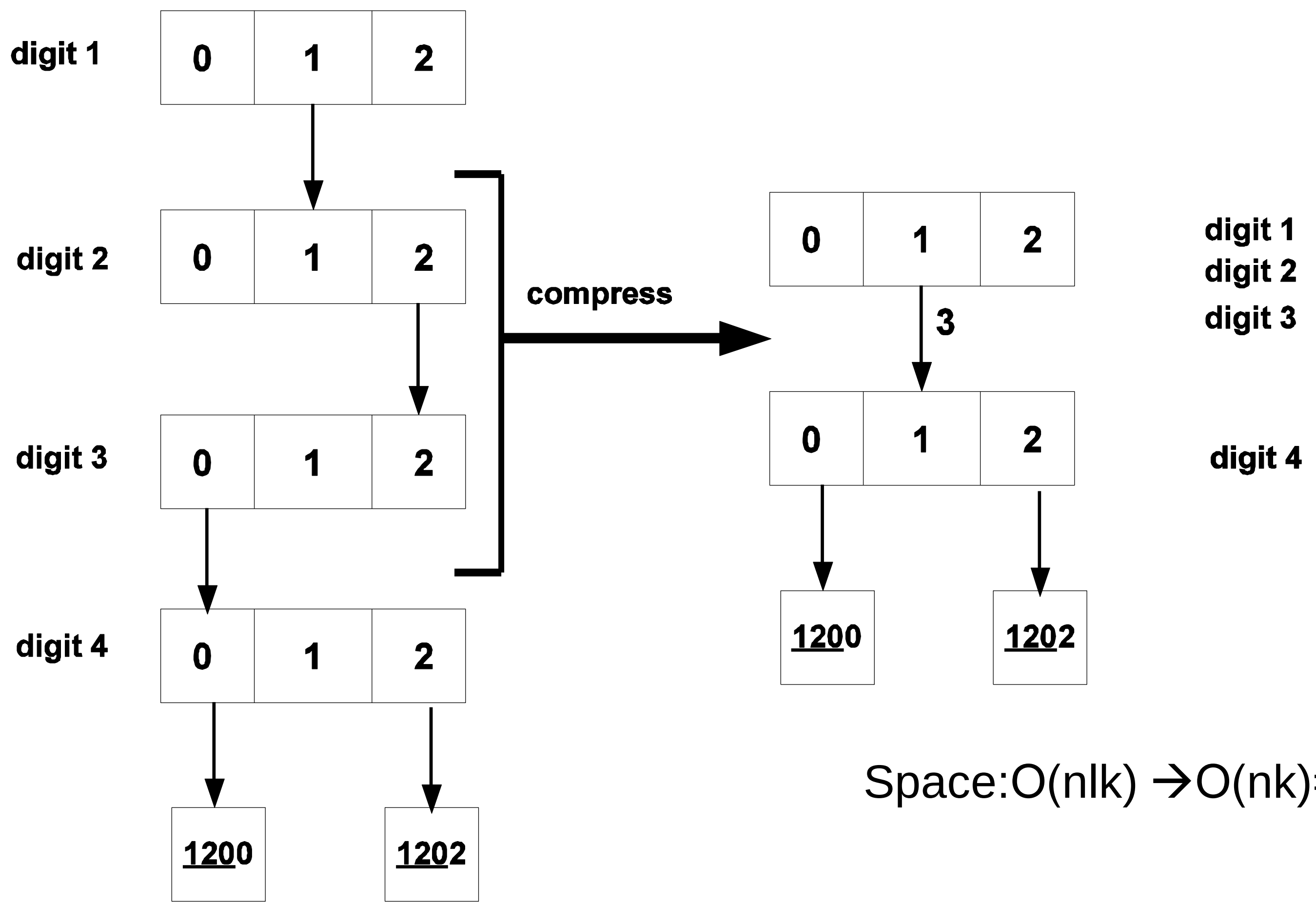
S={ 102, 120, 121, 212, 211, 120},   Σ={0,1,2}



Time ins/del/search : O(l)

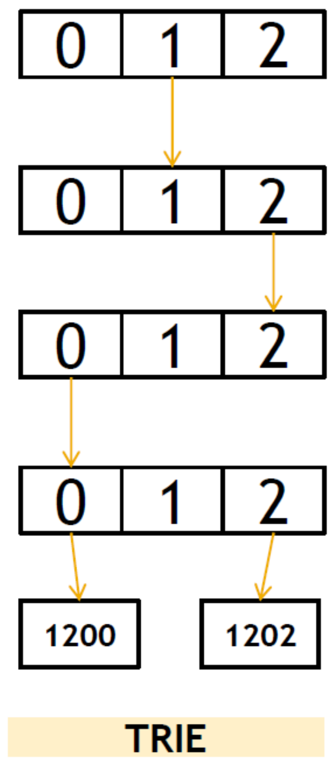
Space: O(nlk)

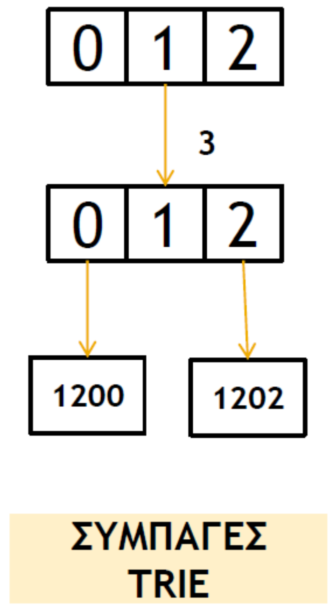
Compressed Trie



Space:O(nlk) →O(nk)=O(n)

Compressed Trie - example





Preprocessing time to build Suffix tree

- Weiner’s algorithm [FOCS, 1973]

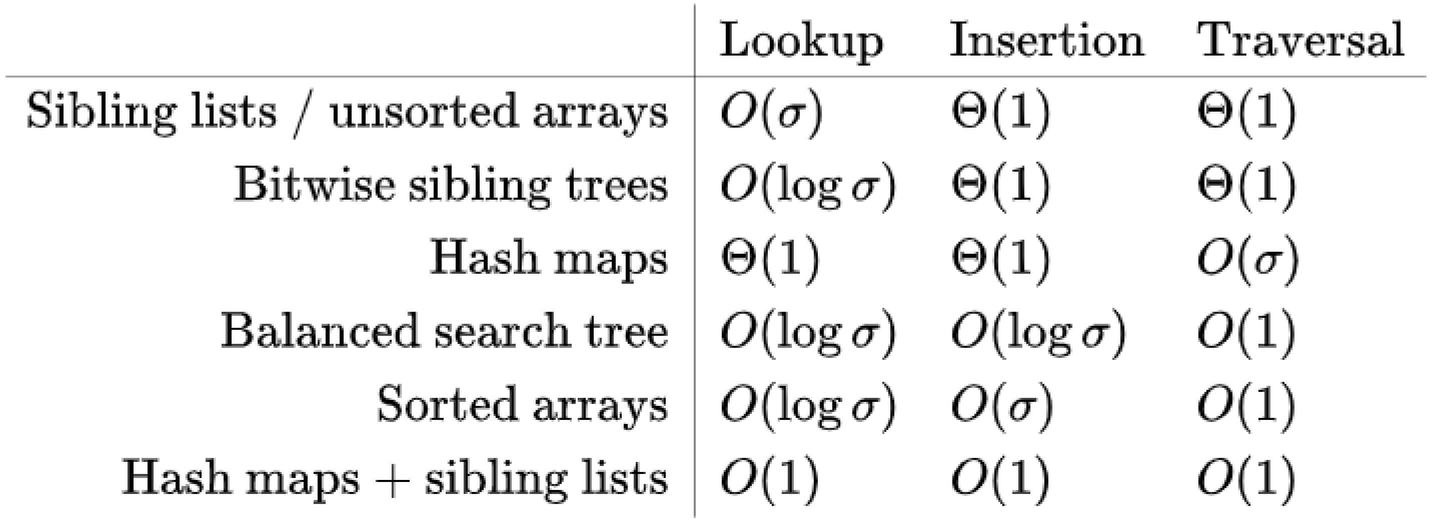
  Knuth ”The algorithm of 1973”
- McCreight’s algorithm [JACM, 1976]

  Linear time and space
- Ukkonnen's algorithm [Algorithmica, 1995]

  Linear time and less space
- Farach’s algorithm [FOCS 1997],

  gave the first linear time alphabet indepedent

Implementation (https://en.wikipedia.org/wiki/Suffix_tree)



	Lookup	Insertion	Traversal
Sibling lists / unsorted arrays	$O(\sigma)$	$\Theta(1)$	$\Theta(1)$
Bitwise sibling trees	$O(\log \sigma)$	$\Theta(1)$	$\Theta(1)$
Hash maps	$\Theta(1)$	$\Theta(1)$	$O(\sigma)$
Balanced search tree	$O(\log \sigma)$	$O(\log \sigma)$	$O(1)$
Sorted arrays	$O(\log \sigma)$	$O(\sigma)$	$O(1)$
Hash maps + sibling lists	$O(1)$	$O(1)$	$O(1)$