Tries

Goldsmiths Computing

Motivation

A data structure

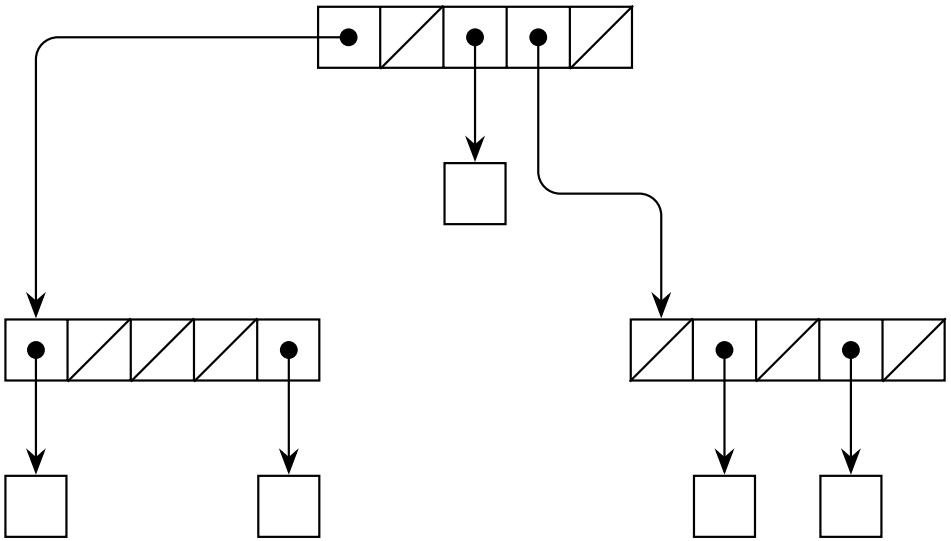
- to hold a set of strings
- to answer efficiently string prefix match
  - (including set membership)

Definition

A trie is a tree structure where each internal node has children labelled by characters from an alphabet. The trie represents the set of strings formed by concatenating labels of traversals from the root to a leaf.

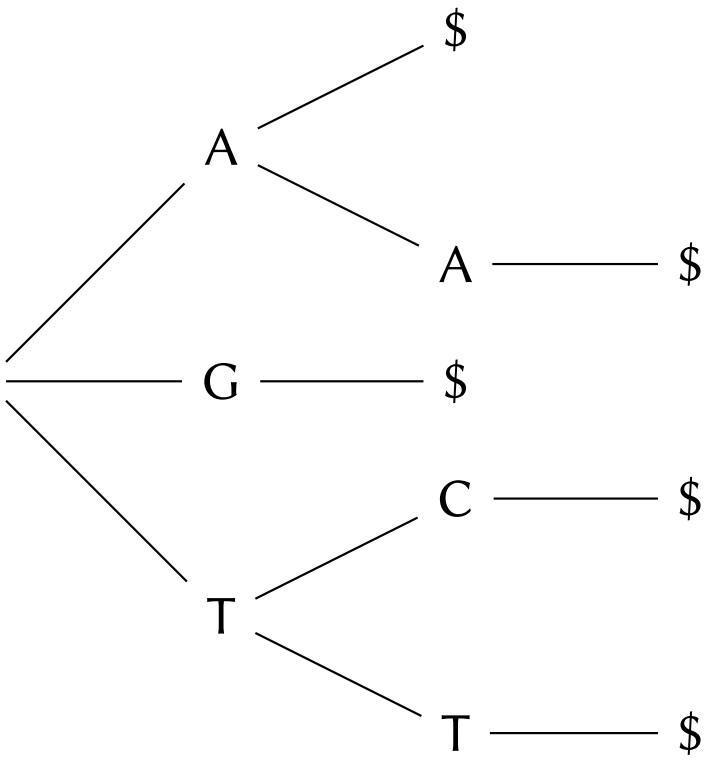
Implementation

- alphabet: `A`, `C`, `G`, `T`
- set of strings: `A`, `AA`, `G`, `TC`, `TT`



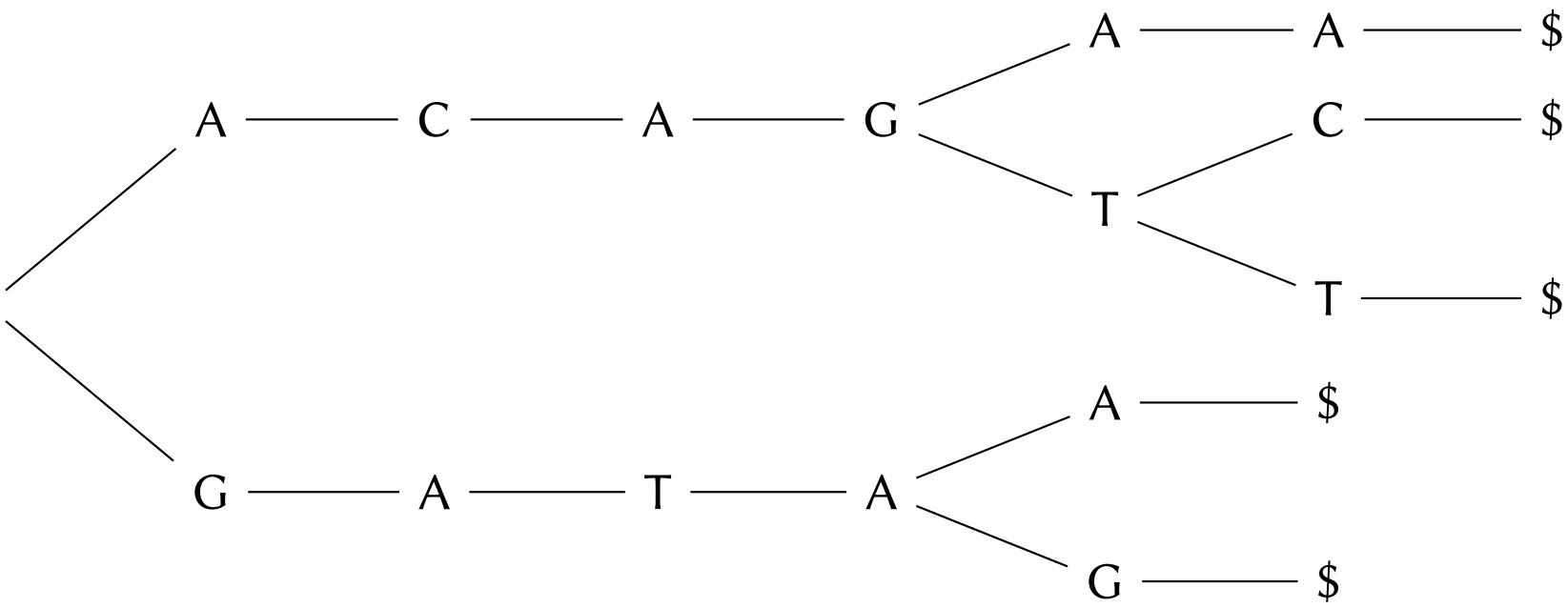
Implementation

- alphabet: `A`, `C`, `G`, `T`
- set of strings: `A`, `AA`, `G`, `TC`, `TT`



Implementation

- alphabet: `A`, `C`, `G`, `T`
- set of strings: `GATAA`, `ACAGAA`, `GATAG`, `ACAGTC`, `ACAGTT`



Algorithm

```
function PREFIX(T,P)
    if EMPTY?(P) then
        return true
    else if LEAF(T) then
        return false
    else if NULL?(T[P[0]]) then
        return false
    else
        return PREFIX(T[P[0]],P[1...])
    end if
end function
```

Algorithm

```
function MEMBER(T,P)
    if EMPTY?(P) then
        if INTERNAL(T) then
            return T[$]
        else
            return true
        end if
    else if LEAF(T) then
        return false
    else if NULL?(T[P[0]]) then
        return false
    else
        return PREFIX(T[P[0]],P[1...])
    end if
end function
```

Implementation

Size of nodes:

small alphabets fixed size internal nodes

large alphabets most branches non-existent: use variable-sized data structure

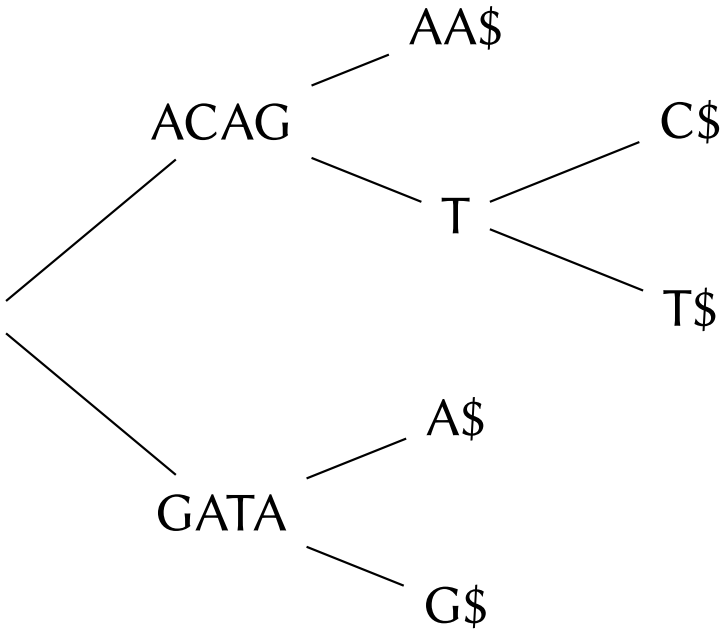
Single-branch internal nodes:

compressed trie collapse the internal nodes and concatenate the labels.

Implementation

Compressed trie:

- alphabet: `A`, `C`, `G`, `T`
- set of strings: `GATAA`, `ACAGAA`, `GATAG`, `ACAGTC`, `ACAGTT`



Suffix trees

- tries allow efficient – $\Theta(m)$ – match
  - at the *beginning* of the text,
  - for *multiple* texts;
- string match performs match
  - at the beginning of all suffixes of a text;
- ... so solve string matching by inserting all *suffixes* of a text into a tree, then using prefix match of the pattern.

But:

- construction of suffix tree in $\Theta(n)$ time is tricky;
- worthwhile if doing multiple string matches (with arbitrary patterns) on the same text.

Work

1. Reading
    - Drozdek, sections 7.2–7.4
    - Mark Nelson, *Fast String Searching with Suffix Trees*, Dr Dobb's Journal (August 1996)
        - and references therein