Strings (1/2)

Prof. Rhadamés Carmona

Last revision: 30/01/2020

Content

- Trie
- Suffix Trie
- Suffix tree
- Suffix array (next meeting)

Trie

- A Trie is a tree where words are stored to find them quickly
- Suppose the words are made up of characters in an alphabet with cardinality n
- Each Trie node can have between 0 and n children

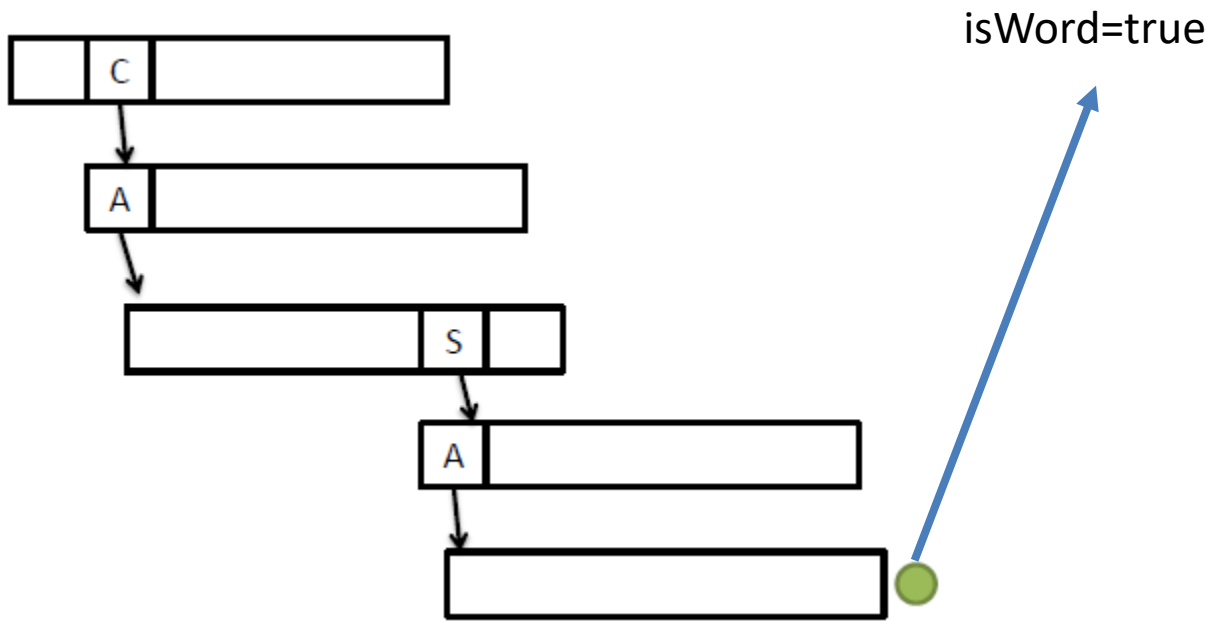
Trie

- For example, if words can be made up of uppercase letters (26 characters), a node looks like this:

```
struct NodoTrie
{
        NodoTrie * child[26];
        bool isWord;
};
```

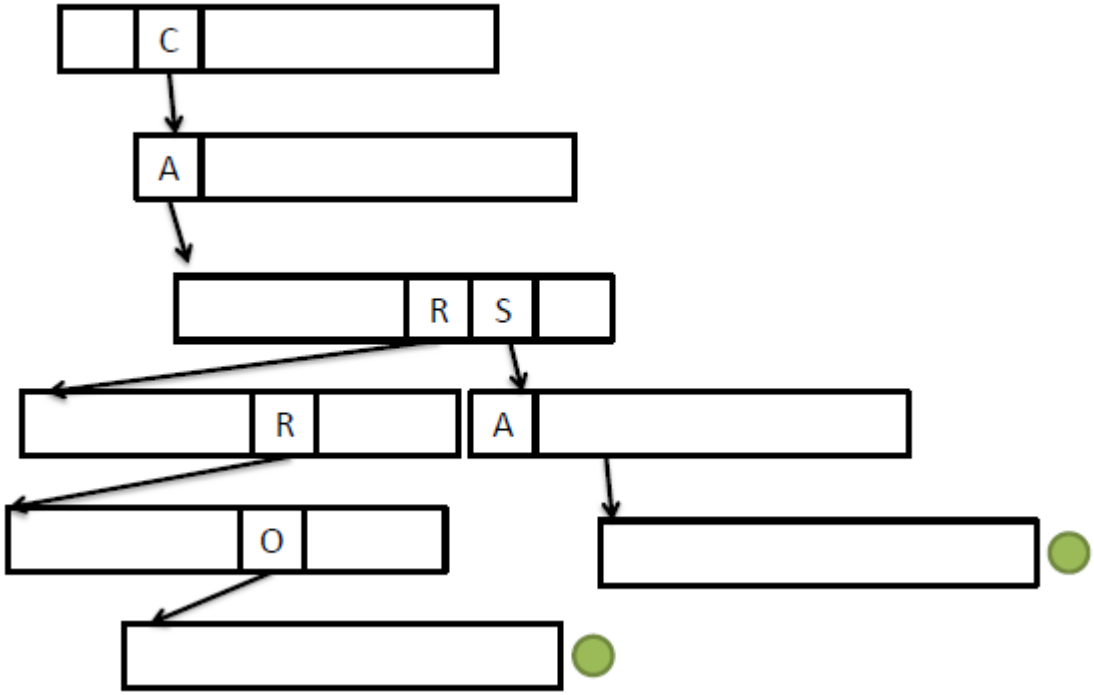
Trie

- Insert “CASA”



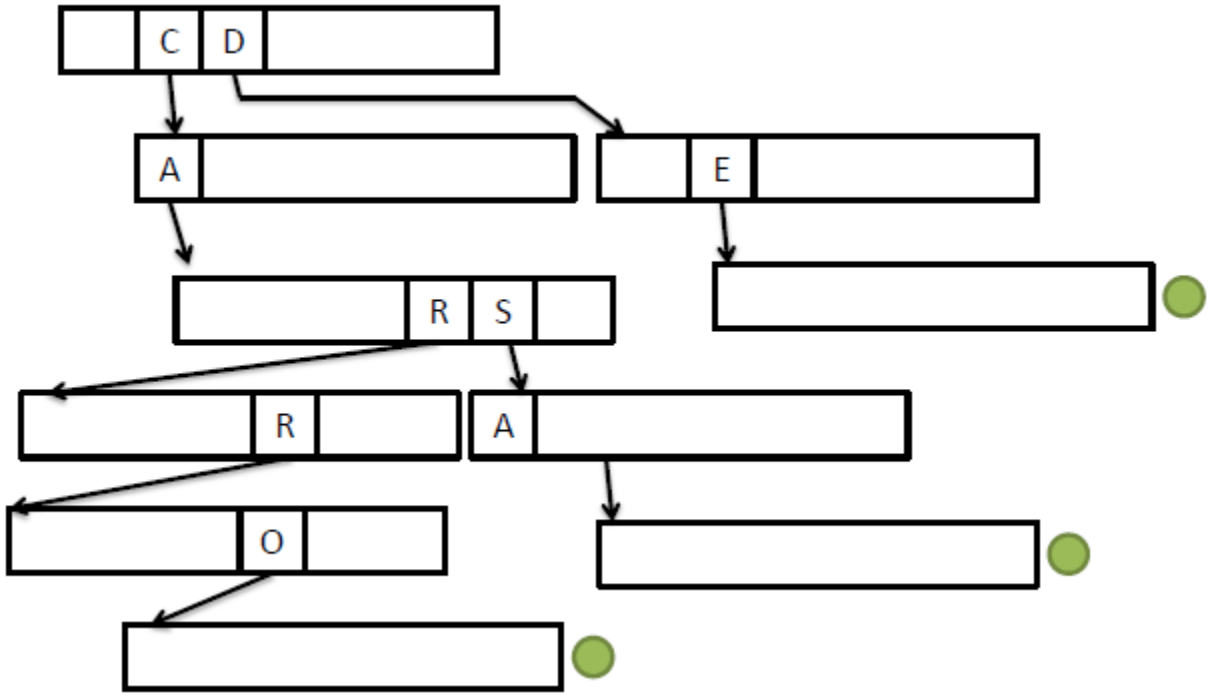
Trie

- Insert “CARRO”



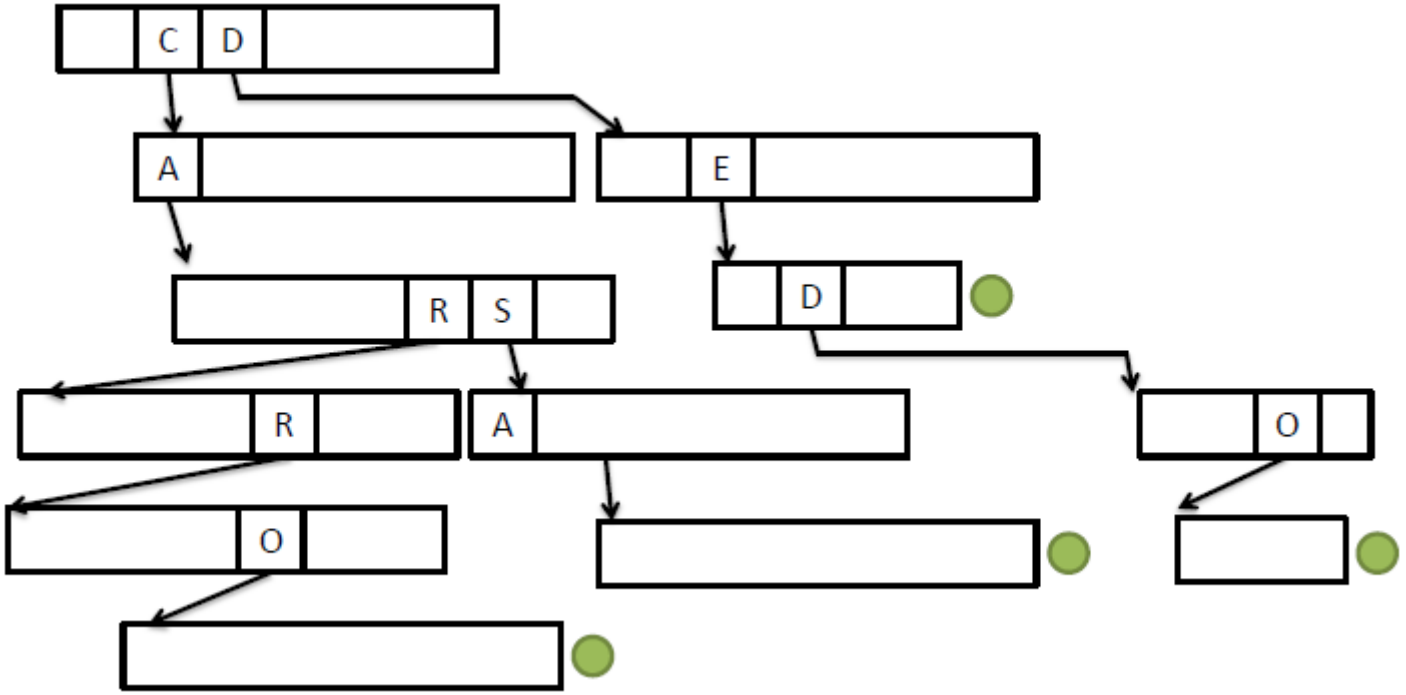
Trie

- Insert “DE”



Trie

- Insert “DEDO”



Trie

- Follow the links depending on the value of the current letter
- If a link is **NULL**, the word is not in the Trie
- If at the end of the word we reach a node with isWord = **false**, the word is not in the Trie
- If at the end of the word we reach a node with isWord = **true**, the word is in the Trie

```
bool find(char *w, Trie * T){
  if(*w==NULL) return T->isWord;
  if(T->child[*w-'A']==NULL) return false;
  return find(w+1, T->child[*w-'A']);
}
```

Trie

- Another way to implement it? You know, we are wasting too much memory in 26 null pointers or even more. It is fast but…

```
struct Trie
{
        unordered_map<char, Trie *> children;
        bool isWord;
};
```

Suffix Trie

- It is used to store all the suffixes of various strings in order to be able to recover them quickly
- With a Trie we can quickly know if a word is stored
- With a Suffix Trie we can quickly know if any substring is stored

Suffix Trie

- Consider the Spanish word “CASA”
- The suffixes are:

CASA

ASA

SA

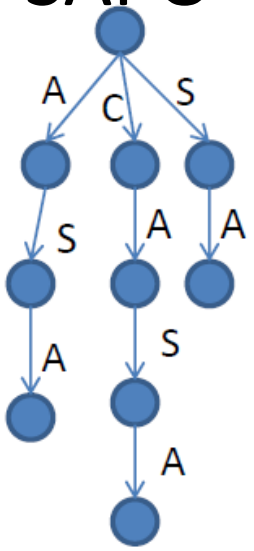
A

Suffix Trie

- If we want to add a word to a Suffix Trie, we add each of its suffixes (it may already exist)
- For example, let’s take the words CASA, CARRO, ASAR, SAPO

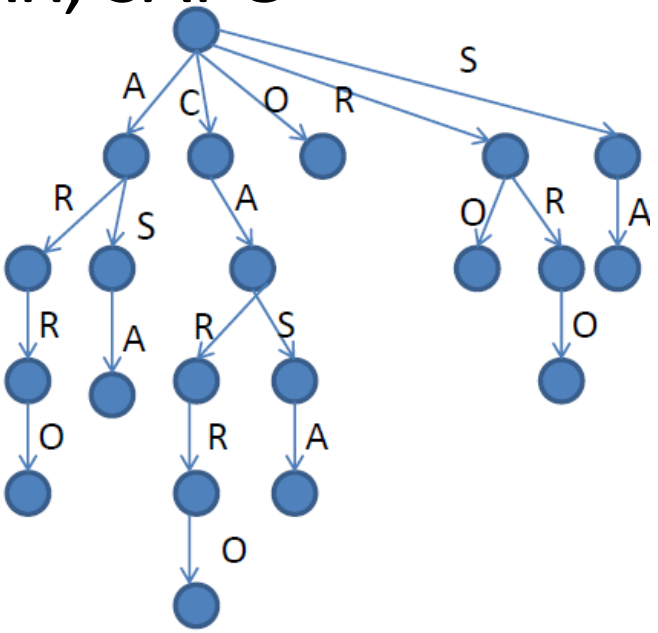
Suffix Trie

- If we want to add a word to a Suffix Trie, we add each of its suffixes (it may already exist)
- For example, let's take the words **CASA**, CARRO, ASAR, SAPO



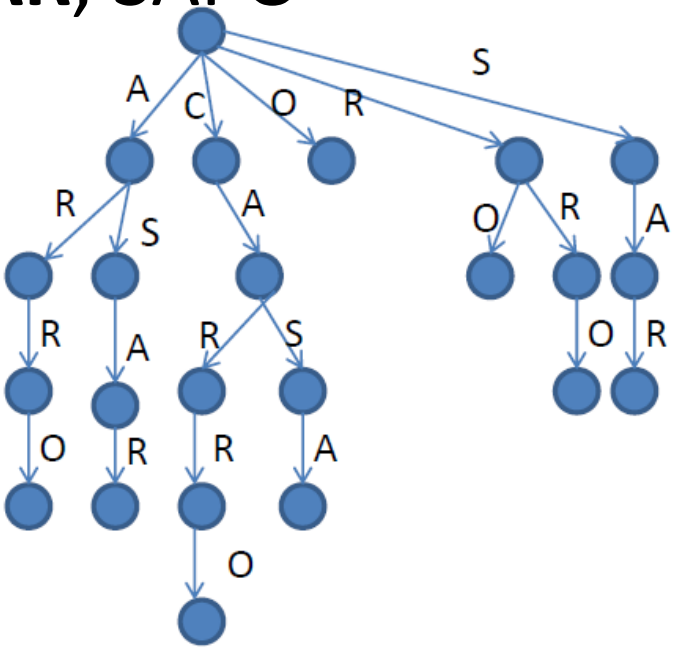
Suffix Trie

- If we want to add a word to a Suffix Trie, we add each of its suffixes (it may already exist)
- For example, let's take the words **CASA**, **CARRO**, ASAR, SAPO



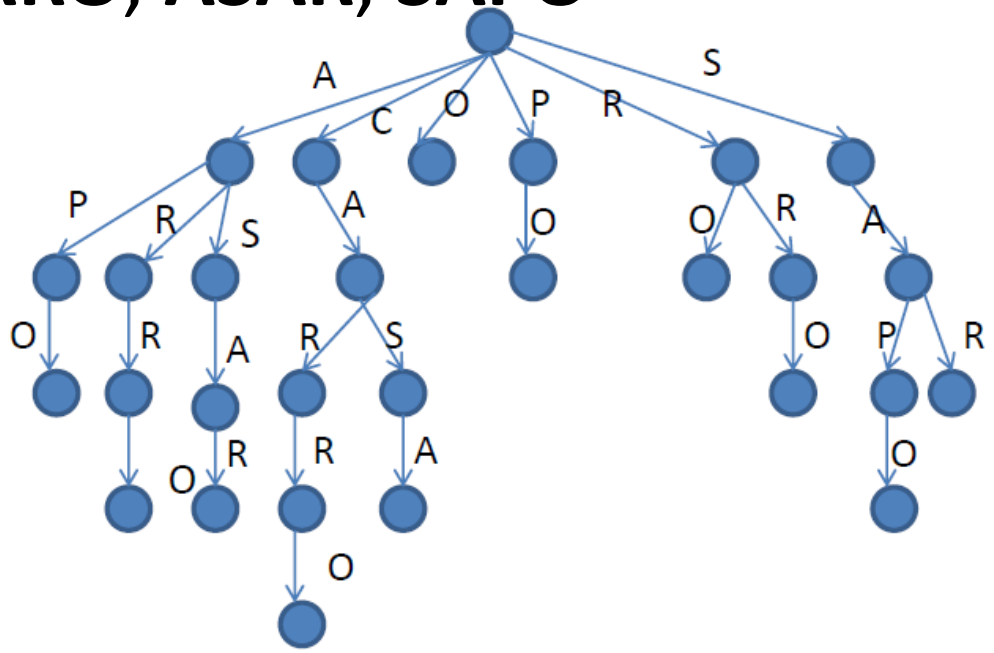
Suffix Trie

- If we want to add a word to a Suffix Trie, we add each of its suffixes (it may already exist)
- For example, let's take the words **CASA**, **CARRO**, **ASAR**, SAPO



Suffix Trie

- If we want to add a word to a Suffix Trie, we add each of its suffixes (it may already exist)
- For example, let's take the words **CASA**, **CARRO**, **ASAR**, **SAPO**

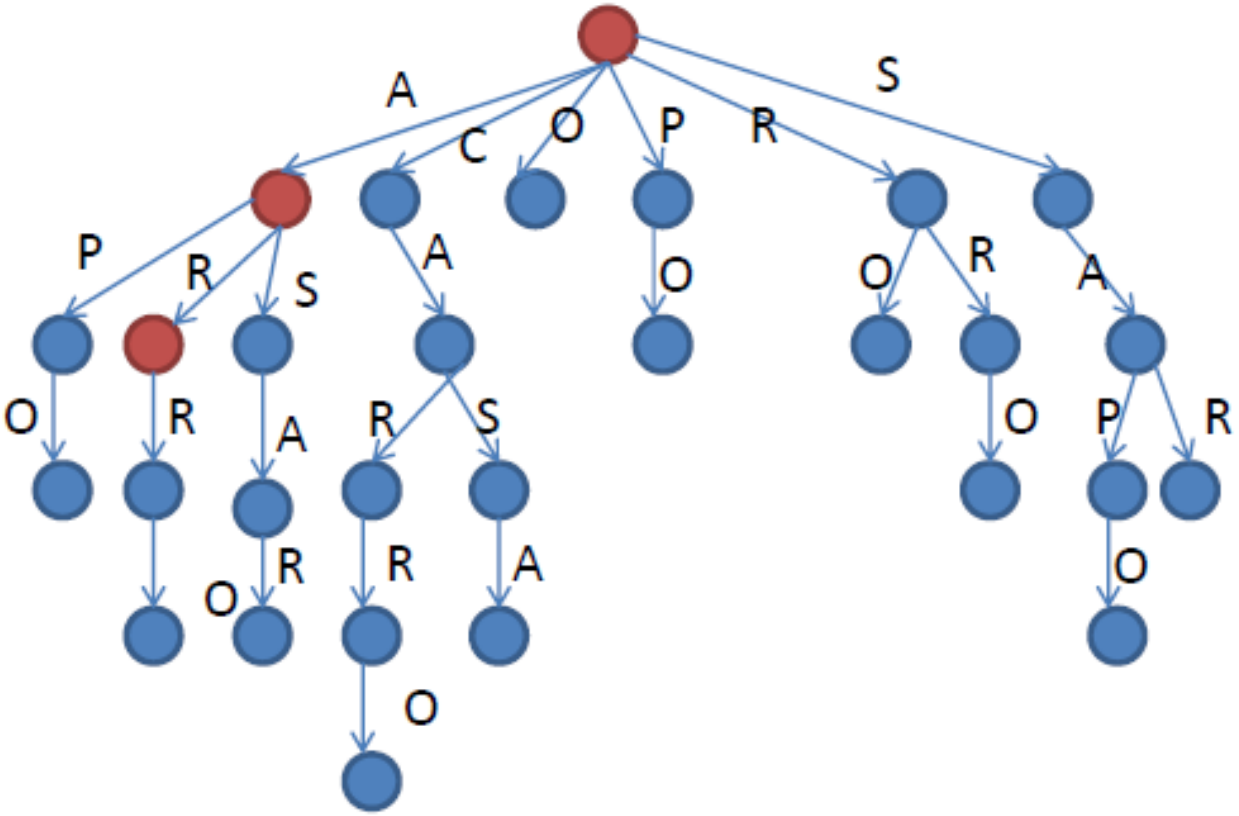


Suffix Trie

- Now we can search a substring of any dictionary word very easily
- Remember that the Trie allows to recognize string prefixes
- Since we store string suffixes, we can recognize any prefix of any suffix
- A substring is just that, a prefix of a suffix

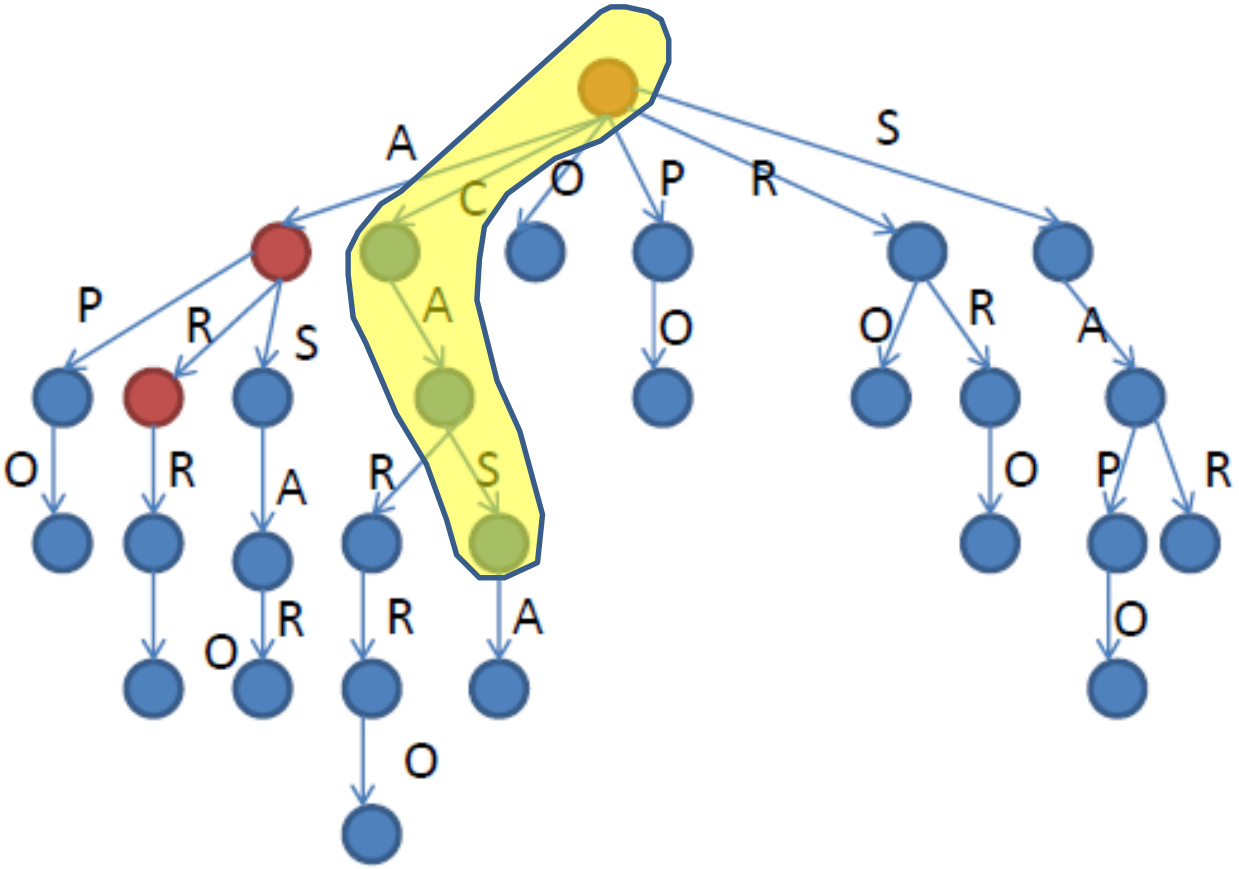
Suffix Trie

- Where is the substring “AR” ?



Suffix Trie

- Where is the substring “CAS” ?

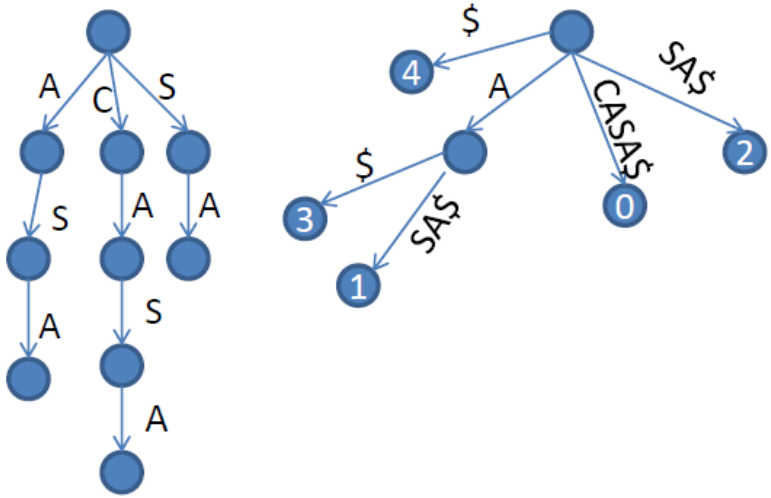


Suffix Trie Applications

- Find a substring s in S: just do a DFS
- Longest substring between 2 strings: build the suffix trie for each string, and compute the intersection. Example, triangle - angle
- Longest repeated substring: Given a string S, find the longest substring s of S that appears in at least two different positions = (find the deepest node with more than one child)

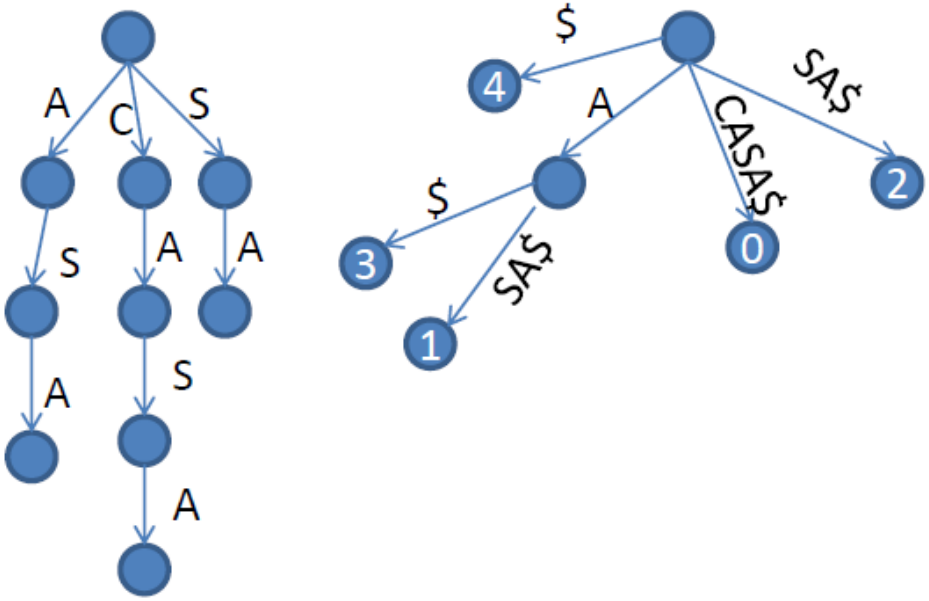
Suffix Tree

- Is used to store all the suffix of **ONE** string efficiently
- A trie can have a long paths without branches
- We can compress the tree by merging paths with only one child per node



Suffix Tree

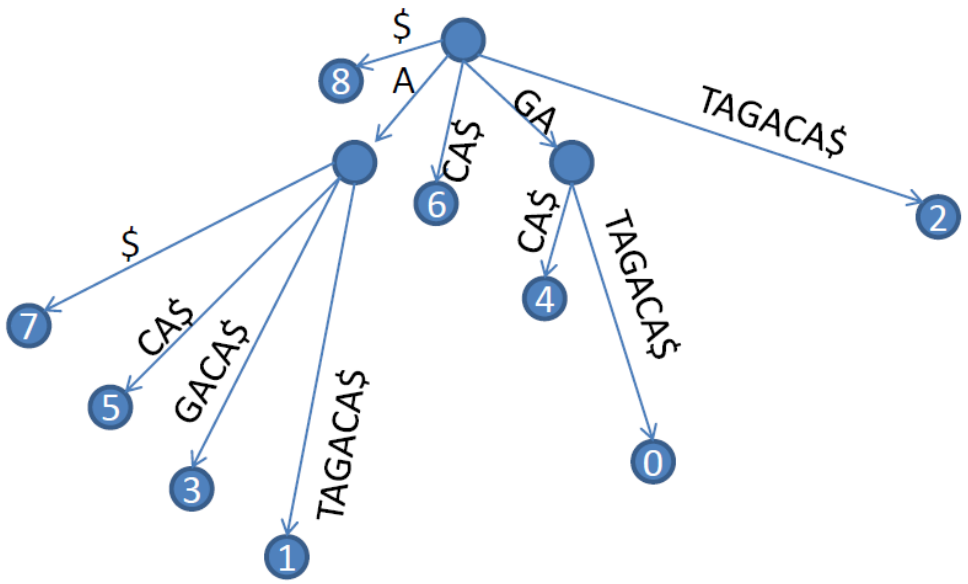
- Use a special character to mark the end of the string ($), whose ASCII code is less than that of the rest of the string characters (we may use NULL instead)



Suffix Tree

- On each leaf store the index of the suffix. The number of leaves is O(n)

i	Sufijo
0	GATAGACA$
1	ATAGACA$
2	TAGACA$
3	AGACA$
4	GACA$
5	ACA$
6	CA$
7	A$
8	$



- https://www.coursera.org/lecture/dna-mutations/from-a-trie-to-a-suffix-tree-iWrbu

Suffix Tree Applications

- Find a substring **s** in **S**: just do a DFS
- Is **s** a suffix of **S**?: similar to find a substring, but this time the end should be $
- String matching: how many times exist a substring **s** in **S** ?
- DFS can find all matches of a substring **s** in time O(n + z), where z is the number of matches

Suffix Tree Applications

- Longest Common Substring
- Longest Common Subsequence
- http://web.stanford.edu/class/archive/cs/cs166/cs166.1146/lectures/10/Small10.pdf
- http://berkri.web.elte.hu//Theses/Vasarhelyi_2.pdf

Exercises

Trie, Suffix Trie:

https://www.spoj.com/problems/ADAINDEX/

https://www.spoj.com/problems/PHONELST/

https://www.spoj.com/problems/MORSE/

https://www.spoj.com/problems/PRHYME/

Exercises

Suffix Tree:

https://www.spoj.com/problems/NEXTLEX/

https://www.spoj.com/problems/STRSTR/

https://www.spoj.com/problems/SUBLEX/

Exercises

Suffix Array:

https://www.spoj.com/problems/SUBST1/

https://www.spoj.com/problems/DISUBSTR/

?

https://www.spoj.com/problems/BEADS/