Applied Algorithmics COMP526 – tutorial 5

L.A. Gąsieniec and D. Cartwright

1 Questions

1.1 Suffix-trees and compact suffix trees

Draw a *suffix tree* for an input string $abbabbaa$ and further form a *compact suffix tree.* Comment briefly on sizes of *suffix trees* and *compact suffix trees* for strings built over constant size alphabets.

1.2 Off-line pattern matching

Write a pseudocode of a recursive procedure that finds a pattern $p = p[0,..,m-1]$ in a text $t = t[0,..,n-1] \in \{a,b\}^*$ available in the form of a compact suffix tree T.

2 Solutions

2.1 Suffix-trees and compact suffix trees

The suffix tree, see Figure 1, for the string *abbabbaa* is a trie that contains all suffixes of *abbabbaa*. A compact suffix tree for the string is obtained by exchange of all chains (including single edges) in the suffix tree by a reference to an appropriate substring of *abbabbaa*.

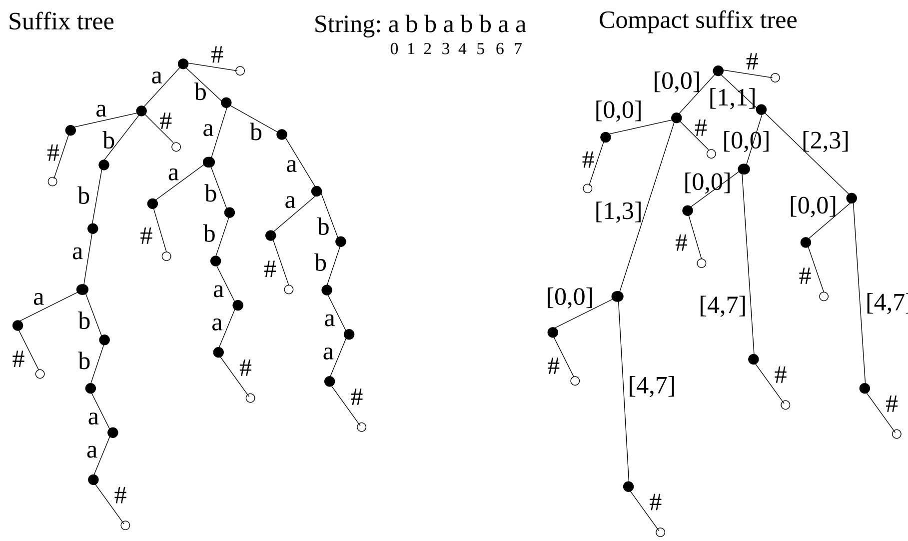

Figure 1: The suffix tree and a compact suffix tree for *abbababa*

A standard suffix tree might be as large as $\Omega(n^2)$, e.g., when all symbols in the input string are different. However, one can construct a compact suffix tree of size $O(n)$, which is a standard suffix tree in which all chains (including single edges) are replaced by a reference to the appropriate substring.

2.2 Off-line pattern matching

Assume that each node v in the suffix T tree keeps the following information:

- $v.lchild$ and $v.rchild$ to denote links to the left child and to the right child respectively. Any of these links is set to *null* if there is no branch leading to the appropriate child.
- $v.hash$ which is set to *true* is v represents some suffix,
- $(v.li, v.lj)$ and $(v.ri, v.rj)$ to denote labels on edges leading to children, where each label is a pair of corresponding indices in the text t.

Assume also that we have a linear function $pref(x,y)$ that checks whether a string x is a prefix of another string y.

function $match(v : node, k : integer) : boolean;$

if $(p[k] = a)$ and $(v.lchild \neq null)$ **then** *(if the next symbol in p is a, go to the left subtree)*
  if $pref(t[v.li, .., v.lj], p[k, .., m-1])$ **then**
    return ($match(v.lchild, k + v.lj - v.li + 1)$);
  elseif $pref(p[k, .., m-1], t[v.li, .., v.lj])$ **then**
    return (*true*);
  else return (*false*);

elseif $(p[k] = b)$ and $(v.rchild \neq null)$ **then** *(if the next symbol in p is b, go to the right subtree)*
  if $pref(t[v.ri, .., v.rj], p[k, .., m-1])$ **then**
    return ($match(v.rchild, k + v.rj - v.ri + 1)$);
  elseif $pref(p[k, .., m-1], t[v.ri, .., v.rj])$ **then**
    return (*true*);
  else return (*false*);

else return (*false*); *(if there is no appropriate child, exit)*

end $match;$

(somewhere in the Main program)

...
$k \leftarrow 0;$
if $match(T, 0)$ **then** report that p occurs in t
  else report that p does not occur in t.
...