# R3 tree

Property XXX of suffix trees would be sufficient for us to be able to report all *right maximal* repeats. To ensure left maximality of $(p_1, p_2, l)$, it has to hold that $LC[p_1] \neq LC[p_2]$. In following section we'll define another refinement of the structure of the suffix tree, so that we can address this requirement. This will be done by partitioning the $Pos(v)$ sets of nodes of $T$ according to left contexts of its suffixes. The result will be a conceptual structure *LC-bucket tree* that will be used to illustrate problems we need to cope with, when we want to implement *findPairs* query.

## 1.1.1 Definition (LC-buckets)

Let $\Sigma_{¢} = \Sigma \cup \{¢\}$. For an internal node $v$ of a suffix tree $T$ for string $S$ of length $n$ and a symbol $a$, we define **LC-bucket** to be a set of positions returned by function $b: V_I(T) \times \Sigma_{¢} \rightarrow 2^{N_n}$ which is defined as follows:

$$b(v,a) = \{u \in Pos(v) | LC(u) = a\}$$

We also define analogues for sets $Pos^+(v)$, $Pos^{2+}(v)$

$$b^+(v,a) = \{u \in Pos^+(v) | LC(u) = a\}$$
$$b^{2+}(v,a) = \{u \in Pos^{2+}(v) | LC(u) = a\}$$
$$b^*(v,a) = \{u \in Pos^*(v) | LC(u) = a\}$$

and set of all LC-buckets for a suffix tree $T$

$$B(T) = \{b(v,a) | v \in V_I(T), a \in \Sigma_{¢}, b(v,a) \neq \emptyset\}$$

For a LC-bucket, we define it's left context as the left context of any of it's elements:

$$\forall b \in B(T): LC(b) = LC(i), i \in b$$

As LC-buckets group suffixes with the same left context, this function is well-defined.
We define sets $B^+(T), B^{2+}(T), B^*(T)$ and $LC$ for their elements analogically.

## 1.1.2 Definition (LC-bucket tree)

LC-bucket tree is created from suffix tree by removing leaves and appending LC-buckets to respective internal nodes.

Formally **LC-bucket tree** for string $S$ is a tree $T = (V, E, root)$, such that

1. $V = V_I(ST(S)) \cup B(ST(S))$
2. $E = E_I(ST(S)) + E_B$
3. $root = root(ST(S))$

Where $E_B = \{(v, b(v,a)) | v \in V_I(ST(S)), a \in \Sigma_{¢}, b(v,a) \neq \emptyset\}$

We will use term $LCBT(S)$ to denote LC-bucket tree for string $S$.

All nodes from $V_I(ST(S))$ have at least one bucket as it's child and bucket nodes don't have any children. Therefore $V_I(T) = V_I(ST(S))$, $V_L(T) = B(T)$, $E_I(T) = E_I(ST(S))$ and $E_L(T) = E_B$

Now we can define *lcplen* function also for LC-bucket tree $T$

$$\forall v \in V_I(ST(S)): lcplen(v) = lcplen_{ST(S)}(v)$$

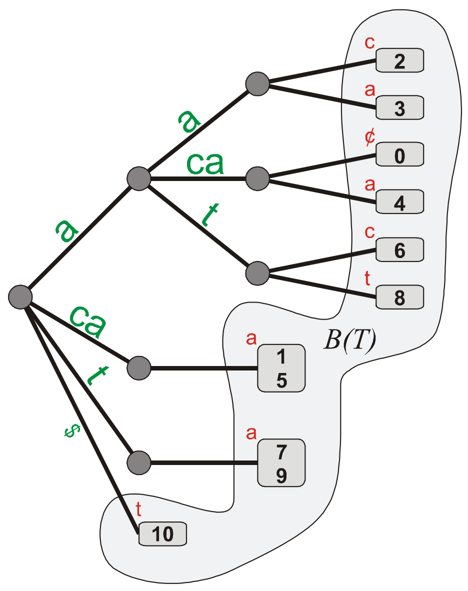

*Figure 3: LC-bucket tree for string 'acaaacatat'*

### 1.1.3 Non-optimal findPairs query on LC-bucket tree

```
$combineSubtree(p, v, visited, l)$
1       for all children $c$ of $v$
2               if $c \neq visited$
3                       if $c$ is bucket and $LC(c) \neq LC(p)$
4                               for all $p_2$ from $c$
5                                       report pair $(p_2, l)$
6                       if $c$ is iternal node
7                               $combineSubtree(p, c, l)$
```

```
$findPairsNonOptimal(p, k)$
1       v := $map(p)$
2       visited := NULL
3       while $v \neq$ NULL and $lcplen(v) \geq k$ do
4               $combineSubtree(p, v, visited, lcplen(v))$
5               $visited := v$
6               $v := parent(v)$
```

**Theorem.**

Procedure $findPairsNonOptimal(p_1, k)$ reports pair $(p_2, l)$ if and only if $(p_1, p_2, l)$ is maximal repeat in $S$ such that $l \geq k$

Proof:

Let $T = LCBT(S)$, $v_1 = map(p_1)$, $v_2 = map(p_2)$ and $w = LCA_T(v_1, v_2)$. Let $u_1 = v_1, u_2 = parent(u_1), \ldots, u_p = parent(u_{p-1})$ be the sequence of nodes on path from $v_1$ to $root(T)$ visited by procedure *findPairsNonOptimal* in the while loop on lines 3-6. $u_p$ is the last node with $lcplen(u_p) \geq k$. Note that once a subtree of a node $u_j$ is marked as visited in line 5 it is never visited again in any subsequent calls of *combineSubtree* on line 4. This is because next node to be marked is it's parent $u_{j+1}$, whose marking also excludes subtree of $u_j$.

(if) :
Let $(p_1, p_2, l)$ be a maximal repeat in $S$ such that $l \geq k$. Since $(p_1, p_2, l)$ is right maximal, by Lemma 1 $lcplen_T(w) = l \geq k$and therefore $\exists j \in \{1..p\} : w = u_j$. It means that procedure $combineSubtree(p, v, visited, l)$ is called at least once with parameters $p := p_1$, $v := w, visited := u_{j-1}, l := lcplen(w)$ ($u_0 =$NULL). Procedure *combineSubree* recursively visits all nodes in subtree of $w$ that weren't already visited by previous call from *findPairsNonOptimal* and for all

buckets $B$ with $LC(B) \neq LC(p_1)$ reports pair $(p, l)$ for each $p \in B$. Since $w$ is ancestor of $v_2$, this node is also visited by *combineSubree* and since $w$ is lowest common ancestor of $v_1$ and $v_2$, $v_2$ hasn't been visited in any previous call. Since $(p_1, p_2, l)$ is left maximal $LC(p_1) \neq LC(p_2)$ and therefore $(p_2, l)$ is also reported.

(only if) :
Let $(p_2, l)$ be a pair reported by a call *findPairsNonOptimal* $(p_1, k)$. It had to be reported by a call of procedure *combineSubree* on $u_j$ for some $j$ in line 4 of *findPairsNonOptimal*. This means that $l \geq k$. The pair $(p_2, l)$ could only be reported if $LC(p_2) \neq LC(p_1)$, because of condition in line 3 of *combineSubree*. $u_j$ is common ancestor of $v_1$ and $v_2$. We will show that it is also lowest common ancestor $w$. Suppose that $w = u_i$ for some $i$ such that $depth(u_i) > depth(u_j)$. This means that $i < j$ and that $(p_2, l)$ is reported by a call *combineSubree* on the node $u_i$ which is then marked as *visited* on line 5 before *combineSubtree* on $u_j$ is called. This means however that combineSubtree on $u_j$ couldn't report pair $(p_2, l)$, which is contradiction. Since $u_j = w$, it holds $l = lcplen_T(w)$ and since $w = LCA_T(v_1, v_2)$, by lemma 1 we have $(p_1, p_2, l)$ is right maximal repeat. Therefore $(p_1, p_2, l)$ is maximal repeat in $S$ with $l \geq k$.

The problem is, that *findPairsNonOptimal* may take O(n) time, while only 1 pair is reported. Let's take string $a^n$ for example. $LCBT(a^n)$ has $n$ internal nodes that form a single path, each having one LC-bucket. All of suffixes have left context $a$ except the suffix 0 which has left context $\cent$. Let $u_i$ be node with $depth(u_i) = i$. $lcplen(u_i) = i$. LC-bucket of each $u_i$ contains exactly one suffix $n$-$i$. The deepest node $u_{n-1}$ contains also LC-bucket with suffix 0. If we call *findPairsNonOptimal* $(n-i, 1)$ for $n-i \neq 0$, we start while loop at lines 3-6 with node $u_i = map(n-i)$ and end at node $u_1$. First call of *combineSubtree* will traverse $n-i+1$ buckets under $u_i$. Subsequent calls of *combineSubtree* on nodes $u_{i-1}, u_{i-2}, \ldots, u_1$ will traverse only one bucket on each. This means another $i-1$ buckets. Only one of these $n$ buckets has left context other than $a$ and therefore only one pair is reported.

The non-optimality of algorithm *findPairsNonOptimal* comes from following two problems:

1. *combineSubtree* $(p, v, visited, l)$ visits all buckets under node $v$, not only buckets with left context other than $LC(p)$. This makes time consumed by *combineSubtree* not proportional to number of reported pairs.
2. while loop on lines 3-6 of *findPairsNonOptimal* visits all nodes $v$ with $lcplen(v) \geq k$ on the path from $map(p)$ to root, disregarding that the node may not contain any bucket with left context other than $LC(p)$ that wasn't previously visited. This means, that we might visit too many nodes without proportional number of pairs being reported.

The R3 tree structure presented in the following text solves exactly these two problems.

### 1.1.4 Definition (Union trees)

If $T = ST(S)$ and $|S| = n$. $B(T)$ is partition of $N_n$, because $(v_1, a_1) \neq (v_2, a_2) \Rightarrow b(v_1, a_1) \neq b(v_2, a_2)$. We can therefore easily store $B(T)$ in $O(n)$ space.

In later section we will need to access $b^+(v, a)$ sets for nodes of suffix tree. $B^+(T)$ may contain overlapping sets and therefore we need to apply a small trick to achieve $O(n)$ space requirements. $B^+(T)$ is superset of $B(T)$. Elements of the set $B^u(T) = B^+(T) \setminus B(T)$ will be called union nodes, because they can be constructed by unioning of buckets from $B(T)$:
$\forall b \in B^u(T) : \exists b_1, b_2, \ldots, b_k \in B(T) : b = b_1 \cup b_2 \cup \ldots \cup b_k$. Our next conceptual structure – union tree - captures the structure of union operators applied to LC-buckets to compose into union node.

**Union tree** for a node $v$ of suffix tree $T = ST(S)$ and symbol $a$, is a tree $UT(v, a) = (V, E, root)$ such that $V = \{ b^+(u, a) | (v, u) \in E_I(T)^* \}$, where $E_I(T)^*$ is transitive and reflexive closure of relation $E_I(T)$.
$E = \{ (M_1, M_2) \in V \times V | M_1 \supset M_2 \wedge \neg (\exists M_3 \in V : M_1 \supset M_3 \supset M_2) \}$
$root = b^+(v, a)$

Sets from $B(T)$ - leaves of union tree, will be represented as sets – we will explicitly store their contents. Sets from $B^u(T)$ will be represented by pointers/edges to subsets from which they are composed. As every union node has at least two descendants $|B^u(T)| < |B(T)|$ and therefore this representation needs

$O(|B(T)|)$ space. Moreover, enumeration of all elements of $b^+(u,a) \in V$ can be done in $O(|b^+(u,a)|)$.

This representation of sets in union nodes also allows union operation in $O(1)$ time which will prove useful later on. For a suffix tree $T = ST(S)$, $UT(T)$ will denote forest of union trees $UT(root(T), a)$. $UT(T)$ can be stored in $O(n)$ space.( $n = |S|$ )

**Property of union trees 1.**

For each internal node $v$ of $ST(S)$ and $a \in \Sigma_¢$, number of union nodes in $UT(v,a)$ is smaller than number of buckets.

Proof : TODO

**Property of union trees 2.**

Let $u$, $v$ be internal nodes of $ST(S)$ and $a \in \Sigma_¢$, such that $v$ is descendant of $u$. Then $UT(v,a)$ is subtree of $UT(u,a)$.

Proof: TODO

### 1.1.5 Definition (R3 tree)

R3 Tree for a string S is a 6-tuple $T = (V, E, root, UT, bp, up)$ such that
$V = V_I(ST(S))$
$E = E_I(ST(S))$
$root = root(ST(S))$
$UT = UT(ST(S))$

$bp: V \times \Sigma_¢ \to UT$ is function returning $b^+(v,a)$ represented by a node in union tree for each node and left context.

$up: V \times \Sigma_¢ \to V$
$up(v,a) = u$ s.t. $(u,v) \in E^+, \exists b \in \Sigma_¢, b \neq a : b^+(u,b) \supset b^+(v,b)$
$\wedge \forall w, (w,v) \in E^+, \exists b \in \Sigma_¢, b \neq a : b^+(u,b) \supset b^+(v,b) : depth(w) < depth(u)$
if such $u$ doesn't exist $up(v,a)$ is undefined

$up$ is navigation function. It returns nearest ancestor $u$ of $v$, such that $Pos^+(u)$ contains at least one more suffix $i$, with $LC(i) \neq a$, than $Pos^+(v)$.

$bpsize(v)$ is number of different symbols $a$, such that $bp(v,a) \neq \emptyset$
$upsize(v)$ is number of different symbols $a$, such that $up(v,a)$is defined

TODO linearity of bpsize

### 1.1.6 Optimal findPairs query on R3 tree

```
combineUnionSubtree(p, u, visited, l)
1    if $u$ = $visited$ exit
2    for all children $c$ of $u$
3        if $c$ is bucket node
4            for all $p_2$ from $c$
5                report pair $(p_2, l)$
6        if $c$ is union node
7            $combineUnionSubtree(p, c, visited, l)$
```

```
findPairs(p, k)
1    $v := map(i)$
2    for each symbol $a$
3        $visited[a]$ := NULL
```

```
4    while $v \neq$ NULL and $lcplen(v) \geq k$ do
5        for all $a$ such that $bp(v, a)$ is defined
6            if $a \neq LC(p)$
7                $combineUnionSubtree(p, bp(v, a), visited[a], lcplen(v))$
8                $visited[a] := bp(v, a)$
9        $v := up(v, LC(p))$
```

**Lemma.**

$combineUnionSubtree(p, bp(v, a), visited, l)$ runs in time $O(z)$where $z$ is number of reported pairs.

Proof:
Let $n_b$ be number of buckets and $n_u$ number of union trees in$UT(v, a)$. From property of union trees 1, we know that $n_u < n_b$. If we prune node *visited* and it's subtree we still have $n_u \leq n_b$. Procedure *combineUnionSubtree* traverses $UT(v, a)$ (except the *visited* node and it's subtree) and reports at least one pair for each bucket it encounters. Therefore $n_b \leq z$ and *combineUnionSubtree* runs in time $O(z)$.

**Theorem.**

Procedure $findPairs(p_1, k)$ reports pair $(p_2, l)$ if and only if $(p_1, p_2, l)$ is maximal repeat in $S$ such that $l \geq k$

Proof:

Let $T = R3T(S)$, $v_1 = map_T(p_1)$, $v_2 = map_T(p_2)$ and $w = LCA_T(v_1, v_2)$. Let $w_1 = v_1, w_2 = parent(w_1), \ldots, w_s = parent(w_{s-1})$ be the sequence of nodes on path from $v_1$ to $root(T)$ such that $lcplen(w_i) \geq k$ for $i \in \{1..s\}$ and $lcplen(parent(w_s)) < k$ or $w_s = root(T)$. This sequence may be empty if $lcplen(v_1) < k$. Let $u_1 = v_1, u_2 = up_T(u_1, LC(p_1)), \ldots, u_p = up_T(u_{p-1}, LC(p_1))$ be the subsequence of $w_1, w_2, \ldots, w_s$ visited by procedure *findPairs* in the while loop on lines 4-9. Note that once subtree $UT(u_j, a)$ is marked as visited in line 8 it is never processed again in any subsequent calls of *combineUnionSubtree* on line 7. This is because next node $u_{j+1}$ is ancestor of $u_j$ and by property 2 $UT(u_j, a)$ is subtree of $UT(u_{j+1}, a)$, whose marking also excludes $UT(u_j, a)$.

(if) :
Let $(p_1, p_2, l)$ be a maximal repeat in $S$ such that $l \geq k$. Since $(p_1, p_2, l)$ is right maximal, by Lemma XXXTODO $lcplen_T(w) = l \geq k$ and therefore $s \geq 1$ and $\exists j \in \{1..s\} : w = w_j$. We need to show that also $\exists j \in \{1..p\} : w = u_j$. If $w = v_1$ this holds for $j$=1. Let's consider the case $w \neq v_1$. Let $i$ be the greatest index such that $w$ is ancestor of $u_i$. It has to hold that either $w = u_{i+1}$ or $u_{i+1}$ is ancestor of $w$. $u_{i+1} = up_T(u_i, LC(p_1))$ therefore

$$(1)\ (u_{i+1}, u_i) \in E(T)^+, \exists b \in \Sigma_{¢}, b \neq LC(p_1) : b^+(u_{i+1}, b) \supset b^+(u_i, b)$$

and

$$(2)\ (\forall z)\big((z, u_i) \in E(T)^+, \exists b \in \Sigma_{¢}, b \neq LC(p_1) : b^+(u, b) \supset b^+(v, b)\big) : depth(z) < depth(u_{i+1})$$

We also have $(w, u_i) \in E(T)^+$ and $b^+(w, LC(p_2)) \supset b^+(u_i, LC(p_2))$, because $w = LCA_T(v_1, v_2)$ and therefore $w$ is lowest ancestor of $v_1$ in which $p_2$ occurs. $LC(p_2) \neq LC(p_1)$ because $(p_1, p_2, l)$ is left maximal repeat.

Let $u_{i+1}$ be ancestor of $w$. Then $depth(u_{i+1}) < depth(w)$. This contradicts (2) and therefore $w = u_{i+1}$.

When node $w$ is visited by while loop $combineUnionSubtree(w, bp(w, LC(p_2)), visited[LC(p_2)], l)$ is called. $bp(w, LC(p_2)) \neq visited[LC(p_2)]$ because $w = LCA_T(v_1, v_2)$. $p_2$ occurs in a bucket of $UT(w, LC(p_2))$ and therefore pair $(p_2, l)$ is reported.

(only if) :
Let $(p_2, l)$ be a pair reported by a call $findPairs(p_1, k)$. It had to be reported by a call of procedure

*combineUnionSubree* on $u_j$ for some $j$ in line 7 of *findPairs*. From condition of while loop on line 4 we have $l \geq k$. The pair $(p_2, l)$ could only be reported if $LC(p_2) \neq LC(p_1)$, because of condition in line 6 of *findPairs*. $u_j$ is common ancestor of $v_1$ and $v_2$. We will show that it is also lowest common ancestor $w$.

We will prove $u_j = w$ by contradiction. Let $u_j = w_i$. Let $U = \{u_1, u_2, \ldots, u_{j-1}\}$ and $W = \{w_1, w_2, \ldots, w_{i-1}\} \setminus U$. If $u_j \neq w$, $w \in U \cup W$. By similar argument as in (if) part of the proof, it can be shown that $w \notin W$. Now let's suppose that $w \in U$ $(\exists m)(1 \leq m < j): u_m = w$. In such case $(p_2, l)$ is reported by a call *combineUnionSubree* on the node $bp(u_m, LC(p_2))$ which is then marked as visited on line 8 before *combineUnionSubtree* on $u_j$ is called. This means however that *combineUnionSubtree* on $bp(u_j, LC(p_2))$ couldn't report pair $(p_2, l)$, which is contradiction.

Since $u_j = w$, it holds $l = lcplen_T(w)$ and since $w = LCA_T(v_1, v_2)$, by lemma 1 we have $(p_1, p_2, l)$ is right maximal repeat. Therefore $(p_1, p_2, l)$ is maximal repeat in $S$ with $l \geq k$.

**Theorem.**

*findPairs* runs in time $O(z)$ where $z$ is number of reported pairs.

Proof:

Let $u_1 = map_T(p_1), u_2 = up_T(u_1, LC(p_1)), \ldots, u_p = up_T(u_{p-1}, LC(p_1))$, be all nodes visited in while loop on lines 4-9.

Let's consider set $C$ of all calls to *combineUnionSubtree* from line 7 in *findPairs*. Let $C_a$ be subset of calls that report at least one pair and $C_b$ subset of calls that don't report any pair because they are called with visited node in argument. By theorem XXTODO total time $t_a$ spent in all $C_a$ calls is $O(z)$. Total time $t_b$ spent by $C_b$ calls is at most $O((p-1)|\Sigma|)$
(if a call occurs in $u_1$, it is $C_a$ call)

From node $u_i$, we continue to node $u_{i+1} = up_T(u_i, LC(p_1))$ if it exists. We know that $\exists b \in \Sigma_{¢}$ $b \neq LC(p_1): b^+(u_{i+1}, b) \supset b^+(u_i, b)$. This means that $bp(u_{i+1}, b)$ is parent of $bp(u_i, b)$ in $UT(u_{i+1}, b)$ and can't have been visited before and $UT(u_{i+1}, b)$ has at least one new bucket that will be visited by *combineUnionSubtree.*

Thus for each node $\{u_2, \ldots, u_p\}$ at least one $C_a$ call is made and $(p-1) \leq |C_a| \leq z$. $t_b$ is therefore $O(z)$too. Initialisation in lines 1-3 takes constant time and time spent in each node for other purpose than for *combineUnionSubtree* calls is also constant. Total time taken by findPairs is therefore $O(z)$.

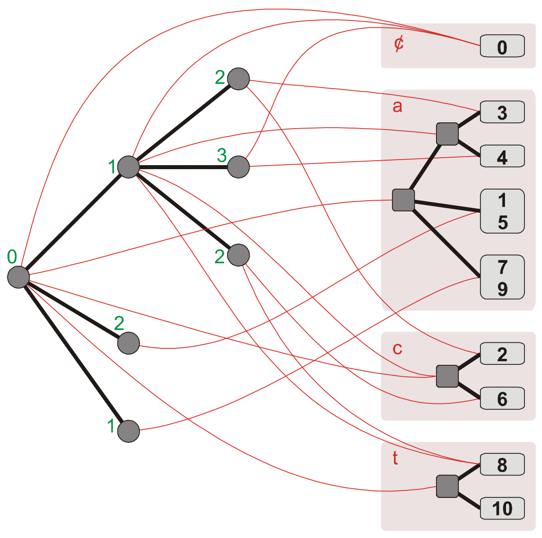

*Figure 2: R3 tree for 'acaaacatat'*