Definitions and notation

Let Σ be finite ordered alphabet. We will use symbols *a, b, ...* for elements of the alphabet. We suppose $|\Sigma|$ is constant. Σ^* is the set of all strings over Σ. Let $<_L$ denote lexicographic ordering on Σ^*. We will use symbols $S, S_1, S_2, x, y, z, w, u$ ... for strings. $|S|$ denotes length of string *S*. We will also use symbol *n* to denote length of a string. We write $S[i]$, where $0 \leqslant i < |S|$, to refer to i-th character of *S*. We define $S[-1] = ¢$ and $S[|S|] = \$$, where $¢, \$ \notin \Sigma$ are special symbols. $S[i..j]$, where $0 \leqslant i \leqslant j < |S|$ refers to substring of *S* starting at position *i* and ending at position *j*. For $0 \leqslant i < |S|$, substring $S[i..|S|-1]$ is called suffix and substring $S[0..i]$ is called prefix. The fact that x is prefix of y, is denoted x⊑y and the fact that x is suffix of y is denoted y⊒x. $N_n = \{0, 1, \ldots, n-1\}$. We will represent suffix $S[i..|S|-1]$ by integer *i*, that represents it's starting position in *S*. Sometimes we will write $suffix_S(i) = S[i..|S|]$ We will write $LC(i)$ (left context of suffix *i)*, to denote $S[i-1]$.

Figure 1: Maximal repeat

A triple $(p_1, p_2, l) \in N_{|S|}^3$ is called **repeat** if $0 \leq p_1 + l \leq |S|$, $0 \leq p_2 + l \leq |S|$, $p_1 \neq p2$ and $S[p_1 .. p_1 + l - 1] = S[p_2 .. p2 + l - 1]$. Repeat (p_1, p_2, l) is called **left maximal** if $S[p_1 - 1] \neq S[p_2 - 1]$ and **right maximal** if $S[p_1 + l] \neq S[p_2 + l]$. A repeat is called **maximal** if it's left and right maximal.

On figure 1, we can see example of maximal repeat (1, 25, 7). Our queries for maximal repeats in string *S* will have form of $Answer := findPairs(p_1, k, S)$, where *findPairs* is a function that returns the set of all pairs (p_2, l) such that (p_1, p_2, l) is maximal repeat in *S* with $l \geq k$. For example the query *findPairs(0, 4, S)* for string from figure 1 would return the set *{ (4, 5), (11,4), (28,4) }*.

Tree *T* is a triple $(V, E, root)$ where *V* is set of nodes, $root \in V$ is the root node, $E \subseteq V \times V$ is set of edges. For all nodes $v \in V \setminus \{root\}$ there is exactly one node $parent(v) \in V$ such that $(parent(v), v) \in E$. For a node $v \in V$ we define $Children(v) = \{u | (v, u) \in E\}$, $Desc(v) = \{u | (v, u) \in E^+\}$, where E^+ is transitive closure of *E*. Depth of a node is defined as follows: $depth(root) = 0$, $depth(v) = depth(parent(v)) + 1$ for $v \in V \setminus \{root\}$. We divide set of nodes *V* into leaves $V_L = \{v | Children(v) = \emptyset\}$, and internal nodes $V_I = V \setminus V_L$. We divide set of nodes *E* into internal edges $E_I = E \cap (V \times V_I)$ and leaf edges $E_L = E \cap (V \times V_L)$.

For a tree T, we use following symbols. $V(T)$ is set of nodes, $E(T)$ is set of edges, $root(T)$ is the root of the tree, $V_I(T)$ is the set of internal nodes, $V_L(T)$ is set of leaves, $E_I(T)$ is the set of internal edges, $E_L(T)$ is the set of leaf edges.

TODO definition of lowest common ancestor

Suffix tree for a string *S* of length *n* is a 5-tuple $ST(S) = (V, E, root, label, path)$

$(V, E, root)$ is a tree. Let V_I, V_L, E_I, E_L denote the same as in the definition of tree.

$label: E \rightarrow \Sigma^+$ is an edge-labeling function , that labels each edge of the tree *T* by some non-empty string.

$path: V \rightarrow \Sigma^*$ is a map from nodes to strings. For a node $v \in V$ and the path $root = v_0, v_1, .., v_k = v$ we define $path(v) = label(v_0, v_1) label(v_1, v_2) .. label(v_{k-1}, v_k)$, $path(root) = \varepsilon$.

Leaves of the suffix tree represent positions of suffixes of *S*. Internal nodes represent sets of positions.

For an internal node $v \in V_I$.
$Pos(v) = Children(v) \cap V_L$
$Pos^+(v) = Desc(v) \cap V_L$

$Pos^{2+}(v)=Pos^{+}(v)\setminus Pos(v)$
For a node $v \in V$
$Pos^{*}(v)=Pos^{+}(v)$ for $v \in V_I$
$Pos^{*}(v)=\{v\}$ for $v \in V_L$.
For an internal node $v \in V_I$, we define $lcplen(v)=|path(v)|$.

Suffix tree satisfies following additional conditions

1. $V_L=N_{n+1}$
2. $\forall i \in V_L: path(i)=S[i..n]$
3. $\forall v \in V_I: |Chidren(v)| \geq 2$
4. $\forall v \in V_I: \forall i \in Pos^{+}(v): path(v) \sqsubseteq path(i)$
5. $\forall v \in V_I, \forall a,b \in \Sigma, x,y \in \Sigma^{*}:$
$$((v,u) \in E \wedge (v,w) \in E \wedge l(v,u)=ax \wedge l(v,w)=by) \Rightarrow a \neq b$$

In other words, (1) the leaves of *T* represent positions of all suffixes of *S*, (2) concatenation of labels on the path from root to a leaf spells exactly the suffix represented by the leaf, (3) each internal node is a branching node – it has at least 2 children, (4) the internal nodes represent common prefix of their descendants and (5) all labels of edges outgoing from a node must begin with distinct characters. Note that suffix tree definition uses suffixes that are extended to the right endmarker *$*.

We deal with suffix trees, because they have an interresting property from the point of view of maximal repeats. Each internal node *v* of a suffix tree represents the longest common prefix *path(v)* of it's descendants. Moving to a child of the vertex *v* means extending the prefix. For suffixes under two distinct children (that are either leaf or internal nodes) the prefix *path(v)* is not right-extensible.

Property.

Let *v* be an internal node of suffix tree *T* for *S*, *l=lcplen(v)*, $c_1, c_2 \in Children(v)$, $c_1 \neq c_2$, $p_1 \in Pos^{*}(c_1)$, $p_2 \in Pos^{*}(c_2)$.Then
(p_1, p_2, l) is right maximal repeat in *S*.

Lemma.

Let $T=ST(S)$ and $p_1, p_2, p_1 \neq p_2$, be positions of suffixes of *S*. Let $v_1=map_T(p_1)$, $v_2=map_T(p_2)$. $w=LCA_T(v_1, v_2)$. Then

(p_1, p_2, l) is right maximal repeat in *S* if and only if $l=lcplen_T(w)$

Proof : It holds that $\exists c_1, c_2 \in Children(w)$, $p_1 \in Pos^{*}(c_1)$, $p_2 \in Pos^{*}(c_2)$. Also $c_1 \neq c_2$ holds, because $w=LCA_T(v_1, v_2)$. Property XXX, $(p_1, p_2, lcplen_T(w))$ is right maximal repeat in *S*. There can't be right maximal repeat (p_1, p_2, l) for *l* other than $lcplen_T(w)$, because it would contradict with properties of right maximal repeat.

Let *S* be a string of length *n*, let's have suffix tree $T=ST(S)$. We define function $map: N_{n+1} \rightarrow V_I$ such that $\forall i \in N_{n+1}: i \in Pos(map(i))$ i.e. map returns node *v* such that *i* is child of *v*. Function *map* can be realised by table that can be easily precomputed in $O(n)$ time by one traversal of *T*. Value $map(i)$ can be therefore accessed in $O(1)$ time.

For $T=ST(S)$, $V(T)$, $E(T)$, $root(T)$, $V_I(T)$, $V_L(T)$, $E_I(T)$, $E_L(T)$ have the same meaning as in the definition of tree. $path_T$, $label_T$, $lcplen_T$, Pos_T, Pos_T^{+}, map_T denote *path, label, lcplen, Pos*, Pos^{+} and *map* functions for *T*.

It is known that suffix tree can be built in $O(n)$ time and space using algorithms of ... TODO

Suffix array is a permutation $sa_S: N_{|S|+1} \rightarrow N_{|S|+1}$ such that

$\forall i, j: 0 \le i < j \le |S|:$
$$suffix_S(sa_S(i)) <_L suffix_S(sa_S(j)). \blacksquare$$

For $0 \le i < j \le |S|$, and and suffix array sa_S, we define $sa_S[i..j] = \{sa_S(i), sa_S(i+1), .., sa(j)\}$
Let's define function $lcplen_2: \Sigma^* \times \Sigma^* \to N$ returning length of longest common prefix of two strings, $\forall S_1, S_2: lcplen_2(S_1, S_2) = max\{l \ge 1 | S_1[0..l-1] = S_2[0..l-1]\}$ (let's suppose that that *max* for empty set is 0).

Lcp-table is a function $lcp_S: (N_{|S|+1} - \{0\}) \to N_{|S|}$ defined as follows

$\forall i: 1 \le i \le |S|:$
$$lcp_S(i) = lcplen_2(suffix_S(sa_S(i-1)), suffix_S(sa_S(i))). \blacksquare$$

Lcp-interval is a triple *(l, i, j)* that satisfies all following conditions

$lcpinterval_S(l, i, j) \Leftrightarrow$
1. $0 \le i < j < |S|$
2. $lcp_S(i) < l$
3. $\forall k: i+1 \le k \le j: lcp_S(k) \ge l$
4. $\exists k: i+1 \le k \le j: lcp_S(k) = l$
5. $lcp_S(j+1) < l$

For an $lcpinterval_S(l, i, j)$, we'll write $prefix_S(l, i, j)$ to denote the longest common prefix of all suffixes $suffix_S(sa_S(i)), suffix_S(sa_S(i+1)), \ldots, suffix_S(sa_S(j))$. Sometimes, instead of triples, we'll use symbols ***I***, ***J***, .. for lcp-intervals. For interval ***I*** = *(l,i,j)* we define ***I***.*lcp* = *l*, ***I***.*left* = *i*, ***I***.*right* = *j*. $\blacksquare$

Lcp-inerval *(m, p, q)* is said to be **embedded** in an lcp-interval *(l, i, j)* if it is subitnerval of *(l, i, j):*

$embedded_S((m, p, q), (l, i, j)) \Leftrightarrow$
1. $lcpinterval_S(l, i, j)$
2. $lcpinterval_S(m, p, q)$
3. $i \le p < q \le j$
4. $m > l$ [1]

(l, i, j) is then called the interval **enclosing** *(m, p, q)*. We call *(m, p, q)* a **child interval** of *(l, i, j)* if it is embedded in *(l, i, j)* and there is no interval embedded in *(l, i, j)* that also encloses *(m, p, q):*

$child_S((m, p, q), (l, i, j)) \Leftrightarrow$
1. $embedded_S((m, p, q), (l, i, j))$
2. $\neg \exists (r, s, t): embedded_S((m, p, q), (r, s, t)) \wedge embedded_S((r, s, t), (l, i, j))$

The predicate $child_S$, can be read also as a relation over lcp-intervals. This parent-child relation defines lcp-interval tree. Let's define set of descendants for given lcp-interval *I*:

1 Note that we cannot have both *i=p* and *q=j* because *m>l*

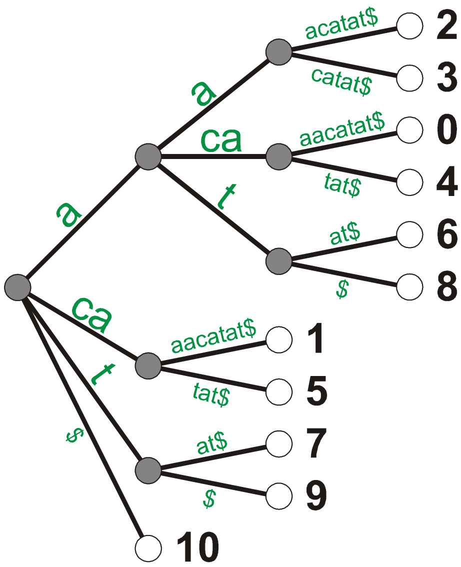

Figure 2: Suffix tree for string 'acaaacatat'

TODO isomorphism of suffix tree and lcp-interval tree