7.12. API.11: Finding all maximal repetitive structures in linear time

Before developing algorithms for finding repetitive structures, we must carefully define those structures. A poor definition may lead to an avalanche of output. For example, if a string consists of $n$ copies of the same character, an algorithm searching for all pairs of identical substrings (an initially reasonable definition of a repetitive structure) would output $\Theta(n^4)$ pairs, an undesirable result. Other poor definitions may not capture the structures of interest, or they may make reasoning about those structures difficult. Poor definitions are particularly confusing when dealing with the set of all repeats of a particular type. Accordingly, the key problem is to define repetitive structures in a way that does not generate overwhelming output and yet captures all the meaningful phenomena in a clear way. In this section, we address the issue through various notions of maximality. Other ways of defining and studying repetitive structures are addressed in Exercises 56, 57, and 58 in this chapter; in exercises in other chapters; and in Sections 9.5, 9.6, and 9.6.1.

**Definition** A maximal pair (or a maximal repeated pair) in a string $S$ is a pair of identical substrings $\alpha$ and $\beta$ in $S$ such that the character to the immediate left (right) of $\alpha$ is different from the character to the immediate left (right) of $\beta$. That is, extending $\alpha$ and $\beta$ in either direction would destroy the equality of the two strings.

**Definition** A maximal pair is represented by the triple $(p_1, p_2, n')$, where $p_1$ and $p_2$ give the starting positions of the two substrings and $n'$ gives their length. For a string $S$, we define $\mathcal{R}(S)$ to be the set of all triples describing maximal pairs in $S$.

For example, consider the string $S = xacyiizabcqabcyrxar$, where there are three occurrences of the substring $abc$. The first and second occurrences of $abc$ form a maximal pair $(2, 10, 3)$, and the second and third occurrences also form a maximal pair $(10, 14, 3)$, whereas the first and third occurrences of $abc$ do not form a maximal pair. The two occurrences of the string $abc$ also form a maximal pair $(2, 14, 4)$. Note that the definition allows the two substrings in a maximal pair to overlap each other. For example, $cxaxaxaxxb$ contains a maximal pair whose substring is $xaxaxx$.

Generally, we also want to permit a prefix or a suffix of $S$ to be part of a maximal pair. For example, two occurrences of $xa$ in $xacyiizabcqabcyrxar$ should be considered as a maximal pair. To model this case, simply add a character to the start of $S$ and one to the end of $S$ that appear nowhere else in $S$. From this point on, we will assume that has been done.

It may sometimes be of interest to explicitly find and output the full set $\mathcal{R}(S)$. However, in some situations $\mathcal{R}(S)$ may be too large to be of use, and a more restricted reflection of the maximal pairs may be sufficient or even preferred.

**Definition** Define a maximal repeat $\alpha$ as a substring of $S$ that occurs in a maximal pair in $S$. That is, $\alpha$ is a maximal repeat in $S$ if there is a triple $(p_1, p_2, |\alpha|) \in \mathcal{R}(S)$ and $\alpha$ occurs in $S$ starting at position $p_1$ and $p_2$. Let $\mathcal{R}'(S)$ denote the set of maximal repeats in $S$.

For example, with $S$ as above, both strings $abc$ and $abcy$ are maximal repeats. Note that no matter how many times a string participates in a maximal pair in $S$, it is represented only once in $\mathcal{R}'(S)$. Hence $|\mathcal{R}'(S)|$ is less than or equal to $|\mathcal{R}(S)|$ and is generally much smaller. The output is more modest, and yet it gives a good reflection of the maximal pairs.

In some applications, the definition of a maximal repeat does not properly model the desired notion of a repetitive structure. For example, in $S = aabxayaab$, substring $\alpha$ is
a maximal repeat but so is \textit{aab}, which is a \textit{superstring} of string \( \alpha \), although not every occurrence of \( \alpha \) is contained in that superstring. It may not always be desirable to report \( \alpha \) as a repetitive structure, since the larger substring \( aab \) that sometimes contains \( \alpha \) may be more informative.

**Definition** A \textit{supermaximal repeat} is a maximal repeat that never occurs as a substring of any other maximal repeat.

Maximal pairs, maximal repeats, and supermaximal repeats are only three possible ways to define exact repetitive structures of interest. Other models of exact repeats are given in the exercises. Problems related to palindromes and tandem repeats are considered in several sections throughout the book. Inexact repeats will be considered in Sections 9.5 and 9.6.1. Certain kinds of repeats are elegantly represented in graphical form in a device called a \textit{landscape} [104]. An efficient program to construct the landscape, based essentially on suffix trees, is also described in that paper. In the next sections we detail how to efficiently find all maximal pairs, maximal repeats, and supermaximal repeats.

### 7.12.1. A linear-time algorithm to find all maximal repeats

The simplest problem is that of finding all maximal repeats. Using a suffix tree, it is possible to find them in \( O(n) \) time for a string of length \( n \). Moreover, there is a \textit{compact} representation of all the maximal repeats, and it can also be constructed in \( O(n) \) time, even though the total length of all the maximal repeats may be \( \Omega(n^2) \). The following lemma states a necessary condition for a substring to be a maximal repeat.

**Lemma 7.12.1.** Let \( T \) be the suffix tree for string \( S \). If a string \( \alpha \) is a maximal repeat in \( S \) then \( \alpha \) is the path-label of a node \( v \) in \( T \).

**Proof** If \( \alpha \) is a maximal repeat then there must be at least two copies of \( \alpha \) in \( S \) where the character to the right of the first copy differs from the character to the right of the second copy. Hence \( \alpha \) is the path-label of a node \( v \) in \( T \). \( \square \)

The key point in Lemma 7.12.1 is that path \( \alpha \) must end at a node of \( T \). This leads immediately to the following surprising fact:

**Theorem 7.12.1.** There can be at most \( n \) maximal repeats in any string of length \( n \).

**Proof** Since \( T \) has \( n \) leaves, and each internal node other than the root must have at least two children, \( T \) can have at most \( n \) internal nodes. Lemma 7.12.1 then implies the theorem. \( \square \)

Theorem 7.12.1 would be a trivial fact if at most one substring starting at any position \( i \) could be part of a maximal pair. But that is not true. For example, in the string \( S = xabcyiizabcqabcyr \) considered earlier, both copies of substring \( abcy \) participate in maximal pairs, while each copy of \( abc \) also participates in maximal pairs.

So now we know that to find maximal repeats we only need to consider strings that end at nodes in the suffix tree \( T \). But which specific nodes correspond to maximal repeats?

**Definition** For each position \( i \) in string \( S \), character \( S(i-1) \) is called the \textit{left character of} \( i \). The \textit{left character of a leaf} of \( T \) is the left character of the suffix position represented by that leaf.

**Definition** A node \( v \) of \( T \) is called \textit{left diverse} if at least two leaves in \( v \)’s subtree have different left characters. By definition, a leaf cannot be left diverse.
Note that being left diverse is a property that propagates upward. If a node \( v \) is left diverse, so are all of its ancestors in the tree.

**Theorem 7.12.2.** The string \( \alpha \) labeling the path to a node \( v \) of \( T \) is a maximal repeat if and only if \( v \) is left diverse.

**Proof.** Suppose first that \( v \) is left diverse. That means there are substrings \( xa \) and \( ya \) in \( S \), where \( x \) and \( y \) represent different characters. Let the first substring be followed by character \( p \). If the second substring is followed by any character but \( p \), then \( \alpha \) is a maximal repeat and the theorem is proved. So suppose that the two occurrences are \( xap \) and \( yap \). But since \( v \) is a (branching) node there must also be a substring \( aq \) in \( S \) for some character \( q \) that is different from \( p \). If this occurrence of \( aq \) is preceded by character \( x \) then it participates in a maximal pair with string \( yap \), and if it is preceded by \( y \) then it participates in a maximal pair with \( xap \). Either way, \( \alpha \) cannot be preceded by both \( x \) and \( y \), so \( \alpha \) must be part of a maximal pair and hence \( \alpha \) must be a maximal repeat.

Conversely, if \( \alpha \) is a maximal repeat then it participates in a maximal pair and there must be occurrences of \( \alpha \) that have distinct left characters. Hence \( v \) must be left diverse.

\( \Box \)

**The maximal repeats can be compactly represented**

Since the property of being left diverse propagates upward in \( T \), Theorem 7.12.2 implies that the maximal repeats of \( S \) are represented by some initial portion of the suffix tree for \( S \). In detail, a node is called a "frontier" node in \( T \) if it is left diverse but none of its children are left diverse. The subtree of \( T \) from the root down to the frontier nodes precisely represents the maximal repeats in that every path from the root to a node at or above the frontier defines a maximal repeat. Conversely, every maximal repeat is defined by one such path. This subtree, whose leaves are the frontier nodes in \( T \), is a compact representation of the set of all maximal repeats of \( S \). Note that the total length of the maximal repeats could be as large as \( \Theta(n^2) \), but since the representation is a subtree of \( T \) it has \( O(n) \) total size (including the symbols used to represent edge labels). So if the left diverse nodes can be found in \( O(n) \) time, then a tree representation for the set of maximal repeats can be constructed in \( O(n) \) time, even though the total length of those maximal repeats could be \( \Omega(n^2) \). We now describe an algorithm to find the left diverse nodes in \( T \).

**Finding left diverse nodes in linear time**

For each node \( v \) of \( T \), the algorithm either records that \( v \) is left diverse or it records the character, denoted \( x \), that is the left character of every leaf in \( v \)'s subtree. The algorithm starts by recording the left character of each leaf of the suffix tree \( T \) for \( S \). Then it processes the nodes in \( T \) bottom up. To process a node \( v \), it examines the children of \( v \). If any child of \( v \) has been identified as being left diverse, then it records that \( v \) is left diverse. If none of \( v \)'s children are left diverse, then it examines the characters recorded at \( v \)'s children. If these recorded characters are all equal, say \( x \), then it records character \( x \) at node \( v \). However, if they are not all \( x \), then it records that \( v \) is left diverse. The time to check if all children of \( v \) have the same recorded character is proportional to the number of \( v \)'s children. Hence the total time for the algorithm is \( O(n) \). To form the final representation of the set of maximal repeats, simply delete all nodes from \( T \) that are not left diverse. In summary, we have

---

5 This kind of tree is sometimes referred to as a compact tree, but we will not use that terminology.
**Theorem 7.12.3.** All the maximal repeats in $S$ can be found in $O(n)$ time, and a tree representation for them can be constructed from suffix tree $T$ in $O(n)$ time as well.

### 7.12.2. Finding supermaximal repeats in linear time

Recall that a supermaximal repeat is a maximal repeat that is not a substring of any other maximal repeat. We establish here efficient criteria to find all the supermaximal repeats in a string $S$. To do this, we solve the more general problem of finding near-supermaximal repeats.

**Definition** A substring $\alpha$ of $S$ is a near-supermaximal repeat if $\alpha$ is a maximal repeat in $S$ that occurs at least once in a location where it is not contained in another maximal repeat. Such an occurrence of $\alpha$ is said to witness the near-supermaximality of $\alpha$.

For example, in the string $aabxyaaba$, substring $\alpha$ is a maximal repeat but not a supermaximal or a near-supermaximal repeat, whereas in $aabxyaaba$, substring $\alpha$ is again not supermaximal, but it is near-supermaximal. The second occurrence of $\alpha$ witnesses that fact.

With this terminology, a supermaximal repeat $\alpha$ is a maximal repeat in which every occurrence of $\alpha$ is a witness to its near-supermaximality. Note that it is not true that the set of near-supermaximal repeats is the set of maximal repeats that are not supermaximal repeats.

The suffix tree $T$ for $S$ will be used to locate the near-supermaximal and the supermaximal repeats. Let $v$ be a node corresponding to a maximal repeat $\alpha$, and let $w$ (possibly a leaf) be one of $v$'s children. The leaves in the subtree of $T$ rooted at $w$ identify the locations of some (but not all) of the occurrences of substring $\alpha$ in $S$. Let $L(w)$ denote those occurrences. Do any of those occurrences of $\alpha$ witness the near-supermaximality of $\alpha$?

**Lemma 7.12.2.** If node $w$ is an internal node in $T$, then none of the occurrences of $\alpha$ specified by $L(w)$ witness the near-supermaximality of $\alpha$.

**Proof** Let $\gamma$ be the substring labeling edge $(v, w)$. Every index in $L(w)$ specifies an occurrence of $\alpha \gamma$. But $w$ is internal, so $|L(w)| > 1$ and $\alpha \gamma$ is the prefix of a maximal repeat. Therefore, all the occurrences of $\alpha$ specified by $L(w)$ are contained in a maximal repeat that begins $\alpha \gamma$, and $w$ cannot witness the near-supermaximality of $\alpha$. $\Box$

Thus no occurrence of $\alpha$ in $L(w)$ can witness the near-supermaximality of $\alpha$ unless $w$ is a leaf. If $w$ is a leaf, then $w$ specifies a single particular occurrence of substring $\beta = \alpha \gamma$. We now consider that case.

**Lemma 7.12.3.** Suppose $w$ is a leaf, and let $i$ be the (single) occurrence of $\beta$ represented by leaf $w$. Let $x$ be the left character of leaf $w$. Then the occurrence of $\alpha$ at position $i$ witnesses the near-supermaximality of $\alpha$ if and only if $x$ is the left character of no other leaf below $v$.

**Proof** If there is another occurrence of $\alpha$ with a preceding character $x$, then $x\alpha$ occurs twice and so is either a maximal repeat or is contained in one. In that case, the occurrence of $\alpha$ at $i$ is contained in a maximal repeat.

If there is no other occurrence of $\alpha$ with a preceding $x$, then $x\alpha$ occurs only once in $S$. Now let $y$ be the first character on the edge from $v$ to $w$. Since $w$ is a leaf, $\alpha y$ occurs only once in $S$. Therefore, the occurrence of $\alpha$ starting at $i$, which is preceded...
by $x$ and succeeded by $y$, is not contained in a maximal repeat, and so witnesses the near-supermaximality of $\alpha$. □

In summary, we can state

**Theorem 7.12.4.** A left diverse internal node $v$ represents a near-supermaximal repeat $\alpha$ if and only if one of $v$'s children is a leaf (specifying position $i$, say) and its left character, $S(i - 1)$, is the left character of no other leaf below $v$. A left diverse internal node $v$ represents a supermaximal repeat $\alpha$ if and only if all of $v$'s children are leaves, and each has a distinct left character.

Therefore, all supermaximal and near-supermaximal repeats can be identified in linear time. Moreover, we can define the degree of near-supermaximality of $\alpha$ as the fraction of occurrences of $\alpha$ that witness its near-supermaximality. That degree of each near-supermaximal repeat can also be computed in linear time.

### 7.12.3. Finding all the maximal pairs in linear time

We now turn to the question of finding all the maximal pairs. Since there can be more than $O(n)$ of them, the running time of the algorithm will be stated in terms of the size of the output. The algorithm is an extension of the method given earlier to find all maximal repeats.

First, build a suffix tree for $S$. For each leaf specifying a suffix $i$, record its left character $S(i - 1)$. Now traverse the tree from bottom up, visiting each node in the tree. In detail, work from the leaves upward, visiting a node $v$ only after visiting every child of $v$. During the visit to $v$, create at most $\sigma$ linked lists at each node, where $\sigma$ is the size of the alphabet. Each list is indexed by a left character $x$. The list at $v$ indexed by $x$ contains all the starting positions of substrings in $S$ that match the string on the path to $v$ and that have the left character $x$. That is, the list at $v$ indexed by $x$ is just the list of leaf numbers below $v$ that specify suffixes in $S$ that are immediately preceded by character $x$.

Letting $n$ denote the length of $S$, it is easy to create (but not keep) these lists in $O(n)$ total time, working bottom up in the tree. To create the list for character $x$ at node $v$, link together (but do not copy) the lists for character $x$ that exist for each of $v$'s children. Because the size of the alphabet is finite, the time for all linking is constant at each node. Linking without copying is required in order to achieve the $O(n)$ time bound. Linking a list created at a node $v'$ to some other list destroys the list for $v'$. Fortunately, the lists created at $v'$ will not be needed after the lists for its parent are created.

Now we show in detail how to use the lists available at $v$'s children to find all maximal pairs containing the string that labels the path to $v$. At the start of the visit to node $v$, before $v$'s lists have been created, the algorithm can output all maximal pairs $(p_1, p_2, \alpha)$, where $\alpha$ is the string labeling the path to $v$. For each character $x$ and each child $v'$ of $v$, the algorithm forms the Cartesian product of the list for $x$ at $v'$ with the union of every list for a character other than $x$ at a child of $v$ other than $v'$. Any pair in this list gives the starting positions of a maximal pair for string $\alpha$. The proof of this is essentially the same as the proof of Theorem 7.12.2.

If there are $k$ maximal pairs, then the method works in $O(n + k)$ time. The creation of the suffix tree, its bottom up traversal, and all the list linking take $O(n)$ time. Each operation used in a Cartesian product produces a maximal pair not produced anywhere else, so $O(k)$ time is used in those operations. If we only want to count the number of
maximal pairs, then the algorithm can be modified to run in $O(n)$ time. If only maximal pairs of a certain minimum length are requested (this would be the typical case in many applications), then the algorithm can be modified to run in $O(n + k_m)$ time, where $k_m$ is the number of maximal pairs of length at least the required minimum. Simply stop the bottom-up traversal at any node whose string-depth falls below that minimum.

In summary, we have the following theorem:

**Theorem 7.12.5.** All the maximal pairs can be found in $O(n + k)$ time, where $k$ is the number of maximal pairs. If there are only $k_m$ maximal pairs of length above a given threshold, then all those can be found in $O(n + k_m)$ time.

### 7.13. APL12: Circular string linearization

Recall the definition of a circular string $S$ given in Exercise 2 of Chapter 1 (page 11). The characters of $S$ are initially numbered sequentially from 1 to $n$ starting at an arbitrary point in $S$.

**Definition** Given an ordering of the characters in the alphabet, a string $S_1$ is *lexically* (or *lexicographically*) smaller than a string $S_2$ if $S_1$ would appear before $S_2$ in a normal dictionary ordering of the two strings. That is, starting from the left end of the two strings, if $i$ is the first position where the two strings differ, then $S_1$ is lexically less than $S_2$ if and only if $S_1(i)$ precedes $S_2(i)$ in the ordering of the alphabet used in those strings.

To handle the case that $S_1$ is a proper prefix of $S_2$ (and should be considered lexically less than $S_2$), we follow the convention that a space is taken to be the first character of the alphabet.

The *circular string linearization problem* for a circular string $S$ of $n$ characters is the following: Choose a place to cut $S$ so that the resulting linear string is the lexically smallest of all the $n$ possible linear strings created by cutting $S$.

This problem arises in chemical databases for circular molecules. Each such molecule is represented by a circular string of chemical characters; to allow faster lookup and comparisons of molecules, one wants to store each circular string by a *canonical* linear string. A single circular molecule may itself be a part of a more complex molecule, so this problem arises in the “inner loop” of more complex chemical retrieval and comparison problems.

A natural choice for canonical linear string is the one that is lexically least. With suffix trees, that string can be found in $O(n)$ time.

#### 7.13.1. Solution via suffix trees

Arbitrarily cut the circular string $S$, giving a linear string $L$. Then, double $L$, creating the string $LL$, and build the suffix tree $T$ for $LL$. As usual, affix the terminal symbol $\$\$ at the end of $LL$, but interpret it to be lexically *greater* than any character in the alphabet used for $S$. (Intuitively, the purpose of doubling $L$ is to allow efficient consideration of strings that begin with a suffix of $L$ and end with a prefix of $L$.) Next, traverse tree $T$ with the rule that, at every node, the traversal follows the edge whose first character is lexically smallest over all first characters on edges out of the node. This traversal continues until the traversed path has string-depth $n$. Such a depth will always be reached (with the proof left to the reader). Any leaf $l$ in the subtree at that point can be used to cut the string. If $1 < l \leq n$, then cutting $S$ between characters $l - 1$ and $l$ creates a lexically smallest