1 Preliminaries; depth-first search; easy cases

Notations. Depth-first search provides a numbering of the vertices, and it classifies the edges of the tree as *tree edges* or *backedges*. We orient tree edges away from the root and backedges towards the root. We denote this by $v \underset{T}{\rightarrow} w$ for tree edges and $v \underset{R}{\rightarrow} w$ for backedges. We denote by $\overset{*}{\underset{T}{\rightarrow}}$ or $\overset{+}{\underset{T}{\rightarrow}}$ a path of zero or more (one or more) tree edges. $v \overset{*}{\underset{T}{\rightarrow}} w$ means that v is an ancestor of w in the tree, including the possibility that $v = w$.

We will assume that vertices are ordered according to DFS-numbers. When we write $v < w$ for two vertices v and w we will refer to this numbering. In fact we will compare two vertices in this way only if one is an ancestor of the other one in the tree.

$v \underset{T}{\rightarrow} w \implies v < w$, $v \underset{R}{\rightarrow} w \implies v > w$.

Later we will compute different indices and numberings for vertices (and edges), but this will be indicated explicitly.

Define $LOW(v)$ and $LOW2(v)$.

Lemma 1 *Let G be connected. If $v_0 \underset{T}{\rightarrow} v_1$ is a tree edge with $LOW(v_1) \geq v_0$ then v_0 is a splitting vertex (an articulation vertex), unless v_0 is the root of the tree and (v_0, v_1) is the only tree edge leaving v_0.*

Lemma 2 *Let G be biconnected. If $v_0 \underset{T}{\rightarrow} v_1$ is a tree edge with $LOW2(v_1) \geq v_0$ then $\{LOW2(v_1), LOW(v_1)\}$ is a splitting pair of vertices, unless*

(i) v_0 is the root of the tree (and then (v_0, v_1) is the only tree edge leaving v_0), or

(ii) v_0 is the only child of the root and (v_0, v_1) is the only tree edge leaving v_0.

These splitting pairs are easy to find in linear time, by computing LOW and $LOW2$ pointers. Also if the graph has a vertex of degree two (and more than three vertices), then its two neighbors are clearly a splitting pair. This case, and the case identified by Lemma 2 are called *easy splitting pairs*.

We will from now on assume that the graph is biconnected, a dfs-tree has been computed, and no easy splitting pairs have been found.

1.1 Reordering of the adjacency lists

We now describe the reordering of the adjacency lists. The adjacency list of vertex v contains all directed edges leading to the children in the dfs tree, and all backedges that lead to ancestors. Thus each edge of the graph occurs only once as a directed edge in such an adjacency list.

Let us define $LOW(e)$ and a Boolean predicate *has_proper_attachments*(e) for an edge $e = u \to v$ as follows.

$$LOW(e) := \begin{cases} LOW(v), & \text{if } e = (u,v) \text{ is a tree edge} \\ v, & \text{if } e = (u,v) \text{ is a back-edge} \end{cases}$$

The predicate *has_proper_attachments*(e) is true if and only if $e = (u,v)$ is a tree edge and $LOW2(v) < u$.

We sort the edges leaving a vertex v lexicographically according to two criteria. The first criterion is the LOW index. e comes before f whenever $LOW(e) < LOW(f)$ The secondary criterion is the $LOW2$ index. If $LOW(e) = LOW(f)$ then e comes before f whenever *has_proper_attachments*(e) is false and *has_proper_attachments*(e) is true. In all other cases, the order of e and f can be arbitrary.

[In fact, the secondary criterion is not really needed for the algorithm to work correctly. However, it is a little complicated to argue why it is not needed. [Hopcroft and Tarjan cite[p. xxx]HT mention that they need the secondary criterion only as a convenient way to ensure that parallel edges that result from splitting off ..components end up next to each other in the adjacency list. In fact, it seems that they use the secondary criterion in the opposite way than mentioned above. The above ordering is the one that must be used for testing planarity.]

2 The Cycle Tree

For a tree edge $e = (v_0, v_1)$ we define the *spine path* (Hauptweg) $v_0 \underset{T}{\to} v_1 \overset{*}{\underset{T}{\to}} v_k \underset{B}{\to} v_{k+1} = LOW(v_1)$ $(k \geq 1)$ as follows. We select the first edge $v_1 \to v_2$ leaving v_1, continue from v_2 in the same way, and go on until we use a back edge for the first time. By the reordering of adjacency lists we have ensured that we end up in $LOW(v_1)$.

Together with the tree path from $v_{k+1} = v_{-l}$ to v_0, this forms the spine cycle $C(e)$ of e.

$$LOW(v_1) = v_{-l} \underset{T}{\rightarrow} v_{-l+1} \overset{*}{\underset{T}{\rightarrow}} v_0 \underset{T}{\rightarrow} v_1 \overset{*}{\underset{T}{\rightarrow}} v_k \underset{B}{\rightarrow} v_{k+1} = LOW(v_1) = v_{-l}$$

The tree path from $v_{-l} \overset{+}{\underset{T}{\rightarrow}} v_0$ is called the *stem* of the cycle.

Lemma 3 *Let $S < T$ be two segments of $C(e)$ within $S(e)$. Then S and T interlace iff S has an attachment x with $LOW(T) < x < HIGH(T)$.*

By $\mathcal{A}$ be denote the segment graph of $C(e)$ within the subgraph $S(e) \cup C(e)$.

We will construct $\mathcal{A}$ incrementally, adding nodes (segments) and edges between them.

We define the partial segment graph $\mathcal{A}_i$ as follows, for $i = k, k-1, \ldots$. The nodes of $\mathcal{A}_i$ are the segments S of $C(e)$ within $S(e)$ with $high(S) > v_i$. Two nodes $S < T$ are linked by an edge of $\mathcal{A}_i$ if S has an attachment x with $LOW(T) < x < HIGH(T)$ and $x \geq v_i$.

It is clear that, for decreasing i, $\mathcal{A}_i$ gets more and more nodes and edges, and the final graph $\mathcal{A}_{-l}$ is equal to $\mathcal{A}$.

We maintain the following information about the connected components of $\mathcal{A}_i$.