ORDERLY SPANNING TREES WITH APPLICATIONS

YI-TING CHIANG†, CHING-CHI LIN†, AND HSUEH-I LU‡

Abstract. We introduce and study orderly spanning trees of plane graphs. This algorithmic tool generalizes canonical orderings, which exist only for triconnected plane graphs. Although not every plane graph admits an orderly spanning tree, we provide an algorithm to compute an orderly pair for any connected planar graph G, consisting of an embedded planar graph H isomorphic to G, and an orderly spanning tree of H. We also present several applications of orderly spanning trees: (1) a new constructive proof for Schnyder’s realizer theorem, (2) the first algorithm for computing an area-optimal 2-visibility drawing of a planar graph, and (3) the most compact known encoding of a planar graph with $O(1)$-time query support. All algorithms in this paper run in linear time.

Key words. planar graph algorithm, graph drawing, realizer, visibility representation, canonical ordering, orderly spanning tree, graph encoding, triangulation, unit-cost RAM model, succinct data structure, data compression

AMS subject classifications. 05C62, 05C85, 68P05, 68W35, 68U05, 68R10, 94C15

DOI. 10.1137/S0097539702411381

1. Introduction. A plane graph is a planar graph equipped with a plane embedding. Canonical orderings of triconnected plane graphs [11, 19, 27, 28] are crucial in several graph-drawing and graph-encoding algorithms [7, 8, 9, 16, 20, 22]. This paper introduces the orderly spanning tree as an algorithmic tool that generalizes the concept of canonical orderings for plane graphs that are not required to be triconnected. The concept of orderly spanning trees of plane graphs originates from that of canonical spanning trees of triconnected plane graphs [9], but the former is more general even for triconnected plane graphs (see section 2.1 and Figure 2.1(b)).

We say that $(H, T)$ is an orderly pair of $G$ if (1) $T$ is an orderly spanning tree of plane graph $H$, and (2) $G$ and $H$, ignoring their embeddings, are isomorphic planar graphs. Although not every connected plane graph admits an orderly spanning tree (see section 2.1 and Figure 2.2(a)), we provide a linear-time algorithm (see section 2.2) to compute an orderly pair for any connected planar or plane graph. We also present three applications of orderly spanning trees in the paper.

Application 1: Realizers of planar graphs. A graph is simple if it contains no multiple edges. For the first application of orderly spanning trees, we present a new linear-time algorithm to compute a realizer for any plane triangulation (i.e., simple triangulated plane graph with at least three nodes). Schnyder [38] gave the first known linear-time algorithm that computes a realizer for any plane triangulation, thereby settling the open question on the dimension [42, 14] of planar graphs. This celebrated result also yields the best known straight-line drawing of planar graphs on
the grid [39]. Our proof, based upon the existence of an orderly spanning tree for any simple plane triangulation, is quite simple.

**Application 2: Optimal 2-visibility drawings of planar graphs.** For the second application of orderly spanning trees, we give an $O(n)$-time algorithm that produces a 2-visibility drawing of area at most $(n - 1) \times \left\lceil \frac{2n+1}{3} \right\rceil$ for any $n$-node simple plane graph $H$ with $n \geq 3$. Let $v_1, v_2, \ldots, v_n$ be the nodes of $H$. A 2-visibility drawing [16] of $H$ consists of $n$ nonoverlapping rectangles $b_1, b_2, \ldots, b_n$ such that if $v_i$ and $v_j$ are adjacent in $H$, then $b_i$ and $b_j$ are visible to each other either horizontally or vertically.\(^1\) For example, the picture in Figure 1.1(a) is a 2-visibility drawing of the plane graph in Figure 1.1(a). Fößmeier, Kant, and Kaufmann [16] gave an $O(n)$-time algorithm to compute an $x \times y$ 2-visibility drawing for $H$ with $x + y \leq 2n$ and conjectured that it is “not trivial” to improve their upper bound. Moreover, they showed an $n$-node plane triangulation whose $x \times y$ 2-visibility drawing requires $x + y \geq n - 1 + \left\lceil \frac{2n+1}{3} \right\rceil$ and $\min\{x, y\} \geq \left\lceil \frac{2n+1}{3} \right\rceil$.\(^2\) According to their lower bounds, the 2-visibility drawing produced by our algorithm is worst-case optimal.

In order to take advantage of the wonderful properties of canonical orderings, many drawing algorithms work on triangulated versions of input plane graphs. As pointed out in [19], the initial triangulation tends to ruin the original plane graph’s structure. Our orderly pair algorithm appears as a promising tool for drawing graphs neatly and compactly, without first triangulating the given plane graphs. The concept of an orderly pair is more general than that of a canonical ordering, since all known canonical orderings are only defined for triconnected plane graphs. The technique of orderly pairs is potentially more powerful, since it exploits the flexibility of planar graphs whose planar embeddings are not predetermined.

**Application 3: Convenient encodings of planar graphs.** For the third application of orderly spanning trees, we investigate the problem of encoding a graph $G$ into a binary string $S$ with the requirement that $S$ can be decoded to reconstruct $G$. This problem has been extensively studied with three objectives: (1) minimizing the length of $S$, (2) minimizing the time required to compute and decode $S$, and (3) supporting queries efficiently. As these objectives are often conflicting, a number of coding schemes with different trade-offs have been proposed in the literature. The widely useful adjacency-list encoding of an $n$-node $m$-edge graph $G$ requires $2m\lceil \log_2 n \rceil$ bits. See [9, 22, 23, 29, 34, 37, 43] for $O(n)$-bit encodings of $n$-node planar graphs without efficient query supports.

Under the model of unit-cost RAM [5, 10, 17, 40, 41, 45], where operations such as read, write, and add on $O(\log n)$ consecutive bits take $O(1)$ time, an encoding $S$ of $G$ is weakly convenient [9] if it takes (i) $O(m + n)$ time to encode $G$ and decode $S$, (ii) $O(1)$ time to determine from $S$ the adjacency of any two nodes in $G$, and (iii) $O(d)$ time to determine from $S$ the neighbors of a degree-$d$ node in $G$. If the degree of a node can be determined from a weakly convenient encoding $S$ in $O(1)$ time, then $S$ is convenient [9]. For a planar graph $G$ having multiple edges but no self-loops, Munro and Raman [35] gave the first nontrivial convenient encoding of $G$ with $2m + 8n + o(m + n)$ bits. Their result is based on the four-page decomposition of planar graphs [46] and auxiliary strings, encoding an involved three-level data

---

\(^1\)A closely related rectangle-visibility drawing [13, 12, 25, 3] of $H$ requires that $v_i$ and $v_j$ are adjacent in $H$ if and only if $b_i$ and $b_j$ are visible to each other.

\(^2\)The lower bounds stated in [16] are $x + y \geq \frac{2n}{3}$ and $\min\{x, y\} \geq \frac{2n}{3}$. Based on the given sketch of proof, however, it is not hard to see that their lower bound should be corrected as $x + y \geq n - 1 + \left\lceil \frac{2n+1}{3} \right\rceil$ and $\min\{x, y\} \geq \left\lceil \frac{2n+1}{3} \right\rceil$. 
structure for any string of parentheses. For a planar graph $G$ that has (respectively, has no) multiple edges, Chuang, Garg, He, Kao, and Lu [9] improved the bit count to $2m + (5 + \frac{1}{k})n + o(m + n)$ (respectively, $\frac{3}{7}m + (5 + \frac{1}{k})n + o(n)$) for any positive constant $k$. They also provided a weakly convenient encoding of $2m + \frac{1}{k}n + o(m + n)$ (respectively, $\frac{3}{7}m + 5n + o(n)$) bits for a planar graph $G$ that has (respectively, has no) multiple edges. Based on our orderly pair algorithm, in this paper we present the best known convenient encodings for a planar graph $G$: If $G$ may (respectively, does not) contain multiple edges, then the bit count of our encoding is $2m + 3n + o(m + n)$ (respectively, $2m + 2n + o(n)$), which is even less than that of the weakly convenient encodings of Chuang et al. [9]. The bit counts are very close to Tutte’s information-theoretical lower bound of roughly $3.58m$ bits for encoding connected plane graphs without any query support [44]. The bit count of our encoding for a planar graph $G$ without multiple edges matches that of the best known convenient encoding for an outerplanar graph [35]. Besides relying on the orderly pair algorithm, our results are also based on an improved auxiliary string for a folklore encoding [9, 21, 35] of a rooted tree $T$. With the auxiliary strings of Munro and Raman [35], computing the degree of a degree-$d$ node in $T$ requires $\Theta(d)$ time. In this paper, we present a nontrivial auxiliary string, in Lemma 5.3, to support the degree query in $O(1)$ time.

Recent applications. Besides the applications presented in the present paper, our orderly pair algorithm also yields the following recent results:

- Improved compact distributed routing tables for any $n$-node distributed planar network [33], improving the best previously known design of Gavoille and Hanussse [18] by reducing the worst-case table size count from $8n + o(n)$ bits to $7.181n + o(n)$ bits, without increasing the time complexity of preprocessing and query.
- A linear-time algorithm for compact floor-plans of plane triangulations [30, 31], which is not only much simpler than the previous methods in the literature [20, 47] but also provides the first known nontrivial upper bound on the floor-plan’s area.
- Compact Podevs drawings for plane graphs and an alternative proof for the sufficient and necessary condition for a planar graph to admit a rectangular dual [6].
- Improved upper bounds on the number of planar graphs via so-called well orderly spanning trees, which are orderly spanning trees with additional prop-

Fig. 1.1. (a) A plane graph $H$ with an orderly spanning tree of $H$ rooted at node 1 represented by the thick edges. (b) A 2-visibility drawing of $H$. (c) A realizer $(T_1, T_2, T_{12})$ of $H$, where $T_1$ (respectively, $T_2$ and $T_{12}$) consists of the thick (respectively, dashed and thin) edges.
ORDERLY SPANNING TREES WITH APPLICATIONS

Properties [2].

Organization of the paper. The rest of the paper is organized as follows. Section 2 gives the linear-time algorithm for computing an orderly pair of any given planar graph. Applications are given in sections 3–5. Section 3 gives the linear-time algorithm for computing a realizer of any given plane triangulation. Section 4 shows the linear-time algorithm for obtaining an area-optimal 2-visibility drawing of any given plane graph. Section 5 presents the best known convenient encodings for planar graphs.

2. Orderly spanning trees for plane graphs.

2.1. Basics. Unless stated otherwise, all graphs in sections 2–4 are simple. Let $H$ be a plane graph. The outer boundary of $H$ is the boundary of the external face of $H$. The nodes and edges on the outer boundary of $H$ are external in $H$; and the other nodes and edges are internal in $H$.

Let $T$ be a rooted spanning tree of a connected plane graph $H$. Two distinct nodes of $H$ are unrelated with respect to $T$ if neither of them is an ancestor of the other in $T$. An edge $e$ of $H$ is unrelated with respect to $T$ if the endpoints of $e$ are unrelated with respect to $T$. Let $v_1, v_2, \ldots, v_n$ be the counterclockwise preordering of the nodes in $T$. A node $v_i$ is orderly in $H$ with respect to $T$ if the neighbors of $v_i$ in $H$ form the following four blocks of $H$ with respect to $T$ in counterclockwise order around $v_i$, where each block could be empty.

- $P(v_i)$: the parent of $v_i$ in $T$;
- $U_{<}(v_i)$: the nodes $v_j$ with $j < i$ that are unrelated to $v_i$ with respect to $T$;
- $D(v_i)$: the children of $v_i$ in $T$; and
- $U_{>}(v_i)$: the nodes $v_j$ with $j > i$ that are unrelated to $v_i$ with respect to $T$.

(See Figure 2.1(c).) $T$ is an orderly spanning tree of $H$ if (i) $v_1$ is on the outer boundary of $H$ and (ii) each node $v_i$ is orderly in $H$ with respect to $T$. If $T$ is an orderly spanning tree of $H$, then each incident edge of $v_1$ in $H$ belongs to $T$. An example of an orderly spanning tree is given in Figure 1.1(a). Figure 2.1(a) provides a negative example of an orderly spanning tree, where nodes 1, 3, 8, and 10 are not orderly in $H$ with respect to $T$.

Not every connected plane graph admits an orderly spanning tree. However, as to be shown in this section, there always exists a planar embedding for any given planar graph that admits an orderly spanning tree. For example, consider the plane graph
Assume for a contradiction that \( H \) admits an orderly spanning tree \( T \) rooted at node 1. Observe that the thick edges must be in \( T \), and thus the thin edges cannot be in \( T \). Now, \( T \) contains exactly one of the dashed edges. In either case, however, the parent of node 6 in \( T \) is not orderly in \( H \) with respect to \( T \), thereby contradicting the assumption that \( T \) is an orderly spanning tree rooted at node 1. Since \( H \) is rotationally symmetric, \( H \) admits no orderly spanning trees. If we change the planar embedding of \( H \) by moving edge \((2, 5)\) to the interior of \( H \), as shown in Figure 2.2(b), then the new plane graph has an orderly spanning tree rooted at node 1 consisting of the thick edges.

The concept of orderly spanning trees of plane graphs originates from those of canonical orderings and canonical spanning trees of triconnected plane graphs. Let \( H \) be a triconnected plane graph. A canonical ordering of \( H \) is a certain ordering of the vertices in \( H \), first introduced by de Fraysseix, Pach, and Pollack [11] for plane triangulations, and extended to triconnected plane graphs by Kant [27]. Specifically, let \( v_1, v_2, \ldots, v_n \) be an ordering of the nodes of \( H \). Let \( H_i \) be the subgraph of \( H \) induced by \( v_1, v_2, \ldots, v_i \). Let \( B_i \) be the outer boundary of \( H_i \). This ordering is canonical if the interval \([3, n]\) can be partitioned into \( I_1, \ldots, I_K \) with the following properties for each \( I_j \).

- \( H_{k+q} \) is biconnected. \( B_{k+q} \) contains the edge \((v_1, v_2)\) and \( C_j \). \( C_j \) has no chord in \( H \).
- If \( q = 0 \), \( v_k \) has at least two neighbors in \( H_{k-1} \), all on \( B_{k-1} \). If \( q > 0 \), \( C_j \) has exactly two neighbors in \( H_{k-1} \), both on \( B_{k-1} \), where the left neighbor is incident to \( C_j \) only at \( v_k \) and the right neighbor only at \( v_{k+q} \).
- For each \( v_i \) with \( k \leq i \leq k+q \), if \( i < n \), \( v_i \) has at least one neighbor in \( H - H_{k+q} \).

Chuang et al. [9] defined a canonical spanning tree \( T \) of \( H \) as a way to find parents in \( T \) for all except one node of \( H \) according to any given canonical ordering of \( H \). Specifically, for the given canonical ordering of \( H \), the canonical spanning tree \( T \) of \( H \) rooted at \( v_1 \) is the one formed by the edge \((v_1, v_2)\) together with the paths \( C_j \) and the edges \((v_i, v_k)\) over all \( I_j \), where \( v_i \) is the leftmost neighbor of \( C_j \) on \( B_{k-1} \).

By the triconnectivity of \( H \), it is implicit in [9] that (a) any canonical spanning tree \( T \) of \( H \) has to be an orderly spanning tree of \( H \), and (b) the counterclockwise preordering of \( T \) is the given canonical ordering of \( H \). As shown in Figure 2.1(b), however, the counterclockwise preordering of an orderly spanning tree for a triconnected plane graph \( H \) may not be a canonical ordering of \( H \). Therefore, the concept of orderly spanning trees is more general than that of canonical spanning trees even for triconnected plane graphs. The counterclockwise preordering of any orderly spanning tree of a plane triangulation \( \Delta \) has to be a canonical ordering of \( \Delta \), though.
2.2. The orderly pair algorithm. For plane graphs $H$ and $G$, $H \sim G$ denotes that $H$ and $G$ (ignoring their plane embeddings) are isomorphic planar graphs. We say that $(H, T)$ is an orderly pair of a connected planar graph $G$ with respect to $r$ if (i) $H \sim G$ and (ii) $T$ rooted at $r$ is an orderly spanning tree of $H$. This subsection shows how to compute an orderly pair for any planar graph in linear time. Without loss of generality, we may assume that the input planar graph is already equipped with a planar embedding represented by an adjacency list, where each node $v$ keeps a doubly linked list, storing its neighbors in counterclockwise order around $v$. Moreover, the two copies of each edge are cross-linked to each other. Such a representation can be obtained as a by-product by running the linear-time planarity-testing algorithm of Hopcroft and Tarjan [24] or that of Boyer and Myrvold [4]. Based upon this representation, deleting an edge takes $O(1)$ time. Moreover, moving an edge $e$ to the interior of a face $F$ can be conducted in $O(1)$ time, as long as an edge on the boundary of $F$ incident to each endpoint of $e$ is provided. (Our algorithm moves an edge $e$ to the interior of $F$ only when it traverses the boundary of $F$.)

To describe the algorithm, we need some definitions for a 2-connected plane graph $H$. If $v$ is an external node in $H$, then let next$_H(v)$ (respectively, prev$_H(v)$) denote the external node of $H$ that immediately succeeds (respectively, precedes) $v$ in counterclockwise order around the outer boundary of $H$. For any two distinct external nodes $r$ and $v$ of $H$, let ccw$_H(r, v)$ (respectively, cw$_H(r, v)$) denote the sequence of the external nodes of $H$ from $r$ to $v$ in counterclockwise (respectively, clockwise) order around the outer boundary of $H$. Observe that prev$_H(v) \in$ ccw$_H(r, v)$ and next$_H(v) \in$ cw$_H(r, v)$. Define boundary$_H(r) =$ ccw$_H(r, \text{prev}_H(r))$, i.e., the sequence of the external nodes of $H$ from $r$ to prev$_H(r)$ in counterclockwise order around the outer boundary of $H$. See Figure 2.3 for an illustration. For example, if $H$ is the plane graph shown in Figure 2.2(b), then we have that next$_H(2) =$ 6, prev$_H(2) =$ 1, ccw$_H(1, 2) =$ (1, 2, 6), cw$_H(1, 2) =$ (1, 5, 6), and boundary$_H(1) =$ (1, 2, 6, 5).

The key component of our orderly pair algorithm is the recursive subroutine $\text{block}(G, r, v)$ shown below. Given any 2-connected plane graph $G$ and two distinct external nodes $r$ and $v$ of $G$, the subroutine $\text{block}(G, r, v)$ computes an orderly pair $(H, T)$ of $G$ with respect to $r$. Let us emphasize that $v$ can be any external node of $G$ other than $r$. The high-level strategy of $\text{block}(G, r, v)$ is to identify a neighbor $p$ of $v$ in $G$ that can be the parent of $v$ in $T$. The subroutine then deletes $v$ and its incident edges and recursively works on each 2-connected component of the remaining graph. The difficulty lies in efficiently choosing an appropriate parent of $v$ with possible modification to the plane embedding of $G$. More precisely, we want the subroutine to alter the embedding of $G$ into $H$ and find a neighbor $p$ of $v$ such that (1) the required time is linear in the total size of the internal faces of $G$ that contains $v$ and (2) the
Following property II holds for $H$ and $p$:

For each neighbor $x$ of $v$ in $H$ other than $p$, if $x$ and $\text{prev}(H, v)$ (respectively, $\text{next}(H, v)$) are on the same side of $(v, p)$ in $H$, then $(v, x)$ is on the first (respectively, last) internal face of $H$ containing $v$ and $x$ in counterclockwise order around $v$ starting from the one containing $(v, \text{next}(H, v))$.

(See Figure 2.5(b) for an illustration of property II.) It turns out that a two-phase process serves the purpose: We say that an edge of $G$ is movable if the embedding of $G$ can be changed by moving the edge into a face of $G$. For example, edges $(1, 2)$, $(2, 5)$, and $(5, 1)$ are the movable edges in the plane graph shown in Figure 2.2(a). Imagine that node $v$ is at the “bottom” of $G$. The first phase flips each movable incident edge of $v$ to the leftmost possible face. At the end of the first phase, the node $p$ is exactly the neighbor of $v$ on $\text{cw}(G, r, v)$ that is closest to $r$. After determining $p$, the second phase flips each movable incident edge of $v$ that is to the left of edge $(p, v)$ to the rightmost possible face.

When recursively taking care of each 2-connected component $G_i$ of $G'$, the subroutine has to choose two distinct nodes $r_i$ and $v_i$ on the outer boundary of $G_i$ for the recursive subroutine call $\text{block}(G_i, r_i, v_i)$. For any choice of $r_i$ and $v_i$, the subroutine call will return an orderly pair $(H_i, T_i)$ of $G_i$ with respect to $r_i$. However, to ensure that gluing all returned orderly pairs together yields an orderly pair of $G$, we have to be careful about the choice of each $v_i$.

The detailed description of $\text{block}(G, r, v)$ is as follows.

**Subroutine block($G, r, v$).**

**Step 1.** If $G$ consists of a single edge $(r, v)$, then return $(G, G)$; otherwise, perform steps 2–7.

**Step 2.** Perform step 2.1 for each internal face $F$ incident to node $v$ in $G$ in clockwise order around $v$ starting from the one containing $(v, \text{prev}(G, v))$.

**Step 2.1.** For any node $x$ in $F$ such that $(v, x)$ is an edge of $G$ succeeding $F$ in clockwise order around $v$ starting from $(v, \text{next}(G, v))$, update the planar embedding of $G$ by flipping $(v, x)$ into the interior of $F$.

Remark. For instance, if $v$ and $F$ are as shown in Figure 2.4, then $(v, x_1)$ and $(v, x_2)$ will be flipped into the interior of $F$ by step 2.1.

**Step 3.** Let $p$ be the neighbor of $v$ in $G$ closest to $r$ in $\text{cw}(G, r, v)$.

**Step 4.** Perform step 4.1 for each internal face $F$ of $G$ that succeeds $(v, p)$ in counterclockwise order around $v$ starting from the face containing $(v, p)$:

**Step 4.1.** For any node $x$ in $F$ such that $(v, x)$ is an edge of $G$ succeeding $F$ in counterclockwise order around $v$ starting from $(v, \text{next}(G, v))$, update the planar embedding of $G$ by flipping $(v, x)$ into the interior of $F$.

Remark. For instance, if $v$ and $F$ are as shown in Figure 2.4, then $(v, x_3)$ and $(v, x_4)$ will be flipped into the interior of $F$ by step 4.1.

**Step 5.** Let $G'$ be the graph obtained by deleting all the incident edges of $v$ in $G$, except for $(v, p)$. Compute the 2-connected components of $G'$ by traversing the segment of the outer boundary of $G'$ from $\text{prev}(G, v)$ to $\text{next}(G, v)$ in counterclockwise order around the outer boundary of $G'$.

Remark. Since $G$ is 2-connected, we know that all 2-connected components of $G'$ are external to one another. Therefore, the above traversal of part of the outer boundary will suffice. Also, by the definitions of $G'$ and $p$, one of the 2-connected components of $G'$ consists of the single edge $(v, p)$. 


Step 6. Compute \((H_i, T_i) = \text{block}(G_i, r_i, v_i)\) for each 2-connected component \(G_i\) of \(G\), where \(r_i\) is the node of \(G_i\) closest to \(r\) in \(G\), and \(v_i\) is defined as follows:

Case 1. \(G_i = (v, p)\). Let \(v_i = v\).

Case 2. \(G_i\) and \(\text{prev}(G, v)\) are on the same side of \((v, p)\) in \(G\). Let \(S\) consist of the nodes in both \(\text{ccw}(G_i, \text{next}(G_i, r_i), \text{prev}(G_i, r_i))\) and \(\text{ccw}(G, r, v)\). If \(S\) is empty, then let \(v_i = \text{next}(G_i, r_i)\). Otherwise, let \(v_i\) be the last node of \(S\) in counterclockwise order around the outer boundary of \(G_i\).

Case 3. \(G_i\) and \(\text{next}(G, v)\) are on the same side of \((v, p)\) in \(G\). Let \(S\) consist of the nodes in both \(\text{ccw}(G_i, \text{next}(G_i, r_i), \text{prev}(G_i, r_i))\) and \(\text{ccw}(G, r, v)\). If \(S\) is empty, then let \(v_i = \text{prev}(G_i, r_i)\). Otherwise, let \(v_i\) be the first node of \(S\) in counterclockwise order around the outer boundary of \(G_i\).

Step 7. Return \((H, T)\), where \(H\) is obtained from \(G\) by replacing each \(G_i\) with \(H_i\) and \(T\) is the union of all \(T_i\).

An illustration of \(\text{block}(G, r, v)\) is given in Figure 2.5. Let \(G\) be the 2-connected plane graph shown in Figure 2.5(a). At the completion of step 4, the resulting embedding of \(G\) and \(p\) are as shown in Figure 2.5(b), where the gray ellipse with label \(i\) is the \(i\)th 2-connected component \(G_i\) of \(G\). Note that \((v, p)\) is also a 2-connected component of \(G\). One can verify that after step 6 we have that \(r_1 = r, r_2 = r_6, r_8 = r_9, r_{11} = r_{12} = p, \) and \(v_{11} = v\). For the 2-connected components lying on the same side of \((v, p)\) with \(\text{prev}(G, v)\), we have \(v_1 = r_2, v_2 = r_3, v_3 = r_4, v_4 = \text{prev}(G, v)\), and that \(v_i = \text{next}(G_i, r_i)\) holds for each \(i \in \{5, 6, \ldots, 10\}\). For the 2-connected components lying on the same side of \((v, p)\) with \(\text{next}(G, v)\), we have \(v_{12} = r_{13}, v_{13} = r_{15}, v_{14} = \text{prev}(G_{14}, r_{14}), \) and \(v_{15} = \text{next}(G, v)\).

**Lemma 2.1.** If \(r\) and \(v\) are two distinct external nodes of a 2-connected plane graph \(G\), then \(\text{block}(G, r, v)\) outputs an orderly pair of \(G\) with respect to \(r\).

**Proof.** Let \((H, T)\) be the output of \(\text{block}(G, r, v)\). We prove the following properties of \((G, H, T, r, v)\) by induction on the number of edges in \(G\):

1. Each external node of \(G\) remains external in \(H\). Moreover, \(\text{boundary}(G, r)\) is a subsequence of \(\text{boundary}(H, r)\).

2. (Property Π.) For each neighbor \(x\) of \(v\) in \(H\) other than \(p\), if \(x\) and \(\text{prev}(H, v)\) (respectively, \(\text{next}(H, v)\)) are on the same side of \((v, p)\) in \(H\), then \((v, x)\) is on the first (respectively, last) internal face of \(H\) containing \(v\) and \(x\) in counterclockwise order around \(v\) starting from the one containing \((v, \text{next}(H, v))\).

3. \(T\) rooted at \(r\) is a spanning tree of \(H\) such that exactly one of the following
Fig. 2.5. (a) A 2-connected plane graph $G$, where each gray ellipse is a 2-connected component of $G - \{v\}$. (b) The plane graph $G$ at the completion of performing steps 1–4 of block$(G, r, v)$. Observe that all edges incident to $v$, especially those dashed (i.e., movable) edges, satisfy property $\Pi$.

conditions holds for each node $u$ in ccw$(H, r, v)$ (respectively, cw$(H, r, v)$):
(i) $u$ is a leaf of $T$; or (ii) next$(H, u)$ (respectively, prev$(H, u)$) is the lowest-indexed (respectively, highest-indexed) child of $u$ in $T$.

4. $H \sim G$.
5. $T$ rooted at $r$ is an orderly spanning tree of $H$.

Properties 4 and 5 suffice, but we need the other properties to enable the induction step. When $G$ consists of a single edge $(r, v)$, by step 1 we have $H = T = G$. It is not difficult to see the inductive basis of each property holds. Suppose that $G'$ consists of $k$ 2-connected components. By step 6, we have $r_i \neq v_i$ for each $i$. It follows from the inductive hypothesis that properties 1–5 of $(G_i, H_i, T_i, r_i, v_i)$ hold for each $i \in \{1, 2, \ldots, k\}$. The rest of the proof shows the induction step. For brevity, for each $j = 1, 2, \ldots, 5$, we abbreviate “property $j$ of $(G_i, H_i, T_i, r_i, v_i)$” to “property $j$ of $G_i$” and use “property $j$ (of $G$)” to stand for “property $j$ of $(G, H, T, r, v)$.”

Property 1. Observe that throughout the execution of block$(G, r, v)$, without accounting for its subsequent subroutine calls to block, the embedding of $G$ changes only by flipping edges into the interior of internal faces of $G$ in steps 2 and 4. Thus, based on how $H$ is obtained from $G$ in step 7, it follows from property 1 of $G_i$ for each $i \in \{1, 2, \ldots, k\}$ that the property holds.

Property 2. Let property 2’ stand for the property obtained from property 2 by replacing each $H$ with a $G$. From steps 2 and 4, one can verify that the plane graph $G$ at the completion of performing step 4 satisfies property 2’. From property 2 and how $H$ is obtained from $G$ in step 7, we know that the relative order among the incident edges of $v$ and the faces containing $v$ remains the same in $G$ and $H$. Therefore the property follows from property 2’.

Property 3. For each $i \in \{1, 2, \ldots, k\}$, property 3 of $G_i$ implies that $T_i$ is a spanning tree of $H_i$. Since $H_1, H_2, \ldots, H_k$ are edge disjoint, and each node of $H$ belongs to some $H_i$, we know that $T$, the union of all $T_i$, is a spanning tree of $H$. Since $v$ is a leaf of $T$, the required property holds for $v$. Let $x$ be an external node of $H$ other than $v$. If $(x, v)$ is not an external edge of $H$ belonging to $H - T$, then the required property for $x$ follows from the property of $x$ guaranteed by property 3 of
for each index \(i\) with \(x \in H_i\). Otherwise, by property 2, \(x\) is either \(\text{prev}(H, v)\) or \(\text{next}(H, v)\). Let \(H_j\) be the 2-connected component of \(H'\) containing \(x\). We have that \(v_j = x\). By property 3 of \(G_j\), \(x\) is a leaf of \(T_j\). Since \((x, v) \not\in T\), \(x\) is also a leaf of \(T\) and property 3 holds for \(x\).

**Property 4.** Observe that steps 2 and 4 flip an edge \((v, x)\) into the interior of \(F\) only if \(F\) contains both \(v\) and \(x\). Therefore, the resulting embedding of \(G\) at the completion of step 4 is still planar. According to how \(H\) is obtained from \(G\) in step 7, the property follows from property 1 of \(G\) and properties 4 of \(G_i\) for all indices \(i \in \{1, 2, \ldots, k\}\).

**Property 5.** Each neighbor of \(r\) in \(H\) is a child of \(r\) in \(T\); hence \(r\) is orderly in \(H\) with respect to \(T\). The rest of the proof shows that each node \(x\) other than \(r\) is orderly in \(H\) with respect to \(T\). Let \(H'\) be the graph obtained from \(H\) by deleting each incident edge of \(v\) in \(H - T\). Observe that \(H' \sim G'\) and that each \(H_i\) is a 2-connected component of \(H'\). Let \(I_x\) consist of the indices \(i\) with \(x \in H_i\). As \(x \neq r\), one can verify that there is an index \(j\) in \(I_x\) such that \(x \neq r_j\) and \(x = r_{j_i}\) for each index \(i \in I_x - \{j\}\).

We first show that if \((v, x)\) is an edge of \(H - H'\), then \((v, x)\) is unrelated with respect to \(T\). If the index of \(x\) is higher than that of \(v\), then by the fact that \(v\) is a leaf in \(H\), we know that \((v, x)\) is unrelated. As for the case with the index of \(x\) lower than that of \(v\), let us assume for a contradiction that \(x\) is an ancestor of \(v\) in \(T\). Since \((v, x) \in H - T\) and \(p\) is the parent of \(v\) in \(T\), we know that \(x\) is also an ancestor of \(p\) in \(T\). Let \(P\) be the path of \(T\) between \(r\) and \(p\). Thus, \(x \in P\). Let \(y\) be the node of \(H_j\) closest to \(p\) in \(P\). It is not difficult to see that \(y \in \text{cw}(H, r, p)\) and \(y \in \text{cw}(H_j, r_j, x)\). Since \((v, p) \in T\), \((v, x) \in H - T\), and \(y \in \text{cw}(H, r, p)\), we know \(y \neq x\). Otherwise, \(x\) would have been a neighbor of \(v\) in \(H\) closer to \(r\) than \(p\) in \(\text{cw}(H, r, v)\), thereby contradicting the choice of \(p\) in step 3. As \(y \neq x\), \(x\) is not a leaf of \(T_j\). By step 6(2), \(x \in \text{cw}(H_j, r_j, v_j)\). Let \(z = \text{prev}(H_j, x)\). By property 3 of \(G_j\), node \(z\) has to be the largest-indexed child of \(x\) in \(T_j\). Since \(x \neq r_j\) and \(y \in \text{cw}(H_j, r_j, x)\), we know that \(y\) and \(z\) are on different sides of the path of \(T_j\) between \(r_j\) and \(x\) in \(H_j\), thereby contradicting the fact that \(z\) is the highest-indexed child of \(x\) in \(T_j\).

We then show that \(x\) is orderly in \(H'\) with respect to \(T\). If \(|I_x| = 1\), then the orderly pattern of \(x\) in \(H'\) with respect to \(T\) follows immediately from that in \(H_j\) with respect to \(T_j\), which is ensured by property 5 of \(G_j\). When \(|I_x| \geq 2\), by properties 5 of \(G_i\) for all \(i \in I_x - \{j\}\), each neighbor of \(x\) in \(\bigcup_{i \in I_x - \{j\}} H_i\) is a child of \(x\) in \(T\). It follows from property 3 of \(G_i\) that all children of \(x\) in \(T\) are consecutive in \(H'\) around \(x\). Since \(x\) is orderly in \(H_j\) with respect to \(T_j\), one can see that \(x\) is orderly in \(H'\) with respect to \(T\).

Since \(v\) is a leaf of \(T\), we know that \(v\) is orderly in \(H\) with respect to \(T\). It remains to show that each neighbor \(x\) of \(v\) in \(H - H'\) is orderly in \(H\) with respect to \(T\). Let \(z_1\) (respectively, \(z_2\)) be the neighbor of \(x\) that precedes (respectively, succeeds) \(v\) in counterclockwise order around \(x\). It suffices to show that if the index of \(x\) is lower (respectively, higher) than that of \(v\), then \(z_2\) (respectively, \(z_1\)) belongs to \(P(x)\) or \(D(x)\) (respectively, \(P(x)\) or \(U(x)\)) of \(H'\) with respect to \(T\) as follows: If the index of \(x\) is lower than that of \(v\), then \(z_2 = \text{next}(H_j, x)\) by property 2. By step 6, one can verify that \(x\) belongs to \(\text{cw}(H_j, r_j, v_j)\). By property 3, we have that \(z_2\) belongs to either \(P(x)\) or \(D(x)\) of \(H'\) with respect to \(T\). If the index of \(x\) is higher than that of \(v\), then we know \(z_1 = \text{prev}(H_j, x)\) from property 2. By step 6, one can verify that \(x\) belongs to \(\text{cw}(H_j, r_j, v_j)\). From property 3, we have that \(z_1\) belongs to either \(P(x)\) or \(U(x)\) of \(H'\) with respect to \(T\).

**Lemma 2.2.** If \(r\) and \(v\) are two distinct external nodes of an \(n\)-node 2-connected
plane graph \( G \), then \( \text{block}(G, r, v) \) runs in \( O(n) \) time.

Proof. The execution of \( \text{block}(G, r, v) \) consists of a sequence of subroutine calls to block. One can see that each node of \( G \) can be the parameter \( v \) for no more than two subroutine calls to block—one with \( G \neq (r, v) \) and the other with \( G = (r, v) \). If \( G = (r, v) \), then the subroutine call \( \text{block}(G, r, v) \) runs in \( O(1) \) time. Let \( \ell \) be the number of subroutine calls to \( \text{block}(G, r, v) \) with \( G \neq (r, v) \). For each \( j \in \{1, 2, \ldots, \ell\} \), let \( \text{block}(G_j, r^j, v^j) \) be the \( j \)th subroutine call to block with \( G^j \neq (r^j, v^j) \) throughout the execution of \( \text{block}(G, r, v) \), where \( G^j = G, r^1 = r \) and \( v^1 = v \). Clearly, \( v^j \neq v^{j'} \) holds for any two distinct indices \( j \) and \( j' \), thereby implying that \( \ell \leq n \). Let \( E_j \) consist of the edges of \( G \) belonging to the boundaries of the internal faces of \( G^j \) that contain \( v^j \). Let \( t_j \) be the time required by \( \text{block}(G^j, r^j, v^j) \), without accounting for that required by its subsequent subroutine calls to block. Observe that \( t_j = O(|E_j|) \) holds for each \( j \). It is not difficult to implement the algorithm block such that the running time of \( \text{block}(G, r, v) \) is dominated by \( \sum_{j=1}^{\ell} t_j = \sum_{j=1}^{\ell} O(|E_j|) \). Since \( G \) has \( O(n) \) edges, it suffices to show as follows that any edge \((x, y)\) of \( G \) belongs to no more than two of the sets \( E_1, E_2, \ldots, E_{\ell} \): Let \( j_1 \) be the smallest index \( j \) with \( (x, y) \in E_j \). If \( v^{j_1} \in \{x, y\} \), then \( j_1 \) is also the largest index \( j \) with \( (x, y) \in E_j \). It remains to consider the case \( v^{j_1} \notin \{x, y\} \). Let \( j_2 \) be the smallest index \( j \) with \( j > j_1 \) and \( (x, y) \in E_j \). From the definition of block, edge \((x, y)\) has to be on the outer boundary of \( G^{j_2} \), implying \( v^{j_2} \in \{x, y\} \). Therefore, \( j_2 \) is the largest index \( j \) with \( (x, y) \in E_j \). \( \square \)

Finally, we have the next theorem.

**Theorem 2.3.** It takes \( O(n) \) time to compute an orderly pair for an \( n \)-node connected planar graph.

3. Realizers for plane triangulations. This section provides a new linear-time algorithm for computing a realizer for any \( n \)-node plane triangulation \( \Delta \). As defined by Schnyder [39, 38], \((T_1, T_2, T_n)\) is a realizer of \( \Delta \) if

- the internal edges of \( \Delta \) are partitioned into three edge-disjoint trees \( T_1, T_2, \) and \( T_n \), each rooted at a distinct external node of \( \Delta \); and
- the neighbors of each internal node \( v \) of \( \Delta \) form six blocks \( U_1, D_n, U_2, D_1, U_n, \) and \( D_2 \) in counterclockwise order around \( v \), where for each \( j \in \{1, 2, n\} \), \( U_j \) (respectively, \( D_j \)) consists of the parent (respectively, children) of \( v \) in \( T_j \).

A realizer of the plane triangulation in Figure 1.1(a) is shown in Figure 1.1(c).

**Lemma 3.1.** Given an orderly spanning tree of \( \Delta \), a realizer of \( \Delta \) is computable in \( O(n) \) time.

Proof. Let \( T \) be the given orderly spanning tree of \( \Delta \) rooted at \( v_1 \). Let \( v_1, \ldots, v_n \) be the counterclockwise preordering of \( T \), where \( v_1 \) and \( v_n \) are the external nodes of \( \Delta \) in counterclockwise order. Note that \((v_1, v_2)\) and \((v_1, v_n)\) must be in \( T \). Since \( \Delta \) is a plane triangulation and the edges of \( \Delta - T \) are unrelated with respect to \( T \), both \( U_<(v_i) \) and \( U_>(v_i) \) are nonempty for each \( 3 \leq i \leq n - 1 \). (To see this, one can verify that if \( U_<(v_i) \) or \( U_>(v_i) \) were empty, then the edge between \( v_i \) and the parent of \( v_i \) in \( T \) would belong to a face of \( \Delta \) consisting of at least four edges, contradicting the assumption that \( \Delta \) is a plane triangulation.) Let \( p_i \) (respectively, \( q_i \)) be the index of the last (respectively, first) node in \( U_<(v_i) \) (respectively, \( U_>(v_i) \)) in counterclockwise order around \( v_i \). Let \( T_1 \) be obtained from \( T \) by deleting \((v_1, v_2)\) and \((v_1, v_n)\). Let \( T_2 = \{(v_i, v_{p_i}) \mid 3 \leq i \leq n - 1 \} \) and \( T_n = \{(v_i, v_{q_i}) \mid 3 \leq i \leq n - 1 \} \). An example is shown in Figure 1.1(c). Observe that \( p_i < i < q_i \) holds for each \( 3 \leq i \leq n - 1 \), implying that both \( T_2 \) and \( T_n \) are acyclic. It can be proved as follows that exactly one of the equalities \( i = p_j \) and \( j = q_i \) holds for each edge \((v_i, v_j) \in \Delta - T \) with \( i < j \).
Since each face of $\Delta$ has size three, there is a node $v_k$ that is (i) the neighbor of $v_i$ immediately succeeding $v_j$ in clockwise order around $v_i$ and (ii) the neighbor of $v_j$ immediately preceding $v_i$ in clockwise order around $v_j$. Clearly, $i < j < k$ implies $i = p_j$ and $j \neq q_i$; and $k < i < j$ implies $j = q_i$ and $i \neq p_i$. As for the remaining case that $i < j < k$, one can verify that $v_i$ has to be the parent of $v_k$ in $T$, thereby implying $j = q_i$ and $i \neq p_j$.

It follows that each internal edge of $\Delta$ belongs to exactly one of $T_1$, $T_2$, and $T_n$. By the definitions of $p_i$ and $q_i$, one can verify that the neighbors of each internal node $v_i$ of $\Delta$ indeed form the required pattern for $(T_1, T_2, T_n)$ as a realizer of $\Delta$. Since it takes $O(n)$ time to determine all $p_i$ and $q_i$, the lemma is proved.

**Theorem 3.2** (see also [39, 38]). A realizer of any plane triangulation is derivable in linear time.

*Proof.* The proof is straightforward by Theorem 2.3 and Lemma 3.1. \qed

### 4. 2-visibility drawings for plane graphs

This section shows how to obtain in $O(n)$ time an $(n-1) \times \left\lceil \frac{2n+1}{3} \right\rceil$ 2-visibility drawing for any $n$-node plane graph $G$. For calculating the area of a 2-visibility drawing, we follow the convention of [16], stating that the corner coordinates of each rectangle are integers, and that each rectangle is no smaller than $1 \times 1$. For example, the area of the 2-visibility drawing shown in Figure 1.1(b) is $9 \times 8$. Let $\Delta$ be a plane triangulation obtained by triangulating $G$. Since any 2-visibility drawing of $\Delta$ is also a 2-visibility drawing of $G$, the rest of the section assumes that the input is the plane triangulation $\Delta$.

Let $T$ be an orderly spanning tree of $\Delta$. Let $v_1, v_2, \ldots, v_n$ be the counterclockwise preordering of the nodes in $T$. Our algorithm $\text{draw}(\Delta, T)$ consists of $n$ iterations, where the $i$th iteration performs the following steps:

**Step 1.** If $i \neq 1$ and $v_i$ is not the first child of its parent in $T$, then lengthen each ancestor of $v_i$ in $T$ to the right by one unit.

**Step 2.** Draw $v_i$ as a unit square beneath the parent of $v_i$ in $T$ such that $v_i$ and all ancestors of $v_i$ in $T$ align along the right boundary. Now, $v_i$ is vertically visible to its parent in $T$.

**Step 3.** Lengthen downward $v_i$ and each neighbor $v_j$ of $v_i$ in $\Delta$ with $j < i$, if necessary, so that $v_i$ and $v_j$ are horizontally visible to each other.

If $\Delta$ and $T$ are as shown in Figure 4.1(a), then the intermediate (respectively, resulting) drawing obtained by $\text{draw}(\Delta, T)$ is shown in Figure 4.1 (respectively, Figure 4.1(b)).

**Lemma 4.1.** The algorithm $\text{draw}(\Delta, T)$ obtains an $h \times w$ 2-visibility drawing of $\Delta$ with $h \leq n - 1$ and such that $w$ equals the number of leaves in $T$.

*Proof.* Since $T$ is a spanning tree of $\Delta$, $\text{draw}(\Delta, T)$ is well defined. Also, the output of $\text{draw}(\Delta, T)$ is indeed a 2-visibility drawing of $\Delta$ with width equal to the number of leaves in $T$. The rest of the proof shows that the height of the output of $\text{draw}(\Delta, T)$ is at most $n - 1$. For any two distinct edges $e$ and $e'$ in $\Delta - T$, we say that $e$ encloses $e'$ if $e'$ is in the interior of the cycle consisting of $e$ and the path of $T$ between the endpoints of $e$. By the planarity of $\Delta$, the above “enclosing” relation defines a nesting structure among the edges of $\Delta - T$. Let $\ell(e)$ denote the “level” of edge $e$ in the nesting structure:

If $e$ does not enclose any other edge in $\Delta - T$, then let $\ell(e) = 1$;
otherwise, let $\ell(e)$ be one plus the maximum of $\ell(e')$ over all the edges $e'$ in $\Delta - T$ that are enclosed by $e$.

If we are to draw edge $e = (u, v)$ as a horizontal line segment connecting the rectangles
representing nodes $u$ and $v$ without intersecting other rectangles or line segments, then $1 + \ell(e)$ is the minimum possible (vertical) distance between the line segment representing $e$ and the rectangle representing the lowest common ancestor of $u$ and $v$ in $T$. Let edge $\hat{e}$ be $(v_2, v_n)$, which encloses all the other edges in $\Delta - T$. The height of the output of $\text{draw}(\Delta, T)$ is exactly $1 + \ell(\hat{e})$. It remains to show $\ell(\hat{e}) \leq n - 2$ as follows: Assume for the sake of contradiction that $e_1, e_2, \ldots, e_{n-1}$ is a sequence of edges in $\Delta - T$ such that $e_i$ encloses $e_1, e_2, \ldots, e_{i-1}$ for each $i \in \{2, 3, \ldots, n - 1\}$. For each $i \in \{1, 2, \ldots, n - 1\}$, let $X_i$ consist of the endpoints of $e_i, e_{i+1}, \ldots, e_{n-1}$. For each $i \in \{1, 2, \ldots, n - 2\}$, there must be an endpoint of $e_i$ that is not in $X_{i+1}$. (To see this, assume for the sake of contradiction that $j$ and $k$ with $i < j < k \leq n - 1$ are two indices such that $e_j$ and $e_k$ are incident to the endpoints $u$ and $v$ of $e_i$, respectively. Let $C$ be the cycle consisting of $e_j$ and the path of $T$ that connects the endpoints of $e_j$. Since $e_i$ is in the interior of $C$ and $e_k$ is outside of $C$, the assumption that $e_k$ is incident to $v$ violates the orderly property of $v$.) Therefore, $X_1$ contains at least $n$ distinct nodes. Since $T$ is an orderly spanning tree of $\Delta$, $v_1$ is not incident to any edges of $\Delta - T$. Therefore, $v_1 \notin X_1$, which contradicts that $\Delta$ has $n$ nodes.

**Lemma 4.2.** It takes $O(n)$ time to compute an orderly spanning tree of $\Delta$ with


ORDERLY SPANNING TREES WITH APPLICATIONS

937

\[ \frac{2n+1}{2} \text{ or fewer leaves.} \]

Proof. Let \( v_1, v_2, \) and \( v_n \) be the external nodes of \( \Delta \) in counterclockwise order around the outer boundary of \( \Delta \). By Theorem 3.2, a realizer \( (T'_1, T'_2, T'_n) \) of \( \Delta \), where each \( T'_i \) is rooted at \( v_i \), can be obtained in \( O(n) \) time. Let \( I = \{1, 2, n\} \). For each \( i \in I \), let \( T_i = T'_i \cup \{(v_i, v_{i1}), (v_i, v_{i2})\} \), where \( \{i_1, i_2\} = I - \{i\} \). Observe that \( T_1, T_2, \) and \( T_n \) are three spanning trees of \( \Delta \) with \( T_1 \cup T_2 \cup T_n = \Delta \). We first show that each \( T_i \) is an orderly spanning tree of \( \Delta \). Since the relation between \( T_1, T_2, \) and \( T_n \) is rotationally symmetric, it suffices to verify that each node is orderly with respect to \( T_1 \). Let \( v_1, v_2, \ldots, v_n \) be the counterclockwise preordering of \( T_1 \). For each \( i \in I \) and \( j \in \{1, 2, \ldots, n\} \), let \( Q_{i,j} \) be the path of \( T_i \) between \( v_i \) and \( v_j \). Note that \( Q_{1,j}, Q_{2,j}, \) and \( Q_{n,j} \) are three edge-disjoint paths of \( \Delta \) that intersect only at \( v_j \). If \( v_j \) is not a leaf of \( T_1 \), then the children of \( v_j \) in \( T_1 \) are consecutive in \( \Delta \) in counterclockwise order around \( v_j \). Therefore, to ensure that each node is orderly with respect to \( T_1 \), it suffices to prove that each edge of \( \Delta - T_1 \) is unrelated with respect to \( T_1 \): If \( v_j \) were an ancestor of \( v_j \), that is, also a neighbor of \( v_j \) in \( \Delta - T_1 \), then \( v_j \) and \( v_j \) would be on different sides of \( Q_{2,j'}, Q_{3,j''}, \) and \( Q_{n,j''} \) in \( \Delta \), where \( v_j' \) is the parent of \( v_j \) in \( T_1 \), thereby contradicting the planarity of \( \Delta \).

It remains to show that \( T_1, T_2, \) or \( T_n \) has at most \( \frac{2n+1}{2} \) leaves. For each \( i \in I \), let \( \text{leaf}(T'_i) \) consist of the leaves of \( T'_i \). Since the number of leaves in \( T_i \) is precisely \( 2 + |\text{leaf}(T'_i)| \), it suffices to show that \( \sum_{i \in I} |\text{leaf}(T'_i)| \leq 2n - 5 \). Let \( v \) be a node in \( \text{leaf}(T_i) \). Observe that \( v \) is internal in \( \Delta \). For each \( i \in I \), let \( p_i(v) \) denote the parent of \( v \) in \( T_i \). Let \( i_1 \) and \( i_2 \) be the indices in \( I - \{i\} \). Since \( (T'_1, T'_2, T'_n) \) is a realizer of \( \Delta \), there is a unique internal face \( F_i(v) \) of \( \Delta \) containing \( v, p_{i_1}(v), \) and \( p_{i_2}(v) \). We have that \( p_{i_1}(v) \notin \text{leaf}(T'_i) \) and \( p_{i_2}(v) \notin \text{leaf}(T'_{i_2}) \). It follows that \( F_i(v) \neq F_{i_1}(u_1) \) for any node \( u_1 \) in \( \text{leaf}(T'_i) \) and that \( F_i(v) \neq F_{i_2}(u_2) \) for any node \( u_2 \) in \( \text{leaf}(T'_i) \). Therefore, \( \sum_{i \in I} |\text{leaf}(T'_i)| \leq \) no more than the number of internal faces of \( \Delta \), which is precisely \( 2n - 5 \) by Euler's formula.

Theorem 4.3. An \( (n - 1) \times \left\lfloor \frac{2n+1}{2} \right\rfloor \) 2-visibility drawing of any \( n \)-node planar graph is computable in \( O(n) \) time.

Proof. A naive implementation of algorithm \textit{draw} runs in \( O(n^2) \) time, since one stretching of a node's rectangle affects the horizontal spans of all its descendants. However, it is not difficult to implement the algorithm to run in \( O(n) \) time. The width of the rectangle representing a node \( v \) is exactly the number of leaves in the subtree of \( T \) rooted at \( v \). Therefore, one can easily calculate the \( x \)-coordinates of all rectangles in linear time. Also, the \( y \)-coordinates of all rectangles can be computed in linear time via the values \( \ell(e) \) for all edges \( e \) in \( G - T \). (See, e.g., Figure 3.2 of [32] for related implementation techniques.) Therefore, the theorem follows from Lemmas 4.1 and 4.2.

5. Convenient encodings for planar graphs. This section gives the best known convenient encoding for an \( n \)-node \( m \)-edge planar graph as an application of our orderly pair algorithm. We need some notation to describe the data structures required by our convenient encodings. Let \( |S| \) denote the length of a string \( S \), i.e., the number of symbols in \( S \). Unless clearly stated otherwise, all strings in this section have length \( O(m + n) \). A string \( S \) consisting of \( t \) distinct symbols can be encoded in \( |S| \log_2 t \) bits. For example, if \( S \) consists of parentheses and brackets, including open and closed ones, then \( S \) can be encoded in \( 2|S| \) bits. \( S \) is \textit{binary} if it consists of two distinct symbols. For each \( 1 \leq i < j \leq |S| \), let \( S[i, j] \) be the length-\((j - i + 1)\) substring of \( S \) from the \( i \)th position to the \( j \)th position. If \( i > j \), then let \( S[i, j] \) be the empty string. Define \( S[i] = S[i, i] \). \( S[k] \) is \textit{enclosed} by \( S[i] \) and \( S[j] \) in \( S \) if \( i < k < j \).
Let select($S, i, \Box$) be the position of the $i$th $\Box$ in $S$. Let rank($S, k, \Box$) be the number of $\Box$’s before or at the $k$th position of $S$. If $k = select(S, i, \Box)$, then $i = rank(S, k, \Box)$.

An auxiliary string $\chi$ of $S$ is a binary string with $|\chi| = o(|S|)$ which is obtainable from $S$ in $O(|S|)$ time.

**Fact 5.1** (see [1, 15]). For any strings $S_1, S_2, \ldots, S_k$ with $k = O(1)$, there is an auxiliary string $\chi_0$ such that, given the concatenation of $\chi_0, S_1, S_2, \ldots, S_k$ as input, the index of the first symbol of any given $S_i$ in the concatenation is computable in $O(1)$ time.

Let $S_1 \circ S_2 \circ \cdots \circ S_k$ denote the concatenation of $\chi_0, S_1, S_2, \ldots, S_k$ as in Fact 5.1.

Suppose that $S$ is a string of multiple types of parentheses. Let reverse($S$) be the string $R$ such that $R[i]$ is the opposite parenthesis of the same type as $S[|S|+1-i]$. For example, reverse(“( ) ( )”) = “) ( (“). For an open parenthesis $S[i]$ and a closed one $S[j]$ of the same type with $i < j$, the two match in $S$ if every parenthesis of the same type that is enclosed by them matches one enclosed by them. $S$ is balanced in type $k$ if every parenthesis of type $k$ in $S$ belongs to a matching parenthesis pair. $S$ is balanced if $S$ is empty or is balanced in all types of parentheses. Here are some queries defined for a balanced string $S$. Let match($S, i$) be the position of the parenthesis in $S$ that matches $S[i]$. Let enclose$_k(S, i, j)$ be the position pair $(j_1, j_2)$ of the closest matching parenthesis pair of the $k$th type that encloses $S[i_1]$ and $S[i_2]$.

**Fact 5.2** (see [35, 9]). For any balanced string $S$ of $O(1)$ types of parentheses, there is an auxiliary string $\chi_1(S)$ such that each of rank$S, i, \Box$, select($S, i, \Box$), match($S, i$), and enclose$_k(S, i, j)$ can be determined from $S \circ \chi_1(S)$ in $O(1)$ time.

For a string $S$ of parentheses that may be unbalanced, we define wrapped($S, i$) as follows. For the case that $S[i]$ is an open parenthesis of type $k$, let $S'$ be a string obtained from $S$ by appending some closed parentheses of type $k$, if necessary, such that $S'[i]$ is matched in $S'$. Define wrapped($S, i$) to be the number of indices $j$ satisfying $i < j \leq |S|$, enclose$_k(S', j, \text{match}(S', j))) = (i, \text{match}(S', i))$, and $S'[j]$ is of type $k$. For the case that $S[i]$ is closed, let wrapped($S, i$) = wrapped(reverse($S$), $|S| + 1 - i$).

Therefore, if

\[(1) \quad S = (\left[ (\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{\ldots}}}}}}}}}}}}\right], \ldots, 3.4.4.5.5.5.6.6.7.8.9.9.A.\ldots A.B.B.B.8.7.C.\ldots.C1)\]

then we have wrapped($S, 1$) = 10 and wrapped($S, 6$) = 4. If $S$ is balanced, then wrapped($S, i$) is an even number for each $i$, i.e., twice the number of parenthesis pairs that are enclosed by the parenthesis pair in question. The next lemma extends the set of queries supported in Fact 5.2.

**Lemma 5.3.** For any balanced string $S$ of $O(1)$ types of parentheses, there is an auxiliary string $\chi_2(S)$ such that wrapped($S, i$) can be computed from $S \circ \chi_2(S)$ in $O(1)$ time.

**Proof.** Define width($i, j$) = $|i - j| + 1$. We say that parentheses of the same type with property $\pi$ are $d$-disjoint if

- width($i, \text{match}(S, i)) > d$ holds for any parenthesis $S[i]$ with property $\pi$; and
- any two property-$\pi$ parentheses $S[i]$ and $S[j]$ with $S[i] = S[j]$ satisfy at least one of width($i, j > d$ and width(match($S, i$), match($S, j$)) $> d$.

Intuitively, $d$-disjoint parentheses have to be sparse: Suppose that parentheses with property $\pi$ are $d$-disjoint. Then, for any property-$\pi$ open parenthesis $S[i]$, either $S[i]$ is the only property-$\pi$ open parenthesis in $S[i - d + 1, i]$ or $S[\text{match}(S, i)]$ is the only property-$\pi$ closed parenthesis in $S[\text{match}(S, i), \text{match}(S, i) + d - 1]$. As a result, suppose that we partition $S$ into segments of length $d$. For each segment, let us mark (a) the property-$\pi$ open parenthesis with the smallest index in the segment and
(b) the property-π closed parenthesis with the largest index in the segment. Then, for each parenthesis \( S[i] \) with property \( \pi \), at least one of \( S[i] \) and \( S[\text{match}(S, i)] \) is marked. Therefore, there are only \( O(s/d) \) parentheses with property \( \pi \).

Let \( s = |S| \). For a carefully chosen number \( \ell = \Theta(\log s) \), we say that
- parenthesis \( S[i] \) is narrow if width\((i, \text{match}(S, i))\) \( \leq \ell \);
- parenthesis \( S[i] \) is wide if \( \text{wrapped}(S, i) > 2\ell^2 \); and
- parenthesis \( S[i] \) is medium if it is neither narrow nor wide.

We apply the commonly used preprocessing technique (see, e.g., [10, 35]) in the unit-cost RAM model which allows the query \( \text{wrapped}(S, i) \) for any narrow \( S[i] \) to be answered in \( O(1) \) time from the linear-time precomputable \( o(s) \)-bit table (i.e., the table \( M_1 \circ M_2 \) to be described later). It is not difficult to see that wide parentheses are \( \ell^2 \)-disjoint. Therefore, we can afford to encode \((i, \text{wrapped}(S, i))\) for all wide parentheses \( S[i] \) using \( o(s) \) bits. Although medium parentheses are not necessarily \( \ell \)-disjoint, we identify \( \text{special} \) parentheses, which are medium parentheses that have to be \( \ell \)-disjoint, and encode \((i, \text{wrapped}(S, i))\) for all special medium parentheses \( S[i] \) using \( o(s) \) bits. As for medium parentheses that are not special, we will show that two queries to the \( o(s) \)-bit precomputed table suffice. The details are as follows.

Let \( t \) be the number of distinct types of parentheses in \( S \). Let \( b \) be the smallest integer with \( 2t \leq 2^b \). Each symbol of \( S \) can be encoded in \( b \) bits. As \( t = O(1) \), we have \( b = O(1) \). Let \( \ell = \left\lfloor \frac{1}{2} \log_2 s \right\rfloor \). Any substring \( S[i, j] \) with \( j \leq i + \ell - 1 \) has \( O(\sqrt{s}) \) possible distinct values. Define tables \( M_1 \) and \( M_2 \) for \( S \) by letting \( M_1[S[i, i + \ell - 1]] = \text{wrapped}(S[i, i + \ell - 1], 1) \) and \( M_2[S[i, j]] = \text{wrapped}(\text{reverse}(S[i, j]), 1) \) for any \( i, j \) with \( 1 \leq i \leq j \leq i + \ell - 1 \). One can easily come up with an \( o(s) \)-bit string \( \chi'_2 \) from which each entry of \( M_1 \) and \( M_2 \) can be obtained in \( O(1) \) time.

For each \( k \in \{1, 2, \ldots, \ell\} \), define tables \( M^k_3 \) and \( M^k_4 \) as follows. For each \( i = 1, 2, \ldots, \left\lfloor \frac{s}{2} \right\rfloor \),
- let \( M^k_3[i] = (j, \text{wrapped}(S, j)) \), where index \( j \) is the smallest index, if any, with \((i - 1)\ell^2 < j \leq i\ell^2 \) such that \( S[j] \) is a wide open parenthesis of type \( k \); and
- let \( M^k_4[i] = (j, \text{wrapped}(S, j)) \), where index \( j \) is the largest index, if any, with \((i - 1)\ell^2 < j \leq i\ell^2 \) such that \( S[j] \) is a wide close parenthesis of type \( k \).

Since each entry of \( M^k_3 \) and \( M^k_4 \) can be encoded in \( O(\log s) \) bits, one can easily obtain an \( o(s) \)-bit string \( \chi''_2 \) from which each entry of \( M^k_3 \) and \( M^k_4 \) can be determined in \( O(1) \) time.

A medium open parenthesis \( S[i] \) is \( \text{special} \) if at least one of the inequalities width\((i, j) > \ell \) and width\((\text{match}(S, i), \text{match}(S, j)) > \ell \) holds for each index \( j \) with \( i < j < \text{match}(S, i) \) and \( S[j] = S[i] \). A closed parenthesis \( S[i] \) is \( \text{special} \) if \( S[\text{match}(S, i)] \) is special. One can verify that special parentheses are \( \ell \)-disjoint. For each \( k \in \{1, 2, \ldots, \ell\} \), define tables \( M^k_5 \) and \( M^k_6 \) as follows. For each \( i \in \{1, 2, \ldots, \left\lfloor \frac{s}{2} \right\rfloor \},
- let \( M^k_5[i] = (j, \text{wrapped}(S, j)) \), where index \( j \) is the smallest index, if any, with \((i - 1)\ell < j \leq i\ell \) such that \( S[j] \) is a special open parenthesis of type \( k \); and
- let \( M^k_6[i] = (j, \text{wrapped}(S, j)) \), where index \( j \) is the largest index, if any, with \((i - 1)\ell < j \leq i\ell \) such that \( S[j] \) is a special close parenthesis of type \( k \).

Observe that \( M^k_5[i] = (j, c) \) or \( M^k_6[i] = (j, c) \) implies \( 0 \leq i\ell - j \leq \ell \) and \( 0 \leq c \leq 2\ell^2 \). Therefore, each entry of \( M^k_5 \) and \( M^k_6 \) can be encoded in \( O(\log \ell) = O(\log \log s) \) bits. As a result, one can easily come up with an \( o(s) \)-bit string \( \chi'''_2 \) from which each entry of \( M^k_5 \) and \( M^k_6 \) can be determined in \( O(1) \) time. Let \( \chi_2(S) = \chi'_2 \circ \chi''_2 \circ \chi'''_2 \). The \( o(s) \)-bit string \( \chi_2(S) \) can be derived from \( S \) in \( O(s) \) time.

It remains to show that \( \text{wrapped}(S, i) \) can be determined from \( S \) and \( \chi_2(S) \) by
the algorithm shown in Figure 5.1, which clearly runs in $O(1)$ time. If a value $c$ is returned from steps 4–9, we have $c = \text{wrapped}(S, i)$. The rest of the proof assumes that step 10 is executed. Since wide parentheses are $\ell^2$-disjoint and special parentheses are $\ell$-disjoint, parentheses $S[i_1]$ and $S[i_2]$ at step 10 satisfy $\text{width}(i_1, i_2) > 2\ell$ and form a matching pair of medium parentheses that are not special. By definition of special parentheses, there are indices $j_1$ and $j_2$ with $i_1 < j_1 < j_2$ such that $S[i_1] = S[j_1]$ (thus $S[j_2] = S[i_2]$) such that $\text{width}(i_1, j_1) \leq \ell$ and $\text{width}(j_2, i_2) \leq \ell$. Since $\text{width}(i_1, i_2) > 2\ell$, we have $\text{wrapped}(S, i_1) = M_1[S[i_1, i_1 + \ell - 1]] + M_2[S[i_2 - \ell + 1, i_2]]$. Therefore, step 10 correctly returns $\text{wrapped}(S, i)$. \qed

A folklore encoding [22, 35, 9] $S$ of an $n$-node simple rooted tree $T$ is a balanced string of $2n$ parentheses representing a counterclockwise depth-first traversal of $T$. Initially, an open (respectively, closed) parenthesis denotes a descending (respectively, ascending) edge traversal. Then, this string is enclosed by an additional matching parenthesis pair. For example, the string in (5.2) is the folklore encoding for the tree $T$ in Figure 1.1(a). Let $v_i$ be the $i$th node in the counterclockwise depth-first traversal. Let $\ell$ be the $i$th open parenthesis in $S$. Let $\l_j$ be the closed parenthesis of $S$ that matches $\ell_i$ in $S$. Node $v_i$ corresponds to $(\ell_i, \l_j)$ in that $v_i$ is the parent of $v_j$ in $T$ if and only if $\ell_i$ and $\l_j$ form the closest pair of matching parentheses that encloses $(\ell_i, \l_j)$. Also, the number of children of $v_i$ in $T$ is precisely $\text{wrapped}(S, \text{select}(S, i, )))/2$, which is also equal to $\text{wrapped}(S, \text{match}(S, \text{select}(S, i, ))))/2$.

Let $H$ be an $n$-node connected plane graph that may have multiple edges but no self-loops. Let $T$ be a spanning tree of $H$ rooted at $v_1$. Let $v_1, v_2, \ldots, v_n$ be the counterclockwise preordering of $T$. Let $\text{degree}(i)$ be the number of edges incident to $v_i$ in $H$. Let $\text{children}(i)$ be the number of children of $v_i$ in $T$. Let $\text{above}(i)$ (respectively, $\text{below}(i)$) be the number of edges $(v_i, v_j)$ of $H$ such that $v_j$ is the parent (respectively, a child) of $v_i$ in $T$. Let $\text{low}(i)$ (respectively, $\text{high}(i)$) be the number of edges $(v_i, v_j)$ of $H$ such that $j < i$ (respectively, $j > i$) and $v_j$ is neither the parent nor a child of $v_i$ in $T$. Now, $\text{degree}(i) = \text{above}(i) + \text{below}(i) + \text{low}(i) + \text{high}(i)$. If $H$ has no multiple edges, then $\text{below}(i) = \text{children}(i)$. If $H$ and $T$ are as shown in Figure 1.1(a), for instance, then $\text{above}(3) = 1$, $\text{below}(3) = \text{children}(3) = 2$, $\text{low}(3) = 1$, $\text{high}(3) = 2$, and $\text{degree}(3) = 6$.

The $T$-code of $H$ is a triple $(S_1, S_2, S_3)$ of the binary strings $S_1, S_2, S_3$, where $\delta_{i \geq 2}$ equals 1 if $i \geq 2$ and 0 otherwise.

- $S_1$ is the folklore encoding of $T$.

\begin{figure}
\centering
\begin{algorithm}
\begin{footnotesize}
\begin{algorithmic}
\Function{\text{wrapped}}{S, i}
\State 1. let $k$, with $1 \leq k \leq t$, be the type of $S[i]$;
\State 2. let $i_1 = \min\{i, \text{match}(S, i)\}$;
\State 3. let $i_2 = \text{match}(S, i_1)$;
\State 4. let $(j, c) = M_2[\lfloor i_1/\ell \rfloor]$; if $j = i_1$, then return $c$;
\State 5. let $(j, c) = M_2[\lfloor i_2/\ell \rfloor]$; if $j = i_2$, then return $c$;
\State 6. let $(j, c) = M_2[\lfloor i_1/\ell \rfloor]$; if $j = i_1$, then return $c$;
\State 7. let $(j, c) = M_2[\lfloor i_2/\ell \rfloor]$; if $j = i_2$, then return $c$;
\State 8. if $\text{width}(i_1, i_2) \leq \ell$, then return $M_1[S[i_1, i_2]]$;
\State 9. if $\text{width}(i_1, i_2) \leq 2\ell$, then return $M_1[S[i_1, i_1 + \ell - 1]] + M_2[S[i_1 + \ell, i_2]]$;
\State 10. return $M_1[S[i_1, i_1 + \ell - 1]] + M_2[S[i_2 - \ell + 1, i_2]]$;
\EndFunction
\end{algorithmic}
\end{footnotesize}
\end{algorithm}
\caption{An $O(1)$-time algorithm that computes $\text{wrapped}(S, i)$.}
\end{figure}
ORDERLY SPANNING TREES WITH APPLICATIONS

- Let \( p_i = \text{select}(S_1, i, \cdot) \) and \( q_i = \text{match}(S_1, p_i) \). \( S_2 \) has exactly \( 2n \) copies of 1, in which \( \text{low}(i) \) copies of 0 immediately succeed the \( p_i \)th 1, and \( \text{high}(i) \) copies of 0 immediately succeed the \( q_i \)th 1.
- \( S_3 \) has exactly \( n \) copies of 1, where above\( (i) + \text{below}(i) - \text{children}(i) - \delta_{i \geq 2} \) copies of 0 immediately succeed the \( i \)th 1.

For example, if \( H \) and \( T \) are as shown in Figure 1.1(a), then

\[
\begin{align*}
S_1 &= (\{(\cdot)\})\{(\cdot)\}((\cdot)\{(\cdot)\})\{(\cdot)\}; \\
S_2 &= 1110000010101001001000101010001000101010010000111; \\
S_3 &= 111111111111.
\end{align*}
\]

We have that

\[
\begin{align*}
|S_1| &= 2n; \\
|S_2| &= 2n + \sum_{i=1}^{n} (\text{low}(i) + \text{high}(i)); \\
|S_3| &= 1 + \sum_{i=1}^{n} (\text{above}(i) + \text{below}(i) - \text{children}(i)).
\end{align*}
\]

Therefore, \( |S_1| + |S_2| + |S_3| = 2m + 3n + 2 \). Moreover, if \( H \) has no multiple edges, then \( |S_3| = n \), and thus \( |S_1| + |S_2| = 2n + 2n + 2 \).

The next theorem describes our convenient encoding. The techniques in the proof are mostly adapted from [9]. (Their encoding needs initial augmentation to the input graph to ensure that the resulting graph admits a canonical spanning tree. As a result, their encoding requires an additional number of bits to tell which edges are original.)

**Theorem 5.4.** Let \( G \) be an input \( n \)-node \( m \)-edge planar graph having no self-loops. If \( G \) has (respectively, has no) multiple edges, then \( G \) has a convenient encoding, obtainable in \( O(m + n) \) time, with \( 2m + 3n + o(m + n) \) (respectively, \( 2m + 2n + o(n) \)) bits.

**Proof.** We focus on the case that \( G \) is connected. As sketched at the end of the proof, it is not difficult to remove this restriction. By Theorem 2.3, an orderly pair \((H, T)\) of \( G \) can be derived in \( O(m + n) \) time. Let \((S_1, S_2, S_3)\) be the \( T \)-code of \( H \).

We prove that there exists an \( o(m + n) \)-bit string \( \chi \), obtainable in \( O(m + n) \) time, such that \( S_1 \circ S_2 \circ S_3 \circ \chi \) is a convenient encoding of \( G \). If \( G \) has no multiple edges, then \( S_3 \) consists of \( n \) copies of 1, and thus \( S_1 \circ S_2 \circ \chi \) will suffice.

To support degree queries, let \( p_i = \text{select}(S_1, i, \cdot) \) and \( q_i = \text{match}(S_1, p_i) \). Since \( S[p_i] \) and \( S[q_i] \) are a matching parenthesis pair, we have that

\[
\begin{align*}
\text{low}(i) &= \text{select}(S_2, p_i + 1, 1) - \text{select}(S_2, p_i, 1) - 1, \\
\text{high}(i) &= \text{select}(S_2, q_i + 1, 1) - \text{select}(S_2, q_i, 1) - 1.
\end{align*}
\]

Observe that \( \text{children}(i) = \text{wrapped}(S_1, p_i)/2 \). From the definition of \( S_3 \), we know \( \text{above}(i) + \text{below}(i) - \text{children}(i) = \text{select}(S_3, i + 1, 1) - \text{select}(S_3, i, 1) - 1 + \delta_{i \geq 2} \). Let \( \chi' = \chi_1(S_1) \circ \chi_1(S_2) \circ \chi_1(S_3) \circ \chi_2(S_1) \). From degree\( (i) = \text{above}(i) + \text{below}(i) + \text{low}(i) + \text{high}(i) \), Fact 5.2, and Lemma 5.3, we determine that degree\( (i) \) is computable from \( S_1 \circ S_2 \circ S_3 \circ \chi' \) in \( O(1) \) time.

To support adjacency queries and listing of all neighbors, we introduce a string \( S \) of two types of parentheses derived from \( S_1 \) and \( S_2 \) as follows. Although \( S \) is only implicitly represented in our convenient encoding, any \( O(\log n) \) consecutive parentheses of \( S \) can be obtained from \( S_1, S_2 \), and some auxiliary string in \( O(1) \) time. Let \( \chi \)
and ) be of type 1, and let [ and ] be of type 2. Initially, for each \( i = 1, 2, \ldots, 2n \), replace the \( r \)th 1 of \( S_1 \) with \( S_1[i] \). Then, replace each 0 of \( S_2 \) with a bracket such that the bracket is open if and only if the last parenthesis in \( S \) preceding this 0 is closed. More precisely, for each \( i = 1, 2, \ldots, |S_2| \), let

\[
S[i] = \begin{cases} 
S_1[j_1] & \text{if } S_2[i] = 1, \\
] & \text{if } S_2[i] = 0 \text{ and } S_1[j_1] = \), \\
[ & \text{if } S_2[i] = 0 \text{ and } S_1[j_1] = ).
\end{cases}
\]

where \( j_1 = \text{rank}(S_2, i, 1) \). For example, if \( H \) and \( T \) are as given in Figure 1.1(a), then \( S \) is as in (5.1). There exists an auxiliary string \( \chi_3 \) such that any \( O(\log n) \) consecutive symbols of \( S \) are obtainable from \( S_1 \circ S_2 \circ \chi_3 \) in \( O(1) \) time: Let \( \ell = \lceil \frac{1}{2} \log_2 n \rceil \). Observe that the content of \( S[i, i + \ell - 1] \) can be uniquely determined from the concatenation \( S' \) of \( S_2[i, i + \ell - 1] \) and \( S_1[j, j + \ell - 1] \) with \( j = \text{rank}(S_2, i, 1) \). Also, \( S' \) is obtainable from \( S_1 \circ S_2 \circ \chi_1(S_2) \) in \( O(1) \) time. Since \( S' \) has \( 4^\ell \) distinct values, we can precompute in \( O(n) \) time an \( o(n) \)-bit table \( M \) such that the content of \( S[i, i + \ell - 1] \) is obtainable from \( S' \) and \( M \) in \( O(1) \) time. Hence, it suffices to let \( \chi_3 = M \circ \chi_1(S_2) \).

With the help of \( S \) and its auxiliary strings, adjacency queries can be supported as follows. For any two integers \( a \) and \( b \), let \([a, b]\) consist of the integers \( a, a+1, \ldots, b \). For each \( i \in \{1, 2, \ldots, n\} \), let

\[
L_i = [\ell_i + 1, \text{select}(S_2, \text{rank}(S_2, \ell_i, 1) + 1, 1, 1) - 1],
\]

\[
R_i = [h_i + 1, \text{select}(S_2, \text{rank}(S_2, h_i, 1) + 1, 1) - 1],
\]

where \( \ell_i = \text{select}(S, i, ) \) and \( h_i = \text{match}(S, \ell_i) \). Let \((v_i, v_j)\) and \((v_{i'}, v_{j'})\), with \( i < j \) and \( i' < j' \), be two unrelated edges of \( H \) with respect to \( T \). Since \( T \) is an orderly spanning tree of \( H \), one can see that if \((v_{i'}, v_{j'})\) is enclosed by the cycle of \( H \) determined by \( T \) and \((v_i, v_j)\), then \( h_i < h_{i'} < \ell_{j'} < \ell_j \). One can prove that \( v_i \) and \( v_j \), with \( i < j \), are adjacent in \( H \) if and only if there exists an index \( \ell \in L_i \) with \( \text{match}(S, \ell) \in L_j \) and \( \ell \) is contained in the brackets of \( S \).

The above statement clearly holds when \( b = 0 \). To show the induction step for any \( b \geq 1 \), let \( e = (v_x, v_y) \) be an edge in \( H - T \). We know that \( T \) is also an orderly spanning tree of \( H - \{e\} \). Let \( \hat{S}, \hat{R}_i, \) and \( \hat{L}_i \) be the corresponding notation for \( H - \{e\} \) with respect to \( T \). Observe that \( \hat{S} \) can be obtained from \( S \) by deleting an open bracket in a position in \( R_x \) and deleting a closed bracket in a position in \( L_y \). Since \( v_x \) and \( v_y \) are not adjacent in \( H - \{e\} \), it follows from the inductive hypothesis that match(\( \hat{S}, \ell \) \( \notin \hat{L}_y \) holds for any index \( \ell \) in \( R_x \); and match(\( \hat{S}, \ell \) \( \not\in \hat{R}_x \) holds for any index \( \ell \) in \( \hat{L}_y \). Therefore, there is a position in \( \hat{R}_x \) and a position in \( \hat{L}_y \) such that if we insert an open bracket in the first position and a closed bracket in the second position, then the brackets will match each other in the resulting string, which is exactly \( S \). Thus, the above statement is proved.

Thus, one can determine whether \((v_i, v_j)\) is an unrelated edge of \( H \) with respect to \( T \), by checking whether \( i'' \in R_i \) and \( j'' \in L_j \) hold, where

\[
(i'', j'') = \text{enclose}_2(S, \text{select}(S, \text{rank}(S_2, h_i, 1) + 1, 1, ), \ell_j).
\]

Therefore, the answer to each adjacency query is derivable from \( S_2 \circ S \circ \chi_1(S_2) \circ \chi_1(S) \) in \( O(1) \) time.
ORDERLY SPANNING TREES WITH APPLICATIONS

The neighbors of a degree-$d$ node $v_i$ can be listed from $S \circ \chi_1(S)$ in $O(d)$ time: If $v_i$ is not the root of $T$, then the parent of $v_i$ is $v_j$, where $j$ is computable by letting

\[(j_1, j_2) = \text{enclose}(S, \text{select}(S, i, \cdot), \text{match}(S, \text{select}(S, i, \cdot)));\]

\[j = \text{rank}(S, j_1, \cdot).\]

If $v_i$ is not a leaf of $T$, then $v_{i+1}$ is the first child of $v_i$ in $T$. If $v_j$ is the $t$th child of $v_i$ in $T$, then the $(t+1)$st child of $v_i$ in $T$ is $v_k$, where

\[k = \text{rank}(S, 1 + \text{match}(S, \text{select}(S, j, \cdot), \cdot)).\]

If $t \leq |U_{<}(v_i)|$, the $t$th neighbor of $v_i$ in $U_{<}(v)$ with respect to $T$ is $v_j$, where $j$ is computable by

\[j_1 = \text{match}(S, t + \text{select}(S, i, \cdot));\]

\[j_2 = \text{select}(S, \text{rank}(S, j_1, \cdot), \cdot));\]

\[j = \text{rank}(S, \text{match}(S, j_2), \cdot).\]

If $t \leq |D(v_i)|$, then the $t$th neighbor of $v_i$ in $D(v)$ with respect to $T$ is $v_j$, where $j$ is computable by $j_1 = \text{match}(S, \text{select}(S, i, \cdot))$ and $j = \text{rank}(S, \text{match}(S, j_1 + t), \cdot)$.

It is not difficult to verify that $G$ can be reconstructed from $S$ and $S_3$ in $O(m + n)$ time. Therefore, the theorem for connected planar graphs is proved by letting $\chi = \chi_1 \circ \chi_3 \circ \chi_1(S)$. As for the case that $G$ has $k \geq 2$ connected components, by Theorem 2.3, an orderly pair $(H^i, T^i)$ of the $i$th connected component $G^i$ of $G$ can be derived in overall $O(m + n)$ time. Let $(S^i_1, S^i_2, S^i_3)$ be the $T^i$-code of $H^i$. For each $j = 1, 2, 3$, let $S_j$ be the concatenation of $S^i_1, S^i_2, \ldots, S^i_k$. The theorem can then be proved similarly using $S_1, S_2,$ and $S_3$. \[\square\]

Acknowledgments. We are grateful to the anonymous reviewers for their helpful suggestions which significantly improved the paper. We thank Xin He for helpful comments. We thank Hsu-Chun Yen and Ho-Lin Chen for bringing [16] to our attention. We also thank Richie Chih-Nan Chuang, Yuan-Jiunn Wang, and Kai-Ju Liu for discussions.

REFERENCES

[1] T. C. Bell, J. G. Cleary, and I. H. Witten, Text Compression, Prentice–Hall, Englewood Cliffs, NJ, 1990.

[2] N. Bonichon, C. Gavoille, and N. Hanusse, An information-theoretic upper bound of planar graphs using triangulation, in Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2607, Springer-Verlag, Berlin, 2003, pp. 499–510.

[3] P. Bose, A. M. Dean, and J. P. Hutchinson, On rectangle visibility graphs, in Proceedings of the 4th International Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 1190, Springer-Verlag, Berlin, 1996, pp. 25–44.

[4] J. Boyer and W. Myrvold, Stop minding your $p’s$ and $q’s$: A simplified $O(n)$ planar embedding algorithm, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 140–146.

[5] A. Brodnik and J. I. Munro, Membership in constant time and almost-minimum space, SIAM J. Comput., 28 (1999), pp. 1627–1640.

[6] H.-L. Chen, C.-C. Liao, H.-I. Lu, and H.-C. Yen, Some applications of orderly spanning trees in graph drawing, in Proceedings of the 10th International Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 2528, Springer-Verlag, Berlin, 2002, pp. 332–343.

[7] M. Chrobak and S.-I. Nakano, Minimum-width grid drawings of plane graphs, Comput. Geom., 11 (1998), pp. 29–54.
[8] M. Chrobak and T. H. Payne, *A linear-time algorithm for drawing a planar graph on a grid*, Inform. Process. Lett. 54 (1995), pp. 241–246.

[9] R. C.-N. Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I. Lu, *Compact encodings of planar graphs via canonical ordering and multiple parentheses*, in Proceedings of the 25th International Colloquium on Automata, Languages, and Programming, K. G. Larsen, S. Skyum, and G. Winskel, eds., Lecture Notes in Comput. Sci. 1443, Springer-Verlag, Berlin, 1998, pp. 118–129.

[10] D. R. Clark, *Compact PAT Trees*, Ph.D. thesis, University of Waterloo, Ontario, Canada, 1996.

[11] H. de Fraysseix, J. Pach, and R. Pollack, *How to draw a planar graph on a grid*, Combinatorica, 10 (1990), pp. 41–51.

[12] A. M. Dean and J. P. Hutchinson, *Rectangle-visibility representations of bipartite graphs*, Discrete Appl. Math., 75 (1997), pp. 9–25.

[13] A. M. Dean and J. P. Hutchinson, *Rectangle-visibility layouts of unions and products of trees*, J. Graph Algorithms Appl. 2 (1998), pp. 1–21.

[14] B. Dushnik and E. W. Miller, *Partially ordered sets*, Amer. J. Math., 63 (1941), pp. 600–610.

[15] P. Elias, *Universal codeword sets and representations of the integers*, IEEE Trans. Inform. Theory, 21 (1975), pp. 194–203.

[16] U. Fößmeier, G. Kant, and M. Kaufmann, *2-visibility drawings of planar graphs*, in Proceedings of the 4th International Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 1190, Springer-Verlag, Berlin, 1996, pp. 155–168.

[17] M. L. Fredman and D. E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. System Sci., 48 (1994), pp. 533–551.

[18] C. Gavoille and N. Hanusse, *Compact routing tables for graphs of bounded genus*, in Proceedings of the 26th International Colloquium on Automata, Languages, and Programming, J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds., Lecture Notes in Comput. Sci. 1644, Springer-Verlag, Berlin, 1999, pp. 351–360.

[19] D. Harel and M. Sardas, *An algorithm for straight-line drawing of planar graphs*, Algorithmica, 20 (1998), pp. 119–135.

[20] X. He, *On floor-plan of plane graphs*, SIAM J. Comput., 28 (1999), pp. 2150–2167.

[21] X. He, M.-Y. Kao, and H.-I. Lu, *A fast general methodology for information-theoretically optimal encodings for graphs*, in Proceedings of the 7th Annual European Symposium on Algorithms, J. Nešetřil, ed., Lecture Notes in Comput. Sci. 1643, Springer-Verlag, Berlin, 1999, pp. 540–549.

[22] X. He, M.-Y. Kao, and H.-I. Lu, *Linear-time succinct encodings of planar graphs via canonical orderings*, SIAM J. Discrete Math., 12 (1999), pp. 317–325.

[23] X. He, M.-Y. Kao, and H.-I. Lu, *A fast general methodology for information-theoretically optimal encodings of graphs*, SIAM J. Comput., 30 (2000), pp. 838–846.

[24] J. Hopcroft and R. E. Tarjan, *Efficient planarity testing*, J. ACM, 21 (1974), pp. 549–568.

[25] J. P. Hutchinson, T. Shermer, and A. Vince, *On representations of some thickness-two graphs*, Comput. Geom., 13 (1999), pp. 161–171.

[26] IEEE, *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, Los Alamitos, CA, 1997.

[27] G. Kant, *Drawing planar graphs using the canonical ordering*, Algorithmica, 16 (1996), pp. 4–32.

[28] G. Kant and X. He, *Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems*, Theoret. Comput. Sci., 172 (1997), pp. 175–193.

[29] K. Keeler and J. Westbrook, *Short encodings of planar graphs and maps*, Discrete Appl. Math., 58 (1995), pp. 239–252.

[30] C.-C. Liao, H.-I. Lu, and H.-C. Yen, *Floor-planning via orderly spanning trees*, in Proceedings of the 9th International Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 2265, Springer-Verlag, Berlin, 2001, pp. 367–377.

[31] C.-C. Liao, H.-I. Lu, and H.-C. Yen, *Compact floor-planning via orderly spanning trees*, J. Algorithms, 48 (2003), pp. 441–451.

[32] C.-C. Lin, H.-I. Lu, and I.-F. Sun, *Improved compact visibility representation of planar graph via Schnyder’s realizer*, SIAM J. Discrete Math., 18 (2004), pp. 19–29.

[33] H.-I. Lu, *Improved compact routing tables for planar networks via orderly spanning trees*, in Proceedings of the 8th Annual International Computing and Combinatorics Conference, Lecture Notes in Comput. Sci. 2367, Springer-Verlag, Berlin, 2002, pp. 57–66.

[34] H.-I. Lu, *Linear-time compression of bounded-genus graphs into information-theoretically optimal number of bits*, in Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2002, pp. 223–224.
[35] J. I. Munro and V. Raman, Succinct representation of balanced parentheses and static trees, SIAM J. Comput., 31 (2001), pp. 762–776.

[36] S. North, ed., Proceedings of the 4th International Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 1190, Springer-Verlag, Berlin, 1996.

[37] J. Rossignac, Edgebreaker: Connectivity compression for triangle meshes, IEEE Trans. Visualization and Computer Graphics, 5 (1999), pp. 47–61.

[38] W. Schnyder, Planar graphs and poset dimension, Order, 5 (1989), pp. 323–343.

[39] W. Schnyder, Embedding planar graphs on the grid, in Proceedings of the 1st Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1990, pp. 138–148.

[40] M. Thorup, Undirected single source shortest paths in linear time, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1997, pp. 12–21.

[41] M. Thorup, On RAM priority queues, SIAM J. Comput., 30 (2000), pp. 86–109.

[42] W. T. Trotter, Combinatorics and Partially Ordered Sets—Dimension Theory, Johns Hopkins University Press, Baltimore, MD, 1992.

[43] G. Turán, On the succinct representation of graphs, Discrete Appl. Math., 8 (1984), pp. 289–294.

[44] W. T. Tutte, A census of planar maps, Canad. J. Math., 15 (1963), pp. 249–271.

[45] P. van Emde Boas, Machine models and simulations, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 1–60.

[46] M. Yannakakis, Embedding planar graphs in four pages, J. Comput. System Sci., 38 (1989), pp. 36–67.

[47] K.-H. Yeap and M. Sarrafzadeh, Floor-planning by graph dualization: 2-concave rectilinear modules, SIAM J. Comput., 22 (1993), pp. 500–526.