with one determinant calculation.

This completes our merge algorithm. To apply it to compute all proper antipodal pairs we need only arrange the input in the correct form: i.e. construct the two input lists corresponding to the upper and lower support maps. However, there is no need to explicitly create these lists! Since they both have the same number of nodes and ordering as the list representing P, we may just as well work with P. All that is required is to show that for each node v of P, we can quickly compute the fields $first(v)$ and $last(v)$ for the respective support maps. But this is easily done. For instance, for the upper support map let v correspond to node p_i. Then $first(v)$ is the vector obtained by rotating $p_i - p_{i-1}$ counterclockwise by $\pi/2$ and normalizing it. Similarly, $last(v)$ is the vector obtained by rotating $p_{i+1} - p_i$ counterclockwise. In practice one can avoid rotating the vectors, and consider the algorithm to be working on upper and lower support maps which are rotated $\pi/2$ *clockwise*.

In $merge(a, b)$, a, b can be two pointers to the list representing P. They can be thought of as the contact points of a pair of rotating calipers. The main step of the algorithm can then be interpreted as a simulation of the calipers rotating about the point set. The reader is urged to simulate the operation of merge on a small set of points, interpreting the steps in these terms.

2.5. Computing the Convex Hull

There are several algorithms for computing the convex hull of a set of n points in the plane in $O(n \log n)$ time. The first published algorithm with this complexity was due to Ron Graham. It has two steps. In the first step, the points are sorted in angular order about some interior point of the convex hull. The second step scans the points in this order and deletes all points interior to the convex hull using a local test. There have since been several variants of this algorithm. Although conceptually very simple, there are certain technical difficulties due to collinearity of points and other "degeneracies". A simple way around this is to assume that the point set does not contain these unpleasant features and leave it to the reader (or program-

mer) to take care of them. Most programmers complain, however, that it is just the degeneracies and special cases that take up most of the code. Since the convex hull problem is relatively straight forward, we will try to develop as complete a procedure as possible in this section, without making any assumptions about the input data.

We describe a variant of Graham's algorithm in which the points are initially sorted lexicographically according to their first and second coordinates. The points are processed in this order updating the convex hull as each new point is added. We will see that at each step, the new point being considered is a vertex of the updated convex hull. Some other points may no longer be vertices and must be deleted from the list of convex hull vertices. In fact, the heart of the algorithm is in identifying these vertices quickly using some local tests. Figure 2.13 contains an example of the main step of the procedure. In Figure 2.13(a) a point p is added to the set of convex hull vertices $P = \{p_1, ..., p_7\}$. Supporting lines are drawn from p to P, and the supports *first* and *last* are identified. In case a supporting line contains an edge, as is the case of the line containing $[p_2, p_3]$, the point furthest from p is identified. In Figure 2.13(b), the chain between *first* and *last* is deleted and replaced by the single vertex p.

We will now describe the theoretical basis of the procedure suggested by the example, then we will discuss the implementation. Suppose we have so far obtained a set $P=\{p_1, \cdots, p_n\}$ of $n \geq 3$ points that are all convex hull vertices of P arranged in sorted clockwise order. Let each point p_i have coordinates (p_i^1, p_i^2). We now define a lexicographic ordering "$\ll$" on P by saying that $p_i \ll p_j$ whenever

$$\text{(i) } p_i^1 < p_j^1 \text{ , or}$$

$$\text{(ii) } p_i^1 = p_j^1 \text{ and } p_i^2 < p_j^2.$$

According to this order, we may assume that the points of P have been labelled so that the first point is p_1, i.e. $p_1 \ll p_i$, $i \neq 1$. Let the last point in the order be p_j. This lexicographic ordering is compatible with the angular order of the points in the following sense (which you are asked to prove in Exercise 2.6)

Lemma 2.6 $p_1 \ll p_2 \ll \cdots \ll p_j$ and $p_1 \ll p_n \ll p_{n-1} \ll \cdots \ll p_j$. □

Since we are processing points in lexicographic order, our next point p will satisfy $p \gg p_j$. Thus, for any point $p \gg p_j$ in the plane, let $P'=P \cup \{p\}$. We would like to efficiently compute the vertices of $CH(P')$ from P, the vertices of $CH(P)$. The following lemma is crucial for this computation.

Lemma 2.7 p is a vertex of $CH(P')$.

Proof: Let $a=(1,0)$ and let $b=ap$. Since $p \gg p_j$, it is easy to see that the vertical line $l_{a,b}$ is an upper supporting line through p for $CH(P')$. If the support is p, we are done, otherwise the support is the vertical segment having the form $[p, p_j]$ or $[p, p_k]$, with $p_j \gg p_k$. In the first case $p^1=p_j^1$ and $p^2 > p_j^2$, so p is an endpoint of the segment. In the second case, $p^1 = p_k^1$ and $p^2 > p_j^2 > p_k^2$ with the same conclusion. □

In order to compute $CH(P')$, it will suffice to compute the endpoints of the two edges that originate at p. This will indicate a (possibly empty) chain of vertices in P to be deleted. The endpoints are characterized by the following theorem.

Theorem 2.8. Let $P = \{p_1, ..., p_n\}$, $n \geq 3$ be a set of n convex hull vertices in clockwise order, with p_j defined as in Lemma 2.6. Let $p \gg p_j$ be a point in the plane, and let $P'=P \cup \{p\}$. The segment $[p_i,p]$ is an edge of $CH(P')$ if and only if there is a line $l_{a,b}$ containing p_i and p such that

(i) $a(p_{i-1}-p_i) < 0$, $a(p_i-p_{i+1}) \geq 0$ when $i \leq j$, and

(ii) $a(p_{i+1}-p_i) < 0$, $a(p_i-p_{i-1}) \geq 0$ when $i \geq j$.

Proof: Suppose $[p_i,p]$ is an edge of $CH(P')$. Then there is an upper supporting line $l_{a,b}$ of $CH(P')$ with support $[p_i,p]$. Therefore $a(p_k-p) \leq 0$, $k=1,...,n$. Suppose that $i \leq j$ and $a(p_{i-1}-p) = 0$. Then $l_{a,b}$ contains the points p_{i-1}, p_i, p and by Lemma 2.6 $p_{i-1} \ll p_i \ll p$, so p_i is not a vertex. Thus (i) holds for $i \leq j$, and a similar argument shows that (ii) holds when $i \geq j$.

Conversely, suppose that there is a line $l_{a,b}$ containing $[p_i,p]$, for some $i \leq j$, and satisfying (i). Recall Section 2.3 where we characterized upper supporting lines for vertices. As p_i is a vertex of $CH(P)$, a must belong to the set

$$U_i=\{a : a(p_{i+1}-p_i) \leq 0,\ a(p_i-p_{i-1}) \geq 0\}.$$

Noting that $b=ap_i=ap$, we see that $a \in U_i$ and so $a(p_k-p_i) \leq 0$, $k=1,...,n$. Condition (i) implies that a is not contained in

$$U_{i-1}=\{a : a(p_i-p_{i-1}) \leq 0,\ a(p_{i-1}-p_{i-2}) \geq 0\},$$

since the first inequality is violated. $l_{a,b}$ supports $CH(P')$ at precisely $[p_i, p]$. A similar argument applies to the case when $i \geq j$. □

Theorem 2.8 implies that *first* is the unique vertex of $p_1, ..., p_j$ satisfying condition (i), and *last* is the unique vertex of $p_j, ..., p_n$ satisfying condition (ii). Note that conditions are exclusive, so that $first \neq last$.

2.6. Implementing the Convex Hull Algorithm

The first step in the implementation is to choose an appropriate data structure for the convex hull vertices. From the discussion in the last section, such a data structure should keep the vertices in sorted angular order and allow for traversing the vertices in either direction. A doubly linked circular list is ideal for this purpose. At each node v we store the coordinates of the vertex it represents, along with pointers $pred(v)$ and $next(v)$ to the preceding and succeeding vertices in clockwise order around the convex hull. For simplicity, we use v to represent both the node and the point it represents. We need one additional pointer, *max*, that points to the last vertex inserted into the list. Figure 2.14 shows the data structure for the example in Figure 2.13.

We next convert the test of Theorem 2.8 into left and right turn tests. Let the new point be p and let v initially be set to the vertex pointed to by *max*. The test involves the four points p, v, $pred(v)$ and $next(v)$. Test (i) says that the vertices $pred(v)$ and $next(v)$ must lie in the

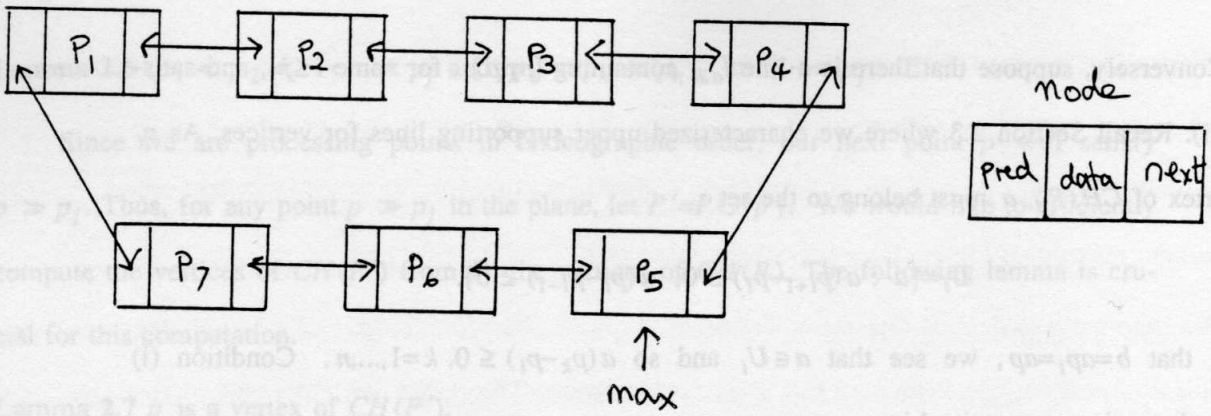

Figure 2.14

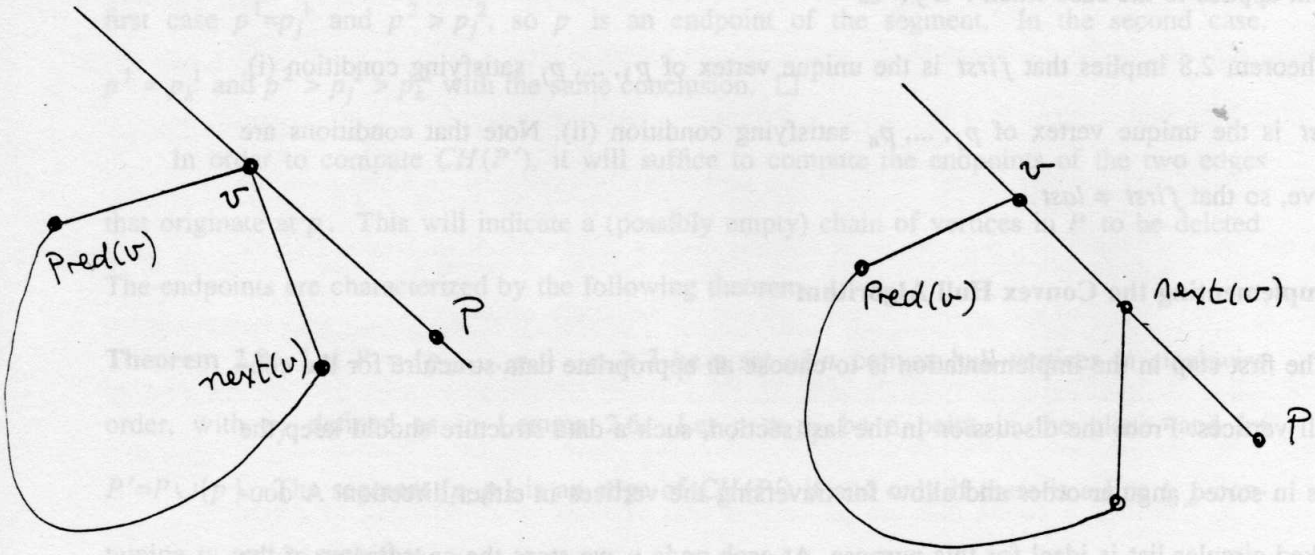

Figure 2.15

same halfplane bounded by the line through p and v. Furthermore, $pred(v)$ must lie in the open halfplane. The two favourable configurations are shown in Figure 2.15(a) and (b). In terms of left and right turns, condition (i) is satisfied if and only if

$$left(p, v, pred(v)) \textbf{ and not } right(p, v, next(v)) = \textbf{true}.$$

Case (ii) is handled analogously. The procedure *tangent*$(p, first, last, max)$ implements the test of Theorem 2.8.

```
procedure tangent(p, first, last, max);
    v:=max;
    while ( not left(p, v, pred(v) )) or right(p, v, next(v))
        do v:=pred(v);
    first:=v; v:=max;
    while ( not right(p, v, next(v) )) or left(p, v, pred(v))
        do v:=next(v);
    last:=v
    return;
```

Any vertices that are both successors of *first* and predecessors of *last* are deleted from the circular list, and p is inserted in this place. The above code can in fact be simplified: the second part of each "or" condition can be dropped (see exercise). Putting all of the pieces together we have the following algorithm to find the convex hull.

algorithm *convex hull*;

Step 1. (Sort)
Sort the n input points into increasing lexicographic order and store them in an array $p_1, \ldots, p_n$.

Step 2. (Initialize)
Find the first point p_i that is not collinear with p_1 and p_2. If no such point exists then return the convex hull $\{p_1, p_n\}$. Else, construct the circular linked list for p_1, p_{i-1}, p_i.

Step 3. (Update)
For each $j = i+1,\ldots,n$ call *tangent*$(p_j, first, last, p_{j-1})$. Update the circularly linked list by removing all vertices that are successors of *first* and *predecessors* of *last*. Insert p_j between *first* and *last*.

2.7. Exercises

2.1 (a) Show there do not exist 4 points in the plane, for which the distance between each pair of points is one.

(b) Show there do not exist 4 points in the plane, for which the distance between each

pair of points is either one or two.

2.2 (a) Let P be a set of n points in the plane. Let L be the set of upper supporting lines for P that each contains at least two points of P. Show that

$$CH(P) = \bigcap_{l \in L} l^{-}.$$

(b) Prove that the boundary of $CH(P)$ is a convex polygon.

2.3 Let P be a set of n points in the plane. Show that

$$CH(P) = \{ x : x = \sum_{i=1}^{n} \lambda_i p_i, \ \sum_{i=1}^{n} \lambda_i = 1, \lambda_i \geq 0, i = 1, ..., n \}$$

2.4 Show that every convex polygon has at least $\lceil \frac{n}{2} \rceil$ proper antipodal pairs. Construct examples for each even n to show the bound is tight.

2.5 (a) Write a short Pascal-like procedure $sign(x,y,z)$ to compute the sign of the determinant Δ_{xyz}. Try to minimize the number of multiplications.

(b) Write a procedure to implement $disjoint(a,b)$ using at most three calls to $sign(x,y,z)$.

(c) Write a procedure to implement the case statement of $merge(a,b)$ using one call to $sign(x,y,z)$.

2.6 Prove Lemma 2.6.

2.7 Prove that in the procedure $tangent(p, first, last, max)$ the second condition in each while statement can be dropped.