Chapter 6

OUTPUT PROCEDURES AND A SAMPLE OUTPUT

§6.1 Output procedures

We have seen, in the preceding chapter, that a k-connected component of figure F is detected when the cycle of its outer edge is closed. Recalling our discussion in Subsection 5.3.4, this event is detected in the procedure *closechain* when the condition $q \uparrow .whi = 0$ is satisfied. In *restricted surrounding*, this is sufficient information to tell us that a connected component of F—taken in isolation—has been completely disclosed. In *full surrounding* this is less of a major event unless the component in question is *maximal for surrounding*, ($sm[w_0] = 1$). Our ultimate step will be to include in *closechain* a procedure—called *outcy*—which transfers in an output file the contents of the *objrec* records of the connected components, and the *cyrec* records of the maximal strings. At the same time, that procedure must erase these records from the primary memory. The procedure *outcy* can be found in Appendix F, Sections 1 and 2.

Here, we are confronted with two problems:

(a) Records of type *objrec* and *cyrec* were created dynamically. So, they are accessed by pointers. However, pointers cannot be represented as such in the output file which can contain characters only. Thus, these pointers must be replaced by integer labels.

(b) In *full surrounding*, the *objrec* records corresponding to a connected component of F should be transferred to the output file as soon as that component is detected, in order to take full advantage of dynamic storage management. On the other hand, the *cyrec* records attached to that component should be transferred only when a maximal string containing them is found. Thus, in the meantime, we must maintain some additional information telling us, somehow, which *cyrec* records correspond to the *objrec* records already transferred to the output file.

Our solution to problem (a) is to define a dummy component in the *objrec* and *cyrec* records. This component will be called *num* and will be a nonnegative integer. Given a record r, it is numbered $r.num$, and any pointer pointing to r can be represented by $r.num$. Readily, we must take $r.num > 0$, and reserve the value 0 for the pointer NIL. Thus, given a record r containing a component p of pointer type, we have

$$\begin{aligned}(r.p)\uparrow.num &= 0 &&\text{if } r.p = NIL,\\ &= i > 0 &&\text{otherwise.}\end{aligned} \tag{6.1}$$

We wish to examine briefly how this numbering is implemented and used in the case of *objrec* records. At the time such a record r is created, we make the provisional assignment

$$r.num := 0 \tag{6.2}$$

Next, when a connected component of F is detected, its objects can be accessed by following the cycles of its edge: In each of these cycles, we start on the left edge of some object whose record is $acces\uparrow$, and we follow that cycle using the variables $fol0(1)point(side)$ introduced in Section 5.2, until we come back to the left edge of the initial object. With such a scan, every object is visited twice. The numbering of the *objrec* records occurs at the first visit. We use a counter variable nvp (initialized at 0). Any object accessed by pointer p is numbered by the following conditional statement, executed by the procedure *idobj* (see Appendix E, Sections 5 and 6).

$$
\begin{array}{l}
\texttt{IF } p\uparrow.num = 0 \texttt{ THEN} \\
\texttt{BEGIN} \\
nvp := nvp + 1; \\
p\uparrow.num := nvp \\
\texttt{END;}
\end{array}
\tag{6.3}
$$

Even with this numbering of records settled, we still need, in principle, to scan the very same objects again in order to write the contents of their records in the output buffer and to deallocate storage space. Clearly, it would be desirable not to perform the second scan by following the cycles another time, because this would be a waste of time.

As it frequently happens, we can gain on execution time at the expense of some additional extra-storage. Thus, we also define a dummy array *vp* of type *link* which is used to access directly the records visited during the first scan. Whenever a record is numbered i we let $vp[i]$ point to it. This can be combined with the code in (6.3), and gives rise to the modified statement:

$$
\begin{array}{l}
\texttt{IF } p\uparrow.num = 0 \texttt{ THEN} \\
\texttt{BEGIN} \\
nvp := nvp + 1; \\
vp[nvp] := p; \\
p\uparrow.num := nvp \\
\texttt{END;}
\end{array}
\tag{6.4}
$$

Then, the objects of any connected component can be accessed by scanning $vp[i]$ for $i := 1$ to *nvp*. One scan of *vp* activates the writing, in the output file, of the records contents. This is done by the procedure *outobj*, (see Appendix F, Sections 3 and 4). A second scan deallocates the corresponding storage space. This completes our solution to problem (a).

Next, let us go on with problem (b) and the output of maximal strings.

When the objects of a connected component are discarded from main memory, we must record the link between the data written in the output file and the *cyrec* record of the outer cycle of that component. When a *cyrec* record is created, it does not get the value $num = 0$ as it was the case for *objrec* records. We define instead a counter variable ncy (initialized at 0). When a new record $q \uparrow$ is created in *closechain*, ncy is increased by 1 and the assignment $q \uparrow .num := ncy$ is executed. Thus, the records are numbered 1, 2, ..., and it looks as if we were using a dynamic array of records.

When a connected component is taken away from the main memory, and the contents of *objrec* records written in the output, we write also the number $c.num$ of the outer cycle c of that component's edge. Then, when a maximal string is written in the output file, we do it by representing each cycle c by its number $c.num$, thereby preserving the link between connected components of F and cycles in maximal strings.

Eventually, let us briefly comment on the processing of edge-strings.

In *restricted surrounding*, upon completion of the detection of a connected component, its string of cycles is regarded as being maximal. So, all we can do is to count the number of holes in that component and write that information—together with the *objrec* records—in the output file, as this is all the topological information which is considered under this restriction. Simultaneously, we can erase the records of these cycles from the main memory.

In *full surrounding*, upon completion of the detection of a connected component, *objrec* records are written in the output file together with the number of the outer cycle of its edge as we just explained. However, *cyrec* records must be kept in main memory until the time that a maximal string is found in *closechain*. Then, a recursive procedure, called *interncy*, is activated (see Apendix F, Section 8). This procedure scans the string, writes the contents of the underlying neighborhood tree, and erases the *cyrec* records from the main memory.

Presently, it should be clear that these output procedures preserve all the information on connected components, adjacency trees, and surrounding relations that was patiently acquired with the methods of the previous chapters.

Let us now say a word about the output. The output file can contain the content of the records (produced by *outobj*), information for a graphic output (produced by a procedure called *outxy*) or any other information suitable for a particular application.

§6.2 Sample Output

In this section we wish to exhibit a sample output of the program for an illustrative example. The algorithm was applied to the (artificial) picture in Figure 6.1. 4-connectivity was chosen for the picture, constant d (for d-blocks) was set to 3, (but remained inoperative in this example), while values of 10 and 5 were chosen for *blen* and *clen* repectively.

Figure 6.1.a shows the image together with the numbering of cycles as described above. Recall that the number of the outer cycle of some connected component may be interpreted as the number of that component. Figure 6.1.b shows the decomposition of the image into blocks (B), hinges (H), and block-continuations (C and D). Also shown are the numbers assigned to every object—or *objrec* record—within their component as the lower left pixel(s) of that object.

In the first place, we examine the results obtained under *resticted adjacency* and *full surrounding*. Next, we consider *full adjacency* and *restricted surrounding*.

6.2.1 Restricted adjacency and full surrounding

Connected component #2 is the first to be completely disclosed, when the scan has reached pixel $p(19, 37)$. The five objects that make up that component are numbered counterclockwise, starting from the lowermost run which is a hinge ($ty = 0$). Hence, the following output:

```
object:  1-------------------------------------cycle :  2
fol0poin= 1       fol1poin=  2
fol0side= 1       fol1side=  1
ty=0      hro= 19     hbe= 33     hen= 37
object:  2-------------------------------------cycle :  2
fol0poin= 5       fol1poin=  3
fol0side= 1       fol1side=  1
ty=1      fr= 18      b= 36      e= 37      bll=  0
object:  3-------------------------------------cycle :  2
fol0poin= 5       fol1poin=  4
fol0side= 0       fol1side=  1
ty=0      hro= 17     hbe= 33     hen= 37
```

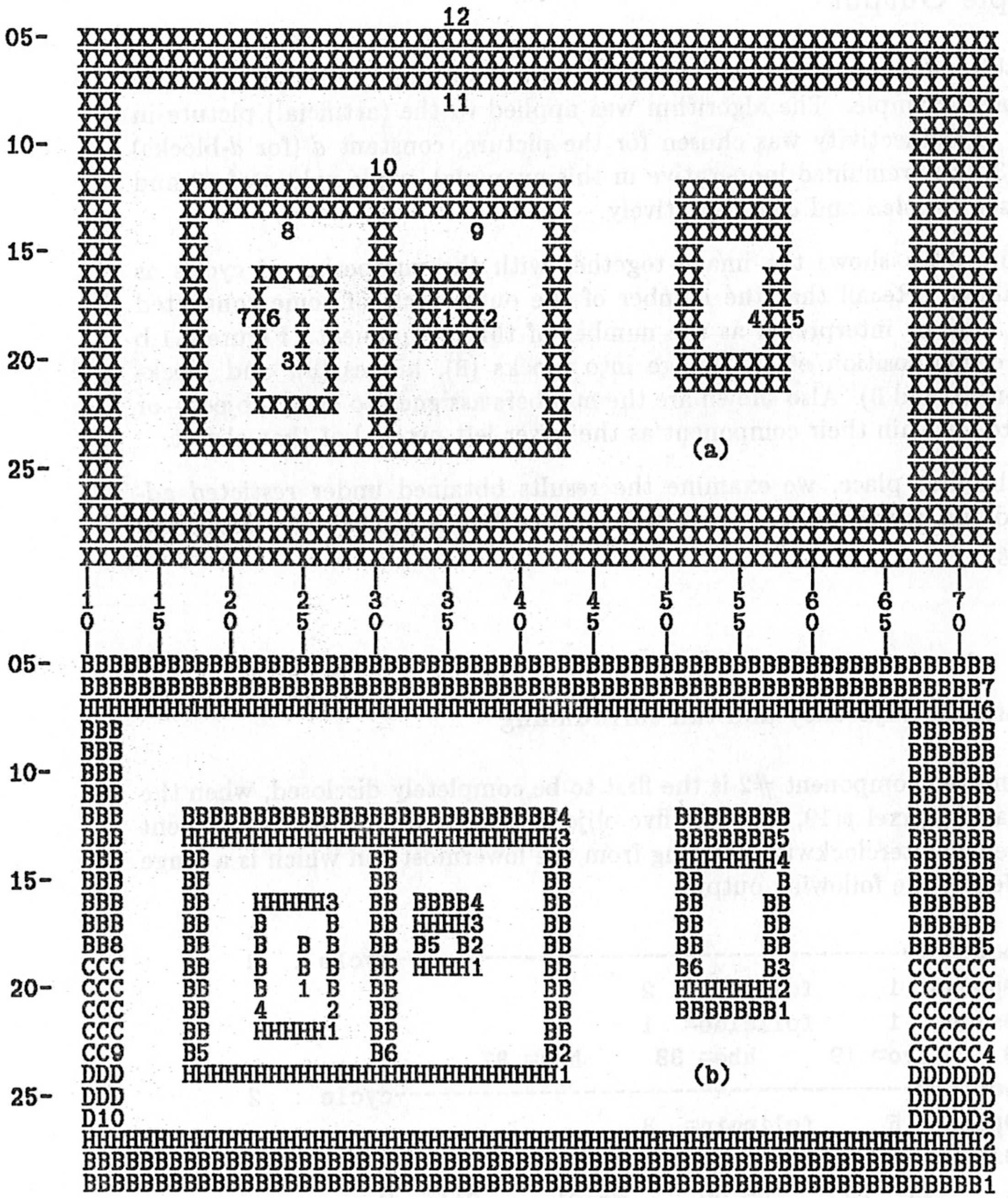

Figure 6.1. An image with
(a) the numbering of cycles (connected components),
(b) the numbering of objects in connected components.

```
object:  4-------------------------------------cycle :  2
fol0poin=  3       fol1poin=  4
fol0side=  0       fol1side=  0
ty=1      fr= 16      b= 33      e= 37      bll=  0
object:  5-------------------------------------cycle :  2
fol0poin=  1       fol1poin=  2
fol0side=  0       fol1side=  0
ty=1      fr= 18      b= 33      e= 34      bll=  0
```

One may note that pointers *fol0poin* and *fol1poin* have also been converted into integer labels in accordance with our discussion in Section 6.1.

Components #3, 5, and 7 are the next three ones to be output. For the sake of conciseness, their records are not reproduced here. For component #10, we get the following information:

```
object:  1-------------------------------------cycle : 10
fol0poin=  1       fol1poin=  2
fol0side=  1       fol1side=  1
ty=0      hro= 24      hbe= 17      hen=43
object:  2-------------------------------------cycle : 10
fol0poin=  6       fol1poin=  3
fol0side=  1       fol1side=  1
ty=1      fr= 14      b= 42      e= 43      bll= 9
 (0)(1)(2)(3)(4)(5)(6)(7)(8)(9)
  0  0  0  0  0  0  0  0  0  0      {blbedif}
  0  0  0  0  0  0  0  0  0  0      {blendif}
object:  3-------------------------------------cycle : 10
fol0poin=  5       fol1poin=  4
fol0side=  0       fol1side=  1
ty=0      hro= 13      hbe= 17      hen= 43
object:  4-------------------------------------cycle : 10
fol0poin=  3       fol1poin=  4
fol0side=  0       fol1side=  0
ty=1      fr= 12      b= 17      e= 43      bll=  0
```

```
object:  5-------------------------------------cycle : 10
fol0poin=  1      fol1poin=  6
fol0side=  0      fol1side=  0
ty=1      fr= 14     b= 17      e= 18      bll=  9
 (0)(1)(2)(3)(4)(5)(6)(7)(8)
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0
object:  6-------------------------------------cycle : 10
fol0poin=  5      fol1poin=  2
fol0side=  1      fol1side=  0
ty=1      fr= 14     b= 30      e= 31      bll=  9
 (0)(1)(2)(3)(4)(5)(6)(7)(8)
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0
```

Finally, connected component #12, comprises hinges, blocks, block continuations of maximal lenght (5), and two block-continuations of lenght less than five. We get the following information:

```
object:  1-------------------------------------cycle : 12
fol0poin=  1      fol1poin=  2
fol0side=  1      fol1side=  1
ty=1      fr= 28     b= 10      e= 72      bll=  1
 (0)
  0
  0
object:  2-------------------------------------cycle : 12
fol0poin=  1      fol1poin=  3
fol0side=  0      fol1side=  1
ty=0      hro= 27     hbe= 10     hen= 72
object:  3-------------------------------------cycle : 12
fol0poin= 10      fol1poin=  4
fol0side=  1      fol1side=  1
ty=2      ctl=  3
 (0)(1)(2)
  0  0  0                    {ctbedif}
  0  0  0                    {ctendif}
```

```
object:  4-------------------------------------cycle : 12
fol0poin=  3      fol1poin=  5
fol0side=  0      fol1side=  1
ty=2      ctl=  5
 (0)(1)(2)(3)(4)
  0  0  0  0  0
  0  0  0  0  0
object:  5-------------------------------------cycle : 12
fol0poin=  4      fol1poin=  6
fol0side=  0      fol1side=  1
ty=1      fr=  8     b= 67     e= 72     bll= 10
 (0)(1)(2)(3)(4)(5)(6)(7)(8)(9)
  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0
object:  6-------------------------------------cycle : 12
fol0poin=  8      fol1poin=  7
fol0side=  0      fol1side=  1
ty=0      hro=  7      hbe= 10      hen= 72
object:  7-------------------------------------cycle : 12
fol0poin=  6      fol1poin=  7
fol0side=  0      fol1side=  0
ty=1      fr=  5     b= 10     e= 72     bll=  1
 (0)
  0
  0
object:  8-------------------------------------cycle : 12
fol0poin=  9      fol1poin=  5
fol0side=  0      fol1side=  0
ty=1      fr=  8     b= 10     e= 12     bll= 10
 (0)(1)(2)(3)(4)(5)(6)(7)(8)(9)
  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0
object:  9-------------------------------------cycle : 12
fol0poin= 10      fol1poin=  8
fol0side=  0      fol1side=  1
ty=2      ctl=  5
 (0)(1)(2)(3)(4)
  0  0  0  0  0
  0  0  0  0  0
```

```
object: 10-------------------------------------cycle : 12
fol0poin=  2      fol1poin=  9
fol0side=  0      fol1side=  1
ty=2      ctl=  3
 (0)(1)(2)
  0  0  0
  0  0  0
```

The information output so far enable us to trace out adjacency relations between objects belonging to the same connected component. Surrounding relations between components (*viz.*, the edge-string) can be output only at the time that the last black pixel of the figure has been scanned. We get the following message:

```
MAXIMAL COMPONENT :  12
 12     --->  11 hole
 11     --->   5
  5     --->   4 hole
 11     --->  10
 10     --->   9 hole
  9     --->   2
  2     --->   1 hole
 10     --->   8 hole
  8     --->   7
  7     --->   6 hole
  6     --->   3
```

This completes the information collected under *restricted adjacency* and *full surrounding*

6.2.2 Full adjacency and restricted surrounding

Under *full adjacency*, we get more detailed *objrec* records. Under *restricted surrounding* the sole topological information is confined in the number of holes of a given connected component. For instance, at the time the last pixel $p(24,43)$ of connected component #10 has been scanned, we receive the following information:

```
object:  1-----------------------------------cycle : 10
precnnb= 3       succnnb= 0
prefi= 5      prela= 2      sucfi= 0      sucla= 0
preletori= 0   preritole= 0   sucletori= 0   sucritole= 0
fol0= 1      fol1= 2
ty=0      hro= 24      hbe= 17      hen= 43

object:  2-----------------------------------cycle : 10
precnnb= 1       succnnb= 1
prefi= 3      prela= 3      sucfi= 1      sucla= 1
preletori= 0   preritole= 6   sucletori= 0   sucritole= 6
fol0= 3      fol1= 2
ty=1      fr= 14      b= 42      e= 43      bll= 9
 (0)(1)(2)(3)(4)(5)(6)(7)(8)
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0

object:  3-----------------------------------cycle : 10
precnnb= 1       succnnb= 3
prefi= 4      prela= 4      sucfi= 5      sucla= 2
preletori= 0   preritole= 0   sucletori= 0   sucritole= 0
fol0= 2      fol1= 2
ty=0      hro= 13      hbe= 17      hen= 43

object:  4-----------------------------------cycle : 10
precnnb= 0       succnnb= 1
prefi= 0      prela= 0      sucfi= 3      sucla= 3
preletori= 0   preritole= 0   sucletori= 0   sucritole= 0
fol0= 2      fol1= 1
ty=1      fr= 12      b= 17      e= 43      bll= 0

object:  5-----------------------------------cycle : 10
precnnb= 1       succnnb= 1
prefi= 3      prela= 3      sucfi= 1      sucla= 1
preletori= 6   preritole= 0   sucletori= 6   sucritole= 0
fol0= 2      fol1= 3
ty=1      fr= 14      b= 17      e= 18      bll= 9
 (0)(1)(2)(3)(4)(5)(6)(7)(8)
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0
```

```
object:  6-----------------------------------cycle : 10
precnnb= 1       succnnb= 1
prefi= 3     prela= 3     sucfi= 1     sucla= 1
preletori= 2  preritole= 5  sucletori= 2  sucritole= 5
fol0= 3     fol1= 3
ty=1      fr= 14      b= 30      e= 31      bll= 9
 (0)(1)(2)(3)(4)(5)(6)(7)(8)
  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0
```

The final message is:

```
end of a component containing  2  holes
```