Equivalence Relations

R is an *equivalence relation* on a set M iff (read if and only if)

(i) R is reflexive: for every $p \in M$ we have pRp.

(ii) R is symmetric: for $p, q \in M$, if pRq, then also qRp.

(iii) R is transitive: for $p, q, r \in M$, if pRq and qRr, then also pRr.

Example:

For a given picture P, defined on a finite grid $\mathbb{G}$, and pixels $p, q \in \mathbb{G}$ let pRq iff $P(p) = P(q)$. We regard the (infinite) complement of $\mathbb{G}$ as consisting of pixels that all have the same value, so they belong to one of the components (the *background component*) of P.

Relation R is an equivalence relation on $\mathbb{Z}^2$; it is

reflexive:

symmetric:

transitive:

Equivalence classes:

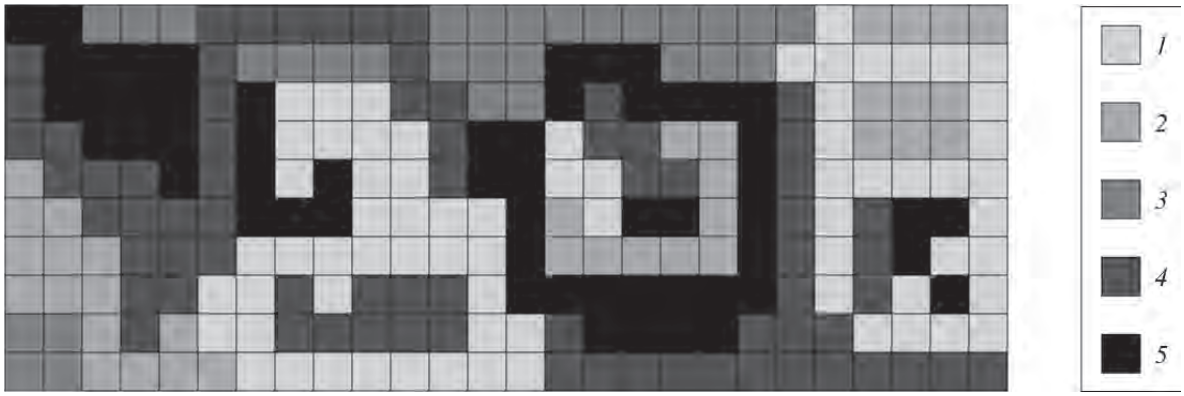
Any equivalence relation on a set M partitions M into pairwise disjoint *equivalence classes* $M(p) = \{q : \ q \in M \wedge qRp\}$, for $p \in M$.

Let P be a 2D multivalued picture defined on a finite grid $\mathbb{G}$ in which pixel p has value $P(p)$ in $\{0, 1, \ldots, G_{max}\}$.

We say that two pixels p and q are *P-equivalent* iff $P(p) = P(q)$.

Let M_u be the set of all $q \in \mathbb{G}$ such that $P(q) = u$.

If there exists $p \in \mathbb{G}$ such that $u = P(p)$, M_u is an equivalence class with respect to the relation of P-equivalence on $\mathbb{G}$ (in brief, a *P-equivalence class*).



A picture that has five P-equivalence classes; the numbers on the right are used to refer to the values defining these classes.

Let us assume 4-adjacency for all these P-equivalence classes M_1, M_2, M_3, M_4, and M_5.

For P we assume a background component with value 1. It follows that M_1 has two components (i.e., the background component of P and another component containing four pixels).

Class M_5 consists of six 4-components which define two *complementary 4-components* (a "hole" and the background 4-component of class M_5).

Component Labeling

Task which arises frequently in picture analysis and computer graphics (it is known as *labeling, filling,* or *region detection*):

Let the P-equivalence classes have a total of k components (including the background component).

The task is to assign k labels (e.g., integers or colors) to the pixels of P in such a way that all of the pixels in each component have the same label and pixels in different components have different labels.

To keep P unaltered, we can put the labels into an array of the same size as P.

FILL Procedure

Scan the picture until a pixel p is found that has not yet been labeled.

Suppose $P(p) = u$ and that labels $L_1, \ldots, L_{k-1}$ have already been used.

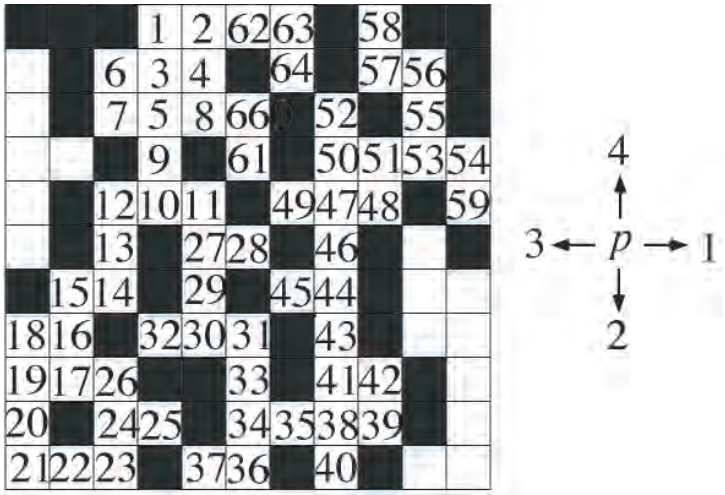
Choose a new label L_k, and call the procedure FILL(p, u, L_k) shown below:

1. Label p with L_k.
2. Put p into a stack.
3. If the stack is empty, stop.
4. Pop r out of the stack.
5. Label with L_k all pixels $q \in A(r)$ that have value u and have not yet been labeled, and put these qs into the stack.
6. Go to Step 3.

Note that the adjacency set $A(r)$ may depend on u (e.g., if we use 1-adjacency for 1*s* and 0-adjacency for 0*s*).

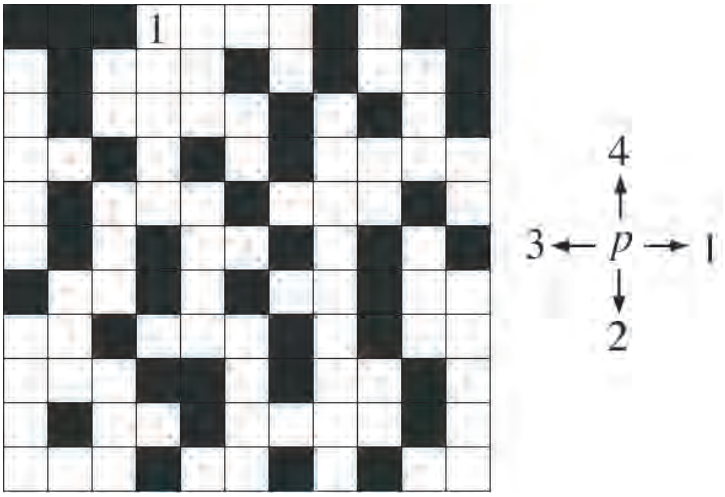
After labeling the component that contains p, continue scanning the picture until all of the pixels have been labeled. Make sure that all pixels in the background component of P have the same label (which way?).

The figure below shows the order of the pixel visits (assuming the order shown on the right is used for visiting 4-adjacent pixels) when this algorithm is used to label the white pixels.



The numbers show the order in which the pixels are labeled, assuming a *standard scan* of the grid (i.e., left to right, top to bottom).

Which order would result if the stack is replaced by a first-in-first-out queue?



The FILL algorithm requires that the stack can be as large as the total number of pixels in the given picture (normally not [anymore] a problem).

This FILL algorithm uses a depth-first strategy to visit all of the pixels in a component. The books

> W.M. Newman and R.F. Sproull. *Principles of Interactive Computer Graphics*, 2nd edition. McGraw Hill, New York, 1979.
>
> T. Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, Rockville, Maryland, 1982.

discuss variants of this strategy, such as recursive and nonrecursive (i.e., "filling by connectivity"). The time complexity of this strategy can be improved by stacking horizontal runs of pixels, see

> C. Ronse and P.A. Devijver. *Connected Components in Binary Images: The Detection Problem*. Wiley, New York, 1984.

In some versions of the FILL algorithm the stack is replaced by a first-in-first-out queue.

Rosenfeld-Pfaltz Labeling Algorithm

(published in 1966)

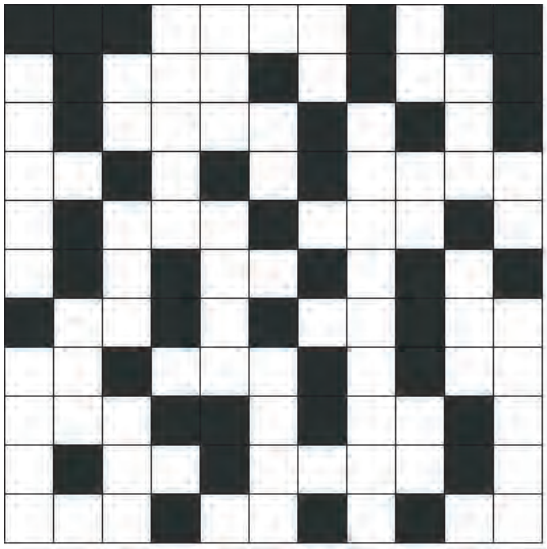
1. In the first scan, propagate the labels until the end of the picture is reached:
   (a) If the current pixel p is adjacent to one or more previously visited pixels that all have the same label, assign that label to p, and continue the scan.
   (b) If the current pixel p is adjacent to two or more previously visited pixels that have different labels, assign the smallest of those labels (e.g., L) to p, enter the other labels into the table as being equivalent to L, and continue the scan.
   (c) Otherwise, assign to p a label that has not yet been used, and continue the scan.
2. Determine the equivalence classes of the labels by computing the transitive closure of the equivalent pairs of labels detected in Step 1. Choose one label from each equivalence class as its pivot.
3. Scan the picture a second time, and replace every label with the pivot of its equivalence class.

First scan: we propagate smallest labels, and, whenever labels merge, we note this fact in a table of equivalent pairs of labels.

Second scan: we replace each label with a representative of its equivalence class.

Example

Background: assumed to be black



A	A	A	B	B	B	B	A	C	A	A
D	A	E	B	B	F	B	A	C	C	A
D	A	E	B	B	B	A	G	A	C	A
D	D	A	B	H	B	A	G	G	C	C
D	A	I	B	B	A	J	G	G	K	C
D	A	I	L	B	B	A	G	K	M	A
A	N	I	L	B	A	O	G	K	M	M
P	N	L	Q	B	B	A	G	K	M	M
P	N	N	L	L	B	A	G	G	K	M
P	R	N	N	L	B	B	B	B	K	M
P	P	N	A	S	B	A	B	A	T	M

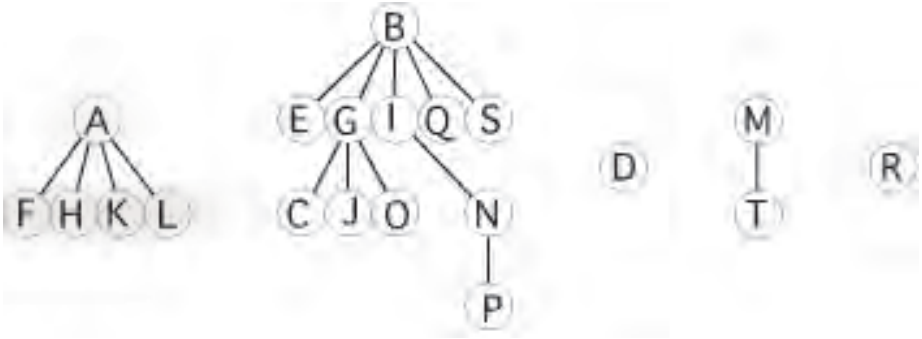
In the label propagation step of the algorithm, we use 4-adjacency for white pixels and 8-adjacency for black pixels.

At the end of the first scan, we have the following equivalence table:

Label	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
Equivalent					B	A	C	A	B		A	A		I	G	N	B		B	M
labels					B		B				A					N				
																N				

Label *A* is also the label of all of the pixels of the background component. Note that some equivalences are detected more than once.

The equivalences between pairs of the 20 labels $A, \ldots, T$ can be shown in the form of trees:



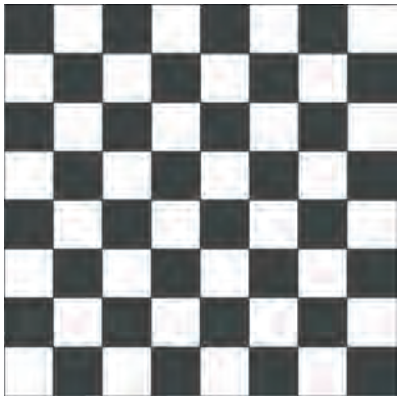
Standard graph traversal algorithms (e.g., depth-first, breadth-first) can be used to find the labels equivalent to a given label.

Alternatively, we scan the labels in order, smallest first. Any label that is equivalent to A has pivot A and can be marked and replaced with A. We next examine the smallest label that has not yet been marked; this label must be the pivot of its equivalence class so all of the labels equivalent to it can be marked and replaced with it.

We will use the smallest label in each equivalence class as the pivot of that class:

Label	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
Pivot	A	B	B	D	B	A	B	A	B	B	A	A	M	B	B	B	B	R	B	M

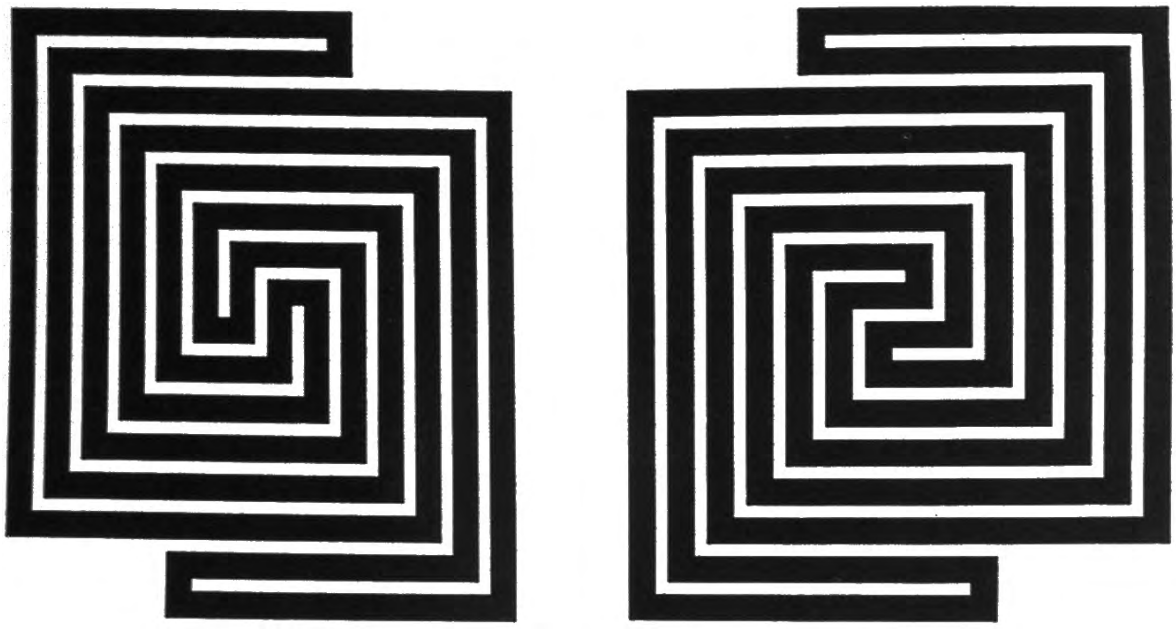
A binary picture of size $m \times n$ in which 0*s* and 1*s* alternate ("a chessboard") requires $\mathcal{O}(mn)$ labels.



(How many exactly, if we use 4-adjacency for white pixels, 8-adjacency for black pixels, and assume a black background?)

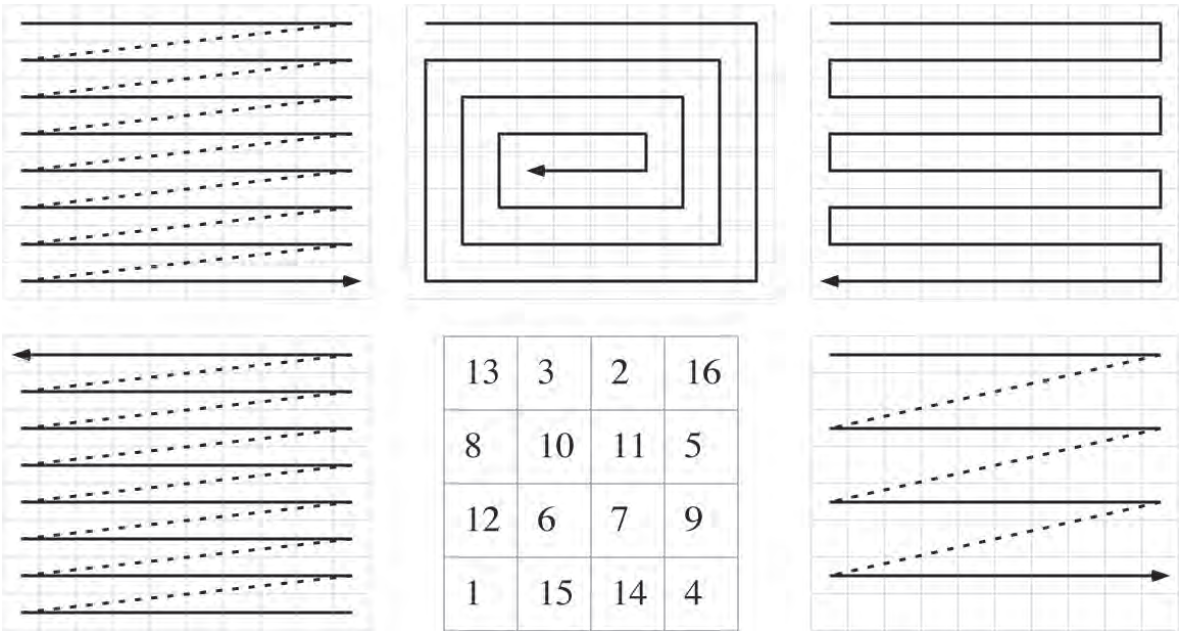
An a priori threshold on the number of labels can be used to limit the size of the equivalence table.

In one of the following binary pictures, the black pixels are connected, and in the other one, there are two 4-components of black pixels; can you tell which is which?



Scan Orders

The basic control structure of a picture analysis program (e.g., for component labeling) typically specifies a scan order for visiting all, or some of the pixels.



Scan orders: standard (upper left), inward spiral (upper middle), meander (upper right), reverse standard (lower left), magic square (lower middle), and selective, as used in interlaced scanning: standard, every second row (lower right).

How would be the impact of non-standard scans onto the FILL or Rosenfeld-Pfaltz labeling algorithm?

Coursework

Related material in textbook: Sections 1.3, 2.2.1, and 2.2.3.

A.3. [6 marks] Do exercise 3 on page 31, with $n_0 = 10$. After implementing this Hilbert scan, generate gray-level pictures P of size $2^n \times 2^n$ ($n = 8, 9, 10$) with random gray levels in the range 0 to 255 (hint: call a system function RANDOM for each visited pixel), and process these random pictures as follows, using your implemented Hilbert scan as control structure:

(i) at each pixel p, calculate the mean MEAN(p) of the 5×5 window centered at p, and replace the previous value $P(p)$ by MEAN(p). Visualize the sequence of value replacements (following the Hilbert scan) by slowing down your program such that the order of replacements can be followed by an observer;

(ii) do the same using MEDIAN(p) instead of MEAN(p), where the median of m numbers is that number which would be at the middle position if all m numbers would have been sorted (m assumed to be odd).

You may also use gray-level pictures of your choice instead of these generated (purely random) pictures (hint: alter gray levels of a given picture at a specified rate, by adding noise of random amplitude at random positions; you may discuss the use of uniform or Gaussian noise).

Appendix: Hilbert Scan

A recursive procedure for the Hilbert scan was described by L. M. Goldschlager in 1981, generalized by W. Skarbek in 1992 to more variants of picture scans.

Figure 1.8 in the textbook specifies the Hilbert scan in a way that we enter the picture at its north-west corner, and we leave it at its north-east corner. Let us denote the four corners of a $2^k \times 2^k$ picture by a, b, c, d, starting at the north-west corner and in clock-wise order. In the textbook we assume a Hilbert scan $H_k(a, d, c, b)$, where we start at corner a, continue then with corner d, proceed to corner c, and terminate at corner b.

$H_0(a, b, c, d)$ is a scan of a 1×1 picture, where we just visit the only pixel in this picture (and the order a, b, c, d can be replaced by any permutation of these four letters).

$H_{k+1}(a, d, c, b)$ is a scan where we start at the north-west corner, perform $H_k(a, b, c, d)$, followed by one step down, then $H_k(a, d, c, b)$, followed by one step to the right, then (again) $H_k(a, d, c, b)$, followed by one step up, and finally $H_k(c, d, a, b)$ (which takes us to the north-east corner of the $2^{k+1} \times 2^{k+1}$ picture.