暑期課程
基本影像處理 day6-2

指導教授：顏淑惠、林慧珍
http://163.13.127.10
http://pria.cs.tku.edu.tw

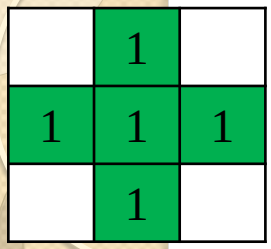
指導教授：涂瀞珽
http://mail.tku.edu.tw/cttu

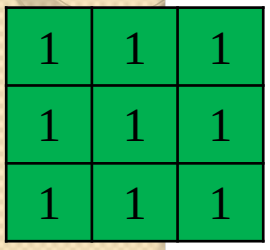
2015.07.08

Course Outline

- 型態影像學
  - Dilation (膨脹)
  - Erosion (侵蝕)
  - Opening and Closing

- Connected Component Labeling (連通元件標記法)
  - Flood fill
  - Two-pass

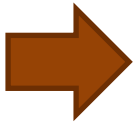




- Connected Component Labeling
  - 對4-connected或8-connected的foreground pixel標示相同的Label，形成連通元件(connected component)
  - 未連接則以不同label標示

1	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	1
1	1	0	1	1	0	0	1	1
0	0	0	1	1	1	0	0	1
0	0	0	1	1	1	0	0	0
1	1	0	0	1	1	0	1	1
0	1	1	0	0	1	1	0	1
0	1	1	0	0	0	0	0	1

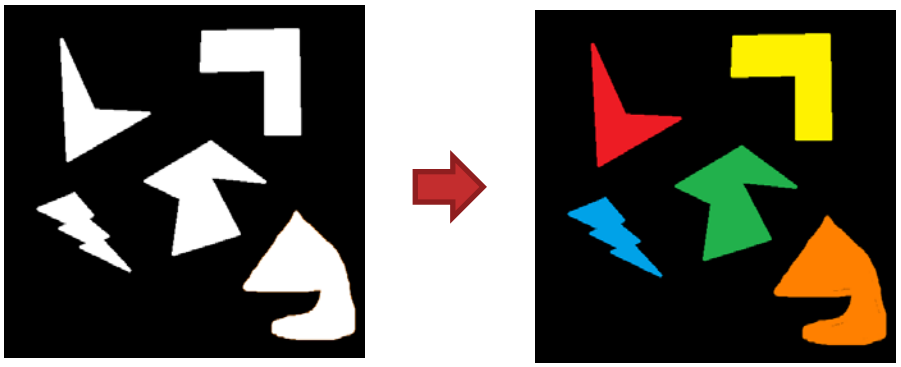
binary map

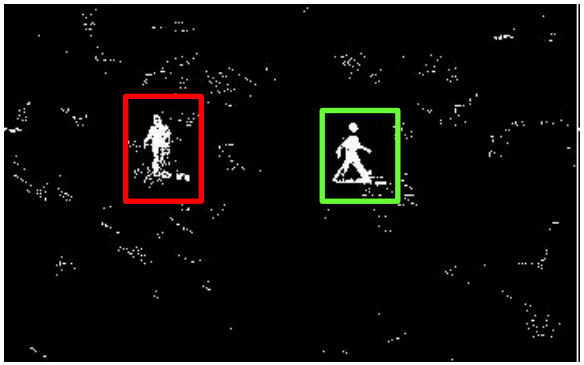


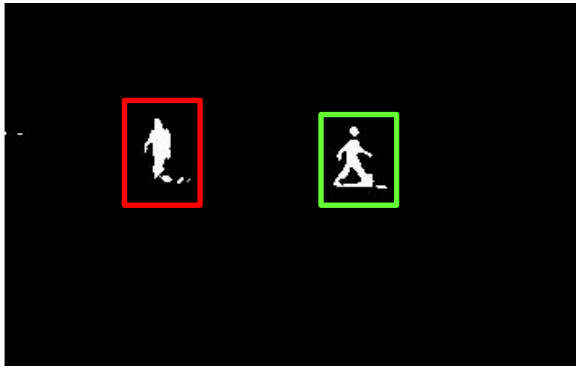
1	0	0	0	0	0	0	0	2
1	1	0	0	0	0	0	0	2
1	1	0	3	3	0	0	2	2
0	0	0	3	3	3	0	0	2
0	0	0	3	3	3	0	0	0
4	4	0	0	3	3	0	5	5
0	4	4	0	0	3	3	0	5
0	4	4	0	0	0	0	0	5

4-connected component map

- Why Connected Component Labeling?



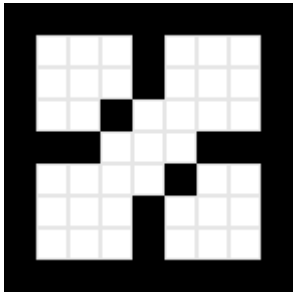




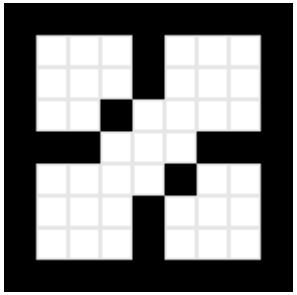
Algorithm

- Flood fill
- Two-pass

- Flood fill
  - Recursive implementation
  - drawback:
    - May cause stack overflow while running on a large map. (recursive function goes too deep)



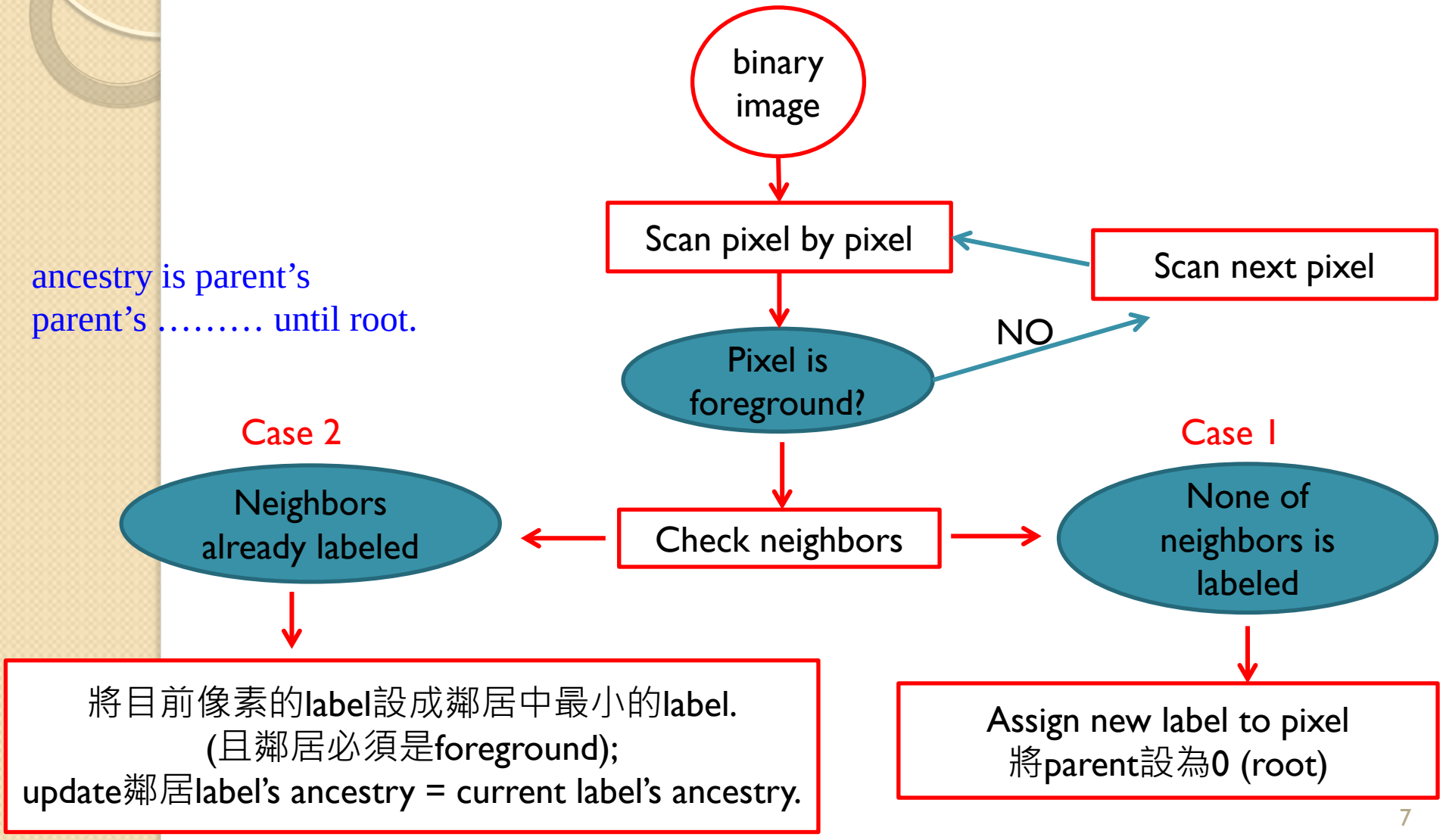
Recursive flood fill with 4 directions



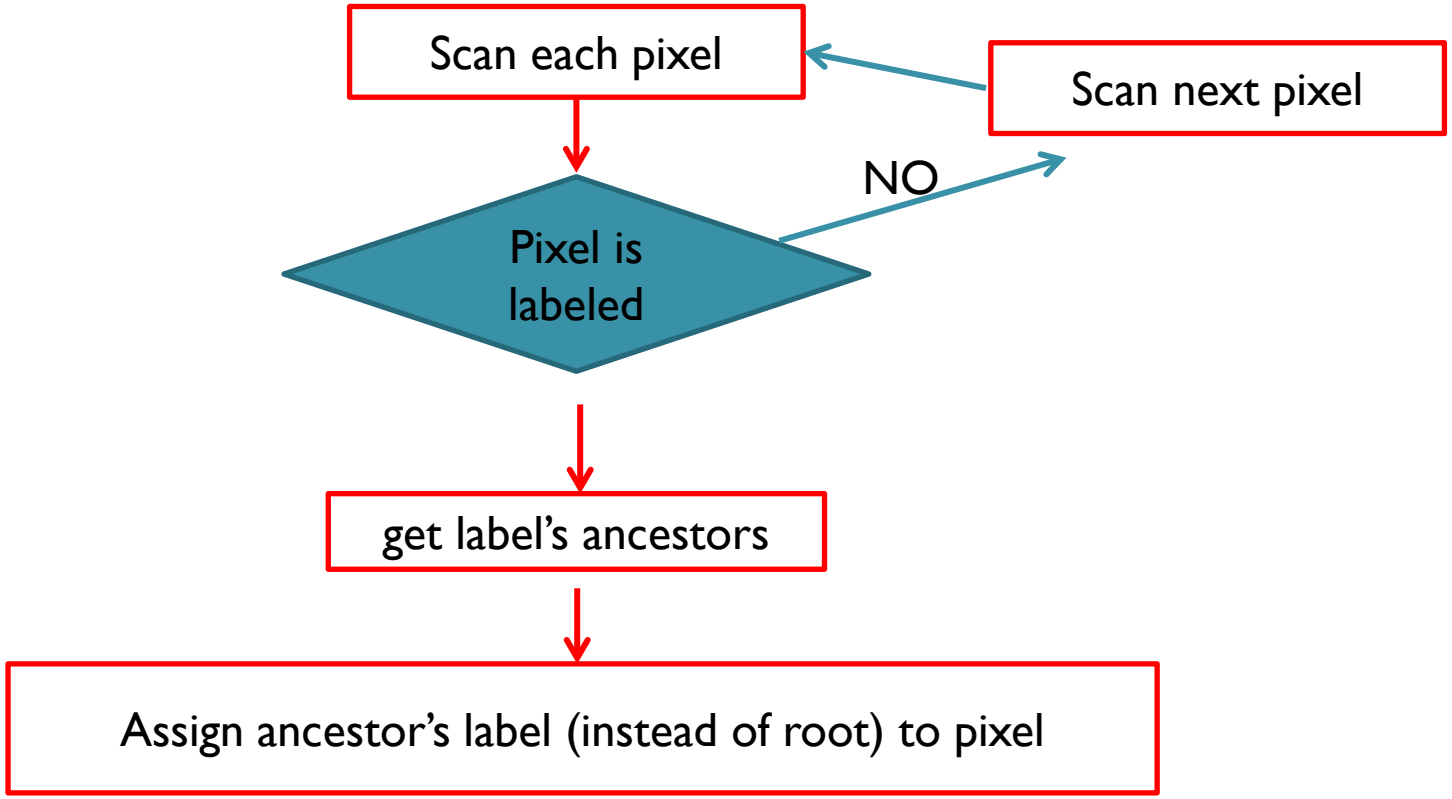
Recursive flood fill with 8 directions

```
void flood_fill(int x, int y, int color) {
        area[x][y]=color;
        if(x>0&&area[x-1][y]==0) flood_fill(x-1,y,color);
        if(y>0&&area[x][y-1]==0) flood_fill(x,y-1,color);
        if(x<MAX_X&&area[x+1][y]==0) flood_fill(x+1,y,color);
        if(y<MAX_Y&&area[x][y+1]==0) flood_fill(x,y+1,color);
}
```

- Two-pass Algorithm
  - First pass (assigning labels)



- Two-pass Algorithm
  - Second pass (aggregation)



ancestry or ancestor is parent's parent's …… until root.

Two-pass Algorithm

- ancestor(x, parent);

//parent array

ancestor label

Find the parent label of a set.
X is a label of the set.
PARENT is the array containing the union-find data structure.

```
procedure find(X, PARENT);
{
j := X;                          //j = x;
while PARENT[j] <> 0             //while parent[j] != 0;  //find parent until root
   j := PARENT[j];               //    j = parent[j];
return(j);
}
```

Algorithm 2: Find

Two-pass Algorithm

- update_ancestor(x, y, parent)

Construct the union of two sets.
X is the label of the first set.
Y is the label of the second set.
PARENT is the array containing the union-find data structure.

```
procedure union(X, Y, PARENT);
{
j := X;
k := Y;
while PARENT[j] <> 0      //find parent of set 1 until root
   j := PARENT[j];
while PARENT[k] <> 0      //find parent of set 2 until root
   k := PARENT[k];
if j <> k then PARENT[k] := j;     //if( j != k ) parent[k] = j;
}                                  //將某一subtree接到另一個subtree之下
```

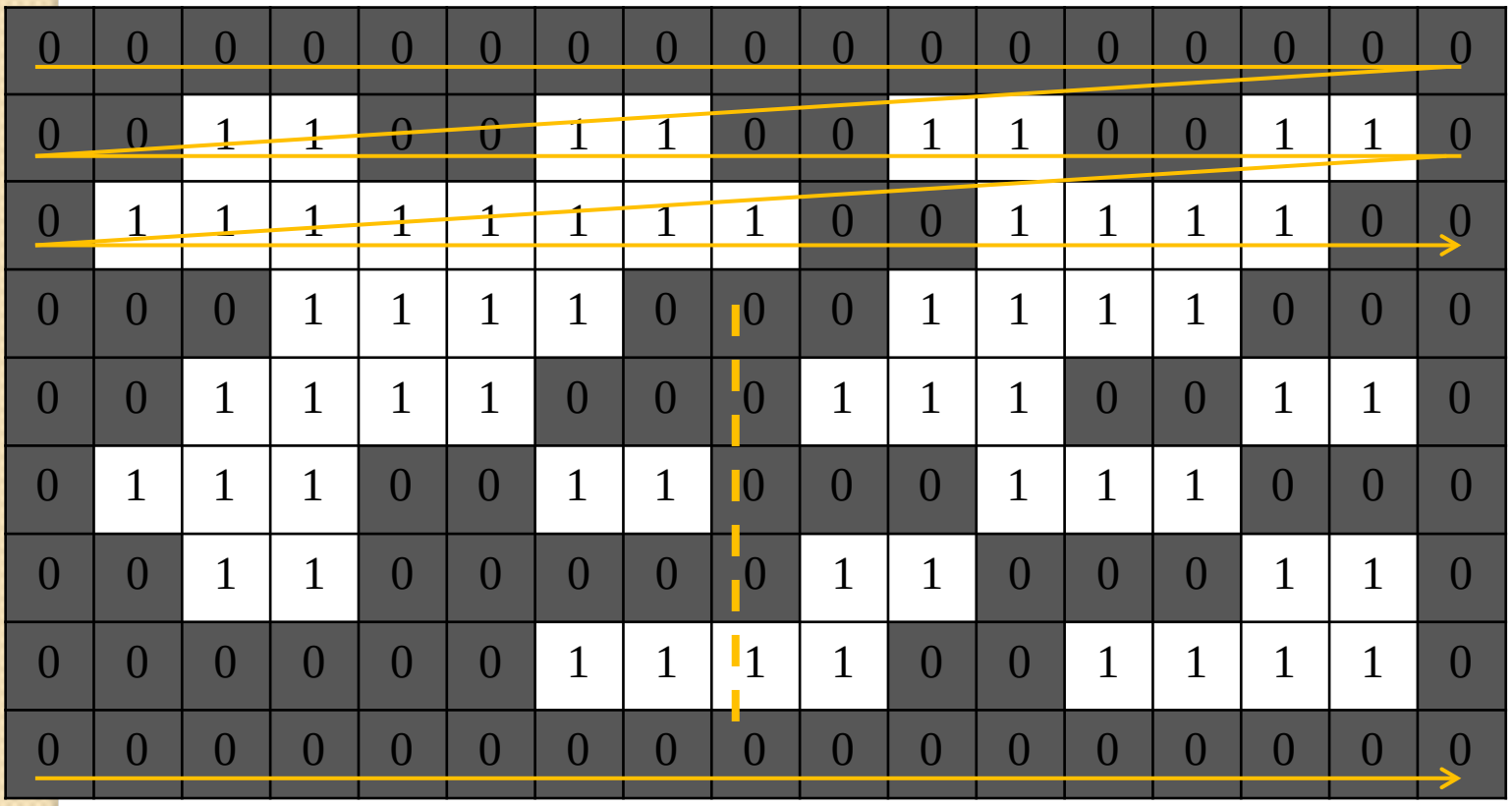
Algorithm 3: Union

Example

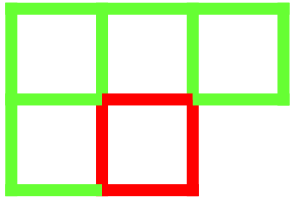
- First pass (assigning labels)

input: binary map

row-wise scanning



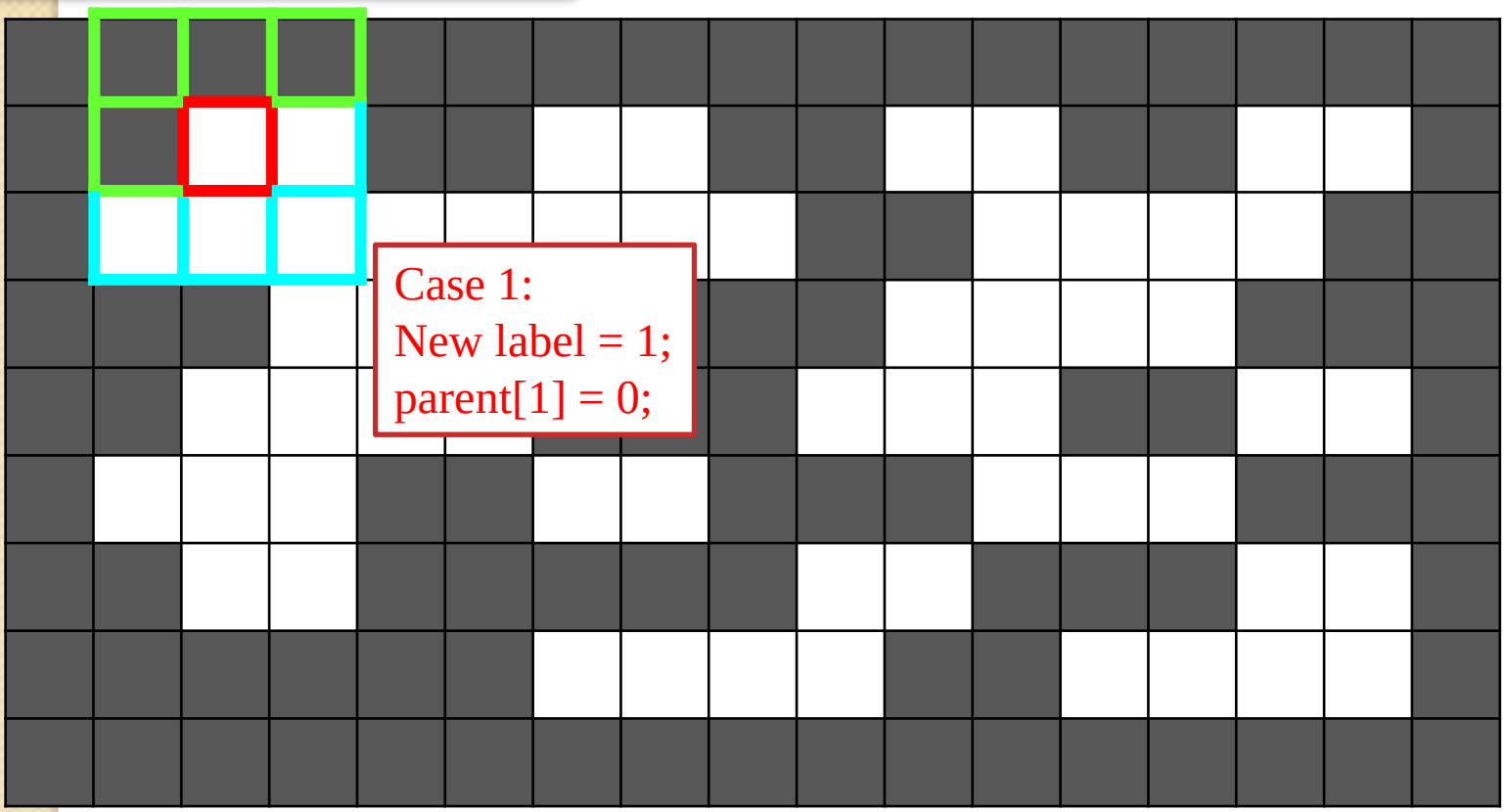
Example



step1 8-connected

Label							
parent							

start with currentLabelCount = 1



Case 1:
New label = 1;
parent[1] = 0;

Example

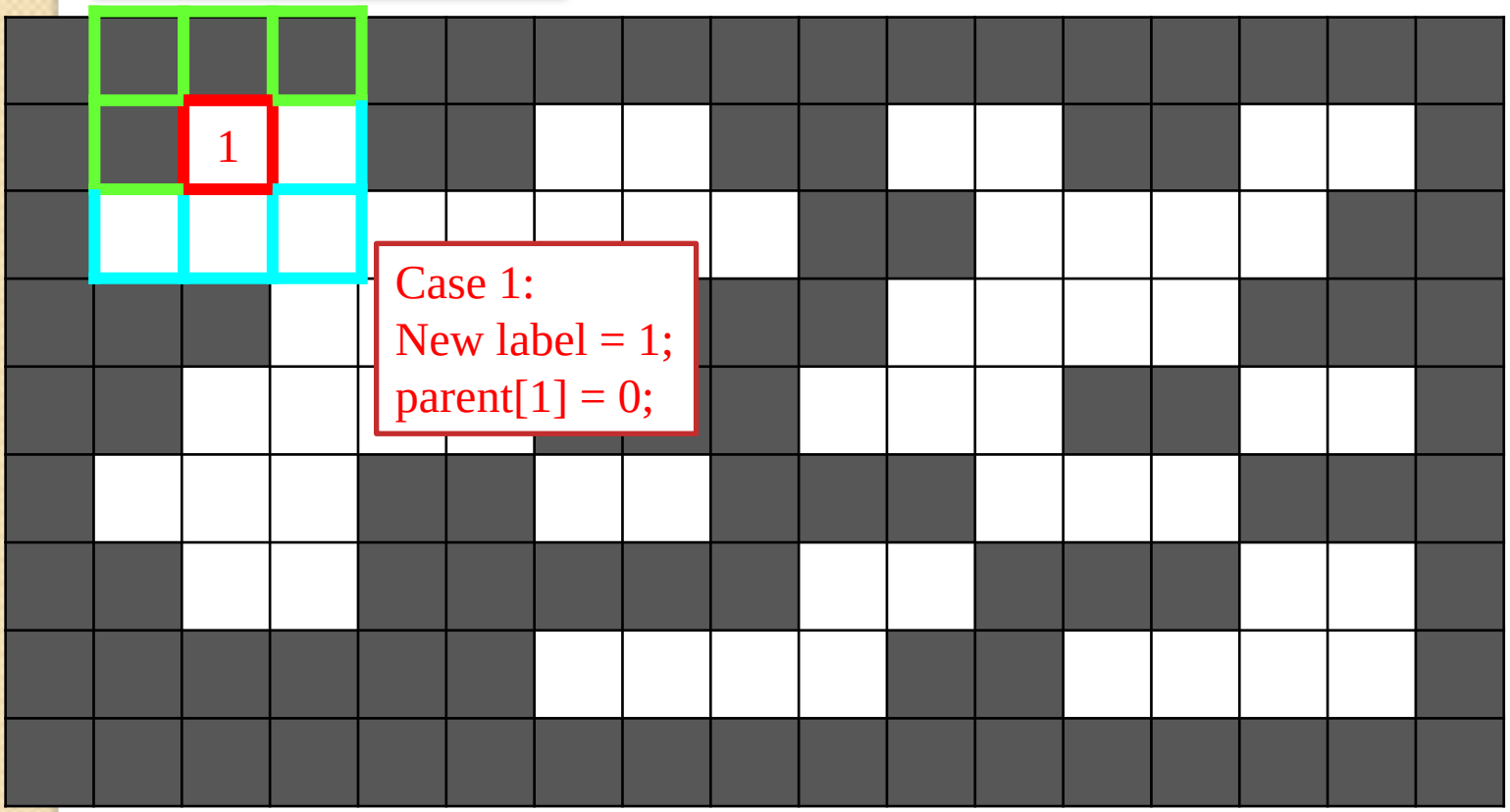
root

1

step1

Label	1						
parent	0						

currentLabelCount = 1



Example

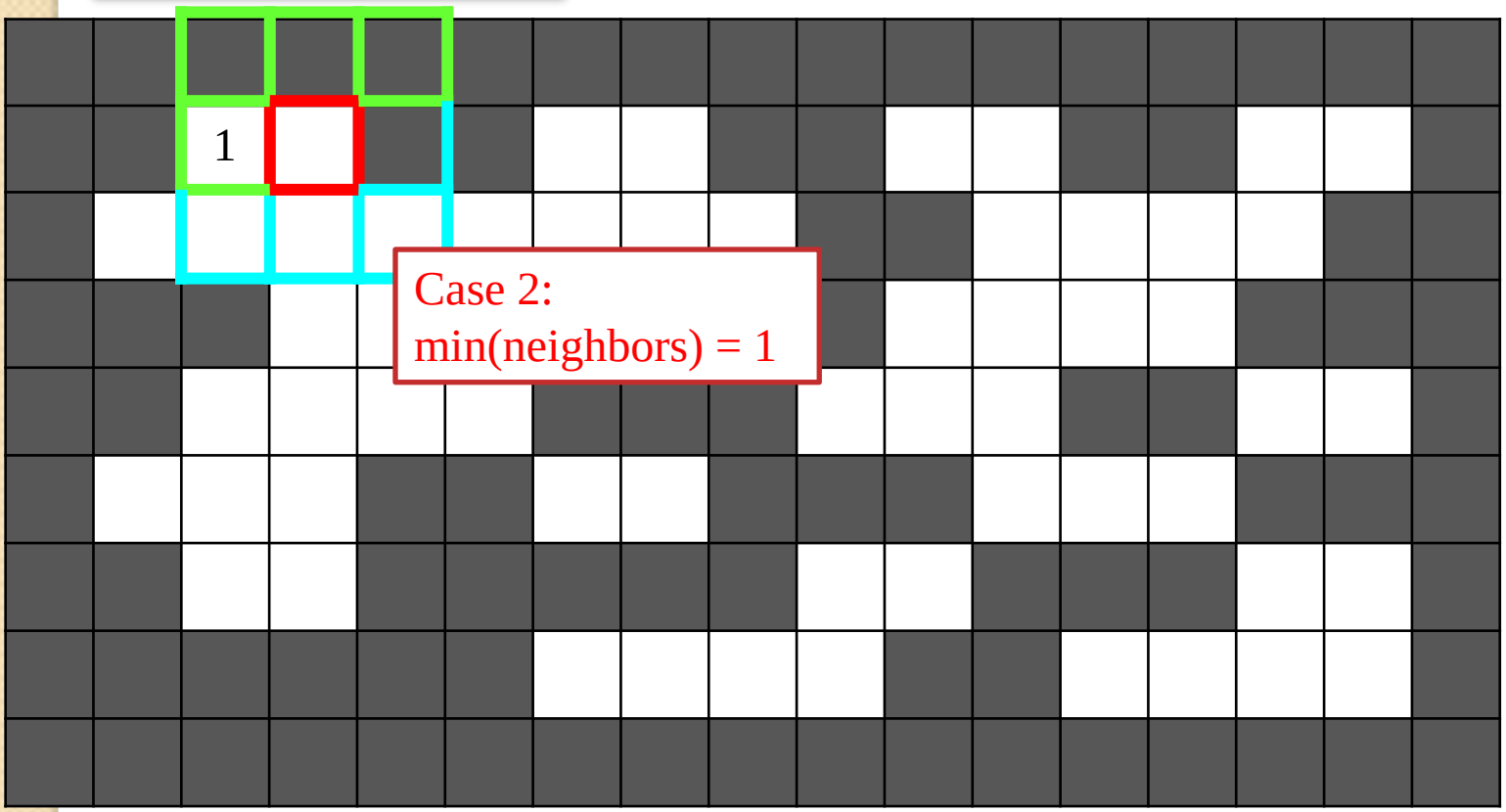
root

1

step2

Label	1						
parent	0						

currentLabelCount = 1



Example

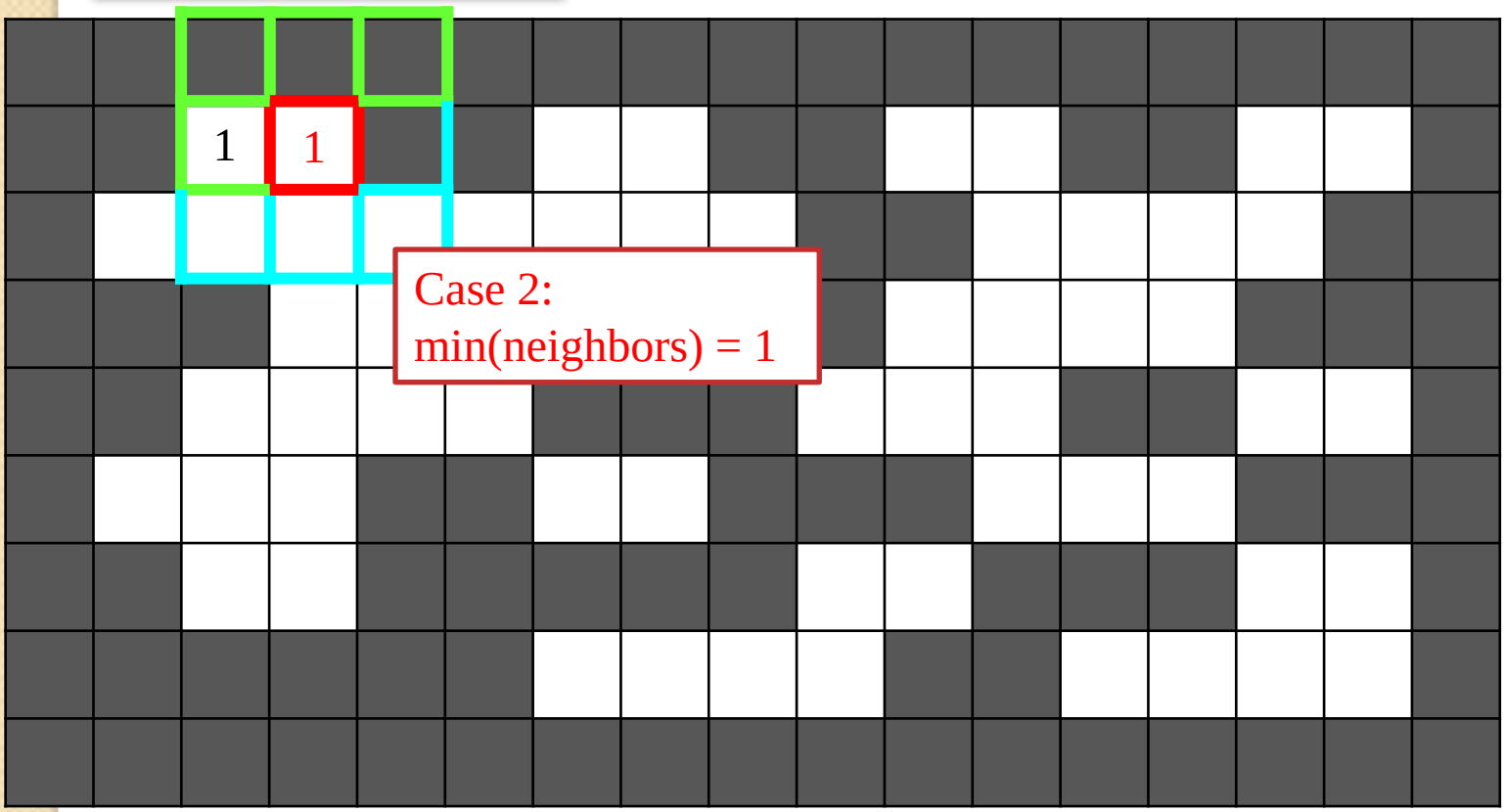
root

1

step2

Label	1						
parent	0						

currentLabelCount = 1



Example

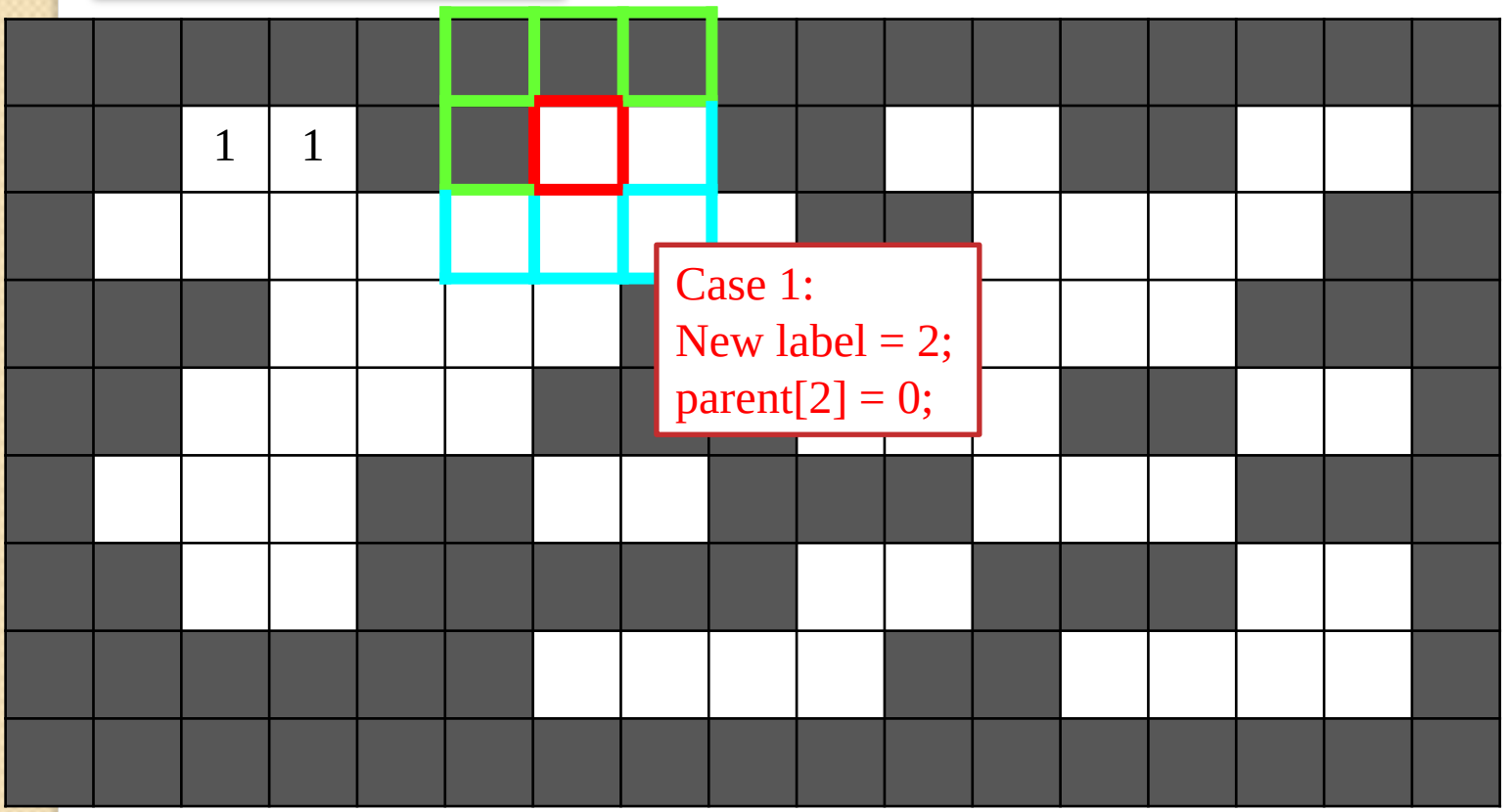
root

1

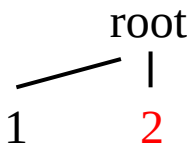
step3

Label	1						
parent	0						

currentLabelCount = 2



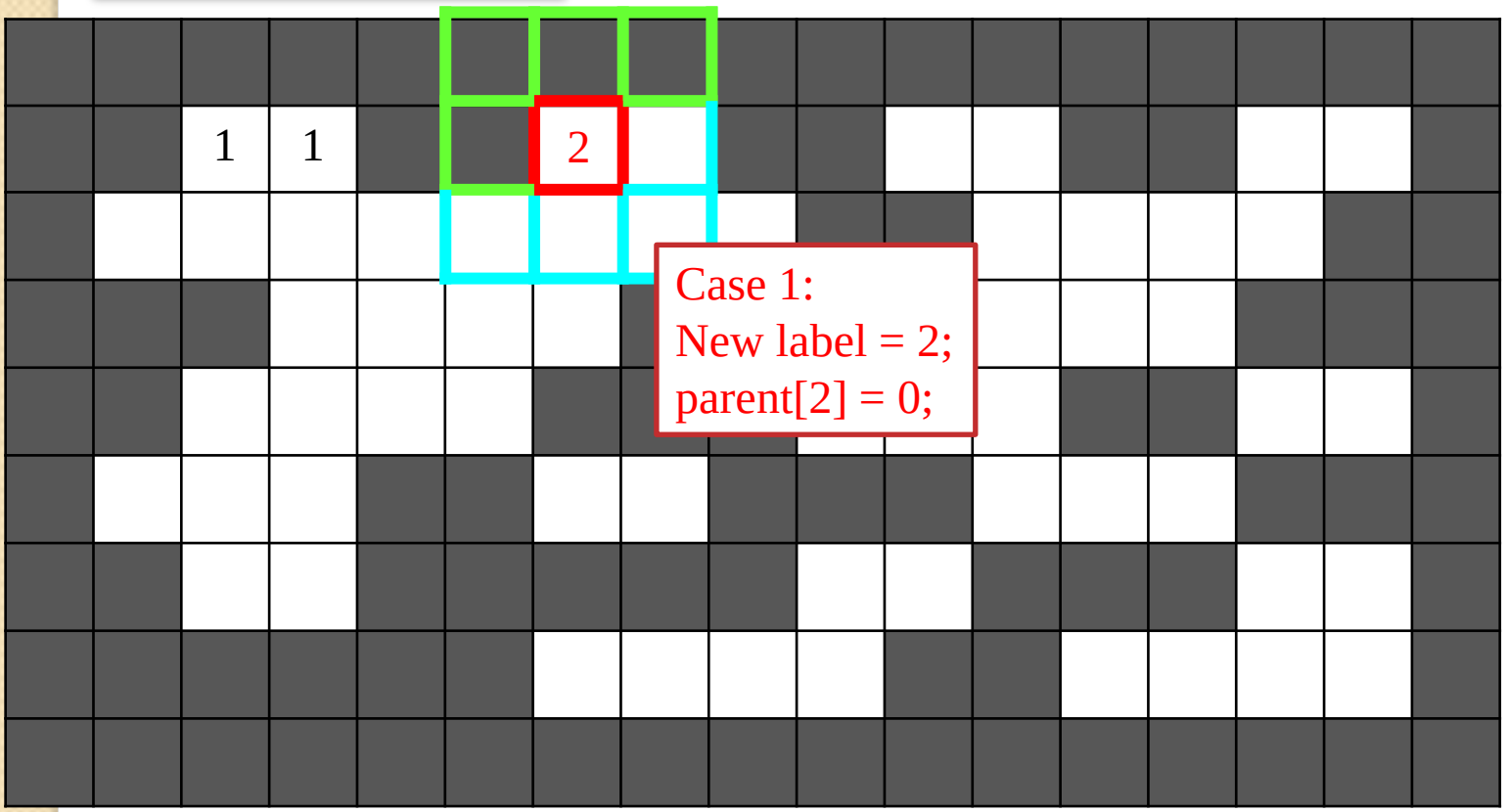
Example



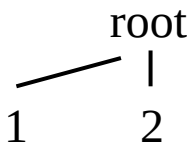
step3

Label	1	2					
parent	0	0					

currentLabelCount = 2



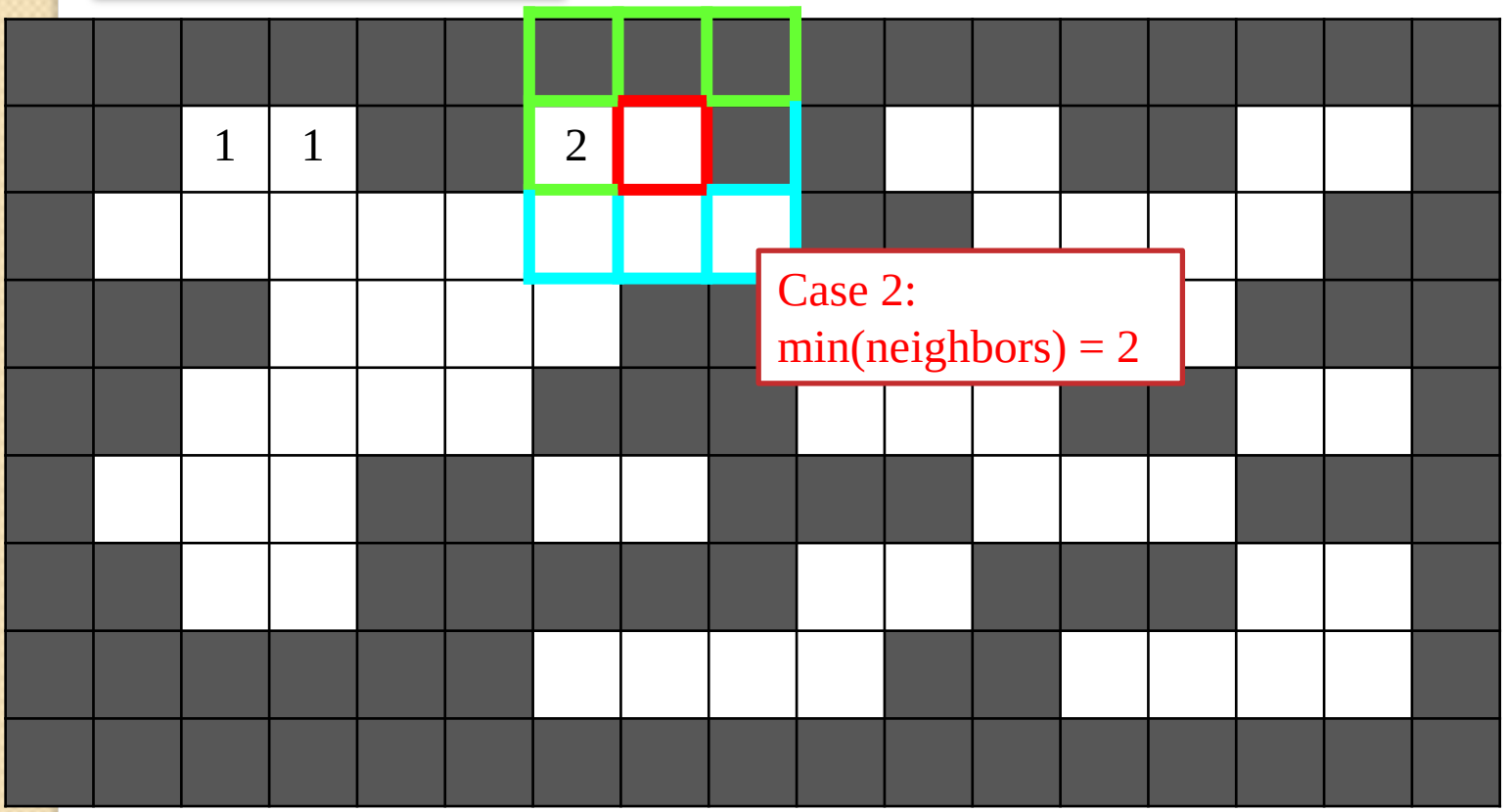
Example



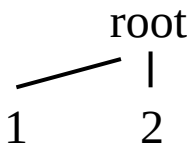
step4

currentLabelCount = 2

Label	1	2					
parent	0	0					



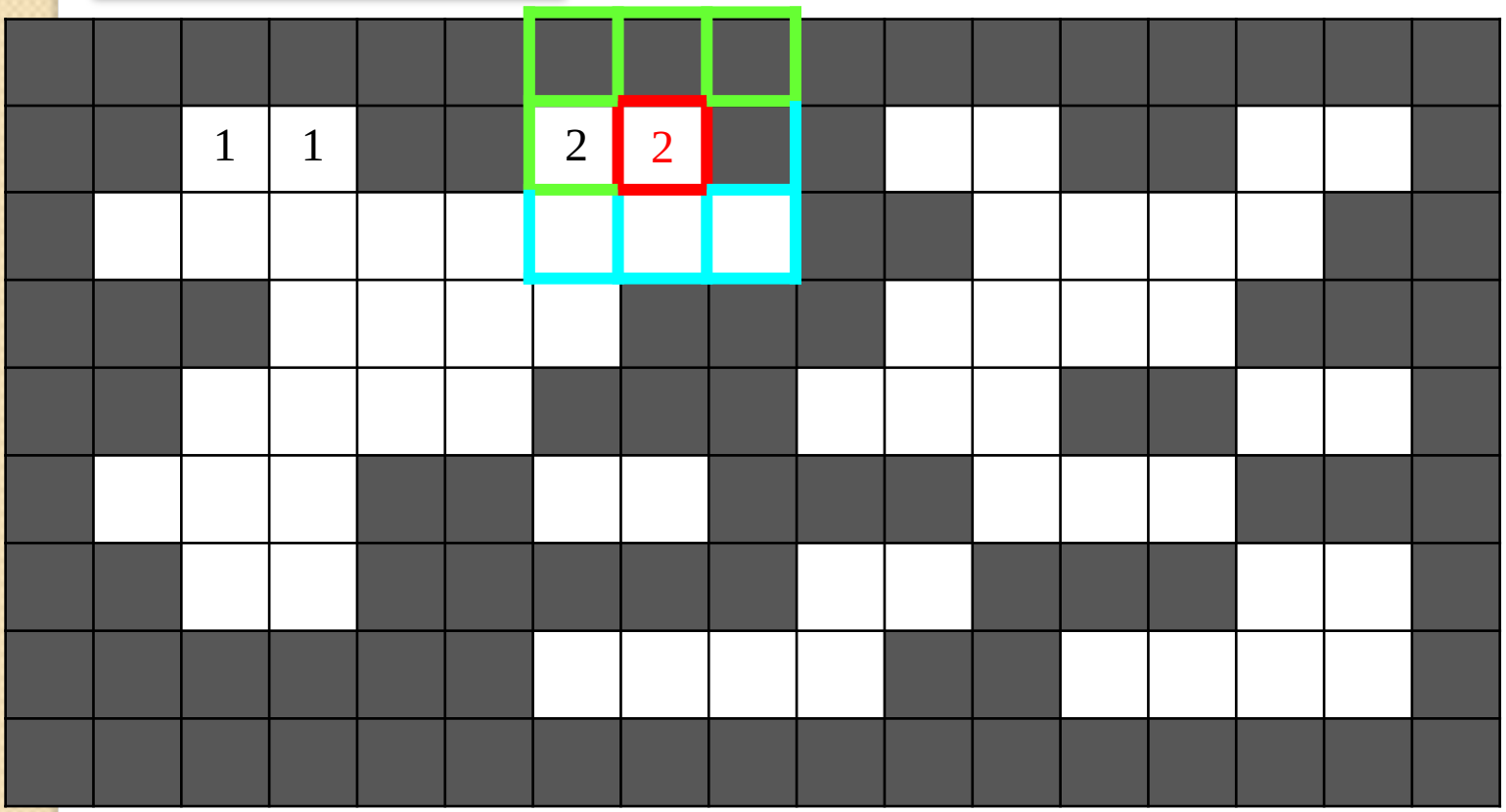
Example



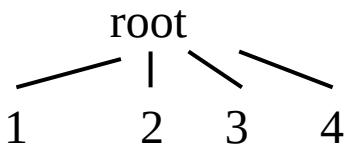
step4

currentLabelCount = 2

Label	1	2					
parent	0	0					



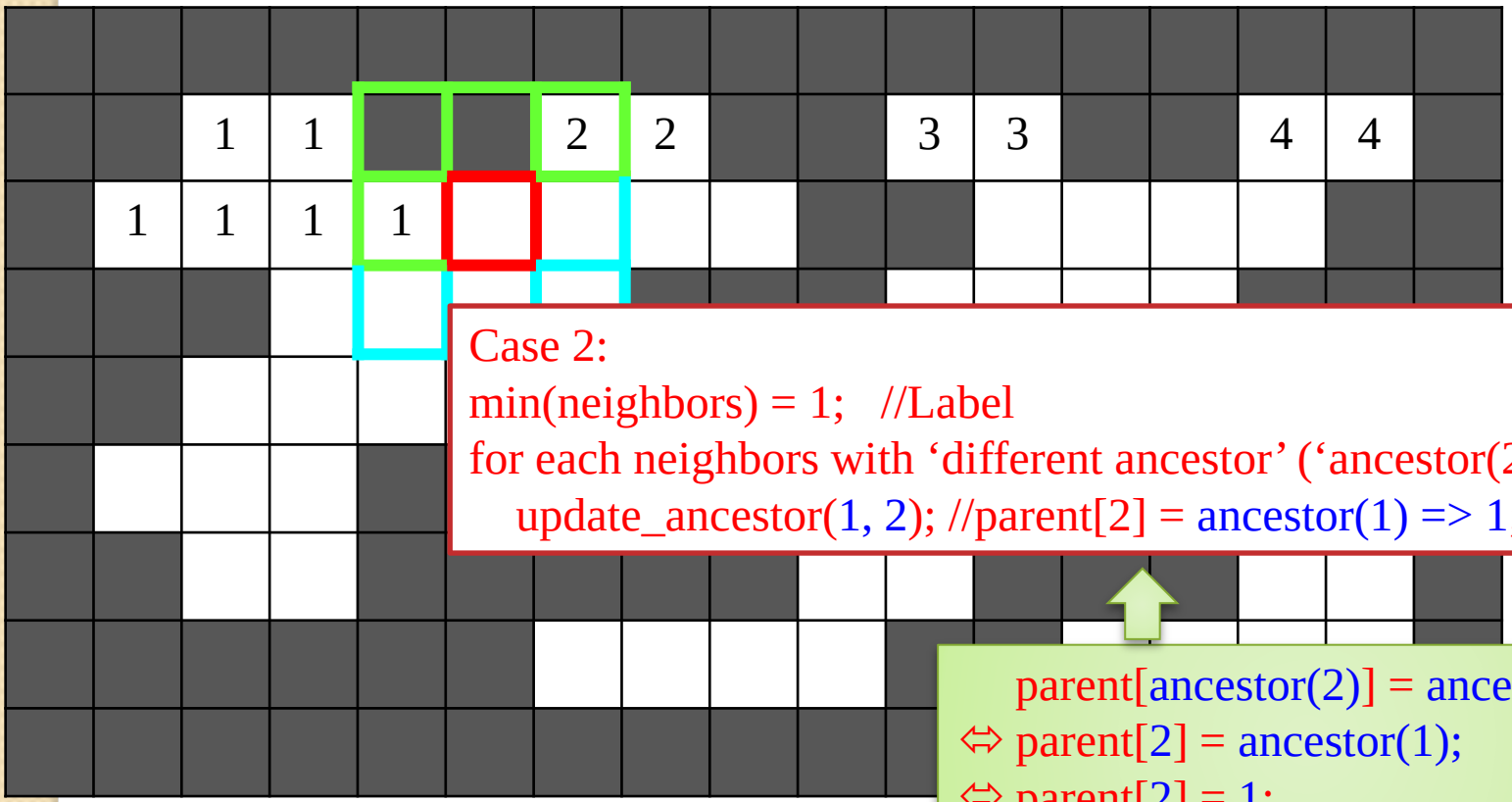
Example



step13

currentLabelCount = 4

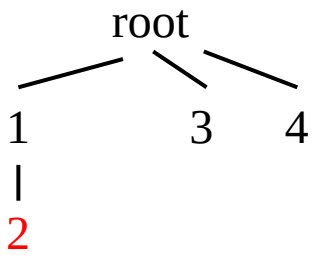
Label	1	2	3	4			
parent	0	0	0	0			



Case 2:
min(neighbors) = 1; //Label
for each neighbors with 'different ancestor' ('ancestor(2)' here)
update_ancestor(1, 2); //parent[2] = ancestor(1) => 1;

parent[ancestor(2)] = ancestor(1);
⇔ parent[2] = ancestor(1);
⇔ parent[2] = 1;

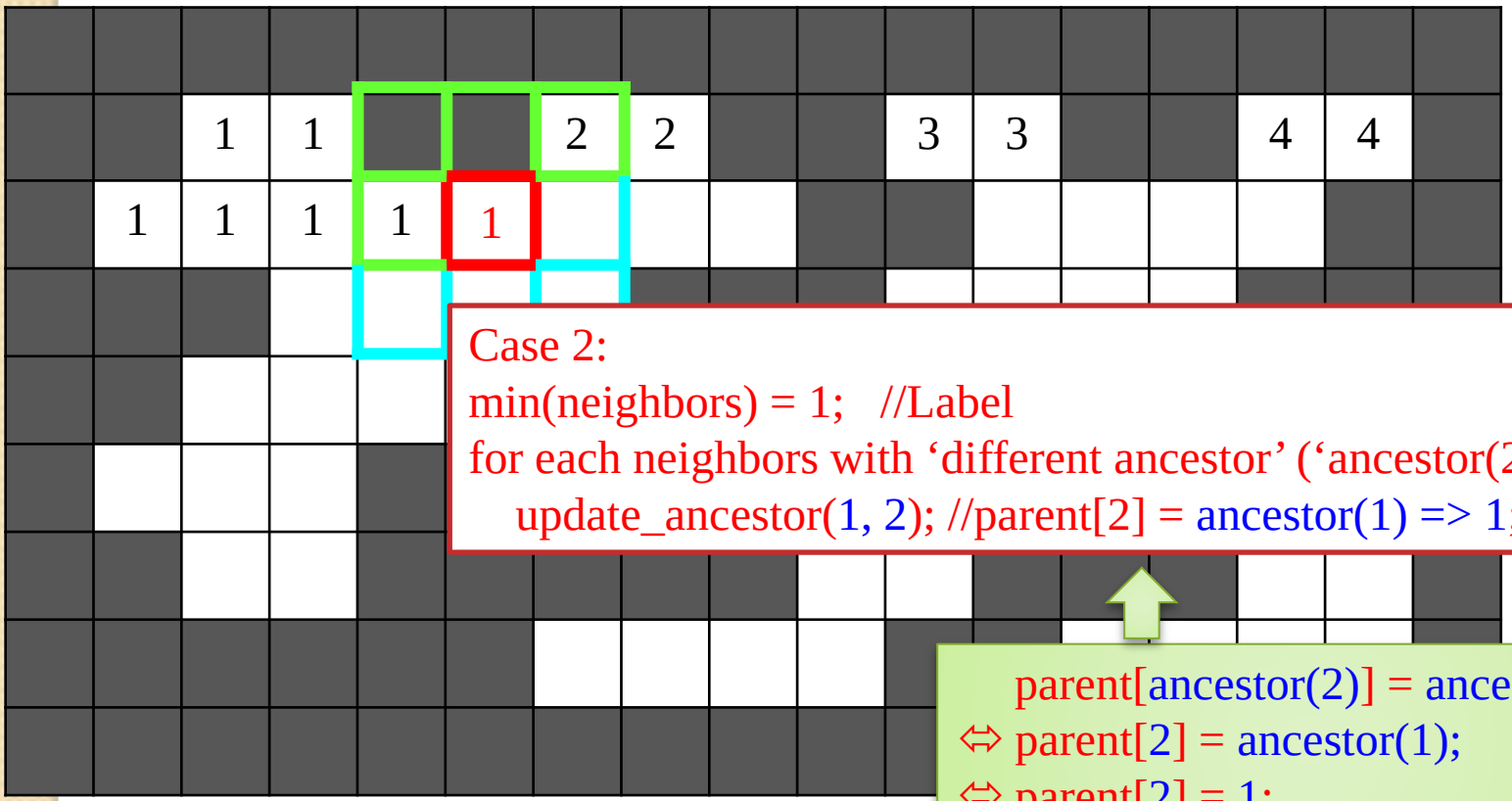
Example



step13

Label	1	2	3	4			
parent	0	1	0	0			

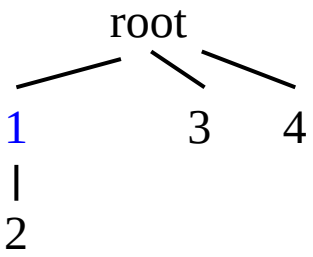
currentLabelCount = 4



Case 2:
min(neighbors) = 1; //Label
for each neighbors with 'different ancestor' ('ancestor(2)' here)
update_ancestor(1, 2); //parent[2] = ancestor(1) => 1;

parent[ancestor(2)] = ancestor(1);
⇔ parent[2] = ancestor(1);
⇔ parent[2] = 1;

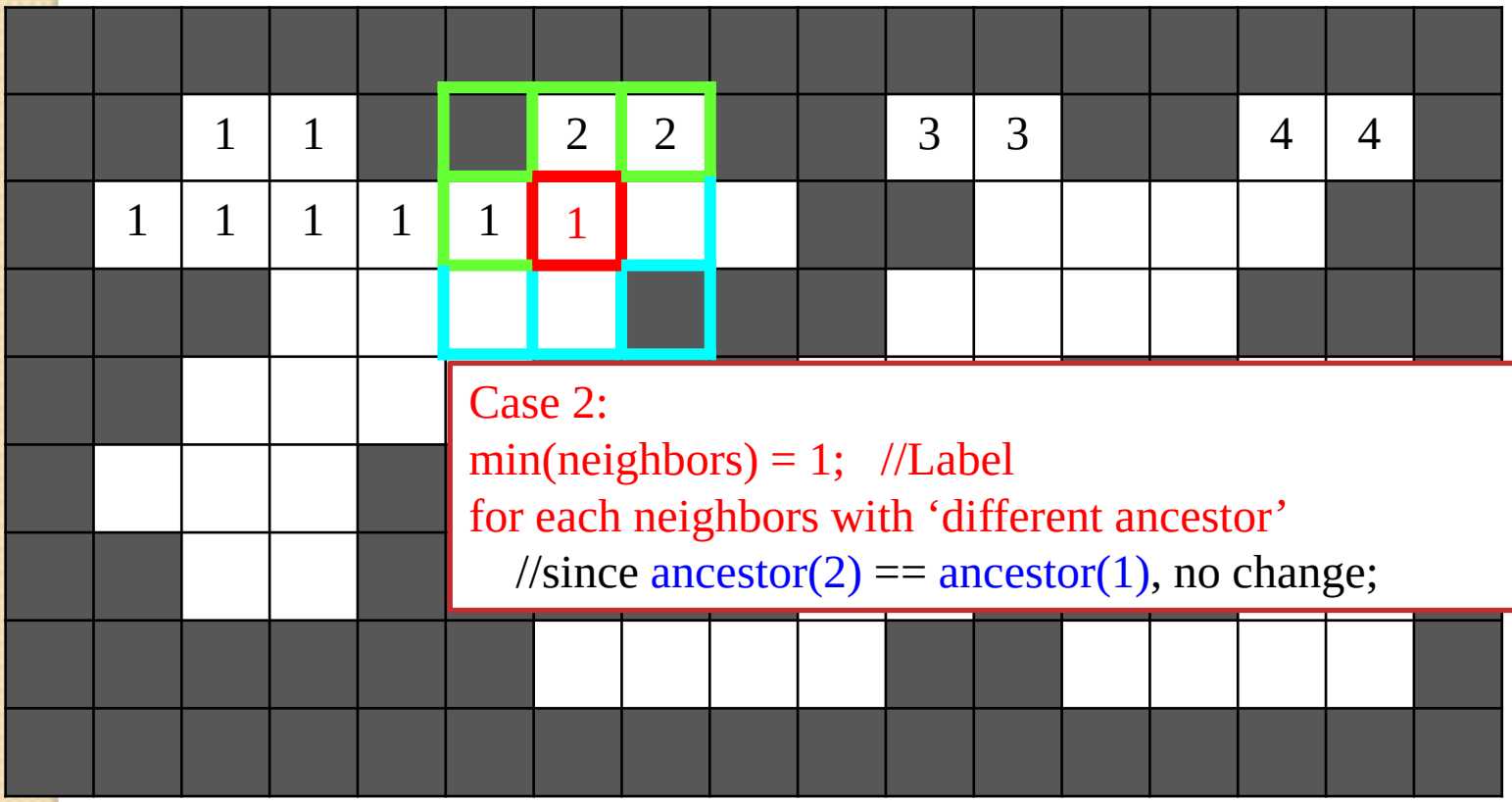
Example



step14

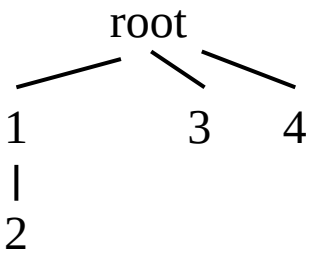
currentLabelCount = 4

Label	1	2	3	4			
parent	0	1	0	0			



Case 2:
min(neighbors) = 1; //Label
for each neighbors with 'different ancestor'
//since ancestor(2) == ancestor(1), no change;

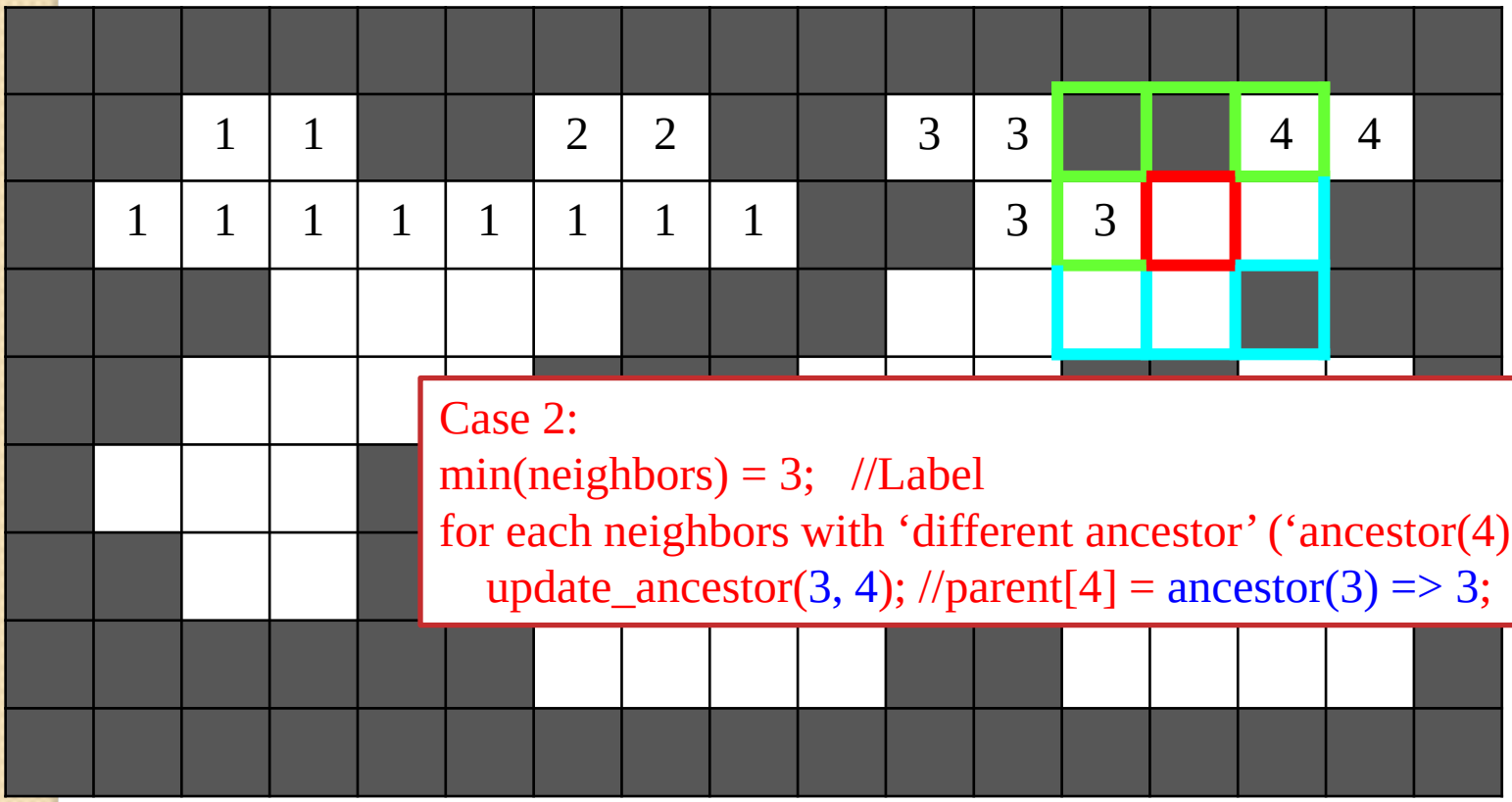
Example



step19

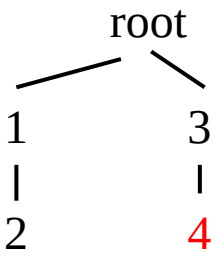
currentLabelCount = 4

Label	1	2	3	4			
parent	0	1	0	0			



Case 2:
min(neighbors) = 3; //Label
for each neighbors with 'different ancestor' ('ancestor(4)' here)
update_ancestor(3, 4); //parent[4] = ancestor(3) => 3;

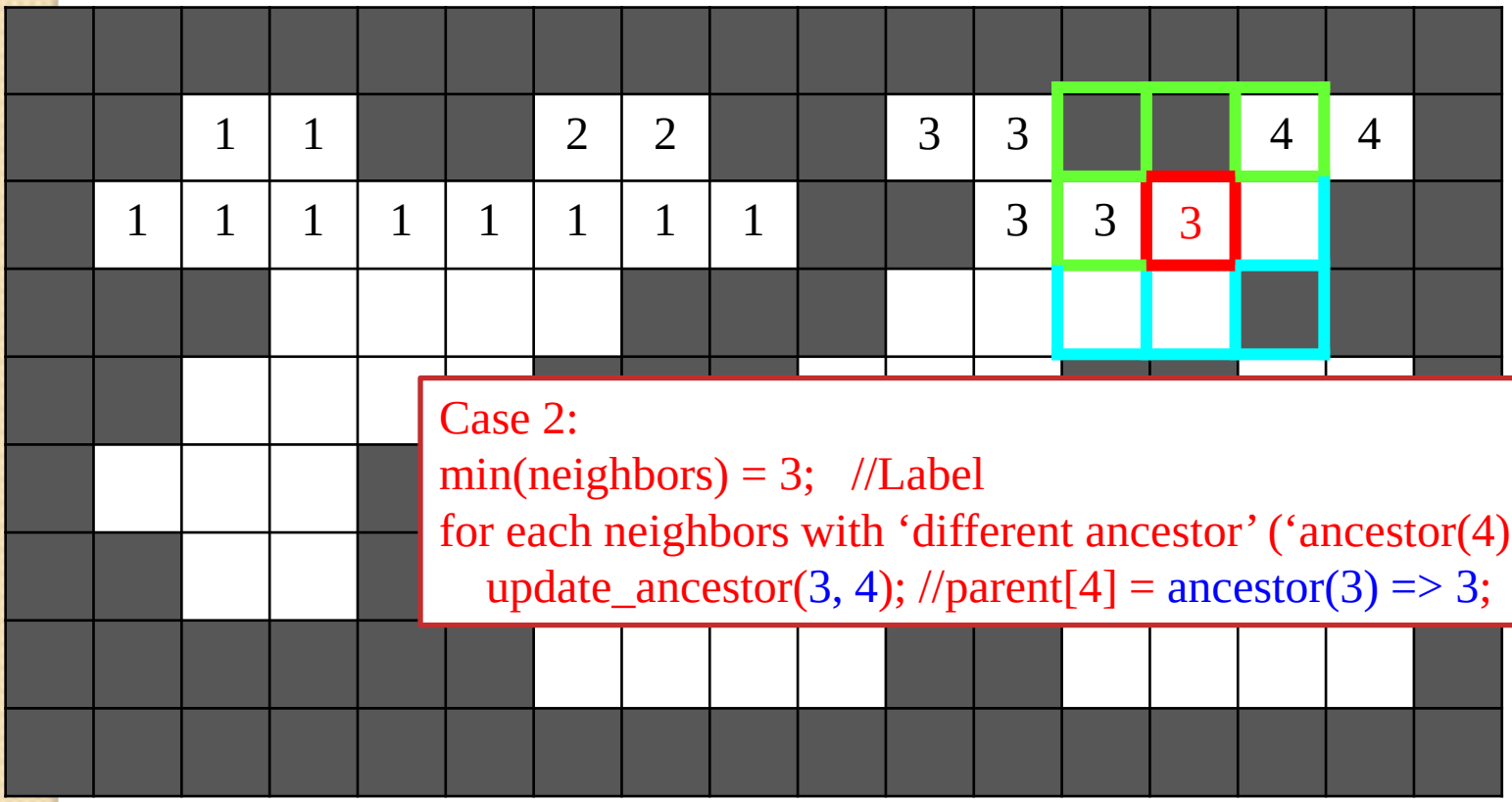
Example



step19

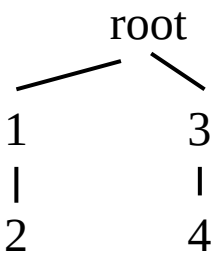
currentLabelCount = 4

Label	1	2	3	4			
parent	0	1	0	3			



Case 2:
min(neighbors) = 3; //Label
for each neighbors with 'different ancestor' ('ancestor(4)' here)
update_ancestor(3, 4); //parent[4] = ancestor(3) => 3;

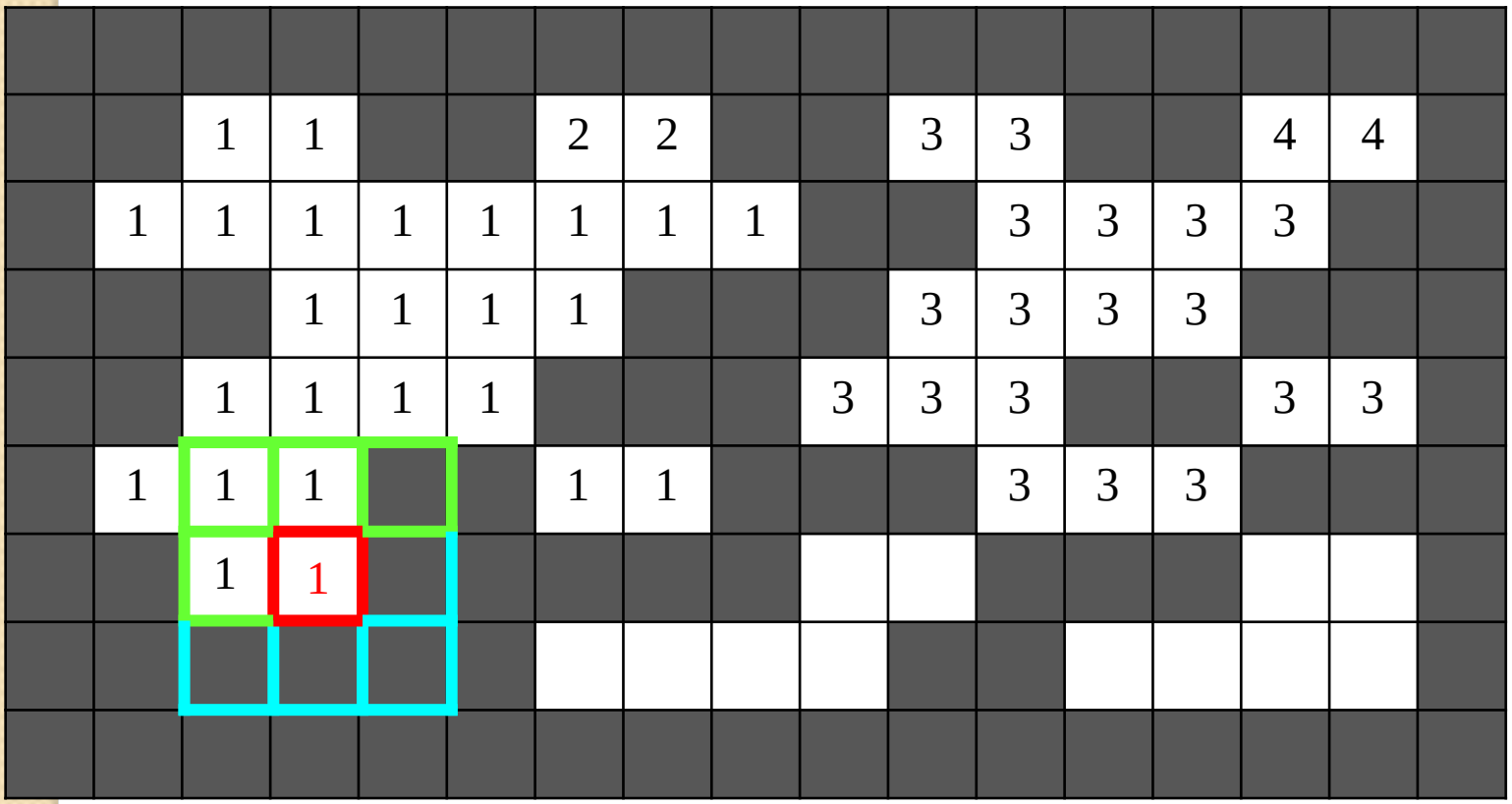
Example



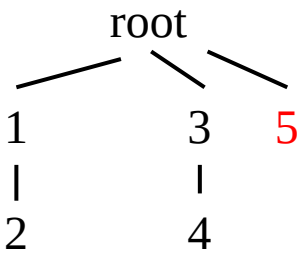
step47

currentLabelCount = 4

Label	1	2	3	4			
parent	0	1	0	3			



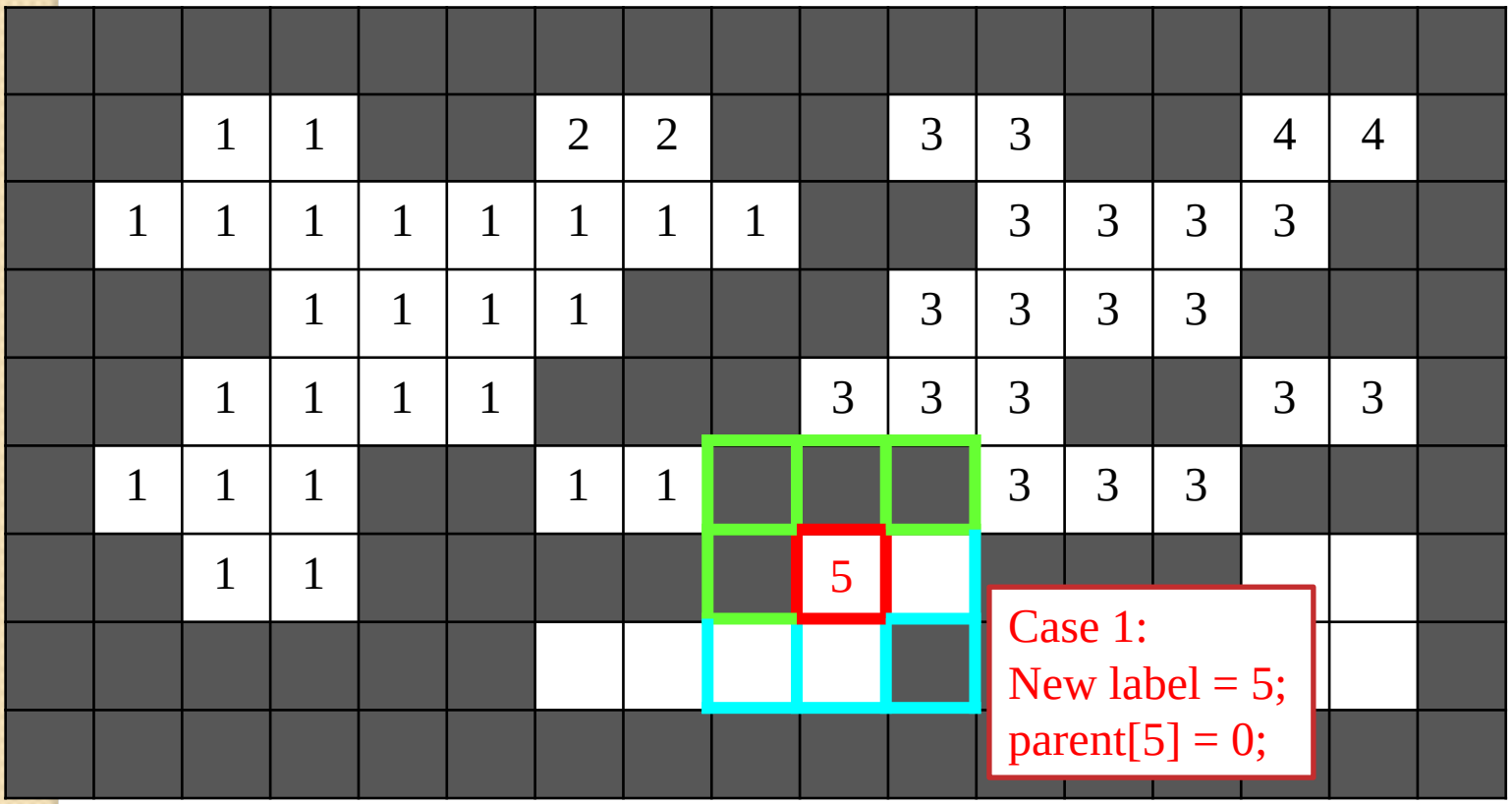
Example



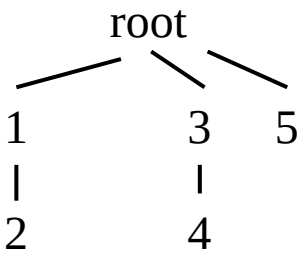
step48

currentLabelCount = 5

Label	1	2	3	4	5		
parent	0	1	0	3	0		



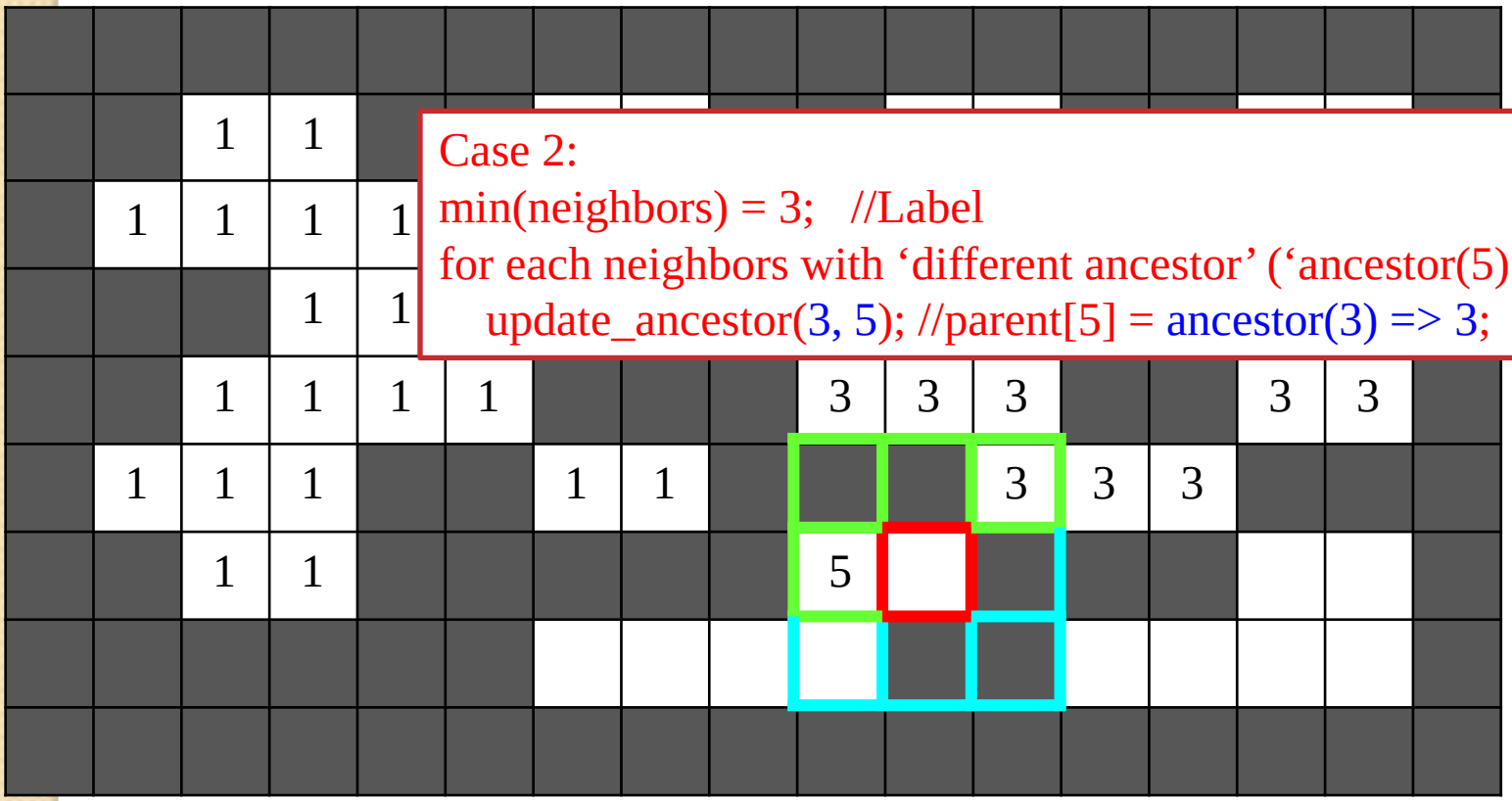
Example



step49

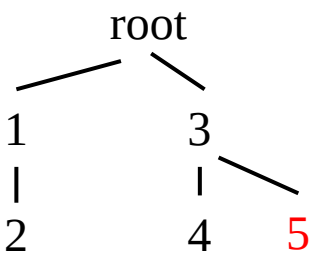
Label	1	2	3	4	5		
parent	0	1	0	3	0		

currentLabelCount = 5



Case 2:
min(neighbors) = 3; //Label
for each neighbors with 'different ancestor' ('ancestor(5)' here)
update_ancestor(3, 5); //parent[5] = ancestor(3) => 3;

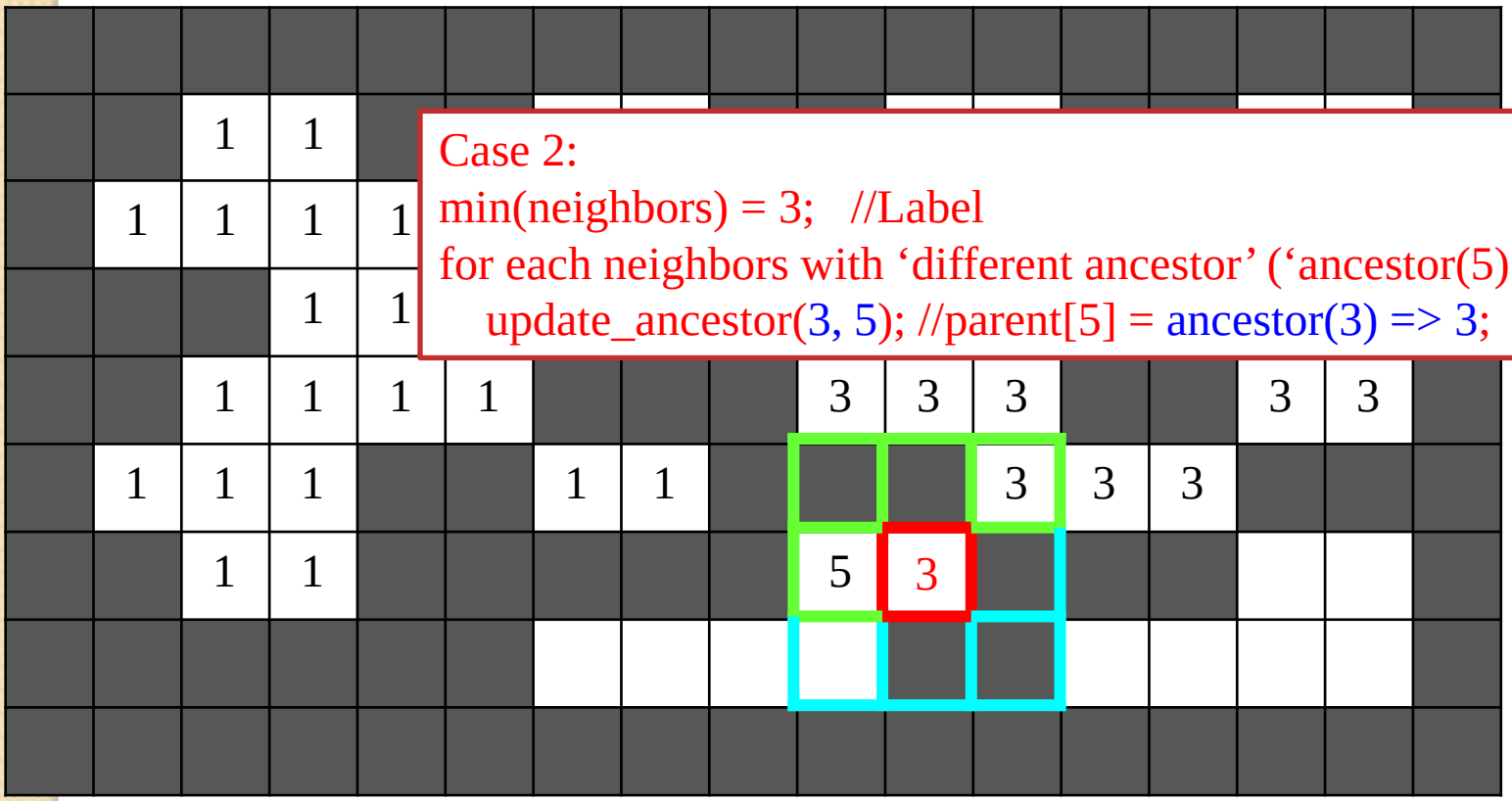
Example



step49

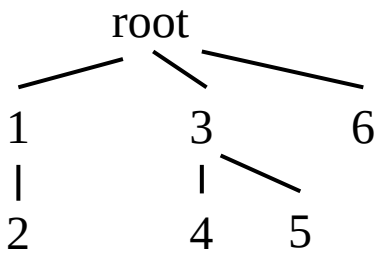
currentLabelCount = 5

Label	1	2	3	4	5		
parent	0	1	0	3	3		



Case 2:
min(neighbors) = 3; //Label
for each neighbors with 'different ancestor' ('ancestor(5)' here)
update_ancestor(3, 5); //parent[5] = ancestor(3) => 3;

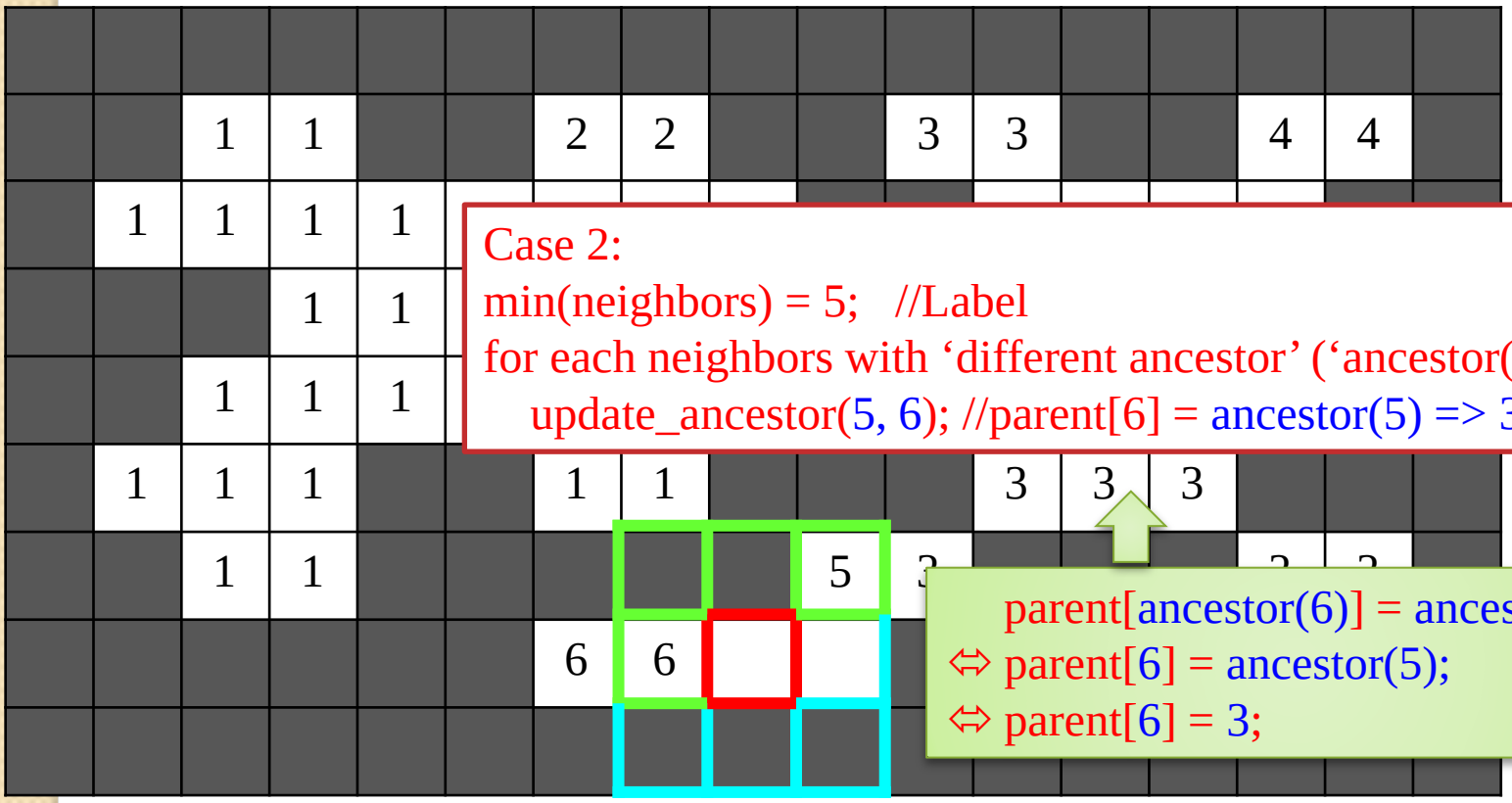
Example



step54

Label	1	2	3	4	5	6	
parent	0	1	0	3	3	0	

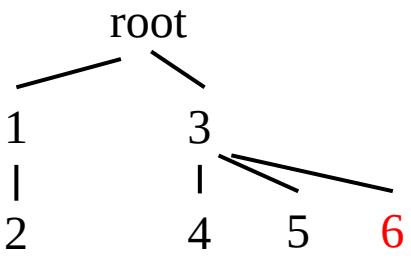
currentLabelCount = 6



Case 2:
min(neighbors) = 5; //Label
for each neighbors with 'different ancestor' ('ancestor(6)' here)
update_ancestor(5, 6); //parent[6] = ancestor(5) => 3;

parent[ancestor(6)] = ancestor(5);
⇔ parent[6] = ancestor(5);
⇔ parent[6] = 3;

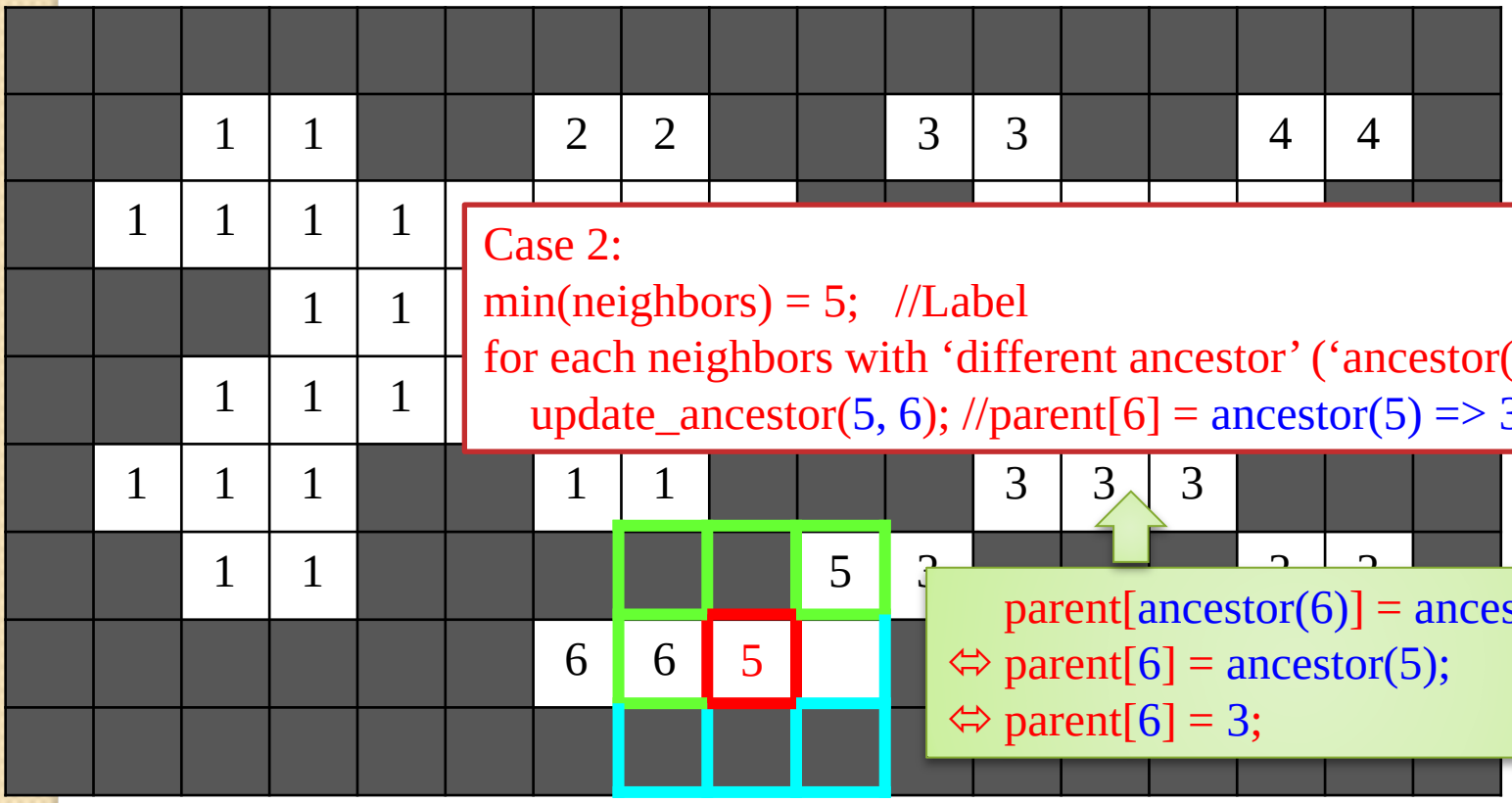
Example



step54

Label	1	2	3	4	5	6	
parent	0	1	0	3	3	3	

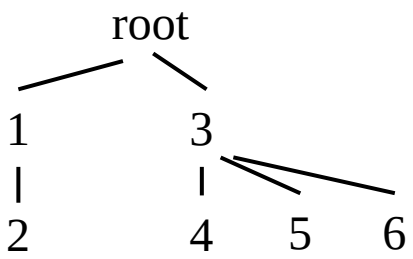
currentLabelCount = 6



Case 2:
min(neighbors) = 5; //Label
for each neighbors with 'different ancestor' ('ancestor(6)' here)
update_ancestor(5, 6); //parent[6] = ancestor(5) => 3;

parent[ancestor(6)] = ancestor(5);
⇔ parent[6] = ancestor(5);
⇔ parent[6] = 3;

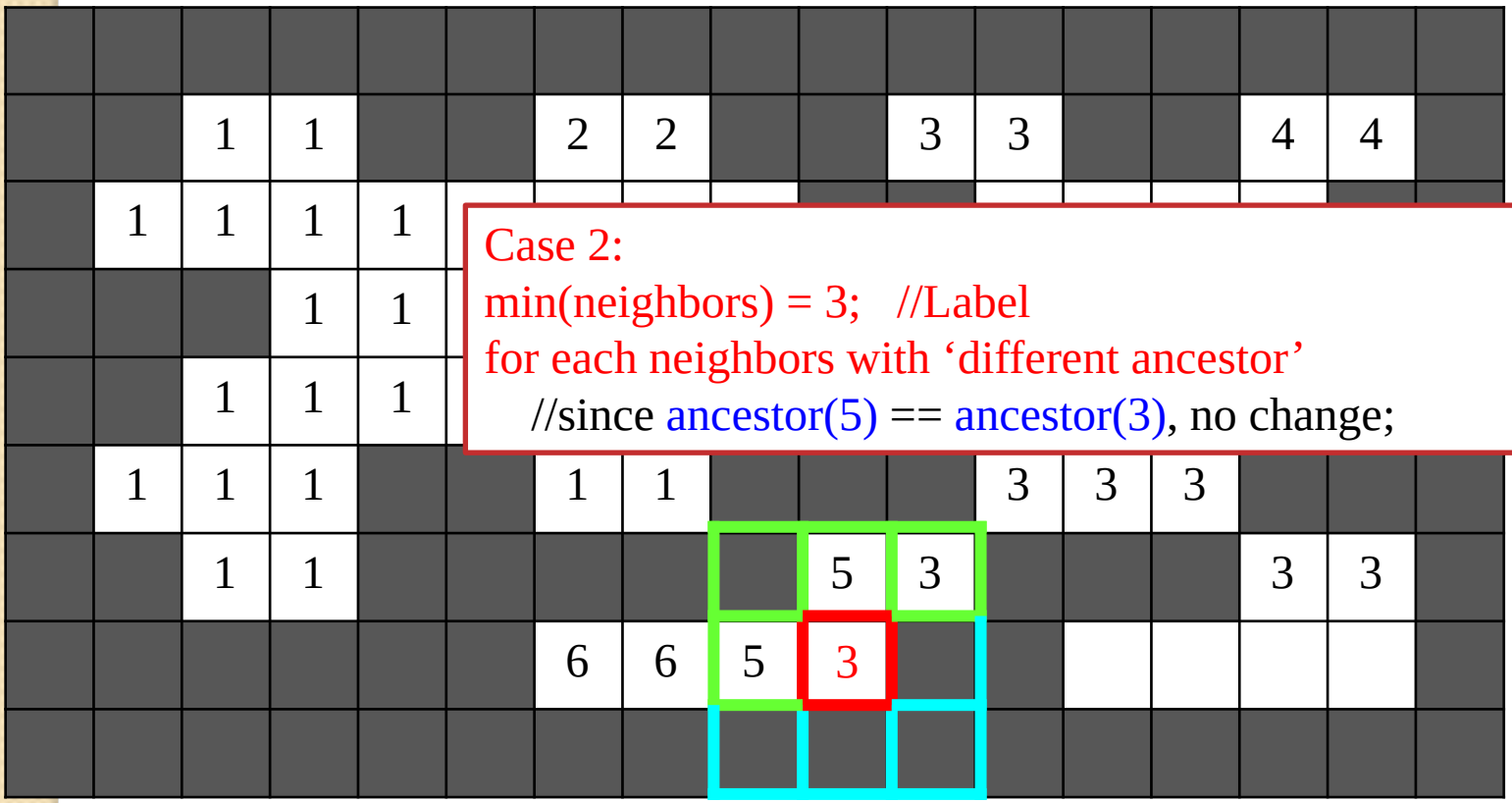
Example



step55

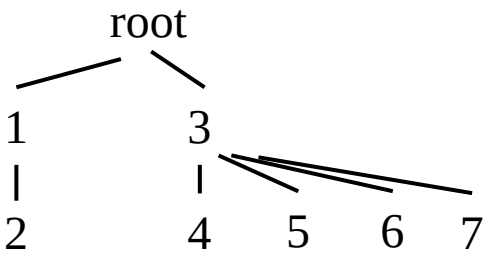
currentLabelCount = 6

Label	1	2	3	4	5	6	
parent	0	1	0	3	3	3	



Case 2:
min(neighbors) = 3; //Label
for each neighbors with 'different ancestor'
//since ancestor(5) == ancestor(3), no change;

Example

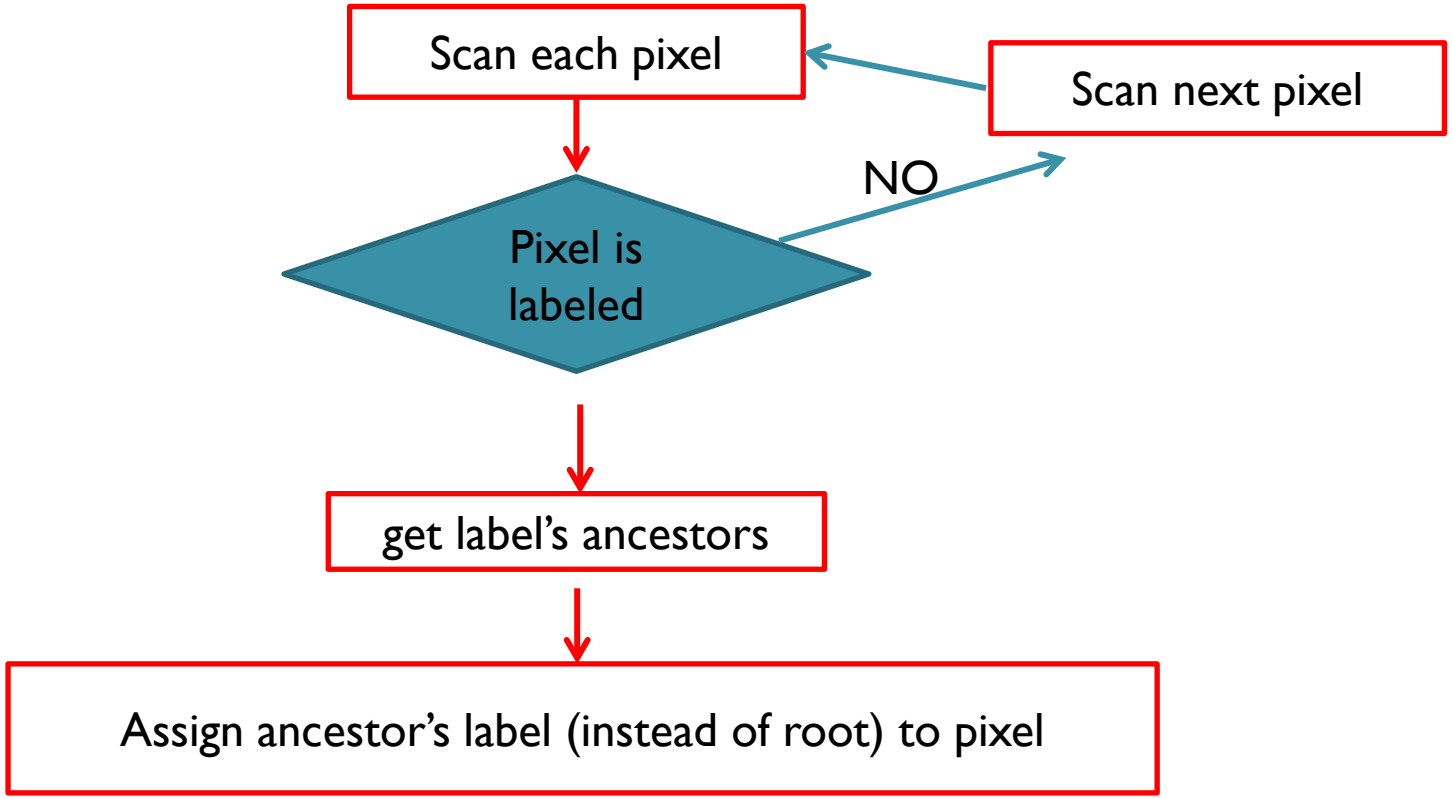


currentLabelCount = 6

Label	1	2	3	4	5	6	7
parent	0	1	0	3	3	3	3

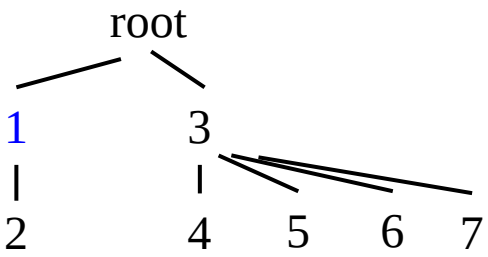
		1	1			2	2			3	3			4	4	
	1	1	1	1	1	1	1	1			3	3	3	3		
			1	1	1	1				3	3	3	3			
		1	1	1	1				3	3	3			3	3	
	1	1	1			1	1				3	3	3			
		1	1						5	3				3	3	
						6	6	5	3			7	3	3	3	

- Two-pass Algorithm
  - Second pass (aggregation)



ancestry or ancestor is parent's parent's …… until root.

Example



ancestry is parent's parent's …… until root.

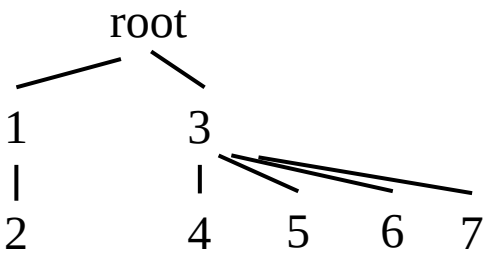
step1 (Second pass)

Label	1	2	3	4	5	6	7
parent	0	1	0	3	3	3	3

		1	1			2	2			3	3			4	4
	1										3	3	3	3	
										3	3	3	3		
		1	1	1	1				3	3	3			3	3
	1	1	1			1	1				3	3	3		
		1	1						5	3				3	3
						6	6	5	3			7	3	3	3

Label = ancestor(Label);
⇒ Label = ancestor(1);
⇒ Label = 1;

Example



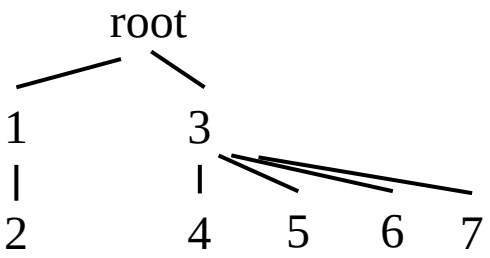
step3

Label	1	2	3	4	5	6	7
parent	0	1	0	3	3	3	3

		1	1			2	2			3	3			4	4
	1	1	1	1	1								3	3	
			1	1	1								3		
		1	1	1	1				3	3	3			3	3
	1	1	1			1	1				3	3	3		
		1	1						5	3				3	3
						6	6	5	3			7	3	3	3

Label = ancestor(Label);
⇒ Label = ancestor(2);
⇒ Label = 1;

Example

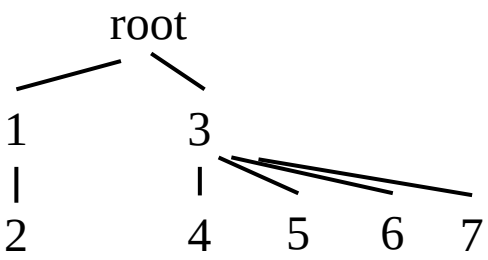


step3

Label	1	2	3	4	5	6	7
parent	0	1	0	3	3	3	3

Label = ancestor(Label);
⇒ Label = ancestor(2);
⇒ Label = 1;

Example



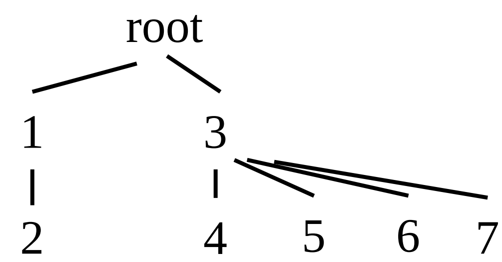
step7

Label	1	2	3	4	5	6	7
parent	0	1	0	3	3	3	3

		1	1			1	1			3	3			4	4
	1	1	1	1	1	1	1	1							
			1	1	1	1									
		1	1	1	1				3						
	1	1	1			1	1				3	3	3		
		1	1						5	3				3	3
						6	6	5	3			7	3	3	3

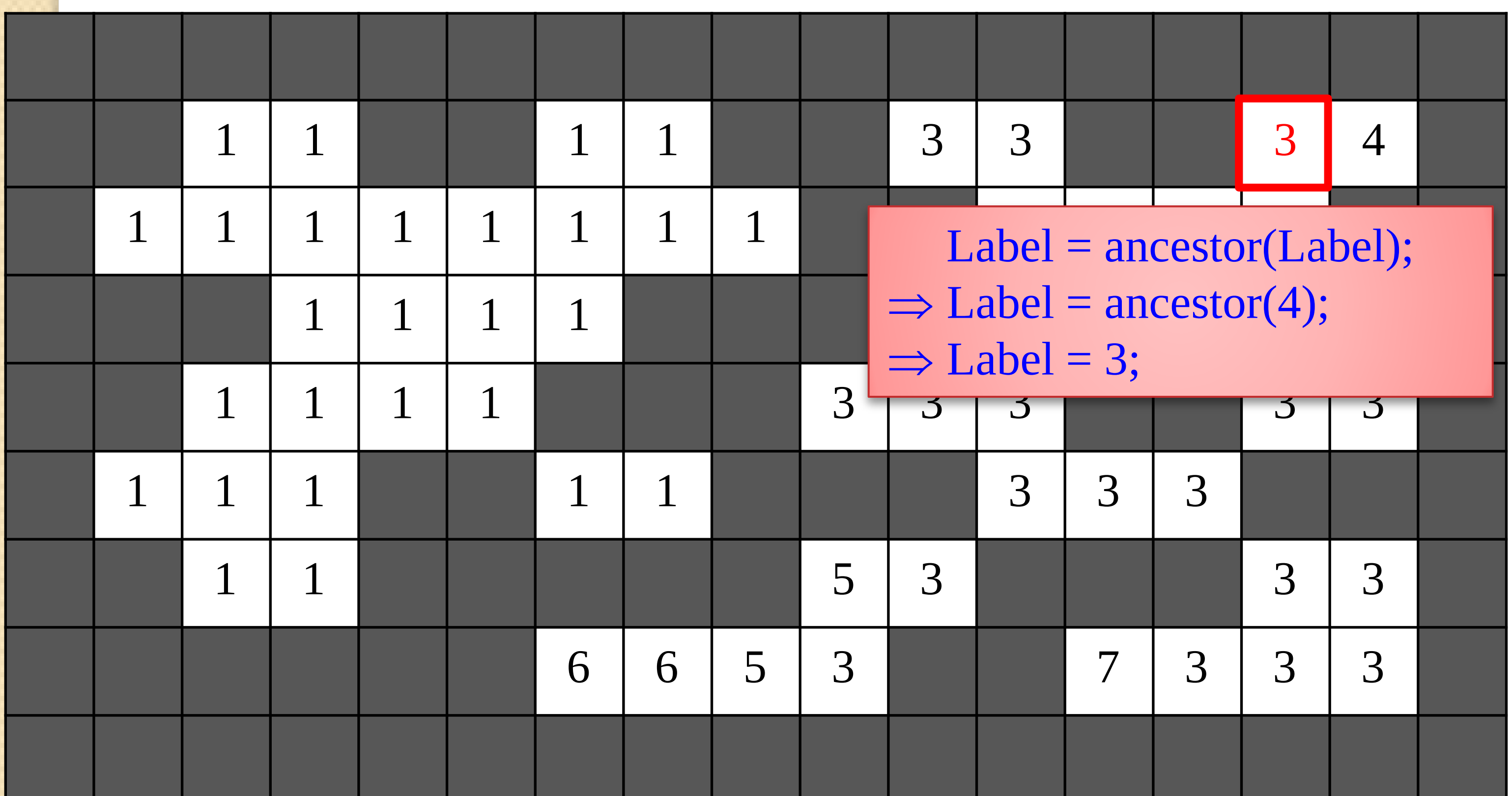
Label = ancestor(Label);
⇒ Label = ancestor(4);
⇒ Label = 3;

Example

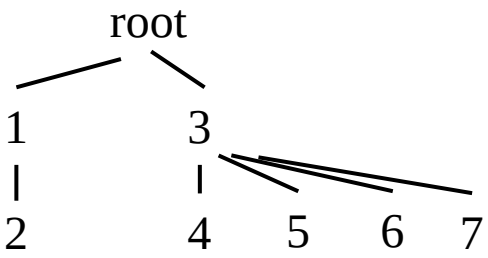


step7

Label	1	2	3	4	5	6	7
parent	0	1	0	3	3	3	3

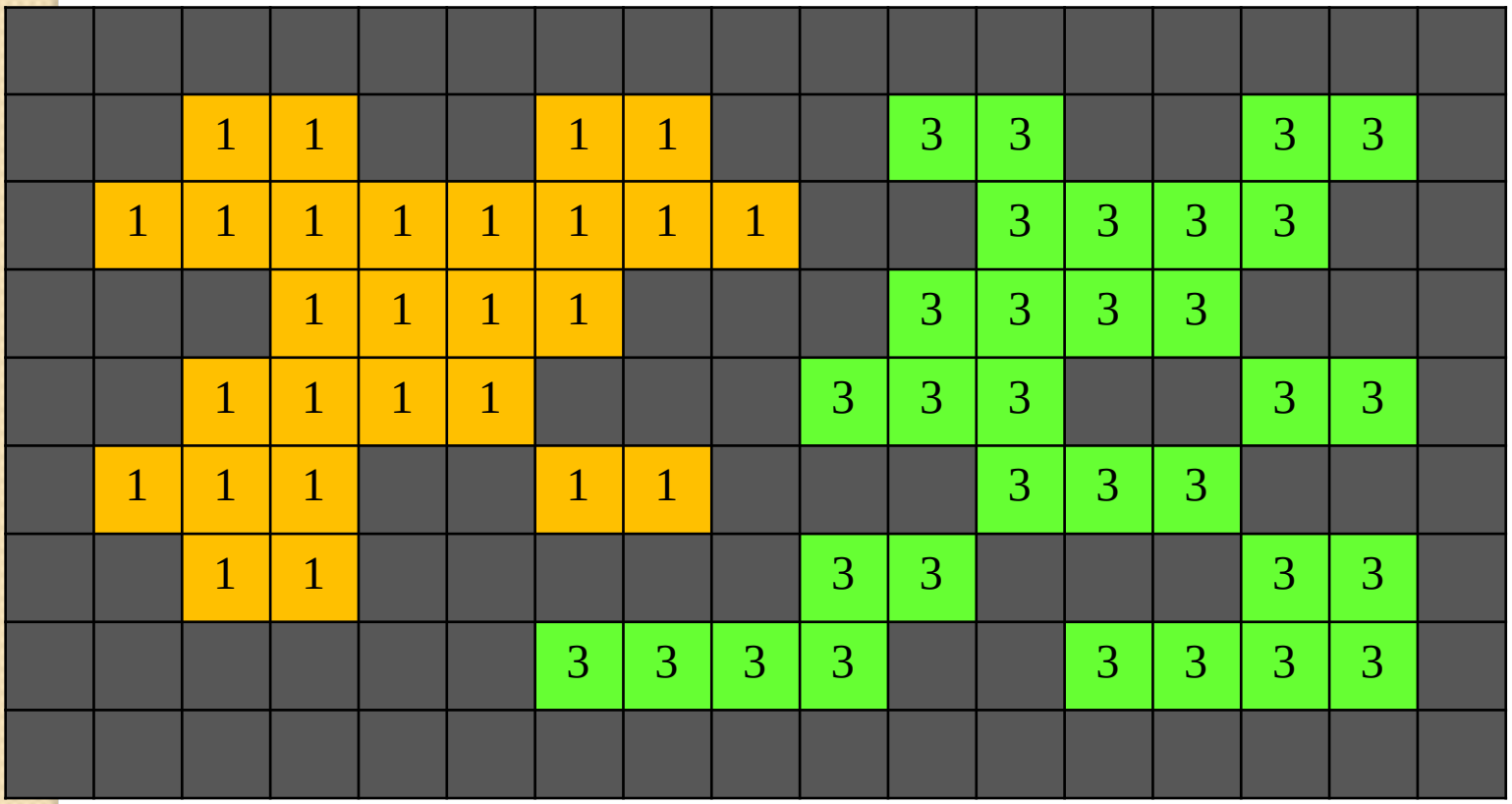


Example



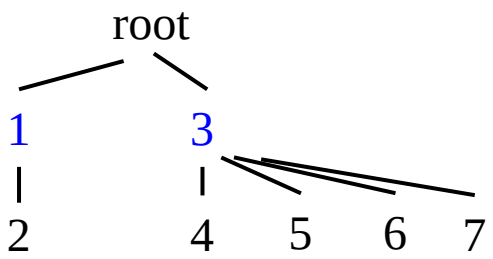
End process

Label	1	2	3	4	5	6	7
parent	0	1	0	3	3	3	3



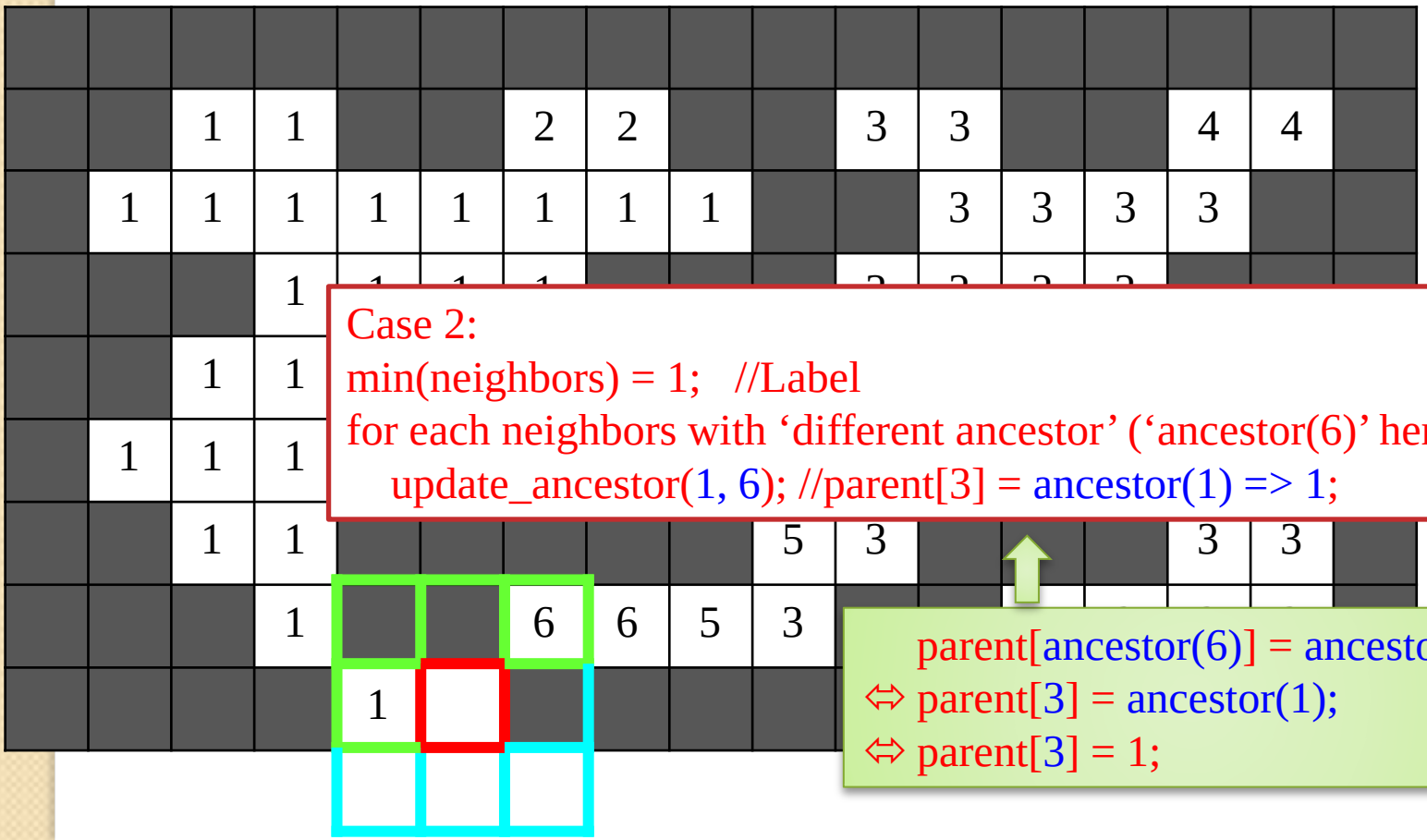
EXAMPLE 2

Example 2



(First pass)

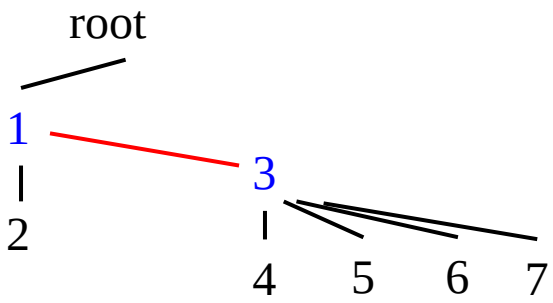
Label	1	2	3	4	5	6	7
parent	0	1	0	3	3	3	3



Case 2:
min(neighbors) = 1; //Label
for each neighbors with 'different ancestor' ('ancestor(6)' here)
update_ancestor(1, 6); //parent[3] = ancestor(1) => 1;

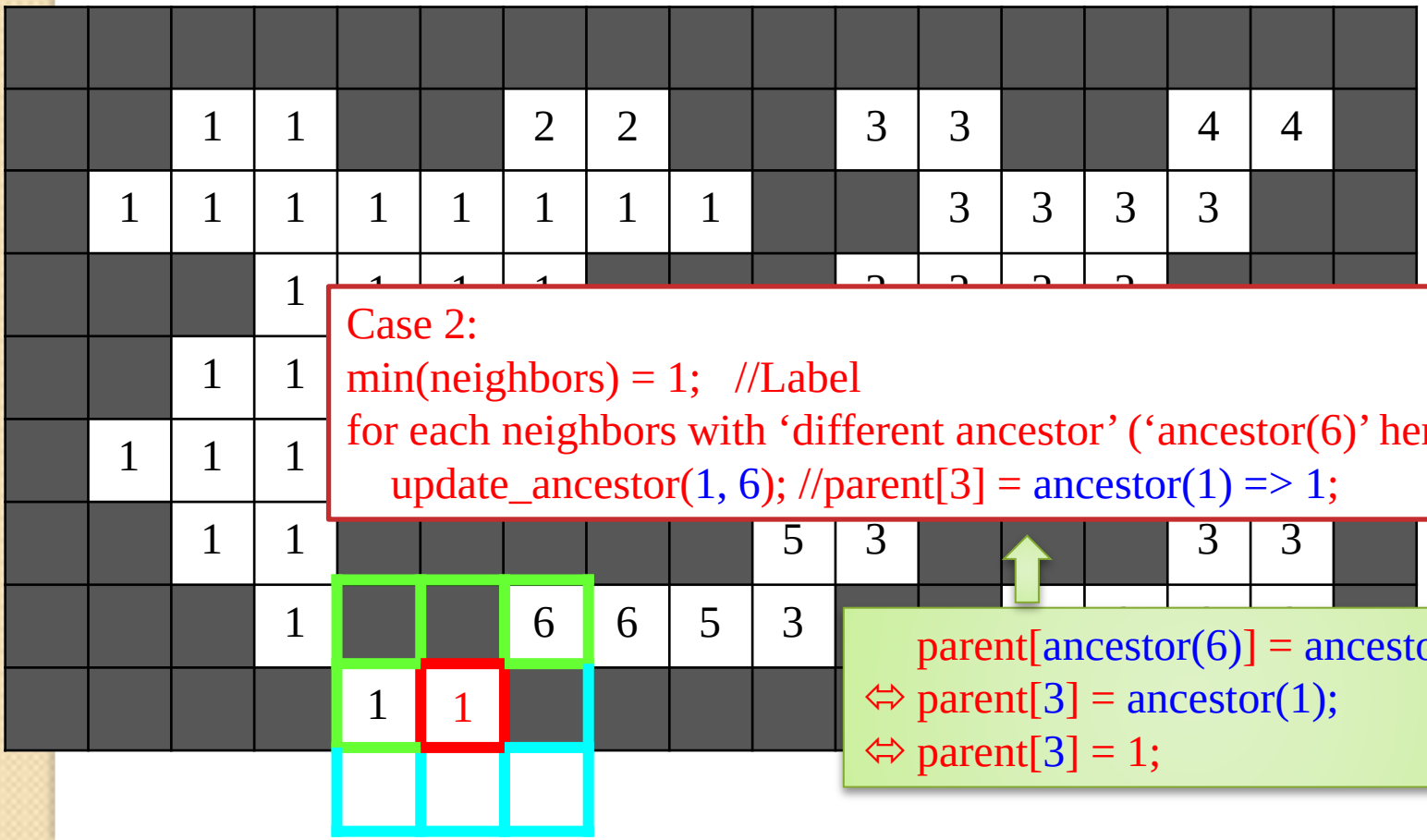
parent[ancestor(6)] = ancestor(1);
⇔ parent[3] = ancestor(1);
⇔ parent[3] = 1;

Example 2

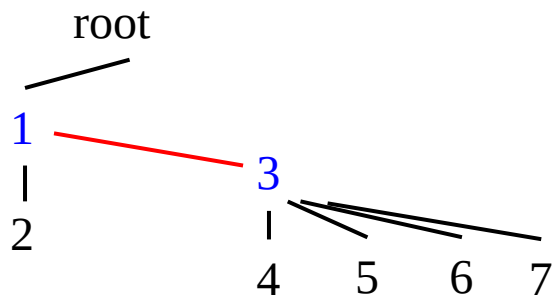


(First pass)

Label	1	2	3	4	5	6	7
parent	0	1	1	3	3	3	3



Example 2

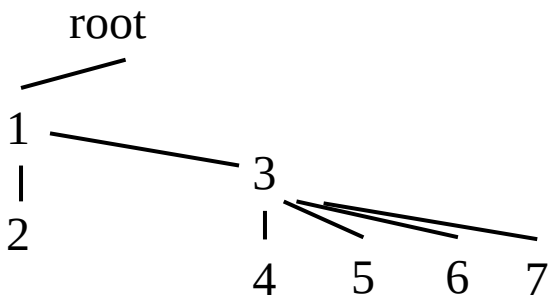


(First pass)

Label	1	2	3	4	5	6	7
parent	0	1	1	3	3	3	3

		1	1			2	2			3	3			4	4	
	1	1	1	1	1	1	1	1			3	3	3	3		
			1	1	1	1				3	3	3	3			
		1	1	1	1				3	3	3			3	3	
	1	1	1			1	1				3	3	3			
		1	1						5	3				3	3	
			1			6	6	5	3			7	3	3	3	
				1	1											

Example 2

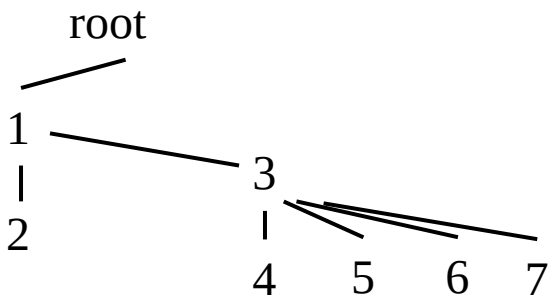


step1 (Second pass)

Label	1	2	3	4	5	6	7
parent	0	1	1	3	3	3	3

		1	1			2	2			3	3			4	4
	1	1	1	1	1	1	1	1			3	3	3	3	
			1	1	1	1				3	3	3	3		
		1	1	1	1				3	3	3			3	3
	1	1	1			1	1				3	3	3		
		1	1						5	3				3	3
			1			6	6	5	3			7	3	3	3
				1	1										

Example 2



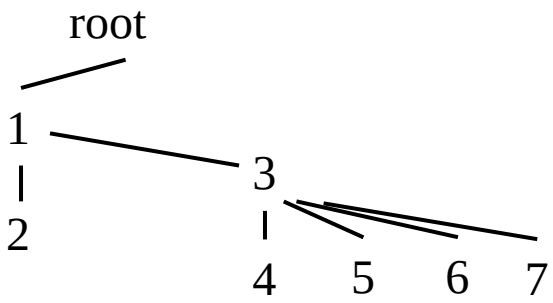
step5 (Second pass)

Label	1	2	3	4	5	6	7
parent	0	1	1	3	3	3	3

		1	1			1	1			3	3			4	4
	1	1	1	1	1	1	1	1							
			1	1	1	1									
		1	1	1	1				3	3	3			3	3
	1	1	1			1	1				3	3	3		
		1	1						5	3				3	3
			1			6	6	5	3			7	3	3	3
				1	1										

Label = ancestor(Label);
⇒ Label = ancestor(3);
⇒ Label = 1;

Example 2



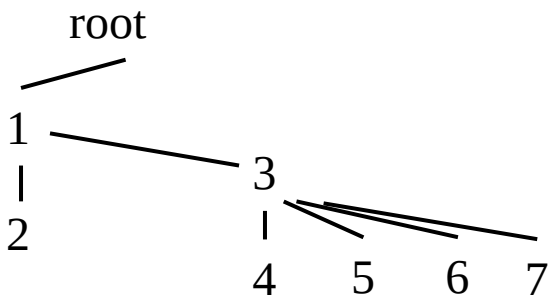
step5 (Second pass)

Label	1	2	3	4	5	6	7
parent	0	1	1	3	3	3	3

		1	1			1	1			1	3			4	4	
	1	1	1	1	1	1	1	1								
			1	1	1	1										
		1	1	1	1				3	3	3			3	3	
	1	1	1			1	1				3	3	3			
		1	1						5	3				3	3	
			1			6	6	5	3			7	3	3	3	
				1	1											

Label = ancestor(Label);
⇒ Label = ancestor(3);
⇒ Label = 1;

Example 2



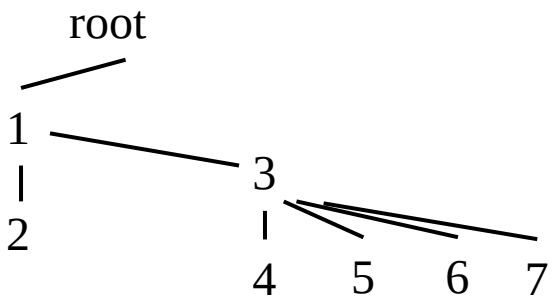
step7 (Second pass)

Label	1	2	3	4	5	6	7
parent	0	1	1	3	3	3	3

		1	1			1	1			1	1			4	4	
	1	1	1	1	1	1	1	1								
			1	1	1	1										
		1	1	1	1				3	3	3			3	3	
	1	1	1			1	1				3	3	3			
		1	1						5	3				3	3	
			1			6	6	5	3			7	3	3	3	
				1	1											

Label = ancestor(Label);
⇒ Label = ancestor(4);
⇒ Label = 1;

Example 2



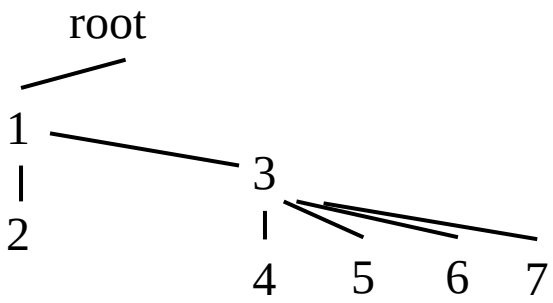
step7 (Second pass)

Label	1	2	3	4	5	6	7
parent	0	1	1	3	3	3	3

		1	1			1	1			1	1			1	4
	1	1	1	1	1	1	1	1							
			1	1	1	1									
		1	1	1	1				3	3	3			3	3
	1	1	1			1	1				3	3	3		
		1	1						5	3				3	3
			1			6	6	5	3			7	3	3	3
				1	1										

Label = ancestor(Label);
⇒ Label = ancestor(4);
⇒ Label = 1;

Example 2



(Second pass)

Label	1	2	3	4	5	6	7
parent	0	1	1	3	3	3	3

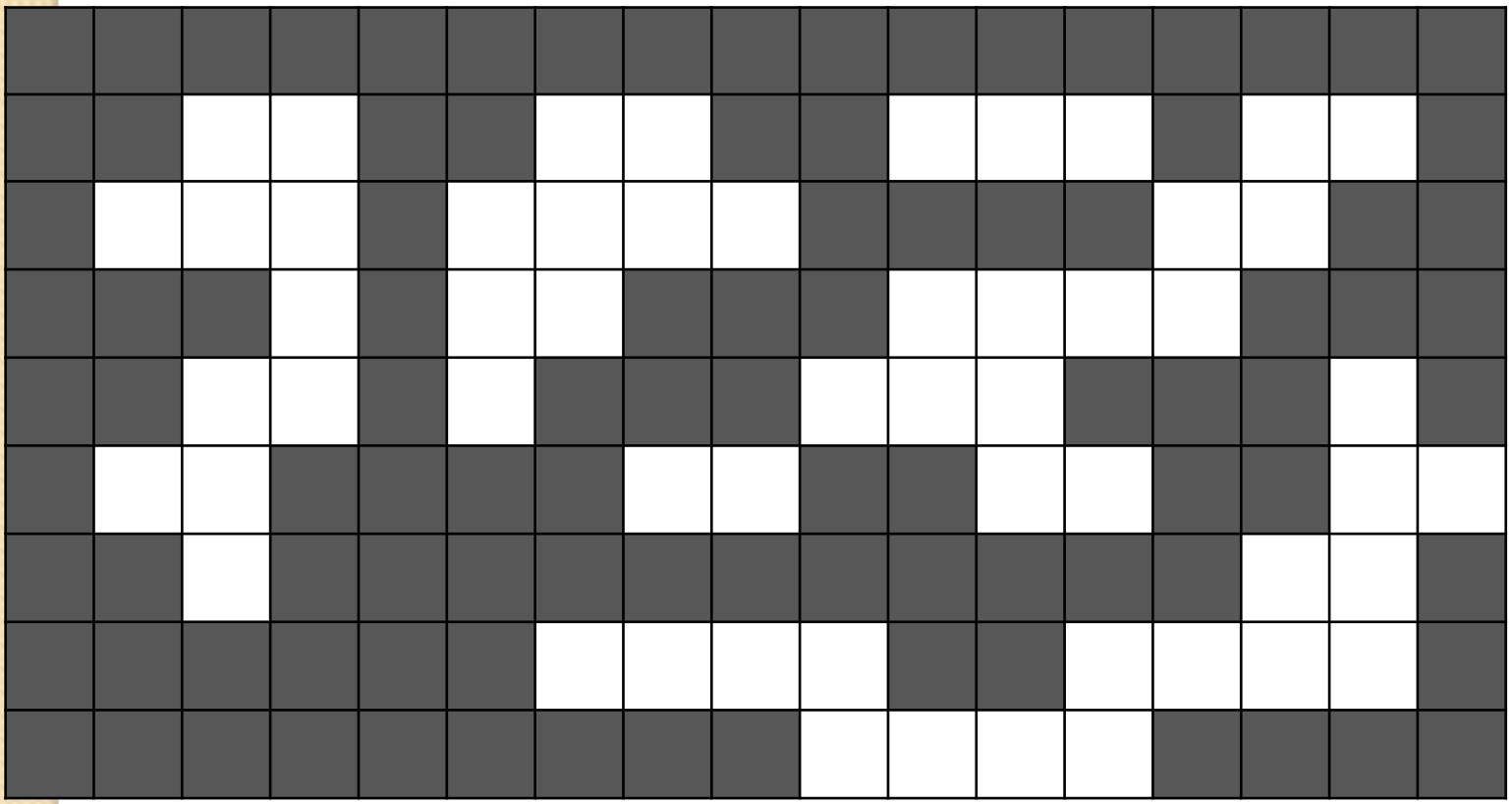
		1	1			1	1			1	1			1	1	
	1	1	1	1	1	1	1	1			1	1	1	1		
			1	1	1	1				1	1	1	1			
		1	1	1	1				1	1	1			1	1	
	1	1	1			1	1				1	1	1			
		1	1						1	1				1	1	
			1			1	1	1	1			1	1	1	1	
				1	1											

隨堂練習

- 手寫: first-pass, (8連通)

17

9



隨堂練習

- 手寫: first-pass,

Label	1	2	3	4	5	6	7	8	9
parent	0	0	0	3	3	0	3	6	6

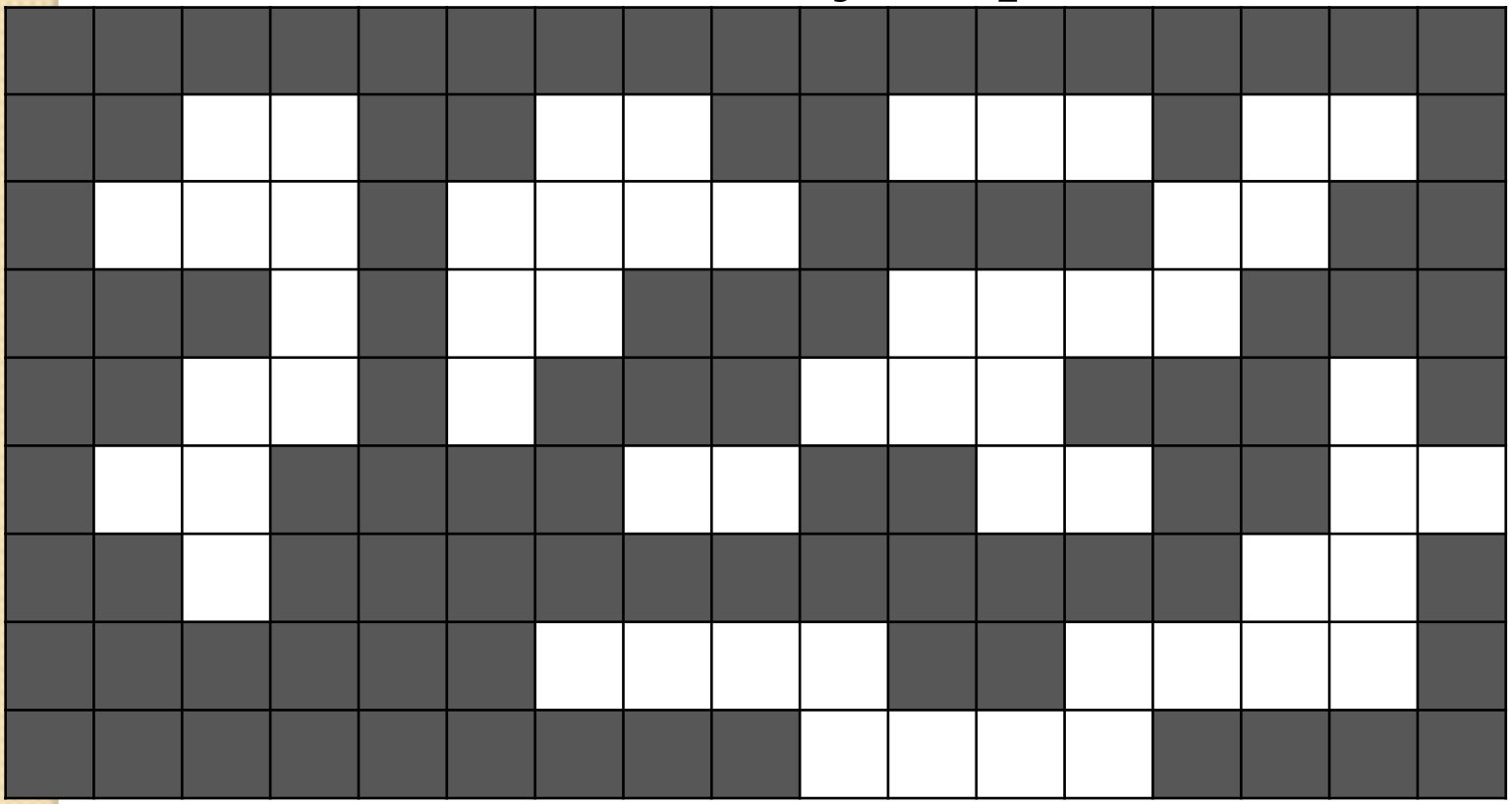
17

9

		1	1			2	2			3	3	3		4	4	
	1	1	1		2	2	2	2					3	3		
			1		2	2				5	5	3	3			
		1	1		2				5	5	3				6	
	1	1					7	5			3	3			6	6
		1												6	6	
						8	8	8	8			9	6	6	6	
									8	8	8	6				

Homework #6.3

- 1. 以練習題目實做出Two-pass (8連通)
- 2. 讀取一張binary map並show出結果



```
algorithm TwoPass(data)
   linked = []
   labels = structure with dimensions of data, initialized with the value of Background

   First pass

   for row in data:
       for column in row:
           if data[row][column] is not Background

               neighbors = connected elements with the current element's value

               if neighbors is empty
                   linked[NextLabel] = set containing NextLabel
                   labels[row][column] = NextLabel
                   NextLabel += 1

               else

                   Find the smallest label

                   L = neighbors labels
                   labels[row][column] = min(L)
                   for label in L
                       linked[label] = union(linked[label], L)

   Second pass

   for row in data
       for column in row
           if data[row][column] is not Background
               labels[row][column] = find(labels[row][column])

   return labels
```