Bildregionen und Konturen

Seminar Bildsegmentierung und Computer Vision WS05/06

Heike Zierau

23.01.2006

Inhalt

Binärbilder

Binärbilder:
Ein Pixel kann nur **zwei verschiedene** Werte annehmen, Vordergrundwert, bzw. Hintergrundwert.

$$I(x,y) = \left\{ \begin{array}{ll} 0, & \textit{Hintergrund} \\ 1, & \textit{Vordergrund} \end{array} \right\}$$

Aufgabe: verbundene Bildregionen auffinden sowie diese **isolieren** und **beschreiben**.

Binärbilder

Eine **verbundene binäre Bildregion** ist eine Gruppe von **aneinandergrenzenden** Vordergrundpixeln nach der **Vierer**- oder **Achter-Nachbarschaft**
Aufgabe: Auffinden von Objekten.

Binärbilder

- **Vierernachbarschaft**:

$$N(x,y) = \begin{array}{|c|c|c|}\hline & N_2 & \\ \hline N_1 & \mathrm{x} & N_3 \\ \hline & N_4 & \\ \hline \end{array}$$

- **Achternachbarschaft**:

$$N(x,y) = \begin{array}{|c|c|c|}\hline N_2 & N_3 & N_4 \\ \hline N_1 & \mathrm{x} & N_5 \\ \hline N_8 & N_7 & N_6 \\ \hline \end{array}$$

Binärbilder

Aufgabe bei gegebenem Binärbild:

1. **Welche** Pixel gehören zu welcher Region?
2. **Wie viele** Regionen gibt es im Bild?
3. **Wo** befinden sich diese Regionen?

Binärbilder

$\Rightarrow$ **Regionenmarkierung**:
Vorgehensweise :

- Schrittweise zueinander **benachbarte** Pixel zu Regionen **zusammenfügen**
- allen Pixeln innerhalb einer Region eine **eindeutige Identifikationsnummer** zuweisen, sogenannte ***Labels***.

Flood Filling

IDEE : Ausgehend von einem gegebenen **Startpunkt** wird eine einzelne Region in alle Richtungen **ausgefüllt**.
Binäres Ausgangsbild

$$I(x,y) = \left\{ \begin{array}{ll} 0, & \textit{Hintergrund} \\ 1, & \textit{Vordergrund} \\ 2,3,..., & \textit{Labels} \end{array} \right\}$$

Flood Filling

Flood Filling-Algorithmus:

- Suche ein noch **unmarkiertes Vordergrundpixel** ($I(x, y) = 1$).
- **Fülle** den Rest der zugehörigen Region **aus**.

Flood Filling

Eine naheliegende Implementierung hierfür wäre eine **Rekursive**:

FLOOD FILLING ALGORITHMUS

$FloodFill(I, x, y, label)$
IF(x, y) innerhalb der Bildgrenzen AND $I(x, y) = 1$ THEN
$I(x, y) := label$
$FloodFill(I, x + 1, y, label)$
$FloodFill(I, x, y - 1, label)$
$FloodFill(I, x, y + 1, label)$
$FloddFill(I, x - 1, y, label)$

Flood Filling

Problem:

- Rekursive Programme dieser Form benötigen **äußerst viel Speicherplatz**, wodurch sich Bilder mit mehr als ca. 200x200 Pixeln mit solch einem Programm **nicht** mehr bearbeiten lassen. Daher wird dieser Algorithmus in der Praxis selten verwendet.
- Es gibt jedoch auch **iterative Lösungen** für Flood Filling, die auf **größere Bilder** anwendbar sind. Wir wollen hier aber nicht näher darauf eingehen.

Sequentielle Regionenmarkierung

Sequentielle Regionenmarkierung ist eine **nichtrekursive Technik**, auch bekannt als **„region labeling”**.**IDEE: Sequentiell** das Bild **zeilenweise** durchlaufen und die Regionen **markieren**.

Sequentielle Regionenmarkierung

ALGORITHMUS:

1. **Sequentielles Durchlaufen** des Bildes von links oben nach rechts unten. Jedes **Vordergrundpixel vorläufig markieren**.
2. **Auflösen** der kollidierten Markierungen und **verbinden** der zusammengehörigen Teilregionen, d.h. für jede Region ein **gemeinsames Label** finden.

Beispiel: Schritt 1

Das betrachtete Binärbild enthält drei Objekte, die wir mit Hilfe der sequentiellen Regionenmarkierung finden wollen.

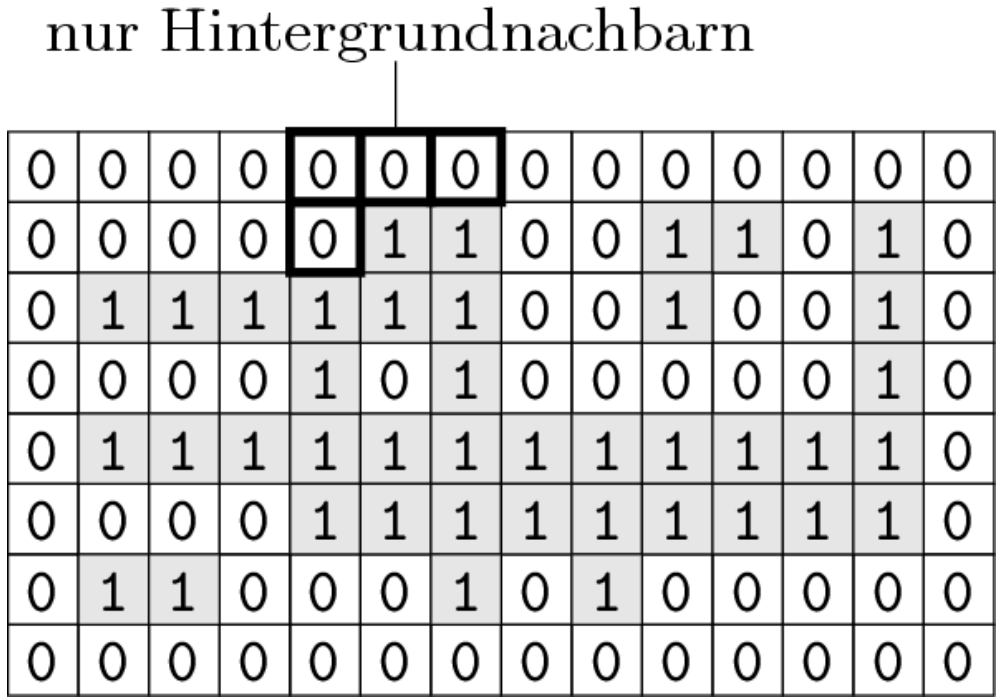
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	1	1	0	1	0
0	1	1	1	1	1	1	0	0	1	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	1	1	1	1	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

0 Hintergrund

1 Vordergrund

Beispiel: Schritt 1

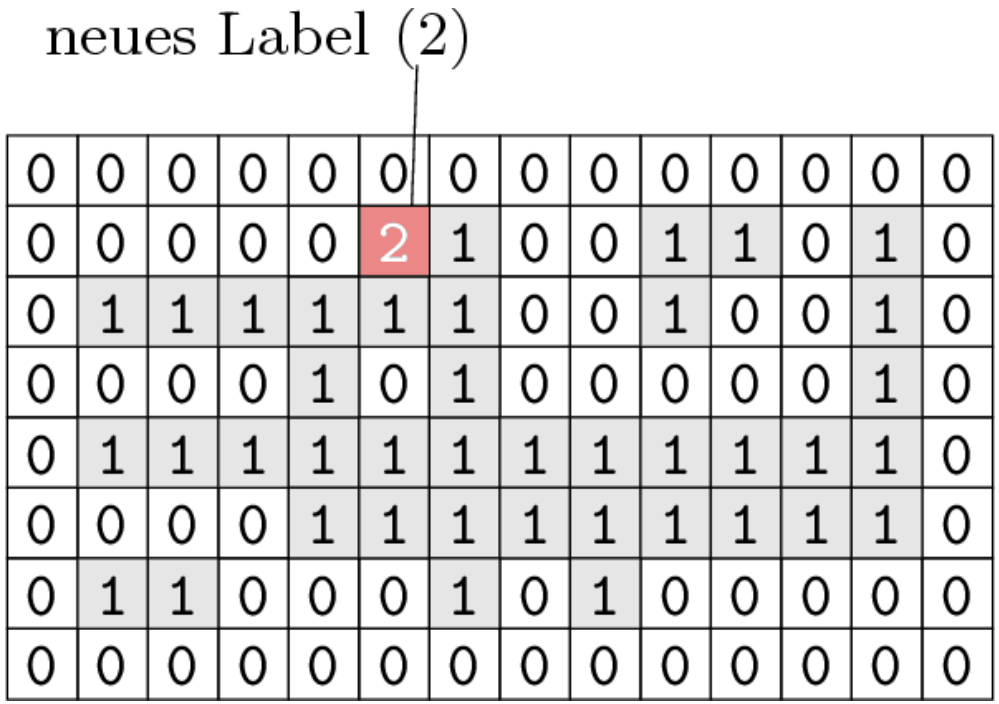
Wir durchlaufen das Bild zeilenweise bis wir zum ersten unmarkierten Vordergrundpixel gelangen. Das hat nur Hintergrundnachbarn, da wir nur die Nachbarn betrachten können, die wir schon besucht haben, also die oben und links.



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

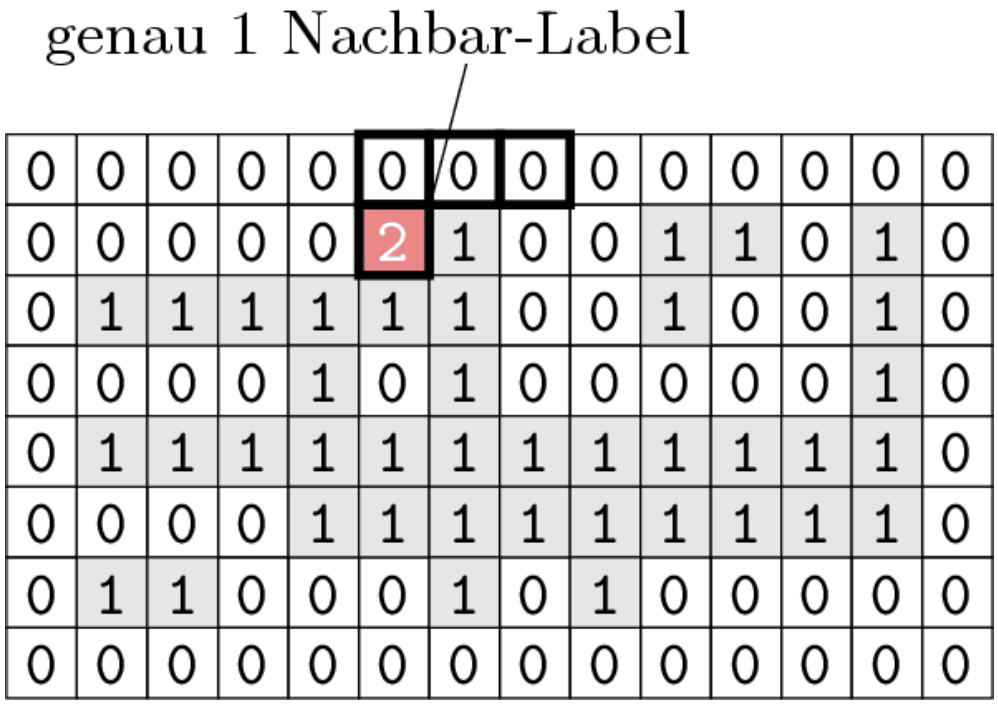
Beispiel: Schritt 1

Wir erzeugen ein neues Label $= 2$ und das Pixel erhält den Wert 2.



Beispiel: Schritt 1

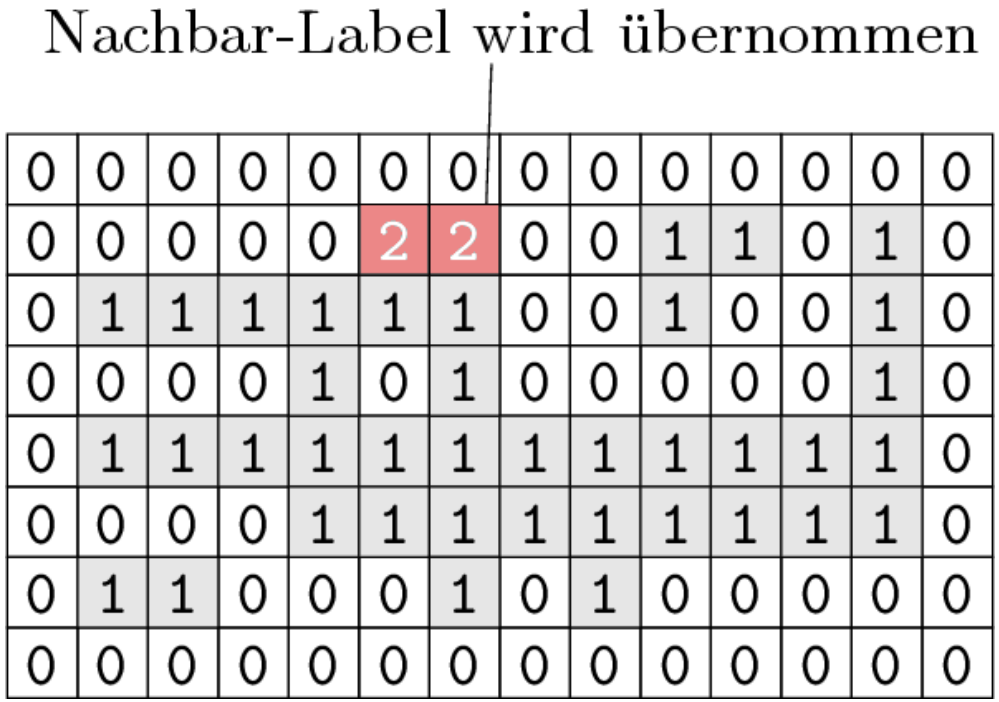
Das nächste Pixel hat genau einen markierten Vordergrundnachbarn.



0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	1	0	0	1	1	0	1	0
0	1	1	1	1	1	1	0	0	1	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	1	1	1	1	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

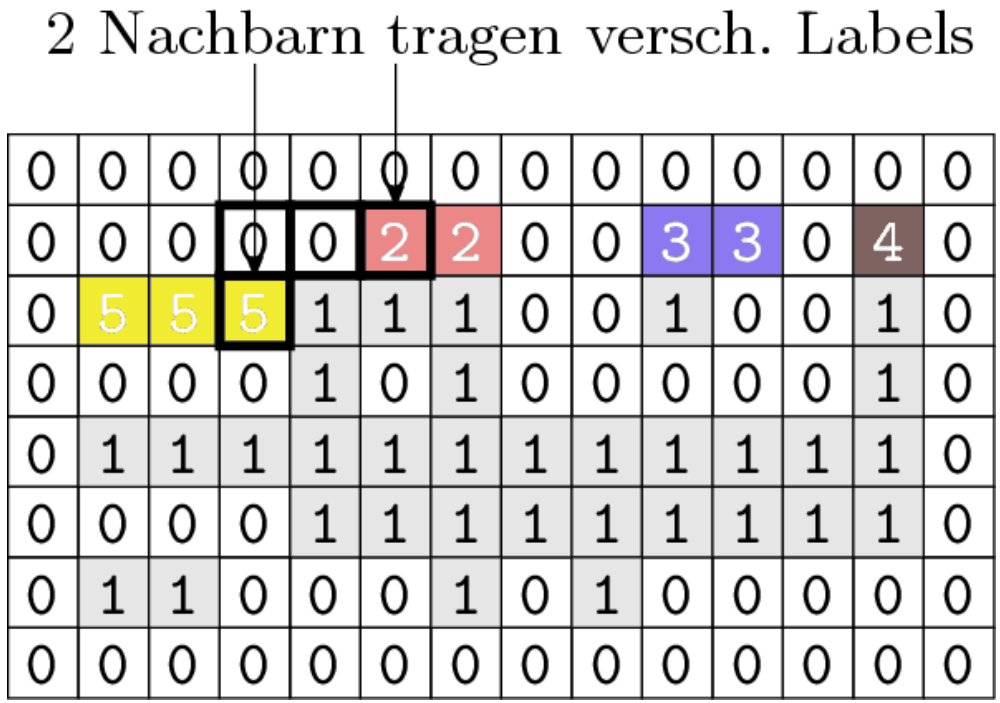
Beispiel: Schritt 1

Wir übernehmen das Label des Vordergrundnachbarn.



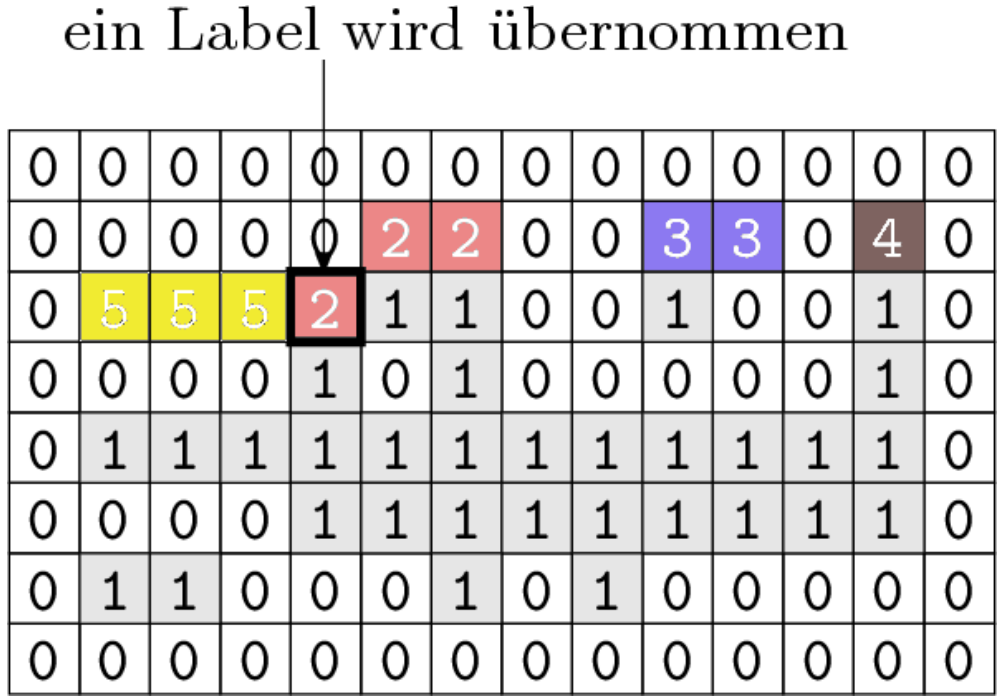
Beispiel: Schritt 1

Wir führen das Verfahren in diesem Sinne weiter, bis wir auf das erste Pixel stoßen, das zwei verschiedene Nachbarpixel hat.



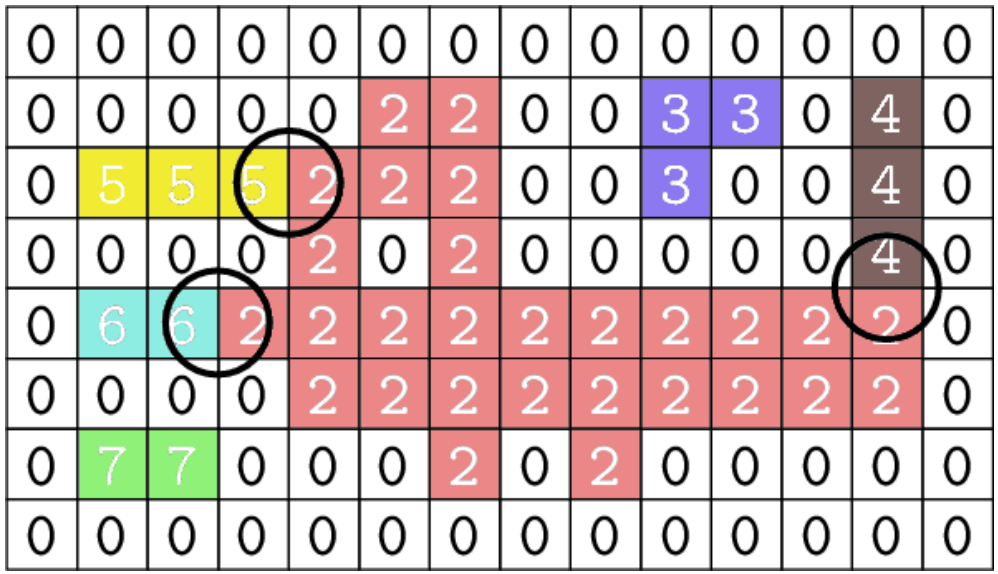
Beispiel: Schritt 1

Wir müssen uns für ein Label entscheiden.



Beispiel: Schritt 2

Haben wir das gesamte Bild so markiert, treten drei Kollisionen im mittleren Objekt auf, die wir beheben müssen.



Beispiel: Schritt 2

So soll das Bild am Ende aussehen.

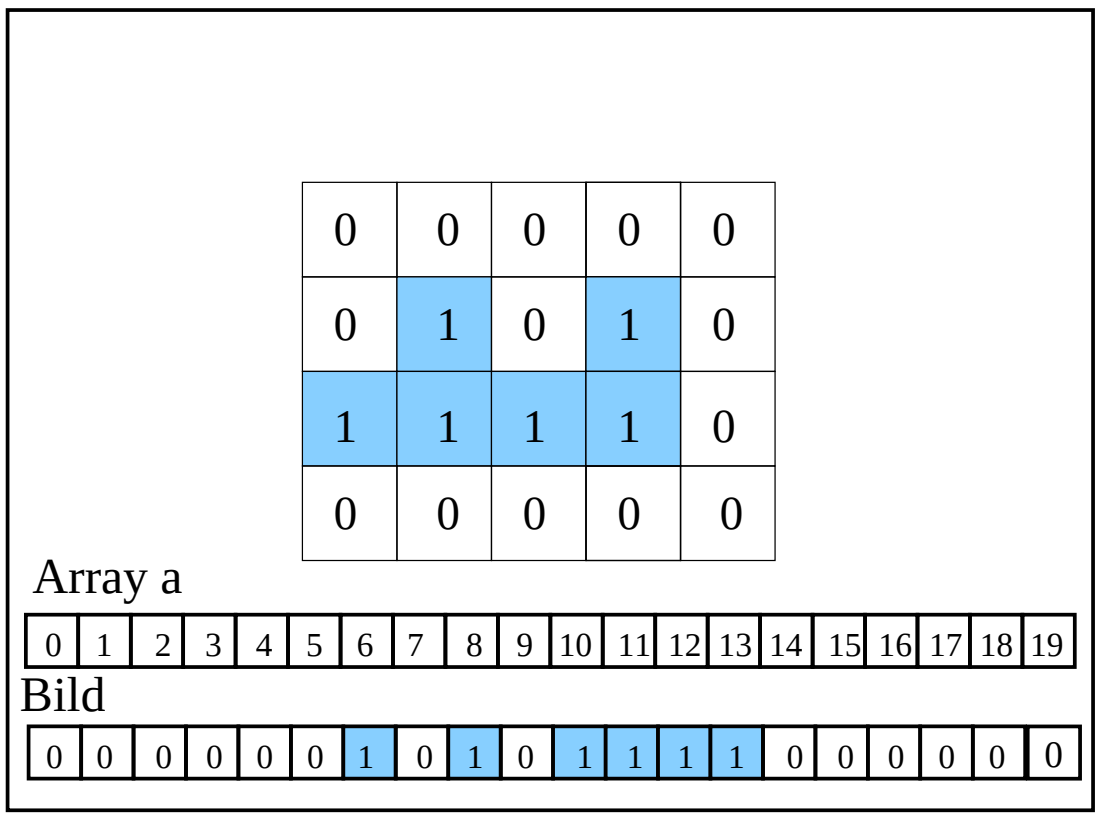
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	2	0	0	3	3	0	2	0
0	2	2	2	2	2	2	0	0	3	0	0	2	0
0	0	0	0	2	0	2	0	0	0	0	0	2	0
0	2	2	2	2	2	2	2	2	2	2	2	2	0
0	0	0	0	2	2	2	2	2	2	2	2	2	0
0	7	7	0	0	0	2	0	2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Union-Find

Eine effiziente Möglichkeit, die Kollisionen zu beheben:
Union-Find
Die Operationen **Union** und **Find** lassen sich sehr einfach auf einer $n-$elementigen Grundmenge mit Hilfe eines Arrays $a[0...n-1]$ implementieren.
Man initialisiert $a[i] = i$, d.h. i ist das **kanonische Element** der Menge $\{i\}$.
PROCEDURE *Init*
  FOR $i := 0$TO $n-1$ DO $a[i] := i$

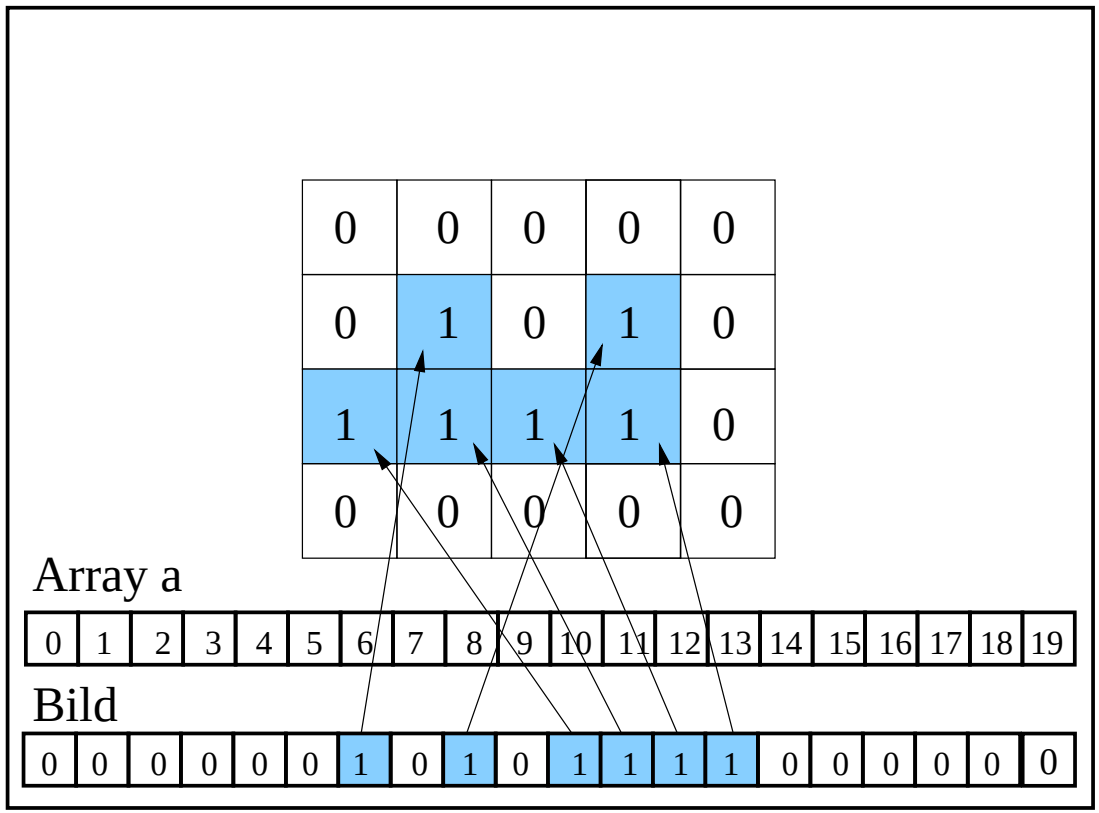
Union-Find

Das Bild wird als Array dargestellt, indem die Zeilen des Bildes zu einem Array aneinandergefügt werden.
Die Elemente des mit $a[i] = i$ initialisierten Arrays gehören dabei zu den entsprechenden darunterliegenden Arrayelementen des Bildarrays.



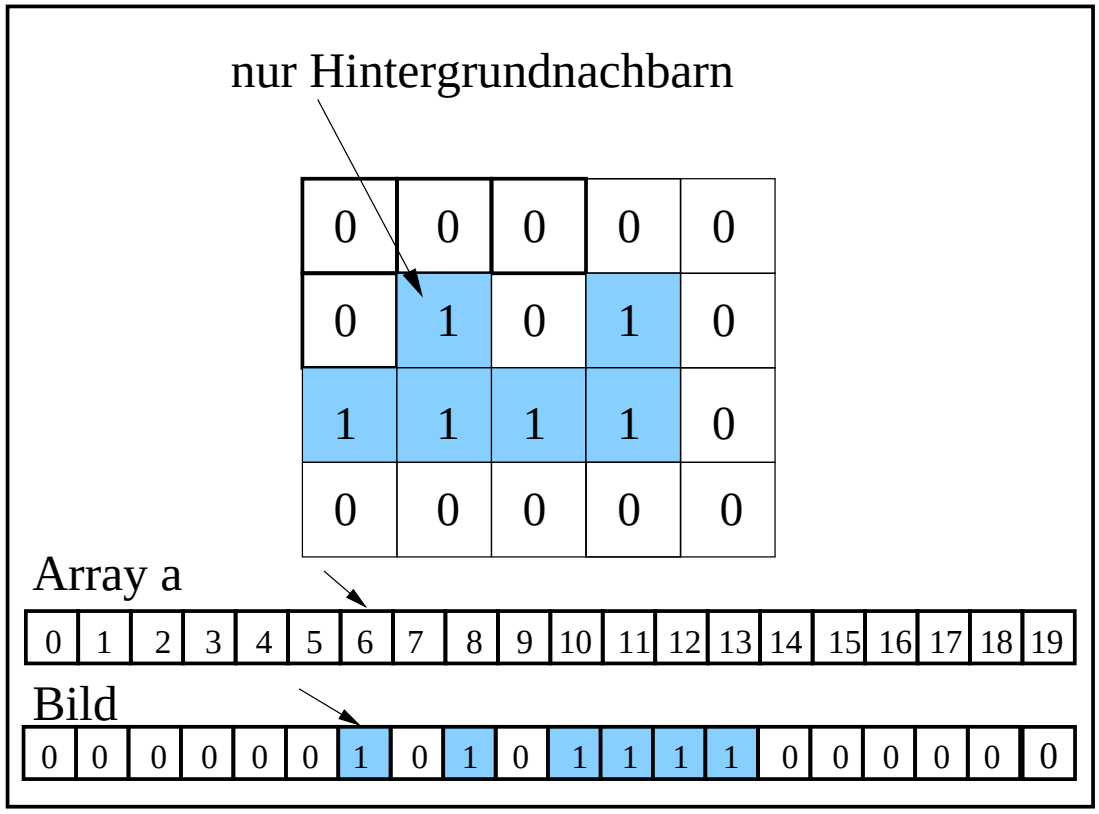
Union-Find

Durch die Pfeile werden die Bildpunkte des Arrays mit den entsprechenden im Bild gekennzeichnet.



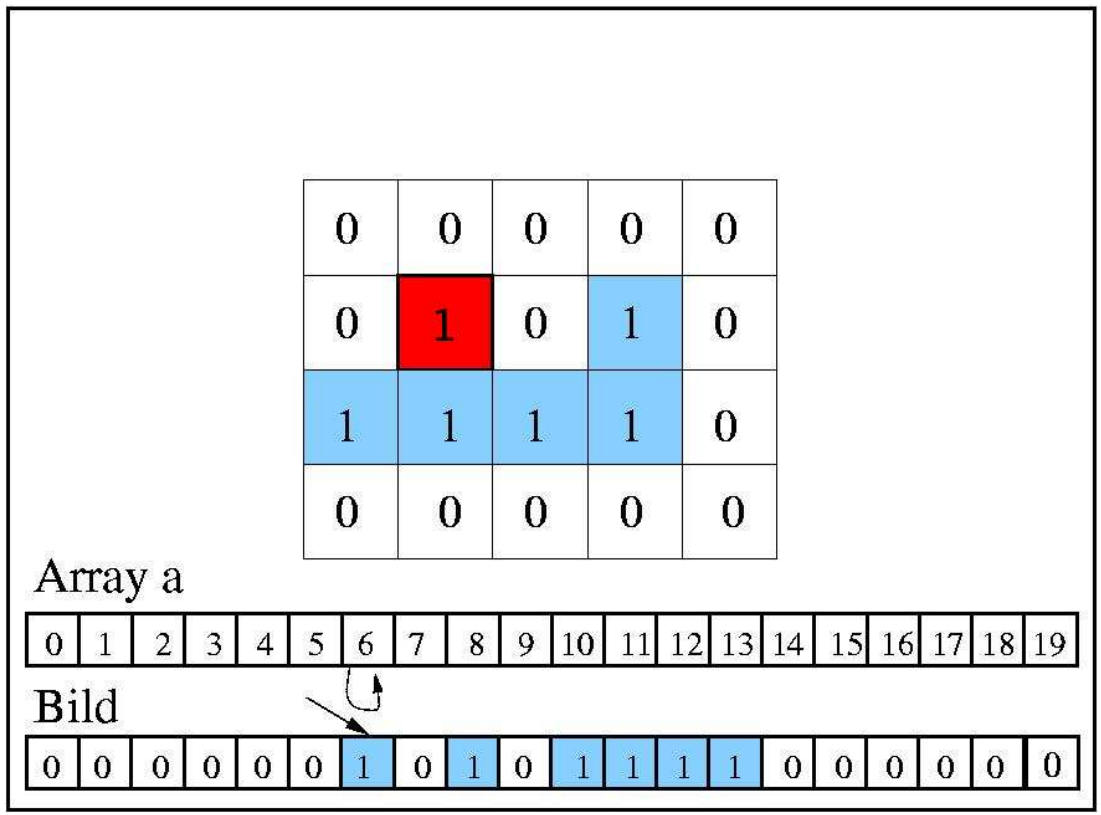
Union-Find

Sequentielles Durchlaufen des Bildes, bzw. der Arrays, bis der erste Pixelwert 1 ist.



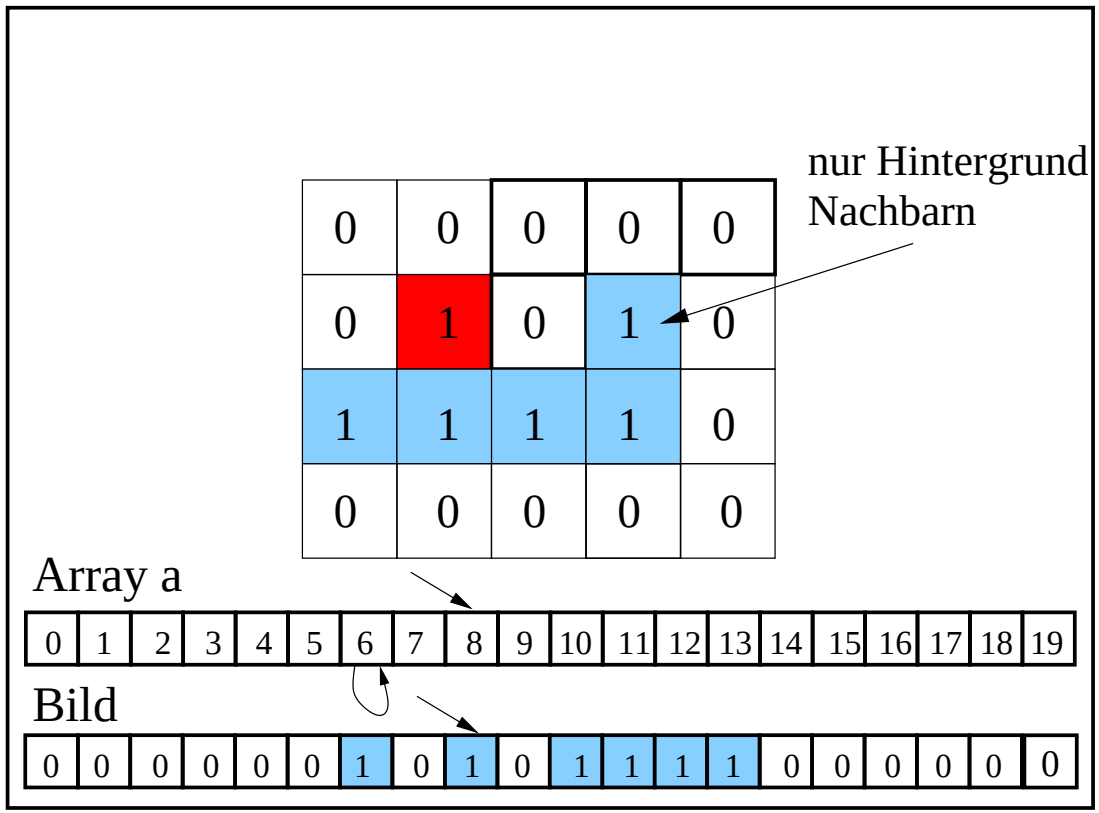
Union-Find

Vorläufiges kennzeichnen des bereits besuchten Pixels mit rot. Das 6. Arrayelement wird durch den Verweis auf sich selbst mit Hilfe des Pfeiles gekennzeichnet.



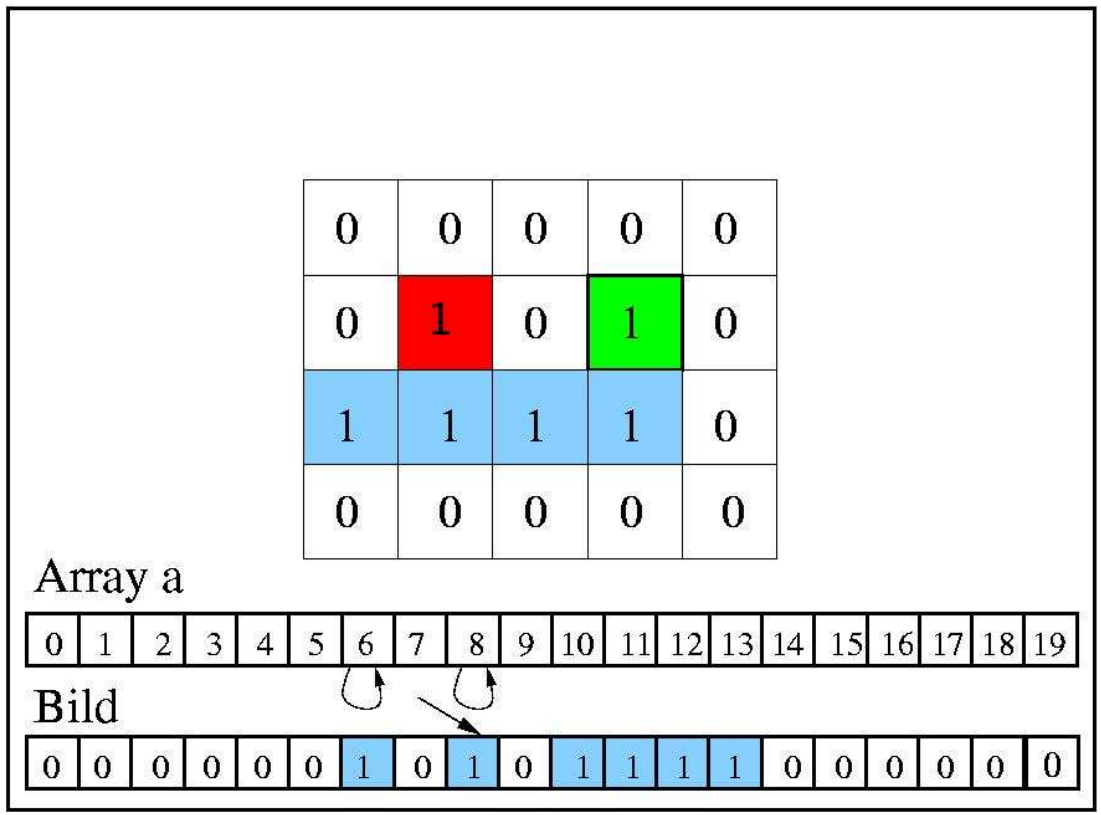
Union-Find

Das nächste noch nicht besuchte Vordergrundpixel hat ebenfalls nur Hintergrundnachbarn.



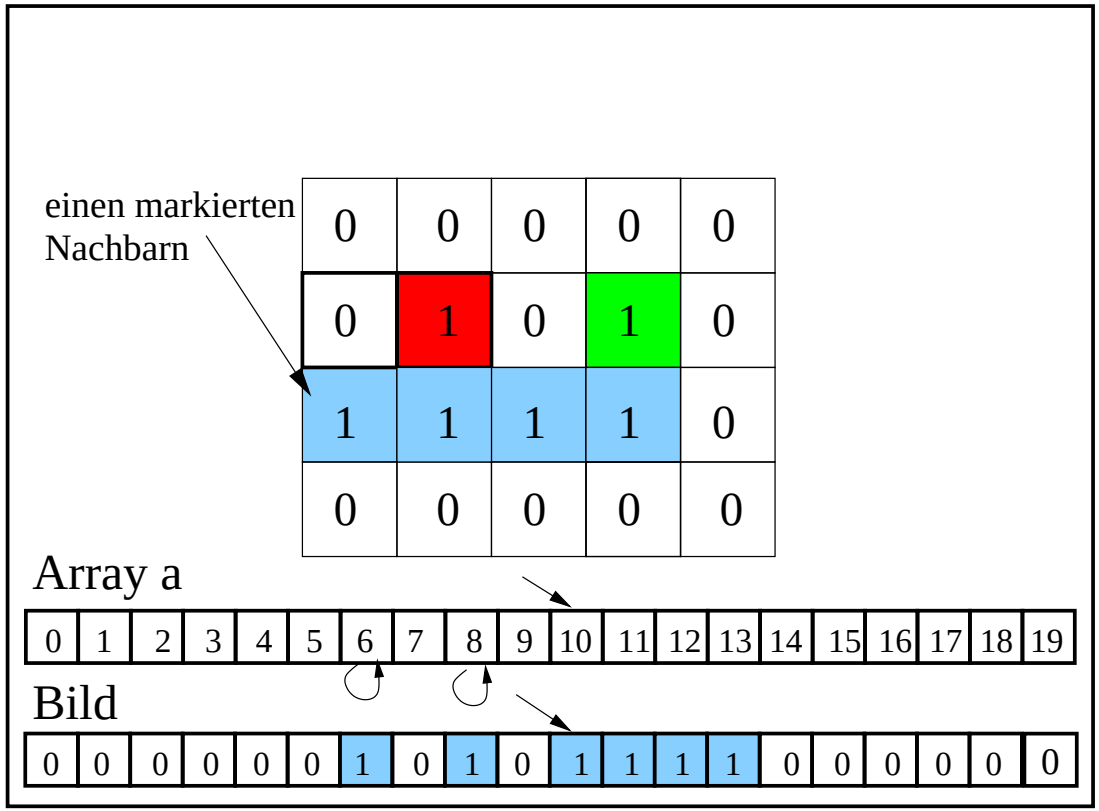
Union-Find

Wir kennzeichnen es mit der neuen Farbe grün und das 8. Arrayelement verweist ebenfalls auf sich selbst, mit Hilfe des Pfeiles gekennzeichnet.



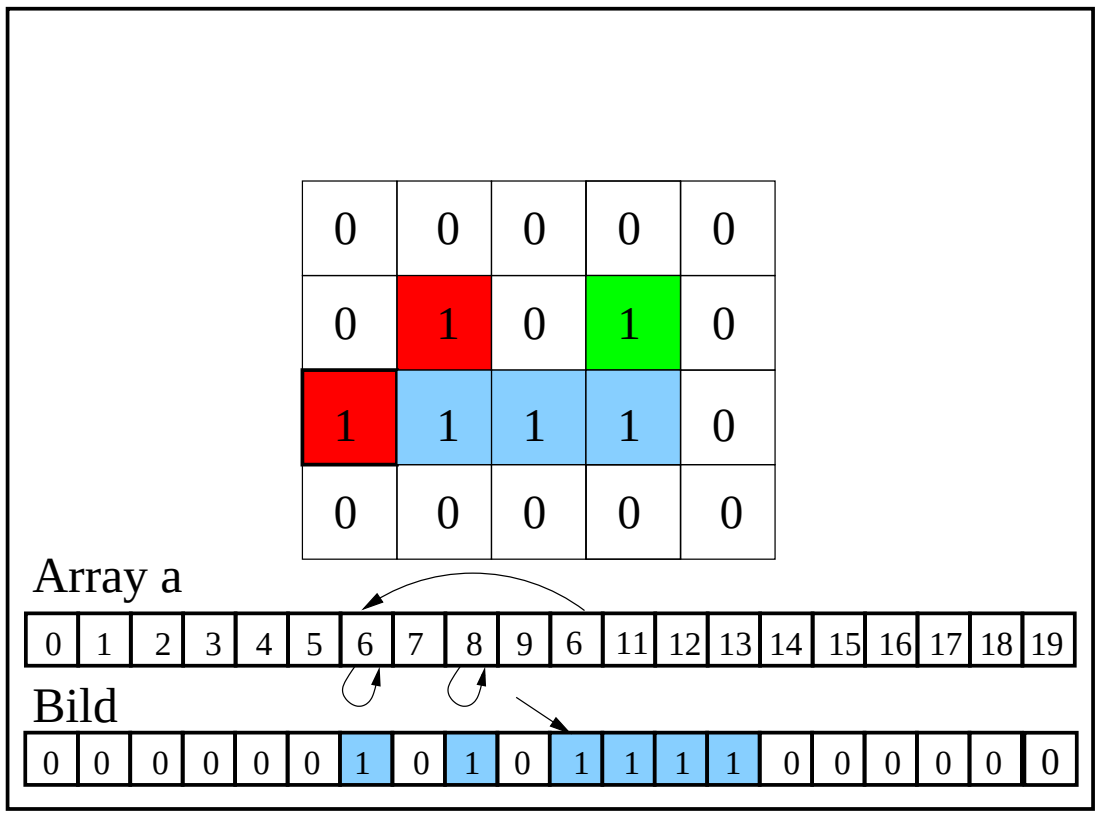
Union-Find

Hier haben wir genau einen roten Vordergrundnachbarn.



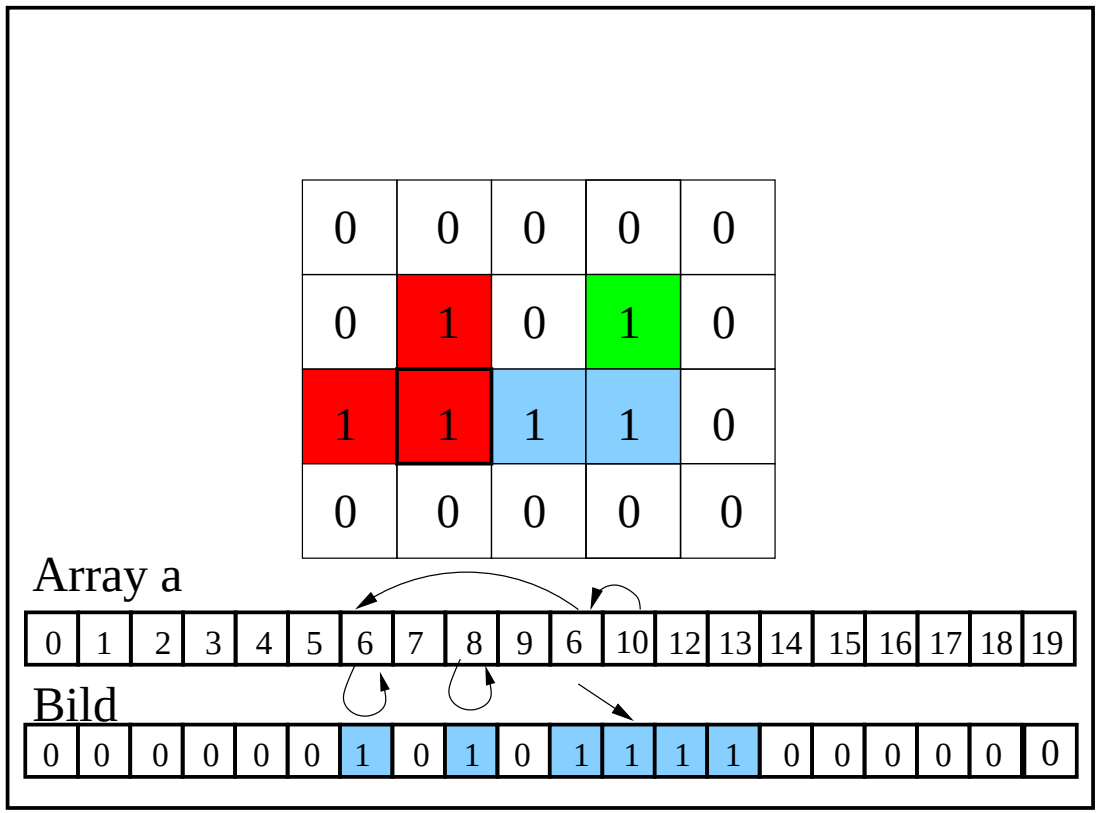
Union-Find

Wir verweisen vom 11. Arrayelement auf das 6. da die beiden Elemente offensichtlich zu einer Region gehören.



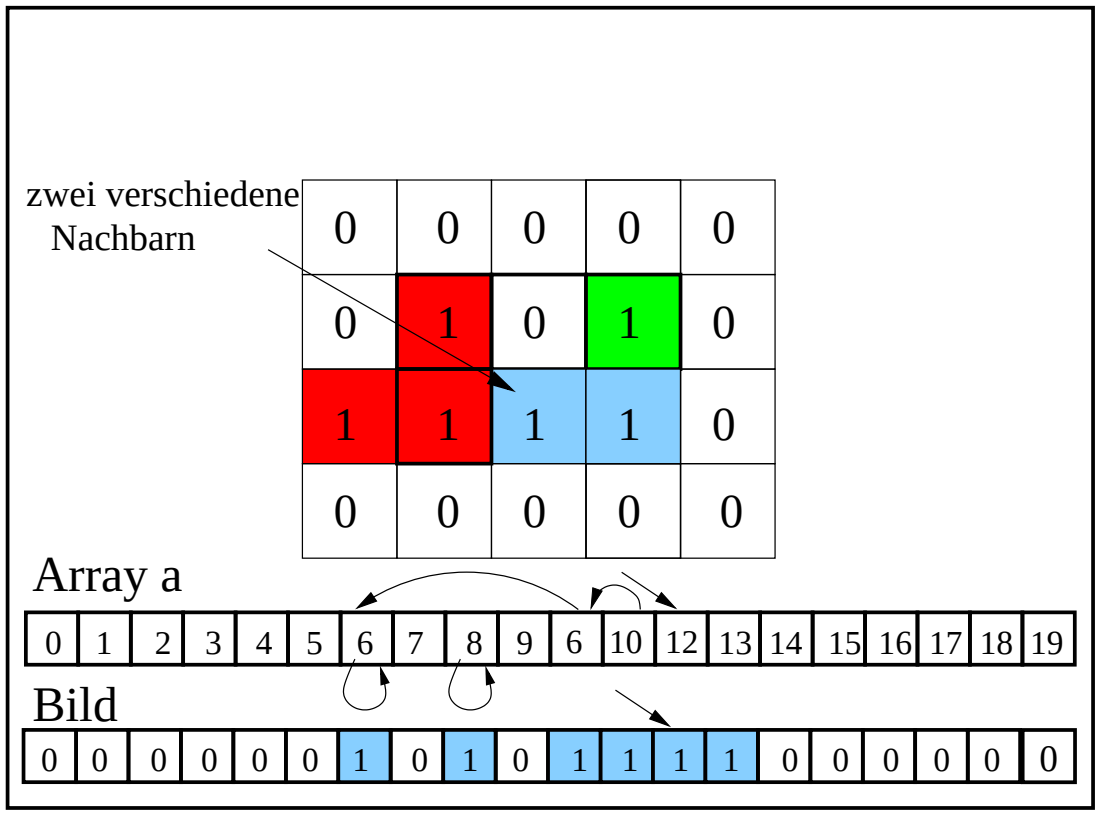
Union-Find

Hier verweisen wir vom 12. Arrayelement auf das 11., da das jetzt betrachtete Pixel zu der roten Region gehört.



Union-Find

Es tritt wieder die Situation zweier verschiedener Nachbarn auf. Nun brauchen wir die Operationen Union und Find.



Union-Find

PROCEDURE *Find(i)*
  IF $i = a[i]$ THEN RETURN i
  ELSE
    $a[i] := Find(a[i])$
    RETURN $a[i]$

Man findet das **kanonische Element** der Menge, die das Element i enthält, durch u.U. mehrfaches sondieren von $a[i], a[a[i]]$, ...bis für ein k gilt $a[k] = k$, d.h. das kanonische Element bildet immer die **Baumwurzel**.
Durch den Befehl $a[i] = Find(a[i])$ wird die **Verzweigungsstruktur** des Baumes so **umgebaut**, dass sich die Wege bis zur Wurzel **verkürzen**, denn in alle angesprochenen $a[i]$-Werte wird ein direkter Verweis auf die Wurzel eingetragen.

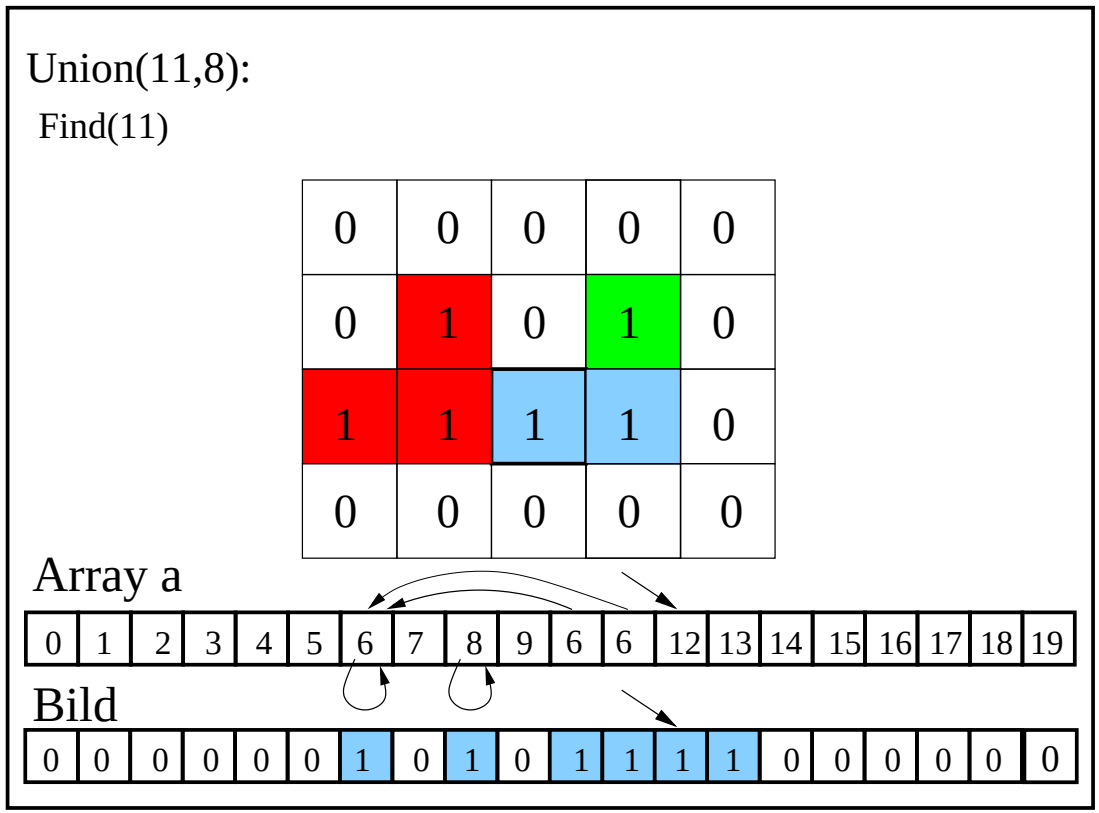
Union-Find

PROCEDURE *Union(i,j)*
  $c1 = Find(i)$
  $c2 = Find(j)$
  RANDOM z IN $\{0, 1\}$
  IF $z = 0$ THEN $a[c1] := c2$
  ELSE $a[c2] := c1$

Bei der **Union-Operation** entscheidet der **Zufall**, ob die Menge i an die Menge j gekoppelt wird oder umgekehrt. Statt den Zufall entscheiden zu lassen, kann man auch die **Anzahl der Elemente** einer Menge speichern und bei der Union-Operation die kleinere an die größere Menge hängen.

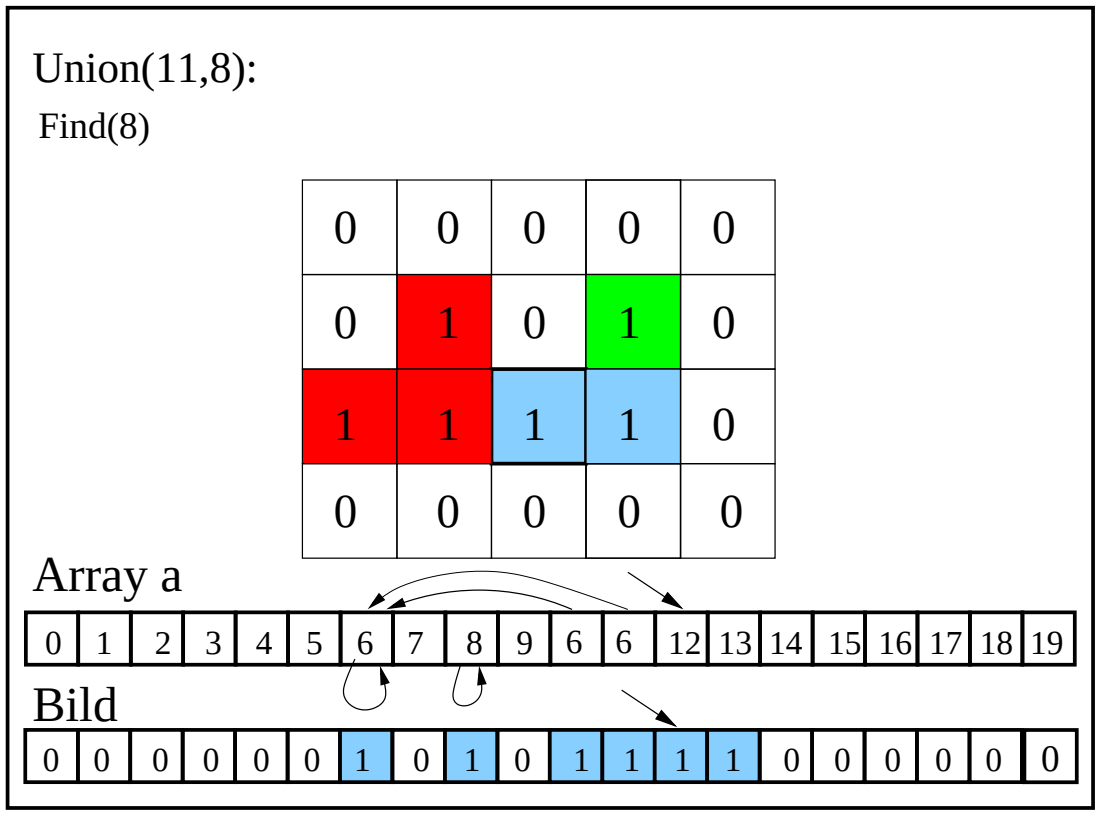
Union-Find

Durch die Find-Operation verweisen nun alle angesprochenen Elemente der Menge 6 auf das kanonische Element.



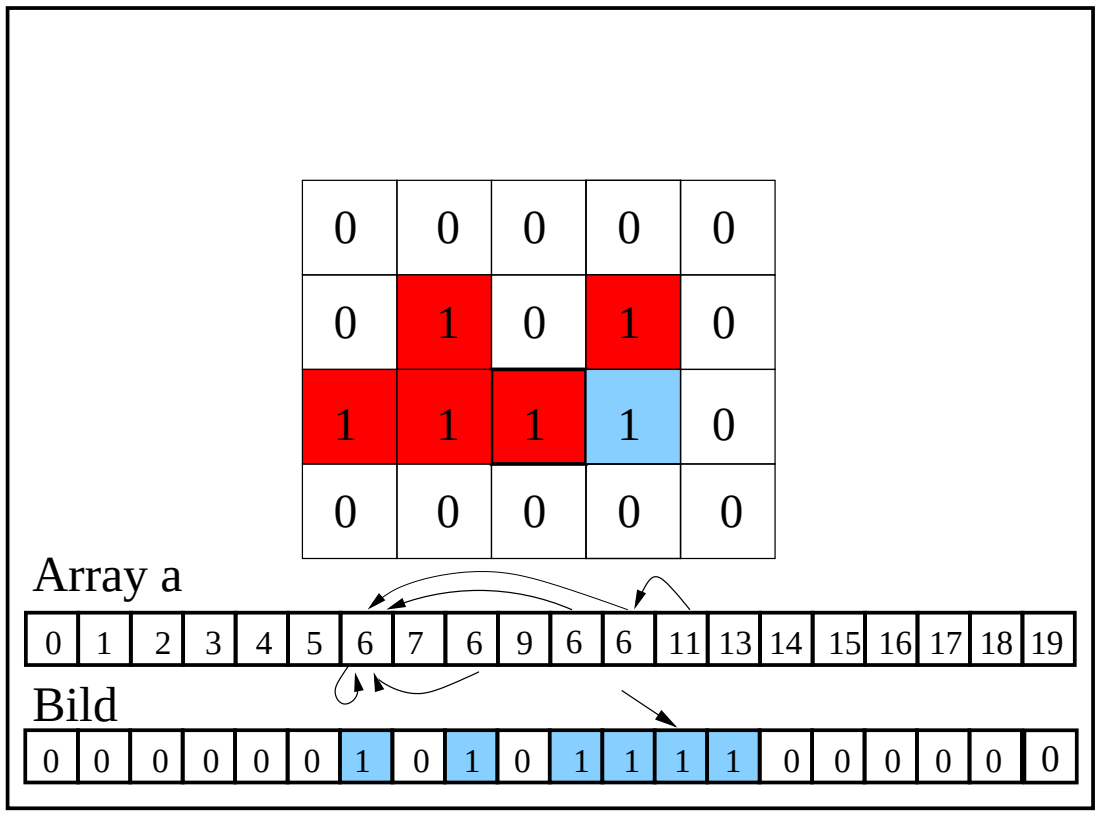
Union-Find

Durch Find(8) verändert sich nichts, da es nur ein Element der grünen Menge gibt, d.h. 8 ist bereits das kanonische Element.



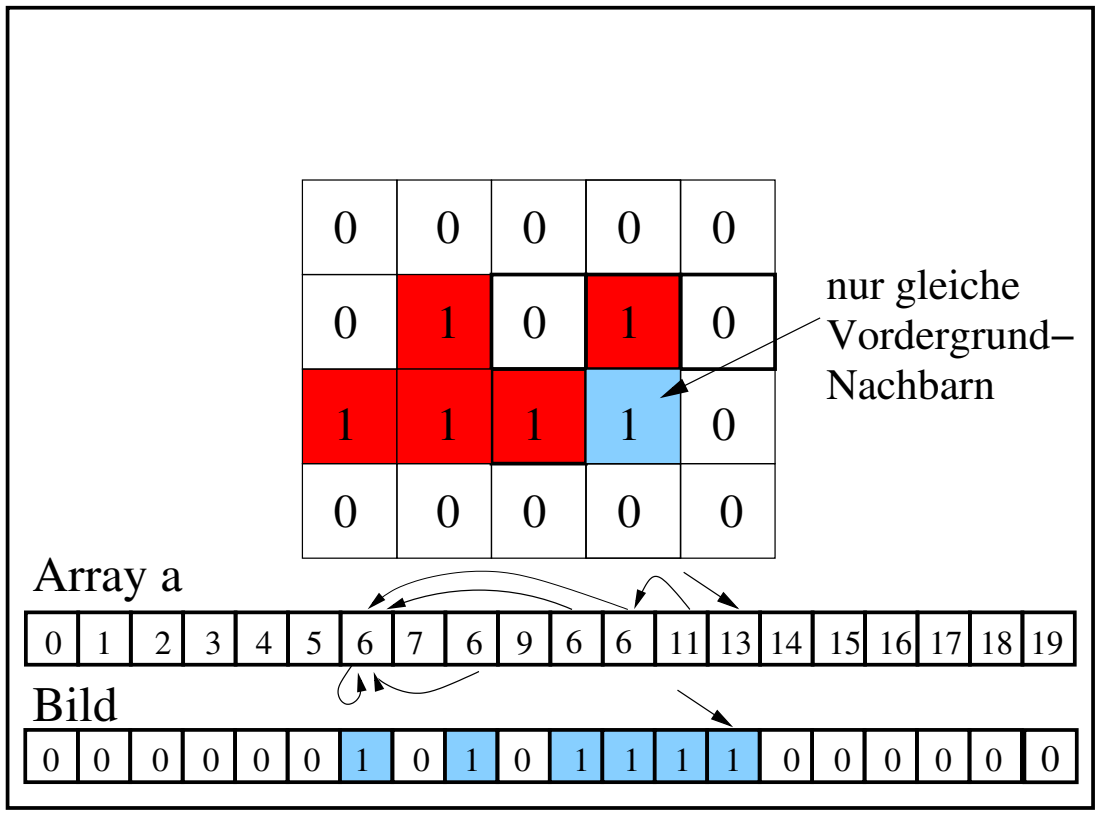
Union-Find

Durch *Union*(11, 8) werden die beiden Mengen vereint, d.h. das 8. Element verweist nun auch auf das 6. Das betrachtete 12. Element verweist wie gehabt auf das 11.



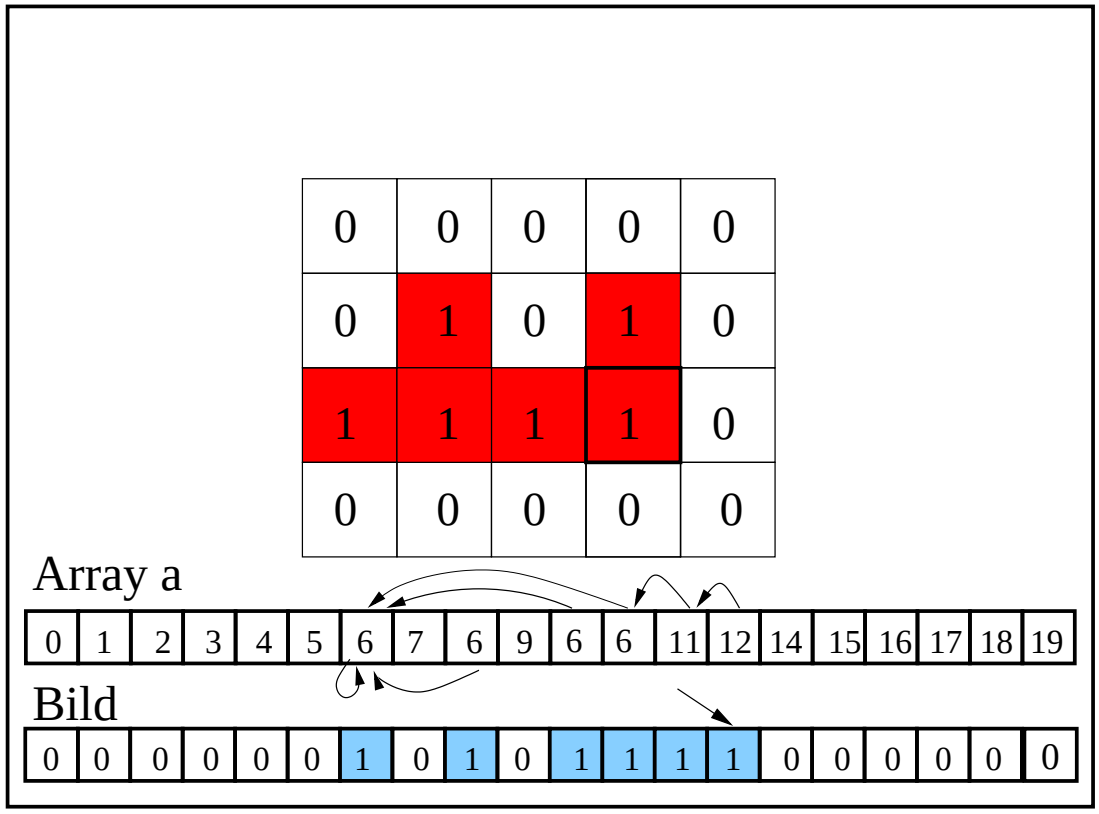
Union-Find

Das letzte Vordergrundpixel hat wieder nur Vordergrundnachbarn gleicher Farbe.



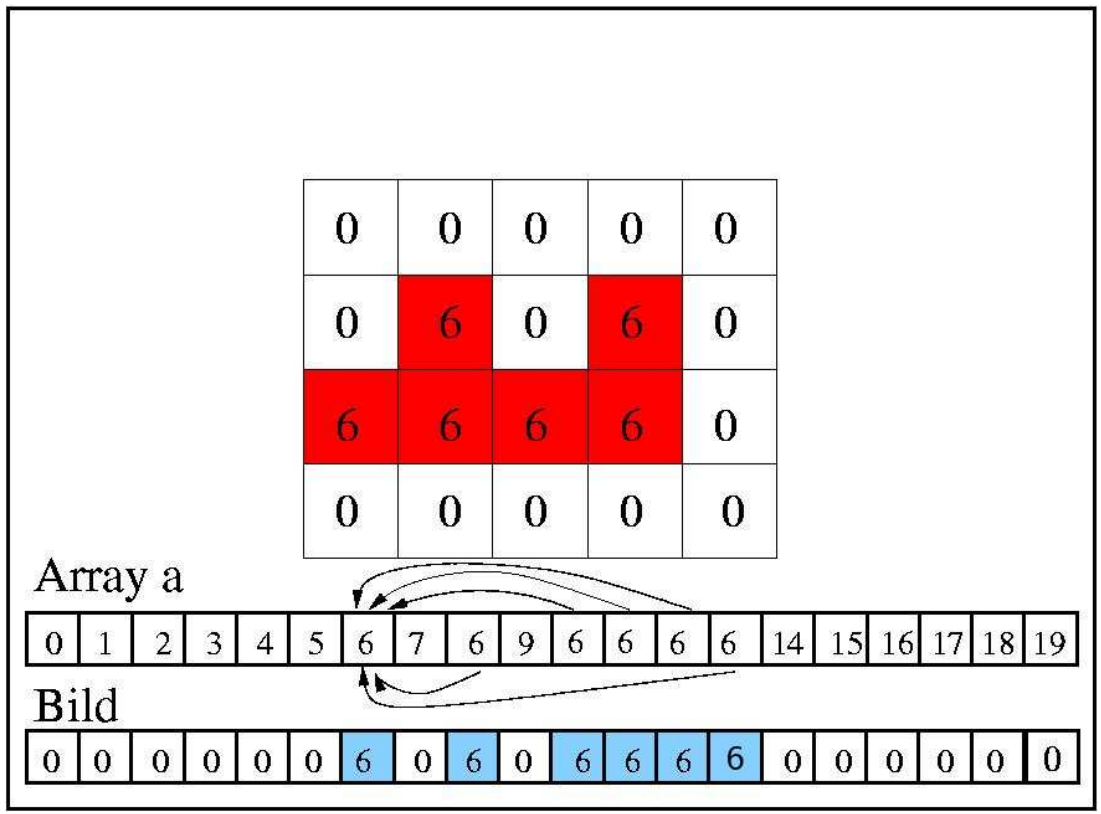
Union-Find

Es bekommt einen Verweis auf das vorhergehende und die rote Farbe.



Union-Find

Durch eine letzte Find-Operation verweist nun jedes Element des Objektes auf das kanonische Element. Wir können nun das Bildarray durchlaufen und bei jeder 1 den Wert des Arrays *a* an der jeweiligen Stelle eintragen.



Union-Find

Es wird eine **Menge von Daten** verwaltet, in unserem Fall die Menge der Bildpunkte $\{v_1, ..., v_n\}$ mit $v_i = (x_j, y_k), j \epsilon \{1, ..., m\}, k \epsilon \{1, ...s\}$eines Bildes der Größe $m \cdot s$ mit $n = m \cdot s$.
Zunächst versteht man jeden Bildpunkt v_i als **einelementige Menge**, d.h. jedes Element v_i zeigt auf sich selbst.

Union-Find

Durch die Operation **Union** werden die Mengen v_i so **vereinigt**, dass ein **System von Teilmengen** von $\{v_1, ..., v_n\}$ entsteht.
Die Funktionen **Union** und **Find** sind dabei wie folgt definiert:

- *Find*(v_i): liefert das **kanonisches Element der Menge** zurück, in der sich der Punkt v_i befindet.
- *Union*(v_i, v_j): **vereinigt** die beiden Mengen, die v_i und v_j enthalten.

Union-Find

Die **Komplexität** einer beliebigen Folge von n Union- und Find-Operationen lässt sich durch eine **Amortisationsanalyse** mit $O(nlog^*n)$ abschätzen, wobei

$$log^*n = min\{k | \underbrace{log...log}_{k-mal} n \leq 1\}.$$

log^*n wächst sehr sehr langsam: für $n \leq 10^{19728}$ ist $log^*n \leq 5$, d.h. die amortisierte Komplexität ist so gut wie **konstant**.

Kombiniertes Verfahren

IDEE: Verbinden von sequentieller **Regionenmarkierung** und traditioneller **Konturverfolgung**, d.h. in einem Bilddurchlauf werden Regionen markiert und Konturen gefunden.

Vorteile:

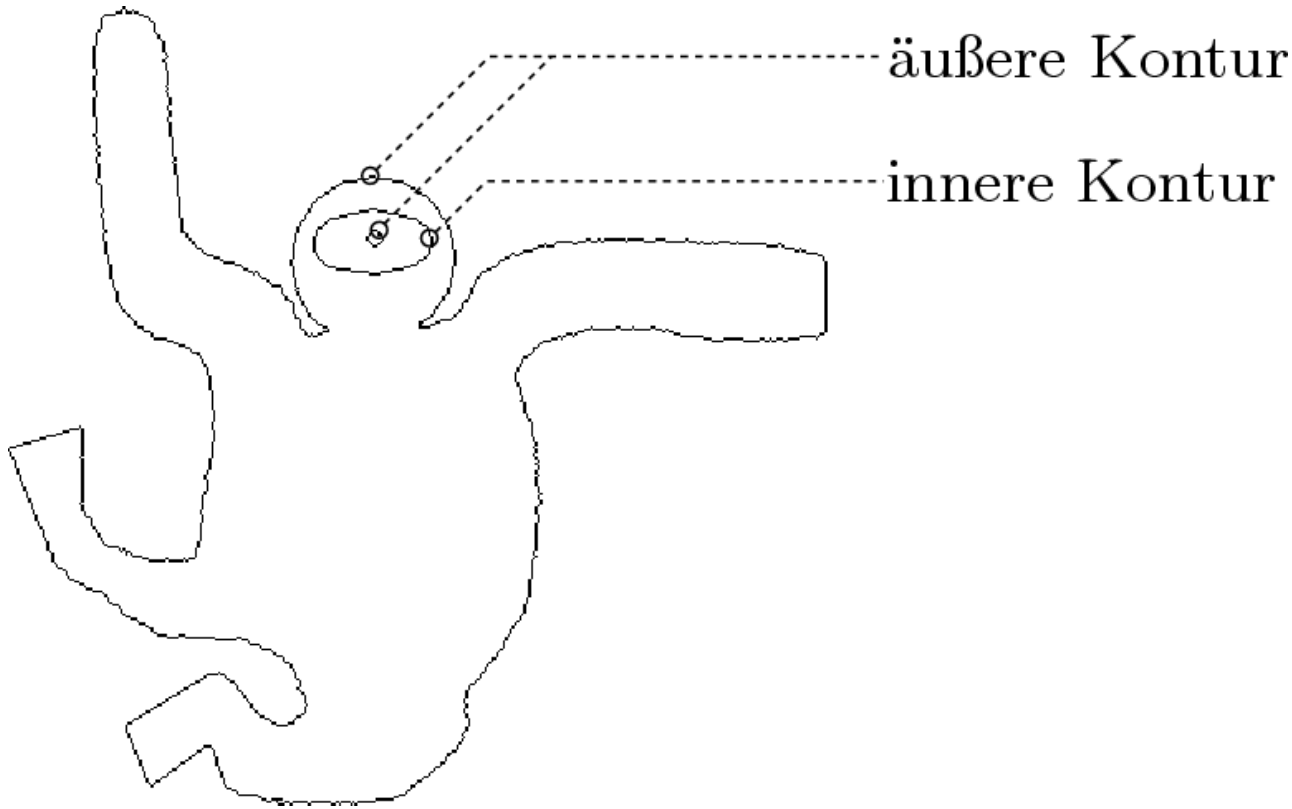
- nur **ein Bilddurchlauf**
- **Identifizierung** von äußeren und inneren Konturen
- **keine** komplexe Datenstruktur
- sehr **effizient** im Vergleich zu ähnlichen Verfahren

Kombiniertes Verfahren

Beispiel für ein Binärbild mit äußeren und inneren Konturen:

Kombiniertes Verfahren

Dabei sind innere und äußere Konturen die beschriebenen



Kombiniertes Verfahren

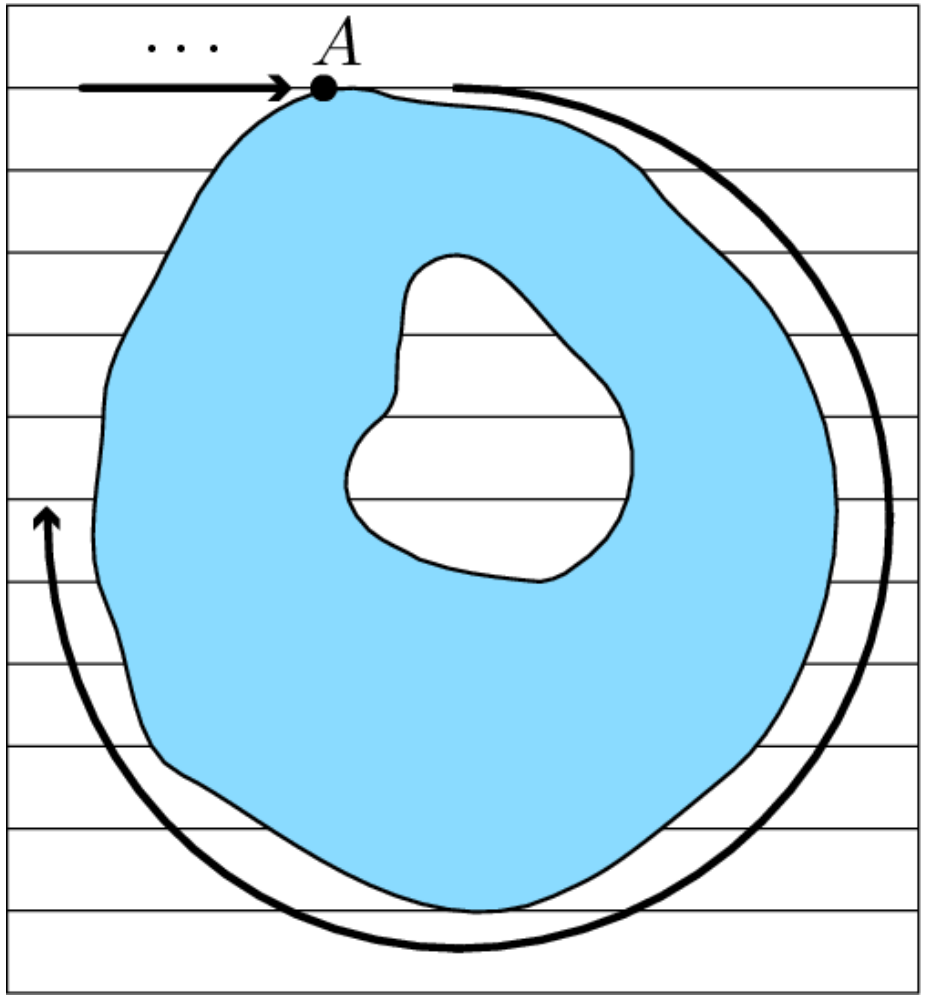
Algorithmus:

1. **Durchlaufen** des Binärbildes von links oben nach rechts unten und **Markierung** der Pixel. (Entsprechend der sequentiellen Regionenmarkierung)
2. Betrachtung der **aktuellen Bildposition** (x, y).
   Mögliche Situationen:

Kombiniertes Verfahren

- Übergang eines **Hintergrundpixels** zu einem nicht markierten **Vordergrundpixel A**.
  $\Rightarrow$ **A** liegt am **Außenrand** einer neuen Region.
  $\Rightarrow$ **Neues Label** erzeugen und die zugehörige **äußere Kontur umfahren** und **markieren**. Alle unmittelbar **angrenzenden Hintergrundpixel** mit dem Wert **-1** markieren.

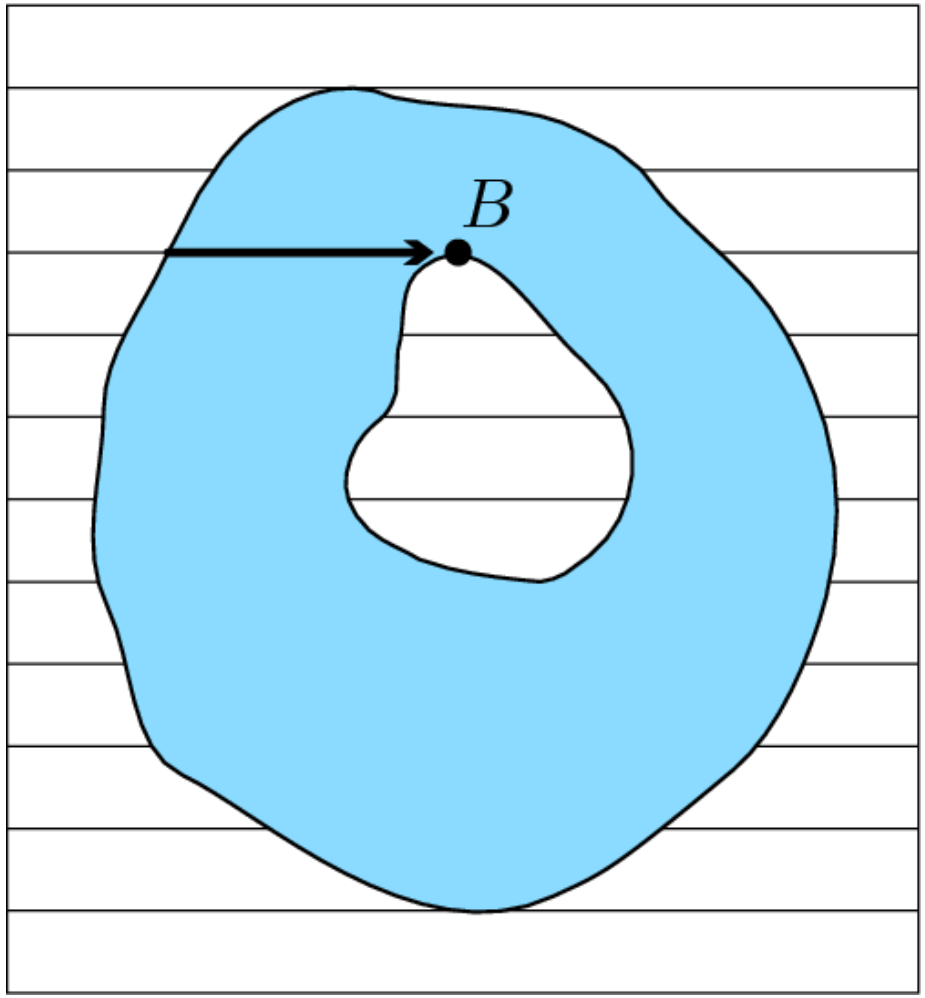
Kombiniertes Verfahren



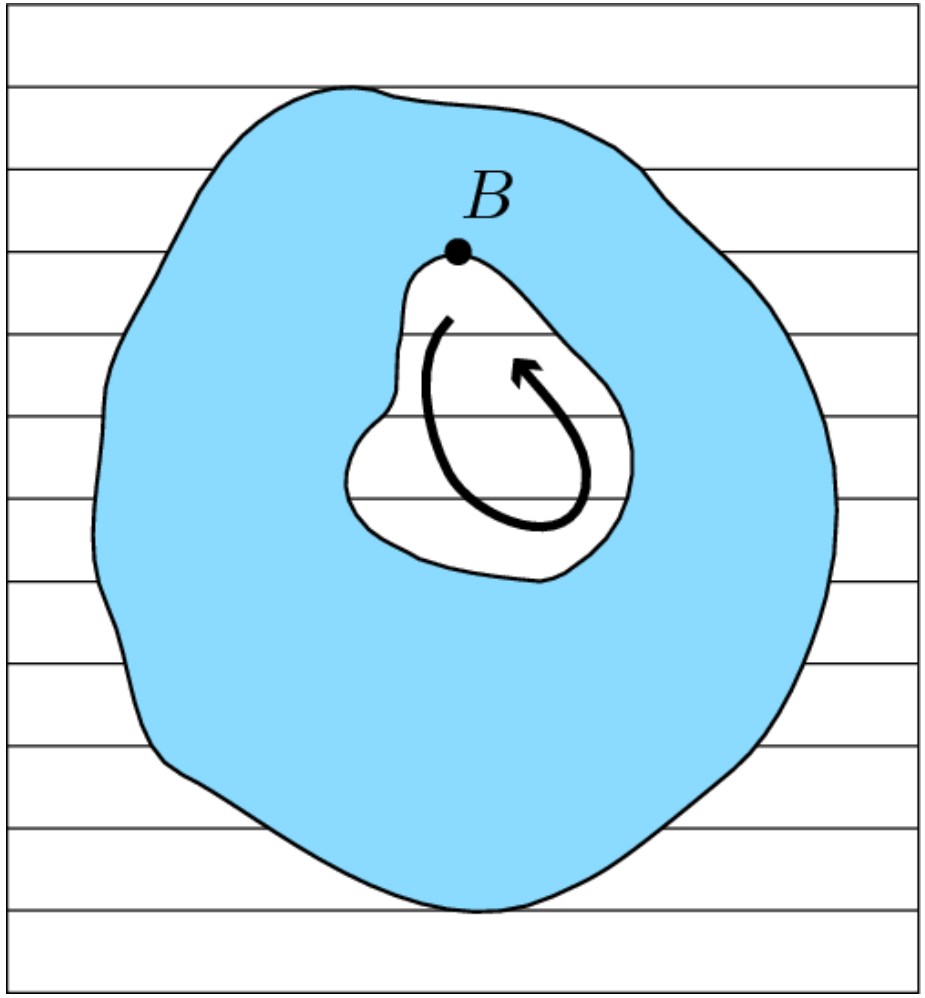
Kombiniertes Verfahren

- Übergang eines **Vordergrundpixels B** auf ein nicht markiertes **Hintergrundpixel**.
  $\Rightarrow$ **B** liegt am **Rand** einer **inneren Kontur**.
  $\Rightarrow$ ausgehend von **B** die **innere Kontur umfahren** und deren Pixel mit dem **gleichen Label** wie B markieren.
  Alle **angrenzenden Hintergrundpixel** bekommen den Wert -1.

Kombiniertes Verfahren



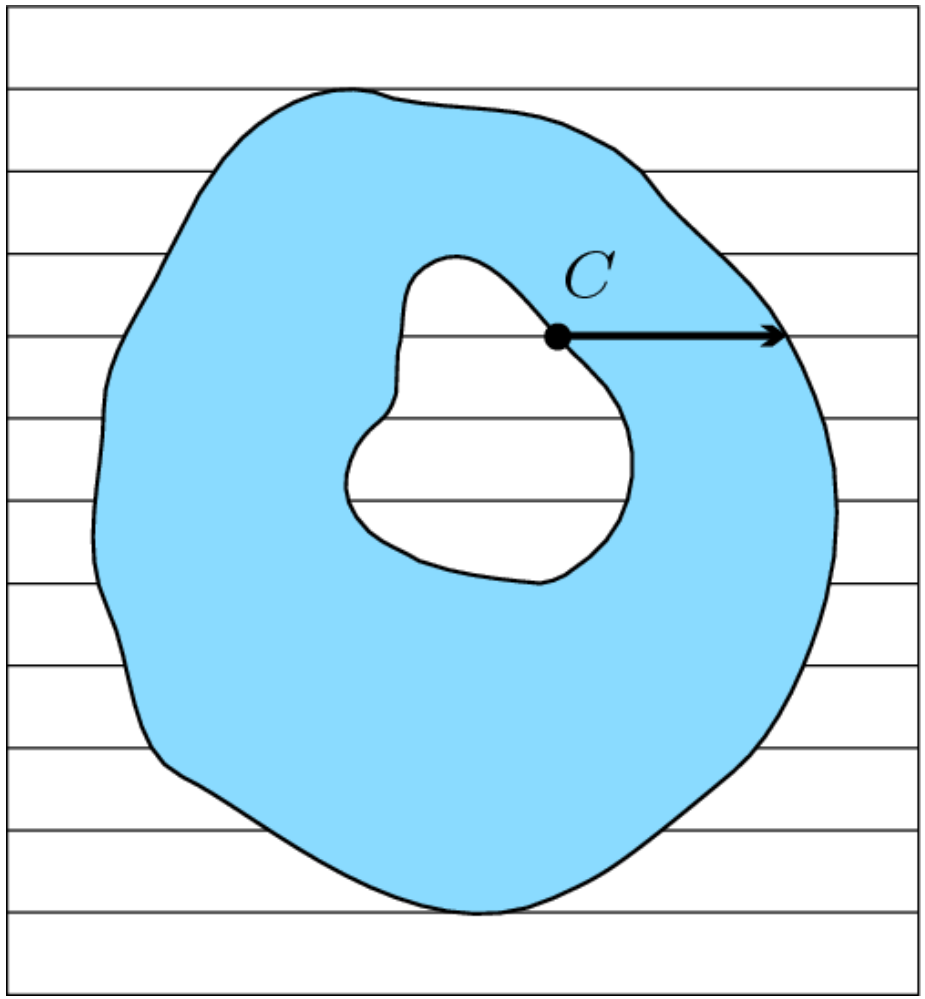
Kombiniertes Verfahren



Kombiniertes Verfahren

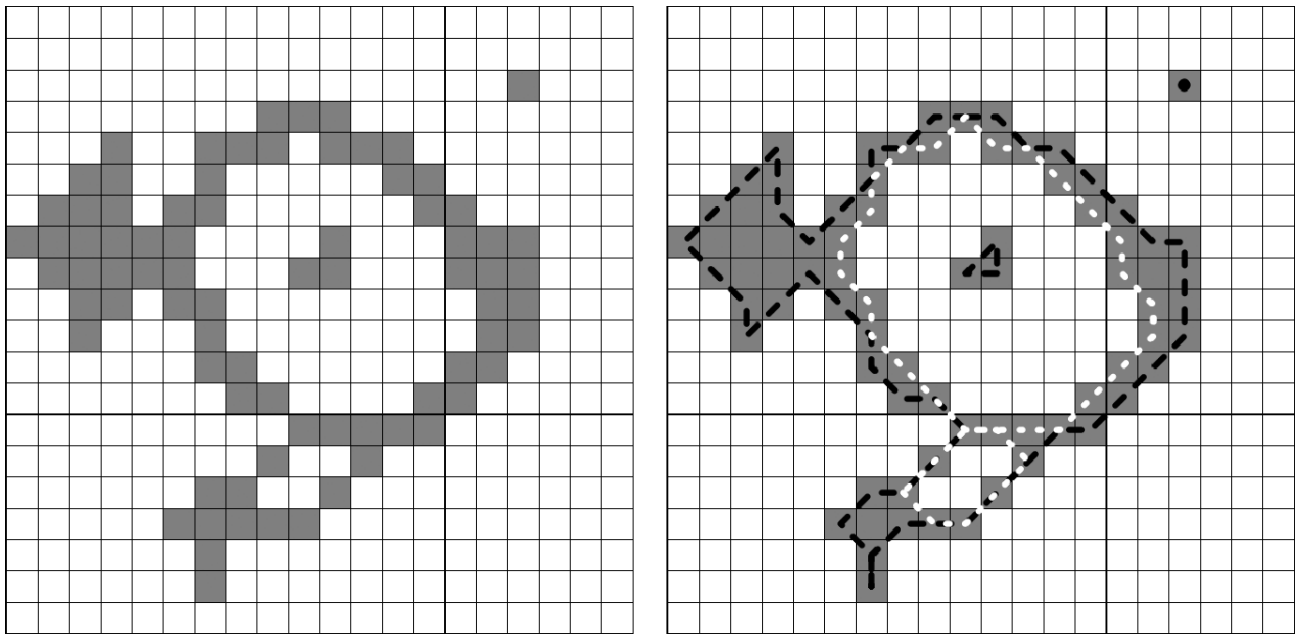
- Übergang eines **Vordergrund**- auf ein **Vordergrundpixel**
  $\Rightarrow$ **Markieren** des neuen Pixels mit dem **Label** des **vorhergehenden**.

Kombiniertes Verfahren



Beispiel

Bei diesem Bild bezeichnen die schwarzen Linien die äußeren Konturen, die weißen die inneren Konturen. Innere und äußere Konturen können sich bei z.T. auch überschneiden.



Grauwertbilder

Definition

Grauwertbilder sind Bilder, die durch die Funktion

$$B : \mathbb{N}x\mathbb{N} \rightarrow [0, g_{max}]$$

beschrieben werden können. Ebenso kann ein **RGB-Farbbild** (red-green-blue) in seine drei Farbanteile zerlegt und durch B beschrieben werden.

Aufgabe:

- Unterscheidung zwischen **Objekten** und **Hintergrund**
- Auffinden der **Begrenzungslinien**

Konturenverstärkung durch Differenzenbildung

Ansatz: Um **Kanten** zu erkennen wendet man **Masken** an:

1. Eine **quadratische** $(2n+1)\,x\,(2n+1)$ **Matrix** M, **Maske**, wird **zentriert** über den betrachteten **Bildpunkt** gelegt.
2. **Multiplizieren** der Grauwerte des Bildes mit den Matrixelementen
3. Die **Summe dieser Produkte** ergibt den neuen Bildwert.

Konturenverstärkung

Mögliche **Masken** zur Auffindung von **Konturen** in **x- und y-Richtung** sind:

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix},$$

Konturenverstärkung

Masken zur Auffindung von Konturen in **Diagonalenrichtung** sind:

$$D_1 = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}, D_2 = \begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix}$$

Konturenverstärkung

Wie erhält man diese Masken?
⇒Differenzenbildung:
Wir interpretieren das Grauwertbild als eine **differenzierbare Funktion**

$$B : \mathbb{R}^2 \to \mathbb{R}$$

Betrachtung des **Gradienten** $(B_x, B_y)^T$ in einem Punkt (x, y).

- $(B_x, B_y)^T$ ist **betragsmäßig groß** $\Rightarrow (x, y)$ ist ein **Kantenpunkt**

Konturenverstärkung

Bildung der Differenzen über die **angenäherten partiellen Ableitungen** von B_x und B_y:

$$B_x = \frac{\Delta B}{\Delta x} = 0.5(B(x+1, y) - B(x-1, y))$$

$$B_y = \frac{\Delta B}{\Delta y} = 0.5(B(x, y+1) - B(x, y-1))$$

Konturenverstärkung

Nehmen wir eine **größere Umgebung** für die Berechnung der Ableitungen, erhalten wir folgende Gleichung:

$$B_y = \frac{\Delta B}{\Delta y} = \frac{1}{8}((B(x-1, y+1) - B(x-1, y-1))$$

$$+2(B(x, y+1) - B(x, y-1)) + (B(x+1, y+1) - B(x+1, y-1))).$$

Dies entspricht genau der Anwendung von G_y im Punkt (x, y) multipliziert mit einem Faktor $\frac{1}{8}$.
Entsprechend lassen sich die Masken G_x, D_1 und D_2 ermitteln.

Konturenverstärkung

Zur **Kantenverstärkung** lässt sich folgende Maske bestimmen:

$$\frac{\partial^2 B}{\partial x^2}+\frac{\partial^2 B}{\partial y^2} \cong ((B(x+1,y)-B(x,y))-(B(x,y)-B(x-1,y)))$$

$$+((B(x,y+1)-B(x,y))-(B(x,y)-B(x,y-1))) =$$

$$= B(x+1,y)+B(x-1,y)+B(x,y-1)+B(x,y-1)-4B(x,y)$$

$$=: L$$

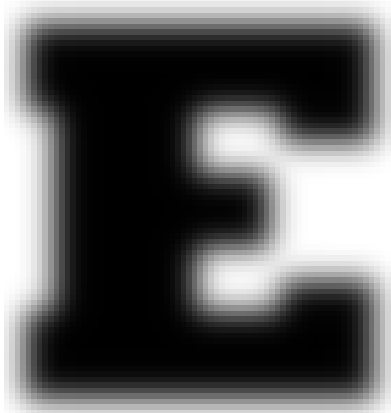
Konturenverstärkung

$$\Rightarrow \quad I - L = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} =$$

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

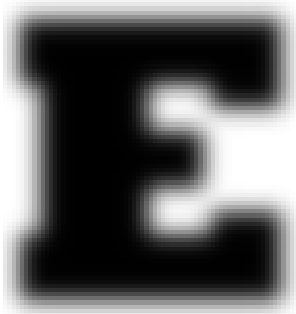
Beispiel

Mit Hilfe eines kleinen Programmes lassen sich die verschiedenen Masken auf ein Bild anwenden.
Orginalbild (mit dem Gaußfilter geglättet):



Beispiel

Wendet man die Maske $I - L$ auf das Bild an, erhält man folgendes Ergebnis:



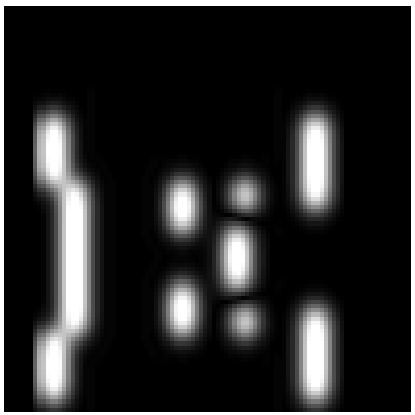
Orginal

1.*Programmdurchlauf*

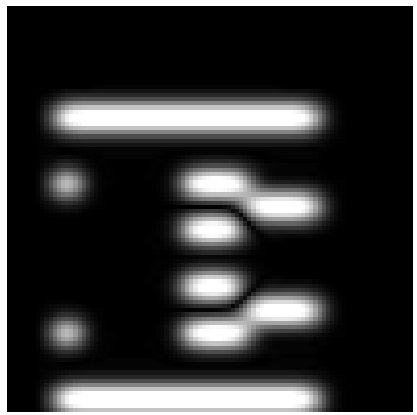
2.*Programmdurchlauf*

Beispiel

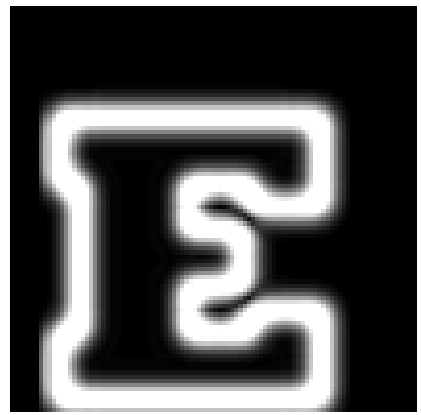
Wendet man die Masken G_x und G_y an, erhält man folgendes Ergebnis:



Maske : G_x



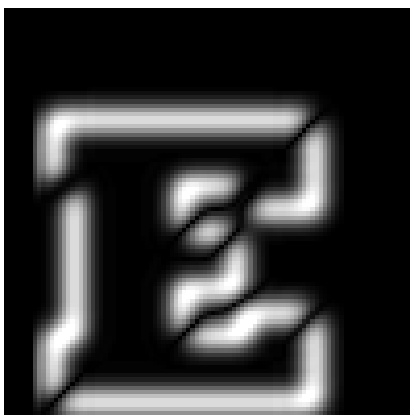
Maske : G_y



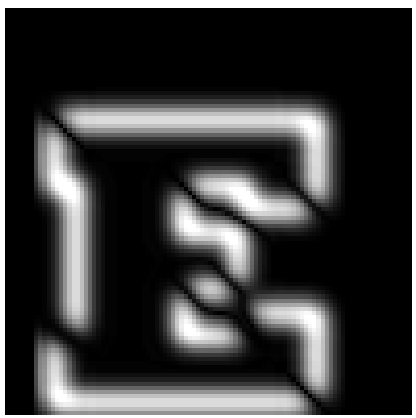
Maske : $G_x + G_y$

Beispiel

Mit den Maske D_1 und D_2 ergeben sich folgende Bilder:



Maske : D_1



Maske : D_2

Maske : $D_1 + D_2$

Literatur

- W. Burger, M. J. Burge „Digitale Bildverarbeitung", Springer 2005.
- A. Janser, W. Luther, W. Otten „Computergraphikund Bildverarbeitung", Vieweg 1996
- www.imagingbook.com
- U. Schöning „Algorithmik", Spektrum Akademischer Verlag 2001

Vielen Dank für die Aufmerksamkeit!