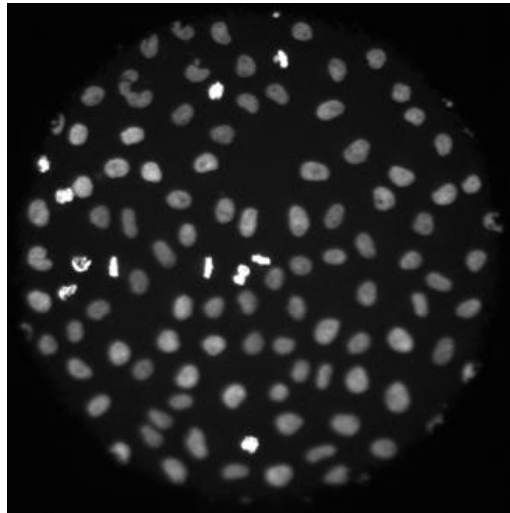


## Aufgabe 1 – Bildverarbeitung mit Graphen

20 Punkte

Gegeben ist das folgende Mikroskop-Bild aus einer Hochdurchsatz-Studie am BioQuant (siehe `cells.pgm` auf Moodle):



Das Bild zeigt Zellkerne, die unterschiedlich gefärbt wurden (grau und weiß). Die Aufgabe besteht darin, die Zellkerne beider Arten zu finden und zu zählen. Für Menschen ist dies im Prinzip kein Problem, aber moderne automatische Mikroskope erzeugen in kurzer Zeit Tausende oder sogar Millionen solcher Bilder, so dass eine manuelle Auswertung nicht mehr möglich ist.

Das Bild ist im PGM-Format abgespeichert und kann z.B. mit den Programmen `gimp` und `IrfanView` angezeigt werden. PGM-Bilder bieten sich für diese Übung an, weil sie leicht gelesen und gespeichert werden können. Dazu verwenden Sie das Modul `pgm.py` (auf Moodle), das Sie am Anfang Ihres Programms einfach importieren können:

```
from pgm import readPGM, writePGM
width, height, data = readPGM('cells.pgm')
writePGM(width, height, data, 'mein_bild.pgm')
```

Dabei geben `width` und `height` die Breite und Höhe des Bildes an, und `data` ist ein Pythonarray (Typ `list`) mit den Intensitätswerten der Pixel (von 0 = schwarz bis 255 = weiß). Auf das Pixel mit der Koordinate `(x,y)` greift man genauso zu wie auf die Matrix in Übung 4:

```
value = data[x + y*width]          # lesender Zugriff
data[x + y*width] = value          # schreibender Zugriff
index = x + y*width                # Umrechnung zwischen Koordinaten ...
x, y = index % width, index // width # ... und Pixelindex
```

Die letzten beiden Zeilen zeigen, wie man zwischen den Koordinaten eines Pixels und seinem Index im Array `data` umrechnet. Alle Funktionen sollen im File `image.py` abgegeben werden.

- Es ist bekannt, dass alle dunklen Pixel mit einem Wert unter 60 zum Hintergrund gehören, während alle anderen zum Vordergrund (also zu Zellkernen) gehören. Schreiben Sie eine Funktion `mask = createMask(width, height, data, threshold)`, die ein Array erzeugt, wo alle Hintergrundpixel den Wert 0 und alle Vordergrundpixel den Wert 255 haben. Dabei bezeichnet `threshold` die gewählte Schwelle zwischen Vorder- und Hintergrund (hier: 60). Exportieren Sie das resultierende Maskenbild mittels `writePNG(width, height, mask, 'mask.pgm')` in ein PGM-File und geben Sie dieses File ebenfalls ab. 4 Punkte
- Schreiben Sie eine Funktion `graph = createGraph(width, height, mask)`, die aus dem Maskenbild einen Graphen in Adjazenzlisten-Darstellung nach folgenden Regeln erzeugt: 6 Punkte

- Jeder Pixel stellt einen Knoten des Graphen dar, wobei die Knotennummer dem Pixelindex entspricht.
  - Die Kanten des Graphen verbinden Pixel (=Knoten) mit ihren nächsten Nachbarn, d.h. Pixel  $(x,y)$  kann mit den Pixeln  $(x-1,y)$ ,  $(x+1,y)$ ,  $(x,y-1)$ ,  $(x,y+1)$  verbunden werden. Dies bezeichnet man als "Vierer-Nachbarschaft". Beachten Sie, dass bei Randpixeln nicht alle Nachbarn existieren (z.B. hat Pixel  $(0,0)$  keinen Nachbarn  $(-1,0)$ , denn der wäre außerhalb des Bildes).
  - Eine mögliche Kante wird nur dann tatsächlich eingefügt, wenn ihre Endknoten entweder beide zum Vordergrund oder beide zum Hintergrund gehören.
- c) Bestimmen Sie mit einer Funktion `anchors, labeling = connectedComponents(graph)` die Zusammenhangskomponenten des Graphen aus b). Wenn Sie diese Funktion mittels Tiefensuche implementieren, müssen Sie die Stack-basierte Version aus Übung 9 verwenden, um den Fehler "maximum recursion depth exceeded" zu vermeiden. Alternativ können Sie den Union-Find-Algorithmus aus der Vorlesung verwenden. Neben der Ankerliste soll die Funktion ein Array `labeling` zurückgeben, das jedem Knoten die laufende Nummer der Zusammenhangskomponente zuordnet, zu der der Knoten gehört. Dabei soll die Hintergrundregion das Label 0 erhalten, der erste Zellkern das Label 1 usw. Exportieren Sie dieses Array als Bild in ein File `labeling.pgm` und geben Sie dieses File ebenfalls ab. Wie viele Zellkerne hat Ihr Algorithmus gefunden? 4 Punkte
- d) Schreiben Sie Funktionen `size = getSize(labeling)` und `intensity = getMaxIntensity(data, labeling)`, die für jede Region die Größe (=Anzahl der Pixel) und die maximale Intensität berechnen und in einem Array zurückgeben, so dass z.B. `size[regionLabel]` die Größe der Region mit der Nummer `regionLabel` angibt. Wie viel Prozent des Bildes sind Hintergrund? Wie viele Regionen haben weniger als 30 Pixel (und sind dann wahrscheinlich keine Zellkerne, sondern Detektionsfehler)? Wie viele weiße Zellkerne gibt es (maximale Intensität  $> 220$ )? Schreiben Sie eine Funktion `output = createOutput(labeling, size, intensity)`, die ein Bild mit folgenden Eigenschaften erzeugt: Alle Hintergrundpixel haben den Wert 0, alle Pixel in kleinen Regionen (Detektionsfehler) haben den Wert 255, alle Pixel in weißen Zellkernen haben den Wert 160, und alle Pixel in grauen Zellkernen haben den Wert 80. Exportieren Sie das Bild in ein File `output.pgm` und geben Sie das File ebenfalls ab. 6 Punkte