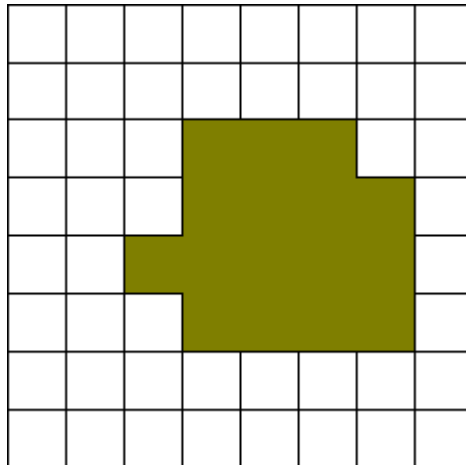


B.12 Segmentierung 2

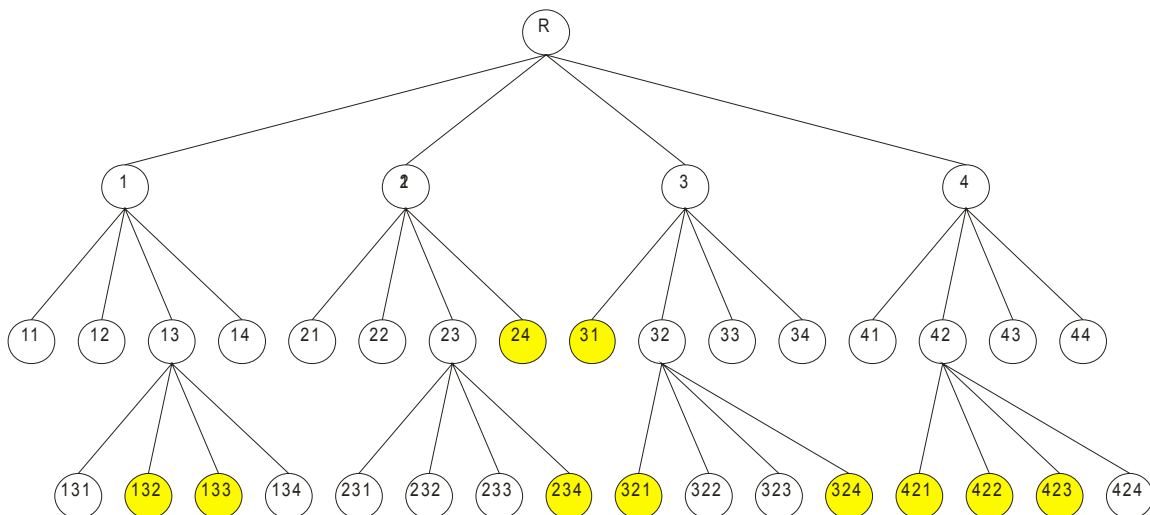
1. Split and Merge



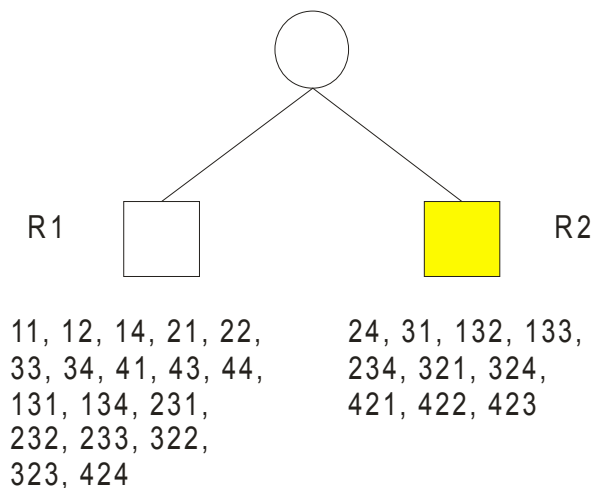
(a) Wenden Sie auf das folgende Bild den Split-and-Merge-Algorithmus an: Gemeinsame Eigenschaft: gleicher Grauwert (Das darunterliegende Gitter ist nur als Hilfestellung gedacht)!

11	12	21	22		
14	131	132	231	232	
	134	133	24	234	233
41	421	422	31	321	322
	424	423	31	324	323
44	43	34	33		

(b) Zeichnen Sie den zugehörigen Quadtree!



(c) Listen Sie die zu den zusammengehörigen Bildteilen gehörigen Teilstücke auf!



2. Split and Merge: Matlab

Auf dem Skripteserver liegt eine Funktion `splitmerge.m`, die ein rudimentäres Split & Merge bereitstellt. Das Split erfolgt wie in der Vorlesung beschrieben. Das Merge kontrolliert nur, ob die beiden zu mergenden Regionen die Eigenschaft einzeln erfüllen, aber nicht, ob die Vereinigung es nicht tut. Man muss also dafür sorgen, dass der Hintergrund die Eigenschaft nicht hat.

Schauen Sie sich die Funktion an `splitmerge.m`, stellen Sie die benötigte Funktion `predicate` zur Verfügung (welche Eigenschaft soll Ihre Region haben?) und testen Sie die Funktion an einem Bild, wo sich Region und Hintergrund gut unterscheidet.

```
function g = splitmerge(f, mindim, fun)
%SPLITMERGE Segment an image using a split-and-merge algorithm.
% G = SPLITMERGE(F, MINDIM, @PREDICATE) segments image F by using a
% split-and-merge approach based on quadtree decomposition. MINDIM
% (a positive integer power of 2) specifies the minimum dimension
% of the quadtree regions (subimages) allowed. If necessary, the
% program pads the input image with zeros to the nearest square
% size that is an integer power of 2. This guarantees that the
% algorithm used in the quadtree decomposition will be able to
% split the image down to blocks of size 1-by-1. The result is
% cropped back to the original size of the input image. In the
% output, G, each connected region is labeled with a different
% integer.
%
% Note that in the function call we use @PREDICATE for the value of
% fun. PREDICATE is a function in the MATLAB path, provided by the
% user. Its syntax is
%
%     FLAG = PREDICATE(REGION) which must return TRUE if the pixels
%     in REGION satisfy the predicate defined by the code in the
%     function; otherwise, the value of FLAG must be FALSE.
%
% The following simple example of function PREDICATE is used in
% Example 10.9 of the book. It sets FLAG to TRUE if the
% intensities of the pixels in REGION have a standard deviation
% that exceeds 10, and their mean intensity is between 0 and 125.
% Otherwise FLAG is set to false.
%
%     function flag = predicate(region)
```

```

%      sd = std2(region);
%      m = mean2(region);
%      flag = (sd > 10) & (m > 0) & (m < 125);

% Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
% Digital Image Processing Using MATLAB, Prentice-Hall, 2004
% $Revision: 1.6 $ $Date: 2003/10/26 22:36:01 $

% Pad image with zeros to guarantee that function qtdecomp will
% split regions down to size 1-by-1.
Q = 2^nextpow2(max(size(f)));
[M, N] = size(f);
meanIm = mean2(f);
f = padarray(f, [Q - M, Q - N], 'post');

%Perform splitting first.
S = qtdecomp(f, @split_test, mindim, fun , meanIm);

% Now merge by looking at each quadregion and setting all its
% elements to 1 if the block satisfies the predicate.

% Get the size of the largest block. Use full because S is sparse.
Lmax = full(max(S(:)));
% Set the output image initially to all zeros. The MARKER array is
% used later to establish connectivity.
g = zeros(size(f));
MARKER = zeros(size(f));
% Begin the merging stage.
for K = 1:Lmax
    [vals, r, c] = qtgetblk(f, S, K);
    if ~isempty(vals)
        % Check the predicate for each of the regions
        % of size K-by-K with coordinates given by vectors
        % r and c.
        for I = 1:length(r)
            xlow = r(I); ylow = c(I);
            xhigh = xlow + K - 1; yhigh = ylow + K - 1;
            region = f(xlow:xhigh, ylow:yhigh);
            flag = feval(fun, region, meanIm);
            if flag
                g(xlow:xhigh, ylow:yhigh) = 1;
                MARKER(xlow, ylow) = 1;
            end
        end
    end
end
end

% nnz(MARKER)/prod(size(MARKER))
%figure, imshow(MARKER)

% Finally, obtain each connected region and label it with a
% different integer value using function bwlabel.
g = bwlabel(imreconstruct(MARKER, g));

% Crop and exit
g = g(1:M, 1:N);

%-----%
function v = split_test(B, mindim, fun, meanIm)
% THIS FUNCTION IS PART OF FUNCTION SPLIT-MERGE. IT DETERMINES
% WHETHER QUADREGIONS ARE SPLIT. The function returns in v
% logical 1s (TRUE) for the blocks that should be split and
% logical 0s (FALSE) for those that should not.

```

```

% Quadregion B, passed by qtdecomp, is the current decomposition of
% the image into k blocks of size m-by-m.

% k is the number of regions in B at this point in the procedure.
k = size(B, 3);

% Perform the split test on each block. If the predicate function
% (fun) returns TRUE, the region is split, so we set the appropriate
% element of v to TRUE. Else, the appropriate element of v is set to
% FALSE.
v(1:k) = false;
for I = 1:k
    quadregion = B(:, :, I);
    if size(quadregion, 1) <= mindim
        v(I) = false;
        continue
    end
    flag = feval(fun, quadregion, meanIm);
    if ~flag
        v(I) = true;
    end
end
end

```

```

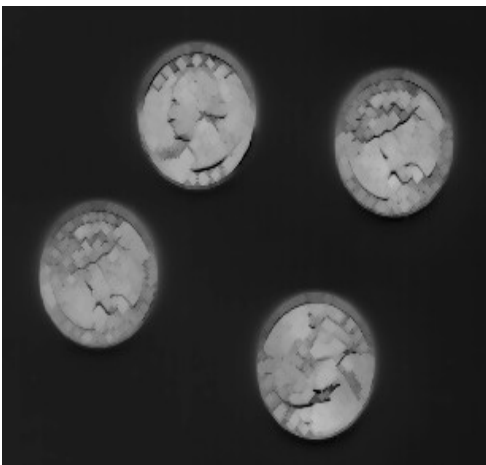
function result = predicate(region, meanIm)
%UNTITLED1 Summary of this function goes here
% Detailed explanation goes here

```

```

sd = std2(region);
result = (sd < 10);

```



3. Kantendetektion: Matlab

nützliche Funktionen: `imlter`, `fspecial`, `im2bw`, `bwmorph`

(a) Laden Sie ein Bild mit Kanten, z.B. `haus.jpg`!

```
bild = imread('haus.jpg');
```

(b) Lassen Sie den Betrag des Gradienten (z.B. aus Sobelfiltern) berechnen!

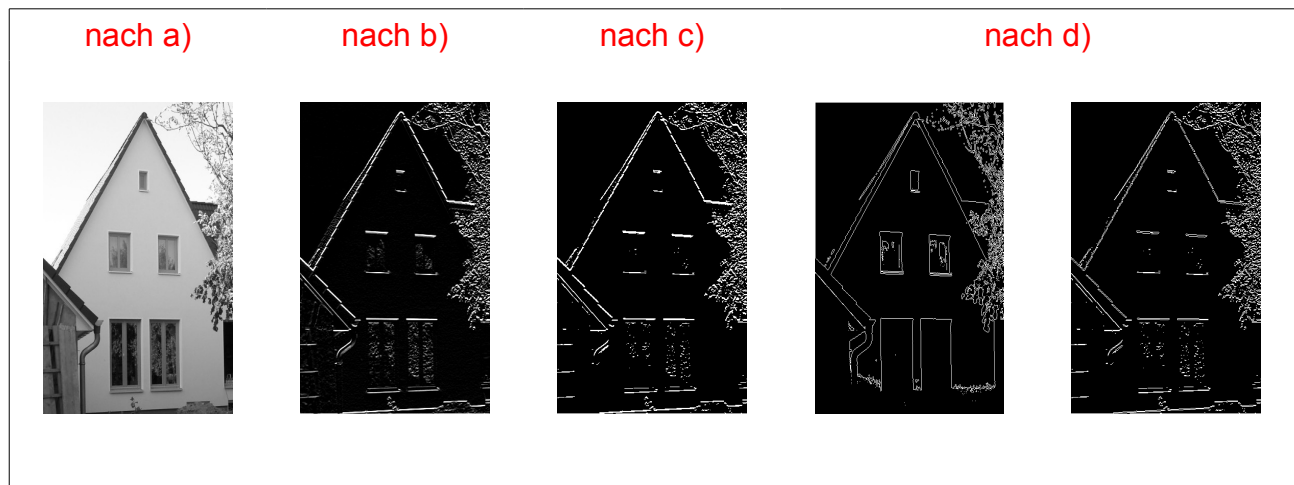
```
filter = fspecial('sobel');  
bild2 = imfilter(bild, filter);
```

(c) Binarisieren Sie das Gradientenbild mit einem geeigneten Threshold!

```
bild3 = imadjust(bild2);  
bild4 = im2bw(bild3, graythresh(bild3));
```

(d) Verdünnen Sie die Kanten zu einzelnen Pixeln!

```
bild5 = imadjust(bild);  
bild6 = im2bw(bild5, graythresh(bild5));  
bild7 = bwmorph(bild6, 'remove');  
  
bild8 = bwmorph(bild4, 'remove');  
  
figure; imshow(bild7);  
figure; imshow(bild8);
```



4. Kantendetektion: Matlab

nützliche Funktionen: edge

(a) Laden Sie ein Bild mit Kanten, z.B. haus.jpg!

```
bild = imread('haus.jpg');
```

(b) Informieren Sie sich über die Funktion edge!

(c) Lassen Sie das Kantenbild mit edge mit den default-Einstellungen berechnen für Sobel, LoG, Canny!

```
function result = aufgabe4( )
%UNTITLED1 Summary of this function goes here
% Detailed explanation goes here
bild = imread('haus.jpg');

bild1 = edge(bild,'sobel');
bild2 = edge(bild,'log');
bild3 = edge(bild,'canny');

figure('Name','Sobel'); imshow(bild1);
figure('Name','LoG'); imshow(bild2);
figure('Name','Canny'); imshow(bild3);
```

Sobel



LoG



Canny



(d) Was macht der Canny-Kantendetektor alles zusätzlich zur Filterung?

(e) Wählen Sie die Parameter der Funktion edge per Hand und versuchen Sie die interessanten Kanten anzeigen zu lassen (und die weniger interessanten nicht)!