A·Muse·Wiki Sandbox

test

test

sandbox.amusewiki.org

```
# Bestimmen wieviele Flächen eines jeden Würfels außen lieg
sub set_open_sides($)
{
 my $map=shift(@_);
 for my $x (1..$#$map-1)
 {
  for my $y (1..$#{$$map[$x]}-1)
  {
    if($$map[$x][$y]>=0)
    {
     $$map[$x][$y]-- if($$map[$x-1][$y]>=-1);
     $$map[$x][$y]-- if($$map[$x+1][$y]>=-1);
     $$map[$x][$y]-- if($$map[$x][$y-1]>=-1);
     $$map[$x][$y]-- if($$map[$x][$y+1]>=-1);
    }
  }
 }
}

# das ganze ausführen
my ($matrix,$points)=create_matrix(\@points);
mark_outside($matrix);
set_open_sides($matrix);
print_map($matrix);

# Außenflächen zusammenzählen
my $sum=0;
$sum+=$$_ for (@$points);
print "Anzahl der Außenflächen: $sum\n";
```

```
  $$p[1]-=$sub{x};
  $$p[0]-=$sub{y};
  $map[$$p[1]][$$p[0]]=4;
  $p=\$map[$$p[1]][$$p[0]];
 }
 return \@map,\@pnts;
}

# Den Äußeren Rand des Objektes bestimmen
sub mark_outside($)
{
 my $map=shift(@_);
 my $end=1;
 $$map[0][0]=-2;
 while($end)
 {
  $end=0;
  for my $x (0..$#$map)
  {
    for my $y (0..$#{$$map[$x]})
    {
     if($$map[$x][$y]==-1 and (
         ( $x-1>=0 and $$map[$x-1][$y]==-2 ) or
         ( $y-1>=0 and $$map[$x][$y-1]==-2 ) or
         ( $x+1<@{$map} and $$map[$x+1][$y]==-2 ) or
         ( $y+1<@{$$map[$x]} and $$map[$x][$y+1]==-2 )
        ))
     {
      $$map[$x][$y]=-2;
      $end=1;
     }
    }
  }
 }
}
```


Contents

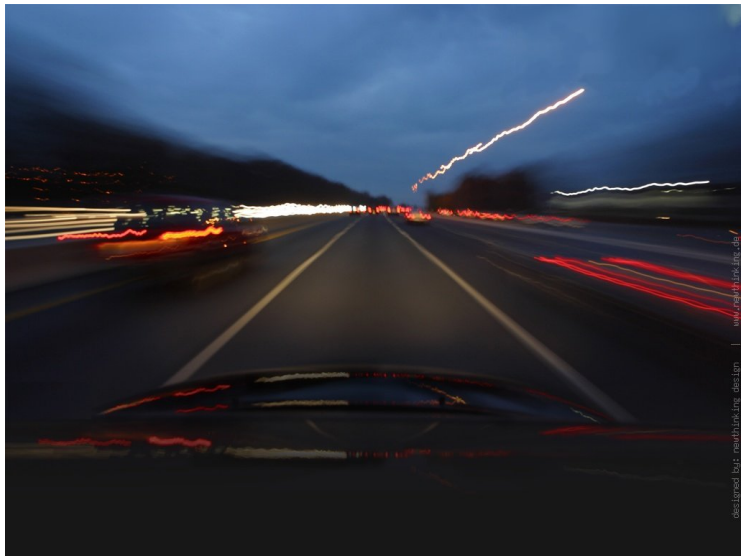
```
sub create_matrix($)
{
 my @pnts=@{$_[0]};

 my %max=(x=>0,y=>0);
 for my $p (@pnts)
 {
  $$p[0]=int($$p[0]);
  $$p[1]=int($$p[1]);
  $max{x}=$$p[1] if($$p[1]>$max{x});
  $max{y}=$$p[0] if($$p[0]>$max{y});
 }
 my %sub=(x=>$max{x},y=>$max{y});
 for my $p (@pnts)
 {
  $sub{x}=$$p[1] if($$p[1]<$sub{x});
  $sub{y}=$$p[0] if($$p[0]<$sub{y});
 }
 $sub{x}-=1;
 $sub{y}-=1;
 $max{x}+=1;
 $max{y}+=1;
 $max{x}-=$sub{x};
 $max{y}-=$sub{y};

 # die Matrix generieren
 # -1 ist eine Freie Fläche
 # -2 wird eine Freie Fläche Außerhalb des Objektes sein
 # >-1 Anzahl der Flächen die "außen" liegen
 my @map;
 for my $p (0..$max{x})
 {$map[$p]=[grep{$_=-1}(0..$max{y})];}

 for my $p (@pnts)
 {
```

```
        $eCt++ if ($x < $xmax && $field{($x + 1) . ",$y"} e
        $eCt++ if ($x > $xmin && $field{($x - 1) . ",$y"} e
        $eCt++ if ($y < $ymax && $field{"$x," . ($y + 1)} e
        $eCt++ if ($y > $ymin && $field{"$x," . ($y - 1)} e
    }

}
print "Umfang=$eCt\n";
```

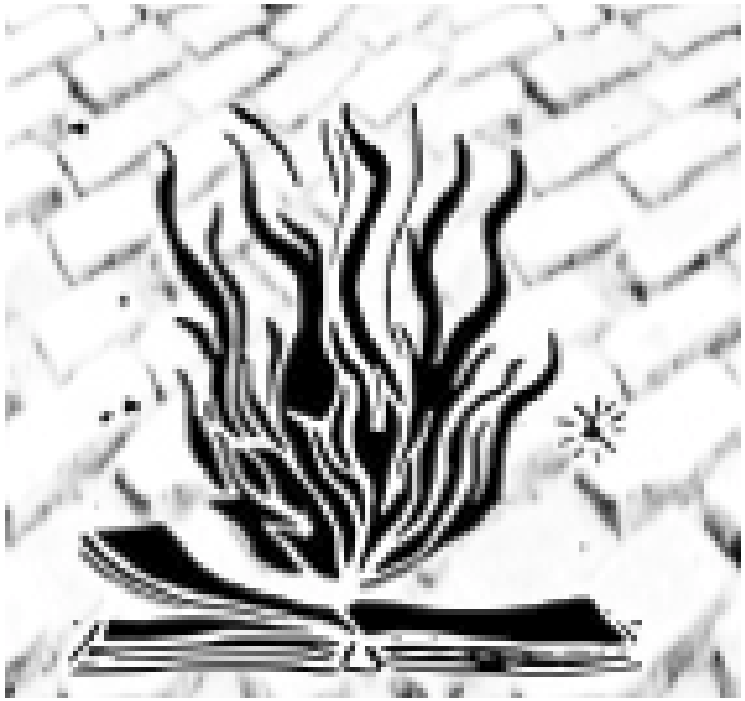
topeg

Auch hier wird wieder das Äußere markiert, diesmal aber ohne Rekusion.

```
#!/usr/bin/perl

use strict;
use warnings;

my @points=grep{$_=($_=~/,/)?[split(/,/,$_)]:[split(//,$_)]

# Die Ausgebe
sub print_map($)
{
 for my $p (@{$_[0]})
 {
  print $_>=0?'#':' ' for (@$p);
  print "\n";
 }
}

# Die benötigte Matrix generieren
# Hier erzeuge ich eine Matrix,
# in der alle übergebenen Punkte
# mormiert und eingesetzt werden.
```

asdf asdfa
x
#title Test
Dies ist ein Template für den Skripte-Unterbereich %TOPIC%.




```
        findEdge($x, $ymin, $xmin, $xmax, $ymin, $ymax);
        findEdge($x, $ymax, $xmin, $xmax, $ymin, $ymax);
    }
}

# main

# Einlesen der Koordinaten und Bestimmung eines minimalen R
my ($xmin, $xmax, $ymin, $ymax);
foreach my $pt (split(/;/, $input)) {
    my ($x, $y) = split($pt =~ /,/ ? qr{,} : qr{}, $pt);
    if (! defined $x || ! defined $y) {
        die "'$pt'";
    }
    $xmin = $x if (! defined $xmin || $x < $xmin);
    $xmax = $x if (! defined $xmax || $x > $xmax);
    $ymin = $y if (! defined $ymin || $y < $xmin);
    $ymax = $y if (! defined $ymax || $y > $ymax);
    $field{"$x,$y"} = '#';
}


printField($xmin, $xmax, $ymin, $ymax);        # Ausgabe de
markEdge($xmin, $xmax, $ymin, $ymax);        # markieren al

# Auszählen der Kanten
my $eCt = 0;
foreach my $pt (keys %field) {
    my ($x, $y) = split(/,/, $pt);
    if ($field{"$x,$y"} eq '#') {            # liegt auf de
        $eCt++ if ($x == $xmin);
        $eCt++ if ($x == $xmax);
        $eCt++ if ($y == $ymin);
        $eCt++ if ($y == $ymax);
    } elsif ($field{"$x,$y"} eq '.') {        # zählen, wen
```

```
        foreach my $x ($xmin..$xmax) {
            if (exists $field{"$x,$y"}) {
                print $field{"$x,$y"}
            } else {
                print ' ';
            }
        }
        print "\n";
    }
}

# Markieren einer leeren zusammenhängenden Fläche innerhalb
sub findEdge($$$$$$);        # Deklaration für rekursiven A
sub findEdge($$$$$$)
{
    my ($x, $y, $xmin, $xmax, $ymin, $ymax) = @_;
# print "($x, $y, $xmin, $xmax, $ymin, $ymax)\n";
    if (! exists $field{"$x,$y"}) {
        $field{"$x,$y"} = '.';
        findEdge($x + 1, $y, $xmin, $xmax, $ymin, $ymax) if
        findEdge($x - 1, $y, $xmin, $xmax, $ymin, $ymax) if
        findEdge($x, $y + 1, $xmin, $xmax, $ymin, $ymax) if
        findEdge($x, $y - 1, $xmin, $xmax, $ymin, $ymax) if
    }
}

# Beginnend am äußeren Rand alle leeren Felder markieren
sub markEdge($$$$)
{
    my ($xmin, $xmax, $ymin, $ymax) = @_;
    foreach my $y ($ymin..$ymax) {
        findEdge($xmin, $y, $xmin, $xmax, $ymin, $ymax);
        findEdge($xmax, $y, $xmin, $xmax, $ymin, $ymax);
    }
    foreach my $x ($xmin..$xmax) {
```


Aufgabe

http://board.perl-community.de/cgi-bin/ikonboard/ikonboard.cgi?act=ST;f=6;st=0;t=3849;

```
Die Aufgabe:
~~~~~~~~~~~~
          Gib zu einer Liste von Feldern (auf einem Feld mi
                Koordinaten), welche zusammenhängen, den Um
                Löcher zu ignorieren sind.

Beispiel:
~~~~~~~~~
                Die Ascii-Zeichnungen bei der Eingabe gehör
                Eingabe, sondern dienen nur zum schnelleren
                Es genügt, wenn man eine der im Beispiel ge
                Parametervarianten implementiert (also ;-se
                zwei einstelligen Zahlen, oder ;-separierte
                beliebigen ganzen Zahlen)
            Beispiel:
                Eingabe:
                11
                                    #
                Ausgabe:
                4
                Eingabe:
                -3,1;-3,2;-2,2
                                    #
                                    ##
                Ausgabe:
```

```
  8
Eingabe:
11;22
                   #
                    #
Ausgabe:
8
Eingabe:
11;12;22;10;20;30;40;31;32
                  ####
                  # #
                  ###
Ausgabe:
  14
```



```
}
```

sesth

Ähnlich wie bei docsnyder wird hier das Äußere markiert. Dazu wird für jeden Punkt auf dem Rand des umgebenden Rechtecks eine Rekursion gestartet.

```
#!/usr/bin/perl
# Autor:    Thomas Hoffmann (SESTH)
# Datum:    2007-03-01T09:15:08
# Zweck:    Gib zu einer Liste von Feldern (auf einem Feld
#            Koordinaten), welche zusammenhängen, den Umfan
#            Löcher zu ignorieren sind.

# Eingabestring
my $input = '11;12;22;10;20;30;40;31;32';
#$input = '-3,1;-3,2;-2,2';
#$input = '11;22';
#$input = '11';

# globaler Hash zum Speichern der Fläche
my %field;

# Ausgabe des 2-dim Feldes (nur für Debug-Zwecke, funtional
sub printField($$$$)
{
    my ($xmin, $xmax, $ymin, $ymax) = @_;
    print ' ' x 4, '|';
    foreach my $x ($xmin..$xmax) {
        print abs($x) % 10;
    }
    print "\n", '-' x 4, '+', '-' x ($xmax - $xmin + 6), "\
    foreach my $y ($ymin..$ymax) {
        printf "%4d|", $y;
```

```
     my ($line,$to_find) = @_;
     my @array;

     my $last = 0;
     while(my $index = index($line,$to_find,$last)){
          if($index == -1){
               last;
          }
          else{
               push @array,$index;
               $last = $index + 1;
          }
     }

     return @array;
}

sub print_usage{
     print qq~Usage: $0 "<Coordlist>"

     Die Koordinatenliste kann so aufgebaut sein:

        xy;x2y2

     oder

        x1,y1;x2,y2

     Zu beachten ist, dass x und y nur einstellig sein dürf
     Es geht jeweils von -9 bis 9...

     Beispielaufruf: perl $0 "11;12;22;10;20;30;40;31;32"

~;
     exit 0;
```


Teilnehmer

- bedivere
- betterworld
- docsnyder
- Ishka (eine normale Lösung und eine Golflösung)
- renee
- sesth
- topeg

Resultate

Allgemeine Eigenschaften

Teilnehmer	*bedivere*	*betterworld*	*docsnyder*	*Ishka*	*Ishka (Golf)*	*renee*	*sesth*	*topeg*
Koordinatenparser wie gefordert	ja	ja	ja***	ja	nein*	ja**	ja	ja
Löcher werden gefüllt	x	x				x		
kommt ohne Löcherfüllen aus			x	x	x		x	x
rekursiv		x	x		x		x	

Anmerkungen


```
                    # eine "1" oder eine "2" steht wenn ja, mus
                    # Umfang 1 abgezogen werden

                    # rechts
                    if( substr($line,$i+1,1) == 1 ||
                         substr($line,$i+1,1) == 2 ){
                         $sum--;
                    }

                    # links
                    if( substr($line,$i-1,1) == 1 ||
                         substr($line,$i-1,1) == 2 ){
                         $sum--;
                    }

                    # oben
                    if( substr($boardref->[$index-1],$i,1) == 1
                         substr($boardref->[$index-1],$i,1) == 
                         $sum--;
                    }

                    # unten
                    if( substr($boardref->[$index+1],$i,1) == 1
                         substr($boardref->[$index+1],$i,1) == 
                         $sum--;
                    }
              }
         }

     return $sum;
}

# suche alle Positionen von $to_find innerhalb des
# String; liefert Array mit allen Positionen
sub find_indexes{
```

```
     # überprüfe, ob es wirklich ein Loch ist...
     while($counter > 0){
          $counter = 0;
          for my $index( 0..scalar(@$boardref)-1 ){
               my $line = $boardref->[$index];
               next unless $line =~ tr/2// > 0 ;
               my @indexes = find_indexes($line,2);
               for my $i( @indexes ){
                    if( rindex($line,0,$i) == $i-1 ||
                         index($line,0,$i) == $i+1 ||
    substr($boardref->[$index-1],$i,1) == 0 ||
    substr($boardref->[$index+1],$i,1) == 0 ){
                         $counter++;
    substr $boardref->[$index],$i,1,0;
                    }
               }
          }
     }
}

# Zähle den Umfang
sub count_length{
     my ($boardref) = @_;

     my $sum = 0;

     for my $index( 0..scalar(@$boardref)-1 ){
          my $line = $boardref->[$index];
          $sum += ($line =~ tr/1// * 4);

          my @indexes = find_indexes($line,1);
          for my $i( @indexes ){

               # überprüfe, ob oben, unten, rechts oder li
```


Die Schreibweise mit Komma wird nicht unterstützt, folglich nur Koordinaten von 0 bis 9 möglich

Nur Koordinaten in einem begrenzten Feld möglich <nop>***</nop> Es gibt Perl-Warnungen, wenn das Komma weggelassen wird

Verhalten bei den Testfällen

Bitte beachten: Die Testfälle _"Zwei entfernte Blöcke"_ und _"Zwei verschiedene Ringe und ein einzelner Block"_ gehen über die Aufgabenstellung hinaus, weil sie nicht zusammenhängend sind. Wenn ein Script diese Tests nicht besteht, stellt Euch bitte einfach vor, dass es dafür keine Punkte abgezogen bekommt. Ich habe diese Testfälle trotzdem aufgelistet, weil es interessant ist und Aufschlüsse über die Art des Algorithmus gibt, zu sehen, ob auch nicht-zusammenhängende Felder richtig gezählt werden. Der Test "nix" ist auch etwas spitzfindig (und das Scheitern mancher Programme liegt nur an gutgemeinter Eingabevalidierung), also werden auch hier keine virtuellen Punkte abgezogen ;-)


```
     my ($field,$boardref) = @_;
     my @coords  = split /,/,$field;

     if(scalar @coords == 1){
          @coords = split //,$field;
     }

     unless(scalar @coords == 2){
          croak "incorrect input!\n";
     }

     $_ += 9 for @coords;

     substr $boardref->[$coords[1]],$coords[0],1,1;
}

# Finde die Löcher und trage eine "2" dort ein.
sub find_holes{
     my ($boardref) = @_;
     my $counter     = 0;

     # fülle alles zwischen zwei "1"en mit "2"
     for my $index( 0..scalar(@$boardref)-1 ){
          my $line = $boardref->[$index];
          next unless $line =~ tr/1// > 1 ;
          my @indexes = find_indexes($line,1);
          next if join(q{},@indexes) eq join(q{},($indexes[
          for my $i( 0..scalar(@indexes)-2 ){
                for my $j( $indexes[$i]+1 .. $indexes[$i+1]
                     substr $boardref->[$index],$j,1,2;
                     $counter++;
                }
          }
     }
```

Dabei entstehen natürlich auch ”2”en in Feldern, die nicht wirklich ein Loch sind. Um diese zu eliminieren, werden so lange ”2”en, die an ”0”en grenzen, zur ”0” konvertiert, bis keine ”2” mehr an eine ”0” grenzt.

```
#!/usr/bin/perl

use strict;
use warnings;
use Data::Dumper;
use Carp;

#my $input  = '-3,1;-3,2;-2,2';
#my $input  = '11;12;22;10;20;30;40;31;32';
#my $input  = '11;22;31;42;';
#my $input = 11;
my $input  = $ARGV[0];
print_usage() unless $input;
my @fields = split /;/,$input;

my @board  = (0 x 19) x 19;

for my $field( @fields ){
    add_field($field,\@board);
}
find_holes(\@board);
print "Umfang: ",count_length(\@board),"\n\n";

#------------------------------------------------------------
#                                              Subroutines
#------------------------------------------------------------

# trage die Koordinaten in das Spielfeld ein, und überprüfe
# (ist allerdings keine 100%ige Überprüfung)
sub add_field{
```


Teilnehmer	*hedrive*	*better-luck*	*Ishka*	*Ishka (Golf)*	*Ishka*	*renee*	*sesth*	* topeg*	*richtig*
Einzelkorn	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	4
Block Winkel	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	8
Zwei versetzte Bloecke	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	8
Fahne	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	14
Kreis mit Ecke	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	16
Großes Quadrat	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	24
Großes Quadrat mit Insel	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	korrekt	24
Oben offen	24	korrekt	korrekt	korrekt	korrekt	korrekt (W2,3)	25	korrekt	36
Unten offen	24	korrekt	korrekt	korrekt	korrekt	korrekt (W2,3)	korrekt	korrekt	36
Links offen	24	korrekt	korrekt	korrekt	korrekt	korrekt (W4)	korrekt	korrekt	36
Rechts offen	24	korrekt	korrekt	korrekt	korrekt	korrekt (W4)	26	korrekt	36
Riesiges L	korrekt	korrekt	korrekt (W1)	korrekt	K	E1, B	korrekt	korrekt	2004
Riesiges Quadrat	korrekt	korrekt	korrekt (W1)	korrekt	K	E1, B	korrekt	korrekt	2004 13
Etwas riesiges Quadrat	korrekt	korrekt (W1)	korrekt (W1)	korrekt	K	E1, B	korrekt	korrekt	524
Raute	korrekt	korrekt	korrekt	korrekt	K	korrekt	korrekt	korrekt	12
Spirale	korrekt	korrekt	korrekt	korrekt	K	W3	korrekt	korrekt	2042

Anmerkungen

W1: Warnungen: Deep recursion
W2: Warnungen: Use of uninitialized value
W3: Warnungen: substr outside of string
W4: Warnungen: Argument ... isn't numeric
E1: Bricht ab mit: substr outside of string. Man beachte, dass "substr outside of string" sowohl eine Warnung als auch eine fatale Exception sein kann, je nach dem, ob substr mit 3 oder 4 Argumenten aufgerufen wurde.
B: Renées Programm ist leider nur für Spielfelder vorgesehen, die maximal die Dimension 20x20 haben. Der Algorithmus hat also wahrscheinlich à priori kein Problem mit den gegebenen Formen, nur sind sie leider zu groß, um es auszuprobieren.
S: Es existiert spezieller Code hierfür :-)
L: Braucht zu lange und frisst mir den Swap auf, also habe ich es abgebrochen.
K: Dieses Programm unterstützt die Schreibweise mit Komma nicht und folglich nur Koordinaten von 0-9. (Für dieses Script wurden diejenigen Testfälle, die maximal 10x10 groß sind, so verschoben, dass man die Koordinaten ohne Komma schreiben konnte).

Performance

Wie weiter unten auch beschrieben, sind das riesige L und das riesige Quadrat dazu konzipiert worden, die Effizienz der Algorithmen auszureizen. Hier die Aufstellung der Laufzeiten (und der Speicherverbrauch... soon to come... oder auch nicht)

Zum Testen habe ich einen Linuxrechner mit perl 5.8.8 und Intel Pentium M 2.00 GHz und 1GB RAM verwendet. Ich habe nicht den Anspruch, hier hochqualitative Messwerte einzutragen, die Ihr in Eurer Doktorarbeit verwenden könnt, sondern vielmehr will ich nur einen Überblick geben, wie die ver-

Rechteck gearbeitet, sondern auf einer etwas größeren Form, für die sich die Abbruchbedingung kürzer aufschreiben lässt.

```
map$h{$_}=2,split/;/,pop;sub f{my($i,$j)=@_;*k=\$h{$i.$j};$
```

Eine etwas übersichtlichere Version:

```
$h{$_} = 2 for split /;/, pop;

sub f {
     my ( $i, $j ) = @_;
     my $k = \$h{ $i . $j };
     if ( 0 == $$k ) {
          if ( abs $j < 21 - abs $i ) {
         $$k = 1;
         f( $i - 1, $j );
         f( $i, $j - 1 );
         f( $i + 1, $j );
         f( $i, $j + 1 );
          }
     } elsif ( 2 == $$k ) {
          ++$u;
     }
}
f (-1, 1);
print $u || 0, "\n";
```

renee

Hier wird ein 20x20-Feld mittels einem Array von 20 Strings der Länge 20 dargestellt. Mit Ausnutzung der Macht von Perls Stringmanipulations-Funktionen werden dann die gegebenen Blöcke als das Zeichen "1" eingetragen (freie Felder sind "0") und die Löcher gefüllt. Dazu werden zunächst alle "0"en, die in einer Dimension von "1" eingegrenzt sind, zur "2" konvertiert.

```
                $aktx += 1 - 2 * int( $richtung / 2 );
            }
        } else {
            if ( $feld{ ( $aktx + 0.5 ) . ' ' . ( $akty + 0.5
                && not $feld{ ( $aktx + 0.5 ) . ' ' . ( $ak
            {
                $aktx++;
                $richtung = 0;
            } elsif ( $feld{ ( $aktx - 0.5 ) . ' ' . ( $akty
                && not $feld{ ( $aktx + 0.5 ) . ' ' . ( $ak
            {
                $akty++;
                $richtung = 1;
            } elsif ( $feld{ ( $aktx - 0.5 ) . ' ' . ( $akty
                && not $feld{ ( $aktx - 0.5 ) . ' ' . ( $ak
            {
                $aktx--;
                $richtung = 2;
            } elsif ( $feld{ ( $aktx + 0.5 ) . ' ' . ( $akty
                && not $feld{ ( $aktx - 0.5 ) . ' ' . ( $ak
            {
                $akty--;
                $richtung = 3;
            }
        }
}

print "Der Umfang der gegebenen Figur beträgt: $umfang\n";
```

Ishka (Golf)

Dies ist im Prinzip der Algorithmus, der auch von docsnyder benutzt wird. Nur wird nicht nur auf dem einschließenden

schiedenen Algorithmen mit diesem Szenario umgehen können.

Teilnehmer	*hedivere*	*betterwaldo*	*docsnyder*	*Ishka*	*renee*	*sesth*	*topeg*
Laufzeit riesiges L	2.1s	0.1s	9.2s	0.1s	s. o.	9.4s	9m40s
Laufzeit riesiges Quadrat	3.9s	s.o.	0.5s	0.1s	s. o.	0.4s	6.4s
Laufzeit etwas riesiges Quadrat	0.4s	7.1s	0.1s	0.1s	s. o.	0.1s	0.5s

Allgemeine Analyse des Problems und der Algorithmen

Wenn die Bedingung mit den Löchern nicht wäre, wäre der Algorithmus trivial: Man zählt für jeden Block 4 Kanten und zieht dann die Anzahl seiner direkten Nachbarn ab. Einige Teilnehmer haben sich dies wohl auch gedacht, und sind dann darauf gekommen, dass man diesen Algorithmus tatsächlich anwenden kann, wenn man vorher irgendwie die Löcher gefüllt bekommt.

Eine andere Möglichkeit ist, innerhalb des Rechteckes, welches die gesamte Anordnung umgibt, alle Felder zu markieren, die eine Verbindung nach außen haben. Danach kann man dann einfach die Kanten dieses Äußeren zählen, so wie im letzten Absatz beschrieben wurde, dass man die Kanten der Blöcke zählt. Bei einigen Anordnungen hat dieser Algorithmus den Nachteil, dass das umgebende Rechteck ungefähr quadratisch so viele Blöcke zählt wie die gegebenen Koordinaten. Bei anderen Anordnungen (wie dem riesigen Quadrat) hingegen hat dieser Algorithmus den Vorteil, dass er sich nicht mit dem ausgedehnten Inneren befassen muss.


```
     exit;
}

my ( $minx, $miny ) = ();
{
     for ( split /;/, $ARGV[ 0 ] ) {
          m#^(-?\d+),?(-?\d+)$#
  | die "$_ ist kein Parameter der spezifizierten Form.\n";
          $feld{"$1 $2"} = 1;
          $minx = $1 unless defined $minx && $minx <= $1;
     }

     for ( keys %feld ) {
          if ( m#$minx (.+)# ) {
                $miny = $1 unless defined $miny && $miny <=
          }
     }
}

my ( $aktx, $akty ) = map { $_ - 0.5 } ( $minx, $miny );

my $umfang    = 0;
my $richtung = 0;

while ( 0 == $umfang || $aktx + 0.5 != $minx || $akty + 0.5
     $umfang++;
     if (    $feld{ ( $aktx + 0.5 ) . ' ' . ( $akty + 0.5 )
          && $feld{ ( $aktx - 0.5 ) . ' ' . ( $akty - 0.5 )
  | $feld{ ( $aktx + 0.5 ) . ' ' . ( $akty - 0.5 ) }
          && $feld{ ( $aktx - 0.5 ) . ' ' . ( $akty + 0.5 )
     {
          $richtung = ( $richtung - 1 ) % 4;
          if ( $richtung % 2 ) {
                $akty += 1 - 2 * int( $richtung / 2 );
          } else {
```

```
  printf("Coords: '%s'\n", $testCoords);
  getCoords($testCoords);
  countEdges(0, 0);
  printf("# edges -> $numEdges\n");
}
```

Ishka

Dieser Algorithmus ist so einfach wie die Vorstellung, die jeder auf den ersten Blick von diesem Problem bekommt, aber dann nicht in Code fassen kann. Zunächst scrollt man in diesem Programm, um die entscheidende Stelle zu suchen, ist dann aber plötzlich schon am Ende.

Die gegebene Figur wird einmal umlaufen, und dabei werden die Schritte gezählt. Am Schluss hat sich die Anzahl der Schritte zum Umfang aufsummiert.

Zum Anfangen wird die obere linke Ecke genommen. Von dort wird nach rechts fortgeschritten. Wenn man irgendwo gegenstößt, wird am Hindernis entlanggelaufen. Wenn die Kante zuende ist, an der man entlangläuft, geht man an der Ecke entlang.

Interessant ist, dass dieser Algorithmus auf die Bedingung angewiesen ist, dass die Blöcke zusammenhängen müssen. Bevor ich ihn sah, konnte ich keinen Algorithmus ersinnen, der diese Bedingung braucht.

```
#!/usr/bin/perl

use strict;
use warnings;

my %feld = ();

unless ( defined $ARGV[ 0 ] && length $ARGV[ 0 ] ) {
     print "Der Umfang der gegebenen Figur beträgt: 0\n";
```


Testfälle

Wenn Dein Browser hinreichend leichtsinnig ist, Bilder von externen Seiten anzuzeigen, sollte nachfolgend zu jedem Testfall eine Skizze erscheinen.

Bei der Namensgebung und den Zeichnungen der Tests ist zu beachten: Die erste Koordinaten geht von oben nach unten, die zweite von links nach rechts.

Einzelner Block

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Einzelner%20Block.png">

11

Der Umfang ist 4.

Der simpelste Test. Er gehört zu den vier Tests, die in der Aufgabenstellung als Beispiel genannt wurden. Diese vier Tests sollte jeder Lösung richtig lösen, da jeder Autor die Möglichkeit hatte, sie zu testen.

Winkel

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Winkel.png">

-3,1;-3,2;-2,2

Der Umfang ist 8.

Kam auch in der Aufgabenstellung als Beispiel.

Zwei versetzte Bloecke

11;22

Der Umfang ist 8.

Kam auch in der Aufgabenstellung als Beispiel.

Fahne

11;12;22;10;20;30;40;31;32

Der Umfang ist 14.

Kam auch in der Aufgabenstellung als Beispiel.

Kreis mit Ecke

33;34;3,5;45;4,2;55;52;63;64

Der Umfang ist 16.

Großes Quadrat

00;01;02;03;04;05;10;15;20;25;30;35;40;45;50;51;52;53;54;55

Der Umfang ist 24.

Hierbei soll getestet werden, ob auch Löcher richtig gefüllt werden, die etwas ausgedehnter sind.


```
  $fields[pop(@xCoords)][pop(@yCoords)] = $isSet while ( @x
}

sub isSet {
  my($x, $y, $val) = @_;

  return(($x >= 0) && ($y >= 0) && defined($fields[$x][$y])
}

sub countNeighbours {
  my($x, $y) = @_;

  return(isSet($x-1, $y, $isSet) + isSet($x, $y-1, $isSet)
}

sub countEdges {
  my($x, $y) = @_;

  if ( isSet($x, $y, $isDef) ) {
     $numEdges        += countNeighbours($x, $y);
     $fields[$x][$y] = $isDone;

     countEdges($x-1, $y);
     countEdges($x+1, $y);
     countEdges($x  , $y-1);
     countEdges($x  , $y+1);
  }
}

while ( defined($testCoords=readCoords()) ) {
  @fields = undef;
  chomp($testCoords);
  ($minX, $maxX, $minY, $maxY, $numEdges) = (undef, undef,

  printf("-------------------------------------\n");
```

```
$| = 1;

sub readCoords {
  return(shift(@ARGV)) if ( @ARGV );
  printf("enter coordinate string: ");
  return(<STDIN>);
}

sub getCoords {
  my($coordStr) = @_;
  my(@pairs, @xCoords, @yCoords, $x, $y, $i);

  $coordStr =~ s/\s+//g;
  @pairs     = split(';', $coordStr);

  for ( @pairs ) {
     ($x, $y) = split(',', $_);

     $minX = $x if ( ! defined($minX) || ($x < $minX) );
     $minY = $y if ( ! defined($minY) || ($y < $minY) );
     $maxX = $x if ( ! defined($maxX) || ($x > $maxX) );
     $maxY = $y if ( ! defined($maxY) || ($y > $maxY) );

     push(@xCoords, $x);
     push(@yCoords, $y);
  }

  map { $_ -= $minX - 1 } @xCoords;
  map { $_ -= $minY - 1 } @yCoords;

  $maxX -= $minX - 2; $minX = 0;
  $maxY -= $minY - 2; $minY = 0;

  map { my $x=$_; map { $fields[$x][$_] = $isDef } ( $minY.
```


Großes Quadrat mit Insel

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Grosses%20Quadrat%20mit%20Insel.png">

00;01;02;03;04;05;10;15;20;22;25;30;35;40;45;50;51;52;53;54;55

Der Umfang ist auch 24.

Zwei entfernte Bloecke

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Zwei%20entfernte%20Bloecke.png">

11;33

Der Umfang ist 8.

Dies ist zwar ein simpler Test, geht aber insofern über die Aufgabenstellung hinaus, als die Blöcke nicht zusammenhängend sind.

Zwei verschiedene Ringe und ein einzelner Block

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Zwei%20verschiedene%20Ringe%20und%20ein%20einzelner%20Block.p

-3,2;-3,22;-9,-1;-4,-3;-2,-1;-4,3;-9,1;-6,4;-8,-2;-2,1;-5,-4;-8,2;-9,0;59;-3,-2;-4,21;-7,3;-5,4;-3,20;-2,21;-6,-4;-7,-3;-2,0

Der Umfang ist 50.

Auch nicht zusammenhängend.

Oben offen

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Oben%20offen.png">

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

Der Umfang ist 36.

Unten offen

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Unten%20offen.png">

1,2;1,6;-1,2;1,3;1,9;0,8;1,7;-1,9;-1,8;-1,3;-1,1;1,1;-1,6;-1,4;1,8;0,2;-1,7;0,1;0,9;-1,5;1,4

Der Umfang ist 36.

Links offen

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Links%20offen.png">

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

Der Umfang ist 36.

Rechts offen

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Rechts%20offen.png">

1,-1;3,-3;4,-1;2,-2;6,-1;1,-2;8,-3;2,-1;5,-3;3,-1;7,-1;2,-3;9,-2;8,-1;6,-3;7,-3;1,-3;9,-1;9,-3;4,-3;8,-2

Der Umfang ist 36.

Riesiges L

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Riesiges%20L.png">

Der Umfang ist 2004.

Dieser Perl-Code erzeugt die Koordinaten:

```
join ';', (map "0,$_", 0..500), (map "$_,0", 1..500)
```

Die Darstellung ist etwas verzerrt und nicht maßstabsgetreu. Dieses L besteht aus zwei Armen, die jeweils 500 Einheiten lang sind.


```
$absolute_bounds[1] = max(map $_->[1], values %bounds);
debug 1, "Before filling:";
paint() if $verbose >= 1;
fill_holes();
debug 1, "After filling:";
paint() if $verbose >= 1;

# Nun da die Loecher gefuellt sind, ist der Rest recht triv
print perimeter(), "\n";
```

docsnyder

Hier wird auch mit einem rekursiven Algorithmus nach zusammenhängenden Gebieten gesucht. Allerdings werden dabei nicht die Innenräume gesucht, sondern der Außenraum. Damit wird jedes freie Feld gefunden, das eine Verbindung nach außen hat. Für all diese Felder werden die direkten Nachbarn gezählt, die besetzt sind. Die Summe ist dann der gesuchte Umfang.

Angefangen wird oben links direkt außerhalb des umgebenden Rechteckes. Von dort aus werden alle Nachbarn überprüft, die weder blockiert noch schon überprüft wurden, und so weiter, rekursiv.

Da dieser Algorithmus im Gegensatz zu dem von betterworld nicht rekursiv über das Innere geht, sondern über das Äußere, zeigen sich die Performance-Einbußen der Rekursion eher nicht beim riesigen Quadrat, sondern beim riesigen L.

```
#!/usr/bin/perl

use strict;
use warnings;

my($isDef, $isSet, $isDone) = ( ' ', 'x', 'O' );
my($testCoords, @fields, $minX, $maxX, $minY, $maxY, $numEd
```

```
     my $perimeter = 0;
     my @x = keys %bounds;
     for my $x (@x) {
    for my $y ($bounds{$x}[0] .. $bounds{$x}[1]) {
         next unless BLOCKED == $felder{$x}{$y};
         $perimeter += 4;
         for (neighbours($x, $y)) {
        my $freedom = $felder{$_->[0]}{$_->[1]};
        $perimeter-- if $freedom && BLOCKED == $freedom;
         }
    }
     }
     return $perimeter;
}

for (map {split /;/} @ARGV) {
     my ($x, $y);
     ($x, $y) = m{^(-?\d+),(-?\d+)\z} or
     ($x, $y) = m{^(\d)(\d)\z}          or die $_;

     # Die angegebenen Koordinaten werden in %felder als BL

     $felder{$x}{$y} = BLOCKED;

     # Ferner werden in %bounds fuer jeden x-Wert die Begre
     # gespeichert.

     if ($bounds{$x}) {
    $bounds{$x}[0] = min($bounds{$x}[0], $y);
    $bounds{$x}[1] = max($bounds{$x}[1], $y);
     } else {
    $bounds{$x} = [$y, $y];
     }
}
$absolute_bounds[0] = min(map $_->[0], values %bounds);
```


Dieser Test testet die Effizienz der Algorithmen. Wenn der Algorithmus jedes einzelne Feld auf dem umschließenden Rechteck betrachtet, dauert das hierbei sehr lange, und es wird möglicherweise viel Speicher verbraucht.

Ursprünglich hatte ich zum Testen der Effizienz einen Test eingeplant, der nur zwei einzelne Blöcke enthält, die aber sehr weit auseinander liegen. Das würde dann aber wiederum nicht der Aufgabenstellung entsprechen, weil die Blöcke zusammenliegen müssen.

Riesiges Quadrat

Dieser Perl-Code erzeugt die Koordinaten:

```
join ';', map {
                1;
         "0,$_",
         (500-$_).',0',
         '500,'.(500-$_),
         "$_,500",
} 0.499
```

Der Umfang ist auch 2004.
Auch zum Testen der Effizienz geeignet.

Etwas riesiges Quadrat

Dieser Perl-Code erzeugt die Koordinaten:

```
join ';', map {
                1;
         "0,$_",
         (130-$_).',0',
         '130,'.(130-$_),
         "$_,130",
```

```
} 0..129
```

Der Umfang ist 524.

Auch zum Testen der Effizienz geeignet. Die Größe wurde so dimensioniert, dass die Lösung von betterworld (= meine Lösung) auf meinem Rechner keinen Swap braucht.

Raute

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Raute.png">

11;22;13;02

Der Umfang ist 12.

nix

(Ein leerer String)

Der Umfang ist 0.

Spirale

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Spirale.png">

1,1;1,2;1,3;1,4;1,5;1,6;1,7;1,8;1,9;1,10;1,11;1,12;1,13;1,14;1,15;1,16;1,17;1,18;1,19

Der Umfang ist 242.

Ein sehr kunstvoll gestaltetes Gebilde. Man beachte, dass in der Mitte ein eingeschlossenes Loch ist. Ansonsten haben alle freien Felder eine Verbindung nach außen.


```
            # $freedom == BLOCKED: Dieses Feld ist blockiert (
            #                      dann leer sein)
            #
            # $freedom == FREE:     Die Rekursion ist ans Frei
            #                       ganze (u. U. verworr
            #                       Freie wird auf FREE
            #
            # $freedom == PENDING: Wir sind in einem Loch -> D
            #                       wird auf BLOCKED ges

            $freedom = BLOCKED if PENDING == $freedom;
            $$_ = $freedom for @pending;
        }
     }
}

# Actually x is for rows and y is for columns, but doncha l
sub paint {
     my @x = sort {$a<=>$b} keys %bounds;
     my $last_x = $x[0];
     for my $x (@x) {
        print STDERR "\n" while $last_x++ < $x;
        for my $y ($absolute_bounds[0] .. $absolute_bounds[1])
             print STDERR (
            defined $felder{$x}{$y} && $felder{$x}{$y} == BLOCK
             );
        }
        print STDERR "\n";
     }
}

# Diese Subroutine zaehlt alle freien Oberflaechen, auch di
# Innenraeumen (jedoch sollten diese schon gefuellt sein, w
# aufgerufen wird).
sub perimeter {
```

```
    debug 2, ' ' x $depth, "neighbour $_->[0], $_->[1]";
    my $freedom = check_free(@$_, $pending, $depth+1);
    debug 2, ' ' x $depth, "that's recursively $freedom";

    # OK, wir haben einen freien Nachbarn gefunden.  Damit
    # frei und die ganze Rekursion kann abgebrochen werden.

    return $felder{$x}{$y} = FREE if FREE == $freedom;
     }

     # Alle Nachbarn sind BLOCKED oder PENDING -> also blei

     # Dieses Feld wird in @$pending eingetragen, damit der
     # nachgetragen werden kann, sobald er feststeht.
     push @$pending, \$felder{$x}{$y};

     debug 2, ' ' x $depth, "that's set to pending";
     return PENDING;
}

# Diese Subroutine soll alle Loecher auffuellen.
#
# Dazu wird der Freiheitsgrad von allen Feldern festgestell
# moeglicherweise in einem Loch liegen.  Das sind diejenige
# y-Richtung nicht zu beiden Seiten unendlich viel Sicht ha
#
sub fill_holes {
     my @x = keys %bounds;
     for my $x (@x) {
    for my $y ($bounds{$x}[0] .. $bounds{$x}[1]) {
         my @pending;
         my $freedom = check_free($x, $y, \@pending, 0);

         # Drei Moeglichkeiten:
         #
```


Kleinere Spirale

<img src="http://www.math.lmu.de/~pepe/rdw2007-5/images/Kleinere%20Spirale.png">

00;10;20;30;40;50;60;70;80;90;91;92;93;94;95;96;97;98;99;89;79;69;59;49;39;29;1

Der Umfang ist 112.

Lösungen

bedivere

Das ist Magie.

Wenn man sich das genau anguckt, werden nur Löcher gefüllt, die nur 1x1 groß sind. Und trotzdem wird für Testfälle wie ”Großes Quadrat” das richtige Ergebnis geliefert. Bei ”# counting the outer lines” werden sowohl Umrisse von besetzten als auch unbesetzten Feldern gezählt. Ich weiß nicht, warum das funktioniert.

```
#!/usr/bin/perl

use strict;
use warnings;

use List::Util qw(min max);

#http://board.perl-community.de/cgi-bin/ikonboard/ikonboard

############################################################
# at least we do something
my $input = "";
print qq("q" eingeben zum Beenden);
while ($input !~ /^q/) {
    if ($input) {
        if ($input =~ /^([-\d,;]+)\s*$/) {
            calcIt($1);
        } else {
```



```
}

# Diese rekursive Subroutine ueberprueft, ob ein gegebenes
# liegt.
#
# Rueckgabewert ist BLOCKED, FREE oder PENDING.  Letzteres
# Ergebnis von demjenigen Ergebnis abhaengt, welches in den
# Rekursionsebenen herauskommen wird.
#
# BLOCKED wird nur fuer die Felder zurueckgegeben, die scho
# Rekursion als BLOCKED in %felder eingetragen waren.
#
sub check_free {
     my ($x, $y, $pending, $depth) = @_;
     debug 2, ' ' x $depth, "checking freedom for $x, $y";

     if (defined $felder{$x}{$y}) {
    debug 2, ' ' x $depth, "that's $felder{$x}{$y}";
    return  $felder{$x}{$y};
     }

     if (!$bounds{$x} || $bounds{$x}[0] > $y || $bounds{$x}

    # Dieses Feld hat in y-Richtung mindestens zu einer Sei
    # viel Sicht --> FREE

    debug 2, ' ' x $depth, "that's FREE";
    return FREE;
     }

     # Wir stellen uns erstmal auf PENDING, waehrend wir re
     # Nachbarn befragen.  (Ansonsten Endlosrekursion.)
     $felder{$x}{$y} = PENDING;

     for (neighbours($x, $y)) {
```

```
# Mit -vv ausfuehren, wenn man gerade Lust hat, zu raetseln
# Debug-Zeug bedeuten soll.
#
#
use strict;
use warnings;
use List::Util qw(min max);
use Getopt::Long qw(:config bundling);

use constant FREE     => 2;
use constant BLOCKED => 1;
use constant PENDING => 0;

my $verbose = 0;
GetOptions(
     'v+' => \$verbose,
) or die;

my %felder;
my %bounds;
my @absolute_bounds;

sub debug {
     my ($level, @msg) = @_;
     print STDERR @msg, "\n" if $level <= $verbose;
}

sub neighbours {
     my ($x, $y) = @_;
     return (
    [$x-1, $y],
    [$x+1, $y],
    [$x, $y-1],
    [$x, $y+1],
     );
```



```
            print qq(falsches Eingabformat (etwas wie "-12,
            print qq("q" eingeben, zum Beenden);
        }
    }
    print "\n\nEingabe: ";
    $input = <STDIN>;
}
print "Auf Wiedersehen!\n";

################################################################
# now, lets see what we got here
sub calcIt {
    my $text = shift;
    my @x = (); my @y = ();

    # figure out the points
    my @coords = split /;/,$text;
    for (@coords) {
        if ($_ =~ /^(\d)(\d)$/) {
            push @x, $1;
            push @y, $2;
        } elsif ($_ =~ /^(-?\d+),(-?\d+)$/) {
            push @x, int $1;
            push @y, int $2
        } else {
            print qq(falsches Eingabformat von "$_"!);
            return 0
        }
    }

    # getting it all up to positive values
    my $minx = min @x;
    my $miny = min @y;
    my $maxx = max @x;
    my $maxy = max @y;
```

```
$x[$_] -= $minx for (0..$#x);
$y[$_] -= $miny for (0..$#y);

$maxx -= $minx;
$maxy -= $miny;

my $field = {};
my ($x,$y) = (0,0);
for $x(@x) {
    $field->{$x} = {} unless defined $field->{$x};
    for $y(@y){
        $field->{$x}->{$y} = 1
    }
}

# closing the holes!
for $x(0..$maxx) {
    for $y(0..$maxy){
        if (
            defined $field->{$x-1}->{$y} and
            defined $field->{$x+1}->{$y} and
            defined $field->{$x}->{$y-1} and
            defined $field->{$x}->{$y+1}
        ) {
            $field->{$x}->{$y} = 1
        }
    }
}

# counting the outer lines
my $lines = 0;
for $x(0..$maxx) {
    for $y(0..$maxy){
        $lines++ unless defined $field->{$x-1}->{$y};
```



```
            $lines++ unless defined $field->{$x+1}->{$y};
            $lines++ unless defined $field->{$x}->{$y-1};
            $lines++ unless defined $field->{$x}->{$y+1};
        }
    }
    print "Ausgabe: $lines";
    return 1
}

1;
```

betterworld

Hier werden zunächst mögliche Kandidaten für Felder in Löchern gesucht, und zwar all diejenigen, die unbesetzt sind und in der zweiten Dimension in beiden Richtungen von Blöcken umgeben sind. (Im Falle des riesigen L gibt es keine solchen Blöcke, also ist recht schnell klar, dass keine Löcher vorhanden sind). Von diesen Feldern ausgehend werden rekursiv alle unbesetzten Nachbarn untersucht. Sobald die Rekursion ins Freie läuft (das auch überprüft, indem in der zweiten Dimension nach umgebenden Blöcken gesehen wird), werden alle Felder als frei markiert, die während der Rekursion durchlaufen wurden. Wenn irgendwann keine unbesetzten Nachbarn mehr verfügbar sind, wurde das ganze Loch durchlaufen, und alle Felder werden blockiert, die während der Rekursion gefunden wurden.

Dieses Programm ist das einzige, was bei einem Testfall so viel Zeit und Speicher gebraucht hat, dass ich es abbrechen musste.

```
#!/usr/bin/perl
#
# Mit -v ausfuehren, wenn man ein bisschen was Schoenes seh
#
```