004 CodeGolfing - XMAS Tree SOLUTION

Maximilian Noppel, max@noppelmax.online

AUFGABE

Der Baum: Der Baum soll n Etagen haben. Jede Etage besteht aus $n+1$ Zeilen, wobei die oberen n aus zwei auseinanderlaufenden # besteht und in der $(n+1)$ten auf jeder Seite n Hashes stehen. Die oberste Etage beginnt mit einem Hash in der Mitte, jede Weitere wieder ein Zeichen weiter innen als die darüberliegende Zeile (also die letzte Zeile der vorherigen Etage). Das Baum ist linkbündig, also die breiteste Zeile liegt direkt links an. Aber vermutlich sind Beispiele sprechender. Siehe unten!

Der Stamm: Der Stamm ist n Zeilen hoch und zentriert unter der untersten Etage. Die Breite ergibt sich über $2(n-1)-1$.

INPUT

$n \in \{2, ..., 10\}$: Anzahl an Etagen des Baumes

I. LÖSUNG 1 (PERL, 191B)

A. Der Baum

In diesem Fall beginnen wir mit der Erkenntnis, dass der Baum gespielt ist. Also die linke Seite eigentlich gleich der rechten Seite ist. Das können wir ausnutzen. Betrachten wir also einmal nur die linke Hälfte inklusive der mittleren Spalte.

```
1111111#
111111#2
11111#22
1111###2
11111#22
1111#222
111#2222
11###222
111#2222
11#22222
1#222222
########
```

Wir erkennen, dass wir die Abfolge `1`, `#`, `2` in jeweils unterschiedlicher Anzahl haben. Die Breite einer Zeile ist aber immer gleich. Es reicht also wenn wir zwei Zahlen entgegennehmen, die Anzahl an #es und die Anzahl an `2`en. Die Anzahl von `1`sen können wir uns dann über die Breite berechnen. Die Breite ist $n*(n-1)-n+2$. Das lässt sich umstellen nach $(n-1)^2+(n-1)+2$. Da wir n auch im weiteren Verlauf nicht brauchen definieren wir $m := n-1$ und erhalten m^2+m+2 als die Breite der linken Hälte. Um den Baum dann zu spiegeln müssen wir das letzte Zeichen löschen und den String nochmal anders rum ausgeben, sowie einen Zeilenumbruch anhängen. Das fassen wir also in einer Funktion zusammen.

```
1 sub p{
2   ($h,$s) = @_;
3   $_ = "1"x($m**2+$m+2-$s-$h)
4        ."#"x$h
5        ."2"x$s;
6   print;
7   chop;
8   print~~reverse.$/
9 }
```

Die verwendete Sprache ist hier Perl. In Perl ist `$_` der Standardparameter für Funktionen. Wenn wir den String also an `$_` zuweisen können wir `print` ohne Parameter aufrufen. Das Selbe gilt für `chop`, was das letzte Zeichen weg löscht. Ein `$/` entspricht in Perl einem Zeilenumbruch.

Mit Hilfe dieser Funktion lässt sich der Baum nun ganz einfach aufbauen.

Zunächst unterteilen wir den Baum weiter in zwei Teile.

```
1  1111111#
2  111111#2
3  11111#22
4  1111###2
5  11111#22
6  1111#222
7  111#2222
8  11###222
9  111#2222
10 11#22222
11 1#222222
12 ########
```

Zum einen die Zeilen mit mehreren #es und die mit nur einem. Die mit mehreren sind also in dem gezeigten Baum die Zeilen 4, 8 und 12. Wir bauen also zwei geschachtelte Schleifen, eine für die Etagen und eine für die Zeilen. Innerhalb der inneren Schleife plotten wir alle Zeilen mit einem #. Außerdem sehen wir, dass wir immer $m*e+z$ viele `2`er ausgeben müssen. Der Code dafür sieht also so aus:

```
1 for $e (0..$m){
2   for $z (0..$m){
3     p( 1, $m *$e+$z )
4   }
5   # Print line with multiple #s.
6 }
```

Als Nächstes betrachten wir alle anderen Zeilen ohne die letzte. Also im Beispiel die Zeilen 4 und 8. Hier müssen wir n Hashes printen und dafür nur $m*e+1$ `2`er. Für die letzte Zeilen müssen wir einen Sonderfall anfangen. Wenn $e \stackrel{?}{=} m$ müssen wir also die ganze Zeile mit Hashes drucken:

```
1  for $e (0..$m){
2    for $z (0..$m){
3      p( 1, $m*$e+$z )
4    }
5    if($e == $m){
6      p( $m+2+$m*$e, 0 )
7    }else{
8      p( $m+1, $m*$e+1 )
9    }
10 }
```

Wir sehen die Formel $m*e$ nun dreimal und können $w := m+e$ definieren.

```
for $e (0..$m){
  $w = $m * $e
  for $z (0..$m){
    p( 1, $w+$z )
  }
  if($e == $m){
    p( $m+2+$w, 0 )
  }else{
    p( $m+1, $w+1 )
  }
}
```

Das `if`-Statement lässt sich auch zusammenfassen in dem wir $b := (w+1) * (e \overset{?}{\neq} m)$ definieren.

```
for $e (0..$m){
  $w = $m * $e
  for $z (0..$m){
    p( 1, $w+$z )
  }
  $b = ($w + 1) * ($e != $m);
  p( $m+2+$w-$b, $b )
}
```

In Perl kann man außerdem in einer Schleife die Standardvariable `$_` verwenden und auch Schleifen für einzelne Befehle durchlaufen. Damit können wir den Code kompakter machen:

```
for(0..$m){
  $w = $m*$_;
  p( 1, $w + $_ ) for(0..$m);
  $b = ($w + 1) * ($_ != $m);
  p( $m+2+$w-$b, $b )
}
```

Auch der Stamm wird dadurch denkbar einfach. Insbesondere kann man nur den ersten Parameter übergeben und der Zweite wird als 0 interpretiert.

```
p( $m ) for(0..$m)
```

Zeilenumbrüche und so weiter rausgekürzt kommen wir also auf 191 Bytes.

```
$m=$ARGV[0]-1;sub p{($h,$s)=@_;$_=$"x($m**2+$m+2-
    $s-$h)."#"x$h.$"x$s;print;chop;print~~reverse
    .$/}for(0..$m){$w=$m*$_;p(1,$w+$_)for(0..$m);
    $b=($w+1)*($_!=$m);p($m+2+$w-$b,$b)}p($m)for
    (0..$m)
```