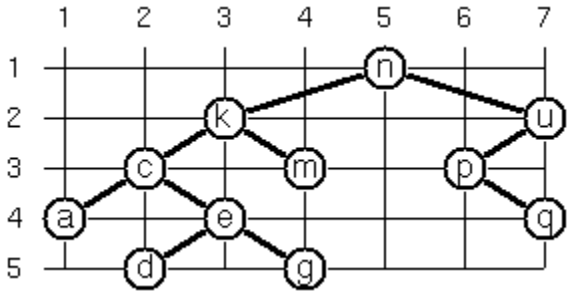
99 questions/Solutions/66

From HaskellWiki

< 99 questions | Solutions

Yet another layout strategy is shown in the illustration below:



The method yields a very compact layout while maintaining a certain symmetry in every node. Find out the rules and write the corresponding Prolog predicate. Hint: Consider the horizontal distance between a node and its successor nodes. How tight can you pack together two subtrees to construct the combined binary tree?

Use the same conventions as in problem P64 and P65 and test your predicate in an appropriate way. Note: This is a difficult problem. Don't give up too early!

```
layout :: Tree a -> Tree (a, Pos)
layout t = t'
  where (l, t', r) = layoutAux x1 1 t
        x1 = maximum l + 1

        layoutAux :: Int -> Int -> Tree a -> ([Int], Tree (a, Pos), [Int])
        layoutAux x y Empty = ([], Empty, [])
        layoutAux x y (Branch a l r) = (ll', Branch (a, (x,y)) l' r', rr')
          where (ll, l', lr) = layoutAux (x-sep) (y+1) l
                (rl, r', rr) = layoutAux (x+sep) (y+1) r
                sep = maximum (0:zipWith (+) lr rl) `div` 2 + 1
                ll' = 0 : overlay (map (+sep) ll) (map (subtract sep) rl)
                rr' = 0 : overlay (map (+sep) rr) (map (subtract sep) lr)

-- overlay xs ys = xs padded out to at least the length of ys
-- using any extra elements of ys
overlay :: [a] -> [a] -> [a]
overlay [] ys = ys
overlay xs [] = xs
overlay (x:xs) (y:ys) = x : overlay xs ys
```

The auxiliary function is passed the x- and y-coordinates for the root of the subtree and the subtree itself. It returns

- a list of distances the laid-out tree extends to the left at each level,
- the subtree annotated with positions, and
- a list of distances the laid-out tree extends to the right at each level.

These distances are usually positive, but may be 0 or negative in the case of a skewed tree. To put two subtrees side by side, we must determine the least even separation so that they do not overlap on any level. Having determined the separation, we can compute the extents of the composite tree.

The definitions of `layout` and its auxiliary function use local recursion to compute the x-coordinates. This works because nothing else depends on these coordinates.

Here is another solution. The helper function tries to place a node at (x,y), taking account of the guards at level y. Building up the tree from left to right.

```
import Data.Maybe


tree66 t =
    fromJust $ helper [] 0 0 t

helper gs x y (Branch _ t0 t1) =
    if isGuarded gs x                                   -- are we prevented from placing
      then Nothing
      else case helper (guards gs) (x - 1) (y + 1) t0 of    -- no, but can we also place t0 o
        Nothing   -> Nothing
        Just t0'  -> placeNode (t0':guards gs) x y 1 t0' t1 -- yes, now place node and t1, tr

helper gs x y Empty =
    Just Empty

-- node with subtrees symmetrically placed, as in example picture
placeNode gs x y r t0' t1 =
    case helper gs (x + r) (y + 1) t1 of
      Nothing   -> placeNode gs (x + 1) y (r + 1) t0' t1    -- could not place t1 here, tryin
      Just t1'  -> Just $ Branch (x, y) t0' t1'             -- ok

---- node with right subtree packed to the left. this also places the subtree at (0, 0)
--placeNode gs x y r t0' t1 =
--    case helper gs (x + r) (y + 1) t1 of
--      Nothing   -> placeNode gs x y (r + 1) t0' t1
--      Just t1'  -> Just $ Branch (x, y) t0' t1'


isGuarded (Branch (x', y') _ _:gs) x =
    x <= x'
isGuarded _ _ =
    False

guards gs =
    concatMap children gs
    where
      children (Branch _ t0 t1) = [t1, t0]
      children Empty = []
```

/66&oldid=40847"