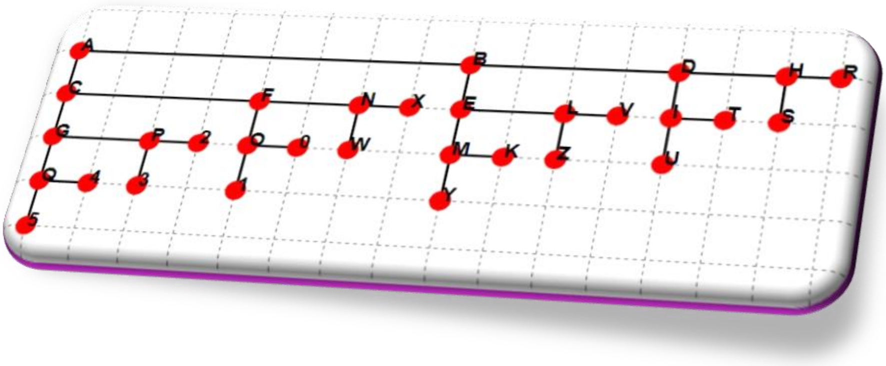
CORSO VISUALIZZAZIONE DELLE INFORMAZIONI 2021/2022

UNIVERSITÀ ROMA TRE

Tree Drawing with HV algorithms

**MARIO CUOMO**



# Sommario



*articolo del 1992 - Minimum Size h-v Drawings (P. Eades, Tao Lin, Xuemin Lin)

# Il progetto

Questo elaborato è di supporto al progetto proposto per il corso di Visualizzazione delle Informazioni dell'Università degli Studi Roma Tre.

Di seguito sono descritti 3 algoritmi per il disegno di alberi con approccio horizontal-vertical.

- Right Heavy
- Alternate
- Peter Eades

È disponibile un progetto realizzato in Javascript con la libreria D3.js al seguente indirizzo *https://github.com/mariocuomo/InfoVis/tree/main/progetto2*

Per lanciare il progetto è sufficiente utilizzare qualsiasi tool che avvii un server web in locale - per esempio python.

```
python -m http.server
```

## HV Drawing for Trees

L'Horizontal – Vertical Drawing permette di disegnare alberi – ideato per gli alberi binari ma esteso ad alberi di grado arbitrario – con un estetica e convenzione $straight\ line\ grid$ in cui i nodi sono posizioni su vertici di una griglia e gli archi sono dei segmenti orizzontali o verticali.

In particolare se $u$ è un genitore di $v$, il nodo $v$ può essere posizionato in due soli modi: allineato orizzontalmente oppure allineato verticalmente a $u$. Il disegno finale risulta essere ortogonale e, con determinati accorgimenti, planare.

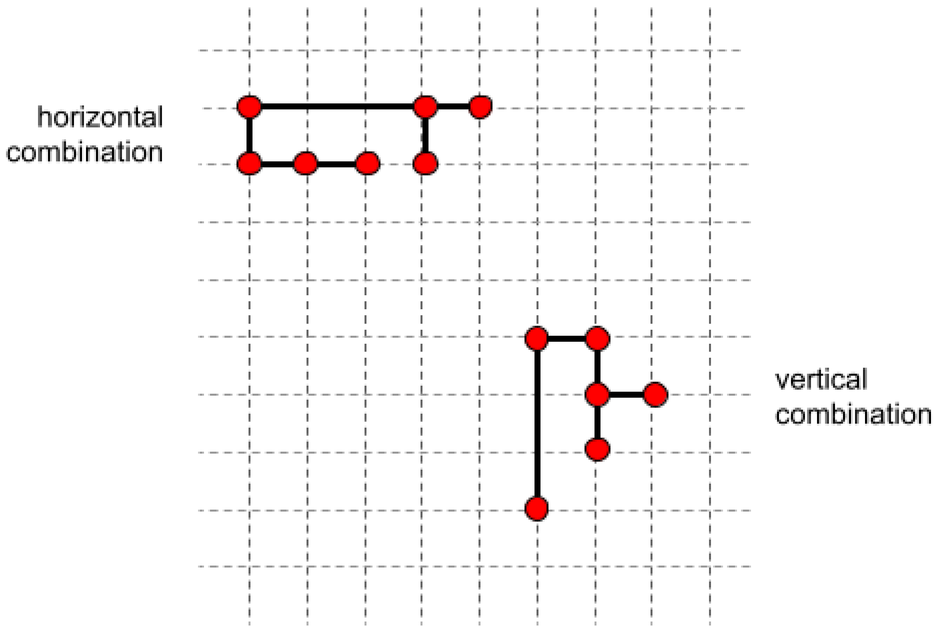
Se consideriamo un nodo $u$ con figli $v_{sx}$ e $v_{dx}$ e una griglia che si estende da sinistra verso destra e dall'alto verso il basso, si utilizzano 2 tipi di combinazioni:

- horizontal combination

$$\begin{aligned} x(v_{sx}) &= x(u) \\ y(v_{sx}) &= y(u) + 1 \\ x(v_{dx}) &= x(u) + k \\ y(v_{dx}) &= y(u) \end{aligned} \quad \text{oppure} \quad \begin{aligned} x(v_{dx}) &= x(u) \\ y(v_{dx}) &= y(u) + 1 \\ x(v_{sx}) &= x(u) + k \\ y(v_{sx}) &= y(u) \end{aligned}$$

- vertical combination

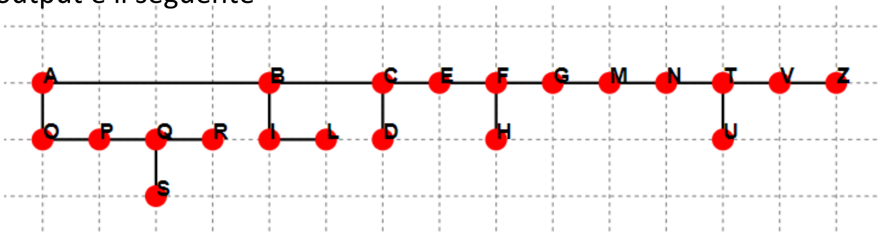
$$\begin{aligned} x(v_{sx}) &= x(u) + 1 \\ y(v_{sx}) &= y(u) \\ x(v_{dx}) &= x(u) \\ y(v_{dx}) &= y(u) + k \end{aligned} \quad \text{oppure} \quad \begin{aligned} x(v_{dx}) &= x(u) + 1 \\ y(v_{dx}) &= y(u) \\ x(v_{sx}) &= x(u) + k \\ y(v_{sx}) &= y(u) + k \end{aligned}$$



## (Binary) Right heavy

Con questo approccio si applicano solo combinazioni orizzontali in cui a destra del nodo $u$ si posiziona il sottoalbero che ha il numero di nodi maggiore. Si applica ricorsivamente a tutti i nodi dell'albero.

Un esempio di output è il seguente

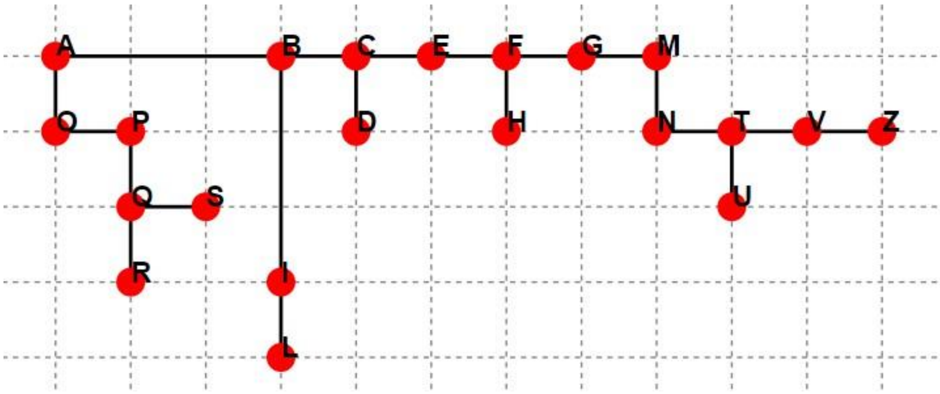



```
Tree (val, left, right, x, y)


posiziona(tree)
  //max_subtree è il figlio di tree con piu nodi
  //min_subtree è il figlio di tree con meno nodi

  min_subtree.x = tree.x
  min_subtree.y = tree.y+1
  posiziona(min_subtree)

  max_subtree.x = spostamentoADestra(tree)+1
  max_subtree.y = tree.y
  posiziona(max_subtree)



spostamentoADestra(tree)
  if tree==null
      return 0

  return max(tree.x,
             spostamentoADestra(tree.left)
             spostamentoADestra(tree.right)
```

## (Binary) Alternate – versione ingenua

Con questo approccio si applicano in modo alternato combinazioni orizzontali e verticali in base alla profondità del nodo $u$: se è pari si effettua l'orizzontale, se è dispari si effettua la verticale.

Il termine ingenuo indica il fatto che non si applica nessuna analisi sulla forma dei sottoalberi. Un esempio di output è il seguente




```
Tree (val, left, right, x, y, profondita)

posiziona(tree)
   if tree.profondita%2==0
       tree.right.x = tree.x
       tree.right.y = tree.y+1
       posiziona(tree.right)

       tree.left.x = spostamentoADestra(tree)+1
       tree.left.y = tree.y
       posiziona(tree.left)
       return

   else
       tree.left.x = tree.x+1
       tree.left.y = tree.y
       posiziona(tree.left)

       tree.right.x =  spostamentoSotto(tree)+1
       tree.right.y = tree.y
       posiziona(tree.right)
       return


function spostamentoADestra(tree)
  if tree==null
      return 0

  return max(tree.x,
             spostamentoADestra(tree.left)
             spostamentoADestra(tree.right)


function spostamentoSotto(tree)
  if tree==null
      return 0

  return max(tree.y,
             spostamentoADestra(tree.left)
             spostamentoADestra(tree.right)
```

## (Binary) Peter Eades – 'metodo degli atomi'

Peter Eades propone un metodo per disegnare un albero binario che occupa la minore $\mathrm{size}$ possibile. La $\mathrm{size}$ è una funzione $\psi(\mathrm{larghezza}, \mathrm{altezza})$ del disegno come per esempio il perimetro, l'area o il massimo tra altezza e larghezza. Di seguito si considera l'area.

Il metodo proposto da Eades permette di trovare il disegno con size ottima in $O(\mathrm{n}^2)$ – dove $\mathrm{n}$ è il numero di nodi dell'albero.

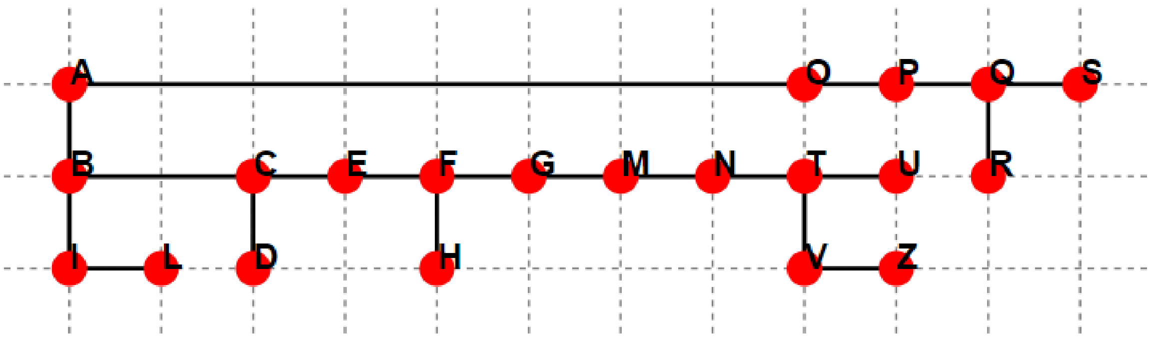
Si effettuano 3 passi principali

- Creare – per ogni nodo $u$ – l'insieme degli enclosing rectangles $P_u$.
  Gli elementi di questo insieme sono delle coppie $(\mathrm{a}, \mathrm{b})$ e ogni coppia rappresenta un rettangolo caratterizzato da larghezza e altezza. La semantica di un rettangolo $(\mathrm{a}, \mathrm{b}) \in P_u$ è che esiste un disegno dell'albero radicato a $u$ contenuto in un rettangolo di larghezza $\mathrm{a}$ e altezza $\mathrm{b}$.

- Creare – per ogni nodo $u$ – gli atomi $A_u$
  Tra tutti i rettangoli possibili in $P_u$ si è interessati a quelli che hanno size minima.
  $A_u$ è ottenuto a partire da $P_u$ ed indica l'insieme delle coppie che non ne dominano altre.
  Una coppia $(\mathrm{a}, \mathrm{b})$ è dominata da una coppia $(\mathrm{c}, \mathrm{d})$ se $\mathrm{a} \leq \mathrm{c}$ e $\mathrm{b} \leq \mathrm{d}$ – o detto in maniera semplicistica se un rettangolo di dimensioni $(\mathrm{a}, \mathrm{b})$ può essere disegnato all'interno di un rettangolo di dimensioni $(\mathrm{c}, \mathrm{d})$.

- Identificazione dei minimi & disegno
  Tra i vari atomi relativi a uno stesso nodo $u$ si seleziona quello che ha size minima e si combinano tra loro i rettangoli minimi dei figli di $u$ per occuparne lo spazio a disposizione ed evitare sovrapposizioni.

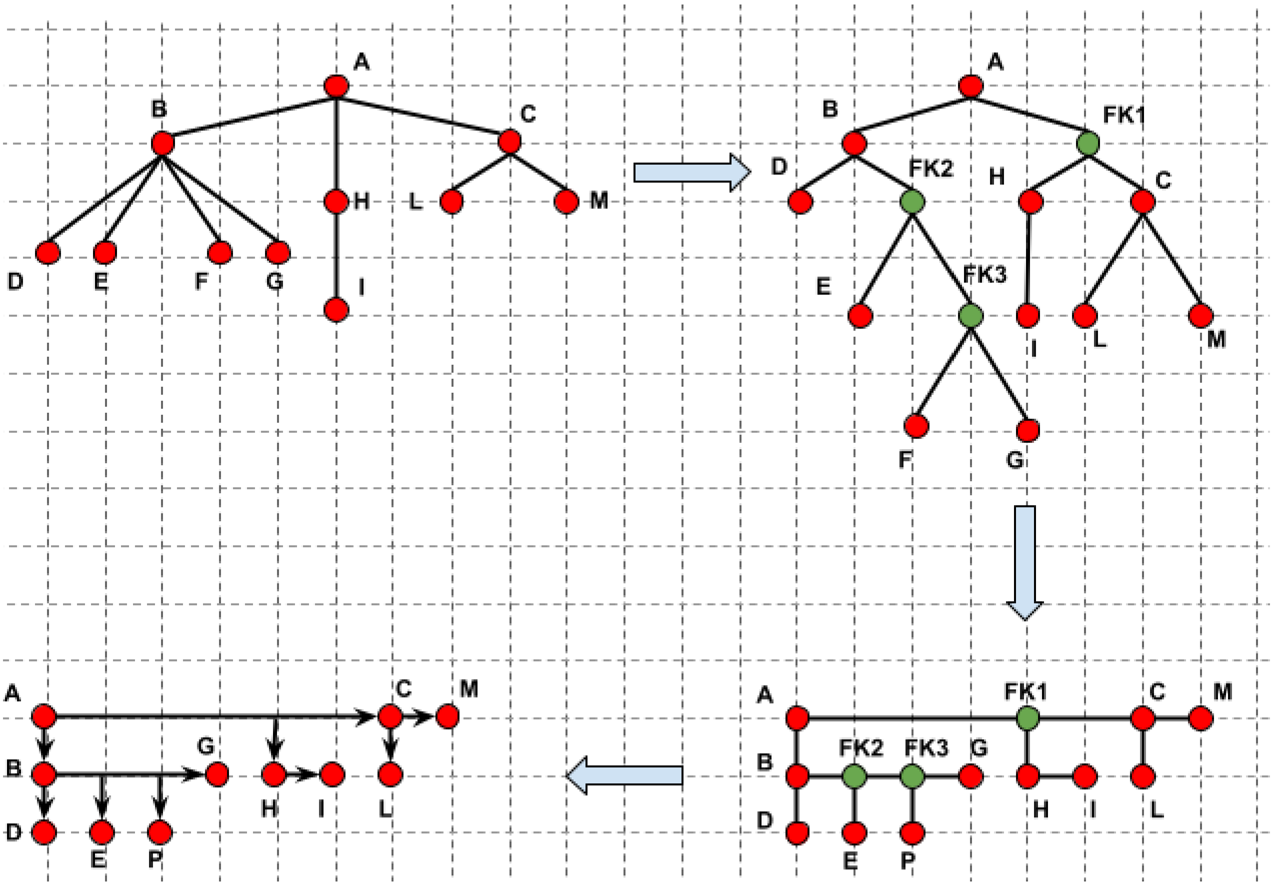
Un esempio di output – relativo allo stesso albero mostrato nel metodo alternate – è il seguente



### (Arbitrary) Right heavy

È possibile estendere gli algoritmi al disegno di alberi con grado arbitrario.
Se un nodo ha $n$ figli si posiziona a destra quello che ha più nodi e sotto – da sinistra verso destra in ordine crescente – gli altri $n-1$.

L'algoritmo si basa sulla trasformazione di un albero n-ario in un albero binario attraverso l'introduzione di fake nodes, la computazione dell'algoritmo Right heavy per il calcolo delle coordinate e la sostituzione dei nodi fittizi con gomiti.




```
fromNToBinary(tree)
       // figlio_con_max_sottoalbero  è il figlio con sottoalbero più grande

       fittizio=new NTree()

       fittizio.children = tree.children - figlio_con_max_sottoalbero

       tree.children = figlio_con_max_sottoalbero + fittizio

       for(figlio in tree.children)
              fromNToBinary(figlio)

       return


NTree(val, children[], x, y)
```