Algoritmi per la visualizzazione

Prof.ssa Tiziana Calamoneri

VISUALIZZAZIONE DI OGGETTI CON STRUTTURA AD ALBERO

Alberi

Esempio di oggetti con struttura ad albero (1)

- organigrammi di aziende
- alberi di ricerca
- parse trees di programmi
- alberi genealogici
- file systems
- modello gerarchico nei DB

Terminologia per gli alberi

- *albero*: grafo connesso aciclico
- *albero radicato*: T+*v* (naturale orientamento)
- *padre, figlio, foglia, sottoalbero*
- *albero ordinato* (figlio sx e figlio dx)
- *profondità, altezza* $n \leq 2^{h+1}-1$
- *albero completo* $n=2^{h+1}-1$

H-trees

H-trees (1)

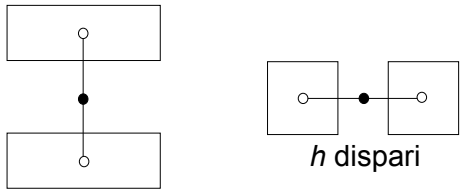
- Risultato presentato indipendentemente da Leiserson (‘80) e da Vliant (81)
- Rappresentazione rettilinea su griglia ortogonale piana di un albero binario completo con *n* nodi in area *O(n)*.
- Risultato ottimo (area *Ω(n)* banale)
- Brent e Kung (‘80) provano che se le foglie sono vincolate a giacere sul bordo allora *Ω(n log n)* area.
- N.B. Per ogni albero, si può trovare un sovra-albero completo che lo contenga quindi questo algoritmo disegna tutti gli alberi, ma se l’albero non è completo l’area non è ottima!

H-trees (2)

Costruzione induttiva:

Passo base: *h=0*

Passo induttivo:

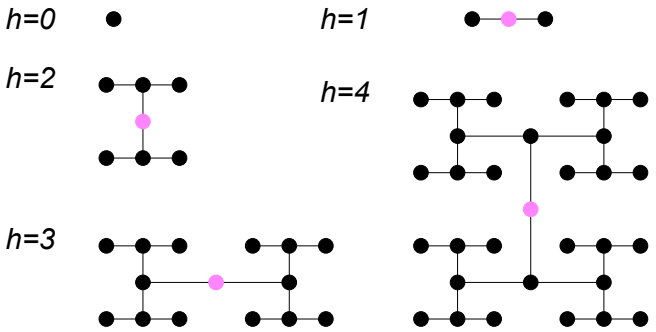


h pari

h dispari

H-trees (3)

Un esempio:



H-trees (4)

TH: L'area occupata da un H-tree con n nodi è $2(n+1)+o(n)$

DIM: $l_0=w_0=0$; $l_1=0$; $w_1=2$;
$l_2=2$; $w_2=2$; $l_3=2$; $w_3=6$; casi base
h dispari: $l_h=l_{h-1}$; $w_h=2\,w_{h-1}+2$;
h pari: $l_h=2\,l_{h-1}+2$; $w_h=w_{h-1}$; caso generico

Risolvendo le equazioni di ricorrenza…. **CVD**

Algoritmi basati sul livellamento

Algoritmi di disegno upward basati su livellamento dell'albero (1)

H-tree:
pregio: area ottima (solo per alberi completi!)
difetto: non evidenzia la struttura ad albero (i-esimo livello?)

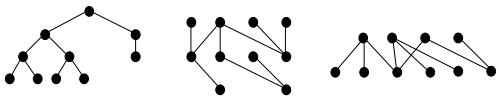
Aggiungiamo la convenzione di disegno downward a quella di disegno piano per evidenziare il livellamento.

DEF. Un *grafo livellato* G=(V,E) è tale che:
- $V=V_1\cup V_2\cup\ldots\cup V_k$, $V_i\cap V_j=\varnothing$
- per ogni arco $e=(u,v)$ esiste i t.c. $u\in V_i$ **e** $v\in V_{i+1}$.

Gli alberi sono grafi livellati ed un livellamento ammissibile è indotto dalla distanza dalla radice.

Algoritmi di disegno upward basati su livellamento dell'albero (2)

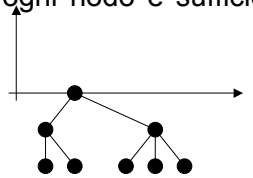
I grafi livellati sono una classe importante quindi:
molti algoritmi per disegnare grafi livellati che evidenzino il livellamento, ma…



… un algoritmo per grafi livellati non funziona come vorremmo sugli alberi, quindi:
algoritmi appositi!

Algoritmi di disegno upward basati su livellamento dell'albero (3)

In un disegno livellato ogni nodo che appartenga al livello *i* giace ad ordinata *-i*, quindi per ogni nodo è sufficiente determinare l'ascissa.

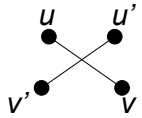


PROPRIETA' RICHIESTE:

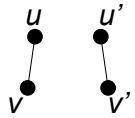
1. disegno downward: Un disegno livellato di un albero i cui livelli siano definiti come le distanze dalla radice è, di fatto, un disegno downward.

Algoritmi di disegno upward basati su livellamento dell'albero (4)

2. Planarità: se *u* è padre di *v* e *u'* è padre di *v'*, e *u* giace alla sinistra di *u'*, allora dobbiamo garantire che *v* sia alla sinistra di *v'*.



NO

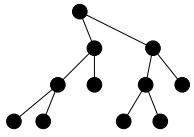


OK

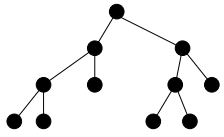
Automatico se l'albero è ordinato!

Algoritmi di disegno upward basati su livellamento dell'albero (5)

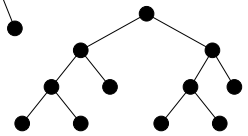
3. Criterio estetico: ascissa di ogni nodo compresa tra le ascisse dei suoi figli



NO



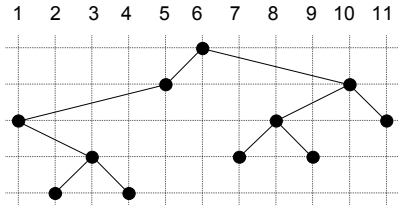
OK



MEGLIO

Algoritmi basati su livellamento (6): un semplice algoritmo

Algoritmo: assegna ad ogni nodo dell'albero radicato e ordinato un'ascissa pari al suo numero di visita *inorder.*



Otteniamo:
- un disegno livellato
- nodo tra i due figli
- evidenza delle simmetrie e degli isomorfismi.

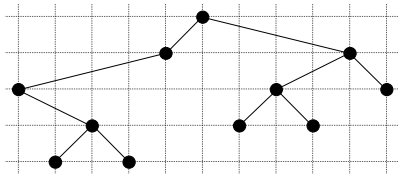
Algoritmi basati su livellamento (7): un semplice algoritmo

TH. *L'area del disegno prodotto da questo algoritmo è $O(n^2)$.*

DIM. Le ascisse sono intere e tutte distinte, quindi l'ampiezza è *n.*
L'altezza è limitata dall'altezza dell'albero.
Segue area $O(n^2)$. **CVD**

Algoritmi basati su livellamento (8): un semplice algoritmo

L'output dell'algoritmo ha 2 difetti:
- il disegno è molto più largo del necessario ⇒area troppo grande
- l'ascissa di un nodo non è necessariamente centrata rispetto a quella dei suoi figli.

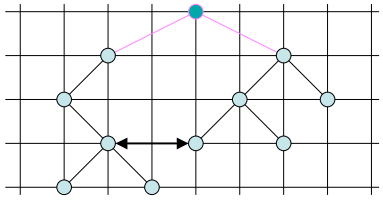


Algoritmi basati su livellamento (9): Reingold & Tilford ('81)

Algoritmo basato sulla tecnica del *divide et impera,* ideato per ovviare i problemi dell'algoritmo precedente.

Input: *T* binario radicato ordinato
Output: un disegno livellato di *T*
Passo base:
se *T* è un vertice *r*, disegnalo
Passo divide:
applica l'alg. ricorsivamente sui sottoalb. sx e dx di *r*
Passo impera:
se *r* ha 2 figli:
• muovi i disegni dei suoi sottoalberi fino a che la loro dist. sia 2
• fissa l'ascissa di *r* al centro tra i suoi figli
se *r* ha solo figlio sx (dx):
• fissa l'ascissa di *r* pari a quella del figlio +1 (-1)

Algoritmi basati su livellamento (10): Reingold & Tilford ('81)

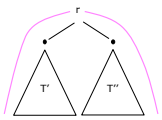


Proprietà del disegno:
- nodo al centro tra i suoi figli
- ampiezza "piccola"
- evidenza delle simmetrie e degli isomorfismi
- complessità lineare…

Algoritmi basati su livellamento (11): Reingold & Tilford ('81)

TH. *La complessità dell'algoritmo di Reingold & Tilford è lineare.*

DIM. Definiamo il contorno sinistro (destro) *cs(T)* *(cd(T))* di un albero *T(r,h)* una sequenza di vertici $v_0, v_1, \ldots, v_h$ tale che v_i è il nodo a prof. *i* più a sx (dx) dell'albero.

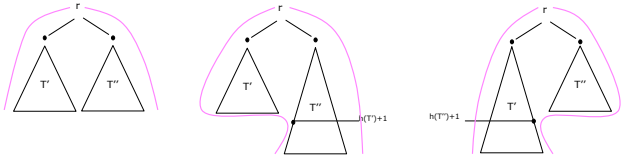


Mantenere i due sottoalberi a dist. ≥2 su ogni livello = posizionare il nodo *i*-esimo del contorno sx del sottoalb.dx a dist. ≥2 dal nodo *i*-esimo del controno dx del sottoalb. sx.

Per mantenere l'informazione *cs(T)* e *cd(T)* ad ogni passo:…

Algoritmi basati su livellamento (12): Reingold & Tilford ('81)

… memorizziamo *cs(T)* e *cd(T)* come liste:



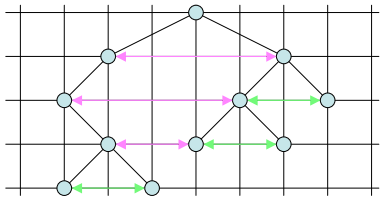
Si scandiscono i contorni "interni" per accumulare la quantità di cui bisogna distanziare i due sottoalberi. Nel contempo, si determina il nodo sul *cs(T'') (cd(T''))* ad altezza *h(T')+1 (h(T'')+1). …*

Algoritmi basati su livellamento (13): Reingold & Tilford ('81)

… quantità totale di lavoro:

$\sum_{v \in T}(O(1)+min(h_s(v),h_d(v))=O(n)+\sum_{v \in T}(min(h_s(v), h_d(v))$.

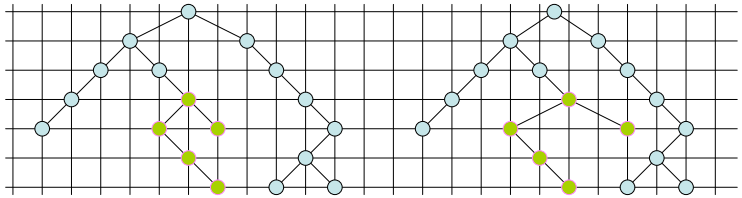
Per stimare la sommatoria:



Lavoro totale: *O(n)*.

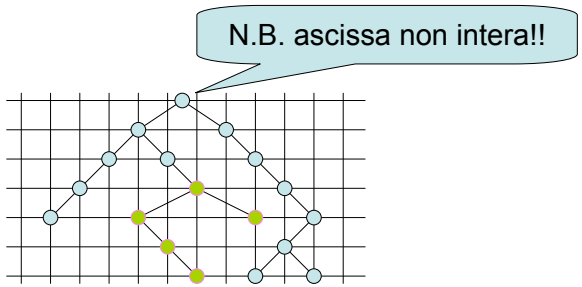
Algoritmi basati su livellamento (14): Reingold & Tilford ('81)

ATTENZIONE: Questo algoritmo non garantisce l'ottimo della larghezza!!



Una QUALUNQUE strategia *divide et impera* che non modifichi il disegno calcolato ricorsivamente NON PUO' ottenere area ottima!!

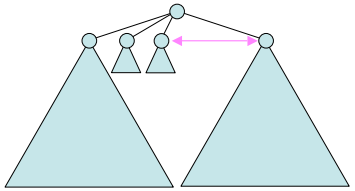
Algoritmi basati su livellamento (15): Reingold & Tilford ('81)



…il problema è facilmente risolvibile, ma trovare un disegno di area ottima diventa NP-arduo! (Supowit & Reingold '83)

Algoritmi basati su livellamento (16): Generalizzazione ad alberi *k*-ari

L'algoritmo di Reingold & Tilford si generalizza ad alberi *k*-ari, ma…



problema dello sbilanciamento!

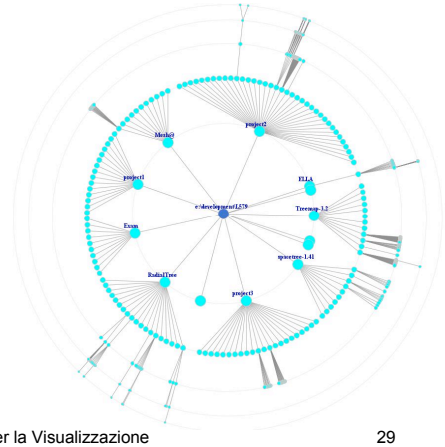
Disegni upward più efficienti

- La rappresentazione upward è la più naturale per gli alberi ma l'area occupata passa da $O(n)$ (H-tree) ad $O(n^2)$ (Reingold & Tilford).
- Si può migliorare RT ottenendo area $O(n)$ aggiungendo upward e togliendo ortogonalità?
- no: esiste una famiglia di alberi che richiede almeno area $\Omega(n \log n)$ e questa è suff. se non si richiede l'ordinamento (Crescenzi & Piperno '92).
- Se basta upward (non strettamente) e non si richiede l'ordinamento $O(n)$ area (Garg, Goodrich e Tamassia '96). facoltativo
- Se l'ordinamento va mantenuto, area $\Theta(n \log n)$.
- Se invece di polyline serve ortogonale, area $\Theta(n \log \log n)$.

Disegno radiale

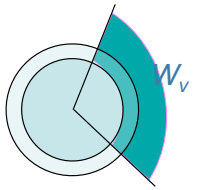
Disegno radiale (1)

Il disegno radiale è una variante del disegno livellato in cui i livelli vengono rappresentati su cerchi concentrici e la radice è al centro.



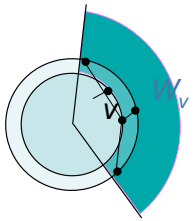
Disegno radiale (2)

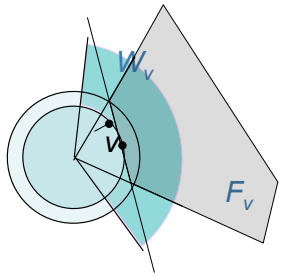
- livello *i* sulla circonferenza C_i di raggio r_i.
- per ben distribuire il disegno, ogni sottoalbero radicato in *v* è in un settore circolare W_v la cui ampiezza è proporzionale al numero di foglie del sottoalbero *l(v)*



Disegno radiale (3)

- problema: in questo modo si possono generare incroci!
- invece di W_v si usa la regione convessa F_v contenuta in W_v





Disegno radiale (4)

- detti β_v e τ_v gli angoli relativi a W_v e F_v, *l(v)* il numero di foglie dell'albero radicato in *v,* e *p(v)* il padre di *v,* si ha:

$$\beta_v = \min (l(v)\ \beta_{p(v)}/\ l(p(v)),\ \tau_v)$$

- Questa formula è ben definita, infatti, ricordando che la somma del numero di foglie nei sottoalberi radicati in *v* e nei suoi fratelli è pari al numero di foglie nel sottoalbero radicato in *p(v)*:

$$\sum_{i=1..m}(\beta_{vi}) \leq \sum_{i=1..m} (l(v_i)\ \beta_{p(v)}/l(p(v))) =$$

$$\beta_{p(v)}/l(p(v))\ \sum_{i=1..m} (l(v_i)) = l(p(v))\ \beta_{p(v)}/l(p(v)) = \beta_{p(v)}.$$

Disegno radiale (5)

Area del disegno:

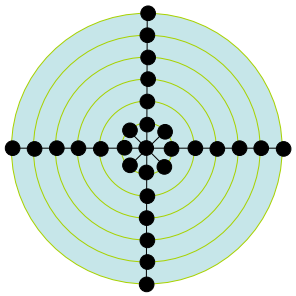
- regole di risoluzione:
  - il raggio r_i è definito in modo che $r_{i+1} - r_i = \text{cost} = r_1$
  - ogni coppia di nodi è a distanza almeno 1
- se *h* è l'altezza dell'albero e *d il max grado:*
  - perimetro del primo cerchio = $2\pi\ r_1$
  - su tale perimetro dobbiamo posizionare al più *d* nodi, quindi $r_1 = O(d)$
  - il raggio dell'ultimo cerchio è $r_h = h\ r_1 = h\ O(d)$
- Area totale=$O(h^2 d^2)$

Disegno radiale (6)

- Il disegno radiale è usato per rappresentare gli alberi liberi (senza una radice prefissata).
- Per essi si può individuare un nodo-radice tale che rispetto ad esso l'altezza sa minima.
- Algoritmo di esfoliazione
- Risultato:
  - 1 nodo radice
  - 1 coppia di nodi

Disegno radiale (7)

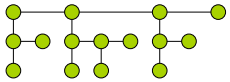
OSSERVAZIONE: il disegno radiale non è, in generale, efficiente!



Disegno HV

Un algoritmo di disegno HV (1)

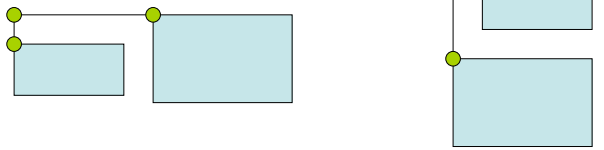
- Disegno HV: disegno piano, ortogonale, rettilineo, su griglia, downward (non strettamente) t.c.
- per ogni *u* di *T*:
  - i suoi figli sono allineati orizzontalmente o verticalmente
  - le porzioni di spazio riservate a sottoalb. dx e sx sono disgiunte



Un algoritmo di disegno HV (2)

Per ogni nodo *u* di *T*, sia $T_m(u)$ il sottoalbero con meno nodi e $T_M(u)$ il sottoalbero con più nodi.

- combinazione orizzontale: $T_m(u)$ sotto *u* a distanza 1, e $T_M(u)$ a destra di *u* a distanza "sufficiente"
- combinazione verticale : $T_m(u)$ a destra di *u* a distanza 1, e $T_M(u)$ sotto *u* a distanza "sufficiente"



Un algoritmo di disegno HV (3)

Algoritmo *divide et impera:*
Input: un albero binario *T* radicato in *r;*
Output: un disegno HV di *T;*

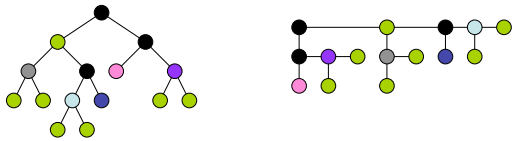
- *passo divide*: costruisci ricorsivamente un disegno HV per i sottoalberi destro e sinistro del nodo corrente *u;*
- *passo impera*: esegui una combinazione orizzontale o verticale dei sottoalberi di *u*.

- Questo algoritmo preserva l'ordinamento dei nodi se le combinazioni orizzontali o verticali al passo impera vengono scelte assecondando le dimensioni dei sottoalberi, cioè, se si sceglie una combinazione orizzontale quando il sottoalbero sinistro è minore di quello destro, e se ne sceglie una verticale altrimenti.

Un algoritmo di disegno HV (3)

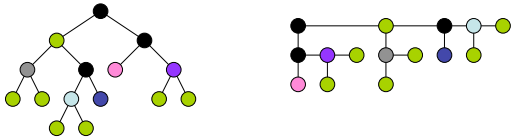
TH: l'area di un disegno HV è $O(n^2)$.
DIM: ogni riga ed ogni colonna contiene almeno un nodo. **CVD**

ATTENZIONE: l'area si può ridurre a $O(n \log n)$ applicando la variante Right-Heavy:
vengono usate solo combinazioni orizzontali.
N.B. La variante Right-Heavy, in generale, non preserva l'ordine.



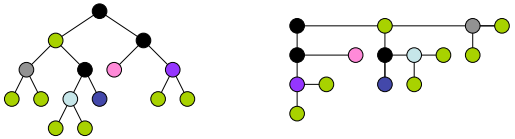
Un algoritmo di disegno HV (4)

TH: l'area di un disegno HV right-heavy è *O(n log n).*
DIM: larghezza=*O(n).*
altezza: considera il cammino verticale dalla foglia più a sx alla radice. Ogni arco ha lunghezza 1; il numero di nodi sul cammino non può essere più di *log n*. Quindi *O(log n).* **CVD**



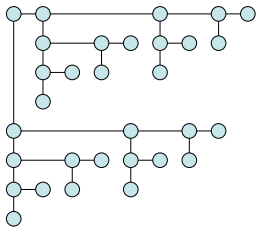
Un algoritmo di disegno HV (5)

Il disegno HV right-heavy ha pessima aspect ratio.
Per migliorarla: alternanza di combinazioni orizzontali e verticali:
- comb. orizzontali per i nodi ad altezza dispari
- comb. verticali per i nodi ad altezza pari.



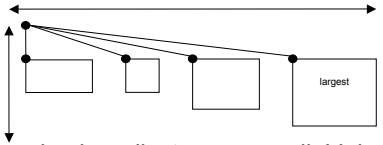
Un algoritmo di disegno HV (6)

Questa variante, nel caso di ALBERI BINARI COMPLETI, determina un disegno di area *O(n)* e aspect ratio costante.
→ Dimostrare per casa.



Un algoritmo di disegno HV (7)

L'algoritmo Right-Heavy può essere generalizzato ad alberi *k*-ari:



TH. Sia *T* un albero *k*-ario radicato con n nodi. L'algoritmo Right-Heavy costruisce un disegno Γ di *T* in *O(n)* tempo tale che:
- Γ è downward, piano, su griglia, rettilineo
- la larghezza di Γ è al più *n-1* e la sua altezza al più *log n*, quindi l'area è *O(n log n)*;
- sottoalberi semplicemente ed assialmente isomorfi hanno disegni congruenti.

Rappresentazione di gerarchie

Rappresentazione di gerarchie (1)

- Una gerarchia è un'organizzazione, basata sul raggruppamento ricorsivo, di una mole di informazioni
- Le gerarchie sono, tipicamente, espresse tramite alberi
- Esempi: struttura organizzativa di un file system, struttura di un sistema di classificazione, classificazione biologica di tutti gli animali, …

Rappresentazione di gerarchie (2)

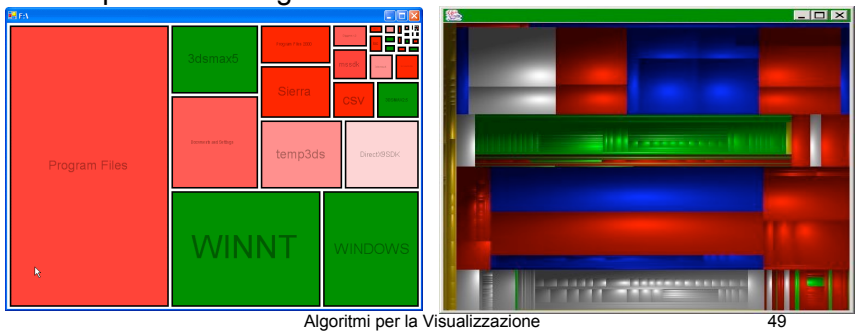
- Visualizzare gerarchie è uno degli obiettivi fondamentali nel campo della visualizzazione dell'informazione
- La struttura ad albero ha un significato più ampio di quello visto fin ora, poiché fornisce un significato addizionale cruciale nel rappresentare una struttura più complessa
- Abbastanza semplice rappresentare gerarchie medio-piccole trasmettendo con successo le informazioni che esse contengono, ma più difficile estrarre informazioni da gerarchie di grandi dimensioni.

Rappresentazione di gerarchie (3)

- Esistono molti metodi per visualizzare gerarchie.
- Il più immediato consiste nell'utilizzare uno degli algoritmi studiati per disegnare alberi in 2D, ma…
- essi – se evidenziano la struttura gerarchica - non producono disegni che sfruttano lo spazio al meglio; se sono ottimi dal punto di vista dell'area non garantiscono nemmeno un disegno strettamente downward.
- Nel caso di alberi di grandi dimensioni, perciò la loro applicabilità è limitata ad alberi relativamente piccoli.
- Altri metodi di rappresentazione…

Tree maps (1)

Le tree maps utilizzano un algoritmo di riempimento dello spazio che riempie ricorsivamente aree rettangolari con componenti della gerarchia



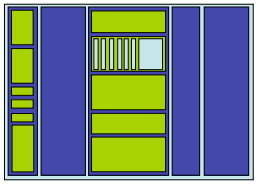
Tree maps (2)

- gli algoritmi per disegnare tree maps lavorano dividendo l'area visualizzata in una sequenza annidata di rettangoli le cui aree corrispondono ad un attributo dell'insieme dei dati (ad es. la dim. del file, se stiamo rappresentando un file system)
- le tree maps combinano gli aspetti positivi dei diagrammi di Venn con i diagrammi a torta
- le tree maps scalano bene e possono essere usate per visualizzare insieme fino ad un milione di elementi
- le tree maps possono essere usate per visualizzare gerarchie in evoluzione in modo dinamico, grazie alla semplicità degli algoritmi di visualizzazione

Tree maps (3)

Algoritmo *slice-and-dice* (Shneiderman '92)

- Usa linee parallele per dividere un rettangolo che rappresenta un certo elemento in rettangoli che rappresentano i suoi figli
- Ad ogni livello della gerarchia l'orientamento delle linee - orizzontale o verticale - viene cambiato
- Sebbene semplice da implementare, questo algoritmo produce disegni difficili da leggere, per via dei rettangoli troppo sproporzionati e quindi difficili da selezionare, confrontare in dimensione, ed etichettare.



Tree maps (4)

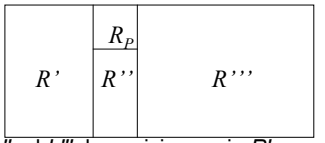
- Per migliorare l'aspect ratio: rettangoli → quadrati
- NUOVO PROBLEMA 1: i cambiamenti nell'insieme dei dati possono causare dei cambiamenti enormi nel disegno prodotto, mentre l'output dell'algoritmo slice-and-dice cambia in modo continuo rispetto alla variazione dell'input
- NUOVO PROBLEMA 2: gli algoritmi che mirano a migliorare l'aspect ratio difficilmente preservano l'ordine delle strutture, mentre l'ordine intrinseco dei dati è spesso un punto importante

Tree maps (5)

- Algoritmo per produrre una tree map ordinata, che mantenga l'ordine originario ma eviti i rettangoli molto sottili
  [Shneiderman & Wattenberg '01].
- Basato su un semplice processo ricorsivo, in parte ispirato dall'idea di trovare l'analogo bidimensionale dell'algoritmo di Quick-Sort
  1. viene scelto un elemento speciale, il *pivot*, che va posizionato a contatto del bordo del rettangolo che contiene i rettangoli considerati;
  2. gli elementi rimasti vengono assegnati a 3 grandi rettangoli che permettono di ricoprire il resto dell'area;
  3. l'algoritmo viene applicato ricorsivamente a questi 3 rettangoli.

Tree maps (6)

Input: Rettangolo R da suddividere; elementi con area $L_1, L_2, ..., L_n$;
Output: Rettangoli $R_1, R_2, ..., R_n$ corrispondenti agli elementi.



1. Se $n<=4$, posiziona i rettangoli secondo fig.
2. Sia il pivot P l'elemento con area maggiore
3. Dividi R nelle 4 aree mostrate in fig.
4. Posiziona P nel rettangolo R_P
5. Dividi gli elementi diversi da P in 3 liste, L', L'' ed L''' da posizionare in R', R'' ed R'''.
   L' =elementi con indice minore di P nell'ordinamento;
   Dividi gli elementi rimanenti in L'' ed L''' t.c. gli elementi in L'' abbiano indice minore degli elementi di L''' e l'aspect ratio di R_P sia più vicino possibile ad 1.
6. Ricorsivamente inserisci L', L'' ed L''' in R', R'' ed R'''.

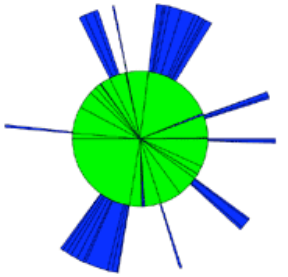
Tree maps (7)

- Possibili piccole variazioni dell'algoritmo, che consistono sostanzialmente nel modificare la scelta del pivot
- L'algoritmo descritto sceglie il pivot come l'elemento di area maggiore con l'idea che un elemento più è grande e più è difficile da inserire
- Approcci alternativi:
  - selezionare l'elemento centrale della lista ordinata in modo da creare in qualche modo un disegno bilanciato,
  - selezionare P in modo da creare le liste L' ed L''' approssimativamente della stessa area totale, calcolando per ciascun elemento quale sarebbe l'area di L' ed L''' se esso fosse il pivot; anche qui lo scopo è quello di generare dei disegni bilanciati.

Sun burst

- il sun burst è simile alla tree map, ma è circolare:
  - la radice della gerarchia è al centro della visualizzazione,
  - i livelli successivi sono irraggiati nei confini definiti dall'angolo assegnato al padre

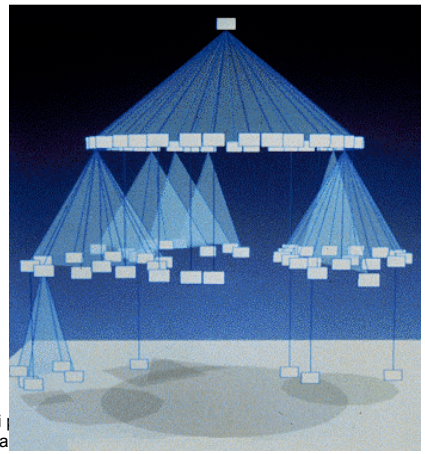
entrambe le strutture sono volte al riempimento dello spazio: la tree map riempie completamente un'area rettangolare mentre il sun burst parzialmente un'area circolare



Cone trees (1)

I *cone trees* [Robertson, Mackinlay, Card] sono delle strutture tridimensionali interattive, realizzate tramite ombreggiature, trasparenze e animazioni

From Computer Desktop Encyclopedia
Reproduced with permission.
© 1996 Xerox Palo Alto Research Center



Cone trees (2)

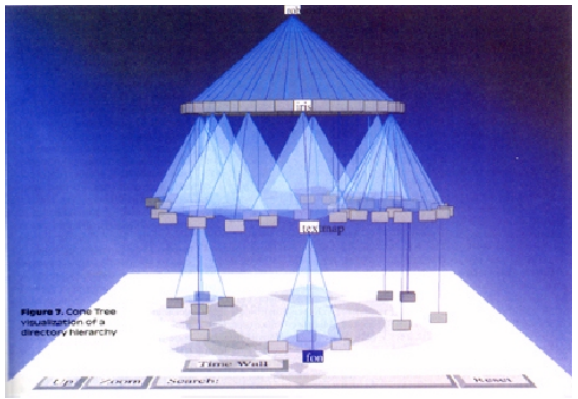
Disegno 3D di alberi etichettati:

- i nodi sono rettangoli di dimensione fissa in modo da poter contenere delle etichette;
- la radice della gerarchia è posta in corrispondenza del centro della faccia superiore del parallelepipedo che contiene il disegno, ed è l'apice di un cono lungo alla cui base sono uniformemente posti i figli
- il secondo livello di nodi viene rappresentato sotto il primo, con i figli sul bordo della base di altri coni
- l'aspect ratio dell'albero è fissata in modo da far sempre entrare l'intero albero nello spazio a disposizione
- I diametri di base dei coni su ciascun livello sono ridotti in una progressione che assicuri che l'ultimo livello entri nella larghezza dello spazio a disposizione …..

Cone trees (3)

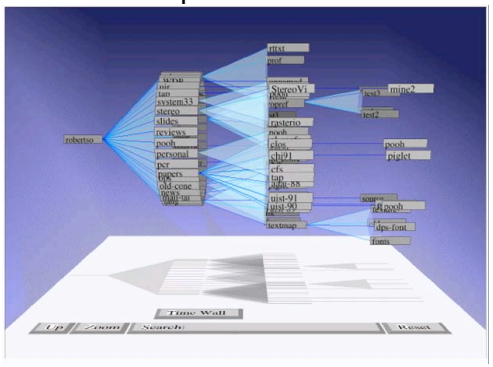
- I coni sono ombreggiati e trasparenti in modo da non percepirli come blocchi che coprono la visuale di quello che c'è dietro di loro
- Quando viene selezionato un nodo, l'intero cone tree ruota in modo che il nodo selezionato ed ogni nodo sul cammino da quel nodo alla radice si trovino sul davanti del disegno
- Le rotazioni di tutti i coni sono condotte in parallelo, scegliendo l'angolo di rotazione minore, e sono animate, in modo che l'utente possa seguire la trasformazione ad una velocità tale che il suo sistema di percezione possa tracciare
- Poiché spesso il rettangolo per l'etichetta non è di dimensione sufficiente, essa viene mostrata solo sul cammino corrispondente al nodo selezionato. …..

Cone trees (4)



Cone trees (4)

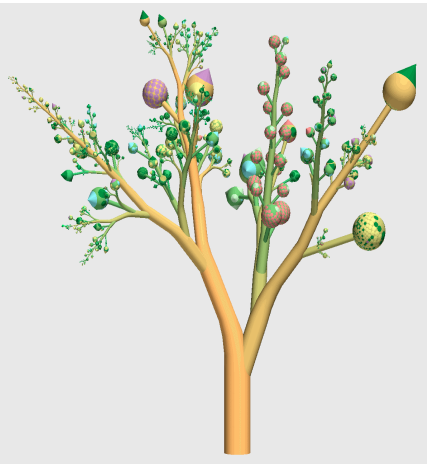
Alternativamente, è possibile posizionare l'albero in orizzontale in modo che le etichette risultino più lunghe; in tal caso la struttura prende il nome di *cam tree*.



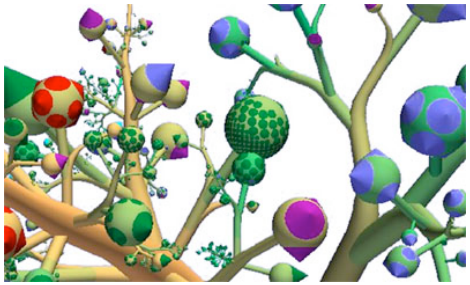
Alberi botanici (1)

Gli alberi botanici [Kleiberg, van de Wetering, van Wijk] ha l'estetica di un albero naturale, pensato per i file systems, in cui sidistinguono:

- grandi files (frutti a cono, il cui colore dà indicazioni sul tipo),
- directories con grandi e/o tanti files (frutti a puntini),
- directories con grande contenuto (rami grossi).



Alberi botanici (2)



Gli alberi botanici hanno un'estetica interessante, ma non è ancora chiaro quali siano le applicazioni reali in cui essi siano veramente utili