Insiemi disgiunti

1

Classificazione

"gli animali si dividono in:

a) appartenenti all'imperatore, *b*) imbalsamati,

c) addomesticati, *d*) lattonzoli, *e*) sirene, *f*) favolosi,

g) cani randagi, *h*) inclusi in questa classificazione,

i) che s'agitano come pazzi, *j*) innumerevoli,

k) disegnati con un pennello finissimo di pelo di cammello,

l) eccetera, *m*) che hanno rotto il vaso,

n) che da lontano sembrano mosche".

Emporio Celeste di Conoscimenti Benevoli

(J.L. Borges)

2

Classificazione

"Classification is the cognitive process in which ideas and objects are recognised, differentiated and understood.

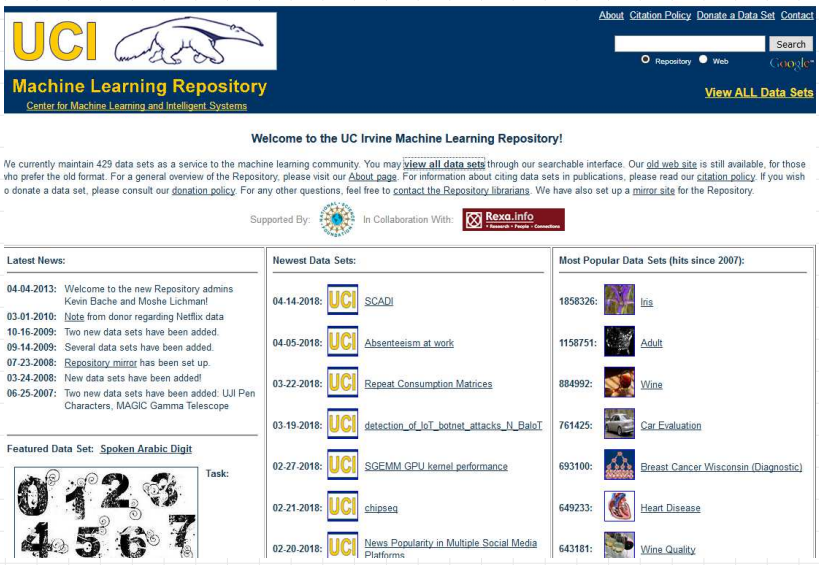
Classification implies that objects are grouped into categories, usually for some specific purpose.

Categorization is fundamental in decision making and in all kinds of interaction with the environment. "

Wikipedia

3

Classificazione

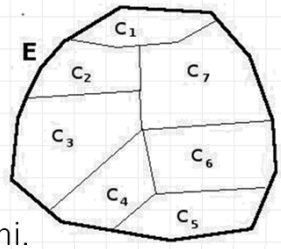


4

La ADS per insiemi disgiunti

Utilizzata principalmente per rappresentare:

- relazioni di equivalenza
  - riflessive,
  - simmetriche,
  - transitive
- inducono partizionamenti di insiemi.



Algoritmi molto semplici, analisi di complessità molto difficile.

5

Relazioni di equivalenza

Esempi?

- Insieme degli abitanti dell'Italia e relazione "*abita nello stesso comune di*"?
- Numeri naturali e relazione "*è maggiore di*"?
- Una famiglia e relazione "*è fratello di*"?
- Una famiglia e relazione "*è padre di*"?
- Una rete di computer e relazione "*è connesso con*"?
- . . .

6

Classi di equivalenza e partizioni

Relazione di equivalenza S definita sull'insieme S={a_1,a_2, … , a_n}.

Le classi di equivalenza sono sottinsiemi disgiunti di S.

Possibile identificare in $\Theta(1)$ se due elementi a_i e a_j sono nella stessa classe, utilizzando una matrice esplicita di dimensioni n^2.

- Relazione implicita, usando meno memoria?
- Algoritmi on-line?

Strutture dati per insiemi disgiunti

E' dato un insieme S scomposto in insiemi disgiunti S_1, …, S_k.

Ogni insieme è identificato da un suo membro **rappresentante**.

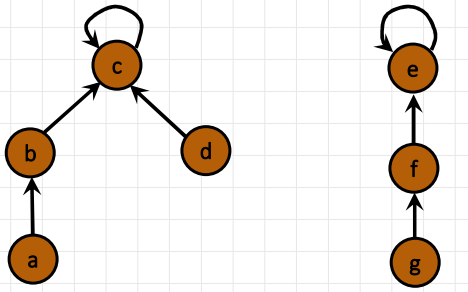
Si vogliono realizzare le seguenti operazioni:

- **Make-Set(x)**: inizializza un nuovo insieme contenente il solo elemento *x*
- **Find-Set(x)**: trova l'insieme a cui appartiene l'elemento *x*
- **Union(x,y)**: unisce gli elementi degli insiemi che contengono x e y, S e T rispettivamente, nell'unico insieme $S \cup T$

Strutture dati per insiemi disgiunti

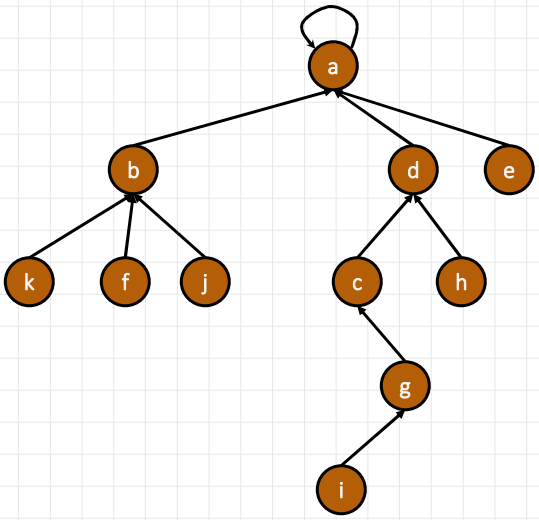
Gli insiemi possono essere rappresentati da alberi radicati (**up-tree**), in cui ogni nodo contiene un elemento e ogni albero rappresenta un insieme.

- Ogni elemento ha un *puntatore solo al padre*.
- La radice contiene il rappresentante, che è padre di se stesso.



9

Up-Tree



10

Up-tree, make-set e find-set

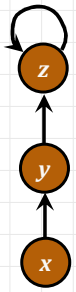
Make-set(x)

Inizializza un nuovo up-tree contenente il solo nodo x. Θ(1).

Find-set(x)

Percorre la catena dei puntatori fino a trovare il rappresentante di x. O(h).

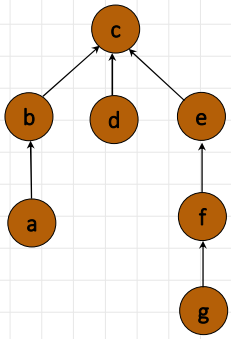
```
Find-Set(x)
while x ≠ p[x]
   x = p[x]
return x
```

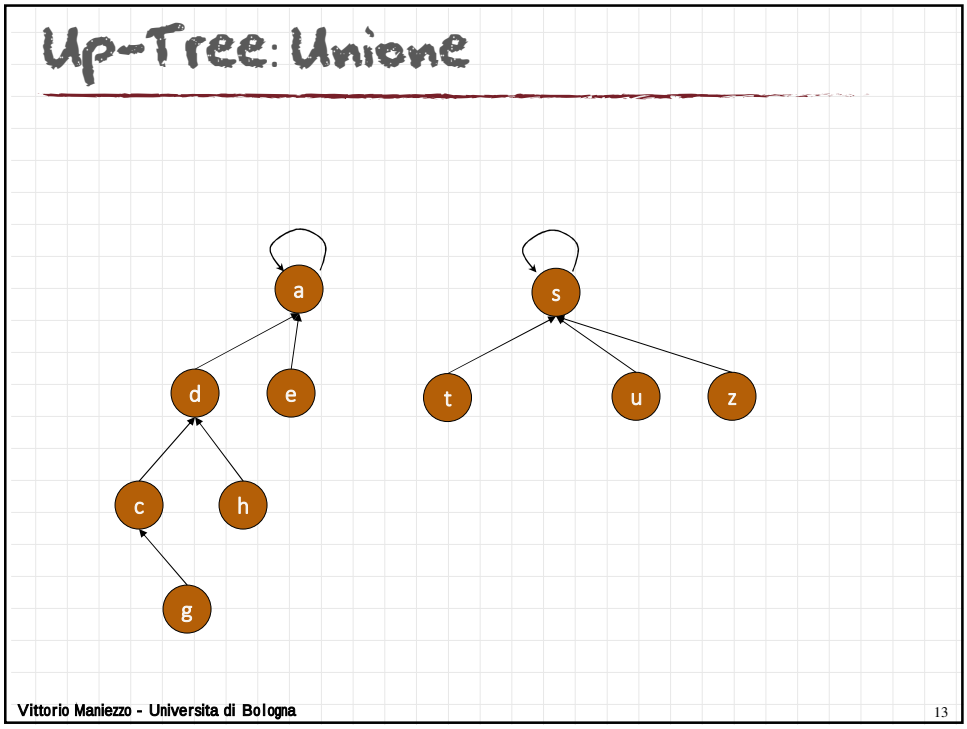


Up-Tree: unione

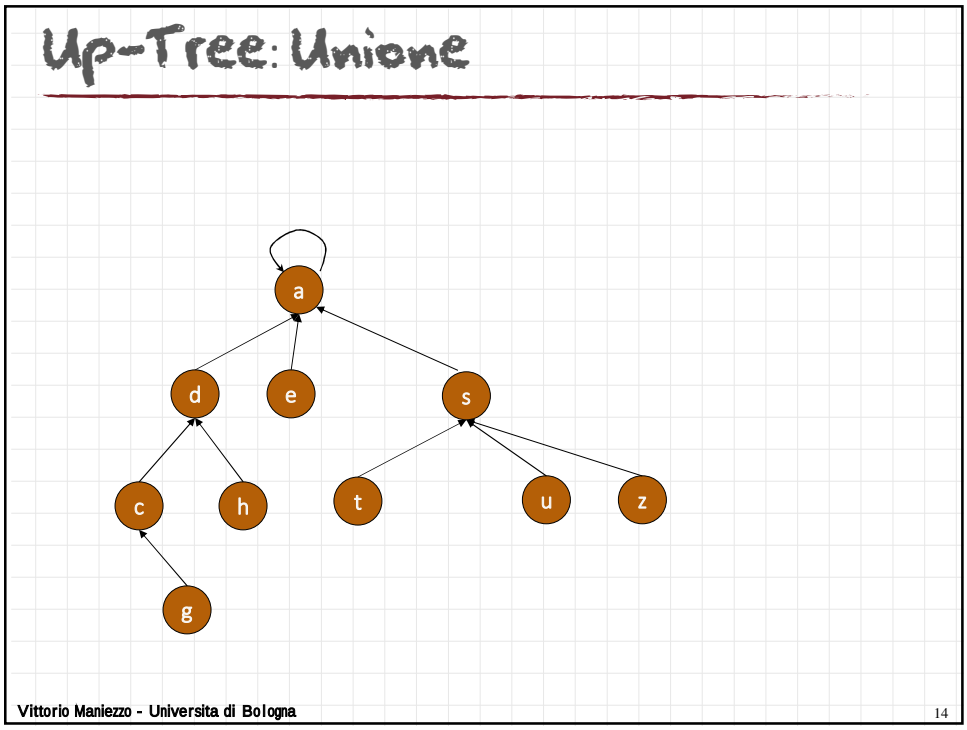
L'unione è semplicissima: si fa sì che la radice dell'albero che ha meno nodi punti (come padre) a alla radice dell'albero con più nodi.

Il numero dei nodi viene approssimato dal rango (limite superiore all'altezza) associato ad ogni nodo.




Up-Tree: Unione
a
s
d
e
t
u
z
c
h
g
Vittorio Maniezzo - Universita di Bologna
13

13

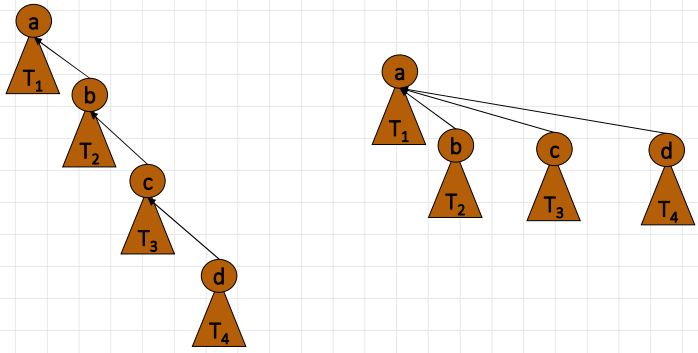

Up-Tree: Unione
a
d
e
s
c
h
t
u
z
g
Vittorio Maniezzo - Universita di Bologna
14

14

Compressione di cammini

Serve nel corso della Find-Set, fa puntare direttamente alla radice ogni nodo del cammino d'accesso al nodo dato.

Migliora la complessità asintotica se si eseguono più *find* che *union*.



15

Algoritmi per up-tree

Si associa ad ogni nodo x un intero *rank*[x], limite superiore all'altezza di x (num. archi del cammino più lungo fra x e una foglia discendente).

```
Make-Set(x)
p[x] = x
rank[x]=0
```

```
Find-Set(x)
if x ≠ p[x]
then p[x] = Find-Set(p[x])
return p[x]
```

```
Union(x,y)
Link(Find-Set(x),
     Find-Set(y))
```

```
Link(x,y)
if rank[x] > rank[y]
then  p[y] = x
else  p[x] = y
      if rank[x] == rank[y]
      then rank[y]++
```


16

Up-Tree - sommario

MakeSet	$\Theta(1)$
Find	$\Theta(h)$
Union	$\Theta(1)$

Ma quanto vale h?
Gli alberi hanno altezza logaritmica nel numero di nodi contenuti?

Necessaria una premessa.

17

Esponenziali di esponenziali

$F(i) = 2^{F(i-1)}$ per ogni $i > 0$

- $F(0) = 1$
- $F(1) = 2^1 = 2$
- $F(2) = 2^2 = 4$
- $F(3) = 2^{2^2} = 16$
- $F(4) = 2^{2^{2^2}} = 65536$
- $F(5) = 2^{2^{2^{2^2}}} = 2^{65536} \approx 10^{19728}$

Tutti i numeri incontrati normalmente sono più piccoli di $F(5)$

18

log*

$log^* n$ = il più piccolo i tale che $F(i) \geq n$ =

= il più piccolo i tale che $\underbrace{log\ log \ldots log}_{i \text{ volte}}\ n \leq 1$.

$\log^* n \leq 5$ per ogni numero n incontrato in pratica

La funzione di Ackerman

$A(1, j) = 2^j$ per $j \geq 1$

$A(i, 1) = A(i - 1, 2)$ per $i > 1$

$A(i, j) = A(i - 1, A(i,j - 1))$ per $i,j > 1$

Inversa della funzione di Ackerman (per $m \geq n$):

$\alpha(m,n)$ = il più piccolo $i \geq 1$ tale che $A(i, \lfloor m/n \rfloor) > \log n$

$\alpha(m,n) \leq 4$ per ogni valore comune di m e n

Up-Tree, complessitá

Lemma 1

Per tutte le radici x di alberi, $size[x] \geq 2^{rank[x]}$

Dimostrazione

Per induzione.

- Base, $rank[x] = 0$, ovvia.
- T: albero rango r, radice x, si cerca il minimo $size[x]$ possibile.
- T derivato da unione di T_1 e T_2, x era radice di T_1.
- Ipotesi induttiva: $size[T_1] \geq 2^{rank[T_1]}$, $size[T_2] \geq 2^{rank[T_2]}$
- Rango di $T_1 = r - 1$ (se fosse r, $size[T] > size[T_1] \geq 2^{rank[T_1]} = 2^{rank[T]}$, per ipotesi induttiva).
- Rango di $T_2 \leq$ rango di T_1, quindi rango di $T_2 = r - 1$.
- $size[T] \geq 2^{rank[T_1]} + 2^{rank[T_2]} = 2^{r-1} + 2^{r-1} = 2 \cdot 2^{r-1} = 2^r$
- Per l'ipotesi induttiva, lemma dimostrato.

Up-Tree: complessitá

Teorema

Una sequenza di m operazioni Make-Set, ***Link*** e Find-Set, di cui n sono operazioni Make-Set, può essere eseguita su una foresta di up-tree con unione per rango e compressione di cammini in tempo $O(m \log^* n)$.

Corollario

Una sequenza di m operazioni Make-Set, ***Union*** e Find-Set, di cui n sono operazioni Make-Set, può essere eseguita su una foresta di up-tree con unione per rango e compressione di cammini in tempo $O(m \log^* n)$.

NOTA: entrambi i bound sono in realtà migliorabili a $O(m\alpha(m,n))$, ma la dimostrazione è complessa.

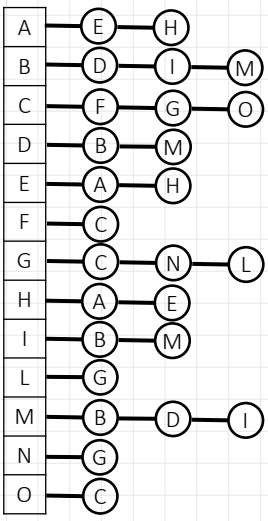
Una applicazione

Rete di calcolatori, con una rete di connessioni punto a punto bidirezionali.

E' possibile collegarsi da un qualsiasi calcolatore a qualsiasi altro?

E' possibile considerare le connessioni una alla volta e avere in ogni momento gli insiemi di calcolatori fra loro connessi (risoluzione *on-line*)?

Esercizio



Inserire i nodi del grafo in ordine alfabetico e determinare la struttura degli up-tree che si costruiscono a seguito delle corrispondenti chiamate a make-set e union.