Strutture dati per insiemi disgiunti

Sommario

- Strutture dati per insiemi disgiunti
- Determinazione delle componenti connesse di un grafo

Strutture dati per insiemi disgiunti

- In alcune applicazioni siamo interessati a mantenere gruppi distinti di oggetti
- Per queste applicazioni risultano importanti operazioni quali: determinare a quale insieme appartiene un oggetto e unire più insiemi

Insiemi disgiunti

- Una struttura dati per insiemi disgiunti mantiene una collezione $S=\{S_1,S_2,\ldots,S_k\}$ di insiemi dinamici disgiunti
- Ogni insieme S_i è identificato da un *rappresentante*
- Un rappresentante per un insieme S_i può essere
  - un qualsiasi membro dell'insieme S_i
  - un membro particolare dell'insieme S_i

Rappresentante di un insieme

- Se il rappresentante di un insieme è un qualunque elemento allora:
  - a fronte di una serie di operazioni di ricerca del rappresentante di uno stesso insieme deve sempre essere restituito lo stesso rappresentante
  - solo nel caso in cui l'insieme venga modificato tramite l'unione con un altro insieme l'elemento rappresentante può cambiare

Rappresentante di un insieme

- Il rappresentante può essere un elemento specifico dell'insieme
- Si devono definire delle caratteristiche degli insiemi e una regola per caratterizzare il rappresentante

- Ex: l'elemento più piccolo/grande di un insieme

Operazioni

- Dato un elemento x le operazioni definibili su una struttura dati per insiemi disgiunti sono:
  - Make-Set(x)
  - Union(x,y)
  - Find-Set(x)

Make-Set

- Crea un nuovo insieme il cui unico membro e rappresentante è x.
- Dato che gli insiemi sono disgiunti si deve garantire che x non appartenga già ad un altro insieme

Union

- Unisce due insiemi S_x e S_y che contengono gli elementi x e y.
- Si deve garantire che gli insiemi siano disgiunti.
- L'elemento rappresentante può essere un qualsiasi elemento dell'insieme unione $S_x \cup S_y$.
- Di solito si utilizza il rappresentante di S_x o di S_y come rappresentante finale.
- Dato che gli insiemi sono disgiunti l'operazione di unione deve distruggere i vecchi insiemi S_x e S_y.

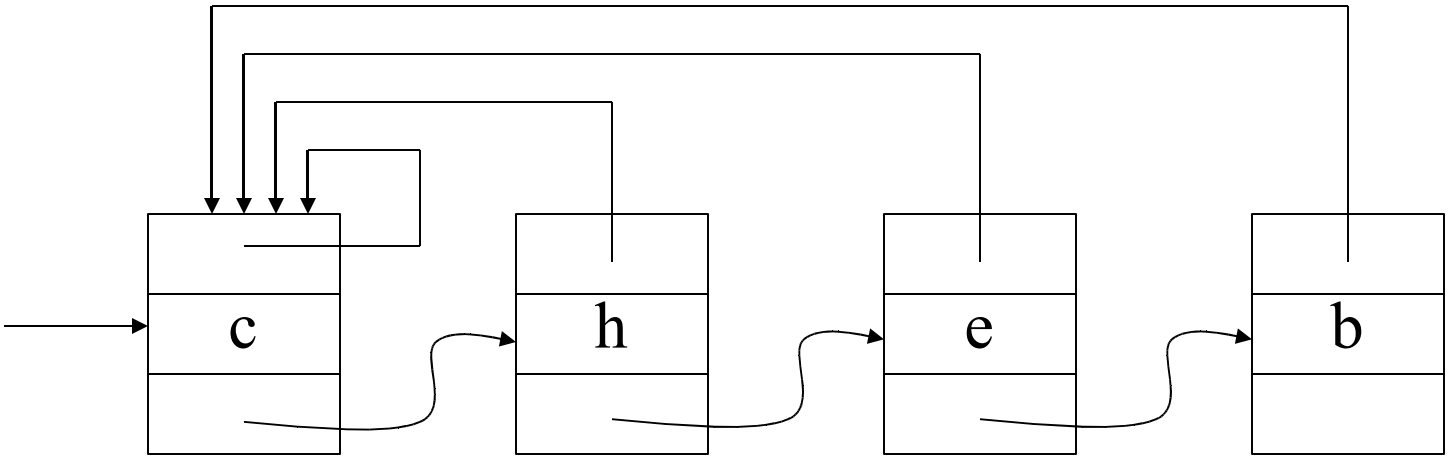
Find-Set

- Restituisce il rappresentante dell'unico insieme contenete x

Implementazione tramite **liste concatenate**

- Ogni insieme viene rappresentato con una lista concatenata
- Il primo oggetto di una lista viene utilizzato come rappresentante dell'insieme
- Ogni elemento nella lista contiene:
  - un oggetto
  - un puntatore all'elemento successivo
  - un puntatore al rappresentante

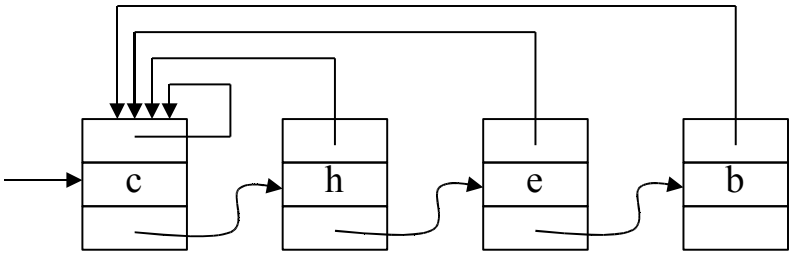
Visualizzazione



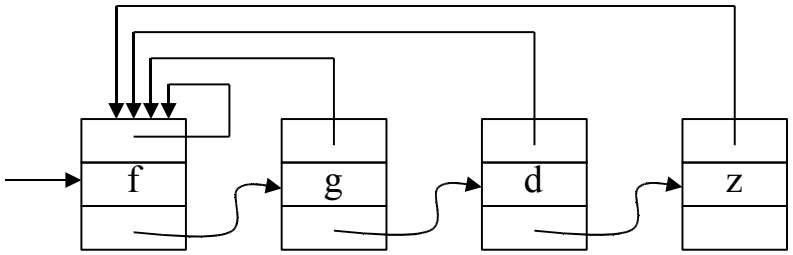
Operazioni

- Le operazioni Make-Set e Find-Set sono semplici e richiedono un tempo O(1):
  - Make-Set(x): crea una nuova lista concatenata in cui l'unico oggetto è x e inizializza il puntatore al rappresentante a x stesso
  - Find-Set(x): restituisce il puntatore da x al rappresentante
- L'operazione Union(x,y) richiede più tempo:
- La versione più semplice è quella in cui si appende la lista contentente x alla lista contentente y
- L'elemento rappresentante per l'unione è il rappresentante della lista contenente y

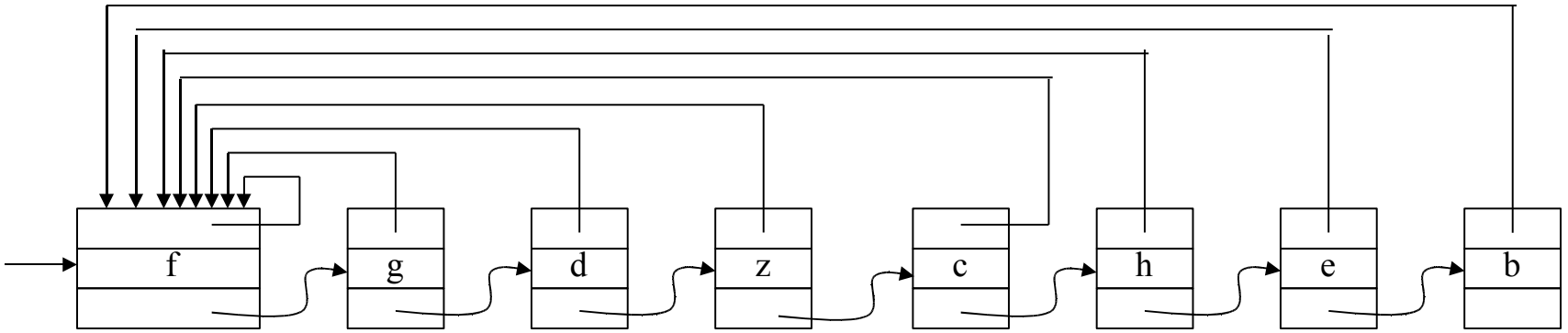
Visualizzazione dell'unione



x



y



Union(x,y)

Costo dell'unione

- L'operazione di Union richiede la modifica di tutti i puntatori al rappresentante dell'oggetto x e questo richiede in media un tempo lineare nella lunghezza della lista contentente x
- Nota: se si creano n insiemi di un unico oggetto e si uniscono nel seguente ordine:
  - Union(x_1,x_2)
  - Union(x_2,x_3)
  - Union(x_3,x_4)
  - Union(x_4,x_5)
  - Union(x_5,x_6)
- si devono fare 1, 2, 3,…,n modifiche ai puntatori, per un totale di $O(n^2)$ operazioni!

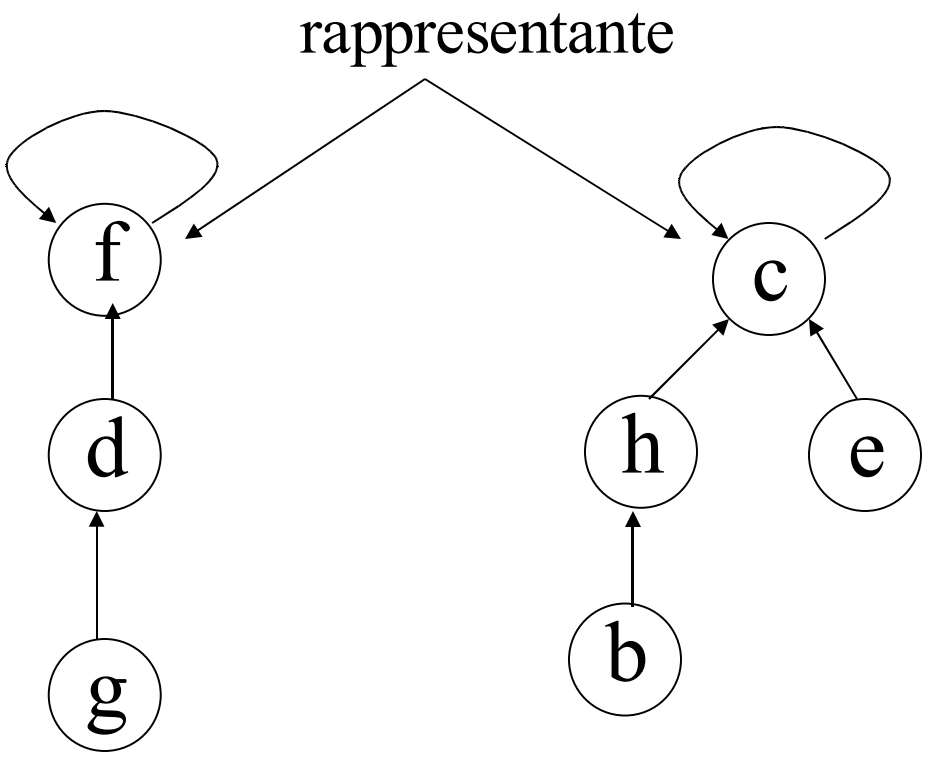
Eurisitica del peso

- Una strategia per diminuire il costo dell'operazione Union consiste nel:
  - memorizzare l'informazione della lunghezza della lista
  - così da appendere la lista più corta a quella più lunga
- Risulta poi facile mantenere l'informazione di lunghezza della lista dopo l'unione in tempi O(1)
  - la lunghezza complessiva è la somma delle lunghezze dei due set uniti, operazione che richiede un tempo costante, indipendente dalla dimensione dei set.

Implementazione tramite **foreste**

- Si può realizzare una implementazione più veloce delle strutture dati per insiemi disgiunti rappresentando ogni insieme tramite un albero radicato
- Ogni nodo dell'albero contiene l'oggetto e un puntatore al padre
- L'oggetto rappresentante è l'oggetto contenuto nella radice dell'albero
- La radice ha come padre un puntatore a se stessa

Visualizzazione

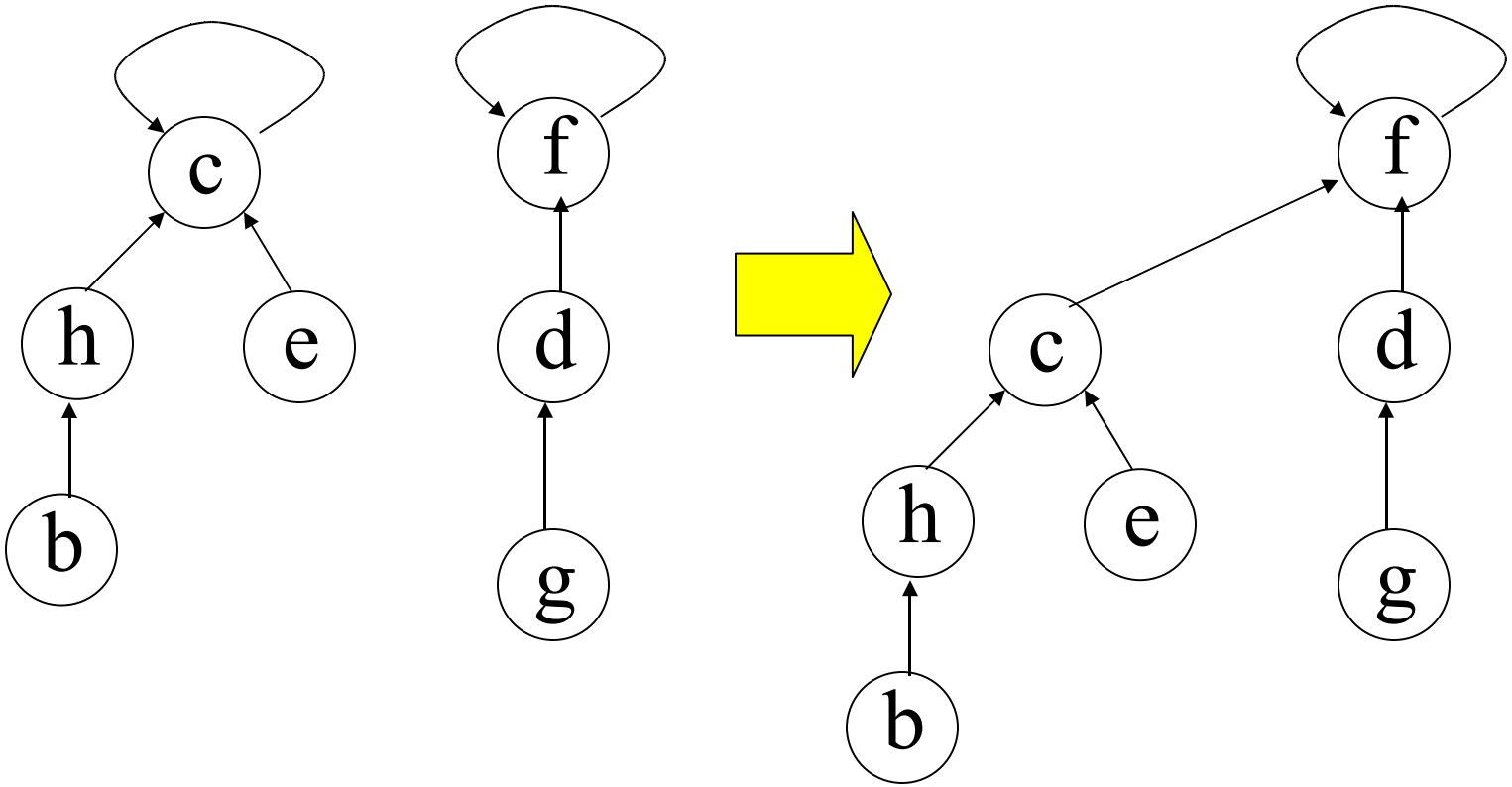


Nota: gli alberi sono rappresentati tramite nodi con un unico puntatore!

Operazioni

- Make-Set(x): crea un albero con un unico nodo x e con padre se stesso
- Find-Set(x): risale la lista dei padri di x fino a trovare la radice e restituisce la radice come oggetto rappresentante
- Union-Set(x,y): determina le radici dei due alberi contenteti x e y e fa diventare la radice dell'albero contentente x un figlio della radice dell'albero contentente y

Visualizzazione dell'operazione di unione



Euristiche

- Di per sé la rappresentazione tramite foreste non è più veloce della rappresentazione con liste, ma su questa rappresentazione è possibile introdurre delle euristiche che rendono questa implementazione la più veloce conosciuta

- Euristiche:
  - unione per rango
  - compressione dei cammini

Euristica del **rango**

- Si mantiene una informazione aggiuntiva per ogni nodo: il *rango* memorizzata nell'attributo rank[x]
- Il rango di un nodo è il *limite superiore all'altezza* di x, ovvero il limite superiore per il numero di archi del cammino più lungo fra x e una foglia sua discendente
- Nota: il rango non e' l'altezza dell'albero

Euristica del **rango**

- E’ simile all’unione pesata con le liste concatenate
- L’idea è di fare sì che mediamente l’albero più basso diventi un sottoalbero di quello più alto
- Ovvero che la radice dell’albero più basso punti alla radice dell’albero più alto in una operazione di Union

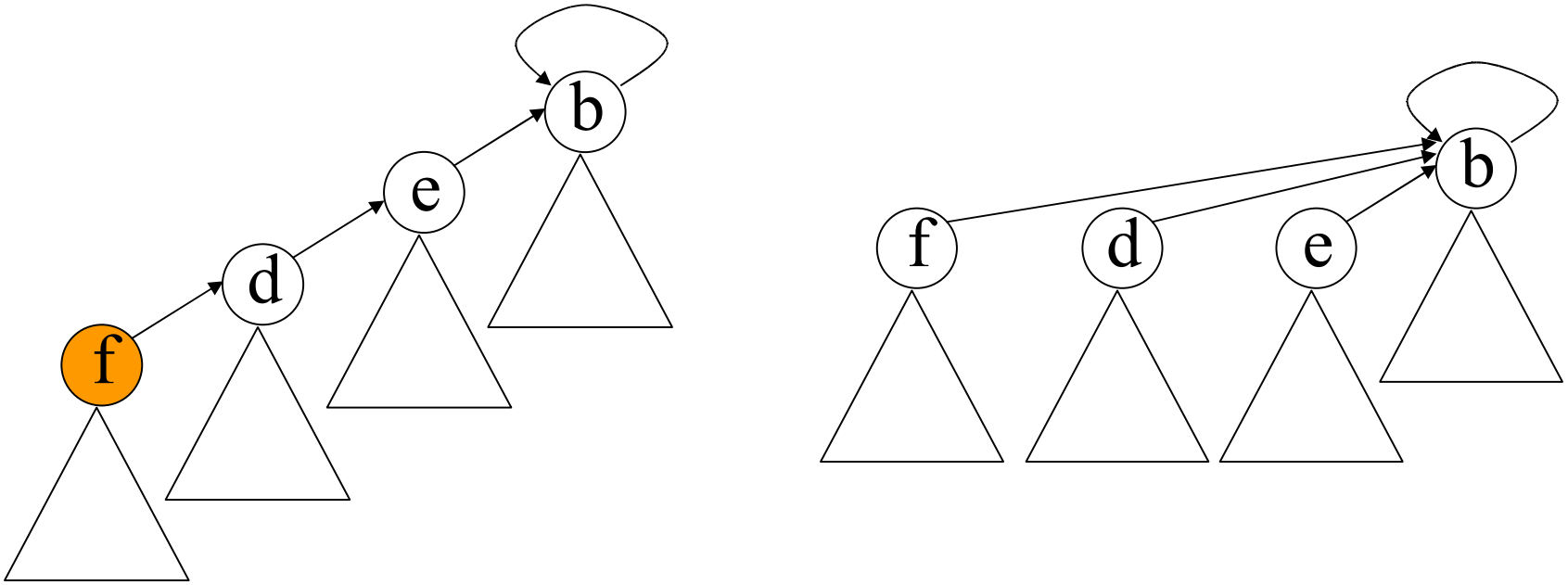
Euristica del **rango**

- Nota:il rango della radice non è l'altezza dell'albero, ma solo un limite superiore.
- Questo rende più semplice la sua manipolazione e meno costoso il suo mantenimento.
- In particolare se gli alberi sono manipolati in modo da diminuirne l'altezza, questo non si riflette sul rango della radice che rimane inalterato
- Cioè non si aggiorna l'altezza se questa diminuisce ma solo se aumenta

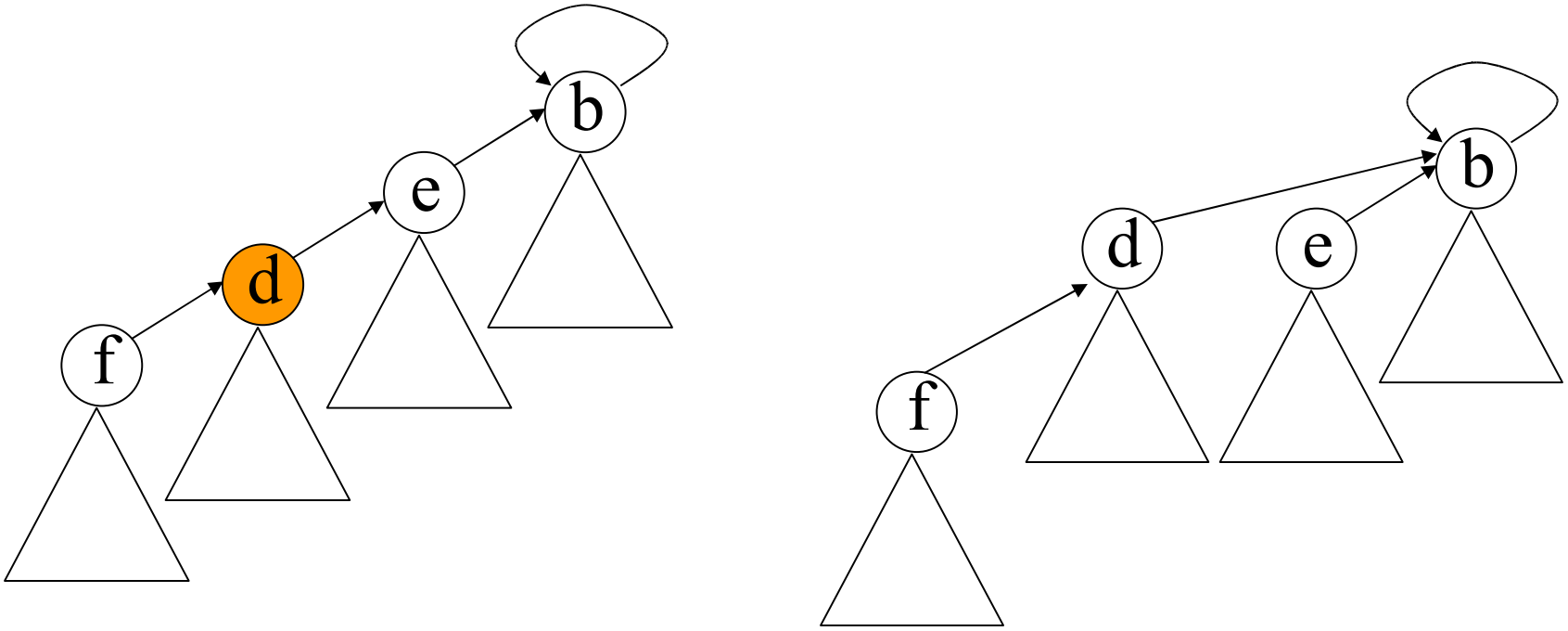
Euristica della **compressione dei cammini**

- La compressione dei cammini viene usata per le operazioni Find-Set(x)
- L'idea è di modificare un albero in modo che in ricerche successive di x, l'elemento rappresentante venga restituito in modo più efficiente (in O(1)!)
- Per fare questo si modifica il puntatore al padre di ogni nodo sul cammino da x alla radice perché punti direttamente alla radice

Visualizzazione compressione dei cammini per l'operazione Find-Set(f)



Visualizzazione compressione dei cammini per l'operazione Find-Set(d)



Spiegazione intuitiva

- Quando viene creato un nuovo insieme con Make-Set il rango iniziale della radice (l'unico nodo nell'albero) è 0
- L'esecuzione dell'operazione Find-Set non modifica i ranghi
  - infatti non cambia il limite superiore dell'altezza (riduce l'altezza degli alberi, non la incrementa)
- Per l'operazione di Union
  - si rende la radice con rango maggiore padre di quella con rango minore; il rango della radice non cambia
  - in caso di eguaglianza si sceglie arbitrariamente una delle due radici come padre; il rango della radice aumenta di 1

Pseudocodice

```
Make-Set(x)
1  p[x] ← x
2  rank[x] ← 0

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

Link(x,y)
1  if rank[x] > rank[y]
2  thenp[y] ← x
3  elsep[x] ← y
4       if rank[x]=rank[y]
5       then   rank[y] ← rank[y]+1
```

Pseudocodice

```
Find-Set(x)
1  if x ≠ p[x]
2  then p[x] ← Find-Set(p[x])
3  return p[x]
```

Find-Set

- La procedura Find-Set è una procedura a due passate:
  - nella prima si risale il cammino di accesso per determinare la radice
  - nella seconda si discende e si aggiornano tutti i nodi in modo che puntino alla radice
- Se x=p[x], ovvero siamo arrivati alla radice, la procedura restituisce p[x]. Questo è il passo base della ricorsione.
- Altrimenti si chiama ricorsivamente la procedura aspettando di determinare e poi restituire il puntatore alla radice

Analisi

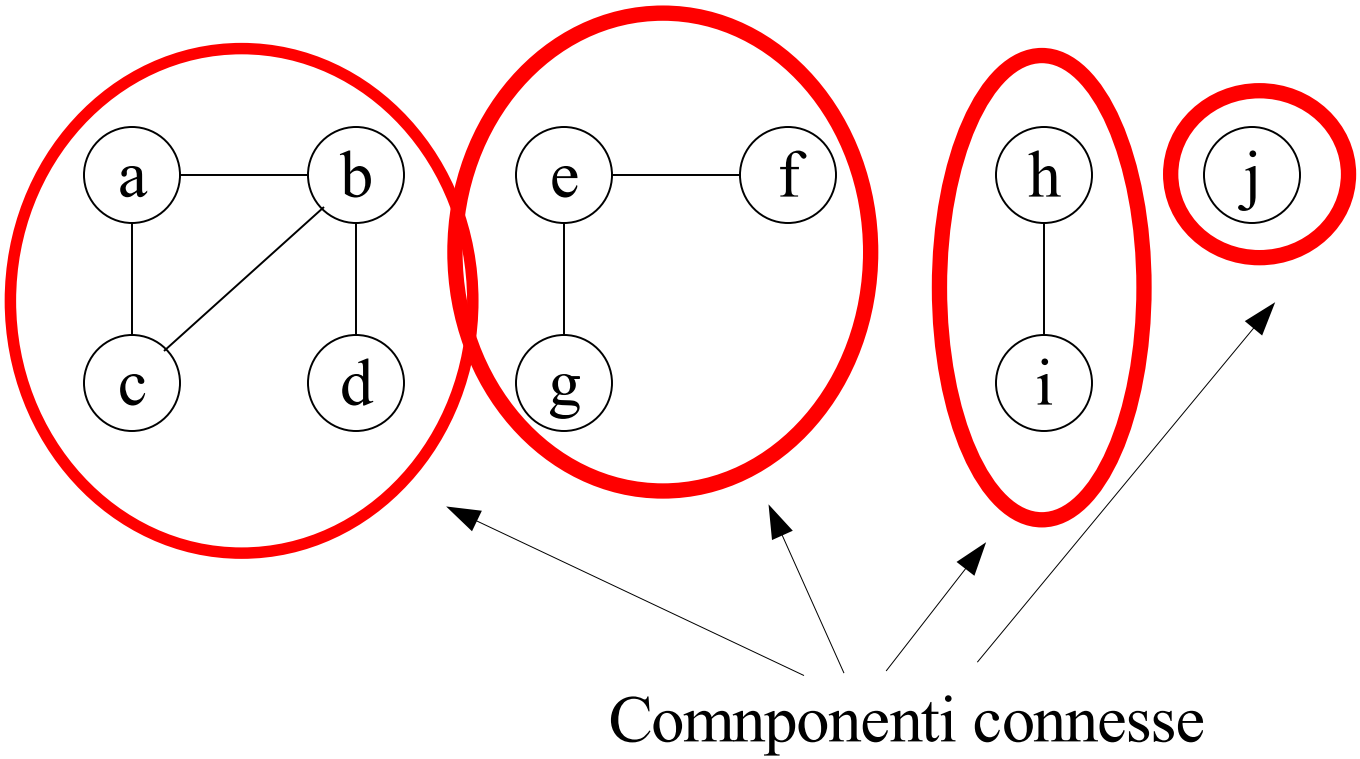
- Non vedremo una analisi dettagliata
- L'uso delle euristiche dell'unione per rango e della compressione dei cammini con la rappresentazione di foreste di insiemi disgiunti porta ad una efficienza pari a O(n) in tutti i casi di applicazioni pratiche con n pari al numero di operazioni Make,Union,Find che si vogliono eseguire sugli insiemi disgiunti.

Componenti connesse di un grafo

- Una applicazione delle strutture dati per insiemi disgiunti consiste nella determinazione delle componenti connesse di un grafo non orientato

- Una *componente connessa* di un grafo G è un sottografo G' tale per cui esiste un cammino semplice (tutti i vertici del cammino sono distinti) per ogni coppia di vertici di G'

Visualizzazione di componenti connesse di un grafo non orientato



Idea intuitiva

- L’idea è di partire da componenti connesse costituite da un unico vertice
- Ogni componente connessa viene poi unita ad altre componenti connesse tramite l’operazione di Union
- L’unione fra due insiemi viene fatta se esiste un arco fra due qualsiasi vertici in tali insiemi

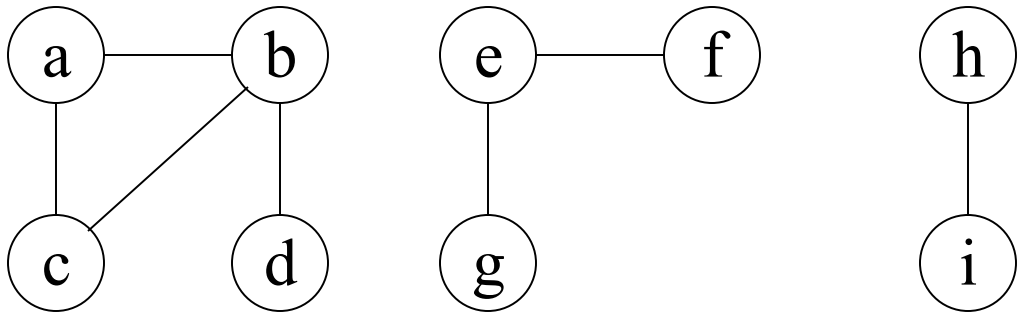
Idea intuitiva

- Infatti se abbiamo che:
  - un vertice x è raggiungibile da ogni altro vertice in un insieme S_x
  - e analogamente per un vertice y in un insieme S_y
  - ed esiste un arco che collega x e y
- Allora ne consegue che l'unione fra S_x e S_y è un insieme connesso dato che da ogni elemento in S_x si può adesso raggiungere un qualsiasi elemento in S_y tramite l'arco (x,y) e viceversa

Pseudocodice

```
Connected-Components(G)
1  for all v in V[G]
2  do  Make-Set(v)
3  for all (u,v) in E[G]
4  do  if Find-Set(u)≠ Find-Set(v)
5      then Union(u,v)
```

Esempio



Arco	Collezione insiemi disgiunti							
	a	b	c	d	e	f	g	h	i
(b,d)	a	b,d	c		e	f	g	h	i
(e,g)	a	b,d	c		e,g	f		h	i
(a,c)	a,c	b,d			e,g	f		h	i
(h,i)	a,c	b,d			e,g	f		h,i	
(a,b)	a,c,b,d				e,g	f		h,i	
(e,f)	a,c,b,d				e,g,f			h,i	
(b,c)	a,c,b,d				e,g,f			h,i	