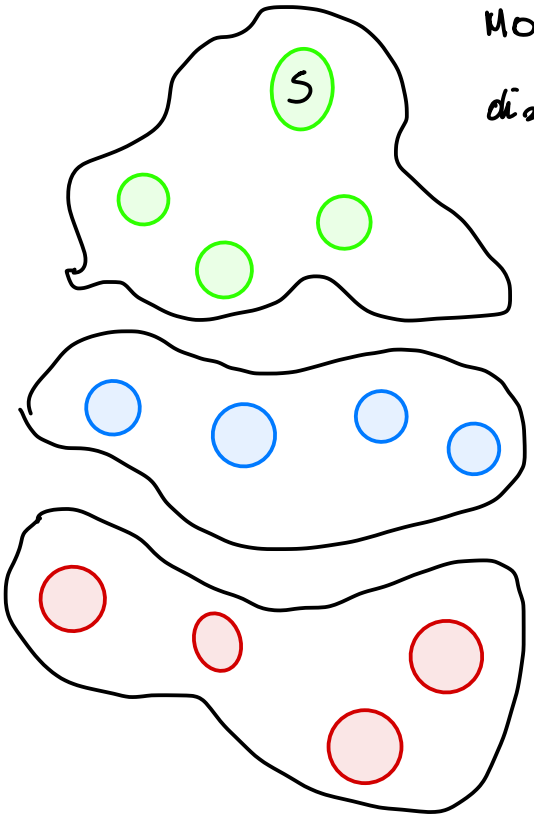
- GENERICA ITERAZIONE ALGORITMO DI DIJKSTRA



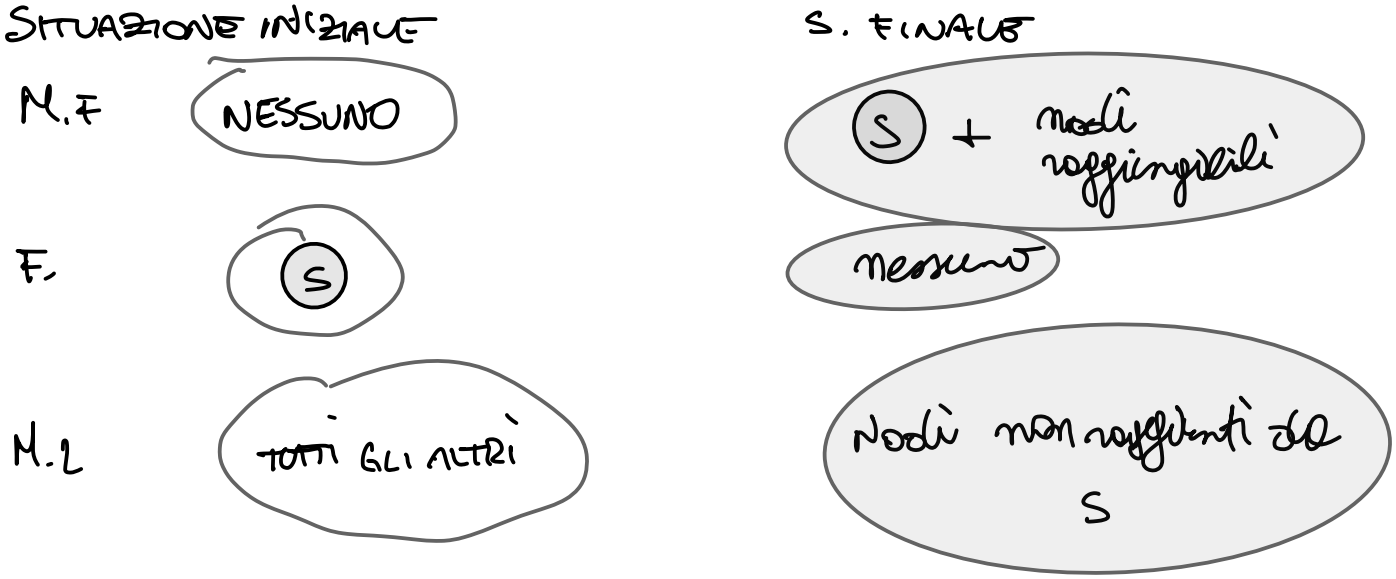
MONDO FERMO (VISITATI)

dist[u] = +∞ → non cambierà più

FRONTIERA (SCOPERTI, NON VISITATI)
nodi già scoperti, dist[u] ≠ +∞ ma potrebbe ancora cambiare sono ancora in coda)

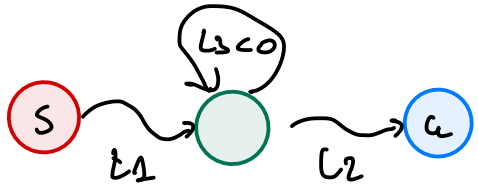
NON SCOPERTI dist[u] = +∞

nodi non ancora raggiunti (sono ancora in coda)


SITUAZIONE INIZIALE
M.F
NESSUNO
F.
s
M.2
TUTTI GLI ALTRI
S. FINALE
s + nodi raggiungibili
nessuno
Nodi non raggiunti da s

COSA SUCCEDE SE $c: E \to \mathbb{R}$ (INVECE CHE $\mathbb{R}^+$) ?

- gli archi potranno quindi avere pesi negativi



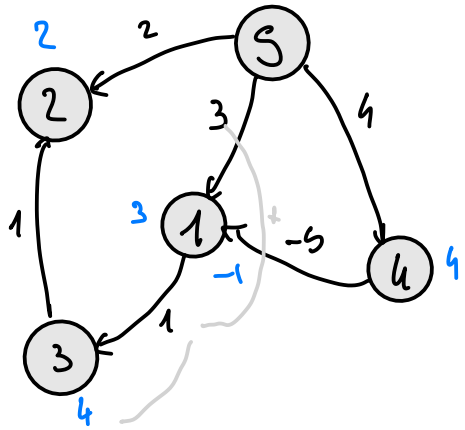
distanza da s a u ?

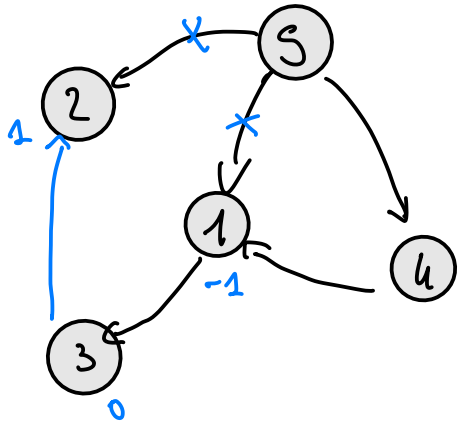
- NON riesco a definire un cammino minimo tra due nodi perché non c'è; continuerei a fare il ciclo all'infinito per trovare il cammino minimo
- Devo controllare che non esistano cicli negativi altrimenti il problema è malposto

PROBLEMA

INPUT
- $G = (V, E)$ (Diretto / indiretto) senza cicli negativi
- funzione di costo non negativa sugli archi $c: E \to \mathbb{R}$
- nodo sorgente $s \in V$

OUTPUT Albero dei cammini minimi radicato in s





- CAMMINI MINIMI CON DIJKSTRA
- CAMMINI MINIMI CORRETTI

- NON posso permettermi di non considerare alcuni nodi considerati in precedenza

ALGORITMO DI BELLMAN-FORD (CAMMINI MINIMI DA SORGENTE SINGOLA)

```
Bellman-Ford(G,c,s)
for all u ∈ V
  dist[u] := +∞
  prev[u] := -1
dist[s] := 0
```

INIZIALIZZAZIONE

```
for i=1 to |V|-1 do
  for all (u,v) ∈ E do
    if dist[v] > dist[u] + c(u,v)
     then
       dist[v] := dist[u] + c(u,v)
       prev[v] := u
return prev[]
```

$O(n-1)$

← RILASSAMENTO

↓

$O(m)$

$O((n-1)m)$

↓

$O(n \cdot m)$

CAMMINO MINIMO DA s A u

prev[u_2] = s

↓

1° ITERAZIONE DI RILASSAMENTI

Cammino + breve da s a u_2

3° ITERAZIONE DI RILASSAMENTI

prev[u_3] = u_2

e ogni iterazione +1 arco per il cammino minimo s → v

2° ITERAZIONE DI RILASSAMENTI

prev[u_2] = u_2

4° ITERAZIONE DI RILASSAMENTI

prev[v] = u_3

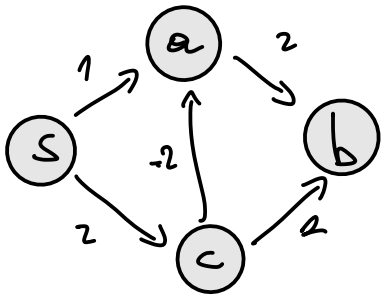
- Nelle successive iterazioni i cammini minimi non cambieranno più
- Ad ogni iterazione scegliamo il cammino più breve tra nodi interni per trovare il cammino minimo tra s e v.
- # iterazioni → |V|-1

CASO NON CICLI NEGATIVI.

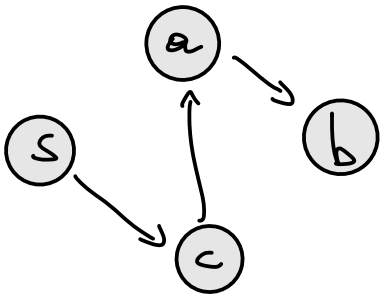
- si aggiunge un'iterazione di rilassamento ⇒ se si trova un cammino più breve per un solo nodo allora ∃ un ciclo negativo

nodi \ # iterazioni	0	1	2	3
s	0	0	0	0
a	+∞	1	0	0
b	+∞	3	2	2
c	+∞	2	2	2

prev	a	b	c
s	a	b	c
-1	~~x~~	~~x~~	~~x~~
.	~~s~~	a	s
	c		



|V| - 1 = 3



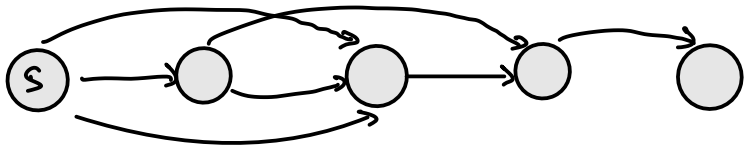
ALBERO DEI CAMMINI MINIMI

3° ORA

CAMMINI MINIMI DA SORGENTE SINGOLA SU DAG

→ Abbiamo una proprietà che ci permette di scegliere l'ordine dei nodi da considerare ?

DAG LINEARIZZATO



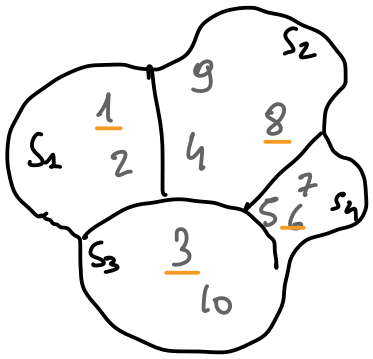
ordine registrazione (primi 10 m)

DISJOINT SET

- Struttura dati astratta che memorizza una partizione di un insieme generico (insieme di elementi → numeri da 1 ... n).

- Partizione di un insieme: insieme S di k sottoinsiemi t.c. ognuno degli S_i è un sottoinsieme dell'insieme di partenza ($S_i \subseteq X$) e contemporaneamente non ci sono intersezioni ($S_i \cap S_j = \emptyset \quad i \neq j$). L'unione di tutti gli S_i è X ($\bigcup_{i=1}^{k} S_i = X$)

Esempio



PRIMITIVE

- **Make-Set(x)**: restituisce un Disjoint set in cui $S_i = \{i\}, \forall i = 1, \dots, |x|$
- **Union(D,x,y)**: unisce i due sottoinsiemi a cui appartengono x e y
- **Find-Set(D,x)**: dato un elemento restituisce il rappresentante dell'insieme a cui appartiene x.
  Serve per capire se due elementi sono nello stesso insieme.

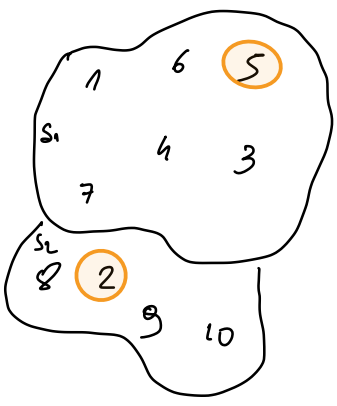
→ Una volta messi insieme due sottoinsiemi non posso più separarli!!!

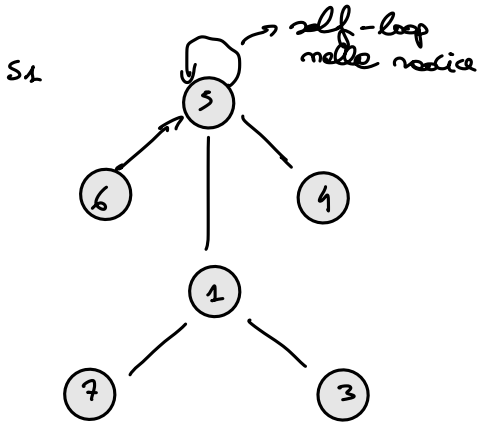
Come si può rappresentare questi dati per implementare questa struttura dati?

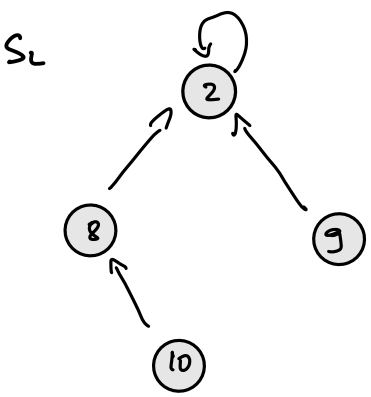
→ FORESTA di ALBERI → ogni albero contiene gli elementi di un sottoinsieme
VANTAGGIO: semplicità e poco spazio occupato!

IMPLEMENTAZIONE MEDIANTE "ARRAY DI PADRI"

Esempio







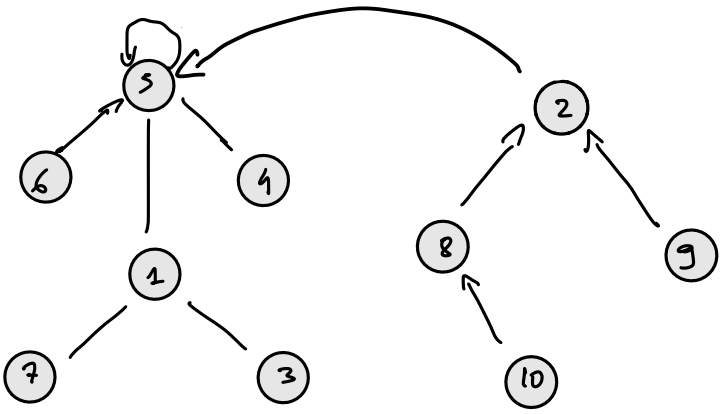
ARRAY DI PADRI

1	2	3	4	5	6	7	8	9	10
5	2	1	5	5	5	1	2	2	8

Il self loop serve per indicare la radice

Esempio

Union (DS, 8, 5)

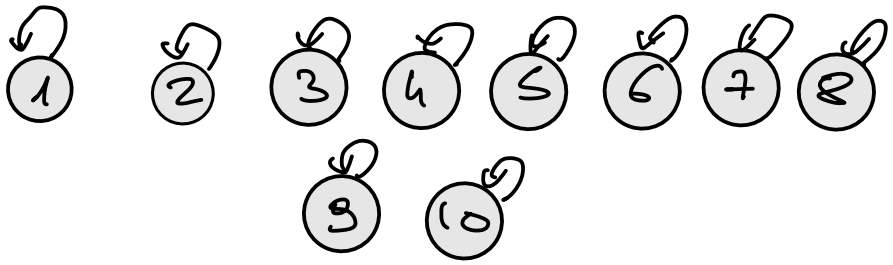


ARRAY DI PADRI DOPO LA UNION

1	2	3	4	5	6	7	8	9	10
5	~~2~~ 5	1	5	5	5	1	2	2	8

Esempio

Make-Set(x)



Ognuno è padre di sè stesso

↳ 10 self-loop → ogni nodo è anche radice

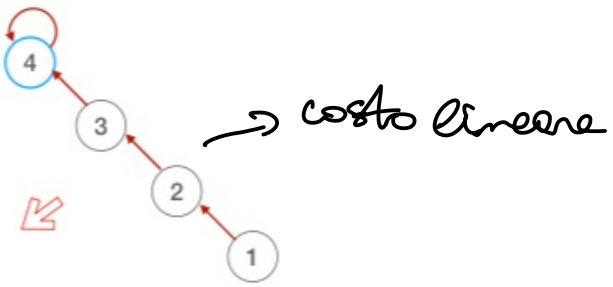
COSTI COMPUTAZIONALI

Make-Set(x) → DS O(n)

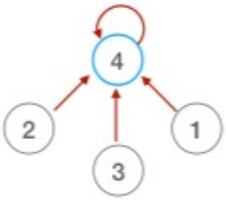
Find-Set(DS, x) → rappresentante O(h)

Union(DS, x, y) → O(h) costo impiegato per trovare il rappresentante di ogni DisJoint


```
Union(DS,1,2)
Union(DS,1,3)
Union(DS,1,4)
....
```



→ costo lineare



diverso albero, stessa informazione

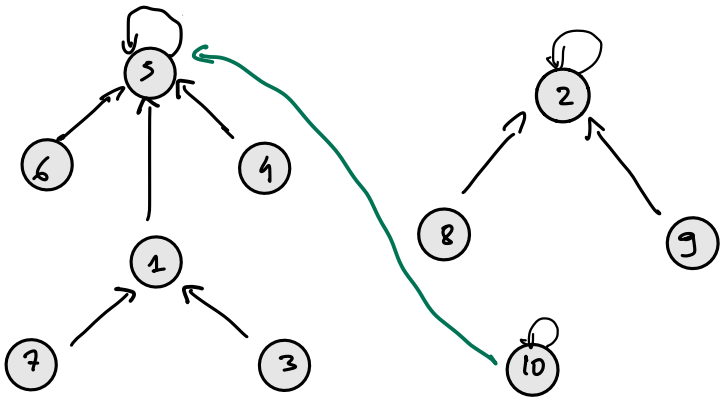
costo costante nella ricerca del rappresentante

L'obiettivo è non aumentare l'altezza dell'albero

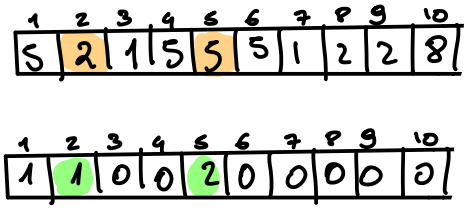
- Durante una Union l'albero più basso diventa sottoalbero di quello più alto
- Quando ho due alberi alti uguali l'altezza sarà h+1, in tutti gli altri casi non aumenta

Per compiere queste operazioni devo considerare anche il costo per la ricerca dell'altezza dell'albero.

↳ Oltre al vettore di padri avrò un array rank che memorizza l'altezza degli alberi radicati nei rappresentanti.



DS



Union (DS, 5, 10)

Pseudo-Codice

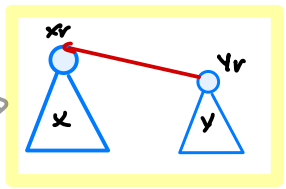
```
Make-Set(x):
    DS.p := new_array[1...n]
    DS.rank := new_array[1...n]
    for i = 1 to n
        DS.p[i] = i
        DS.rank[i] := 0
    return DS
```

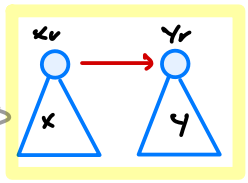
Pseudo-Codice

```
Find-Set(DS, x)
  if DS.p[x] = x
     then return x
  else return
        Find-Set(DS, DS.p[x])
```

Pseudo-Codice

```
Union(DS, x, y):
    xr := Find-Set(DS, x)      } rappresentanti di x e y
    yr := Find-Set(DS, y)      }
    if xr ≠ yr
       then if DS.rank[xr] > DS.rank[yr]
               then DS.p[yr] := xr
               else DS.p[xr] := yr
                    if DS.rank[xr] = DS.rank[yr]
                       then DS.rank[yr] := DS.rank[yr] + 1
```





Proposizione → un albero con radice R ha almeno $2^{rank[R]}$ nodi.

⇓

un albero di radice r con k nodi

⇓

$k \geq 2^{rank[r]}$

$\log k \geq rank[r]$

Dimostrazione

— o CASO BASE . caso in cui non sono ancora state fatte union

⇓

caso dopo Make-Set(x) prima della prima Union

⇓

ogni albero ha esattamente un nodo e, per ogni nodo r, rank[r] = 0

⇓

$2^{rank[r]} \leq 1$ ok!

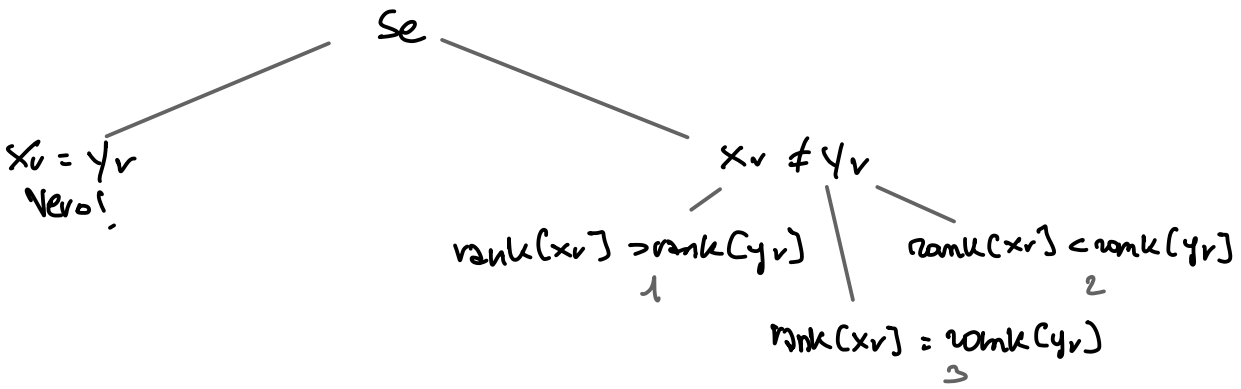
→ IPOTESI INDUTTIVA

Obiettivo: dimostrare che dopo $t \geq 0$ op. di Union per ogni albero di k sn nodi con rappresentante r vale che

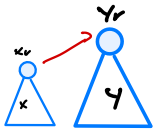
$$2^{rank[r]} \leq k$$

→ PASSO INDUTTIVO.

Obiettivo: dimostrare che dopo $t+1$ operazioni di Union l'ipotesi rimane verificata



CASO 1



$\Rightarrow$ Il nuovo albero ha come nodi $|S_x \cup S_y| = |S_x| + |S_y| \geq |S_y| \geq 2^{rank[y_r]}$

non ci sono elementi comuni

insieme di x

insieme di y

nuovo rappresentante

Ho dimostrato la proposizione

CASO 2 → dimostrato analogamente

CASO 3

HP INDUTTIVA

#nodi nuovo albero $|S_x \cup S_y| = |S_x| + |S_y| \geq 2^{rank[x_r]} + 2^{rank[y_r]}$

$\Downarrow$

$2^{rank[y_r]} + 2^{rank[y_r]}$

$\Downarrow$

$2^{rank[y_r]+1}$

$\Rightarrow$ # nodi nuovo albero $\geq$ $2^{\text{nuovo rank } y_r}$ $2^{rank[y_r]+1}$

Abbiamo quindi dimostrato che la Find e la Union possono essere fatte con costo logaritmico invece che lineare instanziando però un array in più in memoria per l'altezza dei vari alberi