ALGORITMI E STRUTTURE DATI

Prof. Manuela Montangero

A.A. 2022/23

STRUTTURE DATI:
Disjoint Set

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Disjoint Set

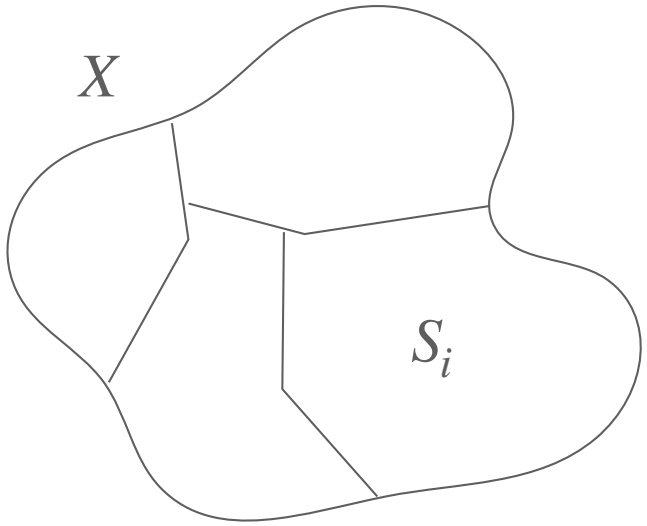
a.k.a Union-Find set o Merge-Find set

Disjoint Set:

Memorizza una collezione di sottoinsiemi disgiunti di un insieme generativo.
Ammette operazioni di ricerca e unione, ma non inserzione e cancellazione.

Insieme generativo: $X = \{1,2,\ldots,n\}$

Insieme di sottoinsiemi disgiunti: $S = \{S_1, \ldots, S_k\}$, con $1 \leq k \leq n$, tali che



$$S_i \subseteq X \; \forall i = 1,\ldots,k$$

$$S_i \cap S_j = \varnothing \; \forall i \neq j, \;\; i,j = 1,\ldots,k$$

$$\cup_{i=1}^{k} S_i = X$$

Disjoint Set

a.k.a Union-Find set o Merge-Find set

Disjoint Set:

Memorizza una collezione di sottoinsiemi disgiunti di un insieme generativo.
Ammette operazioni di ricerca e unione, ma non inserzione e cancellazione.

Insieme generativo: $X = \{1,2,\ldots,n\}$

Insieme di sottoinsiemi disgiunti: $S = \{S_1, \ldots, S_k\}$, con $1 \leq k \leq n$, tali che

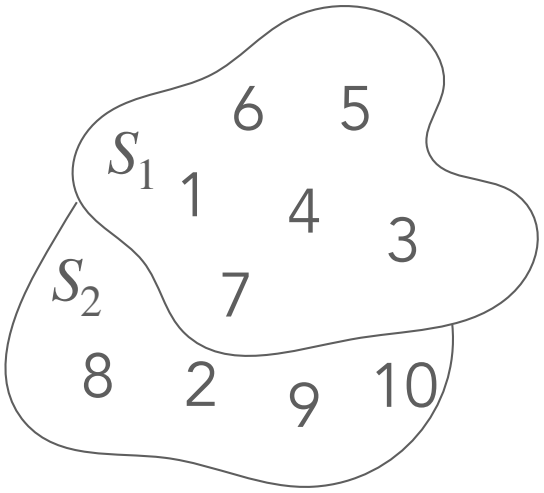
$$S_i \subseteq X, \forall i = 1,\ldots,k$$
$$S_i \cap S_j = \emptyset,\ \forall i \neq j,\ \ i,j = 1,\ldots,k$$
$$\cup_{i=1}^{k} S_i = X$$

Primitive:

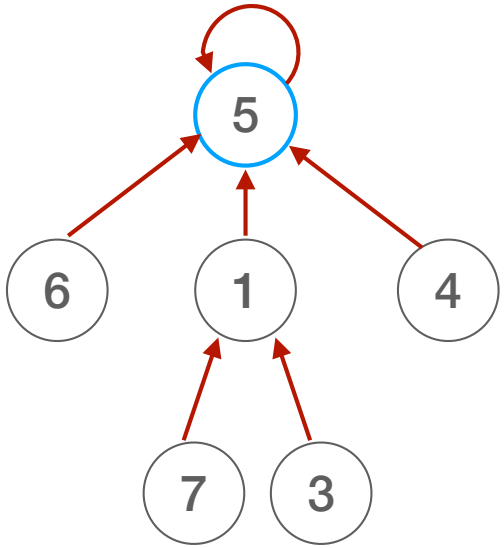
- `Make-set(`X`):` restituisce un DisjointSet in cui $S_i = \{i\}, \forall i = 1,\ldots,|X|$
- `Find-set(DS,`x`):` restituisce un rappresentante dell'insieme a cui appartiene x
- `Union(DS,`x`,`y`):` unisce gli insiemi a cui appartengono x e y

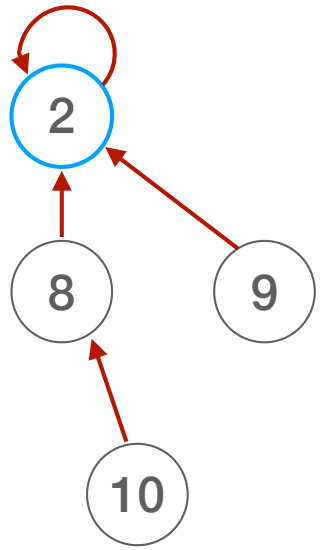
Disjoint Set

Realizzazione mediante FORESTA di alberi



Ogni insieme
è rappresentato
da un albero





Ogni insieme ha un rappresentate,
che è la radice dell'albero

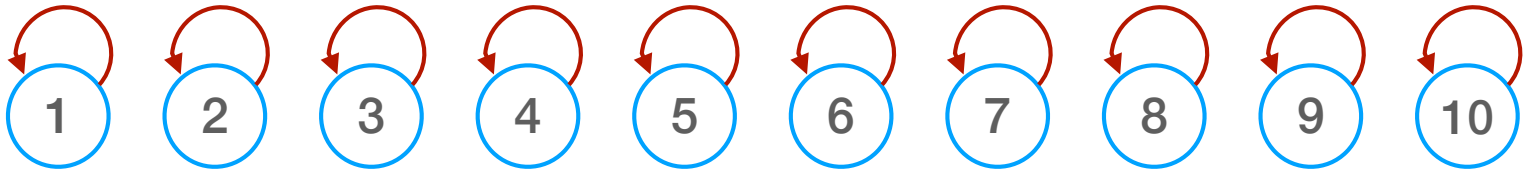
La foresta è rappresentata con il vettore dei padri

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

Disjoint Set

Primitive:

- `Make-set(`X`)`: restituisce un DisjointSet in cui $S_i = \{i\}, \forall i = 1,\ldots,|X|$



DS

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

```
Make-set(X)
 DS := new_array[1..n]
 for i=1 to n
  DS[i] := i
 return DS
```

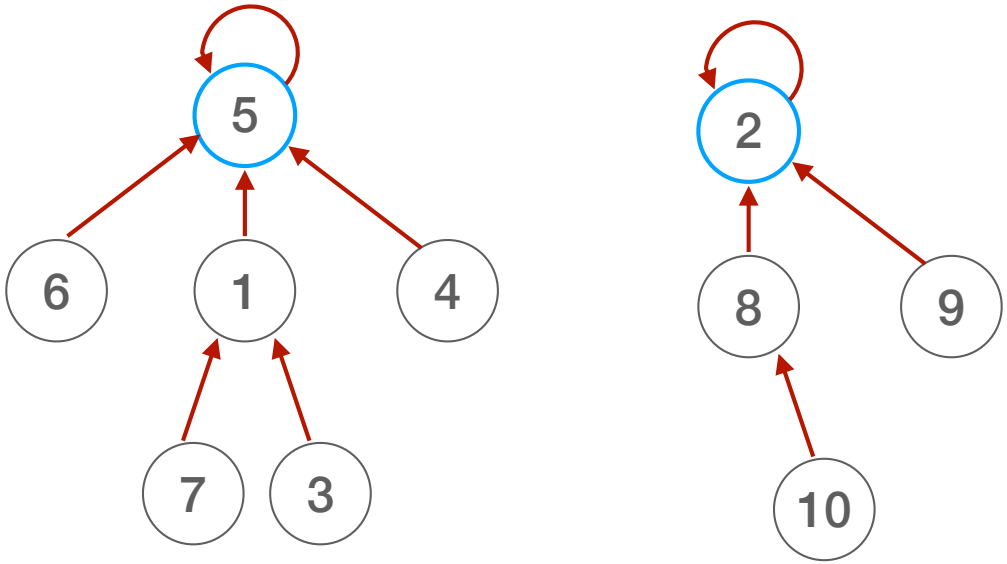
Costo computazionale

$O(n)$

Disjoint Set

Primitive:

- `Find-set(DS,`x`):` restituisce un rappresentante dell'insieme a cui appartiene x




```
Find-set(DS,5) = 5
Find-set(DS,7) = 5
Find-set(DS,9) = 2
```

DS

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

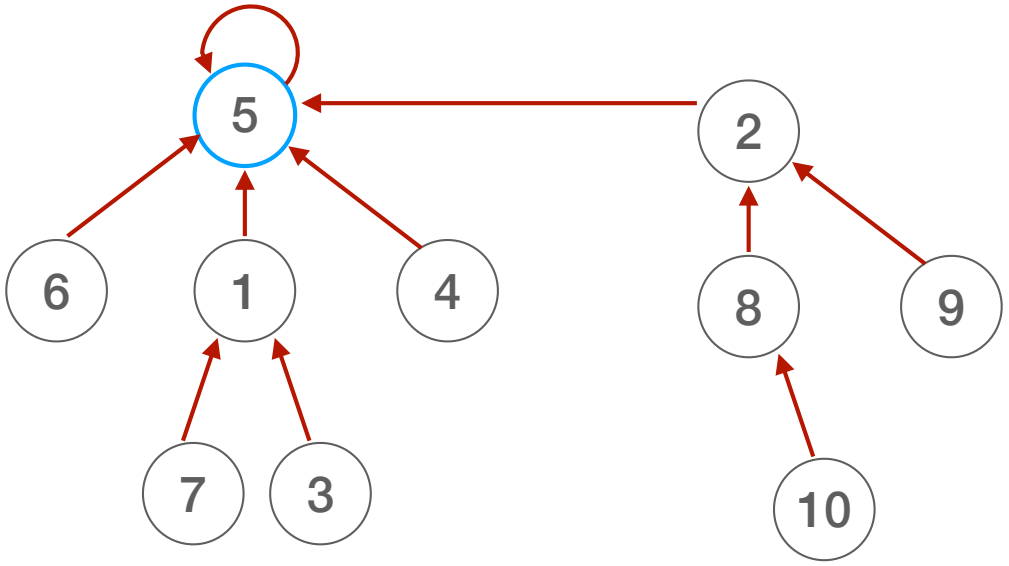
```
Find-set(DS,x)
 if DS[x] = x
   then return x
   else return Find-set(DS,DS[x])
```

Costo computazionale
lineare nell'altezza dell'albero
a cui appartiene x $O(n)$

Disjoint Set

Primitive:

- `Union(DS,`x`,`y`):` unisce gli insiemi a cui appartengono x e y



`Union(DS,8,5)`

```
Union(DS,x,y)
 xr := Find-set(DS,x)
 yr := Find-set(DS,y)
 if xr ≠ yr
   then DS[xr] := yr
```

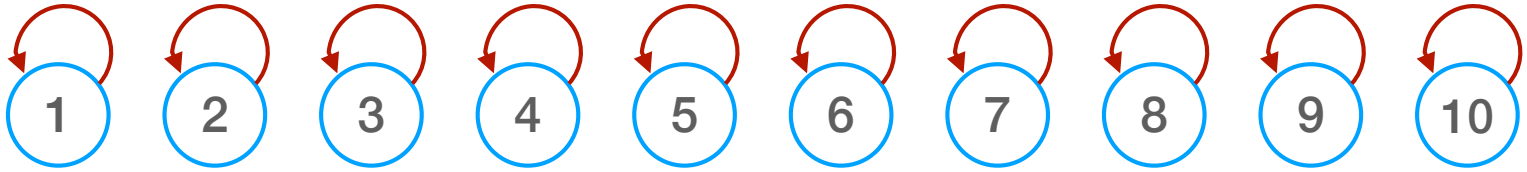
DS

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

Costo computazionale
lineare nella somma delle altezze
degli alberi a cui appartengono x e y $O(n)$

Disjoint Set

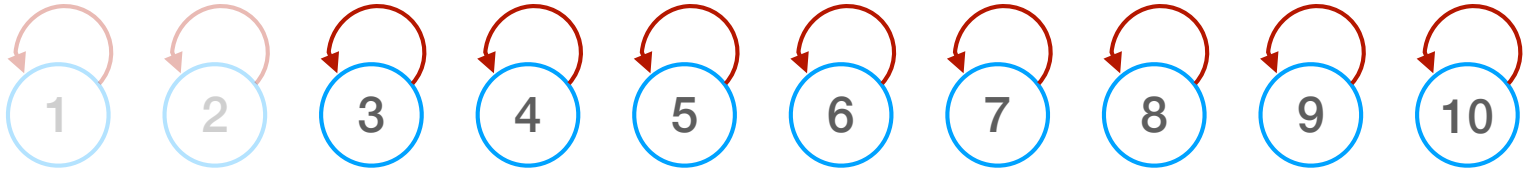
Quale sequenza di operazioni genera un albero di altezza $\Omega(n)$?



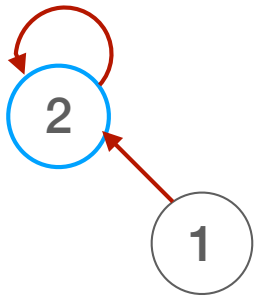

```
Union(DS,1,2)
```

Disjoint Set

Quale sequenza di operazioni genera un albero di altezza $\Omega(n)$?

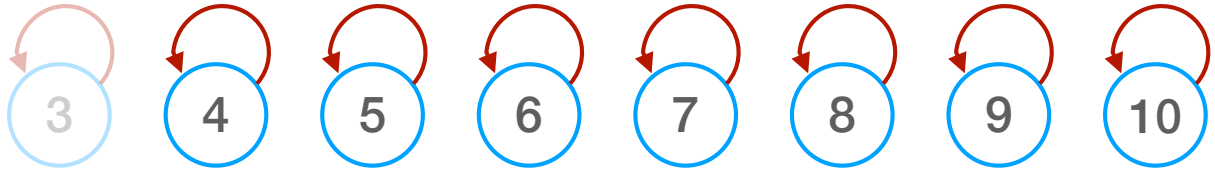



```
Union(DS,1,2)
Union(DS,1,3)
```

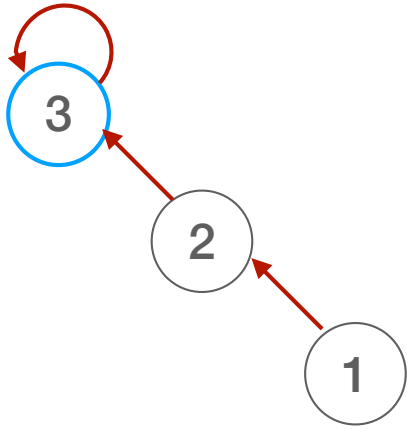


Disjoint Set

Quale sequenza di operazioni genera un albero di altezza $\Omega(n)$?

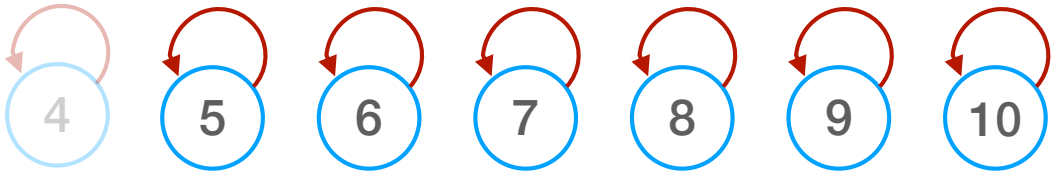



```
Union(DS,1,2)
Union(DS,1,3)
Union(DS,1,4)
```



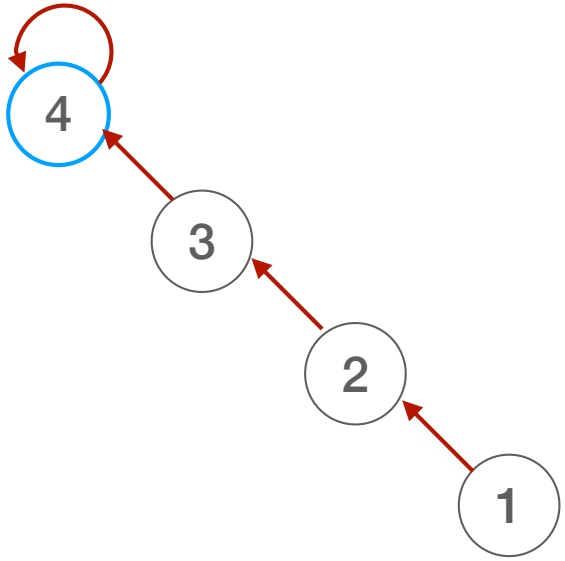
Disjoint Set

Quale sequenza di operazioni genera un albero di altezza $\Omega(n)$?



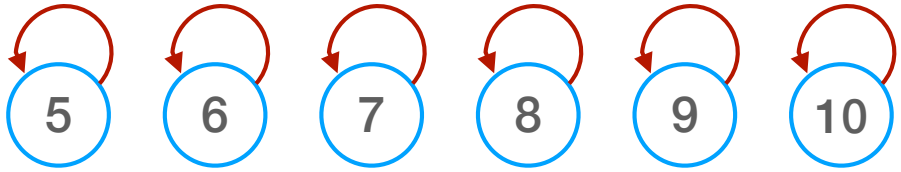

```
Union(DS,1,2)
Union(DS,1,3)
Union(DS,1,4)

    ....
```

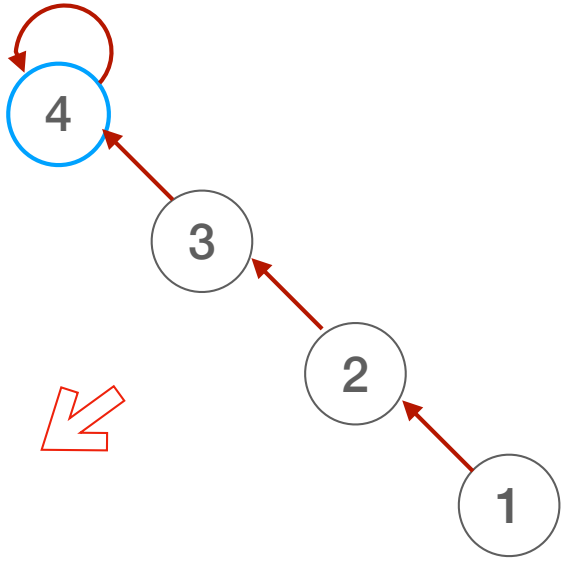


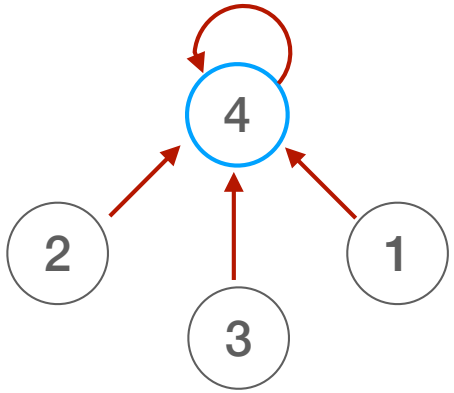
Disjoint Set

Quale sequenza di operazioni genera un albero di altezza $\Omega(n)$?




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



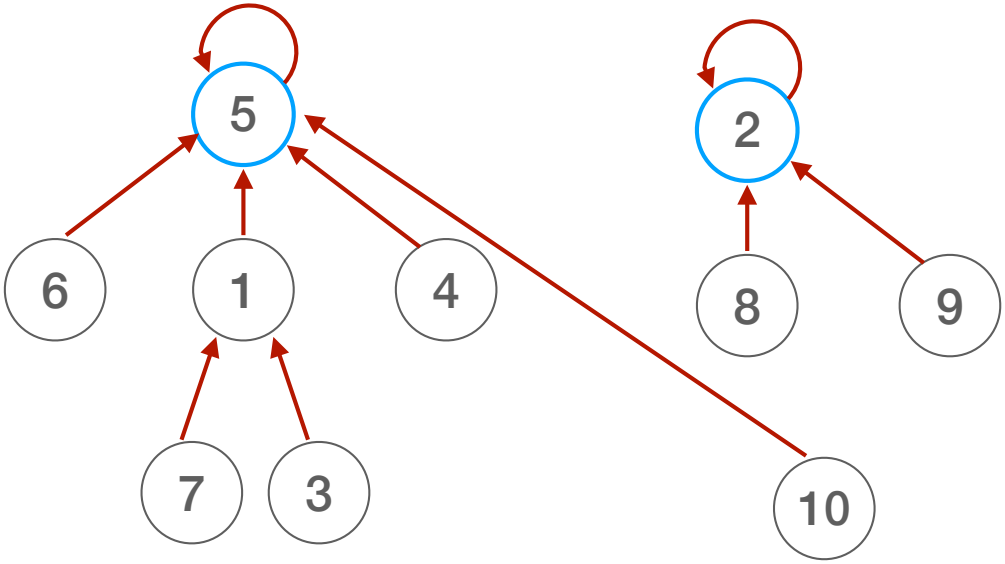


diverso albero, stessa informazione

Disjoint Set

EURISTICA del RANGO

Nella `Union` l'albero più basso diventa sottoalbero di quello più alto



`rank[1..n]` memorizza l'altezza degli alberi radicati nei rappresentanti

`p[1..n]` vettore dei padri

```
Union(DS,5,10)
Union(DS,10,5)
```

DS

`p`

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

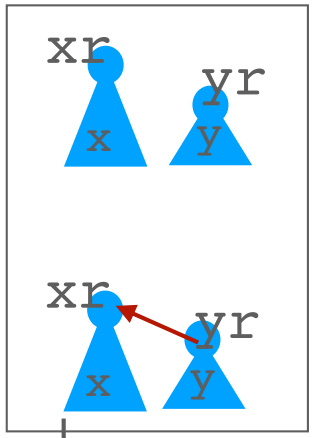
`rank`

1	2	3	4	5	6	7	8	9	10
1	1	0	0	2	0	0	0	0	0

Disjoint Set

EURISTICA del RANGO

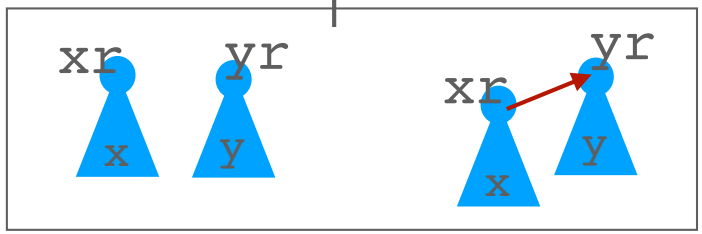
Nella `Union` l'albero più basso diventa sottoalbero di quello più alto




```
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 := 0
 return DS
```

```
Union(DS,x,y)
 xr := Find-set(DS,x)
 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
```

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



Disjoint Set

EURISTICA del RANGO

Nella `Union` l'albero più basso diventa sottoalbero di quello più alto

```
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 := 0
 return DS
```

$O(n)$

```
Union(DS,x,y)
 xr := Find-set(DS,x)
 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[xr] := DS.rank[xr] +1
```

O(rank[rappresentante di x] +rank[rappresentante di y])

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

O(rank[rappresentante di x])

Costo computazionale?

Disjoint Set

EURISTICA del RANGO

PROPOSIZIONE:

Usando l'euristica del rango, un albero della foresta con radice `r` ha almeno $2^{\texttt{rank[r]}}$ nodi

un albero di radice `r` con `k` nodi

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

$$\log \texttt{k} \geq \texttt{rank[r]}$$

Costo computazionale `Find` e `Union` $O(\log n)$

Disjoint Set

EURISTICA del RANGO

PROPOSIZIONE:

Usando l'euristica del rango, un albero della foresta con radice `r` ha almeno $2^{\texttt{rank[r]}}$ nodi

Dimostrazione per induzione sul numero di operazioni **`Union`**

CASO BASE:

dopo **`Make-set(X)`**, prima della prima **`Union`**

- ogni albero ha esattamente un nodo $\Rightarrow 1 = 2^0$
- per ogni nodo `r` si ha `rank[r] = 0`

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

Disjoint Set

EURISTICA del RANGO

PROPOSIZIONE:

Usando l'euristica del rango, un albero della foresta con radice `r` ha almeno $2^{\texttt{rank[r]}}$ nodi

Dimostrazione per induzione sul numero di operazioni

IPOTESI INDUTTIVA:

dopo $\texttt{t} \geq 0$ operazioni `Union`, per un albero di $\texttt{k} \leq \texttt{n}$ nodi con rappresentante `r` vale che

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

PASSO INDUTTIVO: dimostriamo che l'ipotesi induttiva vale anche dopo `t+1` operazioni `Union`

Disjoint Set

EURISTICA del RANGO

IPOTESI INDUTTIVA:

dopo `t` operazioni **`Union`**, per un albero di $k \leq n$ nodi con rappresentante `r` vale che

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

PASSO INDUTTIVO: dimostriamo che l'ipotesi induttiva vale anche dopo `t+1` operazioni **`Union`**

`Union(DS,x,y)` se **`xr = yr`** —> VERO

Disjoint Set

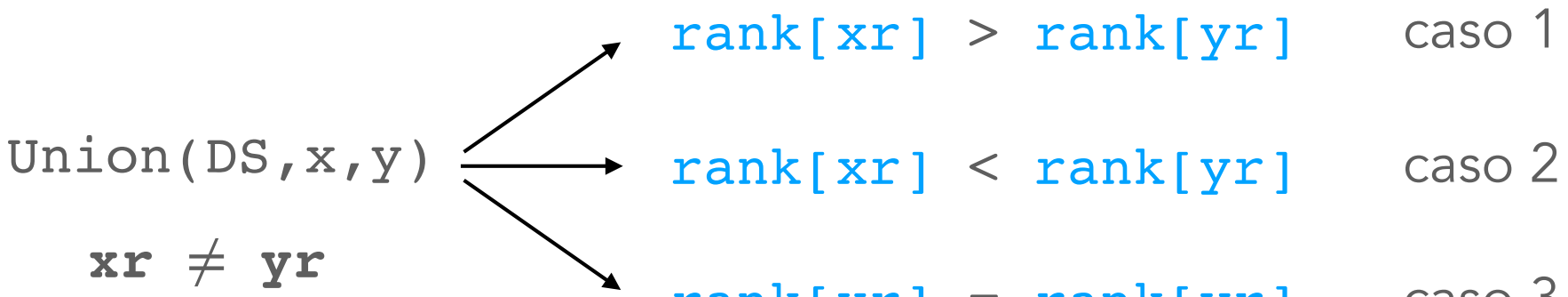
EURISTICA del RANGO

IPOTESI INDUTTIVA:

dopo `t` operazioni **`Union`**, per un albero di $k \leq n$ nodi con rappresentante `r` vale che

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

PASSO INDUTTIVO: dimostriamo che l'ipotesi induttiva vale anche dopo `t+1` operazioni **`Union`**



Disjoint Set

EURISTICA del RANGO

IPOTESI INDUTTIVA:

dopo `t` operazioni **`Union`**, per un albero di $k \leq n$ nodi con rappresentante `r` vale che

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

PASSO INDUTTIVO: dimostriamo che l'ipotesi induttiva vale anche dopo `t+1` operazioni **`Union`**

`Union(DS,x,y)`

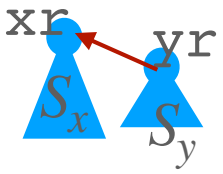
$xr \neq xy$

S_x = insieme a cui appartiene `x`

S_y = insieme a cui appartiene `y`

$rank[xr] > rank[yr]$

`xr` diventa il nuovo rappresentante
ma il suo rango non cambia



$$|S_x \cup S_y| > |S_x| \geq 2^{rank[xr]}$$

di nodi nuovo insieme

Ipotesi induttiva

Disjoint Set

EURISTICA del RANGO

IPOTESI INDUTTIVA:

dopo `t` operazioni **`Union`**, per un albero di $k \leq n$ nodi con rappresentante `r` vale che

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

PASSO INDUTTIVO: dimostriamo che l'ipotesi induttiva vale anche dopo `t+1` operazioni **`Union`**

`Union(DS,x,y)`

$xr \neq xy$

S_x = insieme a cui appartiene `x`

S_y = insieme a cui appartiene `y`

$rank[xr] < rank[yr]$

simmetrico rispetto al caso 1

Disjoint Set

EURISTICA del RANGO

IPOTESI INDUTTIVA:

dopo `t` operazioni **`Union`**, per un albero di $k \leq n$ nodi con rappresentante `r` vale che

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

PASSO INDUTTIVO: dimostriamo che l'ipotesi induttiva vale anche dopo `t+1` operazioni **`Union`**

`Union(DS,x,y)`

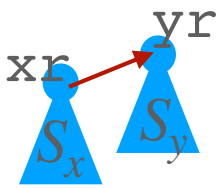
$xr \neq xy$

S_x = insieme a cui appartiene `x`

S_y = insieme a cui appartiene `y`

$rank[xr] = rank[yr]$

`xr` diventa il nuovo rappresentante
e il suo rango viene incrementato di 1



perché disgiunti

Ipotesi induttiva

$$|S_x \cup S_y| = |S_x| + |S_y| \geq$$

nuovo rango di `yr`

$$2^{rank[xr]} + 2^{rank[yr]} = 2^{rank[yr]+1}$$

Disjoint Set

EURISTICA del RANGO

Nella `Union` l'albero più basso diventa sottoalbero di quello più alto

```
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 := 0
 return DS
```
$O(n)$

```
Union(DS,x,y)
 xr := Find-set(DS,x)
 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[xr] := DS.rank[xr] +1
```
$O(\log n)$

```
Find-set(DS,x)
 if DS.p[x] = x
   then return x
   else return Find-set(DS,DS.p[x])
```
$O(\log n)$

Costo computazionale