Strutture dati per insiemi disgiunti (union-find)

- Una struttura dati per insiemi disgiunti mantiene una collezione $S = \{ S_1, S_2, ..., S_k \}$ di insiemi dinamici disgiunti.
- Ciascun insieme è identificato da un *rappresentante*, che è un elemento dell'insieme.
- Ogni elemento di un insieme è rappresentato da un oggetto x.

Operazioni

`MAKE-SET(x)`

crea un nuovo insieme il cui unico elemento, e rappresentante, è x.

`UNION(x,y)`

unisce gli insiemi dinamici disgiunti che contengono x e y (ad esempio S_x e S_y) in un unico insieme che è l'unione dei due insiemi. Gli insiemi S_x e S_y sono eliminati dalla collezione S.

`FIND(x)`

restituisce un puntatore al rappresentante dell'unico insieme che contiene x.

Osservazione

- I tempi di esecuzione sono analizzati in funzione di due parametri:

  `n` = numero di operazioni `MAKE-SET`

  `m` = numero totale di operazioni `MAKE-SET`, `UNION`, `FIND`.

- Dato che gli insiemi sono disgiunti, ciascuna `UNION` riduce di una unità il numero degli insiemi:
  - dopo `n-1` `UNION`, resta un solo insieme;
  - dunque ci possono essere al più `n-1` `UNION`.
- Le operazioni `MAKE-SET` sono incluse nel numero totale `m` di operazioni, quindi $m \geq n$.
- Si suppone infine che le `n` operazioni `MAKE-SET` siano le prime operazioni eseguite.

Semplice implementazione

- Ogni **insieme** è implementato come una lista concatenata.
- Il primo oggetto in ogni lista è il rappresentante del suo insieme.
- Ogni oggetto x nella lista contiene:
  - un elemento dell'insieme;
  - un puntatore all'oggetto che contiene il successivo elemento dell'insieme;
  - un puntatore alla propria lista.
- Ogni lista mantiene i puntatori:
  - *head* (che punta al rappresentante)
  - *tail* (che punta all'ultimo oggetto nella lista)

MAKE-SET e FIND

```
MAKE-SET(x)
   lista.head = lista.tail = x;
   x.lista = lista;
   x.next = null;

FIND(y)
   return y.lista.head;
```

Le due operazioni richiedono tempo costante O(1).

UNION(x,y)

- Si aggiunge la lista di `x` alla fine della lista di `y`.
- Si utilizza il puntatore `tail` per la lista di `y` per trovare rapidamente il punto in cui appendere la lista di `x`.
- Il rappresentante del nuovo insieme è l'elemento che originariamente era il rappresentante dell'insieme che conteneva `y`.
- Infine, si aggiorna il puntatore alla propria lista per ogni oggetto che originariamente si trovava nella lista di `x`.
- Questo passo richiede tempo lineare nella lunghezza della lista di `x`.

UNION(x,y)

```
UNION(x,y)
   listaX = x.lista;
   listaY = y.lista;
   z = listaX.head;
   while (z != null) {
       z.lista = listaY;
       z = z.next;
   }
   listaY.tail.next = listaX.head;
   listaY.tail = listaX.tail;
```

Esempio

- Si può trovare una sequenza di *m* operazioni su *n* oggetti che richiede tempo $\Theta(n^2)$:
- Oggetti: $x_1, x_2, \ldots, x_n$.
- Si eseguono *n* `MAKE-SET` seguite da *n-1* `UNION`:

  `UNION(`x_1`,`x_2`)`

  `UNION(`x_2`,`x_3`)`

  `UNION(`x_3`,`x_4`)`

  …

  `UNION(`x_{n-1}`,`x_n`)`

Esempio

- $m = 2n - 1$
- Tempo $\Theta(n)$ per le n operazioni `MAKE-SET`
- L'i-esima operazione `UNION` aggiorna il riferimento alla propria lista di i oggetti
- Quindi il numero totale di oggetti aggiornati da tutte le $n-1$ `UNION` è $\Theta(n^2)$
- Il numero totale di operazioni è $2n-1$, e ciascuna richiede in media un tempo $\Theta(n)$.

Euristica dell'unione pesata

- *Si mantiene per ogni lista un campo che memorizza la sua lunghezza.*
- *Si appende sempre la lista più corta alla lista più lunga.*

OPERAZIONI

```
MAKE-SET(x)
   lista.head = lista.tail = x;
   lista.length = 1;
   x.lista = lista;
   x.next = null;

FIND(y)
   return y.lista.head;
```

```
UNION(x,y)
   if (x.lista.length <= y.lista.length) {
       corta = x.lista;
       lunga = y.lista;
   }
   else {
       corta = y.lista;
       lunga = x.lista;
   }
   z = corta.head;
   while (z != null) {
       z.lista = lunga;
       z = z.next;
   }
   lunga.tail.next = corta.head;
   lunga.tail = corta.tail;
   lunga.length = corta.length + lunga.length;
```

ANALISI

- Una singola operazione `UNION` può ancora richiedere tempo $\Omega(n)$, se entrambi gli insiemi hanno $\Omega(n)$ elementi.
- Ma ...

TEOREMA

Con l'euristica dell'unione pesata, una sequenza di m operazioni `MAKE-SET`, `UNION` e `FIND`, n delle quali sono `MAKE-SET`, impiega tempo

$O(m + n \log n)$.

Dimostrazione

- Consideriamo un oggetto x. Ogni volta che x.lista viene aggiornato, x deve trovarsi nell'insieme più piccolo.
  - La prima volta che questo puntatore viene aggiornato, l'insieme risultante deve contenere almeno 2 elementi;
  - la seconda volta, almeno 4 elementi;
  - …
  - l'i-esima volta, l'insieme risultante deve avere almeno 2^i elementi.
- Dato che il numero complessivo di elementi è n, l'insieme più grande può avere al più n elementi, da cui segue $2^i \leq n$, quindi $i = O(\log n)$.
- Quindi, il puntatore alla propria lista di ciascun oggetto può essere aggiornato $O(\log n)$ volte in tutte le operazioni `UNION`.

Dimostrazione

- Gli aggiornamenti degli altri puntatori (*head* e *tail*) e delle lunghezze delle liste richiedono invece tempo costante, per ogni `UNION`.
- Il tempo totale per aggiornare gli *n* oggetti è *O(n log n).*
- Dato che `MAKE-SET` e `FIND` impiegano tempo *O(1)*, e ci sono *O(m)* di queste operazioni, il tempo per l'intera sequenza di *m* operazioni risulta pari a *O(m + n log n).*