07/03/2024

PROBLEMA GESTIONE DI INSIEMI DISGIUNTI (UNION-FIND).

→ STRUTTURA DATI X RISOLVERE IL PROBLEMA MST

TIPO DI DATO

- SERIE (A, B, C) DISGIUNTI TRA LORO

  ES: S = 1, 2, 3

  A = {1} / B = {2, 3}

OP

- MAKESET(x) → DA ELE. x CREA SET CON NOME x
- UNION(A, B) → NEW SET CON EL = A ∪ B, NOME SET = A.
- FIND(x) → RETURN NOME SET CHE CONTIENE x.

OBB!

IMPLEMENTARE UNA S.D. EFFICIENTE X UN NUM. ARBITRARIO DI OPERAZIONI (BETTER COSTO AMMORTIZZATO).

vedremo due tipi di approccio, tutti e due con un idea generale in comune...

IDEA: INSIEMI DISGIUNTI = FOREST DI ALBERI RADICATI

↓

ALBERI = INSIEMI → RADICE = NAME SET.

1° APPROCCIO: ALBERO QUICKFIND

STRUTTURA

- FOREST DI ALBERI $h = 1$,
- $\forall$ ALBERO →
  - RADICE = NOME SET
  - FOGLIE = ELEM. (INCLUSO ROOT)

IMPLEMENTAZIONE

classe `QuickFind` **implementa** `UnionFind`:
dati: $S(n) = O(n)$
una collezione di insiemi disgiunti di elementi $elem$; ogni insieme ha un nome $name$.

operazioni:

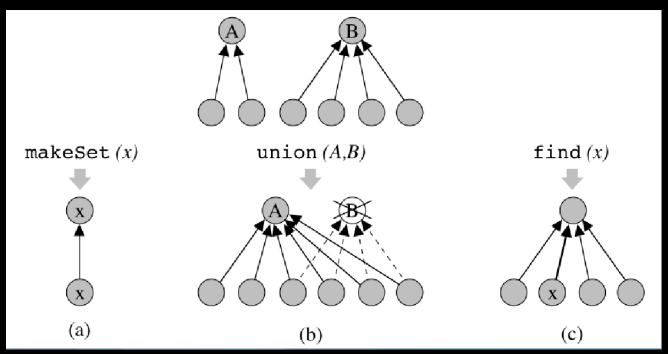
`makeSet`($elem\ e$) $T(n) = O(1)$
crea un nuovo albero, composto da due nodi: una radice ed un unico figlio (foglia). Memorizza e sia nella foglia dell'albero che come nome nella radice.

`union`($name\ a, name\ b$) $T(n) = O(n)$
considera l'albero A corrispondente all'insieme di nome a, e l'albero B corrispondente all'insieme di nome b. Sostituisce tutti i puntatori dalle foglie di B alla radice di B con puntatori alla radice di A. Cancella la vecchia radice di B.

`find`($elem\ e$) $\rightarrow name$ $T(n) = O(1)$
accede alla foglia x corrispondente all'elemento e. Da tale nodo segue il puntatore al padre, che è la radice dell'albero, e restituisce il nome memorizzato in tale radice.

→ COSTO + PARTICOLARE MA SEE + AVANTI

un esempio delle operazioni



QUANTO COSTA OGNUNO?

COSTO SINGOLA OPERAZIONE:

- MAKESET(x) = O(1) → CREI 1 e
- FIND(e) = O(1) → CAMBIO POINTER
- UNION(A,B) = O(m) con m = ELEM. DI B

$\Downarrow$

$O(m)$.

se li eseguiamo un tot arbitrario di volte?

- FIND e MAKESET RIMANGONO $O(1)$
- UNION PARTICOLARI SOLO LUNGHI:

ESEMPIO

```
union (n-1, n)
union (n-2, n-1)
union (n-3, n-2)
   ⋮
union (2, 3)
union (1, 2)
```

→ 1 CAMBIO PUNTATORE
→ 2 " "
⋮
→ n " "

TOT: $\Theta(n^2)$

SI PUÒ MIGLIORARE!

QUICK FIND: EURISTICA UNION BY SIZE

- IDEA:

  UNIONE UNION(A,B), CONTROLLO CHE SIZE(A) < SIZE(B).
  SE SI, UNION(B,A) MA ALLA FINE NOME RADICE DI B = A.

  così le union sono le più piccole possibili.

- REALIZZAZIONE
  - FIND, MAKESET UGUALI A PRIMA. → COSTO SINGOLO E TOT OPERAZIONI $O(1)$
  - UNION:

union($name\ a, name\ b$) $T_{am} = O(\log n)$
considera l'albero A corrispondente all'insieme di nome a, e l'albero B corrispondente all'insieme di nome b. Se size(A) $\geq$ size(B), muovi tutti i puntatori dalle foglie di B alla radice di A, e cancella la vecchia radice di B. Altrimenti (size(B) $>$ size(A)) memorizza nella radice di B il nome A, muovi tutti i puntatori dalle foglie di A alla radice di B, e cancella la vecchia radice di A. In entrambi i casi assegna al nuovo insieme la somma delle cardinalità dei due insiemi originali (size(A) + size(B)).

COSTO:
- CASO PEGGIORE → $O(n)$
- CASO AMMORTIZZATO → $O(\log n)$

N.B. COSTO AMMORTIZZATO : COSTO DI UNA OP. SINGOLA IN UN NUMERO ARBITRARIO DI OPERAZIONI

ANALISI COSTO AMMORTIZZATO

- NEL CASO PEGGIORE, $\Theta(n)$ = CASO UNION(A,B) $A = B = n/2$

- NELL'ALTRO, CONSIDERIAMO, DOPO n UNION, UN SINGOLO NODO:

① OSS. CHE IL NODO CHANGE PADRE $\leq O(\lg n)$

⇓

POICHÈ IL NODO SI TROVERÀ IN UN NUOVO INSIEME GRANDE $\geq 2 \cdot$ INSIEME VECCHIO DI NODO.

DOVE
- 1 INSERIRE OG
- 2 2° SET
- i -> 2^ SET -> DOPO n SCAMBI

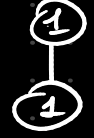
$O(\lg n)$

QUINDI $\times$ n VOLTE:

$O(n \lg n)$

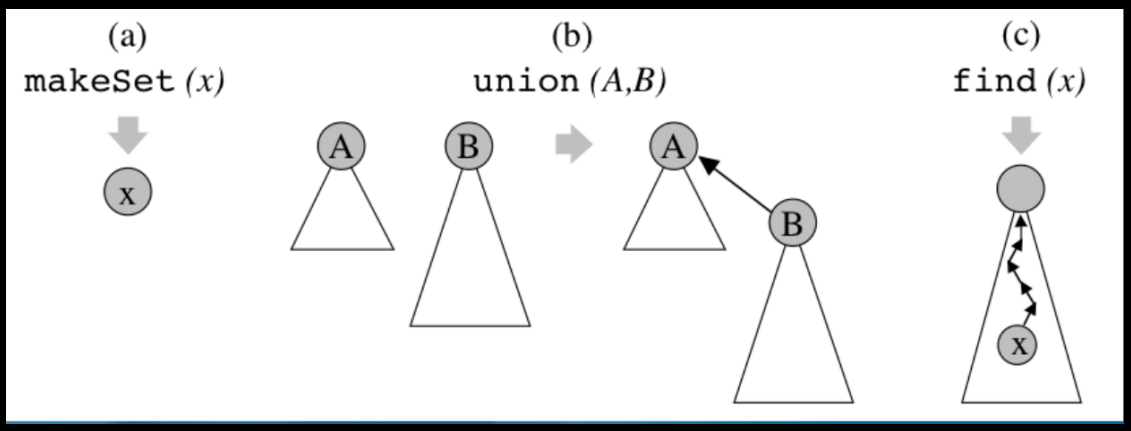
2° APPROCCIO: UNION BY QUICK UNION

= APPROCCIO CON QUICK FIND, MA CON:

- ELEMENTO = ALTRI ELEMENTI ESCLUSA RADICE -> # 

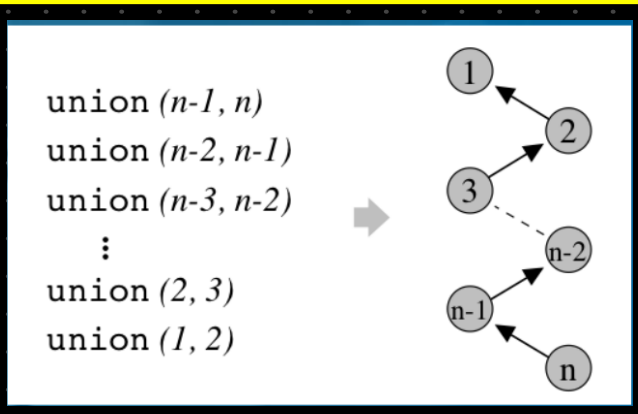
COMANDI:

- MAKESET(x) → = A PAG. PRIMA
- UNION(A,B) → B PUNTA AD A.
- FIND(x) → "SALGO" FINO A RADICE.



COSTO SINGOLA OPERAZIONE: O(1) | O(1) CAMBIO POINTER | O(h) → h = ALTEZZA ALBERO

ANCHE QUI, UN CASO PARTICOLARE DI UNION È INEFFICIENTE



⇒ COSTO

O(n) SINGOLO

m FIND:

$m \cdot O(n) = O(n \cdot m)$

anche qui, l'euristica union by size aiuta a migliorare il costo

UNION BY SIZE: UNION(A,B), CONFRONTO SIZE(A) e SIZE(B)

↳ NUM. NODI

- SE SIZE(A) < SIZE(B) → UNION(B, A) → DOPO CAMBIO NOME IN A

IL COSTO ?

LEMMA: ALBERO QUICKUNION CON n. NODI r E ALTEZZA h VALE CHE $r \geq 2^h$

QUINDI

COSTO AMMORTIZZATO

COSTO FIND → $O(\log n)$? ⇒ DATO CHE

n = N. NODI

h = ALT. ALBERO

$h \leq \log(n)$

QUINDI, COME X QUICK FIND ....

SE FACCIAMO m MAKESET,

n UNION

m FIND

⇒ $O(n + m \log n)$

X CAPIRE PERCHÉ LEMMA IS VALID

DIM

Lemma 9.2 *Un albero QuickUnion bilanciato in altezza con radice x ha almeno $2^{\text{rank}(x)}$ nodi.* N.B → RANK(A) = ALTEZZA ALBERO A.

Dimostrazione. Procediamo per induzione sul numero di operazioni effettuate. All'inizio, non ci sono alberi e quindi la base dell'induzione è banalmente verificata. Assumiamo che il lemma sia verificato prima di un'operazione, e dimostriamo che sarà valido anche dopo. Dato che le `find` non modificano alberi o rank, consideriamo solamente le operazioni `makeSet` e `union`.

Consideriamo prima un'operazione `makeSet`(x). Tale operazione non modifica alberi e rank preesistenti, ed introduce un albero di altezza 0, contenente solo il nodo x, con $\text{rank}(x) = 0$. Tale albero ha $2^{\text{rank}(x)} = 2^0 = 1$ nodo e pertanto verifica il lemma.

Consideriamo ora un'operazione `union`(A, B). Indichiamo con $\text{rank}(A)$ e $\text{rank}(B)$ il rank delle radici dei due alberi A e B prima della `union`(A, B), e con $\text{rank}(A \cup B)$ il rank della radice dell'albero risultante. Similmente, indichiamo con $|A|$ e $|B|$ il numero di nodi nei due alberi A e B prima della `union`(A, B), e con $|A \cup B|$ il numero di nodi nell'albero risultante. Osserviamo che avremo sempre $|A \cup B| = |A| + |B|$. Distinguiamo ora tre casi.

1. Se $\text{rank}(B) < \text{rank}(A)$, l'operazione `union` rende la radice dell'albero B figlia della radice dell'albero A. La radice dell'albero risultante avrà pertanto $\text{rank}(A \cup B) = \text{rank}(A)$, e l'albero risultante avrà $|A \cup B| = |A| +$ $|B|$ nodi. Utilizzando l'ipotesi induttiva sugli insiemi A e B, il numero di nodi nell'albero risultante è pertanto

$$|A \cup B| = |A| + |B| \geq 2^{\text{rank}(A)} + 2^{\text{rank}(B)} > 2^{\text{rank}(A)} = 2^{\text{rank}(A \cup B)}$$

(UGUALI)

Se $\text{rank}(A) < \text{rank}(B)$, l'operazione `union` rende la radice dell'albero A figlia della radice dell'albero B, e memorizza A come nome nella radice del nuovo albero. La radice dell'albero risultante avrà pertanto $\text{rank}(A \cup B) = \text{rank}(B)$, e l'albero risultante avrà $|A \cup B| = |A| + |B|$ nodi. Utilizzando l'ipotesi induttiva sugli insiemi A e B, il numero di nodi nell'albero risultante è pertanto

$$|A \cup B| = |A| + |B| \geq 2^{\text{rank}(A)} + 2^{\text{rank}(B)} > 2^{\text{rank}(B)} = 2^{\text{rank}(A \cup B)}$$

Se $\text{rank}(A) = \text{rank}(B)$, l'operazione `union` rende la radice dell'albero B figlia della radice dell'albero A, ed aggiorna il rank di A a $\text{rank}(A) + 1$.

La radice dell'albero risultante avrà pertanto $\text{rank}(A \cup B) = \text{rank}(A) + 1$, e l'albero risultante avrà $|A \cup B| = |A| + |B|$ nodi. Utilizzando l'ipotesi induttiva sugli insiemi A e B, il numero di nodi nell'albero risultante è pertanto

$$\begin{aligned} |A \cup B| = |A| + |B| &\geq 2^{\text{rank}(A)} + 2^{\text{rank}(B)} \geq \\ &\geq 2 \cdot 2^{\text{rank}(A)} = 2^{\text{rank}(A)+1} = 2^{\text{rank}(A \cup B)} \end{aligned}$$

In ognuno dei tre casi, il lemma sarà quindi verificato anche dopo l'operazione $\texttt{union}(A, B)$. □

Il seguente corollario è una conseguenza immediata del Lemma 9.2.