Algoritmo di Cole

Giuseppe Crisafulli

10-05-2011

Algoritmi Avanzati

a.a.2010/2011

Outline

Cosa abbiamo visto ieri

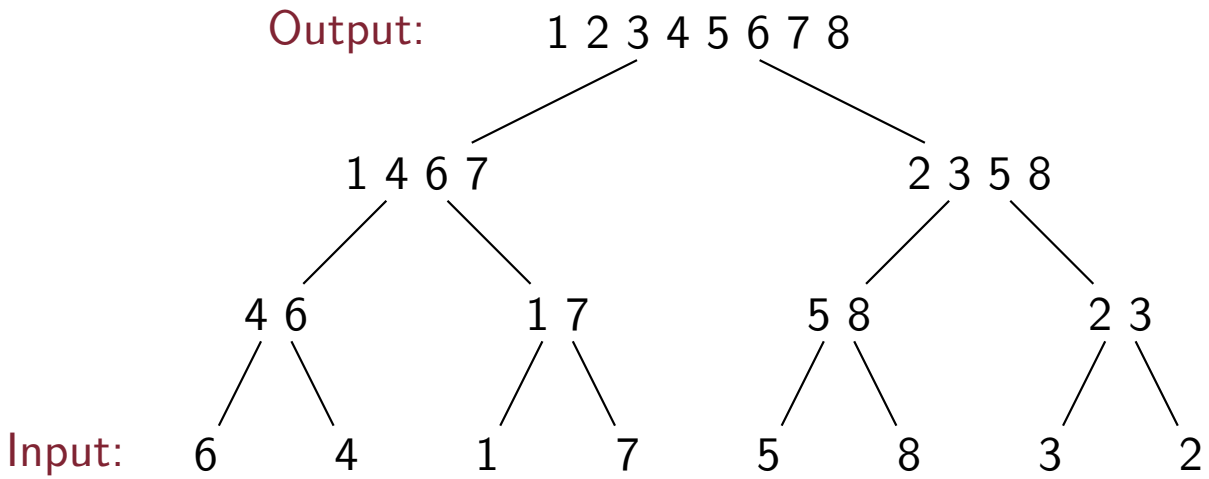
- Circuiti Comparatori
  - Tempo $O(\frac{1}{2}\log n(\log n + 1))$
- PRAM CREW
  - $\frac{n}{\log n}$ processori in tempo $O(\log^2 n)$

Vedremo oggi:

- PRAM CREW
  - n processori in tempo $O(\log n)$

Come ridurre il tempo di esecuzione

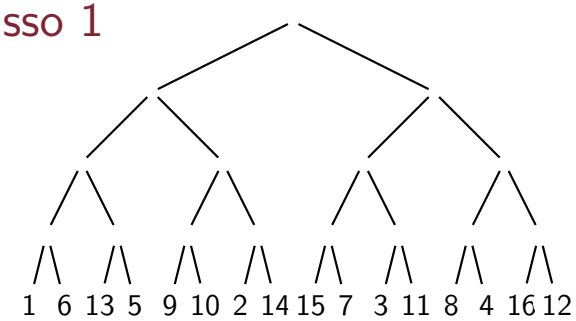
Per ridurre il tempo parallelo da $O(\log^2 n)$ a $O(\log n)$ è utile vedere l'algoritmo di ordinamento come un albero di ordinamento (Merge Tree)



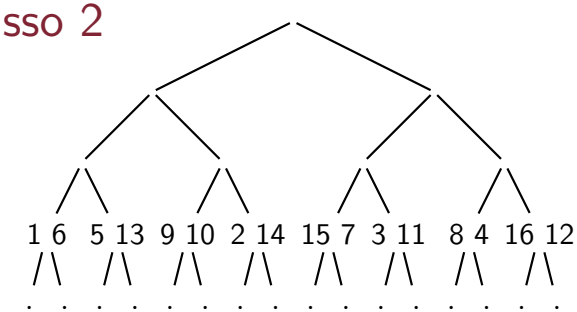
Idea

L'algoritmo procede per fasi, inizialmente solo le foglie contengono elementi, nei passi successivi per ogni nodo u un vettore $M_t(u)$ viene aggiornato.

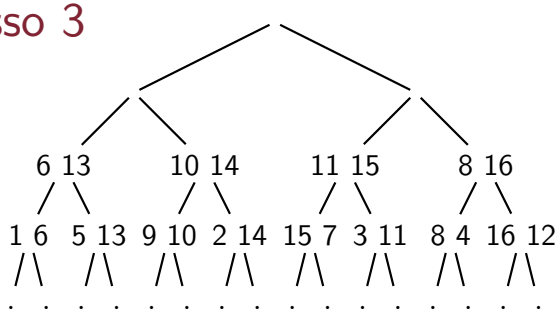
passo 1



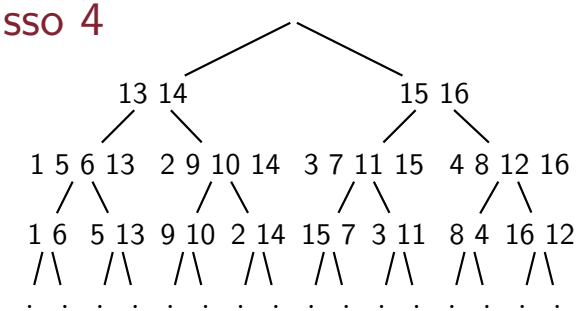
passo 2



passo 3



passo 4



Alcune definizioni

Predecessore-Successore

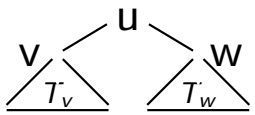
Dato un vettore A, un elemento $a \in$A e un elemento $x \neq a$ si dice che:

- a è predecessore in A di x, se a è il massimo in A più piccolo di x.
- a è successore in A di x se a è il minimo in A più grande di x.

3-cover

Dati due vettori C e D, si dice che C ha copertura di grado 3 (3-cover) su D se per ogni due elementi adiacenti e_1 e e_2 in C, l'intervallo $[e_1, e_2)$ contiene al più 3 elementi contenuti in D.

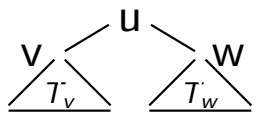
Sono necessarie alcune strutture aggiuntive:



- Vettore O (Origin), indica con un etichetta (L o R) da dove proviene un elemento. Per un elemento $e \in M_t(u)$, $O_t(u)[e] = L$ (R) se e proviene dal sottoalbero T_v (T_w).
- Vettore E (Extract), $E_{t+1}(u)$ contiene gli elementi di $M_t(u)$ che andranno in $M_{t+1}(padre(u))$.
- Vettore B_t (Before), per ogni elemento $e \in M_t(u)$ proveniente da T_v , punta al predecessore in $M_t(u)$ di e proveniente dal sottoalbero T_w.
- Vettore A_t (After), per ogni elemento $e \in M_t(u)$ proveniente da T_v, punta al successore in $M_t(u)$ di e proveniente dal sottoalbero T_w.

NOTA: Le definizioni di A_t e B_t si applicano simmetricamente se il nodo e proviene da T_w.

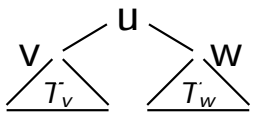
Altre strutture sono:



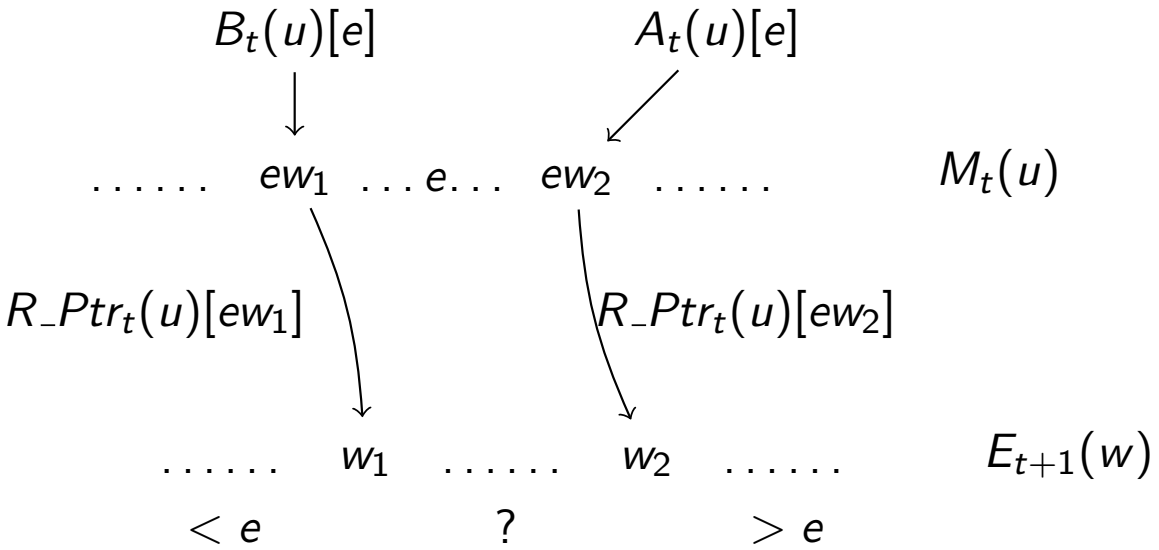
- Vettore *R_Ptr*, per ogni elemento $e \in M_t(u)$, se e proviene dal sottoalbero T_w, punta al predecessore di e in $E_{t+1}(v)$.
- Vettore *L_Ptr*, per ogni elemento $e \in M_t(u)$, se e proviene dal sottoalbero T_v, punta al predecessore di e in $E_{t+1}(w)$.
- Vettore *P_Ptr*, per ogni elemento $ev \in E_{t+1}(v)$ punta al predecessore di ev in $M_t(u)$.

Merge 1.1

I passi dell'algoritmo sono 4:

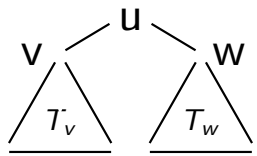


1.1 - Per ogni $e \in M_t(u)$ che viene dal sottoalbero T_v calcolare il suo predecessore in $E_{t+1}(w)$ e puntarlo in $R\_Ptr_t(u)[e]$.
Il calcolo viene effettuato dal processore legato all'elemento e.

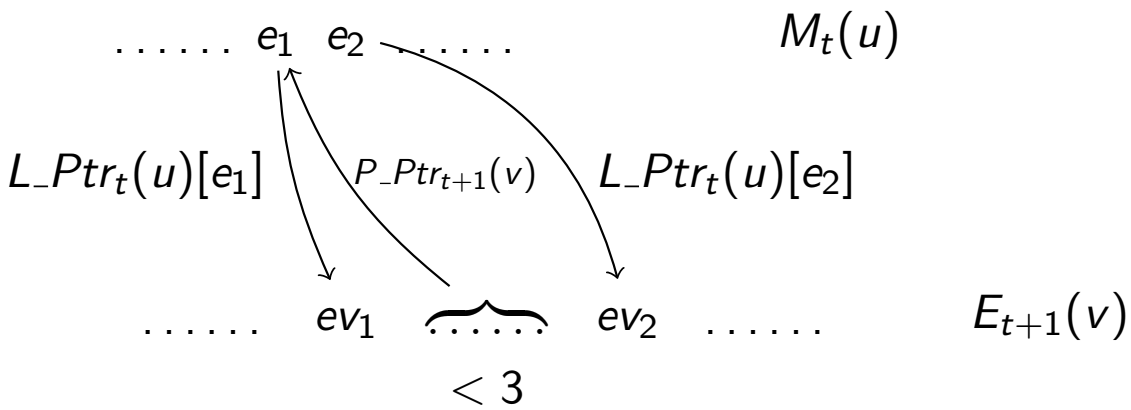


Nota: Questo passo prevede un numero costante di confronti se $E_t(u)$ 3-cover $E_{t+1}(u)$.

Merge 1.2

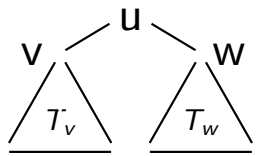


1.2 - Per ogni $ev \in E_{t+1}(v)$ calcolo il predecessore in $M_t(u)$, il puntatore a questo elemento viene salvato in $P\_Ptr_{t+1}(v)$.
Questo calcolo viene effettuato dai processori legati agli elementi di $M_t(u)$.

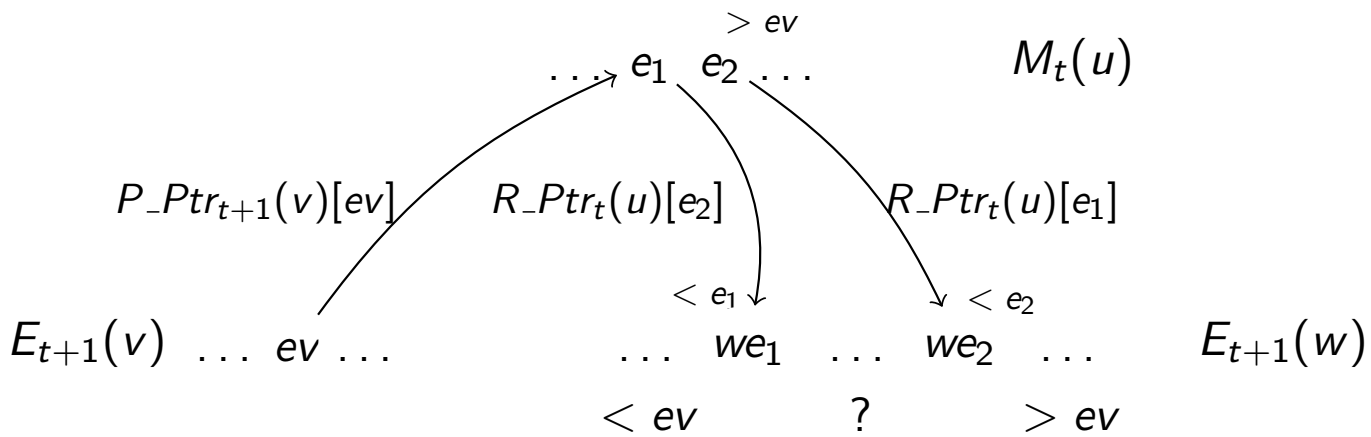


Nota: La proprietà di 3-cover di $E_t(u)$ su $E_{t+1}(u)$ assicura un numero di assegnamenti costante.

Merge 2.1



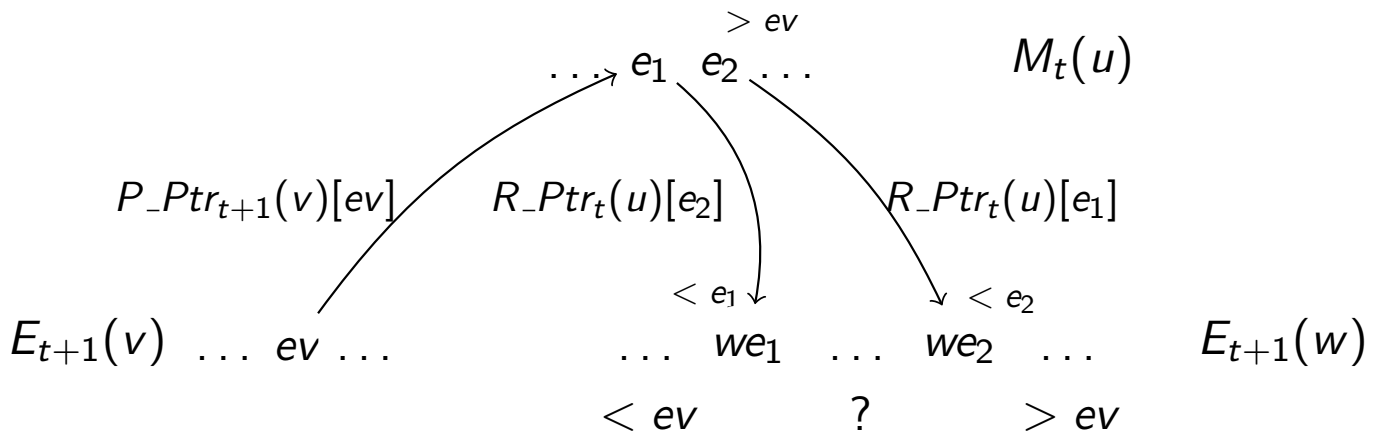
2.1 Fondiamo le sequenze $E_{t+1}(v)$ e $E_{t+1}(w)$ in $M_{t+1}(u)$. Per ogni $ev \in E_{t+1}(v)$ determinare il predecessore in $E_{t+1}(w)$ in modo da sapere dove collocare ev in $M_{t+1}(u)$.



Nota: La proprietà di 3-cover di $E_t(u)$ su $E_{t+1}(u)$ assicura un numero di confronti costante.

Merge 2.2

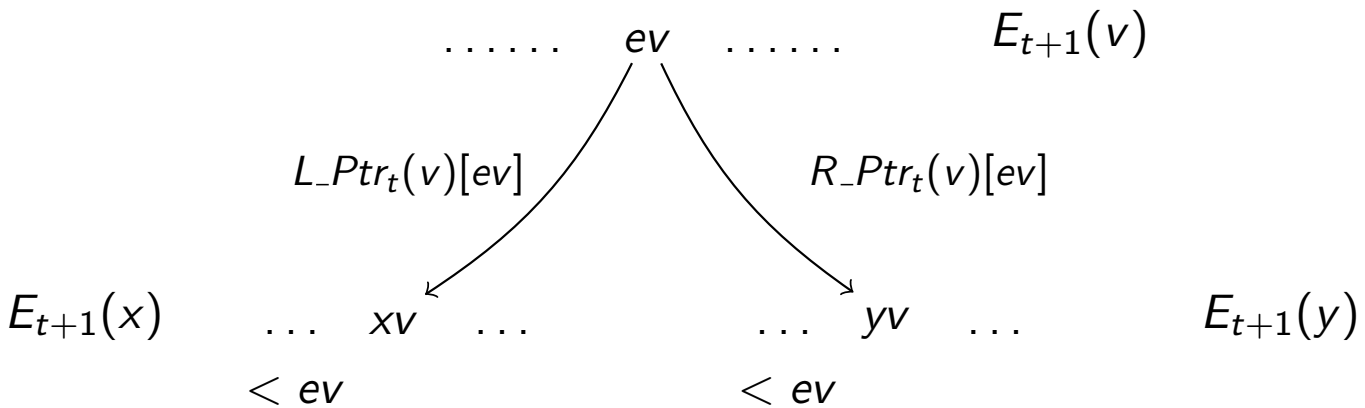
2.2 - Se $M_{t+1}(u)$ non ha raggiunto la massima dimensione è necessario aggiornare le strutture dati per il prossimo passo.
In realtà solo A_{t+1}, B_{t+1}, R_Ptr e L_Ptr non sono state già aggiornate nel corso dei passi precedenti.
Vediamo come costruire A_{t+1}, ricordando che:
Def: $A_{t+1}(u)[ev]$ esiste se $O_{t+1}(u)[ev] = L$ e punta al successore in $M_{t+1}(u)$ di ev proveniente dal sottoalbero T_w.



Merge 2.2 (continua)

Ogni elemento $ev \in E_{t+1}(v)$ è anche un elemento di $M_t(v)$ quindi si possono usare le informazioni note sul nodo v per computare $L\_Ptr_{t+1}(v)[ev]$.

Def. *L_Ptr*: per ogni elemento $e \in M_{t+1}(u)$, se e proviene dal sottoalbero T_v, punta al predecessore di e in $E_{t+2}(v)$.



$$L\_Ptr_{t+1}(u)[ev] = L\_Ptr_t(v)[ev] + R\_Ptr_t(v)[ev]$$

Dimensione $M_t(u)$

Inizialmente abbiamo assunto che $|M_{t+1}(u)| = 2 \cdot |M_t(u)|$ finché non si raggiunge la dimensione massima, in realtà per assicurare la proprietà di 3-cover è necessario precisare che, per un nodo u ad altezza h:

- $|M_t(u)|$ raddoppia fino al passo t' in cui raggiunge la dimensione massima per la prima volta;
- $E_{t''}(u)$, per $2h + 1 \leq t'' \leq 3h - 2$, preleva un elemento ogni 4 da $M_{t''-1}(u)$;
- $E_{3h-1}(u)$ preleva ogni elemento pari in $M_{3h-2}(u)$;
- $E_{3h}(u)$ comprende ogni elemento in $M_{3h-1}(u)$.

Lemma 2.1: Se u è un nodo interno di altezza h, $M_{3h}(u)$ contiene tutti gli elementi del sottoalbero radicato in u.
Dim: Induzione su h . . . .

Dimensione $M_t(u)$ (continua)

Lemma 2.2: Se u è un nodo interno di altezza h:

1. Se u non è il nodo più a destra allora $|M_{2h+1}(u)| = 2$ e $|M_t(u)|$ raddoppia per $2h+1 < t \leq 3h$.
2. Se u è il nodo più a destra allora esiste una prima iterazione $k \geq 2h+1$ per cui $|M_k(u)| > 0$ ed uguale a 1 o 2, e per le fasi $k < t \leq 3h$, $|M_t(u)|$ vale $2|M_{t-1}(u)|$ o $2|M_{t-1}(u)|+1$.

Dim: Si procede per induzione su h.
2.1: Passo base: $h = 1 \Rightarrow M_3 = 2$.
Passo induttivo:
Per $t = 2h+1$ sappiamo $|M_t(u)| = \lfloor \frac{|M_{t-1}(v)|}{4} \rfloor + \lfloor \frac{|M_{t-1}(w)|}{4} \rfloor$
Ma v ha altezza $h' = h-1$:
$M_{2h'+1}(v) = 2 = M_{2(h-1)+1}(v) = M_{2h-1}(v) = M_{t-2}(v)$
$\Rightarrow M_{t-1}(v) = M_{t-1}(w) = 4 \Rightarrow M_{2h+1}(u) = 2$

□

Dimensione $M_t(u)$ (continua)

2. Se u è il nodo più a destra allora esiste una prima iterazione $k \geq 2h+1$ per cui $|M_k(u)| > 0$ ed uguale a 1 o 2, e per le fasi $k < t \leq 3h$, $|M_t(u)|$ vale $2|M_{t-1}(u)|$ o $2|M_{t-1}(u)|+1$.

Dim.

Per $t = k$ si dimostra che $|M_k(u)| \leq 2$ come in 2.1.

Consideriamo $\max\{k, 2h+1\} < t \leq 3h-2$ e distinguiamo due casi:

1 - Il figlio destro w esiste:

$$|M_t(u)| = \frac{|M_{t-1}(v)|}{4} + \lfloor \frac{|M_{t-1}(w)|}{4} \rfloor$$

Per $t-1 \geq 2h-1$ per ipotesi induttiva:

$$|M_{t-1}(v)| = 4|M_{t-3}(v)| \quad \text{e} \quad \lfloor \frac{|M_{t-1}(w)|}{4} \rfloor = |M_{t-3}(w)|$$
$$|M_t(u)| \leq |M_{t-3}(v)| + \max\{1, |M_{t-3}(w)|\}$$

Allo stesso modo:

$$|M_{t-1}(u)| = \frac{|M_{t-3}(v)|}{2} + \lfloor \frac{|M_{t-3}(w)|}{2} \rfloor$$
$$\Rightarrow |M_t(u)| = 2|M_{t-1}(u)| \quad \text{o} \quad 2|M_{t-1}(u)|+1.$$

Dimensione $M_t(u)$ (continua)

2. Se u è il nodo più a destra allora esiste una prima iterazione $k \geq 2h+1$ per cui $|M_k(u)| > 0$ ed uguale a 1 o 2, e per le fasi $k < t \leq 3h$, $|M_t(u)|$ vale $2|M_{t-1}(u)|$ o $2|M_{t-1}(u)|+1$.

2 - Il figlio destro w non esiste:

$$|M_t(u)| = \frac{|M_{t-1}(v)|}{4} + 0$$

Per $t-1 \geq 2h-1$ per ipotesi induttiva:

$$|M_{t-3}(v)| = 0 \quad \text{oppure} \quad \lfloor \frac{|M_{t-1}(v)|}{4} \rfloor = |M_{t-3}(v)|$$
$$|M_t(u)| \leq |M_{t-3}(v)|$$

Allo stesso modo:

$$|M_{t-1}(u)| = \lfloor \frac{|M_{t-3}(v)|}{2} \rfloor$$
$$\Rightarrow |M_t(u)| = 2|M_{t-1}(u)| \quad \text{o} \quad 2|M_{t-1}(u)|+1.$$

□

Lemma 2.3

Lemma 2.3: Scelto un $k \geq 1$, ogni intervallo di adiacenti ampio k in $E_t(u)$ contiene al più $2k+1$ elementi di $E_{t+1}(u)$.

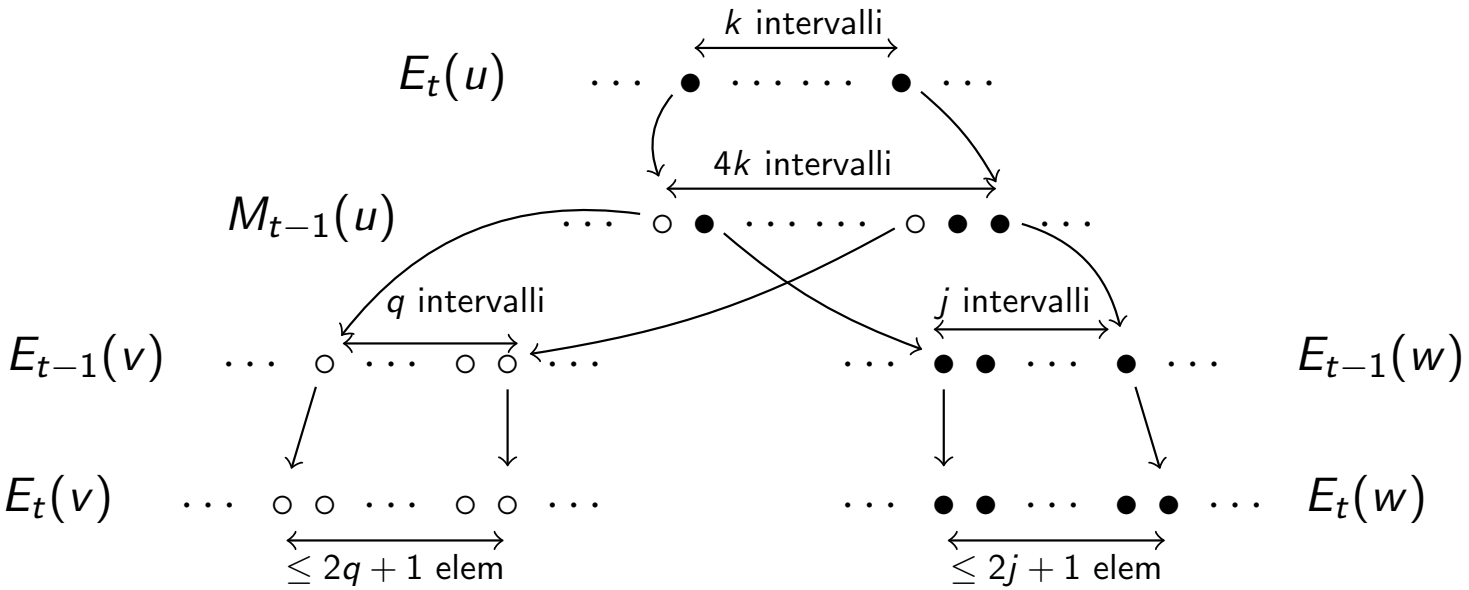
Dim. Procediamo per induzione su t.

Passo base: Se $E_t(u) = \emptyset$ allora $|E_{t+1}(u)| \leq 1$.

Passo Induttivo:

$q+j = 4k+1 \Rightarrow |M_t(u)| \cong 2q+1+2j+1 = 2(q+j)+2 = 8k+4$

$| \Rightarrow E_{t+1}(u)| = \lfloor M_t(u)/4 \rfloor = 2k+1$ □



Numero di operazioni per processore

Lemma 2.4: Indichiamo con $L(u)$ il numero di elementi contenuti nel sottoalbero radicato in u. Al più $\frac{15}{4} \cdot L(u)$ confronti vengono effettuati durante la computazione dei vettori $M_t(u)$.

Dim:Sia t' la fase in cui $M_{t'}(u)$ acquisisce la dimensione s, il numero di confronti in questa fase sono:
Passo 1.1: 2 per ognuno degli $\frac{s}{2}$ elementi di $M_{t'-1}(u) \Rightarrow 2 \cdot \lfloor \frac{s}{2} \rfloor$
Passo 2.1: 2 per ogni elemento in $M_{t'}(u) \Rightarrow 2s$
Totale: $3s$ confronti nella fase t'.

La somma su tutte le iterazioni $t \quad \Rightarrow \quad 3 \cdot \sum_{i \geq 0} \frac{L(u)}{2^i} \leq 6L(u)$

Ma quando si raggiunge la dimensione massima, per le due fasi successive viene fatto un solo confronto per elemento.

$\Rightarrow \quad \frac{15}{4} L(u)$

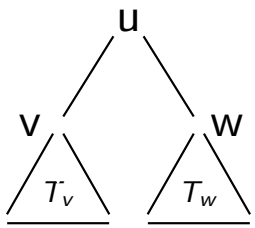
□

Numero di iterazioni

Lemma 2.5: L'algoritmo termina in $3 \log n$ iterazioni.

Se $M_t(v)$ raggiunge la massima dimensione al passo $t = t'$ allora $M_t(u)$ raggiunge la sua massima dimensione al passo $t' + 3$. Ma l'albero contiene $\log n$ livelli.

□



Numero di processori

Def: Al passo t i nodi z per cui $M_{t'}(z)$ ha raggiunto la massima dimensione ($t' \leq t$) si dicono inattivi, tutti gli altri si dicono attivi.

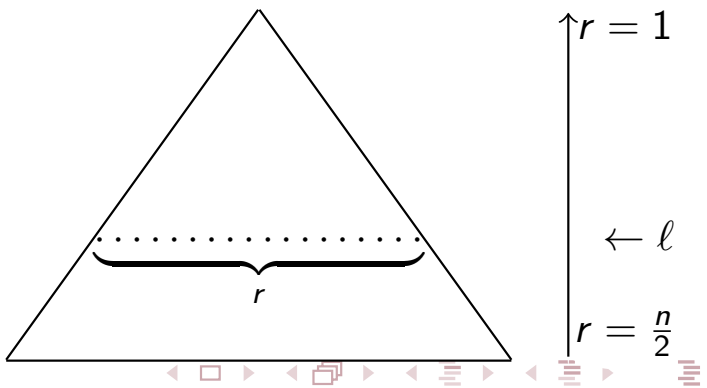
Lemma 2.6: L'algoritmo usa $\frac{12}{7}n + (4n + 4\lceil \log n \rceil)$ processori.

Dim: Consideriamo la fase t', se $|M_{t'+1}(u)|$ ha dimensione s i processori legati a $M_{t'}(u)$,$E_{t'+1}(v)$ e $E_{t'+1}(w)$ sono $\lfloor 3 \cdot \frac{s}{2} \rfloor + 4$.
I nodi che all'iterazione t' sono inattivi non hanno bisogno di processori.
Al livello più profondo ℓ, che contiene r nodi, usiamo $\lfloor 3 \cdot \frac{n}{2} \rfloor + 4r$ processori.

Sommando su tutti i nodi attivi:

$$\frac{12}{7}n + (4n + 4\lceil \log n \rceil)$$

□



Teorema 2.1

Teorema 2.1: In una PRAM CREW l'algoritmo di sorting parallelo può lavorare in tempo $O(\log n)$ usando $5\frac{5}{7}n + 4\lceil \log n \rceil$ che eseguono al più $\frac{15}{4}n$ confronti.

Segue dai lemma 2.4,2.5 e 2.6.

Spazio utilizzato

Un implementazione efficiente può utilizzare spazio $O(n)$.

Un implementazione semplice usa $O(n \log n)$ spazio. Questa usa spazio $O(L(u))$ per ogni nodo u.

Tuttavia molto spazio può essere riutilizzato, infatti in un passo generico $t+1$ sono necessarie solo le strutture per il passo t e $t+1$.

Domande??