ESERCIZIO DI ASD DEL 10 NOVEMBRE 2008

Intervalli

Algoritmo Intervalli Disgiunti. Condizione necessaria e sufficiente affinché tra n intervalli della forma $[l_i, u_i]$ ce ne siano almeno due disgiunti è che quello che termina più a sinistra termini prima dell'inizio di quello che inizia più a destra. Per esempio se consideriamo $S = \{[1,5],[2,8],[4,6]\}$ abbiamo che l'intervallo che termina più a sinistra è $[1,5]$ e l'intervallo che inizia più a destra è $[4,6]$. Visto che 5 è maggiore o uguale di 4 non ci sono intervalli disgiunti. Infatti $[1,5]$ e $[2,8]$ non sono disgiunti, $[1,5]$ e $[4,6]$ non sono disgiunti, $[2,8]$ e $[4,6]$ non sono disgiunti. Se invece consideriamo $S = \{[1,5],[2,8],[7,9]\}$ abbiamo che l'intervallo che termina più a sinistra è $[1,5]$ e l'intervallo che inizia più a destra è $[7,9]$. Visto che 5 è minore di 7 abbiamo che ci sono due intervalli disgiunti. Infatti $[1,5]$ e $[7,9]$ sono disgiunti.

Descriviamo quindi un algoritmo che si basa su questa condizione necessaria e sufficiente. Supponiamo che l'insieme S sia memorizzato in un vettore V di lunghezza n i cui elementi hanno due campi: $V[i].lower = l_i$ è l'estremo di sinistra dell'i-esimo intervallo e $V[i].upper = u_i$ è l'estremo di destra. Eseguremo un unico ciclo for durante il quale determineremo il minimo degli estremi di destra ed il massimo degli estremi di sinistra. Controlleremo che il minimo degli estremi di destra sia minore del massimo degli estremi di sinistra.

Algorithm 1 Due_Intervalli_Disgiunti(V)

```
1: min_u ← V[1].upper
2: max_l ← V[1].lower
3: for i ← 2 to length[V] do
4:    if V[i].upper < min_u then
5:       min_u ← V[i].upper
6:    end if
7:    if V[i].lower > max_l then
8:       max_l ← V[i].lower
9:    end if
10: end for
11: if min_u < max_l then
12:    return TRUE
13: else
14:    return FALSE
15: end if
```

Date: 10 Novembre 2008.

Correttezza. Dimostriamo che la condizione su cui abbiamo basato il nostro algoritmo è corretta.

Lemma 1. *Sia $S = \{[l_1, u_1], [l_2, u_2], \ldots, [l_n, u_n]\}$ un insieme di n intervalli chiusi (estremi inclusi) con $l_i \leq u_i$. Esistono $i, j \leq n$ tali che $[l_i, u_i] \cap [l_j, u_j] = \emptyset$ se e soltanto se il minimo dell'insieme $U = \{u_1, u_2, \ldots, u_n\}$ è minore del massimo dell'insieme $L = \{l_1, l_2, \ldots, l_n\}$.*

Dimostrazione. $\Rightarrow$)

Hp) Esistono $i, j \leq n$ tali che $[l_i, u_i] \cap [l_j, u_j] = \emptyset$.

Ts) Il minimo dell'insieme $U = \{u_1, u_2, \ldots, u_n\}$ è minore del massimo dell'insieme $L = \{l_1, l_2, \ldots, l_n\}$.

Visto che $[l_i, u_i] \cap [l_j, u_j] = \emptyset$ deve essere $u_i < l_j$. Sia min_u il minimo dell'insieme U e max_l il massimo dell'insieme L. Abbiamo che $min\_u \leq u_i < l_j \leq max\_l$, da cui otteniamo $min\_u < max\_l$.

$\Leftarrow$)

Hp) Il minimo dell'insieme $U = \{u_1, u_2, \ldots, u_n\}$ è minore del massimo dell'insieme $L = \{l_1, l_2, \ldots, l_n\}$.

Ts) Esistono $i, j \leq n$ tali che $[l_i, u_i] \cap [l_j, u_j] = \emptyset$.

Sia u_i il minimo dell'insieme U ed l_j il massimo dell'insieme L. Per ipotesi abbiamo che $u_i < l_j$. Quindi visto che $l_i \leq u_i$ abbiamo che sicuramente u_i ed l_j sono estremi di due intervalli distinti. Se consideriamo i due intervalli $[l_i, u_i]$ e $[l_j, u_j]$ abbiamo che visto che $u_i < l_j$ vale che $[l_i, u_i] \cap [l_j, u_j] = \emptyset$ □

Ora possiamo dimostrare la correttezza della procedura proposta. Come prima cosa dimostriamo l'invariante per il ciclo for.

Invariante 1. *All'inizio della i-esima iterazione del ciclo for abbiamo che:*

- *per ogni j tale che $1 \leq j < i$ vale che $V[j].upper \geq min\_u$;*
- *per ogni j tale che $1 \leq j < i$ vale che $V[j].lower \leq max\_l$.*

Dimostrazione. Procediamo per induzione su i.

BASE) $i = 2$. All'inizio dell'iterazione in cui i vale 2 abbiamo che $min\_u = V[1].upper$ e $max\_l = V[1].lower$. Quindi vale la tesi.

PASSO)

HpInd) All'inizio dell'iterazione in cui i vale $k-1$ abbiamo che per ogni j tale che $1 \leq j < k-1$ vale che $V[j].upper \geq min\_u$ e $V[j].lower \leq max\_l$.

Ts) All'inizio dell'iterazione in cui i vale k abbiamo che per ogni j tale che $1 \leq j < k$ vale che $V[j].upper \geq min\_u$ e $V[j].lower \leq max\_l$.

Dobbiamo esaminare ciò che accade durante l'iterazione del ciclo for in cui i vale $k-1$. Durante tale iterazione vengono esaminati $V[k-1].upper$ e $V[k-1].lower$. Se $V[k-1].upper \geq min\_u$, allora min_u non viene modificato. In tal caso abbiamo dall'ipotesi induttiva che per ogni j tale che $1 \leq j < k-1$ vale che $V[j].upper \geq min\_u$, inoltre vale che $V[k-1].upper \geq min\_u$, quindi otteniamo che per ogni j tale che $1 \leq j < k$ vale che $V[j].upper \geq min\_u$. Se invece $V[k-1].upper = a < min\_u = b$, allora a min_u viene assegnato valore a. Avevamo che per ipotesi induttiva valeva $V[j].upper \geq b$ per ogni j tale che $1 \leq j < k-1$, quindi visto che $a < b$ a maggior ragione abbiamo $V[j].upper \geq a$ per ogni j tale che $1 \leq j < k-1$. Inotre abbiamo $V[k-1].upper = a$ quindi possiamo concludere che vale $V[j].upper \geq a = min\_u$ per ogni j tale che $1 \leq j < k$.

Analogamente si dimostra che vale $V[j].lower \leq max\_l$ per ogni $1 \leq j < k$. □

Theorem 1. DUE_INTERVALLI_DISGIUNTI(V) *termina sempre e restituisce* $TRUE$ *se e soltanto se tra gli intervalli* $\{[V[1].lower, V[1].upper], [V[2].lower, V[2].upper], \ldots, [V[length[V]].lower, V[length[V]].upper]\}$ *ce ne sono due disgiunti.*

Dimostrazione. DUE_INTERVALLI_DISGIUNTI(V) è costituito principalmente da un ciclo for che termina sicuramente.

Il ciclo for termina quando la variabile i assume valore $length[V] + 1$. Dall'invariante abbiamo che al termine del ciclo for vale che per ogni j compreso tra 1 e $length[V]$ vale che $V[j].upper \geq min\_u$ e $V[j].lower \leq max\_l$.

Dal Lemma 1 abbiamo che tra gli intervalli dell'insieme $\{[V[1].lower, V[1].upper], [V[2].lower, V[2].upper], \ldots, [V[length[V]].lower, V[length[V]].upper]\}$ ce ne sono due disgiunti se e soltanto se $min\_u < max\_l$ e l'algoritmo termina restituendo $TRUE$ solo in questo caso. □

Complessità. Sia $n = length[V]$.

La procedura DUE_INTERVALLI_DISGIUNTI esegue un ciclo for di lunghezza n. Quindi la complessità della procedura è $\Theta(n)$.

Algoritmo Intervalli Non Disgiunti. Condizione necessaria e sufficiente affinché tra n intervalli della forma $[l_i, u_i]$ ce ne siano almeno due non disgiunti è che ci sia un intervallo che termina dopo (o contemporaneamente con) l'inizio dell'intervallo successivo. Per esempio se consideriamo $S = \{[1, 5], [6, 8], [10, 11]\}$ abbiamo che ogni intervallo termina prima dell'inizio del successivo. Quindi non ci sono due intervalli non disgiunti. Se invece consideriamo $S = \{[1, 5], [4, 8], [10, 11]\}$ abbiamo che l'intervallo $[1, 5]$ termina in 5, mentre l'intervallo successivo è $[4, 8]$ che inizia in 4. Visto che $5 \geq 4$ abbiamo che i due intervalli sono non disgiunti. Va osservato che per parlare di intervallo successivo ad un dato intervallo abbiamo considerato gli intervalli ordinati rispetto agli estremi di sinistra.

Descriviamo quindi un algoritmo che si basa su questa condizione necessaria e sufficiente. Come prima supponiamo che l'insieme S sia memorizzato in un vettore V di lunghezza n i cui elementi hanno due campi: $V[i].lower = l_i$ è l'estremo di sinistra dell'i-esimo intervallo e $V[i].upper = u_i$ è l'estremo di destra. Ordineremo il vettore V rispetto ai valori contenuti nei campi $lower$. Poi eseguiremo una scansione e controlleremo se $V[i].upper \geq V[i + 1].lower$. Se questa condizione è soddisfatta abbiamo trovato due intervalli non disgiunti. Se lungo tutta la lunghezza del vettore la condizione non è mai soddisfatta possiamo concludere che nel vettore non ci sono due intervalli non disgiunti.

Algorithm 2 DUE_INTERVALLI_NON_DISGIUNTI(V)

```
1: MERGESORT(V, lower)
2: for i ← 1 to length[V] − 1 do
3:    if V[i].upper ≥ V[i + 1].lower then
4:       return TRUE
5:    end if
6: end for
7: return FALSE
```

Anche se non è richiesto dall'esercizio dimostriamo parzialmente la correttezza della soluzione proposta dimostrando la correttezza della condizione necessaria e sufficiente.

Lemma 2. *Sia Sia* $S = \{[l_1, u_1], [l_2, u_2], \ldots, [l_n, u_n]\}$ *un insieme di* n *intervalli chiusi (estremi inclusi) con* $l_i \leq u_i$ *e tale che* $l_1 \leq l_2 \leq \cdots \leq l_n$*, ovvero gli estremi di sinistra sono ordinati. Esistono* $i, j \leq n$ *tali che* $[l_i, u_i] \cap [l_j, u_j] \neq \emptyset$ *se e soltanto se esiste* k *tale che* $u_k \geq l_{k+1}$*.*

Dimostrazione. Assumiamo che gli estremi di sinistra degli intervalli siano tutti distinti (se non lo sono il lemma è immediato).
⇒)

Hp) Esistono $i, j \leq n$ tali che $[l_i, u_i] \cap [l_j, u_j] \neq \emptyset$.

Ts) Esiste k tale che $u_k \geq l_{k+1}$.

Non è restrittivo supporre che l_i venga prima di l_j nell'ordinamento. Quindi abbiamo che $l_i < l_{i+1} \leq l_j$. Visto che i due intervalli si intersecano e $l_i < l_j$ deve essere $u_i \geq l_j$. Ma visto che $l_{i+1} \leq l_j$ e $u_i \geq l_j$ otteniamo che $u_i \geq l_{i+1}$. Quindi vale la tesi con $k = i$.
⇐)

Hp) Esiste k tale che $u_k \geq l_{k+1}$.

Ts) Esistono $i, j \leq n$ tali che $[l_i, u_i] \cap [l_j, u_j] \neq \emptyset$.

Consideriamo gli intervalli $[l_k, u_k]$ e $[l_{k+1}, u_{k+1}]$. Visto che $l_k < l_{k+1}$ e $u_k \geq l_{k+1}$ abbiamo che i due intervalli si intersecano. Quindi vale la tesi con $i = k$ e $j = k + 1$. □

In questo caso la complessità della procedura è data dalla complessità dell'algoritmo di ordinamento ed è quindi $\Theta(n \log n)$, dove $n = length[V]$.