

OPERAZIONI SU INSIEMI

Dato un insieme S ed un elemento k possiamo svolgere operazioni di

RICERCA (S, k), CANCELLAZIONE (S, k) = $S \setminus \{k\}$, INSERIMENTO (S, k) = $S \cup \{k\}$

Data una coppia (S, \leq) possiamo valutare

$\text{MIN}(S)$, $\text{MAX}(S)$, SUCCESSORE (S, k), PREDECESSORE (S, k)

L'efficienza di queste operazioni puo' dipendere dal tipo di struttura adottato:

Se adottiamo un array non ordinato la RICERCA ha un costo $T_p = O(n)$, se adottiamo l'array ordinato esistono diversi algoritmi che hanno diversa efficienza.

RICERCA BINARIA

L'algoritmo di ricerca binaria considera **Solo** array ordinati, quindi vale sempre $\forall i, j \in \{1, \dots, |S|\}, i \leq j \Rightarrow S[i] \leq S[j]$.

Si ricordi che le relazioni d'ordine sono transitive.

L'algoritmo di ricerca binaria adotta un'esecuzione divide et impera, in particolare analizziamo un solo sottinsieme alla volta.

Se presa una posizione i del nostro array, se $A[i] \neq K$ dobbiamo cercare l'elemento alla sua sinistra (indice minore di i), vale l'analogo. Quando la dimensione della sottosequenza considerata è 1 allora l'elemento è quello ricercato oppure non è presente.

Se scegliersero come indice di partizione sempre il primo, l'algoritmo non è molto efficiente escludendo un solo elemento alla volta (a meno che non si esca al primo confronto).

Per maximizzare l'efficienza prendiamo sempre l'elemento di mezzo $\left\lfloor \frac{|S|}{2} \right\rfloor$ così da escludere ogni volta metà array.

Sotto queste ipotesi vediamo l'algoritmo

RB(S, k, i, j)

IF $i \leq j$

$$q = \left\lfloor \frac{i+j}{2} \right\rfloor$$

IF $A[q] = K$ THEN

POS = Q

ELSE

IF $A[q] < K$ THEN

Reso l'array, l'elemento e gli estremi, restituisce la posizione prevista

$\text{POS} = \text{RB}(S, k, q+1, j)$

ELSE

$\text{POS} = \text{RB}(S, k, i, q-1)$

ELSE

$\text{POS} = j$

RETURN POS

Il costo dell'algoritmo e' (nel caso peggiore in particolare)

$$T(m) = \begin{cases} \Theta(1) & m=0 \\ \Theta(1) + T\left(\frac{m}{2}-1\right) & m>1 \end{cases}$$

$\frac{m}{2}-1$ perche' un elemento viene scartato

L'albero di ricerca e' degenero' (abbiamo solo una chiomata ricorsiva)

Ogni livello ha un contributo locale $\Theta(1)$, l'altezza e' $\log_2 m$

quindi il costo totale e' $O(\log_2 m)$

OSS

Analogamente gli algoritmi di ricerca del massimo, minimo etc.

derivano tutti dal tempo dell'algoritmo di ricerca

ALGORITMI DI MODIFICA DELL'INSIEME

Studiamo il caso dell'algoritmo di **INSERIMENTO** di un elemento in un insieme. Questo prevede una ricerca previa della posizione prevista nel nostro insieme ordinato.

In posizione prevista j potrebbe esserci un elemento diverso dal nostro k e quindi dovremo spostare tutti gli elementi in posizione $i \geq j$, aumentando il costo di $O(n)$.

Questo problema non si pone per array non ordinati che permettono l'inserimento in coda (evitando lo shift).

LISTA

L'è una lista se vale una delle seguenti affermazioni:

- $L \neq \emptyset$
- $L = \{t\} \cup L'$ dove t è un elemento e L' è una lista

La lista viene quindi definita ricorsivamente scomponendo la struttura in **TESTA** (t) e **CODA** (L')

LISTA ORDINATA

L è una lista ORDINATA se vale una delle seguenti affermazioni:

- $L \neq \emptyset$
- $L = \{t\} \cup L' \wedge t \leq x \forall x \in L'$

Una lista (ordinata) è una collezione di coppie composte da un elemento e un indice (indirizzo di memoria) che indica il modo successivo

RICERCA IN LISTA NON ORDINATA

SEARCH(L, K)

IF $L = \text{NIL}$ THEN

RETURN L

ELSE

IF $L \rightarrow \text{Key} = K$ THEN

RETURN L

ELSE

RETURN SEARCH ($L \rightarrow \text{Next}$, K)

INSEGNAMENTO IN LISTA IN TESTA

INSERT(L,k)

$x = \text{AllocaNodo}()$

$x \rightarrow \text{Key} = k$

$x \rightarrow \text{Next} = L$

RETURN x

ELIMINAZIONE NODO IN LISTA

In questo caso l'algoritmo di ricerca non ci aiuta perché dobbiamo modificare il campo next del nodo precedente, cosa impossibile con l'algoritmo che abbiamo definito.

CANCELLA(L,k)

IF $L = \text{NIL}$ THEN

RETURN L

ELSE

IF $L \rightarrow \text{key} = k$ THEN

$\text{RET} = L \rightarrow \text{Next}$

DEALLOCATE(L)

RETURN RET

ELSE

RET=CANCELLA ($L \rightarrow \text{Next}, k$)

scorrere alla fine e cancellare

$L \rightarrow \text{Next} = \text{RET}$

RETURN L