Esame di Algoritmi e Strutture di Dati 1
Lunedì 17 Maggio 2004

Nome:
Cognome:
Matricola:

Scrivere in forma leggibile il proprio nome, cognome e matricola sul testo del compito e su ogni foglio consegnato.

Consegnare solo la bella copia e il testo del compito.

Non è possibile consultare alcun tipo di materiale didattico.

Non è possibile uscire dopo l'inizio dello scritto.

Esercizio 1 (Punti 25)

Un *multi-insieme* è un insieme che può contenere elementi ripetuti. Ad esempio $\{1, 3, 5, 1, 5, 7\}$.

1. Scegliere una opporturna struttura di dati per rappresentare un multi-insieme.
2. Sulla base della struttura di dati scelta, implementare in modo *efficiente* le seguenti procedure:
   (a) $MultiSetInsert(S, a)$ che inserisce una occorrenza dell'elemento a nel multi-insieme S;
   (b) $MultiSetDelete(S, a)$ che rimuove una occorrenza dell'elemento a dal multi-insieme S;
   (c) $MultiSet2Set(S, T)$ che trasforma il multi-insieme S in un insieme (privo di ripetizioni) T. Ad esempio, se $S = \{1, 3, 5, 1, 5, 7\}$, allora $T = \{1, 3, 5, 7\}$.
3. Valutare la complessità computazionale (pessima) delle procedure scritte.

Soluzione

Rappresento il multi-insieme mediante una lista i cui oggetti hanno quattro campi: *key* contiene l'elemento dell'insieme, *mult* contiene la sua molteplicità, cioè il numero di occorrenze dell'elemento nell'insieme, *next* è un puntatore all'oggetto seguente nella lista e *prev* è un puntatore all'oggetto precedente nella lista.

La procedura $MultisetInsert(S, z)$ prende in ingresso un multi-insieme S (rappresentato come sopra) e un oggetto z tale che $key[z]$ è l'elemento da inserire in S. La complessià è lineare nel numero di elementi *distinti* di S.

Algoritmo 1 MultiSetInsert(S,z)

MultiSetInsert(S,z)

```
1:  x ← head(S)
2:  while x ≠ NIL and key[x] ≠ key[z] do
3:     x ← next[x]
4:  end while
5:  if x ≠ NIL then
6:     mult[x] ← mult[x] + 1
7:  else
8:     mult[z] ← 1
9:     next[z] ← head(S)
10:    prev[z] ← NIL
11:    prev[head(S)] ← z
12:    head(S) ← z
13: end if
```

La procedura $MultisetDelete(S, z)$ prende in ingresso un multi-insieme S (rappresentato come sopra) e un oggetto z tale che $key[z]$ è l'elemento da cancellare da S. La complessià è costante.

Algoritmo 2 MultiSetDelete(S,z)

MultiSetDelete(S,z)

```
1:  if mult[z] > 1 then
2:     mult[x] ← mult[x] − 1
3:  else
4:     if prev[z] ≠ NIL then
5:        next[prev[z]] ← next[z]
6:     else
7:        head(S) ← next[z]
8:     end if
9:     if next[z] ≠ NIL then
10:       prev[next[z]] ← prev[z]
11:    end if
12: end if
```

La procedura $Multiset2SetInsert(S, T)$ prende in ingresso un multi-insieme S (rappresentato come sopra) e restituisce una pila T che contiene gli elementi di S senza ripetizioni. La complessià è lineare nel numero di elementi *distinti* di S.

Algoritmo 3 Multiset2SetInsert(S,T)

Multiset2SetInsert(S,T)

1: $x \leftarrow head(S)$
2: **while** $x \neq$ NIL **do**
3: $\quad Push(T, key[x])$
4: **end while**

Esercizio 2 (Punti 5)

Si produca almeno un esempio per ciascuna delle seguenti categorie: albero ternario, albero ternario pieno, albero ternario completo. Gli esempi devono essere diversi tra loro.

Ha più foglie un albero binario completo di altezza $2h$ oppure un albero quaternario completo di altezza h? Motivare la risposta.

Soluzione

Il numero di foglie è lo stesso. Infatti, un albero binario completo di altezza $2h$ ha 2^{2h} foglie, mentre un albero quaternario completo di altezza h ha $4^h = (2^2)^h = 2^{2h}$ foglie.