ESERCIZIO DI ASD DEL 6 APRILE 2009

Massimo di Insiemi Disgiunti

Struttura Dati e Operazioni. Utilizziamo per implementare le operazioni richieste le liste concatenate già usate a lezione per implementare Weighted-Union. In particolare ogni elemento x avrà i seguenti campi:

- $key[x]$. La chiave dell'elemento x;
- $rap[x]$. Il puntatore al rappresentante della lista contenente x;
- $next[x]$. Il puntatore al successore di x nella lista;
- $last[x]$. Il puntatore all'ultimo elemento della lista contenente x. Questo campo viene mantenuto aggiornato solo nel rappresentante.
- $length[x]$. La lunghezza della lista contenente x. Questo campo viene mantenuto aggiornato solo nel rappresentante.

Per implementare efficientemente la richiesta del massimo facciamo in modo che il massimo si trovi sempre nel rappresentante. Per garantire questa proprietà è sufficiente che nell'unione si scambino le chiavi dei due rappresentanti in modo da mettere per prima quella maggiore.

Algorithm 1 MAKE(x)

```
1: rap[x] ← x
2: next[x] ← NIL
3: last[x] ← x
4: length[x] ← 1
```

Algorithm 2 FIND(x)

```
1: return rap[x]
```

Algorithm 3 FIND-MAX(x)

```
1: return rap[x]
```

Algorithm 4 UNION(x, y)

```
1: if rap[x] ≠ rap[y] then
2:     LINK(rap[x], rap[y])
3: end if
```

Date: 6 Aprile 2009.

Algorithm 5 LINK(z, w)

```
1: if length[z] > length[w] then
2:    if key[w] > key[z] then
3:       ScambiaKey(z, w)
4:    end if
5:    length[z] → length[z] + length[w]
6:    u ← last[z]
7:    last[z] ← last[w]
8:    next[u] ← w
9:    while u ≠ NIL do
10:      rap[u] ← z
11:      u ← next[u]
12:   end while
13: else
14:   Link(w, z)
15: end if
```

Algorithm 6 SCAMBIAKEY(z, w)

```
1: k ← key[z]
2: key[z] ← key[w]
3: key[w] ← k
```

Algorithm 7 SET(x)

```
1: y ← rap[x]
2: while y ≠ NIL do
3:    Print(key[y])
4:    y ← next[y]
5: end while
```

Correttezza. Mentre è immediato vedere che SET è corretta, occorre dimostrare formalmente che FIND-MAX è corretta. Questo equivale a dire che occorre dimostrare formalmente che il massimo è sempre il primo elemento della lista. Mostriamo una bozza di dimostrazione di correttezza.

Theorem 1. *Con le operazioni sopra descritte il massimo della lista contenente x è il rappresentante della lista contenente x.*

Dimostrazione. Procediamo per induzione sul numero di operazioni di unione che sono state applicate per ottenere la lista contenente x.

BASE. Non sono state effettuate unioni. In questo caso la lista contenente x contiene solo x che è sia il massimo che il rappresentante.

PASSO. Supponiamo per ipotesi induttiva che la tesi sia vera se la lista è stata ottenuta con al più $n-1$ unioni e dimostriamo che vale la tesi anche se la lista è stata ottenuta con n unioni.

Se la lista contentente x è stata ottenuta con n unioni, allora l'ultima unione ha unito due liste L_1 ed L_2 che erano state ottenute con al più $n-1$ unioni ciascuna. Quindi per ipotesi induttiva L_1 ha come rappresentante l'elemento y_1 con chiave maggiore ed L_2 ha come rappresentante l'elemento y_2 con chiave maggiore. Quindi

la chiave maggiore nella lista ottenuta dopo n unioni è il più grande tra la chiave di y_1 e quella di y_2. Quando si uniscono L_1 ed L_2 viene messa per prima la lista con più elementi, ma se questa non ha il rappresentante con chiave maggiore, allora le chiavi dei due rappresentanti vengono scambiate. □

Complessità.

- Un'operazione di MAKE costa $\Theta(1)$;
- Un'operazione di FIND costa $\Theta(1)$;
- Un'operazione di FIND-MAX costa $\Theta(1)$;
- Un'operazione di UNION costa $\Theta(min(length(L_1), length(L_2))$, dove L_1 ed L_2 sono le due liste che vengono unite;
- Un'operazione di SET costa $\Theta(length(L))$, dove L è la lista che viene stampata.

Se consideriamo m operazioni di cui n MAKE e p SET, abbiamo che:

- m operazioni di MAKE, UNION, FIND, e FIND-MAX costano $O(m + n \log n)$ (la dimostrazione è la stessa vista a lezione nel caso di Weighted-Union);
- p operazioni di SET costano $O(p * n)$, perchè nel caso peggiore possono essere effettuate tutte su un unico insieme contenente tutti gli elementi.

Quindi complessivamente abbiamo costo $O(m + n \log n + pn)$.