008AA – ALGORITMICA E LABORATORIO

Verifica del 24 Maggio 2010

Cognome Nome: **N. Matricola:** **Corso: A B**

Esercizio 1. *(8 punti)* Siano dati k vettori di interi, aventi ciascuno dimensione n. Questi vettori sono ordinati in modo crescente e sono memorizzati in una matrice M di dimensione $k \times n$. Si progetti un algoritmo che esegue la loro *fusione* (merge) producendo in output un vettore $A[1, N]$ ordinato di dimensione $N = kn$ in modo crescente. L'algoritmo proposto deve avere complessità in tempo al caso pessimo pari a $O(N \log k)$.

(Dettagliare le strutture dati utilizzate, e spiegare ANCHE a parole il funzionamento dell'algoritmo proposto.)

La soluzione consiste nell'uso di uno heap di minimo che contiene in ogni istante (al più) un elemento per ciascuno dei k vettori, e quindi è di dimensione $\leq k$. Le operazioni su questo heap richiedono dunque un tempo $O(\log k)$ al caso pessimo.

All'inizio della computazione l'heap contiene il primo elemento di ogni vettore, e quindi ha dimensione esattamente k. A ogni passo del merge, si estrae il minimo dallo heap, lo si aggiunge alla sequenza A, e poi lo si "sostituisce" nello heap con l'elemento che lo segue nel suo vettore, se esiste, altrimenti la corrispondente sequenza è esaurita e quindi l'heap decresce in dimensione (da ciò il $\leq k$). Siccome il merge richiede N passi, e ciascuno costa $O(\log k)$, la complessità totale è proprio quella richiesta dall'esercizio.

Unico accorgimento è quello di definire come chiave dello heap il valore degli elementi $M[i, j]$, mentre i dati satellite sono dati dalla coppia $\langle i, j \rangle$ così da recuperare, dato $M[i, j]$, il vettore di provenienza e la sua posizione. Assumiamo di indicare questa tripla con i campi auto-esplicativi `key`, `col` e `row`.

```
for (i=0; i < k; i++) Heap_Insert( H, < M[i,0],i,0 > );

for (i=0; i < N; i++){
    m = Extract_Min(H);
    A[i] = m.key;
    if (m.col < n-1) { Heap_Insert(H, <M[m.row,m.col+1], m.row, m.col+1>); }
    }
```

Si osserva che avremmo potuto procedere anche per BinaryMerge() successivi, che operavano prima su coppie di vettori entrambi di dimensione n, poi $2n$, e così via. La differenza con la soluzione precedente sarebbe stato l'uso di uno spazio di lavoro pari a $\Theta(N)$, mentre la soluzione suddetta usa spazio $O(k)$.

Cognome Nome: **N.Matr:**

Esercizio 2. *(6 punti)* Sia data una tabella hash T di dimensione m, con funzione hash $h(k) = k$ `mod` m e collisioni gestite mediante liste di trabocco. Si progetti un algoritmo di ricerca per una chiave k in T che, se presente, la sposta in testa alla sua lista di trabocco se k è maggiore della chiave memorizzata nell'elemento precedente della sua lista.

Siccome le liste di trabocco sono mono-direzionali, per poter accedere l'elemento precedente a quello contenente la chiave k (sia esso x), dobbiamo adottare lo schema del traino visto in classe e quindi utilizzare una variabile d'appoggio y che punta all'elemento che precede x.

```
Search_hash(T,k)
{
    h = k mod m;
    if (T[h] == NULL) return NULL;
    x = T[h].succ; y = T[h];
    while ((x != NULL) && (x.key != k))
        { y=x; x = x.succ; }
    if ((x != NULL) && (y.key < k))
       // muovi in testa alla lista T[h], codice dato in classe
    return x;
}
```

Cognome Nome: **N.Matr:**

Esercizio 3. *(5+5+8 punti)* Sia dato il grafo pesato e non-orientato $G = (V, E)$ costituito da $n = 6$ vertici e $m = 9$ archi (non orientati) così definiti: $\{(1, 2, 2), (2, 3, 3), (2, 4, 1), (1, 5, 7), (4, 5, 4), (3, 6, 9), (5, 6, 8)\}$ dove le prime due componenti indicano gli estremi dell'arco, e la terza il suo peso.

- Simulare su G il funzionamento dell'algoritmo di Kruskal, specificando l'ordine con il quale vengono inseriti gli archi nell'MST.
- Simulare su G il funzionamento dell'algoritmo di Prim a partire dal vertice 6, specificando l'ordine con il quale vengono inseriti gli archi nell'MST e l'evoluzione della coda di priorità.
- Dimostrare che, dato un grafo pesato G i cui archi hanno peso distinto, l'arco di peso massimo di un qualunque ciclo di G non può appartenere ad alcun MST di G.

Kruskal: (2,4) (1,2) (2,3) (4,5) (5,6)

Prim: (6,5) (5,4) (4,2) (2,1) (2,3). L'evoluzione della coda di priorità segue tenendo conto del fatto che ogni vertice in essa non fa ancora parte della componente di 6 e la chiave associata corrisponde all'arco di peso minimo che incide su quel vertice e attraversa il tagli corrente.

Sia C un ciclo di G e (u, v) il suo arco di peso massimo. Se (u, v) appartiene a un MST di G, sia questo T, allora la sua cancellazione porterebbe T a formare due componenti connesse. D'altra parte C è un ciclo e quindi il cammino che connette u a v in esso deve disporre di un arco che connette le due componenti di T. Questo arco ha peso inferiore a (u, v) per ipotesi, e quindi il suo inserimento in $T/\{(u, v)\}$ porta a formare un nuovo albero di perso inferiore a T. Contraddizione!