008AA – ALGORITMICA E LABORATORIO

Appello del 14 Luglio 2009

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

Esercizio 1. *(4+5+7 punti)* L'elemento *mediano* di un insieme S di m elementi (distinti) è quell'elemento che ha esattamente $\lfloor m/2 \rfloor$ elementi minori in S. Ad esempio, l'insieme $S = \{1, 4, 6, 8, 9, 12\}$ ha mediano 8. Siano dati due insiemi A e B di n elementi ciascuno, tutti distinti tra loro, rappresentati come sequenze ordinate memorizzate negli array $S_A[0, n-1]$ e $S_B[0, n-1]$.

(a) Progettare un algoritmo che trova il *mediano* di $A \cup B$ in tempo $O(n)$ al caso pessimo.

(b) Progettare un algoritmo che dato un elemento x di S_A o S_B, calcola il numero di elementi di $A \cup B$ che sono minori di x in tempo $O(\log n)$.

(c) Utilizzare l'algoritmo del punto precedente per progettare un algoritmo ricorsivo che calcola il mediano di $A \cup B$ in tempo $O(\log^2 n)$.

(Si prega di accompagnare ogni pseudo-codice con una descrizione informale dell'idea algoritmica sottostante.)

Soluzione (a): Basta utilizzare la procedura di Fusione dell'algoritmo MergeSort per fondere le due sequenze e fermarsi non appena sono stati scanditi n elementi, visto che $|A \cup B| = 2n$. Si noti che durante la fusione non si esaurisce mai interamente una delle due sequenze.

```
count = 0;
for(i=j=0; count<n; ){
    if(SA[i] < SB[j]){ i++; } else { j++; }
    count++;
    }
if(i == n) return SB[j];
if(j == n) return SA[i];
if(SA[i] < SB[j]) { return SA[i]; } else { return SB[j]; }
```

Soluzione (b): Basta usare la ricerca binaria su S_A e S_B notando che in un caso l'elemento viene sicuramente trovato (visto che appartiene alla sequenza), mentre nell'altro occorre garantire che la procedura restituisca la posizione della chiave cercata.

Soluzione (c): Sia `RicBin(S,n,k)` la procedura che risolve il punto (b) su un array S di n elementi e restituisce la posizione p tale che $S[0, p-1]$ contiene tutti gli elementi (strettamente) minori di k. A questo punto è sufficiente fare una ricerca binaria sugli elementi di S_A e S_B verificando se uno di essi è il mediano di $A \cup B$. Notiamo che se stiamo cercando sugli elementi di S_A (risp. S_B) ed esaminiamo l'elemento $x = S_A[i]$ (risp. $S_B[i]$), allora x è banalmente maggiore di i elementi in S_A (risp. S_B), per cui è necessario eseguire solo `RicBin`(S_B, n, x) (risp. `RicBin`(S_A, n, x)).

Ne consegue, quindi, che possiamo eseguire la procedura che segue prima come `Mediano(SA,SB,n)` e poi come `Mediano(SB,SA,n)` e prendere quell'unico valore non nullo che viene restituito da una delle chiamate (visto che il mediano occorre in una sola delle due sequenze, essendo tutti gli elementi distinti).

```
Mediano(SA,SB,n)  // verifica se un elemento di SA  il mediano di SA+SB
    L=0; E=n-1;
    while(L <= E){
        m = (L+E)/2;
        p = RicBin(SB,n,SA[m]);
        if (p + m == n) return SA[m]; // trovato
        if (p + m < n) { L = m+1; } else { E = m-1; }
        }
    return null; // non trovato
```

Cognome Nome: **N.Matr:**

Esercizio 2. *(6 punti)* Sia dato un albero binario T contenente n nodi, aventi ciascuno un campo binario `flag` (che quindi assume valore 0 oppure 1). Progettare un algoritmo ricorsivo di complessità $O(n)$ che, ricevuto in input un intero $h \geq 0$, restituisce in output il numero di nodi di T che hanno *altezza* h e campo `flag` uguale a 1. L'altezza di un nodo è la sua distanza massima da una foglia discendente, per cui le foglie hanno altezza 0.

Soluzione: Standard, basata su ricorsione.

Cognome Nome: **N.Matr:**

Esercizio 3. *(6+2 punti)* Sia G un grafo non orientato e connesso rappresentato mediante liste di adiacenza. Ciascuno dei vertici di G ha un campo binario `flag` (che quindi assume valore 0 oppure 1). Sia $d_1(v, w)$ la *1-distanza* tra due vertici v e w di G, ossia la lunghezza del cammino minimo che li connette in G utilizzando *solo* vertici il cui campo `flag` è uguale a 1 (inclusi v e w): nel caso tale cammino non esista, vale $d_1(v, w) = -1$. Si definisce *1-diametro* di G la *massima* 1-distanza tra tutte le coppie di vertici: $\max_{v,w \in G} d_1(v, w)$.

(a) Progettare un algoritmo che calcola l'1-diametro di G.

(b) Valutare la sua complessità in tempo, assumendo che G consista di n vertici e m archi.

Soluzione: Sia `BFS1(G,s)` la classica procedura BFS, eseguita a partire dal vertice s, e modificata in modo tale che un vertice viene visitato soltanto se il suo `flag` è uguale a 1. A questo punto possiamo scrivere:

```
Diametro1(G)
    max = -1;
    for(i=0; i < n; i++){
        BFS1(G,i);
        for(j=0; j < n; j++)
            if( (d[j] > max) && (raggiunto[j])) max = d[j];
        }
    print max;
```