008AA – ALGORITMICA E LABORATORIO

Verifica del 7 aprile 2009

Esercizio 1. *(8 punti)*

Sia dato un array $A[0, n-1]$ di stringhe (visto come `char **A`) il cui valore può essere soltanto: {`vero`, `forse`, `falso`}.

Scrivere una funzione `C` che riceve in input l'array A e la sua lunghezza n, e in tempo $O(n)$ riarrangia le stringhe di A in modo che tutte le stringhe `falso` precedano le stringhe `forse`, e queste ultime precedano le stringhe `vero`.

```
void Sort3(char **A, int n)
{
    int num_falso, num_forse;
    int pos_falso, pos_forse, pos_vero;
    int i;

    char **B = (char **) malloc(n * sizeof(char *));
    num_falso = num_forse = 0;
    for(i=0; i < n; i++){
        if(strcmp(A[i],"falso") == 0) num_falso++;
        if(strcmp(A[i],"forse") == 0) num_forse++;
        }

    pos_falso = 0;  //posizione partenza stringhe "falso"
    pos_forse = num_falso;  //posizione partenza stringhe "forse"
    pos_vero = num_falso + num_forse; //posizione partenza stringhe "vero"
    for(i=0; i < n; i++){
        if(strcmp(A[i],"falso") == 0){
            B[pos_falso] = A[i];
            pos_falso++;
        } else if(strcmp(A[i],"forse") == 0){
            B[pos_forse] = A[i];
            pos_forse++;
        } else {
            B[pos_vero] = A[i];
            pos_vero++;
        }
    }
    for(i=0; i < n; i++) A[i] = B[i];
    free(B);
}
```

Si noti che abbiamo permutato le stringhe in A, piuttosto che riscrivere le tre possibili stringhe {`falso, forse, vero` } in base alle loro frequenze, per i motivi discussi in classe sulla possibile presenza di *dati satellite*.

Esercizio 2. *(4+4 punti)* Sia data la funzione ricorsiva:

```
Foo( n ) {
   if(n==1) return 1;
   a = 0;
   for( i = 0; i < n; i = i+1 )
      for( j = 0; j < n/2; j = j+1 )
         for( k = 0; k < 5; k = k+1 )
            a += 1;

   return Foo( n/2 ) + a;
}
```

- Scrivere la relazione di ricorrenza $T(n)$ per la complessità in tempo al caso pessimo della funzione `Foo`.
- Trovare la soluzione per $T(n)$ in forma chiusa.

Soluzione. La relazione di ricorrenza risulta $T(n) = T(n/2) + O(n^2)$, con $T(1) = O(1)$. Utilizzando il Teorema master si imposta: $f(n) = cn^2$, e si ricava $\gamma > 0$ tale che $f(n/2) = \gamma f(n)$, ossia $cn^2/4 = \gamma cn^2 \Rightarrow \gamma = 1/4$ (caso 1). Quindi $T(n) = O(n^2)$.

Esercizio 3. *(2+6 punti)*

Un array $V[0, n-1]$ è detto *convesso semplice* se esiste un indice $0 \le j < n-1$ tale che $V[0] = V[1] = V[2] = \cdots = V[j]$ e, per ciascun $i \ge j$, vale $V[i] < V[i+1]$. Per esempio, l'array $V = \{3, 3, 3, 5, 8, 14, 15\}$ è convesso semplice, mentre $V' = \{3, 3, 3, 5, 8, 8, 14, 15\}$ e $V'' = \{5, 8, 14, 5\}$ non lo sono. Progettare un algoritmo che, ricevuto in input un array convesso semplice V, trova l'indice j in tempo: `(a)` $O(n)$, e `(b)` $O(\log n)$.

Soluzione (a). Essa consiste di un semplice ciclo `for` che trova j per scansione, partendo da $j = 0$. Si ferma quando $V[j] < V[j+1]$. Dalla traccia si osserva che $j < n-1$ per cui $j + 1 < n$ e quindi il confronto avviene sempre all'interno di A.

Soluzione (b). Modifichiamo la procedura di ricerca binaria in modo da trovare l'occorrenza più a destra del valore $A[0]$. Per fare ciò, prendiamo l'elemento centrale e lo confrontiamo con il suo adiacente sulla destra (che esiste sempre nel caso di $s < d$, visto la scelta del perno). Se sono diversi (e quindi essendo convesso semplice, $A[m] < A[m+1]$), possiamo ricorrere sulla metà sinistra di A; altrimenti, siamo giustificati a saltare la posizione m poichè per questa vale $A[m] = A[m+1]$. Tale approccio funziona anche nel caso di vettore con due elementi uguali.

```
Cerca(A,s,d)
{
   if(s==d) return s;
   m = (s+d)/2;
   if (A[m] < A[m+1]) return Cerca(A,s,m);
      else return Cerca(A,m+1,d);
}
```

Basta invocare la procedura come `Cerca(A,0,n-1)`. La sua complessità in tempo è $O(\log n)$.

Esercizio 4. *(8 punti)*

Sono date n persone identificate dagli interi $\{0, 1, \ldots, n-1\}$, il cui guadagno mensile è memorizzato nell'array $G[0, n-1]$. Dati due interi k e g, scrivere un algoritmo che stabilisce se esiste un sottoinsieme di k persone il cui guadagno mensile totale è esattamente g. Per esempio, se $G = \{5, 2, 3, 2, 7\}$, la risposta è positiva per $k = 3$ e $g = 7$, in quanto possiamo scegliere $\{2, 3, 2\}$; la risposta è invece negativa per $k = 3$ e $g = 5$.

Soluzione. Si tratta dell'applicazione di `GeneraBinarie` per cui è sufficiente specializzare la procedura `Elabora`(B, G, k, g, n) come segue.

```
Elabora(B,G,g,k,n)
{
    num=0; guadagno = 0;
    for(i=0; i< n; i++){
        num += B[i];
        guadagno += G[i] * B[i];
        }
    if ((num == k) && (guadagno == g)){
        print "trovato"; STOP;
        }
}
```

La complessità in tempo è $O(n\,2^n)$.