Esercizi vari

Alberto Montresor

19 Agosto, 2014

Alcuni degli esercizi che seguono sono associati alle rispettive soluzioni. Se il vostro lettore PDF lo consente, è possibile saltare alle rispettive soluzioni tramite collegamenti ipertestuali. Altrimenti, fate riferimento ai titoli degli esercizi. Ovviamente, si consiglia di provare a risolvere gli esercizi personalmente, prima di guardare la soluzione.
Per molti di questi esercizi l'ispirazione è stata presa dal web. In alcuni casi non è possibile risalire alla fonte originale. Gli autori originali possono richiedere la rimozione di un esercizio o l'aggiunta di una nota di riconoscimento scrivendo ad `alberto.montresor@unitn.it`.

1 Problemi

1.1 Vero o falso?

- Per determinare se due alberi binari di ricerca sono identici, uno può fare una in-visita in entrambi e confrontare i risultati. Vero o falso?
- Avete a disposizione un'operazione di moltiplicazione di interi, in grado di operare con input di qualsiasi dimensione. È possibile calcolare a^{2^n} in $\Theta(n)$ operazioni.

Soluzione: Sezione 2.1

1.2 Bounding Box

Una nozione importante in computer graphics è quella di bounding box di un'insieme di punti, che ìl più piccolo rettangolo tale per cui tutti i punti sono contenuti nel rettangolo o giacciono sui bordi (nota: parallelo agli assi x,y). Ovviamente, un bounding box può essere descritto da due coordinate y per i lati superiore-inferiore e due coordinate x per i lati sinistro-destro.
Ancora una volta, consideriamo un insieme *dinamico* di punti; vogliamo costruire una struttura dati "insieme di punti" in grado di supportare le operazioni `isPresent(x,y)` e `insert(x,y)`. Inoltre deve supportare le operazioni

- `bbox()` che ritorna una quadrupla $(x_{min}, x_{max}, y_{min}, y_{max})$ rappresentante il bounding box per i punti;
- `bbox(`x_0`,` x_1`)` che ritorna il bounding box per tutti i punti la cui coordinata x è inclusa nel range x_0, x_1.

Che tipo di struttura dati utilizzereste? Quali sono i costi computazionali delle varie operazioni, detto n il numero di punti presenti nell'insieme?

Soluzione: Sezione 2.2

1.3 Distribuzione

Descrivere un algoritmo che, dato un vettore di n interi compresi fra 1 e k, lo preprocessa per poter rispondere in seguito a domande del tipo "quanti valori ci sono nell'intervallo $[a \cdots b]$?" in tempo $O(1)$. Tempo richiesto: $O(n + k)$.

Soluzione: Sezione 2.3

1.4 Secondo "terzile"

Sia $A[1 \ldots n]$ un vettore contenente $n = 3m$ interi distinti. Si consideri il problema di stampare gli elementi di A maggiori o uguali ad almeno m interi in A e minori o uguali ad almeno m interi in A (non necessariamente in ordine).

- Si proponga un algoritmo lineare per risolvere il problema proposto;
- Si discuta la correttezza e la complessità dell'algoritmo definito.

Nota: si consiglia di *utilizzare* un algoritmo noto, senza doverlo riscrivere.

Soluzione: Sezione 2.4

1.5 0-1

Data una stringa composta da 0 e 1, trovare la lunghezza della più lunga sottostringa contenente un numero k di 0 consecutivi seguita da un numero k di 1 consecutivi. Calcolare la complessità e spiegarne il funzionamento.
Si noti che è possibile che immediatamente prima della vostra sottostringa vi siano ancora 0 o che immediatemente dopo la vostra sottostringa vi siano ancora 1, ma non può accadere contemporaneamente, altrimenti la vostra sottostringa non è la più lunga.

Soluzione: Sezione 2.5

1.6 Un algoritmo alla "moda"

Scrivere un algoritmo per trovare la "moda" di un insieme di dati (ovvero il valore che compare più spesso). Nel caso di più valori che possono essere definiti come moda, ritornare il valore numericamente più alto. Ad esempio, nel vettore

```
1 2 7 12 3 6 7 18 28 11 28 11 12 12 14 3 5 8 9 2 38 28
```

la moda è 28.

Soluzione: Sezione 2.6

1.7 Cerca la somma

Sia A un vettore di n valori reali e x un valore reale. Scrivere un algoritmo $O(n \log n)$ nel caso pessimo che ritorni vero se esistono due elementi del vettore la cui somma è x. Spiegare il suo funzionamento.

Soluzione: Sezione 2.7

2 Soluzioni

2.1 Vero o falso?

- Falso. Si prenda due alberi di ricerca negli elementi 1,2 (uno con radice 1, l'altro con radice 2). Hanno la stessa invisita.
- Vero. È sufficiente fare elevamenti al quadrato successivi, che richiederanno $\log 2^n = n$ operazioni.

2.2 Bounding Box

Utilizziamo alberi RB, che mantengono le proprietà di ordinamento e ci assicurano la possibilità di fare ricerca.

2.3 Distribuzione

L'algoritmo è in realtà molto semplice: si utilizza Counting Sort, con il quale si ottiene, per ogni $i \in [1 \cdots k]$, il numero di elementi uguali a i. Con un semplice ciclo, è possibile ottenere il conteggio di tutti gli elementi che sono minori o uguali a i. A quel punto, la risposta alla domanda si ottiene semplicemente tramite una sottrazione.

```
pre-process(int[] A, int[] B, int n, int k)
    for j = 1 to k do
        B[j] = 0
    for i = 1 to n do
        B[A[i]] = B[A[i]] + 1
    for j = 2 to k do
        B[j] = B[j] + B[j − 1]
```

```
range(int[] B,int a, int b)
    return B[b] − B[a − 1]
```

2.4 Secondo "terzile"

L'idea è quella di utilizzare l'algoritmo della selezione, che permette di individuare la posizione dell'elemento in posizione m e $2m$ nel vettore dei valori ordinati, senza effettivamente ordinarli. Il costo di tali operazioni è $O(n)$. A questo punto, si scorre tutto il vettore e si individuano i valori superiori e inferiori, come richiesto.

```
printOneThird(int[] A, int n)
    int m = n/3
    int min = selezione(A, 1, n, m)
    int max = selezione(A, 1, n, 2m)
    for i = 1 to n do
        if A[i] > min and A[i] < max then
            print A[i]
```

dove selezione$(A, primo, ultimo, k)$ è contenuta nel Capitolo 16.

2.5 0-1

Un algoritmo semplice scorre la stringa cercando tutte le sequenze di 0 e 1 consecutivi, le misura e mantiene una variabile *max* per mantenere le informazioni sulla sequenza più lunga.

```
zerouno(int[] S, int n)
    int count_0 = iif(S[1] = 0, 1, 0)
    int count_1 = iif(S[1] = 1, 1, 0)
    int max = 0
    for int i = 2 to n do
        if S[i] == 1 then
            if S[i − 1] = 0 then
                count_1 = 0
            count_1 = count_1 + 1
        else
            if S[i − 1] == 1 then
                max = max(max, min(count_0, count_1))
                count_0 = 0
            count_0 = count_0 + 1
    return max
```

2.6 Un algoritmo alla "moda"

L'algoritmo funziona pre-ordinando i valori e cercando la più lunga sequenza di valori uguali consecutivi.

```
int sort(int[] A, int n)
    sort(A, n)                          % Con qualche algoritmo O(n log n)
    int max = 0
    int moda
    int count = 1
    last = A[1]
    for i = 2 to n + 1 do
        if i > n or A[i] ≠ last then
            if count ≥ max then
                max = count
                moda = last
            count = 0
            last = A[i]
        count = count + 1
    return moda
```

2.7 Cerca la somma

Si ordina il vettore tramite HeapSort o MergeSort (che hanno un costo nel caso pessimo di $O(n \log n)$. Per tutti i valori $A[i]$, utilizziamo un algoritmo di ricerca binaria per cercare il valore $x - A[i]$. Il costo è pari a

$O(n \log n)$ per l'ordinamento e $O(n \log n)$ per le n ricerche binarie di costo $O(\log n)$.

```
search-sum(int[] A, int n, int x)
    sort(A, n)                          % Con qualche algoritmo O(n log n)
    for i = 1 to n do
        if binarySearch(A, n, x − A[i]) then
            return true
    return false
```

3 Problemi aperti

3.1 h-index

L'h-index è un indice che misura la capacità di un ricercatore di attirare citazioni. Un ricercatore ha un h-index pari a k se k è il più alto valore tale per cui le sue k pubblicazioni più citate hanno ricevuto almeno k citazioni. Ad esempio, l'h-index di Alberto Montresor è pari a 21: le 21 pubblicazioni più citate hanno ricevuto almeno 21 citazioni, mentre non è vero che le prime 22 abbiano ricevuto almeno 22 citazioni.
Siano dati un insieme di citazioni rappresentato come grafo, dove i vertici sono dati dalle pubblicazioni e un arco (a, b) significa che la pubblicazione a cita la pubblicazione b; e sia data una funzione $isauthor(v, r)$ che ritorna vero se la pubblicazione v è stata scritta dal ricercatore r.
Scrivere un algoritmo che calcola $h - index(r)$ e valutarne la complessità.