Corso di laurea in Matematica
Insegnamento di Informatica generale
Lezioni in modalità mista o a distanza

Esercizi assegnati in vecchi scritti – 1/2

Giancarlo Bongiovanni
Ivano Salvo

SAPIENZA
UNIVERSITÀ DI ROMA

Queste dispense sono state realizzate sulla base delle slide
preparate da T. Calamoneri e G. Bongiovanni
per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

Esercizio 1 (1) – compito del 28/1/2020

Siano A e B i puntatori alle radici di due alberi binari di ricerca memorizzati tramite record e puntatori di cui sia noto che hanno rispettivamente n_1 ed n_2 nodi. Progettare un algoritmo che, presi in input A e B, restituisca in output un vettore ordinato di lunghezza massima $n_1 + n_2$ contenente l'unione delle chiavi dei due alberi (senza ripetizioni). L'algoritmo deve essere lineare nella somma del numero dei nodi dei due alberi.

IDEE

- Si potrebbe pensare di visitare i due ABR, ricopiare i loro elementi in un vettore e poi ordinarlo. Ma questo costerebbe troppo, non potendo utilizzare il counting sort.
- Sappiamo che la visita inorder su un ABR produce la sequenza ordinata ed ha un costo lineare nel numero di nodi dell'ABR;
- Sappiamo anche che la funzione `Fondi()` del Mergesort, partendo da due vettori ordinati, produce un unico vettore ordinato ed ha un costo lineare nel numero totale di elementi.

Esercizio 1 (2) – compito del 28/1/2020

SOLUZIONE

1. Si opera un visita inorder su ciascuno dei due ABR, riempiendo un corrispondente vettore: $\Theta(n_1 + n_2)$.
2. Si applica ai due vettori così ottenuti la fusione del mergesort, modificata per eliminare i doppioni: $\Theta(n_1 + n_2)$.
3. La modifica della fusione opera come segue:
   - una volta selezionato il minimo da ricopiare e fatta la copia, la fusione scorre su entrambi i vettori fino a trovare in ciascuno un elemento maggiore di tale minimo;
   - prosegue poi i confronti a partire da tali elementi;
   - anche la eventuale copia finale della porzione residua di uno dei due vettori dovrà ignorare i doppioni;
   - questa modifica non altera il costo computazionale complessivo, dato che i due vettori si scorrono in ogni caso una sola volta.

Esercizio 2 (1) – compito del 18/7/2019

Siano dati due max-heap H_1 ed H_2, contenenti rispettivamente n_1 ed n_2 chiavi. Si progetti un algoritmo che prenda in input i due vettori su cui sono memorizzati H_1 ed H_2 e restituisca in output un vettore con n_1 + n_2 elementi su cui sia memorizzato un nuovo max-heap che contiene le chiavi di H_1 ed H_2. L'algoritmo dovrebbe essere lineare nella somma del numero dei nodi dei due alberi.

IDEE

- Si potrebbe pensare di ricopiare nel nuovo vettore uno dei due heap e poi aggiungere, uno alla volta, gli elementi dell'altro utilizzando la `heapify_bottom_up()`.
- Questa soluzione però costa troppo perché ogni inserimento può richiedere un riaggiustamento di costo logaritmico.
- Ma sappiamo che `Buildheap()` ha un costo lineare e trasforma un generico vettore in uno heap.

Esercizio 2 (2) – compito del 18/7/2019

SOLUZIONE

1. Copiamo i due heap H_1 ed H_2 in un nuovo vettore, uno dopo l'altro così come sono:

$$\Theta(n_1 + n_2)$$

2. Chiamiamo `Buildheap()` sul nuovo vettore:

$$\Theta(n_1 + n_2)$$

Esercizio 3 (1) – compito del 10/9/2018

Sia G = (V, E) un grafo e si assuma che i nodi in G siano colorati mediante il colore rosso oppure verde. Si consideri il problema di determinare se G possiede un ciclo contenente solo nodi verdi. Si descrivano le strutture dati che si preferisce usare per rappresentare il grafo ed il colore dei nodi. Poi si progetti un algoritmo efficiente che risolva il problema.

IDEE

- Bisogna utilizzare un vettore aggiuntivo *colore[ ]* per memorizzare il colore di ciascun nodo.
- Disponendo di tale informazione si può modificare la visita in modo che scelga come sorgente un nodo verde e prenda in considerazione gli adiacenti di un nodo solo se sono verdi, ignorando quelli rossi. Questo corrisponde a visitare il sottografo indotto *G' = (V', E')* costituito dai soli vertici verdi e dagli archi in essi incidenti.
- Quindi si verifica se G' contiene un ciclo.

Esercizio 3 (2) – compito del 10/9/2018

SOLUZIONE

1. Aggiungiamo un vettore *colore[ ]*, con un elemento per ciascun nodo.
2. Scegliamo come nodo di partenza un nodo verde.
3. Modifichiamo la visita che risolve il problema di determinare se un grafo ha un ciclo (già vista in un esercizio precedente) facendole ignorare gli adiacenti rossi.
4. Il costo è quello della normale visita, applicata al sottografo indotto *G' = (V', E')* costituito dai soli vertici verdi e dagli archi che li uniscono

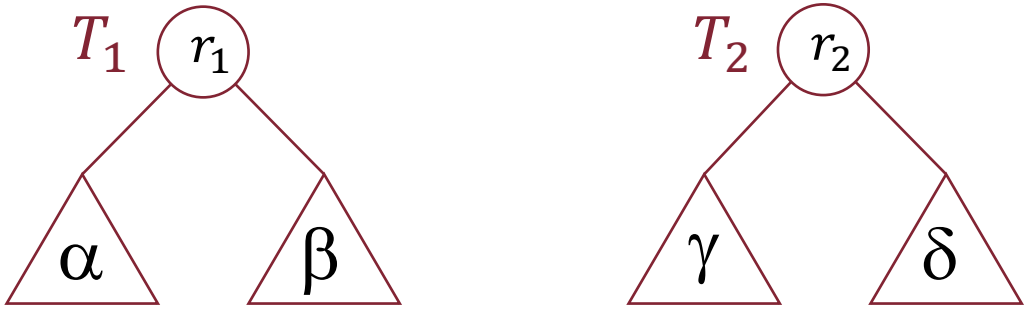
Esercizio 3 (3) – compito del 10/9/2018

```
Funzione CicliVerdi (G: grafo)
   q ← NULL
   Assegna a tutti i nodi v
      G.marked[v] ← false; G.pred[v] ← NULL
   Scegli il nodo (verde) di partenza s
   G.marked[s] ← true
   Enqueue(q, s)
   Finché la coda q non è vuota
      u ← Dequeue(q) //si lavora su u
      Visita il vertice u
      Per ogni vertice v adiacente ad u
         if (G.colore[v]=verde)
            if G.marked[v] = false then
               G.marked[v] ← true
               G.pred[v] ← u
               Enqueue(q, v)
            else if v ≠ G.pred[u] then return true
   return false
```

Costo computazionale: limitato superiormente (O) dal costo della visita.

Esercizio 5 (1) – compito del 11/6/2015

Siano dati due alberi binari di ricerca T_1 con n_1 nodi e T_2 con n_2 nodi. T_1 e T_2 sono memorizzati tramite record e puntatori; inoltre T_i , i = 1, 2, è costituito da una radice r_i , dal suo sottoalbero sinistro T_i^{sx} e dal suo sottoalbero destro T_i^{dx} con la proprietà che per ogni quaterna di chiavi k_1^{sx} in T_1^{sx} , k_1^{dx} in T_1^{dx} , k_2^{sx} in T_2^{sx} , k_2^{dx} in T_2^{dx} vale che $k_1^{sx} < k_2^{sx} < k_1^{dx} < k_2^{dx}$. Si progetti un algoritmo con costo lineare nell'altezza dei due alberi che fonda T1 e T2 in un unico albero binario di ricerca con $n_1 + n_2$ nodi.



$$\alpha < \gamma < \beta < \delta$$

Esercizio 5 (2) – compito del 11/6/2015

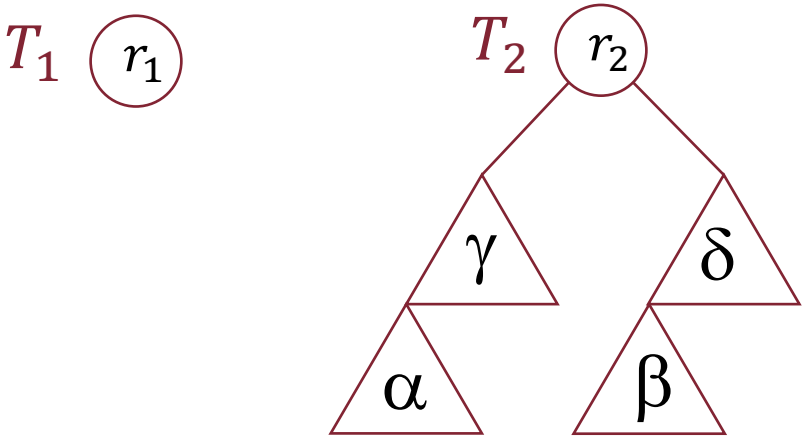
IDEE

- Si potrebbe pensare di inserire in uno dei due alberi tutte le chiavi dell'altro, visitandone uno e chiamando per ogni nodo visitato la `ABR_insert()` sull'altro albero.
- Questa soluzione però costa decisamente troppo, perché un costo lineare nell'altezza va speso per ciascun inserimento.
- Possiamo invece sfruttare il fatto che conosciamo le relazioni di grandezza fra i sottoalberi.

Esercizio 5 (3) – compito del 11/6/2015

SOLUZIONE (1)

1. Troviamo il minimo di γ e gli appendiamo α, costo $O(h_2)$;
2. troviamo il minimo di δ e gli appendiamo β, costo $O(h_2)$;

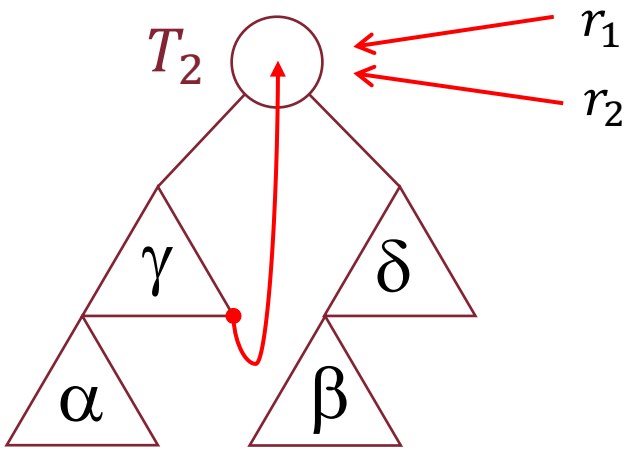


3. Ora però rimangono da fare due cose:
   - sistemare r_2 (perché non sappiamo come si colloca rispetto alle chiavi β);
   - Inserire r_1 che è rimasto fuori.

Esercizio 5 (4) – compito del 11/6/2015

SOLUZIONE (2)

1. Salviamo r_2 in una variabile d'appoggio, costo *Θ(1)*;
2. troviamo il massimo di γ, lo copiamo nella radice ed eliminiamo il nodo che lo contiene, costo: *O(h_2)*;
3. Inseriamo r_1 ed r_2 in T_2 con `ABR_insert()`, costo: *O(h_1+h_2)*



Corso di laurea in Matematica
Insegnamento di Informatica generale
Lezioni in modalità mista o a distanza

Esercizi assegnati in vecchi scritti - 2

Giancarlo Bongiovanni
Ivano Salvo

SAPIENZA
UNIVERSITÀ DI ROMA

Queste dispense sono state realizzate sulla base delle slide
preparate da T. Calamoneri e G. Bongiovanni
per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

Esercizio 1 (1) – compito del 16/6/2020 mattina

Si consideri il problema di determinare gli m elementi più grandi in un array di n elementi e si confrontino dal punto di vista del tempo di esecuzione asintotico i due algoritmi qui sotto proposti:

1. ***Algoritmo A**. Si ordinano gli n elementi e si prendono i primi m;*
2. ***Algoritmo B**. Si costruisce un max-heap dall'array e poi si esegue per m volte l'estrazione del massimo.*

Si diano i tempi di esecuzione dei due algoritmi, dettagliando i ragionamenti fatti.

Esercizio 1 (2) – compito del 16/6/2020 mattina

Algoritmo A. Si ordinano gli n elementi e si prendono i primi m.

Ordinare il vettore costa Θ(n log n) utilizzando un algoritmo efficiente, modificato per ordinare al contrario; prelevare i primi m elementi costa Θ(m).

Poiché n ≥ m il costo è Θ(n log n).

Algoritmo B. Si costruisce un max-heap dall'array e poi si esegue per m volte l'estrazione del massimo.

Costruire il max-heap costa O(n); prelevare il massimo per m volte e ricostruire il nuovo heap, privato del massimo, costa

$$O(\sum_{i=1}^{m} \log(n-i)) < O(m \log n) \leq O(n \log n).$$

Dunque l'algoritmo B risulta meno oneroso.

Esercizio 2 (1) – compito del 16/6/2020 mattina

Si mostrino gli alberi binari di ricerca che risultano dalla ripetuta esecuzione dell'algoritmo di inserimento negli ABR, a partire dall'albero vuoto, quando i nodi da inserire contengono, nell'ordine, i seguenti dati:

- *1 3 5 7 9*
- *5 9 1 3 7*

Si motivi la risposta.

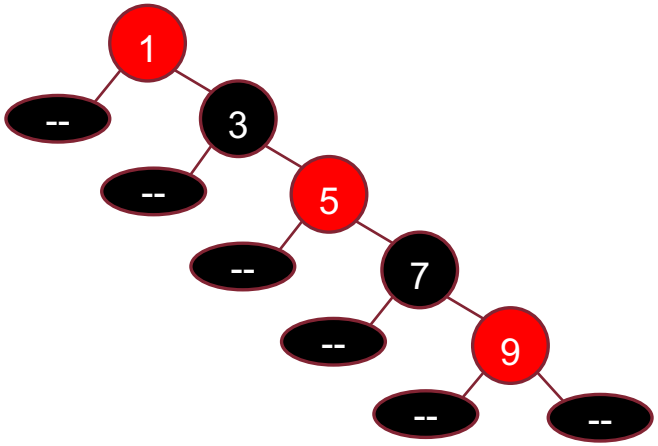
E' possibile che uno od entrambi gli alberi rappresentati rappresentino degli alberi rosso-neri (in cui, evidentemente, risultano nascosti i colori e le foglie fittizie)?

Si motivi la risposta.

Esercizio 2 (2) – compito del 16/6/2020 mattina

Inserimento di 1 3 5 7 9:

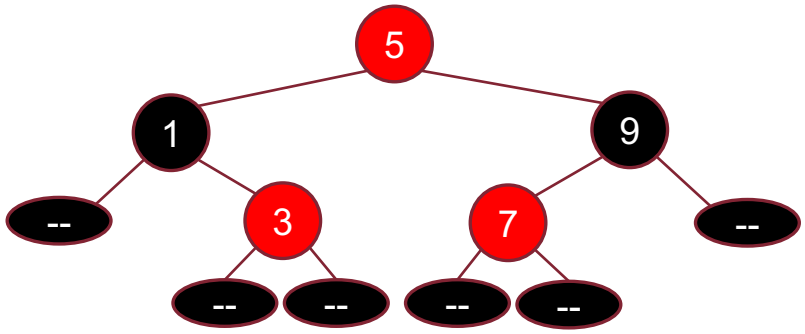
L'ABR risultante è degenere e dunque, anche se per n=5 la sua altezza (pari a 4) non è maggiore di 2 log 6 = 5,17 non può essere un R&B perché si viola la proprietà dello stesso numero di nodi neri su tutti i cammini, anche colorando di rosso il massimo numero di nodi possibile.



Esercizio 2 (3) – compito del 16/6/2020 mattina

Inserimento di 5 9 1 3 7:

L'ABR risultante può essere un R&B perché è possibile colorare i nodi in modo da non violare nessuna delle proprietà:



Esercizio 3 – compito del 16/6/2020 pomeriggio

Qual'è l'algoritmo migliore da usare per riordinare un array contenente q valori che rappresentano alcune potenze di 3 (eventualmente ripetute), nell'intervallo incluso tra 3^0 3^r, con esponente intero e r = O(q) nel modo più efficiente possibile? Si motivi la risposta.

Gli esponenti dei valori da ordinare sono interi compresi in un intervallo da zero a O(q), quindi si può usare il Counting sort calcolando il $\log_3$ di ogni valore prima di conteggiarlo nel vettore C.

Al momento di ricopiare un valore nel vettore finale lo si calcola come $3^{C[i]}$.

Poiché il calcolo del logaritmo e l'elevazione a potenza sono operazioni elementari, il costo è Θ(q).

Esercizio 4 (1) – compito del 16/6/2020 pomeriggio

*Dati due numeri interi x ed y, definiamo la loro **distanza** come dist(x; y) = |x-y|.*

Sia dato un albero binario di ricerca T, contenente chiavi intere, memorizzato mediante record e puntatori.

Si progetti un algoritmo il più efficiente possibile che determini le chiavi di T aventi distanza massima e se ne calcoli il costo computazionale.

Si discuta brevemente sul modo in cui (eventualmente) cambiano l'algoritmo ed il costo computazionale nel caso in cui T sia un albero rosso-nero.

Esercizio 4 (2) – compito del 16/6/2020 pomeriggio

In un ABR le chiavi a distanza massima sono invariabilmente la chiave minima e quella massima contenute nell'ABR.

Sappiamo che per trovare il minimo in un ABR bisogna scendere a sinistra finché possibile e per trovare il massimo bisogna scendere a destra finché possibile:

```
Funzione MinMax(p: puntatore radice)
   if (p = NULL) return {NULL,NULL}
   p_min ← p; p_max ← p
   while(left[p_min]≠NULL)
      p_min ← left[p_min]
   while(right[p_max]≠NULL)
      p_max ← right[p_max]
   return {p_min, p_max}
```

Esercizio 4 (3) – compito del 16/6/2020 pomeriggio

La funzione ha costo O(h), dove h è l'altezza dell'albero.

In un generico ABR si ha h = O(n), mentre in un R&B si ha che h = O(log n).

Per un R&B l'algoritmo non cambia nella sostanza (ma il test nei due **`while`** deve essere modificato: al posto di **`NULL`** si deve verificare che il figlio non sia una foglia fittizia).

Quello che cambia è il costo computazionale, che scende da O(n) a O(log n).

Esercizio 5 (1) – compito del 7/7/2020

Dato un albero binario qualunque T di fissata altezza h, dire qual è il minimo e il massimo numero di nodi n se:

a) T è un max-heap;

b) T è un albero binario di ricerca.

Dato un albero binario qualunque S con n nodi, dire qual'è il valore massimo e minimo dell'altezza h se:

c) S è un albero qualunque;

d) S è un albero Red & Black.

Esercizio 5 (2) – compito del 7/7/2020

a) T è un max-heap di altezza h
- n minimo: 2^h
- n massimo: $2^{h+1} - 1$

b) T è un albero binario di ricerca di altezza h
- n minimo: h + 1
- n massimo: $2^{h+1} - 1$

c) S è un albero binario qualunque di n nodi
- h minima: log((n + 1)/2)
- h massima: n - 1

d) S è un albero Red & Black di n nodi (veri)
- h minima: log((n + 1)/2)
- h massima: 2 log(n + 1)