DIETI. UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica

CORSO DI ALGORITMI E STRUTTURE DATI

Prof. ROBERTO PIETRANTUONO

Homeworks set #1

Istruzioni

Si prepari un file PDF riportante il vostro nome e cognome (massimo 2 studenti). Quando è richiesto di fornire un algoritmo, si alleghi un file editabile (ad esempio, .txt, .doc) riportante l'algoritmo in un linguaggio a scelta, corredato da almeno tre casi di test. Laddove opportuno, si fornisca una breve descrizione della soluzione: l'obiettivo non è solo eseguire l'esercizio e riportare il risultato, ma far comprendere lo svolgimento.

Esercizio 1.1. Notazione asintotica e crescita delle funzioni

Per ogni gruppo di funzioni, ordina le funzioni in ordine crescente di complessità asintotica (big-O):

1)
f1(n) = $n^{0.999999} \log n$
f2(n) = 10000000n
f3(n) = 1.000001^n
f4(n) = n^2

2)
f1(n) = $2^{2^{1000000}}$
f2(n) = 2^{10000n}
f3(n) = $\binom{n}{2}$
f4(n) = $n\sqrt{n}$

3)
f1(n) = $n^{\sqrt{n}}$
f2(n) = 2^n
f3(n) = $n^{10} \cdot 2^{n/2}$
f4(n) = $\sum_{i=1}^{n}(i+1)$

Esercizio 1.2. Notazione asintotica e crescita delle funzioni

Per ognuna delle seguenti funzioni si determini se f(n) = O(g(n)), g(n) = O(f(n)) o entrambe.

- f(n) = $(n^2 - n)/2$ g(n) = 6n
- f(n) = $n+2\sqrt{n}$ g(n) = n^2
- f(n) = nlogn g(n) = $n\sqrt{n}/2$
- f(n) = n+logn g(n) = $\sqrt{n}$

- $f(n) = 2(\log n)^2$ $g(n) = \log n+1$
- $f(n) = 4n\log n+n$ $g(n)=(n^2 - n)/2$

Esercizio 1.3. Notazione asintotica e crescita delle funzioni

Si indichi se le seguenti affermazioni sono vere o false
- $2^{n+1} = O(2^n)$
- $2^{2n} = O(2^n)$

Esercizio 1.4. Ricorrenze
Fornire il limite inferiore e superiore per *T(n)* nella seguente ricorrenza, usando il metodo dell'albero delle ricorrenze ed il teorema dell'esperto se applicabile. Si fornisca il limite più stretto possibile giustificando la risposta.

$T(n)=2T(n/2)+O(\sqrt{n})$
$T(n)=T(\sqrt{n}) + \Theta(\log \log n)$ (suggerimento: si operi una sostituzione di variabili)
$T(n)=10T(n/3)+17n^{1.2}$

Utilizzando l'albero di ricorsione, dimostrate che la soluzione della ricorrenza
$T(n) = T(n/3) + T(2n/3) + cn$, dove c è una costante, è $\Omega(n\log n)$

Problema 1.1
Sia data una lista di *2n* nomi memorizzati in una *Linked List.* Si implementi un algoritmo con complessità O(n) che modifichi la lista in modo tale da invertire l'ordine degli elementi della seconda metà della lista (dunque, in modo tal che l'(n+1)-esimo elemento diventi l'ultimo elemento (posizione 2n), l''(n+2)-esimo diventi il penultimo e così via, l'ultimo elemento diventi l'(n+1)-esimo. L'algoritmo non deve creare nuovi nodi nella lista né istanziare nuove strutture dati.

Si alleghi al PDF un file editabile riportante l'implementazione in un linguaggio a scelta, corredato da almeno tre casi di test (dunque, una sequenza di stringhe in questo caso, con il corrispondente output atteso)

Problema 1.2
Sia data una sequenza di *n* numeri interi, memorizzata in un vettore A. Si implementi un algoritmo che esegua in tempo O(*n*log*n*) per determinare il numero di elementi minori uguali dell'elemento i-esimo, riportando il risultato corrispondente in un secondo vettore B (sempre di dimensione *n*) alla posizione *i*.
Si alleghi al PDF un file editabile riportante l'implementazione in un linguaggio a scelta, corredato da almeno tre casi di test (dunque, una sequenza di numeri iteri in questo caso, con il corrispondente output atteso)

Problema 1.3
Descrizione.
Se inseriamo un insieme di *n* elementi in un albero di ricerca binario (*binary search tree*, BST) utilizzando TREE-INSERT, l'albero risultante potrebbe essere molto sbilanciato. Tuttavia, ci si aspetta che i BST costruiti casualmente siano bilanciati (ossia ha un'altezza attesa O(lg n)). Pertanto, se vogliamo costruire un BST con altezza attesa O(lg n) per un insieme fisso di elementi, potremmo permutare casualmente gli elementi e quindi inserirli in quell'ordine nell'albero.

Cosa succede se non abbiamo tutti gli elementi a disposizione in una sola volta? Se riceviamo gli elementi uno alla volta, possiamo ancora costruire casualmente un albero di ricerca binario da essi? Nel seguito è proposta una struttura dati che risponde affermativamente a questa domanda. Un ***treap*** è un albero binario di ricerca che usa una strategia diversa per ordinare i nodi. Ogni elemento x nell'albero ha una chiave *key*[x]. Inoltre, assegniamo *priority*[x], che è un numero casuale scelto indipendentemente per ogni x. Assumiamo che tutte le priorità siano distinte e anche che tutte le chiavi siano distinte. I nodi del *treap* sono ordinati in modo che (1) le chiavi obbediscano alla proprietà del *binary search tree* e (2) le priorità obbediscano alla proprietà *min-heap order* dell'heap. In altre parole:

- se *v* è un figlio sinistro di *u*, allora *key*[*v*] < *key*[*u*];
- se *v* è un figlio destro di *u*, allora *key*[*v*] > *key*[*u*]; e
- se *v* è un figlio di *u*, allora *priority*(*v*) > *priority*(*u*).

(Questa combinazione di proprietà è il motivo per cui l'albero è chiamato "***treap***": ha caratteristiche sia di un albero di ricerca binario che di un *heap*)

È utile pensare ai *treaps* in questo modo: supponiamo di inserire i nodi $x_1, x_2, \ldots x_n$, ciascuno con una chiave associata, in un *treap* in ordine arbitrario. Quindi il *treap* risultante è l'albero che si sarebbe formato se i nodi fossero stati inseriti in un normale albero binario di ricerca nell'ordine dato dalle loro priorità (scelte casualmente). In altre parole, *priority*[x_i] < *priority*[x_j] significa che x_i è effettivamente inserito prima di x_j.

Per inserire un nuovo nodo *x* in un *treap* esistente, si assegna dapprima ad *x* una priorità casuale *priority*[*x*]. Quindi si chiama l'algoritmo di inserimento, che chiameremo TREAP-INSERT, il cui funzionamento è illustrato nella Figura 1.

Quesito. **Fornire il codice della procedura TREAP-INSERT** in un linguaggio a scelta, allegando un file editabile**.** *Suggerimento: effettuare il consueto inserimento del BST, ed eseguire le rotazioni per ripristinare la proprietà del min-heap* (*min-heap order*)*).*

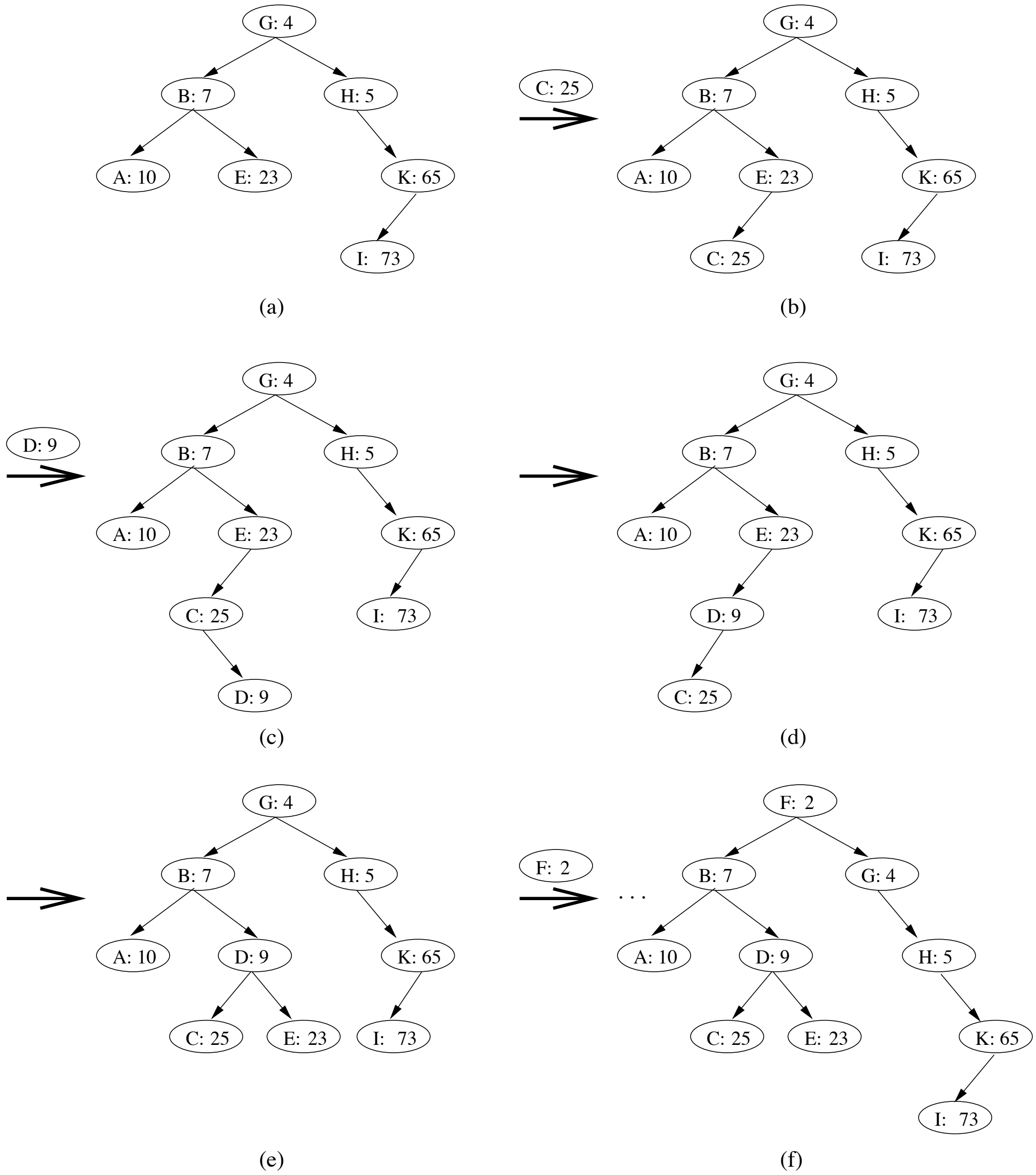

Figura 1. Operazioni di TREAP-INSERT. Ogni nodo è etichettato con key[x] : priority[x]. a) *Treap* prima dell'inserimento; b) *Treap* dopo aver inserito un nodo con chiave C e priorità 25; c-d) stadi intermedi quando si inserisce D (priorità 9); e) *Treap* dopo il completamento dell'inserimento delle parti c)-d); f) *Treap* dopo l'inserimento di F (priorità 2)