Algoritmi e Strutture Dati
4 Luglio 2022

Cognome .... **Nome** .... **Matricola** ....

Note

1. La leggibilità è un prerequisito: parti difficili da leggere potranno essere ignorate.
2. Quando si presenta un algoritmo è fondamentale spiegare l'idea soggiacente e motivarne la correttezza.
3. L'efficienza e l'aderenza alla traccia sono criteri di valutazione delle soluzioni proposte.
4. Si consegnano tutti i fogli, con nome, cognome, matricola e l'indicazione *bella copia* o *brutta copia*.

Domande

Domanda A (7 punti) Dare la definizione della classe $\Theta(f(n))$. Mostrare che la ricorrenza

$$T(n) = \frac{1}{4}\,T(n-1) + 3n^2$$

ha soluzione in $\Theta(n^2)$.

Soluzione: Per provare che $T(n) = O(n^2)$ dobbiamo dimostrare che $T(n) \leq cn^2$, per un'opportuna costante $c > 0$. Procediamo per induzione:

$$\begin{aligned} T(n) &= 1/4\,T(n-1) + 3n^2 \\ &\leq 1/4\,c(n-1)^2 + 3n^2 \\ &\leq 1/4\,cn^2 + 3n^2 \\ &\leq cn^2 \end{aligned}$$

dove, per la validità dell'ultima disuguaglianza $1/4\,cn^2 + 3n^2 \leq cn^2$, occorre che

$$3/4\,cn^2 \geq 3n^2$$

ovvero, $c \geq 4$, con n qualunque.

Per provare $T(n) = \Omega(n^2)$ dobbiamo dimostrare che $T(n) \geq dn^2$, per un'opportuna costante $d > 0$. La prova per induzione non usa neppure l'ipotesi induttiva. Infatti

$$T(n) = 1/4\,T(n-1) + 3n^2 \geq dn^2$$

purché $d \leq 3$.

Domanda B (6 punti) Si calcoli la lunghezza della longest common subsequence (LCS) tra le stringhe *armo* e *oro*, calcolando tutta la tabella $L[i, j]$ delle lunghezze delle LCS sui prefissi usando l'algoritmo visto in classe.

Soluzione: Si ottiene

		o	r	o
	0	0	0	0
a	0	0	0	0
r	0	0	1	1
m	0	0	1	1
o	0	1	1	2

La lunghezza della longest common subsequence tra *armo* e *oro* è quindi 2.

Esercizi

Esercizio 1 (9 punti) Realizzare una funzione `strongBST(T)` che dato un albero binario `T` con chiavi numeriche non negative, verifica se, per ogni nodo, la chiave è maggiore uguale del doppio di ogni chiave nel sottoalbero sinistro e minore o uguale della metà di ogni chiave nel sottoalbero destro, e ritorna conseguentemente un valore booleano (la radice dell'albero è `T.root` e ogni nodo `x` ha i campi `x.left` `x.right` e `x.key`). Valutarne la complessità.

Soluzione:

```
strongBST(T)
   s, m, M = strongBST(T.root)
   return v


# strongBSTRec(x):
# verifica se il sottoalbero radicato in x e' uno strongBST e ritorna tre valori:
# - un booleano (che indica l'esito della verifica)
# - il massimo delle chiavi nel sottoalbero sx
# - il minimo delle chiavi nel sottoalbero dx

strongBSTRec(x)

if x = nil
   return true, 0, +infty
else
   # ispeziono i sottoalberi sinistro e destro
   sl, ml, Ml = strongBSTRec(x.left)
   sr, mr, Mr = strongBSTRec(x.right)

   # se la chiave del nodo in esame e' maggiore o uguale del doppio
   # del massimo delle chiavi nel sottoalbero sinistro (quindi di
   # tutte le chiavi nel sottoalbero sinistro) e minore o uguale della
   # meta' del minimo delle chiavi nel sottoalbero sinistro (quindi di
   # tutte le chiavi nel sottoalbero destro) delle chiavi dei
   # discendenti, ritorna true
   s = (x.key >= 2*Ml) and (x.key <= 1/2 mr)

   # calcola il minimo e il massimo delle chiavi nel sottoalbero radicato in x
   m =   min { ml, mr, x.key }
   M =   max { Ml, Mr, x.key }

   return s, m, M
```

Si tratta di una visita e quindi la complessità è $\Theta(n)$.

Esercizio 2 (10 punti) Dato un insieme di n numeri reali positivi e distinti $S = \{a_i, a_2, \ldots, a_n\}$, con $0 < a_i < a_j < 1$ per $1 \le i < j \le n$, un *(2,1)-boxing* di S è una partizione $P = \{S_1, S_2, \ldots, S_k\}$ di S in k

sottoinsiemi (cioè, $\bigcup_{j=1}^{k} S_j = S$ e $S_r \cap S_t = \emptyset, 1 \leq r \neq t \leq k$) che soddisfa inoltre i seguenti vincoli:

$$|S_j| \leq 2 \qquad \text{e} \qquad \sum_{a \in S_j} a \leq 1, \quad 1 \leq j \leq k.$$

In altre parole, ogni sottoinsieme contiene al più due valori la cui somma è al più uno. Dato S, si vuole determinare un (2,1)-boxing che minimizza il numero di sottoinsiemi della partizione.

1. Scrivere il codice di un algoritmo greedy che restituisce un (2,1)-boxing ottimo in tempo lineare. (Suggerimento: si creino i sottoinsiemi in modo opportuno basandosi sulla sequenza ordinata.)
2. Si enunci la proprietà di scelta greedy per l'algoritmo sviluppato al punto precedente e la si dimostri, cioè si dimostri che esiste sempre una soluzione ottima che contiene la scelta greedy.

Soluzione:

1. L'idea è provare ad accoppiare il numero più piccolo (a_1) con quello più grande (a_n). Se la loro somma è al massimo 1, allora $S_1 = \{a_1, a_n\}$, altrimenti $S_1 = \{a_n\}$. Poi si procede analogamente sul sottoproblema $S \setminus S_1$.

```
(2,1)-BOXING(S)
n <- |S|
P <- empty_set
first <- 1
last <- n
while (first <= last)
   if (first < last) and a_first + a_last <= 1 then
      P <- P U {{a_first, a_last}}
      first <- first + 1
   else
      P <- P U {{a_last}}
   last <- last - 1
return P
```

   Questo algoritmo scansiona ogni elemento una sola volta, quindi la sua complessità è lineare.

2. La scelta greedy è $\{a_1, a_n\}$ se $n > 1$ e $a_1 + a_n \leq 1$, altrimenti $\{a_n\}$. Ora dimostriamo che esiste sempre una soluzione ottima che contiene la scelta greedy. I casi $n = 1$ e $a_1 + a_n > 1$ sono banali, visto che in questi casi ogni soluzione ammissibile deve contenere il sottoinsieme $\{a_n\}$. Quindi assumiamo che la scelta greedy sia $\{a_1, a_n\}$. Consideriamo una qualsiasi soluzione ottima dove a_1 e a_n non sono accoppiati nello stesso sottoinsieme. Quindi, esistono due sottoinsiemi S_1 e S_2, con $a_1 \in S_1$ e $a_n \in S_2$. Sostituiamo questi due sottoinsiemi con $S'_1 = \{a_1, a_n\}$ (cioè, la scelta greedy) e $S'_2 = S_1 \cup S_2 \setminus \{a_1, a_n\}$. $|S'_2| \leq 2$ e, se $|S'_2| = 2$, allora $S'_2 = \{a_s, a_t\}$ con $a_s \in S_1$ e $a_t \in S_2$. Siccome a_t era precedentemente accoppiato con a_n, a maggior ragione può essere accoppiato con $a_s < a_n$, quindi la nuova soluzione così creata è ammissibile e ancora ottima.