Prima prova pratica in itinere

Ingegneria degli Algoritmi 2018/2019

Ovidiu Daniel Barba
ovidiudaniel.barba@alumni.uniroma2.eu

Luca Pepè Sciarria
luca.pepesciarria@alumni.uniroma2.eu

14 Novembre 2018

1 Modalità di svolgimento

Il progetto può essere svolto singolarmento oppure in gruppi da massimo 3 persone (consigliato gruppo di 2 persone).

È altamente **sconsigliato** di copiare codice reperito on-line o da altri elaborati e di usare librerie esterne, poichè provoca l'annullamento della consegna con conseguente esclusione dalle prove in itinere.
È invece **consigliato** di usare e manipolare il codice visto a lezione e disponibile sul sito della parte pratica del corso.

2 Scelta del progetto

Una volta individuato il gruppo, definiamo s come la somma delle matricole dei componenti del gruppo. Il progetto assegnato è dato dal valore di $s \, mod \, 2 + 1$. Per esempio, se il gruppo è composta da 2 persone con matricola 0244962 e 0246425, $s = 244962 + 246425 = 491387$ e il progetto assegnato è il secondo poichè $s \, mod \, 2 + 1 = 491387 \, mod \, 2 + 1 = 1 + 1 = 2$.
Una volta individuato il progetto da svolgere, compilare il form (una sola volta per gruppo) entro il **1 Dicembre 2018**.

3 Materiale da consegnare

- Breve relazione in formato pdf contenente:
  - scelte implementative
  - risultati sperimentali sia in forma tabulare che grafica con unità di misura appropriate
  - commento dei risultati ottenuti

- Codice sorgente:
  - implementazione richiesta nella traccia
  - esempio di uso dell'algoritmo o struttura dati implementata
  - esperimenti effettuati

4 Consegna

Tutto il materiale richiesto dovrà essere caricato su un repository online (ad esempio GitHub, Dropbox, Google Drive, ecc), il cui link dovrà essere inviato al tutor di riferimento del progetto (Progetto 1 - Barba, Progetto 2 - Pepè Sciarria) entro il **14 Dicembre 2018**.

5 Progetti

5.1 Progetto 1 [*Barba*]

Dati i seguenti parametri appartenenti ai numeri interi $\mathbb{Z}$:

- min e max con $max > min$
- $r = 6$
- $b > r$ con $(max - min)$ multiplo di b,

definiamo $d = |\frac{max-min}{b}|$ e gli insiemi $B_i = \{n \in \mathbb{Z} \,|\, min+i \cdot b \leq n < min+(i+1) \cdot b\}$ con $i \in \{0, ..., d-1\}$, $B_d = \{n \in \mathbb{Z} \,|\, n < min\}$ e $B_{d+1} = \{n \in \mathbb{Z} \,|\, n \geq max\}$. Dalle definizioni precedenti si nota che $\bigcup_{i=0}^{d+1} B_i = \mathbb{Z}$ e $B_i \cap B_j = \emptyset \ \forall i, j \in \{0, ..., d+1\}$ con $i \neq j$, quindi $\{B_i\}_{i=0}^{d+1}$ costituisce una partizione di $\mathbb{Z}$.
Implementare una struttura dati che gestisce coppie di tipo $(key, value)$ (per semplicità $key \in \mathbb{Z}$) e ha le seguenti caratteristiche:

- un array v di dimensione $d+2$ dove ogni cella è indicizzata da $i \in \{0, ..., d+1\}$ e punta ad una struttura dati nota (lista concatenata oppure albero AVL) con n_i elementi
  - $v[i]$ punta ad una lista concatenata $\iff n_i < r$
  - $v[i]$ punta ad un albero AVL $\iff n_i \geq r$
- implementa i metodi `insert(key, value)`, `delete(key)`, `search(key)` del `Dictionary`
  - le operazioni vengono eseguite sulla struttura dati puntata da $v[i] \iff key \in B_i$ con $i \in \{0, ..., d+1\}$

N.B.: le operazioni `insert` e `delete` modificano il numero n_i di elementi presenti nella struttura dati puntata da $v[i]$. Tale struttura dati, ove necessario, deve essere cambiata di conseguenza. Per esempio, se nella lista concatenata puntata da $v[1]$ ci sono 5 elementi ($n_1 < r$) e ne viene aggiunto un altro, n_1 è maggiore o uguale a r e la lista viene trasformata in un albero AVL con gli stessi elementi. Adesso $v[1]$ punterà all'albero e non più alla lista concatenata. Se invece dall'albero di $v[1]$ rimuoviamo un elemento arriviamo a $n_1 = 5$ e l'albero viene trasformato in una lista concatenata.

Infine, confrontare sperimentalmente il tempo di esecuzioni medio dei metodi `search` e `delete` della struttura dati descritta precedentemente e del `dictionary` di Python al variare del numero n di elementi presenti.

5.2 Progetto 2 [*Pepè Sciarria*]

Questo è un progetto sugli algoritmi di ordinamento e selezione. L'obiettivo è quello di implementare una nuova versione dell'algoritmo `select` e di modificare l'algoritmo `quickSort` in cui la scelta del pivot, invece di avvenire in modo random, avviene tramite un algoritmo di selezione. Il nuovo algoritmo di selezione (`sampleMedianSelect`), basato su quello di Floyd e Rivest, sceglie il pivot su cui effettuare la partizione nel seguente modo:

- Sceglie un sottoinsieme V di m elementi in modo random
- Seleziona il mediano di V e lo usa come pivot

Come si può notare, questo algoritmo è una versione più generale dell'algoritmo deterministico `select` in cui il sottinsieme V è formato dai mediani delle quintuple. La scelta del parametro m è lasciata allo studente.
Si richiede inoltre di confrontare sperimentalmente, al variare della dimensione della lista in input, il tempo di esecuzione delle varie varianti (con select randomizzata, deterministica e `sampleMedianSelect`) del nuovo algoritmo quickSort con

- `quickSort` classico
- gli altri algoritmi di ordinamento visti e implementati a lezione
- il metodo `sort()` di Python