Progetto Algoritmi e Strutture Dati
Algoritmi di selezione

Riccardo Rossetto, Mattia Monti, Daniele Gnudi

10 maggio 2022

1 Introduzione

Per la realizzazione del progetto del corso di "Algoritmi e Strutture Dati", abbiamo trattato lo studio, l'implementazione e analisi empirica del tempo medio di esecuzione di tre algoritmi di selezione:

"quick select", "heap select" e "median-of-medians select". Questi sono stati, dunque, adeguatamente modificati per rispettare i requisiti della consegna:

dato in input un array di numeri interi, non ordinato, e un indice k, restituire in output l'elemento in posizione k se l'array venisse ordinato. Sono quindi state effettuate diverse misurazioni su input generati in maniera pseudocasuale con le due seguenti modalità: ???????dimensione del vettore variabile e indice fisso, e dimensione array fissata con indice variabile, concentrandoci in particolare su tre casi di studio che verranno descritti e analizzati in seguito.

2 Struttura del Programma

- ""
- ""
- ""

3 Scelte Implementative

Di seguito esponiamo le principali scelte implementative adottate nella realizzazione del codice per la misurazione empirica dei tempi di esecuzione, riportando brevemente an- che i principi di funzionamento alla base degli algoritmi di selezione. Inoltre, verranno sottolineate le differenze che caratterizzano le varianti di uno stesso processo di selezione.

3.1 Aspetti generali

?????

3.2 Generatore test

??????

3.3 Clock e tempo minimo di esecuzione

Prima di poter procedere al calcolo vero e proprio dei tempi di esecuzione, abbiamo dovuto ricavare la risoluzione del clock di sistema, indispensabile per il controllo degli errori di misura. Pertanto, è stata usata la funzione *clock_gettime* della libreria time.h che, grazie all'opzione CLOCK_MONOTONIC , permette l'utilizzo di un orologio di tipo monotono. Ottenuta dunque la risoluzione, ricavata dalla mediana di un buon numero di misurazioni, è possibile calcolare un tempo minimo di esecuzione che assicuri l'accuratezza richiesta. In particolare, si procede ad applicare la seguente formula $T_{min} \leq res \times (1 + \frac{1}{\epsilon})$ così da garantire la massimizzazione dell'errore relativo . È importante specificare che l'errore usato per i calcoli corrisponde in realtà alla metà dell'errore relativo richiesto. Ciò sarà necessario per rispettare i limiti indicati nella consegna quando andremo ad effettuare i calcoli finali.

3.4 Rilevazione tempi di esecuzione

All'interno dei due cicli for innestati della procedura principale, viene rilevato il tempo di esecuzione di uno degli algoritmi su un vettore di dimensione e seme di generazione prestabiliti. La misurazione vera e propria è effettuata dalla funzione *getExecutionTimes*. Questa, in base agli argomenti ricevuti, provvede a ripetere il processo di misurazione per il numero di volte richiesto, restituendo alla fine la serie dei tempi registrarti. Ogni iterazione, in seguito alla corretta generazione di un array di input, consiste nella copiatura di quest'ultimo e nell'esecuzione dell'algoritmo di selezione, ripetendo eventualmente questi due passaggi fino al raggiungimento del tempo minimo di esecuzione precedentemente discusso o al superamento del tempo limite imposto dall'utente. Questo è un'indicazione del tempo massimo che l'utente è disposto ad aspettare per l'intero processo di misurazione sullo specifico input e ha lo scopo di evitare lunghe attese nel caso di algoritmi inefficienti o errati. In particolare, nel caso in cui questa limitazione venga superata, la procedura restituirà il valore NULL .

3.5 Tempo generazione vettore

3.6 Calcoli finali ed outputs

4 Algoritmi

??????

4.1 QuickSelect

L'algoritmo QuickSelect si basa sull'algoritmo di ordinamento QuickSort ogni chiamata ricorsiva, sull'intervallo definito tra i e j del vettore fornito in input, termina in tempo costante ogni volta che il parametro k, relativo alla posizione dell'elemento da cercare, non sia contenuto tra gli indici i e j. L'algoritmo QuickSelect sfrutta la funzione *partition* che utilizza l'ultimo elemento dell'intervallo come perno per partizionare l'array in due sottoarray tali che ogni elemento del primo sia minore o uguale ad ogni elemento del secondo. Quindi procede ricorsivamente, richiamando la procedura quickSortSelection sul sotto-array contenente l'elemento in posizione k. Esso terminerà quando la procedura partition restituirà l'indice cercato, ovvero k. L'algoritmo ha complessità temporale asintotica $\Theta(n^2)$ nel caso pessimo e $O(n)$ nel caso medio, dove n è il numero di elementi dell'array.