Analisi empirica degli algoritmi di ordinamento

Graziano Francesco [1] , Ongaro Michele [2] , Petri Riccardo [3] e Ungaro Marco [4]

Università degli Studi di Udine, Dipartimento di Matematica e Informatica

A.A. 2024-2025

[1]Email: graziano.francesco@spes.uniud.it, Matricola: 166680
[2]Email: ongaro.michele@spes.uniud.it, Matricola: 168049
[3]Email: petri.riccardo@spes.uniud.it, Matricola: 167623
[4]Email: ungaro.marco@spes.uniud.it, Matricola: 168934

Contenuti

Capitolo 1

Introduzione

Il progetto richiede l'implementazione di quattro algoritmi di ordinamento per array interi di dimensioni variabili. Gli algoritmi che andremo ad analizzare sono il Counting Sort, il Quick Sort, il Quick Sort 3 way e il Radix Sort (algoritmo a scelta). Oltre alla corretta implementazione viene richiesto di effettuare un analisi empirica dei tempi medi di esecuzione degli algorimti al variare della dimensione dell'array e del range dei valori interi. Per stimare i tempi di esecuzione di questi algoritmi garanantendo un errore relativo massimo pari a 0.001 adotteremo le seguenti metodologie:

- Utilizzeremo un clock di sistema monotono per garantire precisione nelle misurazioni (ad esempio, `perf_counter()` del modulo *time* in Python);
- Andremo a generare almeno 100 campioni per ciascun grafico, con i valori dei parametri (dimensione dell'array n e intervallo dei valori m) distribuiti secondo una progressione geometrica;
- Effettueremo più esecuzioni per ogni campione, per stimare in modo affidabile il tempo medio di esecuzione e, eventualmente, il relativo errore.

Dopo aver stimato i tempi di esecuzione per ciascun algoritmo, risulterà interessante confrontare i grafici ottenuti per analizzare il comportamento degli algoritmi in diverse situazioni, come il caso peggiore, quello migliore o in quello medio.

Da tutto questo potremmo ottenere una verifica empirica dell'andamento asintotico dei tempi di esecuzione di ogni algoritmo.

Capitolo 2

Counting Sort

Counting Sort è un algoritmo di ordinamento non comparativo, cioè che non ordina gli elementi confrontandoli tra loro come fanno invece altri algoritmi classici come QuickSort, Merge Sort o Bubble Sort. Invece, conta quante occorrenze di ciascun valore sono presenti nell'array da ordinare e utilizza queste informazioni per posizionare gli elementi nell'array ordinato. Passaggi principali che effettua l'algoritmo:

1. Determinazione del range: Identifica il valore massimo nell'array in input per determinare la dimensione necessaria dell'array di conteggio.
2. Conteggio delle occorrenze: Crea un array ausiliario (`count`) in cui ciascun indice rappresenta un valore possibile dell'array originale, e si conta quante volte ciascun valore appare.
3. Costruzione dell'array cumulativo: Trasforma l'array di conteggio in un array cumulativo, dove ogni elemento indica la posizione finale di un dato valore nell'array ordinato.
4. Costruzione dell'array ordinato: Itera sull'array originale (in genere in ordine inverso per mantenere la stabilità), e si posiziona ogni elemento nella posizione corretta dell'array di output, decrementando il valore corrispondente nell'array di conteggio.

L'algoritmo Counting Sort è particolarmente efficiente quando il range dei valori da ordinare è limitato rispetto alla dimensione dell'array.

Analisi della complessità

La complessità temporale di Counting Sort è $O(n + k)$, dove:

- n è il numero di elementi nell'array da ordinare;
- k è il valore massimo presente nell'array.

Nel caso in cui si ha $k = O(n)$ allora la complessità diventa $O(n)$

Grafico dei tempi di esecuzione: TODO.

Capitolo 3

Quick Sort

Spiegazione dell'algoritmo di ordinamento
- Analisi della complessità
- Grafico dei tempi di esecuzione

Capitolo 4

Quick Sort 3 way

Capitolo 5

Radix Sort

Capitolo 6

Conclusioni