Insertion Sort e QuickSort

Manuel Cecere Palazzo

2019

Contents

1 Introduzione

Obiettivo di questa relazione è descrivere il comportamento degli algoritmi Insertion Sort e Quick Sort. Particolare attenzione è data al tempo di esecuzione al variare della dimensione dell'input, prendendo in considerazione i casi di vettore casuale, vettore ordinato e ordinato al contrario.

2 Teoria degli Algoritmi

2.1 Insertion Sort

Insertion Sort è un algoritmo di ordinamento che costruisce il vettore finale un elemento alla volta, spostandolo nella corretta posizione.Questo procedimento viene ripetuto fino a quando l'intero vettore non è ordinato.

Questo metodo di ordinamento è associabile al modo più semplice di riordinare le carte di un mazzo, prendendo una carta alla volta.

2.1.1 Costo

In questa relazione consideriamo il costo come la quantità di tempo e spazio utilizzata dall'algoritmo , anche se , come già detto nell'introduzione , studieremo principalmente il tempo impiegato.

Il costo dell'algoritmo Insertion Sort cambia al variare della dimensione dell'array di input ma anche al presentarsi di particolari istanze.

Il **caso ottimo** si presenta quando il vettore è già ordinato. In quel caso il tempo impiegato dall'algoritmo cresce linearmente rispetto alle dimensioni dell'input.

Il **caso peggiore** si presenta con un vettore ordinato al contrario. In tal caso l'algoritmo dovrà effettuare *x-1* passi per ogni iterazione, dove con x indichiamo la dimensione del sottovettore preso in considerazione. Il tempo impiegato a questo punto cresce con velocità quadratica.

Si può dimostrare che il **caso medio** cresce asintoticamente con la stessa velocità del **caso peggiore**.

2.2 Quick Sort

Quick Sort è un algoritmo di ordinamento di tipo ricorsivo. Preso un elemento chiamato **pivot** nell'array questo viene posto nella posizione corretta e si richiama l'algoritmo sui sottoarray alla sinistra e alla destra dell'array, in cui saranno presenti ,rispettivamente, gli elementi minore e quelli maggiori del pivot.

2.2.1 Costo

Il costo dell'algoritmo Quick Sort cambia alla dimensione dell'array in input e alla dimensione dei sottoarray in cui viene diviso.

Il **caso ottimo** si presenta quando la dimensione dei sottoarray sono sempre uguali fra loro. In tal caso il tempo impiegato è una funzione di *nlog n* dove n è la dimensione dell'array di input.

Il **caso peggiore** si presenta con un vettore ordinato o ordinato al contrario. In questo caso la dimensione dei sottoarray sarà *m-1* e *1* dove m è la dimensione del array su cui è chiamato l'algoritmo.Il tempo impiegato a questo punto cresce con velocità quadratica.

Si può dimostrare che il **caso medio** cresce asintoticamente con la stessa velocità del **caso ottimo**.

3 Esperimenti

Andremo ad effettuare una serie di test usando gli algoritmi citati. Gli array in input sono di quattro tipologie: ordinati, ordinati al contrario ,randomizzati e una tipologia costruita ad hoc per essere il caso migliore dell'algoritmo Quick Sort. Saranno generati in dimensione crescente fino ad un massimo per cui l'esecuzione complessiva del programma non superi qualche minuto. Per

ogni dimensione vengono testati 40 array, i tempi vengono registrati e ne viene effettuata la media, per ammortizzare eventuali comportamenti anomali della cpu causati dall'esecuzione di altri programmi

Calcolatore utilizzato

- **Processore:** Intel Core i7-7700 HQ CPU 2.80 GHz, 4 core
- **Sistema operativo:** Windows 10 Home 10.0.18362 64 bit
- **Memoria:** 8GB SDRAM DDR4, 240GB SSD
- **Versione Python:** Python 3.7

4 Documentazione del codice

Il codice è composto delle funzioni che generano le quattro tipologie di array già citate,utilizzando la libreria numpy. Sono ovviamente presenti gli algoritmi di quicksort e insertion sort e due funzioni che effettuano il testing per i due algoritmi. In quest'ultime vengono utilizzate la funzione default-timer dal modulo timeit dalla libreria standard di Python per registrare i tempi e il modulo pyplot dalla libreria matplotlib per tracciare i grafici.

5 Risultati sperimentali e Conclusione

I grafici 1 e 2 confermano le ipotesi teoriche. Le medie dei tempi per ogni dimensione ammortizzano in maniera soddisfacente, permettendoci di notare che la crescita era quella teorizzata nel paragrafo 2. Osservare che il caso medio cresce alla stessa velocità del caso ottimo per il Quick Sort ci porta a prediligere questo per input di grandi dimensioni.

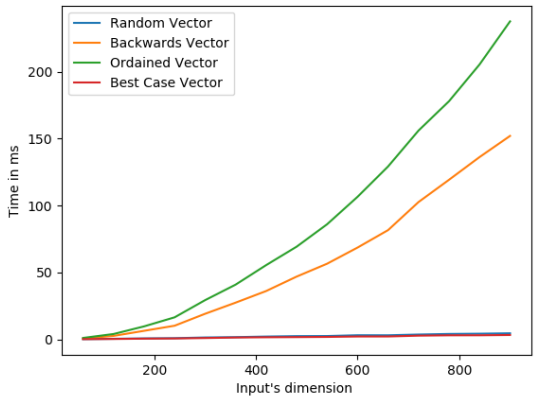

Figure 1: Grafico dei tempi del Quick Sort sulle tipologie di array citate

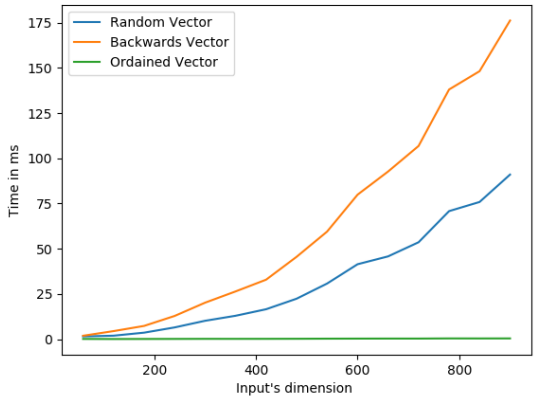

Figure 2: Grafico dei tempi dell'Insertion Sort sulle tipologie di array citate a meno del caso migliore del Quick Sort