ingegneria degli algoritmi

progetto n°2

A cura di

Simone Grillo 0258129

Michele Granatiero 0253496

Ivan Palmieri 0259244

10 Dicembre 2018

Indice:

1. Traccia
2. Richiami Teorici: l’algoritmo quickSort
3. L’algoritmo quickSortPrandom
   1. La scelta dl valore di
   2. Il mediano tra gli , l’algoritmo sampleMedianSelect
4. Grafici e risultati sperimentali

1. Traccia

Questo è un progetto sugli algoritmi di ordinamento e selezione. L’obbiettivo 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 sottoinsieme V `e formato dai mediani delle quintuple. La scelta del parametro m `e 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

2. RICHIAMI TEORICI: L’algoritmo quickSort

L’algoritmo quickSort si basa sulla scelta randomica del pivot per effettuare la partizione. Scegliendo randomicamente il pivot si possono verificare 4 casi:

1. L’elemento scelto è il mediano

2. L’elemento scelto è il massimo

3. L’elemento scelto è il minimo

4. L’elemento scelto non è nessuno dei precedenti

Dall’analisi asintotica possiamo ridurre questi 4 casi a 2:

1. L’elemento scelto è il massimo o il minimo (o molto vicino ad essi)

2. L’elemento scelto è uno di tutti gli altri elementi

Proseguendo con l’analisi asintotica sappiamo che il caso peggiore si verificherà soltanto se il pivot scelto ricadrà nel primo caso per un tempo di esecuzione altrimenti i tempi d’esecuzione del caso migliore o atteso saranno Pertanto il quickSort sfrutta la bassa probabilità di estrarre i valori di massimo o di minimo (o molto vicini ad essi)

3. L’ALGORITMO quickSortPrandom

L’algoritmo quickSortPrandom si propone di ridurre ulteriormente la probabilità di ricadere nel caso peggiore del quickSort, in favore del caso atteso.

Affinché ciò avvenga invece di estrarre un solo elemento, l’estrazione viene reiterata per volte e ne si prende il mediano. Pertanto è cruciale la scelta del valore di poiché più è grande, maggiore sarà la precisione nel trovare il mediano, tuttavia al crescere di aumenta anche il costo computazionale per trovare il mediano stesso, portando al paradosso che per evitare di andare nel caso peggiore, si eseguano operazioni supplementari che facciano perdere il tempo guadagnato per la scelta del pivot ottimale.

Alla luce di ciò il quickSortPrandom è conveniente solo quando è valida la relazione

Però sappiamo che entrambi gli algoritmi impiegano tempo per la partizione, quindi la relazione diventa

Quindi

3.1 LA SCELTA DEL VALORE DI

Nella nostra implementazione del quickSortParandom viene scelto proporzionalmente all’ordine di grandezza della lista ordinare nella funzione recursiveQuickSort.

A questa funzione vengono passati come parametri la lista da ordinare, gli indici su cui poi verrà eseguita la partizione, e dei parametri booleani che determineranno quale variante quickSort eseguire:

Nel caso della variante Prandom, viene calcolata in l’ordine di grandezza della distanza tra gli indici: se il valore è piccolo vorrà dire che la distanza sarà piccola a sua volta. Come minimo abbiamo scelto il valore 3: infatti se , esisteranno al più elementi differenti, cioè 999, ora la probabilità di avere il caso peggiore con singola estrazione sarebbe , quindi basta un molto piccolo, quindi abbiamo scelto 3.

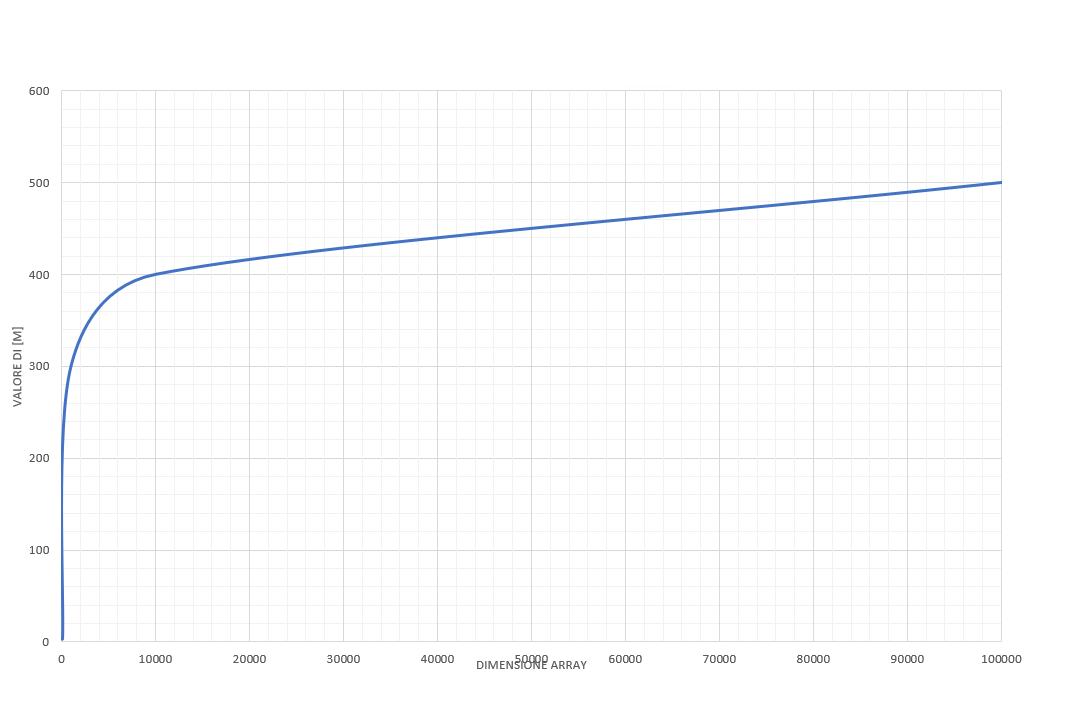
Se invece abbiamo imposto , la costante 100 è stata scelta sperimentalmente basandosi sul concetto che *“maggiore è il valore di , maggiore sarà il tempo del calcolo del mediano”*, e *“maggiore è il campione più preciso sarà il mediano”*, trovando così un valore campione relativamente piccolo rispetto all’ordine di grandezza. 

Grafico dell’andamento di m

3.2 Il mediano tra gli :

l’algoritmo sampleMedianSelect

Una volta stabilito il valore di , esso viene dato come parametro insieme ai valori degli indici (left, right) alla funzione sampleMedianSelect. Quest’ultima crea una sottolista V di tuple (valore, indice elemento lista).

Successivamente si valuta la dimensione di V: verrà usato il trivialSelect, se V è molto piccolo, per restituire l’indice del mediano rispetto alla lista salvandolo nella variabile mid, oppure V verrà ordinato tramite l’algoritmo di ordinamento mergeSort, salvando nella variabile mid il valore di V[][1] (indice rispetto alla lista del mediano).

Una volta trovato l’indice del mediano, si scambiano gli elementi di lista[left] e lista[mid] e si esegue la funzione partition, sfruttando il meccanismo del quickSort deterministico che prende come pivot il primo elemento (quello di posizione left).

In generale il tempo di esecuzione della funzione sampleMedianSelect è

Quindi

Poiché allora:

4 grafici e risultati sperimentali

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DIM. LISTA | selectionSort | inserctionSortDown | inserctionSortUp | bubbleSort |
| 100 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1000 | 0.0343668460 | 0.0249943971 | 0.0312421321 | 0.0140591621 |
| 10000 | 3.3891240119 | 2.7096731925 | 3.8020524740 | 1.4760579204 |
| 100000 | 338.96951104 | 273.02775278 | 378.59165534 | 148.99187885 |
| 200000 | 1389.0775099 | 1119.3102414 | 1557.7507861 | 617.09978142 |

Media dei tempi su test eseguiti per 10 volte

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DIM. LISTA | mergeSort | heapSort | radixSort | sort python |
| 100 | 0.0003127789 | 0.0006252169 | 0.0001561975 | 0.0 |
| 1000 | 0.0034394619 | 0.0107849510 | 0.0006263235 | 0.0 |
| 10000 | 0.0414312604 | 0.1463229641 | 0.0078172611 | 0.0 |
| 100000 | 0.4884238417 | 1.8339940380 | 0.0775596069 | 0.0006248688 |
| 200000 | 1.0230730082 | 3.9043273485 | 1.5638274942 | 0.0012559530 |

Media dei tempi su test eseguiti per 100 volte

|  |  |  |  |
| --- | --- | --- | --- |
| DIM. LISTA | quickSortRand | quickSortDet. | quickSortPrandom |
| 100 | 0.0006248569 | 0.0003128385 | 0.0006252098 |
| 1000 | 0.0110970088 | 0.0056267932 | 0.0079731185 |
| 10000 | 0.1602294426 | 0.1050315010 | 0.1562933008 |
| 100000 | 6.2223044776 | 5.6564580560 | 7.1455402484 |
| 200000 | 22.761754738 | 21.699926849 | 169.45930278 |

Media dei tempi su test eseguiti per 100 volte

