Ninivaggi Luca 5312342

Las Vegas Quicksort

Il laboratorio è stato implementato usando una porzione di codice in c++, per il calcolo del numero di confronti, e in Python, per la parte successiva in cui andiamo a graficare i risultati ottenuti.

L’ implementazione in Python anche per la prima parte non è stata presa in considerazione poiché decisamente troppo lenta (svariate ore di computazione per ottenere risultati); implementandolo multithreading si sarebbe ottenuta una velocità decisamente maggiore, ma alla fine è stato deciso di utilizzare c++.

Il codice dell’ algoritmo è stato copiato dal codice proposto nel corso di ASD, per garantire correttezza del codice.

PARTE 1 – C++

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <cmath>

using namespace std;

int SIZE = 10000;
int RUN  = 100000;
const int N_BIN = 50;
vector<int> BIN(N_BIN);

vector<int> X(RUN);
int I = 0;

void scambia(vector<int>& v, int i, int j)
{
  int tmp = v[j];
  v[j] = v[i];
  v[i] = tmp;
}

int partizionaInPlace(vector<int>& v, int inizio, int fine)
{
   int pivotIndex = inizio+rand()%(fine-inizio);
   scambia(v, pivotIndex, fine);
   int pivotValue = v[fine];
   int i = inizio -1;
   for (int j=inizio; j < fine; ++j) {
        X[I]++;
       if (v[j] <= pivotValue)
          scambia(v, ++i, j);
```

```
    }
 scambia(v, i+1, fine);
 return i+1;
}

void qs(vector<int>& v, int inizio, int fine)
{
    if (inizio < fine)
      {
          int pivot_index=partizionaInPlace(v, inizio, fine);
          qs(v, inizio, pivot_index-1);
          qs(v, pivot_index+1, fine);
      }
}

void quickSortRandom(vector<int>& v)
{
   qs(v, 0, v.size()-1);
}

void print(vector<int>& v) {
    for (size_t i = 0; i < v.size(); i++) cout <<  v[i] <<endl ;
}

int main(int argc, char* argv[]) {
    srand(time(NULL));
    cout.precision(10);

    for (; I < RUN; I++) {
        vector<int> array(SIZE);
        for (int i = 0; i < SIZE; i++) array[i] = std::rand() ;

        quickSortRandom(array);

        if(I%(RUN/100)==0) cerr<<  I*100/RUN <<"% completato" << endl;
    }
    cerr<< "100% completato" << endl;

    print(X);
}
```

Il codice in c++ è composto dal file “raccolta_dati.cpp”; l’ eseguibile, produce in output 100.000 iterazioni in cui conta i confronti effettuati in un array lungo 10.000 celle.

A ogni iterazione, andiamo a creare un nuovo array con valori casuali, e poi andiamo a ordinarlo tramite quicksort.

Dal punto di vista pratico, poiché a noi va a interessare solo il numero di confronti, non sarebbe nemmeno necessario andare a calcolare un nuovo array ogni volta, basterebbe avere la conferma che i valori sono il più eterogenei possibili: come caso limite, immaginiamo che tutti i valori siano uguali tra loro, rappresentati da x. Allora, a ogni chiamata di QS, il pivot sarà per forza messo tutto a sx, andando a creare un albero di chiamate ricorsive che è una catena, e quindi con un totale di n livelli e con 49995000 confronti.

Per poter comunicare con il codice python, andiamo a ridirezionare l' output su un file chiamato "output.txt".

Testato su un portatile, l' esecuzione dura un paio di minuti in totale.

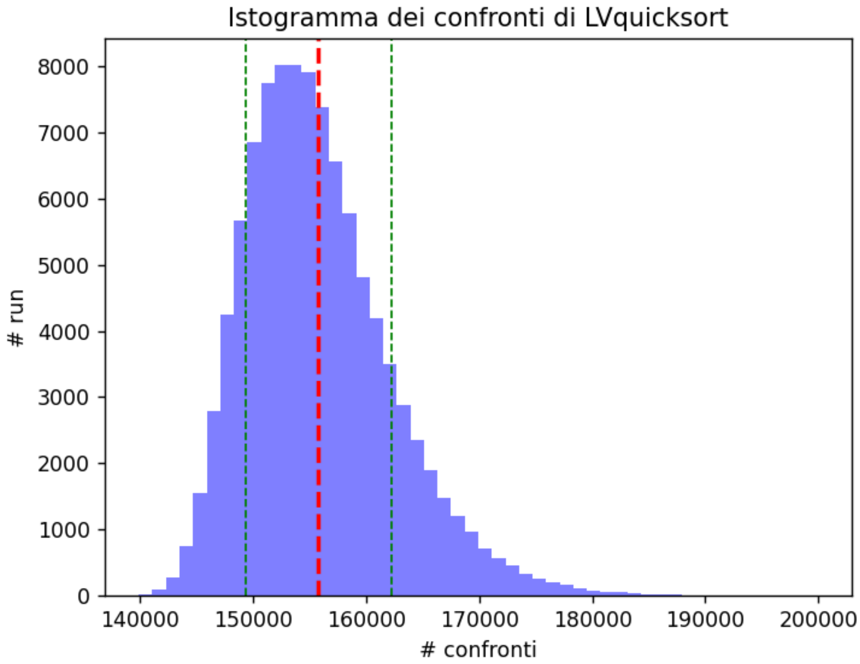
PARTE 2 – PYTHON

```
import math
import matplotlib.pyplot as plt

file_path = 'output.txt'
with open(file_path, 'r') as file:
    column_values = [int(line.strip()) for line in file.readlines()]

plt.hist(column_values, 50, facecolor='blue', alpha=0.5)

media = sum(column_values)/len(column_values)
varianza_2 = sum([(v-media)**2 for v in column_values])/(len(column_values)-1)
dev_std = math.sqrt(varianza_2).real
print(media,varianza_2,dev_std)

plt.axvline(x=media, color='red', linestyle='dashed', linewidth=2,
label='media')
#plt.axvline(x=media+dev_std, color='green', linestyle='dashed', linewidth=1,
label='deviazione standard')
#plt.axvline(x=media-dev_std, color='green', linestyle='dashed', linewidth=1,
label='deviazione standard')

plt.axvline(x=media*2, color='yellow', linestyle='dashed', linewidth=1, label='2
* media')
plt.axvline(x=media*3, color='purple', linestyle='dashed', linewidth=1, label='3
* media')

plt.xlabel('# confronti')
plt.ylabel('# run')
plt.title('Istogramma dei confronti di LVquicksort')
plt.show()
```

Il programma in Python legge dal file "output.txt" e produce un istogramma tramite la libreria "matplotlib".

Il grafico prodotto ci permette di intuire la distribuzione del numero di confronti, e possiamo determinare una media e una deviazione standard empirica.

RISULTATI OTTENUTI

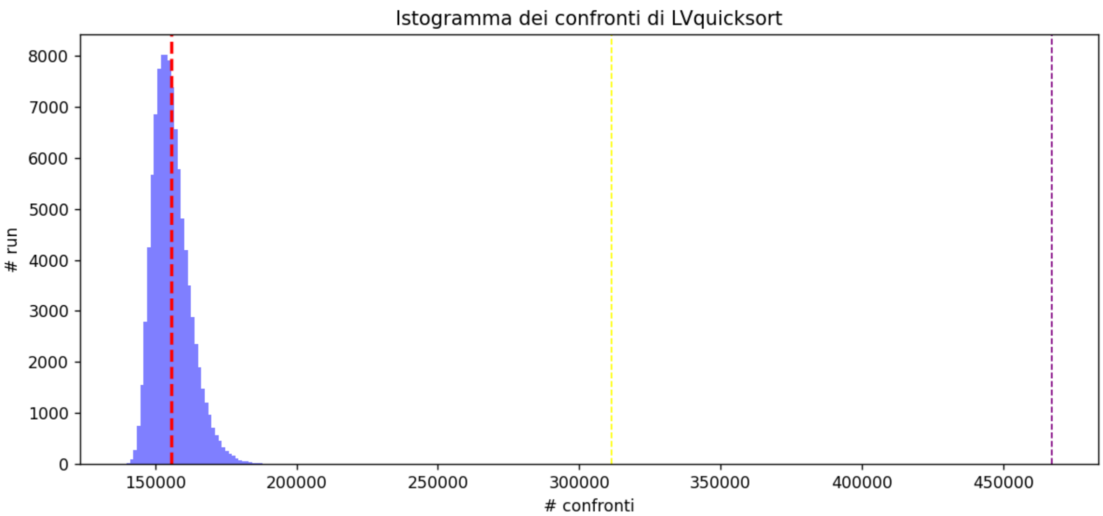


Partendo dai dati calcolati, ci possiamo ricavare empiricamente la media (rossa) e la deviazione standard (in verde).

Media = 155769

Varianza = 41632607

Deviazione Standard = 6452



Indichiamo inoltre il valore calcolato che assume il doppio e triplo della media: come possiamo osservare, non esistono run che abbiamo fatto un tale numero di confronti.

DISUGUAGLIANZE DI MARKOV E CHEBYSHEV

La 2 disuguaglianze ci forniscono informazioni sugli scostamenti che posso osservare empiricamente nei dati calcolati: quanto è la probabilità che ottenga un valore di confronti maggiore rispetto alla media?

Grazie a Markov osserviamo:

$$\Pr\{X \geq v\mu\} \leq \frac{\mu}{v\mu} = \frac{1}{v}$$

Ovvero ci garantisce che la probabilità che una variabile casuale realizzi un valore maggiore o uguale alla sua media moltiplicata per un fattore v, è inferiore all' inverso di v.

Nel nostro esempio, ci garantisce che

- la probabilità che X superi 2*μ è inferiore a 1/2
- la probabilità che X superi 3*μ è inferiore a 1/3

Ciò è verificato dai nostri dati empirici: entrambe le probabilità sembra essere inferiore a 0,00001; infatti anche con un numero così elevato di iterazioni, nemmeno una volta si è verificato un caso favorevole.

Chebyshev invece garantisce:

$$\Pr\{X \geq v\mu\} \leq \frac{\sigma^2}{(v-1)^2\mu^2}$$

Anche questa disuguaglianza è verificata

- Dato v=2, la probabilità risulta inferiore a

  $\frac{41632607}{155769^2 * (2-1)^2}$ = 0,0017158

  Infatti 0,00001 < 0,0017158

- Dato v=3, la probabilità risulta inferiore a

  $\frac{41632607}{155769^2 * (3-1)^2}$ = 0,00042895

  Infatti 0,00001 < 0,00042895

Osservando che le disuguaglianze sono verificate, abbiamo un' ulteriore conferma che i dati ottenuti siano corretti.