**Protocollo Las Vegas - Lorenzo Foschi - 4989646**

#include <iostream>

#include <random> // Per il random engine

#include <vector>

#include <chrono> // Per il seed del random\_gen

#include <unordered\_map> // Per l’output (k, v)

typedef std::vector<bool> bits;

typedef std::vector<bits> matrix;

typedef std::unordered\_map<int, int> mappa;

// Distribuzione uniforme (meglio del rand)

unsigned seed = std::chrono::system\_clock::now().time\_since\_epoch().count();

std::default\_random\_engine **generator**(seed);

// Ritorna il valore con più occorrenze

int **maj**(const bits& v) {

int c1 = 0;

int c0;

// Per ogni bit dell’array bits, conto gli 1

for (auto bit : v) c1 += bit;

// Il complemento sono le occorrenze di 0

c0 = v.size() - c1;

return (c1 > c0);

}

// Ritorna il numero di occorrenze di maj

int **tally**(const bits& v) {

int c1 = 0;

int c0;

for (auto bit : v) c1 += bit;

c0 = v.size() - c1;

return (c1 > c0) ? c1 : c0;

}

/\*

Nota sullo standard utilizzato per la matrice:

9 righe i (la 9a è il malizioso quindi non memorizzata);

ogni colonna j è il bit ricevuto al proc. i dal proc. j

[x, y,… !v[j]],

[x, y,… !v[j]],

[x, y,… !v[j]],

…

[...(9a riga non memorizzata)]]

\*/

int **main**() {

int r = pow(2, 10); // Round

int t = 1; // Maliziosi

int p = 7 \* t + 1; // Affidabili

int n = 8 \* t + 1; // processi

int L = 5 \* t + 1; // soglia L

int H = L + t; // soglia H

mappa rounds; // Array di rounds

bits **b**(n); // b (bit in trasmissione) di dimensione n

// Distribuzione uniforme valori (moneta onesta)

std::uniform\_int\_distribution<int> distribution(0, 1);

// Variabile per il valore medio del #round

double E = 0;

// Calcolo dei dati empirici (ciclo su r run)

for (int i = 0; i < r; ++i) {

// Creazione vettore bit casuali

bits **v**(n);

for (int j = 0; j < p; ++j) v[j] = distribution(generator);

// Creazione matrice con valori casuali generati a riga preced.

matrix **m**(p);

for (int j = 0; j < p; ++j) {

m[j] = v; // Inserimento vettori processi onesti

m[j].back() = !v[j]; // Inserimento valore malzioso

}

// (turno 1: bit iniziali (possibile accordo istantaneo come da specifica)

int cround = 0; // Contatore turni

while (true) {

// Lancio della moneta e incremento round

bool coin = distribution(generator);

cround++;

// Controllo terminazione (tutti i valori uguali su riga)

bool term = true;

for (int j = 1; j < p; ++j)

if (m[0][j] != m[0][j - 1]) term = false;

if (term) break;

for (int j = 0; j < p; ++j) {

// Calcolo di maj e tally

int mag = maj(m[j]);

int tl = tally(m[j]);

// Scelta random della soglia L o H e

// inserimento bit (maggioranza o 0)

int soglia = (coin ? L : H); // scelta rand soglia

b[j] = (tl >= soglia ? mag : 0); // prox bit

if (tl >= p) b[j] = mag; // accordo raggiunto

}

// Inserimento dei valori in ricezione

for (auto& v : m) v = b;

// Impostazione valori del malizioso (per ogni riga,

// …set dell’ultima colonna : 1 - sent)

for (int j = 0; j < p; ++j) m[j].back() = !b[j];

}

rounds[cround]++;

}

// stampo valori

for (const auto& [n, fqz] : rounds)

std::cout << n << " " << fqz << '\n';

// Calcolo del valore medio del #round

for (const auto& [n, fqz] : rounds)

E += (n \* fqz);

E /= r;

std::cout << "Valore medio del #round: " << E << '\n';

}

*Per l’implementazione si sceglie di scrivere un codice in cpp per il protocollo Las Vegas. Per i processi si utilizza una matrice, la cui convenzione è riportata nei commenti del codice cpp in prima pagina, e per memorizzare le frequenze si utilizza una unordered map in cui si registrano le occorrenze. Nel main si segue lo pseudocodice proposto, utilizzando le due funzioni aux per il calcolo dei valori di maj e tally.*

Su 210 prove i risultati ottenuti sono i seguenti (valore - frequenza e E[#round]):



I casi possibili sono i seguenti:

* Processi d’accordo all’unanimità → consenso

Il consenso al primo round (qui registrato) ha una *p* = 2/256 *(tutti “1” o tutti “0”)*

Dato j ∈ affidabili(*p*), indipendentemente dai maliziosi: tl(j) >= *p* > H > L

* p affidabili inizializzati in modo diverso ma maj(j) ha uguale valore

1. tl(j\*) < L per qualche j\* ∈ *p* → tl(j\*) < H **∀** j\* ∈ *p*
2. tl(j\*) >= H per qualche j\* ∈ *p* → tl(j\*) >= L **∀** j\* ∈ *p*
3. L <= tl(j\*) < H **∀** j\* ∈ *p (se ∃ j\* < L o >= H allora lo sono tutti (casi 1 e 2)*

La soglia casuale, in questi tre casi, vanifica l’azione dei maliziosi con p = ½

* p affidabili init in modo diverso e maj(j) ha diverso valore per qualche j (possibile se sugli 8 j\* ∈ *p ci sono 4 “1” e 4 “0” ed è il malizioso a far pendere l’ago della bilancia).* Se i due j hanno maggioranze diverse non può essere stata superata L, e a maggior ragione H, dunque tl(j) <= L **∀** j ∈ *p* → il bit di ogni k vale “0” per effetto dell’else al punto 6 dello pseudocodice → consenso al round2.