Compito 7.1

Il compito riguarda la soluzione del problema dell'Accordo Bizantino (Byzantine Generals) di Leslie Lamport tramite il metodo Monte Carlo.
In particolare, nel caso non si abbia la sicurezza che i processi trustworthy siano d'accordo ci si affida al lancio di una moneta comune.
In questo caso è stato scelto di impostare il numero totale di processi a 4, di cui 3 trustworthy e 1 faulty, questo per rispettare la formula secondo la quale -affinchè il problema risulti risolvibile- $T = 2f + 1$ (e quindi $N = 3f + 1$).
Per analizzare empiricamente l'effiecienza di questo algoritmo, è stato scelto di effettuare 2^{10} run.

Il codice:

Per la codifica dell'algoritmo ho scelto di usare il linguaggio C++ con nomi di variabili e funzioni in lingua inglese e commenti in italiano, come per il compito precedente la casualità (in questo caso legata alla scelta del comando da parte del campanile) è stata affidata alla funzione rand() della stdlib di C++.
Ho poi scritto il numero di round impiegati per giungere alla soluzione su un file di testo, che ho usato poi per fare il grafico.
La differenza principale tra la mia implementazione e lo pseudocodice contenuto nelle note è che gestendo i vari generali non come processi ma sequenzialmente, nel MCByzantine vero e proprio sono presenti in sequenza le azioni dei vari generali.

Inizializzazioni:
Qui ho scelto di usare una struct per rappresentare l'ambito di visiblità di ogni generale, questa scelta mi è sembrata rendere il codice più chiaro e maneggevole.

```
#define ATTACK 1
#define RETREAT 0

const int RunNumber = 1024; //Numero di run
const int N = 4; //Numero di generali totali
const int T = 3; //Numero di generali affidabili
bool bellTower; //Variabile casuale comune

enum names {ARMANDO, BOB, CARL, DAVID};

struct general{
    names name;
```

```
    bool curr_decision; //b(j), la decisione che il generale intende prendere e
che comunichera' al prossimo round

    //Le lettere ricevute dai generali
    bool letter_A; //La decisione comunicata dal generale A
    bool letter_B; //La decisione comunicata dal generale B
    bool letter_C; //La decisione comunicata dal generale C
    bool letter_D; //La decisione comunicata dal generale D
} Armando, Bob, Carl, David;

struct Tally{
    int A = 0;
    int B = 0;
    int C = 0;
};

struct Maj{
    bool A;
    bool B;
    bool C;
};
```

Singola run:

```
int run(){
    //Inizializzazioni
    Armando.curr_decision = ATTACK;
    Bob.curr_decision = ATTACK;
    Carl.curr_decision = RETREAT;

    //Inizio algoritmo
    int finalDecision = -1;

    Maj maj;
    Tally tally;
    int round = 0;
    while(finalDecision == -1){
        round++;
```

```
        evening();

        maj.A = majNtally(Armando, tally.A);
        maj.B = majNtally(Bob, tally.B);
        maj.C = majNtally(Carl, tally.C);

        bellTower = rand() % 2;

        finalDecision = MCByzantineGeneral(maj, tally);
    }

    return round;
}
```

La funzione evening() racchiude l'invio di messaggi dei trustworthy e del faulty:

```
void evening(){
    //La sera, ognuno invia la propria decisione agli altri generali
    broadcast_message(Armando.curr_decision, Armando);
    broadcast_message(Bob.curr_decision, Bob);
    broadcast_message(Carl.curr_decision, Carl);
    spyLetter();
}
```

```
int MCByzantineGeneral(Maj& maj, Tally& tally){
    //Ritorna 1 se la decisione finale e' ATTACK, 0 se e' RETREAT, -1 se non si
e' raggiunta una decisione

    int final_decision = -1;

    //Analisi Armando
    if(tally.A >= T){
        Armando.curr_decision = maj.A;
        if(maj.A == bellTower){
            assert( (tally.B < T || maj.B == maj.A )&& (tally.C < T || maj.C ==
maj.A) );
            final_decision = maj.A;
        }
```

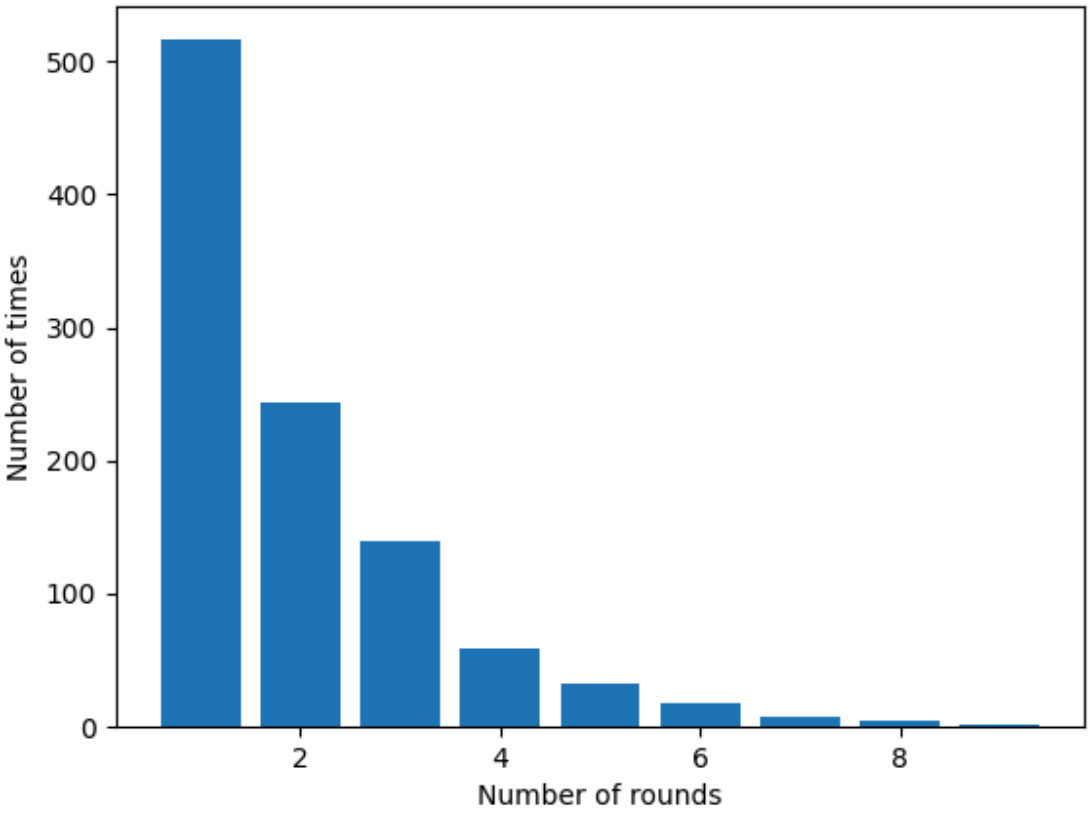
```
    } else {
        Armando.curr_decision = bellTower;
    }

    //Analisi Bob
    if(tally.B >= T){
        Bob.curr_decision = maj.B;
        if(maj.B == bellTower){
            assert( (tally.A < T || maj.A == maj.B )&& (tally.C < T || maj.C ==
maj.B) );
            final_decision = maj.B;
        }
    }  else {
        Bob.curr_decision = bellTower;
    }

    //Analisi Carl
    if(tally.C >= T){
        Carl.curr_decision = maj.C;
        if(maj.C == bellTower){
            assert( (tally.A < T || maj.A == maj.C )&& (tally.B < T || maj.B ==
maj.C) );
            final_decision = maj.C;
        }
    } else {
        Carl.curr_decision = bellTower;
    }

    return final_decision;
}
```

Il Grafico e conclusioni:



Nella parte centrale del MCByzantine per ogni generale G, sè il tally da lui rilevato è pari o maggiore al numero di processi trustoworthy, ciò significa che o i leali sono già tutti d'accordo (nel caso in cui per qualche motivo i faulty abbiano tutti comunicato la scelta opposta), o lo sono tutti appena diventati rilevando la stessa tally rilevata da G, oppure non hanno rilevato il superamento della soglia e se l'estrazione della moneta coincide con tale comando al prossimo round saranno certamente d'accordo, cosa che avviene con probabilità ½ .
Questo perchè se G ha rilevato 2f+1 lettere comunicanti un ordine, al massimo un altro generale G' potra' rilevare a favore dell'ordine opposto gli N - 2f + 1 eventuali generali affidabili rimanenti + f inaffidabili che hanno comunicato a lui un'informazione diversa, questa somma da come risultato 2f, che è inferiore alla soglia T 2f+1.

A scanso di equivoci, nel ho verificato quest'ultima asserzione tramite degli assert() nel codice.
Affidando la conferma decisione al lancio di una moneta, in via del tutto teorica si potrebbe rischiare ad affermare che è possibile una decisione non la si raggiunga mai. Tale affermazione sarebbe però realisticamente errata, la probabilità che l'accordo continui a non trovarsi dimezza ad ogni run, diventando essenzialmente nulla molto rapidamente, il suo valore atteso dalla nostra prova empirica è infatti molto basso: 2,00391.
L'efficienza di questo algoritmo è quindi considerabile costante (O(1)), poichè il suo costo sarà pari al valore atteso al più moltiplicato per qualche costante.