# AMS Frequency Moments

Il seguente documento è tratto da un analisi del paper di Alon, Matias, e Szegedy, "The Space Complexity of Approximating the Frequency Moments" [1].

Questo documento si propone di definire uno pseudocodice per l'algoritmo AMS Frequency Moments, per poi procedere con la sua implementazione in C/C++.

L'attenzione è stata concentrata sui momenti di ordine $k$ pari a 0 e 1.

## Introduzione

Sia $A = (a_1, a_2, ..., a_m)$ una sequenza di m elementi, con $a_i \in N = \{1, 2, ..., n\}$.

Denotiamo con $m_i$ le occorrenze i-esime nella sequenza A.

Il momento di ordine k è dato da $F_k = \sum_{i=1}^{n} m_i^k$, dove $m_i$ è il numero di occorrenze di $i$ nella sequenza.

$F_k$ è definito come la somma k-esime potenze dei conteggi $m_i$.

L'obiettivo dell'algoritmo AMS consiste nel fornire una stima accurata di $F_k$ usando un algoritmo randomizzato.

## Estimating $F_k$

Estraiamo un numero casuale $a_p$ della sequenza, dove l'indice $p$ è stato scelto in modo casuale e uniforme tra gli indici $1, ..., m$. Possiamo quindi definire $r$ come segue.

Sia $r = |\{q : q \geq p, a_q = a_p\}|$ il conteggio di $a_p$ in $A$, da un fissato $p$ in poi.

Definiamo la variabile $X = m(r^k - (r-1)^k)$.

Nel caso in cui $m$ sia sconosciuta: quando $a_m$ (elemento m-esimo della sequenza) arriva, viene rimpiazzato ad $a_p$ con probabilità $1/m$, in caso di rimpiazzo $r$ viene impostato a 1, altrimenti viene incrementato se $a_m = a_p$.

Come dimostrato da Alon et al. [1], si ha che:
$E(X) = \sum_{i=1}^{n} m_i^k = F_k$
$Var(X) = E(X^2) - (E(X))^2$ dove $E(X^2) \leq kF_1F_{2k-1}$ .

# Algoritmo

## PSEUDOCODE

Procedimento per una singola variabile X.

Case: m known

```
AMS_Frequency_Moment_m_known(A, k, m): # A stream, k moment order, m length of the stream
  initialize...
  p <- rand uniform (1,m)
  r <- 1
  q <- p + 1

  # procedure...
  while(steam, starting from q)
    if(a_q == a_p)
      r <- r +1
    q <- q + 1
  end while
  compute X = m(r^k - (r-1)^k)
  return X
```

Case: m unknown

```
AMS_Frequency_Moment(A, k): # A stream, k moment order
  # initialize...
  r <- 1
  a_p <- A[0]
  m <- 1

  # procedure...
  while stream do:
      pick random number in U(0,1)
      if random number < 1 / m: # with prob. 1/m accept replacement
          a_p <- A[m]
          r <- 1
      else if A[m] == a_p:  # increase r
          r <- r + 1
      end if
      m <- m + 1 # update m
  end while
  X <- m * (r^k - (r - 1)^k)  # compute X
  return X
```

# Input file e premesse

Lo stream utilizzato dai programmi *ams_f0* ed *ams_f1* è stato generato dal programma "generate_stream", disponibile nella directory "stream_generator".

I programmi *ams_f0* ed *ams_f1* sono stati testati e progettati per funzionare con un file di input contenente una serie di numeri interi e non negativi, ognuno disposto su una riga e separata da un punto e virgola. È inoltre disponibile uno script denominato "pulizia_stream.sh" che consente la rimozione di tutti i caratteri ad esclusione dei caratteri non numerici e del carattere separatore.

Esempio del formato del file di input:

```
11;
11;
19;
97;
8;
52;
54;
```

## Makefile

Per semplificare la compilazione dei programmi, è stato progettato un **Makefile**, i comandi disponibili sono "make" ed "make clean".

## Script di Pulizia (pulizia_stream.sh)

Lo script di pulizia è stato pensato per facilitare la fase preliminare della preparazione del file di input.

Uso dello script: ./pulizia_stream.sh [file_input] [file_output]

- Se non vengono passati argomenti, verranno utilizzati dei valori di default per file_input e file_output
- Se viene passato un solo argomento, verrà usato lo stesso argomento sia come file_input che come file_output
- Se vengono passati due argomenti, verranno utilizzati come file_input e file_output rispettivamente
- Viene inoltre effettuato un controllo sul numero di argomenti, il numero degli argomenti deve essere <= 2

## Stream generator

Per generare lo stream utilizzato in questo lavoro è stato implementato in C++ un **generatore di numeri pseudo-casuali**. Questo generatore, disponibile nella directory "stream_generator", si occupa inoltre del salvataggio dello stream in formato CSV, con possibilità di salvare l'output anche in altre estensioni.

Nella stessa directory sono presenti due script di utility scritti in bash: "test_generatore.sh" e

"controllo_input.sh". Questi script hanno lo scopo di testare le combinazioni di input disponibili, verificando sia il corretto funzionamento delle opzioni disponibili che il corretto funzionamento dei meccanismi di controllo dell'input inserito da utente.

Le distribuzioni implementate per la generazione dei numeri sono le seguenti:

- uniforme
- esponenziale
- poisson

## Usage

Il programma dispone di un opzione di usage, richiamabile tramite l'opzione -h, che stampa il seguente messaggio.

```
Utilizzo: ./generate_stream [-d distribuzione] [-l lambda] [-a min] [-b max] [-n lunghezza
Il seguente programma genera uno stream di numeri pseudo-casuali, salvando il risultato in
ATTENZIONE: Il seguente programma fornisce in output un file CSV di numeri interi, quindi
Le opzioni disponibili sono le seguenti:
  -h                Messaggio di aiuto
  -d distribuzione  Permette di specificare una distribuzione da usare: uniforme, esponenz
  -l lambda         Permette di specificare il parametro lambda usato per le distribuzioni
  -a min            Permette di specificare il limite inferiore per la distribuzione unifo
  -b max            Permette di specificare il limite superiore per la distribuzione unifo
  -n lunghezza      Permette di specificare la lunghezza dello stream. Default = 200
  -x cifre          Permette di specificare il numero di cifre decimali da mantenere. Defa
  -f file           Permette di specificare il nome del file CSV fino ad un massimo di 49
  -e estensione     Permette di specificare l'estensione del file fino ad un massimo di 4
  NOTA - caratteri non accettati: spazi, stringhe vuote, stringhe con solo spazi, caratter
```

## Implementazione

La libreria getopt è stata utilizzata in modo da permettere l'utilizzo delle opzioni.

La funzione err_sys è stata implementata per gestire gli errori. Tale funzione ha lo scopo di mostrare a schermo un messaggio di errore descrittivo e di terminare l'esecuzione del programma.

Le funzioni "uniforme" e "esponenziale" sono state adattate per generare numeri interi. Il

numero generato viene moltiplicato per $10^x$, dove $x$ è uno dei parametri modificabili, e successivamente troncato.

Sono stati eseguiti i seguenti controlli sull'input inserito da utente:

- dopo i numeri (a, b, lambda, x ed n) non ci siano lettere o caratteri simili
- la lunghezza dello stream e x siano un numeri interi e positivi
- a,b e lambda siano numeri decimali positivi
- i numeri decimali e interi (a, b, lambda, x e n) siano inferiori al valore massimo possibile.
- b sia maggiore di a
- l'input relativo alla distribuzione è troncato per accettare massimo 12 caratteri
- il nome del file sia più corto di 50 caratteri e che non contenga spazi, stringhe vuote, stringhe con solo spazi e caratteri speciali diversi da virgola, trattino e punto
- il nome dell'estensione sia più corto di 5 caratteri

Implementazione della generazione dello stream:

```
// ---------------------- PSEUDO NUMBER GENERATOR ----------------------
int uniforme(std::default_random_engine& generator, double a, double b, int x) {
    std::uniform_real_distribution<double> distribuzione(a, b);
    double numero = distribuzione(generator);
    return static_cast<int>(numero * std::pow(10, x));
}

int esponenziale(std::default_random_engine& generator, double lambda, int x) {
    std::exponential_distribution<double> distribuzione(lambda);
    double numero = distribuzione(generator);
    return static_cast<int>(numero * std::pow(10, x));
}


double poisson(std::default_random_engine& generator, double lambda) {
    std::poisson_distribution<int> distribuzione(lambda);
    return distribuzione(generator);
}
```

```
// --- GENERAZIONE STREAM ---
for (int i = 0; i < n; i++) {
    switch (distribuzione) {
        case UNIFORME:
            file << uniforme(generator, a, b, x) << "\n";      // genero numero e sal
            break;
        case ESPONENZIALE:
            file << esponenziale(generator, lambda, x) << "\n";
            break;
        case POISSON:
            file << poisson(generator, lambda) << "\n";
            break;

    }
}
```

# Momento di ordine 0: $F_0$

Alon et al., [1] per l'implementazione del momento di ordine 0 propone una modifica dell'algoritmo Flajolet-Martin [2] per calcolare il momento di ordine 0.
L'implementazione introduce randomicità mediante l'uso della funzione z_hash, permettendo di ottenere una stima $F_0$. Così facendo è necessario utilizzare solo O(log n) bit di memoria per contenere l'informazione.

Il momento di ordine k = 0, indicato anche $F_0$, è utilizzato per stimare il numero di elementi distinti in uno stream.

Sia definito il campo $F = GF(2^d)$, dove d è il più grande intero t.c. $2^d > n$.

Siano $a, b$ due numeri casuali definiti in $F$, si computi $z_i = a * a_i + b$, con prodotto e somma riferiti al campo $F$.

La funzione z così definita fornisce un mapping pairwise independent [1].

Sia $r(z)$ il numero di trailing $0s$. Sia $R$ il massimo valore di $r_i$, dove $r_i = r(z_i)$.

L'output dell'algoritmo è dato da $Y = 2^R$.

## Pseudocode

```
AMS_Frequency_Moment_0(A): # A stream
  // initialize...
  a, b random chosen

  // procedure...
  R <- (- inf)
  while(stream)
    z_i <- z(a_i)
    r_i <- r(z_i)
    R <- Max(r_i,R)
  end
  return 2^R

define z: z=a*x+b
define r: r calculate number of trailing 0s
```

## Output

Il programma offre diverse opzioni di output:

- Stampa a schermo dei risultati, è possibile disabilitare questa opzione
- Salvataggio in un file in formato csv

AVVERTENZA: Il programma non crea automaticamente il file csv su cui salvare i risultati, pertanto bisogna assicurarsi dell'effettiva presenza del file.

L'output del programma consiste in una rappresentazione eelle seguenti variabili: stima del momento di ordine 0 e tempo di esecuzione in secondi dell'algoritmo.

File csv:

```
algoritmo,stima,esecuzione
ams,128,0.000087
```

Output del terminale:

```
AMS Frequency Moments – momento di ordine 0
Distinct item stimati: 128
Tempo di esecuzione: 0.000087 [s]
```

## Usage

Il programma dispone di un opzione di usage, richiamabile tramite l'opzione -h, che stampa il seguente messaggio.

```
Utilizzo: ./ams_f0 [-f nome_file] [-p path] [-o output_file] [-d path_output_file] [-s sep
Il seguente programma utilizza l'algoritmo AMS per stimare calcolare il numero di F0, il
Le opzioni disponibili sono le seguenti:
  -h                    Messaggio di aiuto
  -f nome_file          Permette di specificare il nome del file da utilizzare per il calc
  -p path               Permette di specificare il percorso del file da utilizzare per il
  -o output_file        Permette di specificare il nome del file da utilizzare per salvare
  -d output_path        Permette di specificare il percorso in cui si trova il file di outp
  -s separatore         Permette di specificare il carattere di separazione degli elementi
  -q                    L'opzione quiet permette di sopprimere l'output a schermo.
ATTENZIONE: Il programma non crea in automatico il file di output, quindi bisogna assicura
```

## Implementazione

La libreria getopt è stata utilizzata in modo da permettere l'utilizzo delle opzioni.

La funzione err_sys è stata implementata per gestire gli errori. Tale funzione ha lo scopo di mostrare a schermo un messaggio di errore descrittivo e di terminare l'esecuzione del programma.

```
sprintf(formato_input, "%%d%c", separatore);
delta_t = -clock();

while (fscanf(file, formato_input, &a_i) == 1){   // lettura per riga
    if (a_i >= 0 && a_i <= INT_MAX) {     //  controllo validità valore
        z_i = z_hash(a, a_i, b);
        r_i = trailing_0s(z_i);
        R = max(R,r_i);
    } else {
        printf("Errore: Letto valore sconosciuto, il valore letto verrà scartato");
    }
}

delta_t += clock();
delta_t = delta_t / CLOCKS_PER_SEC;  // tempo di esecuzione in secondi
distinct_item_estimate = 1 << R;    // elevamento a potenza usando shift a sinistra di R
```

Per il calcolo di $2^R$ si è utilizzato lo shift a sinistra, invece dell'utilizzo della libreria math. Questa opzione è stato resa possibile in quanto R è un numero intero (non negativo) e distinct_item_estimate è una potenza di due.

Per l'implementazione completa, si rimanda ad "ams_f0.c".

## Controllo input utente

Per il controllo dell'input inserito da utente è stato fatto uso delle espressioni regolari in modo da limitare i caratteri inseribili.

Espressione regolare per i filename: `"^[a-zA-Z0-9_.-\\ ]+$"`
Espressione regolare per i path: `"^[a-zA-Z0-9_.-/\\ ]+$"`

```c
// utilizzo di strncpy per limitare i caratteri
strncpy(filename, optarg, MAXLENGTH - 1);
filename[MAXLENGTH - 1] = '\0';

// compilazione espressione
int result_compilazione_f = regcomp(&regex_function_filename, regex_filename, REG_EXTENDE
if (result_compilazione_f) {
  char error_mex[20],error_mex_output[100];
  regerror(result_compilazione_f, &regex_function_filename, error_mex, sizeof(error_mex))
  sprintf(error_mex_output, "Errore durante la compilazione della regex per il filename: %
  regfree(&regex_function_filename);
  err_sys(error_mex_output);
}

// controllo espressione regolare
int result_controllo_regex_f = regexec(regex_function_filename, optarg, 0, NULL, 0);
if (result_controllo_regex_f){
 char error_mex[20],error_mex_output[100];
 regerror(result_controllo_regex_f, regex_function_filename, error_mex, sizeof(error_mex)
 sprintf(error_mex_output, "Errore durante il controllodella regex per il filename: %s\t"
 regfree(&regex_function_filename);
 err_sys(error_mex_output);
}

regfree(&regex_function_filename);  // libero memoria filename regex
```

## Trailing 0s

La seguente implementazione in c calcola il numero di trailing 0s di un dato numero in input.
La funzione accetta solo numeri interi non negativi.

```
int trailing_0s(int a_i) {

    int zeros = 0;
    while ((a_i & 1) == 0) {    // finchè il bit meno significativo è 0
        zeros++;    // counter trailing_0s
        a_i = a_i >> 1; // applico l'operazione bit a bit di spostamento a destra di 1 pos
    }
    return zeros;
}
```

### Hash function z

La funzione hash $z_i = a * a_i + b$ è stata implementata come segue

```
srand(3454256);  // seed
a = rand() % 100;  // [0;99]
b = rand() % 100;  // [0;99]
```

```
int z_hash(int a, int x, int b) {
    return a*x + b;
}
```

# Momento di ordine 1: $F_1$

Ricordiamo: $X = m(r^k - (r-1)^k)$

Il momento di ordine k pari a 1 ($F_1$) corrisponde alla lunghezza dello Stream (la somma di tutti gli elementi distinti).

Per come è definito X, è sufficiente utilizzare un singolo contatore per calcolare il numero della lunghezza di uno stream.

Il funzionamento dell'algoritmo è il seguente: con probabilità pari ad 1/m viene accettato un elemento dello stream, incrementando il conteggio di m (contatore della lunghezza dello stream).

```
#include <stdlib.h>
unsigned int seed = 3454256;
srand(seed);

int m = 1;

sprintf(formato_input, "%%d%c", separatore);

delta_t = -clock();
while (fscanf(file, formato_input, &a_i) == 1){
  double p_i = (double)rand() / RAND_MAX;
  if (rand_num < 1.0 / m) {
    if (a_i >= 0 && a_i <= INT_MAX) { // se l'elemento è valido
        m++;
    } else {
        printf("Errore: Letto valore sconosciuto, il valore letto verrà scartato");
    }
  }
}

delta_t += clock();
```

Come *ams_f0*, questa implementazione di ams_f1 implementa gli stessi controlli sull'input dell'utente.

Per l'implementazione completa, si rimanda ad *ams_f1.c*.

## Output

*ams_f1* offre le medesime opzioni di output di *ams_f0*:

- Stampa a schermo dei risultati, è possibile disabilitare questa opzione
- Salvataggio in un file in formato csv

AVVERTENZA: Il programma non crea automaticamente il file csv su cui salvare i risultati, pertanto bisogna assicurarsi dell'effettiva presenza del file.

L'output del programma consiste in una rappresentazione eelle seguenti variabili: stima del momento di ordine 0 e tempo di esecuzione in secondi dell'algoritmo.

File csv:

```
algoritmo,stima,esecuzione
ams,157,0.000746
```

Output del terminale:

```
AMS Frequency Moments - momento di ordine 1
Lunghezza dello stream stimata: 157
Tempo di esecuzione: 0.000746 [s]
```

## Usage

Il programma dispone di un opzione di usage, richiamabile tramite l'opzione -h, che stampa il seguente messaggio.

```
Utilizzo: ./ams_f1 [-f nome_file] [-p path] [-o output_file] [-d path_output_file] [-s sep
Il seguente programma utilizza l'algoritmo AMS per stimare calcolare il numero di F1, il
Le opzioni disponibili sono le seguenti:
  -h                   Messaggio di aiuto
  -f nome_file         Permette di specificare il nome del file da utilizzare per il calc
  -p path              Permette di specificare il percorso del file da utilizzare per il
  -o output_file       Permette di specificare il nome del file da utilizzare per salvare
  -d output_path       Permette di specificare il percorso in cui si trova il file di outp
  -s separatore        Permette di specificare il carattere di separazione degli elementi
  -q                   L'opzione quiet permette di sopprimere l'output a schermo.
ATTENZIONE: Il programma non crea in automatico il file di output, quindi bisogna assicura
```

# Migliorare la stima di $F_0$ tramite Median of Means

Per migliorare la stima di $F_0$ è possibile utilizzare la tecnica "Median of Means". Questa tecnica consiste nell'eseguire l'algoritmo con multiple funzioni hash, raggrupparne i risultati, calcolare la media e infine selzionare la mediana come stima finale.

**Parametri**

- R: array contenente il numero massimo di trailing 0s per ogni funzione hash.
- a,b: array contenente i valori di a e b per ogni funzione hash.
- NHASH: numero di funzioni hash
- GROUHASH: numero di gruppi

Per l'implementazione si è deciso di utilizzare (NHASH) 10 funzioni hash e di suddividerle in (GROUHASH) 5 gruppi.

```
while (fscanf(file, formato_input, &a_i) == 1) {
    if (a_i >= 0 && a_i <= INT_MAX) {
        for (int i = 0; i < NHASH; i++) {

            z_i = z_hash(a[i], a_i, b[i]);
            r_i = trailing_0s(z_i);
            R[i] = max(R[i], r_i);

        }
    }
}
```

Per calcolare la stima degli elementi distinti, si raggruppano i risultati in gruppi (NHASH / GROUHASH) e si calcola la media di ogni gruppo.
Successivamente, si calcola la mediana di queste medie in modo da avere una stima pià accurata degli elementi distinti.

```
for (int i = 0; i < NHASH; i++) {
    d_i_estimates[i] = 1 << R[i];
}

for (int i = 0; i < NHASH / GROUHASH; i++) {
    double sum = 0.0;
    for (int j = 0; j < GROUHASH; j++) {
        sum += d_i_estimates[i * GROUHASH + j];
    }
    means[i] = sum / GROUHASH;
}

double distinct_item_estimate = median(means, NHASH / GROUHASH);
```

# Simulazione

Per acquisire le informazioni necessarie circa le prestazioni dei programmi è stato fatto uso gnu-time (su mac).
Quindi, per eseguire la simulazione su mac bisogna installare gnu-time, per rendere disponibili informazioni come ‘Maximum resident set size’.

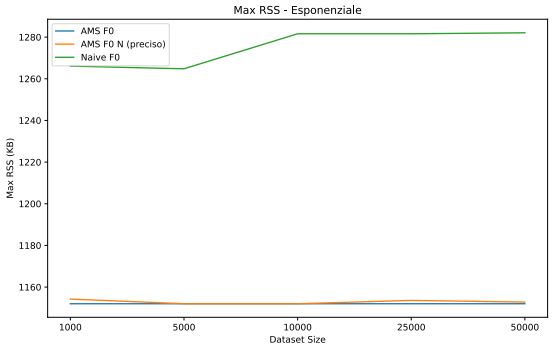
```
brew install gnu-time
```

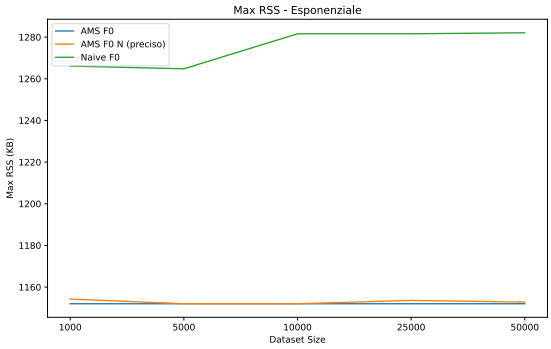
Le metriche raccolte sono:

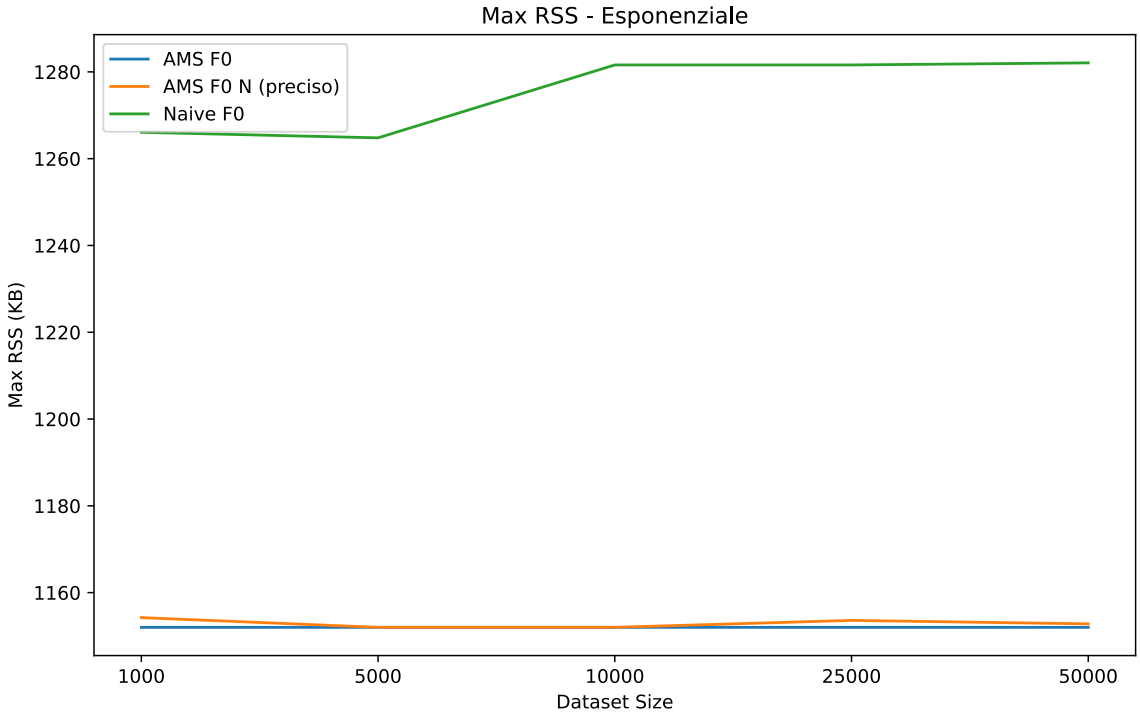
- Stima di $F_k$
- Tempo di esecuzione (in secondi)
- Percentuale di utilizzo della cpu
- Maximum RSS: Maximum Resident Set Size che sarebbe la massima memoria utilizzata dall'algoritmo

# Confronto tra AMS e un implementazione Naive

Per valutare le prestazioni dell'implementazione degli algoritmi ams_f0 e ams_f1 sono state utilizzate delle implementazioni naive per il calcolo di $F_0$ ed $F_1$.




# Bibliografia

[1] Alon, N., Matias, Y., & Szegedy, M. (1999). The Space Complexity of Approximating the

Frequency Moments. *Journal of Computer and System Sciences, 58*(1), 137-147. https://doi.org/10.1006/jcss.1997.1545

[2] P. Flajolet and G. N. Martin, "Probabilistic counting," 24th Annual Symposium on Foundations of Computer Science (sfcs 1983), Tucson, AZ, USA, 1983, pp. 76-82, doi: 10.1109/SFCS.1983.46.