Relazione MinHash

Algoritmo di minhash sviluppato in versione seriale e parallela, utilizzando Pthread e OpenMP

1. Presentazione

1.1 MinHash

MinHash è una tecnica algoritmica per stimare velocemente il grado di somiglianza di due insiemi. È stato inizialmente utilizzato dal motore di ricerca AltaVista per identificare duplicati tra le pagine web ed eliminarli dai risultati di ricerca. Viene anche utilizzato per raggruppare documenti per somiglianza dei loro insiemi di parole [1].

Per il calcolo della somiglianza tra insiemi si usa comunemente il coefficiente di similarità di Jaccard, che è definito dal rapporto tra la dimensione dell'intersezione e la dimensione dell'unione degli insiemi campionari [2] :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}. \tag{1}$$

L'obbiettivo del MinHash è di stimare velocemente J(*A,B*) senza calcolare esplicitamente l'intersezione e l'unione, questo per poter lavorare su grandi quantità di dati rendendo i tempi accettabili. Tramite una funzione di hash e delle permutazioni applicate a insiemi di dati, di cui vengono ricavati dei valori di hash minimi, il minHash produce una stima (probabilistica) approssimativa dell'Indice di Jaccard.

Ogni documento, dopo essere stato tokenizzato (eliminando formattazione, maiuscole, ecc) viene ridotto in tempo lineare a uno *sketch* di qualche centinaia di bytes composto da una collezione di *fingerprint* (o *signature*, ottenute tramite una funzione di hash e permutazioni) di *shingle*. Quest'ultimi sono sottosequenze contigue di parole di lunghezza fissa che differiscono l'una dall'altra per un solo elemento e vengono usati per rendere i documenti compatibili con il problema dell'intersezione. Per convenienza non vengono usati direttamente, ma mappati in *signatures* da 64 bit, con il presupposto che se due *signatures* sono uguali, con grande probabilità anche gli oggetti a cui si riferiscono lo sono. Per ridurre al minimo l'impatto delle collisioni sul risultato finale si applica una permutazione randomica sugli *shingle* e il *fingerprint* del valore minimo (in pratica sono sufficienti permutazioni pseudo-randomiche) [3].

1.2 Stima della similarità

Poniamo di avere due documenti *A* e *B*, e per ciascuno un insieme S_A e S_B delle *signatures* di 64 bit ottenute dai rispettivi *shingles*. *S* sarà quindi un sottoinsieme di S_n, l'insieme cioè dei valori in $[2^{64}]$. Sia π una permutazione presa randomicamente dall'insieme di tutte le possibili permutazioni di $[2^{64}]$[1]: la probabilità che il minimo della permutazione π sui valori in S_A sia uguale al minimo della permutazione π sui valori in S_B è uguale all'Indice di Jaccard su *A* e *B*:

$$\Pr\big(\min\{\pi(S_A)\} = \min\{\pi(S_B)\}\big) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|} \tag{2}$$

1. Ricordiamo che, dato un insieme *X*, una permutazione è un'applicazione biiettiva da *X* in *X*, quindi π mapperà i valori in S_A e S_B in valori in $[2^{64}]$. [4]

Per dimostrare questa uguaglianza, notiamo che applicando una permutazione π su X, tutti gli elementi x in X hanno uguale probabilità di diventare il minimo *(*detta *min-wise independence condition)*:

$$\Pr\big(\min\{\pi(X)\} = \pi(x)\big) = \frac{1}{|X|}. \tag{3}$$

Sia α la più piccola immagine in $\pi(S_A \cup S_B)$: i minimi valori della permutazione π su S_A e S_B sono uguali se e solo se α è immagine di un elemento in $(S_A \cap S_B)$. Quindi:

$$\begin{aligned}\Pr\big(\min\{\pi(S_A)\} = \min\{\pi(S_B)\}\big) &= \Pr(\pi^{-1}(\alpha) \in S_A \cap S_B) \\ &= \frac{|S_A \cap S_B|}{|S_A \cup S_B|} = r_w(A, B).\end{aligned} \tag{4}$$

Per stabilire la somiglianza tra A e B sarà sufficiente stabilire in anticipo un numero t di permutazioni in modo tale che S_A e S_B siano gli insiemi dei minimi di queste permutazioni. Così per ogni documento, indipendentemente dal numero di *shingles*, si ricava uno stesso numero di *signatures* ottenute tramite questa operazione di *min-Hash*. In pratica, poiché è impossibile avere permutazioni randomiche distribuite uniformemente su S_n è possibile scegliere un piccolo sottoinsieme di queste permutazioni che soddisfino la *min-wise independence condition* (3), poiché questa è condizione sufficiente per la (2) [3], che verranno applicate a tutti i documenti da confrontare.

La permutazioni applicate agli *shingles* fanno sì che eventuali collisioni abbiano un impatto modesto sul risultato, permettendo quindi di avere *signatures* di dimensioni relativamente piccole (64 bit). Uno *sketch* sarà quindi precisamente la collezione ordinata di *signatures* di un documento ottenuta da questa operazione di *min-Hashing* degli *shingles*.

1.3 Clustering di documenti

L'utilizzo degli *sketches* permette di raggruppare una collezione di *m* documenti in un insieme di documenti simili in tempo O(mlogm) invece che O(m^2). Una volta calcolati gli *sketches* nel modo sopra illustrato, il *clustering* avviene in ulteriori 3 fasi:

1. Tutti gli *sketches* di tutti i documenti vengono espansi in una lista di coppie {signature, ID documento}, ordinata per il valore delle *signatures.*

2. Viene generata una lista di tutte le coppie di documenti che condividono almeno una *signature*, insieme al numero di *signatures* che hanno in comune. Per fare questo si genera una lista di triple {ID doc1, ID doc2, 1} per ogni *signature* condivisa dai documenti. Si ordina questa lista per ID doc1 e si salva la somma del numero delle *n* signatures condivise in {ID doc1, ID doc2, *n*}.

3. Si esamina ogni tripla {ID doc1, ID doc2, *n*} e si valuta se supera la soglia scelta di similarità.

1.4 Filtering Near-Duplicates

Per ridurre ulteriormente la dimensione delle *signatures* e il costo computazionale, si usa la tecnica dei *Near-Duplicates*. Due documenti quasi identici condivideranno quasi tutte le *signatures*, che possono essere raggruppate in *k* gruppi di *s* elementi ciascuno, con *k* e *s* opportunamente scelti [3]. Ogni gruppo viene ridotto in una nuova *fingerprint*, anche questa di 64 bit, denominata *feature*. Gli sketches di ogni documento consisteranno quindi di *k features*, riducendo di *s* volte lo spazio richiesto. Queste *features* vengono poi confrontate tra coppie di documenti, in base all'indice i_k del gruppo di appartenenza. In generale i documenti che condividono almeno 2 features possono essere considerati duplicati e uno dei due può essere eliminato.

1.5 Approccio scelto in questo progetto

Il calcolo della somiglianza tramite *clustering*, che confronta le *signatures* senza tenere conto del loro ordine rispetto al documento da cui sono state generate, permette di individuare parti comuni nei documenti anche se si trovano in posizione diverse. È dunque possibile ottenere un indice di similarità molto preciso, ma non è possibile determinare l'identicità dei documenti per quanto riguarda l'ordine delle parti. La tecnica dei *Near-Duplicates* permette invece, confrontando ordinatamente le *features (*che sono una rappresentazione ridotta delle *signatures)*, di individuare coppie di documenti che hanno una probabilità molto alta di essere identici. Questa seconda tecnica, non aggiungendo nulla rispetto alla prima in termini di complessità algoritmica e parallelizzazione, non è stata implementata in questo progetto.

detto questo si può eliminare dal `check_and_print_similarity()` la parte che ricalcola le signatures in comune:

```
for(int signature=0; signature<N_SIGNATURES; signature++)
if (minhashDocumenti[doc1][signature] == minhashDocumenti[doc2]
[signature])
shared+=1;
```

e sostiuire shared con shared_signatures

2. Implementazione algoritmo

2.1 Implementazione algoritmo seriale

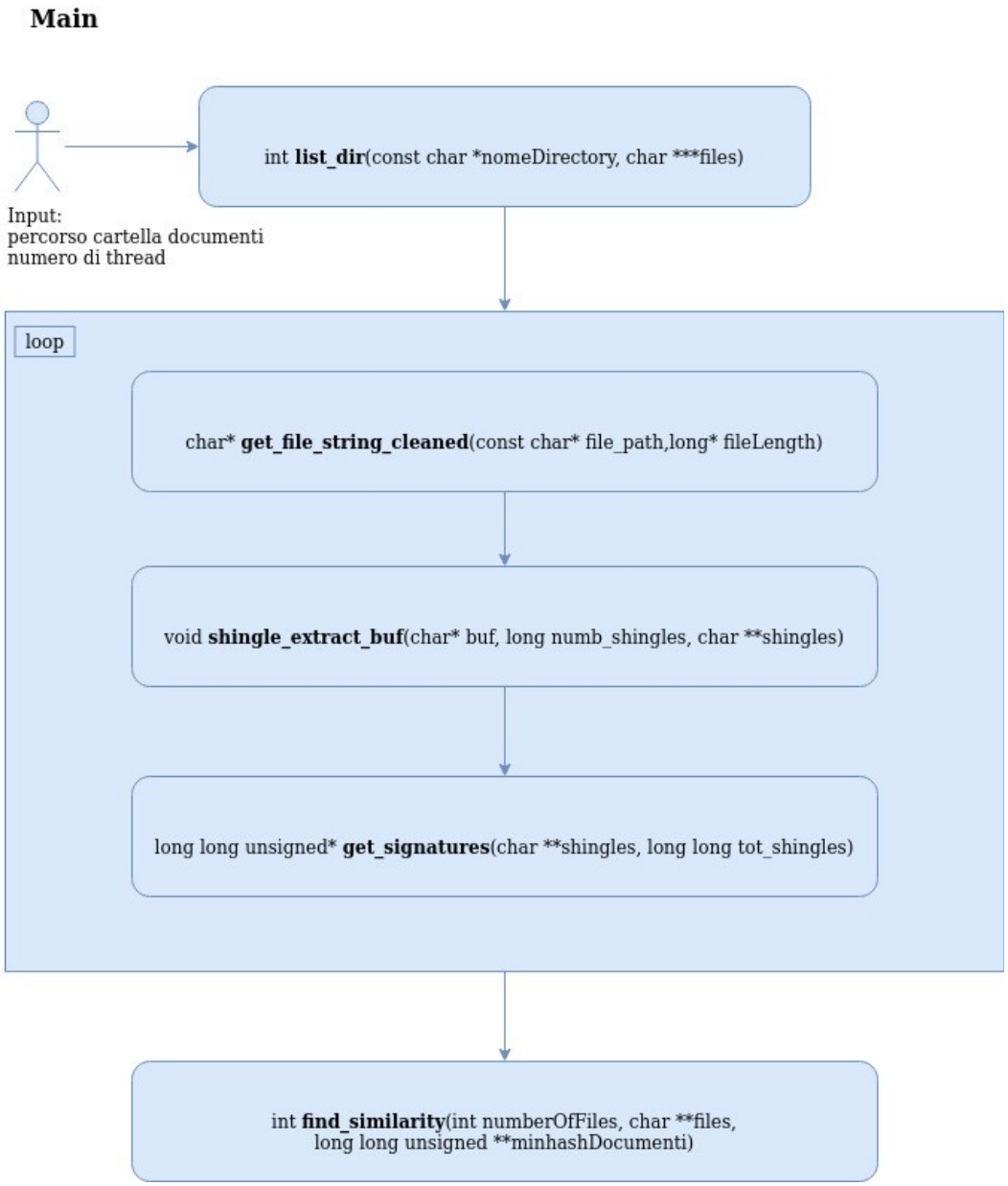
Il programma è composto da più moduli. Il principale è il file main.c che richiama a sua volta i file: documents_getters.c, tokenizer.c, shingle_extract.c, hash_FNV_1.c, get_similarities.c e per fini di test il file time_test.c .

Nel main.c viene avviata l'applicazione che prende come argomento il percorso dei documenti da verificare, assumendo che siano file di testo. Il recupero dei nomi dei documenti viene fatto con la funzione list_dir che preso in input il path della cartella va a popolare per riferimento un array di stringhe con i nomi dei files e restituisce la grandezza dell'array.

Ogni file trovato viene elaborato con le seguenti funzioni:

- il **get_file_string_cleaned** che si occupa di recuperare il contenuto del file e ripulirlo dagli spazi, formattatura, maiuscole e caratteri non ASCII.
- **shingle_extract_buf** estrae gli *shingles* dal testo tokenizzato e li copia in una array di stringhe.
- **get_signatures** crea le *signatures* applicando la funzione di hash e 199 permutazioni.

Raccolte tutte le *signatures* per tutti i files, la funzione **find_similarity** si occupa dell'ordinamento e della comparazione delle *signatures* e calcola la similarità tra i documenti.

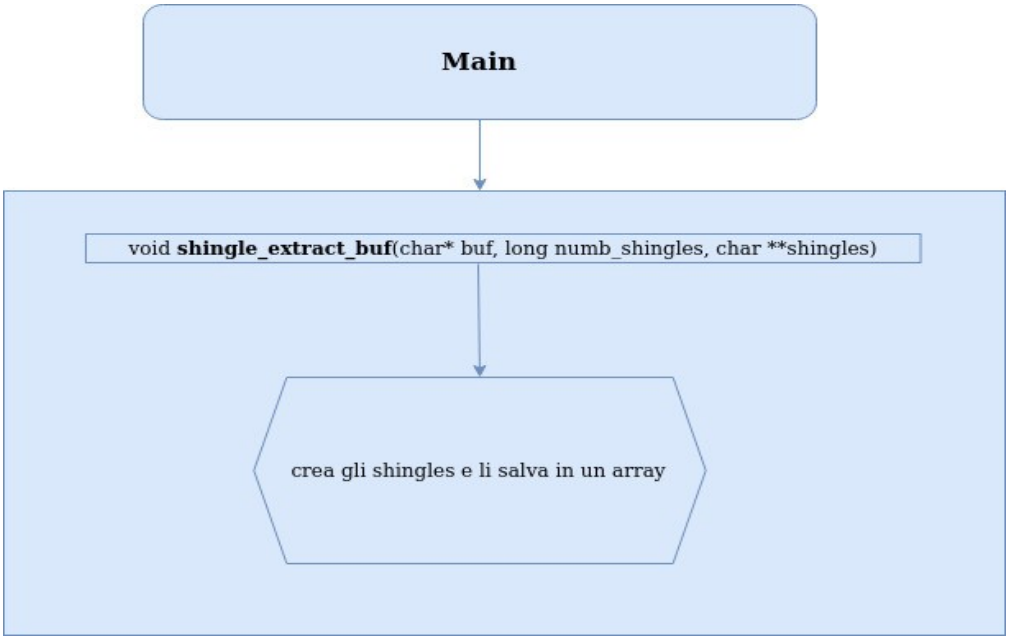


La costante K-SHINGLE permette di scegliere la lunghezza in caratteri degli *shingles*, che dovrà essere uguale per tutti i files da confrontare. Per questo programma è stato scelto il valore 54, che corrisponde a 9 parole da 6 caratteri l'una.

Nel **main**, dopo aver ottenuto la dimensione del file tokenizzato, viene calcolato il numero *N* degli *shingles* che dovranno essere creati per quel file, tramite la formula:

$$N = fileSize - K\_SHINGLE + 1$$

Poi viene chiamata la funzione **shingle_extract_buf** che estrae gli N *shingles* dalla stringa buf in cui è stato salvato il contenuto tokenizzato del file, e li salva tramite puntatore in una matrice di caratteri di dimensione (N x K_SHINGLE).



La funzione **get_signatures** prende il puntatore alla matrice di *shingles* e opera i seguenti passaggi:

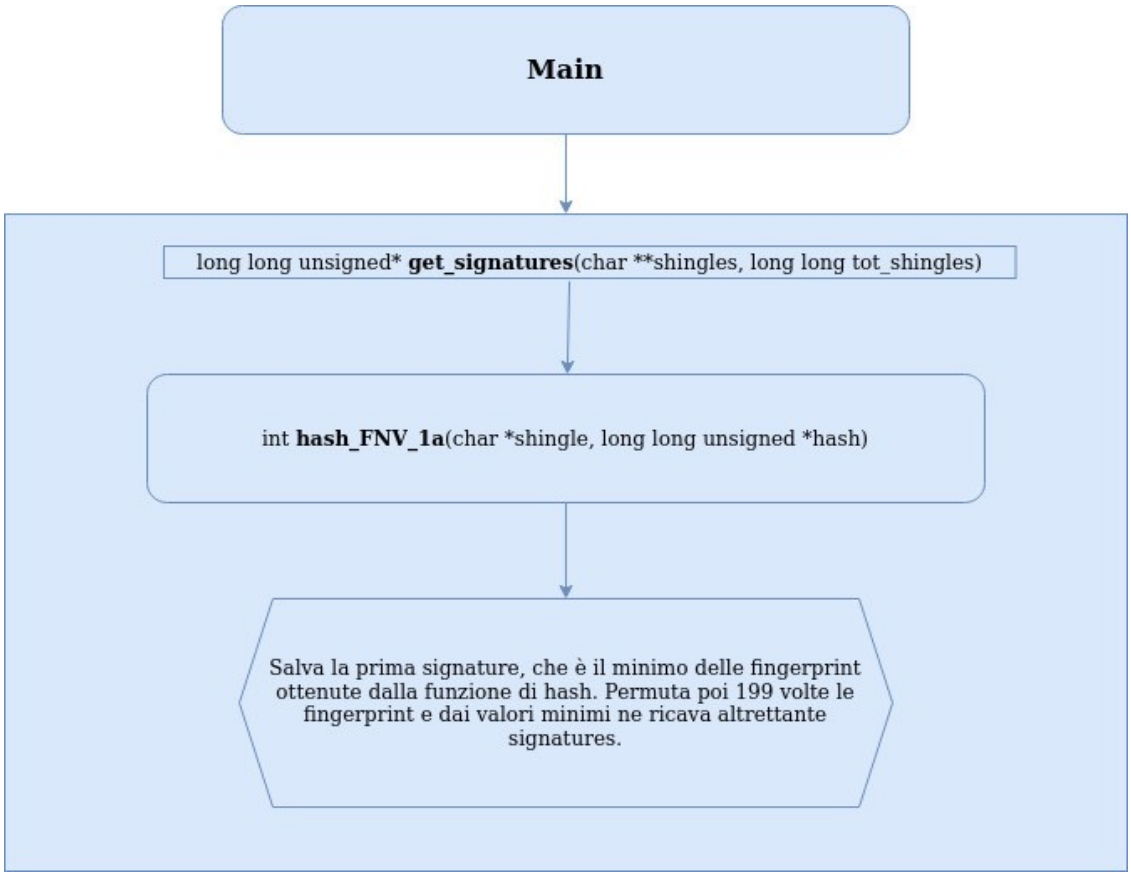
- converte ogni *shingle* in una *fingerprint* da 64 bit tramite la funzione **hash_FNV_1a**. Questa utilizza il numero primo *FNV_PRIME* (scelto per garantire buone proprietà di dispersione[5]) che moltiplica, per ogni carattere *i* dello *shingle*, il valore ottenuto al ciclo precedente (partendo da una base fissa da 64bit detta *FNV offset basis*) per poi applicargli a sua volta lo *XOR* con il carattere *i*.

- Tutte queste *fingerprints* che sono in numero pari al numero degli *shingles*, vengono temporaneamente salvate in un array. La minore di queste costituisce la prima *signature* per questo file.

- Con tutte le *fingerprints* viene poi eseguita l'operazione di XOR con 199 numeri random, che saranno sempre gli stessi per tutti i file, che servono a produrre le permutazioni pseudo-random necessarie per la (2).

- Dopo ogni permutazione di tutte le *fingerprints*, si salva quella di valore minimo.

- Alla fine sono state create le 200 *signatures*, che vengono salvate nella matrice minhashDocumenti contenente tutte le *signatures* di tutti i documenti da confrontare.


Main
long long unsigned* get_signatures(char **shingles, long long tot_shingles)
int hash_FNV_1a(char *shingle, long long unsigned *hash)
Salva la prima signature, che è il minimo delle fingerprint ottenute dalla funzione di hash. Permuta poi 199 volte le fingerprint e dai valori minimi ne ricava altrettante signatures.

References:

[1] https://en.wikipedia.org/wiki/MinHash
[2] https://it.wikipedia.org/wiki/Indice_di_Jaccard
[3] Identifying and Filtering Near-Duplicate Documents, Andrei Z. Broder
[4] http://www.dmf.unisalento.it/~scienze/Download/algebra_2new.pdf
[5] https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Vo_hash_function