Algoritmica per il web

Francesco Tomaselli

29 settembre 2021

Indice

1 Crawling

1.1 Principi di base

Insiemi di nodi in una visita All'interno di un processo di crawling, si distinguono tre insiemi di nodi

- U: nodi sconosciuti
- F: frontiera, ovvero nodi che sono conosciuti ma non ancora visitati
- V: nodi visitati

Una visita del web opera scegliendo un nodo della frontiera, aggiungendolo ai visitati, e aggiungendo i siti raggiungibili da esso alla frontiera.

In una visita di un grafo classica, gli insiemi U e F coincidono nel secondo.

Osservazione 1. *Teoricamente una visita potrebbe esaurire la frontiera, praticamente non finisce mai, a parte in rari casi.*

Inizializzazione frontiera La frontiera va inizializzata, tipicamente si scelgono siti web popolari, per esempio giornali etc. In generale si considerano *semi* di inizializzazione, ovvero un insieme di siti web scelti da umani, che sono promettenti.

La scelta del seme governa l'esplorazione della frontiera, verranno scelti infatti prima siti più vicini ai siti di inizializzazione.

Un'altra strada per la frontiera è scegliere a priori un insieme di siti web, e considerarla direttamente come frontiera, tralasciando i siti raggiungibili da essi.

1.2 Strutture dati

Strutture in memoria Sono cruciali le strutture usate per memorizzare i nodi visitati e la frontiera.

Mantenere i visitati in una hashtable è ragionevole, non lo è per la frontiera, tipicamente di ordini di grandezza più grande dell'insieme V.

Limitare la grandezza della frontiera è cruciale, si potrebbere preferire i link iniziali in una pagina piuttosto che quelli a fondo. Oppure limitare il numero massimo di pagine estratte da un singolo dominio o considerare un limite alla profondità all'interno di un singolo sito web.

Scelta del prossimo nodo Ciò che determina il comportamento una visita di crawling è la scelta del prossimo nodo della frontiera. Una scelta ragionevole potrebbe essere una visita in ampiezza a partire dalla frontiera iniziale. L'assunzione è che pagine di qualità puntino ad altre pagine di qualità.

Operare in profondità non funziona, significherebbe continuare a scendere all'infinito, al contrario di una classica DFS che termina e si *torna indietro* con la ricorsione. Questo perchè i siti non visitati sono praticamente infiniti.

Criteri più sofisticati possono associare una sorta di priorità alle pagine. Tali criteri sono legati ai contenuti delle pagine, alla struttura dell'url, alcuni esempi sono:

- Url corti, con l'assunzione che i livelli siano separati da un backslash;
- Url fuori dal sito corrente;
- Parole chiave di interesse all'interno dell'url;
- Utilizzo il contenuto della pagina corrente per avere informazioni sulla pagina successiva.

In questo caso si utilizza una coda a priorità, con qualche valore di priorità associato ad ogni nodo.

Osservazione 2. *In questo paragrafo si sta assumendo un contesto single thread, ovviamente nella realtà si utilizzerà un approccio multithread, e molti fattori, quali latenza, tempi di risposta, race condition, influenzano sull'ordine di visita effettivo.*

1.3 Esempio strutture dati

Visti hashati Una prima idea per mantenere l'insieme V è non memorizzare url completi ma una certa firma digitale di essi.

Consideriamo ad esempio una funzione di hash:

$$f : URL \longrightarrow 2^{64} = \{0, \ldots, 2^{64-1}\}$$

È possibile che due url collidano, creando errori sui positivi, ovvero un url non davvero visitato viene visto come già esplorato.

Solitamente le collisioni non importano, ma, dati k elementi contenuti nella struttura, data una funzione di hash buona, tipicamente il numero di collisioni è dell'ordine di $\frac{k^2}{2n}$, dove n è la grandezza del codominio.

Nella pratica quindi, funzioni di hash come f, non creano un numero spiacevole di collisioni.

Osservazione 3. *Una tabella di firme è molto più efficiente di mantenere i dati effettivi, basti pensare a problemi di allocazione di memoria inefficiente,*

frammentazione etc. Pagare il prezzo dei conflitti, implica risparmiare molti problemi e spazio.
Mantenere una tabella di firme significa di fatto mantenere in memoria strutture più piccole dell'information theoretical lower bound, pagando il prezzo delle collisioni.

Database NoSQL Sono database che memorizzano entry chiave valore, un esempio è *RocksDB*. Se ordinati sono implementazioni efficienti di B-Tree, altrimenti di dizionari classici, in parte memorizzati su disco.

Si utilizzano anche LSM-Tree, alberi che dividono per livelli le chiavi e mantengono chiavi utilizzate di frequente all'inizio, spostando chiavi poco frequenti nei livelli più bassi.

Memorizzazione offline Si mantengono solo file su disco e non strutture in memoria. I file vengono ordinati e fusi spesso. In particolare, si hanno a disposizione:

- *VIS*: url da visitare
- F: frontiera
- V: url visti

Il crawler accumula gli url che trova nelle pagine nel file F, fino a che si terminano i siti da visitare, contenuti in *VIS*, oppure lo spazio su disco.

A questo punto la frontiera viene ordinata e de-duplicata. L'operazione è necessaria poiché il file potrebbe contenere duplicati, visto che non esiste nessuna struttura in memoria che impedisce l'aggiunta di ripetizioni.

A questo punto si fondono i file F e V:

- Se un url sta in entrambi non faccio nulla;
- Se trovo un url in V ma non in F non faccio nulla;
- Se trovo un url in F ma non in V, lo aggiungo a V e a *VIS*.

La fusione è lineare nella lunghezza dei file F e V, poiché gli url sono in ordine lessicografico. Procedo a questo punto con la visita, dagli url contenuti in *VIS*.

Osservazione 4. *Questa tecnica è utilizzata dal crawler Nutch, le operazioni su disco sono effettuate tipicamente da architetture distribuite quali map-reduce.*

Osservazione 5. *È possibile mantenere all'interno di V solo gli hash dei siti in modo da risparmiare memoria, a patto di ordinare gli url di F secondo il loro hash e di effettuare i confronti sugli url hashati.*