Architetture e Programmazione dei Sistemi di Elaborazione
Progetto a.a. 2017/18

Algoritmo *PageRank*
in linguaggio assembly x86-32+SSE e x86-64+AVX

Fabrizio Angiulli
fabrizio.angiulli@unical.it

1 Decrizione del problema

Dato un grafo orientato $G = (V, E)$, con $V = \{1, 2, \ldots, n\}$ insieme di n nodi (per brevità nel seguito i nodi sono denotati con i numeri da 1 and n, per cui un nodo i è un naturale $i \in V$) ed E insieme di m archi, un problema fondamentale è determinare l'importanza o centralità di ciascun nodo all'interno del grafo.

Ad esempio, se il grafo codifica una rete sociale, i nodi rappresentano gli utenti e gli archi la relazione di follower, mentre i nodi centrali indicano gli utenti più influenti. Come altro esempio, se il grafo codifica il Web, i nodi rappresentano pagine Web e gli archi i link ad altre pagine, mentre i nodi più importanti corrispondono alle pagine più interessanti.

Nel seguito, $\mathbf{A}$ denota la matrice di adiacenza del grafo G, ovvero la matrice $n \times n$ tale che $a_{ij} = 1$ se $(i, j) \in E$ e $a_{ij} = 0$ altrimenti $(1 \leq i, j \leq n)$.

Eigenvector centrality. L'*eigenvector centrality* è una misura di centralità utilizzata nell'ambito delle reti sociali. L'idea alla base di questa misura è che l'importanza di un nodo dipende dall'importanza dei suoi vicini. Dato un parametro λ, l'eigenvector centrality (centralità per brevità nel seguito) c_i del nodo i è proporzionale alla somma delle centralità dei sui *incoming neighbors*:

$$c_i = \frac{1}{\lambda} \sum_{j=1}^{n} a_{ji}\, c_j.$$

Sia $\mathbf{c} = (c_1, c_2, \ldots, c_n)^T$ il vettore delle centralità, allora in notazione matriciale

$$\lambda \mathbf{c} = \mathbf{A}^T \mathbf{c}.$$

Questo significa che $\mathbf{c}$ è un autovettore della matrice $\mathbf{A}^T$ e λ è il corrispondente autovettore. Uno dei requisiti richiesti dalla soluzione è che le centralità siano valori positivi. In particolare, in accordo al Teorema di *Perron-Frobenius*, se $\mathbf{A}$ rappresenta la matrice di adiacenza di un grafo fortemente connesso, l'autovettore $\boldsymbol{v}^*$ (detto anche autovettore principale) associato all'autovalore più grande $\lambda_{\max}$ di $\mathbf{A}^T$ è l'unico ad avere componenti tutte strettamente positive. Quindi l'eigenvector centrality viene definita in termini di questo autovettore: $\mathbf{c} = \boldsymbol{v}^*$.

PageRank. L'algoritmo *PageRank*, che viene utilizzato con successo dai motori di ricerca per determinare l'importanza delle pagine Web, si basa sul concetto di centralità. La definizione di PageRank di una pagina Web corrisponde ad una variante del concetto di eigenvector centrality.

In particolare il PageRank π_i di una pagina Web i è direttamente proporzionale al PageRank π_j di ogni suo incoming neighbor j ed inversamente proporzionale al numero di pagine d_j a cui j fa riferimento:

$$\pi_i = \sum_{j=1}^{n} \frac{a_{ji}}{d_j} \pi_j,$$

dove d_j denota il numero di archi uscenti da j (anche detto *outdegree*).

Sia $\mathbf{P}$ la *matrice* (*delle probabilità*) *di transizione* $n \times n$ tale che $p_{ij} = 1/d_i$ $(1 \leq i, j \leq n)$. Intuitivamente, il PageRank della generica pagina i si può riguardare come la probabilità che un utente del Web (*surfer*), che assumiamo partire da una pagina arbitraria e visitare le pagine seguendo i link in accordo alle probabilità definite da $\mathbf{P}$ (*random walk*), si trovi a visitare la pagina i una volta trascorso un tempo sufficientemente lungo.

In termini di processi aleatori, il PageRank $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)^T$ delle pagine Web si può definire come la distribuzione di probabilità stazionaria $\boldsymbol{\pi}$ della catena di Markov omogenea avente matrice di transizione di probabilità $\mathbf{P}$, ovvero il vettore (a componenti non-negative e somma unitaria) che soddisfa la seguente proprietà

$$\boldsymbol{\pi} = \mathbf{P}^T \boldsymbol{\pi}.$$

Qundi $\boldsymbol{\pi}$ è un autovettore di $\mathbf{P}^T$ associato all'autovalore $\lambda = 1$ di $\mathbf{P}$ (che nel caso del Web corrisponde all'autovalore $\lambda_{\max} = 1$ di valore massimo di $\mathbf{P}$), detto *autovettore principale*. Affinchè $\boldsymbol{\pi}$ esista unico e con componenti tutte positive (in generale non è desiderabile avere pagine con PageRank pari a 0) devono essere soddisfatte alcune condizioni.

Innanzitutto bisogna garantire che $\mathbf{P}$ sia una matrice di transizione valida, ovvero a componenti non-negative $(p_{ij} \geq 0)$ e tale che la somma di ogni riga sia pari ad 1 $(\sum_j p_{ij} = 1)$. Poichè l'ultima proprietà non è soddisfatta dalle righe associate alle pagine prive di link in uscita (con outdegree $d_i = 0$), la matrice $\mathbf{P}$ viene sostituita con la matrice $\mathbf{P}'$:

$$\mathbf{P}' = \mathbf{P} + \mathbf{D},$$

con $\mathbf{D} = \boldsymbol{\delta} \cdot \boldsymbol{v}$, dove $\boldsymbol{v} = (1/n, 1/n, \ldots, 1/n)$ e $\boldsymbol{\delta} = (\delta_1, \delta_2, \ldots, \delta_n)^T$, $\delta_i = 1$ se $d_i = 0$ e $\delta_i = 0$ se $d_i > 0$ $(1 \leq i \leq n)$. In pratica, le righe di $\mathbf{P}'$ associate a nodi con link uscenti coincidono con le corrispondenti righe di $\mathbf{P}$, mentre le restanti righe sono poste uguali a $\boldsymbol{v}$. In termini del random walk, questo significa che se la pagina corrente è priva di link in uscita, l'utente selezionerà la prossima pagina da visitare in maniera casuale.

Si noti che essendo gli elementi di $\boldsymbol{v}$ tutti pari ad $1/n$, ogni pagina avrà la stessa probabilità di essere selezionata. Poichè $\boldsymbol{v}$ determina la probabilità di selezionare casualmente una certa pagina, il vettore $\boldsymbol{v}$ viene chiamato anche vettore di *personalizzazione*. In effetti, è possibile variare tali probabilità, assegnando ad esempio valori maggiori alle pagine di maggior interesse, semplicemente modificando $\boldsymbol{v}$ (garantendo che abbia comunque elementi tutti non-negativi e somma unitaria).

Un'altra proprietà da garantire è che G sia fortemente connesso. Poichè questa proprietà non è in generale posseduta dal Web, si può ovviare a questo problema imponendo che le componenti della matrice di transizione siano tutte non-negative. Quindi, la matrice $\mathbf{P}'$ viene a sua volta sostituita con la matrice $\mathbf{P}''$. Dato un parametro $c \in [0, 1]$, $\mathbf{P}''$ è definita come

$$\mathbf{P}'' = c\mathbf{P}' + (1 - c)\mathbf{E},$$

dove $\mathbf{E} = \boldsymbol{u} \cdot \boldsymbol{v}$, con $\boldsymbol{u} = (1, 1, \ldots, 1)^T$ vettore composto da n valori 1. In accordo a $\mathbf{P}''$, con probabilità c l'utente seguirà uno dei link in uscita dalla pagina corrente, mentre con probabilità

$1 - c$ sceglierà casualmente la prossima pagina da visitare secondo la probabilità definita da $\mathbf{E}$ (si noti che il secondo tipo di transizione viene effettuato con probabilità 1 se la pagina non ha link in uscita). Per questo motivo la matrice $\mathbf{E}$ viene anche detta matrice di *teletrasporto*.

Metodo delle potenze. L'algoritmo standard per il calcolo del PageRank (e più in generale dell'autovettore principale) è un metodo iterativo noto come metodo delle potenze (*power iteration*). L'algoritmo parte dall'approssimazione iniziale della soluzione $\boldsymbol{\pi}^{(0)} = \boldsymbol{v}$ e ad ogni iterazione $k \geq 1$ calcola un'approssimazione più accurata $\boldsymbol{\pi}^{(k+1)} = (\mathbf{P}'')^T \boldsymbol{\pi}^{(k)}$. L'algoritmo termina quando la differenza $\Delta^{(k)} = \|\boldsymbol{\pi}^{(k)} - \boldsymbol{\pi}^{(k+1)}\|_1$ tra due soluzioni successive è minore di una soglia ϵ (valore di default $\epsilon = 10^{-5}$), ovvero $\Delta^{(k)} < \epsilon$, restituendo come soluzione $\boldsymbol{\pi}^* = \frac{\boldsymbol{\pi}^{(k+1)}}{\|\boldsymbol{\pi}^{(k+1)}\|_1}$.

Si può dimostrare che la velocità di convergenza del metodo è legata al valore del parametro c. In particolare, la velocità di convergenza è inversamente proporzionale a c. Tuttavia, poichè c rappresenta la frazione del random walk dettata dall'attraversamento dei link, non è conveniente utilizzare valori piccoli per tale parametro. Per questo motivo, il valore di default di c è posto ad un compromesso che fornisce risultati di buona qualità nella pratica, ovvero $c = 0.85$.

2 Descrizione dell'attività progettuale

Obiettivo del progetto è mettere a punto un'implementazione dell'algoritmo *PageRank* in linguaggio C e di migliorarne le prestazioni utilizzando le tecniche di ottimizzazione basate sull'organizzazione dell'hardware.

L'ambiente sw/hw di riferimento è costituito dal linguaggio di programmazione C (`gcc`), dal linguaggio assembly x86-32+SSE e dalla sua estensione x86-32+AVX (`nasm`) e dal sistema operativo Linux (`ubuntu`).

In particolare, l'algoritmo deve prevedere due formati per l'input:

1. *sparse* (default, oppure parametro `-sparse`), nel qual caso riceve in ingresso il grafo G sotto forma di coppie di nodi (i, j) (archi), con i e j numeri interi a 32 bit, oppure
2. *dense* (parametro `-dense`), nel qual caso riceve in ingresso direttamente la matrice di transizione $\mathbf{P}''$, codificata utilizzando numeri floating-point a precisione doppia.

Inoltre, nel caso di formato di input *sparse*, l'algoritmo deve poter lavorare in due diverse modalità di rappresentazione dei numeri in virgola mobile:

1. *single* (default, oppure parametro `-single`): l'algoritmo esegue i calcoli usando numeri floating-point a precisione singola (32 bit), oppure
2. *double* (parametro `-double`): l'algoritmo esegue i calcoli usando numeri floating-point a precisione doppia (64 bit).

In ogni caso, il vettore dei PageRank dovrà essere restituito usando la precisione doppia (quindi se è stato calcolato utilizzando la precisione singola, il vettore risultato dovrà essere convertito in un vettore a precisione doppia).

Di seguito si riportano ulteriori linee guida per il corretto svolgimento del progetto:

- Si consiglia di affrontare il progetto nel seguente modo:
    1. Inannzitutto, codificare l'algoritmo interamente in linguaggio C, possibilmente come sequenza di chiamate a funzioni;

2. Sostituire una o più (idealmente tutte le) funzioni scritte in linguaggio ad alto livello con corrispondenti funzioni scritte in linguaggo assembly.

   Ciò consentirà (1) di verificare che l'algoritmo che si intende ottimizzare è corretto, (2) di gestire meglio la complessità del progetto e (3) di avere la possibilità di presentare il progetto (funzionante) anche se non si è riusciti a riscrivere in linguaggio assembly tutte le funzioni introdotte.

- Il numero di nodi n ed il numero di archi m, nonché i parametri c (parametro `-c`) ed ϵ (parametro `-eps`), devono poter variare da esecuzione ad esecuzione. I test di prestazione verranno effettuati con valori di n ed m dell'ordine delle centiaia di migliaia/milioni compatibilmente con i tempi di risposta dell'implementazione.

- La versione base dell'algoritmo risolutivo dev'essere basata sul metodo delle potenze (default, oppure parametro `-nopt`).

  È possibile ottimizzare l'algoritmo utilizzando metodi numerici per accellerare la convergenza del metodo delle potenze oppure altri metodi numerici per il calcolo degli autovalori. Ottimizzazioni di questo tipo devono poter essere eseguite sono in presenza del parametro `-opt`. I test di prestazione verranno effettuati sia nella modalità `-nopt` che in quella `-opt`.

- Al fine di migliorare la valutazione dell'attività progettuale è possibile presentare nella relazione un confronto tra le prestazioni delle versioni intermedie, ognuna delle quali introduce una particolare ottimizzazione, e finale del codice. Obiettivo del confronto è sostanziare la bontà delle ottimizzazioni effettuate.

- Occorre lavorare in autonomia e non collaborare con gli altri gruppi. Soluzioni troppo simili riceveranno una valutazione negativa. I progetti verranno messi in competizione.

- Sono richieste due soluzioni software, la prima per l'architettura x86-32+SSE e la seconda per l'architettura x86-64+AVX.

  Per i dettagli riguardanti la redazione del codice fare riferimento ai files `pagerank32c.c`, `pagerank32.nasm`, `runpagerank32` (versione `x86-32+SSE`) e `pagerank64c.c`, `pagerank64.nasm`, `runpagerank64` (versione `x86-64+AVX`) disponibili sulla piattaforma `didattica.dimes.unical.it`.

  Per l'interfacciamento tra programmi in linguaggio C e programmi in linguaggio assembly fare riferimento al documento allegato alla descrizione del progetto.

- Le soluzioni base non devono far uso delle istruzioni `OpenMP`. Opzionalmente, è possibile consegnare delle soluzioni aggiuntive che facciano uso anche di istruzioni `OpenMP`.

- Il software dovrà essere corredato da una relazione. Per la presentazione del progetto è possibile avvalersi di slide.

- Prima della data di consegna del progetto verranno pubblicate le convenzioni da rispettare riguardanti i nomi e la collocazione dei file/directory al fine della compilazione e l'esecuzione dei codici di programma mediante script appositamente predisposti. **Dato l'elevato numero di progetti, sarà cura del candidato accertarsi di aver rispettato pienamente le convenzioni di consegna, pena l'impossibilità di eseguire il progetto e la conseguente valutazione negativa dell'elaborato.**

Buon lavoro!

Interfacciamento tra linguaggio C e linguaggio Assembly

Fabrizio Angiulli

Abstract

Il presente documento fornisce una breve descrizione delle convenzioni di chiamata delle funzioni in linguaggio C (*C Calling Conventions*) utilizzate dal compilatore GNU, sia con riferimento all'architettura x86-32 che all'architettura x86-64. La conoscenza di tali convenzioni consente di interfacciare programmi scritti in linguaggio C con programmi scritti in linguaggio Assembly, mediante l'invocazione di funzioni Assembly da funzioni C e viceversa.

1 C Calling Conventions a 32 bit

I programmi in linguaggio C su architettura x86-32 utilizzano le seguenti convenzioni per il passaggio dei parametri tra funzioni:

- I parametri passati alle funzioni vengono inseriti (*push*) nello stack in ordine inverso. Quindi, data la chiamata di funzione `func(a, b, c)`, il valore del parametro `c` viene posto nello stack per primo, quello di `b` per secondo e infine quello di `a` per terzo;
- La funzione chiamante assume che il contenuto dei registri ESP, EBP, EBX, ESI e EDI non venga alterato dalla funzione chiamata. Questo non significa che questi registri non possono essere utilizzati dalla funzione chiamata, ma piuttosto che il loro valore al termine dell'esecuzione della funzione chiamata deve coincidere con il valore all'inzio dell'esecuzione. Tutti gli altri registri possono essere modificati senza nessun accorgimento dalla funzione chiamata;
- La funzione restituisce il valore di ritorno nel registro EAX se la dimensione del tipo di ritorno è minore o uguale a 32 bit. I puntatori (a qualsiasi tipo) sono da assimilarsi ad interi a 32 bit. Nel caso il valore di ritorno sia a 64 bit, viene restituito nella coppia di registri EAX (i 32 bit meno significativi) e EDX (i 32 bit più significativi). I valori in virgola mobile (ovvero `float` oppure `double`) vengono posti in cima allo stack dei registri x87 (che non abbiamo trattato). Le strutture ed ogni altro tipo avente dimensione maggiore di 32 bit devono essere restituite per riferimento, ovvero la funzione ne restituirà l'indirizzo di partenza nel registro EAX;
- La funzione chiamata non rimuove i parametri dallo stack. La rimozione dei parametri dallo stack è a carico della funzione chiamante e può essere effettuata sommando allo stack pointer ESP la dimensione dei parametri passati.

Ad esempio, si consideri il seguente frammento di codice C, in cui il `main` chiama una funzione esterna di nome `asmfunc` scritta in linguaggio assembly x86-32:

```
extern int asmfunc(float* A, int m, int n);

int main() {
    int m = 10, n = 2;
    float* A = calloc(m*n,sizeof(float));
    int y = asmfunc(A,m,n);
    ...
}
```

Di seguito si riporta la struttura generale della funzione `asmfunc`:

```
global asmfunc       ; rende la funzione visibile all'esterno

; Posizione dei parametri nel Recordi di Attivazione della funzione
; (i primi 8 bytes sono occupati dall'indirizzo di ritorno e da EBP)
;
A          equ       8                    ; puntatore a float, occupa 32 bit (4 bytes)
m          equ       12                   ; intero a 32 bit
n          equ       16                   ; intero a 32 bit

asmfunc:
           ;
           ; sequenza di ingresso nella funzione
           ;
           push      ebp                  ; salva il Base Pointer
           mov       ebp, esp             ; il Base Pointer punta al Record di Attivazione corrente
           push      ebx                  ; salva i registri da preservare
           push      esi
           push      edi
           ;
           ; lettura dei parametri dal Recordi di Attivazione
           ;
           mov       ..., [ebp+A]         ; legge A
           mov       ..., [ebp+m]         ; legge m
           mov       ..., [ebp+n]         ; legge n
           ;
           ; corpo della funzione
           ;
           ...
           mov       eax, <valore-di-ritorno>; asmfunc restituisce un numero intero
           ;
           ; sequenza di uscita dalla funzione
           ;
           pop       edi                  ; ripristina i registri da preservare
           pop       esi
           pop       ebx
           mov       esp, ebp             ; ripristina lo Stack Pointer
           pop       ebp                  ; ripristina il Base Pointer
           ret                            ; ritorna alla funzione chiamante
```

Per completezza, di seguito si mostra come il compilatore `gcc` traduce in assembly la chiamata della funzione `asmfunc`:

```
main:
         ...
         push     <valore-di-n>       ; posiziona i parametri nello stack
         push     <valore-di-m>
         push     <indirizzo-di-A>
         call     asmfunc             ; chiama la funzione asmfunc
         add      esp, 12             ; rimuove i parametri dallo stack
         ...
```

2 C Calling Conventions a 64 bit

I programmi in linguaggio C su architettura x86-64 utilizzano le seguenti convenzioni per il passaggio dei parametri tra funzioni:

- I primi sei parametri interi (scorrendo l'elenco dei parametri da sinistra verso destra) vengono passati, rispettivamente, nei registri RDI, RSI, RDX, RCX, R8 ed R9. Ulteriori parametri interi vengono passati sullo stack. I registri di cui sopra, insieme ai registri RAX, R10 ed R11 possono essere modificati dalla funzione chiamata senza la necessità di ripristinarli ai loro valori originari;
- I valori di ritorno interi vengono restituiti nei registri RAX e RDX;
- Gli argomenti floating-point vengono passati nei registri da XMM0 a XMM7. I valori di ritorno floating point vengono restituiti nei registri XMM0 ed XMM1;
- Tutti i registri SSE ed x87 possono essere alterati dalla funzione chiamata.

Ad esempio, nel caso della chiamata della funzione

`void func(long a, double b, int c)`,

il parametro `a` viene passato in RDI, il parametro `b` viene passato in XMM0, mentre il parametro `c` viene passato in RSI.