Progetto Laboratorio 2B 2023/24

Versione 1.6

June 25, 2024

Questo documento descrive solamente quello che deve essere calcolato all'interno del progetto. Per la definizione di PageRank e la sua importanza fate riferimento ai lucidi fatti a lezione (file `pagerank.pdf`) oppure all'abbondante documentazione su internet (ad esempio partendo da Wikipedia).

Il progetto deve essere svolto da ognuno di voi individualmente: la presenza di porzioni di codice con alta similarità comporta l'annullamento della consegna e l'assegnazione di un diverso progetto. Il corso ha anche l'obiettivo di mettervi in grado di leggere autonomamente la documentazione quindi è normale che sia richiesto usare funzioni che non abbiamo visto in dettaglio a lezione. Lo sviluppo va fatto su macchine Linux: MacOS non supporta ad esempio i semafori unnamed, e Windows è ancora meno compatibile. La correzione avviene sulla macchina `laboratorio2.di.unipi`.

1 Calcolo di PageRank

L'input dell'algoritmo PageRank è un grafo orientato G di N nodi identificati dagli interi $0, \ldots, N-1$. Identifichiamo il grafo G con l'insieme dei suoi archi; ogni arco è una coppia (i, j) con $0 \leq i, j < N$; tale coppia rappresenta l'arco orientato dal nodo i al nodo j. Si suppone che non esistano archi duplicati o archi in cui l'origine coincide con la destinazione.

Per $j = 1, \ldots, N$ definiamo

$$\mathsf{IN}(j) = \{i \mid (i, j) \in G\} \tag{1}$$

che è l'insieme dei nodi i da cui esce un arco che ha destinazione j e

$$\mathsf{out}(j) = |\{i \mid (j, i) \in G\}| \tag{2}$$

che rappresenta il numero di archi uscenti dal nodo j. Infine, definiamo l'insieme dei nodi *dead end* cioè che non hanno archi uscenti:

$$\mathsf{DE} = |\{i \mid out(i) = 0\}|. \tag{3}$$

Fissata questa terminologia possiamo esprimere in maniera formale come avviene il calcolo del vettore PageRank dato il grafo G. Sia d, $0 < d < 1$ la probabilità di con cui il random surfer decide di seguire uno dei link del nodo corrente; con probabilità $1 - d$ invece salta ad un altro nodo del grafo scelto a caso in maniera uniforme. Il parametro d è chiamato nella letteratura *damping factor*; nell'articolo originale dove è stato proposto PageRank è suggerito di prendere $d = 0.85$. Il calcolo di PageRank consiste nel calcolo della successione di vettori colonna N-dimensionali $X^{(1)}, X^{(2),\cdots}$ definita dalla relazione $X^{(1)} = (1/N, 1/N, \ldots, 1/N)^T$, e, per $j = 1, \ldots, N$ la j-esima componente di $X^{(t+1)}$ è data da

$$X_j^{(t+1)} = \frac{1-d}{N} \; + \; \frac{d}{N} \sum_{i \in \mathsf{DE}} X_i^{(t)} \; + \; d \sum_{i \in \mathsf{IN}(j)} \frac{X_i^{(t)}}{\mathsf{out}(i)} \tag{4}$$

Osserviamo che:

- il termine $\frac{1-d}{N}$ deriva dal teleporting; non dipende né da j né da t e di conseguenza lo possiamo calcolare una volta per tutte all'inizio dell'algoritmo
- il termine $\frac{d}{N} \sum_{i \in \mathsf{DE}} X_i^{(t)}$ è il contributo dei nodi *dead end* dipende da t ma non da j. Di conseguenza lo possiamo calcolare una volta sola all'inizio dell'iterazione $t + 1$

- il termine $d \sum_{i \in \mathsf{IN}(j)} \frac{X_i^{(t)}}{\mathsf{out}(i)}$ dipende invece sia da t che da j e rappresenta la parte più costosa del calcolo del nuovo vettore $X^{(t+1)}$; notiamo che essa contiene un termine per ogni nodo che ha un arco diretto al nodo j.

Riassumendo, per velocizzare il conto, all'inizio dell'iterazione $t+1$ possiamo calcolare la somma $S_t = \sum_{i \in \mathsf{DE}} X_i^{(t)}$, e per tutti gli indici i tali che $\mathsf{out}(i) > 0$ i valori $Y_i^{(t)} = X_i^{(t)}/\mathsf{out}(i)$. Dopo aver fatto queste operazioni, la formula (4) puo essere riscritta

$$X_j^{(t+1)} = \frac{1-d}{N} + d \sum_{i \in \mathsf{IN}(j)} Y_i^{(t)} + \frac{d}{N} S_t$$

Ricordiamo che il valore $X_i^{(t)}$ rappresenta la probabilità che il random surfer sia nel nodo i dopo t passi. La teoria dice questi valori convergono cioè all'aumentare di t c'è pochissima differenza tra i valori di $X^{(t)}$ e quelli di $X^{(t+1)}$ quando questo succede possiamo terminare le iterazioni in quanto abbiamo ottenuto una buona approssimazione del PageRank.

Dal punto di vista pratico si definisce l'errore al passo $t+1$ come

$$e_{t+1} = \sum_{i=1}^{N} |X_i^{(t+1)} - X_i^{(t)}|$$

e all'inizio dell'algoritmo si fissa una tolleranza ϵ (ad esempio 10^{-6}) e non appena si verifica che $e_{t+1} < \epsilon$ si interrompe l'algoritmo e si restituisce l'ultimo vettore calcolato $X^{(t+1)}$ come risultato del calcolo del PageRank. Per sicurezza si deve fissare fissare un limite massimo al numero di iterazioni e terminare comunque quando questo limite è stato raggiunto.

2 Implementazione dell'algoritmo in C

La matrice di adiacenza del grafo di input è rappresentata da un file di testo nel formato *Matrix Market*. Potete leggere i dettagli del formato nella documentazione ufficiale, ma per realizzare il progetto sono sufficienti le seguenti informazioni (si veda il file `9nodi.mtx` per un esempio):

- le linee che iniziano con il carattere `%` sono commenti e devono essere ignorate;
- la prima linea non commento deve contenere tre interi separati da uno spazio: r, c, ed n; nel caso di una matrice di adiacenza r e c devono entrambi coincidere con N, il numero di nodi del grafo, mentre n è il numero totale di archi;
- le altre righe che non sono commenti devono contenere due interi i e j separati da uno spazio che identificano l'origine e la destinazione di un arco. Nel formato Matrix Market i nodi sono identificati dagli interi da 1 a N; seguendo la notazione dei moderni linguaggi di programmazione noi identificheremo i nodi con gli interi tra 0 e $N-1$. Di conseguenza durante la lettura del file si deve sottrarre 1 da tutti gli identificatori: quando si legge la coppia di interi i j, nel programma si deve registrare la presenza di un arco dal nodo $i-1$ al nodo $j-1$.

Importante: per il calcolo di PageRank eventuali archi duplicati presenti nel file o archi in cui l'origine coincide con la destinazione devono essere scartati e quindi non rappresentati nel grafo.

Nella prima parte del programma devono essere letti gli archi del file e deve essere inizializzata una struttura che rappresenta il grafo. La struttura deve contenere almeno i seguenti campi:

```
typdef struct {
    int N;       // numero dei nodi del grafo
    int *out;    // array con il numero di archi uscenti da ogni nodo
    inmap *in;   // array con gli insiemi di archi entranti in ogni nodo
} grafo;
```

Ad esempio se `g` di tipo `grafo *` è il puntatore al grafo corrente, `g->N` è il numero di nodi, e per $i = 0, \ldots, N-1$ `g->out[i]` è il numero di nodi uscenti dal nodo i mentre `g->in[i]` deve rappresentare i nodi che entrano nel nodo i. Si noti che `g->in` è un array di elementi di un tipo `inmap` che dovete definire voi in maniera appropriata in modo che `g->in[i]` possa rappresentare i nodi da cui esce arco con destinazione i; ad esempio `g->in[i]` può essere una linked list o un array di interi.

Lo spazio totale utilizzato per rappresentare il grafo deve essere proporzionale al numeri di archi! Non è accettata quindi una soluzione in cui `g->in[i]` è sempre un array di lunghezza N perché in tal caso lo spazio totale occupato sarebbe proporzionale a N^2. Si tenga conto che nei problemi tipici che si devono risolvere N è almeno dell'ordine di 10^6 e il numero dei archi dell'ordine di 10^8. Quindi memorizzare gli archi richiede qualche GB, mentre una soluzione che occupa spazio dell'ordine di $N^2 \approx 10^{12}$ non sarebbe in grado di girare su una macchina standard. Ricordare che eventuali archi duplicati presenti nel file o archi in cui l'origine coincide con la destinazione devono essere scartati e quindi non rappresentati nel grafo.

Dopo la lettura dei dati e la creazione dell'oggetto `grafo` deve essere realizzato il calcolo del vettore PageRank con una funzione che deve avere *esattamente* il seguente prototipo:

```
double *pagerank(grafo *g, double d, double eps, int maxiter, int taux, int *numiter);
```

dove `g` rappresenta il grafo, `d` il damping factor, `eps` la tolleranza ϵ usata per arrestare le iterazioni, `maxiter` il numero massimo di iterazioni, e `taux` il numero di thread ausiliari a disposizione. La funzione deve scrivere in `*numiter` il numero di iterazioni effettuate e restituire il vettore di `double` contenente il PageRank calcolato utilizzando il procedimento illustrato in precedenza. La funzione dovrà allocare dei vettori di `g->N` double per effettuare il calcolo e deallocarli tutti tranne quello che contiene il risultato finale che viene restituito al chiamante. Si noti che è sufficiente lavorare con al massimo 3 array di double, ad esempio `x`, `y` e `xnext`, per mantenere rispettivamente i vettori $X^{(t)}$, $Y^{(t)}$, $X^{(t+1)}$.

2.1 Utilizzo dei thread

Il programma deve utilizzare un numero `t` di thread ausiliari (il valore `t` è passato sulla linea di comando); questi thread devono essere utilizzati sia nella fase di lettura dei dati, che nella fase di calcolo del PageRank.

Durante la fase di lettura il thread principale legge uno alla volta gli archi dal file di input (ricordiamo che ogni arco è una coppia di interi su una singola linea) e li scrive su un buffer-produttori consumatori. I thread ausiliari svolgono il ruolo di consumatori ed effettuano la gestione di ogni arco. Si osservi che la gestione dell'arco (i, j) richiede in particolare le seguenti operazioni:

- ignorare l'arco se $i = j$;
- aggiungere i all'insieme `g->in[j]` dei nodi entranti in j; questa operazione deve prima rilevare se l'arco è duplicato (cioè è già stato incontrato precedentemente) e in tal caso deve ignorarlo;
- se l'arco non è un duplicato si deve incrementare di 1 il numero `g->out[i]` di archi uscenti da i.

Al termine delle operazioni di lettura si può fare un'operazione di join dei thread consumatori per assicurarsi che il grafo sia completo prima di passare alla fase di calcolo.

Durante la fase di calcolo del PageRank deve essere svolta in parallelo ogni iterazione (4). In questo caso una singola unità di lavoro è il calcolo di una componente $X_j^{(t+1)}$ che può essere svolta in maniera indipendente per ogni j. Le varie componenti $X_j^{(t+1)}$ devono essere assegnate ai thread ausiliari utilizzando uno schema produttori/consumatori o un altro schema a vostra scelta (dato che gli identificatori delle unità di lavoro sono gli interi tra 0 e $N-1$ lo schema produttori/consumatori può essere sostituito da qualcosa di più semplice). E' importante però che la divisione del lavoro sia dinamica: il calcolo delle singole componenti $X_j^{(t+1)}$ puo richiedere tempi molto diversi, quindi appena un thread ha finito una computazione deve poter cominciarne una nuova. Inoltre, si devono parallelizzare le fasi successive al calcolo delle componenti di $X^{(t+1)}$, come ad esempio il calcolo dell'errore e_{t+1}, il calcolo del vettore ausiliario Y_t e del valore S_t. Un ulteriore vincolo da rispettare è che durante tutto il calcolo del PageRank i thread devono rimanere sempre attivi: prima della fine del calcolo non è ammesso fare una join e di conseguenza i thread si devono sincronizzare quando necessario utilizzando gli strumenti visti nel corso (semafori, condition variables, o altro).

Infine, deve essere presente un thread per la gestione dei segnali che nel caso riceva un segnale SIGUSR1 visualizzi su *stderr*: il numero corrente di iterazione, il nodo che ha al momento il maggiore PageRank, e il valore del corrispondente PageRank.

3 Input e output del programma

Si deve realizzare un eseguibile `pagerank` che deve poter essere invocato dalla linea di comando con le seguente sintassi:

```
pagerank [-k K] [-m M] [-d D] [-e E] [-t T] infile

positional arguments:
  infile      input file

options:
  -k K        show top K nodes (default 3)
  -m M        maximum number of iterations (default 100)
  -d D        damping factor (default 0.9)
  -e E        max error (default 1.0e-7)
  -t T        number of auxiliary threads (default 3)
```

per la gestione delle opzioni sulla linea di comando dovete usare la funzione di libreria `getopt(3)`. Si noti che tutte le opzioni posso essere omesse (in tal caso si devono utilizzare i valori di default sopra indicati) mentre il nome del file di input è obbligatorio.

Dopo la lettura del grafo si deve stampare su *stdout* il numero dei nodi, il numero dei nodi dead-end, e il numero degli archi. Al termine della computazione del PageRank del grafo devono essere stampati su *stdout* i `K` nodi con il PageRank più alto ed il rispettivo valore del PageRank, oltre al numero di iterazioni e alla somma di tutti i rank (che deve risultare uguale a 1). Per il formato dell'output si segua scrupolosamente il seguente esempio, relativo al file `web-Stanford.mtx`:

```
Number of nodes: 281903
Number of dead-end nodes: 172
Number of valid arcs: 2312497
Did not converge after 100 iterations
Sum of ranks: 1.0000   (should be 1)
Top 3 nodes:
  89072 0.012074
  226410 0.008650
  241453 0.007743
```

Il valore "Sum of ranks" è la somma dei rank di tutti i nodi del grafo ed, essendo i rank derivati da una distribuzione di probabilità, deve essere uguale a 1 (più in particolare la somma di tutte le componenti del vettore $X^{(t)}$ deve sempre essere uguale a 1). Non stampate altre informazioni su *stdout*: siete incoraggiati invece a stampare le informazioni di debug su *stderr* per facilitarvi la verifica della correttezza e la ricerca degli errori. Il programma `pagerank.py` esegue la computazione richiesta (senza usare thread o gestire i segnali) e potete usarlo per verificare la correttezza del vostro programma in C (solo per i grafi più piccoli a causa della lentezza intrinseca di Python).

In caso di errori (ad esempio: file di input non trovato, formato del file non valido,identificatori di nodi non validi, etc) il vostro programma deve terminare con un messaggio di errore esplicativo e un exit code diverso da 0 (usate liberamente le funzioni `termina` o `xtermina` fatte a lezione).

4 Consegna del progetto

La consegna deve avvenire esclusivamente mediante GitHub. Se non lo avete già createvi un account e create un repository *privato* dedicato a questo progetto. Aggiungete come collaboratore al repository l'utente

`Laboratorio2B` in modo che i docenti abbiano accesso al codice. IMPORTANTE: la consegna effettiva del progetto per un dato appello consiste nello scrivere l'url per la clonazione del vostro progetto nell'apposito contenitore consegna su moodle. L'url deve essere della forma `git@github.com:user/progetto.git` (*non* deve quindi iniziare per `https`). Dopo la data di consegna non dovete fare altri commit al progetto. Ricordatevi che almeno cinque giorni prima della data di consegna dovete iscrivervi all'appello su esami.unipi.it.

Il repository deve contenere tutti i file del progetto e il makefile necessario per la compilazione. Il makefile deve essere scritto in modo che la seguente sequenza di istruzioni sulla linea di comando (`progetto` indica il nome del repository, voi usate un nome meno generico):

```
git clone git@github.com:user/progetto.git testdir
cd testdir
make
./pagerank 9nodi.mtx
```

non restituisca errori e produca il seguente output:

```
Number of nodes: 9
Number of dead-end nodes: 2
Number of valid arcs: 11
Converged after 31 iterations
Sum of ranks: 1.0000   (should be 1)
Top 3 nodes:
  5 0.242186
  3 0.211610
  2 0.167547
```

(ci potrebbero essere delle differenze nelle ultime cifre decimali dei rank). Fate la verifica descritta qui sopra sulla macchina `laboratorio2.di.unipi.it`, partendo da una directory vuota. Se tali operazioni non danno il risultato corretto il progetto è considerato non sufficiente senza ulteriori approfondimenti e dovete ripresentarvi all'appello successivo.

Superata questa prima prova passate alle successive. Il comando:

```
valgrind ./pagerank -e 1e-9 -k5 9nodi.mtx 1> 9nodi.rk 2> 9nodi.log
```

deve generare un file `9nodi.rk` simile a `9nodi.sol`; potete usare il comando della shell `diff -bB 9nodi.rk 9nodi.sol` per confrontarli. Inoltre in `9nodi.log` non deve esserci nessun errore di valgrind o segnalazione di memory leak.

Altri test da effettuare prima della consegna:

```
valgrind ./pagerank web-Stanford.mtx -k8 1> web-Stanford.rk 2> web-Stanford.log
valgrind ./pagerank web-cnr.mtx -t5 -m 200 1> web-cnr.rk 2> web-cnr.log
valgrind ./pagerank web-Google.mtx -t8 -e 1e-5 1> web-Google.rk 2> web-Google.log
valgrind ./pagerank web-hudong.mtx -t8 -e 1e-4 1> web-hudong.rk 2> web-hudong.log
```

in tutti i casi il file di log non deve contenere errori/memory leak di valgrind e i risultati nei file `.rk` devono coincidere (a meno delle ultime cifre decimali) con i file `.sol` allegati al progetto.

Il repository deve contenere anche un file README.md contenente una breve relazione che descrive in quale modo sono stati utilizzati thread per parallelizzare l'algoritmo. Nel repository dovete mettere solamente i file veramente essenziali per la compilazione del progetto: quindi non i file `.o` gli eseguibili, i file di configurazione o i file di test con estensione `.mtx` (a parte `9nodi.mtx` che va invece incluso).

5 Progetto completo

Chi deve realizzare il progetto completo, oltre a programma `pagerank` descritto sopra deve realizzare nella stessa directory un server e un client scritti in Python che comunicano attraverso i socket. Si vuole simulare la situazione in cui un server su cui gira un programma ottimizzato per il calcolo del PageRank e a cui i client possono inviare dei grafi per ottenere quali sono i nodi con il PageRank più alto. Più in dettaglio, il

client legge la descrizione del grafo in un file locale e la invia al server che la salva in un file temporaneo e invoca `pagerank` sul file temporaneo, inviando il risultato ottenuto al client.

Si ricorda che nelle comunicazioni tra client e server può essere necessario inviare la lunghezza dei messaggi, queste informazioni aggiuntive non saranno menzionate nella descrizione qui sotto.

5.1 Il server `graph_server.py`

Il server `graph_server.py` si deve mettere in attesa su `127.0.0.1` sulla porta `5XXXX` dove `XXXX` sono le ultime quattro cifre del vostro numero di matricola. Ad ogni client che si connette il server deve assegnare un thread dedicato. Stabilita la connessione, il client invia al server la descrizione di un grafo con il seguente formato: 1) un intero n che indica il numero di nodi, 2) un intero a che indica il numero degli archi, 3) a coppie di interi che rappresentano l'origine e la destinazione di ogni arco. Si può supporre che tutti gli interi in questa sequenza siano rappresentabili con 32 bit.

Il server deve scrivere i dati del grafo in un file di testo temporaneo nel formato MatrixMarket descritto sopra (potete omettere le linee di commento). Dato che i grafi possono consistere in diversi MBs, il server non deve attendere la fine della trasmissione per iniziare a scrivere: deve invece leggere gli archi a gruppi di al massimo 10 per volta e dopo aver letto ogni gruppo di archi deve immediatamente scriverlo nel file temporaneo; il numero di nodi e archi devono essere scritti prima di leggere gli archi. Il server deve scartare eventuali archi in cui gli identificatori dei nodi sono non validi cioè non compresi tra 1 e n.

Dopo aver completato la lettura del grafo e la scrittura nel file temporaneo, il server deve usare la funzione `subprocess.run` per invocare l'eseguibile `pagerank` sul file temporaneo catturando stdout e stderr di `pagerank` nel `CompletedProcess` restituito da `subprocess.run`: vedere gli esempi fatti in `duplicati.py` e `linee.py`. Se `pagerank` è terminato con successo (quindi con exit code 0), il server deve inviare al client l'intero 0 seguito dallo stdout di `pagerank`, altrimenti il server invia invia al client l'exit code di `pagerank` seguito dallo stderr di `pagerank`. A questo punto il server può chiudere la connessione con il client e cancellare il file temporaneo contenente il grafo. Il programma `pagerank` può esser chiamato con le opzioni di default, quindi con solamente il nome del file temporaneo sulla linea di comando. Per creare il file temporaneo utilizzate il modulo tempfile.

Il server deve usare il modulo logging per la gestione di un file di log di nome `server.log`. Per ogni connessione, il server deve scrivere sul file di log 1) il numero di nodi del grafo, 2) il nome del file temporaneo usato, 3) il numero di archi scartati, 4) il numero di archi validi ricevuti, 5) l'exit code restituito da `pagerank`. Il server deve rimanere costantemente in attesa sulla porta assegnata e deve terminare, facendo l'opportuno shutdown, solamente quando viene ricevuto il segnale SIGINT (il segnale SIGINT in Python genera l'eccezione `KeyboardInterrupt`). Alla ricezione di SIGINT, il server deve attendere la terminazione di tutti i thread che ha lanciato in precedenza per la gestione delle singole connessioni; come realizzare questa attesa dipende da quale meccanismo avete usato per generare i thread (un pool oppure un insieme di thread singoli). Dopo la terminazione dei thread ausiliari il server deve stampare su stdout il messaggio "Bye dal server" e terminare lui stesso.

5.2 Il client `graph_client.py`

Il programma `graph_client.py` deve prendere in input sulla linea di comando un elenco di nomi di un file in formato `mtx`. Deve poi generare un numero di thread ausiliari pari al numero di nomi di file, e ogni thread deve connettersi al server e inviare i dati del grafo letti dal suo file (numero dei nodi, numero degli archi, elenco degli archi come descritto sopra). Successivamente, ogni thread si deve mettere in attesa della risposta del server (exit code di `pagerank` e il relativo messaggio), stampare su stdout la risposta ricevuta e un messaggio "Bye" e terminare. Per distinguere l'output dei diversi thread ogni riga del relativo output deve iniziare con il relativo nome del file. Quindi una possibile output del client potrebbe essere:

```
graph_client 9nodi.mtx web-Stanford.mtx
9nodi.mtx  Exit code: 0
9nodi.mtx  Number of nodes: 9
9nodi.mtx  Number of dead-end nodes: 2
9nodi.mtx  Number of valid arcs: 11
9nodi.mtx  Converged after 31 iterations
```

```
9nodi.mtx  Sum of ranks: 1.0000   (should be 1)
9nodi.mtx  Top 3 nodes:
9nodi.mtx    5 0.242186
9nodi.mtx    3 0.211610
9nodi.mtx    2 0.167547
9nodi.mtx  Bye
web-Stanford.mtx  Exit code: 0
web-Stanford.mtx  Number of nodes: 281903
web-Stanford.mtx  Number of dead-end nodes: 172
web-Stanford.mtx  Number of valid arcs: 2312497
web-Stanford.mtx  Did not converge after 100 iterations
web-Stanford.mtx  Sum of ranks: 1.0000   (should be 1)
web-Stanford.mtx  Top 3 nodes:
web-Stanford.mtx    89072 0.012074
web-Stanford.mtx    226410 0.008650
web-Stanford.mtx    241453 0.007743
web-Stanford.mtx  Bye
```

naturalmente è possibile che l'output dei due thread sia mischiato: l'importante che ci sia l'informazione completa per ogni singolo input file. Il client non deve scrivere altre informazioni su stdout o stderr: eventuali informazioni di debug scrivetele su un file di log utilizzando come per il server il modulo logging.

5.3 Consegna del progetto completo

I file `graph_server.py` e `graph_client.py` devono essere contenuti nello stesso repository usato per `pagerank`. Devono essere file eseguibili: se il permesso di esecuzione non viene mantenuto sul repository è probabilmente necessario dare il comando `git config core.filemode true` come indicato in questo post. Dopo aver verificato che il programma `pagerank` funziona correttamente, prima di consegnare verificate che eseguendo i seguenti comandi:

```
./graph_server.py &                    # viene lanciato il server i background
./graph_client.py 21archi.mtx          # viene lanciato il client su un singolo file
pkill -INT  -f graph_server.py         # invia SIGINT al server per farlo terminare
```

si ottenga il seguente output

```
21archi.mtx  Exit code: 0
21archi.mtx  Number of nodes: 9
21archi.mtx  Number of dead-end nodes: 2
21archi.mtx  Number of valid arcs: 11
21archi.mtx  Converged after 31 iterations
21archi.mtx  Sum of ranks: 1.0000   (should be 1)
21archi.mtx  Top 3 nodes:
21archi.mtx    5 0.242186
21archi.mtx    3 0.211610
21archi.mtx    2 0.167547
21archi.mtx  Bye
Bye dal server
```

e dal file `server.log` deve risultare che sono stati scartati 2 archi in quanto contengono identificatori di nodi non validi. (NOTA: il file `21archi.mtx` contiene lo stesso grafo di `9nodi.mtx` con l'aggiunta di due archi non validi che devono essere intercettati ed eliminati dal server; se `pagerank` riceve degli archi illegali termina con Exit code 1 durante la fase di lettura e non calcola il PageRank.)

Nel file README.md oltre alle informazioni su `pagerank` indicate prima deve essere discusso i quali modo sono gestiti i thread all'interno dei due programmi `graph_server.py` e `graph_client.py`.