Progettazione ed Analisi di Algoritmi
Raccolta di Prove pratiche

Armando Tacchella

26 settembre 2018

Indice

1 Simulazione di processi

Obiettivo

Realizzare una simulazione di esecuzione di un insieme di processi, dove ogni processo è caratterizzato da un *identificativo* univoco, una *priorità* e un *tempo* di esecuzione espresso in *quanti* di tempo macchina. In particolare, l'identificativo è un numero intero positivo diverso da 0, la priorità è un numero intero e il tempo di esecuzione è un numero intero positivo diverso da 0. Ricevuta in ingresso la descrizione dei processi, il simulatore deve organizzarli in una coda a priorità, ed "eseguirli" secondo la seguente politica:

- il processo a priorità maggiore viene eseguito per 10 quanti, oppure il tempo rimanente se questo è inferiore a 10 quanti; se vi sono più processi con la stessa priorità massima, viene privilegiato quello con identificativo minore;
- la priorità del processo è diminuita di una unità e il processo viene rimesso in coda.

Il simulatore, per ogni processo di cui è simulata l'esecuzione, deve stampare l'identificativo, la priorità iniziale, la priorità assegnata dopo l'esecuzione e il tempo residuo dopo l'esecuzione.

Formato di input

Il simulatore deve prendere un singolo file di input come argomento. Il contenuto del file è un certo numero di righe, non noto a priori, e una singola riga descrive un singolo processo. Il formato di ogni riga è il seguente:

identificativo *priorità* *tempo*

dove, per ogni campo, valgono le condizioni precisate sopra. I tre campi sono sempre separati da un singolo spazio e la riga termina con un ritorno a capo. Il simulatore non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il simulatore deve stampare a video la traccia di esecuzione dei vari processi. Ogni volta che un processo viene prelevato dalla coda ed "eseguito" il simulatore deve stampare una riga così composta:

identificativo *priorità iniziale* *priorità finale* *tempo residuo*

dove, per ogni campo, le informazioni sono quelle corrispondenti alla descrizione. Il simulatore non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

2 Identificazione di anagrammi

Obiettivo

Realizzare un programma che, dato un elenco di parole, identifica tutti gli anagrammi contenuti nell'elenco stesso. In particolare, partendo da un elenco di parole non ordinate lessicograficamente, si richiede di fornire lo stesso elenco ordinato e, a fianco di ogni parola, la lista di quelle che eventualmente risultano essere suoi anagrammi, anch'esse in ordine alfabetico. Ad esempio, se l'elenco iniziale fosse:

```
orca notte treno arco albero roca tonte albore
```

il programma dovrebbe fornire in uscita il seguente elenco:

```
albero albore
albore albero
arco   orca roca
notte  tonte
orca   arco roca
roca   arco orca
tonte  notte
treno
```

Formato di input

Il programma deve prendere un singolo file di input come argomento. Il contenuto del file è un certo numero di righe, non noto a priori, e **ogni singola riga** contiene **una sola parola**. Il simulatore non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il simulatore deve stampare a video il risultato finale su un numero di righe equivalente al file di ingresso, ed ogni riga risulta così composta:

parola anagramma1 anagramma2 ...

dove le parole risultano ordinate dalla prima all'ultima in senso **ascendente** e, per ogni parola, gli eventuali anagrammi sono ordinati anch'essi da sinitra a destra in senso **ascendente**. Il programma non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

3 Diradamento di punti

Obiettivo

Realizzare un programma che, dato un elenco di punti in termini di coordinate (x, y) con $x, y \in \mathbb{Z}^+$, effettua un "diradamento" degli stessi come descritto nel seguito. Dopo aver calcolato la distanza fra ciascuna coppia di punti, considera i due punti più vicini e toglie dall'elenco iniziale il punto con ordinata (valore y) maggiore. Se le ordinate hanno lo stesso valore, allora elimina il punto con ascissa (valore x) maggiore. Il procedimento viene ripetuto per un numero di punti noto a priori. Ad esempio, se l'elenco in ingresso fosse

```
30  20
50  80
10  60
40  60
10 100
```

e i punti da eliminare fossero 2, l'algoritmo dovrebbe eliminare le coppie (`40 60`) e (`50 80`).

Formato di input

Il programma deve prendere un singolo file di input come argomento. Il contenuto del file è il seguente:

- nella prima riga, il numero n di punti da eliminare;
- dalla seconda riga in poi, per ciascuna riga, una coppia di numeri interi non negativi separati da uno spazio che rappresentano le coordinate (x, y) dei punti.

Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video l'elenco delle coordinate dei punti da eliminare, restituendo una coppia di numeri per ogni riga separati da uno spazio. Le coordinate in output devono essere ordinate in modo crescente rispetto l'ascissa e, nel caso in cui avessero la stessa ascissa, rispetto all'ordinata. Il programma non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

4 Correttezza di un orario

Obiettivo

Realizzare un programma per la valutazione della correttezza di un orario scolastico come segue. Si supponga di avere un elenco di n docenti $D = \{d_1, \ldots, d_n\}$ e un elenco di m aule $A = \{a_1, \ldots, a_m\}$. L'orario è una lista O di quadruple (d, a, g, h) dove $d \in D$ è il docente, $a \in A$ è l'aula, $g \in \{1, 2, 3, 4, 5\}$ è il giorno della settimana, e $h \in \{8, 9, \ldots, 18\}$ è l'ora di inizio della lezione. Per semplicità, si assuma che ogni lezione duri esattamente un'ora. Il programma deve verificare che la lista rispetti i seguenti vincoli:

- Nessun docente si trova contemporaneamente in due aule diverse, ossia se risulta che $(d, a, g, h) \in O$ e $(d, a', g', h') \in O$ e $a \neq a'$, allora $g \neq g'$ e/o $h \neq h'$.
- Nessuna aula è allocata contemporaneamente a due docenti diversi, ossia non deve accadere che $(d, a, g, h) \in O$ e $(d', a, g, h) \in O$ con $d \neq d'$.
- Tutti i docenti devono aver svolto **almeno** k ore di lezione settimanali (k costante nota a priori)

Ad esempio, con $n = 3$, $m = 2$, se un frammento dell'elenco in ingresso fosse

```
...
1 2 1 9
2 2 1 9
...
```

si dovrebbe concludere che l'orario non è corretto in quanto alle ore 9 del Lunedì l'aula 2 è occupata contemporaneamente dal docente 1 e dal docente 2.

Formato di input

Il programma deve prendere un singolo file di input come argomento (utilizzare gli argomenti `argc` e `argv` della funzione `main`). Il contenuto del file è il seguente:

1. nella prima riga, il numero n di docenti, il numero m di aule e il numero k di ore minime da svolgere
2. dalla seconda riga in poi, per ciascuna riga, una quadrupla (d, a, g, h) di numeri interi separati da spazio e tali che $d \in \{1, \ldots n\}$, $a \in \{1, \ldots, m\}$, $g \in \{1, \ldots 5\}$, e $h \in \{8, 9, \ldots, 18\}$.

Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video le statistiche relative alla correttezza dell'orario come segue

```
Violazioni vincolo 1 : v1
Violazioni vincolo 2 : v2
Violazione vincolo 3 : v3
```

dove `v1`, `v2` e `v3` sono il numero di violazioni dei vincoli 1, 2 e 3, rispettivamente. Il programma non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

5 Differenza tra file di testo

Obiettivo

Realizzare un programma che, dati due file di testo contenenti parole separate da spazio, trovi e segnali le porzioni di testo diverse all'interno dei due file. Ad esempio, se i file in ingresso fossero:

```
L'orsa accudisce amorevolmente i suoi piccoli nel bosco e tollera male intrusioni
```

e

```
L'orsa accudisce i suoi piccoli amorevolmente nel bosco e non tollera intrusioni
```

il programma dovrebbe stampare:

```
1: amorevolmente i suoi piccoli
2: i suoi piccoli amorevolmente

1: tollera male
2: non tollera
```

Formato di input

Il programma deve prendere due file di input come argomento. Il contenuto dei file è una sequenza di parole separate da spazio. **Per semplicità** le parole comprendono gli eventuali segni di interpunzione: ad esempio, "`coda,`" è da considerarsi una parola diversa da "`coda`". Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video le porzioni di testo diverse con la sintassi:

```
1: <porzione file 1>
2: <porzione file 2>
```

Le porzioni discrepanti vanno separate da un ritorno a capo (come nell'esempio sopra). Il programma non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

6 Percorsi in un labirinto

Obiettivo

Realizzare una simulazione di ricerca di percorsi all'interno di un labirinto. Il programma riceve in ingresso la topologia del labirinto in un file organizzato in certo numero di righe, in cui ogni riga contiene solo caratteri

- "`*`", per denotare lo spazio occupato da un "muro";
- "`0`", per denotare la posizione di partenza (unica);
- "`1`", "`2`", ..., per denotare una posizioni di arrivo (una o più);
- uno spazio bianco, per denotare una parte libera da ostacoli.

Il numero di righe così come il numero di colonne non sono noti a priori, ma il numero di colonne è uguale su tutte le righe. Il simulatore, per ogni labirinto di cui è richiesta la simulazione, deve trovare se esiste un percorso tra la partenza e ciascuno dei possibili arrivi, stampando l'identificativo dell'arrivo e il numero di passi (spazi vuoti attraversati) necessari a raggiungerlo, oppure -1 nel caso tale arrivo non sia raggiungibile.

Formato di input

Il simulatore deve prendere un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è un certo numero di righe di medesima lunghezza, contenenti solo i caratteri elencati sopra e terminate con un ritorno a capo. Ad esempio:

```
****0***************
**** ********* *****
* ** ********* *****
* **       ***     *
* ** ***** *** *** 1
* ** ** **     *****
* ** ** ** *********
* ** ** ** *********
******* *************
*******   2*********
```

è la descrizione di un labirinto 10×20, in cui vi è una partenza e due arrivi. L'arrivo "1" è raggiungibile dalla partenza, mentre 2 non lo è. Il simulatore non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il simulatore deve stampare a video il risultato della ricerca dei percorsi nel labirinto come una serie di righe in cui ogni riga contiene due campi (separati da spazio)

<identificativo arrivo> <lunghezza percorso>

con il contenuto precisato sopra. Ad esempio, nel caso dell'input precedente il risultato sarebbe:

```
1  22
2 -1
```

Il simulatore non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

7 Percorsi in un labirinto (variante 1)

Obiettivo

Realizzare una simulazione di ricerca di percorsi all'interno di un labirinto. Il programma riceve in ingresso la topologia del labirinto come un grafo non diretto $G = (V, E)$ dove $V = \{x_0, \ldots, x_{N-1}\}$. Il grafo è rappresentato in un file, in cui la prima riga contiene il numero di vertici N, e le righe seguenti contengono gli archi $x_i x_j$ con $0 \leq i, j < N$. Il simulatore deve trovare se esiste un percorso tra la partenza, che è assunta essere sempre il nodo x_0, e l'arrivo, che è assunto essere sempre il nodo x_{N-1}. Il simulatore deve visualizzare la risposta (positiva o negativa) e la lunghezza del percorso **minimo**, se esiste un percorso, oppure -1 nel caso l'arrivo non sia raggiungibile dalla partenza.

Formato di input

Il simulatore deve prendere un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è il grafo specificato come sopra. Ad esempio:

```
4
0 1
0 2
1 3
2 3
```

è la descrizione di un grafo con $N = 4$ nodi. Il simulatore non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il simulatore deve stampare a video il risultato della ricerca del su una riga contenente due campi (separati da spazio)

<risposta> <lunghezza percorso>

dove *risposta* può essere solo `SI` o `NO`. Con l'esempio precedente il risultato sarebbe:

```
SI 2
```

Il simulatore non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

8 Percorsi in un labirinto (variante 2)

Obiettivo

Realizzare una simulazione di ricerca di percorso all'interno di un labirinto. Il programma riceve in ingresso la topologia del labirinto in un file organizzato in certo numero di righe, in cui ogni riga contiene solo "`*`", per denotare lo spazio occupato da un "muro"; "`0`", per denotare la posizione di partenza (unica); "`1`", per denotare la posizione di arrivo (unica); uno spazio bianco, per denotare una parte libera da ostacoli. Il numero di righe così come il numero di colonne non sono noti a priori, ma il numero di colonne è uguale su tutte le righe. Il simulatore deve trovare se esiste un percorso tra la partenza e l'arrivo stampando il numero di passi (spazi vuoti attraversati) necessari a raggiungerlo, oppure -1 nel caso tale arrivo non sia raggiungibile.

Formato di input

Il simulatore deve prendere un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è un certo numero di righe di medesima lunghezza, contenenti solo i caratteri elencati sopra e terminate con un ritorno a capo. Ad esempio:

```
****0***************
**** ********* *****
* ** ********* *****
* **       ***     *
* ** ***** *** *** 1
* ** ** **     *****
* ** ** ** *********
* ** ** ** *********
******* ************
*******    *********
```

è la descrizione di un labirinto 10×20, in cui l'arrivo "`1`" è raggiungibile dalla partenza. Il simulatore non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il simulatore deve stampare a video il risultato della ricerca del percorso nel labirinto come un singolo numero intero. Ad esempio, nel caso dell'input precedente il risultato sarebbe:

```
22
```

Il simulatore non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

9 Minimo albero ricoprente

Obiettivo

Una ditta di impianti telefonici ha necessità di collegare tra loro un certo insieme di siti minimizzando i costi di installazione. Nell'ipotesi che i collegamenti tra i siti siano bidirezionali e data una mappa delle distanze da ogni sito a tutti gli altri, si chiede di calcolare una possibile configurazione che soddisfa tale requisito.

Formato di input

Il programma deve prendere un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è il numero dei siti n nella prima riga, e poi una matrice $n \times n$ con le distanze tra i siti. Ad esempio:

```
4
 0 24 12 52
24  0 16 23
12 16  0 18
52 23 18  0
```

è la descrizione delle distanze tra $n = 4$ siti. Gli elementi della matrice sono separati da spazi e la matrice è ovviamente simmetrica, semidefinita positiva, con traccia nulla. Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video una serie di righe contenenti tre campi (separati da spazio)

<nodo sorgente> <nodo destinazione> <costo>

Ogni riga rappresenta una connessione tra due siti numerati da 1 a n. L'ultima riga del file deve contenere solo il costo totale. Ad esempio, nel caso sopra si avrebbe:

```
1 3 12
2 3 16
3 4 18
46
```

Il programma non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

10 Correttezza di un ordine totale rispetto ad un ordine parziale

Obiettivo

Dato un ordine parziale tra un insieme di attività $A = \{a_1, \ldots, a_n\}$ rappresentato come un grafo diretto aciclico $G = (A, E)$, e dato un ordine totale delle attività $\pi : A \to \{1, \ldots, n\}$, progettare e analizzare le prestazioni di un algoritmo che decide se l'ordine totale è compatibile con l'ordine parziale.

Formato di input

Il programma deve prendere un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è il numero di attività n nella prima riga, seguito dall'elenco π delle attività (su una singola riga) e poi dal grafo G che codifica le precedenze come $<$ *sorgente destinazione* $>$. Ad esempio:

```
4
1 3 2 4
1 2
1 3
3 4
2 4
```

è la descrizione relativa a 4 attività. Gli elementi dell'elenco e degli archi del grafo sono separati da spazi. Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video la risposta, che può essere la stringa `SI` nel caso in cui l'ordine π sia compatibile con G e la stringa `NO` in caso contrario. Ad esempio, nel caso sopra si avrebbe come risposta:

```
SI
```

Il programma non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

11 Test di connessione su grafo non orientato

Obiettivo

Realizzare un test di connessione su grafo non orientato. Il programma riceve in ingresso un grafo $G = (V, E)$ dove $V = \{x_0, \ldots, x_{N-1}\}$. Il grafo è rappresentato in un file, in cui la prima riga contiene il numero di vertici N, e le righe seguenti contengono gli archi (x_i, x_j) con $0 \leq i, j < N$. Il programma deve trovare se esiste un unico componente connesso, ossia se per tutte le coppie (x_i, x_j) con $0 \leq i, j < N$ esiste un percorso tra x_i e x_j. Il programma deve visualizzare la risposta (positiva o negativa).

Formato di input

Il programma deve prendere un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è il grafo specificato come sopra. Ad esempio:

```
4
0 1
0 2
1 2
```

è la descrizione di un grafo con $N = 4$ nodi e 3 archi. Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video il risultato del test su una riga contenente la risposta `SI` oppure `NO`. Con l'esempio precedente il risultato sarebbe:

```
NO
```

Il programma non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

12 Test di connessione su grafo non orientato (variante)

Obiettivo

Realizzare un test di connessione su grafo non orientato. Il programma riceve in ingresso un grafo $G = (V, E)$, dove $V = \{x_0, \ldots, x_{N-1}\}$, e un insieme di coppie di nodi Q. Per ogni coppia $(u, v) \in Q$ il programma deve trovare se esiste un percorso tra u e v in G, visualizzando la relativa risposta (positiva o negativa). Il grafo è rappresentato in un file, in cui:

- la prima riga contiene il numero di vertici N;
- la seconda riga contiene la lettera `G` seguita, a partire dalla terza riga, da archi (x_i, x_j) con $0 \leq i, j < N$ (una arco per riga);
- al termine dell'elenco degli archi, si trova una riga con la lettera `Q`, seguita, dalla riga successiva, da coppie (x_i, x_j) con $0 \leq i, j < N$ (una coppia per riga).

Formato di input

Il programma deve prendere un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è il grafo specificato come sopra. Ad esempio:

```
5
G
0 1
1 2
3 4
Q
0 3
0 2
```

è la descrizione di un grafo con $N = 5$ nodi, 3 archi e un insieme $Q = \{(0, 3), (0, 2)\}$. Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video il risultato del test su una riga contenente la risposta `SI` oppure `NO`. Con l'esempio precedente il risultato sarebbe:

```
NO
SI
```

Il programma non emette **nessun altro tipo di output** se non quello specificato e nelle modalità indicate.

13 Profilazione clienti

Obiettivo

Un sito di e-commerce dispone delle preferenze di un gruppo di m utenti relativamente a n prodotti, organizzata in una matrice A di dimensioni $m \times n$. Per ogni riga $A[i]$ si ha una classifica in cui ad ogni posizione è elencato il codice dell'articolo in ordine decrescente di preferenza. Ad esempio, se $n = 5$ e si ha $A[i] = \{3, 2, 4, 5, 1\}$, significa che l'utente i-esimo preferisce il prodotto 3, seguito dal prodotto 2, e così via. Realizzare un programma che, dato un generico utente i, costruisca una classifica dei restanti $m - 1$ utenti basata sulla *dissimilarità* delle loro preferenze rispetto a i. La dissimilarità $d(A[i], A[j])$ tra due utenti i e j è data dal numero di posizioni k per cui si ha che $A[i, k] \neq A[j, k]$. Ad esempio se $A[i] = \{3, 2, 4, 5, 1\}$ e $A[j] = \{3, 4, 1, 5, 2\}$ la dissimilarità è $d(A[i], A[j]) = 3$ perché vi sono tre posizioni per cui $A[i, k] \neq A[j, k]$.

Formato di input

Il programma prende un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è l'utente prescelto unitamente alla matrice specificata come sopra. Ad esempio:

```
2
3, 2, 4, 5, 1
3, 4, 1, 5, 2
4, 2, 3, 5, 1
1, 4, 3, 5, 2
```

richiede la classifica relativamente all'utente $i = 2$, unitamente alle preferenze di $m = 4$ utenti per $n = 5$ prodotti (i codici partono da 1). Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video le coppie

$$<\textit{id utente}> <\textit{dissimilarità}>$$

ordinate sulla base della dissimilarità (crescente). Con l'esempio precedente il risultato sarebbe:

```
4 2
1 3
3 4
```

Il programma non emette **nessun altro tipo di output** se non quello specificato.

14 Profilazione clienti (variante)

Obiettivo

Un grande magazzino dispone degli acquisti mensili di un gruppo di m utenti relativamente a n prodotti, organizzati in una matrice A di dimensioni $m \times n$. Per ogni cella $A[i, k]$ si ha la quantità dell'articolo k acquistato dal cliente i. Ad esempio, se $n = 5$ e si ha $A[i] = \{31, 12, 24, 57, 16\}$, significa che l'utente i-esimo ha acquistato 31 unità del prodotto con codice 1, 12 del prodotto con codice 2, e così via. Realizzare un programma che, dato un generico utente i, costruisca una classifica dei restanti $m - 1$ utenti basata sulla *dissimilarità* delle loro preferenze rispetto a i. La dissimilarità $d(A[i], A[j])$ tra due utenti i e j è data dalla somma

$$d(A[i], A[j]) = \sum_{k=1}^{n} \mid A[i, k] - A[j, k] \mid$$

Ad esempio se $A[i] = \{31, 12, 24, 57, 16\}$ e $A[j] = \{43, 14, 11, 15, 2\}$ la dissimilarità è $d(A[i], A[j]) = 83$.

Formato di input

Il programma prende un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è l'utente prescelto unitamente alla matrice specificata come sopra. Ad esempio:

```
3
6, 10, 13, 1, 0
43, 14, 11, 15, 2
31, 12, 24, 57, 16
7, 21, 13, 45, 21
```

richiede la classifica relativamente all'utente $i = 3$, unitamente alle preferenze di $m = 4$ utenti per $n = 5$ prodotti (i codici partono da 1). Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video le coppie

$$<id\ utente>\ <dissimilarità>$$

ordinate sulla base della dissimilarità (in senso crescente). Con l'esempio precedente il risultato sarebbe:

```
4 61
2 83
1 110
```

Il programma non emette **nessun altro tipo di output** se non quello specificato.

15 Componenti fortemente connesse

Obiettivo

L'ufficio viabilità di un comune richiede un algoritmo per la verifica della rete stradale cittadina relativamente alla proprietà che ogni luogo sia raggiungibile da ogni altro luogo, indipendentemente dalla presenza di sensi unici. Le ipotesi di lavoro sono le seguenti:

- la rete stradale è rappresentata da un grafo diretto in cui le strade sono archi e i nodi sono i punti di congiunzioni tra le strade (incroci, rotatorie, ecc.);
- la presenza di una strada a senso unico si traduce in un solo arco tra due nodi, mentre i doppi sensi sono rappresentati da due archi orientati in direzioni opposte;

L'algoritmo deve riportare in uscita i nodi raggruppati per porzioni tra di loro connesse correttamente. La presenza di più di una porzione indica un'anomalia.

Formato di input

Il programma prende un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è la rete stradale specificata come sopra. Ad esempio:

```
5
0 1
1 0
1 2
2 1
2 3
3 4
4 3
```

richiede la valutazione di una rete (V, E) con $|V| = 5$ nodi come indicato nella prima riga e le connessioni dirette $(u\ v) \in E$ specificate nelle righe seguenti. Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video i gruppi di nodi tra di loro connessi correttamente. Con l'esempio precedente il risultato sarebbe:

```
0 1 2
3 4
```

Il programma non emette **nessun altro tipo di output** se non quello specificato.

Obiettivo

Dato un grafo $G = (V, E)$, la *lunghezza media di percorso* $\overline{\pi}_G$ è una misura topologica definita come il numero medio di archi nel percorso tra una qualsiasi coppia di nodi $u, v \in V$ considerando un percorso minimo $\pi_{u,v}$ tra di essi. Progettare e realizzare un algoritmo che calcoli la lunghezza media di percorso dato in input un grafo G **non orientato** e **non pesato**. Supponendo inoltre che G sia **connesso**, l'algoritmo deve riportare in uscita $\overline{\pi}_G$, ossia

$$\overline{\pi}_G = \frac{\sum_{(u,v)\in(V\times V)} |\pi_{u,v}|}{|V|^2}$$

dove $|\pi_{u,v}|$ è la lunghezza di un percorso minimo tra $u, v \in V$.

Formato di input

Il programma prende un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è un grafo specificato, ad esempio, come segue:

```
5
0 1
0 2
1 3
2 3
1 4
3 4
```

Nell'esempio si richiede la valutazione di un grafo $G = (V, E)$ con $|V| = 5$ nodi come indicato nella prima riga e le connessioni **non orientate** $(u\ v) \in E$ specificate nelle righe seguenti. Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video la lunghezza media di percorso calcolata sulla base della definizione sopra riportata. Nell'esempio:

```
1.12
```

Il programma non emette **nessun altro tipo di output** se non quello specificato.

16 Lunghezza media di percorso

Obiettivo

Dato un grafo $G = (V, E)$, la *lunghezza media di percorso* $\overline{\pi}_G$ è una misura topologica definita come il numero medio di archi nel percorso tra una qualsiasi coppia di nodi $u, v \in V$ considerando un percorso minimo $\pi_{u,v}$ tra di essi. Progettare e realizzare un algoritmo che calcoli la lunghezza media di percorso dato in input un grafo G **non orientato** e **non pesato**. Supponendo inoltre che G sia **connesso**, l'algoritmo deve riportare in uscita $\overline{\pi}_G$, ossia

$$\overline{\pi}_G = \frac{\sum_{(u,v)\in(V\times V)} |\pi_{u,v}|}{|V|^2}$$

dove $|\pi_{u,v}|$ è la lunghezza di un percorso minimo tra $u, v \in V$.

Formato di input

Il programma prende un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è un grafo specificato, ad esempio, come segue:

```
5
0 1
0 2
1 3
2 3
1 4
3 4
```

Nell'esempio si richiede la valutazione di un grafo $G = (V, E)$ con $|V| = 5$ nodi come indicato nella prima riga e le connessioni **non orientate** $(u\ v) \in E$ specificate nelle righe seguenti. Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video la lunghezza media di percorso calcolata sulla base della definizione sopra riportata. Nell'esempio:

```
1.12
```

Il programma non emette **nessun altro tipo di output** se non quello specificato.

17 Interprete per un linguaggio macchina astratto

Obiettivo

Realizzare un interprete per un linguaggio di una macchina virtuale a variabili e stack, dotato delle seguenti operazioni (con relativo significato):

- Definizione di variabile: `DEF` <*nome_variabile*> <*valore*>, dove il nome della variabile è **una singola lettera alfabetica minuscola** e il valore è un **numero intero**; la definizione crea una nuova variabile e le assegna il corrispondente valore; se la variabile è già definita, il vecchio valore viene rimpiazzato.
- Operazione `PUSH` <*dato*>, dove il dato puó essere un **numero intero** oppure **il nome di una variabile**; l'operazione mette sulla cima dello stack il dato; se il dato è una variabile che non è stata definita, viene visualizzato un errore.
- Operazione `TOP`: il dato sulla cima dello stack viene visualizzato; l'operazione `POP` toglie anche il dato; se lo stack è vuoto, viene visualizzato un errore.
- Operazione `POP`: il dato sulla cima dello stack viene rimosso e visualizzato; se lo stack è vuoto, viene visualizzato un errore.
- Operazioni aritmetiche `ADD`, `SUB`, `MUL` e `DIV`, ossia addizione, sottrazione, moltiplicazione e divisione (tra interi); in questo caso i due dati sulla cima dello stack vengono tolti, usati come operandi e il risultato è immesso sulla cima; se vi sono meno di due dati nello stack, viene visualizzato un errore.

Per semplicità, supporre che nel programma in ingresso vi sia un'unica istruzione per riga e che le istruzioni siano sempre **sintatticamente** corrette. In caso di errori, si deve interrompere l'interprete immediatamente.

Formato di input

Il programma prende un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è un programma nel linguaggio specificato sopra. Ad esempio il programma:

```
DEF x 20
DEF y 16
PUSH 45
PUSH x
ADD
TOP
PUSH y
MUL
POP
```

realizza l'operazione $(x + 45) \cdot y$ con $x = 20$ e $y = 16$, rispettivamente. Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video eventuali errori oppure il risultato delle operazioni. Nell'esempio:

```
65
1040
```

Il programma non emette **nessun altro tipo di output** se non quello specificato.

18 Analisi di social network

Obiettivo

Una rete di relazioni sociali è rappresentata mediante un grafo **orientato** $G = (V, E)$ in cui un generico elemento $v \in V$ rappresenta una persona e, date due persone $v, u \in V$, un arco $(u, v) \in E$ significa che "u conosce v". **Nota**: il grafo è orientato, quindi la relazione non è necessariamente simmetrica, ossia il fatto che "u conosce v" non implica che "v conosce u". Data una persona $s \in V$, progettare e realizzare un programma per visualizzare la rete delle conoscenze di s con le relative "distanze". Ad esempio, se $V = \{v_1, v_2, v_3, v_4, v_5\}$ e $E = \{(v_1, v_2), (v_2, v_3), (v_4, v_3), (v_1, v_5)\}$ fissando $s = v_1$ si ha che:

- v_2 e v_5 sono conoscenze diretta di s (primo livello)
- v_3 è una conoscenza indiretta di s (secondo livello)
- v_4 non è una conoscenza di s

Formato di input

Il programma prende un singolo file di input come argomento (utilizzare gli argomenti da linea di comando). Il contenuto del file è una persona con la relativa rete sociale. Ogni persona è identificata con un intero e le relazioni sono coppie di interi. La persona di cui si vuole calcolare la rete sociale è indicata sulla prima riga e nelle righe seguenti vi sono le relazioni (una per riga). Ad esempio, nel caso dell'esempio sopra:

```
1
1 2
2 3
4 3
1 5
```

Il programma non prende **nessun altro tipo di input** se non quello specificato e nelle modalità indicate.

Formato di output

Il programma deve stampare a video il risultato dell'analisi della rete sociale. L'output consta di un certo numero di righe contenenti interi separati da spazi. La prima riga contiene le persone in relazione diretta con la persona indicata, la seconda riga quelle in relazione indiretta di secondo livello e così a seguire. L'ultima riga contiene le persone che **non sono** in relazione con la persona indicata. Nell'esempio:

```
2 5
3
4
```

Il programma non emette **nessun altro tipo di output** se non quello specificato.