Esercitazione di laboratorio n. 11

(Caricamento sul portale entro le 23.59 del 23/01/2017 dell'esercizio 1 e di almeno uno tra gli esercizi 2 e 3)

Esercizio n.1: Rete di elaboratori

Un grafo non orientato e pesato rappresenta una rete di elaboratori appartenenti ciascuno ad una sottorete. Il peso associato ad ogni arco rappresenta il flusso di dati tra due elaboratori della stessa sottorete o di sotto-reti diverse, come nell'esempio seguente (cfr. figura successiva).

Il grafo è contenuto in un file, il cui nome è passato come argomento sulla linea di comando. Il file ha il seguente formato:

- sulla prima riga un unico intero `N` rappresenta il numero di vertici del grafo
- seguono `N` righe ciascuna delle quali contiene una coppia di stringhe alfanumeriche, di al massimo 30 caratteri, `<id_elab> <id_rete>`
- sulle righe successive, in numero indefinito, appaiono terne nella forma `<id_elab1> <id_elab2> <flusso>`

Si facciano anche le seguenti assunzioni:

- i nomi dei singoli nodi sono univoci all'interno del grafo
- non sono ammessi cappi
- tra due nodi c'è al massimo un arco (non è un multigrafo).
- le sotto-reti sono sotto-grafi non necessariamente connessi

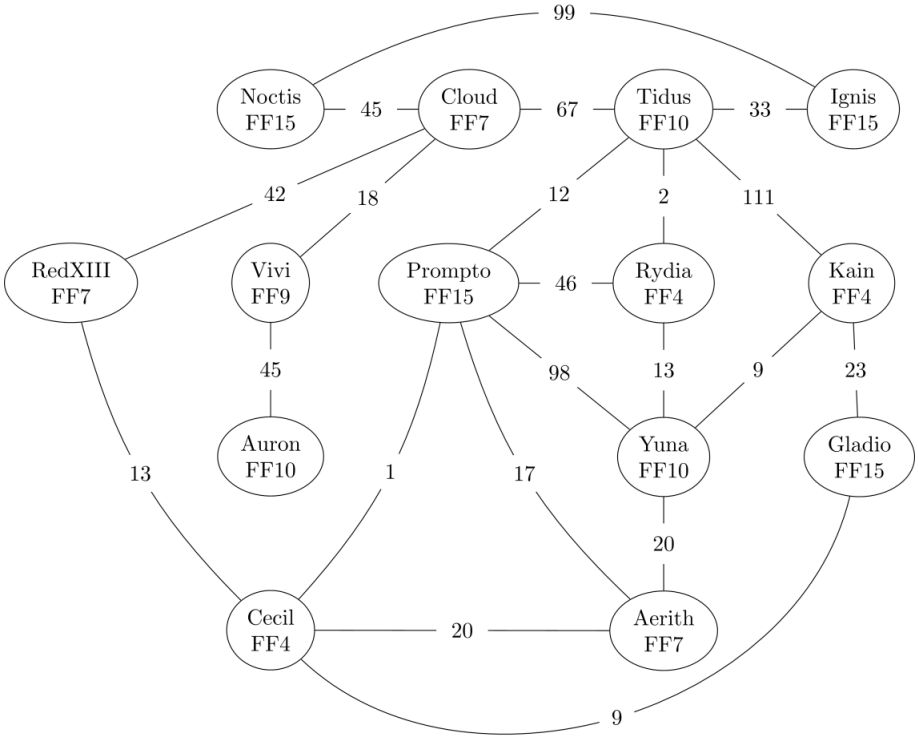
Si scriva un programma in C in grado di caricare in memoria il grafo, leggendone i contenuti da file e di potervi effettuare alcune semplici operazioni.
La rappresentazione della struttura dati in memoria deve essere fatta tenendo conto dei seguenti vincoli:

- il grafo sia implementato come ADT di I classe, predisposto in modo tale da poter contenere sia la matrice sia le liste di adiacenza. Nella fase di caricamento dei dati da file si generino solamente le liste di adiacenza, su comando esplicito va generata anche la matrice di adiacenza
- il grafo sia realizzato identificando ogni nodo con un indice intero. Le informazioni relative agli elaboratori e alle sotto-reti siano memorizzate in un vettore di struct, tale da attribuire ad ogni nodo un indice intero (corrispondenza indice-dato) tra 0 e |V|-1.
- Gli indici siano generati automaticamente man mano che si acquisiscono i dati. A tale scopo si utilizzi una tabella si simboli (ADT di I classe) tale da fornire corrispondenze "*da nome a indice*". Internamente all'ADT le conversioni nome-indice siano gestite mediante una tabella di hash.

Sul grafo, una volta acquisito da file, sia possibile:

- stampare il numero totale di vertici elencandoli esplicitamente per nome
- stampare il numero di archi incidenti su un nodo e l'elenco di vertici ad esso connessi
- generare la matrice di adiacenza, ***SENZA*** leggere nuovamente il file, a partire dalle liste di adiacenza

In allegato al testo è presente il grafo d'esempio (nel file `grafo.txt`) rappresentato a seguire:



Esercizio n.2: PQ con heap
Si realizzi un programma C che permetta di gestire una struttura dati di tipo coda a priorità (PQ), realizzata mediante *heap*. Il tipo Item sia caratterizzato da una priorità (intero) e un codice alfanumerico (stringa di al massimo 10 caratteri).
Ai fini della risoluzione dell'esercizio si chiede che venga svolto partendo dai file messi a disposizione sul portale (`l11e2_main.c`, `pq.h` e `item.h`).

Esercizio n.3: Stringhe e ricorsione
Sia data una stringa `str` di al massimo 30 caratteri e un vettore `lunghezze` di `N` interi distinti che rappresentano la lunghezza delle sottostringhe in cui si vuole decomporre la stringa originale.
Si scriva una funzione ricorsiva in C

```
void decomponi(char *str, int num, int *lungh);
```

che visualizzi tutte le possibili decomposizioni di `str` usando sottostringhe di lunghezza specificata nel vettore `lunghezze`.
Esempi:

- se `str = "tentativo"`, `N = 3`, `lunghezze` contiene 2, 5, 7, una delle possibili decomposizioni è `"te" "nt" "ativo"`. Una seconda decomposizione possibile sarebbe `"te" "ntativo"`. Una terza `"tenta" "ti" "vo"` e così via.
- se `str = "tentativo"`, `N = 2`, `lunghezze` contiene 2, 4 non vi sono decomposizioni.