Commento al Laboratorio n. 12

Esercizio n.1: Grafi e DAG
Si ricordino le seguenti definizioni:

- un grafo **non orientato** si dice **connesso** se e solo se ogni coppia (v_i, v_j) di vertici è connessa da un cammino o equivalentemente se e solo se esiste una sola **componente connessa**
- un grafo **orientato** si dice **fortemente connesso** se e solo se ogni coppia (v_i, v_j) di vertici è **mutuamente** connessa da un cammino o equivalentemente se e solo se esiste una sola **componente fortemente connessa**
- un grafo **orientato** si dice **debolmente connesso** o semplicemente **connesso** se e solo se il corrispondente grafo non orientato è connesso.

Il grafo non orientato corrispondente a un grafo orientato si ottiene da quest'ultimo rimuovendo la direzione degli archi, quindi dato l'arco (v_i, v_j) introducendo l'arco (v_j, v_i) nel grafo.

Come scelta progettuale si opta per non gestire nel `main` le informazioni relative ai nomi dei vertici, tutte trattate internamente al grafo. La tabella di simboli `st` è interna al grafo ed è realizzata come tabella di hash. Per recuperare il nome del vertice dato l'indice si introduce un vettore `vett` di vertici, anch'esso interno al grafo.

Strutture dati:

- **Quasi ADT** `Edge`: tipologia 1 con 3 campi interi per gli indici dei 2 vertici su cui insiste l'arco e un intero per il peso. Vista la semplicità, esso è definito in `Graph.h`. Le funzioni disponibili sono di creazione (`EDGEcreate`) e visualizzazione (`EDGEstore`) e sono realizzate in `Graph.c`
- **ADT di I classe** `ST`: tabella di simboli interna al grafo implementata come tabella di hash con open addressing e linear probing (cfr. es. 4.3.3 in *Algoritmi e programmazione in pratica*). Le funzioni disponibili sono di inizializzazione (`STinit`), liberazione (`STfree`), inserzione di un vertice (`STinsert`) e ricerca (`STsearch`)
- **Quasi ADT** `Vertex`: tipologia 3 con 1 campo stringa `name` (vettore di caratteri sovradimensionato) con funzione di estrazione del nome dato l'indice (`GRAPHgetName`). Vista la semplicità, esso è definito e implementato in `Graph.c`

```
void  DAGts(Graph g, int *ts);
void  DAGmaxPath(Graph d, int *ts);
```

- **ADT di I classe** `Graph`**:** è un grafo orientato e pesato, rappresentato come matrice delle adiacenze con funzioni standard di inizializzazione (`GRAPHinit`), liberazione (`GRAPHfree`), acquisizione (`GRAPHload`), memorizzazione (`GRAPHstore`), inserzione/rimozione di un arco (`GRAPHinsertE`/`GRAPHremovetE`), recupero del numero di vertici/archi (`GRAPHgetNumV`/`GRAPHgetNumE`), elencazione degli archi (`GRAPHedges`), calcolo del peso complessivo di un insieme di archi (`GRAPHedgesWt`). La funzione di visita in profondità `GRAPHdfs` è una versione semplificata di quella standard che si limita a verificare se il grafo è ciclico o aciclico controllando se vi è almeno un arco etichettato Back. La funzione `DAGts` calcola e visualizza un ordinamento topologico del DAG. La scelta di rappresentare il grafo come matrice delle adiacenze rende molto semplice l'algoritmo di ordinamento topologico, in quanto è immediato identificare gli archi incidenti mediante semplice inversione degli indici

dei vertici. Si ricordi che l'ordinamento topologico non è sempre unico. La funzione `DAGmaxPath` calcola e visualizza i cammini massimi da ogni nodo verso ogni altro nodo. Nello specifico, si analizzano i nodi secondo l'ordine topologico calcolato precedentemente e si calcola la distanza da ogni nodo interpretato come "sorgente" verso tutti quelli che lo seguono nell'ordinamento stesso. Poiché gli archi non ammettono pesi negativi, si utilizza il valore -1 per rappresentare l'assenza dell'arco. La funzione applica la relaxation inversa degli archi iterando sui vertici in base all'ordine topologico.

Algoritmo: il `main` inizializza le strutture dati (grafo e tabella di simboli), legge il grafo (funzione `GRAPHload`), verifica se è ciclico (funzione `GRAPHdfs`), genera il set di archi a cardinalità minima e peso massimo che lo trasforma in un DAG (funzione `generaSetE`), ordina topologicamente il DAG (funzione `DAGts`) e calcola su questo DAG i cammini massimi da tutte le sorgenti (funzione `DAGmaxPath`).

Trasformazione in DAG: in Teoria del Grafi, l'insieme di archi la cui rimozione rende un grafo aciclico è noto con il nome di "*feedback arc set*" (FAS). Tra tutti i FAS esistenti in un grafo, lo scopo del problema in esame è l'individuazione del "*minimum feedback arc set*" (MFAS). Il problema MFAS appartiene alla classe di complessità NP-hard. Senza scendere troppo nei dettagli, si può dimostrare che tale problema è tanto difficile quanto l'enumerazione di tutti i cicli semplici all'interno di un grafo. Per un grafo generico il numero di cicli semplici è $\Omega(|V|^2)$. Si ricordi che l'identificazione di archi Back permette di affermare che esiste un ciclo, ma non lo individua. Non è quindi sufficiente la rimozione degli archi Back per risolvere il problema MFAS. Gli approcci con complessità polinomiale sono quantomeno limitati nella loro applicabilità a risolvere il problema nel caso generale. Sebbene esistano casi particolari in cui soluzioni approssimate possano coincidere con l'ottimo effettivo, questa è una casistica piuttosto limitata e fortemente dipendente dalla topologia del grafo in esame. Per questa ragione la soluzione proposta si basa su una esplorazione esaustiva di tutto lo spazio delle soluzioni con i modelli del Calcolo Combinatorio.

Generazione del set di archi da rimuovere: il modello è quello delle combinazioni semplici di dimensioni crescenti da 1 a un massimo. Il set vuoto è escluso in quanto, a valle della funzione `GRAPHdfs`, è già possibile sapere se il grafo è in origina un DAG o meno. Il massimo numero di archi rimuovibili ha un limite superiore, in quanto il minimo numero di archi di un grafo connesso è quello di un albero, quindi $|V|-1$. Si possono quindi al massimo rimuovere $|E| - (|V|-1)$ archi. La rimozione degli archi è fatta mediante applicazione della funzione `GRAPHremoveE`, il ripristino mediante la `GRAPHinsertE`. Una volta trovato il set di archi ottimo (cardinalità minima e peso massimo) questo viene rimosso definitivamente dal grafo originale, generando il DAG desiderato.