Mattia Molinari - 296544

Laboratorio 9 esercizio 1: Grafi e DAG

La soluzione proposta è composta da cinque file:

- **st.h**: definizione della tabella di simboli come ADT di I classe (tipo st_t) e prototipi delle funzioni ad essa associate (vedere commenti per dettagli);
- **st.c**: implementazione della tabella di simboli secondo il modello del vettore non ordinato di stringhe e implementazione delle funzioni relative;
- **graph.h**: definizione del grafo come ADT di I classe (tipo graph_t) e prototipi delle funzioni ad esso associate (vedere commenti per dettagli);
- **graph.c**: implementazione del grafo orientato come **struct** con i seguenti campi:
  - st_t st: tabella di simboli, possibile grazie a #include "st.h"
  - edge_t *edges: elenco degli archi, il tipo edge_t viene dichiarato nel medesimo file come **struct** che ha come campi l'indice del vertice di partenza, quello del vertice di arrivo e il peso dell'arco (tutti interi)
  - int E, V: numero di archi e vertici
  - list_t *adj: vettore di liste per le adiacenze, il tipo list_t viene dichiarato nel medesimo file come **struct** con puntatore a testa della lista

  Risulta particolarmente rilevante l'implementazione delle seguenti funzioni:
  - int GRAPHcheckDAG(graph_t G, int *mark, int *pre, int *post): funzione che viene opportunamente chiamata da main.c per controllare se la rimozione degli archi segnati in mark sia un DAG o meno; per far ciò si utilizzano le funzioni GRAPHremoveEdges e GRAPHaddEdges rispettivamente per rimuovere e poi aggiungere nuovamente gi archi marcati alla lista delle adiacenze con un controllo intermedio di dfsR: quest'ultima è stata opportunamente modificata affinché restituisca un valore se si trovano cicli [0] o meno [1] (vedere commenti per dettagli)
  - void GRAPHmaximumPath(graph_t G, FILE *fout): funzione che utilizza TSdfsR tratta dai lucidi per trovare l'ordine topologico dei vertici del DAG e calcolare le distanze massime di ognuno di essi da ogni sorgente
- **main.c**: file principale.

La soluzione sfrutta il modello del calcolo combinatorio dell'**insieme delle parti** per generare insiemi di archi: in particolare si utilizza il modello che richiama le combinazioni semplici in quanto permette la generazione di insiemi di cardinalità crescente e dunque di trovare prima quelli di cardinalità minima. Più precisamente i passaggi sono i seguenti:

1. inizializzazione variabili utili e grafo G caricando le informazioni da file;
2. allocazione variabili: da notare il vettore mark, la cui posizione i-esima marca l'arco di indice i nel vettore edges (visibile solo in graph.c) come appartenente [1] o meno [0] all'insieme;
3. si controlla che il grafo non sia già un DAG (se lo è si stampa un messaggio a terminale e si chiede se si vuole proseguire comunque);

4. si utilizza la funzione powerset in un opportuno ciclo per generare gli insiemi partendo da quelli a cardinalità minore;

5. all'interno della funzione powerset si controlla per ogni insieme generato se la rimozione degli archi marcati genererebbe un DAG: in caso di risposta affermativa, si aggiorna il flag ad 1 e si stampa la cardinalità k raggiunta, per poi stampare l'insieme corrente; la funzione prosegue generando solo più gli insiemi con cardinalità k, stampando e confrontando il peso totale con *maxWt solo di quelli che se rimossi generano un DAG (si aggiorna eventualmente la soluzione ottima sol);

6. terminata la ricorsione e il ciclo **for**, si stampa la soluzione finale con relativo peso;

7. si rimuovono definitivamente gli archi marcati e si stampano le distanze massime da ogni nodo sorgente verso ogni nodo del DAG;

8. liberazione spazio allocato.