Relazione di laboratorio

Laboratorio 9

Andrea Grillo, S282802

Nella presente relazione verranno discussi gli approcci e le strategie adottate per la risoluzione dell'esercizio di programmazione del Laboratorio 9, con particolare riguardo alle strutture dati utilizzate e agli algoritmi scelti che hanno portato alla risoluzione.

Strutture dati

Il testo richiede di lavorare su un grafo. A tale scopo è stato definito un ADT di prima classe GRAPH nel file header *graph.h*, con implementazione delle funzioni ad esso associate nel file *graph.c*.
Il grafo, per la memorizzazione degli identificatori dei vertici, contiene al suo interno un'istanza di una tabella di simboli, anch'essa a sua volta definita come ADT di prima classe nei file *st.h* e *st.c*, rispettivamente header e implementazione.

Tabella di simboli

Data l'assenza di utilizzo estensivo della tabella di simboli, che viene utilizzata solamente in fase di caricamento in memoria dei dati e nella fase di stampa, si è scelto di mantenere l'implementazione della suddetta il più semplice possibile.
La scelta è ricaduta sull'utilizzo di un vettore non ordinato. Un fattore che ha contribuito a tale decisione è la disponibilità iniziale del numero di nodi presenti nel grafo, che ha reso possibile una singola allocazione dinamica iniziale.
Ciò ha condotto all'implementazione delle seguenti operazioni con la rispettiva complessità:

1. Inserimento in coda al vettore O(1)
2. Ricerca dell'identificatore di un vertice a partire dall'indice O(1)
3. Ricerca dell'indice di un vertice a partire dall'identificatore O(|V|)

Grafo

Per quanto riguarda la rappresentazione in memoria del grafo, oltre all'istanza della tabella di simboli utilizzata per la memorizzazione dei vertici, è stato ritenuto opportuno utilizzare una doppia memorizzazione degli archi. Essi vengono inizialmente memorizzati in un array di strutture di tipo *edge_t*, per poi essere in seguito utilizzati per popolare l'array di liste di adiacenza. La memorizzazione in liste di adiacenza rende più efficiente l'utilizzo degli

algoritmi di visita, mentre la memorizzazione in vettore di archi rende particolarmente semplice la manipolazione degli archi in un algoritmo ricorsivo di ricerca e ottimizzazione.

Scelte algoritmiche

La prima e la seconda richiesta dell'esercizio consistono in un problema di ricerca e ottimizzazione. Tale problema è stato risolto tramite l'applicazione del modello ricorsivo delle combinazioni semplici. Il modello genera tutte le possibili combinazioni di cardinalità via via crescente di archi e ha, nel caso terminale, la chiamata ad una funzione *GRAPHcheckDAG*, che eseguendo una visita in profondità nel grafo ricerca l'esistenza di archi di tipo *back* che violano la condizione dell'aciclicità.
L'algoritmo implementato fa utilizzo della memorizzazione in array degli archi del grafo.
Tale scelta è giustificata dal fatto che è così necessario generare unicamente un array *mark*, che indica se con quella combinazione è necessario eliminare l'arco in quella posizione o no.
Nella *GRAPHcheckDAG* vengono inizialmente eliminati dalle liste di adiacenza i vertici indicati nel vettore *mark* con l'utilizzo della funzione *GRAPHremoveEdges*, in seguito viene eseguita la visita in profondità e infine le liste di adiacenza vengono ripristinate con la funzione *GRAPHaddEdges*.
La *GRAPHremoveEdges*, effettuando una rimozione in lista non ordinata, ha complessità O(|V|OD), dove |V| è la cardinalità dell'insieme di vertici e OD è l'*out-degree* massimo dei vertici, che rappresenta il massimo numero di nodi di ogni lista.
La *GRAPHaddEdges* ha invece complessità O(|V|), in quanto gli archi vengono aggiunti sempre in testa poiché non ha importanza sostanziale l'ordine in lista.
Se la soluzione generata rappresenta un DAG, essa viene stampata a video. Viene inoltre calcolata la somma dei pesi degli archi utilizzando la funzione *GRAPHsumWeight*, che ritorna la somma a partire dal vettore mark già generato.
Una volta individuato l'insieme di archi da rimuovere dal grafo a somma massima dei pesi, esso viene stampato a video e definitivamente rimosso dalle liste di adiacenza.
A questo punto si passa alla risoluzione dell'ultima richiesta dell'esercizio attraverso la chiamata alla funzione *GRAPHdagMaximumPath*.
La suddetta chiamata esegue in primo luogo una variante della visita in profondità, utilizzata per determinare l'ordine topologico del DAG. Poi, seguendo detto ordine, vengono eseguite in sequenza le relaxation inverse degli archi, che vanno via via a determinare i cammini massimi per raggiungere ogni nodo del DAG. Infine viene stampato a video l'ordine topologico del grafo e i cammini massimi relativi ad ogni vertice.