Politecnico di Torino

Algoritmi e Programmazione
Relazione laboratorio 9

Andrea Angelo Raineri - s280848

20/1/22

1 Strutture dati utilizzate

1.1 ADT Grafo (GRAPH.c, GRAPH.h)

ADT (tipo di dato astratto) necessario a memorizzare il grafo letto da file e il grafo aciclico (DAG) successivamente ricavato. Rispettando i requisiti di ADT l'implementazione interna del tipo grafo è nascosta al client. L'implementazione interna è stata effettuata tramite lista delle adiacenze e le corrispondenze indice-vertice sono contenute in tabella di simboli (tipo ST). Questa implementazione prevede l'utilizzo di grafi orientati pesati ma nessuna conoscenza particolare riguardo la ciclicità del grafo, lasciata in responsabilità al client. Non si è quindi scelto di implementare una ADT adhoc per il DAG.

Alle funzioni di libreria di base sono state aggiunte le seguenti funzioni:

- **void ECdfsR()**

  Tramite una visità in profondità classifica gli archi del grafo

- **void GRAPHfindESubsetToDAG()**

  Dato un grafo orientato pesato trova tutti i subset di archi di cardinalità minima che se rimossi rendono il grafo un DAG, in particolare trova il subset a peso maggiore

- **Graph GRAPHcreateFromGraphEdgeSubtraction()**

  Dato un grafo di partenza e un vettore di archi, crea un nuovo grafo sottraendo gli archi del vettore al grafo di input

- **int GRAPHcheckSource()**

  Verifica in un grafo orientato se un vertice è una sorgente (non ha archi entranti)

- **void GRAPHfindMaxPathsFromSources()**

  Dato un DAG trova ed elenca i cammini massimi verso ogni vertice da ogni sorgente

1.2 ADT Tabella di simboli (ST.c, ST.h)

ADT utilizzato per contenere le corrispondenze indice-nome dei vertici del grafo. L'implementazione interna non sfrutta tabelle di simboli efficienti data la relativa piccola dimensione del problema, si è quindi optato per un vettore di Record, ognuno dei quali contiene la corrispondenza indice-label.

2 Strategie algoritmiche

2.1 Ricerca subset

Dato un grafo orientato, era richiesto identificare i sottoinsiemi di archi a cardinalità minima che rendessero il grafo un DAG se rimossi. Un approccio base al calcolo della soluzione sarebbe l'applicazione dei modelli di calcolo combinatorio per effettuare una esplorazione esaustiva dello spazio delle soluzioni con conseguente verifica di aciclicità e minima cardinalità delle soluzioni trovate. Una soluzione di questo tipo, per quanto efficace, è molto costosa da un punto di vista computazionale. Si è quindi cercata una soluzione alternativa. L'algoritmo implementato si basa sull'idea che dato un grafo orientato, a seguito di una classificazione degli archi del grafo, la rimozione di tutti gli archi di tipo backward sia necessaria e sufficiente per soddisfare la richiesta di aciclicità. Il problema quindi si riduce alla ricerca dell'insieme di archi backward a minima cardinalità, effettuata tramite ripetute visite in profondità, una a partire da ogni vertice del grafo. Identificati quindi gli insiemi di archi backward a minima cardinalità, si procede alla

ricerca dell'insieme a massimo peso e alla creazione di un nuovo grafo privato di questo insieme di archi.

La stima di complessità dell'algoritmo implementato è $O(|V||E| + |V| + |E|) = O(|V||E|)$

2.2 Cammini massimi su DAG

Dato il nuovo DAG ottenuto al passo precedente, essendo il grafo orientato e privo di cicli per definizione il problema dei cammini massimi non è NP-completo. L'algoritmo implementato per risolvere il problema si ispira all'algoritmo di Bellman-Ford per la ricerca dei cammini minimi su grafo a partire da una sorgente, con due variazioni che lo adattano al nostro caso:

1) il passo di "relaxation" applicato sugli archi cerca di massimizzare il peso del cammino

2) Prevedendo l'algoritmo di Bellman-Ford la ricerca dei cammini a partire da una sola sorgente, si effettua un ciclo sui vertici del grafo e si effettua la ricerca dei cammini se questi sono nodi sorgente

La stime di complessità dell'algoritmo così implementato è dunque $O(|V|^2|E|)$