Corso di Algoritmi e Strutture Dati
Prova di laboratorio

durata della prova: 3 ore

Testo della prova

Con *logistica intermodale* si intende il processo di pianificazione del trasporto di merce utilizzando più tipologie di trasporto (nave, aereo, gomma, rotaia) anziché una soltanto.

Supponiamo che un operatore voglia entrare nel mercato fornendo servizi efficienti di logistica intermodale in Liguria. La prima azione che dovrà effettuare è l'integrazione delle informazioni sulle reti dedicate al trasporto merci. L'operatore, infatti, potrebbe aver bisogno di una visione integrata dell'intera rete di sistemi di trasporto, comprendente sia la rete ferroviaria ligure che quella stradale (Figura 1), ed eventualmente anche quella marittima.

Nell'ipotesi estremamente semplificatrice che queste reti siano rappresentate come grafi non orientati, il problema consiste nell'unire le informazioni, in parte sovrapposte e in parte complementari, presenti in due o più grafi distinti. Molti vertici compaiono in entrambe le mappe (es. Genova, Savona, Ronco Scrivia, Rapallo) ma molti altri sono caratteristici solo di una delle due (es. Borzoli, Savona P. Doria nella mappa ferroviaria; Ceranesi, Savignone, Favale nella mappa stradale). Analogamente, molti archi potrebbero comparire in entrambe le mappe ma molti altri potrebbero comparire solo in una di esse.

Nello scenario semplificato che prendiamo in considerazione, serve dunque una operazione di **immersione** (*merge*) di una mappa entro un'altra. Tale operazione deve restituire una nuova mappa in cui compaiano tutti i vertici e tutti gli archi che si trovano nelle due mappe di partenza, con la sola cautela di evitare i doppioni che ne potrebbero nascere. Risulta inoltre utile, nelle mappe, distinguere fra tratte di tipo esclusivamente ferroviario, oppure esclusivamente stradale, oppure di entrambi i generi. La lunghezza delle tratte viene invece ignorata per semplicità.

Figure 1: Reti di trasporto in Liguria: ferroviaria (sinistra) e stradale (destra).

Requisiti di realizzazione e punteggi

A seguito dei requisiti narrati nella sezione precedente, in concreto si richiede di implementare il tipo di dato **grafo non orientato con vertici etichettati e archi etichettati**. Le etichette dei vertici rappresentano i nomi delle località collegate, dunque si tratta di semplici stringhe di caratteri. Le etichette degli archi invece rappresentano il tipo di tratta disponibile: "f" se disponibile solo su ferrovia, "s" se disponibile solo su strada, "e" se disponibile su entrambe le modalità.

L'implementazione deve sfruttare l'approccio a **matrice di adiacenza. La eventuale realizzazione con liste di adiacenza non consente di raggiungere la sufficienza**.

Le operazioni da realizzare, e da racchiudere in apposito namespace, sono almeno le seguenti:

- `Graph createEmpty()`: creazione di un grafo vuoto con un numero massimo prefissato di possibili vertici, indicato da una apposita costante nel programma, e realizzato con una tabella per le etichette dei vertici (allocata alla dimensione massima) e una matrice di adiacenza (allocata alla dimensione massima).
  punti 2

- `bool addVertex(Graph &, Vlabel)`: inserimento di un vertice.
  Il parametro di tipo `Vlabel` (in questa prova definito come `string`) rappresenta l'etichetta da attribuire al nuovo vertice (si tratta del nome di una località). La funzione restituisce `false` se l'etichetta è già presente nel grafo, e in tal caso nessun nuovo vertice deve essere inserito. Altrimenti restituisce `true` ad inserimento avvenuto.
  punti 2

- `bool addEdge(Graph &, Vlabel, Vlabel, Elabel)`: inserimento di un arco **non orientato** tra due vertici esistenti e distinti.
  I due vertici sono identificati dalle loro etichette, indicate dai due parametri di tipo `Vlabel`. Invece il parametro di tipo `Elabel` (in questa prova definito come `char`) rappresenta il tipo da attribuire al nuovo arco: "f" se solo ferroviario, "s" se solo stradale, "e" se disponibile in entrambe le modalità. La funzione restituisce `false` se i due vertici sono già collegati da un arco avente la stessa etichetta, o se uno dei due vertici non è presente nel grafo, oppure se i due vertici coincidono; in tali casi nessun nuovo arco deve essere inserito. Altrimenti restituisce `true` ad inserimento avvenuto; se tra i due vertici esisteva già un arco di tipo diverso da quello specificato nel parametro, allora l'etichetta preesistente dell'arco viene sovrascritta con "e", altrimenti viene impostata al valore indicato dal parametro.
  punti 2

- `void print(const Graph &)`: visualizzazione del grafo, elencando, in corrispondenza della etichetta di ciascun vertice, l'elenco delle etichette dei vertici ad esso adiacenti, ciascuno contraddistinto dal tipo di tratta di collegamento ("f" oppure "s" oppure "e").
  punti 2

- `bool connected(const Graph &, Vlabel, Vlabel)`: restituisce `true` se esiste un cammino qualunque tra i vertici indicati dai due parametri, `false` altrimenti. Si suggerisce un approccio ricorsivo e l'impiego di opportuni accorgimenti per evitare che il cammino passi più volte attraverso uno stesso vertice (nel grafo possono esistere cicli). La visualizzazione del cammino trovato non è richiesta (ma può essere utile in fase di debugging).
  punti 2

- `bool merge(const Graph &, const Graph &, const Graph &)`: immersione (*merge*) di un grafo entro un'altro.
  I due grafi da immergere sono denotati dai primi due parametri, mentre il terzo parametro indica il grafo di destinazione che dovrà contenere il risultato dell'immersione. Quest'ultimo non viene creato dalla funzione: esso viene invece creato dal chiamante, che lo passa alla

funzione, la quale lo svuota se non risulta vuoto e poi lo riempie con il *merge* dei primi due grafi. L'operazione deve evitare la creazione di doppioni tra i vertici (ossia vertici distinti aventi stessa etichetta) e di doppioni tra gli archi (più archi che collegano la stessa coppia di vertici). La funzione restituisce `false` se la capienza del grafo di destinazione risulta insufficiente a contenere l'intera immersione; altrimenti restituisce `true` a immersione effettuata.
punti 4

È consentito utilizzare la libreria standard del C++, ad eccezione della STL (Standard Template Library) e dunque l'uso dei vector non è consentito. Inoltre non è consentito ricorrere ad altre librerie.

La sufficienza è ottenuta totalizzando almeno 8 punti. Su ciascuna funzione si raggiunge il punteggio indicato se essa: 1) è corretta, 2) opera con una ragionevole complessità computazionale, 3) evita passaggi inutili, 4) non presenta errori di stile. Con "errori di stile" si intendono ad esempio l'accesso alle strutture dati che non avviene attraverso le funzioni del TDD ma in modo diretto, la mancata fattorizzazione (frammenti di codice ripetuti svariate volte invece che racchiusi in una funzione), codice poco commentato, difficile da seguire, inutilmente complesso.

Il file di intestazione (*graph.h*) contenente i prototipi delle funzioni da realizzare viene fornito già implementato e non deve essere modificato.

Il `main()` del programma viene fornito già implementato. Non è consentito modificarlo. Esso inizialmente crea tre grafi vuoti, e successivamente consente di inizializzare i primi due con dati esterni, acquisendone il contenuto da standard input oppure da file nel formato seguente:

(esempio per rete puramente ferroviaria)

```
ArquataScrivia RoncoScrivia f
RoncoScrivia Bolzaneto f
Bolzaneto Rivarolo f
....
....
VadoLigure FinaleLigure f
0
```

(esempio per rete puramente stradale)

```
SestriLevante CaviDiLavagna s
CaviDiLavagna Lavagna s
Lavagna Chiavari s
....
....
Imperia PieveDiTeco s
0
```

In questo modo si inseriscono simultaneamente sia le etichette dei vertici (località collegate) che gli archi (relazioni di collegamento) e non risulta possibile inserire vertici isolati (che in una rete di trasporti non avrebbero senso).

Successivamente il `main()` mostra un menu che permette la scelta ripetuta tra le seguenti azioni:

1. Visualizzazione di uno dei tre grafi, a scelta.
2. Controllo di esistenza di un cammino tra due vertici a scelta di uno dei tre grafi disponibili.
3. Immersione tra due dei tre grafi disponibili, con risultato salvato nel terzo di essi.
4. Uscita dal programma.