Studente: Sattolo Francesco	Matricola: S235280
Appello del 28/01/2020	Prova di programmazione (18 punti)

Relazione

Come scelta progettuale si decide di non gestire nel `main` le informazioni relative ai vertici, tutte trattate internamente al grafo. Per questo si decide di modificare l'ADT di I classe `Graph` sostituendo l'ADT di I classe `ST` con un Quasi ADT `Vertex` che mantiene più informazioni sui vertici.

Strutture dati:

- **Quasi ADT** `Edge`: è composto da 2 campi interi contenenti gli indici dei vertici su cui insiste. È definito in `Graph.h`, come da implementazioni standard. Viene resa disponibile la funzione di creazione standard (`EDGEcreate`) realizzata in `Graph.c`.
- **Quasi ADT** `Vertex`: struct definita e implementata in `Graph.c` non visibile all'esterno, contenente:
  1. il `nome` del vertice (vettore di caratteri sovradimensionato con dimensione MAXS definita in `Graph.h`, cosicché il client possa all'occorrenza modificarlo)
  2. il suo `grado`, aggiornato durante tutte le inserzioni o rimozioni di archi che insistono su di esso
  3. un flag `valido` che indica se il vertice sia stato cancellato o meno. Viene usato per evitare di effettuare cancellazioni *fisiche* di `struct Vertex`, lavorando con cancellazioni *logiche*, più semplici ed efficienti

  Viene fornita una funzione di ricerca standard in un vettore disordinato di stringhe, la quale permette di passare dal `nome` di un vertice all'indice della struct di cui fa parte, nel vettore `vertici`. (`get_index`)
- **ADT di I classe** `Graph`**:** è un grafo non orientato e non pesato, rappresentato come matrice delle adiacenze e senza ST interna. Al posto della ST esso contiene un vettore di `struct Vertex` chiamato `vertici`. Vengono fornite le funzioni standard di inizializzazione (`GRAPHinit`) (rimossa l'inizializzazione della ST), liberazione (`GRAPHfree`) (sostituita la liberazione della ST con la liberazione del vettore di vertici), inserzione/rimozione di un arco (`GRAPHinsertE`/`GRAPHremoveE`), elencazione degli archi (`GRAPHedges`). L'acquisizione (`GRAPHload`) si occupa anche di inizializzare il vettore `vertici` allocando il numero di vertici necessario, rendendo valido ogni vertice che viene letto e aumentandone il grado durante la lettura degli archi. La funzione di verifica di connessione `connesso` è una versione semplificata della `GRAPHcc` standard che si limita a verificare se il grafo è connesso o meno controllando se vi è più di una componente connessa. Nel caso sia presente più di una componente connessa, la funzione ritorna 0 senza effettuare le rimanenti visite in profondità `dfsRcc` a partire da altri vertici.

Algoritmo:

- **`main`:**
  1. legge dal file fornito tramite linea di comando il grafo (`GRAPHload`)
  2. ne calcola il k-core, se esiste, (con k letto da tastiera) (`calcola_k_core`)
  3. verifica se è j-edge-connected (con j letto da tastiera) verificando che non esista un set di archi a cardinalità minore di j che lo sconnetta, e al contempo esista un set di archi a cardinalità uguale a j che lo sconnetta (`verifica_j_edge`)
- **`cerca_k_core`:** Calcolare il k-core di G significa trovare il sottografo massimale di G in cui ogni vertice abbia grado maggiore o uguale a k. Poiché verranno rimossi gli archi che insistono sui vertici da cancellare, per prima cosa si salva l'elenco degli archi del grafo originale

(GRAPHedges), e al termine dell'algoritmo, prima di far ritornare la funzione, si reinseriscono tutti gli archi che erano stati rimossi (insertE).
L'algoritmo consiste nel rimuovere *logicamente* ad ogni passo i vertici con grado minore del grado corrente (che varia tra 1 e k incluso) con un ciclo sul vettore vertici. In seguito un ciclo scorre la matrice di adiacenza, rimuovendo tutti gli archi che incidono su vertici che sono appena stati rimossi e diminuendo il grado di entrambi i vertici coinvolti. Si è scelto di non unificare i due cicli per maggiore chiarezza. L'intera operazione viene ripetuta finché non ci sono più né vertici né archi da rimuovere, ossia si è trovato il "grado-core" del grafo. A questo punto si incrementa la variabile grado e si ripete il tutto. Al termine si stampano i vertici rimasti validi, ossia la soluzione. Se nessun vertice è rimasto valido, non esiste il k_core del grafo in questione.
Per evitare di salvare gli archi e ripristinarli alla fine, si sarebbe potuto usare una matrice di flag parallela alla matrice di adiacenze e lavorare su quella invece che su quella originale, ma si è preferito l'uso della funzione standard GRAPHedges per non appesantire la struttura dati myGraph e non dover creare una funzione ad-hoc come la GRAPHremoveE che lavorasse sulla matrice di flag.

- **verifica_j_edge**: La soluzione proposta si basa su un'esplorazione esaustiva di tutto lo spazio delle soluzioni con il modello del Calcolo Combinatorio per trovare l'insieme di archi di cardinalità minima la cui rimozione rende il grafo sconnesso. Il modello è quello del powerset con combinazioni semplici di dimensione i crescente da 0 a j. Se la dimensione minima è i allora il grafo è i-edge-connected e non j-edge-connected. Anche nel caso in cui la dimensione minima sia maggiore di j il grafo non è j-edge-connected. Per verificarlo non è necessario trovare la dimensione in questione, ma basta appurare che per j non vi siano soluzioni e interrompere la ricerca.
  La selezione degli archi da rimuovere avviene mediante ciclo sul vettore Edges, riempito inizialmente usando la GRAPHedges, che seleziona ad ogni passaggio quale arco aggiungere alla soluzione.
  La rimozione degli archi è fatta mediante applicazione della funzione GRAPHremoveE, il ripristino mediante la GRAPHinsertE.
  Dovendo solo valutare se esiste una qualunque soluzione con la cardinalità cercata, il powerset ritorna un valore di successo o fallimento a seconda se la soluzione sia stata trovata o meno.
  Il controllo di connessione del grafo viene fatto nel caso terminale, e nel caso il grafo risulti sconnesso si stampa la soluzione trovata usando stampa_sol e si ritorna successo.
  La funzione stampa_sol è la funzione di stampa standard che si ha quando il vettore sol contiene indici al vettore di archi Edges (Edge arco=Edges[sol[i]]). Vengono quindi usati gli indici dei vertici su cui gli archi insistono, per fare accesso diretto al vettore dei vertici e stampare il nome del vertice anziché il suo indice (G->vertici[arco.v].nome).

Modifiche dalla versione cartacea:

- **Main.c:** Rimossa l'inclusione alle librerie ctype e string, non utilizzate. Aggiunta la chiusura del file.
- **Graph.h:** Aggiunti i prototipi delle funzioni standard, i prototipi di cerca_k_core e verifica_j_edge e spostato il prototipo di get_index in Graph.c.
- **Graph.c:** Rimosso int indice dalla struct Vertex (inutilizzato). Aggiunto il parametro int V indicante la dimensione del vettore vertici nel prototipo della funzione get_index. Aggiunto il parametro int k nel prototipo della funzione stampa_sol per la dimensione della soluzione e aggiunta dell'implementazione standard. Aggiunti i prototipi delle funzioni static standard e le implementazioni delle funzioni standard.

- **`GRAPHload:`** Rimossa stringa nome dalla GRAPHload (non utilizzata), aggiunta GRAPHinit standard e controllo di errore sulle get_index (anche se non necessario assumendo formato del file corretto). Modificate le funzioni GRAPHfree e GRAPHcc (chiamata connesso) come spiegato sopra.
- **`cerca_k_core:`** aggiunto salvataggio e ripristino degli archi usando le funzioni standard GRAPHedges e insertE. Spostata la diminuzione del grado dei vertici dal ciclo sui vertici al ciclo sulla matrice, associandola concettualmente alla rimozione degli archi.
- **`verifica_j_edge:`** aggiunta allocazione e liberazione del vettore Edges e modificato indice di partenza della cardinalità a 0 per considerare anche grafi già sconnessi in partenza.
- **`powerset:`** rimossa variabile isAcyclic (utilizzabile come flag nella funzione connesso, ma rimossa perché il nome era fuorviante e si è preferito usare direttamente il valore di ritorno della funzione connesso). Cambiato paradigma risolutivo: invece di usare disposizioni ripetute associate a una funzione che controlli quanti elementi sono a 1 per stabilire la cardinalità della soluzione, si usano le combinazioni semplici aggiungendo start tra i parametri della funzione.