Nome: Gabriele
Cognome: Gioetto
Matricola: s248636
Data compito: 31/01/2019

Strategie risolutive e strutture:
Per creare il grafo ho utilizzato la struttura standard del grafo con lista di adiacenza e tabella di simboli. La tabella di simboli è implementata con un vettore sovradimensionato (2 volte il numero di archi totali).
Le funzioni del grafo e della ST sono standard, a parte 'STinsert' e 'STsearch' che al posto di ritornare l'item(ovvero la stringa), ritornano l'indice della cosiddetta stringa nella ST.

Per creare il grafo ho letto due volte il file. La prima volta conto il numero di righe(ovvero di archi) per inizializzare la ST e la lista di adiacenza del grafo. La seconda volta leggo riga per riga gli identificativi dei due vertici, e per ciascun arco controllo se il vertice esiste già nella ST. Se non esiste lo aggiungo nella ST, altrimenti cerco il suo indice (STsearch).
Al fondo del ciclo creo l'arco e lo aggiungo al grafo con gli indici ricavati in precedenza.

Per identificare i vertici del grafo che appartengono a K, utilizzo un vettore 'sol' in cui i valori possono essere 1 (valore in indice i appartiene a K) oppure 0.
Per verificare se un gruppo di vertici è un kernel, ciclo il vettore due volte(un ciclo per ognuno dei controlli, si poteva fare anche con un ciclo unico).
Nel primo ciclo controllo che ogni vertice di K(sol[i] == 1) non sia connesso a un altro vertice di K. Nel secondo assegno a un vettore 'verticiAdiacenti', di lunghezza V inizializzato con tutti zeri, il valore 1 se il vertice i è appartenente al kernel oppure se esiste un arco (k,i).
Alla fine del ciclo controllo se tutti i valori del vettore sono uguali ad 1. Se la condizione è falsa, allora l'insieme K non è un kernel.

Per identificare il kernel con il numero minimo di vertici ho implementato un powerset con disposizioni ripetute. Ogni volta che ottengo una soluzione controllo che essa sia un kernel e che abbia cardinalità minore della soluzione migliore. Se questa condizione è verificata, allora salvo la nuova soluzione migliore e aggiorno la cardinalità minima.

Per trovare il cammino minimo che attraversa più vertici appartenenti al kernel, ho utilizzato un contatore che conta il numero di vertici appartenenti a K che attraverso nel cammino, un contatore che conta la lunghezza del cammino e un vettore visited che sfrutto per controllare che il cammino sia semplice.
Nella funzione ricorsiva 'lunghezzaCamSemp' controllo se il vertice corrente è appartiene al kernel. In caso affermativo aggiorno il contatore kernel. Se il contatore kernel è maggiore del numero massimo di vertici appartenenti al kernel visitati, aggiorno la soluzione migliore.
Una volta visitato il vertice corrente, visito tutti i vertici adiacenti controllando che non siano già stati visitati nello stesso cammino. Una volta finito il ciclo applico il backtrack sul vertice.
Per trovare la soluzione migliore compio la visita V volte, ogni volta partendo da un vertice differente del grafo e azzerando il vettore visited.

Differenze:

- Nella funzione ‘identificaKernelMin’, nella versione del compito non ho salvato la lunghezza del cammino, ma solamente il numero di kernel attraversati. Nella versione digitale ho creato due nuove variabili per salvare anche la lunghezza del cammino
- Nella funzione ‘lunghezzaCamSemp’ nel compito per un errore di distrazione nel ciclo non ho controllato visited[t->v], ma visited[v]
- Nel compito per mancanza di tempo non ho scritto la funzione ‘getKernel’, che è una semplice lettura da file con assegnazione di un vettore sol con valori 0/1, con 1 negli indici dei valori letti da file nella ST