Relazione prova di programmazione

Appello di “Algoritmi e Programmazione” del 02/02/2015 – 02MNOOA

Enrico Barberis – Matricola: s201030

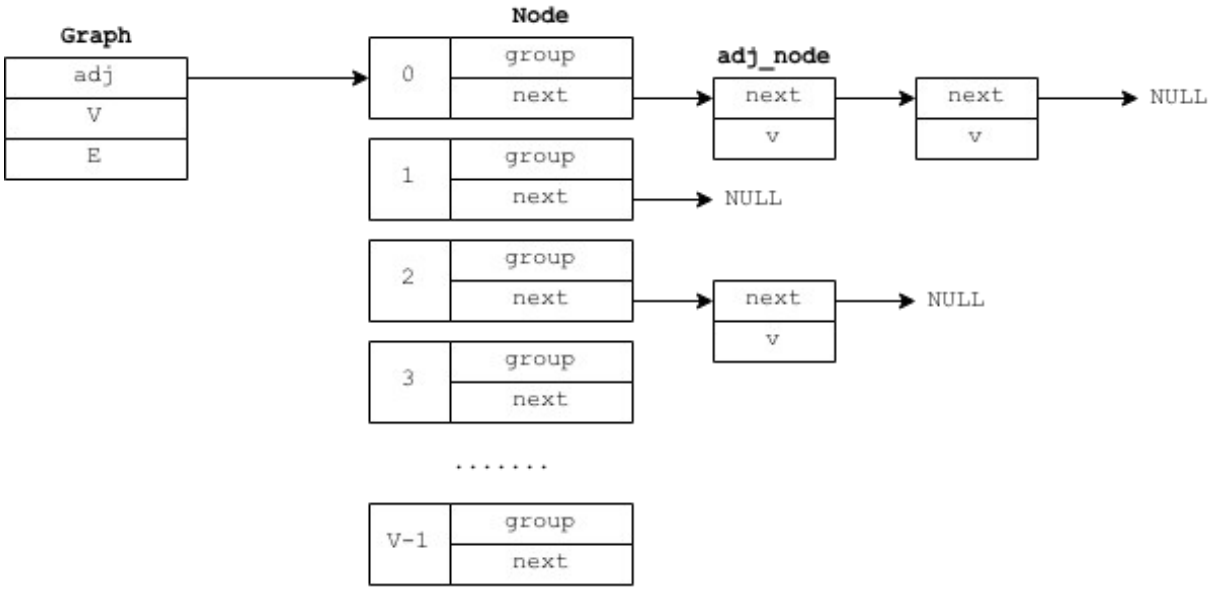
Struttura dati utilizzate

Tabella di simboli (st.c)

Per implementare la tabella di simboli `st` si è scelto di utilizzare un vettore di stringhe in modo da assegnare ad ogni stringa un numero univioco. Per semplicità l'inserimento avviene in modo diretto nella prima cella libera del vettore. Non essendoci alcun tipo di ordinamento la ricerca viene effettuata tramite una ricerca sequenziale.

Grafo (graph.c)

La strutturda dati di `graph` è basata su una lista di adiacenza come si può notare dalla figura:



Ogni nodo rappresenta un amico e i suoi nodi adiacenti indicano le sue amicizie. In ogni `node` è presente un una lista di nodi adiacenti oltre ad un numero intero `group` che identifica il gruppo di tale nodo. Il gruppo serve ad identificare le persone che hanno ricevuto lo stesso regalo: se due nodi hanno lo stesso gruppo vuol dire che hanno ricevuto lo stesso regalo.

Algoritmo utlizzato

Richiesta 1 – Lettura dati di input

Per caricare i due file di input in memoria viene prima effettuato un conteggio delle righe presenti nel file 1 contenente le relazioni di amicizia. Tale conteggio serve per stimare il numero massimo di persone, infatti non è possibile inserire più di *2*(numero di righe)* persone. In tal modo si può inizializzare la tabella di simboli per rileggere il file e conteggiare il numero di persone effettive.
Calcolato tale numero si inizializza la struttra `graph` e si rilegge un'ultima volta il file per caricare le relazioni di amicizia.
Per quanto riguarda il file 2, contenente la soluzione da controllare, viene letto una sola volta per impostare il gruppo di ogni persona tramite la funzione `graph_set_group`.
Nell' elaborato originale manca il controllo dell'uncità del regalo. La correzione risulta immediata aggiungendo una nuova tabella simboli per verificare che ogni persona compaia una ed una sola volta in tale file.

Richiesta 2 – Verifica della soluzione

La verifica della soluzione avviene tramite la funzione `graph_check` e al pre-controllo nel *main.c* durante la lettura del file 2.
Tale funzione scandisce tutti i nodi e verifica che ogni suo nodo adiacente appertenga ad un gruppo diverso. Appena viene trovato un nodo adiacente con lo stessso gruppo del nodo la funzione ritorna il valore 0, altrimenti se questo non accade mai la funziona ritorna il valore 1.

Richiesta 3 – Generazione di una soluzione ottimale

La generazione della soluzione ottimale avviene tramite la funzione `colora` in *graph.c*
Tale funzione ha come prototipo:

```
void colora(Graph g, int pos, int *min, int *sol)
```

Il parametro `pos` serve per sapere a che livello di ricorsione si è arrivati, mentre i parametri `min` e `sol` sono utilizzati per la memorizzazione della soluzione ottimale.
La funzione al passo generico assegna al nodo in poiszione `pos` nel vettore `g->adj` tutti i gruppi possibili, ovvero da `0` a `V-1` compreso. Dopo l'assegnazione viene richiamata la funzione ricorsivamente cambiano il parametro pos in `pos+1`.
In questo modo si provano tutte le combinazioni possibili dei gruppi per ogni nodo.
Non appena `pos` arriva ad essere >= di `V` la funziona verifica se l'attuale configurazione di gruppi sia valida o meno tramite la funzione `graph_check`. In caso affermativo viene controllato se tale soluzione richiede un numero minore di gruppi dell' attuale soluzione ottimale e se ciò accade la soluzione corrente viene memorizzata come ottimale.

Considerazioni finali

Nella correzione dell' elaborato ho notato la presenza di errori di distrazione come ad esempio la mancata allocazione dinamica di un vettore e il mancato richiamo delle funzioni di liberazione di memoria nella fine del *main*.
L'errore più consistente è il mancato controllo dell'uncità del regalo. Tale errore è stato indotto dalla lettura affrettata del problema che mi ha portato ad intendere erroneamente la richiesta come un'assunzione.
L'unico errore di logica presente si trova nella condizione per fermare la funzione ricorsiva `colora`. Infatti i test sulla posizione e sulla validità in un unico `if` causano una ricorsione infinita con conseguente stack overflow. La correzione è immediata spezzando in due `if` tali condizioni.

La complessità della soluzione è $O(|V|+|E|)$ per la funzione `graph_check`, mentre per la funzione `colora` è $O(n^n)$ dove n è il numero di amici.

Nel sorgente allegato tutte le correzioni effettuate sono contrassegnate da un commento del tipo: `//[MODIFICA]`