Diego Signorastri S6307930

Laboratorio Individuale ASD 2023-2024

SCELTE IMPLEMENTATIVE

Il punteggio massimo di quattro punti si raggiunge non solo se le funzioni e le strutture dati sono corrette, ma anche se sono implementate in modo efficiente e non ci sono problemi "di stile". In particolare, sono elementi apprezzati ai fini della valutazione (l'elenco non è esaustivo):

E' richiesto di implementare in modo efficiente le funzioni e strutture dati necessarie per il corretto funzionamento del network. Il significato di efficienza non è assoluto, dipende dalle esigenze. In questo caso non vi è una caratteristica precisa sull'efficienza e di conseguenza mi sembra giusto equilibrare le scelte implementative sia rispetto al dispendio temporale, sia rispetto a quello spaziale. Sappiamo che il fine è quello di implementare un social network nel quale le persone solitamente si iscrivono e si eliminano molto spesso da gruppi, inoltre anche la ricerca è molto utilizzata. La visualizzazione di dati di questo network richiede che utenti,gruppi e amicizie siano ordinati alfabeticamente. Se l'ordine non fosse importante un hashmap potrebbe essere una buona soluzione, ma visti i requisiti di ordinamento ho optato per ridurre il campo a strutture ordinate.
Come dicevo non c'è un' operazione che è poco usata in un social network e quindi il mio fine, quando pensavo all'implementazione a mio avviso ottimale, era di trovare una struttura ordinata che avesse una complessità in ricerca,eliminazione e inserimento bilanciata e non dispendiosa. E' per questo che ho optato per la scelta di un albero bilanciato.
Più precisamente ho scelto la tipologia di albero bilanciato AVL. Quest'ultimo si autobilancia durante le operazioni di inserimento ed eliminazione basandosi sul coefficiente di bilanciamento.
Questa struttura permette ricerca,inserimento ed eliminazione in tempo logaritmico anche nel caso peggiore. Inoltre possiamo molto facilmente stampare o restituire in modo ordinato tutti gli elementi del network.

Alcune funzioni potevano essere più efficienti temporalmente se si dedicava altro spazio in memoria ad esempio ai gruppi. Ogni utente può contenere un AVL di gruppi a cui partecipa e il network contiene tutti i gruppi esistenti. Facendo così ad esempio memberOf() sarebbe passata da complessità pseudo lineare (n * logn) a lineare (n) ma utilizzando molto più spazio. Io ho scelto di duplicare il meno possibile i dati al fine di evitare inconsistenze e spreco di spazio.

CONCLUSIONE

Il network è semplicemente un contenitore di:

- AVL tree contenente i gruppi
- AVL tree contenente i membri del network

Un gruppo è una struttura contenente:

- AVL tree di membri (utenti)
- nome(identificatore)
- creatore (Utente)

Un utente contiene:

- nome(identificatore)
- AVL tree di amicizie (Utenti)

NOTA: ho preferito che nelle amicizie ogni nodo contenesse nome utente e puntatore a nome utente. Si poteva evitare di tenere il nome utente ma ho preferito lasciarlo come chiave del AVL tree al fine di poter rendere generica la struttura. Se in futuro volessimo aggiungere la data data di nascita all'utente o cambiare l'identificativo sarà più semplice.

COMPILAZIONE ED ESECUZIONE

```
 g++ -std=c++14 -Wall network-test.cpp list-array.cpp network.cpp -o
social
./social
```

Nota: ho utilizzato i template, l'implementazione della struttura è inclusa in un .tpp