Biondi Tommaso 5797190

LABORATORIO INDIVIDUALE

Si vogliono implementare le funzioni di base di un social network, in particolare le funzioni relative alla gestione di **utenti**, **gruppi** di utenti e **amici**.

Utenti:

Per la gestione degli utenti, è necessario un tipo di dato che permetta di gestire un gran numero di utenti e di accedervi rapidamente. Una buona implementazione potrebbe essere un dizionario basato su alberi di ricerca binaria (BST) I BST permettono una gestione efficiente di una grande mole di dati e consentono di stampare i dati in maniera ordinata tramite una traversata in ordine simmetrico (in-order-trasversal). Questo è particolarmente utile per creare un elenco di utenti in ordine alfabetico.

Per ottimizzare ulteriormente l'efficienza degli utenti ogni utenti ho implementato un array di 52 elementi ogni cella punta a un albero contenente solo gli utenti con usr_Log con iniziale uguale alla posizione della cella, prima con tutte le lettere maiuscole poi con lettere minuscole

es l'albero in posizione zero conterrà tutti gli usr_Log che iniziano con 'a', quello in posizione 1 conterrà tutti gli usr_Log con iniziale 'b' etc, l'albero in posizione 26 conterrà tutti gli usr_Log con nome utente 'A'.

Si tratta di una implementazione che può rendere le operazioni necessarie per accedere a un elemento su un albero 52 volte più rapide in media, su una complessità che è già logaritmica, quindi non fondamentale ma il costo per l'implementazione è tampo basso che risulta vantaggioso implementarlo.

Grazie a questa implementazione sappiamo che l'albero a cui punta la cella in posizione i-esima conterrà gli usr_Log che in ordine alfabetico saranno precedenti a quelli presenti nell'albero della i+1-esima posizione, quin

Amici:

Per salvare una lista di amici per ogni utente, è necessario un tipo di dato che permette di stampare questa lista in ordine alfabetico. Anche in questo caso, i BST sono una scelta adatta. La struttura deve gestire un numero di inserimenti, cancellazioni e ricerche che può variare significativamente a seconda delle relazioni sociali degli utenti. I BST garantiscono una complessità O(H) per inserimenti, cancellazioni e ricerche, rendendo efficiente la gestione delle liste di amici, a patto che l'albero rimanga bilanciato.

Le amicizie sono simmetrica (ovvero se un utente A e amico di un utente B, allora anche quest'ultimo è amico di A), di conseguenza per memorizzare l'informazione "Amicizia(A,B)", basterebbe inserire l'informazione una sola volta all'interno dell'albero di amicizie A o di B, tuttavia con questo approccio, nel caso si desideri verificare se A,B sono amici come nel caso della funzione (areFriends) bisognerebbe controllare la presenza della loro amicizia, sia nell'albero di amicizie di A, sia in quello di B nel caso peggiore, e questo renderebbe la funzione molto meno efficiente, ho dunque scelto di creare una ridondanza di informazioni, salvando l'informazione dell'amicizia, sia nell'utente A, sia nell'utente B, impiegando il doppio della memoria necessaria per salvare le amicizie, ma rendendo l'operazioni di confronto fra amicizie molto più efficiente

Gruppi:

Per la gestione dei gruppi, è necessario un tipo di dato che permette di gestire l'elenco dei gruppi in maniera efficiente e ordinata. Come per gli utenti ho implementato un array di 52 elementi in cui ogni cella è un puntatore a un albero contenente tutti i gruppi aventi la stessa iniziale ordinati in ordine alfabetico. I BST sono una scelta ottimale per mantenere i gruppi ordinati per nome e garantire operazioni efficienti. Ogni utente avrà accesso a un BST che elenca tutti i gruppi d i cui fa parte e di cui è capo, rendendo più efficienti le funzioni memberOf e creatorOf

Quest'ultima caratteristica non è essenziale e si tratta di una ridondanza, poiché sarebbe possibile far eseguire le funzioni memberOf e creatorOf senza implementare gli alberi sotto ogni utente riguardanti i gruppi di cui è creatore e di cui partecipa. Tuttavia, ciò sarebbe molto meno efficiente, quindi preferiamo sacrificare un po' di memoria in cambio di un notevole incremento delle prestazioni. Considerando inoltre che un social efficiente porterà abbastanza entrate da poter sostenere l'ampliamento dello spazio di memoria e di gestione, mentre un social inefficiente non porterà entrate.

Fonti:

Per l'implementazione dei BST ho ripreso la libreria che avevo progettato per il laboratorio 6

Codice per compilare:

codice per compilare g++ -wall std=c++11 -o o *.cpp