Esercizi 03-12-2024

Liste e code con priorità

1

Scrivere le implementazioni delle funzioni definite all'interno del file `PriorityQ.h` in un file `PriorityQ.cc`. Il file `PriorityQ.h` e il main del programma possono essere scaricati dalla cartella `priority` su Google Drive. La struttura dati da implementare è una sorta di coda a priorità dove ad ogni elemento è appunto associata una priorità.

Quando si deve rimuovere un elemento dalla coda, dovrà essere rimosso per primo l'elemento con priorità più bassa. Per semplificare l'esercizio, la priority Queue è definita come una struttura con un array di code, ciascuna con la propria priorità.

Ciascuna coda deve essere implementata con una lista doppiamente linkata, quindi ogni nodo avrà due puntatori, uno per il successore ed uno per il predecessore.

Note:

- Quando si cerca di estrarre un elemento da una coda vuota dovrà essere restituito il valore -1
- Fare attenzione a come viene deallocata la memoria!

2

Scrivere la dichiarazione e la definizione di una funzione `unisciListe`, che prende in input due puntatori a due liste concatenate contenenti `float`, e restituisce il primo nodo di una nuova lista concatenata, che contiene prima gli elementi della prima lista e poi gli elementi della seconda.

Una volta scritta questa funzione, scrivere una seconda funzione `scorriLista`, che prende in input un puntatore a una lista concatenata, rimuove gli elementi maggiori di 0, e restituisce la somma degli elementi appena rimossi.

Il programma deve terminare correttamente, e per fare ciò la memoria va correttamente deallocata. Implementare la funzione `deallocaLista`, in modo che si occupi di questa mansione.

Note:

Scaricare il file `1-ori.cc` da Google Drive e inserire la corretta implementazione delle funzioni `unisciListe`, `scorriLista` e `deallocaLista`. Il file arriva già fornito con alcune funzioni ausiliarie per le liste ed un main.

Il programma deve essere in grado di riconoscere il contesto dell'input. In particolare:

- se una delle due liste è vuota, verrà restituito un puntatore all'altra;
- se entrambe sono vuote, verrà restituito un puntatore a `NULL`;
- se vengono dati puntatori a nodi centrali (e non all'inizio della lista), il programma dovrà restituire un errore.

Le dimensioni delle liste in input sono generate casualmente e non è possibile fare assunzioni sulle dimensioni.

- Ricordarsi (importante!) di deallocare la memoria.
- È consentito definire ed implementare funzioni ausiliarie che possano aiutarvi nella soluzione del problema.
- All'interno di questo programma non è ammesso l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream`.

Alberi

3

L'ingegner Fuffox ha sviluppato uno strano metodo per superare l'esame di Programmazione 1 senza dover studiare. Ovviamente, per imparare il metodo è necessario seguire un seminario tenuto nientemeno che da Fuffox stesso.

Data l'esclusività del metodo, per partecipare bisogna essere invitati: compreso Fuffox, ogni persona può invitare al massimo 3 persone, che a loro volta possono invitare al massimo 3 persone, e così via. Questa organizzazione gerarchica è rappresentata dalla struttura piramide definita nel file `3-ori.cc`, composta da persone aventi un codice d'invito e un numero variabile (fino a 3) di persone invitate. Al vertice della piramide sta ovviamente Fuffox.

L'ing. Fuffox vuole anche capire quanto facilmente si diffondano gli inviti per il suo seminario. Scrivere la funzione ricorsiva `altezzaPiramide` che restituisca l'altezza della piramide, definita come il numero massimo di persone collegate da un invito, partendo da Fuffox (compreso) fino alla persona più in fondo alla piramide.

Scrivere la funzione ricorsiva `calcolaGuadagno` che permetta all'ing. Fuffox di scoprire quanto potrà guadagnare da questo seminario. Per partecipare infatti è necessario versare una quota di partecipazione alla persona che ti ha invitato. Chi riceve delle quote d'invito può tenerne soltanto una percentuale definita dalle commissioni (`ricavoPersonale = commissioni * quoteInvito`). I soldi rimanenti quindi devono essere mandati alla persona di livello superiore (in aggiunta alla quota personale).

Vista la dubbia legalità dell'intera operazione, l'ing. Fuffox vuole una strategia per sbarazzarsi velocemente di tutti i dati. Nonostante sia un ingegnere, sa che la memoria dinamica va sempre deallocata responsabilmente. Scrivere pertanto la funzione `deallocaPiramide` che svolga questa operazione correttamente.

NOTA: È necessario non fare assunzioni su come è costruita la piramide. Le funzioni devono funzionare anche se una persona ha distribuito meno di 3 inviti.