ASD a.a. 2023/24 - Esame del 10 febbraio 2025

Prima di cominciare lo svolgimento leggete attentamente tutto il testo.

Questa prova è organizzata in tre esercizi.
Vi forniamo un file zip che contiene per ogni esercizio: un file per completare la funzione da scrivere e un programma principale per lo svolgimento di test specifici per quella funzione. Ad esempio, per l'esercizio 1, saranno presenti un file es1.cpp e un file es1-test.o. Per compilare dovrete eseguire `g++ -std=c++11 -Wall es1.cpp hash-func.cpp es1-test.o -o es1-test`. E per eseguire il test, `./es1-test`. Dovete lavorare solo sui file indicati in ciascuno esercizio. Modificare gli altri file è sbagliato (ovviamente a meno di errata corrige indicata dai docenti).
In questi file dovete implementare le funzioni richieste, esattamente con la *segnatura* con cui sono indicate: nome, tipo restituito, tipo degli argomenti nell'ordine in cui sono dati. Non è consentito modificare queste informazioni. Potete invece fare quello che volete all'interno del corpo delle funzioni: in particolare, se contengono già una istruzione `return`, questa è stata inserita provvisoriamente per rendere compilabili i file ancora vuoti, e **dovrete modificarla in modo appropriato**.
Potete inoltre realizzare altre funzioni in tutti i casi in cui lo ritenete appropriato. Potete inserirvi tutti gli `#include` che vi servono oltre a quello relativo allo header con le funzioni da implementare. Attenzione però che **usare una funzione di libreria per evitare di scrivere del codice richiesto viene contato come errore** (esempio: se è richiesto di scrivere una funzione di ordinamento, usare la funzione `std::sort()` dal modulo di libreria standard `algorithm` è un errore).
Per ciascuno esercizio, vi diamo uno programma principale, che esegue i test. Controllate durante l'esecuzione del programma, quanti sono i test che devono essere superati e controllate l'esito (se non ci sono errori deve essere `SI` per tutti).

NB1: soluzioni particolarmente inefficienti potrebbero non ottenere la valutazione anche se forniscono i risultati attesi. Di contro ci riserviamo di premiare con un bonus soluzioni particolarmente ottimali.
NB2: superare positivamente tutti i test di una funzione non implica soluzione corretta e ottimale (e quindi valutazione massima).

1 Presentazione della struttura dati

L'obiettivo di questa prova di laboratorio è programmare tre funzioni per un tipo che memorizzi insiemi di interi positivi dentro una doppia tabella di hash. Una doppia tabella hash è una struttura dati che estende le tabelle hash per limitare le collisioni. Una doppia tabella hash contiene due array `T1` e `T2` (della stessa dimensione `tableDim`) ai quali sono associati due funzioni di hash `h1` e `h2`. Nella nostra versione di doppia tabella, ogni cella punta verso una lista doppiamente collegata **ordinata** in modo crescente. L'idea è che quando si vuole inserire un nuovo valore `v` nella doppia tabella, questo elemento verrà inserito nella lista `T2[h2(v)]` se questa lista è di lunghezza strettamente più piccola della lista `T1[h1(v)]`, altrimenti `v` sarà inserito nella lista `T1[h1(v)]`. Le liste vuote sono rappresentate da `nullptr`.
IMPORTANTE: Un valore `v` è presente soltanto una volta nella doppia tabella, ovvero non può essere sia in `T2[h2(v)]` che in `T1[h1(v)]`.
Alla fine di questo documento forniremo degli esempi. In questi esempi, se un puntatore punta a `nullptr`, esso non sarà rappresentato. Per questi esempi, supponiamo che `h1(x)=x%5` e `h2(x)=(x/5)%5`.
Nel file `double-hash.h` troverete la descrizione della struttura dati e i prototipi delle tre funzioni da implementare. **Non è consentito modificare questo file!** Questo file è strutturato nel seguente modo:

```
#ifndef DOUBLE_HASH_H
#define DOUBLE_HASH_H

typedef unsigned int Elem;

struct cell{
  Elem elem;
  cell* next;
  cell* prev;
};

typedef cell* dllist;

const dllist emptydllist=nullptr;

const int tableDim=10;

struct dhash_table{
  dllist* T1;
  dllist* T2;
};
```

```
unsigned int h1(Elem);

unsigned int h2(Elem);

/***********************************************/
/* Funzione da implementare                    */
/***********************************************/
//Es 1
//Ritorna il numero di elementi nella tabella
unsigned int nbElem(const dhash_table&);

//Es 2
//Verifica se un elemento e' presente
bool isPresent(const dhash_table&,Elem);

//Es 3
//Inserisce un nuovo elemento
void insert(dhash_table&,Elem);
```

Nel file `hash-func.cpp` troverete l'implementazione delle due funzione di hash `h1` e `h2`. **Non è consentito modificare questo file!**

2 Esercizio 1

Nel file `es1.cpp`, dovete implementare la funzione `unsigned int nbElem(const dhash_table& dht)`. Questa funzione deve restituire il numero totale di elementi presenti nella doppia tabella `dht`.
Esempi con gli insiemi dati alla fine di questo documento:

- `nbElem(dht1)` => 7
- `nbElem(dht2)` => 8
- `nbElem(dht3)` => 9
- `nbElem(dht4)` => 10
- `nbElem(dht5)` => 11

Per testare questa funzione, potete usare il file `es1-test.o` compilando con il comando:
`g++ -std=c++11 -Wall es1.cpp hash-func.cpp es1-test.o -o es1-test`.

3 Esercizio 2

Nel file `es2.cpp`, dovete implementare la funzione `bool isPresent(const dhash_table& dht,Elem e)`. Questa funzione ritorna `true` se l'elemento `e` è già presente nella doppia tabella `dht` e false altrimenti.
Esempi con gli insiemi dati alla fine di questo documento:

- `isPresent(dht1,7)` ritorna `true`
- `addPair(dht1,8)` ritorna `false`
- `addPair(dht1,22)` ritorna `true`

Per testare questa funzione, potete usare il file `es2-test.o` compilando con il comando:
`g++ -std=c++11 -Wall es2.cpp hash-func.cpp es2-test.o -o es2-test`.

4 Esercizio 3

Nel file `es3.cpp`, dovete implementare la funzione `void insert(dhash_table& dht,Elem e)`. Questa funzione aggiunge l'elemento `e` alla doppia tabella. Se l'elemento è già presente, la funzione non esegue alcuna azione.
Esempi con gli insiemi dati alla fine di questo documento. Ricordiamo che per questi esempi abbiamo `h1(x)=x%5` e `h2(x)=(x/5)%5`:

- `insert(dht1,5)` cambia `dht1` in `dht2` (con `h1(5)=0` e `h2(5)=0`)
- `insert(dht1,15)` non cambia `dht1`
- `insert(dht1,2)` cambia `dht2` in `dht3` (con `h1(2)=0` e `h2(2)=0`)

- `insert(dht3,47)` cambia `dht3` in `dht4` (con `h1(47)=2` e `h2(47)=4`)
- `insert(dht4,18)` cambia `dht4` in `dht5` (con `h1(18)=3` e `h2(18)=3`)

Per testare questa funzione, potete usare il file `es3-test.o` compilando con il comando:
`g++ -std=c++11 -Wall es3.cpp hash-func.cpp es3-test.o -o es3-test`.

5 Consegna

Per la consegna, creare uno zip con tutti i file forniti.