

UNIVERSITÀ DEL PIEMONTE ORIENTALE

Dipartimento di Scienze e Innovazione Tecnologica

Facsimile dell'Esame di Algoritmi 1 – Sperimentazioni (VC)

GG mese AAAA

Testo d'Esame

Esercizio 1

Implementare un algoritmo che, dato un albero binario di ricerca (BST), dica se il **BST non è vuoto**.

Un BST non è vuoto se contiene almeno un nodo. Se il BST non è vuoto, l'algoritmo deve ritornare un valore diverso da 0, altrimenti deve ritornare 0.

L'algoritmo implementato deve essere ottimo, nel senso che non deve visitare parti del BST inutili ai fini dell'esercizio, e la complessità temporale nel caso peggiore dev'essere $O(1)$.

* * *

La funzione da implementare si trova nel file `exam.c` e ha il seguente prototipo:

```
int upo_bst_is_not_empty(const upo_bst_t bst);
```

Parametri:

- `bst`: BST.

Valore di ritorno:

- Se il BST non è vuoto: valore diverso da 0.
- Se il BST è vuoto: valore 0.

Il tipo `upo_bst_t` è dichiarato in `include/upo/bst.h`. Per confrontare il valore di due chiavi del BST si utilizzi la funzione di comparazione memorizzata nel campo `key_cmp` del tipo `upo_bst_t`, la quale ritorna un valore `<`, `=`, o `>` di zero se il valore puntato dal primo argomento è minore, uguale o maggiore del valore puntato dal secondo argomento, rispettivamente.

Nella propria implementazione è possibile utilizzare tutte le funzioni dichiarate in `include/upo/bst.h`. Nel caso s'implementino nuove funzioni, i prototipi e le definizioni devono essere inserite nel file `exam.c`.

Il file `test/bst_is_not_empty.c` contiene alcuni casi di test tramite cui è possibile verificare la correttezza della propria implementazione. Per compilarlo con la propria implementazione, è sufficiente eseguire il comando:

```
make clean all
```

Esercizio 2

Implementare un algoritmo che, data una tabella hash con gestione delle collisioni basata su concatenazioni separate (HT-SC), dica se la **HT-SC non è vuota**.

Una HT-SC non è vuota se contiene almeno una chiave. Se la HT-SC non è vuota, l'algoritmo deve ritornare un valore diverso da 0, altrimenti deve ritornare 0.

L'algoritmo implementato deve essere ottimo, nel senso che non deve visitare parti della HT-SC inutili ai fini dell'esercizio, e la complessità temporale nel caso peggiore dev'essere $O(1)$.

* * *

La funzione da implementare si trova nel file `exam.c` e ha il seguente prototipo:

```
int upo_ht_sepchain_is_not_empty(const upo_ht_sepchain_t ht);
```

Parametri:

- `ht`: HT-SC.

Valore di ritorno:

- Se la HT-SC non è vuota: valore diverso da 0.
- Se la HT-SC è vuota: valore 0.

Il tipo `upo_ht_sepchain_t` è dichiarato in `include/upo/hashtable.h`. Per confrontare il valore di due chiavi si utilizzi la funzione di comparazione memorizzata nel campo `key_cmp` del tipo `upo_ht_sepchain_t`, la quale ritorna un valore `<`, `=`, o `>` di zero se il valore puntato dal primo argomento è minore, uguale o maggiore del valore puntato dal secondo argomento, rispettivamente. Per calcolare il valore hash di una chiave si utilizzi la funzione di hash memorizzata nel campo `key_hash` del tipo `upo_ht_sepchain_t`, la quale richiede come parametri il puntatore alla chiave di cui si vuole calcolare il valore hash e la capacità totale della HT-SC (memorizzata nel campo `capacity` del tipo `upo_ht_sepchain_t`). Per tenere traccia della dimensione della HT-SC, si utilizzi il campo `size` del tipo `upo_ht_sepchain_t`. Infine, gli slot della HT-SC sono memorizzati nel campo `slots` del tipo `upo_ht_sepchain_t`, che è una sequenza di slot, ciascuno dei quali di tipo `upo_ht_sepchain_slot_t` e contenente il puntatore alla propria lista delle collisioni.

Nella propria implementazione è possibile utilizzare tutte le funzioni dichiarate in `include/upo/hashtable.h`. Nel caso si implementino nuove funzioni, i prototipi e le definizioni devono essere inserite nel file `exam.c`.

Il file `test/ht_sepchain_is_not_empty.c` contiene alcuni casi di test tramite cui è possibile verificare la correttezza della propria implementazione. Per compilarlo con la propria implementazione, è sufficiente eseguire il comando:

```
make clean all
```

Informazioni Importanti

Superamento dell'Esame

Un esercizio della prova d'esame viene considerato corretto se tutti i seguenti punti sono soddisfatti:

- è stato svolto,
- è conforme allo standard ISO C11 del linguaggio C,
- compila senza errori,
- realizza correttamente la funzione richiesta,
- esegue senza generare errori,
- non contiene *memory-leak*,
- è ottimo dal punto di vista della complessità computazionale e spaziale.

Per verificare la propria implementazione è possibile utilizzare i file di test nella directory `test`, oppure, se si preferisce, è possibile scriverne uno di proprio pugno. Per verificare la presenza di errori è possibile utilizzare i programmi di debug *GNU GDB* e *Valgrind*.

In ogni caso, l'implementazione deve funzionare in generale, indipendentemente dai casi di test utilizzati durante l'esame. Quindi, il superamento dei casi di test nella directory `test` è una *condizione necessaria ma non sufficiente al superamento dell'esame*.

Istruzioni per la Consegna

- L'unico elaborato da consegnare è il file `exam.c`.
- All'interno del file `exam.c` occorre inserire il proprio nome, cognome e numero di matricola.
- La consegna avviene tramite il caricamento del file `exam.c` nell'apposito form sul sito D.I.R. indicato dal docente.

Gli elaborati consegnati che non rispettano tutte le suddette istruzioni (ad es., mancanza dei dati identificativi dello studente all'interno del file `exam.c`) o che vengono consegnati in ritardo, non saranno soggetti a valutazione.

Regolamento d'Esame

In aggiunta alle regole generali fornite dall'Ateneo e a quelle deliberate dal Consiglio del Corso di Studi, valgono le seguenti regole:

1. Lo studente deve presentarsi all'esame con un documento di riconoscimento valido.
2. Durante la prova d'esame **non è consentito**:
 - uscire dal campo visuale dell'inquadratura della webcam;
 - disconnettersi dalla rete;
 - comunicare in qualunque modo con altri individui (docente escluso);
 - utilizzare, o avere a portata di utilizzo, dispositivi elettronici che permettano l'accesso a Internet o lo scambio di comunicazioni (ad es., computer, tablet, telefoni cellulari, smartwatch, ...), al di fuori di quelli necessari a sostenere l'esame a distanza;
 - utilizzare libri, appunti e altro materiale didattico (cartaceo o digitale), ad eccezione del materiale eventualmente fornito dal docente.
3. Durante la prova d'esame **è consentito** tenere una bottiglia di acqua.

Qualora lo studente violi una delle suddette condizioni, o sia colto in flagranza durante l'atto di copiare o se ne appuri a posteriori durante la correzione della prova d'esame, il docente ha la facoltà di bocciarlo e di segnalare il fatto agli organi d'Ateneo competenti (come il Consiglio del Corso di Studi), i quali potranno prendere ulteriori provvedimenti. Le stesse regole si applicano anche agli studenti che permettono che la loro prova d'esame venga copiata o che si prestino a svolgere la prova per conto di altri. Inoltre, qualora lo studente consegni la sua prova d'esame priva dei suoi dati identificativi, o la consegni in ritardo, dopo che il docente ha già effettuato il ritiro delle altre, la sua prova non sarà valutata e il suo tentativo d'esame verrà conteggiato al pari di un ritiro.

Comandi utili

- Comando di compilazione tramite GNU GCC:

```
gcc -Wall -Wextra -std=c11 -pedantic -g -I./include -o eseguibile sorgente1.c sorgente2.c ... -L./lib -lupoalglib
```
- Comando di compilazione tramite GNU Make:

```
make clean all
```
- Comando di debug tramite GNU GDB:

```
gdb ./eseguibile
```
- Verifica di memory leak e accessi non validi alla memoria tramite Valgrind:

```
valgrind --tool=memcheck --leak-check=full ./eseguibile
```
- Manuale in linea di una funzione standard del C:

```
man funzione
```