Università del Piemonte Orientale

Dipartimento di Scienze e Innovazione Tecnologica

Esame di Algoritmi 1 – Sperimentazioni (VC)

14 febbraio 2024

Testo d'Esame

Esercizio 1 (max 15 punti)

Implementare un algoritmo che, dato un albero binario di ricerca (BST), una chiave k (non necessariamente contenuta nel BST) e un intero n, restituisca:

- l' **n-esima chiave più piccola** del sottoalbero la cui radice ha come chiave k, se esiste e se k è contenuta nel BST;
- *NULL*, se l'n-esima chiave più piccola non esiste, se k non è contenuta nel BST, o se il BST è vuoto.

.

Si noti che l'**n-esima chiave più piccola** è la chiave che si troverebbe nell'n-esima posizione se le chiavi fossero disposte in ordine di grandezza. Per esempio, dato l'albero di Figura 1, si ha:

- $n = 3$ e $k = 3$: 4
- $n = 2$ e $k = 6$: 6
- $n = 2$ e $k = 8$: 3
- $n = 4$ e $k = 14$: *NULL*
- $n = 3$ e $k = 17$: *NULL*

L'algoritmo implementato dev'essere ottimo, nel senso che deve visitare l'albero una sola volta e non deve visitare sotto-alberi inutili ai fini dell'esercizio, e la complessità temporale nel caso peggiore dev'essere $O(n)$, dove n è il numero di chiavi nel BST.

* * *

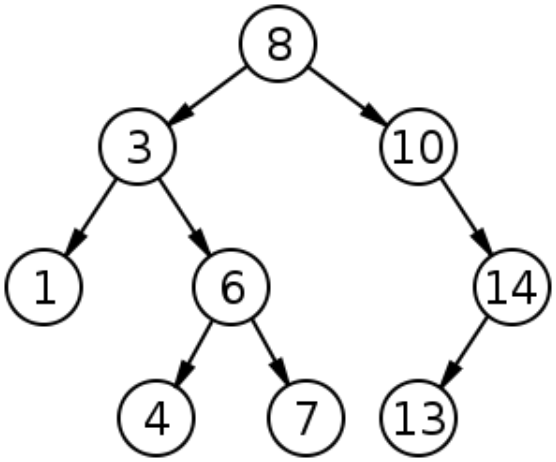

Figura 1: Un esempio di BST.

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

```
void *upo_bst_nmin(const upo_bst_t tree, const void *key, const int n)
```

Parametri:

- `tree`: BST.
- `key`: puntatore alla chiave della radice del sottoalbero di BST in cui si vuole trovare l'n-esima chiave più piccola.
- `n`: intero che definisce n-esima chiave più piccola.

Valore di ritorno:

- Se il BST non è vuoto, la chiave `key` è contenuta nel BST e l'n-esima chiave più piccola esiste: il puntatore all'n-esima chiave più piccola.
- Se il BST è vuoto o la chiave `key` non è contenuta nel BST o non esiste l'n-esima chiave più piccola: $NULL$.

Il tipo `upo_bst_t` è dichiarato in `include/upo/bst.h`. Per confrontare il valore di due chiavi (qualora fosse necessario) 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_nth_minimum_key.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
```

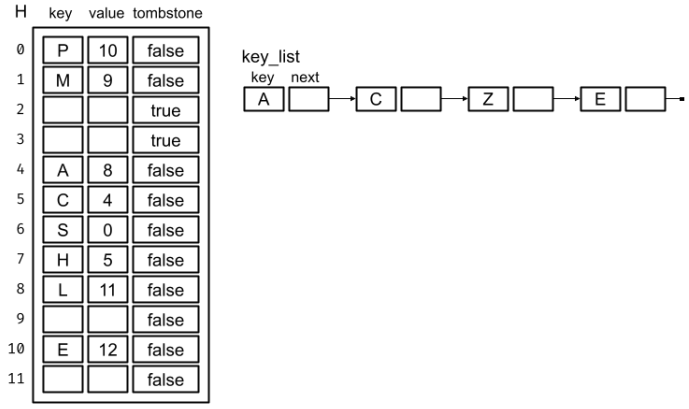

Figura 2: Un esempio di HT-LP.

Esercizio 2 (max 15 punti)

Implementare un algoritmo che, data una tabella hash H, con gestione delle collisioni basata su indirizzamento aperto (i.e. open addressing) e scansione lineare (i.e. linear probing) con uso di *tombstone* (HT-LP), e una lista di chiavi l_{keys}, calcoli la media del numero di collisioni delle chiavi k contenute in l_{keys} in H. In particolare:

- Se una chiave k in l_{keys} non è contenuta in H, non dev'essere considerata nel calcolo della media del numero di collisioni.
- Se H è vuota, o se l_{keys} è vuota, o se nessuna chiave di l_{keys} è contenuta in H, l'algoritmo deve restituire il valore -1.

Si noti che nel calcolare il numero di collisioni di una chiave k non si deve tenere conto dello slot in cui k è memorizzata e che uno slot *tombstone*, se attraversato, è considerato una collisione.

Per esempio, date la tabella e la lista di chiavi in Figura 2, e supponendo che il valore hash delle chiavi A, C, e E sia rispettivamente $0, 0$, e 10, **il numero medio di collisioni è** 3 in quanto:

- il numero di collisioni di A è 4;
- il numero di collisioni di C è 5;
- il numero di collisioni di E è 0;
- Z non è contenuta in H quindi non viene conteggiata nel calcolo della media.

L'algoritmo implementato deve essere ottimo, nel senso che non deve visitare parti di HT-LP inutili ai fini dell'esercizio.

* * *

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

```
double upo_ht_linprob_avg_collisions(const upo_ht_linprob_t ht, const upo_ht_key_list_t key_list)
```

Parametri:

- `ht`: HT-LP
- `key_list`: lista concatenata di chiavi.

I tipi `upo_ht_linprob_t` e `upo_ht_key_list_t` sono dichiarati in `include/upo/hashtable.h`. Le chiavi di cui calcolare la media del numero di collisioni sono memorizzate nel campo `key` del tipo `upo_ht_key_list_t`. Per scorrere la lista di chiavi `key_list` si utilizzi il puntatore memorizzato nel campo `next` del tipo `upo_ht_key_list_t`. Per confrontare il valore di due chiavi si utilizzi la funzione di comparazione memorizzata nel campo `key_cmp` del tipo `upo_ht_linprob_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_linprob_t`, la quale richiede come parametri il puntatore alla chiave di cui si vuole calcolare il valore hash e la capacità totale della HT-LP (memorizzata nel campo `capacity` del tipo `upo_ht_linprob_t`). Infine, gli slot della HT-LP sono memorizzati nel campo `slots` del tipo `upo_ht_linprob_t`, che è una sequenza di slot, ciascuno dei quali di tipo `upo_ht_linprob_slot_t`.

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_linprob_avg_collisions.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`.
- 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 o che vengono consegnati in ritardo, non saranno soggetti a valutazione.

Regolamento d'Esame

1. Lo studente deve presentarsi all'esame con un documento di riconoscimento valido.
2. Durante la prova d'esame **non è consentito**:
   - uscire dall'aula;
   - 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, ...);
   - 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.
4. È necessario consegnare (**anche in caso di ritiro**) tutti i fogli ricevuti, inclusi quelli per la brutta copia, i quali devono essere esplicitamente segnalati come tali (scrivendo "BRUTTA" su ciascuna delle loro facciate), nonchè il testo della prova d'esame.

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
```