Università del Piemonte Orientale

Dipartimento di Scienze e Innovazione Tecnologica

Esame di Algoritmi 1 – Sperimentazioni (VC)

10 luglio 2024

Testo d'Esame

Esercizio 1 (max 15 punti)

Implementare un algoritmo che, dato un albero binario di ricerca (BST) e una chiave k, restituisca:

- **la più grande chiave** del sottoalbero la cui radice è k;
- **NULL**, se la chiave non è presente nell'albero o se il BST è vuoto.

.

Per esempio, dati l'albero in Figura 1, e una chiave:

- $k = 6$ il risultato è 7;
- $k = 8$ il risultato è 14;
- $k = 13$ il risultato è 13;
- $k = 19$ il risultato è $NULL$.

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

* * *

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

```
void* upo_bst_subtree_max(const upo_bst_t tree, const void *key)
```

Parametri:

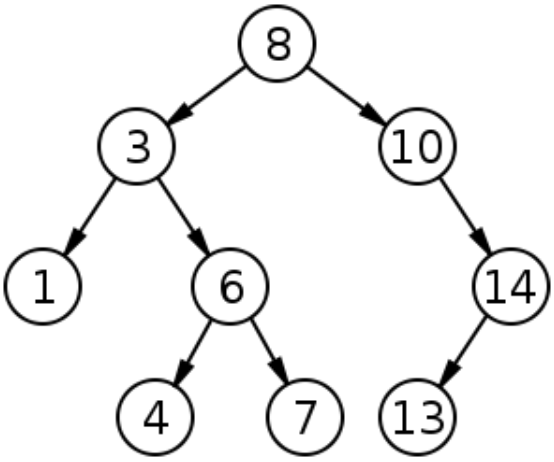

Figura 1: Un esempio di BST.

- `tree`: BST.
- `key`: la radice del sottoalbero di cui trovare la chiave maggiore.

Valore di ritorno:

- Se il BST non è vuoto e la chiave k è contenuta nell'albero: la chiave maggiore del sottoalbero la cui radice è k.
- Se il BST è vuoto o k non è contenuta nell'albero: $NULL$.

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 presenti e inserite nel file `exam.c`.

Il file `test/bst_subtree_max.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
```

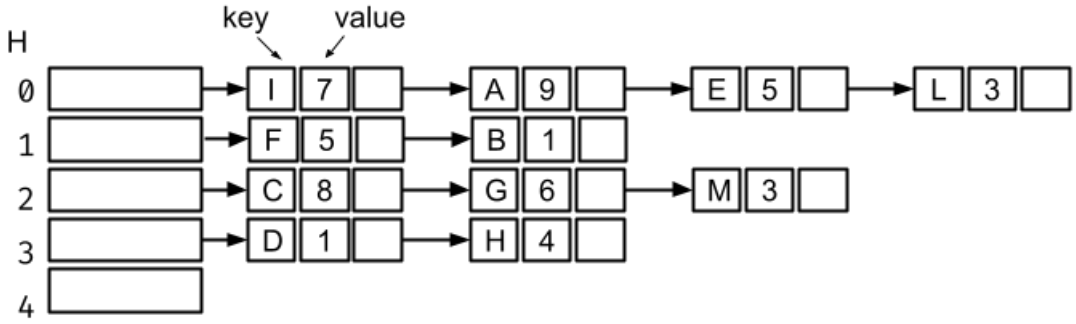

Figura 2: Un esempio di HT-SC.

Esercizio 2 (max 15 punti)

Implementare un algoritmo che, data una tabella hash H con gestione delle collisioni basata su concatenazioni separate (HT-SC - separate chaining) e una chiave k, calcoli la **percentuale di elementi** della tabella che collidono con k (Nota: k non è inclusa nel calcolo della percentuale). In particolare:

- se una chiave k è presente in H, l'algoritmo calcola la percentuale di elementi della tabella che collidono con k.
- se una chiave k non è contenuta in H, o se H è vuota l'algoritmo restituisce *-1*.

Data la tabella hash in Figura 2 e una chiave:

- $k = E$, il risultato è: 20%;
- $k = B$, il risultato è: 10%;
- $k = L$, il risultato è: 30%;
- $k = Z$, il risultato è: -1.

L'algoritmo implementato deve essere ottimo, nel senso che deve visitare le HT-SC una sola volta e non deve visitare parti delle HT-SC inutili ai fini dell'esercizio.

* * *

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

```
float upo_ht_sepchain_perc_collisions(const upo_ht_sepchain_t ht, const void *key)
```

Parametri:

- `ht`: Tabella Hash.
- `key`: chiave.

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`). 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 presenti e inserite nel file `exam.c`.

Il file `test/ht_sepchain_perc_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
```