Sellf Simple Database Challenge

In questo esercizio il candidato dovrà implementare un database chiave-valore residente in memoria, simile a Redis. Per semplicità, il programma non dovrà gestire connessioni multiple o comunicare via rete, ma riceverà invece comandi tramite *standard input (stdin)* e *standard output (stdout)*.

Linee guida

- Si raccomanda al candidato di usare un linguaggio ad alto livello, come Python, Ruby, Javascript/Node, Java, Go, Haskell o similari. Siamo più interessati alla pulizia del codice e alle buone performance *algoritmiche* piuttosto che al puro throughput.
- È preferibile ridurre al minimo le dipendenze esterne al progetto, rendendo la compilazione (se necessaria) del codice il più semplice possibile, e includere le istruzioni per l'esecuzione del codice direttamente da linea di comando
- Il programma dovrà rispettare quanto previsto dai comandi e dalle specifiche di performance dei paragrafi sottostanti

Comandi base

Il database sviluppato dovrà accettare i seguenti comandi:

- `SET key value` : imposta la chiave `key` con il valore `value`. Né chiavi né valori conterranno spazi.
- `GET key` : stampa il valore della chiave `key` o `NULL` se la chiave non è presente.
- `UNSET key` : rimuove la chiave `key`
- `EQUALTO value` : stampa il numero di chiavi che hanno valore settato a `value`. Se non sono presenti chiavi per il valore richiesto, stampa 0.
- `END` : esce dal programma. Il programma riceverà sempre questo comando come ultima istruzione.

I comandi verranno inseriti nel programma uno alla volta, un comando per linea. Qualsiasi output fornito dal programma **deve** terminare con un *carattere di nuova linea* (`\n`).

Di seguito alcuni esempi di sequenze di comandi:

input	output
SET a 10	
GET a	10
UNSET a	
GET a	NULL
END	

input	output
SET a 10	
SET b 10	
EQUALTO 10	2
EQUALTO 20	0
SET b 30	
EQUALTO 10	1
END	

Comandi transazioni

In aggiunta ai comandi di base visti sopra, il database dovrà supportare delle transazioni implementando i seguenti comandi:

- `BEGIN` : inizia una nuova transazione. Le transazioni possono essere innestate, quindi un'istruzione `BEGIN` può essere lanciata dentro una transazione esistente.
- `ROLLBACK` : Annulla tutti i comandi lanciati nella transazione attuale e chiude la transazione. Nel caso in cui nessuna transazione sia aperta, stampa `NO TRANSACTION` , altrimenti nulla.
- `COMMIT` : Chiude **tutte** le transazioni aperte applicando definitivamente i cambiamenti apportati. Nel caso in cui nessuna transazione sia aperta, stampa `NO TRANSACTION` , altrimenti nulla.

Qualsiasi comando eseguito fuori da una transazione deve essere committato immediatamente.

Di seguito alcuni esempi di sequenze di comandi:

input	output
BEGIN	
SET a 10	
GET a	10
BEGIN	
SET a 20	
GET a	20
ROLLBACK	
GET a	10
ROLLBACK	
GET a	NULL
END	

input	output
BEGIN	
SET a 30	
BEGIN	
SET a 40	
COMMIT	
GET a	40
ROLLBACK	NO TRANSACTION
COMMIT	NO TRANSACTION
END	

input	output
SET a 50	
BEGIN	
GET a	50
SET a 60	
BEGIN	
UNSET a	
GET a	NULL
ROLLBACK	
GET a	60
COMMIT	
GET a	60
END	

input	output
SET a 10	
BEGIN	
EQUALTO 10	1
BEGIN	
UNSET a	
EQUALTO 10	0
ROLLBACK	
EQUALTO 10	1
COMMIT	
END	

Requisiti di performance

- I comandi `BEGIN`, `GET`, `UNSET` e `EQUALTO` dovrebbero avere un ordine di complessità medio *O(log N)* o migliore, dove per *N* si indica il numero di dati salvati nel database. Inoltre, il tempo di runtime di questi comandi non dovrebbe dipendere dal numero di transazioni *T* inizializzate (è possibile avere $T >> N$).
- La *stragrande maggioranza* di transazioni aggiorneranno un piccolo numero di chiavi *K* ($K << N$). Di conseguenza, ogni transazione dovrebbe consumare al massimo un ordine *O(K)* aggiuntivo di memoria.

Valutazione

Valuteremo l'esercizio in base a:

- Funzionalità: l'implementazione soddisfa tutti i requisiti logici descritti? Risponde in maniera corretta anche in casi limite ed è sufficientemente testato?
- Performance algoritmica: l'implementazione soddisfa i requisiti richiesti sia in termini di tempo di runtime che di utilizzo di memoria?
- Code style: Quanto è leggibile il codice? Ha una struttura chiara e utile al problema posto? La soluzione è troppo complicata o *over-engineered*?