CSE 208 Online on Hashing

Scenario:

You're building a permission database where each `group_id` maps to users (`user_id`) with access rights (like `"read"`, `"write"`). Use a **two-layer hash table** with **open addressing for** both levels.

Structure:

- **Outer Table** (size `N`): Each element of this table contains a pointer/reference to another hash table on the second layer having size *N. So,* it stores `(group_id, pointer_to_inner_table).`
- **Inner Table** (size `N`): stores `(user_id, permission)`

Hashing:

- **Hash function for outer table:**
  Use the Hash1 function from your offline.

- **Hash function for Inner table:**
  Use the Hash2 function from your offline.

Operations:

`INSERT(group_id, user_id, permission)`

- Use `hash1` + linear probing to insert/find `group_id` in the outer table.
  Linear probing:
  Index = (hash1(group_id) + i) mod N
- Use `hash2` + quadratic probing to insert `(user_id, permission)` in the inner table.

Quadratic probing:

Index = (hash2(user_id) + i^2) mod N

`SEARCH(group_id, user_id)`

- Probe outer table for `group_id`.
- If found, probe the inner table for `user_id`.

- Print value if found; otherwise, report not found.

`SEARCH(group_id)`

- Return all `(user_id, permission)` in the inner table.
- If a group is not found, report it.

Input Format:

1. The first line contains two integers: N and Q
    - `N`: Size of both the outer and inner hash tables.
    - `Q`: Number of operations (commands) to process.
2. Each of the next `Q` lines contains one of the following operations:

Insert Operation

`INSERT <group_id> <user_id> <permission>`

- `group_id`: The key for the outer table.
- `user_id`: The key for the inner table under that group.
- `permission`: A string (e.g., `"read"`, `"write"`) representing access rights
- You don't need to provide any output for this operation.

Search with Two Keys

`SEARCH <group_id> <user_id>`

Searches for a specific `user_id` under a given `group_id` and prints its permission.

Search with One Key

`SEARCH <group_id>`

Prints all `(user_id, permission)` pairs for the given `group_id`.

DELETE

`DELETE <group_id> <user_id>`

Deletes and prints `(user_id, permission)` pair for the given `group_id`.

Output Format:

For each Search keyword, print the values if found. For each DELETE keyword, print the `(user_id, permission)` that is deleted.

Sample Input:

```
10 10
INSERT 101 1 read
INSERT 101 2 write
INSERT 111 2 read-write
SEARCH 101 1
SEARCH 101
DELETE 101 1
SEARCH 101
SEARCH 999
SEARCH 101 2
SEARCH 101 5
```

Sample Output:

```
read
(1, read), (2, write)
(1, read) deleted
(2, write)
Group not found
write
User not found in group 101
```