1NAME :- Manish Shashikant Jadhav
UID :- 2023301005.
BRANCH :- Comps -B. **BRANCH:** B.
EXPERIMENT 9: Implement Hashing using Quadratic Probing.

SUBJECT :- DS (DATA STRUCTURES).

CODE :-

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TABLE_SIZE 23

typedef struct KeyValue {
    char *key;
    char *value;
    bool isDeleted;
} KeyValue;

typedef struct {
    KeyValue **array;
    int size;
    float load_factor;
    int num_keys;
    int num_occupied_indices;
    int num_ops;
} HashTable;

KeyValue *createKeyValue(char *key, char *value) {
    KeyValue *newKeyValue = (KeyValue
*)malloc(sizeof(KeyValue));
    if (newKeyValue != NULL) {
        newKeyValue->key = strdup(key);
        newKeyValue->value = strdup(value);
        newKeyValue->isDeleted = false;
    }
    return newKeyValue;
}

HashTable *createHashTable() {
```

```
    HashTable *newTable = (HashTable
*)malloc(sizeof(HashTable));
    newTable->array = (KeyValue **)malloc(TABLE_SIZE *
sizeof(KeyValue *));
    for (int i = 0; i < TABLE_SIZE; i++)
        newTable->array[i] = NULL;

    newTable->size = TABLE_SIZE;
    newTable->load_factor = 0;
    newTable->num_keys = 0;
    newTable->num_occupied_indices = 0;
    newTable->num_ops = 0;
    return newTable;
}

int key_to_int(char *key) {
    int sum = 0;
    int index = 0;
    while (*key) {
        sum += (*key) * (index + 1);
        key++;
        index++;
    }
    return sum;
}

int hash_function(char *key, int size) {
    return key_to_int(key) % size;
}

int insert_key_value(HashTable *ht, char *key, char *value) {
    if (ht == NULL || key == NULL || value == NULL)
        return -1;

    int index = hash_function(key, ht->size);
    int original_index = index;
    int i = 1; // Quadratic Probing Counter

    while (ht->array[index] != NULL) {
```

```
        if (strcmp(ht->array[index]->key, key) == 0 && !ht-
>array[index]->isDeleted)
            return -1; // Similar Key
        index = (original_index + i * i) % ht->size;
        i++;

        // Table Traversed Completely or Not
        if (index == original_index)
            return -1; // Table Failed
    }

    // Insert the key-value pair
    ht->array[index] = createKeyValue(key, value);
    ht->num_keys++;
    ht->num_occupied_indices++;
    ht->num_ops++;
    return index;
}

char *search_key(HashTable *ht, char *key) {
    if (ht == NULL || key == NULL)
        return NULL;

    int index = hash_function(key, ht->size);
    int original_index = index;
    int i = 1; // Quadratic Probing Counter

    while (ht->array[index] != NULL) {
        if (strcmp(ht->array[index]->key, key) == 0 && !ht-
>array[index]->isDeleted) {
            return ht->array[index]->value;
        }
        index = (original_index + i * i) % ht->size;
        i++;

        // Table Traversed Completely or Not
        if (index == original_index)
            break;
    }
```

```
    return NULL; // Key not found
}

int delete_key(HashTable *ht, char *key) {
    if (ht == NULL || key == NULL)
        return -1;

    int index = hash_function(key, ht->size);
    int original_index = index;

    // Quadratic probing to handle collisions
    while (ht->array[index] != NULL) {
        if (strcmp(ht->array[index]->key, key) == 0 && !ht-
>array[index]->isDeleted) {
            // Mark it Deleted
            ht->array[index]->isDeleted = true;
            ht->num_keys--;
            ht->num_ops++;
            return index;
        }
        index = (index + 1) % ht->size;

        // Table Traversed Completely or Not
        if (index == original_index)
            break;
    }

    return -1; // Key not found
}

int get_load_factor(HashTable *ht) {
    if (ht == NULL || ht->size == 0)
        return -1;
    return (float)ht->num_keys / ht->size;
}

int get_avg_probes(HashTable *ht) {
    if (ht == NULL || ht->num_ops == 0)
        return -1;
    return ht->num_ops / ht->num_keys;
```

```
}

void display(HashTable *ht) {
    if (ht == NULL)
        return;
    printf("Hash Table:\n");
    printf("| %-10s | %-15s | %-15s |\n", "Index", "Key",
"Value");
    printf("|------------|-----------------|----------------
|\n");
    for (int i = 0; i < ht->size; i++) {
        printf("| %-10d |", i);
        if (ht->array[i] != NULL) {
            if (ht->array[i]->isDeleted) {
                printf(" %-15s | %-15s |", "(Deleted)",
"(Deleted)");
            } else {
                printf(" %-15s | %-15s |", ht->array[i]->key,
ht->array[i]->value);
            }
        } else {
            printf(" %-15s | %-15s |", "(Empty)", "(Empty)");
        }
        printf("\n");
    }
}

int main() {
    HashTable *ht = createHashTable();

    // Insert key-value pairs
    insert_key_value(ht, "first name", "Manish");
    insert_key_value(ht, "last name", "Jadhav");
    insert_key_value(ht, "uid", "2023301005");
    insert_key_value(ht, "sport", "Cricket");
    insert_key_value(ht, "food", "Burger");
    insert_key_value(ht, "holiday", "Maldives");
    insert_key_value(ht, "role_model", "Chhatrapati Shivaji
Maharaj");
    insert_key_value(ht, "subject", "Python");
```

```
    insert_key_value(ht, "song", "Aarambh");
    insert_key_value(ht, "movie", "Farjand");
    insert_key_value(ht, "colour", "Orange");
    insert_key_value(ht, "book", "The Hidden Hindu");

    // Test search and delete operations
    char *search_result = search_key(ht, "sport");
    if (search_result != NULL) {
        printf("> Search Result for 'sport': %s\n",
search_result);
    } else {
        printf("> Key 'sport' not found\n");
    }

    int delete_result = delete_key(ht, "holiday");
    if (delete_result != -1) {
        printf("> Deleted key 'holiday' at index [%d]\n",
delete_result);
    } else {
        printf("> Key 'holiday' not found for deletion\n");
    }

    // Display the hash table
    display(ht);
    return 0;
}
```

Output:

```
pi.3gq' '--stderr=Microsoft-MIEngine-Error-l5w0fmax.fzy' '--pid=Microsoft-MIEngine-Pid-q0m4yanc.2tp' '
:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin\gdb.exe' '--interpreter=
> Search Result for 'sport': Cricket
> Deleted key 'holiday' at index [9]
Hash Table:
| Index      | Key             | Value           |
|------------|-----------------|-----------------|
| 0          | movie           | Farjand         |
| 1          | book            | The Hidden Hindu |
| 2          | song            | Aarambh         |
| 3          | (Empty)         | (Empty)         |
| 4          | (Empty)         | (Empty)         |
| 5          | (Empty)         | (Empty)         |
| 6          | uid             | 2023301005      |
| 7          | sport           | Cricket         |
| 8          | (Empty)         | (Empty)         |
| 9          | (Deleted)       | (Deleted)       |
| 10         | role_model      | Chhatrapati Shivaji Maharaj |
| 11         | subject         | Python          |
| 12         | colour          | Orange          |
| 13         | (Empty)         | (Empty)         |
| 14         | (Empty)         | (Empty)         |
| 15         | (Empty)         | (Empty)         |
| 16         | (Empty)         | (Empty)         |
| 17         | (Empty)         | (Empty)         |
| 18         | first name      | Manish          |
| 19         | (Empty)         | (Empty)         |
| 20         | (Empty)         | (Empty)         |
| 21         | last name       | Jadhav          |
| 22         | food            | Burger          |
PS D:\Manish\DS SPIT>
```

Algorithm:

1. KeyValue Structure:

- Contains a key (string), value (string), and a boolean flag to indicate if the entry has been deleted.

2. HashTable Structure:

- An array of pointers to KeyValue structures.

- Size of the array (`size`), load factor, number of keys (`num_keys`), number of occupied indices (`num_occupied_indices`), and number of operations (`num_ops`) are tracked.

Functions:

1. createKeyValue:

- Dynamically allocates memory for a new KeyValue structure and initializes its fields.

2. createHashTable:

- Dynamically allocates memory for a new HashTable structure and initializes its fields.

- Initializes the array of KeyValue pointers to `NULL`.

3. key_to_int:

- Converts a string key to an integer value.

4. hash_function:

- Uses the key_to_int function to calculate the hash index for a given key.

5. insert_key_value:

- Inserts a key-value pair into the hash table using quadratic probing for collision resolution.

6. search_key:

- Searches for a key in the hash table and returns its corresponding value.

7. delete_key:

- Deletes a key from the hash table and marks it as deleted.

8. get_load_factor:

- Calculates and returns the load factor of the hash table.

9. get_avg_probes:

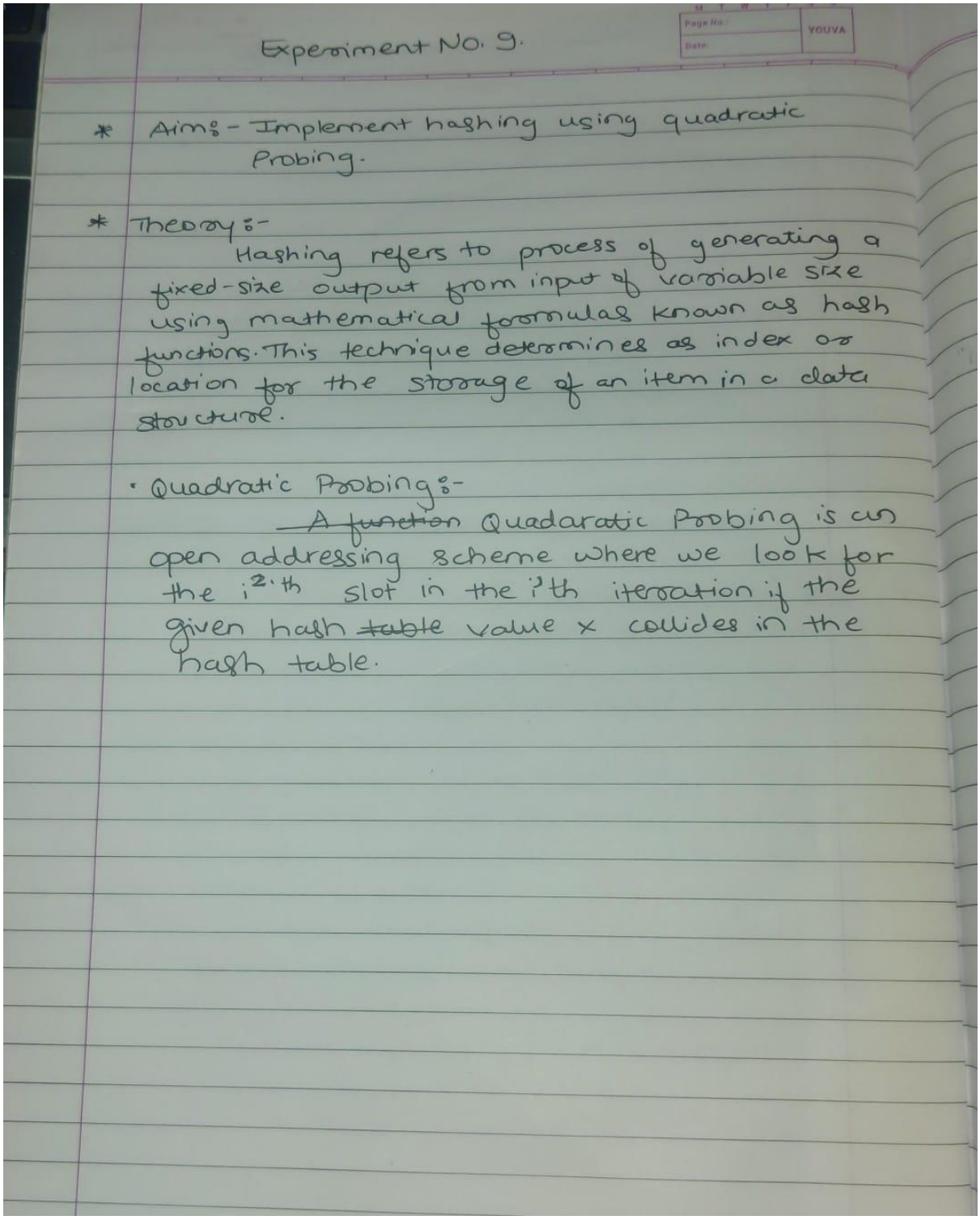
- Calculates and returns the average number of probes per operation.

10. display:

- Displays the contents of the hash table

11. Main Function:

- Creates a hash table using `createHashTable`.
- Inserts several key-value pairs into the hash table using `insert_key_value`.
- Searches for the value of a specific key using `search_key`.
- Deletes a specific key using `delete_key`.
- Displays the contents of the hash table using `display`.



Experiment No. 9.

Page No.:
Date:
YOUVA

* Aim:- Implement hashing using quadratic Probing.

* Theory:-
Hashing refers to process of generating a fixed-size output from input of variable size using mathematical formulas known as hash functions. This technique determines an index or location for the storage of an item in a data structure.

• Quadratic Probing:-
~~A function~~ Quadaratic Probing is an open addressing scheme where we look for the i^{2}'th slot in the i'th iteration if the given hash ~~table~~ value x collides in the hash table.

Conclusion:

Hence, by completing this experiment I came to know about implementing Hashing using Quadratic Probing.