DEPARTMENT OF INFORMATION TECHNOLOGY

Academic Year: 2023 - 24

COURSE CODE: DJS22ITL302 **CLASS: S. Y. B. Tech. Sem III(I1)**

COURSE NAME: Data Structures Lab **SAP ID :60003220045**
Name : Anish Sharma **DATE: 4/12/2023**

EXPERIMENT NO. 10

CO/LO:

Implement Hashing techniques and collision resolution algorithms.

Objective:

Write a program to Implementation of various hashing techniques with different collision resolution algorithms

Code :

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
#define PRIME 7
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
#define PRIME 7
struct Node {
    int data;
    struct Node *next;
};
// Function prototypes
void SortedInsert(struct Node **H, int x);
struct Node *Search(struct Node *p, int key);
int hash(int key);
void Insert(struct Node *H[], int key);
int LinearProbe(int H[], int key);
void InsertLinear(int H[], int key);
```

```
int QuadraticProbe(int H[], int key);
void InsertQuadratic(int H[], int key);
int PrimeHash(int key);
int DoubleHash(int H[], int key);
void InsertDoubleHash(int H[], int key);
void Print(int vec[], int n, const char *s);
void PrintHashTable(struct Node *H[], int n);
int main() {
    struct Node *HT_SC[SIZE];
    struct Node *temp;
    int HT_LP[SIZE] = {0};
    int HT_QP[SIZE] = {0};
    int HT_DH[SIZE] = {0};
    int choice, element, key, result;
    for (int i = 0; i < SIZE; i++)
        HT_SC[i] = NULL;
    do {
        printf("\nMenu:\n");
        printf("1. Separate Chaining\n");
        printf("2. Linear Probing\n");
        printf("3. Quadratic Probing\n");
        printf("4. Double Hashing\n");
        printf("5. Print Hash Tables\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter an element to insert: ");
                scanf("%d", &element);
```

```
        Insert(HT_SC, element);
        printf("Enter an element to search: ");
        scanf("%d", &key);
        temp = Search(HT_SC[hash(key)], key);
        if (temp != NULL) {
            printf("Element found: %d\n", temp->data);
        } else {
            printf("Element not found.\n");
        }
        break;
    case 2:
        printf("Enter an element to insert: ");
        scanf("%d", &element);
        InsertLinear(HT_LP, element);
        printf("Enter an element to search: ");
        scanf("%d", &key);
        result = LinearProbe(HT_LP, key);
        if (result != -1) {
            printf("Key found at: %d\n", result);
        } else {
            printf("Key not found.\n");
        }
        break;
    case 3:
        printf("Enter an element to insert: ");
        scanf("%d", &element);
        InsertQuadratic(HT_QP, element);
        printf("Enter an element to search: ");
        scanf("%d", &key);
        result = QuadraticProbe(HT_QP, key);
```

```
        if (result != -1) {
            printf("Key found at: %d\n", result);
        } else {
            printf("Key not found.\n");
        }
        break;
    case 4:
        printf("Enter an element to insert: ");
        scanf("%d", &element);
        InsertDoubleHash(HT_DH, element);
        printf("Enter an element to search: ");
        scanf("%d", &key);
        result = DoubleHash(HT_DH, key);
        if (result != -1) {
            printf("Key found at: %d\n", result);
        } else {
            printf("Key not found.\n");
        }
        break;
    case 5:
        PrintHashTable(HT_SC, SIZE);
        Print(HT_LP, SIZE, "HT Linear Probing");
        Print(HT_QP, SIZE, "HT Quadratic Probing");
        Print(HT_DH, SIZE, "HT Double Hashing");
        break;
    case 6:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice. Please enter a valid option.\n");
```

```
            break;
        }
    } while (choice != 6);
    return 0;
}
void SortedInsert(struct Node **H, int x) {
    struct Node *t, *q = NULL, *p = *H;
    t = (struct Node *)malloc(sizeof(struct Node));
    t->data = x;
    t->next = NULL;
    if (*H == NULL)
        *H = t;
    else {
        while (p && p->data < x) {
            q = p;
            p = p->next;
        }
        if (p == *H) {
            t->next = *H;
            *H = t;
        } else {
            t->next = q->next;
            q->next = t;
        }
    }
}
struct Node *Search(struct Node *p, int key) {
    while (p != NULL) {
        if (key == p->data) {
            return p;
```

```
        }
        p = p->next;
    }
    return NULL;
}
int hash(int key) {
    return key % SIZE;
}
void Insert(struct Node *H[], int key) {
    int index = hash(key);
    SortedInsert(&H[index], key);
}
int LinearProbe(int H[], int key) {
    int idx = hash(key);
    int i = 0;
    while (H[(idx + i) % SIZE] != 0) {
        i++;
    }
    return (idx + i) % SIZE;
}

void InsertLinear(int H[], int key) {
    int idx = hash(key);
    if (H[idx] != 0) {
        idx = LinearProbe(H, key);
    }
    H[idx] = key;
}
int QuadraticProbe(int H[], int key) {
    int idx = hash(key);
```

```
    int i = 0;
    while (H[(idx + i * i) % SIZE] != 0) {
        i++;
    }
    return (idx + i * i) % SIZE;
}
void InsertQuadratic(int H[], int key) {
    in idx = hash(key);
    if (H[idx] != 0) {
        idx = QuadraticProbe(H, key);
    }
    H[idx] = key;
}
int PrimeHash(int key) {
    return PRIME - (key % PRIME);
}
int DoubleHash(int H[], int key) {
    int idx = hash(key);
    int i = 0;
    while (H[(hash(idx) + i * PrimeHash(idx)) % SIZE] != 0) {
        i++;
    }
    return (idx + i * PrimeHash(idx)) % SIZE;
}
void InsertDoubleHash(int H[], int key) {
    int idx = hash(key);
    if (H[idx] != 0) {
        idx = DoubleHash(H, key);
    }
    H[idx] = key;
```

```
}
void Print(int vec[], int n, const char *s) {
    printf("%s: [", s);
    for (int i = 0; i < n; i++) {
        printf("%d", vec[i]);
        if (i < n - 1) {
            printf(", ");}}
    printf("]\n");
}

void PrintHashTable(struct Node *H[], int n) {
    printf("Hash Table:\n");
    for (int i = 0; i < n; i++) {
        printf("HT[%d]: ", i);
        struct Node *temp = H[i];
        while (temp != NULL) {
            printf("%d", temp->data);
            temp = temp->next;
            if (temp != NULL) {
                printf(" -> ");}}
        printf("\n");}}
```

Output :

```
Menu:
1. Separate Chaining
2. Linear Probing
3. Quadratic Probing
4. Double Hashing
5. Print Hash Tables
6. Exit
Enter your choice: 1
Enter an element to insert: 3
Enter an element to search: 9
Element not found.

Menu:
1. Separate Chaining
2. Linear Probing
3. Quadratic Probing
4. Double Hashing
5. Print Hash Tables
6. Exit
Enter your choice: 4
Enter an element to insert: 2
Enter an element to search: 3
Key found at: 6

Menu:
1. Separate Chaining
2. Linear Probing
3. Quadratic Probing
4. Double Hashing
5. Print Hash Tables
6. Exit
Enter your choice: 5
```

```
Hash Table:
HT[0]:
HT[1]:
HT[2]:
HT[3]:
HT[4]:
HT[5]:
HT[6]: 36
HT[7]:
HT[8]:
HT[9]:
HT Linear Probing: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
HT Quadratic Probing: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
HT Double Hashing: [0, 0, 0, 0, 0, 25, 0, 0, 0, 0]
```