Kartheeka.Repalle

192372289

CSE-AI

5/8/2024

Hashing:

In data structures,

• Hashing is a well-known technique to search any particular element among several elements.

• It minimizes the number of comparisons while performing the search.

Advantage

- Unlike other searching techniques,

• Hashing is extremely efficient.

• The time taken by it to perform the search does not depend upon the total number of elements.

• It completes the search operation with constant time complexity O(1).

Sourse code:

```
#include <stdio.h>
#include <stdlib.h>

#define TABLE_SIZE 10

typedef struct Node {
    int data;
    struct Node* next;
} Node;
```

```
typedef struct HashTable {
    Node* heads[TABLE_SIZE];
} HashTable;

int hash_function(int key) {
    return key % TABLE_SIZE;
}

void insert(HashTable* table, int key) {
    int index = hash_function(key);
    Node* new_node = (Node*)malloc(sizeof(Node));
    new_node->data = key;
    new_node->next = table->heads[index];
    table->heads[index] = new_node;
}

void print_table(HashTable* table) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        Node* current = table->heads[i];
        printf("Index %d: ", i);
        while (current != NULL) {
            printf("%d -> ", current->data);
            current = current->next;
        }
        printf("NULL\n");
    }
}
```

```
int main() {
    HashTable table;
    for (int i = 0; i < TABLE_SIZE; i++) {
        table.heads[i] = NULL;
    }

    insert(&table, 10);
    insert(&table, 12);
    insert(&table, 23);
    insert(&table, 42);
    insert(&table, 53);
    insert(&table, 62);
    insert(&table, 74);
    insert(&table, 85);
    insert(&table, 96);
    insert(&table, 105);
    insert(&table, 116);

    print_table(&table);

    return 0;
}
```

Output:

```
Index 0: NULL
Index 1: 116 -> NULL
Index 2: 12 -> NULL
Index 3: NULL
Index 4: 74 -> NULL
Index 5: 105 -> NULL
Index 6: 62 -> NULL
Index 7: NULL
Index 8: 42 -> NULL
Index 9: 23 -> 85 -> 96 -> 53 -> 10 -> NULL
```

LINEAR PROBING:

```
#include <stdio.h>
#include <stdlib.h>

#define TABLE_SIZE 10

typedef struct HashTable {
    int keys[TABLE_SIZE];
    int values[TABLE_SIZE];
} HashTable;

int hash_function(int key) {
    return key % TABLE_SIZE;
```

```
}

void insert(HashTable* table, int key) {
    int index = hash_function(key);
    while (table->keys[index] != 0) {
        if (table->keys[index] == key) {
            printf("Key already exists\n");
            return;
        }
        index = (index + 1) % TABLE_SIZE;
    }
    table->keys[index] = key;
}

void print_table(HashTable* table) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (table->keys[i] != 0) {
            printf("Index %d: Key = %d\n", i, table->keys[i]);
        } else {
            printf("Index %d: Empty\n", i);
        }
    }
}

int main() {
    HashTable table;
    for (int i = 0; i < TABLE_SIZE; i++) {
```

```
        table.keys[i] = 0;
    }

    insert(&table, 79);
    insert(&table, 17);
    insert(&table, 47);
    insert(&table, 58);
    insert(&table, 69);
    insert(&table, 32);
    insert(&table, 97);

    print_table(&table);

    return 0;
}
```

OUTPUT:

```
Index 0: Empty
Index 1: Key = 97
Index 2: Key = 79
Index 3: Key = 17
Index 4: Key = 47
Index 5: Key = 58
Index 6: Key = 69
Index 7: Key = 32
Index 8: Empty
Index 9: Empty
```