Data Structures and Algorithms

Pierre Collet/ Manal ElZant-Karray

Practical work n°6 : Hash Table

Exercise 1: During the lecture session we implemented a Hash table using a Singly Linked List. Now, Implement a Hash table with Doubly Linked List using chaining to resolve collisions. Use division hash function.

N.B: Implementing hash table using Doubly Linked List is similar to implementing hash table using Singly Linked List. The difference is that every node of Linked List has address of both, the next and the previous node. This will help removing data from this Linked List faster than Singly Linked List.

Exercise 2: Draw the 11-entry hash that results from using the hash function h(i) = (2i+5) mod11 to hash keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5.

1. Assume collisions are handled by chaining.
2. Assume collisions are handled by linear probing.
3. Assume collisions are handled with double hashing, with the secondary hash function g(k) = 7 - (k mod 7).

Exercise 3: Complete the below program to demonstrate Linear Probing in hash table using division method:

The *insertion* function to insert student's information (id (which is the key), name and grade);

The *deletion* function to delete a student's information (using id);

The *search* function based on id;

And the *display* function of all students information.

```
#include<stdio.h>
#include<stdlib.h>
#define LIMIT 30
enum record_status {EMPTY, DELETED, OCCUPIED};
struct Student {
    int student_id, student_grade;
    char student_name[30];
};
struct Record {
    struct Student info;
    enum record_status status;
};
int hash_function(int key) {
    return (key % LIMIT);
}
int search_records(int key, struct Record hash_table[])
void insert_records(struct Student studrec, struct Record hash_table[])
void display_records(struct Record hash_table[])
void delete_records(int key, struct Record hash_table[])
int main() {
    int count, key, option;
    struct Record hash_table[LIMIT];
```

```
    struct Student studrec;
    for(count = 0; count <= LIMIT - 1; count++)
        hash_table[count].status = EMPTY;
    while(1) {
        printf("1. Insert a Record\n");
        printf("2. Delete a Record\n");
        printf("3. Search a Record\n");
        printf("4. Display All Records\n");
        printf("5. Exit\n");
        printf("Enter Your Option:\t");
        scanf("%d", &option);
        switch(option) {
            case 1: printf("\nEnter Student ID:\t");
                    scanf("%d", &studrec.student_id);
                    printf("Enter Student Name:\t");
                    scanf("%s", studrec.student_name);
                    printf("Enter Student Grade:\t");
                    scanf("%d", &studrec.student_grade);
                    insert_records(studrec, hash_table);
                    break;
            case 2: printf("\nEnter the Key to Delete:\t");
                    scanf("%d", &key);
                    delete_records(key, hash_table);
                    break;
            case 3: printf("\nEnter the Key to Search:\t");
                    scanf("%d", &key);
                    count = search_records(key, hash_table);
                    if(count == -1) printf("\nRecord Not Found\n");
                    else
                        printf("\nRecord Found at Index Position:\t%d\n", count);
                    break;
            case 4: display_records(hash_table);
                    break;
            case 5: exit(1);
        }
    }
    return 0;
}
```