**DSA Experiment - 6**

HARSHIT SARASWAT | B-42 | 500097101

**Question 1 :** Implement a Hash table using arrays (dynamic array) and Linear probing as a Collision avoidance strategy. Perform Insert, Delete and Search operations on the Hash table using the above Hash function.

**Code :**

#include <stdio.h>

#include <stdlib.h>

int \* hash\_array;

int count = 0;

int size;

int hash\_function(int hash\_key);

int add\_till\_death(int input);

int main()

{

printf("\t\t\t\tLinear Probing DSA Experiment - 6");

//printf("Enter Size of Hash Table Array : ");

//scanf("%d",&size);

size = 10;

hash\_array = (int \*)malloc(size\*sizeof(int));

for(int i = 0;i < size;i++){

hash\_array[i] = NULL;

}

int choice = 0;

while (1 == 1){

printf("\n1 - To Enter Sap ID\n2 - To Delete SAP\_ID\n3 - Find Hash Key\n4 - Quit\nChoice : ");

scanf("%d",&choice);

switch(choice){

case 1 : insert\_key\_to\_table();break;

case 2 : delete\_key();break;

case 3 : search\_key();break;

case 4 : exit(0);

}

printf("\n");

}

return 0;

}

int hash\_function(int hash\_key){ //RETURN THE ARRAY INDEX FOR THE GIVEN KEY - IN THIS CASE SAP ID

hash\_key = hash\_key - 5000000;

return add\_till\_death(hash\_key);

}

void insert\_key\_to\_table(){ //FRONT END PRETTY FUNCTION

int data = 0;

printf("Enter SAP\_ID To insert : ");

scanf("%d",&data);

insert\_to\_array\_hash\_table(data,hash\_function(data));

}

int insert\_to\_array\_hash\_table(int data,int key){ //TECHNICAL FUNCTION BACKEND

if (hash\_array[key] == NULL){

hash\_array[key] = data;

return;

}

//LINEAR PROBING

int i = 0;

for(i = (key + 1) % size; hash\_array[i]!=NULL && i!=key ; i++,i = i % size); //KEEP GOING TO INDEX THAT IS NULL EVEN AFTER END OF ARRAY

if (i == key){

printf("\nNo Available Space Array is FULL...");

return;

}

hash\_array[i] = data;

}

int add\_till\_death(int input){

int total = 0;

while (input != 0){

total = total + input % 10;

input = input/10;

}

if (total > 9)

total = add\_till\_death(total);

return total;

}

void search\_key(){

int data;

int key\_index;

printf("Enter Entry to Find : ");

scanf("%d", &data);

key\_index = hash\_function(data);

if (hash\_array[key\_index] == data){ //TO MAKE THE FOR LOOP CONDITION USEABLE

printf("\n%d is Present at Index : %d",data,key\_index);

return;

}

int i = key\_index+1;

for (i ; hash\_array[i] != data && i!=key\_index ; i++,i = i % size);

if (i == key\_index){

printf("\nKey not in Table");

return;

}

printf("\n%d is Present at Index : %d",data,i);

}

void delete\_key(){

int data;

printf("\nEnter Key to Delete : ");

scanf("%d",&data);

int key\_index = hash\_function(data);

int i = key\_index + 1;

if (hash\_array[key\_index] == data){

printf("\nDeleted Key : %d",hash\_array[key\_index]);

hash\_array[key\_index] = NULL;

return;

}

for (i ; hash\_array[i] != data && i!= key\_index; i++,i = i % size);

if (i == key\_index){

printf("\nKey not in Table\n");

return;

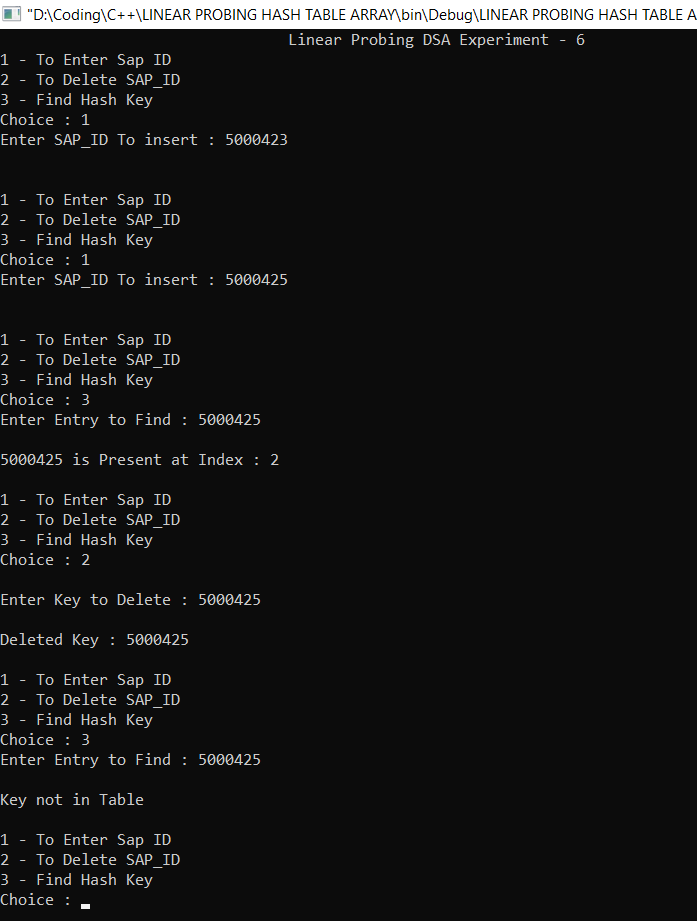
}

printf("\nDeleted Key : %d",hash\_array[i]);

hash\_array[i] = NULL;

}

Output :



**Question 2 :** Implement a Hash table using arrays (dynamic array) and Linear probing as a Collision avoidance strategy. Perform Insert, Delete and Search operations on the Hash table using the above Hash function. Compute Load Factor (LF). When LF>50% apply rehashing (with a new dynamic array size & suitable hash function).

**Code :**

#include <stdio.h>

#include <stdlib.h>

int \* hash\_array;

int count = 0;

int size;

int hash\_function(int hash\_key);

int add\_till\_death(int input);

int main(){

printf("\t\t\t\tLinear Probing DSA Experiment - 6");

//printf("Enter Size of Hash Table Array : ");

//scanf("%d",&size);

size = 10;

hash\_array = (int \*)malloc(size\*sizeof(int));

for(int i = 0;i < size;i++){

hash\_array[i] = NULL;

}

int choice = 0;

while (1 == 1){

printf("\n1 - To Enter Sap ID\n2 - To Delete SAP\_ID\n3 - Find Hash Key\n4 - Quit\nChoice : ");

scanf("%d",&choice);

switch(choice){

case 1 : insert\_key\_to\_table(hash\_array);rehashing();break;

case 2 : delete\_key();break;

case 3 : search\_key();break;

case 4 : exit(0);

}

printf("\n");

}

return 0;

}

void rehashing(){

float load\_factor = (float)count/size;

if (load\_factor < 0.5){

printf("No Need For Rehashing Load Factor = %.2f\n",load\_factor);

return;

}

printf("LOAD FACTOR OVER 0.5... REHASHING...\n");

count = 0;

int \* new\_hash\_array = (int \*)malloc((size\*2)\*sizeof(int));

int old\_size = size;

size = size\*2;

for (int i = 0; i < size\*2; i++){

new\_hash\_array[i] = NULL;

}

for (int i = 0 ; i < old\_size; i++){

if (hash\_array[i] == NULL)

continue;

int data = hash\_array[i];

insert\_to\_array\_hash\_table(data,hash\_function(data),new\_hash\_array);

}

hash\_array = new\_hash\_array;

printf("\nNew Hash Array\n");

for (int i = 0; i < size;i++){

printf("| %d |",hash\_array[i]);

}

}

int hash\_function(int hash\_key){ //RETURN THE ARRAY INDEX FOR THE GIVEN KEY - IN THIS CASE SAP ID

hash\_key = hash\_key - 5000000;

return add\_till\_death(hash\_key);

}

void insert\_key\_to\_table(int \* hash\_array){ //FRONT END PRETTY FUNCTION

int data = 0;

printf("Enter SAP\_ID To insert : ");

scanf("%d",&data);

insert\_to\_array\_hash\_table(data,hash\_function(data),hash\_array);

}

int insert\_to\_array\_hash\_table(int data,int key,int \* hash\_array){ //TECHNICAL FUNCTION BACKEND

if (hash\_array[key] == NULL){

count++;

hash\_array[key] = data;

return;

}

//LINEAR PROBING

int i = 0;

for(i = (key + 1) % size; hash\_array[i]!=NULL && i!=key ; i++,i = i % size); //KEEP GOING TO INDEX THAT IS NULL EVEN AFTER END OF ARRAY

if (i == key){

printf("\nNo Available Space Array is FULL...");

return;

}

count++;

hash\_array[i] = data;

}

int add\_till\_death(int input){

int total = 0;

while (input != 0){

total = total + input % 10;

input = input/10;

}

if (total > 9)

total = add\_till\_death(total);

return total;

}

void search\_key(){

int data;

int key\_index;

printf("Enter Entry to Find : ");

scanf("%d", &data);

key\_index = hash\_function(data);

if (hash\_array[key\_index] == data){ //TO MAKE THE FOR LOOP CONDITION USEABLE

printf("\n%d is Present at Index : %d",data,key\_index);

return;

}

int i = key\_index+1;

for (i ; hash\_array[i] != data && i!=key\_index ; i++,i = i % size);

if (i == key\_index){

printf("\nKey not in Table");

return;

}

printf("\n%d is Present at Index : %d",data,i);

}

void delete\_key(){

int data;

printf("\nEnter Key to Delete : ");

scanf("%d",&data);

int key\_index = hash\_function(data);

int i = key\_index + 1;

if (hash\_array[key\_index] == data){

printf("\nDeleted Key : %d",hash\_array[key\_index]);

hash\_array[key\_index] = NULL;

return;

}

for (i ; hash\_array[i] != data && i!= key\_index; i++,i = i % size);

if (i == key\_index){

printf("\nKey not in Table\n");

return;

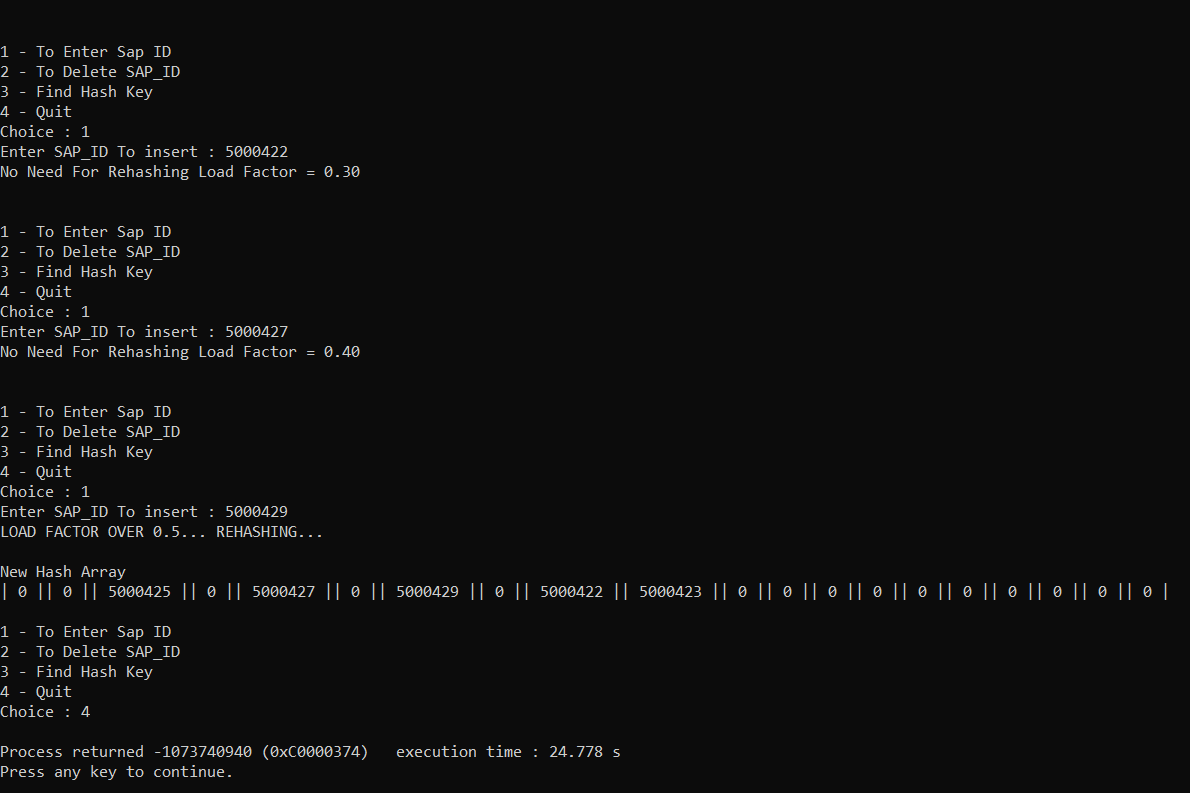
}

printf("\nDeleted Key : %d",hash\_array[i]);

hash\_array[i] = NULL;

}

**Output :**



**Question 3 :** Implement a Hash table using an array of Linked Lists with a Separate Chaining Collision avoidance strategy. Perform Insert, Delete and Search operations on the hash table using a suitable Hash function.

**Code :**

#include <stdio.h>

#include <stdlib.h>

struct node {

int Sap\_id;

struct node \* next;

};

struct node \* ptr;

struct node \* new\_sap;

struct node \* array\_of\_linked\_list [10];

int sap\_hash\_function(int sap\_id);

int add\_of\_all\_digits(int input);

int main(){

for (int i = 0; i < 10; i++){ //INITIALIZE ALL ARRAY POINTERS TO NULL

array\_of\_linked\_list[i] = NULL;

}

int repeat = 1;

while(repeat == 1){

printf("1 - Add a Sap to The Hash Table\n2 - Print The Hash Table\n3 - Quit\nChoice : ");

scanf("%d",&repeat);

switch(repeat){

case 1 : take\_sap\_and\_place\_position();

break;

case 2 : print\_hash\_table();

break;

case 3 : return 0;

}

repeat = 1;

}

return 0;

}

//SAP TAKER

int take\_sap\_and\_place\_position(){

int input\_sap\_id = 0;

int hashed\_index = 0;

printf("Enter Your SAP ID : ");

scanf("%d",&input\_sap\_id);

hashed\_index = sap\_hash\_function(input\_sap\_id);

place\_in\_array(hashed\_index,input\_sap\_id);

}

void print\_hash\_table(){

for (int hash\_key = 0;hash\_key < 10; hash\_key++){

ptr = array\_of\_linked\_list[hash\_key];

printf("HASH KEY : %d",hash\_key);

for (ptr; ptr!=NULL; ptr = ptr->next)

printf(" SAP : %d ",ptr->Sap\_id);

printf("\n");

}

}

void place\_in\_array(int hash\_key, int sap\_id){

if (array\_of\_linked\_list[hash\_key] != NULL){ //INSERT AT START IF NOT NULL

new\_sap = (struct node \*)malloc(sizeof(struct node));

new\_sap->next = array\_of\_linked\_list[hash\_key];

new\_sap->Sap\_id = sap\_id;

array\_of\_linked\_list[hash\_key] = new\_sap;

return;

}

new\_sap = (struct node \*)malloc(sizeof(struct node));

new\_sap->next = NULL;

new\_sap->Sap\_id = sap\_id;

array\_of\_linked\_list[hash\_key] = new\_sap;

}

//THE HASH function

int sap\_hash\_function(int sap\_id){

int hashed\_index = 0;

int temp\_sap = sap\_id;

temp\_sap = sap\_id - 5000000;

hashed\_index = add\_of\_all\_digits(temp\_sap);

while (hashed\_index > 9){

hashed\_index = add\_of\_all\_digits(hashed\_index);

}

return hashed\_index;

}

int add\_of\_all\_digits(int input){

int added = 0;

while(input != 0){

added = added + input%10;

input = input/10;

}

return added;

}

**Output :**

