//CLOSED ADDRESSING

AIM: To implement Hash data structure using Separate chaining as a collision resolution technique.

ALGORITHM :

STEP 1:- Declare an array of a linked list with the hash table size.

STEP 2:- Initialize the table with the size and the cells.

STEP 3:- Implement linear probing, quadratic probing, and double hashing functions.

STEP 4:- Implement rehashing to increase table size and reinsert entries, once half of the table is filled.

STEP 5:- Insert the keys entered by user, using anyone of the above collision resolution techniques, as the desire of user.

STEP 6:- Display the final hash table.

PROGRAM

#include<stdio.h>

#include<stdlib.h>

enum kind\_of\_entry {empty,delete,legitimate};

typedef struct node{

int data;

enum kind\_of\_entry info;

}\*cells;

typedef struct table{

int t\_size;

cells \*list;

}H\_table;

int nextprime(int num)

{ int f;

if( num==1)

{

return 2;

}

while(num)

{

for(int i=2;i<num;i++)

{ f=1;

if(num%i==0)

{f=0;

break;}

}

if(f)

return num;

num ++;

}

}

int isprime(int num)

{

if (num==2)

{

return 1;

}

for(int i=2;i<num;i++)

{

if(num%i==0)

{

return 0;

}

}

return 1;

}

H\_table\* initialize\_Table(int size)

{

size = nextprime(size);

H\_table \*H;

H= (H\_table\*)malloc(sizeof(H\_table));

if(H!=NULL)

{

H->t\_size =size;

H->list = ( cells\*)malloc(sizeof( cells)\*H->t\_size);

if(H->list!=NULL)

{

for(int i=0;i<H->t\_size;i++)

{

H->list[i]=(struct node\*)malloc(sizeof(struct node));

if(H->list[i]==NULL)

{

printf("Fatal Error\n");

return NULL;

}

H->list[i]->info = empty;

}

}

return H;

}

return NULL;

}

int hash(int key,int h\_size)

{

return key%h\_size;

}

int linear\_index(int key,H\_table \*H)

{

int index;

index = hash(key,H->t\_size);

while(H->list[index]->info==legitimate && H->list[index]->data!=key)

index = (index+1)%H->t\_size;

return index;

}

void linear\_probing(int key,H\_table \*H)

{

int i = linear\_index(key,H);

if(H->list[i]->info==empty)

{

H->list[i]->data = key;

H->list[i]->info = legitimate;

}

}

int quadratic\_index(int key , H\_table \*h)

{

int index;

int collision\_num =0 ;

index = hash(key,h->t\_size);

while(h->list[index]->info==legitimate && ( h->list[index]->data!=key ))

{

{

index = (2\*(++collision\_num)-1)+index %h->t\_size;}

}

return index;

}

void quadratic\_probing(int key,H\_table \*H)

{

int i = quadratic\_index(key , H);

if(H->list[i]->info==empty)

{

H->list[i]->data=key;

H->list[i]->info = legitimate;

}

}

int hash2(int key,int size)

{

int R = size-1;

while(!isprime(R))

{

R--;

}

return R-(key%R);

}

int find\_double(int key, H\_table \*h)

{

int pos=hash(key,h->t\_size);

int step=hash2(key,h->t\_size);

while(h->list[pos]->data!=key && h->list[pos]->info!=empty)

{

pos=(pos+step)%h->t\_size;

}

return pos;

}

void double\_hashing(int key,H\_table \*H)

{

int index = find\_double(key,H);

if(H->list[index]->info==empty)

{

H->list[index]->data = key;

H->list[index]->info=legitimate;

}

}

H\_table\* rehashing(H\_table \*H)

{

cells \*oldlist;

int i,old\_size;

old\_size = H->t\_size;

oldlist = H->list;

H = initialize\_Table(2\*H->t\_size);

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

{

if(oldlist[i]->info==legitimate)

{

linear\_probing(oldlist[i]->data,H);

}

}

free(oldlist);

return H;

}

void display(H\_table \*h)

{

for(int i=0;i<h->t\_size;i++)

{

if(h->list[i]->info == legitimate)

{

printf("index %d: %d\n",i,h->list[i]->data);

}

else

{

printf("index %d: NULL\n",i);

}

}

}

int main()

{

H\_table \*H;

int table\_size,n=0,count=0;

printf("Enter the table size:");

scanf("%d",&table\_size);

H = initialize\_Table(table\_size);

printf("You can perform\n1.Linear Probing\n2.Quadratic Probing\n3.Double Hashing\n4.Dispaly\n5.EXIT");

printf("\nNOTE:Rehashing will be performed if table is filled by more than half!!");

int opt,key;

do

{ printf("\nEnter your option:");

scanf("%d",&opt);

switch(opt)

{

case 1:

printf("Enter the no. of keys to insert:");

scanf("%d",&n);

count +=n;

while(n--)

{

printf("Enter the key:");

scanf("%d",&key);

linear\_probing(key,H);

}

if(count >(H->t\_size/2))

{ printf("Rehashing is performed!!\n");

H= rehashing(H);}

break;

case 2:

printf("Enter the no. of keys to insert:");

scanf("%d",&n);

count +=n;

while(n--)

{

printf("Enter the key:");

scanf("%d",&key);

quadratic\_probing(key,H);

}

if(count >(H->t\_size/2))

{ printf("Rehashing is performed!!\n");

H= rehashing(H);}

break;

case 3:

printf("Enter the no. of keys to insert:");

scanf("%d",&n);

count +=n;

while(n--)

{

printf("Enter the key:");

scanf("%d",&key);

double\_hashing(key,H);

}

if(count >(H->t\_size/2))

{ printf("Rehashing is performed!!\n");

rehashing(H);}

break;

case 4:

display(H);

break;

case 5:

printf("EXITING...");

break;

}

}while(opt!=5);

return 0;

}

OUTPUT:

You can perform

1.Linear Probing

2.Quadratic Probing

3.Double Hashing

4.Dispaly

5.EXIT

NOTE: Rehashing will be performed if table is filled by more than half!!

Enter your option:1

Enter the no. of keys to insert:6

Enter the key:34

Enter the key:78

Enter the key:34

Enter the key:24

Enter the key:23

Enter the key:71

Rehashing is performed!!

Enter your option:1

Enter the no. of keys to insert:4

Enter the key:89

Enter the key:54

Enter the key:83

Enter the key:02

Enter your option:4

index 0: 23

index 1: 24

index 2: 71

index 3: 2

index 4: NULL

index 5: NULL

index 6: NULL

index 7: NULL

index 8: 54

index 9: 78

index 10: NULL

index 11: 34

index 12: NULL

index 13: NULL

index 14: 83

index 15: NULL

index 16: NULL

index 17: NULL

index 18: NULL

index 19: NULL

index 20: 89

index 21: NULL

index 22: NULL

Enter your option:5

EXITING....

RESULT: Thus, the program was executed successfully.

//Separate chaining

AIM: To implement Hash data structure using Separate chaining as a collision resolution technique.

ALGORITHM:

STEP 1:- Declare an array of a linked list with the hash table size.

STEP 2:- Initialize an array of a linked list to NULL.

STEP 3:- Find hash key.

STEP 4:- If chain[key] == NULL Make chain[key] points to the key node.

STEP 5:- Otherwise(collision), Insert the key node at the end of the chain[key].

PROGRAM

#include<stdio.h>

#include<stdlib.h>

#define MINSIZE 7

struct node{

int data;

struct node \*next;

};

typedef struct node \*cell;

typedef struct node \*position;

typedef struct table

{

int t\_size;

cell \*list;

}H\_table;

int nextprime(int num)

{

int f;

if( num==1)

{

return 2;

}

while(num)

{

for(int i=2;i<num;i++)

{ f=1;

if(num%i==0)

{f=0;

break;}

}

if(f)

{

return num;

}

num ++;

}

}

H\_table\* initialize\_table(int size)

{

if(size < MINSIZE)

{

printf("Table size too small\n");

return NULL;

}

H\_table \*H;

int i;

H = (H\_table\*)malloc(sizeof(H\_table));

if(H==NULL)

{

printf("No memory allocated!!\n");

return NULL;

}

H->t\_size = nextprime(size);

H->list =(cell\*)malloc(H->t\_size\*sizeof(cell));

for(i=0;i<H->t\_size;i++)

{

H->list[i] = (struct node\*)malloc(sizeof(struct node));

if(H->list[i]==NULL)

{

printf("Error\n");

return NULL;

}

else{

H->list[i]->next=NULL;

}

}

return H;

}

int hash\_index(int key,int h\_size)

{

return key%h\_size;

}

position find(int key,H\_table \*H)

{

position p;

cell l;

l = H->list[hash\_index(key,H->t\_size)];

p = l->next;

while(p!=NULL && p->data!=key)

p= p->next;

return p;

}

void insert(int key,H\_table \*H)

{

cell pos,new,l;

pos = find(key,H);

if(pos==NULL)/\*key is not found\*/

{

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

if(new==NULL)

printf("Out of space!!\n");

else{

l= H->list[hash\_index(key,H->t\_size)];

new->data = key;

new->next = l->next;

l->next= new;

}

}

}

void display(H\_table \*H)

{

int i;

cell temp;

for (i=0;i<H->t\_size;i++)

{

temp = H->list[i]->next;

if(temp!=NULL)

printf("\nKeys in index %d:",i);

while(temp!=NULL)

{

printf("%d",temp->data);

if(temp->next!=NULL)

{

printf("->");

}

temp = temp->next;

}

}

}

int main()

{

H\_table \*H;

int table\_size,n,key;

printf("Enter the size of the table:");

scanf("%d",&table\_size);

H = initialize\_table(table\_size);

printf("Enter the No. of keys to insert:");

scanf("%d",&n);

while(n--){

printf("Enter the data:");

scanf("%d",&key);

insert(key,H);

}

printf("\nThe elements in the table are:");

display(H);

return 0;

}

OUTPUT:

Enter the size of the table:9

Enter the No. of keys to insert:7

Enter the data:2

Enter the data:56

Enter the data:78

Enter the data:22

Enter the data:69

Enter the data:82

Enter the data:61

The elements in the table are:

Keys in index 0:22

Keys in index 1:78->56

Keys in index 2:2

Keys in index 3:69

Keys in index 5:82

Keys in index 6:61

RESULT: Thus, the program was executed successfully.